Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Università degli Studi di Padova
Dipartimento di Ingegneria dell'Informazione
Laurea Triennale in Ingegneria Informatica

Animation of hand-drawn faces using machine learning

Candidate Student

Gaia Casotto

Student ID 2011761

Supervisor

Prof. Simone Milani

University of Padova

Co-supervisor

Dott. Elena Camuffo

University of Padova

19/07/2023

To my parents

Abstract

Today's research in artificial vision has brought us new and exciting possibilities for the production and analysis of multimedia content. Pose estimation is an artificial vision technology that detects and identifies a human body's position and orientation within a picture or video. It locates key points on the bodies, and uses them to create three-dimensional models. In digital animation, pose estimation has paved the way for new visual effects and 3D renderings. By detecting human movements, it is now possible to create fluid realistic animations from still images.

This bachelor thesis discusses the development of a pose estimation based program that is able to animate hand-drawn faces – in particular the caricatured faces in Papiri di Laurea – using machine learning and image manipulation. Working off of existing techniques for motion capture and 3D animation and making use of existing computer vision libraries like *OpenCV* or *dlib*, the project gave a satisfying result in the form of a short video of a hand-drawn caricatured figure that assumes the facial expressions fed to the program through an input video. The *First Order Motion Model* was used to create this facial animation. It is a model based on the idea of transferring the movement detected from a source video to an image.

Possible future developments could include the creation of a website: the user loads their drawing and a video of themselves to get a gif version of their papiro. This could make for a new feature to add to portraits and caricatures, and more specifically to this thesis, a new way to celebrate graduates in Padova.

Sommario

La ricerca sulla visione artificiale ci ha portato nuove ed entusiasmanti possibilità per quanto riguarda la produzione e l'analisi di contenuti multimediali. Pose estimation è una tecnologia di visione artificiale che rileva e riconosce la posizione e l'orientamento di un corpo umano all'interno di un'immagine o di un video. Trova dei punti chiave sui corpi e li utilizza per creare modelli tridimensionali. Per quanto rigurada l'animazione digitale, pose estimation ha aperto la strada a nuovi visual effects e tecniche di rendering 3D. Rilevando i movimenti umani, è ora possibile creare animazioni realistiche con le immagini.

Questa tesi discute lo sviluppo di un programma basato sul pose estimation in grado di animare volti disegnati a mano – in particolare i volti caricaturali dei Papiri di Laurea – utilizzando l'apprendimento automatico e la manipolazione delle immagini. Lavorando con tecniche esistenti di motion capture e facendo uso delle librerie di computer vision come *OpenCV* o *dlib*, il progetto ha dato un risultato soddisfacente nella forma di un breve video di una figura caricaturale disegnata a mano che assume le espressioni facciali fornite al programma attraverso un video di input. Il *First Order Motion Model* è stato usato per creare questa animazione facciale. È un modello che trasferisce il movimento rilevato da un video sorgente a un'immagine.

Possibili sviluppi futuri potrebbero includere la creazione di un sito web: l'utente carica il proprio disegno e un video di se stesso per ottenere un gif del proprio papiro. Questo potrebbe cambiare il modo di fare caricature, e in particolare, cambiare anche modo di celebrare i laureati a Padova.

Contents

List of Figures

1 Computer Graphics and Animation

In 1968, a group of Russian physicists and mathematicians with N. Konstantinov as its head created a mathematical model for the motion of a cat. On a BESM-4 computer they devised a program for solving the ordinary differential equations for this model. The Computer printed hundreds of frames on paper using alphabet symbols that were later filmed in sequence thus creating the first computer animation of a character, a walking cat.
Computer animation has come a long way since then.

1.1 CGI: Computer Generated Imagery

CGI refers to scenes, effects and images created with computer software [7].

In live action filming, it is often employed to insert animated elements into raw footage. A prime example of this would be Steven Spielberg's *Jurassic Park*, where the cast was made to look like they were walking next to huge CGI dinosaurs. More recently, you can consider all the big battle scenes in the hit HBO series *Game of Thrones*.

In animations, CGI gives the possibility to create entirely new realities, removing the need for complex, expensive and time-consuming manual animation. It allows artists far greater freedom to work outside the limits of 2D illustration. Prior to CGI in fact, computer animations heavily relied on traditional cel drawings, which are painstakingly slow processes that involve either hand-drawing each frame or using stop-motion with clay models. So while traditional animation involves manually creating a sequence of frames to create a fluid motion, CGI employs computer software to handle the transition between key moments. In comparison, traditional animations require more resources and budget: *The Lion King* used 800 animators and cost 45 million dollars to create, far more than its contemporary film *Toy Story*, which used 110 animators and cost 30 million dollars.

One of the earliest examples of computer animation using CGI was *A Computer Animated Hand*, a short film created by computer science students Fred Parke and Edwin Catmull. Created as a graduate project, the film features a 3D representation of Catmull's left hand opening and closing and pointing at the viewer. The creation process was very long and laborious: it included making a plaster model of Catmull's hand, and drawing onto it 350 interlocking polygons to create a wire-frame and map a set of coordinates. The data was then fed into a computer to build the realistic digitised moving hand. Catmull then broke more boundaries in CGI animation with another short film *Luxo Jr*, that depicted a sweet father-son interaction between two lamps. This film paved the way for the creation of *Toy Story*, the first ever full-length movie to be completely computer animated [7].

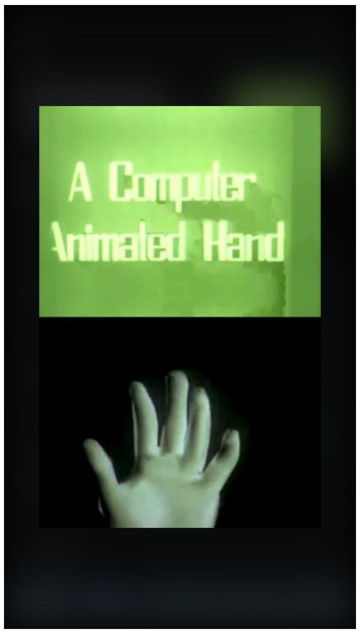

Figure 1.1: A computer animated hand

Catmull would then go on to co-found the animation studio Pixar, alongside John Lasseter, through which CGI became the new standard for animated filmmaking.

Of course, today CGI can be found in many diverse settings outside of the TV and film-making world, like medicine, science, architecture, art, advertising and engineering [7].

1.2 CGI PUPPETRY

Digital puppetry is the manipulation of digitally animated 2D or 3D models. Computers render these models inside virtual environments in real time [1]. This CGI technique is also known as rigging, or skeletal animation: it is closely associated with motion capture technologies and character animation. Models are represented in two ways: a *mesh* or *skin*, which is the representation of the surface of the character, and a skeleton or *rig*, which is a set of "bones" upon which the mesh is built. Each bone represents a set of vertices, and is associated with a portion of the model's mesh through a process called *skinning*. Portions of the character's skin can be associated with multiple bones [1].

After the rigging and skinning processes are complete, animation software can make the model move. A bone can be moved away from its default pose and orientation through three-dimensional transformations. The skeleton structure is often hierarchical: the reason behind this is intuitive when the rigged model is humanoid in shape – if a thigh-bone moves, the lower leg will follow. In this case the thigh-bone is the *parent bone*, and the lower leg bones are *child bones*. In better words, the full transform of a child node is the product of its parent transform and its own [1].

Specific constraints and limitations can be set for certain bones when moving them: to make movements look realistic, a rig can be generally moved by forward and inverse kinematics rules. If the rigged object is humanoid, then these parts would mainly be in the limbs to make sure the model is not moved into any unnatural positions. The character's skin will move according to how the bones associated with it are manipulated.

One of the strengths of rigging in animation is that animators can define the movements of the characters by simply moving the keypoints in the skeletal structure. If the characters where only defined by polygons, then a movement would be the result of manually moving one vertex at a time to the desired position. CGI puppetry can be enhanced with virtual anatomy properties, such as bone strength, muscle reaction and weight of the limbs, which can give more realistic results when the model performs virtual stunts. Virtual anatomy and rigging are often combined with artificial intelligence for further enhancement of animation and simulation technology [1].

1.3 Pose Estimation and DeepFakes

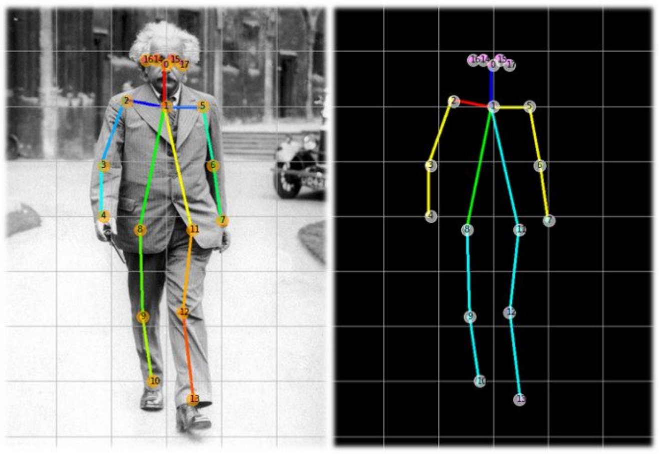

Figure 1.2: 2D pose estimation

Pose estimation is a computer vision task used to detect and track the position and orientation of a person or object in a given input – be it image, video or live feed – relative to the position of the camera [8]. This is usually done by predicting the location of specific landmarks, representative of significant features in the person or object, i.e. separating body parts or limbs that can be considered rigid during motion. So in the case of human pose estimation, such keypoints would be hands, head, feet and so on. The position of the keypoints is then predicted with machine learning models, after a training procedure operating on manually annotated images.. There is a distinction to be made between 2D and 3D pose estimation. The first simply estimates the location of keypoints in 2D pixel coordinates relative to the image or video frame. The second works to transform an object in a 2D image into a 3D object by adding a z-dimension to the prediction. Naturally, 3D pose estimation makes for a more challenging problem for machine learners: detecting the actual spatial positioning of an object or person inside a scene requires larger datasets and algorithms that take into account background, lighting and many more factors. With this technology, we are able to track a person or object in three dimensional space with incredible accuracy. This has a wide range of possible applications, and researchers are coming up with many new tools designed to track human movement and activity, like AI-powered personal trainers or movement trackers on factory floors to ensure worker safety. In addition to these functionalities, pose estimation is shaping the future of augmented reality technologies, as well as gaming, robotics and of course animation [3] [12].

There are several techniques for pose estimation. In general, CNNs (convolutional neural networks) are the most suited deep learning models for pose estimation, and although there are too many specific neural network architectures to discuss them in this paper, some of the most robust architectures are Mask-RCNN, Stacked-Hourglass networks and other encoder-decoder models, like PersonLab and OpenPose [3] [12] [8]. One of the most obvious areas where pose estimation is applied is in human movement tracking and measurement, and this is the most important aspect to consider for the purposes of this thesis. Whereas most inanimate objects are rigid, humans are flexible, so keypoints are located on the major joints. Since we are able to locate and track a person's motions in real-world space, we can easily overlay a digital model onto the real person that is being tracked. This is game-changing when it comes to 3D and 2D animation; with the advent of deep learning approached to pose estimation, the time consuming processes involved in animating characters are becoming automated. Character animation is relying less and less on specialized suits or markers, and is instead combining pose estimation techniques with real-time motion capture. Advances in gesture recognition have contributed as well in creating hyper-realistic characters, often crossing the line into the so-called *Uncanny Valley*. These techniques are becoming more and more common in gaming graphics, thanks to low cost 3D body tracking device like Microsoft's Kinect depth camera [3].

Pose estimation is also used for specific tasks in image morphing and manipulation: figures in images can be animated by transferring onto them the movement detected in a video or live-feed. This has lead to the creation of deep fakes, i.e., computer generated videos realistic enough to be mistaken as real footage. This technology has great potential. Current research suggests it will likely be applied to movies to create realistic dubbing in foreign languages, or even for educational purposes, like the reanimation of historical figures, or even simply in editing and retouching editorial content. This technology is widley available and easy to be learned and used. So, while there are many positive aspects to having non-technologically savvy individuals have access to these innovative tools, there are also many downsides and potential dangers. DeepFakes are in fact expected to advance current levels of misinformation and disinformation sources to the next level. There have already been cases of synthetic pornographic videos being used to damage a person's reputation or to blackmail them, or the creation of fake speeches to generate political or religious unrest, or combinations with synthetic voice recordings to be used for identity theft. Despite noticeable progress in deepFake detection software, it still poses a great threat to the effectiveness of face recognition systems and the integrity of online information [8] [3] [12].

Regardless of the dangers of deepFakes, this thesis aims to explore a less malicious use of pose estimation and image morphing algorithms, by applying it to hand-drawn portraits and caricatures.

2

Facial Morphing and Animation Tools

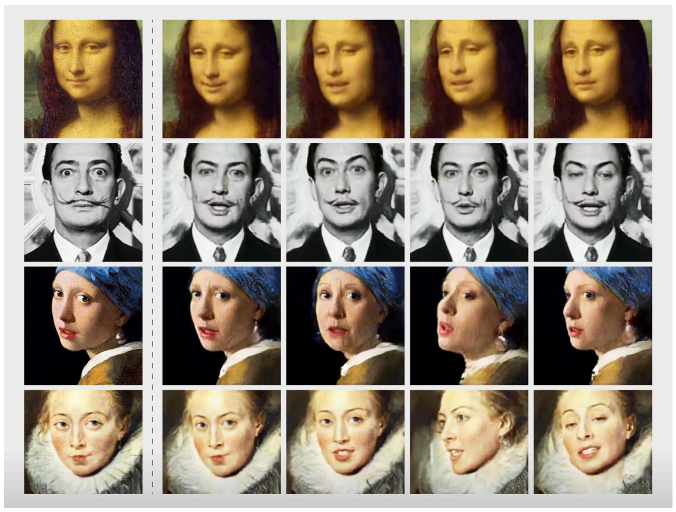
Figure 2.1: DeepFakes generated by a single image

2.1 DeepFake Models: Autoencoders and GANs

A mathematical approach to morphing a face in a picture is called morphing of triangulations: triangular areas are mapped onto the face, and the face is manipulated by applying transformations to the vertices of these triangles.

The creation of the triangular scheme is not random: a number of critical points in the face are selected – the so-called *facial landmarks*, like the tip of the nose, or the corners of the mouth. These keypoints become vertices in the triangulation scheme. This approach is one of the easiest and most intuitive

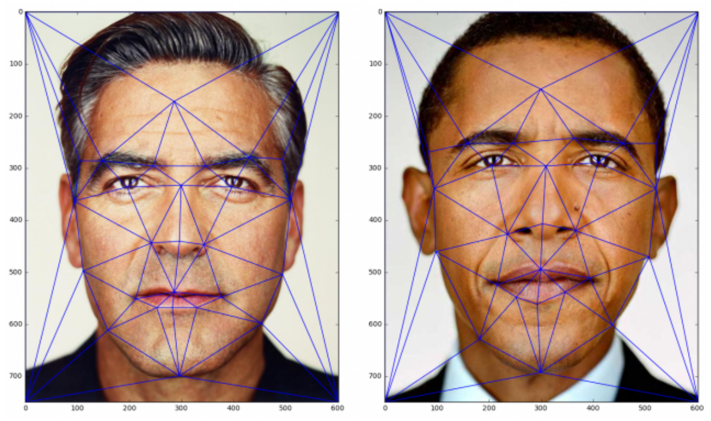
Figure 2.2: Triangulation meshes on faces

to understand, and it is similar in a way to the rigging and skinning process described before. Mapping the keypoints in the triangulation scheme acts as a skeleton structure for the face. By applying linear transforms to the mesh, it is possible to morph images – giving a person different expressions or features, or morphing from one face to an entirely different one smoothly.
This process is often referred to as puppeteering. A first approach to obtaining this sort of result is the traditional use of visual effects and computer-graphics, but most of the technology behind face morphing is based on machine learning algorithms (see pose estimation as an example). DeepFake models fall into two categories; *lip-sync* and *puppet-master*. The first category refers to videos synthetically made from a source audio: the frames are modified to make the mouth movements consistent with the sounds in the recording. The second category instead includes videos made from images of the target (puppet), modified to follow the eye, mouth and head movements of another person (master) [11] [17] [2] [13].

Autoencoders

Considering deep learning approaches, the most widely-used architecture is the autoencoder. An autoencoder is a special type of neural network whose objective is to match the input it was provided with. This is done through the use of an encoder and a decoder. In the diagram above, we see a face being fed into the encoder. The result is a latent face or base vector, which is then passed to the decoder, that reconstructs the face.

It is important to note that two separately trained autoencoders are incompatible with each other; each network will deem a certain set of features as *meaningful*. So what needed to be done to make deepFake technology possible was finding a way for both latent faces to be encoded the same way: both networks share the same encoder, but have two different decoders. In training, network A and network B are two separate entities, one is only trained with faces of A, the other with faces of B. But all latent faces are produced by the same encoder, whose job is to identify common features in both faces. This way, the encoder learns the concept of "face" itself. Once training is complete, latent face A is passed to decoder B. the decoder will try to reconstruct subject B from the information gathered on subject A's features. If the network has learned what makes a face, then the latent face will be a representation of facial expressions. So the decoder will generate subject B's face with the same expression and orientation of subject A.

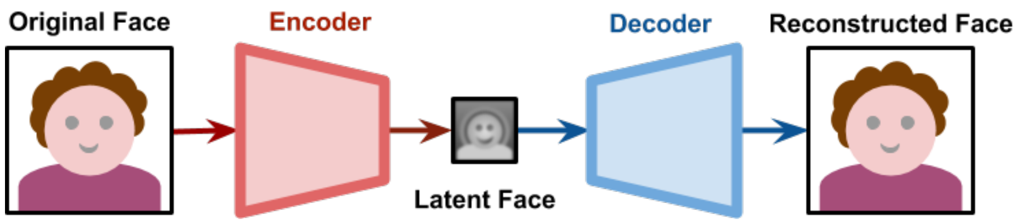

Figure 2.3: Encoder-decoder scheme

So, what is important in the training data is to make sure that subjects A and B have similar features so that the network can learn to generalize well enough. This means that the use of autoencoders for deepFakes is not limited to faces. For instance, it can be used on bodies, or animals, or fruits. Some research on autoencoders pushes the boundaries on what an encoder might deem to be similar features; while converting human faces into animals and vice versa has yielded some interesting results, the same cannot be said for converting faces into fruit [19].

GANs

GANs (Generative Adversarial Networks) are another deep learning architecture. The objective is to create something new from the information collected

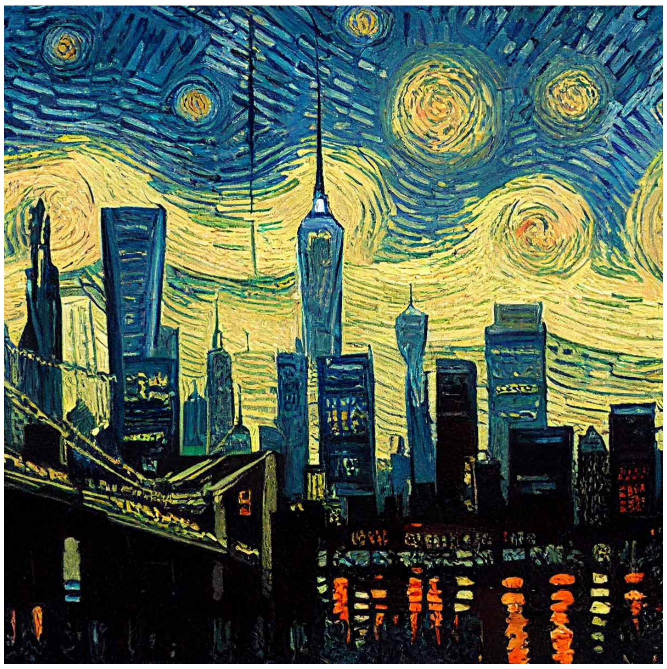
Figure 2.4: New York painted by Van Gogh, according to AI

from the training data. So for example, given all of Van Gogh's art works, a GAN can create a new painting in Van Gogh's style [5].

GANs are the result of two neural networks that are in constant communication with each other but that play adversarial roles. The first is the Generator; its job is to produce a new image based on the learned knowledge. The other neural network is the Discriminator, that needs to determine whether the produced image is authentic or produced by the Generator. So the discriminator must learn how not to be deceived, and the generator learns how to create images that will make the discriminator deem them as real images. The better the discriminator is, the harder the generator will have to work to deceive it, the more realistic the images will be.

DeepFakes are making more and more use of the recent advancements of powerful GAN models. GAN models aiming at facial manipulation in images can be categorized according to their objectives; which are mainly face synthesis, face swap detection and facial attributes and expressions [14].

Face synthesis aims to create non-existent realistic faces. The most popular approach for this is StyleGAN, which uses CNNs to map the input images through several fully connected layers. Gaussian noise is added to the maps after each convolution. This architecture makes it possible to control the image synthesis via scale-specific modifications to the styles.

Face swap detection is continuously evolving since there are many concerns regarding safeguarding human rights from the new threats this technology poses. Most face swap detection systems use CNNs trying to find the "fingerprints" left onto synthesized images from GANs.

When modifying facial attributes and expressions in images, StarGAN has proved to be one of the most effective methods. It uses a single model trained across multiple attributes' domains instead of training multiple generators for every domain. The generator takes as input both an image and a target domain label (for example, the label "happy" or "sad"), and generates a fake image. It tries to generate images indistinguishable from real ones, such that the discriminator will classify them as belonging to the target domain [14].

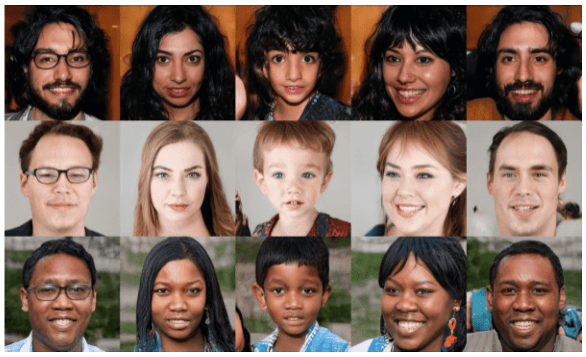

Figure 2.5: These people are not real – they were produced by StyleGAN's generator

2.2 The dlib and OpenCV Libraries

Dlib is a C++ library that provides efficient implementations of a variety of machine learning algorithms, including object detection and facial landmark detection. Dlib includes a variety of machine learning algorithms, such as support vector machines, decision trees, deep learning neural networks, and linear regression. Additionally, Dlib provides a range of other useful functions, such as image manipulation, 3D rendering, and matrix computations.

One of the significant features of Dlib is its fast and efficient implementation of facial landmark detection, object detection, and face recognition.

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It provides a wide range of functions and tools for image and video processing such as image filtering, edge detection and image transformation. Additionally, OpenCV includes several machine learning algorithms such as classification, clustering, and object detection.

With their extensive functionalities and wide range of applications, openCV and dlib have become popular libraries in the fields of computer vision, robotics, and machine learning. They're used worldwide to develop computer vision-based applications and products.

Face Detection:

Face detection can be regarded as a specific case of object-class detection: the task is to locate all objects in an image that belong to a same class. Face detection algorithms not only identify human faces, but also return the locations of all faces within the image. Dlib has two face detection algorithms built in:

- A HOG + Linear SVM face detector that is accurate and computationally efficient. It is fast and efficient, but due to how HOG (Histogram of Oriented Gradients) works, it is not invariant to changes in rotation and viewing angles.
- A Max-Margin (MMOD) CNN face detector that is both highly accurate and very robust, capable of detecting faces from varying viewing angles, lighting conditions, and occlusions.

Python's *face_recognition* package wraps dLib's face recognition functions into a simple, easy to use API. The package uses HOG feature descriptors and Support Vector Machines (SVM) to detect faces in images. This process involves scanning the image with a sliding window and applying the HOG feature descriptor at each location. The SVM classifier then determines whether or not a face is present at each location. The detected face locations are stored in the *face_locations* variable as a list of tuples, where each tuple represents the coordinates of a face's topleft and bottomright corners in the image [4] [9].

Facial Landmarks:

Dlib has a pre-trained facial landmark detector that estimates the location of 68 (x,y)-coordinates that map salient regions of a face, such as eyes, eyebrows, nose and mouth. Detecting facial landmarks is a specific case of the shape prediction problem; given an input, a shape predictor localizes points of interest

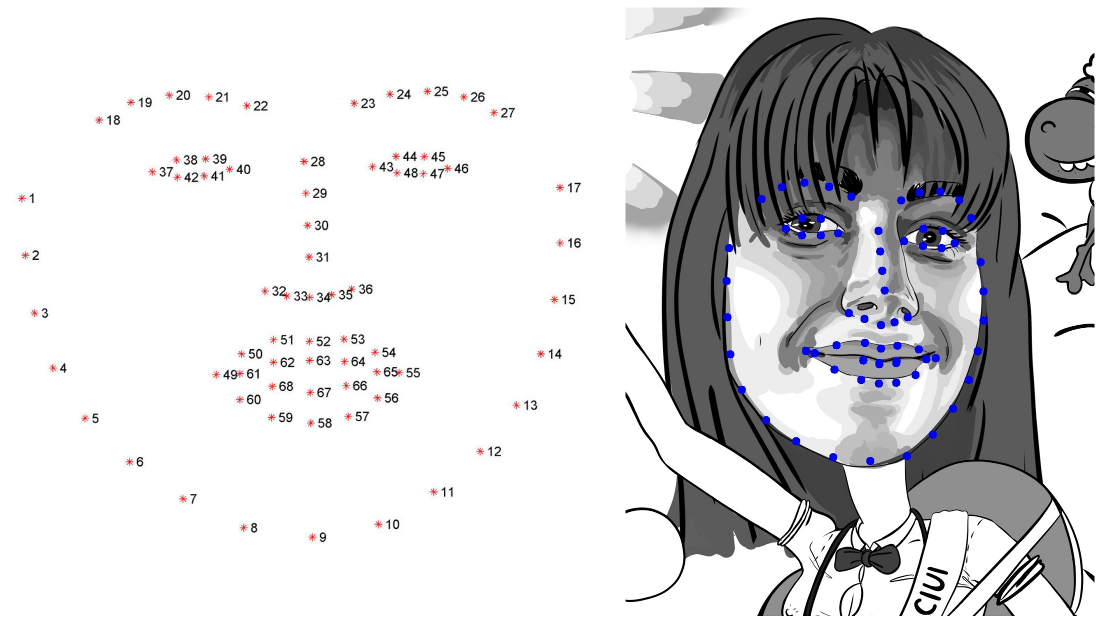

Figure 2.6: Facial landmark detection

along the object of interest. So detecting facial landmarks is a two step process: first the location of the face is found in the image, then the key facial structures are estimated. Once the face region is found, the facial landmarks are found and labeled by the shape predictor. The facial landmark detector in dlib is an ensemble of regression trees trained on the 68 point iBUG 300-W dataset. It is able to make high quality predictions in real-time. The 68 coordinates are stored in a *shape* object, and can be visualized as shown in *Figure 2.6* [15] [6] [9].

3

First Order Motion Model

Deep generative models like GANs and Auto-Encoders have proven to be very effective for image animation and video re-targeting. They have been used to transfer facial expressions or movement patterns between human subjects in photos and videos. These approaches are very costly, and are often referred to as *object-specific* since they usually rely on pre-trained models that extract motion based on sets of specified landmarks or keypoints. As a matter of fact, such methods are either too expensive data-wise or not available for arbitrary object categories. The first order motion model for image animation is an open source library that allows the user to create videos using still images and facial capture. It is able to create an animation of the source image by extracting and retargeting the movement from the driving video. Once trained on a particular kind of object, it applies to all objects of the same category. This sets it apart from similar prior solutions for movement extraction, as it does not need huge amounts of data annotations.
The first order motion model performs very well on faces: it successfully extracts facial expressions, head poses and eye movements. In order to create the animation, the necessary inputs are a source image intended for the animation, and a driving video, that contains the desired motion. The process is divided into motion extraction and generation [10] [16].

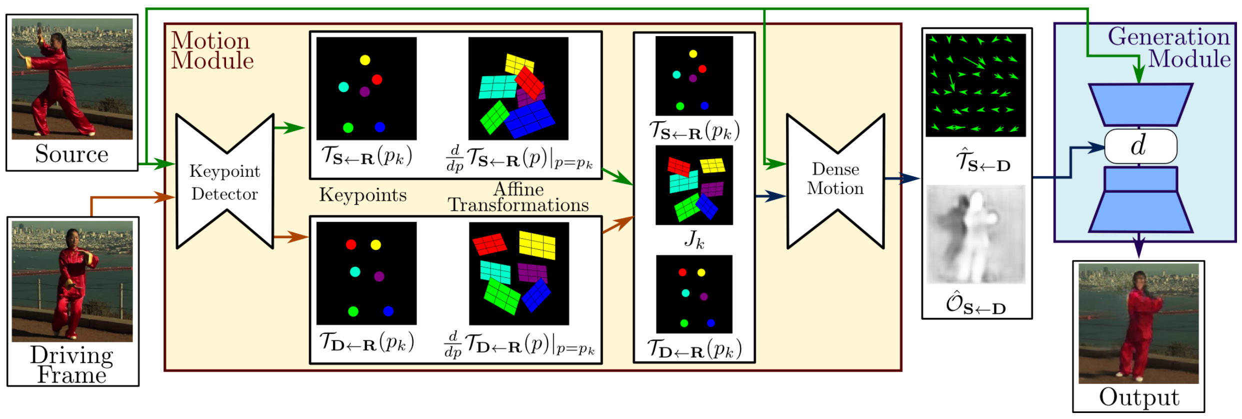
Source
Driving Frame
Motion Module
Keypoint Detector
$\mathcal{T}_{\mathbf{S}\leftarrow\mathbf{R}}(p_k)$
$\frac{d}{dp}\mathcal{T}_{\mathbf{S}\leftarrow\mathbf{R}}(p)|_{p=p_k}$
Keypoints
Affine Transformations
$\mathcal{T}_{\mathbf{D}\leftarrow\mathbf{R}}(p_k)$
$\frac{d}{dp}\mathcal{T}_{\mathbf{D}\leftarrow\mathbf{R}}(p)|_{p=p_k}$
J_k
Dense Motion
$\hat{\mathcal{T}}_{\mathbf{S}\leftarrow\mathbf{D}}$
$\hat{\mathcal{O}}_{\mathbf{S}\leftarrow\mathbf{D}}$
Generation Module
d
Output

Figure 3.1: First order motion model scheme

Disentangling Appearance and Motion

To extract motion, the driving video is passed as a series of consecutive frames to an unsupervised keypoint detector. For each frame, the model returns a set of sparse keypoints, and for each of these, it finds local affine transformations in the neighbourhood of the keypoint. To get motion, the outputs are subtracted from one another, and obtain what is known as *sparse motion* since it is calculated from a sparse set of coordinates within the frame. A second network is used to create *dense motion*; this network additionally provides an occlusion map – so a map of everything that should not be affected by the movement. The occlusion map is especially useful when the driving video contains large motion patterns.

The appearance of the source image is extracted through an encoder-decoder model.

Now the process of generation can begin: after the source image is passed through the encoder, the features are warped using the dense motion extracted from the video, and this is then multiplied by the occlusion map. This way the decoder knows what areas of the image require modification and which do not. The decoder can now generate a new image, giving a new position or expression to the subject in the source image [16].

Training and testing

In training, both the source and the target are frames of the driving video. So the generated video is compared to the driving video, and the model teaches itself to be more accurate. This method is known as *self-supervised training*, and it is this method that makes the model able to work freely from labels, and perform well on varying categories of objects, such as human and animal bodies, faces and also robotic arm movements. During training, the model learns a distinct set of keypoints for each object, sufficient to define complex motion for image animation [16].

The model successfully retargets motion and animates static images.

4 Animating Papiri di Laurea

The research question for this thesis is the creation of a program that animates hand-drawn caricatured faces. Specifically, this program is intended for the animation of Padova's traditional Papiri di Laurea.

The process behind the creation of this project can be subdivided into these steps:

1. face isolation
2. creating the animation: face animation using First Order Motion Model
3. stitching video-frames to create the final animation

Before the technical analysis of these steps, a brief look into the history and tradition of the Papiro di laurea in Padova.

4.1 What is a Papiro di Laurea?

Beloved by graduates and hated by public administrations, Papiri di Laurea are a tradition in Padova that sets it apart from the rest of the university cities in Italy.
The origins of this Paduan tradition plant their roots in the XVI century. They were short manifests born as a way to celebrate the graduate's accomplishment. At first they were only written, then small drawings of the student started taking up more and more space in the papiro, until they became the main protagonist and an artistic statement.

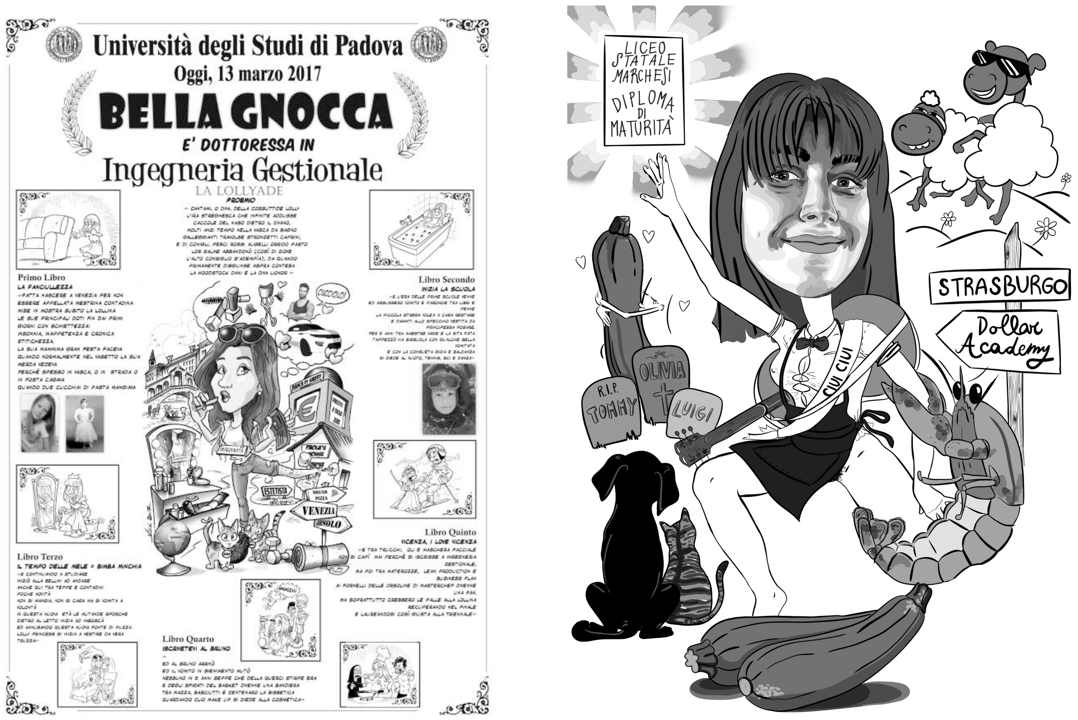

Figure 4.1: Examples of modern papiri di laurea

In the 1950s, papiri started embodying Italian goliardic style; the portraits became caricatures and the explicit references to sex taking center stage. This is when the papiro becomes a tool to ridicule the graduate.
Today, Papiri are usually formatted as follows: the caricature is in the center, often explicit and vulgar. Around it is a collection of anecdotes of the graduate's student life, written in rhyme by the student's friends. It is tradition to dress the graduate up in an embarrassing costume, get them drunk, and make them publicly read the Papiro while flour, eggs, tomato sauce and more are thrown on them.

4.2 Face Isolation

For a satisfactory result, the source image used to generate the animation must be a cropped version of the entire papiro – a close up on the face.
This can be done manually by cropping the caricature with a photo editor, but in sight of possible future applications of this project, the process of cropping needs to be handled by the program itself.

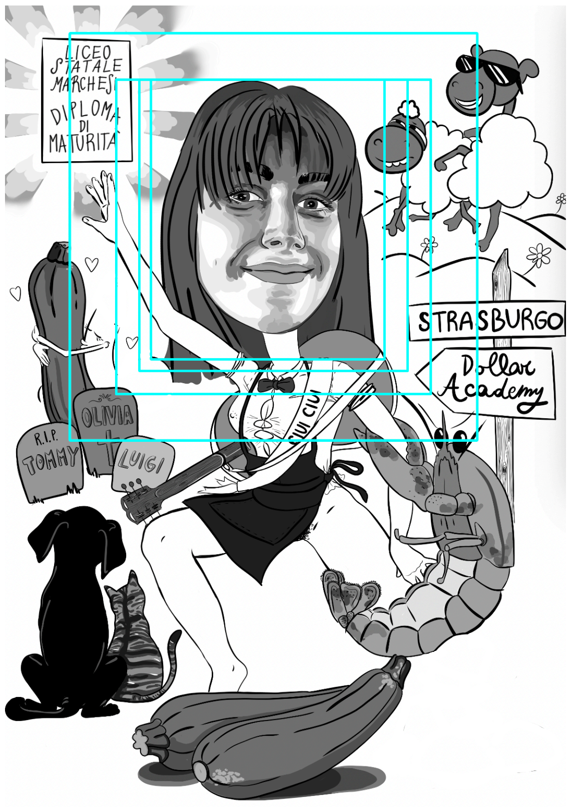

Figure 4.2: Example of result obtained running extract_head

EXTRACT_HEAD.PY

The code for this works as follows:

- the **face_recognition** package finds the face in the image. It returns four values *(x,y,w,h)* that represent the tightest rectangle enclosing the entire face: x and y are the coordinates of the top left vertex of this rectangle, w is the width of the rectangle and h is the height.
  These values are scaled by a default value (10%) and the resulting enlarged rectangle is drawn on the image displayed on screen.

```
        #find face in image
        faces = face_recognition.face_locations(image)

        #10% scale for face location coordinates
        scaleFactor = 1.1

        ....

        #get the resized rectangle points
        newLeft   = max(0, int(cX - scaleFactor * M))
        newTop    = max(0, int(cY - scaleFactor * M))
        newRight  = min(image.shape[1], int(cX + scaleFactor * M
))
```

```
        newBottom = min(image.shape[0], int(cY + scaleFactor * M
    ))

        ....

        #draw scaled rectangle
        cv2.rectangle(image,
                      (newLeft, newTop), (newRight, newBottom),
                      (255, 255, 0), 10
                     )


```

where *cX* and *cY* are the coordinates for the center of the face, and
M = (abs(width) + abs(height)) / 2.

- the user can now adjust the dimensions of the rectangle through the command line until satisfied. The program then saves the image, cropped to the desired dimensions, in the correct directory. The new *(x,y,w,h)* coordinates are saved on a file which will be useful for the process of correctly re-positioning the face within the papiro. The name of the file is decided by the user, and is taken as input. But for the purposes of this thesis, it will be referenced to as *coordinates.txt*.

The obtained image is a close up of the face, which will be referenced to as *face_image* in the following sections. An example of *extract_head* results is reported in *Figure 4.2*.

4.3 Creating the animation

Creating an animation from an image requires a driving video as well as the image that needs to be animated. From the driving video, motion is extracted and then the frames of the animation are generated. This part of the process relies heavily on GPU calculations and on CUDA. Because of this, seeing as the available devices could not support CUDA, this part of the project was executed on Google Colab, a cloud-based Jupyter notebook environment that runs in any web browser. Credit for the code in the google colab page goes to Hui Zhang and Jian Zhao [18].

Python packages

The imported python packages can be seen in the following code.

```
import torch  torch.nn, torch.nn.functional,
...
try:
import imageio
import imageio_ffmpeg
except:
    !pip install imageio_ffmpeg
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from skimage.transform import resize
from IPython.display import HTML
import warnings
import os
...
from skimage import img_as_ubyte
```

There are also several custom defined networks in use: the inpainting network, keypoint detector and dense motion network, which are configured in the *load_checkpoints* function in the *demo.py* file.

```
from demo import load_checkpoints
inpainting, kp_detector, dense_motion_network, avd_network =
load_checkpoints(config_path = config_path, checkpoint_path =
checkpoint_path, device = device)
...
from demo import make_animation
```

DEMO.PY

In this file, functions for loading checkpoints, relative keypoint calculations and making animations are defined. The recommended python version is 3.9, but the minimum requirement is Python 3. An exception is raised if this condition is not met.

Demo.py:

- To calculate relative keypoints between the source and the driving video frames, the Convex Hull algorithm is imported from *scipy.spatial*. This algorithm computes the area of the keypoints and determines a scale factor to adjust the movement between keypoints.
- To load the checkpoints for the various networks used in the animation, a function creates instances of the networks and loads the pre-trained

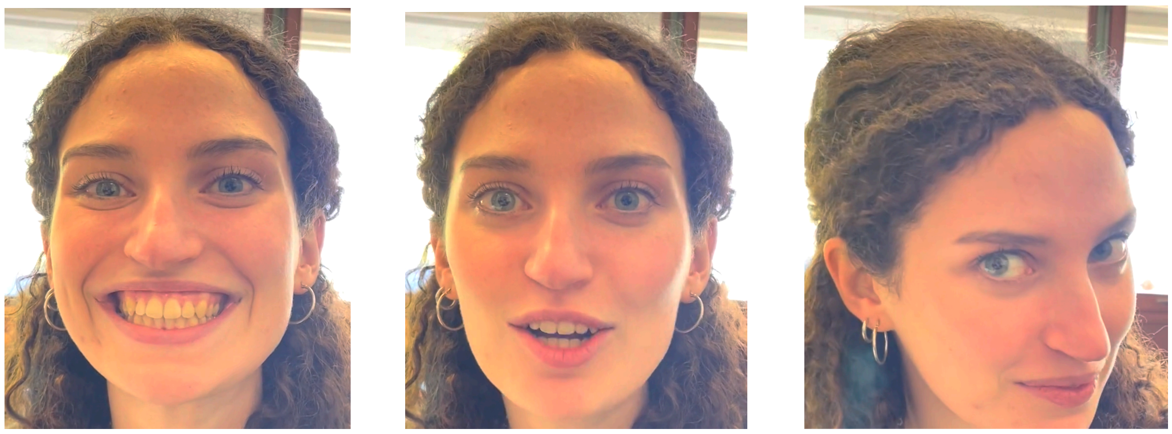
Figure 4.3: A few frames from the driving videos (i.e. the video that is supposed to control Papiro animation)

weights from the checkpoints. The networks are then set to evaluation mode, and then returns them.

- To generate the animation frames, the function *make_animation(...)* is defined. The required parameters are: face_image, the driving video frames, the networks and the device (device = torch.device('cuda:0')). Iterating over the frames in the driving video, the function performs the following steps:
  - converts face_image and driving frame to tensors and moves them to the specified device
  - calculates the keypoints of face_image and initial driving frame using the keypoint detector, and then calculates the normalized keypoints for the current driving frame
  - computes the dense motion between face_image and the normalized keypoints
  - generates the inpainted frame using the inpainting network and the computed dense motion
  - appends the resulting frame to a predicition list

  The function returns the list of generated frames. An example of the results of this procedure is reported in *Figures 4.3* and *4.4*. *Fig. 4.3* reports the control frames while *Fig. 4.4* reports the animated Papiro face.

MAIN.PY

Face_image is opened using the *imread(image_path)* method from *imageio*. For the video, the method *get_reader(video_path)* is used. The video frame rate

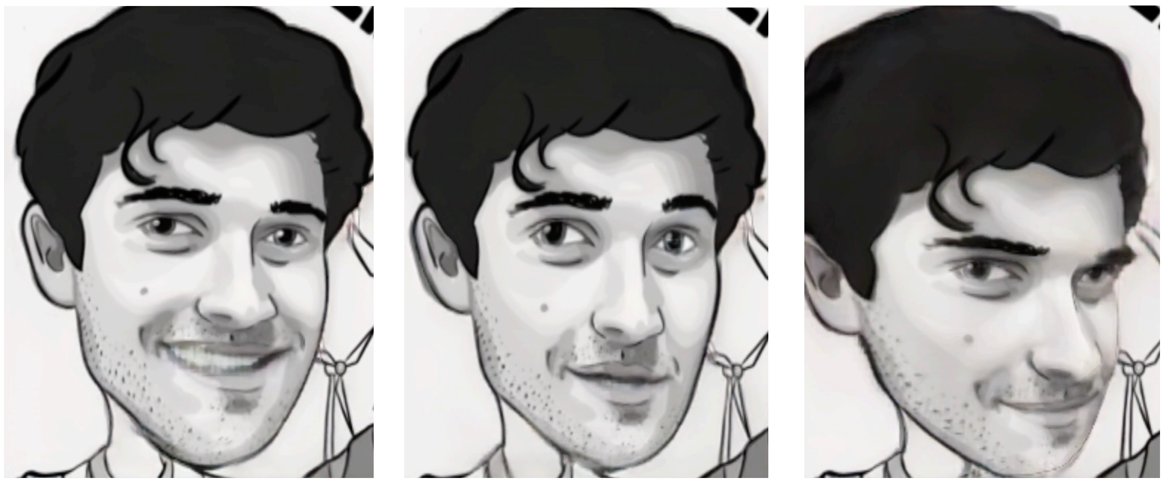

Figure 4.4: Generated video

is saved, as well as the initial height and width of face_image. Both face_image and video are resized to fit into squares of the same dimensions. A predictions vector is created:

```
   predictions = make_animation(
       source_image, driving_video, inpainting, kp_detector,
       dense_motion_network, avd_network, device = device,
       mode = predict_mode
       )
```

The resulting video is saved using *imageio.mimsave* function. Every frame in predictions is resized again, to take on the original proportions of face_image.

4.4 Stitching Video Frames

All that is left to do is to restitch the generated video frames onto the papiro to create the final video.

CREATE_VIDEO.PY

This file reads the four coordinates from the *coordinates.txt* file, and then loads the generated face animation. For each frame in the generated video, the function *replace_face* is called. This function requires the following parameters: the full papiro, a frame of the generated face animation, and an array of four coordinates. From the coordinates, width and height are calculated and the frame is resized to fit back into the original size of the face_image, and is positioned back into it's original place within the papiro

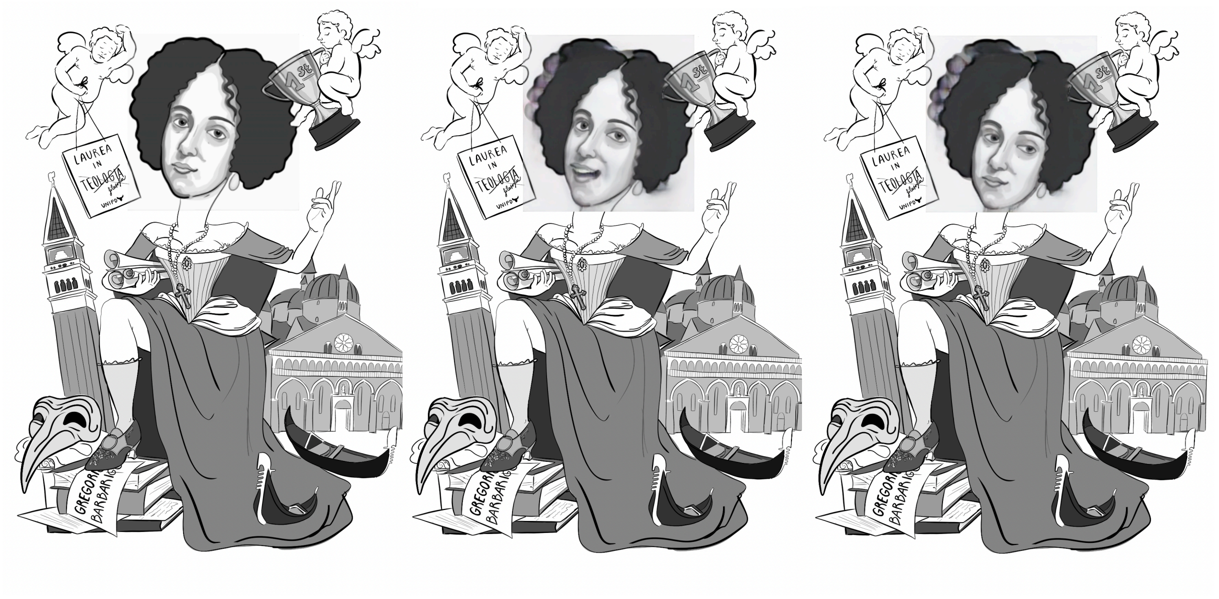

Figure 4.5: An example of stitching face images into the final Papiro

```
    new_im = np.copy(papiro)

    w    = dr - x #calculates width
    h    = db - y #calculates height
    face = cv2.resize(frame, (w,h)) #resizes to appropriate scale

    new_im[y:y+h, x:x+w] = face #replaces face
```

where *(x,y,dr,db)* are the coordinates saved in *coordinates.txt*.

The final video is created with the same frame rate as the generated face animation video.

4.5 Results

Figures 4.5 and 4.6 are frames taken from the resulting videos.

The first image in each sequence is the first frame of the video, in which the papiro is still unmodified. The other images in the sequences showcase the face morphing into different expressions.

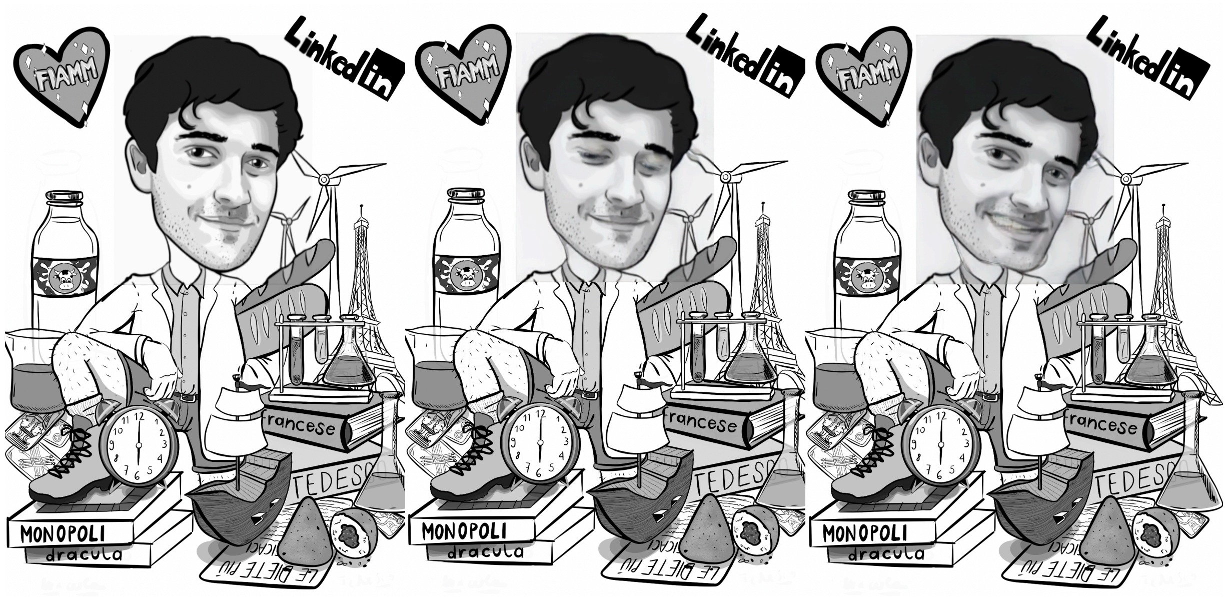

Figure 4.6: Example of final results on the animated Papiro frames

5 Conclusions

This project has successfully reached its initial goal; the animation of a hand-drawn face on a Papiro di Laurea.
First order motion model produces a smooth animation, and the driving video's facial expressions are recognizable and realistic. Thanks to the way the model is trained, missing facial elements are predicted well enough if needed. This can be seen in smiles or blinks or slight twists of the face, where teeth and eyelids and jawlines are realistically predicted and generated.

Taking for example a drawing of a tight-lipped smile, the model is able to generate a wider smile, with teeth. But if the driving video has an expression where the tongue is sticking out, or even if the mouth is open wide, here the generation model struggles to follow and recreate the facial expression. Other than these performance inefficiencies, which require intense training and great generative computations, other limitations to this project are examined below.

5.1 Project Limitations

Heavily caricatured faces

While the face recognition models available in the python library are very efficient tools when working on photos, or realistic drawings, it proved more of a challenge for these face detectors to recognized caricatured hand-drawn faces. The face in *Figure 5.1* is not detected when working with CascadeClassifier.
This means that the program could not work with papiri that are heavily cari-

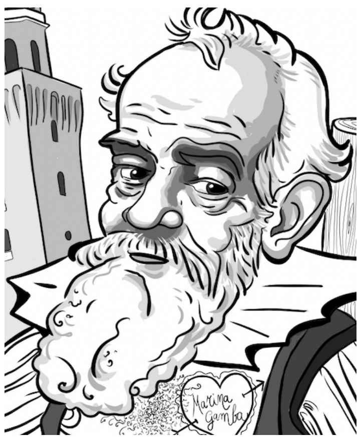

Figure 5.1: Example of heavily caricatured face

catured.

Cropping tightly around the face

The first order motion model does not have any facial recognition itself. It is designed to work on objects of the same type, extracting the movement from the driving video and animating the source video. This characteristic of the first order motion model is what makes it so elastic, able to animate animals, faces, human bodies and more. Driving video and source image just need to be of the same category.
This is why the source image (face_image) needs to be a tight rectangle around the head, and the driving video frames as well need to be adequately tight around the face. The less background noise the motion model has to work around, the cleaner the animation.
Because of this, the source image cannot be the entirety of the papiro, but it has to be limited to the face so that only the face is animated. As a matter of fact, stitching the video on top of the Papiro is a necessary step.
A possible solution to this would be to use a driving video that has been shot in a very similar angle to how the Papiro has been drawn, but this is a high ask for a possible future user. Another solution would be to not use the First Order Motion Model, but rather a specific deep learning architecture.

Different color backgrounds

In the final result, the difference in color between the original background of the image and the rectangle where the generated frame is stitched generates

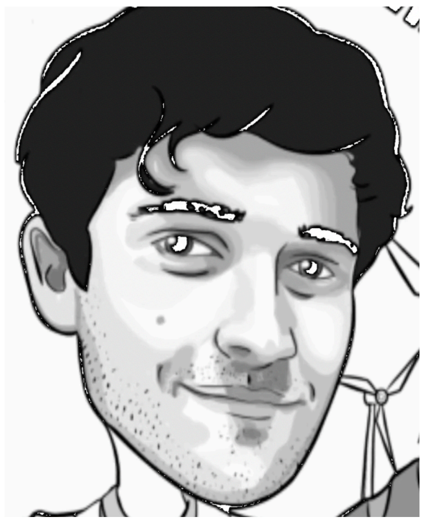
Figure 5.2: Attempting to set the background to white

some visual disturbance to the viewer: the generated frame is a few shades darker. This is due to the approximations and interpolations made by the first order motion model during the prediction of each frame.
This gives the final result a less seamless effect, and overall gives a product that is lower in quality than originally intended.
Processing each frame to bring the background back to its original color permits mitigating this problem, although the solutions attempted so far have not yielded good enough results.

5.2 Future Developments

This project creates a new format for graduation celebrations in Padova; animating a Papiro gives its creators more space to ridicule their graduating friend, as well as the possibility to capture the graduate more in depth than with just a simple drawing.
However, the audience for this project is more global, and does not to be specifically utilized for Papiri in Padova. A future development of a website would make this accessible to anyone. The user would be able to:

- upload a drawing of a human face or of an entire human figure;
- upload a video of themselves;
- download and save the animated version of their uploaded image.

Further work on this project would also focus on fixing the limitations listed above.

References

[1] Adobe. *Rigging e skeletal animation: cosè e come funziona*. last accessed 30 June 2023. 2023. URL: https://www.adobe.com/it/creativecloud/animation/discover/rigging.html.

[2] Zahid Akhtar. *Deepfakes Generation and Detection: A Short Survey*. last accessed 9 May 2023. 2023. URL: https://www.mdpi.com/2313-433X/9/1/18.

[3] Yu Cheng et al. *Occlusion-Aware Networks for 3D Human Pose Estimation in Video*. Last accessed 14 June 2023. 2019. URL: https://openaccess.thecvf.com/content_ICCV_2019/papers/Cheng_Occlusion-Aware_Networks_for_3D_Human_Pose_Estimation_in_Video_ICCV_2019_paper.pdf.

[4] Matt Clarke. *How to perform facial recognition in Python*. last accessed 10 May 2023. 2021. URL: https://practicaldatascience.co.uk/machine-learning/how-to-perform-facial-recognition-in-python.

[5] CVisionLab. *Deepfake (Generative adversarial network)*. last accessed 4 July 2023. 2023. URL: https://www.cvisionlab.com/cases/deepfake-gan/.

[6] Victor Dey. *Facial Motion Capture for Animation Using First Order Motion Model*. last accessed 27 May 2023. 2021. URL: https://analyticsindiamag.com/facial-motion-capture-for-animation-using-first-order-motion-model/.

[7] Adobe Inc. *CGI animation explained: definition, history and examples*. last accessed 18 April 2023. URL: https://www.adobe.com/uk/creativecloud/animation/discover/cgi-animation.html#:~:text=CGI%20(computer%20generated%20imagery)%20is,either%20subtle%20or%20obvious%20ways..

[8] Fritz Labs Inc. *Pose Estimation Guide: almost everything you need to know about how pose estimation works*. last accessed 14 June 2023. 2021. URL: https://www.fritz.ai/pose-estimation/#:~:text=Pose%20estimation%20is%20a%20computer,of%20a%20given%20person%2Fobject..

[9] Italo Josè. *Facial mapping (landmarks) with Dlib + python*. last accessed 15 May 2023. 2018. URL: https://towardsdatascience.com/facial-mapping-landmarks-with-dlib-python-160abcf7d672.

[10] Michal Kostewicz. *How to bring image to life using machine learing*. last accessed 30 June 2023. 2020. URL: http://code-addict.pl/real-time-image-animation/.

[11] Yisroel Mirsky and Wenke Lee. *The Creation and Detection of Deepfakes: A Survey*. last accessed 8 May 2023. 2021. URL: https://dl.acm.org/doi/10.1145/3425780.

[12] Hung-Cuong Nguyen et al. *Unified End-to-End YOLOv5-HR-TCM Framework for Automatic 2D/3D Human Pose Estimation for Real-Time Applications*. Last accessed 14 June 2023. 2022. URL: https://www.mdpi.com/1424-8220/22/14/5419.

[13] Thanh Thi Nguyen et al. *Deep learning for deepfakes creation and detection*. last accessed 4 July 2023. 2022. URL: https://www.sciencedirect.com/science/article/pii/S1077314222001114.

[14] Ilias Papastratis. *Deepfakes: Face synthesis with GANs and Autoencoders*. last accessed 4 July 2023. 2020. URL: https://theaisummer.com/deepfakes/.

[15] Adrian Rosebrock. *Facial landmarks with dlib, OpenCV, and Python*. last accessed 4 July 2023. 2017. URL: https://pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/.

[16] Aliaksandr Siarohin. *First Order Motion Model for Image Animation*. last accessed 1 july 2023. 2020. URL: https://youtu.be/u-0cQ-grXBQ.

[17] Luisa Verdoliva. *Media Forensics and DeepFakes: An Overview*. last accessed 9 May 2023. 2020. URL: https://ieeexplore.ieee.org/document/9115874.

[18] Jian Zhao and Hui Zhang. *Thin-Plate Spline Motion Model for Image Animation*. 2022. arXiv: 2203.14367 [cs.CV].

[19] Alan Zucconi. *Understanding the Technology Behind DeepFakes*. last accessed 4 July 2023. 2018. URL: `https://www.alanzucconi.com/2018/03/14/understanding-the-technology-behind-deepfakes/`.

Acknowledgments

To my parents, to whom I owe everything. Thank you for your constant support and help. You are my role models in life, for your love and dedication to our family, for your kindness and patience, for your spirit and passions. You have taught me what's important in life, and have given me the strength and energy to get to where I am today. Thank you.

To my brother and sister; best friends that I'm so lucky to have by my side forever.

To prof. Simone Milani, for following and helping me in this thesis, and to dott. Elena Camuffo, for assisting with this project, as well as my projects in Bergen.

To prof. Carlo Ferrari, for his kindness and willingness to help his students, including me.

To my best friends, Nanni, Carlo, Arianna, Erica, Laura for being my adventure buddies here in Padova.

To Giacomo, for the precious friendship he has given me, and for helping when help was needed.

To Fabio and Martina and Miguel, for being my partners in crime while in Norway. Bergen for alltid!

To Luca, and to the friends I made at Unipd, Andrea, Alberto, Davide, Andrea, Giacomo, Noah, Melissa, Giovanni, for making the exam sessions easier and more fun.

To the wine tasting friends I stole from my brother, to my exchange friends and to Helene Bakken, for making my last semester here in Padova memorable.