

## GAN 실습

- Topic : Neural style transfer

In [10]:

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.applications import vgg19

base_image_path = keras.utils.get_file("paris.jpg", "https://i.imgur.com/F28w3Ac.jpg")
style_reference_image_path = keras.utils.get_file(
    "starry_night.jpg", "https://i.imgur.com/9ooB60I.jpg"
)
result_prefix = "paris_generated"

# Weights of the different loss components
total_variation_weight = 1e-6
style_weight = 1e-6
content_weight = 2.5e-8

# Dimensions of the generated picture.
width, height = keras.preprocessing.image.load_img(base_image_path).size
img_nrows = 400
img_ncols = int(width * img_nrows / height)
```

In [11]:

```
from IPython.display import Image, display  
  
display(Image(base_image_path))  
display(Image(style_reference_image_path))
```





In [12]:

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

def deprocess_image(x):
    # Util function to convert a tensor into a valid image
    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
```



In [13]:

```
# The gram matrix of an image tensor (feature-wise outer product)

def gram_matrix(x):
    x = tf.transpose(x, (2, 0, 1))
    features = tf.reshape(x, (tf.shape(x)[0], -1))
    gram = tf.matmul(features, tf.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):
    S = gram_matrix(style)
    C = gram_matrix(combination)
    channels = 3
    size = img_nrows * img_ncols
    return tf.reduce_sum(tf.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 tf.reduce_sum(tf.square(combination - base))

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

def total_variation_loss(x):
    a = tf.square(
        x[:, : img_nrows - 1, : img_ncols - 1, :] - x[:, 1:, : img_ncols - 1, :]
    )
    b = tf.square(
        x[:, : img_nrows - 1, : img_ncols - 1, :] - x[:, : img_nrows - 1, 1:, :]
    )
    return tf.reduce_sum(tf.pow(a + b, 1.25))
```

In [14]:

```
# Build a VGG19 model loaded with pre-trained ImageNet weights
model = vgg19.VGG19(weights="imagenet", include_top=False)

# 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])

# Set up a model that returns the activation values for every layer in
# VGG19 (as a dict).
feature_extractor = keras.Model(inputs=model.inputs, outputs=outputs_dict)
```

In [15]:

```
# List of layers to use for the style loss.
style_layer_names = [
    "block1_conv1",
    "block2_conv1",
    "block3_conv1",
    "block4_conv1",
    "block5_conv1",
]
# The layer to use for the content loss.
content_layer_name = "block5_conv2"

def compute_loss(combination_image, base_image, style_reference_image):
    input_tensor = tf.concat(
        [base_image, style_reference_image, combination_image], axis=0
    )
    features = feature_extractor(input_tensor)

    # Initialize the loss
    loss = tf.zeros(shape=())

    # Add content loss
    layer_features = features[content_layer_name]
    base_image_features = layer_features[0, :, :, :]
    combination_features = layer_features[2, :, :, :]
    loss = loss + content_weight * content_loss(
        base_image_features, combination_features
    )

    # Add style loss
    for layer_name in style_layer_names:
        layer_features = features[layer_name]
        style_reference_features = layer_features[1, :, :, :]
        combination_features = layer_features[2, :, :, :]
        sl = style_loss(style_reference_features, combination_features)
        loss += (style_weight / len(style_layer_names)) * sl

    # Add total variation loss
    loss += total_variation_weight * total_variation_loss(combination_image)
    return loss
```



In [16]:

```
@tf.function
def compute_loss_and_grads(combination_image, base_image, style_reference_image):
    with tf.GradientTape() as tape:
        loss = compute_loss(combination_image, base_image, style_reference_image)
        grads = tape.gradient(loss, combination_image)
    return loss, grads
```



In [17]:

```
%time

optimizer = keras.optimizers.SGD(
    keras.optimizers.schedules.ExponentialDecay(
        initial_learning_rate=100.0, decay_steps=100, decay_rate=0.96
    )
)

base_image = preprocess_image(base_image_path)
style_reference_image = preprocess_image(style_reference_image_path)
combination_image = tf.Variable(preprocess_image(base_image_path))

iterations = 4000
for i in range(1, iterations + 1):
    loss, grads = compute_loss_and_grads(
        combination_image, base_image, style_reference_image
    )
    optimizer.apply_gradients([(grads, combination_image)])
    if i % 100 == 0:
        print("Iteration %d: loss=%.2f" % (i, loss))
        img = deprocess_image(combination_image.numpy())
        fname = result_prefix + "_at_iteration_%d.png" % i
        keras.preprocessing.image.save_img(fname, img)
```

Iteration 100: loss=11018.17  
Iteration 200: loss=8513.96  
Iteration 300: loss=7571.53  
Iteration 400: loss=7063.96  
Iteration 500: loss=6736.15  
Iteration 600: loss=6501.74  
Iteration 700: loss=6323.38  
Iteration 800: loss=6181.56  
Iteration 900: loss=6065.41  
Iteration 1000: loss=5967.85  
Iteration 1100: loss=5884.72  
Iteration 1200: loss=5813.02  
Iteration 1300: loss=5750.50  
Iteration 1400: loss=5695.70  
Iteration 1500: loss=5647.25  
Iteration 1600: loss=5604.16  
Iteration 1700: loss=5565.49  
Iteration 1800: loss=5530.76  
Iteration 1900: loss=5499.24  
Iteration 2000: loss=5470.56  
Iteration 2100: loss=5444.41  
Iteration 2200: loss=5420.51  
Iteration 2300: loss=5398.54  
Iteration 2400: loss=5378.33  
Iteration 2500: loss=5359.73  
Iteration 2600: loss=5342.52  
Iteration 2700: loss=5326.58  
Iteration 2800: loss=5311.82  
Iteration 2900: loss=5298.10  
Iteration 3000: loss=5285.33  
Iteration 3100: loss=5273.41  
Iteration 3200: loss=5262.25  
Iteration 3300: loss=5251.80

```
Iteration 3400: loss=5242.03
Iteration 3500: loss=5232.87
Iteration 3600: loss=5224.28
Iteration 3700: loss=5216.18
Iteration 3800: loss=5208.57
Iteration 3900: loss=5201.39
Iteration 4000: loss=5194.62
CPU times: user 12min 42s, sys: 8min 11s, total: 20min 53s
Wall time: 20min 54s
```

In [19]:

```
display(Image(result_prefix + "_at_iteration_100.png"))
```





In [20]:

```
display(Image(result_prefix + "_at_iteration_1000.png"))
```





In [21]:

```
display(Image(result_prefix + "_at_iteration_2000.png"))
```





In [22]:

```
display(Image(result_prefix + "_at_iteration_3000.png"))
```



In [18]:

```
display(Image(result_prefix + "_at_iteration_4000.png"))
```



- REF : [https://keras.io/examples/generative/neural\\_style\\_transfer/](https://keras.io/examples/generative/neural_style_transfer/)  
[\(https://keras.io/examples/generative/neural\\_style\\_transfer/\)](https://keras.io/examples/generative/neural_style_transfer/)

In [ ]: