In [3]:
```
import os
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
from tqdm import tqdm
from numpy import expand_dims
from numpy import zeros
from numpy import ones
from numpy import vstack
from numpy.random import randn
from numpy.random import randint
from keras.datasets.cifar10 import load_data
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import ZeroPadding2D
from keras.layers import BatchNormalization
from keras.layers import Reshape
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import Activation
from keras.layers import UpSampling2D
from keras.layers import Conv2DTranspose
from keras.layers import LeakyReLU
from keras.layers import Dropout
from matplotlib import pyplot
```

Get the Training Data

The data is from the [faces_data_new (https://www.kaggle.com/gasgallo/faces-data-new)](https://www.kaggle.com/gasgallo/faces-data-new) and [lag-dataset (https://www.kaggle.com/gasgallo/lag-dataset)](https://www.kaggle.com/gasgallo/lag-dataset) from Kaggle.

The dataset consists of over 10,000 face pictures

Here we resize the pictures to 64x64 and convet the images to a numpy array

In [4]:
```
def get_training_data(datafolder):
        print("Loading training data...")

        training_data = []
        #Get all the pictures from the data folder except the .DS_Stor
e from MAC OS
        filenames = os.listdir(datafolder)
        for filename in tqdm(filenames):
            if filename == '.DS_Store':
                continue
            #Combines folder name and file name.
            else:
                path = os.path.join(datafolder,filename)
                #Opens an image as an Image object.
                image = Image.open(path)
                #Resizes to a desired size.
                image = image.resize((64,64),Image.ANTIALIAS)
                #Creates an array of pixel values from the image.
                pixel_array = np.asarray(image)

                training_data.append(pixel_array)

        #converted to a numpy array
        training_data = np.reshape(training_data,(-1,64,64,3))
        return training_data
```

Define the Discriminator

The discriminator is bulit in the following a Neural Network:

In the network, we use the LeakyReLU as the activation layer inside the network since it is proved that it works well in GANs

Fron each layer, we also have a Dropout Layer and a BatchNormalization Layer

In classfication layer we have a sigmoid activation which produces a probability that the picture is true

Adam adaptive learning rate optimization algorithm and binary_crossentropy are also used here since it is recommended in the GANs

In [5]:
```
def define_discriminator(in_shape=(64,64,3)):
        model = Sequential()

        model.add(Conv2D(32, kernel_size=3, strides=2, input_shape=in_
shape, padding="same"))
        model.add(LeakyReLU(alpha=0.2))

        # downsample
        model.add(Dropout(0.25))
        model.add(Conv2D(64, kernel_size=3, strides=2, padding="same")
)
        model.add(ZeroPadding2D(padding=((0,1),(0,1))))
        model.add(BatchNormalization(momentum=0.8))
        model.add(LeakyReLU(alpha=0.2))

        # downsample
        model.add(Dropout(0.25))
        model.add(Conv2D(128, kernel_size=3, strides=2, padding="same"
))
        model.add(BatchNormalization(momentum=0.8))
        model.add(LeakyReLU(alpha=0.2))

        # downsample
        model.add(Dropout(0.25))
        model.add(Conv2D(256, kernel_size=3, strides=1, padding="same"
))
        model.add(BatchNormalization(momentum=0.8))
        model.add(LeakyReLU(alpha=0.2))

        # downsample
        model.add(Dropout(0.25))
        model.add(Conv2D(512, kernel_size=3, strides=1, padding="same"
))
        model.add(BatchNormalization(momentum=0.8))
        model.add(LeakyReLU(alpha=0.2))

        model.add(Dropout(0.25))

        # classifier
        model.add(Flatten())
        #model.add(Dropout(0.4))
        model.add(Dense(1, activation='sigmoid'))
        # compile model
        opt = Adam(lr=0.0002, beta_1=0.5)
        model.compile(loss='binary_crossentropy', optimizer=opt, metri
cs=['accuracy'])
        return model
```

Define the Generator

The input is a vecotr of length latent_dim and the Generator outputs image of size 64*64*3

relu activation is used and it also shared similar structure as the discriminator

In [6]:
```
def define_generator(latent_dim):
        model = Sequential()
        # foundation for 4x4 image
        n_nodes = 256 * 4 * 4
        model.add(Dense(n_nodes, activation="relu",input_dim=latent_di
m))

        model.add(Reshape((4, 4, 256)))
        # upsample to 8x8
        model.add(UpSampling2D())
        model.add(Conv2D(256,kernel_size=3,padding="same"))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Activation("relu"))

        # upsample to 16x16
        model.add(UpSampling2D())
        model.add(Conv2D(256,kernel_size=3,padding="same"))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Activation("relu"))

        # upsample to 32x32
        model.add(UpSampling2D())
        model.add(Conv2D(128,kernel_size=3,padding="same"))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Activation("relu"))

    # upsample to 64x64
        model.add(UpSampling2D())
        model.add(Conv2D(128,kernel_size=3,padding="same"))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Activation("relu"))

        # output layer
        model.add(Conv2D(3,kernel_size=3,padding="same"))
        model.add(Activation("tanh"))

        return model
```

Combine the generator and discriminator model

In [7]:
```
def define_gan(g_model, d_model):
        # make weights in the discriminator not trainable
        d_model.trainable = False
        # connect them
        model = Sequential()
        # add generator
        model.add(g_model)
        # add the discriminator
        model.add(d_model)
        # compile model
        opt = Adam(lr=0.0002, beta_1=0.5)
        model.compile(loss='binary_crossentropy', optimizer=opt)
        return model
```

Utility functions

Some other functions that helps to load and select some real images; generate fake images and save the plots that the generartor generates during the training process and print the performances during traning.

In [8]:
```
def load_real_samples():
        # load cifar10 dataset
        (trainX, _), (_, _) = load_data()
        # convert from unsigned ints to floats
        X = trainX.astype('float32')
        # scale from [0,255] to [-1,1]
        X = (X - 127.5) / 127.5
        return X

# select real samples
def generate_real_samples(dataset, n_samples):
        # choose random instances
        ix = randint(0, dataset.shape[0], n_samples)
        # retrieve selected images
        X = dataset[ix]
        # generate 'real' class labels (1)
        y = ones((n_samples, 1))
        return X, y

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
```

```
        # generate points in the latent space
        x_input = randn(latent_dim * n_samples)
        # reshape into a batch of inputs for the network
        x_input = x_input.reshape(n_samples, latent_dim)
        return x_input

# use the generator to generate n fake examples, with class labels
def generate_fake_samples(g_model, latent_dim, n_samples):
        # generate points in latent space
        x_input = generate_latent_points(latent_dim, n_samples)
        # predict outputs
        X = g_model.predict(x_input)
        # create 'fake' class labels (0)
        y = zeros((n_samples, 1))
        return X, y

# create and save a plot of generated images
def save_plot(examples, epoch, n=7):
        # scale from [-1,1] to [0,1]
        examples = (examples + 1) / 2.0
        # plot images
        for i in range(n * n):
                # define subplot
                pyplot.subplot(n, n, 1 + i)
                # turn off axis
                pyplot.axis('off')
                # plot raw pixel data
                pyplot.imshow(examples[i])
        # save plot to file
        filename = 'generated_plot_e%03d.png' % (epoch+1)
        pyplot.savefig(filename)
        pyplot.close()

# evaluate the discriminator, plot generated images, save generator mo
del
def summarize_performance(epoch, g_model, d_model, dataset, latent_dim
, n_samples=150):
        # prepare real samples
        X_real, y_real = generate_real_samples(dataset, n_samples)
        # evaluate discriminator on real examples
        _, acc_real = d_model.evaluate(X_real, y_real, verbose=0)
        # prepare fake examples
        x_fake, y_fake = generate_fake_samples(g_model, latent_dim, n_
samples)
        # evaluate discriminator on fake examples
        _, acc_fake = d_model.evaluate(x_fake, y_fake, verbose=0)
        # summarize discriminator performance
        print('>Accuracy real: %.0f%%, fake: %.0f%%' % (acc_real*100,
acc_fake*100))
        # save plot
```

```
        save_plot(x_fake, epoch)
        # save the generator model tile file
        filename = 'generator_model_%03d.h5' % (epoch+1)
        g_model.save(filename)


from keras.models import load_model
from numpy.random import randn
from matplotlib import pyplot

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
        # generate points in the latent space
        x_input = randn(latent_dim * n_samples)
        # reshape into a batch of inputs for the network
        x_input = x_input.reshape(n_samples, latent_dim)
        return x_input

# plot the generated images
def create_plot(examples, n):
        # plot images
        for i in range(n * n):
                # define subplot
                pyplot.subplot(n, n, 1 + i)
                # turn off axis
                pyplot.axis('off')
                # plot raw pixel data
                pyplot.imshow(examples[i, :, :])
        pyplot.show()
```

Training function:

Step 1: for each epoch we have half of the pictures from the real pictures with label: 1(true) and half of them from the fake pictures with label: 1(true) generated from the generator

Step 2: we use them as the traning set form discriminator to tell if they can tell the fake pictures.

Step 3: The error of discriminator is then sent back to generator and help generator generates better pictures to trick the discriminator.

In [9]:
```
def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=4
000, n_batch=32):
        bat_per_epo = int(dataset.shape[0] / n_batch)
        half_batch = int(n_batch / 2)
        # manually enumerate epochs
        for i in range(n_epochs):
                # enumerate batches over the training set
                for j in range(bat_per_epo):
                        # get randomly selected 'real' samples
                        X_real, y_real = generate_real_samples(dataset
, half_batch)
                        # update discriminator model weights
                        d_loss1, _ = d_model.train_on_batch(X_real, y_
real)
                        # generate 'fake' examples
                        X_fake, y_fake = generate_fake_samples(g_model
, latent_dim, half_batch)
                        # update discriminator model weights
                        d_loss2, _ = d_model.train_on_batch(X_fake, y_
fake)
                        # prepare points in latent space as input for
the generator
                        X_gan = generate_latent_points(latent_dim, n_b
atch)
                        # create inverted labels for the fake samples
                        y_gan = ones((n_batch, 1))
                        # update the generator via the discriminator's
error
                        g_loss = gan_model.train_on_batch(X_gan, y_gan
)
                        # summarize loss on this batch
                        print('>%d, %d/%d, d1=%.3f, d2=%.3f g=%.3f' %
                                (i+1, j+1, bat_per_epo, d_loss1, d_los
s2, g_loss))
                # evaluate the model performance, sometimes
                if (i+1) % 10 == 0:
                        summarize_performance(i, g_model, d_model, dat
aset, latent_dim)
```

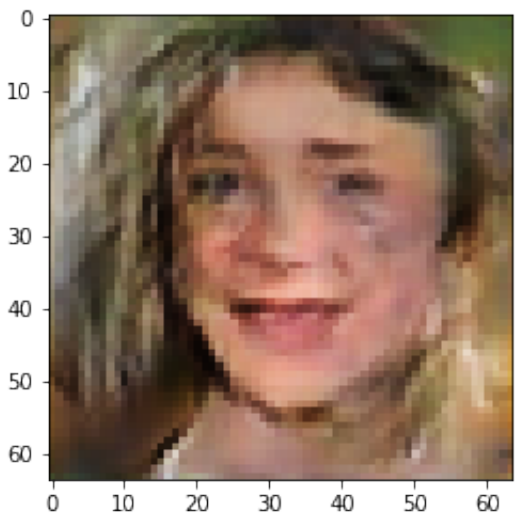
Training Process (Takes a few days to finish the first 300 epoches...)

In [10]:
```
## The data is store in the data/images folder. But it is so large tha
t we can not upload the folder to github,
## please download the data from the links provided in the first secti
on and put them into the correct position.

data_set = get_training_data("..data/images")
data_set = data_set / 127.5 - 1.
# size of the latent space
latent_dim = 100
# create the discriminator
d_model = define_discriminator()
# create the generator
g_model = define_generator(latent_dim)
# create the gan
gan_model = define_gan(g_model, d_model)
# load image data
dataset = load_real_samples()
# train model
train(g_model, d_model, gan_model, data_set, latent_dim)
```

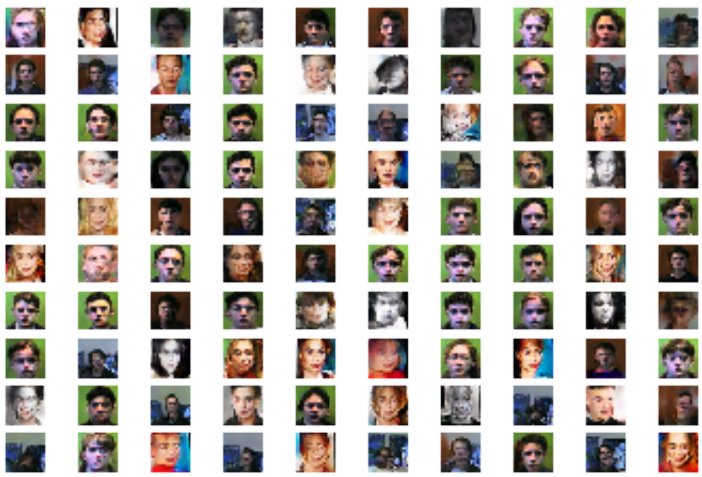
```
  0%|          | 24/11683 [00:00<00:49, 237.26it/s]

Loading training data...

100%|██████████| 11683/11683 [00:34<00:00, 336.86it/s]
```

Load the models that we trained using different numbers of epoches and plot the fake images that the generator creates

In [59]:
```
from keras.models import load_model
from numpy import asarray
from matplotlib import pyplot
# load model
model = load_model('../output/generator_model_280.h5')
# all 0s
vector = asarray([[0.75 for _ in range(100)]])
# generate image
X = model.predict(vector)
# scale from [-1,1] to [0,1]
X = (X + 1) / 2.0
# plot the result
pyplot.imshow(X[0, :, :])
pyplot.show()
```



In [60]:
```
model = load_model('../output/generator_model_200.h5')
# generate images
latent_points = generate_latent_points(100, 100)
# Get fake images
X = model.predict(latent_points)
# scale from [-1,1] to [0,1]
X = (X + 1) / 2.0
# plot the result
create_plot(X, 10)
```



CycleGANs

December 5, 2019

0.0.1 Clone the repository where the pre-trained model is stored

[ ]:
```
!git clone --recursive https://github.com/shaoanlu/fewshot-face-translation-GAN.
↪git
```

[3]:
```
# using the required version of keras and tensorflow
!pip install keras==2.2.4
!pip install tensorflow==1.12.0
```

```
Collecting keras==2.2.4
  Downloading https://files.pythonhosted.org/packages/5e/10/aa32dad071ce52
b5502266b5c659451cfd6ffcbf14e6c8c4f16c0ff5aaab/Keras-2.2.4-py2.py3-none-any.whl
(312kB)
     || 317kB 2.8MB/s
Requirement already satisfied: six>=1.9.0 in
/usr/local/lib/python3.6/dist-packages (from keras==2.2.4) (1.12.0)
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-packages
(from keras==2.2.4) (2.8.0)
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.6/dist-
packages (from keras==2.2.4) (1.17.4)
Requirement already satisfied: keras-preprocessing>=1.0.5 in
/usr/local/lib/python3.6/dist-packages (from keras==2.2.4) (1.1.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.6/dist-packages
(from keras==2.2.4) (3.13)
Requirement already satisfied: keras-applications>=1.0.6 in
/usr/local/lib/python3.6/dist-packages (from keras==2.2.4) (1.0.8)
Requirement already satisfied: scipy>=0.14 in /usr/local/lib/python3.6/dist-
packages (from keras==2.2.4) (1.3.2)
Installing collected packages: keras
  Found existing installation: Keras 2.2.5
    Uninstalling Keras-2.2.5:
      Successfully uninstalled Keras-2.2.5
Successfully installed keras-2.2.4
Collecting tensorflow==1.12.0
  Downloading https://files.pythonhosted.org/packages/22/cc/ca70b78087015d
21c5f3f93694107f34ebccb3be9624385a911d4b52ecef/tensorflow-1.12.0-cp36-cp36m-many
linux1_x86_64.whl (83.1MB)
```

```
     || 83.1MB 41kB/s
Requirement already satisfied: wheel>=0.26 in
/usr/local/lib/python3.6/dist-packages (from tensorflow==1.12.0) (0.33.6)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.6/dist-packages (from tensorflow==1.12.0) (1.1.0)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.6/dist-
packages (from tensorflow==1.12.0) (1.17.4)
Requirement already satisfied: protobuf>=3.6.1 in /usr/local/lib/python3.6/dist-
packages (from tensorflow==1.12.0) (3.10.0)
Requirement already satisfied: keras-preprocessing>=1.0.5 in
/usr/local/lib/python3.6/dist-packages (from tensorflow==1.12.0) (1.1.0)
Requirement already satisfied: grpcio>=1.8.6 in /usr/local/lib/python3.6/dist-
packages (from tensorflow==1.12.0) (1.15.0)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-
packages (from tensorflow==1.12.0) (1.12.0)
Requirement already satisfied: gast>=0.2.0 in /usr/local/lib/python3.6/dist-
packages (from tensorflow==1.12.0) (0.2.2)
Requirement already satisfied: keras-applications>=1.0.6 in
/usr/local/lib/python3.6/dist-packages (from tensorflow==1.12.0) (1.0.8)
Requirement already satisfied: astor>=0.6.0 in /usr/local/lib/python3.6/dist-
packages (from tensorflow==1.12.0) (0.8.0)
Requirement already satisfied: absl-py>=0.1.6 in /usr/local/lib/python3.6/dist-
packages (from tensorflow==1.12.0) (0.8.1)
Collecting tensorboard<1.13.0,>=1.12.0
  Downloading https://files.pythonhosted.org/packages/07/53/8d32ce9471c18f
8d99028b7cef2e5b39ea8765bd7ef250ca05b490880971/tensorboard-1.12.2-py3-none-
any.whl (3.0MB)
     || 3.1MB 33.7MB/s
Requirement already satisfied: setuptools in
/usr/local/lib/python3.6/dist-packages (from
protobuf>=3.6.1->tensorflow==1.12.0) (41.6.0)
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-packages
(from keras-applications>=1.0.6->tensorflow==1.12.0) (2.8.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.6/dist-
packages (from tensorboard<1.13.0,>=1.12.0->tensorflow==1.12.0) (3.1.1)
Requirement already satisfied: werkzeug>=0.11.10 in
/usr/local/lib/python3.6/dist-packages (from
tensorboard<1.13.0,>=1.12.0->tensorflow==1.12.0) (0.16.0)
Installing collected packages: tensorboard, tensorflow
  Found existing installation: tensorboard 1.15.0
    Uninstalling tensorboard-1.15.0:
      Successfully uninstalled tensorboard-1.15.0
  Found existing installation: tensorflow 1.15.0
    Uninstalling tensorflow-1.15.0:
      Successfully uninstalled tensorflow-1.15.0
Successfully installed tensorboard-1.12.2 tensorflow-1.12.0
```

[3]:
```
%cd "fewshot-face-translation-GAN"
```

```
/content/fewshot-face-translation-GAN
```

0.0.2 Download the pretrianed model and import it.

[5]:
```
!gdown https://drive.google.com/uc?id=1DUMmZGTGKMyEYSKy-w34IDHawVF24rIs
!gdown https://drive.google.com/uc?id=1xl8cg7xaRnMsyiODcXguJ83d5hwodckB
```

```
Downloading...
From: https://drive.google.com/uc?id=1DUMmZGTGKMyEYSKy-w34IDHawVF24rIs
To: /content/fewshot-face-translation-GAN/encoder.h5
6.26MB [00:00, 19.9MB/s]
Downloading...
From: https://drive.google.com/uc?id=1xl8cg7xaRnMsyiODcXguJ83d5hwodckB
To: /content/fewshot-face-translation-GAN/decoder.h5
124MB [00:02, 48.0MB/s]
```

[0]:
```
!mkdir weights
!mv decoder.h5 weights/decoder.h5
!mv encoder.h5 weights/encoder.h5
```

[0]:
```
import warnings
warnings.filterwarnings("ignore")
```

[5]:
```
from models import FaceTranslationGANInferenceModel
```

```
Using TensorFlow backend.
```

```
<IPython.core.display.HTML object>
```

[15]:
```
model = FaceTranslationGANInferenceModel()
```

```
Found checkpoints in weights folder. Built model with pre-trained weights.
```

[0]:
```
from face_toolbox_keras.models.verifier.face_verifier import FaceVerifier
fv = FaceVerifier(classes=512)

from face_toolbox_keras.models.parser import face_parser
fp = face_parser.FaceParser()

from face_toolbox_keras.models.detector import face_detector
fd = face_detector.FaceAlignmentDetector()

from face_toolbox_keras.models.detector.iris_detector import IrisDetector
idet = IrisDetector()
```

0.0.3 Upload source and target pictures

[0]:
```python
import numpy as np
from utils import utils
from matplotlib import pyplot as plt
```

[0]:
```python
from google.colab import files
```

[207]:
```python
fn_src = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving IMG_1544.JPG to IMG_1544 (4).JPG
```

[208]:
```python
fns_tar = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving IMG_1582.jpg to IMG_1582 (2).jpg
```

[209]:
```python
fn_src = [k for k,v in fn_src.items()]
if len(fn_src) >= 1:
    fn_src = fn_src[0]

fns_tar = [k for k,v in fns_tar.items()]

print(fn_src)
print(fns_tar)
```

```
IMG_1544.JPG
['IMG_1582.jpg']
```

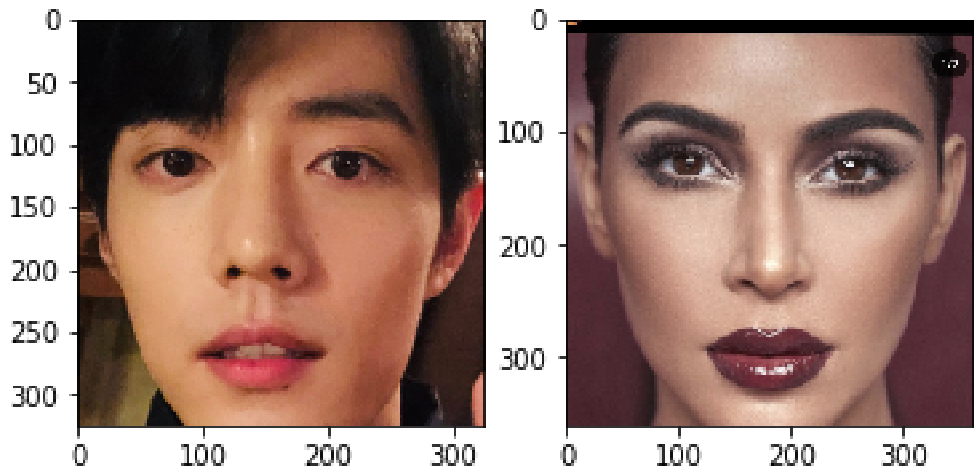
0.0.4 Set variables and make prediction

[0]:
```python
src, mask, aligned_im, (x0, y0, x1, y1), landmarks = utils.
 ↪get_src_inputs(fn_src, fd, fp, idet)
tar, emb_tar = utils.get_tar_inputs(fns_tar, fd, fv)
```

[0]:
```python
out = model.inference(src, mask, tar, emb_tar)
```

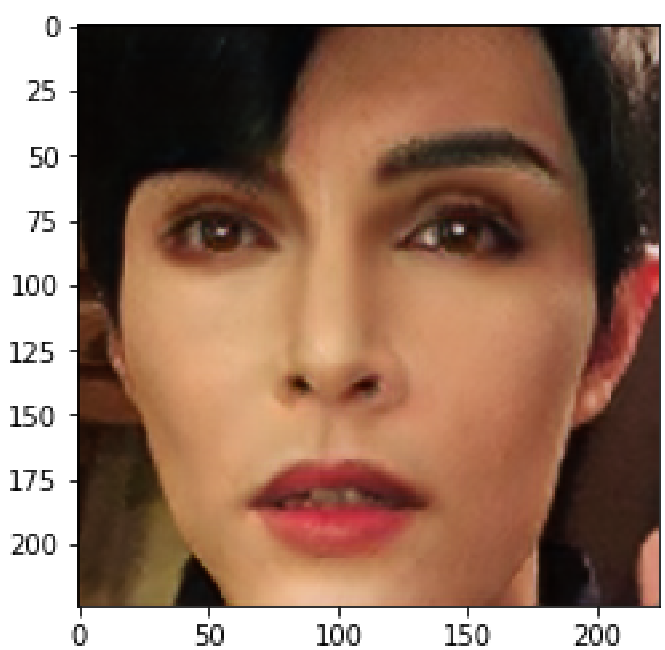
[212]:
```python
plt.figure()
plt.subplot(1,2,1)
plt.imshow(src)
plt.subplot(1,2,2)
plt.imshow(tar)
```

[212]:
```
<matplotlib.image.AxesImage at 0x7f7601d9b358>
```



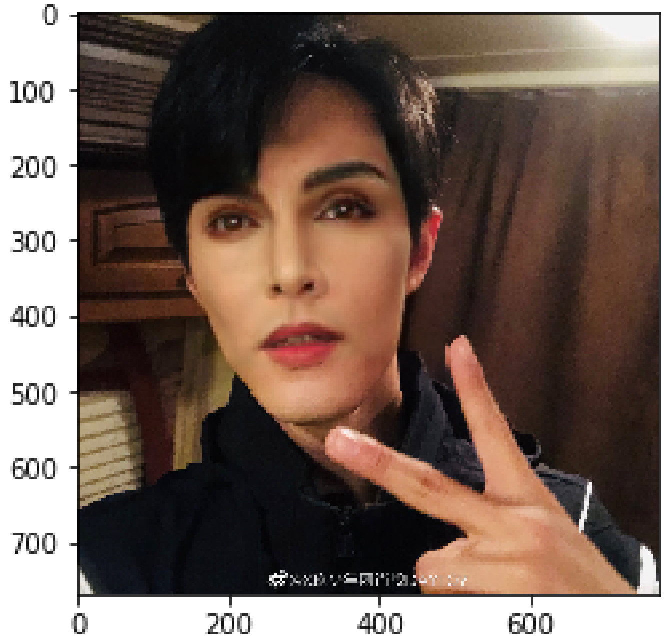
[213]:
```
result_face = np.squeeze(((out[0] + 1) * 255 / 2).astype(np.uint8))
plt.imshow(result_face)
```

[213]: <matplotlib.image.AxesImage at 0x7f7601d04ac8>



[214]:
```
result_img = utils.post_process_result(fn_src, fd, result_face, aligned_im, src, x0, y0, x1, y1, landmarks)
plt.imshow(result_img)
```

[214]: <matplotlib.image.AxesImage at 0x7f7601ce85f8>



We can see that, the source picture is successfully transformed onto target picture. And it look consistent with the original source picture.

[ ]: