# Advance Machine Learning Project : Fader Networks : Manipulating Images by Sliding Attributes

**ABDELHAMID Hocine** HOCINE.ABDELHAMID@ETU.SORBONNE-UNIVERSITE.FR
*Master Ingénierie des Systèmes Intelligents*
*Sorbonne Université*
*Paris, France*

**BOUHAFS Zakaria** ZAKARIA.BOUHAFS@ETU.SORBONNE-UNIVERSITE.FR
*Master Ingénierie des Systèmes Intelligents*
*Sorbonne Université*
*Paris, France*

**OZKUL Sevan** SEVAN.OZKUL@ETU.SORBONNE-UNIVERSITE.FR
*Master Ingénierie des Systèmes Intelligents*
*Sorbonne Université*
*Paris, France*

**BEKIROGLU Begum** BEGUM.BEKIROGLU@ETU.SORBONNE-UNIVERSITE.FR
*Master Ingénierie des Systèmes Intelligents*
*Sorbonne Université*
*Paris, France*

**Editor:** Advanced Machine Learning (2022-2023)

## Abstract

In this project we studied the article "Fader Networks : Manipulating Images by Sliding Attributes" (1). We implemented the encoder-decoder architecture presented in the article and tried to reproduce the same results. The overall aim of this algorithm is to gradually change an attribute of the person in an input image such as his facial expression or age and obtain a realistic output. To do so, we train the algorithm to extract information about certain attributes and to change them by using continuous attribute values in order to decide the degree of change that we want to apply. Although using the same algorithm as the article, we have encountered some problems related to the machines that we used, such as limited memory and GPU. To overcome the memory challenge, we have created a class "data loader" that allowed us to load images of the scale of the "batch size" to the RAM in each iteration and discharge the previous images from the memory. On the other hand, to solve the GPU problem we saved the weights obtained in each epoch on Google drive in case Google Colab stops the learning process and we reduced the number of epoch. At the end even though we weren't able to reproduce the exact number of epochs used in the article we were able to obtain natural images with different attributes than the input images.

**Keywords:** Machine Learning, Fader Networks, attribute change, encoder-decoder

## 1. Introduction

In order to reproduce the algorithm, we started by studying and understanding the methods used in the article. The algorithm that we consider was able to recreate a new image of a face with a different attribute from the input image. For example if the input image contains a face with youthful female traits, in the output image they have managed to obtain a realistic picture of someone who has the opposite gender or someone older. The user can also decide the degree of incorporation of each attribute, it means that we do not only obtain two pictures with opposite features, at the and we have multiple images with gradually changing attributes. The main challenge with the learning process is that this is an unsupervised learning and we do not have a reference picture to use for the learning process. For instance, we do not have a male version of a female portrait to learn from. Consequently, the algorithm has to extract the signs of an attributes only by using a labeled training set. To do so, the article proposes an encoder-decoder architecture combined with a classifier. While the classifier learn to predict a given attribute y, the encoder maps the image to a latent representation z and the decoder reconstruct the image with a new attribute. The aim is to create an output image that fools the classifier. With this algorithm they managed to obtain realistic outputs with changing attributes.


FIGURE 1 – The results obteined by the article (1) : From the original image (on the left) they have reconstructed the same face with different attribute values. Each attribute is considered as continuous variable, this allows the algorithm to change it gradually

In the remainder of the paper, the detailed architecture of the algorithm and the data that we used for this project will be presented, our implementation of the code with the challenges that we had and the solutions that we brought will be discussed and the experimental results that we obtain will be shown.

## 2. Presentation of the Algorithm

**Encoder-decoder architecture** As presented above, the algorithm consists of an encoder-decoder architecture with domain-adversarial training on the latent space. Domain-adversarial training is a method used for feature learning with unlabeled data (2) . The encoder is a convolutional neural network z = $E_{\theta_{enc}}(x)$ which maps an input image $x$ that has an attribute $y$. On the other hand the decoder is a deconvolutional network $D_{\theta_{enc}}(z, y')$ that recreates the same image $x$ with the attribute $y'$. Unlike other methods that aim to accomplish the same task, Fader networks use adversarial training on the latent space in order to obtain the same latent representations $E(x)$ and

$E(x')$ independently from the attribute $y$. The article also proposes an additional neural network called discriminator that predicts an attribute $y$ for a given latent representation $z$.

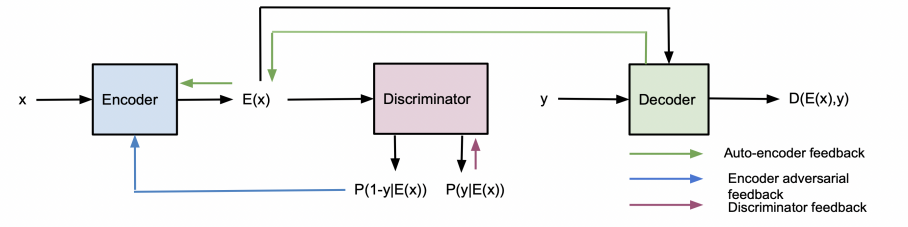


FIGURE 2 – The main architecture proposed by the article

For our project we have implemented the algorithm without changing any parameters. We used the same input images of size 256x256 and we applied 7 layers of the encoder to obtain $E(x)$. We use this output as an input for the discriminator network to obtain a prediction of $y$. The decoder network receives $E(x)$ and $y$ as input to produce the new version of the input image. The encoder and decoder both consist of 7 layers of Convolution-BatchNormalisation-ReLU. Each convolution uses a kernel of size 4x4 with stride of 2 and padding of 1. For the encoder we use the leaky-ReLU with a slope of 0.2 and for decoder we use a simple ReLU as an activation function. We added a dropout layer with a rate of 0.3 in the discriminator as advised in (1). After implementing these layers we obtain the following model :

| Encoder | | |
|---|---|---|
| Layer | Output Shape | Param # |
| conv2D | (None, 128, 128, 16) | 784 |
| conv2D | (None, 64, 64, 32) | 8224 |
| batch_normalization | (None, 64, 64, 32) | 128 |
| conv2D | (None, 32, 32, 64) | 32832 |
| batch_normalization | (None, 32, 32, 64) | 256 |
| conv2D | (None, 16, 16, 128) | 131200 |
| batch_normalization | (None, 16, 16, 128) | 512 |
| conv2D | (None, 8, 8, 256) | 524544 |
| batch_normalization | (None, 8, 8, 256) | 1024 |
| conv2D | (None, 4, 4, 512) | 2097664 |
| batch_normalization | (None, 4, 4, 512) | 2048 |
| conv2D | (None, 2, 2, 512) | 4194816 |
| batch_normalization | (None, 2, 2, 512) | 2048 |

| Discriminator | | |
|---|---|---|
| Layer | Output Shape | Param # |
| conv2D | (None, 1, 1, 512) | 4194816 |
| batch_normalization | (None, 1, 1, 512) | 2048 |
| dropout | (None, 1, 1, 512) | 0 |
| dense | (None, 1, 1, 512) | 262656 |
| dense | (None, 1, 1, 2) | 1026 |
| reshape | (None, 2) | 0 |

| Decoder | | |
|---|---|---|
| Layer | Output Shape | Param # |
| conv2D transpose | (None, 4, 4, 512) | 4211200 |
| conv2D transpose | (None, 8, 8, 512) | 4194816 |
| batch_normalization | (None, 8, 8, 512) | 2048 |
| conv2D transpose | (None, 16, 16, 256) | 2097408 |
| batch_normalization | (None, 16, 16, 256) | 1024 |
| conv2D transpose | (None, 32, 32, 128) | 524416 |
| batch_normalization | (None, 32, 32, 128) | 512 |
| conv2D transpose | (None, 64, 64, 64) | 131136 |
| batch_normalization | (None, 64, 64, 64) | 256 |
| conv2D transpose | (None, 128, 128, 32) | 32800 |
| batch_normalization | (None, 128, 128, 32) | 128 |
| conv2D transpose | (None, 256, 256, 3) | 1539 |
| batch_normalization | (None, 256, 256, 3) | 12 |

FIGURE 3 – Model summary obtained after implementation of the algorithm : $C_{16} - C_{32} - C_{64} - C_{128} - C_{256} - C_{512} - C_{512}$ for the encoder and $C_{512+2n} - C_{512+2n} - C_{256+2n} - C_{128+2n} - C_{64+2n} - C_{32+2n} - C_{16+2n}$. For the discriminator we have a convolution followed by fully connected network of two layers.

## 3. Datas

We have trained an tested the algorithm with the same data set used by the article. We used the CelebA Dataset which contains 202,599 number of face images. We decided to train our model on images of size [256x256] so we had to preprocess the data (images and their annotations) in a first place. Then, during the training we normalize the images between -1 and 1.

The main challenge that we had while using this data set was that we were not able to load all of the images to the RAM at the same time without overloading our computer's memory. To load every images we would need 40GB of RAM. To solve this problem, we created a function that loads a relatively small number of images and discharges them once they have been used for the learning process. The number of picture to load is controlled by a variable called "batch size" and and it can be changed by the user. Loading the images gradually to the RAM is theoretically slower than to have all of the images in the RAM but for our project this speed difference was not a problem.

## 4. Experimental evaluation

### 4.1 Experience description

We have implemented the algorithm as it is presented in the article. We used TensorFlow library to create our network and trained our algorithm on the platform Google Colab. This platform allows us to train our machine learning model directly in the cloud. We used this platform because we didn't have access to our University's GPU. The main challenge of using Google Colab is that we have approximately 6 hours to run our code. After this duration, the execution stops and we loose all of the unsaved results that we obtained during this period. We were able to realize approximately 50 epoch in 6 hours. Given the fact that in the article they have realized 1000 epochs to train the algorithm, we had to save our training results that we obtained during our execution periods and restart the loading process from where we have left. At first, we tried to save all of our model by using the function "keras.model.save" but this function gave us a binary file without the functions that we have implemented. Therefore we decided to save the weights that we obtained in each epoch instead of saving all of the model each time. At the and we have realized ,,,, epoch to train our model.

### 4.2 Results

Although the difference of the number of epoch, we have manged to obtain some realistic results with different attributes. For the eyeglass attribute, the learning is done with 160 epoch and for the male attribute we did 120 epoch. We obtained the following loss functions :

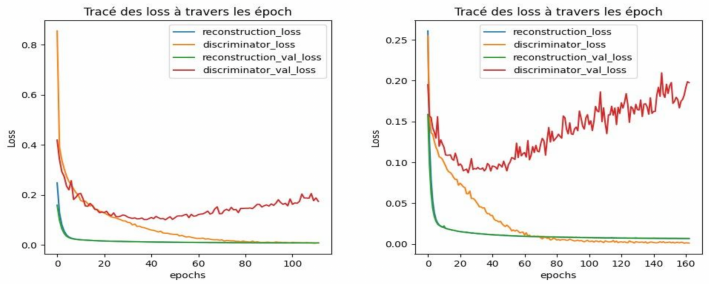


FIGURE 4 – The evolution of loss functions with the epoch number. Graphic on the left represent the learning for the male attribute and on the right it is the eyeglass attribute. We notice that most of the loss functions have a decreasing tendency when we increase the epoch number

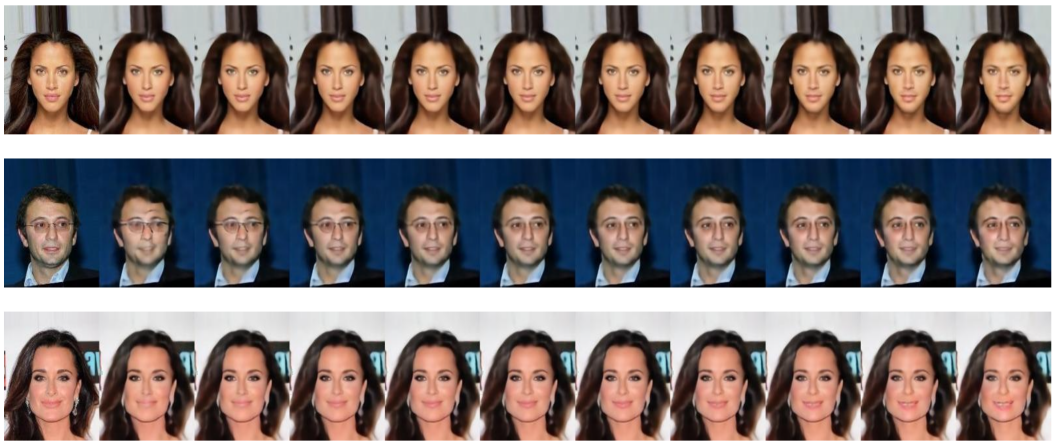

FIGURE 5 – The results obtained for respectively : male, eyeglass and mouth opening attributes

While we were developing the algorithm, we were using the terminal to execute the code. It was via the terminal that we were setting the parameters like $\alpha$ that indicates how much the attribute should change. When we finished the algorithm, we decided to make our code more user friendly by creating a GUI that make the application easier. This graphic interface enables to display our results in real time and we can also show the historic and the loss graphs by using this interface. The GUI as well as the rest of the code can be found in our Github page. (3)

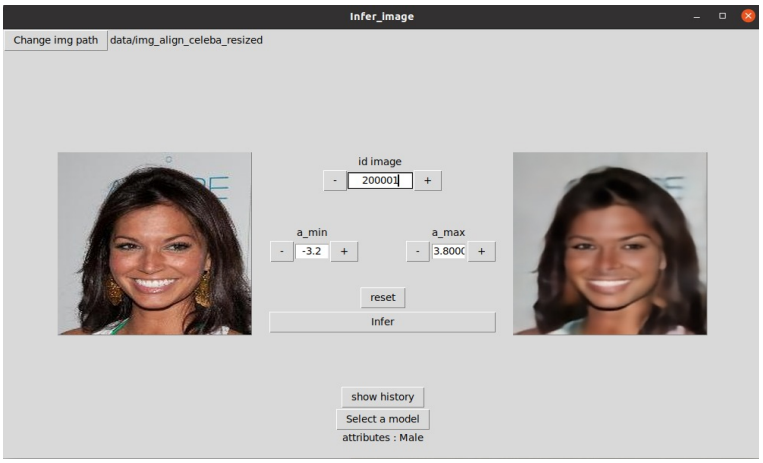


FIGURE 6 – The interface that we created to facilitate the use of the algorithm.

### 4.3 Discussion

We notice that for some attributes we get some very natural results. However sometimes the result obtained is too blurry or deformed. Since we didn't have the same number of epochs as in he article the learning differences are understandable. Yet, for some attributes, the output image that we obtain shows that the algorithm was not capable to learn. We think that this problem is caused by the fact that in our database we do not have a lot of examples of this attribute. For instance, in the data set only 11000 pictures represents people with glasses out of 160000. In the future developments of our code, we are planning to find a way to integrate better this low represented attributes.

FIGURE 7 – The transformation of an attribute with small number of representation in data set. For this example the aim is to put glasses on a face.

## 5. Conclusion

In this project, we have implemented the algorithm that has been explained in details in the article "Fader Networks". This project thought us how to approach to a new method and how to analyze an article in order to reproduce the same results. We have also noticed that even though we apply exactly the same algorithm as in the article with the same data set, we still have to find solutions to the problem related to the performance to our computers. We have also wanted to take the algorithm to a step further and create an interactive interface to make the transformation of the image easier for the user. This project has also allowed us to learn how to use GitHub effectively. It was a great opportunity for us to equally participate to the development of this project via the platform GitHub which is commonly used by the coder community. Using a different data set or trying to change different attributes can be a way to take this project a step further for the future applications.

## Références


[1] N. U. A. B. L. D. M. R. Guillaume Lample, Neil Zeghidour, "Fader networks : Manipulationg images by sliding attributes," 2017. `https://arxiv.org/abs/1706.00409`.

[2] H. A. P. G. H. L. F. L. M. M. V. L. Yaroslav Ganin, Evgeniya Ustinova, "Domain-adversarial training of neural networks," *Journal of Machine Learning Research*, 2016. `https://arxiv.org/pdf/1505.07818.pdf`.

[3] "Projet fadernetwork - implementation of the fader network." `https://github.com/Huss94/FaderNetwork_MLA.git`.