Visión por computador

Marcos Esteve Casademunt

Junio 2020

Contents

1 Basic implementations (1 point)

Código de la VGG: `https://colab.research.google.com/drive/1h7isvmU2S0v2d0nqYPpYXEDsKbWuYNHJ?usp=sharing`

Para el desarrollo de este ejercicio se partió del ejercicio desarrollado en la asignatura de Redes neuronales y disponible en `https://colab.research.google.com/drive/1L1klXF1RNHd7LOdb1_YdNUQaQx7digT5` y se realizó la siguiente experimentación:

- Data augmentation: Se ha realizado una aumento de los datos de entrenamiento con el fin de que la red consiga mejores generalizaciones ante datos no observados en la fase de entrenamiento. Para ello se han utilizado los siguientes parámetros:
  - desplazamiento horizontal
  - desplazamiento vertical
  - zoom de la imagen
  - rotación

  Además se ha probado a utilizar MixUp con alfa 0.2, aunque los mejores resultados no se han utilizado con este. El uso de MixUp permite mezclar imagenes de distintas clases realizando de esta forma un mejor data augmentation con el fin de obtener mejores generalizaciones al mezclar datos de distintas clases, permitiendo de esta manera obtener una red mucho más robusta. Destacar que la implementación de MixUp se ha extraido del Github `https://github.com/yu4u/mixup-generator`

- Modificación del *learning rate*: Se ha implementado un método de de *learning rate annealing* basado en *step decay* y comentado en[1]. Se ha utilizado un *learning rate* inicial de 0.2, un drop de 0.9 y el número de *epochs* necesario para realizar cada drop a 25. A continuación en la figura 1 podemos observar el comportamiento del *learning rate* con el transcurso de las *epochs*

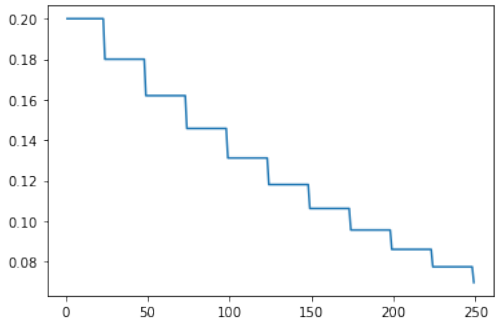


Figure 1: Step Decay

- Cada vez que se llama a la función CBGN se ha añadido una capa convolución extra y se ha modificado el ruido gausiano

[1] `http://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-`

Con todos estos experimentos el mejor resultado obtenido fue de un **0.9035**.

Tras esto se propuso construir las arquitecturas VGG, descritas en el paper `https://arxiv.org/pdf/1409.1556.pdf`. A continuación se muestran las precisiones obtenidas con los distintos modelos VGG:

Table 1: Evolución de la precisión dependiendo del modelo VGG empleado

Tipo	**A**	**B**	**C**	**D**	**E**
# Parámetros	9.648.861	9.834.141	10.424.799	15.099.303	20.364.465
Precisión	0.8891	0.9003	0.8959	0.9010	0.9020

2 Gender Recognition (1 point)

Los detalles de la implementación se pueden apreciar en: `https://colab.research.google.com/drive/1pvR36e8cxE0AHNJDfaVsDFZK8HilRKA8`

Para el desarrollo de este ejercicio se han construido dos modelos descritos en la tabla inferior

Table 2: Descripción de los modelos desarrollados

Modelo	Grande	Pequeño
# parámetros	289.058	94.370
Precisión	0.9705	0.9641

Ambos modelos planteados conseguían el objetivo de conseguir más de un 95% de accuracy utilizando un data augmentation que incluia:

- Desplazamiento vertical
- Desplazamiento horizontal
- zoom
- rotación
- flip horizontal

3 Car Model identification with bi-linear models (2 points)

Los detalles de implementación se pueden apreciar en: `https://colab.research.google.com/drive/1ZbOPSpPyXX4eNZUJlQNq7TmQHLaORPJi`

Para el desarollo de este ejercicio se ha partido de la versión 2 proporcionada en el github de la asignatura `https://github.com/RParedesPalacios/ComputerVisionLab/blob/master/src/cars2.py`. Este modelo esta pensado para cargar un modelo preentrenado y ser utilizado como mapa bilinear. Se ha escogido la VGG 16 pre-entrenada sobre el conjunto de datos ImageNet.

En primer lugar se ha realizado una experimentación congelando los pesos de la red VGG y variando el mapa convolucional que será conectado al *outer-product*. En nuestro caso el mapa que mejor ha funcionado ha sido **block4-conv3**, es decir, la convolución 3 del bloque 4.

Una vez determinado el mejor mapa convolucional se ha realizado el siguiente experimento:

- Entrenar durante 25 epochs con los pesos de la red VGG congelados (learning rate 0.1) y el optimizador Adam (el cual ha demostrado converger a soluciones con precisiones más altas en un número menor de iteraciones)
- Entrenar durante 50 epochs con los peso de la red VGG descongelados (disminuyendo el learning rate de 0.01 a 0.001) y utilizando el optimizador SGD

Con este experimento se ha logrado una precisión de **65.43%**

4 Style transfer (1 point)

Para el desarrollo de este ejercicio se ha utilizado el código proporcionado en el github de la asignatura `https://github.com/RParedesPalacios/ComputerVisionLab/blob/master/notebook/style.ipynb`. Previo a su utilización, ha sido necesario solucionar algunos problemas de incompatibilidad de versiones. Para ello, tal y como se detalla a continuación, ha sido necesario instalar versiones especificas de keras, tensorflow y scipy

```
!pip install keras==2.2.5
!pip install tensorflow==1.15.0
!pip install scipy==1.2.0
```

Una vez solucionado, se ha evaluado los resultados obtenidos por el script al variar el aporte del estilo a la imagen resultante, para ello se han utilizado las siguientes imágenes:

Figure 2: Contenido

Figure 3: Estilo

A continuación se muestran los resultados obtenidos al incrementar el aporte del estilo a la función de perdida de la imagen resultante

Figure 4: Imagen del contenido con un peso 5 en el estilo

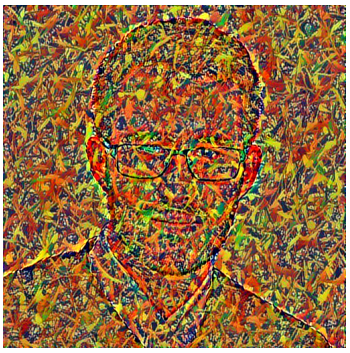


Figure 5: Imagen del contenido con un peso 50 en el estilo

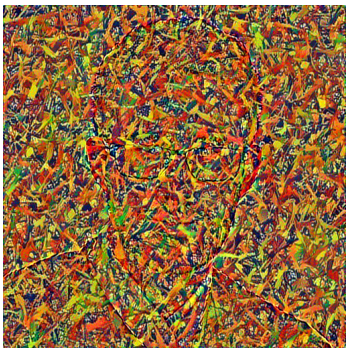


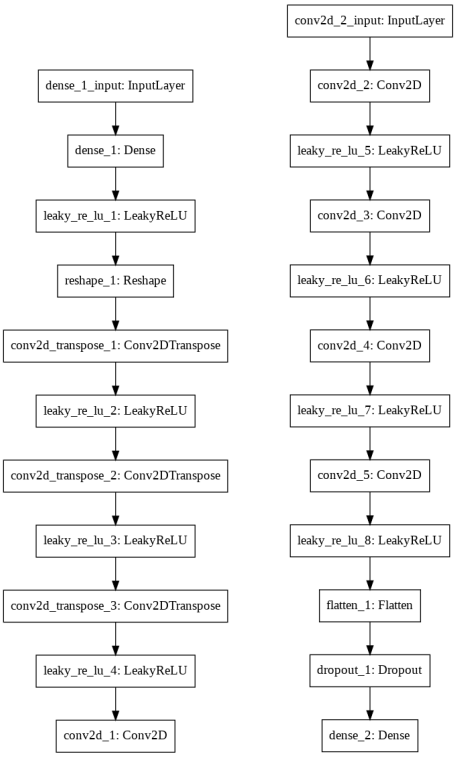
Figure 6: Imagen del contenido con un peso 100 en el estilo

5 Bonus: Implementación de una GAN para la generación de caras

Por último, a modo de ejercicio extra se ha optado por implementar una GAN para la generación de caras. La implementación está en: `https://colab.research.google.com/drive/1X2ViWKfN7cGa-20tyRiC5dyUL3DCpyFA?usp=sharing`. El código desarrollado está inspirado en el desarrollado por los autores y publicado en `https://machinelearningmastery.com/` con titulo *How to Develop a GAN to Generate CIFAR10 Small Color Photographs*

Se ha empleado como dataset de entrenamiento el conjunto de train de *The ORL Database of Faces* proporcionado en la asignatura de Biometría. El conjunto de entrenamiento consta de un total de 200 fotografías de 92x112 píxeles y 256 niveles de gris.

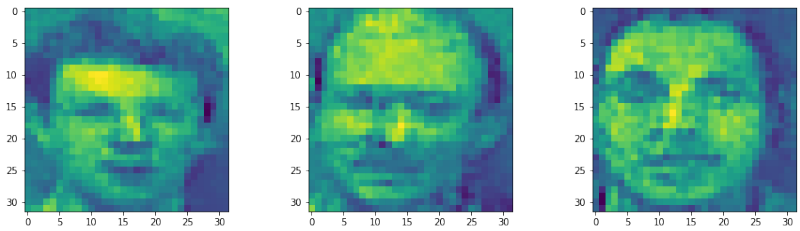
La implementación de la red consta de un generador y un discriminador enfrentados, a continuación se muestran las topologías de cada uno de los modelos:



(a) Generador (b) Discriminador

Mas concretamente, el generador recibe como entrada un vector de números aleatorios de 100 unidades y genera una cara a tamaño 32 x 32 para posteriormente pasársela al discriminador. Detalles más específicos de la red se pueden apreciar en el código adjunto.

Algunas muestras generadas por el generador se pueden apreciar en la siguiente figura:



Tal y como se puede observar los resultados, aunque se consiguen distinguir adecuadamente las zonas de la cara, no son muy precisos. Además las imágenes resultantes guardan grandes similitudes con el corpus de entrenamiento, lo que se puede deber a que solo disponemos de un total de 40 individuos distintos y por tanto puede que no exista suficiente variabilidad en las fotos.

Como trabajo futuro se podría plantear mejorar el sistema empleando un dataset con un mayor número de personas así como emplear una mayor resolución en las imágenes de entrada.