

# Transferencia de estilo

Sara Olías Zapico

6 de junio de 2021

## Índice

1	Introducción	2
2	Redes Neuronales Convolucionales	4
2.1	Redes Neuronales Convencionales . . . . .	4
2.2	Redes Neuronales Convolucionales . . . . .	4
3	Método propuesto	6
3.1	Representación de contenido . . . . .	6
3.2	Representación de estilo . . . . .	7
3.3	Transferencia de estilo . . . . .	8
4	Resultados	10
4.1	Variaciones de estilo y contenido . . . . .	14
4.2	Variacion del orden de las capas . . . . .	15
4.3	Inicialización del descenso de gradiente . . . . .	16
4.4	Transferencia de estilo fotorrealista . . . . .	18
5	Código	19
6	Conclusiones	24
	Bibliografía	25

## 1. Introducción

En el siguiente trabajo se presentan los resultados de los artículos **ArtFlow: Unbiased Image Style Transfer via Reversible Neural Flows**, de Jie An, Siyu Huang, Yibing Song, Dejing Dou, Wei Liu y Jiebo Luo [1]; y **Image Style Transfer Using Convolutional Neural Networks**, de Leon A. Gatys [2]. Además, se han realizado varios experimentos de transferencia de estilo utilizando para ello el código propuesto por Leon A. Gatys.

La transferencia de estilo es una tarea importante en edición de imágenes ya que nos permite la creación de nuevas obras artísticas. Para ello, dadas dos imágenes de entrada, la imagen de contenido y la imagen de estilo, se pretende sintetizar una nueva imagen que, conserve el contenido de la primera imagen pero utilizando las características de estilo de la segunda imagen. El desafío clave aquí es, cómo extraer representaciones efectivas del estilo para posteriormente combinarlo en la imagen del contenido.

En este trabajo, veremos la teoría y el funcionamiento de las redes neuronales convolucionales, en qué consiste lo que llamamos transferencia de estilo y algunas aplicaciones de la transferencia de estilo (tanto las que se incluyen en el artículo como algunas generadas por nosotros).

Es decir, dada una imagen de ejemplo, queremos crear de forma automática una variante de esta imagen que se vea similar, pero con una estructura diferente. Para ello, dividimos la entrada en una imagen estilo y una imagen contenido. La primera imagen describe los bloques de construcción sobre los que se debe crear la imagen de salida y la segunda restringe su diseño. En la siguiente figura podemos observar un ejemplo de imágenes transferidas.



Figura 1: Teniendo las imágenes A y B como entrada, podemos observar los resultados de, en primer lugar combinar el contenido A con el estilo B y en segundo lugar combinar el contenido B con el estilo A.

Tras la aparición de las redes neuronales convolucionales entrenadas y utilizadas para la detección de patrones en imagen y clasificación de imágenes, nace la idea de la mano de Gatys et al [3], de utilizar dichas redes para intentar conseguir una transferencia de estilo entre 2 imágenes, aprovechando el filtrado en cada capa de dichas redes que a través de

distintas operaciones convolucionales lleva a cabo la detección de diferentes patrones y a distintos niveles en imágenes.

Transferir el estilo de una imagen a otra puede considerarse un problema de transferencia de textura. El trabajo tiene el objetivo de conseguir, un sistema de transferencia de estilo y textura entre imágenes pero que permita una conservación considerablemente grande de la semántica o contenido de la imagen original. El problema principal con el que nos encontramos es la gran imprecisión a la hora de seleccionar los parámetros del sistema en función de la salida que queramos obtener. La imagen de salida varía en función de los parámetros seleccionados, en la siguiente figura se muestra un ejemplo donde se puede observar la influencia de un mismo parámetro de estilo aplicada a dos imágenes de estilo diferentes sobre la misma imagen contenido:

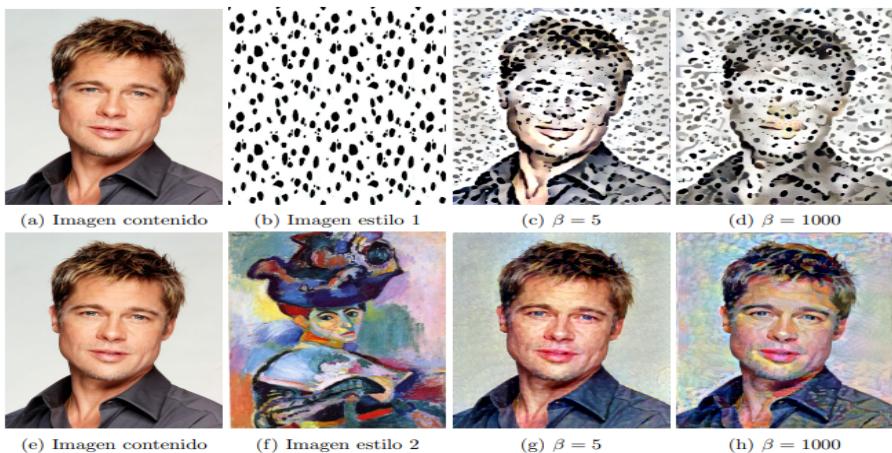


Figura 2: Podemos observar como dando diferentes pesos al parámetro  $\beta$  el resultado es notoriamente diferente.

De esta manera, nos damos cuenta de que ajustando unos u otros parámetros se consiguen realizar diferentes transferencias de estilo que proporcionan imágenes salida con mayor o menor presencia de estilo en las mismas, la cual es inversamente proporcional a la preservación de la información de contenido, esto es, cuanto mas estilo sea transferido mas difuminada quedará la información de contenido, y viceversa.

## 2. Redes Neuronales Convolucionales

### 2.1. Redes Neuronales Convencionales

Las RNAs están compuestas de una gran cantidad de procesadores conectados entre sí y actuando en paralelo. Una red neuronal se compone de un gran número de neuronas que se conectan entre sí y que están distribuidas en diferentes capas. El comportamiento de la red estará determinado por su topología, los pesos de las conexiones y la función característica de las neuronas.

Dentro de estas redes neuronales podemos diferenciar: la capa de entrada, la/las capa/s ocultas y la capa de salida. Cada una de las neuronas de la red posee como hemos mencionado anteriormente un peso, un valor numérico, con el que modifica la entrada recibida. Los nuevos valores recibidos salen de las neuronas y continúan su camino por la red. Este funcionamiento lo podemos ver de forma esquemática en la siguiente imagen:

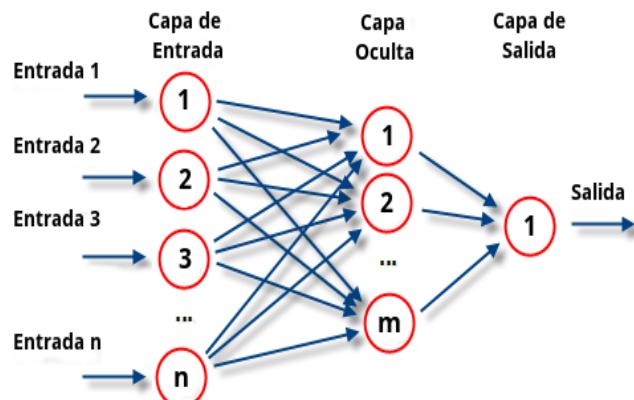


Figura 3: Red neuronal sencilla, con una única capa oculta.

Existen dos fases en toda aplicación de redes neuronales: la fase de aprendizaje y la fase de prueba:

1. **Fase de Aprendizaje o Entrenamiento:** este proceso es el que define y caracteriza una red, las redes aprenden por la actualización o cambio de sus pesos, modificando así las conexiones. Los pesos se adaptan en función de la información que se extrae de los patrones de entrenamiento.
2. **Fase de Prueba:** una vez calculados los pesos de la red, las neuronas de la última capa se comparan con la salida deseada para determinar la validez del diseño.

### 2.2. Redes Neuronales Convolucionales

Las redes neuronales convolucionales son muy similares a las redes convencionales que se han explicado anteriormente. Lo que las diferencia es que las salidas de estas son

imágenes. El objetivo de estas redes es el reconocimiento de objetos en una imagen y para ello se basan en la detección de patrones para su posterior clasificación. Para reconocer estos patrones se realizan en las capas ocultas de la red filtros de convolución, como ahora veremos.

Para realizar el estudio de las imágenes, las capas de una *RNC* organizan las neuronas en tres dimensiones: anchura, altura y profundidad. Además en estas redes no se impone la condición de que cada una de las neuronas de cada capa tenga que estar conectada con todas las neuronas de la siguiente capa, consiguiendo de esta manera una disminución considerable del número de datos a tratar.

Como sabemos, toda red neuronal se compone de una secuencia de capas y cada capa transforma las neuronas, variando sus pesos, en otras que se ajusten más al modelo. A continuación, vamos a describir las capas más importantes que forman una *RNC*:

1. **INPUT:** esta capa se conforma por la imagen de entrada y se compone de 3 dimensiones: ancho, alto y profundo (la profundidad de entrada es 3, correspondiente a los canales de color RGB).
2. **CONV:** es la capa convolucional. En esta capa se consigue llevar a cabo el proceso de detección de patrones en las imágenes. Esta capa realiza la transformación en la dimensión de la profundidad: como hemos visto anteriormente, de entrada una imagen tiene profundidad 3, si aplicamos por ejemplo 12 filtros, tras realizar una convolución, se reduce el tamaño a una única dimensión del canal RGB y posteriormente se transformará en 12 activaciones correspondiéndose cada una a los 12 filtros aplicados.
3. **POOLING:** esta capa tiene la función de realizar un cambio en las dimensiones alto y ancho. Por ejemplo, si el dato que nos llega tiene dimensiones  $[32 \times 32 \times 12]$ , la salida será  $[16 \times 16 \times 12]$ .
4. **FC:** esta capa es la encargada de computar las probabilidades de cada clase, es la última capa de la red y la única que si que está conectada a todas las neuronas de la capa anterior. El output de esta capa consiste en un vector con tantas componentes como clases hayamos determinado y dicho vector contiene los valores de probabilidad de que la imagen input que hayamos seleccionado pertenezca a una clase u otra.

Es fundamental destacar la distribución de los tipos de filtros a lo largo de la red. Estos filtros se distribuyen de forma que al principio de la red se sitúan aquellos encargados de la detección de las características más generales y de bajo nivel de la imagen. Al final de la red, se sitúan los filtros encargados de detectar las características de más alto nivel (los detalles mas pequeños y que describen, normalmente, los detalles más característicos de la imagen).

### 3. Método propuesto

Como ya se ha comentado anteriormente, la introducción de las redes neuronales convolucionales en el mundo de los gráficos computacionales ha supuesto una nueva herramienta para llevar a cabo transferencia de texturas o estilos entre imágenes. Comenzaremos introduciendo la red neuronal convolucional que se ha utilizado. Los resultados que veremos a continuación se basan en una red *VGG*, una red creada por el *Visual Geometry Group* [4] de la Universidad de Oxford. El espacio de características resultante viene dado por una versión normalizada de las 16 capas convolucionales y 5 capas de agrupación de la red *VGG*.

Para normalizar la red, se han escalado los pesos imponiendo que la activación media de cada filtro convolucional sobre las posiciones e imágenes sea igual a uno. Para la síntesis de imágenes reemplazamos la operación de agrupación máxima por la agrupación media. En las siguientes subsecciones veremos en más profundidad la representación del contenido, la representación del estilo y la transferencia de estilo.

#### 3.1. Representación de contenido

Cada capa en la red define un banco de filtros no lineal cuya complejidad va aumentando a medida que avanza la posición de dicha capa en la red. Esto quiere decir que, dada una imagen de entrada, esta se codifica en cada una de las capas por las respuestas del filtro a la imagen. Si consideramos una capa con  $N_l$  filtros distintos, la imagen tendrá  $N_l$  mapeos de características cada uno de ellos con tamaño  $M_l$  (aquí  $M_l$  denota la altura por el ancho de dicho mapeo). Por lo tanto, las respuestas de una capa " $l$ " pueden ser almacenadas en una matriz  $F^l \in R^{N_l \times M_l}$ , donde su componente  $F_{ij}^l$  se corresponde con la activación del  $i$ -ésimo filtro de la posición  $j$  en la capa  $l$ .

Para visualizar la información codificada en cada capa, sea  $\vec{p}$  la imagen original,  $P^l$  su matriz de representación,  $\vec{x}$  la imagen generada y  $F^l$  su matriz de representación característica en la capa  $l$ ; definimos la pérdida del error cuadrático entre estas como:

$$L_{\text{contenido}}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{i,j}^l - P_{i,j}^l)^2$$

La derivada de la pérdida del error cuadrático con respecto a las activaciones de la capa  $l$  es:

$$\frac{\partial L_{\text{contenido}}}{\partial F_{i,j}^l} \begin{cases} (F^l - P^l)_{i,j} & \text{si } F_{i,j}^l > 0 \\ 0 & \text{si } F_{i,j}^l < 0 \end{cases}$$

donde el gradiente respecto a la imagen  $\vec{x}$  puede ser calculado usando el algoritmo de "error-back-propagation", pudiendo cambiar esta imagen inicialmente aleatoria hasta que genere la misma respuesta en una capa de la red convolucional que la imagen original.

Cuando entrenamos las redes neuronales convolucionales en el reconocimiento de objetos, terminan con una representación de la imagen que hace que la información de los objetos sea cada vez más explícita. A lo largo del proceso, la imagen de entrada se

transforma en representaciones que son cada vez más sensibles al contenido real de la imagen, pero que se vuelven invariables a su aspecto. Así, las primeras capas capturan el contenido de alto nivel en términos de objetos y su disposición en la imagen de entrada, pero no limitan los valores exactos de los píxeles de la reconstrucción. Sin embargo, las reconstrucciones de las capas inferiores reproducen los valores exactos de los píxeles de la imagen original. Por lo tanto, nos referiremos a las respuestas de las características en las capas superiores de la red como la representación del contenido.

### 3.2. Representación de estilo

Para obtener una representación de la estilización de una imagen de entrada, utilizamos un espacio de características diseñado para capturar información de textura. Este espacio de características se puede construir sobre las respuestas del filtro en cualquier capa de la red. Consiste en las correlaciones entre las diferentes respuestas de filtro, donde la expectativa se toma sobre la extensión espacial de los mapas de características. Estas correlaciones de características están dadas por la matriz de Gram  $G^l \in \mathcal{R}^{N_l \times N_l}$  donde cada una de sus componentes  $G_{ij}^l$  es el producto interno entre los mapas de características vectoriales  $i$  y  $j$  en capa  $l$ :

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$$

Al incluir las correlaciones de características de todas las capas, obtenemos una representación de múltiples escalas de la imagen de entrada, que define su textura, pero no la disposición de objetos. Podemos visualizar la información capturada por estos espacios de características de estilo en diferentes capas de la red, construyendo una imagen que coincida la representación de estilo de una imagen de entrada dada. Para ello se utiliza el descenso de gradiente de una imagen de ruido blanco minimizando la media cuadrada distancia entre las entradas de las matrices Gram de la imagen original y las matrices Gram de la imagen a ser generada.

El procedimiento a seguir es, por tanto: sea  $\vec{a}$  la imagen original,  $A^l$  su matriz de representación,  $\vec{x}$  la imagen generada y  $G^l$  su matriz de representación característica en la capa  $l$ . La contribución de la capa  $l$  a la pérdida total es:

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$

Y la pérdida de estilo es:

$$L_{estilo} = (\vec{a}, \vec{x}, l) = \sum_{l=0}^L w_l E_l$$

donde  $w_l$  son los pesos de la contribución de cada capa a la pérdida total. La derivada de  $E_l$  con respecto a sus activaciones en la capa  $l$  es:

$$\frac{\partial L_{contenido}}{\partial F_{i,j}^l} \begin{cases} \frac{1}{N_l^2 M_l^2} ((F^l)^T (G^l - A^l))_{ij} & \text{si } F_{i,j}^l > 0 \\ 0 & \text{si } F_{i,j}^l < 0 \end{cases}$$

Donde, al igual que en la representación de contenido, el gradiente respecto a la imagen  $\vec{x}$  puede ser calculado usando el algoritmo de "error-back-propagation".

### 3.3. Transferencia de estilo

Para transferir el estilo de una fotografía  $\vec{a}$  a otra fotografía  $\vec{p}$  sintetizamos una nueva imagen que coincide con la representación de contenido de  $\vec{p}$  y la representación de estilo de  $\vec{a}$ . Para ello, minimizamos la distancia de las representaciones de características de una imagen de ruido blanco, de la representación de contenido de la fotografía en una capa y la representación de estilo definida en varias capas de la Red Neuronal Convolucional. La función de pérdida que minimizamos, teniendo en cuenta los anteriores apartados, es:

$$L_{total} = (\vec{p}, \vec{a}, \vec{x}, l) = \alpha L_{contenido}(\vec{p}, \vec{x}, l) + \beta L_{estilo}$$

donde  $\alpha$  y  $\beta$  son factores de ponderación del contenido y reconstrucción de estilo. El gradiente de  $L_{total}$  puede ser usado para realizar la optimización numérica. En este trabajo se utiliza  $L-BFGS$ , que proporciona mejores resultados para la síntesis de imágenes. Para extraer la información de la imagen en escalas comparables, se redimensiona la imagen de estilo al mismo tamaño que la imagen de contenido.

Vamos a ver un esquema del proceso:

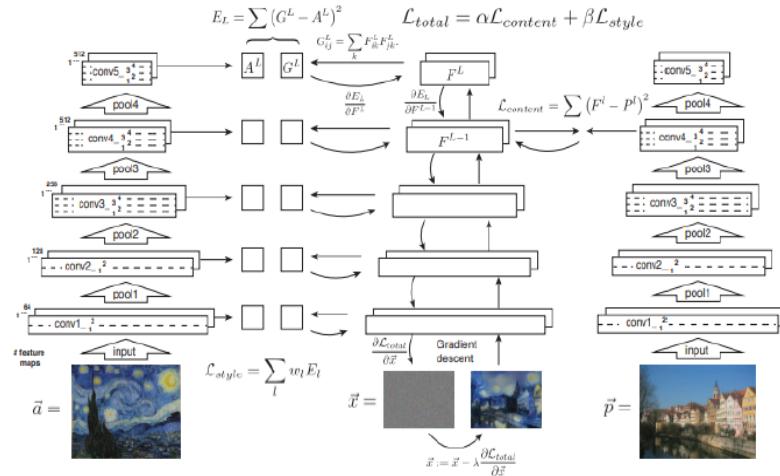


Figura 4: Algoritmo de transferencia de estilo.

Como podemos observar, en primer lugar, las características de estilo y contenido son extraídas y guardadas. La imagen del estilo  $\vec{a}$  pasa por la red y su representación de estilo  $A_l$  se calcula y guarda en cada etapa de la red (izquierda). La imagen de contenido  $\vec{p}$  pasa por la red y se calcula y guarda su representación de contenido en la cuarta etapa de la red (derecha). Creamos una imagen de ruido blanco que pasa por la red y se calculan sus características de estilo y de contenido  $G_l$  y  $P_l$ . En cada capa se calcula  $L_{estilo}$  y  $L_{contenido}$  mediante las fórmulas anteriores. La pérdida total es una combinación lineal de ambas pérdidas, que se minimiza usando descenso por el gradiente (medio). Este método se usa iterativamente para actualizar la imagen  $\vec{x}$  hasta que en ella se represente simultáneamente el estilo de  $\vec{a}$  y el contenido de  $\vec{p}$ .

Algo importante a tener en cuenta es que es una suma ponderada, es decir, se le da un peso al inicio del proceso a la imagen que aporta la información de contenido ( $\alpha$ ) y otro a la que aporta la información de estilo o textura ( $\beta$ ) y esto es fundamental a la hora de trabajar con la red, pues serán estos parámetros los que definirán la imagen output

final, de forma que si el ratio  $\alpha/\beta$  es muy elevado nuestra output será una imagen con la información de contenido relativamente bien preservada y con poca presencia del estilo transferido, por otro lado un ratio muy bajo significaría todo lo contrario y tendremos entonces una imagen output en la que la información de contenido ha quedado totalmente difuminada y hay una gran presencia del estilo transferido. En la siguiente figura se muestra un ejemplo de transferencia de estilo a una imagen para varios valores de  $\beta$ :

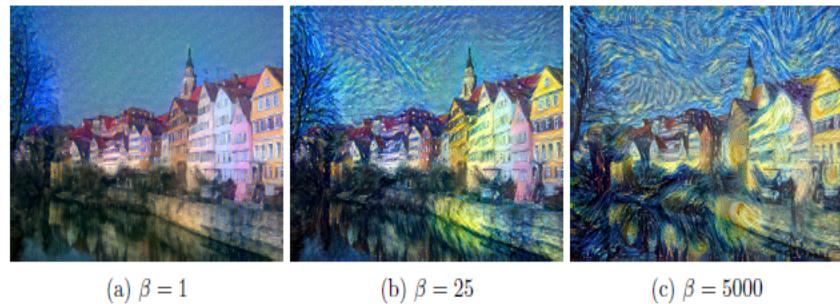


Figura 5: Ejemplos para diferentes valores del parámetro de estilo, donde se observa como para un valor bajo la transferencia de estilo es mucho inferior que para un valor alto, a su vez, para este último se produce una mayor distorsión de la información de contenido.

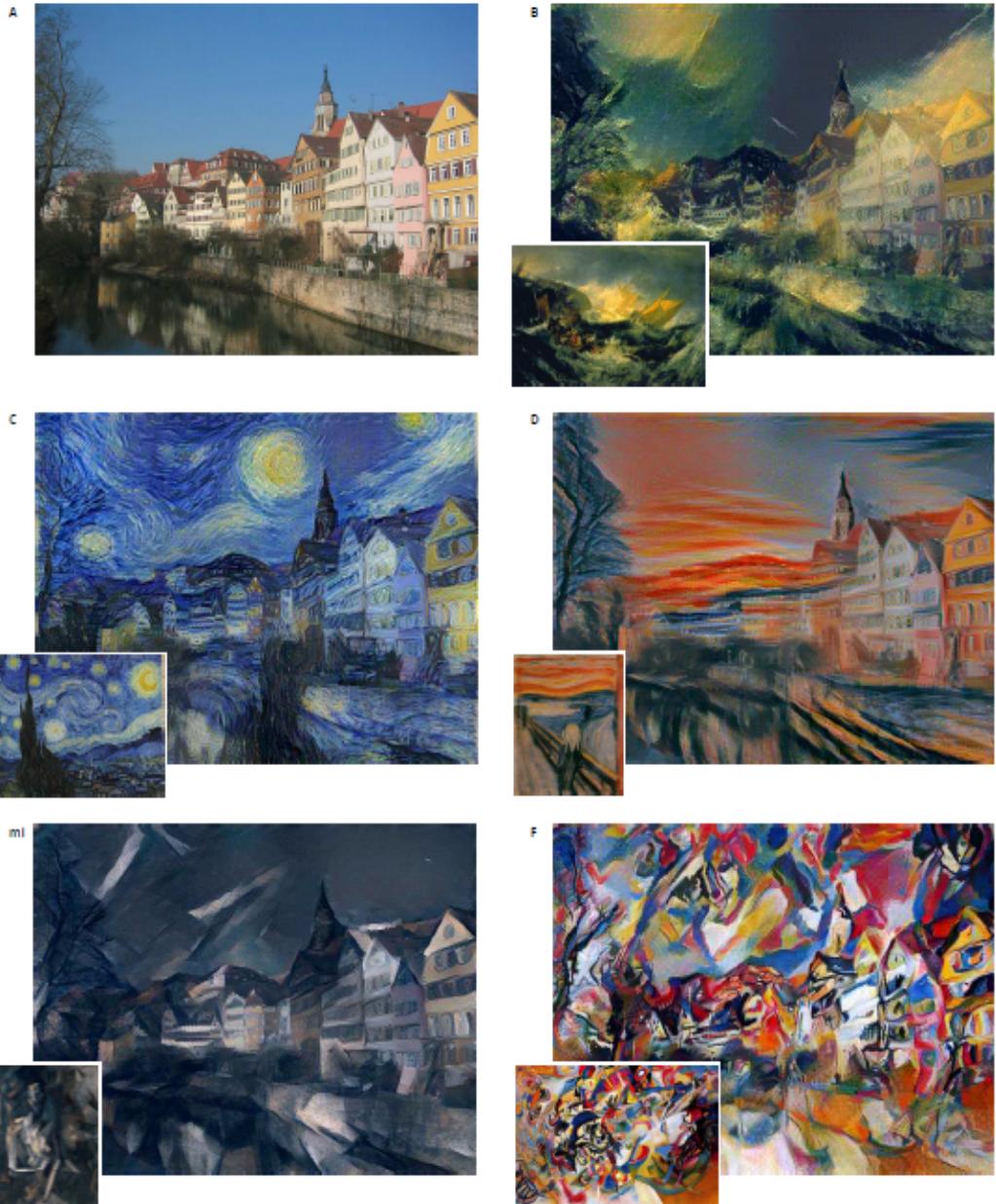
## 4. Resultados

En este artículo hemos demostrado que las representaciones de contenido y de estilo son separables para nuestra red neuronal, podemos manipular los parámetros asociados a cada una de ellas independientemente para producir nuevas imágenes. Para ver esto gráficamente se han realizado varios experimentos: se ha aplicado a la misma imagen de contenido, una fotografía de la orilla del río Neckar en Tubinga (Alemania), varias obras de arte de diferentes períodos históricos.

Las imágenes que presentamos se han creado realizando la representación de contenido en la capa '*conv4\_2*' y la representación de estilo en las capas '*conv1\_1*', '*conv2\_1*', '*conv3\_1*', '*conv4\_1*' y '*conv5\_1*' ( $w_l = 1/5$  en esas capas y  $w_l = 0$  en las demás).

En la siguiente figura podemos observar los resultados obtenidos:

- **Imagen A:** Imagen de contenido, fotografía de la orilla del río Neckar en Tubinga (Alemania). Foto: Andreas Praefcke.
- **Imagen B:** Imagen de estilo: *Naufragio del Minotauro*. Ratio  $\alpha/\beta = 1 \times 10^{-3}$ .
- **Imagen C:** Imagen de estilo: *La noche estrellada*. Ratio  $\alpha/\beta = 8 \times 10^{-4}$ .
- **Imagen D:** Imagen de estilo: *El grito*. Ratio  $\alpha/\beta = 5 \times 10^{-3}$ .
- **Imagen E:** Imagen de estilo: *Femme nue*. Ratio  $\alpha/\beta = 5 \times 10^{-4}$ . Picasso, 1910.
- **Imagen F:** Imagen de estilo: *Composición VII*. Ratio  $\alpha/\beta = 5 \times 10^{-4}$ .



Se añaden a las imágenes procedentes del proyecto, unas imágenes que hemos realizado. En ellas hemos combinado la siguiente imagen de la torre del oro de sevilla con varias obras. La imagen de contenido es la siguiente:

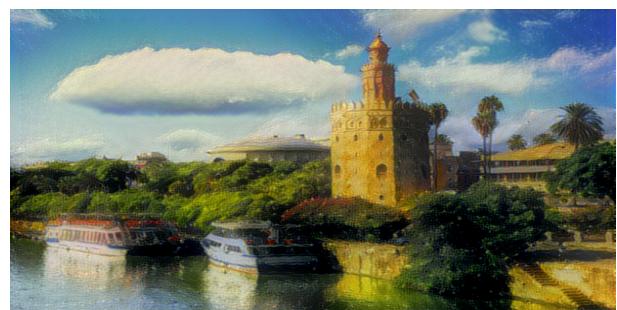


Representamos a continuación, la imagen de estilo a la izquierda, y la imagen generada a la derecha.

1. *El beso* de Klimt, Gustav Klimt



2. *Naufragio del Minotauro*, Turner



3. *La noche estrellada*, Vincent van Gogh



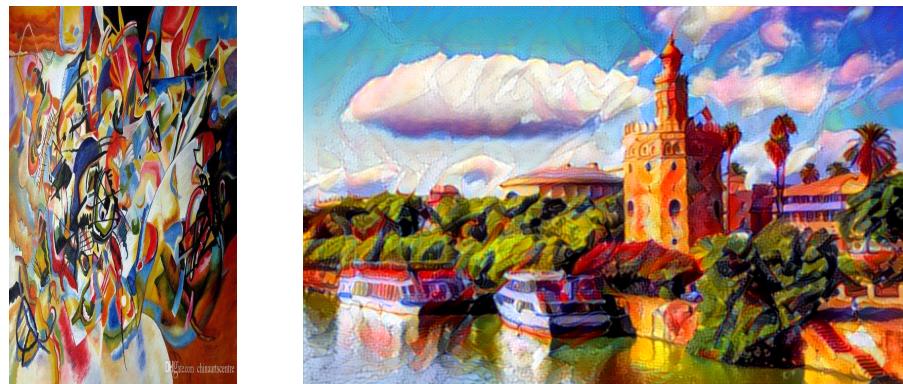
4. *El grito*, Edvard Munch



5. *Femme nue*, Pablo Picasso



## 6. Composición VII, Wassily Kandinsky



### 4.1. Variaciones de estilo y contenido

Variando los parámetros y, de esta forma el radio  $\alpha/\beta$  podemos regular el ajuste de estilo y contenido que pretendemos que tenga la imagen final. Es evidente que el estilo y el contenido, aunque hemos visto que son separables, no se pueden separar completamente. Sin embargo, por ello hemos visto que la función de pérdida que se minimiza durante la síntesis de la imagen es una combinación lineal entre las funciones de pérdida de estilo y contenido.

En la siguiente figura hemos modificado el valor de  $\alpha/\beta$  para estudiar las diferencias que supone en el resultado de la imagen de salida:



En la primera imagen (arriba a la izquierda), se da mas énfasis al estilo que al contenido y, como se puede observar, el contenido se pierde totalmente. El efecto contrario se puede observar en la última imagen (abajo a la derecha), donde el contenido se aprecia perfectamente pero no hay mucho rastro del estilo. En las imágenes intermedias, existe una combinación más equilibrada y un mejor resultado del método que pretendemos aplicar.

#### 4.2. Variacion del orden de las capas

Cuando anteriormente definimos la red, indicamos que el orden de las capas también afecta al resultado final de la imagen. El número y la posición de las capas determina la escala en la que se empareja el estilo. Entonramos por tanto que a coincidencia de las representaciones de estilo hasta las capas más altas de la red preserva las imágenes locales estructura a una escala cada vez mayor, lo que conduce a una más suave y una experiencia visual más continua.

Por lo tanto, las imágenes más atractivas se crean normalmente haciendo coincidir la representación de estilo en las altas capas de la red, que es la razón por la que para todas las imágenes mostradas aplicamos las características de estilo en las capas '*conv1\_1*', '*conv2\_1*', '*conv3\_1*', '*conv4\_1*' y '*conv5\_1*'.

Veamos un ejemplo gráfico obtenido al estilizar la misma fotografía con la misma obra pero, en la primera imagen coincidiendo en la capa '*conv2\_2*' y en la segunda, en la capa '*conv4\_2*'.



En la imagen central, podemos observar que al hacer coincidir el contenido en una capa inferior de la red, el algoritmo hace coincidir gran parte de la información detallada de los píxeles en la fotografía y la imagen generada aparece como si la textura de la obra de arte se mezclara simplemente sobre la fotografía.

En la última imagen, al hacer coincidir las características del contenido en una capa superior de la red, la información detallada de los píxeles de la fotografía no está tan fuertemente limitada y la textura de la obra de arte y el contenido de la fotografía se fusionan mucho mejor. Es decir, los bordes y el mapa de colores, cambian de tal forma que concuerda con el estilo de la obra de arte mientras sin variar el contenido de la fotografía.

#### 4.3. Inicialización del descenso de gradiente

Cuando hemos descrito nuestra red, inicializamos las imágenes mostradas hasta ahora con ruido blanco. ¿cambiaría mucho el resultado final si en vez de esto, utilizamos la imagen de estilo o la imagen de contenido? En la siguiente figura se realiza una comparativa en la que se utilizan:

- **Imagen A:** Partimos de la imagen de contenido.
- **Imagen B:** Partimos de la imagen de estilo.
- **Imagen C:** Partimos de imágenes de ruido blanco aleatorias.

Únicamente si se realiza la inicialización con ruido blanco es posible generar mas de una imagen nueva, en cambio, si la inicialización es con una imagen fija, el resultado sera único y no variará. Podemos ver la comparativa de la imagen en la siguiente página.

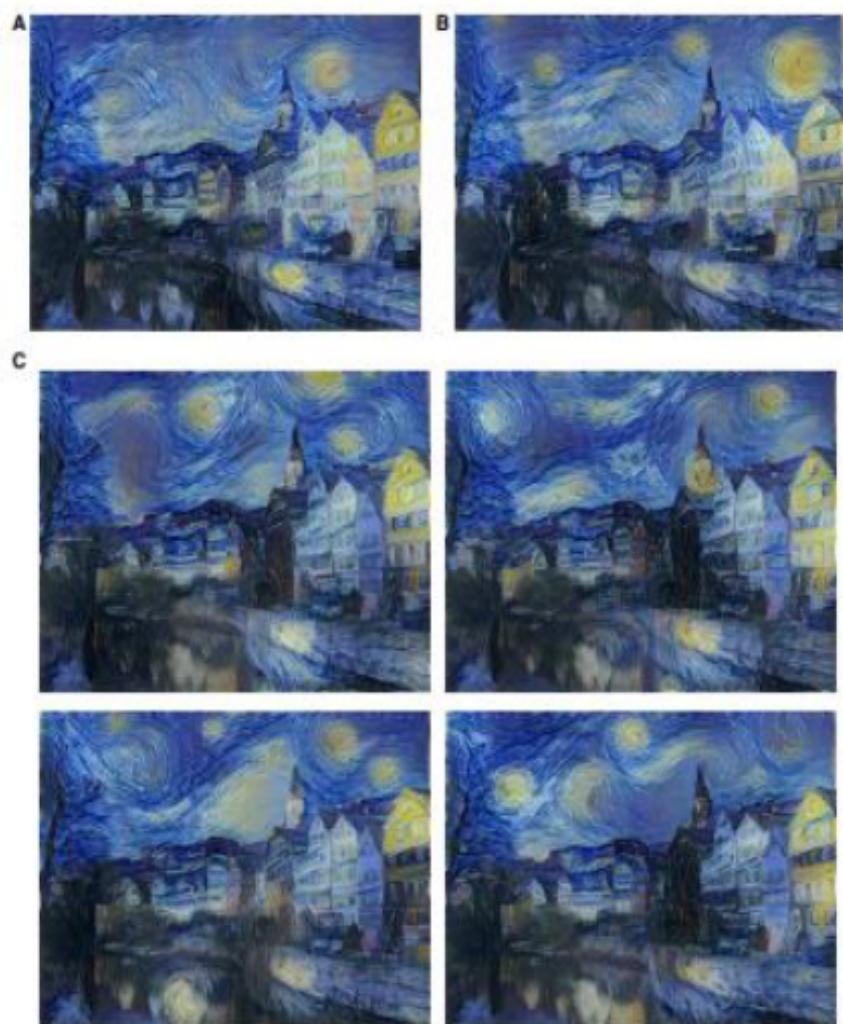
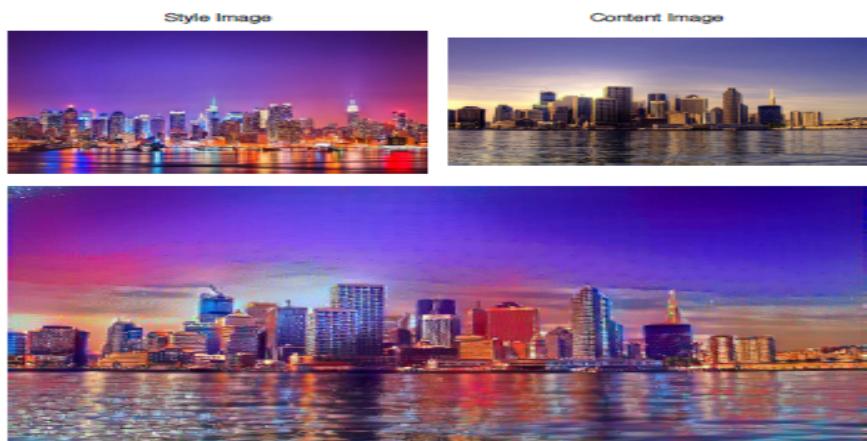


Figura 6: Inicializacion del descenso de gradiente

#### 4.4. Transferencia de estilo fotorrealista

Hasta ahora, las transferencias de estilo se han realizado con obras de arte. Este mismo algoritmo lo podemos utilizar tambien para transferir el estilo desde una imagen. En la siguiente figura se muestra una fotografía que da el estilo de Nueva York por la noche a una imagen de Londres durante el día:



En estos casos podemos observar que el fotorealismo no se conserva totalmente, pero la imagen sí se asemeja mucho a los colores de la imagen de estilo. Hemos realizado una prueba tansfiriendo el estilo de una imagen de la noche en Nueva York a la foto de Torre del Oro utilizada anteriormente, obteniendo el siguiente resultado:



Imagen de contenido



Imagen de estilo



## 5. Código

En esta última sección incluimos el código utilizado en el trabajo para realizar la transferencia de estilo en las imágenes. Además, se añaden comentarios para, de esta manera, facilitar la compresión.

Para comenzar, importamos las librerías:

```
from __future__ import print_function
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

import tensorflow as tf
tf.compat.v1.disable_eager_execution()
```

Definimos nuestros argumentos. Los parámetros opcionales son:

- *iter* : para especificar el numero de iteraciones
- *content\_weight* : el peso dado a la pérdida de contenido
- *style\_weight* : el peso dado a la pérdida de estilo
- *tv\_weight* : el peso dado a la pérdida de variación total

```
parser = argparse.ArgumentParser(description='Neural style transfer with Keras.')
parser.add_argument('base_image_path', metavar='base', type=str,
                    help='Path to the image to transform.')
parser.add_argument('style_reference_image_path', metavar='ref', type=str,
                    help='Path to the style reference image.')
parser.add_argument('result_prefix', metavar='res_prefix', type=str,
                    help='Prefix for the saved results.')
parser.add_argument('--iter', type=int, default=50, required=False,
                    help='Number of iterations to run.')
parser.add_argument('--content_weight', type=float, default=0.01, required=False,
                    help='Content weight.')
parser.add_argument('--style_weight', type=float, default=1.0, required=False,
                    help='Style weight.')
parser.add_argument('--tv_weight', type=float, default=1.0, required=False,
                    help='Total Variation weight.')

args = parser.parse_args()
base_image_path = args.base_image_path
style_reference_image_path = args.style_reference_image_path
result_prefix = args.result_prefix
iterations = args.iter
```

Definimos los pesos de los diferentes componentes de pérdida (pérdida total, pérdida de estilo y pérdida de contenido):

```
total_variation_weight = args.tv_weight
style_weight = args.style_weight
content_weight = args.content_weight
```

Dimensión de la figura generada:

```
width, height = load_img(base_image_path).size
img_nrows = 400
img_ncols = int(width * img_nrows / height)
```

Con esta función pretendemos cambiar el tamaño y formato de los tensores (pasar la imagen a matriz):

```
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
```

Con esta función pretendemos pasar los tensores a imagen (matriz a imagen):

```
def deprocess_image(x):
    if K.image_data_format() == 'channels_first':
        x = x.reshape((3, img_nrows, img_ncols))
        x = x.transpose((1, 2, 0))
    else:
        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
```

Obtenemos la representación en tensores(matriz multidimensional) de nuestra imagen para poder trabajar con ellas:

```
base_image = K.variable(preprocess_image(base_image_path))
style_reference_image = K.variable(preprocess_image(style_reference_image_path))
```

Aquí almacenaremos los datos de la imagen generada:

```
if K.image_data_format() == 'channels_first':
    combination_image = K.placeholder((1, 3, img_nrows, img_ncols))
else:
    combination_image = K.placeholder((1, img_nrows, img_ncols, 3))
```

Combinamos las tres imágenes en un solo tensor de keras:

```
input_tensor = K.concatenate([base_image,
                            style_reference_image,
                            combination_image], axis=0)
```

Construimos la red VGG19 con nuestras 3 imágenes como entrada, el modelo cargará los pesos de ImageNet, previamente entrenados:

¿Qué es ImageNet? ImageNet es un proyecto que proporciona una gran base de datos de imágenes con sus correspondientes anotaciones indicando el contenido de las imágenes. Una base de datos de este tipo es imprescindible para la investigación en visión artificial. Cabe destacar ILSVRC o ImageNet Large Scale Visual Recognition Challenge, que es una competición anual organizada por el equipo de ImageNet. En ella diferentes equipos de investigación prueban sus algoritmos de reconocimiento visual. En las competiciones se usa un subconjunto de imágenes de ImageNet que contiene 1,2 millones de imágenes de 1000 clases diferentes. El resultado de esta competición es una buena manera de saber qué redes son las mejores en clasificación de imágenes.

```

model = vgg19.VGG19(input_tensor=input_tensor,
                     weights='imagenet', include_top=False)
print('Model loaded.')

```

Obtener los nombres identificadores de cada capa

```

outputs_dict = dict([(layer.name, layer.output) for layer in model.layers])

```

Para calcular la pérdida de estilo neuronal, necesitamos definir estas cuatro funciones:

1. La matriz de Grahm para obtener las características de estilo:

```

def gram_matrix(x):
    assert K.ndim(x) == 3
    if K.image_data_format() == 'channels_first':
        features = K.batch_flatten(x)
    else:
        features = K.batch_flatten(K.permute_dimensions(x, (2, 0, 1)))
    gram = K.dot(features, K.transpose(features))
    return gram

```

2. La pérdida de estilo está diseñada para mantener el estilo de la imagen de referencia en la imagen generada. Se basa en las matrices de Grahm (que capturan el estilo) de las funciones de características de la imagen de estilo y de la imagen generada.

```

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))

```

3. La siguiente es una función de pérdida auxiliar y está diseñada para mantener el contenido de la imagen base en la imagen generada:

```

def content_loss(base, combination):
    return K.sum(K.square(combination - base))

```

4. Esta función de pérdida, pérdida de variación total, esta diseñada para mantener la imagen localmente coherente:

```

def total_variation_loss(x):
    assert K.ndim(x) == 4
    if K.image_data_format() == 'channels_first':
        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:])
    else:
        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))

```

Ahora vamos a combinar estas funciones de pérdida en un solo escalar, para ello, inicializamos el valor de pérdida en cero y este valor se irá actualizando.

```
loss = K.variable(0.0)
```

La capa de la que sacamos la pérdida de contenido:

```
layer_features = outputs_dict['block3_conv2']
base_image_features = layer_features[0, :, :, :]
combination_features = layer_features[2, :, :, :]
loss = loss + content_weight * content_loss(base_image_features,
                                             combination_features)
```

Las capas de las que sacamos la pérdida de estilo:

```
feature_layers = ['block1_conv1', 'block2_conv1',
                  'block3_conv1', 'block4_conv1',
                  'block5_conv1']
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)
```

Con la siguiente función obtenemos los gradientes de la imagen generada con respecto a la pérdida:

```
grads = K.gradients(loss, combination_image)

outputs = [loss]
if isinstance(grads, (list, tuple)):
    outputs += grads
else:
    outputs.append(grads)

f_outputs = K.function([combination_image], outputs)

def eval_loss_and_grads(x):
    if K.image_data_format() == 'channels_first':
        x = x.reshape((1, 3, img_nrows, img_ncols))
    else:
        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
```

Definimos la siguiente clase para calcular las pérdidas y los gradientes de una sola pasada, recuperándolos a través de funciones separadas.

```
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.grad_values = grad_values
        return self.loss_value

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

evaluator = Evaluator()
```

Para finalizar, necesitamos optimizar los píxeles de la imagen generada para minimizar la pérdida de estilo neuronal.

```
for i in range(iterations):
    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)
```

Esta última función es para guardar la imagen generada tras cada iteración:

```
img = deprocess_image(x.copy())
fname = result_prefix + '_at_iteration_%d.png' % i
save_img(fname, img)
end_time = time.time()
print('Image saved as', fname)
print('Iteration %d completed in %ds' % (i, end_time - start_time))
```

## 6. Conclusiones

Las conclusiones fundamentales que hemos podido sacar a lo largo de la realización del trabajo son las siguientes. En primer lugar, hemos podido comprobar lo poco predecibles que son este tipo de sistemas de transferencia de textura o estilo, es decir, no se puede predecir con total certeza que salida dará el sistema dadas dos imágenes de entrada. Esto se debe, a los

filtros de convolución de las diferentes capas de la red neuronal convolucional entrenados para la extracción y detección de patrones en imágenes, ya que estos serán más efectivos para unas imágenes que para otras.

Hemos visto que el resultado de esta transferencia depende de los parámetros  $\alpha$  (ponderación de estilo) y  $\beta$  (ponderación del contenido), si sustituimos o no el ruido blanco por otra imagen, el orden de las capas en la red y, por supuesto, de las imágenes de entrada. Es por ello que variando cualquiera de estos parámetros el resultado varía de forma notable, por lo que es difícil ajustar dichos parámetros para cada imagen.

En conclusión, la fuga de contenido y el ajuste de parámetros son los principales problemas que presenta esta red. Por una parte, si queremos ser suficientemente fieles al estilo, produciremos una fuga de contenido que hará que la imagen resultante no se parezca a la original. Por otra parte, si no "difuminamos" algo el contenido, no se verá de forma clara el estilo que queremos transmitir.

## Bibliografía

- [1] Jie An, Siyu Huang, Yibing Song, Dejing Dou, Wei Liu y Jiebo Luo. *ArtFlow: Unbiased Image Style Transfer via Reversible Neural Flows.* University of Rochester,2021.
- [2] Leon A Gatys. *Image Style Transfer Using Convolutional Neural Networks.* Centre for Integrative Neuroscience, University of Tübingen, Germany.
- [3] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. *A neural algorithm of artistic style.* arXiv preprint arXiv:1508.06576,2015.
- [4] Andrej Karpathy. *Convolutional neural networks for visual recognition.* University Lecture,2016.
- [5] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, Ming-Hsuan Yang. *Universal Style Transfer via Feature Transforms.*, 2017.
- [6] Chuan Li, Michael Wand. *Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis.* Johannes Gutenberg University,2016.
- [7] Dmitry Ulyanov, Andrea Vedaldi, Victor Lempitsky. *Instance Normalization: The Missing Ingredient for Fast Stylization.* Computer Vision Group,2017.