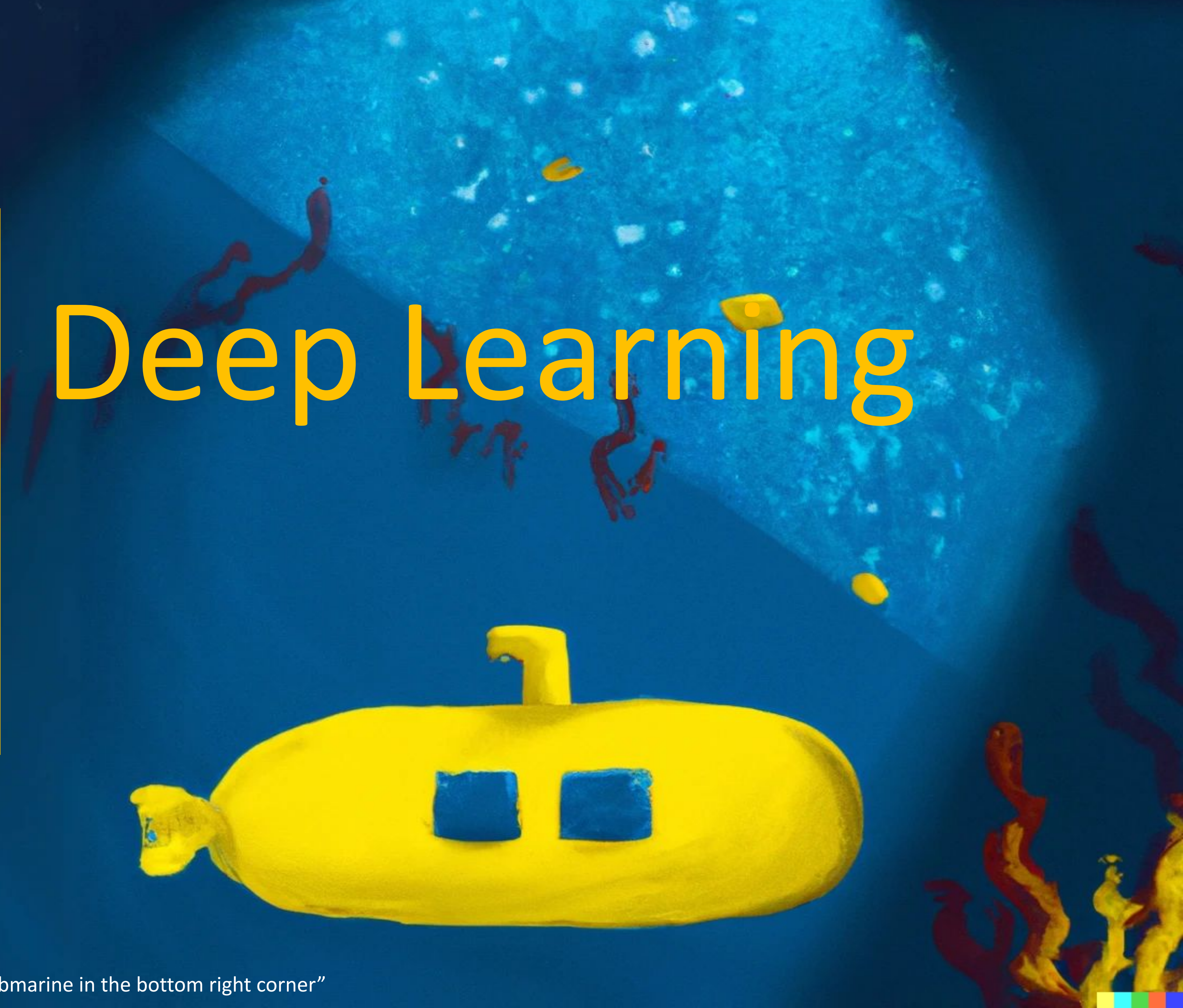
CSCI 1470/2470
Spring 2023

Deep Learning

Ritambhara Singh

February 22, 2023
Wednesday

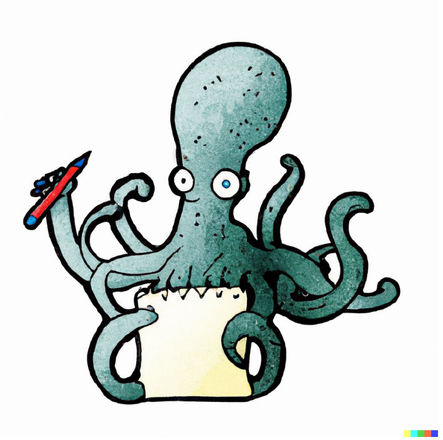


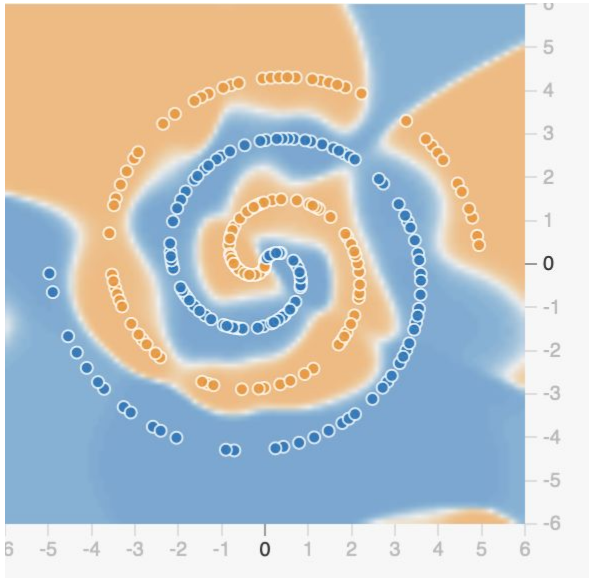
DALL-E 2 prompt “a painting of deep underwater with a yellow submarine in the bottom right corner”

Recap

Building multi-layer neural networks

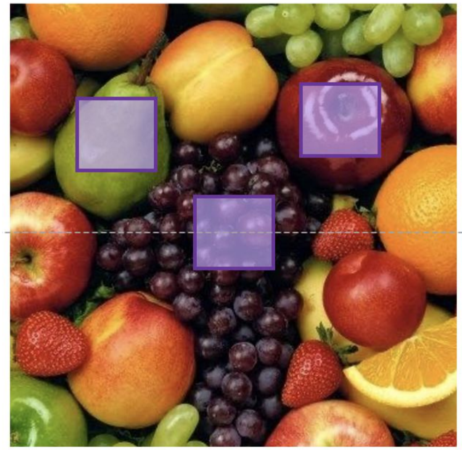
- Hidden layers
- What a one-hidden layer network can learn
- What a multi-layer network can learn





Introduction to CNNs

- Partially connected networks are useful (e.g., for images!)
- Fully connected networks are not transitionally invariant
- Convolutional filter



Today's goal – continue to learn about CNNs

(1) Convolution (contd.) – stride

(2) Learning covolutional filters – connection to partially connected networks

(3) Convolution in Tensorflow – padding and other considerations

What Convolution Does (Visually)

In summary:

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

What Convolution Does (Mathematically)

$$V(x,y) = (\mathrm{I} \otimes K)(x,y) = \sum_m \sum_n I(x+m, y+n)K(m,n)$$

The output at pixel (x,y)

"Image I convolved with kernel K"

Sum over kernel columns

Sum over kernel rows

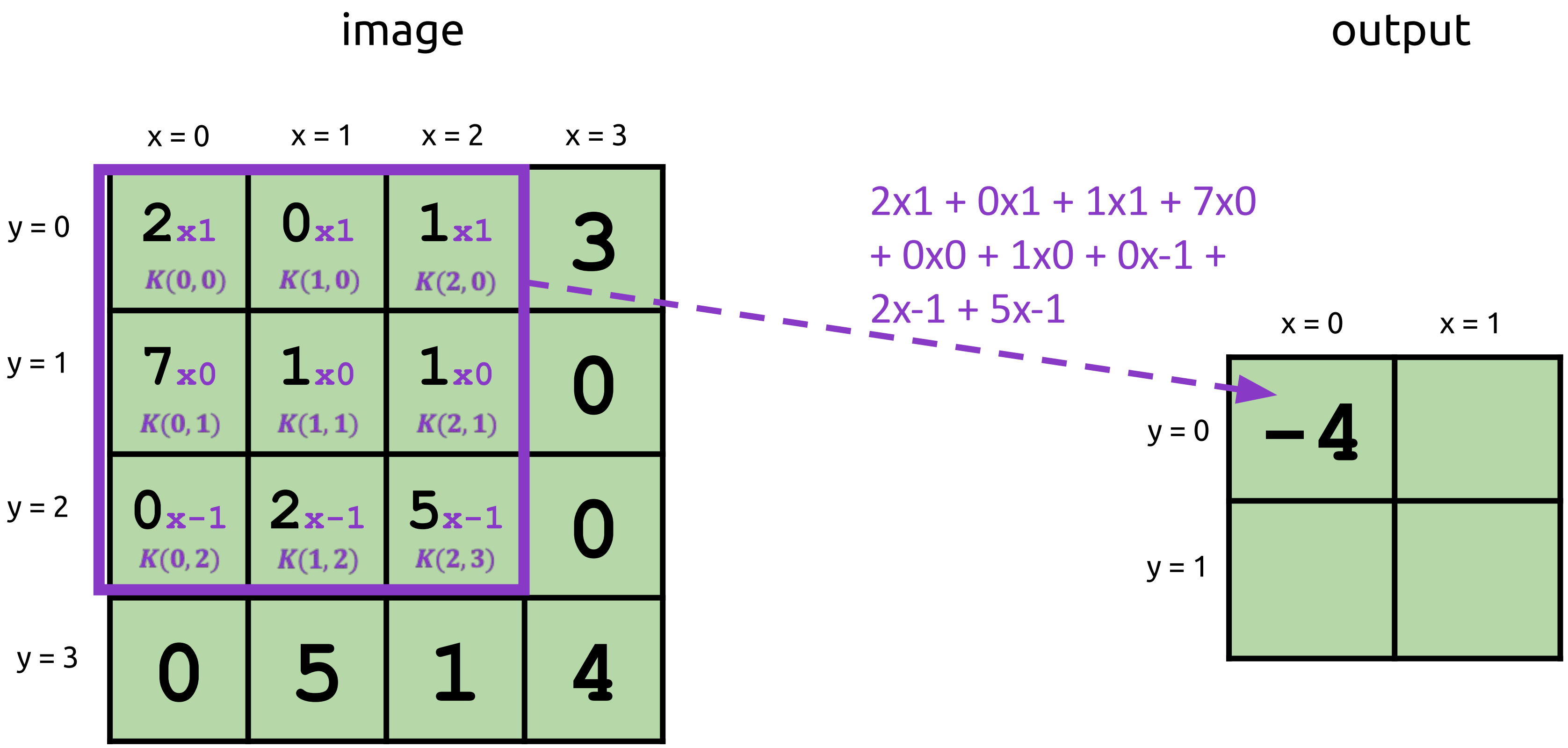
Multiply kernel value with corresponding image pixel value

What Convolution Does (Mathematically)

image | filter/kernel | output

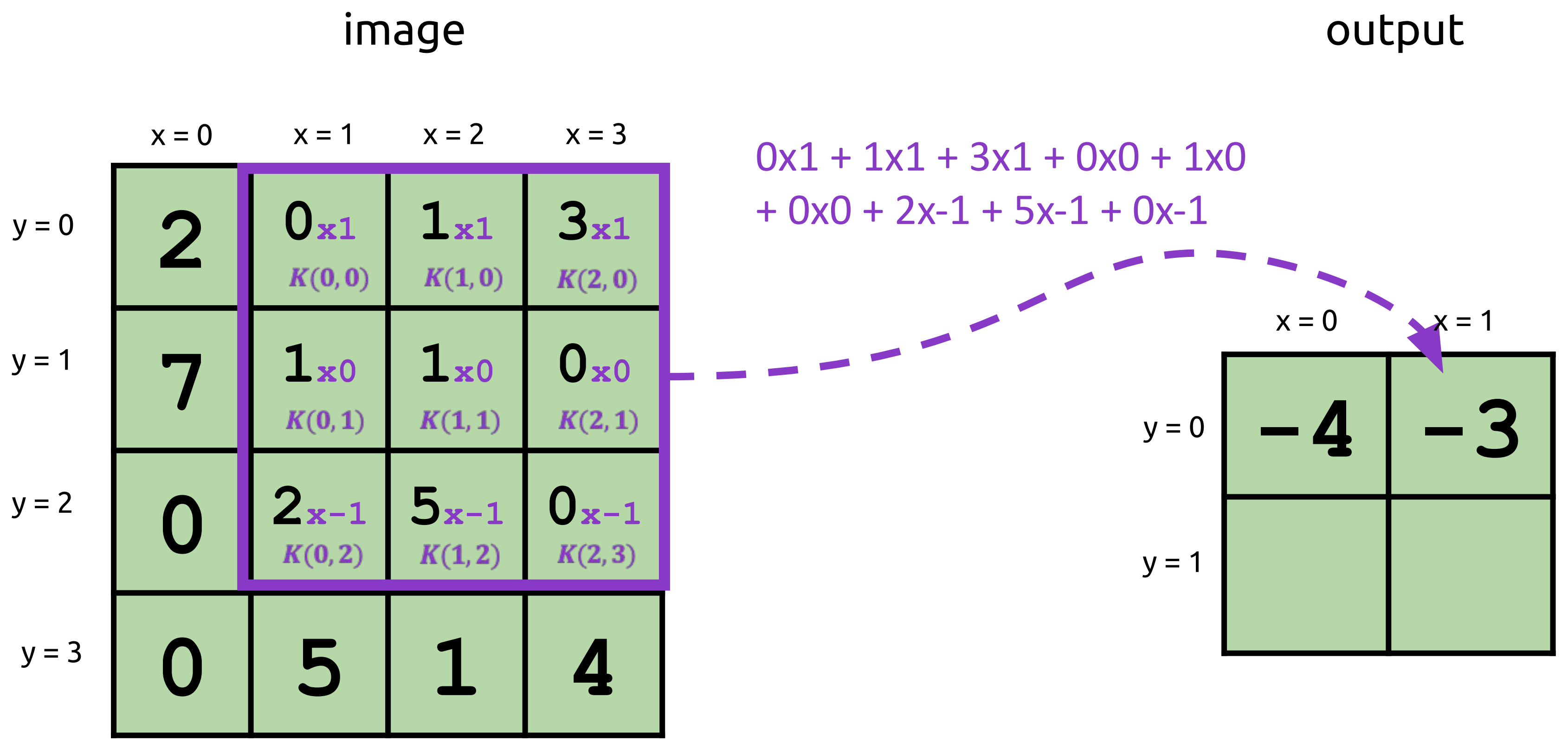
$$
\begin{array}{cc}
 & \begin{array}{cccc} x=0 & x=1 & x=2 & x=3 \end{array} \\
\begin{array}{c} y=0 \\ y=1 \\ y=2 \\ y=3 \end{array} & \begin{bmatrix} 2 & 0 & 1 & 3 \\ 7 & 1 & 1 & 0 \\ 0 & 2 & 5 & 0 \\ 0 & 5 & 1 & 4 \end{bmatrix}
\end{array}
\otimes
\begin{array}{cc}
 & \begin{array}{ccc} m=0 & m=1 & m=2 \end{array} \\
\begin{array}{c} n=0 \\ n=1 \\ n=2 \end{array} & \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}
\end{array}
=
\begin{array}{cc}
 & \begin{array}{cc} x=0 & x=1 \end{array} \\
\begin{array}{c} y=0 \\ y=1 \end{array} & \begin{bmatrix} -4 & -3 \\ 3 & -8 \end{bmatrix}
\end{array}
$$

What Convolution Does (Mathematically)



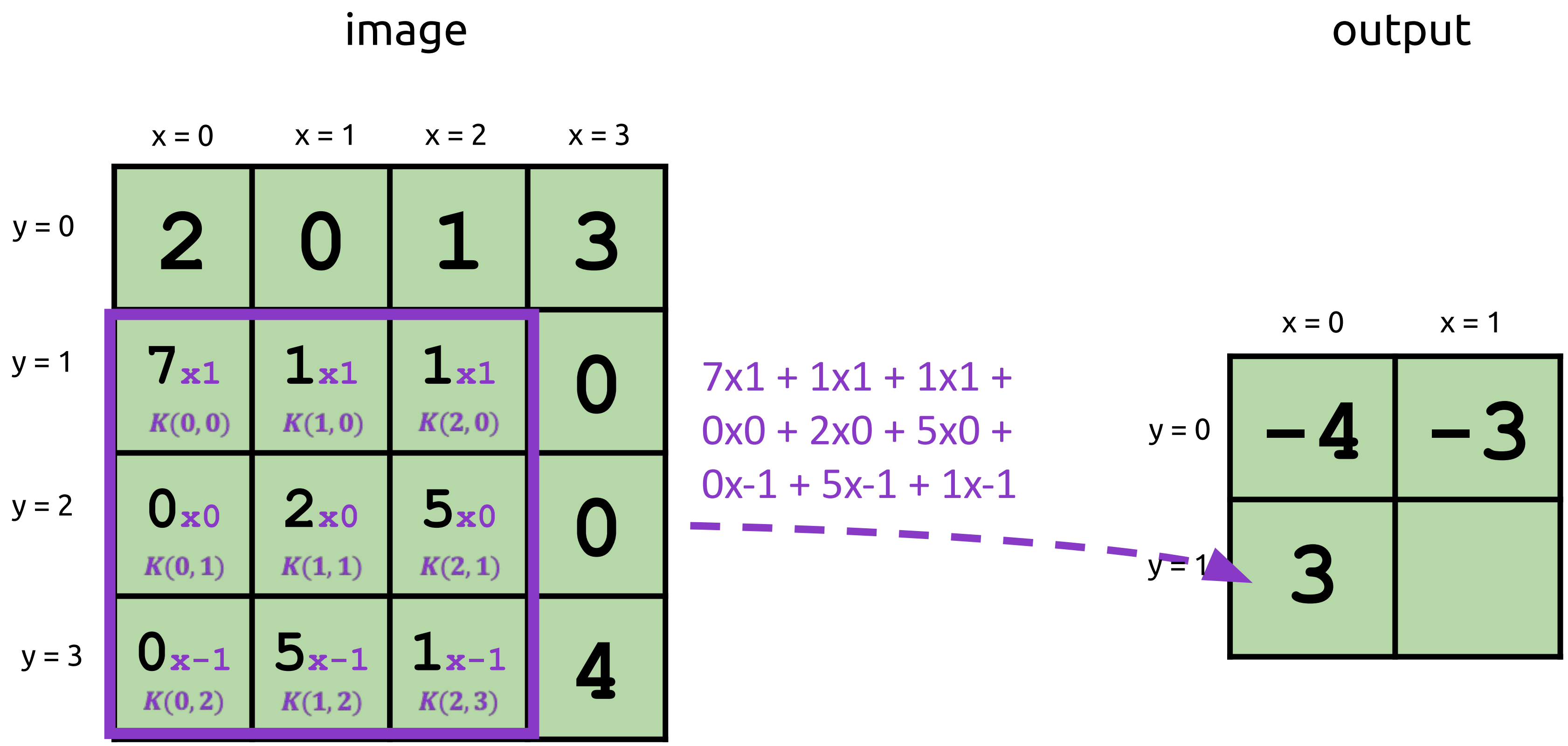
$$V(0,0) = (\mathrm{I} \otimes K)(0,0) = \sum_{m=0}^{2} \sum_{n=0}^{2} I(0+m, 0+n) K(m,n)$$

What Convolution Does (Mathematically)



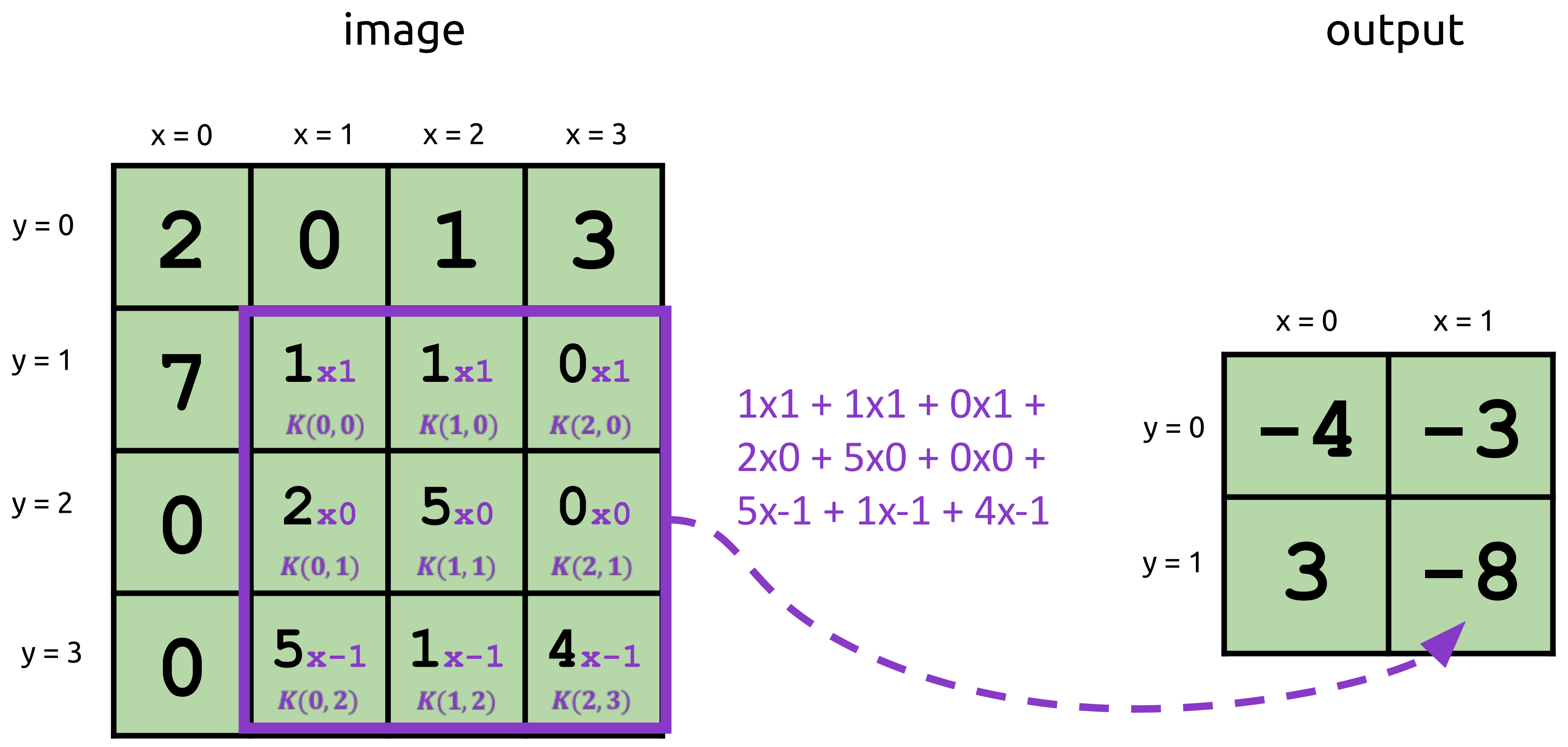
$$V(1,0) = (\mathrm{I} \otimes K)(1,0) = \sum_{m=0}^{2} \sum_{n=0}^{2} I(1+m, 0+n) K(m,n)$$

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)
```

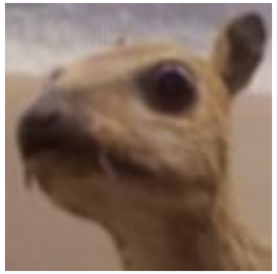
$$Equation\text{: } V(x,y) = (\mathrm{I} \otimes K)(x,y) = \sum_m \sum_n I(x+m, y+n) K(m,n)$$

Different filters = different effects

https://setosa.io/ev/image-kernels/

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

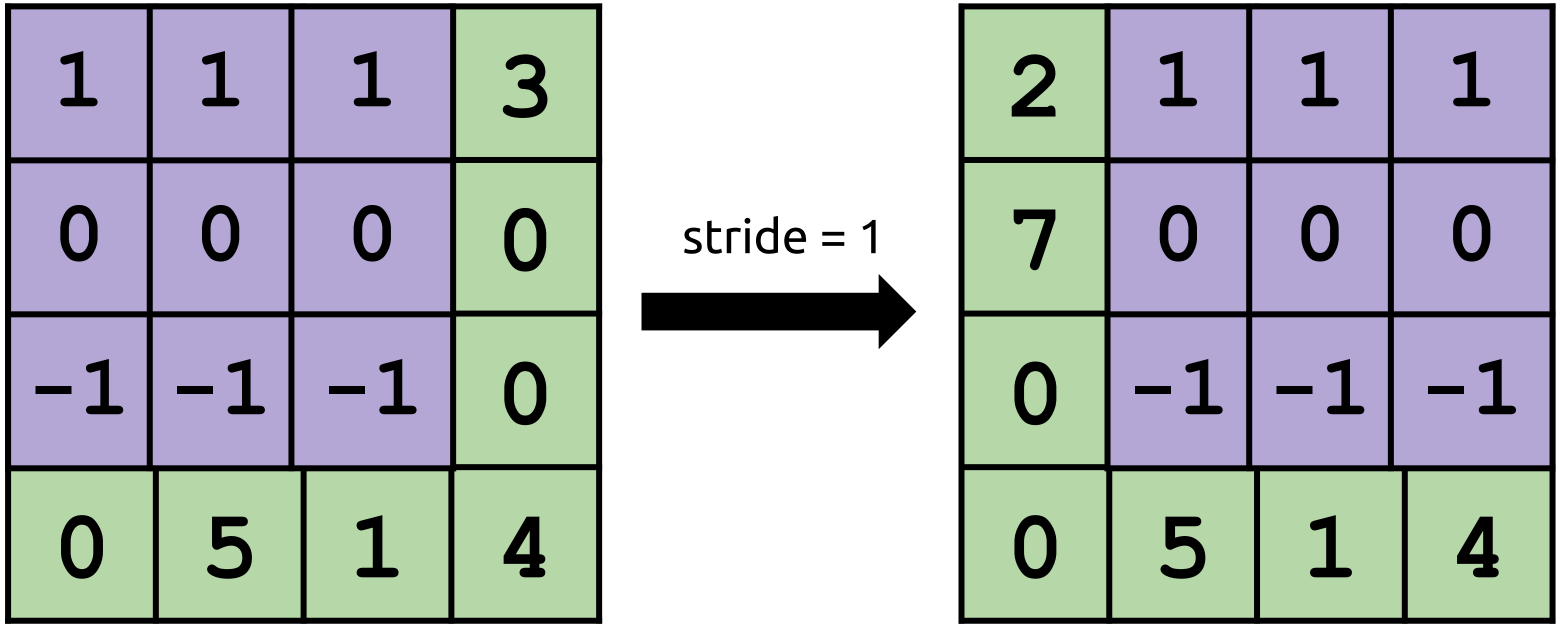
0	0	0
1	0	0
0	0	0

*

* exaggerated

Stride

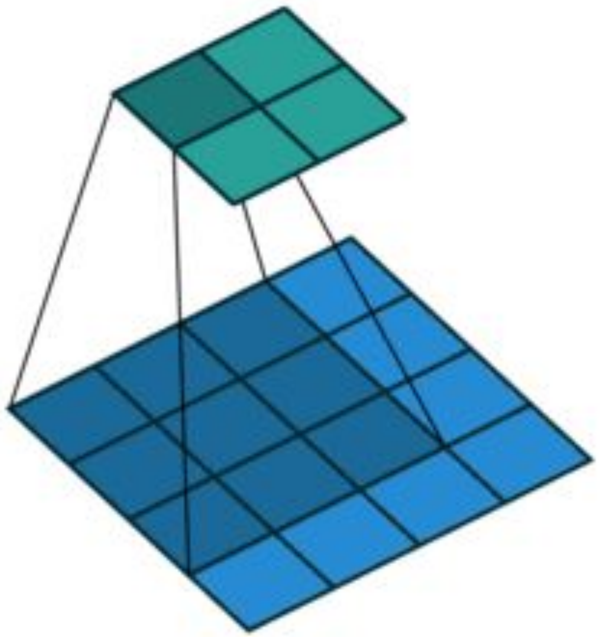
- We don’t just have to slide the filter by one pixel every time
- The distance we slide a filter by is called ***stride***
  - All the examples we’ve seen thus far have been stride = 1



Stride in Action

Stride: 1

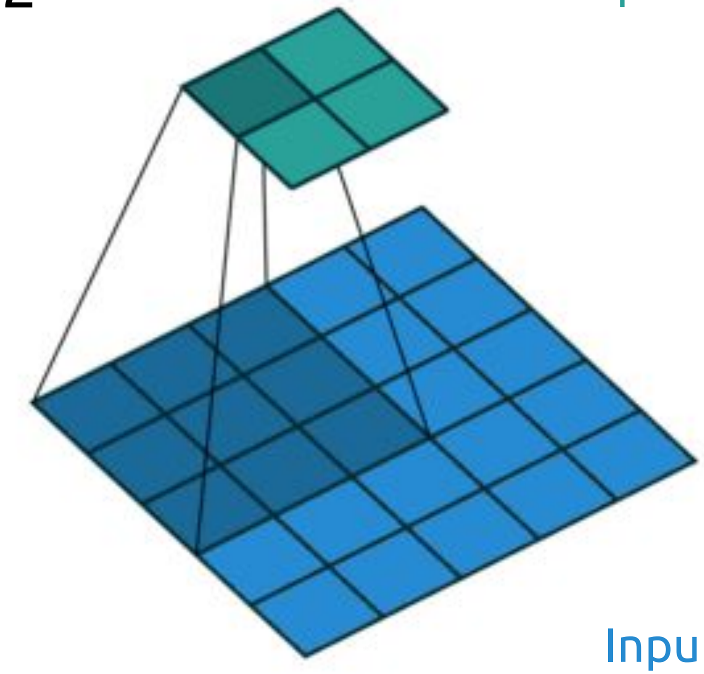
Output



Input

Stride: 2

Output

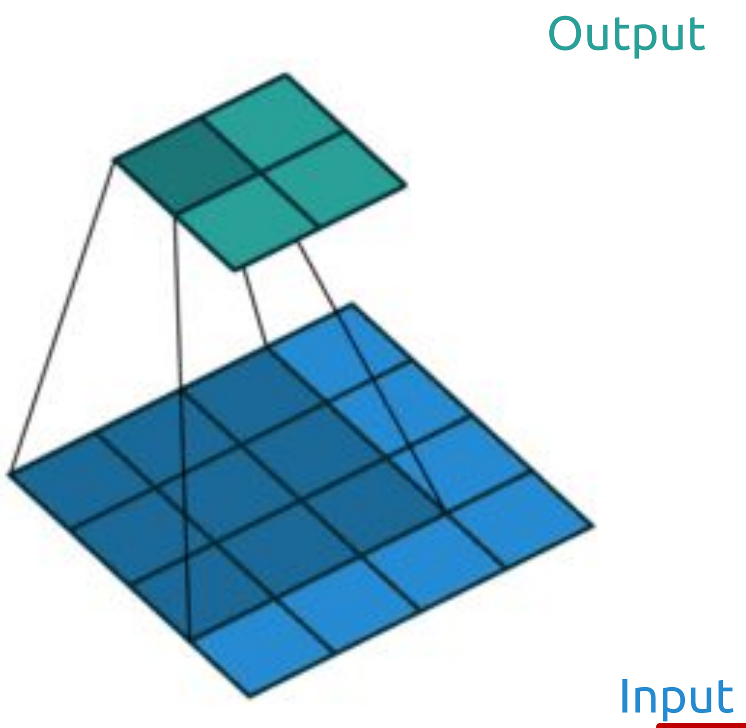


Input

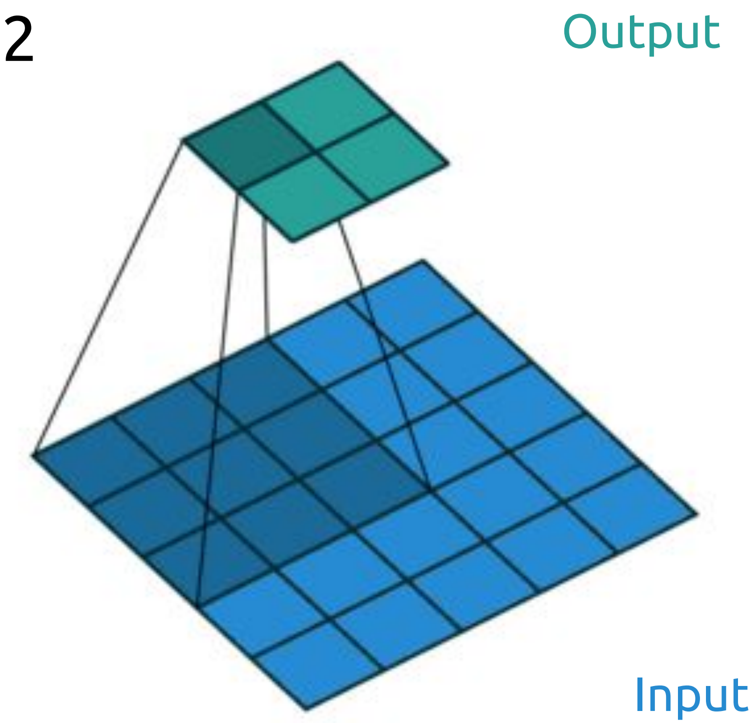
http://deeplearning.net/software/theano_versions/0.9.X/tutorial/conv_arithmetic.html

Why would we want stride > 1?

Stride: 1



Stride: 2

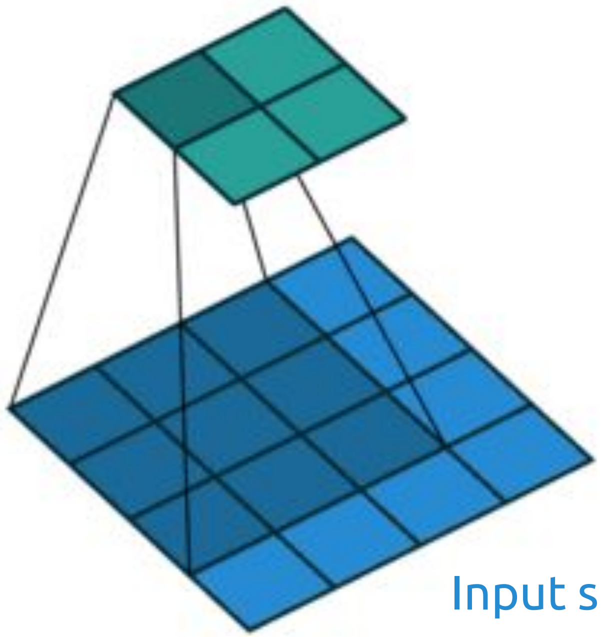


Any connection between input and output size?

Why would we want stride > 1?

Stride: 1

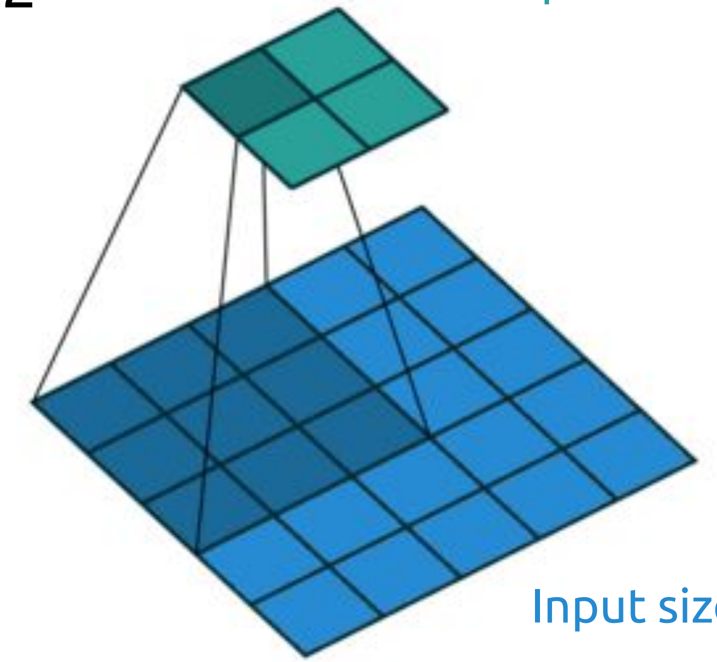
Output size: 2x2



Input size: 4x4

Stride: 2

Output size: 2x2



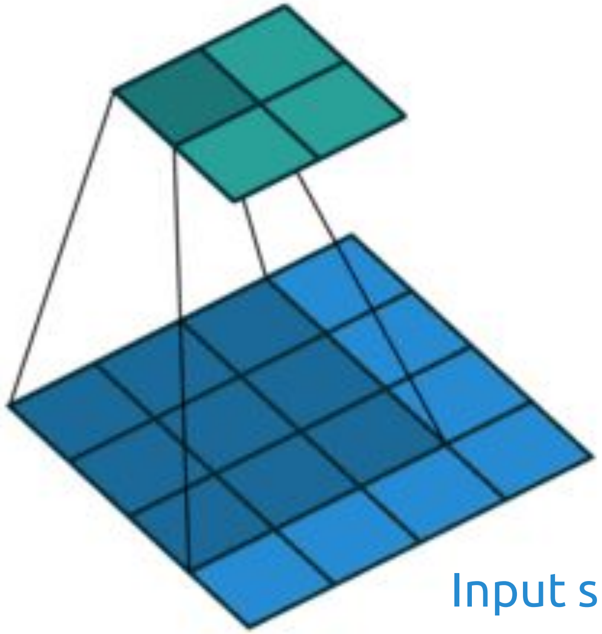
Input size: 5x5

Larger stride turns a bigger input into the same size output

Why would we want stride > 1?

Stride: 1

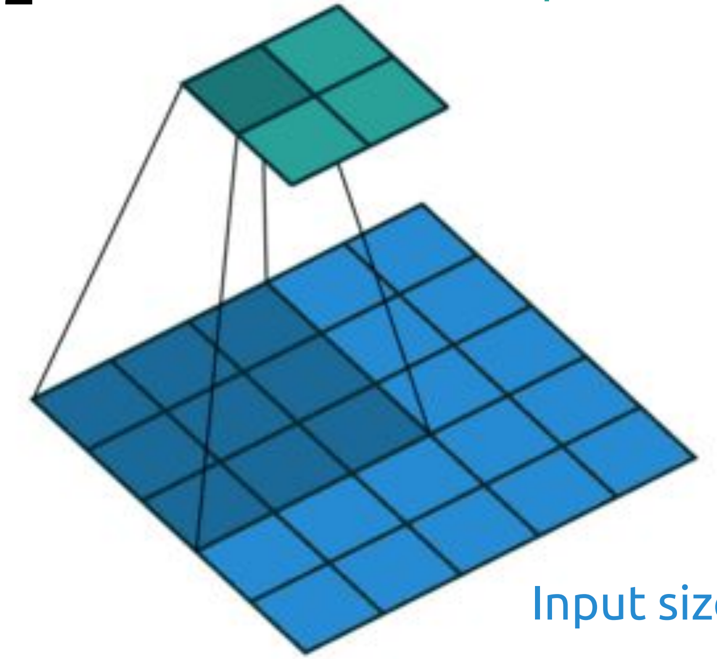
Output size: 2x2



Input size: 4x4

Stride: 2

Output size: 2x2



Input size: 5x5

Larger stride turns a bigger input into the same size output
Corollary: Larger stride turns the same size input into a *smaller* output
Use this to (controllably) decrease image resolution!

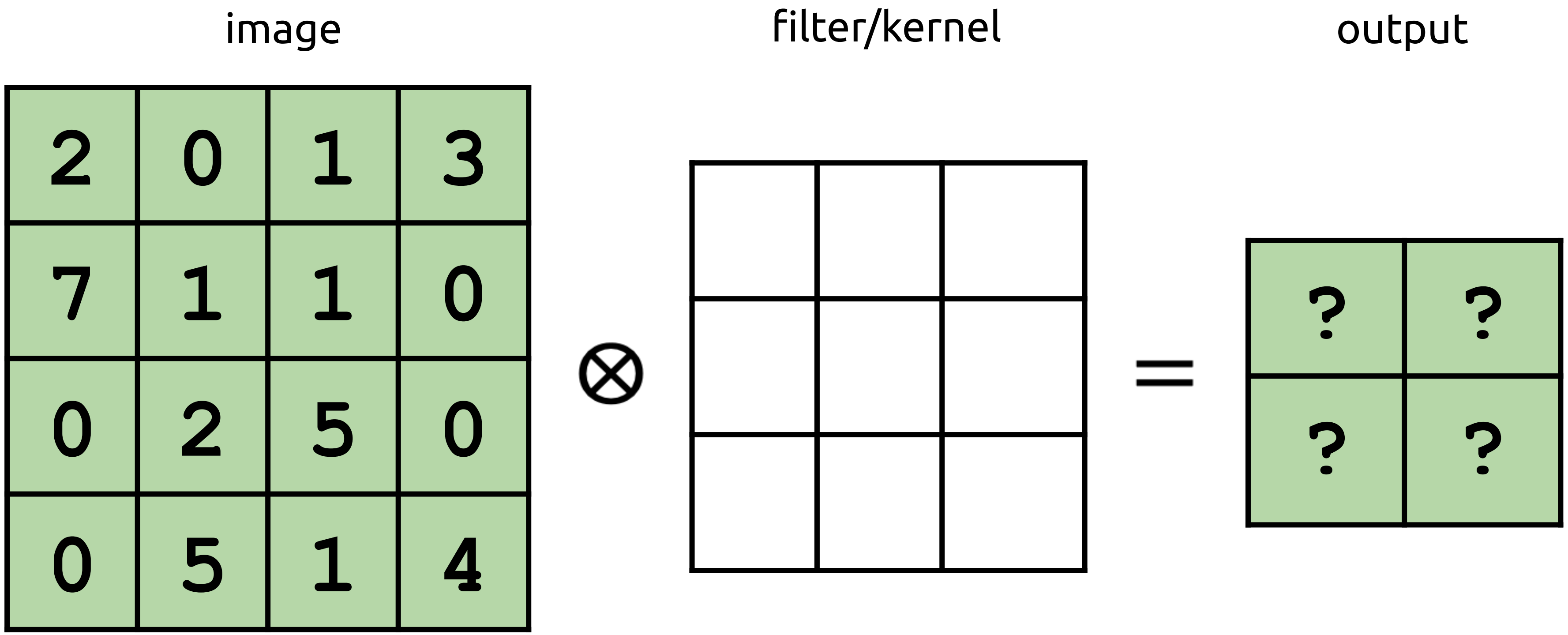
OK but...where's the *learning*?

Can you guess what do we learn in CNNs? (what are our parameters?)

Key Idea 1: Filters are ***Learnable***

$$\underset{\text{image}}{\begin{bmatrix} 2 & 0 & 1 & 3 \\ 7 & 1 & 1 & 0 \\ 0 & 2 & 5 & 0 \\ 0 & 5 & 1 & 4 \end{bmatrix}} \otimes \underset{\text{filter/kernel}}{\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}} = \underset{\text{output}}{\begin{bmatrix} -4 & -3 \\ 2 & -9 \end{bmatrix}}$$

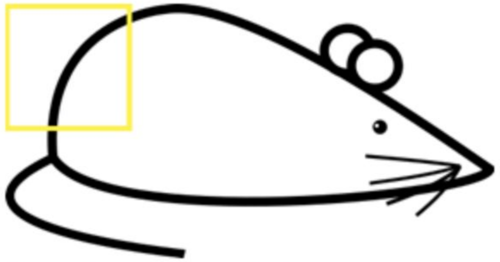
Key Idea 1: Filters are ***Learnable***



$k_{i,j}$ are learnable parameters

Key Idea 1: Filters are ***Learnable***

Original image

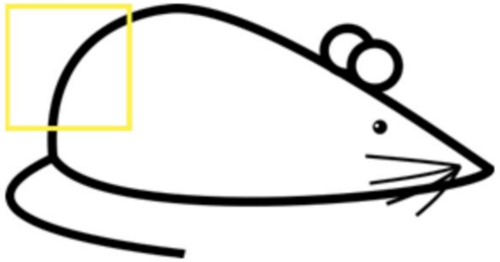


Visualization of the filter on the image

Label="Mouse"

Detecting patterns using learned filters

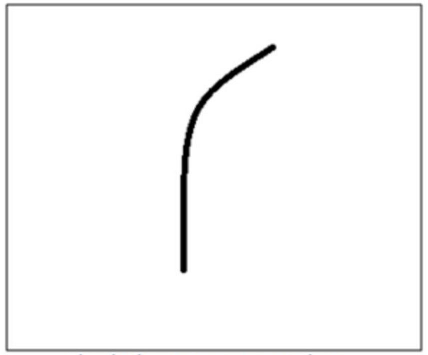
Original image



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!)

Image courtesy: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

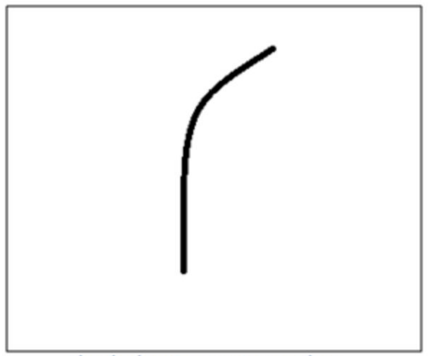
Detecting patterns using learned filters

Original image

How to detect other patterns?

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
[illegible]	[illegible]	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 filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

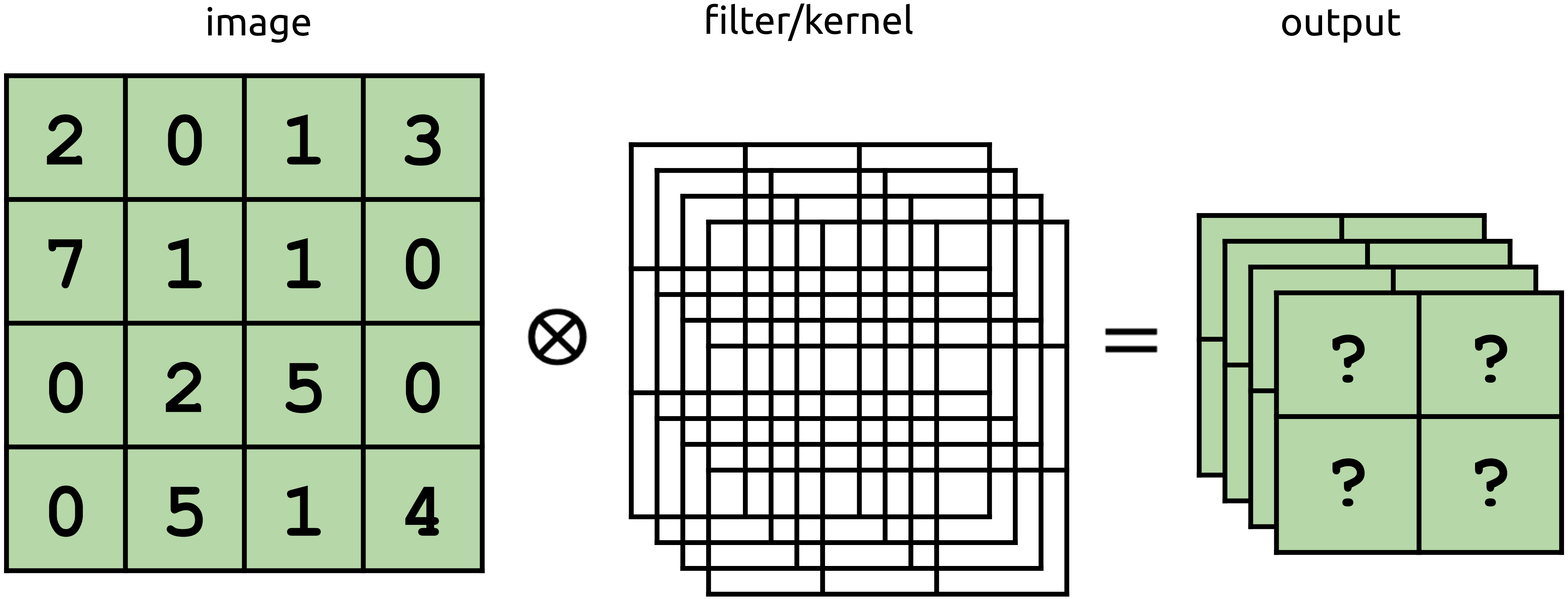
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

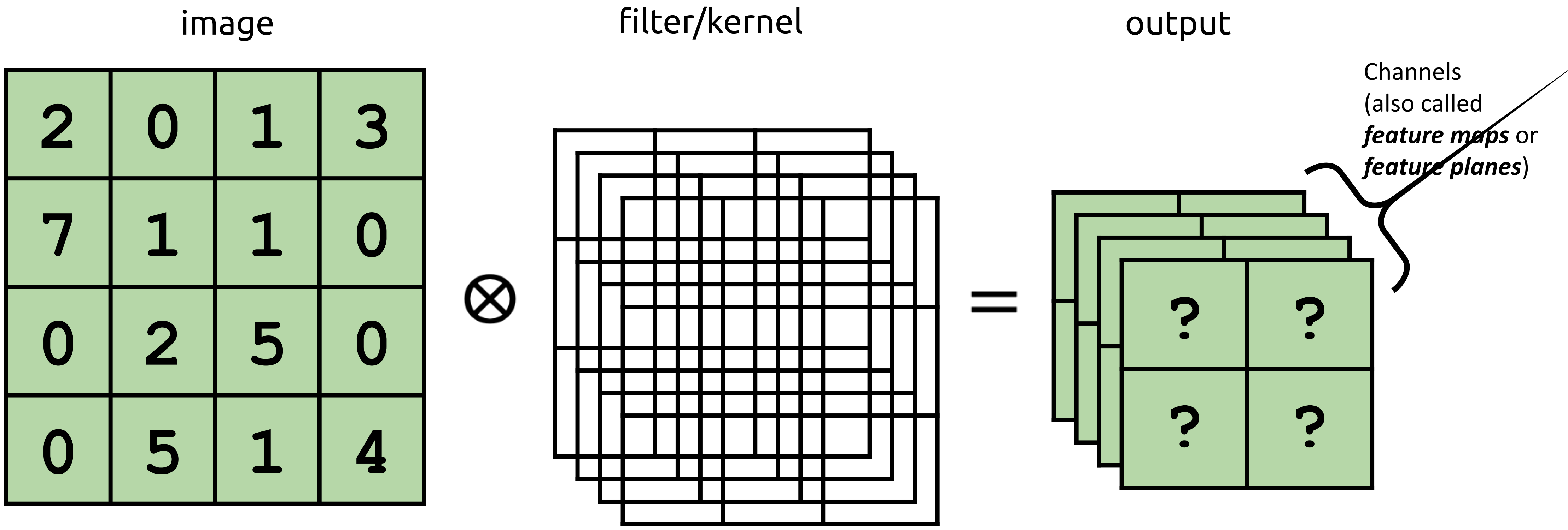
Image courtesy: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Key Idea 2: Learn *many* filters



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

Key Idea 2: Learn ***many*** filters

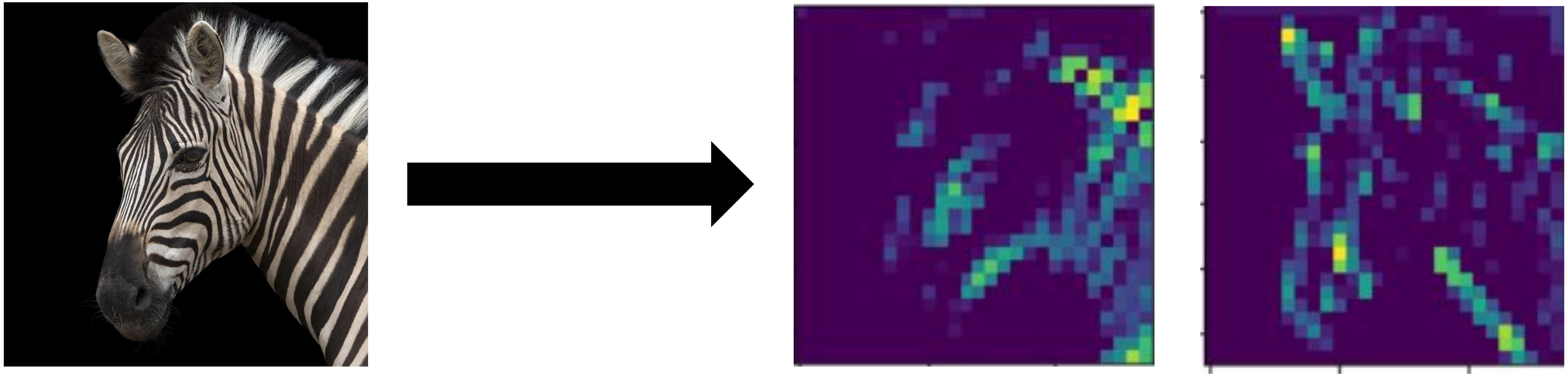


The output is now a multi-channel image

Any questions?

Key Idea 2: Learn *many* filters

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



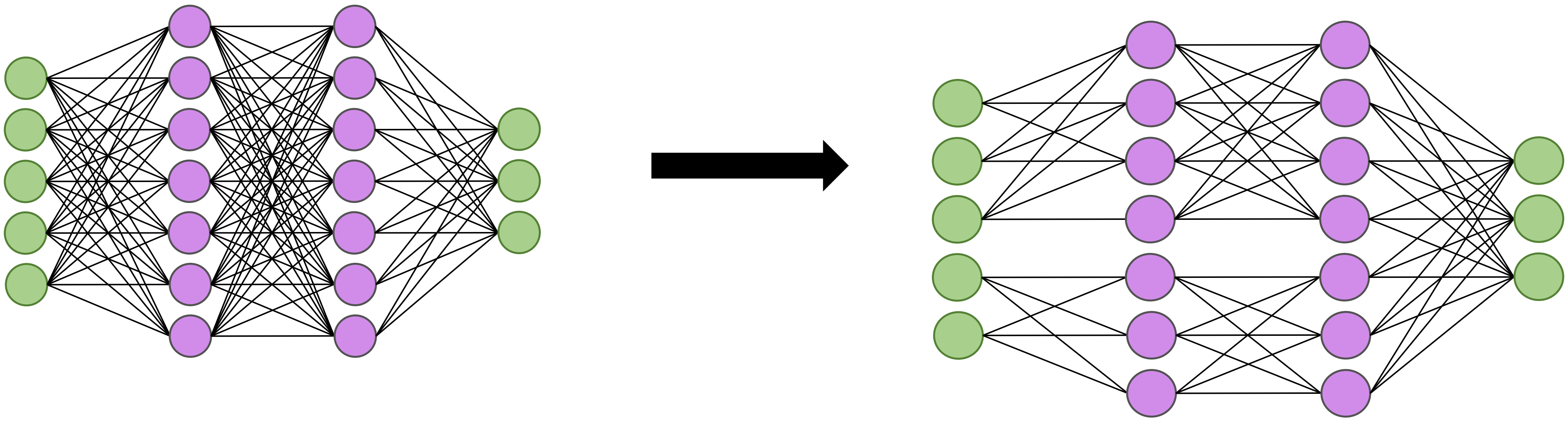
Input image Output of filter 1 Output of filter 2

You will explore this more in lab!

How is convolution “partially connected?”

Fully Connected

Partially Connected

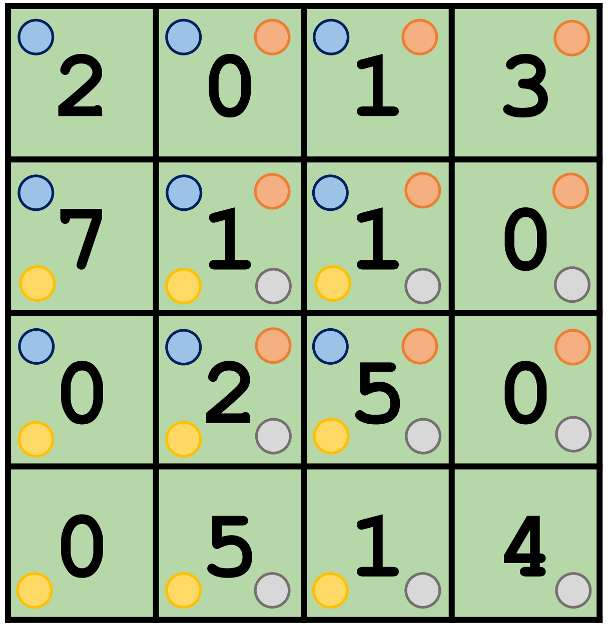


Only certain input pixels are "connected" to certain output pixels

$$\underset{\text{image}}{\begin{bmatrix} 2 & 0 & 1 & 3 \\ 7 & 1 & 1 & 0 \\ 0 & 2 & 5 & 0 \\ 0 & 5 & 1 & 4 \end{bmatrix}} \otimes \underset{\text{filter/kernel}}{\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}} = \underset{\text{output}}{\begin{bmatrix} -4 & -3 \\ 2 & -9 \end{bmatrix}}$$

Only certain input pixels are “connected” to certain output pixels

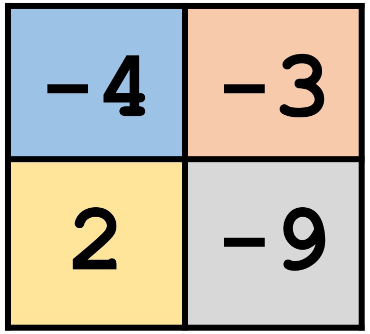
image



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?

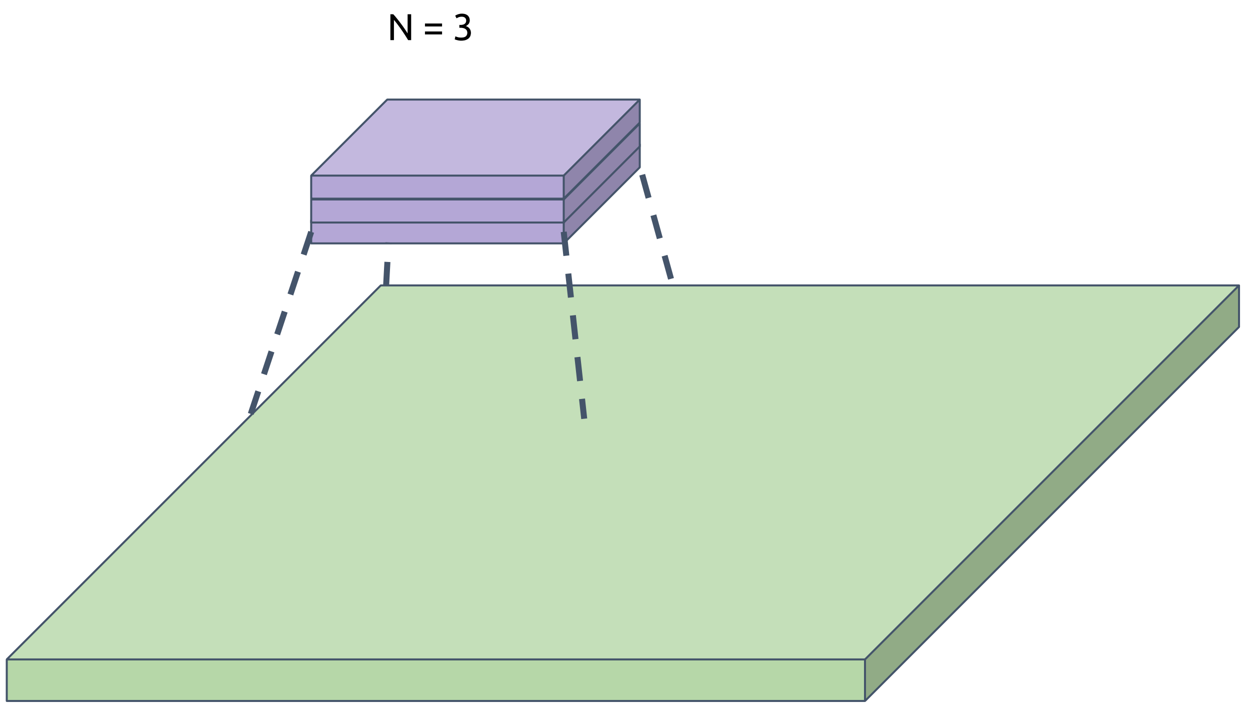
Input Image (4-D Tensor)
Shape:
[batchSz, input_height, input_width, input_channels]

Full documentation here: https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv2d

Output Size of a Convolution Layer

The output size of a convolution layer depends on 4 Hyperparameters:

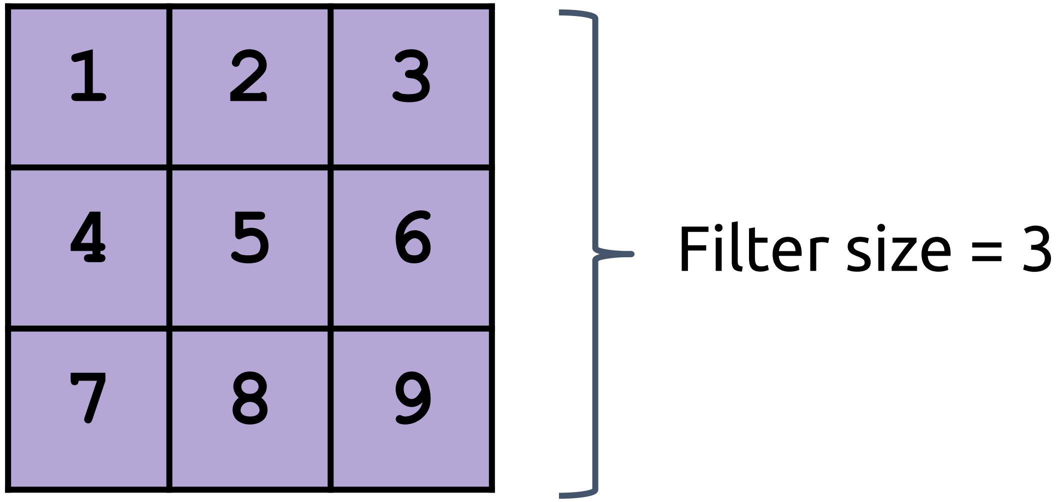
- Number of filters, **N**



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**



Convolution in Tensorflow

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

Can you guess the shape?

Kernel (4-D Tensor)
Shape:
[f_height, f_width, in_channels, out_channels]

Full documentation here: https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv2d

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

Stride = 2

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]

Full documentation here: https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv2d

Convolution in Tensorflow

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

?

Full documentation here: https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv2d

“Problem” With Convolution

$$\begin{bmatrix} 2 & 0 & 1 & 3 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 0 & -1 \end{bmatrix}$$

- 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

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

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?

?	?	?	?	?	?	?
?	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

What Values to Use For These Pixels?

Standard practice: fill with zeroes

- Zero-valued padding pixels just result in some terms in the convolution sum being zero

$$V(x,y) = (I \otimes K)(x,y) = \sum_m \sum_n I(x+m, y+n)K(m,n)$$

This is zero for a padding pixel

- End result: equivalent to a applying a ‘masked’ version of the filter that only covers the valid pixels

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

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

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

VALID Padding in Tensorflow

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

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

VALID Padding in Tensorflow

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

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

VALID Padding in Tensorflow

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

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

We already tried this! (reduced output size)

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

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 (Try it as HW)

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

Convolution in Tensorflow

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

The mode of padding to use (String)
Either “Valid” or “Same”
Case-insensitive

Full documentation here: https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv2d

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

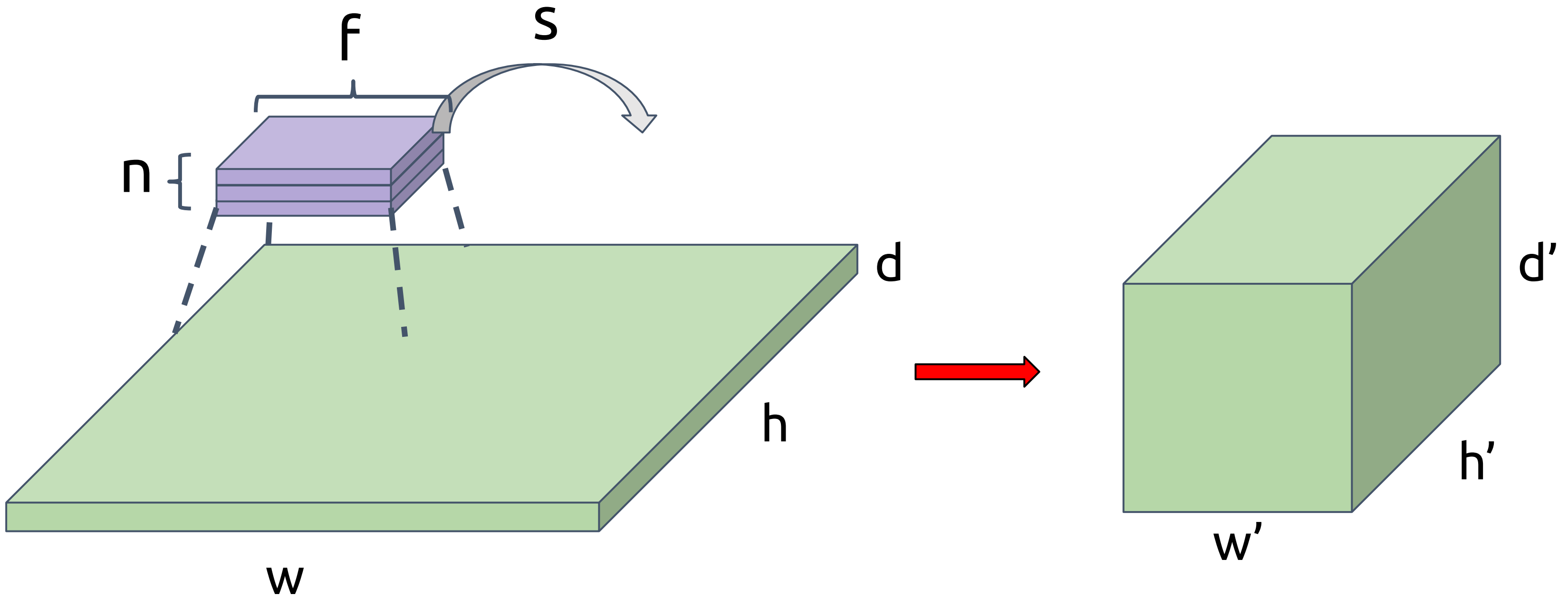
Suppose we know the number of filters, their size, the stride, and padding (**n,f,s,p**).

Then for a convolution layer with input dimension **w x h x d**, the output dimensions **w’ x h’ x d’** are:

$$w' = \frac{w - f + 2p}{s} + 1$$

$$h' = \frac{h - f + 2p}{s} + 1$$

$$d' = n$$



Output Size for “VALID” Padding

$$w' = \frac{w \ - f + 2p}{s} + 1$$

Let $w = 4$

num filters n = 1
filter size f = 3
stride s = 1
padding p = 0

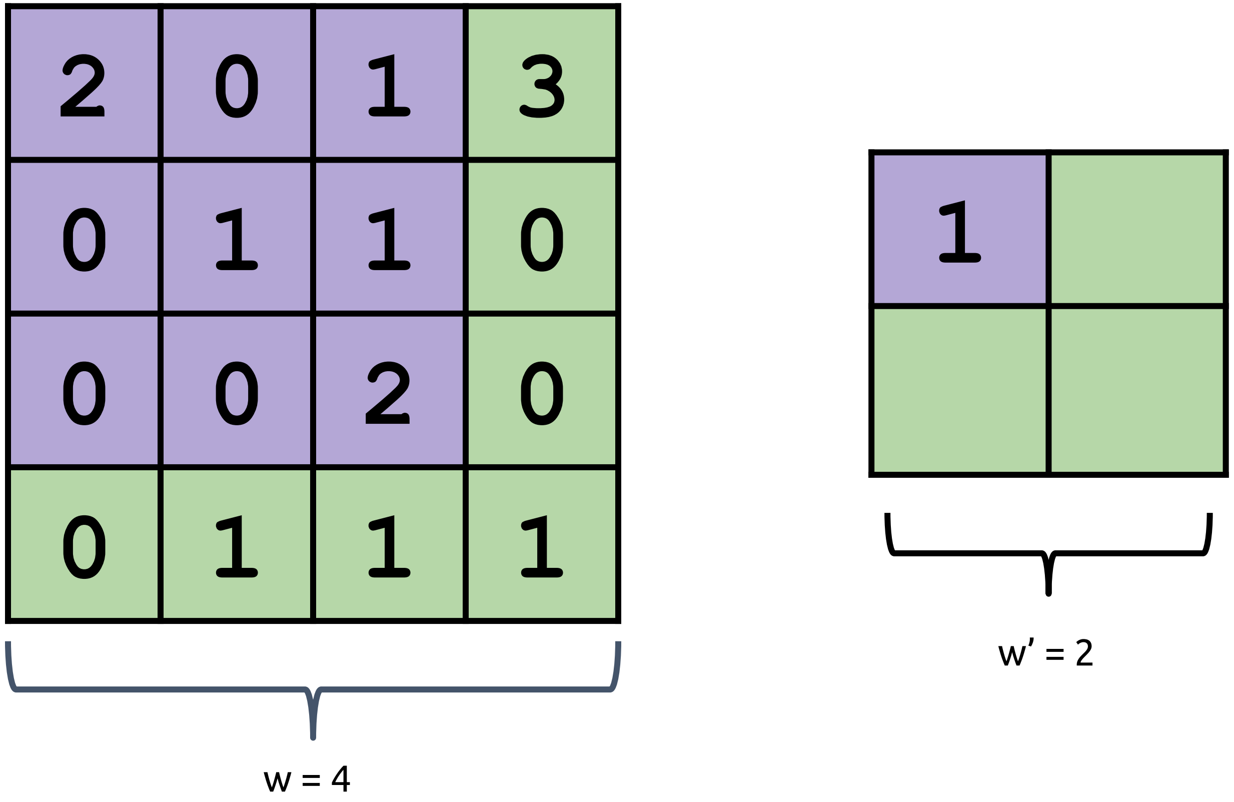
$$w' = \frac{4 \ - 3 + 2 \cdot 0}{1} + 1$$

$$= 1 + 1 = 2$$

Output Size for “VALID” Padding

$$w' = \frac{w - f + 2p}{s} + 1$$

num filters n = 1
filter size f = 3
stride s = 1
padding p = 0



Output Size for “VALID” Padding

$$w' = \frac{w - f + 2p}{s} + 1$$

num filters n = 1
filter size f = 3
stride s = 1
padding p = 0

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

w = 4

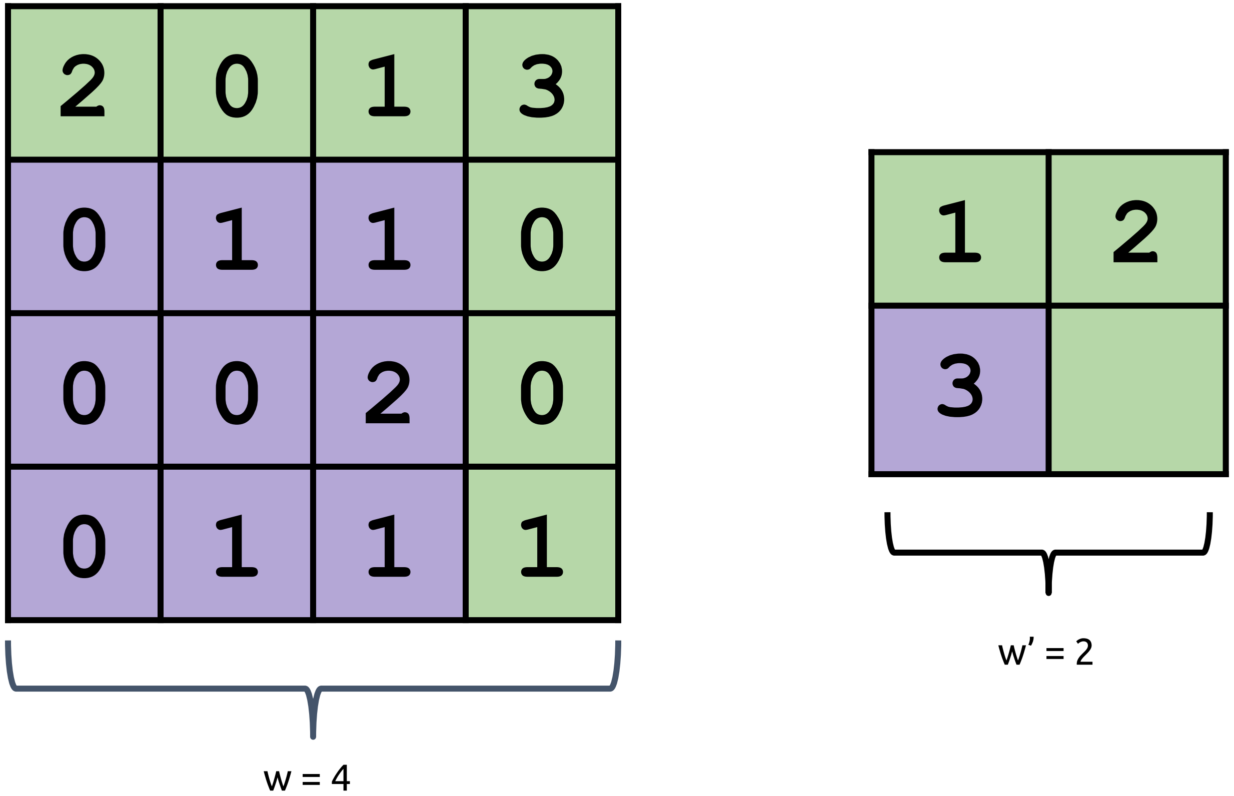
1	2

w’ = 2

Output Size for "VALID" Padding

$$w' = \frac{w - f + 2p}{s} + 1$$

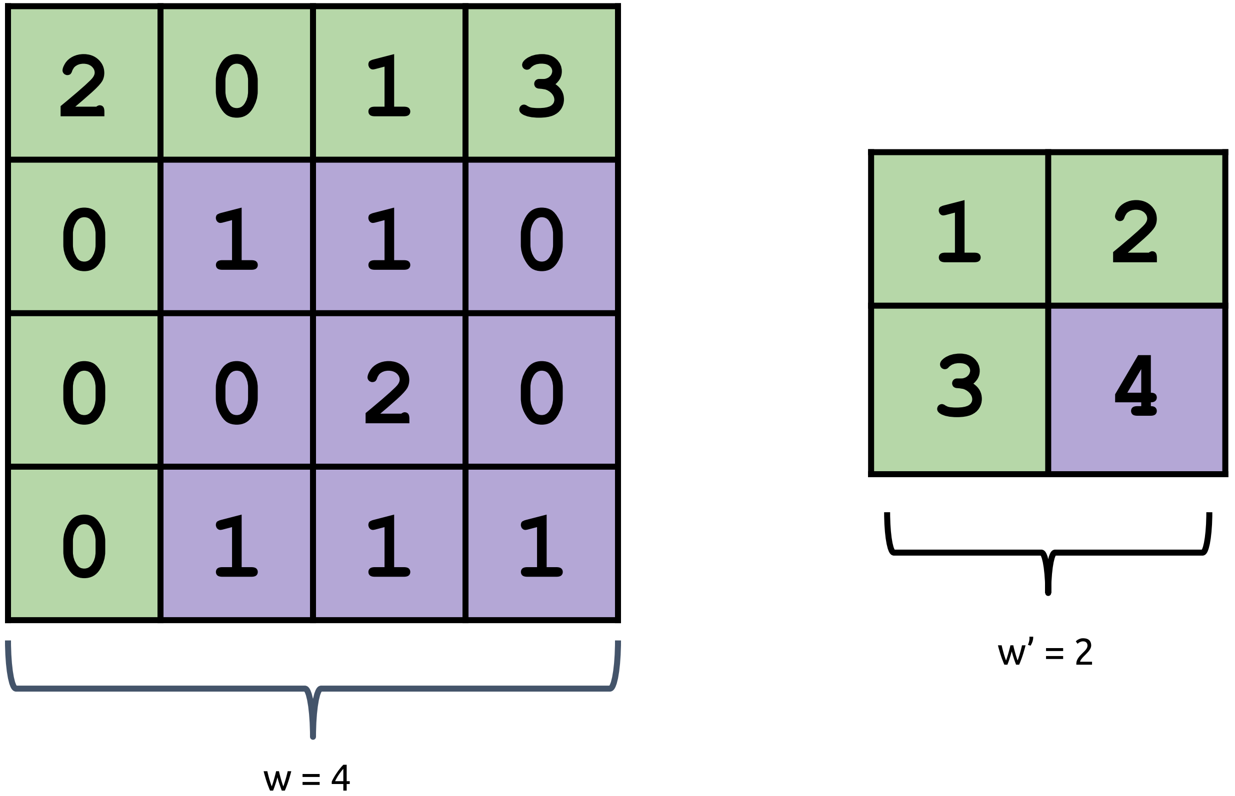
num filters n = 1
filter size f = 3
stride s = 1
padding p = 0



Output Size for “VALID” Padding

$$w' = \frac{w - f + 2p}{s} + 1$$

num filters n = 1
filter size f = 3
stride s = 1
padding p = 0



Output Size for “SAME” Padding

$$w' = \frac{w \ - f + 2p}{s} + 1$$

Let $w = 4$

num filters n = 1
filter size f = 3
stride s = 1
padding p = 1*

$$w' = \frac{4 \ - 3 + 2 \cdot 1}{1} + 1$$

$$= 3 + 1 = 4$$

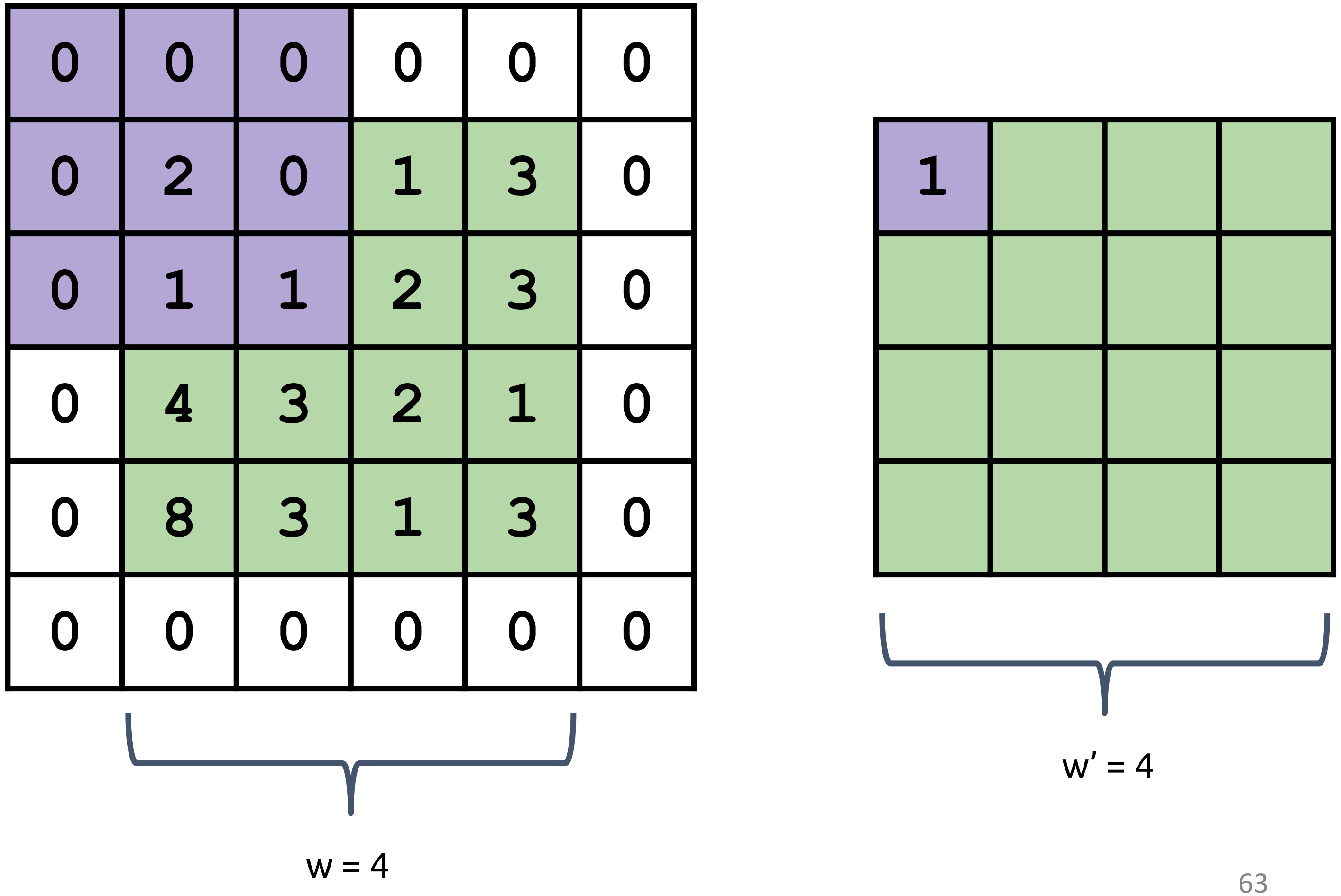
*Chosen so output size is the same

Output Size for “SAME” Padding

$$w' = \frac{w - f + 2p}{s} + 1$$

num filters n = 1
filter size f = 3
stride s = 1
padding p = 1*

*Chosen so output size is the same



Output Size for "SAME" Padding

$$w' = \frac{w - f + 2p}{s} + 1$$

num filters n = 1
filter size f = 3
stride s = 1
padding p = 1*

*Chosen so output size is the 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

w = 4

1	2		

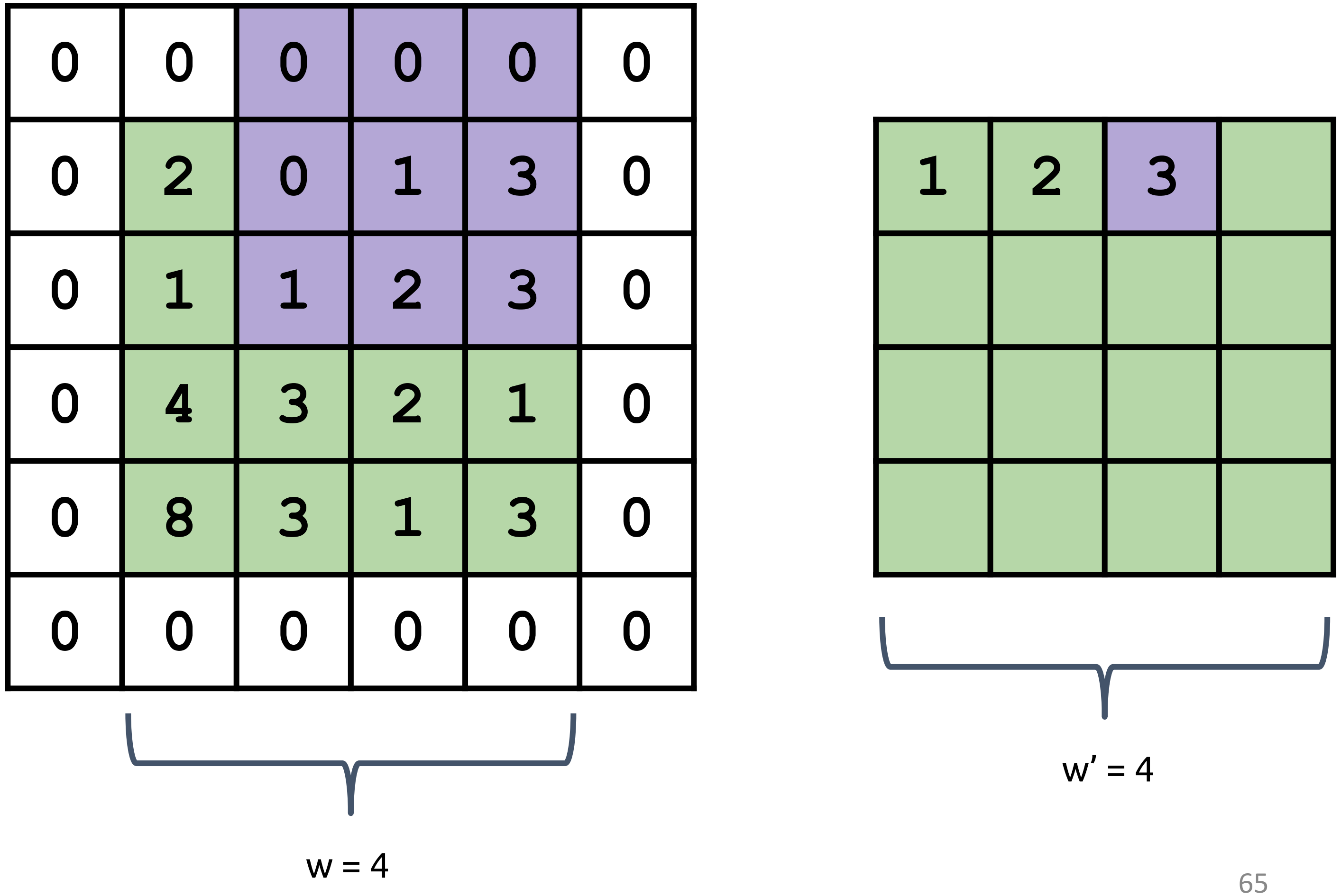
w' = 4

Output Size for “SAME” Padding

$$w' = \frac{w - f + 2p}{s} + 1$$

num filters n = 1
filter size f = 3
stride s = 1
padding p = 1*

*Chosen so output size is the same



Any questions?

Output Size for "SAME" Padding

$$w' = \frac{w - f + 2p}{s} + 1$$

num filters n = 1
filter size f = 3
stride s = 1
padding p = 1*

*Chosen so output size is the 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

w = 4

1	2	3	4

w' = 4

Convolution in Tensorflow

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

Input Image
(4-D Tensor)

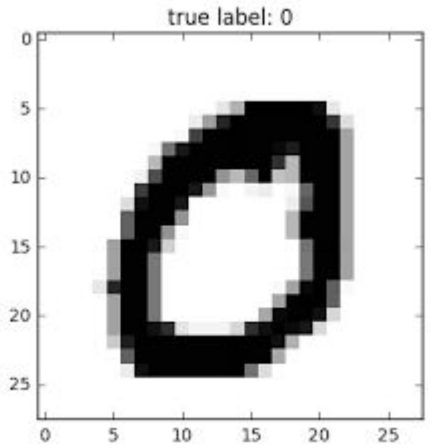
Filter/Kernel
(4-D Tensor)

Strides along
each dimension

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

Full documentation here: https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv2d

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 ? input channels and 16 output channels
self.filter = tf.Variable(tf.random.normal([?, ?, ?, ?], stddev=0.1))

# Convolves the input batch with our defined filter
conv = tf.nn.conv2d(?,?,?,?)
```

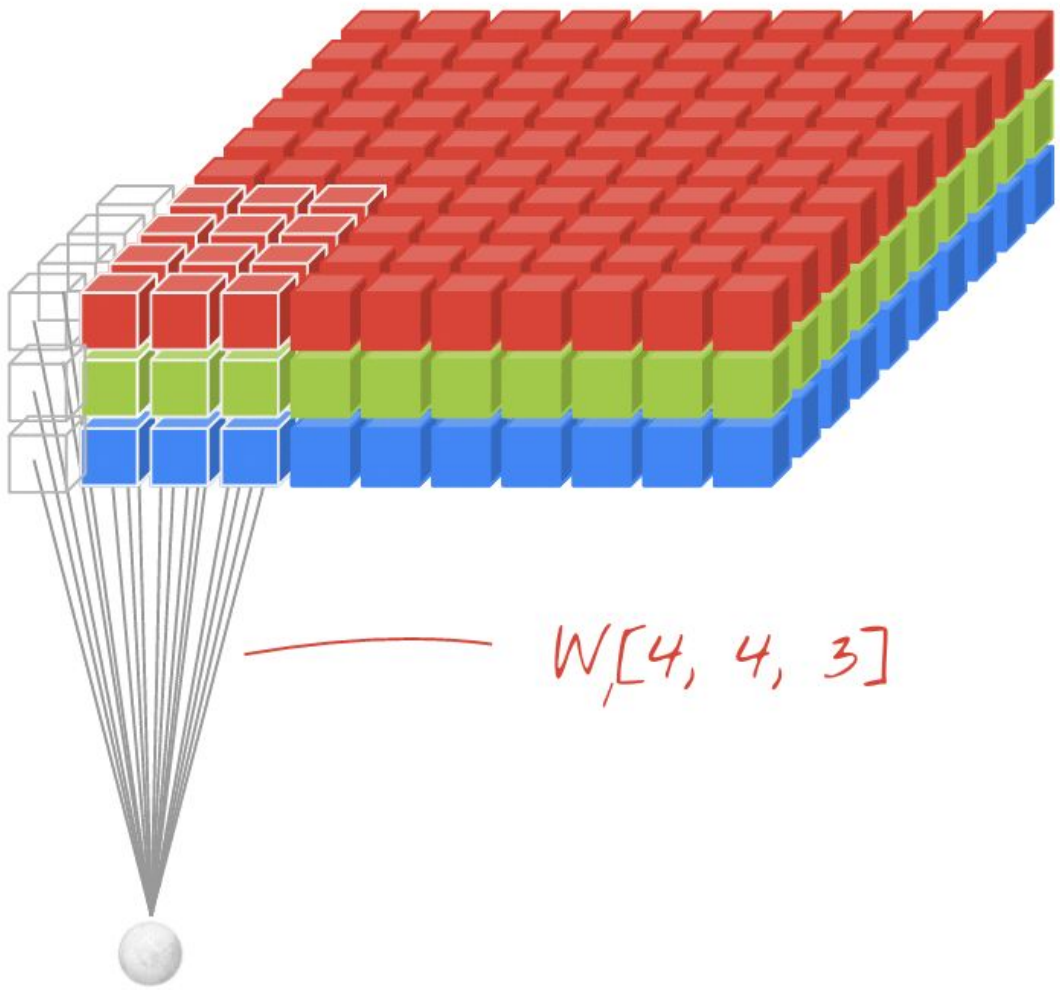
Application to Real World Data (CIFAR)

```
# 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")
```

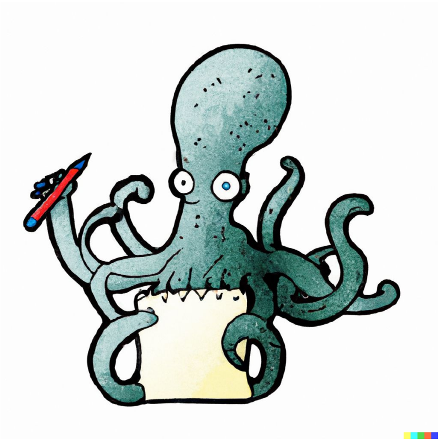
2D Convolution for 3D Image



Source, Author: Martin Görner

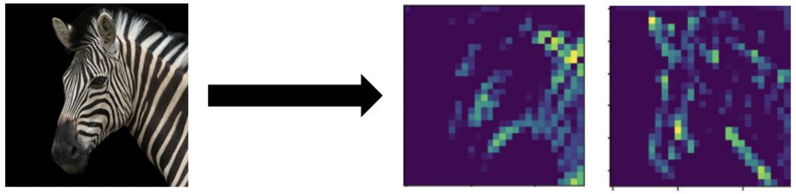
See also: Understanding Convolution by hand vs TensorFlow

Recap



Convolution

- Filters/Kernels and Stride
- Learning filters
- CNNs are partially connected networks



Convolution in Tensorflow

- Tensorflow conv2d function
- Padding
- Application to MNIST/CIFAR

```
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")