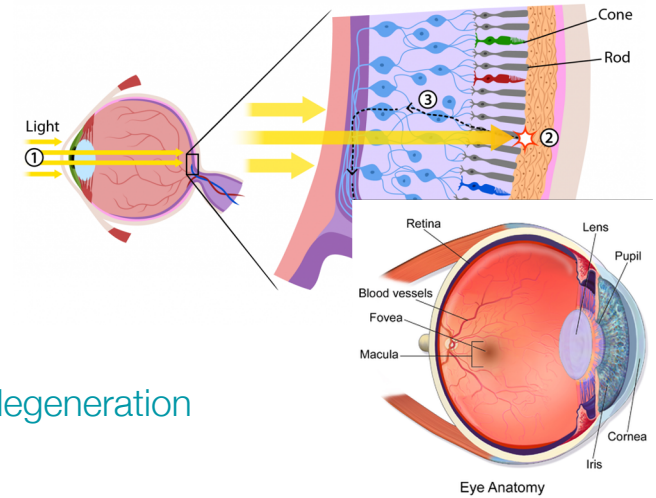
The convolution operation

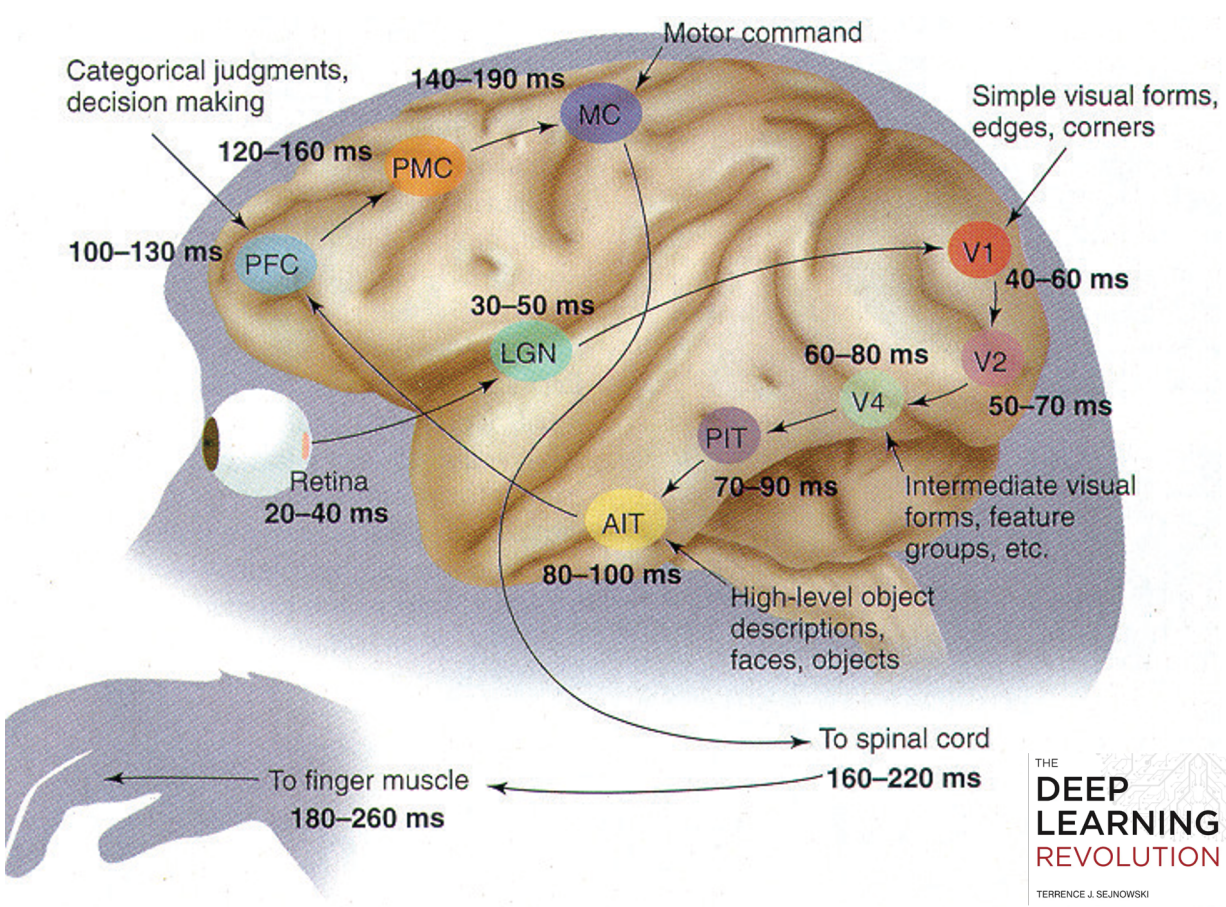
Our eyes are not as powerful as we think

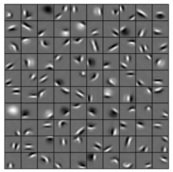
- We have ‘cone’ and ‘rod’ cells in our eye (photoreceptive cells)
  - Rods are responsible for vision at low light levels; do not mediate color vision; have low spatial acuity
  - Cones are active at higher light levels; are capable of color vision; are responsible for high spatial acuity. The central fovea is populated exclusively by cones.
- The peripheral vision has low spatial resolution
  - But, it is very sensitive to changes in brightness and motion
- The fovea (sharp vision) is only 1.1 mm wide
  - Effectively only 1 degree of arc across (in our vision)
  - This is about the size of thumb at arm’s length
  - We are legally blind beyond fovea
  - If the cells in the fovea (macula) deteriorate you have macular degeneration
- But, “I can see a full computer screen so clearly”
  - We have illusion of high resolution everywhere because we can rapidly reposition our eyes
  - When you gaze at an object, our eyes dart back and forth over the object three times a second
- Our ‘eyes’ also have a blind spot (each has separate spot)

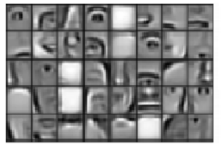


Eye Anatomy

Human visual cortex is hierarchical



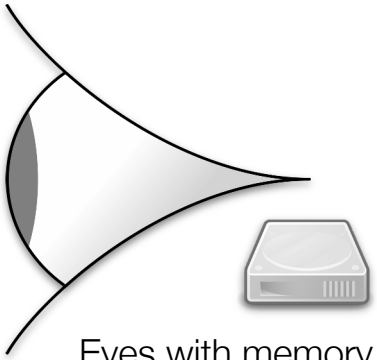




LGN: lateral geniculate nucleus
V1: primary visual cortex
V2: secondary visual cortex
V4: visual area 4
AIT and PIT: anterior and posterior inferotemporal cortex
PFC: prefrontal cortex
PMC: premotor cortex
MC: motor cortex

Densely connected NNs vs Convolutional NNs

- A convolutional neuron does what a lifeguard does at a pool
- A lifeguard scans the pool's surface creating an output map
- Image multiple lifeguards looking at a pool
  - One knows the relationship between light reflections and drowning
  - Another knows the relationship between the body size and drowning
  - Another knows the relationship between the water splashing and drowning



Eyes with memory

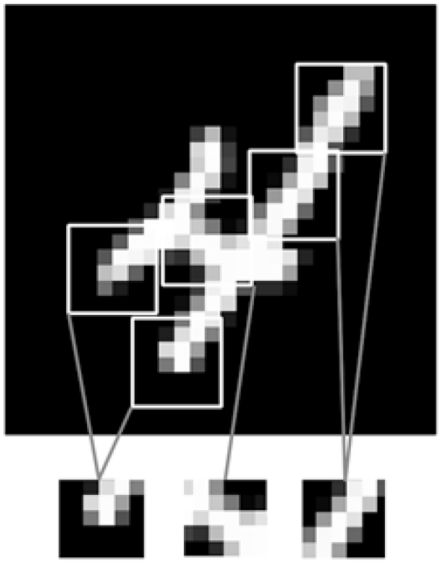
Densely connected NNs vs Convolutional NNs

https://www.youtube.com/watch?v=sMkGwCw7iv8

5.1.1 The convolution operation

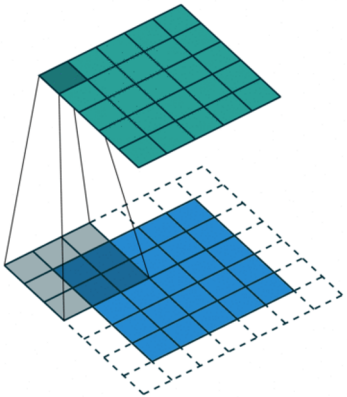
What is the difference between a densely connected layer and a convolution layer?

- Dense layers learn global patterns in their input feature space
- Convolution layers learn local patterns (patterns found in small 2D windows of 3x3)



Global pattern

Local patterns



If dense layers can learn “global” patterns (generalization) then why do we need a convolutional layer?

Property 1 of CNNs - Translation invariance

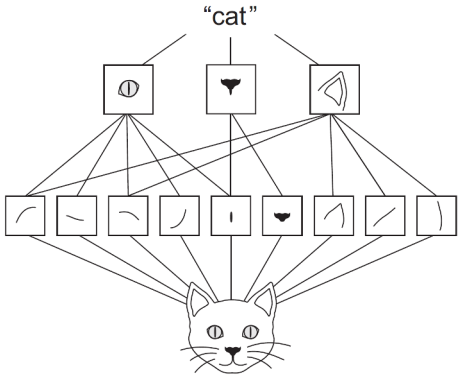
The patterns they learn are “translation invariant”

- The visual world is translation invariant
  - You don’t always have Chicago on your North
- Convolutional layer learns a pattern in lower-right corner of a picture!
  - Then, it can recognize it anywhere else in another/same picture
- This makes convnets data efficient
  - They can learn more patterns with fewer training samples

Property 2 of CNNs - Hierarchies of patterns

Property 2 of CNNs - They can learn spatial “hierarchies of patterns”

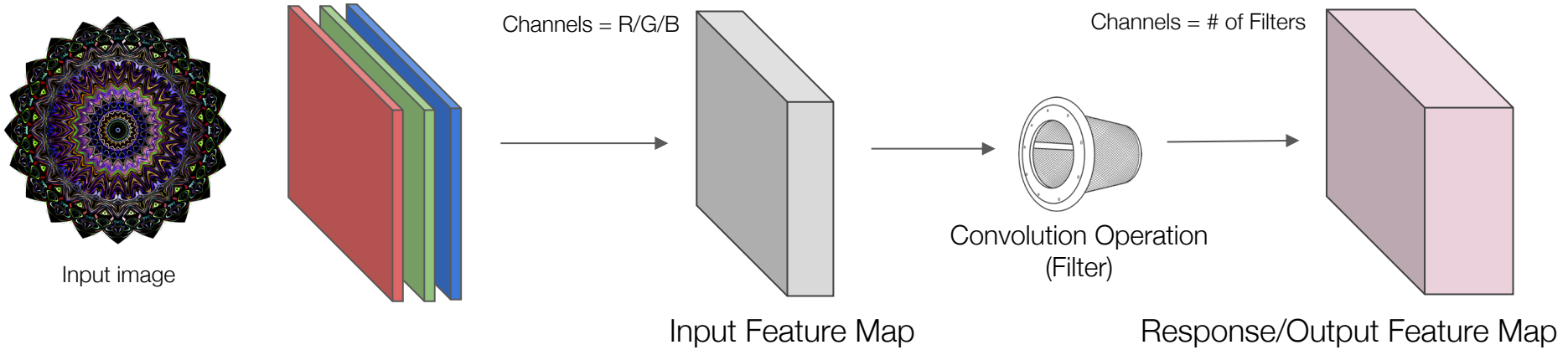
- The visual world is spatially hierarchical



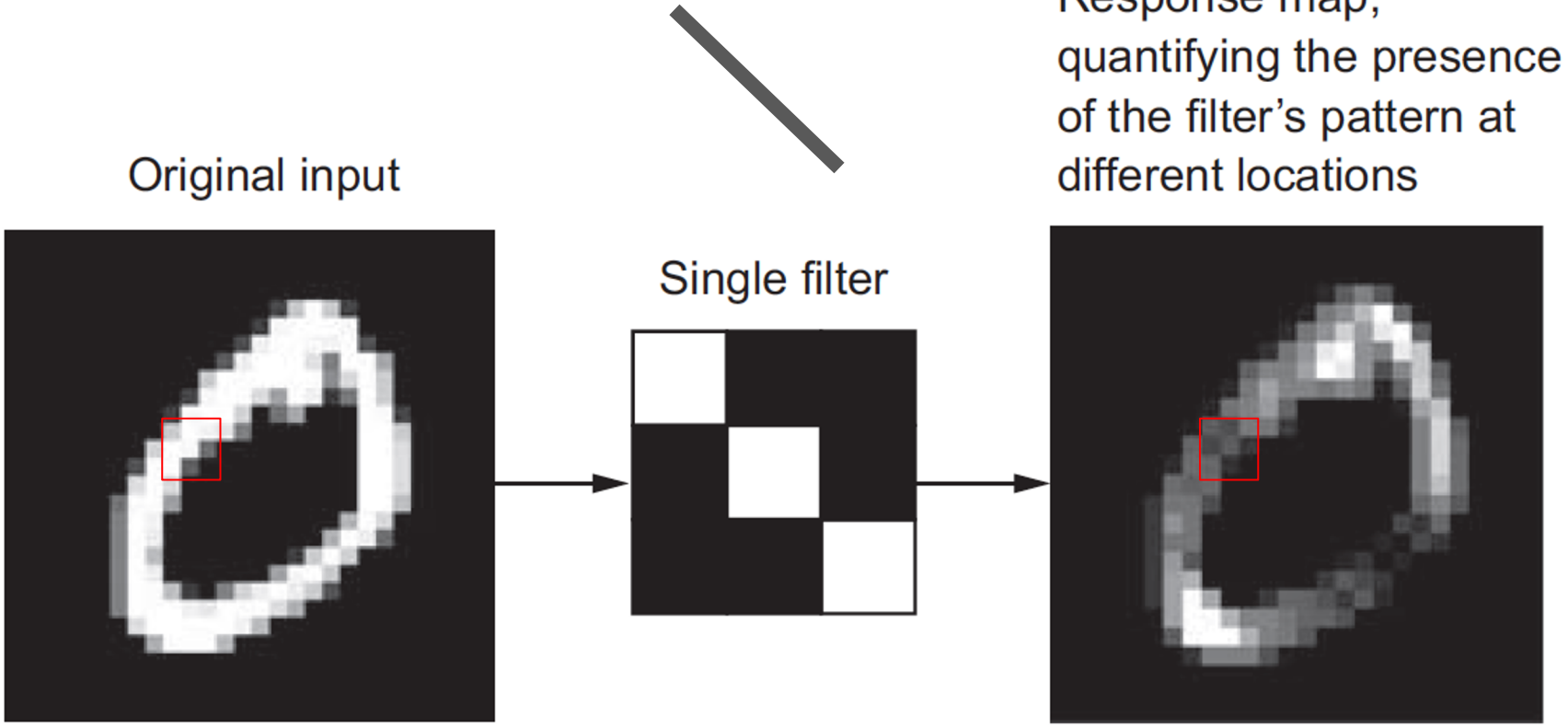
- A first convolution layer can learn small local patterns such as edges
  - Second convolution layer can learn larger patterns made of the features of the first layers, and so on
- This allows convnets to efficiently learn increasingly complex and abstract visual concepts

Basics of a convolutional layer (Conv2D)

- Convolutions operate over 3D tensors, called feature maps
  - Feature map - two spatial axes (height and width) & a depth axis (also called the channels axis)
  - For an RGB image, # of channels = 3
  - For a black-and-white picture, depth/channels = 1
- The convolution operation *extracts patches from its "input feature map"* and *applies the same transformation to all of these patches*, producing an "output feature map"
  - Depth of a feature map can be arbitrary (channels in the depth axis stand for filters)



Example of a filter



Two parameters of a convolutional layer

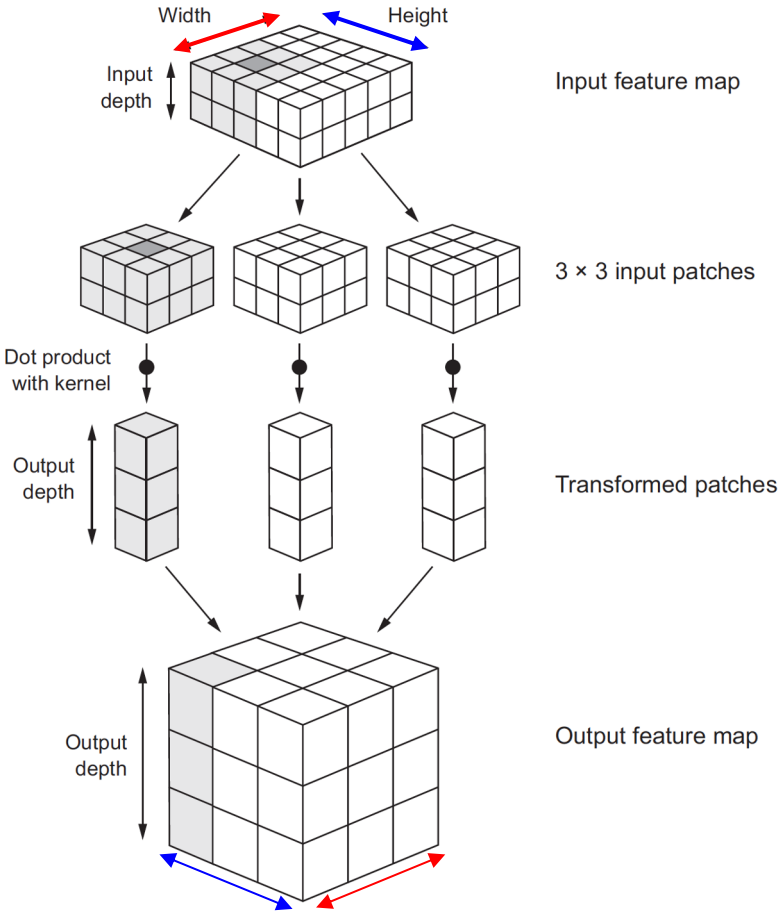
1. Size of the patches extracted from the inputs
   - Typically 3 × 3 or 5 × 5

2. Depth of the output feature map
   - The number of filters computed by the convolution

```
keras.layers.Conv2D(filters, kernel_size, …)
```

Understood as `(kernel_size, kernel_size)`

The convolution process



The height and width of the output feature map may not be same as the input. Why ? (two reasons)

Parameters in a CNN (Example 1)

```
1 my_input = Input(shape = (10, 10, 1))
2 my_output = Convolution2D(1, 3)(my_input)
3 my_output = Activation('sigmoid')(my_output)
4 model = Model(my_input, my_output)
```

```
1 model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_17 (InputLayer)        [(None, 10, 10, 1)]       0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 8, 8, 1)           10
_________________________________________________________________
activation_19 (Activation)   (None, 8, 8, 1)           0
=================================================================
Total params: 10
Trainable params: 10
Non-trainable params: 0
_________________________________________________________________
```

Parameters in a CNN (Example 2)

```
1 my_input = Input(shape = (10, 10, 1))
2 my_output = Convolution2D(5, 3)(my_input)
3 my_output = Activation('sigmoid')(my_output)
4 my_output = Convolution2D(2, 3)(my_output)
5 my_output = Activation('sigmoid')(my_output)
6 model = Model(my_input, my_output)
```

```
1 model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_19 (InputLayer)        [(None, 10, 10, 1)]       0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 8, 8, 5)           50
_________________________________________________________________
activation_22 (Activation)   (None, 8, 8, 5)           0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 6, 6, 2)           92
_________________________________________________________________
activation_23 (Activation)   (None, 6, 6, 2)           0
=================================================================
Total params: 142
Trainable params: 142
Non-trainable params: 0
_________________________________________________________________
```

Parameters in a CNN (Example 3)

```
1 my_input = Input(shape = (100, 100, 3))
2 my_output = Convolution2D(8, 3)(my_input)
3 my_output = Activation('sigmoid')(my_output)
4 my_output = Convolution2D(1, 3)(my_output)
5 my_output = Activation('sigmoid')(my_output)
6 model = Model(my_input, my_output)
```

```
1 model.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
input_20 (InputLayer)        [(None, 100, 100, 3)]     0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 98, 98, 8)         224
_________________________________________________________________
activation_24 (Activation)   (None, 98, 98, 8)         0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 96, 96, 1)         73
_________________________________________________________________
activation_25 (Activation)   (None, 96, 96, 1)         0
=================================================================
Total params: 297
Trainable params: 297
Non-trainable params: 0
_________________________________________________________________
```

Parameters in a CNN (Example 4)

```
1 my_input = Input(shape = (100, 5))
2 my_output = Convolution1D(8, 3)(my_input)
3 my_output = Activation('sigmoid')(my_output)
4 my_output = Convolution1D(1, 3)(my_output)
5 my_output = Activation('sigmoid')(my_output)
6 model = Model(my_input, my_output)
```

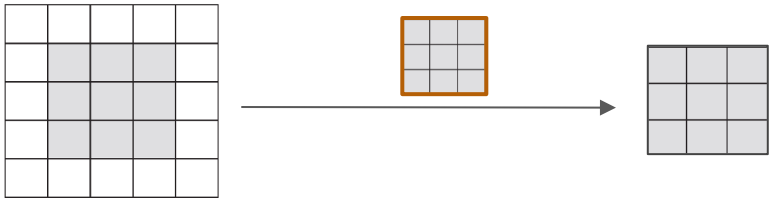
```
1 model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_25 (InputLayer)        [(None, 100, 5)]          0
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 98, 8)             128
_________________________________________________________________
activation_30 (Activation)   (None, 98, 8)             0
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 96, 1)             25
_________________________________________________________________
activation_31 (Activation)   (None, 96, 1)             0
=================================================================
Total params: 153
Trainable params: 153
Non-trainable params: 0
_________________________________________________________________
```

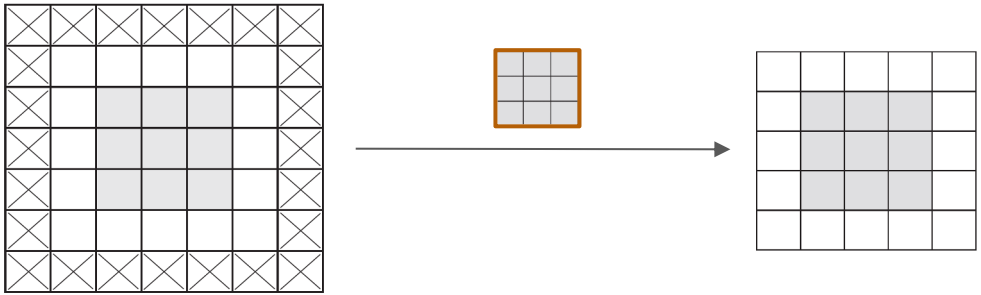
In addition to the convolution operation..

Border effect & padding

When a convolution of 3x3 is applied to an input of 5x5, the output is 3x3



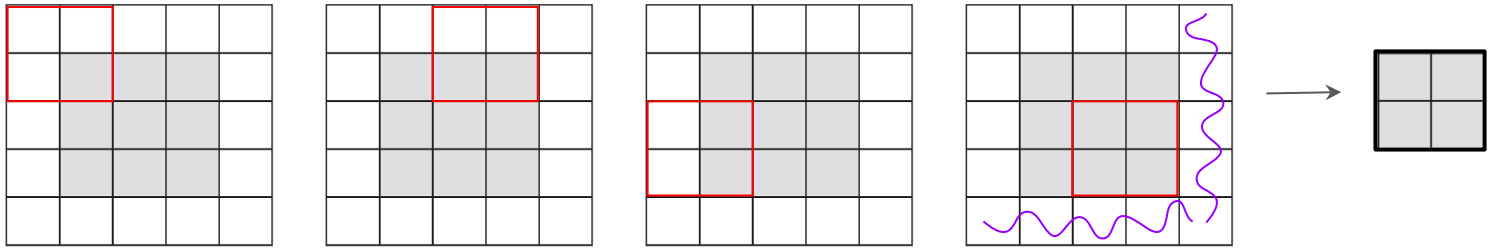
To obtain a 5x5 output, we can pad the input with zeros (called **zero-padding**)



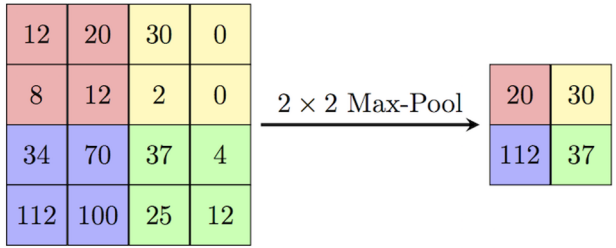
```
my_output = Convolution2D(1, 3, padding = 'same')(my_input)
```

Max-pooling

- Max pooling consists of extracting windows from the input feature maps and outputting the max value of each channel
- MaxPooling2D halves the size of the feature maps
  - For example, input of 26 × 26 halves to 13 × 13

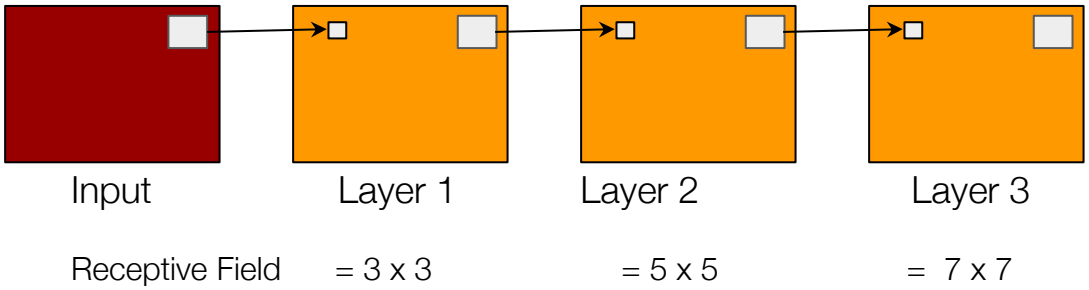


- It's conceptually similar to convolution
  - Instead of transforming local patches via a learned linear transformation (the convolution kernel), they're transformed via a hardcoded **max** tensor operation



Why Maxpool?

```
model_no_max_pool = models.Sequential()
model_no_max_pool.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model_no_max_pool.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_no_max_pool.add(layers.Conv2D(64, (3, 3), activation='relu'))
```



- The 3 x 3 windows in the third layer will contain information coming from 7 x 7 windows in the initial input
- We need the features from the last convolution layer to contain information about the totality of the input
  - How can we achieve that?

Why Maxpool?

```
model_no_max_pool = models.Sequential()
model_no_max_pool.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model_no_max_pool.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_no_max_pool.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

- We need to add at least one dense layer at the end
  - So that we have one neuron that can predict [0, 1]
- If we add 1 Dense layer to this network, how many parameters are we adding?
  - 64 x (22 x 22) + 1 = 30,977 parameters
- If we add 1 Dense layer with 100 neurons and second with 1 neuron?
  - ?
- Having too many parameters will cause intense overfitting issues!
  - Stacks of Max-pooling layers can decrease the size of feature maps

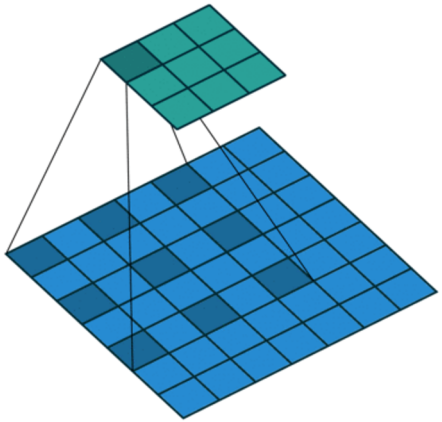
Different Types of Convolutions

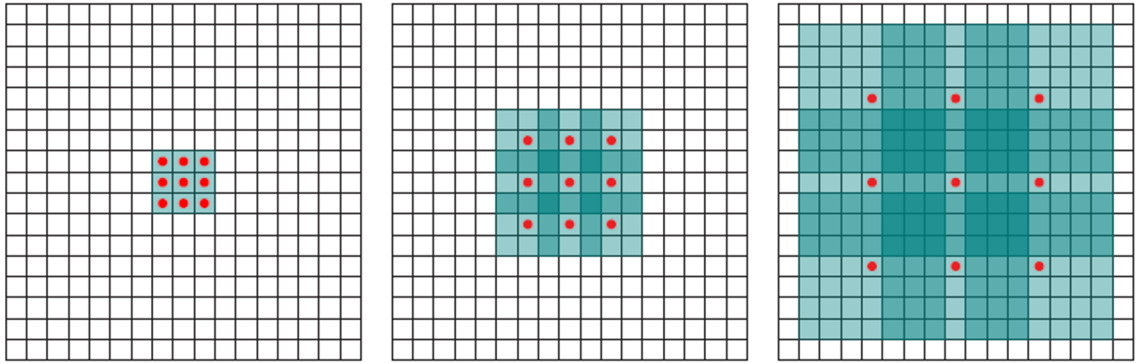
https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d &
https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728

Dilated convolutions

```
keras.layers.Conv2D(filters, kernel_size, dilation_rate=(1, 1))
```

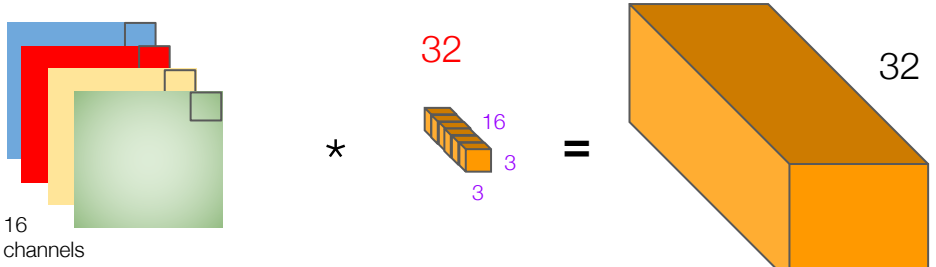
Dilation rate defines a spacing between the values in a kernel.





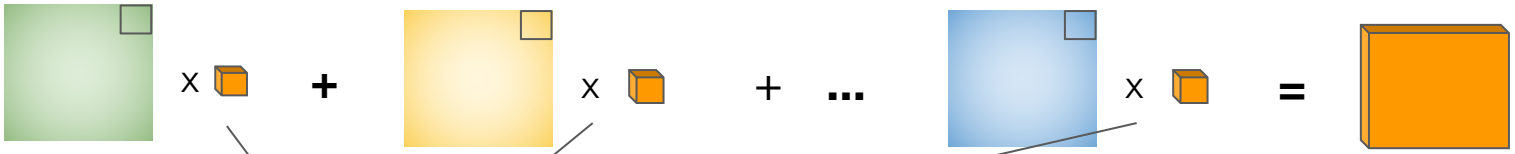
- For increasing the receptive field - Pooling and/or Strided Convolutions are common
  - But both reduce the resolution
- Dilated convolutions allow exponential expansion of the receptive field without loss of resolution
  - With same computation and memory costs
    - Practically, however, it is not equally ‘exciting’ for all problems!

Depthwise separable convolutions

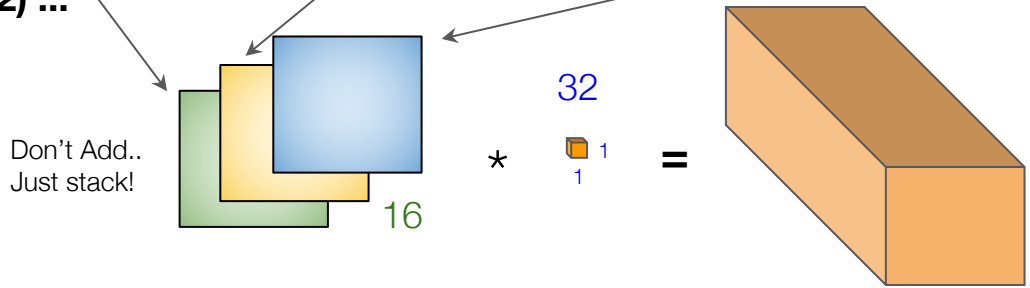


# of params = 32 x 3 x 3 x 16

Each convolution may be seen as ...



Depthwise Separable (32) ...



# of params = 16 x 3 x 3 + 32 x 1 x 1 x 16