

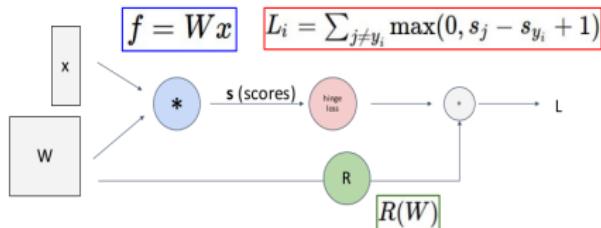
Convolutional Neural Networks

Ranjeet Ranjan Jha

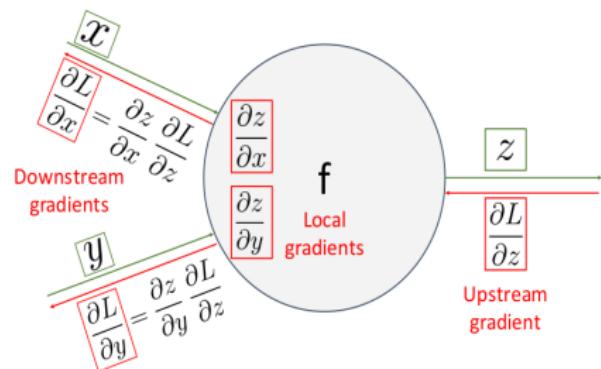
Mathematics Department
Indian Institute of Technology Patna

Last Time: Backpropagation

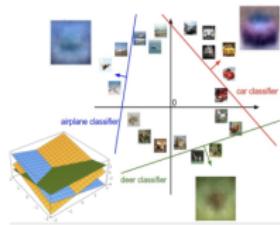
Represent complex expressions as **computational graphs**



During the backward pass, each node in the graph receives **upstream gradients** and multiplies them by **local gradients** to compute **downstream gradients**

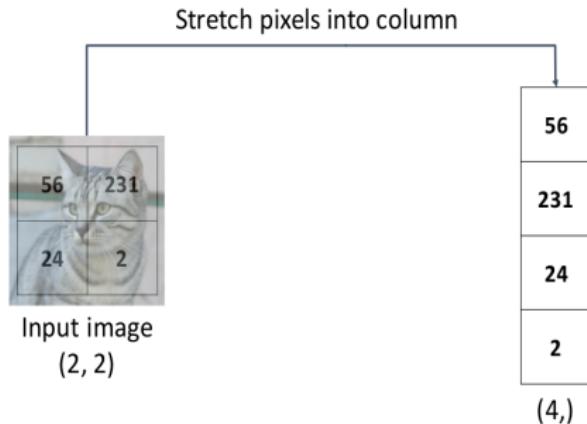
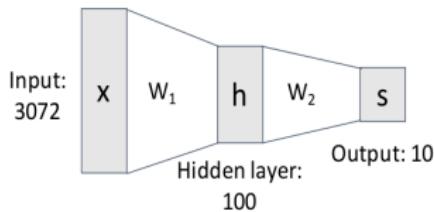


$$f(x, W) = Wx$$

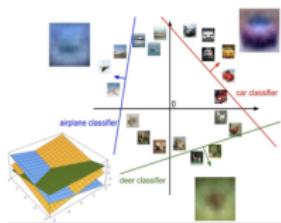


Problem: So far our classifiers don't respect the spatial structure of images!

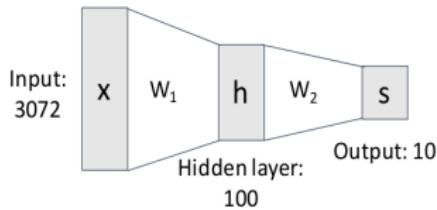
$$f = W_2 \max(0, W_1 x)$$



$$f(x, W) = Wx$$

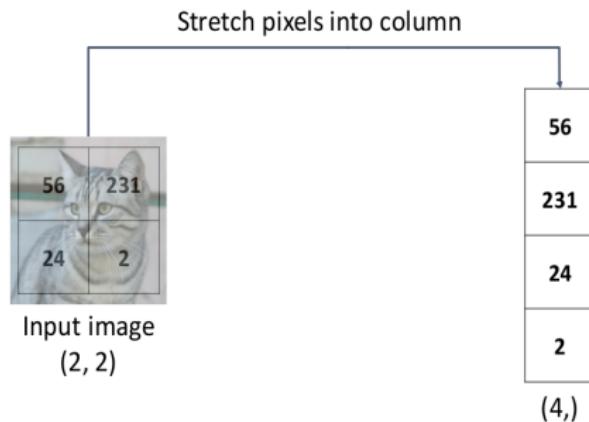


$$f = W_2 \max(0, W_1 x)$$



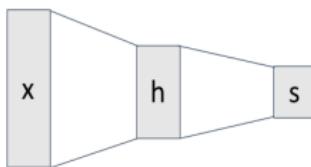
Problem: So far our classifiers don't respect the spatial structure of images!

Solution: Define new computational nodes that operate on images!

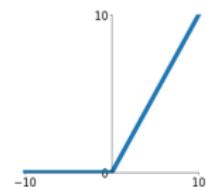


Components of a Fully-Connected Network

Fully-Connected Layers

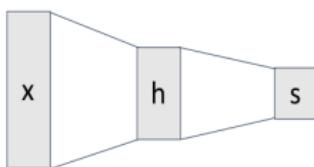


Activation Function

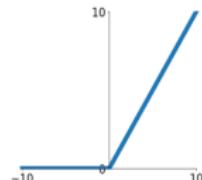


Components of a Convolutional Network

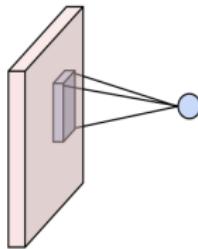
Fully-Connected Layers



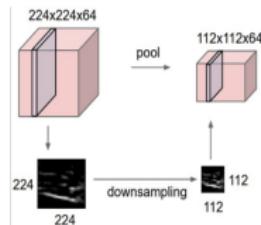
Activation Function



Convolution Layers



Pooling Layers

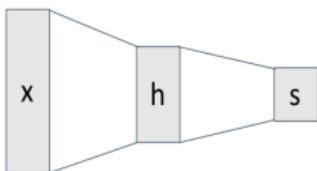


Normalization

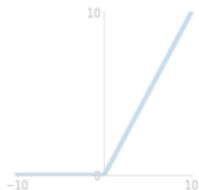
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Components of a Convolutional Network

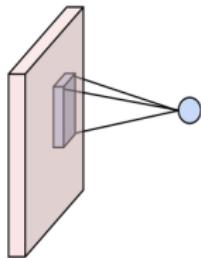
Fully-Connected Layers



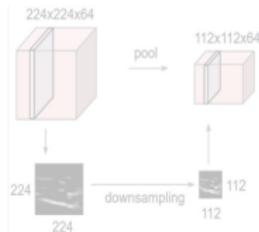
Activation Function



Convolution Layers



Pooling Layers

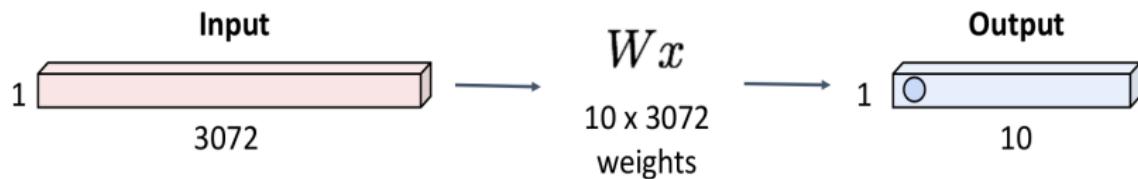


Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

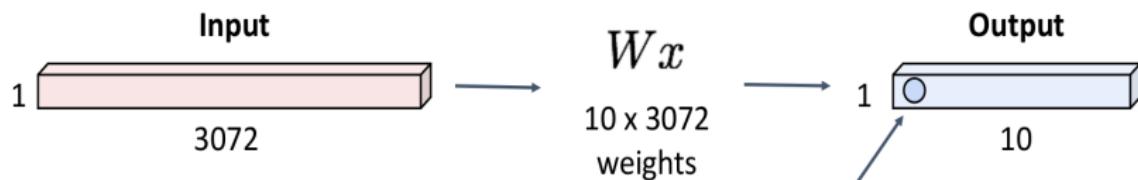
Fully-Connected Layer

32x32x3 image -> stretch to 3072 x 1



Fully-Connected Layer

32x32x3 image \rightarrow stretch to 3072 x 1

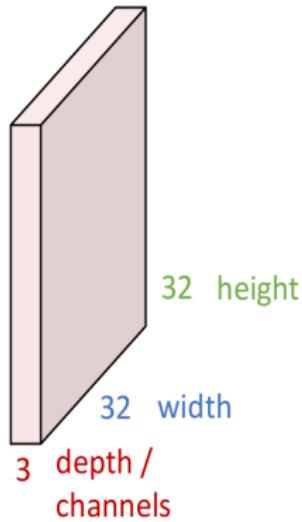


1 number:

the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

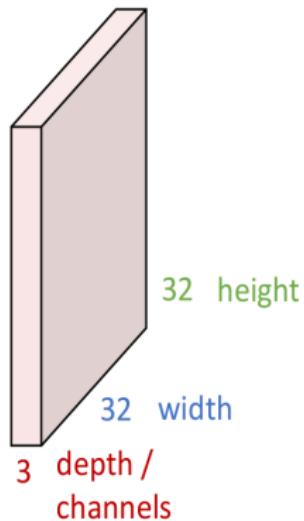
Convolution Layer

3x32x32 image: preserve spatial structure



Convolution Layer

$3 \times 32 \times 32$ image



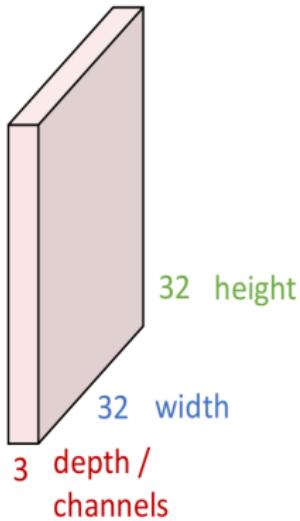
$3 \times 5 \times 5$ filter



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

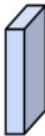
Convolution Layer

$3 \times 32 \times 32$ image



Filters always extend the full depth of the input volume

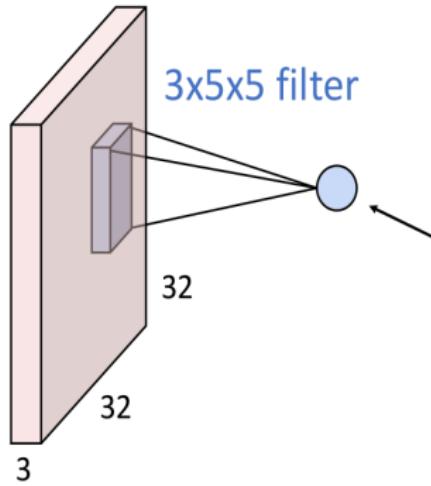
$3 \times 5 \times 5$ filter



Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products"

Convolution Layer

3x32x32 image



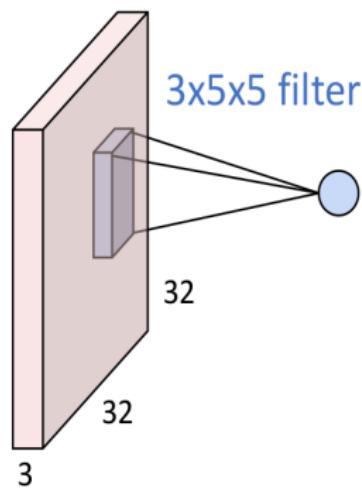
1 number:

the result of taking a dot product between the filter and a small 3x5x5 chunk of the image
(i.e. $3 \times 5 \times 5 = 75$ -dimensional dot product + bias)

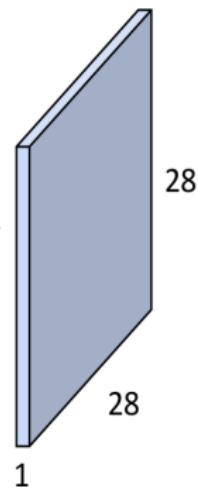
$$w^T x + b$$

Convolution Layer

3x32x32 image

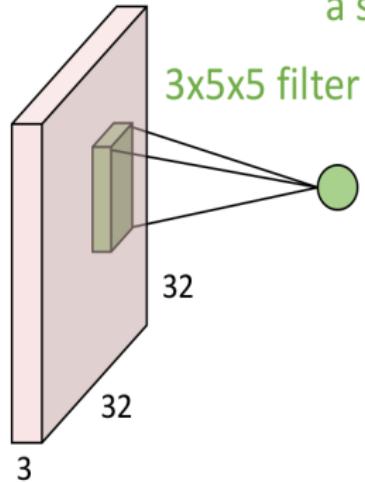


1x28x28
activation map



Convolution Layer

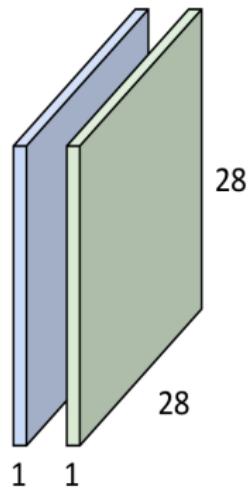
3x32x32 image



Consider repeating with
a second (green) filter:

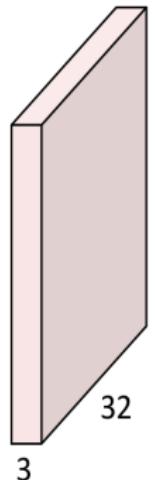
convolve (slide) over
all spatial locations

two 1x28x28
activation map



Convolution Layer

3x32x32 image



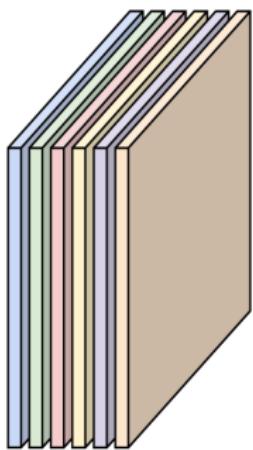
Consider 6 filters,
each 3x5x5



6x3x5x5
filters



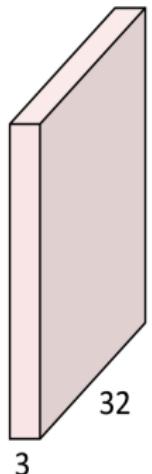
6 activation maps,
each 1x28x28



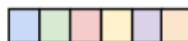
Stack activations to get a
6x28x28 output image!

Convolution Layer

3x32x32 image



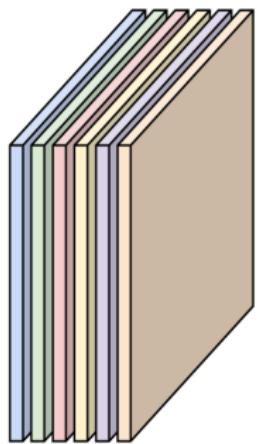
Also 6-dim bias vector:



6x3x5x5
filters



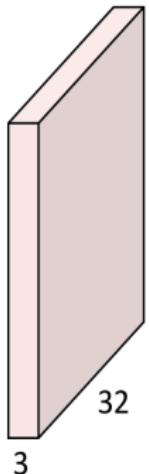
6 activation maps,
each 1x28x28



Stack activations to get a
6x28x28 output image!

Convolution Layer

3x32x32 image



Also 6-dim bias vector:

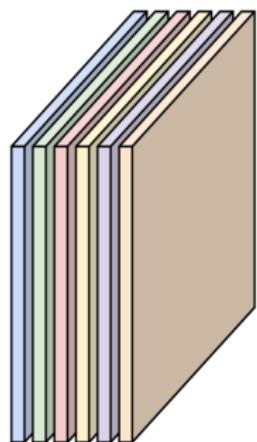


Convolution
Layer

6x3x5x5
filters



28x28 grid, at each
point a 6-dim vector

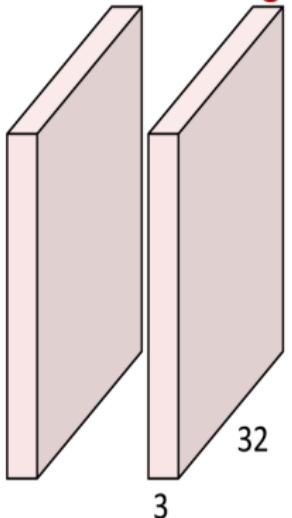


Stack activations to get a
6x28x28 output image!

Convolution Layer

$2 \times 3 \times 32 \times 32$

Batch of images



Also 6-dim bias vector:



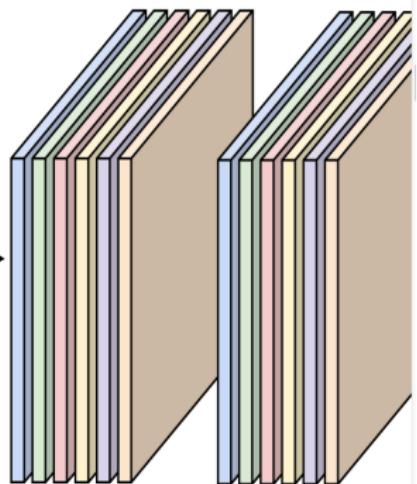
Convolution
Layer

$6 \times 3 \times 5 \times 5$
filters



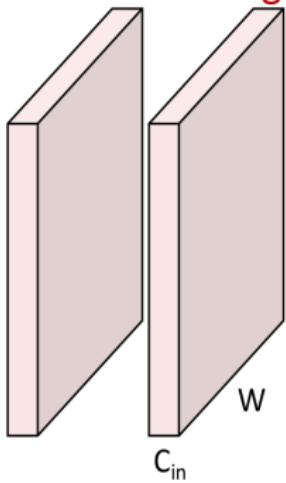
$2 \times 6 \times 28 \times 28$

Batch of outputs



Convolution Layer

$N \times C_{in} \times H \times W$
Batch of images



Also C_{out} -dim bias vector:

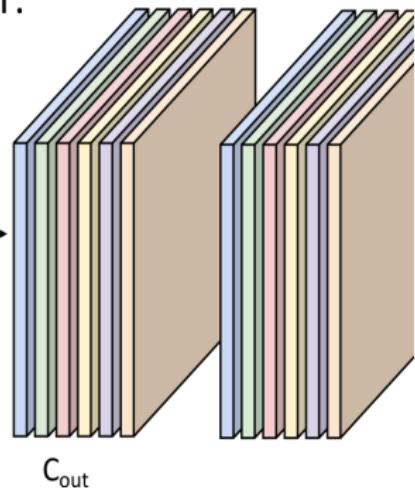


Convolution
Layer

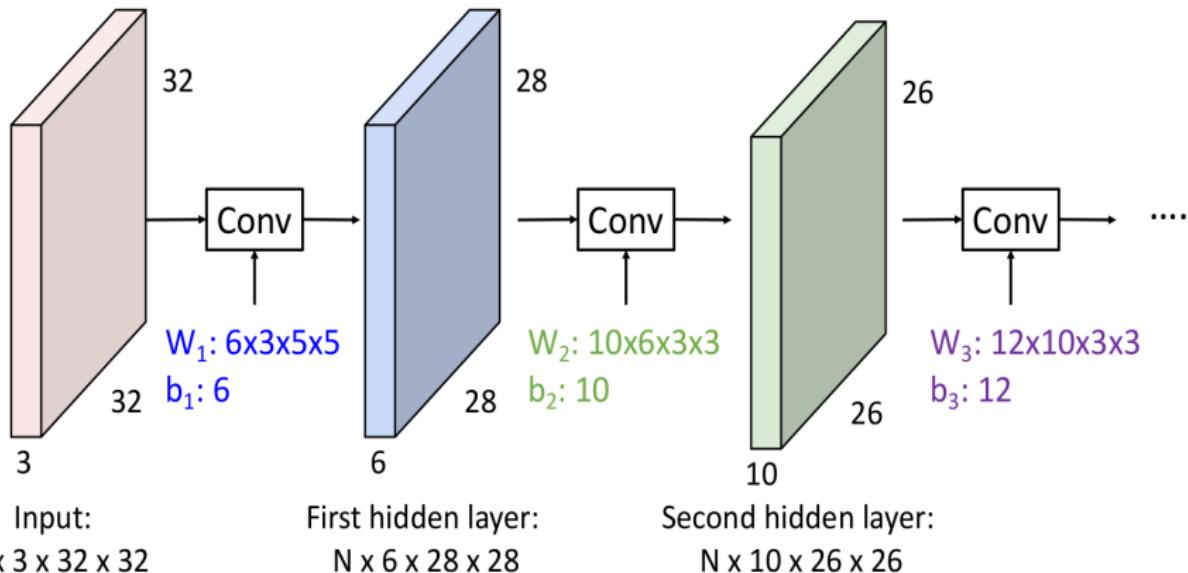
$C_{out} \times C_{in} \times K_w \times K_h$
filters



$N \times C_{out} \times H' \times W'$
Batch of outputs

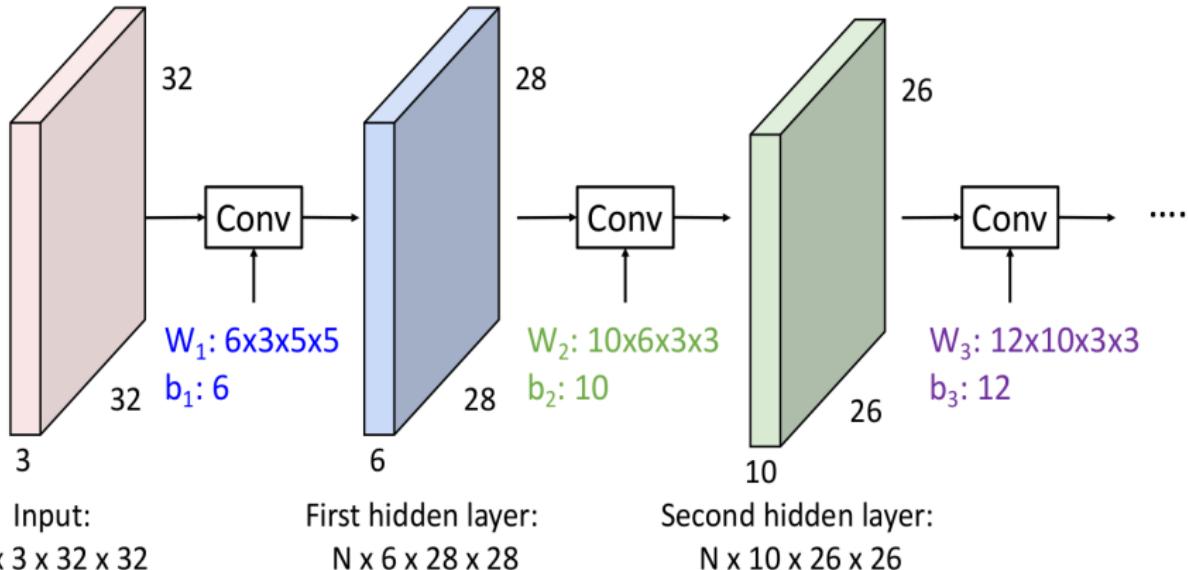


Stacking Convolutions

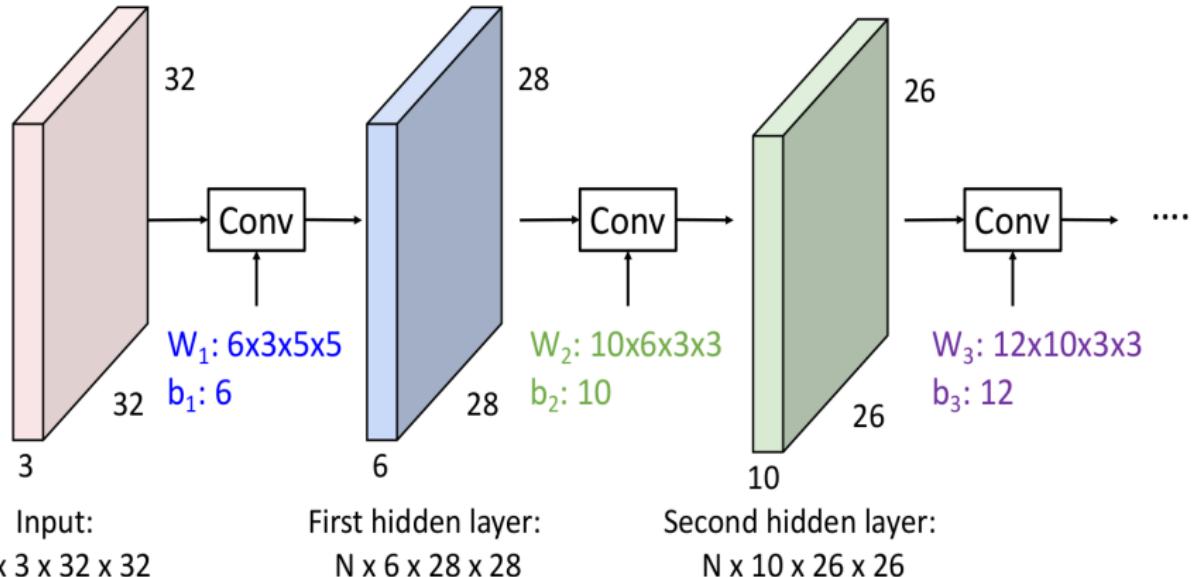


Stacking Convolutions

Q: What happens if we stack two convolution layers?



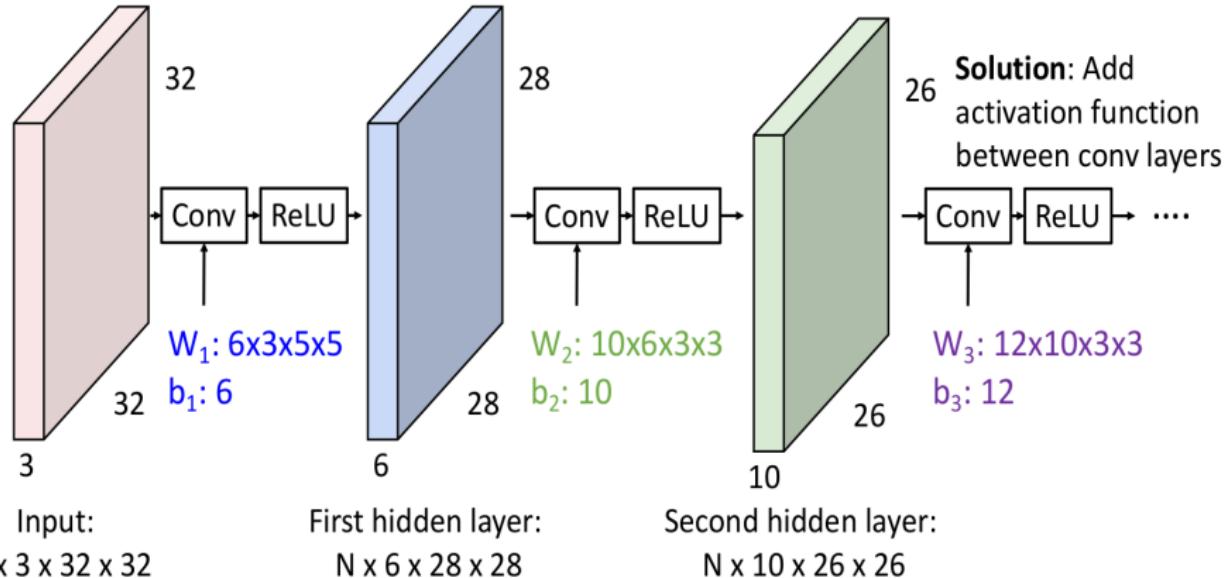
Stacking Convolutions



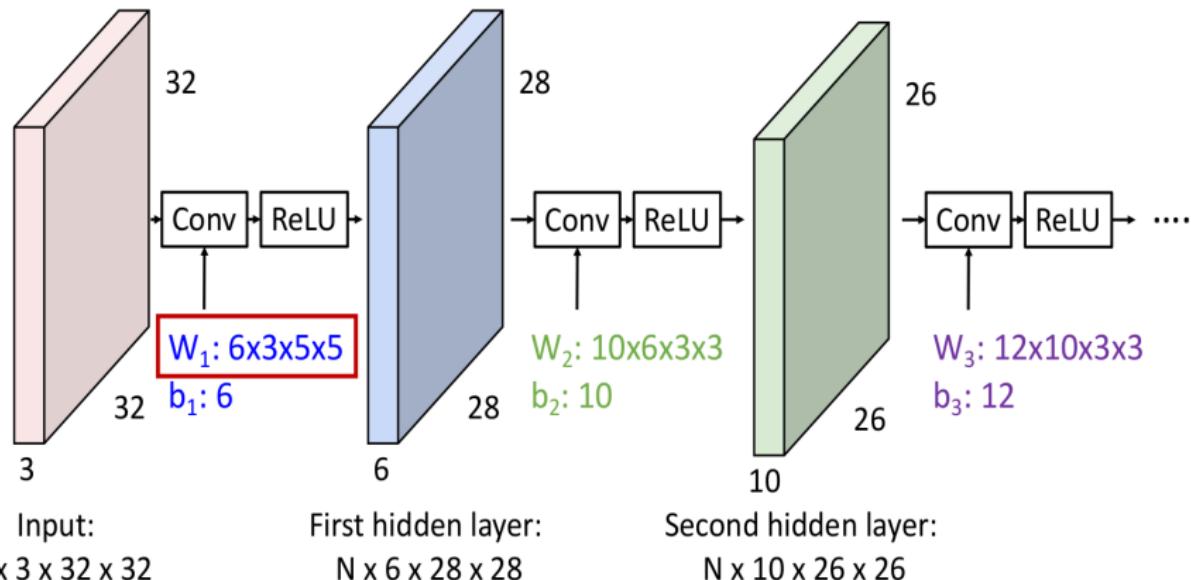
Stacking Convolutions

Q: What happens if we stack two convolution layers? (Recall $y=W_2W_1x$ is a linear classifier)

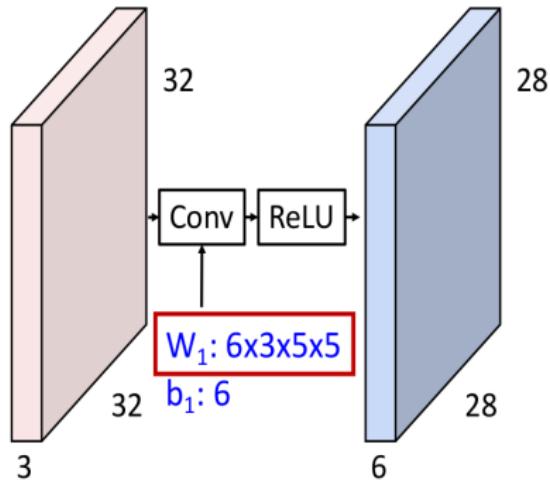
A: We get another convolution!



What do convolutional filters learn?



What do convolutional filters learn?



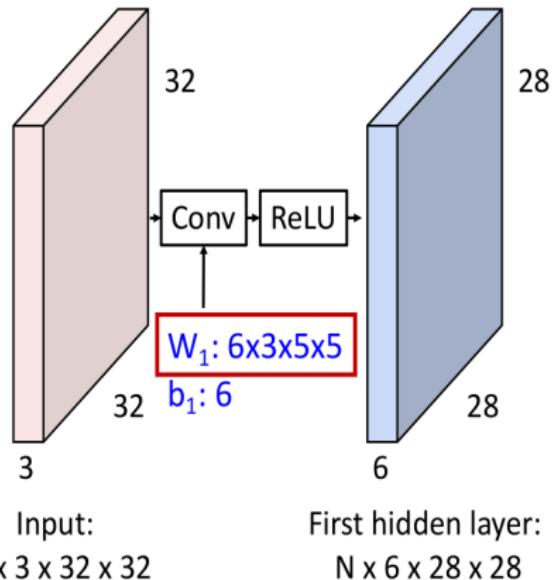
Input:
 $N \times 3 \times 32 \times 32$

First hidden layer:
 $N \times 6 \times 28 \times 28$

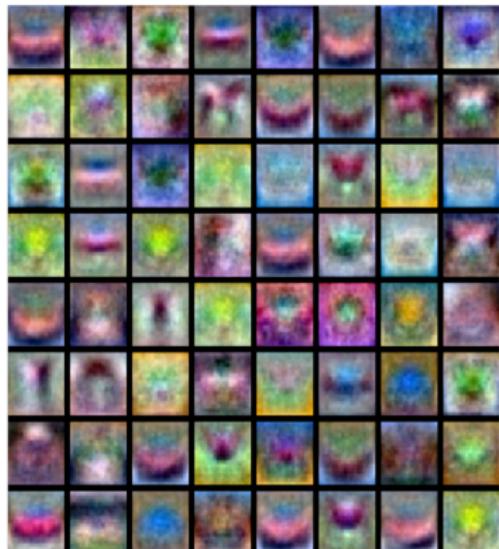
Linear classifier: One template per class



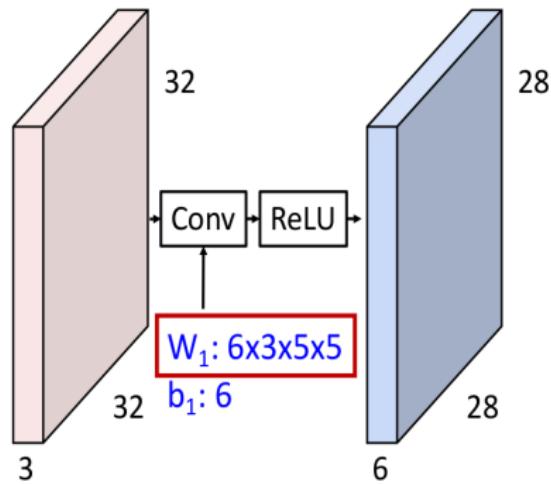
What do convolutional filters learn?



MLP: Bank of whole-image templates



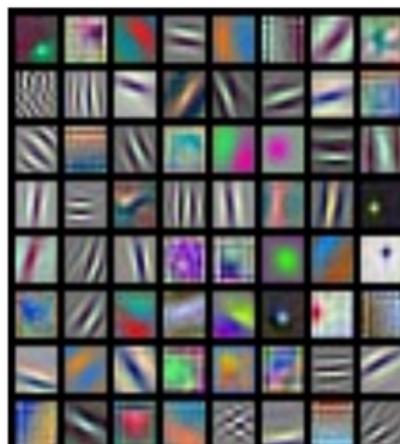
What do convolutional filters learn?



Input:
 $N \times 3 \times 32 \times 32$

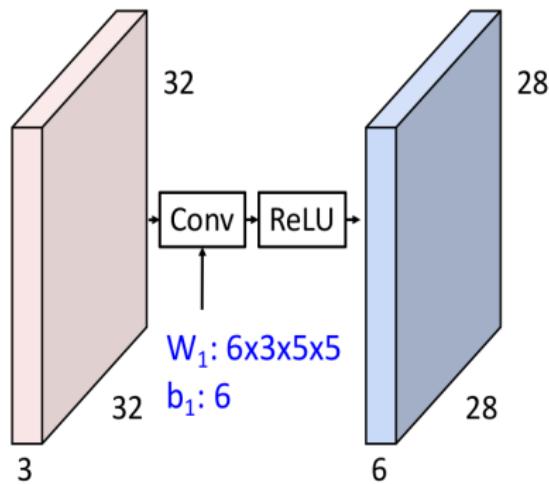
First hidden layer:
 $N \times 6 \times 28 \times 28$

First-layer conv filters: local image templates
(Often learns oriented edges, opposing colors)



AlexNet: 64 filters, each $3 \times 11 \times 11$

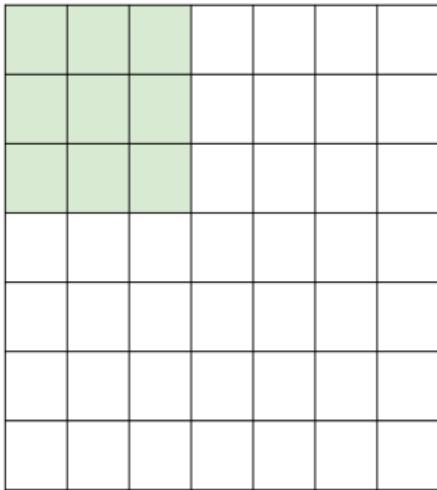
A closer look at spatial dimensions



Input:
 $N \times 3 \times 32 \times 32$

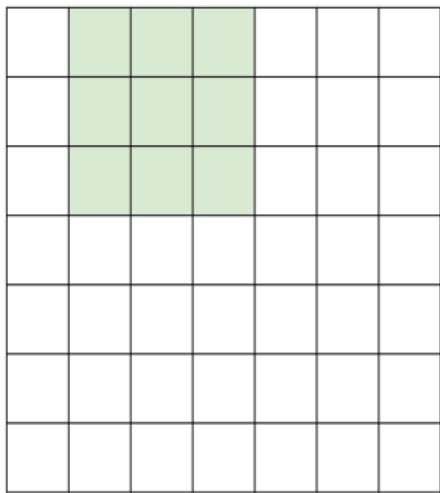
First hidden layer:
 $N \times 6 \times 28 \times 28$

A closer look at spatial dimensions



Input: 7x7
Filter: 3x3

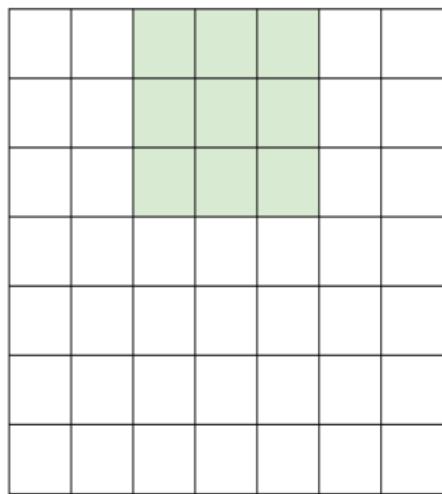
A closer look at spatial dimensions



7

Input: 7x7
Filter: 3x3

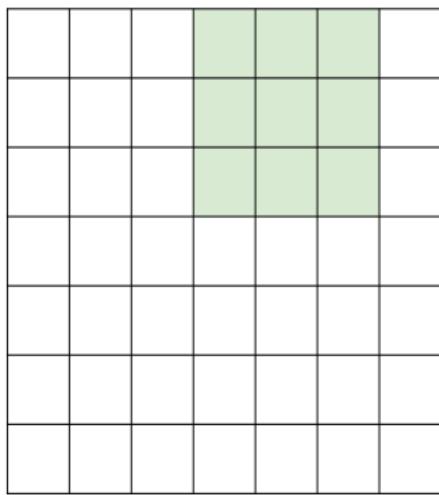
A closer look at spatial dimensions



7

Input: 7x7
Filter: 3x3

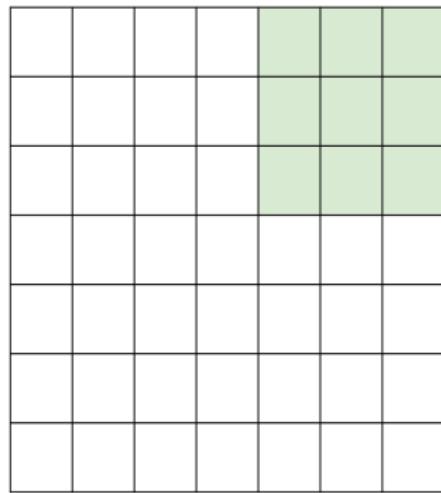
A closer look at spatial dimensions



Input: 7x7
Filter: 3x3

7

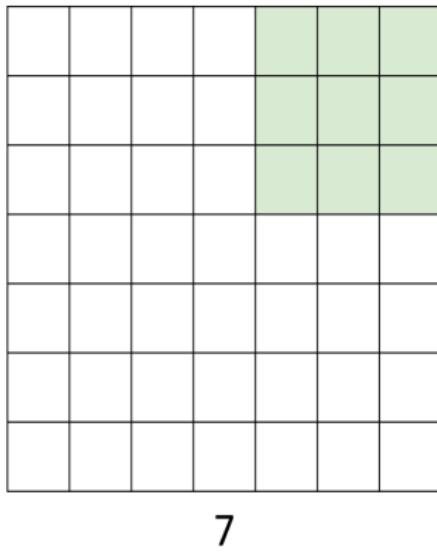
A closer look at spatial dimensions



7

Input: 7x7
Filter: 3x3
Output: 5x5

A closer look at spatial dimensions



Input: 7x7

Filter: 3x3

Output: 5x5

In general:

Input: W

Filter: K

Output: $W - K + 1$

Problem: Feature
maps “shrink”
with each layer!

A closer look at spatial dimensions

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

Input: 7x7

Filter: 3x3

Output: 5x5

In general: Problem: Feature

Input: W maps "shrink"

Filter: K with each layer!

Output: $W - K + 1$

Solution: padding

Add zeros around the input

A closer look at spatial dimensions

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

Input: 7x7

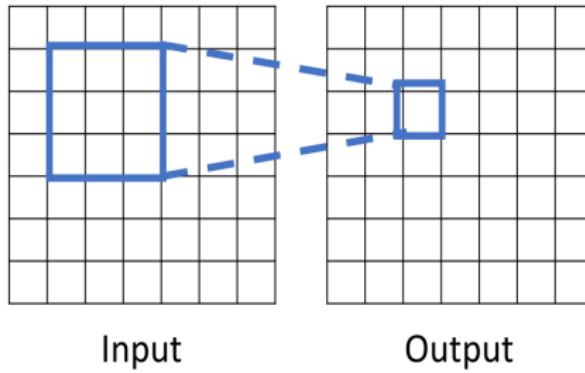
Filter: 3x3

Output: 5x5

In general: Very common:
Input: W Set $P = (K - 1) / 2$ to
Filter: K make output have
Padding: P same size as input!
Output: $W - K + 1 + 2P$

Receptive Fields

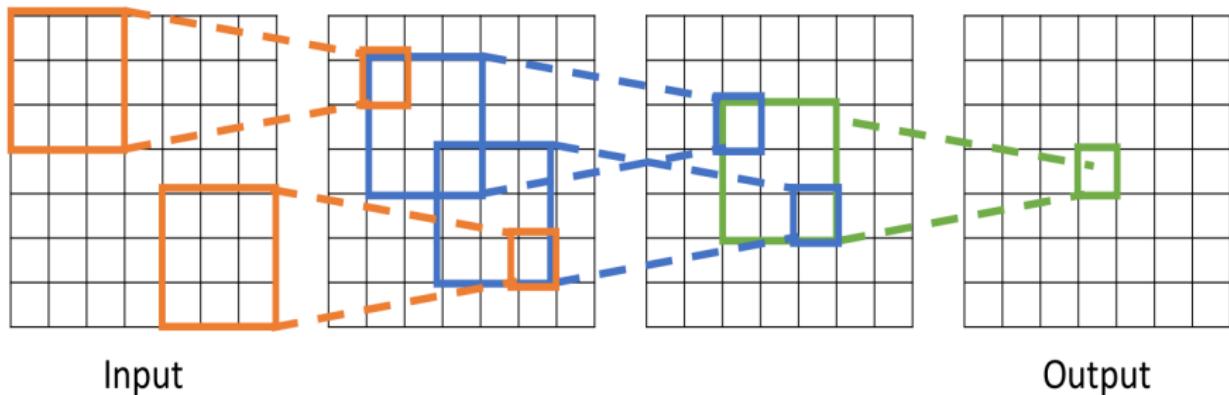
For convolution with kernel size K, each element in the output depends on a $K \times K$ **receptive field** in the input



Receptive Fields

Each successive convolution adds $K - 1$ to the receptive field size

With L layers the receptive field size is $1 + L * (K - 1)$



Input

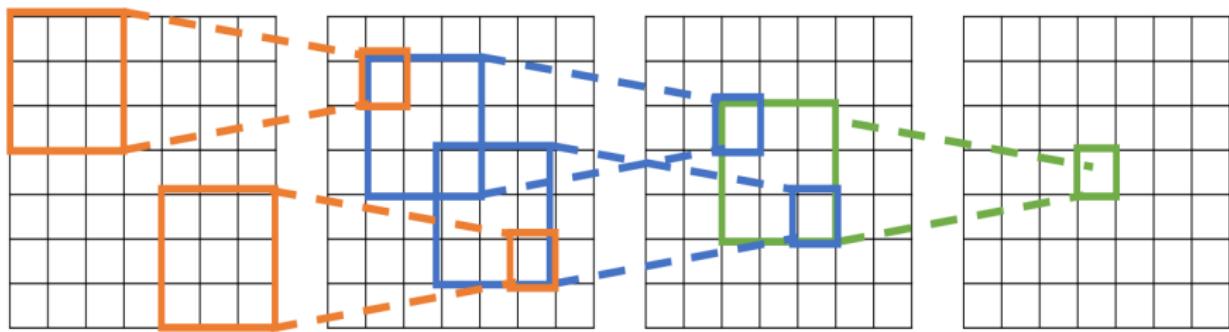
Output

Be careful – “receptive field in the input” vs “receptive field in the previous layer”
Hopefully clear from context!

Receptive Fields

Each successive convolution adds $K - 1$ to the receptive field size

With L layers the receptive field size is $1 + L * (K - 1)$



Input

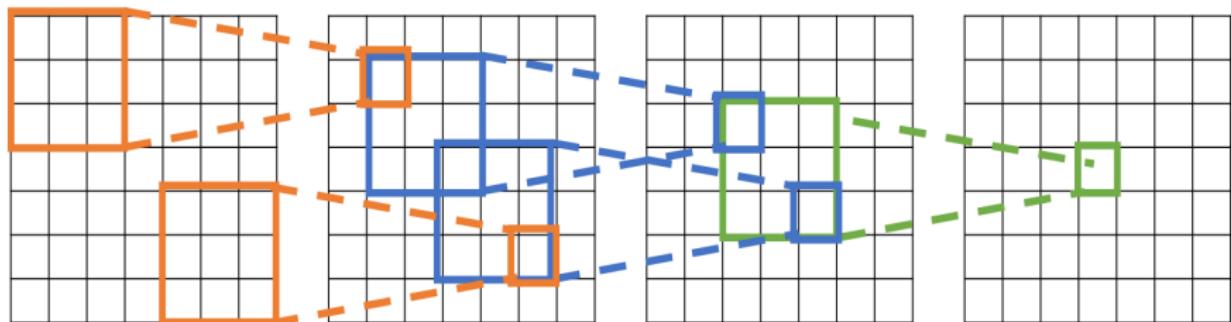
Problem: For large images we need many layers
for each output to “see” the whole image

Output

Receptive Fields

Each successive convolution adds $K - 1$ to the receptive field size

With L layers the receptive field size is $1 + L * (K - 1)$



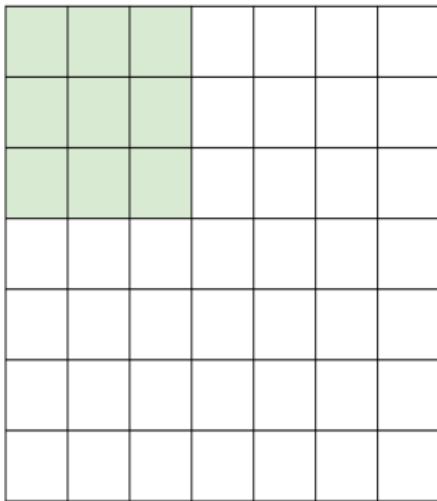
Input

Problem: For large images we need many layers
for each output to “see” the whole image

Output

Solution: Downsample inside the network

Strided Convolution

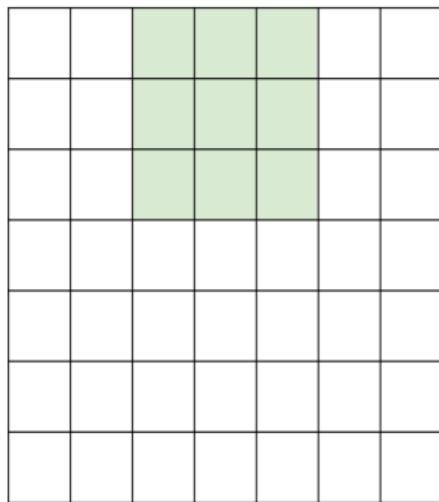


Input: 7x7

Filter: 3x3

Stride: 2

Strided Convolution

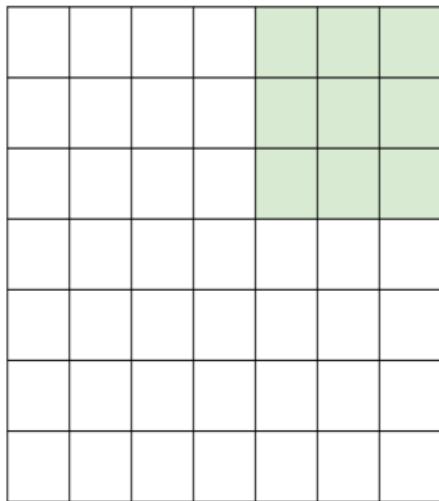


Input: 7x7

Filter: 3x3

Stride: 2

Strided Convolution



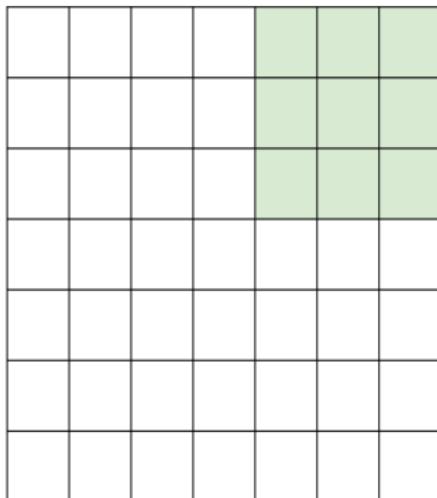
Input: 7x7

Filter: 3x3

Stride: 2

Output: 3x3

Strided Convolution



Input: 7x7

Filter: 3x3

Output: 3x3

Stride: 2

In general:

Input: W

Filter: K

Padding: P

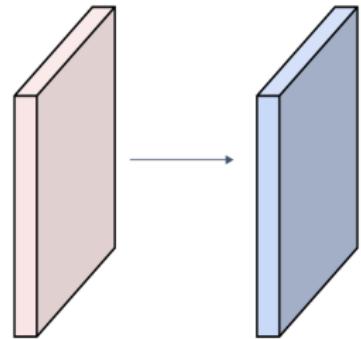
Stride: S

Output: $(W - K + 2P) / S + 1$

Convolution Example

Input volume: $3 \times 32 \times 32$

10 5x5 filters with stride 1, pad 2

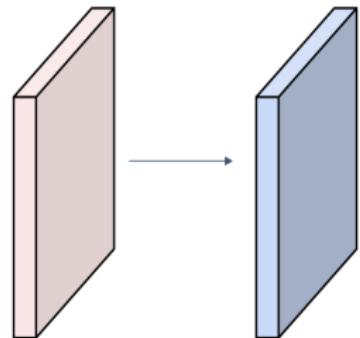


Output volume size: ?

Convolution Example

Input volume: $3 \times 32 \times 32$

10 **5x5** filters with stride **1**, pad **2**



Output volume size:

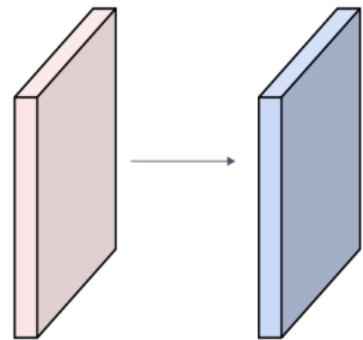
$(32+2*2-5)/1+1 = 32$ spatially, so

10 $\times 32 \times 32$

Convolution Example

Input volume: $3 \times 32 \times 32$

10 5x5 filters with stride 1, pad 2



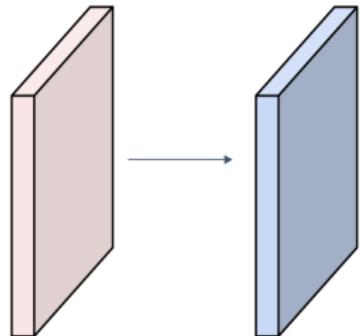
Output volume size: $10 \times 32 \times 32$

Number of learnable parameters: ?

Convolution Example

Input volume: **3** x 32 x 32

10 **5x5** filters with stride 1, pad 2



Output volume size: 10 x 32 x 32

Number of learnable parameters: **760**

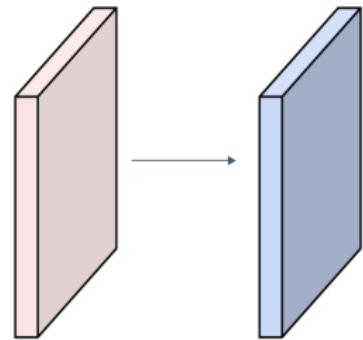
Parameters per filter: **3*5*5 + 1** (for bias) = **76**

10 filters, so total is **10 * 76 = 760**

Convolution Example

Input volume: $3 \times 32 \times 32$

10 5x5 filters with stride 1, pad 2



Output volume size: $10 \times 32 \times 32$

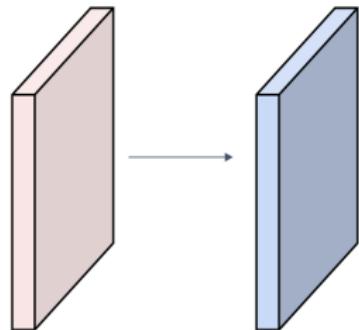
Number of learnable parameters: 760

Number of multiply-add operations: ?

Convolution Example

Input volume: **3 x 32 x 32**

10 **5x5** filters with stride 1, pad 2



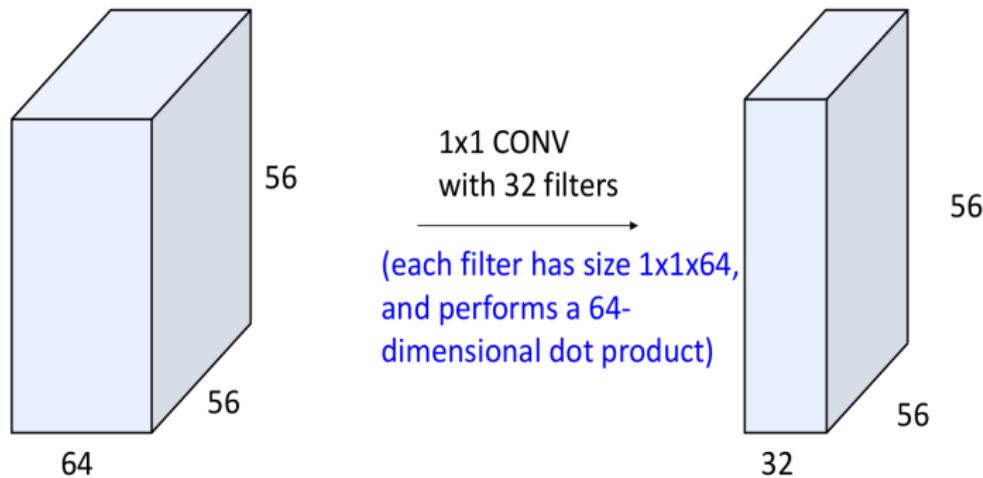
Output volume size: **10 x 32 x 32**

Number of learnable parameters: 760

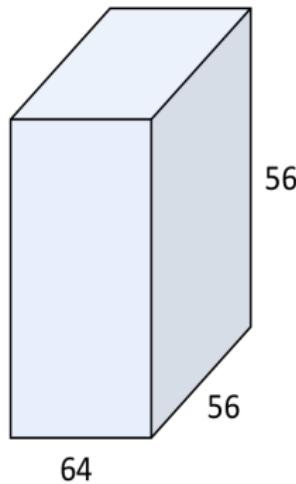
Number of multiply-add operations: **768,000**

$10*32*32 = 10,240$ outputs; each output is the inner product of two **$3x5x5$** tensors (75 elems); total = $75*10240 = 768K$

Example: 1x1 Convolution



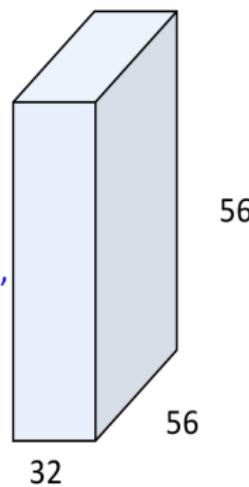
Example: 1x1 Convolution



1x1 CONV
with 32 filters

(each filter has size $1 \times 1 \times 64$,
and performs a 64-dimensional dot product)

Stacking 1x1 conv layers
gives MLP operating on
each input position



Lin et al, "Network in Network", ICLR 2014

Convolution Summary

Input: $C_{in} \times H \times W$

Hyperparameters:

- **Kernel size:** $K_H \times K_W$
- **Number filters:** C_{out}
- **Padding:** P
- **Stride:** S

Weight matrix: $C_{out} \times C_{in} \times K_H \times K_W$

giving C_{out} filters of size $C_{in} \times K_H \times K_W$

Bias vector: C_{out}

Output size: $C_{out} \times H' \times W'$ where:

- $H' = (H - K + 2P) / S + 1$
- $W' = (W - K + 2P) / S + 1$

Convolution Summary

Input: $C_{in} \times H \times W$

Hyperparameters:

- **Kernel size:** $K_H \times K_W$
- **Number filters:** C_{out}
- **Padding:** P
- **Stride:** S

Weight matrix: $C_{out} \times C_{in} \times K_H \times K_W$
giving C_{out} filters of size $C_{in} \times K_H \times K_W$

Bias vector: C_{out}

Output size: $C_{out} \times H' \times W'$ where:

- $H' = (H - K + 2P) / S + 1$
- $W' = (W - K + 2P) / S + 1$

Common settings:

$K_H = K_W$ (Small square filters)

$P = (K - 1) / 2$ ("Same" padding)

$C_{in}, C_{out} = 32, 64, 128, 256$ (powers of 2)

$K = 3, P = 1, S = 1$ (3x3 conv)

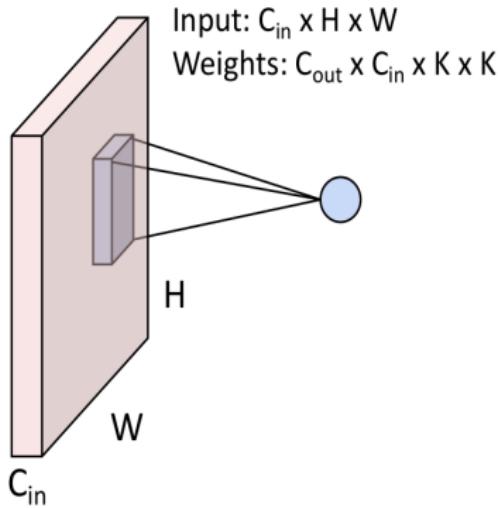
$K = 5, P = 2, S = 1$ (5x5 conv)

$K = 1, P = 0, S = 1$ (1x1 conv)

$K = 3, P = 1, S = 2$ (Downsample by 2)

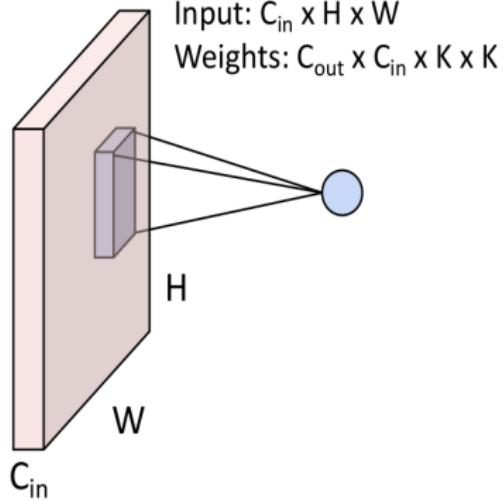
Other types of convolution

So far: 2D Convolution

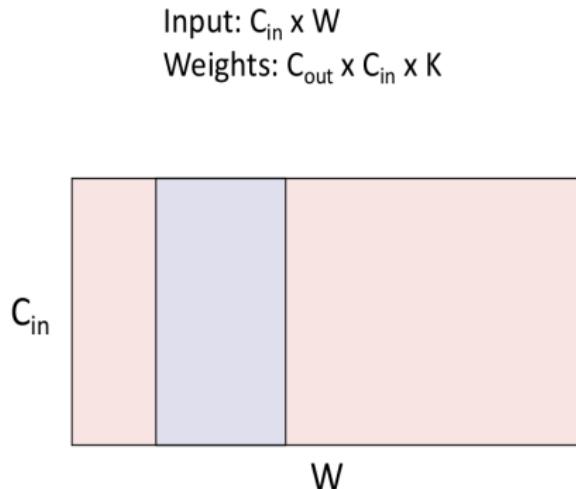


Other types of convolution

So far: 2D Convolution

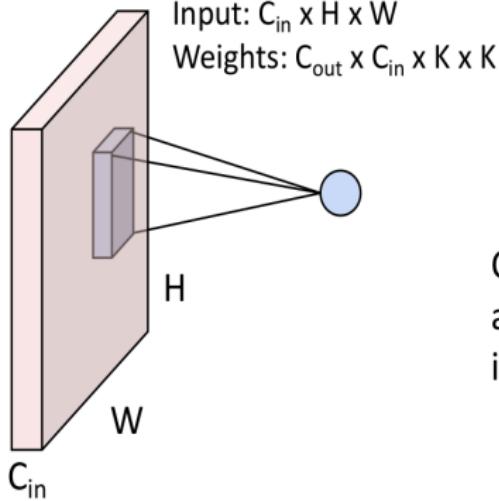


1D Convolution

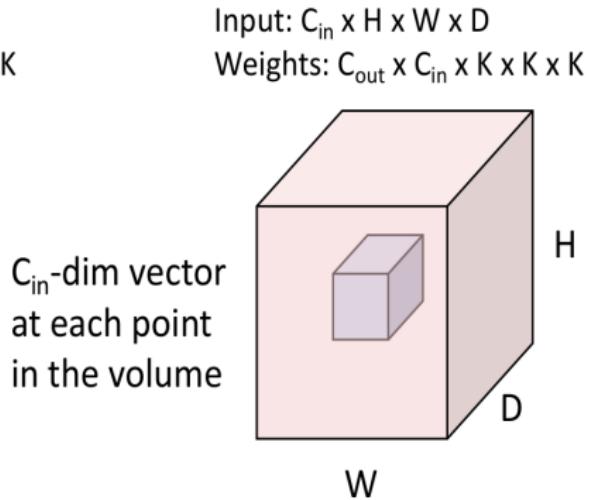


Other types of convolution

So far: 2D Convolution



3D Convolution



PyTorch Convolution Layer

Conv2d

```
CLASS torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,  
dilation=1, groups=1, bias=True, padding_mode='zeros')
```

[SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

PyTorch Convolution Layers

Conv2d

```
CLASS torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,  
dilation=1, groups=1, bias=True, padding_mode='zeros')
```

[SOURCE]

Conv1d

```
CLASS torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=1, padding=0,  
dilation=1, groups=1, bias=True, padding_mode='zeros')
```

[SOURCE] ↗

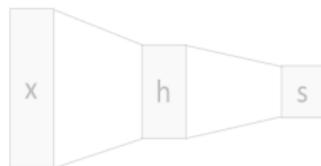
Conv3d

```
CLASS torch.nn.Conv3d(in_channels, out_channels, kernel_size, stride=1, padding=0,  
dilation=1, groups=1, bias=True, padding_mode='zeros')
```

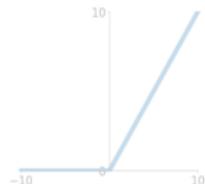
[SOURCE]

Components of a Convolutional Network

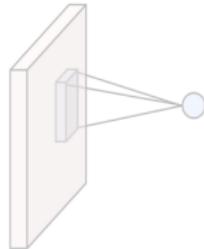
Fully-Connected Layers



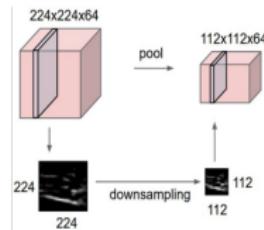
Activation Function



Convolution Layers



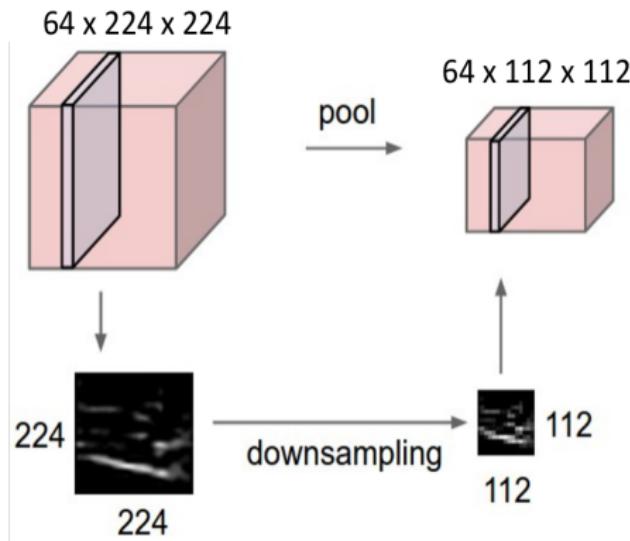
Pooling Layers



Normalization

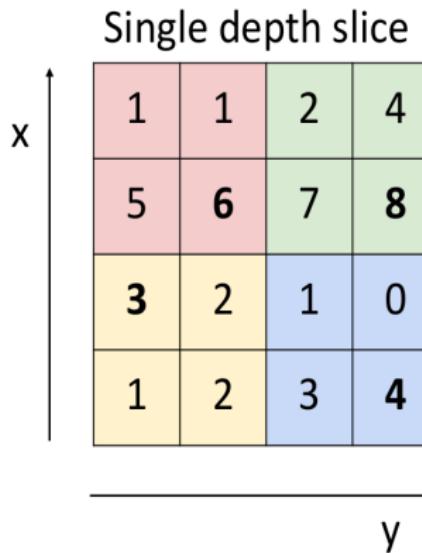
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Pooling Layers: Another way to downsample

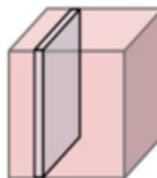


Hyperparameters:
Kernel Size
Stride
Pooling function

Max Pooling



64 x 224 x 224



Max pooling with 2x2 kernel size and stride 2



6	8
3	4

Introduces **invariance** to
small spatial shifts
No learnable parameters!

Pooling Summary

Input: $C \times H \times W$

Hyperparameters:

- Kernel size: K
- Stride: S
- Pooling function (max, avg)

Common settings:

max, $K = 2, S = 2$

max, $K = 3, S = 2$ (AlexNet)

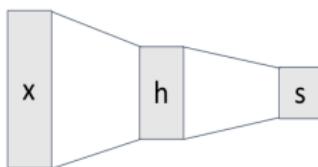
Output: $C \times H' \times W'$ where

- $H' = (H - K) / S + 1$
- $W' = (W - K) / S + 1$

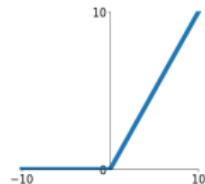
Learnable parameters: None!

Components of a Convolutional Network

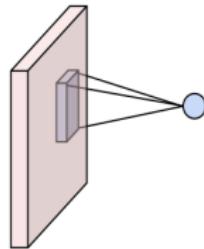
Fully-Connected Layers



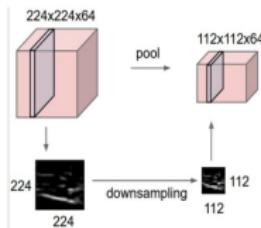
Activation Function



Convolution Layers



Pooling Layers



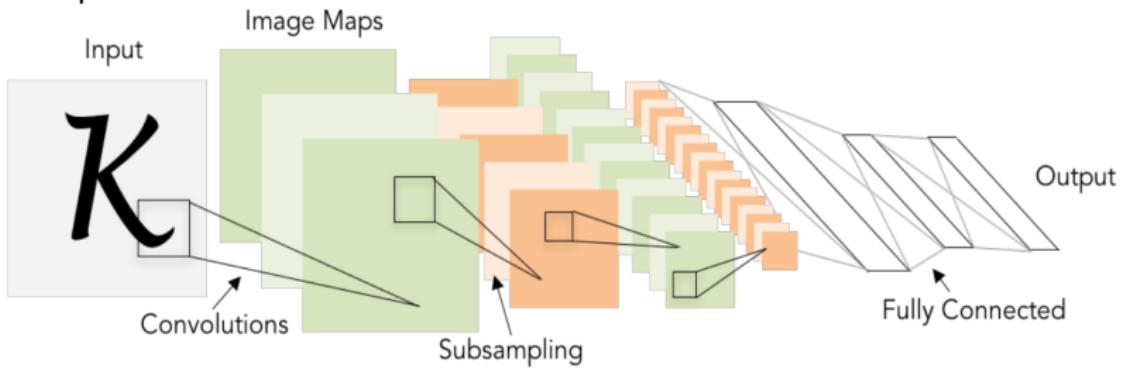
Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Convolutional Networks

Classic architecture: [Conv, ReLU, Pool] x N, flatten, [FC, ReLU] x N, FC

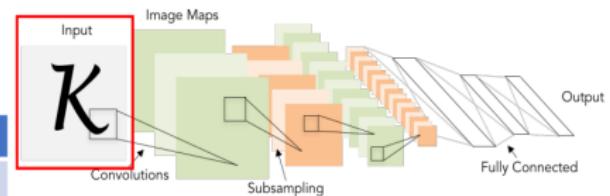
Example: LeNet-5



Lecun et al, "Gradient-based learning applied to document recognition", 1998

Example: LeNet-5

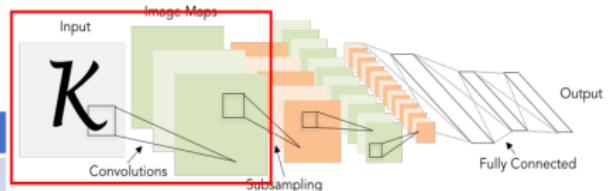
Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	



Lecun et al, "Gradient-based learning applied to document recognition", 1998

Example: LeNet-5

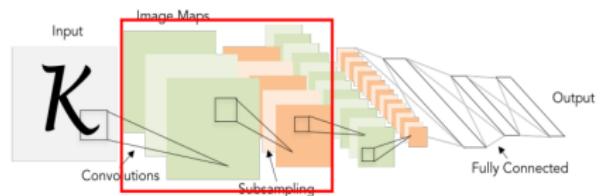
Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	
Conv ($C_{out}=20$, $K=5$, $P=2$, $S=1$)	$20 \times 28 \times 28$	$20 \times 1 \times 5 \times 5$
ReLU	$20 \times 28 \times 28$	



Lecun et al, "Gradient-based learning applied to document recognition", 1998

Example: LeNet-5

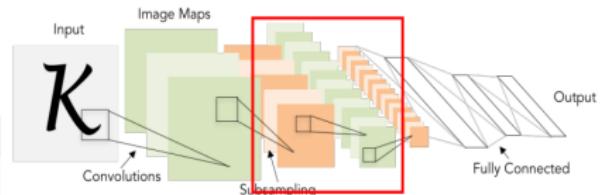
Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	
Conv ($C_{out}=20$, $K=5$, $P=2$, $S=1$)	$20 \times 28 \times 28$	$20 \times 1 \times 5 \times 5$
ReLU	$20 \times 28 \times 28$	
MaxPool($K=2$, $S=2$)	$20 \times 14 \times 14$	



Lecun et al, "Gradient-based learning applied to document recognition", 1998

Example: LeNet-5

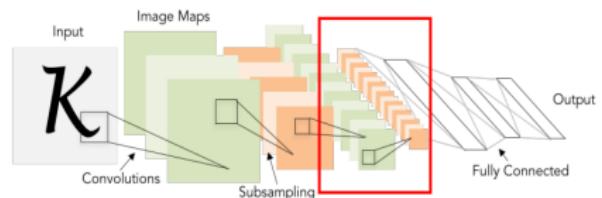
Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	
Conv ($C_{out}=20$, $K=5$, $P=2$, $S=1$)	$20 \times 28 \times 28$	$20 \times 1 \times 5 \times 5$
ReLU	$20 \times 28 \times 28$	
MaxPool($K=2$, $S=2$)	$20 \times 14 \times 14$	
Conv ($C_{out}=50$, $K=5$, $P=2$, $S=1$)	$50 \times 14 \times 14$	$50 \times 20 \times 5 \times 5$
ReLU	$50 \times 14 \times 14$	



Lecun et al, "Gradient-based learning applied to document recognition", 1998

Example: LeNet-5

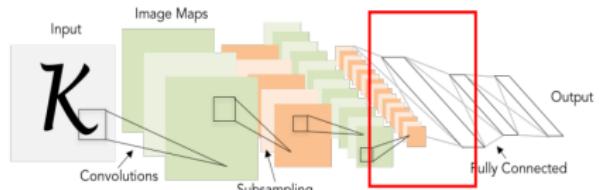
Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	
Conv ($C_{out}=20, K=5, P=2, S=1$)	$20 \times 28 \times 28$	$20 \times 1 \times 5 \times 5$
ReLU	$20 \times 28 \times 28$	
MaxPool($K=2, S=2$)	$20 \times 14 \times 14$	
Conv ($C_{out}=50, K=5, P=2, S=1$)	$50 \times 14 \times 14$	$50 \times 20 \times 5 \times 5$
ReLU	$50 \times 14 \times 14$	
MaxPool($K=2, S=2$)	$50 \times 7 \times 7$	



Lecun et al, "Gradient-based learning applied to document recognition", 1998

Example: LeNet-5

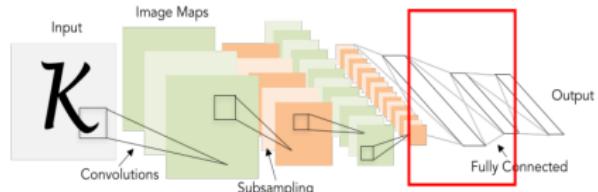
Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	
Conv ($C_{out}=20, K=5, P=2, S=1$)	$20 \times 28 \times 28$	$20 \times 1 \times 5 \times 5$
ReLU	$20 \times 28 \times 28$	
MaxPool($K=2, S=2$)	$20 \times 14 \times 14$	
Conv ($C_{out}=50, K=5, P=2, S=1$)	$50 \times 14 \times 14$	$50 \times 20 \times 5 \times 5$
ReLU	$50 \times 14 \times 14$	
MaxPool($K=2, S=2$)	$50 \times 7 \times 7$	
Flatten	2450	



Lecun et al, "Gradient-based learning applied to document recognition", 1998

Example: LeNet-5

Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	
Conv ($C_{out}=20, K=5, P=2, S=1$)	$20 \times 28 \times 28$	$20 \times 1 \times 5 \times 5$
ReLU	$20 \times 28 \times 28$	
MaxPool($K=2, S=2$)	$20 \times 14 \times 14$	
Conv ($C_{out}=50, K=5, P=2, S=1$)	$50 \times 14 \times 14$	$50 \times 20 \times 5 \times 5$
ReLU	$50 \times 14 \times 14$	
MaxPool($K=2, S=2$)	$50 \times 7 \times 7$	
Flatten	2450	
Linear (2450 -> 500)	500	2450×500
ReLU	500	



Lecun et al, "Gradient-based learning applied to document recognition", 1998

Example: LeNet-5

Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	
Conv ($C_{out}=20, K=5, P=2, S=1$)	$20 \times 28 \times 28$	$20 \times 1 \times 5 \times 5$
ReLU	$20 \times 28 \times 28$	
MaxPool($K=2, S=2$)	$20 \times 14 \times 14$	
Conv ($C_{out}=50, K=5, P=2, S=1$)	$50 \times 14 \times 14$	$50 \times 20 \times 5 \times 5$
ReLU	$50 \times 14 \times 14$	
MaxPool($K=2, S=2$)	$50 \times 7 \times 7$	
Flatten	2450	
Linear ($2450 \rightarrow 500$)	500	2450×500
ReLU	500	
Linear ($500 \rightarrow 10$)	10	500×10

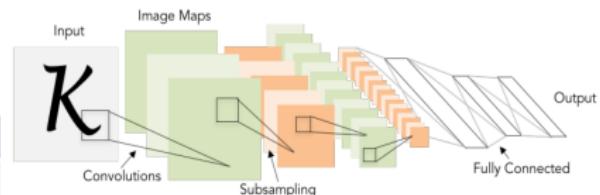


Lecun et al, "Gradient-based learning applied to document recognition", 1998

Example: LeNet-5

Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	
Conv ($C_{out}=20$, $K=5$, $P=2$, $S=1$)	$20 \times 28 \times 28$	$20 \times 1 \times 5 \times 5$
ReLU	$20 \times 28 \times 28$	
MaxPool($K=2$, $S=2$)	$20 \times 14 \times 14$	
Conv ($C_{out}=50$, $K=5$, $P=2$, $S=1$)	$50 \times 14 \times 14$	$50 \times 20 \times 5 \times 5$
ReLU	$50 \times 14 \times 14$	
MaxPool($K=2$, $S=2$)	$50 \times 7 \times 7$	
Flatten	2450	
Linear (2450 \rightarrow 500)	500	2450×500
ReLU	500	
Linear (500 \rightarrow 10)	10	500×10

Lecun et al, "Gradient-based learning applied to document recognition", 1998



As we go through the network:

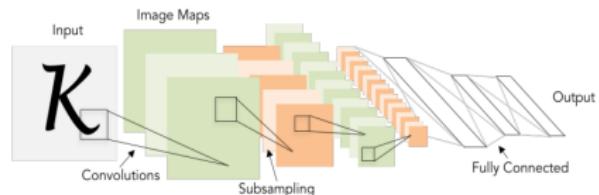
Spatial size **decreases**
(using pooling or strided conv)

Number of channels **increases**
(total "volume" is preserved!)

Example: LeNet-5

Layer	Output Size	Weight Size
Input	$1 \times 28 \times 28$	
Conv ($C_{out}=20$, $K=5$, $P=2$, $S=1$)	$20 \times 28 \times 28$	$20 \times 1 \times 5 \times 5$
ReLU	$20 \times 28 \times 28$	
MaxPool($K=2$, $S=2$)	$20 \times 14 \times 14$	
Conv ($C_{out}=50$, $K=5$, $P=2$, $S=1$)	$50 \times 14 \times 14$	$50 \times 20 \times 5 \times 5$
ReLU	$50 \times 14 \times 14$	
MaxPool($K=2$, $S=2$)	$50 \times 7 \times 7$	
Flatten	2450	
Linear (2450 \rightarrow 500)	500	2450×500
ReLU	500	
Linear (500 \rightarrow 10)	10	500×10

Lecun et al, "Gradient-based learning applied to document recognition", 1998



As we go through the network:

Spatial size **decreases**
(using pooling or strided conv)

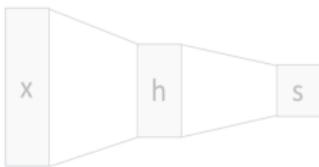
Number of channels **increases**
(total "volume" is preserved!)

Some modern architectures
break this trend -- stay tuned!

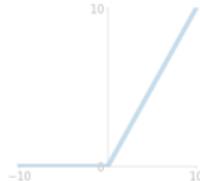
Problem: Deep Networks very hard to train!

Components of a Convolutional Network

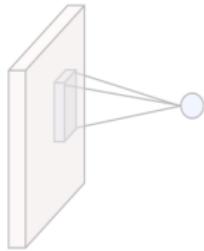
Fully-Connected Layers



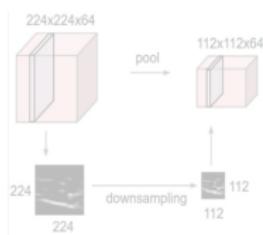
Activation Function



Convolution Layers



Pooling Layers



Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Batch Normalization

Idea: “Normalize” the outputs of a layer so they have zero mean and unit variance

Why? Helps reduce “internal covariate shift”, improves optimization

We can normalize a batch of activations like this:

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}$$

This is a **differentiable function**, so we can use it as an operator in our networks and backprop through it!

Ioffe and Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift”, ICML 2015

Batch Normalization

Input: $x \in \mathbb{R}^{N \times D}$

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j}$$

Per-channel
mean, shape is D

N

X



$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (x_{i,j} - \mu_j)^2$$

Per-channel
std, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Normalized x,
Shape is $N \times D$

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch Normalization

Input: $x \in \mathbb{R}^{N \times D}$

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j}$$

Per-channel
mean, shape is D

N

X



D

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (x_{i,j} - \mu_j)^2$$

Per-channel
std, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Normalized x,
Shape is $N \times D$

Problem: What if zero-mean, unit variance is too hard of a constraint?

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch Normalization

Input: $x \in \mathbb{R}^{N \times D}$

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j}$$

Per-channel mean, shape is D

Learnable scale and shift parameters:

$$\gamma, \beta \in \mathbb{R}^D$$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function (in expectation)

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (x_{i,j} - \mu_j)^2$$

Per-channel std, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x,
Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output,
Shape is N x D

Batch Normalization

Problem: Estimates depend on minibatch; can't do this at test-time!

Input: $x \in \mathbb{R}^{N \times D}$

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j} \quad \text{Per-channel mean, shape is } D$$

Learnable scale and shift parameters:

$$\gamma, \beta \in \mathbb{R}^D$$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function (in expectation)

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (x_{i,j} - \mu_j)^2 \quad \text{Per-channel std, shape is } D$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}} \quad \text{Normalized x, Shape is } N \times D$$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j \quad \text{Output, Shape is } N \times D$$

Batch Normalization: Test-Time

Input: $x \in \mathbb{R}^{N \times D}$

μ_j = (Running) average of values seen during training
Per-channel mean, shape is D

Learnable scale and shift parameters:

$\gamma, \beta \in \mathbb{R}^D$

Learning $\gamma = \sigma, \beta = \mu$
will recover the identity function (in expectation)

σ_j^2 = (Running) average of values seen during training
Per-channel std, shape is D

$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$
Normalized x,
Shape is N x D

$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$
Output,
Shape is N x D

Batch Normalization: Test-Time

Input: $x \in \mathbb{R}^{N \times D}$

μ_j = (Running) average of
values seen during
training

Per-channel
mean, shape is D

**Learnable scale and
shift parameters:**

$$\gamma, \beta \in \mathbb{R}^D$$

Learning $\gamma = \sigma$, $\beta = \mu$
will recover the identity
function (in expectation)

$$\mu_j^{test} = 0$$

For each training iteration:

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j}$$

$$\mu_j^{test} = 0.99 \mu_j^{test} + 0.01 \mu_j$$

(Similar for σ)

Batch Normalization: Test-Time

Input: $x \in \mathbb{R}^{N \times D}$

μ_j = (Running) average of values seen during training
Per-channel mean, shape is D

Learnable scale and shift parameters:

$\gamma, \beta \in \mathbb{R}^D$

σ_j^2 = (Running) average of values seen during training
Per-channel std, shape is D

$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$
Normalized x,
Shape is N x D

$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$
Output,
Shape is N x D

Learning $\gamma = \sigma, \beta = \mu$
will recover the identity function (in expectation)

Batch Normalization: Test-Time

Input: $x \in \mathbb{R}^{N \times D}$

μ_j = (Running) average of values seen during training
Per-channel mean, shape is D

Learnable scale and shift parameters:

$\gamma, \beta \in \mathbb{R}^D$

σ_j^2 = (Running) average of values seen during training
Per-channel std, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Normalized x,
Shape is N x D

During testing batchnorm becomes a linear operator!

Can be fused with the previous fully-connected or conv layer

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output,
Shape is N x D

Batch Normalization for ConvNets

Batch Normalization for
fully-connected networks

$$x : N \times D$$

Normalize

$$\mu, \sigma : 1 \times D$$
$$\gamma, \beta : 1 \times D$$
$$y = \frac{(x - \mu)}{\sigma} \gamma + \beta$$

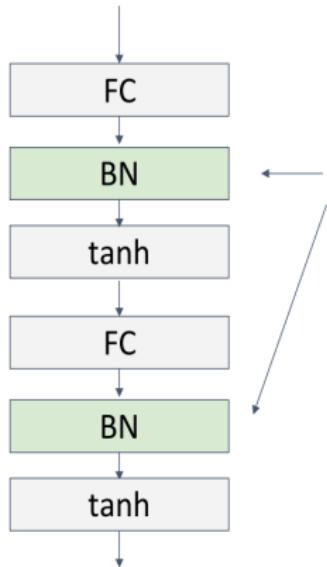
Batch Normalization for
convolutional networks
(Spatial Batchnorm, BatchNorm2D)

$$x : N \times C \times H \times W$$

Normalize

$$\mu, \sigma : 1 \times C \times 1 \times 1$$
$$\gamma, \beta : 1 \times C \times 1 \times 1$$
$$y = \frac{(x - \mu)}{\sigma} \gamma + \beta$$

Batch Normalization

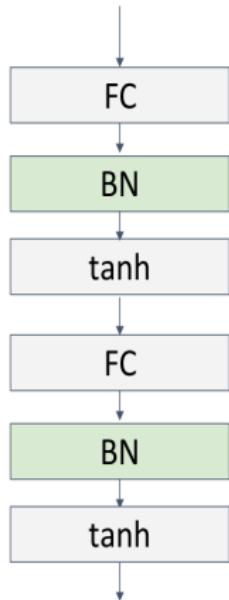


Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

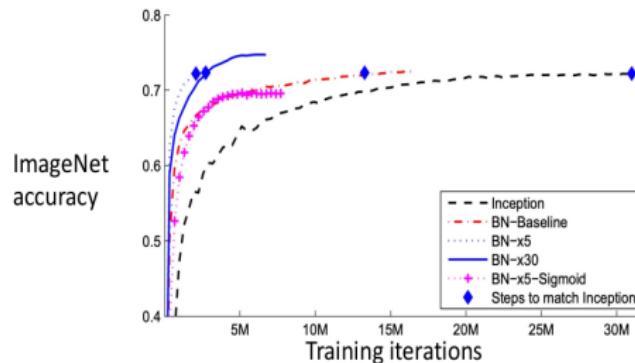
$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}$$

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch Normalization

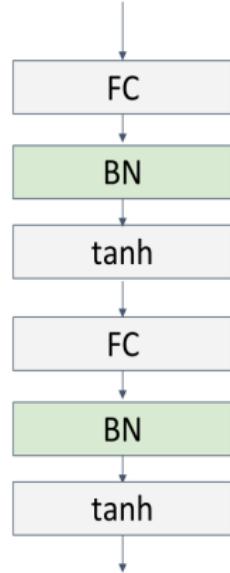


- Makes deep networks **much** easier to train!
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!



Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch Normalization

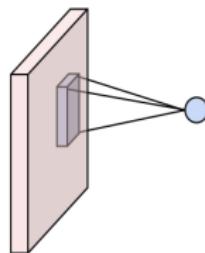


- Makes deep networks **much** easier to train!
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!
- Not well-understood theoretically (yet)
- Behaves differently during training and testing: this is a very common source of bugs!

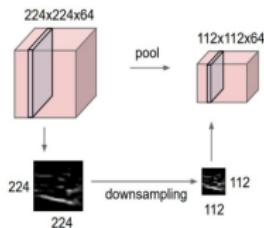
Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Components of a Convolutional Network

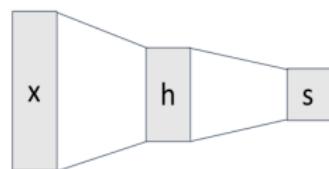
Convolution Layers



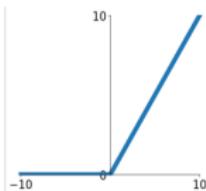
Pooling Layers



Fully-Connected Layers



Activation Function

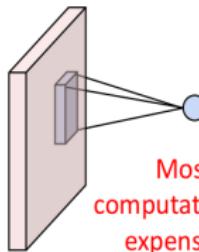


Normalization

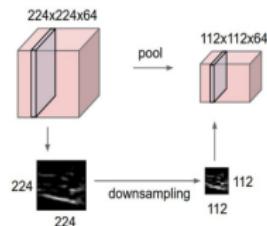
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Components of a Convolutional Network

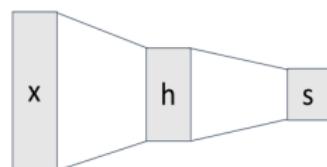
Convolution Layers



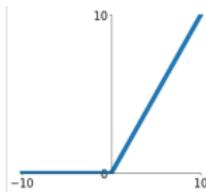
Pooling Layers



Fully-Connected Layers



Activation Function

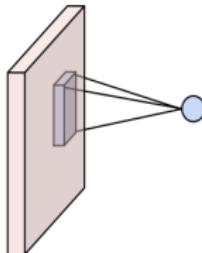


Normalization

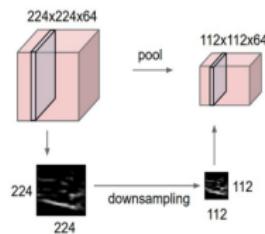
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Summary: Components of a Convolutional Network

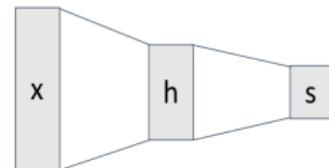
Convolution Layers



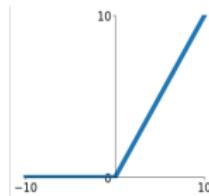
Pooling Layers



Fully-Connected Layers



Activation Function

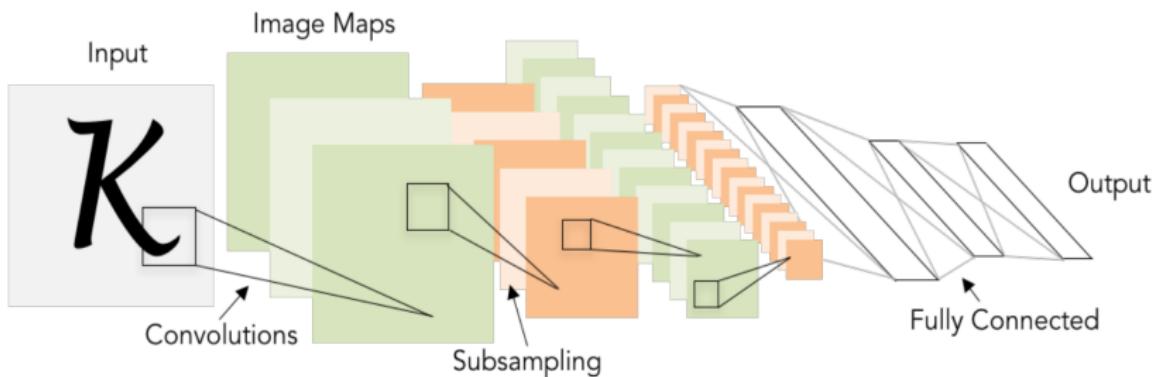


Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Summary: Components of a Convolutional Network

Problem: What is the right way to combine all these components?



Next time:
CNN Architectures

Thank You!

Any questions?