import os import tensorflow as tf os.environ['TFHUB\_MODEL\_LOAD\_FORMAT'] = 'COMPRESSED'

from google.colab import drive drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call

import IPython.display as display

import matplotlib.pyplot as plt import matplotlib as mpl mpl.rcParams['figure.figsize'] = (12, 12) mpl.rcParams['axes.grid'] = False

import numpy as np import PIL.Image import time import functools

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 PIL.Image.fromarray(tensor)

def load\_img(path\_to\_img):

max\_dim = 512

img = tf.io.read\_file(path\_to\_img)

img = tf.image.decode\_image(img, channels=3)

img = tf.image.convert\_image\_dtype(img, tf.float32)

shape = tf.cast(tf.shape(img)[:-1], tf.float32)

long\_dim = max(shape)

scale = max\_dim / long\_dim

new\_shape = tf.cast(shape \* scale, tf.int32)

img = tf.image.resize(img, new\_shape)

img = img[tf.newaxis, :]

return img

def imshow(image, title=None):

if len(image.shape) > 3:

image = tf.squeeze(image, axis=0)

plt.imshow(image) if title:

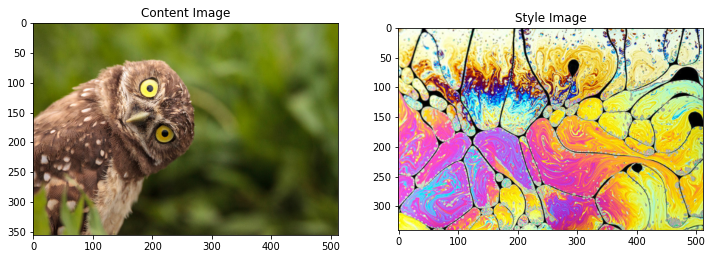
if title:

plt.title(title)

content\_image = load\_img("/owl.jpg") style\_image = load\_img("/style.jpeg")

plt.subplot(1, 2, 1) imshow(content\_image, 'Content Image')

plt.subplot(1, 2, 2) imshow(style\_image, 'Style Image')



import tensorflow\_hub as hub hub\_model = hub.load('https://tfhub.dev/google/magenta/arbitrary-image-stylizationstylized\_image = hub\_model(tf.constant(content\_image), tf.constant(style\_image))[0] tensor\_to\_image(stylized\_image)



x = tf.keras.applications.vgg19.preprocess\_input(content\_image\*255) x = tf.image.resize(x, (224, 224))

tf k li ti VGG19(i l d t T i ht 'i t')

vgg = tf.keras.applications.VGG19(include\_top=True, weights='imagenet') prediction\_probabilities = vgg(x) prediction\_probabilities.shape

TensorShape([1, 1000])

predicted\_top\_5 = tf.keras.applications.vgg19.decode\_predictions(prediction\_probabi

[(class\_name, prob) for (number, class\_name, prob) in predicted\_top\_5]

[('great\_grey\_owl', 0.73142934),

('ringlet', 0.051973876),

('bittern', 0.049039505),

('vulture', 0.03952588), ('kite', 0.029020881)]

vgg = tf.keras.applications.VGG19(include\_top=False, weights='imagenet')

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

input\_17 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

content\_layers = ['block5\_conv2']

style\_layers = ['block1\_conv1', 'block2\_conv1',

'block3\_conv1',

'block4\_conv1',

'block5\_conv1']

num\_content\_layers = len(content\_layers) num\_style\_layers = len(style\_layers)

def vgg\_layers(layer\_names):

vgg = tf.keras.applications.VGG19(include\_top=False, weights='imagenet')

vgg.trainable = False

outputs = [vgg.get\_layer(name).output for name in layer\_names]

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

return model

style\_extractor = vgg\_layers(style\_layers) style\_outputs = style\_extractor(style\_image\*255) for name, output in zip(style\_layers, style\_outputs):

print(name)

print(" shape: ", output.numpy().shape)

print(" min: ", output.numpy().min())

print(" max: ", output.numpy().max())

print(" mean: ", output.numpy().mean()) print()

block1\_conv1

shape: (1, 341, 512, 64) min: 0.0 max: 836.31885 mean: 49.30245

block2\_conv1

shape: (1, 170, 256, 128) min: 0.0 max: 5778.871 mean: 260.07843

block3\_conv1

shape: (1, 85, 128, 256) min: 0.0 max: 12648.359 mean: 290.02325

block4\_conv1

shape: (1, 42, 64, 512) min: 0.0 max: 25104.59 mean: 917.1499

block5\_conv1

shape: (1, 21, 32, 512) min: 0.0 max: 3712.8596 mean: 59.95148

def gram\_matrix(input\_tensor):

result = tf.linalg.einsum('bijc,bijd->bcd', input\_tensor, input\_tensor)

input\_shape = tf.shape(input\_tensor)

num\_locations = tf.cast(input\_shape[1]\*input\_shape[2], tf.float32) return result/(num\_locations)

class StyleContentModel(tf.keras.models.Model):

def \_\_init\_\_(self, style\_layers, content\_layers):

super(StyleContentModel, self).\_\_init\_\_()

self.vgg = vgg\_layers(style\_layers + content\_layers)

self.style\_layers = style\_layers

self.content\_layers = content\_layers

self.num\_style\_layers = len(style\_layers)

self.vgg.trainable = False

def call(self, inputs):

"Expects float input in [0,1]"

inputs = inputs\*255.0

preprocessed\_input = tf.keras.applications.vgg19.preprocess\_input(inputs)

outputs = self.vgg(preprocessed\_input)

style\_outputs, content\_outputs = (outputs[:self.num\_style\_layers], outputs[self.num\_style\_layers:])

style\_outputs = [gram\_matrix(style\_output)

for style\_output in style\_outputs]

content\_dict = {content\_name: value

for content\_name, value

in zip(self.content\_layers, content\_outputs)}

style\_dict = {style\_name: value

for style\_name, value

in zip(self.style\_layers, style\_outputs)}

return {'content': content\_dict, 'style': style\_dict}

extractor = StyleContentModel(style\_layers, content\_layers)

results = extractor(tf.constant(content\_image)) print('Styles:') for name, output in sorted(results['style'].items()):

print(" ", name)

print(" shape: ", output.numpy().shape)

print(" min: ", output.numpy().min())

print(" max: ", output.numpy().max())

print(" mean: ", output.numpy().mean())

print() print("Contents:") for name, output in sorted(results['content'].items()):

print(" ", name)

print(" shape: ", output.numpy().shape)

print(" min: ", output.numpy().min())

print(" max: ", output.numpy().max())

print(" mean: ", output.numpy().mean()) Styles: block1\_conv1 shape: (1, 64, 64) min: 0.002655763 max: 16853.451 mean: 337.22418

block2\_conv1 shape: (1, 128, 128) min: 0.0 max: 48416.816 mean: 9061.926

block3\_conv1 shape: (1, 256, 256) min: 0.0 max: 317588.75 mean: 8140.998

block4\_conv1 shape: (1, 512, 512) min: 0.0 max: 3212877.0 mean: 145063.1

block5\_conv1 shape: (1, 512, 512) min: 0.0 max: 129496.65 mean: 1231.7739

Contents:

block5\_conv2 shape: (1, 22, 32, 512) min: 0.0 max: 1756.408 mean: 12.337271

style\_targets = extractor(style\_image)['style'] content\_targets = extractor(content\_image)['content']

image = tf.Variable(content\_image)

def clip\_0\_1(image):

return tf.clip\_by\_value(image, clip\_value\_min=0.0, clip\_value\_max=1.0)

opt = tf.optimizers.Adam(learning\_rate=0.02, beta\_1=0.99, epsilon=1e-1)

style\_weight=1e-2 content\_weight=1e4

def style\_content\_loss(outputs): style\_outputs = outputs['style']

content\_outputs = outputs['content']

style\_loss = tf.add\_n([tf.reduce\_mean((style\_outputs[name]-style\_targets[name])

for name in style\_outputs.keys()])

style\_loss \*= style\_weight / num\_style\_layers

content\_loss = tf.add\_n([tf.reduce\_mean((content\_outputs[name]-content\_targets[

for name in content\_outputs.keys()])

content\_loss \*= content\_weight / num\_content\_layers

loss = style\_loss + content\_loss

return loss

@tf.function() def train\_step(image):

with tf.GradientTape() as tape:

outputs = extractor(image)

loss = style\_content\_loss(outputs)

grad = tape.gradient(loss, image)

opt.apply\_gradients([(grad, image)]) image.assign(clip\_0\_1(image))

train\_step(image) train\_step(image) train\_step(image) tensor\_to\_image(image)



import time start = time.time() epochs = 10

steps\_per\_epoch = 100

step = 0 for n in range(epochs): for m in range(steps per epoch): for m in range(steps\_per\_epoch):

step += 1

train\_step(image)

print(".", end='', flush=True)

display.clear\_output(wait=True)

display.display(tensor\_to\_image(image))

print("Train step: {}".format(step))

end = time.time()

print("Total time: {:.1f}".format(end-start))



Train step: 1000

Total time: 69.7

def high\_pass\_x\_y(image):

x\_var = image[:, :, 1:, :] - image[:, :, :-1, :] y\_var = image[:, 1:, :, :] - image[:, :-1, :, :]

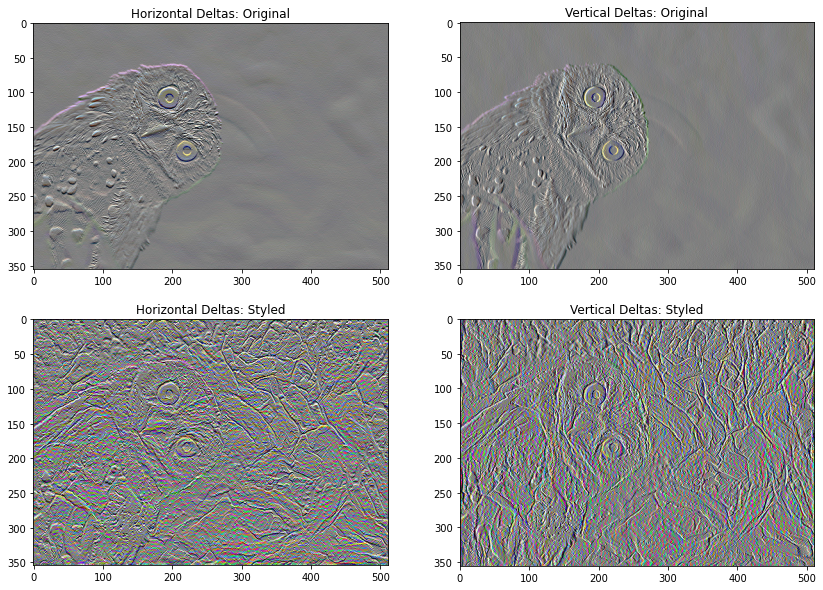
return x\_var, y\_var

x\_deltas, y\_deltas = high\_pass\_x\_y(content\_image) plt.figure(figsize=(14, 10)) plt.subplot(2, 2, 1) imshow(clip\_0\_1(2\*y\_deltas+0.5), "Horizontal Deltas: Original")

plt.subplot(2, 2, 2) imshow(clip\_0\_1(2\*x\_deltas+0.5), "Vertical Deltas: Original")

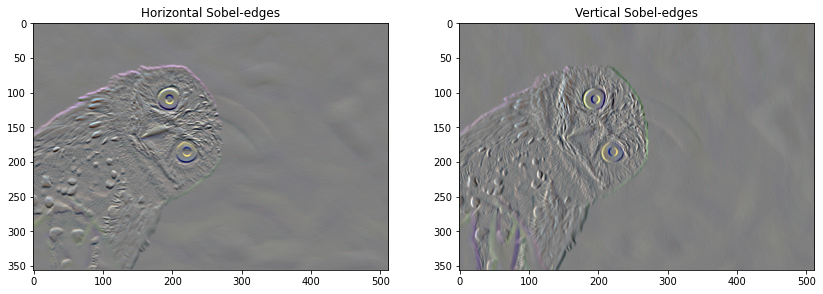
x\_deltas, y\_deltas = high\_pass\_x\_y(image) plt.subplot(2, 2, 3) imshow(clip\_0\_1(2\*y\_deltas+0.5), "Horizontal Deltas: Styled")

plt.subplot(2, 2, 4) imshow(clip 0 1(2\*x deltas+0.5), "Vertical Deltas: Styled") ( p\_ \_ ( \_ ), y )



plt.figure(figsize=(14, 10))

sobel = tf.image.sobel\_edges(content\_image) plt.subplot(1, 2, 1) imshow(clip\_0\_1(sobel[..., 0]/4+0.5), "Horizontal Sobel-edges") plt.subplot(1, 2, 2) imshow(clip\_0\_1(sobel[..., 1]/4+0.5), "Vertical Sobel-edges")



def

total\_variation\_loss

(

image

):

x\_deltas, y\_deltas = high\_pass\_x\_y(image)

return

tf.reduce\_sum(tf.

abs

(

x\_deltas)) + tf.reduce\_sum(tf.

abs

(

y\_deltas

))

total\_variation\_loss(image).numpy()

144818.97

tf.image.total\_variation(image).numpy() array([144818.97], dtype=float32)

total\_variation\_weight=30

@tf.function() def train\_step(image):

with tf.GradientTape() as tape:

outputs = extractor(image)

loss = style\_content\_loss(outputs)

loss += total\_variation\_weight\*tf.image.total\_variation(image)

grad = tape.gradient(loss, image)

opt.apply\_gradients([(grad, image)]) image.assign(clip\_0\_1(image))

image = tf.Variable(content\_image)

import time start = time.time() epochs = 10 steps\_per\_epoch = 100

step = 0 for n in range(epochs):

for m in range(steps\_per\_epoch):

step += 1

train\_step(image)

print(".", end='', flush=True)

display.clear\_output(wait=True)

display.display(tensor\_to\_image(image))

print("Train step: {}".format(step))

end = time.time() print("Total time: {:.1f}".format(end-start))



Train step: 1000

Total time: 69.6

file\_name =

'stylized\_tranfer\_image.png'

tensor\_to\_image(image).save(file\_name)

try

:

from

google.colab

import

files

except

ImportError:

pass

else

:

files.download(file\_name)