BITS PILANI DUBAI CAMPUS
ACADEMIC CITY
DUBAI
15TH December 2020

Research Article

GENERATING ART LIKE CLAUDE MONET USING GENERATIVE ADVERSARIAL NETWORKS

Abhigyan Gandhi, 2018A7PS0168U
f20180168@dubai.bits-pilani.ac.in
Sandhi Reddy Tarun Teja Reddy, 2018A7PS0142U
f20180142@dubai.bits-pilani.ac.in

Submitted To: Ma'am Angel Arul Jothi, Instructor-In-Charge,
CS F415 Data Mining

Abstract - This report gives an overview of the data mining project we are currently working on. The objective of the project is to generate art like an actual artist which we will be doing using generative adversarial networks. We found this problem as a challenge on the website Kaggle. The project is really interesting as it mixes art and (data) science. It is pretty challenging to re-create art which should look like it is made by a reputable artist just by using lines of codes. Currently we are working on the development of a generator model for the creation of images. We will mainly be using Python as the base programming language for the project.

CHAPTER 1

INTRODUCTION

General introduction to the project area

Our main project area is generative adversarial networks. GANs are a deep learning based model which contain two neural networks which are used to train and test the data. The two models are a generator model and a discriminator model. The generator model is trained using the discriminator model. The two models work against each other to get an optimal output. GANs include generative modelling which is an unsupervised task in machine learning which means that we have input variables but do not have output variables, so the model is made by evaluating the input data and identifying the patterns in it to generate an output. In GANs we frame the problem as a supervised task as we include two models and train them together to classify data as real or fake (generated). The goal is to trick the discriminator for more than half of the input to get a plausible output.

The two models work differently, the generator model takes input of any random vector from a distribution of the data and creates a sample domain for the discriminator model. After the training is complete the points inside the vector correspond to the points in the domain created which creates a compact representation of the data. The vector is referred to as latent (hidden) which has got meaning by the generative process carried by the generator model, so that new points can be used by this model again to develop new and different outputs. The discriminator model simply takes input from the domain created and classifies the data as real or fake.

This model is very straightforward and upcoming in this field as it has the ability to generate realistic outputs for a wide variety of problems. The advantage of using this model is that it uses deep learning techniques which allows data augmentation resulting in better developed models which reduce the errors and increase skill.

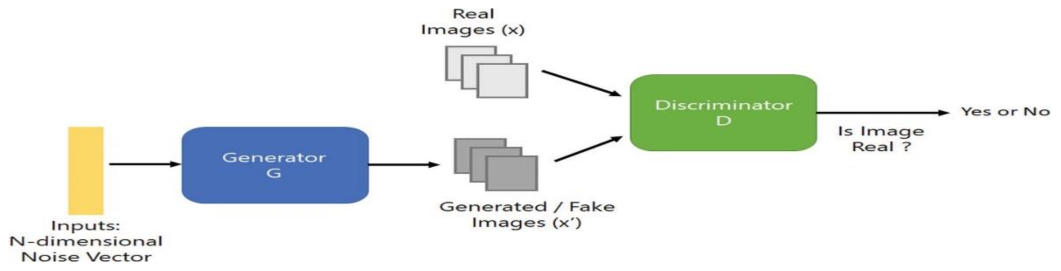


Fig 1: A representation of how GAN works.

OBJECTIVES

The Objective of our project is to take images as input and generate them as images which are in the style of the renowned French artist Claude Monet. Every artist has a unique style and way of creating art which cannot be replicated easily, but now with the use of deep learning methods we aim to transform normal images into Monet-style images. We are using generative adversarial networks about which we have explained in the previous section.

The first objective will be to develop the generator model for the network which will give a base domain for the discriminator model to classify the images. We will be using a sub-category of GANs known as CycleGAN. This is a technique which basically trains data from an image-to-image model without any paired examples. This is a powerful way to easily transform images of a type into another which are visually very appealing and a very good resemblance of the actual output required. We will use thousands of images as input data and try to get a large number of Monet-style images. A credible output will be if more than half the images generated by the generator model are classified by the discriminator model as a real image.

This is a very good learning opportunity for us to understand deep learning models and to increase our knowledge about data science by doing a meaningful and exciting project. The biggest objective for us is to make something useful and provide something of our own for others to learn from and make use of.

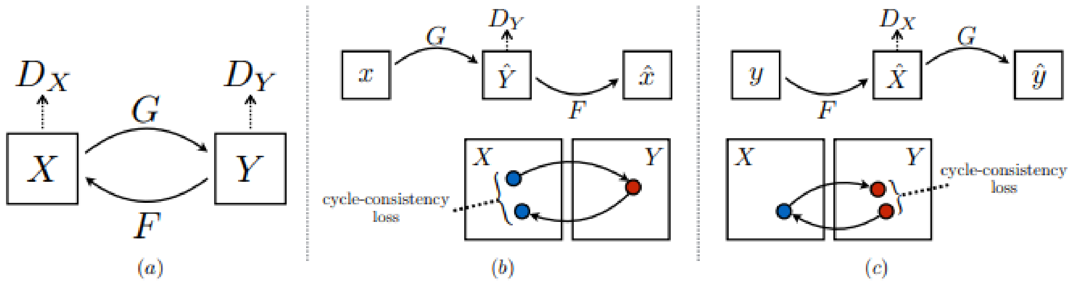


Fig 2: CycleGAN representation.

MOTIVATIONS / CHALLENGES

The project we have chosen is an ongoing competition on the website Kaggle which has a lot of data science projects, competitions and datasets. Already 156 teams are working on this problem and many have submitted their ways of doing this. We decided to go ahead with this problem simply because of the very intriguing and challenging nature of it. The problem combines art and data science which to us was really interesting. The field of data science involved in the problem is very new-age and will help us to gain knowledge about a various number of concepts about deep learning methods and neural networks.

Image-to-image translation has become popular recently and is being used in a wide variety of fields for different purposes, although the problem we have is just for beginners and just for learning purposes, we think that it can be put to real use and the model can be developed for generating a lot of new art and can also help in reviving lost and antique art forms. The motivation is very intrinsic for us and we will do our best to produce something useful and original.

The challenge is to create a GAN that will generate around 7,000-10,000 Monet-style images. The objective has been explained in detail in the previous section, but to look at it as a whole we will use generative deep learning techniques to implement data augmentation for the transformation of normal images into Monet-style images. Another challenge for us is to study about CycleGANs and to learn how to use them. We have the knowledge about basic python coding but this project will require the use of advanced python algorithms which are developed specifically for machine learning. We will use the PyTorch machine learning library which is used in the field of computer vision. It is a widely used library and makes it very easy for the implementation of machine learning techniques.

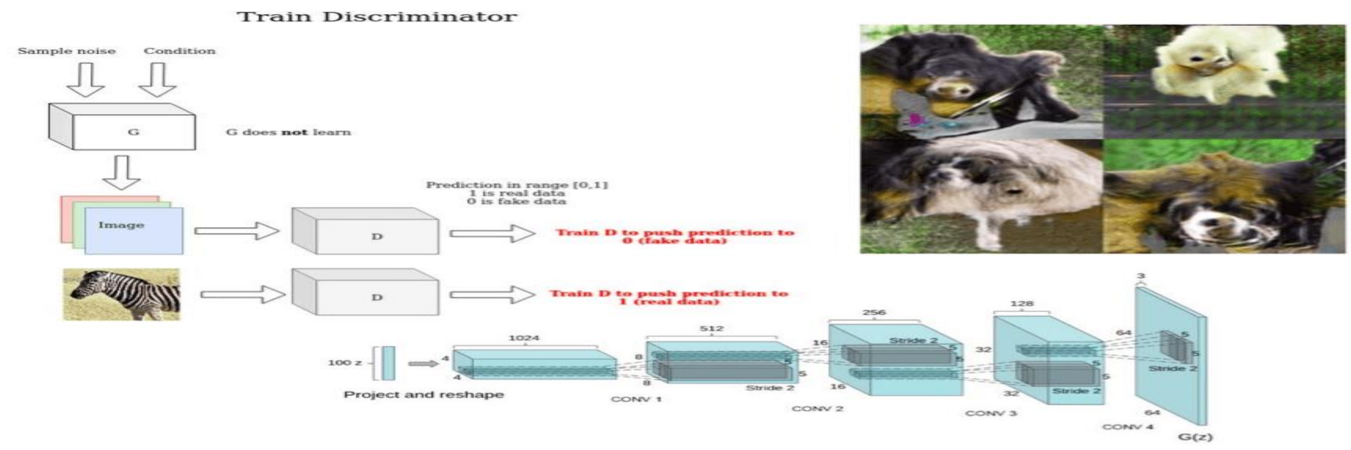


Fig 3: Image to image translation.

ORGANISATION OF THE REPORT

The first chapter contains the details about our project area, the objectives, motivations and challenges of the project. We briefly explained about the project area which is generative adversarial networks – a generative unsupervised model which uses deep learning methods to implement data augmentation and is mainly used for image-to-image translation.

The main objective of the project is to convert normal images to Monet-style images by using machine learning techniques. The project is pretty challenging for us as we do not have any experience in this field and therefore we have a self-developed motivation for gaining knowledge about the field and completing the project.

The second chapter is the main part of this report which incorporates the literature survey we did about the different research papers from journals and conferences in the same field of our projects. By reading and interpreting the papers we got to learn about GANs and CycleGANs a lot more and it helped us to understand how they work and what challenges we will face during the creation of the generator and discriminator model. We read around 30 different papers from the past 5 years and all the papers had something new to offer. The newer papers were able to overcome the difficulties faced in the older projects and gave a sense of belief that we could also overcome the complexities revolving around the different aspects of the project.

The third chapter simply informs you about the dataset which we will be using for the project. We found the dataset on Kaggle.com and it contains some paintings of Monet and thousands of normal images which we will try to transform into Monet-style images. After that we have explained how we implemented the CycleGAN for the project.

The fourth and final chapter covers the results we found on testing our model and how we reduced the errors by using appropriate optimizers, and the conclusion of the whole experience of doing this project.

CHAPTER 2

LITERATURE SURVEY

We surveyed research papers from the past 4-5 years related to generative adversarial networks. People have been using GANs for various different purposes including satellite imagery, image colorization, multi-domain image translation and many more. The common trend we found in all the papers was that everyone was focusing on realistic outputs and overall better performing models. GANs are simple deep learning models which have two sub-models, generator model and discriminator model. They work against each other to generate images and try to distinguish them into real and fake. The most common category of GANs we found which was used in a significant amount of research were the CycleGANs. These were used for super-resolution imagery, changing emotions of an image, shape-agnostic image translation, etc.. Some of the notable researchers proposed in the field of CycleGANs were the following, Lu et al. proposed an identity-guided conditional CycleGAN to translate low-resolution face images to high-resolution face images. Ehsan Hosseini-Asl et al. proposed a multi-discriminator CycleGAN model for speech domain adaptation. Hiasa et al. extend CycleGAN by adding a gradient-consistency loss to improve accuracy at the boundaries of MRI and CT image synthesis. Choi et al. introduced StarGAN, a cycle-consistent adversarial network that performs multi-domain image-to-image translation using a single generator and a single discriminator using multiple datasets.
Our project is also based on image-to-image translation, what we require to do is that convert an image by adding textures and styles to it, a lot of work has been done in this specific field, most of them used unpaired data because we saw that earlier GANs were being used as a supervised task and paired data was used, but to reduce the loss caused by paired data input, unsupervised GAN frameworks were introduced. Another thing which we found was the use of multiple generator models for cross-domain and multi-domain image translation, we observed that GANs are not that much capable for handling more than two domains at once. The newer models are now able to overcome that difficulty and use just a single model for multi-domain purposes like the StarGAN proposed for multi-domain image-to-image translation.

After reviewing the papers it was pretty clear to us about the advancements that have been made in this field. The most prominent and significant impact of GANs has been in computer vision, which is the up and coming field for research. It is a field of artificial intelligence which deals with training computers to understand and interpret the visual world. Image translation, increasing resolution of images, identifying and classifying images all come under computer vision.

The use of GANs has increased hugely in recent times, and they are being used for a lot of other applications like fashion, art, video games, etc.. We found a discussion regarding the misuse of GANs and what challenges have risen. People are using GANs for human image synthesis for menacing purposes by creating fake and incriminating images and videos. For example there is a GAN model called Speech2Face which can reconstruct an image of a person's face after listening to their voice, this can be used in many wrong ways. Some states have taken up this issue and even banned the use of such software for stopping such problems.
For our project we will only be needing a single generator model to process the image as we are working with a single domain of normal images. We got a lot of inspiration for the previous work and are fully determined towards creating our own model and completing the project.

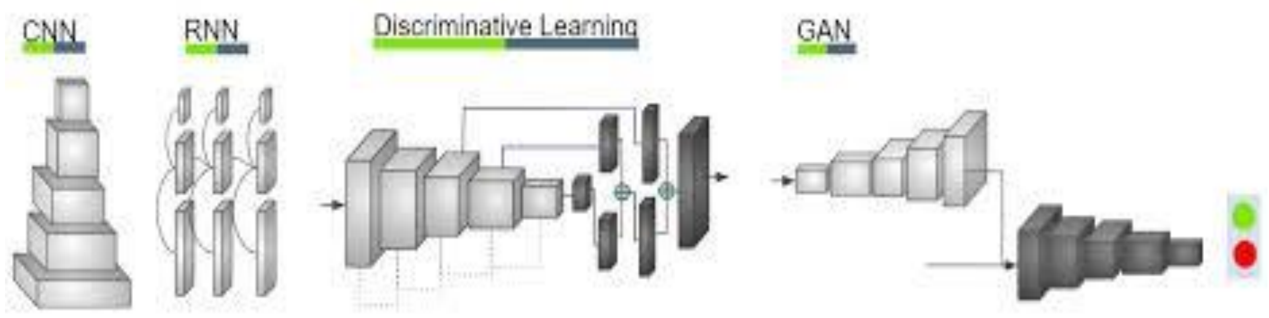


Fig 4: Different types of neural networks.

CHAPTER 3

METHODOLOGY

DATASET DESCRIPTION

The dataset contains two main components, the first are the Monet style images, and the second are the normal images. There are four directories: monet_tfrec, monet_jpg, photo_tfrec, photo_jpg. The monet_tfrec and monet_jpg directories contain the same image files, they are just of different formats TFRecords and JPG images same goes for the photo_tfrec and photo_jpg directories. The monet directories contain Monet paintings which we will use to train our model. The photo directories are images to which we will apply Monet-style.

Files:
monet_jpg - 300 Monet paintings sized 256x256 in JPEG format
monet_tfrec - 300 Monet paintings sized 256x256 in TFRecord format
photo_jpg - 7028 photos sized 256x256 in JPEG format
photo_tfrec - 7028 photos sized 256x256 in TFRecord format

Link to the dataset: https://www.kaggle.com/c/gan-getting-started/data

IMPLEMENTATION DETAILS

For this project we have implemented a CycleGAN for image translation. To learn the basics of using this model we used a pre-made kernel as a reference and made our model based on that. The methodology was fairly simple, first we loaded the data which was in TFRecords format. TFREC is a file format used to store binary strings. Image datasets can be converted to this format, stored and used easily. After loading the dataset, the images were extracted and stored again.

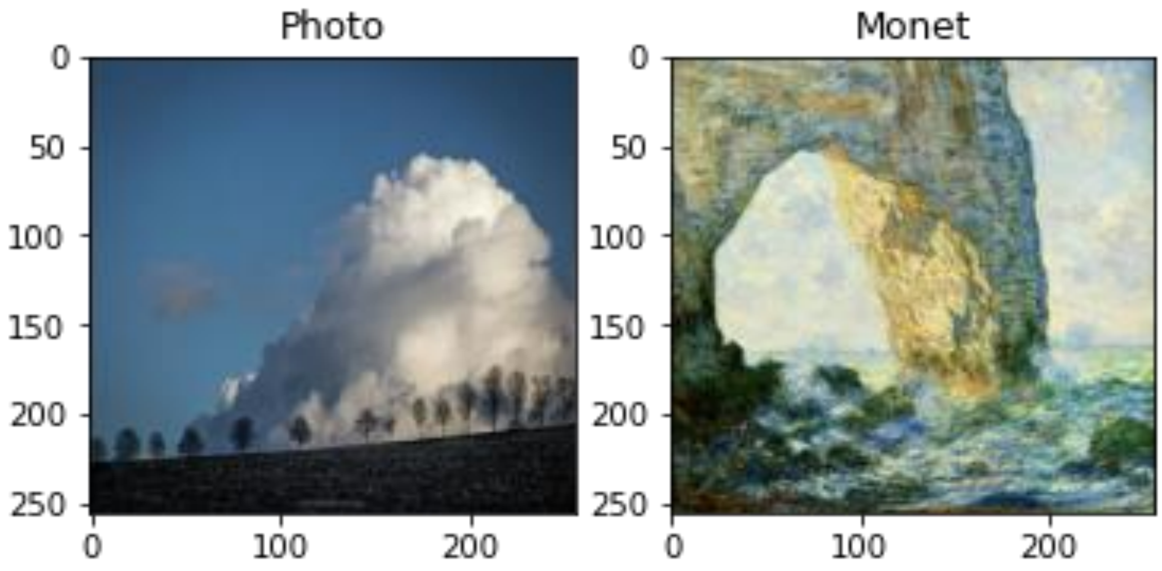


Fig 5: Examples of normal and Monet images.

Then the generator was built which converts normal photos to Monet-esque, then these images are sent to the discriminator which differentiates between real and generated images. This model is then trained using the CycleGAN class which automatically trains the data in an unsupervised fashion without requiring paired data items. We have used the TensorFlow API for this project which is a simple machine learning library which has a lot of useful features for training deep networks. For building the generator we used the UNET architecture.

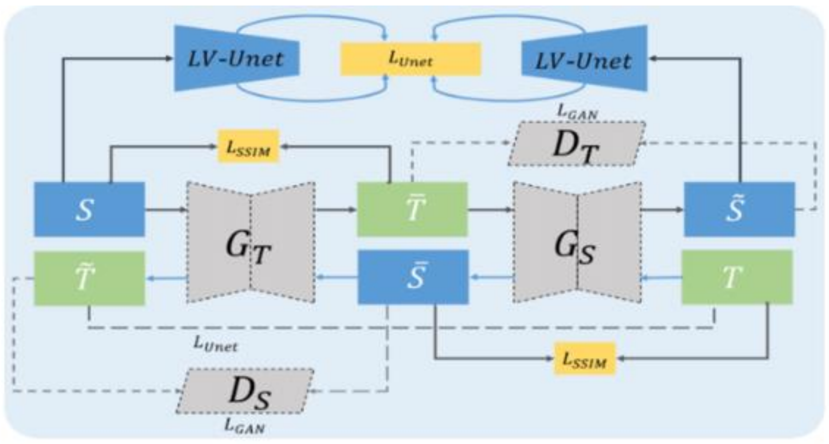


Fig 6: UNET-GAN architecture for CycleGAN.

There are two methods used inside a generator function, downsample and upsample. The first method is used to reduce the size of the images, the height and width. We decreased it by 2. We used instance normalization for style transferring, this helps in increasing the convergence of output layers during training. Instance normalization can be used from the TensorFlow-addons library. The downsample method uses normalization and the function Conv2D() to decrease the size and add the layer to the result. Then the upsample method uses the function Conv2DTranspose to increase the dimensions by 2 and stores the layers in result. Then the generator function passes the images in batches to the downsample function followed by sending them to the upsample function. Then the output layers of the downsample function are concatenated to the upsample layers one by one. This completes the generating of images. Then the discriminator comes into play which has a simple function. It takes an image input and classifies it as real or fake. If the image is real it outputs a 2D image with high resolution and if it is fake it outputs a low resolution 2D image. Next we built the CycleGAN using tf.keras.model which is a class in the TensorFlow library which helps us to group the layers into one object which contains the features we have applied during training. This model is more useful because when we subclass it we can configure the model with losses and easily train it by using the functions model.compile() and model.fit() respectively. In the CycleGAN class we define the generators and discriminators and some losses which will help us to estimate the correctness of our outputs.

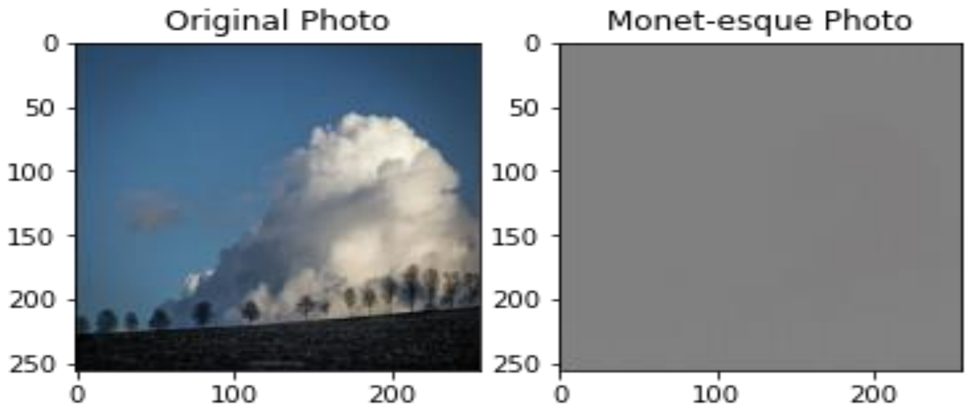


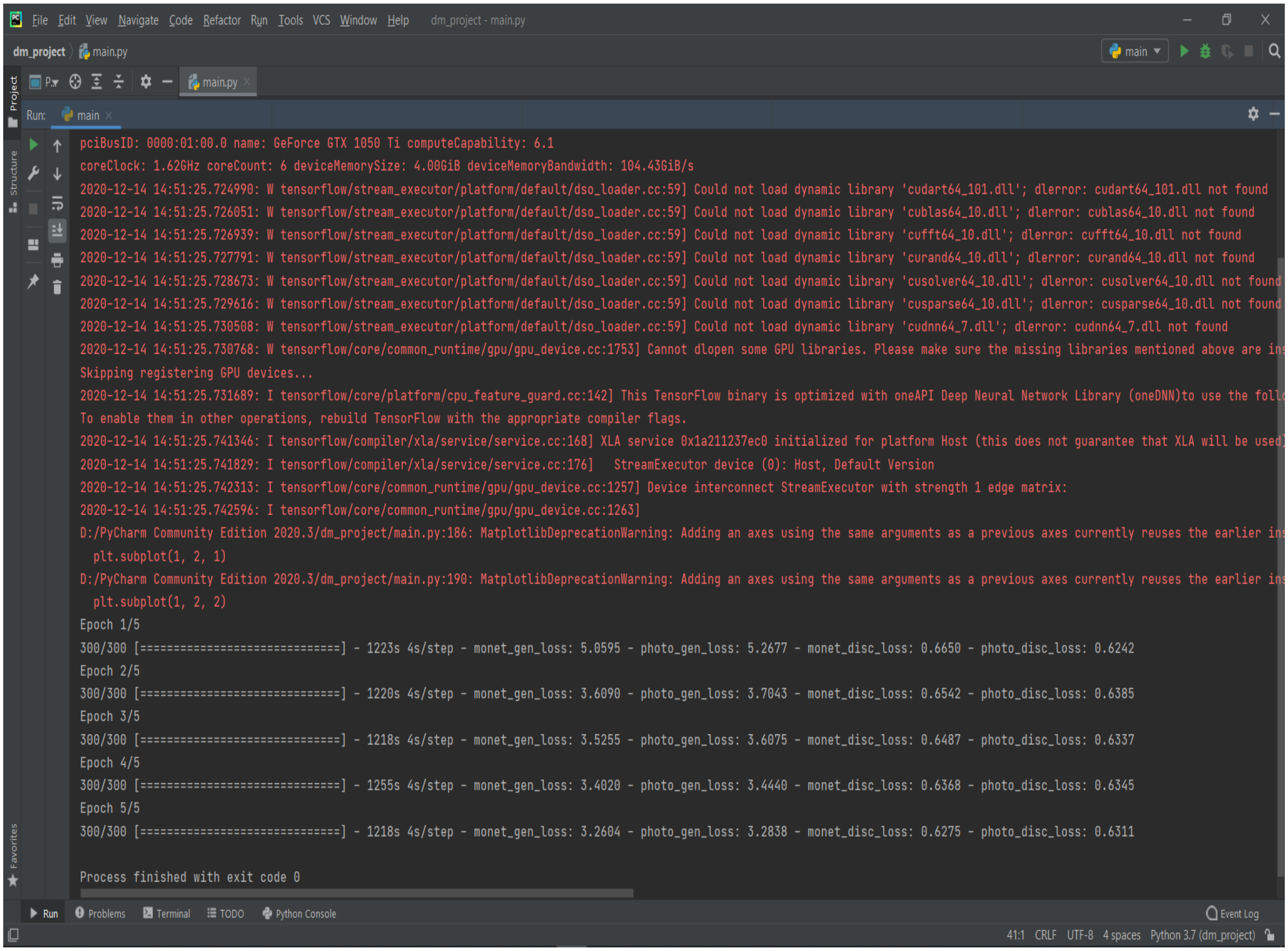
Fig 7: Photos before training the model.

We train the model by using a method in which we translate an image to Monet-esque style then translate it back to normal and see the difference between the two to find out how much is the loss. If they come out to be pretty similar then it means that the model is giving feasible output. After training the losses and the gradients which are applied to the optimizers.
We are taking four types of losses, discriminator loss, generator loss, cycle loss and identity loss. The discriminator loss function calculates the average of the real and generated loss. Ideally the discriminator should give 1 for a real image and 0 for a fake image so, both are respectively compared to matrices of 1's and 0's. The generator loss function compares the generated images to a matrix of 1's as the generator wants to trick the discriminator into believing that the generated images are real. The cycle loss function compares the input image to the image we get after transforming it twice, first generated and then normal again and computes the average of their difference. The identity loss function compares the input image to the generator with its output to find out how much difference is there between them. Lastly we use the model.fit() function to train our model , we took epoch to be 5 only as it is a heavy process and took nearly 26 minutes to complete one epoch.

CHAPTER 4

RESULTS AND CONCLUSIONS

After running the model for 5 epochs we found the following output log:


File Edit View Navigate Code Refactor Run Tools VCS Window Help dm_project - main.py
dm_project
main.py
main
Run:
Project
Structure
Favorites
Problems
Terminal
TODO
Python Console
Event Log
41:1 CRLF UTF-8 4 spaces Python 3.7 (dm_project)

```
pciBusID: 0000:01:00.0 name: GeForce GTX 1050 Ti computeCapability: 6.1
coreClock: 1.62GHz coreCount: 6 deviceMemorySize: 4.00GiB deviceMemoryBandwidth: 104.43GiB/s
2020-12-14 14:51:25.724990: W tensorflow/stream_executor/platform/default/dso_loader.cc:59] Could not load dynamic library 'cudart64_101.dll'; dlerror: cudart64_101.dll not found
2020-12-14 14:51:25.726051: W tensorflow/stream_executor/platform/default/dso_loader.cc:59] Could not load dynamic library 'cublas64_10.dll'; dlerror: cublas64_10.dll not found
2020-12-14 14:51:25.726939: W tensorflow/stream_executor/platform/default/dso_loader.cc:59] Could not load dynamic library 'cufft64_10.dll'; dlerror: cufft64_10.dll not found
2020-12-14 14:51:25.727791: W tensorflow/stream_executor/platform/default/dso_loader.cc:59] Could not load dynamic library 'curand64_10.dll'; dlerror: curand64_10.dll not found
2020-12-14 14:51:25.728673: W tensorflow/stream_executor/platform/default/dso_loader.cc:59] Could not load dynamic library 'cusolver64_10.dll'; dlerror: cusolver64_10.dll not found
2020-12-14 14:51:25.729616: W tensorflow/stream_executor/platform/default/dso_loader.cc:59] Could not load dynamic library 'cusparse64_10.dll'; dlerror: cusparse64_10.dll not found
2020-12-14 14:51:25.730508: W tensorflow/stream_executor/platform/default/dso_loader.cc:59] Could not load dynamic library 'cudnn64_7.dll'; dlerror: cudnn64_7.dll not found
2020-12-14 14:51:25.730768: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1753] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are ins
Skipping registering GPU devices...
2020-12-14 14:51:25.731689: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN)to use the foll
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2020-12-14 14:51:25.741346: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x1a211237ec0 initialized for platform Host (this does not guarantee that XLA will be used)
2020-12-14 14:51:25.741829: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecutor device (0): Host, Default Version
2020-12-14 14:51:25.742313: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1257] Device interconnect StreamExecutor with strength 1 edge matrix:
2020-12-14 14:51:25.742596: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1263]
D:/PyCharm Community Edition 2020.3/dm_project/main.py:186: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier ins
  plt.subplot(1, 2, 1)
D:/PyCharm Community Edition 2020.3/dm_project/main.py:190: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier ins
  plt.subplot(1, 2, 2)
Epoch 1/5
300/300 [==============================] - 1223s 4s/step - monet_gen_loss: 5.0595 - photo_gen_loss: 5.2677 - monet_disc_loss: 0.6650 - photo_disc_loss: 0.6242
Epoch 2/5
300/300 [==============================] - 1220s 4s/step - monet_gen_loss: 3.6090 - photo_gen_loss: 3.7043 - monet_disc_loss: 0.6542 - photo_disc_loss: 0.6385
Epoch 3/5
300/300 [==============================] - 1218s 4s/step - monet_gen_loss: 3.5255 - photo_gen_loss: 3.6075 - monet_disc_loss: 0.6487 - photo_disc_loss: 0.6337
Epoch 4/5
300/300 [==============================] - 1255s 4s/step - monet_gen_loss: 3.4020 - photo_gen_loss: 3.4440 - monet_disc_loss: 0.6368 - photo_disc_loss: 0.6345
Epoch 5/5
300/300 [==============================] - 1218s 4s/step - monet_gen_loss: 3.2604 - photo_gen_loss: 3.2838 - monet_disc_loss: 0.6275 - photo_disc_loss: 0.6311

Process finished with exit code 0
```

Fig 8: Output log.

If we take the averages of all the losses over the 5 epochs we get:

Average Monet generator loss = 3.7713
Average photo generator loss = 3.8615
Average Monet discriminator loss = 0.6464
Average photo discriminator loss = 0.6324

We notice that the errors of the discriminators are fairly less which means that the generated images are very Monet-like and it is difficult for the discriminator to distinguish between them. The generator losses are around 3-4 which is fine for a standard generator model.
We also notice that with iterations taking place the error is getting reduced which is a positive result.

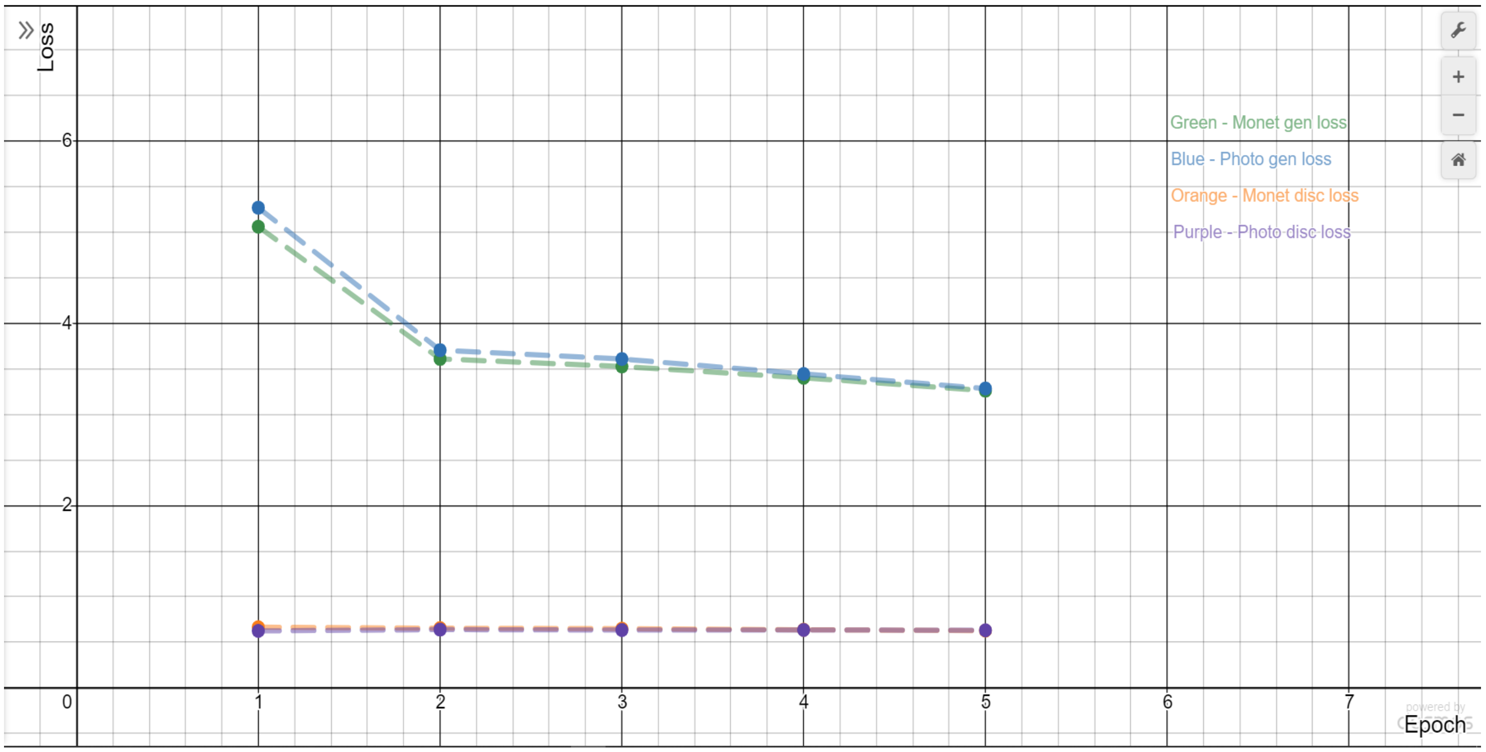


Fig 9: Loss vs Epoch graph for all the losses.

For the optimizers in the CycleGAN class, we have used four different optimizers for each attribute, this helps in both networks to get better.
We have used the Adam optimizer for estimation. This optimizer is very useful as after every epoch it changes the learning rate, bias and weights, this helps in achieving a good result. Following is the visualization of some images:

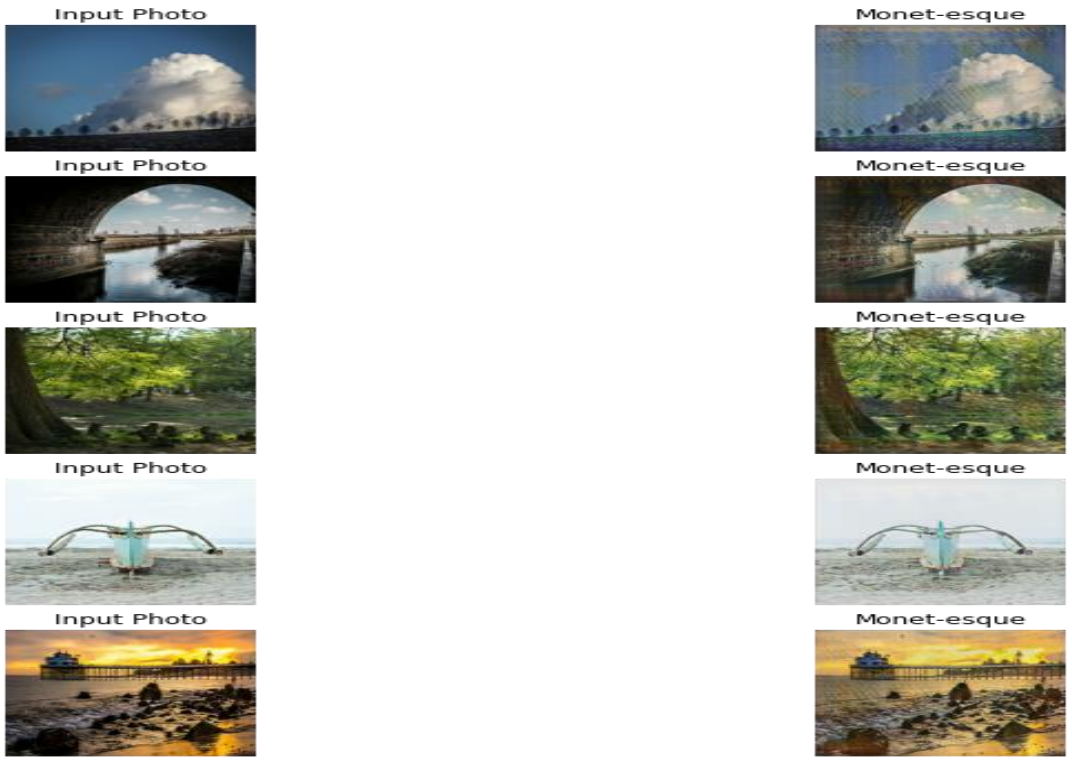


Fig 10: Some input images and their outputs.

We can conclude that our model is working viably with reducing error as iterations increase. The output images we are getting are pretty Monet-like. A lot of models have been proposed for image translation tasks, we found that the CycleGAN model is very reliable and practical for the same. It is very easy to understand and implement .
Working on this project was a very good opportunity for us as we have gained a tremendous amount of knowledge about deep learning techniques, TensorFlow, UNET architecture and working with GANs which has only helped us to develop our coding skills and understand machine learning even better.

REFERENCES

1. Yongyi Lu, Yu-Wing Tai, and Chi-Keung Tang, “Attribute-guided face generation using conditional cyclegan,” arXiv preprint arXiv:1705.09966v2, 2018.

2. Ehsan Hosseini-Asl, Yingbo Zhou, and Richard Socher, “A multi-discriminator cyclegan for unsupervised nonparallel speech domain adaptation,” arXiv preprint arXiv:1804.00522v4, 2018.

3. Yunjey Choi, Minje Choi, Munyoung Kim, JungWoo Ha, Sunghun Kim, and Jaegul Choo, “Stargan: Unified generative adversarial networks for multidomain image-to-image translation,” arXiv preprint arXiv:1711.09020v3, 2018.

4. Ram Longman and Raymond Ptucha, “EMBEDDED CYCLEGAN FOR SHAPE-AGNOSTIC IMAGE-TO-IMAGE TRANSLATION”, IEEE Xplore, ICIP 2019.

5. B. Chang, Q. Zhang, S. Pan and L. Meng, "Generating Handwritten Chinese Characters Using CycleGAN," 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), Lake Tahoe, NV, 2018, pp. 199-207, doi: 10.1109/WACV.2018.00028.

6. H. Zhang et al., "StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 41, no. 8, pp. 1947-1962, 1 Aug. 2019, doi: 10.1109/TPAMI.2018.2856256.

7. J. Chen, L. Wang, R. Feng, P. Liu, W. Han and X. Chen, "CycleGAN-STF: Spatiotemporal Fusion via CycleGAN-Based Image Generation," in IEEE Transactions on Geoscience and Remote Sensing, doi: 10.1109/TGRS.2020.3023432.

8. Y. Xiao, A. Jiang, C. Liu and M. Wang, "Single Image Colorization Via Modified Cyclegan," 2019 IEEE International Conference on Image Processing (ICIP), Taipei, Taiwan, 2019, pp. 3247-3251, doi: 10.1109/ICIP.2019.8803677.

9. W. Zheng, L. Yan, W. Zhang, C. Gou and F. Wang, "Guided Cyclegan Via Semi-Dual Optimal Transport for Photo-Realistic Face Super-Resolution," 2019 IEEE International Conference on Image Processing (ICIP), Taipei, Taiwan, 2019, pp. 2851-2855, doi: 10.1109/ICIP.2019.8803393.

10. R. Longman and R. Ptucha, "Embedded Cyclegan For Shape-Agnostic Image-To-Image Translation," 2019 IEEE International Conference on Image Processing (ICIP), Taipei, Taiwan, 2019, pp. 969-973, doi: 10.1109/ICIP.2019.8803082.

11. Sicheng Zhao, Chuang Lin, Pengfei Xu, Sendong Zhao, Yuchen Guo, Ravi Krishna, Guiguang Ding, Kurt Keutzer, “CycleEmotionGAN: Emotional Semantic Consistency Preserved CycleGAN for Adapting Image Emotions”, Proceedings of the AAAI conference on Artificial Intelligence, 2019.

12. Casey Chu, Andrey Zhmoginow, Mark Sandler, “CycleGAN, a master of stenography”, arVix.org, arVix:1712.02950, 2017.

13. Problem question, https://www.kaggle.com/c/gan-getting-started/overview/description, 2020.

14. How to write a literature review, “https://www.scribbr.com/dissertation/literature-review/”, 2019.

15. Per Welander, Simon Karlsson, Anders Eklund, “Generative Adversarial Networks for Image-to-Image Translation on Multi-Contrast MR Images - A Comparison of CycleGAN and UNIT”, arVix.org, arVix:1806.07777, 2018.

16. Joonyoung Song, Jae-Heon Jeong, Dae-Soon Park, Hyun-Ho Kim, Doo-Chun Seo, Jong Chul Ye, “Unsupervised Denoising for Satellite Imagery using Wavelet Subband CycleGAN”, JOURNAL OF LATEX CLASS FILES, VOL. X, NO. X, XXX 2020.