

Week 9a:

Deep Learning and Convolutional Neural Networks

G6061: Fundamentals of Machine Learning [23/24]

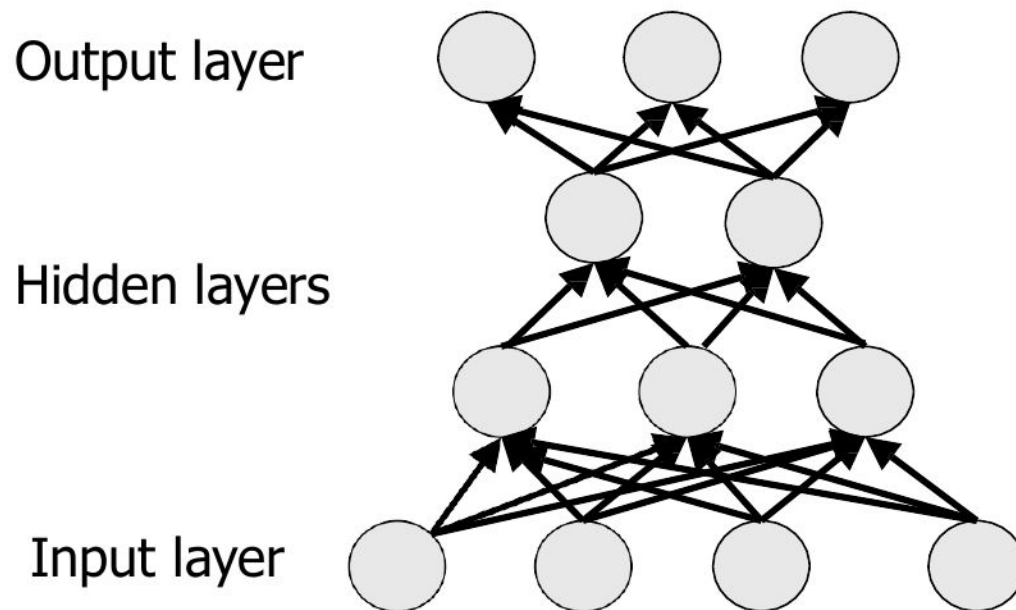
Dr. Johanna Senk

What *is* deep learning

- Deep learning is the set of multi-layer neural network based approaches for machine learning.
 - Multi-layer perceptrons are one form of deep learning.
 - The deep part comes from having intermediate “hidden” layers that form a key part of the processing.
 - This week we will talk about deep convolutional neural networks.

Deep architectures

Defintion: Deep architectures are composed of multiple levels of non-linear operations, such as neural nets with many hidden layers.



Examples of non-linear activations:

$$\tanh(x)$$

$$\sigma(x) = (1 + e^{-x})^{-1}$$

$$\max(0, x)$$



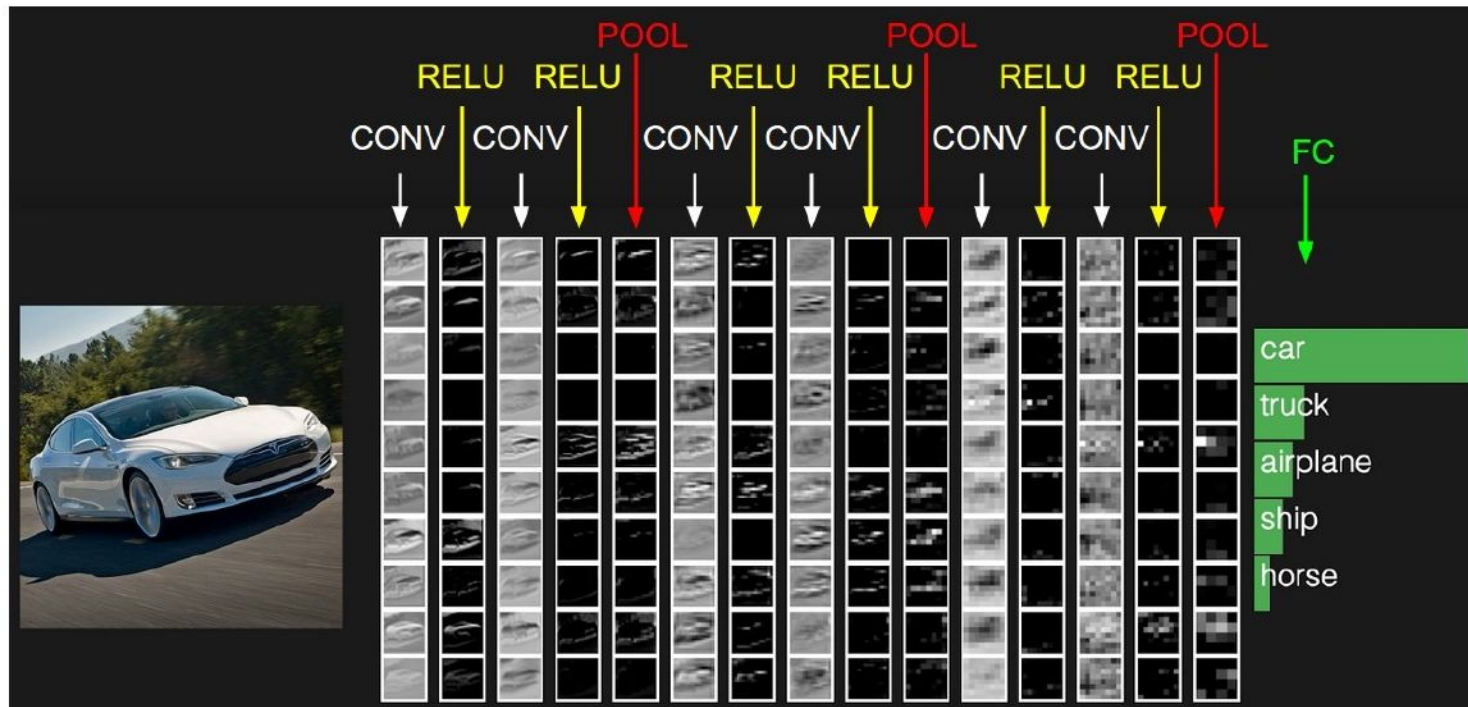
today

□ In practice, NN with multiple hid. layers work better than with a single hid. layer.

Goal of Deep architectures

Goal: Deep learning methods aim at

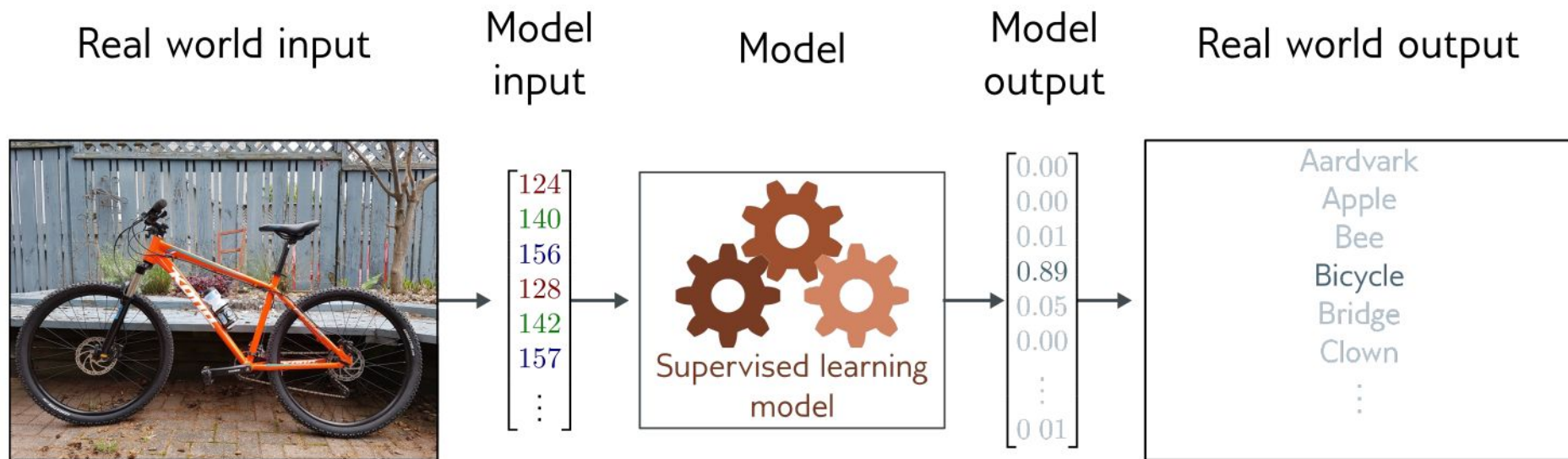
- learning feature hierarchies, where features from higher levels of the hierarchy are formed by lower level features
- learning guided by the machine learning task (e.g. classification)



Learning Outcomes

- Identify the building blocks of convolutional neural networks, and how they can be combined together to create flexible models.
- Understand the purpose and benefit of CNNs for image based tasks.

Image Classification



- Multiclass classification problem (discrete classes, >2 possible classes)
- Convolutional network

Object Detection

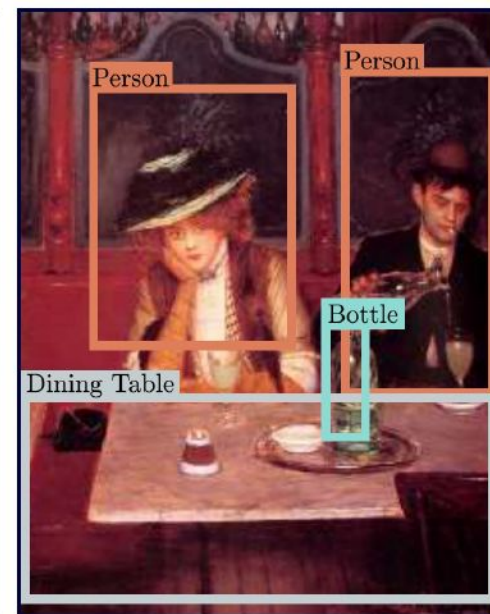
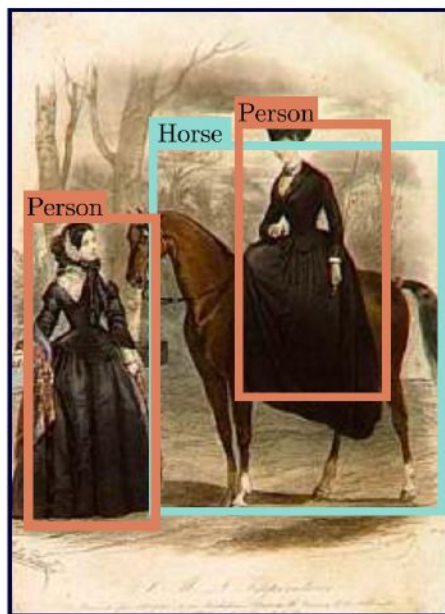
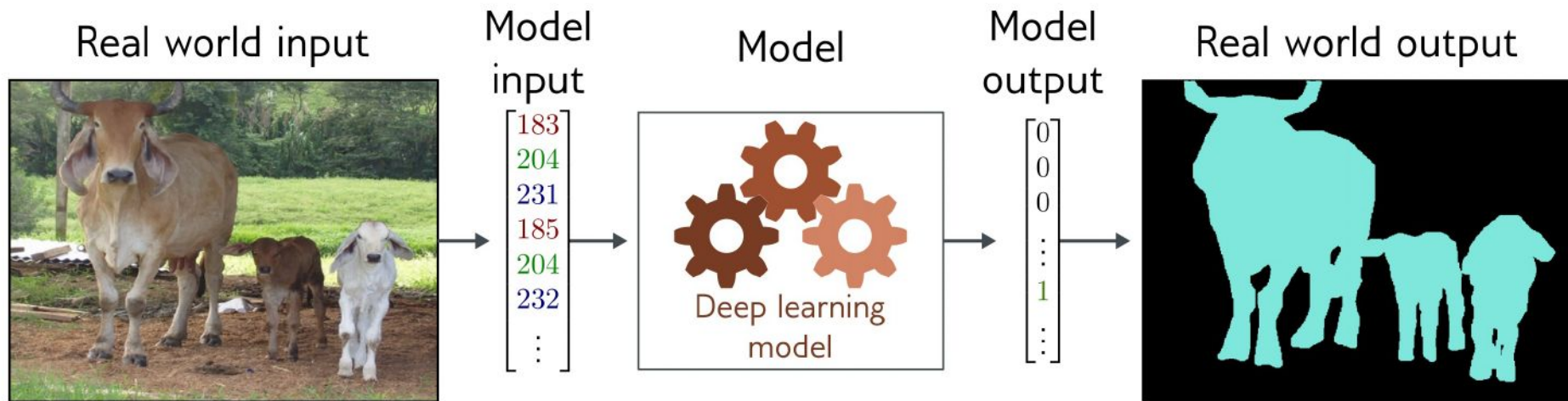


Image Segmentation



- Multivariate binary classification problem (many outputs, two discrete classes)
- Convolutional encoder-decoder network

What is special about image data?

- Dimensionality!
 - A 224x224 RGB image = 150,528 dimensions
 - Hidden layers generally larger than inputs
 - One hidden layer = 150,520x150,528 weights -- 22 billion
- Pixels have a spatial relationship - nearby pixels statistically related.
- We want consistent predictions under transformations.
 - Today we will introduce *convolutional* neural networks that recognize patterns of pixels in patches of the image.
 - These are only suitable for machine learning tasks on a regularly structured domain, e.g. images, video, audio.

Invariance

- A function $f[x]$ is invariant to a transformation $t[]$ if:

$$f[t[x]] = f[x]$$

- i.e., the function output is the same even after the transformation is applied.

Invariance Example

- e.g., Image classification
 - Image has been translated, but we want our classifier to give the same result



Equivariance

- A function $f[x]$ is equivariant to a transformation $t[]$ if:

$$f[t[x]] = t[f[x]]$$

- i.e., the output is transformed in the same way as the input

Equivariance Example

- e.g., Image segmentation
 - Image has been translated and we want segmentation to translate with it



Software Packages

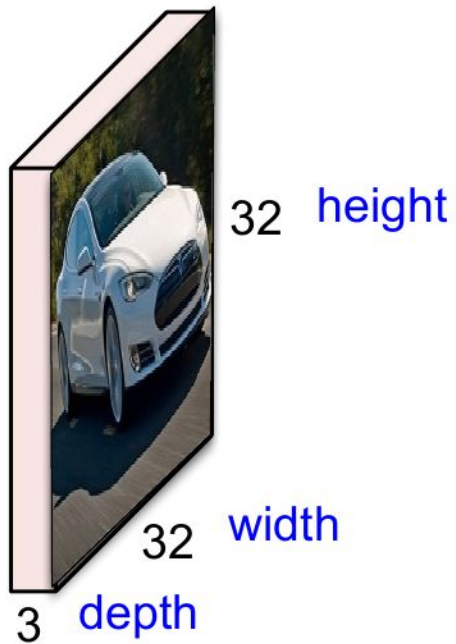
- Lots of useful software tools:
 - [TensorFlow](#), via the Keras interface
 - [PyTorch](#), with things like PyTorch lightning.
 - [SciKitLearn](#) also has some MLP things for [classification/regression](#), but these are much less flexible.
 - For the purposes of the lecture and lab, I will talk about TensorFlow, as it has excellent documentation and I think is easier for beginners.
 - PyTorch seems to be preferred by researchers at the moment.
 - If you run things on Colab, enable the TPU session for faster runtime.

Deep Convolutional Networks

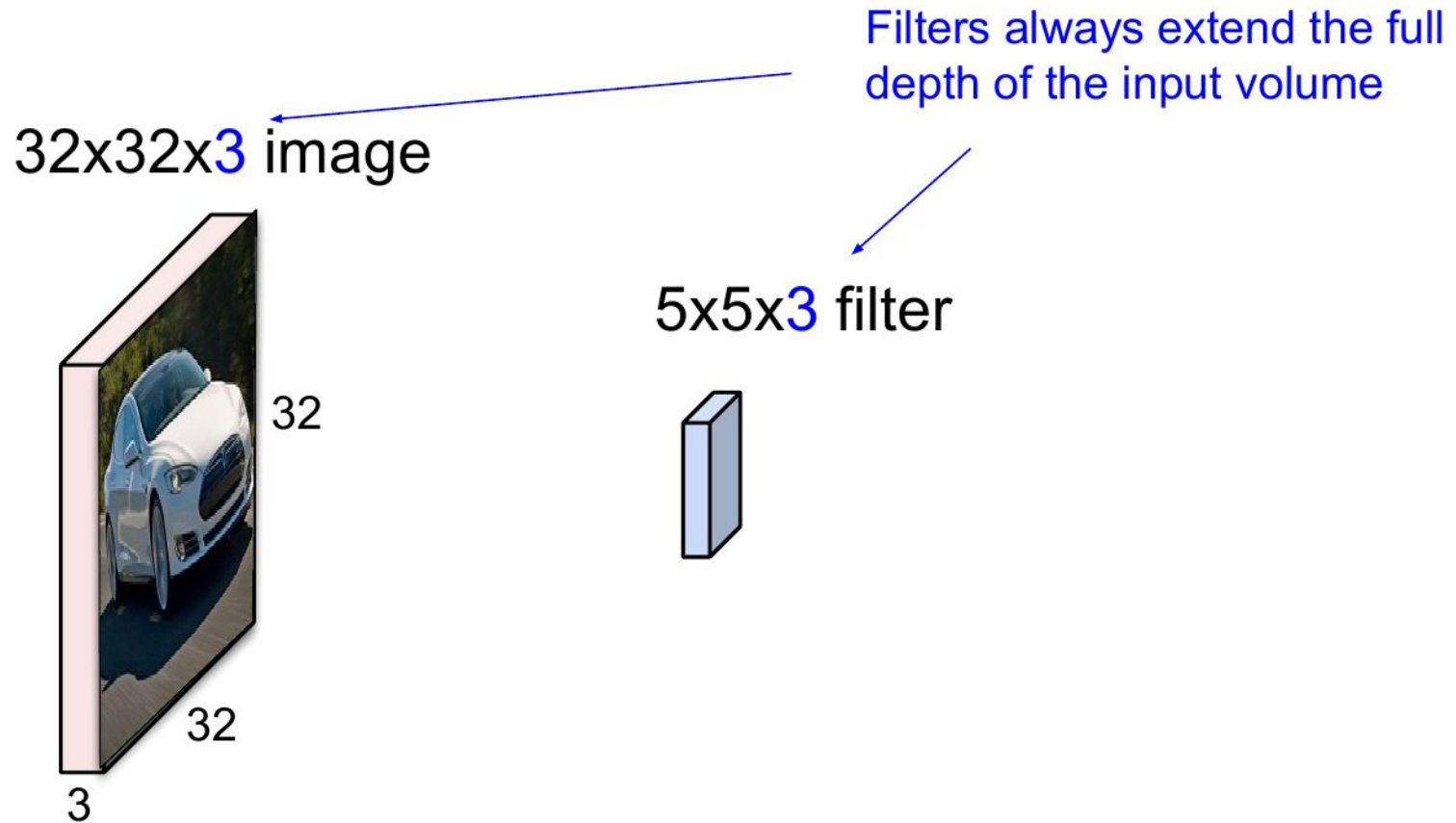
- ☐ Convolutional layer
- ☐ Non-linear activation function ReLU
- ☐ Max pooling layer
- ☐ Fully connected layer

Convolutional layer

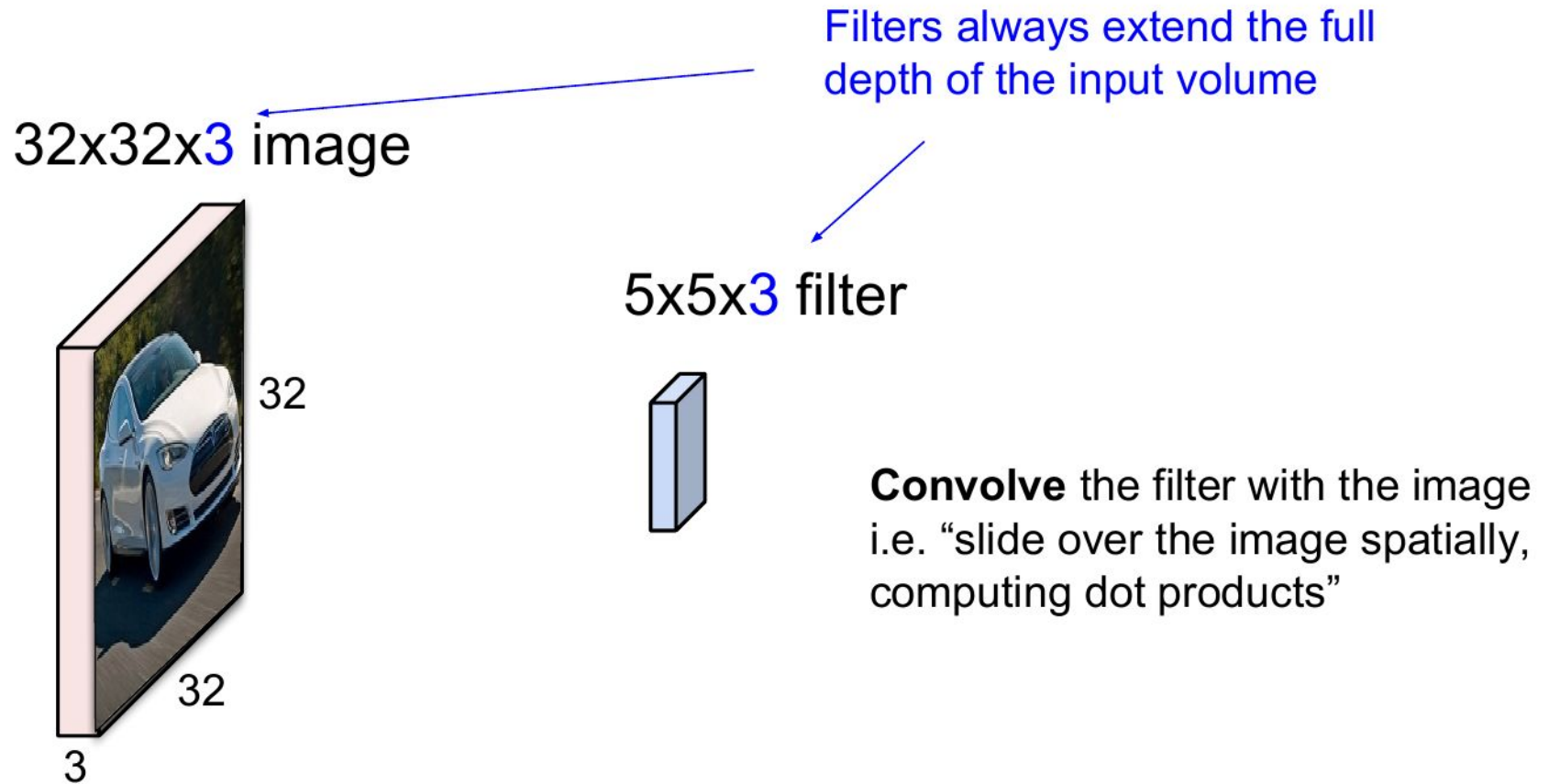
32x32x3 image



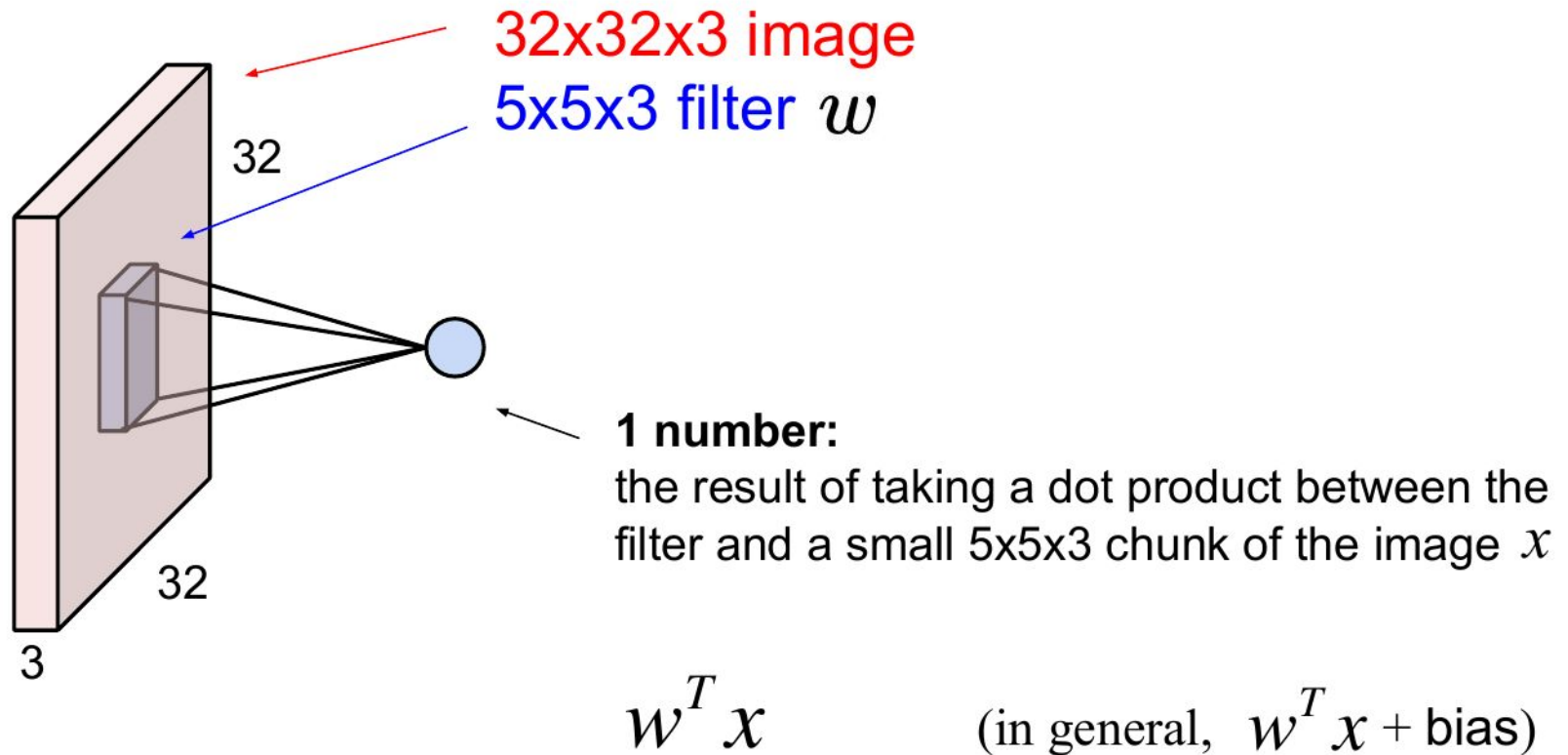
Convolutional layer



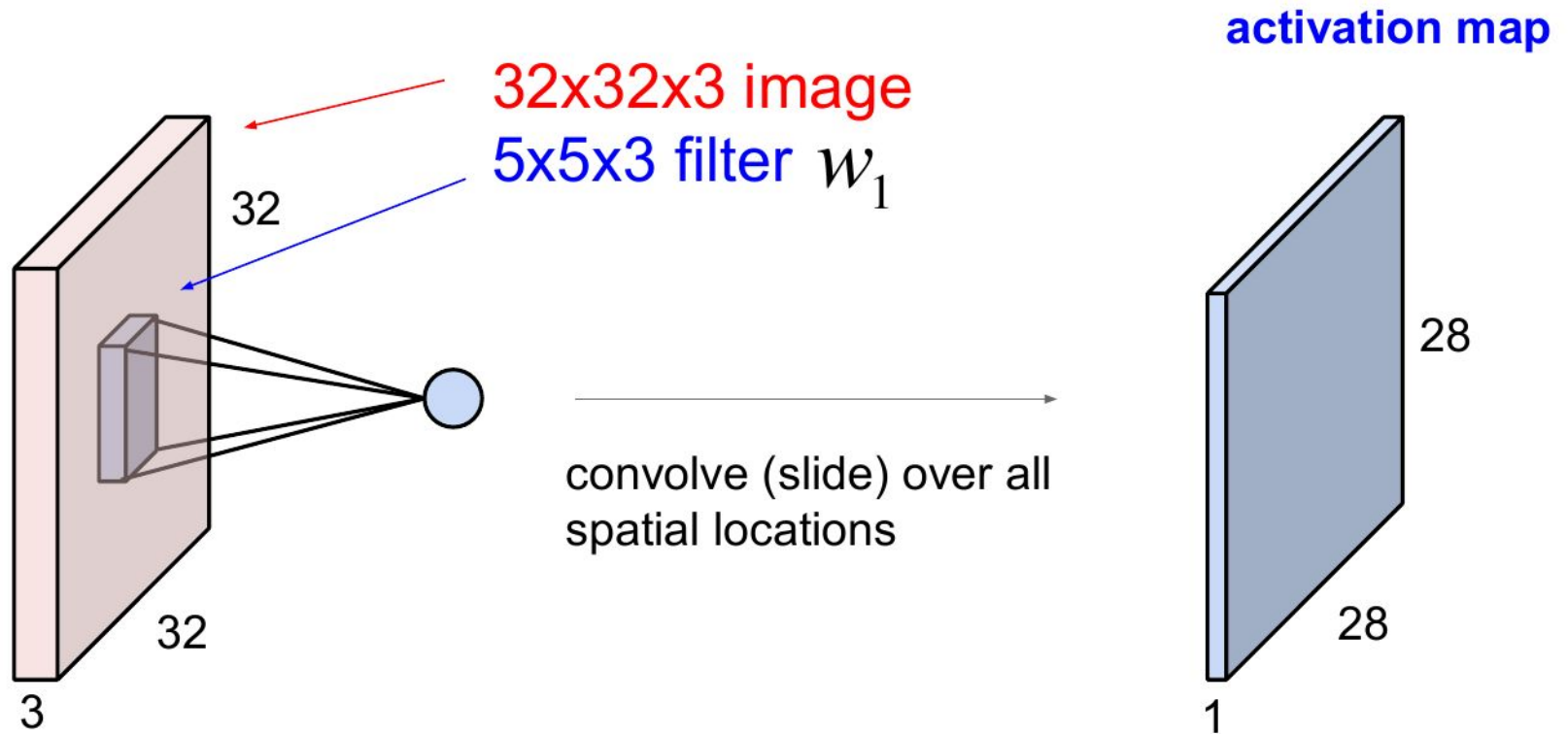
Convolutional layer



Convolutional layer

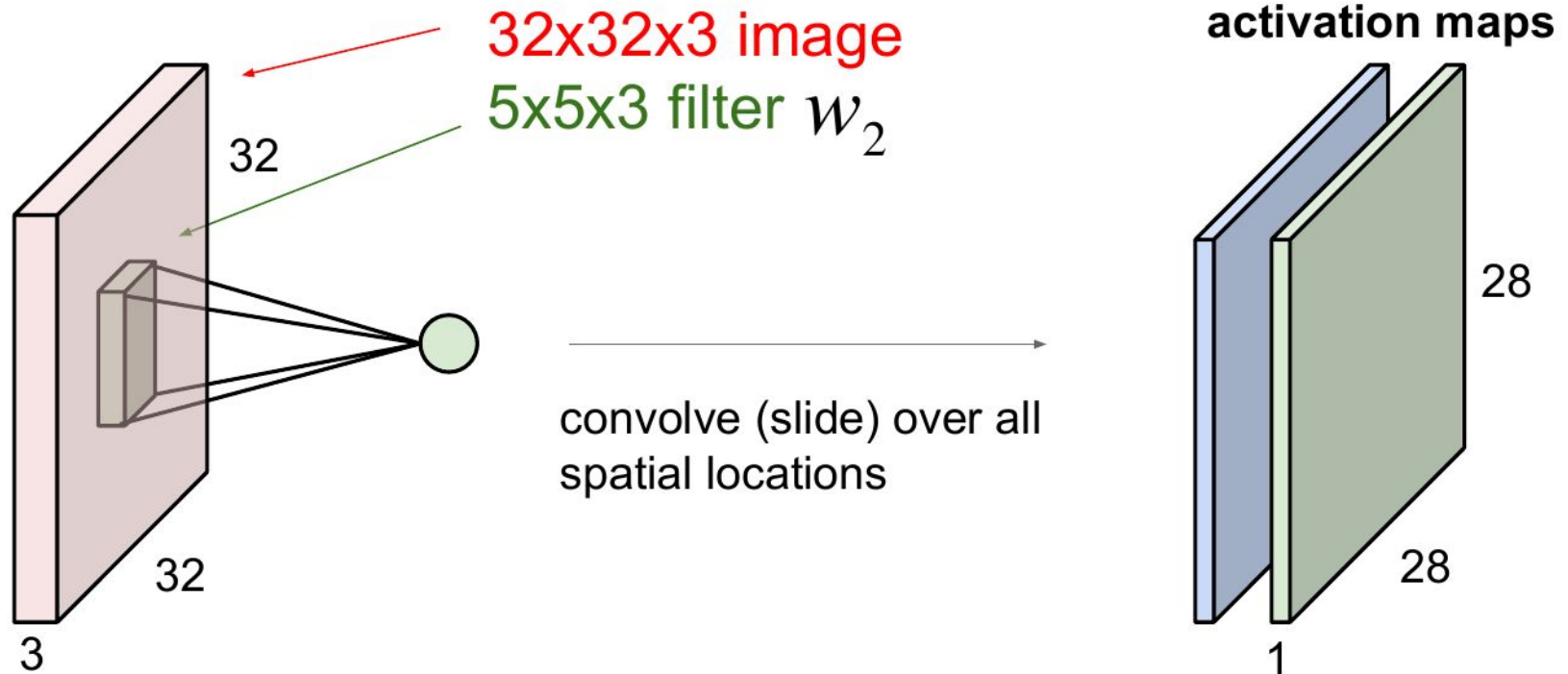


Convolutional layer



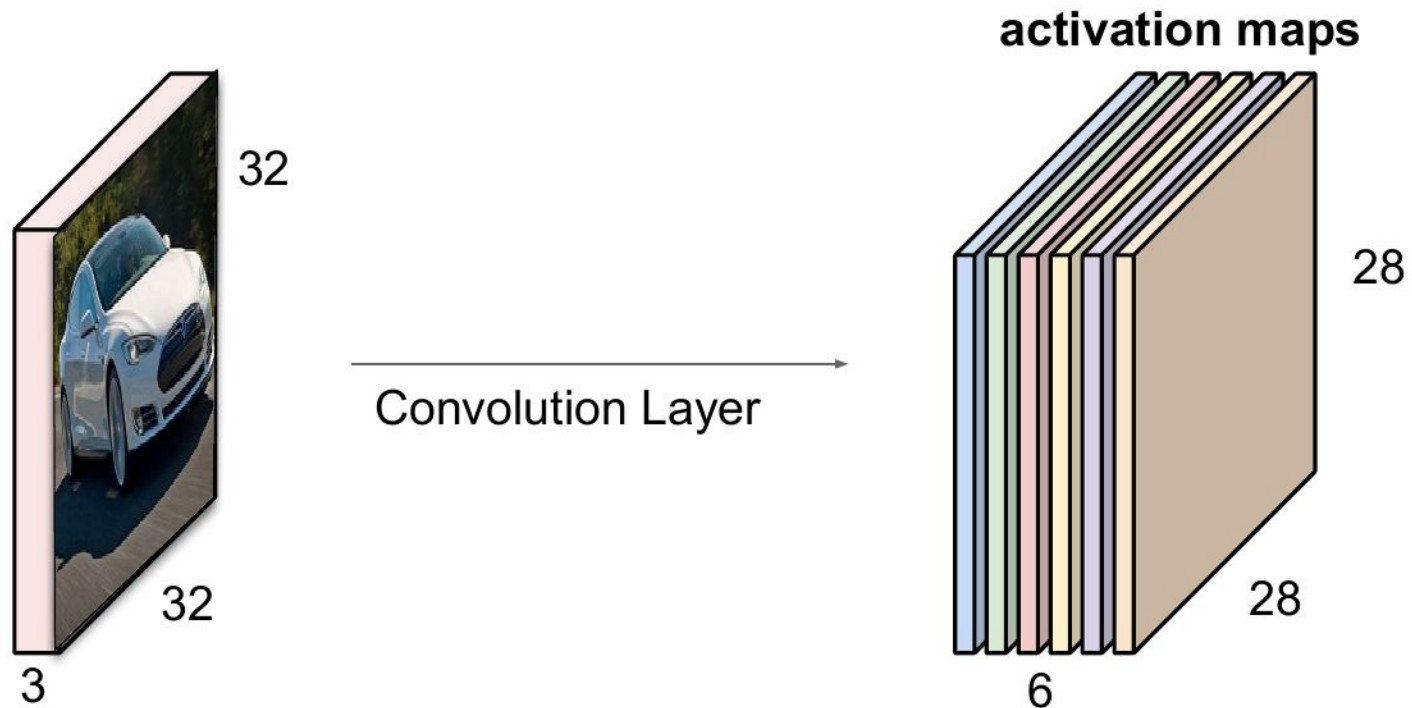
Convolutional layer

consider a second, **green** filter



Convolutional layer

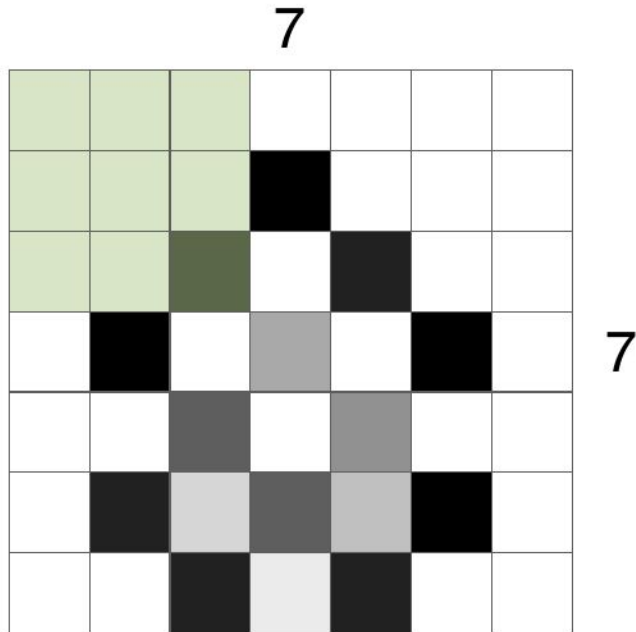
For example, if we use 6 $5 \times 5 \times 3$ filters, we'll get 6 separate activation maps:



We stack these up to get a “new image” of size $28 \times 28 \times 6$!

Spatial dimensions

A closer look at spatial dimensions

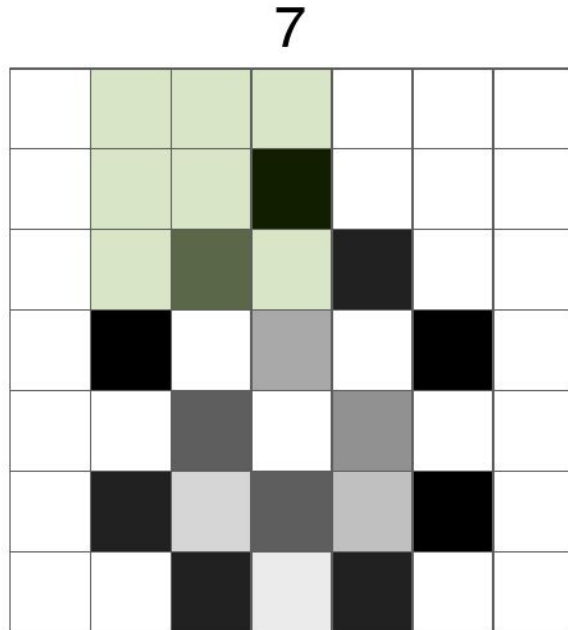


7x7x1 image

3x3x1 filter w

Spatial dimensions

A closer look at spatial dimensions



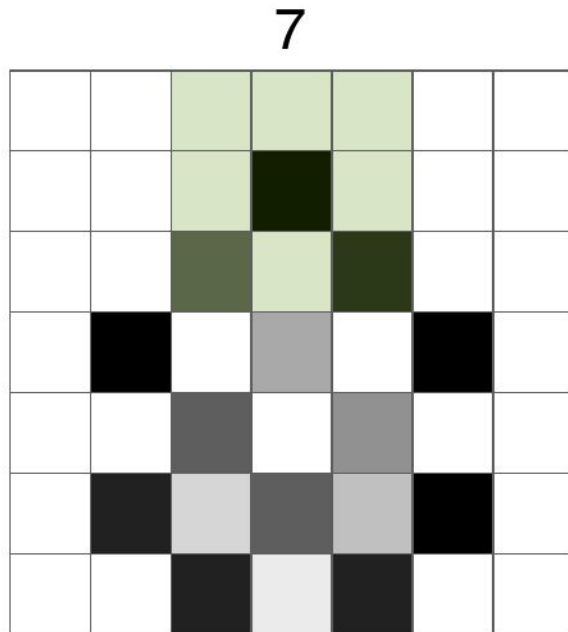
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, $S=1$

Spatial dimensions

A closer look at spatial dimensions



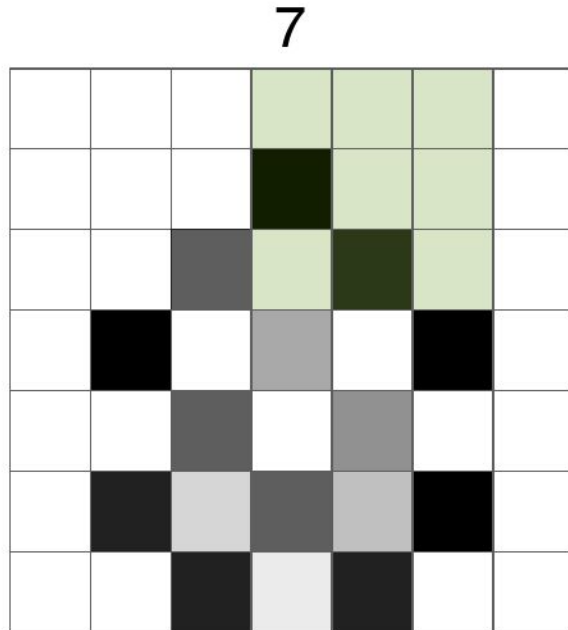
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, $S=1$

Spatial dimensions

A closer look at spatial dimensions



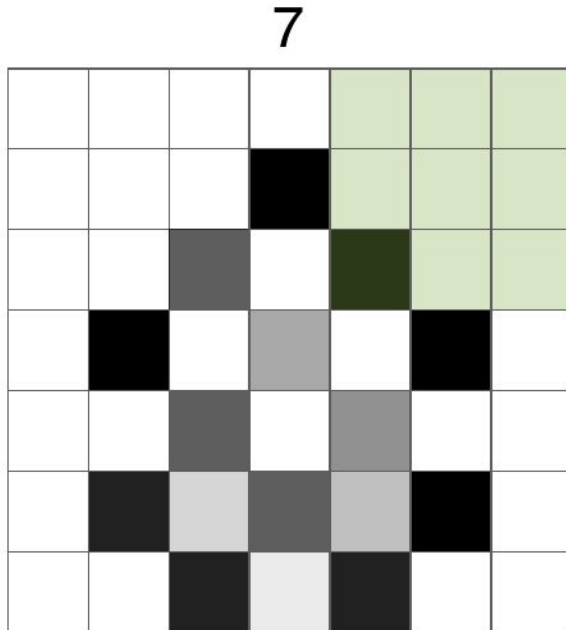
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, $S=1$

Spatial dimensions

A closer look at spatial dimensions



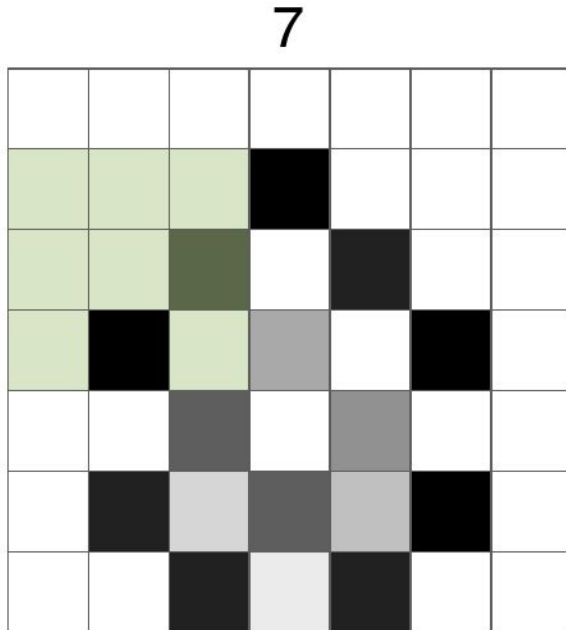
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, $S=1$

Spatial dimensions

A closer look at spatial dimensions



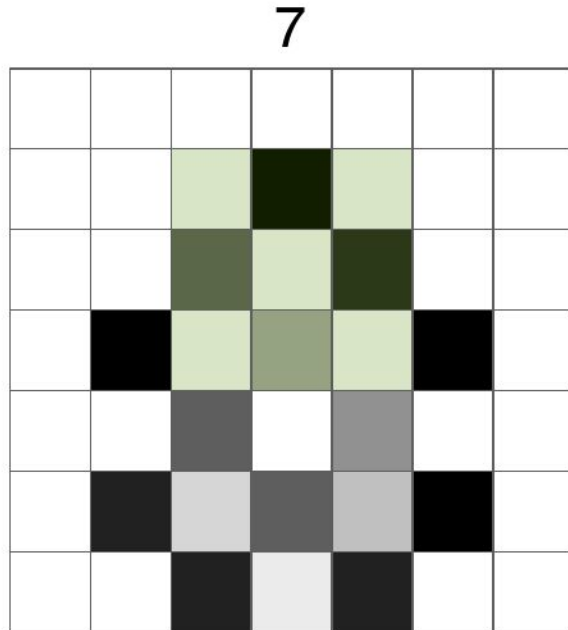
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, $S=1$

Spatial dimensions

A closer look at spatial dimensions



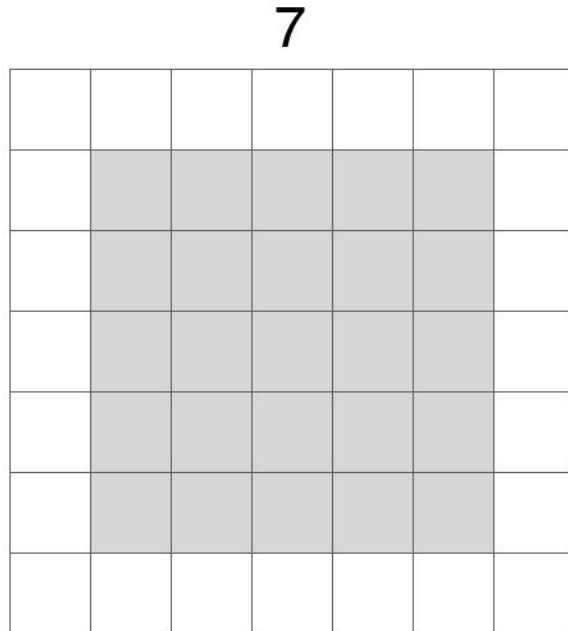
7x7x1 image

3x3x1 filter w

Slide over all locations using **stride 1** horizontally and vertically, $S=1$

Spatial dimensions

A closer look at spatial dimensions



7x7x1 image

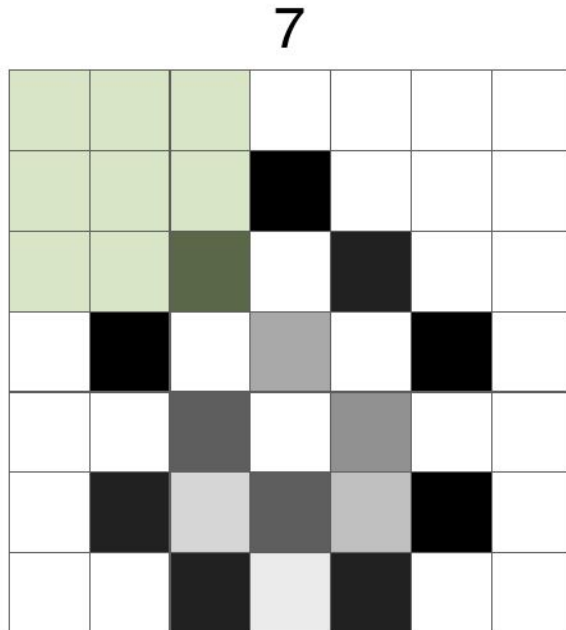
3x3x1 filter w

stride $S=1$

\Rightarrow **5x5 output**
activation map

Spatial dimensions

A closer look at spatial dimensions



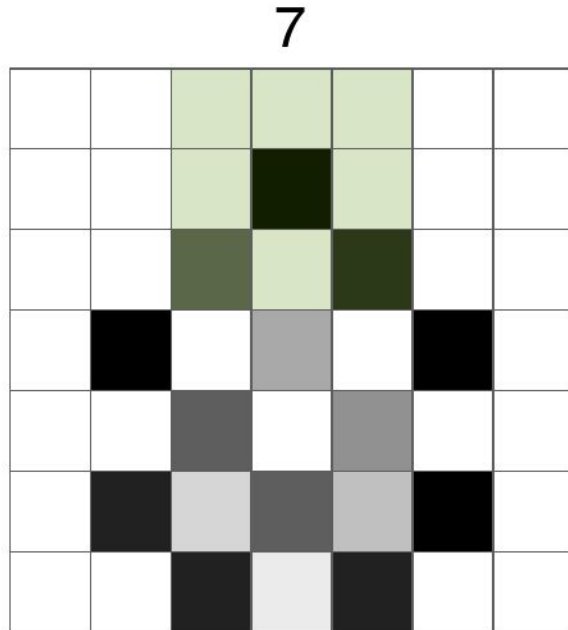
7x7x1 image

3x3x1 filter w

Slide over all locations **using stride 2**
horizontally and vertically, $S=2$

Spatial dimensions

A closer look at spatial dimensions



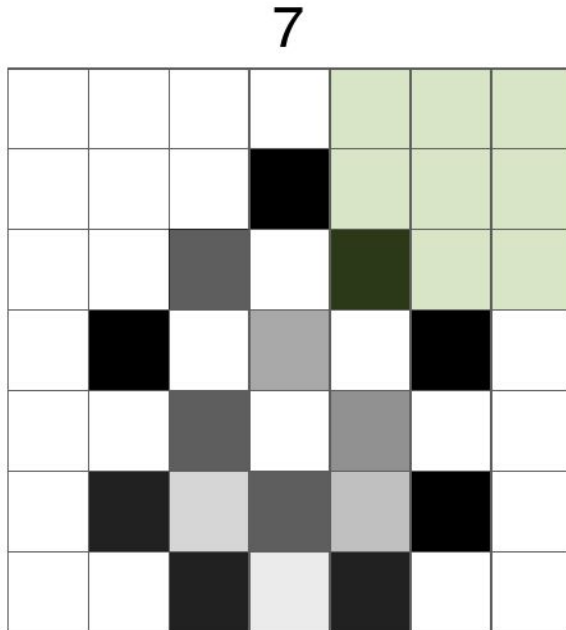
7x7x1 image

3x3x1 filter w

Slide over all locations **using stride 2**
horizontally and vertically, $S=2$

Spatial dimensions

A closer look at spatial dimensions



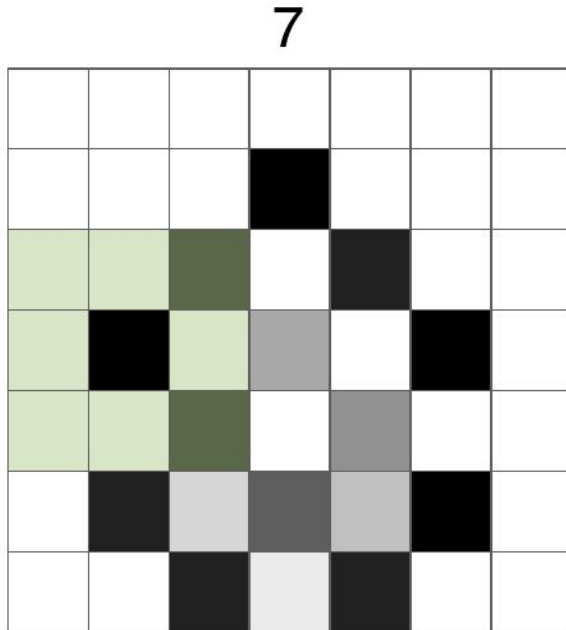
7x7x1 image

3x3x1 filter w

Slide over all locations **using stride 2**
horizontally and vertically, $S=2$

Spatial dimensions

A closer look at spatial dimensions



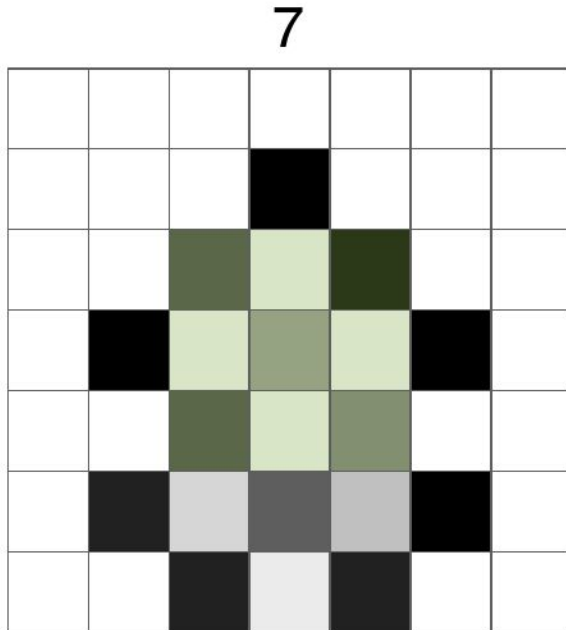
7x7x1 image

3x3x1 filter w

Slide over all locations **using stride 2**
horizontally and vertically, $S=2$

Spatial dimensions

A closer look at spatial dimensions



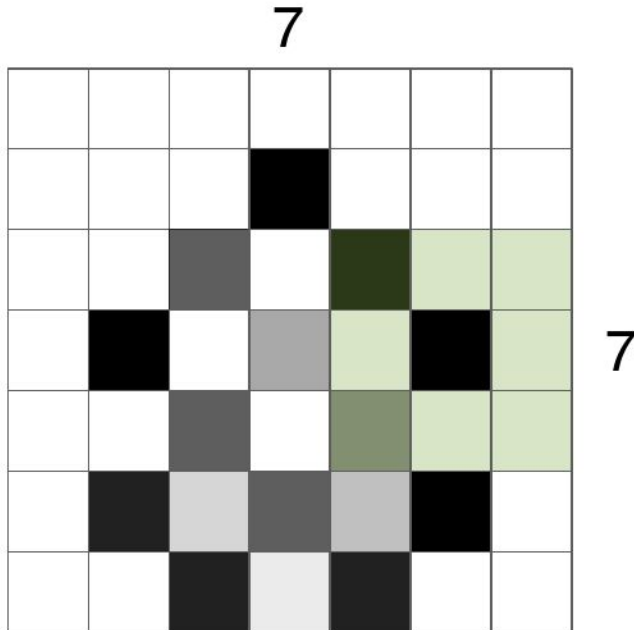
7x7x1 image

3x3x1 filter w

Slide over all locations **using stride 2**
horizontally and vertically, $S=2$

Spatial dimensions

A closer look at spatial dimensions



7x7x1 image

3x3x1 filter w

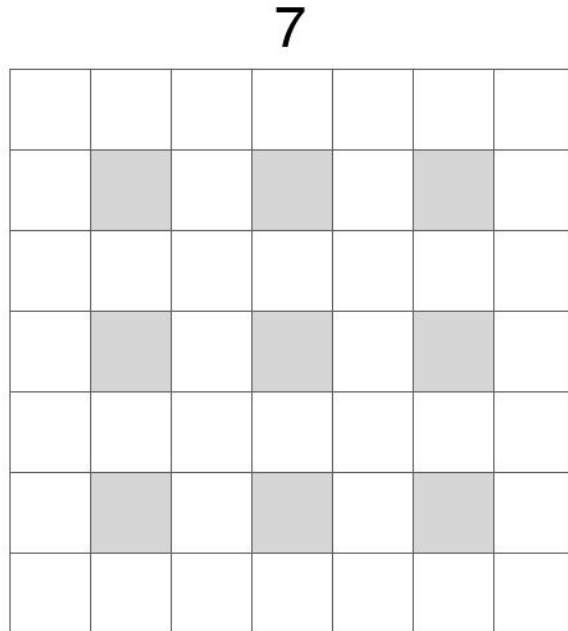
Slide over all locations **using stride 2**
horizontally and vertically, $S=2$

...

=> ? output

Spatial dimensions

A closer look at spatial dimensions

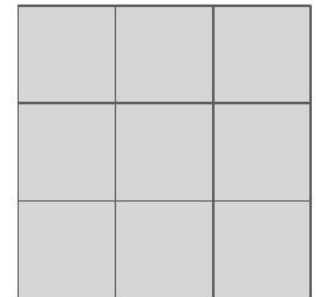


7x7x1 image

3x3x1 filter w

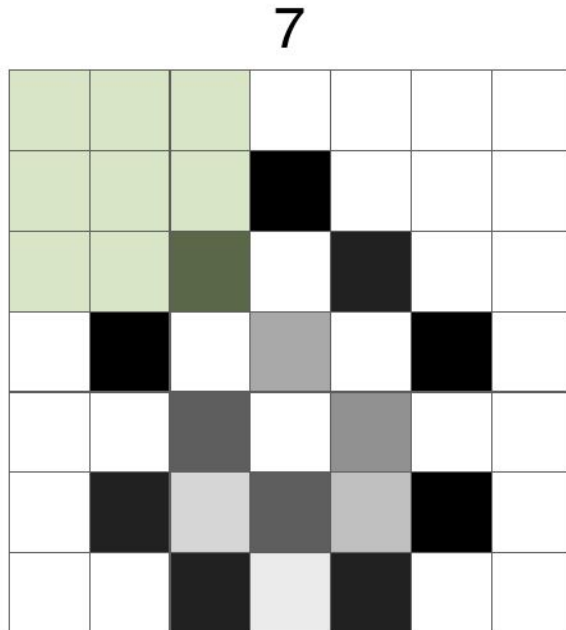
stride $S=2$

\Rightarrow **3x3 output**
activation map



Spatial dimensions

A closer look at spatial dimensions



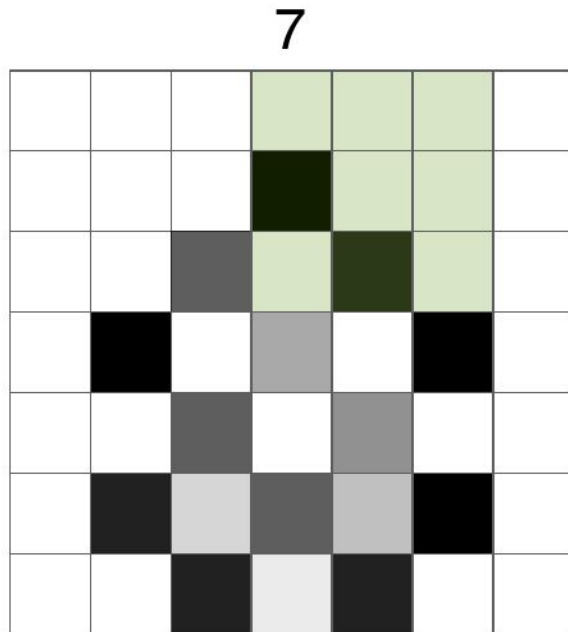
7x7x1 image

3x3x1 filter w

stride $S=3$

Spatial dimensions

A closer look at spatial dimensions



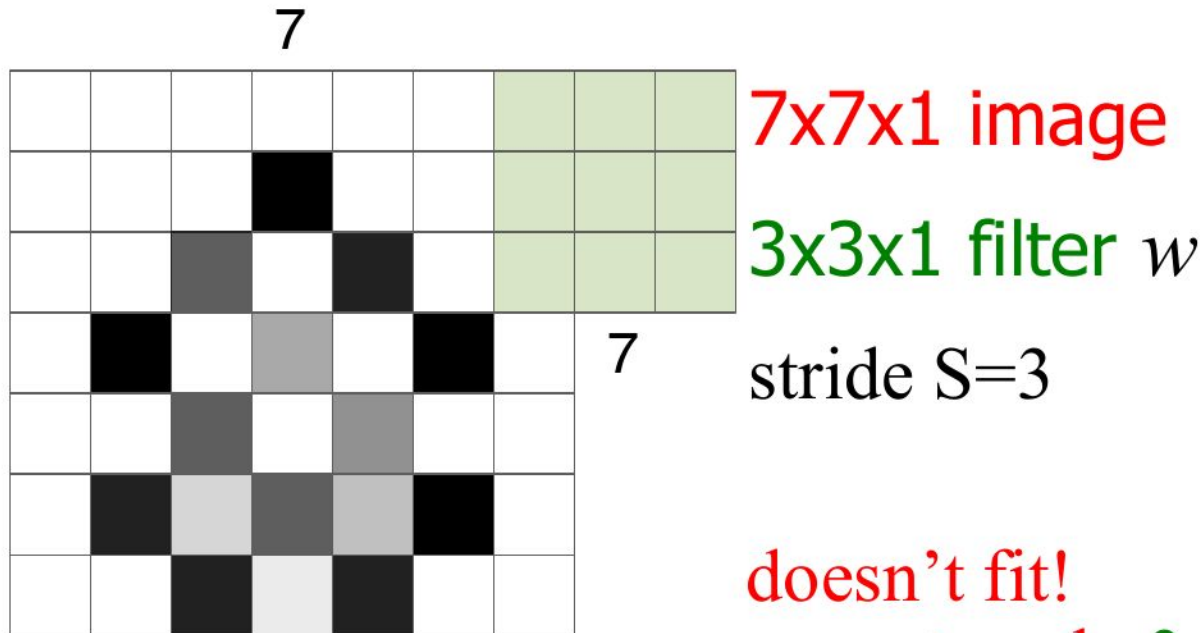
7x7x1 image

3x3x1 filter w

stride $S=3$

Spatial dimensions

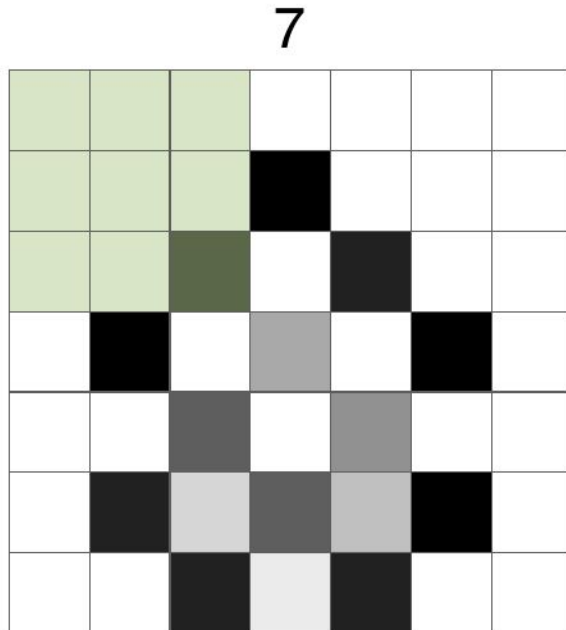
A closer look at spatial dimensions



doesn't fit!
cannot apply 3x3x1 filter on
7x7x1 image with stride 3

Spatial dimensions

A closer look at spatial dimensions



7x7x1 image

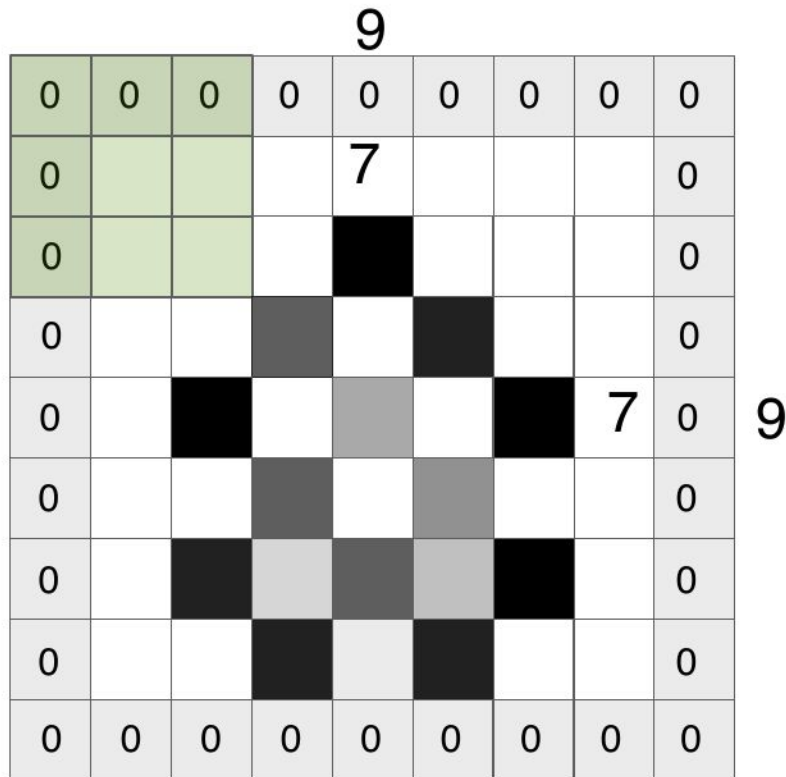
3x3x1 filter w

stride $S=3$

?

Spatial dimensions

- ❑ Add zero padding around the border



7x7x1 image

3x3x1 filter w

stride $S=3$

padding = 1

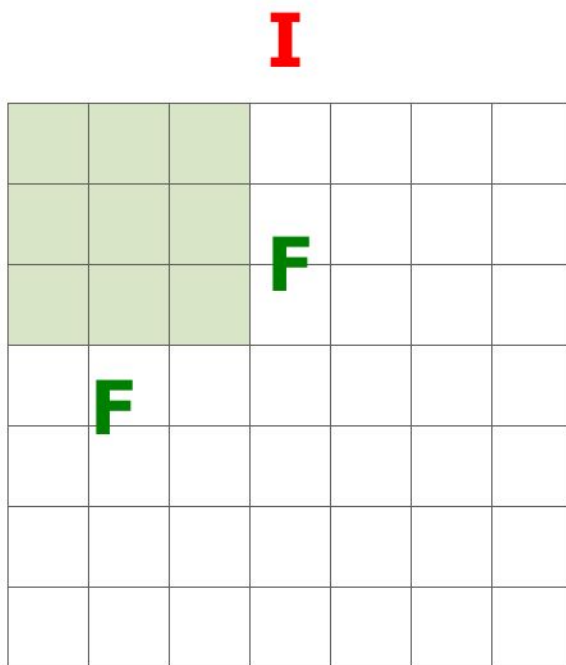
⇒ **3x3 output**

activation
map

Spatial dimensions

□ Spatial dimension of the output

$$\frac{I - F + 2P}{S} + 1$$



Ix**I**x**d** input

Fx**F**x**d** filter w

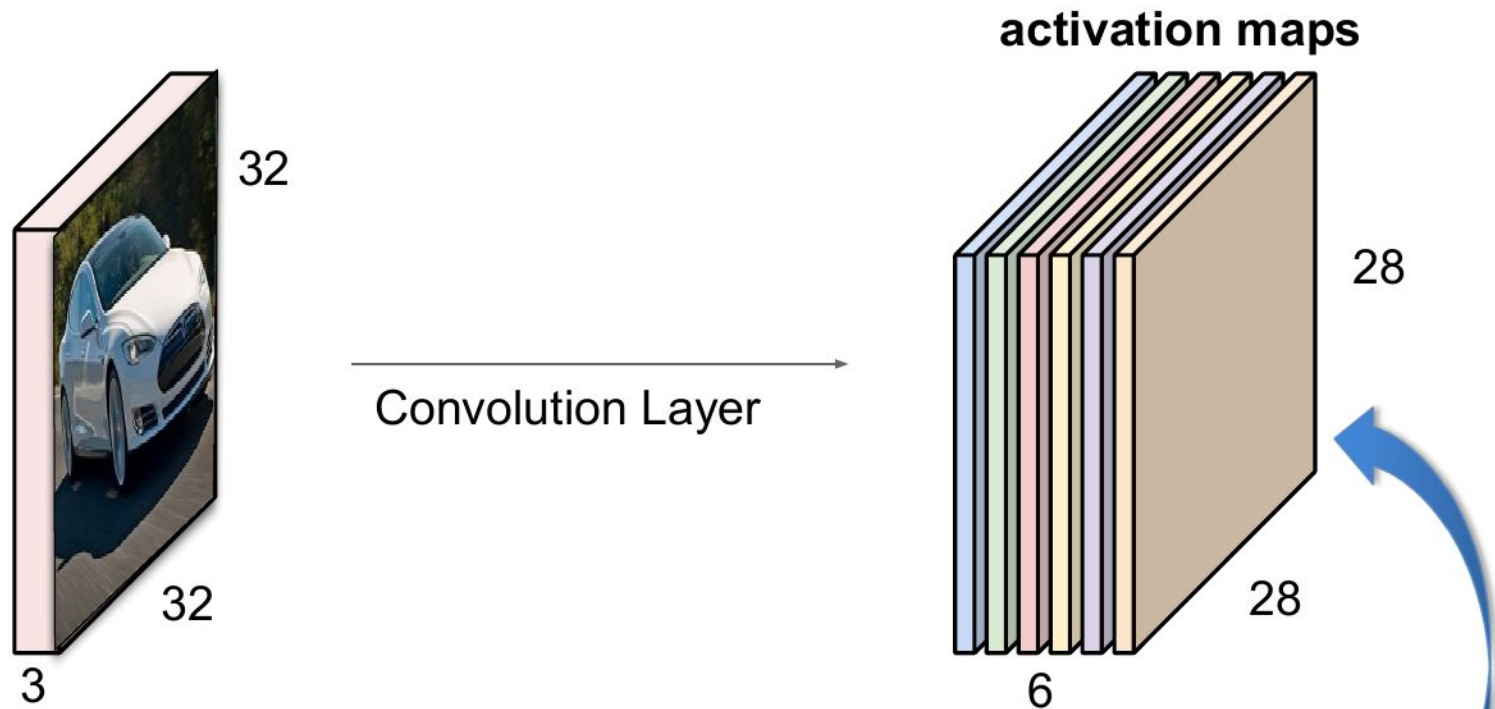
stride **S**

padding **P**

□ If width **I**_{width} and height **I**_{height} of the input differ, this formula is applied independently for **I**_{width} and **I**_{height} .

Back to convolutional layer

For example, if we use 6 $5 \times 5 \times 3$ filters, we'll get 6 separate activation maps:



We stack these up to get a “new image” of size $28 \times 28 \times 6$!

$$\frac{I - F + 2P}{S} + 1 \quad \text{Spatial dimension:} \quad \frac{32 - 5 + 2 \cdot 0}{1} + 1 = 28$$

Deep Convolutional Networks CNNs

- CNNs have much fewer connections to the previous layer (and therefore parameters)
- Inductive bias towards spatial patterns.
- Convolutions allow us to learn spatial filters that are applied everywhere in the image
 - the result is **equivariant** to translation.
- CNNs typically have more than five hidden layers (a number of layers which makes fully-connected neural networks almost impossible to train properly when initialized randomly)

LeNet, 1998 LeCun Y, Bottou L, Bengio Y, Haffner P: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE 1998

AlexNet, 2012 Krizhevsky A, Sutskever I, Hinton G: ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

Convolutional layers in Code

```
from tensorflow import keras
# Build a simple model that takes a 32x32 RGB im
img_input = keras.Input(shape=(32, 32, 3))
# create a convolutional layer as an object
conv_layer = keras.layers.Conv2D(n_channels,
filter_width, activation=activation_fn,
padding={'valid', 'same'},
strides=(x_stride, y_stride))
# Pass the image through the convolution
img_output = conv_layer(img_input)
# Create a model (using the keras functional API)
model = keras.Model(img_input, img_output)
```

Convolutional layer: summary

❑ Accepts an input of size **IxIx d**

❑ Requires four specifications:

- Number of filters **K**
- Filter size **Fx Fx d**
- The stride **S**
- Padding **P**

❑ Outputs a volume of size **OxOx K**, where $O = \frac{I - F + 2P}{S} + 1$

❑ In the output volume, the i-th activation map is the result of a convolution of the i-th filter over the input with a stride **S** and padding **P**.

❑ **Local connectivity and parameter sharing:**

each convolutional layer has **(FxFx d)xK** weight parameters to be learned (the fully connected layer would have **IxIx d x O x O x K** par.)

Often in practice:

K is power of 2, e.g. 32, 64, 128

F = 3, **S**=1, **P**=1

F = 5, **S**=1, **P**=2

F = 5, **S**=2, **P** is set accordingly

F = 1, **S**=1, **P**=0

Convolutional layer: summary

- ❑ Convolutions are interesting for a number of reasons
 - ❑ They are trivial to parallelise, each output pixel is independent of the others. This is why they can be made to run very fast on GPUs.
 - ❑ Because they only perform operations in a local neighbourhood they require far fewer parameters than a full matrix multiplication. E.g. convolutional layer has $(F \times F \times d) \times K$ weight parameters to be learned (the fully connected layer would have $I \times I \times d \times O \times O \times K$ par.)
 - ❑ Convolutions can be thought of as a big matrix multiplication, but where the matrix has lots of 0s and repeating patterns.

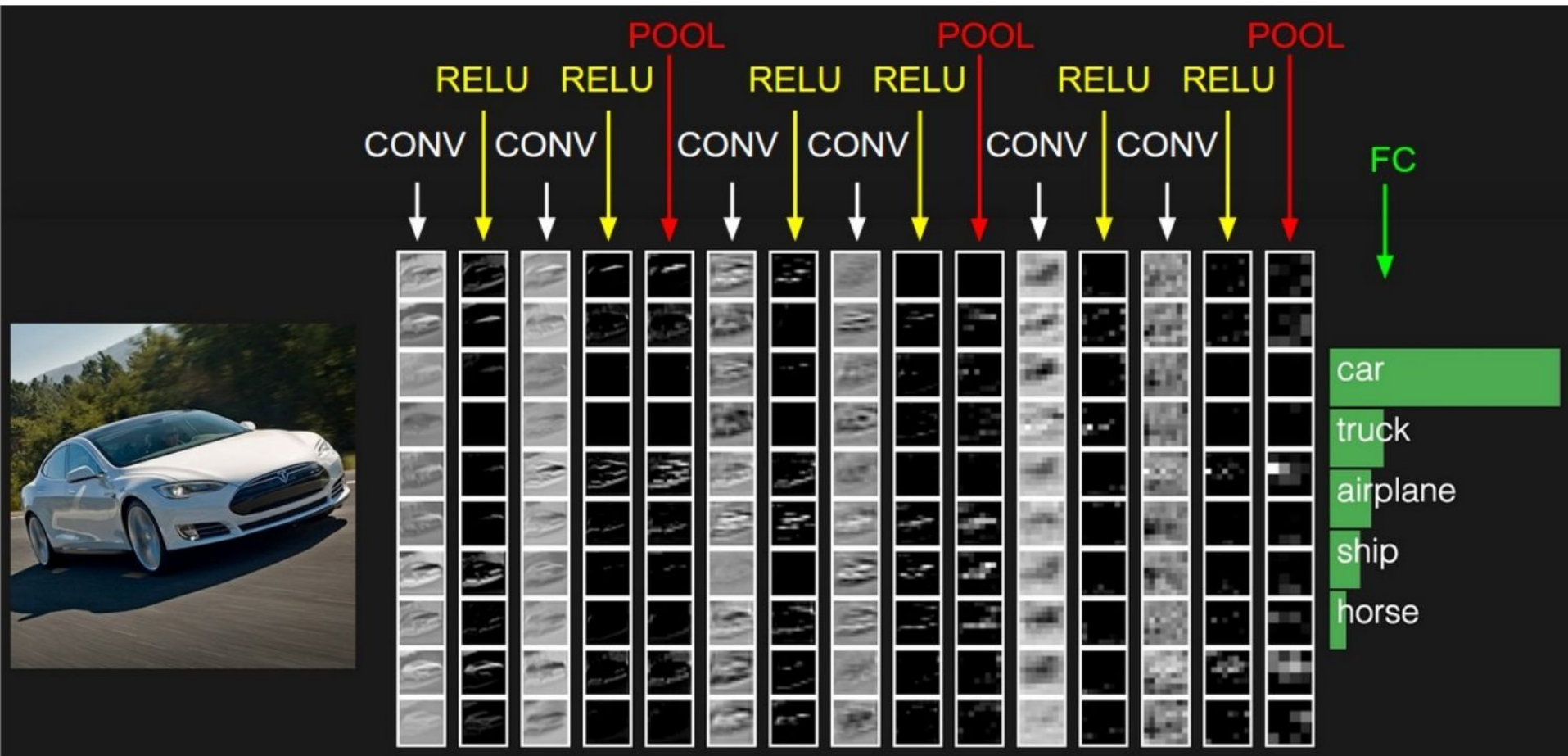
Deep Convolutional Networks

- ☒ Convolutional layer
- ☐ Non-linear activation function (ReLU)
- ☐ Max pooling layer
- ☐ Fully connected layer

Non-linearities

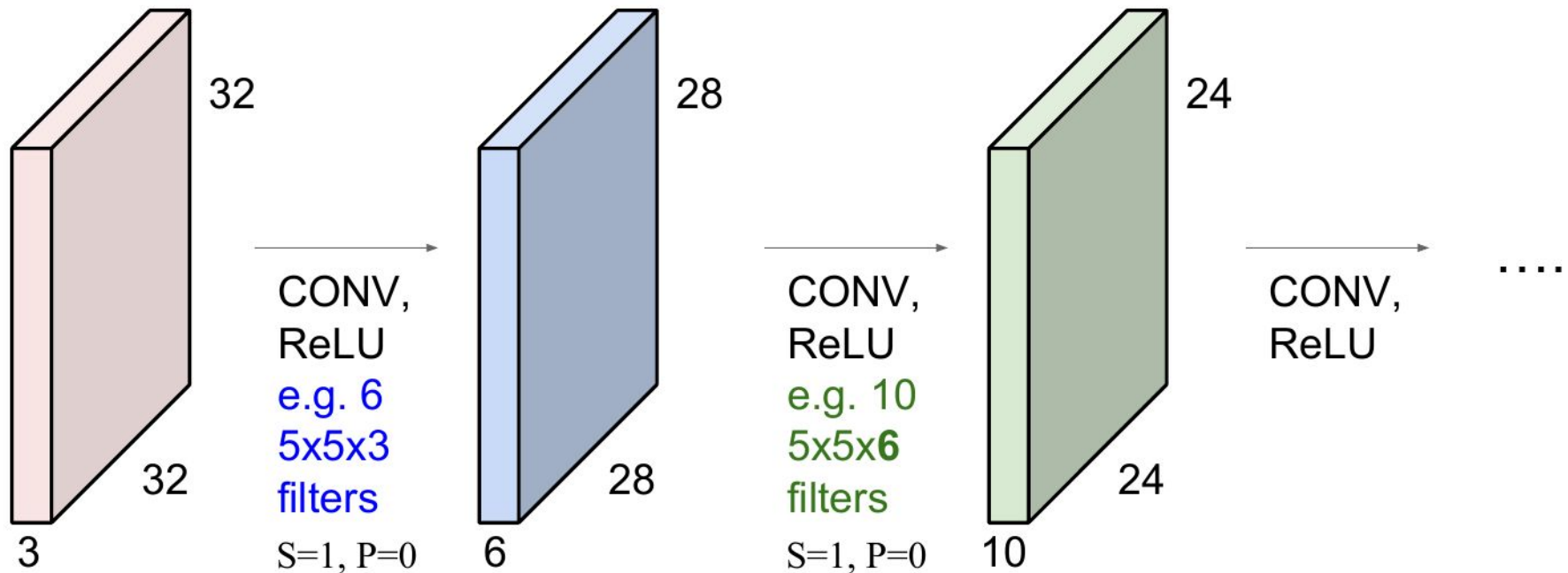
- ❑ Why do we need non-linear activation functions?
- ❑ Where do they go?

Where are the non-linearities?



Where is ReLU?

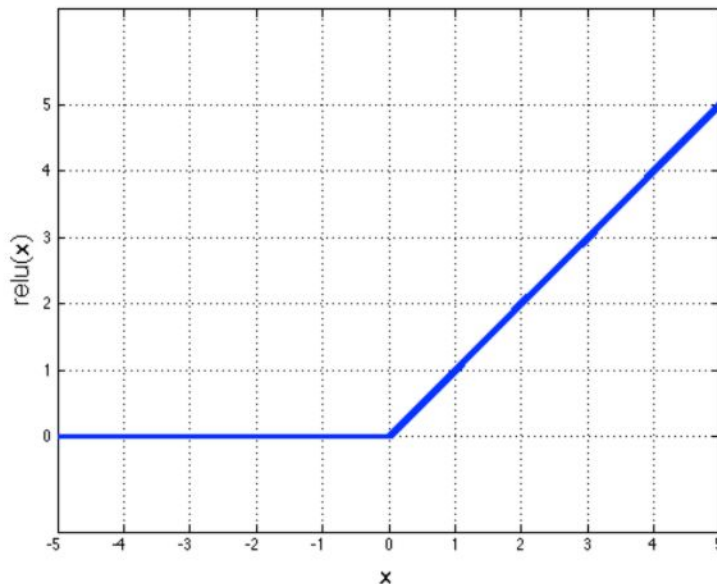
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Rectified Linear Unit, ReLU

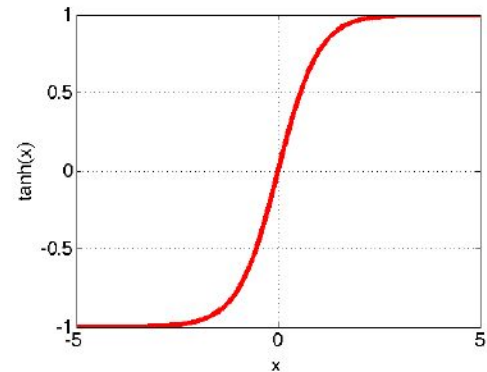
- ❑ Non-linear activation functions are applied per-element
- ❑ Rectified linear unit (ReLU):

- $\max(0, x)$
- makes learning faster (in practice x6)
- avoids saturation issues (unlike sigmoid, tanh)
- simplifies training with backpropagation
- preferred option (works well)

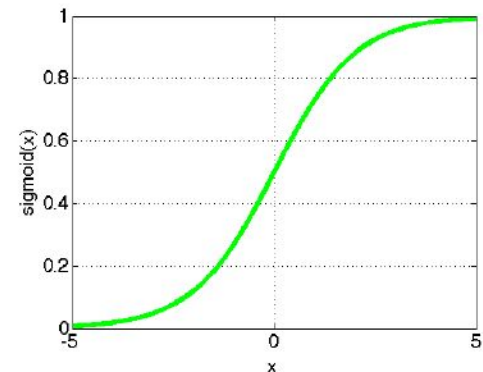


Other examples:

$\tanh(x)$

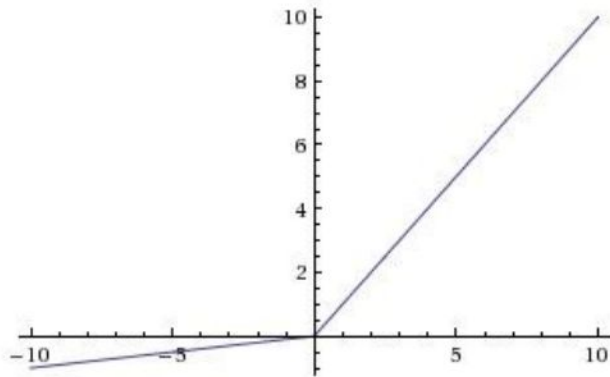


$\text{sigmoid}(x) = (1 + e^{-x})^{-1}$



[Activation functions: extra]

❑ State-of-the-art



Leaky ReLU

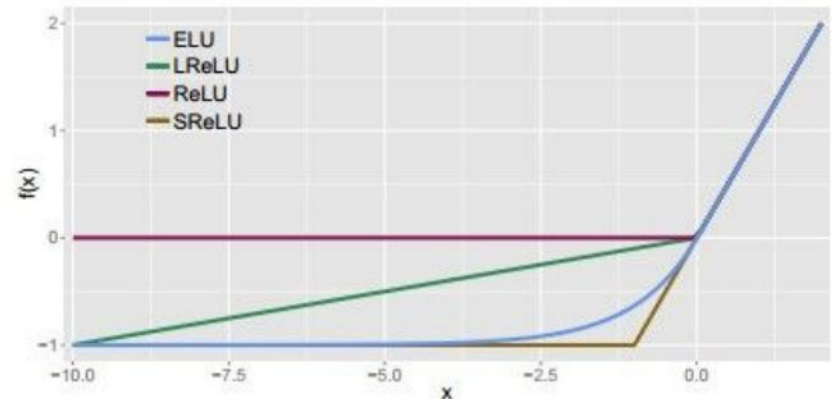
$$f(x) = \max(0.01x, x)$$

Parametric Rectifier (PReLU)

$$f(x) = \max(\alpha x, x)$$

[Mass et al., 2013]

[He et al., 2015]



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

Exponential Linear Units (ELU)

[Clevert et al., 2015]

Activations Code

```
from tensorflow import keras
# Build a simple model that takes a 32x32 RGB im
img_input = keras.Input(shape=(32, 32, 3))
# create a convolutional layer and pass the image
conv1_op = keras.layers.Conv2D(...,
activation='relu')(img_input)
# you can also create a keras activation layer
keras.layers.Activation('relu')
# Create a model (using the keras functional API)
model = keras.Model(img_input, resid_op)
```

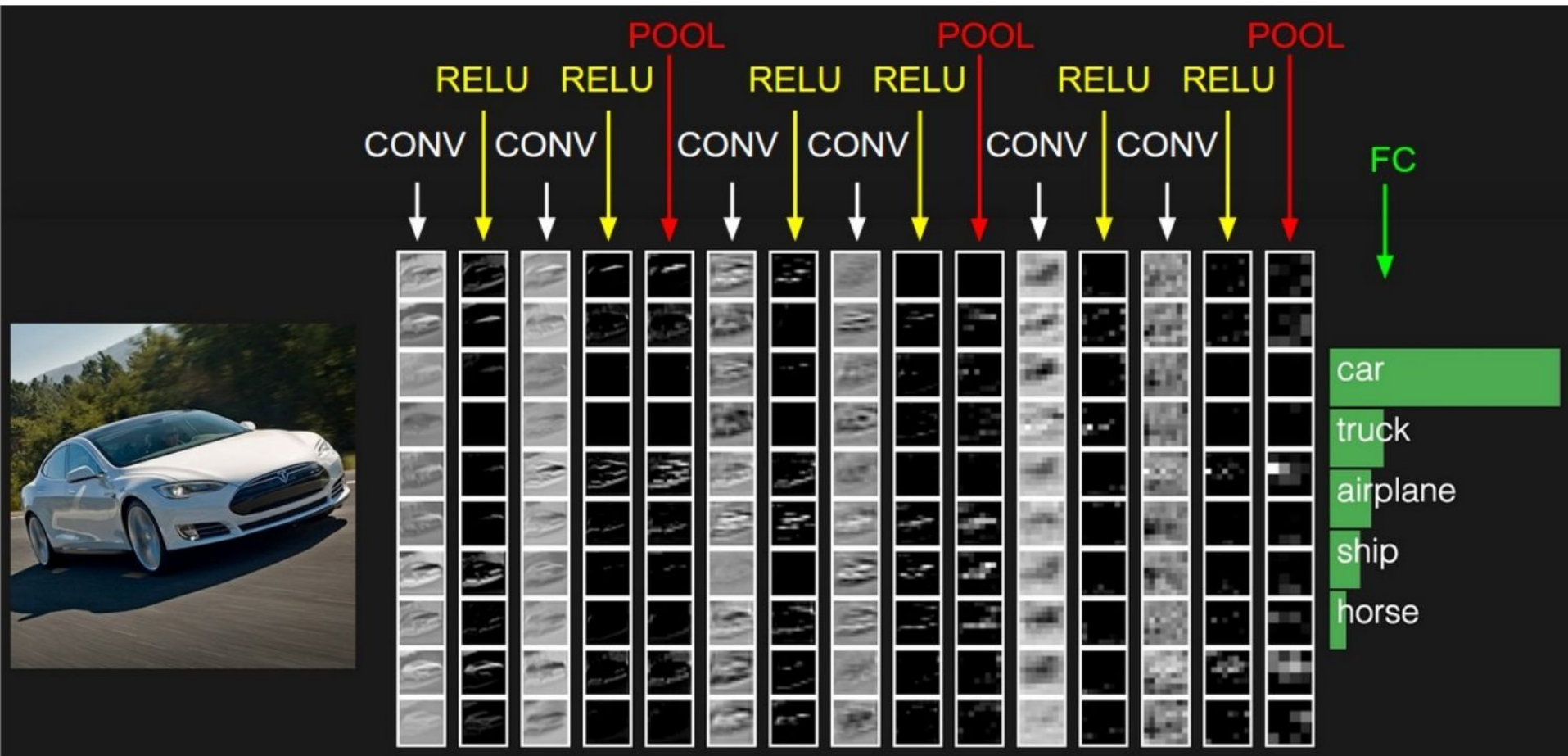

Deep Convolutional Networks

- ☒ Convolutional layer
- ☒ Non-linear activation function ReLU
- ☒ Max pooling layer
- ☐ Fully connected layer

Dimensionality

- ❑ As we apply convolutional filters we increase the number of numbers at each pixel.
- ❑ This means we are increasing the dimensionality (complexity) of our already high dimensional data!
- ❑ What could we do to reduce the dimensionality?

Where is the pooling?

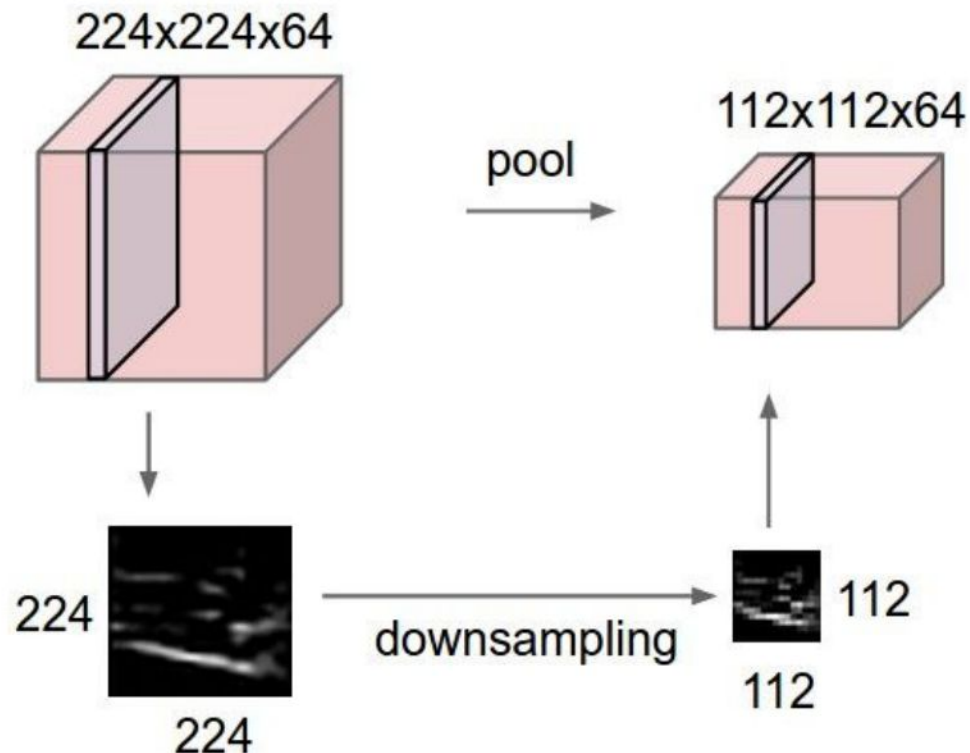


Two more layers to go: pooling and fully connected layers ☺

Spatial pooling

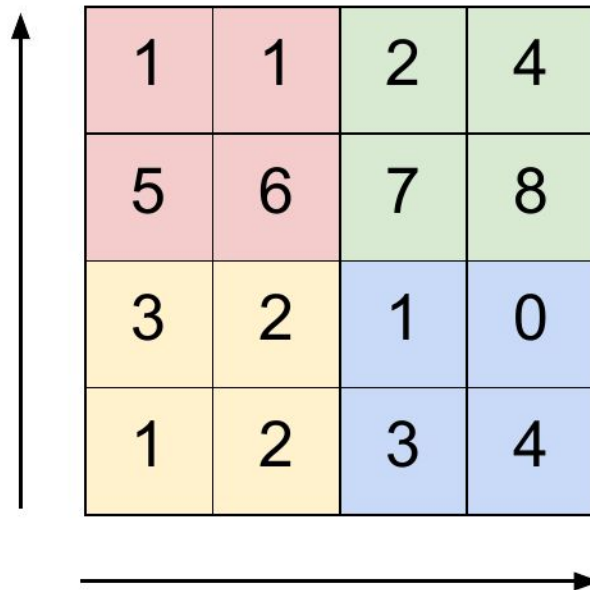
❑ Pooling layer:

- ❑ Makes the representations smaller (downsampling)
- ❑ Operates over each activation map (channel) independently
- ❑ Role: invariance to small geometric transformations (i.e. translations)
- ❑ Reduces the effects of noise by keeping the strongest signals



Max pooling

Single activation map



A 4x4 grid of numbers with color-coded quadrants: top-left (red), top-right (green), bottom-left (yellow), and bottom-right (blue). A vertical arrow on the left points upwards, and a horizontal arrow at the bottom points to the right.

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters
and stride 2



A 2x2 grid of numbers representing the result of max pooling with a 2x2 filter and stride 2. The quadrants are color-coded: top-left (red), top-right (green), bottom-left (yellow), and bottom-right (blue).

6	8
3	4

❑ Alternatives:

- Sum/average pooling
- overlapping pooling

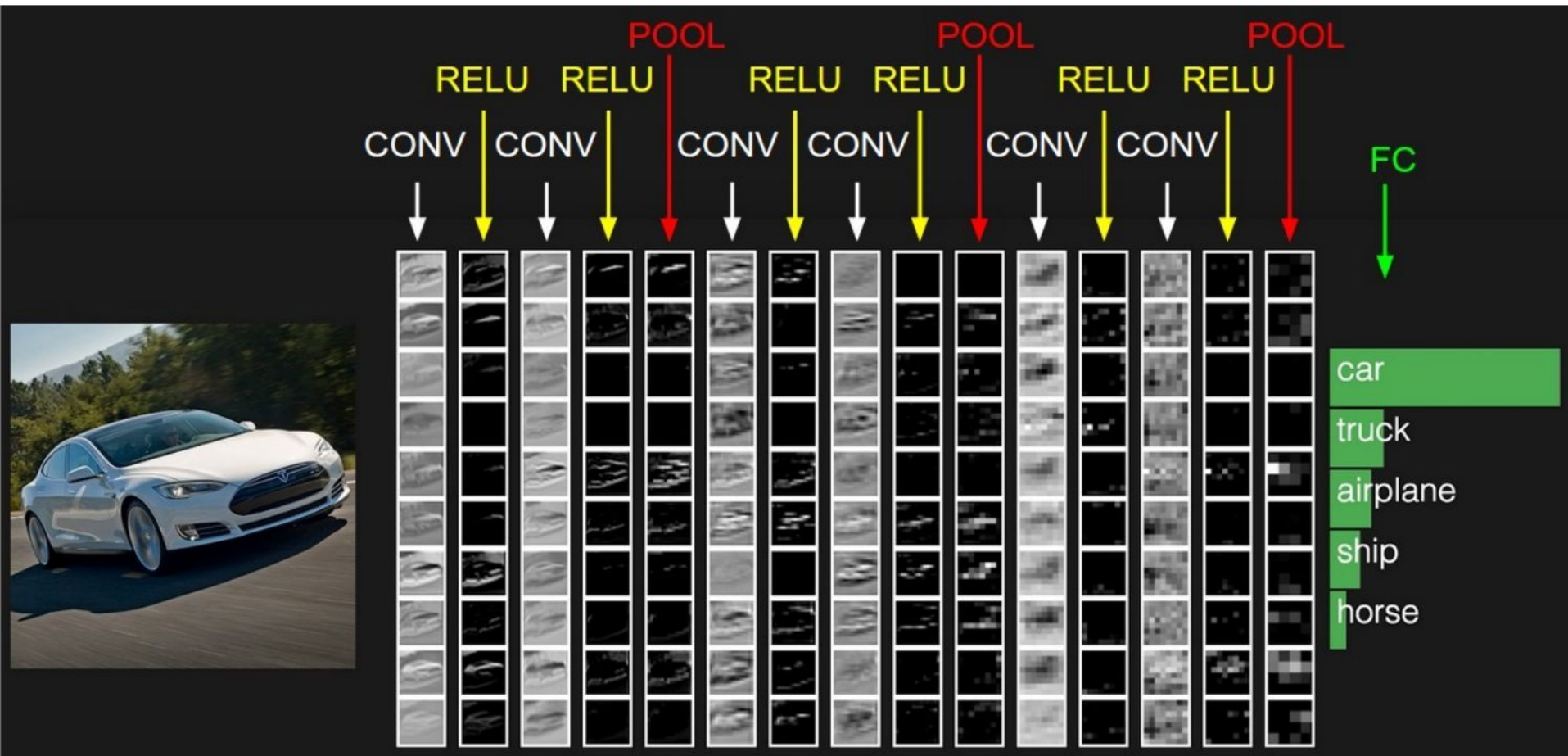
Pooling Code

```
from tensorflow import keras
# Build a simple model that takes a 32x32 RGB im
img_input = keras.Input(shape=(32, 32, 3))
# create a convolutional layer and pass the image
conv1_op = keras.layers.Conv2D(...,
activation='relu')(img_input)
# make a max pool layer
max_pooled_op =
keras.layers.MaxPool2D(pool_size=(2,2),
strides=(1,1))(conv1_op)
# Create a model (using the keras functional API)
model = keras.Model(img_input, max_pooled_op)
```

Deep Convolutional Networks

- ☒ Convolutional layer
- ☒ Non-linear activation function ReLU
- ☒ Max pooling layer
- ☐ Fully connected layer

Where is a fully connected layer?



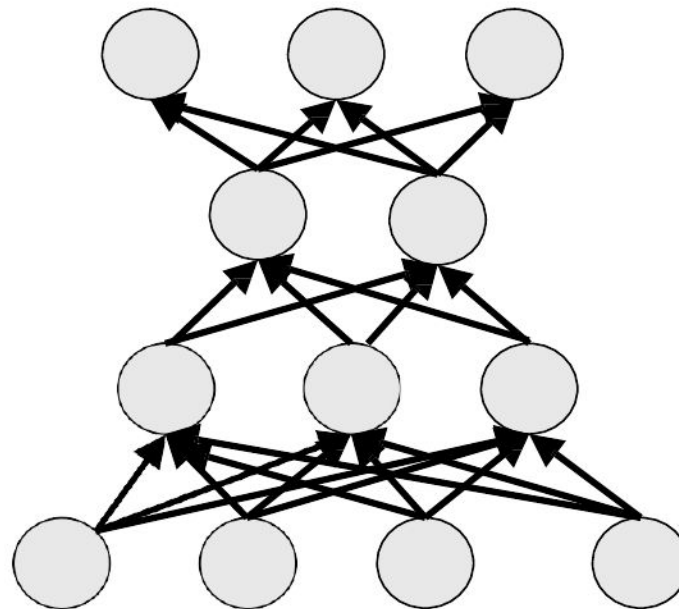
Fully connected layer

Each neuron is connected to the entire set of input neurons, as in ordinary Neural Networks or Multilayer Perceptron.

Output layer

Hidden layer

Hidden layer



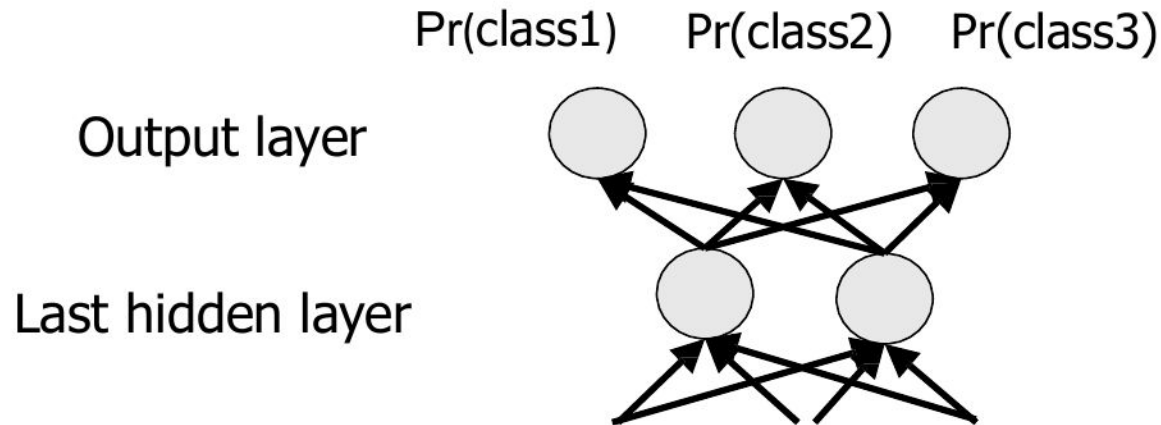
Fully connected layer

- Each hidden neuron is a linear combination of the input layers neurons...
 - This means we can define fully connected layers using matrix multiplication, just like in linear regression
- To make it more complex the output of this linear combination is put through a non-linear (activation) function.
- **$\mathbf{O} = f(\mathbf{iW} + \mathbf{b})$**
- Neurons between two adjacent layers are fully pairwise connected, but neurons within a single layer share no connections

Output layer

In classification problems:

- the output layer is often fully connected with number of neurons equal to number of classes
- followed by softmax non-linear activation, this means the sum of the output neurons sums to 1.



Fully Connected Code

```
from tensorflow import keras
# Build a simple model that takes a 32x32 RGB im
img_input = keras.Input(shape=(32, 32, 3))
# create a convolutional layer and pass the image
conv1_op = keras.layers.Conv2D(...,
activation='relu')(img_input)
# make a max pool layer
max_pooled_op =
keras.layers.MaxPool2D(pool_size=(2,2),
strides=(1,1))(conv1_op)
model_op = keras.layers.Dense(no_op_units,
activation='softmax')(max_pooled_op)
# Create a model (using the keras functional API)
model = keras.Model(img_input, model_op)
```


Do we need fully connected layers?

- Fully connected (or dense) layers allow every neuron in an input layer to influence a neuron in the output layer.
 - This means the neuron has a global “receptive field” and all the information can be used to make predictions.
 - It also means the input images must all be the same size, otherwise the matrix dimensions will not match.
- Convolutional neural networks can exist without fully connected layers, which means they can operate on images with variable resolutions.
 - Such networks are referred to as fully convolutional.
 - Each output neuron will have a more local receptive field, as information is not shared across the whole image.
 - The models are typically used for tasks where spatial resolution is important, such as segmentation.

Take Home Messages

- ❑ Understanding the structure of convolutional neural networks
 - ❑ Convolutional layer
 - ❑ ReLu
 - ❑ Max pooling layer
 - ❑ Fully connected layer
 - ❑ How to compute spatial dimensions
 - ❑ How to compute number of parameters

Summary

- ❑ Convolutional Neural Networks provide a flexible approach to build very deep neural networks.
 - ❑ Convolutions learn function that look at a neighbourhood of pixels, only suitable for spatial / spatiotemporal data
 - ❑ CNNs often use max pooling, and fully-connected layers to solve tasks
- ❑ Next time we'll talk about some more details for how to learn the parameters of these models.