Convolutional Neural Networks (CNN)

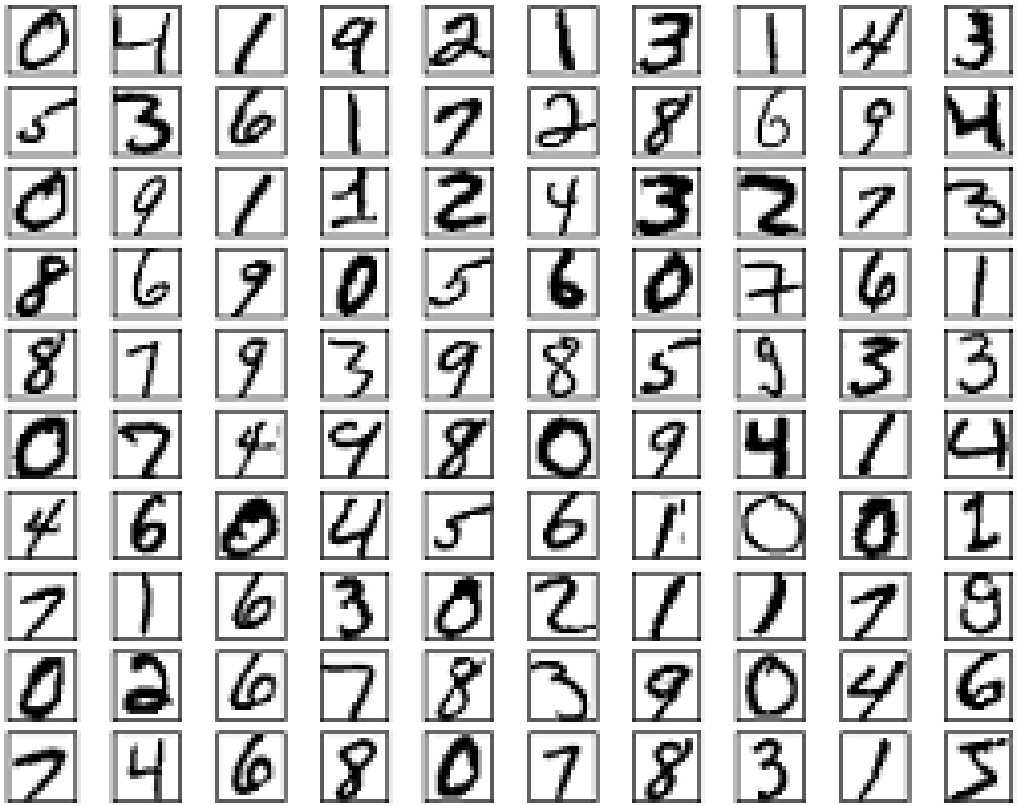
Basics of CNN, the Receptive Field

Nina Hernitschek
Centro de Astronomía CITEVA
Universidad de Antofagasta

August 9, 2023

Neural Network Architectures

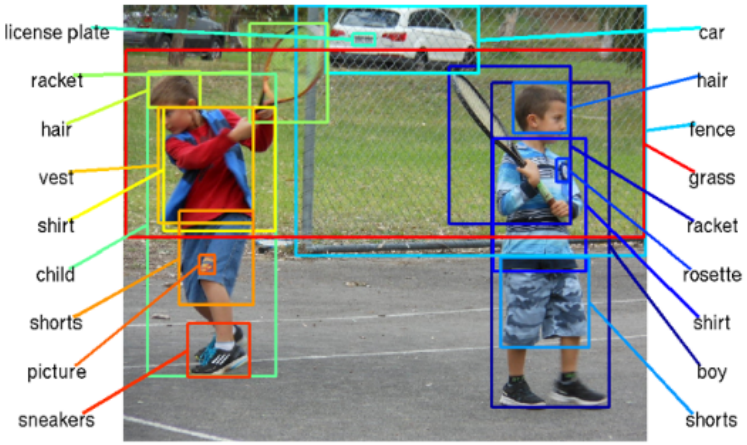
'classic' trained feed-forward neural networks can handle object recognition

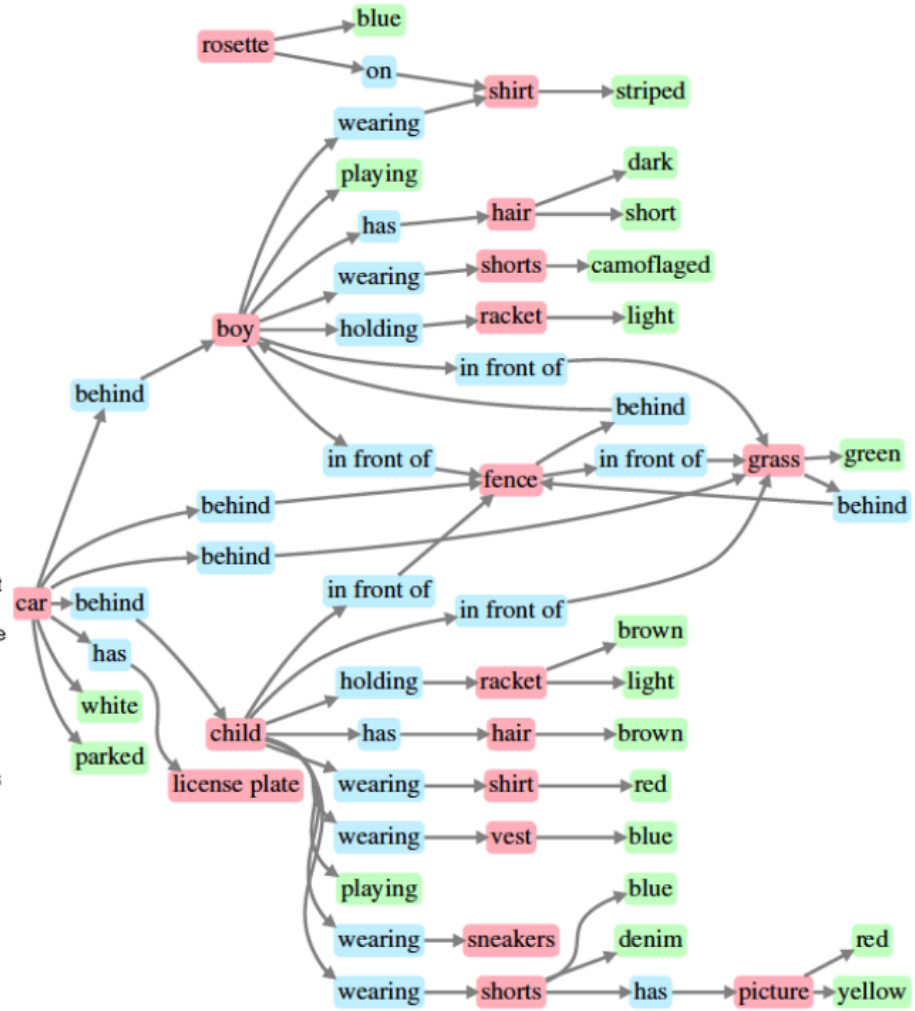


A subset of the MNIST Handwritten Digits dataset.

Neural Network Architectures

but we want not only object recognition...

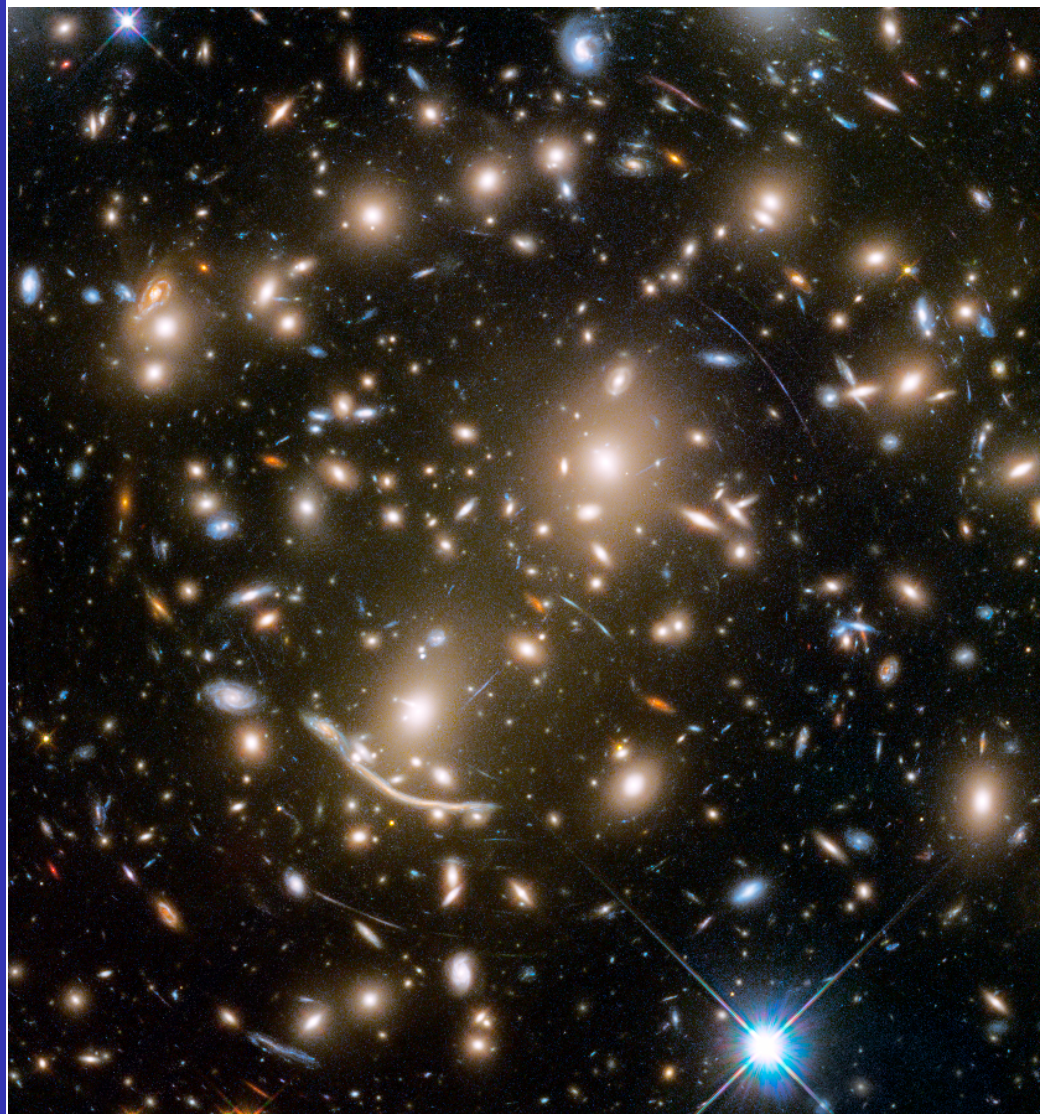




Johnson *et al.*, "Image Retrieval using Scene Graphs", CVPR 2015

Neural Network Architectures

but we want not only object recognition...



interacting galaxies

galaxy clusters

blending

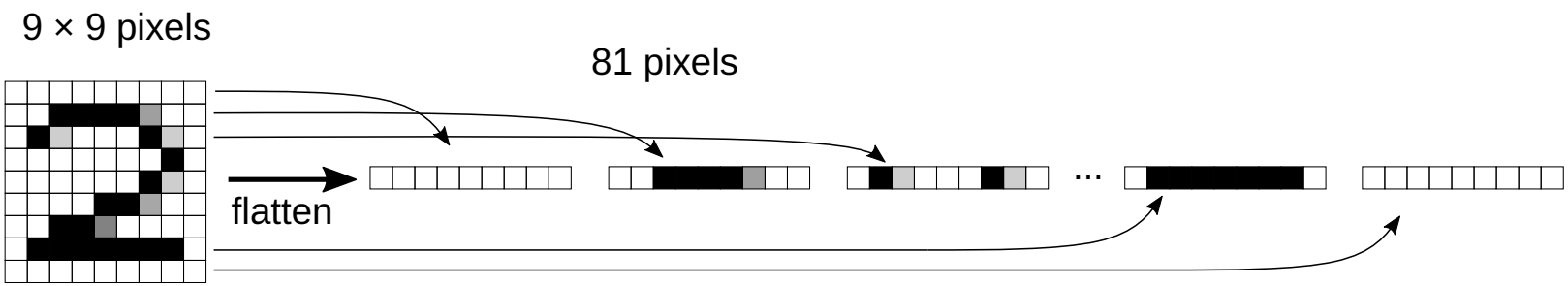
gravitational lenses

foreground stars

...

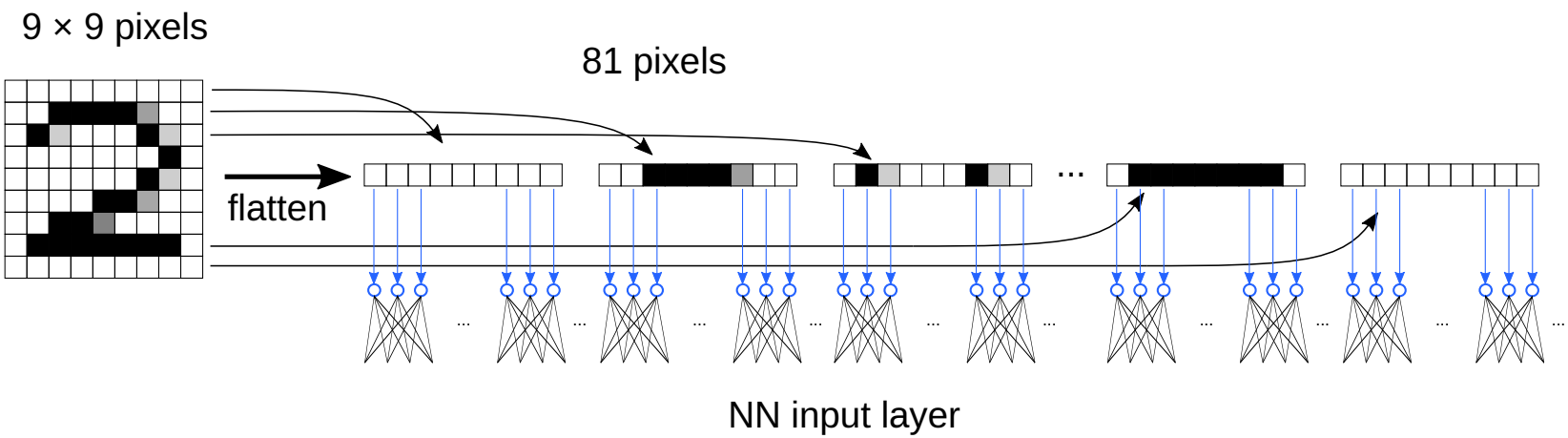
Neural Network Architectures

classic approach to **computer vision**:
treat image pixels as individual features to feed into a deep neural network (Multi-Layer Perceptron)



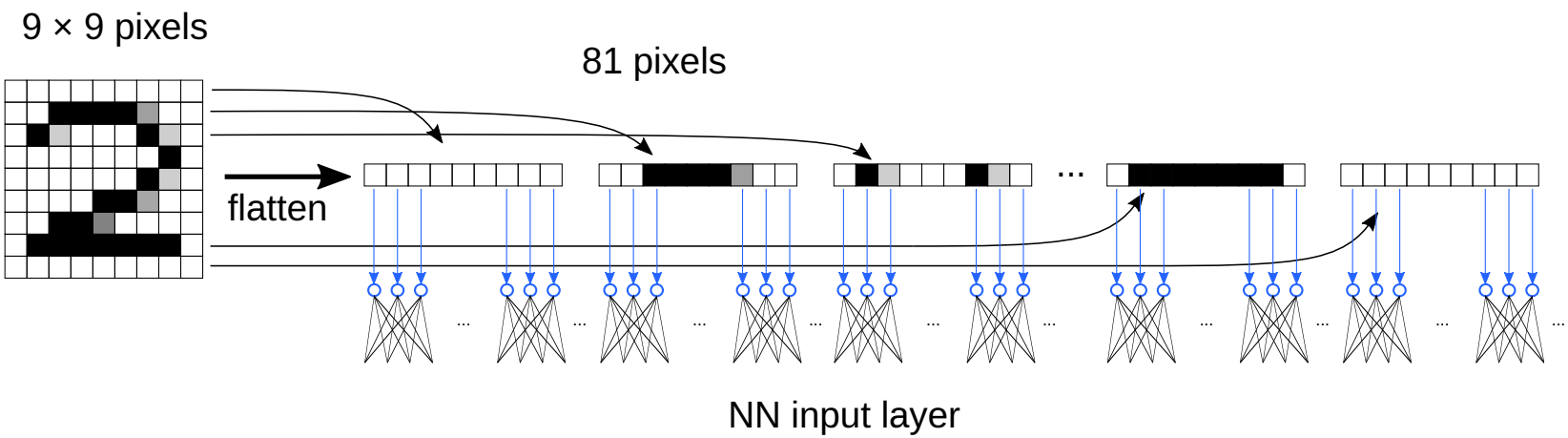
Neural Network Architectures

classic approach to **computer vision**:
treat image pixels as individual features to feed into a deep neural network (Multi-Layer Perceptron)



Neural Network Architectures

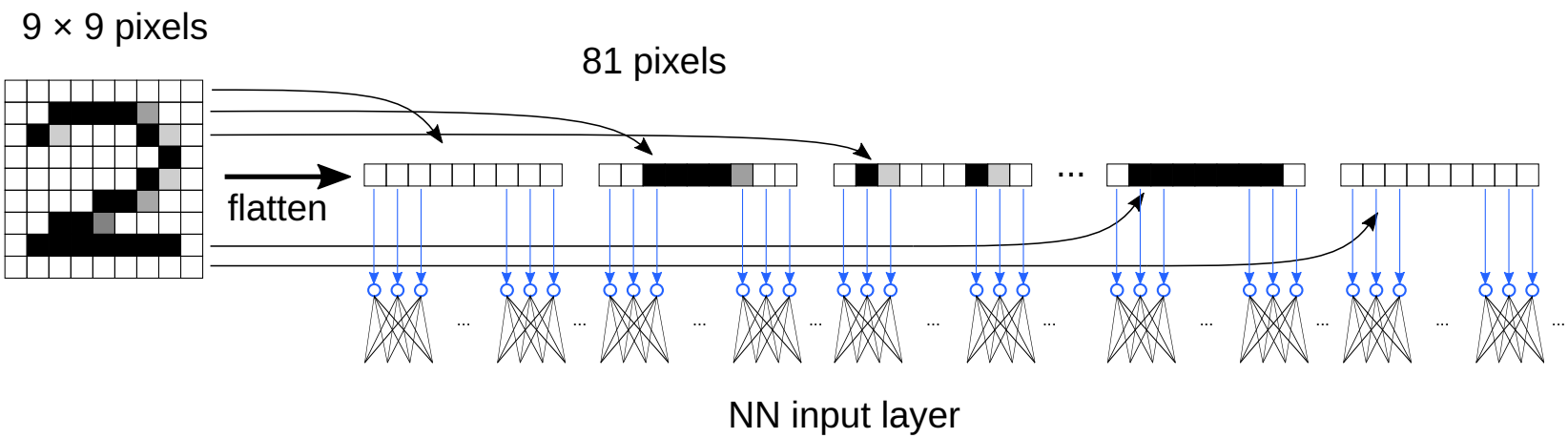
classic approach to **computer vision**:
treat image pixels as individual features to feed into a deep neural network (Multi-Layer Perceptron)



okay for simple and low-resolution images

Neural Network Architectures

classic approach to **computer vision**:
treat image pixels as individual features to feed into a deep neural network (Multi-Layer Perceptron)



flattening the image removes spatial information

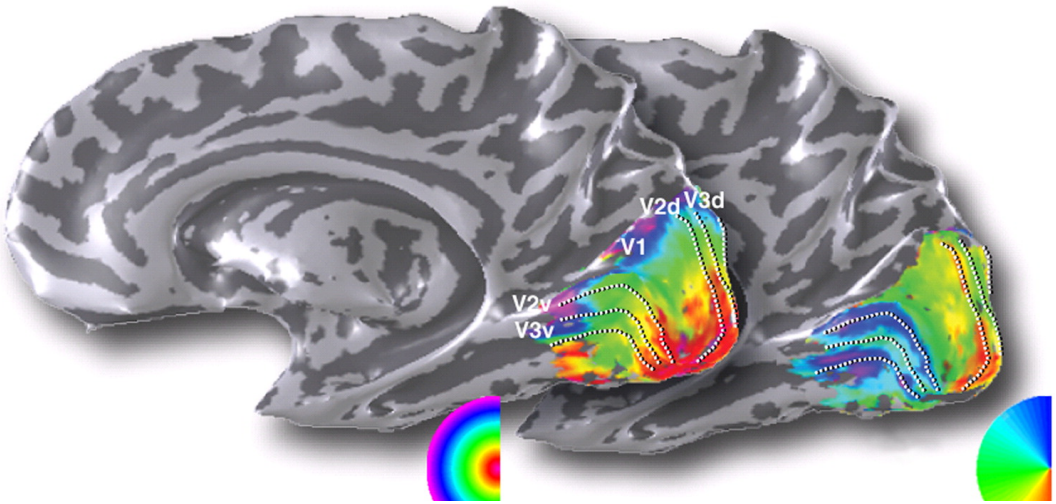
for complex and high-resolution dataset: Convolutional Neural Network (CNN) is the solution

Artificial Vision

Convolutional Neural Networks (CNN) were not invented overnight

topographical mapping of the visual cortex (in humans and animals):

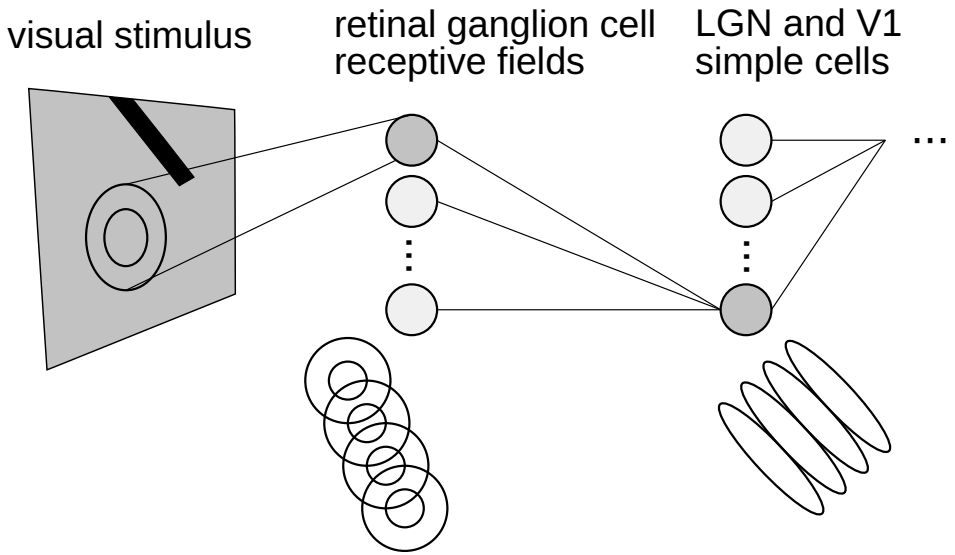
nearby cells in the cortex represent nearby regions in the visual field



credit: Wandell et al. (2007)

Artificial Vision

the visual cortex is organized hierarchically:



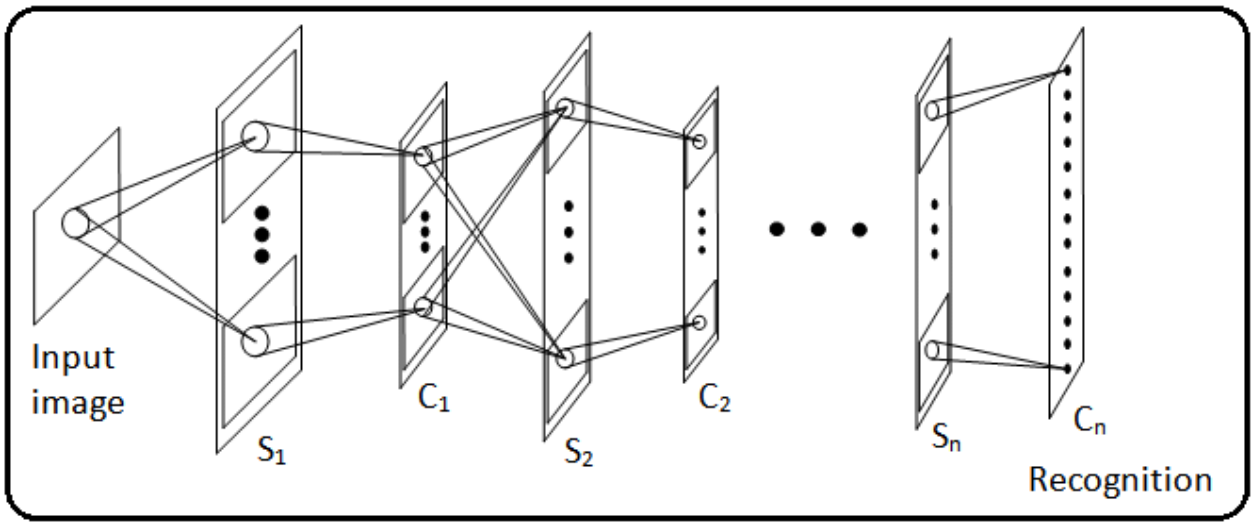
adapted from: Lane MJcIntosh, CS231n (2017)

simple cells: respond to light orientation
complex cells: respond to light orientation and movement
hypercomplex cells: respond to movement with an end point

Artificial Vision

Neocognitron (Fukushima 1980)

uses a "sandwich" architecture of simple cells and complex cells where simple cells contain modifiable parameters, complex cells perform pooling operation



⇨ the first CNN

Convolutional Neural Networks - The Idea

Convolutional neural networks (CNNs) are similar to feedforward neural networks: A convolutional neural network consists of an input layer, hidden layers for feature learning and an output layer for classification.
The difference lies in the operations performed by the hidden layers: In a CNN, the hidden layers include layers that perform convolutions.
These networks harness principles from linear algebra, particularly matrix multiplication, to **identify patterns within an image**.

Convolutional Neural Networks - The Idea

Convolutional neural networks (CNNs) are similar to feedforward neural networks: A convolutional neural network consists of an input layer, hidden layers for feature learning and an output layer for classification.
The difference lies in the operations performed by the hidden layers: In a CNN, the hidden layers include layers that perform convolutions.
These networks harness principles from linear algebra, particularly matrix multiplication, to **identify patterns within an image**.

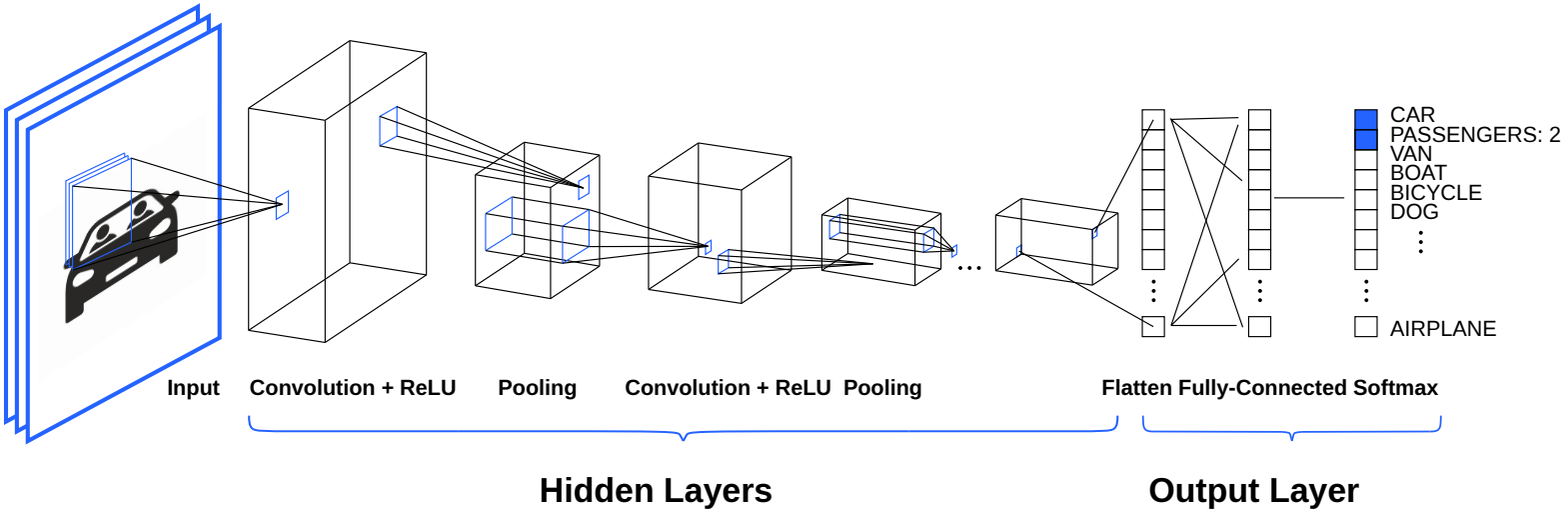
Basic underlying idea:
An image is a matrix of pixels. Important parts on the image (objects, edges) are within close distances. Individual neurons respond to input only in a restricted region, like in the *visual cortex* of the brain.

Recap: Tensors

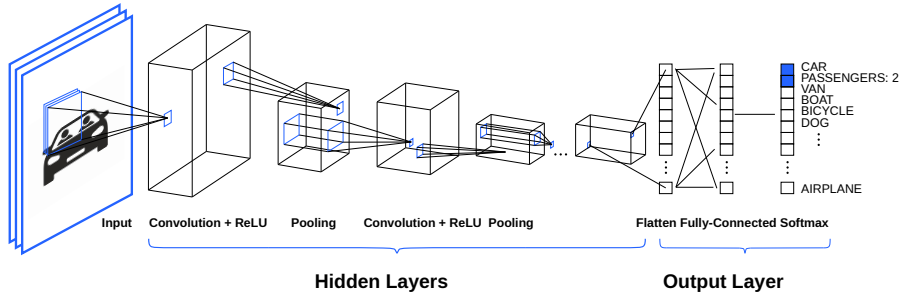
name	dimensionality (rank, order)	example
scalar	0	42
vector	1	$(6.6\ 8\ 2)$
matrix	2	$\begin{pmatrix} 1 & 2 & 3 \\ 55 & 0.6 & 1 \end{pmatrix}$
cube	3	$\begin{pmatrix} (6.7\ 7.5\ 2) & (4\ 8.4\ 5.1) & (55.7\ 1\ 2.5) \\ (7.5\ 3\ 2) & (3\ 8\ 9.1) & (6.5\ 9.4\ 2) \\ (2.7\ 2\ 2.5) & (1.4\ 8\ 1.5) & (6.2\ 8.3\ 2) \end{pmatrix}$

Convolutional Neural Networks - The Idea

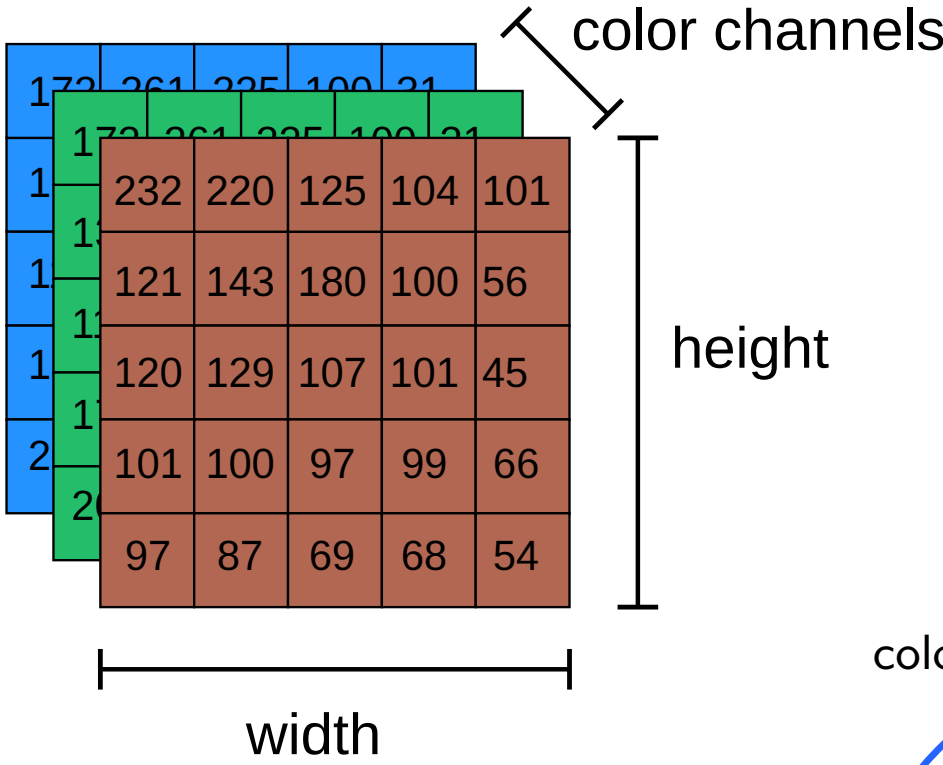
A typical CNN architecture consists of alternating **Convolutional layers** and **Pooling layers** that extract features. Finally, their output is flattened into a one-dimensional feature layer that is passed into a Multi-Layer Perceptron (fully-connected layers) to obtain a prediction.



The Input



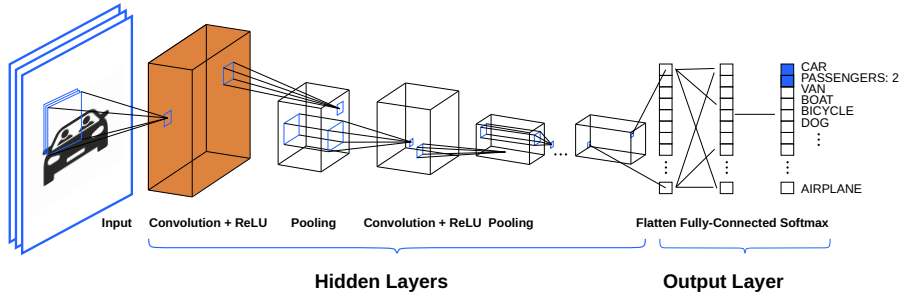
preserving the input image's spatial and temporal dependencies



color image movie

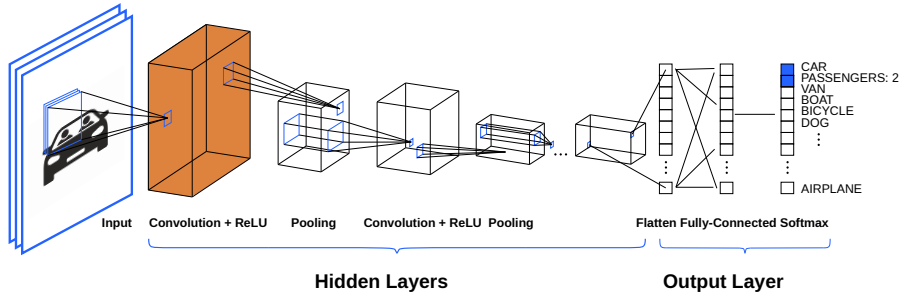
In a CNN, the input is a tensor with shape
(input height) $\times$ (input width) $\times$ (input channels) $\times$ (number of inputs).

Convolution Layer



The objective of the Convolution Layer is to extract the high-level features, such as edges, from the input image.

Convolution Layer



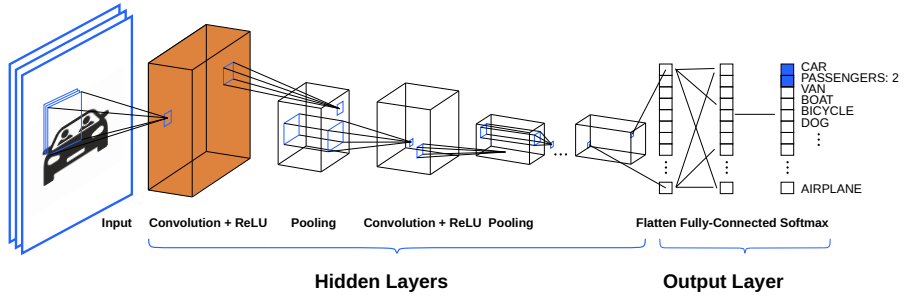
If you have used image/ photo editing, you likely have already applied convolutions, whether you realize it or not:

blurring/smoothing

sharpening

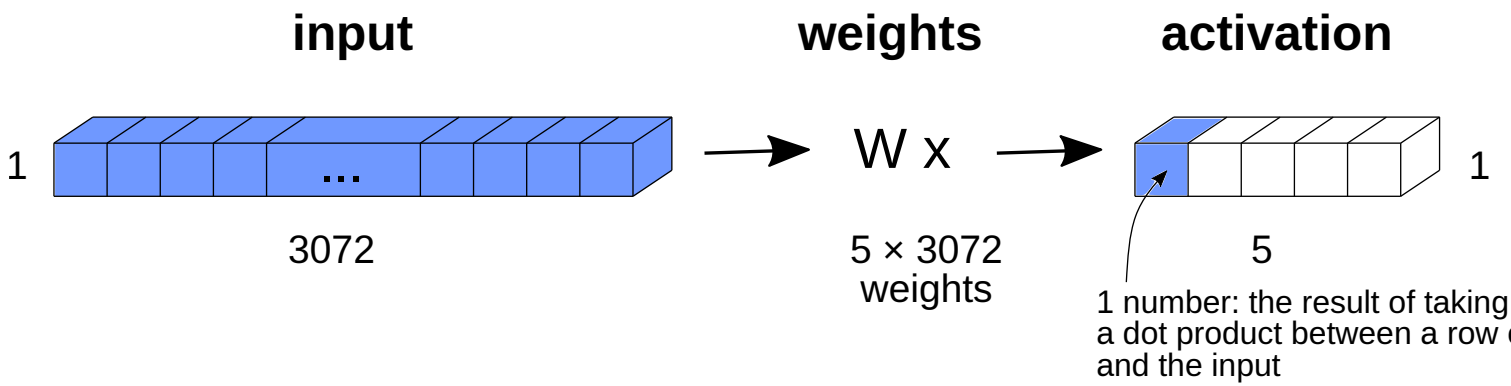
edge detection

Convolution Layer

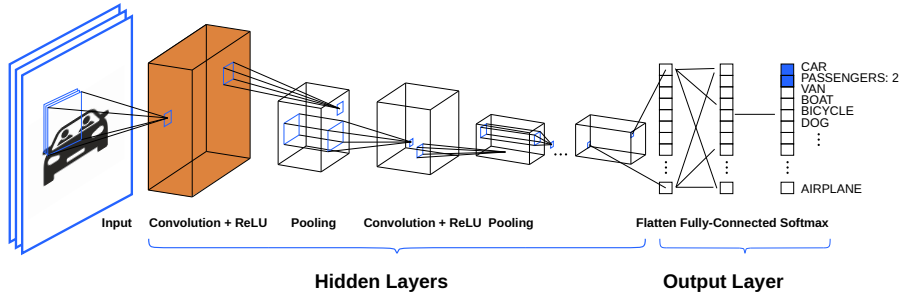


in feed-forward neural network:

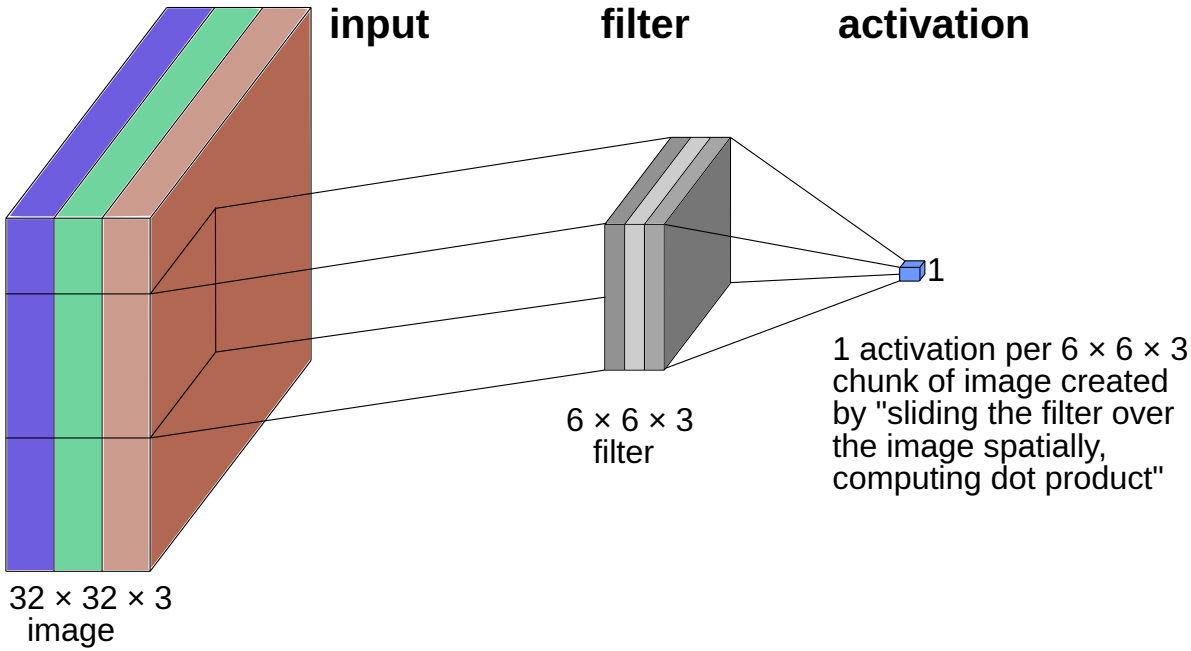
a 32×32 pixel $\times$ 3 channels image is flattened



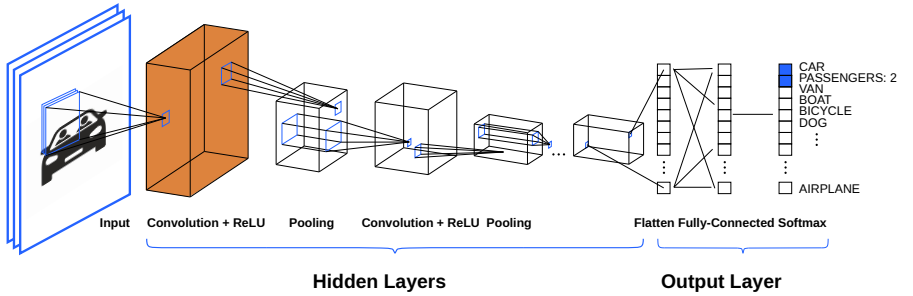
Convolution Layer



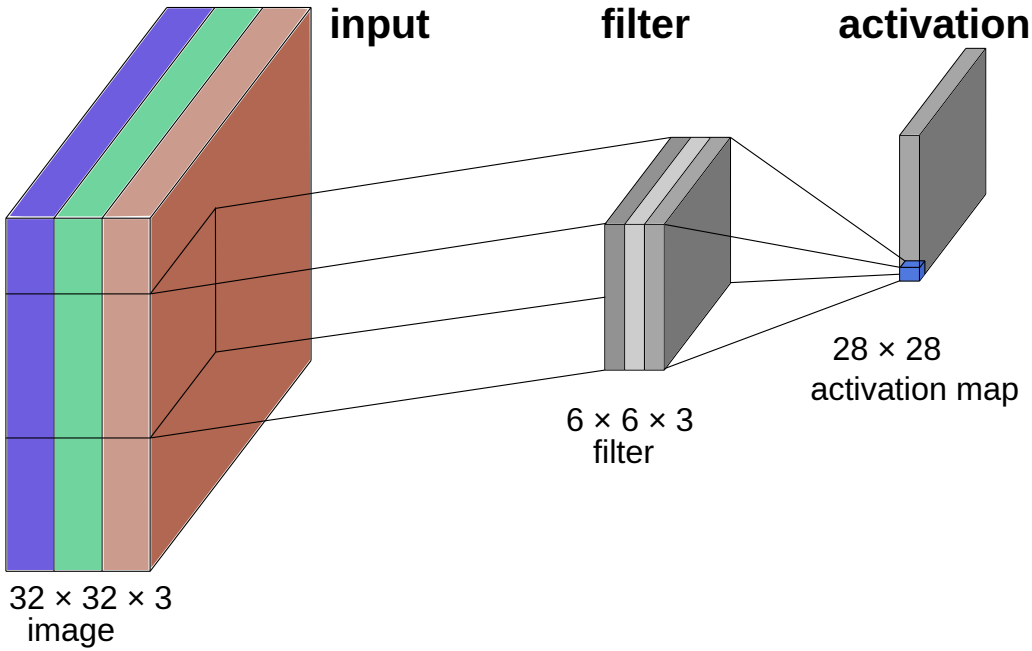
in a CNN:
processing a 32 × 32 pixel × 3 channels image as tensor



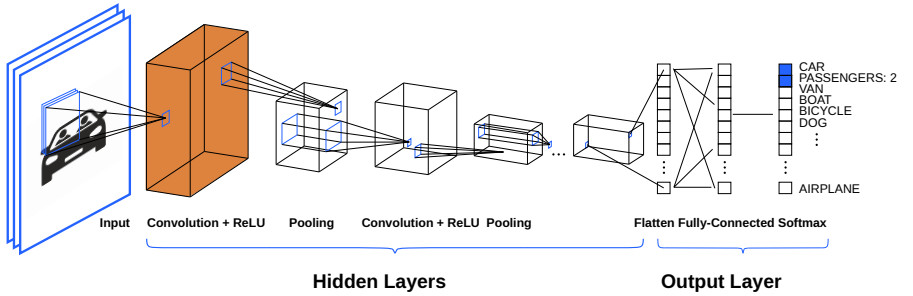
Convolution Layer



in a CNN:
processing a 32 × 32 pixel × 3 channels image as tensor



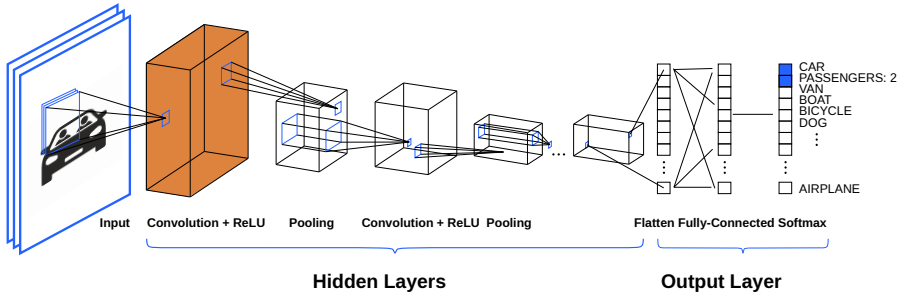
Convolution Layer



What happens internally:
convolution is an element-wise multiplication of two matrices followed by summation

The operation performed in CNN is technically a **cross-correlation**, not a convolution:

Convolution Layer



What happens internally:
convolution is an element-wise multiplication of two matrices followed by summation

The operation performed in CNN is technically a **cross-correlation**, not a convolution:

Convolution:

input

filter

$$y_{i,j} = \sum_l \sum_k x_{i-l,j-m} w_{l,m}$$

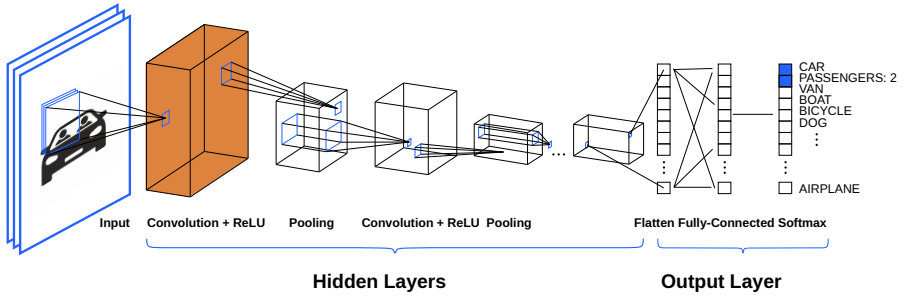
Cross-Correlation (as carried out here):

$$y_{i,j} = \sum_l \sum_k x_{i+l,j+m} w_{l,m}$$

the difference is for cross-correlation we don't have to 'flip' the filter's kernel relative to the input

both can be interchanged through a simple rotation operation

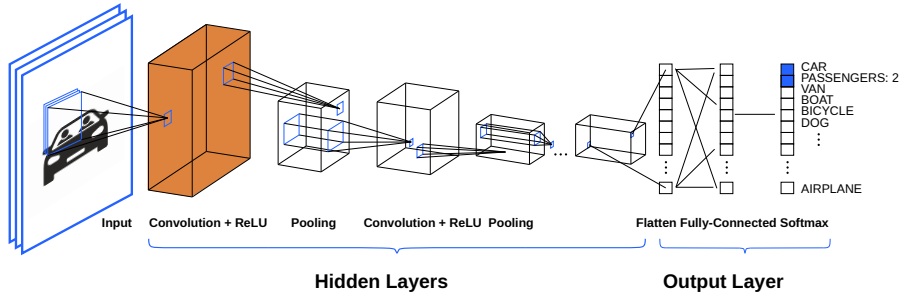
Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

We illustrate this in the following **example**:

Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

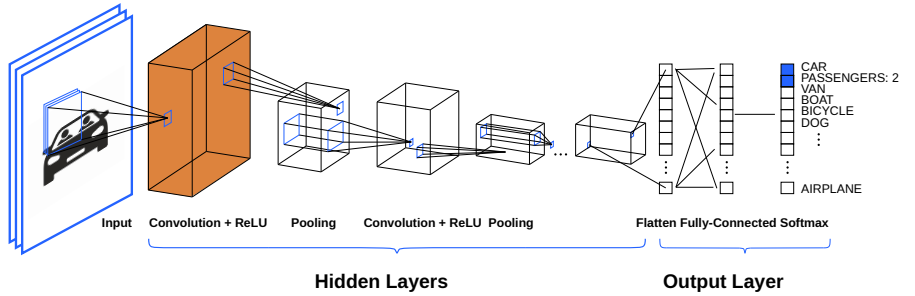
image

$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

convolved feature

4		

Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

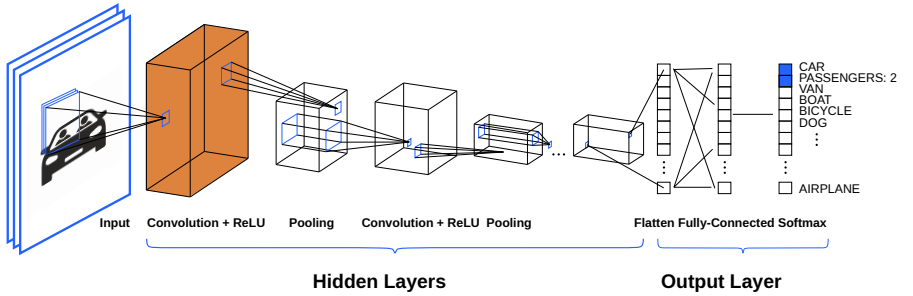
image

1	1 (×1)	1 (×0)	0 (×1)	0
0	1 (×0)	1 (×1)	1 (×0)	0
0	0 (×1)	1 (×0)	1 (×1)	1
0	0	1	1	0
0	1	1	0	0

convolved feature

4	3	

Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

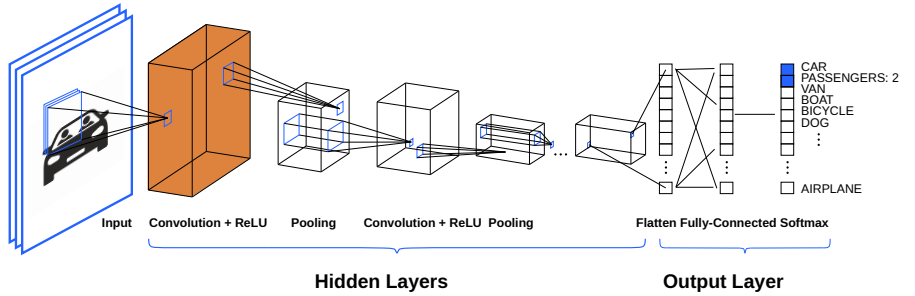
image

1	1	$1_{\times 1}$	$0_{\times 0}$	$0_{\times 1}$
0	1	$1_{\times 0}$	$1_{\times 1}$	$0_{\times 0}$
0	0	$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$
0	0	1	1	0
0	1	1	0	0

convolved feature

4	3	4

Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

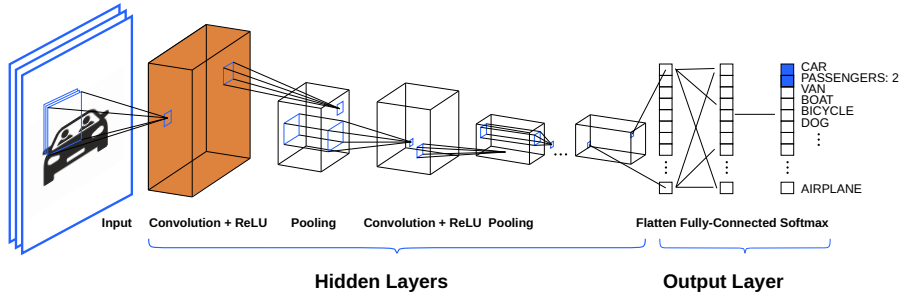
image

1	1	1	0	0
$0_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	1	0
$0_{\times 0}$	$0_{\times 1}$	$1_{\times 0}$	1	1
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	0
0	1	1	0	0

convolved feature

4	3	4
2		

Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

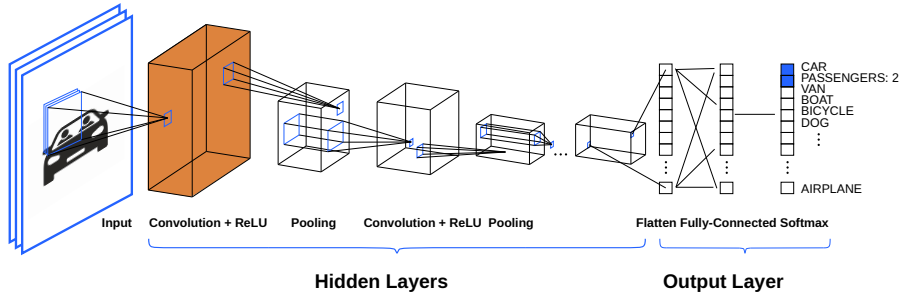
image

1	1	1	0	0
0	$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0
0	$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1
0	$0_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0
0	1	1	0	0

convolved feature

4	3	4
2	4	

Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

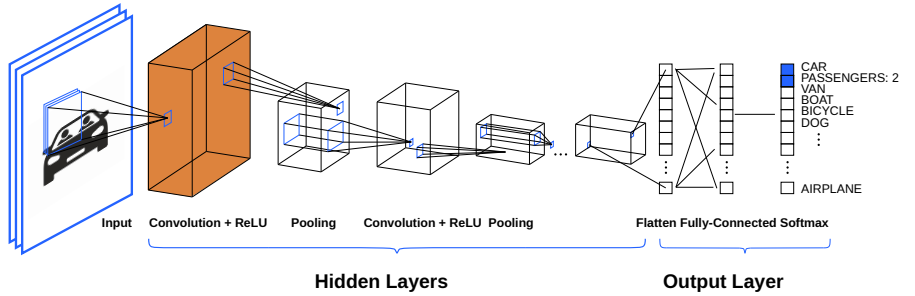
image

1	1	1	0	0
0	1	$1_{\times 1}$	$1_{\times 0}$	$0_{\times 1}$
0	0	$1_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$
0	0	$1_{\times 1}$	$1_{\times 0}$	$0_{\times 1}$
0	1	1	0	0

convolved feature

4	3	4
2	4	3

Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

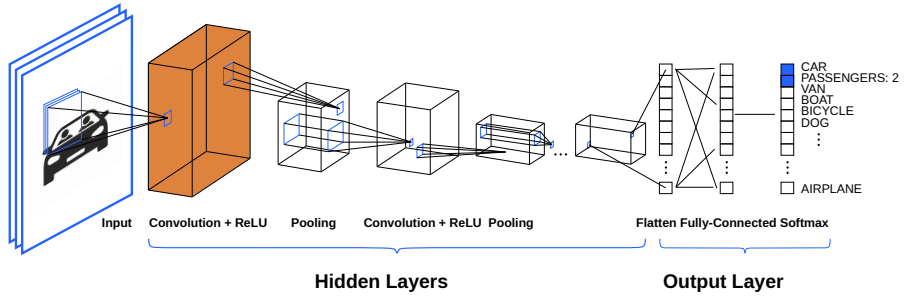
image

1	1	1	0	0
0	1	1	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
$0_{\times 0}$	$0_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0

convolved feature

4	3	4
2	4	3
2		

Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

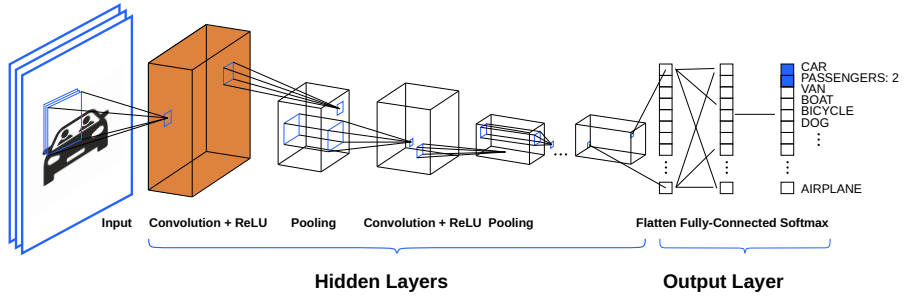
image

1	1	1	0	0
0	1	1	1	0
0	$0_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	1
0	$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	0
0	$1_{\times 1}$	$1_{\times 0}$	$0_{\times 1}$	0

convolved feature

4	3	4
2	4	3
2	3	

Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

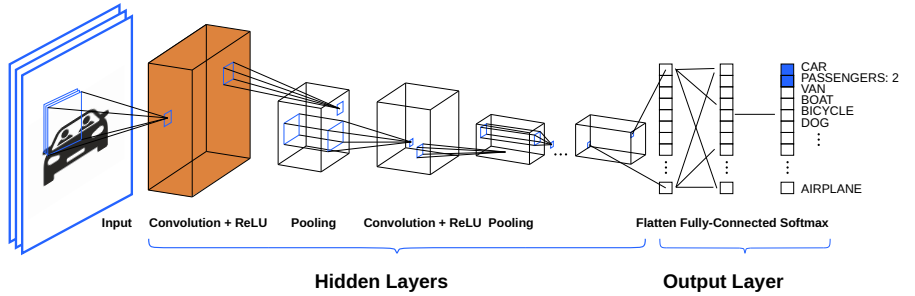
image

1	1	1	0	0
0	1	1	1	0
0	0	$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$
0	0	$1_{\times 0}$	$1_{\times 1}$	$0_{\times 0}$
0	1	$1_{\times 1}$	$0_{\times 0}$	$0_{\times 1}$

convolved feature

4	3	4
2	4	3
2	3	4

Convolution Layer



The process of convolution is basically the sum-product of the filter with an overlapping part of the image in a step-wise, strideful manner.

We illustrate this in the following **example**:

Convoluting a 5×5×1 image with a 3×3×1 kernel to get a 3×3×1 feature

kernel

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

image

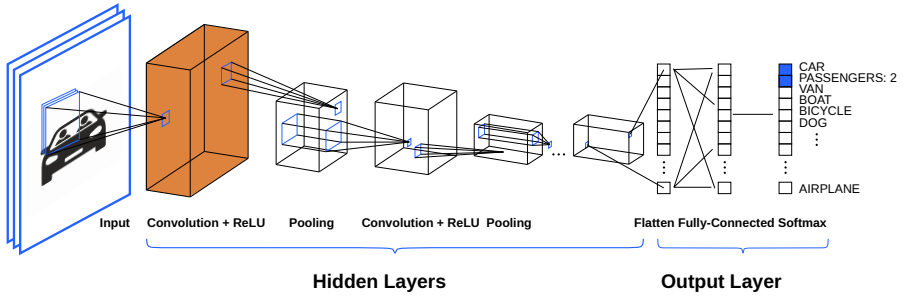
1	1	1	0	0
0	1	1	1	0
0	0	$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$
0	0	$1_{\times 0}$	$1_{\times 1}$	$0_{\times 0}$
0	1	$1_{\times 1}$	$0_{\times 0}$	$0_{\times 1}$

convolved feature

4	3	4
2	4	3
2	3	4

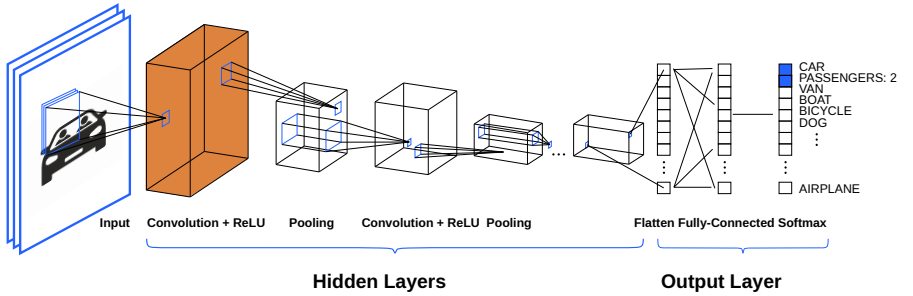
$\Rightarrow$ The kernel K shifts 9 times because of Stride Length = 1 (Non-Strided)

Convolution Layer



Design consideration: **The size of the convolution output.**

Convolution Layer

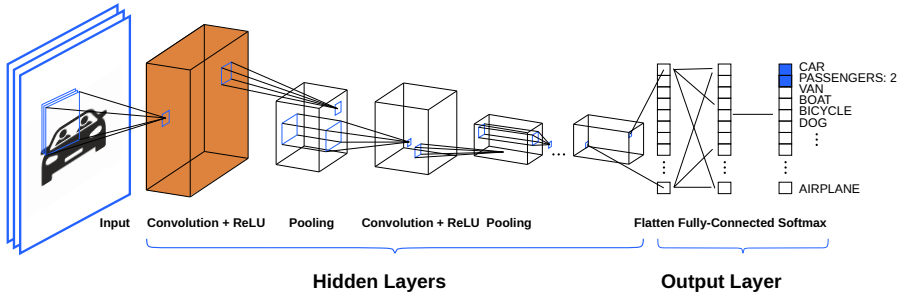


Design consideration: **The size of the convolution output.**

The stride of the convolution (how many pixel the filter jumps each time) is not necessarily 1 pixel.

The size of the convolution output depends on implementation factors and might not be identical to the input.

Convolution Layer



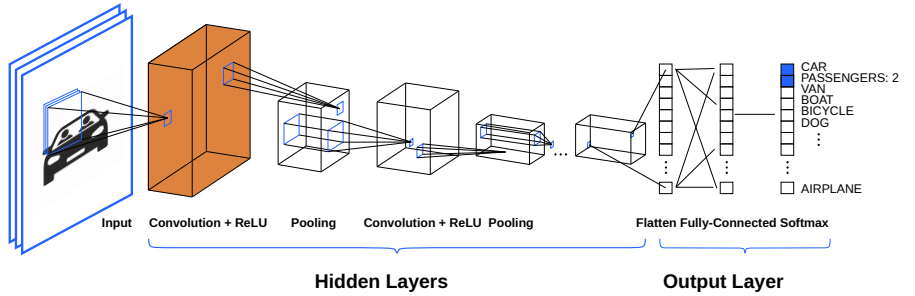
Design consideration: **The size of the convolution output.**

Image size: $I \times I$
Filter size: $F \times F$
Stride: S

the **floor** operator: the smallest integer less than or equal to the input

Output size (in each dimension): $\lfloor (I - F)/S \rfloor + 1$
assuming the filter is not allowed to go beyond the edge

Convolution Layer



Design consideration: **The size of the convolution output.**

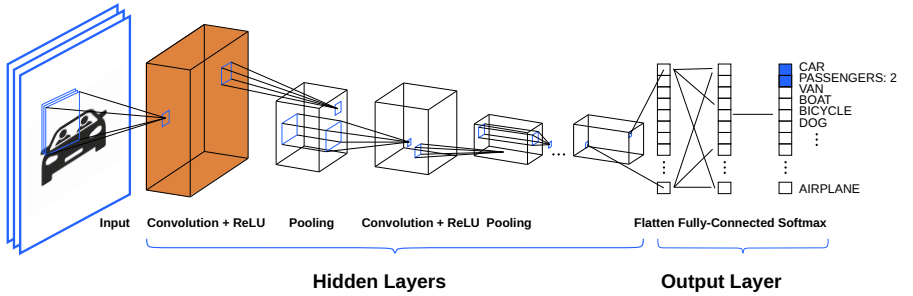
Image size: $I \times I$
Filter size: $F \times F$
Stride: S

the **floor** operator: the smallest integer less than or equal to the input

Output size (in each dimension): $\lfloor (I - F)/S \rfloor + 1$
assuming the filter is not allowed to go beyond the edge

⇒ The result is a reduction in the output size, even for S=1
If this is not acceptable: Use **padding**.

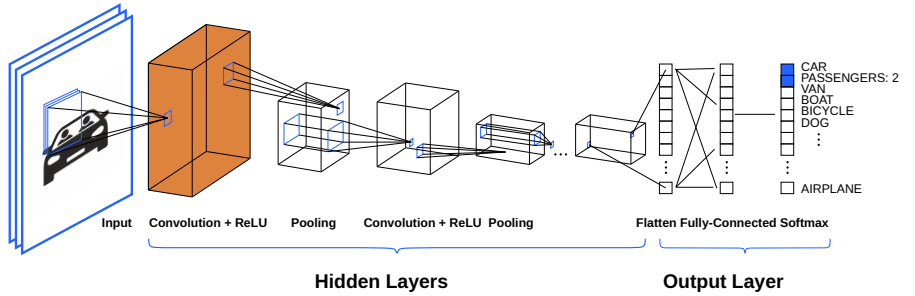
Convolution Layer



The solution: **Zero Padding**

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

Convolution Layer



The solution: **Zero Padding**

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

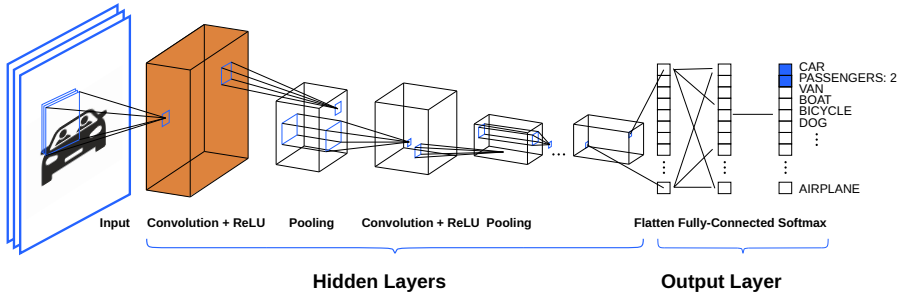
For an F width filter:

odd F: pad on left, right, top, bottom with a width of $(F-1)/2$ zeros
even F: pad one side and top with $F/2$ zeros, and the other side and bottom with a width of $F/2-1$ zeros

The resulting image width is $I+F-1$, the result of the convolution has width I

For a stride $S>1$, zero padding is adjusted to ensure that the result of the convolution has width $\lceil I/S \rceil$: first zero-padding the image with $S\lceil I/S \rceil - I$ zeros and then apply the rules above.

Convolution Layer



The **Computational Cost** of carrying out the convolution (cross-correlation) operation:

the cost of scanning an $I \times I$ image with a $F \times F$ filter is $\mathcal{O}(I^2F^2)$:
we have F^2 multiplications at each of I^2 image pixels in the equation

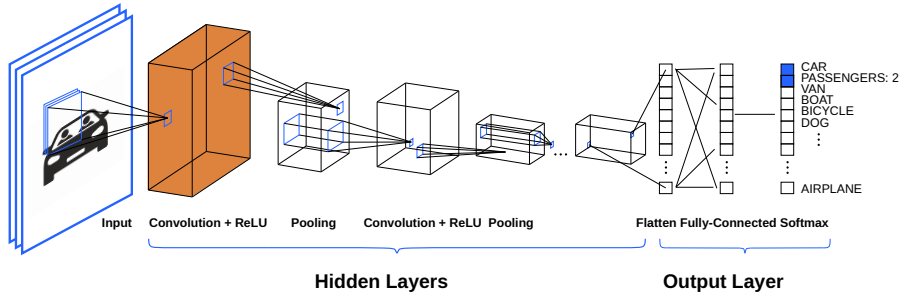
$$y_{i,j} = \sum_l \sum_k x_{i+l,j+m} \; w_{l,m}$$

and carry this out $(I - F)/S$ times (the size of the output),

so the cost of scanning an $I \times I$ image with a $F \times F$ filter is $\mathcal{O}(I^2F^2)$

despite also $(I - F)/S$ (how often we carry out this operation as the filter with size F moves with stride S over the image with size I) contributes to the computing time, for the $\mathcal{O}$ we only give the highest order

Convolution Layer



The **Computational Cost** of carrying out the convolution (cross-correlation) operation:

the cost of scanning an $I \times I$ image with a $F \times F$ filter is $\mathcal{O}(I^2F^2)$:
we have F^2 multiplications at each of I^2 image pixels in the equation

$$y_{i,j} = \sum_l \sum_k x_{i+l,j+m} \; w_{l,m}$$

and carry this out $(I - F)/S$ times (the size of the output),
so the cost of scanning an $I \times I$ image with a $F \times F$ filter is $\mathcal{O}(I^2F^2)$

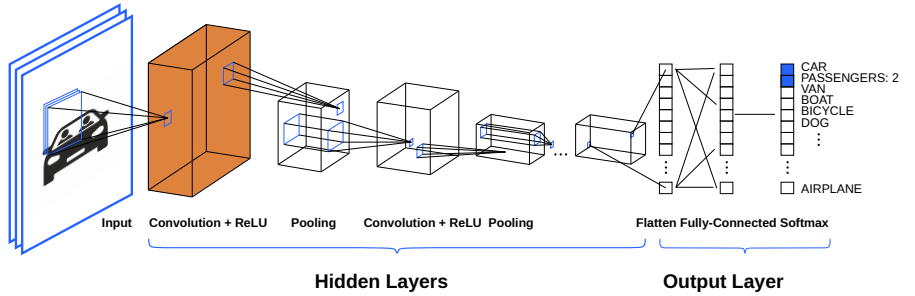
⇒ expensive for large filters

a solution: convolutions using **Discrete Fourier Transforms (DFTs)**:

$$Y = IDFT2(DFT2(X) \circ conj(DFT2(W)))$$

where IDFT2 is the **second order inverted discrete Fourier transform**
computational cost $\mathcal{O}(I^2 \log F)$

Convolution Layer



The **Computational Cost** of carrying out the convolution (cross-correlation) operation:

we have F^2 multiplications at each of I^2 image pixels in the equation

$$y_{i,j} = \sum_l \sum_k x_{i+l,j+m} \; w_{l,m}$$

and carry this out $(I - F)/S$ times (the size of the output),
so the cost of scanning an $I \times I$ image with a $F \times F$ filter is $\mathcal{O}(I^2 F^2)$

⇒ expensive for large filters

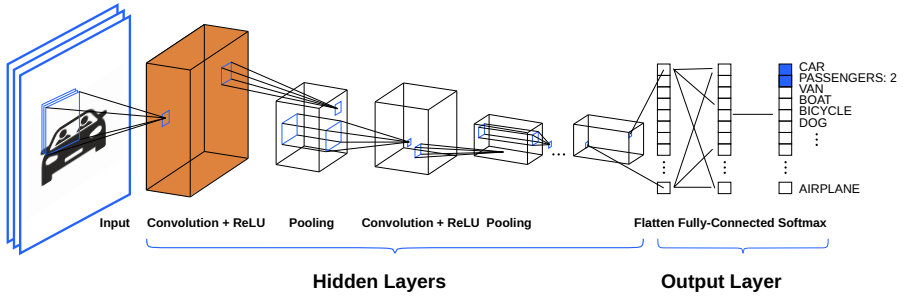
a solution: convolutions using **Discrete Fourier Transforms (DFTs)**:

$$Y = IDFT2(DFT2(X) \circ conj(DFT2(W)))$$

where IDFT2 is the **second order inverted discrete Fourier transform** computational cost $\mathcal{O}(I^2 \log F)$

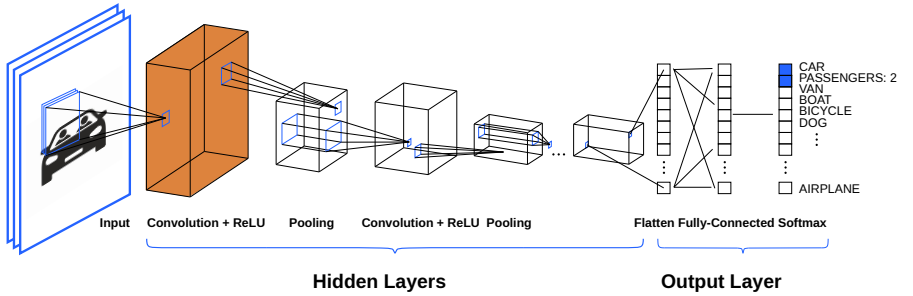
⇒ significant reduction in computational cost

Convolution Layer



How to **set** the kernel values of the filters?

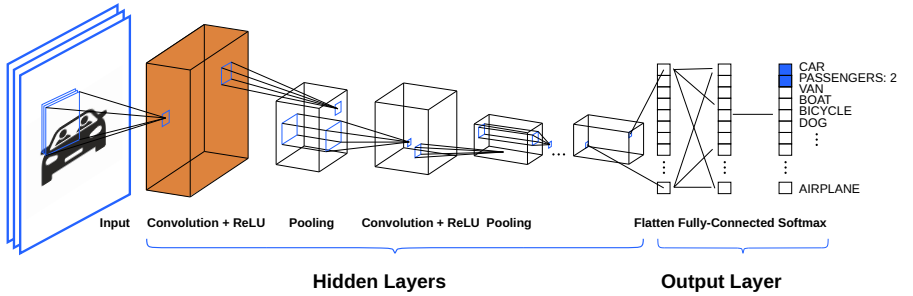
Convolution Layer



How to **set** the kernel values of the filters?

In the Convolutional Neural Network, these kernel values in the filter are **essentially weights to be learned and trained** to automatically capture spatial edge patterns in images.

Convolution Layer

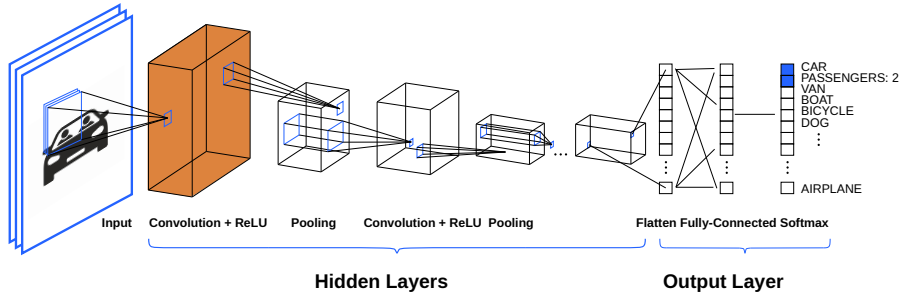


How to **set** the kernel values of the filters?

In the Convolutional Neural Network, these kernel values in the filter are **essentially weights to be learned and trained** to automatically capture spatial edge patterns in images.

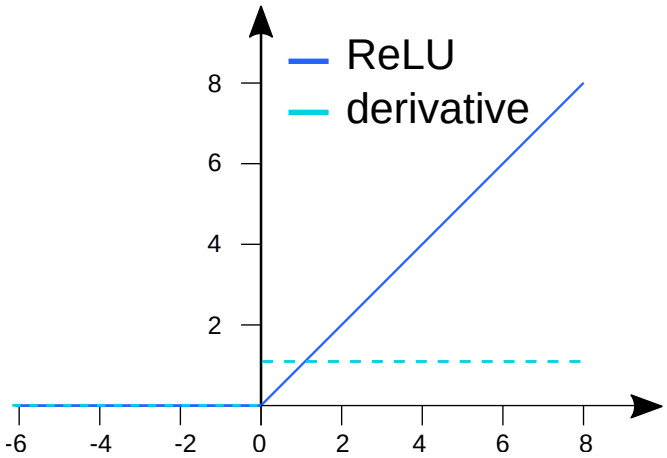
In addition to its ability to capture complex patterns, the small number of weights (filters are usually small relative to the images) in each layer makes the CNN very efficient to train.

Convolution Layer



After each convolution operation, a **Rectified Linear Unit (ReLU) transformation** is applied to the feature map, introducing nonlinearity to the model.
This is the activation function we have seen before.

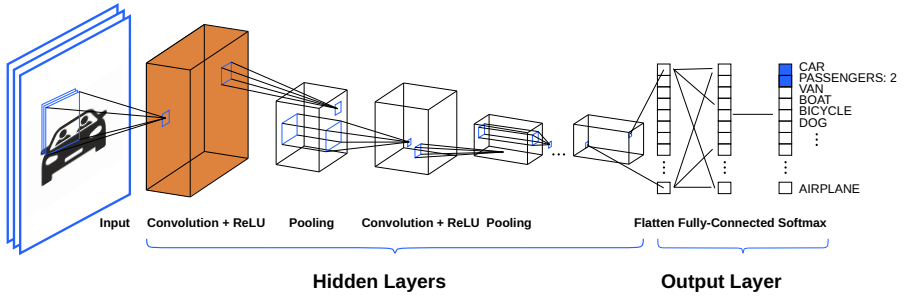
ReLU effectively removes negative values from an activation map by setting them to zero.



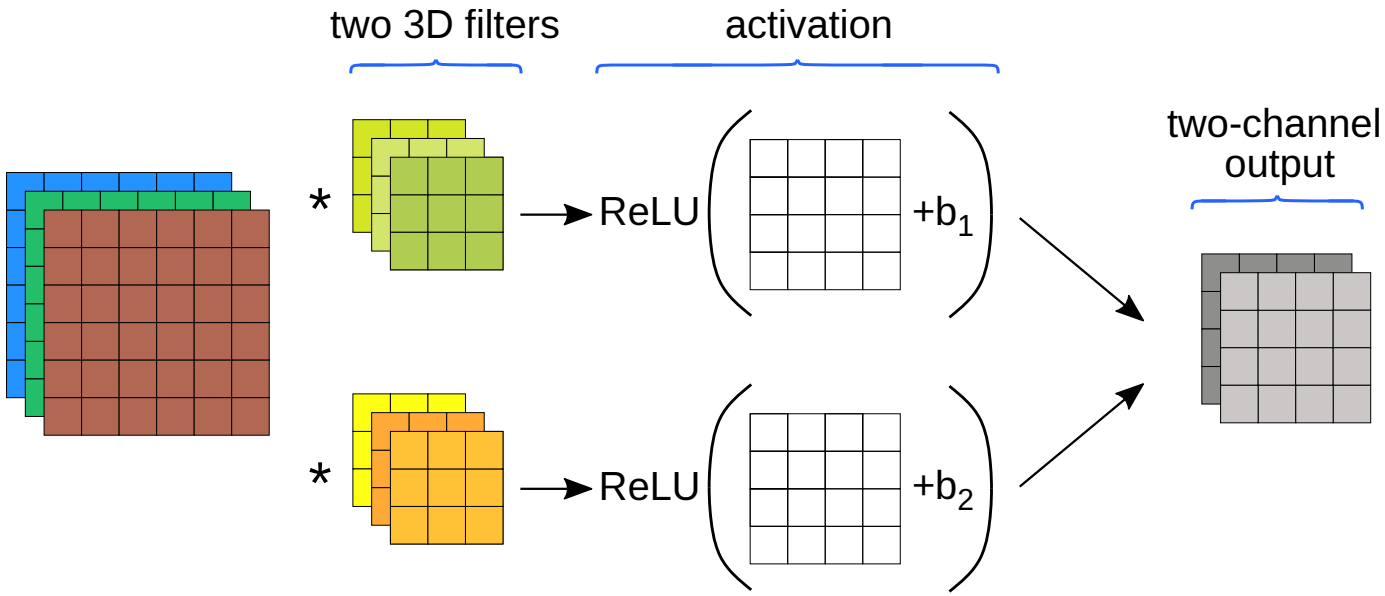
$$\varphi(z) = \max(0, z)$$

$$\varphi'(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{otherwise.} \end{cases}$$

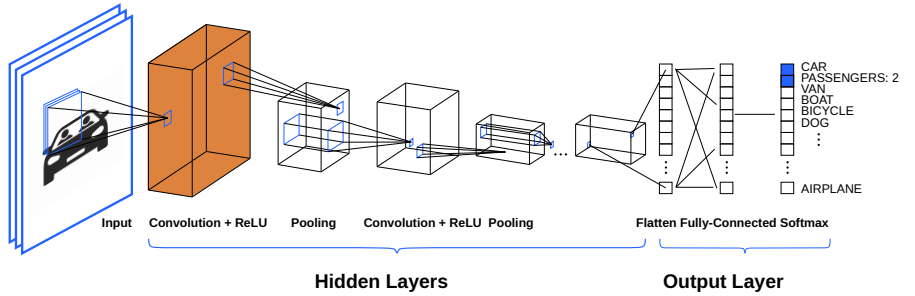
Convolution Layer



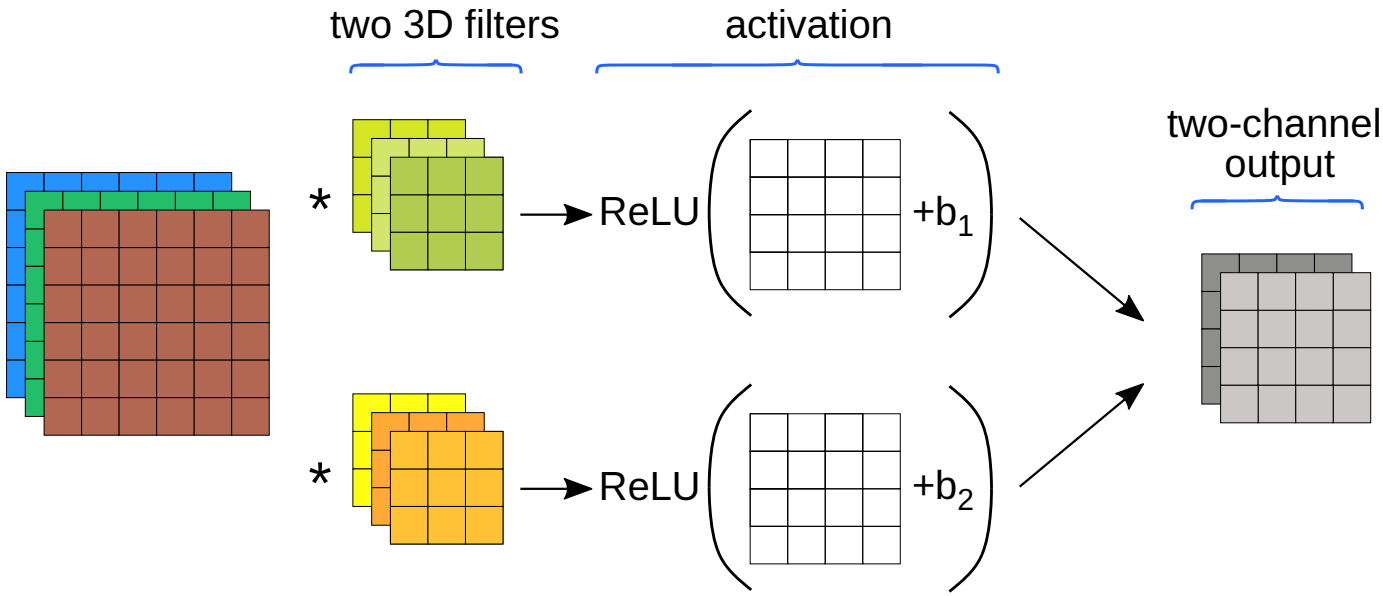
A hypothetical, simple Convolutional Layer:



Convolution Layer



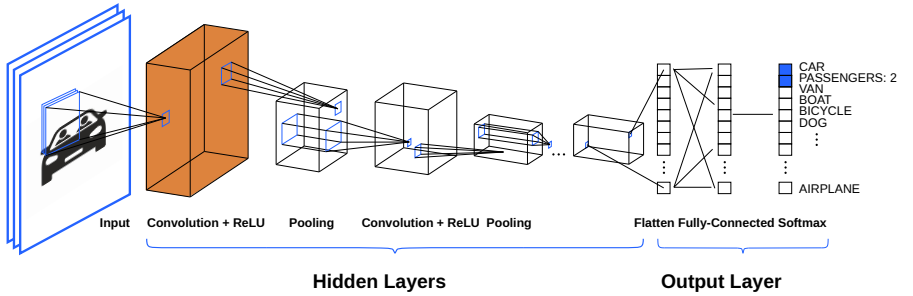
A hypothetical, simple Convolutional Layer:



The number of channels in a Convolutional Layer output equals the number of 3D (length $\times$ width $\times$ channels) filters used in the layer.

During the 3D convolutions, each output channel, as a 2D matrix, has additional bias broadcasted to each element, before a ReLU activation function is applied.

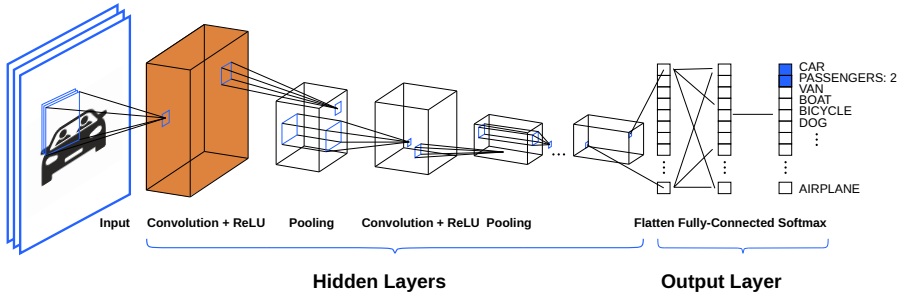
Convolution Layer



Three **hyperparameters** control the size of the output volume of the convolutional layer:

1. The **filter depth** affects the depth of the output. E.g.: three distinct filters would yield three different feature maps.

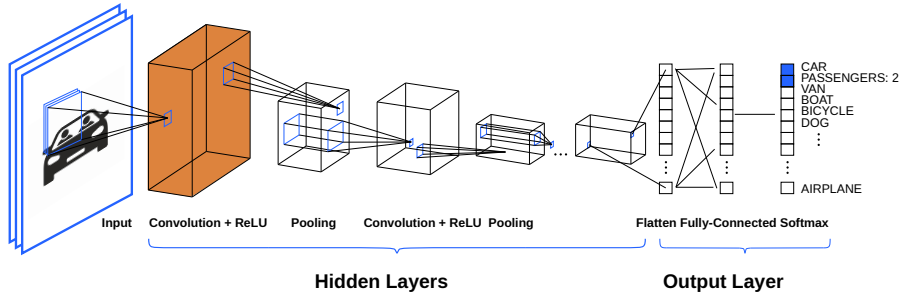
Convolution Layer



Three **hyperparameters** control the size of the output volume of the convolutional layer:

1. The **filter depth** affects the depth of the output. E.g.: three distinct filters would yield three different feature maps.
2. **Stride** is the number of pixels that the kernel moves over the input matrix. A larger stride means smaller overlap of receptive fields and smaller spatial dimensions of the output volume.

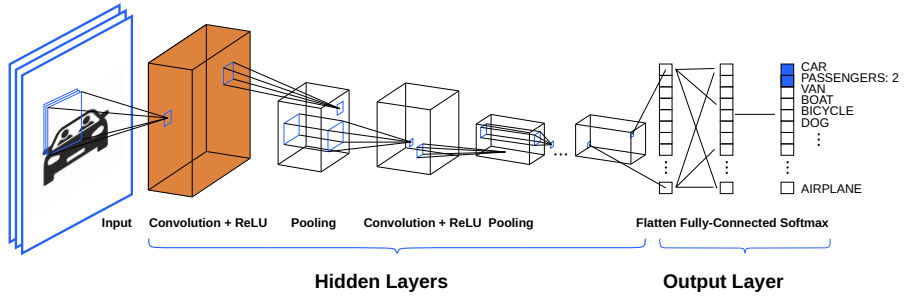
Convolution Layer



Three **hyperparameters** control the size of the output volume of the convolutional layer:

1. The **filter depth** affects the depth of the output. E.g.: three distinct filters would yield three different feature maps.
2. **Stride** is the number of pixels that the kernel moves over the input matrix. A larger stride means smaller overlap of receptive fields and smaller spatial dimensions of the output volume.
3. **Zero-padding** is usually used when the filters do not fit the input image. This sets all elements that fall outside of the input matrix to zero.

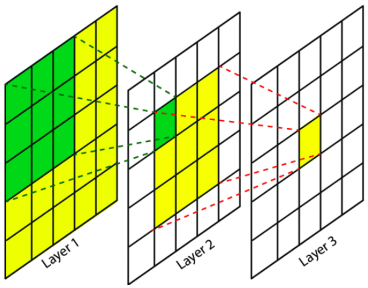
Convolution Layer



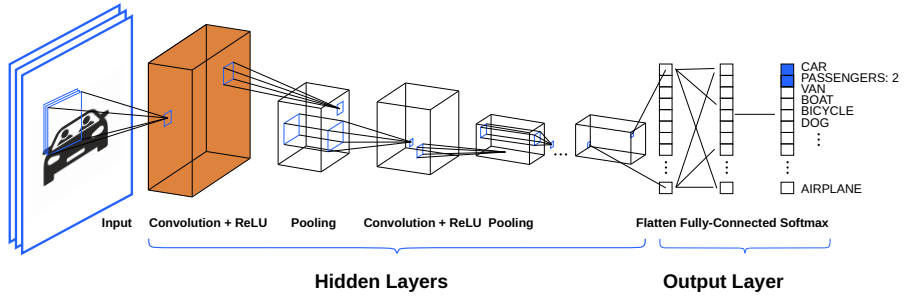
Three **hyperparameters** control the size of the output volume of the convolutional layer:

1. The **filter depth** affects the depth of the output. E.g.: three distinct filters would yield three different feature maps.
2. **Stride** is the number of pixels that the kernel moves over the input matrix. A larger stride means smaller overlap of receptive fields and smaller spatial dimensions of the output volume.
3. **Zero-padding** is usually used when the filters do not fit the input image. This sets all elements that fall outside of the input matrix to zero.

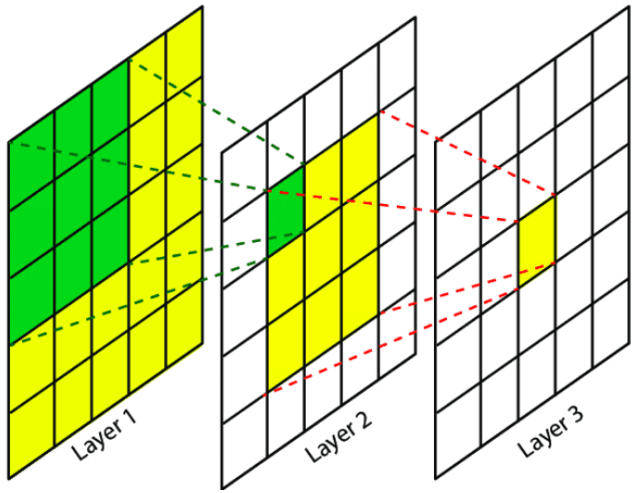
⇨ They define the **Receptive Field**



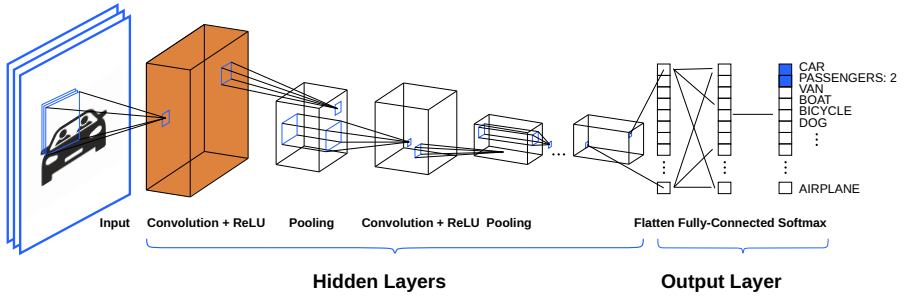
Convolution Layer



The Receptive Field is defined as the region in the input space that a particular CNN's feature is looking at (i.e. is affected by). A receptive field of a feature can be described by its center location and its size.

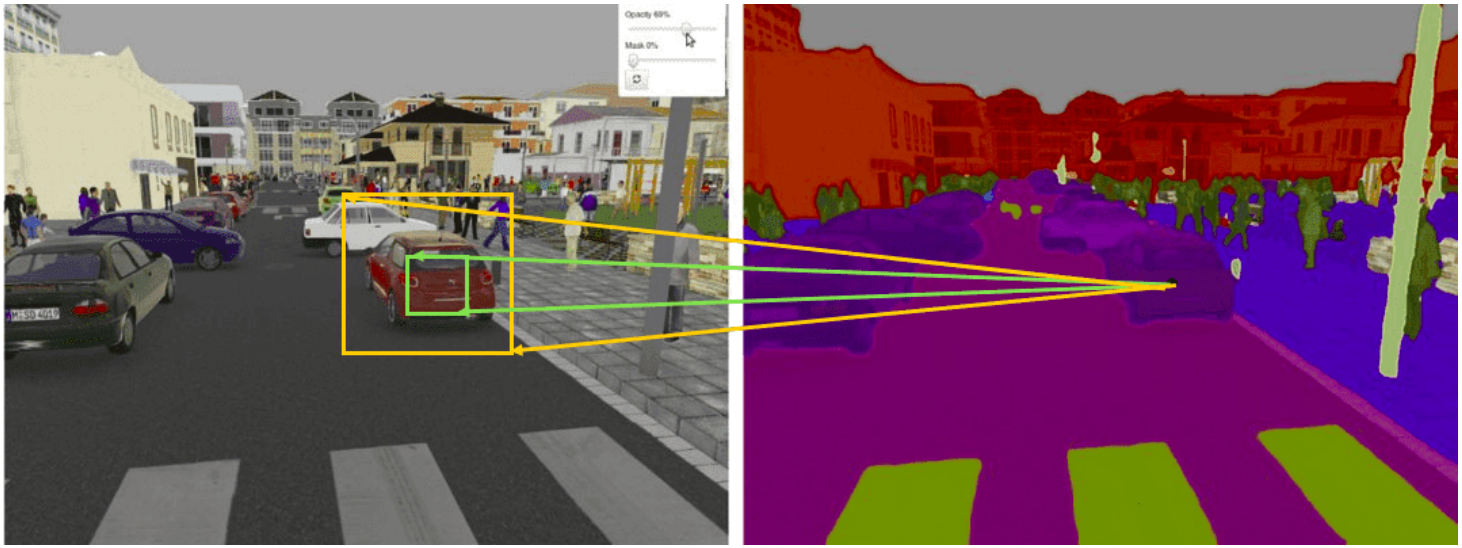


Convolution Layer



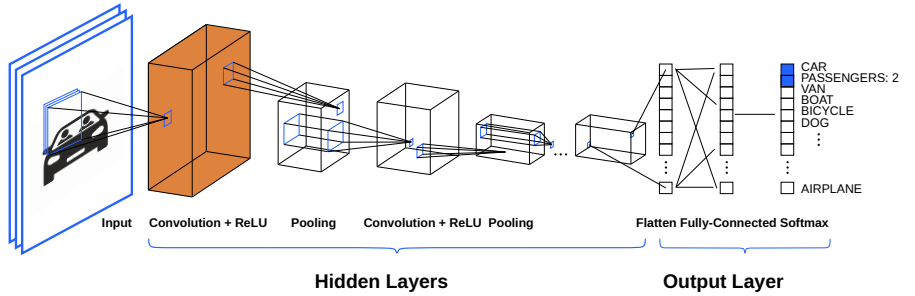
Why do we care about the receptive field of a CNN?

In computer vision, ideally, we would like each output pixel of the label map to have a big receptive field, so as to ensure that no crucial information was not taken into account. For instance, for predicting the boundaries of an object it is important to provide the model access to all relevant parts of the input.
The image below shows two receptive fields (green, orange). Which one would you like to have in your CNN?



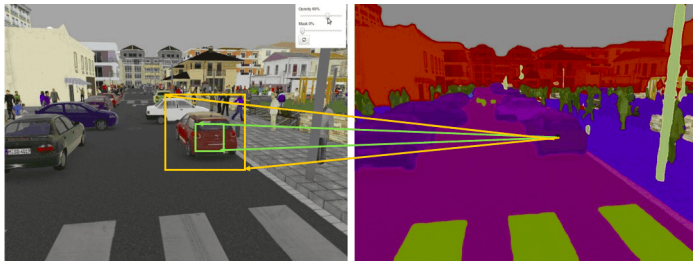
credit: Nvidia

Convolution Layer



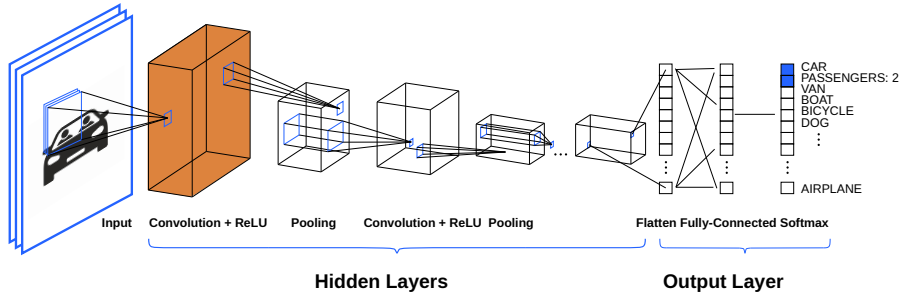
Why do we care about the receptive field of a CNN?

In computer vision, ideally, we would like each output pixel of the label map to have a big receptive field, so as to ensure that no crucial information was not taken into account. For instance, for predicting the boundaries of an object it is important to provide the model access to all relevant parts of the input.
The image below shows two receptive fields (green, orange). Which one would you like to have in your CNN?

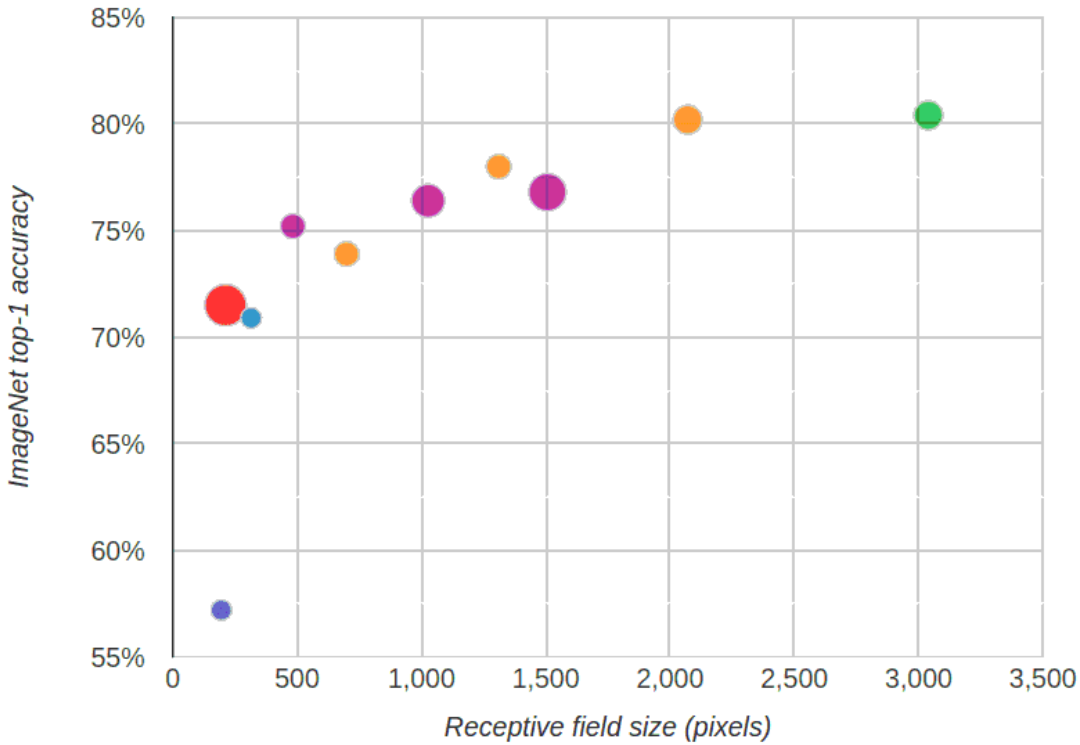


Goal: design a convolutional model so that its receptive field covers the entire **relevant** input image region.

Convolution Layer

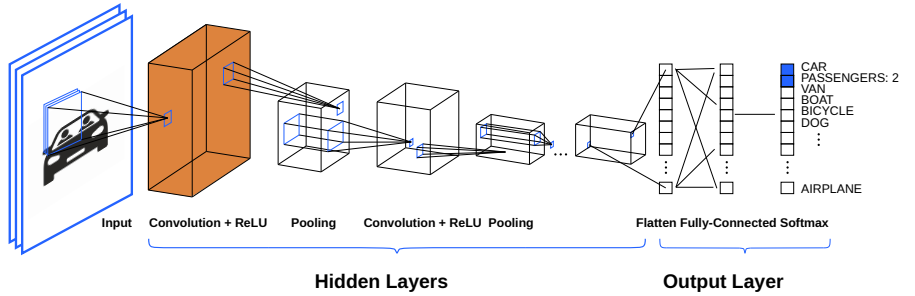


Why do we care about the receptive field of a CNN?

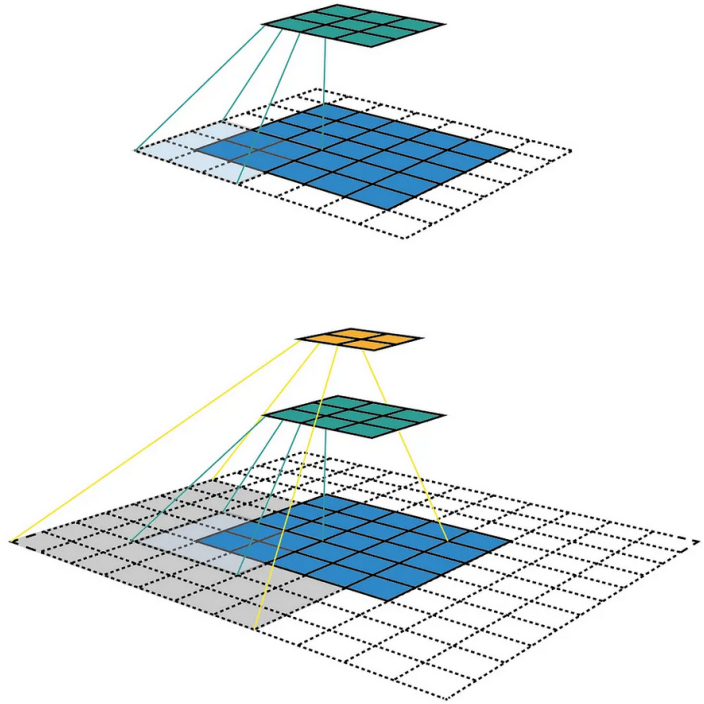


Relation of the receptive field to the classification accuracy in ImageNet. The radius refers to the amount of floating-point operations (FLOPs) of each model. The colors refer to different CNN architectures.
credit: Araujo, A., Norris, W., & Sim, J. (2019)

Convolution Layer

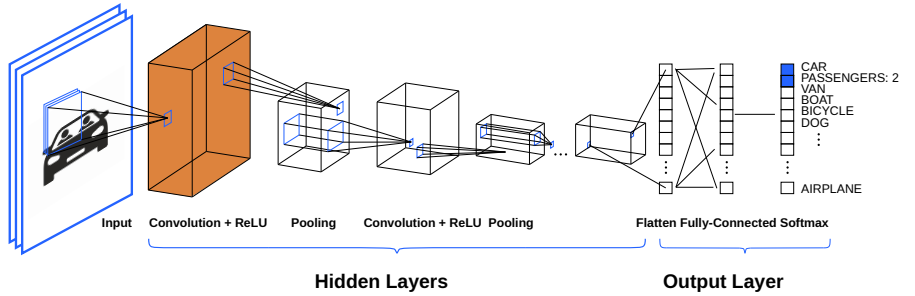


Visualizing the receptive field of a CNN

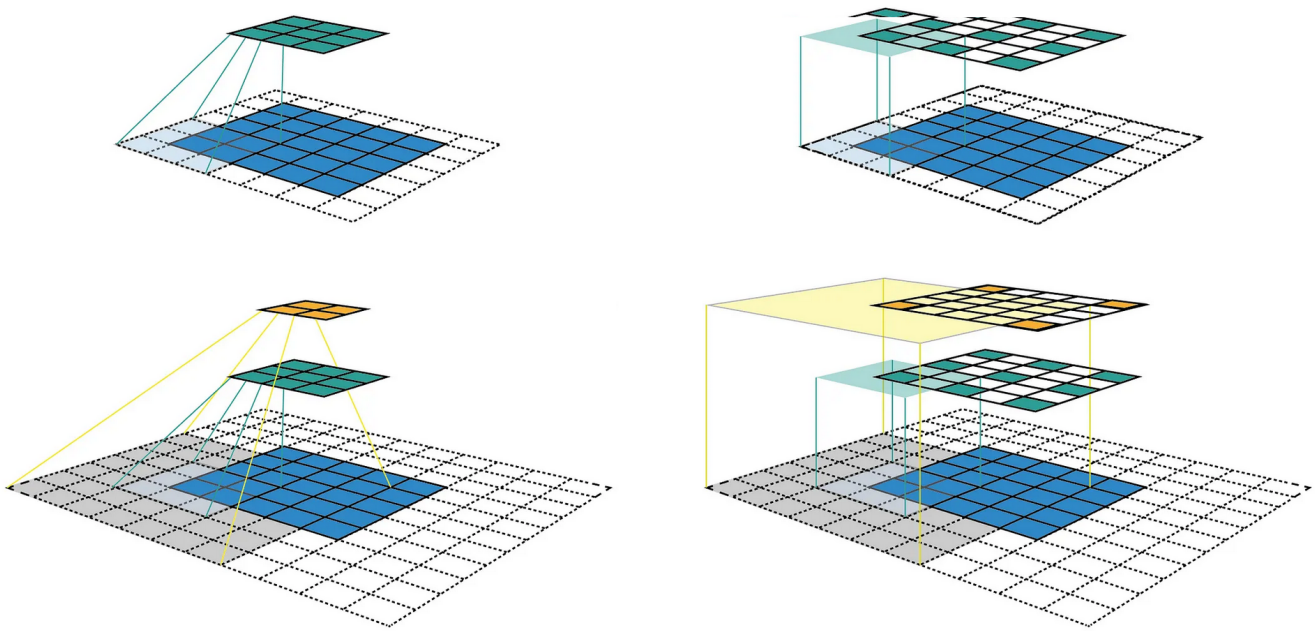


This figure shows two receptive field examples. By applying a convolution C with kernel size $F = 3 \times 3$, padding size $P = 1 \times 1$, stride $S = 2 \times 2$ on an input map of size 5×5, we will get an output feature map of size 3×3 (green map). Applying the same convolution on top of the 3×33 feature map, we will get a 2×2 feature map (orange map).

Convolution Layer



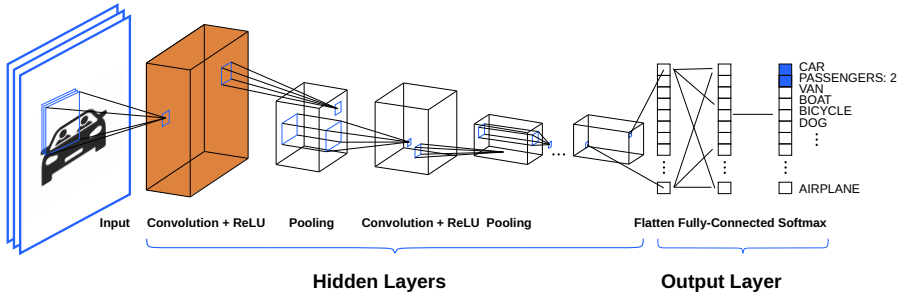
Visualizing the receptive field of a CNN



Left: The common way of visualization doesn't tell us location and size of a feature (the center location and size of its receptive field), making it impossible to keep track of receptive field information in a deep CNN.

Right: The fixed-sized CNN feature map visualization, with the size of each feature map is fixed, and the feature is at the center of its receptive field.

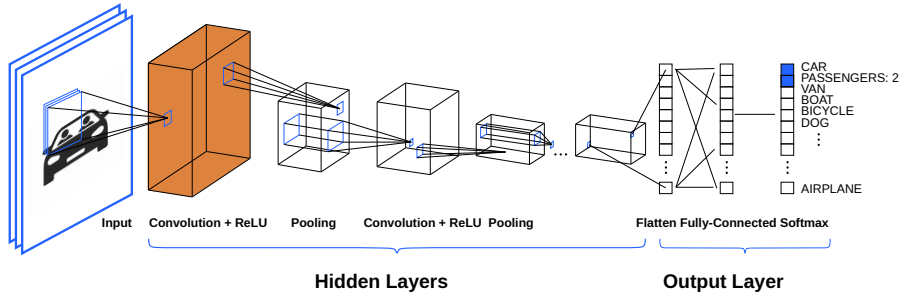
Convolution Layer



Receptive Field Arithmetic

To calculate the **properties of the receptive field** in each layer, we need the following information:

- n_{in} number of input features
- r current receptive field size
- j distance between two adjacent features (jump)
- F filter (kernel) size
- P padding size
- S stride size
- $start_{in}$ center position of the input



Receptive Field Arithmetic

To calculate the **properties of the receptive field** in each layer, we need the following information:

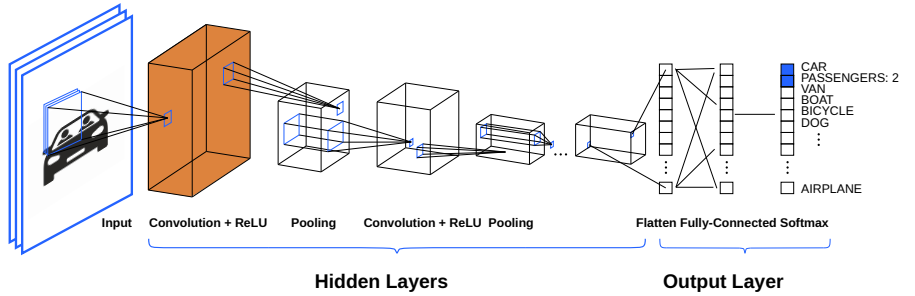
n_{in} number of input features
r current receptive field size
j distance between two adjacent features (jump)
F filter (kernel) size
P padding size
S stride size
$start_{in}$ center position of the input

the **floor** operator: the smallest integer less than or equal to the input

With this, we get:

- **The number of output features**:

$$n_{out} = \left\lfloor \frac{n_{in} + 2P - F}{S} \right\rfloor + 1$$



Receptive Field Arithmetic

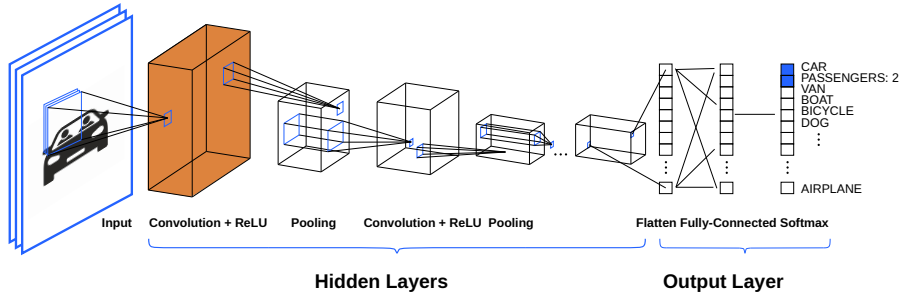
To calculate the **properties of the receptive field** in each layer, we need the following information:

n_{in} number of input features
r current receptive field size
j distance between two adjacent features (jump)
F filter (kernel) size
P padding size
S stride size
$start_{in}$ center position of the input

With this, we get:

- **The jump in the output feature map**, which is equal to the jump in the input map times the number of input features that you jump over when applying the convolution (the stride size S):

$$j_{out} = j_{in} \times S$$



Receptive Field Arithmetic

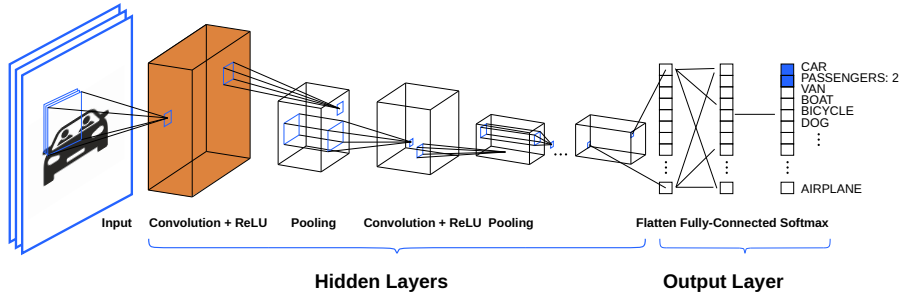
To calculate the **properties of the receptive field** in each layer, we need the following information:

n_{in} number of input features
r current receptive field size
j distance between two adjacent features (jump)
F filter (kernel) size
P padding size
S stride size
$start_{in}$ center position of the input

With this, we get:

- **The receptive field size of the output feature map**, which is equal to the area that covered by F input features $(F-1) \times j_{in}$ plus the extra area covered by the receptive field on the border:

$$r_{out} = r_{in} + (F-1) \times j_{in}$$



Receptive Field Arithmetic

To calculate the **properties of the receptive field** in each layer, we need the following information:

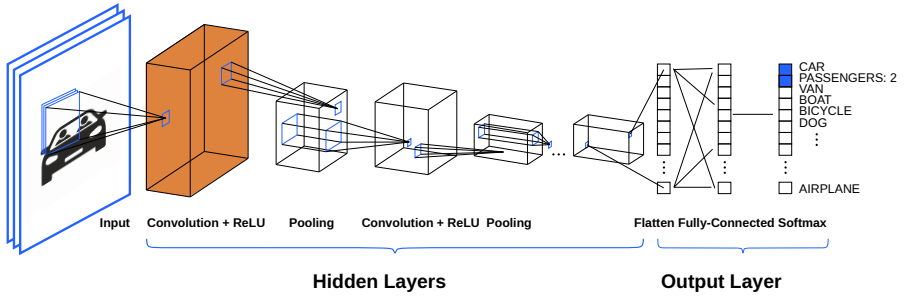
n_{in} number of input features
r current receptive field size
j distance between two adjacent features (jump)
F filter (kernel) size
P padding size
S stride size
$start_{in}$ center position of the input

With this, we get:

- **The receptive field center position** of the first output feature:

$$start_{out} = start_{in} + ((F-1)/2 - P) \times j_{in}$$

This is equal to $start_{in}$ plus the distance from the location of the first input feature to the center of the first convolution $(F-1)/2 \times j_{in}$ minus the padding space $P \times j_{in}$.

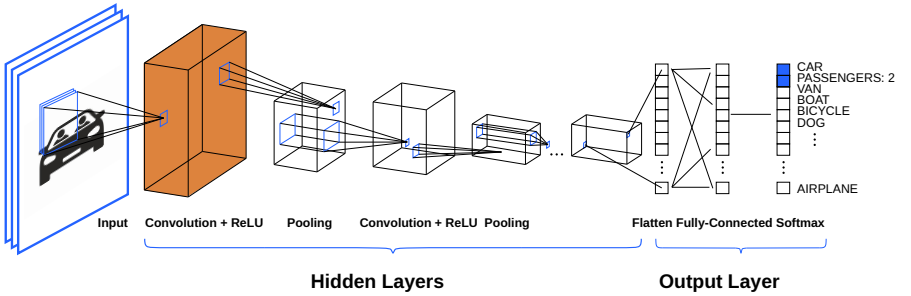


Receptive Field Arithmetic

By applying these equations recursively, we can calculate the receptive field information for all feature maps in a CNN. The first layer is always the input layer which has n_{in} equal to the image size, $r = 1$, $start_{in} = 0.5$.

Araujo et al. (2019) provide an **intuitive analytic way** to calculate the receptive field of a CNN.
We want to see the influence of the receptive field starting from the last layer towards the input. So, in that sense, we go backwards.

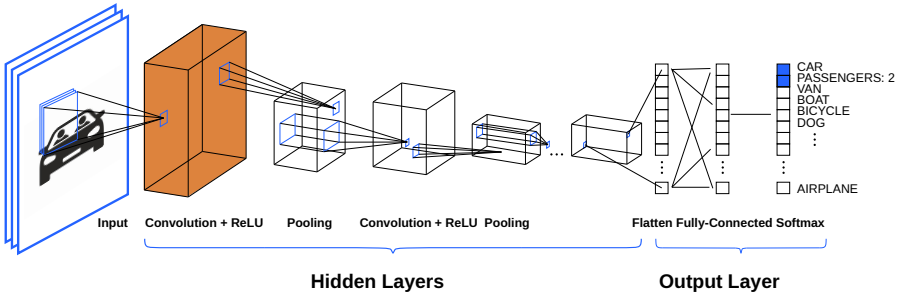
Convolution Layer



Receptive Field Arithmetic

For two sequential convolutional layers f_2, f_1 with kernel size k, stride s, receptive field r:

$$r_1 = s_2 \times r_2 + (k_2 - s_2)$$



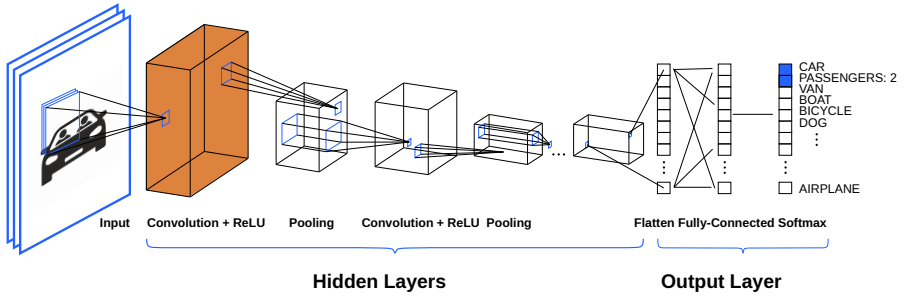
Receptive Field Arithmetic

For two sequential convolutional layers f_2, f_1 with kernel size k, stride s, receptive field r:

$$r_1 = s_2 \times r_2 + (k_2 - s_2)$$

Or in a more general form:

$$r_{(i-1)} = s_i \times r_i + (k_i - s_i)$$



Receptive Field Arithmetic

For two sequential convolutional layers f_2, f_1 with kernel size k, stride s, receptive field r:

$$r_1 = s_2 \times r_2 + (k_2 - s_2)$$

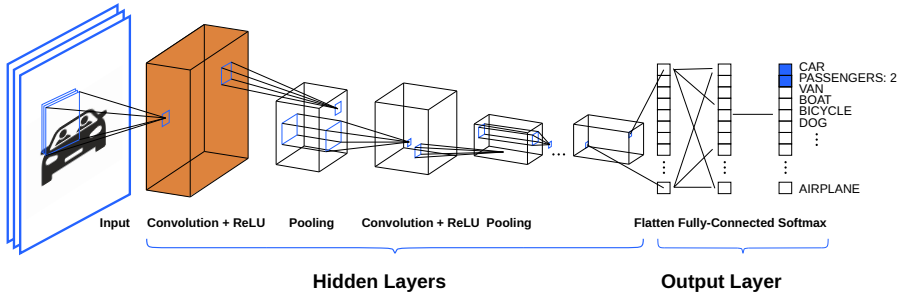
Or in a more general form:

$$r_{(i-1)} = s_i \times r_i + (k_i - s_i)$$

This equation can even more be generalized to apply the operation **recursively** for L layers. By further analyzing the recursive equation, we can derive a closed form solution for the desired receptive field r_0 that depends only on the convolutional parameters of kernels and strides:

$$r_0 = \sum_{i=1}^{L} \left((k_i - 1) \prod_{j=1}^{i-1} s_j \right) + 1$$

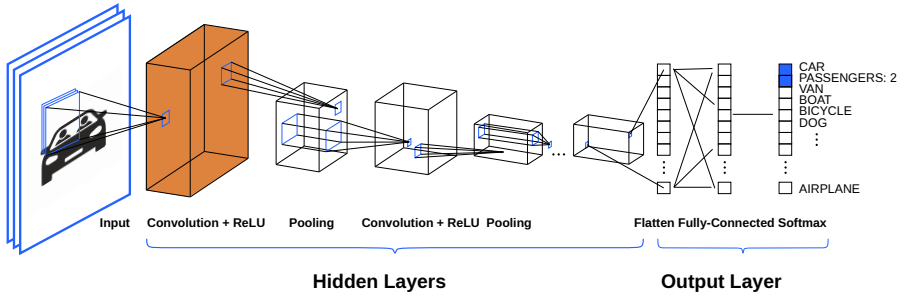
Convolution Layer



Optimizing the Receptive Field

How can we **increase** the receptive field in a convolutional network?

Convolution Layer

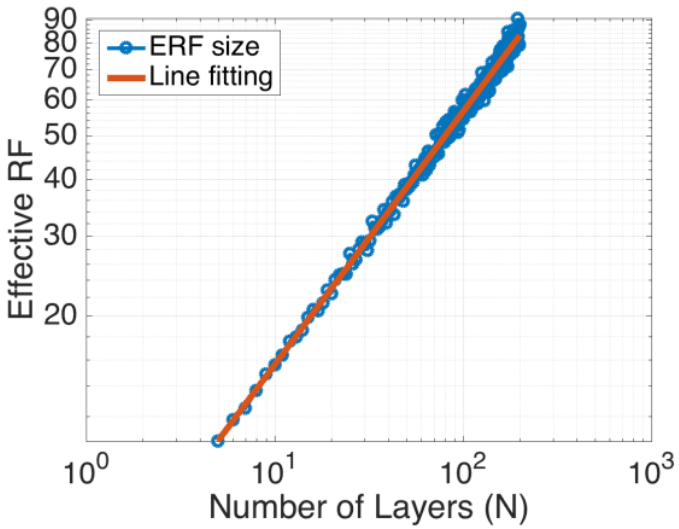


Optimizing the Receptive Field

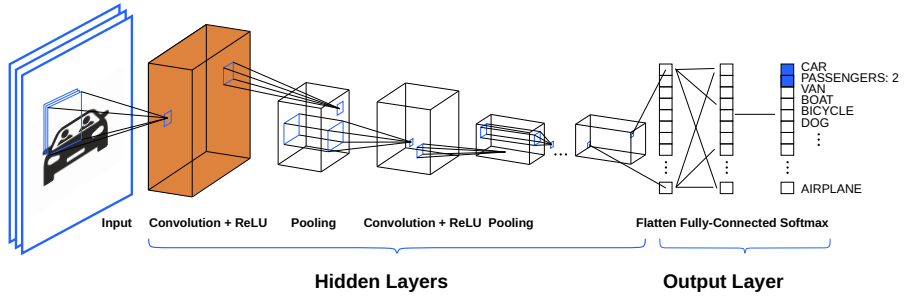
How can we **increase** the receptive field in a convolutional network?

- the most obvious one: Add more convolutional layers (make the network deeper)

Increasing the number of convolutional layers increases the receptive field size linearly: Each extra layer adds the kernel size to the receptive field size.



Credit: Luo et al. (2016)



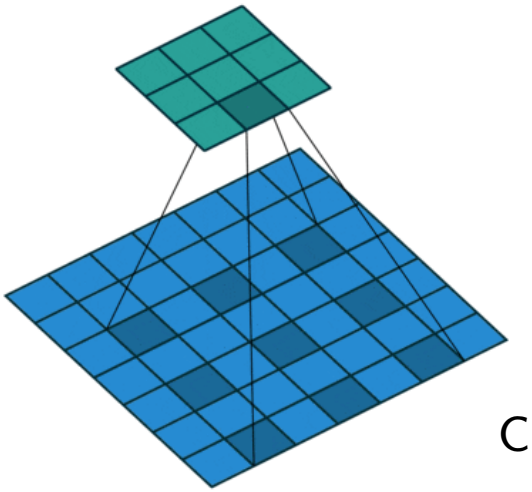
Optimizing the Receptive Field

How can we **increase** the receptive field in a convolutional network?

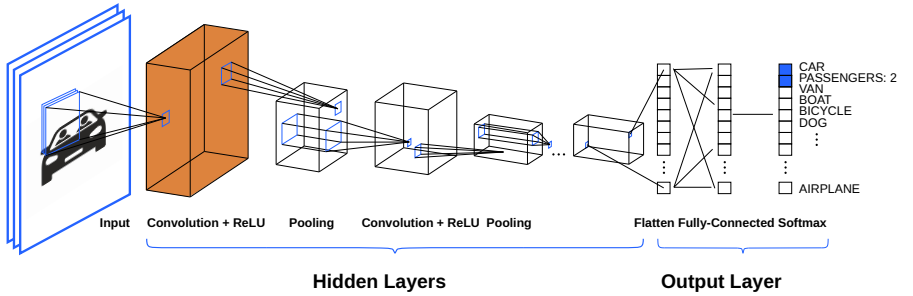
- the more sophisticted one: Use **dilated convolutions**

Dilations define a spacing between values in the convolutional kernel values by a dilation rate r. While the number of weights is unchanged, they are no longer applied to spatially adjacent samples.
The pre-described equations can be reused by replacing the kernel size k for all layers using dilations:

$$k' = r(k-1)+1$$



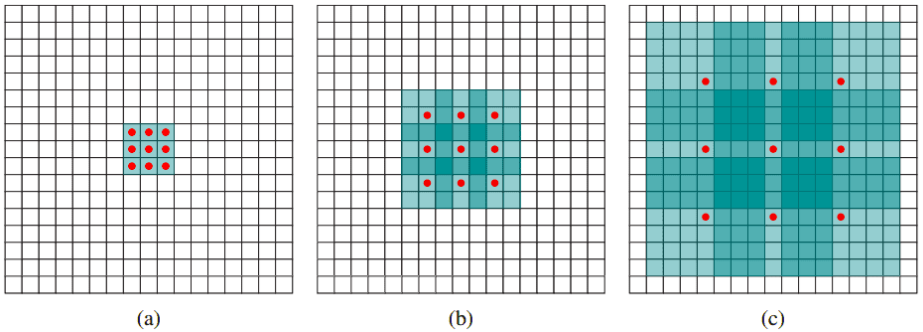
Credit: Dumoulin et al. (2016)



Optimizing the Receptive Field

How do **dilated convolutions** influence the receptive field?

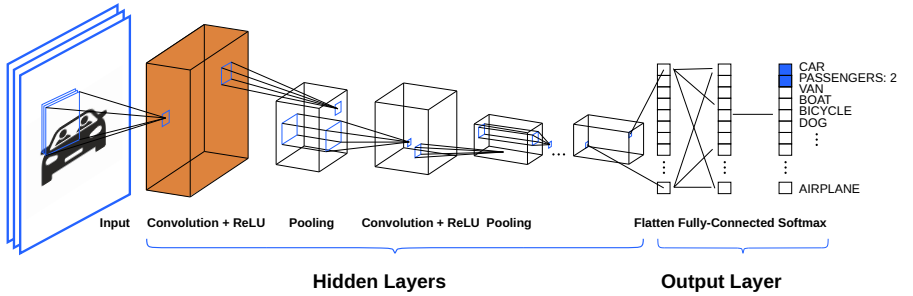
Let's take a look at 3 sequential convolutional layers with a) normal convolution, b) dilation factor $r = 2$, c) dilation factor $r = 4$.



Credit: Yu et al. (2015)

In (a) we have a receptive field 3×3.

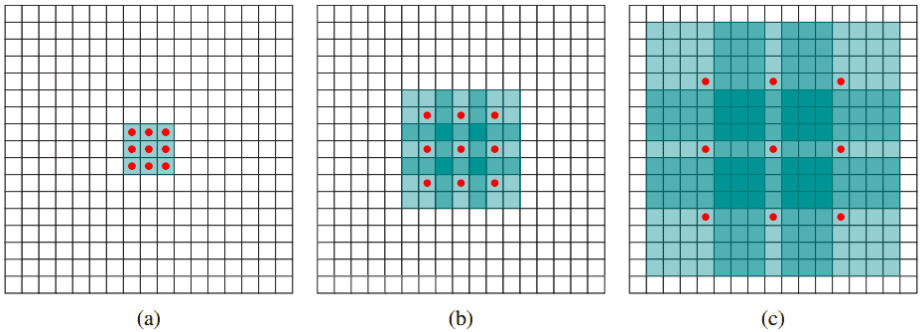
Convolution Layer



Optimizing the Receptive Field

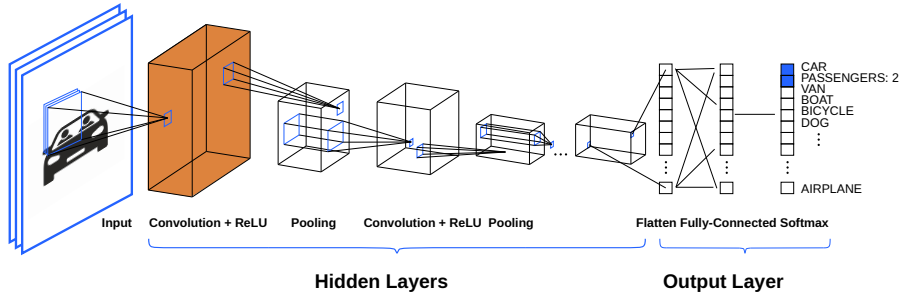
How do **dilated convolutions** influence the receptive field?

Let's take a look at 3 sequential convolutional layers with a) normal convolution, b) dilation factor $r = 2$, c) dilation factor $r = 4$.



Credit: Yu et al. (2015)

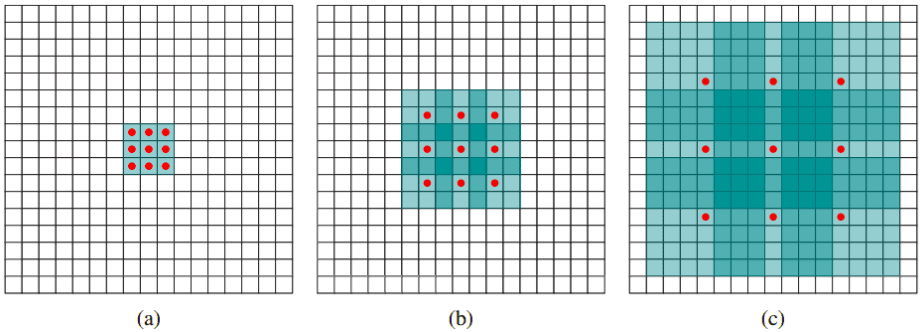
In (b) we have a 2-dilated 3×3 convolution that is applied in the output of layer (a). As a result, each element in the 2 coupled layers now has a receptive field of 7×7. If we studied 2-dilated convolution alone the receptive field would be 5×5 with the same number of parameters.



Optimizing the Receptive Field

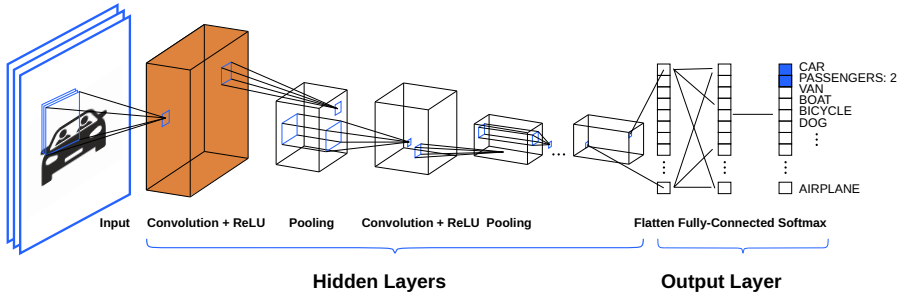
How do **dilated convolutions** influence the receptive field?

Let's take a look at 3 sequential convolutional layers with a) normal convolution, b) dilation factor $r = 2$, c) dilation factor $r = 4$.



Credit: Yu et al. (2015)

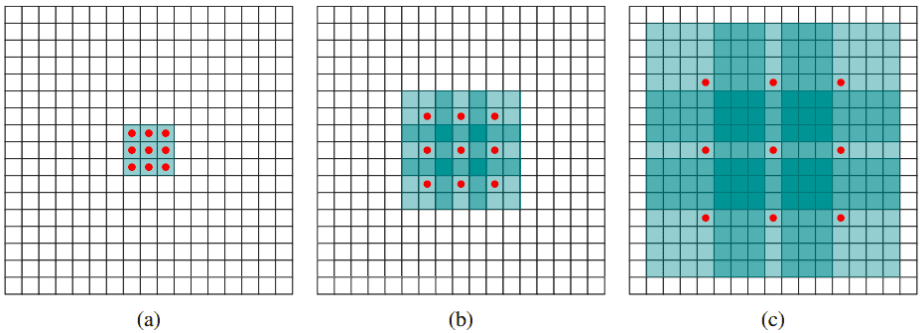
In (c) by applying a 4-dilated convolution, each element in the third sequential conv layer now has a receptive field of 15×15 - an exponential grow of the receptive field while the number of parameters grows linearly.



Optimizing the Receptive Field

How do **dilated convolutions** influence the receptive field?

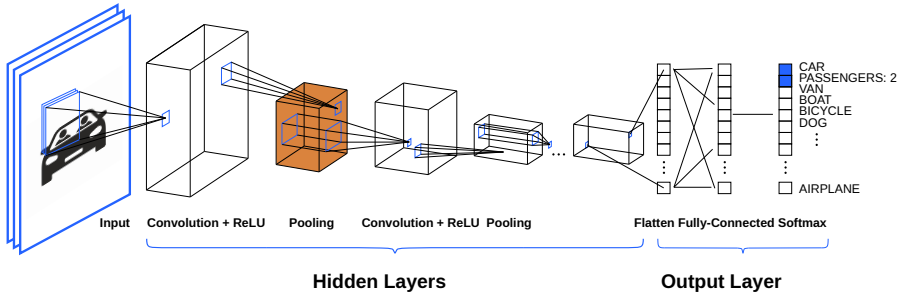
Let's take a look at 3 sequential convolutional layers with a) normal convolution, b) dilation factor $r = 2$, c) dilation factor $r = 4$.



Credit: Yu et al. (2015)

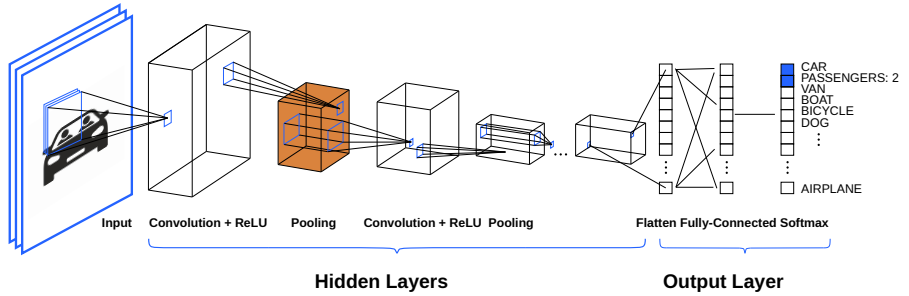
A 3×3 kernel with a dilation rate of 2 will have the same receptive field as a 5×5 kernel, ut using 9 parameters. Similarly, a 3×3 kernel with a dilation rate of 4 will have the same receptive field as a 9×9 kernel without dilation.
Mathematically: $r(k-1)+1 = k_{prev}$

Pooling Layer



Each Convolutional Layer is followed by a Pooling Layer.
Similar to the Convolution Layer, the Pooling Layer reduces the spatial size of the Convolved Feature. It suppresses noise and extracts dominant features.

Pooling Layer

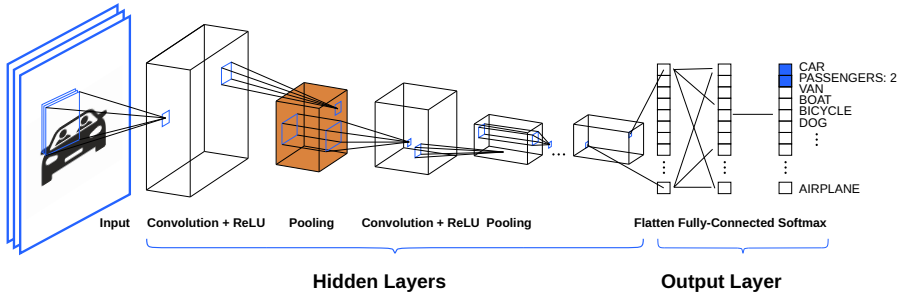


Each Convolutional Layer is followed by a Pooling Layer.
Similar to the Convolution Layer, the Pooling Layer reduces the spatial size of the Convolved Feature. It suppresses noise and extracts dominant features.

In detail, a pooling layer carries out the following tasks:

- Extract the most important (relevant) features by getting the maximum number or averaging the numbers.
- Reduce the dimensionality (number of pixels) of the output returned from previous convolutional layers.
- Reduce the number of parameters in the network.
- Remove any noise present in the features extracted by previous convolutional layers.
- Increase the accuracy of CNNs.

Pooling Layer



The pooling layer contains three elements: Feature Map, Filter and Pooled Feature Map.
The **Pooling Operation** happens between a section of the feature map and the filter. It outputs the pooled feature map which is a downsampled version of the feature map.

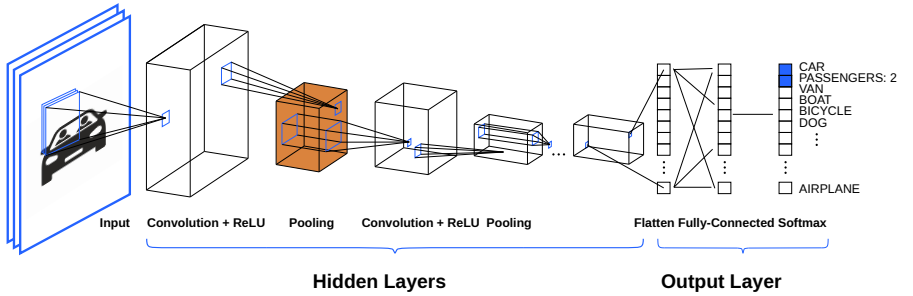
Example: Maximum Pooling here: with a 2×2 filter and $S = 2$

7	1	5	8
4	6	5	1
9	0	7	5
3	2	1	0

→

7	8
9	7

Pooling Layer



alternatives:

p-Norm Pooling

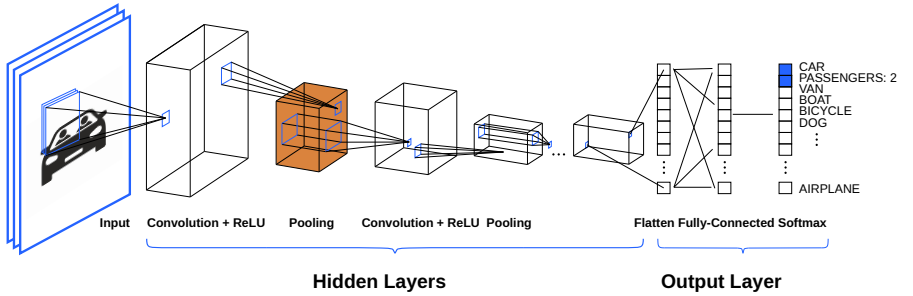
here: with a 2×2 filter and $S = 2$, $p = 5$

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

$$\sqrt[p]{\frac{1}{p^2}\sum_{i,j} x_{ij}^p}$$

4.86	8
2.38	3.16

Pooling Layer



alternatives:

p-Norm Pooling

here: with a 2×2 filter and $S = 2$, $p = 5$

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

$$\sqrt[p]{\frac{1}{p^2}\sum_{i,j} x_{ij}^p}$$

4.86	8
2.38	3.16

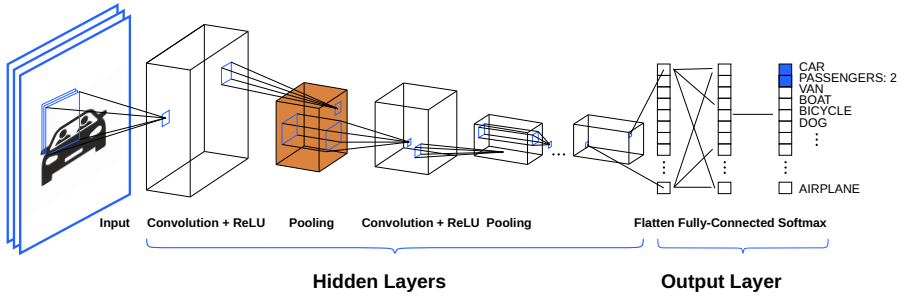
Average Pooling

here: with a 2×2 filter and $S = 2$

7	1	5	8
4	6	5	1
9	0	7	5
3	2	1	0

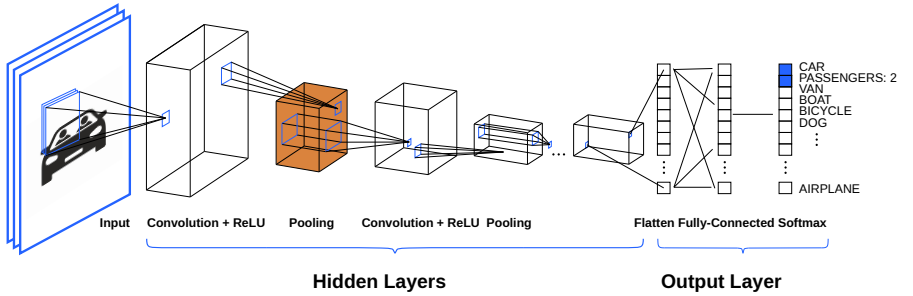
4.5	4.75
3.5	3.25

Pooling Layer



Other than convolutional filters which are learned, pooling filters are typically not learned.

Pooling Layer

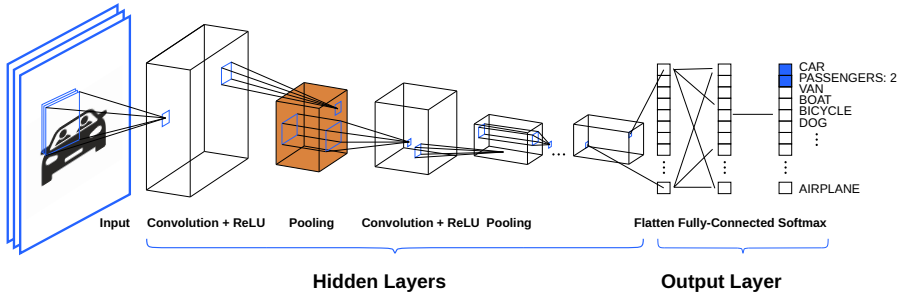


Other than convolutional filters which are learned, pooling filters are typically not learned.

But it is still possible to do so!

In this case, the same network is applied on each block when moving the pooling filter.

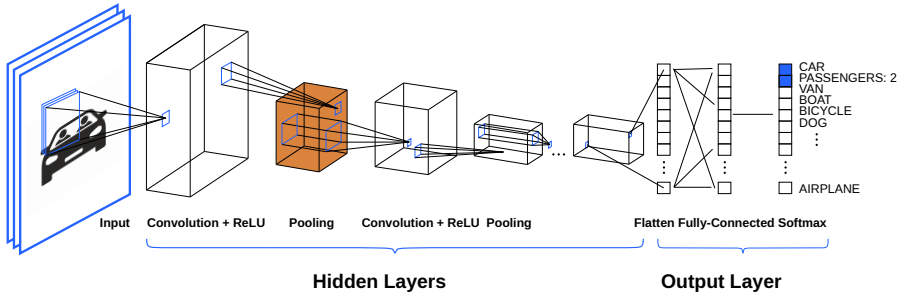
Pooling Layer



Which pooling should we chose?

The most common pooling layer is **Max Pooling**, with a pooling window size of 2 and stride of 2 - resulting in a output dimension that is halved. By picking the most activated value in the pooling window, the Max Pooling layer removes redundant features and helps in preventing over-fitting.

Pooling Layer

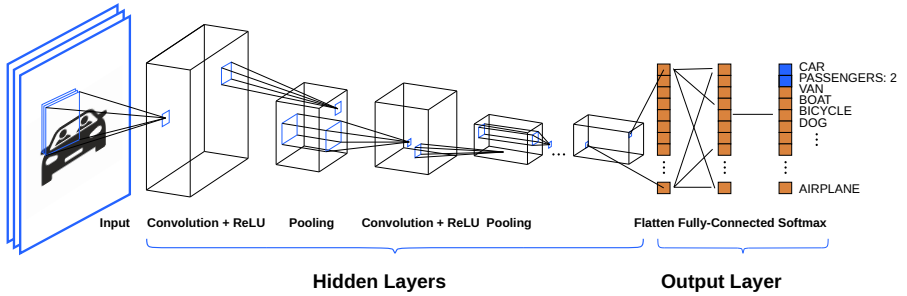


Which pooling should we chose?

The most common pooling layer is **Max Pooling**, with a pooling window size of 2 and stride of 2 - resulting in a output dimension that is halved. By picking the most activated value in the pooling window, the Max Pooling layer removes redundant features and helps in preventing over-fitting.

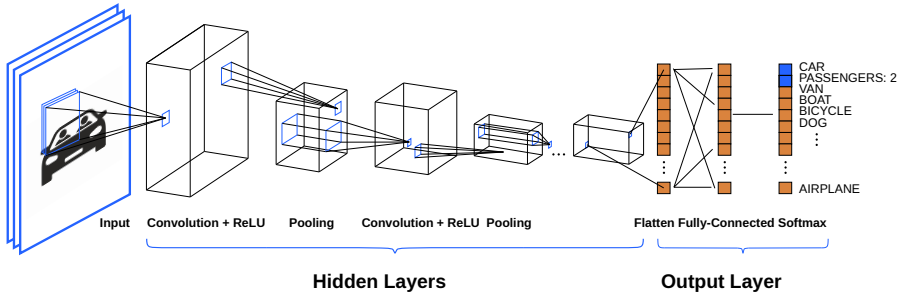
On the other hand, **Average Pooling** typically happens between the Convolutional Layers and the Fully Connected Layers, in place of the Flatten operation. Average Pooling is sometimes preferred compared to the Flatten operation due to it having less features passed into the Fully Connected Layers, and may reduce over-fitting.

Fully Connected Layers



The fully connected layers classify the detected features in the image into a class label. These are the final layers in a CNN.

Fully Connected Layers

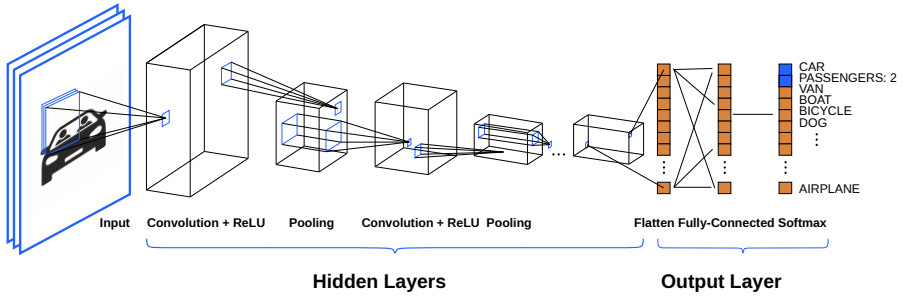


The fully connected layers classify the detected features in the image into a class label. These are the final layers in a CNN.

As input, the previous layer is flattened. There can be multiple fully connected layers. The final layer carries out the classification.

The ReLU activation is used in each fully connected layer except in the final layer in which we use the Softmax activation for multiclass classification.

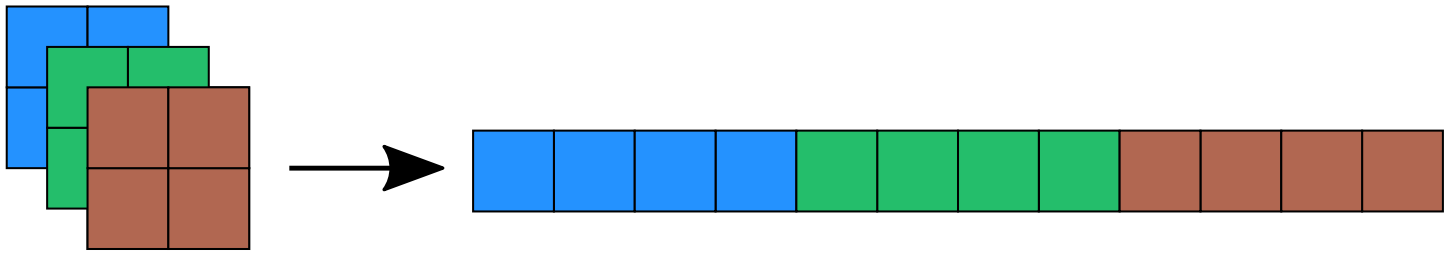
Fully Connected Layers



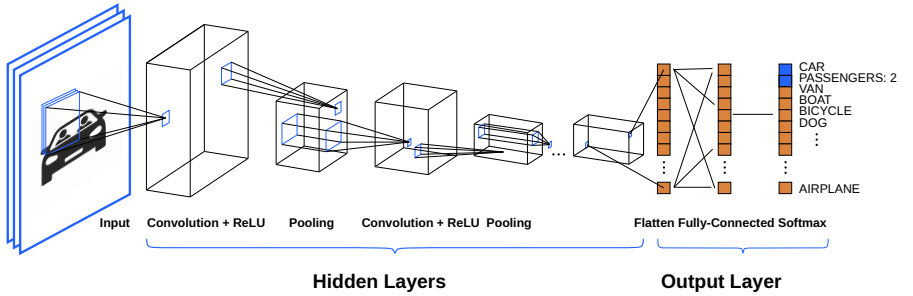
A **Flatten Operation** is necessary:

In a CNN, the final pooled feature map is fed to a Multilayer Perceptron (MLP) to classify the feature map it into a class label.

MPLs only accept one-dimensional data - this requires flattening the final pooled feature map into a single column. Unlike flattening the original image, important pixel dependencies are retained when pooled maps are flattened.



Fully Connected Layers

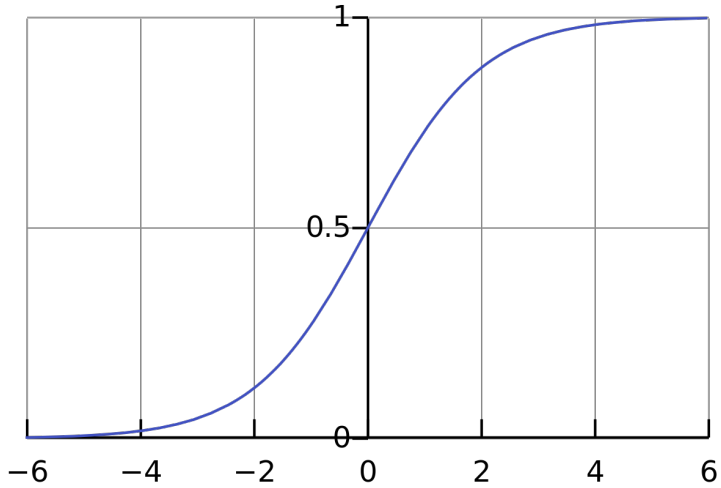


The Fully Connected Layers learn non-linear combinations of the high-level features as represented by the output of the convolutional layer.

Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular (non-convolutional) artificial neural networks.

The **activation function** is usually a softmax function:

output



input from previous layer

Summary: The Building Blocks of a CNN

The CNN is a built along how the vision is processed in mammals.

Summary: The Building Blocks of a CNN

The CNN is a built along how the vision is processed in mammals.

It includes:

- convolutional layers compromising learned filters that scan the outputs of previous layers
- pooling layers that downsample outputs from convolutional layers
- an output layer that works on flattened data from the last pooling layer.

Summary: The Building Blocks of a CNN

The CNN is a built along how the vision is processed in mammals.

It includes:

- convolutional layers compromising learned filters that scan the outputs of previous layers
- pooling layers that downsample outputs from convolutional layers
- an output layer that works on flattened data from the last pooling layer.

Convolution can change the size of the output. This can be controlled with zero padding.

Summary: The Building Blocks of a CNN

The CNN is a built along how the vision is processed in mammals.

It includes:

- convolutional layers compromising learned filters that scan the outputs of previous layers
- pooling layers that downsample outputs from convolutional layers
- an output layer that works on flattened data from the last pooling layer.

Convolution can change the size of the output. This can be controlled with zero padding.

Pooling layers may perform operations such as max, p-norm, or learned downsampling networks.

Training CNN

As in the case of regular neural networks, training happens through backpropagation. The only difference comes from the **structure** of the neural network.

Training CNN

As in the case of regular neural networks, training happens through backpropagation. The only difference comes from the **structure** of the neural network.

Again, as in the case of a regular neural network:

- a training sample is provided
- the difference between the desired ouput and the true output is defined and calculated in response to any output of the training sample
- network parameters are trained through backpropagation with gradient descent

Training CNN

CNNs give us two key benefits: local invariance and compositionality.

Training CNN

CNNs give us two key benefits: local invariance and compositionality.

In the final **multi-layer perceptron**, all weights and biases for classification are learned to make predictions based on detected higher-level features.

Training CNN

CNNs give us two key benefits: local invariance and compositionality.

In the final **multi-layer perceptron**, all weights and biases for classification are learned to make predictions based on detected higher-level features.

In the **convolutional layers**, the filters must learned. In the context of image classification, our CNN may learn to:

- Detect edges from raw pixel data in the first layer.
- Use these edges to detect shapes (i.e., *blobs*) in the second layer.
- Use these shapes to detect higher-level features such as car parts, lensed galaxies etc.

Training CNN

CNNs give us two key benefits: local invariance and compositionality.

In the final **multi-layer perceptron**, all weights and biases for classification are learned to make predictions based on detected higher-level features.

In the **convolutional layers**, the filters must learned. In the context of image classification, our CNN may learn to:

- Detect edges from raw pixel data in the first layer.
- Use these edges to detect shapes (i.e., *blobs*) in the second layer.
- Use these shapes to detect higher-level features such as car parts, lensed galaxies etc.

The **number of parameters**:
Let each layer j have K_j maps.
Let the filters in the jth layer be of size $L_j \times L_j$
For the jth layer we will require $K_j(K_{j-1}L_j^2 + 1)$ filter parameters.
The total number of parameters for each convolutional layer is then

$$\sum_j K_j(K_{j-1}L_j^2 + 1)$$

Training CNN

Computing the Loss

Given a training set with input-output pairs:
$(x_1, \hat{y}_1), (x_2, \hat{y}_2), ..., (x_T, \hat{y}_T)$

The loss on the ith instance is $\Delta(x_i, \hat{y}_i)$.

The total loss is $E = \frac{1}{T} \sum_{i=1}^{T} \Delta(x_i, \hat{y}_i)$.

The loss is minimized w.r.t. the weights and biases θ

Training CNN

Computing the Loss

Given a training set with input-output pairs:
$(x_1, \hat{y}_1), (x_2, \hat{y}_2), ..., (x_T, \hat{y}_T)$

The loss on the ith instance is $\Delta(x_i, \hat{y}_i)$.

The total loss is $E = \frac{1}{T} \sum_{i=1}^{T} \Delta(x_i, \hat{y}_i)$.

The loss is minimized w.r.t. the weights and biases $\boldsymbol{\theta}$

Training through gradient descent

initialize all weights and biases $\boldsymbol{\theta}$

for every layer l for all filter indices m, until loss has converged update $\boldsymbol{\theta}_k \rightarrow \boldsymbol{\theta}_{k+1}$:

$$\boldsymbol{\theta}_{k+1}(l, m, j, x, y) = \boldsymbol{\theta}_k(l, m, j, x, y) - \eta \frac{\partial E}{\partial \boldsymbol{\theta}_k(l, m, j, x, y)}$$

Training CNN

Computing the Loss

Given a training set with input-output pairs:
$(x_1, \hat{y}_1), (x_2, \hat{y}_2), ..., (x_T, \hat{y}_T)$

The loss on the ith instance is $\Delta(x_i, \hat{y}_i)$.

The total loss is $E = \frac{1}{T} \sum_{i=1}^{T} \Delta(x_i, \hat{y}_i)$.

The loss is minimized w.r.t. the weights and biases $\boldsymbol{\theta}$

Training through gradient descent

more on this tomorrow

initialize all weights and biases $\boldsymbol{\theta}$

for every layer l for all filter indices m, until loss has converged update $\boldsymbol{\theta}_k \rightarrow \boldsymbol{\theta}_{k+1}$:

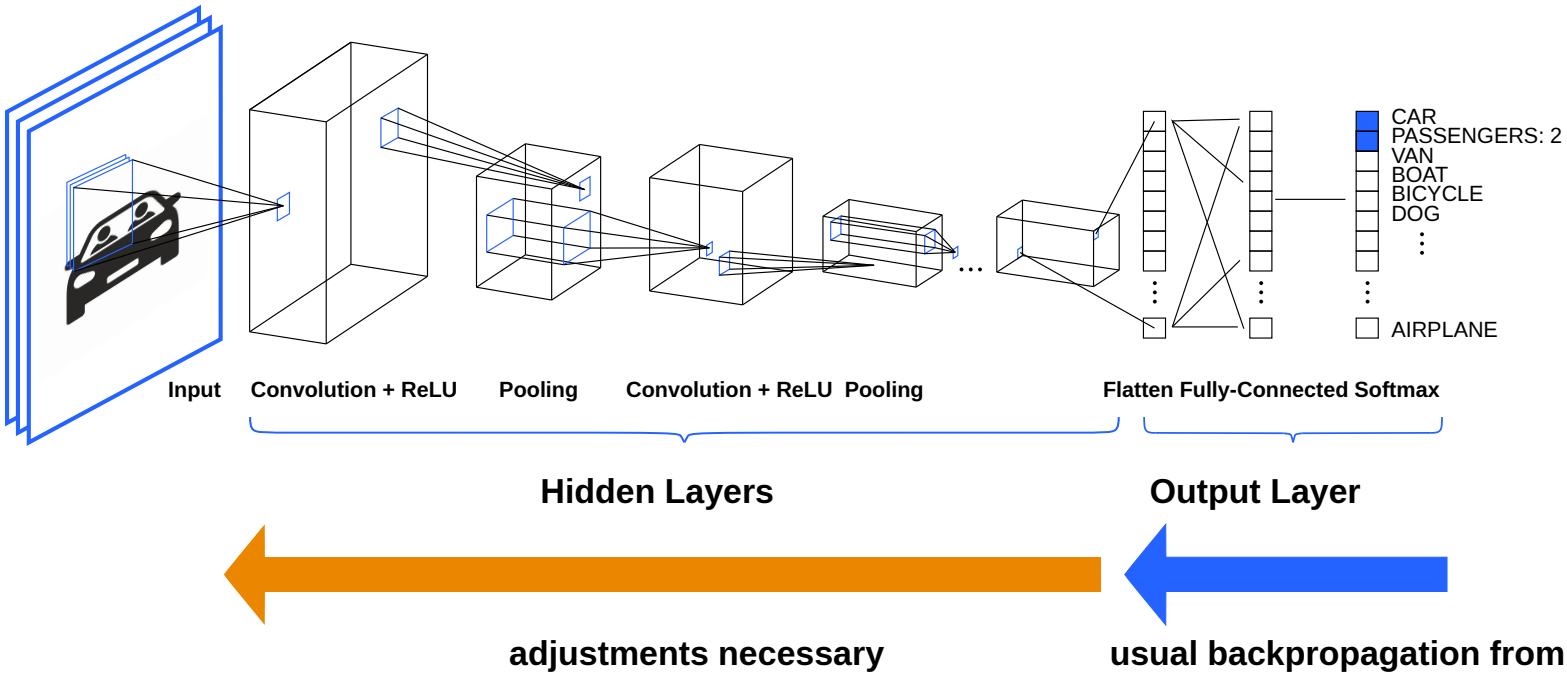
$$\boldsymbol{\theta}_{k+1}(l, m, j, x, y) = \boldsymbol{\theta}_k(l, m, j, x, y) - \eta \frac{\partial E}{\partial \boldsymbol{\theta}_k(l, m, j, x, y)}$$

Training CNN

Backpropagation continues in the usual manner until the computation of the derivative of the divergence w.r.t. its inputs to the first "flat" layer

Backpropagation from the flat MLP requires special consideration:

- the shared computation in the convolution layers
- the pooling layers



Training CNN

Both the Forward pass and the Backpropagation of a Convolutional layer are Convolutions.

So the gradients for backpropagation become, with filter F, layer output y_l and layer input $x = y_{l-1}$:

$$\frac{\partial E}{\partial F} = \text{Conv}\left(\text{input } x, \text{loss gradient } \frac{\partial E}{\partial y}\right)$$

$$\frac{\partial E}{\partial x} = \text{Conv}\left(180^\circ \text{ rotated filter } F', \text{ loss gradient } \frac{\partial E}{\partial y}\right)$$

Summary

CNN include convolutional layers comprising learned filters that scan the outputs of the previous layer.

Summary

CNN include convolutional layers comprising learned filters that scan the outputs of the previous layer.

Pooling layers operate on groups of outputs from the convolutional layer to reduce the network size.

Summary

CNN include convolutional layers comprising learned filters that scan the outputs of the previous layer.

Pooling layers operate on groups of outputs from the convolutional layer to reduce the network size.

The parameters of the CNN can be learned through backpropagation.

Convolutional Neural Networks in Practice

Advantages of CNNs:

- used for deep learning with few parameters
- less parameters to learn as compared to fully connected layer
- tool for image processing

Convolutional Neural Networks in Practice

Advantages of CNNs:

- used for deep learning with few parameters
- less parameters to learn as compared to fully connected layer
- tool for image processing

Advantages of CNNs:

- Comparatively complex to design and maintain
- Comparatively slow [depends on the number of hidden layers]

Convolutional Neural Networks in Practice

Advantages of CNNs:

- used for deep learning with few parameters
- less parameters to learn as compared to fully connected layer
- tool for image processing

Advantages of CNNs:

- Comparatively complex to design and maintain
- Comparatively slow [depends on the number of hidden layers]

General Applications of CNNs:

- Image processing
- Computer Vision
- Speech Recognition

Astronomy with CNN

1. extracting prominent features from astronomical images, e.g.
   - craters
   - gravitational lenses

Astronomy with CNN

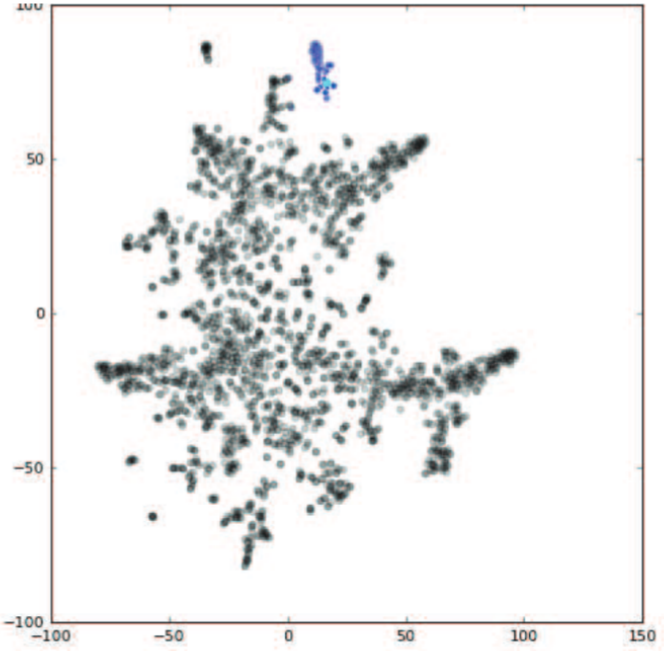
1. extracting prominent features from astronomical images, e.g.
   - craters
   - gravitational lenses
2. classifying objects on astronomical images

Astronomy with CNN

1. extracting prominent features from astronomical images, e.g.
   - craters
   - gravitational lenses
2. classifying objects on astronomical images
3. classifying time-series data based on a plot (light curve)

Astronomy with CNN

example:



left: the Hubble Heritage image of the Carina Nebula
right: the locations of each of the 2240 sub images in the clustering space

Convolutional Neural Networks in Astronomy, and Applications for Diffuse Structure Discovery, Peek, J. E. G.; Jones, Craig K.; Hargis, Jonathan, Astronomical Data Analysis Software and Systems XXVII, April 2020

Astronomy with CNN

Rotation-invariant convolutional neural networks for galaxy morphology prediction, S. Dieleman, K. W. Willett, J. Dambre (2015)

Measuring the morphological parameters of galaxies is a key requirement for studying their formation and evolution. The deep neural network model for galaxy morphology classification exploits translational and rotational symmetry.

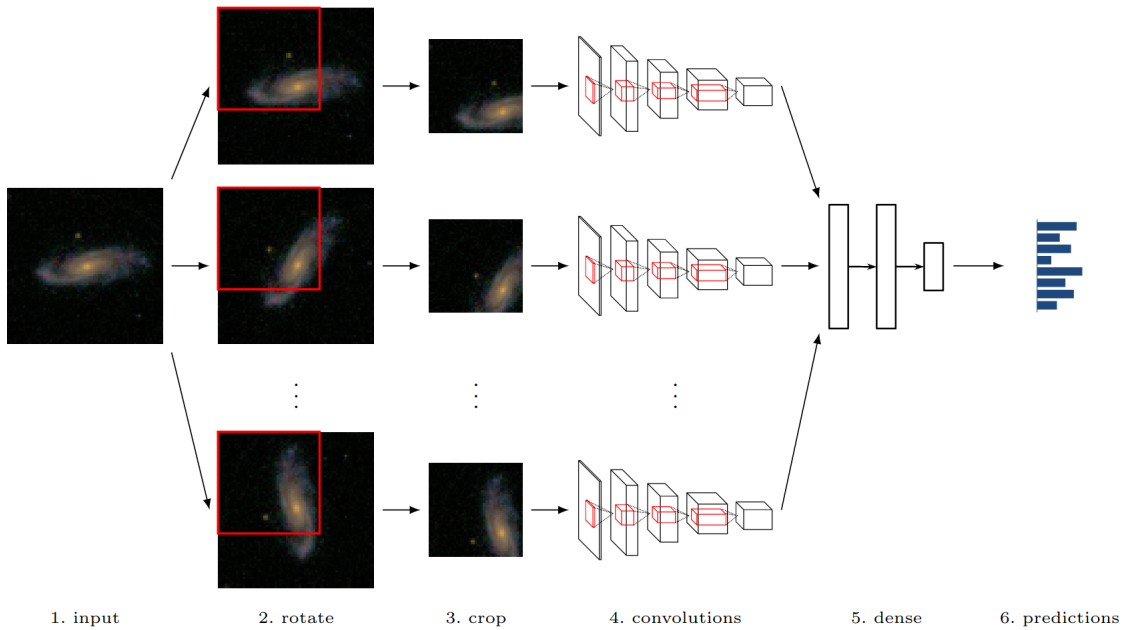
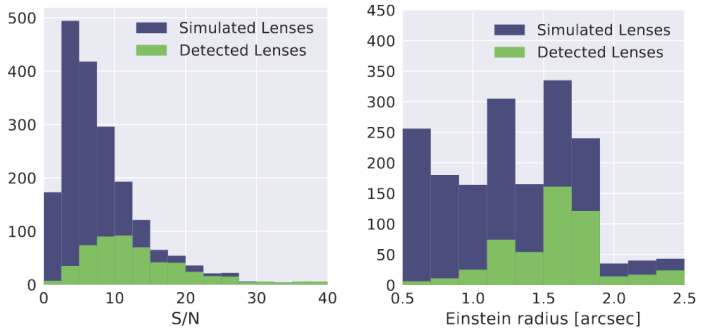


Figure 4: Schematic overview of the CNN architecture. The input (1) is rotated/ flipped for different viewpoints (2), which are cropped to reduce redundancy (3). Processing is done by convolutional and pooling layers (4), their output is concatenated and processed by dense layers (5) to obtain predictions (6).

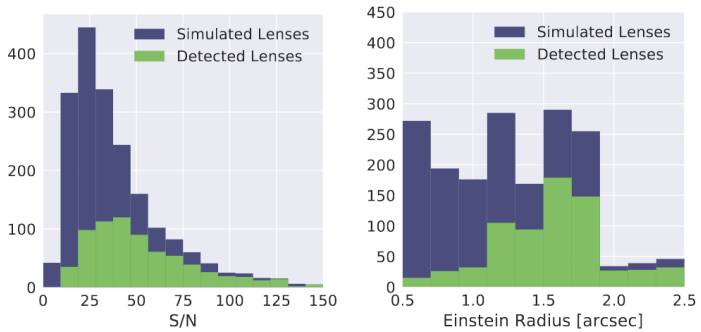
Astronomy with CNN

CMU DeepLens: Deep Learning For Automatic Image-based Galaxy-Galaxy Strong Lens Finding, F. Lanusse, Qu. Ma, N, Li, et al. (2017)

CMU DeepLens is a fully automated galaxy-galaxy lens finding method based on Deep Learning, trained and validated on 20,000 LSST-like mock observations including a range of lensed systems of various sizes and signal-to-noise ratios.



(a) Single best epoch images



(b) Stack images

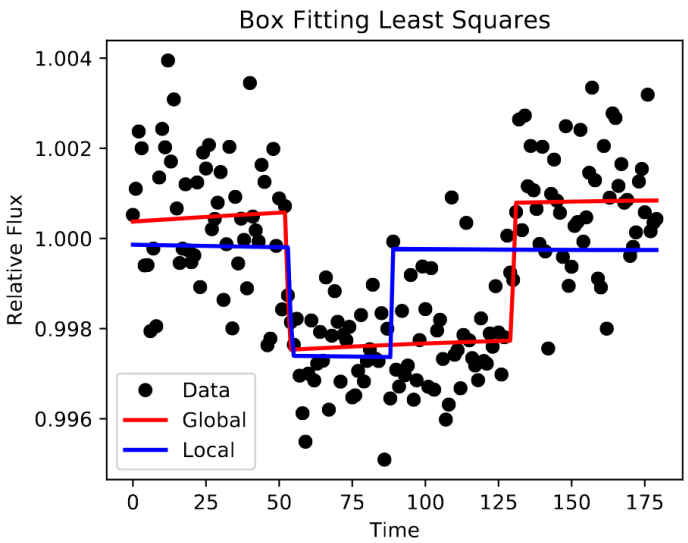
Figure 7. S/N and Einstein radius distribution of the simulated and recovered lens populations, for our fiducial 1% FPR detection threshold.

Astronomy with CNN

Searching for exoplanets using artificial intelligence, K. A. Pearson, L. Palafox, C. A. Griffith (2018)

the ideal algorithm should be

- fast
- robust to noise
- capable of learning and abstracting highly nonlinear systems



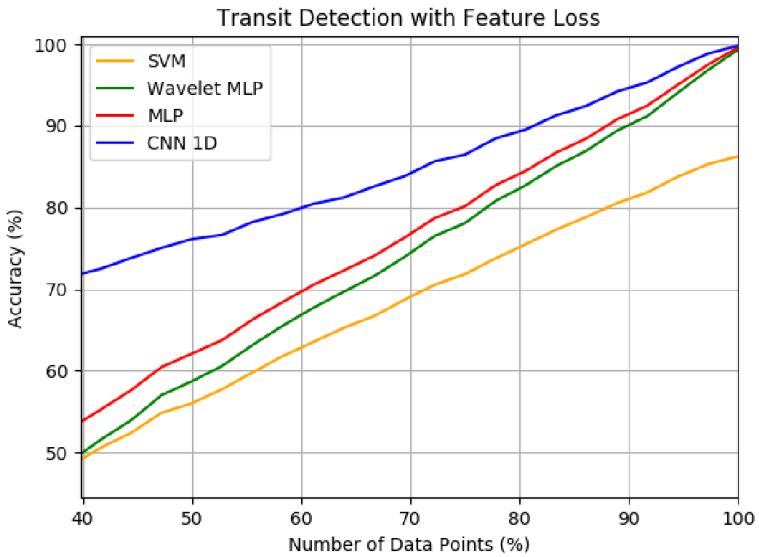


Figure 2. Past transit detection algorithms are accomplished by correlating the data with a simple box model through a least-squares optimization.

Astronomy with CNN

Finding Strong Gravitational Lenses in the Kilo Degree Survey with Convolutional Neural Networks, C. E. Petrillo, C. Tortora, S. Chatterjee et al. (2017)

A morphological classification method based on a Convolutional Neural Network (CNN) for recognizing strong gravitational lenses is applied to the 255 square degrees of the Kilo Degree Survey (KiDS), one of the current-generation optical wide surveys.

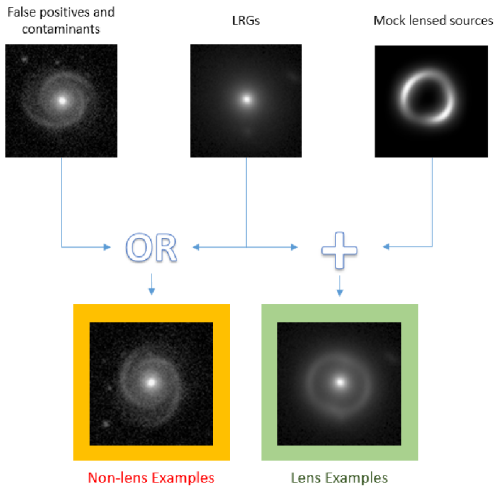
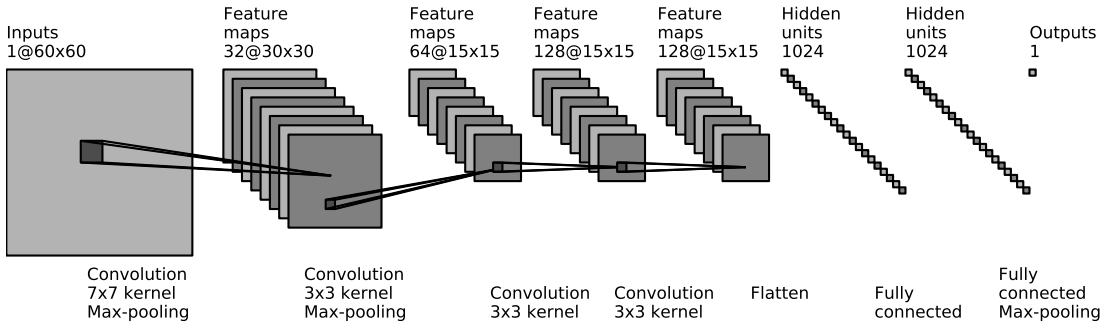


Figure 3. A schematic of the training-set creation.



Astronomy with CNN

Quasar microlensing light curve analysis using deep machine learning, G. Vernardos & G. Tsagkatakis (2019)

Objective: Use CNN to model quasar microlensing light curves and extract the size and temperature profie of the accretion disc.
The effects of microlensing are simulated by magnification maps: pixellated representations of the source plane caustics due to the microlenses.

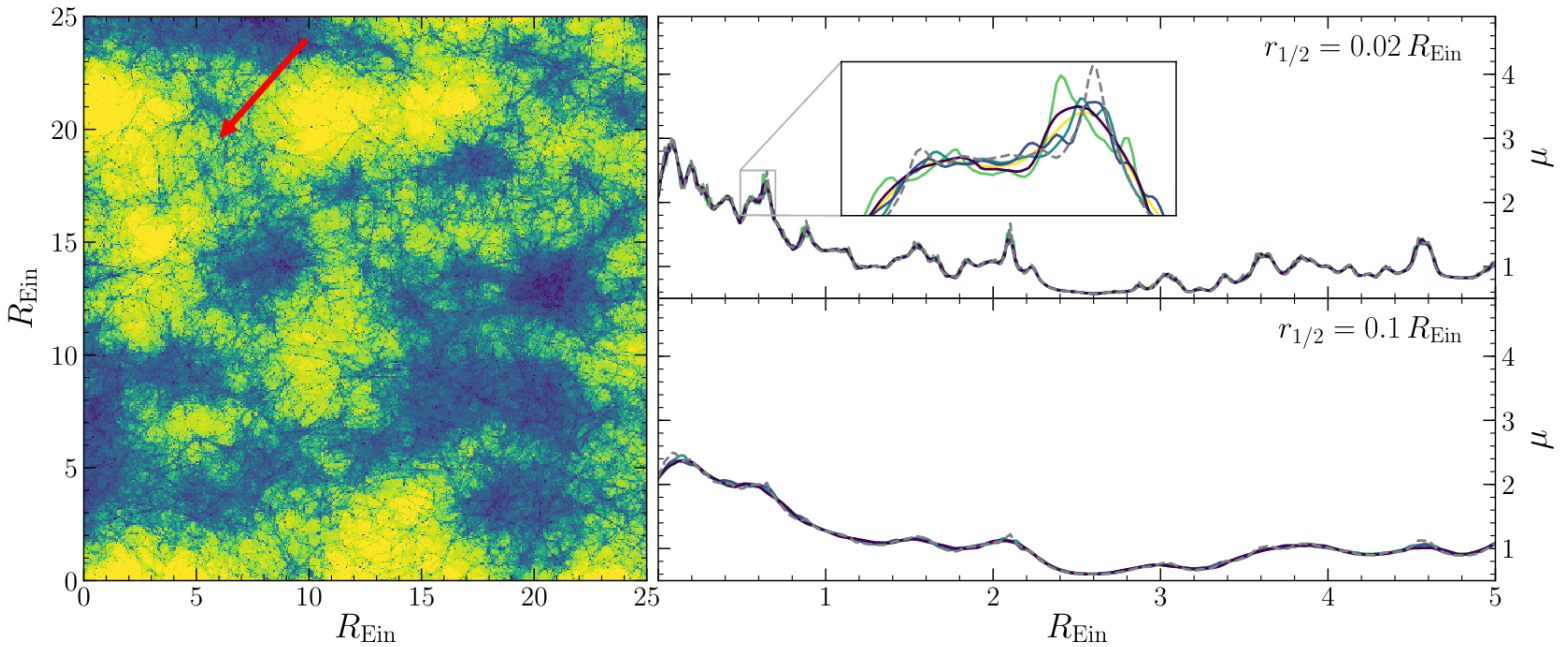


Figure 2. Mock light curve from magnification map. The light curve trajectory on the map is depicted as a red arrow.

Summary and Outlook

CNNs enable us to deal with images more naturally.

Summary and Outlook

CNNs enable us to deal with images more naturally.

Tomorrow we will see how to **visualize** a CNN's internal data, and how to make **training** more efficient.