

Style Transfer



La idea de este trabajo final es reproducir el siguiente paper:

<https://arxiv.org/pdf/1508.06576.pdf> (<https://arxiv.org/pdf/1508.06576.pdf>)

El objetivo es transferir el estilo de una imagen dada a otra imagen distinta.

Como hemos visto en clase, las primeras capas de una red convolucional se activan ante la presencia de ciertos patrones vinculados a detalles muy pequeños.

A medida que avanzamos en las distintas capas de una red neuronal convolucional, los filtros se van activando a medida que detectan patrones de formas cada vez más complejos.

Lo que propone este paper es asignarle a la activación de las primeras capas de una red neuronal convolucional (por ejemplo VGG19) la definición del estilo y a la activación de las últimas capas de la red neuronal convolucional, la definición del contenido.

La idea de este paper es, a partir de dos imágenes (una que aporte el estilo y otra que aporte el contenido) analizar cómo es la activación de las primeras capas para la imagen que aporta el estilo y cómo es la activación de las últimas capas de la red convolucional para la imagen que aporta el contenido. A partir de esto se intentará sintetizar una imagen que active los filtros de las primeras capas que se activaron con la imagen que aporta el estilo y los filtros de las últimas capas que se activaron con la imagen que aporta el contenido.

A este procedimiento se lo denomina neural style transfer.

En este trabajo se deberá leer el paper mencionado y en base a ello, entender la implementación que se muestra a continuación y contestar preguntas sobre la misma.

Una metodología posible es hacer una lectura rápida del paper (aunque esto signifique no entender algunos detalles del mismo) y luego ir analizando el código y respondiendo las preguntas. A medida que se planteen las preguntas, volviendo a leer secciones específicas

del paper terminará de entender los detalles que pudieran haber quedado pendientes.

Lo primero que haremos es cargar dos imágenes, una que aporte el estilo y otra que aporte el contenido.

```
In [7]: # Imagen para estilo
!wget https://upload.wikimedia.org/wikipedia/commons/5/52/La_noche_estrellada1.jpg

# Imagen para contenido
!wget https://upload.wikimedia.org/wikipedia/commons/thumb/f/f4/Neckarfront_T%C3%BCbingen_Mai_2017.jpg/775px-Neckarfront_T%C3%BCbingen_Mai_2017.jpg

# Creamos el directorio para los archivos de salida
!mkdir /content/output

--2020-09-26 14:51:36-- https://upload.wikimedia.org/wikipedia/commons/5/52/La_noche_estrellada1.jpg (https://upload.wikimedia.org/wikipedia/commons/5/52/La_noche_estrellada1.jpg)
Resolviendo upload.wikimedia.org (upload.wikimedia.org)... 208.80.153.240, 2620:0:860:ed1a::2:b
Conectando con upload.wikimedia.org (upload.wikimedia.org)[208.80.153.240]:443... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 223725 (218K) [image/jpeg]
Guardando como: "La_noche_estrellada1.jpg.1"

La_noche_estrellada 100%[=====] 218,48K 287KB/s en 0,8s

2020-09-26 14:51:38 (287 KB/s) - "La_noche_estrellada1.jpg.1" guardado [223 725/223725]

--2020-09-26 14:51:38-- https://upload.wikimedia.org/wikipedia/commons/thumb/f/f4/Neckarfront_T%C3%BCbingen_Mai_2017.jpg/775px-Neckarfront_T%C3%BCbingen_Mai_2017.jpg (https://upload.wikimedia.org/wikipedia/commons/thumb/f/f4/Neckarfront_T%C3%BCbingen_Mai_2017.jpg/775px-Neckarfront_T%C3%BCbingen_Mai_2017.jpg)
Resolviendo upload.wikimedia.org (upload.wikimedia.org)... 208.80.153.240, 2620:0:860:ed1a::2:b
Conectando con upload.wikimedia.org (upload.wikimedia.org)[208.80.153.240]:443... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 153015 (149K) [image/jpeg]
Guardando como: "775px-Neckarfront_T%C3%BCbingen_Mai_2017.jpg.1"

775px-Neckarfront_T%C3%BCbingen_Mai_2017.jpg 100%[=====] 149,43K 196KB/s en 0,8s

2020-09-26 14:51:40 (196 KB/s) - "775px-Neckarfront_T%C3%BCbingen_Mai_2017.jpg.1" guardado [153015/153015]

mkdir: no se puede crear el directorio «/content/output»: No existe el archivo o el directorio
```

```
In [8]: from keras.preprocessing.image import load_img, save_img, img_to_array
import numpy as np
from scipy.optimize import fmin_l_bfgs_b
import time
import argparse

from keras.applications import vgg19
from keras import backend as K
from pathlib import Path
import tensorflow as tf
tf.compat.v1.disable_eager_execution()
```

Definimos las imágenes que vamos a utilizar, y el dispositivo en el que se ejecutarán.

directorio de salida.

```
In [9]: base_image_path = Path("/content/775px-Neckarfront_Tübingen_Mai_2017.jpg") #base image
style_reference_image_path = Path("/content/La_noche_estrellada1.jpg") #este es el estilo
result_prefix = Path("/content/output") # carpeta donde estan las imagenes, i
iterations = 100
```

1) En base a lo visto en el paper ¿Qué significan los parámetros definidos en la siguiente celda?

Respuesta:

- Total_variation_weight: Este factor no esta definido en el paper.
- style_weight: factor que controla la proporcion de aporte de estilo en la loss total, el mismo multiplicara el componente de estilo de la loss total (beta).
- content_weight: de forma similar al anterior determinara el impacto que tendra la loss del contexto a la loss total(alpha).

```
In [10]: # con estos 3 puedes jugar para determinar el estilo final
total_variation_weight = 0.3
style_weight = 0.95
content_weight = 0.5
```

```
In [ ]: load_img(base_image_path).size
```

```
In [ ]: # Definimos el tamaño de las imágenes a utilizar
width, height = load_img(base_image_path).size
img_nrows = 400 # si aumentas este aumenta el tamaño de la imagen de salida
img_ncols = int(width * img_nrows / height)
```

2) Explicar qué hace la siguiente celda. En especial las últimas dos líneas de la función antes del return. ¿Por qué?

Ayuda: <https://keras.io/applications/>

Respuesta:

- En la implementacion en keras de la vgg19 se requiere un preprocessamiento de los inputs para que coincida con el shape del resto de la red. En el caso de la ultima linea es un metodo que adapta el formato de la imagen al modo "cafe" que es en el que inicialmente se implemento el Transfer Style.

```
In [7]: def preprocess_image(image_path):
    img = load_img(image_path, target_size=(img_nrows, img_ncols))
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = vgg19.preprocess_input(img)
    return img
```

3) Habiendo comprendido lo que hace la celda anterior, explique de manera muy concisa qué hace la siguiente celda. ¿Qué relación tiene con la celda anterior?

Respuesta:

- Para poder tener una imagen en un formato mas estandar se desprocesa nuevamente de caffe a alto, ancho, canales para guardarla como imagen.

```
In [8]: def deprocess_image(x):
    x = x.reshape((img_nrows, img_ncols, 3))
    # Remove zero-center by mean pixel
    x[:, :, 0] += 103.939
    x[:, :, 1] += 116.779
    x[:, :, 2] += 123.68
    # 'BGR'->'RGB'
    x = x[:, :, ::-1]
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

```
In [9]: # get tensor representations of our images
# K.variable convierte un numpy array en un tensor, para
base_image = K.variable(preprocess_image(base_image_path))
style_reference_image = K.variable(preprocess_image(style_reference_image_path))
```

```
In [10]: combination_image = K.placeholder((1, img_nrows, img_ncols, 3))
```

Aclaración:

La siguiente celda sirve para procesar las tres imágenes (contenido, estilo y salida) en un solo batch.

```
In [11]: # combine the 3 images into a single Keras tensor
input_tensor = K.concatenate([base_image,
                             style_reference_image,
                             combination_image], axis=0)
```

```
In [12]: # build the VGG19 network with our 3 images as input
# the model will be loaded with pre-trained ImageNet weights
model = vgg19.VGG19(input_tensor=input_tensor,
                     weights='imagenet', include_top=False)
print('Model loaded.')

# get the symbolic outputs of each "key" layer (we gave them unique names).
outputs_dict = dict([(layer.name, layer.output) for layer in model.layers])
Model loaded.
```

4) En la siguientes celdas:

- ¿Qué es la matriz de Gram? ¿Para qué se usa?
 - Es una matriz que contiene la multiplicacion escalar de un vector por el mismo transpuesto, brindando una independencia o correlacion de los datos. Aca se usa en la función de costo precisamente para determinar que tan parecido es el estilo de una imagen de la otra.
- ¿Por qué se permutan las dimensiones de x?
 - No estoy seguro pero, para me parece que es para darle mas relevancia a los canales que al alto y ancho.

```
In [13]: def gram_matrix(x):
    features = K.batch_flatten(K.permute_dimensions(x, (2, 0, 1)))
    gram = K.dot(features, K.transpose(features))
    return gram
```

5) Losses:

Explicar qué mide cada una de las losses en las siguientes tres celdas.

Rta:

- Style_loss: mide el error o la diferencia entre la imagen que proporciona estilo y la generada, mediante la matriz de Gram, esta normalizado por las dimensiones de las mismas
- content_loss: mide el error cuadrado entre la imagen de contexto y la generada.
- total_variation_loss: Parece ser un regulador que sustituye la imagen de ruido blanco que no se está incluyendo en esta implementación.

```
In [14]: def style_loss(style, combination):
    assert K.ndim(style) == 3
    assert K.ndim(combination) == 3
    S = gram_matrix(style)
    C = gram_matrix(combination)
    channels = 3
    size = img_nrows * img_ncols
    return K.sum(K.square(S - C)) / (4.0 * (channels ** 2) * (size ** 2))
```

```
In [15]: def content_loss(base, combination):
    return K.sum(K.square(combination - base))
```

```
In [16]: def total_variation_loss(x):
    assert K.ndim(x) == 4
    a = K.square(
        x[:, :img_nrows - 1, :img_ncols - 1, :] - x[:, 1:, :img_ncols - 1, :])
    b = K.square(
        x[:, :img_nrows - 1, :img_ncols - 1, :] - x[:, :img_nrows - 1, 1:, :])
    return K.sum(K.pow(a + b, 1.25))
```

```
In [ ]: outputs_dict
```

```
In [17]: # Armamos la loss total
loss = K.variable(0.0)
layer_features = outputs_dict['block5_conv2'] # SI modificas este de acuerdo
                                              # cambia el resultado final
base_image_features = layer_features[0, :, :, :]
combination_features = layer_features[2, :, :, :]
loss = loss + content_weight * content_loss(base_image_features, combination_features)

feature_layers = ['block1_conv1', 'block2_conv1',
                  'block3_conv1', 'block4_conv1',
                  'block5_conv1'] ## SI modificas este de acuerdo a los
                      # cambia el resultado final
for layer_name in feature_layers:
    layer_features = outputs_dict[layer_name]
    style_reference_features = layer_features[1, :, :, :]
    combination_features = layer_features[2, :, :, :]
    sl = style_loss(style_reference_features, combination_features)
    loss = loss + (style_weight / len(feature_layers)) * sl
loss = loss + total_variation_weight * total_variation_loss(combination_image)
```

```
In [17]: 
```

```
In [18]: grads = K.gradients(loss, combination_image)

outputs = [loss]
if isinstance(grads, (list, tuple)):
    outputs += grads
else:
    outputs.append(grads)
print(outputs)
[<tf.Tensor 'add_7:0' shape=() dtype=float32>, <tf.Tensor 'gradients/AddN_16:0' shape=(1, 400, 517, 3) dtype=float32>, <tf.Tensor 'gradients/AddN_16:0' shape=(1, 400, 517, 3) dtype=float32>]
```

In [19]: `f_outputs = K.function([combination_image], outputs)`

6) Explique el propósito de las siguientes tres celdas. ¿Qué hace la función fmin_l_bfgs_b? ¿En qué se diferencia con la implementación del paper? ¿Se puede utilizar alguna alternativa?

Respuesta:

- Se crea una función para evaluar la loss y el gradiente de la imagen evaluada en la K.function.
- Se crea una clase Evaluator que inicializa la loss y el gradiente en None y usa la f_outputs para devolver el valor de la loss y del gradiente para una imagen evaluada.
- Se crea el loop de aprendizaje, en el cual se aprovecha la función fmin_l_bfgs_b de scipy como optimizador de la función de costo.
- La función fmin_l_bfgs_b sirve para minimizar la función de costo usando el algoritmo L-BFGS-B, sería nuestro optimizador, nos actualiza la imagen y el mínimo de la función.
- Difiere del paper en que en esta implementación se incluye una función de costo total donde se agrega un componente de distorsión de la imagen en lugar de usar una imagen de ruido blanco sobre la imagen de contexto.
- Si, se podría hacer la extracción de features a otra altura y agregar capas distintas, ejemplo salir en Block5_conv2 y agregar el averagepooling, lo que tendría un mayor impacto en el resultado, podría usarse otros optimizadores de la función de costo.

In [20]: `def eval_loss_and_grads(x):
 x = x.reshape((1, img_nrows, img_ncols, 3))
 outs = f_outputs([x])
 loss_value = outs[0]
 if len(outs[1:]) == 1:
 grad_values = outs[1].flatten().astype('float64')
 else:
 grad_values = np.array(outs[1:]).flatten().astype('float64')
 return loss_value, grad_values

this Evaluator class makes it possible
to compute loss and gradients in one pass
while retrieving them via two separate functions,
"loss" and "grads". This is done because scipy.optimize
requires separate functions for loss and gradients,
but computing them separately would be inefficient`

In [21]: `class Evaluator(object):

 def __init__(self):
 self.loss_value = None
 self.grads_values = None

 def loss(self, x):
 assert self.loss_value is None
 loss_value, grad_values = eval_loss_and_grads(x)
 self.loss_value = loss_value
 self.grads_values = grad_values
 return self.loss_value

 def grads(self, x):
 assert self.loss_value is not None
 grad_values = np.copy(self.grads_values)
 self.loss_value = None
 self.grads_values = None
 return grad_values`

7) Ejecute la siguiente celda y observe las imágenes de salida en cada iteración.

```
In [22]: evaluator = Evaluator()

# run scipy-based optimization (L-BFGS) over the pixels of the generated image
# so as to minimize the neural style loss
x = preprocess_image(base_image_path)

for i in range(100):
    print('Start of iteration', i)
    start_time = time.time()
    x, min_val, info = fmin_l_bfgs_b(evaluator.loss, x.flatten(),
                                       fprime=evaluator.grads, maxfun=20)
    print('Current loss value:', min_val)
    # save current generated image
    img = deprocess_image(x.copy())
    fname = result_prefix / ('output_at_iteration_%d_6.png' % i)
    save_img(fname, img)
    end_time = time.time()
    print('Image saved as', fname)
    print('Iteration %d completed in %.1f s' % (i, end_time - start_time))
```

```
Start of iteration 0
Current loss value: 2816652300.0
Image saved as /content/output/output_at_iteration_0_6.png
Iteration 0 completed in 10s
Start of iteration 1
Current loss value: 1751025700.0
Image saved as /content/output/output_at_iteration_1_6.png
Iteration 1 completed in 6s
Start of iteration 2
Current loss value: 1436789600.0
Image saved as /content/output/output_at_iteration_2_6.png
Iteration 2 completed in 6s
Start of iteration 3
Current loss value: 1294916600.0
Image saved as /content/output/output_at_iteration_3_6.png
Iteration 3 completed in 6s
Start of iteration 4
Current loss value: 1200080300.0
Image saved as /content/output/output_at_iteration_4_6.png
Iteration 4 completed in 6s
```

8) Generar imágenes para distintas combinaciones de pesos de las losses. Explicar las diferencias. (Adjuntar las imágenes generadas como archivos separados.)

Respuesta:

1 Salida original



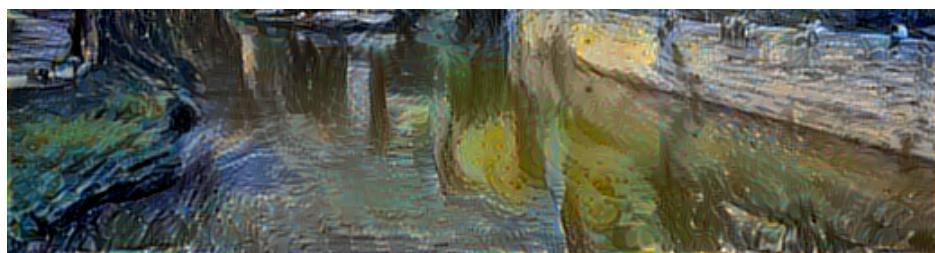


2 Aumentando el peso del contexto y disminuyendo el del estilo; no se notan cambios relevantes



3 Aumentando aun mas el peso del contexto y disminuyendo el del estilo; no se identifican cambios notables.





4 Aumentando la cantidad de iteraciones.



5 Colocando los pesos por debajo de 1, 0.9 para estilo y 0.5 para contexto, se aprecia cambio de saturacion en los colores y lineas mas suaves



9) Cambiar las imágenes de contenido y estilo por imágenes elegidas por usted. Adjuntar el resultado.

In [6]: [# Resuesta](#)

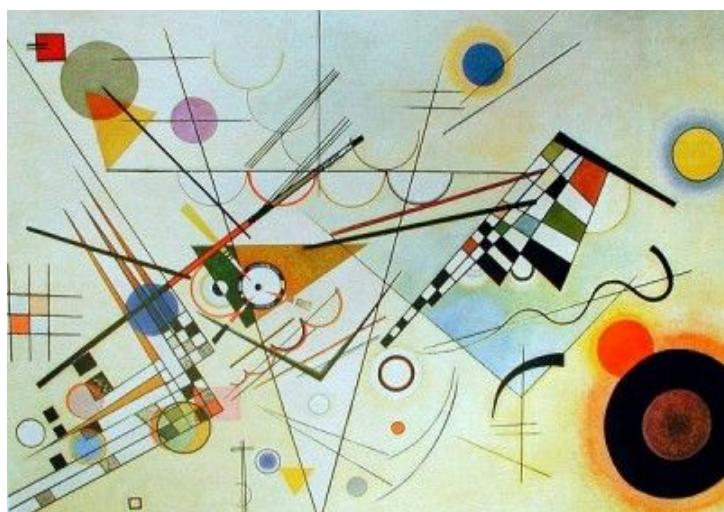
In [7]: [# INPUT 1](#)

1 Imagen de contexto:



In [9]: [# INPUT 2](#)

2 Imagen de estilo:



In [11]: [# OUTPUT](#)

3 Imagen generada:



In []: