PRÁCTICA 3 - VPC

Car Model identification with bi-linear models

Cristian Villarroya Sánchez

01-06-2021

Índice

1. Descripción de la tarea

Se dispone de un dataset de imágenes de coches y se ha de ser capaz de reconocer las marcas de estos (20 diferentes) utilizando una red bilineal. Además, se persigue ser capaz de nombrar e identificar las capas, construir diferentes modelos, entender el tamaño de los tensores, conectar modelos con operaciones (outerproduct), crear un generador de imágenes que devuelva una lista de tensores y crear un data Flow con diferentes entradas para el modelo.

Con todo ello, se pide construir un modelo con una red bilineal que alcance una precisión superior al 65% en el conjunto de test.

Con el fin de probar diferentes modelos pre entrenados, se han probado los modelos VGG16 y ResNet50 con los pesos de imagenet.

Las imágenes tienen un tamaño de 250x250 en RGB

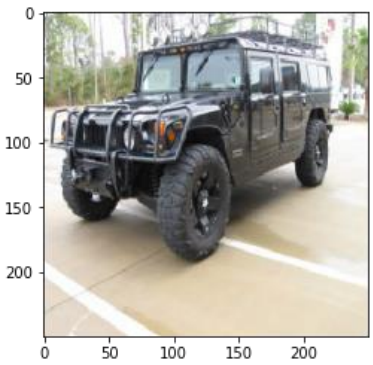

Ilustración 1. Imagen del dataset

Se han realizado dos versiones, una con un modelo bilineal partiendo de una VGG16 (cars_vgg16.ipynb) y otro partiendo de una ResNet50V2 (cars_resnet50v2.ipynb), siendo la primera la propuesta como solución a este ejercicio por superar el 65% de precisión.

2. Modelo Bilineal VGG16

Para crear este modelo, se ha seguido el código de referencia disponible en el GitHub del profesor de la asignatura [1].

Se ha diseñado una red bilineal en la que se usa el mismo modelo pre entrenado simulando ser dos modelos diferentes. Esto se consigue de la siguiente manera:

```
conv=model1.get_layer('block5_conv3')
d1=Dropout(0.5)(conv.output)    ## Why??
d2=Dropout(0.5)(conv.output)    ## Why??
```

Ilustración 2. Modelo Bilineal

Se coge la última capa convolucional del modelo pre entrenado VGG16, que tiene el nombre “block5_conv3” y a continuación, se une su salida con una capa dropout (d1) y otra capa dropout (d2), ambas con dropout de 0.5. Con esto se consigue simular una red bilineal separando en dos caminos diferentes la misma salida del modelo pre entrenado con un droput del 0.5 para que todas las neuronas tengan un 50% de posibilidad de desactivarse y así simular el efecto de separar el modelo en la mitad.

La capa “block5_conv3” tiene el siguiente tamaño:

Ilustración 3. Capa de la red VGG

Sería una convolución 512@15x15 donde la profundidad es 512 y el tamaño es 15. Este dato es importante a la hora de hacer juntar las dos redes mediante el outer product.

La operación outer product en primer lugar realiza la suma de Einstein en los elementos de las dos redes, después hace un reshape al resultado, pasando de tener la dimensión [batch_size,depth,depth] a [batch_size, depth*depth]. Como la profundidad de la capa de la VGG a la cual se ha dividido en dos para generar la bilineal es de 512, el valor de la profundidad es de 512. Después de este reshape, se divide por el tamaño del mapa de características, que es de 15*15. Finalmente, obtiene la raíz cuadrada del resultado y le aplica una normalización L2 para realizar la fusión.

Al resultado de esta fusión es al que se le aplica la capa densa con la softmax para realizar las predicciones.

Se realizaron pruebas cogiendo otra capa (la última convolución del bloque 4, block4_conv3) y también obtuvieron buenos resultados, pero finalmente con la capa block5_conv3 se obtuvo un resultado mejor.

Como siempre, uno de los aspectos más importantes es el data augmentation. Tras varias pruebas, teniendo en cuenta que valores extremos pueden resultar en errores de clasificación como puede ser una rotación extrema, la mejor configuración que se ha obtenido ha sido la siguiente:

- width_shift_range=0.22,
- height_shift_range=0.22,
- rotation_range=15,
- zoom_range=[1.0,1.2],
- horizontal_flip=True

Otro apartado es el optimizador del gradiente, se han estudiado SGD, Adam y RMSProp y, como suele ser habitual, Adam ha sido el que mejor resultados han obtenido. También se han probado diferentes learning rates, desde 0.1 hasta 0.001 y tras varias combinaciones se ha decidido utilizar el optimizador Adam con un learning rate de 0.01.

Con un learning rate de 0.01, se llegó a una precisión, antes de descongelar los pesos de la VGG16, del 56.38%, mientras que con un learning rate de 0.1, se obtuvo un valor parecido, pero algo inferior, de 53.15%, ambos experimentos con 30 épocas de entrenamiento.

```
Epoch 00025: val_accuracy improved from 0.53954 to 0.56378, saving model to drive/MyDrive/VPC/gender/weights_freeze.hdf5
Epoch 26/30
49/49 [==============================] - 14s 293ms/step - loss: 0.7090 - accuracy: 0.8746 - val_loss: 1.5511 - val_accuracy: 0.5523

Epoch 00026: val_accuracy did not improve from 0.56378
Epoch 27/30
49/49 [==============================] - 14s 291ms/step - loss: 0.6621 - accuracy: 0.8790 - val_loss: 1.5486 - val_accuracy: 0.5306

Epoch 00027: val_accuracy did not improve from 0.56378
Epoch 28/30
49/49 [==============================] - 14s 291ms/step - loss: 0.6386 - accuracy: 0.9219 - val_loss: 1.5493 - val_accuracy: 0.5395

Epoch 00028: val_accuracy did not improve from 0.56378
Epoch 29/30
49/49 [==============================] - 14s 292ms/step - loss: 0.6163 - accuracy: 0.9128 - val_loss: 1.5186 - val_accuracy: 0.5625

Epoch 00029: val_accuracy did not improve from 0.56378
Epoch 30/30
49/49 [==============================] - 14s 292ms/step - loss: 0.6153 - accuracy: 0.9147 - val_loss: 1.5268 - val_accuracy: 0.5548

Epoch 00030: val_accuracy did not improve from 0.56378
```

Ilustración 4. VGG congelada

El modelo se ha entrenado realizando un fine tunning con el modelo pre entrenado. Para ello, primero se han congelado los pesos de la VGG16 y se ha entrenado el modelo durante 30 épocas. Después, se han descongelado los pesos de la VGG16 y se ha entrenado el modelo durante otras 60 épocas. Es muy importante que, al descongelar los pesos, el learning rate sea muy pequeño, pues si se pone un valor alto, el modelo puede desaprender lo aprendido en las épocas anteriores, de hecho, se realizo una prueba con un learning rate de 0.01 tras descongelar y la precisión en la primera época bajo de un 56.37% a un 9%.

Con esto en cuenta, se ha reducido el valor del learning rate a 0.00001 y utilizado Adam como optimizador del gradiente.

Tras las 51 épocas de entrenamiento con los pesos de la VGG descongelados, se ha llegado a un 70.15% de precisión.

```
Epoch 00050: val_accuracy did not improve from 0.69898
Epoch 51/60
49/49 [==============================] - 19s 378ms/step - loss: 0.0128 - accuracy: 0.9977 - val_loss: 1.0147 - val_accuracy: 0.7015

Epoch 00051: val_accuracy improved from 0.69898 to 0.70153, saving model to drive/MyDrive/VPC/gender/weights_defrost.hdf5
Epoch 52/60
49/49 [==============================] - 19s 380ms/step - loss: 0.0200 - accuracy: 0.9947 - val_loss: 1.0411 - val_accuracy: 0.6786
```

Ilustración 5. VGG descongelada

3. Modelo Bilineal ResNet50V2

En esta ocasión, se ha querido probar el modelo ResNet50V2. La idea es la misma que en el apartado anterior, simular dos redes y juntarlas al final con un outer product, solo que ahora el modelo pre entrenado no es una VGG16 si no una ResNet50V2.

En esta ocasión, se han realizado pruebas separando desde 2 capas diferentes:

conv5_block3_3_conv (Conv2D)	(None, 8, 8, 2048)	1050624	conv5_block3_2_relu[0][0]
conv5_block3_out (Add)	(None, 8, 8, 2048)	0	conv5_block2_out[0][0] conv5_block3_3_conv[0][0]

Ilustración 6. Capas Resnet

La capa conv5_block3_3_conv que es la convolución del modelo y la conv5_block3_out, que es el resultado de añadirle a la convolución anterior el valor residual. Con ambas se han obtenido resultados parecidos, con conv5_block3_3_conv se llegó a una precisión final del 60.32 y con la conv5_block3_out se llegó al 62.88%. Finalmente, se opto por escoger la capa conv5_block3_out, que es con la que se han realizado los experimentos.

Se ha realizado un mismo proceso de fine tunning del modelo, en este caso con 60 épocas con los pesos de la ResNet congelados y 60 con los pesos de la ResNet descongelados. Tras las 60 épocas con los pesos congelados se llego a una precisión del 57.02%. El mejor resultado se obtuvo en la época 40:

```
Epoch 00039: val_accuracy did not improve from 0.56250
Epoch 40/60
49/49 [==============================] - 14s 291ms/step - loss: 1.9061 - accuracy: 0.8420 - val_loss: 8.6635 - val_accuracy: 0.5702

Epoch 00040: val_accuracy improved from 0.56250 to 0.57015, saving model to drive/MyDrive/VPC/gender/weights_freeze.hdf5
Epoch 41/60
49/49 [==============================] - 15s 295ms/step - loss: 4.4700 - accuracy: 0.8015 - val_loss: 9.7715 - val_accuracy: 0.5179

Epoch 00041: val_accuracy did not improve from 0.57015
```

Ilustración 7. ResNet congelada

Después, se descongelaron los pesos y tras 60 épocas se llego al 62.88% de precisión.

```
Epoch 00059: val_accuracy did not improve from 0.62628
Epoch 60/60
49/49 [==============================] - 18s 367ms/step - loss: 0.2152 - accuracy: 0.9368 - val_loss: 1.1889 - val_accuracy: 0.6288

Epoch 00060: val_accuracy improved from 0.62628 to 0.62883, saving model to drive/MyDrive/VPC/gender/weights_defrost_120.hdf5
```

Ilustración 8. ResNet descongelada

Para llegar a esta precisión también se han estudiado diferentes tipos de data augmentation, optimizadores del gradiente y learning rates, pero no se ha sido capaz de superar el umbral del 65% requerido. La configuración final ha sido:

- width_shift_range=0.28,
- height_shift_range=0.28,
- rotation_range=19,
- zoom_range=[1.0,1.2],
- horizontal_flip=True
- Optimizador Adam
- Learning Rate ResNet congelada 0.01
- Learning Rate ResNet congelada 0.000001

Decir que, para poder implementar la ResNet, debido a un cambio de versión de Tensorflow en Google Colab, ha sido necesario hacer unos pequeños cambios. Ahora, las capas hay que importarlas de la librería tensorflow.keras.layers, antes eran de keras.layers. Si no se hace así, se obtiene un error relacionado con transformaciones de numpy a tensores de keras. Lo mismo para el modelo, ahora es de la librería tensorflow.keras.models y hay que importar la ResNet de tf.keras.applications.

4. Conclusiones

Se han diseñado dos modelos bilineales cada uno con un modelo pre entrenado diferente.

Solamente uno de ellos, el de VGG16, ha sido capaz de superar el umbral del 65% de precisión, por lo que sería el modelo propuesto para este trabajo.

Se han realizado diferentes pruebas con el fin de mejorar la precisión, tanto un modelo como para el otro.

Las ResNet trabajan mejor con imágenes de 224x224, sin embargo, las imágenes de este dataset son de 250x250 y no han sido cambiadas de tamaño. Es posible que, si se hiciera este cambio de tamaño, la ResNet mejorase la precisión.

Además, se ha visto el como obtener las capas de un modelo pre entrenado mediante su nombre y como obtener su salida para poder usarla en un modelo bilineal. También, se ha visto la importancia del tamaño de los tensores y el como conectar modelos mediante el outer product. Finalmente, se ha realizado un fine tunning congelando y descongelando pesos del modelo pre entrenado, obteniendo así una mejora en la precisión final del modelo.

El modelo propuesto, por ser el que supera el umbral de precisión requerida, es el modelo bilineal basado en una VGG16 con una precisión final sobre el conjunto de test del 70.15%.

5. Bibliografía

[1]. R. Paredes Palacios, “Computer vision lab.” https://github.com/RParedesPalacios/ComputerVisionLab,2020.