In [1]:

```python
import os
import sys
import scipy.io
import scipy.misc
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
from PIL import Image as img
import numpy as np
import tensorflow as tf
from tensorflow.python.framework.ops import EagerTensor
```

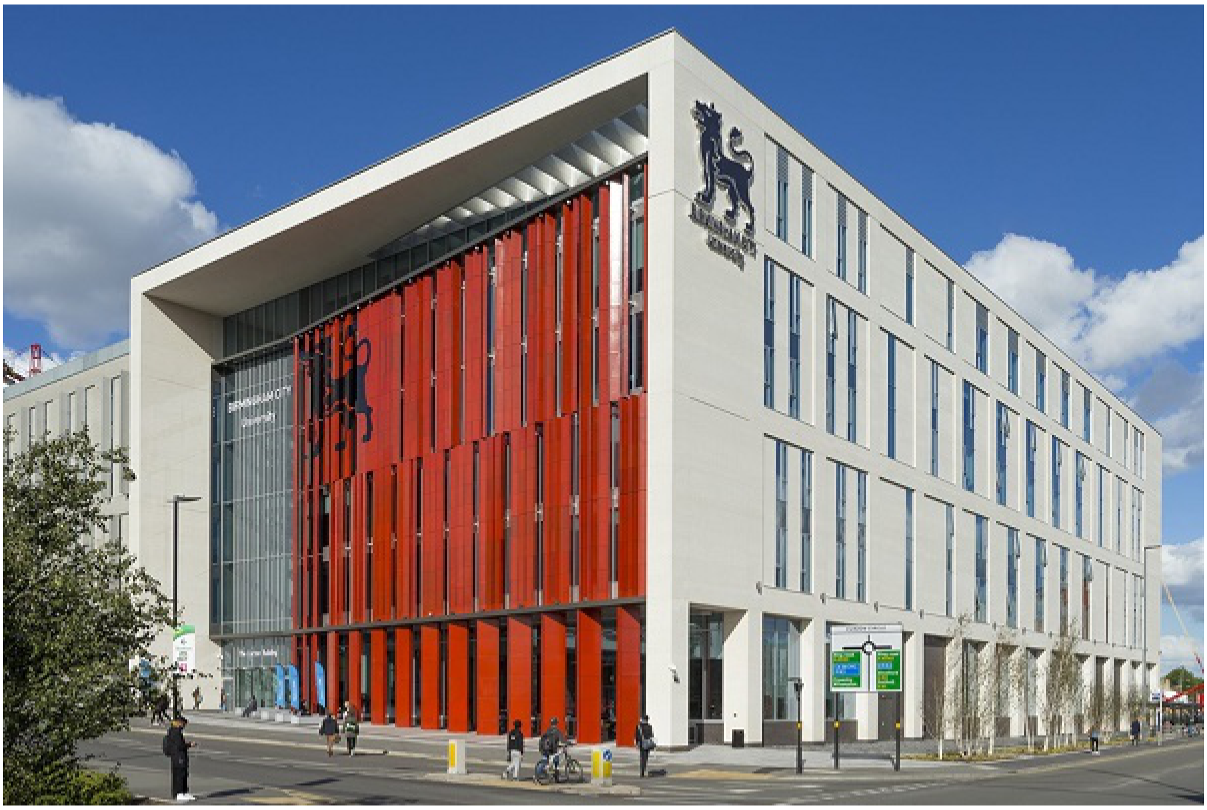
```
WARNING:tensorflow:From C:\Users\Mukul\AppData\Roaming\Python\Python311\site-package
s\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is depre
cated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [6]:

```python
tf.random.set_seed(272)
img_size = 400
vgg = tf.keras.applications.VGG19(include_top=False,
                                  input_shape=(img_size, img_size, 3),
                                  weights='vgg19_weights_tf_dim_ordering_tf_kerne

vgg.trainable = False
im = img.open("3.jpeg")
im
```

Out[6]:



In [7]:

```
def compute_content_cost(content_output, generated_output):
    a_C = content_output[-1]
    a_G = generated_output[-1]
    m, n_H, n_W, n_C = a_G.get_shape().as_list()
    a_C_unrolled = tf.reshape(a_C, shape=[m, n_H * n_W, n_C]) # Or tf.reshape(a_
    a_G_unrolled = tf.reshape(a_G, shape=[m, n_H * n_W, n_C]) # Or tf.reshape(a_
    J_content =  tf.reduce_sum(tf.square(a_C_unrolled - a_G_unrolled))/(4.0 * n_
    return J_content
```

In [8]:

```
def gram_matrix(A):
    GA = tf.matmul(A, tf.transpose(A))
    return GA
```

In [9]:

```

def compute_layer_style_cost(a_S, a_G):
    m, n_H, n_W, n_C = a_G.get_shape().as_list()
    a_S = tf.transpose(tf.reshape(a_S, shape=[-1, n_C]))
    a_G = tf.transpose(tf.reshape(a_G, shape=[-1, n_C]))
    GS = gram_matrix(a_S)
    GG = gram_matrix(a_G)
    J_style_layer = tf.reduce_sum(tf.square(GS - GG))/(4.0 *(( n_H * n_W * n_C)*
    return J_style_layer
```

In [10]:

```
for layer in vgg.layers:
    print(layer.name)
```

```
input_5
block1_conv1
block1_conv2
block1_pool
block2_conv1
block2_conv2
block2_pool
block3_conv1
block3_conv2
block3_conv3
block3_conv4
block3_pool
block4_conv1
block4_conv2
block4_conv3
block4_conv4
block4_pool
block5_conv1
block5_conv2
block5_conv3
block5_conv4
block5_pool
```

In [11]:

```
vgg.get_layer('block5_conv4').output
```

Out[11]:

```
<KerasTensor: shape=(None, 25, 25, 512) dtype=float32 (created by layer 'block5_conv
4')>
```

In [12]:

```
STYLE_LAYERS = [
    ('block1_conv1', 1.0),
    ('block2_conv1', 0.8),
    ('block3_conv1', 0.7),
    ('block4_conv1', 0.2),
    ('block5_conv1', 0.1)]
```

In [13]:

```
def compute_style_cost(style_image_output, generated_image_output, STYLE_LAYERS=S
    J_style = 0
    a_S = style_image_output[1:]
    a_G = generated_image_output[1:]
    for i, weight in zip(range(len(a_S)), STYLE_LAYERS):
        J_style_layer = compute_layer_style_cost(a_S[i], a_G[i])
        J_style += weight[1] * J_style_layer
    return J_style
```

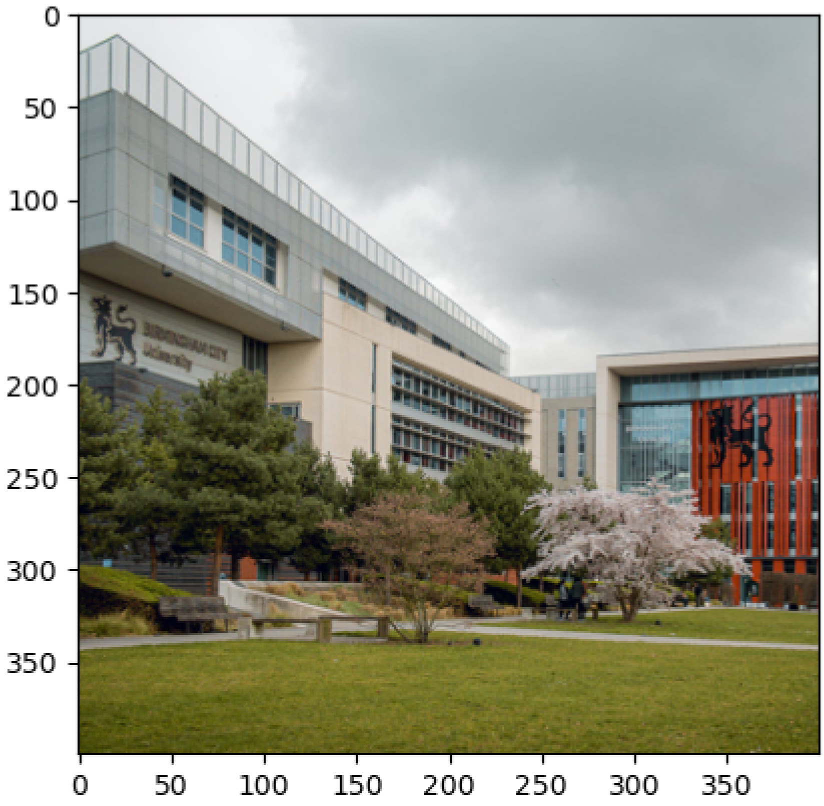
In [14]:

```
def total_cost(J_content, J_style, alpha = 10, beta = 40):
    J = alpha * J_content + beta * J_style
    return J
```

In [19]:

```
content_image = np.array(img.open("1.jpeg").resize((img_size, img_size)))
content_image = tf.constant(np.reshape(content_image, ((1,) + content_image.shap
print(content_image.shape)
imshow(content_image[0])
plt.show()
```

```
(1, 400, 400, 3)
```

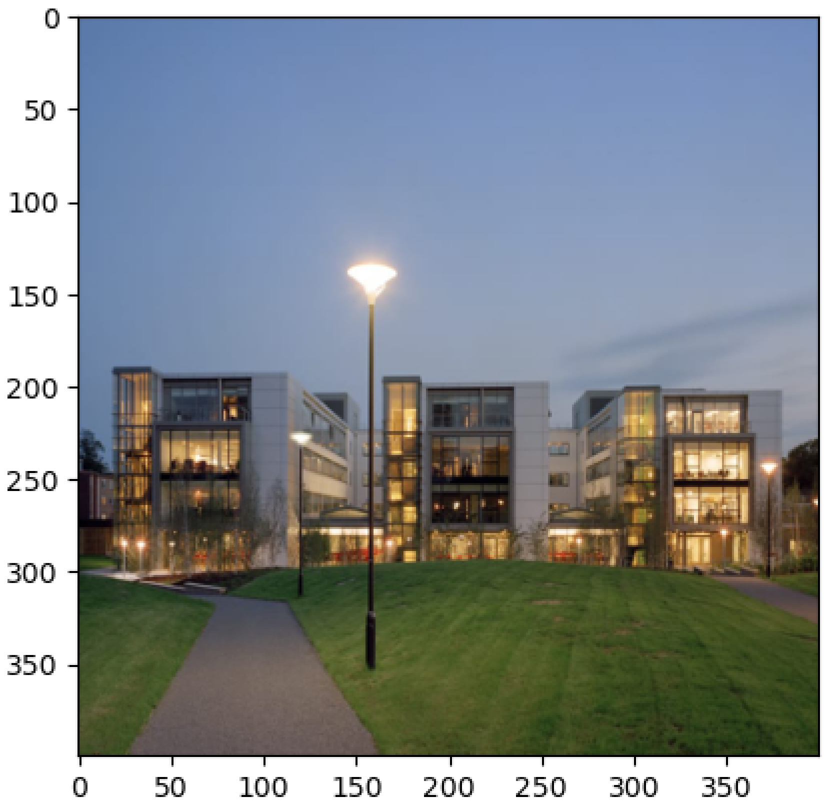


In [20]:

```
style_image =  np.array(img.open("2.jpeg").resize((img_size, img_size)))
style_image = tf.constant(np.reshape(style_image, ((1,) + style_image.shape)))

print(style_image.shape)
imshow(style_image[0])
plt.show()
```

```
(1, 400, 400, 3)
```

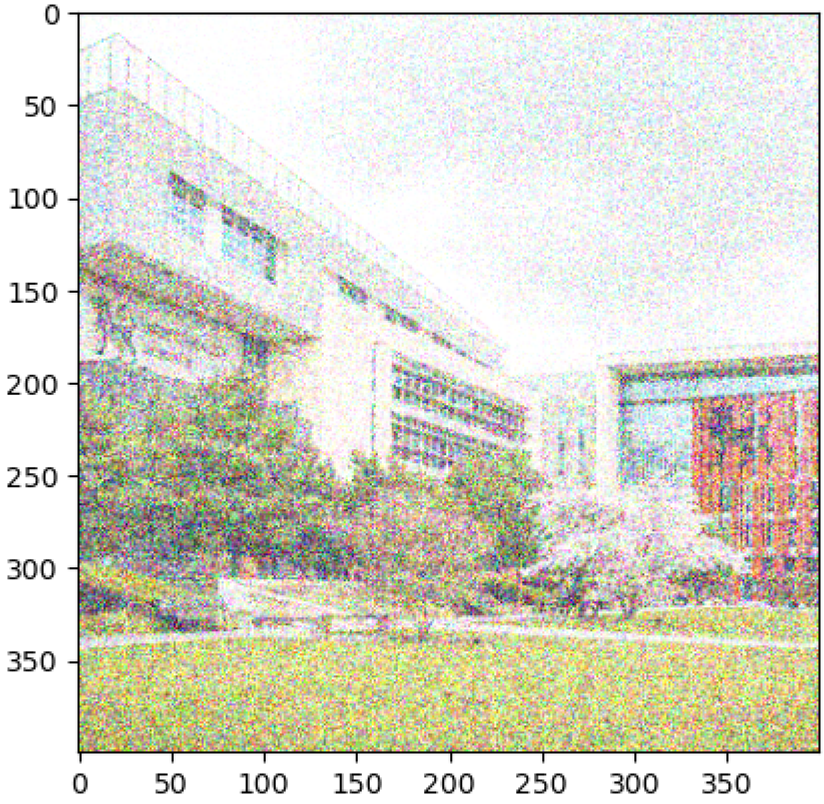


In [21]:

```
generated_image = tf.Variable(tf.image.convert_image_dtype(content_image, tf.flo
noise = tf.random.uniform(tf.shape(generated_image), 0, 0.8)
generated_image = tf.add(generated_image, noise)
generated_image = tf.clip_by_value(generated_image, clip_value_min=0.0, clip_valu

print(generated_image.shape)
imshow(generated_image.numpy()[0])
plt.show()
```

```
(1, 400, 400, 3)
```



In [22]:

```
def get_layer_outputs(vgg, layer_names):
    outputs = [vgg.get_layer(layer[0]).output for layer in layer_names]

    model = tf.keras.Model([vgg.input], outputs)
    return model
```

In [23]:

```
content_layer = [('block5_conv4', 1)]
vgg_model_outputs = get_layer_outputs(vgg, STYLE_LAYERS + content_layer)
```

In [24]:

```
content_target = vgg_model_outputs(content_image)
style_targets = vgg_model_outputs(style_image)
```

In [25]:

```
preprocessed_content =  tf.Variable(tf.image.convert_image_dtype(content_image, 
a_C = vgg_model_outputs(preprocessed_content)
a_G = vgg_model_outputs(generated_image)
J_content = compute_content_cost(a_C, a_G)
```

In [26]:

```
preprocessed_style =  tf.Variable(tf.image.convert_image_dtype(style_image, tf.f
a_S = vgg_model_outputs(preprocessed_style)
J_style = compute_style_cost(a_S, a_G)
print(J_style)
```

```
tf.Tensor(3956.828, shape=(), dtype=float32)
```

In [30]:

```
def clip_0_1(image):
    return tf.clip_by_value(image, clip_value_min=0.0, clip_value_max=1.0)

def tensor_to_image(tensor):
    tensor = tensor * 255
    tensor = np.array(tensor, dtype=np.uint8)
    if np.ndim(tensor) > 3:
        assert tensor.shape[0] == 1
        tensor = tensor[0]
    return img.fromarray(tensor)


```

In [31]:

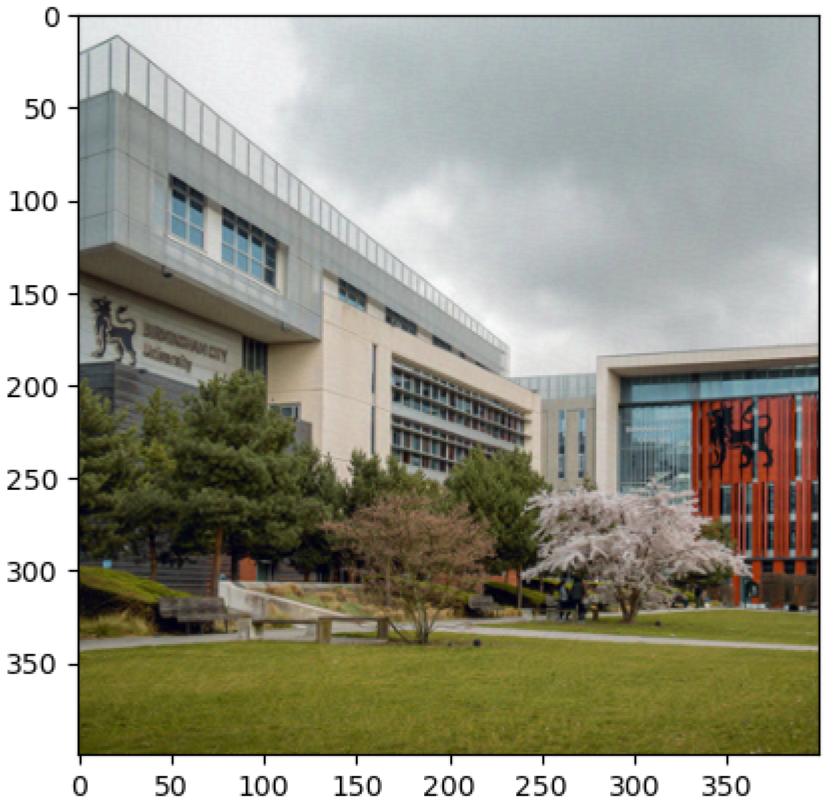
```
def train_step(generated_image, alpha = 10, beta = 40):
    with tf.GradientTape() as tape:
        a_G = vgg_model_outputs(generated_image)
        J_style = compute_style_cost(a_S, a_G)
        J_content = compute_content_cost(a_C, a_G)
        J = total_cost(J_content, J_style,alpha = alpha, beta = beta)
    grad = tape.gradient(J, generated_image)

    optimizer.apply_gradients([(grad, generated_image)])
    generated_image.assign(clip_0_1(generated_image))
    return J
```

In [ ]:

```
generated_image = tf.Variable(tf.image.convert_image_dtype(content_image, tf.floa
optimizer = tf.optimizers.Adam(learning_rate=0.01)
epochs = 1000
for i in range(epochs):
    train_step(generated_image,alpha = 100, beta = 10**2)
    if i % 250 == 0:
        print(f"Epoch {i} ")
    if i % 250 == 0:
        image = tensor_to_image(generated_image)
        imshow(image)
        plt.show()
```

```
Epoch 0
```



In [ ]: