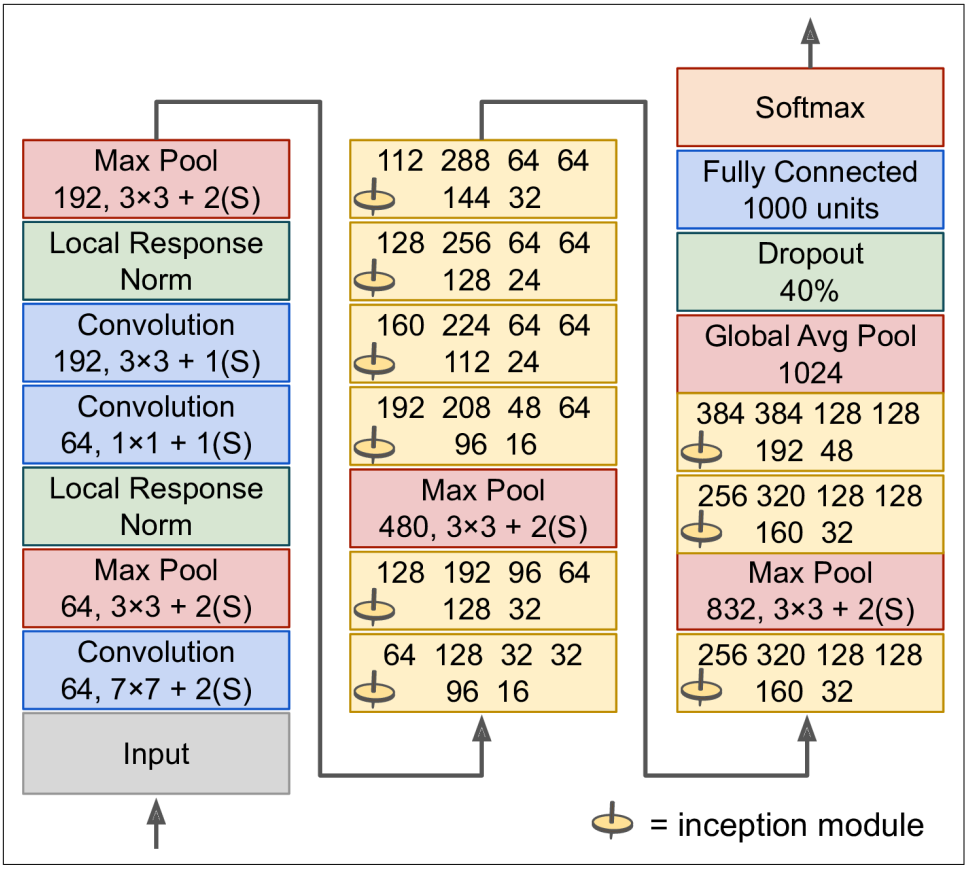

Figura 14-14. arquitectura GoogLeNet

Pasemos por esta red:

- Las dos primeras capas dividen la altura y el ancho de la imagen por 4 (por lo que su área se divide por 16), para reducir la carga computacional. La primera capa utiliza un tamaño de núcleo grande, por lo que aún se conserva gran parte de la información.
- Luego, la capa de normalización de respuesta local asegura que las capas anteriores aprendan una amplia variedad de características (como se discutió anteriormente).
- Siguen dos capas convolucionales, donde la primera actúa como un*capa de cuello de botella*. Como se explicó anteriormente, puede pensar en este par como una única capa convolucional más inteligente.
- Nuevamente, una capa de normalización de respuesta local asegura que las capas anteriores capturen una amplia variedad de patrones.

- A continuación, una capa de agrupación máxima reduce la altura y el ancho de la imagen en 2, de nuevo para acelerar los cálculos.
- Luego viene la pila alta de nueve módulos de inicio, intercalados con un par de capas de agrupación máxima para reducir la dimensionalidad y acelerar la red.
- A continuación, la capa de agrupación de promedio global simplemente genera la media de cada mapa de características: esto descarta cualquier información espacial restante, lo cual está bien ya que no quedaba mucha información espacial en ese punto. De hecho, normalmente se espera que las imágenes de entrada de GoogLeNet tengan 224 × 224 píxeles, por lo que después de un máximo de 5 capas de agrupación, cada una dividiendo la altura y el ancho por 2, los mapas de características se reducen a 7 × 7. Además, es una tarea de clasificación. , no localización, por lo que no importa dónde esté el objeto. Gracias a la reducción de dimensionalidad que aporta esta capa, no es necesario tener varias capas totalmente conectadas en la parte superior de la CNN (como en AlexNet), y esto reduce considerablemente el número de parámetros en la red y limita el riesgo de sobreajuste
- Las últimas capas se explican por sí mismas: abandono para la regularización, luego una capa completamente conectada con 1000 unidades, ya que hay 1000 clases, y una función de activación softmax para generar probabilidades de clase estimadas.

Este diagrama está ligeramente simplificado: la arquitectura original de GoogLeNet también incluía dos clasificadores auxiliares conectados en la parte superior de los módulos de inicio tercero y sexto. Ambos estaban compuestos por una capa de agrupación promedio, una capa convolucional, dos capas completamente conectadas y una capa de activación softmax. Durante el entrenamiento, su pérdida (reducida en un 70%) se agregó a la pérdida total. El objetivo era combatir el problema de los gradientes que desaparecen y regularizar la red. Sin embargo, más tarde se demostró que su efecto era relativamente menor.

Posteriormente, los investigadores de Google propusieron varias variantes de la arquitectura de GoogLeNet, incluidas Inception-v3 e Inception-v4, utilizando módulos de inicio ligeramente diferentes y alcanzando un rendimiento aún mejor.

VGGNet

El subcampeón del desafío ILSVRC 2014 fueVGGNet[14], desarrollado por K. Simon-yan y A. Zisserman. Tenía una arquitectura muy simple y clásica, con 2 o 3 capas convolucionales, una capa de agrupación, luego nuevamente 2 o 3 capas convolucionales, una capa de agrupación, y así sucesivamente (con un total de solo 16 capas convolucionales), más una red densa final con 2 capas ocultas y la capa de salida. Usó solo filtros 3 × 3, pero muchos filtros.

14 “Redes convolucionales muy profundas para el reconocimiento de imágenes a gran escala”, K. Simonyan y A. Zisserman (2015).

ResNet

El desafío ILSVRC 2015 se ganó usando un *Red Residual* (o *ResNet*), desarrollado por Kaiming He et al.,[15] que entregó una asombrosa tasa de error de los 5 primeros por debajo del 3,6%, utilizando una CNN extremadamente profunda compuesta de 152 capas. Confirmó la tendencia general: los modelos son cada vez más profundos, con cada vez menos parámetros. La clave para poder entrenar una red tan profunda es usar *omitir conexiones* (también llamado *conexiones de acceso directo*): la señal que alimenta una capa también se agrega a la salida de una capa ubicada un poco más arriba en la pila. Veamos por qué esto es útil.

Al entrenar una red neuronal, el objetivo es hacer que modele una función objetivo *h*(**X**). Si agregas la entrada **X** a la salida de la red (es decir, agrega una conexión de salto), entonces la red se verá obligada a modelar *F*(**X**) = *h*(**X**) - **X** más bien que *h*(**X**). Se llama *aprendizaje residual* (ver Figura 14-15).

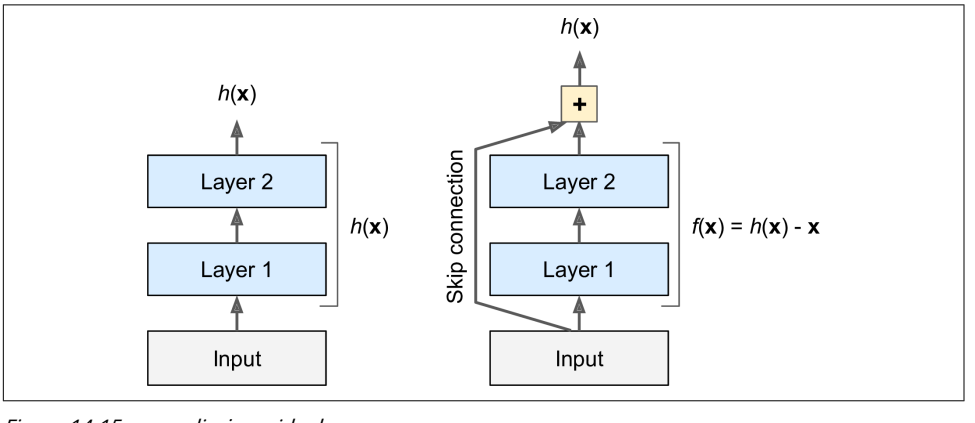

Figura 14-15. aprendizaje residual

Cuando inicializa una red neuronal regular, sus pesos son cercanos a cero, por lo que la red solo genera valores cercanos a cero. Si agrega una conexión de salto, la red resultante solo genera una copia de sus entradas; en otras palabras, inicialmente modela la función identidad. Si la función de destino está bastante cerca de la función de identidad (que suele ser el caso), esto acelerará considerablemente el entrenamiento.

Además, si agrega muchas conexiones de salto, la red puede comenzar a progresar incluso si varias capas aún no han comenzado a aprender (ver Figura 14-16). Gracias a las conexiones salteadas, la señal puede atravesar fácilmente toda la red. La red residual profunda puede verse como una pila de *unidades residuales*, donde cada unidad residual es una pequeña red neuronal con una conexión de salto.

15 "Aprendizaje residual profundo para el reconocimiento de imágenes", K. He (2015).

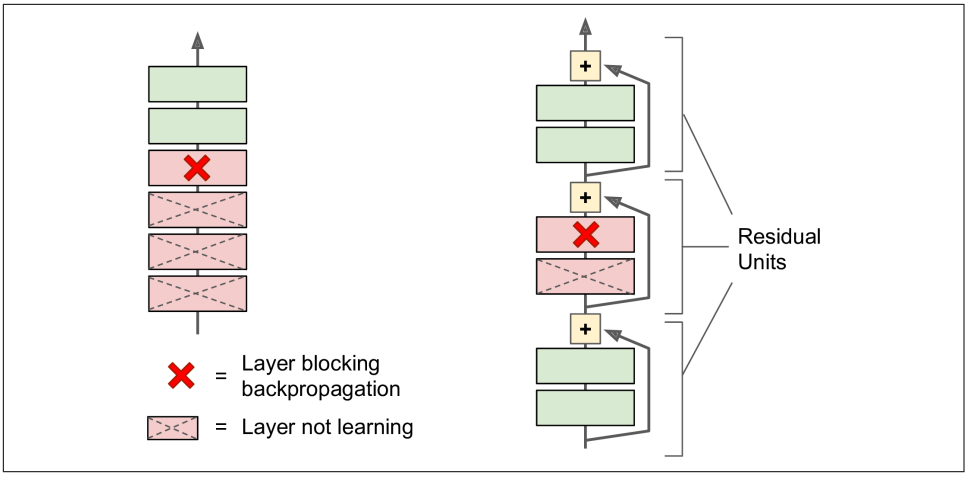

Figura 14-16. Red neuronal profunda regular (izquierda) y red residual profunda (derecha)

Ahora veamos la arquitectura de ResNet (verFigura 14-17). En realidad, es sorprendentemente simple. Comienza y termina exactamente como GoogLeNet (excepto sin una capa de abandono), y en el medio hay solo una pila muy profunda de unidades residuales simples. Cada unidad residual se compone de dos capas convolucionales (¡y ninguna capa de agrupación!), con normalización por lotes (BN) y activación de ReLU, utilizando núcleos 3 × 3 y conservando las dimensiones espaciales (paso 1, relleno MISMO).

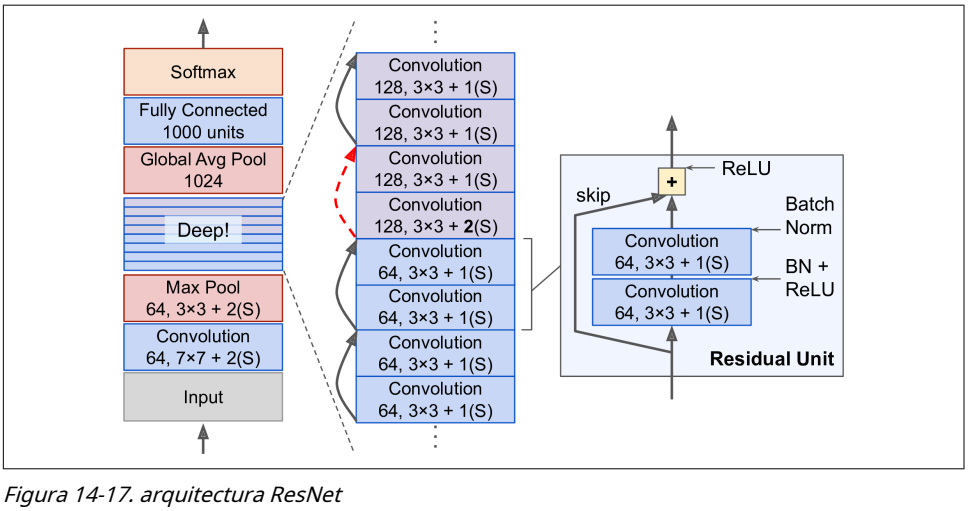

Figura 14-17. arquitectura ResNet

Tenga en cuenta que la cantidad de mapas de características se duplica cada pocas unidades residuales, al mismo tiempo que su altura y ancho se reducen a la mitad (usando una capa convolucional con paso 2). Cuando esto sucede, las entradas no se pueden sumar directamente a las salidas de la unidad residual ya que no tienen la misma forma (por ejemplo, este problema afecta el salto

conexión representada por la flecha discontinua enFigura 14-17). Para resolver este problema, las entradas se pasan a través de una capa convolucional de 1 × 1 con paso 2 y el número correcto de mapas de características de salida (verFigura 14-18).

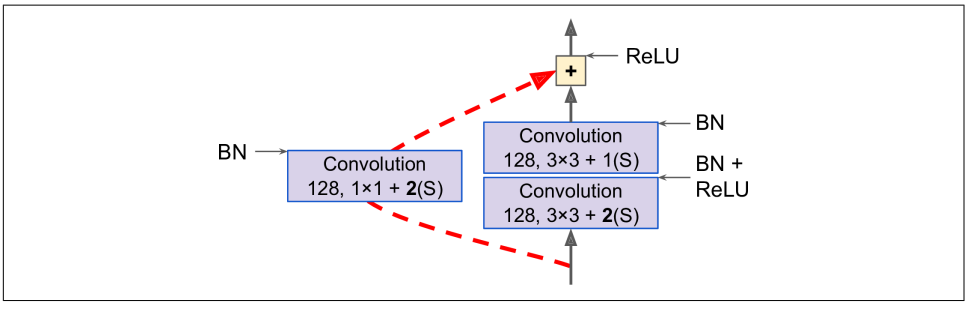

Figura 14-18. Omitir conexión al cambiar el tamaño y la profundidad del mapa de características

ResNet-34 es ResNet con 34 capas (contando solo las capas convolucionales y la capa completamente conectada) que contiene tres unidades residuales que generan 64 mapas de características, 4 RU con 128 mapas, 6 RU con 256 mapas y 3 RU con 512 mapas. Implementaremos esta arquitectura más adelante en este capítulo.

ResNets más profundas que eso, como ResNet-152, usan unidades residuales ligeramente diferentes. En lugar de dos capas convolucionales de 3 × 3 con (digamos) 256 mapas de características, utilizan tres capas convolucionales: primero, una capa convolucional de 1 × 1 con solo 64 mapas de características (4 veces menos), que actúa como una capa de cuello de botella (como ya se discutió).), luego una capa de 3 × 3 con 64 mapas de características, y finalmente otra capa convolucional de 1 × 1 con 256 mapas de características (4 veces 64) que restaura la profundidad original. ResNet-152 contiene tres RU que generan 256 mapas, luego 8 RU con 512 mapas, 36 RU con 1024 mapas y finalmente 3 RU con 2048 mapas.

de GoogleOrigen-v4dieciséisLa arquitectura fusionó las ideas de GoogLe-Net y ResNet y logró una tasa de error de cerca del 3 % entre los 5 primeros en la clasificación de ImageNet.

Xcepción

También vale la pena señalar otra variante de la arquitectura GoogLeNet:Xcepción[17] (Lo que significa*Inicio extremo*) fue propuesta en 2016 por François Chollet (el

16 "Inception-v4, Inception-ResNet y el impacto de las conexiones residuales en el aprendizaje", C. Szegedy et al. (2016).

17 "Xception: aprendizaje profundo con circunvoluciones separables en profundidad", François Chollet (2016)

autor de Keras), y superó significativamente a Inception-v3 en una gran tarea de visión (350 millones de imágenes y 17 000 clases). Al igual que Inception-v4, también fusiona las ideas de GoogLeNet y ResNet, pero reemplaza los módulos de inicio con un tipo especial de capa llamada*convolución separable en profundidad*(o*convolución separable*para abreviar[18]). Estas capas se habían utilizado antes en algunas arquitecturas de CNN, pero no eran tan centrales como en la arquitectura de Xception. Mientras que una capa convolucional regular usa filtros que intentan capturar simultáneamente patrones espaciales (p. ej., un óvalo) y patrones de canales cruzados (p. ej., boca + nariz + ojos = cara), una capa convolucional separable asume firmemente que los patrones espaciales y los canales cruzados Los patrones se pueden modelar por separado (verFigura 14-19). Por lo tanto, se compone de dos partes: la primera parte aplica un solo filtro espacial para cada mapa de características de entrada, luego la segunda parte busca exclusivamente patrones de canales cruzados; es solo una capa convolucional regular con filtros 1 × 1.

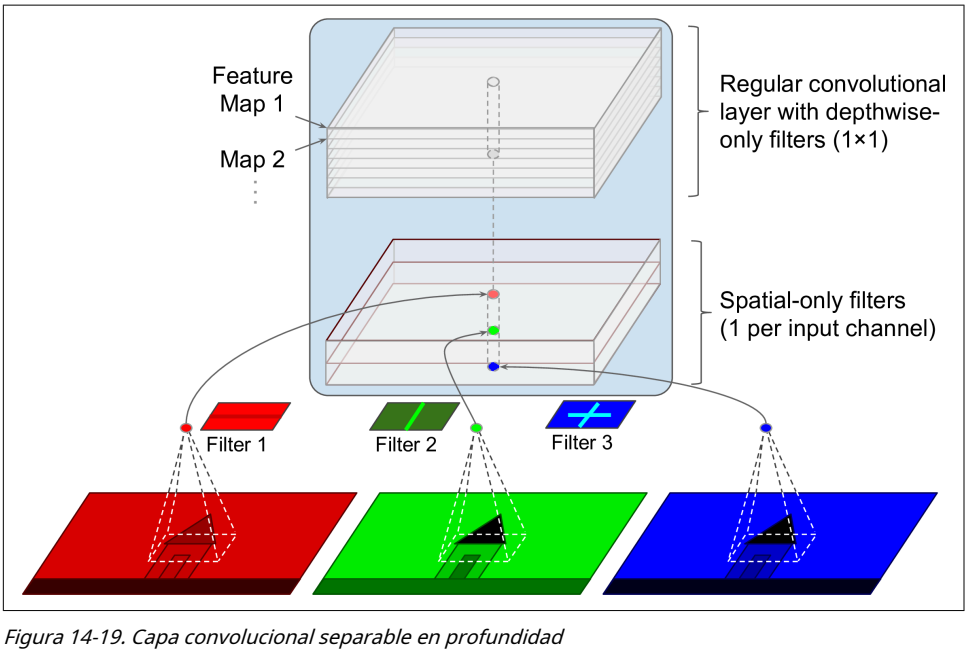

Figura 14-19. Capa convolucional separable en profundidad

Dado que las capas convolucionales separables solo tienen un filtro espacial por canal de entrada, debe evitar usarlas después de las capas que tienen muy pocos canales, como la capa de entrada (concedido, eso es lo queFigura 14-19representa, pero es solo con fines ilustrativos). Por esta razón, la arquitectura Xception comienza con 2 capas convolucionales regulares, pero luego el resto de la arquitectura usa solo convoluciones separables (34 pulgadas).

18 Este nombre a veces puede ser ambiguo, ya que las circunvoluciones separables espacialmente a menudo se denominan "circunvoluciones separables". circunvoluciones" también.

all), más algunas capas de agrupación máxima y las capas finales habituales (una capa de agrupación promedio global y una capa de salida densa).

Quizás se pregunte por qué Xception se considera una variante de GoogLeNet, ya que no contiene ningún módulo de inicio. Bueno, como comentamos anteriormente, un módulo de Inception contiene capas convolucionales con filtros 1 × 1: estos buscan exclusivamente patrones de canales cruzados. Sin embargo, las capas de convolución que se encuentran encima de ellas son capas convolucionales regulares que buscan patrones tanto espaciales como de canales cruzados. Entonces, puede pensar en un módulo Inception como un intermedio entre una capa convolucional regular (que considera patrones espaciales y patrones de canales cruzados en conjunto) y una capa convolucional separable (que los considera por separado). En la práctica, parece que las circunvoluciones separables generalmente funcionan mejor.

Las convoluciones separables utilizan menos parámetros, menos memoria y menos cálculos que las capas convolucionales normales y, en general, incluso funcionan mejor, por lo que debería considerar usarlas de forma predeterminada (excepto después de las capas con pocos canales).

El desafío ILSVRC 2016 fue ganado por el equipo CUImage de la Universidad China de Hong Kong. Usaron un conjunto de muchas técnicas diferentes, incluido un sofisticado sistema de detección de objetos llamadoGBD-Net[19], para lograr una tasa de error de los 5 primeros por debajo del 3 %. Aunque este resultado es sin duda impresionante, la complejidad de la solución contrasta con la sencillez de ResNets. Además, un año después, otra arquitectura bastante simple funcionó aún mejor, como veremos ahora.

SENet

La arquitectura ganadora del desafío ILSVRC 2017 fue laRed de compresión y excitación (SENet)[20]. Esta arquitectura amplía las arquitecturas existentes, como las redes de inicio o ResNet, y aumenta su rendimiento. ¡Esto permitió a SENet ganar la competencia con una asombrosa tasa de error de 2.25% entre los 5 primeros! Las versiones extendidas de Inception Networks y ResNet se denominan*SE-Inicio*y*SE-ResNet*respectivamente. El impulso proviene del hecho de que SENet agrega una pequeña red neuronal, llamada *Bloque SE*, a cada unidad en la arquitectura original (es decir, cada módulo de inicio o cada unidad residual), como se muestra enFigura 14-20.

19 “Elaboración de GBD-Net para la detección de objetos”, X. Zeng et al. (2016). 20 “Redes de compresión y excitación”, Jie Hu et al. (2017)

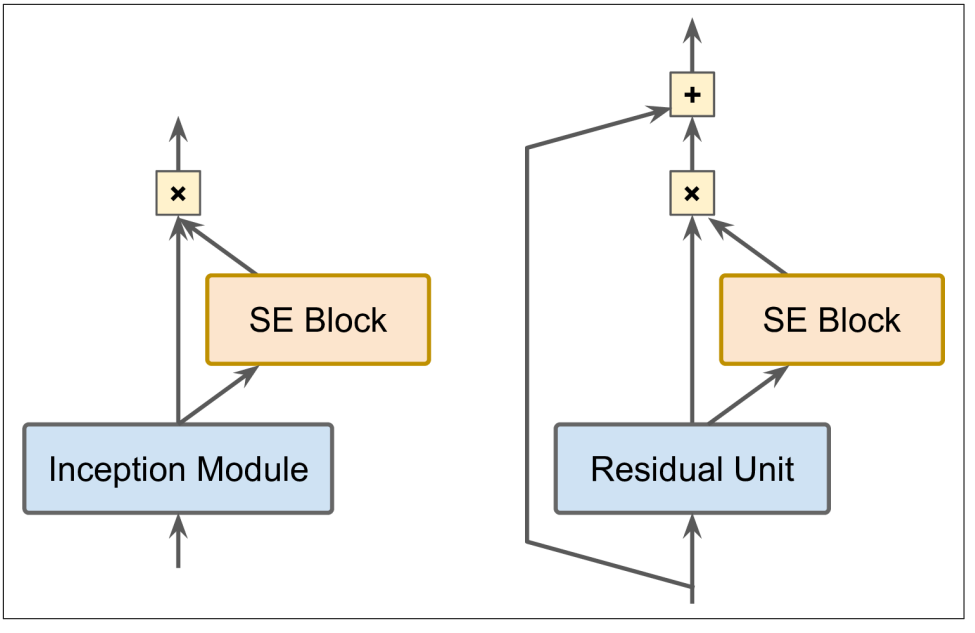

Figura 14-20. Módulo SE-Inception (izquierda) y Unidad SE-ResNet (derecha)

Un bloque SE analiza la salida de la unidad a la que está conectado, centrándose exclusivamente en la dimensión de profundidad (no busca ningún patrón espacial) y aprende qué características suelen estar más activas juntas. Luego usa esta información para recalibrar los mapas de características, como se muestra enFigura 14-21. Por ejemplo, un Bloque SE puede aprender que la boca, la nariz y los ojos generalmente aparecen juntos en las imágenes: si ve una boca y una nariz, debe esperar ver ojos también. Entonces, si un SE Block ve una fuerte activación en los mapas de funciones de la boca y la nariz, pero solo una activación leve en el mapa de funciones de los ojos, aumentará el mapa de funciones de los ojos (más precisamente, reducirá los mapas de funciones irrelevantes). Si los ojos estaban algo confundidos con otra cosa, esta recalibración del mapa de funciones ayudará a resolver la ambigüedad.

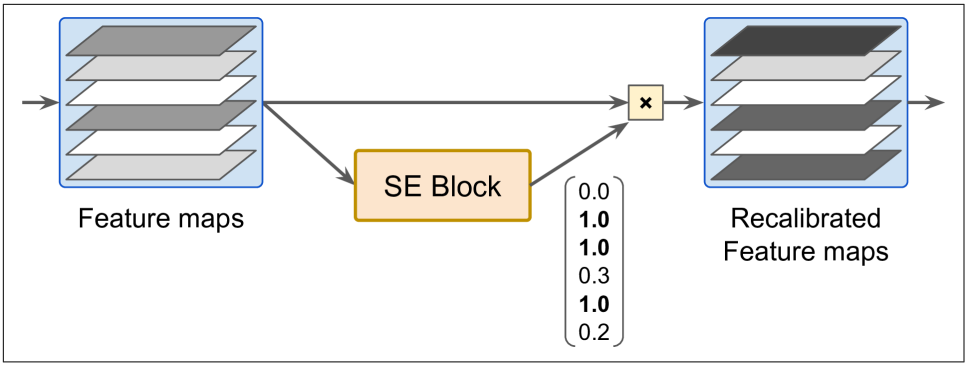

Figura 14-21. Un bloque SE realiza la recalibración del mapa de características

Un bloque SE se compone de solo 3 capas: una capa de agrupación promedio global, una capa densa oculta que usa la función de activación ReLU y una capa de salida densa que usa la función de activación sigmoide (verFigura 14-22):

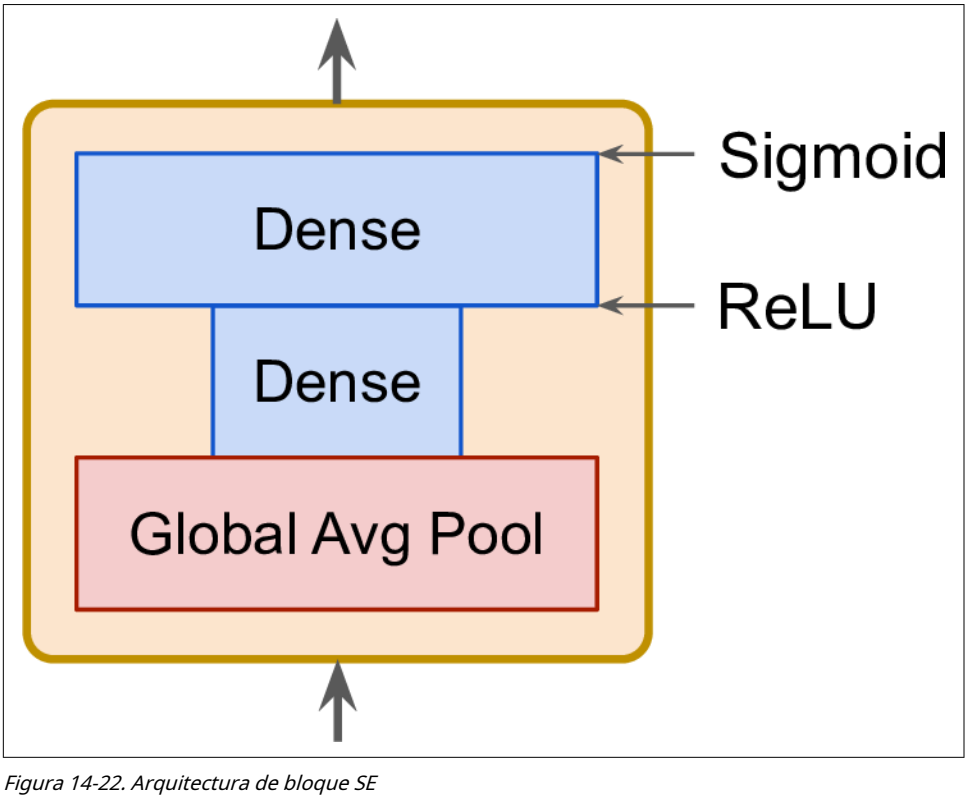

Figura 14-22. Arquitectura de bloque SE

Como antes, la capa de agrupación promedio global calcula la activación media para cada mapa de funciones: por ejemplo, si su entrada contiene 256 mapas de funciones, generará 256 números que representan el nivel general de respuesta para cada filtro. La siguiente capa es donde ocurre el "apretón": esta capa tiene mucho menos de 256 neuronas, normalmente 16 veces menos que el número de mapas de características (por ejemplo, 16 neuronas), por lo que los 256 números se comprimen en un pequeño vector (por ejemplo, 16 dimensiones). Esta es una representación vectorial de baja dimensión (es decir, una incrustación) de la distribución de respuestas de características. Este paso de cuello de botella obliga al bloque SE a aprender una representación general de las combinaciones de características (veremos este principio en acción nuevamente cuando discutamos los codificadores automáticos en???). Finalmente, la capa de salida toma la incrustación y genera un vector de recalibración que contiene un número por mapa de características (por ejemplo, 256), cada uno entre 0 y 1. Los mapas de características se multiplican luego por este vector de recalibración, por lo que las características irrelevantes (con un baja puntuación de recalibración) se reducen mientras que las características relevantes (con una puntuación de recalibración cercana a 1) se dejan solas.

Implementación de una CNN ResNet-34 usando Keras

La mayoría de las arquitecturas de CNN descritas hasta ahora son bastante sencillas de implementar (aunque, como veremos, generalmente, en su lugar, cargaría una red preentrenada). Para ilustrar el proceso, implementemos un ResNet-34 desde cero usando Keras. Primero, vamos a crear unUnidad residualcapa:

```
DefaultConv2D=parcial(queras.capas.Conv2D,kernel_size=3,zancadas=1,
                      relleno="MISMO",use_bias=Falso)

claseUnidad residual(queras.capas.Capa):
    definitivamente__en eso__(uno mismo,filtros,zancadas=1,activación="relu",**kwargs):
        súper().__en eso__(**kwargs)
        uno mismo.activación=queras.activaciones.obtener(activación) uno
        mismo.capas_principales=[
            DefaultConv2D(filtros,zancadas=zancadas),
            queras.capas.Normalización por lotes(), uno
            mismo.activación, DefaultConv2D(filtros),

            queras.capas.Normalización por lotes()]
        uno mismo.saltar_capas=[] sizancadas>1:

            uno mismo.saltar_capas=[
                DefaultConv2D(filtros,kernel_size=1,zancadas=zancadas),
                queras.capas.Normalización por lotes()]

    definitivamentellamar(uno mismo,entradas):
        Z=entradas
        porcapaenuno mismo.capas_principales:
            Z=capa(Z)
        saltar_Z=entradas
        porcapaenuno mismo.saltar_capas:
```

```
            saltar_Z=capa(saltar_Z) devolveruno
        mismo.activación(Z+saltar_Z)
```

Como puede ver, este código coincideFigura 14-18muy de cerca En el constructor, creamos todas las capas que necesitaremos: las capas principales son las del lado derecho del diagrama y las capas de salto son las del lado izquierdo (solo son necesarias si el paso es mayor que 1). Entonces en elllamar()método, simplemente hacemos que las entradas pasen por las capas principales, y las capas de salto (si las hay), luego agregamos ambas salidas y aplicamos la función de activación.

A continuación, podemos construir el ResNet-34 simplemente usando unSecuencialmodelo, ya que en realidad es solo una larga secuencia de capas (podemos tratar cada unidad residual como una sola capa ahora que tenemos laUnidad residualclase):

```
modelo=queras.modelos.Secuencial()
modelo.agregar(DefaultConv2D(64,kernel_size=7,zancadas=2,
                             entrada_forma=[224,224,3]))
modelo.agregar(queras.capas.Normalización por lotes())
modelo.agregar(queras.capas.Activación("relu"))
modelo.agregar(queras.capas.MaxPool2D(tamaño de la piscina=3,zancadas=2,relleno="MISMO"))
filtros_anteriores=64
porfiltrosen[64]*3+[128]*4+[256]*6+[512]*3:
    zancadas=1sifiltros==filtros_anterioresmás2 modelo.
    agregar(Unidad residual(filtros,zancadas=zancadas))
    filtros_anteriores=filtros
modelo.agregar(queras.capas.GlobalAvgPool2D())
modelo.agregar(queras.capas.Aplanar())
modelo.agregar(queras.capas.Denso(10,activación="softmax"))
```

La única parte un poco complicada en este código es el ciclo que agrega elUnidad residualcapas al modelo: como se explicó anteriormente, las primeras 3 RU tienen 64 filtros, luego las siguientes 4 RU tienen 128 filtros y así sucesivamente. Luego establecemos los pasos a 1 cuando el número de filtros es el mismo que en la RU anterior, o bien lo establecemos a 2. Luego agregamos elunidad residual, y finalmente actualizamosfiltros_anteriores.

¡Es bastante sorprendente que en menos de 40 líneas de código podamos construir el modelo que ganó el desafío ILSVRC 2015! Demuestra tanto la elegancia del modelo ResNet como la expresividad de la API de Keras. Implementar las otras arquitecturas de CNN no es mucho más difícil. Sin embargo, Keras viene con varias de estas arquitecturas integradas, entonces, ¿por qué no usarlas en su lugar?

Uso de modelos preentrenados de Keras

En general, no tendrá que implementar modelos estándar como GoogLeNet o ResNet manualmente, ya que las redes preentrenadas están disponibles con una sola línea de código, en elkeras.aplicacionespaquete. Por ejemplo:

```
modelo=queras.aplicaciones.resnet50.ResNet50(pesos="imagenet")
```

¡Eso es todo! Esto creará un modelo ResNet-50 y descargará pesos previamente entrenados en el conjunto de datos de ImageNet. Para usarlo, primero debe asegurarse de que las imágenes tengan el tamaño correcto. Un modelo ResNet-50 espera imágenes de 224 × 224 (otros modelos pueden esperar otros tamaños, como 299 × 299), así que usemos TensorFlowtf.imagen.redimensionar()función para cambiar el tamaño de las imágenes que cargamos anteriormente:

```
imágenes_redimensionadas=t.f..imagen.cambiar el tamaño(imágenes, [224,224])
```

lostf.imagen.redimensionar()no conservará la relación de aspecto. Si esto es un problema, puede intentar recortar las imágenes a la relación de aspecto adecuada antes de cambiar el tamaño. Ambas operaciones se pueden hacer en una tiro contf.image.crop_and_resize().

Los modelos preentrenados asumen que las imágenes son preprocesadas de una manera específica. En algunos casos, pueden esperar que las entradas se escalen de 0 a 1, o de -1 a 1, y así sucesivamente. Cada modelo proporciona unaentrada_preprocesamiento()función que puede utilizar para preprocesar sus imágenes. Estas funciones asumen que los valores de los píxeles van de 0 a 255, por lo que debemos multiplicarlos por 255 (ya que antes los escalamos al rango de 0 a 1):

```
entradas=queras.aplicaciones.resnet50.entrada_preprocesamiento(imágenes_redimensionadas*255)
```

Ahora podemos usar el modelo preentrenado para hacer predicciones:

```
Y_proba=modelo.predecir(entradas)
```

Como de costumbre, la salidaY_probaes una matriz con una fila por imagen y una columna por clase (en este caso, hay 1000 clases). Si desea mostrar las K predicciones principales, incluido el nombre de la clase y la probabilidad estimada de cada clase pronosticada, puede usar el decodificar_predicciones()función. Para cada imagen, devuelve una matriz que contiene las principales K predicciones, donde cada predicción se representa como una matriz que contiene el identificador de clase.[21], su nombre y la puntuación de confianza correspondiente:

```
arriba_K=queras.aplicaciones.resnet50.decodificar_predicciones(Y_proba,parte superior=3)
poríndice_imagenenrango(Len(imágenes)):
    impresión("Imagen #{}".formato(índice_imagen))
    poridentificador de clase,nombre,y_probaenarriba_K[índice_imagen]:
        impresión(" {} - {:12s} {:.2f}%".formato(identificador de clase,nombre,y_proba*100)) impresión
        ()
```

La salida se ve así:

```
Imagen #0
  n03877845 - palacio        42,87%
  n02825657 - bell_cote      40,57%
  n03781244 - monasterio     14,56%
```

21 En el conjunto de datos de ImageNet, cada imagen está asociada a una palabra en elConjunto de datos de WordNet: el ID de clase es solo un Identificación de WordNet.

```
Imagen #1
  n04522168 - florero          46,83%
  n07930864 - taza             7,78%
  n11939491 - margarita        4,87%
```

Las clases correctas (monasterio y margarita) aparecen en los 3 primeros resultados para ambas imágenes. Eso es bastante bueno teniendo en cuenta que el modelo tuvo que elegir entre 1000 clases.

Como puede ver, es muy fácil crear un clasificador de imágenes bastante bueno usando un modelo previamente entrenado. Otros modelos de visión están disponibles en keras.aplicaciones,incluidas varias variantes de ResNet, variantes de GoogLeNet como InceptionV3 y Xception, variantes de VGGNet, MobileNet y MobileNetV2 (modelos livianos para usar en aplicaciones móviles) y más.

Pero, ¿qué sucede si desea utilizar un clasificador de imágenes para clases de imágenes que no forman parte de ImageNet? En ese caso, aún puede beneficiarse de los modelos preentrenados para realizar transferencias de aprendizaje.

Modelos preentrenados para el aprendizaje por transferencia

Si desea crear un clasificador de imágenes, pero no tiene suficientes datos de entrenamiento, a menudo es una buena idea reutilizar las capas inferiores de un modelo previamente entrenado, como discutimos en Capítulo 11. Por ejemplo, entrenemos un modelo para clasificar imágenes de flores, reutilizando un modelo Xception previamente entrenado. Primero, carguemos el conjunto de datos usando TensorFlow Datasets (ver Capítulo 13):

```
importartensorflow_conjuntos de datoscomotfds

conjunto de datos,información=tfds.carga("tf_flores",como_supervisado=Verdadero,con_info=Verdadero)
dataset_size=información.divisiones["tren"].num_ejemplos#3670 nombres_de_clase=información.
caracteristicas["etiqueta"].nombres# ["diente de león", "margarita", ...] n_clases=información.
caracteristicas["etiqueta"].num_clases#5
```

Tenga en cuenta que puede obtener información sobre el conjunto de datos configurandowith_info=Verdadero.Aquí, obtenemos el tamaño del conjunto de datos y los nombres de las clases. Desafortunadamente, solo hay un "tren"conjunto de datos, ningún conjunto de prueba o conjunto de validación, por lo que necesitamos dividir el conjunto de entrenamiento. El proyecto TF Datasets proporciona una API para esto. Por ejemplo, tomemos el primer 10 % del conjunto de datos para pruebas, el siguiente 15 % para validación y el 75 % restante para entrenamiento:

```
prueba_dividida,división_válida,tren_dividido=tfds.Separar.TREN.subdividir([10,15,75])

equipo de prueba=tfds.carga("tf_flores",separar=prueba_dividida,como_supervisado=Verdadero)
conjunto_valido=tfds.carga("tf_flores",separar=división_válida,como_supervisado=Verdadero) juego de
trenes=tfds.carga("tf_flores",separar=tren_dividido,como_supervisado=Verdadero)
```

A continuación debemos preprocesar las imágenes. La CNN espera imágenes de 224 × 224, por lo que debemos cambiar su tamaño. También necesitamos ejecutar la imagen a través de Xceptionpreprof cess_input()función:

```
definitivamentepreprocesar(imagen,etiqueta):
    imagen_redimensionada=t.f..imagen.cambiar el tamaño(imagen, [224,224])
    imagen_final=queras.aplicaciones.xcepción.entrada_preprocesamiento(imagen_redimensionada)
    devolverimagen_final,etiqueta
```

Apliquemos esta función de preprocesamiento a los 3 conjuntos de datos, y también mezclemos y repitamos el conjunto de entrenamiento, y agreguemos procesamiento por lotes y búsqueda previa a todos los conjuntos de datos:

```
tamaño del lote=32
juego de trenes=juego de trenes.barajar(1000).repetir()
juego de trenes=juego de trenes.mapa(preprocesar).lote(tamaño del lote).captación previa(1)
conjunto_valido=conjunto_valido.mapa(preprocesar).lote(tamaño del lote).captación previa(1) equipo
de prueba=equipo de prueba.mapa(preprocesar).lote(tamaño del lote).captación previa(1)
```

Si desea realizar algún aumento de datos, simplemente puede cambiar la función de preprocesamiento para el conjunto de entrenamiento, agregando algunas transformaciones aleatorias a las imágenes de entrenamiento. Por ejemplo, usetf.imagen.random_crop()para recortar aleatoriamente las imágenes, utilice tf.imagen.random_flip_left_right()para voltear aleatoriamente las imágenes horizontalmente, y así sucesivamente (consulte el cuaderno para ver un ejemplo).

A continuación, carguemos un modelo Xception, previamente entrenado en ImageNet. Excluimos la parte superior de la red (estableciendoinclude_top=Falso):esto excluye la capa de agrupación promedio global y la capa de salida densa. Luego agregamos nuestra propia capa de agrupación promedio global, basada en la salida del modelo base, seguida de una capa de salida densa con 1 unidad por clase, usando la función de activación softmax. Por último, creamos las Keras.Modelo:

```
modelo_base=queras.aplicaciones.xcepción.Xcepción(pesos="imagenet",
                                                  include_top=Falso)
promedio=queras.capas.GlobalAveragePooling2D()()modelo_base.producción)
producción=queras.capas.Denso(n_clases,activación="softmax")(promedio) modelo=
queras.modelos.Modelo(entradas=modelo_base.aporte,salidas=producción)
```

Como se explica enCapítulo 11, suele ser una buena idea congelar los pesos de las capas preentrenadas, al menos al comienzo del entrenamiento:

```
porcapaenmodelo_base.capas:
    capa.entrenable=Falso
```

Dado que nuestro modelo utiliza las capas del modelo base directamente, en lugar de lamodelo_baseobjeto en sí mismo, ajustebase_model.trainable=Falso no tendría ningún efecto.

Finalmente, podemos compilar el modelo y comenzar a entrenar:

```
optimizador=queras.optimizadores.EUR(yo=0.2,impulso=0.9,decadencia=0.01) modelo.compilar(
pérdida="esparse_categorical_crossentropy",optimizador=optimizador,
                    métrica=["precisión"])
historia=modelo.adaptar(juego de trenes,
                                pasos_por_época=En t(0.75*dataset_size/tamaño del lote),
                                datos_de_validación=conjunto_valido,
                                pasos_de_validación=En t(0.15*dataset_size/tamaño del lote),
                                épocas=5)
```

Esto será muy lento, a menos que tenga una GPU. Si no lo hace, debe ejecutar el cuaderno de este capítulo en Colab, usando un tiempo de ejecución de GPU (¡es gratis!). Consulte las instrucciones en *https://github.com/ageron/handson-ml2*.

Después de entrenar el modelo durante algunas épocas, su precisión de validación debería alcanzar alrededor del 75-80 % y dejar de progresar mucho. Esto significa que las capas superiores ahora están bastante bien entrenadas, por lo que estamos listos para descongelar todas las capas (o puede intentar descongelar solo las superiores) y continuar con el entrenamiento (no olvide compilar el modelo cuando congele o descongele capas). Esta vez usamos una tasa de aprendizaje mucho más baja para evitar dañar los pesos preentrenados:

```
porcapaenmodelo_base.capas:
    capa.entrenable=Verdadero

optimizador=queras.optimizadores.EUR(yo=0.01,impulso=0.9,decadencia=0.001)
modelo.compilar(...) historia=modelo.adaptar(...)
```

Tomará un tiempo, pero este modelo debería alcanzar alrededor del 95 % de precisión en el equipo de prueba. ¡Con eso, puedes comenzar a entrenar increíbles clasificadores de imágenes! Pero hay más en la visión artificial que solo la clasificación. Por ejemplo, ¿qué pasa si también quieres saber *dónde* la flor esta en la foto? Veamos esto ahora.

Clasificación y Localización

La localización de un objeto en una imagen se puede expresar como una tarea de regresión, como se explica en Capítulo 10: para predecir un cuadro delimitador alrededor del objeto, un enfoque común es predecir las coordenadas horizontales y verticales del centro del objeto, así como su altura y ancho. Esto significa que tenemos 4 números para predecir. No requiere muchos cambios en el modelo, solo necesitamos agregar una segunda capa de salida densa con 4 unidades (generalmente encima de la capa de agrupación promedio global), y se puede entrenar usando la pérdida MSE:

```
modelo_base=queras.aplicaciones.xcepción.Xcepción(pesos="imagenet",
                                                  include_top=Falso)
promedio=queras.capas.GlobalAveragePooling2D()()modelo_base.producción)
clase_salida=queras.capas.Denso(n_clases,activación="softmax")(promedio)
```

```
salida_loc=queras.capas.Denso(4)(promedio)
modelo=queras.modelos.Modelo(entradas=modelo_base.aporte,
                             salidas=[clase_salida,salida_loc])
modelo.compilar(pérdida=["esparse_categorical_crossentropy","mse"],
                perdida_de_peso=[0.8,0.2],#depende de lo que mas te importe
                optimizador=optimizador,métrica=["precisión"])
```

Pero ahora tenemos un problema: el conjunto de datos de flores no tiene cuadros delimitadores alrededor de las flores. Así que tenemos que agregarlos nosotros mismos. Esta suele ser una de las partes más difíciles y costosas de un proyecto de Machine Learning: obtener las etiquetas. Es una buena idea dedicar tiempo a buscar las herramientas adecuadas. Para anotar imágenes con cuadros delimitadores, es posible que desee utilizar una herramienta de etiquetado de imágenes de código abierto como VGG Image Annotator, LabelImg, OpenLabeler o ImgLab, o quizás una herramienta comercial como LabelBox o Supervisely. También puede considerar plataformas de crowdsourcing como Amazon Mechanical Turk o CrowdFlower si tiene una gran cantidad de imágenes para anotar. Sin embargo, es bastante trabajo configurar una plataforma de crowdsourcing, preparar el formulario para enviar a los trabajadores, para supervisarlos y asegurarse de que la calidad de los cuadros delimitadores que producen sea buena, así que asegúrese de que valga la pena el esfuerzo: si solo hay unos pocos miles de imágenes para etiquetar, y no planea hacerlo con frecuencia, puede ser preferible hacerlo uno mismo. Adriana Kovashka et al. escribió una muy prácticapapel[22]sobre el crowdsourcing en Computer Vision, te recomiendo que le eches un vistazo, aunque no tengas pensado utilizar el crowdsourcing.

Así que supongamos que obtuvo los cuadros delimitadores para cada imagen en el conjunto de datos de flores (por ahora supondremos que hay un solo cuadro delimitador por imagen), luego necesita crear un conjunto de datos cuyos elementos serán lotes de imágenes preprocesadas junto con sus etiquetas de clase y sus cuadros delimitadores. Cada elemento debe ser una tupla de la forma: (imágenes, (class_labels,bounding_boxes)).Entonces estás listo para entrenar a tu ¡modelo!

Los cuadros delimitadores deben normalizarse para que las coordenadas horizontales y verticales, así como la altura y el ancho, oscilen entre 0 y 1. Además, es común predecir la raíz cuadrada de la altura y el ancho en lugar de la altura y el ancho directamente. : de esta forma, un error de 10 píxeles para un cuadro delimitador grande no se penalizará tanto como un error de 10 píxeles para un cuadro delimitador pequeño.

El MSE a menudo funciona bastante bien como una función de costo para entrenar el modelo, pero no es una gran métrica para evaluar qué tan bien el modelo puede predecir los cuadros delimitadores. La métrica más común para esto es la Intersección sobre Unión (IoU): es el área de superposición entre el cuadro delimitador predicho y el cuadro delimitador objetivo, dividido por el

22 “Crowdsourcing en Computer Vision”, A. Kovashka et al. (2016).

área de su unión (verFigura 14-23). En tf.keras, es implementado por el tf.keras.metrics.MeanIoUclase.

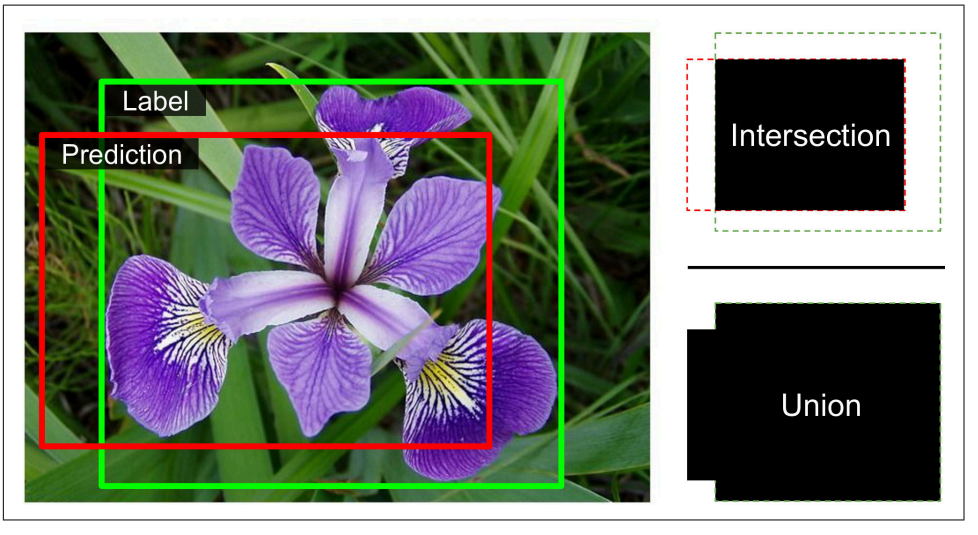

Figura 14-23. Métrica de intersección sobre unión (IoU) para cuadros delimitadores

Clasificar y localizar un único objeto está bien, pero ¿qué pasa si las imágenes contienen varios objetos (como suele ser el caso en el conjunto de datos de flores)?

Detección de objetos

La tarea de clasificar y localizar múltiples objetos en una imagen se denomina*detección de objetos*. Hasta hace unos años, un enfoque común era tomar una CNN que estaba entrenada para clasificar y ubicar un solo objeto, luego deslizarla por la imagen, como se muestra enFigura 14-24. En este ejemplo, la imagen se cortó en una cuadrícula de 6 × 8 y mostramos una CNN (el rectángulo negro grueso) deslizándose por todas las regiones de 3 × 3. Cuando la CNN estaba mirando la parte superior izquierda de la imagen, detectó parte de la rosa más a la izquierda, y luego detectó esa misma rosa nuevamente cuando se movió por primera vez un paso a la derecha. En el siguiente paso, comenzó a detectar parte de la rosa más alta, y luego la volvió a detectar una vez que se movió un paso más hacia la derecha. Luego continuaría deslizando la CNN a través de toda la imagen, observando todas las regiones de 3 × 3. Además, dado que los objetos pueden tener diferentes tamaños, también deslizaría la CNN a través de regiones de diferentes tamaños. Por ejemplo, una vez que haya terminado con las regiones de 3 × 3, es posible que también desee deslizar la CNN en todas las regiones de 4 × 4.

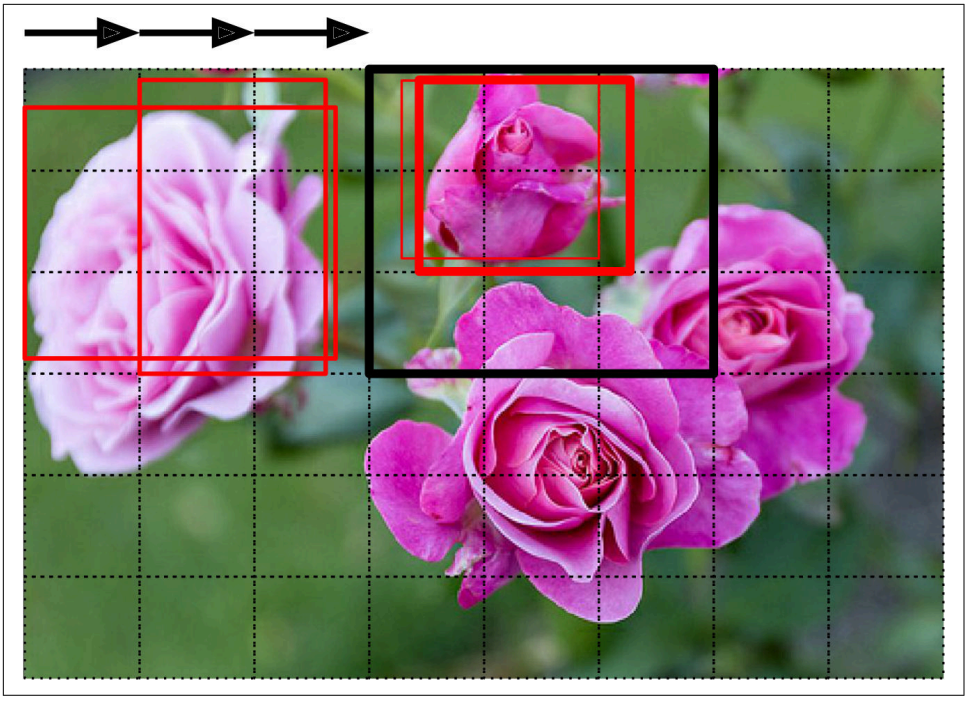

Figura 14-24. Detección de múltiples objetos deslizando una CNN a través de la imagen

Esta técnica es bastante sencilla, pero como puede ver, detectará el mismo objeto varias veces, en posiciones ligeramente diferentes. Luego será necesario un procesamiento posterior para deshacerse de todos los cuadros delimitadores innecesarios. Un enfoque común para esto se llama *supresión no máxima*:

- Primero, debe agregar un extra *objetividad* salida a su CNN, para estimar la probabilidad de que una flor esté realmente presente en la imagen (alternativamente, podría agregar una clase "sin flor", pero esto generalmente no funciona tan bien). Debe usar la función de activación sigmoide y puedes entrenarlo usando el "tropía binaria_crossen"pérdida. Luego, deshágase de todos los cuadros delimitadores para los que la puntuación de objetividad está por debajo de cierto umbral: esto eliminará todos los cuadros delimitadores que en realidad no contienen una flor.

- En segundo lugar, busque el cuadro delimitador con la puntuación de objetividad más alta y elimine todos los demás cuadros delimitadores que se superponen mucho con él (p. ej., con un IoU superior al 60%). por ejemplo, enFigura 14-24, el cuadro delimitador con la puntuación máxima de objetividad es el cuadro delimitador grueso sobre la rosa superior (la puntuación de objetividad está representada por el grosor de los cuadros delimitadores). El otro cuadro delimitador sobre esa misma rosa se superpone mucho con el cuadro delimitador máximo, por lo que nos desharemos de él.

• En tercer lugar, repita el paso dos hasta que no haya más cuadros delimitadores de los que deshacerse.

Este enfoque simple para la detección de objetos funciona bastante bien, pero requiere ejecutar la CNN muchas veces, por lo que es bastante lento. Afortunadamente, existe una manera mucho más rápida de deslizar una CNN a través de una imagen: usando un *Red completamente convolucional*.

Redes totalmente convolucionales (FCN)

La idea de FCN se introdujo por primera vez en un papel de 2015[23] de Jonathan Long et al., para la segmentación semántica (la tarea de clasificar cada píxel de una imagen según la clase del objeto al que pertenece). Señalaron que podría reemplazar las capas densas en la parte superior de una CNN por capas convolucionales. Para entender esto, veamos un ejemplo: supongamos que una capa densa con 200 neuronas se asienta sobre una capa convolucional que genera 100 mapas de características, cada uno de tamaño 7 × 7 (este es el tamaño del mapa de características, no el tamaño del kernel). Cada neurona calculará una suma ponderada de las 100 × 7 × 7 activaciones de la capa convolucional (más un término de sesgo). Ahora veamos qué sucede si reemplazamos la capa densa con una capa de convolución usando 200 filtros, cada uno de 7 × 7, y con relleno VÁLIDO. Esta capa generará 200 mapas de características, cada uno 1 × 1 (dado que el núcleo es exactamente del tamaño de los mapas de características de entrada y estamos usando relleno VÁLIDO). En otras palabras, generará 200 números, tal como lo hizo la capa densa, y si observa detenidamente los cálculos realizados por una capa convolucional, notará que estos números serán exactamente los mismos que los producidos por la capa densa. La única diferencia es que la salida de la capa densa fue un tensor de forma [tamaño de lote, 200] mientras que la capa convolucional generará un tensor de forma [tamaño de lote, 1, 1, 200].

Para convertir una capa densa en una capa convolucional, el número de filtros en la capa convolucional debe ser igual al número de unidades en la capa densa, el tamaño del filtro debe ser igual al tamaño de los mapas de características de entrada y usted debe usar relleno VÁLIDO. El paso puede establecerse en 1 o más, como veremos en breve.

¿Porque es esto importante? Bueno, mientras que una capa densa espera un tamaño de entrada específico (ya que tiene un peso por función de entrada), una capa convolucional procesará felizmente imágenes de cualquier tamaño.[24] (sin embargo, espera que sus entradas tengan un número específico de canales, ya que cada núcleo contiene un conjunto diferente de pesos para cada canal de entrada). Dado que un FCN contiene solo capas convolucionales (y capas de agrupación, que tienen la misma propiedad), se puede entrenar y ejecutar en imágenes de cualquier tamaño.

23 "Redes totalmente convolucionales para la segmentación semántica", J. Long, E. Shelhamer, T. Darrell (2015).

24 Hay una pequeña excepción: una capa convolucional que usa relleno VÁLIDO se quejará si el tamaño de entrada es más pequeño que el tamaño del núcleo.

Por ejemplo, supongamos que ya entrenamos a una CNN para la clasificación y localización de flores. Fue entrenado en imágenes de 224 × 224 y emite 10 números: las salidas 0 a 4 se envían a través de la función de activación softmax, y esto da las probabilidades de clase (una por clase); la salida 5 se envía a través de la función de activación logística, y esto da la puntuación de objetividad; las salidas 6 a 9 no utilizan ninguna función de activación y representan las coordenadas del centro del cuadro delimitador, su altura y anchura. Ahora podemos convertir sus capas densas en capas convolucionales. De hecho, ni siquiera necesitamos volver a entrenarlo, ¡solo podemos copiar los pesos de las capas densas a las capas convolucionales! Alternativamente, podríamos haber convertido la CNN en una FCN antes del entrenamiento.

Ahora suponga que la última capa convolucional antes de la capa de salida (también llamada capa de cuello de botella) genera mapas de características de 7 × 7 cuando la red se alimenta con una imagen de 224 × 224 (ver el lado izquierdo deFigura 14-25). Si alimentamos al FCN con una imagen de 448 × 448 (ver el lado derecho deFigura 14-25), la capa de cuello de botella ahora generará mapas de características de 14 × 14.[25] Dado que la capa de salida densa se reemplazó por una capa convolucional que usa 10 filtros de tamaño 7 × 7, relleno VÁLIDO y zancada 1, la salida estará compuesta por 10 mapas de características, cada uno de tamaño 8 × 8 (ya que 14 - 7 + 1 = 8). En otras palabras, el FCN procesará la imagen completa solo una vez y generará una cuadrícula de 8 × 8 donde cada celda contiene 10 números (5 probabilidades de clase, 1 puntaje de objetividad y 4 coordenadas de cuadro delimitador). Es exactamente como tomar la CNN original y deslizarla por la imagen utilizando 8 pasos por fila y 8 pasos por columna: para visualizar esto, imagine cortar la imagen original en una cuadrícula de 14 × 14 y luego deslizar una ventana de 7 × 7 a lo largo de esta cuadrícula. : habrá 8 × 8 = 64 ubicaciones posibles para la ventana, por lo tanto, 8 × 8 predicciones. Sin embargo, el enfoque FCN es*mucho*más eficiente, ya que la red solo mira la imagen una vez. En realidad,*Solo miras una vez*(YOLO) es el nombre de una arquitectura de detección de objetos muy popular.

25 Esto supone que usamos solo el MISMO relleno en la red: de hecho, el relleno VÁLIDO reduciría el tamaño de los mapas de características. Además, 448 se puede dividir claramente por 2 varias veces hasta llegar a 7, sin ningún error de redondeo. Si alguna capa usa una zancada diferente a 1 o 2, entonces puede haber algún error de redondeo, por lo que nuevamente los mapas de características pueden terminar siendo más pequeños.

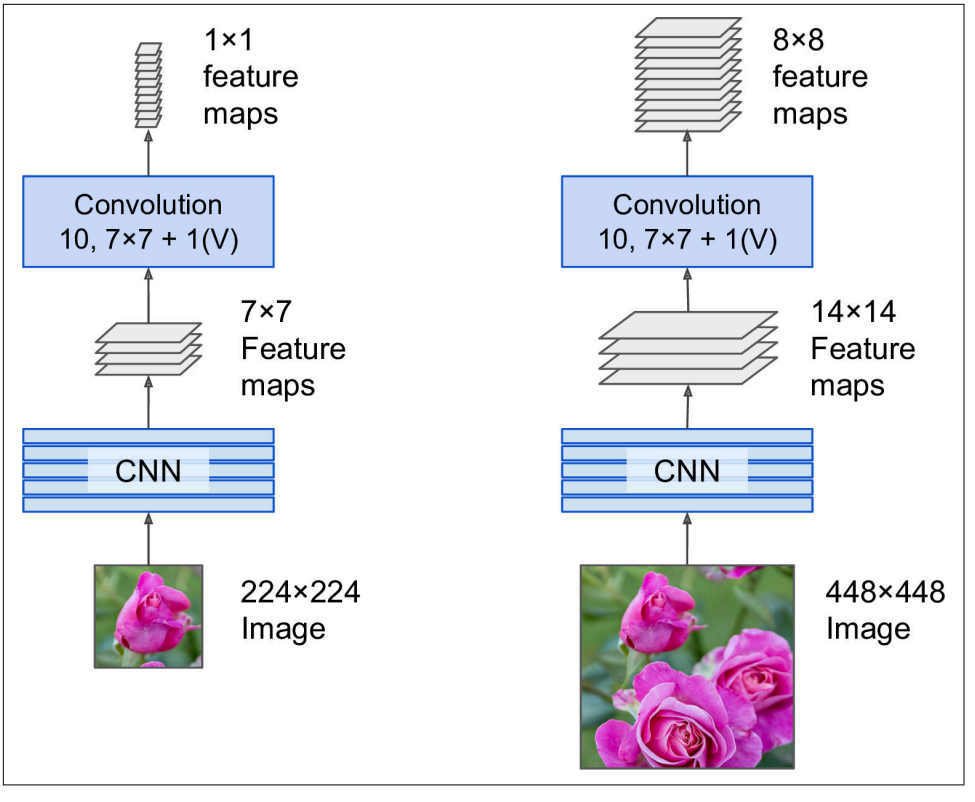

Figura 14-25. Una red totalmente convolucional que procesa una imagen pequeña (izquierda) y una grande (derecha)

Solo se mira una vez (YOLO)

YOLO es una arquitectura de detección de objetos extremadamente rápida y precisa propuesta por Joseph Redmon et al. en unpapel de 2015[26], y posteriormente mejoradoen 2016[27] (YOLOv2) yen 2018[28](YOLOv3). Es tan rápido que puede ejecutarse en tiempo real en un video (mira este bonitomanifestación).

La arquitectura de YOLOv3 es bastante similar a la que acabamos de discutir, pero con algunas diferencias importantes:

26 "Solo se mira una vez: Detección unificada de objetos en tiempo real", J. Redmon, S. Divvala, R. Girshick, A. Farhadi (2015).

27 "YOLO9000: mejor, más rápido, más fuerte", J. Redmon, A. Farhadi (2016). 28

"YOLOv3: Una mejora incremental", J. Redmon, A. Farhadi (2018).

- Primero, genera 5 cuadros delimitadores para cada celda de la cuadrícula (en lugar de solo 1), y cada cuadro delimitador viene con una puntuación de objetividad. También genera 20 probabilidades de clase por celda de cuadrícula, ya que se entrenó en el conjunto de datos PASCAL VOC, que contiene 20 clases. Eso es un total de 45 números por celda de la cuadrícula (5 * 4 coordenadas del cuadro delimitador, más 5 puntajes de objetividad, más 20 probabilidades de clase).
- En segundo lugar, en lugar de predecir las coordenadas absolutas de los centros del cuadro delimitador, YOLOv3 predice un desplazamiento relativo a las coordenadas de la celda de la cuadrícula, donde (0, 0) significa la parte superior izquierda de esa celda y (1, 1) significa abajo a la derecha. Para cada celda de la cuadrícula, YOLOv3 está capacitado para predecir solo los cuadros delimitadores cuyo centro se encuentra en esa celda (pero el cuadro delimitador en sí generalmente se extiende mucho más allá de la celda de la cuadrícula). YOLOv3 aplica la función de activación logística a las coordenadas del cuadro delimitador para garantizar que permanezcan en el rango de 0 a 1.
- En tercer lugar, antes de entrenar la red neuronal, YOLOv3 encuentra 5 dimensiones representativas del cuadro delimitador, denominadas *cajas de ancla* (o *anteriores del cuadro delimitador*): lo hace aplicando el algoritmo K-Means (ver ???) a la altura y anchura de los cuadros delimitadores del conjunto de entrenamiento. Por ejemplo, si las imágenes de entrenamiento contienen muchos peatones, es probable que una de las cajas de anclaje tenga las dimensiones de un peatón típico. Luego, cuando la red neuronal predice 5 cuadros delimitadores por celda de cuadrícula, en realidad predice cuánto cambiar la escala de cada uno de los cuadros ancla. Por ejemplo, suponga que un cuadro ancla tiene 100 píxeles de alto y 50 píxeles de ancho, y la red predice, digamos, un factor de cambio de escala vertical de 1,5 y un cambio de escala horizontal de 0,9 (para una de las celdas de la cuadrícula), esto dará como resultado un cuadro delimitador de tamaño 150 × 45 píxeles. Para ser más precisos, para cada celda de la cuadrícula y cada cuadro de anclaje, la red predice el registro de los factores de reescalado vertical y horizontal.
- Cuarto, la red se entrena usando imágenes de diferentes escalas: cada pocos lotes durante el entrenamiento, la red elige aleatoriamente una nueva dimensión de imagen (de 330 × 330 a 608 × 608 píxeles). Esto permite que la red aprenda a detectar objetos a diferentes escalas. Además, permite usar YOLOv3 a diferentes escalas: la escala más pequeña será menos precisa pero más rápida que la escala más grande, por lo que puede elegir la compensación adecuada para su caso de uso.

Hay algunas innovaciones más que podrían interesarle, como el uso de conexiones de salto para recuperar parte de la resolución espacial que se pierde en la CNN (hablaremos de esto en breve cuando veamos la segmentación semántica). Además, en el artículo de 2016, los autores presentan el modelo YOLO9000 que usa clasificación jerárquica: el modelo predice una probabilidad para cada nodo en una jerarquía visual llamada *Árbol de palabras*. Esto hace posible que la red prediga con gran confianza que una imagen representa, por ejemplo, un perro, aunque no esté seguro de qué tipo específico de perro se trata.

Así que lo animo a seguir adelante y leer los tres documentos: son bastante agradables de leer y es un excelente ejemplo de cómo los sistemas de aprendizaje profundo se pueden mejorar gradualmente.

Precisión media media (mAP)

Una métrica muy común utilizada en tareas de detección de objetos es la*Precisión media media* (mapa). "Mean Average" suena un poco redundante, ¿no? Para entender esta métrica, volvamos a dos métricas de clasificación que discutimos enCapítulo 3: precisión y recuerdo. Recuerde la compensación: cuanto mayor sea el recuerdo, menor será la precisión. Puede visualizar esto en una curva de Precisión/Recuperación (verFigura 3-5). Para resumir esta curva en un solo número, podríamos calcular su Área bajo la curva (AUC). Pero tenga en cuenta que la curva Precisión/Recuperación puede contener algunas secciones en las que la precisión aumenta cuando aumenta la recuperación, especialmente en valores bajos de recuperación (puede ver esto en la parte superior izquierda deFigura 3-5). Esta es una de las motivaciones para la métrica mAP.

Supongamos que el clasificador tiene una precisión del 90 % con una recuperación del 10 %, pero una precisión del 96 % con una recuperación del 20 %: realmente no hay compensación aquí: simplemente tiene más sentido usar el clasificador con una recuperación del 20 % en lugar de una recuperación del 10 %, como obtendrá un mayor recuerdo y una mayor precisión. Así que en lugar de mirar la precisión*a*10% recuerda, realmente deberíamos estar mirando el*máximo*precisión que el clasificador puede ofrecer con *al menos*10% de recuerdo. Sería el 96%, no el 90%. Entonces, una forma de tener una idea clara del rendimiento del modelo es calcular la precisión máxima que puede obtener con al menos 0 % de recuperación, luego 10 % de recuperación, 20 %, etc. hasta 100 %, y luego calcular la media de estas máximas precisiones. Esto se llama el*Precisión promedio*(AP) métrica. Ahora, cuando hay más de 2 clases, podemos calcular el AP para cada clase y luego calcular el AP medio (mAP). ¡Eso es todo!

Sin embargo, en los sistemas de detección de objetos, hay un nivel adicional de complejidad: ¿qué pasa si el sistema detecta la clase correcta, pero en la ubicación incorrecta (es decir, el cuadro delimitador está completamente fuera de lugar)? Seguramente no deberíamos considerar esto como una predicción positiva. Entonces, un enfoque es definir un umbral de IOU: por ejemplo, podemos considerar que una predicción es correcta solo si el IOU es mayor que, digamos, 0.5, y la clase pronosticada es correcta. El mAP correspondiente generalmente se indica mAP@0.5 (o mAP@50 %, o a veces solo AP_{50}). En algunas competiciones (como el desafío Pascal VOC), esto es lo que se hace. En otros (como la competencia COCO), el mAP se calcula para diferentes umbrales de IOU (0.50, 0.55, 0.60, ..., 0.95), y la métrica final es la media de todos estos mAP (anotado AP@ [.50:. 95] o AP@ [.50:0.05:.95]). Sí, eso es un promedio medio medio.

Varias implementaciones de YOLO creadas con TensorFlow están disponibles en github, algunas con pesos previamente entrenados. Al momento de escribir, se basan en TensorFlow 1, pero cuando lea esto, las implementaciones de TF 2 seguramente estarán disponibles. Además, otros modelos de detección de objetos están disponibles en el proyecto TensorFlow Models, muchos

con pesos previamente entrenados, y algunos incluso se han transferido a TF Hub, lo que los hace extremadamente fáciles de usar, comoSSD[29]yMás Rápido-RCNN.[30], que son bastante populares. SSD también es un modelo de detección de "disparo único", bastante similar a YOLO, mientras que Faster R-CNN es más complejo: la imagen primero pasa por una CNN y la salida se pasa a una Red de propuesta regional (RPN) que propone cuadros delimitadores. que es más probable que contengan un objeto, y se ejecuta un clasificador para cada cuadro delimitador, en función de la salida recortada de la CNN.

La elección del sistema de detección depende de muchos factores: velocidad, precisión, modelos preentrenados disponibles, tiempo de entrenamiento, complejidad, etc. Los documentos contienen tablas de métricas, pero hay mucha variabilidad en los entornos de prueba y la tecnología. - Las teorías evolucionan tan rápido que es difícil hacer una comparación justa que sea útil para la mayoría de las personas y siga siendo válida durante más de unos pocos meses.

¡Excelente! Entonces podemos ubicar objetos dibujando cuadros delimitadores alrededor de ellos. Pero quizás quieras ser un poco más preciso. Veamos cómo bajar al nivel de píxeles.

Segmentación Semántica

En*segmentación semántica*, cada píxel se clasifica según la clase del objeto al que pertenece (p. ej., carretera, automóvil, peatón, edificio, etc.), como se muestra enFigura 14-26. Tenga en cuenta que diferentes objetos de la misma clase son*no*distinguido. Por ejemplo, todas las bicicletas en el lado derecho de la imagen segmentada terminan como una gran masa de píxeles. La principal dificultad en esta tarea es que cuando las imágenes pasan por una CNN normal, pierden gradualmente su resolución espacial (debido a las capas con zancadas mayores que 1): por lo que una CNN normal puede terminar sabiendo que hay una persona en el image, en algún lugar de la parte inferior izquierda de la imagen, pero no será mucho más preciso que eso.

29 “SSD: Detector MultiBox de disparo único”, Wei Liu et al. (2015).

30 “R-CNN más rápido: hacia la detección de objetos en tiempo real con redes de propuestas regionales”, Shaoqing Ren et al. (2015).

Figura 14-26. Segmentación semántica

Al igual que para la detección de objetos, existen muchos enfoques diferentes para abordar este problema, algunos bastante complejos. Sin embargo, en el artículo de 2015 de Jonathan Long et al. se propuso una solución bastante sencilla. discutimos anteriormente. Comienzan tomando una CNN previamente entrenada y convirtiéndose en una FCN, como se discutió anteriormente. La CNN aplica un paso de 32 a la imagen de entrada en general (es decir, si suma todos los pasos mayores que 1), lo que significa que la última capa genera mapas de características que son 32 veces más pequeños que la imagen de entrada. Esto es claramente demasiado grueso, por lo que agregan un solo *capa de muestreo superior* que multiplica la resolución por 32. Hay varias soluciones disponibles para el sobremuestreo (aumento del tamaño de una imagen), como la interpolación bilineal, pero solo funciona razonablemente bien hasta ×4 o ×8. En su lugar, utilizaron un *capa convolucional transpuesta*: [31] es equivalente a primero estirar la imagen insertando filas y columnas vacías (llenas de ceros), luego realizando una convolución regular (ver Figura 14-27). Alternativamente, algunas personas prefieren pensar en ella como una capa convolucional regular que utiliza pasos fraccionarios (p. ej., 1/2 pulgada). Figura 14-27). los *capa convolucional transpuesta* puede inicializarse para realizar algo parecido a la interpolación lineal, pero dado que es una capa entrenable, aprenderá a hacerlo mejor durante el entrenamiento.

31 Este tipo de capa a veces se denomina *capa de deconvolución*, pero lo hace *no* realizar lo matemático Los ciudadanos llaman desconvolución, por lo que debe evitarse este nombre.

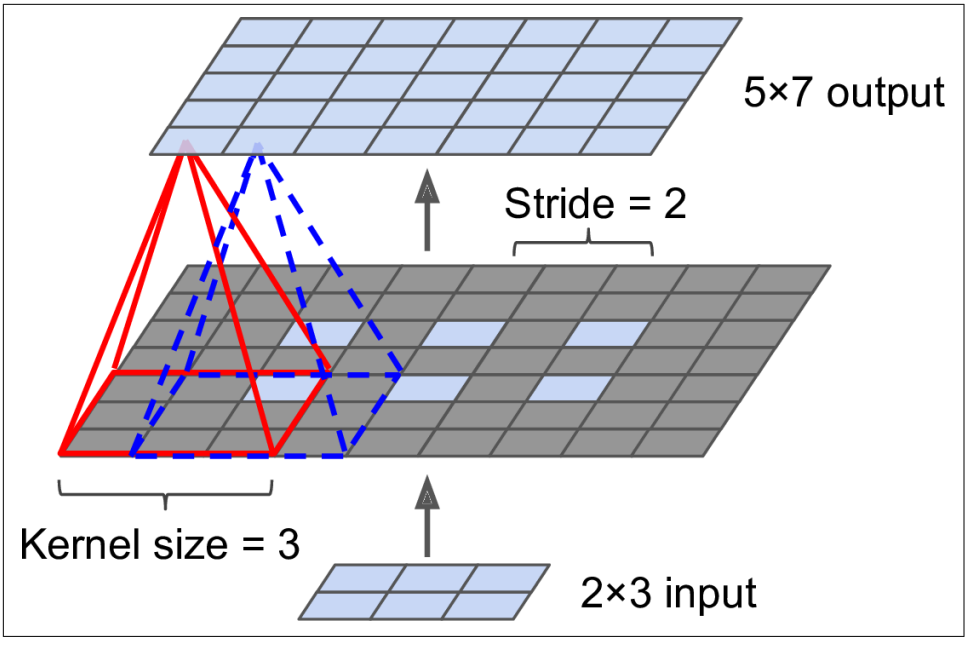

Figura 14-27. Sobremuestreo usando una capa convolucional de transposición

En una capa de convolución transpuesta, la zancada define cuánto se estirará la entrada, no el tamaño de los pasos del filtro, por lo que cuanto mayor sea la zancada, mayor será la salida (a diferencia de las capas convolucionales o las capas de agrupación).

Operaciones de convolución de TensorFlow

TensorFlow también ofrece algunos otros tipos de capas convolucionales:

- keras.layers.Conv1Dcrea una capa convolucional para entradas 1D, como series temporales o texto (secuencias de letras o palabras), como veremos en???.
- keras.layers.Conv3Dcrea una capa convolucional para entradas 3D, como un escaneo PET 3D.
- Configuración detasa_de_dilataciónhiperparámetro de cualquier capa convolucional a un valor de 2 o más crea un*capa convolucional à-trous*(“à trous” en francés significa “con agujeros”). Esto es equivalente a usar una capa convolucional regular con un filtro dilatado al insertar filas y columnas de ceros (es decir, agujeros). Por ejemplo, un filtro de 1 × 3 igual a [[1,2,3]]se puede dilatar con*tasa de dilatación*de 4, resultando en un *filtro dilatado*[[1, 0, 0, 0, 2, 0, 0, 0, 3]].Esto permite que la capa convolucional

tener un campo receptivo más grande sin costo computacional y sin utilizar parámetros adicionales.

- tf.nn.profundidad_conv2d()se puede utilizar para crear un*capa convolucional en profundidad* (pero necesita crear las variables usted mismo). Aplica cada filtro a cada canal de entrada individual de forma independiente. Así, si hay*fn*filtros y*fn*canales de entrada, entonces esto generará*fn*×*fn*mapas de características.

Esta solución está bien, pero sigue siendo demasiado imprecisa. Para hacerlo mejor, los autores agregaron conexiones de salto desde las capas inferiores: por ejemplo, aumentaron la muestra de la imagen de salida en un factor de 2 (en lugar de 32) y agregaron la salida de una capa inferior que tenía esta doble resolución. Luego sobremuestrearon el resultado por un factor de 16, lo que llevó a un factor de sobremuestreo total de 32 (verFigura 14-28). Esto recuperó parte de la resolución espacial que se perdió en las capas de agrupación anteriores. En su mejor arquitectura, usaron una segunda conexión de salto similar para recuperar detalles aún más finos de una capa aún más baja: en resumen, la salida de la CNN original pasa por los siguientes pasos adicionales: escalar ×2, agregar la salida de una capa más baja (de la escala adecuada), aumentar la escala ×2, agregar la salida de una capa aún más baja y, finalmente, aumentar la escala ×8. Incluso es posible escalar más allá del tamaño de la imagen original: esto se puede usar para aumentar la resolución de una imagen, que es una técnica llamada*súper resolución*.

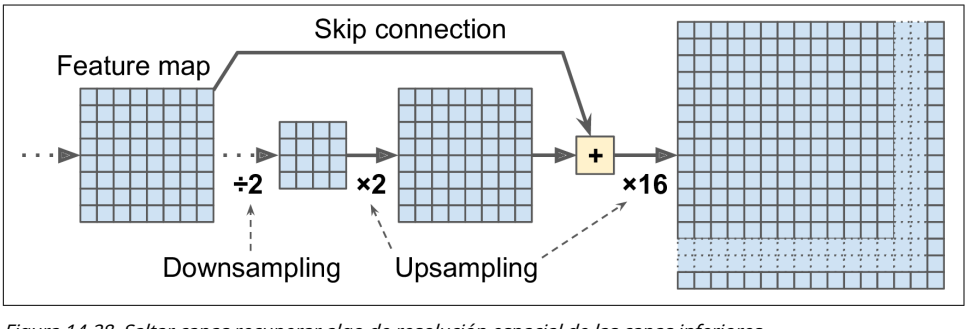

Figura 14-28. Saltar capas recuperar algo de resolución espacial de las capas inferiores

Una vez más, muchos repositorios de github brindan implementaciones de segmentación semántica de TensorFlow (TensorFlow 1 por ahora), e incluso encontrará un *segmentación de instancias*modelo en el proyecto TensorFlow Models. La segmentación de instancias es similar a la segmentación semántica, pero en lugar de fusionar todos los objetos de la misma clase en una gran masa, cada objeto se distingue de los demás (por ejemplo, identifica cada bicicleta individual). En la actualidad, proporcionan múltiples implementaciones del*Máscara R-CNN* arquitectura, que fue propuesta en unpapel de 2017: amplía el modelo Faster R-CNN al producir adicionalmente una máscara de píxeles para cada cuadro delimitador. Entonces, no solo obtiene un cuadro delimitador alrededor de cada objeto, con un conjunto de probabilidades de clase estimadas, sino que también obtiene una máscara de píxeles que ubica los píxeles en el cuadro delimitador que pertenecen al objeto.

Como puede ver, el campo de Deep Computer Vision es amplio y se mueve rápidamente, con todo tipo de arquitecturas surgiendo cada año, todas basadas en redes neuronales convolucionales. El progreso realizado en solo unos pocos años ha sido asombroso, y los investigadores ahora se están enfocando en problemas cada vez más difíciles, como *aprendizaje adversario* (que intenta hacer la red más resistente a las imágenes diseñadas para engañarla), explicabilidad (entender por qué la red hace una clasificación específica), realista *generación de imágenes* (al que volveremos en ???), *aprendizaje de un solo disparo* (un sistema que puede reconocer un objeto después de haberlo visto solo una vez), y mucho más. Algunos incluso exploran arquitecturas completamente novedosas, como la de Geoffrey Hinton. *redes de cápsulas*[32] (Los presenté en un par vídeos, con el código correspondiente en un cuaderno). Pasemos ahora al siguiente capítulo, donde veremos cómo procesar datos secuenciales, como series temporales, utilizando redes neuronales recurrentes y redes neuronales convolucionales.

Ejercicios

1. ¿Cuáles son las ventajas de una CNN sobre una DNN completamente conectada para la clasificación de imágenes?
2. Considere una CNN compuesta por tres capas convolucionales, cada una con núcleos de 3 × 3, un paso de 2 y el MISMO relleno. La capa más baja genera 100 mapas de características, la del medio genera 200 y la superior genera 400. Las imágenes de entrada son imágenes RGB de 200 × 300 píxeles. ¿Cuál es el número total de parámetros en la CNN?
   Si usamos flotantes de 32 bits, ¿cuánta RAM requerirá esta red al menos al hacer una predicción para una sola instancia? ¿Qué pasa cuando se entrena en un mini-lote de 50 imágenes?
3. Si su GPU se queda sin memoria mientras entrena una CNN, ¿cuáles son cinco cosas que podría intentar para resolver el problema?
4. ¿Por qué querría agregar una capa de agrupación máxima en lugar de una capa convolucional con el mismo paso?
5. ¿Cuándo le gustaría agregar un *normalización de la respuesta local* ¿capa?
6. ¿Puede nombrar las principales innovaciones de AlexNet, en comparación con LeNet-5? ¿Qué pasa con las principales innovaciones en GoogLeNet, ResNet, SENet y Xception?
7. ¿Qué es una red totalmente convolucional? ¿Cómo se puede convertir una capa densa en una capa convolucional?
8. ¿Cuál es la principal dificultad técnica de la segmentación semántica?
9. Cree su propia CNN desde cero e intente lograr la mayor precisión posible en MNIST.

32 “Cápsulas de matriz con enrutamiento EM”, G. Hinton, S. Sabour, N. Frost (2018).

10. Utilice el aprendizaje por transferencia para la clasificación de imágenes grandes.

una. Cree un conjunto de entrenamiento que contenga al menos 100 imágenes por clase. Por ejemplo, puede clasificar sus propias imágenes en función de la ubicación (playa, montaña, ciudad, etc.) o, alternativamente, puede usar un conjunto de datos existente (p. ej., de TensorFlow Datasets).

b. Divídalo en un conjunto de entrenamiento, un conjunto de validación y un conjunto de prueba.

C. Cree la canalización de entrada, incluidas las operaciones de preprocesamiento adecuadas y, opcionalmente, agregue el aumento de datos.

d. Ajuste un modelo previamente entrenado en este conjunto de datos.

11. Ir a través de TensorFlowTutorial de sueño profundo. Es una forma divertida de familiarizarse con varias formas de visualizar los patrones aprendidos por una CNN y generar arte usando Deep Learning.

Las soluciones a estos ejercicios están disponibles en???.

Sobre el Autor

Aurelien Gerones un consultor de Machine Learning. Ex miembro de Google, dirigió el equipo de clasificación de videos de YouTube de 2013 a 2016. También fue fundador y CTO de Wifirst de 2002 a 2012, un ISP inalámbrico líder en Francia; y fundador y CTO de Polyconseil en 2001, la empresa que ahora gestiona el servicio de coches compartidos eléctricos Autolib'.

Antes de esto, trabajó como ingeniero en una variedad de dominios: finanzas (JP Morgan y Société Générale), defensa (DOD de Canadá) y atención médica (transfusión de sangre). Publicó algunos libros técnicos (sobre C ++, WiFi y arquitecturas de Internet) y fue profesor de informática en una escuela de ingeniería francesa.

Algunos datos curiosos: enseñó a sus tres hijos a contar en binario con los dedos (hasta 1023), estudió microbiología y genética evolutiva antes de dedicarse a la ingeniería de software y su paracaídas no se abrió en el segundo salto.

Colofón

El animal de la portada de*Aprendizaje automático práctico con Scikit-Learn y TensorFlow*es la salamandra de fuego (*salamandra salamandra*), un anfibio que se encuentra en la mayor parte de Europa. Su piel negra y brillante presenta grandes manchas amarillas en la cabeza y la espalda, lo que indica la presencia de toxinas alcaloides. Esta es una posible fuente del nombre común de este anfibio: el contacto con estas toxinas (que también pueden rociar distancias cortas) provoca convulsiones e hiperventilación. Los venenos dolorosos o la humedad de la piel de la salamandra (o ambos) llevaron a la creencia equivocada de que estas criaturas no solo podían sobrevivir al fuego, sino que también podían extinguirlo.

Las salamandras de fuego viven en bosques sombreados, escondidas en grietas húmedas y debajo de troncos cerca de las piscinas u otros cuerpos de agua dulce que facilitan su reproducción. Aunque pasan la mayor parte de su vida en la tierra, dan a luz a sus crías en el agua. Subsisten principalmente con una dieta de insectos, arañas, babosas y gusanos. Las salamandras de fuego pueden crecer hasta un pie de largo y, en cautiverio, pueden vivir hasta 50 años.

El número de salamandras rojas se ha reducido por la destrucción de su hábitat forestal y su captura para el comercio de mascotas, pero la mayor amenaza es la susceptibilidad de su piel permeable a la humedad a los contaminantes y microbios. Desde 2014, se extinguieron en partes de los Países Bajos y Bélgica debido a la introducción de un hongo.

Muchos de los animales de las portadas de O'Reilly están en peligro de extinción; todos ellos son importantes para el mundo. Para obtener más información sobre cómo puede ayudar, vaya a*animales.oreilly.com*.

La imagen de portada es de*Historia natural ilustrada de Wood*. Las fuentes de la portada son URW Typewriter y Guardian Sans. La fuente del texto es Adobe Minion Pro; la fuente del encabezado es Adobe Myriad Condensed; y la fuente del código es Ubuntu Mono de Dalton Maag.