2021年臺灣國際科學展覽會
研究報告

科別：電腦科學與資訊工程科

作品名稱：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

使用對抗式生成神經網路生成動態天空材質

一、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。

二、內文

（一）、前言

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。

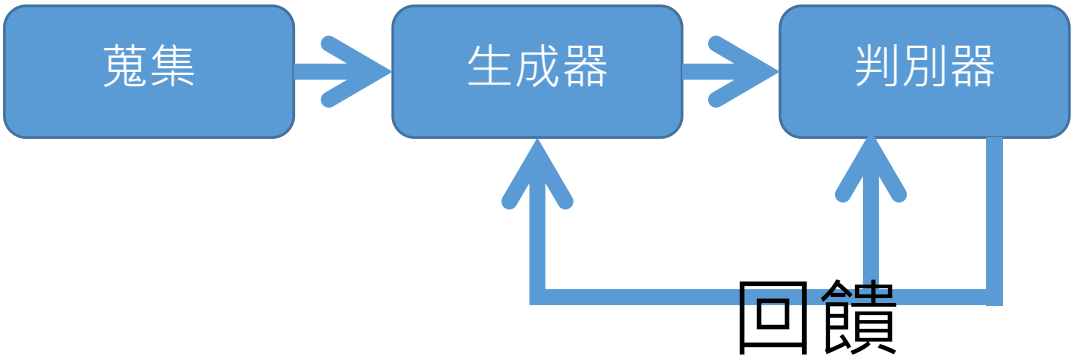
（二）、研究方法或過程

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：

此為藝術家之作品，下圖為真實影像。

至於不採用天空和模式的原因為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。

對於對抗式生成神經網路可繪製訓練流程圖如下：



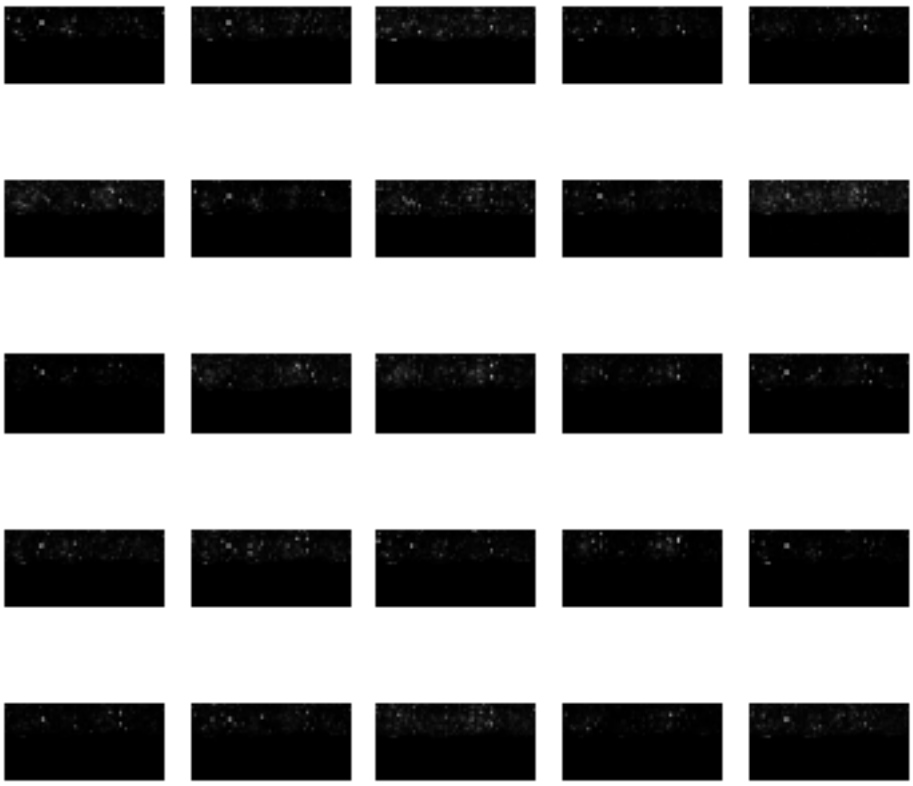
經過一番搜索後，我很幸運的找到了我要的大量資料。提供者很貼心的拍攝時就採用高動態範圍成像，並且一開始就是扭曲成我們要的格式了。研究的一大重點是如何拼裝影像中不同亮度的部分。

上圖為目前能夠批次轉換的結果，下圖為手工分區調整的結果。

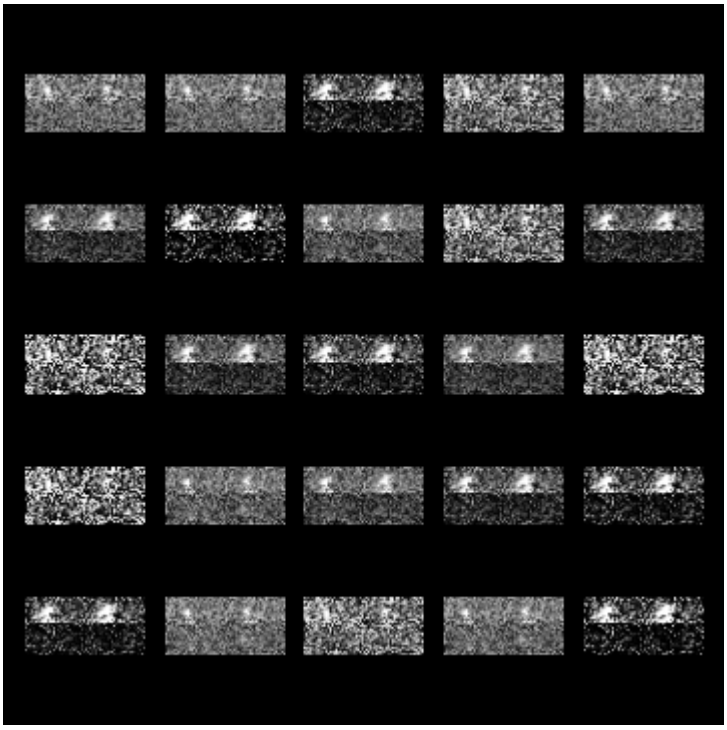
上方的影像為理想的轉換，可以發現亮處較暗，暗處較亮，整體清晰許多，也比較自然。下方則明暗對比度過高，HDR的特色就是同一張影像擁有許多不同的曝光條件。理想的轉換會取各種版本中較好的部分拼裝在一起。

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。可以看到生成的影像很暗，因為很多訓練的資料很多都是全黑的夜空，或是只有太陽特別明顯的影像。

這導致了訓練資料品質欠佳，目前尚無法發揮HDR資料的優勢，取得各區域亮度適中的影像，且來源中有部分時段的天空是沒有日光的。故訓練成效不彰。可以在下圖中看到生成的影像非常漆黑。



使用我手工挑選適合的圖片 進行訓練後，整體的形狀明顯許多，圖中雜訊較多是因為正在訓練中。



但是現在天空上出現了兩個很像是太陽的東西，推測應該是判別器同時對某些太陽在左邊某位置的影像和太陽在右邊某位置的影像很敏感。導致生成器學會生成兩顆太陽。

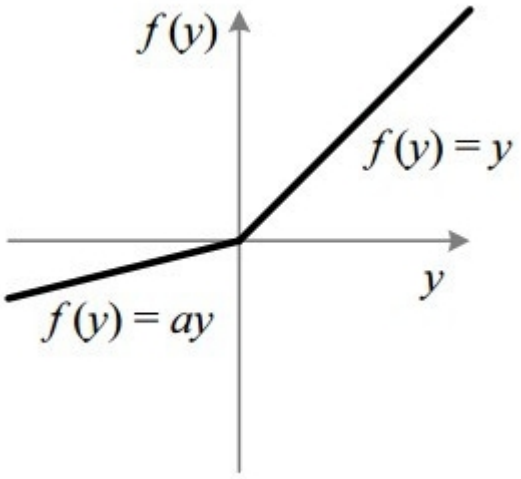
可以發現這個形狀和生成出來強勢的特徵形狀很像。

1. 生成器

由於目前的架構限制，只能生成大小較小的黑白照片。且採用多層全連接層生成影像。模型架構圖如下。

Layer (type)	Output Shape	Param #
dense_32 (Dense)	(None, 256)	25856
leaky_re_lu_24 (LeakyReLU)	(None, 256)	0
batch_normalization_8 (Batch	(None, 256)	1024
dense_33 (Dense)	(None, 512)	131584
leaky_re_lu_25 (LeakyReLU)	(None, 512)	0
batch_normalization_9 (Batch	(None, 512)	2048
dense_34 (Dense)	(None, 2048)	1050624
reshape_4 (Reshape)	(None, 64, 32, 1)	0

當中我使用LeakyReLU，因為ReLU會在 $x < 0$ 時發生梯度消失的問題。LeakyReLU示意圖如下。我使用的a值為0.2。



2. 判別器

經由多層全連接層後，判斷這張影像是否在訓練集內，並且將結果回傳給生成器，作為改進的依據。架構圖如下。

flatten_2 (Flatten)	(None, 2048)	0
dense_19 (Dense)	(None, 2048)	4196352
leaky_re_lu_14 (LeakyReLU)	(None, 2048)	0
dense_20 (Dense)	(None, 1024)	2098176
leaky_re_lu_15 (LeakyReLU)	(None, 1024)	0
dense_21 (Dense)	(None, 1024)	1049600
leaky_re_lu_16 (LeakyReLU)	(None, 1024)	0
dense_22 (Dense)	(None, 1024)	1049600
leaky_re_lu_17 (LeakyReLU)	(None, 1024)	0
dense_23 (Dense)	(None, 1)	1025

3. 影像大小

經過幾次修改後，我犧牲了一點運算的效率，換取面積四倍的結果，雖然目前已經有方法可以使的生成的影像任意大，但是相對的對效率的犧牲過於慘烈，所以目前不採用。

方法一：

將批次載入到GPU中的數據減少，從32張減少到2張，成功取得將圖片放大四倍的空間。犧牲了少許計算的效能和訓練的成效。

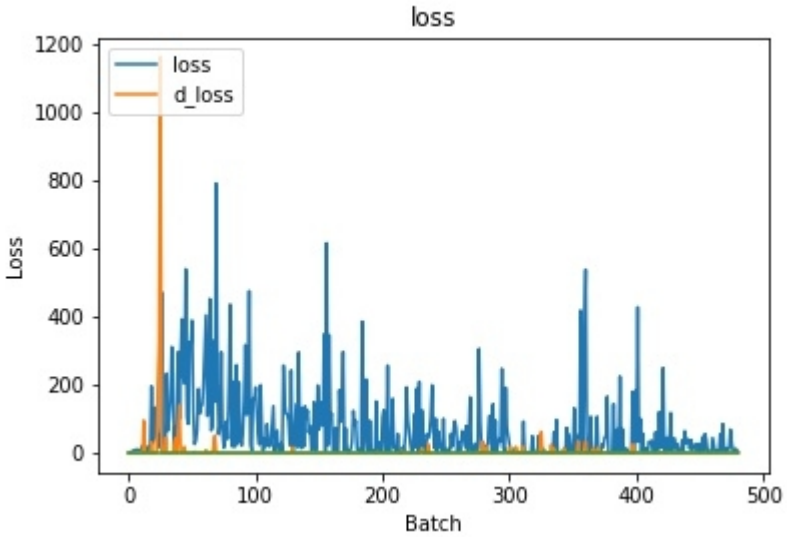
方法二：

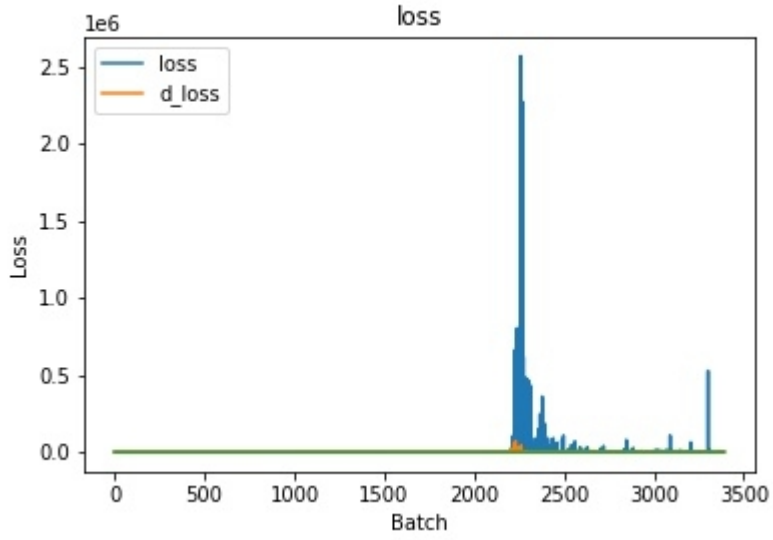
不使用GPU，純粹將圖片載入記憶體，使用CPU運算，效率非常差，但是生成的圖片可以非常大。

Loss圖

d_loss 是判別器的loss，該值越低表示此模型正確分辨影像的能力越高。

Loss 是生成器的loss，該值越低表示判別器認為這張圖片越像真實的圖片 。



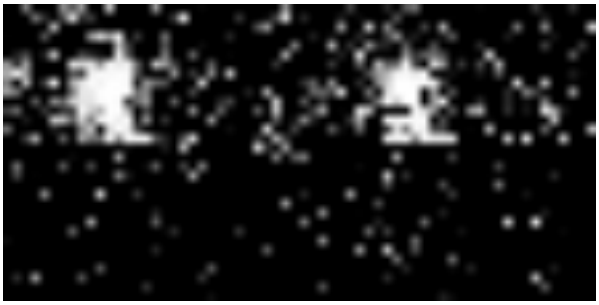


可以發現當訓練回數突破2000時，結果趨向穩定，生成器開始找不到方向學習。
當前最主要的目標應該是改善判別器的模型，目前有了幾種能突破記憶體限制的方法一定大有助益。

（三）、研究結果與討論

目前成果：

訓練十分鐘：



訓練十小時：

（四）、結論與應用

（五）、參考文獻

1. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, "Generative Adversarial Networks" Jun. 2014

（六）、附錄

1. 模型-生成器

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	25856
leaky_re_lu (LeakyReLU)	(None, 256)	0
batch_normalization (BatchNo	(None, 256)	1024
dense_1 (Dense)	(None, 512)	131584
leaky_re_lu_1 (LeakyReLU)	(None, 512)	0
batch_normalization_1 (Batch	(None, 512)	2048
dense_2 (Dense)	(None, 8192)	4202496
reshape (Reshape)	(None, 128, 64, 1)	0

2. 模型-判別器

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 8192)	0
dense_3 (Dense)	(None, 8192)	67117056
leaky_re_lu_2 (LeakyReLU)	(None, 8192)	0
dense_4 (Dense)	(None, 4096)	33558528
leaky_re_lu_3 (LeakyReLU)	(None, 4096)	0
dense_5 (Dense)	(None, 4096)	16781312
leaky_re_lu_4 (LeakyReLU)	(None, 4096)	0
dense_6 (Dense)	(None, 4096)	16781312
leaky_re_lu_5 (LeakyReLU)	(None, 4096)	0
dense_7 (Dense)	(None, 1)	4097

3. 主程式

```
# -*- 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 cv2
import matplotlib.pyplot as plt

# import os
os.environ["CUDA_VISIBLE_DEVICES"] = "0"


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, width=48, height=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=(width, height))
        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=128, height=64, 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.history = {
            "d_loss": [],
            "loss": []
        }

        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)))
    model.summary()
    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 = 2, save_interval = 10):
    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)
        self.history["loss"].append(g_loss)
```

```
            self.history["d_loss"].append(d_loss)
            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 '''
        path =  "C:\\python_environments\\tf-gpu\\source\\MIA\\SkyGen\\output\\"


        filename = os.path.join(path, "%d.png" % step)
        noise = np.random.normal(0, 1, (samples, 100))

        images = self.G.predict(noise)

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


        nor_noise = np.array([[0.5] * 100])

        img = self.G.predict(nor_noise)
        img = img[0, :, :, :] * 256
        img = np.reshape(img, [64, 128, 1])
        cv2.imwrite(os.path.join(path, f'{step}output.png'), img)

        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()

        plt.plot(self.history['loss'])
        plt.plot(self.history['d_loss'])
        plt.title(f'loss')
        plt.ylabel('Loss')
        plt.xlabel('Batch')
        plt.legend(['loss', 'd_loss'], loc='upper left')
        plt.savefig(os.path.join(path, f'{step}loss.jpg'))
        plt.close()


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\\SkyGen\\data\\good_sky\\',
width=gan.width, height=gan.height)
    # Xtrain = get_original_data('C:\\python_environments\\tf-gpu\\source\\MIA\\combine', 10)
    gan.train(Xtrain)
```