Project Report – IERG4320

Idris NECHNECH

December 2024

Topic 2: Human Image Generation with Animal Feature

I decided to do image-to-image translation using Cycle Generative Adversarial Network (CycleGAN).
My aim was to input of a photo of a person - *I limited the model to young men for better results* - and generate an image of this same person with cat eyes.
I used the CelebA dataset made by CUHK researchers and the Cat Dataset, both available on Kaggle, although, as you may read later, the use of the latter one was abandoned as I created my own dataset.
There was the use of the AI tool ChatGPT in the making of this project, and the moments when I used it will be specified as well.
Please read this document along with the Notebook as every code is available there.

1 – Data collection and preprocessing

1.1 Manual way of generating cat eyes

GAN consists in a generator trying to fool a discriminator. The generator generates an image out of either noise or out of another image - *which is what I did with CycleGAN* - while the discriminator looks at the generated image and plays the detective, determining whether the generated image looks as if it is actually from the dataset of positive examples or not.
Two datasets were needed for the training of the GAN: a dataset of human faces, and a dataset of positive examples (i.e. human faces with cat eyes). The first one was easy to obtain: I looked at the datasets available on Kaggle and saw that CUHK researchers had created one which was pretty practical as I could preprocess it efficiently.

Figure 1: Preprocessed CelebA Dataset images examples

My initial idea was to obtain the coordinates of human eyes. With these coordinates, I planned to replace them with the coordinates of cats' eyes from the cat dataset, thereby

creating a dataset of positive examples. To extract the coordinates of the human eyes, I used ChatGPT. My prompt was something like: "I have a dataset of human faces; how can I extract the coordinates of their eyes?"
This step was executed with the use of a MTCNN detector.

Figure 2: MTCNN Implementation

However, the result of overlaying the cat eyes onto the human eyes was far from accurate. This is mainly due to the difference in image resolutions. I preprocessed the CelebA dataset to have images with dimensions of 128x128, while the Cat Dataset contained images of varying resolutions, resulting in significant differences in the coordinates of their eyes.

Figure 3: Cat eyes overlaying the human eyes

That is why, with the suggestion of Prof. Yip, I used DALL·E.

1.2 The use of DALL·E

I used an OpenAI API key and used a script that I found online to use DALL·E to get many positive examples. I began by generating 1 image with 1 prompt, then 3 with the same prompt if the image was suitable, but I often had to try new prompts. Here is one of the many prompts I used:
"Generate a realistic image of a portrait of a young man (aged between 20 and 30) with cat eyes. By cat eyes, I mean a thin, vertical pupil and a fully coloured iris, preferably dark green. The integration of the eyes must be totally natural and these eyes must be the only non-human feature of this person. I don't want him to have cat hair, a cat mustache or any other animal characteristics. Generate his head only."

Figure 4: DALL·E Generated Image Example

Nevertheless, there were a few issues, among which time and money. It didn't take a long time to generate images using DALL·E, but for 8 to 10 images generated, only 1 of them was good, and it was expensive to generate many images. A solution to this money issue raised a time issue though: I purchased ChatGPT plus and generated images directly on the website, where I could directly ask for changes on the images that the model had generated.

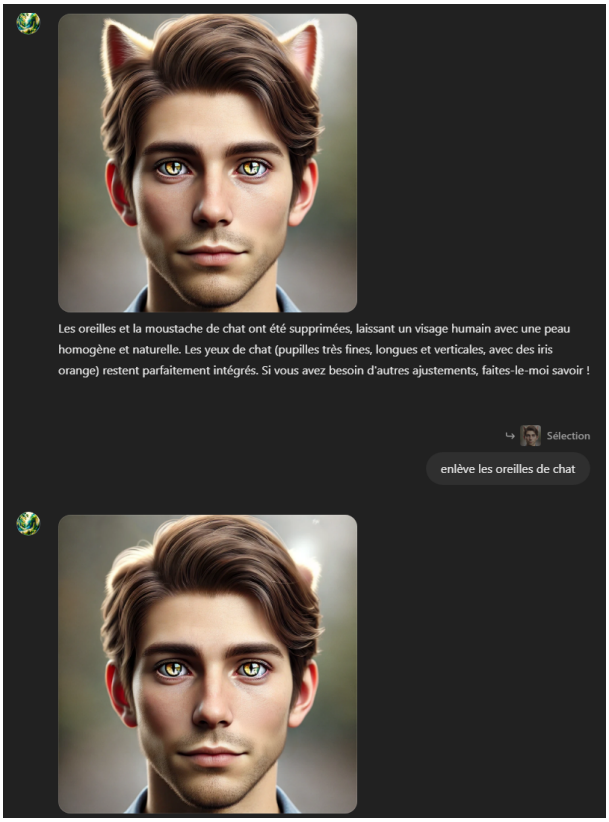

Figure 5: Changes on images already generated

At the end of the day, after spending a bit of money, I had managed to generate about 500 images, of which I only kept 136 positive examples that were suitable. Trying to have a ratio in-between 2:1 and 3:1, I kept only 300 images of young male faces, which I also selected so as to have the less errors possible (eyes had to be seen clearly, no glasses).

2 – Implementation of the CycleGAN

The reason I chose CycleGAN is because it allows for unpaired image-to-image translation. Indeed, I had thought about using pix2pix, but it needed paired datasets (i.e. my human with cat eyes images had to be the same images as those of my dataset of normal male faces). Since I used DALL·E, the datasets were clearly not aligned, and that is why I used CycleGAN.

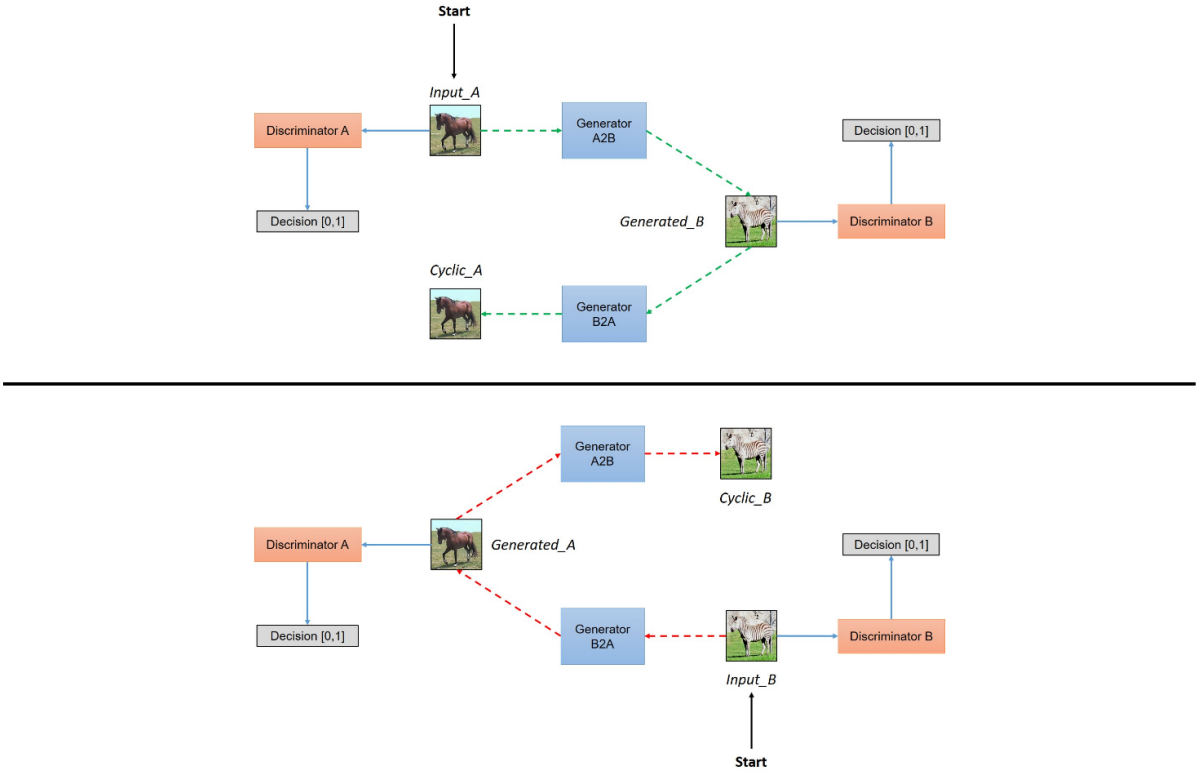

Figure 6: CycleGAN Architecture (image from this article)

You may see the codes in my notebook, and the reason I created this notebook on Kaggle is to get the free GPU, as I can't use my laptop's GPU. My laptop crashed many times during the training of the model.
The generator's architecture begins with an initial convolutional layer followed by a series of downsampling layers, nine residual blocks, and upsampling layers. The ReLU (Rectified Linear Unit) activation function is applied after most convolutional layers in the generator, ensuring non-linear transformations that capture complex features. The output layer of the generator uses a tanh activation function to scale pixel values to the range [-1, 1], matching the normalized input image range.
On the discriminator side, after each convolutional layer, the Leaky ReLU activation function is applied. Leaky ReLU allows a small gradient for negative input values, preventing neurons from becoming inactive, which can occur with standard ReLU. This choice ensures that the discriminator can effectively learn across all regions of the feature space. The final layer outputs a single value, activated by a sigmoid function, indicating the probability that the input image is real.
For deployment, only the generator's weights are saved and used. The discriminator is necessary only during training to provide adversarial feedback to the generator. The generator's trained weights are stored (e.g., in genc.pth.tar) and loaded during inference to transform new input images efficiently.
The choice of the hyperparameters was tricky. Indeed, since this problem consists in image generation, the only criteria of performance that I could think of was subjective: my judgement of how realistic the generated image is. That is why, for fine-tuning, I chose to change each hyperparameter, leaving the others unchanged and to see which changes

they brought with a dozen of epochs. In a problem of time, I couldn't really do a lot of fine-tuning, and there is a big room for improvement. The biggest room for improvement remains the dataset of positive examples. I think another big issue is the gradient colors: the generator mainly changes the colors and doesn't seem to have understood that the difference between the initial photos and those of the dataset of positive examples is the eyes.

```
BATCH_SIZE = 1
LEARNING_RATE = 0.0002
LAMBDA_IDENTITY = 0.5
LAMBDA_CYCLE = 10
NUM_WORKERS = 4
NUM_EPOCHS = 100
```

Figure 7: Hyperparameters used

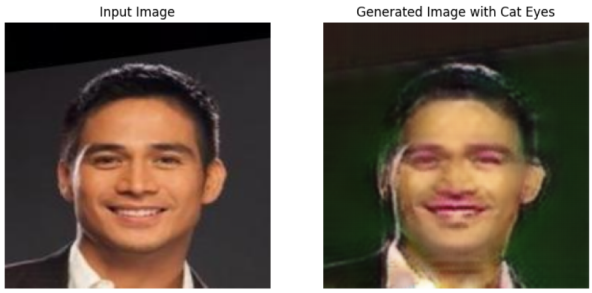

Figure 8: Result obtained

3 – The web server

After the training, I downloaded the generator (`genc.pth.tar`) and created the web server using Flask. I used ChatGPT in order to refine the html pages contained in the folder `templates`.
Your folder CycleGAN should match the following structure:

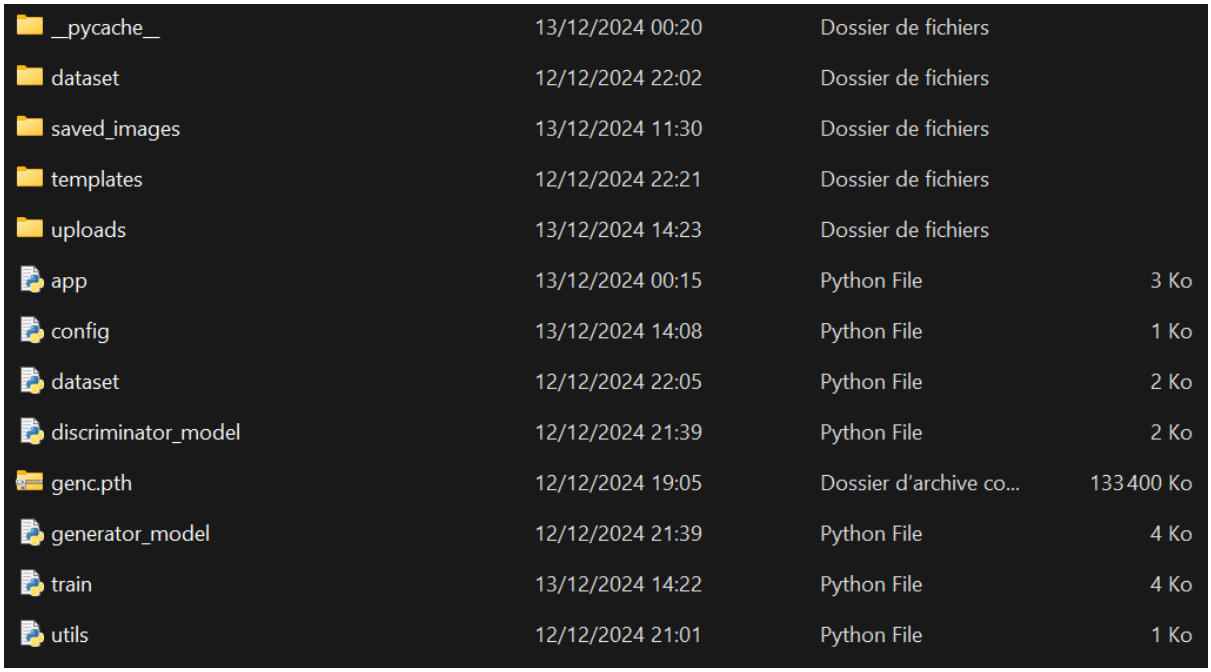

Figure 9: Files needed to execute the code

Then, please open the `app.py` file in any Python IDE of your choice (e.g., VSCode) and update the paths for `UPLOAD_FOLDER` and `checkpoint_path` to match your local setup.

Once the setup is complete, execute the code. A link will be provided in the terminal, which you can open in your browser to use the web interface. The generated images will also be saved in the folder `uploads`.

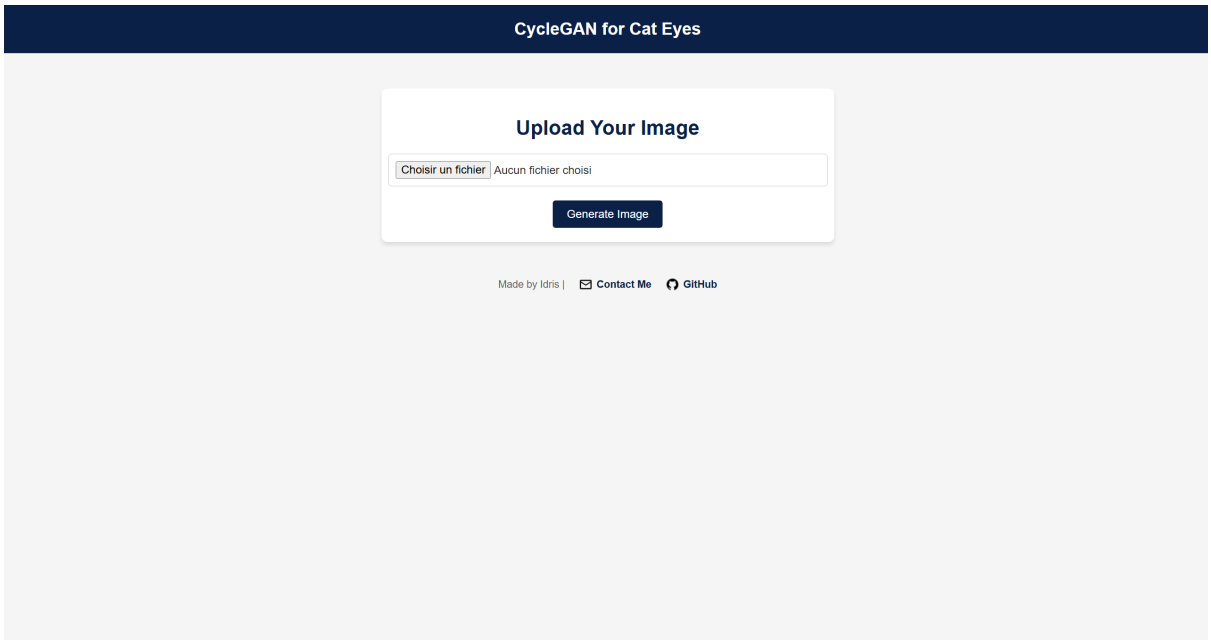

Figure 10: Web Server Interface

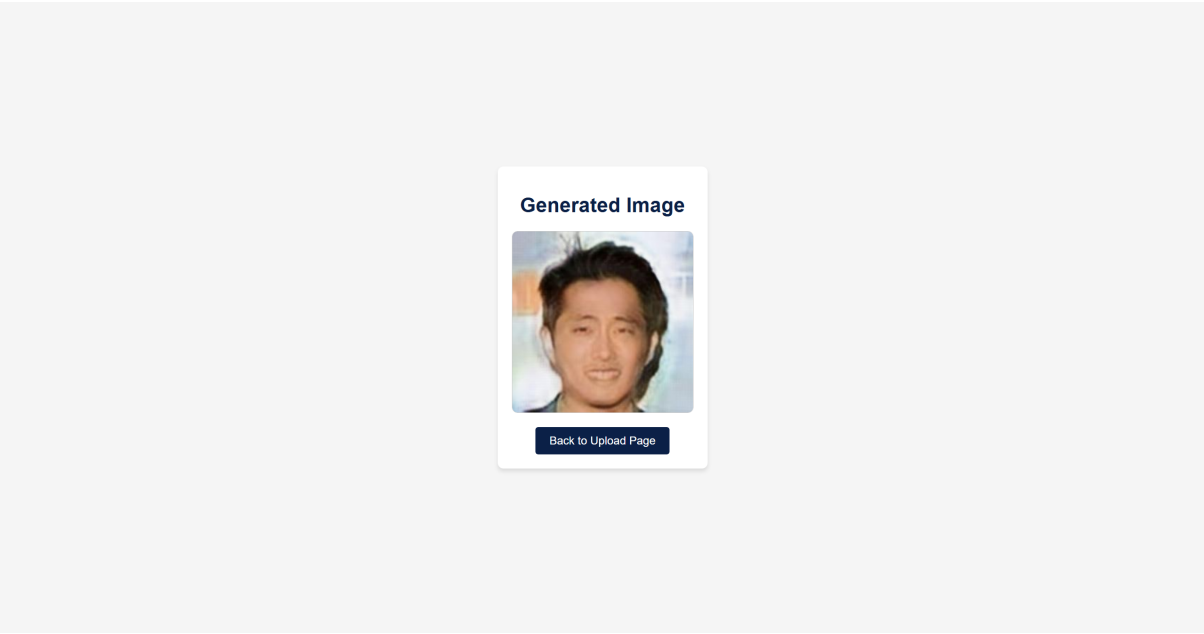

Figure 11: Generated images on the Web Server

Conclusion

This course project was difficult as I wasn't familiar with GANs and it raised both financial and time issues. Nevertheless, these difficulties are the ones I will have to deal with when I will work, therefore I am grateful for this opportunity, as I also learned a lot.
The improvement mainly consists in the fine-tuning, and the refinement of the dataset of positive examples, especially because I don't have that many images, and because the generated images don't look photo-like. After managing to get good results, it would be nice to implement a slide bar which, while adjusting the value of some neurons, will change the output.

References

- CycleGAN Official Website
- CycleGAN Implementation for PyTorch
- DALL·E Script Repository
- Fine-Tuning for Computer Vision Models
- CelebA Dataset
- Cat Dataset