National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analytics

Hsin-liang Liu
Student ID: X19116829

School of Computing
National College of Ireland

Supervisor: Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Hsin-liang Liu
Student ID:	X19116829
Programme:	MSc in Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Christian Horn
Submission Due Date:	14/08/2020
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	7

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	31st July 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Hsin-liang Liu
X19116829

1 Introduction

I built my deep neural network entirely on the colaboratory or colab. In this manual, I will share my view on colab and show you the configuration to help users get an overview of the working environment for the two stage-architectures. I will also introduce some of important API we use in this project and what is the difficulty we encounter while we work on building the model. The challenge includes the design of the neural network and familiar ourselves to the cv2 module and keras.layers methods.

In general, this manual can be divided into three part which are application environment, image processing Artefacts and neural network configuration so the users can get a better understanding about how the project is built. In addition to project report, this manual provide more details and implementation about the two-stage architectures.

2 Working Environment and Configuration

2.1 Colab

Colaboratory or colab is one of google service that provide developers an environment to construct executable python code. It is on google cloud server so it is convenient for developers who are authorized to access google cloud server when it is connected. For the sake of the computational ability, I choose colab platform for this project because it is free and it offer GPU accelerator and 12GB Ram without much of configuration. The environment is similar to the Jupyter where user code in the cell and is able to rearrange the sequence of the cell and it allows you to compile the code or load library only once every time you open it. It is already equipped with library or models that associated with deep learning project such as Tensorflow or Pytorch. However, there are also some of disadvantages using colab. The connection between users' local environment and google cloud server is not guarantee due to the quality of connection. In fact, it happen many times that I fail connecting to colab server and have to wait for a while to be able to connect to the server. Second, whatever you have to install in colab, you have to install them again in every new session. Once the session is finished it will clear the memory.

I save the weights of our neural network as .h5 file in users files and download them to my local environments otherwise, it will be deleted it when is session is finished in the project. The data and the other object such as haar are also stored in google drive so it can be easily retrieved from colab.

2.2 Working Environment

Below is some of the colab and environment configuration:

- hardware:
  - RAM: 12.72GB
  - Disk: 107.7DB
  - GPU: True
- software:
  - Python version: 3.6.7(default)
  - Tensorflow: version: 2.2.0
  - Keras: version: 2.3.1
  - openCV version: 4.1.2

You can follow the instruction from the google document to make GPU available (Edit → Notebook Settings → Hardware Accelerator → change it from None to GPU). Usually, we use the default version of Python, Tensorflow, Keras and cv2 in this project unless you specify them.

3 Two-stage Architecture Parameters

In early experiment, I built different version of model and different value for parameters to find out what version can lead to better performance. The higher the version is the newer.

Images		
ima_width	128	images is 128 × 128
sharpen_image	True	whether to performan image denoisng
Neural Network		
kernel_size	3	
filters	8	
encoder_stride	(2, 2, 1)	represent the stride in (E1, E2, E3)
decoder_stride	(2, 2, 1)	represent the stride in (D1, D2, D3)
latent_dim	28	size of latent variable version 6
latent_dim_v5	23	size of latent variable version 5
batch_size	32	
epochs	30	
Feed-forward Scaling		
λ_1	0.3	attention scale between E2 output to D3 output
λ_2	0.5	attention scale between E1 output to D2 output
λ_3	0.1	scale of z_c
λ_4	0.3	
λ_5	1	scale of σ_{z_c}
λ_6	0.9	scale of σ_{z_m}
File Source		
filename_prefix	/content/gdrive/My Drive/data/face_128px_	
face_cascade_file	/content/gdrive/My Drive/haarcascade/haarcascade_face.xml	
eye_cascade_file	/content/gdrive/My Drive/haarcascade/haarcascade_eye.xml	
cvae_one_file	v6_one_weights.h5	
cvae_two_file	v6_two_weights.h5	
cvae_one_file_v5	v5_one_weights.h5	
cvae_two_file_v5	v5_two_weights.h5	

4 Application Method List

This project is built by different components such as ImageTool, CVAE and pltTool. Moreover, some components can be breakdown to even smaller sub-component. For instance, the random mask block and face construction block. Each components serve different purpose. Below is a list of some of the methods that were used in the project.

4.1 Data Preparation and Measuring

retrieve_batch()	retrieving batch images from index start to the index end
drive.mount()	mount the current working environment to specific folder
getPSNR()	derive the PSNR between two images
getSSIM()	derive the SSIM between two images
getPSNR_SSIM()	get PSNR and SSIM for an array of two images
verify_eyes()	verify if arrays contains eyes position is valid
retrieve_eyes()	retrieve eyes position
get_face_center()	return the center line of face
random_hole_construct()	given an images, create random holes and construct the damaged images
construct()	given a corrupted images and generated images by CVAE, return the reconstructed images
plt_images()	given an array of corrupted images and constructed images, display them for comparison

4.2 Image Processing

The methods related to image processing are wrapped in ImageTool class. For example, we have methods for convert image to bgr format, create random mask for our first stage or remove background noise for images. Users can view some of methods for image processing below.

rgb2BGR()	convert image from rgb to bgr format
bgr2RGB()	convert image from bgr to rgb format
bgr_overlap()	overlap two bgr images
rgb_overlap()	overlap two rgb images
bgr_overlap_weight()	overlap two rgb images with specify scale
bgr_denoise()	bgr images denoising
rgb_denoise()	rgb images denoising
bgr_denoise()	bgr images denoising
bgr_blank_2_black()	convert the white patch to black patch fro bgr images
rgb_blank_2_black()	convert the white patch to black patch fro rgb images
bgr_comp_mask()	get the mask that results in the given bgr complement images
rgb_comp_mask()	get the mask that results in the given rgb complement images
bgr_flip_construct()	return the patch from the left half-face for the corrupted images
rgb_flip_construct()	return the patch from the left half-face for rgb images
bgr_random_blank()	return the bgr images with random holes
rgb_random_blank()	return the rgb images with random holes
bgr_random_mask()	return the random mask for bgr images
rgb_random_mask()	return the random mask for rgb images
bgr_mask_out()	return the bgr images after filtered by mask and inverse mask
rgb_mask_out()	return the rgb images after filtered by mask and inverse mask
getEyes()	get eyes position

4.3 Dual-pipeline Network

Our dual-pipeline network are mainly composed of 3 sub-encoder and 3 sub-decoder. We decouple the encoder to E1, E2 and E3 and so does D1, D2 and D3 for decoder. Here is the list of methods related to dual-pipeline network.

reparameterization()	sampling from the distribution with specify mean and standard deviation
kl_m_c()	derive the kl-divergence between two distribution
construction_loss()	construction loss between original and generated images
getE1()	create E1 subnet version 6
getE2()	create E2 subnet version 6
getE3()	create E3 subnet version 6
getD1()	create D1 subnet version 6
getD2()	create D2 subnet version 6
getD3()	create D3 subnet version 6
getEncoder_two()	create encoder for dual-pipeline network v6
getDecoder_two()	create decoder for dual-pipeline network v6
getCVAE_two()	create CVAE dual-pipeline network v6
getE1_v5()	create E1 subnet version 5
getE2_v5()	create E2 subnet version 5
getE3_v5()	create E3 subnet version 5
getD1_v5()	create D1 subnet version 5
getD2_v5()	create D2 subnet version 5
getD3_v5()	create D3 subnet version 5
getEncoder_two_v5()	create encoder for dual-pipeline network v5
getDecoder_two_v5()	create decoder for dual-pipeline network v5
getCVAE_two_v5()	create CVAE dual-pipeline network v5

5 Process Steps

1. Open the file v5, v6 in colab and make sure the GPU is available and set the parameters save_config to True which we tell the system that will save the model's weights after training. (v5 and v6 are different version of dual-pipeline models)

2. Specify the start and end parameter in retriev_batch(). We have 24 batch images so we will open from the start index to the end index batch for training.

3. Determine how many runs you like to train the data in a for loop. Training time might take a while and after that you can decide to retrieve test data for testing or continue training and using different batch images

4. After you satisfy with the training result, save the model's weight to the user folder.

5. Open the v7 file and make the GPU is available otherwise, it will take much longer to process the data.

6. Load the previous v5 and v6 weights to the user folder and make sure the parameters load_config is True because we will the load the model v5 and v6's weights later.

7. When everything is ready, you can run v7 and plot the result and save the weight and parameters and evaluate the outcome.

6 Summary

Since we can execute cells in colab individually, we break down the code to different methods and in different cell that make the mechanism more flexible while we building the two-stage architectures. In the future, we can use the provided interface to build the deep learning model that might shorten the building time and it is more flexible. Besides, tensorflow, we can also use Pytorch which is also an open-source and is available in colab. Pytorch provides dynamic computation graphs that allow us to monitor the computation time. Overall it is a good learning experience. It is not my first time using cv2 and tensorflow, but it enhance our knowledge about tensorflow module and help us learn CNN as well.

References

APPENDIX

July 31, 2020

[ ]:
```
import tensorflow as tf
import tensorflow_probability as tfp
from keras.layers import Dense, Input
from keras.layers import Conv2D, Flatten, Lambda, Conv3D
from keras.layers import Reshape, Conv2DTranspose, Conv3DTranspose
from keras.layers import concatenate, Add, Multiply
from keras.models import Model
from keras.losses import mse, binary_crossentropy
from keras import backend as K
from skimage.measure import compare_ssim

import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from google.colab.patches import cv2_imshow

import h5py
import math
import random
from math import log10, sqrt
from functools import reduce
import time

import imutils
from imutils import face_utils

from google.colab import drive

%matplotlib inline
```

```
Using TensorFlow backend.
```

[ ]:
```
# mount the drive, can the file is retrievable under this path
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call
drive.mount("/content/gdrive", force_remount=True).
```

[ ]:
```
# image and model configuration
image_size = 128
batch_size = 32
kernel_size = 3
filters = 8
stride = 1
estrides= (2,2,1)
dstrides = (2,2,1)
latent_dim = 28
epochs = 30

# feedward input scaling
lam1 = 0.3    #eo1
lam2 = 0.5    #eo2

lam3 = 0.1    #zc
lam4 = 0.3    #kl_div

lam5 = 1
lam6 = 0.9

# image configuration
sharpen_image = True

# model configuration
load_config = True
save_config = True

# file configuration
filename_prefix = '/content/gdrive/My Drive/data/face_128px_'
filename_test = '/content/gdrive/My Drive/data/face_test_128px_1.h5'
filename_demo = '/content/gdrive/My Drive/data/demo1.h5'
data_column_name = 'input_data'
cvae_one_weight_filename = 'v6_one_weights.h5'
cvae_two_weight_filename = 'v6_two_weights.h5'
face_cascade_filename = '/content/gdrive/My Drive/haarcascade/
 ↪haarcascade_frontalface_default.xml'
eye_cascade_filename = '/content/gdrive/My Drive/haarcascade/haarcascade_eye.xml'

# model data size configuration
ins1 = int(image_size/4)
ins2 = int(image_size/16)
e1_input_shape = (image_size, image_size, 3)
rate_input_shape = (1, )
e2_input_shape = (ins1, ins1, int(filters*4))
e3_input_shape = (ins2, ins2, int(filters*16))
e3_output_shape = (latent_dim,)
```

```
d3_input_shape = (latent_dim,)
d2_input_shape = (ins2, ins2, int(filters*16))
d1_input_shape = (ins1, ins1, int(filters*4))
```

[ ]:
```
# retrieve face and eye haar cascade file
face_cascade = cv2.CascadeClassifier(face_cascade_filename)
eye_cascade = cv2.CascadeClassifier(eye_cascade_filename)
```

[ ]:
```
def get_image(filename):
    '''
    Retreve images

    Parameters:

    Returns:
    array of images
    '''

    dataset = []

    try:
      with h5py.File(filename, "r") as f:
        dataset.append(f[data_column_name][:])
    except:
      print('Fail open file on ')

    x = [im for batch in dataset for im in batch]
    test_img = np.reshape(x, [-1, image_size, image_size, 3])

    return test_img


def retrieve_batch(start=1, end=2):
  '''
  get specified batch images

  Parameters:
  start (int): starting index
  end   (int): ending index

  Returns:
  array of images
  '''

  dataset = []

  for i in range(start, end, 1):
```

```
    try:
      filename = filename_prefix + str(i) + ".h5"
      with h5py.File(filename, "r") as f:
        dataset.append(f[data_column_name][:])
    except:
      print('Fail open file on ' + filename_prefix + str(i) + ".h5" )

  x = [im for batch in dataset for im in batch]
  batch_img = np.reshape(x, [-1, image_size, image_size, 3])

  return batch_img

def retrieve_test(filename = filename_test):
    '''
    Retreve test images

    Parameters:

    Returns:
    array of images
    '''

    return get_image(filename)
```

[ ]:
```
class ImageTool:
  def __init__(self, img_width):
    self.img_width = img_width
    self.half_width = int(img_width/2)
    self.total_pixel = img_width * img_width

  def rgb2BGR(self, img):
    '''
    convert rgb images to bgr images

    Parameters:
    img : rgb images you want to convert to bgr files

    Returns:
    bgr images
    '''
    img = np.array(img * 255, dtype=np.uint8)
    return cv2.cvtColor(img, cv2.COLOR_RGB2BGR)

  def bgr2RGB(self, img):
    '''
    convert bgr images to rgb images
```

```
    Returns:
    rgb images
    '''
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    img = img/255
    return np.array(img, dtype=np.float32)

  def bgr_overlap(self, img1, img2):
    '''
    add two bgr images together
    '''
    return cv2.addWeighted(img1, 1, img2, 1, 0)

  def rgb_overlap(self, img1, img2):
    '''
    add two rgb images together
    '''
    img1b = self.rgb2BGR(img1)
    img2b = self.rgb2BGR(img2)
    imgb = self.bgr_overlap(img1b, img2b)
    return self.bgr2RGB(imgb)

  def bgr_overlap_weight(self, img1, w1, img2, w2):
    '''
    overlap two bgr images with specified weight

    Parameters:
    img1  : image one
    w1    : weight of image one
    img2  : image two
    w2    : weight of image two

    Returns:
    bgr images
    '''
    return cv2.addWeighted(img1, w1, img2, w2, 0)

  def rgb_overlap_weight(self, img1, w1, img2, w2):
    '''
    overlap two rgb images with specified weight

    Parameters:
    img1  : image one
    w1    : weight of image one
    img2  : image two
    w2    : weight of image two
```

```
    '''
    img1b = self.rgb2BGR(img1)
    img2b = self.rgb2BGR(img2)
    imgb = self.bgr_overlap_weight(img1b, w1, img2b, w2)
    return self.bgr2RGB(imgb)

  def bgr_denoise(self, img, h = 3):
    '''
    denoise bgr images

    Parameters:
    h : default is 3, higher h represents more smooths but lack of details
    '''
    return cv2.fastNlMeansDenoisingColored(img, None, h, h, 7, 21)

  def rgb_denoise(self, img, h = 3):
    '''
    denoise rgb images
    '''
    converted_img = self.rgb2BGR(img)
    denoise_img = self.bgr_denoise(converted_img, h)
    return self.bgr2RGB(denoise_img)

  def bgr_blank_2_black(self, img):
    '''
    convert white pixels(missing) to black pixels of bgr images
    '''
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv, np.array([0, 0, 254]), np.array([0, 0, 254]))
    mask_inv = cv2.bitwise_not(mask)
    return cv2.bitwise_and(img, img, mask = mask_inv), mask, mask_inv

  def rgb_blank_2_black(self, img):
    '''
    convert white pixels(missing) to black pixels of rgb images
    '''
    img = self.rgb2BGR(img)
    img_dist, mask, mask_inv = self.bgr_blank_2_black(img)
    return self.bgr2RGB(img_dist), mask, mask_inv

  def bgr_comp_mask(self, img):
    '''
    return complement mask of bgr images

    Returns:
    bgr image
    '''
```

```
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv, np.array([0, 0, 0]), np.array([0, 0, 0]))

    return mask

  def rgb_comp_img(self, corr_img, full_img):
    '''
    return complement mask of rgb images according to the corrupted images

    Parameters:
    corr_img: corrupted images or damaged images
    full_img: complete images
    '''
    corr_img = self.rgb2BGR(corr_img)
    full_img = self.rgb2BGR(full_img)
    mask = self.bgr_comp_mask(corr_img)
    img_out = cv2.bitwise_and(full_img, full_img, mask = mask)
    return self.bgr2RGB(img_out)

  def rgb_black_hole(self, img):
    return self.rgb_blank_2_black(img)[0]

  def bgr_flip_construct(self, img, w = None):
    '''
    face construction block: given damaged images return Ic and
    the corresponding mask of bgr images

    Returns:
    bgr images
    '''
    if w is None:
      w = self.half_width

    img, mask, mask_inv = self.bgr_blank_2_black(img)
    img_rgb = self.bgr2RGB(img)


    mask_non_zero_count = np.count_nonzero(mask)
    mask_non_zero_rate = round(mask_non_zero_count/self.total_pixel, 2)

    img_output = None
    imgrb = None
    if w < self.half_width:
      crop_x_s = 0
      imgrb = np.zeros([128, (128-2*w), 3], dtype=np.float32)
    else:
      crop_x_s = (2*w-self.img_width)
```

```
    cropped = img_rgb[0:self.img_width, crop_x_s:w]
    flip = cv2.flip(cropped, 1)
    flip = np.array(flip, dtype=np.float32)
    imglb = np.zeros([128, w, 3], dtype=np.float32)
    full = cv2.hconcat([imglb, flip])
    if w < self.half_width:
      full = cv2.hconcat([full, imgrb])

    full = self.rgb2BGR(full)
    return cv2.bitwise_and(full, full, mask = mask), img, mask_non_zero_rate

  def rgb_flip_construct(self, img, w = None):
    '''
    face construction block: given damaged images return Ic and the
    corresponding mask of rgb images
    '''
    img = self.rgb2BGR(img)
    fli_img, cor_img, fli_img_non_zero_rate  = self.bgr_flip_construct(img, w)
    return self.bgr2RGB(fli_img), self.bgr2RGB(cor_img), fli_img_non_zero_rate

  def bgr_circle_blank(self, img, x, y, r):
    return cv2.circle(img, (int(x), int(y)), int(r), [255, 255, 255],␣
↪thickness=-1)

  def rgb_circle_blank(self, img, x, y, r):
    img = self.rgb2BGR(img)
    img_output = cv2.circle(img, (int(x), int(y)), int(r), [255,255,255],␣
↪thickness=-1)
    return self.bgr2RGB(img_output)

  def rgb_random_blank(self, img):
    '''
    randomly generate hole

    Returns:
    rgb images filtered by random mask
    '''
    img_out = self.rgb2BGR(img)
    x_range = (int(self.img_width*0.56), int(self.img_width*0.8))
    y_range = (int(self.half_width/5), int(self.img_width*0.8))
    r_range = (int(self.half_width/15), int(self.half_width/4))
    w_range = (int(self.half_width/18), int(self.half_width/5))
    ex = random.randint(x_range[0], x_range[1])
    ey = random.randint(y_range[0], y_range[1])
    er1 = random.randint(r_range[0], r_range[1])
    er2 = random.randint(r_range[0], r_range[1])
```

```
    ea = random.randint(0, 60)

    img_out = cv2.ellipse(img_out, (ex, ey), (er1, er2), ea, 0, 360,␣
↪[255,255,255], thickness=-1)
    cx = random.randint(x_range[0], x_range[1])
    cy = random.randint(y_range[0], y_range[1])
    cr = random.randint(r_range[0], r_range[1])
    img_out = cv2.circle(img_out, (cx, cy), cr, [255, 255, 255], thickness=-1)
    w = random.randint(w_range[0], w_range[1])
    rx = random.randint(x_range[0], x_range[1])
    ry = random.randint(y_range[0], y_range[1])
    h = random.randint(w_range[0], w_range[1])
    img_out = cv2.rectangle(img_out, (rx, ry), (rx+w, ry+h), [255, 255, 255],␣
↪thickness=-1)
    img_out = self.bgr2RGB(img_out)

    return img_out

 def bgr_random_mask(self):
    '''
    randomly generaly mask

    Returns:
    random masks
    '''
    x_range = (int(self.half_width*0.97), int(self.img_width*0.84))
    y_range = (int(self.half_width/6), int(self.img_width*0.84))
    r_range = (int(self.half_width/14), int(self.half_width/4))
    w_range = (int(self.half_width/18), int(self.half_width/6))
    ex = random.randint(x_range[0], x_range[1])
    ey = random.randint(y_range[0], y_range[1])
    er1 = random.randint(r_range[0], r_range[1])
    er2 = random.randint(r_range[0], r_range[1])
    ea = random.randint(0, 60)
    mask = np.zeros([self.img_width, self.img_width, 3], dtype=np.uint8)
    mask = cv2.ellipse(mask, (ex, ey), (er1, er2), ea, 0, 360, [255,255,255],␣
↪thickness=-1)
    cx = random.randint(x_range[0], x_range[1])
    cy = random.randint(y_range[0], y_range[1])
    cr = random.randint(r_range[0], r_range[1])
    mask = cv2.circle(mask, (cx, cy), cr, [255, 255, 255], thickness=-1)
    w = random.randint(w_range[0], w_range[1])
    rx = random.randint(x_range[0], x_range[1])
    ry = random.randint(y_range[0], y_range[1])
    h = random.randint(w_range[0], w_range[1])
    mask = cv2.rectangle(mask, (rx, ry), (rx+w, ry+h), [255, 255, 255],␣
↪thickness=-1)
```

```
    if random.getrandbits(1) == 1:
      rx = int((rx + cx)/2)
      ry = int((cy + ey)/2)
      w = int((w + h)/3)
      h = int((cr + er1)/4)
      mask = cv2.rectangle(mask, (rx, ry), (rx+w, ry+h), [255, 255, 255],␣
↪thickness=-1)

    mask = cv2.cvtColor(mask, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(mask, np.array([0, 0, 255]), np.array([0, 0, 255]))
    mask_inv = cv2.bitwise_not(mask)
    return mask_inv, mask

  def rgb_mask_out(self, img):
    '''
    given images, filters it by randomly generated mask
    mask block

    Returns:
    rgb images
    '''
    mask_inv, mask = self.bgr_random_mask()
    mask_inv_non_zero_count = np.count_nonzero(mask_inv)
    mask_inv_non_zero_rate = round(mask_inv_non_zero_count/self.total_pixel, 2)
    img = self.rgb2BGR(img)
    img1 = cv2.bitwise_and(img, img, mask = mask_inv)
    img2 = cv2.bitwise_and(img, img, mask = mask)
    img1 = self.bgr2RGB(img1)
    img2 = self.bgr2RGB(img2)

    return img1, img2, mask_inv_non_zero_rate

  def getEyes(self, img):
    '''
    retrieve eyes position
    '''
    copy = img.copy()
    copy = self.rgb2BGR(copy)
    gray = cv2.cvtColor(copy, cv2.COLOR_BGR2GRAY)
    gray = np.array(gray, dtype=np.uint8)
    eyes = eye_cascade.detectMultiScale(gray, 1.2, 5)

    return eyes

  def showEyesImage(self, img):
    '''
    display eyes position on the images
```

```
    '''
    copy = img.copy()
    copy = self.rgb2BGR(copy)
    gray = cv2.cvtColor(copy, cv2.COLOR_BGR2GRAY)
    gray = np.array(gray, dtype='uint8')
    eyes = eye_cascade.detectMultiScale(gray, 1.2, 5)

    for (ex,ey,ew,eh) in eyes:
      cv2.rectangle(copy, (ex,ey), (ex+ew,ey+eh), (0, 0, 100), 2)
    cv2_imshow(copy)

    return copy

itool = ImageTool(image_size)
```

[ ]:
```
# retrieve training and test batch images
batch_img_train = retrieve_batch(1,2)
batch_img_test = retrieve_batch(2,3)
```

[ ]:
```
# denoise images
if sharpen_image:
  batch_img_train = [itool.rgb_denoise(img) for img in batch_img_train]
  batch_img_test = [itool.rgb_denoise(img) for img in batch_img_test]
```

[ ]:
```
def getPSNR(orig, comp):
  '''
  Parameters:
  orig: image1
  comp: image2

  Returns:
  int: psnr
  '''
  mse = np.mean((orig - comp) ** 2)
  if mse == 0: return 100
  psnr = 20 * log10(1/sqrt(mse))

  return psnr

def getSSIM(orig, comp):
  '''
  Parameters:
  orig: image1
  comp: image2

  Returns:
  return ssim
```

```
  '''
  return compare_ssim(orig, comp, full=True, multichannel=True)

def avg(arr):
  '''
  Returns:
  int: average value
  '''
  av = reduce(lambda a, b: a + b, arr) / len(arr)

  return round(av, 2)
```

[ ]:
```
'''
Part I: Building Model
'''
```

[ ]:
```
'\nPart I: Building Model\n'
```

[ ]:
```
def reparameterization(args):
    '''
    Reparameterization

    Parameters:
    args : compose of z_mu and z_log_var
    '''
    z_mu, z_log_var = args
    batch = K.shape(z_mu)[0]
    dim = K.int_shape(z_mu)[1]
    e = K.random_normal(shape=(batch, dim))

    return z_mu + K.exp(0.5 * z_log_var) * e
```

[ ]:
```
kl = tf.keras.losses.KLDivergence()

def kl_m_c(x, y):
  '''
  kl divergences of two distribution

  Parameters:
  x : first distribution
  y : second distribution
  '''
  return kl(x, y)

def kl_div(z_mean, z_log_var):
  '''
  kl divergence loss function
```

```
    '''
    kl_loss = 1 + z_log_var - K.square(z_mean) - K.exp(z_log_var)
    kl_loss = K.sum(kl_loss, axis=-1)
    kl_loss *= -0.5
    return kl_loss

def construction_loss(dinput, doutput):
    '''
    construction loss function

    Parameters:
    dinput  : original image
    doutput : constructed image
    '''
    construct_loss = lam5 * mse(K.flatten(dinput), K.flatten(doutput))
    construct_loss += (1-lam5) * binary_crossentropy(K.flatten(dinput), K.
 ↪flatten(doutput))
    construct_loss *= image_size * image_size

    return construct_loss

def getLoss_one(dinput, doutput, corr_z_mu, corr_z_logvar):
    '''
    loss fucntion for one pipeline network
    '''
    con_loss = construction_loss(dinput, doutput)
    k_loss = kl_div(corr_z_mu, corr_z_logvar)

    cvae_loss = K.mean(con_loss + k_loss)

    return cvae_loss


def getLoss_two(dinput, doutput, corr_z_mu, corr_z_logvar, corr_z, comp_z, l4):
    '''
    loss function for dual-pipeline network
    '''
    con_loss = construction_loss(dinput, doutput)
    k_loss = kl_div(corr_z_mu, corr_z_logvar)
    k_mc = kl_m_c(corr_z, comp_z)
    k_loss += l4 * k_mc

    cvae_loss = K.mean(con_loss + k_loss)

    return cvae_loss
```

[ ]:
```
def getE1():
  '''
  create E1 model

  Returns:
  E1 model
  '''

  filters_e1 = filters
  e1_input = Input(shape=e1_input_shape, name='e1_input')
  x1 = e1_input

  x1 = Conv2D(filters=filters_e1,
              kernel_size=kernel_size,
              activation='relu',
              strides=stride,
              padding='same')(x1)

  for i in range(2):
    filters_e1 *= 2
    x1 = Conv2D(filters=filters_e1,
              kernel_size=kernel_size,
              activation='relu',
              strides=estrides[i],
              padding='same')(x1)
  e1_output = x1
  e1 = Model(e1_input, e1_output)
  return e1
```

[ ]:
```
def getE2():
  '''
  create E2 model
  '''

  filters_e2 = filters * 4
  e2_input = Input(shape=e2_input_shape, name='e2_input')
  x1 = e2_input

  x1 = Conv2D(filters=filters_e2,
              kernel_size=kernel_size,
              activation='relu',
              strides=stride,
              padding='same')(x1)

  for i in range(2):
    filters_e2 *= 2
    x1 = Conv2D(filters=filters_e2,
```

```
                kernel_size=kernel_size,
                activation='relu',
                strides=estrides[i],
                padding='same')(x1)
    e2_output = x1
    e2 = Model(e2_input, e2_output)
    return e2
```

[ ]:
```
def getE3(l6 = lam6):
    '''
    create E3 model
    '''

    e3_input = Input(shape=e3_input_shape, name='e3_input')
    x1 = e3_input
    x1 = Flatten()(x1)
    x1 = Dense(512, activation='relu')(x1)

    z_mu = Dense(latent_dim, activation='linear', name='z_mu')(x1)
    z_logvar = Dense(latent_dim, activation='linear', name='z_log_var')(x1)
    z_logvar = Lambda(lambda x: x * l6)(z_logvar)
    z = Lambda(reparameterization, output_shape=e3_output_shape, name='z')([z_mu,␣
↪z_logvar])
    e3 = Model(e3_input, [z_mu, z_logvar, z])

    return e3
```

[ ]:
```
def getD3():
    '''
    create D3 model
    '''
    latent_input = Input(shape=d3_input_shape, name='d3_input')
    x1 = Dense(512, activation='relu')(latent_input)
    x1 = Dense(e3_input_shape[0] * e3_input_shape[1] * e3_input_shape[2],␣
↪activation='relu')(x1)
    d3_output = Reshape((e3_input_shape))(x1)
    d3 = Model(latent_input, d3_output)
    return d3
```

[ ]:
```
def getD2():
    '''
    create D2 model
    '''
    filters_d2 = filters * 16
    d2_input = Input(shape=d2_input_shape, name='d2_input')
    x1 = d2_input
    for i in range(2):
```

```
    filters_d2 //= 2
    x1 = Conv2DTranspose(filters=filters_d2,
                         kernel_size=kernel_size,
                         activation='relu',
                         strides=dstrides[i],
                         padding='same')(x1)

  x1 = Conv2DTranspose(filters=filters_d2,
                         kernel_size=kernel_size,
                         activation='relu',
                         strides=stride,
                         padding='same')(x1)
  d2_output = x1
  d2 = Model(d2_input, d2_output)

  return d2
```

[ ]:
```
def getD1():
  '''
  create D1 model
  '''
  filters_d1 = filters * 4
  d1_input = Input(shape=d1_input_shape, name='d1_input')
  x1 = d1_input
  for i in range(2):
    filters_d1 //= 2
    x1 = Conv2DTranspose(filters=filters_d1,
                         kernel_size=kernel_size,
                         activation='relu',
                         strides=dstrides[i],
                         padding='same')(x1)

  x1 = Conv2DTranspose(filters=3,
                           kernel_size=kernel_size,
                           activation='sigmoid',
                           padding='same',
                           name='decoder_output')(x1)
  d1_output = x1
  d1 = Model(d1_input, d1_output, name='d1')

  return d1
```

[ ]:
```
def getEncoder_one():
  '''
  create Encoder for one pipeline network
  '''
  corr_input = Input(shape=e1_input_shape, name='corr_input_1')
```

```
  real_input = Input(shape=e1_input_shape, name='real_input_1')

  e1 = getE1()
  e2 = getE2()
  e3 = getE3(lam6);

  eo1 = e1(corr_input)
  eo2 = e2(eo1)
  [corr_z_mu, corr_z_logvar, corr_z] = e3(eo2)

  encoder = Model([corr_input, real_input], [corr_z_mu, corr_z_logvar, corr_z],␣
 ↪name='encoder_one')

  return encoder


def getEncoder_two(l3 = lam3):
  '''
  create Encoder for dual-pipeline network

  Parameters:
  l3 : lambda_3

  Returns:
  encoder
  '''
  corr_input = Input(shape=e1_input_shape, name='corr_input_1')
  comp_input = Input(shape=e1_input_shape, name='comp_input_1')
  real_input = Input(shape=e1_input_shape, name='real_input_1')

  e1 = getE1()
  e2 = getE2()
  e3 = getE3(lam6)
  e3_ = getE3(1 - lam6)

  eo1_0 = e1(corr_input)
  eo1_1 = e1(comp_input)

  eo2_0 = e2(eo1_0)
  eo2_1 = e2(eo1_1)

  [corr_z_mu, corr_z_logvar, corr_z] = e3(eo2_0)
  [comp_z_mu, comp_z_logvar, comp_z] = e3_(eo2_1)
  comp_z_ = Lambda(lambda x: x * l3)(comp_z)
  z = Add()([corr_z, comp_z_])
```

```
  encoder = Model([corr_input, comp_input, real_input], [eo1_1, eo2_1,␣
↪corr_z_mu, corr_z_logvar, corr_z, comp_z, z], name='encoder_two')

  return encoder
```

[ ]:

[ ]:
```
def getDecoder_one():
  '''
  create decoder for one-pipeline network
  '''
  z_input = Input(shape=d3_input_shape, name='z_input')
  d3 = getD3()
  d2 = getD2()
  d1 = getD1()

  do3 = d3(z_input)
  do2 = d2(do3)
  do1 = d1(do2)

  decoder = Model(z_input, do1)

  return decoder


def getDecoder_two(l1, l2):
  '''
  create decoder for dual-pipeline

  Parameters:
  l1  : lambda_1
  l2  : lambda_2

  Returns:
  decoder
  '''
  eo1_input = Input(shape=e2_input_shape, name='eo1')
  eo2_input = Input(shape=e3_input_shape, name='eo2')
  z_input = Input(shape=d3_input_shape, name='z_input')

  eo2 = Lambda(lambda x: x * l1)(eo2_input)
  eo1 = Lambda(lambda x: x * l2)(eo1_input)

  d3 = getD3()
  d2 = getD2()
  d1 = getD1()
```

```
  do3 = d3(z_input)

  do3 = Add()([do3, eo2])

  do2 = d2(do3)

  do2 = Add()([do2, eo1])

  do1 = d1(do2)

  decoder = Model([eo1_input, eo2_input, z_input], do1)
  print('final shape: ', K.int_shape(do1))

  return decoder
```

[ ]:
```
def getCVAE_one():
  '''
  create CVAE model for one-pipeline network
  '''
  corr_input = Input(shape=e1_input_shape, name='corr_input')
  real_input = Input(shape=e1_input_shape, name='real_input')

  encoder = getEncoder_one()
  decoder = getDecoder_one()

  encoder_output = encoder([corr_input, real_input])
  corr_z_mu = encoder_output[0]
  corr_z_logvar = encoder_output[1]
  corr_z = encoder_output[2]

  decoder_input = corr_z
  decoder_output = decoder(decoder_input)

  cvae = Model([corr_input, real_input], decoder_output, name='cvae_one')
  cvae_loss = getLoss_one(real_input, decoder_output, corr_z_mu, corr_z_logvar)
  cvae.add_loss(cvae_loss)
  cvae.compile(optimizer='rmsprop')

  return cvae

def getCVAE_two(l1 = lam1, l2 = lam2, l3 = lam3, l4 = lam4):
  '''
  create CVAE model for dual-pipeline network

  Returns:
  CVAE model
  '''
```

```
  corr_input = Input(shape=e1_input_shape, name='corr_input')
  comp_input = Input(shape=e1_input_shape, name='comp_input')
  real_input = Input(shape=e1_input_shape, name='real_input')

  encoder = getEncoder_two()
  decoder = getDecoder_two(l1, l2)

  encoder_output = encoder([corr_input, comp_input, real_input])
  eo1 = encoder_output[0]
  eo2 = encoder_output[1]
  corr_z_mu = encoder_output[2]
  corr_z_logvar = encoder_output[3]
  corr_z = encoder_output[4]
  comp_z = encoder_output[5]
  z = encoder_output[6]
  decoder_input = [eo1, eo2, z]

  decoder_output = decoder(decoder_input)

  cvae = Model([corr_input, comp_input, real_input], decoder_output,␣
↪name='cvae_two')

  cvae_loss = getLoss_two(real_input, decoder_output, corr_z_mu, corr_z_logvar,␣
↪corr_z, comp_z, l4)
  cvae.add_loss(cvae_loss)
  cvae.compile(optimizer='rmsprop')

  return cvae
```

[ ]:
```
cvae_one = getCVAE_one()
cvae_two = getCVAE_two()
```

```
/usr/local/lib/python3.6/dist-packages/keras/engine/training_utils.py:819:
UserWarning: Output model_6 missing from loss dictionary. We assume this was
done on purpose. The fit and evaluate APIs will not be expecting any data to be
passed to model_6.
  'be expecting any data to be passed to {0}.'.format(name))

final shape:  (None, 128, 128, 3)

/usr/local/lib/python3.6/dist-packages/keras/engine/training_utils.py:819:
UserWarning: Output model_13 missing from loss dictionary. We assume this was
done on purpose. The fit and evaluate APIs will not be expecting any data to be
passed to model_13.
  'be expecting any data to be passed to {0}.'.format(name))
```

[ ]:
```
'''
Part II: Training
```

```
'''
```

[ ]:
```
# load weights to the deep learning model
if load_config:
  print('loading weights ....')
  cvae_one.load_weights(cvae_one_weight_filename)
  cvae_two.load_weights(cvae_two_weight_filename)
  print('weights loaded')
```

```
loading weights ...
weights loaded
```

[ ]:
```
# prepare the images for training
# x_real_{train, test} : original images (Ig)
# x_corr_{train, test} : damaged images (Im)
# x_comp_{train, test} : complement images (Ic)
x_real_train = batch_img_train
x_proc_train = [(itool.rgb_mask_out(img)) for img in batch_img_train]
x_corr_train = [img_p[0] for img_p in x_proc_train]
x_comp_train = [img_p[1] for img_p in x_proc_train]
x_corr_rate_train = [img_p[2] for img_p in x_proc_train]
x_comp_rate_train = [round(1 - img_p[2],2) for img_p in x_proc_train]

# prepare the images for testing
x_real_test = batch_img_test
x_proc_test = [(itool.rgb_mask_out(img)) for img in batch_img_test]
x_corr_test = [img_p[0] for img_p in x_proc_test]
x_comp_test = [img_p[1] for img_p in x_proc_test]
x_corr_rate_test = [img_p[2] for img_p in x_proc_test]
x_comp_rate_test = [round(1 - img_p[2],2) for img_p in x_proc_test]
```

[ ]:
```
# train the one-pipeline model
cvae_one.fit([x_corr_train, x_real_train],
        epochs=epochs,
        batch_size=batch_size,
        validation_data=([x_corr_test, x_real_test], None))

# train the dual-pipeline model
cvae_two.fit([x_corr_train, x_comp_train, x_real_train],
        epochs=epochs,
        batch_size=batch_size,
        validation_data=([x_corr_test, x_comp_test, x_real_test], None))
```

```
Train on 1024 samples, validate on 1024 samples
Epoch 1/30
1024/1024 [==============================] - 3s 3ms/step - loss: 112.5431 -
val_loss: 175.4465
```

```
Epoch 2/30
1024/1024 [==============================] - 3s 3ms/step - loss: 109.9171 -
val_loss: 177.8724
Epoch 3/30
1024/1024 [==============================] - 3s 3ms/step - loss: 107.9367 -
val_loss: 171.7836
Epoch 4/30
1024/1024 [==============================] - 3s 3ms/step - loss: 107.9376 -
val_loss: 187.3106
Epoch 5/30
1024/1024 [==============================] - 3s 3ms/step - loss: 106.6637 -
val_loss: 175.2485
Epoch 6/30
1024/1024 [==============================] - 3s 3ms/step - loss: 106.2192 -
val_loss: 176.5980
Epoch 7/30
1024/1024 [==============================] - 3s 3ms/step - loss: 106.4724 -
val_loss: 172.4199
Epoch 8/30
1024/1024 [==============================] - 3s 3ms/step - loss: 108.7632 -
val_loss: 173.1914
Epoch 9/30
1024/1024 [==============================] - 3s 3ms/step - loss: 103.2873 -
val_loss: 178.1364
Epoch 10/30
1024/1024 [==============================] - 3s 3ms/step - loss: 104.2104 -
val_loss: 176.2289
Epoch 11/30
1024/1024 [==============================] - 3s 3ms/step - loss: 104.7643 -
val_loss: 174.9843
Epoch 12/30
1024/1024 [==============================] - 3s 3ms/step - loss: 104.3136 -
val_loss: 174.5447
Epoch 13/30
1024/1024 [==============================] - 3s 3ms/step - loss: 102.7491 -
val_loss: 187.5161
Epoch 14/30
1024/1024 [==============================] - 3s 3ms/step - loss: 104.6713 -
val_loss: 177.6894
Epoch 15/30
1024/1024 [==============================] - 3s 3ms/step - loss: 102.4322 -
val_loss: 174.8830
Epoch 16/30
1024/1024 [==============================] - 3s 3ms/step - loss: 102.3633 -
val_loss: 184.0606
Epoch 17/30
1024/1024 [==============================] - 3s 3ms/step - loss: 102.5719 -
val_loss: 174.8174
```

```
Epoch 18/30
1024/1024 [==============================] - 3s 3ms/step - loss: 102.5009 -
val_loss: 178.2848
Epoch 19/30
1024/1024 [==============================] - 3s 3ms/step - loss: 101.7103 -
val_loss: 174.2776
Epoch 20/30
1024/1024 [==============================] - 3s 3ms/step - loss: 101.6642 -
val_loss: 182.6636
Epoch 21/30
1024/1024 [==============================] - 3s 3ms/step - loss: 101.4300 -
val_loss: 174.9613
Epoch 22/30
1024/1024 [==============================] - 3s 3ms/step - loss: 100.9220 -
val_loss: 184.0848
Epoch 23/30
1024/1024 [==============================] - 3s 3ms/step - loss: 100.7866 -
val_loss: 175.2938
Epoch 24/30
1024/1024 [==============================] - 3s 3ms/step - loss: 101.6890 -
val_loss: 178.3906
Epoch 25/30
1024/1024 [==============================] - 3s 3ms/step - loss: 99.9041 -
val_loss: 177.3040
Epoch 26/30
1024/1024 [==============================] - 3s 3ms/step - loss: 101.0121 -
val_loss: 174.9505
Epoch 27/30
1024/1024 [==============================] - 3s 3ms/step - loss: 99.7680 -
val_loss: 175.5146
Epoch 28/30
1024/1024 [==============================] - 3s 3ms/step - loss: 99.4466 -
val_loss: 177.8252
Epoch 29/30
1024/1024 [==============================] - 3s 3ms/step - loss: 100.3886 -
val_loss: 180.1948
Epoch 30/30
1024/1024 [==============================] - 3s 3ms/step - loss: 100.0586 -
val_loss: 177.3328
```

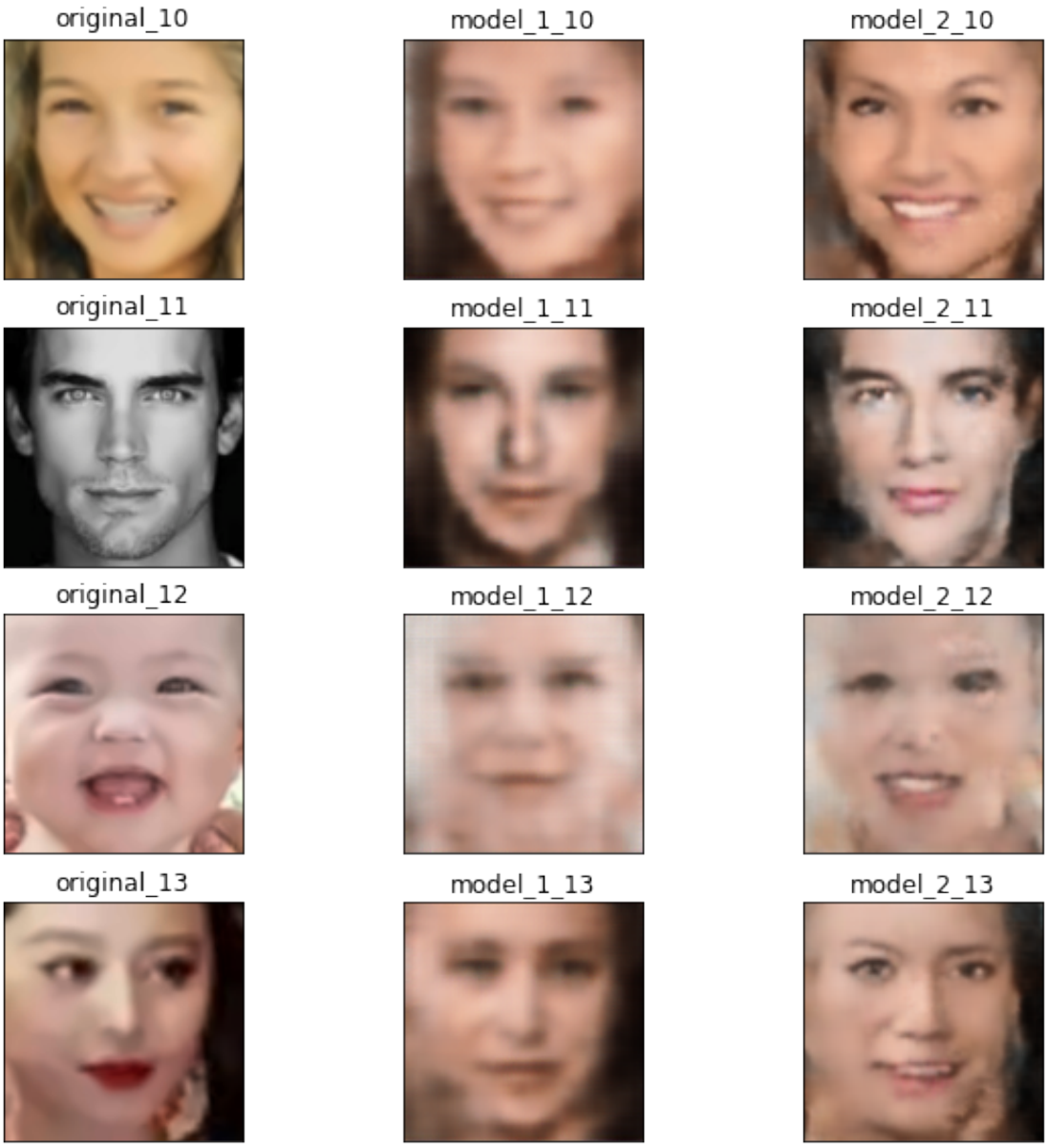
[ ]:
```
<keras.callbacks.callbacks.History at 0x7fb001948208>
```

[ ]:
```
# save the weights to the local file
if save_config:
  print('saving weights ....')
  cvae_one.save_weights(cvae_one_weight_filename)
  cvae_two.save_weights(cvae_two_weight_filename)
```

```
saving weights ...
```

[ ]:
```
# generate images based using one-pipeline network
predict_one = cvae_one.predict([x_corr_test, x_real_test], batch_size=128)

# generate images based using dual-pipeline network
predict_two = cvae_two.predict([x_corr_test, x_comp_test, x_real_test],␣
 ↪batch_size=128)
```

[ ]:
```
# display the results
# (left: original images, middle: one-pipeline network, right: dual-pipeline␣
 ↪network)

plt.figure(num='image', figsize=(10,10))

for i in range(1, 13, 3):
  idx = int((i-1)/3) + 10
  plt.subplot(4, 3, i)
  plt.title('original_{}'.format(idx))
  plt.xticks([])
  plt.yticks([])
  plt.imshow(x_real_test[idx])

  plt.subplot(4, 3, i+1)
  plt.title('model_1_{}'.format(idx))
  plt.xticks([])
  plt.yticks([])
  plt.imshow(predict_one[idx])

  plt.subplot(4, 3, i+2)
  plt.title('model_2_{}'.format(idx))
  plt.xticks([])
  plt.yticks([])
  plt.imshow(predict_two[idx])

plt.show()
```



[ ]:
```
psnr_one = []
ssim_one = []

psnr_two = []
ssim_two = []
for i in range(len(x_real_test)):
  pv_1 = getPSNR(x_real_test[i], predict_one[i])
  sv_1 = getSSIM(x_real_test[i], predict_one[i])
  psnr_one.append(pv_1)
  ssim_one.append(sv_1[0])
```

```
  pv_2 = getPSNR(x_real_test[i], predict_two[i])
  sv_2 = getSSIM(x_real_test[i], predict_two[i])
  psnr_two.append(pv_2)
  ssim_two.append(sv_2[0])

print('*-----------* Model One *--------------*')
print('Average of PSNR: ', avg(psnr_one))
print('Average of SSIM: ', avg(ssim_one))

print('*-----------* Model two *--------------*')
print('Average of PSNR: ', avg(psnr_two))
print('Average of SSIM: ', avg(ssim_two))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:25: UserWarning:
DEPRECATED: skimage.measure.compare_ssim has been moved to
skimage.metrics.structural_similarity. It will be removed from skimage.measure
in version 0.18.

*-----------* Model One *--------------*
Average of PSNR:  18.06
Average of SSIM:  0.69
*-----------* Model two *--------------*
Average of PSNR:  17.6
Average of SSIM:  0.68
```

[ ]:
```
# load the weights
if load_config:
  print('loading weights ....')
  cvae_one.load_weights(cvae_one_weight_filename)
  cvae_two.load_weights(cvae_two_weight_filename)
  print('weights loaded')
```

[ ]:
```
# load the batch images for training and testing
tic = time.time()

train_batch_number = random.randint(1, 23)
test_batch_number = random.randint(1, 24)

batch_img_train = retrieve_batch(train_batch_number, train_batch_number + 1)
batch_img_test = retrieve_batch(test_batch_number, test_batch_number + 1)

if sharpen_image:
  batch_img_train = [itool.rgb_denoise(img) for img in batch_img_train]
  batch_img_test = [itool.rgb_denoise(img) for img in batch_img_test]

toc = time.time()
print('Process Time: ', ((toc-tic) * 1000), 'ms')
```

```
x_real_train = batch_img_train
x_real_test = batch_img_test
```

```
Process Time:  180268.5351371765 ms
```

[ ]:
```
# generate different mask for Im, Ic in each iteration
for i in range(1):
  x_proc_train = [(itool.rgb_mask_out(img)) for img in batch_img_train]
  x_corr_train = [img_p[0] for img_p in x_proc_train]
  x_comp_train = [img_p[1] for img_p in x_proc_train]

  x_proc_test = [(itool.rgb_mask_out(img)) for img in batch_img_test]
  x_corr_test = [img_p[0] for img_p in x_proc_test]
  x_comp_test = [img_p[1] for img_p in x_proc_test]

  # train the model
  cvae_one.fit([x_corr_train, x_real_train],
          epochs=epochs,
          batch_size=batch_size,
          validation_data=([x_corr_test, x_real_test], None))


  cvae_two.fit([x_corr_train, x_comp_train, x_real_train],
          epochs=epochs,
          batch_size=batch_size,
          validation_data=([x_corr_test, x_comp_test, x_real_test], None))
```

```
Train on 1024 samples, validate on 1024 samples
Epoch 1/30
1024/1024 [==============================] - 3s 3ms/step - loss: 212.7904 -
val_loss: 175.2642
Epoch 2/30
1024/1024 [==============================] - 3s 3ms/step - loss: 189.6176 -
val_loss: 182.4616
Epoch 3/30
1024/1024 [==============================] - 3s 3ms/step - loss: 181.2772 -
val_loss: 177.7584
Epoch 4/30
1024/1024 [==============================] - 3s 3ms/step - loss: 173.4009 -
val_loss: 179.4410
Epoch 5/30
1024/1024 [==============================] - 3s 3ms/step - loss: 168.1153 -
val_loss: 174.2208
Epoch 6/30
1024/1024 [==============================] - 3s 3ms/step - loss: 163.6044 -
val_loss: 177.6573
Epoch 7/30
```

```
1024/1024 [==============================] - 3s 3ms/step - loss: 160.5682 -
val_loss: 173.0528
Epoch 8/30
1024/1024 [==============================] - 3s 3ms/step - loss: 156.7438 -
val_loss: 169.8622
Epoch 9/30
1024/1024 [==============================] - 3s 3ms/step - loss: 154.6262 -
val_loss: 172.4182
Epoch 10/30
1024/1024 [==============================] - 3s 3ms/step - loss: 150.7208 -
val_loss: 175.8395
Epoch 11/30
1024/1024 [==============================] - 3s 3ms/step - loss: 148.1582 -
val_loss: 174.1887
Epoch 12/30
1024/1024 [==============================] - 3s 3ms/step - loss: 145.6746 -
val_loss: 175.6756
Epoch 13/30
1024/1024 [==============================] - 3s 3ms/step - loss: 144.2912 -
val_loss: 174.5040
Epoch 14/30
1024/1024 [==============================] - 3s 3ms/step - loss: 142.0843 -
val_loss: 180.4005
Epoch 15/30
1024/1024 [==============================] - 3s 3ms/step - loss: 139.9976 -
val_loss: 174.9471
Epoch 16/30
1024/1024 [==============================] - 3s 3ms/step - loss: 138.8402 -
val_loss: 178.5229
Epoch 17/30
1024/1024 [==============================] - 3s 3ms/step - loss: 136.9113 -
val_loss: 179.2263
Epoch 18/30
1024/1024 [==============================] - 3s 3ms/step - loss: 136.3520 -
val_loss: 179.6938
Epoch 19/30
1024/1024 [==============================] - 3s 3ms/step - loss: 134.4993 -
val_loss: 187.1263
Epoch 20/30
1024/1024 [==============================] - 3s 3ms/step - loss: 134.2550 -
val_loss: 177.3005
Epoch 21/30
1024/1024 [==============================] - 3s 3ms/step - loss: 131.9135 -
val_loss: 176.1318
Epoch 22/30
1024/1024 [==============================] - 3s 3ms/step - loss: 131.6325 -
val_loss: 180.5103
Epoch 23/30
```

```
1024/1024 [==============================] - 3s 3ms/step - loss: 130.1081 -
val_loss: 179.7242
Epoch 24/30
1024/1024 [==============================] - 3s 3ms/step - loss: 129.3563 -
val_loss: 184.3981
Epoch 25/30
1024/1024 [==============================] - 3s 3ms/step - loss: 127.9787 -
val_loss: 181.8298
Epoch 26/30
1024/1024 [==============================] - 3s 3ms/step - loss: 128.9726 -
val_loss: 179.8844
Epoch 27/30
1024/1024 [==============================] - 3s 3ms/step - loss: 126.4702 -
val_loss: 194.5360
Epoch 28/30
1024/1024 [==============================] - 3s 3ms/step - loss: 126.2300 -
val_loss: 183.0592
Epoch 29/30
1024/1024 [==============================] - 3s 3ms/step - loss: 125.1947 -
val_loss: 181.0428
Epoch 30/30
1024/1024 [==============================] - 3s 3ms/step - loss: 123.8060 -
val_loss: 183.1246
```

[ ]:
```
# save the weights to the local file
if save_config:
  print('saving weights ....')
  cvae_one.save_weights(cvae_one_weight_filename)
  cvae_two.save_weights(cvae_two_weight_filename)
  print('weights saved')
```

[ ]:
```
# generate images using one-pipeline network
predict_one = cvae_one.predict([x_corr_test, x_real_test], batch_size=128)

# generate images using dual-pipeline network
predict_two = cvae_two.predict([x_corr_test, x_comp_test, x_real_test],␣
 ↪batch_size=128)
```

[ ]:
```
# display the results
# (left: original images, middle: one-pipeline network, right: dual-pipeline␣
 ↪network)
plt.figure(num='image', figsize=(10,10))

for i in range(1, 13, 3):
  idx = int((i-1)/3)
  offset = 16
  plt.subplot(4, 3, i)
```

```
  plt.title('original_{}'.format(idx))
  plt.xticks([])
  plt.yticks([])
  plt.imshow(x_real_test[idx + offset])

  plt.subplot(4, 3, i+1)
  plt.title('model_1_{}'.format(idx))
  plt.xticks([])
  plt.yticks([])
  plt.imshow(predict_one[idx + offset])

  plt.subplot(4, 3, i+2)
  plt.title('model_2_{}'.format(idx))
  plt.xticks([])
  plt.yticks([])
  plt.imshow(predict_two[idx + offset])

plt.show()
```

[ ]:
```
def getPSNR_SSIM(real_data, pred_data):
  '''
  get both psnr and ssim

  Parameters:
  real_data: original images
  pred_data: reconstructed images
  '''
  psnr = []
  ssim = []
```

```
  for i in range(min(len(real_data), len(pred_data))):
    pv = getPSNR(real_data[i], pred_data[i])
    sv = getSSIM(real_data[i], pred_data[i])
    psnr.append(pv)
    ssim.append(sv[0])

  return psnr, ssim
```

[ ]:
```
psnr_one, ssim_one = getPSNR_SSIM(x_real_test, predict_one)
psnr_two, ssim_two = getPSNR_SSIM(x_real_test, predict_two)


print('*-----------* Model One *--------------*')
print('Average of PSNR: ', avg(psnr_one))
print('Average of SSIM: ', avg(ssim_one))

print('*-----------* Model two *--------------*')
print('Average of PSNR: ', avg(psnr_two))
print('Average of SSIM: ', avg(ssim_two))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:9: UserWarning:
DEPRECATED: skimage.measure.compare_ssim has been moved to
skimage.metrics.structural_similarity. It will be removed from skimage.measure
in version 0.18.
  if __name__ == '__main__':
*-----------* Model One *--------------*
Average of PSNR:  20.92
Average of SSIM:  0.77
*-----------* Model two *--------------*
Average of PSNR:  21.12
Average of SSIM:  0.78
```

[ ]:
```
```

[ ]:
```
'''
  Part III: Prediction & Reconstruction
'''
```

[ ]:
```
def retrieve_eyes(eyes):
  '''
  get eyes position
  '''
  eyes_temp = []
  left_eye = None
  right_eye = None
  for (ex, ey, ew, eh) in eyes:
```

```
    if (ey + eh/2 < image_size/2):
      if (ex + ew/2 < image_size/2):
        left_eye = (ex, ey, ew, eh)
      else:
        right_eye = (ex, ey, ew, eh)

  if left_eye is not None: eyes_temp.append(left_eye)
  if right_eye is not None: eyes_temp.append(right_eye)

  return eyes_temp

def verify_eyes(eyes):
  '''
  verify eye's position
  '''
  if len(eyes) is not 2: return False

  for (ex, ey, ew, eh) in eyes:
    if (ey + eh/2 > 64): return False

  if (eyes[0][0] + eyes[0][2]/2 < eyes[1][0]): return True
  elif (eyes[1][0] + eyes[1][2]/2 < eyes[0][0]): return True
  return False

def get_mid_eyes(eyes):
  '''
  given eyes position, return the middle of the two eyes
  '''
  ex = eyes[0][0]
  ew = eyes[0][2]
  ex_ = eyes[1][0]
  ew_ = eyes[1][2]

  center = int((2 * (ex + ex_) + ew + ew_)/4)
  damp = abs(center - image_size/2)
  if (center > image_size/2):
    center_damp = int(image_size/2 + damp * 0.7)
  else:
    center_damp = int(image_size/2 - damp * 0.7)

  return center_damp

def get_face_center(img):
  '''
  given frontal face image, return the center of the face
  '''
  eyes = itool.getEyes(img)
```

```
  eyes = retrieve_eyes(eyes)
  if verify_eyes(eyes):
    return get_mid_eyes(eyes)
  else:
    return int(image_size/2)

def generate_random_hole_construct(img):
  '''
  create random hole and reconstruct it
  '''
  w = get_face_center(img)
  img_hole = itool.rgb_random_blank(img)
  img_comp, img_corr, img_comp_rate = itool.rgb_flip_construct(img_hole, w)
  return img_corr, img_comp, img_hole, 1 - img_comp_rate

def construct(img_corr, generate_img):
  img_comp_dist = itool.rgb_comp_img(img_corr, generate_img)
  img_complete = itool.rgb_overlap(img_comp_dist, img_corr)

  return img_complete


def construct_(img_corr, img_comp, generate_img):
  img_comp_dist = itool.rgb_comp_img(img_corr, generate_img)
  img_comp_dist = itool.rgb_overlap_weight(img_comp_dist, 0.95, img_comp, 0.05)
  img_complete = itool.rgb_overlap(img_comp_dist, img_corr)

  return img_complete
```

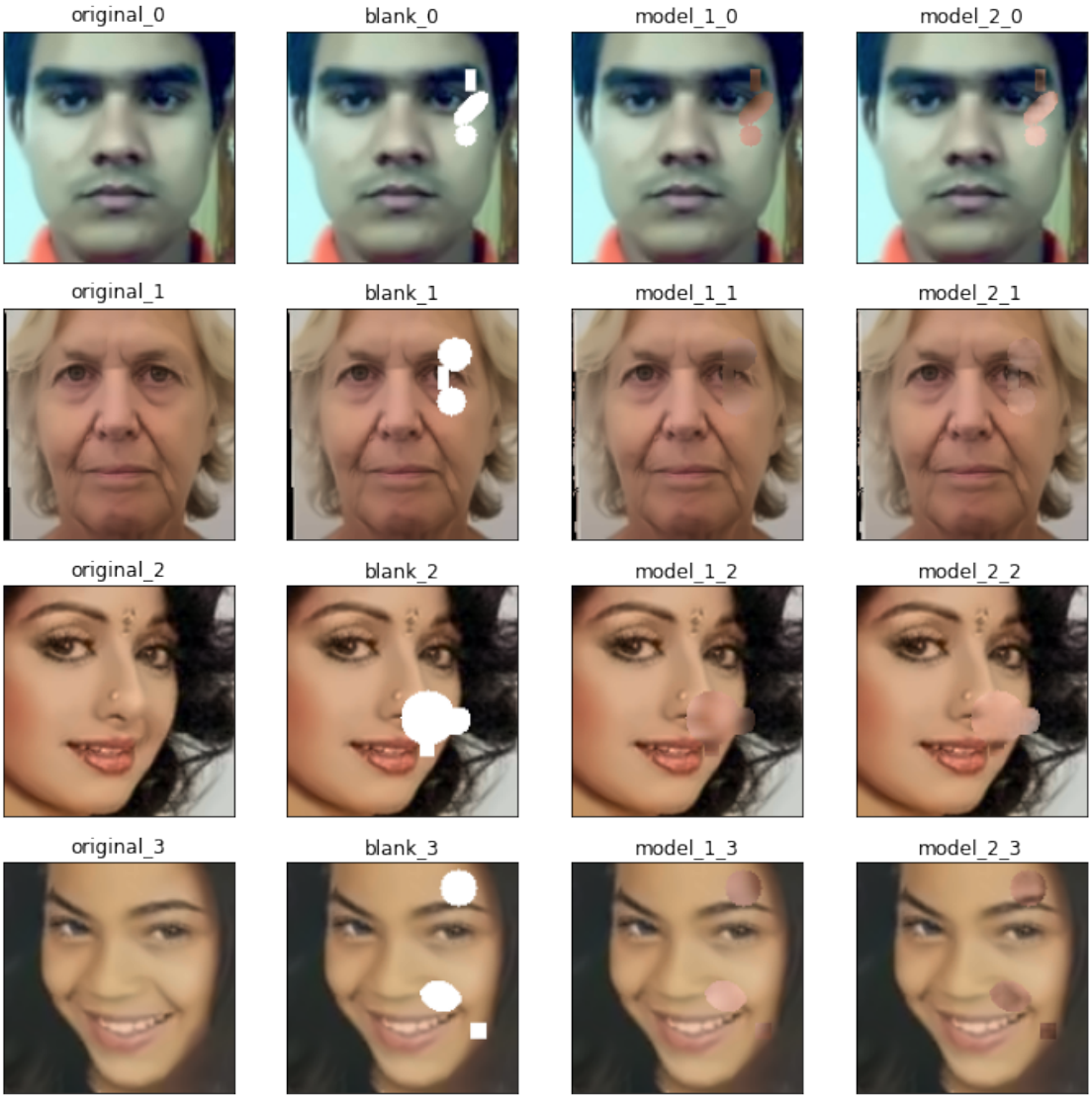
[ ]:
```
test_start = 80
test_end = 120

test_img = [generate_random_hole_construct(img) for img in␣
 ↪x_real_test[test_start: test_end]]
test_img_corr = [img[0] for img in test_img]
test_img_comp = [img[1] for img in test_img]
test_img_blank = [img[2] for img in test_img]
test_img_corr_rate = [img[3] for img in test_img]
test_img_real = x_real_test[test_start: test_end]

print(test_img_corr_rate)
```

```
[0.98, 0.97, 0.95, 0.96, 0.94, 0.97, 0.94, 0.95, 0.95, 0.97, 0.97, 0.97, 0.97,
0.9299999999999999, 0.94, 0.9299999999999999, 0.95, 0.95, 0.99, 0.95, 0.94,
0.97, 0.96, 0.92, 0.95, 0.95, 0.96, 0.98, 0.9299999999999999, 0.96, 0.92,
0.9299999999999999, 0.96, 0.98, 0.9, 0.96, 0.97, 0.96, 0.98, 0.9299999999999999]
```

[ ]:
```
cvae_one_generate = cvae_one.predict([test_img_corr, x_real_test],␣
 ↪batch_size=128)
cvae_one_predict = []
for i in range(len(cvae_one_generate)):
  img_pred = construct(test_img_corr[i], cvae_one_generate[i])
  cvae_one_predict.append(img_pred)
```

[ ]:
```
cvae_two_generate = cvae_two.predict([test_img_corr, test_img_comp,␣
 ↪x_real_test], batch_size=128)
cvae_two_predict = []
for i in range(len(cvae_two_generate)):
  img_pred = construct(test_img_corr[i], cvae_two_generate[i])
  cvae_two_predict.append(img_pred)
```

[ ]:
```
plt.figure(num='image_complete', figsize=(12,12))

for i in range(1, 17, 4):
  idx = int((i-1)/4)

  plt.subplot(4, 4, i)
  plt.title('original_{}'.format(idx))
  plt.xticks([])
  plt.yticks([])
  plt.imshow(test_img_real[idx])

  plt.subplot(4, 4, i+1)
  plt.title('blank_{}'.format(idx))
  plt.xticks([])
  plt.yticks([])
  plt.imshow(test_img_blank[idx])

  plt.subplot(4, 4, i+2)
  plt.title('model_1_{}'.format(idx))
  plt.xticks([])
  plt.yticks([])
  plt.imshow(cvae_one_predict[idx])

  plt.subplot(4, 4, i+3)
  plt.title('model_2_{}'.format(idx))
  plt.xticks([])
  plt.yticks([])
  plt.imshow(cvae_two_predict[idx])

plt.show()
```



[ ]:
```
psnr_one, ssim_one = getPSNR_SSIM(x_real_test, predict_one)
psnr_two, ssim_two = getPSNR_SSIM(x_real_test, predict_two)


print('*-----------* Model One *--------------*')
print('Average of PSNR: ', avg(psnr_one))
print('Average of SSIM: ', avg(ssim_one))

print('*-----------* Model two *--------------*')
print('Average of PSNR: ', avg(psnr_two))
print('Average of SSIM: ', avg(ssim_two))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:25: UserWarning:
```

```
DEPRECATED: skimage.measure.compare_ssim has been moved to
skimage.metrics.structural_similarity. It will be removed from skimage.measure
in version 0.18.

*-----------* Model One *--------------*
Average of PSNR:  18.06
Average of SSIM:  0.69
*-----------* Model two *--------------*
Average of PSNR:  17.6
Average of SSIM:  0.68
```

[ ]:

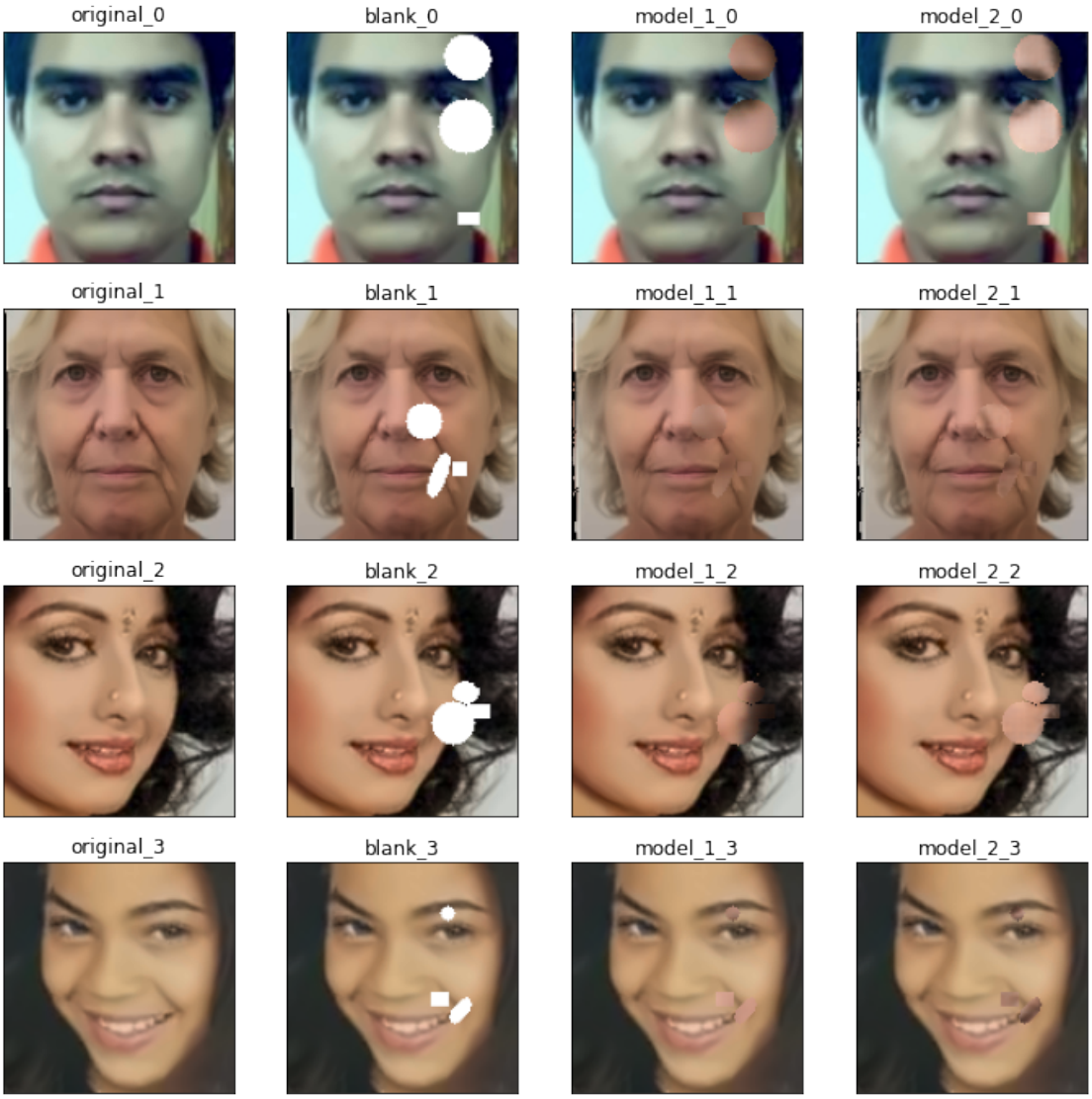
[ ]:
```
test_start = 80
test_end = 120

test_img = [generate_random_hole_construct(img) for img in␣
 ↪x_real_test[test_start: test_end]]
test_img_corr = [img[0] for img in test_img]
test_img_comp = [img[1] for img in test_img]
test_img_blank = [img[2] for img in test_img]
test_img_corr_rate = [img[3] for img in test_img]
test_img_real = x_real_test[test_start: test_end]
```

[ ]:
```
cvae_one_generate = cvae_one.predict([test_img_corr, x_real_test],␣
 ↪batch_size=128)
cvae_one_predict = []
for i in range(len(cvae_one_generate)):
  img_pred = construct(test_img_corr[i], cvae_one_generate[i])
  cvae_one_predict.append(img_pred)
```

[ ]:
```
cvae_two_generate = cvae_two.predict([test_img_corr, test_img_comp,␣
 ↪x_real_test], batch_size=128)
cvae_two_predict = []
for i in range(len(cvae_two_generate)):
  img_pred = construct_(test_img_corr[i], test_img_comp[i], cvae_two_generate[i])
  cvae_two_predict.append(img_pred)
```

[ ]:
```
plt.figure(num='image_complete', figsize=(12,12))

for i in range(1, 17, 4):
  idx = int((i-1)/4)

  plt.subplot(4, 4, i)
  plt.title('original_{}'.format(idx))
  plt.xticks([])
  plt.yticks([])
```

```
  plt.imshow(test_img_real[idx])

  plt.subplot(4, 4, i+1)
  plt.title('blank_{}'.format(idx))
  plt.xticks([])
  plt.yticks([])
  plt.imshow(test_img_blank[idx])

  plt.subplot(4, 4, i+2)
  plt.title('model_1_{}'.format(idx))
  plt.xticks([])
  plt.yticks([])
  plt.imshow(cvae_one_predict[idx])

  plt.subplot(4, 4, i+3)
  plt.title('model_2_{}'.format(idx))
  plt.xticks([])
  plt.yticks([])
  plt.imshow(cvae_two_predict[idx])

plt.show()
```



[ ]:
```
psnr_one, ssim_one = getPSNR_SSIM(x_real_test, predict_one)
psnr_two, ssim_two = getPSNR_SSIM(x_real_test, predict_two)


print('*-----------* Model One *--------------*')
print('Average of PSNR: ', avg(psnr_one))
print('Average of SSIM: ', avg(ssim_one))

print('*-----------* Model two *--------------*')
print('Average of PSNR: ', avg(psnr_two))
print('Average of SSIM: ', avg(ssim_two))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:9: UserWarning:
```

```
DEPRECATED: skimage.measure.compare_ssim has been moved to
skimage.metrics.structural_similarity. It will be removed from skimage.measure
in version 0.18.
  if __name__ == '__main__':

*-----------* Model One *--------------*
Average of PSNR:  20.92
Average of SSIM:  0.77
*-----------* Model two *--------------*
Average of PSNR:  21.12
Average of SSIM:  0.78
```

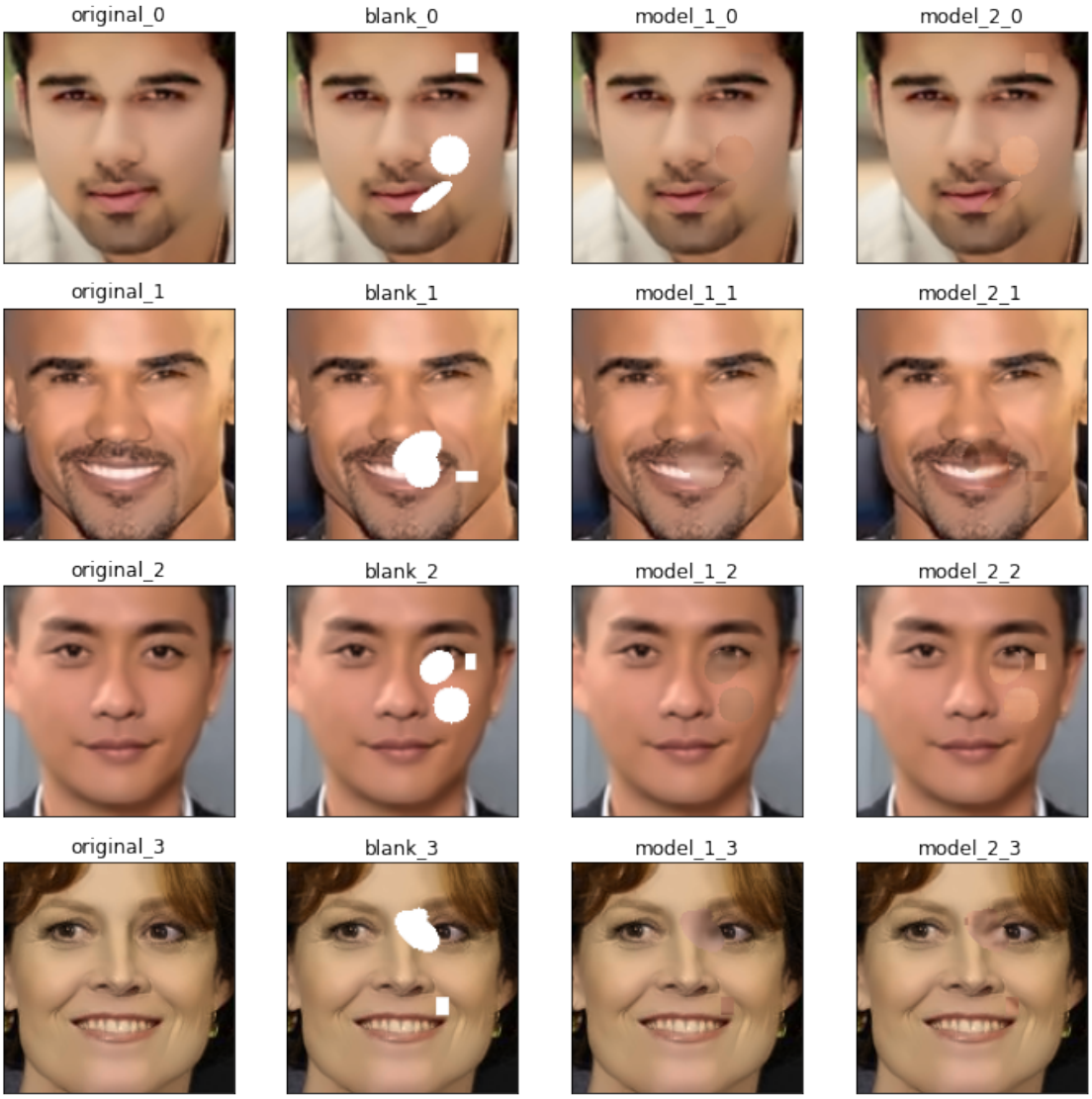
[ ]:

[ ]:
```
img_test = retrieve_test()
img_test = [itool.rgb_denoise(img) for img in img_test]
imgs = [generate_random_hole_construct(img) for img in img_test]
img_corr = [img[0] for img in imgs]
img_comp = [img[1] for img in imgs]
img_blank = [img[2] for img in imgs]
img_corr_rate = [img[3] for img in imgs]
img_real = img_test
```

[ ]:
```
cvae_one_generate = cvae_one.predict([img_corr, img_real], batch_size=128)
cvae_one_predict = []
for i in range(len(cvae_one_generate)):
  img_pred = construct(img_corr[i], cvae_one_generate[i])
  cvae_one_predict.append(img_pred)

cvae_two_generate = cvae_two.predict([img_corr, img_comp, img_real],␣
 ↪batch_size=128)
cvae_two_predict = []
for i in range(len(cvae_two_generate)):
  img_pred = construct_(img_corr[i], img_comp[i], cvae_two_generate[i])
  cvae_two_predict.append(img_pred)
```

[ ]:
```
plt.figure(num='image_complete', figsize=(12,12))

for i in range(1, 25, 4):
  idx = int((i-1)/4)

  plt.subplot(6, 4, i)
  plt.title('original_{}'.format(idx))
  plt.xticks([])
  plt.yticks([])
  plt.imshow(img_real[idx])

  plt.subplot(6, 4, i+1)
```

```
  plt.title('blank_{}'.format(idx))
  plt.xticks([])
  plt.yticks([])
  plt.imshow(img_blank[idx])

  plt.subplot(6, 4, i+2)
  plt.title('model_1_{}'.format(idx))
  plt.xticks([])
  plt.yticks([])
  plt.imshow(cvae_one_predict[idx])

  plt.subplot(6, 4, i+3)
  plt.title('model_2_{}'.format(idx))
  plt.xticks([])
  plt.yticks([])
  plt.imshow(cvae_two_predict[idx])

plt.show()
```



[ ]:
```
psnr_one, ssim_one = getPSNR_SSIM(img_real, cvae_one_predict)
psnr_two, ssim_two = getPSNR_SSIM(img_real, cvae_two_predict)


print('*-----------* Model One *--------------*')
print('Average of PSNR: ', avg(psnr_one))
print('Average of SSIM: ', avg(ssim_one))

print('*-----------* Model two *--------------*')
print('Average of PSNR: ', avg(psnr_two))
print('Average of SSIM: ', avg(ssim_two))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:9: UserWarning:
```

```
DEPRECATED: skimage.measure.compare_ssim has been moved to
skimage.metrics.structural_similarity. It will be removed from skimage.measure
in version 0.18.
  if __name__ == '__main__':
```

```
*-----------* Model One *--------------*
Average of PSNR:  35.09
Average of SSIM:  0.98
*-----------* Model two *--------------*
Average of PSNR:  31.55
Average of SSIM:  0.98
```

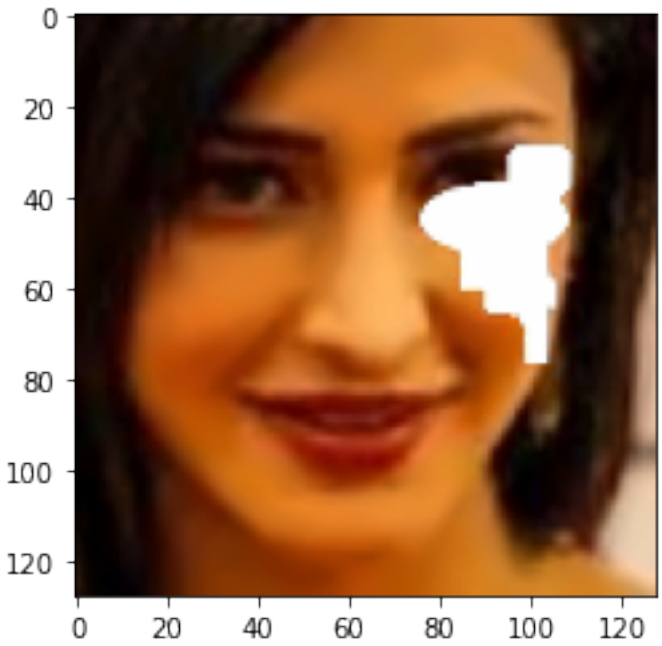
[ ]:

[ ]:
```
'''
  Part IV: Demo
'''
```

[ ]:
```
images_demo = get_image(filename_demo)
images_demo = [itool.rgb_denoise(img) for img in images_demo]
```

[ ]:
```
plt.imshow(images_demo[0])
```

[ ]: <matplotlib.image.AxesImage at 0x7f041e0a6860>



[ ]:
```
for img in images_demo:
  for i in range(128):
    for j in range(128):
      temp = img[i][j]
      if temp[0] > 0.90 and temp[1] > 0.90 and temp[2] > 0.90:
        img[i][j] = [0.998, 0.998, 0.998]
```

[ ]:
```
def construct_demo(images):
  img_corrs = []
  img_comps = []
  images_construct = []

  for img in images:
    w = get_face_center(img)
    img_comp, img_corr, img_comp_rate = itool.rgb_flip_construct(img, w)
    img_corrs.append(img_corr)
    img_comps.append(img_comp)
    images_construct.append(itool.rgb_overlap(img_comp, img_corr))

  cvae_two_generate = cvae_one.predict([img_corrs, images_construct],␣
 ↪batch_size=128)

  cvae_two_predict = []

  for i in range(len(cvae_two_generate)):
    img_pred = construct_(img_corrs[i], img_comps[i], cvae_two_generate[i])
    cvae_two_predict.append(img_pred)

  return cvae_two_predict
```

[ ]:
```
cvae_construct = construct_demo(images_demo)
```

[ ]:
```
plt.figure(num='image_complete_demo', figsize=(6,6))


plt.subplot(1, 2, 1)
plt.title('before_construct')
plt.xticks([])
plt.yticks([])
plt.imshow(images_demo[0])

plt.subplot(1, 2, 2)
plt.title('after_construct')
plt.xticks([])
plt.yticks([])
plt.imshow(cvae_construct[0])
```

```
plt.show()
```