Deep Learning

Cycle GAN

U Kang
Seoul National University

In This Lecture

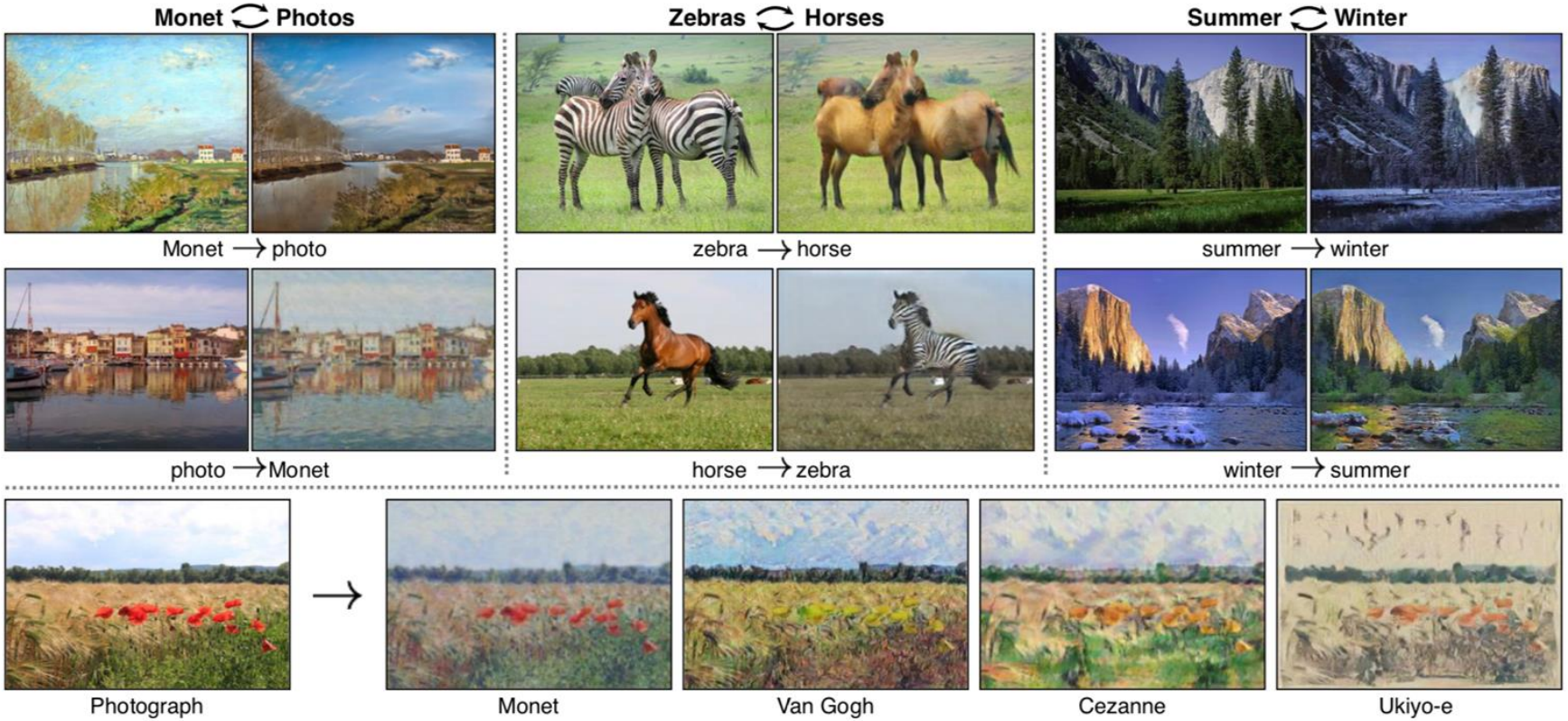
- Understand and implement cycle GAN, a GAN model for unpaired image translation

Outline

- **CycleGAN**
  - **Overview**
  - Method
  - Experiments
- Practice Problem
- Preprocessing Codes
- Answers

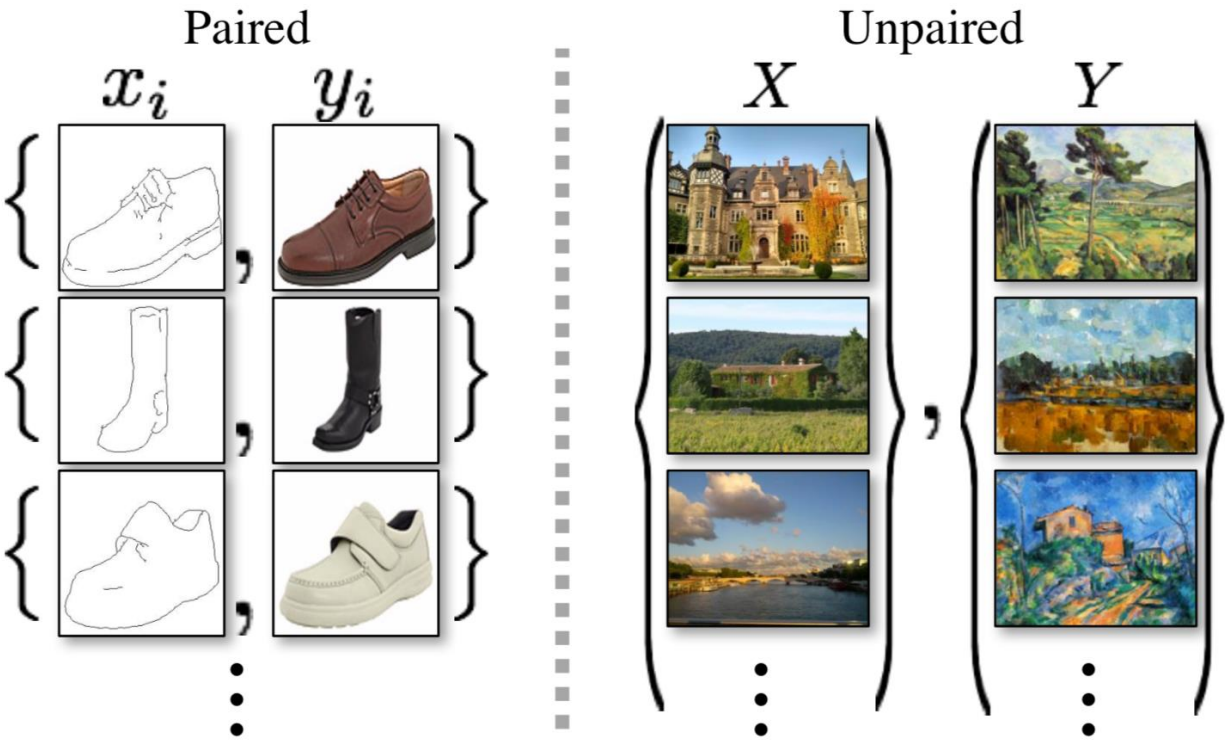
Image-to-Image Translation

- Learning the mapping between input/output images



Unpaired Image-to-Image Translation

- Learning the mapping between images in the absence of paired examples



Problem Definition

- Given: images in two different domain (X and Y)
  - $\{x_i\}_{i=1}^{N}$ where $x_i \in X$
  - $\{y_j\}_{j=1}^{N}$ where $y_j \in Y$
  - in the absence of any paired training examples
- Goal: learn mapping functions (G and F) between the two domains
  - $G : X \rightarrow Y$
  - $F : Y \rightarrow X$

Applications

- Colorization

- semantic segmentation

- Style transfer

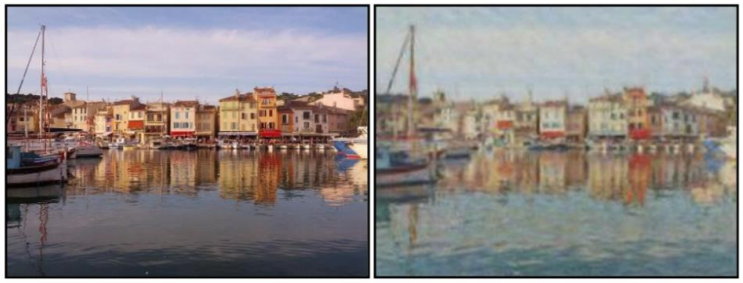
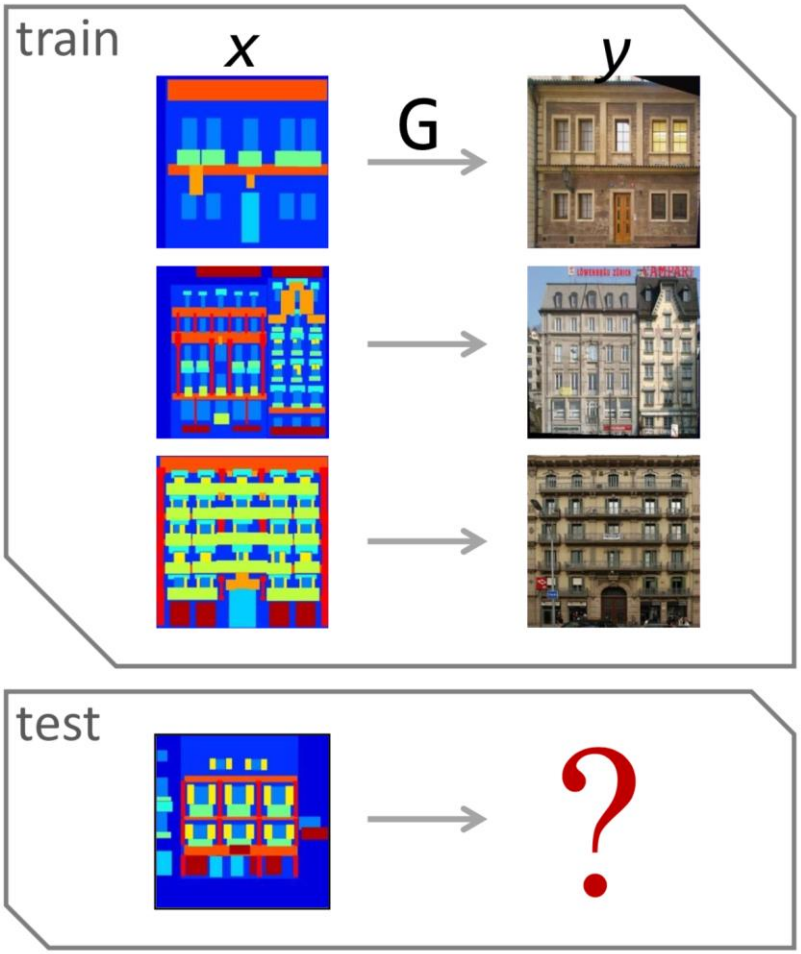


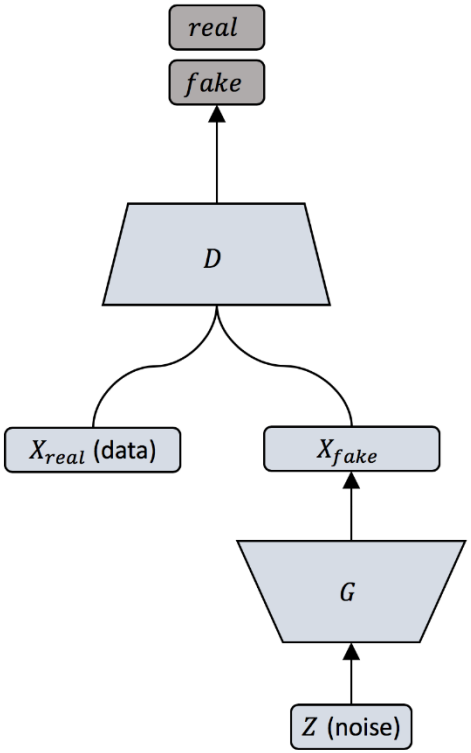
photo → Monet

Pix2pix



- Supervised
  - All training images are paired: $\{x_i, y_i\}_{i=1}^{N}$
- GAN-based image translation method

Generative Adversarial Networks



- The **discriminator** tries to distinguish real data from fake data
- The **generator** turns random noise into fake data and tries to fool the discriminator

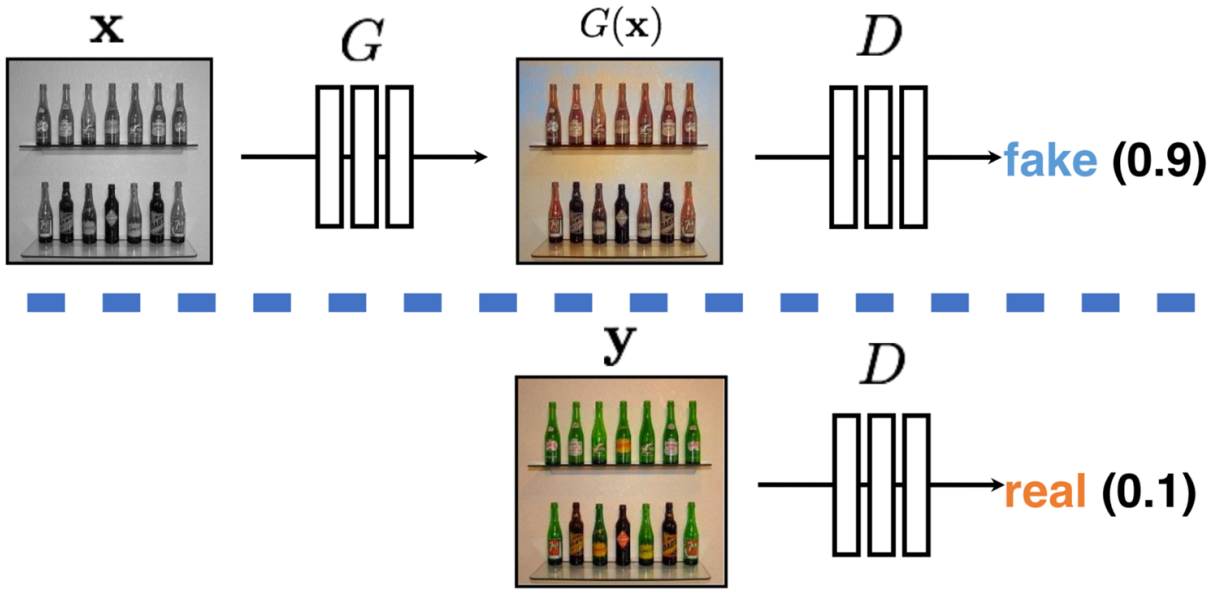
$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

Pix2pix (cont'd)

- Objective: $\sum_{(x,y)} \|y - G(x)\|_1 + L_{GAN}(G(x), y)$

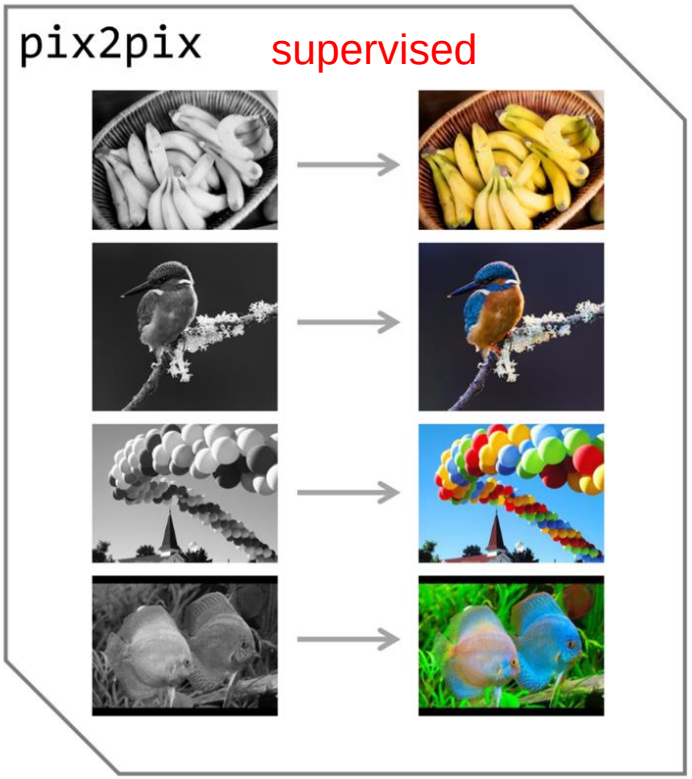
  (reconstruction loss: $\|y - G(x)\|_1$; GAN loss: $L_{GAN}(G(x), y)$)

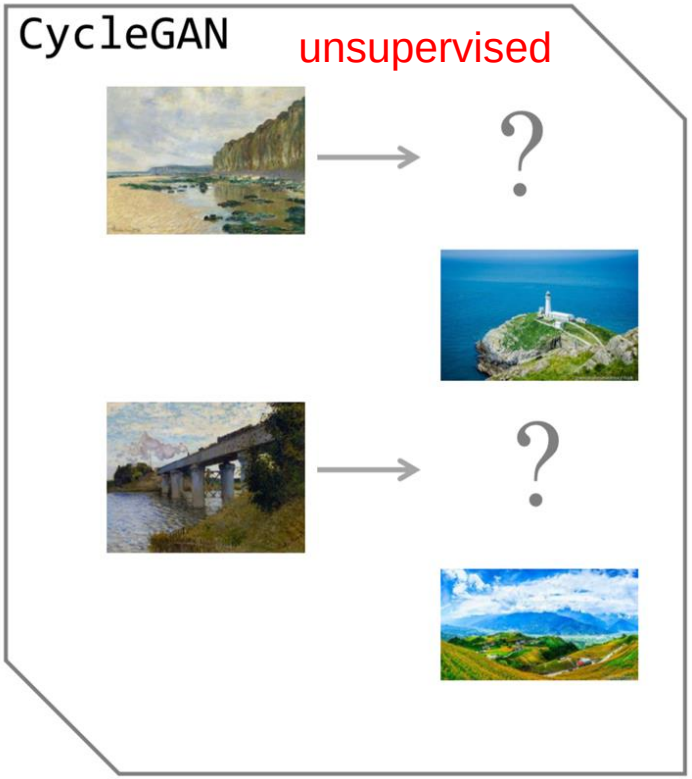
  - $L_{GAN}(G(x), y) = \arg \min_G \max_D \mathbb{E}_{\mathbf{x},\mathbf{y}}[\log D(G(\mathbf{x})) + \log(1 - D(\mathbf{y}))]$



Challenges

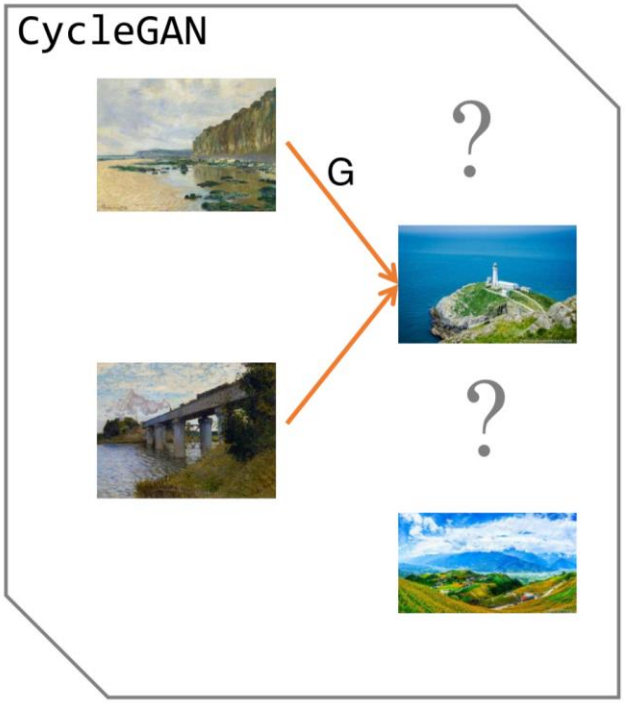
- Cannot formulate reconstruction loss: $\|y - G(x)\|_1$
  - No paired training data





Challenges

- GAN loss can cause "mode collapse": $L_{GAN}(G(x), y)$
  - mode collapse: all input images are mapped to the same output image

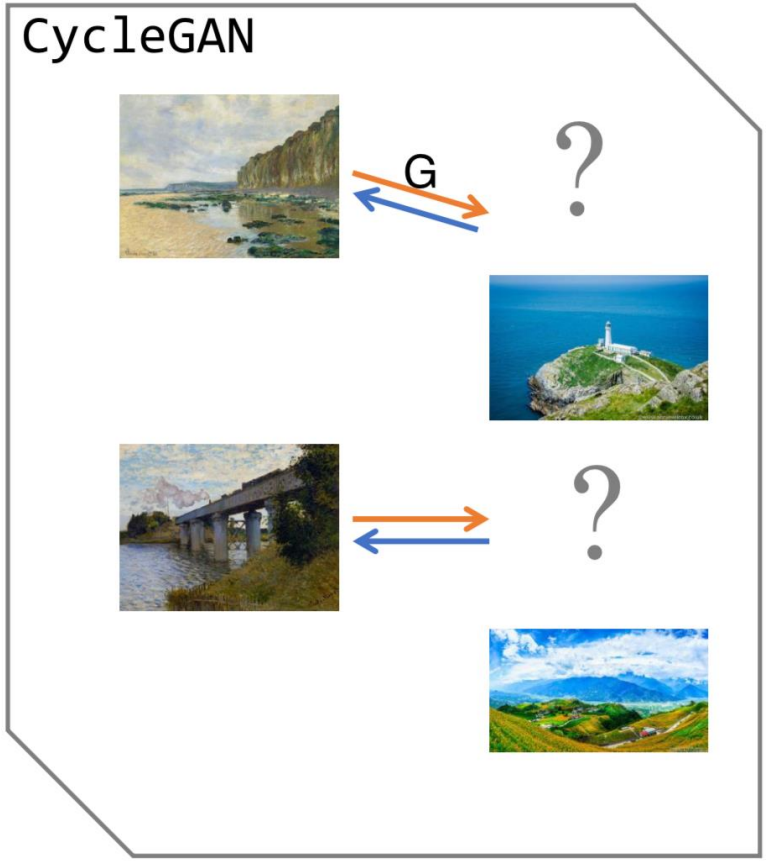


Outline

- **CycleGAN**
  - Overview
  - **Method**
  - Experiments
- Practice Problem
- Preprocessing Codes
- Answers

Intuition

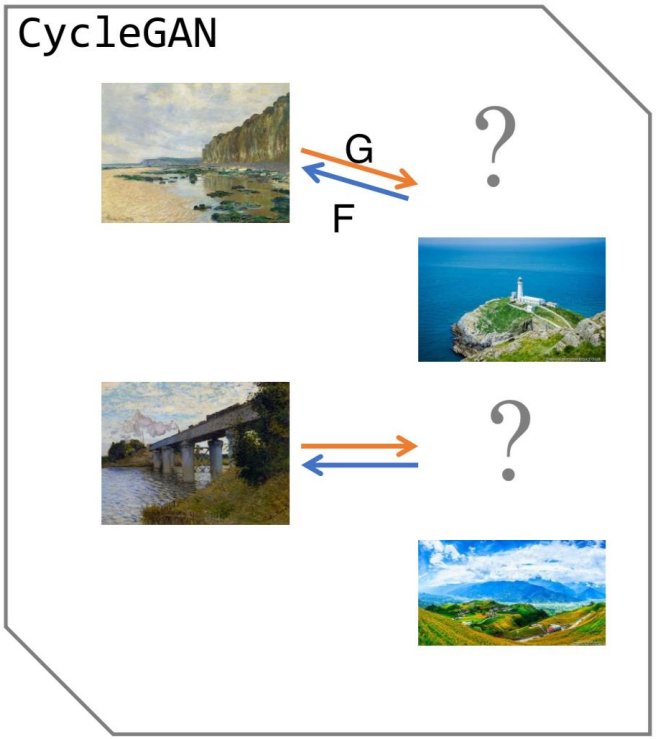
- GAN loss: $L_{GAN}(G(x), y)$
  - $G(x)$ should look like it belongs to domain Y
  - and $G(x)$ should be able to reconstruct x



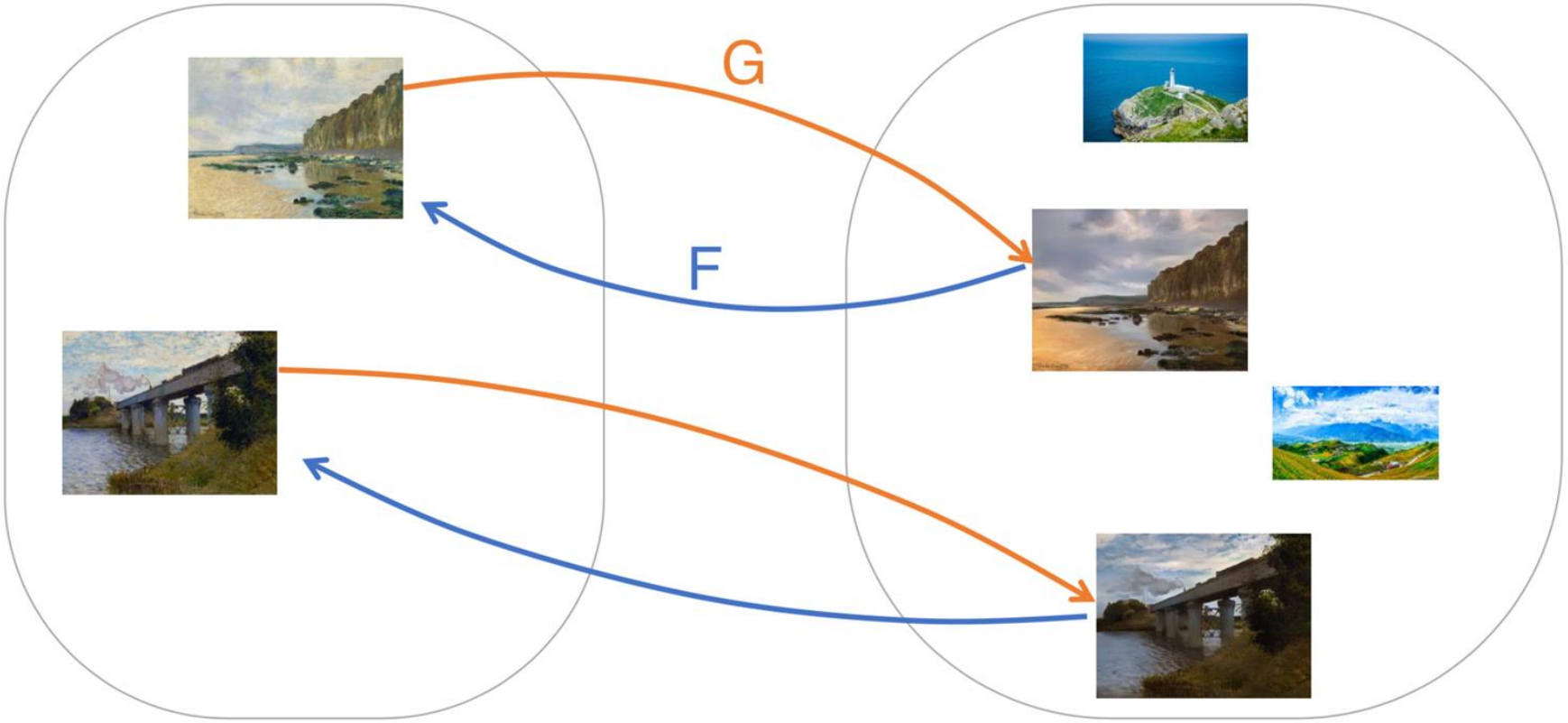
Main idea

- Cycle consistency
  - If we have $G: X \rightarrow Y$ and $F: Y \rightarrow X$, then G and F should be inverses of each other
- Loss

$$L_{GAN}(G(x), y) + \left\| F\left(G(x)\right) - x \right\|_1$$

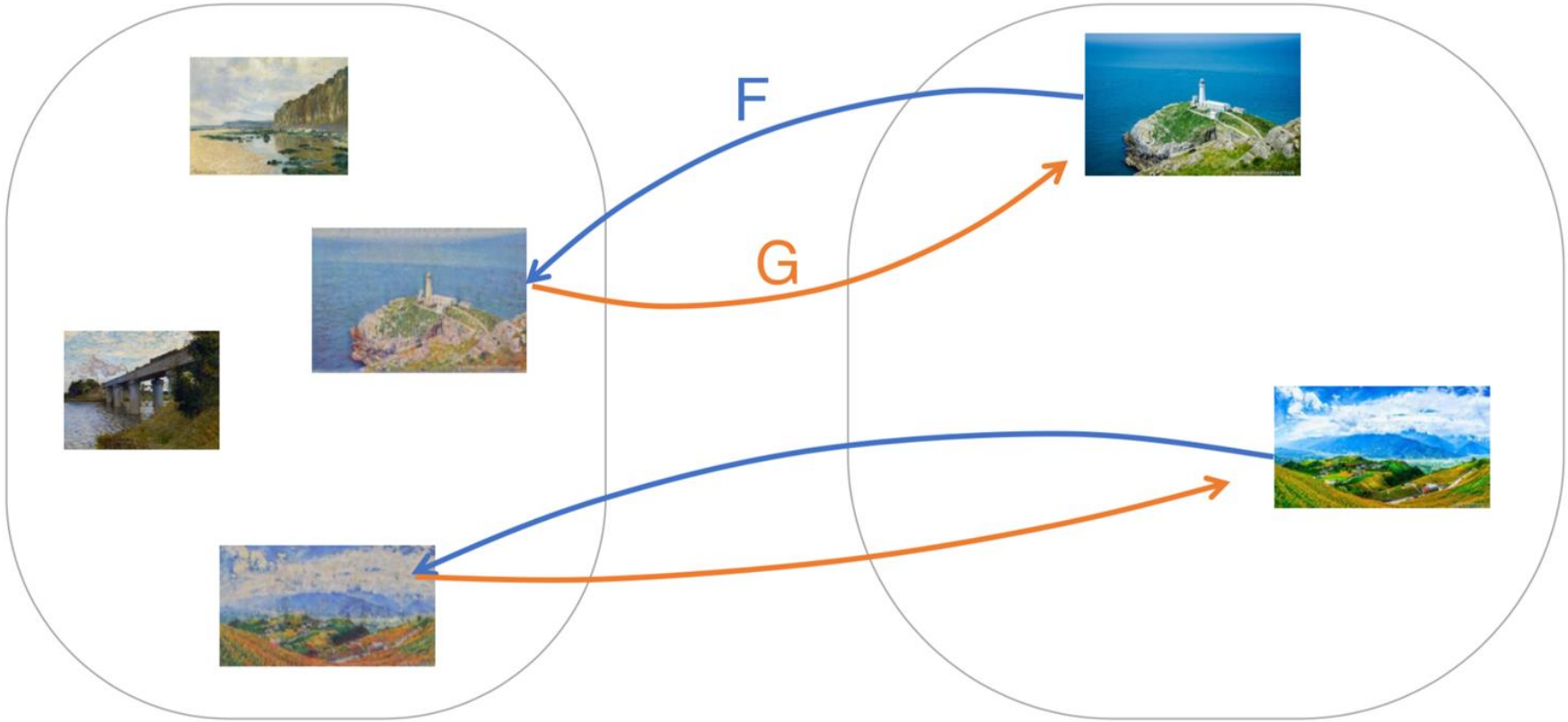


Cycle consistency



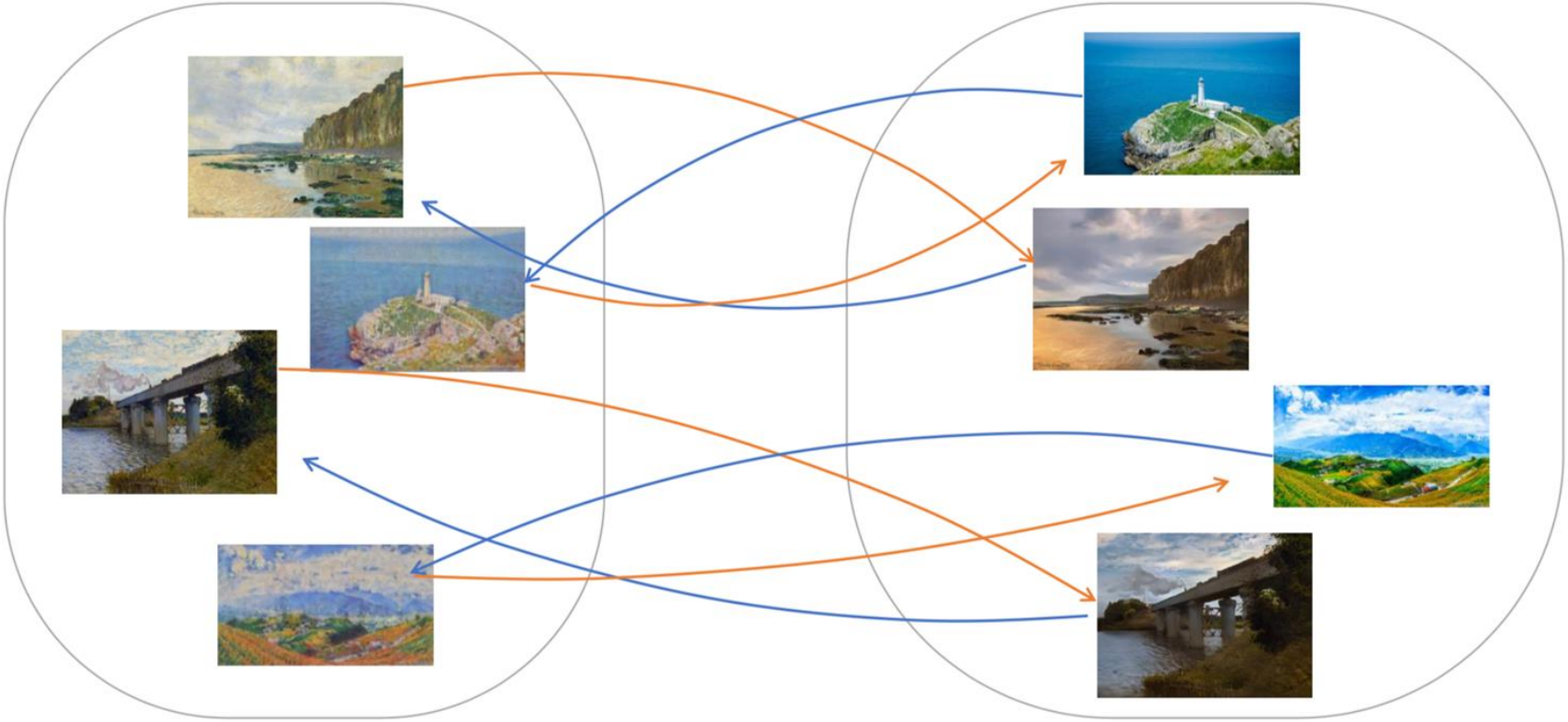
$$L_{GAN}(G(x), y) + \left\| F(G(x)) - x \right\|_1$$

Cycle consistency



$$L_{GAN}(F(y), x) + \left\| G(F(y)) - y \right\|_1$$

Cycle consistency



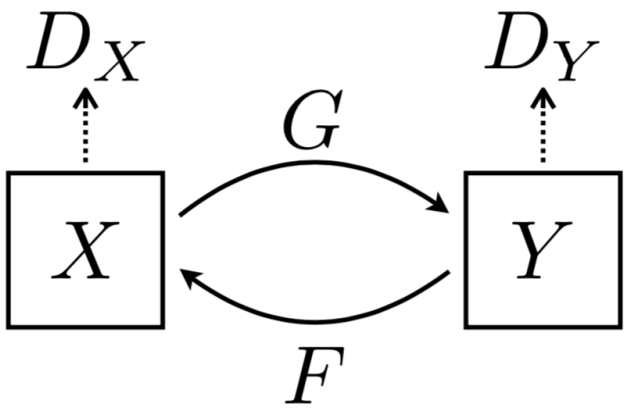
$$L_{GAN}(G(x), y) + \|F(G(x)) - x\|_1 \quad + \quad L_{GAN}(F(y), x) + \|G(F(y)) - y\|_1$$

Formulation

- Adversarial loss

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))]$$

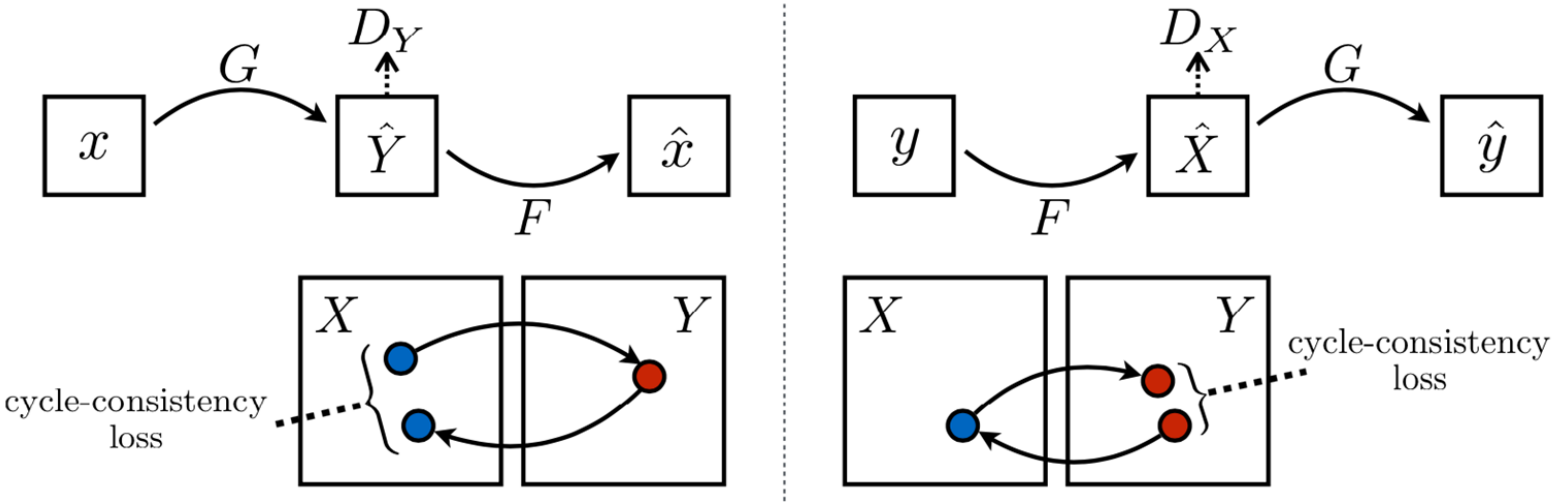
  - where $G: X \rightarrow Y$ is mapping function,
  - D_Y is discriminator.



Formulation

- Cycle consistency loss

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].$$



Formulation

- Full objective

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) =& \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) \\ &+ \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) \\ &+ \lambda \mathcal{L}_{\text{cyc}}(G, F),\end{aligned}$$

  - where λ controls the balance of the two losses

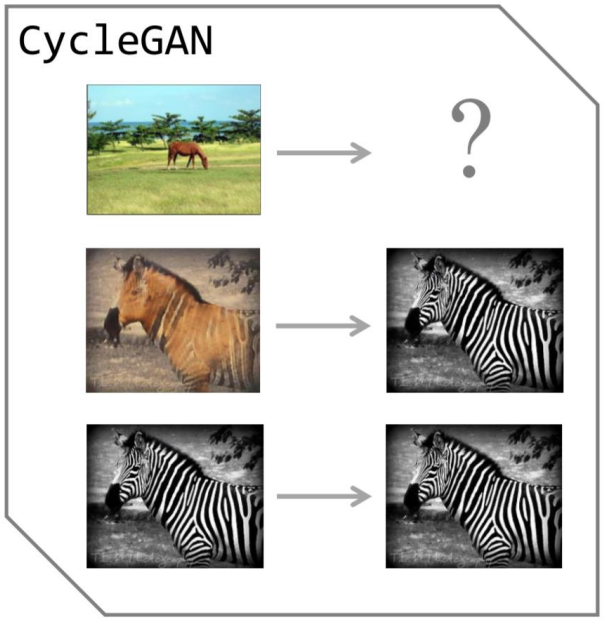
- Optimization

$$G^*, F^* = \arg\min_{G,F} \max_{D_x, D_Y} \mathcal{L}(G, F, D_X, D_Y)$$

Training details

- Combine with as much L1 loss as possible
  - L1 loss as a stable guiding force in GAN training

$$\mathcal{L}_{\text{identity}}(G, F) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(y) - y\|_1] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(x) - x\|_1]$$



Outline

- **CycleGAN**
  - Overview
  - Method
  - **Experiments**
- Practice Problem
- Preprocessing Codes
- Answers

Experiments

- Experimental questions
  - Q1. How much better compared to baselines?
    - Q1-1. Qualitatively
    - Q1-2. Quantitatively
  - Q2. How each component of the full loss function affects the results? (ablation study)
  - Q3. How $L_{identity}$ affects the results?

Experiments (cont'd)

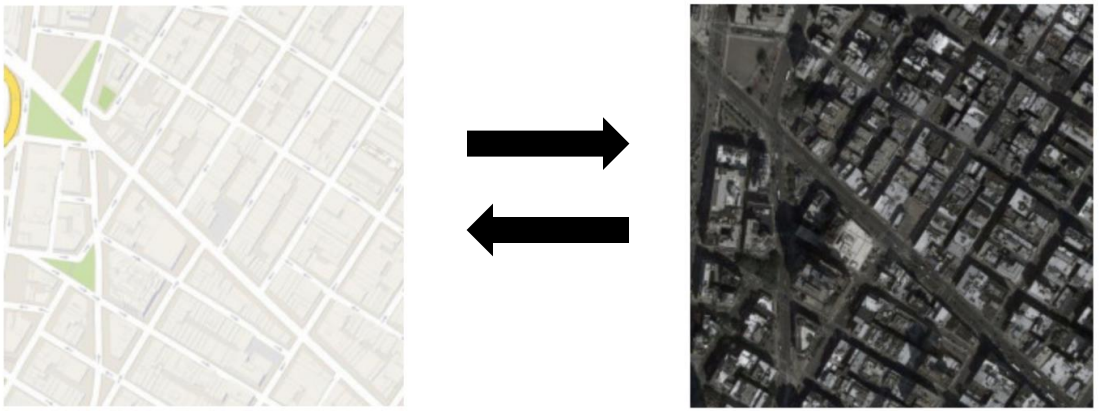
- Evaluation metrics
  - Perceptual study (qualitative)
    - Measure how much fake data deceive users
    - High is better
  - Semantic segmentation metrics (quantitative)
    - per-pixel accuracy
    - per-class accuracy
    - IOU (Intersection Over Union)
    - Higher is better

Experiments (cont'd)

- Q1-1. How much better compared to baselines qualitatively?

map

photo



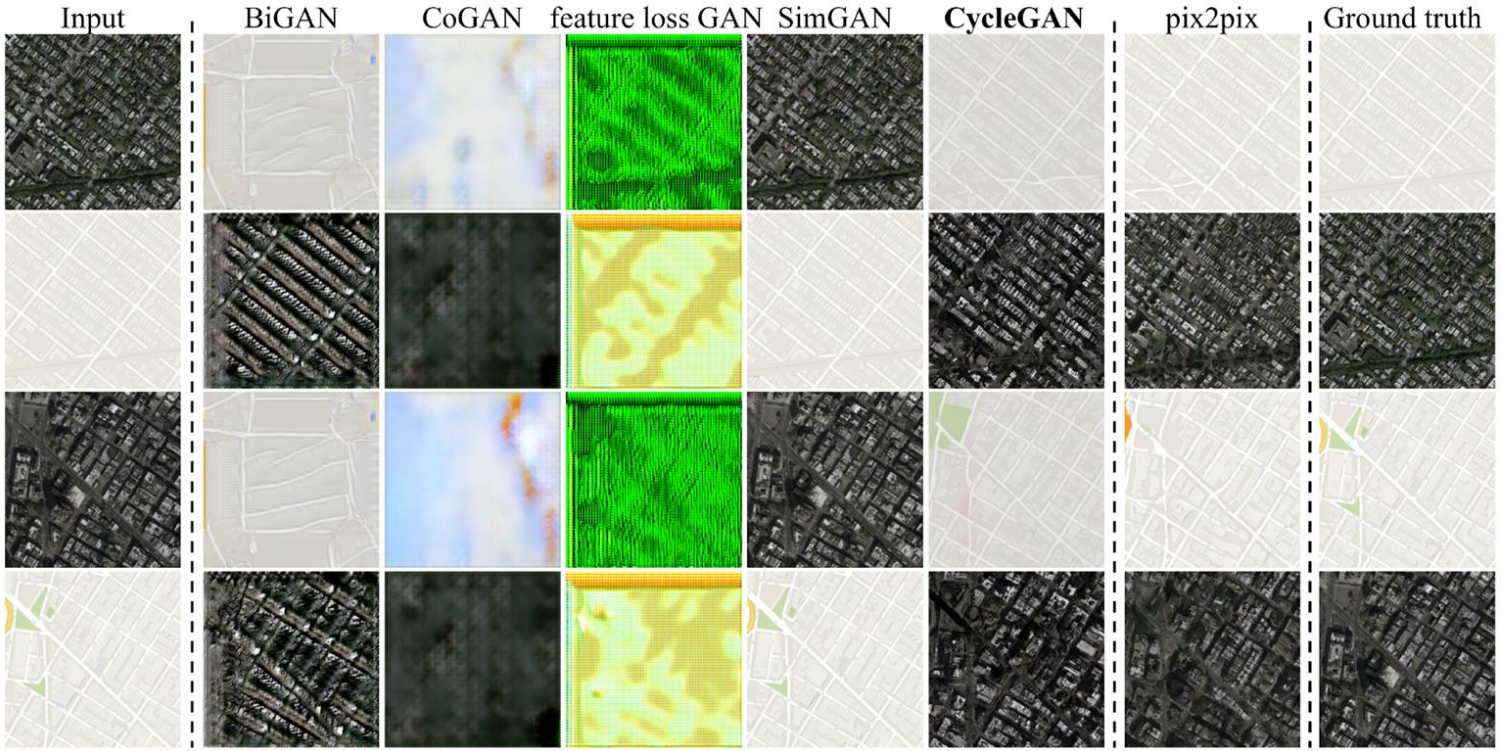
Experiments (cont'd)

- Q1-1. How much better compared to baselines qualitatively?

Loss	Map → Photo % Turkers labeled *real*	Photo → Map % Turkers labeled *real*
CoGAN [32]	0.6% ± 0.5%	0.9% ± 0.5%
BiGAN/ALI [9, 7]	2.1% ± 1.0%	1.9% ± 0.9%
SimGAN [46]	0.7% ± 0.5%	2.6% ± 1.1%
Feature loss + GAN	1.2% ± 0.6%	0.3% ± 0.2%
CycleGAN (ours)	**26.8% ± 2.8%**	**23.2% ± 3.4%**

Experiments (cont'd)

- Q1-1. How much better compared to baselines qualitatively?



Experiments (cont'd)

- Q1-2. How much better compared to baselines quantitatively?

labels

photo

Experiments (cont'd)

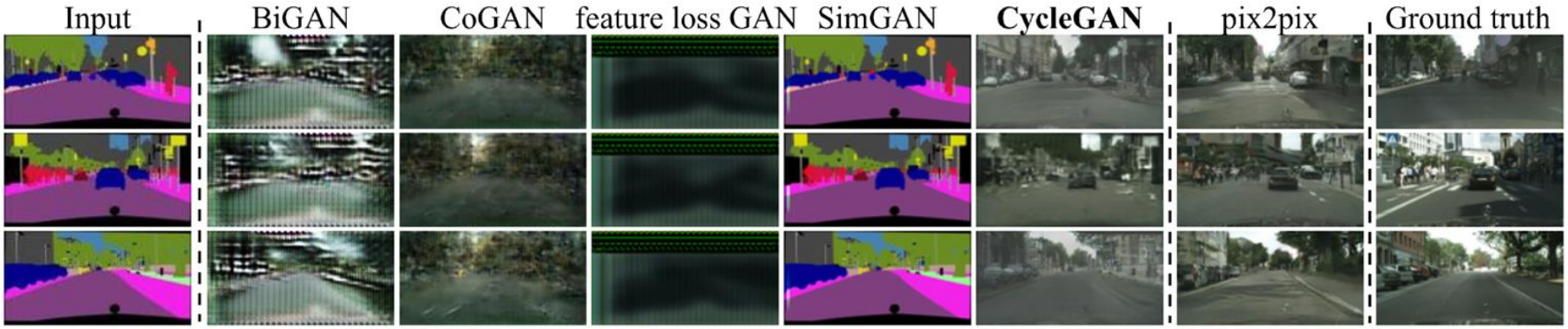
- Q1-2. How much better compared to baselines quantitatively?

Loss	Per-pixel acc.	Per-class acc.	Class IOU
CoGAN [32]	0.40	0.10	0.06
BiGAN/ALI [9, 7]	0.19	0.06	0.02
SimGAN [46]	0.20	0.10	0.04
Feature loss + GAN	0.06	0.04	0.01
CycleGAN (ours)	**0.52**	**0.17**	**0.11**
pix2pix [22]	0.71	0.25	0.18

Loss	Per-pixel acc.	Per-class acc.	Class IOU
CoGAN [32]	0.45	0.11	0.08
BiGAN/ALI [9, 7]	0.41	0.13	0.07
SimGAN [46]	0.47	0.11	0.07
Feature loss + GAN	0.50	0.10	0.06
CycleGAN (ours)	**0.58**	**0.22**	**0.16**
pix2pix [22]	0.85	0.40	0.32

Experiments (cont'd)

- Q1-2. How much better compared to baselines quantitatively?



Experiments (cont'd)

- Q2. How each component of the full loss function affects the results? (ablation study)

labels

photo

Experiments (cont'd)

- Q2. How each component of the full loss function affects the results? (ablation study)

Loss	Per-pixel acc.	Per-class acc.	Class IOU
Cycle alone	0.22	0.07	0.02
GAN alone	0.51	0.11	0.08
GAN + forward cycle	**0.55**	**0.18**	**0.12**
GAN + backward cycle	0.39	0.14	0.06
CycleGAN (ours)	0.52	0.17	0.11

Loss	Per-pixel acc.	Per-class acc.	Class IOU
Cycle alone	0.10	0.05	0.02
GAN alone	0.53	0.11	0.07
GAN + forward cycle	0.49	0.11	0.07
GAN + backward cycle	0.01	0.06	0.01
CycleGAN (ours)	**0.58**	**0.22**	**0.16**

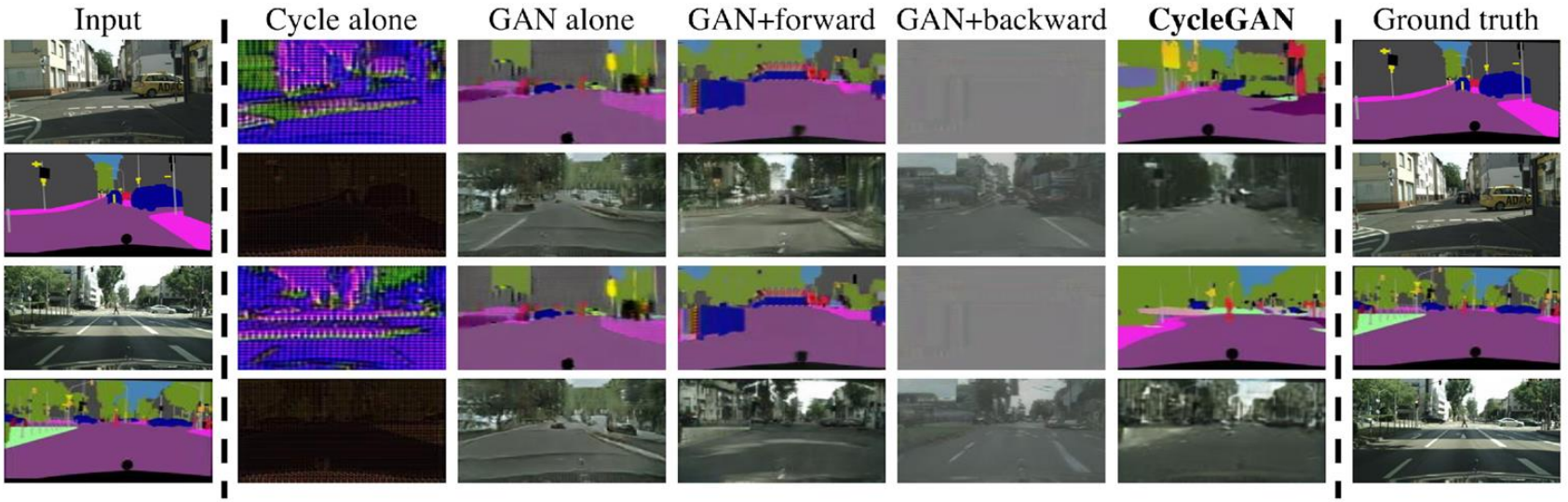
$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X)$$ GAN loss

$$+ \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].$$

forward cycle loss | backward cycle loss

Experiments (cont'd)

- Q2. How each component of the full loss function affects the results? (ablation study)



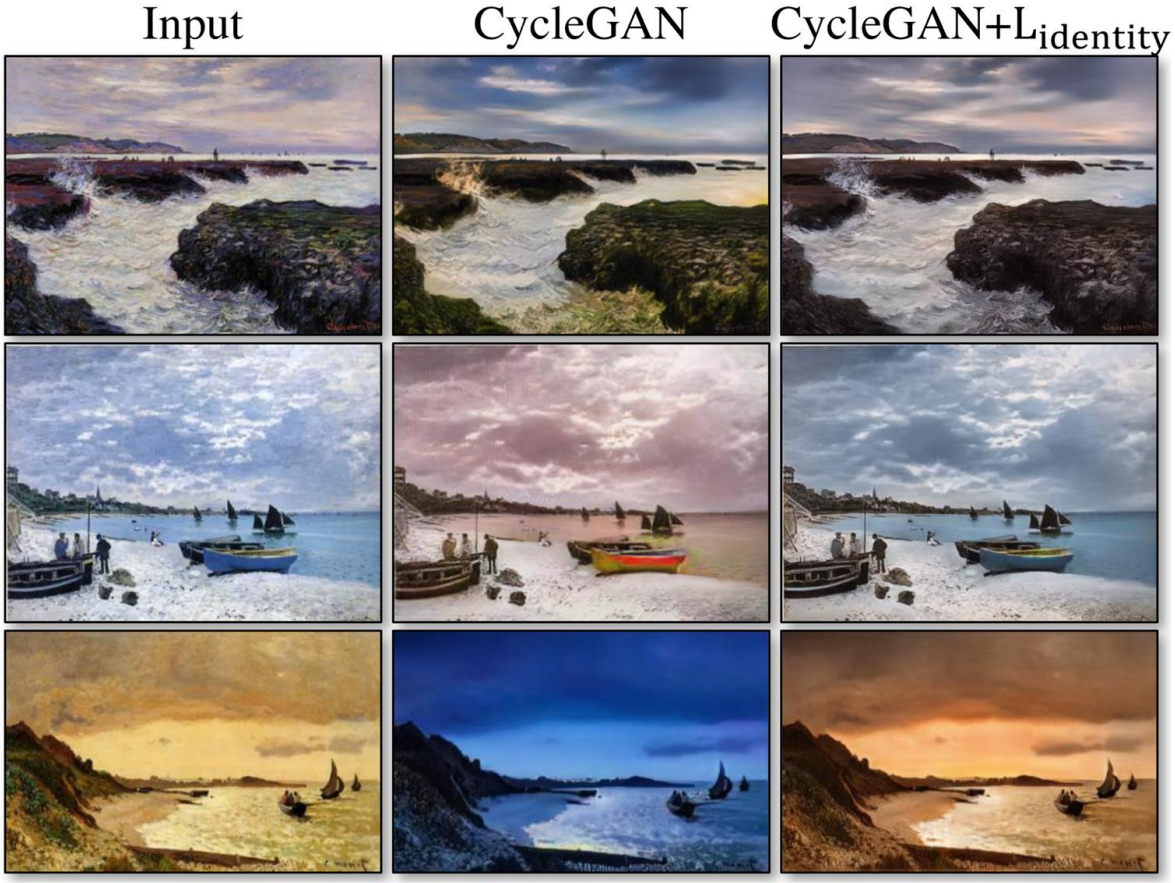
$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X)$$ GAN loss

$$+ \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].$$

forward cycle loss | backward cycle loss

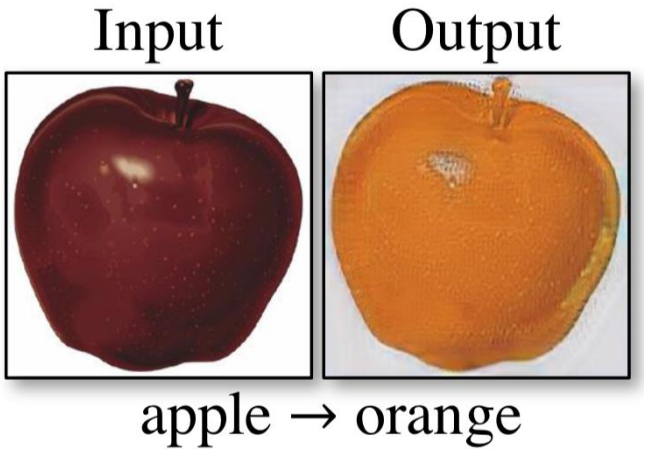
Experiments (cont'd)

- Q3. How $L_{identity}$ affects the results?

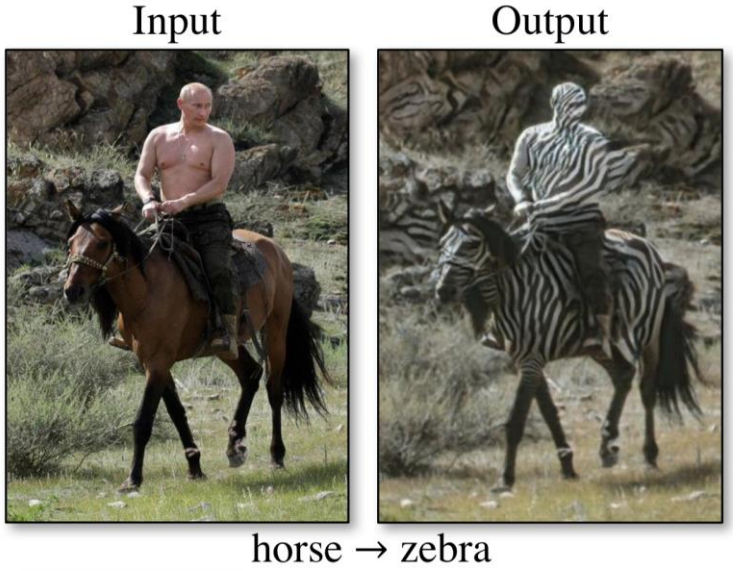


Limitations

- Hard to change compositions in an image



- Does not work well when a test image looks unusual



Conclusion

- CycleGAN
  - A new approach to solve unpaired image-to-image translation problem
  - Key idea: “cycle consistency”
  - Simple but much better results than baselines

Outline

- ☑ CycleGAN
- ☐ **Practice Problem**
- ☐ Preprocessing Codes

Dataset

- Data from berkeley
  - https://people.eecs.berkeley.edu/~taesung_park/CycleGAN/datasets/apple2orange.zip
  - There also exist another datasets such as horse2zebra, …etc
- 4 types of data
  - trainA: train image in domain A (995 instances)
  - trainB: train image in domain B (1019 instances)
  - testA: test image in domain A (266 instances)
  - testB: test image in domain B (248 instances)

Problem Definition

- We are given images of 2 domains
  - Domain A and domain B
- We want to translate an image from one domain to another domain, e.g.,
  - Given an apple image, translate it into an orange
  - Given an orange image, translate it into an apple

Outline

- ☑ CycleGAN
- ☑ Practice Problem
- ☐ **Preprocessing Codes**

Import libraries (1)

- We will use the following libraries
  - keras
  - matplotlib
  - numpy
  - scipy
  - glob
  - keras_contrib

Import libraries (2)

- If there exist some errors with scipy, e.g. imread or imresize, try the following
  - pip install scipy==0.19.1
  - pip install pillow
- To install keras_contrib, insert the following command
  - pip install git+https://www.github.com/keras-team/keras-contrib.git

Import libraries (3)

- Let's start with importing libraries

```
import datetime
import matplotlib.pyplot as plt
import numpy as np
import os
import scipy
import cv2
#pip install opencv-contrib-python
from glob import glob # to handle files

from keras_contrib.layers.normalization.instancenormalization import InstanceNormalization
# pip install git+https://www.github.com/keras-team/keras-contrib.git
from keras.layers import Input, Dropout, Concatenate
from keras.layers.advanced_activations import LeakyReLU
from keras.layers.convolutional import UpSampling2D, Conv2D
from keras.models import Model
from keras.optimizers import Adam
```

Prepare data (1)

- 'DataLoader' class has methods helpful to load data
  - 'imread'
  - 'load_data'
  - 'load_batch'
- Configure settings in constructor

```
1  class DataLoader():
2      def __init__(self, dataset_name, img_res=(128, 128)):
3          self.dataset_name = dataset_name
4          self.img_res = img_res
5
```

Prepare data (2)

- Method ‘imread’ is wrapper of cv2.imread
- This function reads image and returns numpy array

```
def imread(self, path):
    return cv2.imread(path).astype(np.float)
```

Prepare data (3)

- Method ‘load_data’ load images from given domain
- At train time, it performs data augmentation stochastically, e.g. flipping left to right

```
def load_data(self, domain, batch_size=1, is_testing=False, seed=None):
    data_type = "train%s" % domain if not is_testing else "test%s" % domain
    path = glob('./data/%s/%s/*' % (self.dataset_name, data_type))

    # set seed to show train data
    if is_testing == False and seed is not None:
        np.random.seed(seed)

    batch_images = np.random.choice(path, size=batch_size)

    imgs = []
    for img_path in batch_images:
        img = self.imread(img_path)
        img = cv2.resize(img, self.img_res)

        if not is_testing:
            # left right flipping
            if np.random.random() > 0.5:
                img = np.fliplr(img)

        imgs.append(img)

    imgs = np.array(imgs)/127.5 - 1.

    return imgs
```

Prepare data (4)

- Method ‘load_batch’ also loads images from domains in training data
- This method is almost the same as ‘load_data’ except
  - Reads data from 2 domains
  - Uses python generator which can save memories when data are too large

Prepare data (5)

- Below is 'load_batch' method

```
def load_batch(self, batch_size=1):
    data_type = "train"

    path_A = glob('./data/%s/%sA/*' % (self.dataset_name, data_type))
    path_B = glob('./data/%s/%sB/*' % (self.dataset_name, data_type))

    self.n_batches = int(min(len(path_A), len(path_B)) / batch_size)
    total_samples = self.n_batches * batch_size

    # Sample n_batches * batch_size from each path list so that model sees all
    # samples from both domains
    path_A = np.random.choice(path_A, total_samples, replace=False)
    path_B = np.random.choice(path_B, total_samples, replace=False)

    for i in range(self.n_batches):
        batch_A = path_A[i*batch_size:(i+1)*batch_size]
        batch_B = path_B[i*batch_size:(i+1)*batch_size]
        imgs_A, imgs_B = [], []
        for img_A, img_B in zip(batch_A, batch_B):
            img_A = self.imread(img_A)
            img_B = self.imread(img_B)

            img_A = cv2.resize(img_A, self.img_res)
            img_B = cv2.resize(img_B, self.img_res)

            if np.random.random() > 0.5:
                    img_A = np.fliplr(img_A)
                    img_B = np.fliplr(img_B)

            imgs_A.append(img_A)
            imgs_B.append(img_B)

        imgs_A = np.array(imgs_A)/127.5 - 1.
        imgs_B = np.array(imgs_B)/127.5 - 1.

        yield imgs_A, imgs_B
```

Questions?