使用對抗式生成神經網路生成動態天空材質

關 鍵 詞：對抗式生成神經網路、LSTM、天空材質

摘要

希望能使用 GAN（對抗式生成神經網路）配合 LSTM 訓練出能產生無止境的動態天空材質的模型。

壹、研究動機

生活在都市之中，放眼望去最貼近大自然的風景就是天空了吧。現今各種3D 技術發達，舉凡虛擬實境，3D 視覺引擎，如此之類到進步的最終都必然走向自然化的發展，當中被大部分人所熟悉的真實天空重現將會是一大考驗。

貳、研究目的

參、研究設備及器材

肆、研究過程或方法

天空材質主要運作的方式有兩種，球或立方體，俗稱天空球和天空盒。本研究將會採用天空球的模式生成材質。範例如下：

天空盒的材質將會是六張天空的照片，但是同樣有圖片扭曲程度上下不一的問題，因此選擇生成單張圖片即可滿足的天空球材質。

所以我可能需要三個模型來達成這個目標。

模型一：生成一個開始的天空

模型二：接收上一個模型的天空產生下一張

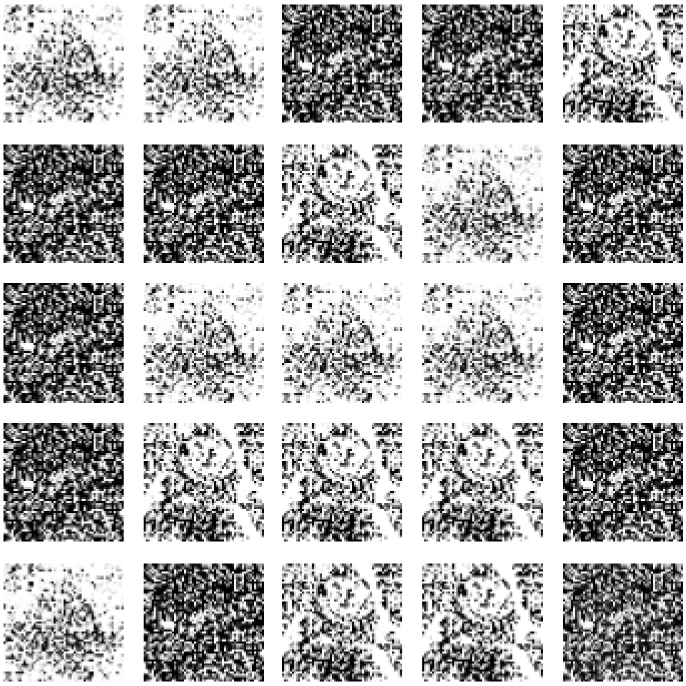
模型三：扭曲天空照片使得其符合天空球的格式。

當中應該只有模型一和二需要大量訓練。模型三應該可以用手刻程式碼。

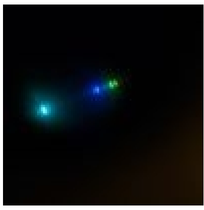
資料集：http://sky.hdrdb.com/

經過一番搜索後，我很幸運的找到了我要的大量資料。提供者很貼心的拍攝時就採用高動態範圍成像，並且一開始就是扭曲成我們要的格式了。所以問題將會是怎麼選擇訓練的亮度。

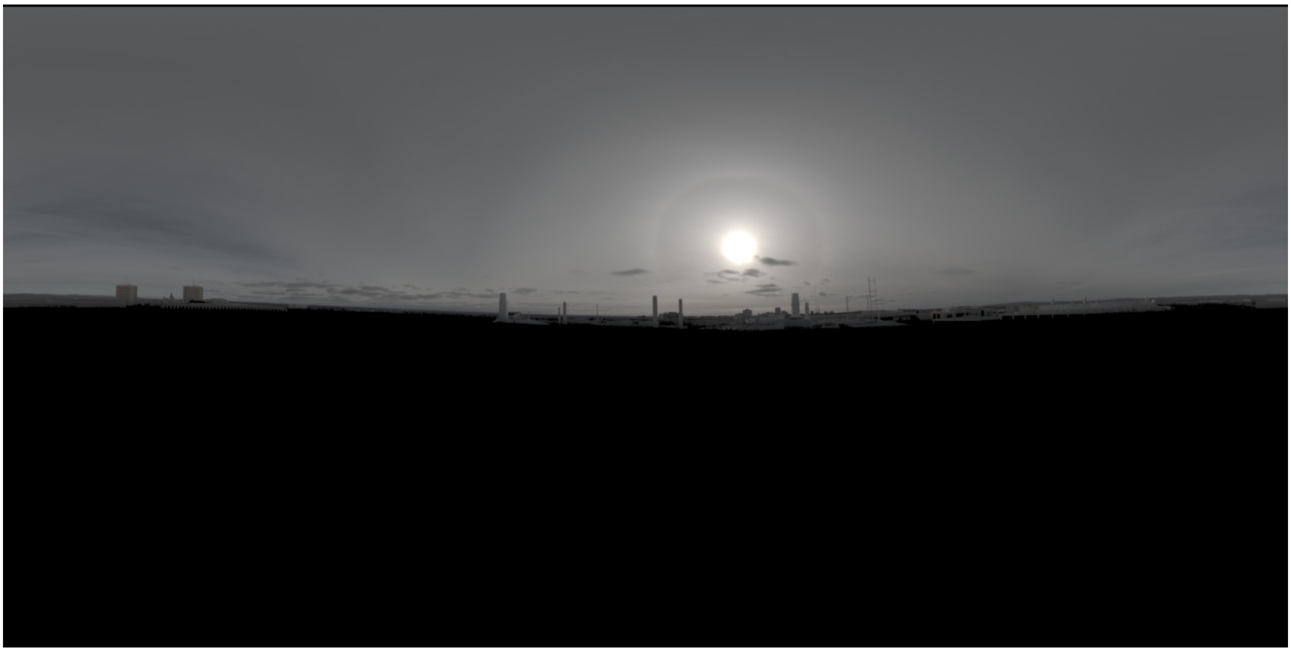
以往選擇 HDR 亮度都是採用人手工調整亮度，並且不同區域選取不同亮度以達到更佳的圖片品質的效果。

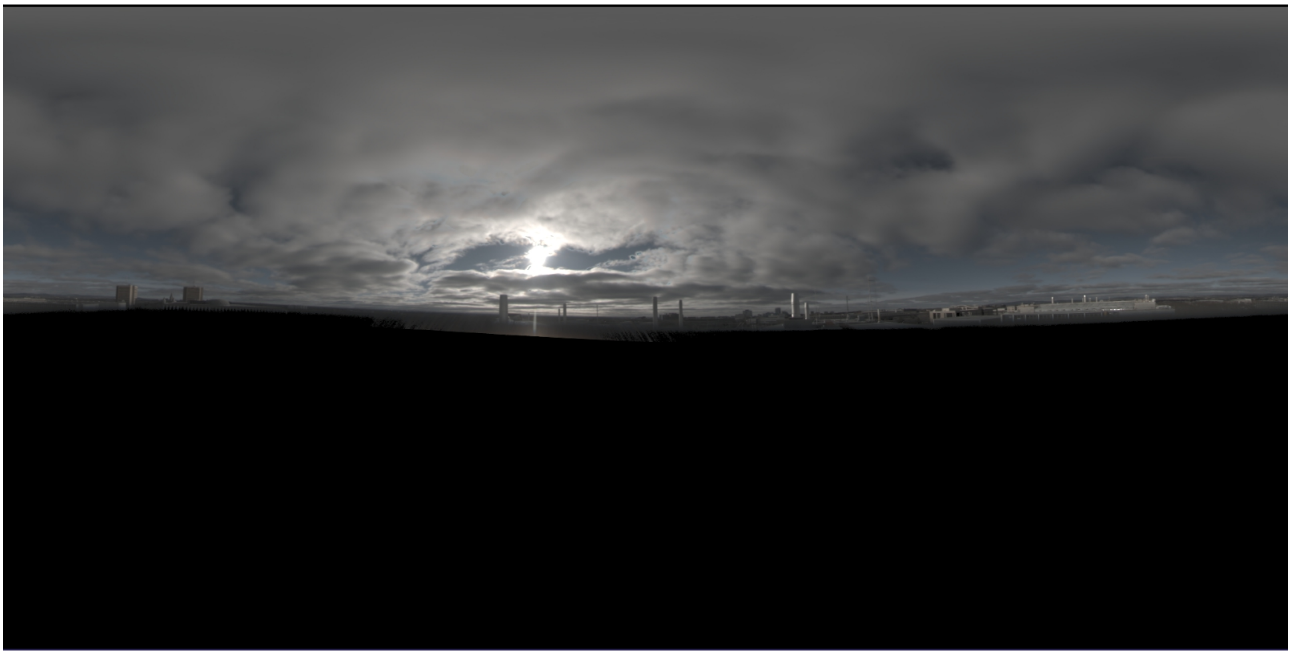


這是將全班臉書照片融合到一半的結果，可以看到某些人的特徵。像是這個

和 .，可以看到有明顯的輪廓浮現，但是並不完全一樣。

我取得的資料大概像這樣

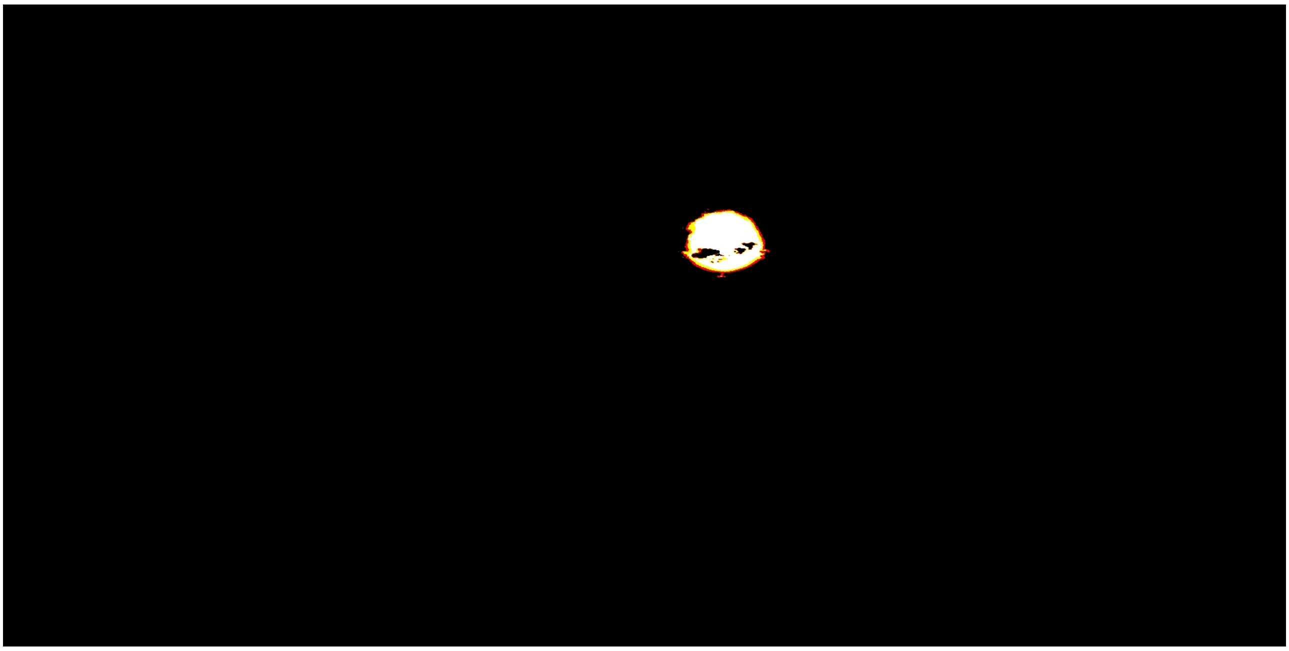




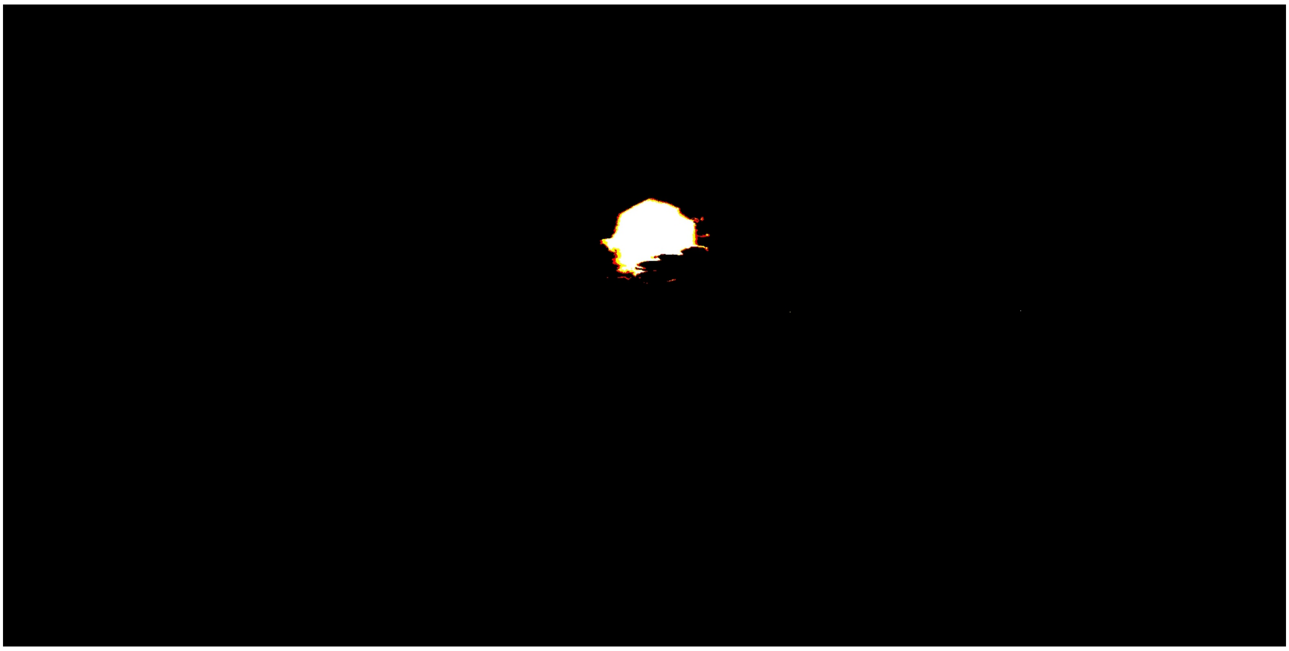
和這樣，目前需要先將他提供的檔案轉成圖片才有辦法訓練。

以上為手動調整亮度之後的結果。目前嘗試用程式將全部像素的亮度調整成適合觀看的

樣子但是都失敗了，大概長



和



這樣。

我一定是哪邊搞錯了。

伍、研究結果

陸、討論

一、文獻探討

在我們的研究中，我們生成的是上下扭曲程度不同的天空球材質，而在[1]這個程式可以做到生成一個不存在的天空的風景照，而在[2]這篇論文，他說他可以透過讓模型學習自然環境的物理行為，產生下一張天空的照片。這兩篇嚴格來講並不是生成天空球的材質，所以應該算沒有人做過的題目吧。

備註：我使用 generate sky texture gan 和 generate dynamic sky texture gan 作為關鍵字搜尋。中文的方面沒有查到用人工智慧生成天空照片或材質的文獻，但是有查到不使用人工智慧的。

柒、結論

捌、參考資料及其他

一、參考資料

[1]https://github.com/aleju/sky-generator (生成單張有天空的風景照)

[2]http://img.cs.uec.ac.jp/pub/conf19/191126horita_0.pdf (預測下一個瞬間自然景物的模樣，但是似乎只有做成功天空的照片)

二、程式碼

```
# -*- coding: utf-8 -*-
""" Simple implementation of Generative Adversarial Neural Network """
import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from IPython.core.debugger import Tracer
import tensorflow as tf
from keras.datasets import mnist
from keras.layers import Input, Dense, Reshape, Flatten, Dropout
from keras.layers import BatchNormalization
from keras.layers.advanced_activations import LeakyReLU
from keras.models import Sequential
from keras.optimizers import Adam

import matplotlib.pyplot as plt



gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    # Restrict TensorFlow to only allocate 1GB of memory on the first GPU
    try:
        tf.config.experimental.set_virtual_device_configuration(
                gpus[0],
                [tf.config.experimental.VirtualDeviceConfiguration(memory_limit=4096)])
        logical_gpus = tf.config.experimental.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
    except RuntimeError as e:
        # Virtual devices must be set before GPUs have been initialized
        print(e)



def preprocessing_imgdatagen(x,imgsize=256):
        '''==============================================
        (+)a. rescale pixel value from
        [0,255] into [-1,1]
        (-)b. doing histogram equalization by skimage function
        ** the skimage function "exposure.equalize_hist" will
           turn pixcel value to [0,1] so the step (a) wasnt used.
        ===============================================''' 

        #x = rgb2gray(x)

        #imghist = exposure.equalize_hist(x)
        #imghist = imghist.astype(np.float32)
        #x = imghist

        x /= 255.
        x -= 0.5
        x *= 2.

        #print(x.shape)
        return x


    def get_datagen(DATA_PATH, BATCH_SIZE, RESIZE=(48,48)):
        classnum = os.listdir(f'{DATA_PATH}train\\')    # os.path.join(DATA_PATH,'train')
        datalen = 0
        for i in classnum:
            datalen += len(os.listdir(f'{DATA_PATH}train\\{i}')) #
os.path.join(DATA_PATH,'train',i)

        train_datagen = ImageDataGenerator(
                            preprocessing_function=preprocessing_imgdatagen,
                            rotation_range=0,
                            width_shift_range=0.01,
                            height_shift_range=0.01,
                            shear_range=0.01,
                            zoom_range=0.01,
                            channel_shift_range=0.01,
                            horizontal_flip=False,
                            vertical_flip=False
                            )
        train_gen = train_datagen.flow_from_directory(
                            DATA_PATH+'train',
                            target_size=RESIZE,
                            batch_size=BATCH_SIZE,

classes=['0_plane','1_car','2_bird','3_cat','4_deer','5_dog','6_frog','7_horse','8_ship','9_truck'],
                            # class_mode='binary'
                            )
        return train_gen, datalen



    def get_original_data(DATA_PATH, times=1, data_size=48):
        data_path = DATA_PATH
        Xtrain = []
        class_num = 0

        for img in os.listdir(f'{data_path}\\'):
```

```
        image = load_img(f'{data_path}\\{img}',target_size=(data_size, data_size))
        image = np.array(image)
        image = np.dot(image[...,:3], np.ones((3)))
        image = image.astype('float32')
        image /= 255
        image -= 0.5
        image *= 2
        Xtrain.append(image)

    Xtrain = Xtrain * times
    Xtrain = np.array(Xtrain)
    return Xtrain



class GAN(object):
    """ Generative Adversarial Network class """
    def __init__(self, width=48, height=48, channels=1):

        self.width = width
        self.height = height
        self.channels = channels

        self.shape = (self.width, self.height, self.channels)

        self.optimizer = Adam(lr=0.0002, beta_1=0.5, decay=8e-8)

        self.G = self.__generator()
        self.G.compile(loss='binary_crossentropy', optimizer=self.optimizer)


        self.D = self.__discriminator()
        self.D.compile(loss='binary_crossentropy', optimizer=self.optimizer,
metrics=['accuracy'])


        self.stacked_generator_discriminator = self.__stacked_generator_discriminator()


        self.stacked_generator_discriminator.compile(loss='binary_crossentropy',
optimizer=self.optimizer)



    def __generator(self):
        """ Declare generator """


        model = Sequential()
        model.add(Dense(256, input_shape=(100,)))
        model.add(LeakyReLU(alpha=0.2))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Dense(512))
        model.add(LeakyReLU(alpha=0.2))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Dense(self.width   * self.height * self.channels, activation='tanh'))
        model.add(Reshape((self.width, self.height, self.channels)))


        return model
```

```
def __discriminator(self):
    """ Declare discriminator """

    model = Sequential()
    model.add(Flatten(input_shape=self.shape))
    model.add(Dense((self.width * self.height * self.channels), input_shape=self.shape))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(np.int64((self.width * self.height * self.channels)/2)))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(np.int64((self.width * self.height * self.channels)/2)))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(np.int64((self.width * self.height * self.channels)/2)))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(1, activation='sigmoid'))
    model.summary()

    return model

def __stacked_generator_discriminator(self):

    self.D.trainable = False

    model = Sequential()
    model.add(self.G)
    model.add(self.D)

    return model
    def train(self, X_train, epochs=200000, batch = 32, save_interval = 10000):
        cnt = 0
        while True:

            ## train discriminator
            random_index = np.random.randint(0, len(X_train) - np.int64(batch/2))
            legit_images = X_train[0:np.int64(batch/2)].reshape(np.int64(batch/2),
self.width, self.height, self.channels)

            gen_noise = np.random.normal(0, 1, (np.int64(batch/2), 100))
            syntetic_images = self.G.predict(gen_noise)

            x_combined_batch = np.concatenate((legit_images, syntetic_images))
            y_combined_batch = np.concatenate((np.ones((np.int64(batch/2), 1)),
np.zeros((np.int64(batch/2), 1))))

            d_loss = self.D.train_on_batch(x_combined_batch, y_combined_batch)



            # train generator

            noise = np.random.normal(0, 1, (batch, 100))
            y_mislabled = np.ones((batch, 1))

            g_loss = self.stacked_generator_discriminator.train_on_batch(noise,
y_mislabled)
```

```
            print ('epoch: %d, [Discriminator :: d_loss: %f], [ Generator :: loss: %f]' % (cnt,
d_loss[0], g_loss) + ' '*10, end='\r')

            if cnt % save_interval == 0:
                self.plot_images(save2file=True, step=cnt)
            cnt += 1


    def plot_images(self, save2file=False, samples=25, step=0):
        ''' Plot and generated images '''
        if not os.path.exists("./images"):
            os.makedirs("./images")
        filename = "./images/mnist_%d.png" % step
        noise = np.random.normal(0, 1, (samples, 100))

        images = self.G.predict(noise)

        plt.figure(figsize=(5, 5))

        for i in range(images.shape[0]):
            plt.subplot(5, 5, i+1)
            image = images[i, :, :, :]
            image = np.reshape(image, [self.height, self.width, self.channels])
            plt.imshow(image, cmap='gray')
            plt.axis('off')
        plt.tight_layout()
        if save2file:
            plt.savefig(filename)
            plt.close('all')
        else:
            plt.show()


if __name__ == '__main__':
    '''
    (X_train, _), (_, _) = mnist.load_data()

    # Rescale -1 to 1
    X_train = (X_train.astype(np.float32) - 127.5) / 127.5
    X_train = np.expand_dims(X_train, axis=3)
    Xtrain = []
    Ytrain = []
    data_path = 'C:\\python_environments\\tf-gpu\\source\\cifar10\\cifar10\\'
    data_num = 100
    traingen, datalen = get_datagen(data_path, data_num)
    n=1
    for images, labels in traingen:
      # plt.figure(figsize=(8,8))
      # print(images)
        for i in range(data_num):
            Xtrain.append(images[i])
            Ytrain.append(labels[i])
            # plt.show()
            if n == 1:
                break
            else:
                n += 1
        # int(images[-1])
        Xtrain = np.array(Xtrain)
        Ytrain = np.array(Ytrain)
        '''
        gan = GAN()
        Xtrain = get_original_data('C:\\python_environments\\tf-gpu\\source\\MIA\\1509_s\\1509_s')
        # Xtrain = get_original_data('C:\\python_environments\\tf-gpu\\source\\MIA\\combine', 10)
        gan.train(Xtrain)
```