



Save the date: Google I/O returns May 18-20. Register now (<https://events.google.com/io/>).

# 神经风格迁移



在  
Google  
Colab  
上运行

([https://colab.research.google.com/github/tensorflow/docs-site/blob/master/site/zh-cn/tutorials/generative/style\\_transfer.ipynb](https://colab.research.google.com/github/tensorflow/docs-site/blob/master/site/zh-cn/tutorials/generative/style_transfer.ipynb))



在  
GitHub  
上查看  
源代码

([https://github.com/tensorflow/docs-site/blob/master/site/zh-cn/tutorials/generative/style\\_transfer.ipynb](https://github.com/tensorflow/docs-site/blob/master/site/zh-cn/tutorials/generative/style_transfer.ipynb))

我们的 TensorFlow 社区翻译了这些文档。因为社区翻译是尽力而为，所以无法保证它们是最准确的，并且我们的 官方英文文档 (<https://tensorflow.google.cn/?hl=en>)。如果您有改进此翻译的建议，请提交 pull request 到 <https://github.com/tensorflow/docs> GitHub 仓库。要志愿地撰写或者审核译文，请加入 [docs tensorflow.org Google Group](https://groups.google.com/a/tensorflow.org/forum/#!forum/docs-zh-cn) (<https://groups.google.com/a/tensorflow.org/forum/#!forum/docs-zh-cn>)。

本教程使用深度学习来用其他图像的风格创建一个图像（曾经你是否希望可以像毕加索或梵高一样绘画？）。这被称为神经风格迁移，该技术概述于 [A Neural Algorithm of Artistic Style](https://arxiv.org/abs/1508.06576) (<https://arxiv.org/abs/1508.06576>) (Gatys et al.)。

本教程演示了原始的风格迁移算法。它将图像内容优化为特定样式。最新的一些方法训练模型以直接生成类似于 [cyclegan](https://tensorflow.google.cn/tutorials/generative/cyclegan) ([/tutorials/generative/cyclegan](https://tensorflow.google.cn/tutorials/generative/cyclegan))。原始的这种方法要快得多（高达 1000 倍）。[TensorFlow Hub](https://tensorflow.google.cn/hub) (<https://tensorflow.google.cn/hub>) 和 [TensorFlow Lite](https://tensorflow.google.cn/lite/models/style_transfer/overview) ([https://tensorflow.google.cn/lite/models/style\\_transfer/overview](https://tensorflow.google.cn/lite/models/style_transfer/overview)) 中提供了预训练的任意图像风格化模块 ([https://colab.sandbox.google.com/github/tensorflow/hub/blob/master/examples/colab/tf2\\_arbitrary\\_image\\_style\\_transfer.ipynb](https://colab.sandbox.google.com/github/tensorflow/hub/blob/master/examples/colab/tf2_arbitrary_image_style_transfer.ipynb))。

神经风格迁移是一种优化技术，用于将两个图像——一个内容图像和一个风格参考图像（如著名画家的一个作品）——混合在一起，使输出的图像看起来像内容图像，但是用了风格参考图像的风格。

这是通过优化输出图像以匹配内容图像的内容统计数据和风格参考图像的风格统计数据来实现的。这些统计数据可以使用卷积网络从图像中提取。

例如，我们选取这张小狗的照片和 Wassily Kandinsky 的作品 7：



黄色拉布拉多犬的凝视 ([https://commons.wikimedia.org/wiki/File:YellowLabradorLooking\\_new.jpg](https://commons.wikimedia.org/wiki/File:YellowLabradorLooking_new.jpg))  
，来自 Wikimedia Commons



如果 Kandinsky 决定用这种风格来专门描绘这只海龟会是什么样子？ 是否如下图一样？





## 配置

### 导入和配置模块

```
import tensorflow as tf

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)
```

下载图像并选择风格图像和内容图像:

```
content_path = tf.keras.utils.get_file('YellowLabradorLooking_new.jpg', 'http:
# https://commons.wikimedia.org/wiki/File:Vassily_Kandinsky,_1913_-_Compositi
style_path = tf.keras.utils.get_file('kandinsky5.jpg', 'https://storage.google:
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/1
90112/83281 [=====] - 0s 0us/step
Downloading data from https://storage.googleapis.com/download.tensorflow.org/1
196608/195196 [=====] - 0s 0us/step
```

## 将输入可视化

定义一个加载图像的函数，并将其最大尺寸限制为 512 像素。

```
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
```

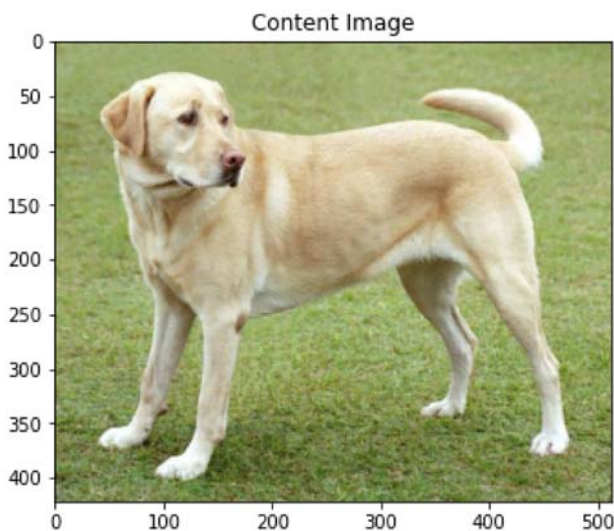
创建一个简单的函数来显示图像：

```
def imshow(image, title=None):  
    if len(image.shape) > 3:  
        image = tf.squeeze(image, axis=0)  
  
    plt.imshow(image)  
    if title:  
        plt.title(title)
```

```
content_image = load_img(content_path)  
style_image = load_img(style_path)
```

```
plt.subplot(1, 2, 1)  
imshow(content_image, 'Content Image')
```

```
plt.subplot(1, 2, 2)  
imshow(style_image, 'Style Image')
```

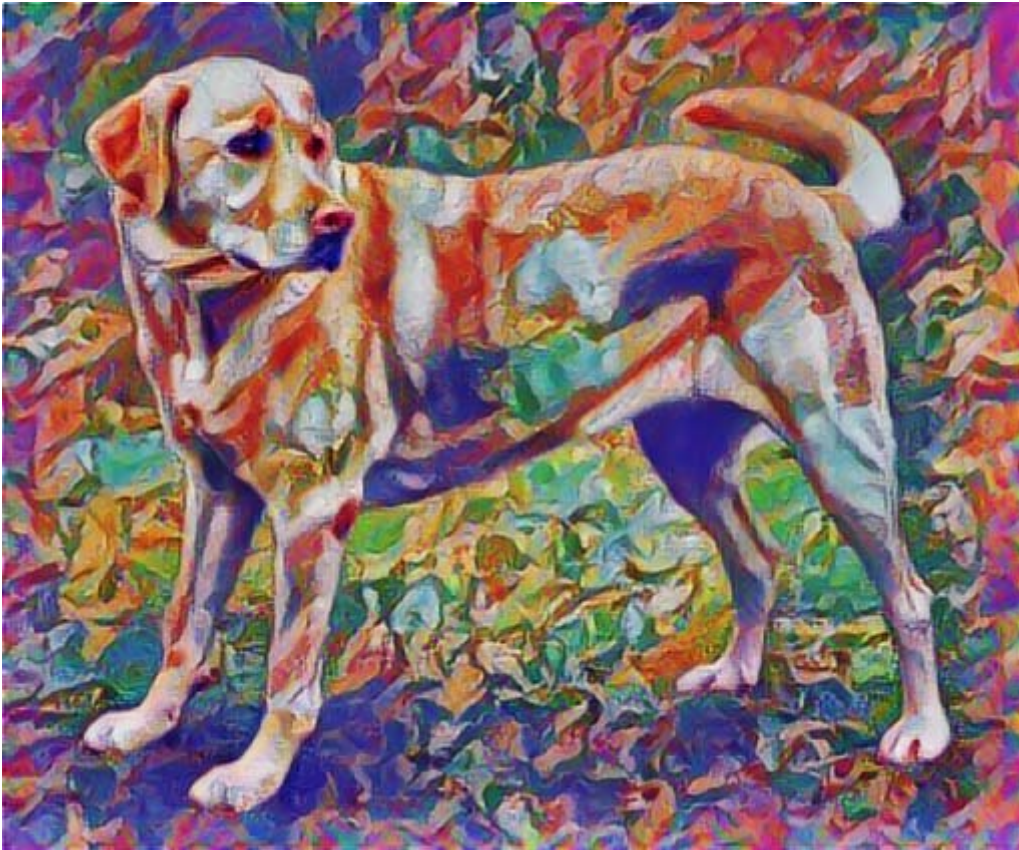


## 使用 TF-Hub 进行快速风格迁移

本教程演示了原始的风格迁移算法。其将图像内容优化为特定风格。在进入细节之前，让我们看一下 [TensorFlow Hub](https://tensorflow.google.cn/hub) (<https://tensorflow.google.cn/hub>) 模块如何快速风格迁移：



```
import tensorflow_hub as hub
hub_module = hub.load('https://tfhub.dev/google/magenta/arbitrary-image-styliz:
stylized_image = hub_module(tf.constant(content_image), tf.constant(style_image))
tensor_to_image(stylized_image)
```



## 定义内容和风格的表示

使用模型的中间层来获取图像的内容和风格表示。从网络的输入层开始，前几个层的激励响应表示边缘和纹理等低级 feature (特征)。随着层数加深，最后几层代表更高级的 feature (特征)——实体的部分，如轮子或眼睛。在此教程中，我们使用的是 VGG19 网络结构，这是一个已经预训练好的图像分类网络。这些中间层是从图像中定义内容和风格的表示所必需的。对于一个输入图像，我们尝试匹配这些中间层的相应风格和-content 目标的表示。

加载 VGG19 (<https://keras.io/applications/#vgg19>) 并在我们的图像上测试它以确保正常运行：

```
x = tf.keras.applications.vgg19.preprocess_input(content_image*255)
x = tf.image.resize(x, (224, 224))
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_p
[(class_name, prob) for (number, class_name, prob) in predicted_top_5]
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/4
40960/35363 [=====] - 0s 0us/step
[('Labrador_retriever', 0.493171),
 ('golden_retriever', 0.23665288),
 ('kuvasz', 0.036357544),
 ('Chesapeake_Bay_retriever', 0.024182763),
 ('Greater_Swiss_Mountain_dog', 0.0186461)]
```

现在，加载没有分类部分的 VGG19，并列出各层的名称：

```
vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applic
80142336/80134624 [=====] - 1s 0us/step
```

```
input_2
block1_conv1
block1_conv2
block1_pool
block2_conv1
block2_conv2
block2_pool
block3_conv1
block3_conv2
block3_conv3
```

从网络中选择中间层的输出以表示图像的风格和内容：

```
# 内容层将提取出我们的 feature maps （特征图）
content_layers = ['block5_conv2']

# 我们感兴趣的风格层
style_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'block5_conv1']

num_content_layers = len(content_layers)
num_style_layers = len(style_layers)
```

用于表示风格和内容的中间层

那么,为什么我们预训练的图像分类网络中的这些中间层的输出允许我们定义风格和内容的表示?

从高层理解,为了使网络能够实现图像分类(该网络已被训练过),它必须理解图像。这需要将原始图像作为输入像素并构建内部表示,这个内部表示将原始图像像素转换为对图像中存在的 feature (特征)的复杂理解。

这也是卷积神经网络能够很好地推广的一个原因:它们能够捕获不变性并定义类别(例如猫与狗)之间的 feature (特征),这些 feature (特征)与背景噪声和其他干扰无关。因此,将原始图像传递到模型输入和分类标签输出之间的某处的这一过程,可以视作复杂的 feature (特征)提取器。通过这些模型的中间层,我们就可以描述输入图像的内容和风格。

## 建立模型

使用 **tf.keras.applications** ([https://www.tensorflow.org/api\\_docs/python/tf/keras/applications](https://www.tensorflow.org/api_docs/python/tf/keras/applications)) 中的网络可以让我们非常方便的利用Keras的功能接口提取中间层的值。

在使用功能接口定义模型时,我们需要指定输入和输出:

```
model = Model(inputs, outputs)
```

以下函数构建了一个 VGG19 模型,该模型返回一个中间层输出的列表:



```
def vgg_layers(layer_names):
    """ Creates a vgg model that returns a list of intermediate output values."
    # 加载我们的模型。 加载已经在 imagenet 数据上预训练的 VGG
    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
```

然后建立模型:

```
style_extractor = vgg_layers(style_layers)
style_outputs = style_extractor(style_image*255)

#查看每层输出的统计信息
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, 336, 512, 64)
  min: 0.0
  max: 835.5256
  mean: 33.97525

block2_conv1
  shape: (1, 168, 256, 128)
  min: 0.0
  max: 4625.8857
  mean: 199.82687

block3_conv1
  shape: (1, 84, 128, 256)
```

## 风格计算

图像的内容由中间 feature maps (特征图)的值表示。

事实证明，图像的风格可以通过不同 feature maps (特征图)上的平均值和相关性来描述。通过在每个位置计算 feature (特征)向量的外积，并在所有位置对该外积进行平均,可以计算出包含此信息的 Gram 矩阵。对于特定层的 Gram 矩阵，具体计算方法如下所示：

$$G_{cd}^l = \frac{\sum_{ij} F_{ijc}^l(x) F_{ijd}^l(x)}{IJ}$$

这可以使用`tf.linalg.einsum` ([https://www.tensorflow.org/api\\_docs/python/tf/einsum](https://www.tensorflow.org/api_docs/python/tf/einsum))函数来实现：

```
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)
```

## 提取风格和内容

构建一个返回风格和内容张量的模型。

```
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}

```

在图像上调用此模型，可以返回 style\_layers 的 gram 矩阵（风格）和 content\_layers 的内容：

```

extractor = StyleContentModel(style_layers, content_layers)

results = extractor(tf.constant(content_image))

style_results = results['style']

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:  0.0055228462
    max:  28014.562
    mean:  263.79025

  block2_conv1

```

```

shape: (1, 128, 128)
min: 0.0
max: 61479.49
mean: 9100.949

```

```
block2_conv1
```

## 梯度下降

使用此风格和内容提取器，我们现在可以实现风格传输算法。我们通过计算每个图像的输出和目标的均方误差来做到这一点，然后取这些损失值的加权和。

设置风格和内容目标值：

```

style_targets = extractor(style_image)['style']
content_targets = extractor(content_image)['content']

```

定义一个 **`tf.Variable`** ([https://www.tensorflow.org/api\\_docs/python/tf/Variable](https://www.tensorflow.org/api_docs/python/tf/Variable)) 来表示要优化的图像。为了快速实现这一点，使用内容图像对其进行初始化（**`tf.Variable`** ([https://www.tensorflow.org/api\\_docs/python/tf/Variable](https://www.tensorflow.org/api_docs/python/tf/Variable)) 必须与内容图像的形状相同）

```
image = tf.Variable(content_image)
```

由于这是一个浮点图像，因此我们定义一个函数来保持像素值在 0 和 1 之间：

```

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

```

创建一个 optimizer 。本教程推荐 LBFGS，但 Adam 也可以正常工作：

```
opt = tf.optimizers.Adam(learning_rate=0.02, beta_1=0.99, epsilon=1e-1)
```

为了优化它，我们使用两个损失的加权组合来获得总损失：

```

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[name])
                                         for name in content_outputs.keys())])
    content_loss *= content_weight / num_content_layers
    loss = style_loss + content_loss
    return loss
```

使用 **tf.GradientTape** ([https://www.tensorflow.org/api\\_docs/python/tf/GradientTape](https://www.tensorflow.org/api_docs/python/tf/GradientTape)) 来更新图像。

```
@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))
```

现在，我们运行几个步来测试一下：

```
train_step(image)
train_step(image)
train_step(image)
tensor_to_image(image)
```



运行正常，我们来执行一个更长的优化：

```
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='')
    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: 20.4

## 总变分损失

此实现只是一个基础版本，它的一个缺点是它会产生大量的高频误差。我们可以直接通过正则化图像的高频分量来减少这些高频误差。在风格转移中，这通常被称为总变分损失：

```
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")

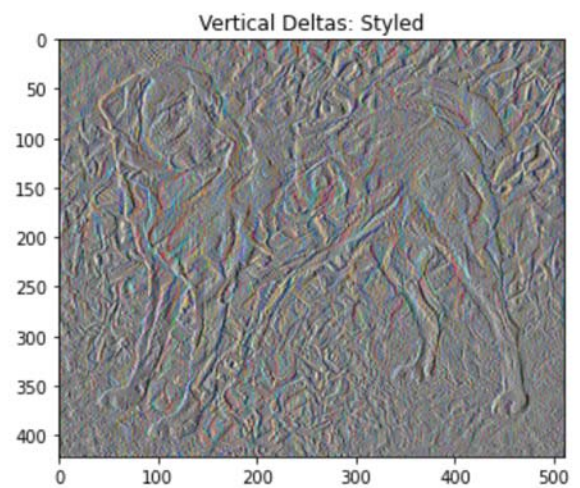
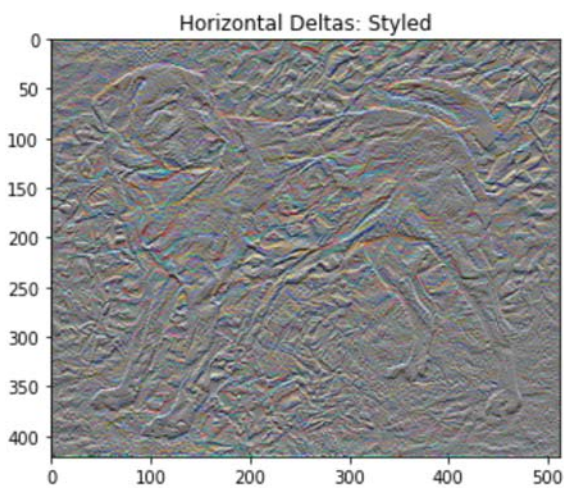
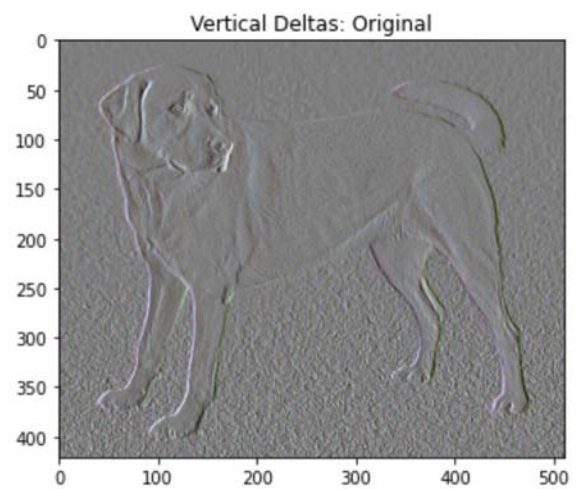
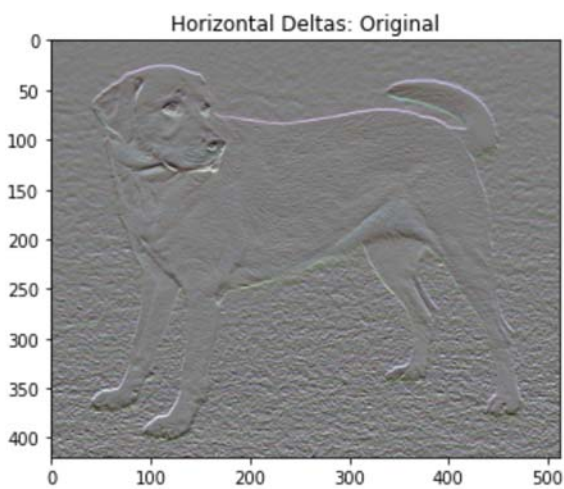
plt.subplot(2,2,2)
imshow(clip_0_1(2*x_deltas+0.5), "Vertical Deltas: Original")

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")

```



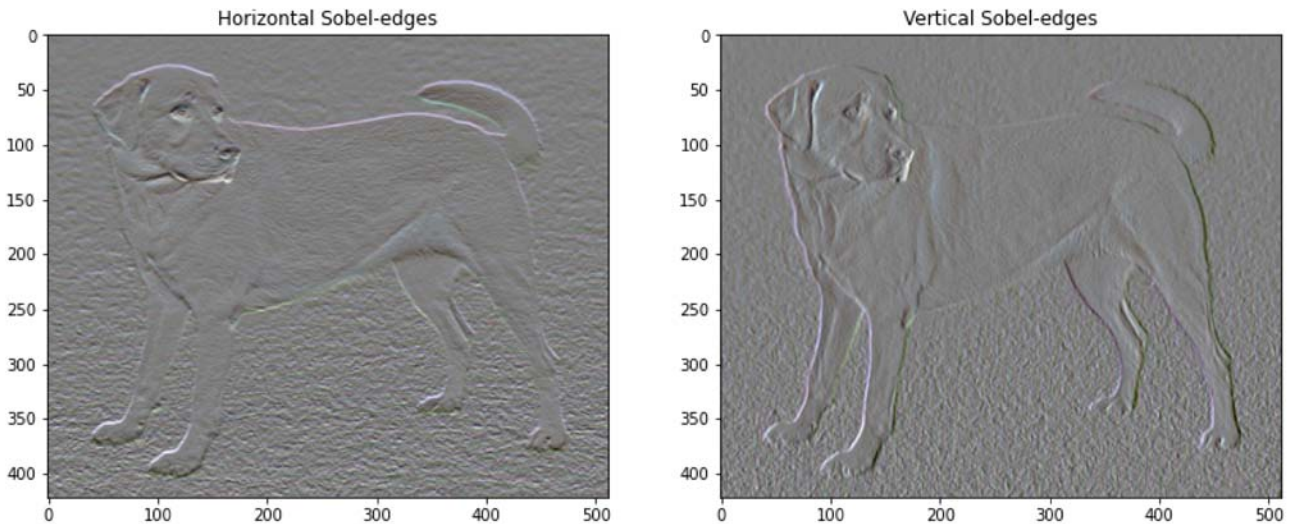
这显示了高频分量如何增加。

而且，本质上高频分量是一个边缘检测器。我们可以从 Sobel 边缘检测器获得类似的输出，例如：

```
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")
```



与此相关的正则化损失是这些值的平方和：

```
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()
```

```
149342.6
```

以上说明了总变分损失的用途。但是无需自己实现，因为 TensorFlow 包含了一个标准实现：

```
tf.image.total_variation(image).numpy()
```

```
array([149342.6], dtype=float32)
```

## 重新进行优化

选择 `total_variation_loss` 的权重:

```
total_variation_weight=30
```

现在, 将它加入 `train_step` 函数中:

```
@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))
```

重新初始化优化的变量:

```
image = tf.Variable(content_image)
```

并进行优化:

```
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='')
    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: 21.7
```

最后，保存结果：

```
file_name = 'stylized-image.png'
tensor_to_image(image).save(file_name)

try:
    from google.colab import files
except ImportError:
    pass
else:
    files.download(file_name)
```

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 4.0 License](https://creativecommons.org/licenses/by/4.0/) (<https://creativecommons.org/licenses/by/4.0/>), and code samples are licensed under the [Apache 2.0 License](https://www.apache.org/licenses/LICENSE-2.0) (<https://www.apache.org/licenses/LICENSE-2.0>). For details, see the [Google Developers Site](https://developers.google.com/terms)

Policies (<https://developers.google.com/site-policies>). Java is a registered trademark of Oracle and/or its affiliates.

Last updated 2021-03-22 UTC.