Mouth Animation

Realization

Bachelor Applied Computer Science

Yori Verbist

Academic year 2020-2021

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

Contents

1 Introduction

Why are we here? Most of us are here for a simple reason: AI is becoming more and more a part of our daily life. We already see some powerful and amazing new possibilities and realizations with AI. That's why we want to know more about it: how it works and how we can implement it ourselves That's what this thesis is about, implementing those technologies ourselves.

Here you can read how I implemented AI that autonomously generates dynamic mouth movements on top of a originally static picture, given an audio sample. This results in the illusion, a natural feel, that the person in the picture is really talking to you.

Today the day it's hard to predict what AI will fully capable of. Because AI techniques advances so rapidly, it's hard to keep track on what AI is already capable of. That's also the goal of this project, to show what AI is capable of today. There are a lot of fields in AI ranging from computer vision to natural language processing. I'm mostly interested in CV that's why I mainly chose this project.

1.1 Computer Vision

How important is vision? We use it everyday, to look where we're going, prepare food, watch videos, etc. From all our senses, it gives the most information about the state of the world and how to act on it. This is the reason why scientist are trying to give computers vision, creating the field of computer vision.

So what is computer vision exactly? It's letting a computer 'see'. Well, not seeing like we humans do but on pixel level and learning patterns in these pixels. All pictures and videos exist of these pixels. They're colored points on a screen, consisting of three values, red, green and blue also called RGB values.

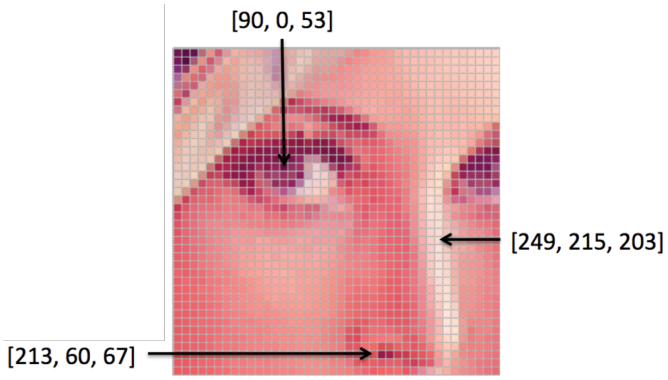


Figure 1: Example of the pixel values of an image

Now we know how computers see pictures, how do they recognize faces? This is done by putting self-learning filters on top of the picture. This is called a convolution, every filter is going to detect different parts of the face. All these different kind of filters are then put together to detect more complex patterns. Some examples can be seen in figure 2. In face recognition we use filters that detect facial features, like eyes and a nose.

By using different filters, which all search for useful patterns, we create multiple filtered images from the original image. To keep this expansion under control and to reduce it to the essence of the pattern, we use pooling techniques.

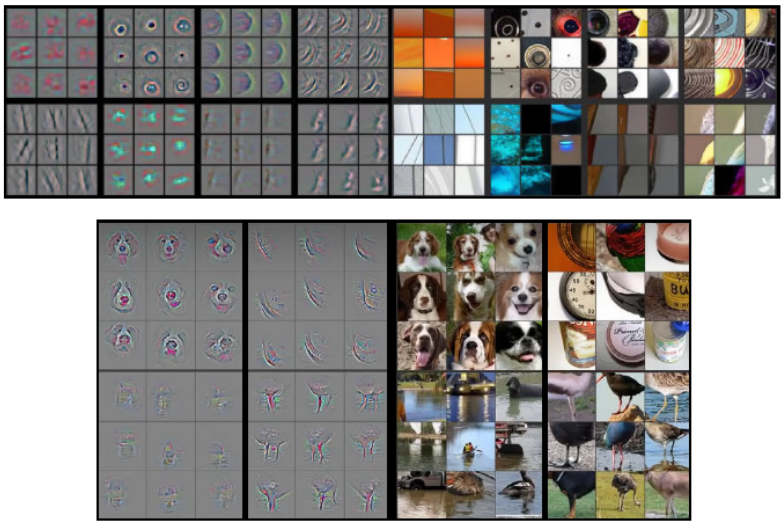


Figure 2: Examples of what the results of the different filters will look like.

1.2 Generative Adversarial Network

A generative adversarial network (GAN) is a network that learns by generating images and evaluating how realistic these generated images are. It consists of a generator and a discriminator.

The generator network is going to try to create a realistic image. To do this we need some kind of metric that tells the network what a realistic image is. In basic neural networks a mathematical metric is used, like the MSE loss. But it's hard to mathematically calculate what a realistic image looks likes. As you can see in figure 3, the error between the original and all the other images is the same, while they're clearly all different. That's why the discriminator network exists.

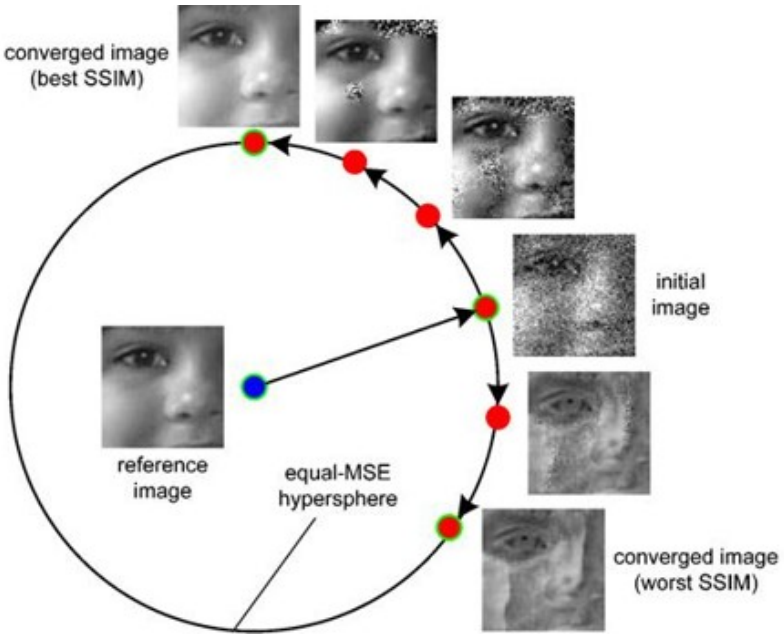


Figure 3: MSE used to calculate the distance between two images, the inner and outer images. Source:[1]

The generator does the reverse of the convolutions that are done for face detection, called deconvolutions. Instead of getting only the essence of the picture, they start from the essence and

up-sample to a normal image.
The discriminator network is a neural network trained for classification. When given an image, it will tell if that images looks real enough compared to the images it was trained on.

The generator and the discriminator have a competition between themselves. Where the goal of the generator is to create realistic images and fool the discriminator, the goal of the discriminator is to know when a unrealistic image is created by the generator. Because of this, they always get better and better to out-preform the other.

2 Wav2Lip

I started searching for papers that did something around mouth/lip generation. This is when I found this amazing work of a research group[2]. It's the same group that worked on LipGAN[3].

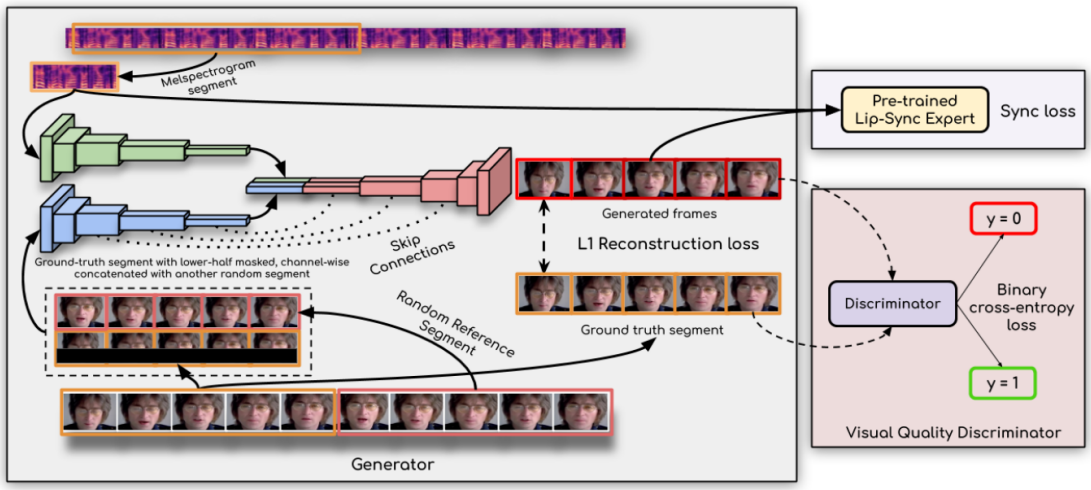


Figure 4: An overview of the model. The important components are discussed below.

2.1 Pre-trained Lip-Sync Expert

This discriminator is pre-trained and should not be fine-tuned on the generated images. The face reconstruction loss is calculated on the whole face, the lip region is only 4% of the total reconstruction loss.

So a lot of surrounding image reconstruction is first optimized before the network optimizes the lip region. That's why it's important to have this additional discriminator. Also by using a pre-trained discriminator there are no noisy generated images from the generator. Because the discriminator will focus more on the visual artifacts instead of the audio-lip correspondence. It's also important to train the discriminator on more than 1 video frame.

Model	Fine-tuned?	Off-sync Acc.	LSE-D	LSE-C
$Tv = 1$ [3]	✓	55.6%	10.33	3.19
Ours $Tv = 1$	×	79.3%	8.583	4.845
Ours $Tv = 3$	✓	72.3%	10.14	3.214
Ours $Tv = 3$	×	87.4%	7.230	6.533
Ours $Tv = 5$	✓	73.6%	9.953	3.508
Ours $Tv = 5$	×	**91.6%**	**6.386**	**7.789**

Table 1: Here you can see the differences between the amount of video frames (Tv) and if it's further fine-tuned during training or not.

In table 1 you can see that a larger temporal window allows for better lip-sync discrimination. On the other hand, training the lip-sync discriminator on the generated faces deteriorates its ability to detect off-sync audio-lip pairs. Consequently, training a lip-sync generator using such a discriminator leads to poorly lip-synced videos.

2.2 Generator

The generator contains three blocks: Identity Encoder, Speech Encoder, Face Decoder.
The Identity Encoder (blue blocks) is a stack of residual convolutional layers that encode a random reference frame concatenated with a pose-prior (target-face with lower half masked).

The Speech Encoder (green blocks) is also a stack of 2D-convolutions to encode the speech segment. These are then concatenated together.

The decoder (red blocks) is also a stack of 2D-convolutional layers, along with transpose convolutions for up-sampling. The generator is trained to minimize the L1 reconstruction loss between the generated frames and ground-truth frames.

2.3 Penalizing inaccurate lip generation

Since the discriminator is trained on 5 contiguous frames at a time, we would also need the generator to generate 5 frames. This is done by sampling a random contiguous window for the reference frames, to ensure as much temporal consistency of pose, etc. across the window.

The generator processes each frame independently, so they stack the time-stamps along the batch dimension while feeding the reference frames. While feeding the generated frames to the expert discriminator, the time-steps are concatenated along the channel-dimension as was also done during the training of the discriminator.

The resulting input shape to the expert discriminator is (N, $H/2$, W, $3 \cdot Tv$), where only the lower half of the generated face is used for discrimination.

2.4 Visual quality generator

Since they use such a strong lip-sync discriminator it forces the generator to produce accurate lip shapes. Sometimes this results in the morphed regions to be slightly blurry. To migrate this minor loss in quality, they train a simple visual quality generator in a GAN setup along with the generator. This discriminator does not make any checks on lip-sync and only penalizes unrealistic face generations. The discriminator consists of a stack of convolutional blocks. Each block consists of a convolutional layer followed by a Leaky ReLu activation.

2.5 Summary

The main goal of this library is to generate mouth movement for videos, to dub them. Because of this it also works better on videos than on pictures. This is also because they feed the network five continuous frames, instead of one frame when predicting the mouth movements. When using a picture it also works better when the persons teeth aren't visible. This is a known problem with mouth movement generation, thus it's outside of the scope of this thesis.

It's also trained English data [4]. My goal is to see if there are any differences to when it's fine-tuned on Dutch data.

Sometimes the face-detection doesn't work 100%. These times you can clearly see that not the whole chin region is included in the generated frames. This results in a moving mouth, but the chin is standing still, which is not realistic.

At the moment when you feed the model a picture or video with multiple people in it, it generates the mouth movement of the first face it detects. My goal is when you feed it a picture with multiple people that you can choose for which person you want to predict the mouth movements.

3 Dutch Data

Since the original model is trained on English data, I'll be fine-tuning the model on self collected Dutch data.

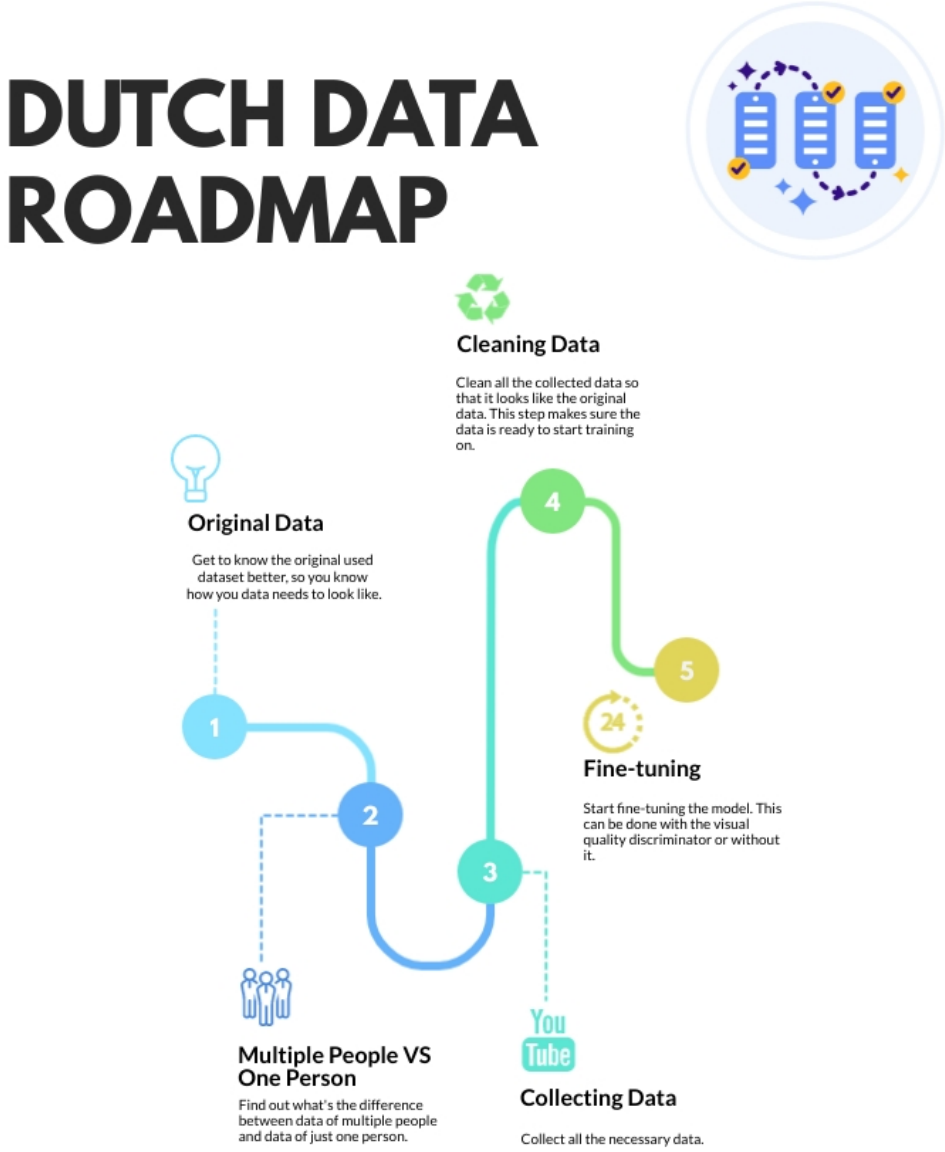


Figure 5: Roadmap of how I got all the data ready to fine-tune the model.

3.1 Original Data

The original data on which the model is trained is a dataset from BBC television. It's the Lip Reading Sentences 2 (LRS2) dataset [4]. The videos in there are heavily preprocessed. They are sync-corrected, and every video is only max one sentence long resulting in a max length of ten seconds. The person that's talking is also always visible and there are a lot of different people in all the clips. They made sure to split the training and validation set so that the same person doesn't show up in both (so you won't overfit on one specific person). This means that these little video clips are cut from longer video's, done by people. The dataset consists of approximately 50GB of data. This is a lot of data to clean yourself.

3.2 Multiple people VS one person

The original data consists of a lot of different people. When I was searching for Dutch data, I concluded it's hard to find a lot of videos of different people. When you have multiple people you also need to be sure that every person approximately shows up the same amount of times. If one person shows up a lot more than all the other people, the model will overfit on that one specific person. So I decided to just focus on one person and see what the results will be when the model is fine-tuned on one specific person.

This is when a decided to use the videos of VPRO Zondag met Lubach, it's a Dutch talkshow comparable to Last Week Tonight with John Oliver.

3.3 Collecting data

Once I knew which video clips I was going to use, I downloaded 80 videos, since these came from a YouTube channel, it was easy to just put them in a playlist and download the whole playlist.

3.4 Cleaning data

When I had enough videos I started by cutting the videos in smaller video clips, because the original videos were 5 to 20 minutes long. I wrote a script that looped over all the videos and cut them in smaller clips. To do this I used the Pydub and MoviePy libraries. Pydub is used to get the timestamps of all the silences in a video. While MoviePy is used to cut a video in smaller clips. Pydub returns a list with all the timestamps of when silent or non-silent segment starts and when it ends.

With these timestamps I knew when to cut the video in smaller clips. Since sometimes the non-silent parts are still longer than ten seconds I wrote an extra loop. When there was a non-silent part longer than ten seconds it cut that part in smaller pieces of five seconds each. This is done because feeding the model clips longer than ten seconds will use to much memory. At the end of all this I around 5800 video clips, which should be plenty to fine-tune a model.

When the videos were cut in smaller clips I cropped each clip so only the right half of it remained. This is done because most of the time Lubach is on the right side of the video. This cropping results in smaller clips, which will later result in faster prepossessing of the data.

The data was almost cleaned enough to start training, but there is still on problem in some clips. There were some clips where the wrong person was talking or where someone was talking and nobody was in the frame. This results in problems during training because the model doesn't have a mouth to predict because there is no face in the frame or the wrong face is in the frame.

To solve this problem I wrote a script that looped over every video and checked if Lubach was in that frame. If he wasn't the video clip just got deleted. This resulted in leaving me left with around 4700 video clips, which still should be plenty to train on.

3.5 Fine-tuning

Now the data is collected and cleaned it's time to start fine-tuning the original model. There are two ways to train the model: with or without the visual quality discriminator. The only job of this discriminator is to check the quality of the generated outputs of the generator. Training without it results in faster training with the downside the the generated mouth region is noticeable blurry.

First I trained without the extra discriminator to see if the lesser quality would outweigh the longer training times. The resulting model generated really blurry mouth regions. So I decided to start fine-tuning with the extra discriminator. Since I still had plenty of other things to do, the extra training time didn't matter that much.

4 Face Detection

Sometimes the face detection to generate the mouth animation on doesn't work perfectly. I'm going to look deeper at why this sometimes occurs.

4.1 Face Recognition

The library uses a modified version of the face recognition library [5]. Originally it detects face landmarks and not a box in which the face is located. So the authors of the Wav2Lip library adjusted it so that it returns the coordinates in which the face is located.

4.2 Predicting mouth movements

When you upload a picture, the script does face detection to get the coordinates in which the face is located. This is done because it's useless to feed the whole image to the model, when only the face is needed. Feeding the whole picture would require a lot more video memory.

For the next example I have used figure 6 as the input image. I've used this picture because its a picture of the Lubach talkshow I've used in the Dutch data part.

Figure 6: The image that's used as input.

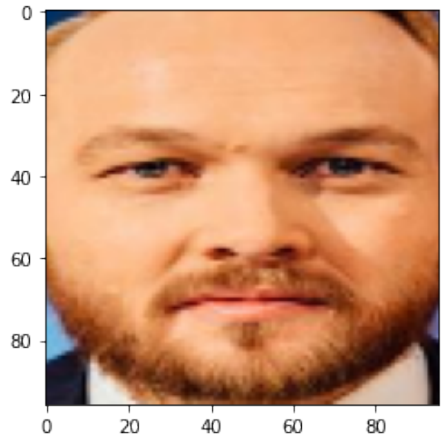
As said in Chapter 2.2, the generator is fed an image with the face and one with the lower half masked. This can be seen in figure 7. Here you can already see how big the difference is between the input image and the detected face that's used for the model.

As you can see, the face region is pretty good. When I did this on the image that I used when it gave a bad result, it also gave me the whole face. After this I ran the script again on that image and it didn't have the problem anymore.

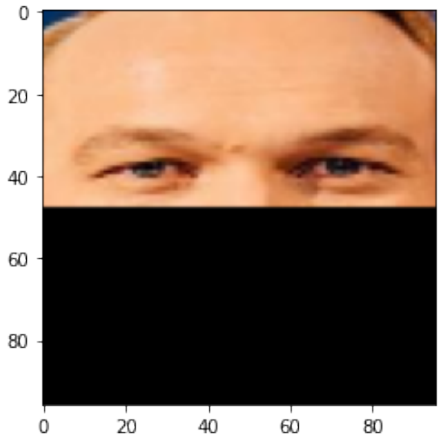
This work was not for nothing, I got a better understanding of how the face detection is done. It gave me also a better understanding how the script itself works.

What the script really does is, it predicts the face region as seen in figure 7a. After this, it makes a copy of this image and masks the bottom half (as seen in figure 7b). The whole face and the masked are both send to the model, it uses the masked face to predict the lower half on the face. The whole face is used to know how chin and mouth looks like. This is also the reason that the model works better with videos.

When you use a video, instead of just these two pictures being send to the model, five continuous frames are send. Because of this the model has a better understanding what the face looks like in multiple face expressions (mouth open and closed). While when you use a picture it only



(a) The detected face from the input image.



(b) The detected face with the lower half masked.

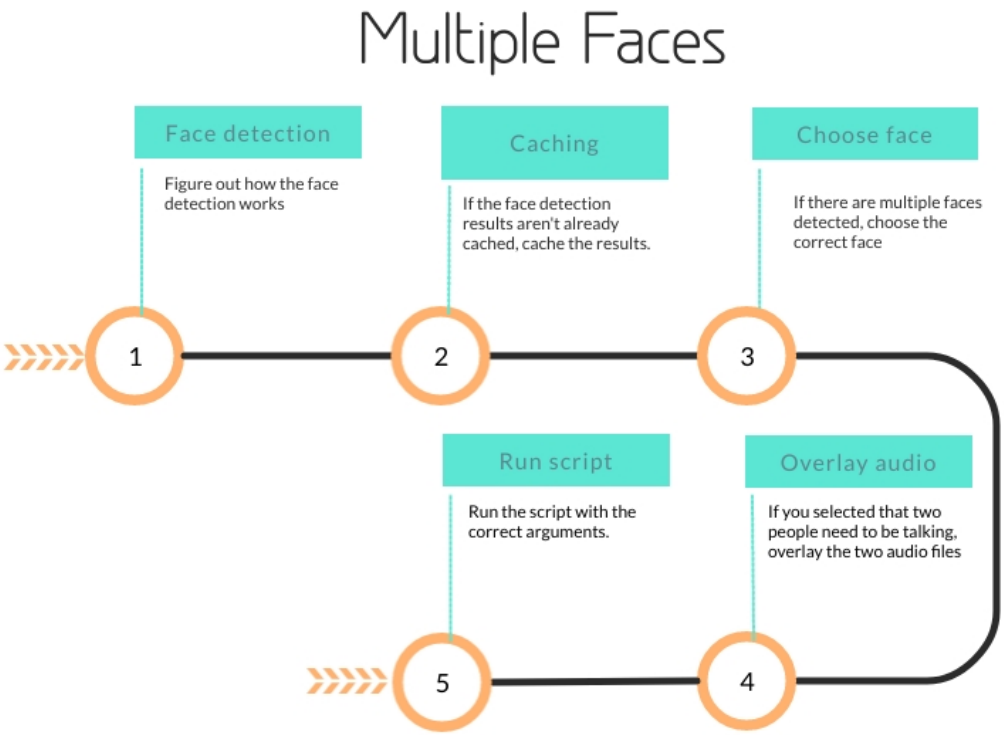
Figure 7: Example of what is fed to the generator when predicting the mouth movements.

has one frame of reference instead of five.

The mel spectrogram (a visual representation of audio) of the audio file is also send to the model, because of this the model knows how many mouth positions needs to be generated. When the model has predicted all the mouth positions they are all pasted on top of the original detected face. This results in a video of the illusion that the person is talking.

5 Multiple Faces

With the original model it's only possible to generate mouth movements when there is only one person in the picture or video. When there are multiple people in the frame(s), the model just generates the mouth movements of the first person that is detected. So, this is where I implemented changes so it's possible to choose for which person you want to generate the mouth movements when there are multiple people in the frame(s).



5.1 Face detection

When I looked deeper in how the face detection works I found out that it always returns the first face it detects. So I changed it that it returns all the detected faces, so you can later select the correct face.

5.2 Web App

To make this user friendly I made a web app in Flask. Without a web app it's hard to select the correct face.

5.2.1 Multiple people

When you upload a picture and audio file, it firsts checks how many people are in the picture. If there is only one person in it, it just runs the script on that one person. When it detects multiple people, it gives you a new page where you can select the correct face.

5.2.2 Caching

The face detection in the previous section is done outside of the script. Because of this it always has to do it twice, one outside the script and once inside. Also when you upload a long video (longer than 20 seconds) it takes a couple of minutes to detect those faces. It takes so long because it needs to detect the face in every frame.

Because of these two problems I implemented a caching of the face detection. The web app first checks if the picture or video is already cached, if this is the case it just loads the data into it (which is instant). When it doesn't find the cache of that picture/video, it does face detection on it once and writes these results to a pandas data frame and then to a CSV file.

When the script is ran after the face detection, the path of the cached results is also given to the script. Because of this it also skips the face detection part. The face detection is the part that takes the longest of the script.

5.2.3 Multiple people talking

It's also possible to first let the model generate the mouth movements of one person and then of another person. If you then upload the correct audio for each person, you can make it look like these two people are talking to each other.

When I implemented this first I had the problem that the audio of the video (the generated mouth animation video) gets deleted and only the input audio is put on top of the new generated mouth animated video. This happens because the model generates the mouth positions for every frame. When this is done, it pastes all the frames back together and puts the input audio on top. So the audio of the input video gets lost.

I added a checkbox so the model knows when it needs to overlay the input audio and the audio of the input video. This overlay is done with Pydub.

References

[1] Bram Heyns. Big data - cutting edge. 2020.

[2] K R Prajwal, Rudrabha Mukhopadhyay, Vinay P. Namboodiri, and C.V. Jawahar. A lip sync expert is all you need for speech to lip generation in the wild. In *Proceedings of the 28th ACM International Conference on Multimedia*, MM '20, page 484–492, New York, NY, USA, 2020. Association for Computing Machinery.

[3] Prajwal K R, Rudrabha Mukhopadhyay, Jerin Philip, Abhishek Jha, Vinay Namboodiri, and C V Jawahar. Towards automatic face-to-face translation. *Proceedings of the 27th ACM International Conference on Multimedia*, Oct 2019.

[4] T. Afouras, J. S. Chung, A. Senior, O. Vinyals, and A. Zisserman. Deep audio-visual speech recognition. In *arXiv:1809.02108*, 2018.

[5] Adrian Bulat and Georgios Tzimiropoulos. How far are we from solving the 2d & 3d face alignment problem? (and a dataset of 230,000 3d facial landmarks). In *International Conference on Computer Vision*, 2017.