Final Project Demo

CSIE Tsung-Shu Chiu

大綱

- 介紹
- 程式碼
- 成果
- 未來展望
- 參考文獻

介紹 — 問題

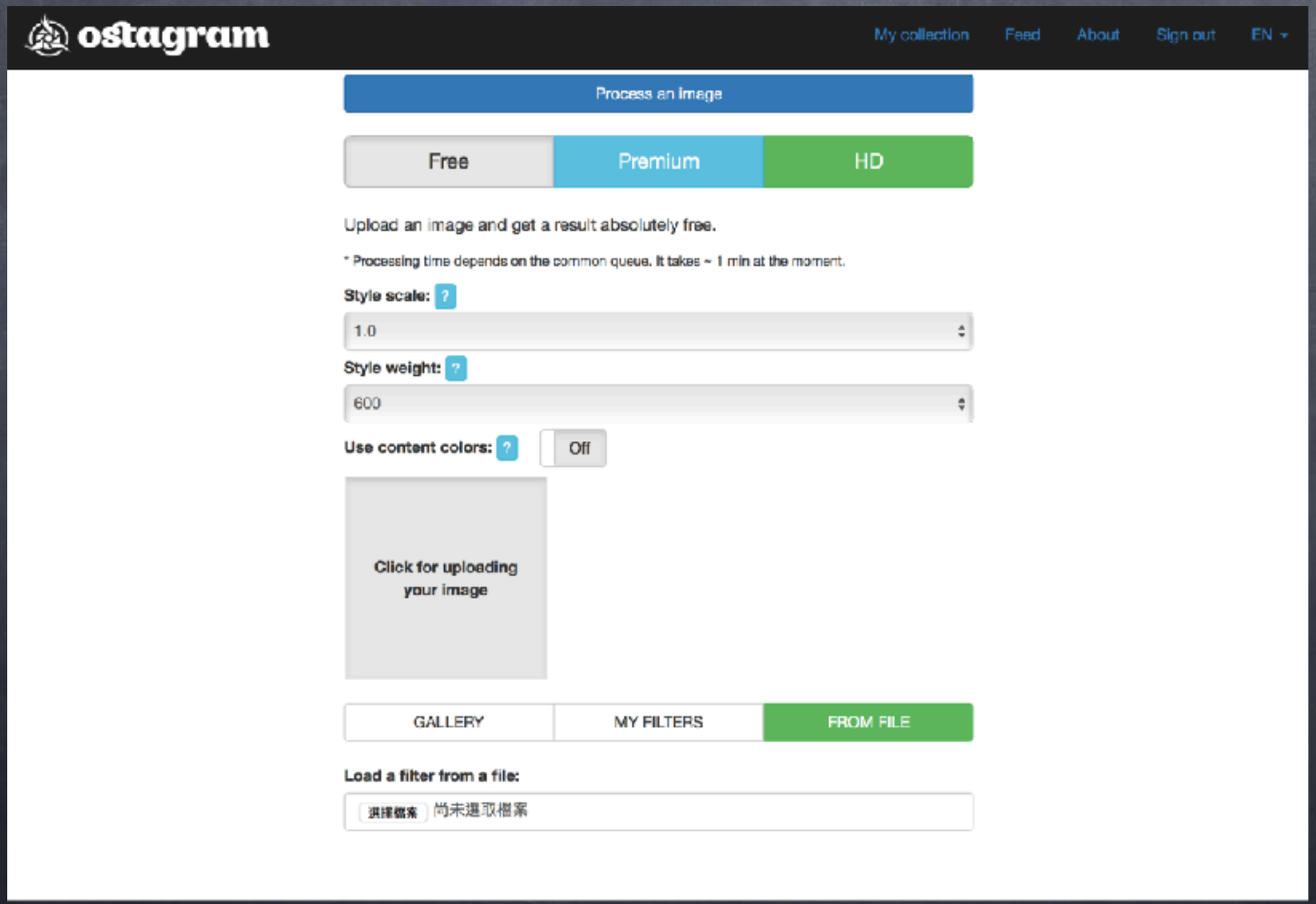
圖片風格轉換

把圖片A當作骨幹，提取圖片B的風格，融合出圖片C

A C B

介紹 — 法一

ostagram



介紹 — 法二

Generative Adversarial Network(GAN)

- 生成對抗網路
- 2014年提出
- 可以同時訓練兩個神經網路
- 由「生成網路」和「判別網路」所組成
- 非監督式學習的一種

Generative Adversarial Nets

Abstract

1 Introduction

https://arxiv.org/abs/1406.2661

Machine Learning

supervised

Train Set

1. Feature: (X1, X2, ... Xn) Label: y
2. Feature: (X1, X2, ... Xn) Label: y
3. Feature: (X1, X2, ... Xn) Label: y

\+

Algorithm

svm, decision tree, ...

=

Model

↓

Labels

semi-supervised

Unsupervised

Train Set

1. Feature: (X1, X2, ... Xn)
2. Feature: (X1, X2, ... Xn)
3. Feature: (X1, X2, ... Xn)

\+

Algorithm

cluster, Neural network, ...

=

Model

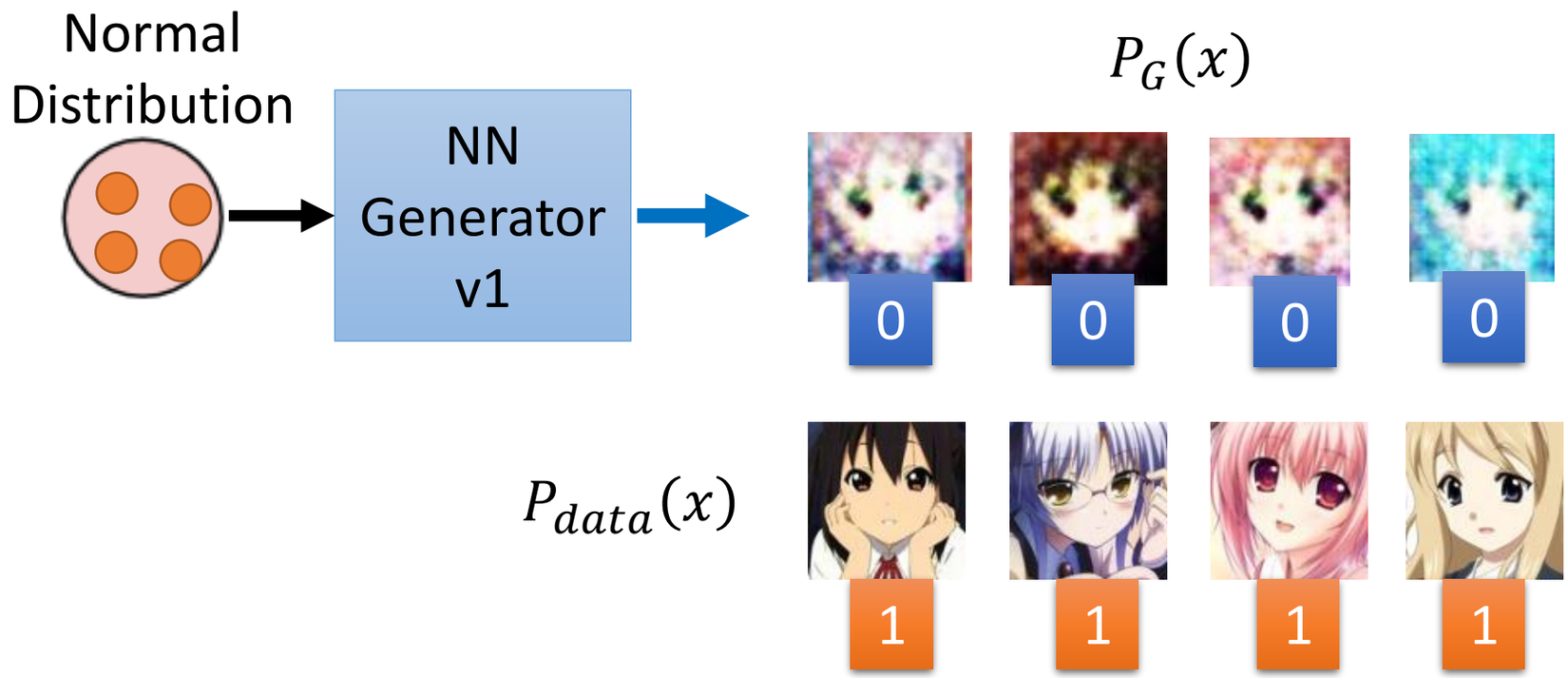
↓

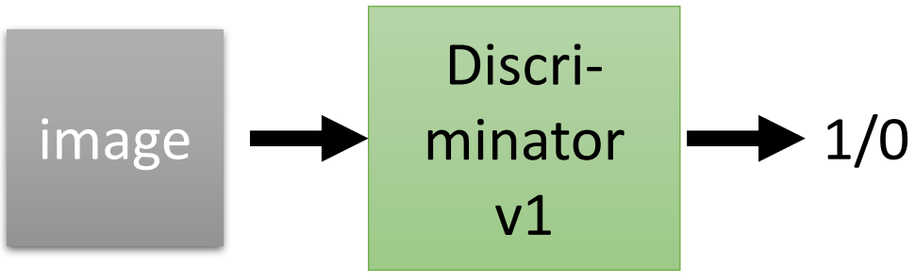
Labels

test set

1. Feature: (X1, X2, ... Xn)
2. Feature: (X1, X2, ... Xn)
3. Feature: (X1, X2, ... Xn)

Basic Idea of GAN



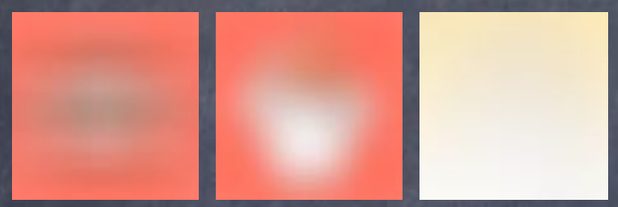


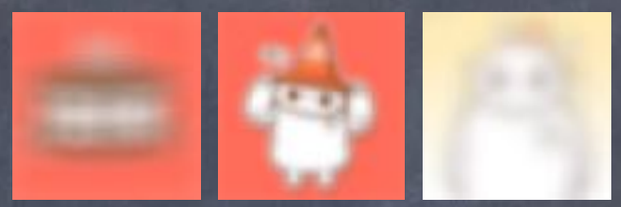
It can be proofed that the ***loss the discriminator*** related to ***JS divergence***.

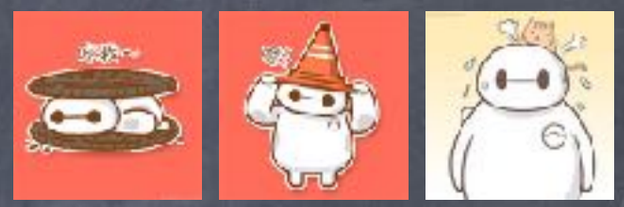
擷取自李弘毅教授ML2017投影片 GAN.pdf

GAN Evolution

Generator V1 → Generator V2 → Generator V3







Discriminator V1 → Discriminator V2 → Discriminator V3

Real Images

但是，GAN在Image-to-image上有個缺點，以下引用自GAN 的論文:

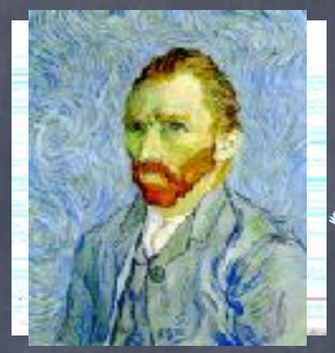
"Adversarial training can, in theory, learn mappings G and F that produce outputs identically distributed as target domains Y and X respectively. However, with large enough capacity, a network can map the same set of input images to any random permutation of images in the target domain, where any of the learned mappings can induce an output distribution that matches the target distribution. Thus, an adversarial loss alone cannot guarantee that the learned function can map an individual input xi to a desired output yi."

總而言之就是，Output不一定是你想要的

Domain X

G:X2Y

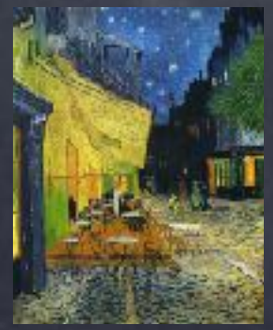
Similar to Y

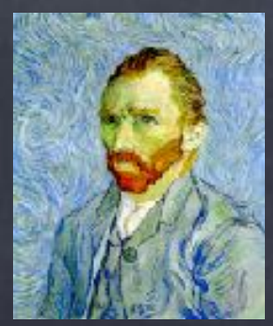


Input is noise

D:Y

isY?





Domain Y

Cycle-consistent GAN (CycleGAN)

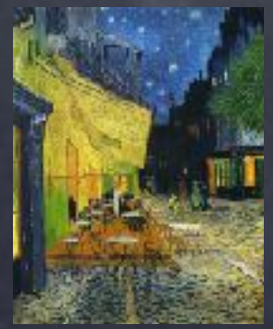
as close as possible

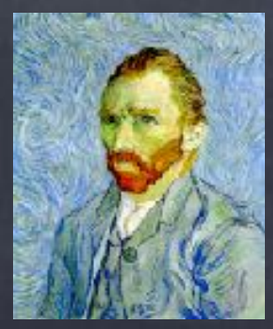
G:X2Y

G:Y2X

D:Y

isY?





Domain Y

as close as possible

G:X2Y

G:Y2X

D:X

D:Y

G:Y2X

G:X2Y

as close as possible

Generator

Input

↓

Encoder

↓

Transform

↓

Decoder

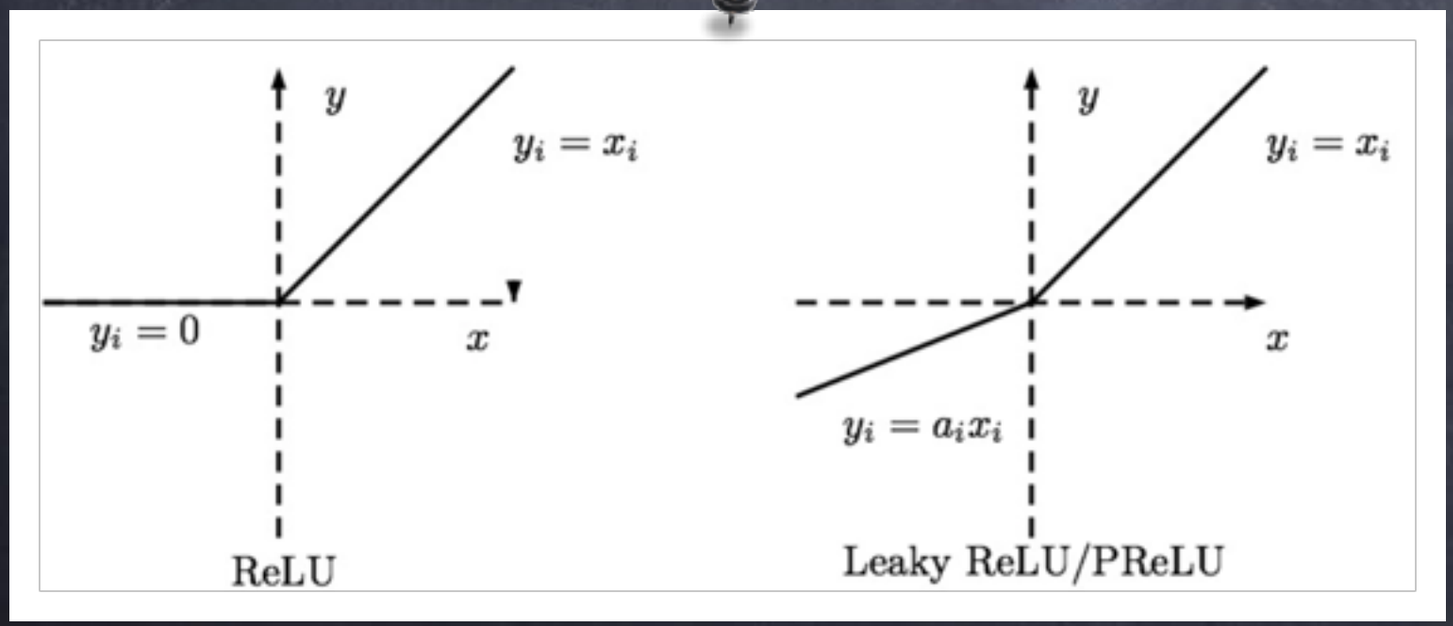
↓

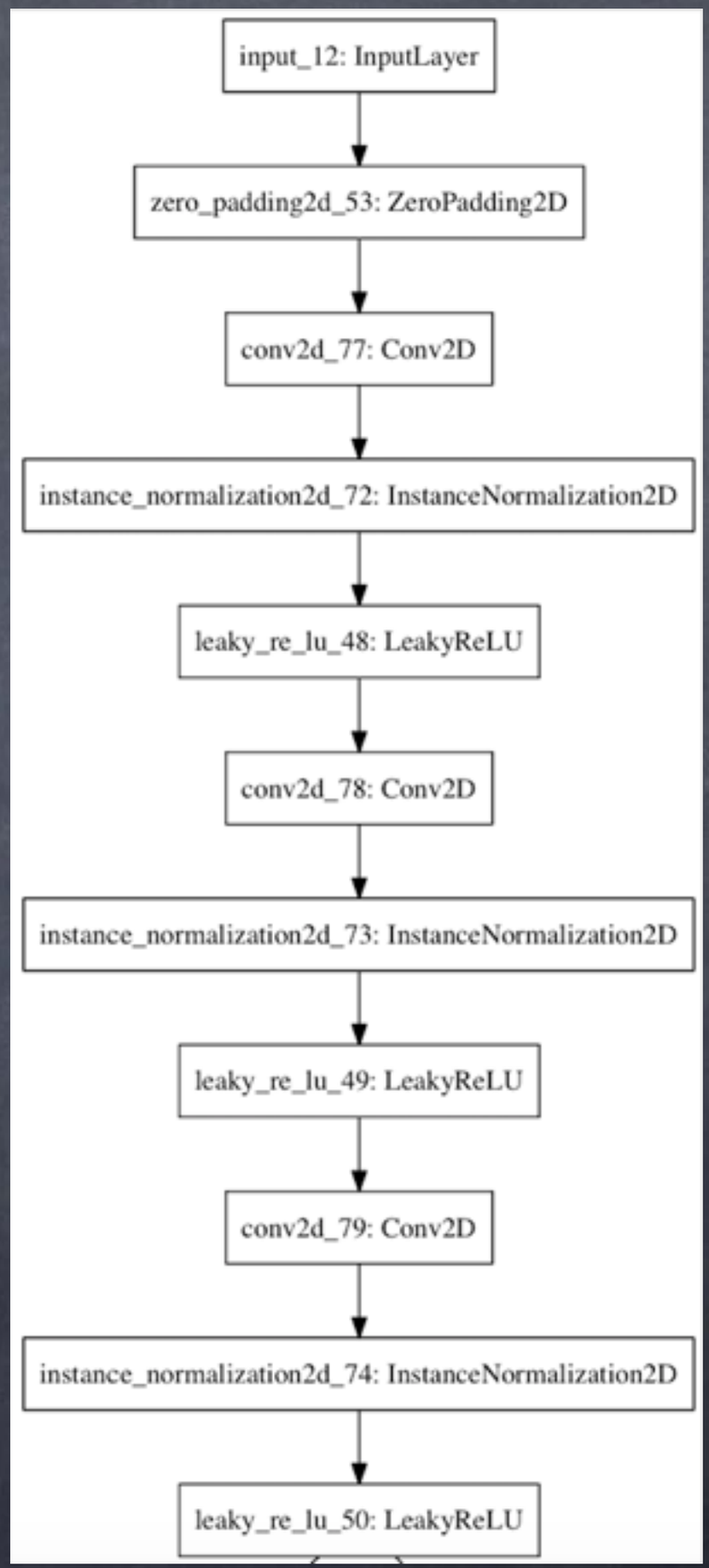
Output

Generator

Encoder

- Feature Extraction with CNN
- Normalization
- Activation with LeakyReLU

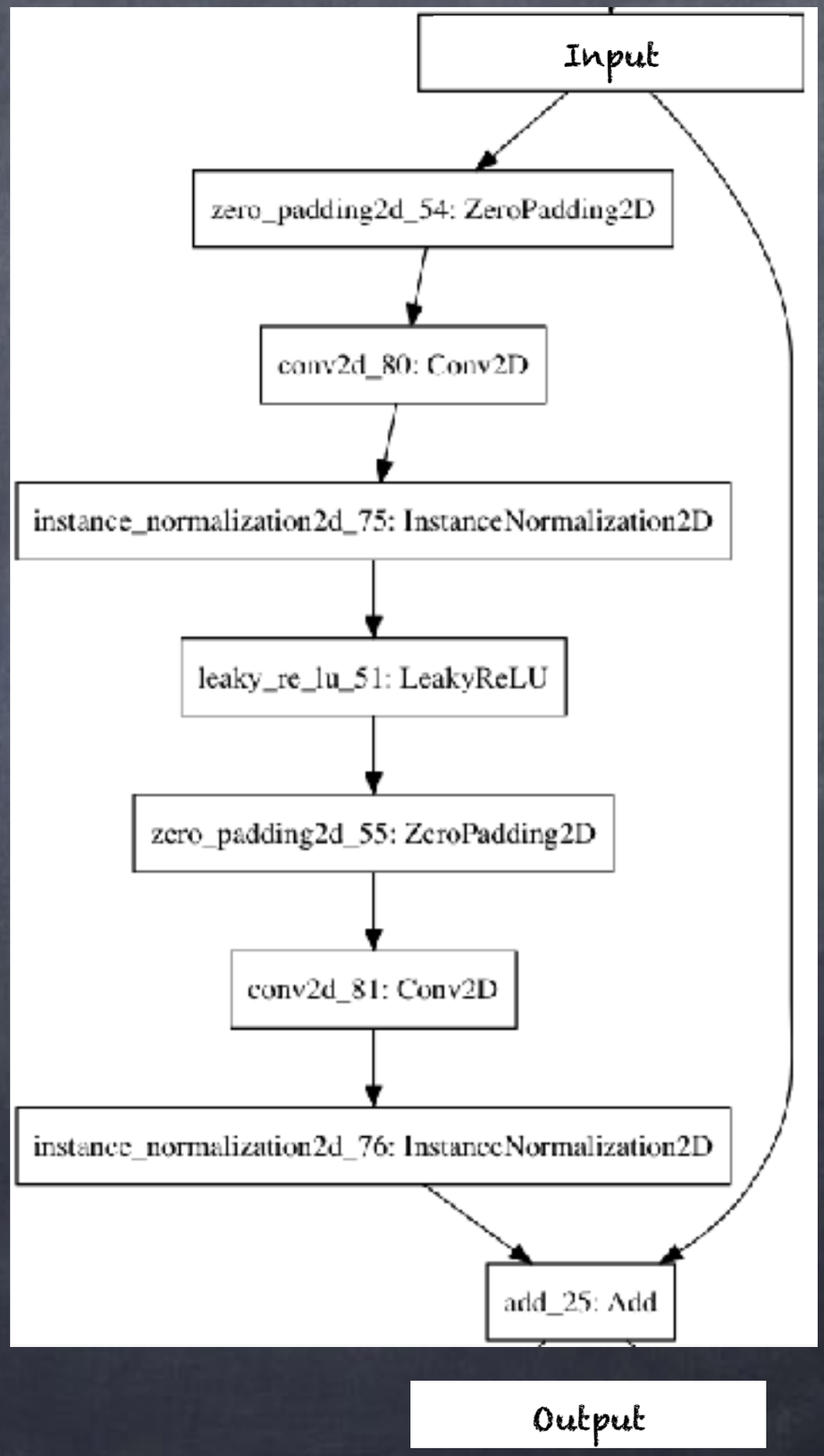




Generator

Transform

- "Residual Network"
- retained input feature
- extract other feature

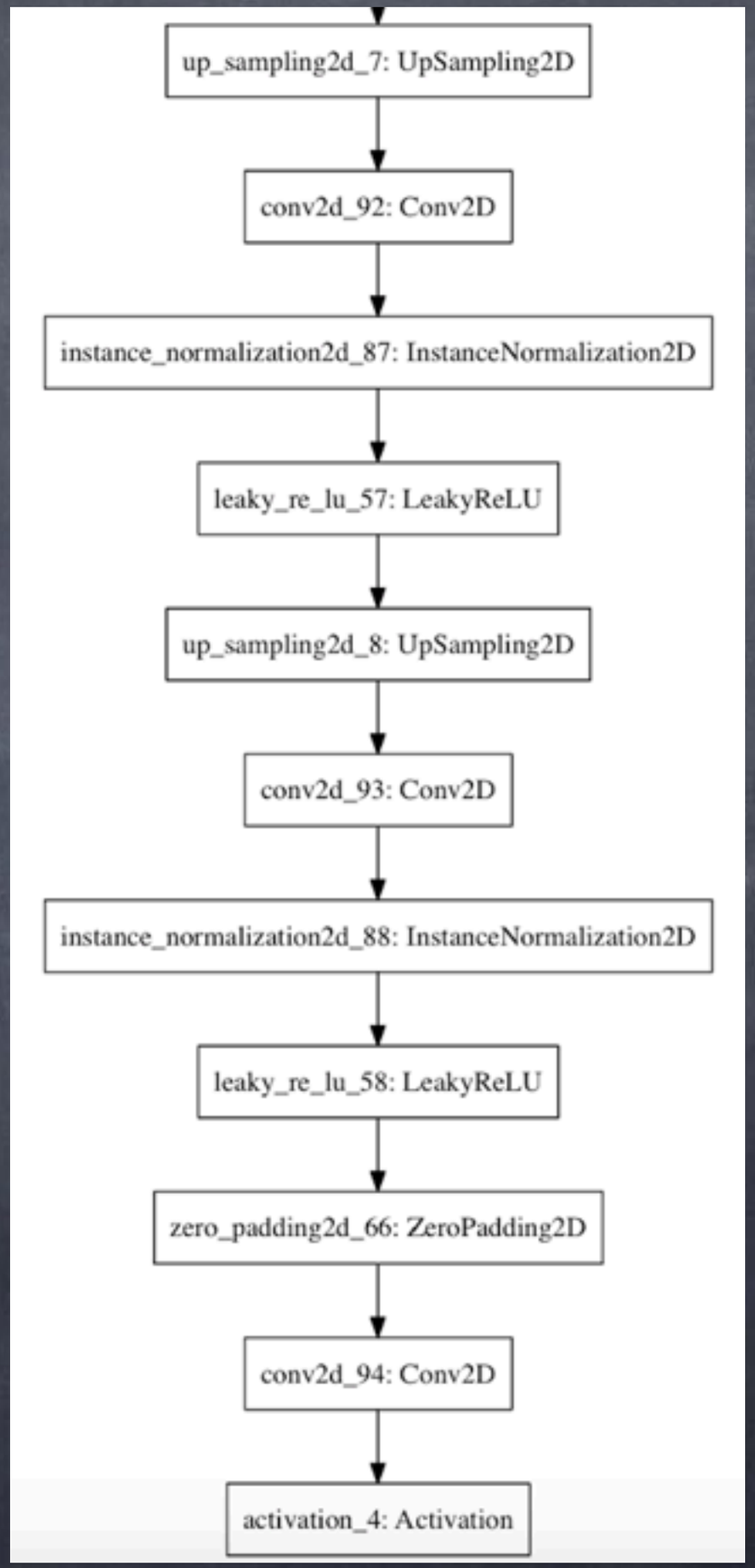


https://arxiv.org/abs/1512.03385

Generator

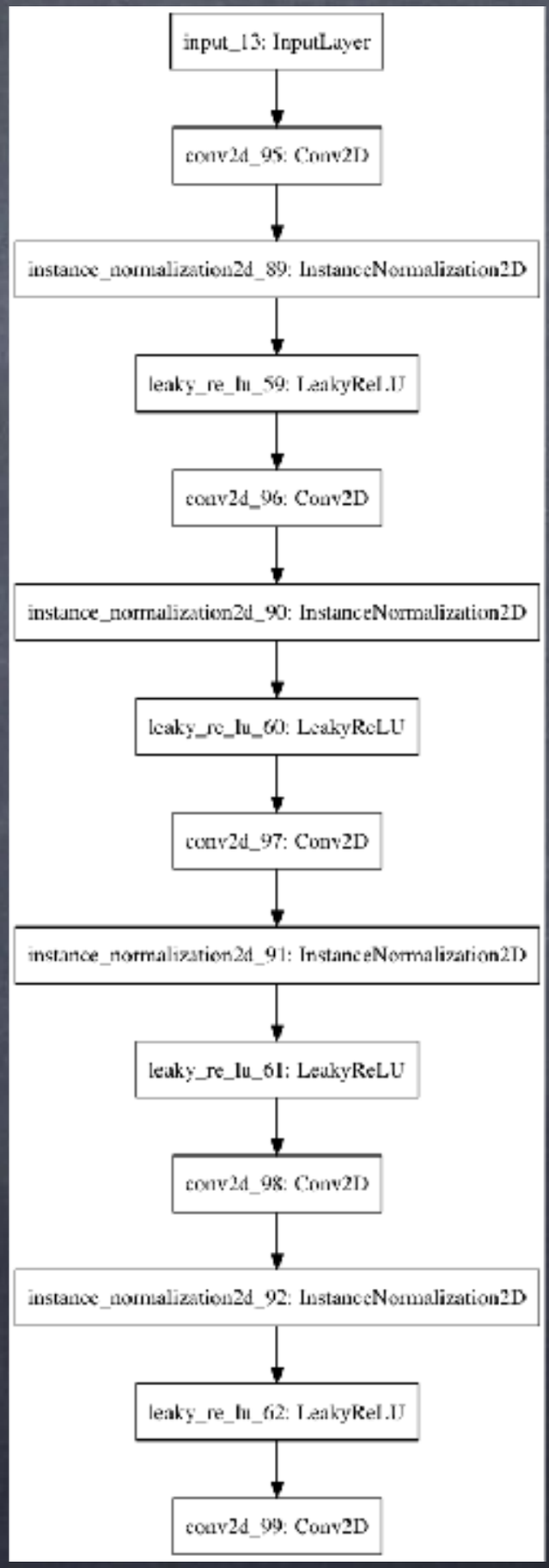
Decoder

- Dual with Encoder



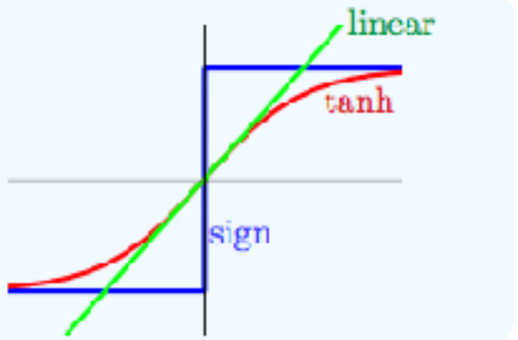
Discriminator

- Take image as input
- Discriminate real/fake image
- Only CNN + norm + ReLU



Small talk

Why not use Hyperbolic tangent?



$$\tanh(s) = \frac{\exp(s) - \exp(-s)}{\exp(s) + \exp(-s)}$$

ReLU(s) = s if s > 0 else 0

程式碼 — 1

```python
class Generator:

    def __init__(self, num_features=64, im_shape = (256, 256, 3), name='generator', Cmodel = None, res_cn
        self.nf = num_features
        self.img_size = im_shape
        self.name = name
        self.input_num = 0  # input_num
        self.model = self.build_model(res_cnt) if Cmodel is None else Cmodel

    def build_model(self, res_cnt, needSum = False):
        input_gen = Input(shape=self.img_size)

        # encoding
        nn = ZeroPadding2D((3, 3))(input_gen)
        nn = conv2d(nn, self.nf, (7, 7), strides=(1, 1))
        nn = conv2d(nn, self.nf*2, (3, 3), strides=(2, 2), padding='same')
        nn = conv2d(nn, self.nf*4, (3, 3), strides=(2, 2), padding='same')

        # transform
        for i in range(res_cnt):
            nn = build_resnet_block(nn, self.nf*4)

        # decoding
        nn = conv2d(nn, self.nf*2, (3, 3), strides=(1, 1), up_sample=2, padding='same')
        nn = conv2d(nn, self.nf, (3, 3), strides=(1, 1), up_sample=2, padding='same')
        if res_cnt == 6:
            nn = ZeroPadding2D((3, 3))(nn)
        nn = conv2d(nn, 3, (7, 7), strides=(1, 1), norm=False, relu=False, padding='valid' if res_cnt ==
        gen = Activation('tanh')(nn)

        generator = Model(inputs=input_gen, outputs=gen)
        if needSum:
            generator.summary()

        return generator
```

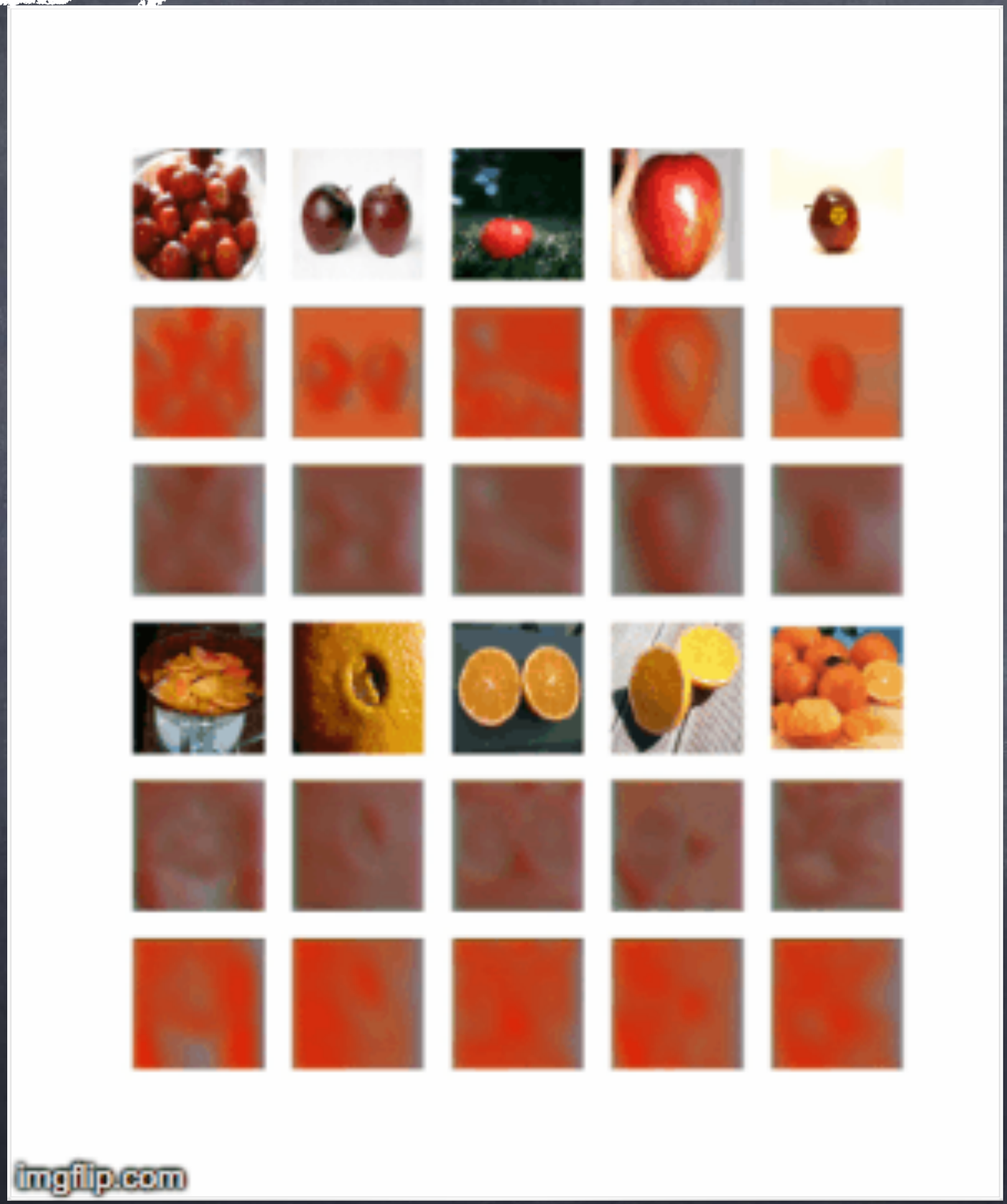
程式碼 – 2

```python
class Discriminator:
    def __init__(self, num_features=64, im_shape = (256, 256, 3), name='discriminator', Cmodel = None):
        self.nf = num_features
        self.img_size = im_shape
        self.name = name
        self.input_num = 0  # input_num
        self.model = self.build_model() if Cmodel is None else Cmodel

    def build_model(self, needSum = False, needSigmoid = False):
        filter_w = 4
        input_dis = Input(shape=self.img_size)

        nn = conv2d(input_dis, self.nf, (filter_w, filter_w), strides=(2, 2), padding='same',
            kernel_std=0.02, bias_init=0.0, relu_alpha=0.2)

        nn = conv2d(nn, self.nf*2, (filter_w, filter_w), strides=(2, 2), padding='same',
            kernel_std=0.02, bias_init=0.0, relu_alpha=0.2)

        nn = conv2d(nn, self.nf*4, (filter_w, filter_w), strides=(2, 2), padding='same',
            kernel_std=0.02, bias_init=0.0, relu_alpha=0.2)

        nn = conv2d(nn, self.nf*8, (filter_w, filter_w), strides=(1, 1), padding='same',
            kernel_std=0.02, bias_init=0.0, relu_alpha=0.2)

        nn = conv2d(nn, 1, (filter_w, filter_w), strides=(1, 1), padding='same',
            kernel_std=0.02, bias_init=0.0, relu=False, norm=False)
        if needSigmoid:
            nn = Activation('sigmoid')(nn)

        discriminator = Model(inputs=input_dis, outputs=nn)
        if needSum:
            discriminator.summary()
        return discriminator
```

```python
def setup_model(self):

    self.genB = Generator(name='GenA2B', im_shape=self.shp, num_features=self.ngf, res_cnt=6 if self.
    self.genA = Generator(name='GenB2A', im_shape=self.shp, num_features=self.ngf, res_cnt=6 if self.
    self.clf_A = Discriminator(name='clf_A', im_shape=self.shp, num_features=self.ndf)  # clf input0
    self.clf_B = Discriminator(name='clf_B', im_shape=self.shp, num_features=self.ndf)

    self.genA.model = init_network(self.genA.model)
    self.genB.model = init_network(self.genB.model)
    self.clf_A.model = init_network(self.clf_A.model)
    self.clf_B.model = init_network(self.clf_B.model)

    self.realA, self.realB = Input(self.shp), Input(self.shp)
    self.fakeA, self.fakeB = self.genA.model(self.realB), self.genB.model(self.realA)

    self.clf_genA = self.clf_A.model(self.fakeA)
    self.clf_genB = self.clf_B.model(self.fakeB)

    self.cyc_A = self.genA.model(self.fakeB)
    self.cyc_B = self.genB.model(self.fakeA)

    self.trainnerG = Model([ self.realA, self.realB ],
        [self.clf_genA, self.cyc_A, self.clf_genB, self.cyc_B])
    self.trainnerG.compile(optimizer=self.gopt,
        loss=['MSE', 'MAE', 'MSE', 'MAE'],
        loss_weights=[1, lambda_gan, 1, lambda_gan])

    realA, realB = Input(self.shp), Input(self.shp)
    fakeA, fakeB = Input(self.shp), Input(self.shp)

    clf_fakeA = self.clf_A.model(fakeA)
    clf_fakeB = self.clf_B.model(fakeB)
    clf_realA = self.clf_A.model(realA)
    clf_realB = self.clf_B.model(realB)
    self.trainnerD = Model([ realA, fakeA, realB, fakeB ],
        [clf_realA, clf_fakeA, clf_realB, clf_fakeB])
    self.trainnerD.compile(optimizer=self.dopt, loss='MSE')
```

成果－1

成果 - 2



未來展望

- Ostagram 轉換風格只需要花一分鐘，而且各個Domain只需要一張照片就可以Train
- 分析CycleGAN與Ostagram作法之優劣
- 不只圖片，聲音訊號轉換...等等，都可以拿來嘗試
- 幫自己買一顆比GeForce 750更好的GPU
- 或是...選擇其他領域來做

參考文獻

1. Generative Adversarial Networks (2014, Jun), Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio
2. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks (2017, March), Jun-Yan Zhu, Taesung Park, Phillip Isola Alexei A. Efros
3. Deep Residual Learning for Image Recognition (2015, Dec), Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
4. 李弘毅教授課程影片 https://www.youtube.com/watch?v=KSN4QYgAtao
5. https://hardikbansal.github.io/CycleGANBlog/

特別感謝

謝謝教授願意提供裝有CuDNN的實驗室server
讓我在崩潰的期末不用擔心實驗結果生不出來

Q&A

The End

Thanks for all your listening