Identificación de señales de tráfico

Computer Vision

Autor: Juan Antonio López Ramírez

Introducción

En este trabajo, se va a explicar el procedimiento seguido para la correcta realización de un modelo que pueda diferenciar entre 43 señales de tráfico distintas.

Para ello, se han implementado redes convolucionales basadas en la arquitectura VGG, cuyas especificaciones se puede apreciar en la figura 1. En concreto, se ha hecho uso de los modelos VGG16 (2) y VGG19 (3).

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figura 1: Configuraciones de la red VGG.

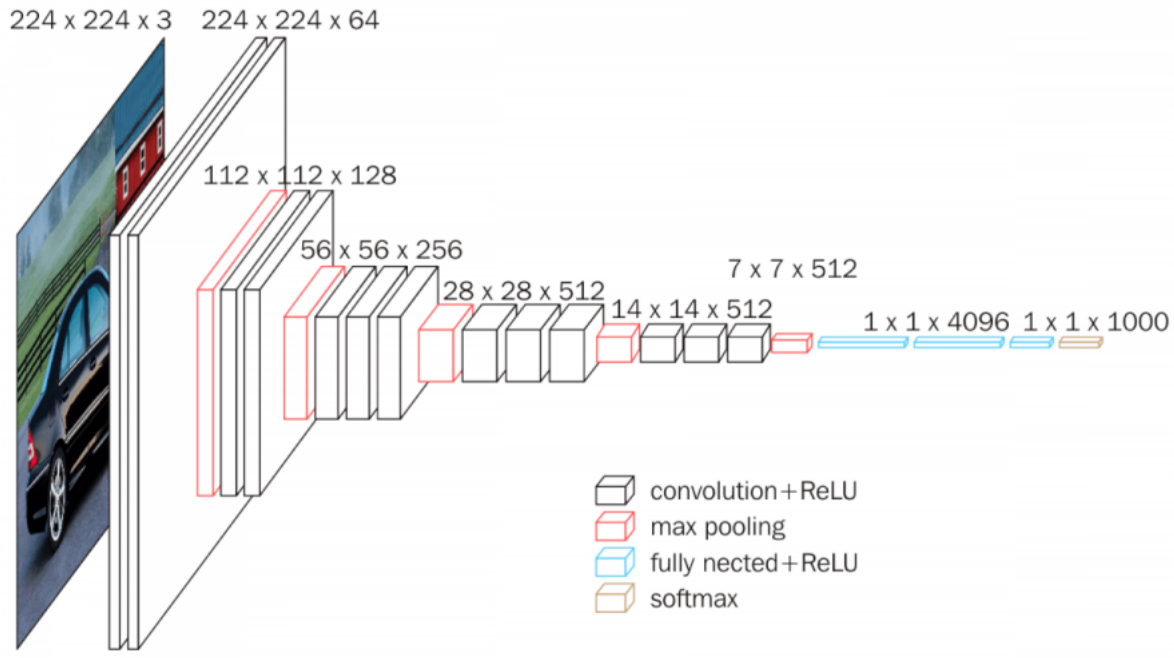

Figura 2: Arquitectura de redes convolucionales VGG16.

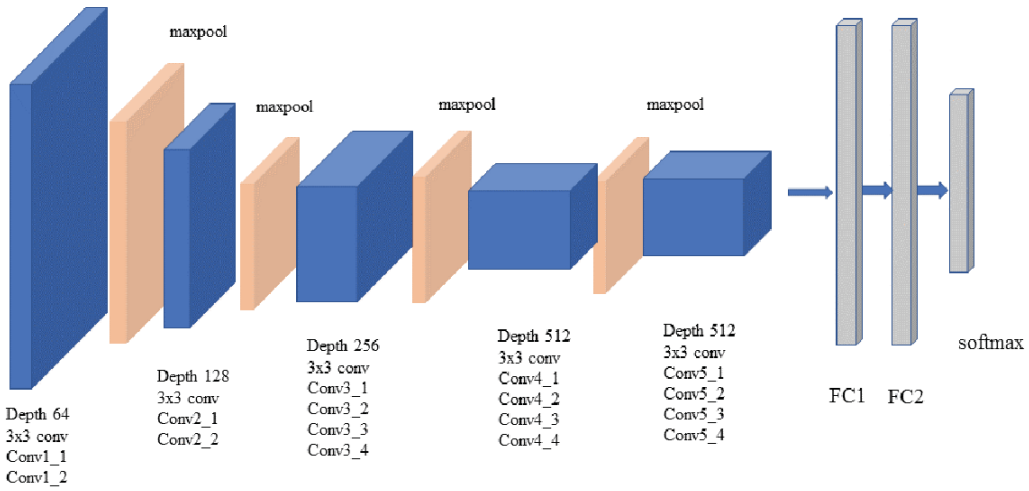

Figura 3: Arquitectura de redes convolucionales VGG19.

El dataset que se ha empleado para entrenar nuestro sistema ha sido el *German Traffic Sign Detection Benchmark*[1], que consta de 900 imágenes, 600 para entrenamiento y 300 para evaluación, donde estas contienen carreteras o paisajes viales donde pueden haber, o no, señales de tráfico.

Para el proceso de aprendizaje mediante diferenciación automática, se utilizó Keras y el entorno de trabajo de Collaboratory, proporcionado por Google, donde se nos permite ejecutar código con aceleración por GPU. La tarjeta gráfica proporcionada fue una Tesla K80, de unos 12 GB de VRAM.

Todo el código y los datos empleados para la realización de este trabajo quedarán alojados en GitHub[2].

Procesamiento de las imágenes

Para preparar el dataset para la clasificación de las distintas señales de tráfico, primeramente se han recortado de las imágenes principales aquellas regiones donde hay señales de tráfico y se han agrupado por la clase a la que pertenecen, para posteriormente quedarnos con un subset de imágenes. Concretamente, el número de imágenes a utilizar para entrenamiento es de 19604 y, para evaluación, 12630. Esto se ha conseguido realizando y ejecutando un script denominado *reducirData.py*.

Una vez que tenemos el conjunto de imágenes que se van a emplear para entrenamiento y evaluación, se procesan de manera que quedan almacenadas en matrices de NumPy (biblioteca para el lenguaje de programación Python que da soporte para crear vectores y matrices grandes multidimensionales), con las que será mas sencillo entrenar el sistema de clasificación. Para ello, se ha hecho uso de las librerías de python PIL (para procesamiento de imágenes) y SciPy (que contiene herramientas y algoritmos matemáticos). De esta forma, el resultado son 4 matrices de NumPy con extensión .npy, 2 para las imágenes de entrenamiento y evaluación, y 2 para las etiquetas de cada una de sus muestras. Los archivos guardados se han llamado *x_train.npy*, *x_test.npy*, *y_train.npy* e *y_test.npy*, donde la *x* corresponde a las muestras e *y* a las etiquetas. Esto se ha conseguido realizando y ejecutando un script denominado *procesarImgs.py*.

[1] http://benchmark.ini.rub.de/?section=gtsdb&subsection=dataset
[2] https://github.com/jualora/TrafficSigns

Experimentación y resultados

Para empezar, se ha aplicado un *Data Augmentation* de un 20 % de altura y de anchura, junto con un rango de rotación de 20 grados y un zoom también del 20 %.

Cada bloque convolucional cuenta con un *Batch Normalization*, un ruido gaussiano del 30 % y una función de activación ReLu.

El *learning rate annealing* está planificado para que el factor de aprendizaje varía en la *epoch* 75 (pasando a valer 0.01) y en la 125 (pasando a valer 0.001).

Finalmente, creamos un método al que llamamos *vgg_model()* y cuyo parámetro de entrada es el modelo de VGG que se desea utilizar. Por ejemplo, para usar el modelo de VGG16, es necesario una línea de código similar a la siguiente:

```
model = VGG_model(16)
```

De esta forma, podemos implementar y usar los distintos modelos de VGG (véase la figura 1). Debido a que el modelo C es muy similar al D, salvo que algunas convoluciones aplican un *kernel_size* de 1x1 en lugar de 3x3, se ha decidido omitir su implementación.

Para un tamaño de lote de 50 y 150 *epochs*, los resultados de precisión en evaluación que se han obtenido han sido los siguientes:

Modelo de VGG	**Resultados (%)**
16 capado	88
16	90.74
19	**94.57**

VGG16 capado hace referencia a que se recortó la red, de forma que se suprimió una de las capas densas y la otra se redujo a 512 neuronas. Esto se hizo con la intención de reducir el tiempo de entrenamiento, llegando a tardar 30 minutos y, teniendo en cuenta que con las configuraciones de VGG estándar los sistemas tardaron unos 60 minutos, la mejora en el tiempo es considerable, a pesar de perder en precisión.

Como se puede apreciar, con cualquier modelo de VGG se obtienen unos resultados competentes, siendo el mejor resultado con VGG19, que es de **94.57 %**.

El modelo se guardó en el archivo $identTrafficSigns\_batch50\_VGG19$.