I AM SOMETHING OF A PAINTER MYSELF

Kaiqi Yu
Vishal Pathak
Zichu Chen
Sanat Lal

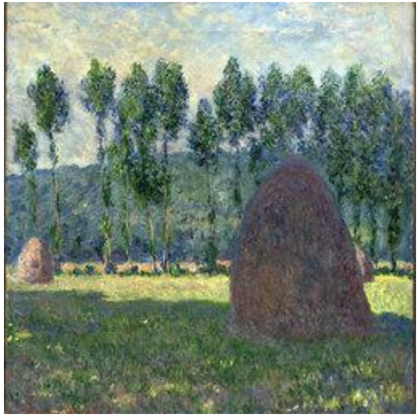
Introduction

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.

TABLE OF CONTENTS

DATASET DETAILS

Monet directories contain 300 Monet paintings sized 256x256
Photo directories contain 7038 photos sized 256x256



NETWORK ARCHITECTURE

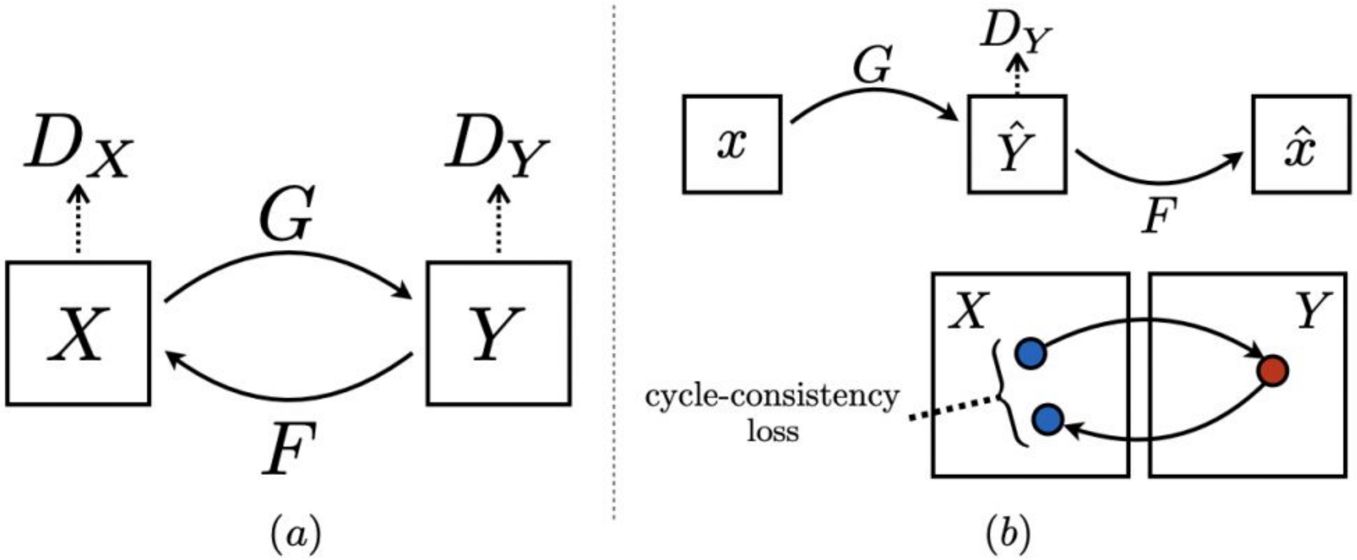
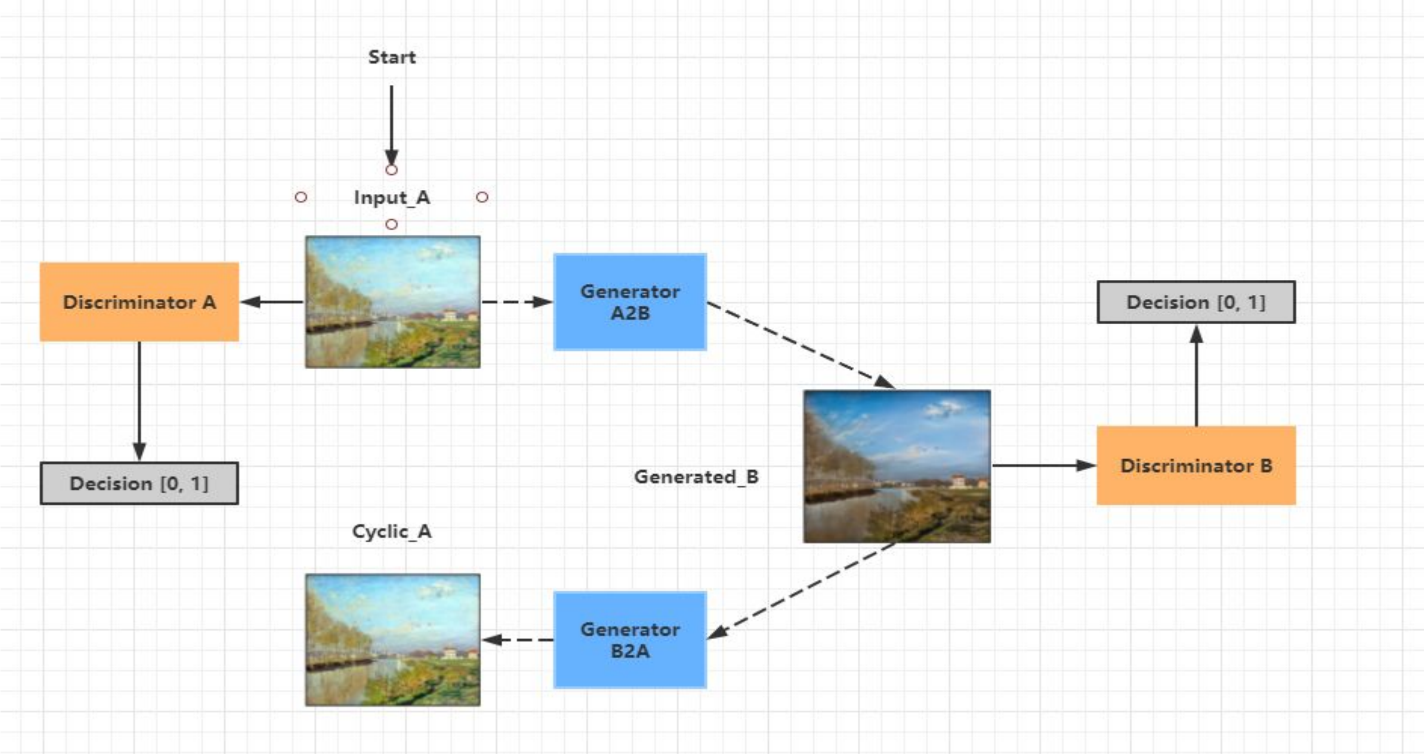
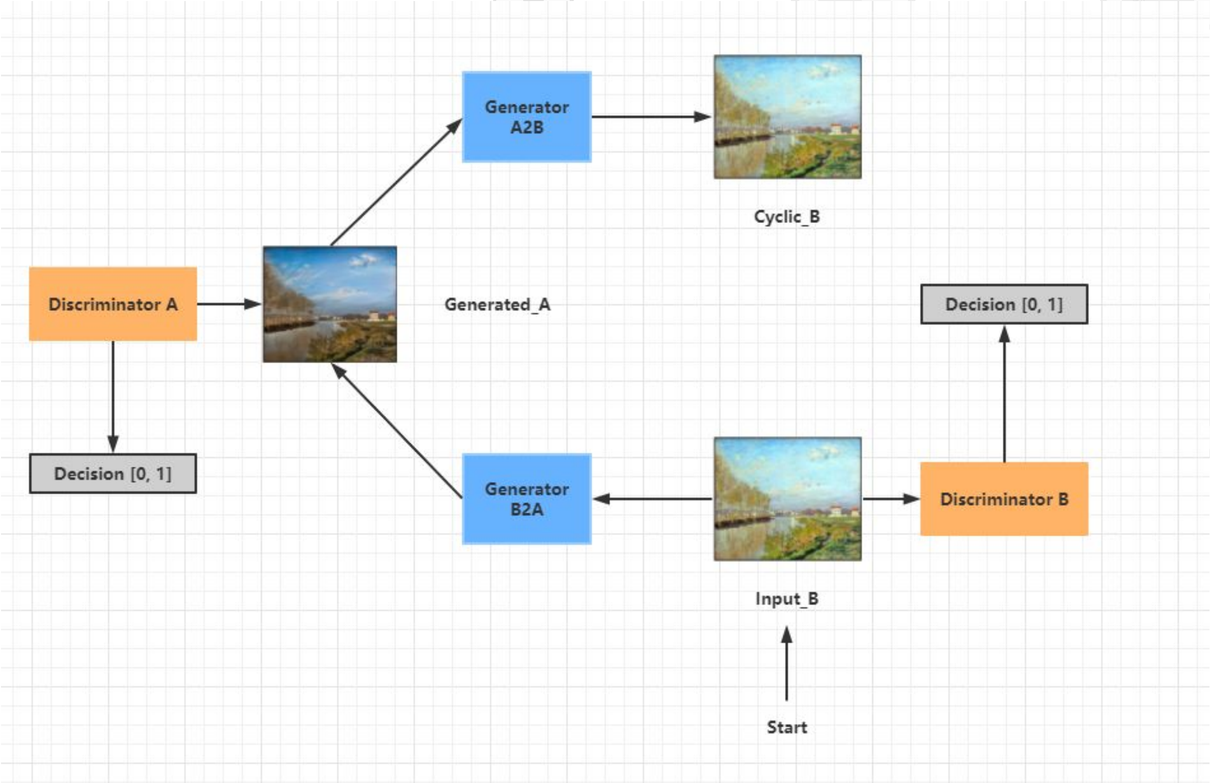


Photo to Fake Monet to Photo



Monet to Fake Photo to Monet



Experimental Setup

Data Augmentation

Generator:

- Encoder
- Transformer
- Decoder

Discriminator

Adam optimizer

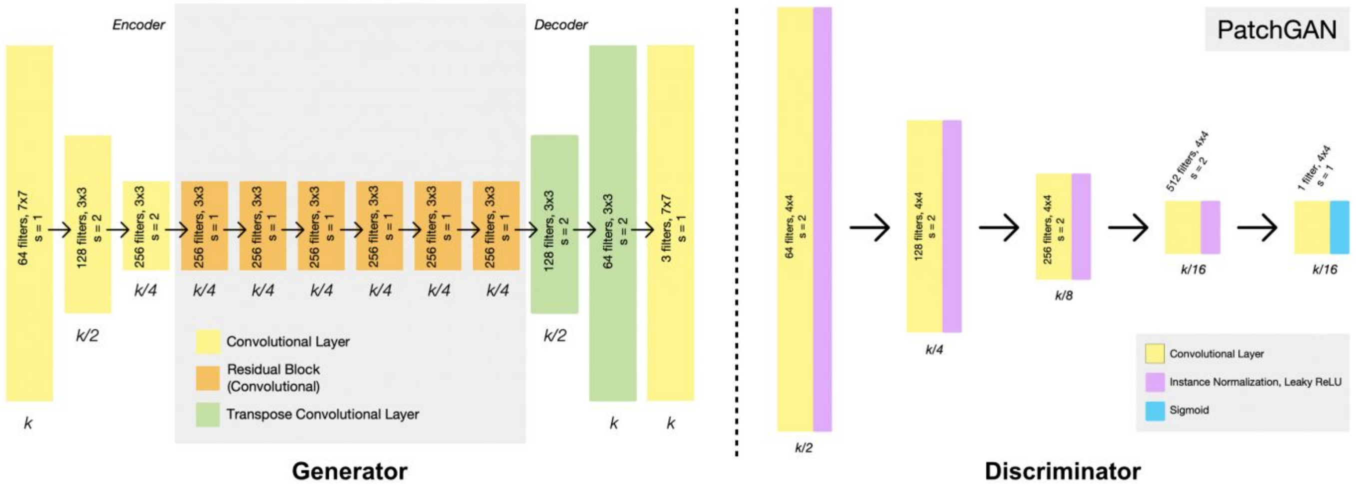
Learning rate = 0.0002

Beta_1 = 0.5

Data Augmentation

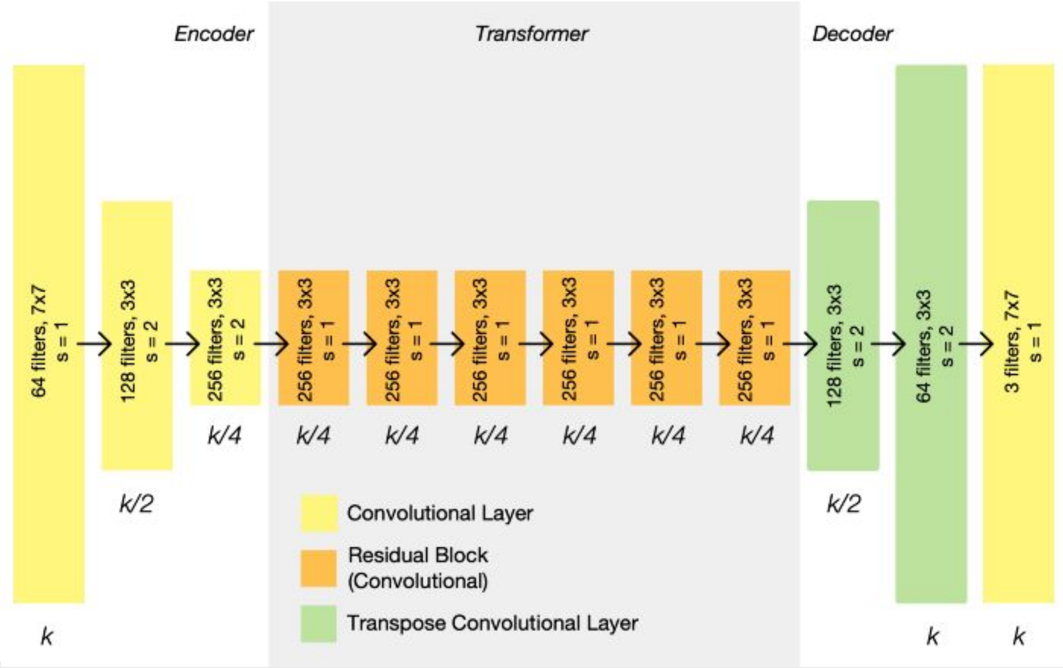
Augment Monet dataset from 300 to 600
Randomly Rotate, flip and transpose

Model Implement



GENERATOR

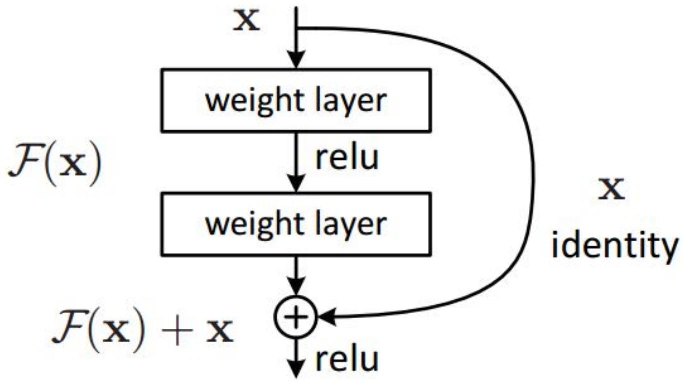
1. Instance normalization
2. LeakyReLU or ReLU
3. Transformer with Resnet blocks
4. Skip connection to solve gradient problem



Encoder

Layer	Filters	Kernel Size	Stride	Normalization	Activation
Conv2D	64	7*7	1	Instance Normalization	ReLU
Conv2D	128	3*3	2		
Conv2D	256	3*3	2		

Resnet block



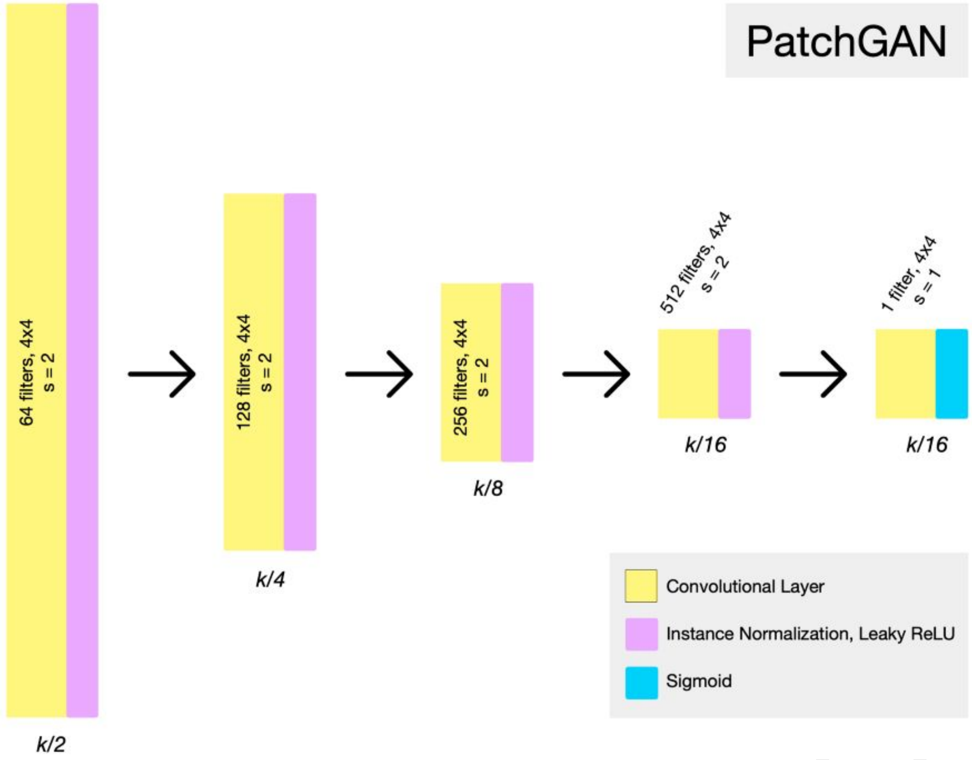
Layer	Filters	Kernel Size	Stride	Normalization	Activation
Conv2D	256	3*3	1	Instance Normalization	ReLU
Conv2D	256	3*3	1		/

Decoder

Layer	Filters	Kernel Size	Stride	Normalization	Activation
Conv2DTranspose	256	3*3	2	Instance Normalization	LeakyReLU
Conv2DTranspose	128	3*3	2		
Conv2DTranspose	65	7*7	1		

DISCRIMINATOR

1. Instance normalization
2. LeakyReLU instead of ReLU
3. Sigmoid as output activation function

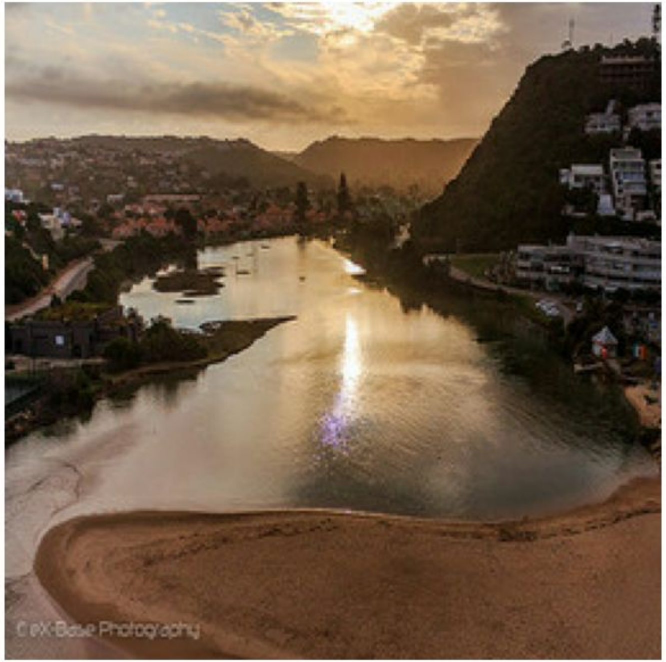


DISCRIMINATOR

Layer	Filters	Kernel Size	Stride	Normalization	Activation
Conv2D	64	4*4	2	/	LeakyReLU
Conv2D	128	4*4	2	Instance Normalization	
Conv2D	256	4*4	2		
Conv2D	512	4*4	1		

RESULTS

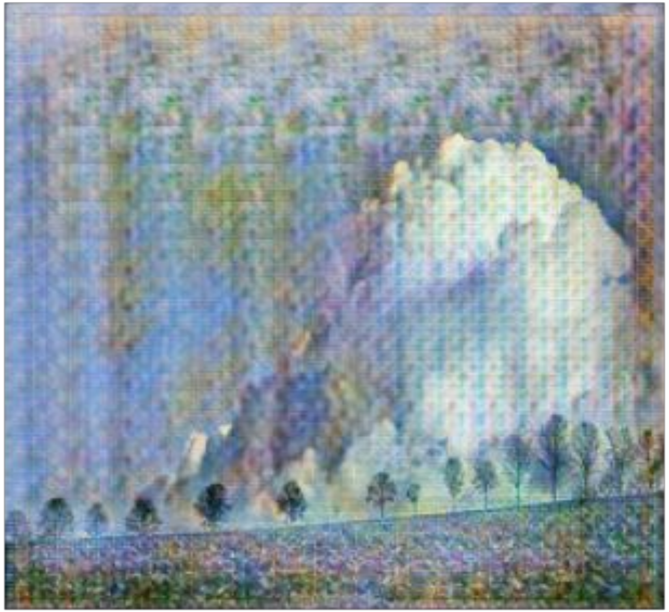
Input image



Generated image

Input Image

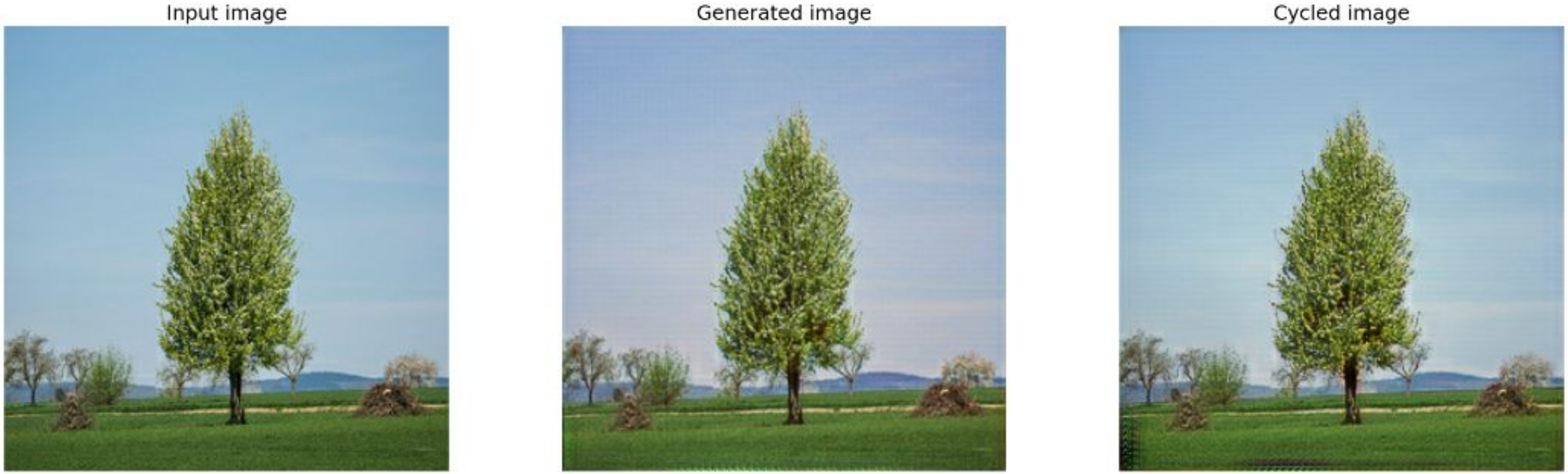
Predicted Image



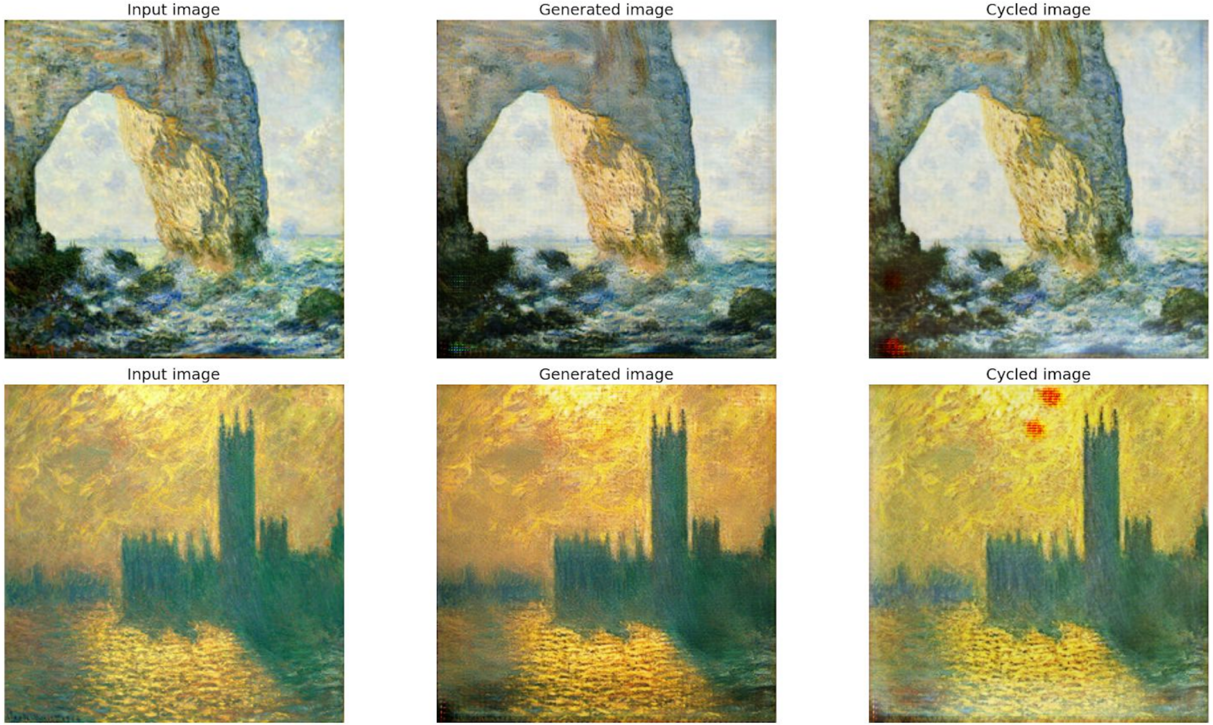

Epoch 1


Epoch 1

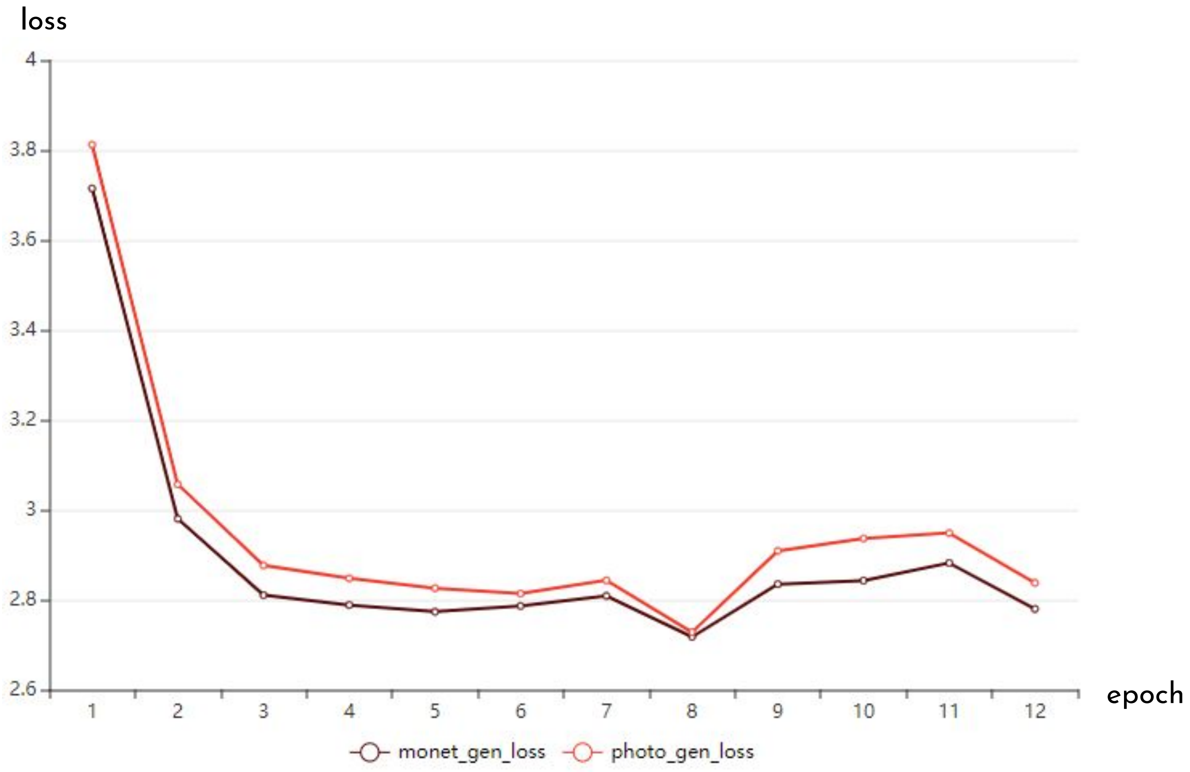
Photo->Monet->Photo



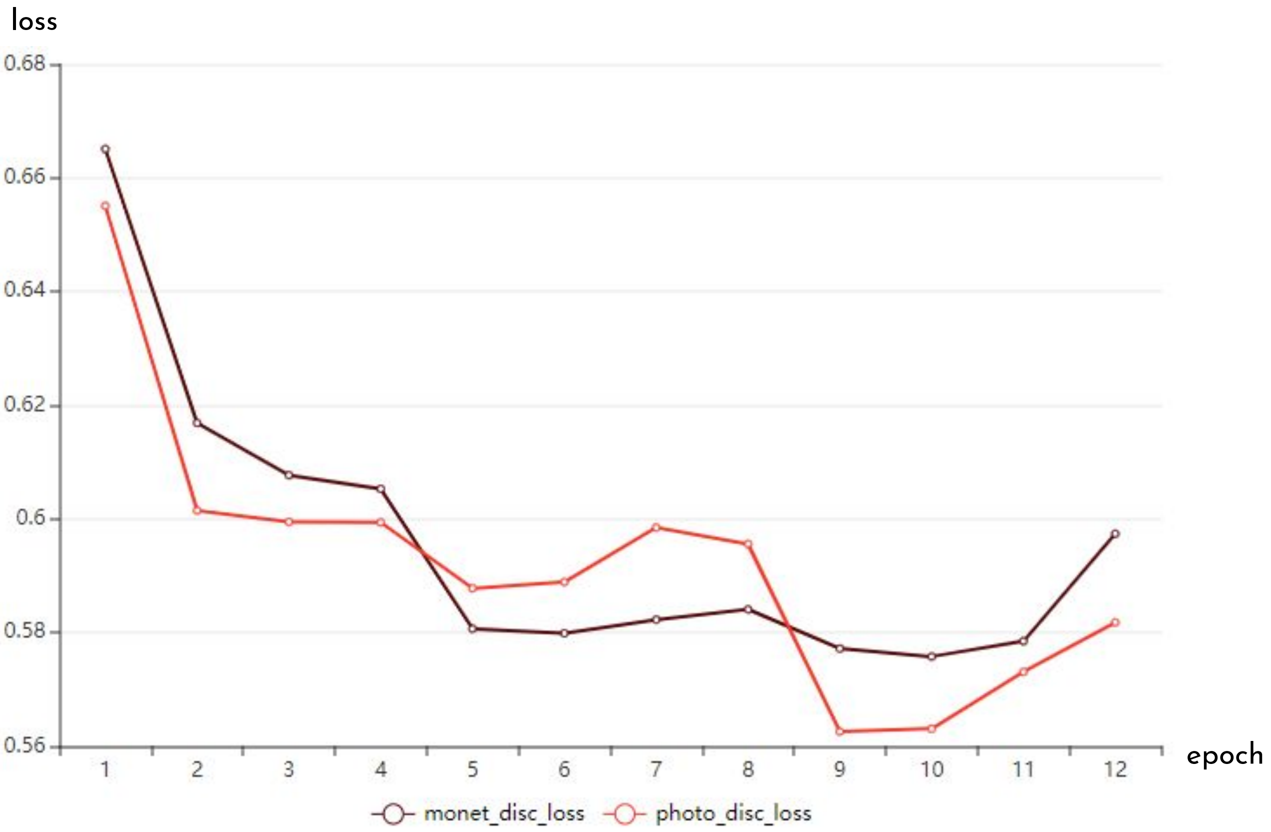
Monet->Photo->Monet



GENERATOR LOSS



DISCRIMINATOR LOSS



Conclusion

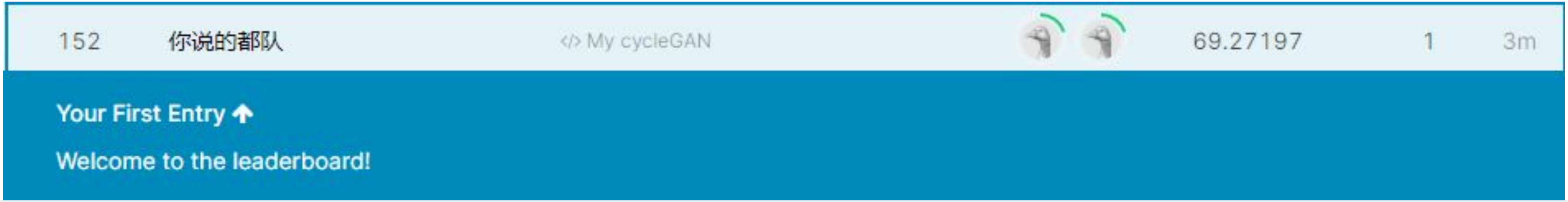
Loss:
monet generator: 2.75
photo generator: 2.77.
monet discriminator: 0.57
photo discriminator: 0.56

Run time comparison:

- ❑ GPU on AWS: 600s/epoch
- ❑ GPU on Kaggle: 233s/epoch
- ❑ TPU on Kaggle: 113s/epoch

Limitation/Need to improve:

- Use learning rate scheduler



Thank you!
Q&A