**[Redes neuronales de aprendizaje profundo](https://sitiobigdata.com/2019/07/08/redes-neuronales-aprendizaje-profundo/" \o "Redes neuronales de aprendizaje profundo)**

Redes neuronales de aprendizaje profundo posee la convolución y la capa convolucional que son los principales bloques de construcción utilizados en las redes neuronales convolucionales.

Una convolución es la simple aplicación de un filtro a una entrada que da como resultado una activación.

La aplicación repetida del mismo filtro a una entrada da como resultado un mapa de activaciones denominado mapa de características. Que indica la ubicación y la fuerza de una característica detectada en una entrada, como una imagen.

La innovación de las redes neuronales de aprendizaje profundo convolucionales es la capacidad para aprender automáticamente un gran número de filtros en paralelo específico a un conjunto de datos de entrenamiento bajo las restricciones de un problema de modelado predictivo específico.

Como la clasificación de imágenes. El resultado son características altamente específicas que se pueden detectar en cualquier lugar en las imágenes de entrada.

En este tutorial, descubrirá cómo funcionan las convoluciones en la red neuronal convolucional.

Después de completar este tutorial, sabrá:

* Convolutional Las redes neuronales de aprendizaje profundo aplican un filtro a una entrada para crear un mapa de características que resume la presencia de las características detectadas en la entrada.
* Los filtros pueden ser hechos a mano, como los detectores de línea, pero la innovación de las redes neuronales de aprendizaje profundo convolucionales es aprender los filtros durante entrenamiento en el contexto de un problema de predicción específico.
* Cómo calcular el mapa de características para capas convolucionales de una y dos dimensiones en una red neuronal convolucional.

**Redes neuronales de aprendizaje profundo: Descripción general del tutorial**

Este tutorial está dividido en cuatro partes; son:

1. Convolución en redes neuronales de aprendizaje profundo
2. Convolución en visión computacional
3. Potencia de los filtros aprendidos
4. Ejemplo de capas convolucionales

**Redes neuronales de aprendizaje profundo: Convolución en redes neuronales**

Central a la red neuronal convolucional es la capa convolucional que da nombre a la red. Esta capa realiza una operación llamada “*convolución*“.

En el contexto de una red neuronal convolucional, una convolución es una operación lineal que implica la multiplicación de un conjunto de pesos con la entrada. Muy similar a Una red neuronal tradicional.

**Dado que la técnica fue diseñada para una entrada bidimensional, la multiplicación se realiza entre una matriz de datos de entrada y una matriz bidimensional de ponderaciones, llamada filtro o kernel.**

El filtro es más pequeño que la información de entrada y el tipo de multiplicación aplicada entre un parche de tamaño de filtro de la entrada y el filtro es un producto de puntos.

Un [punto producto](https://en.wikipedia.org/wiki/Dot_product)es la multiplicación de elementos entre el parche del tamaño de filtro de la entrada y el filtro, que luego se suma, dando como resultado un solo valor. Debido a que da como resultado un solo valor, la operación a menudo se denomina “*producto escalar*“.

El uso de un filtro más pequeño que la entrada es intencional ya que permite el mismo filtro (conjunto de pesos) para ser multiplicado por la matriz de entrada varias veces en diferentes puntos de la entrada.

Específicamente, el filtro se aplica sistemáticamente a cada parte superpuesta o parche del tamaño de filtro de los datos de entrada, de izquierda a derecha, de arriba a abajo.

Esta aplicación sistemática del mismo filtro a través de una imagen es una idea poderosa. Si el filtro está diseñado para detectar un tipo específico de característica en la entrada.

Entonces la aplicación de ese filtro sistemáticamente en toda la imagen de entrada le permite al filtro la oportunidad de descubrir esa característica en cualquier lugar de la imagen.

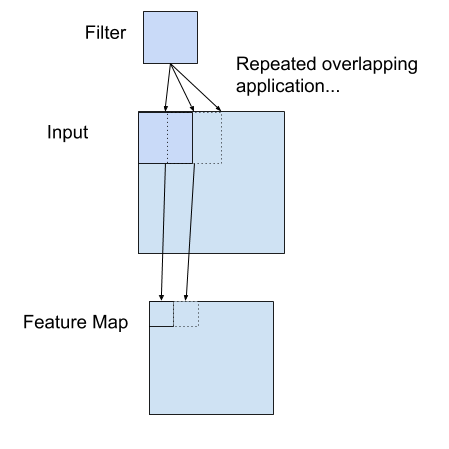
Esta capacidad se conoce comúnmente como invariancia de la traducción, por ejemplo, el interés general en si la característica está presente en lugar de donde estaba presente.

La invariabilidad a la traducción local puede ser una propiedad muy útil si nos importa más si alguna característica está presente en lugar de dónde está exactamente. Por ejemplo, al determinar si una imagen contiene una cara, no necesitamos conocer la ubicación de los ojos con una precisión de píxel, solo necesitamos saber que hay un ojo en el lado izquierdo de la cara y un ojo en la derecha. lado de la cara.

La salida de multiplicar el filtro con la matriz de entrada una vez es un solo valor. Como el filtro se aplica varias veces a la matriz de entrada.

El resultado es una matriz bidimensional de valores de salida que representan un filtrado de la entrada. Como tal, la matriz de salida bidimensional de esta operación se denomina “*mapa de características*“.

Una vez que se crea un mapa de características, podemos pasar cada valor en el mapa de características a través de una no linealidad, como como ReLU, como hacemos con las salidas de una capa completamente conectada.



**Ejemplo de un filtro aplicado a una entrada bidimensional para crear un mapa de características**

Si proviene de un campo de procesamiento de señal digital o área relacionada de matemáticas, puede entender que la operación de convolución en una matriz es algo diferente.

Específicamente, el filtro (kernel) se voltea antes de ser aplicado a la entrada. Técnicamente, la convolución como se describe en el uso de de aprendizaje profundo convolucionales es en realidad una “*correlación cruzada”*.

Sin embargo, en las redes neuronales de aprendizaje profundo, se le conoce como una operación de “*convolución*“.

Muchas bibliotecas de aprendizaje automático implementan correlación cruzada, pero se denomina convolución.

En resumen, tenemos una entrada como una imagen de valores de píxeles, y un filtro que es un conjunto de pesos , y el filtro se aplica sistemáticamente a los datos de entrada para crear un mapa de características .

**Convolución en la visión por computadora**

La idea de aplicar la operación convolucional a los datos de imagen no es nueva ni única a la convolucional Redes neuronales; es una técnica común utilizada en la visión artificial.

Históricamente, los filtros fueron diseñados a mano por expertos en visión artificial, que luego se aplicaron a una imagen para obtener un mapa de características o un resultado de la aplicación del filtro y luego realiza el análisis de la imagen. la imagen es más fácil de alguna manera.

Por ejemplo, a continuación se muestra un filtro de 3 × 3 elementos hechos a mano para detectar líneas verticales:

0.0, 1.0, 0.0  
0.0, 1.0, 0.0  
0.0, 1.0, 0.0

La ​​aplicación de este filtro a una imagen dará como resultado un mapa de características que solo contiene líneas verticales. Es un detector de línea vertical.

Puede ver esto desde los valores de peso en el filtro; cualquier valor de píxeles en la línea vertical central se activará positivamente y cualquiera de los lados se activará negativamente. Arrastrar este filtro sistemáticamente a través de los valores de píxeles en una imagen solo puede resaltar los píxeles de la línea vertical.

También se podría crear un detector de línea horizontal y aplicar a la imagen, por ejemplo:

0.0, 0.0, 0.0  
1.0, 1.0, 1.0  
0.0, 0.0, 0.0

Combinando los resultados de ambos filtros, por ej. Al combinar ambos mapas de características, todas las líneas de una imagen serán resaltadas.

Se puede diseñar un conjunto de decenas o incluso cientos de otros filtros pequeños para detectar otras características en la imagen.

La innovación de usar La operación de convolución en una red neuronal es que los valores del filtro son los pesos que deben aprenderse durante el entrenamiento de la red.

La red aprenderá qué tipos de funciones extraer de la entrada. Específicamente, en el entrenamiento bajo descenso de gradiente estocástico.

La red se ve obligada a aprender a extraer características de la imagen que minimizan la pérdida de la tarea específica para la cual se está entrenando la red, por ejemplo. Extraiga las características que son más útiles para clasificar imágenes como perros o gatos.

En este contexto, puede ver que esta es una idea poderosa.

**Potencia de los filtros aprendidos**

Aprender un filtro único específico para una máquina La tarea de aprendizaje es una técnica poderosa.

Sin embargo, las redes neuronales convolucionales logran mucho más en la práctica.

**Filtros múltiples**

Las redes neuronales convolucionales no aprenden un solo filtro; de hecho, aprenden múltiples características en paralelo para una entrada dada.

Por ejemplo, es común que una capa convolucional aprenda de 32 a 512 filtros en paralelo para una entrada dada.

Esto le da al modelo 32 , o incluso 512, diferentes formas de extraer características de una entrada, o muchas formas diferentes de “*aprendiendo a ver*” y después del entrenamiento, muchas formas diferentes de “*viendo*” la entrada

Esta diversidad permite la especialización, por ejemplo, no solo las líneas, sino las líneas específicas que se ven en sus datos de entrenamiento específicos.

**Múltiples canales**

Las imágenes en color tienen múltiples canales, generalmente uno para cada canal de color, como rojo, verde y azul.

perspectiva de datos, lo que significa que una sola imagen proporcionada como entrada al modelo es, de hecho, tres imágenes.

Un filtro siempre debe tener el mismo número de canales que la entrada, a menudo denominado “*profundidad*“. Si una imagen de entrada tiene 3 canales (por ejemplo, una profundidad de 3), entonces un filtro aplicado a esa imagen también debe tener 3 canales (por ejemplo, una profundidad de 3).

En este caso, un filtro de 3 × 3 sería de hecho 3x3x3 o [3, 3, 3] para filas, columnas y profundidad. Independientemente de la profundidad de la entrada y la profundidad del filtro. El filtro se aplica a la entrada mediante una operación de producto puntual que produce un solo valor.

Esto significa que si una capa convolucional tiene 32 filtros, estos 32 filtros son no solo bidimensional para la entrada de imagen bidimensional.

Sino también tridimensional, con pesos de filtro específicos para cada uno de los tres canales. Sin embargo, cada filtro da como resultado un único mapa de características.

Lo que significa que la profundidad de la salida de la aplicación de la capa convolucional con 32 filtros es 32 para los 32 mapas de características creados.

**Redes neuronales de aprendizaje profundo: Capas múltiples**

Las capas convolucionales no se aplican solo a los datos de entrada, por ejemplo. valores de píxel sin procesar, pero también pueden aplicarse a la salida de otras capas.

El apilamiento de capas convolucionales permite una descomposición jerárquica de la entrada.

Considere que los filtros que operan directamente en los valores de píxel sin procesar aprenderán para extraer entidades de bajo nivel, como líneas.

Los filtros que operan en la salida de las capas de la primera línea pueden extraer características que son combinaciones de características de nivel inferior, como las características que comprenden varias líneas para expresar formas. [19659005] Este proceso continúa hasta que capas muy profundas extraen caras, animales, casas, etc.

Esto es exactamente lo que vemos en la práctica. A medida que aumenta la profundidad de la red, aumenta la abstracción de funciones a órdenes altas y superiores.

**Ejemplo resuelto de capas convolucionales**

La biblioteca de redes neuronales de aprendizaje profundo de Keras proporciona un conjunto de capas convolucionales.

Podemos entender mejor la convolución Operación mirando algunos ejemplos elaborados con datos artificiales y filtros hechos a mano.

En esta sección, veremos una capa convolucional unidimensional y un ejemplo de capa convolucional bidimensional para hacer que la operación de convolución sea concreta y proporcione un ejemplo práctico del uso de las capas Keras.

**Ejemplo de la capa convolucional 1D**

Podemos definir una entrada unidimensional que tiene ocho elementos, todos con el valor de 0.0, con una protuberancia de dos elementos en el centro con los valores 1.0.  
[0, 0, 0, 1, 1, 0, 0, 0]  
La ​​entrada a Keras debe ser tridimensional para una capa convolucional 1D.

La primera dimensión se refiere a cada muestra de entrada; En este caso, solo tenemos una muestra. La segunda dimensión se refiere a la longitud de cada muestra; En este caso, la longitud es de ocho. La tercera dimensión se refiere al número de canales en cada muestra; en este caso, solo tenemos un canal.

Por lo tanto, la forma de la matriz de entrada será [1, 8, 1].

# define input data  
data = asarray([0, 0, 0, 1, 1, 0, 0, 0])  
data = data.reshape(1, 8, 1)

Definiremos un modelo que espera que las muestras de entrada tengan la forma [8, 1].

El modelo tendrá un solo filtro con la forma de 3 o tres Elementos de ancho. Keras se refiere a la forma del filtro como *kernel\_size*.

# create model  
model = Sequential()  
model.add(Conv1D(1, 3, input\_shape=(8, 1)))

Por defecto, los filtros en una capa convolucional se inicializan con pesos aleatorios. En este ejemplo creado, especificaremos manualmente los pesos para el filtro único. Definiremos un filtro que sea capaz de detectar baches, que es un valor de entrada alto rodeado de valores de entrada bajos, como definimos en nuestro ejemplo de entrada.

El filtro de tres elementos que definiremos tiene el siguiente aspecto:  
[0, 1, 0]  
La ​​capa convolucional también tiene un valor de entrada de polarización que también requiere un peso que estableceremos en cero.

Por lo tanto, podemos forzar los pesos de nuestra capa convolucional unidimensional para usar nuestro filtro artesanal de la siguiente manera :

# define a vertical line detector  
weights = [asarray([[[0]],[[1]],[[0]]]), asarray([0.0])]  
# store the weights in the model  
model.set\_weights(weights)

Los pesos deben especificarse en una estructura tridimensional, en términos de filas, columnas y canales. El filtro tiene una sola fila, tres columnas y un canal.

Podemos recuperar los pesos y confirmar que se configuraron correctamente.

# confirm they were stored  
print(model.get\_weights())

Finalmente, podemos aplicar el filtro único a nuestros datos de entrada.

Podemos lograr esto llamando a la función *predict ()*en el modelo. Esto devolverá el mapa de características directamente: es el resultado de aplicar el filtro sistemáticamente a través de la secuencia de entrada.

# apply filter to input data  
yhat = model.predict(data)  
print(yhat)

Al vincular todo esto, el ejemplo completo se muestra a continuación.

# example of calculation 1d convolutions  
from numpy import asarray  
from keras.models import Sequential  
from keras.layers import Conv1D  
# define input data  
data = asarray([0, 0, 0, 1, 1, 0, 0, 0])  
data = data.reshape(1, 8, 1)  
# create model  
model = Sequential()  
model.add(Conv1D(1, 3, input\_shape=(8, 1)))  
# define a vertical line detector  
weights = [asarray([[[0]],[[1]],[[0]]]), asarray([0.0])]  
# store the weights in the model  
model.set\_weights(weights)  
# confirm they were stored  
print(model.get\_weights())  
# apply filter to input data  
yhat = model.predict(data)  
print(yhat)

Al ejecutar el ejemplo, primero se imprimen los pesos de la red; esa es la confirmación de que nuestro filtro artesanal se configuró en el modelo como esperábamos.

A continuación, el filtro se aplica al patrón de entrada y el mapa de características se calcula y se muestra. Podemos ver en los valores del mapa de características que la protuberancia se detectó correctamente.

[array([[[0.]],  
[[1.]],  
[[0.]]], dtype=float32), array([0.], dtype=float32)]  
[[[0.]  
[0.]  
[1.]  
[1.]  
[0.]  
[0.]]]

Veamos más de cerca lo que sucedió aquí.

Recuerde que la entrada es un vector de ocho elementos con los valores:

[0, 0, 0, 1, 1, 0, 0, 0].

Primero, el filtro de tres elementos [0, 1, 0] se aplicó a las primeras tres entradas de la entrada [0, 0, 0] al calcular el producto punto (operador “.”), Lo que dio como resultado un valor de salida único en el mapa de características de cero.

Recuerde que un producto punto es la suma de las multiplicaciones de elementos, o aquí es (0 x 0) + (1 x 0) + (0 x 0) = 0. En NumPy.

Esto se puede implementar manualmente como:

from numpy import asarray  
print(asarray([0, 1, 0]).dot(asarray([0, 0, 0])))

En nuestro ejemplo de manual, esto es como sigue:

[0, 1, 0]. [0, 0, 0] = 0

El filtro se movió a lo largo de un elemento de la secuencia de entrada y se repitió el proceso. Específicamente, el mismo filtro se aplicó a la secuencia de entrada en los índices 1, 2 y 3, lo que también resultó en una salida cero en el mapa de características.

[0, 1, 0] . [0, 0, 1] = 0

Estamos siendo sistemáticos, así que de nuevo, el filtro se mueve a lo largo de un elemento más de la entrada y se aplica a la entrada en los índices 2, 3 y 4. Esta vez la salida es un valor de uno en el mapa de características.

Detectamos la característica y activamos apropiadamente.

[0, 1, 0] . [0, 1, 1] = 1

El proceso se repite hasta que calculamos el mapa de características completo.

[0, 0, 1, 1, 0, 0]

Tenga en cuenta que el mapa de características tiene seis elementos, mientras que nuestra entrada tiene ocho elementos.

Este es un artefacto de cómo se aplicó el filtro a la secuencia de entrada. Hay otras formas de aplicar el filtro a la secuencia de entrada que cambia la forma del mapa de características resultante, como el relleno. Pero no discutiremos estos métodos en esta publicación.

Puedes imaginar que con diferentes entradas, podemos detectamos la característica con más o menos intensidad, y con diferentes pesos en el filtro, que detectaríamos diferentes características en la secuencia de entrada.

**Ejemplo de Capa Convolucional 2D**

Podemos expandir el ejemplo de detección de bultos en la sección anterior a un detector de línea vertical en una imagen bidimensional.

Nuevamente, podemos restringir la entrada, en este caso a una imagen de entrada cuadrada de 8 × 8 píxeles con un solo canal (por ejemplo, escala de grises) con una sola línea vertical en el medio.

[0, 0, 0, 1, 1, 0, 0, 0]  
[0, 0, 0, 1, 1, 0, 0, 0]  
[0, 0, 0, 1, 1, 0, 0, 0]  
[0, 0, 0, 1, 1, 0, 0, 0]  
[0, 0, 0, 1, 1, 0, 0, 0]  
[0, 0, 0, 1, 1, 0, 0, 0]  
[0, 0, 0, 1, 1, 0, 0, 0]  
[0, 0, 0, 1, 1, 0, 0, 0]

La ​​entrada a una capa Conv2D debe ser de cuatro dimensiones.

La primera dimensión define las muestras; En este caso, solo hay una muestra. La segunda dimensión define el número de filas.

En este caso, ocho. La tercera dimensión define el número de columnas, nuevamente ocho en este caso, y finalmente el número de canales, que es uno en este caso.

Por lo tanto, la entrada debe tener la forma de cuatro dimensiones [samples, columns, rows, channels] o [1, 8, 8, 1] en este caso.

# define input data  
data = [[0, 0, 0, 1, 1, 0, 0, 0],  
[0, 0, 0, 1, 1, 0, 0, 0],  
[0, 0, 0, 1, 1, 0, 0, 0],  
[0, 0, 0, 1, 1, 0, 0, 0],  
[0, 0, 0, 1, 1, 0, 0, 0],  
[0, 0, 0, 1, 1, 0, 0, 0],  
[0, 0, 0, 1, 1, 0, 0, 0],  
[0, 0, 0, 1, 1, 0, 0, 0]]  
data = asarray(data)  
data = data.reshape(1, 8, 8, 1)

Definiremos el Conv2D con un solo filtro como hicimos en la sección anterior con el ejemplo de Conv1D.

El filtro será bidimensional y cuadrado. Con la forma 3 × 3. La capa esperará que las muestras de entrada tengan la forma [columns, rows, channels] o [8,8,1].

# create model  
model = Sequential()  
model.add(Conv2D(1, (3,3), input\_shape=(8, 8, 1)))

Definiremos un filtro detector de línea vertical para detectar la línea vertical única en nuestros datos de entrada.

El filtro tiene el siguiente aspecto:

0, 1, 0  
0, 1, 0  
0, 1, 0

Podemos implementar esto de la siguiente manera:

# define a vertical line detector  
detector = [[[[0]],[[1]],[[0]]],  
[[[0]],[[1]],[[0]]],  
[[[0]],[[1]],[[0]]]]  
weights = [asarray(detector), asarray([0.0])]  
# store the weights in the model  
model.set\_weights(weights)  
# confirm they were stored  
print(model.get\_weights())

Finalmente, aplicaremos el filtro a la imagen de entrada, lo que resultará en un mapa de características que esperamos que muestre la detección de la línea vertical en la imagen de entrada.

# apply filter to input data  
yhat = model.predict(data)

La ​​forma de la salida del mapa de características será de cuatro dimensiones con la forma [batch, rows, columns, filters].

Realizaremos un solo lote y tendremos un solo filtro (un filtro y un canal de entrada), por lo tanto, la forma de salida es [1, ?, ?, 1].

Podemos imprimir el contenido del mapa de características únicas de la siguiente manera:

for r in range(yhat.shape[1]):  
# print each column in the row  
print([yhat[0,r,c,0] for c in range(yhat.shape[2])])

Uniendo todo esto, el ejemplo completo se muestra a continuación.

# example of calculation 2d convolutions  
from numpy import asarray  
from keras.models import Sequential  
from keras.layers import Conv2D  
# define input data  
data = [[0, 0, 0, 1, 1, 0, 0, 0],  
[0, 0, 0, 1, 1, 0, 0, 0],  
[0, 0, 0, 1, 1, 0, 0, 0],  
[0, 0, 0, 1, 1, 0, 0, 0],  
[0, 0, 0, 1, 1, 0, 0, 0],  
[0, 0, 0, 1, 1, 0, 0, 0],  
[0, 0, 0, 1, 1, 0, 0, 0],  
[0, 0, 0, 1, 1, 0, 0, 0]]  
data = asarray(data)  
data = data.reshape(1, 8, 8, 1)  
# create model  
model = Sequential()  
model.add(Conv2D(1, (3,3), input\_shape=(8, 8, 1)))  
# define a vertical line detector  
detector = [[[[0]],[[1]],[[0]]],  
[[[0]],[[1]],[[0]]],  
[[[0]],[[1]],[[0]]]]  
weights = [asarray(detector), asarray([0.0])]  
# store the weights in the model  
model.set\_weights(weights)  
# confirm they were stored  
print(model.get\_weights())  
# apply filter to input data  
yhat = model.predict(data)  
for r in range(yhat.shape[1]):  
# print each column in the row  
print([yhat[0,r,c,0] for c in range(yhat.shape[2])])

Ejecutar el ejemplo primero confirma que el filtro artesanal se definió correctamente en los pesos de capa.

A continuación, se imprime el mapa de características calculado. Podemos ver en la escala de los números que, efectivamente, el filtro ha detectado la línea vertical única con una fuerte activación en el centro del mapa de características.

[array([[[[0.]],  
[[1.]],  
[[0.]]],  
[[[0.]],  
[[1.]],  
[[0.]]],  
[[[0.]],  
[[1.]],  
[[0.]]]], dtype=float32), array([0.], dtype=float32)]

[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]  
[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]  
[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]  
[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]  
[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]  
[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]

Veamos más de cerca lo que se calculó.

Primero, el filtro se aplicó a la esquina superior izquierda de la imagen, o un parche de imagen de 3 × 3 elementos. Técnicamente, el parche de imagen es tridimensional con un solo canal, y el filtro tiene las mismas dimensiones.

No podemos implementar esto en NumPy usando la función [punto ()](https://docs.scipy.org/doc/numpy/reference/generated/numpy.dot.html)en cambio, debemos usar la función [tensordot ()](https://docs.scipy.org/doc/numpy/reference/generated/numpy.tensordot.html)para que podamos sumar adecuadamente todas las dimensiones, por ejemplo:

de numpy import asarray  
de numpy import tensordot  
m1 = asarray ([[0, 1, 0],  
[0, 1, 0],  
[0, 1, 0]])  
m2 = asarray ([[0, 0, 0],  
[0, 0, 0],  
[0, 0, 0]])  
imprimir (tensordot (m1, m2))

Este cálculo da como resultado un solo valor de salida de 0.0, por ejemplo, la característica no se detectó. Esto nos da el primer elemento en la esquina superior izquierda del mapa de características.

Manualmente, esto sería como sigue:

0, 1, 0 0, 0, 0  
0, 1, 0. 0, 0, 0 = 0  
0, 1, 0 0, 0, 0

El filtro se mueve a lo largo de una columna a la izquierda y el proceso se repite. Nuevamente, la característica no se detecta.

0, 1, 0 0, 0, 1  
0, 1, 0. 0, 0, 1 = 0  
0, 1, 0 0, 0, 1

Un movimiento más a la izquierda a la siguiente columna y la función se detecta por primera vez, lo que resulta en una fuerte activación.

0, 1, 0 0, 1 1  
0, 1, 0. 0, 1, 1 = 3  
0, 1, 0 0, 1, 1

Este proceso se repite hasta que el borde del filtro se apoya contra el borde o la columna final de la imagen de entrada. Esto da el último elemento en la primera fila completa del mapa de características.  
[0.0, 0.0, 3.0, 3.0, 0.0, 0.0]  
El filtro se mueve hacia abajo una fila y regresa a la primera columna y el proceso se relaciona de izquierda a derecha para dar la segunda fila del mapa de características.

Y hasta que la parte inferior del filtro se apoye en la parte inferior o en la última fila de la imagen de entrada.

Nuevamente, como en la sección anterior, podemos ver que el mapa de características es una matriz de 6 × 6, más pequeña que la de 8 × 8 imagen de entrada debido a las limitaciones de cómo se puede aplicar el filtro a la imagen de entrada.

**Resumen**

En este tutorial, descubrió cómo hacer las riendas trabajo en la red neuronal convolucional.

Específicamente, aprendiste:

* Convolución Las redes neuronales aplican un filtro a una entrada para crear un mapa de características que resume la presencia de características detectadas en la entrada.
* Los filtros pueden ser hechos a mano, como los detectores de línea, pero la innovación de las redes neuronales convolucionales es aprender los filtros durante el entrenamiento en el contexto de un problema de predicción específico.
* Cómo calcular el mapa de características para capas convolucionales de una y dos dimensiones en una red neuronal convolucional.

¿Tiene alguna pregunta? Haz tus preguntas en los comentarios a continuación y haré todo lo posible para responder.