Applications of Artificial Intelligence

EAI 6010, CRN 70749

Professor Vladimir Shapiro

Module 2: Assignment Week 2 - Distributed AI Applications

Submitted By - Richa Umesh Rambhia

Distributed AI Applications - Style Transfer for Arbitrary Styles

---

Table of Contents

Introduction

Style transfer for Arbitrary styles is the technique under **computer vision** which helps to **recompose the original image based on the style image** of another. For example, any content image that you upload will be recomposed based on the style image you want it to be, and thus it blends the two images resulting to a desired output. [1]

With the improvement and advancement of AI technology, style transfer is possible which is an application in the designing and content creation field.

The working of style transfer is based on the various **deep learning methods** and algorithms. In this assignment, the **TensorFlow Hub** [2] is been used which has a **pretrained model** for the working of the *Fast style transfer for arbitrary styles*. Considering the working of the style models, the algorithm takes two images, content image and style image which is the input to the model and based on the input images the style transfer is obtained and thus the model is able to extract the style that is given as the input and apply it to the image for desired output image.

The task is here is to perform **style transfer on various images** by changing the content and style images using the TensorFlow Hub which has predefined models that would help in the arbitrary style transfer process. The various functions defined for the same are *crop_center, load_image, show_n, load_local_image* which perform the various tasks that are required for the style transfer.

Analysis

This assignment helps to transfer styles from one image to another with the help of predefined models using TensorFlow Hub. The style transfer algorithm is beneficial in ways for the various applications of images and content creation or designing.

At first, the libraries are imported and the TensorFlow Hub module is loaded and various functions for the implementation of the style transfer is executed. The algorithm is executed for various images of arbitrary style transfer.

In [ ]:
```
# Copyright 2019 The TensorFlow Hub Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ==============================================================================
```

Fast Style Transfer for Arbitrary Styles

Based on the model code in magenta (https://github.com/tensorflow/magenta/tree/master/magenta/models/arbitrary_image_stylization) and the publication:

Exploring the structure of a real-time, arbitrary neural artistic stylization network (https://arxiv.org/abs/1705.06830). *Golnaz Ghiasi, Honglak Lee, Manjunath Kudlur, Vincent Dumoulin, Jonathon Shlens*, Proceedings of the British Machine Vision Conference (BMVC), 2017.

== Additions and modifications by Vladimir Shapiro

Setup

Let's start with importing TF-2 and all relevant dependencies.

In [ ]:
```
import functools
import os

from matplotlib import gridspec
import matplotlib.pylab as plt
import numpy as np
import tensorflow as tf
import tensorflow_hub as hub

print("TF Version: ", tf.__version__)
print("TF-Hub version: ", hub.__version__)
print("Eager mode enabled: ", tf.executing_eagerly())
print("GPU available: ", tf.test.is_gpu_available())
```

```
WARNING:tensorflow:From <ipython-input-1-1b40332be70f>:13: is_gpu_available (from tensorflow.python.framework.
test_util) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.

TF Version:  2.9.2
TF-Hub version:  0.12.0
Eager mode enabled:  True
GPU available:  True
```

In [ ]:
```
# @title Define image loading and visualization functions  { display-mode: "form" }

def crop_center(image):
  """Returns a cropped square image."""
  shape = image.shape
  new_shape = min(shape[1], shape[2])
  offset_y = max(shape[1] - shape[2], 0) // 2
  offset_x = max(shape[2] - shape[1], 0) // 2
  image = tf.image.crop_to_bounding_box(
      image, offset_y, offset_x, new_shape, new_shape)
  return image

@functools.lru_cache(maxsize=None)
def load_image(image_url, image_size=(256, 256), preserve_aspect_ratio=True):
  """Loads and preprocesses images."""
  # Cache image file locally.
  image_path = tf.keras.utils.get_file(os.path.basename(image_url)[-128:], image_url)
  # Load and convert to float32 numpy array, add batch dimension, and normalize to range [0, 1].
  img = plt.imread(image_path).astype(np.float32)[np.newaxis, ...]
  if img.max() > 1.0:
    img = img / 255.
  if len(img.shape) == 3:
    img = tf.stack([img, img, img], axis=-1)
  img = crop_center(img)
  img = tf.image.resize(img, image_size, preserve_aspect_ratio=True)
  return img

def show_n(images, titles=('',)):
  n = len(images)
  image_sizes = [image.shape[1] for image in images]
  w = (image_sizes[0] * 6) // 320
  plt.figure(figsize=(w  * n, w))
  gs = gridspec.GridSpec(1, n, width_ratios=image_sizes)
  for i in range(n):
    plt.subplot(gs[i])
    plt.imshow(images[i][0], aspect='equal')
    plt.axis('off')
    plt.title(titles[i] if len(titles) > i else '')
  plt.show()

def load_local_image(image_bytes, image_size=(256, 256), preserve_aspect_ratio=True):
  """Loads and preprocesses images."""
   # Load and convert to float32 numpy array, add batch dimension, and normalize to range [0, 1].
  img = image_bytes
  if img.max() > 1.0:
    img = img / 255.
  if len(img.shape) == 3:
    img = tf.stack([img, img, img], axis=-1)
  img = crop_center(img)
  img = tf.image.resize(img, image_size, preserve_aspect_ratio=True)
  return img
```

Import TF-Hub module

In [ ]:
```
# Load TF-Hub module.

hub_handle = 'https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2'
hub_module = hub.load(hub_handle)
```

First, try the original (from the Tensorflow source) files:

In [ ]:
```
# @title Load example images  { display-mode: "form" }

content_image_url = 'https://upload.wikimedia.org/wikipedia/commons/thumb/f/fd/Golden_Gate_Bridge_from_Battery_S
style_image_url = 'https://upload.wikimedia.org/wikipedia/commons/0/0a/The_Great_Wave_off_Kanagawa.jpg'  # @para
output_image_size = 384  # @param {type:"integer"}

# The content image size can be arbitrary.
content_img_size = (output_image_size, output_image_size)
# The style prediction model was trained with image size 256 and it's the
# recommended image size for the style image (though, other sizes work as
# well but will lead to different results).
style_img_size = (256, 256)  # Recommended to keep it at 256.

content_image = load_image(content_image_url, content_img_size)
style_image = load_image(style_image_url, style_img_size)
style_image = tf.nn.avg_pool(style_image, ksize=[3,3], strides=[1,1], padding='SAME')
show_n([content_image, style_image], ['Content image', 'Style image'])
```

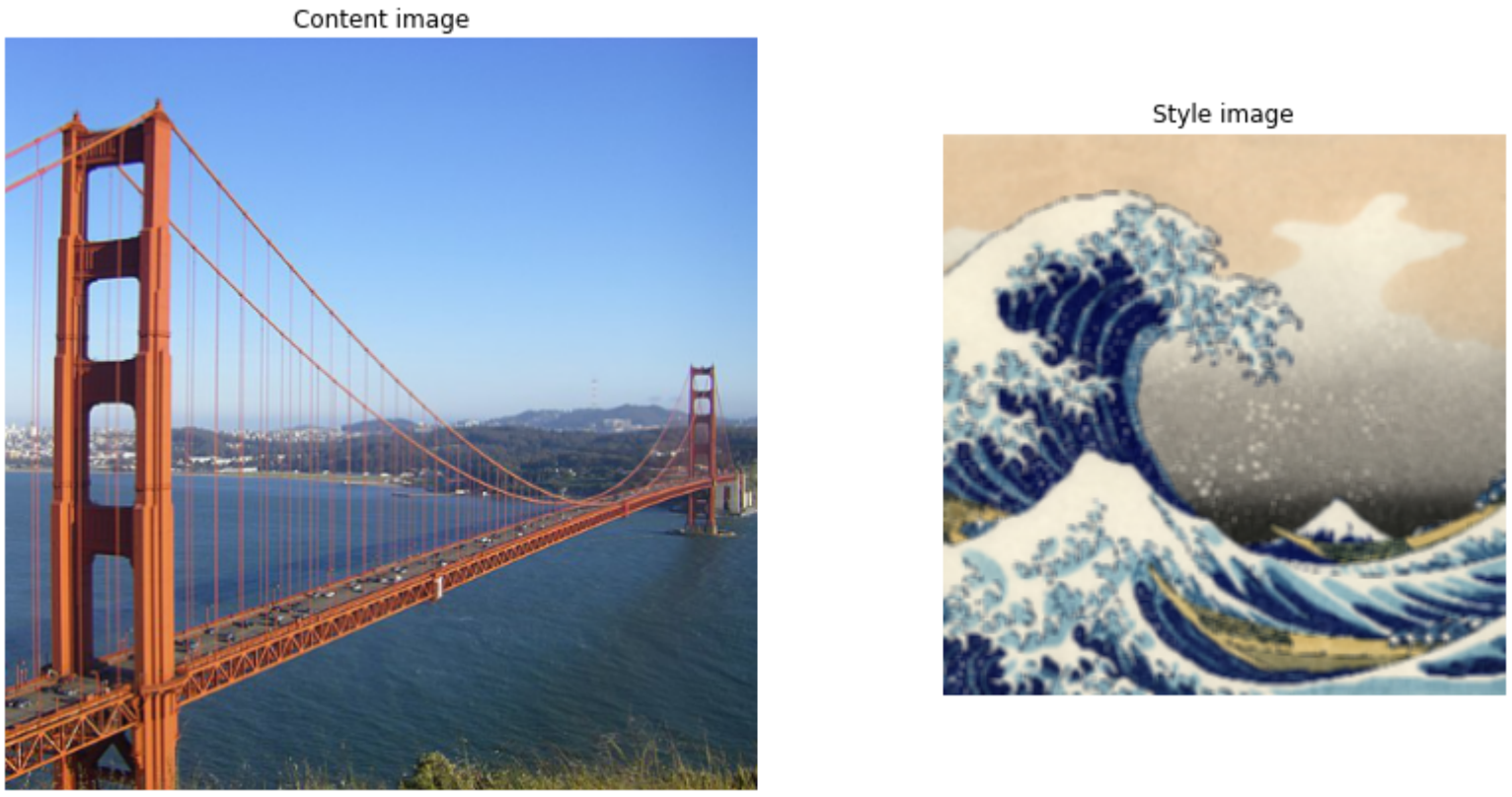

Figure 1. Loading content & style images

Demonstrate image stylization

The signature of this hub module for image stylization is:

```
outputs = hub_module(content_image, style_image)
stylized_image = outputs[0]
```

Where `content_image`, `style_image`, and `stylized_image` are expected to be 4-D Tensors with shapes `[batch_size, image_height, image_width, 3]`.

In the current example we provide only single images and therefore the batch dimension is 1, but one can use the same module to process more images at the same time.

The input and output values of the images should be in the range [0, 1].

The shapes of content and style image don't have to match. Output image shape is the same as the content image shape.

Calculate and Display

In [ ]:
```
# Stylize content image with given style image.
# This is pretty fast within a few milliseconds on a GPU.

outputs = hub_module(tf.constant(content_image), tf.constant(style_image))
stylized_image = outputs[0]

# Visualize input images and the generated stylized image.

show_n([content_image, style_image, stylized_image], titles=['Original content image', 'Style image', 'Stylized
```

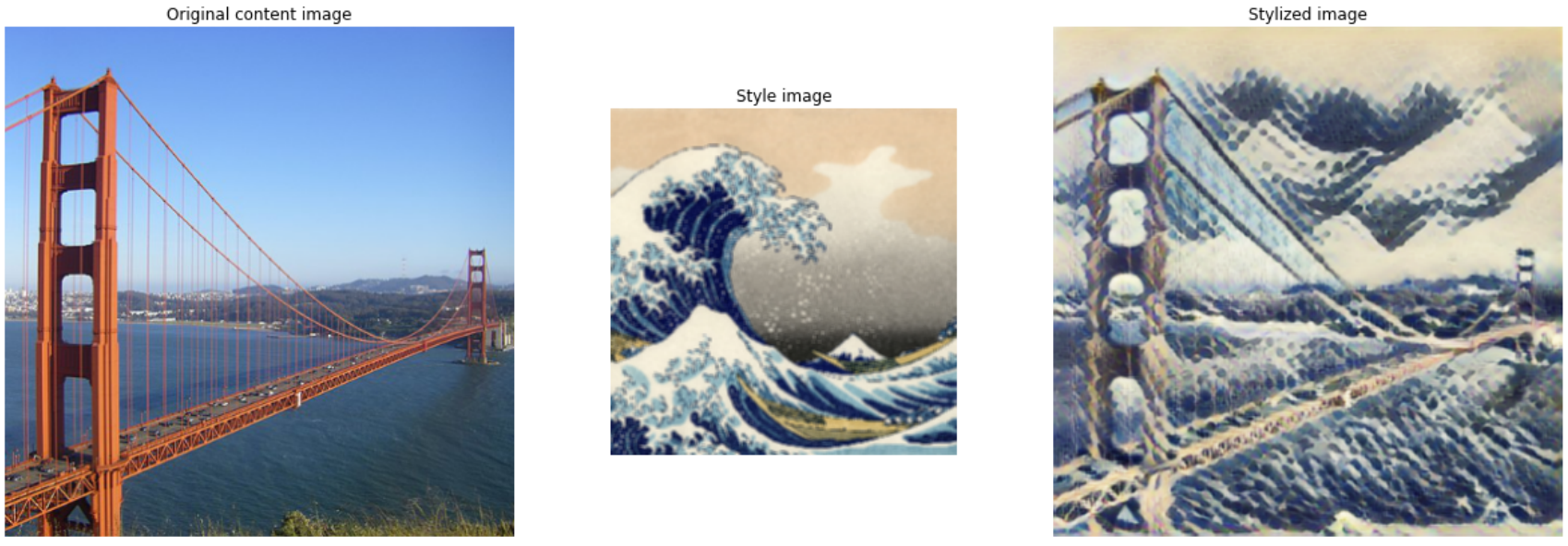

Figure 2. Style Transfer 1 - Golden Gate & Style Image Transfer

Let's get as well some images to play with.

In [ ]:
```
from google.colab import files
# load style image from the local filesystem
print("Browse to the style file  to be uploaded")
style_uploaded = files.upload()
for fn in style_uploaded.keys():
  print("User uploaded file {name} with length {length} fbytes".format(name=fn,
                                                       length=len(style_uploaded[fn])))
style_img = plt.imread(fn).astype(np.float32)[np.newaxis, ...]

# load content image from the local filesystem
print("Browse to the content file to be uploaded")
content_uploaded = files.upload()
for fn in content_uploaded.keys():
  print("User uploaded content file {name} with length {length} fbytes".format(name=fn,
                                                       length=len(content_uploaded[fn])))
content_img = plt.imread(fn).astype(np.float32)[np.newaxis, ...]

# @title Load example images  { display-mode: "form" }
output_image_size = 500  # @param {type:"integer"}

# The content image size can be arbitrary.
content_img_size = (output_image_size, output_image_size)
# The style prediction model was trained with image size 256 and it's the
# recommended image size for the style image (though, other sizes work as
# well but will lead to different results).
style_img_size = (256, 256)  # Recommended to keep it at 256.

content_image = load_local_image(content_img, content_img_size) #load_image(content_image_url, content_img_size)
style_image = load_local_image(style_img, style_img_size)
style_image = tf.nn.avg_pool(style_image, ksize=[3,3], strides=[1,1], padding='SAME')
#show_n([content_image, style_image], ['Content image', 'Style image'])

# Stylize content image with given style image.
# This is pretty fast within a few milliseconds on a GPU.
outputs = hub_module(tf.constant(content_image), tf.constant(style_image))
stylized_image = outputs[0]

# Visualize input images and the generated stylized image.
show_n([content_image, style_image, stylized_image], titles=['Original content image', 'Style image', 'Stylized
```

```
Browse to the style file  to be uploaded
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving NortheasternLogo_256x256.jpg to NortheasternLogo_256x256 (2).jpg
User uploaded file NortheasternLogo_256x256.jpg with length 24546 fbytes
Browse to the content file to be uploaded
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Doraemon-Story-of-Seasons-Nobita-Doraemon-DualShockers-Feature.jpg to Doraemon-Story-of-Seasons-Nobita-
Doraemon-DualShockers-Feature.jpg
User uploaded content file Doraemon-Story-of-Seasons-Nobita-Doraemon-DualShockers-Feature.jpg with length 2214
47 fbytes
```

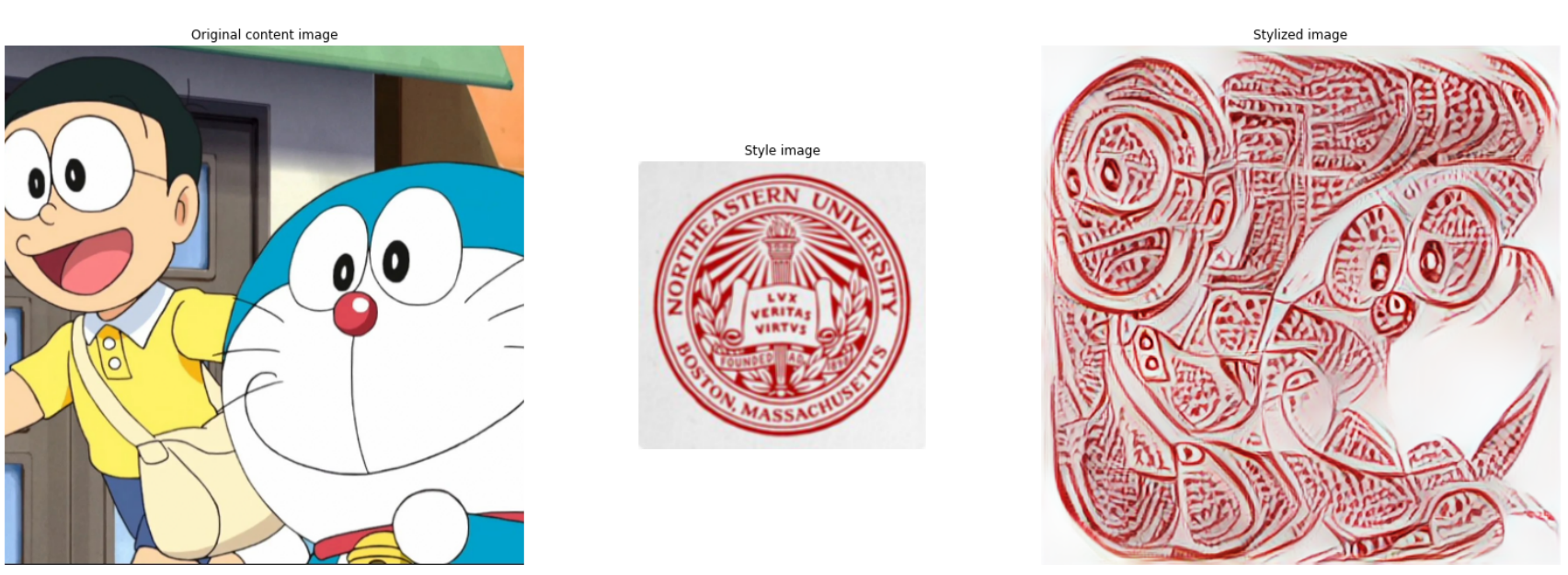

Figure 3. Style Transfer 2 - Cartoon character & Northeastern University Logo Image Transfer

Load Example Image - 2

Considering the same logic and algorithm to style various other images with respect to the content image and the style image is shown in another example as below.

In [ ]:
```
from google.colab import files
# load style image from the local filesystem
print("Browse to the style file  to be uploaded")
style_uploaded = files.upload()
for fn in style_uploaded.keys():
  print("User uploaded file {name} with length {length} fbytes".format(name=fn,
                                                        length=len(style_uploaded[fn])))
style_img = plt.imread(fn).astype(np.float32)[np.newaxis, ...]

# load content image from the local filesystem
print("Browse to the content file to be uploaded")
content_uploaded = files.upload()
for fn in content_uploaded.keys():
  print("User uploaded content file {name} with length {length} fbytes".format(name=fn,
                                                        length=len(content_uploaded[fn])))
content_img = plt.imread(fn).astype(np.float32)[np.newaxis, ...]

# @title Load example images  { display-mode: "form" }
output_image_size = 500  # @param {type:"integer"}

# The content image size can be arbitrary.
content_img_size = (output_image_size, output_image_size)
# The style prediction model was trained with image size 256 and it's the
# recommended image size for the style image (though, other sizes work as
# well but will lead to different results).
style_img_size = (256, 256)  # Recommended to keep it at 256.

content_image = load_local_image(content_img, content_img_size) #load_image(content_image_url, content_img_size)
style_image = load_local_image(style_img, style_img_size)
style_image = tf.nn.avg_pool(style_image, ksize=[3,3], strides=[1,1], padding='SAME')
#show_n([content_image, style_image], ['Content image', 'Style image'])

# Stylize content image with given style image.
# This is pretty fast within a few milliseconds on a GPU.
outputs = hub_module(tf.constant(content_image), tf.constant(style_image))
stylized_image = outputs[0]

# Visualize input images and the generated stylized image.
show_n([content_image, style_image, stylized_image], titles=['Original content image', 'Style image', 'Stylized
```

```
Browse to the style file  to be uploaded
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Avengers.jpg to Avengers (1).jpg
User uploaded file Avengers.jpg with length 112917 fbytes
Browse to the content file to be uploaded
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Robert Jr Downey.jpg to Robert Jr Downey (2).jpg
User uploaded content file Robert Jr Downey.jpg with length 116005 fbytes
```

Figure 4. Style Transfer 3 - Robert Downey Jr. & Avengers Logo Image Transfer

Thus, the images are styled based on the input images, i.e., the content image and the style image. The style transfer for arbitrary styles is implemented for various content images over the style images.

The TensorFlow Hub thus helped in the implementation of the style transfer which has a predefined model that helps in the rapid implementation of the same.

Conclusion

Style Transfer is one of the well-known techniques used in the various applications of designing and marketing. TensorFlow Hub allows us to implement the style transfer mechanism with the help of its prededined models and create image based on the different styles for the original image.

This technology of computer vision has impacted many industries especially the art and entertainment field, where the same application can be implemented in various other fields and domains too such as medicine. The style transfer techniques are rapidly growing in the digital art and market domains and with the help of the pretrained models by TensorFlow Hub, the model execution is rapid and we can obtain the desired styled image.

Thus, with the help of TensorFlow Hub model, the content image was recreated based on the styles of the images, and we could observe the style being transfered to the original image where the recreated image showed the final changes. From the examples of style transfer images created above, we observe that the two images are blend together in order to obtain a desired image which has the style implemented on the content image.

References

[1] Style Transfer Guide | Fritz AI. (n.d.). https://www.fritz.ai/style-transfer/ (https://www.fritz.ai/style-transfer/)

[2] Neural style transfer | TensorFlow Core. (n.d.). TensorFlow. https://www.tensorflow.org/tutorials/generative/style_transfer (https://www.tensorflow.org/tutorials/generative/style_transfer)

[3] Ghiasi, G., Lee, H., Kudlur, M., Dumoulin, V., & Shlens, J. (2017). Exploring the structure of a real-time, arbitrary neural artistic stylization network. Procedings of the British Machine Vision Conference 2017. https://doi.org/10.5244/c.31.114 (https://doi.org/10.5244/c.31.114)