Generating medieval portraits from modern images using generative adversarial network

Zsombor Élő
BME
elozsombor@gmail.com

Gábor Szolnok
BME
szolnokgabcsi@gmail.com

Abstract

This document contains the technical report for our final submission of the subject "Deep learning" at Aquincum Institute of Technology. The goal of the project is to cascade a face detector and a generative adversarial network (referring to it as GAN from now on), making it possible to identify a person on a real life image and then make a portrait of them in a renaissance style. Unfortunately, we could not achieve a perfect solution, however the results of the evaluation show remarkable changes from the original images.

1 Introduction

We have chosen this task for our work as it can be separated into two parts easily and so that we can work on it independently. Therefore, throughout this documentation we are explaining our work in two sections. The first part of the model aims to detect faces on everyday images. Many state of the art models were already created for this task so we have decided not to focus on improving on this part. We are using an xml exported cv2 model published by Shantu [1].

Even if we did not modify the cascade, we have attempted to optimize its parameters. We have also defined some validation rules for the input using this part of the model. The result of this part is not a model that can be trained and evaluated, this is rather an architecture that serves as a service for validating and preprocessing input data.

The second part of the model is for translating the faces and making medieval style portraits. For this task we used a GAN (Generative Adversarial Network), more specificly a CycleGAN. There are many state of the art models, which are producing very great results, though every one of them was trained with basically as powerful GPUs, as powerful it was necessary, and there wasn't any time restriction on them. Because Google Colab[2] gives resources limited in some way, therefore we couldn't train our model to the utmost perfection, though some results show very good progression toward our goal. We tried some different architectures, though the first implementation was the best (U-NET for generators).

2 Datasources

As we are not training the face detector, we only need datasets for the GAN. We have collected a set of faces from the UTKface[3] and from the widerface[4] sources. These were quite large and had a huge amount of low quality images which was not proper for the training of the model, therefore we needed to filter them out. We have collected 1000 faces overall for the training. Before using them hovewer, we have preprocessed them by the face detector, zooming in on the faces found on the images.

The other dataset we have used for training is the WikiArt's dataset collected into the ArtGAN[5] repository. This source has also been too huge, so we've collected the renaissance styled portraits from them creating a set of 1000 training paintings and 100 test paintings.

3 Architecture

3.1 Face detector

We have used Shantu's cascad with the help of the cv2 library. We have tried various parameters and found that these were the most optimal: scaleFactor = 1.1, minNeighbors=2. The reason behind this is that the images were filtered so that they contain at max 2 faces and at max one of them can be spotted

[1] https://github.com/shantnu/FaceDetect/

[2] https://colab.research.google.com/

[3] https://drive.google.com/drive/folders/0BxYys69jI14kSVdWWllDMWhnN2c

[4] https://paperswithcode.com/dataset/wider-face-1

[5] https://github.com/cs-chan/ArtGAN

Figure 1: A sample of an image before and after processing it with the help of the face detector

easily. This part of the model has the same function during training, evaluation and when inferencing too: it detects a face zooms on it and exports the result image cropped as if it was a portrait.

We have also added a validation rule here: if the model detects more or less than one face, the image will not be preprocessed. This step helped us increase the quality of the preprocessed dataset.

When training the model, we push the whole face dataset through this part and use the output as a resource for the training of the GAN. Therefore the output is also resized, unifying the sizes of images to 256x256.

3.2 CycleGAN

We tried different architectures for implementing the CycleGAN [6]. There were 2 entirely different conceptions for the generators: U-NET and ResNet. There were some different implementations for both of them, but in the end the first trial, the pix2pix model's generator's implementation by TensorFlow[7] won. As I said, we tried to improve the results with different architectures, but simply this was the best among them. We decided this fact by looking at generated images and using an evaluation method called Frechét Inception Distance (introduced here[8]).

4 Results

4.1 Face detector

The cv2 model has proven to be quite useful for this task. Most of the faces are detected and cropped as expected creating a proper input for the training of the GAN. However some of the images are cropped improperly, meaning the result does not contain a 'portrait like' face, so the input for the GAN is not 100 percent clean.

[6] https://arxiv.org/pdf/1703.10593.pdf

[7] https://www.tensorflow.org/tutorials/generative/pix2pix

[8] https://arxiv.org/pdf/1706.08500.pdf

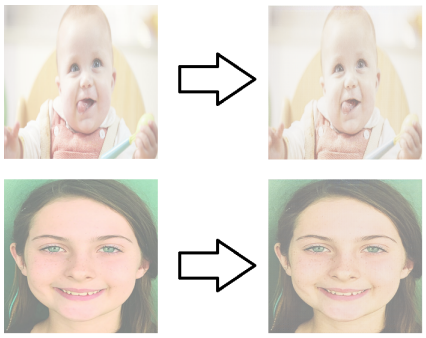

Figure 2: Bad results. It was expected, that some of them will show up, because the training time was shorter, than necessary.

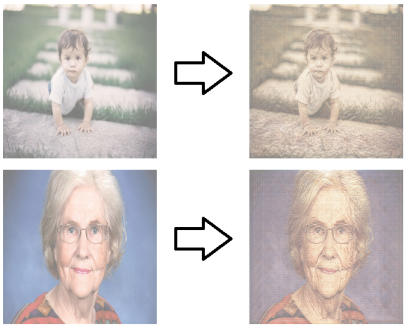

Figure 3: Better results. These outputs show some progress toward the medieval paintings, especially in the color department.

4.2 CycleGAN

There are bad and good (not so bad) results. This can be, because of the limitations we had during the project. The good results show some of the features, that old / medieval paintings. The colors kinda match, and the lines / outlines become more bold. The results are shown in 2, 3 and 1.

FID	412.0842

Table 1: Used the Inception V3 model for measuring the FID. The score is between 1 and 2048, because the Inception V3 embeds the input into (2048,) before making the prediction in the last layer.

4.3 Conclusion

I think, we tried our best, to deliver, what we were expected to, unfortunately we couldn't improve on an already (somewhere else, for some other specific tasks) implemented model. The road was a bit bumpy, we've ran into some issues during the project, but in the end we finished it in time. The results are showing, that the effort was there, bit more time, and it could have been better we think. We learned many things throughout this project, for example the concatenation of two models, using Google Colab properly and implementing things, we aren't expert in. We are glad, that we chose this project, it was fun while it lasted.

4.4 Future works

Due to lack of time and resources, we could not train the model with more than 35 epochs. Having the model trained with more time, the results might also look better.