Monet Painting Generation

Gosman, Mircea
mircea.gosman@mail.mcgill.ca
Douady, Arno
arno.douady@epfl.ch

***Abstract*—The goal of this project in artificial intelligence is to develop a Generative Adversarial Network (GAN) to create new paintings in the style of the French painter Claude Monet. In this paper, we describe our several approaches to the problem and discuss the issues we encountered.**

I. Introduction

This is no coincidence if Artificial Intelligence is part of the 2022 *words of the year*: accessible uses of AI on several new websites such as *Dall · e* [2] and other art generation agents caught the public's attention. In this context, we decided to concentrate our project on generating new paintings in the style of Monet. To progress toward our entertained goal, we built and trained Generative Adversarial Networks (GANs): a recent form of unsupervised learning agents. At first, we aimed to generate the paintings out of thin air. However, this proved especially challenging, and we pivoted our later experiments towards a reduced problem. That is, performing image translation to create paintings out of modern photographs. In this paper, we mainly comment on our failed first experiments which used a DC-GAN and an SN-GAN. Furthermore, we detail a solution to the image translation problem using a more complex Cycle Gan architecture.

II. Extended Coursework

GANs are an ongoing deep-learning research topic. They are a form of unsupervised learning as they do not rely on pre-labeled output to verify their results during training. Instead, GANs rely on a set of sets (networks) of convolutional neural layers for feature extraction. For ECSE 526 course purposes, we consider our work expands on class content related to recurrent neural networks (objective/loss, normalization, and activation functions), deep learning, and unsupervised learning.

III. Related Work

A. Generative Adversarial Network

1) Components: The first GAN was brought by Ian Goodfellow [5] in 2014. Given training data, the model can create and classify images using the competitive nature of its two sub-models: a generator and a discriminator.

As the generator is trained, it becomes precise in generating *fake* data instances. It is positively rewarded when the discriminator fails to classify the generator's output as fake. In parallel, as the discriminator gets trained, it gains accuracy in differentiating real data instances from generated ones. Its goal is to always correctly classify images, regardless if they come from the original dataset or from the generator (Fig.1). To be noted, the discriminator is discarded for post-training predictions as we then only care about the generator's output: new Monet paintings in our case.

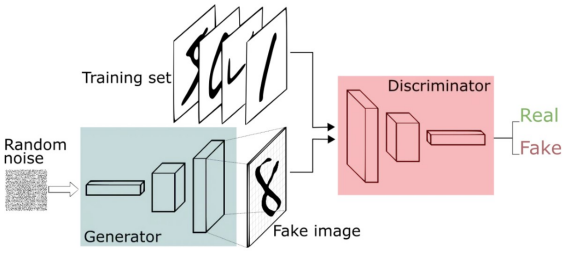


Fig. 1: Simplified View of the GAN Data-flow [3]

2) Training Mathematics: For every training epoch, the generator (G) will create a batch of samples of vectors x following a distribution $p_g(x)$. Its goal is to match the distribution of the produced vector x with the distribution of the real data vectors $p_{data}(x)$. On the other hand, the discriminator (D) tries to differentiate between x and $p_{data}(x)$: $D(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$. When both submodels are put together, the training of the GAN solves the following min-max equation [1]:

$$\max_D \min_G V(G, D) \tag{1}$$

for

$$V(G, D) = \mathbb{E}_{p_{data}(x)} log D(x) + \mathbb{E}_{p_g(x)} log(1 - D(x)) \tag{2}$$

where and $\mathbb{E}$ represents the expected value of a distribution.

3) Use Cases: Prior to implementing a GAN, one has to decide the input type of the generator. In fact, we are aware of two possible use cases for an image-processing generator. The first is 2D image generation from a one-dimensional noise vector. In this case, the generator attempts to generate images similar to those from the dataset. The second is image translation from one domain to another. In this configuration, the generator uses a dataset of pictures from domain "A" to try to create images that would fit within the dataset of another domain "B" fed into the discriminator.

B. GAN Variants

In our project, we have explored both outlined GAN use cases using three main approaches.

1) Deep Convolutional GAN: Traditionally, a DC-GAN's generator consumes a noise input which is reshaped to the needed output image size. For feature extraction, it alternates convolutions and upsampling layers to maintain the image size across operations. As for the discriminator, it relies on a series of convolutions and flattens the final output to a single Boolean value using the sigmoid activation function. It is the most simple architecture to start with as it contains the least structure to implement.

2) Spectral Normalization for GANs: Whilst DC-GANs are a great starting point, we expected to have a hard time training them. This is mainly because the competitive relationship between the inner networks leads to vanishing gradient issues. In turn, extreme gradient values lead to poor learning and are mostly observed in the discriminator network. Their cause is, at least in part, rooted in the ineffectiveness of the batch normalization layers typically used by DC-GANs. To counteract this issue, Mufti et al.[10] wrap convolutional layers in spectral normalization ones instead. This newly introduced operation tries to enforce 1-Lipschitz continuity [9] (Fig. 2) on the gradient. GANs that make use of spectral normalization typically outperform DC-GANs and are commonly known as SN-GANs.

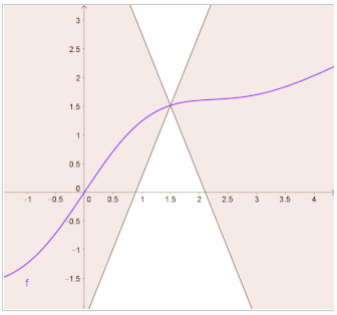


Fig. 2: Lipshitz continuity: the cone's lines never touch the function [8]

3) Cycle GAN: A Cycle GAN differs from the previous variants in that it tries to perform image translation. To do so, it comprises two generator-discriminator pairs, each related to a domain of data instances. One pair creates images of domain B using samples from domain A. The other produces pictures of domain A from domain B. Effectively, a Cycle GAN is a pair of GANs that train together. Its main advantage is the introduction of cycle consistency loss [13]. This ensures that each generated image maps back to / resembles its input picture. This architecture is especially helpful in cases where other GANs succeed in producing images of the desired domain but fail to maintain any output-to-input resemblance. Our results will

show our SN-GAN experiments have led us toward that situation.

When implementing a Cycle GAN, it is recommended the following architectures are used for the sub-models:

a) Encoder-Decoder Generator: The classic DC-GAN uses its generator to up-size its input. Downsizing is only performed in the discriminator. In the encoder-decoder model, the generator synthesizes features by first downsampling the input with convolution layers. When at a pre-determined size bottleneck, the network begins an up-sampling phase with transpose convolution layers to create an adequately sized output. This allows to separate the feature extraction and output generation phases such that lower level features are learned with greater efficiency [7].

b) Patch-GAN Discriminator: A traditional discriminator is a binary classifier. However, such a classifier loses local feature information when generalizing over the whole input image. To reduce this issue, some discriminators produce a matrix of Boolean decisions where each decision corresponds to some area of the input image. It has been demonstrated that discriminators that map their decisions to 70x70 "patches" of the input images achieve better performance than those who map back to 1x1 or full image (like in traditional discriminators) areas [7]. These numbers are relevant for sizes of images found in our dataset.

IV. Dataset

A. Repartition

The dataset we use is the one provided on Kaggle's *I'm Something of a Painter Myself* competition [6]. It is composed of 300 Monet paintings and more than 7000 photographs. Each data sample is 256x256 RGB pixels. The channel values are normalized between -1 and 1 before each experiment to allow for proper GAN training.

In experiments that generate paintings from random noise, a test dataset was not required. There, we used all 300 available paintings for training. However, in experiments that involved image translation, we split the available paintings so as to obtain a 2:1 ratio of training to testing data. In this case, the photographs samples in use matched the paintings both in quantity and in the aforementioned ratio.

B. Using the Dataset to Validate Results

When starting out with GANs, many approach [10] result evaluation from a qualitative (alas, subjective) perspective. To progress in our methodology, we evaluated our results based on subjective judgment as well as on the sliced Wasserstein distance (SWD) metric. SWD is a distance that is computed between a generated sample and a randomly picked image from the Dataset [11]. Mufti et al.[10] make use of the method to benchmark their models. That being said, the calculation of the SWD is a computationally intensive task. Our results were limited to 15 SWD samples per model.

V. Methodology

We began by trying to generate paintings from random noise. Our first DC-GAN underperformed, thus we attempted to improve it by using an SN-GAN. Finally, we reduced the problem to image translation and created a Cycle GAN in order to obtain a result to present in this paper.

A. Painting Generation from Noise

1) Deep Convolutional GAN: The DC-GAN we built borrows tips from an online tutorial [4] and from the architecture proposed by Radford et al. [12]. Notably, it starts by resizing the 256x256 images into a more manageable 128x128. The complete architecture we used for both the generator and the discriminator can be found in Appendix A. It mainly corresponds to the DC-GAN description discussed previously. For the generator, we note having used an input noise size of 100, convolution kernels of size 3, and Leaky ReLu activation functions with an alpha of 0.2 [7]. The discriminator differed from the generator in that it used ReLu functions instead [7]. Overall, this only ever produced random patches of colors. We venture that this is due to the previously expected gradient explosion problem.

2) Adding Spectral Normalization: The construction of our first SN-GAN follows the architectures illustrated in Appendix A for its sub-models. Essentially, we wrap convolutional layers of the DC-GAN's discriminators in spectral normalization layers. In doing so, we could remove batch normalization layers from the discriminator model.

To move our implementation closer to that of SN-GANs proposed by Zhu et al.[13], we also tweaked the generator to make use of half-strided convolutions. This required the replacement of upsampling layers with transpose convolutions ones. For context, upsampling layers size up their input by using KNN. Transpose convolutions are more adequate in that they give the network the opportunity to learn the best kernel to use for the upsizing operation. All this normalization effort led to the rectification of the vanishing gradient problem on the discriminator. We even started seeing Monet-like texture on some of the generated images. However, the images themselves still resembled patches of random abstract colors.

We aimed to obtain distinguishable objects, like in real paintings, but we hypothesize that our dataset is too small for our technique to work. We support this hypothesis by pointing out that the reference material for the SN and DC GANs use datasets of 7 to 10 thousand elements [7] Using only 300 paintings in our dataset, we may be able to replicate the general brush stroke of the painter, but not specific objects in the paintings.

As a side note, we also unsuccessfully tuned the dropout rate, batch size, and epoch counts of both models. Further details will follow the results section.

B. Image Translation

1) Cycle GAN: The Cycle Gan sub-models architecture that we used may be found in Appendix A. While the reference Cycle GAN material suggests the use of instance normalization, we maintained spectral normalization layers to build upon our previous work. The discriminator is a PatchGAN discriminator and outputs 16x16 Boolean verdicts that map to 70x70 areas on the input images. Its convolution kernels are of size 4, of stride 2, and use LeakyReLu activation functions. As for the generator, it uses convolutions with kernels of size 4. However, it is more complex as it not only uses the aforementioned encoder-decoder model, but also implements skip connections. These are needed because encoder-decoders may lose global context in their feature extraction [7]. By providing a skip connection from each downsampling encoder layer to a matching up sampling decoder layer (Fig. 3), the network aims to retain the global features of the input while it learns the lower level ones. We stopped our method with this model as the results it produced were starting to resemble those of available online material.

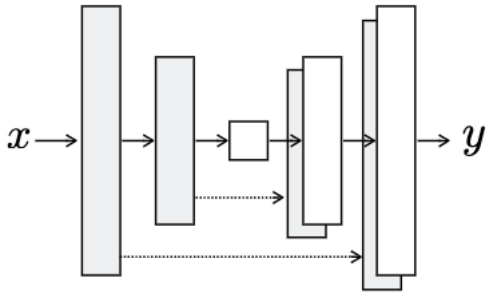


Fig. 3: Encoder-decoder with skip connections [7]

VI. Results & Discussion

A. Vanishing Gradient

In our test runs, the DC-GAN's accuracy jumps to 100% after 40 Epochs. Showcased by Fig. 4, this means that the error - the accuracy's complement - then becomes nill and never recovers. This need for better normalization, along with the poor qualitative results from Fig. 6's top row, motivated the SN-GAN architecture attempt. Fig. 5 shows that the SN-GAN succeded in keeping the discriminator's gradient in check, but that it did not improve its generator's situation. The sub-model collapses to 100% error and rarely recovers. Presumably, this may be because spectral normalization is not applied to the generator (both in our experiment and in the literature). As a result, Fig. 6's bottom row

shows the SN-GAN is somewhat capable of texture, but not of detail.

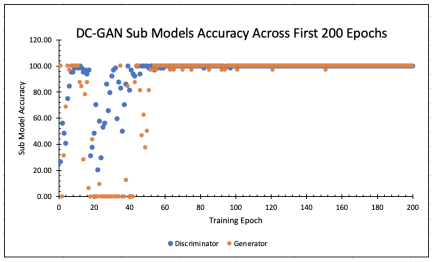


Fig. 4: Both DC-GAN submodels reach and hold perfect accuracy, hindering their learning.

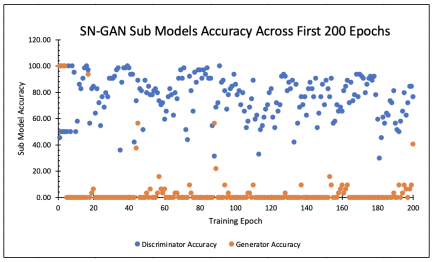


Fig. 5: Spectral Normalization allows the discriminator to recover from extreme accuracy values.

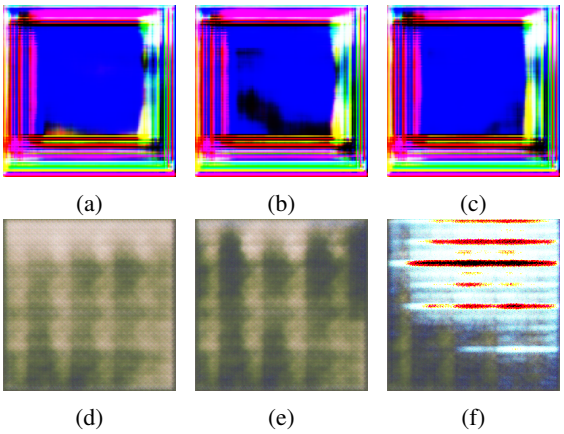


Fig. 6: Best results from the DC-GAN (first row) and SN-GAN (bottom row)

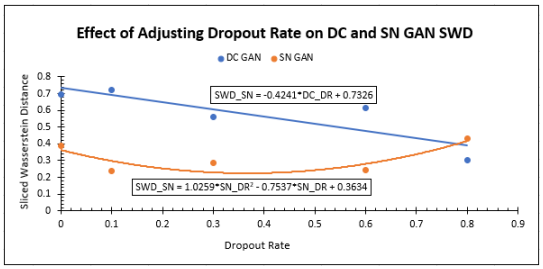


Fig. 7: A dropout rate of 0.3 benefits the SN-GAN

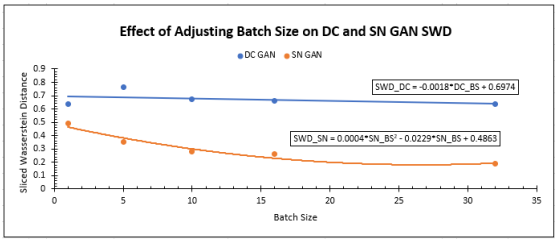


Fig. 8: A batch size of 30 improves the SN-GAN's SWD

B. Sliced Wasserstein Distance & Visuals

Individual model optimization was done by minimizing the sliced Wasserstein distance previously described. The parameters that were tweaked (only) on the DC & SN GAN models are the dropout rate, the batch size, and the total training epoch count.

On the DC-GAN, we observed that the SWD is minimal at high dropout rates (Fig. 7). Dropout discards part of the learned behaviour kernels. The higher it is, the more forgetful the network with respect to its vanishing gradient problem. Thus, this behaviour is expected, but it does not fully address the issue, and the network learns slower. As for the epoch (Fig. 9) and batch size (Fig. 8), we did not observe any meaningful visual improvements when varying them. The gradient vanishes so quickly that additional learning or learning segmentation does not matter. The DC-GAN has the highest SWD of all of our models (Fig. 10), which correlates with the worst visual results (Fig. 6, top row).

The SN-GAN showed its best results with a 0.3 dropout rate (Fig. 7), a batch size of 30 (Fig. 8), and an epoch count between 500 & 750 (Fig. 9). In

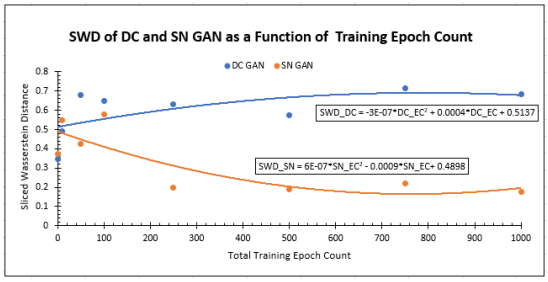
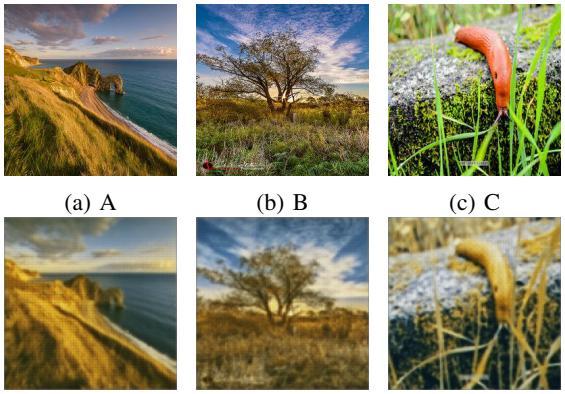


Fig. 9: An epoch count of 500-700 is best for DC and SN GANs

	DC GAN	SN GAN	Cycle GAN
SWD	0.562172924	0.188542533	0.066774307

Fig. 10: SWD of the best version of each model



(a) A (b) B (c) C

(d) G(A) (e) G(B) (f) G(C)

Fig. 11: Cycle GAN input/best output (top/bottom)

its optimal configuration, it obtained an SWD that is 3 times smaller than that of the DC-GAN. While this shows progress, this model remains stuck without admissible visual results. As discussed in the methodology, its lack of low-level visual features may be consequential to the generator's gradient or to the small size of the dataset.

As for the Cycle-GAN, it achieved an SWD almost 10 times smaller than that of the DC-GAN (Fig. 10). This is not surprising, as its cycle consistency loss and the generators' skip connections allow the networks to mimic part of the input image whilst adding Monet characteristics on top. Some input-output pairs have been cherry-picked for display in Fig. 11. However, even in these best results, we obtain a very soft 'Monet-ification'. What's more is that the new paintings contain artifacts. Specifically, Isola et al. [7], when they brought forth the patch-GAN, attributed such results to a mismatch between the discriminator's patch and the input image's size. We leave patch size experimentation as future work.

VII. Conclusion

In conclusion, this paper described an attempt at Monet Painting generation from noise input. The first proposed DC-GAN was improved with spectral normalization to offset the vanishing gradient problem on the discriminator. However, this did not correlate with an improvement in the generator's gradient explosion. Regardless of dropout rate, batch size, or epoch count variation, both DC & SN-GANs were incapable of producing the desired paintings. Thus, we proposed a cycle GAN to perform image translation of photographs into paintings. We were successful in doing so, but our results still contain visual artifacts from processing. As even our best sliced wasserstein distance remains twice as big as that of other cycle GANs, we suggest exploring discriminator patch size experimentation as future work.

References

[1] Antonia Creswell et al. "Generative Adversarial Networks: An Overview". In: *IEEE Signal Processing Magazine* 35.1 (Jan. 2018), pp. 53–65. DOI: 10.1109/msp.2017.2765202. URL: https://doi.org/10.1109%2Fmsp.2017.2765202.

[2] *DALL·E 2*. en. URL: https://openai.com/dall-e-2/ (visited on 01/22/2023).

[3] Rohith Gandhi. *Generative Adversarial Networks — Explained*. en. May 2018. URL: https://towardsdatascience.com/generative-adversarial-networks-explained-34472718707a (visited on 01/23/2023).

[4] *Generating Modern Art using Generative Adversarial Network(GAN) on Spell — by Anish Shrestha — Towards Data Science*. URL: https://towardsdatascience.com/generating-modern-arts-using-generative-adversarial-network-gan-on-spell-39f67f83c7b4 (visited on 01/14/2023).

[5] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. DOI: 10.48550/ARXIV.1406.2661. URL: https://arxiv.org/abs/1406.2661.

[6] *I'm Something of a Painter Myself*. en. URL: https://kaggle.com/competitions/gan-getting-started (visited on 01/12/2023).

[7] Phillip Isola et al. *Image-to-Image Translation with Conditional Adversarial Networks*. 2016. DOI: 10.48550/ARXIV.1611.07004. URL: https://arxiv.org/abs/1611.07004.

[8] *Lipschitz continuity*. en. URL: https://en.wikipedia.org/wiki/Lipschitz_continuity (visited on 01/22/2023).

[9] Takeru Miyato et al. *Spectral Normalization for Generative Adversarial Networks*. 2018. DOI: 10.48550/ARXIV.1802.05957. URL: https://arxiv.org/abs/1802.05957.

[10] Adeel Mufti, Biagio Antonelli, and Julius Monello. *Conditional GANs For Painting Generation*. 2019. DOI: 10.48550/ARXIV.1903.06259. URL: https://arxiv.org/abs/1903.06259.

[11] Khai Nguyen and Nhat Ho. *Revisiting Sliced Wasserstein on Images: From Vectorization to Convolution*. 2022. DOI: 10.48550/ARXIV.2204.01188. URL: https://arxiv.org/abs/2204.01188.

[12] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". en. In: *CoRR* (2015). URL: https://www.semanticscholar.org/reader/8388f1be26329fa45e5807e968a641ce170ea078 (visited on 01/11/2023).

[13] Jun-Yan Zhu et al. *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*. 2017. DOI: 10.48550/ARXIV.1703.10593. URL: https://arxiv.org/abs/1703.10593.

APPENDIX

A: DC-GAN Sub-models

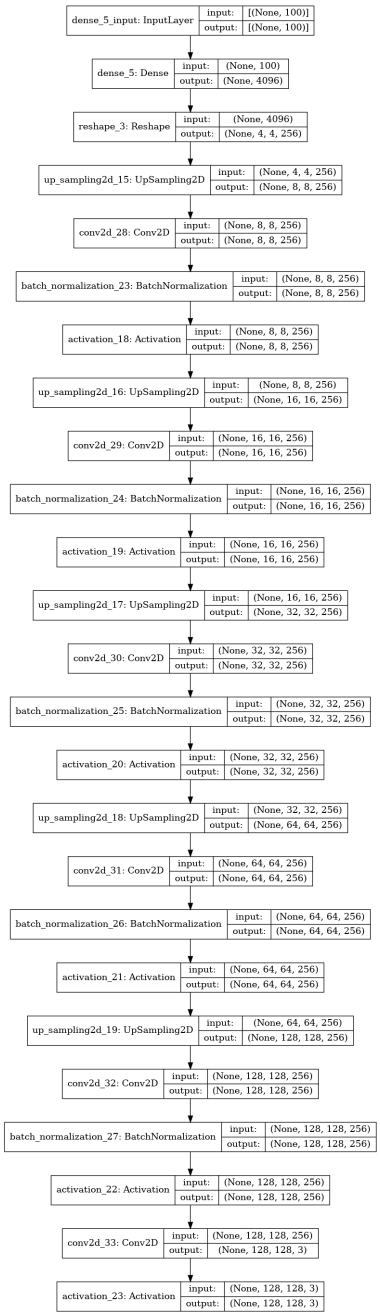


Fig. 12: DCGAN Generator Architecture

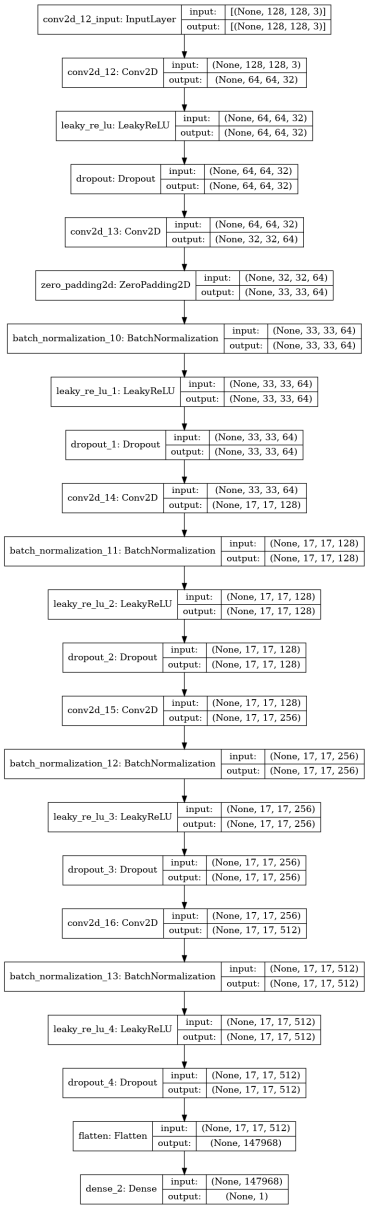


Fig. 13: DCGAN Discriminator Architecture

B: SN-GAN Sub-models

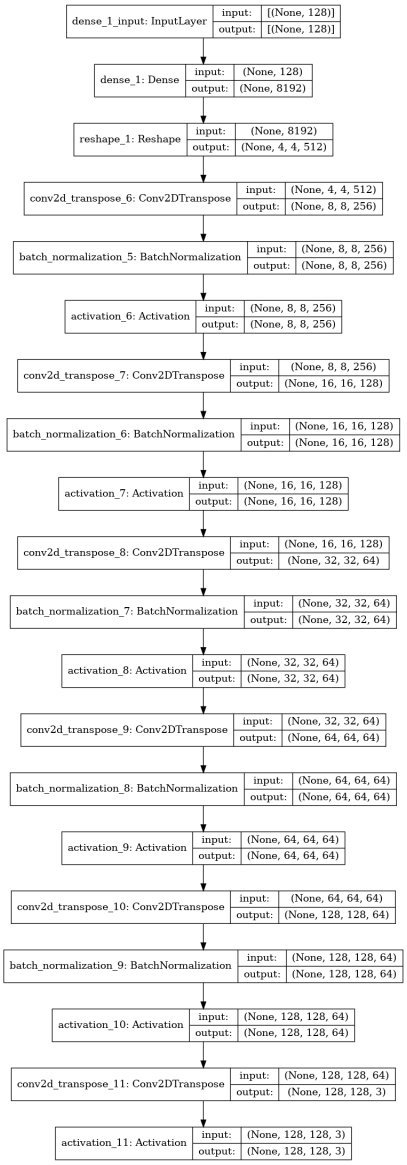


Fig. 14: SN-GAN Generator Architecture

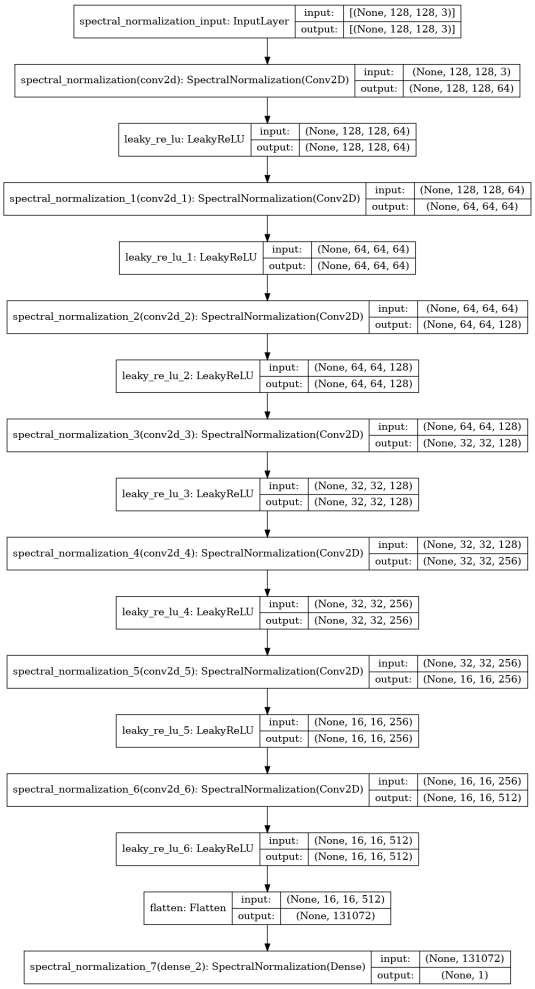


Fig. 15: SN-GAN Discriminator Architecture

C: Cycle GAN Finest Sub-models

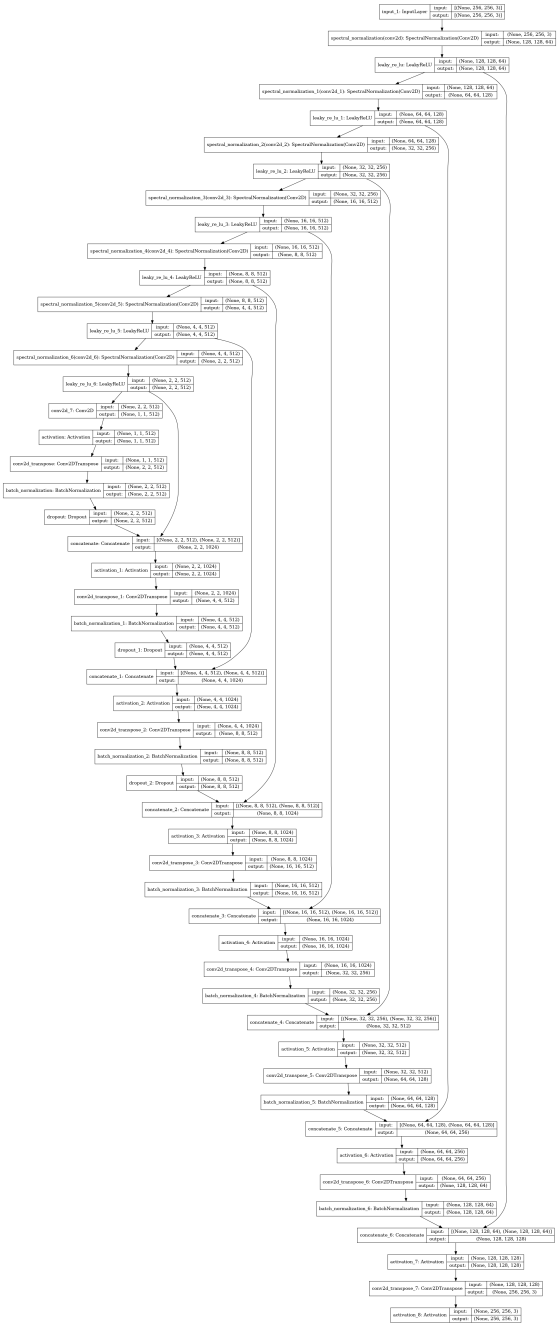


Fig. 16: Cycle GAN Generator Architecture

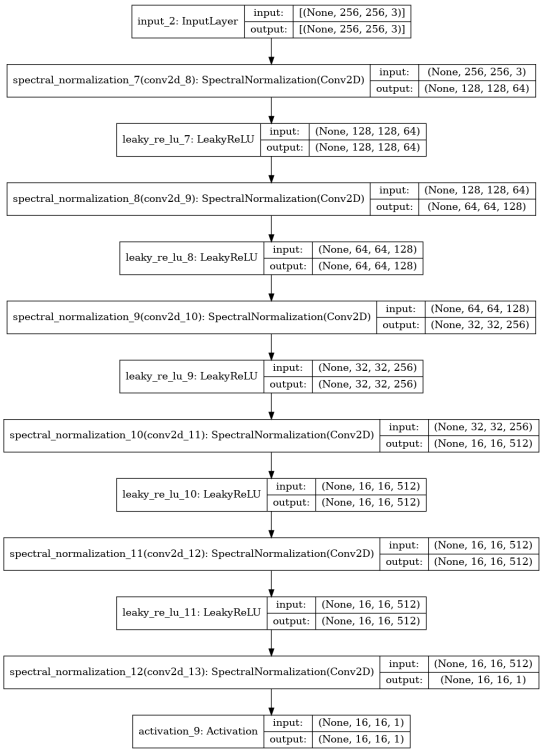


Fig. 17: Cycle GAN Discriminator Architecture