Práctica 2: Redes neuronales convolucionales

Visión por Computador

Alejandro Palencia Blanco

28/11/2020

1. BaseNet en CIFAR100

He definido el modelo BaseNet y lo he compilado usando el optimizador SGD. Este, junto con el optimizador Adam, serán los únicos que use durante toda la práctica. Además, también usaré en todos los entrenamientos un batch size de 128 (excepto en el apartado 3, que lo bajo a 64) y un validation split de 0.1. Después de entrenar el modelo durante 30 épocas, la predicción sobre el conjunto de test me ha dado un accuracy en torno al 44-45 %.

2. Mejora del modelo BaseNet

2.1. Normalización

Primero, he intentado mejorar el modelo aplicando una normalización a los conjuntos de imágenes. Usando el optimizador SGD, he entrenado durante 30 épocas y he obtenido un accuracy en torno al 43-45 %. No parece que el modelo mejore tras haber aplicado una normalización a las imágenes.

2.2. Early Stopping

Para evitar los fenómenos de underfitting y overfitting, decido usar Early Stopping para entrenar el modelo durante un número de épocas adecuado. Lo he usado con un valor de 4 para el parámetro patience y tiene en cuenta la función de pérdida del conjunto de validación para realizar la parada. Con SGD, el modelo ha entrenado durante 16-21 épocas y he obtenido accuracy en torno al 44-46 %. Vemos que el número de épocas se reduce obteniendo un accuracy ligeramente superior, por lo que parece que ha ayudado al modelo en cierta medida.

2.3. Aumento de profundidad

En este apartado, he intentado aumentar la profundidad de muchas formas distintas. En primer lugar, probé a sustituir las convoluciones 5x5 por dos convoluciones 3x3. Además, añadí otra convolución 3x3 después del segundo MaxPooling y aumenté el número de filtros que se obtenían de cada una de ellas. Al entrenar el modelo con el optimizador Adam y un valor de 4 para patience, éste invirtió entre 11 y 14 épocas y obtuve un accuracy en torno al 46-48 % (luego mejora ligeramente).

Tal y como se aconsejaba, probé a añadir una capa de UpSampling al modelo anterior. El modelo que diseñé se podría dividir en los siguietes bloques (todas las convoluciones son 3x3 y los Pooling y UpSampling son 2x2):

1. Conv2D, ReLU, Conv2D, ReLU, MaxPooling2D
2. Conv2D, ReLU, Conv2D, ReLU, UpSampling2D
3. Conv2D, ReLU, Conv2D, ReLU, MaxPooling2D
4. Conv2D, ReLU, Conv2D, ReLU, AveragePooling2D

5. Flatten, Dense, ReLU, Dense, Softmax

La predicción que me dio este modelo al entrenarlo en las mismas condiciones que el anterior fue bastante peor: 34 % de accuracy invirtiendo 28 épocas en el entrenamiento.

Después de este, diseñé otro modelo que incorporase convoluciones 3x3 seguidas de convoluciones 1x1 que redujesen el número de filtros. EL modelo es el siguiente:

1. Conv2D(3x3), ReLU, Conv2D(1x1), ReLU
2. Conv2D(3x3), ReLU, Conv2D(1x1), ReLU, MaxPooling2D
3. Conv2D(3x3), ReLU, Conv2D(1x1), ReLU
4. Conv2D(3x3), ReLU, Conv2D(1x1), ReLU, MaxPooling2D
5. Conv2D(3x3), ReLU
6. Flatten, Dense, ReLU, Dense, Softmax

Este modelo tampoco me dio buenos resultados: 43 % de accuracy entrenando durante 20 épocas.

Finalmente, decidí quedarme con el primer modelo que cambiaba convoluciones 5x5 por dos convoluciones 3x3.

2.4. Batch Normalization

En esta mejora, añado la capa BatchNormalization2D justo después de todas las convoluciones y antes de aplicar la función ReLU. Obtuve resultados ligeramente favorables: con Adam y patience igual a 4, el modelo entrenaba entre 8 y 13 épocas y conseguía un accuracy en torno al 52-55 %.

2.5. Dropout

Ahora, añado capas Dropout que dejan inactivas de forma aleatoria un porcentaje de neuronas para cada mini-batch. Decido colocar tres capas Dropout, todas con un rate de 0.2. Dos de ellas se encuentran justo después de las capas de MaxPooling, y la otra justo antes de la primera capa Fully Connected. Esta modificación también mejoró los resultados: con el optimizador SGD y un valor de patience de 6, el modelo entrenaba durante 23-28 épocas y tenía un accuracy en torno al 60-62 %. Subí el patience de 4 a 6 porque pensé que todavía tenía un poco más de margen de mejora si aumentaban las épocas. También cambié el optimizador porque con Adam los resultados eran ligeramente peores.

2.6. Data Augmentation

Por último, decido implementar un aumento de los datos mediante distintas transformaciones como giros, traslaciones o zoom. A lo largo de este apartado siempre uso el optimizador SGD. Al principio probé con algo sencillo:

- width_shift_range=[-8,8]
- height_shift_range=[-8,8]
- zoom_range=[0.9,1.1]
- horizontal_flip=True

Esta configuración no dio buenos resultados: después de 150 épocas, obtuve un accuracy del 31 %.

A continuación, reduje los intervalos de width_shift_range y height_shift_range a [-4,4], y simplemente con este cambio obtuve un accuracy de 63 % después de 150 épocas, el cual doblaba a la anterior configuración e incluso superaba al modelo sin data augmentation en un 1 %.

Layer No.	Layer Type	Kernel size	Input | Output dimension	Input | Output channels
1	Conv2D	3	32 | 30	3 | 8
2	BatchNormalization			
3	ReLU			
4	Conv2D	3	30 | 28	8 | 16
5	BatchNormalization			
6	ReLU			
7	MaxPooling2D	2	28 | 14	
8	Dropout			
9	Conv2D	3	14 | 12	16 | 32
10	BatchNormalization			
11	ReLU			
12	Conv2D	3	12 | 10	32 | 64
13	BatchNormalization			
14	ReLU			
15	MaxPooling2D	2	10 | 5	
16	Dropout			
17	Conv2D	3	5 | 3	64 | 128
18	BatchNormalization			
19	ReLU			
20	Flatten			
21	Dropout			
22	Linear		1152 | 400	
23	ReLU			
24	Linear		400 | 25	
25	Softmax			

Cuadro 1: Modelo mejorado final

Posteriormente, añadí un rotation_range de 90 y, después de entrenar 150 épocas, el accuracy bajó a 52 %. Decidí entrenarlo durante 50 épocas más y mantuvo este porcentaje. Pensé que esto podía ser debido a que el ángulo de rotación era muy grande; entonces probé a entrenarlo con un rotation_range de 20, lo cual me dio un accuracy de 63.5 % después de 150 épocas. Decidí ejecutarlo otras 25 épocas y el accuracy subió a 64.5 %. Para comprobar que el modelo no podía mejorar más, decidí entrenarlo un poco más con un Early Stopping con patience 5, pero el accuracy se mantuvo igual.

También pensé que el modelo con convoluciones 3x3 y 1x1, que creé en el apartado de aumento de profundidad, podría mejorar con data augmentation. Antes de probarlo, le añadí capas de BatchNormalization y Dropout de forma análoga a como hice con el modelo anterior. Después de 177 épocas, logró llegar a un 54.16 % de accuracy, lo cual mejoraba su anterior marca pero no consigue superar al modelo anterior en las mismas condiciones.

2.7. Modelo mejorado final

La arquitectura del modelo resultante se muestra en la tabla 1.

3. Transferencia de modelos y ajuste fino con ResNet50 para la base de datos Caltech-UCSD

3.1. Transferencia de modelos

En este apartado, usaremos la red ResNet50 ya entrenada con la base de datos ImageNet. Si miramos las capas de la red, veremos que sus tres últimas capas son un GlobalAveragePooling, una Fully Connected y una capa de salida. Es importante que antes de entrenar, congelemos las capas correspondientes a ResNet50 para que no se reentrenen, pues en este caso vamos a usar esta red como extractor de características, luego suponemos que los pesos que tiene la red ya son adecuados para nuestro modelo.

El primer experimento consiste en eliminar la capa Fully Connected y de salida y añadir nuevas capas de la misma naturaleza que posteriormente serán entrenadas. En este modelo, he añadido tres capas Fully Connected cuyas dimensiones de salida son 1024, 512 y 200, respectivamente. Justo después de las dos primeras he añadido dos funciones ReLU, y después de la tercera una salida Softmax. Después de entrenarla con Early Stopping (patience 3) durante 13 épocas, me ha dado un 43.1 % de accuracy. Por otro lado, he diseñado un modelo en el que solo se añade y reentrena una capa Fully Connected con salida Softmax. Con el mismo Early Stopping, ha entrenado durante 17 épocas y ha obtenido un 41.3 % de accuracy. Parece que al añadir más capas Fully Connected, el modelo mejora ligeramente.

El segundo experimento es similar al primero, pero en éste también eliminamos la capa GlobalAveragePooling para luego añadir más capas. He diseñado un modelo que tiene dos convoluciones 3x3 con BatchNormalization (entre la convolución y la función ReLU) y Dropout con rate 0.2 (justo antes de la convolución). Las convoluciones devuelven 3072 y 4096 filtros, respectivamente. Después de las convoluciones, hay una capa GlobalAveragePooling seguida de un Dropout y dos capas Fully Connected cuyas dimensiones de salida son 1024 y 200, respectivamente. Entre ellas, hay una función ReLU. Por último, añado una capa de salida Softmax. El modelo se podría dividir en los siguientes bloques:

- Dropout(0.2), Conv2D(3072, 3x3), BatchNormalization, ReLU
- Dropout(0.2), Conv2D(3072, 3x3), BatchNormalization, ReLU
- GlobalAveragePooling2D, Dropout(0.2), Dense(1024), ReLU, Dense(200), Softmax

Este segundo modelo también lo he entrenado con Early Stopping durante 13 épocas llegando a tener un accuracy de 47.8 %. Vemos que experimenta una mejora del 5 % con respecto al modelo del primer experimento.

3.2. Ajuste fino

Para llevar a cabo el ajuste fino, primero he tomado el modelo ResNet50 ya entrenado con ImageNet y le he quitado las dos últimas capas (Fully Connected y salida). Luego, he añadido al final de la red una capa Fully Connected seguida de una salida de tipo Softmax. Para entrenar la red resultante, he decidido aplicar Early Stopping con un patience bajo para que entrene durante un número pequeño de épocas. Concretamente, he usado un patience igual a 2.

Con esta configuración, la red ha entrenado durante 10 épocas y me ha devuelto un accuracy de 46 %. Después, también he decidido entrenarlo solo durante 6 épocas y he obtenido el mismo accuracy. Si comparamos el ajuste fino con el segundo modelo del apartado anterior (ResNet50 como etractor de características), vemos que se obtienen valores muy similares para accuracy.

4. Bonus

He pensado que se podrían obtener mejores resultados si implementamos una arquitectura similar a alguna red neuronal convolucional que ya ha demostrado ser exitosa, la cual no tendrá por qué ser secuencial en todos sus bloques. En concreto, voy a inspirarme en la arquitectura de GoogLeNet para construir otra red mucho más reducida que utilice las mismas técnicas. GoogLeNet hace uso de un módulo llamado inception, que tiene convoluciones 1x1, 3x3 y 5x5 en paralelo. En mi versión reducida, diseñaré mi propio módulo inception pero con un menor número de convoluciones. Además, diseñaré otro módulo downsample que me permita reducir las dimensiones del tensor mediante capas en paralelo.

Antes de diseñar la arquitectura del modelo, definiré los módulos que usaré para construirlo:

- Módulo conv: devuelve el output resultante de aplicar una convolución seguida de un Batch Normalization y una activación ReLU a un input dado. Sus parámetros son:
  - x: input al que se le aplican las capas
  - filters: número de filtros de salida que devuelve la convolución
  - kernel_size: tamaño de la máscara de la convolución
  - strides: stride que se aplica en la convolución
  - padding: padding que se aplica en la convolución
- Módulo inception: devuelve el output resultante de concatenar las salidas de dos módulos de convolución, uno con máscara 1x1 y otro con máscara 3x3. Sus parámetros son:
  - x: input al que se le aplican las capas
  - filters_1x1: número de filtros de salida que devuelve el módulo de convolución 1x1
  - filters_3x3: número de filtros de salida que devuelve el módulo de convolución 3x3
- Módulo downsample: devuelve el output resultante de concatenar las salidas de un módulo de convolución y un MaxPooling, ambos con tamaño de máscara/ventana 3x3 y stride 2x2. Sus parámetros son:
  - x: input al que se le aplican las capas
  - filters: número de filtros de salida que usa el módulo de convolución

En los dos último modelos hago uso de la capa Concatenate, la cual no había usado hasta ahora. Esta capa me permite concatenar los filtros obtenidos por distintas capas en paralelo en un único tensor.

La arquitectura que he diseñado a partir de los módulos anteriores es la siguiente:

- Bloque 1:
  - conv(32 filtros, kernel 3x3)
- Bloque 2:
  - inception(32 filtros conv1x1, 32 filtros conv3x3)
  - inception(32 filtros conv1x1, 48 filtros conv3x3)
  - downsample(80 filtros conv)
- Bloque 3:
  - inception(112 filtros conv1x1, 48 filtros conv3x3)
  - inception(96 filtros conv1x1, 64 filtros conv3x3)
  - inception(80 filtros conv1x1, 80 filtros conv3x3)
  - inception(48 filtros conv1x1, 96 filtros conv3x3)
  - downsample(96 filtros conv)

- Bloque 4:
  - inception(176 filtros conv1x1, 160 filtros conv3x3)
  - inception(176 filtros conv1x1, 160 filtros conv3x3)
  - AveragePooling(kernel)
- Bloque 5:
  - Flatten
  - Dropout(rate 0.2)
  - Dense(512)
  - ReLU
  - Dense(25)
  - Softmax

He compilado el modelo con el optimizador SGD y lo he entrenado durante 100 épocas normalizando los conjuntos de imágenes y aplicando un aumento de datos con las siguientes transformaciones:

- rotation_range=20
- width_shift_range=[-4,4]
- height_shift_range=[-4,4]
- zoom_range=[0.9,1.1]
- horizontal_flip=True

La predicción sobre el conjunto de test me ha dado un 73 % de accuracy. Para asegurarme que no podía mejorar más, he reentrenado el modelo obtenido con un Early Stopping con patience 6. Después de eso, el accuracy bajó a 70 %. Por tanto, vemos que este modelo supera al modelo final del apartado 2 en casi un 10 %.

La bibliografía usada en este apartado se encuentra en el siguiente enlace.