Deep Learning- Final Course Project

Eran Smuel (ID. 301796561), Eliyahu Shneider ID. 302516463)

Submitted as final project report for the DL course, BIU, 2021

1 Introduction

In this project we tried to solve the Kaggle challenge "I'm Something of a Painter Myself". The goal of this challenge is to create 7,000-10,000 images in the style of Monet by using generative models. We had one additional restriction: we only used 30 Monet images instead of 300. We were able to create a model with a public score of 55.18, which is in the middle of the leaderboard even though we used only 10% of the Monet images.

1.1 Related Works

A Generative Adversarial Network (GAN) is a framework in the deep learning field[1]. The framework contains two neural networks: the generator and the discriminator. The goal of the generator is to create images that fool the discriminator, and the goal of the discriminator is to distinguish between real images and the fake images that were created by the generator.

CycleGan is a variation of the GAN framework[2]. The goal of this network is to transfer images between 2 domains. The framework contains 4 models: 2 generators which takes images from domain a or b and transfer them to the second domain, and 2 discriminators, one for each domain, each one designed to distinguish between images from its own domain and fake images that created by the domain's generator. In order to make the framework more stable, 2 more losses are added to the generator loss: cycle loss, which means that images transferred from domain a to domain b and back from domain b to domain a should be close to the original image, and identity loss, which means that when a generator is applied to images from it's own domain, he should return the same image.

StarGan is a variation of the CycleGan framework[3], designed for more then 2 domains. Instead of using 2 discriminator and 2 generators, the StyleGan framework contain only one of each. The generator gets an image x and a destination domain y and creates xy, which is x on domain y. The discriminator gets image x and tries to predict if it's real or fake, and also tries to predict its domain. The domain loss is then added to the generator in addition to the identity loss, the cycle loss and the Gan loss. It's also added to the discriminator but only when training real images.

Transfer learning is a subject in the neural networks field. The idea behind transfer learning is to use a network which was trained on a similar task to initialize the desired model. Sometimes, some of the layers might be replaced

due to the input and output of the network or for better training. More often than not, transfer learning is used when there is a task with not enough data, or when a network takes a lot of time to train. When there is a lack of data, freezing some of the pretrained network might be important. Sometimes it's even enough to train only the bayes layers[4] . Previous research showed that transfer learning can be useful when training GAN[5].

Data augmentation is another way to deal with lack of data. The idea is to edit the data in a way that keeps the data as reliable as possible, but still different from the source. This might be done by resizing, cropping, rotating, coloring the image etc. Previous research showed that data augmentation is important when training CycleGan[6].

2 Solution

2.1 General approach

Our main approach in this challenge was to use datasets from other painters and use CycleGan with transfer learning and data augmentation in order to overcome the small number of Monet images. Another approach we had is to use StarGan. Due to the fact that StarGan learns mutual features between domain transfers, we thought that it can use the features learned from other painters for Monet, and we viewed it as a different way for transfer learning.

Transfer learning: We wanted to test multiple transfer learning methods. We tried multiple combination of frozen layers and we also tried to train all the bayes layers, with or without more layers. We also used two different datasets from which we trained the framework: paintings by the painter Cezanne, and a bigger data which contains paintings from Cezanne, Van Gogh and Ukiyoe.

Data augmentation: For increasing the number of images we first increased the image size by a random number between 2 hyper parameters min-resize and max-resize, and then the image is randomly cropped to a fixed size of 256*256. We also did a random horizontal flip.

2.2 Design

CycleGan architecture: The CycleGan framework has 4 models: generator-Monet(G_M), generator-normal(G_N), discriminator-Monet(D_M), discriminator-normal(D_N). Both generators and discriminators share the same architecture. The Monet generators tries to minimize 3 loss functions, when x_m are paintings from Monet and x_n are normal images

1. $ganLoss = (D_M(G_M(x_n)) - 1)^2 + (D_N(G_N(x_m)) - 1)^2$
2. $CycleLoss = |G_M(G_N(x_m)) - x_m| + |G_N(G_M(x_n)) - x_n|$
3. $IdentityLoss = |G_M(x_m) - x_m| + |G_N(x_n) - x_n|$

The final loss is: $ganLoss + alpha\_cycle * CycleLoss + alpha\_Identity * IdentityLoss$

The discriminators are trained separately. The Monet discriminator tries to minimize the loss function $(D_M(x_m)-1)^2+(D_M(G_M(x_n)))^2$ and the Normal discriminator tries to minimize the loss function $(D_N(x_n)-1)^2+(D_N(G_N(x_m)))^2$

StarGan architecture: The StarGan framework contains 2 models, generator and discriminator. The generator gets an image and a target domain and tries to minimize 4 loss functions. The Gan loss which is $(D(g(x,y))-1)^2$. The CycleLoss which is $|G(G(x_1,y_2),y_1)-x_1|$. The IdentityLoss which is $|G(x_1,y_1)-x_1|$ and the domain loss which is $(G(x,y).preds-one\_hot(y))^2$

The Discriminator gets an image and outputs a prediction and scores for every domain. The loss function is $(D(x_1)-1)^2+(D(G(x_2,y_3))-0)^2+domain\_alpha*(D(x_1).preds-one\_hot(y_1))^2$

Generator architecture: The generator has 3 convolutional layers which resizes the image to scale*4/64/64, where scale is a hyper parameter, then 9 res-blocks, which are two convolutional layers with a residual connection. Finally there are 2 convolutional transpose layers which resizes the image to scale/256/256, and a final convolutional layer which resizes the image back to 3/256/256. After every convolutional and convolutional transpose layer there is a channel normalization layer and a Relu activation function, besides the last convolutional layer which uses Tanh activation function. The res-blocks also uses normalization layer after every convolutional layer but they only use Relu activation after the first one.

Generator					
Type	Channels In	Channels out	kernel size	padding	stride
Conv	3	scale	7	3	1
Conv	scale	scale*2	3	1	2
Conv	scale*2	scale*4	3	1	2
ResBlock*9	scale*4	scale*4	3	1	1
ConvT	scale*4	scale*2	3	1	2
ConvT	scale*2	scale	3	1	2
Conv	scale	3	7	3	1

Table 1: Generator layers.

Discriminator architecture: The discriminator contains 5 convolutional layers, followed by an average pooling layer. We used Leaky Relu activation function on the first 4 convolutional layers, and channel normalization after every layer besides the last one and the first one.

StarGan discriminator and generator: The StarGan discriminator and generator are the same as the CycleGan, except that the generator also gets a

domain destination as an input, and the discriminator also outputs the domain predictions. This is done by increasing the channels size in the generator input and the discriminator output.

Hyper Parameters: We trained the model for 25 epochs which took us about 20 hours. The learning rate is 0.0002, and we multiplied it by 0.8 every epoch after the tenth epoch. We used batch size of 4 with the Adam optimizer. For the loss function, we used alpha_cycle of 10 and alpha_identity of 1. For the data augmentation part, we increased every image by a minimum of 10 and maximum of 100 before cropping it randomly to the original size. We used a scale of 64 for both the generator and the discriminator.

Discriminator					
Type	Channels In	Channels out	kernel size	padding	stride
Conv	3	scale	4	1	1
Conv	scale	scale*2	4	1	2
Conv	scale*2	scale*4	4	1	2
Conv	scale*4	scale*2	3	1	1
Conv	scale*8	scale	4	1	1

Table 2: Discriminator layers.

3 Experimental results

3.1 Evaluation

We used two methods to evaluate the results. The first one is the Kaggle's MiFid score. The second method is our own Frechet Inception Distance (FID) score[7]. We found out that using the pretrained Inception v3 model didn't give us good indication for how good every model is, so we decided to train the pretrained model by training it on a dataset we created from Monet, Cezanne, Vangogh, Ukiyoe and regular photos. The models objective is to classify between those 5 classes. In order to fit the model to the new objective, we replaced the last layer to a fully connected layer which maps the features to 5 values.

3.2 results

We tried several methods of transfer learning approaches (table 3). The use of transfer learning helped for most of the methods and performed better than the pure Cyclegan solution. Transfer Learning from the bigger dataset which contains 3 painters gave better results when freezing one or no layers, while transfer learning from the Cezanne dataset gave better results when freezing 3 or 7 layers. Training only the bayes gave bad results, while training bayes with 3 more layers gave worst results than training only the 3 layers. We also tried to train the generator on the first and the last 3 layers, which gave similar results to training only the last 3 layers. We had much better results on CycleGan

CycleGan					
Transferred learned from	generator unfrozen layers	discr unfrozen layers	Unfrozen Bayes	Kaggle's MiFid score	Fid score
-	-	-	-	71.63	64.32
Cezanne	all	all	-	62.97	62.23
All painters	all	all	-	61.20''	60.86
Cezanne	7	2	-	64.28	62.53
All painters	7	2	-	64.71	59.32
Cezanne	3	2	-	57.96	61.79
Cezanne	3	2	yes	61.98	67.55
All painters	3	2	-	61.46	64.8
Cezanne	1	1	-	75.05	75.96
All painters	1	1	-	67.49	61.79
Cezanne	0	0	yes	80.09	88.62

Table 3: CycleGan results after 25 epochs.

StarGan			
Domain loss function	Identity_alpha	Kaggle's MiFid score	Fid score
MSE	0	118.93	111.06
MSE	1	74.8	129.63
Cross Entropy	1	118.93	99.6

Table 4: StarGan results after 25 epochs.

comparing to StarGan. After viewing those results, we trained the best model for 50 epochs and were able to get a score of 55.18 MiFid.

Data augmentation: We tried different ways of resizing the image before cropping it to its original size, each sets of values gave us different results as you can see in table 4. random increment of 10 to 50px gave the best results.

Data augmentation			
Minimum increment	Maximum increment	Kaggle's MiFid score	Fid score
10	50	67.83	68.78
10	100	57.96	61.79
30	200	86.20	70.62

Table 5: All models were trained with transfer learning from Cezanne dataset, with 2 unfrozen layers on the discriminator and 3 on the generator

Pool size: one more parameter that effects the results is the pool size of the discriminator. Because the generator might learn a pattern of faking images, we used a pool of fake images and trained the discriminator on random images

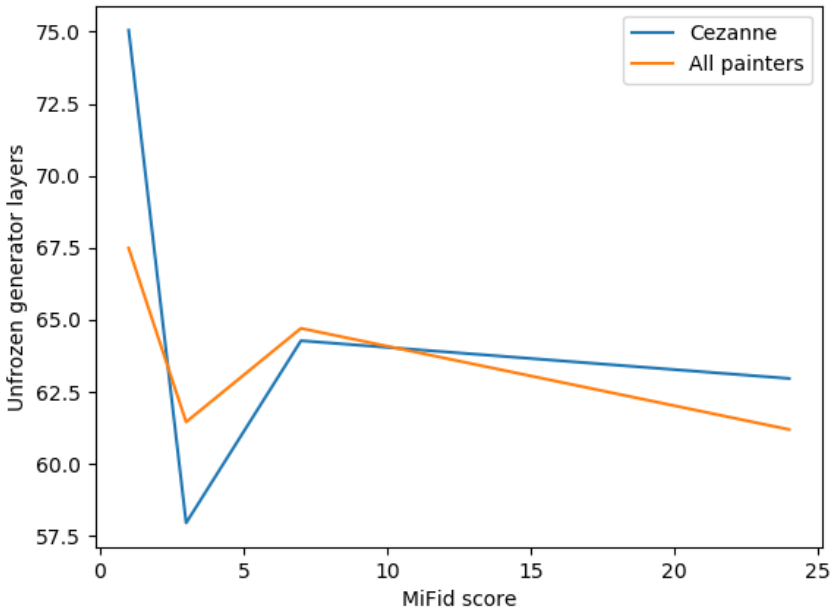

Figure 1: Number of frozen layers / MiFid score for each of the datasets

from that pool. In table 5 you can see the effect of the pool size.

Pool size	Kaggle's MiFid score	Fid score
none	59.23	67.5
1000	57.96	61.79
10000	91.46	75.78

Table 6: All models were trained with transfer learning from Cezanne dataset, with 2 unfrozen layers on the discriminator and 3 on the generator

4 Discussion

From the experiments, we came to a conclusion that transfer learning can be very helpfull when training CycleGan networks, especially when there is a lack of data from at least one domain. Furthermore, it's important to choose the right amount of unfrozen layers to train, as training one layer might not be enough and training all layers might lose important features learned in the pretrained model. One more important thing is to choose a domain which is close as possible to your domain. Combining more then one domain as we did when we combined 3 painters, might give better results when the all CycleGan framework is trained because of the bigger dataset, but worst results when only

Figure 2: best model's transformations between normal photos and Monet

a few of the layers are trained. In our opinion, it might be because the features are more accurate when learned from one domain in contrast to multiple similar domains.

As for StarGan, while we think the algorithm can be good when trying to learn domain with less data, , it takes much longer time to train because it tries to learn multiple domains transfer.

Another insight we got is that training only bayes is not good enough for CycleGan tasks, and training all the bayes layers in addition to a few full layers is giving worst results than training those few layers alone.

As for pooling size for the discriminator, we discovered that it could be helpful, but it's a low reward, high risk feature, as using a big pool can destroy the model performance and a good pool size can help the model slightly. We recommend to try multiple values for this feature when used and check it's effect on the model's FID score.

5 Further research

While there are good reasons why the StarGan framework can help when dealing with lack of data (leveraging other domains with more data), we still got worst results comparing to CycleGan. We believe that using one generator to transform photos to painters and also painters to photos made the training process underfit on our data. We thought about a hybrid solution. Because the domains contain one main domain (normal images), and 4 similar domains (4 painters), we can leverage the data boost of StarGan configuration with the dedicated generators of CycleGan, by using one generator for translating images from the main domain to the other domains like in StarGan and one generator that translates images from the other domains to the main domain. In this case, we will also use 2 discriminators, but only the painters discriminator will have a domain loss.

6 Code

Github repository:

https://github.com/eran88/CycleGan_StarGan

Colab: If you want to rerun the the training or inference please first make a shortcut in your google drive to our data:

https://drive.google.com/drive/folders/1rD62zeFpCH7RAuNnvZubaD0oaDVK9ELX?usp=sharing

Colab notebook Train:

https://colab.research.google.com/drive/1Zr58a45eBAZsVQhBiO-3lDb4PvctvxKz?usp=sharing

Colab notebook Inference:

https://colab.research.google.com/drive/1YnTgg7CKRY0uJ7GsIeBw2XsIX3xP-dzs?usp=sharing

Code attribute: When building the network we used ideas from this repository (without coping the code). We also used the FID calculation function from the article Fréchet Inception Distance function

Kaggle's team name: Eran-Eliahu

7 References

1. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair†, Aaron Courville, Yoshua Bengio: Generative Adversarial Nets, 2014
2. Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros: Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, 2020
3. Yunjey Choi1, Minje Choi1, Munyoung Kim, Jung-Woo Ha Sunghun Kim, Jaegul Choo, Korea University: StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation, 2018
4. Elad Ben-Zaken1 Shauli Ravfoge, Yoav Goldberg: BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models, 2021
5. Yael Fregier, Jean-Baptiste Gouray: Transfer Learning for GANs, 2020
6. Zhengli Zhao, Zizhao Zhang, Ting Chen, Sameer Singh, Han Zhang: Image Augmentations for GAN Training, 2020
7. Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler: GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium, 2018