import tensorflow as tf

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.applications import vgg19

from tensorflow.keras.models import Model

from tensorflow.keras.preprocessing import image as kp\_image

import PIL.Image

# Helper to load and preprocess image

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

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

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

img = tf.image.resize(img, (224, 224))

img = vgg19.preprocess\_input(img)

return tf.expand\_dims(img, axis=0)

# Helper to deprocess image (convert back to displayable format)

def deprocess\_img(processed\_img):

x = processed\_img.copy()

x = x.reshape((224, 224, 3))

x[:, :, 0] += 103.939

x[:, :, 1] += 116.779

x[:, :, 2] += 123.68

x = x[:, :, ::-1]

x = np.clip(x, 0, 255).astype('uint8')

return x

# Content and style layers

content\_layers = ['block5\_conv2']

style\_layers = ['block1\_conv1','block2\_conv1','block3\_conv1','block4\_conv1','block5\_conv1']

def get\_model():

vgg = vgg19.VGG19(include\_top=False, weights='imagenet')

vgg.trainable = False

outputs = [vgg.get\_layer(name).output for name in style\_layers + content\_layers]

model = Model(inputs=vgg.input, outputs=outputs)

return model

# Gram matrix (for style)

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

# Compute style and content features

def get\_feature\_representations(model, content\_path, style\_path):

content\_image = load\_and\_process\_img(content\_path)

style\_image = load\_and\_process\_img(style\_path)

style\_outputs = model(style\_image)

content\_outputs = model(content\_image)

style\_features = [gram\_matrix(style\_layer) for style\_layer in style\_outputs[:len(style\_layers)]]

content\_features = content\_outputs[len(style\_layers):]

return style\_features, content\_features

# Compute loss

def compute\_loss(model, loss\_weights, init\_image, gram\_style\_features, content\_features):

style\_weight, content\_weight = loss\_weights

model\_outputs = model(init\_image)

style\_output\_features = model\_outputs[:len(style\_layers)]

content\_output\_features = model\_outputs[len(style\_layers):]

style\_score = tf.add\_n([tf.reduce\_mean((gram\_matrix(style\_output) - gram\_style)\*\*2)

for style\_output, gram\_style in zip(style\_output\_features, gram\_style\_features)])

content\_score = tf.add\_n([tf.reduce\_mean((content\_output - content\_target)\*\*2)

for content\_output, content\_target in zip(content\_output\_features, content\_features)])

style\_score \*= style\_weight

content\_score \*= content\_weight

loss = style\_score + content\_score

return loss

# Apply style transfer

@tf.function()

def compute\_grads(cfg):

with tf.GradientTape() as tape:

all\_loss = compute\_loss(\*\*cfg)

total\_loss = all\_loss

return tape.gradient(total\_loss, cfg['init\_image']), total\_loss

def run\_style\_transfer(content\_path, style\_path, epochs=250, style\_weight=1e-2, content\_weight=1e4):

model = get\_model()

for layer in model.layers:

layer.trainable = False

style\_features, content\_features = get\_feature\_representations(model, content\_path, style\_path)

init\_image = load\_and\_process\_img(content\_path)

init\_image = tf.Variable(init\_image, dtype=tf.float32)

opt = tf.optimizers.Adam(learning\_rate=5)

cfg = {

'model': model,

'loss\_weights': (style\_weight, content\_weight),

'init\_image': init\_image,

'gram\_style\_features': style\_features,

'content\_features': content\_features

}

best\_loss = float('inf')

best\_img = None

for i in range(epochs):

grads, loss = compute\_grads(cfg)

opt.apply\_gradients([(grads, init\_image)])

clipped = tf.clip\_by\_value(init\_image, -128.0, 128.0)

init\_image.assign(clipped)

if loss < best\_loss:

best\_loss = loss

best\_img = init\_image.numpy()

if i % 50 == 0:

print(f"Iteration {i}: loss={loss:.2f}")

return deprocess\_img(best\_img)

# Paths to content and style images

content\_path = 'path\_to\_your\_content\_image.jpg'

style\_path = 'path\_to\_your\_style\_image.jpg'

stylized\_image = run\_style\_transfer(content\_path, style\_path)

plt.imshow(stylized\_image)

plt.title("Stylized Image")

plt.axis('off')

plt.show()