Sketch To Face Generation

Implementing the Conditional Generative Adversarial Networks for Sketch to Face transformation over CUHK student dataset.

Installations

```
pip install scipy==1.1.0
```

```
Collecting scipy==1.1.0
  Downloading https://files.pythonhosted.org/packages/40/de/0c22c6754370ba6b1fa8e53b
     |████████████████████████████████| 31.2MB 103kB/s
Requirement already satisfied: numpy>=1.8.2 in /usr/local/lib/python3.7/dist-package
ERROR: plotnine 0.6.0 has requirement scipy>=1.2.0, but you'll have scipy 1.1.0 whic
ERROR: albumentations 0.1.12 has requirement imgaug<0.2.7,>=0.2.5, but you'll have i
Installing collected packages: scipy
  Found existing installation: scipy 1.4.1
    Uninstalling scipy-1.4.1:
      Successfully uninstalled scipy-1.4.1
Successfully installed scipy-1.1.0
```

```
pip install git+https://www.github.com/keras-team/keras-contrib.git
```

```
Collecting git+https://www.github.com/keras-team/keras-contrib.git
  Cloning https://www.github.com/keras-team/keras-contrib.git to /tmp/pip-req-build-
  Running command git clone -q https://www.github.com/keras-team/keras-contrib.git /
Requirement already satisfied: keras in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: scipy>=0.14 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from h
Building wheels for collected packages: keras-contrib
  Building wheel for keras-contrib (setup.py) ... done
  Created wheel for keras-contrib: filename=keras_contrib-2.0.8-cp37-none-any.whl si
  Stored in directory: /tmp/pip-ephem-wheel-cache-h2m0gwt0/wheels/11/27/c8/4ed56de7b
Successfully built keras-contrib
Installing collected packages: keras-contrib
Successfully installed keras-contrib-2.0.8
```

```
from __future__ import print_function, division

from keras_contrib.layers.normalization.instancenormalization import InstanceNormalizatior
from keras.layers import Input, Dense, Reshape, Flatten, Dropout, Concatenate, BatchNormal
from keras.layers.advanced_activations import LeakyReLU
from keras.layers.convolutional import UpSampling2D, Conv2D
from keras.models import Sequential, Model
from keras.optimizers import Adam
```

```
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import load_img
from keras.models import load_model

from sklearn.utils import shuffle
import matplotlib.pyplot as plt
import numpy as np
import datetime
import natsort
import scipy
import sys
import os
import cv2

import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import cv2
import os
import numpy as np
import glob
from PIL import Image
import natsort
```

Mounting Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
%cd /content/drive/MyDrive/
```

```
/content/drive/MyDrive
```

Unzipping Dataset

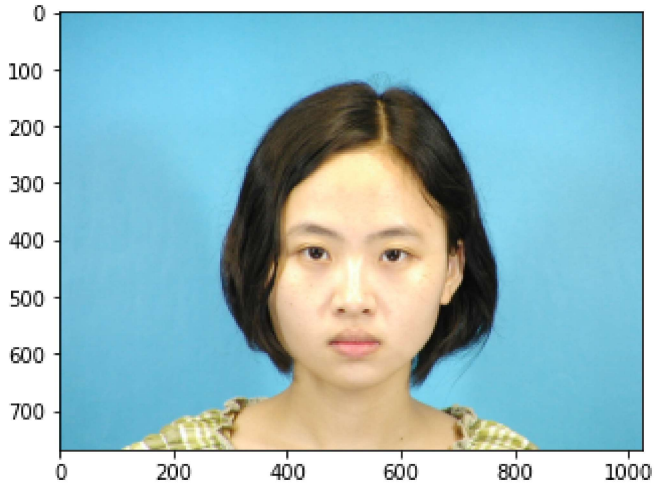
```
!unzip CUHK.zip
```

Dataset Display

```
# dataset path
# b_photo_path = '/content/drive/MyDrive/CUHK/Dataset/Augmented photo/'
# b_sketch_path = '/content/drive/MyDrive/CUHK/Dataset/Augmented sketch/'

b_photo_path = '/content/drive/MyDrive/CUHK/Training photo/'
b_sketch_path = '/content/drive/MyDrive/CUHK/Training sketch/'
blue_photo = load_filename(b_photo_path)
```

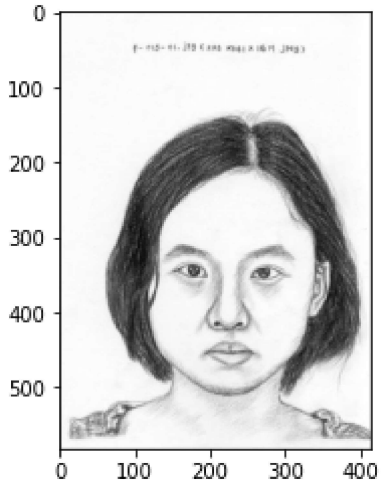
```
blue_sketch = load_filename(b_sketch_path)


plt.imshow(cv2.cvtColor(cv2.imread(blue_photo[10]).astype('uint8'), cv2.COLOR_BGR2RGB))
cv2.imread(blue_photo[10]).shape
```

```
(768, 1024, 3)
```




```
plt.imshow(cv2.cvtColor(cv2.imread(blue_sketch[10]).astype('uint8'), cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7fd1b45243d0>
```



Augmenting Dataset

```
def random_shearing(img, num, border):
    rows = img.shape[0]
    cols = img.shape[1]
    if num == 0:
        pts1 = np.float32([[5,5],[20,5],[2,20]])
        pts2 = np.float32([[10,10],[20,5],[5,25]])
    elif num == 1:
        pts1 = np.float32([[5,5],[15,5],[2,20]])
        pts2 = np.float32([[5,10],[10,10],[5,25]])
    elif num == 2:
        pts1 = np.float32([[5,5],[15,5],[5,20]])
        pts2 = np.float32([[5,10],[10,10],[5,25]])
```

```
        elif num == 3:
            pts1 = np.float32([[5,5],[10,5],[2,20]])
            pts2 = np.float32([[5,10],[10,10],[5,25]])
        elif num == 4:
            pts1 = np.float32([[5,5],[10,5],[2,20]])
            pts2 = np.float32([[5,10],[10,10],[5,30]])
        else:
            pts1 = np.float32([[5,5],[10,5],[10,20]])
            pts2 = np.float32([[5,10],[10,10],[5,30]])
        M = cv2.getAffineTransform(pts1,pts2)
        return cv2.warpAffine(img, M, (cols,rows), borderValue=border)

    def random_rotation(img, degree, border):
        rows = img.shape[0]
        cols = img.shape[1]
        M = cv2.getRotationMatrix2D((cols/2,rows/2),degree,1)
        return cv2.warpAffine(img,M,(cols,rows), borderValue=border)


    def transform_image(img, skt, ang_range, shear_range, trans_range):
        '''
        This function transforms images to generate new images.
        The function takes in following arguments,
        1- Image
        2- ang_range: Range of angles for rotation
        3- shear_range: Range of values to apply affine transform to
        4- trans_range: Range of values to apply translations over.

        A Random uniform distribution is used to generate different parameters for transformat

        '''
        # Rotation

        ang_rot = np.random.uniform(ang_range)-ang_range/2
        rows,cols,ch = img.shape
        Rot_M = cv2.getRotationMatrix2D((cols/2,rows/2),ang_rot,1)

        # Translation
        tr_x = trans_range*np.random.uniform()-trans_range/2
        tr_y = trans_range*np.random.uniform()-trans_range/2
        Trans_M = np.float32([[1,0,tr_x],[0,1,tr_y]])

        # Shear
        pts1 = np.float32([[5,5],[20,5],[5,20]])

        pt1 = 5+shear_range*np.random.uniform()-shear_range/2
        pt2 = 20+shear_range*np.random.uniform()-shear_range/2

        pts2 = np.float32([[pt1,5],[pt2,pt1],[5,pt2]])

        shear_M = cv2.getAffineTransform(pts1,pts2)

        # Border
        idx = 0
        border_img = tuple([int(img[idx][0][0]), int(img[idx][0][1]), int(img[idx][0][2])])
        border_skt = tuple([int(skt[0][0][0]), int(skt[0][0][1]), int(skt[0][0][2])])
```

```
    border_skt = tuple([int(skt[0][0][0]), int(skt[0][0][1]), int(skt[0][0][2])])

    img = cv2.warpAffine(img,Rot_M,(cols,rows), borderValue=border_img)
    img = cv2.warpAffine(img,Trans_M,(cols,rows), borderValue=border_img)
    img = cv2.warpAffine(img,shear_M,(cols,rows), borderValue=border_img)

    skt = cv2.warpAffine(skt,Rot_M,(cols,rows), borderValue=border_skt)
    skt = cv2.warpAffine(skt,Trans_M,(cols,rows), borderValue=border_skt)
    skt = cv2.warpAffine(skt,shear_M,(cols,rows), borderValue=border_skt)

    return img, skt


sketch_dir = '/content/drive/MyDrive/CUHK/Dataset/Augmented sketch/'
photo_dir = '/content/drive/MyDrive/CUHK/Dataset/Augmented photo/'

if not os.path.exists(sketch_dir):
    os.mkdir(sketch_dir)

if not os.path.exists(photo_dir):
    os.mkdir(photo_dir)

p_filenames = glob.glob('/content/drive/MyDrive/CUHK/Training photo/*')
s_filenames = glob.glob('/content/drive/MyDrive/CUHK/Training sketch/*')

cnt = 0
for i in range(len(p_filenames)):
    im = cv2.imread(p_filenames[i])
    sk = cv2.imread(s_filenames[i])

    for j in range(200):
        img, skt = transform_image(im, sk, 40, 10, 10)

        cv2.imwrite(photo_dir + str(cnt) + ".jpg", img)
        cv2.imwrite(sketch_dir + str(cnt) + ".jpg", skt)

        cnt += 1
```

Helper Functions

```
def load_filename(path):
    dirFiles = os.listdir(path)
    for i, file in enumerate(dirFiles):
        dirFiles[i] = path + file
    return natsort.natsorted(dirFiles ,reverse=False)

# load all images in a directory into memory
def load_images(list_path, size=(256, 256)):
    img_list = list()
    # enumerate filenames in directory, assume all are images
    for filename in list_path:
        # load and resize the image
        pixels = load_img(filename, target_size=size)
        # convert to numpy array
```

```
            pixels = img_to_array(pixels)
            pixels = (pixels - 127.5) / 127.5
            img_list.append(pixels)
        return np.asarray(img_list)


    # select a batch of random samples, returns images and target
    def generate_real_samples(dataset, n_samples, patch_shape):
        # unpack dataset
        trainA, trainB = dataset

        # choose random instances
        ix = np.random.randint(0, trainA.shape[0], n_samples)

        # retrieve selected images
        X1, X2 = trainA[ix], trainB[ix]

        # generate 'real' class labels (1)
        y = np.ones((n_samples, patch_shape, patch_shape, 1))

        return [X1, X2], y

    # generate a batch of images, returns images and targets
    def generate_fake_samples(g_model, samples, patch_shape):
        # generate fake instance
        X = g_model.predict(samples)

        # create 'fake' class labels (0)
        y = np.zeros((len(X), patch_shape, patch_shape, 1))

        return X, y


    # generate samples and save as a plot and save the model
    def summarize_performance(step, g_model, d_model, dataset, target_dir='', n_samples=3):
        if target_dir and not os.path.exists(target_dir):
            os.mkdir(target_dir)
        # select a sample of input images
        [X_realA, X_realB], _ = generate_real_samples(dataset, n_samples, 1)
        # generate a batch of fake samples
        X_fakeB, _ = generate_fake_samples(g_model, X_realA, 1)
        # scale all pixels from [-1,1] to [0,1]
        X_realA = (X_realA + 1) / 2.0
        X_realB = (X_realB + 1) / 2.0
        X_fakeB = (X_fakeB + 1) / 2.0
        # plot real source images
        for i in range(n_samples):
            plt.subplot(3, n_samples, 1 + i)
            plt.axis('off')
            plt.imshow(X_realA[i])
        # plot generated target image
        for i in range(n_samples):
            plt.subplot(3, n_samples, 1 + n_samples + i)
            plt.axis('off')
            plt.imshow(X_fakeB[i])
```

```
        # plot real target image
        for i in range(n_samples):
            plt.subplot(3, n_samples, 1 + n_samples*2 + i)
            plt.axis('off')
            plt.imshow(X_realB[i])
        # save plot to file
        filename1 = 'plot_%06d.png' % (step+1)
        plt.savefig(target_dir + filename1)
        plt.close()
        # save the generator model
        g_model.save(target_dir + 'g_model.h5')

        # save the discriminator model
        d_model.save(target_dir + 'd_model.h5')

        print('>Saved: %s and %s' % (filename1, 'g_model & d_model'))
```

Generator & Discriminator

```
def generator(img_shape):
    def conv2d(layer_in, n_filter, norm=True):
        d = Conv2D(n_filter, kernel_size=4, strides=2, padding='same')(layer_in)
        d = LeakyReLU(0.2)(d)
        if norm:
            d = InstanceNormalization()(d)
        return d

    def deconv2d(layer_in, skip_in, n_filter, dropout=0.5):
        d = UpSampling2D(size=2)(layer_in)
        d = Conv2D(n_filter, kernel_size=4, strides=1, padding='same', activation='relu')(
        if dropout:
            d = Dropout(dropout)(d)
        d = InstanceNormalization()(d)
        d = Concatenate()([d, skip_in])
        return d

    # Input Layer
    in_img = Input(shape=img_shape)

    # Downsampling
    d1 = conv2d(in_img, 64, norm=False)
    d2 = conv2d(d1, 128)
    d3 = conv2d(d2, 256)
    d4 = conv2d(d3, 512)
    d5 = conv2d(d4, 512)
    d6 = conv2d(d5, 512)
    d7 = conv2d(d6, 512)

    # Upsampling
    u1 = deconv2d(d7, d6, 512)
    u2 = deconv2d(u1, d5, 512)
    u3 = deconv2d(u2, d4, 512)
    u4 = deconv2d(u3, d3, 256, dropout=0)
```

```
    u5 = deconv2d(u4, d2, 128, dropout=0)
    u6 = deconv2d(u5, d1, 64, dropout=0)
    u7 = UpSampling2D(size=2)(u6)

    out_img = Conv2D(3, kernel_size=4, strides=1, padding='same', activation='tanh')(u7)

    return Model(in_img, out_img, name='generator')


def discriminator(img_shape):
    def d_layer(layer_in, n_filter, norm=True):
        d = Conv2D(n_filter, kernel_size=4, strides=2, padding='same')(layer_in)
        d = LeakyReLU(0.2)(d)
        if norm:
            d = InstanceNormalization()(d)
        return d

    in_src_img = Input(shape=img_shape)
    in_target_img = Input(shape=img_shape)

    merged = Concatenate()([in_src_img, in_target_img])

    d1 = d_layer(merged, 64, norm=False)
    d2 = d_layer(d1, 128)
    d3 = d_layer(d1, 256)
    d4 = d_layer(d1, 512)

    out = Conv2D(1, kernel_size=4, strides=1, padding='same')(d4)

    return Model([in_src_img, in_target_img], out, name='discriminator')
```

GAN Model

```
def GAN(g_model, d_model, img_shape):
    d_model.trainable = False
    in_img = Input(shape=img_shape)
    gen_out = g_model(in_img)
    dis_out = d_model([in_img, gen_out])
    model = Model(in_img, [dis_out, gen_out], name='GAN')
    return model
```

Training Model

```
def train(d_model, g_model, gan_model, data, target_dir, n_epochs=100, n_batch=16):
    # determine the output square shape of the discriminator
    n_patch = d_model.output_shape[1]

    blue_photo = data[0]
    blue_sketch = data[1]

    for i in range(n_epochs):
```

```
        print(' ========== Epoch', i+1, '========== ')

        blue_photo, blue_sketch = shuffle(blue_photo, blue_sketch)
        # print(len(blue_photo)/n_batch)
        for j in range(int(len(blue_photo)/n_batch)):

            start = int(j*n_batch)
            end = int(min(len(blue_photo), (j*n_batch)+n_batch))

            dataset = [load_images(blue_photo[start:end]), load_images(blue_sketch[start:e

            # select a batch of real samples
            [X_realA, X_realB], y_real = generate_real_samples(dataset, n_batch, n_patch)

            # generate a batch of fake samples
            X_fakeB, y_fake = generate_fake_samples(g_model, X_realA, n_patch)

            # update discriminator for real samples
            d_loss1 = d_model.train_on_batch([X_realA, X_realB], y_real)

            # update discriminator for generated samples
            d_loss2 = d_model.train_on_batch([X_realA, X_fakeB], y_fake)

            d_loss = 0.5 * np.add(d_loss1, d_loss2)

            # update the generator
            g_loss, _, _ = gan_model.train_on_batch(X_realA, [y_real, X_realB])

            # summarize performance
            print('Batch : %d, D Loss : %.3f | G Loss : %.3f' % (j+1, d_loss, g_loss))

        # summarize model performance
#         if (i+1) % 10 == 0:
        summarize_performance(i, g_model, d_model, dataset, target_dir)
```

Loss Functions

```
import tensorflow as tf
import keras.backend as K
from keras.losses import mean_absolute_error

def pixel_loss(y_true, y_pred):
    return K.mean(K.abs(y_true - y_pred))

def contextual_loss (y_true, y_pred):
    a = tf.image.rgb_to_grayscale(tf.slice(
                                y_pred,
                                [0,0,0,0],
                                [16, 256, 256, 3]))

    b = tf.image.rgb_to_grayscale(tf.slice(
                                y_true,
                                [0,0,0,0],
```

```
                                    [16, 256, 256, 3]))

    y_pred = tf.divide(tf.add(tf.reshape(a, [tf.shape(a)[0], -1]), 1), 2)
    y_true = tf.divide(tf.add(tf.reshape(b, [tf.shape(b)[0], -1]), 1), 2)

#     tf.assert_rank(y_true,2)
#     tf.assert_rank(y_pred,2)

    p_shape = tf.shape(y_true)
    q_shape = tf.shape(y_pred)
#     tf.assert_equal(p_shape, q_shape)

    # normalize sum to 1
    p_ = tf.divide(y_true, tf.tile(tf.expand_dims(tf.reduce_sum(y_true, axis=1), 1), [1,p_
    q_ = tf.divide(y_pred, tf.tile(tf.expand_dims(tf.reduce_sum(y_pred, axis=1), 1), [1,p_

    return tf.reduce_sum(tf.multiply(p_, tf.math.log(tf.divide(p_, q_))), axis=1)

def total_loss (y_true, y_pred):

    px_loss = pixel_loss(y_true, y_pred)

    ctx_loss = contextual_loss(y_true, y_pred)

    return (0.2 * px_loss) + (0.8 * ctx_loss)
```

Model Run

```
img_shape = (256, 256, 3)

d_model = discriminator(img_shape)

g_model = generator(img_shape)

gan_model = GAN(g_model, d_model, img_shape)


gan_model.summary()
```

```
Model: "GAN"
_________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================
input_4 (InputLayer)         [(None, 256, 256, 3) 0
_________________________________________________________________________________
generator (Functional)       (None, 256, 256, 3)  41825691    input_4[0][0]
_________________________________________________________________________________
discriminator (Functional)   (None, 64, 64, 1)    539203      input_4[0][0]
                                                               generator[0][0]
=================================================================================
Total params: 42,364,894
Trainable params: 41,825,691
Non-trainable params: 539,203
_________________________________________________________________________________
```

```
opt = Adam(lr=2e-4, beta_1=0.5)
# opt = Adam(lr=2e-4, beta_1=0.7)
# opt = Adam(lr=2e-5, beta_1=0.9)
# opt = Adam(lr=0.01, beta_1=0.9)

d_model.compile(loss='binary_crossentropy', optimizer=opt, loss_weights=[0.3])
g_model.compile(loss='binary_crossentropy', optimizer=opt, loss_weights=[0.2])
gan_model.compile(loss=['binary_crossentropy', total_loss], optimizer=opt, loss_weights=[1


train(d_model, g_model, gan_model, [blue_sketch, blue_photo], '/content/drive/MyDrive/CUHK
```

```
 ========== Epoch 1 ==========
Batch : 1, D Loss : 2.236 | G Loss : 10.307
Batch : 2, D Loss : 2.214 | G Loss : 15.780
Batch : 3, D Loss : 2.165 | G Loss : 11.200
Batch : 4, D Loss : 2.202 | G Loss : 9.656
Batch : 5, D Loss : 2.213 | G Loss : 11.215
>Saved: plot_000001.png and g_model & d_model
 ========== Epoch 2 ==========
Batch : 1, D Loss : 2.228 | G Loss : 10.702
Batch : 2, D Loss : 2.231 | G Loss : 11.097
Batch : 3, D Loss : 2.185 | G Loss : 9.909
Batch : 4, D Loss : 2.203 | G Loss : 9.909
Batch : 5, D Loss : 2.211 | G Loss : 10.261
>Saved: plot_000002.png and g_model & d_model
 ========== Epoch 3 ==========
Batch : 1, D Loss : 2.235 | G Loss : 9.895
Batch : 2, D Loss : 2.196 | G Loss : 9.742
Batch : 3, D Loss : 2.188 | G Loss : 9.667
Batch : 4, D Loss : 2.219 | G Loss : 9.786
Batch : 5, D Loss : 2.207 | G Loss : 9.614
>Saved: plot_000003.png and g_model & d_model
 ========== Epoch 4 ==========
Batch : 1, D Loss : 2.202 | G Loss : 9.372
Batch : 2, D Loss : 2.198 | G Loss : 10.378
Batch : 3, D Loss : 2.213 | G Loss : 10.595
Batch : 4, D Loss : 2.206 | G Loss : 10.617
Batch : 5, D Loss : 2.228 | G Loss : 9.677
>Saved: plot_000004.png and g_model & d_model
 ========== Epoch 5 ==========
Batch : 1, D Loss : 2.196 | G Loss : 10.633
Batch : 2, D Loss : 2.224 | G Loss : 10.611
Batch : 3, D Loss : 2.198 | G Loss : 9.632
Batch : 4, D Loss : 2.186 | G Loss : 9.841
Batch : 5, D Loss : 2.209 | G Loss : 9.190
>Saved: plot_000005.png and g_model & d_model
```

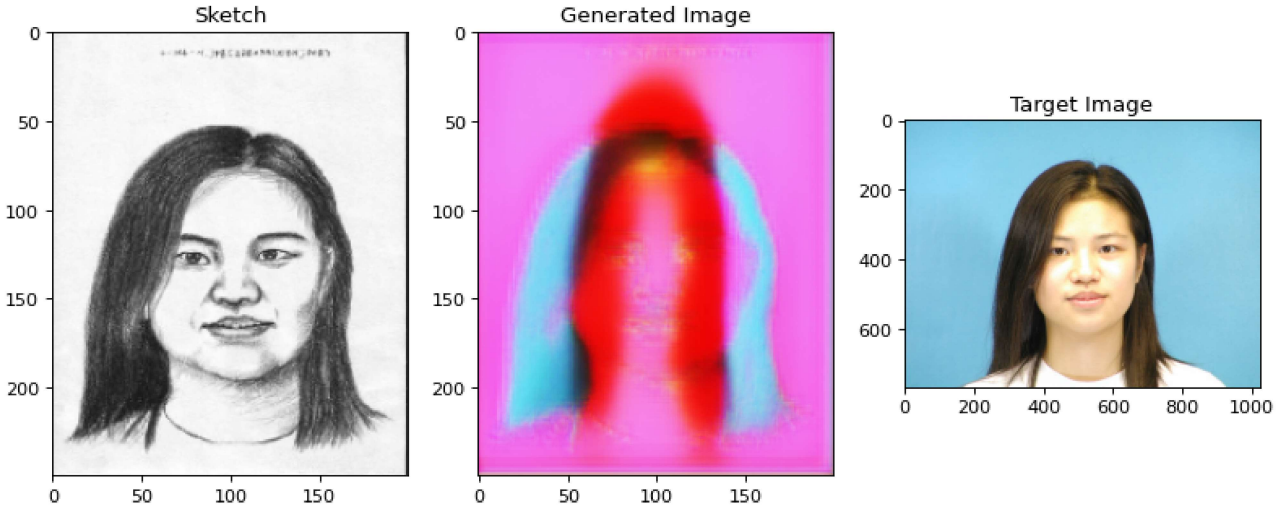
```
def load_filename(path):
    dirFiles = os.listdir(path)
    for i, file in enumerate(dirFiles):
        dirFiles[i] = path + file
    return natsort.natsorted(dirFiles ,reverse=False)

# load all images in a directory into memory
def load_images(list_path, size=(256, 256)):
```

```
        img_list = list()
        # enumerate filenames in directory, assume all are images
        for filename in list_path:
            # load and resize the image
            pixels = load_img(filename, target_size=size)
            # convert to numpy array
            pixels = img_to_array(pixels)
            pixels = (pixels - 127.5) / 127.5
            img_list.append(pixels)
        return np.asarray(img_list)

    def pred_images(g_model, target_dir, filenames, batch_size=128):
        if not os.path.exists(target_dir):
            os.mkdir(target_dir)

        imgs = load_images(filenames)
        g_img = g_model.predict(imgs)
        g_img = g_img * 127.5 + 127.5
        for j, _img in enumerate(g_img):
            cv2.imwrite(target_dir + "/" + os.path.basename(filenames[j]), cv2.resize(cv2.cvtC
        print("Image has been successfully saved in \"" + target_dir + "\" folder")


    filenames = load_filename('/content/drive/MyDrive/CUHK/Testing sketch/')
```

Results for different Hyper-parameters

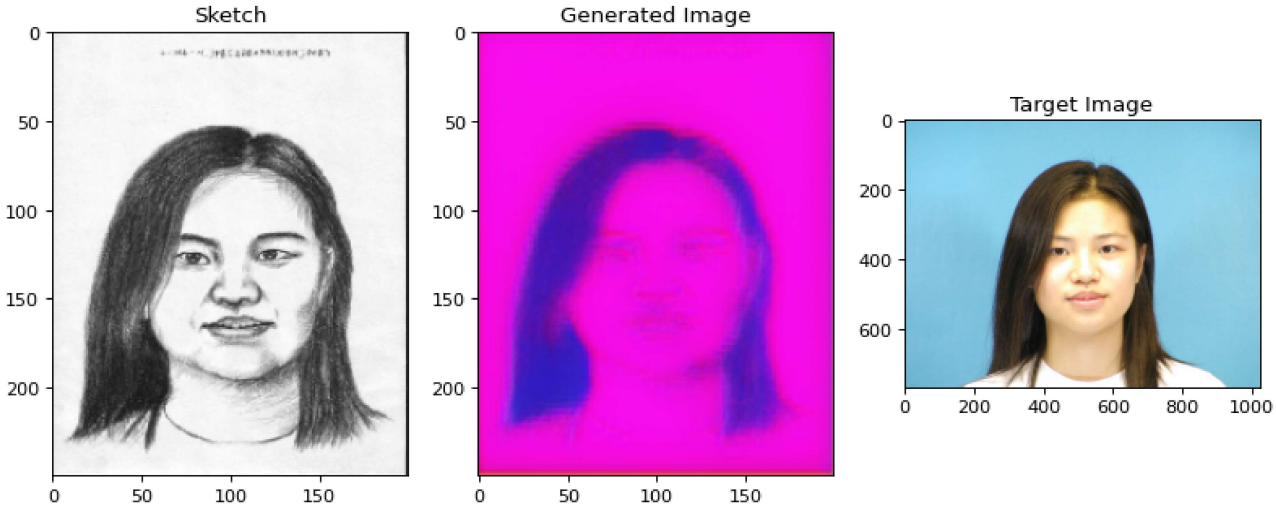
```
 Load Model
_model = load_model('/content/drive/MyDrive/CUHK/Dataset/Models/XI/g_model.h5',custom_obje

 load and resize the image
mg = load_img('/content/drive/MyDrive/CUHK/Testing sketch/f1-004-01-sz1.jpg', target_size=
arget = cv2.cvtColor(cv2.imread('/content/drive/MyDrive/CUHK/Testing photo/f1-004-01.jpg')

 convert to numpy array
mg = img_to_array(img)
orm_img = (img.copy() - 127.5) / 127.5

_img = g_model.predict(np.expand_dims(norm_img, 0))[0]
_img = g_img * 127.5 + 127.5

mg = cv2.resize(img, (200, 250))
_img = cv2.resize(g_img, (200, 250))

 = plt.figure(num=None, figsize=(12, 6), dpi=80)
x1 = f.add_subplot(1,3, 1)
lt.imshow(img.astype('uint8'))
x2 = f.add_subplot(1,3, 2)
lt.imshow(g_img.astype('uint8'))
x3 = f.add_subplot(1,3, 3)
lt.imshow(target.astype('uint8'))
x1.set_title('Sketch')
```

```
x2.set_title('Generated Image')
x3.set_title('Target Image')

lt.show(block=True)
```



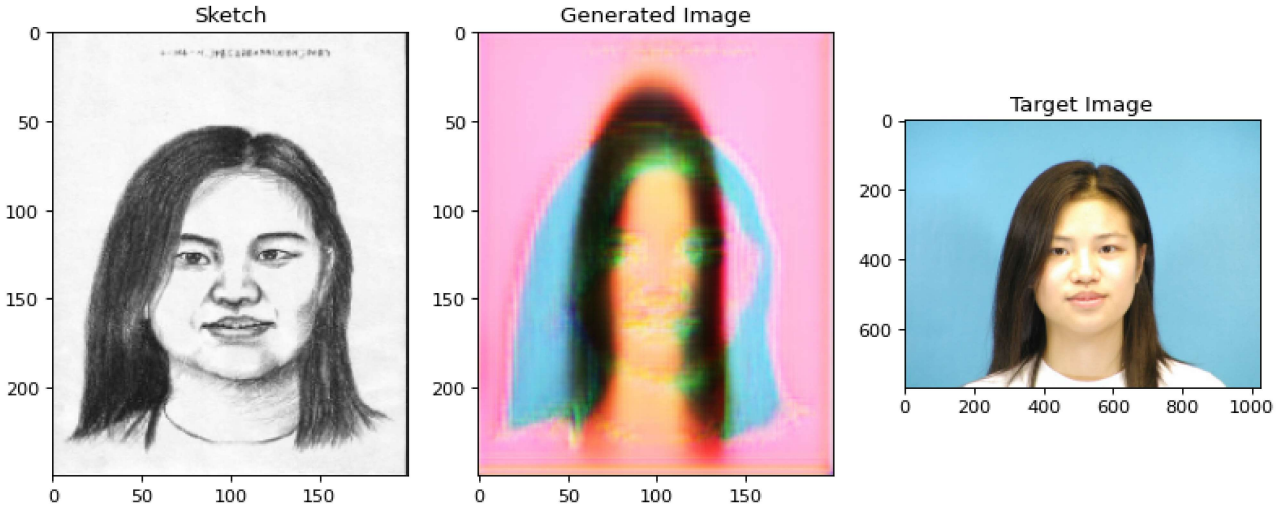

```
# Load Model
g_model = load_model('/content/drive/MyDrive/CUHK/Dataset/Models/Aug_I/g_model.h5',custom_

# load and resize the image
img = load_img('/content/drive/MyDrive/CUHK/Testing sketch/f1-004-01-sz1.jpg', target_size
target = cv2.cvtColor(cv2.imread('/content/drive/MyDrive/CUHK/Testing photo/f1-004-01.jpg'

# convert to numpy array
img = img_to_array(img)
norm_img = (img.copy() - 127.5) / 127.5

g_img = g_model.predict(np.expand_dims(norm_img, 0))[0]
g_img = g_img * 127.5 + 127.5

img = cv2.resize(img, (200, 250))
g_img = cv2.resize(g_img, (200, 250))

f = plt.figure(num=None, figsize=(12, 6), dpi=80)
ax1 = f.add_subplot(1,3, 1)
plt.imshow(img.astype('uint8'))
ax2 = f.add_subplot(1,3, 2)
plt.imshow(g_img.astype('uint8'))
ax3 = f.add_subplot(1,3, 3)
plt.imshow(target.astype('uint8'))
ax1.set_title('Sketch')
ax2.set_title('Generated Image')
ax3.set_title('Target Image')

plt.show(block=True)
```

```
WARNING:tensorflow:11 out of the last 11 calls to <function Model.make_predict_funct
```



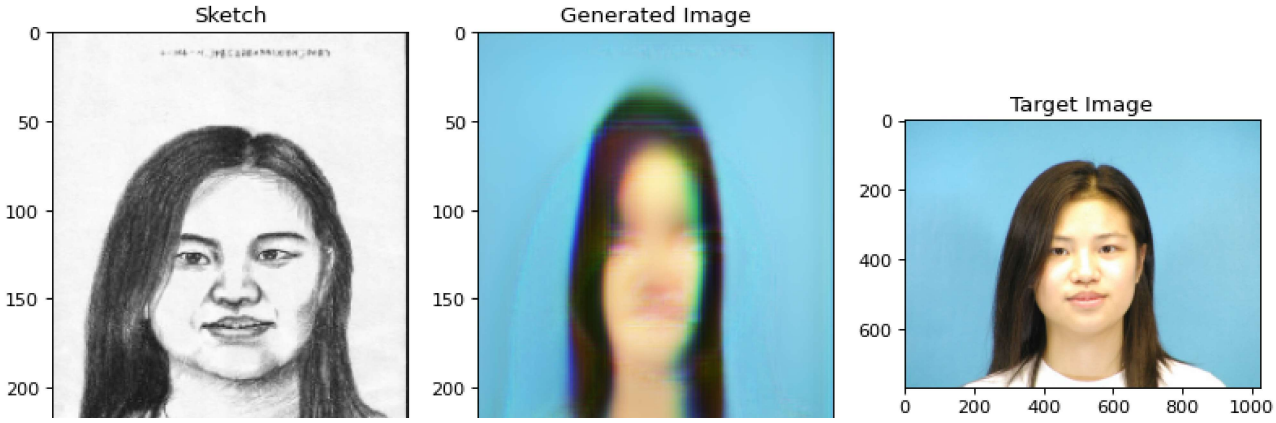

```
# Load Model
g_model = load_model('/content/drive/MyDrive/CUHK/Dataset/Models/I/g_model.h5',custom_obje

# load and resize the image
img = load_img('/content/drive/MyDrive/CUHK/Testing sketch/f1-004-01-sz1.jpg', target_size
target = cv2.cvtColor(cv2.imread('/content/drive/MyDrive/CUHK/Testing photo/f1-004-01.jpg'

# convert to numpy array
img = img_to_array(img)
norm_img = (img.copy() - 127.5) / 127.5

g_img = g_model.predict(np.expand_dims(norm_img, 0))[0]
g_img = g_img * 127.5 + 127.5

img = cv2.resize(img, (200, 250))
g_img = cv2.resize(g_img, (200, 250))

f = plt.figure(num=None, figsize=(12, 6), dpi=80)
ax1 = f.add_subplot(1,3, 1)
plt.imshow(img.astype('uint8'))
ax2 = f.add_subplot(1,3, 2)
plt.imshow(g_img.astype('uint8'))
ax3 = f.add_subplot(1,3, 3)
plt.imshow(target.astype('uint8'))
ax1.set_title('Sketch')
ax2.set_title('Generated Image')
ax3.set_title('Target Image')

plt.show(block=True)
```

```
WARNING:tensorflow:11 out of the last 11 calls to <function Model.make_predict_funct
```



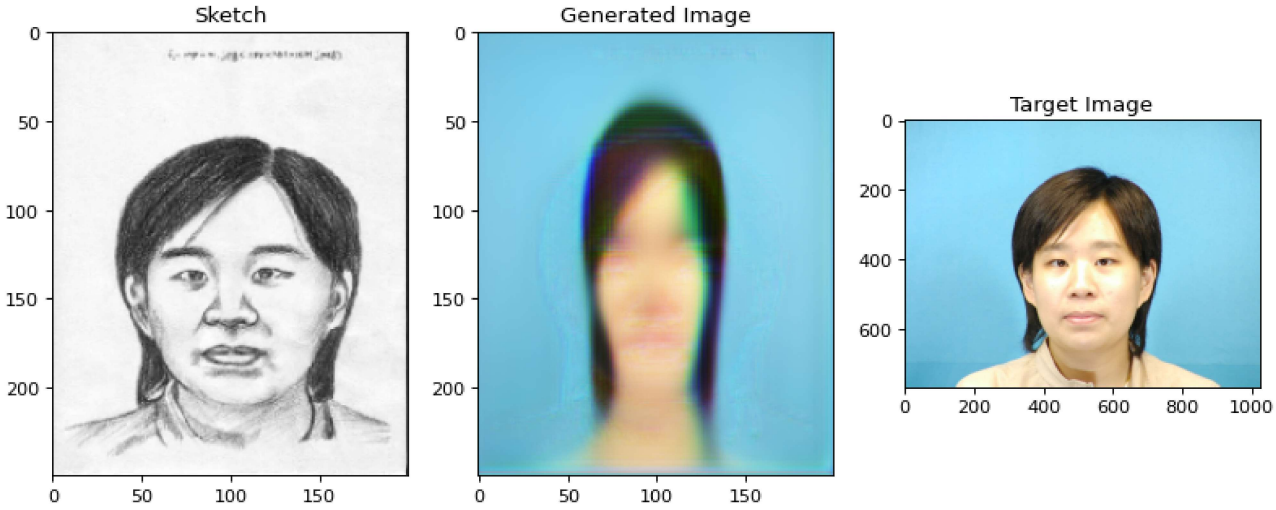

```
# Load Model
g_model = load_model('/content/drive/MyDrive/CUHK/Dataset/Models/VII/g_model.h5',custom_ob

# load and resize the image
img = load_img('/content/drive/MyDrive/CUHK/Testing sketch/f1-004-01-sz1.jpg', target_size
target = cv2.cvtColor(cv2.imread('/content/drive/MyDrive/CUHK/Testing photo/f1-004-01.jpg'

# convert to numpy array
img = img_to_array(img)
norm_img = (img.copy() - 127.5) / 127.5

g_img = g_model.predict(np.expand_dims(norm_img, 0))[0]
g_img = g_img * 127.5 + 127.5

img = cv2.resize(img, (200, 250))
g_img = cv2.resize(g_img, (200, 250))

f = plt.figure(num=None, figsize=(12, 6), dpi=80)
ax1 = f.add_subplot(1,3, 1)
plt.imshow(img.astype('uint8'))
ax2 = f.add_subplot(1,3, 2)
plt.imshow(g_img.astype('uint8'))
ax3 = f.add_subplot(1,3, 3)
plt.imshow(target.astype('uint8'))
ax1.set_title('Sketch')
ax2.set_title('Generated Image')
ax3.set_title('Target Image')

plt.show(block=True)
```

```
WARNING:tensorflow:11 out of the last 11 calls to <function Model.make_predict_funct
```




```
# Load Model
g_model = load_model('/content/drive/MyDrive/CUHK/Dataset/Models/Better/g_model.h5',custom

# load and resize the image
img = load_img('/content/drive/MyDrive/CUHK/Testing sketch/f1-009-01-sz1.jpg', target_size
target = cv2.cvtColor(cv2.imread('/content/drive/MyDrive/CUHK/Testing photo/f1-009-01.jpg'

# convert to numpy array
img = img_to_array(img)
norm_img = (img.copy() - 127.5) / 127.5

g_img = g_model.predict(np.expand_dims(norm_img, 0))[0]
g_img = g_img * 127.5 + 127.5

img = cv2.resize(img, (200, 250))
g_img = cv2.resize(g_img, (200, 250))

f = plt.figure(num=None, figsize=(12, 6), dpi=80)
ax1 = f.add_subplot(1,3, 1)
plt.imshow(img.astype('uint8'))
ax2 = f.add_subplot(1,3, 2)
plt.imshow(g_img.astype('uint8'))
ax3 = f.add_subplot(1,3, 3)
plt.imshow(target.astype('uint8'))
ax1.set_title('Sketch')
ax2.set_title('Generated Image')
ax3.set_title('Target Image')

plt.show(block=True)
```

```
WARNING:tensorflow:10 out of the last 39 calls to <function Model.make_predict_funct
```



✓ 4s completed at 11:42 AM