

Machine Learning (CE 40717)

Fall 2024

Ali Sharifi-Zarchi

CE Department
Sharif University of Technology

October 27, 2024



1 Motivation

2 CNN vs. FCN

3 Convolution

4 Pooling

1 Motivation

2 CNN vs. FCN

3 Convolution

4 Pooling

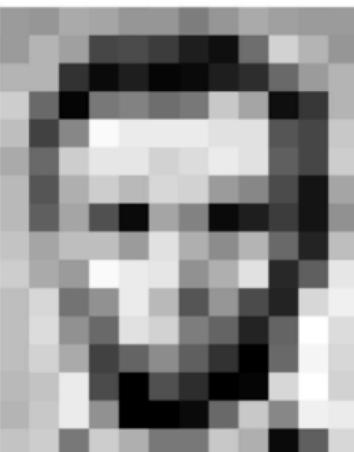
How do humans see?



How can we enable computers to see?

What computers see: images as numbers

What you see



Input Image

What you both see

157	153	174	168	150	152	129	161	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	54	6	10	33	48	106	159	187
256	109	5	124	131	111	120	204	166	15	56	186
194	68	137	251	237	239	239	228	227	87	71	207
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	216	211	158	139	76	20	161
189	97	165	84	10	168	134	11	31	62	22	149
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	91	234
190	216	116	149	236	187	85	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	222
190	214	173	66	103	143	96	50	2	109	249	211
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	211

Input Image + values

What the computer "sees"

157	153	174	168	150	152	129	151	172	161	155	151
155	182	163	74	75	62	33	17	110	210	180	151
180	180	50	14	34	6	10	33	48	106	199	181
206	109	5	124	131	111	120	204	166	15	56	181
194	68	137	251	237	239	239	228	227	87	71	25
172	105	207	233	233	214	220	239	228	98	74	25
188	88	179	209	185	215	211	158	139	75	20	16
189	97	165	84	10	168	134	11	31	62	22	14
199	168	191	193	158	227	178	143	182	106	36	19
205	174	155	252	236	231	149	178	228	43	95	21
190	216	116	149	236	187	86	150	79	38	218	24
190	224	147	108	227	210	127	102	36	101	255	22
190	214	173	66	103	143	96	50	2	109	249	21
187	196	235	75	1	81	47	0	6	217	295	21
183	202	237	145	0	0	72	108	200	138	243	21
195	206	123	207	177	121	123	200	175	13	96	21

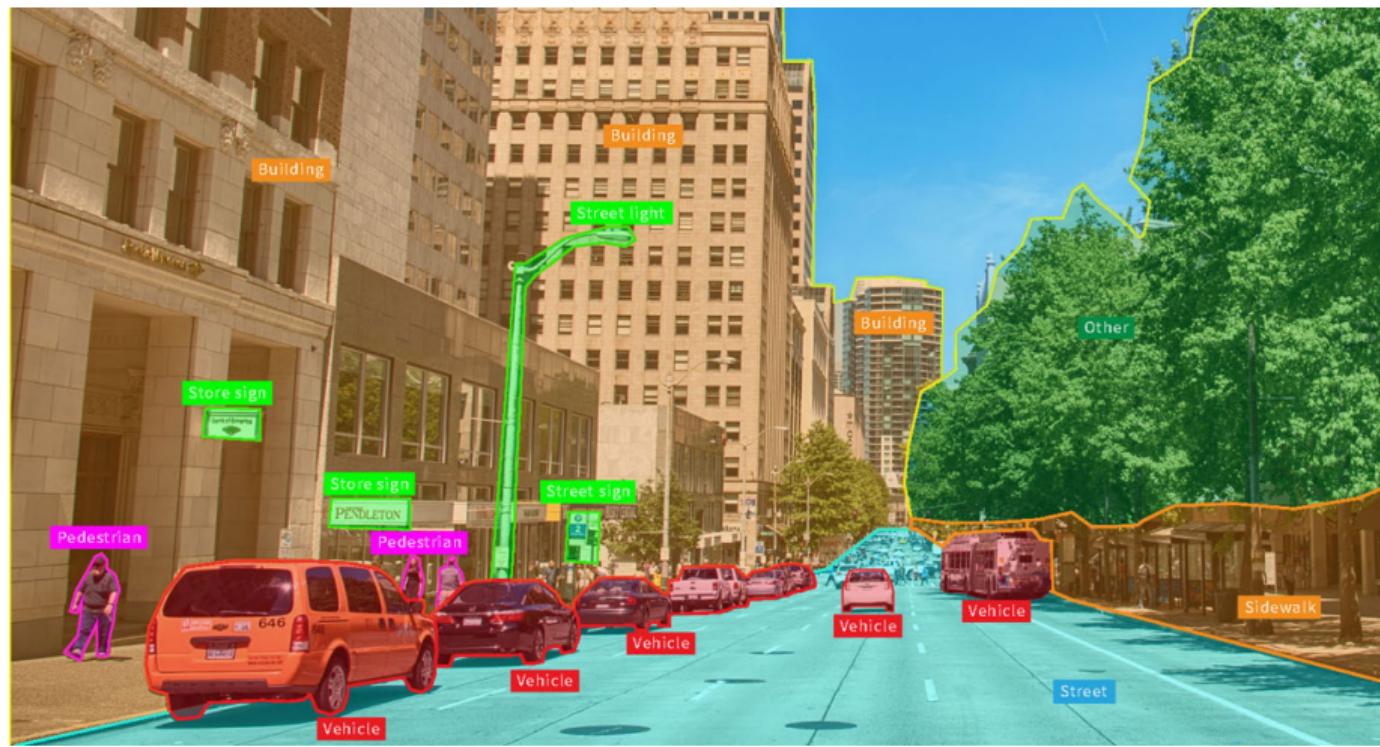
Pixel intensity values
("pix-el"=picture-element)

Ergonomic Design of Twin Image Processing & Computer Vision

An image is just a matrix of numbers .i.e., $1080 \times 1080 \times 3$ for an RGB image.

Question: is this Lincoln? Washington? Jefferson? Obama? How can the computer answer this question?

What is computer vision? (1)



What is computer vision? (2)



Noisy Image

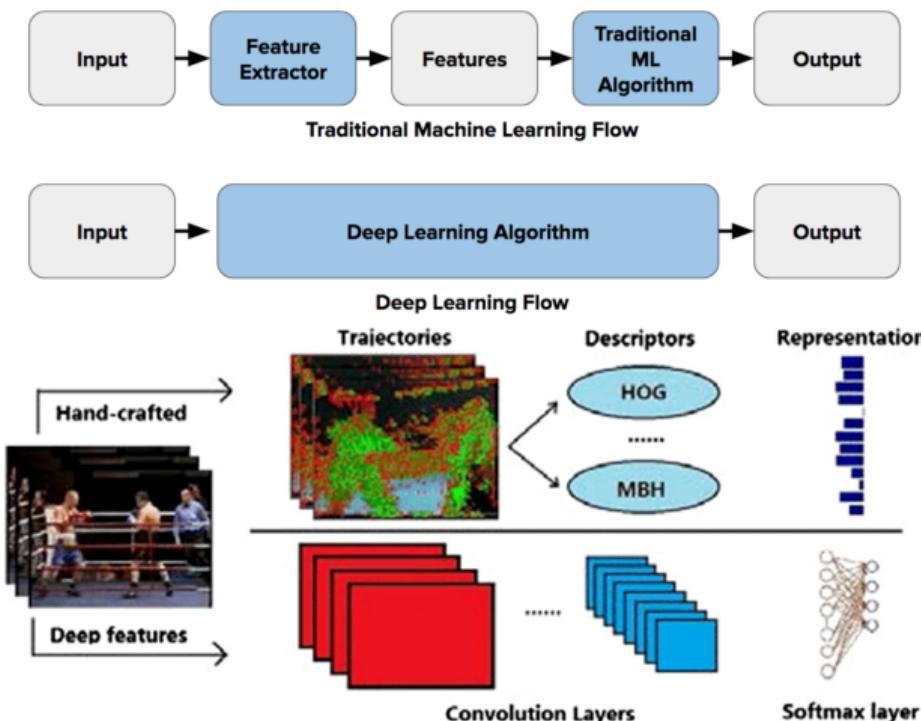


Denoised Image

What is computer vision? (3)



Hand-crafted features: before deep learning features (1)



Two types of features used for action recognition, i.e., hand-crafted features and deep learning features

Hand-crafted features: before deep learning features (2)

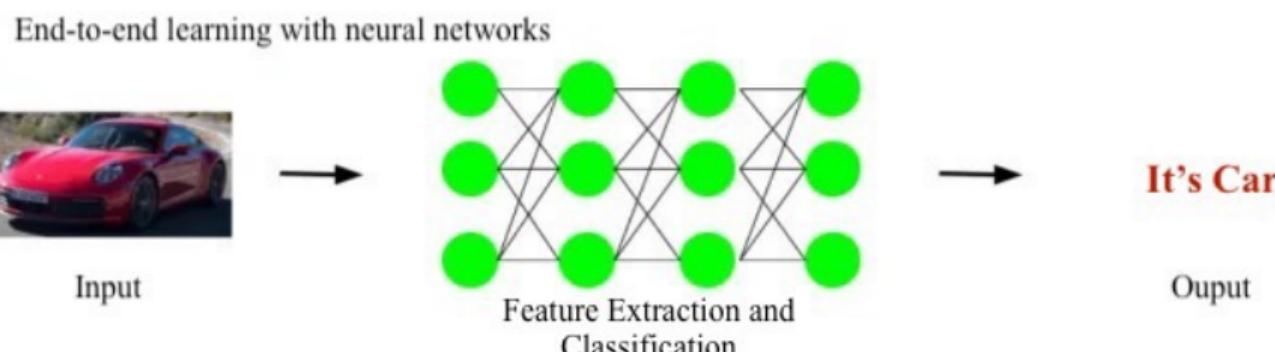
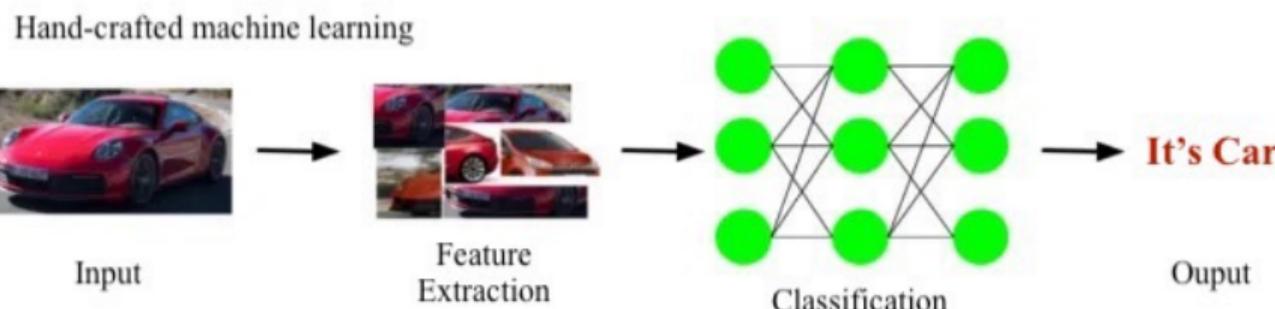
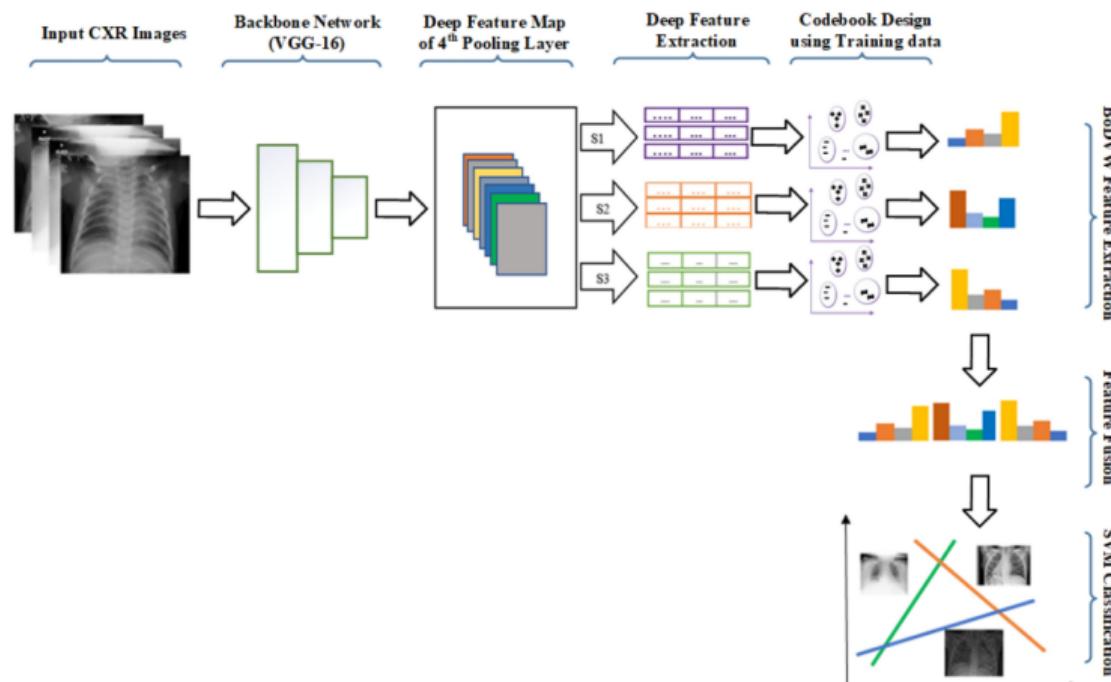
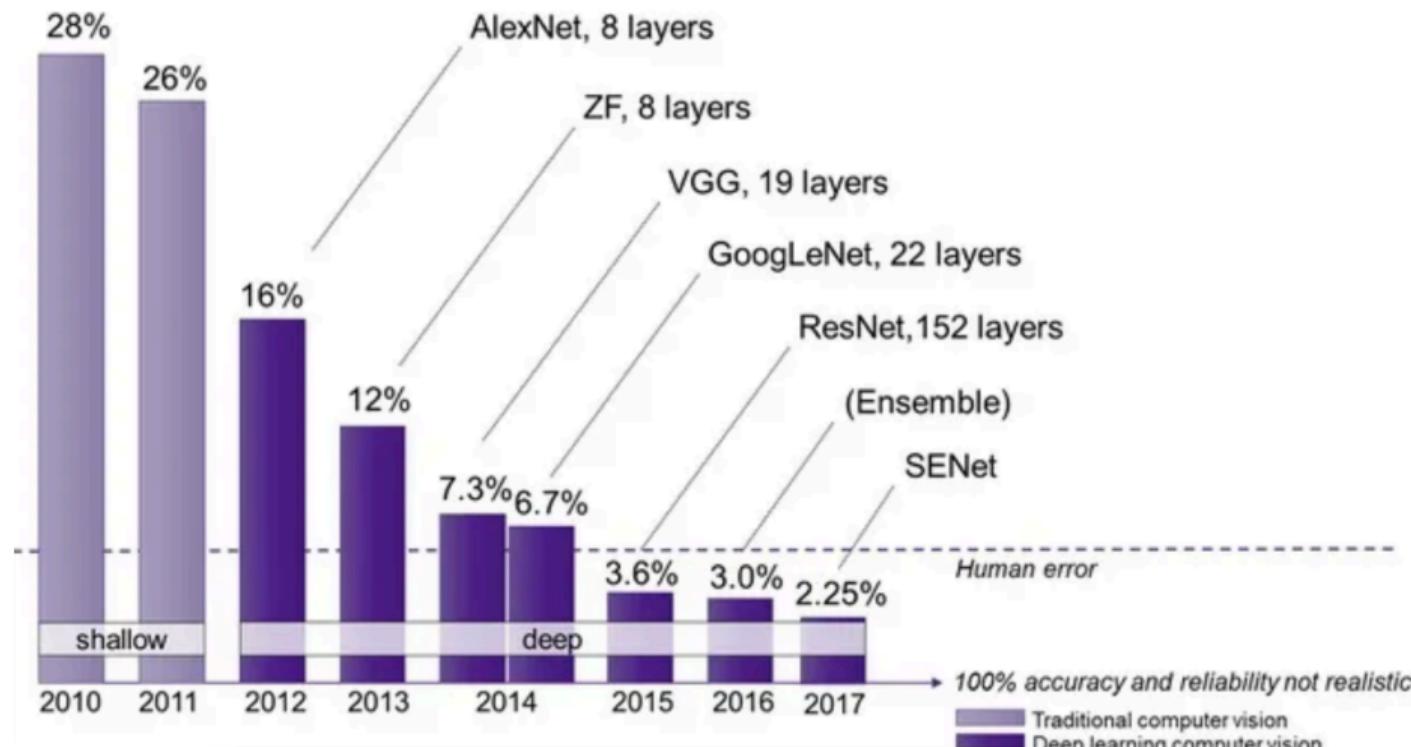


Image features



Fusion of multi-scale bag of deep visual words features of chest X-ray images to detect COVID-19 infection

Effect of dl



Comparison between Deep Learning and Traditional Models

1 Motivation

2 CNN vs. FCN

3 Convolution

4 Pooling

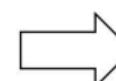
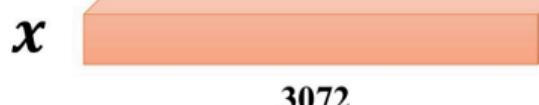
FCNs on images

- What we've been using?
 - Dense Vector Multiplication

32×32×3 image

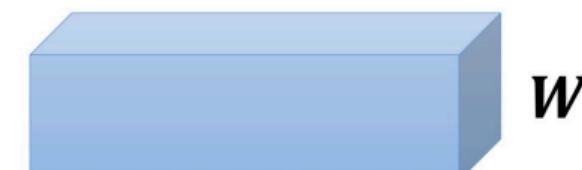


Reshape it into
a vector



An input-sized weight vector
for each hidden neuron

100×3072



$$Wx^T$$



Each element contains the
activation of 1 neuron

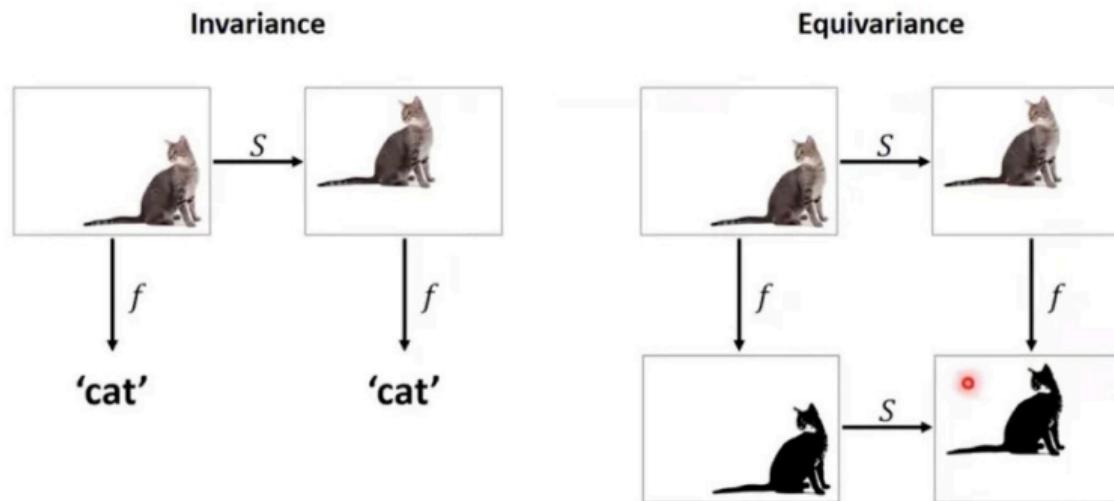
Fully connected layers

- Neurons in a single layer operate independently and do not share connections, even for inputs.
- To be shift-invariant, many samples (in various time or locations) must be shown to them.
- **Regular Neural Nets dont scale well to full images**
 - Parameters would add up quickly!
 - Full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

Solution?

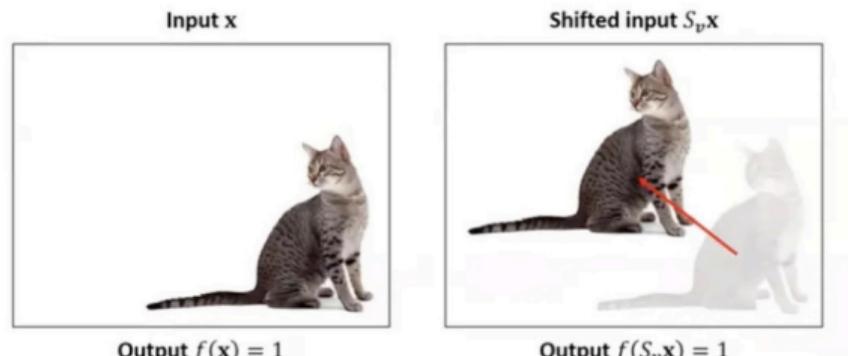
What can we do?

Invariance vs. equivariance



- **Invariance:** The output remains the same when the input undergoes a transformation.
- **Equivariance:** The output varies predictably when the input undergoes a transformation.

Shift-invariance



- Cat detector $f : \mathbb{R}^d \rightarrow \mathbb{R}$
- Shift operator $S_v : \mathbb{R}^d \rightarrow \mathbb{R}^d$ shifting the image by vector v

Shift invariance: $f(\mathbf{x}) = f(S_v \mathbf{x})$

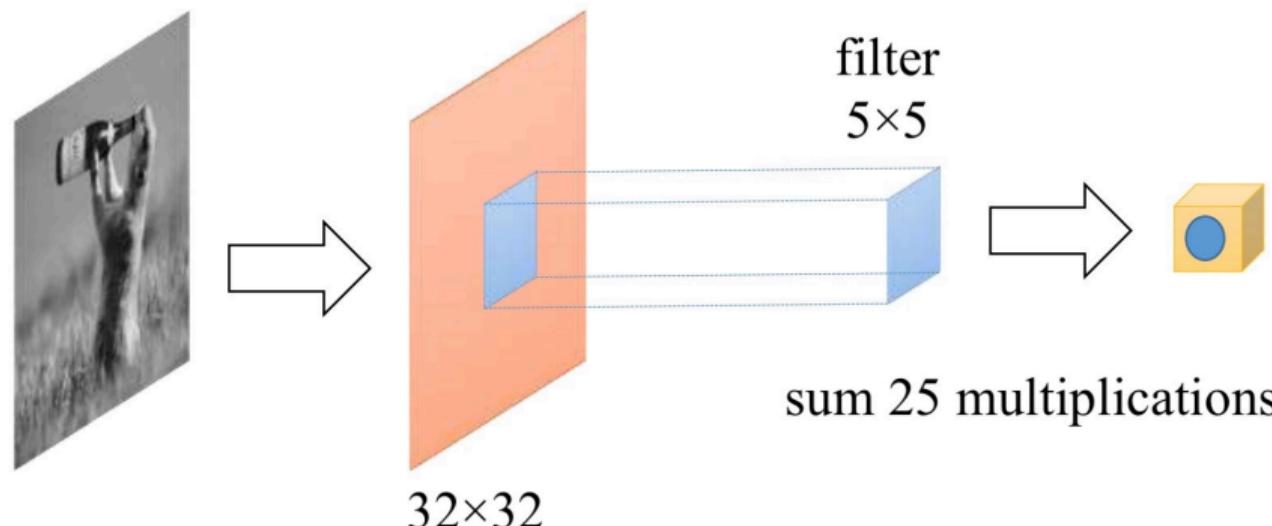
Will an MLP that recognizes the left image as a cat also recognize the shifted image on the right as a cat?

A problem

- In many problems the *location* of a pattern is not important
 - Only the presence of the pattern matters
- Traditional MLPs are sensitive to pattern location
 - Moving it by one component results in an entirely different input that the MLP wont recognize
- **Requirement:** Network must be *shift invariant*

Convolution on images

Convolution (Refresher)



Matrix input preserving **spatial structure**

Fully connected or convolution

Question: Can I just do classification on an flatten image (e.g., a 1080x1080x3 image) with fully connected layers?

Answer: No, using fully connected layers on such a large image is inefficient due to:

- Loss of spatial structure
- High parameter count

Why Use Convolution:

- **Exploit spatial structure:** Convolution sees local patterns by using filters over small patches of the image.
- **Reduce parameters:** By sharing weights across the image, convolution reduce the number of parameters.
- **Build hierarchical features:** Convolution starts with small patterns, then combines them to recognize more complex patters.

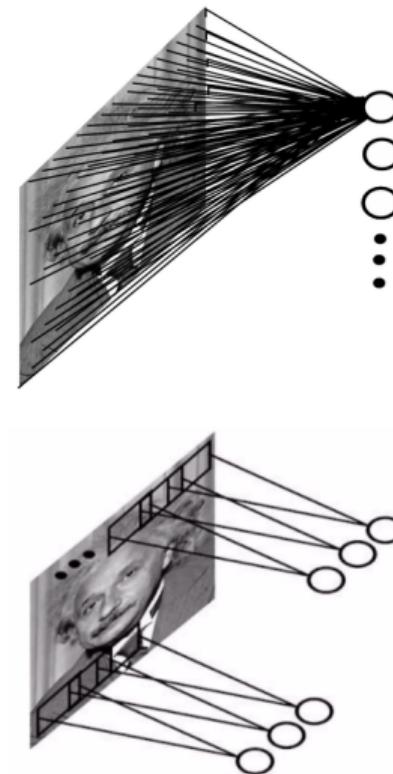
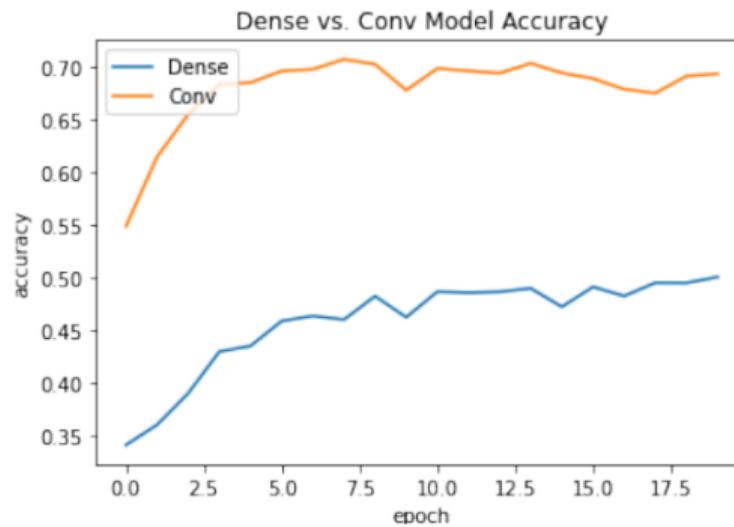
Motivation
oooooooooooo

CNN vs. FCN
oooooooooooo●oooooooooooo

Convolution
oooooooooooooooooooooooooooo

Pooling
oooooooooooooooo

Model Architecture	# Params	Test Accuracy
CNN	2M	67.8%
FCN	9M	50.6%

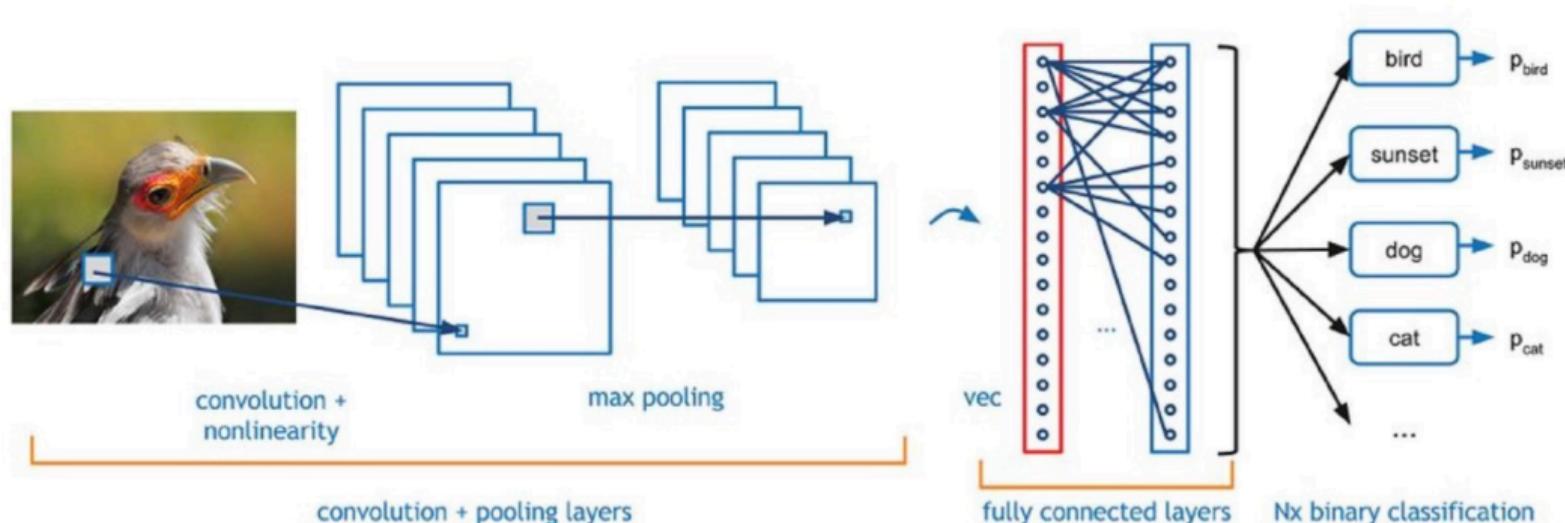


What is cnn?

- It is a class of deep learning.
- Convolutional neural network (ConvNets or CNNs) is one of the main categories to do image recognition, image classifications, object detection, recognition of faces, etc.
- It is similar to the basic neural network. CNNs also have learnable parameters like weights and biases, similar to neural networks.
- CNN is heavily used in computer vision.
- There are 3 basic components to define CNN:
 - The Convolution Layer
 - The Pooling Layer
 - The Output Layer (or) Fully Connected Layer

Architecture of CNN

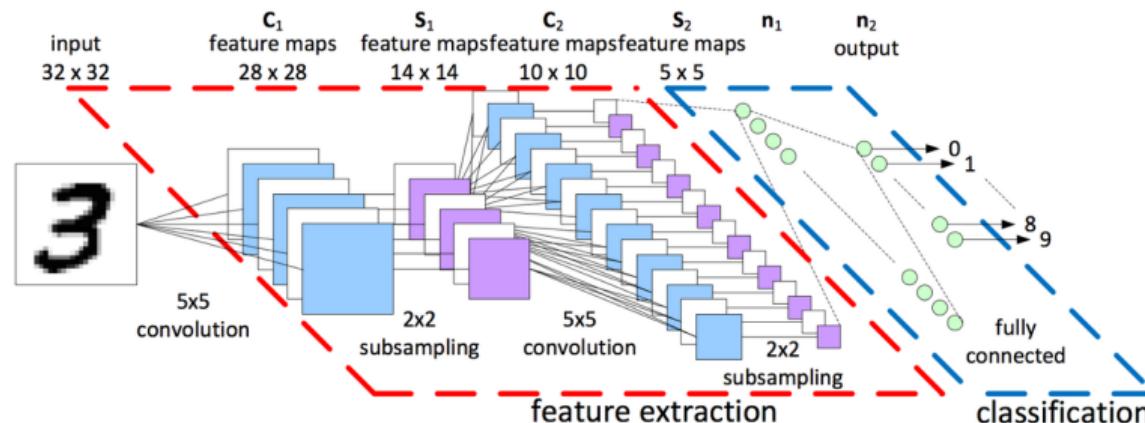
The basic idea of Convolutional Neural Networks (CNNs) is similar to Backpropagation Neural Networks (BPNNs) but differs in implementation.



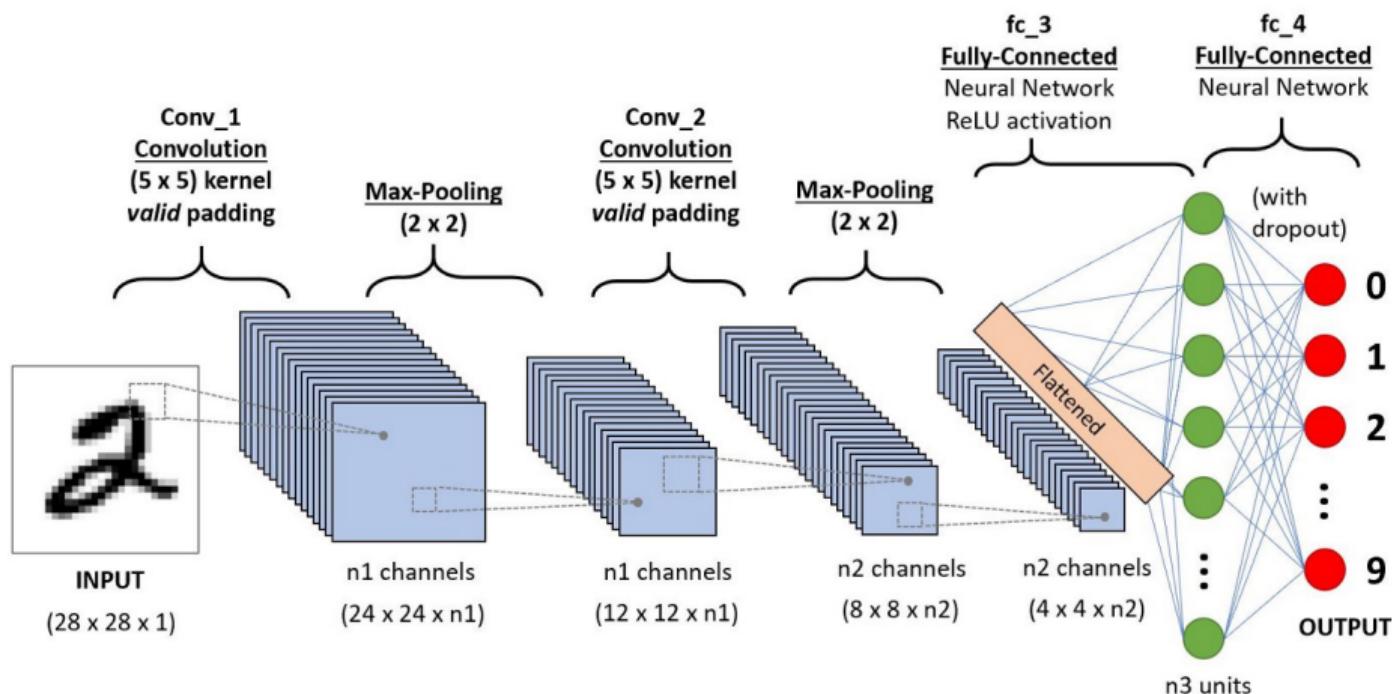
After vectorized (vec), the 2D arranged inputs become ID vectors. Then the network is just like a BPNN (Back propagation neural networks)

The basic structure (1)

- Alternating Convolution (C) and subsampling layer (S)
- Subsampling allows flexible positioning of features.



The basic structure (2)



The basic structure (3)

Three Main Types of Layers

- **Convolutional Layer**
 - Output of neurons are connected to local regions in the input.
 - Applying the same filter across the entire image.
 - CONV layers parameters consist of a set of learnable filters.
- **Pooling Layer**
 - Performs a downsampling operation along the spatial dimensions.
- **Fully-Connected Layer**
 - Typically used in the final stages of the network to combine high-level features and make predictions.

Cnn vs. FCN

- **Fully Connected Networks (FCNs):**

- High number of parameters, leading to overfitting on large inputs (e.g., images).
- Lack of spatial awareness, making it sensitive to the exact positioning of patterns.
- Suitable for structured data but inefficient for image processing.

- **Convolutional Neural Networks (CNN):**

- Uses filters to process only small parts of the input at a time (locality).
- Weight sharing and local connectivity reduce the number of parameters.
- Can recognize patterns independent of their location within the image (shift invariance).
- Efficient for image, video, and speech data.

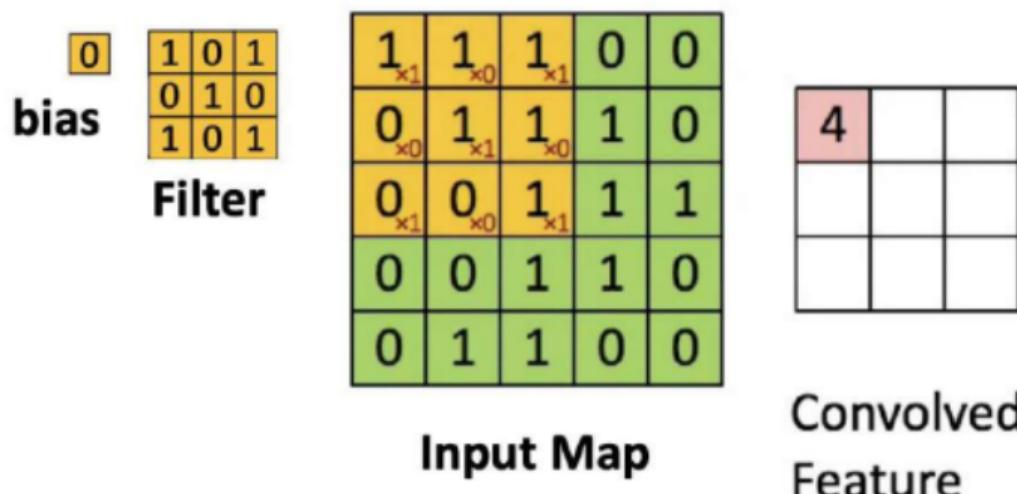
1 Motivation

2 CNN vs. FCN

3 Convolution

4 Pooling

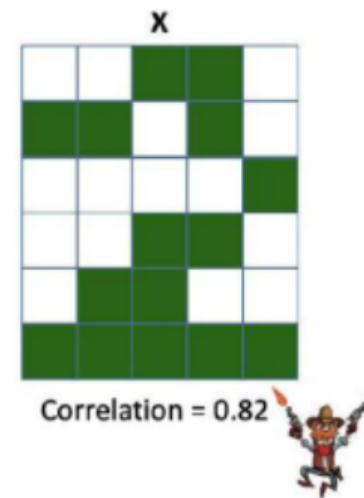
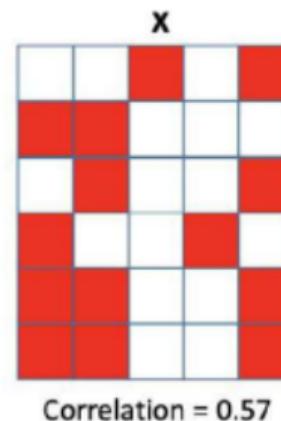
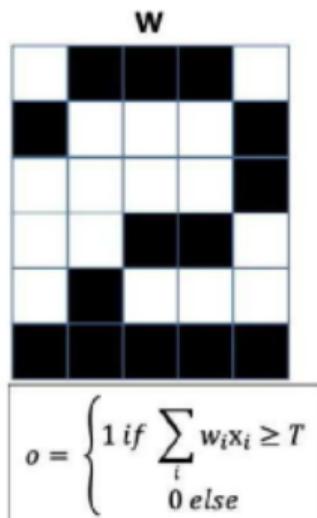
What is a convolve



Scanning an image with a "filter"

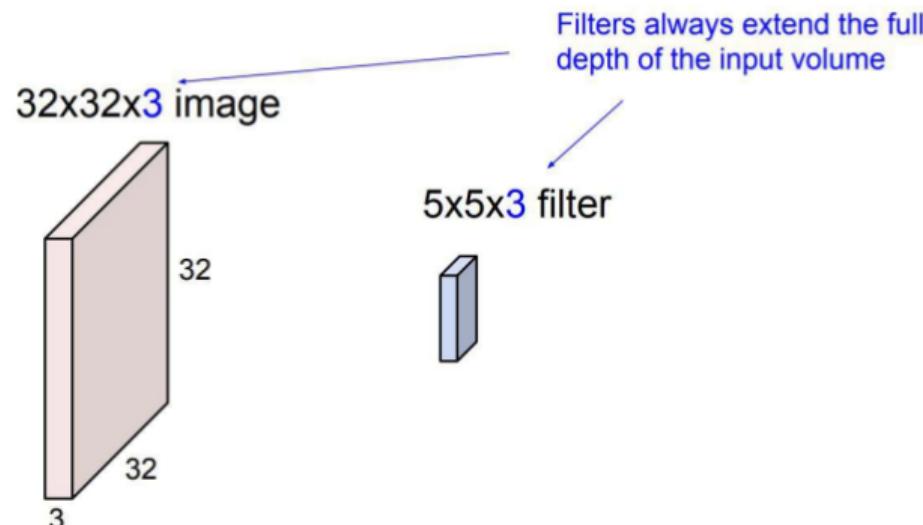
- A filter is really just a perceptron, with weights and a bias.
- At each location, the filter is multiplied component-wise with the underlying map values, and the products are summed along with the bias.

Weights showing correlation



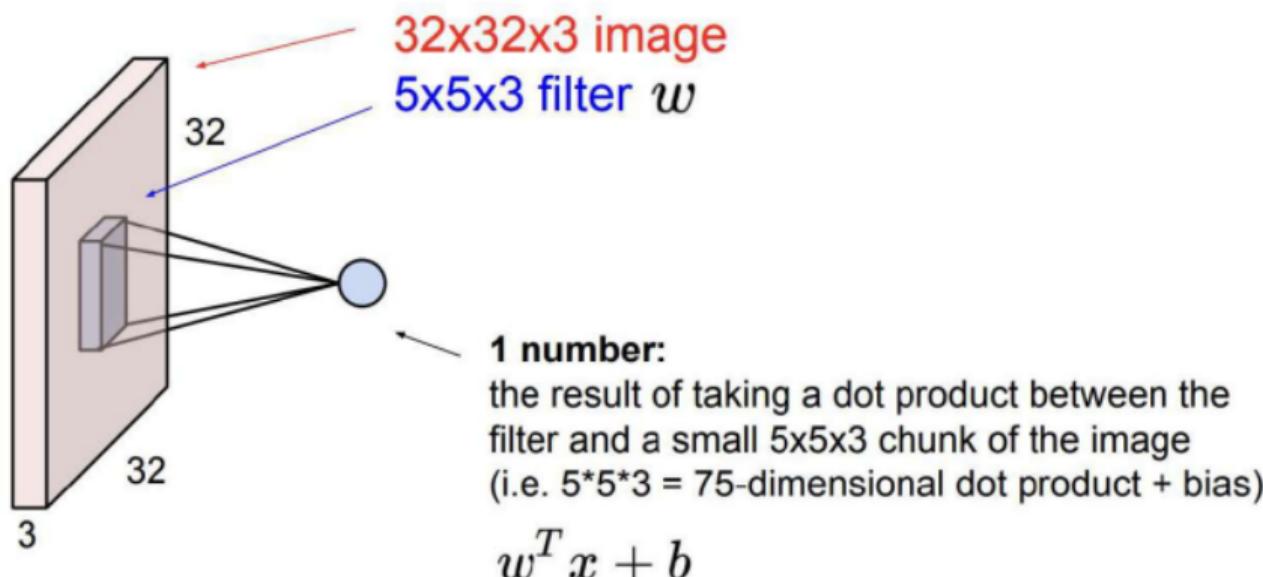
- The weights of the filter represent the appearance of the number "2".
- The **green** has a higher correlation with the filter compared to the **red**.
- The **green pattern** is more likely to represent the number "2".

What is convolution



- **Convolve:** Slide over the image spatially, computing dot products.
- This allows us to preserve the spatial structure of the input.

What is the output



It's simply a neuron with local connectivity!

Convolution process (1)

- Consider this as the filter we are going to use:

1	0	1
0	1	0
1	0	1

Convolution process (2)

- This is how we calculate the convolutional layer's output:

$$\text{Convolved Feature}(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a, b)$$

I: Input Image

K: Our Kernel

k_h and k_w : The height and width of the Kernel

1	0	1
0	1	0
1	0	1

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

Convolution process (3)

- This is how we calculate the convolutional layer's output:

$$\text{Convolved Feature}(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image

K: Our Kernel

k_h and k_w : The height and width of the Kernel

1	0	1
0	1	0
1	0	1

1	1 _{x1}	1 _{x0}	0 _{x3}	0
0	1 _{x0}	1 _{x1}	1 _{x0}	0
0	0 _{x1}	1 _{x0}	1 _{x3}	1
0	0	1	1	0
0	1	1	0	0

Image

4	3	

Convolved Feature

Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

Convolution process (4)

- This is how we calculate the convolutional layer's output:

$$\text{Convolved Feature}(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image

K: Our Kernel

k_h and k_w : The height and width of the Kernel

1	0	1
0	1	0
1	0	1

1	1	1 _{x1}	0 _{x0}	0 _{x1}
0	1	1 _{x0}	1 _{x1}	0 _{x0}
0	0	1 _{x1}	1 _{x0}	1 _{x1}
0	0	1	1	0
0	1	1	0	0

Image

4	3	4

Convolved Feature

Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

Convolution process (5)

- This is how we calculate the convolutional layer's output:

$$\text{Convolved Feature}(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image

K: Our Kernel

k_h and k_w : The height and width of the Kernel

1	0	1
0	1	0
1	0	1

1	1	1	0	0
0 _{x1}	1 _{x0}	1 _{x1}	1	0
0 _{x0}	0 _{x1}	1 _{x0}	1	1
0 _{x1}	0 _{x0}	1 _{x1}	1	0
0	1	1	0	0

Image

4	3	4
2		

Convolved Feature

Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

Convolution process (6)

- This is how we calculate the convolutional layer's output:

$$\text{Convolved Feature}(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image

K: Our Kernel

k_h and k_w : The height and width of the Kernel

1	0	1
0	1	0
1	0	1

1	1	1	0	0
0	1 _{x1}	1 _{x0}	1 _{x3}	0
0	0 _{x0}	1 _{x1}	1 _{x0}	1
0	0 _{x1}	1 _{x0}	1 _{x3}	0
0	1	1	0	0

Image

4	3	4
2	4	

Convolved Feature

Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

Convolution process (7)

- This is how we calculate the convolutional layer's output:

$$\text{Convolved Feature}(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image

K: Our Kernel

k_h and k_w : The height and width of the Kernel

1	0	1
0	1	0
1	0	1

1	1	1	0	0
0	1	1 _{x1}	1 _{x0}	0 _{x1}
0	0	1 _{x0}	1 _{x1}	1 _{x0}
0	0	1 _{x1}	1 _{x0}	0 _{x1}
0	1	1	0	0

Image

4	3	4
2	4	3

Convolved Feature

Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

Convolution process (8)

- This is how we calculate the convolutional layer's output:

$$\text{Convolved Feature}(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image

K: Our Kernel

k_h and k_w : The height and width of the Kernel

1	0	1
0	1	0
1	0	1

1	1	1	0	0
0	1	1	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0 _{x0}	0 _{x1}	1 _{x0}	1	0
0 _{x1}	1 _{x0}	1 _{x1}	0	0

Image

4	3	4
2	4	3
2		

Convolved Feature

Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

Convolution process (9)

- This is how we calculate the convolutional layer's output:

$$\text{Convolved Feature}(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image

K: Our Kernel

k_h and k_w : The height and width of the Kernel

1	0	1
0	1	0
1	0	1

1	1	1	0	0
0	1	1	1	0
0	0 _{x1}	1 _{x0}	1 _{x1}	1
0	0 _{x0}	1 _{x1}	1 _{x0}	0
0	1 _{x1}	1 _{x0}	0 _{x1}	0

Image

4	3	4
2	4	3
2	3	

Convolved Feature

Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

Convolution process (10)

- This is how we calculate the convolutional layer's output:

$$\text{Convolved Feature}(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image

K: Our Kernel

k_h and k_w : The height and width of the Kernel

1	0	1
0	1	0
1	0	1

1	1	1	0	0
0	1	1	1	0
0	0	1 _{x1}	1 _{x0}	1 _{x1}
0	0	1 _{x0}	1 _{x1}	0 _{x0}
0	1	1 _{x1}	0 _{x0}	0 _{x1}

Image

4	3	4
2	4	3
2	3	4

Convolved Feature

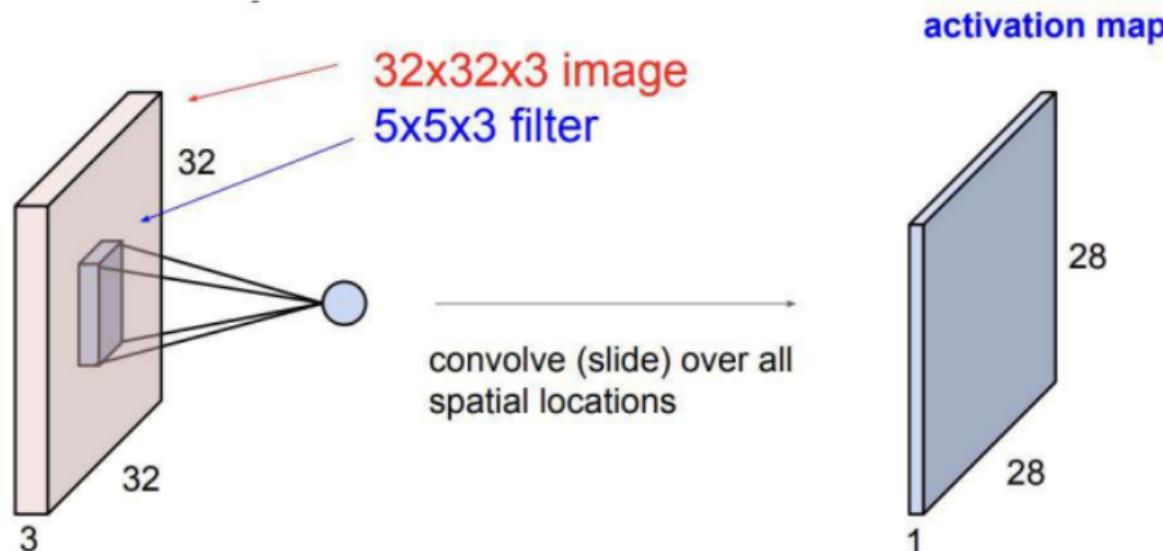
Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

To summarize

- We apply the same filter on different regions of the input.
- Convolutional filters learn to make decisions based on local spatial input, which is an important attribute when working with images.
- Uses a lot fewer parameters compared to Fully Connected Layers.

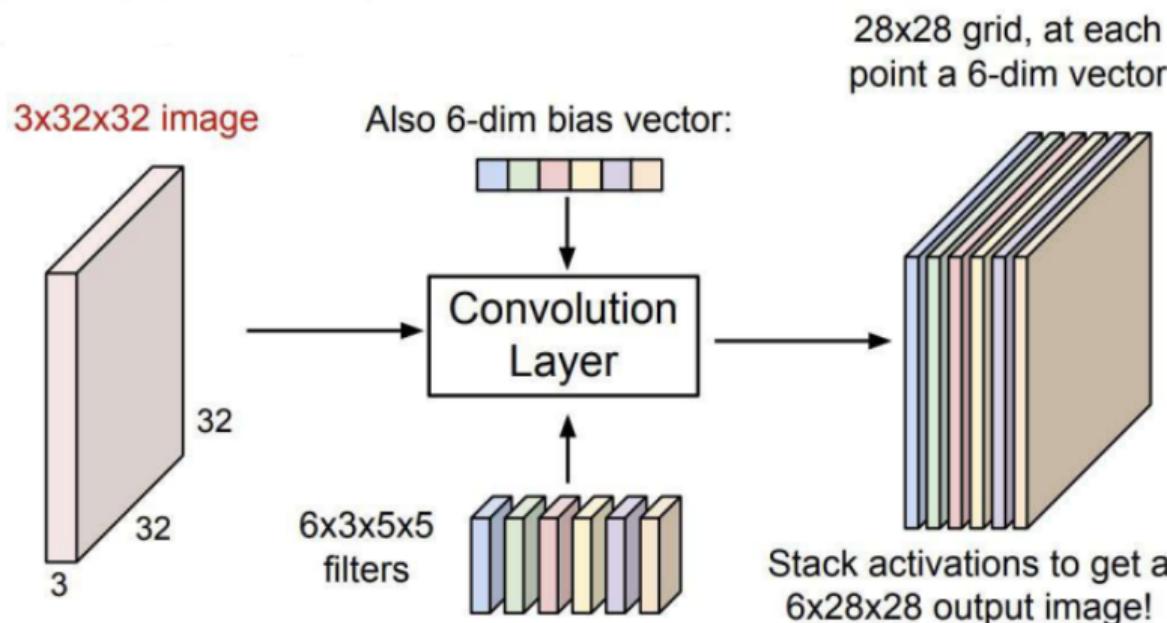


Convolution outlook (1)



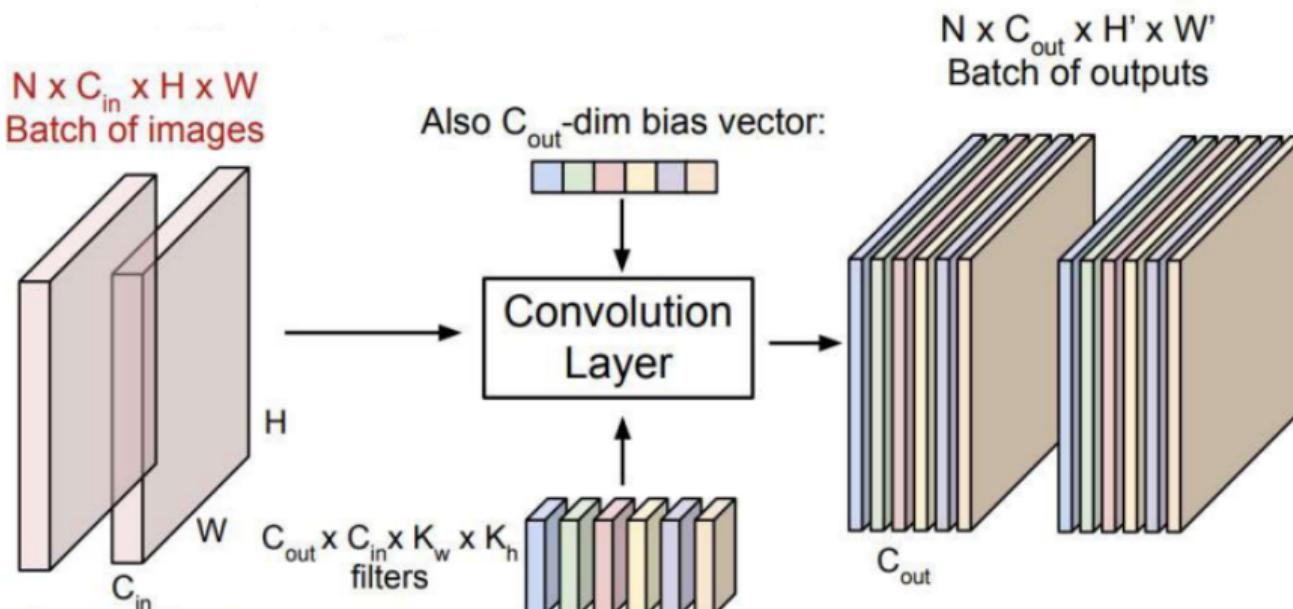
If we consider the **image** to be of size $n \times m \times b$ and the **filter** to be of size $n' \times m' \times b$, we will have an **activation map** of size $(n - n' + 1) \times (m - m' + 1) \times 1$ for each filter.

Convolution outlook (2)



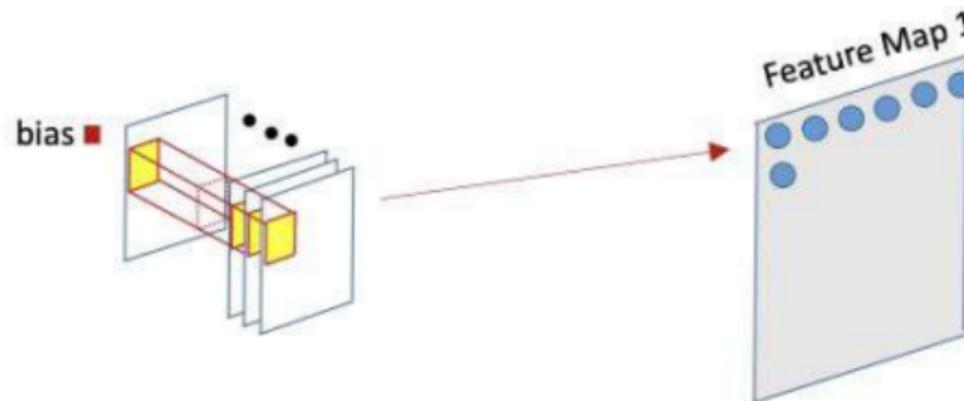
Multiple filters can be applied, each with its own bias, to extract different features from the input.

Convolution outlook (3)



Batching input images can provide a more generalized representation, as described above.

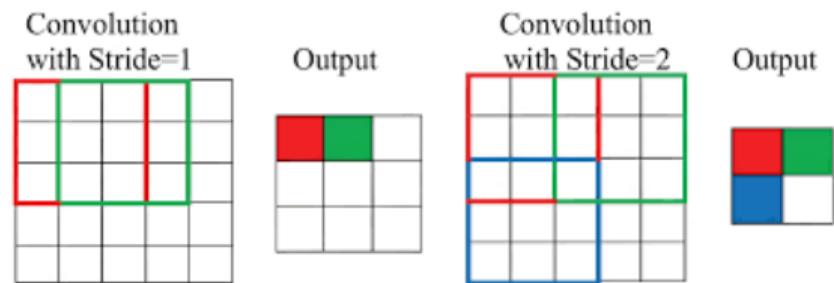
A different notation



$$z(i, j, s) = \underbrace{\sum_p \sum_{k=1}^L \sum_{l=1}^L w(k, l, p, s) a(i + l - 1, j + k - 1, p)}_{\text{One map}} + b(s)$$

$\underbrace{\qquad\qquad\qquad}_{\text{All maps}}$

What is stride

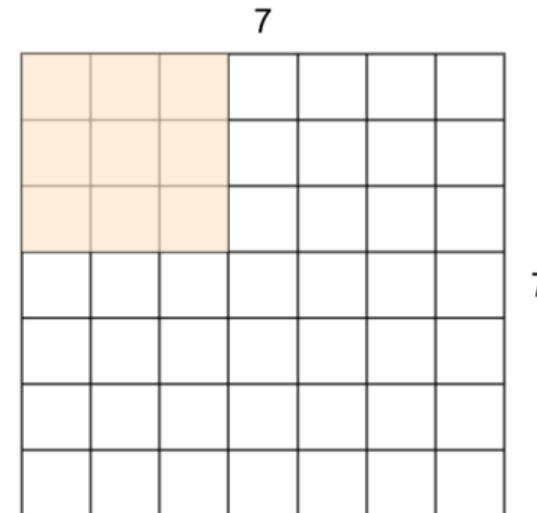


- The scans of the individual “filters” may advance by more than one pixel at a time
 - The “stride” may be greater than 1
 - Effectively increasing the granularity of the scan
 - Saves computation, sometimes at the risk of losing information
- This results in a reduction in the size of the resulting maps
 - They will shrink by a factor equal to the stride
- This can happen at any layer

Stride-1 (1)

Closer look

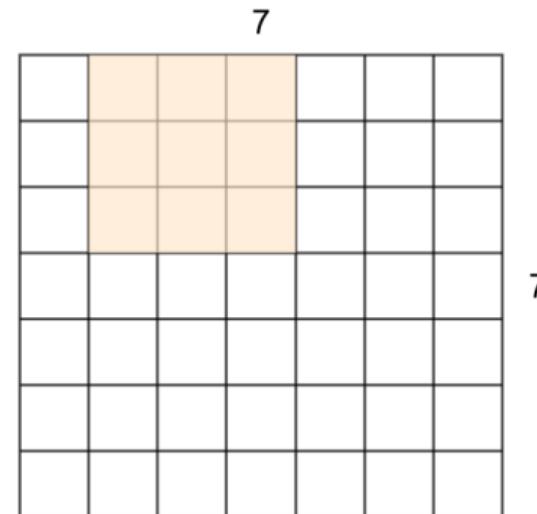
- 7x7 input with 3x3 filter



Stride-1 (2)

Closer look

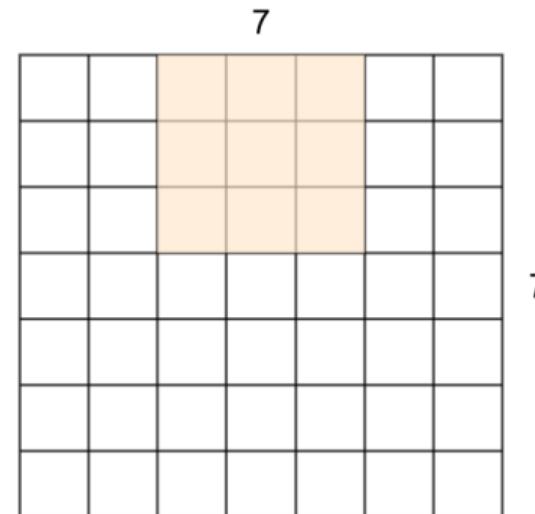
- 7x7 input with 3x3 filter



Stride-1 (3)

Closer look

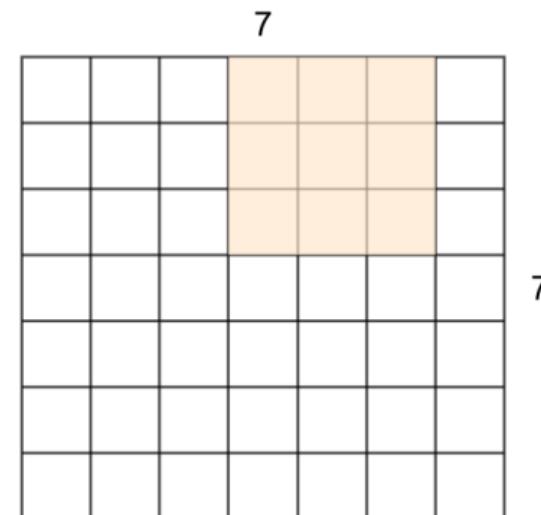
- 7x7 input with 3x3 filter



Stride-1 (4)

Closer look

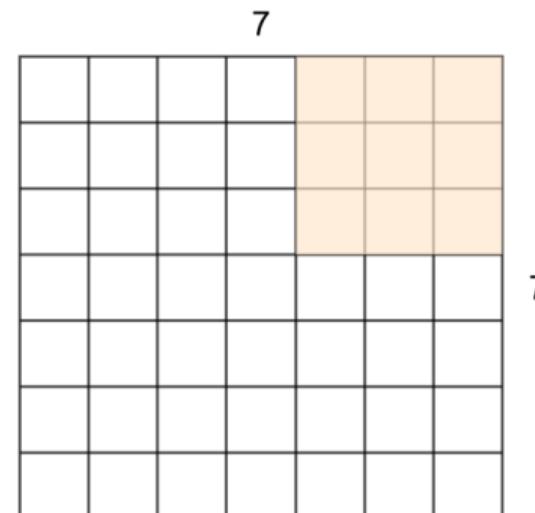
- 7x7 input with 3x3 filter



Stride-1 (5)

Closer look

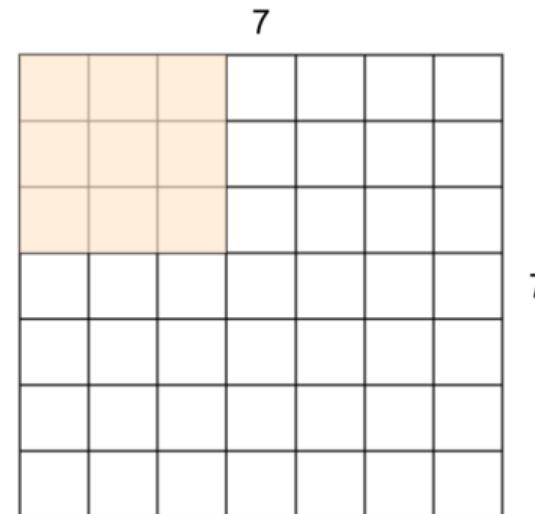
- 7x7 input with 3x3 filter
- This was a Stride 1 filter
- => Outputs 5x5



Strides-2 (1)

Now let's use Stride 2

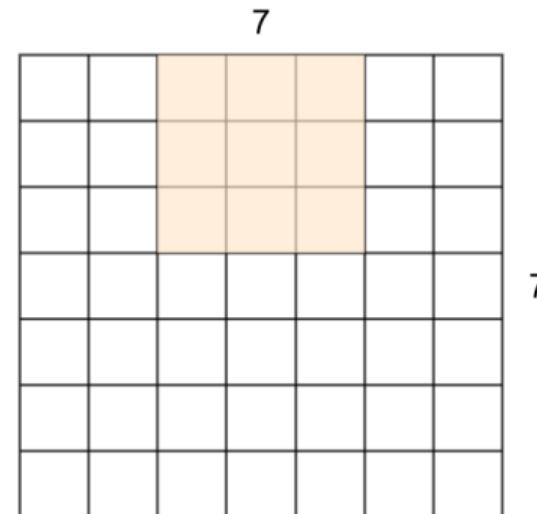
- 7x7 input with 3x3 filter



Strides-2 (2)

Now let's use Stride 2

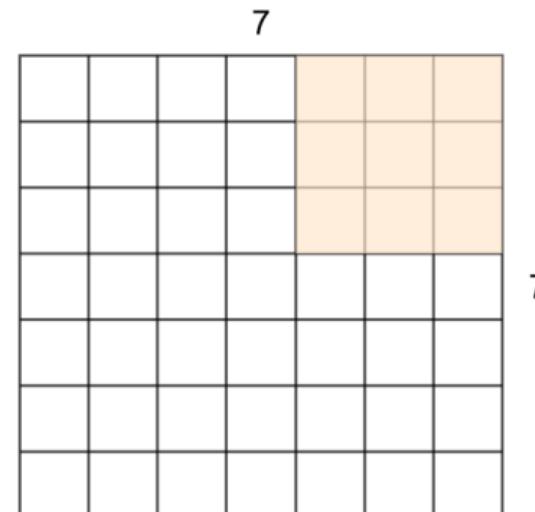
- 7x7 input with 3x3 filter



Strides-2 (3)

Now let's use Stride 2

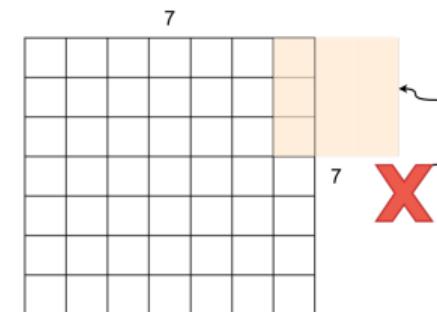
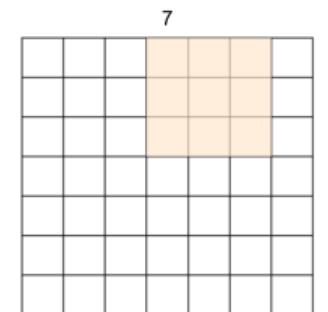
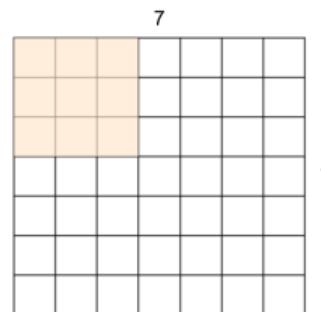
- 7x7 input with 3x3 filter
- This was a Stride 2 filter
- => Outputs 3x3



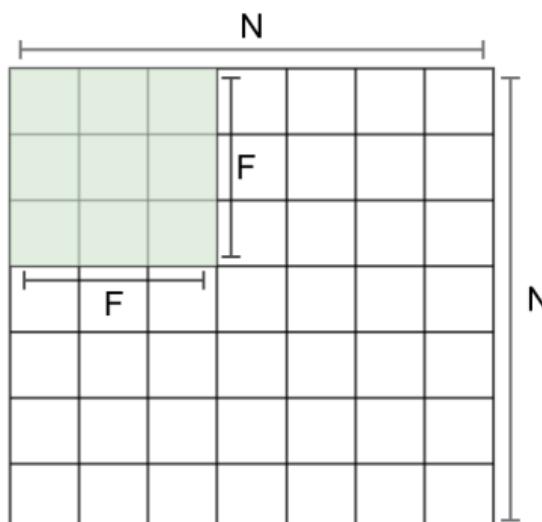
Strides-3

Stride 3?

- 7x7 input with 3x3 filter



Strides Formula



$$\text{OutputSize} = \frac{(N-F)}{\text{Stride}} + 1$$

$$N = 7, F = 3 \Rightarrow$$

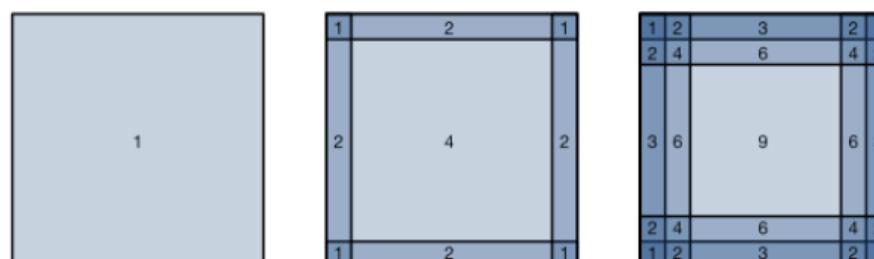
- Stride 1: $(7 - 3)/1 + 1 = 5$
- Stride 2: $(7 - 3)/2 + 1 = 3$
- Stride 3: $(7 - 3)/3 + 1 = 2.33 !!!$

A problem?

What could cause us problems?

Problem (1)

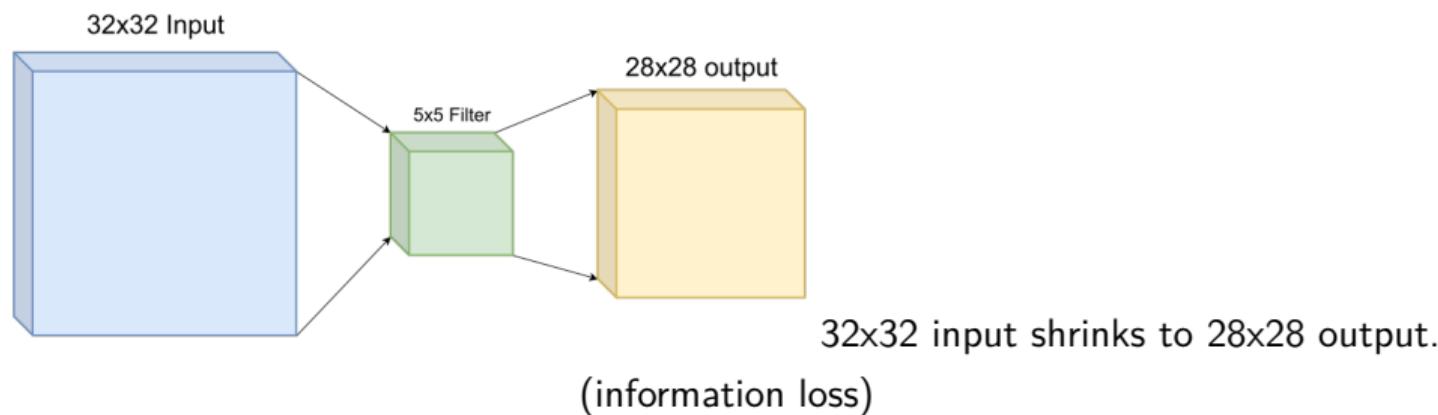
- 1. Borders don't get enough attention.
- 2. Outputs shrink!



Pixel utilization for convolutions of size 1×1 , 2×2 , and 3×3 respectively.

Problem (2)

- 1. Borders don't get enough attention.
- 2. Outputs shrink!



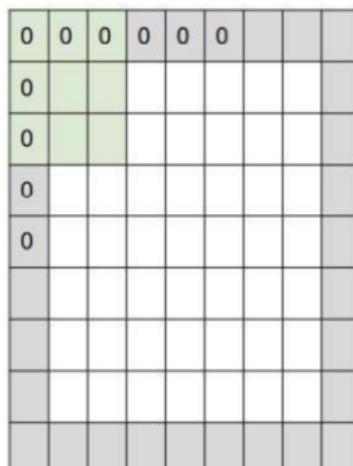
Solution?

What is the solution?

Padding

Padding: the secret ingredient to keep strides in line!

Padding (1)



In practice, it is common to zero-pad the border.

Recall: The original formula for convolution without padding:

$$\frac{N - F}{\text{stride}} + 1$$

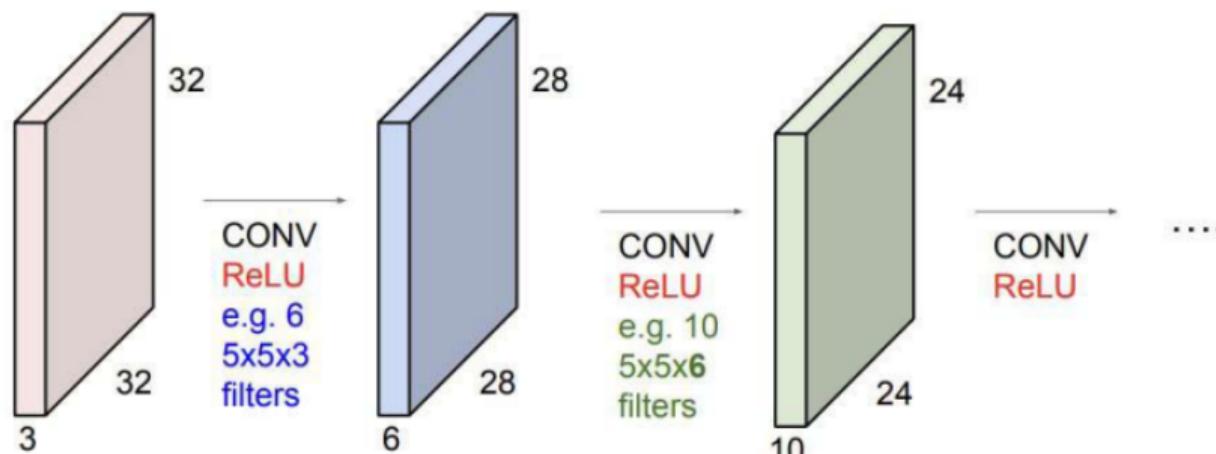
Now, we should adjust the formula considering padding P :

$$\frac{N + 2P - F}{\text{stride}} + 1$$

Padding (2)

- Zero-padding is used not only for stride > 1 , but also to prevent a reduction in output size even when $S = 1$.
- For stride > 1 , zero padding is adjusted to ensure that the size of the convolved output is $\left\lceil \frac{N}{S} \right\rceil$
This is achieved by zero padding the image with $P = S \left\lceil \frac{N}{S} \right\rceil - N$
- For an F width filter:
 - **Odd** F : Pad on both left and right with $\frac{F-1}{2}$ columns of zeros
 - **Even** F : Pad one side with $\frac{F}{2}$ columns of zeros, and the other with $\frac{F}{2}-1$ columns of zeros
 - The resulting image is width $N+F-1$
- The top/bottom zero padding follows the same rules to maintain map height after convolution

Convnet (1)



- **ConvNet:** sequence of convolution layers, **interspersed with activation functions.**

Convnet (2)

Question: What do convolutional filters at different levels of a ConvNet learn?

Answer:

- Filters in the early layers typically detect simple features, such as edges, textures, and basic shapes.
- As we move deeper into the network, the filters learn more complex and abstract features, such as specific parts of objects (e.g., a nose, eyes, or other high-level patterns).

Example-1

Input volume: 32x32x3

10 5x5 filters with stride 1, pad 2

Output volume size:

$$\left(\frac{32 + 2 \times 2 - 5}{1} \right) + 1 = 32 \text{ spatially, so } 32 \times 32 \times 10$$

Example-2

Input volume: 32x32x3

10 5x5 filters with stride 1, pad 2

How many parameters are in this layer?

Each filter has $5*5*3 + 1 = 76$ params (*+1 for bias*)
 $\Rightarrow 76*10 = 760$

Parameter setting (1)

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyper-parameters:
 - Number of filters K
 - Their spatial extent F
 - The stride S
 - The amount of zero padding P
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = \left(\frac{W_1 - F + 2P}{S}\right) + 1$
 - $H_2 = \left(\frac{H_1 - F + 2P}{S}\right) + 1$
 - $D_2 = K$

Common settings:

- $K = \text{powers of 2}$ (e.g., 32, 64, ...)
- $F = 3, S = 1, P = 1$
- $F = 5, S = 1, P = 2$
- $F = 5, S = 2, P = \text{whatever fits!}$
- $F = 1, S = 1, P = 0$

Parameter setting (2)

- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d -th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d -th filter over the input volume with a stride of S , and then offset by the d -th bias.

1 Motivation

2 CNN vs. FCN

3 Convolution

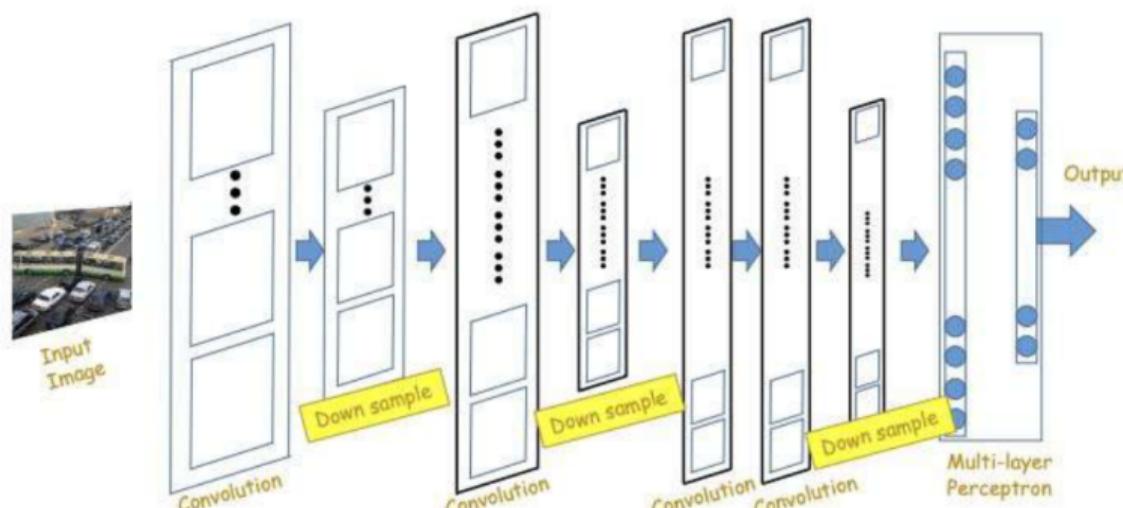
4 Pooling

Review

Three Main Types of Layers

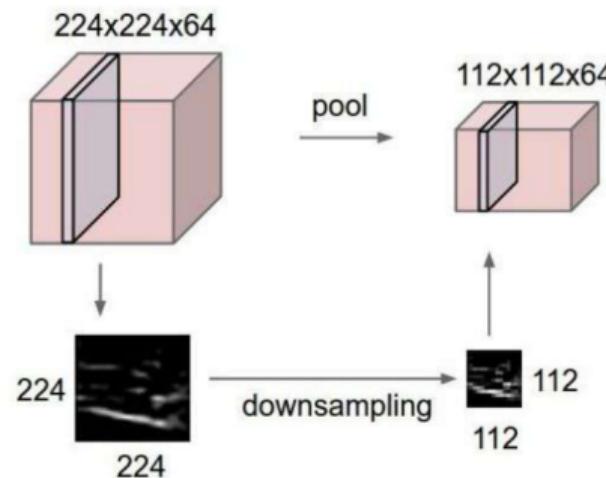
- **Convolutional Layer**
 - Output of neurons are connected to local regions in the input.
 - Applying the same filter across the entire image.
 - CONV layers parameters consist of a set of learnable filters.
- **Pooling Layer**
 - Performs a downsampling operation along the spatial dimensions.
- **Fully-Connected Layer**
 - Typically used in the final stages of the network to combine high-level features and make predictions.

Pooling (1)



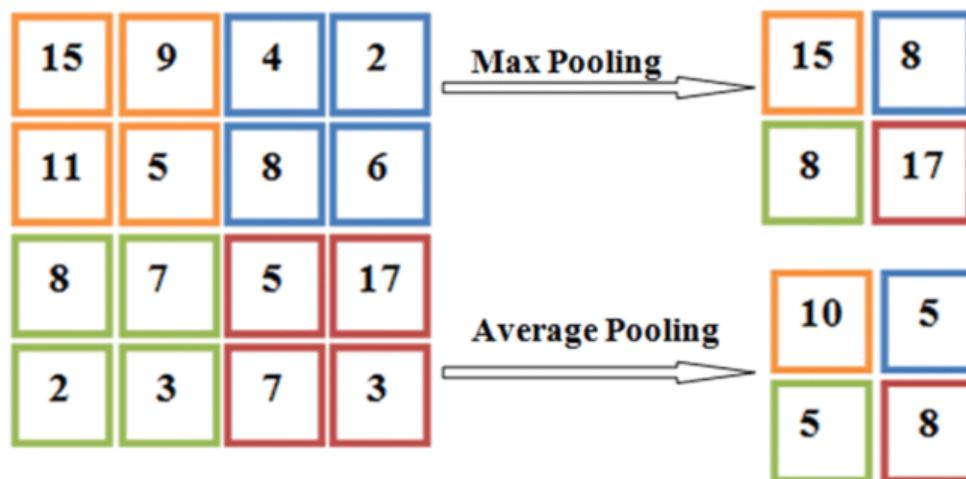
- Convolution & activation layers are followed intermittently by pooling layers.
 - Often, they alternate with convolution, though this is not necessary.

Pooling (2)



- Reduce the spatial size of the representation.
 - To reduce the number of parameters and computational demands within the network.
 - To reduce variance.
- Helps the network become invariant to small translations or distortions.
- Operates over each activation map independently.

Pooling type (1)



Two Primary Types of Pooling:

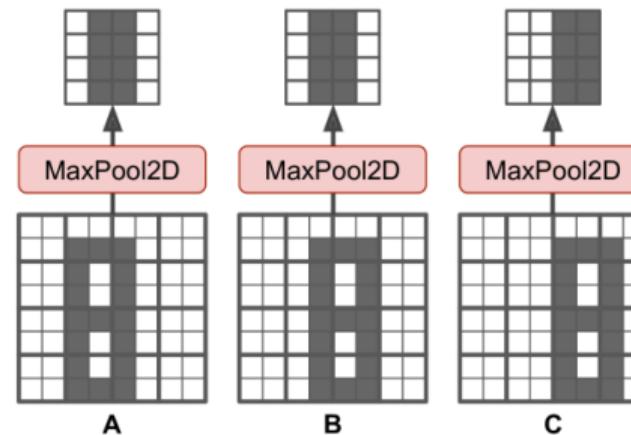
- **Max Pooling:** Selects the maximum value from each patch of the feature map.
- **Min Pooling:** Selects the minimum value from each patch of the feature map.
- **Average Pooling:** Computes the average of the values in each patch of the feature map.

Pooling type (2)

- Max Pooling: performs well when the background is light, and the object is dark
- Min Pooling: performs well when the background is dark, and the object is light
- Average Pooling: smooths the image



Pooling advantage



- Most common type of pooling layer
- Invariance to small translations

Parameter setting (1)

Question:

How does a pooling layer affect the dimensions of the input image?

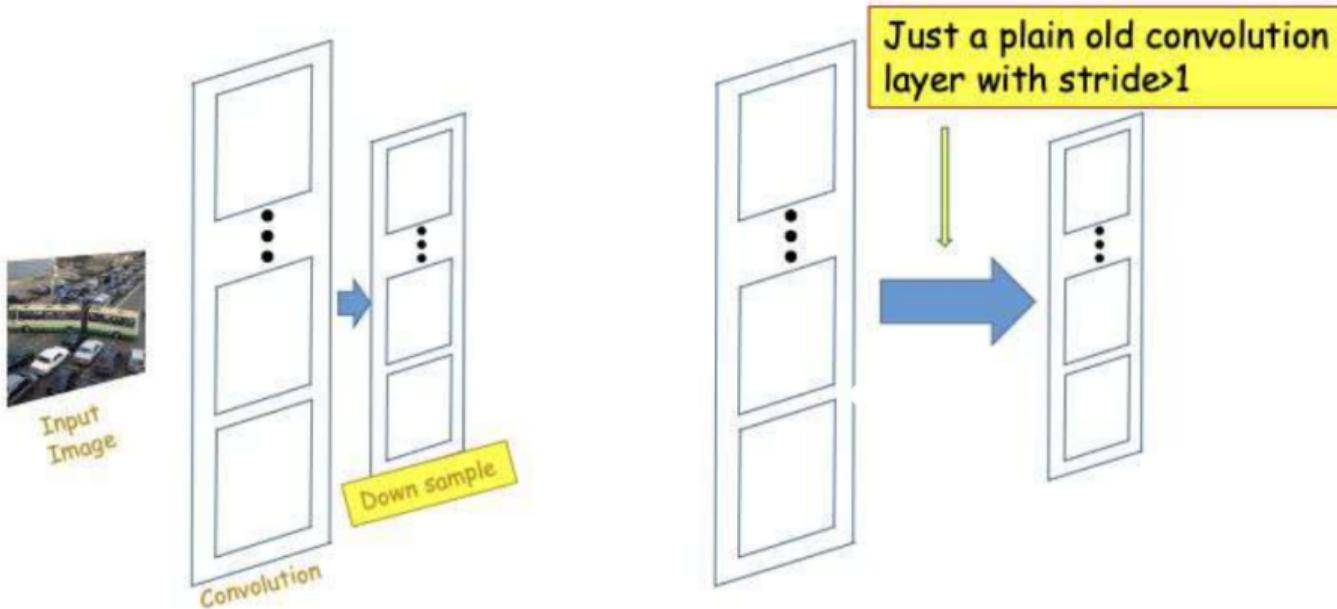
Answer:

- An $N \times N$ picture compressed by a $P \times P$ pooling filter with stride S results in an output map of side $\left\lceil \frac{(N-P)}{S} \right\rceil + 1$.
- Typically, pooling layers do not use zero-padding.

Parameter setting (2)

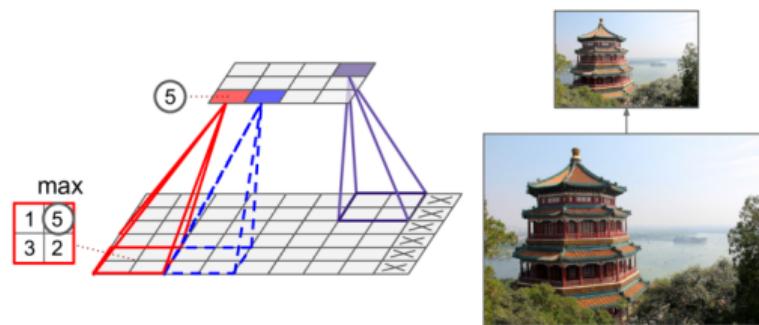
- Pooling accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
 - Their spatial extent F ,
 - The stride S .
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = \frac{(W_1 - F)}{S} + 1$
 - $H_2 = \frac{(H_1 - F)}{S} + 1$
 - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input.
- It is uncommon to use zero-padding for pooling layers.

Alternative to pooling



Downsampling can be done by a simple convolution layer with stride larger than 1,
Replacing the max pooling layer with a convolution layer.

Pooling Summary



Max pooling layer (2×2 pooling kernel, stride 2, no padding)

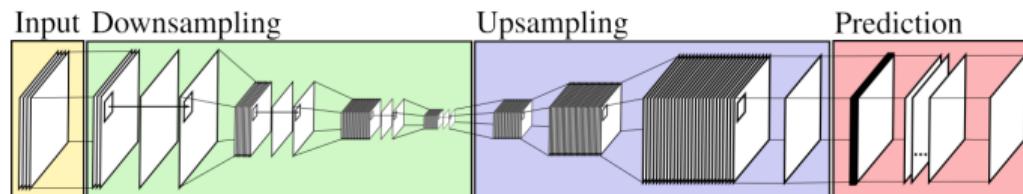
- Goal is to sub-sample the input to reduce:
 - Computational load
 - Memory usage
 - Number of parameters
 - Risk of overfitting

Question

- What if we want to increase the size?

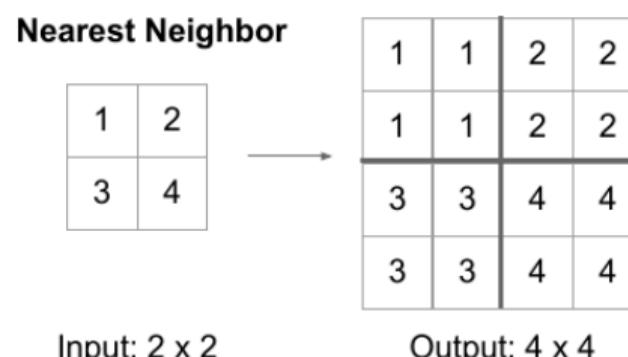
Up-sampling CNN

- Resizing feature maps is a common operation in many neural networks, especially those that perform some kind of image segmentation task.
- This kind of architecture is famously known as the Encoder-Decoder network.



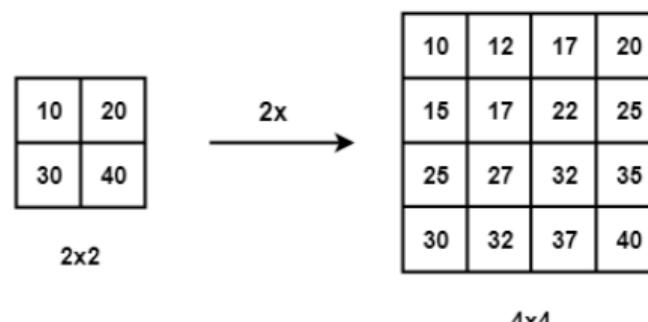
Nearest Neighbors

- Nearest Neighbors: In Nearest Neighbors, as the name suggests, we take an input pixel value and copy it to the K-Nearest Neighbors, where K depends on the expected output.



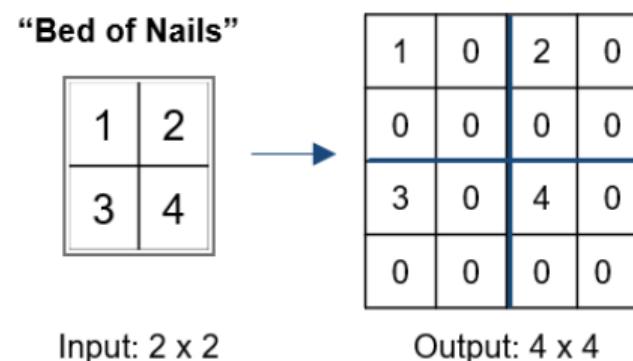
Bi-Linear Interpolation

- Bi-Linear Interpolation: In Bi-Linear Interpolation, we take the 4 nearest pixel values of the input pixel and perform a weighted average based on the distance of the four nearest cells, smoothing the output.



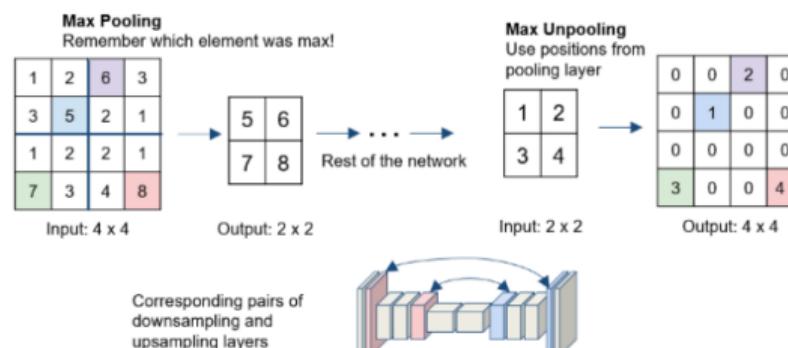
Bed Of Nails

- Bed Of Nails: In Bed of Nails, we copy the value of the input pixel at the corresponding position in the output image and fill zeros in the remaining positions.



Max-Unpooling

- Max-Unpooling: To perform max-unpooling, first, the index of the maximum value is saved for every max-pooling layer during the encoding step. The saved index is then used during the decoding step where the input pixel is mapped to the saved index, filling zeros everywhere else.



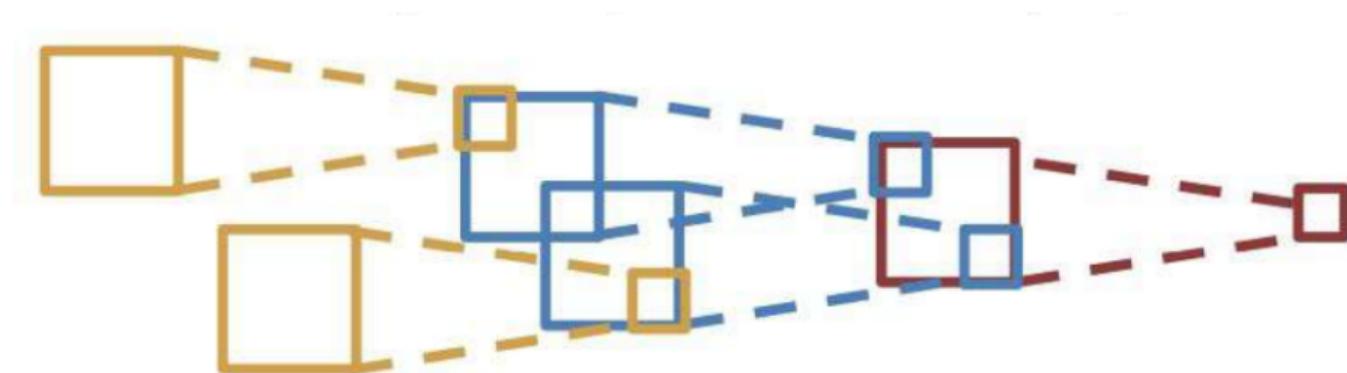
1 Motivation

2 CNN vs. FCN

3 Convolution

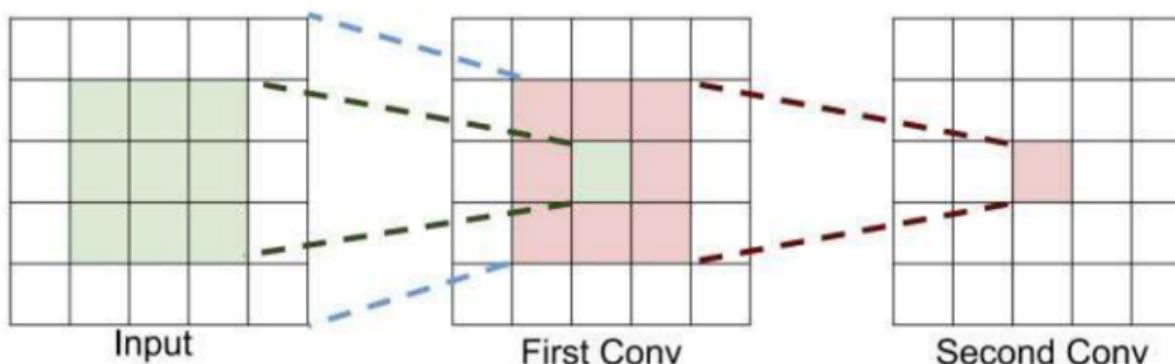
4 Pooling

Receptive field (1)



- **Receptive Field** : How big of a region in the **input or previous** layer does a neuron on the n-th conv-layer see?
- For convolution with kernel size K , each element in the next layer depends on a $K \times K$ **receptive field** in the previous layer.

Receptive field (2)



- Units in the deeper layers can be **indirectly** connected to most of the input image.
- Each successive convolution adds $K - 1$ to the receptive field size. With L layers, the receptive field size is $1 + L \cdot (K - 1)$.
- **Problem:** For large images, we need many layers for each output to "see" the whole image.
 - **Solution:** Downsample inside the network using strides and pooling.

Power of small filters

Suppose the input is $H \times W \times C$ and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W).

one CONV with 7×7 filters

Number of weights

$$= C \times (7 \times 7 \times C) = \mathbf{49}C^2$$

three CONV with 3×3 filters

Number of weights

$$= 3 \times C \times (3 \times 3 \times C) = \mathbf{27}C^2$$

Both options achieve a receptive field of 7; however, using multiple smaller filters reduces the number of parameters, introduces more nonlinearity, and generally leads to a more efficient, expressive model.

Question

Question: In a convolutional neural network, each layer increases the receptive field size. Suppose a network has 3 convolutional layers, each with a kernel size of $K = 3$ and a stride of $S = 1$.

- Calculate the receptive field size after each layer, starting from an initial receptive field size of 1.
- How large is the receptive field after the third layer?
- Why is the growing receptive field important in deeper layers?

Answer

Answer:

- The receptive field size increases by $K - 1$ with each layer.
 - After the 1st layer: $1 + (3 - 1) = 3$
 - After the 2nd layer: $3 + (3 - 1) = 5$
 - After the 3rd layer: $5 + (3 - 1) = 7$
- Thus, the receptive field after the third layer is 7.
- A larger receptive field allows neurons in deeper layers to capture more context from the input image, which is essential for understanding higher-level patterns.

Inductive bias in CNNs

Inductive Bias:

Refers to the assumptions a model incorporates to generalize from training data to unseen data.

Key Features of Inductive Bias in CNNs:

- **Weight Sharing:**

A single filter is applied across different regions of the input, significantly reducing the number of parameters.

- **Locality:**

CNNs use small filters (e.g., 3×3) that focus on local regions, aligning well with image data where local structures are important.

- CNNs are more sample-efficient than FCNs due to their inductive biases.

- CNNs reduce sample complexity from $O(d^2)$ in FCNs to $O(\log^2 d)$.

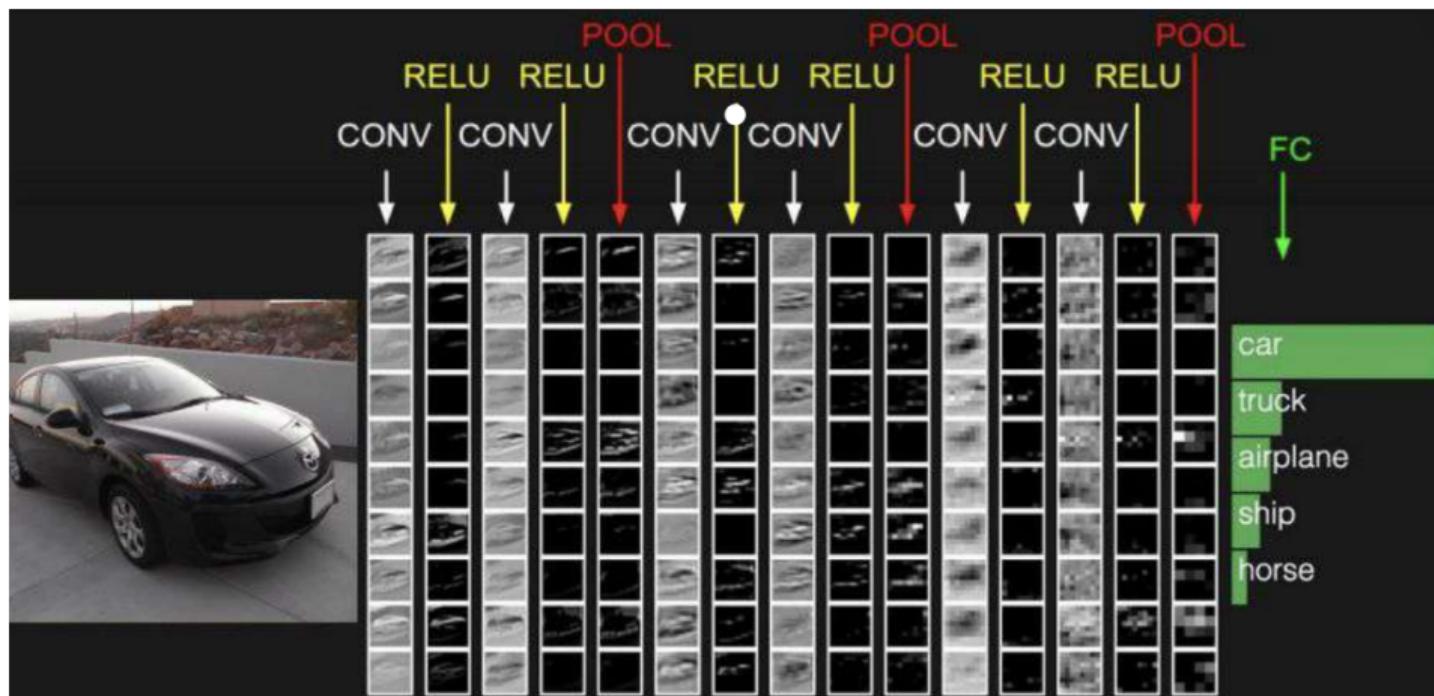
1 Motivation

2 CNN vs. FCN

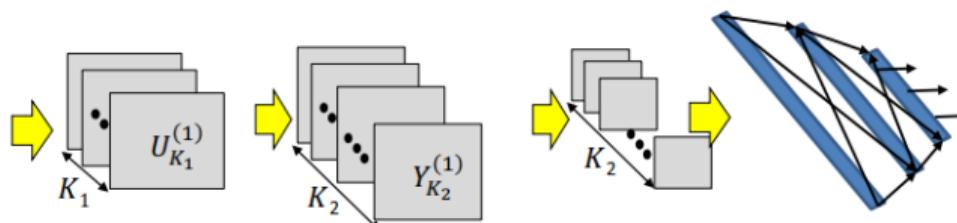
3 Convolution

4 Pooling

Summary of concepts

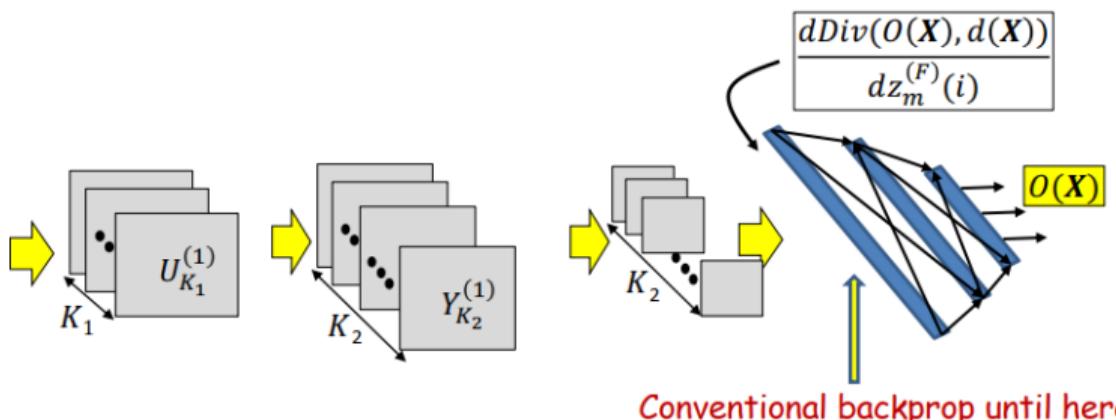


Training (1)



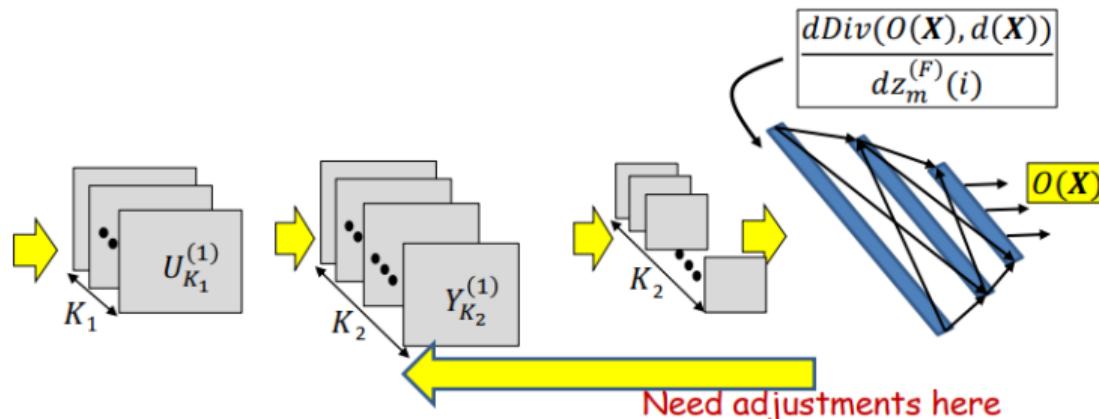
- Training is as in the case of the regular MLP
- Training examples of (Image, class) are provided
- The difference between the desired and actual network output for any input
- **Network parameters are optimized using gradient descent variants**
- **Gradients are computed through backpropagation**

Training (2)



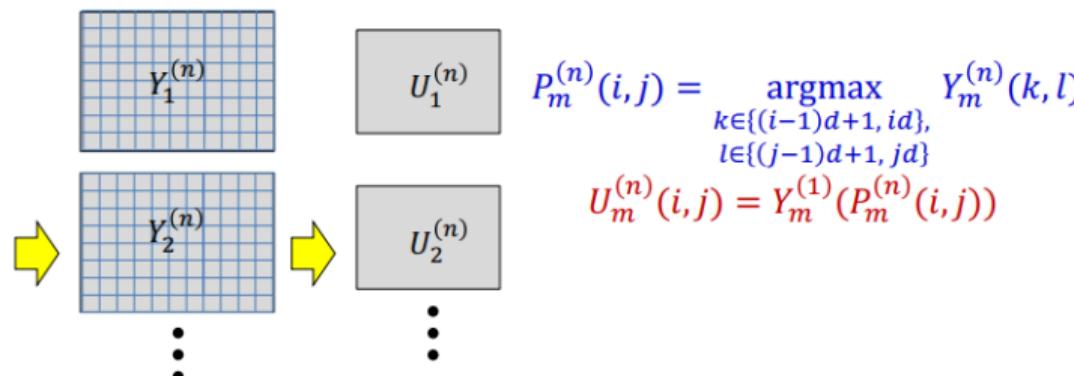
- Backpropagation continues in the usual manner until the computation of the derivative of the divergence w.r.t the inputs to the first “flat” layer

Training (3)



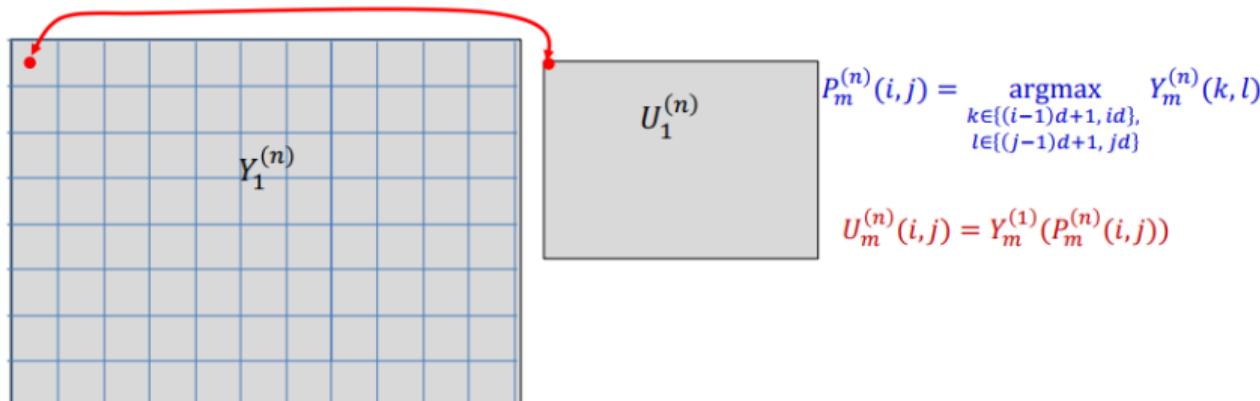
- Backpropagation from the flat MLP requires special consideration of:
 - The pooling layers, particularly Maxout
 - The shared computation in the convolution layers

Backpropagation maxout (1)



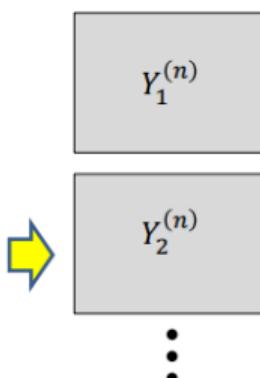
- The derivative w.r.t $U_m^{(n)}(i,j)$ can be computed via backprop.
- But this cannot be propagated backwards to compute the derivative w.r.t. $Y_m^{(n)}(k,l)$.
- **Max and argmax are not differentiable.**

Backpropagation maxout (2)



- $$\frac{d\text{Div}(O(X), d(X))}{dY_m^{(n)}(k, l)} = \begin{cases} \frac{d\text{Div}(O(X), d(X))}{dU_m^{(n)}(i, j)} & \text{if } (k, l) = P_m^{(n)}(i, j) \\ 0 & \text{otherwise} \end{cases}$$
- Approximation:** Derivative w.r.t the $\textcolor{blue}{Y}$ terms that did not contribute to the maxout map is 0.

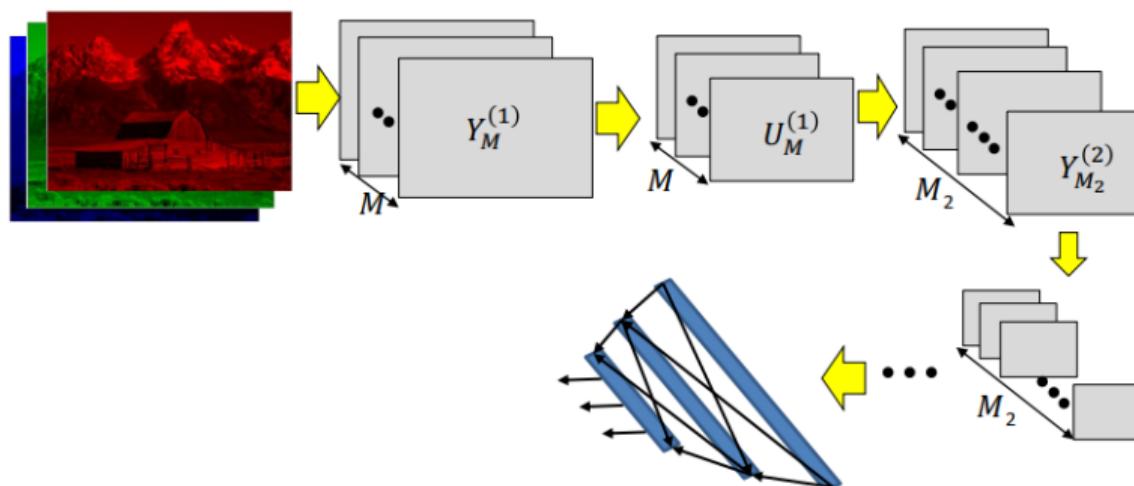
Backpropagation weights



$$z_m^{(n)}(i, j) = \sum_{r=1}^{M_{n-1}} \sum_{k=1}^{L_n} \sum_{l=1}^{L_n} w_m^{(n)}(r, k, l) U_r^{(n-1)}(i + k, j + l)$$
$$Y_m^{(n)}(i, j) = f(z_m^{(n)}(i, j))$$

- **Note:** each weight contributes to *every* position in the map at the output of the convolutional layer.
- **Every position will contribute to the derivative of the weight**
 - Updating shared parameters

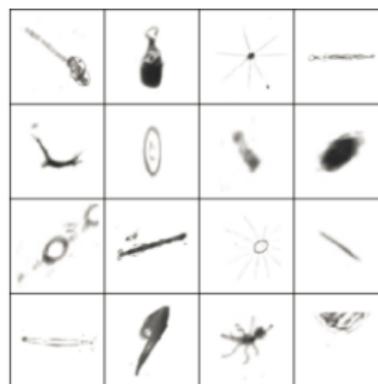
Learning the network



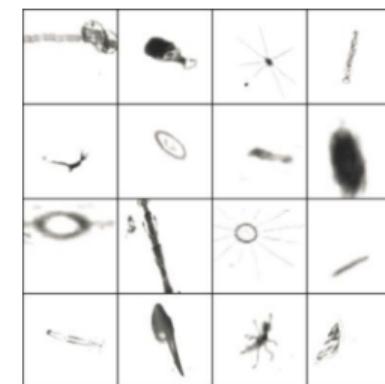
- Derived the divergence concerning every intermediate output and free parameter (filter weights).
- Can now be embedded in the gradient descent framework to learn the network.

A problem

Original data



Augmented data



Question: How can generalization be improved? **Answer:** Rotation, Translation, Rescaling, Flipping, etc.

Design choices

- **Number of convolutional and downsampling layers**
 - Arrangement (order in which they follow each other)
- **For each convolutional layer:**
 - Number of filters d^l
 - Spatial extent of filter $F^l \times F^l$
 - The "depth" of the filter is fixed by the number of filters in the previous layer d^{l-1}
 - The stride S^l
- **For each downsampling or pooling layer:**
 - Spatial extent of filter $P^l \times P^l$
 - The stride S^l
- **For the final multi-layer perceptron (MLP):**
 - Number of layers, and number of neurons in each layer

Typical Architecture for CNNs

N POOL? *M (FC-RELU)*K SOFTMAX

- Where N is usually up to ~ 5
- M is a large number
- $0 \leq K \leq 2$

But recent advances, such as ResNet and GoogleNet, challenge this paradigm!

Conclusion

- Highly structured nature of image data
- Local correlations are important
- CNNs have local and shared connections
- CNNs introduce strong inductive biases
- To leverage the two-dimensional structure of image data and introduce inductive biases, four interrelated concepts can be used:
 - Hierarchy
 - Locality
 - Invariance
 - Sharing

1 Motivation

2 CNN vs. FCN

③ Convolution

4 Pooling

content...