Fader Networks : Manipulating Images by Sliding Attributes

FEDDAK Lynda FEDDAKLYNDA00@GMAIL.FR
Master Ingénierie des Systèmes Intelligents
Sorbonne Université
Paris, France

NAIT ACHOUR Younes YOUNES.N.A.2000@GMAIL.COM
Master Ingénierie des Systèmes Intelligents
Sorbonne Université
Paris, France

HAMDI Massyl Yanis HMASSYLYANIS@GMAIL.COM
Master Ingénierie des Systèmes Intelligents
Sorbonne Université
Paris, France

Editor: Advanced Machine Learning

You can find our code and the results obtained on the following GitHub : **Click Here**

Abstract

In this project, we focused on implementing "**Fader Networks : Manipulating Images by Sliding Attributes**" Guillaume Lample (2017), as described in the paper by Guillaume Lample, Neil Zeghidour, Nicolas Usunier, Antoine Bordes, Ludovic Denoyer, Marc'Aurelio Ranzato, using Pytorch. The model architecture is a straightforward encoder-decoder setup aimed at reconstructing images by sorting out key details in the image and directly capturing attribute values in the latent space. Through training, our model becomes capable of generating different and realistic versions of an input image by tweaking attribute values, like changing facial expressions or adding accessories such as glasses, age, or gender. The algorithm is trained using labeled images with 40 attributes, making it effective in untangling facial features. Inspired by Fader Networks, the model allows users to easily tweak images using sliders. The model consists of an Encoder creating a latent space and a Discriminator figuring out attributes from this space. This interplay results in a sturdy encoder independent of specific image features and a decoder reconstructing images based on the latent space and desired attributes. The study uses the easily accessible CelebA dataset, implementing the encoder-decoder architecture from the paper to explore the nuances of Fader Networks in image manipulation.

Keywords: Machine Learning, Fader Networks, encoder-decoder, discriminator, loss function, optimizer

1. Introduction

We've tried to reproduce exactly the architecture described in the article. First, we studied the article carefully to understand the methods and details, such as the metrics and parameters used. The algorithm we trained could create a new image of a face with different characteristics from those of the original image. It was exciting to start at the beginning and see how our trained models modified

the original images. We trained three models : one to change a person's facial expression by adding or removing a smile, another to add or remove glasses, and the third to change gender. These models provide various possibilities across two distinct applications. Firstly, the Single-attribute swap feature enables the selective modification of a specific characteristic in the original image. Lastly, we came up with a way to show the changes step by step using attribute interpolation. By assigning several weights to the same attribute, we can witness a gradual transformation over a number of images, observing the changes intensify as the weights change. These applications demonstrate versatility and potential of the trained model in dynamically manipulating visual attributes with precision When we examined the changes between the original image. A big challenge we faced in the learning process was making the new pictures look real. Some features, like glasses, didn't show up a lot in the dataset. For example, out of 202,599 pictures, only 13,193 had people wearing glasses. This challenges the autoencoder to search deeply and extract the distinctive signs of attributes.

2. Algorithm presentation

An (image, attribute) pair (x, y) is given as input. The encoder maps x to the latent representation $E(x) = z$; the discriminator is trained to predict y given z whereas the encoder is trained to make it impossible for the discriminator to predict y given z only. The decoder should reconstruct x given (z, y). At test time, the discriminator is discarded and the model can generate different versions of x when fed with different attribute values.

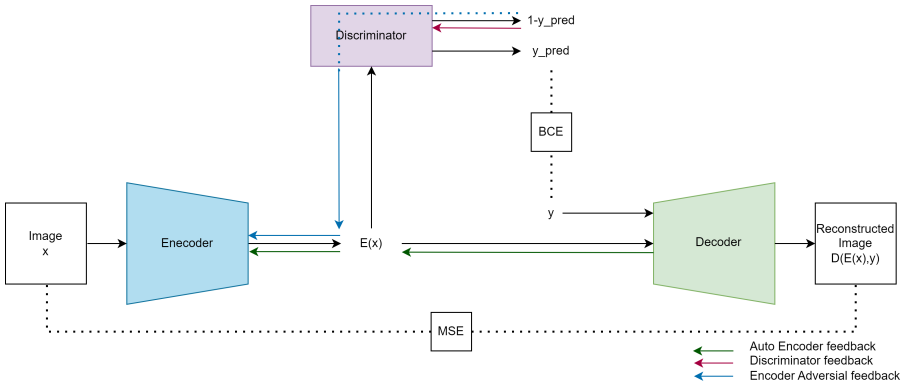

FIGURE 1 – Main architecture : Encoder, decoder and discriminator

Encoder : The encoder comprises a series of convolutional layers designed to progressively reduce the dimensions of the image from 256x256x3 to 2x2x512. We employed a **kernel** size of 4x4 with a **stride** of 2 to effectively halve the size of the image at each convolutional operation. Additionally, we applied **Batch Normalization** to ensure stable and efficient training. Furthermore, a **ReLU activation function** was employed to introduce non-linearity to the model, enhancing its capacity to capture complex patterns and features.

Decoder : The decoder, the inverse of the encoder, comprises a series of deconvolutional layers aimed at progressively restoring the images to their original size of 256x256x3. We utilized identical parameters, including a **kernel** size of 4x4, a **stride** of 2, **Batch Normalization**, and a **ReLU activation function**. The latent code is seamlessly integrated as additional constant input channels for all decoder convolutions. Denoted by n representing the number of attributes,

Discriminator : The discriminator consists of one convolutional layer followed by two fully connected layers. A **dropout** rate of 0.3, as mentioned in the article, was found to be highly beneficial. Additionally, a **sigmoid** function is applied to obtain the predicted attributes. We hesitated

between the softmax function and the sigmoid function, but it seems that the sigmoid function gives us better results.

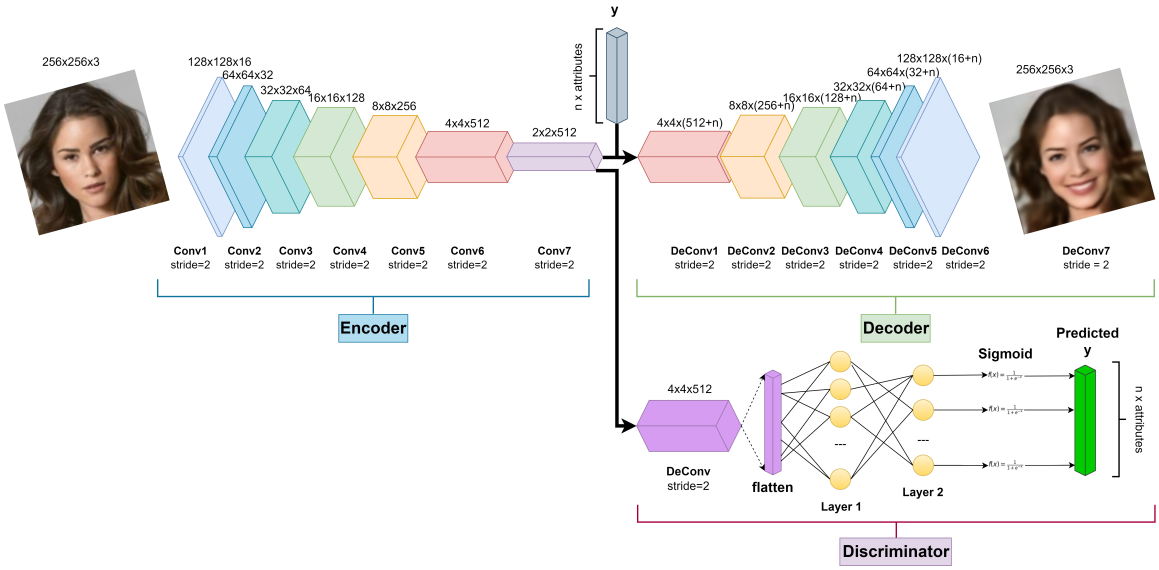

FIGURE 2 – Detailed architecture : Encoder, decoder and discriminator

To evaluate the model, three key loss functions are computed during both training and evaluation. The Mean Squared Error **MSE** loss is expressed as :

$$\text{MSE Loss}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(\text{reconstructed}_i(\theta) - \text{original}_i)^2$$

Simultaneously, the Binary Cross Entropy **BCE** loss is calculated using the following equation :

$$\text{BCE Loss}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}[\text{original}_i \cdot \log(\text{predicted}_i(\theta)) + (1 - \text{original}_i) \cdot \log(1 - \text{predicted}_i(\theta))]$$

The adversarial loss, combining the MSE-based reconstructive loss with a weighted discriminator loss, is given by :

$$\text{Adversarial Loss}(\theta) = \text{MSE Loss}(\theta) + \lambda_E \cdot \text{BCE Loss}(\theta)$$

Here, N represents the number of samples in the dataset, and θ denotes the model parameters. The adversarial loss introduces a dynamically adjusted weight, λ_E, for the discriminator loss coefficient.

3. Data

We used the CelebA dataset, the same one used in the referenced article, which includes 202,599 images. The dataset is easily accessible and includes a file, list_attr_celeba.txt, containing the attributes associated with each of the 40 attributes listed. As part of the preprocessing of the dataset, we resized the images from 178x218x3 to 256x256x3. In addition, we mapped the attributes according to the following scheme : -1 is translated into [0, 1], and 1 is translated into [1, 0]. For example, if an individual has the attribute **smiling** with a value of -1, this indicates that the person is not smiling. Conversely, if the attribute **Eyeglasses** is equal to 1, this means that the person is

wearing eyeglasses. Our algorithm was trained and tested using the same dataset as presented in the referenced article. Throughout the training process, we normalized the images to lie between -1 and 1.

FIGURE 3 – Preview of Data with Selected Attributes

80% of images and their corresponding attributes were used for the training, 10% for the validation, and the last 10% for the training. One of the major problems encountered when using this dataset was the impossibility of loading all the images into RAM due to their large size.

4. Experimental evaluation

4.1 Description of the experiment

We adopted the algorithm presented in the article and leveraged the **Pytorch** Framework to construct our neural network, which was subsequently trained on the university's GPU. Due to various issues encountered with the GPU, we were unable to train our model for a large number of epochs as was done in the article where they trained their model for 1000 epochs. We were unable to train all three at the same time because as soon as two people started training simultaneously, the learning would stop, and this could not be resolved even by changing the number of workers in the DataLoader. We trained the autoencoder and the discriminator simultaneously for approximately 60 epochs. However, due to a lack of clarity on the desired final image output, we were unable to use specific metrics for model evaluation or conduct qualitative evaluation by human users as was done in the referenced article. We developed a system to save the trained models every 10 epochs, providing a safety net in the event of a kernel crash so that we always have a backup.

4.2 Results

FIGURE 4 – Results obtained by training the model on the attribute **Smiling**

FIGURE 5 – Results obtained by training the model on the attribute **Eyeglasses**

FIGURE 6 – Results obtained by training the model on the attribute **Male**

As we can see from the obtained grid, by using a multiplicative factor α during inference, which controls the rate of change, we are able to achieve a satisfactory result. The reconstruction and discriminator losses exhibit the expected behavior with significant fluctuations, indicating that there is potential for improvement if our model were trained for a greater number of epochs.

Smiling trained model	**Eyeglasses trained model**	**Male trained model**
60 epochs	45 epochs	140 epochs

TABLE 1 – Training Epochs for Different Models

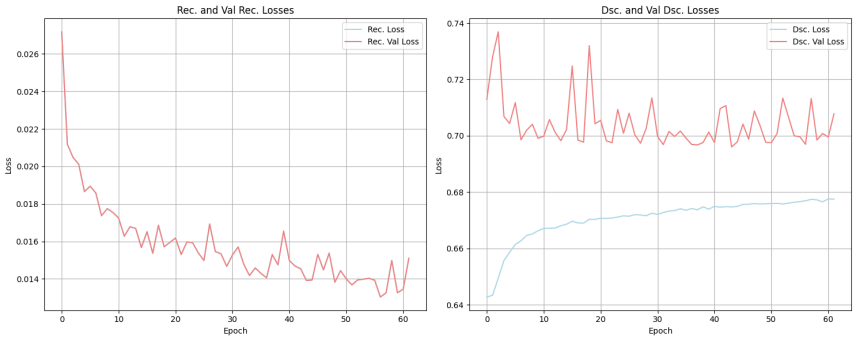

FIGURE 7 – Losses graph of **Smiling** trained model

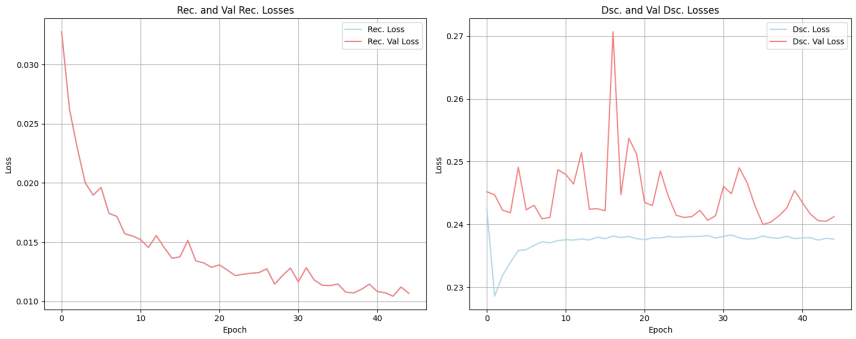

FIGURE 8 – Losses graph of **Eyeglasses** trained model

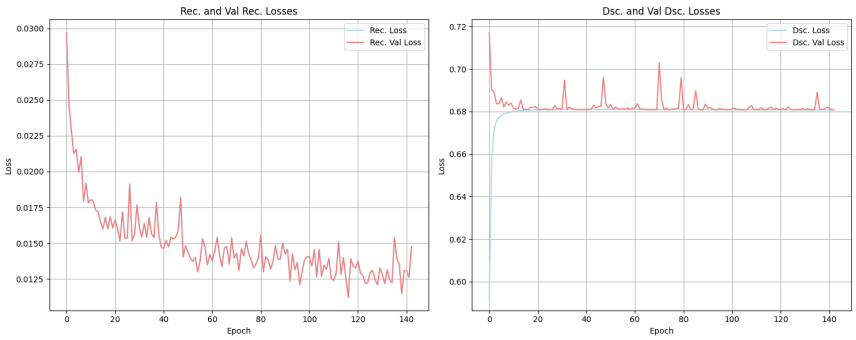

FIGURE 9 – Losses graph of **Male** trained model

4.3 Discussion

During training, we observed that the change in certain attributes was more effective than others, attributed to the quantity of data available in the CelebA database. For instance, in our training set, we noticed that the model required significantly fewer epochs to learn to remove the **eyeglasses** attribute compared to adding it. Our histogram presentation, illustrating the frequency of each attribute, revealed that only **6%** of all images included glasses. To address this issue, data augmentation could be employed by adding rotations and translations to existing images. Referring to the histogram, we gained some confidence in the successful training of models for **smiling** and **gender** attributes. This stems from the observation that approximately **50%** of the training set comprises males, and the remaining **50%** consists of females, reflecting a similar distribution for the **smiling** attribute. Furthermore, we also observed some facial distortions during training on the **smiling** attribute, indicating that with a higher number of epochs, this could potentially be resolved as the model would better learn to reconstruct a smiling face.

5. Conclusion

This project centered on the implementation of the "Fader Networks" algorithm, offering a holistic learning journey from data preprocessing to model evaluation. Beyond the detailed explanations provided in the article, the experience emphasized the necessity for a profound understanding of key components such as losses, optimizers, and tensor usage. Despite encountering challenges, notably related to computer performance limitations, the project provided valuable problem-solving insights.

Effective use of GitHub emerged as a crucial skill, enhancing collaborative coding practices and teamwork within the project. The potential improvements on the horizon include enhancing generalization by delving into various data augmentation techniques. This exploration aims to diversify the dataset, thereby bolstering the model's ability to adapt across a spectrum of scenarios. This is significant because certain attributes, such as glasses, are not very common in our entire dataset. we would have liked to develop the Multi-attributes swap function that allows you to tweak multiple aspects of an image simultaneously. For instance, you could play around with the gender, introduce accessories like glasses, and make all these changes in one cohesive operation. Another avenue for advancement involves investigating the integration of augmented reality elements into the model. By incorporating such elements, the goal is to achieve more authentic transformations of visual attributes, ultimately enriching the user experience and the applicability of visual modifications. The application of Fader Networks not only proved enjoyable but also showcased the model's versatility in dynamically manipulating visual attributes with precision.

6. Bibliographie

Références

Nicolas Usunier Antoine Bordes Ludovic Denoyer Marc'Aurelio Ranzato Guillaume Lample, Neil Zeghidour. Fader networks : Manipulating images by sliding attributes. 2017. https://arxiv.org/abs/1706.00409.