
NATIONAL CHIAO TUNG UNIVERSITY

**Institute of Computer Science and Engineering**

# Selected Topics in Visual Recognition using Deep Learning

# Homework 2

Author: Alessandro Saviolo

Student Number: 0845086

**6th November 2019**

# Summary

# 1. Introduction

The purpose of this document is to present and analyze the Generative Adversarial Network model built for Homework 2. The dataset used for training the model is CelebFaces Attributes (CelebA) [3–5].

Generative Adversarial Networks (GANs) [1] are a deep learning method generating synthetic data from a given distribution. Such networks are implemented by a system of two neural networks, namely the Generator and the Discriminator, contesting with each other. The generator model takes as input a random noise vector $z$ taken from the latent space and outputs a RGB image. This is achieved by using a fully connected layer to interpret the point in the latent space and provide sufficient activations that can be reshaped into many copies of a low-resolution version of the output image. This is then upsampled multiple times using transpose convolutional layers. The discriminator model takes as input one RGB image and outputs a binary prediction as to whether the image is real (CelebA dataset) or fake (produced by the generator).

The generator and the discriminator are implemented using the best practices for GAN design such as using the LeakyReLU activation function, a $2 \times 2$ stride to downsample, the Adam optimizer with a learning rate, the hyperbolic tangent (TanH) activation function in the output layer of the generator, the vector $z$ sampled from a gaussian distribution (many of them suggested in [7]).

The code of the project has been uploaded to Github at the repository [8].

# 2. Methodology

## 2.1 Data Preprocessing

CelebFaces Attributes Dataset (CelebA) is a large-scale face attributes dataset which contains more than 200K celebrity images (Figure 2.1 displays some example images from CelebA dataset).

Since the task requires to work only on celebrity faces and Generative Adversarial Networks are very hard to train, I have cropped the portion of image which includes the background and resized the images down to $56 \times 56$.

Figure 2.1: Images from CelebA Dataset.

## 2.2 Tuning Hyperparameters

The Generative Adversarial Network has been trained for 10 epochs, using Adam optimizer with learning rate equal to 0.00025 and exponential decay rate for the 1st moment estimates equal to 0.45. The batch size is fixed to 64. The random noise vector $z$ passed in input to the generator has been sampled from a gaussian distribution. The dimension for such noise vector is 100.

## 2.3 Model

Generative Adversarial Networks (GANs) are a class of artificial algorithms used in unsupervised learning algorithm, implemented by a system of two neural networks: the Generator and the Discriminator. Both networks are contesting with each other. The discriminator has the task of determining whether a given image looks like an image from the dataset or if it looks like it has been artificially created. On the other hand, the task of the generator is to create natural looking images that are similar to the original data distribution, images that look natural enough to fool the discriminator network.

The generator is trying to fool the discriminator while the discriminator is trying to not get fooled by the generator. As the models train through alternating optimization, both methods are improved until a point where the fake images are indistinguishable from the dataset images.

### 2.3.1 Generator

The generator takes a random noise vector $z$ and passes it to a series of upsampling layers. Each upsampling layer represents the transpose convolution operation (i.e., deconvolution operation). The series of transpose convolutions reduce the input from 1024 all the way down to 3 — which represents an RGB color image.

The generator architecture is presented in Table 2.1. Initially, the model has a dense layer followed by batch normalization, leaky relu and a dropout layer. Then, the structure is defined by modules of deconvolutional layers, each followed by batch normalization, leaky relu and a dropout layer (except for the output layer). The output layer is defined by a deconvolution operation and outputs a $56 \times 56 \times 3$ tensor through the Hyperbolic Tangent (TanH) function.

### 2.3.2 Discriminator

The discriminator takes an image of size $56 \times 56 \times 3$ and performs strided convolutions on it. At last, the discriminator shows the output probabilities for deciding whether the image is real or fake using the Sigmoid function.

The discriminator architecture is presented in Table 2.2.

| Layer (type) | Input Shape | Output Shape |
|---|---|---|
| Dense | (100) | (4, 4, 1024) |
| Batch Normalization | | |
| Leaky ReLU | | |
| Dropout | | |
| Conv2D Transpose | (4, 4, 1024) | (7, 7, 512) |
| Batch Normalization | | |
| Leaky ReLU | | |
| Dropout | | |
| Conv2D Transpose | (7, 7, 512) | (14, 14, 256) |
| Batch Normalization | | |
| Leaky ReLU | | |
| Dropout | | |
| Conv2D Transpose | (14, 14, 256) | (28, 28, 128) |
| Batch Normalization | | |
| Leaky ReLU | | |
| Dropout | | |
| Conv2D Transpose | (28, 28, 128) | (56, 56, 3) |
| TanH | | |

Table 2.1: Generator architecture. Dropout is applied with rate 0.5. Batch Normalization is applied to reduce the amount by what the hidden unit values shift around (i.e., covariance shift). Leaky ReLU is used instead of standard ReLU to worry less about the initialization of the models and to face the "Dead ReLU" problem (i.e., a bad initialization may cause some neurons to be deactivated at the start of the training phase). In particular, Leaky ReLU is used with slope 0.2.

| Layer (type) | Input Shape | Output Shape |
|---|---|---|
| Conv2D | (56, 56, 3) | (28, 28, 64) |
| Batch Normalization | | |
| Leaky ReLU | | |
| Dropout | | |
| Conv2D | (28, 28, 64) | (14, 14, 128) |
| Batch Normalization | | |
| Leaky ReLU | | |
| Dropout | | |
| Conv2D | (14, 14, 128) | (7, 7, 256) |
| Batch Normalization | | |
| Leaky ReLU | | |
| Dropout | | |
| Conv2D | (7, 7, 256) | (4, 4, 512) |
| Batch Normalization | | |
| Leaky ReLU | | |
| Dropout | | |
| Dense | (4, 4, 512) | (1) |
| Sigmoid | | |

Table 2.2: Discriminator architecture. Dropout is applied with rate 0.5. All convolutional operations use stride equal to 2 to apply downsampling. Batch Normalization is applied to reduce the amount by what the hidden unit values shift around (i.e., covariance shift). Leaky ReLU is used instead of standard ReLU to worry less about the initialization of the models and to face the "Dead ReLU" problem (i.e., a bad initialization may cause some neurons to be deactivated at the start of the training phase). In particular, Leaky ReLU is used with slope 0.2.

## 2.4 Loss

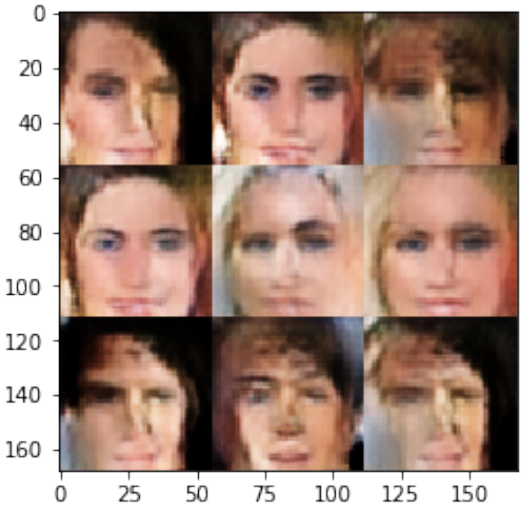
The discriminator is receiving the images from both the training set and the generator, so while calculating discriminator's loss we have to add loss due to both fake and real images. Moreover, both networks are trained simultaneously so we need two optimizers. We want from discriminator to output the probabilities close to 1 if the images are real and close to 0 if the images are fake.

The loss function used is the Minimax Loss, where the generator tries to minimize the following function while the discriminator tries to maximize it:

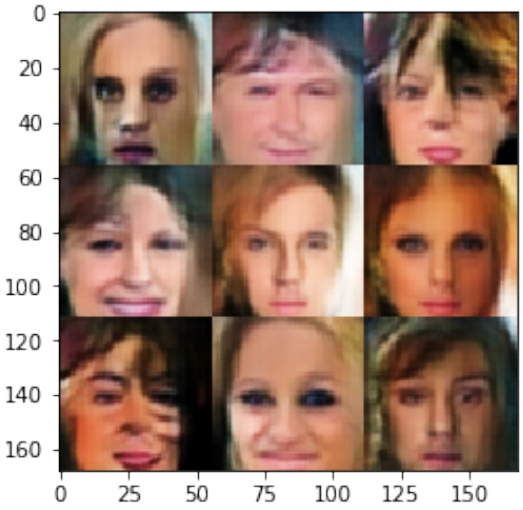
$$E_x[\log(D(x))] + E_z[\log(1 - D(G(z)))]$$

where $D(x)$ is the discriminator's estimate of the probability that real data instance x is real, $E_x$ is the expected value over all real data instances, $G(z)$ is the generator's output when given noise $z$, $D(G(z))$ is the discriminator's estimate of the probability that a fake instance is real, $E_z$ is the expected value over all random inputs to the generator.

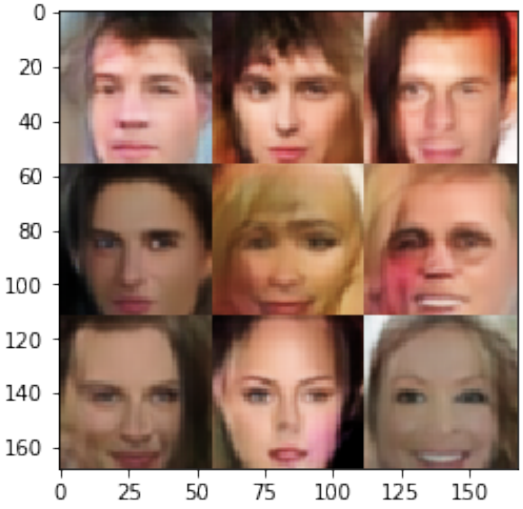
# 3. Results



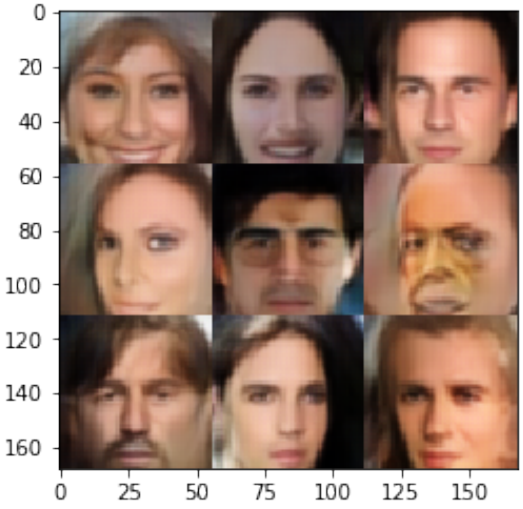

(a) Epoch 1



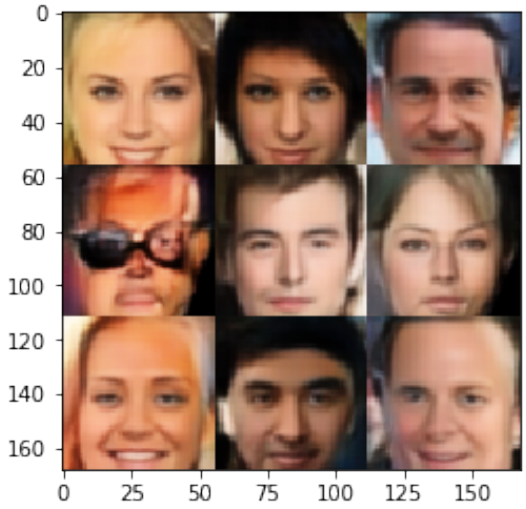

(b) Epoch 2



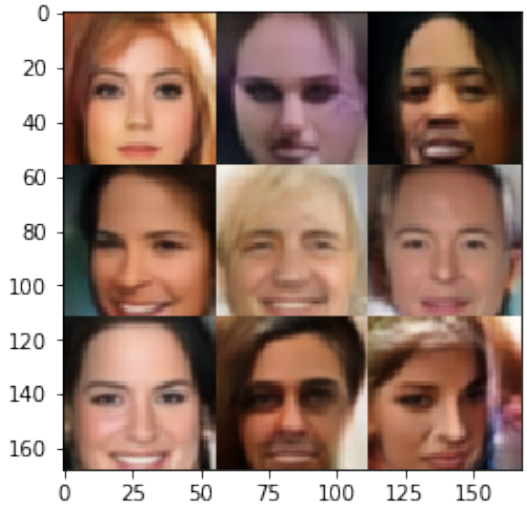

(c) Epoch 3



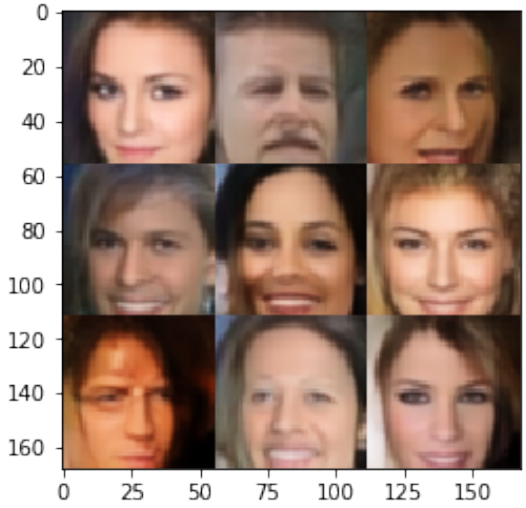

(d) Epoch 4



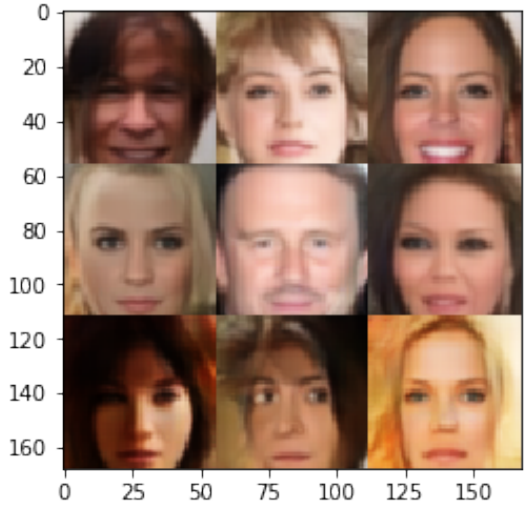

(e) Epoch 5



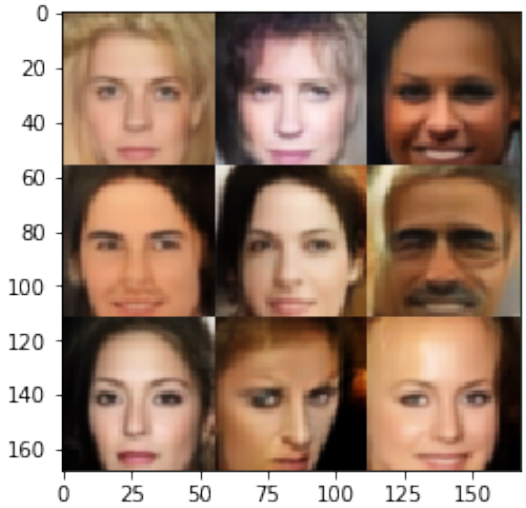

(f) Epoch 6



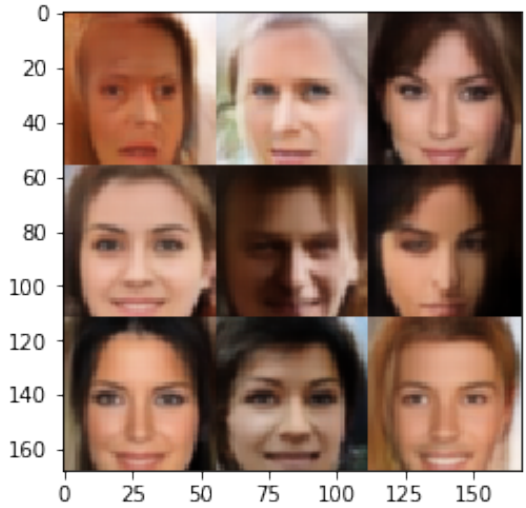

(g) Epoch 7




(h) Epoch 8

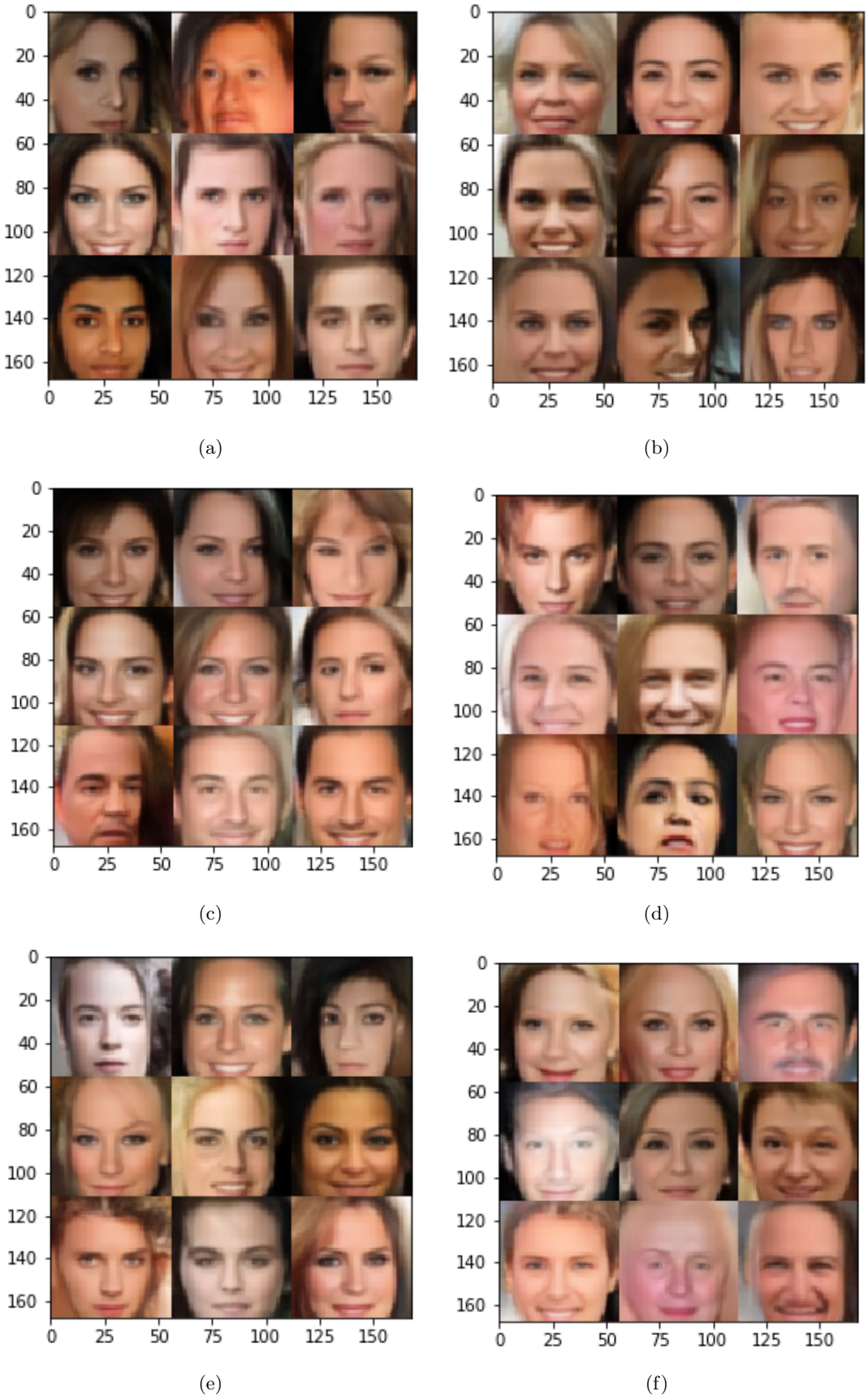



(i) Epoch 9




(j) Epoch 10

Figure 3.1: Image grids produced by the model **while** training for 10 epochs. Each image has size $56 \times 56$. When the training process is going on, the generator produces the set of images and after every epoch it gets better and better.




(a) (b) (c) (d) (e) (f)

Figure 3.2: Image grids produced by the model **after** training for 10 epochs. Each image has size $56 \times 56$. Some grids contain images that are not realistic (e.g., second image of the first row in $(a)$), but in general the solutions produced by the GAN model are accurate. The model required about 3 hours and 47 minutes to be trained for 10 epochs using Google Colab [9].

# 4. Summary

In this document, I have presented and described the Generative Adversarial Network model built for Homework 2. Each part of the model has been discussed and motivated.

When the training process is going on, the generator produces the set of images and after every epoch it gets better and better (Figure 3.1), so that the discriminator can not identify whether it is real image or fake image. After fully training, the model is capable of producing new fake images similar to the ones in CelebA dataset. The trickiest part is to tune the parameters so that the objective function is stable and converges, either the generator or the discriminator does not become too strong as compared to the other during training, and to avoid mode collapse (i.e., generator does not produce diverse images [10]).

To conclude, the presented Generative Adversarial Network model is capable of generating convincing images of size $56 \times 56$. The bottleneck for improving the presented model is the computational power and time required for training. If such bottleneck can be reduced, further work can be made trying to increase the size of the images and tuning the GAN architecture and the hyper parameters.

# APPENDIX

## A.1 GitHub Repository

The code of the project is available on GitHub, in the repository [8].

# Bibliography


[1] Goodfellow I., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., and Bengio Y. *Generative Adversarial Nets*. Departement d'informatique et de recherche operationnelle, Universite de Montreal, Montreal, 2014.

[2] Kingma D., and Ba J. *Adam: A Method for Stochastic Optimization*. Conference paper at ICLR, 2015.

[3] Radford A., Metz L., and Chintala S. *Unsupervised representation learning with deep convolutional generative adversarial networks*. Conference paper at ICLR, 2016.

[4] Yang S., Luo P., Loy L., and Tang X. *From Facial Parts Responses to Face Detection: A Deep Learning Approach*. Conference paper at ICLR, 2015.

# Sitography


[5] CelebFaces Attributes Dataset (CelebA) (2015), last access 30 Oct. 2019, [Online]. *http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html*

[6] Celebrity Face Generation using GANs (2018), last access 3rd Nov. 2019, [Online]. *https://medium.com/coinmonks/celebrity-face-generation-using-gans-tensorflow-implementation-eaa2001eef86*

[7] Gan Hacks (2016), last access 2nd Nov. 2019, [Online]. *https://github.com/soumith/ganhacks*

[8] Github Repository (2019), last access 2nd Nov. 2019, [Online]. *https://github.com/AlessandroSaviolo/CS_IOC5008_0845086_HW2*

[9] Google Colab (2015), last access 4th Nov. 2019, [Online]. *https://colab.research.google.com*

[10] Mode collapse in GANs (2017), last access 2nd Nov. 2019, [Online]. *https://aiden.nibali.org/blog/2017-01-18-mode-collapse-gans/*