Fast Style Transfer with Pytorch

Introduction

Implementation is based off of a combination of Gatys' A Neural Algorithm of Artistic Style, Johnson's Perceptual Losses for Real-Time Style Transfer and Super-Resolution, and Ulyanov's Instance Normalization.

Dependencies

Pytorch 1.7.1
Torchvision 0.8.2
OpenCV 4.1.0
Numpy 1.16.3
Matplotlib 3.0.3

Style Images

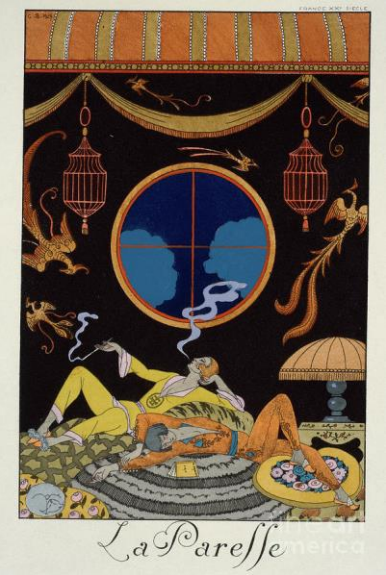
These are images used to extract the styles. If you wish to have more style, you could train the network with new image to obtain new style. Detail of training process is described in training session.

Udnie

Starry night

Bayanihan painting

Mosaic



The laziness

Tokyo ghoul – illustrate anime style

Get you images styled

stylize.py: This file contains function called **stylize()** to transform style of one image to another.

Input:

- **image** (array): The image need to be styled as np.array or cv2 format
- **style** (string): string represent style from trained model
  There are 5 style you passed in style
  -> bayanihan, lazy, mosaic, starry, tokyo_ghoul, udnie, wave
- **preserve_color** (boolean): if True, keep the color distribution of the original image
  if False, use the color distribution of given style
- **output_height** (int): desired height of output image
- **output_width** (int): desired width of output image
-

Note that output_height and output_width also used to resize the input image
if both value are None, keep shape of the input image
if output_height is None, use output_width and infer output_height from the original h/w ratio
if output_width is None, use output_height and infer output_width from the original h/w ratio
if both value are not None, use output_width and infer output_height from the original h/w ratio

Output:
styled_image (array) styled image

Training Style Transformation Network

train.py: trains the transformation network that learns the style of the style image. Each model in transforms folder was trained for roughly 23 minutes, with single pass (1 epoch) of 40,000 training images, and a batch size of 4, on a GTX 1080 Ti.

python train.py

Options

- TRAIN_IMAGE_SIZE: sets the dimension (height and weight) of training images. Bigger GPU memory is needed to train with larger images. Default is 256px.
- DATASET_PATH: folder containing the MS-COCO train2014 images. Default is "dataset"
- NUM_EPOCHS: Number of epochs of training pass. Default is 1 with 40,000 training images
- STYLE_IMAGE_PATH: path of the style image
- BATCH_SIZE: training batch size. Default is 4
- CONTENT_WEIGHT: Multiplier weight of the loss between content representations and the generated image. Default is 8
- STYLE_WEIGHT: Multiplier weight of the loss between style representations and the generated image. Default is 50
- ADAM_LR: learning rate of the adam optimizer. Default is 0.001

- SAVE_MODEL_PATH: path of pretrained-model weights and transformation network checkpoint files. Default is "models/"
- SAVE_IMAGE_PATH: save path of sample tranformed training images. Default is "images/out/"
- SAVE_MODEL_EVERY: Frequency of saving of checkpoint and sample transformed images. 1 iteration is defined as 1 batch pass. Default is 500 with batch size of 4, that is 2,000 images
- SEED: Random seed to keep the training variations as little as possible

Tuning the network

transformer.py: contains the architecture definition of the trasnformation network. It includes 2 models, TransformerNetwork() and TransformerNetworkTanh(). TransformerNetwork doesn't have an extra output layer, while TransformerNetworkTanh, as the name implies, has for its output, a Tanh layer and a default output multiplier of 150. TransformerNetwork faithfully copies the style and colorization of the style image, while Tanh model produces images with darker color; which brings a **retro style effect**.

Options

- norm: sets the normalization layer to either Instance Normalization "instance" or Batch Normalization "batch". Default is "instance"
- tanh_multiplier: output multiplier of the Tanh model. The bigger the number, the bright the image. Default is 150