

Project Report

Face Detection - Orientation - Recognition

Students: Julien Choukroun, Jessica Gourdon, Luc Sagnes

Referent teacher: Frédéric Précioso

Polytech Nice Sophia - Department Applied Mathematics and Modeling



Table of contents

Introduction.....	3
What is a Neural Network?.....	4
Study and selection of our algorithms.....	8
First step: Find algorithms.....	8
Second step: Choose algorithms that we will use.....	11
Third step: Choose examples and try our algorithms.....	13
4th step: Compare their efficiency, advantage/problem.....	18
5th step: Application of our algorithms in real world.....	19
Conclusion.....	21
Bibliography.....	22

Introduction

One of the key features when analyzing video content (such as movies for instance) is the detection of faces because we are naturally attracted towards other humans.

The real challenge in this context is to be able to detect faces with high precision in real-time to be able to extract facial features which will allow us to capture the orientation of the face. From this point, we will try to identify who is talking to who in tv series or movies in dialogue scenes. In parallel to this task, we will track the faces detected in the first frame and follow them through the whole scene.

We will mainly focus on deep learning techniques but we will also consider other machine learning algorithms to get the best and fastest accurate results.

The goals of this project are:

- Detecting face and extract facial features to define face orientation.
- Track faces along the video.

First, we will test our algorithms in tv series such as “Emily In Paris” on Netflix. Then the goal is to use all these techniques in different contexts.

For example locally, it would be very useful to use facial recognition in the tramway of Nice. Indeed, putting some cameras inside the tramway near the commercial could give interesting information on who is watching which commercial from where. Then by analysing this data, Nice could use it to attract different companies to use tramways to advertise. All the money granted could maybe make the tramways free for passengers.

Also, in most of the new technologies, using face recognition is more and more used by companies to develop a new aspect: giving orders by our voice.

For example, Mercedes is developing voice commands to turn on music or lower the heating. But the big issue is to know and recognize who is talking so that's why they want to put cameras in the system. This system must work even if the radio is on so the car must be capable of knowing where voices come from.

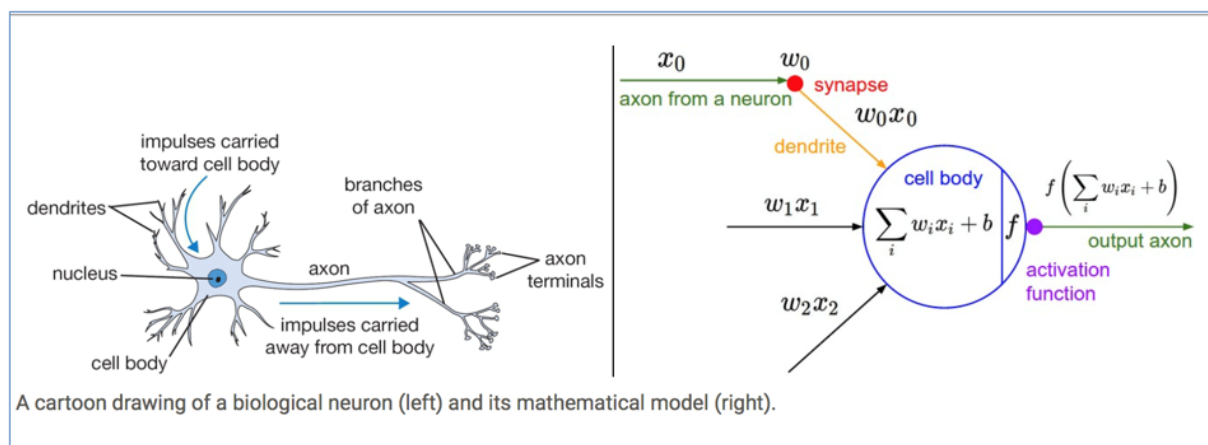
Another example is Portal, a technology that provides video chat via Facebook Messenger and WhatsApp with a camera that can automatically zoom and track people's movements. The camera will follow them to make the call more comfortable and make it more alive.

This was to show you that face recognition can be used in very different fields. Now we will present to you how we did manage our project.

What is a Neural Network?

There are many variants of neural networks. Only neural networks used for image processing will be discussed here.

The neural network called "multilayer perceptron" has the particularity of having each neuron "fully connected" to each of the neurons of the previous and the next layer. It therefore has difficulties in managing large images, due to too many connections between neurons. For this reason, as part of this project, we will use convolutional neural networks inspired by the visual cortex of animals. These limit the number of connections between a neuron and the neurons of adjacent layers and will therefore be able to process more complex images (as is the case in our project).



But to explain generally the operation of a neural network, we will take a simple example and use the "multilayer perceptron" neural network. A classic example for introducing the topic is the handwritten digits recognition.

Firstly, it is important to know that a neuron is a thing that holds a number. Specifically, a number between zero and one.

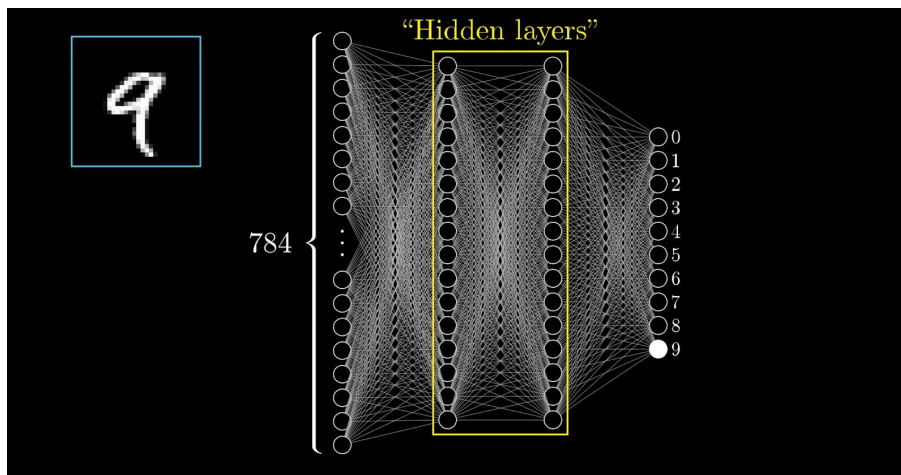
The network starts with a bunch of neurons corresponding to each of the 28 times 28 pixels of the input image which is 784 neurons in total. Each one of these holds a number that represents the grayscale value of the corresponding pixel ranging from 0 for black pixels up to 1 for white pixels.

So, all of these 784 neurons make up the first layer of our network.

The last layer has ten neurons each representing one of the digits. To determine which digits will be the result, there is a value representing how much the system thinks that a given image corresponds with a given digit.

There are also a couple layers in between called the "hidden layers".

The next layer is mostly influenced by the previous layer.

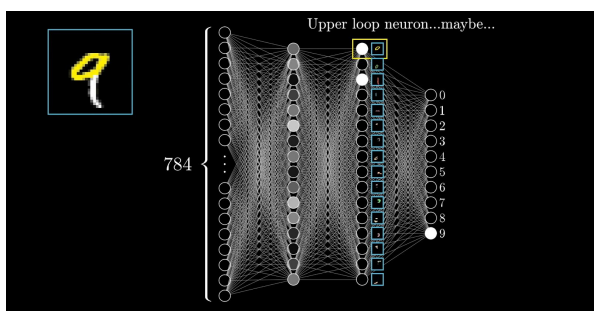
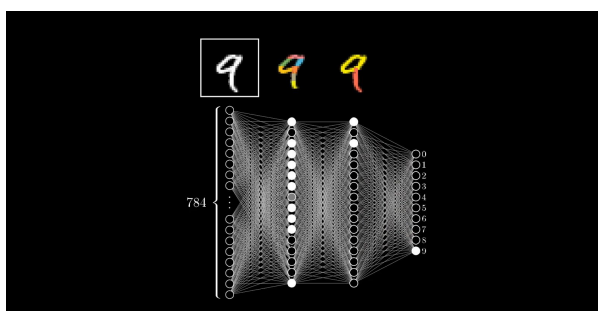


Now the network we are showing here has already been trained to recognize digits. When you recognize digits we piece together various components. For example, the nine has a loop up top and a line on the right.



That way going from the third layer to the last one, just requires learning which combination of sub components corresponds to which digits.

To recognize a sub component, it will be needed to first recognize the various little edges that make it up. In our example, in order to recognize the loop of the 9 (first sub-component), the shape will be divided into several small pieces which will be recognized by the algorithm to form a loop. This work is done in the second layer (first hidden layer).












The goal, for one particular neuron in the second layer, is to pick up on whether or not the image has an edge in this region here. To do that, a weight is assigned to each one of the connections between our neuron and the neurons from the first layer. These weights are numbers.

The value of the neuron of the previous layer as well as its weight will determine the passage to the next layer.

For this, the algorithm computes the sum of the products of each neuron with its weight, then we add a specific value to the current neuron. But the aim is to obtain a value between 0 and 1 so a common thing to do is to pump this weighted sum into some function that squishes the real number line into the range between 0 and 1.

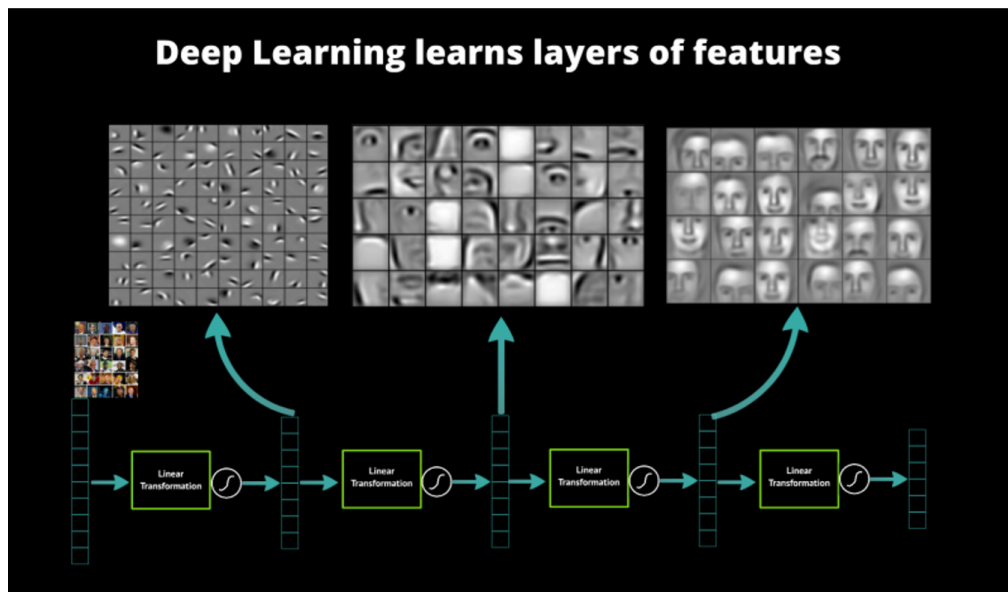
And a common function that does this is called an activation function. Here, we use the sigmoid function (also called Logistic). Basically, very negative inputs end up close to zero, very positive inputs end up close to 1 and it steadily increases around the input 0.

List of different activation function :

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

In the training test, according to the data and the result that we want to have, we may be led to want a sum greater than 10 instead of 0, for example, to activate the neuron. For this, we will add an additional value to the sum before applying the function. That additional number is called the bias.

To have a good model in machine learning, we have to train our model with data where we know the result. With this training set we will be able to fix our different weight vector so that the desired output is produced whenever a training instance is presented. This is the method of backpropagation.



An example of a Convolutional Neural Network used for our project in order to detect faces.

Study and selection of our algorithms







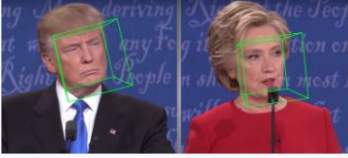
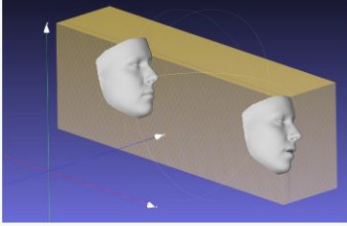
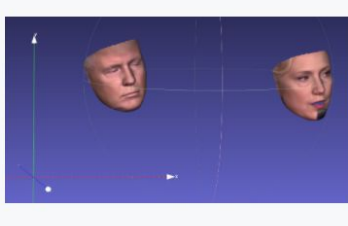
First Step: Find algorithms

3DDFA_V2

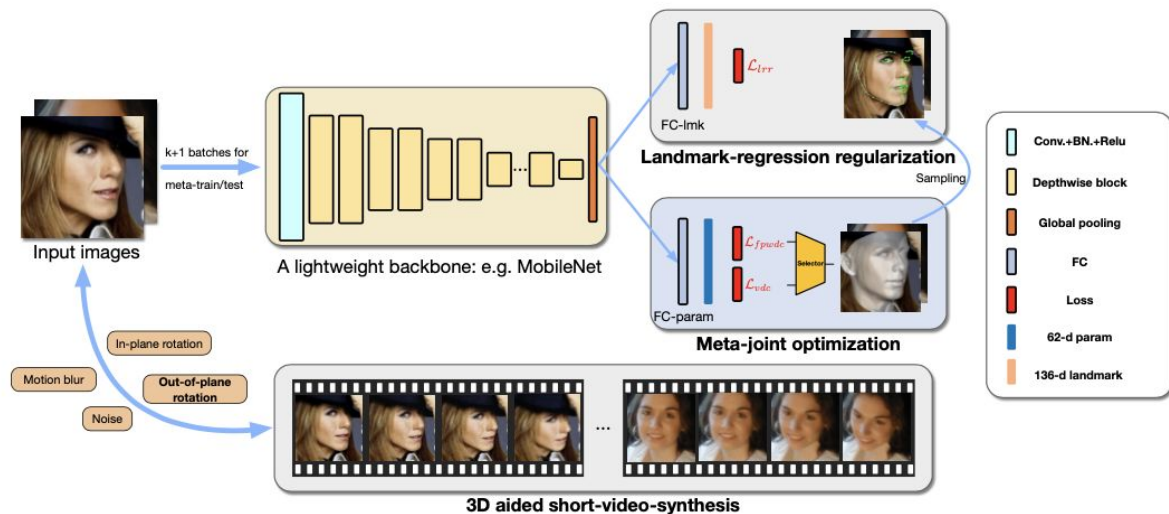
The first thing we did was to find different algorithms on Face detection. We found some algorithms that are working only on pictures and some others on videos.

The first algorithm that we found is the 3 Dimensions Dense Face Alignment Version 2 (3DDFA_V2), from the [Toward Fast, Accurate and Stable 3D Dense Face Alignment](#) publication. This one can be used on images and videos.

The implementation of tracking is simply by alignment. If the head pose is upper than 90° or the motion is too fast, the alignment may fail.

2D sparse	2D dense	3D
		
Depth	PNCC	UV texture
		
Pose	Serialization to .ply	Serialization to .obj
		

Some examples of what the algorithm can do.



Overview of the algorithm.

The MobileNet is a Convolutional Neural Network so each layer contains a small area of the face (corner of the mouth, corner of the eyes, etc.) and each neuron is a piece of this area. There is invariance by translation (independent of the spatial location in the image, the shapes can be in different places in the image), and scale (at the image level).

Then, there are 2 neural networks. One that allows to have the face with many points like a cage with layers that contain, for example the corners of the mouth, the corners of the eyes, etc. The other network allows to have only certain areas of the face, for example the mouth, the eyes, etc. The 2 networks are linked.

To train it, we give it images at the input and we give it the points that are at the output and as long as the error is too big we pass into the algorithm.

PRNet

The Position Regression Network (PRNet) algorithm is from the [Joint 3D Face Reconstruction and Dense Alignment with Position Map Regression Network](#) publication. We can use this algorithm to reconstruct 3D Face from a picture but also permits to edit some textures.

Since we will not use this algorithm, we are not going to deepen our research on it.



Some examples of what the algorithm can do.

DECA

The Detailed Expression Capture and Animation (DECA) algorithm is from the [Learning an Animatable Detailed 3D Face Model from In-The-Wild Images](#) publication.

DECA is an algorithm that reconstructs a 3D head model with detailed facial geometry from a single input image. The resulting 3D head model can be easily animated.

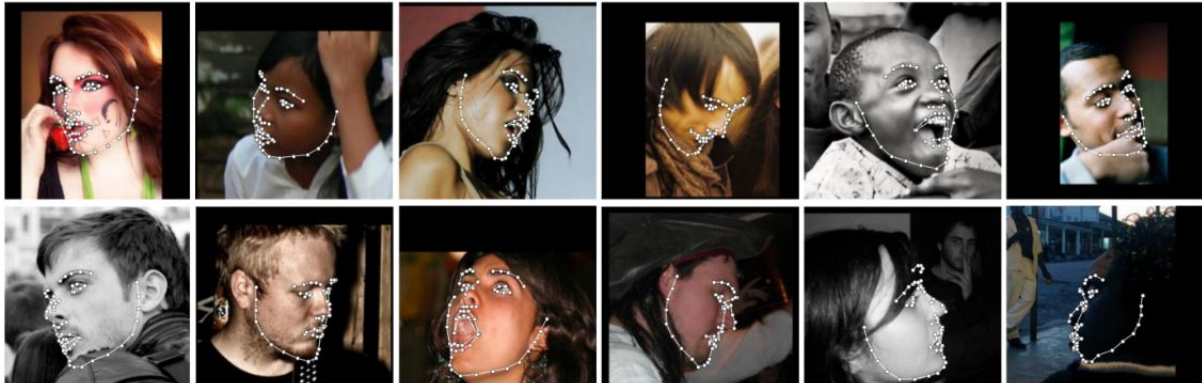
Same as PRNet, since we will not use this algorithm, we are not going to deepen our research on it.



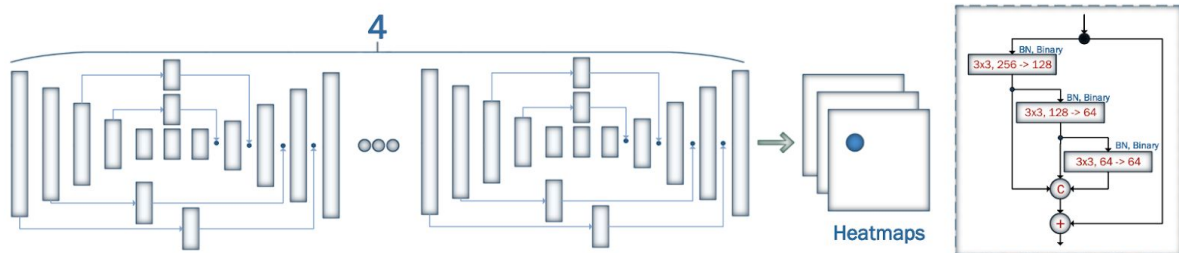
Some examples of what the algorithm can do.

FAN with S3FD and BlazeFace

The Face Alignment Network (FAN) algorithm is from the [How far are we from solving the 2D & 3D Face Alignment problem?](#) publication.



Some examples of what the algorithm can do.



Overview of the algorithm.

This algorithm uses Hourglass Networks.

Hourglass networks are a type of convolutional encoder-decoder network, meaning it uses convolutional layers to break down and reconstruct inputs. They take an input (in our case, an image), and they extract features from this input by deconstructing the image into a feature matrix.

It then takes this feature matrix and combines it with earlier layers which have a higher spatial understanding than the feature matrix (have a better sense of where objects are in the image than the feature matrix).

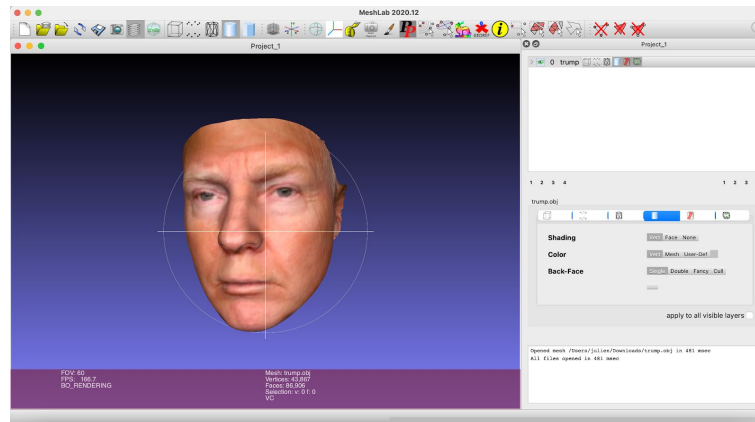
Second step: choose algorithms that we will use

Now that we chose our algorithm, we decided to try these different algorithms on pictures and images.

We noticed that the 3DDFA_V2 works pretty well for images but it becomes more complicated with videos. In fact, it is working well when there is only one person on the

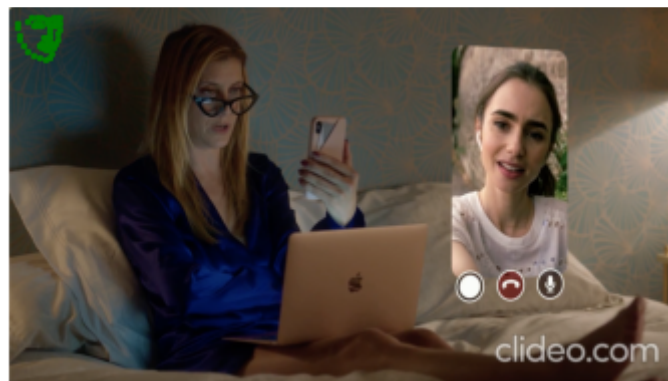
video, but it becomes wrong when there are more people. The algorithm puts faces where there is no one and forgets other faces. Moreover, when the video is too big, the algorithm does not want to execute. So we decided to work on the 3DDFA_V2 algorithm for images.

The PRNet only works on images but it is working well. The only issue is that the images in output are not in .jpg or .png format but in .obj format, so it requires a special software to open them and see the image with face recognition.



Example of an output in .obj format open with MeshLab software.

The DECA algorithm only works with images, but we have some issues. Indeed, this algorithm only detects one face although there are several faces. The second problem is that this algorithm returns the reconstruction of the face in 3D and does not return directly the points of the faces such as in the 3DDFA_V2.



Example of an output. The algorithm has detected just one face and does not come on top of the actual face.

The FAN algorithm also works with images. This algorithm uses two different methods to detect faces: the S3FD and the BlazeFace.

Finally, we decided, after this research, to work only with the 3DDFA_V2 and the FAN algorithms.

Third step: Choose example and try our algorithms

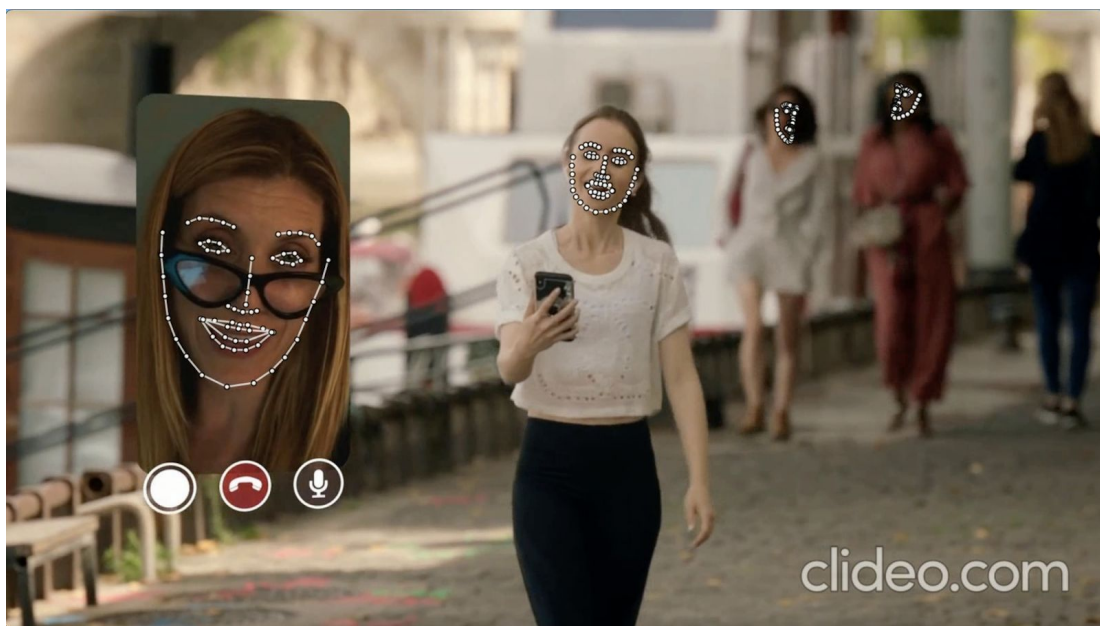
To focus on the goals of this project, we wanted to find an algorithm that works with videos so that we can do the same next in real life.

Instead of using the 3DDFA_V2 that works with videos but isn't really efficient, we decided to transform videos into frames so that we could use our algorithms that work well with images. Then, after the use of the algorithm, transform it again into videos so we get a good result.

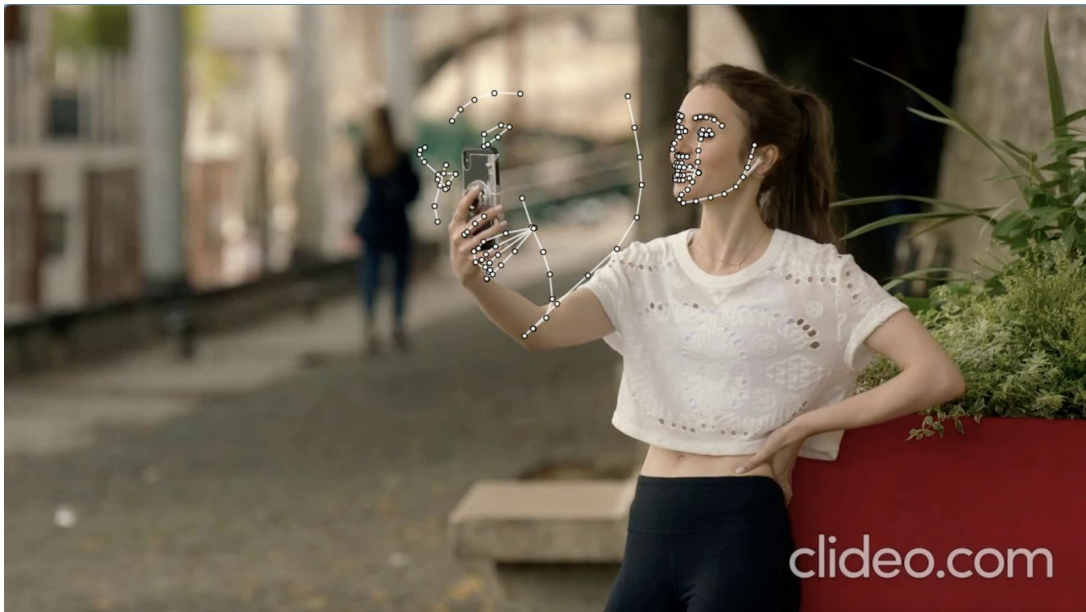
In order to do so, we first had to change the code of the algorithms so that it was working well with Google Colab.

Then, we created a method to transform a video into a multitude of images (frames) as well as his inverse operation. We noticed at this moment the RAM (the computer memory) given by Google Colab was enough for short videos (around 2min) but wasn't enough for larger one.

We choose to take a short extract of an episode of the series "Emily in Paris" on Netflix. We transformed it into frames and applied the 3DDFA_V2 algorithm and we had a pretty good result: faces of the actors were well covered during all the video. Even blurred people in the background far away were signaled by the algorithm.



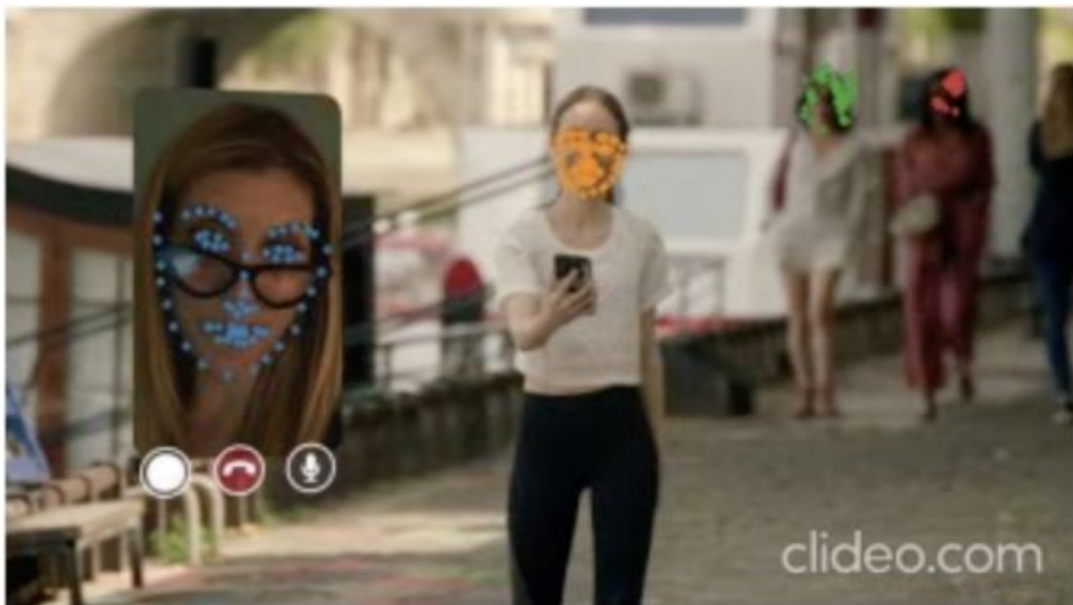
We only had for a few times some faces that appeared from nowhere due to the curve of the arms of the actress.



Overall, we found that the results of the 3DDFA_V2 were quite good, even if there were sometimes some problems.

We also use the FAN algorithm so we can after compare them.

This algorithm uses two 2 different methods to detect faces. The first, the S3FD (Single Shot Scale-invariant Face Detector) which is very accurate:

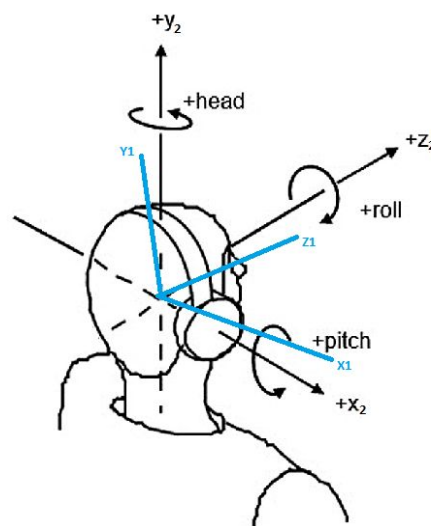


The second method, the BlazeFace which is less accurate:



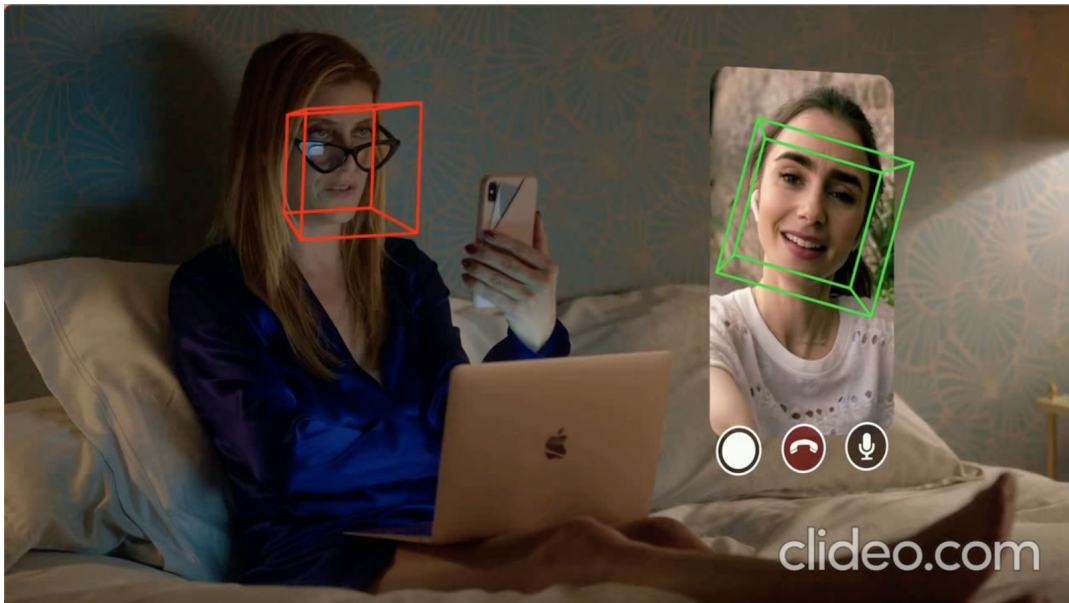
One of our goals was to detect the orientation of the faces in order to determine if the person being filmed is looking at the camera. To do this, we use the 3DDFA_V2 algorithm. Depending on the way we call the functions of this algorithm, it can return a result in "2d_sparse" as in the first example we gave. This corresponds to points detecting the face. It can also return a result in "pose". In this case, it returns a parallelepiped at the level of the faces of the characters. The back face will delimit the face of the character while the front face will be oriented according to the orientation of the face of the character. One of the functions of this algorithm returns the value of the yaw, pitch and roll corresponding to the aircraft principal axes.

In our context, the yaw is modified when you turn your head to the right or to the left, the pitch is modified when you tilt your head up or down, and the roll when you tilt your head to the left or to the right.

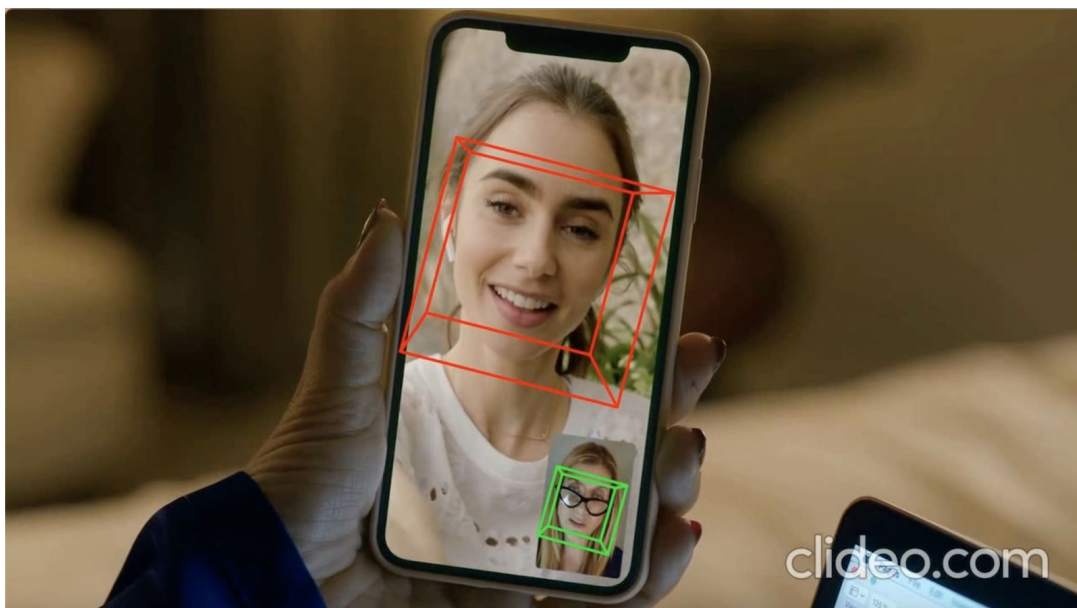


The 3 axes corresponding to the aircraft principal axes.

Knowing this, we modified the algorithm to put conditions on yaw and pitch. If these 2 data are within the interval we gave them, then the parallelepiped will be colored green, meaning that the person is looking at the camera. If at least one of the data is not included in the interval, the figure will be colored red, meaning that the character is not looking at the camera.

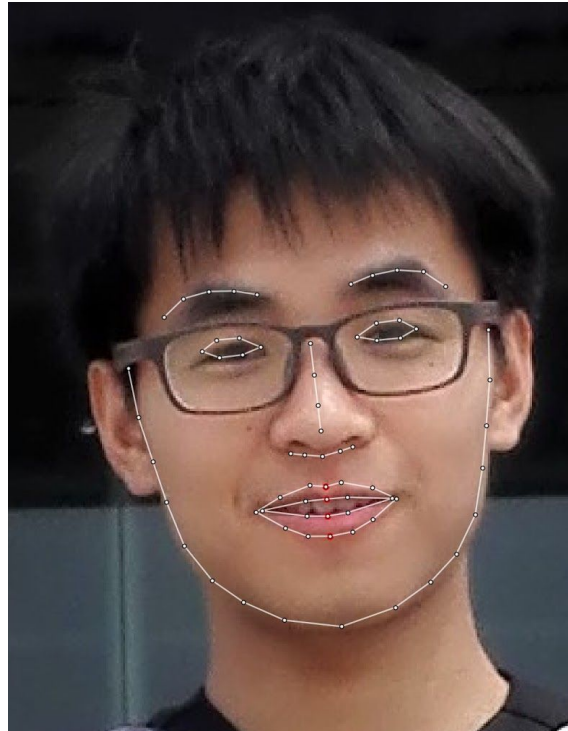


The result is not perfect. Depending on the position of the face in relation to the camera (high camera for example), the color of the rectangle may not be adapted to the real look of the character.



The other goal was to detect who is speaking. To do this, we also use the 3DDFA_V2 algorithm. For this, we used the distances between the upper and lower lips.

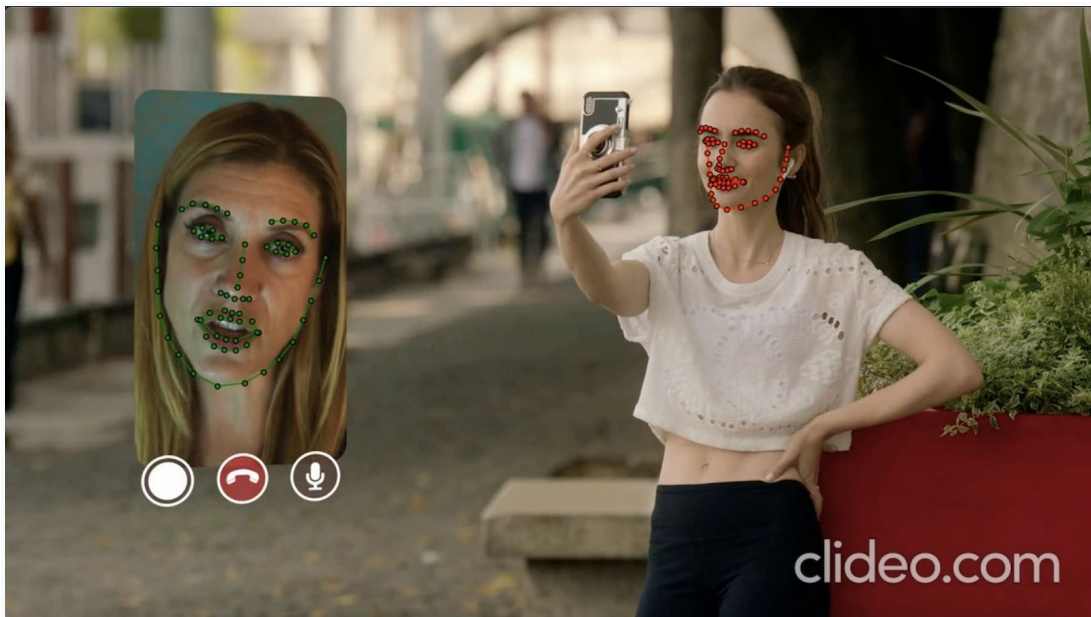
From the result of the algorithm in "2D_sparse" mode, we have recovered 4 different points: the point located on the upper part of the upper lip, the point located on the lower part of the upper lip, the point located on the lower part of the lower lip and the point located on the upper part of the lower lip. We can see these 4 points on the image :



The red points are the 4 points described above.

From this, we calculated the distance between the 2 points inside the lips. This directly represents the distance between the lips and will allow us to determine if the person opens his mouth. Then, we divided this amount by the distance between the 2 upper points of the lips to scale and avoid having differences if the person is filmed more or less close to the camera.

Then, from tests performed, we determined a threshold value from which the person opens his mouth. In this case, we consider that the person is speaking and the points delimiting his face are then colored green. If the distance does not reach this value, the points delimiting the face are colored red.



However, this method is not perfect. Indeed, if the person is smiling or laughing, the algorithm will detect that the person is talking and won't know the difference.



4th step: Compare their efficiency, advantage/ problem

The FAN algorithm is faster than the 3DDFA_V2 algorithm, with 1975 seconds for the execution of the 3DDFA_V2 and 357 seconds for the first method of detection (S3FD) of the FAN and 240 seconds for the second method of detection (BlazeFace) of the FAN. Although the BlazeFace is the quickest, it is the least accurate. The S3FD is more accurate than the 3DDFA_V2.

The advantage of the 3DDFA_V2 algorithm is the versatility of the algorithm, for example we can detect the pose or who is speaking.

5th step: Application of our algorithms in real world

To adapt our algorithm, detecting if the person is looking at the camera or not, on the videos of the Nice tramway, we slightly modified the algorithm we had realized and tested on "Emily in Paris".

Since the camera is located in height, we detect the faces looking upwards in front of them, so we had to adapt the pitch axis.

First of all we can observe one thing about the detection of faces in these videos. Since these are recent videos, recorded in the midst of a health crisis, the people are wearing masks. It has been observed that with the mask, the algorithm detects faces with much more difficulty.

Nevertheless, as we can see on the screenshots, the algorithm is working nicely on faces that the algorithm detects. There are only some mistakes when someone is not looking at us but has his face with a good angle pose with the camera (for example the men with the hat).





Conclusion

We can finally say, thanks to Neural Networks, we had the opportunity to discover Face Detection algorithms that we can apply in the real world.

If we had more time for this project, we would have the idea to go deeper with the algorithms. For example, putting a laser on the eyes to get the orientation of where people are looking at would be very interesting.

This project has also allowed us to get to the heart of the matter of Deep Learning, subject that we still have not learnt yet.

We will have next semester Mr Precioso in a course of Computer Vision and Machine Learning where we will go deeper and learn more about this subject.

Bibliography

Link of the github with the publications of our algorithms:

https://github.com/cleardusk/3DDFA_V2
<https://guojianzhu.com/assets/pdfs/3162.pdf>

<https://github.com/YadiraF/PRNet>
https://openaccess.thecvf.com/content_ECCV_2018/papers/Yao_Feng_Joint_3D_Face_ECCV_2018_paper.pdf

<https://github.com/YadiraF/DECA>
<https://arxiv.org/pdf/2012.04012.pdf>

<https://github.com/1adrianb/face-alignment>
<https://www.adrianbulat.com/downloads/FaceAlignment/FaceAlignment.pdf>

Link to all our collab:

3DDFA_V2 algorithm:
https://colab.research.google.com/drive/1pUzsmuTPOIsoNg1CvWlxxqjKBVm_a768

Face Alignment algorithm:
<https://colab.research.google.com/drive/1mtKZo1TY3d5jsM9Bilgirrc1pt9Pmm6g#scrollTo=yQK0kDk5Gexy>

Explanation of MobileNet algorithm :
<https://www.quantmetry.com/blog/mobilenet-optimisation-de-la-convolution-pour-les-reseaux-de-neurones-embarques/>