1- Neural Style Transfer with 'Abstract Art' Painting Style

```
import numpy as np
from PIL import Image
import requests
from io import BytesIO

from keras import backend
from keras.models import Model
from keras.applications.vgg16 import VGG16

from scipy.optimize import fmin_l_bfgs_b
```

```
Using TensorFlow backend.
```

```
ITERATIONS = 15
CHANNELS = 3
IMAGE_SIZE = 500
IMAGE_WIDTH = IMAGE_SIZE
IMAGE_HEIGHT = IMAGE_SIZE
IMAGENET_MEAN_RGB_VALUES = [123.68, 116.779, 103.939]
CONTENT_WEIGHT = 0.05
STYLE_WEIGHT = 5.5
TOTAL_VARIATION_WEIGHT = 0.995
TOTAL_VARIATION_LOSS_FACTOR = 1.30
```

```
input_image_path = "input.png"
style_image_path = "style.png"
output_image_path = "output.png"
```

```
#Input visualization
input_image = Image.open("/content/Csueastbay.jpg")
input_image = input_image.resize((IMAGE_WIDTH, IMAGE_HEIGHT))
input_image.save(input_image_path)
input_image
```


```
# Style visualization
style_image = Image.open("/content/28.jpg")
style_image = style_image.resize((IMAGE_WIDTH, IMAGE_HEIGHT))
style_image.save(style_image_path)
style_image
```

```
# Data normalization and reshaping from RGB to BGR
input_image_array = np.asarray(input_image, dtype="float32")
input_image_array = np.expand_dims(input_image_array, axis=0)
input_image_array[:, :, :, 0] -= IMAGENET_MEAN_RGB_VALUES[2]
input_image_array[:, :, :, 1] -= IMAGENET_MEAN_RGB_VALUES[1]
input_image_array[:, :, :, 2] -= IMAGENET_MEAN_RGB_VALUES[0]
input_image_array = input_image_array[:, :, :, ::-1]

style_image_array = np.asarray(style_image, dtype="float32")
style_image_array = np.expand_dims(style_image_array, axis=0)
style_image_array[:, :, :, 0] -= IMAGENET_MEAN_RGB_VALUES[2]
style_image_array[:, :, :, 1] -= IMAGENET_MEAN_RGB_VALUES[1]
style_image_array[:, :, :, 2] -= IMAGENET_MEAN_RGB_VALUES[0]
style_image_array = style_image_array[:, :, :, ::-1]


# Model
input_image = backend.variable(input_image_array)
style_image = backend.variable(style_image_array)
combination_image = backend.placeholder((1, IMAGE_HEIGHT, IMAGE_SIZE, 3))

input_tensor = backend.concatenate([input_image,style_image,combination_image], axis=
model = VGG16(input_tensor=input_tensor, include_top=False)
```

```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/d
58892288/58889256 [==============================] - 2s 0us/step
```

```
def content_loss(content, combination):
    return backend.sum(backend.square(combination - content))

layers = dict([(layer.name, layer.output) for layer in model.layers])

content_layer = 'block2_conv2'
layer_features = layers[content_layer]
content_image_features = layer_features[0, :, :, :]
combination_features = layer_features[2, :, :, :]

loss = backend.variable(0.)
loss =loss+ CONTENT_WEIGHT * content_loss(content_image_features,
                                      combination_features)


def gram_matrix(x):
    features = backend.batch_flatten(backend.permute_dimensions(x, (2, 0, 1)))
    gram = backend.dot(features, backend.transpose(features))
    return gram


def compute_style_loss(style, combination):
    style = gram_matrix(style)
    combination = gram_matrix(combination)
    size = IMAGE_HEIGHT * IMAGE_WIDTH
    return backend.sum(backend.square(style - combination)) / (4. * (CHANNELS ** 2) *

style_layers = ["block1_conv2", "block2_conv2", "block3_conv3", "block4_conv3", "bloc
for layer_name in style_layers:
    layer_features = layers[layer_name]
    style_features = layer_features[1, :, :, :]
    combination_features = layer_features[2, :, :, :]
    style_loss = compute_style_loss(style_features, combination_features)
    loss =loss+ (STYLE_WEIGHT / len(style_layers)) * style_loss


def total_variation_loss(x):
    a = backend.square(x[:, :IMAGE_HEIGHT-1, :IMAGE_WIDTH-1, :] - x[:, 1:, :IMAGE_WID
    b = backend.square(x[:, :IMAGE_HEIGHT-1, :IMAGE_WIDTH-1, :] - x[:, :IMAGE_HEIGHT-
    return backend.sum(backend.pow(a + b, TOTAL_VARIATION_LOSS_FACTOR))

loss =loss+ TOTAL_VARIATION_WEIGHT * total_variation_loss(combination_image)


outputs = [loss]
outputs += backend.gradients(loss, combination_image)

def evaluate_loss_and_gradients(x):
    x = x.reshape((1, IMAGE_HEIGHT, IMAGE_WIDTH, CHANNELS))
```

```
        outs = backend.function([combination_image], outputs)([x])
        loss = outs[0]
        gradients = outs[1].flatten().astype("float64")
        return loss, gradients

    class Evaluator:

        def loss(self, x):
            loss, gradients = evaluate_loss_and_gradients(x)
            self._gradients = gradients
            return loss

        def gradients(self, x):
            return self._gradients

    evaluator = Evaluator()


    outputs = [loss]
    outputs += backend.gradients(loss, combination_image)

    def evaluate_loss_and_gradients(x):
        x = x.reshape((1, IMAGE_HEIGHT, IMAGE_WIDTH, CHANNELS))
        outs = backend.function([combination_image], outputs)([x])
        loss = outs[0]
        gradients = outs[1].flatten().astype("float64")
        return loss, gradients

    class Evaluator:

        def loss(self, x):
            loss, gradients = evaluate_loss_and_gradients(x)
            self._gradients = gradients
            return loss

        def gradients(self, x):
            return self._gradients

    evaluator = Evaluator()


    x = np.random.uniform(0, 255, (1, IMAGE_HEIGHT, IMAGE_WIDTH, 3)) - 128.

    for i in range(ITERATIONS):
        x, loss, info = fmin_l_bfgs_b(evaluator.loss, x.flatten(), fprime=evaluator.gradi
        print("Iteration %d completed with loss %d" % (i, loss))

    x = x.reshape((IMAGE_HEIGHT, IMAGE_WIDTH, CHANNELS))
    x = x[:, :, ::-1]
    x[:, :, 0] += IMAGENET_MEAN_RGB_VALUES[2]
    x[:, :, 1] += IMAGENET_MEAN_RGB_VALUES[1]
    x[:, :, 2] += IMAGENET_MEAN_RGB_VALUES[0]
    x = np.clip(x, 0, 255).astype("uint8")
```

```
output_image = Image.fromarray(x)
output_image.save(output_image_path)
output_image
```

```
Iteration 0 completed with loss 356573970432
Iteration 1 completed with loss 99040116736
Iteration 2 completed with loss 66125950976
Iteration 3 completed with loss 53006192640
Iteration 4 completed with loss 48568373248
Iteration 5 completed with loss 46689280000
Iteration 6 completed with loss 45687005184
Iteration 7 completed with loss 45101993984
Iteration 8 completed with loss 44735623168
Iteration 9 completed with loss 44479975424
Iteration 10 completed with loss 44279324672
Iteration 11 completed with loss 44121337856
Iteration 12 completed with loss 43996372992
Iteration 13 completed with loss 43894128640
Iteration 14 completed with loss 43807543296
```