I AM SOMETHING OF A PAINTER MYSELF

INDIVIDUAL REPORT

Author: Vishal Pathak

DATS6203: Machine Learning II

The George Washington University

December 7, 2020

Table Of Contents

1. Introduction _____

2. Data Set Details _____

3. Network Architecture_____

4. Experimental Setup_____

    a. Description of Individual work ----------

5. Results_____

6. Conclusion_____

7. References-_____

INTRODUCTION

I'm Something of a Painter Myself Image-to-image translation has been an increasingly popular topic over the last years. One Sample of such a task is art style transfer. Style transfer algorithms in the context of art try to capture the general style of an artist or an image and apply it to one or many content pictures. As an example, think of the latest holiday picture and try to imagine how Monet or Van Gogh would have painted the scene. So can Data science, in the form of GANs, trick classifiers into believing that the image created is a true Monet

DATASET DETAIL

The dataset contains four directories: monet_tfrec, photo_tfrec, monet_jpg, and photo_jpg. The monet_tfrec and monet_jpg directories contain the same painting images, and the photo_tfrec and photo_jpg directories contain the same photos.

The monet directories contain Monet paintings.

The photo directories contain photos

- 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

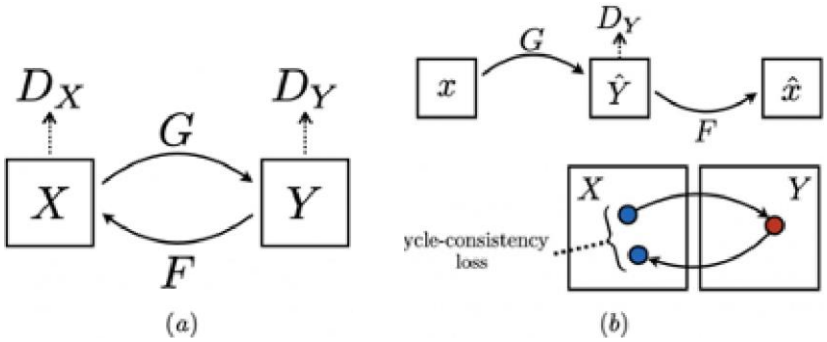
- photo_tfrec - 7028 photos sized 256x256 in TFRecord format

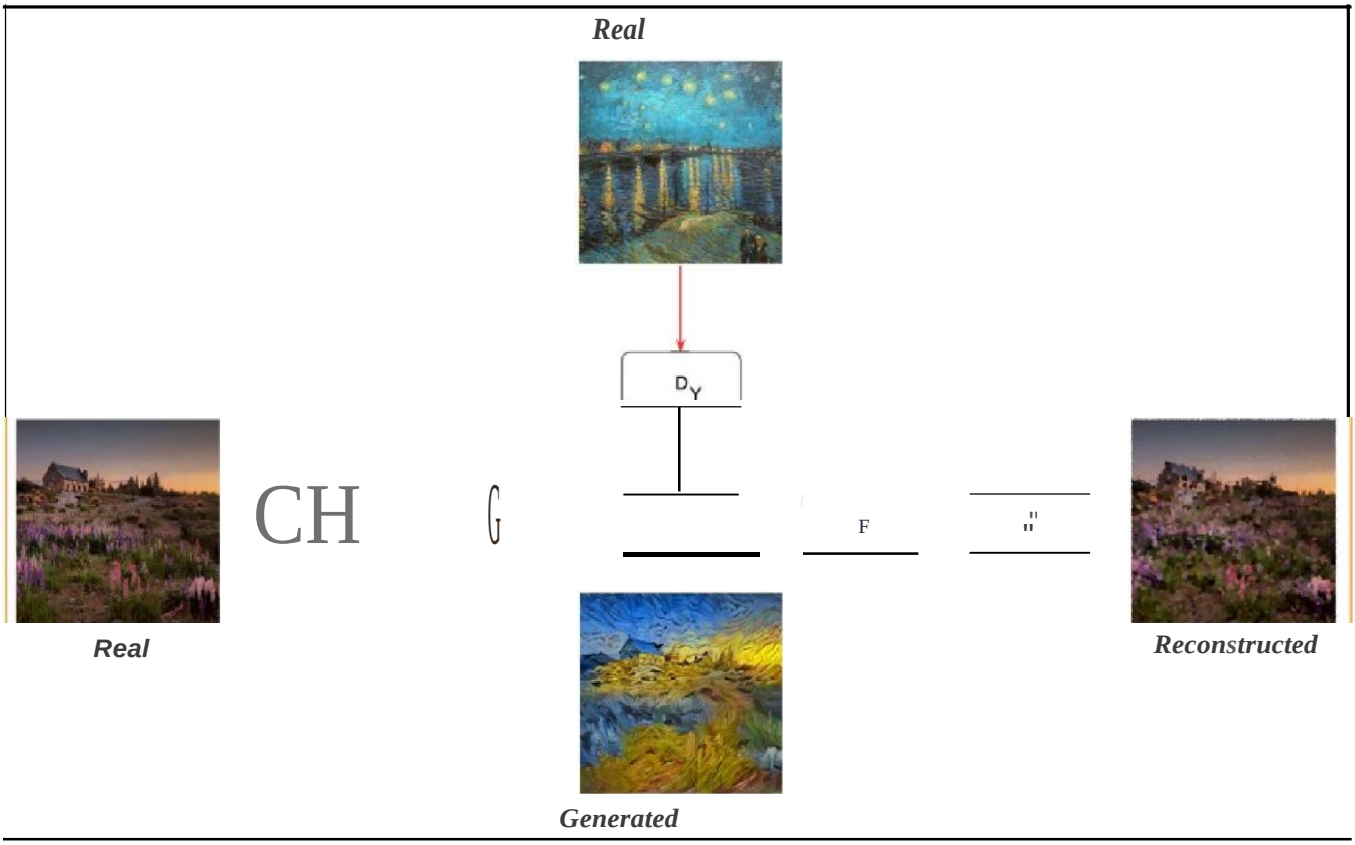
NETWORK ARCHITECTURE

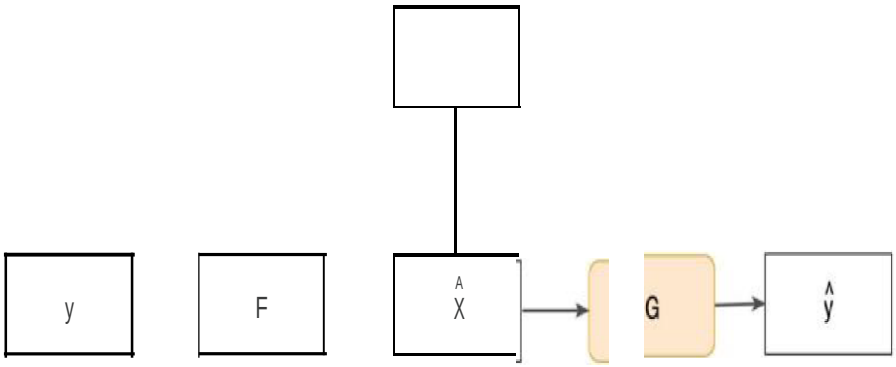
A generative adversarial network(GAN) is a type of machine learning architecture to generate new photographs which have realistic characteristics. It consists of two models: a generator model and a discriminator model. The generator takes a point from a latent space as input and generates new plausible images from the domain and the discriminator takes an image as input and predicts whether it is real or fake. Both models are trained and expected to generate more realistic images.

The CycleGAN is an extension of the GAN architecture that contains two generators and two discriminators. In this case, we can summarize the generator and discriminator models as follows:

- Generator G: Convert real photos to Monet style paintings
- Discriminator D_X: Identify real Monet paints or fake Monet paintings generated by G
- Generator F: Convert real Monet paintings to photos
- Discriminator D_Y: Identify real photos or fake photos generated by F


D_X
D_Y
G
X
Y
F
(a)
x
G
D_Y
$\hat{Y}$
$\hat{x}$
F
X
Y
ycle-consistency
loss
(b)


Real
Real
CH
G
D Y
F
"
Reconstructed
Generated


y
F
A
X
G
^
y

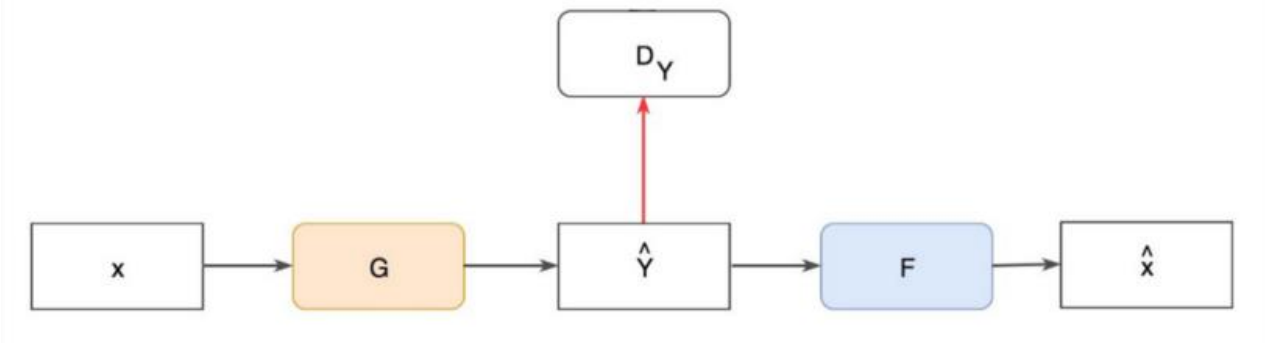


Figure 1. Network architecture

The loss of CycleGAN contains adversarial loss and cycle consistency loss.

Adversarial loss is a standard GAN loss. In this case we have 2 adversarial loss:

Cycle consistency loss compares an input photo to the Cycle GAN to the generated photo and calculates the difference between the two:

The whole loss is:

In this case, the architecture we use different models in tensorflow and pytorch for Generator and Discriminator.

EXPERIMENTAL SETUP

This section puts forwards the research methods that have been applied

to classify images.

DATA PREPROCESSING

As images were already in 3* 256*256 format so the image was resized to 128* 128 , furthermore it was scaled as well as transposed . These methods were applied to both sets of images i.e monet & photos. The images were resized and standardized in the training set while retaining color for the images. This process only cuts out the noise in the outer part of the painted images.

DISCRIPTION OF INDIVIDUAL WORK

DISCRIMINATOR

For Discriminator we have used 5 convolution layers . The Inner layers are followed by batch normalization. The Exit layer is a linear layer with 1 output neuron. In the forward function, we have used the same Leaky ReLU activation function.

Layer	Output Neuron	Kernel size	Stride	BatchNormal
Conv2d	64	4	2	FALSE
Conv2d	128	4	2	TRUE
Conv2d	256	4	2	TRUE
Conv2d	512	4	2	TRUE

Conv2d	512	4	2	FALSE
Linear	1	NA	NA	NA

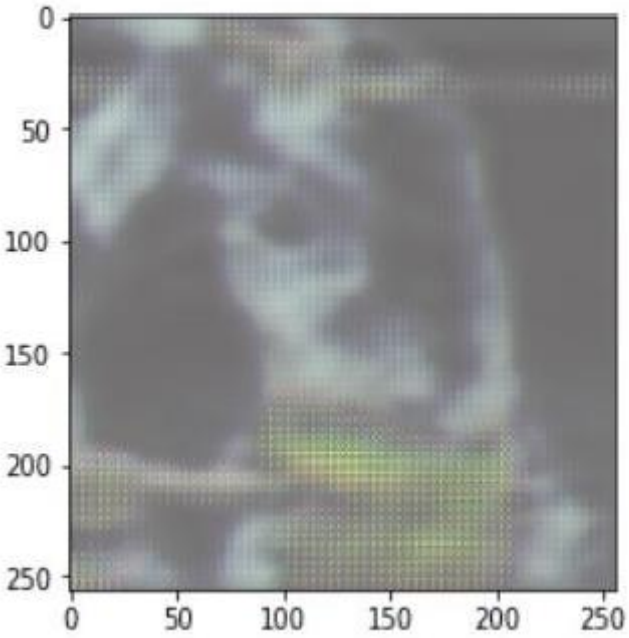
```
class disc(nn.Module):
    def __init__(self):
        super(disc, self).__init__()

        self.conv1 = nn.Conv2d(3, 64, kernel_size=4, stride=2)

        self.conv2 = nn.Conv2d(64, 128, kernel_size=4, stride=2)
        self.bn2 = nn.BatchNorm2d(128)

        self.conv3 = nn.Conv2d(128, 256, kernel_size=4, stride=2)
        self.bn3 = nn.BatchNorm2d(256)

        self.conv4 = nn.Conv2d(256, 512, kernel_size=4, stride=2)
        self.bn4 = nn.BatchNorm2d(512)

        self.conv5 = nn.Conv2d(512, 512, kernel_size=2, stride=1)

        self.head = nn.Linear(512, 1)

    def forward(self, input):
        x = Fn.leaky_relu(self.conv1(input), negative_slope=0.2)
        x = Fn.leaky_relu(self.bn2(self.conv2(x)), negative_slope=0.2)
        x = Fn.leaky_relu(self.bn3(self.conv3(x)), negative_slope=0.2)
        x = Fn.leaky_relu(self.bn4(self.conv4(x)), negative_slope=0.2)
        x = Fn.leaky_relu(self.conv5(x), negative_slope=0.2)

        x = x.view(x.size(0), -1)
        x = self.head(x)

        return torch.sigmoid(x)
```

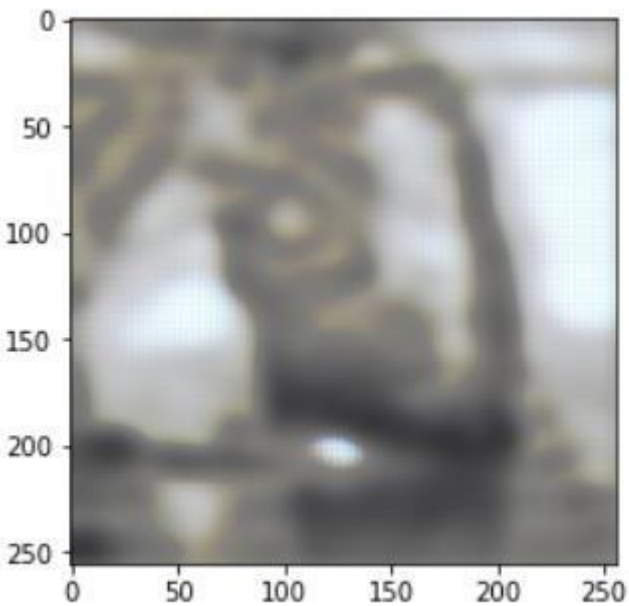
Figure 3 - Discriminator

GENERATOR

For the Generation layer, 18 layers were added in which most of the layers are 2D convolutional. In the last of the middle layers we have used the 2D Transposed Convolutional layer to generate an output feature map which has a spatial dimension greater than that of the input. In most of our layers we also added Reflection pad to add boundaries of certain length. This is useful as it ensures that the output will transition smoothly into the padding.

Reflection PAD 2D	Layer	Output Neuron	Kernel size	Stride	Padding	BatchNormal
TRUE	Conv2d	32	7	1	0	TRUE
FALSE	Conv2d	64	2	1	1	TRUE
FALSE	Conv2d	128	3	2	1	TRUE
TRUE	Conv2d	128	3	1	0	TRUE
TRUE	Conv2d	128	3	1	0	TRUE
TRUE	Conv2d	128	3	1	0	TRUE
TRUE	Conv2d	128	3	1	0	TRUE
TRUE	Conv2d	128	3	1	0	TRUE
TRUE	Conv2d	128	3	1	0	TRUE
TRUE	Conv2d	128	3	1	0	TRUE
TRUE	Conv2d	128	3	1	0	TRUE
TRUE	Conv2d	128	3	1	0	TRUE
TRUE	Conv2d	128	3	1	0	TRUE
TRUE	Conv2d	128	3	1	0	TRUE
TRUE	Conv2d	128	3	1	0	TRUE
TRUE	ConvTrans2d	64	3	2	1	TRUE
TRUE	ConvTrans2d	32	3	2	1	TRUE
TRUE	Conv2d	3	7	1	0	NA





As we can see the cyclic images are not very clear , this may be due to any number of factors , such as non-patch discriminator usage , excessive usage of convolutions layers or may be due to not so much preprocessing. However, we did all the things in our final model which was presented during presentation which has (encoder -resnet- decoder). Also for that preprocessing , for our final code , we did all the mandatory cleaning steps like transpose , resize and rescale , flip . we also used Instance Normalization instead of Batch normalization in our final submission(TensorFlow).

CONCLUSION

- Preprocessing steps like flipping, scaling, resizing, re-centering is very necessary for a model to perform better.
- Usage of Particular Normalization depends on problem to problem , we need to have better understanding of every type of normalization.
- Resnet sometimes does wonder if used at proper space.
- Learning a model and finetuning was one of the toughest task for me , it actually requires deep knowledge of domain and models

REFRENCES

- https://medium.com/coding-blocks/introduction-to-cyclegans-1dbdb8fbe781
- https://arxiv.org/abs/1703.10593