##Transfer the style of image color.jpg into SanFrancisco.jpg

##Put the python file and the two images noted above in the same folder and then run the python file

##might take 15-30 minutes on a normal laptop

# Imports

import os

os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '2'

import numpy as np

import tensorflow as tf

physical\_devices = tf.config.experimental.list\_physical\_devices('GPU')

if len(physical\_devices) > 0:

tf.config.experimental.set\_memory\_growth(physical\_devices[0], True)

from tensorflow.keras.preprocessing import image

from tensorflow.keras.models import Model

from tensorflow.keras.applications import vgg16

from tensorflow import keras

from datetime import datetime

def print\_with\_time(\*args,\*\*kwargs):

print(str(datetime.now())+'[{}]:'.format(\_\_file\_\_), \*args, \*\*kwargs)

# Hyperparams

style\_weight = 0.00002

content\_weight = 1.0

variation\_weight = 0.001

learning\_rate = 5.

ITERATIONS = 50

# Paths

content\_image\_path = "SanFrancisco.jpg" # San Francisco

style\_image\_path = "color.jpg" # color

IMAGENET\_MEAN\_RGB\_VALUES = [123.68, 116.779, 103.939]

# Content layer where will pull our feature maps

content\_layers = ['block2\_conv2']

# Style layer of interest

style\_layers = ["block1\_conv2", "block2\_conv2", "block3\_conv3", "block4\_conv3", "block5\_conv3"]

# Content Image

content\_image = image.load\_img(content\_image\_path)

content\_image = image.img\_to\_array(content\_image)

content\_image = np.expand\_dims(content\_image, axis=0)

content\_image\_processed = vgg16.preprocess\_input(content\_image.copy())

# Style Image

style\_image = image.load\_img(style\_image\_path)

style\_image = image.img\_to\_array(style\_image)

style\_image = np.expand\_dims(style\_image, axis=0)

style\_image\_processed = vgg16.preprocess\_input(style\_image.copy())

# CNN Model

vgg\_network = vgg16.VGG16(weights='imagenet', include\_top=False)

vgg\_network.trainalbe = False

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)

style\_extractor = keras.Model([vgg\_network.input],

[gram\_matrix(vgg\_network.get\_layer(name).output) for name in style\_layers])

content\_extractor = keras.Model([vgg\_network.input],

[vgg\_network.get\_layer(name).output for name in content\_layers])

output\_image = tf.Variable(content\_image\_processed)

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

# G(style\_image)

style\_targets = style\_extractor(style\_image\_processed)

# F(content\_image)

content\_targets = content\_extractor(content\_image\_processed)

def train\_step():

with tf.GradientTape() as tape:

# G(output\_image)

style\_of\_output = style\_extractor(output\_image)

# F(output\_image)

content\_of\_output = content\_extractor(output\_image)

# style loss

style\_loss = [tf.reduce\_mean((output - target) \*\* 2)

for output,target in zip(style\_of\_output, style\_targets)]

style\_loss = style\_weight \* tf.reduce\_mean(style\_loss)

# content loss

content\_loss = [tf.reduce\_mean((output - target) \*\* 2)

for output,target in zip(content\_of\_output, content\_targets)]

content\_loss = content\_weight \* tf.reduce\_mean(content\_loss)

# variation loss

row\_deltas, col\_deltas = 0., 0.

for i in range(5):

row\_deltas += (output\_image[:, :, i:-5+i, :] - output\_image[:, :, 5:, :])

col\_deltas += (output\_image[:, i:-5+i, :, :] - output\_image[:, 5:, :, :])

variation\_loss = variation\_weight \* (tf.reduce\_sum(tf.abs(row\_deltas)) + tf.reduce\_sum(tf.abs(col\_deltas)))

loss = style\_loss + content\_loss + variation\_loss

print\_with\_time("| {:10.1f} | {:10.1f} | {:12.1f} | {:13.1f} |".format(

loss.numpy(), style\_loss.numpy(), content\_loss.numpy(),

variation\_loss.numpy()))

# minize the loss by the optimizer

grad = tape.gradient(loss, output\_image)

opt.apply\_gradients([(grad, output\_image)])

print\_with\_time("| total\_loss | style\_loss | content\_loss | variation\_loss |")

for i in range(ITERATIONS+1):

train\_step()

if i%10 == 0:

file\_name = "output{}.jpg".format(i)

# depreprocess it

output = output\_image[0, :, :, ::-1]

output = output.numpy()

output += IMAGENET\_MEAN\_RGB\_VALUES

output[output<0.] = 0.

output[output>255.] = 255

image.array\_to\_img(output).save(file\_name)

print\_with\_time('save to ' + file\_name)