The World of CNN

Debanga Raj Neog, MFSDS&AI, IIT Guwahati

Slides courtesy:

- My course DA 526 (Image Processing with Machine Learning)
- Dr. Konda Reddy Mopuri, Assistant Prof., MFSDS&AI, IIT Guwahati
- stanford cs231n

Digital Image Formation

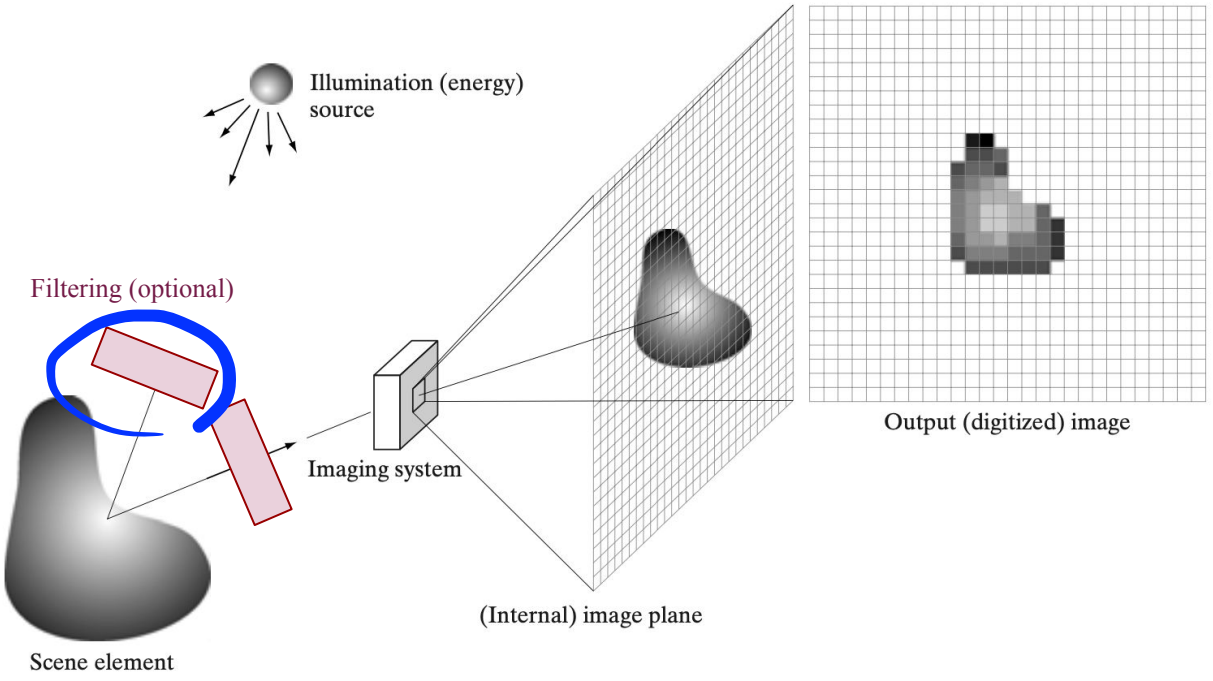


Figure adapted from Digital Image Processing by Rafael C. Gonzalez, Richard E. Woods

Digital Image Formation

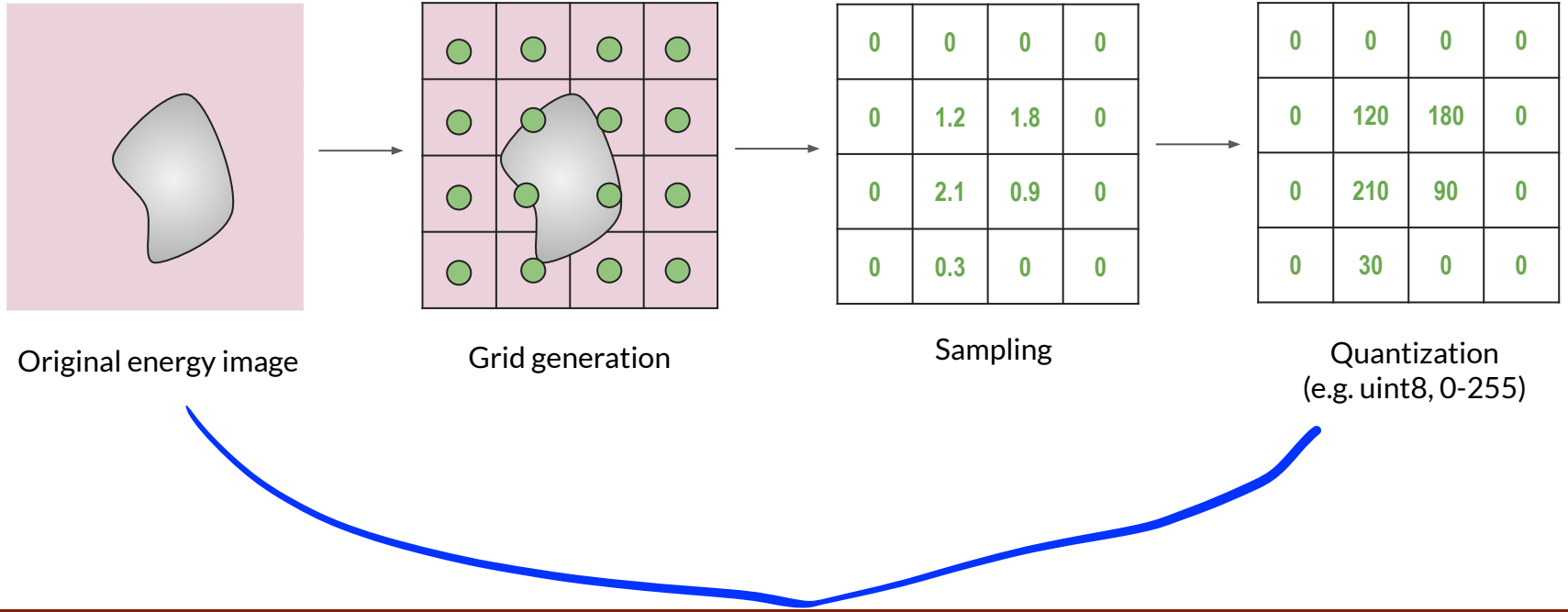
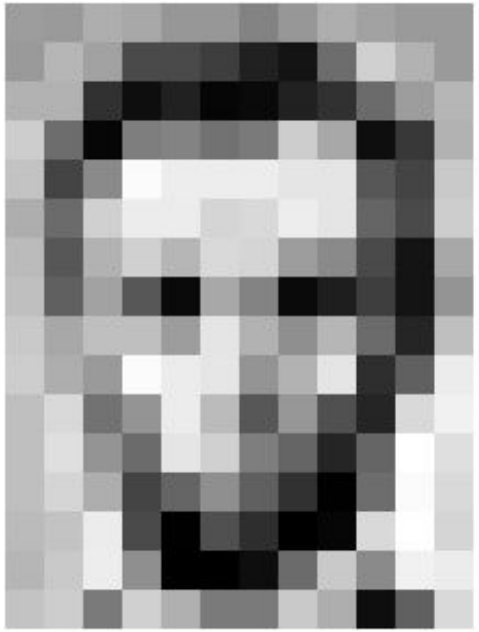


Image ~~Histogram~~



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

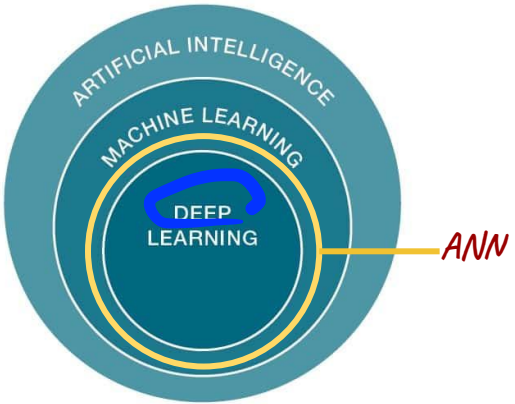
Grayscale Pixel Values

This is what computers see!

Artificial Neural Network

Artificial neural networks (ANNs) are a type of machine learning algorithm inspired by the structure and function of the human brain.

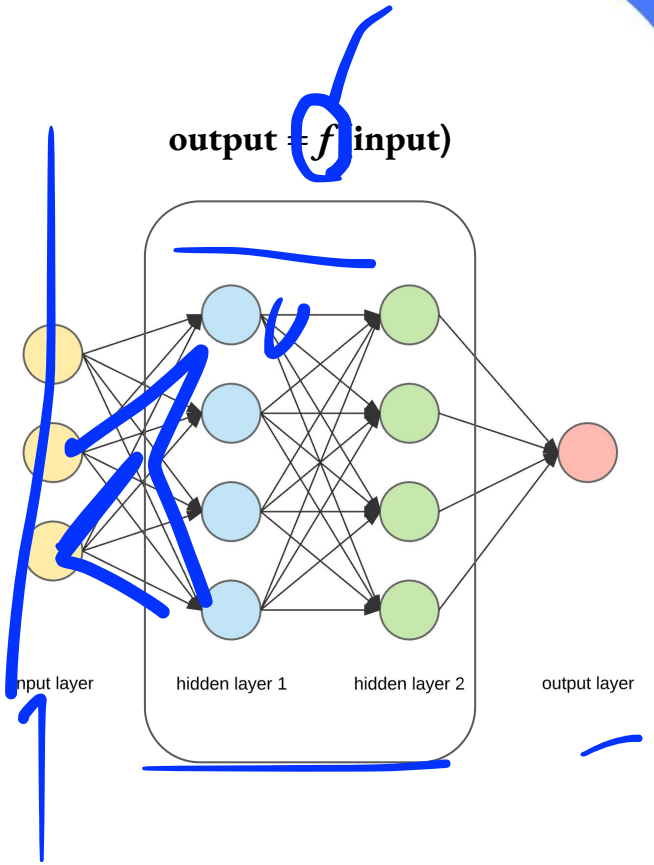
- They are used for a wide range of tasks such as classification and regression.

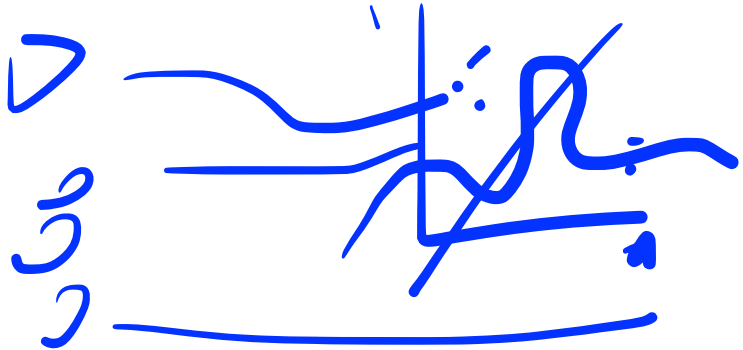


Artificial Neural Network

Artificial neural networks (ANNs) are a type of machine learning algorithm inspired by the structure and function of the human brain.

- In can be learned in a supervised fashion with examples of (input, output) pairs
- It is good at learning non linear mappings

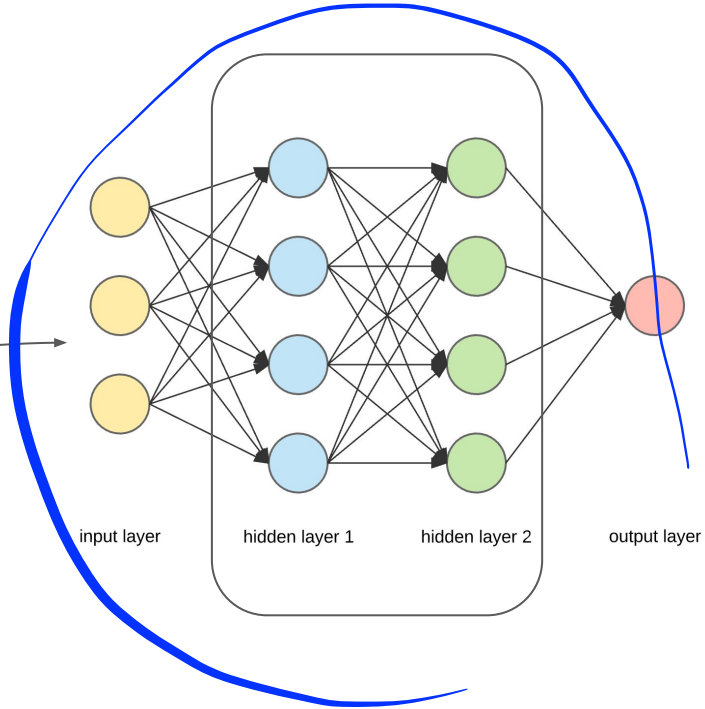




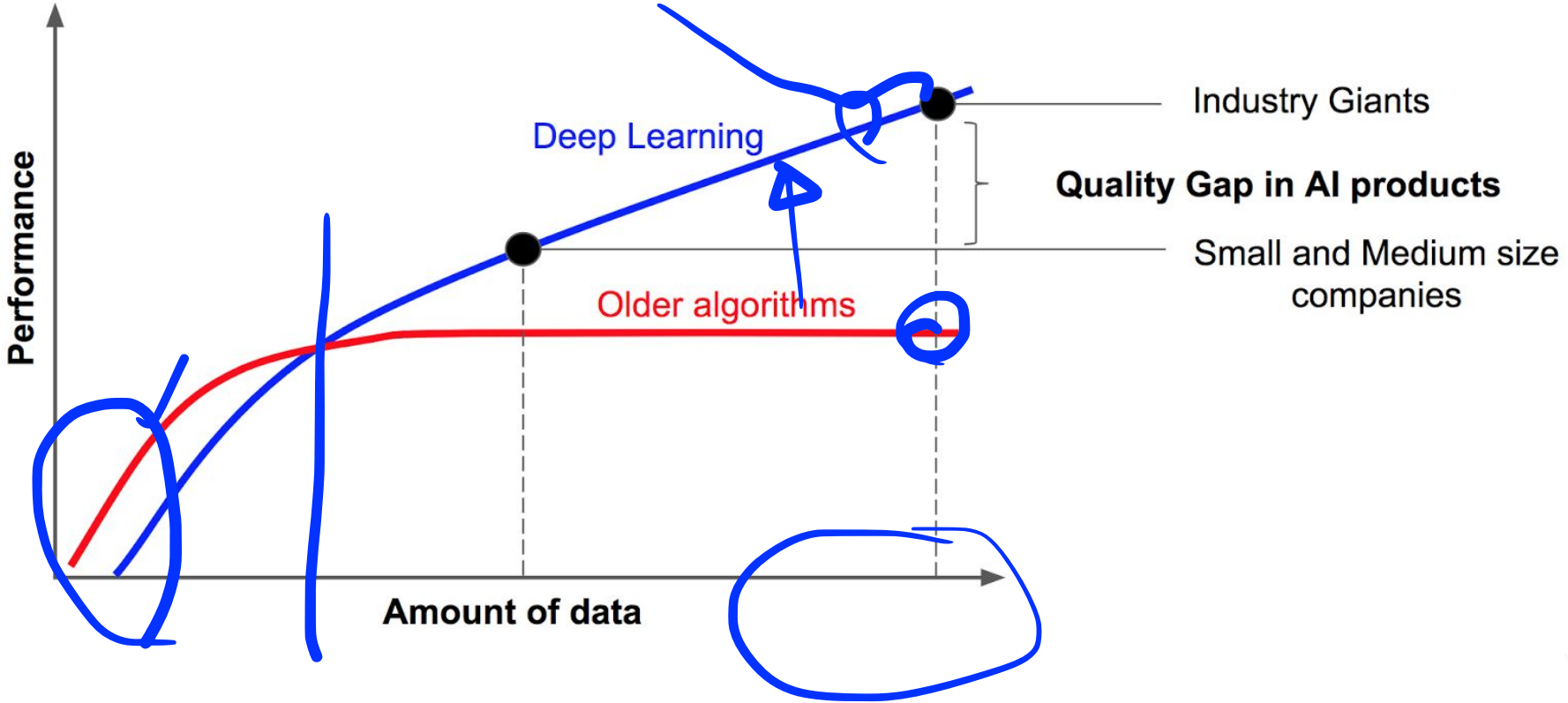
Artificial Neural Network

In its original form ANNs are not suitable for tasks related to image, such as image classification.

- Large number of parameters
- High computation

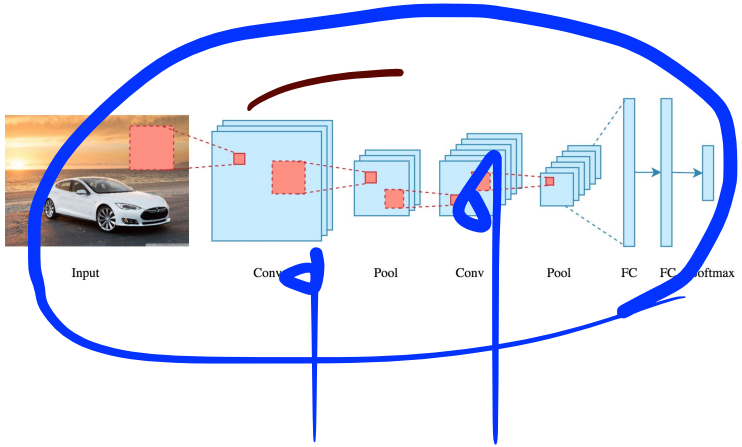


For a 200x200 image, number of input nodes will be 40,000!


Performance
Amount of data
Deep Learning
Older algorithms
Industry Giants
Quality Gap in AI products
Small and Medium size companies

Why Convolutional Neural Networks?

- ConvNet architectures make the **explicit assumption that the inputs are images** and encodes certain properties into the architecture.
  - Efficient to implement (suitable for GPU-based parallel implementation)
  - Reduce the number of parameters (reducing computational complexity)
- The name comes from a very special layer it uses that is known as **Convolutional Layer.**

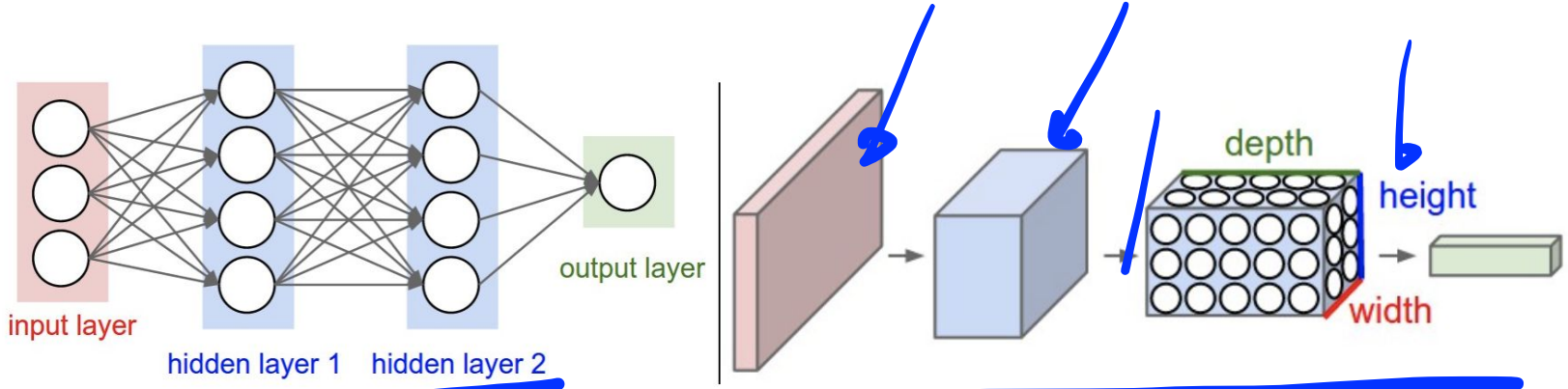


Features

- **Spatial arrangement:** Unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. This implementation fits well with image data.
- **Parameter sharing:** Consider an image of size, e.g. 200x200x3, would lead to neurons that have 200*200*3 = 120,000 weights. This full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting. ConvNet uses filters for feature extraction across the image, that share parameters.

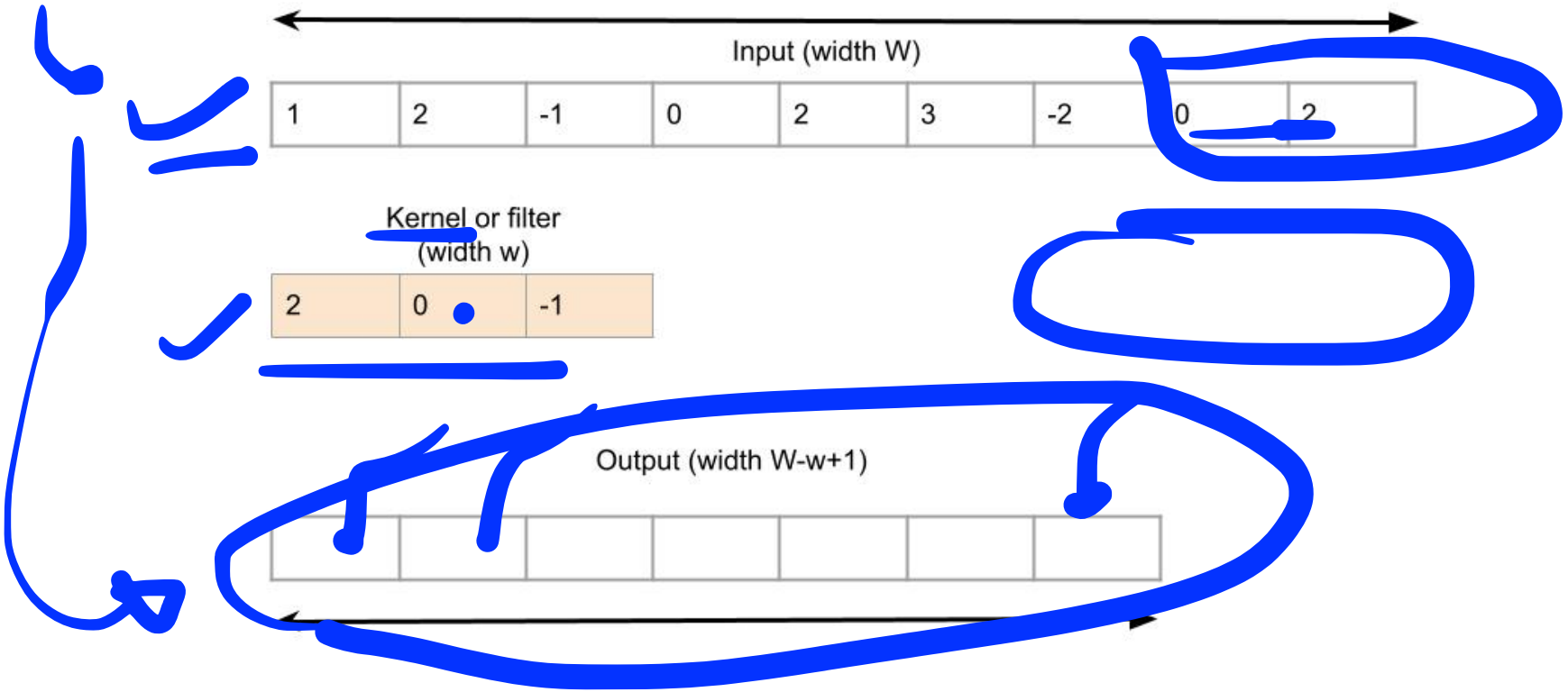
We will discuss more during the discussion of the convolutional layers.

Features

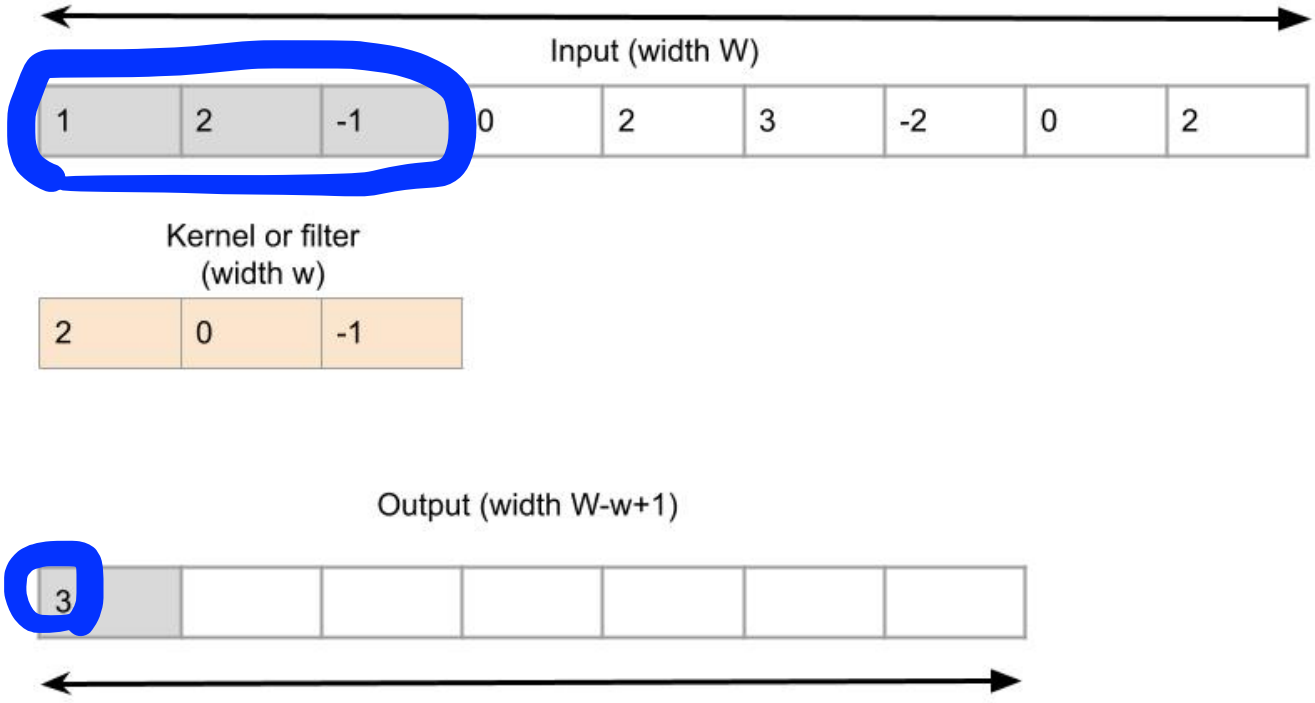


Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

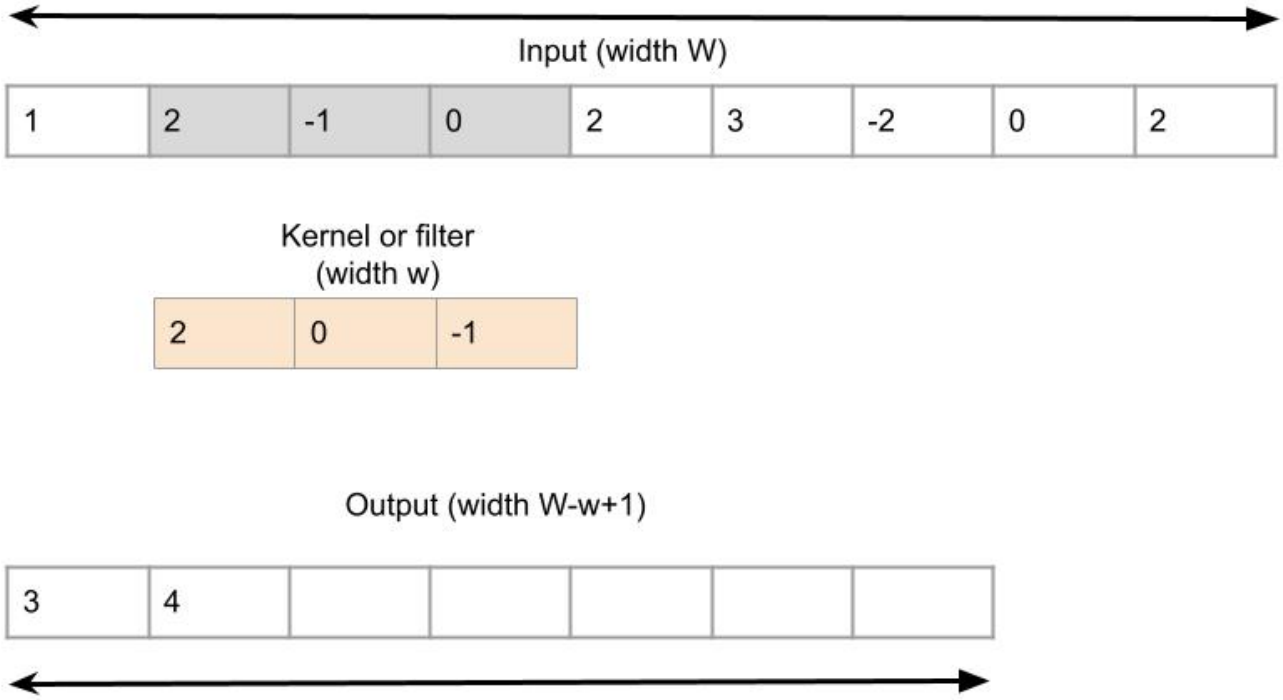
Convolution



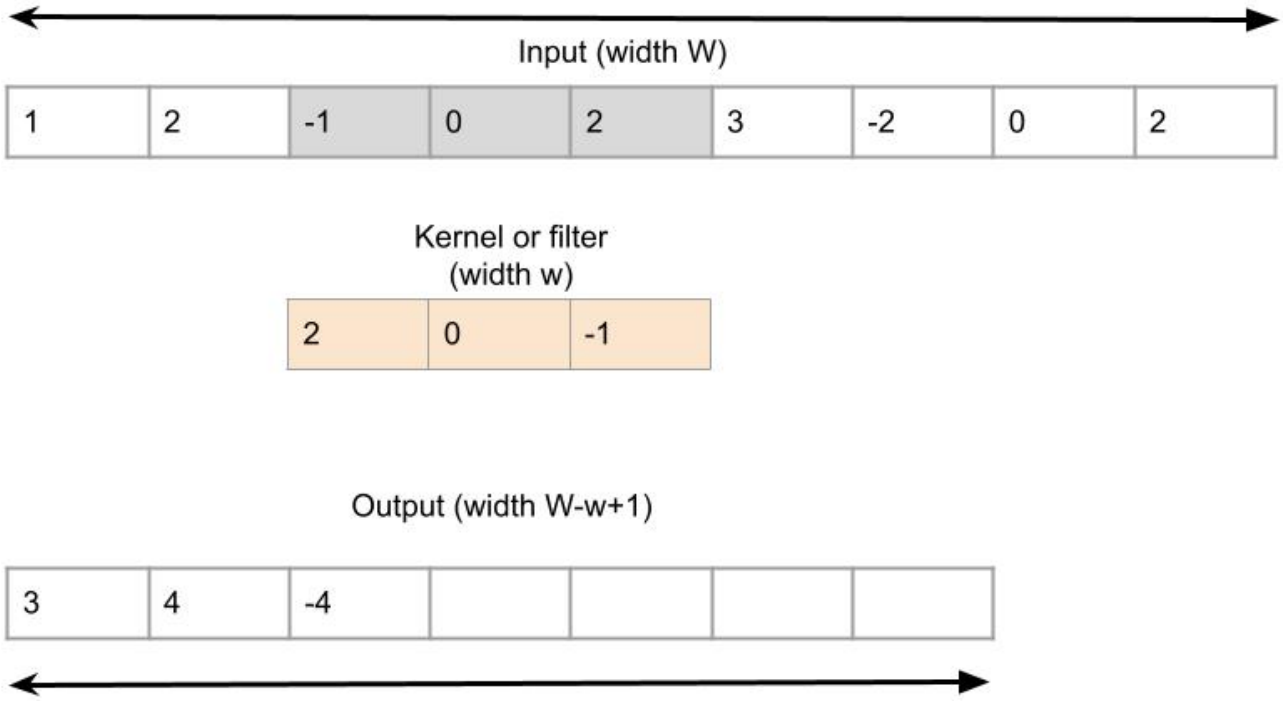
Convolution



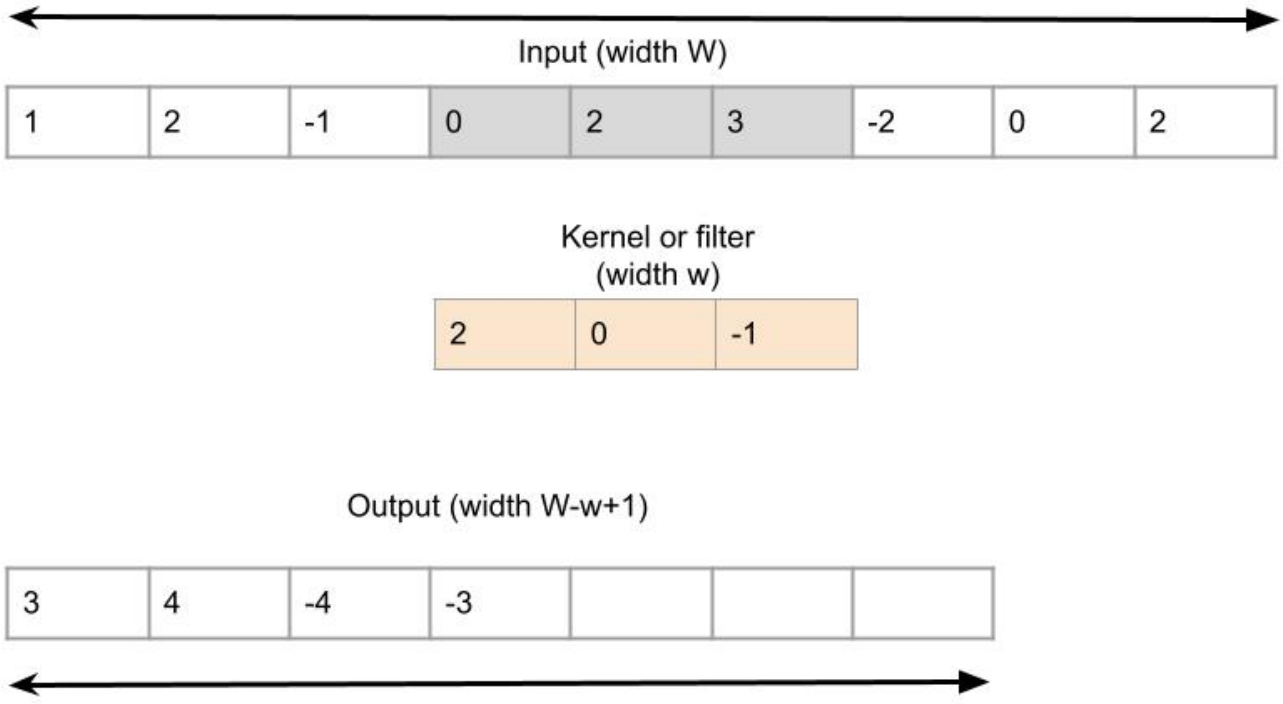
Convolution



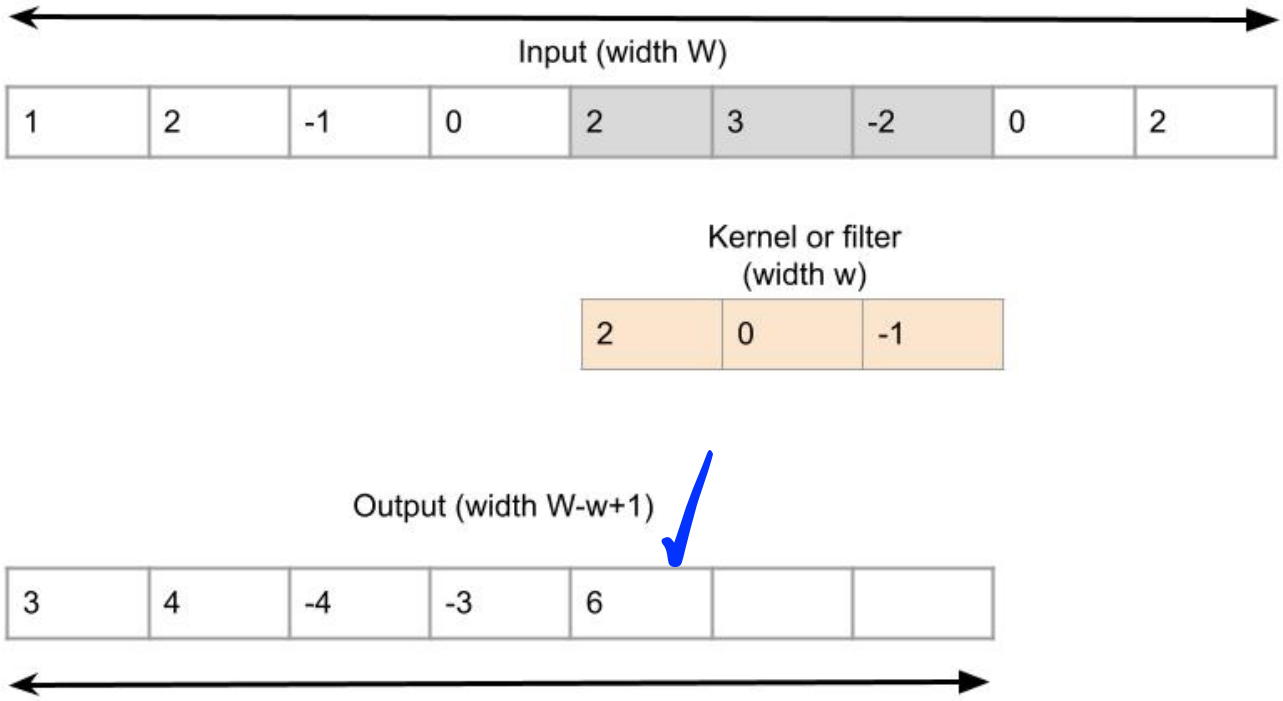
Convolution



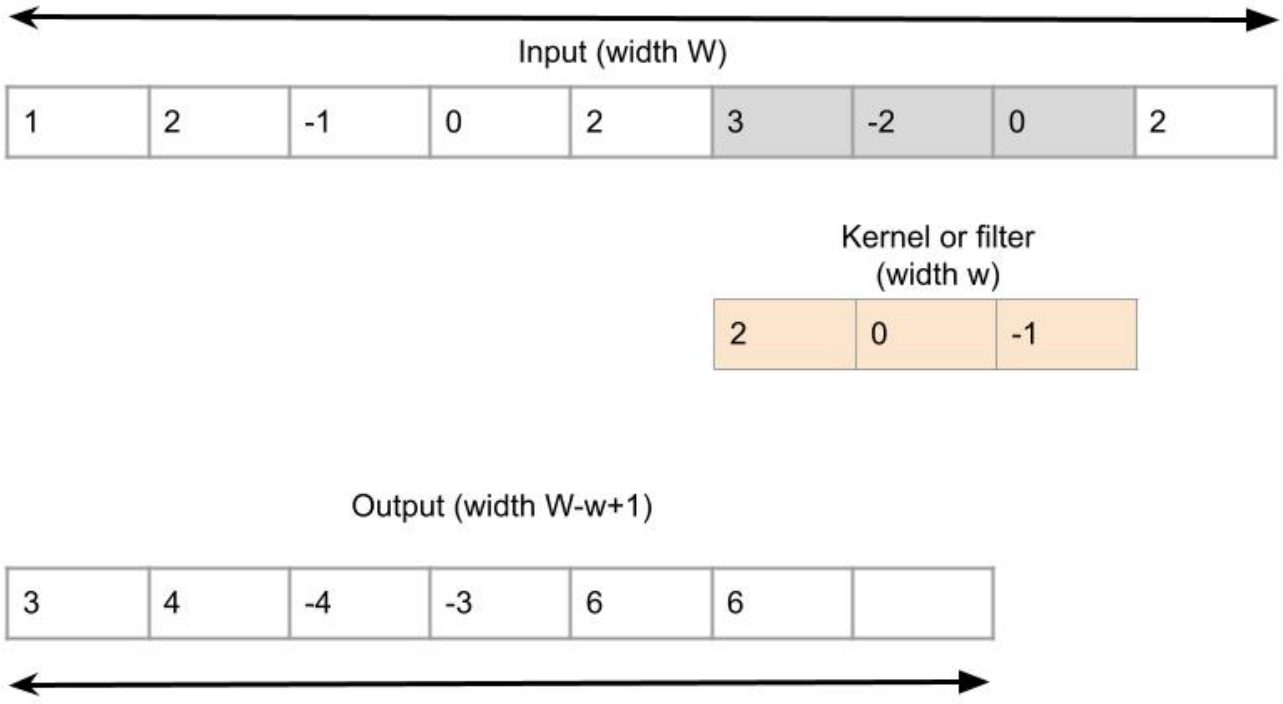
Convolution



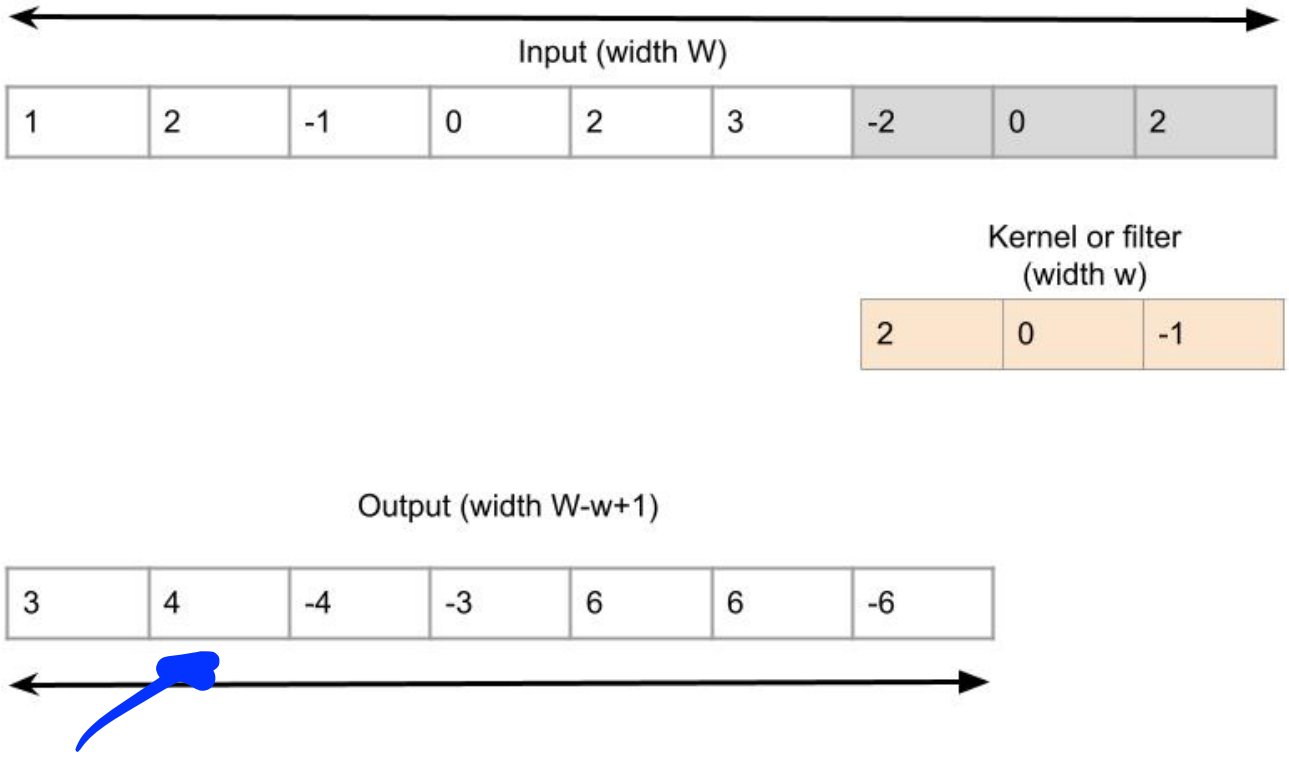
Convolution



Convolution



Convolution



Convolution

- Preserves the structure

Convolution

- Preserves the structure
  - if the i/p is a 2D tensor $\rightarrow$ o/p is also a 2D tensor

Convolution

- Preserves the structure
  - if the i/p is a 2D tensor $\rightarrow$ o/p is also a 2D tensor
  - There exist a relation between the locations of i/p and o/p values

Convolution

- Let $\mathbf{x} = (x_1, x_2, \ldots x_W)$ is the input, $\mathbf{k} = (k_1, k_2, \ldots k_w)$ is the kernel

Convolution

- Let $\mathbf{x} = (x_1, x_2, \ldots x_W)$ is the input, $\mathbf{k} = (k_1, k_2, \ldots k_w)$ is the kernel
- The result $(x \circledast k)$ of convolving $\mathbf{x}$ with $\mathbf{k}$ will be a 1D tensor of size $W - w + 1$

$$
\begin{aligned}
(x \circledast k)_i &= \sum_{j=1}^{w} x_{i-1+j} k_j \\
&= (x_i, \ldots x_{i+w-1}) \cdot \mathbf{k}
\end{aligned}
$$

Convolution

- Powerful feature extractor

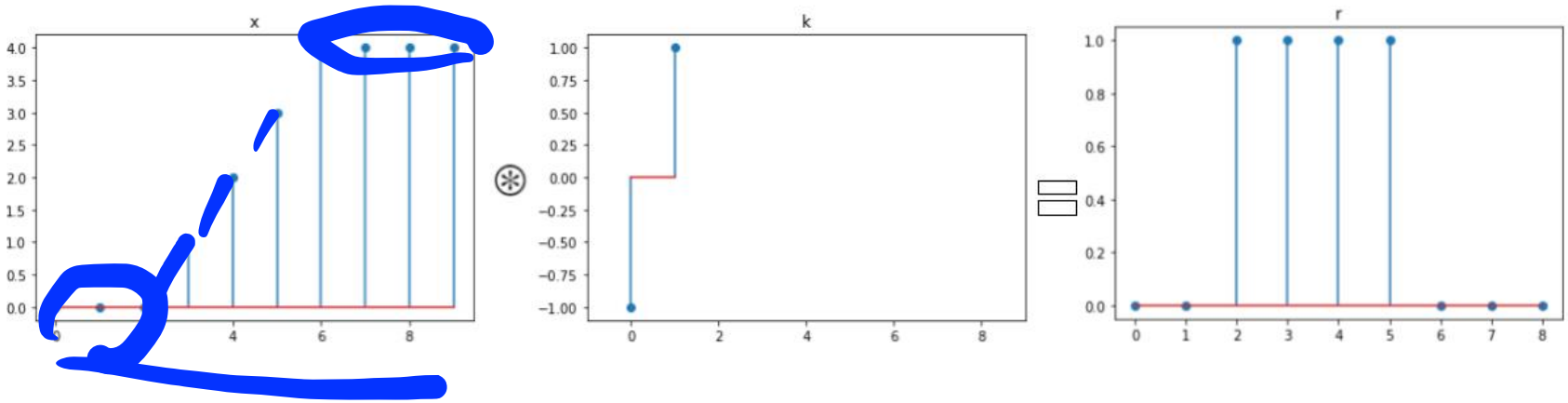
Convolution

- Powerful feature extractor
- For instance, it can perform differential operation and look for interesting patterns in the input

Convolution

- Powerful feature extractor
- For instance, it can perform differential operation and look for interesting patterns in the input
-

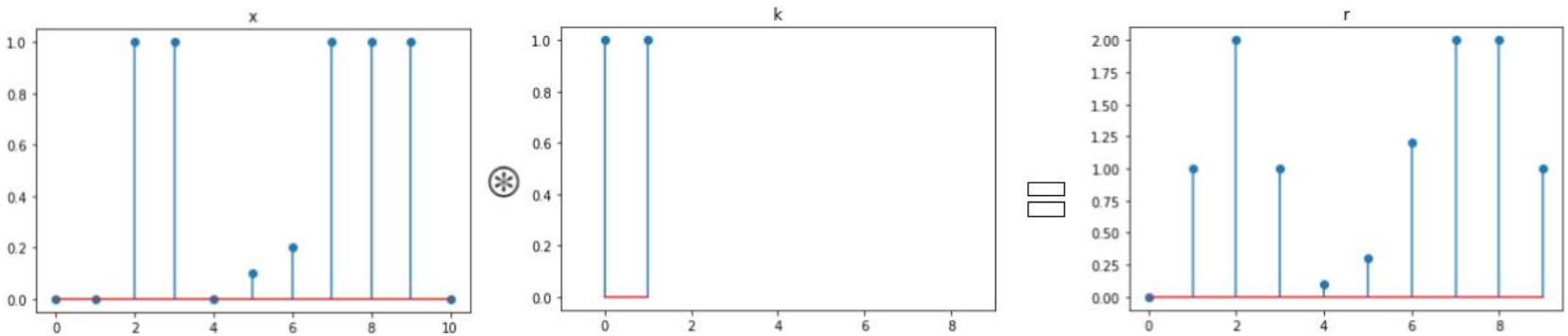
$$(0, 0, 0, 1, 2, 3, 4, 4, 4, 4) \circledast (-1, 1) = (0, 0, 1, 1, 1, 1, 0, 0, 0)$$



Convolution

- Powerful feature extractor
- For instance, it can perform differential operation and look for interesting patterns in the input
-

$$(0, 0, 1, 1, 0, 0.1, 0.2, 1, 1, 1, 0) \circledast (1, 1) = (0, 1, 2, 1, 0.1, 0.3, 1.2, 2, 2, 1)$$



Convolution

- Naturally generalizes to multiple dimensions

Image Filtering

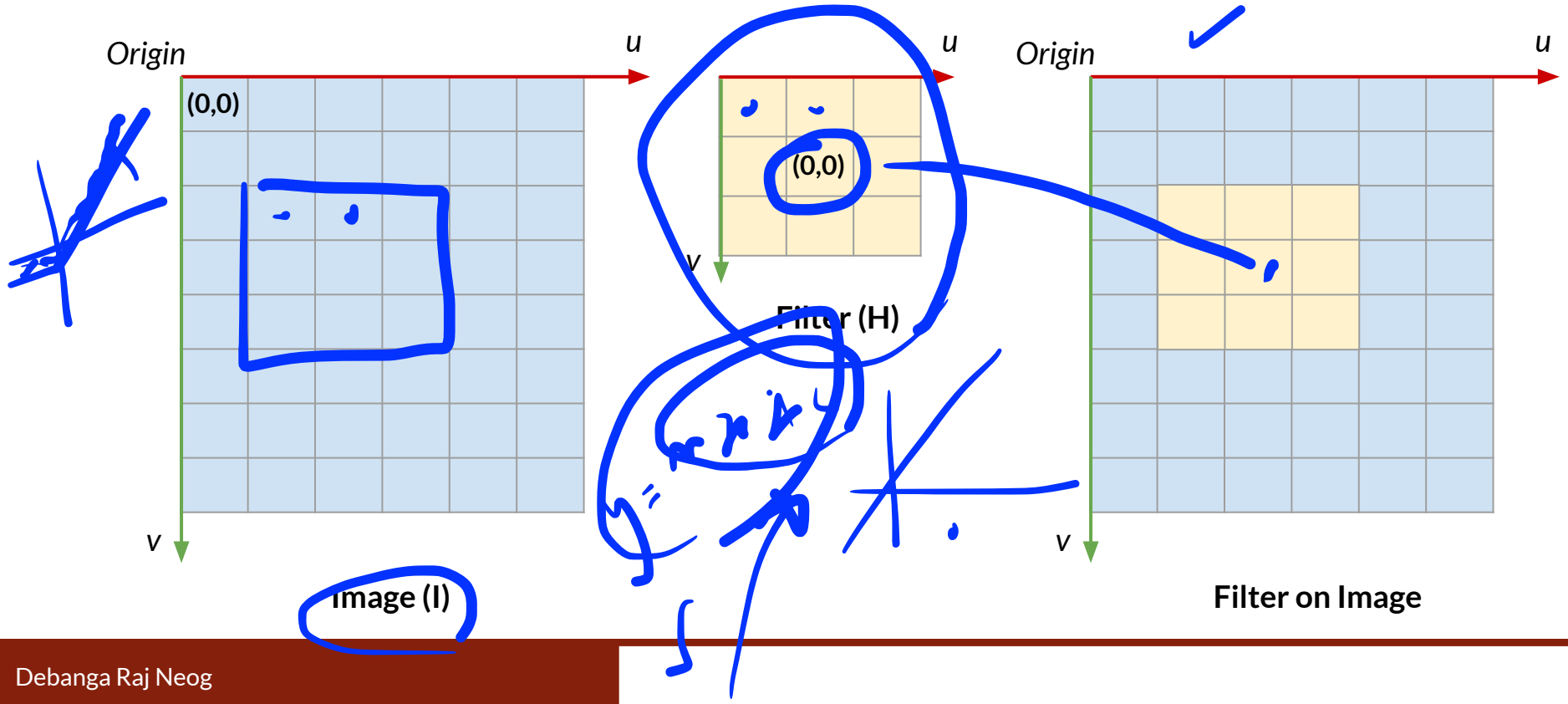
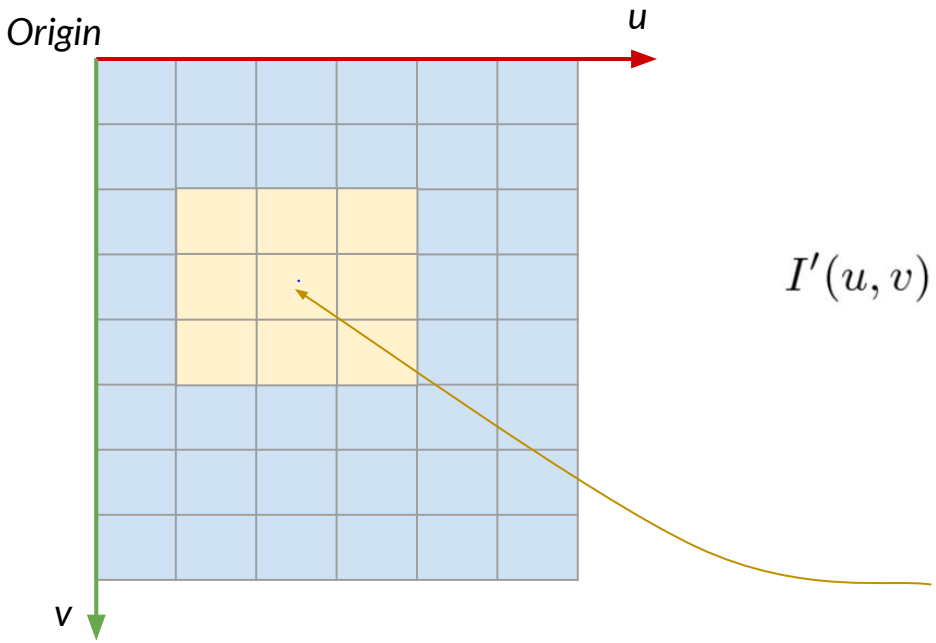


Image Filtering



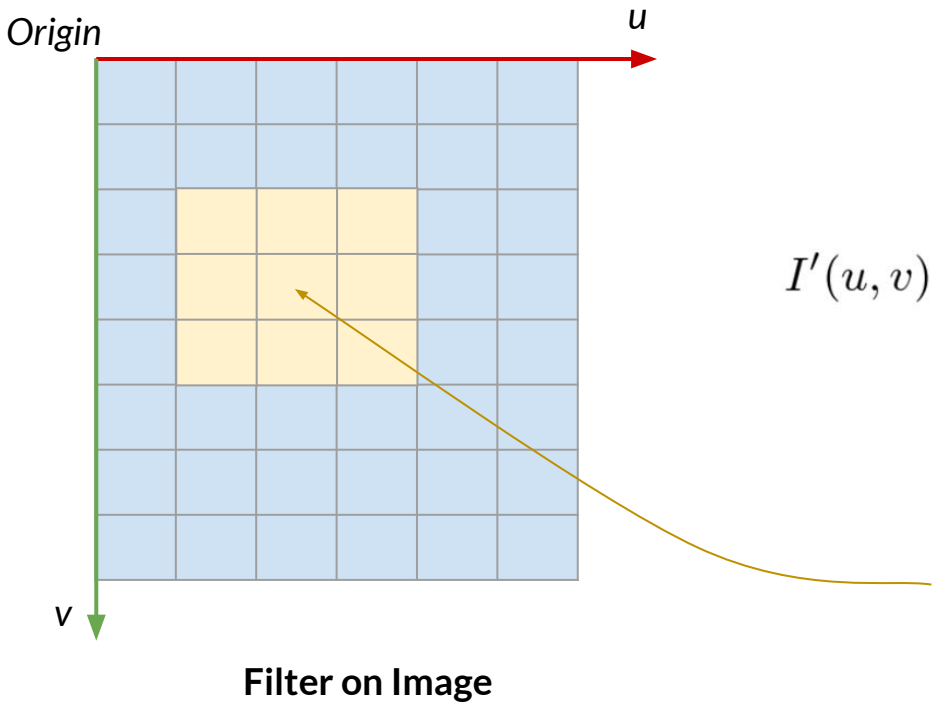
Filter on Image

Linear Convolution

$$I'(u,v) \leftarrow \sum_{i=-1}^{i=1} \sum_{j=-1}^{j=1} I(u+i, v+j) \cdot H(i,j)$$

1. Image coordinates: ?
2. Filter coordinates: ?

Image Filtering



Filter on Image

Linear Convolution

$$I'(u,v) \leftarrow \sum_{i=-1}^{i=1} \sum_{j=-1}^{j=1} I(u+i, v+j) \cdot H(i,j)$$

1. Image coordinates: (u,v)=(2,3)
2. Filter coordinates: (0,0)

Image Filtering

Example of **H**:

0	0	0
0	1	0
0	0	0

All pass / Identity

$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Averaging

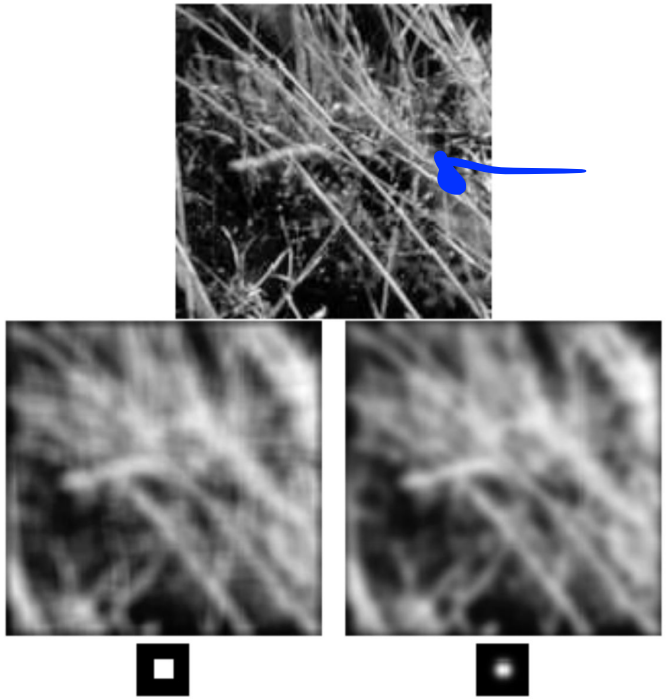
$$\frac{1}{41} \times \begin{bmatrix} 3 & 5 & 3 \\ 5 & 9 & 5 \\ 3 & 5 & 3 \end{bmatrix}$$

Weighted Average

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

Edge detection

Image Filtering



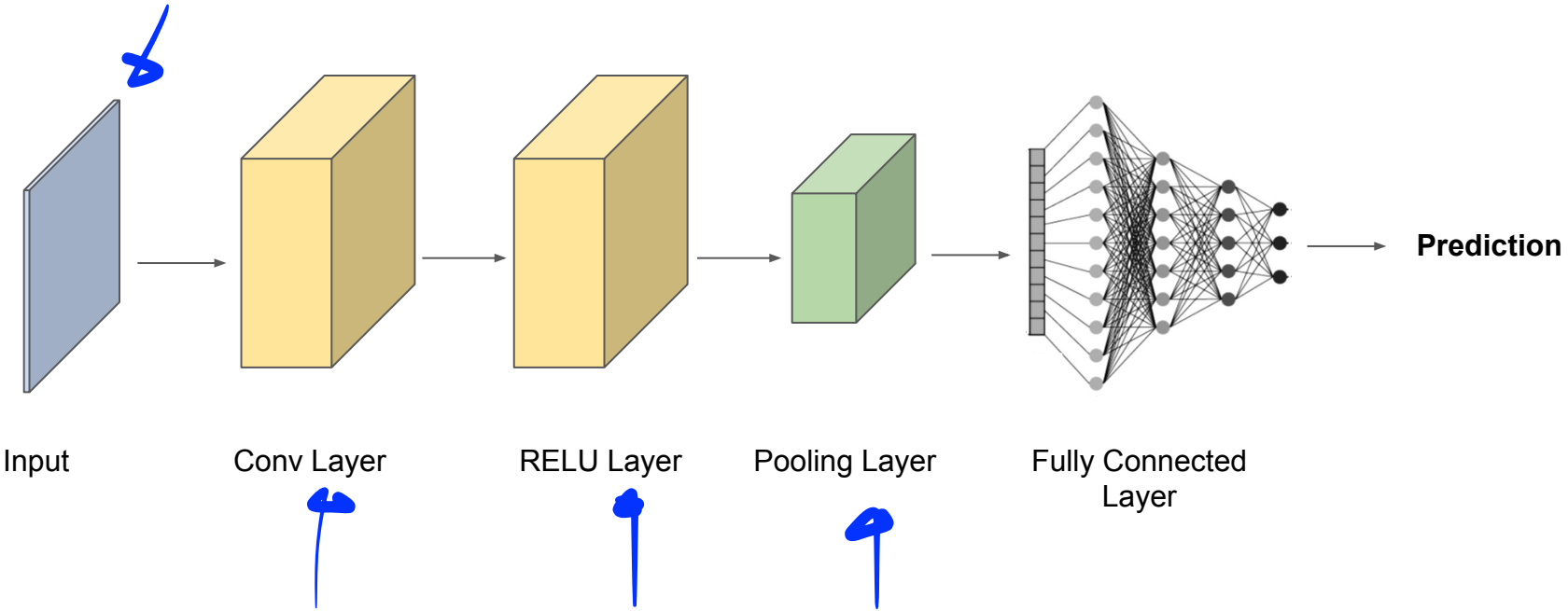
Average vs. Gaussian Smoothing

Layers

- A ConvNet is made up of Layers.
- Every Layer transforms **an input 3D volume** to **an output 3D volume** with some differentiable function that may or may not have parameters.
- Three main types of layers are used to build ConvNet architectures:
  - Convolutional Layer
  - Pooling Layer
  - Fully-Connected Layer (we have already seen)

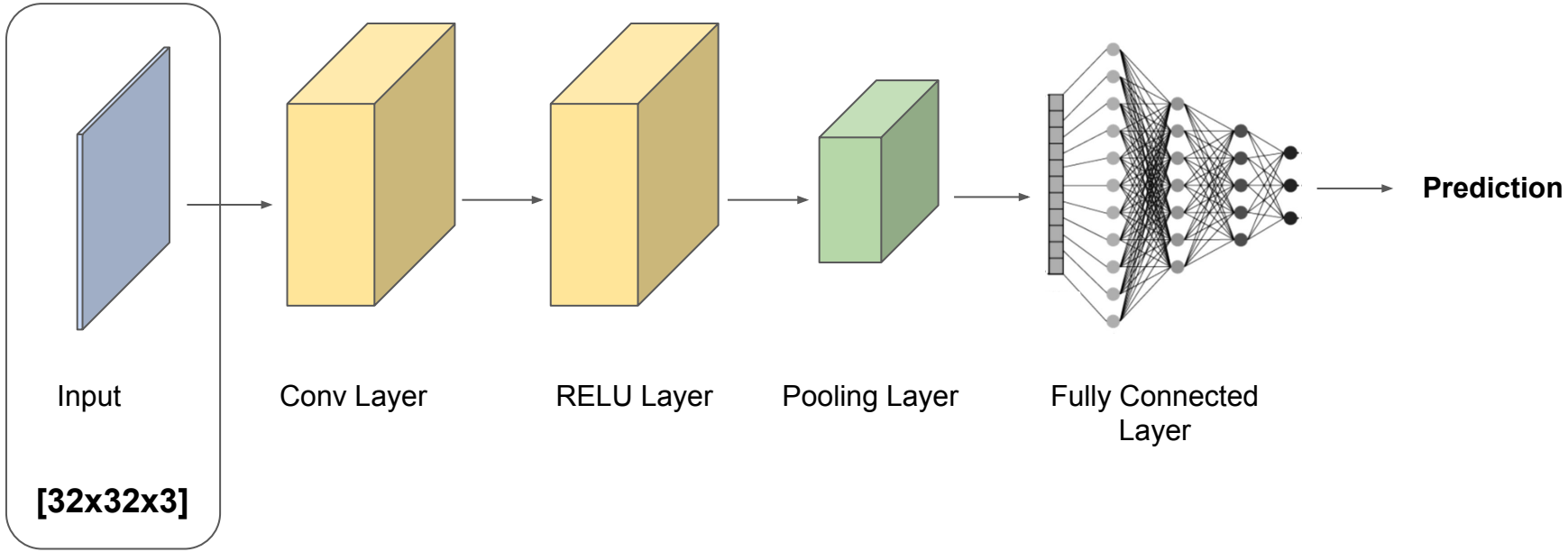
  There can be other layers: RELU layer, Dropout layer, BatchNorm layer and so on.

A Shallow Toy ConvNet



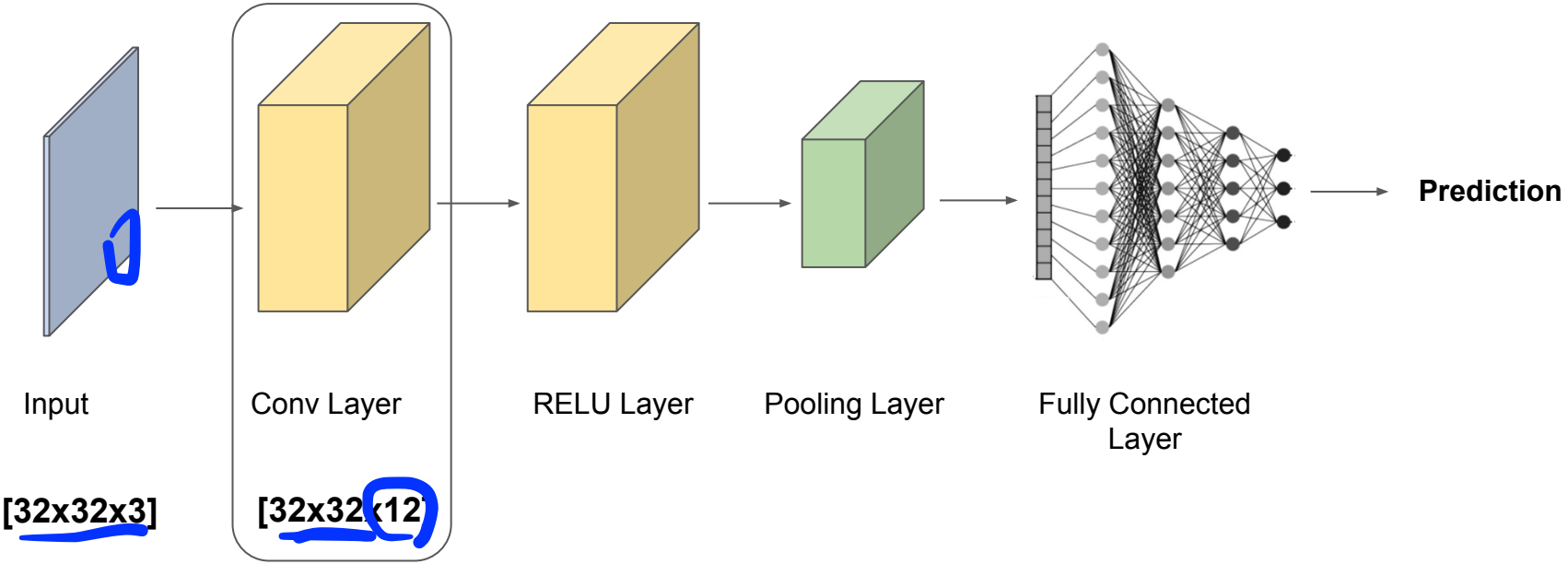
A Shallow Toy ConvNet

- INPUT: an image with width 32 pixels, height 32 pixels, and 3 channels (R,G,B)



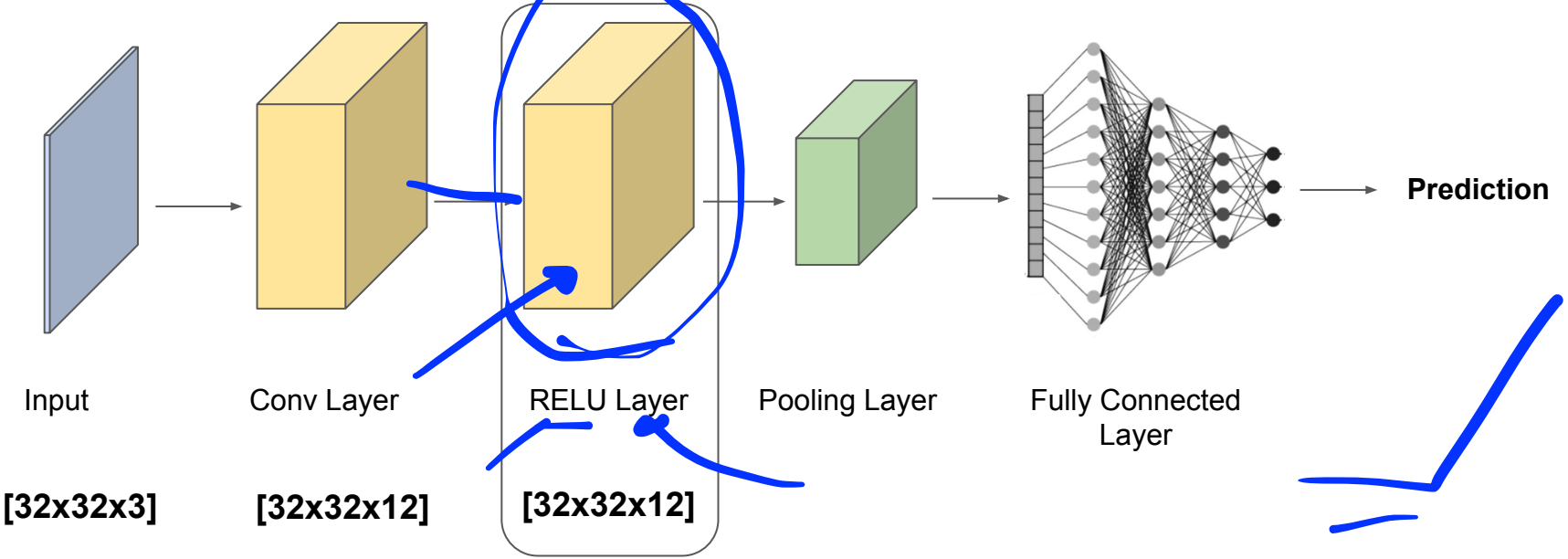
A Shallow Toy ConvNet

- CONV: 12 filters, output can be 32x32x12 feature maps.



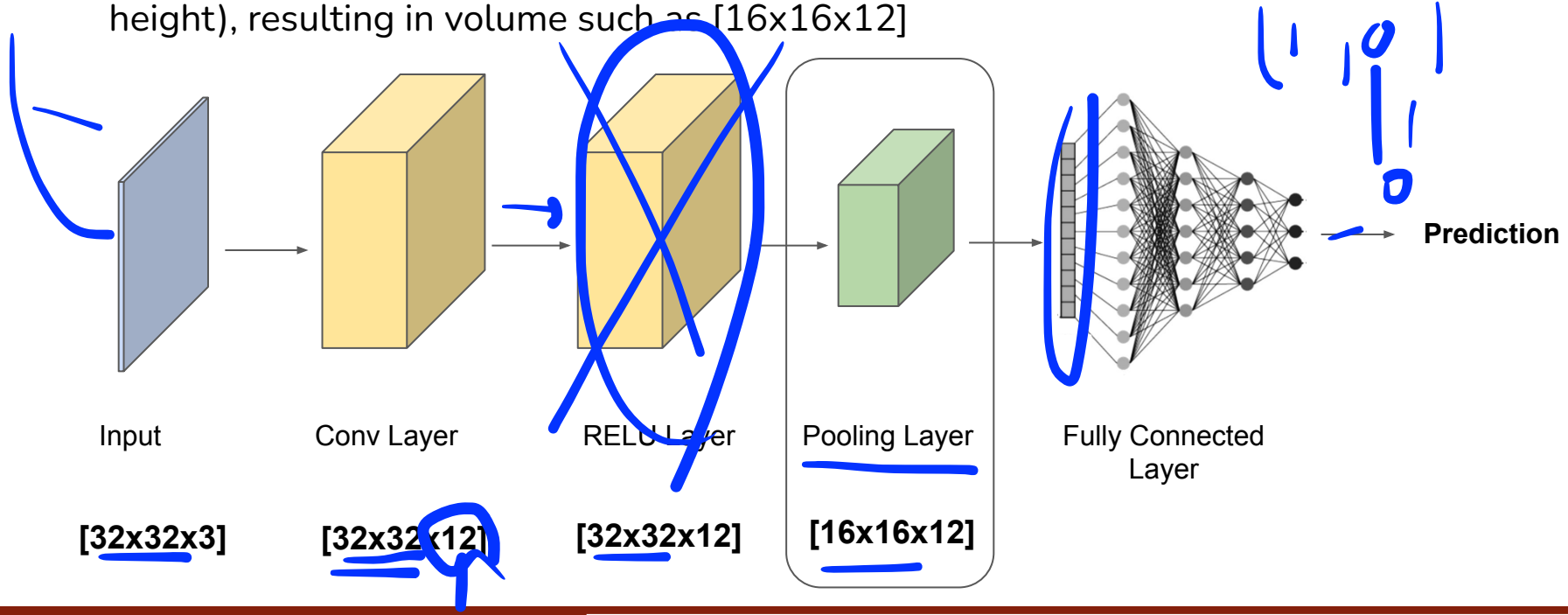
A Shallow Toy ConvNet

- RELU: **Element-wise activation** function, eg. max(0,x) and dimension remains same. It introduced the **non-linearity** in the transformations.



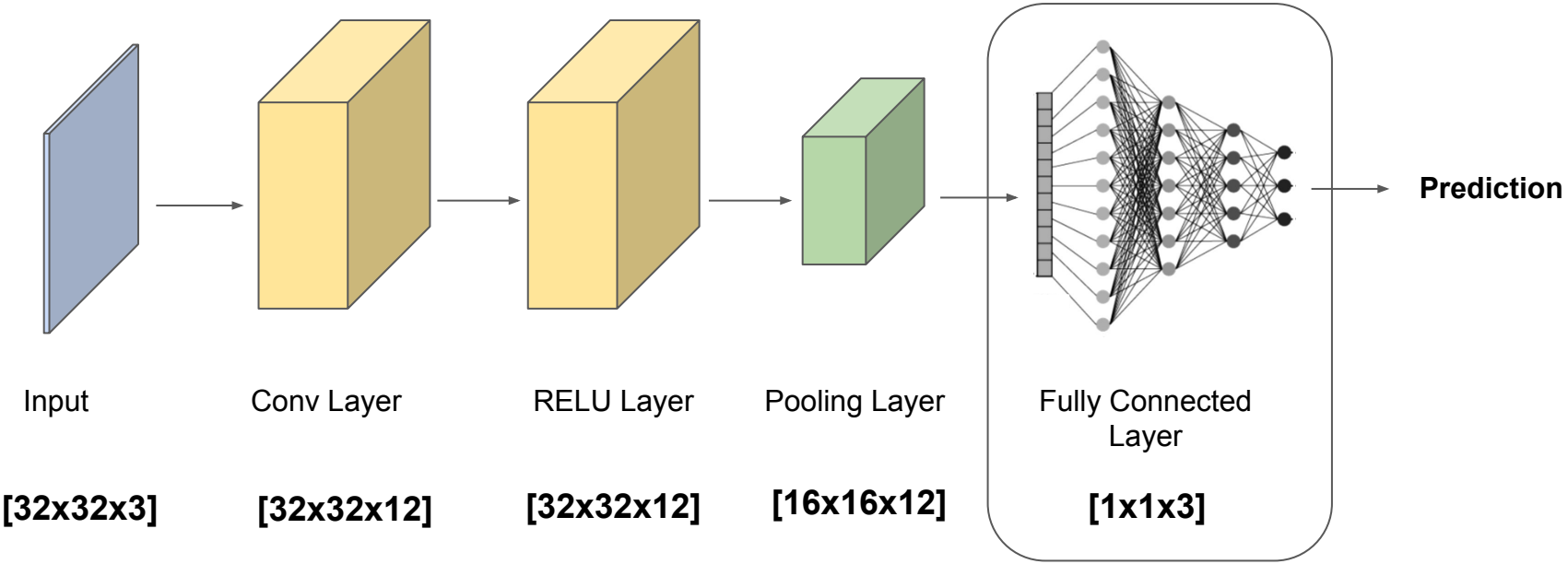
A Shallow Toy ConvNet

- POOL: perform a **downsampling operation** along the spatial dimensions (width, height), resulting in volume such as [16x16x12]



A Shallow Toy ConvNet

- FC: **Performs linear classification.** Each neuron will be connected to all neurons in the previous layer flattened. Output can be 1x1x3 in case of a 3 class classification.



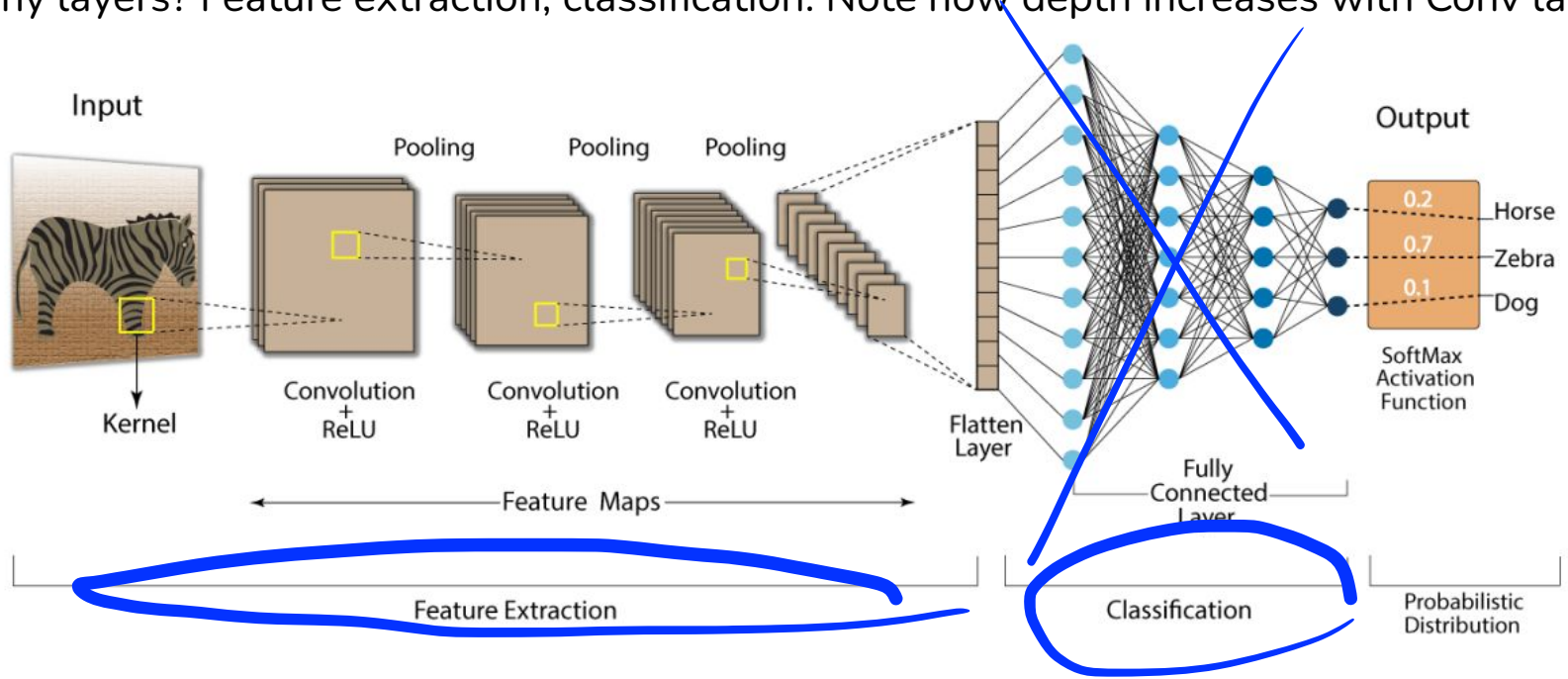
A Shallow Toy ConvNet

In summary:

- Transforms the image volume into an output volume (e.g. holding the class scores)
- There are a few distinct types of Layers (e.g. CONV/FC/RELU/POOL are by far the most popular)
- Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function
- Each Layer may or may not have parameters (e.g. CONV/FC do, RELU/POOL don't)
- Each Layer may or may not have additional hyperparameters (e.g. CONV/FC/POOL do, RELU doesn't)
- The parameters in the CONV/FC layers will be trained with gradient descent so that the class scores that the ConvNet computes are consistent with the labels in the training set for each image.

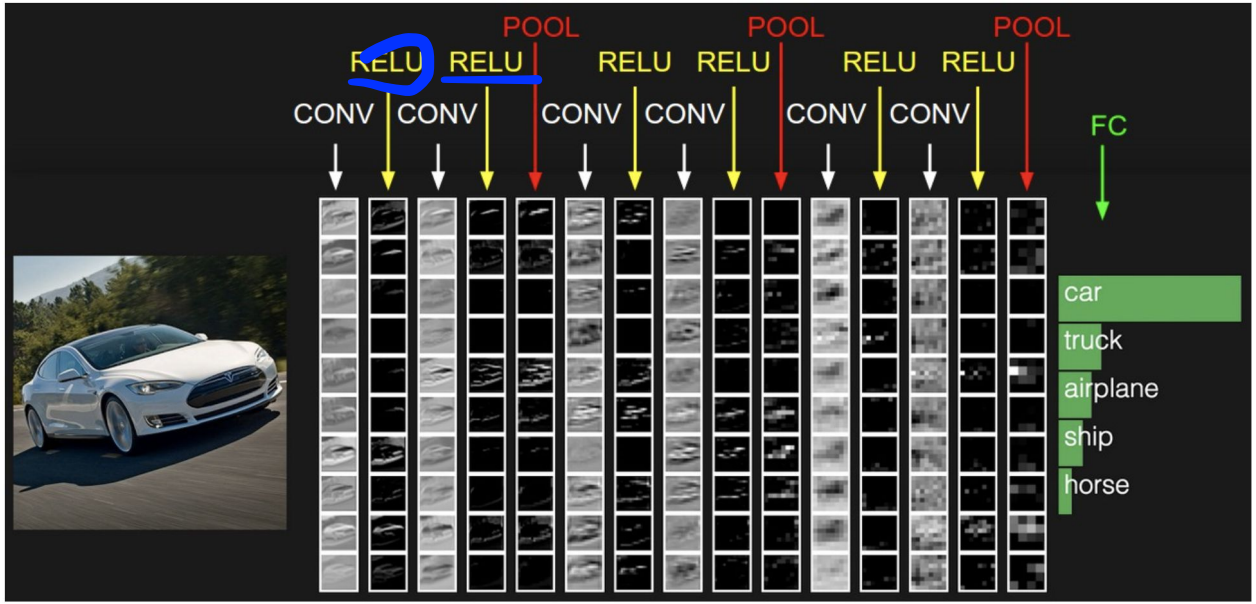
ConvNet

- Why layers? Feature extraction, classification. Note how depth increases with Conv layers.



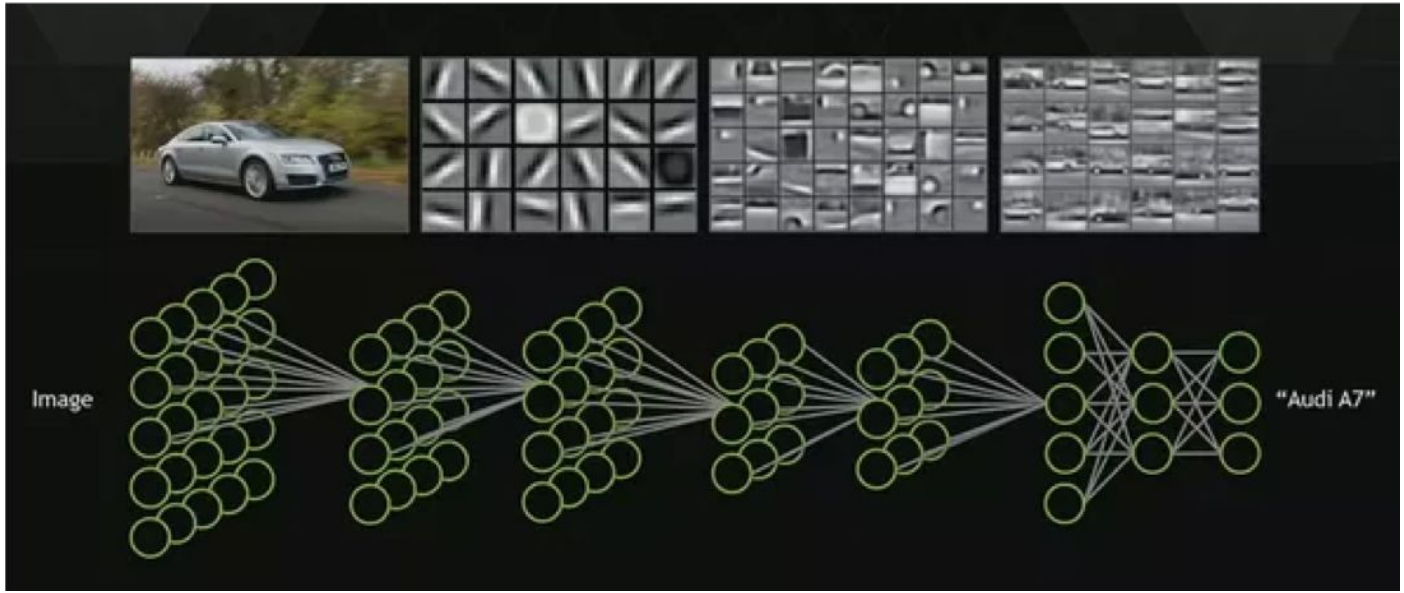
ConvNet

- A more practical convnet may look like this:



ConvNet

- Low level features to high level features.

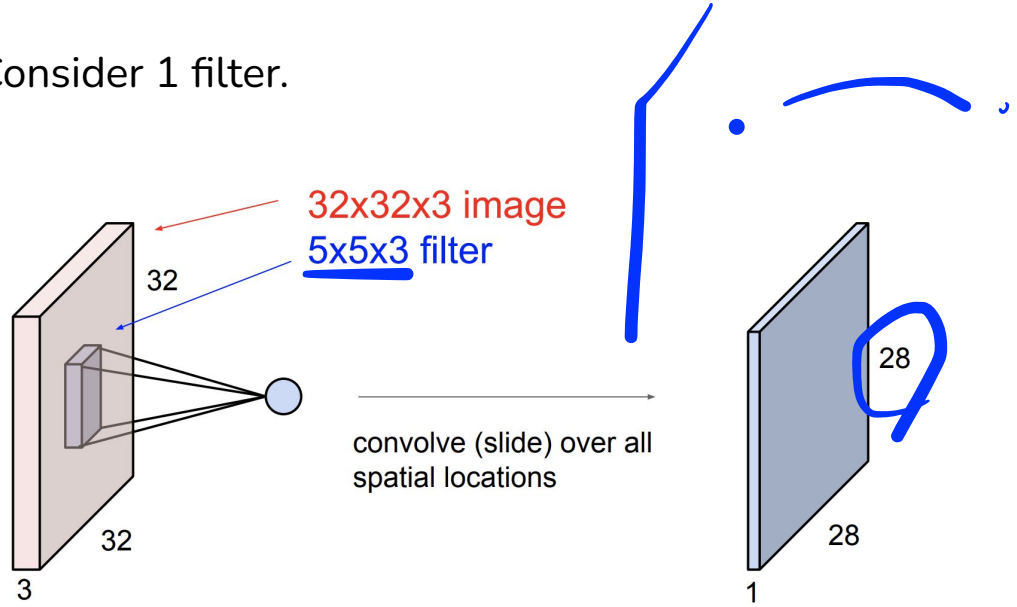


Conv Layer

- The Conv layer is the core building block of a Convolutional Network that does most of the computational heavy lifting
- The CONV layer parameters consist of a set of learnable filters
- Every filter is small spatially (along width and height), but extends through the full depth of the input volume.
- For example, a typical filter on a first layer of a ConvNet might have size 5x5x3 (i.e. 5 pixels width and height, and 3 because images have depth 3, the color channels)

Conv Layer

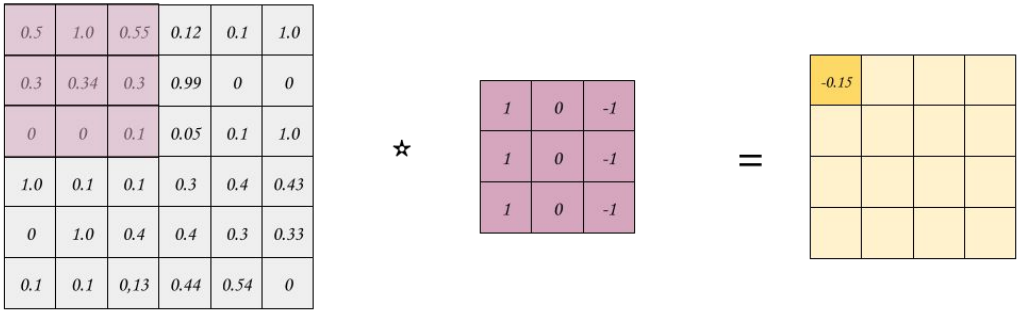
Consider 1 filter.



- During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and **compute dot products** between the entries of the filter and the input at any position.

Conv Layer

Convolution or correlation?



$\mathcal{M}$: 6x6 $\mathcal{K}$: 3x3 $\mathcal{C}$: 4x4

$$O_j = \sum_{i \in \mathcal{N}_j} F_i I_i$$

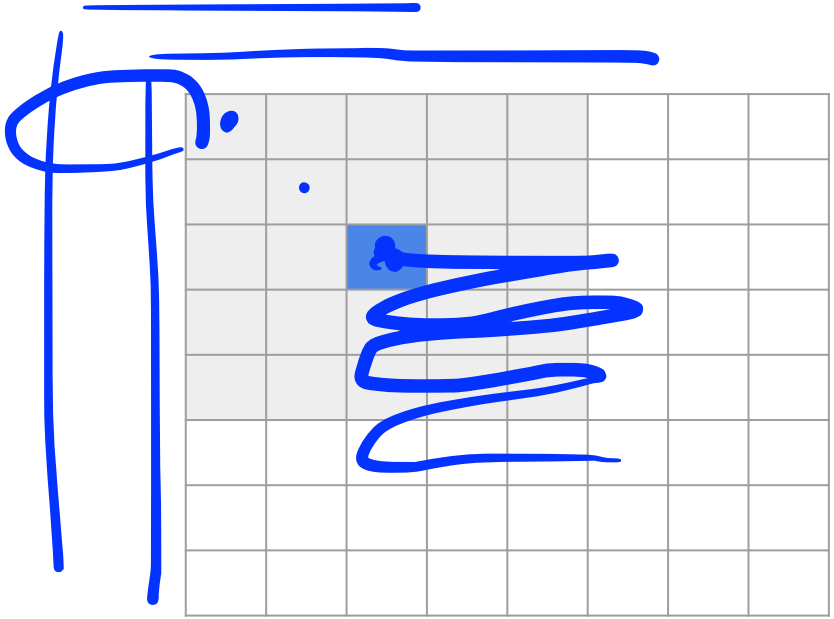
- Actually correlation!
- But note that when filter is symmetric: convolution=correlation

Conv Layer

Consider 1 filter.

Toy example:

- Input image: 8x8
- Filter size: 5x5

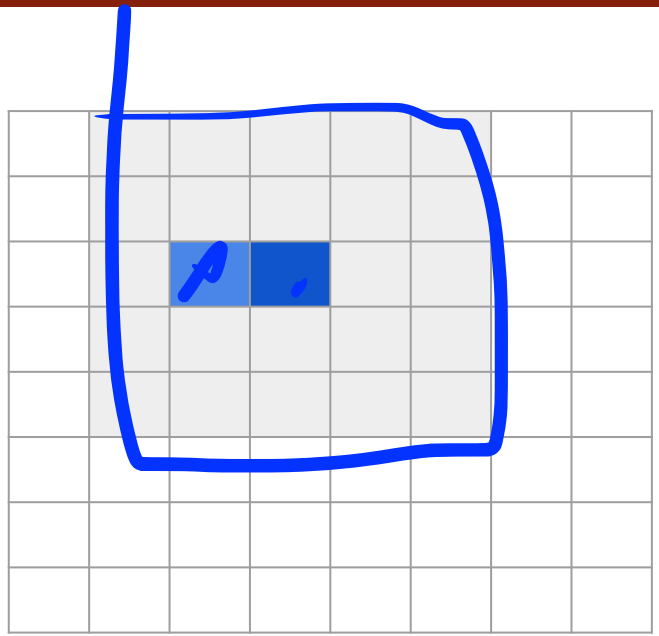


Conv Layer

Consider 1 filter.

Toy example:

- Input image: 8x8
- Filter size: 5x5

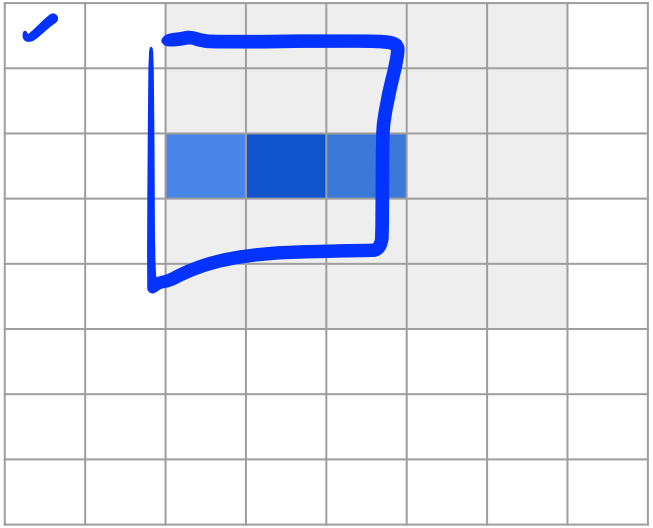


Conv Layer

Consider 1 filter.

Toy example:

- Input image: 8x8
- Filter size: 5x5

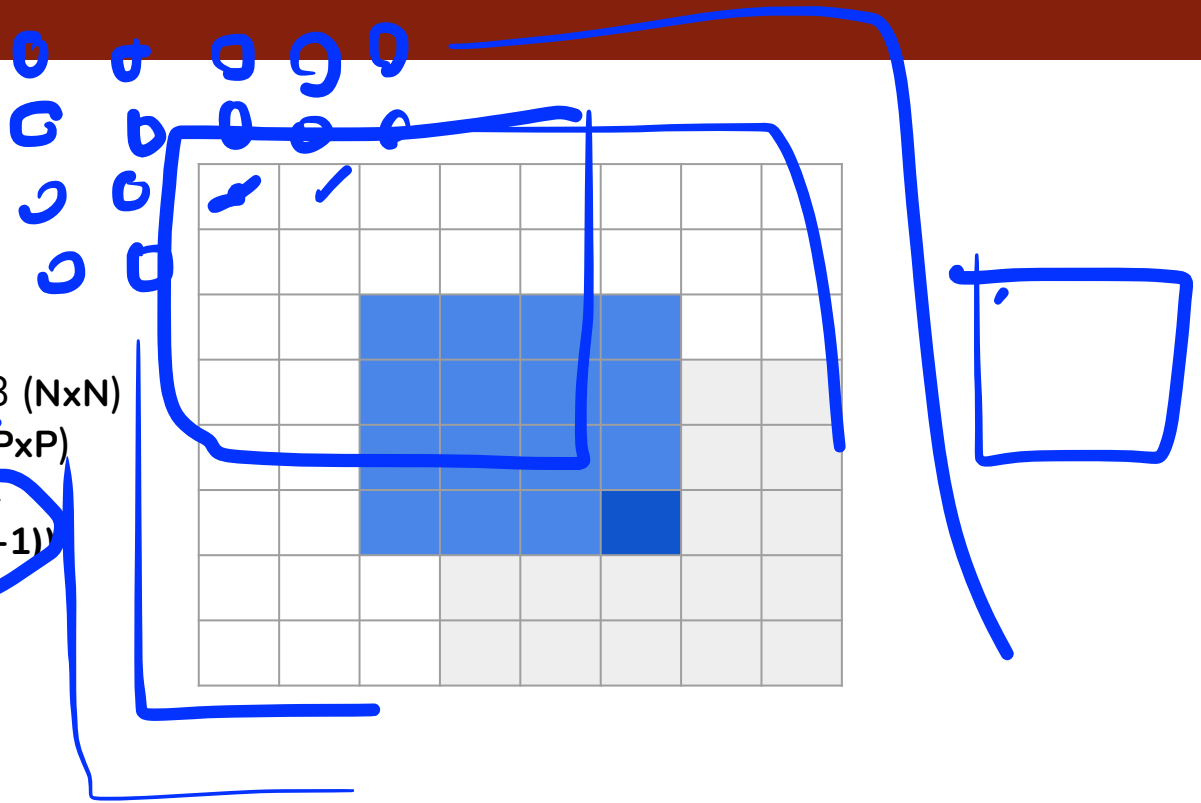


Conv Layer

Consider 1 filter.

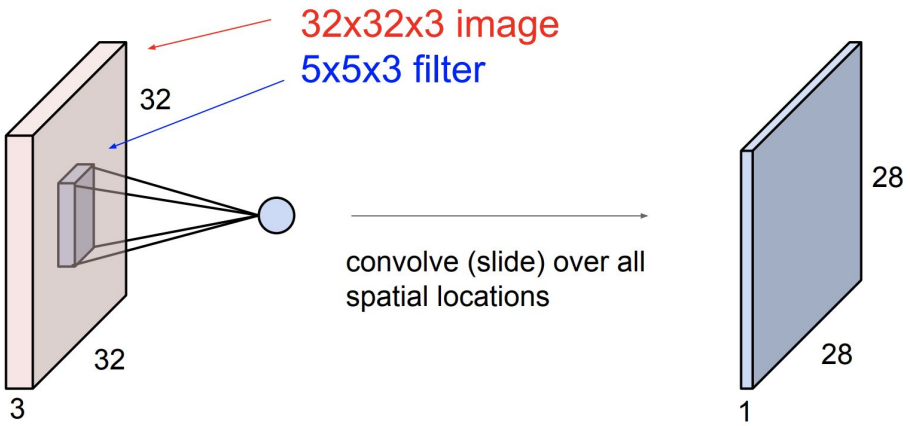
Toy example:

- Input image: 8x8 (**NxN**)
- Filter size: 5x5 (**PxP**)
- Output size: 4x4
  (N-(P-1)) x (N-(P-1))



Conv Layer

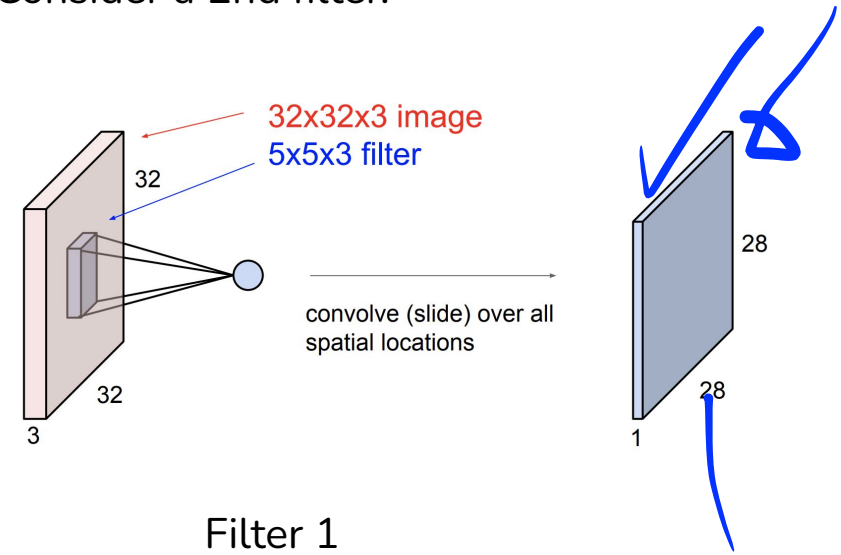
Consider 1 filter.



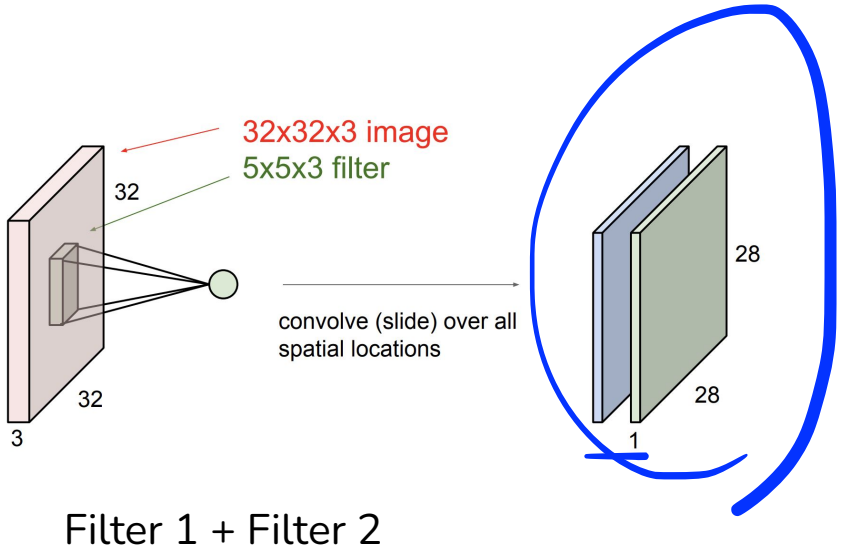
- N=32
- P=5
- N-(P-1) = 28

Conv Layer

Consider a 2nd filter.



Filter 1

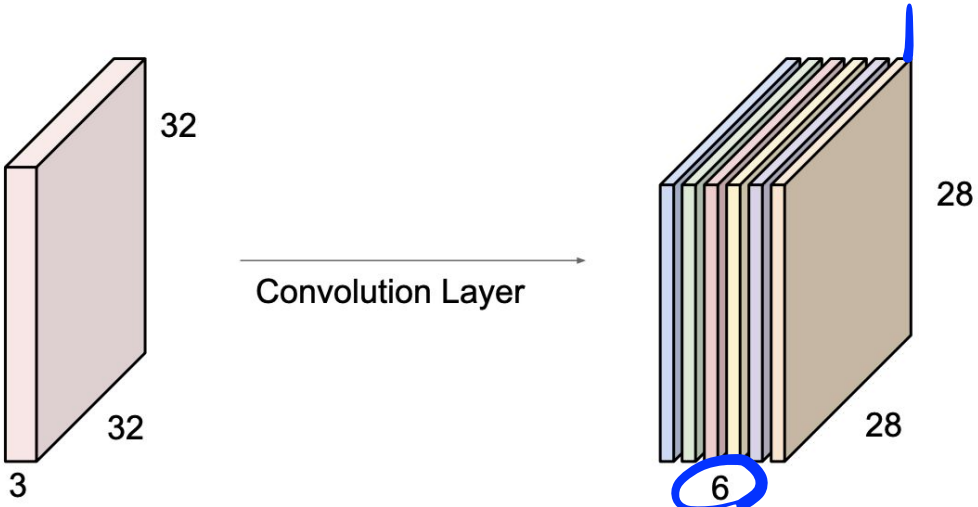


Filter 1 + Filter 2

Conv Layer

Consider 6 filters.

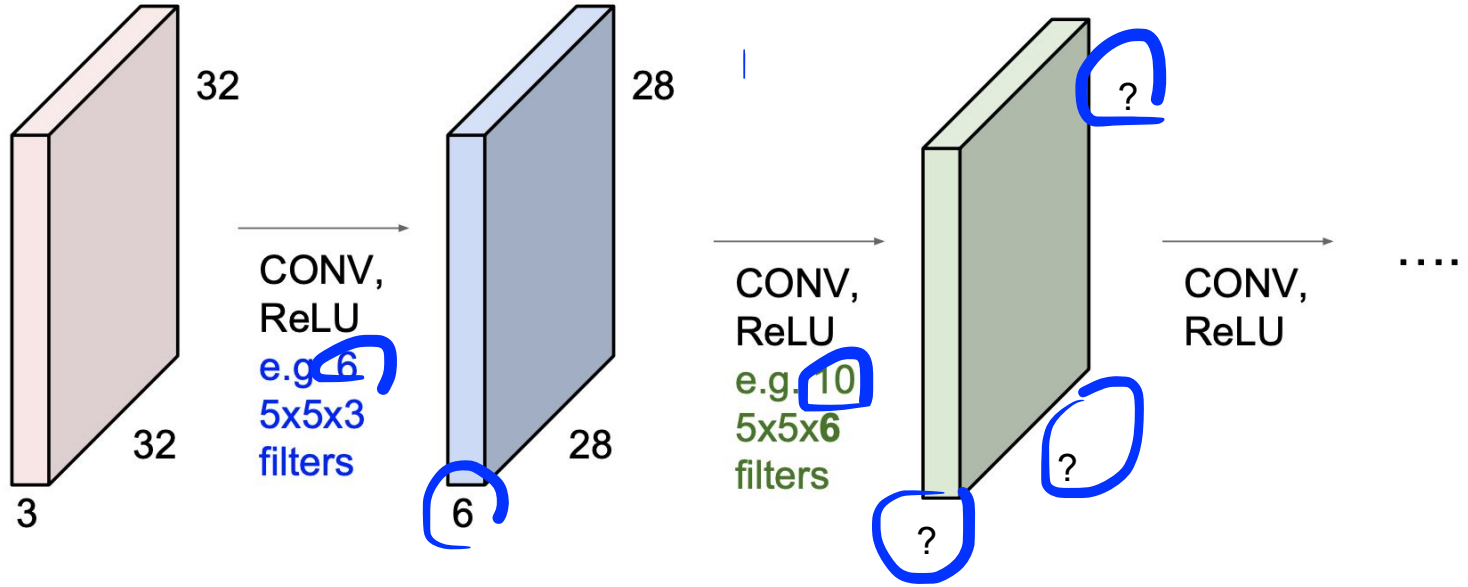
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

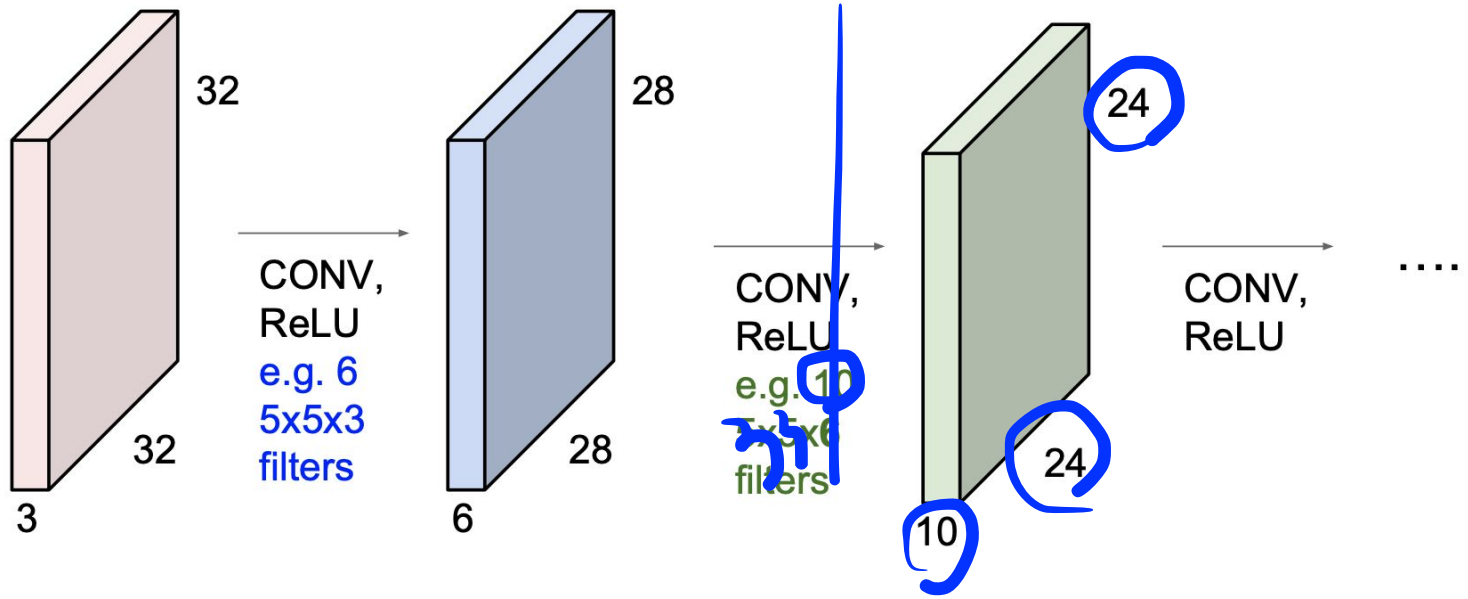
Conv Layer

Consider stacked Conv Layers:



Conv Layer

Consider stacked Conv Layers:



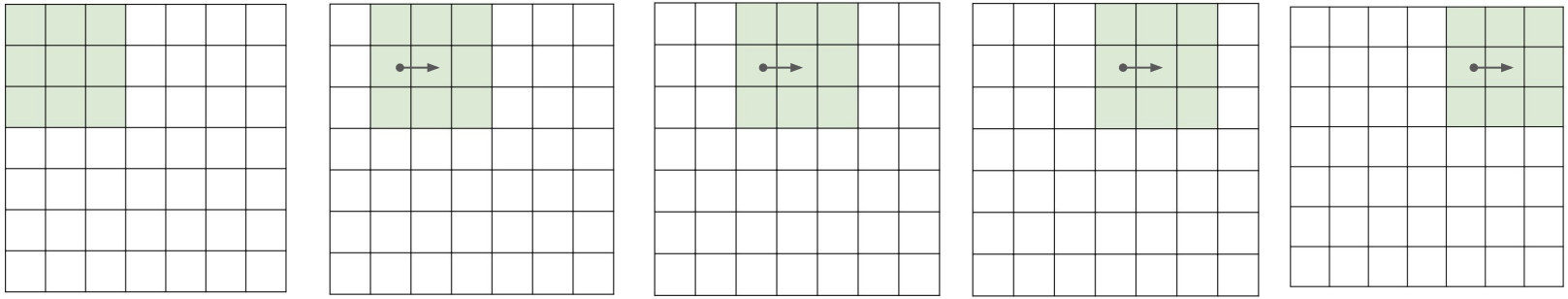
Conv Layer: Hyperparameters

Hyperparameters:

- Number of filters
- Filter size
- Stride
- Padding

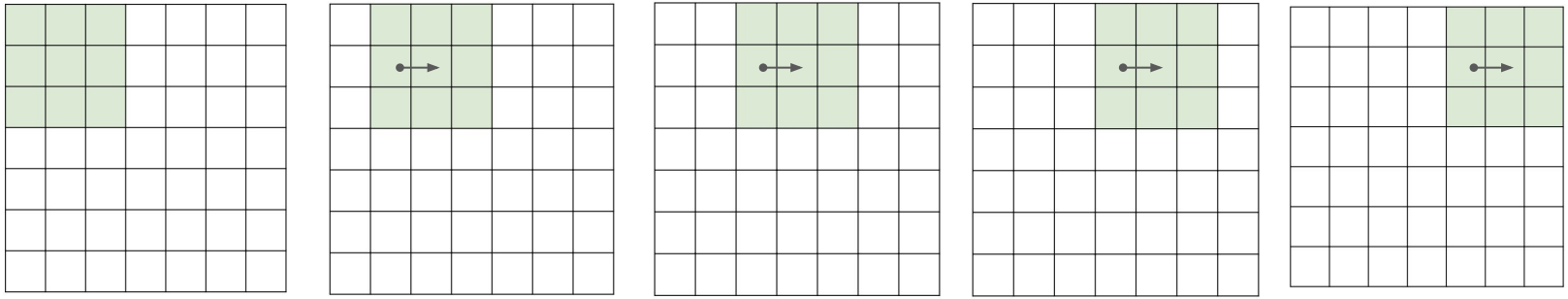
Conv Layer: Strides

- We must specify the **stride** with which we slide the filter.
- When the stride is 1 then we move the filters one pixel at a time.
- E.g., N=7, P=3, stride=1
- What's the output size?



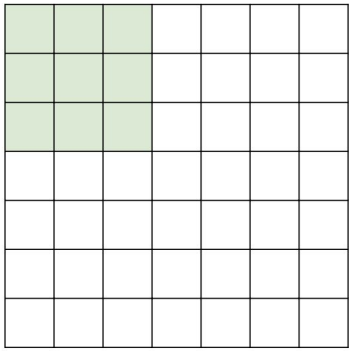
Conv Layer: Strides

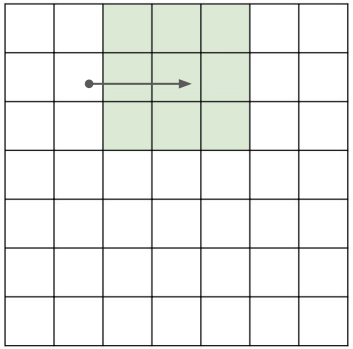
- We must specify the **stride** with which we slide the filter.
- When the stride is 1 then we move the filters one pixel at a time.
- E.g., N=7, P=3, stride=1
- What's the output size? **5x5**

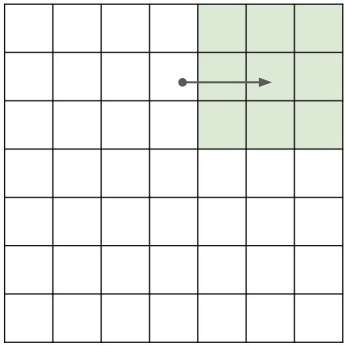


Conv Layer: Strides

- We must specify the **stride** with which we slide the filter.
- When the stride is 1 then we move the filters one pixel at a time.
- E.g., N=7, P=3, stride=2
- What's the output size?

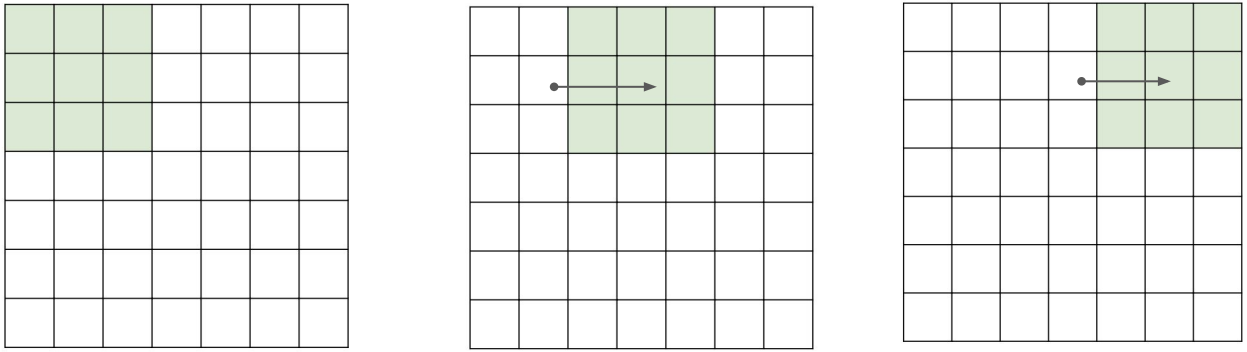






Conv Layer: Strides

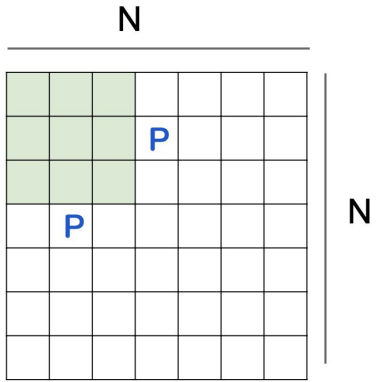
- We must specify the **stride** with which we slide the filter.
- When the stride is 1 then we move the filters one pixel at a time.
- E.g., N=7, P=3, stride=2
- What's the output size? **3x3**



Conv Layer: Strides

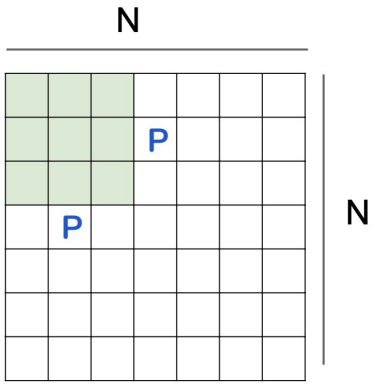
- We must specify the **stride** with which we slide the filter.
- When the stride is 1 then we move the filters one pixel at a time.
- E.g., N=7, P=3, stride=2
- What's the output size? **3x3**

Output size: **(N - P) / stride + 1**



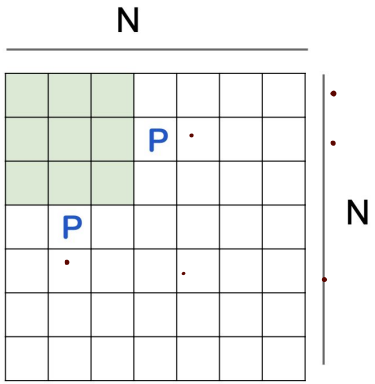
Conv Layer: Strides

- What happens if: N=7, P=3, stride=3?



Conv Layer: Strides

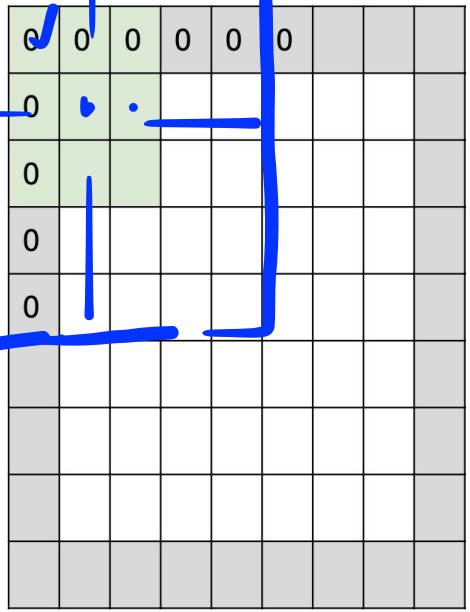
- What happens if: N=7, P=3, stride=3?



Output size: (N - P) / stride + 1
= (7-3)/3 + 1
= 2.33

Conv Layer: Padding

In practice: Common to zero pad the border

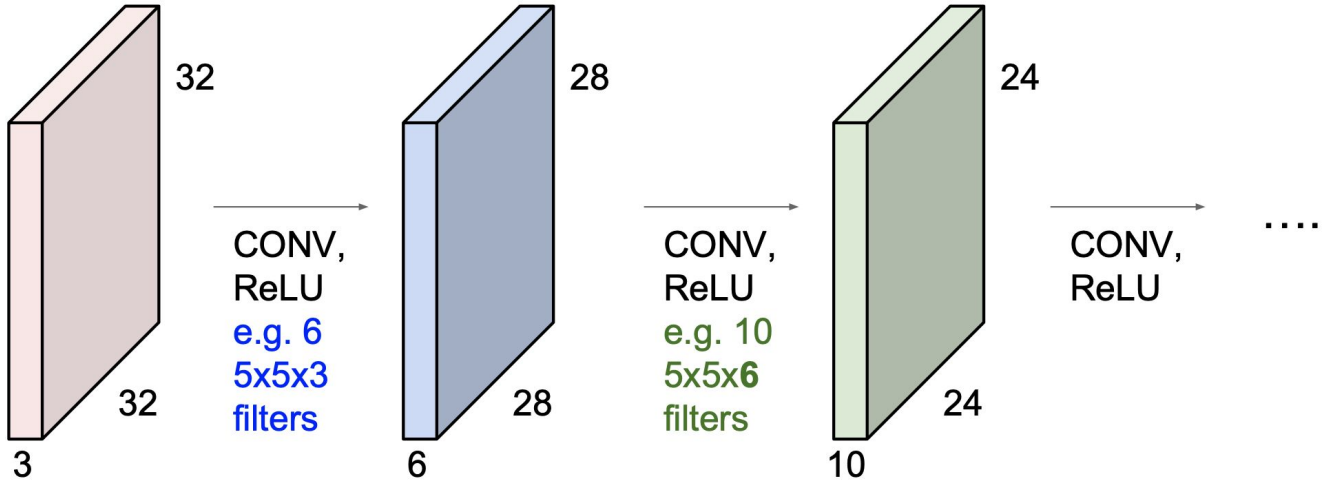


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size PxP and zero-padding with (P-1)/2. (will preserve size spatially)
e.g. P = 3 => zero pad with 1
P = 5 => zero pad with 2
P = 7 => zero pad with 3

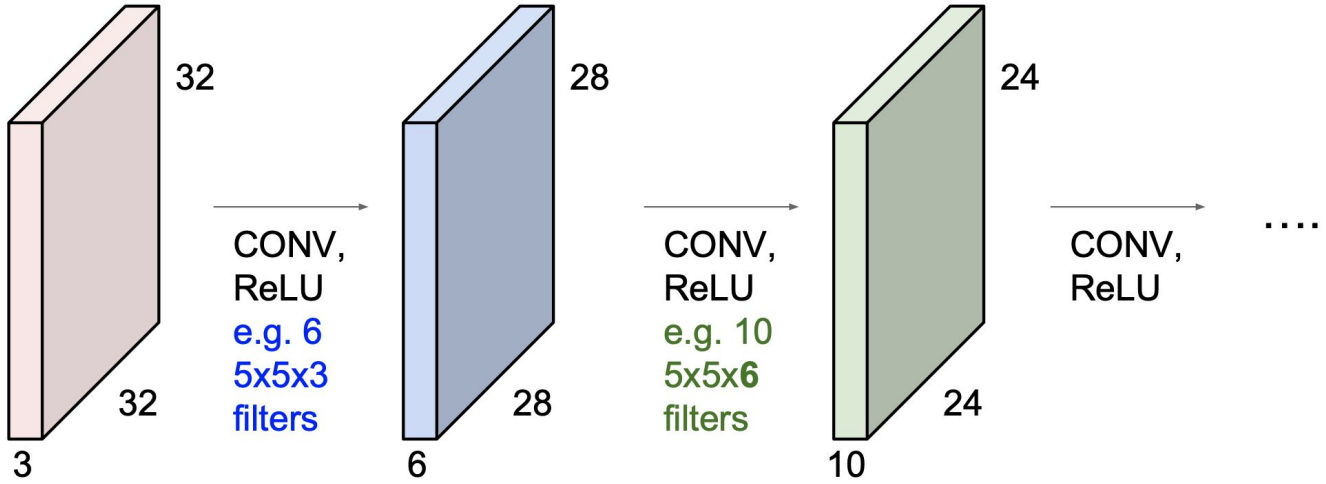
Conv Layer

- Without padding, convolving with same filter size will reduce spatial dimension.



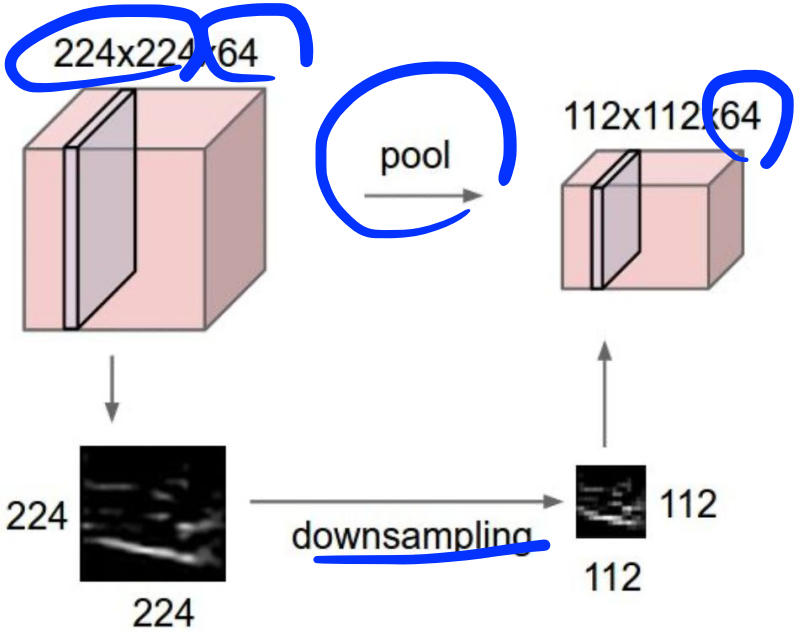
Conv Layer: No. of filters

- Number of filters are increased over the layers to increase depth of the outputs.



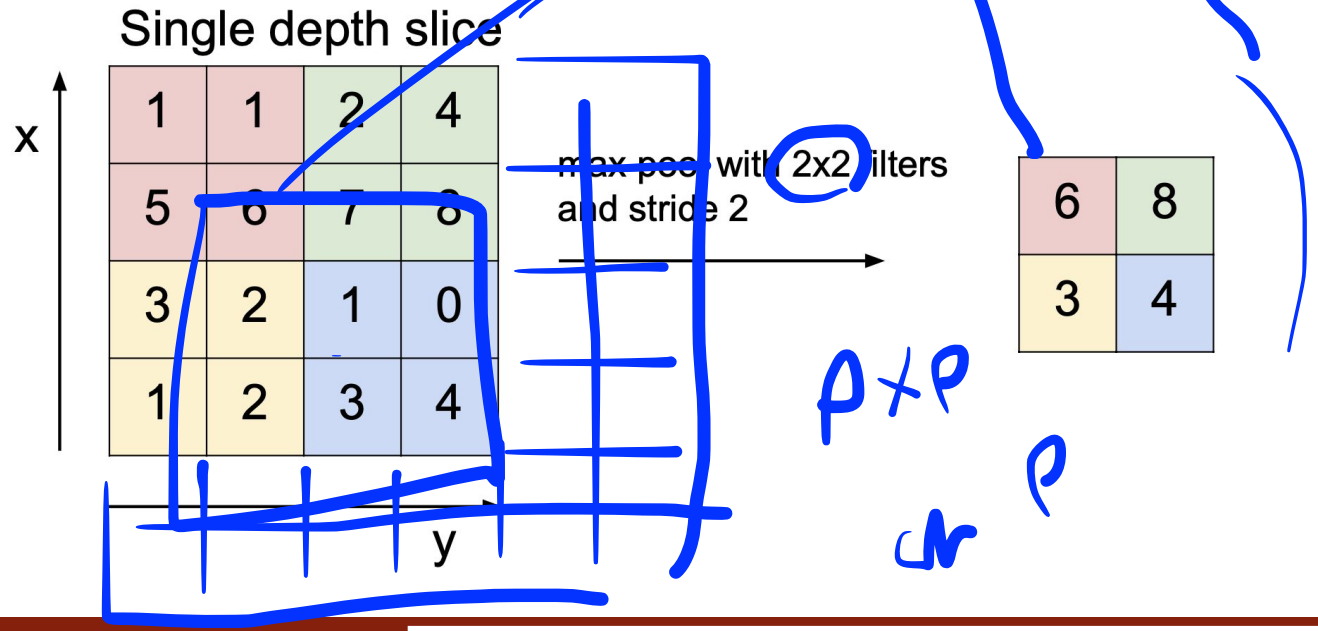
Pooling Layer

- Makes representations smaller and more manageable.
- Global features



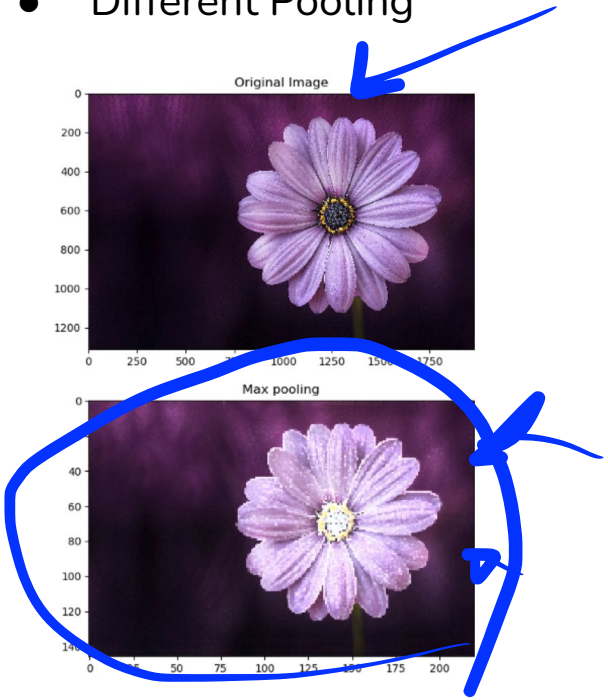
Pooling Layer

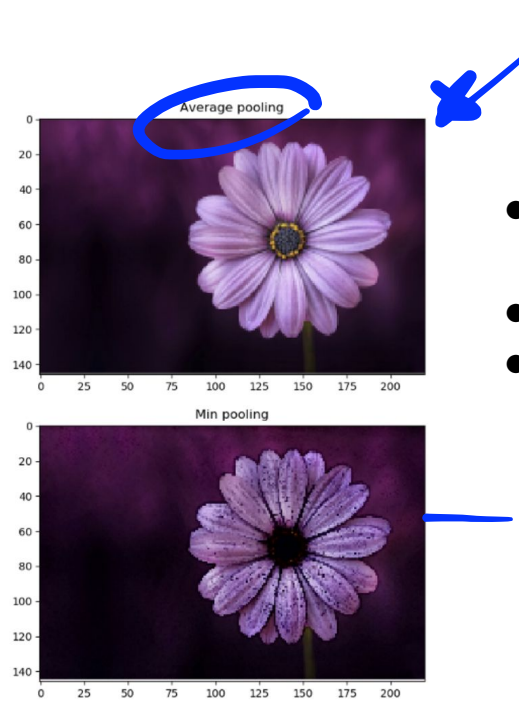
- MAX Pooling



Pooling Layer

- Different Pooling





- MAX pooling is commonly used as it works well
- Extracts sharp features
- Faster to compute than convolution

Architecture of a simple CNN

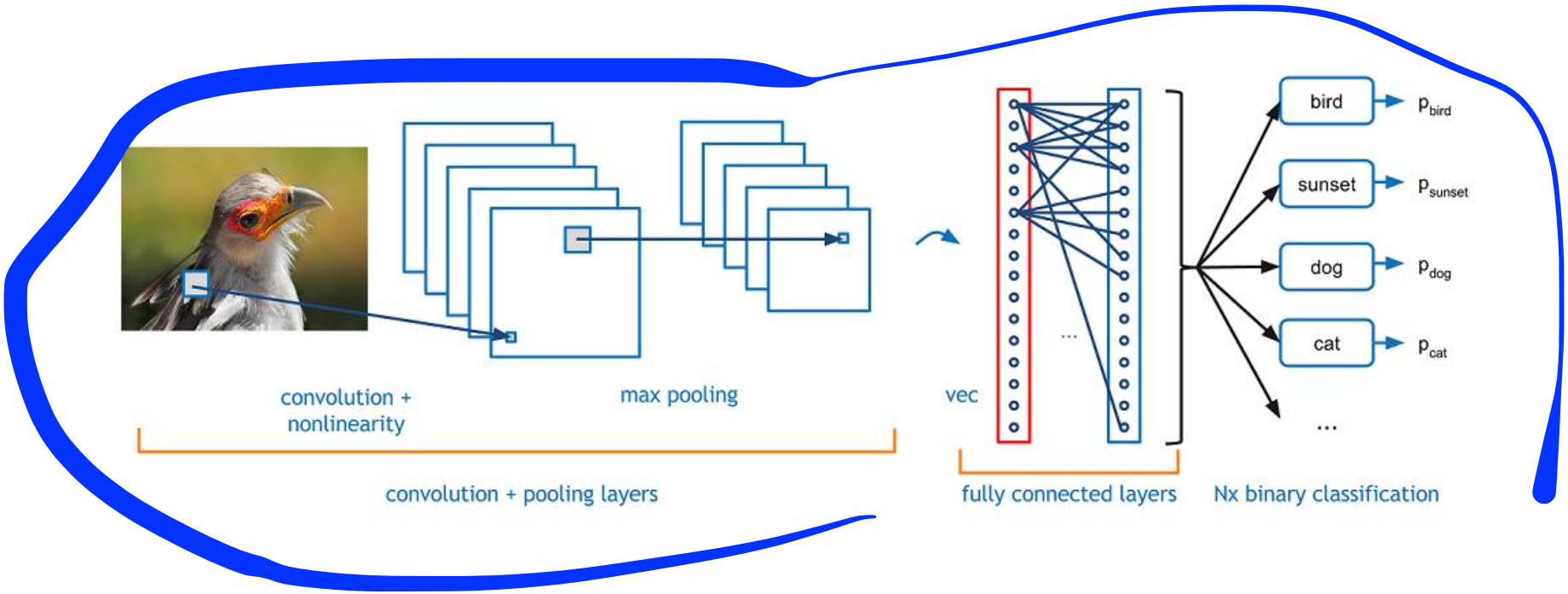
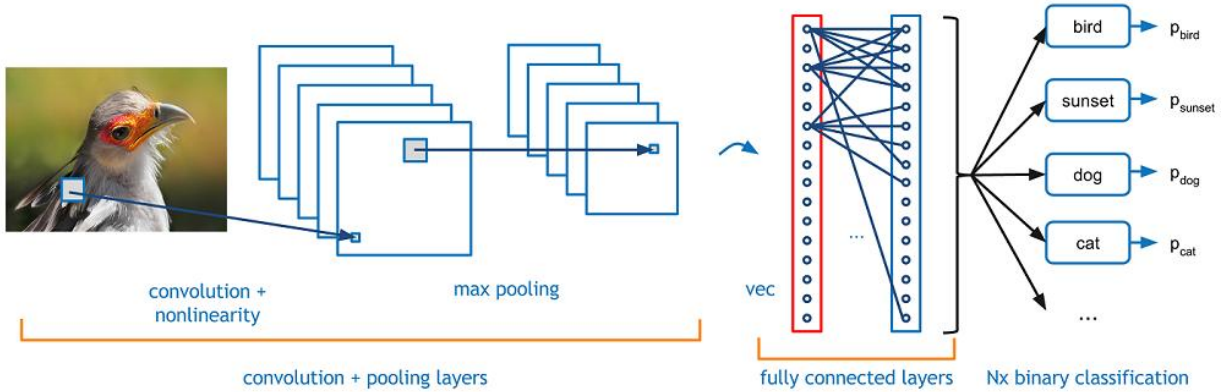


Figure credits: Adit Deshpande

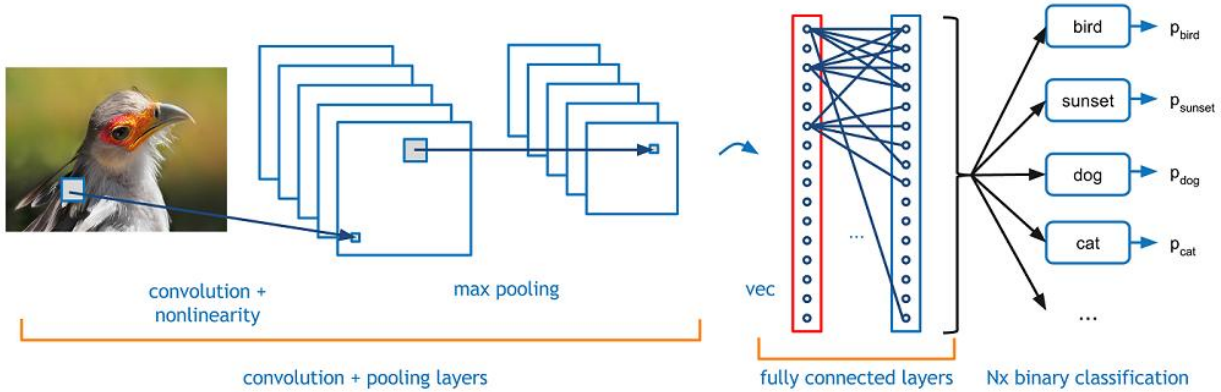
Architecture of a simple CNN



- Initially Conv layer with nonlinearity

Figure credits: Adit Deshpande

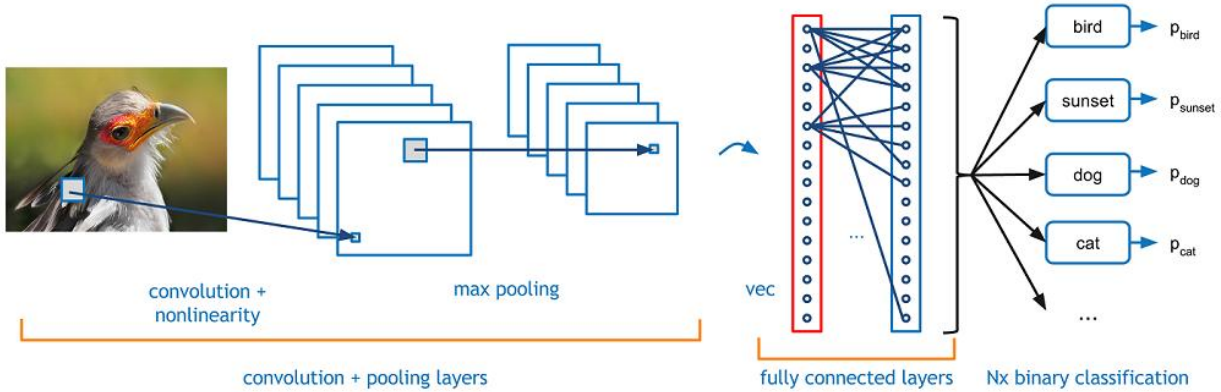
Architecture of a simple CNN



- Initially Conv layer with nonlinearity
- Followed by a few Conv + Nonlinearity layers

Figure credits: Adit Deshpande

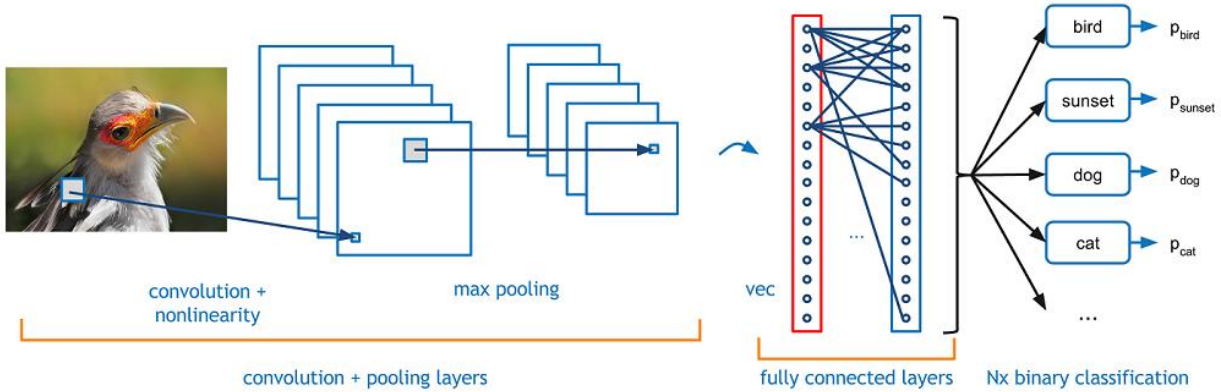
Architecture of a simple CNN



- Initially Conv layer with nonlinearity
- Followed by a few Conv + Nonlinearity layers
- Have Pooling layers in between Conv layers $\rightarrow$ reduce the feature map size sufficiently

Figure credits: Adit Deshpande

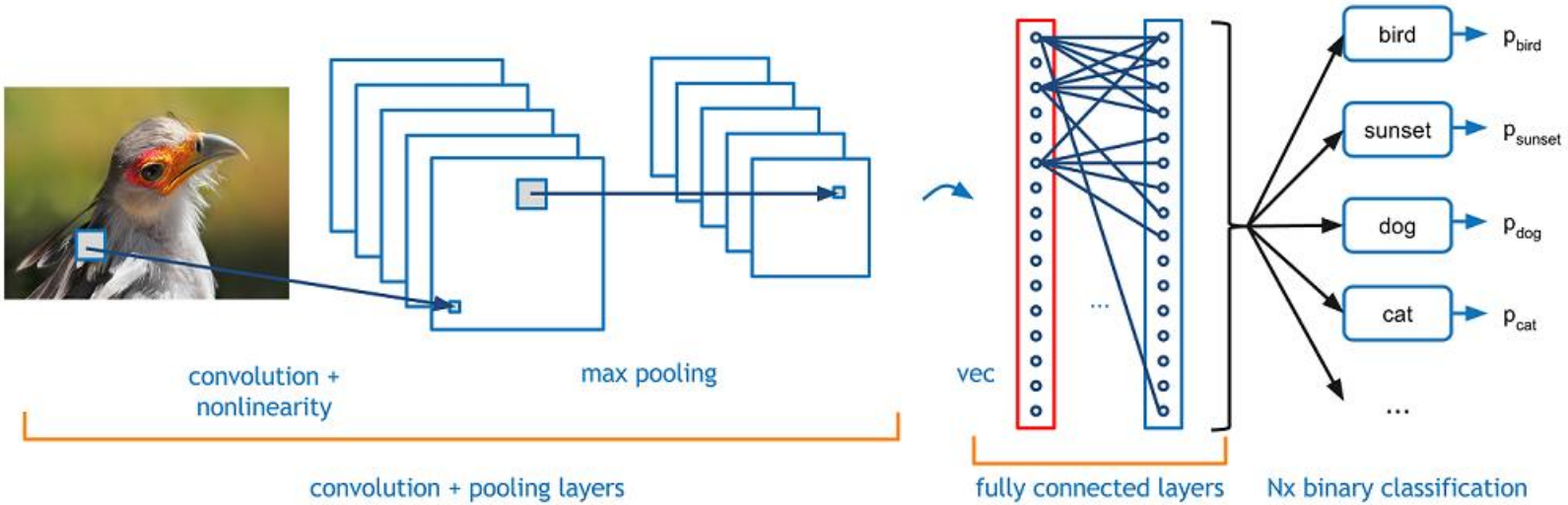
Architecture of a simple CNN



- Initially Conv layer with nonlinearity
- Followed by a few Conv + Nonlinearity layers
- Have Pooling layers in between Conv layers $\rightarrow$ reduce the feature map size sufficiently
- Vectorize and and fully connected layers

Figure credits: Adit Deshpande

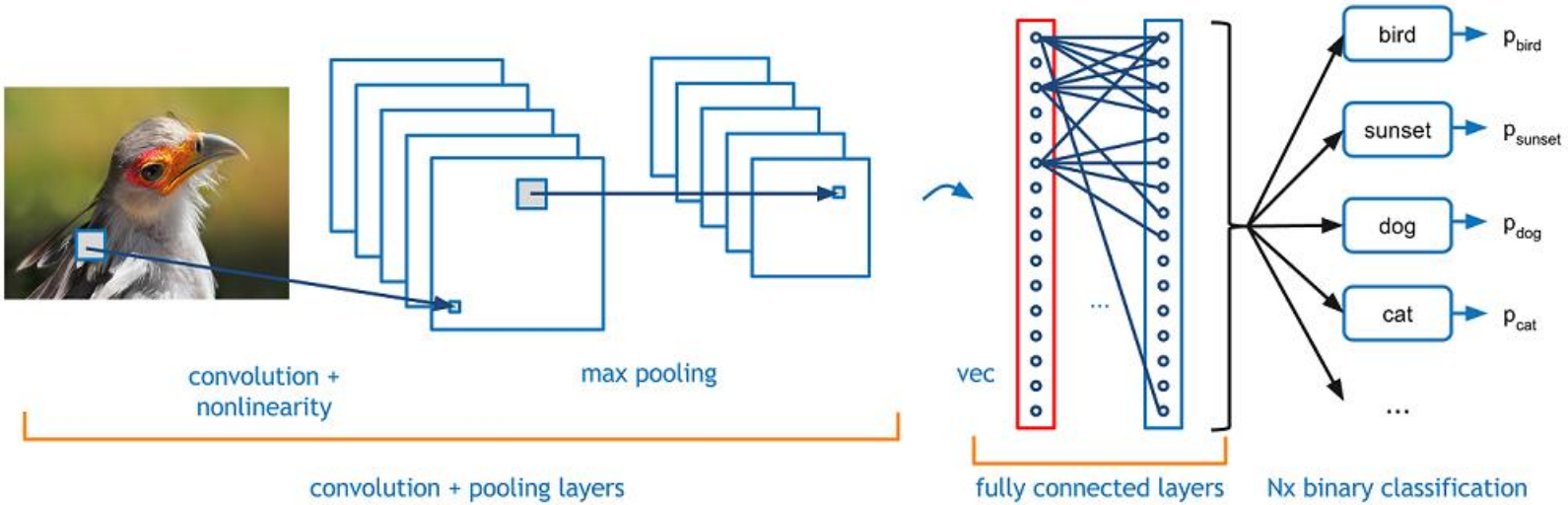
Architecture of a simple CNN



INPUT -> [[CONV -> RELU] *N -> POOL]*M -> [FC->RELU]*K -> FC

Figure credits: Adit Deshpande

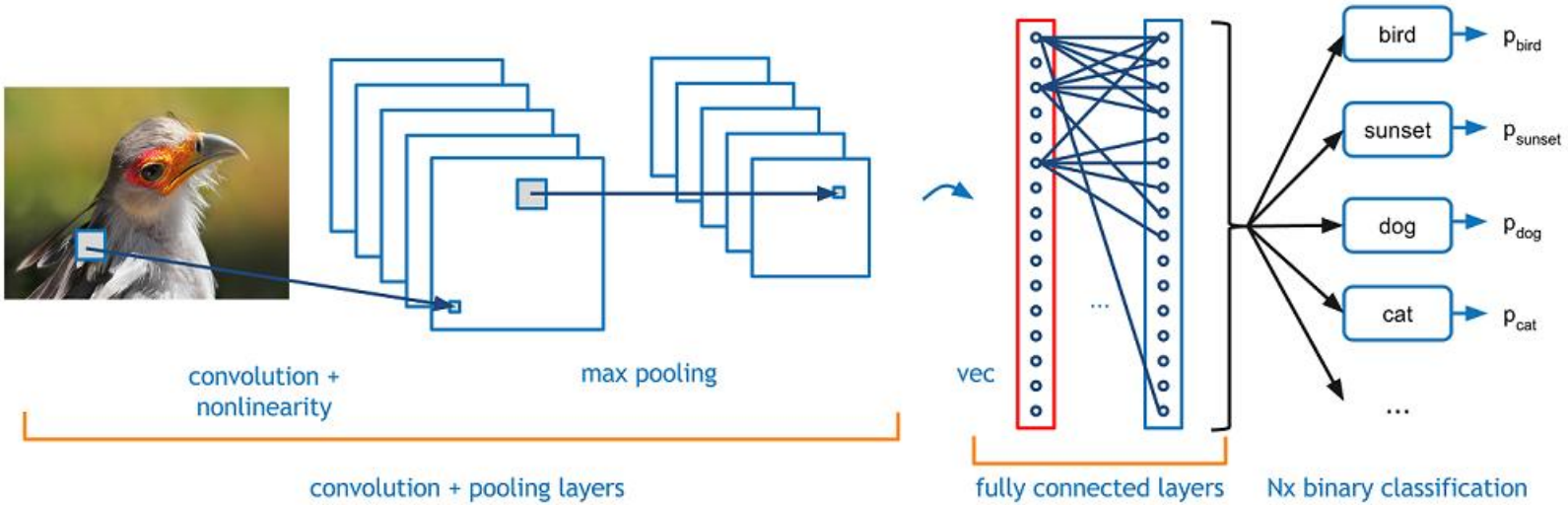
Architecture of a simple CNN



INPUT -> [[CONV -> RELU] *N -> POOL]*M -> [FC->RELU]*K -> FC

Figure credits: Adit Deshpande

Architecture of a simple CNN



INPUT -> [[CONV -> RELU] *N -> POOL]*M -> [FC->RELU]*K -> FC

Figure credits: Adit Deshpande

Case study: LeNet-like architecture

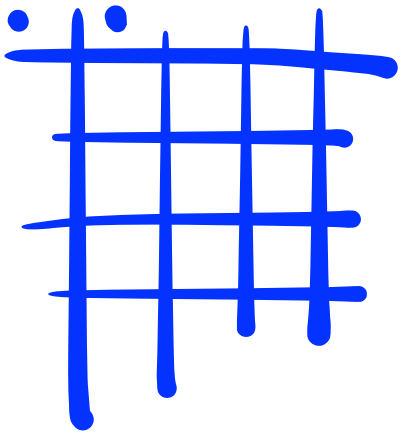
input size/ layer information	output size	# parameters	# products
$1 \times 28 \times 28$ `nn.Conv2d(1, 32, kernel_size=5)`			

Case study: LeNet-like architecture

input size/ layer information	output size	# parameters	# products
$1 \times 28 \times 28$ `nn.Conv2d(1, 32, kernel_size=5)`	$32 \times 24 \times 24$		

Case study: LeNet-like architecture

input size/ layer information	output size	# parameters	# products
$1 \times 28 \times 28$ `nn.Conv2d(1, 32, kernel_size=5)`	$32 \times 24 \times 24$	$32.(5^2 + 1)$ $= 832$	



Case study: LeNet-like architecture

input size/ layer information	output size	# parameters	# products
$1 \times 28 \times 28$ `nn.Conv2d(1, 32, kernel_size=5)`	$32 \times 24 \times 24$	$32.(5^2 + 1)$ $= 832$	$32.24^2.5^2$ $= 460800$
$32 \times 24 \times 24$ `F.max_pool2d(., kernel_size=3)`	✓	✓	