Neural Style Transfer

- Created for Understanding NST (Neural Style Transfer) and Creating Art with Keras & Colab.
- Used for a coursework (717303) at Hallym Univ.

```
!pwd
```

```
/content
```

```
import os
#os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"
#os.environ["CUDA_VISIBLE_DEVICES"]="{}".format(3) # gpu idx
```

```
from __future__ import print_function
from keras.preprocessing.image import load_img, save_img, img_to_array
import numpy as np
from scipy.optimize import fmin_l_bfgs_b
import time
import argparse
import matplotlib.pyplot as plt
from keras.applications import vgg19
from keras import backend as K
```

```
Using TensorFlow backend.
```

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/co
```

```
!ls
```

```
drive  sample_data
```

- 아래와 같이 'AICapstone-2019Fall'을 만들었다고 가정합니다. (Google Drive)

```
!ls drive/'My Drive'/'Colab Notebooks'/AICapstone-2019Fall/images
```

```
20180829_115847.jpg  400x300
20180904_161416.jpg  Claude_Monet_Poster_Poppy_Field_at_Argenteuil_640x480.jpg
20181023_085206.jpg  monet_640x480.jpg
20181023_161555.jpg  van-gogh-starry-night.jpg
20181031_180133.jpg
```

```
basedir = "/content/drive/My Drive/Colab Notebooks/AICapstone-2019Fall/"
print(basedir)
!ls '$basedir'
```

```
outDir = basedir+'/out'

import os
if not os.path.exists(outDir):
    os.makedirs(outDir)
    print('created...{}'.format(outDir))

#!ls "$outDir/../"
```

더블클릭 또는 Enter 키를 눌러 수정

- 주의: 640x480 이미지로 변경하였습니다.

```
base_image_path = basedir+'images/20180904_161416.jpg'
#style_reference_image_path = basedir+'images/Claude_Monet_Poster_Poppy_Field_at_Argenteuil_640x480
style_reference_image_path = basedir+'images/monet_640x480.jpg'#gogh_starry

print(base_image_path)
print(style_reference_image_path )
```

설정

```
import datetime
str_time = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")

#result_prefix = 'generated'
#result_prefix = 'pic1'
result_prefix = 'pic_'+str_time
print(result_prefix)
```

```
iterations  = 100
```

```
content_weight = 0.025

style_weight = 1.0
#Total Variation weight.
total_variation_weight = 1.0
```

그림 불러와서 테스트

- 주의 h, w 순서

```
img1 = load_img(base_image_path)
print(img1.size)
```

```
print(img1.size)
img2 = load_img(style_reference_image_path, target_size=(480,640))
print(img2.size)
```

```
imgplot = plt.imshow(img1)
```

```
imgplot = plt.imshow(img2)
```

The weights of the different loss components

Dimensions of the generated picture

```
width, height = load_img(base_image_path).size
print(width, height)
```

```
img_nrows = height
img_ncols = width#int(width * img_nrows / height)
print(img_nrows, img_ncols)
```

```
print(img_nrows, img_ncols)
```

Util functions

```
#  to open, resize and format pictures into appropriate tensors
def preprocess_image(image_path):
    img = load_img(image_path, target_size=(img_nrows, img_ncols))
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = vgg19.preprocess_input(img)
    return img
```

```
def deprocess_image(x):
    if K.image_data_format() == 'channels_first':
        x = x.reshape((3, img_nrows, img_ncols))
        x = x.transpose((1, 2, 0))
    else:
        x = x.reshape((img_nrows, img_ncols, 3))
    # Remove zero-center by mean pixel
    x[:, :, 0] += 103.939
    x[:, :, 1] += 116.779
    x[:, :, 2] += 123.68
    # 'BGR'->'RGB'
    x = x[:, :, ::-1]
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

```
# get tensor representations of our images
base_image = K.variable(preprocess_image(base_image_path))
style_reference_image = K.variable(preprocess_image(style_reference_image_path))
```

```
K.image_data_format()
```

```
# this will contain our generated image
if K.image_data_format() == 'channels_first':
    combination_image = K.placeholder((1, 3, img_nrows, img_ncols))
else:
    combination_image = K.placeholder((1, img_nrows, img_ncols, 3))
```

```
# combine the 3 images into a single Keras tensor
input_tensor = K.concatenate([base_image,
                              style_reference_image,
                              combination_image], axis=0)
```

build the VGG19 network

```
# build the VGG19 network with our 3 images as input
# the model will be loaded with pre-trained ImageNet weights
model = vgg19.VGG19(input_tensor=input_tensor,
                    weights='imagenet', include_top=False)
print('Model loaded.')
```

```
# get the symbolic outputs of each "key" layer (we gave them unique names).
outputs_dict = dict([(layer.name, layer.output) for layer in model.layers])
```

Compute the neural style loss

```
# first we need to define 4 util functions
# the gram matrix of an image tensor (feature-wise outer product)

def gram_matrix(x):
    assert K.ndim(x) == 3
    if K.image_data_format() == 'channels_first':
        features = K.batch_flatten(x)
    else:
        features = K.batch_flatten(K.permute_dimensions(x, (2, 0, 1)))
    gram = K.dot(features, K.transpose(features))
    return gram
```

```
# the "style loss" is designed to maintain
# the style of the reference image in the generated image.
# It is based on the gram matrices (which capture style) of
# feature maps from the style reference image
# and from the generated image
def style_loss(style, combination):
    assert K.ndim(style) == 3
    assert K.ndim(combination) == 3
    S = gram_matrix(style)
    C = gram_matrix(combination)
    channels = 3
    size = img_nrows * img_ncols
    return K.sum(K.square(S - C)) / (4.0 * (channels ** 2) * (size ** 2))
```

```
# an auxiliary loss function
```

```
# designed to maintain the "content" of the
# base image in the generated image


def content_loss(base, combination):
    return K.sum(K.square(combination - base))
```

```
# the 3rd loss function, total variation loss,
# designed to keep the generated image locally coherent

def total_variation_loss(x):
    assert K.ndim(x) == 4
    if K.image_data_format() == 'channels_first':
        a = K.square(
            x[:, :, :img_nrows - 1, :img_ncols - 1] - x[:, :, 1:, :img_ncols - 1])
        b = K.square(
            x[:, :, :img_nrows - 1, :img_ncols - 1] - x[:, :, :img_nrows - 1, 1:])
    else:
        a = K.square(
            x[:, :img_nrows - 1, :img_ncols - 1, :] - x[:, 1:, :img_ncols - 1, :])
        b = K.square(
            x[:, :img_nrows - 1, :img_ncols - 1, :] - x[:, :img_nrows - 1, 1:, :])
    return K.sum(K.pow(a + b, 1.25))
```

Combine these loss functions into a single scalar

```
loss = K.variable(0.0)
layer_features = outputs_dict['block5_conv2']
base_image_features = layer_features[0, :, :, :]
combination_features = layer_features[2, :, :, :]
loss = loss + content_weight * content_loss(base_image_features,
                                            combination_features)
```

```
feature_layers = ['block1_conv1', 'block2_conv1',
                  'block3_conv1', 'block4_conv1',
                  'block5_conv1']
```

```
for layer_name in feature_layers:
    layer_features = outputs_dict[layer_name]
    style_reference_features = layer_features[1, :, :, :]
    combination_features = layer_features[2, :, :, :]
    sl = style_loss(style_reference_features, combination_features)
    loss = loss + (style_weight / len(feature_layers)) * sl
```

```
loss = loss + total_variation_weight * total_variation_loss(combination_image)
```

```
# get the gradients of the generated image wrt the loss
grads = K.gradients(loss, combination_image)
```

```
outputs = [loss]
if isinstance(grads, (list, tuple)):
    outputs += grads
else:
    outputs.append(grads)
```

```
f_outputs = K.function([combination_image], outputs)
```

```
def eval_loss_and_grads(x):
    if K.image_data_format() == 'channels_first':
        x = x.reshape((1, 3, img_nrows, img_ncols))
    else:
        x = x.reshape((1, img_nrows, img_ncols, 3))
    outs = f_outputs([x])
    loss_value = outs[0]
    if len(outs[1:]) == 1:
        grad_values = outs[1].flatten().astype('float64')
    else:
        grad_values = np.array(outs[1:]).flatten().astype('float64')
    return loss_value, grad_values
```

```
# this Evaluator class makes it possible
# to compute loss and gradients in one pass
# while retrieving them via two separate functions,
# "loss" and "grads". This is done because scipy.optimize
# requires separate functions for loss and gradients,
# but computing them separately would be inefficient.


class Evaluator(object):

    def __init__(self):
        self.loss_value = None
        self.grads_values = None

    def loss(self, x):
        assert self.loss_value is None
        loss_value, grad_values = eval_loss_and_grads(x)
        self.loss_value = loss_value
        self.grad_values = grad_values
        return self.loss_value

    def grads(self, x):
        assert self.loss_value is not None
        grad_values = np.copy(self.grad_values)
        self.loss_value = None
        self.grad_values = None
        return grad_values


evaluator = Evaluator()
```

```
evaluator = Evaluator()
```

```
print(outDir)
```

```
!ls "$outDir"
```

```
# run scipy-based optimization (L-BFGS) over the pixels of the generated image
# so as to minimize the neural style loss
x = preprocess_image(base_image_path)

for i in range(iterations):
    print('Start of iteration', i)
    start_time = time.time()
    x, min_val, info = fmin_l_bfgs_b(evaluator.loss, x.flatten(),
                                     fprime=evaluator.grads, maxfun=20)
    print('Current loss value:', min_val)
    # save current generated image
    img = deprocess_image(x.copy())
    fname = outDir + '/' + result_prefix + '_at_iteration_%d.png' % i
    #fname = '/content/drive/' + result_prefix + '_at_iteration_%d.png' % i

    #!ls "/content/drive/My Drive"

    #fname = '/content/drive/My Drive/Colab Notebooks/out/'+ result_prefix + '_at_iteration_%d.png'

    print(fname)
    save_img(fname, img)
    end_time = time.time()
    print('Image saved as', fname)
    print('Iteration %d completed in %ds' % (i, end_time - start_time))
```

```
!ls "$outDir"
```

```
fnameout = outDir + '/' + result_prefix + '_at_iteration_%d.png' % (iterations-1)
imgout = load_img(fnameout, target_size=(300,400))
imgplot = plt.imshow(imgout)
print(fnameout)
```

References

[1] Neural style transfer with Keras. https://keras.io/examples/neural_style_transfer/