IDIAP Research Institute

Learning of non-photorealistic renderings for a caricaturist robot

Johnny BORKHOCHE - SCIPER 296169

Supervisors: Prof. Dr. Bourlard Hervé, Dr. Sylvain Calinon

Doctoral Student: Tobias Loew

Project Report in light of my Autumn Bachelor Semester Project

January, 2022

Abstract

The project's goal is to map an image to a set of trajectories that a robot can then draw on a canvas, by taking the example of portrait caricatures as an example of generative non-photorealistic rendering (NPR) problem. My mission was to focus on the aspect of generating realistic caricatures through warping or distorting input images.

I will be explaining the reasoning behind my choices as well as the constraints I faced which led to me deciding on an image processing approach instead of a deep learning solution. This has led me to interesting results as playing with image transformations has a natural ceiling of some sorts, which is partly due to the fact that I am looking for realistic outputs unlike other projects that have opted for cartoon-like results.

There are many intermediary steps involved, notably computing an average face from a self-created database of portrait images, computing some interesting statistics regarding the so-called 'average face' and finally using these metrics to warp any new image.

Contents

Chapter 1

Introduction

In this paper I wish to introduce my solution to the realistic facial caricaturization problem. I will be explaining step-by-step the reasoning behind my choices as well as showcase some of the interesting results I ended up with.
I have opted for an image processing approach instead of a deep learning solution for reasons that will be expounded in further chapters.

My research is part of a bigger project which had the goal of having a robot hand-draw caricaturized human portrait images on a canvas. This has led to me going down the trail of realistic-looking caricaturizations that maintain the identity of the person, unlike other solutions that have gone for cartoon-like results.

This paper also culminates my work towards my bachelor semester project at EPFL in affiliation with the IDIAP Research Institute. I would like to extend my thanks and respect to all persons that have helped guide, mentor and advise me along my work.

Chapter 2

State-of-the-art

In this section I will be going over my most interesting findings of articles and papers related to this project that I've discovered during my literature review.

2.1 Facial Landmark Detection Techniques

Some foundational methods used from years ago up until this day are to be discussed in this section:

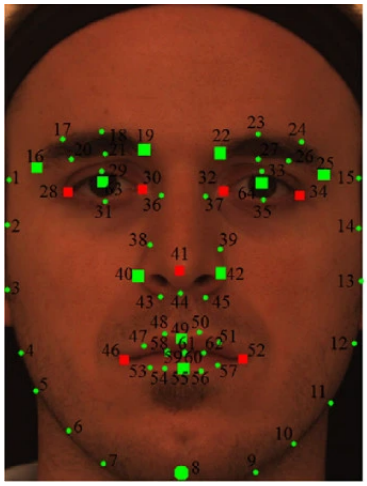
2.1.1 Classification of face landmarking techniques

Face landmarking, defined as the detection and localization of certain characteristic points on the face, is an important intermediary step for many subsequent face processing operations that range from biometric recognition to the understanding of mental states. Despite its conceptual simplicity, this computer vision problem has proven extremely challenging due to inherent face variability as well as the multitude of confounding factors such as pose, expression, illumination and occlusions (Guo et al. (2019)).
There is always some preprocessing before a method engages in landmark detection (Chai et al. (2016)). Typical of these steps are the following: illumination artifact removal, modest geometric corrections, segmentation of the face, use of color information.

Types of Landmarks

It is convenient to consider facial landmarks in two groups, primary and secondary landmarks (see Figure 2.1). For example, the corners of the eyes, of the mouth, the nose tip, and sometimes the eyebrows can be detected relatively easily. These directly detected landmarks are referred to as the primary ones, and they play a more determining role in facial identity and face tracking. The landmarks in the secondary category such as nostrils, chin, cheek contours, non-extremity points on lips or eyebrow

midpoints, eyelids etc. often present scant image evidence, and the search for them is often guided by the primary landmarks. The secondary group of landmarks take more prominent roles in facial expressions, although the demarcation between these two tasks is not always clear-cut (Çeliktutan et al. (2013)).



Primary landmarks		Secondary landmarks	
Number	Definition	Number	Definition
16	Left eyebrow outer corner	1	Left temple
19	Left eyebrow inner corner	8	Chin tip
22	Right eyebrow inner corner	2-7, 9-14	Cheek contours
25	Right eyebrow inner corner	15	Right temple
28	Left eye outer corner	16-19	Left eyebrow contours
30	Left eye inner corner	22-25	Right eyebrow corners
32	Right eye inner corner	29, 33	Upper eyelid centers
34	Right eye outer corner	31, 35	Lower eyelid centers
41	Nose tip	36, 37	Nose saddles
46	Left mouth corner	40, 42	Nose peaks (Nostrils)
52	Right mouth corner	38-40, 42-45	Nose contours
63,64	Eye centers	47-51,53-62	Mouth contours

Figure 2.1: *Classification of 64 landmarks*

Two basic categories of facial landmark detection

First off, we have "model-based methods" such as Activate Shape Model (Cootes et al. (1998)) which consider the face image and the ensemble of facial landmarks as a whole shape. The mode learns 'face shapes' from labeled training images, then at the testing stage they try to fit the proper shape by an unknown face.

On the other hand, we have "textured-based methods" such as Gabor Wavelet Networks (Feris et al. (2002)) which generate landmark candidates independently from each local detector. These result in a score surface for each landmark on the test face, whether the score is the outcome of the matched filter or of the classifier, e.g., SVM. The peaks on the score surface, judiciously chosen in the light of a prior model for face geometry form the landmarks. Algorithms often attempt to solve the combinatorial search problem using various heuristics or invoking learned face models. The enforcement of the prior information plays the role of a regularizer.

2.1.2 Histogram of Gradients (HOG)

According to Dalal & Triggs (2005), the Histogram Of Gradients method produces near-perfect pedestrian detection rates on the MIT pedestrian database. This method converts the image into a series of histograms based on the orientation and magnitude of pixel gradients within the image. An SVM classifier is then used to identify the face

within the image based on the values of the histogram.
While HOG may no longer be considered state of the art, the method is still demonstrating its relevance and flexibility through its direct application in hardware which has significant power consumption advantage over more modern object detection methods that require the use of a dedicated GPU; an example would be NVidia's GTX 1080TI GPU which consumes 250W of power and recommends a 600W power supply, effectively eliminating its usage in embedded systems.

To better illustrate how HoG works, let us look at a quick example:
We consider a 64*128 image, we divide the image into 7*15 = 105 blocks of 50% overlap, with each block being 2*2 cells and each cell 8*8 pixels (Figure 2.2).

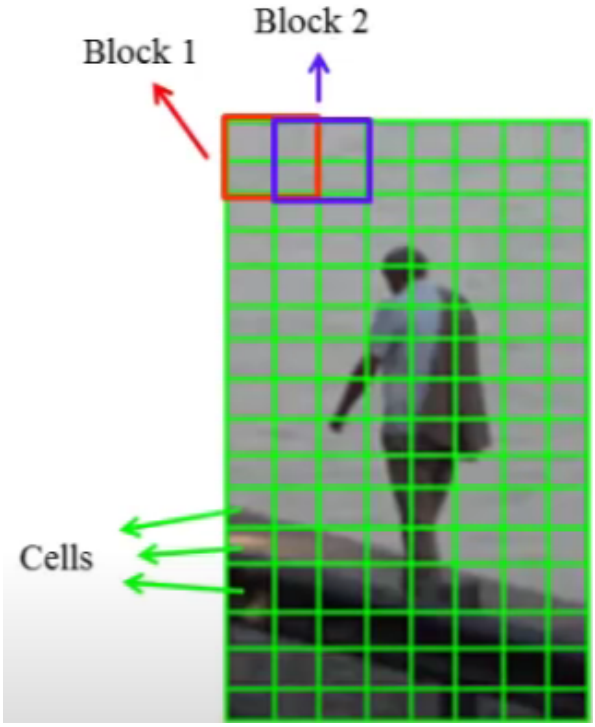


Figure 2.2: *Image divided in cells*

We now need to look at the gradient orientation: if for example we have 85^o, we are between the Bin 70^o and Bin 90^o (Figure 2.3) with a distance of 15^o and 5^o respectively. Therefore, the ratios on the histogram are $15/20 = 3/4$ and $5/20 = 1/4$ at 70^o and 90^o. The histograms are then converted to a 1D matrix and thus we obtain the following result (Figure 2.4)

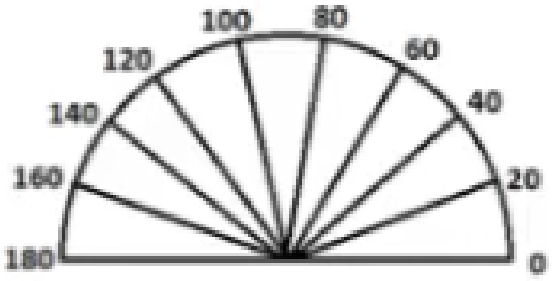
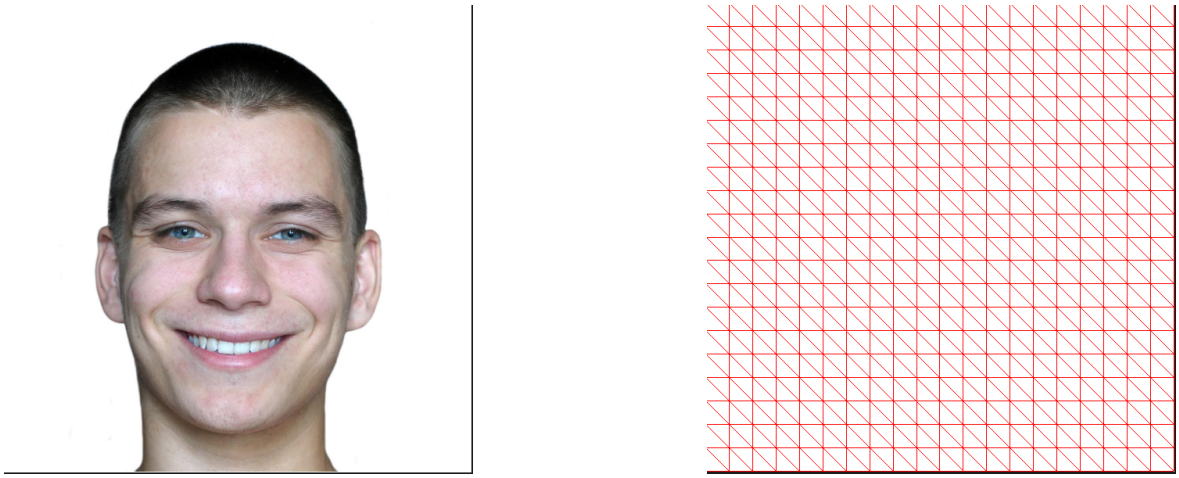


Figure 2.3: *Bin of angles*

Figure 2.4: *HOG final result*

2.1.3 Viola-Jones

According to Viola & Jones (2001), The Viola-Jones method uses a combination of two main techniques: the representation of Haar-like features through construction of an 'integral image' and a series of 'weak' classifiers boosted using the Adaboost learning algorithm. At the very last stage of the cascading classifier is a high-performing perceptron and a decision threshold. Using the boosted classifiers and the perceptron layer Viola-Jones were able to produce a high performing system. While generally not used in more modern applications, Viola-Jones face detection still holds its place in literature as an effective reference method.



(a) *Example portrait image* **(b)** *Triangular grid behind the picture*

Figure 2.5: *Initial example image and its mesh structure*

2.2 Facial Warping Techniques

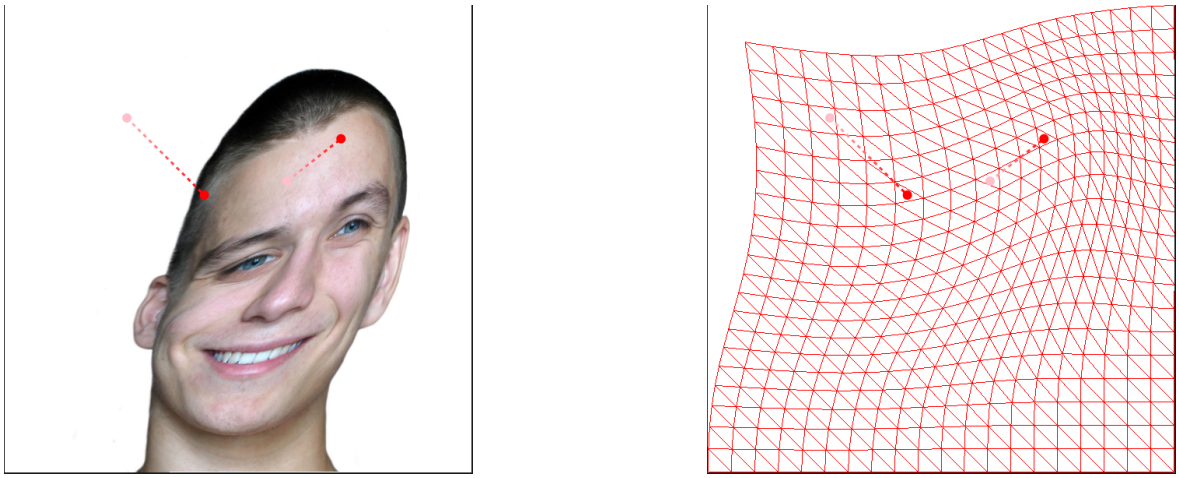
In this part I will showcase some of the most commonly used techniques and/or libraries to distort images. Many of the related papers are pretty old, while most recent technologies come under the form of libraries whose internal workings are not accessible.

2.2.1 Mesh Warping

Mesh distortion is an image warping technique which is driven by the mapping between equivalent families of curves or meshes, arranged in a grid structure (Steyvers (1999*a*)).

The user aligns a rectangular grid of points to key feature locations of images. For warping, the coordinates of the source and destination grids (calculated with coordinates that are weighted averages of the corresponding coordinates of the source grid) are fitted with piece-wise continuous mapping functions. The warping phase then proceeds in two passes. In the first pass, an intermediate destination image is generated in which every row of the image is warped independently of the other rows. Then, in the second pass, the final destination image is created by warping every column of the intermediate image.
It is most easily shown through an example:
The following is done using WebGL (taken from Khronos (n.d.)). We use a random portrait image taken from the web and align it with a triangular grid (Figure 2.5). The transformation is done using a GPU.

By distorting the image in the 2 different places shown in Figure 2.6.a, new uniform points are set on the GPU which then runs a vertex shader to distort the mesh using



(a) *Example image distorted in 2 places* **(b)** *Triangular grid of the image*

Figure 2.6: *Distorted image and its mesh structure*

said uniform points.

It is interesting to note that some might refer to this technique as "grid distortion" and not "mesh distortion". In fact, from my understanding, both are very often used interchangeably in many cases.
Grids are typically a set of simulation elements that have a well defined structure to their alignment with square or rectangular grids being the most prototypical. Meshes are often more general. They may be unstructured and use various shapes of elements, sometimes even mixing elements of different types in the same mesh. (StackExchange (n.d.))

2.2.2 PyChubby

Pychubby (Krepl (2019)) is a package for automated face warping. It allows the user to programatically change facial expressions and shapes of any person in an image.

Its logic can be summarized into two main blocks:

- Landmark Detection: Pychubby relies on the standard 68 facial landmarks framework. Specifically, a pretrained dlib model is used to achieve this task. Once the landmarks are detected one can query them via their index.
- Reference Space Mapping (Figure 2.7): In general, faces in images appear in different positions, angles and sizes. Defining actions purely based on coordinates of a given face in a given image is not a great idea. One way to solve the above issues is to first transform all the landmarks into some reference space, define actions in this reference space and then map it back into the original domain. Indeed, five

selected landmarks are used to estimate an affine transformation between the reference and input space. This transformation is encoded in a 2x3 matrix A used to transform a location of points (x, y). The first two columns of this matrix encodes rotation and scale, and the last column encodes translation. Transforming from reference to input space and vice versa is then just a simple matrix multiplication:

$$\boldsymbol{A} \times \boldsymbol{x}_{input} = \boldsymbol{x}_{ref}$$
$$\boldsymbol{x}_{ref} \times \boldsymbol{A}^{\dagger} = \boldsymbol{x}_{input}$$

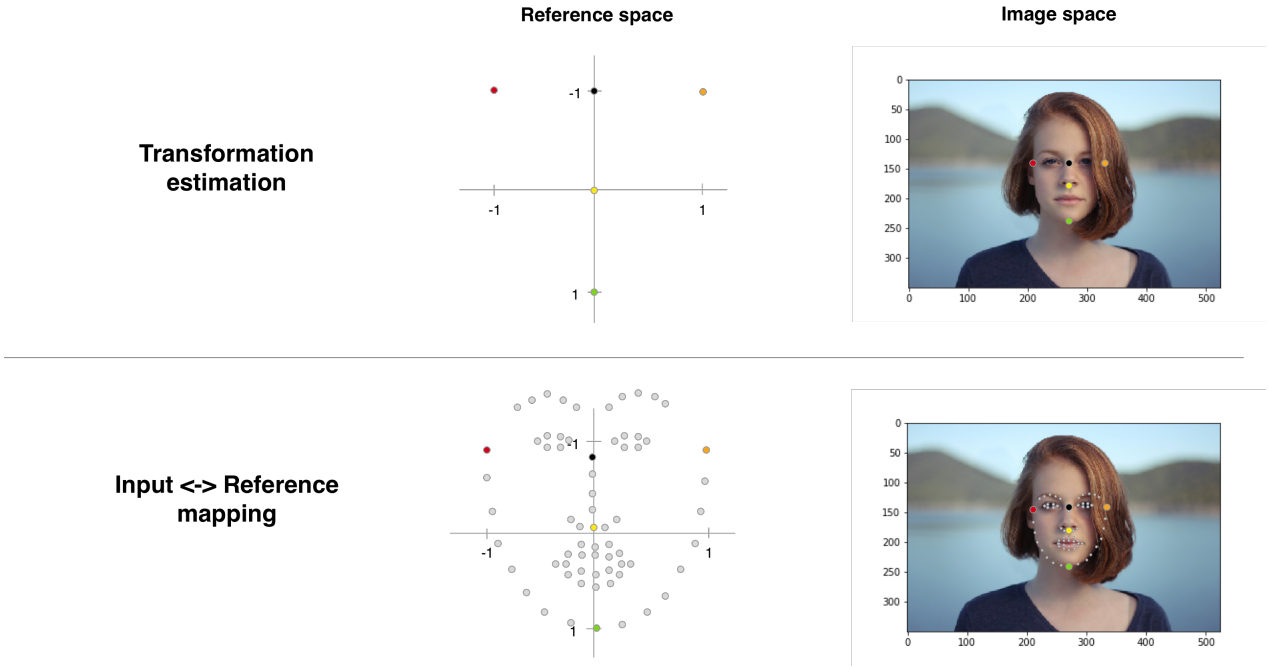


Figure 2.7: *Reference Space Mapping*

2.2.3 Other general techniques

Previously we've discussed the techniques and packages that I personally found to be the most relevant to my work. It is to be exepected, however, that these are not the only procedures for facial distortion. In fact, some older methods consider morphing to manipulate two-dimensional human face images and even three-dimensional models of the human head. Applications of these techniques show how to generate composite faces based on any number of component faces, how to change only local aspects of a face, and how to generate caricatures of faces.

Typically, the term *morphing* is used to describe an image processing method that transforms an image A to another image B, and vice-versa. The morphing algorithm

can also generate any image residing on the continuum between A and B. According to Steyvers (1999*b*), there are two distinct phases in the calculation of a morph between A and B: warping and cross-dissolving. First, the image A is warped to A' and the image B is warped to B', which means that the shapes A and B are distorted toward A' and B' such that A' and B' have similar shapes. Then, A' and B' are cross-dissolved, meaning that the colors or grayscale values of images A' and B' are combined to form a new image that can share both shape and color aspects of A and B.

Furthermore, there exist many methods that can be considered for facial expression transformation which could have been considered during the caricaturization process. As mentioned by Dong et al. (2011), we can mainly define two categories:

- Warping based on scattered point interpolation (algorithm based on the radial basis function) can produce realistic warped images. But usually, the RBF functions selected are very complex, so the results are slow, and doesn't guarantee stability at the borders of the warped image.
- Warping based on fragments, mainly based on **triangulation** and/or on **grid distortion** which was mentioned previously. Triangulation is good for local warping in faces, but pre-processing of images divided into triangular pieces is quite complex. As I'll discuss in the next chapter, however, triangulation techniques can and will be used in this project, as they are quite useful.

2.3 Generative Adversarial Networks

Most of the previous methods mentioned had to do with image processing, yet deep learning approaches are actually those that are most used today. Indeed, since the creation of Generative Adversarial Networks (GAN) in 2014, most of today's modern architectures can achieve pretty satisfying results by using GANs. In fact, its applications have reached tons of varied fields, ranging from fashion, art and advertising to video games and even transfer learning.

According to Goodfellow et al. (2014), the idea behind GANs is the following: Two neutral networks contest with each other in a game (in the form of a zero-sum game, where one agent's gain is another agent's loss).

Given a training set, this technique learns to generate new data with the same statistics as the training set. For example, a GAN trained on photographs can generate new photographs that look at least superficially authentic to human observers, having many realistic characteristics. Though originally proposed as a form of generative model someil for unsupervised learning insomenie, GANs have also proved useful for semi-supervised learning, fully supervised learning, and reinforcement learning.

The core idea of a GAN is based on the "indirect" training through the discriminator, which itself is also being updated dynamically. This basically means that the

generator is not trained to minimize the distance to a specific image, but rather to fool the discriminator. This enables the model to learn in an unsupervised manner.

2.3.1 WarpGAN

As a first example, WarpGAN (Shi et al. (2019)) is a fully automatic network that can generate caricatures given an input face photo. As we observe in Figure 2.8, WarpGAN is capable of generating caricatures that not only preserve the identities but could also output a diverse set of caricatures for each input photo when tweaking parameters.

Figure 2.8: *WarpGAN example*

2.3.2 CariGAN

On the other hand, the very first GAN for unpaired photo-to-caricature translation is called CariGAN (Cao et al. (2018)). As seen in Figure 2.9, the pictures generated by CariGAN somewhat resemble hand-drawn caricatures and also persevere the identity of the person drawn. Moreover, CariGAN allows users to control the shape exaggeration degree and change the color/texture style by tuning the parameters or giving an example caricature.

Figure 2.9: *CariGAN example (last picture is carigan)*

Chapter 3

Pipeline

This section will discuss and explain the methods I've decided on to try and realistically caricaturize portrait images. From the previous section, we've seen different techniques that could be used at different stages to achieve wanted results; we will be using or taking inspiration from some of them.

3.1 Architecture

By observing the results obtained through Generative Adversarial Networks such as CariGAN and WarpGAN, we realize that despite their impressiveness, they are not exactly what we are looking for as the outputs have turned out cartoon-like. However, as stated initially, my goal is to try and caricaturize human portraits while keeping them as *realistic* as possible.
In addition, after a few weeks of discussion with my advisor and project supervisor, it was concluded that there might not be a need to resort to a machine learning model to caricaturize the images, and that going for an image processing solution might be enough for what we need. This decision came with both advantages and disadvantages:

- **Advantage** There is no more need to find a well-suited and large enough database of portrait images with similar lighting, angles and no obstructions that could be used to train an appropriate model.
- **Advantage** There exists a couple of libraries (notably *OpenCV* and *dlib*) with very useful modules for our case.
- **Disadvantage** As it will become clear later on, using only image processing will allow us to achieve some results, but it will also make us hit a natural ceiling pretty quickly; without redrawing from scratch it seems impossible to achieve outputs as good as those of CariGAN and WarpGAN.

In light of everything that has just been said, I took a couple of weeks to define and refine a possible approach of tackling this problem using computer vision and finally came up with the following architecture (Figure 3.1):

1. Compute the average face from a small database of portrait images
2. Use the average face to compute a set of statistics and average values
3. When there's a new input image, measure it's biggest difference when compared to the average face statistics and hence choose what parts should be exaggerated
4. When the parts to warp are chosen, go ahead and warp them to obtain the final output

It is interesting to note that there are some slight differences when it comes to men and women portrait; men faces are on average *bigger* than women faces so this distinction will play a role later on.

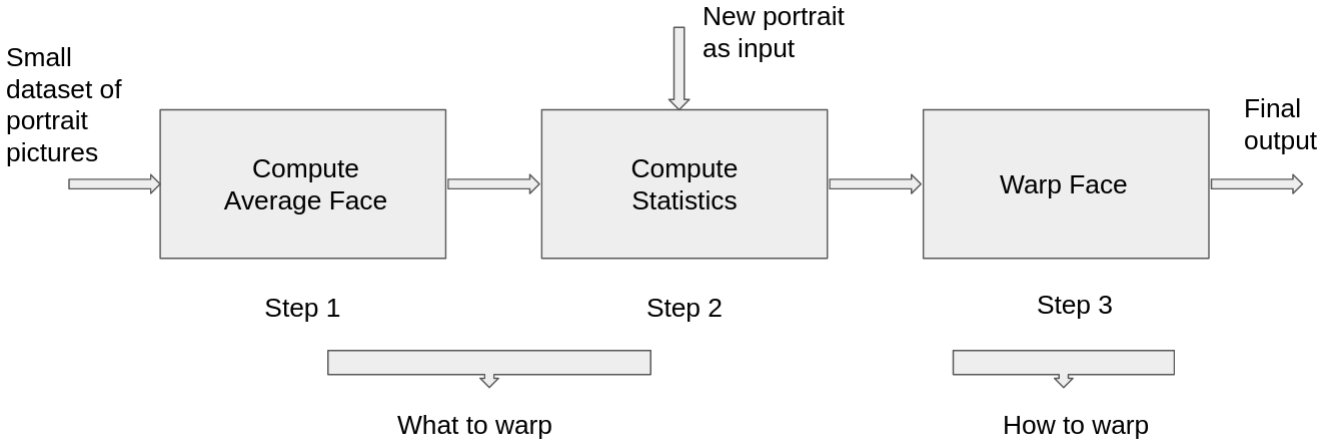


Figure 3.1: *Realistic Caricaturization Pipeline*

3.2 Computing an Average Face

Face averaging started with Francis Galton (Charles Darwin's cousin) back in 1878 when he came up with a new photographic technique for compositing faces by aligning the eyes. He came up with the idea of calculating an 'average criminal face' to try and more accurately guess who was a criminal!
One thing he had noted is that the average face was always more attractive than the faces used to create it. In fact, In one amusing experiment in 2002 (Regensburg (n.d.)), researchers averaged the faces of 22 miss Germany finalists of 2002. People rated the average face to be much more attractive than every one of the 22 contestants, including miss Berlin who won the competition (Figure 3.2).

An average face is also more symmetric because the variations on the left and right side of the face are averaged out.

Figure 3.2: *Miss Germany Experiment*

The question, however, remains. How do we compute an average face?

Database Creation

The very first required step is to set up a small database (about 20 pictures) of portrait images. I used free online images to make up my own. These will be the images used to compute an average face.
Note that this was only possible due to the fact that not many images were needed to be gathered, but I also had to make sure that all of the pictures were front-facing and looking straight at the camera since this will play a big role in the facial feature detection engine we will be using. If I was hoping to train a ML model, then gathering a good amount of suitable free pictures would have taken me weeks at the very least.

Facial Feature Detection

Next, for each image in the database, we use the *Dlib library*'s (King (2015)) pre-trained facial recognition model to detect the coordinates of the recognized 68 facial landmarks. See Figure 3.4 for an example.

Coordinate Transformation

We run into an issue right off the bat; all images in the database are not necessarily of the same size. We must hence find a way to normalize the faces and bring them to an identical reference frame. A simple way of doing this would be to warp the detected faces to a 600x600 image where the origin is on the bottom left such that the left corner of the left eye is at pixel location $(180, 200)$ and the right corner of the right eye is at pixel location $(420, 200)$. We define this as the **output coordinate system** whereas the coordinates of the original image is the **input coordinate system**.
This was done to make sure that the eyes were on a horizontal line and that the face was centered at about of the height from the top of the image, I chose the left and

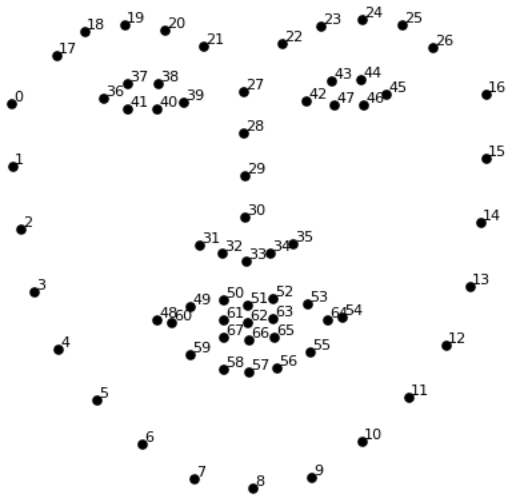


Figure 3.3: *Dlib 68 facial landmarks with their indices*

Figure 3.4: *Portrait of Barack Obama with landmarks detected by dlib*

right eye corners to respectively be at $(0.3{*}width, height/3)$ and $(0.7{*}width, height/3)$.

Since we also have the location of the corners of the eyes of the original image i.e. in the input coordinate system (they're landmarks 36 and 45), we can hence calculate a similarity transform that encodes rotation, translation and scale to transform all pixels in the input coordinate system to the output coordinate system. This is very simply done using an *OpenCV* function called *estimateRigidTransform*. A clear example can be seen on Figure 3.5.

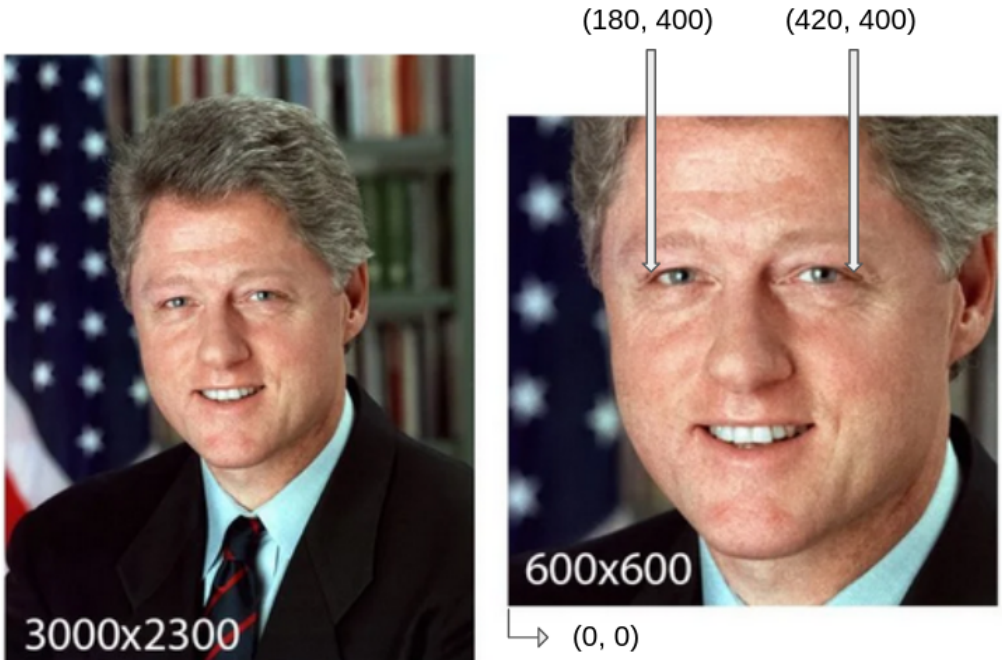


Figure 3.5: *Coordinate Transformation example*

Face Alignment

We previously transformed the input coordinate system to the output coordinate system. This resulted in same-size images in the database with the eye corners well aligned and on the same horizontal line. One might be tempted to now directly obtaining the average face by averaging the pixel values of these images. Unfortunately, doing so would end up with a pretty bad output as all of the facial features apart from the eyes are not aligned. See Figure 3.6 for an example.

Figure 3.6: *Naive face averaging result*

As we have the position of the 68 facial landmarks before and after the alignment, to solve this, we can divide the face into triangular regions and align them before averaging the pixel values. And so, to calculate the average face where the features are aligned, we must first calculate the average of all transformed landmarks in the output coordinate system i.e. by averaging the x and y values of the landmarks in the output image coordinates.
After obtaining the landmark locations for the average face, we add 8 boundary points on the output image (in the 4 corners and middles between the corners). With this, we calculate a Delaunay Triangulation which allows us to break the image into triangles. The result is a list of triangles that are represented by the indices of points taken from the 76 total points (68 landmarks + 8 boundary). See Figure 3.7 for an example (without the boundary points).

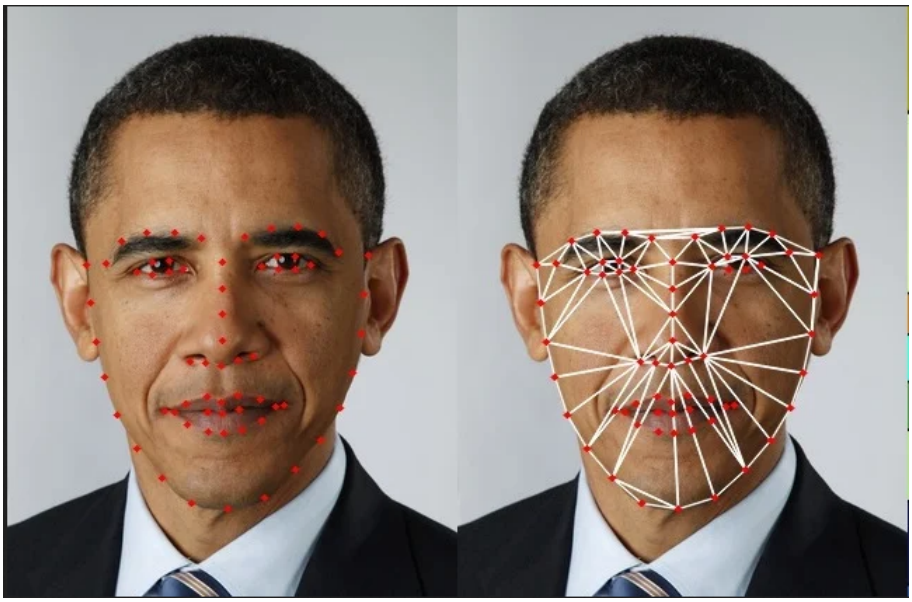


Figure 3.7: *Delaunay Triangulation example*

Triangle Warping

Now that we've found a Delaunay triangulation (that we call T_{final}) of the average values of the facial landmarks we initially calculated, all we have to do is also calculate a Delaunay Triangulation for every face in the database as well (which will be called T_i with i ranging in the number of images in the database) and finally calculate an affine transform using OpenCV from every T_i to T_{final} and apply this transformation on every respective unaligned image in the database.

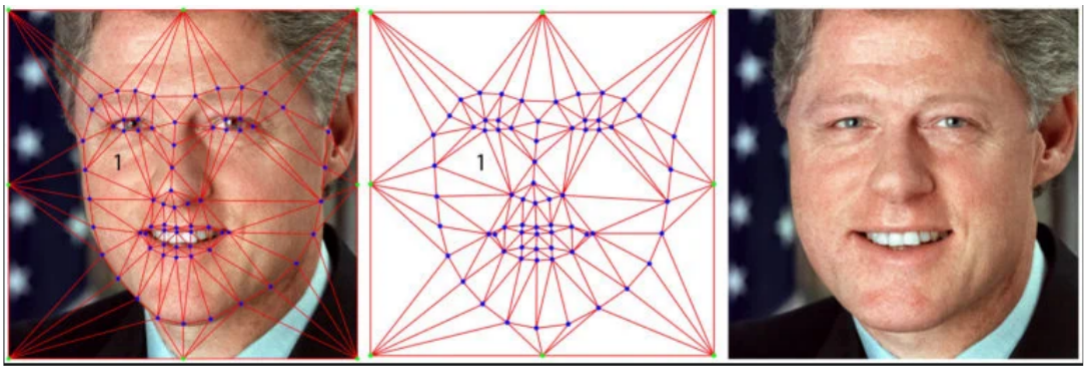


Figure 3.8: *Triangle warping example*

Face Averaging

When the previous step is applied to all images in the database we obtain correctly warped faces proportional to the average landmark coordinates.
We can now simply add the pixel intensities of all warped images and normalize them to obtain our final average face! See Figure 3.9 for an example.

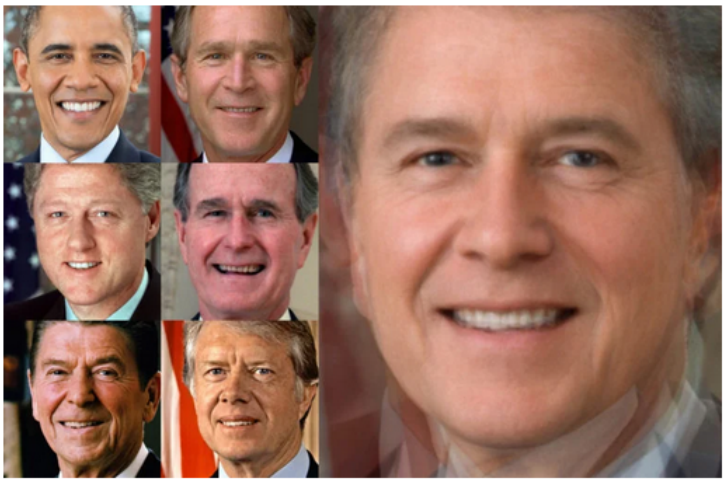


Figure 3.9: *Average Face example: Carter to Obama*

3.3 Computing Statistics

Now that we've established a good method to compute an average face of the pictures in our small database, we would like to find out some interesting statistics taken from

said faces with the goal of interpreting the results. Feel free to check Figure 3.3 to remember what the 68 detected facial landmarks look like.
Looking at a human face, I've heuristically decided on computing 7 statistics that I found most interesting:

- Average face width in three different spots: top, middle, bottom
- Average face length in two different spots: left and right parts of the face
- Average mouth length
- Average mouth width

I also had to figure out a way to actually find these values. This ended up being simple enough; I more or less already did parts of it in the earlier steps. Indeed, as we've already obtained the average face that will be used when comparing with any new portrait input to the system, all that's left to be done is to track the distance between the 68 different landmarks on that average face and pick out the most interesting ones. The plan is simple:

- First detect the 68 landmarks of the average face (see Figure 3.8)
- Convert their positions into an array of 68 pixel (x, y) coordinates and place it in a text file
- Calculate the distance between the landmarks that correspond to the 7 statistics chosen earlier, namely;

    1. Average face width in three different spots: top *landmark indices 1 and 15*, middle *landmark indices 3 and 13*, bottom *landmark indices 6 and 10*
    2. Average face length in two different spots: left *landmark indices 7 and 21* and right *landmark indices 9 and 22* parts of the face
    3. Average mouth length *landmark indices 51 and 57*
    4. Average mouth width *landmark indices 48 and 54*

To better visualize what each statistic I am calculating is, imagine a straight line between the indices specified just above for each one; this empirically corresponds to the distance in pixels between the landmarks and hence is a measure of "how big/small" the statistic is.

This is the strict equivalent of going through all of the processed images in our database, computing their 68 landmarks, converting their coordinates to an array of coordinates and then picking out the interesting coordinates that correspond to the wanted statistics and finding the average distance between them.

By following the previous steps, we are now able to compute some interesting statistics about the average face and use them for the next step of portrait warping. The results will be presented in chapter 4.

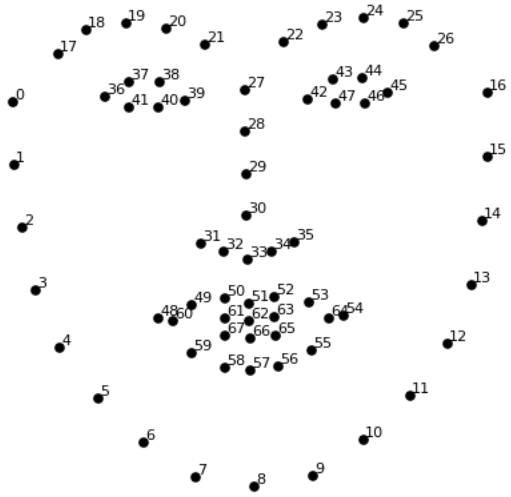


Figure 3.10: *Reminder: dlib 68 facial landmarks with their indices*

3.4 Warping a portrait

At this point, we've not only calculated an average face from the portrait images in our database (that we had to preprocess first), but we've also managed to compute some interesting statistics regarding it. In the last part of our pipeline which has takes a new portrait image as an input, compares it to the average face, chooses which parts of the new face are the most different from average and exaggerate those features.

When a new image is provided to the system, the first step is to also transform and align the image, just like previously described in the first 2 steps of Chapter 3.2; facial landmark detection and coordinate transformation. This is simply to make sure that the image size is also 600x600 pixels (just like the average face) and that the eyes are on a horizontal line (which makes sure that the head isn't tilted).

Moving on, we also compute the distances computed in 3.3, namely:

- Average face width in three different spots: top, middle, bottom
- Average face length in two different spots: left and right parts of the face
- Average mouth length
- Average mouth width

We do this for the new image as this will constitute a basis of comparison with the average face we've found. The idea here is to choose what parts of the new portrait need to be warped, and this is done by seeing which "areas" of the face are the most *distant* from the average face.

As such, we compare all 7 statistics of the new face to those of the average face, and see which ones are the most 'far away'. There are multiple ways to do this, with some being better than the others.
The very first thing I did was to measure the distance between statistics by simply calculating the **distance ratio** of the differences. The formula is very simple, it is the following:

$$\gamma = \frac{\alpha - \beta}{\alpha} \tag{3.1}$$

where

- γ is the distance ratio
- α is the average face's distance between the landmarks of the chosen statistic
- β is the new portrait's distance between the landmarks of the chosen statistic

It worked out fine, I was able to find which ratios were the absolute largest (hence what statistics of the new portrait were the furthest away from the average), but I also found the direction of the difference i.e if the new image's landmark is actually bigger or smaller than the average.

Using ratios is fine, but I found a better option: the **Mahalonobis distance**. Introduced in 1936, it is a measure of how many standard deviations is a point P (which here represents the coordinates of a landmark of the new face) from the mean of a distribution D (which represents the coordinates of that same landmark of all faces used to compute the average face) (Joseph et al. (2013)). The reason this is useful is because the Mahalonobis distance also takes into consideration the correlations of the data set, unlike simply calculating the ratio. So instead of using only the means of the statistics, I can calculate the covariance matrices of all of the interesting landmarks that I have and use the *SciPy* library to calculate the distance between the new image's landmarks and those of the average face.

Now that we've established a way of finding the facial regions that are the furthest from the average face, it is time to distort them. After some testing and deliberation, I've chosen to find and caricaturize the 2 most distant regions of the face, and not just 1. The reason behind this choice is that warping only 1 area of the face didn't seem to be enough to be considered a caricature, while 3 or more areas being distorted went beyond a caricature. 2 seemed like the sweet spot.
To warp the chosen parts, I first considered the triangle warping method used in chapter 3.2. This unfortunately did not work out as in that step we had both an initial triangulation T_i and a final triangulation T_{final} (of the average face) that we were trying to reach. This made it so that we could easily calculate a transformation T_i from to T_{final}. However, in our current step, we can only find the triangulation T_i of the

processed new portrait image, but we do not know what the final triangulation will be like, hence we cannot use it to find a transformation.

What we will do, however, is something somewhat similar. As we know which landmarks constitute the 2 statistics of the new face that we will be warping, and as we also know how many standard deviations these statistics are from the average, we define a general measure of *shifting* their position. Taking Figure 3.11 as an example, we can define the **scale** and **angle** of the landmark distortion that we want to enforce. Empirically, the further the statistic is from the average, the bigger the scale of the caricaturization will be. For a specific distortion action (such as changing the size of the mouth for example) the absolute value of the angle will remain the same, but its sign will depend on whether or not the selected region is smaller or bigger than the average (and hence defining whether it has to be minimized or exaggerated).
With the scale and angle defined, we offset the coordinates of the affected landmarks by an appropriate amount and generate a *pixel displacement field* which is essentially a coordinate transformation from the old landmark to the new landmark points. We then warp all pixels in the affected area using this field to obtain the exaggerated facial region.

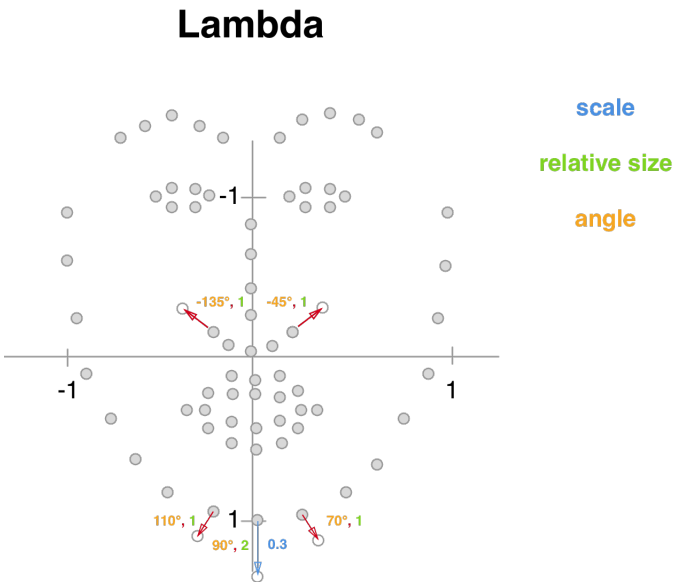


Figure 3.11: *Scheme used to caricaturize the face*

Here are a few examples of different possible caricaturizations made using the method just mentioned (Figure 3.12).

(a) *Original Image*

(b) *Image with bigger nose and longer chin*

(c) *Image with thinner bottom face*

(d) *Image with wider bottom face*

Figure 3.12: *Image warping example*

Chapter 4

Results

Following along the methods described in chapter 3 section 2, I self-created a small database of about 20 portrait images which I then computed the average face of, see Figure 4.1. It turned out pretty good, I am happy with how the facial features are indeed well-aligned.

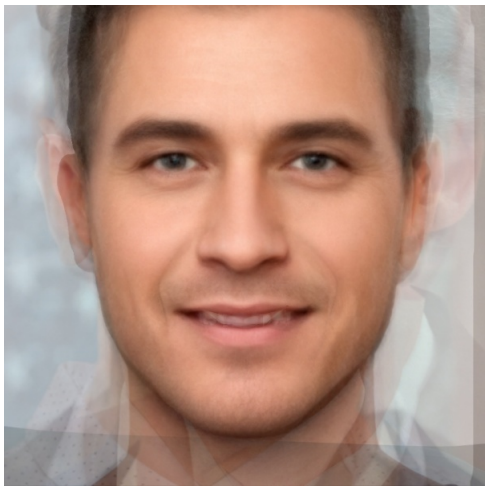


Figure 4.1: *Average face computed from my self-created database*

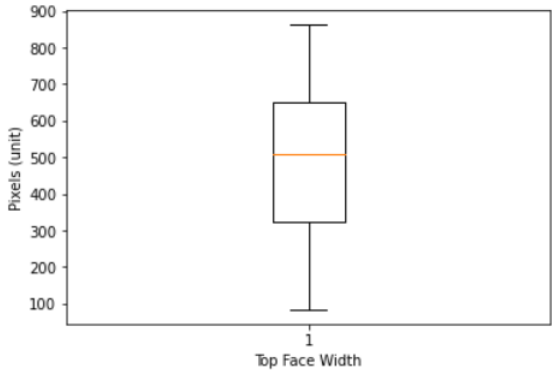
Using this average face, I set out to compute the statistics described in chapter 3 section 3, and ended up finding an interesting distribution with their average values. I chose to represent my results using box plots as they represent a good visual summary of the data.
All and all, I've calculated distributions for the 7 statistics mentioned in the previous chapter:

1. Top of face width
2. Middle of face width

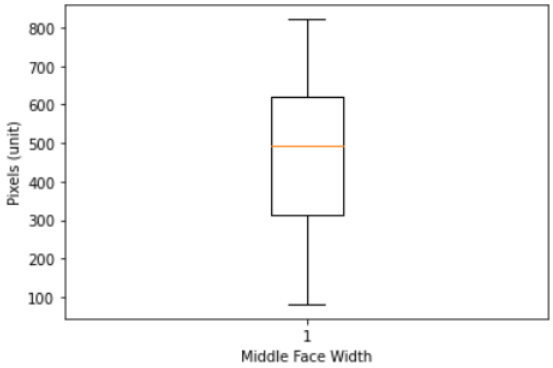
3. Bottom of face width
4. Left side of face length
5. Right side of face length
6. Mouth length
7. Mouth width

They can be seen on Figures 4.2 through 4.4.

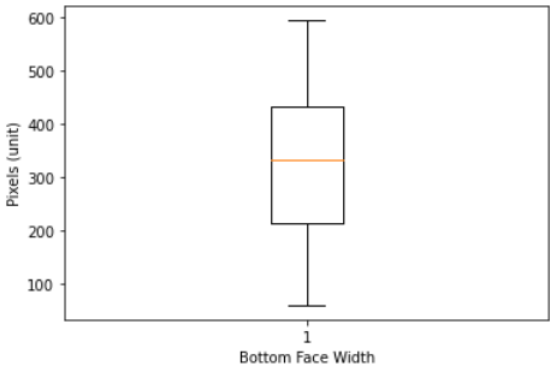
I also included a plot of all 7 statistics next to each other (Figure 4.5) to get a sense of how much bigger or smaller the different statistics are compared to each other, on average.



(a) *Distribution obtained for the measurement of the width of the top of the average face*

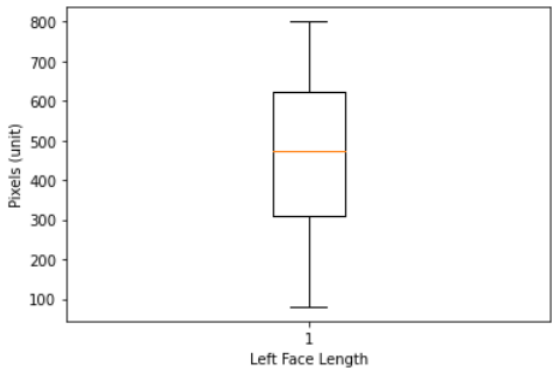


(b) *Distribution obtained for the measurement of the width of the middle of the average face*

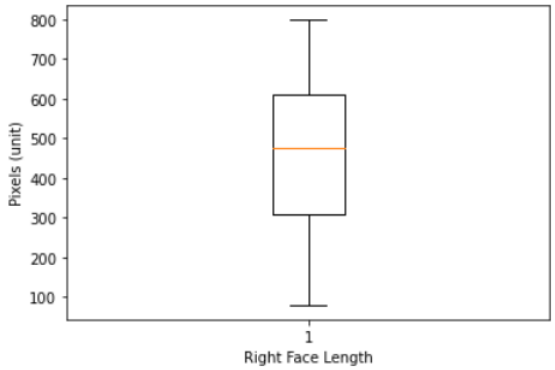


(c) *Distribution obtained for the measurement of the width of the bottom of the average face*

Figure 4.2: *Different distributions obtained for the face width*

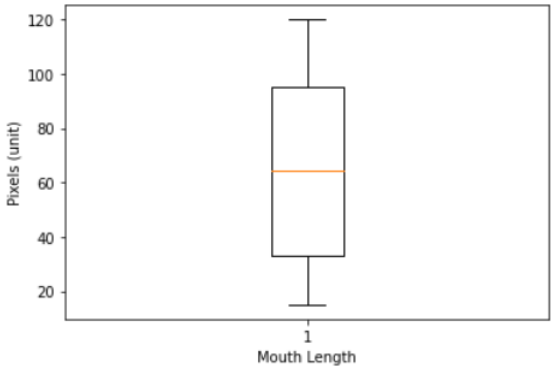


(a) *Distribution obtained for the measurement of the length of the left side of the average face*

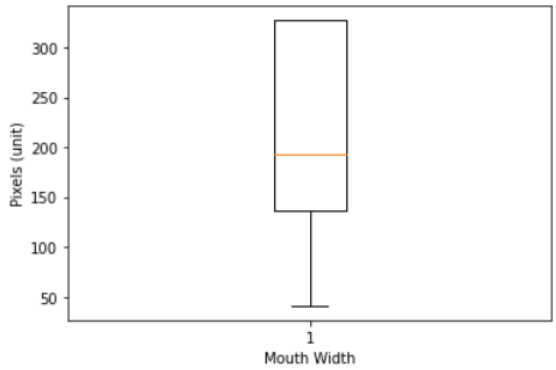


(b) *Distribution obtained for the measurement of the length of the right side of the average face*

Figure 4.3: *Different distributions obtained for the face length*



(a) *Distribution obtained for the measurement of the length of the mouth of the average face*



(b) *Distribution obtained for the measurement of the width of the mouth of the average face*

Figure 4.4: *Different distributions obtained for the mouth*

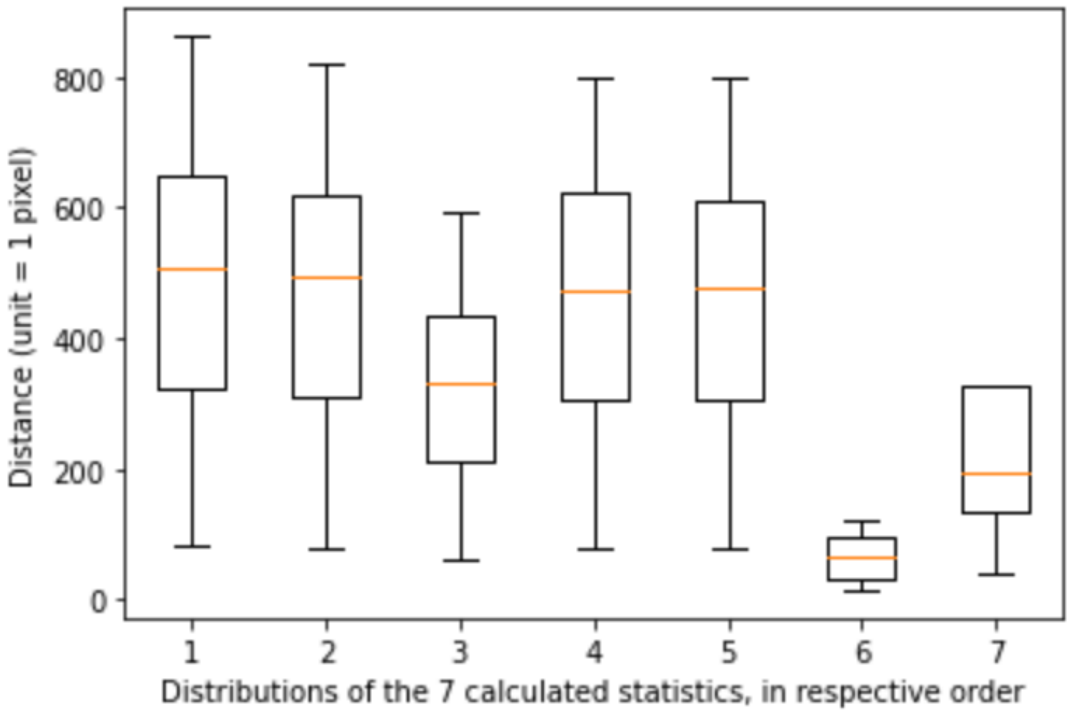


Figure 4.5: *All of the previous 7 distributions shown in order*

Finally, I set out to caricaturize new portrait images that aren't in my database to see how well the method I came up with in chapter 3 section 4 performs. Of course, as mentioned in the previous chapters, all new images are also pre-processed in part, in the same way of the pictures in the database (namely warped to 600x600 pixels and have the eyes on the same horizontal line only).

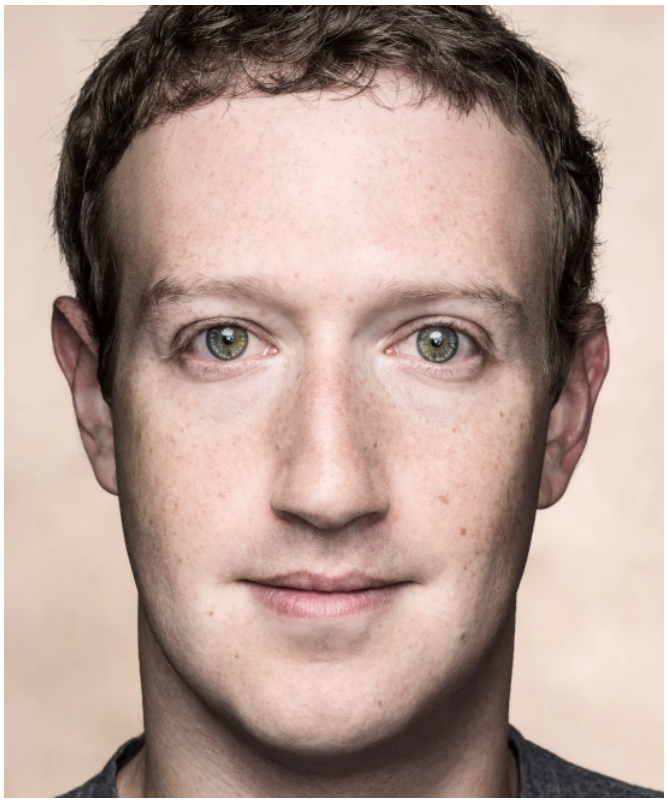
I've chosen 2 male portraits: one of Mark Zuckerberg (Figure 4.6 initially), and one of Elon Musk (Figure 4.7 initially).

For Zuckerberg, I calculated the Mahalonobis distance between his 7 statistics and those of the average face, and I found out that on average, his lips are thinner than the average face and that the bottom of his face is wider. Hence I warped his face to reflect this difference which can be seen in Figure 4.6.

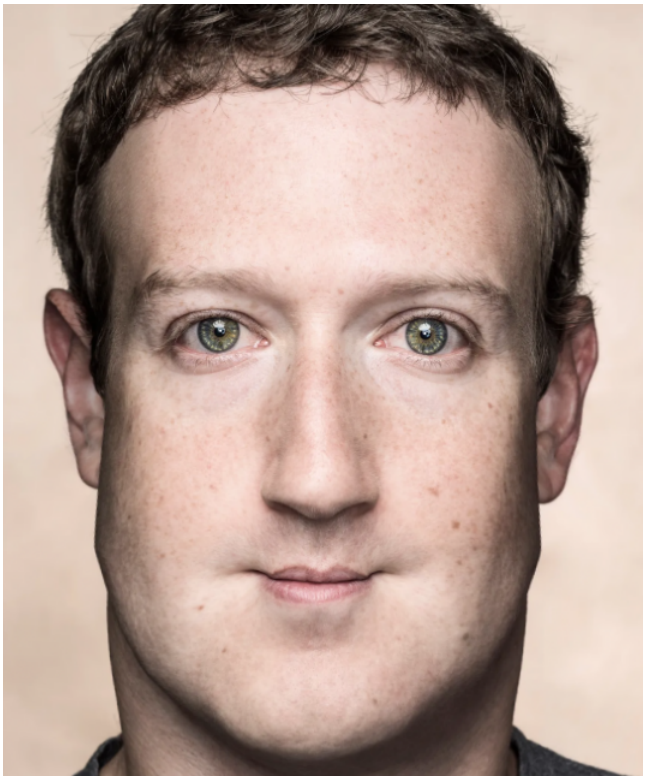
As for Musk, using the same method I found that his mouth is actually bigger than average, and that his forehead is also larger than average. So I set out to enlarge them even further, as much as possible, while trying to preserve his 'facial identity' (as I call it). See Figure 4.7 for the results.

Note that all of the statistics of the new portrait images are bigger/smaller than those of the average face, but I chose the biggest 2 distances to warp as mentioned previously. These 2 statistics are the ones I discuss above.

One thing I also found is that, overall, using the 7 Mahalonobis distances calculated from both Zuckerberg and Musk pictures, I found that overall, Musk is somewhere between 12 to 20% "away" or "different" from the average face when it purely comes to distances while Zuckerberg is about 20 to 35% "different" from the average face.
This means that Elon Musks face is computationally closer to the calculated average face than Mark Zuckerbergs face.



(a) *Mark Zuckerberg initial portrait*



(b) *Caricaturized Zuckerberg with thinner lips and wider bottom face*

Figure 4.6: *First warp example*

(a) *Elon Musk initial portrait*

(b) *Caricaturized Musk with larger forehead and bigger mouth*

Figure 4.7: *Second warp example*

I thought that it would also be interesting to compute the statistics of female portrait and compare it with the a male average face. Using Figure 4.14 as the initial woman portrait, and by calculating its statistics and Mahalonobis distance to the average face, it turns out that she has a longer face than average as well as bigger lips. So I reflected these changes by caricaturizing them in Figure 4.15.

Figure 4.8: *Woman initial portrait*

Figure 4.9: *Caricaturized woman portrait with longer face and a bigger mouth*

Comparing the raw values I obtained for the woman to those of Musk and Zuckerberg, I had a pretty cool realization: there is actually about a 30% scale difference between the Mahalonobis of the woman and those of the men; in other words, the men values were about 30% "larger" than those of the woman.
This makes sense as on average women have smaller faces/features than men.

Chapter 5

Discussion

Regarding the methods discussed in the previous sections, I am quite pleased with the fact they worked out pretty well. Coming into this project at the start, I had no idea regarding what I was going to do or how I was going to do it. Hence, seeing the results of my own pipeline is quite rewarding.

Overall, all of the steps discussed in Chapter 3 hammered out as intended; the database creation took some time to find good pictures (well angled etc) but the resulting average face turned out pretty clean; the Delaunay triangulation method ended up being a great precursor to the face averaging.
Furthermore, the 7 statistics computed are somewhat empiric; I've come up and chosen them as I think that they more or less represent the most interesting areas in the human face; real artists definitely consider these distances when drawing a portrait. One might choose to calculate even more arbitrary statistics, but I don't think that they would be too interesting or bring out much more value than the current ones.
What was left was to choose which parts to exaggerate using the computed distributions, and I chose to exaggerated (or minimize) the "top 2" statistics as choosing only 1 wouldn't qualify the exaggeration as a caricature while anything more than 3 would be the equivalent of changing the entire face.

Arguably, the most limiting step is the final one; the exaggeration of the chosen facial features. Keep in mind that my goal was to caricaturize human portraits while keeping them as realistic as possible (and not having cartoon-like results as seen in WarpGAN and CariGAN). It goes without saying that I would also like to persist the identity of the person whose portrait I am warping, and not just transform them into an exaggerated mess that doesn't resemble them anymore.
With these considerations in mind, I felt like I hit the 'natural' ceiling of simple image processing; I couldn't exaggerate the faces past a certain limit, or I would end up with monstrosities that lost the realistic aspect that I wanted to keep. See Figure 5.1 as an example; I maximized Zuckerbergs bottom face, but after a certain extent it stops looking natural and doesn't coincide with the realistic results that I was looking for-

ward to. After all, the goal of the bigger project was to then have a robot draw the caricaturized face.

This natural limit of image processing had me feeling a bit unsatisfied, and try as I may, I couldn't find a way to breakthrough this ceiling. I think that if we had opted for a deep learning solution, we might have been able to disregard this limit, but in that case, the entire pipeline would have to be scrapped for something more suited to creating machine learning models.

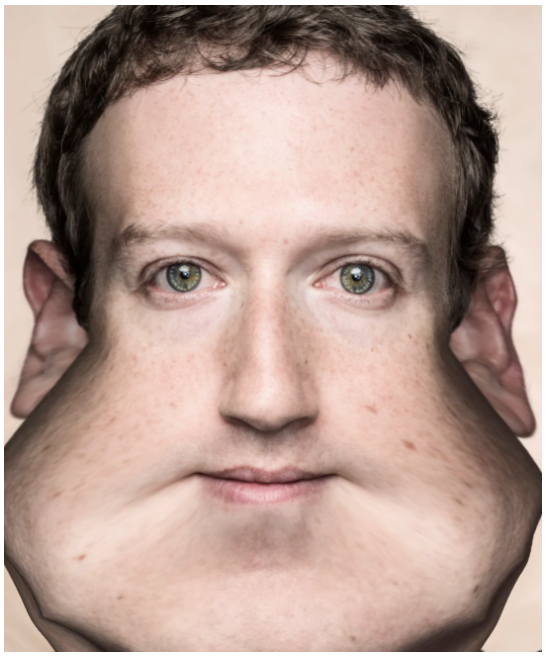


Figure 5.1: *Monstrosity achieved from exaggerating way too much*

Chapter 6

Conclusion

This semester project was about realistically caricaturizing human portrait images so that a robot could then draw the picture on a canvas. I worked on the caricaturization part.

This project was a great learning experience for me as nothing was pre-defined; I had to find and browse related papers to then come up with my own pipeline and methods that are suited to my goal. I am quite satisfied with how it turned out.
Overall, I went through the creation of a small database, the computation of an average face using said database, the calculation of interesting statistics regarding the average face and finally defining metrics which were used exaggerate chosen facial features of new portait images.

This was my first image processing experience. It is safe to say that I learned a whole lot about OpenCV, dlib, image manipulation techniques, facial recognition and so on. I am happy with the results of every step of the pipeline, but the final results weren't as good as I was expecting. Using image processing to caricaturize a human face made me hit some type of natural limit for the 'realistic' part of the project; if I exaggerated the warp too much then the end result wouldn't fit the criteria I was looking for. But overall I think what I achieved is pretty good with regards to the time constraints that I had and my initial knowledge of the subject.

For anyone looking to do the same, and if you have the capabilities, I think that for the best looking results a deep learning solution, but it remains to be seen as to how to escape the cartoon-like results that appeared with other projects such as CariGAN and WarpGAN.

Lastly, I would like to thank my PhD supervisor Tobias Loew for guiding me every step of the way and providing me with very helpful insights, as well as Dr. Sylvain Calinon for giving me the opportunity to work on this interesting project.

Bibliography

Cao, K., Liao, J. & Yuan, L. (2018), 'Carigans: Unpaired photo-to-caricature translation', *ACM Transactions on Graphics (Proc. of Siggraph Asia 2018)* .
URL: *https://cari-gan.github.io*

Çeliktutan, O., Ulukaya, S. & Sankur, B. (2013), 'A comparative study of face landmarking techniques', *EURASIP Journal on Image and Video Processing* **2013**(1), 13.
URL: *https://doi.org/10.1186/1687-5281-2013-13*

Chai, X., Wang, Q., Zhao, Y. & Li, Y. (2016), 'Robust facial landmark detection based on initializing multiple poses', *International Journal of Advanced Robotic Systems* **13**.

Cootes, T. F., Edwards, G. J. & Taylor, C. J. (1998), 'Active appearance models', pp. 484–498.

Dalal, N. & Triggs, B. (2005), 'Histograms of oriented gradients for human detection', **1**, 886–893 vol. 1.

Dong, L., Wang, Y., Ni, K. & Lu, K. (2011), 'Facial animation system based on image warping algorithm', pp. 2648–2653.

Feris, R., J., G., Toyama, K. & Krüger, V. (2002), 'Wavelet hierarchical wavelet networks for facial feature localization'. ISSN ; -; Wavelet Hierarchical Wavelet Networks for Facial Feature Localization ; Conference date: 19-05-2010.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. (2014), 'Generative adversarial networks'.

Guo, X., Li, S., Yu, J., Zhang, J., Ma, J., Ma, L., Liu, W. & Ling, H. (2019), 'Pfld: A practical facial landmark detector'.

Joseph, E., Galeano, P. & Lillo, R. E. (2013), 'The mahalanobis distance for functional data with applications to classification'.

Khronos (n.d.), 'Webgl'.
URL: *khronos.org/webgl*

King, D. (2015), 'Dlib'.
URL: *dlib.net*

Krepl, J. (2019).
URL: *https://github.com/jankrepl/pychubby*

Regensburg, U. (n.d.).
URL: *https://www.uni-regensburg.de/Fakultaeten/phil$_F$ak$_I$I/Psychologie/Psy$_I$I/beautycheck/engli*

Shi, Y., Deb, D. & Jain, A. K. (2019), 'Warpgan: Automatic caricature generation'.

StackExchange, C. (n.d.), 'Difference between grid based and mesh based methods'.
URL: *https://scicomp.stackexchange.com/questions/17606/whats-the-difference-between-grid-based-and-mesh-based-methods-for-pdes/17609*

Steyvers, M. (1999*a*), 'Morphing techniques for manipulating face images', *Behavior Research Methods, Instruments, & Computers* **31**(2), 359–369.
URL: *https://doi.org/10.3758/BF03207733*

Steyvers, M. (1999*b*), 'Morphing techniques for manipulating face images', *Behavior Research Methods, Instruments, & Computers* **31**(2), 359–369.
URL: *https://doi.org/10.3758/BF03207733*

Viola, P. & Jones, M. J. (2001), 'Robust real-time object detection', *International Journal of Computer Vision* .
URL: *https://doi.org/10.1.1.23.2751*