Image-to-Image Domain Translation: XGAN

Roger Almató Baucells, Claudio Curieses Damas, Bernat Torres Bellido, Jorge Mario Uribe Gil.
https://github.com/rogeralmato/XGAN

Abstract: Our project lies within the field of image-to-image translation, I2I: translate an image from a domain (e.g. a picture of a real person) to a different domain (e.g. a human cartoon face). We implement a GAN model for single domain generation of cartoon faces, and a XGAN model (from scratch) able to translate between multiple domains (from human to cartoon and vice-versa). Our implementation was informed by many experiments conducted during the execution of the proposal, regarding aspects such as data preprocessing, testing of various combinations of model parameters, and comparisons with other models in the state-of-the-art of the literature. We fully achieved the milestones that we committed in the pre-assessment of the project. The GitHub repository of the project contains the TensorFlow codes necessary to replicate our implementation regarding our main model, the XGAN. Additional experimental results are documented in this report.

1. Motivation

Our motivation comes from three facts. First, the wide range of applications of I2I models, which make these models interesting and useful. These applications include: semantic image synthesis, image segmentation, style transfer, image painting, 3D pose estimation, image/video colorization, among others [1]. Second, the deep generative models on which I2I are generally based are particularly interesting for us in terms of our future professional development: Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs). Finally, I2I is a task that is easily communicable to every person in intuitive and graphical ways. So that we are able to share our progress outside of the field.

2. Proposal

We follow the previous literature on XGAN [2] to solve the task of image-to-image (I2I) translation. In particular we are interested in both: generating a cartoon face starting from a

real human face, and generating a human face starting from a cartoon face. The XGAN consists of two adversarial auto-encoders that capture a shared representation of the common domain semantic content in the two domains. It learns the domain-to-domain image translation in both directions: from human to cartoon and from cartoon to humans.

3. Milestones

In our mid-term presentation we defined our milestones (see Figure 2). In what follows we recall our six milestones, and explain how we accomplished them.

3.1. Understanding of GANs

We fully accomplished to understand GANs and their surrounding fields during the development of the project. Through the experiments we kept on reading the literature in the field of GANs and we took tips given by state-of-the-art implementations in order to improve our results. According to our Gann chart (Figure 1) we get to our first milestone "Understanding GANs" at the end of January (on January 20 2021), but we would like to highlight that we feel that we are still learning about GANs. Indeed, until the very last days of training and writing in our project, we still had to re-examine key concepts that became more transparent at the end of our implementation. GANs are really rich architectures that offer a great deal of modeling possibilities and we feel that we still can take advantage of them.

3.2. Single-domain GANs - cartoon faces

As can be seen in the results-experiment section we are happy to say we accomplished this goal, as we could generate different cartoon faces from a single noise vector. This experiment did not only led us to achieve this goal but also helped us to get a better understanding of TensorFlow and its environment, which in turn helped us to develop the XGAN implementation. Our second milestone "Generating single domain GANs" was fully achieved at the end of January (on January 25 2021), at least the final implementation.

3.3. Encoder-decoder- cross-domain generation

The third milestone was achieved on February 20 2021, as it corresponded to one of our more ambitious objectives. Namely, implementing the cross encoder-decoder structure that

constitutes the main architecture of the XGAN model. The main challenge involved by this milestone was to really understand how XGAN works. Namely the role of the consistency loss between the different domains, how the model was supposed to treat the encoded representations of the human and cartoon domains, etc.

3.4. Definition of complementary losses for quality

We understand and define the complementary losses that would be put to use in our final implementation on March 9 2021. This milestone consisted of defining the semantic consistency loss as our only main complementary loss (we discarded other losses and focused on the main results in the sake of time).

3.5. Training XGAN

Although "training the XGAN" was achieved on March 10. We still added some experiments to our final report after that date. Indeed, as we will see in the rest of this document, we get to this milestone after conducting various experiments, fully described in the results section. They allowed us to improve our original results and to be able to generate carton faces starting from human faces, and implementing from scratch our model.

3.6. Comparison with alternatives

While we were implementing our own XGAN from scratch, following the consecution of objectives 1 to 5 in our project, we were also exploring alternatives in the literature to compare the performance of our results with some valid benchmarks. The main comparison was done with StarGAN V2 models, which constitutes the state-of-the-art in the literature as far as we could learn. This milestone was finally achieved on March 12 2021 after experimenting in this regard as well, as we will see at the end of the results section. Moreover we also compared our code with other implementations of XGANs available on github and we documented the outputs in terms of knowledge that we got from such comparisons.

4. Project Plan

Figure 1 and Table 1 describe our objective, milestones and duration in days.

Figure 1. Timeline of the Project

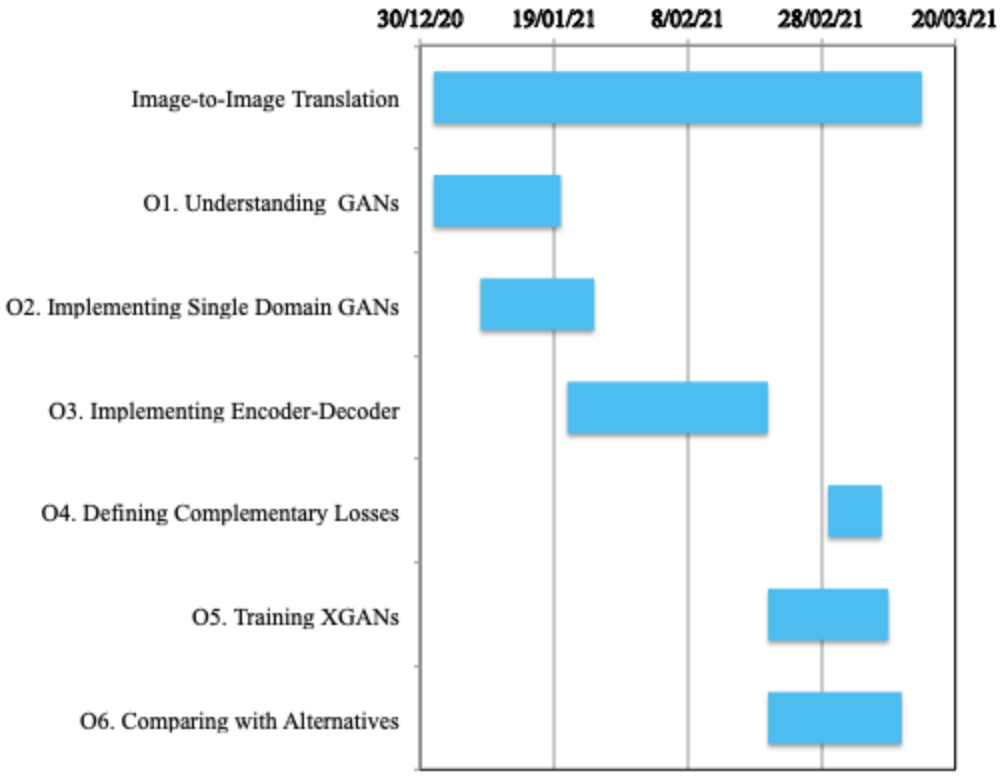


Table 1. Objectives and Milestones

OBJECTIVE	MILESTONE	START	END	DURATION in days
Image-to-Image Translation	**XGAN implementation**	**1/01/21**	**15/03/21**	**73**
O1. Understanding GANs	Understanding GANs	1/01/21	20/01/21	19
O2. Implementing Single Domain GANs	Generating single domain images	8/01/21	25/01/21	17
O3. Implementing Encoder-Decoder	Encoder-Decoder cross-generation	21/01/21	20/02/21	30
O4. Defining Complementary Losses	Definition of complementary losses for quality	1/03/21	9/03/21	8
O5. Training XGANs	Training XGANs	20/02/21	10/03/21	18
O6. Comparing with Alternatives	Comparison with alternatives	20/02/21	12/03/21	20

5. Implementation

Although our project could only consisting of a single implementation (XGAN architecture), we also created a new dataset from scratch (based on an existing dataset), we formatted the cartoon set data as well using a variety of technologies that helped us to execute our code in local and in Google computing engine machines provided to us with GPUs.

5.1. XGAN referenced paper

This project has been developed from the *XGAN: Unsupervised Image to Image Translation for Many to Many Mappings* [1] paper, contributed to the literature by Google Brain at London in 2018. As we can read from the title, the paper proposes a way to perform unsupervised image-to-image translation across two domains using a model they called XGAN. It can be understood as an extension of traditional GANs. The first reference we found about this paper was on the Cartoon Dataset website, also from google[1]. The main purpose of the paper is to generate physically similar cartoon faces starting from human faces, and vice versa. Our project implementation is based on this paper although some slight changes have been made to the original contribution, which allowed us to better manage the time in our execution, based on the experience and knowledge that we gained through it.

5.2. Cartoon faces dataset

Our project idea began when we found the Cartoon set, this dataset consisted of 100k (or 10k if we choose the small version) images of cartoon faces. Since this dataset was created directly for training, images were cleaned and centered. In order to improve our results, as explained in our experiments, we decided to crop all the dataset images to remove unnecessary white pixels on the background and to focus on the relevant pixels. This also allowed us to increase the size of the input image feed to the network , which helped the network to produce better-looking results.

5.3. Real faces dataset

Since our objective was to go from one domain (real) to a different domain (cartoon), we also needed a dataset of real human faces. XGAN authors propose to use the VGG Face dataset.

[1] Publicly available at https://google.github.io/cartoonset/download.html

We explored this possibility, but we noted that this dataset is not centered, nor cropped, nor cleaned. Having this in mind, we decided to search for a dataset that had these characteristics but we did not manage to find one. We finally opted for using the CelebA dataset[2] as it contained celebrity faces centered but not cleaned (they have different noisy backgrounds). In order to have better results we decided to remove the background from the images using an online API service called RemoveBG[3], since all background removal services have a direct cost associated with them for a batch of images, we developed a small Python script with key rotation that helped us do this job. We managed to gather a set of 1k images centered, cleaned and cropped to the same size that we later used for our experiments.

5.4. Technology stack

As mentioned in previous sections, we worked mainly on Python with TensorFlow 2.4 with Keras and we worked with Google Cloud VM to execute the final version of our implementation. Moreover, as provided by this postgraduate, we used the Google Cloud coupon codes to perform executions of our XGAN implementation. We need to resort to cloud-computation because data-handling in XGAN was considerably more difficult than in single-domain GANs. With this coupon there wasn't enough credit to include a GPU in the VM. However, we requested it and the Google Cloud support team quickly granted us one with no additional fees. Through SSH access, the code could be cloned and tested.

6. Results of the experiments

In this section we present the main experiments that we conducted. We present our starting hypothesis in each case, the description of the experiment, our main results and a section with discussion and conclusions. We also describe further experiments that we did not conduct (or document) in the sake of time and space, but that we envisioned while executing the project.

6.1. Single domain GAN to generate cartoon faces

Before getting our hands dirty with the XGAN paper, we decided to start with a simpler implementation of a traditional single domain GAN to generate cartoon faces from a latent random vector. This experiment might seem a bit simple, but for us it was not. Indeed, it was

[2] Publicly available at http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html
[3] Available at https://play.google.com/store/apps/details?id=bg.remove.android&hl=es_PE

our first time using the cartoon dataset, the first time we were implementing a whole experiment from scratch using TensorFlow and finally it was also the first time we designed and trained a GAN model.

6.1.1. Hypothesis

Our hypothesis was that if we really understand the dataset in our hands (i.e. the cartoon dataset) and we also understand how generative adversarial networks work in practice, we should be able to develop a GAN able to generate some decent cartoon faces. Also in the process we should learn some knowledge about our dataset, GANs and TensorFlow that would lead us to better results on the following experiments (XGAN).

6.1.2. Experiment

Our experiment began reading the documentation about GANs on Tensorflow [3] and also revising the lecture we had related to GANs[4] . The experiment was developed using Google Colaboratory because it was the easiest way to conduct it in group, and it was also the same environment we have used in the practical session of this course.

The first step of the experiment was to read the cartoon set dataset [4], in order to see and understand the data and also to apply the preprocessing steps described before, if needed (see Figure 2). To read the dataset we used the `tf.keras.preprocessing.image_dataset_from_directory` method.

Figure 2. Sample of cartoon faces from our dataset

[4] The lecture can be consulted at https://docs.google.com/presentation/d/1_lFVEE2ZHeKN_MvT0SSkGz5HaJvbj22OwYKcTX4E6QU/edit

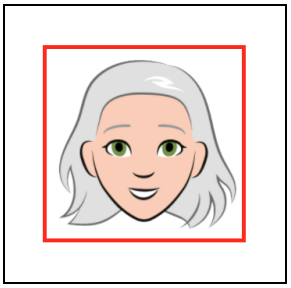
The cartoon set [4] as explained in the section that describes the dataset of this report, is ready for use. Thus, little preprocessing was required. Nevertheless, we apply three steps of preprocessing in order to improve the performance of the model:

- We normalize the tensors into the range -1 and 1. The keras method for reading the dataset converts the images into tensors of shape `(64, 64, 3)` and values from 0 to 255. We normalize the values into a range of -1 and 1 by applying:

```
(image - 127.5) / 127.5
```

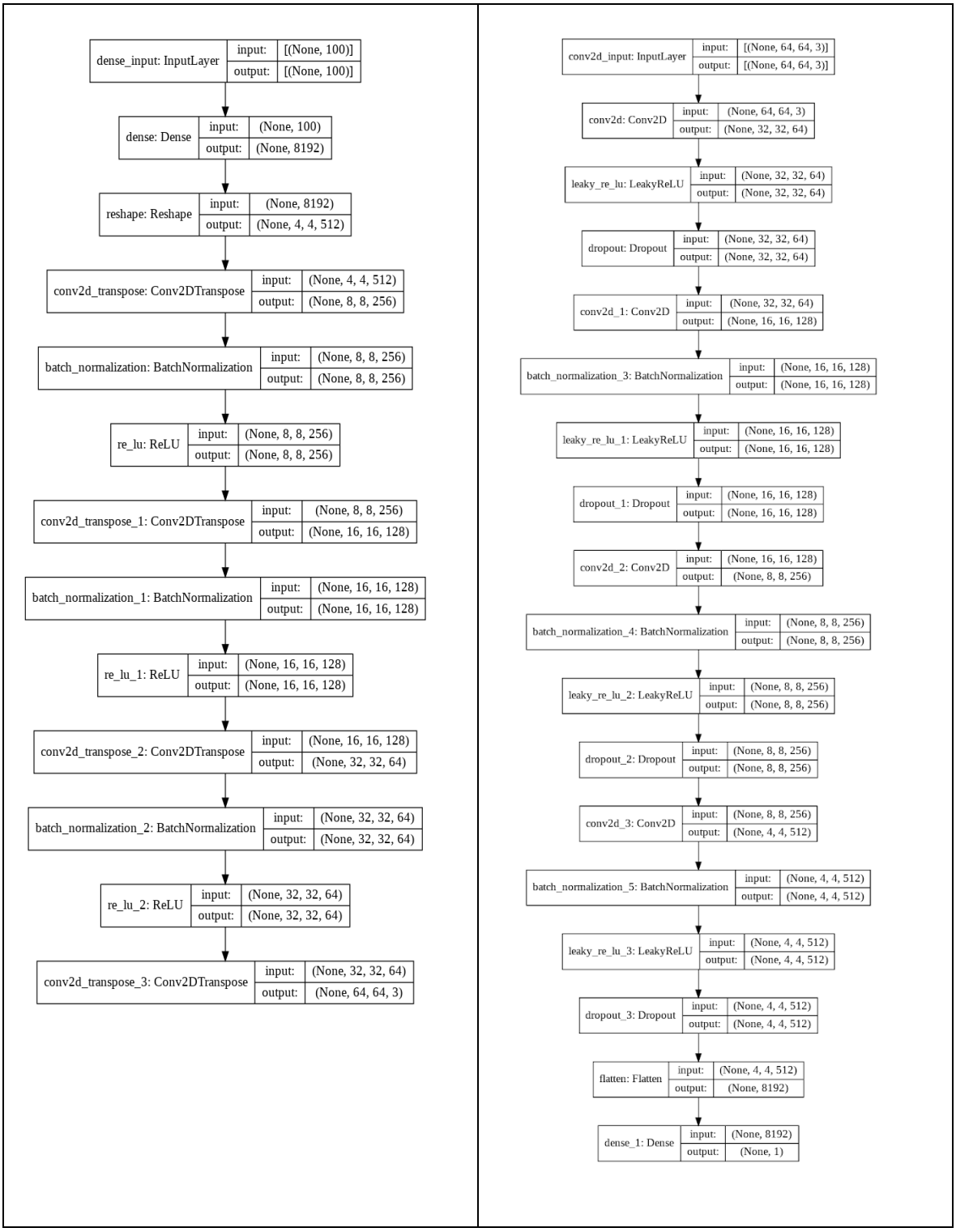
- We cast the values of the tensor to `tf.float32`.
- We crop the image to the center (see Figure 3). This step proved very important since we managed to reduce the white space around the faces and to focus on the relevant pixels within the image.

Figure 3. Sample of cropped image



The second step was to design and train the GAN model. On the one hand, we have the generator which reads a random noise (tensor of shape 100) and generates a cartoon face. On the other hand, we have the discriminator which reads an image (tensor of shape `(64, 64, 3)`) and returns 1 or 0, depending on the original input image. The first version of our generator had the standard configuration of layers, used in Tensorflow documentation [3], but we realized that the better results were obtained when we used the network architecture presented in the left panel of Figure 4. Some of the changes we made to the network were inspired by the Github project *How to Train a GAN? Tips and tricks to make GANs work* [5]. For the discriminator architecture we followed the same learning path as we did for the generator, and the best results were associated with the network architecture on the right panel of Figure 4. Both models were implemented using the `tf.keras.Sequential` method and wrapped together into a Python class `tf.keras.Model`.

Figure 4. Generator (left) and Discriminator (right)



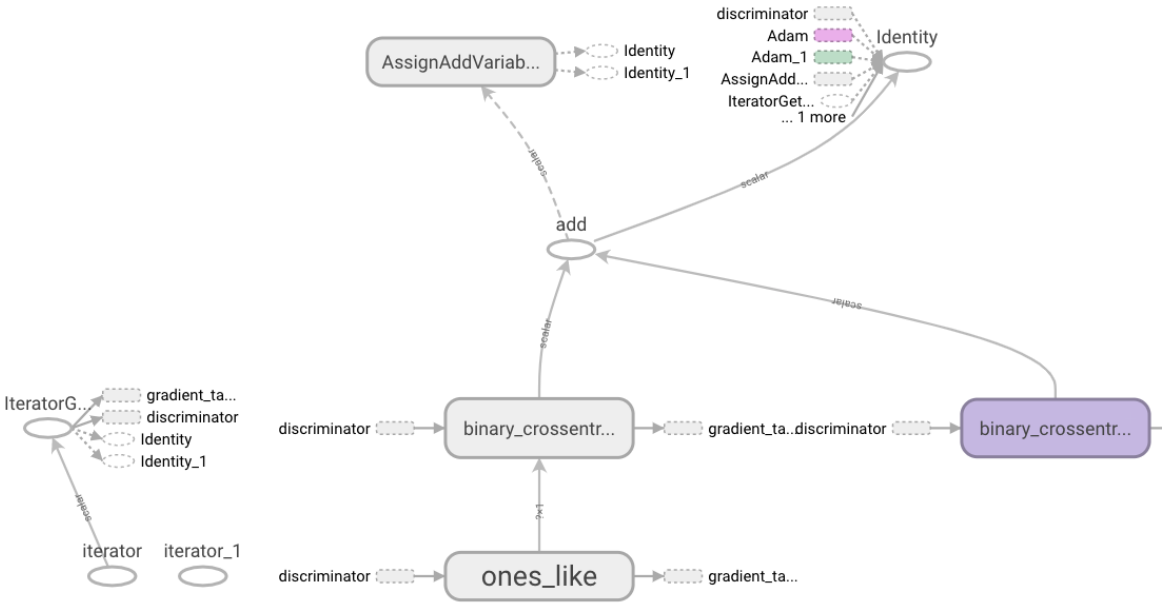
Since the generator and the discriminator are two separate models, they have two different optimizers and two losses that need to be defined. For the optimizer we use the standard Adam optimizer. On the one hand, the discriminator loss quantifies how good the model is distinguishing real images from generated fake images. We applied the Binary Cross Entropy loss to measure how many of the real images were correctly classified and how many of the fake images were correctly classified. On the other hand, the generator loss quantifies how well the model was tricking the discriminator. We applied the Binary Cross Entropy Loss in this case too, to measure how many fake images from the generator were classified as real images.

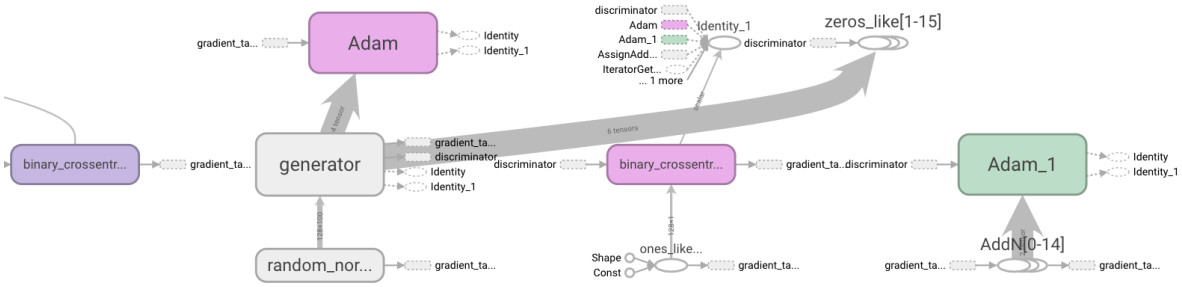
We trained our GAN using a noise vector of shape 100 and during 300 epochs. The goal of the training process was to minimize the generator (images from the generator are classified as real) and the discriminator losses (images are classified correctly).

6.1.3. Results

In order to get a general picture of the final implementation, we observed the generated graph by TensorFlow, mainly to make sure that everything was implemented in the way we expected (see Figure 5).

Figure 5. TensorFlow Graph of Single Domain GAN



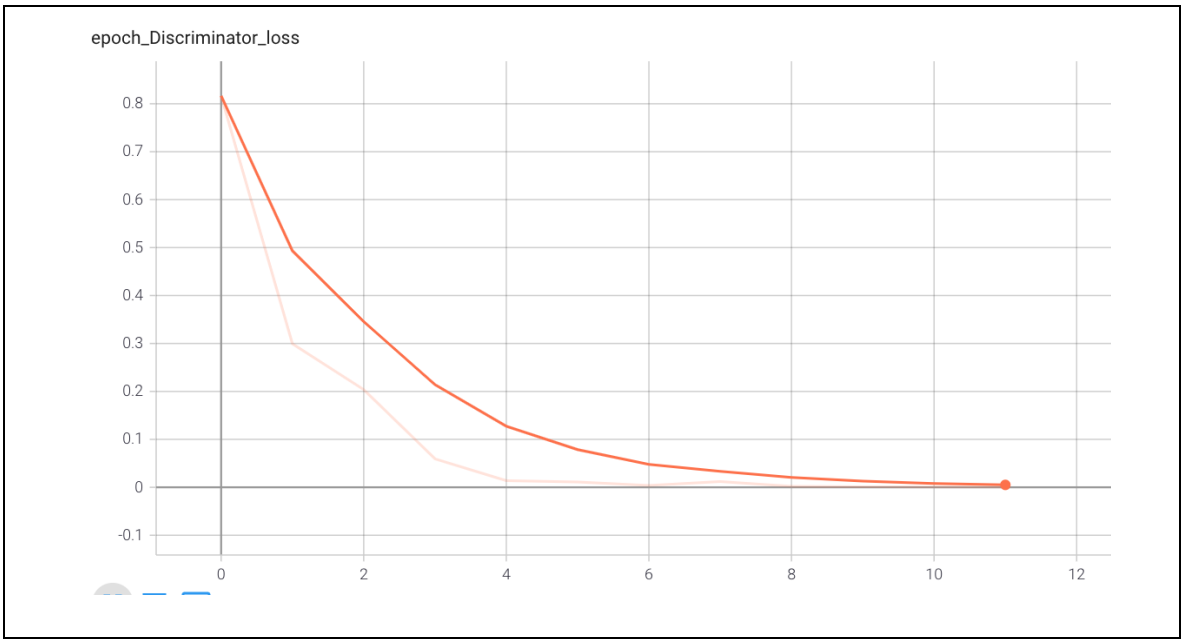


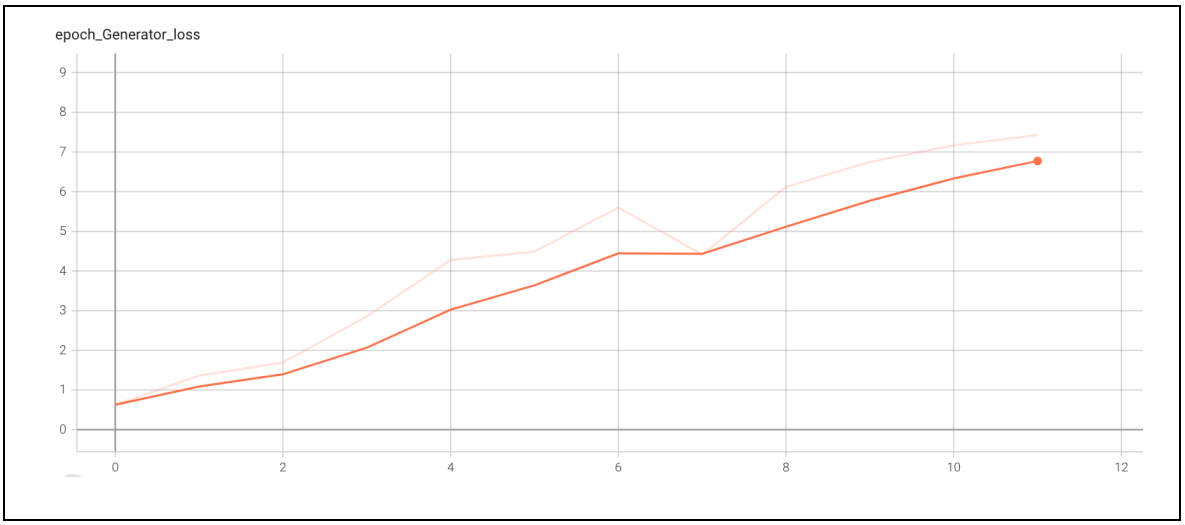
The results of our execution are documented in Figure 6. We document the three main experiment results in this figure which consists of two panels in each case, the evolution of the losses while training and the evolution of the cartoon faces.

Figure 6. Results of The Experiments- GAN implementation

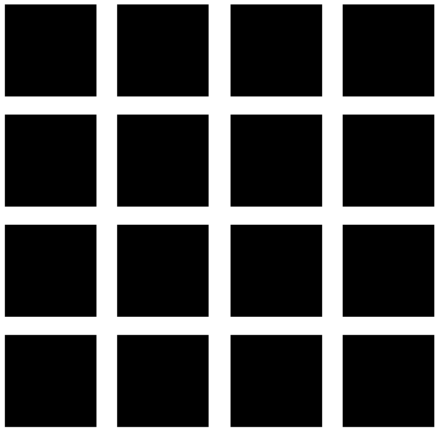
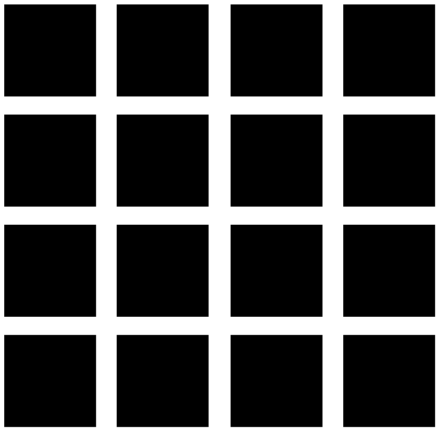
Experiment 1: 100 epochs - lr 0.0002 - betas = (0.5, 0.999)- noise_size = 100 - batch_size = 128 (without normalizing the data into -1 to 1 range)

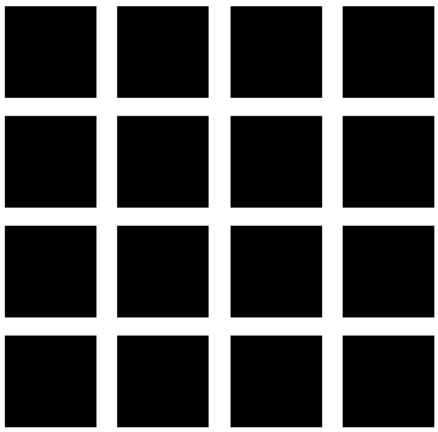
Panel A. Evolution of the losses while training





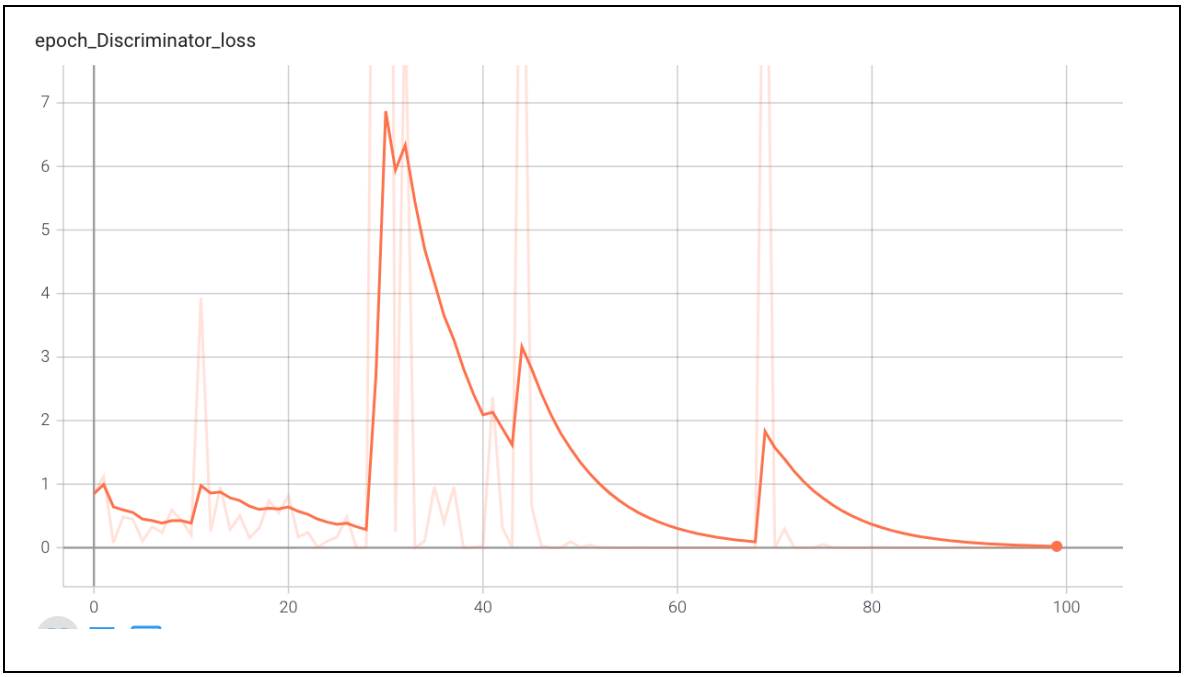
Panel B. Evolution of the generated cartoon faces while training

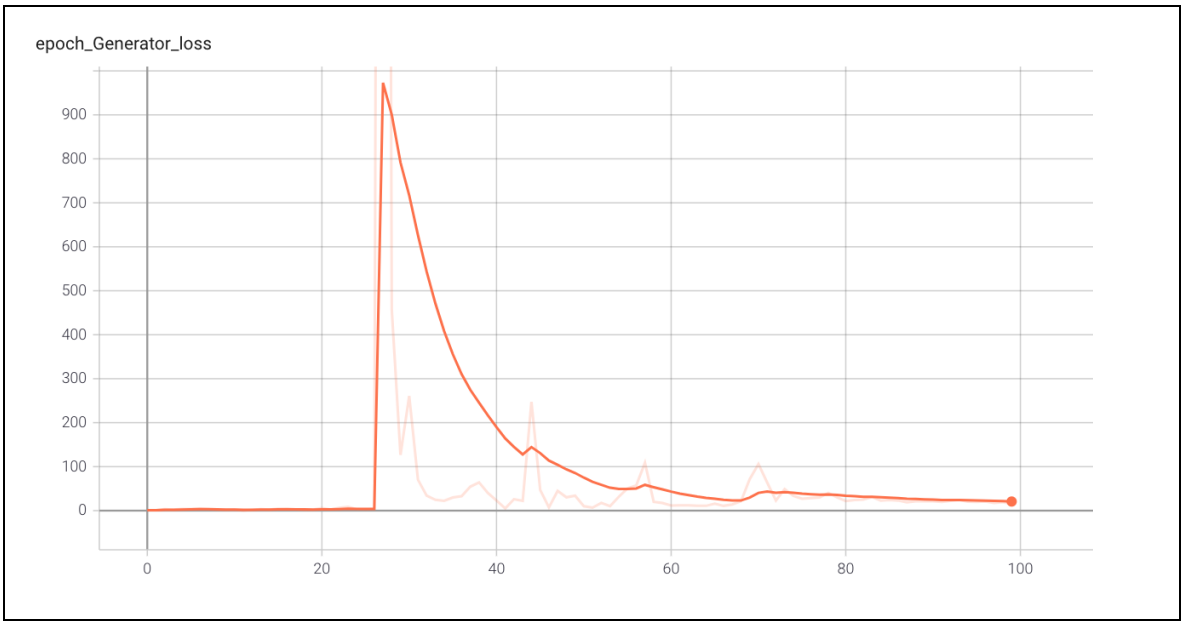
	
epoch 0	epoch 5

	
epoch 10	

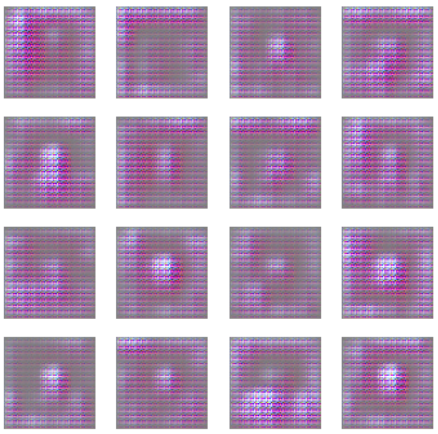
Experiment 2: 100 epochs - lr 0.001 - betas = (0.5, 0.999)- noise_size = 100 - batch_size = 128

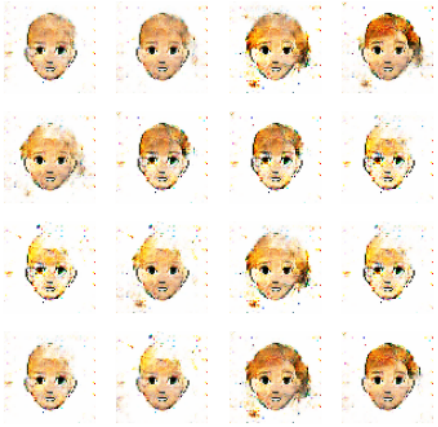
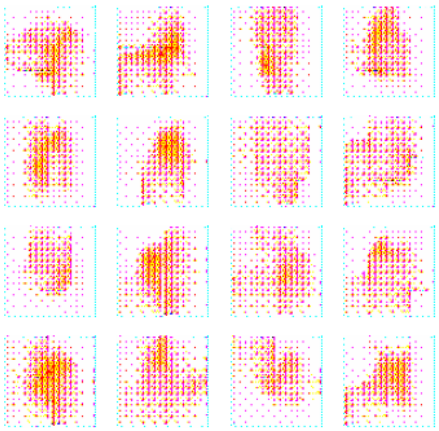
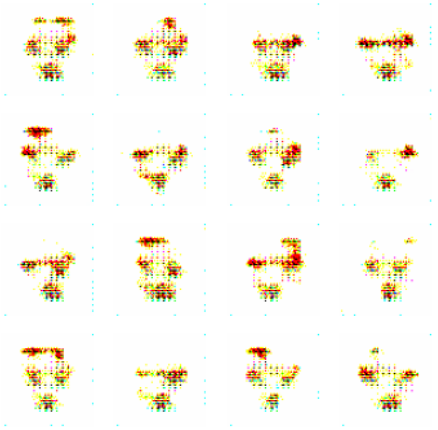
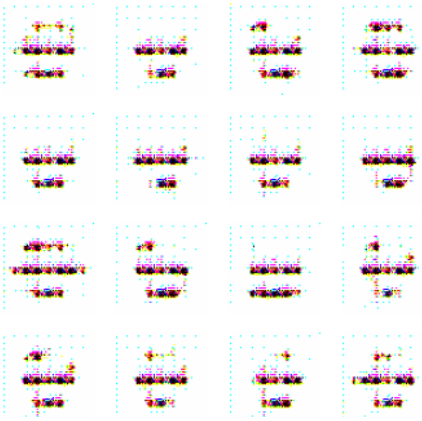
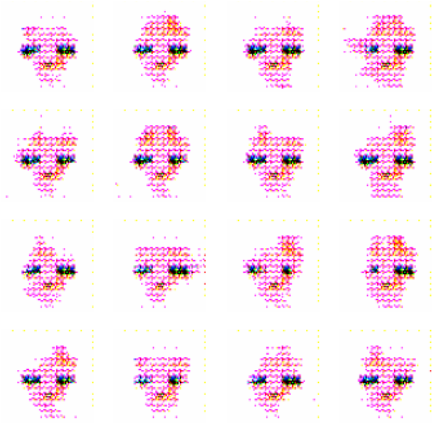
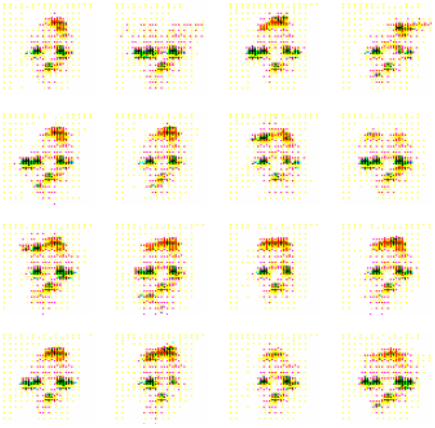
Panel A. Evolution of the losses while training

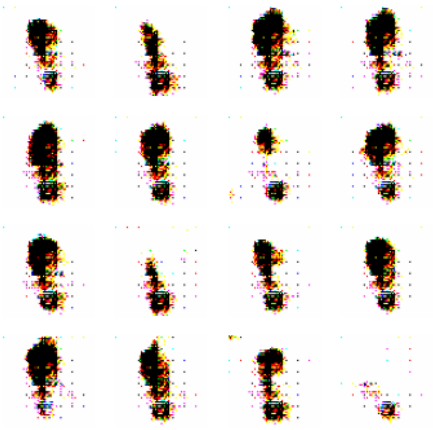
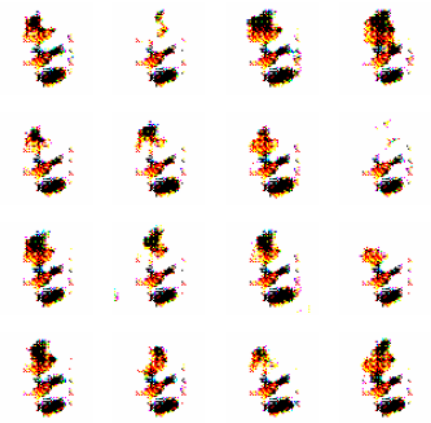
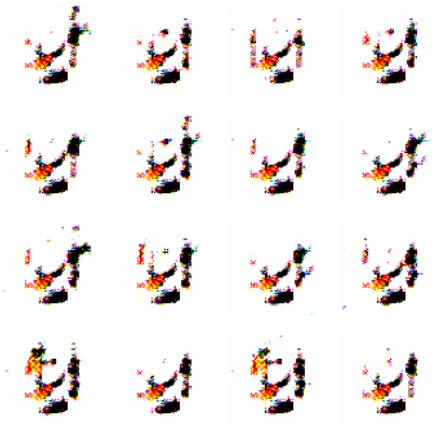




Panel B. Evolution of the generated cartoon faces while training

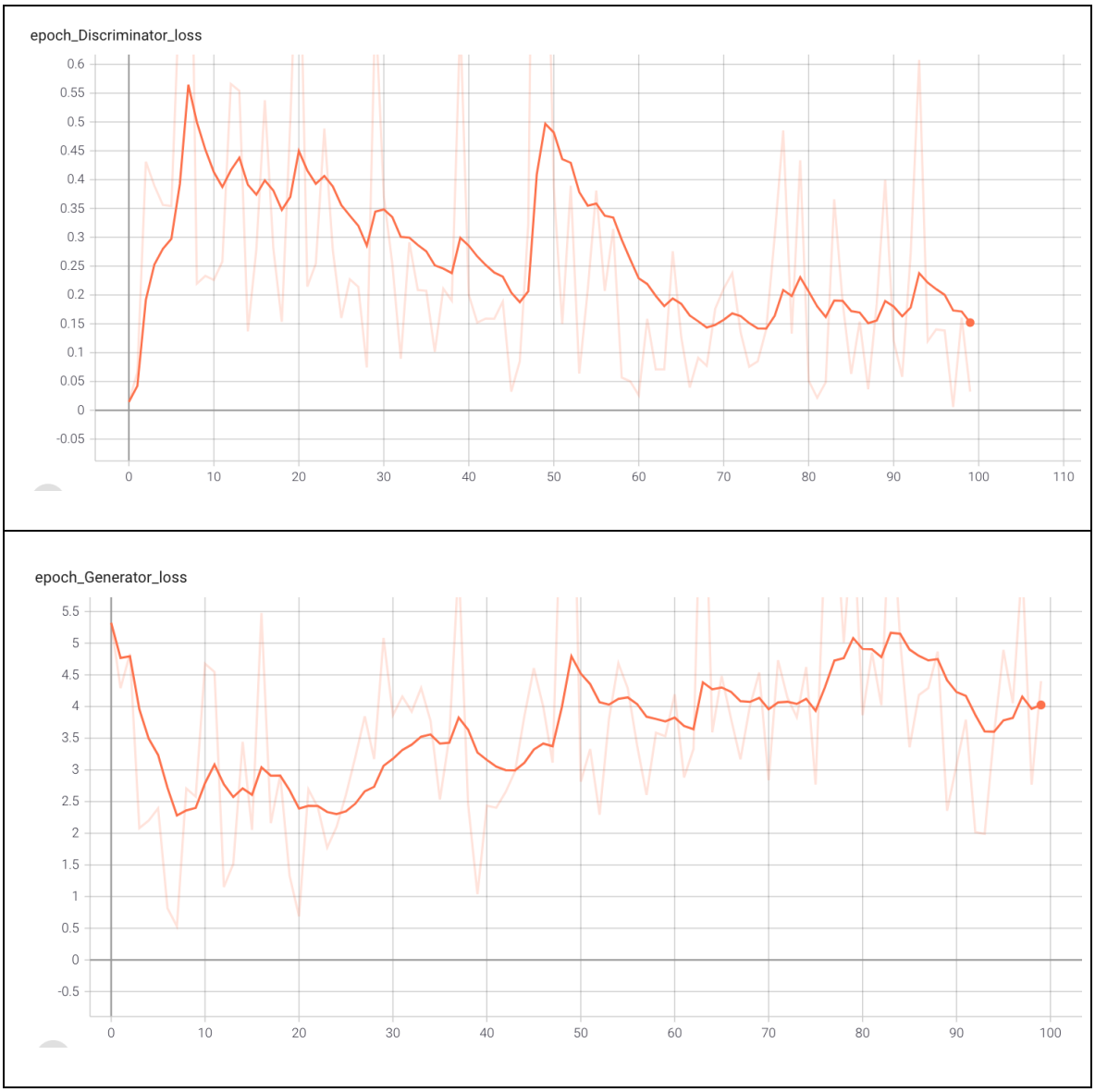
	
epoch 0	epoch 10

	
epoch 20	epoch 30
	
epoch 40	epoch 50
	

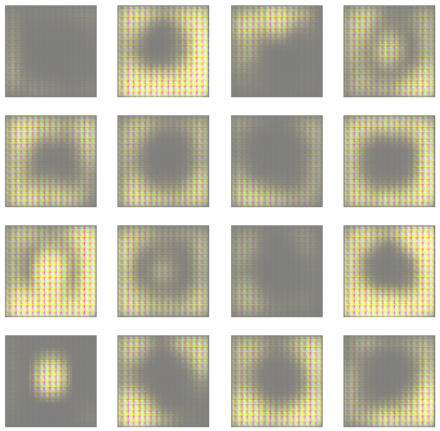
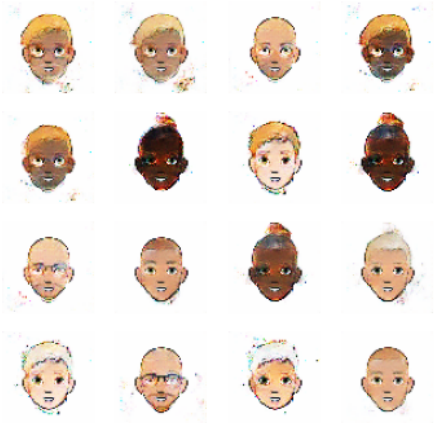
epoch 60	epoch 70
	
epoch 80	epoch 90
	
epoch 100	

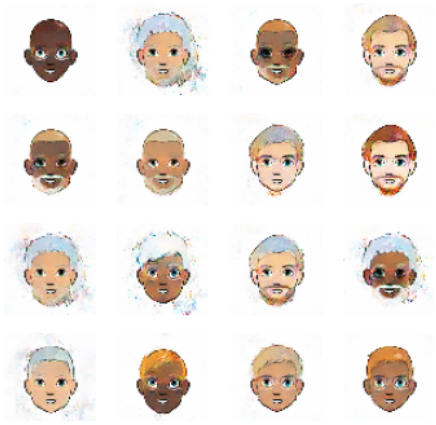
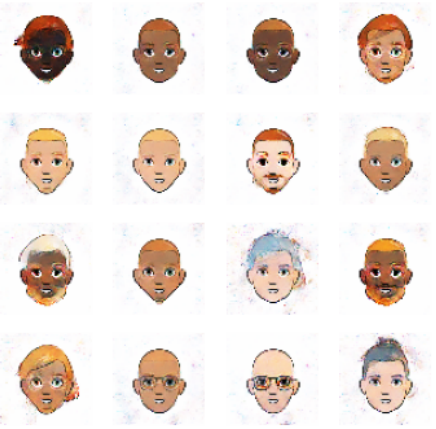
Experiment 3: 100 epochs - lr 0.0002 - betas = (0.5, 0.999)- noise_size = 100 - batch_size = 128

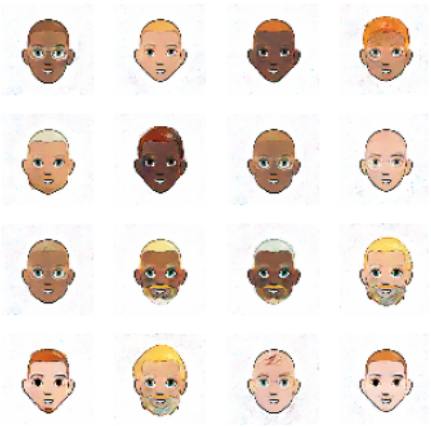
Panel A. Evolution of the losses while training



Panel B. Evolution of the generated cartoon faces while training

	
epoch 0	epoch 10
	
epoch 20	epoch 30

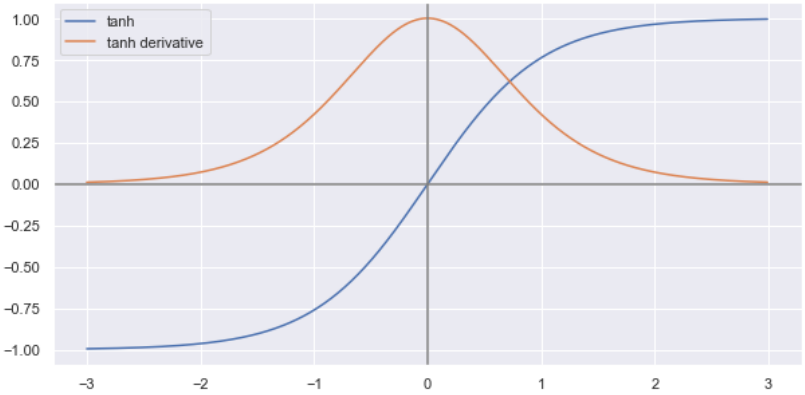
	
epoch 40	epoch 50
	
epoch 60	epoch 70
	
epoch 80	epoch 90

	
epoch 100	

6.1.4. Conclusions

First, we would like to emphasize that we understood the importance of preprocessing the input date. In our case, normalizing the data speeded up the training, led to a faster convergence and, in general, to better results. As we explained in the experiment subsection the changes we made to the network were inspired by the *How to Train a GAN? Tips and tricks to make GANs work* [5] document. In order to understand the why behind the improvements, we found a paper from Timo Stöttner [6] which explained the improvements of normalizing the data into a -1 to 1 range, together with using the Tanh activation function.

Figure 7. Tanh activation



The generator uses the Tanh function as the activation method of the last convolutional layer. As we can see on Figure 7, the tanh activation function has a mean of 0 and takes only values between -1 and 1. The tanh derivative is also centered at 0. As we can observe in our results, training the GAN without normalizing the data derived into black images. If we observe the plot of the discriminator and generator loss (Figure 6, Panel A, First experiment), we realize that the black images were a consequence of the discriminator learning to a faster pace than the generator. It turns out that, at the end, the discriminator knew very easily when an image was coming from the generator or from the original dataset because, because the generated images were always black. This reflects the well-known fact that in order for GANs to perform well the generator and the discriminator need to be trained simultaneously at a similar pace. That is the main reason why Generative Adversarial models are known to be unstable and hard to train, and with this experiment we could confirm this fact by ourselves.

The other preprocessing step was to crop the image as explained in Figure 3. This step really improved the quality of the cartoon faces coming from the generator. The main reason for this improvement was that lacking the center crop, the generator believed that it was sufficient to predict the background of the image to white, to achieve its objective. Indeed, it was doing quite well (as there were a lot of white parts), but the face was too blurred. By removing a lot of the white background with the center crop, we forced the generator to learn more details of the cartoon face, while in the training phase.

As we can see on the results section, we obtained the best results using a learning rate of 0.0002. In Figure 6 we can see how good the generator is to produce cartoon faces at the epoch 100. With a higher learning rate, such as 0.001, we can observe that the networks learn faster at the beginning (i.e. they generate decent cartoon faces sooner) but at some point they diverged and got lost. The reason for this could be we were doing steps too big while applying gradient descent, so that when the network was getting closer to the minimum it jumped into another point. Here we realized that using a learning rate scheduler could be a great option, but we decided not to implement it and move on with the next experiment, in order to follow the timeline of the project.

Finally, we would also point out that putting the generator and discriminator together in a Python class which inherits from the `tf.keras.model` was something that we learned throughout this experiment and it was very useful. It really simplified the training process and made easy the process of logging the results into the Tensorboard using the callbacks.

6.2. Domain to domain translation - XGAN

After finishing the latter experiment of designing and training a single domain GAN, we were ready and we had the knowledge to succeed in building the XGAN model.

6.2.1. Hypothesis

Our hypothesis for this experiment was that by implementing a paired cross- encoder-decoder structure linked through a semantic consistency loss is possible to generate cartoonized faces that preserve the attributes of human original faces.

As advised by our professors, we kept in mind the difficulty in this experiment as it was a complex task to solve.

6.2.2. Experiment

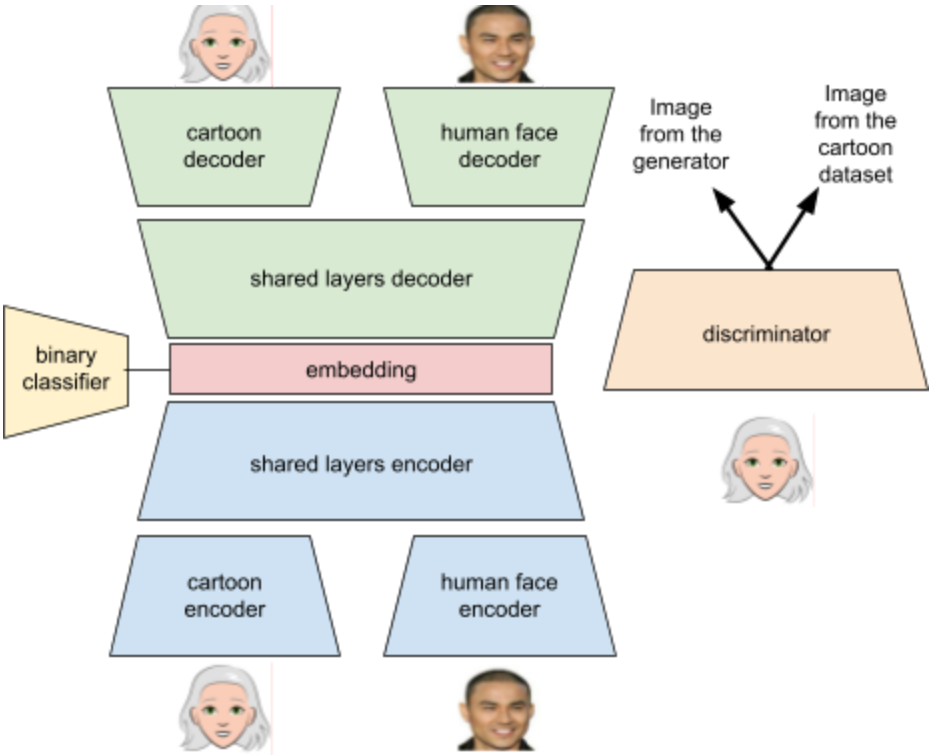
To begin with this experiment, we needed two datasets sharing the same semantic content (human faces) in two different domains: real human faces and cartoon faces. We used the real face dataset as the starting domain, and the cartoon dataset which (also used for the single domain GAN), as the target domain.

We applied the same preprocessing to both datasets, as described in the subsection 6.1. Namely, we normalized image tensors to force values within the range -1 to 1. Once the data was normalized, we applied a center crop to avoid having a lot of white background.

The XGAN model is a Generative Adversarial Network formed by a generator, a discriminator and a binary classifier. The difference is that the generator instead of having a random noise tensor as input, has an image from the starting domain. The output is supposed to be an image of the target domain. On its side, the discriminator is the same that we had in the single

domain GAN experiment. Figure 8 represents the XGAN model (the generator is on the right and discriminator on the left).

Figure 8. XGAN graphical representation



If we take a closer look at the autoencoder that acts as the generator, we can see how both, encoder and decoder, have some shared layers and some others which are exclusive to the domain they belong to. The autoencoder should encode an image from one domain into an embedding of size 1024, and decode it into two images (one for each domain).

As explained before, the image-to-image mapping is unsupervised. For that to be possible the generator autoencoder consists of three losses:

- *The autoencoder loss.* This loss makes sure the encoder - decoder flow is kept. The logic behind the loss is that if we feed a cartoon face to the encoder, we should recover the same exact image from the decoder cartoon domain as output. The same reasoning applies for real face images. This loss is represented in the XGAN paper [1] as follows:

$$\mathcal{L}_{rec,1} = \mathbb{E}_{\mathbf{x}\sim p_{\mathcal{D}_1}} \left(\|\mathbf{x} - d_1(e_1(\mathbf{x}))\|_2 \right), \text{ likewise for domain } \mathcal{D}_2$$

- *Domain adversarial loss.* Following the original paper, this loss forces the embeddings encoded from both domains to lie in the same subspace. The model should achieve this by training a binary classifier to identify when an embedding is encoded from one or another domain. It is represented as:

$$\mathcal{L}_{dann} = \mathbb{E}_{p_{\mathcal{D}_1}} \ell(1, c_{dann}(e_1(\mathbf{x}))) + \mathbb{E}_{p_{\mathcal{D}_2}} \ell\left(2, c_{dann}(e_2(\mathbf{x}))\right)$$

- *Semantic consistency loss.* This loss is the most important of the autoencoder generator, since it forces the model to learn the semantic self-supervision. The generated cartoon face should be physically similar to the real face image. To achieve this, the embedding generated from a real face, should be equal to the embedding obtained from the cartoon image generated by the same image when it is introduced back to the encoder. On the paper this loss is represented by the following equation:

$$\mathcal{L}_{sem,1\to 2} = \mathbb{E}_{\mathbf{x}\sim p_{\mathcal{D}_1}} \|e_1(\mathbf{x}) - e_2(g_{1\to 2}(\mathbf{x}))\|, \text{likewise for } \mathcal{L}_{sem,2\to 1}.$$
$\|\cdot\|$ denotes a distance between vectors.

As it can be observed the semantic consistency loss and the domain adversarial loss are quite the opposite. On the one hand, the semantic consistency loss aims to optimize the autoencoder to have the same embedding, the real face image and the cartoon image generated from the previous image. On the other hand the domain adversarial loss tries to distinguish when an embedding comes from one domain or from another. The purpose of this second loss is to make the semantic consistency loss better. The autoencoder generator total loss will be the sum of those three losses.

As explained before the decoder of the XGAN is the same we had in the previous single domain experiment and as a consequence, the discriminator loss is also the same. In our context, the discriminator loss employs binary cross-entropy over how many cartoon images were correctly classified, depending on the image source, the generator and the dataset.

Both the generator and the discriminator will use the standard Adam optimizers at the training phase.

Due to the complexity of the model, we follow the following steps to develop the XGAN:

- Developing the encoder.
- Developing the decoder.
- Developing the small binary classifier.
- Puting the encoder, the decoder and the binary classifier together into the same Python object which will represent the generator.
- Adding the loss functions to the autoencoder following the XGAN paper.
- Adding the discriminator of the previous experiment and the loss function.

6.2.3. Results

Figure 9. Results of The Experiments- XGAN implementation

Panel A. Evolution of the generated cartoon faces while training

Note: lr 0.002 - betas = (0.5, 0.999) - batch_size = 16

	
Epoch 20	Epoch 60

epoch 100

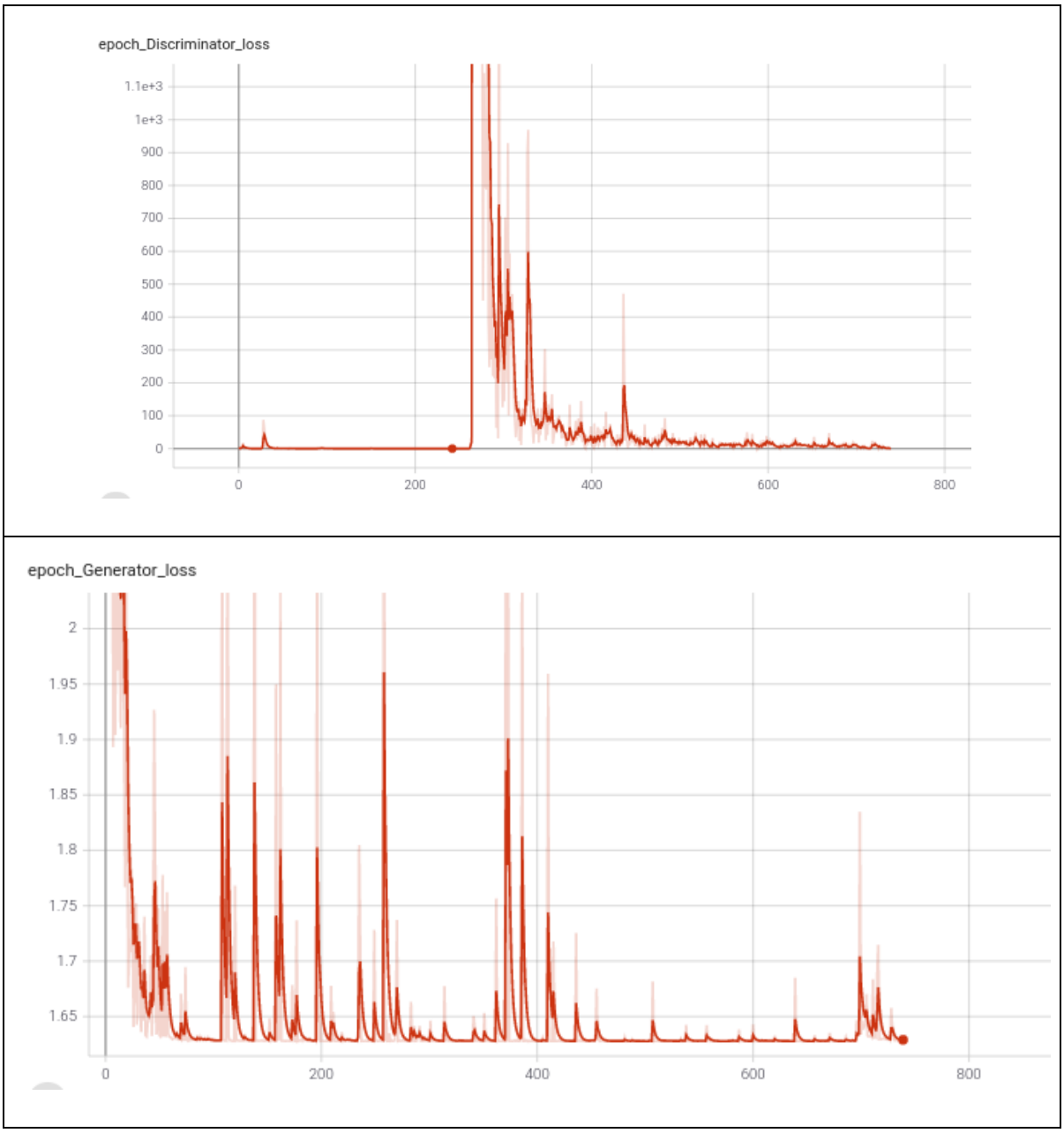
epoch 200

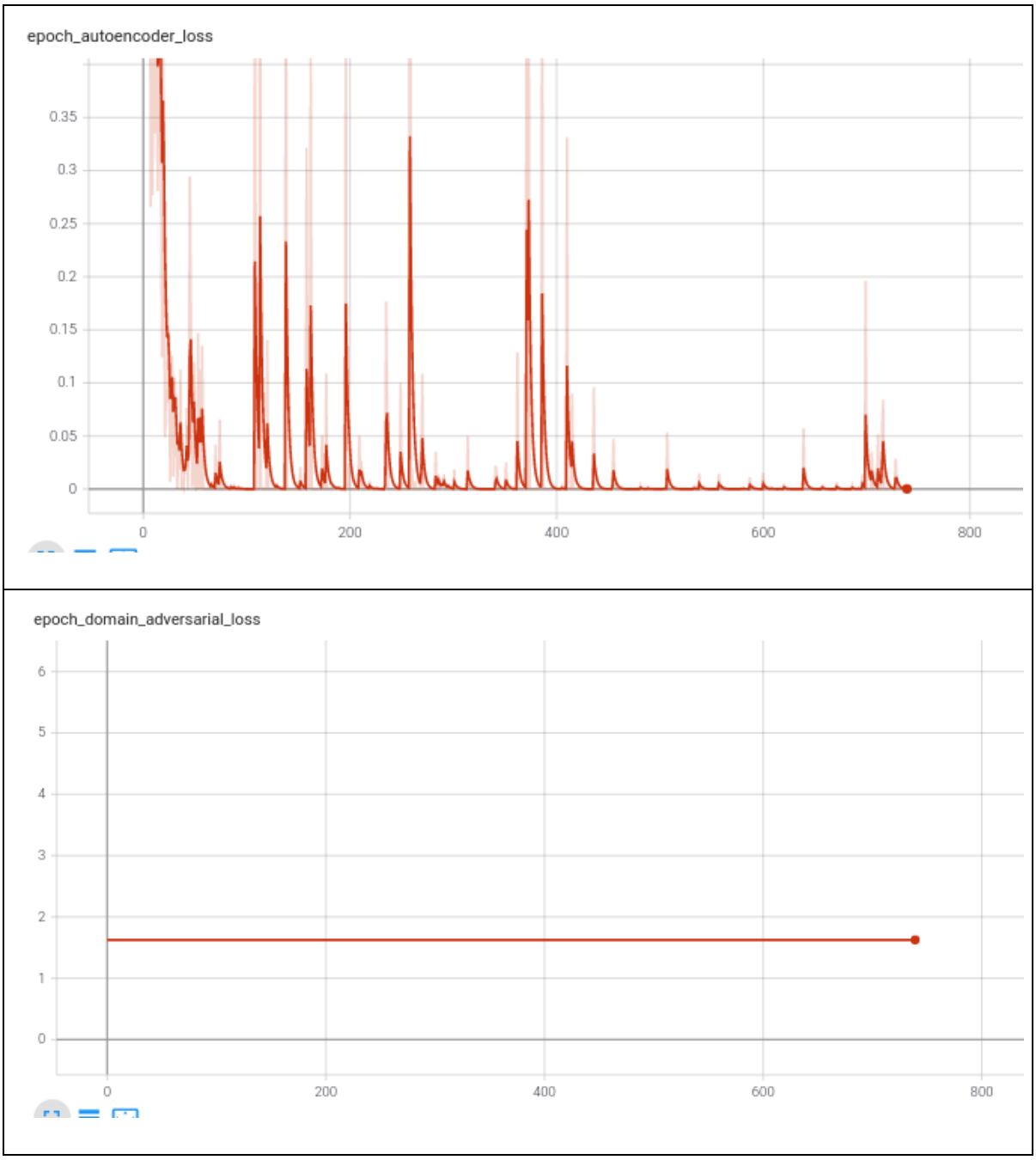
epoch 211	epoch 300
	
epoch 400	epoch 500

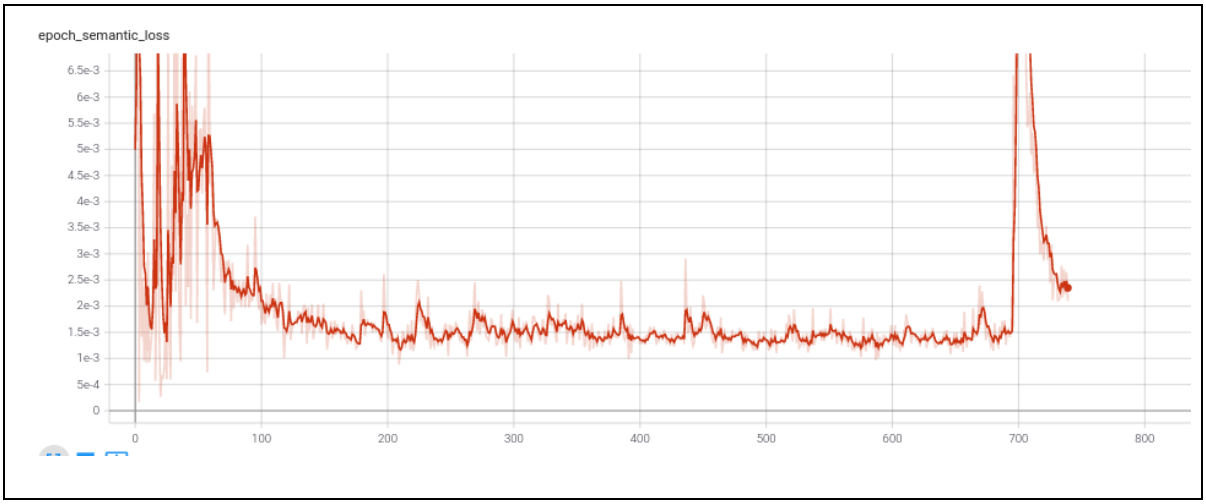
	
epoch 600	epoch 700

Panel B. Evolution of the Losses

Note: The Figure shows the evolution of the losses from epoch 0 to 760. The horizontal axis is relative time from the start of the execution, while the vertical axis is the magnitude of the loss.




epoch_autoencoder_loss
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
200
400
600
800
epoch_domain_adversarial_loss
6
5
4
3
2
1
0
0
200
400
600
800

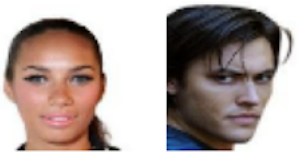


6.2.4. Conclusion

We would like to emphasize, one more time, the importance of the input data and correct preprocessing of it, as to get a ready-to use set images for training. As explained in the dataset section, it has been a challenge to have one-thousand images of human faces correctly sized, cleaned and cropped. Indeed, we realized that the unbalanced sample of our domain datasets could have kept us from having the expected results that we aimed at the beginning of our project.

One of the first things that can be inferred from our results is that the human real face images in our original data are not ideal, in the sense that they frequently are not fully frontal, and this significantly impacts our results. If we observe the images in Figure 9, we will see how the generator is failing when the real human face image is not centered in the same way the cartoon image is.

Figure 9. frontal versus rotated faces



We realized that a way to solve this could have been to apply oversampling techniques to the cartoon dataset in order to have images which are a bit rotated and shifted from the center (as we have with the real face dataset by default). Since the time of the project was limited, we decided to invest our time in other parts of the project.

When we analyzed the plot of all the loss functions, we realized that some conclusions could be extracted (Figure 9, Panel B). The dynamics of the different losses in our model correspond with the patterns observed in the generated faces. For instance, it can be observed that the generator starts producing very similar faces at the beginning or the execution, while the loss is flat, and it is only after a bunch of epochs that we can observe generated faces of increased quality (as to trick the generator) and therefore the loss starts to consistently reduce.

Although we manage to generate cartoon faces starting from human real faces, we acknowledge that our results still could be improved. For instance, there is room for improvement regarding the complementary losses to include in our architectures. We decided to work only with the traditional adversarial loss, the reconstruction loss and the semantic consistency loss, even though in the original study two additional losses were considered, namely, the "GAN Objective loss" and the "Teacher loss". The two losses are optional. The former is supposed to increase the sample quality, while the latter aims to incorporate prior knowledge into the model. The inclusion of these two losses would be definitely a follow-up option to improve the quality of our results, which we did not push forward due to the time constraints of our project.

In our single-domain GAN implementation we could still rely on our CPUs, while for the implementation of our XGAN, which was feed by two different input datasets (i.e. cartoon and real faces), we need to resort to cloud-Computation and naturally to GPUs. The advantages of GPUs when working with large data sets (i.e. images of relatively high

resolution) over CPUs have been extensively documented in the field and mentioned during the lectures as well. They can be tracked to the fact that GPUs provide increased processing power, higher memory bandwidth, and a capacity for parallelism. From our perspective, this latter point is key when training XGANs, which require the simultaneous estimation and re-estimation through SGD of a large set of model parameters corresponding to the generator and the discriminator networks.

6.3. Comparison with other alternatives in the literature

A leading model in the task of image-to-image translation is StarGAN-v2 [3] presented in the conference of *Computer Vision and Pattern Recognition 2020,* so we decided to compare our X-GAN implementation with it. StarGAN-v2 represents a progress over the popular StarGAN [4], as it is a scalable approach able to generate diverse images across multiple domains. This means that for a given input-image it does not produce a unique output-image, but several, which reflect a variety of possible styles in the reference domain. In terms of architecture, although it shares a common objective with our baseline X-GAN model, StarGAN-v2 is very different: i) StarGAN-v2 possesses only one generator and one discriminator (unlike X-GAN that is based on a paired-structure), and ii) instead of including a semantic consistency loss, StarGAN-v2 relies on a *mapping network* and a *style encoder*. The mapping network learns to transform random Gaussian noise into a style code, and the style encoder learns to extract the style code from a source image. Thus, from our point of view it tackles more directly the problem of image-to-image translation, and enhances the possible results with respect to our baseline.

6.3.1. StarGAN-v2: Source cartoon- Reference human

6.3.1.1. Hypothesis:

The aforementioned facts led as to conjecture the following hypothesis: "By directly compressing the problem into a structure of one generator and one discriminator, complemented with a mapping network and a style encoder, StarGAN-v2 can produce more realistic and diverse outputs than XGAN, given a source-image".

6.3.1.2. Experiment

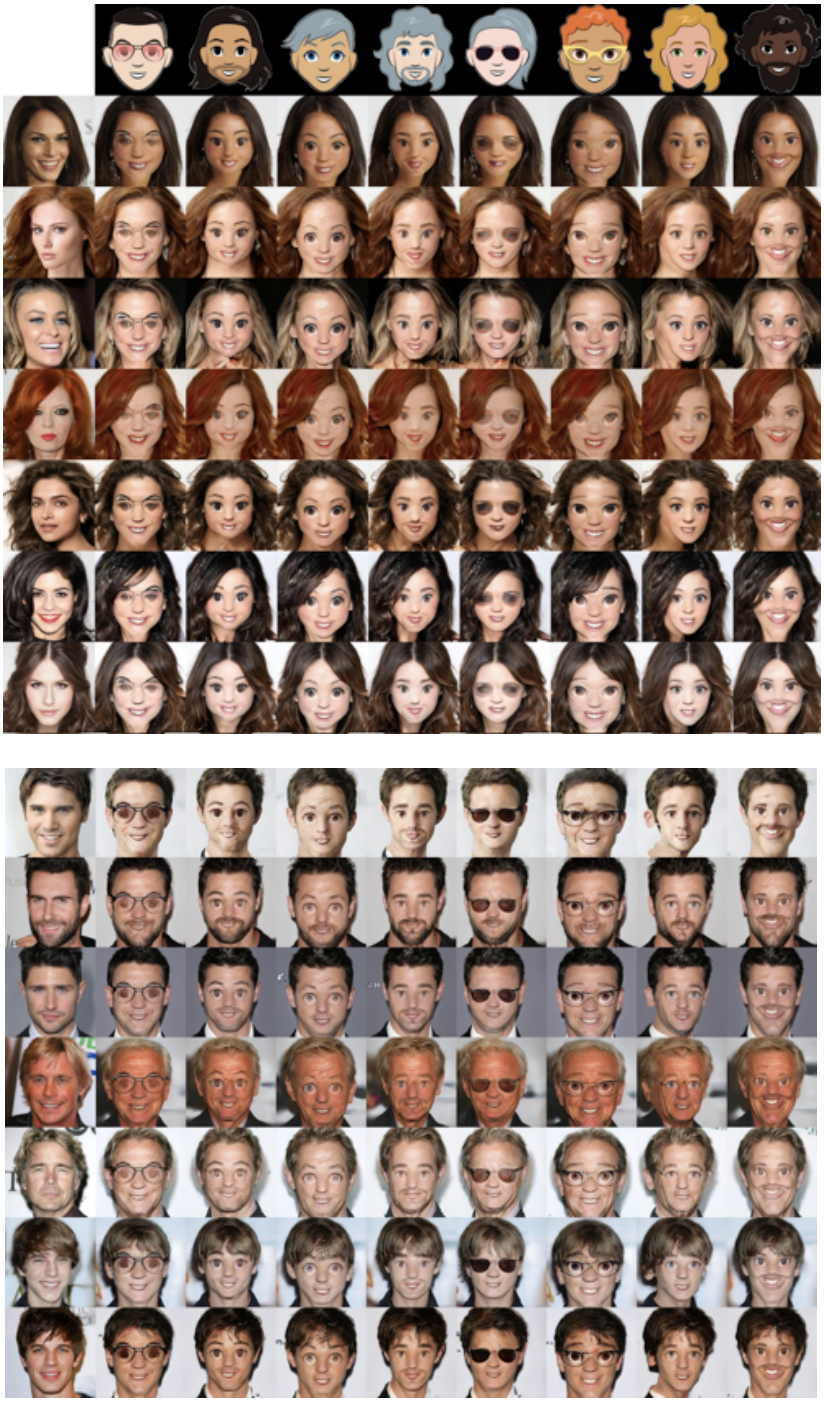
There exist official code and 5 community implementations of StarGAN-v2 available at: https://arxiv.org/abs/1912.01865. We used the official code that contained the pre-trained network using the CelebA-HQ dataset, but we adapted it to generate realistic human images starting from source cartoon faces. We would like to emphasize that StarGAN has not been tried before using as source-images cartoon faces, and it was trained using only human faces from CelebA-HQ.

Before inclusion into StarGAN-v2 we cropped and resized the cartoon faces as we explained in our X-GAN main implementation. StarGAN-v2 official implementation is on Pythorch although there is also an official implementation in TensorFlow. We use the former and we restrict our attention to human faces (there is also a set of animal faces dataset introduced to the field by the authors in the original implementation that we ignored in our experiments).

6.3.1.3. Results:

In Figure 10 we present our results regarding the translation of cartoon faces to human faces with different styles as allowed by StarGAN-v2. Figure # reads as follows. For each source cartoon face at the top of the figure, StarGAN-v2 produces 14 human faces that preserve the style of the (reference) human faces located in the first column of the figure (seven male and seven female faces from CelebA-HQ).

Figure 10.StarGAN-V2 Source Cartoon Reference Human



Note: We run already pre-trained Starv2 model using cartoon faces from our data set

6.3.1.4. Conclusions

From visual inspection, the results of StarGAN-v2 are remarkable, indeed, well above what we got from our main implementation of X-GAN using CelebA-HQ and carton faces. We

concluded that including in the architecture a network specialized in learning styles and an style encoder indeed benefits the approach, as it enhances the possible generated faces (multiple styles from the same cartoon face), and also increases the visual performance associated to each face (it generates outputs that look more "natural"). In fact, even though StarGAN was not trained with cartoon faces, it was able to preserve the general cartoon attributes when it did the domain translation. One point merits further attention. StarGAN crashes when the pose in the reference image is altered even a little bit (row six in Figure ##), so it works best at translating from fully-frontal poses to fully-frontal poses. After conducting this experiment we wondered whether by changing the reference input-images of the model (the first column on the left of Figure ##) we could translate a human face into a cartoon face, which led us to conduct the following experiment.

6.3.2. StarGAN-v2: Source human- reference cartoon

6.3.2.1. Hypothesis

"By changing the reference input-images of the StarGAN-v2 model instead of the source images, we could translate a human face into a cartoon face as is required for a proper comparison with our X-GAN implementation"

6.3.2.2. Experiment

The modeling set-up was the same as in the last experiment, but this time we modified the reference-input, instead of the source-input.

6.3.2.3. Results

In Figure 11 we present our main results from this experiment. Figure # reads as follows. For each source human face at the top of the figure, StarGAN-v2 produces 9 faces that preserve the style of the (reference) cartoon faces located in the first column of the figure.

Figure 11. StarGAN-V2 Reference Cartoon Source Human

Note: We run already pre-trained Starv2 model using selected cartoon faces from our data set

6.3.2.4. Conclusions

This time our results from StarGAN-v2 are directly comparable with our X-GAN results in section 6.2. since we translated from a human-source-face to a carton-reference-face. StarGAN-v2 does not generate proper "cartoon faces" while X-GAN does. In other words, the generated faces from StarGAN still look "humanoid". Some attributes like the hair color, hairstyle, skin color, and color of the background of the cartoon faces are successfully translated into the generated faces while preserving attributes of the person in the source-images. The generated faces look *cartoonized* in the sense that the skin, in most of the cases, does not contain wrinkles or other human expressions, but it can be said that they are still human. This is not really surprising since the pertained- network that we use as a starting point did not include cartoon faces, so it does not really know how to properly translate from

the human to the cartoon domain. Indeed, it does a very good job considering such a limitation.

Unexpectedly, other features of the images (in both the source and the reference dimensions) such as dark skin seem to be largely underrepresented in the original data set of CelebA-HQ as well. We know this because it is evident that the network is better translating white skin into the generated images than dark skin. Let us compare the results in Figure 11 in rows 5 and 7 with the results in rows 8 and 9. In the former cases the skin color is almost perfectly translated, while in the latter cases it is "whitened". This issue is related to the problem that DL models have when dealing with atypical observations, consisting of relatively few examples in the dataset. In this case, we hypothesize that it would be convenient to artificially increase the proportion of darker-skin faces in the human CelebA-HQ dataset used to train the network. In other words, by artificially inducing a higher proportion of darker-faces in the dataset we would force the network to translate this attribute better. We did not push forward this experiment because it would imply training the StarGAN again (or to use some kind of knowledge distillation) and we ran out of time. But the path to follow is relatively straightforward: we would create a new data set for celebrities removing a significant proportion of faces with lighter skin as to over-represent the faces with darker skin. The resulting dataset would not be a representative sample of Hollywood, which indeed is biased towards lighter-skin subjects, but the new dataset would allow to train a better network for I2I translation when introducing cartoon (or human) faces with dark skin. Finally, we also note the generated faces also do not look very realistic in the neck, because cartoon faces in our data set do not have necks.

6.4. Comparing our XGAN with other implementations

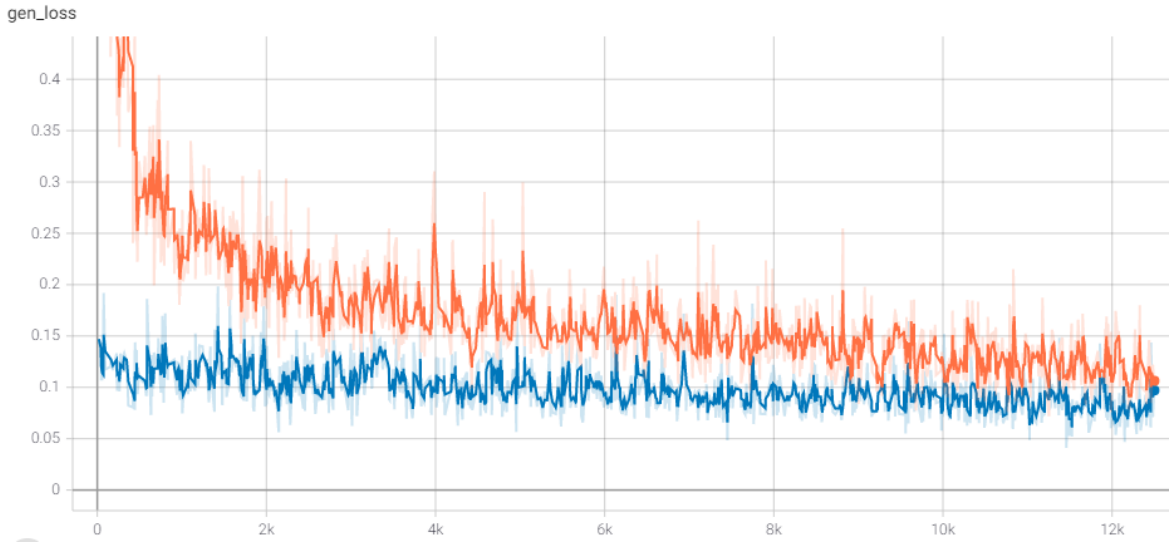
In parallel to the implementation from scratch of an XGAN, we also tested two existing implementations that can be found on GitHub.

6.4.1. CS2470FinalProject version

This version was submitted in 2018, the same year as the XGAN paper. In order to test it, we migrated the code to the latest TensorFlow 2 version and other old libraries like `scipy` were replaced by new ones like `imageio`.

For the cartoon domain we used the same Google cartoonset. For the real faces, we made tests with two different datasets: CelebA and Public-IvS. Although the code looked really good, with dozens of customizable parameters and proper code design, it didn't produce any useful result. During the execution, the loss quickly converged. This is the evolution of the general loss when training the XGAN two times, one in each direction. 5 epochs of 2,5 thousand iterations each.

Figure 12. StarGAN-V2 Reference Cartoon Source Human



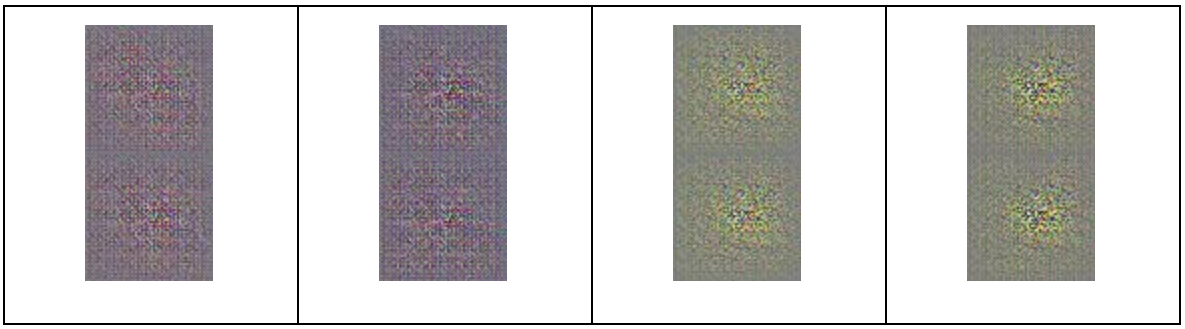
Orange: Cartoon → real face

Blue: Real face → cartoon

Despite the loss seemed reasonable, the results did not improve substantially as can be observed in Figure 13.:

Figure 13. Generated cartoon faces

C → RF Epoch 0, 100th iteration	C → RF Epoch 4, 2499th iteration	RF → C Epoch 0, 100th iteration	RF → C Epoch 4, 2499th iteration



6.4.2. Conclusions:

We were not able to determine the exact cause of this phenomenon. For some reason, the generator only learnt to produce noise and the discriminator took as correct any input passed by the generator.

6.4.3. paper2code-pucp/avatar-image-generator version

This version is not finished at the moment this report was written. Attempting to run the code was unsuccessful, due to some error at reading the images from the dataset folder.

7. Conclusions

In this section we review the most important conclusions of our experiments and our learning process, while more detailed conclusions associated with each experiment were provided in the results section. First, the importance of the input data should not be underestimated in any Deep Learning project. From the beginning, the election of our research topic was informed by the availability of a public cartoon dataset that we could use for our implementation. Indeed, everything worked out smoothly when we managed to generate single-domain cartoon faces using this public dataset. However, it was only when we moved from a single-domain generation to a multiple-domain generation (i.e. human and cartoon) that we realized how important it was that human faces had a noiseless background and the correct size and position before entering our encoders. Moreover, it was only after constructing our own data set of resized, cropped and free-of-background human images that we could succeed in the translation from the human domain to the cartoon domain.

Second, implementing GANs for single-domain generation before implementing XGAN for multiple-domain generation was a useful strategy that helped us to envision XGAN models

with cross-domains and to better understand the internal machinery of GAN models before moving to the more complex task. There were various shared structures and concepts between GANs and XGANs. Unfortunately, we could not take advantage of the single-domain GANs that we trained at the beginning of the project, because XGAN architectures involved shared parts across the encoder-decoder structures of the two domains (for humans and cartoons) that were naturally not present in single domain GANs. Therefore, the isolated single domain generation networks that we trained at the beginning *per se* were not really useful for our final implementation. Nevertheless, the knowledge and experience that we gained in this first step was key for our final implementation.

Third, by reducing the learning rate in the implementation of both our single-domain and multiple domain tasks we were able to improve our results. In particular, we could finally avoid the networks to get lost and diverge, generating even worse images at the end that at the beginning of the epochs, by reducing the learning rate. We acknowledge that if we had more time, exploring a learning rate scheduler would be indeed a great option for future implementations. Fourth, we learned that a large part of the time we spent on the project was spent preparing the input data or organizing the internal workflow of our models (e.g., putting the generator and discriminator together in a class of Python etc) and not really about designing the architectures. In part this is because we were following an existing model (the XGAN), but we also suspect that this must be the case most of the time.

Finally, from our comparisons with StarGANs we have a general reflection. Cartoon faces seem a good device to learn in I2I frameworks, even in cases that do not deal with cartoonization as in our case study. That is, the cartoon faces allows you to focus on obvious attributes of the images such as hair and skin color, which in turn enables you to better understand how the network is working inside, detecting possible failures or limitations (like the limitation related to the inability of the network to translate dark-skin into the generated images). This in turn, helps you to propose alternatives for improvement. In short, cartoon-faces are simplified examples of human real faces, and using them allows you to focus on specific attributes during the learning process that otherwise might be ignored.

References

[1] Pang, Y., Lin, J., Qin, T., Chen, Z. (2021) Image-to-Image Translation: Methods and Applications. Working paper available at arXiv:2101.08629

[2] Royer, A., Bousmalis, K., Gouws, S., Bertsch, F., Mosseri, I., Cole, F., Murphy, K. (2017) XGAN: Unsupervised Image-to-Image Translation for Many-to-Many Mappings . Working paper available at arXiv:1711.05139+

[3] Google - Deep Convolutional Generative Adversarial Network. Available at: https://www.tensorflow.org/tutorials/generative/dcgan

[4] Google - Cartoon Set. Available at: https://google.github.io/cartoonset/

[5] Soumith Chintala, ELM (2018). How to Train a GAN? Tips and tricks to make GANs work. Available at: https://github.com/soumith/ganhacks

[6] Timo Stöttner (2019).Why Data should be Normalized before Training a Neural Network. Available at: https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d

[7] Choi, Y., Uh Y., Yoo, J., Ha, J.W. (2020) StarGAN v2: Diverse Image Synthesis for Multiple Domains. CVPR, 2020

[8] Choi, Y., Choi, M. Kim, M.,. Ha, J.-W, Kim, S. Choo, J. (2018) Stargan: Unified generative adversarial networks for multidomain image-to-image translation. CVPR, 2018.

[9] Claudio, Jorge, Bernat, Roger. XGAN. Available: https://github.com/rogeralmato/XGAN