Convolutional Neural Networks

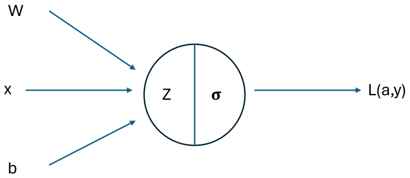
Instructor: Xuechen Zhang

Washington State University Vancouver

Slides adapted from R. Singh@Brown and K. Wong@UTK

Simple NN with cross entropy loss

- (w,x,b) $\rightarrow$ z=wx+b $\rightarrow$ a=$\sigma(z)$ $\rightarrow$ L=loss(a,y)



Simple NN with cross entropy loss

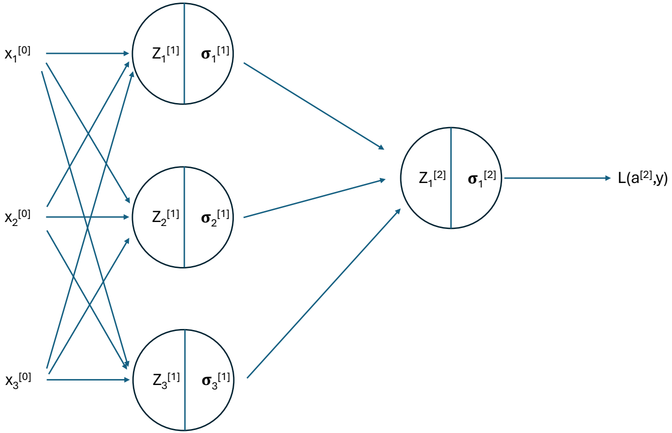
- (w,x,b) $\rightarrow$ z=wx+b $\rightarrow$ a=$\sigma(z)$ $\rightarrow$ L=loss(a,y)
- $z = wx + b$, $\hat{y} = a = \sigma(z)$
- $loss(a, y) = -(ylog(a) + (1 - y)log(1 - a))$
- What is dw (=$\frac{dL}{dw}$) and db(=$\frac{dL}{db}$)?

Simple NN with cross entropy loss

- (w,x,b) $\rightarrow$ z=wx+b $\rightarrow$ a=$\sigma(z)$ $\rightarrow$ L=loss(a,y)
- $z = wx + b$, $\hat{y} = a = \sigma(z)$
- $loss(a, y) = -(ylog(a) + (1 - y)log(1 - a))$
- What is dw (=$\frac{dL}{dw}$) and db(=$\frac{dL}{db}$)?
- $da = \frac{dL}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$
- $dz = \frac{dL}{dz} = \frac{dL}{da} * \frac{da}{dz} = da * \frac{da}{dz}$ where $\frac{da}{dz} = a(1 - a)$, thus, dz = a - y
- $dw = \frac{dL}{da} * \frac{da}{dz} * \frac{dz}{dw}$ => dw=x * dz = x(a-y).
- Similarly, db = dz.

Simple NN with one hidden layer

- $(W^{[1]}, x, b^{[1]}) \rightarrow z^{[1]} = W^{[1]}x + b^{[1]} \rightarrow a^{[1]} = \sigma(z^{[1]}) \rightarrow z^{[2]} = W^{[2]}x + b^{[2]} \rightarrow a^{[2]} = \sigma(z^{[2]}) \rightarrow L = loss(a^{[2]}, y)$



Understanding the dimensions/shapes

- Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$
- $N_x = n^{[0]}, n^{[1]}, n^{[2]}$
- Shapes of parameters: $(n^{[1]}, n^{[0]})$, $(n^{[1]}, 1)$, $(n^{[2]}, n^{[1]})$, $(b^{[2]}, 1)$.
- Cost function: $J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{n} L(\hat{y}, y)$
- $dw^{[1]} = \frac{dJ}{dw^{[1]}}$, $db^{[1]} = \frac{dJ}{db^{[1]}}$, ...
- $w^{[1]} = w^{[1]} - \alpha * dw^{[1]}$, $b^{[1]} = b^{[1]} - \alpha * db^{[1]}$, ...

Formulas for forward propogation

- $Z^{[1]} = W^{[1]}X + b^{[1]}$
- $A^{[1]} = g^{[1]}(Z^{[1]})$
- $Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$
- $A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$

Gradient descent for Layer 2

- $dz^{[2]} = \frac{dL}{dz^{[2]}} = a^{[2]} - y$
- $dw^{[2]} = \frac{dL}{dw^{[2]}} = \frac{dL}{da^{[2]}} * \frac{da^{[2]}}{dz^{[2]}} * \frac{dz^{[2]}}{dw^{[2]}} = dz^{[2]} {a^{[1]}}^{T}$
- $db^{[2]} = dz^{[2]}$

Gradient descent for Layer 1

- $dz^{[1]} = \frac{dL}{dz^{[1]}} = \frac{dL}{dz^{[2]}} \frac{dz^{[2]}}{da^{[1]}} \frac{da^{[1]}}{dz^{[1]}} = dz^{[2]} w^{[2]} g^{[1]'}(Z^{[1]})$
- $dW^{[1]} = \frac{dL}{dW^{[1]}} == \frac{dL}{dz^{[2]}} \frac{dz^{[2]}}{da^{[1]}} \frac{da^{[1]}}{dz^{[1]}} \frac{dz^{[1]}}{dW^{[1]}} = dz^{[1]} x^T$
- $db^{[1]} = dz^{[1]}$

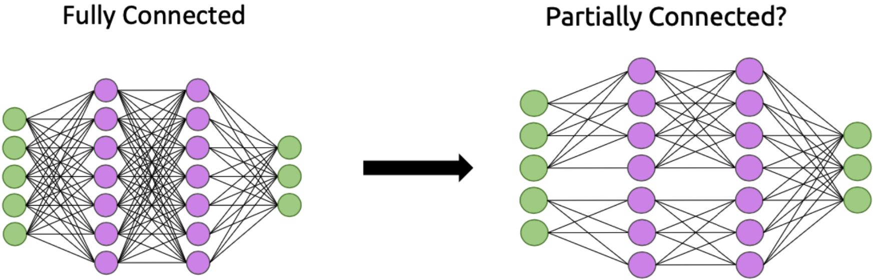
Formulas for backward propogation

- $dZ^{[2]} = A^{[2]} - Y$
- $dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]^T}$
- $db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$
- $dZ^{[1]} = W^{[2]^T} dZ^{[2]} g^{[1]'}(Z^{[1]})$
- $dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$
- $db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$

Today’s goal – learn about convolution

- **Intro to CNNs – Convolution**
- Convolution - Stride
- Learning in CNNs

Does a network have to be fully connected?



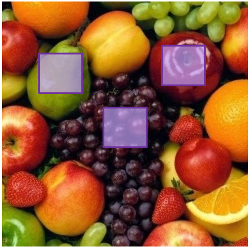
Why would you ever want to do this?

Advantages of Partial Connections

- Fewer connections -> fewer weights to learn
  - Faster training; more compact models; better generalization performance
- Can design connectivity pattern that exploits knowledge of the data (like connecting patterns in features)

When partially connected networks are useful

- **Observation**: Nearby pixels are more likely to be related
- **Assumption**: It is okay to only connect the nearby pixels
- Focusing on local patterns = partial connections
- How?



The Main Building Block: Convolution

- Convolution is an operation that takes two inputs:

(1) An image (2D – B/W)

(2) A filter (also called a kernel)

1	1	1
0	0	0
-1	-1	-1

2D array of numbers; could be any values

What Convolution Does (Visually)

image

2	0	1	3
7	1	1	0
0	2	5	0
0	5	1	4

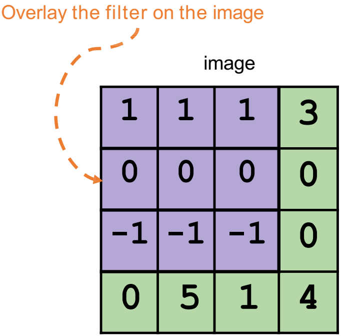
$\otimes$

filter/kernel

1	1	1
0	0	0
-1	-1	-1

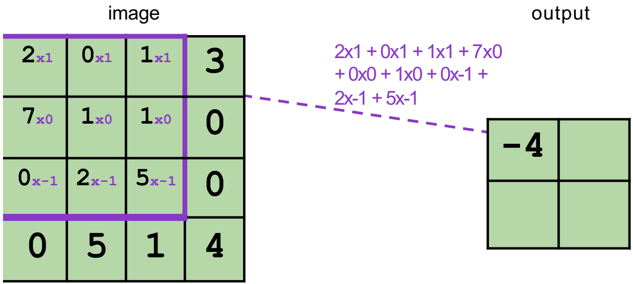
(We use this symbol for convolution)
(The verb form is "convolve")

What Convolution Does (Visually)



What Convolution Does (Visually)

Sum up multiplied values to produce output value



What Convolution Does (Visually)

Move the filter over by one pixel

image

1	1	1	3
0	0	0	0
-1	-1	-1	0
0	5	1	4

output

-4	

What Convolution Does (Visually)

Move the filter over by one pixel

image

2	1	1	1
7	0	0	0
0	-1	-1	-1
0	5	1	4

output

-4	

What Convolution Does (Visually)

Repeat (multiply, sum up)

image

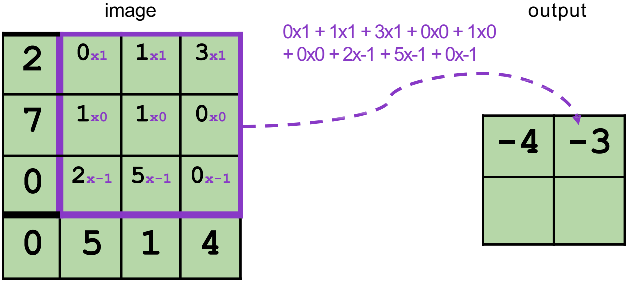
2	0 x1	1 x1	3 x1
7	1 x0	1 x0	0 x0
0	2 x-1	5 x-1	0 x-1
0	5	1	4

output

-4	

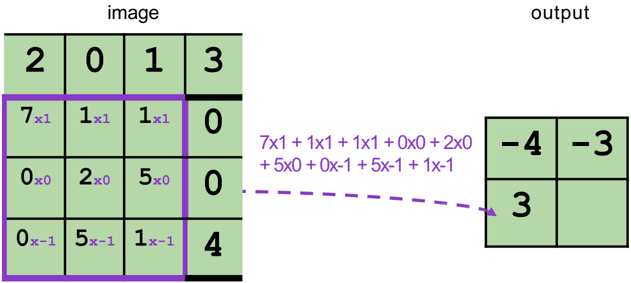
What Convolution Does (Visually)

Repeat (multiply, sum up)



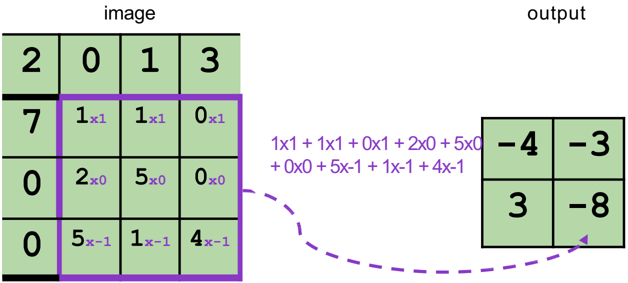
What Convolution Does (Visually)

Repeat...



What Convolution Does (Visually)

Repeat...



What Convolution Does (Visually)

In summary:

$$\underset{\text{image}}{\begin{bmatrix} 2 & 0 & 1 & 3 \\ 7 & 1 & 1 & 0 \\ 0 & 2 & 5 & 0 \\ 0 & 5 & 1 & 4 \end{bmatrix}} \otimes \underset{\text{filter/kernel}}{\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}} = \underset{\text{output}}{\begin{bmatrix} -4 & -3 \\ 3 & -8 \end{bmatrix}}$$

Try it out yourself!

Convolve this image

2	0	3	1
1	1	0	0
1	0	2	0
1	0	1	2

$\otimes$

With this filter

1	0	-1
2	0	-2
1	0	-1

What Convolution Does (Mathematically)

$$V(x,y) = (\mathrm{I} \otimes K)(x,y) = \sum_m \sum_n \underbrace{I(x+m, y+n)K(m,n)}_{\text{Multiply kernel value with corresponding image pixel value}}$$

The output at pixel (x,y)

"Image I convolved with kernel K"

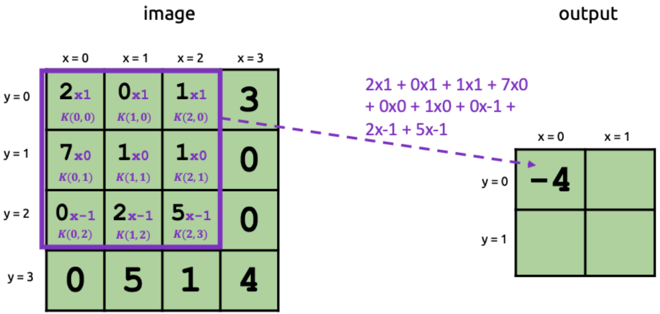
Sum over kernel rows

Sum over kernel columns

What Convolution Does (Mathematically)

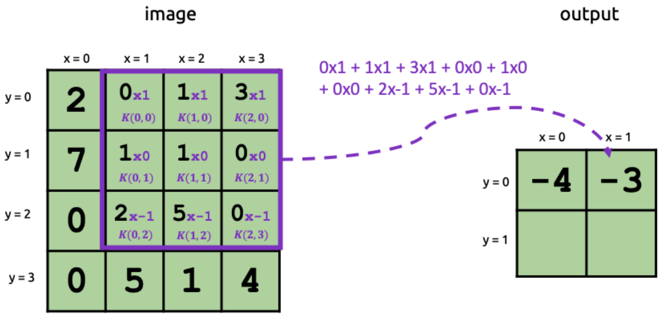
$$\overset{\text{image}}{\begin{array}{c|cccc} & x=0 & x=1 & x=2 & x=3 \\ \hline y=0 & 2 & 0 & 1 & 3 \\ y=1 & 7 & 1 & 1 & 0 \\ y=2 & 0 & 2 & 5 & 0 \\ y=3 & 0 & 5 & 1 & 4 \end{array}} \otimes \overset{\text{filter/kernel}}{\begin{array}{c|ccc} & m=0 & m=1 & m=2 \\ \hline n=0 & 1 & 1 & 1 \\ n=1 & 0 & 0 & 0 \\ n=2 & -1 & -1 & -1 \end{array}} = \overset{\text{output}}{\begin{array}{c|cc} & x=0 & x=1 \\ \hline y=0 & -4 & -3 \\ y=1 & 3 & -8 \end{array}}$$

What Convolution Does (Mathematically)



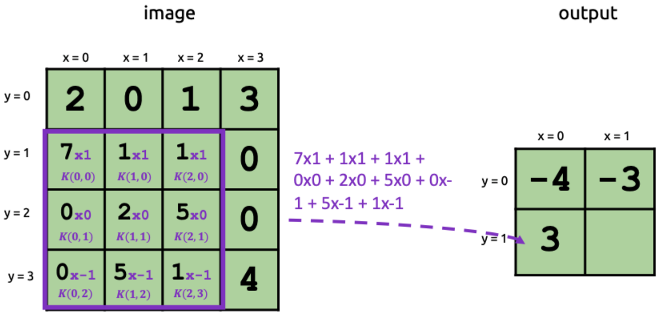
$$V(0,0) = (\mathrm{I} \otimes K)(0,0) = \sum_{m=0}^{2} \sum_{n=0}^{2} I(0+m, 0+n) K(m,n)$$

What Convolution Does (Mathematically)



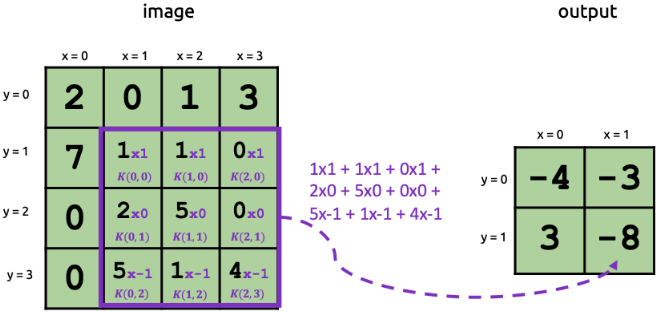
$$V(1,0) = (\mathrm{I}\otimes K)(1,0) = \sum_{m=0}^{2}\sum_{n=0}^{2} I(1+m, 0+n)K(m,n)$$

What Convolution Does (Mathematically)



$$V(0,1) = (\mathrm{I} \otimes K)(0,1) = \sum_{m=0}^{2} \sum_{n=0}^{2} I(0+m, 1+n)K(m,n)$$

What Convolution Does (Mathematically)



$$V(1,1) = (\mathrm{I} \otimes K)(1,1) = \sum_{m=0}^{2} \sum_{n=0}^{2} I(1+m, 1+n) K(m,n)$$

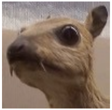
What Convolution Does (In Code)

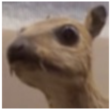
```
// Input: Image I, Kernel K, Output V, pixel index x,y
// Assumes K is 3x3
function apply_kernel(I, K, V, x, y)
   for m = 0 to 2:
      for n = 0 to 2:
         V(x,y) += K(m,n) * I(m+x, n+y)
```

Different filters = different effects

Blur

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9





Edge Detection / Outline Kernel

0	-1	0
-1	5	-1
0	-1	0

Shift

0	0	0
1	0	0
0	0	0

*

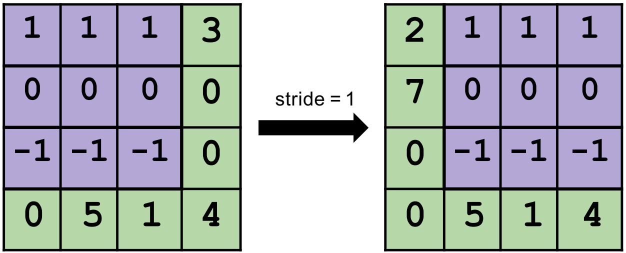
* exaggerated

Today's goal – learn about convolution

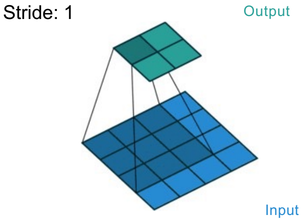
- Intro to CNNs – Convolution
- **Convolution - Stride**
- Learning in CNNs

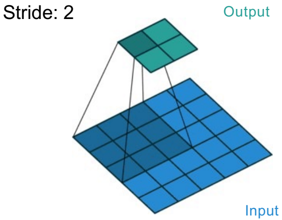
Stride

- We don't just have to slide the filter by one pixel every time
- The distance we slide a filter by is called stride
  - All the examples we have seen have been stride = 1



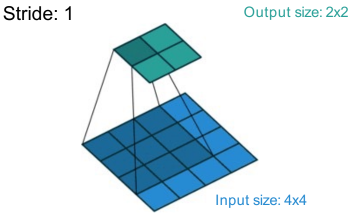
Stride in Action

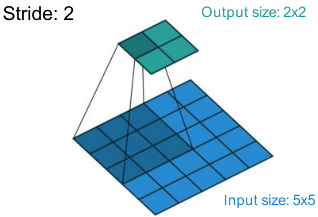




Why would we want stride > 1?

- Any connection between input and output size?
  - Larger stride turns a bigger input into the same size output
  - Larger stride turns the same size input into a smaller output
  - Use this to (controllably) descrese image resolution!





Today's goal – learn about convolution

- Intro to CNNs – Convolution
- Convolution - Stride
- **Learning in CNNs**

Key Idea 1: Filters are Learnable

image

$$\begin{array}{c|cccc} & x=0 & x=1 & x=2 & x=3 \\ \hline y=0 & 2 & 0 & 1 & 3 \\ y=1 & 7 & 1 & 1 & 0 \\ y=2 & 0 & 2 & 5 & 0 \\ y=3 & 0 & 5 & 1 & 4 \end{array} \otimes$$

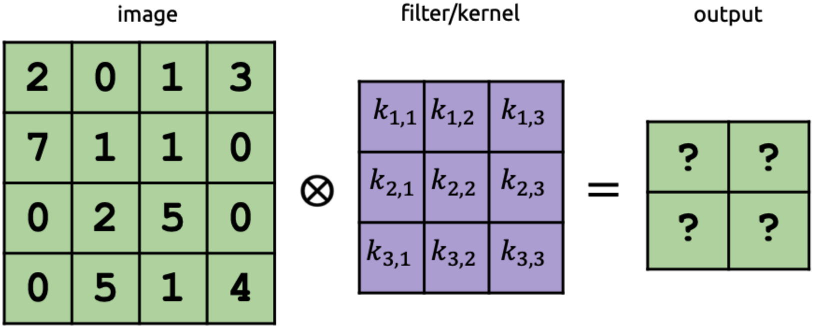
filter/kernel

$$\begin{array}{c|ccc} & m=0 & m=1 & m=2 \\ \hline n=0 & 1 & 1 & 1 \\ n=1 & 0 & 0 & 0 \\ n=2 & -1 & -1 & -1 \end{array} =$$

output

$$\begin{array}{c|cc} & x=0 & x=1 \\ \hline y=0 & -4 & -3 \\ y=1 & 3 & -8 \end{array}$$

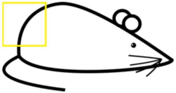
Key Idea 1: Filters are Learnable



$k_{i,j}$ are learnable parameters

Detecting patterns using learned filters

Original image

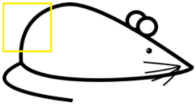


Visualization of the filter on the image

Label="Mouse"

Detecting patterns using learned filters

Original image



Visualization of the filter on the image

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Visualization of a curve detector filter

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

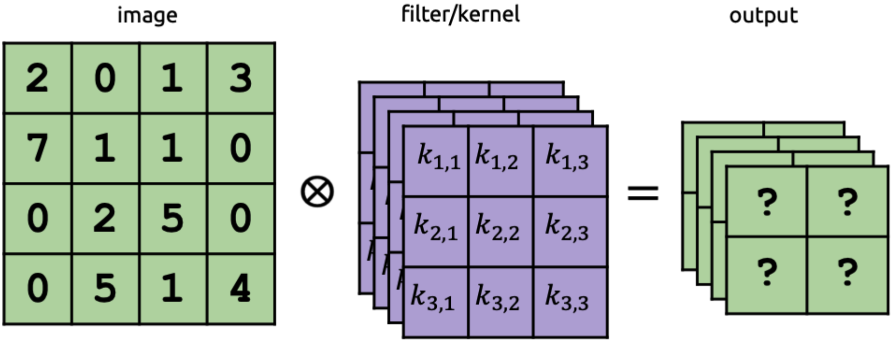
$*$

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

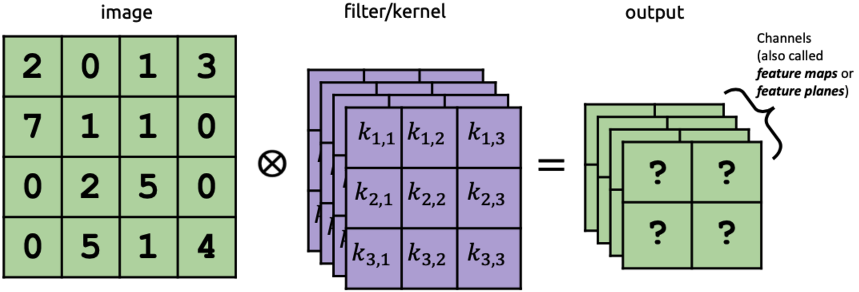
Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Key Idea 2: Learn many filters



This block of filters is called a ***filter bank***

Key Idea 2: Learn many filters



The output is now a multi-channel image

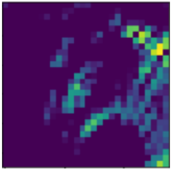
Key Idea 2: Learn many filters

- Why are multiple filters a good idea?

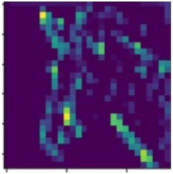
Key Idea 2: Learn many filters

- Why are multiple filters a good idea?
- Can learn to extract different features of the image

Input image

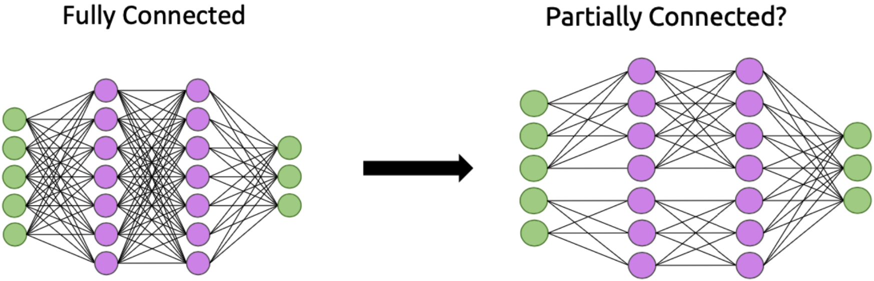


Output of filter 1



Output of filter 2

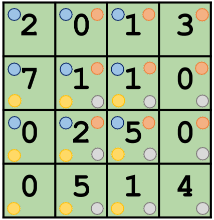
How is convolution “partially connected”?



Why would you ever want to do this?

Only certain input pixels are "connected" to certain output pixels

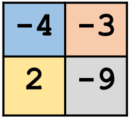
image



Colored dots in the input pixels represent which output pixels that input pixel contributes to

If this were fully connected, every input pixel would have all four output colors

output



Convolution in Tensorflow

```
tf.nn.conv2d(input, filter, strides, padding)
```

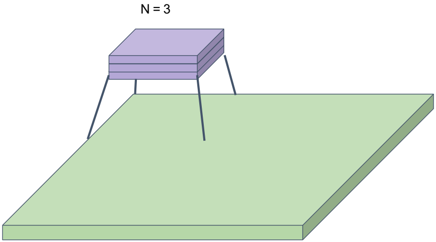
Can you guess the shape?

Input Image (4-D Tensor)
Shape:

[batchSz, input_height, input_width, input_channels]

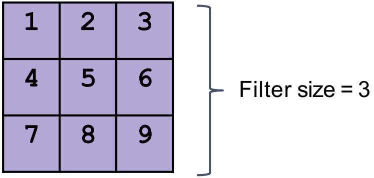
Output Size of a Convolution Layer

- The output size of a convolution layer depends on 4 Hyperparameters:
  - Number of filters, N



Output Size of a Convolution Layer

- The output size of a convolution layer depends on 4 Hyperparameters:
  - Number of filters, N
  - The size of these filters, F



Output Size of a Convolution Layer

- The output size of a convolution layer depends on 4 Hyperparameters:
  - Number of filters, N
  - The size of these filters, F
  - The stride, S

2	0	3	1	0
2	4	5	2	3
0	0	3	3	1
2	9	9	7	8
3	4	7	2	1

2	0	3	1	0
2	4	5	2	3
0	0	3	3	1
2	9	9	7	8
3	4	7	2	1

Convolution in Tensorflow

```
tf.nn.conv2d(input, filter, strides, padding)
```

List of ints of length 4

Represents the strides along each dimension of the input

[batch_stride, stride_along_height, stride_along_width, stride_along_input_channels]

Convolution in Tensorflow

```
tf.nn.conv2d(input, filter, strides, padding)
```

?

“Problem” With Convolution

- Output of convolution is always smaller than the input
- Why might we want the output size to be the same?
  - To avoid the filter “eating at the border” of the image when applying multiple conv layers

In summary:

$$\overset{\text{image}}{\begin{bmatrix} 2 & 0 & 1 & 3 \\ 7 & 1 & 1 & 0 \\ 0 & 2 & 5 & 0 \\ 0 & 5 & 1 & 4 \end{bmatrix}} \otimes \overset{\text{filter/kernel}}{\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}} = \overset{\text{output}}{\begin{bmatrix} -4 & -3 \\ 3 & -8 \end{bmatrix}}$$

Solution: Padding

- Apply the kernel to ‘imaginary’ pixels surrounding the image

?	?	?	?	?	?	?
?	2	0	3	1	1	?
?	1	1	0	0	2	?
?	4	3	2	0	1	?
?	1	0	5	2	0	?
?	0	1	0	3	0	?
?	?	?	?	?	?	?

What Values to Use For These Pixels?

- Standard practice: fill with zeroes

0	0	0	0	0	0	0
0	2	0	3	1	1	0
0	1	1	0	0	2	0
0	4	3	2	0	1	0
0	1	0	5	2	0	0
0	0	1	0	3	0	0
0	0	0	0	0	0	0

Padding Modes in Tensorflow

- 2 available options: ‘VALID’ and ‘SAME’:

Valid

Filter only slides over “Valid” regions of the data

2	0	1	3
0	1	1	0
0	0	2	0
0	1	1	1

Same

Filter slides over the bounds of the data, ensuring output size is the “Same” as input size (when stride = 1)

0	0	0	0	0	0
0	2	0	1	3	0
0	1	1	2	3	0
0	4	3	2	1	0
0	8	3	1	3	0
0	0	0	0	0	0

VALID Padding in Tensorflow

$$\begin{bmatrix} 2 & 0 & 3 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 \\ 1 & 0 & 1 & 2 \end{bmatrix} \underset{\substack{\text{“VALID”} \\ \text{Stride = 1}}}{\otimes} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -1 \end{bmatrix}$$

SAME Padding in Tensorflow

```
tf.nn.conv2d(input, filter, strides,
         padding='SAME')
```

0	0	0	0	0	0
0	2	0	1	3	0
0	1	1	2	3	0
0	4	3	2	1	0
0	8	3	1	3	0
0	0	0	0	0	0

SAME Padding in Tensorflow

```
tf.nn.conv2d(input, filter, strides,
padding='SAME')
```

0	0	0	0	0	0
0	2	0	1	3	0
0	1	1	2	3	0
0	4	3	2	1	0
0	8	3	1	3	0
0	0	0	0	0	0

SAME Padding in Tensorflow

```
tf.nn.conv2d(input, filter, strides,
             padding='SAME')
```

0	0	0	0	0	0
0	2	0	1	3	0
0	1	1	2	3	0
0	4	3	2	1	0
0	8	3	1	3	0
0	0	0	0	0	0

SAME Padding in Tensorflow

```
tf.nn.conv2d(input, filter, strides,
        padding='SAME')
```

0	0	0	0	0	0
0	2	0	1	3	0
0	1	1	2	3	0
0	4	3	2	1	0
0	8	3	1	3	0
0	0	0	0	0	0

SAME Padding example

$$
\begin{bmatrix} 2 & 0 & 3 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 \\ 1 & 0 & 1 & 2 \end{bmatrix}
\underset{\substack{\text{“Same”} \\ \text{Stride = 1}}}{\otimes}
\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}
=
\begin{bmatrix} -1 & -1 & -1 & 6 \\ -2 & 0 & 1 & 5 \\ -1 & -1 & -1 & 5 \\ 0 & -1 & -4 & 4 \end{bmatrix}
$$

Output Size of a Convolution Layer

- The output size of a convolution layer depends on 4 Hyperparameters:
  - Number of filters, N
  - The size of these filters, F
  - The stride, S
  - The amount of padding, P

0	0	0	0	0	0
0	0	0	0	0	0
0	0	2	3	0	0
0	0	9	2	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Padding = 2

Output Size of a Convolution Layer

- Suppose we know the number of filters, their size, the stride, and padding (n,f,s,p).
- Then for a convolution layer with input dimension w * h * d, the output dimensions w’ * h’ * d’ are:
  - $w' = \frac{w-f+2p}{s} + 1$
  - $h' = \frac{h-f+2p}{s} + 1$
  - $d' = n$

Output Size for “VALID” Padding

- (n,f,s,p) = (1, 3, 1, 0)
- If w=4, w’=2.

Output Size for “SAME” Padding

- (n,f,s,p) = (1, 3, 1, 1)
- If w=4, w’=4.
- We choose p=1 so output size is the same.

Convolution in Tensorflow

```
tf.nn.conv2d(input, filter, strides, padding)
```

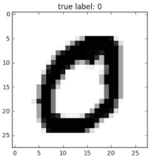
Input Image
(4-D Tensor)

Filter/Kernel
(4-D Tensor)

Strides along
each dimension

Type of Padding
(String “Valid” or
“Same”)

Application to Real World Data (MNIST)




```
# Should be of shape (batch_sz, 28, 28, 1) for MNIST
inputs = MNIST_image_batch

# Sets up a 5x5 filter with 1 input channels and 16 output channels
self.filter = tf.Variable(tf.random.normal([5, 5, 1, 16], stddev=0.1))

# Convolves the input batch with our defined filter
conv = tf.nn.conv2d(inputs, self.filter, [1, 2, 2, 1], padding="SAME")
```

Application to Real World Data (CIFAR)

```
# Should be of shape (batch_sz, 32, 32, 3) for CIFAR10
inputs = CIFAR_image_batch

# Sets up a 5x5 filter with 3 input channels and 16 output channels
self.filter = tf.Variable(tf.random.normal([5, 5, 3, 16], stddev=0.1))

# Convolves the input batch with our defined filter
conv = tf.nn.conv2d(inputs, self.filter, [1, 2, 2, 1], padding="SAME")
```