CartoonGAN

Valerio Longo, Luigi Sigillo

February 2021

Contents

1 Introduction

The goal of our project was to re-implement *CartoonGAN* (1), a Generative Adversarial Network able to transform real life photos into cartoonized images. More specifically, once given as input an photo and a name of a certain author, CartoonGAN gave as output an image modified with the art style of that author. The project has been developed taking as authors three very famous japanese artists: Hayao Miyazaki, Makoto Shinkai and Satoshi Kon. In this report we will focus about the implementation, the issues, the results and the general ideas behind them. Moreover we will talk about the parallelisms of our work with respect to other similar projects made.

2 Datasets

The whole dataset is composed by a set of 2217 real-life photos took from COCO Datasets (7). Due to the fact that different artists have different styles, when creating cartoon images of real-world scenes, in order to obtain a set of cartoon images with the same style, we used the key frames of cartoon films drawn and directed by the same artist. So for Hayao Miyazaki we decided to use *Spirited away* (5051), for Makoto Shinkai *Your Name* (3207) and for Satoshi Kon we took *Paprika* (3591).

2.1 Image preprocessing

All images are resized in 256x256 resolutions. Moreover, in order to follow the paper, we had to manipulate the three datasets containing anime frames. We built an *edge smoothed* version for each set because the presentation of clear edges is an important characteristic of cartoon images and it's crucial for the discriminator in reaching the goal to correctly choose which image has been generated and which has not. The steps used to generate the "edge-smoothed" version of a certain dataset were:

- Usage of a Canny edge detector in order to detect edge pixels
- Dilation of the edge regions detected
- Application of a Gaussian smoothing in the dilated edge regions

In order to re-implement the whole work in a more completed way, we approached two different variants for the Canny filter

(a) Normal images (b) Canny 1 (c) Canny 2

Figure 1: Different preprocessing

3 Architecture

In a GAN, the generator network **G** has to be trained in order to *fool* the discriminator network **D**. On the other hand, the discriminator has to decide whether an image given is a fake (generated) or a real one. In order to have similar results with the original work, we built the same architecture of CartoonGAN.

3.1 The Generator

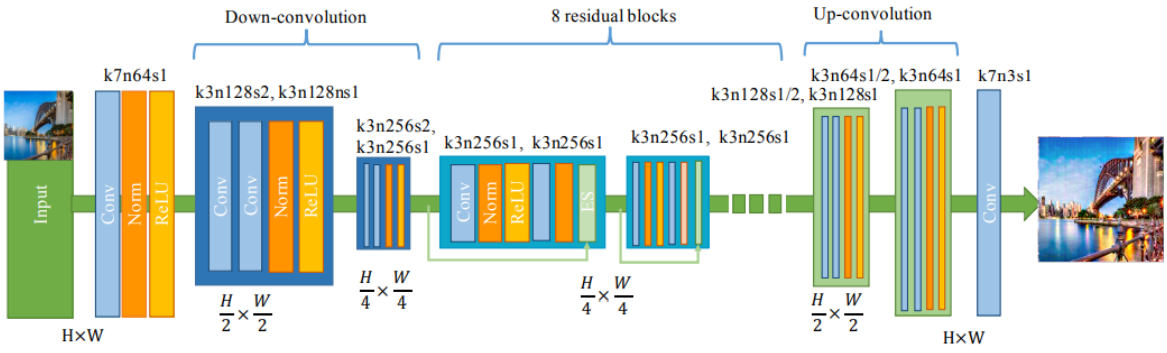


Figure 2: Architecture of the generator G proposed by CartoonGAN, in which k is the kernel size, n is the number of feature maps and s is the stride in each convolutional layer, 'norm' indicates a normalization layer and 'ES' indicates elementwise sum.

As the figure shows, our generator is composed by a flat convolution stage as first, then is followed by two down-convolution blocks to spatially compress and encode the images. Then, eight

residual blocks with identical layout are used to construct the content and manifold feature. In the end, the output cartoon style images are reconstructed by two up-convolution blocks which contain fractionally strided convolutional layer with stride 1/2 followed by a final convolutional layer with 7×7 kernels.

3.2 The Discriminator

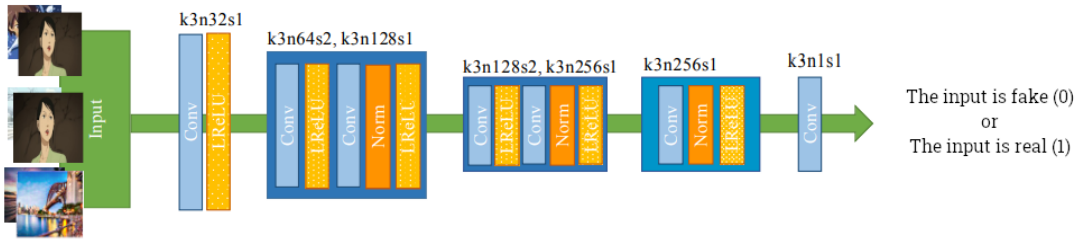


Figure 3: Architecture of the discriminator D proposed by CartoonGAN, in which k is the kernel size, n is the number of feature maps and s is the stride in each convolutional layer, 'norm' indicates a normalization layer

We can see from Figure 3 that our discriminator is composed by a first stage with flat layers,two strided convolutional blocks (in order to reduce the resolution and encode essential local features for classification), a feature construction block and a 3×3 convolutional layer used to obtain the classification response. After each normalization layer a Leaky RElu activation function has been used

3.3 The choice of Normalization

The original authors used batch normalization inside the structure of the two networks. Due to the fact that nowadays the instance normalization is preferred in GAN implementations and after saw other cartoonGAN re-implementations where it's used, we also tried to use this kind of normalization with results explained later. (2)

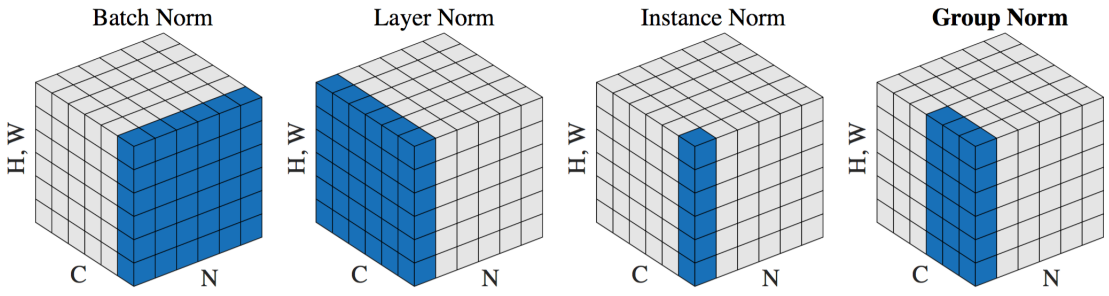


Figure 4: Different type of normalizations

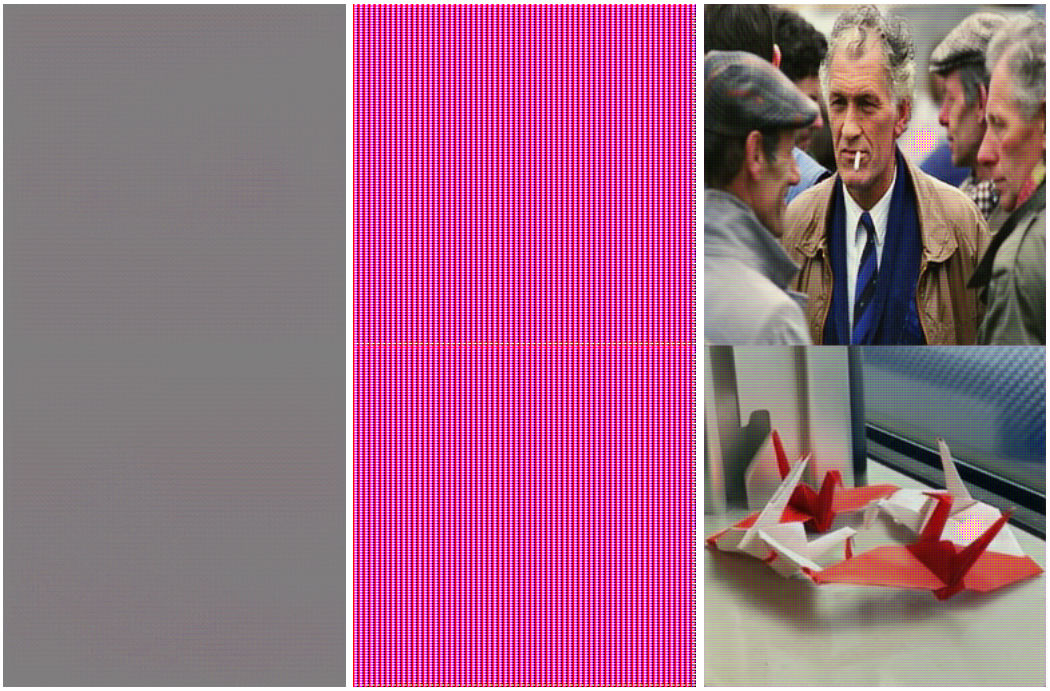
Each subplot shows an input tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes (Height and Width of a picture for example). **The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.**

4 Training

As proposed by the original cartoonGAN, the training of the whole system is preceded by a *pre-training phase*. So now, Let's see it in details.

4.1 Pre-training phase

This initialization phase has the purpose to converge faster the GAN into a good configuration, without premature convergence. As the original authors suggested, we trained the generator for 10 epochs with *only the content loss* in order to slightly change the original weights of the VGG with respect to the new dataset.



(a) Without pre-training (b) Batch pretrained (c) Instance pretrained

Figure 5: Different preprocessing

So, after this initialization phase, we started various experiences that we will discuss later, training for a maximun of 200 epochs. The training has been made using *Google Colaboratory* that exploits as a GPU the Nvidia Tesla T4 with a compute capability of 7.5".

4.2 The Loss Functions

The original paper presents **two** different loss functions: The **Adversarial Loss** and the **Content Loss**. This is because while the adversarial loss should drives the generator to achieve the desired manifold transformation, the content loss should preserves the image content during cartoon stylization. They are added together as:

$$L(G, D) = L_{adv}(G, D) + \omega L_{con}(G, D)$$

Adversarial Loss indicates if the output looks like a cartoon image or not. The paper highlights that a characteristic part of cartoons images are the clear edges, which are a small detail of the image and it must be preserved to generate clear edges in the result. For achieving this, the authors

defined the edge-promoting adversarial loss function as follows:

$$L_{adv}(G, D) = E_{\mathrm{ci}}[logD(ci)] + E_{\mathrm{ej}}[log(1 - D(ej))] + E_{\mathrm{pk}}[log(1 - D(G(pk)))$$

With ci the i-th resized anime frame, ej the j-th frame in the edge smoothed version set, and pk the k-th real-life photo.
For the **discriminator**, this is the formula for the whole loss function, because output of the discriminator plays no role within the content loss part of the loss function.
For the **generator**, only a part of the formula is used within its loss function, and is :

$$E_{\mathrm{pk}}[log(1 - D(G(pk)))]$$

Content loss ensure that the resulting cartoon image must have the right semantic content of the input photos. So we, as made by the authors, have chosen to transfer learning from the VGG network, more precisely from the 'conv4_4' layer. The Content loss ha been defined has follows

$$L_{con}(G, D) = E_{\mathrm{pi}}[\|VGG_{\mathrm{l}}(G(pi))) - VGG_{\mathrm{l}}(pi)\|_{\mathrm{L1}}]$$

with pi the i-th real-life photo and L1 as the sparse regularization, the Mean Absolute Error. Our implementation of these loss functions are:

- Mean absolute error for the content loss.
- The binary cross entropy for the generator and discriminator losses.

4.3 Extra: Style Loss

We found that in other works (6) different from the original paper, in order to retrieve a particular artistic style from a photo, a new kind of loss function is defined and is added to the previous ones: **The Style Loss** (3). For a give layer l, it is:

$$Lstyle_{\mathrm{l}} = \omega \frac{1}{4N_{\mathrm{l}}M_{\mathrm{l}}} \sum (G_{\mathrm{ij}}{}^{\mathrm{l}} - A_{\mathrm{ij}}{}^{\mathrm{l}})^2$$

with A and G the Gram matrices that represents the style (in layer l) respectively of the original work and the generated one, and the ω (different from the ω of the original work) is a simple weight to balance this function, such weight we called it L_1. N is the number of feature maps each of size M. The Gram matrices are obtained with the inner product between the i-th and the j-th vectorised feature maps

5 Experiments

We performed multiple experiments using different values for the hyperparameters in particular for these:

- ω is the weight that we assign at the content loss (the VGG one)
- g_l is the weight that we assign at the adversarial loss
- L_1 is the weight that we assign at the style loss
- The values of the learning rates for both discriminator and generator
- The size of the cartoon datasets to be considered
- A different preprocessing for the images, for instance using different approaches for the canny filter or normalizing the images between 0 and 1

and using different couple of generators and discriminators:

- Using the instance normalization for both models
- Using the batch normalization for both models
- A mix of the two normalization

5.1 Original paper $\omega = 10$

We trained for 140 epochs, following the hyperparameters suggested by the original paper:

- $\omega = 10$
- $g_l = 1$
- $L_1 = 0$.
- $1e - 5$ as value for the Adam optimizer learning rate
- Using the whole dataset of spirited away
- Using approach [1] for the canny filter

So we have reproduced exactly the paper, but the results with this hyperparams are not as expected. Since we are using the content loss with this higher value wrt to the adversarial loss, we have too many information of the original images meaning that the generated are basically equal to the original images.

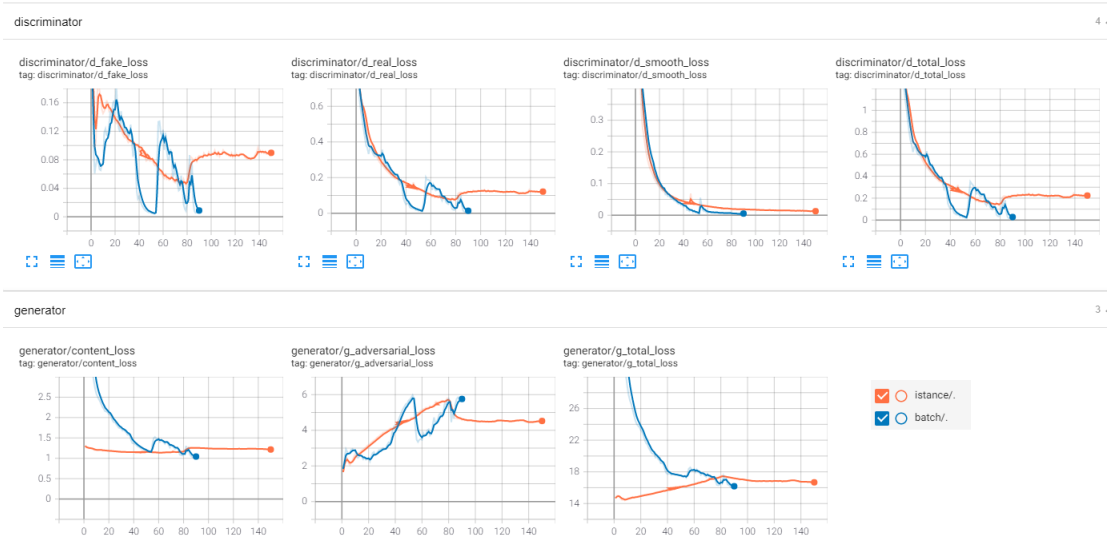
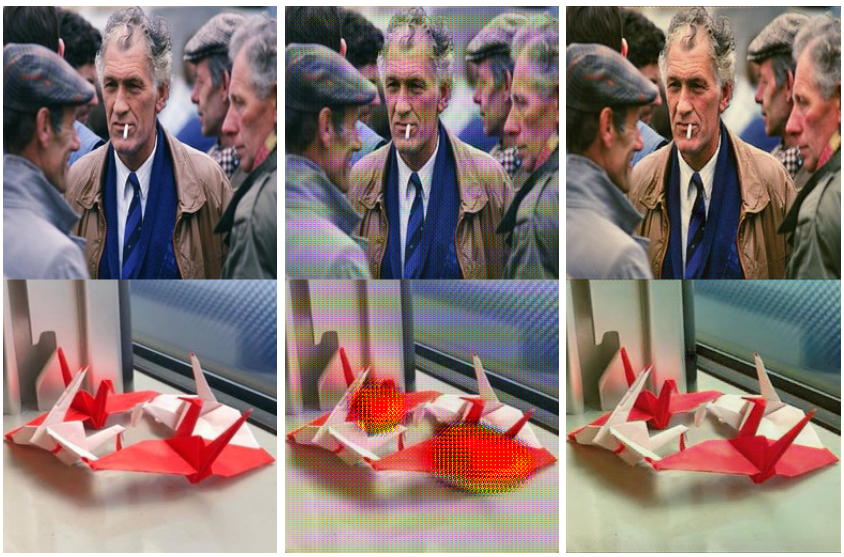


Figure 6: Plots of loss functions

We trained with the same hyperparameters but changing the batch with the instance normalization, and also with this model the images produced are not cartoonized at all.



(a) Normal images (b) Generated images with Batch (c) Generated images with Instance

Figure 7: Results of trial with $\omega = 10$

5.2 Tensorflow "official" re-implementation) $\omega = 0,4$

We choose this ω because this is the value suggested by one of the implementation (6) we have seen of the project, that has very good results. In this trial we added also the style loss(3) as suggested. We are using another discriminator using the batch normalization instead of the Instance, used for the generator, and the results are again changing. The loss functions are respecting the paper's one. We stopped the training at 35 epochs, because of the not interesting results.

- $\omega = 0,4$
- $g_l = 8$
- $L_1 = 25$
- $8e-5$ as value for the generator and $3e-5$ for the discriminator learning rates, using Adam
- Using the whole dataset of paprika
- Using approach [1] for the canny filter

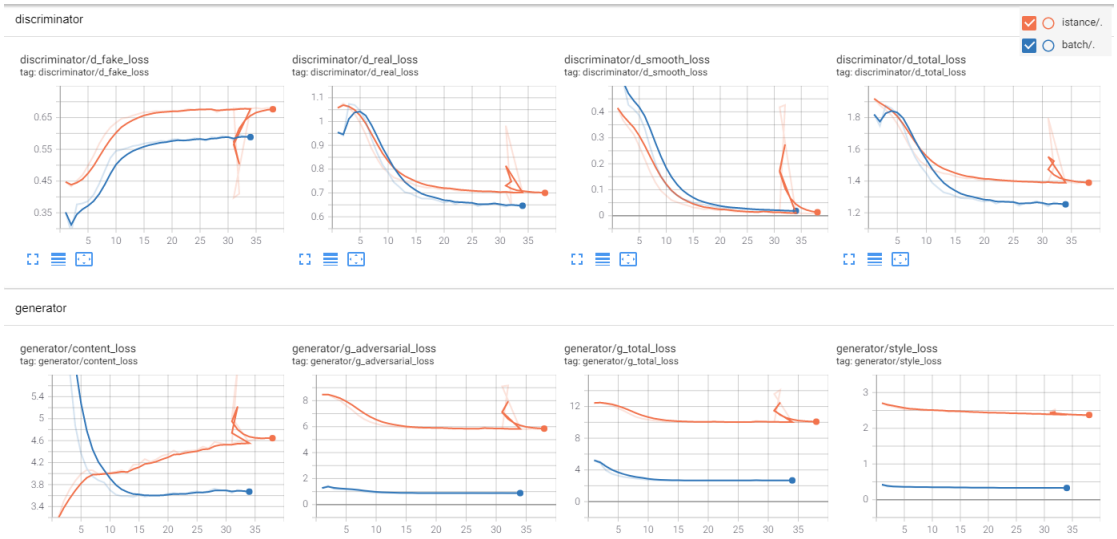


Figure 8: Results of loss

(a) Normal images (b) Generated images instance (c) Generated images batch

Figure 9: Results of trial with $\omega = 0,4$

5.3 An interesting pytorch re-implementation $\omega = 5e - 6$

We trained using these hyperparameters, following the result obtained by this project (5):

- $\omega = 5e - 6$
- $g_l = 1$
- $L_1 = 0$.
- $1e - 5$ as value for the Adam optimizer learning rate
- Using the whole dataset of paprika
- Using approach [1] for the canny filter

In terms of loss function this model performs in a appropriate way but the resulting image are really bad and not cartoon at all. There is some kind of noise in the form of holes in the image reconstruction. We are using a very small value for the content loss to give more importance to the adversarial and so to the cartoons, this may cause the holes in the images. The losses are all converging. We decided to interrupt the training at 37 epochs, because of the holes in the resulting images and too many detail lost from the vgg layer.

We performed the same test also on a model using the instance normalizations or with normalized images or using different canny filter.

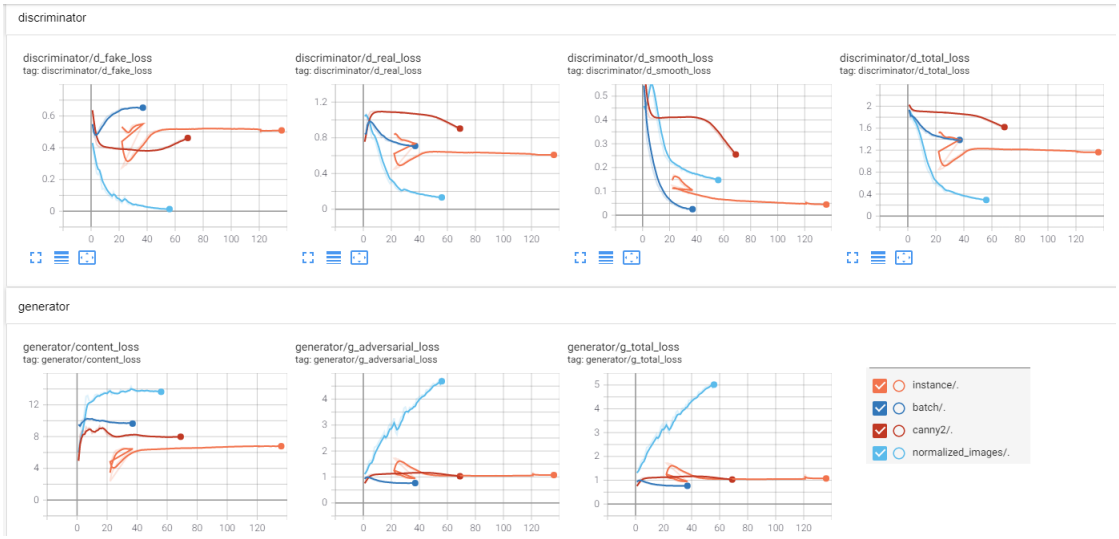
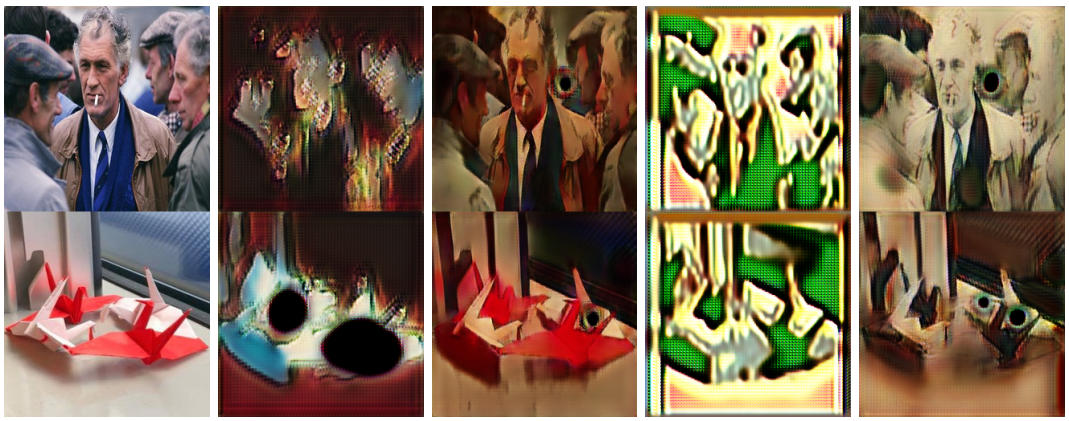


Figure 10: Results of loss functions



(a) Normal images (b) Generated images Batch (c) Generated images Instance (d) Generated images with normalized images (e) Generated images with canny 2

Figure 11: Results of trial with $\omega = 5e - 6$

5.4 Our best values for the hyperparameters

After many experiments, we find out the hyperparameters that best suits our datasets and provide us a pretty good result. In particular we are using the Batch Normalization in both the discriminator and the generator. We decided to not use the instance normalization, because of the poor results with respect to the batch as we can see from the photos below, and also from the comparison of the loss functions.
These are the hyperparameters:

- $\omega = 1.5$
- $g_l = 1$
- $L_1 = 1.$
- $2e - 4$ as value for the Adam optimizer learning rate
- Using a complete dataset
- Using approach [2] for the canny filter

(a) Normal images (b) Instance normalization (c) Batch normalization (d) Batch with style loss

Figure 12: Results of trials with 45 epochs

So we decided to keep going with the batch with style loss, because as we can see after 45 epochs it provides better results.

5.4.1 Results

These are the results obtained with the different datasets using the same hyperparameters, as we can see there are intersting difference in the values of the loss functinos, while the genral trends are similar.

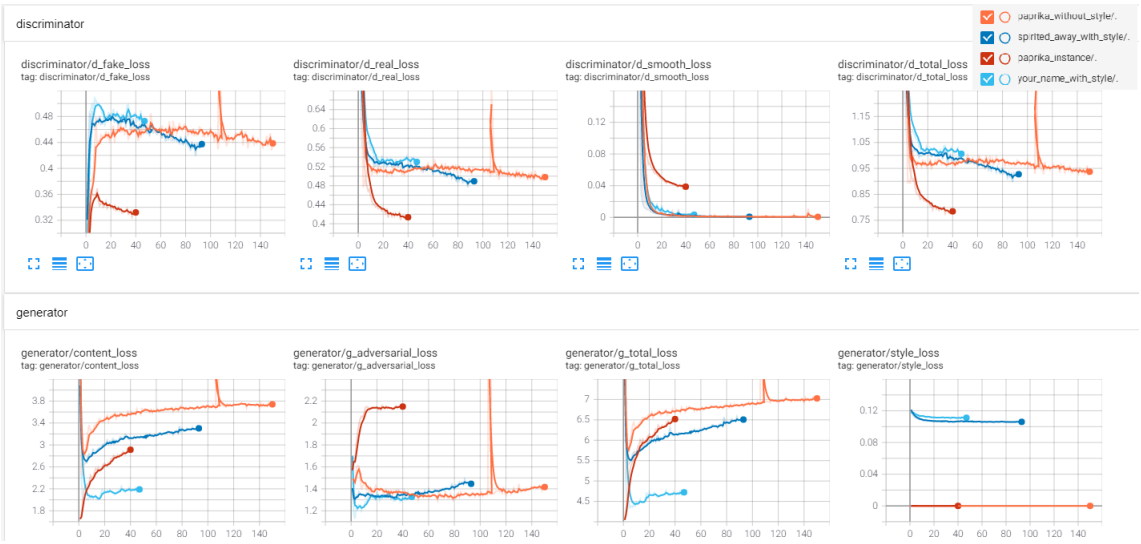


Figure 13: Results of loss functions

We can observe that the adversarial loss in each trial, in a slow way but is continuing to grow and this is somehow expected by the fact that also the paper loss are behaving in this way. We can observe the fact that after the 40 epochs the loss are growing without we can take as our best model before the 40 epoch. We can compare these loss with the ones of the 5.2 (6):

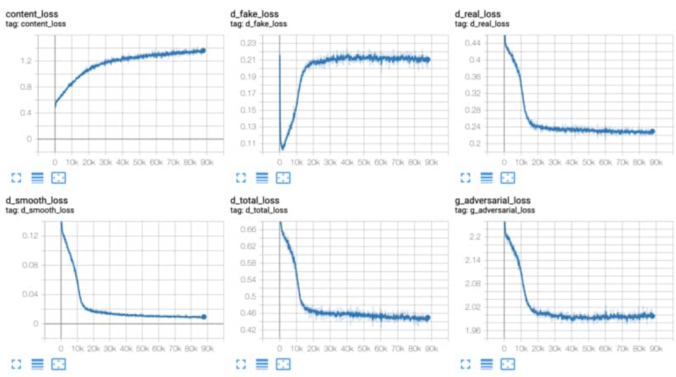


Figure 14: Results of loss functions for mcnic implementation

We can compare the different results using the different style also with the paper results:

(a) Normal images (b) Our Spirited away style (c) Paper Spirited away (d) Our Your name (e) Paper Your name

Figure 15: Results comparison with paper

5.5 Differences between datasets

Once had very interesting results using the above mentioned group of hyperparameters obtained with *Paprika* dataset, we applied the same ones to the other two: *Your Name* and *Spirited Away*.

(a) Normal images (b) Paprika generator (c) Spirited away generator (d) Your Name generator

Figure 16: Results after 40 epochs with different generators but with the same (optimal) hyperparameters

As we can see from Figure 12, the result, in terms of "*cartoonization*" are very different. In fact, while Spirited Away completely reached the goal, Paprika has similar but lower results and Your Name has very slight changes. So, this leads to the fact that **the hyperparameters of the network are dataset-dependant**

6 Conclusion

CartoonGAN provides a robust network capable to transform real life photos into cartoon-based images. Our work of re-implementation of this Generative Adversarial Model was very interesting, in fact we learned how difficult is to tune the network in order to have the best results. Moreover, we learned that its hyperparameters are very dataset-dependant even if the semantic context of these are similar (in this case, all datasets came from anime movies). We reached our optimal settings that are different from all other work. Despite this, we noticed that **every work of re-implementation** of this network provides different hyperparameters. So, due to this and due to our output images, we can conclude that we found our optimal settings.

References

[1] Yang Chen, Yu-Kun Lai, Yong-Jin Liu. *CartoonGAN: Generative Adversarial Networks for Photo Cartoonization*. CVPR 2018.
https://openaccess.thecvf.com/content_cvpr_2018/papers/Chen_CartoonGAN_Generative_Adversarial_CVPR_2018_paper.pdf

[2] Dmitry Ulyanov, Andrea Vedaldi, Victor Lempitsky. *Instance Normalization: The Missing Ingredient for Fast Stylization*.
https://arxiv.org/abs/1607.08022

[3] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge *A Neural Algorithm of Artistic Style*.
https://arxiv.org/abs/1508.06576

[4] Longo Valerio, Sigillo Luigi *CartoonGAN*
https://github.com/LuigiSigillo/CartoonGAN

[5] Tobias Sunderdiek *CartoonGAN*
https://github.com/TobiasSunderdiek/cartoon-gan

[6] Lee Meng *CartoonGAN*
https://github.com/mnicnc404/CartoonGan-tensorflow

[7] *COCO Dataset 2014 train*
https://cocodataset.org/#home