ROB 498/599: Deep Learning for Robot Perception (DeepRob)

Lecture 7: Convolutional Networks (components)

02/03/2025

https://deeprob.org/w25/

ROBOTICS

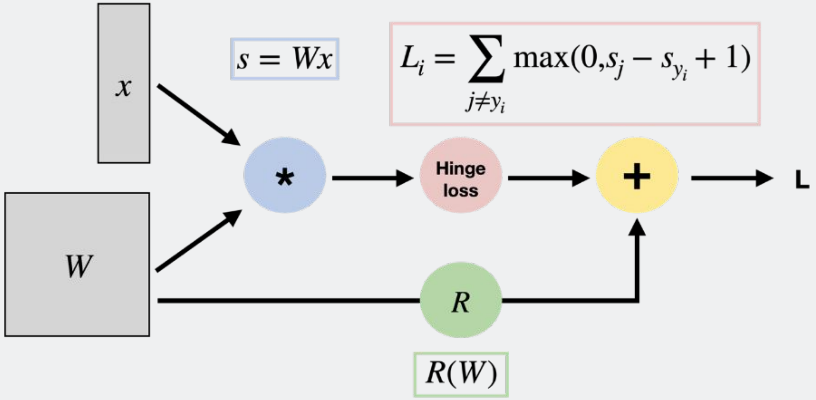
Today

- Feedback and Recap (5min)
- Five Components of Convolutional Networks
  - Fully connected Layers and Convolution Layer (15min)
  - Spatial Dimensions (20min)
  - Pooling Layer (15min)
  - Batch Normalization (15min)
- Summary and Takeaways (5min)

Recap

P2 released, due Feb. 16, 2025

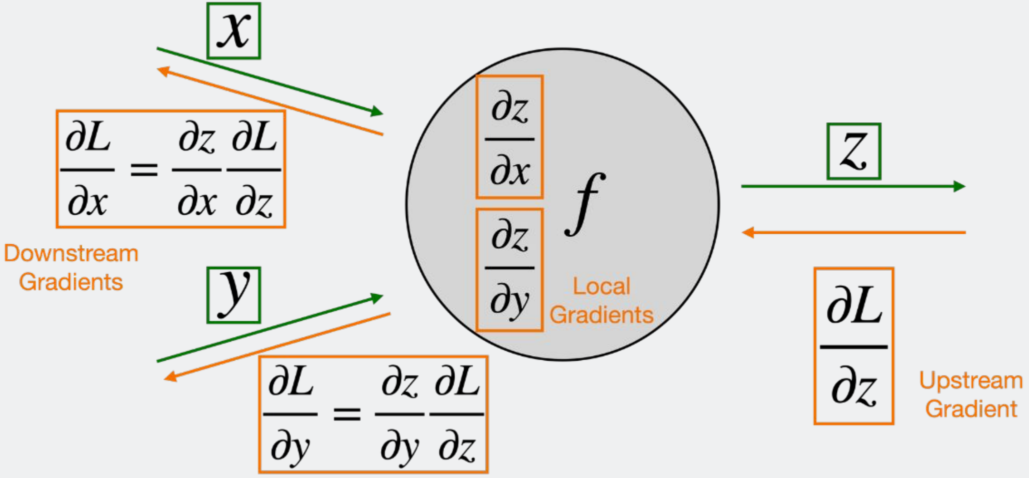
Represent complex expressions as **computational graphs**



1. **Forward pass**: Compute outputs

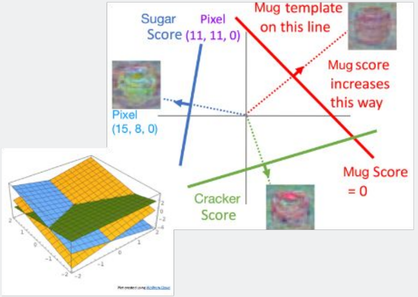
2. **Backward pass**: Compute gradients

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

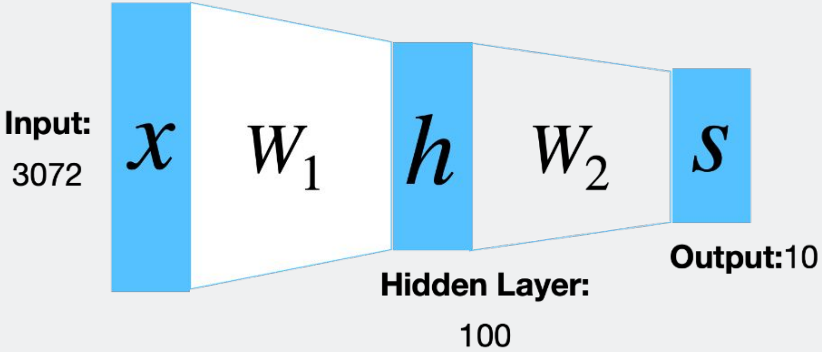


ROBOTICS

Recap

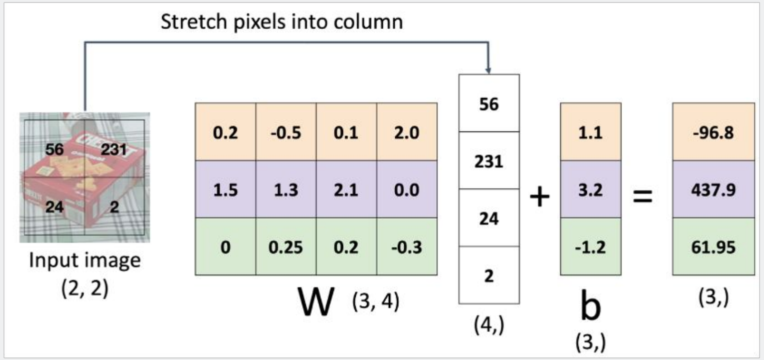


$$f(x) = W_2 \max(0, W_1 x + b_1) + b_2$$



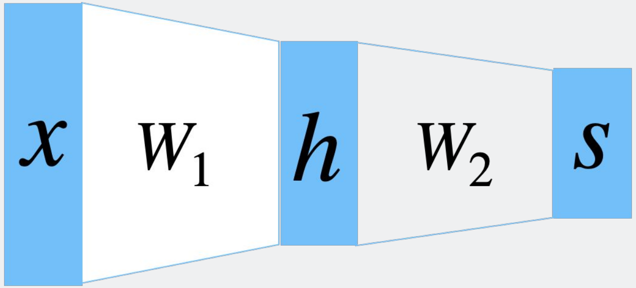
Problem: So far our classifiers don't respect the spatial structure of images!

Solution: Define new computational nodes that operate on images!

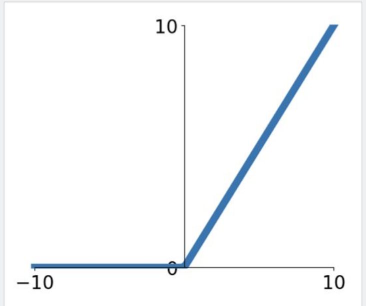


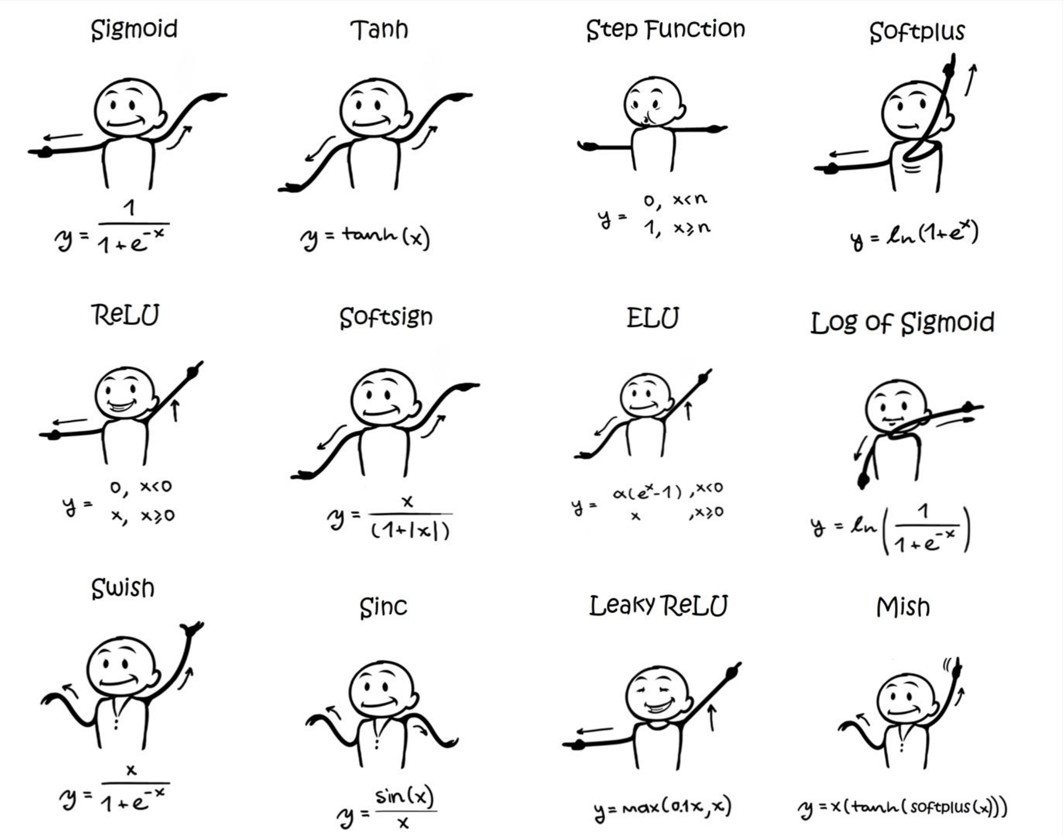
Components of Fully Connected Networks

Fully-Connected Layers



Activation Functions

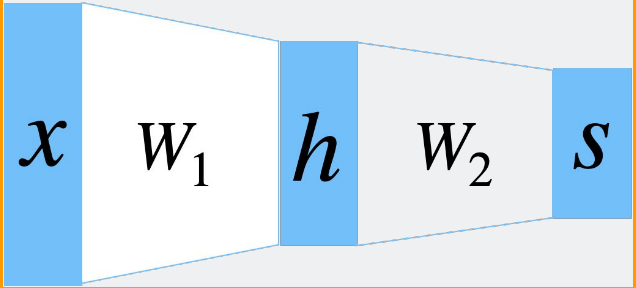




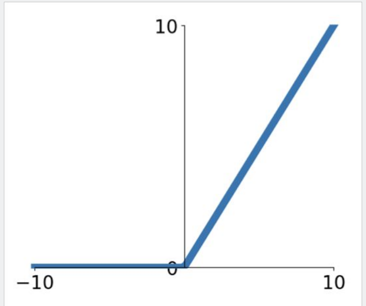
"Dance Moves of Deep Learning Activation Functions"
https://sefiks.com/2020/02/02/dance-moves-of-deep-learning-activation-functions/

Components of Fully Connected Networks

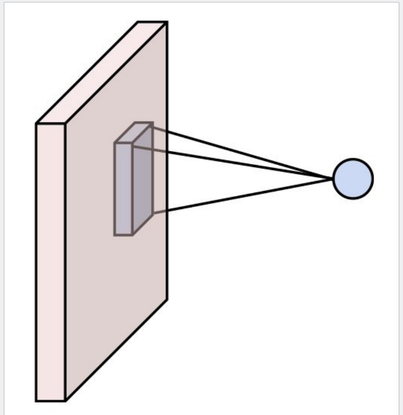
Fully-Connected Layers



Activation Functions

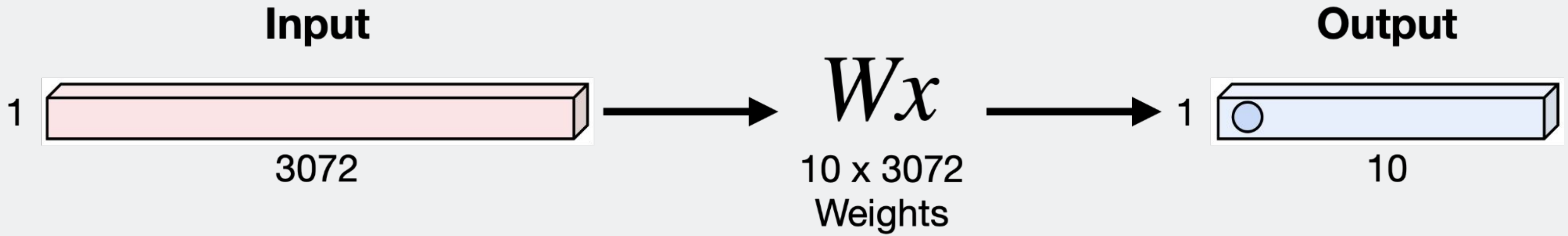


Convolution Layers



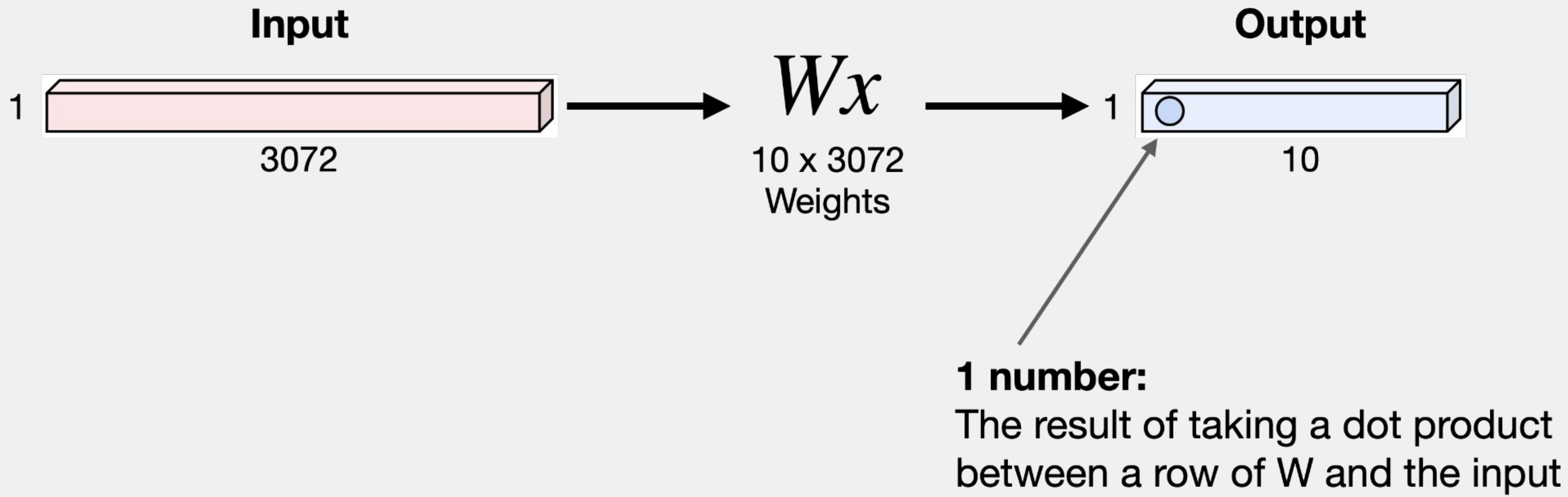
Fully Connected Layer

3x32x32 image → stretch to 3072x1



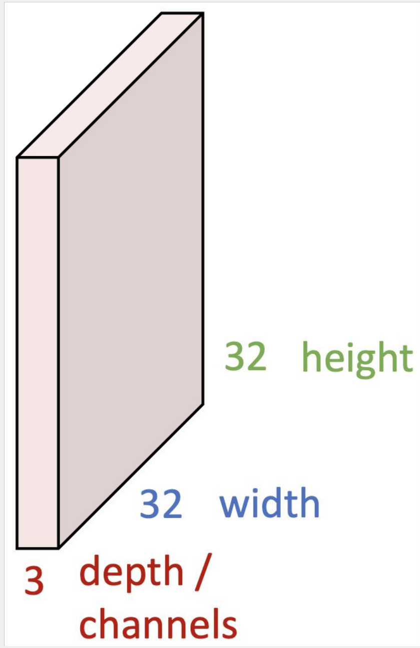
Fully Connected Layer

3x32x32 image → stretch to 3072x1

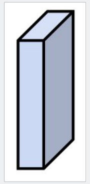


Convolution Layer

3x32x32 image: preserve spatial structure



3x5x5 filter



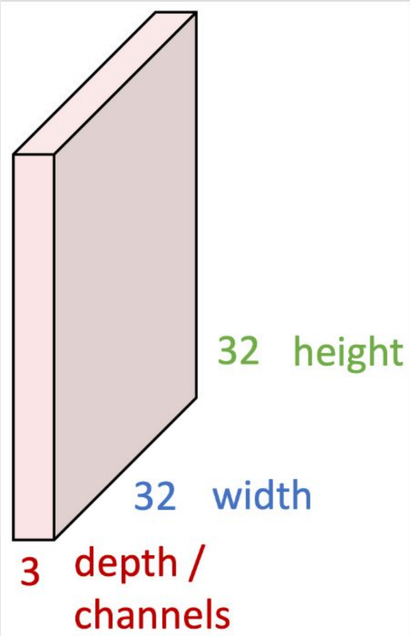
“learnable”

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

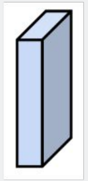
Convolution Layer

3x32x32 image

Filters always extend the full depth of the input volume



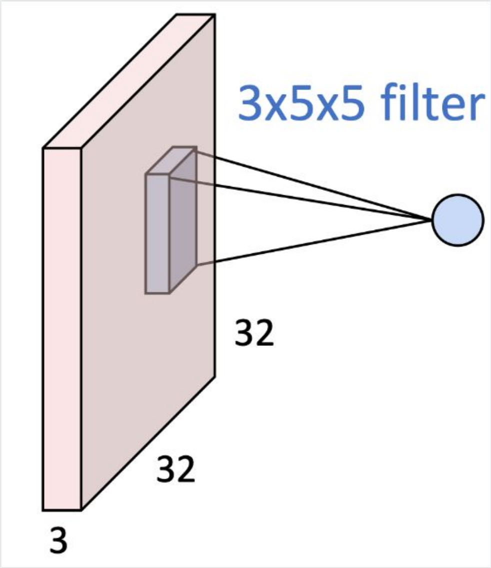
3x5x5 filter



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

Convolution Layer

3x32x32 image



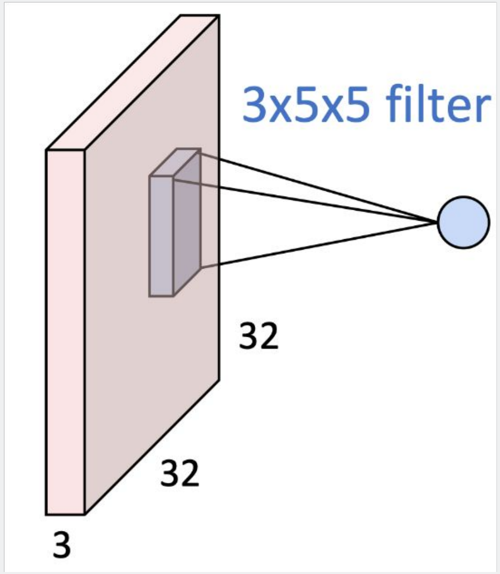
1 number:
The result of taking a dot product between the filter and a small 3x5x5 portion of the image (i.e. 3*5*5=75-dimensional dot product + bias)

$$w^T x + b$$

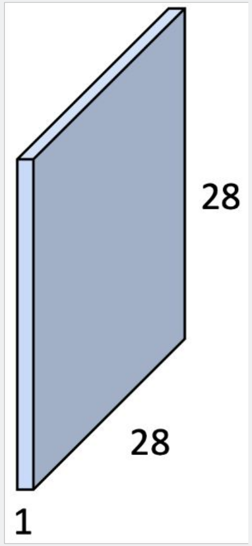
Convolution Layer

3x32x32 image

1x28x28 activation map



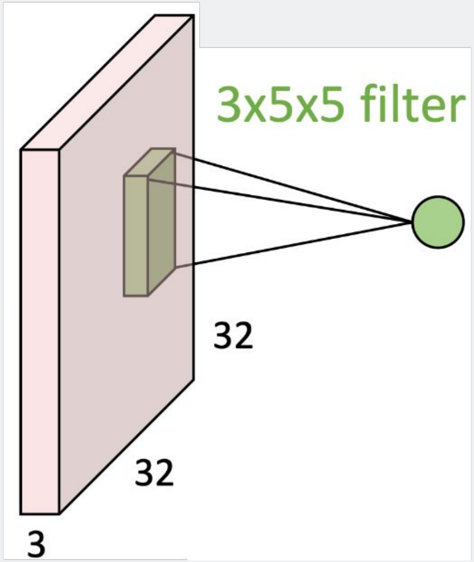
convolve (slide) over all spatial locations



Convolution Layer

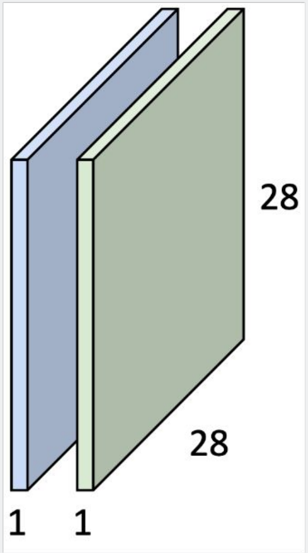
3x32x32 image

two 1x28x28 activation map



Consider repeating with a second (green) filter

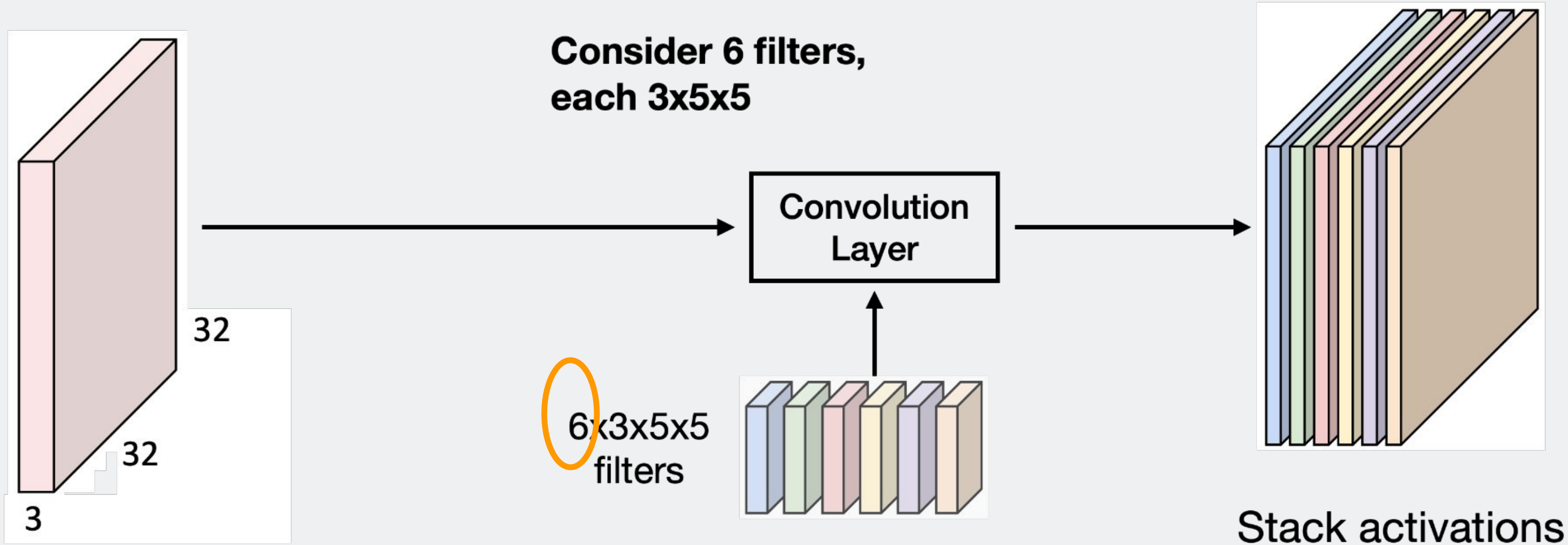
convolve (slide) over all spatial locations



Convolution Layer

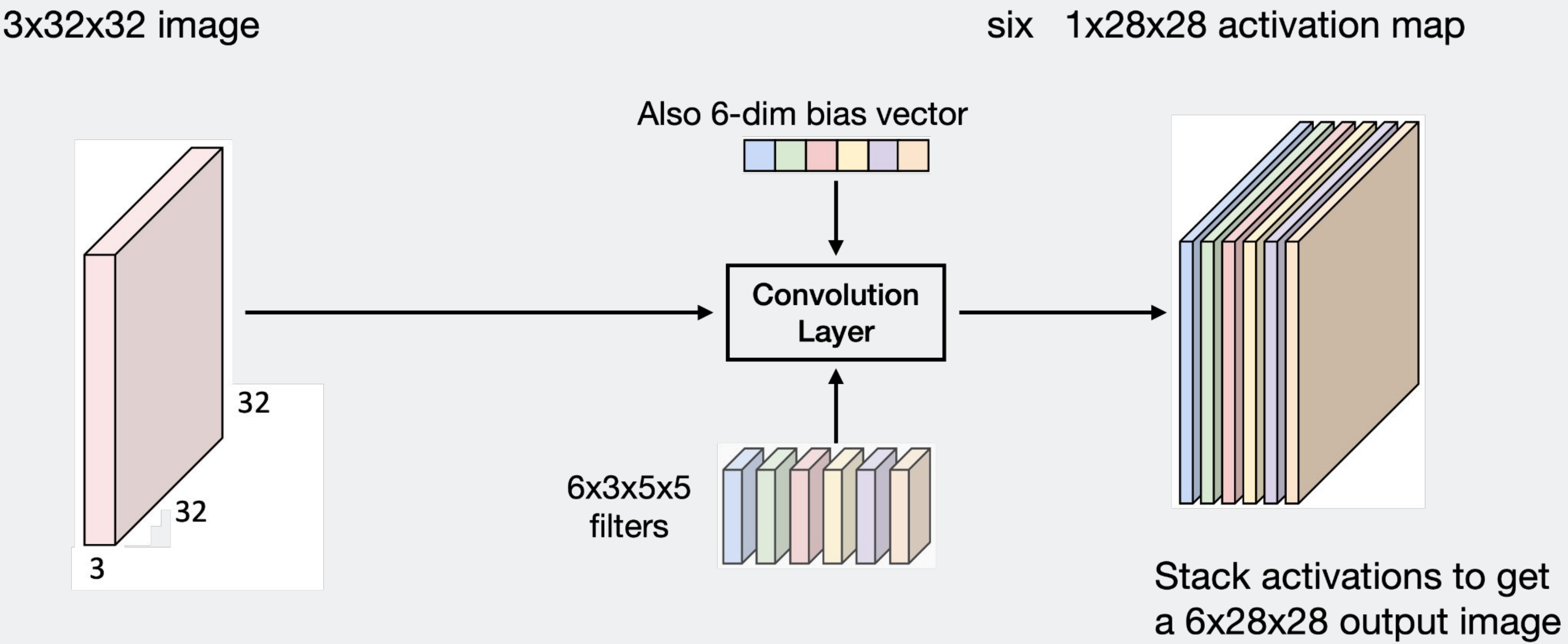
3x32x32 image

six 1x28x28 activation map

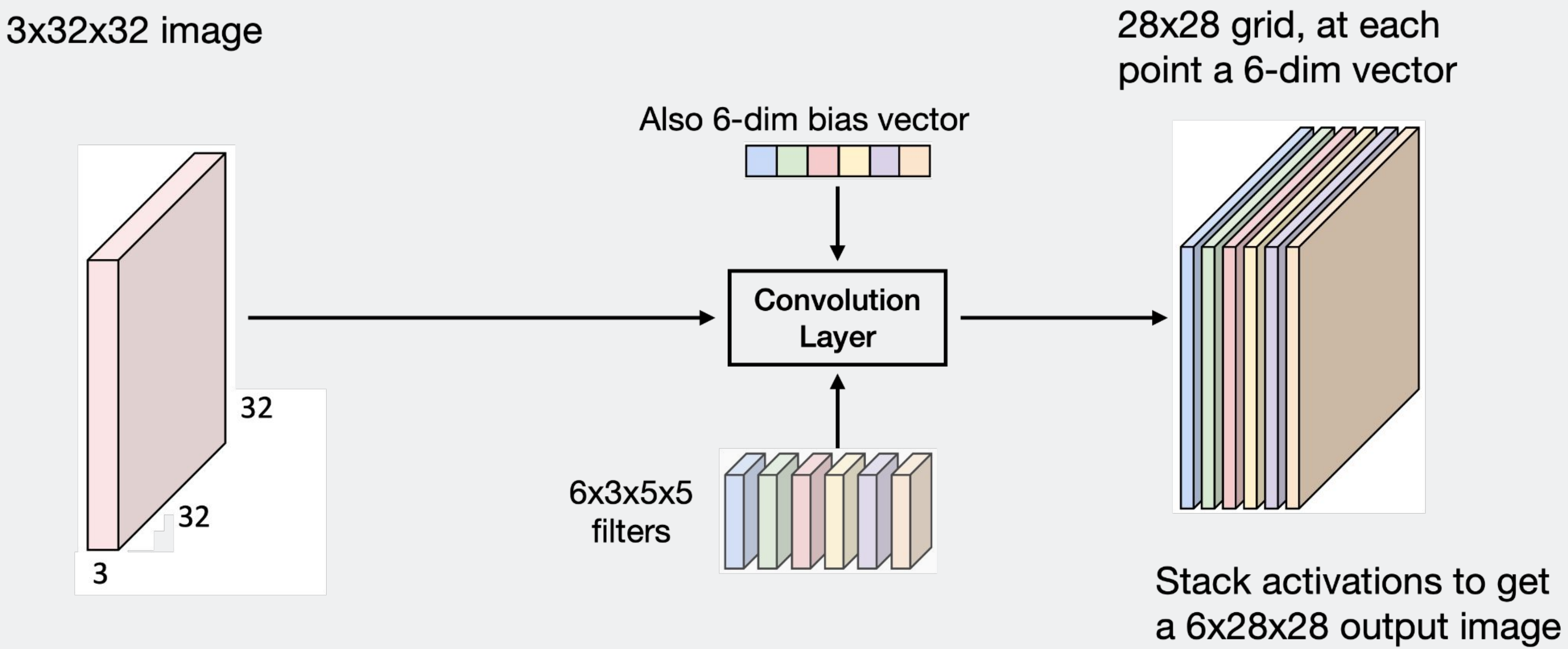


Stack activations to get
a 6x28x28 output image

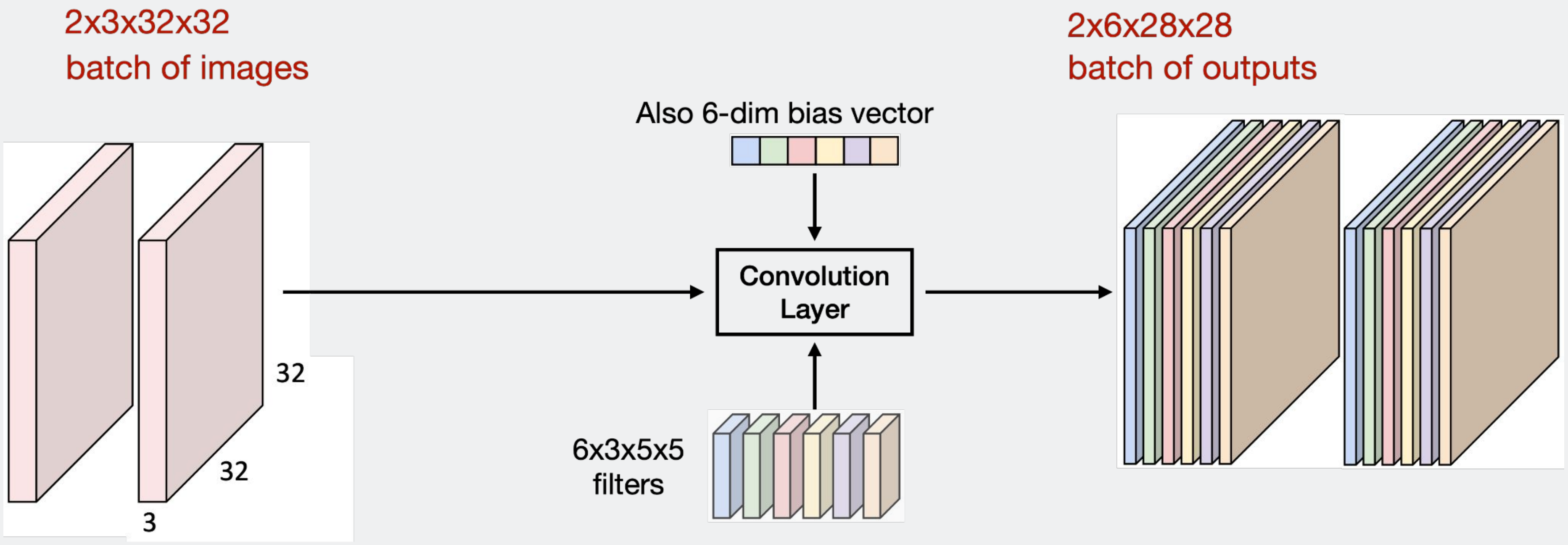
Convolution Layer



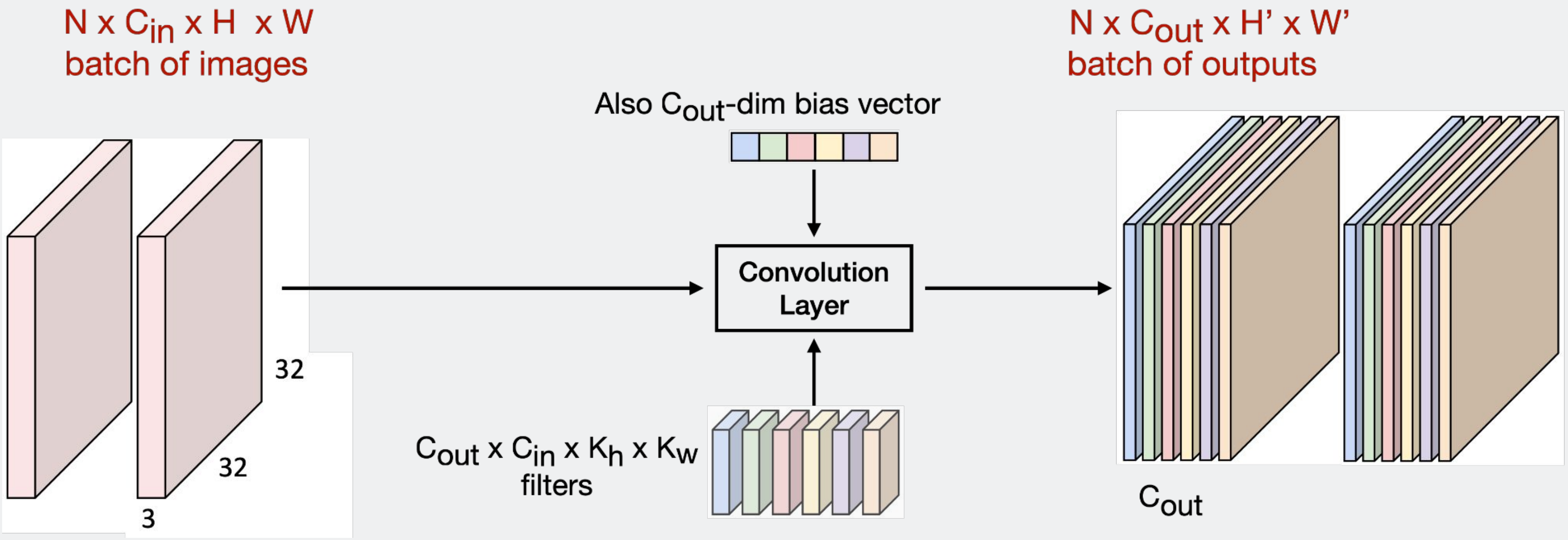
Convolution Layer



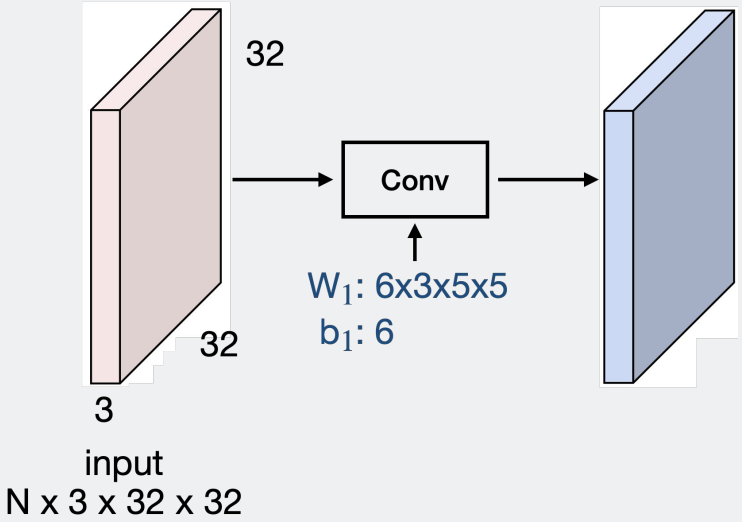
Convolution Layer



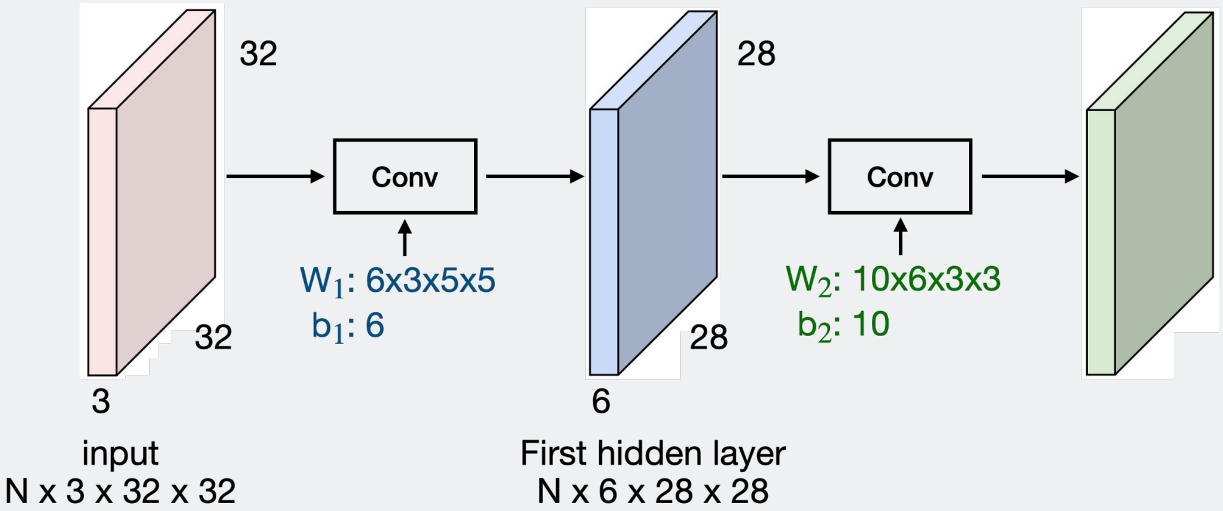
Convolution Layer



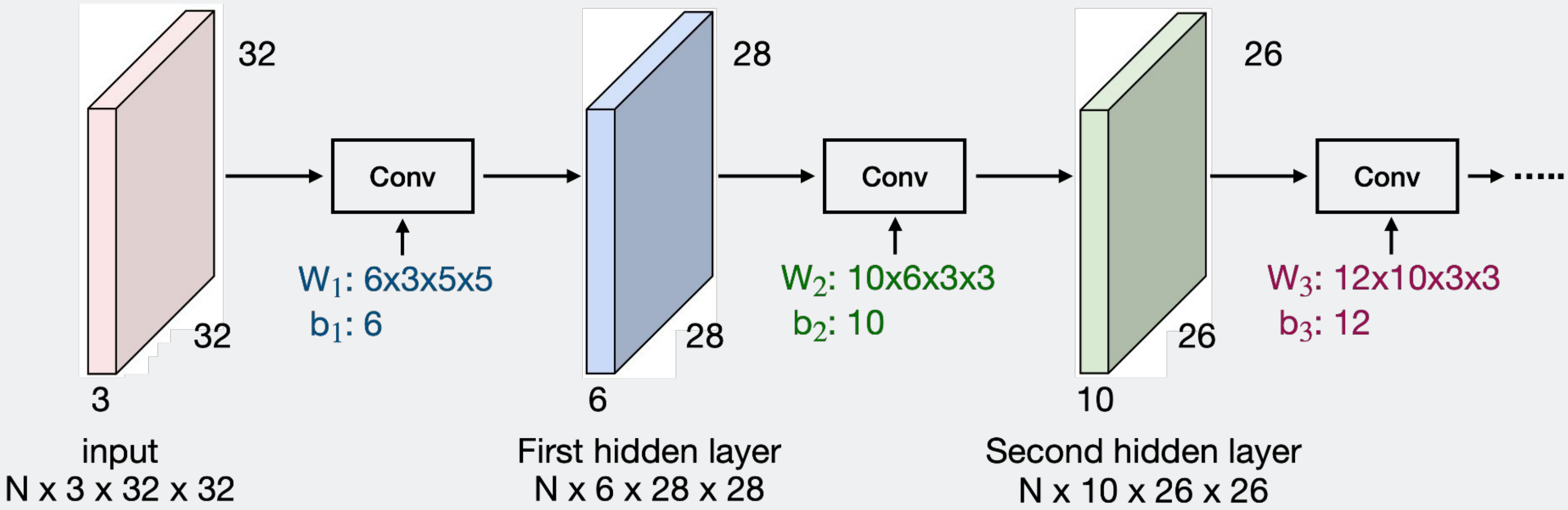
Stacking Convolutions



Stacking Convolutions

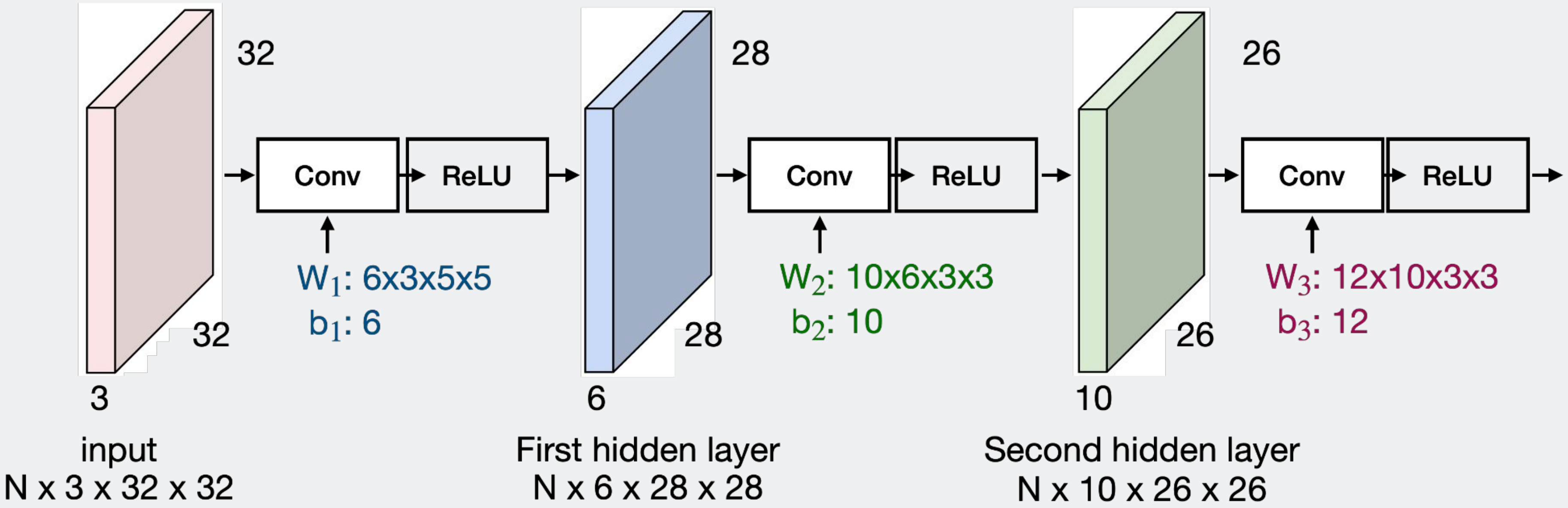


Stacking Convolutions



Q: What happens if we stack two convolution layers?

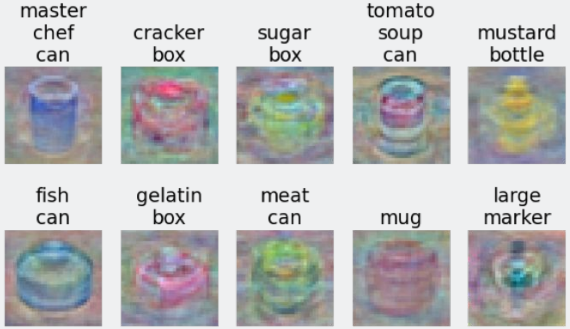
Stacking Convolutions

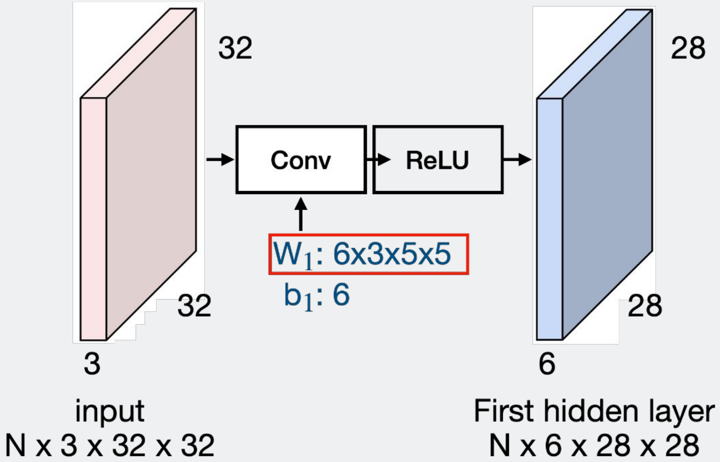


ROBOTICS

What do convolutions filters learn?

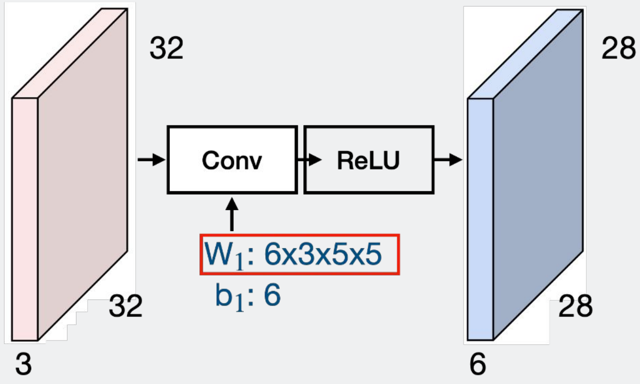
Linear classifier: One template per class





MLP: Bank of whole-image templates

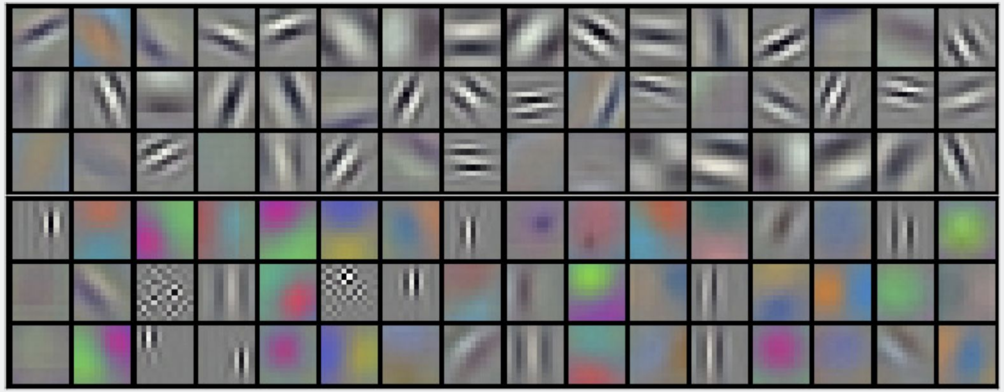
What do convolutions filters learn?



input
N x 3 x 32 x 32

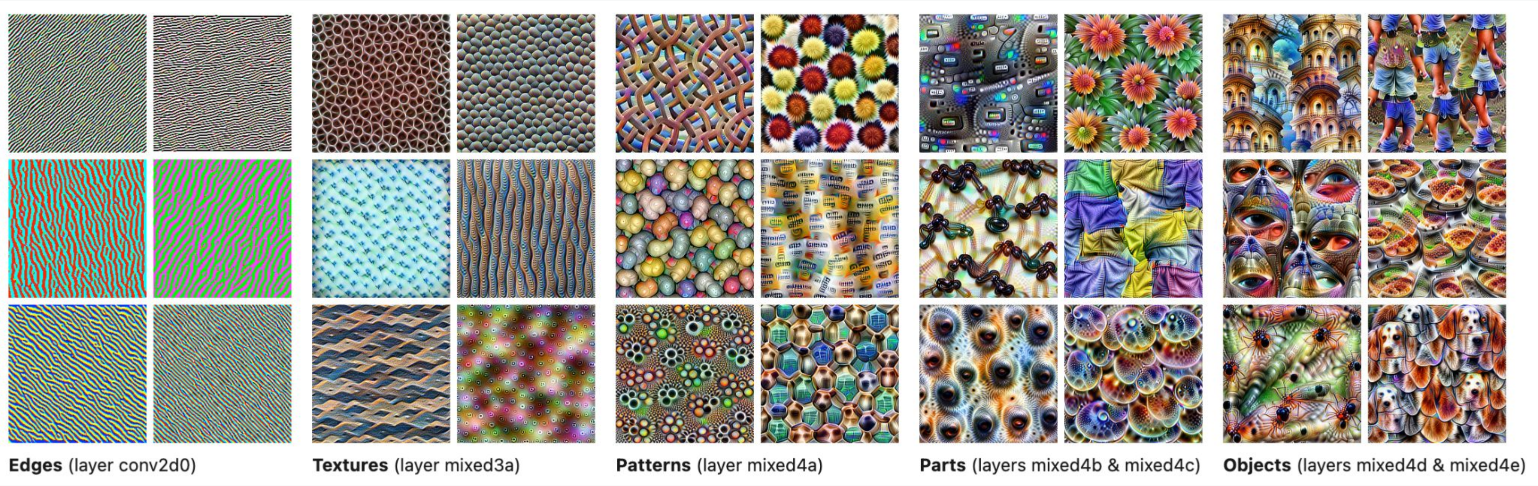
First hidden layer
N x 6 x 28 x 28

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

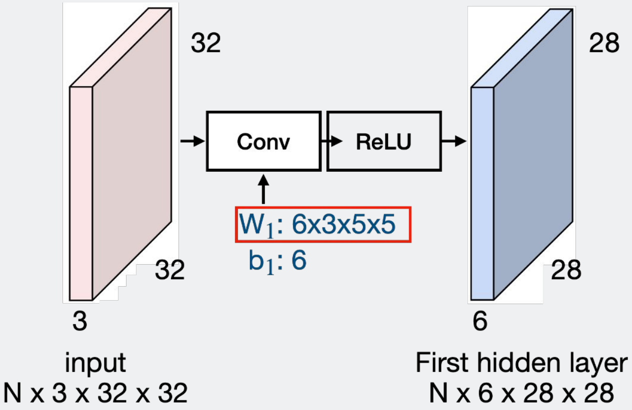


AlexNet: 96 filters, each 3x11x11

What do convolutions filters learn?

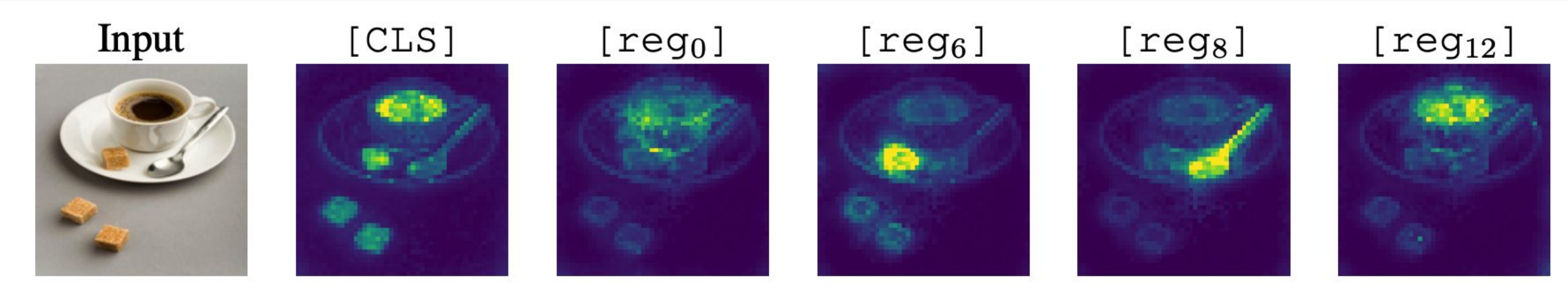


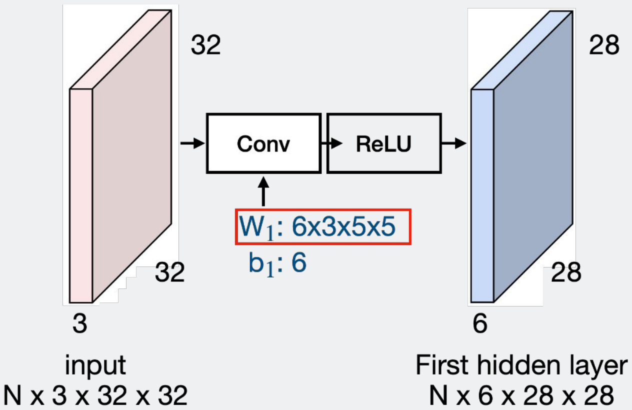
Feature visualization (2017)



Olah, et al., "Feature Visualization", Distill pub, 2017.
https://distill.pub/2017/feature-visualization/

What do ~~convolutions filters~~ vision transformers learn?





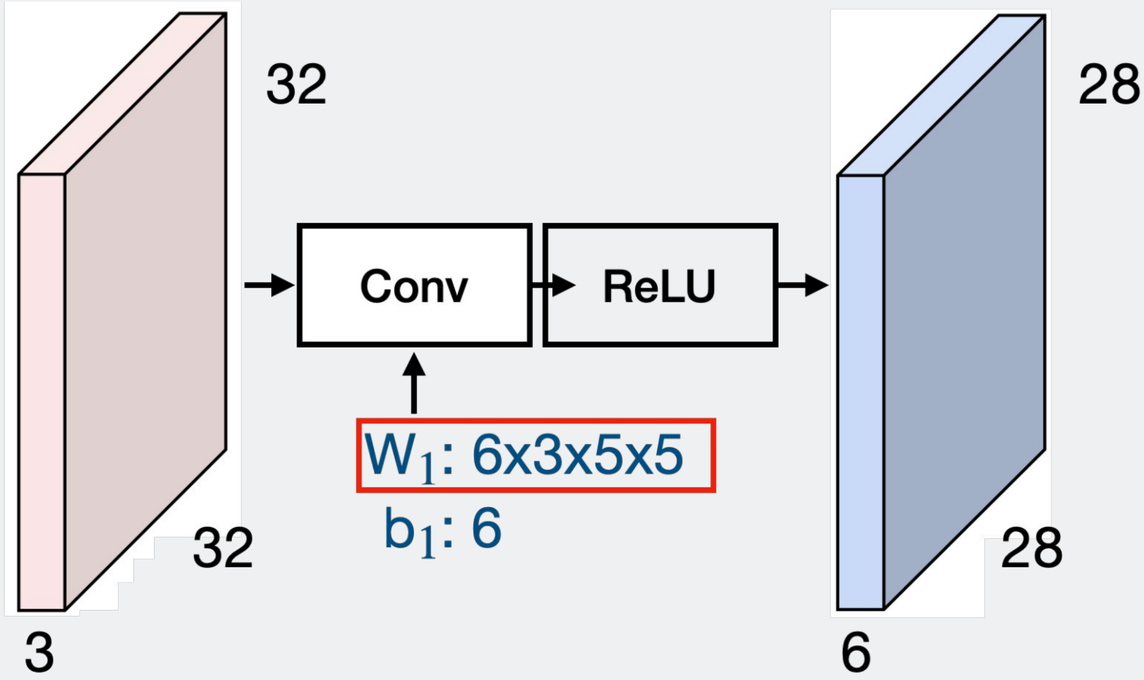
Interpretable Attention Maps (2014)

Darcet et al., Vision Transformers Need Registers (2024)
https://arxiv.org/abs/2309.16588 (Accepted ICLR 2024)

(more on transformers later)

ROBOTICS

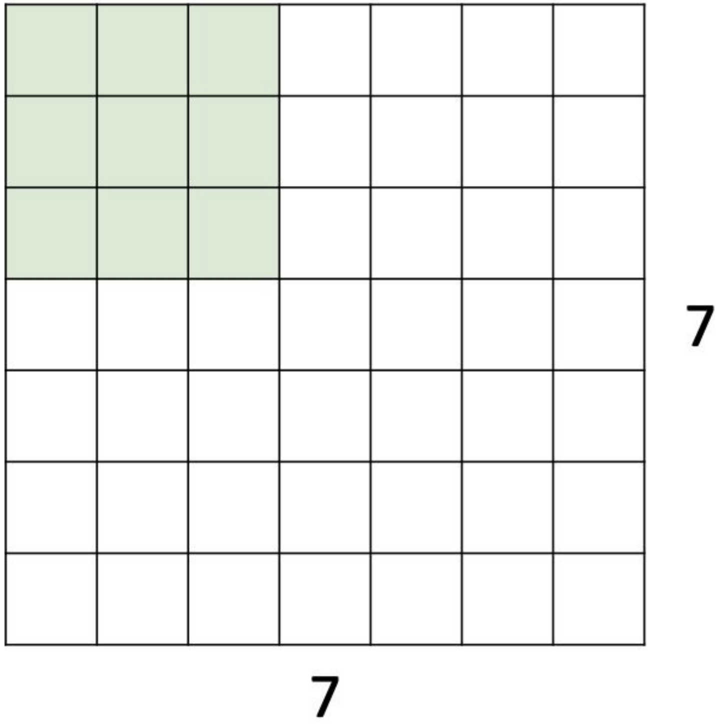
A closer look at the spatial dimensions



input
N x 3 x 32 x 32

First hidden layer
N x 6 x 28 x 28

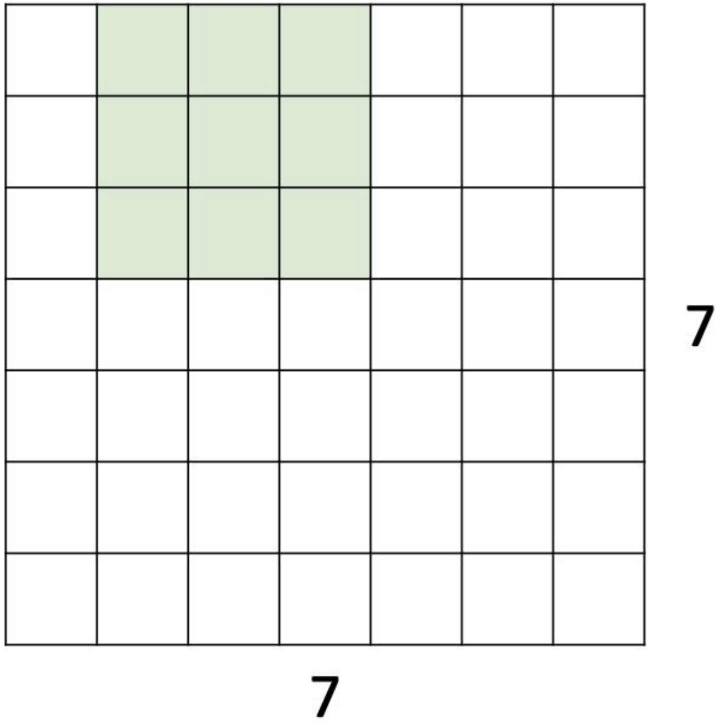
A closer look at the spatial dimensions



Input: 7x7

Filter: 3x3

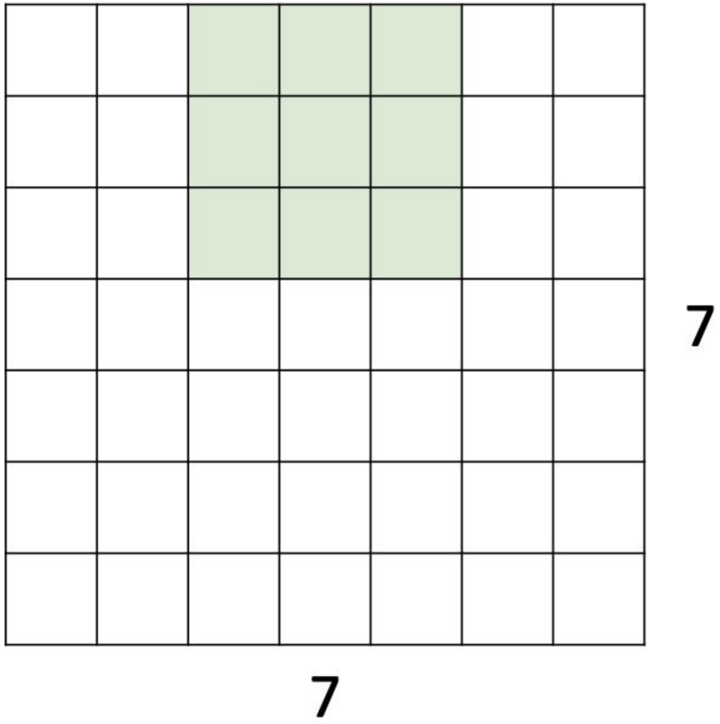
A closer look at the spatial dimensions



Input: 7x7

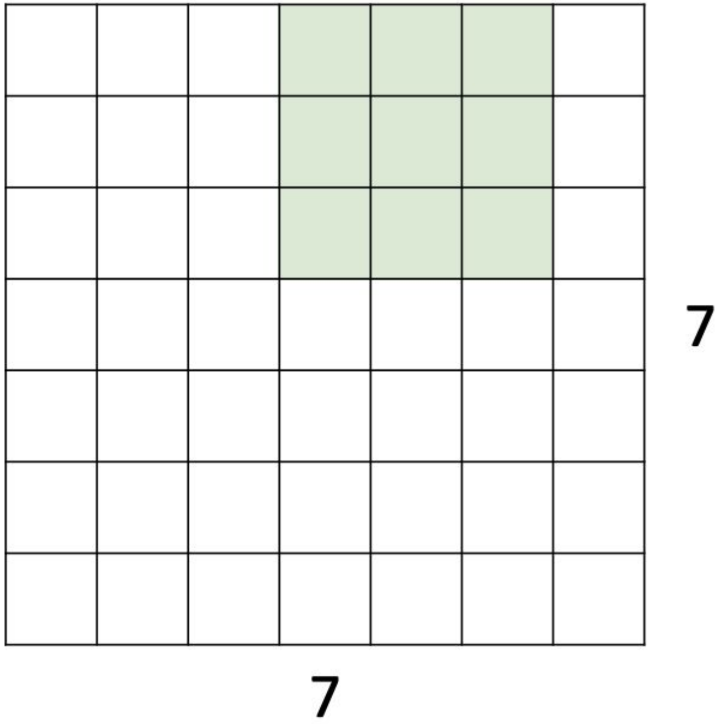
Filter: 3x3

A closer look at the spatial dimensions



Input: 7x7
Filter: 3x3

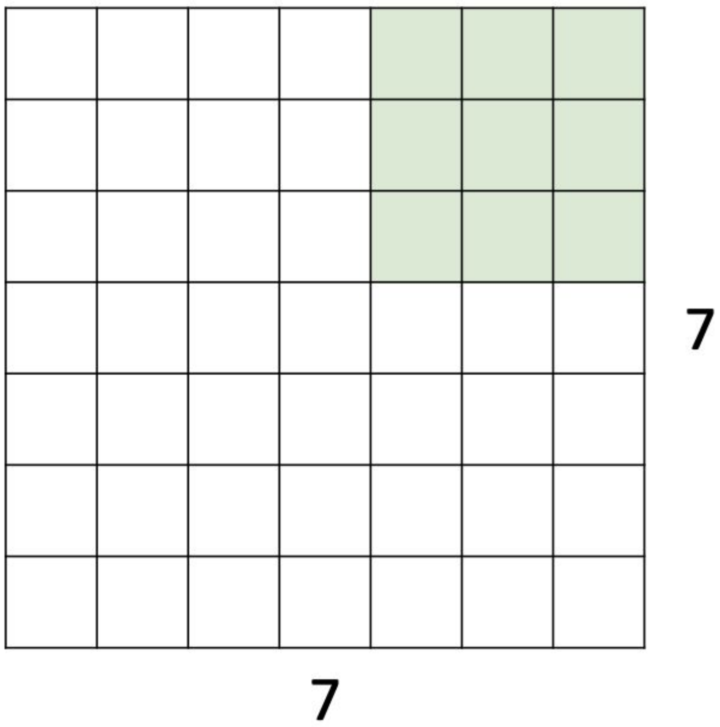
A closer look at the spatial dimensions



Input: 7x7

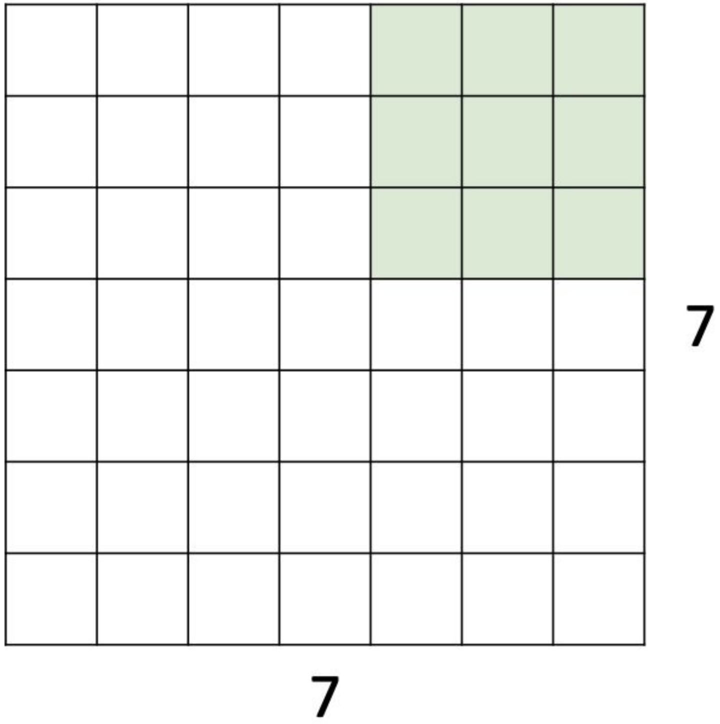
Filter: 3x3

A closer look at the spatial dimensions



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

A closer look at the 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 the 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:
Input: W
Filter: K
Output: W – K + 1

Problem: Feature maps "shrink" with each layer!

Solution: **padding**
Add zeros around the input

A closer look at the 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

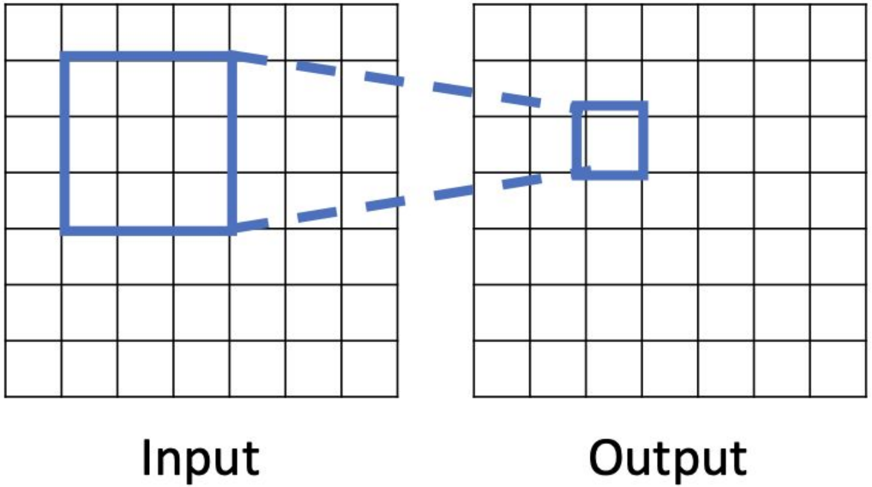
In general:
Input: W
Filter: K
Padding: P
Output: W – K + 1 + 2P

Very common:
Set P = (K – 1) / 2 to make output have same size as input!

Receptive Fields

Receptive Fields

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



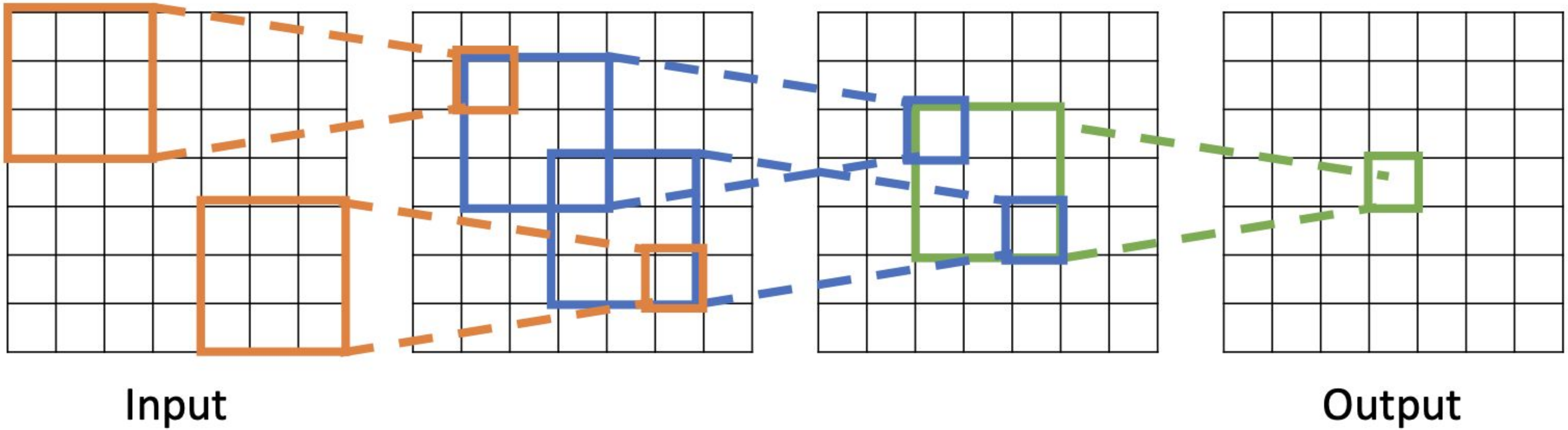
Formally, it is the region in the input space that a particular CNN's feature is affected by.

Informally, it is the part of a tensor that after convolution results in a feature.

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)

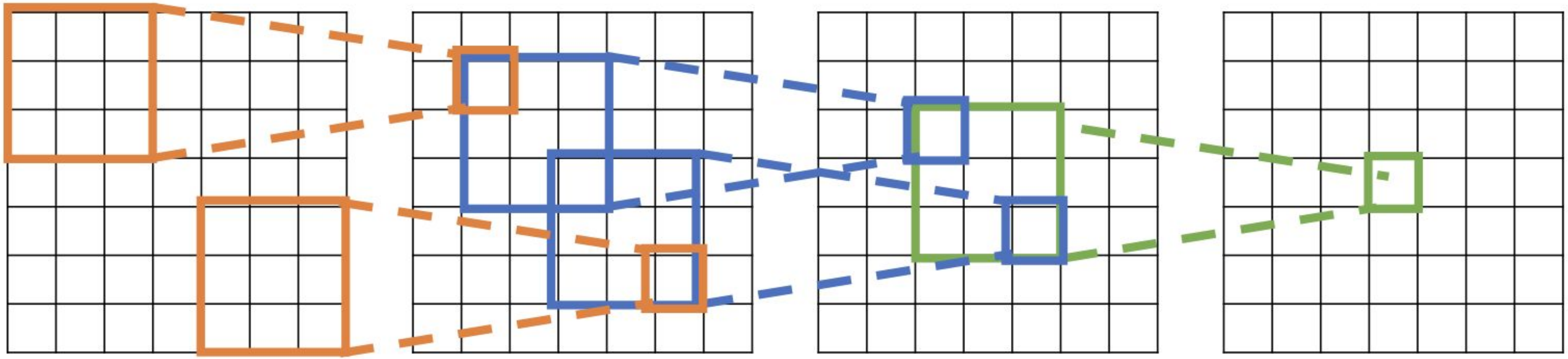


Be careful – "receptive field in the input" vs "receptive field in the previous layer"

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)

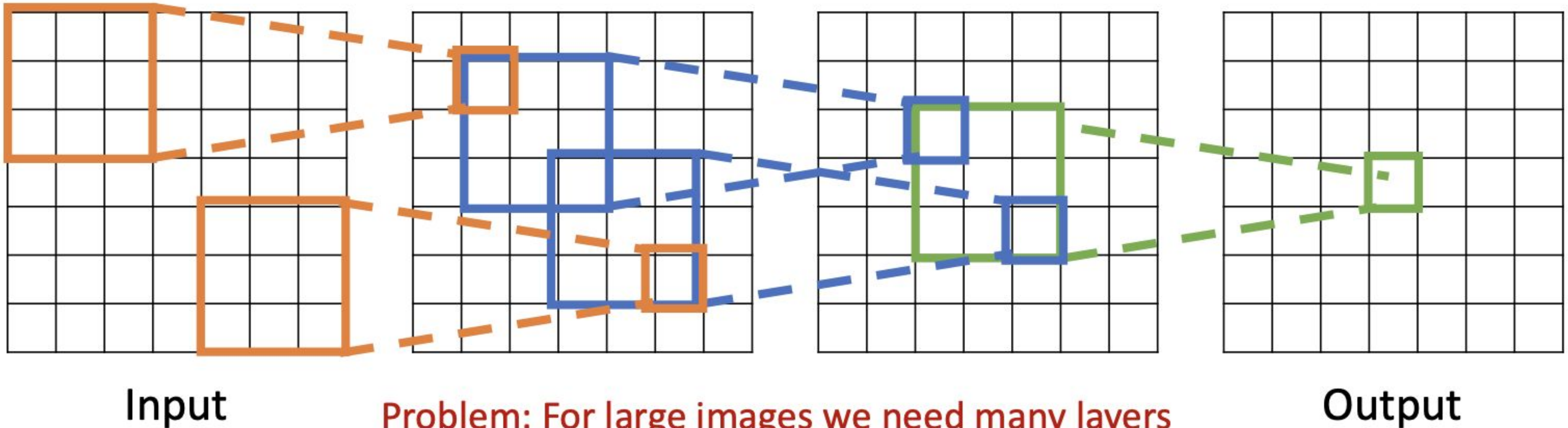


Problem: For large images we need many layers for each output to "see" the whole image image

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)

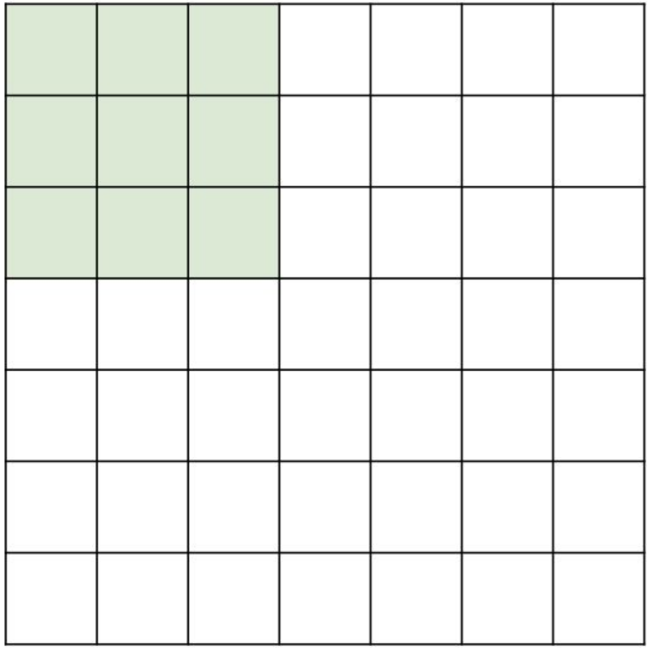


Problem: For large images we need many layers for each output to "see" the whole image image

Solution: Downsample inside the network

Strided Convolution

https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html

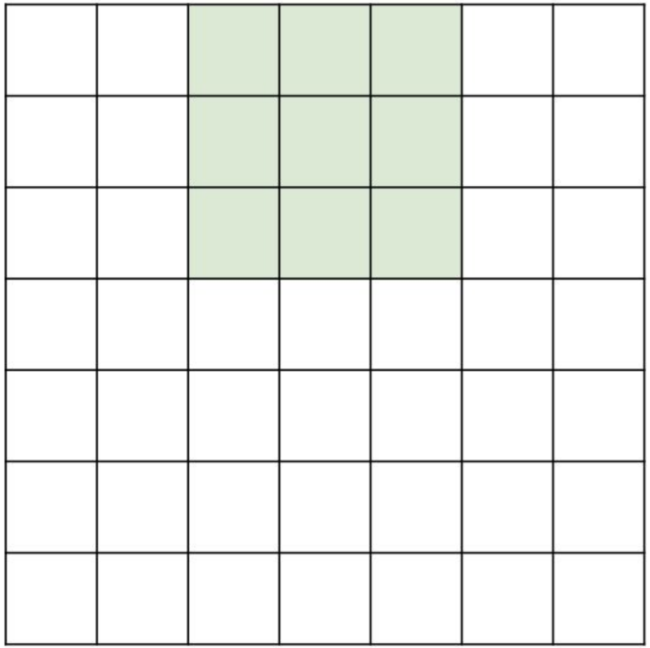


Input: 7x7

Filter: 3x3

Stride: 2

Strided Convolution

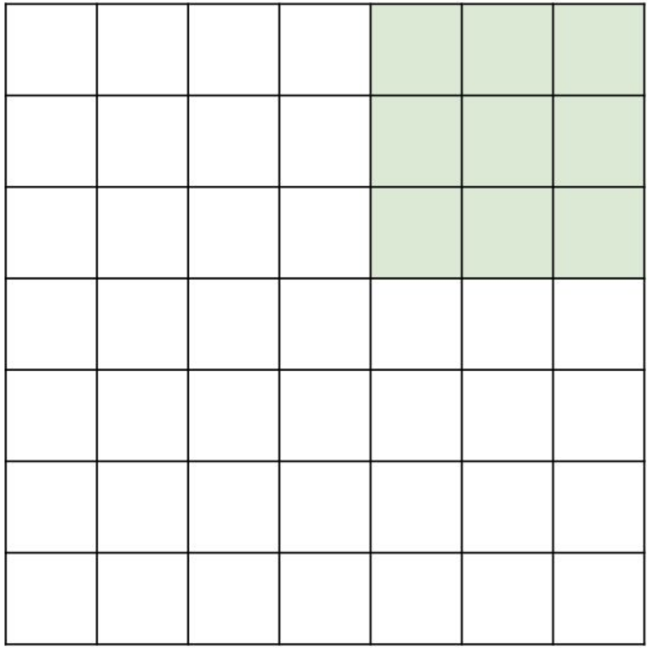


Input: 7x7

Filter: 3x3

Stride: 2

Strided Convolution



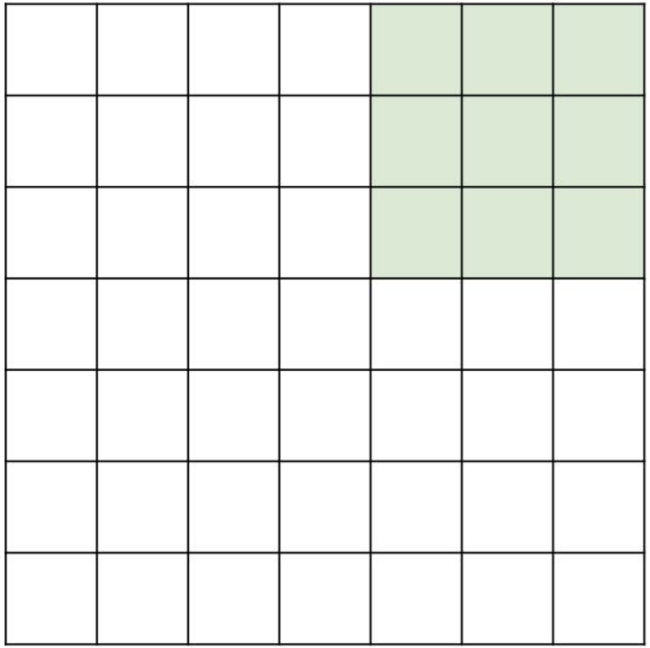
Input: 7x7

Filter: 3x3

Stride: 2

Output: 3x3

Strided Convolution



Input: 7x7

Filter: 3x3

Stride: 2

Output: 3x3

In general:

Input: W

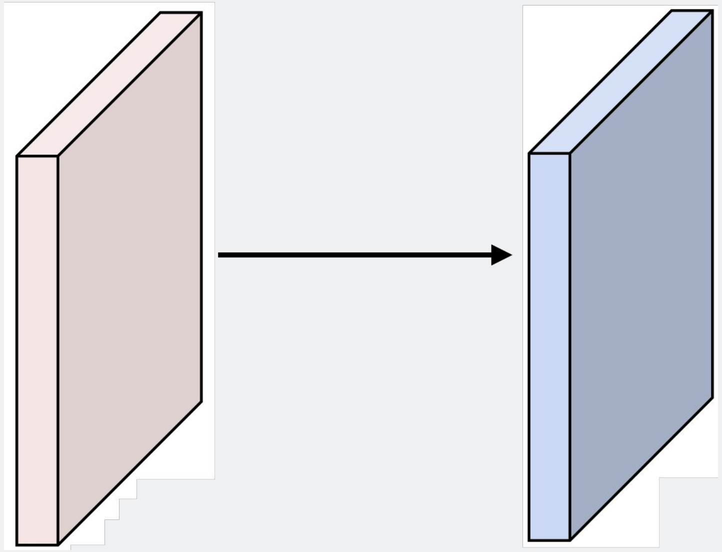
Filter: K

Padding: P

Stride: S

Output: (W – K + 2P) / S + 1

Convolution Example



Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

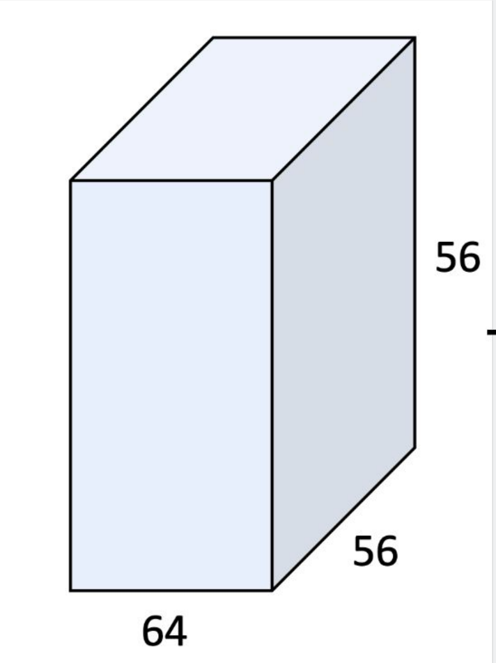
Q1: What is the output volume size?
Q2: What is the number of learnable parameters?
Q3: What is the number of multiply-add operations?

https://ahaslides.com/D5HXR

Aha Slides
(In-class participation)

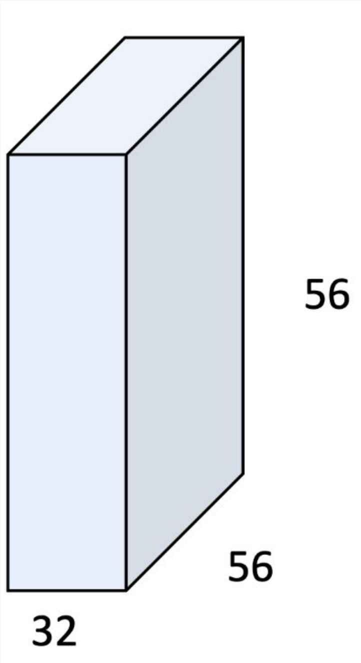
https://ahaslides.com/D5HXR

Example: 1x1 Convolution



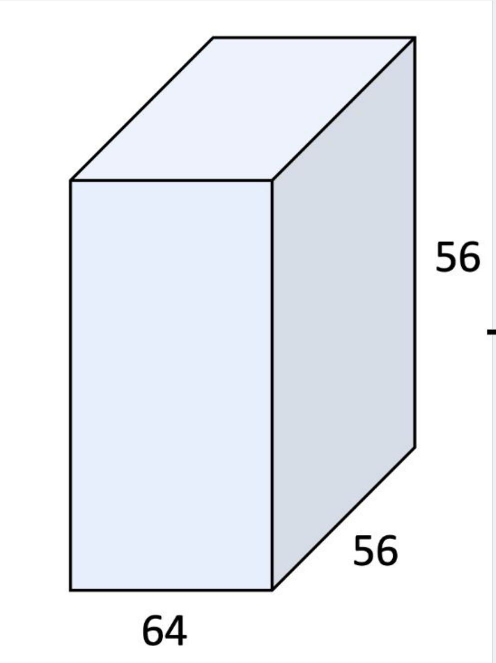
1x1 Conv
with 32 filters

Each filter has size 1x1x64 and
performs a 64-dimensional dot product



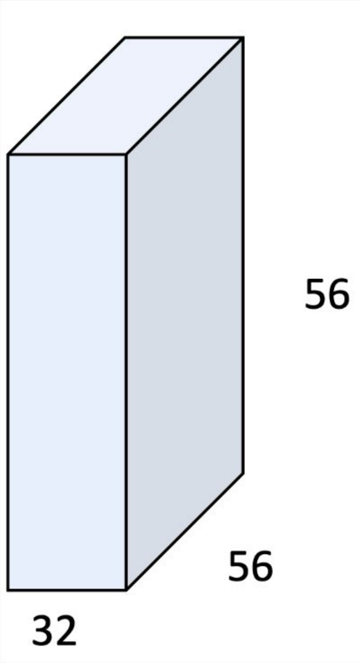
Lin et al., “Network in Network”, ICLR 2014
https://arxiv.org/abs/1312.4400

Example: 1x1 Convolution



1x1 Conv
with 32 filters

Each filter has size 1x1x64 and
performs a 64-dimensional dot product



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

ROBOTICS

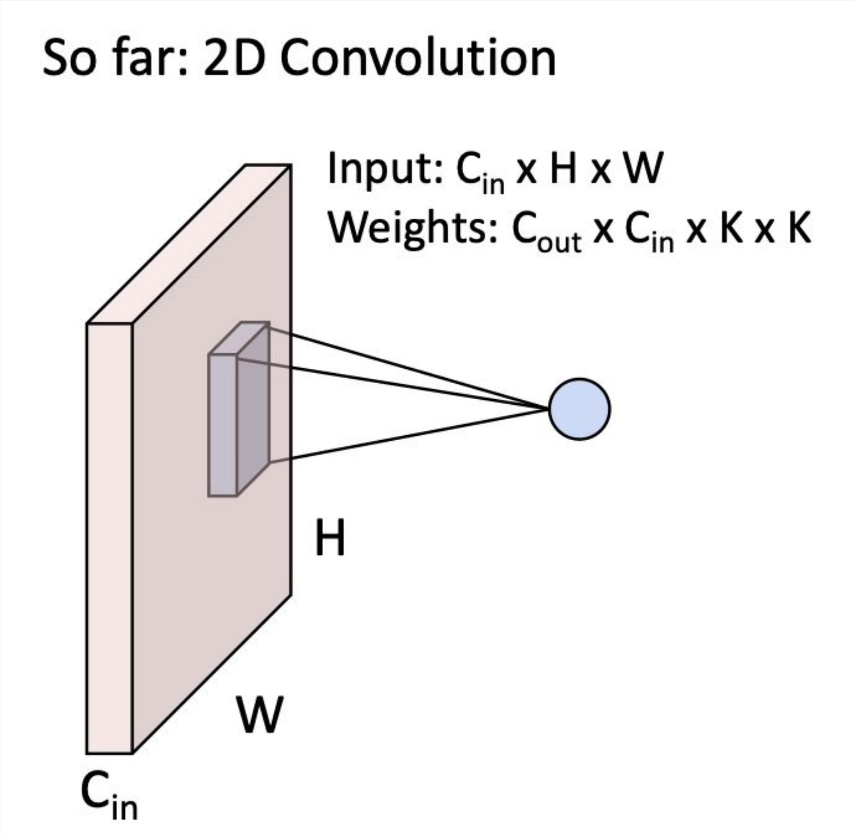
Convolution Summary

Input: C_{in} x H x W
Hyperparameters:
- **Kernel size**: K_H x K_W
- **Number filters**: C_{out}
- **Padding**: P
- **Stride**: S

Weight matrix: C_{out} x C_{in} x K_H x K_W giving C_{out} filters of size C_{in} x K_H x K_W
Bias vector: C_{out}
Output size: C_{out} x H' x 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)

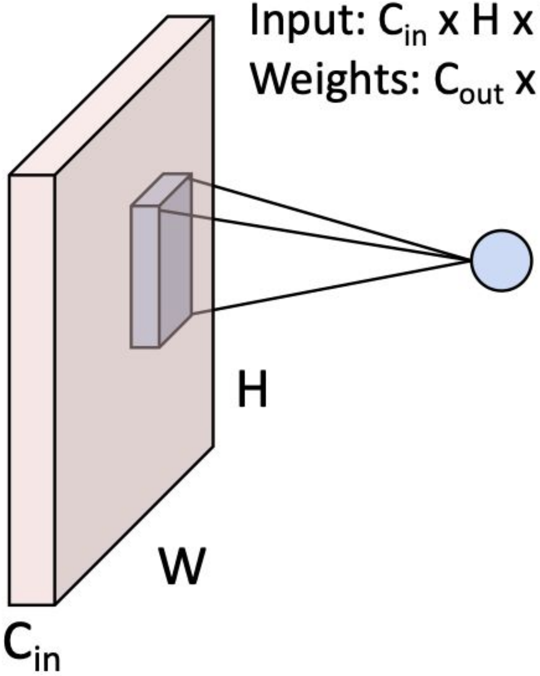
Other types of convolutions



Other types of convolutions

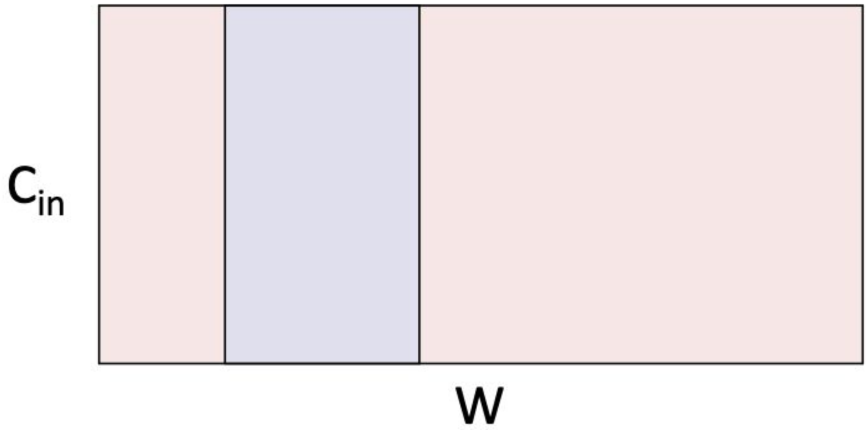
So far: 2D Convolution

Input: C_{in} x H x W
Weights: C_{out} x C_{in} x K x K

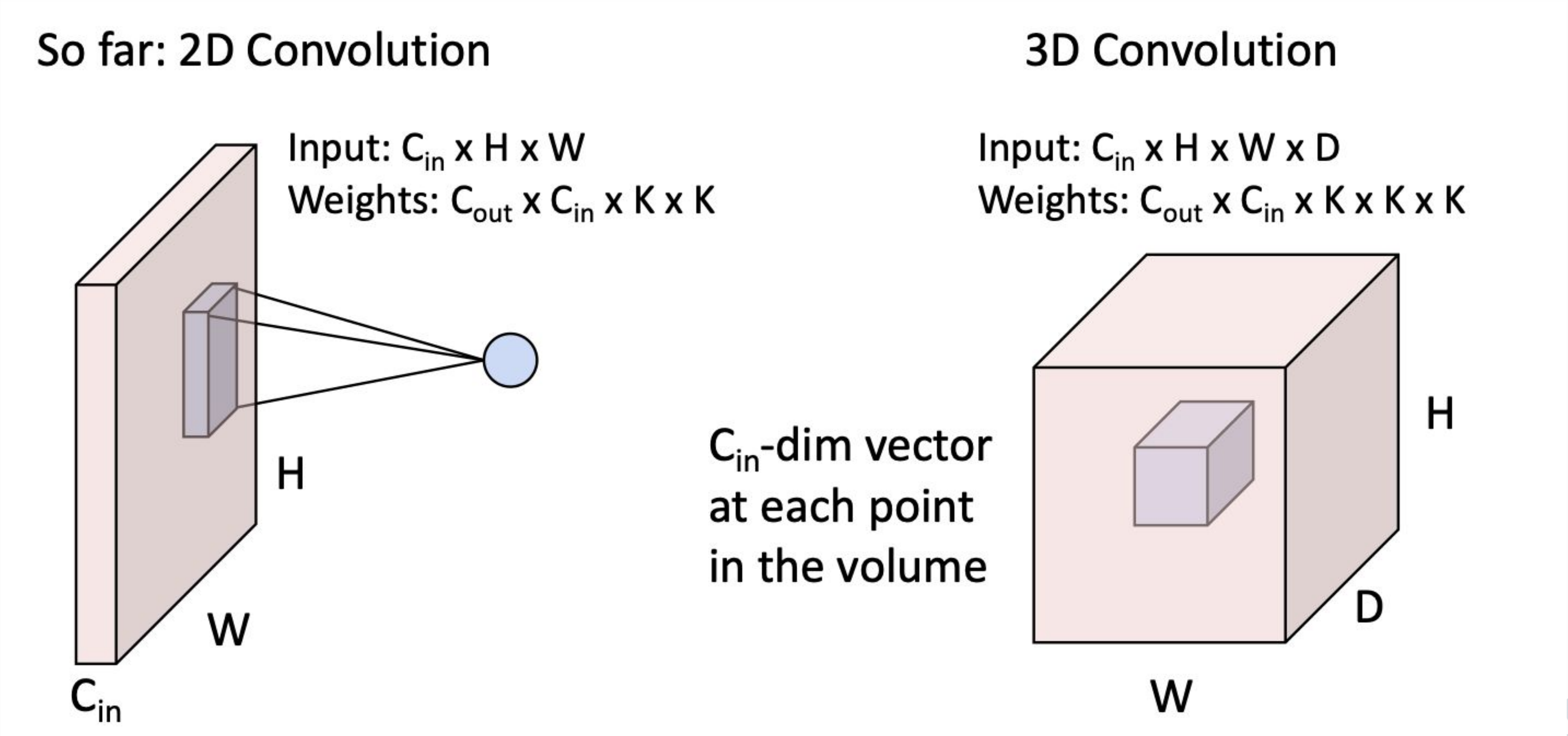


1D Convolution

Input: C_{in} x W
Weights: C_{out} x C_{in} x K



Other types of convolutions



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 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]

Conv1d

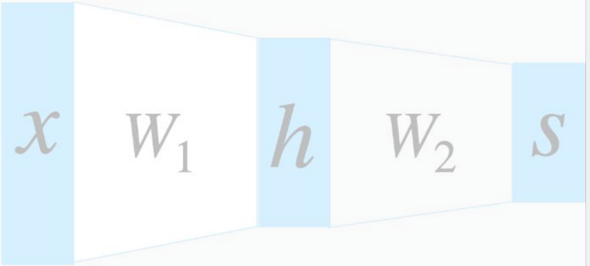
CLASS `torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

Conv3d

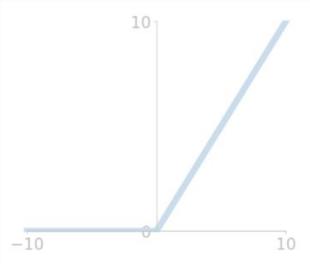
CLASS `torch.nn.Conv3d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

Components of Convolutional Neural Networks

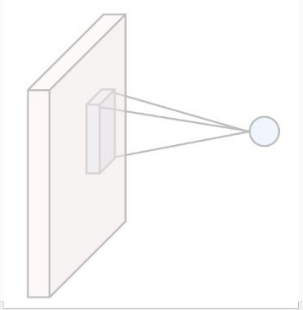
Fully-Connected Layers



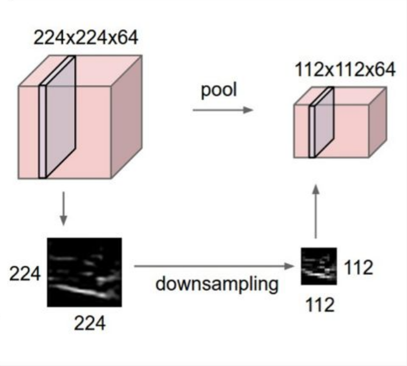
Activation Functions



Convolution Layers



Pooling Layers

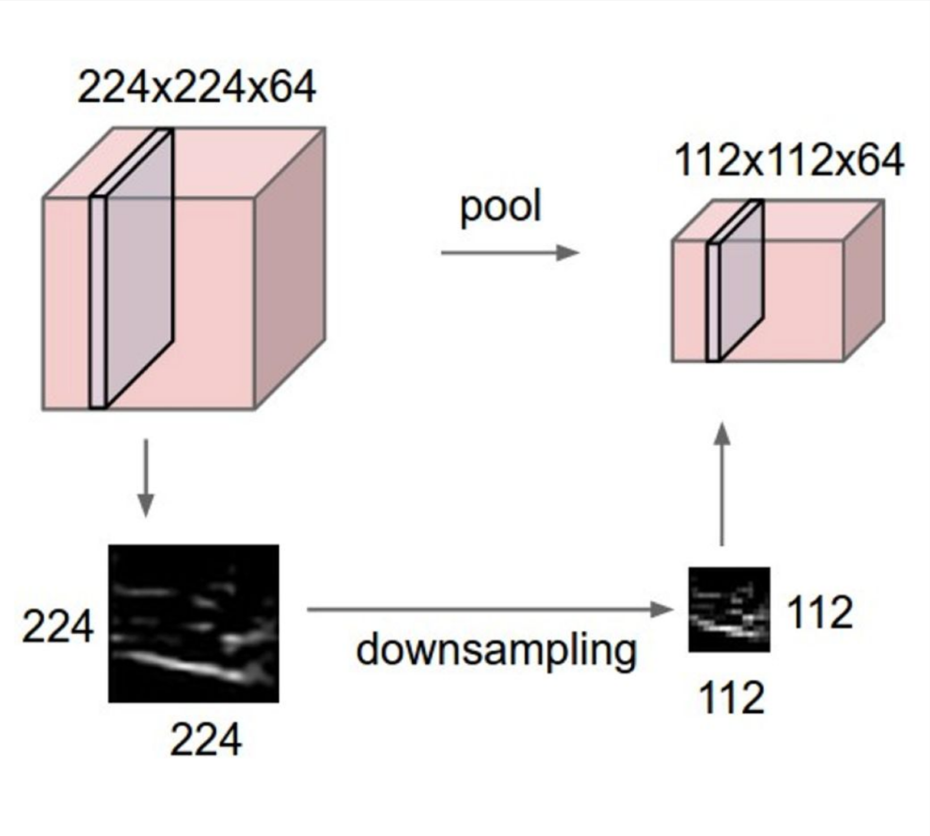


Normalization

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

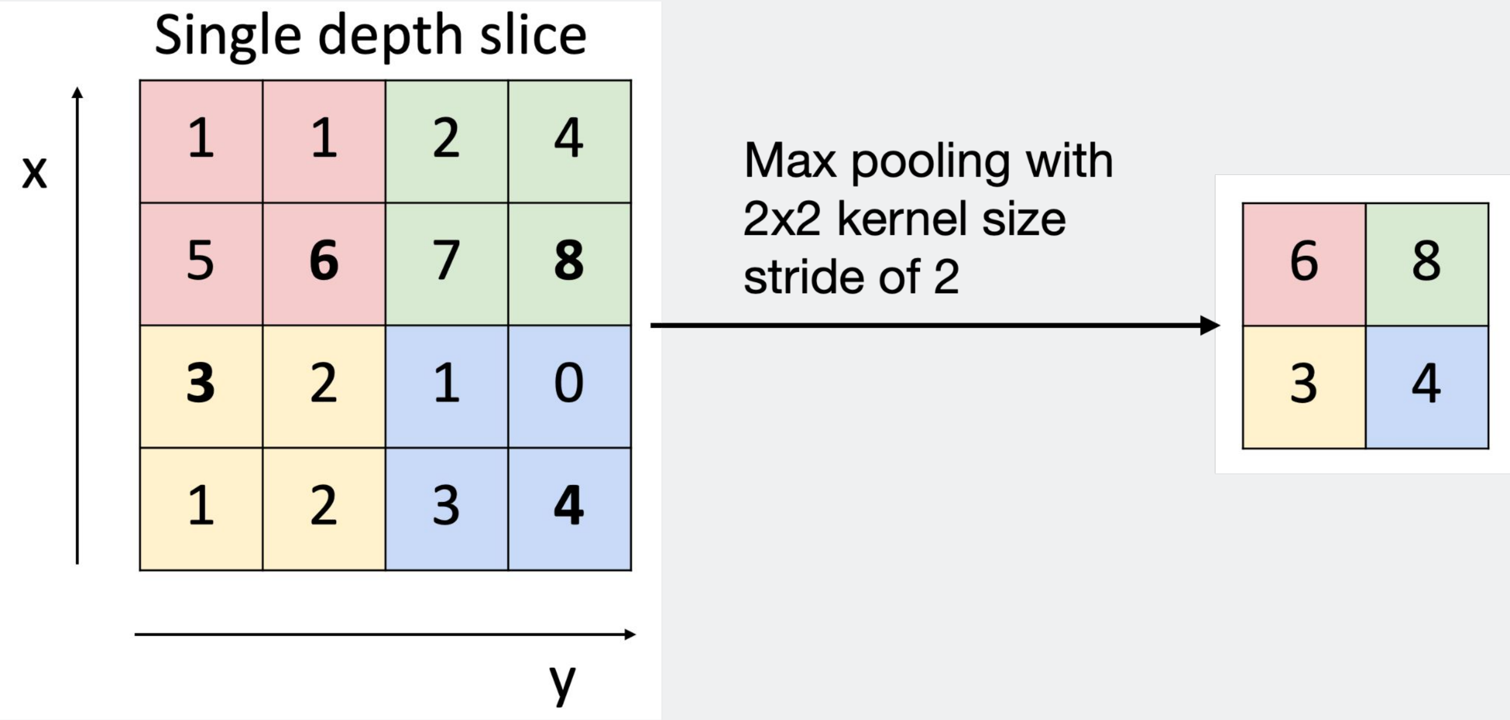
Pooling Layer

Another way to Downsample



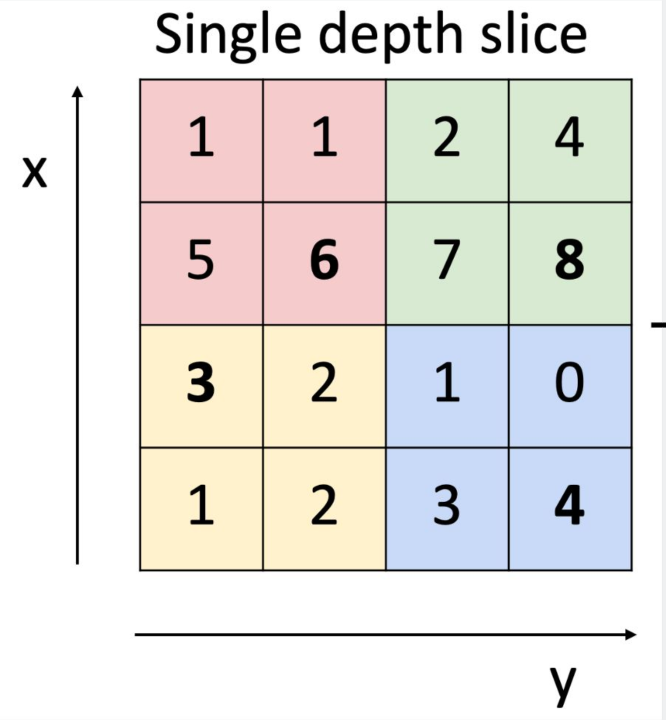
Hyperparameters:
Kernel size
Stride
Pooling function

Max Pooling

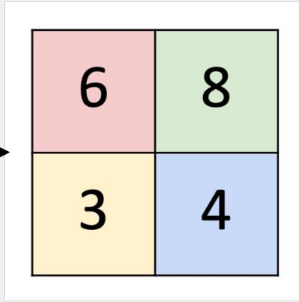


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
Max pooling with
2x2 kernel size
stride of 2
6 8
3 4

Max Pooling



Max pooling with
2x2 kernel size
stride of 2



Introduces invariance to
small spatial shifts

No learnable parameters!

Pooling Summary

Input: C x H x W

Hyperparameters:

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

Output: C x H' x W' where

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

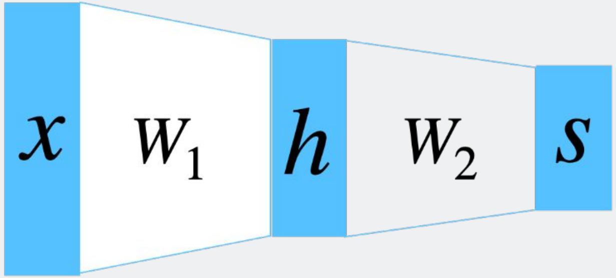
Learnable parameters: None!

Common settings:
max, K = 2, S = 2
max, K = 3, S = 2 (AlexNet)

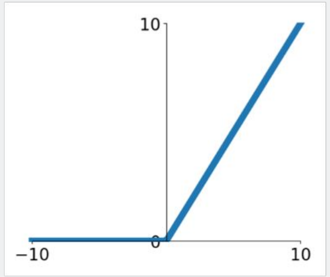
ROBOTICS

Components of Convolutional Neural Networks

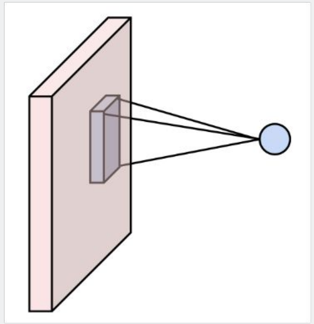
Fully-Connected Layers



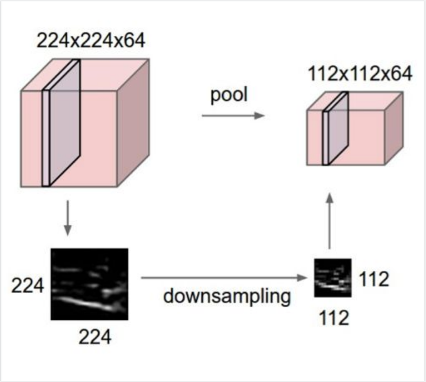
Activation Functions



Convolution Layers



Pooling Layers



Normalization

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

Batch Normalization

Consider a single layer $y = Wx$

The following could lead to tough optimization:

- Inputs x are not *centered around zero* (need large bias)
- Inputs x have different scaling per-element (entries in W will need to vary a lot)

Idea: force inputs to be "nicely scaled" at each layer!

Batch Normalization

Idea: "Normalize" the inputs of a layer so they have zero mean and unit variance

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!

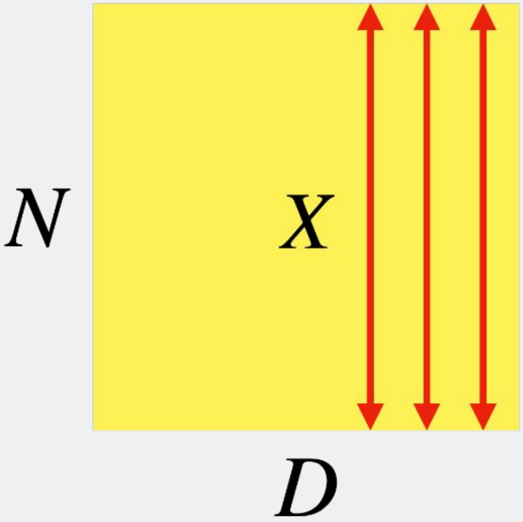
Aha Slides (In-class participation)

https://ahaslides.com/D5HXR

Q

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

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

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

Learnable scale and shift parameters: $\gamma, \beta \in \mathbb{R}^D$

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

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

Per-channel mean, shape is 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 $N \times D$

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

Output, shape is $N \times D$

Batch Normalization

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

Learnable scale and shift parameters: $\gamma, \beta \in \mathbb{R}^D$

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

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

$$\boxed{\begin{array}{ll} \mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j} & \text{Per-channel mean, shape is } D \\ \sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2 & \text{Per-channel std, shape is } D \end{array}}$$

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

Normalized x, shape is $N \times D$

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

Output, shape is $N \times D$

Batch Normalization: Test-Time

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

Learnable scale and shift parameters: $\gamma, \beta \in \mathbb{R}^D$

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

$\mu_j =$ (Running) average of values seen during training

Per-channel mean, shape is D

$\sigma_j^2 =$ (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 $N \times D$

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

Output, shape is $N \times D$

Batch Normalization: Test-Time

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

Learnable scale and shift parameters: $\gamma, \beta \in \mathbb{R}^D$

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

$\mu_j =$ (Running) average of values seen during training

Per-channel mean, shape is D

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

For each training iteration:

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

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

(Similar for σ)

Batch Normalization: Test-Time

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

Learnable scale and shift parameters: $\gamma, \beta \in \mathbb{R}^D$

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

$\mu_j =$ (Running) average of values seen during training

Per-channel mean, shape is D

$\sigma_j^2 =$ (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 $N \times D$

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

Output, shape is $N \times D$

Batch Batch Normalization: Test-Time

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

Learnable scale and shift parameters: $\gamma, \beta \in \mathbb{R}^{D}$

During testing batchnorm becomes a linear operator!
Can be fused with the previous fully-connected or conv layer

$\mu_j =$ (Running) average of values seen during training

Per-channel mean, shape is D

$\sigma_j^2 =$ (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 $N \times D$

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

Output, shape is $N \times D$

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)

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

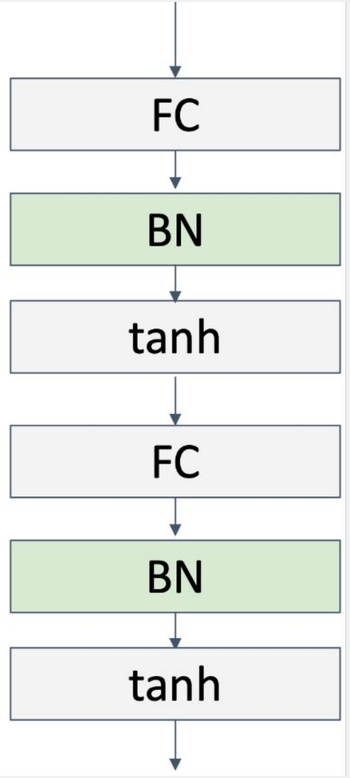
Normalize

$$\mu, \sigma : 1 \times C \times 1 \times 1$$

$$\gamma, \beta : 1 \times C \times 1 \times 1$$

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

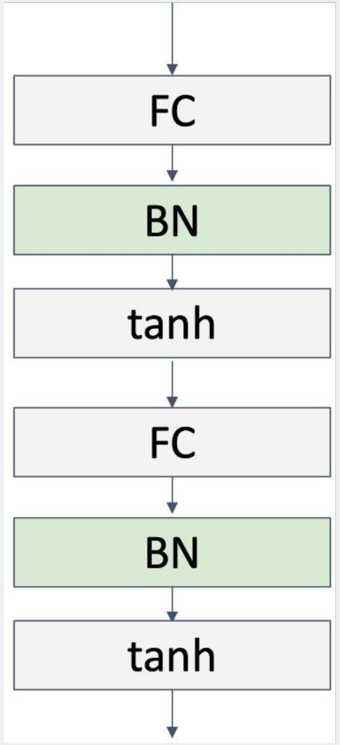
Batch Normalization



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

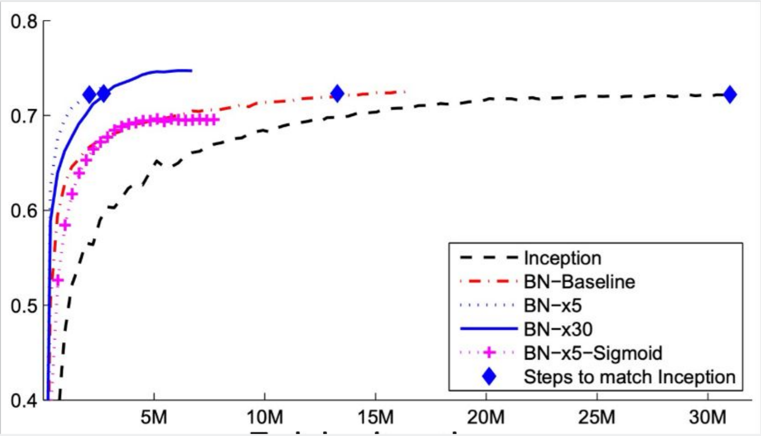
$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}$$

Batch Normalization



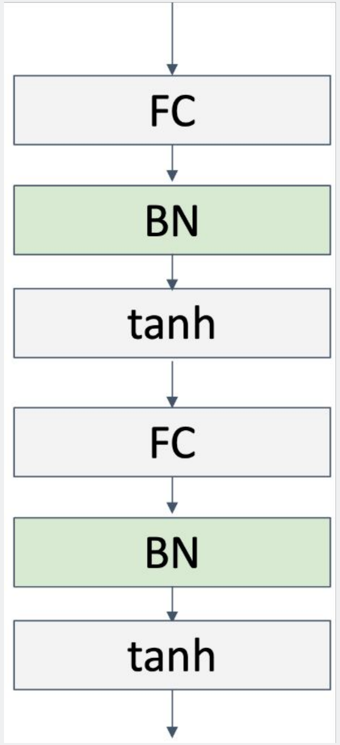
- Makes deep networks **much** easier to train!
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training.
- Zero overhead at test-time: can be fused with conv!

ImageNet accuracy



Training iterations

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 very common source of bugs!

Layer Normalization

Batch Normalization for **fully-connected** networks

$$x : N \times D$$

Normalize
$$\boxed{\mu, \sigma : 1 \times D}$$

$$\gamma, \beta : 1 \times D$$

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Layer Normalization for **fully-connected** networks
Same behavior at train and test!
Used in RNNs, Transformers

$$x : N \times D$$

Normalize
$$\boxed{\mu, \sigma : N \times 1}$$

$$\gamma, \beta : 1 \times D$$

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

ROBOTICS

Instance Normalization

Batch Normalization for **convolutional** networks

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

Normalize

$$\mu, \sigma : 1 \times C \times 1 \times 1$$

$$\gamma, \beta : 1 \times C \times 1 \times 1$$

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Instance Normalization for **convolutional** networks
Same behavior at train / test!

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

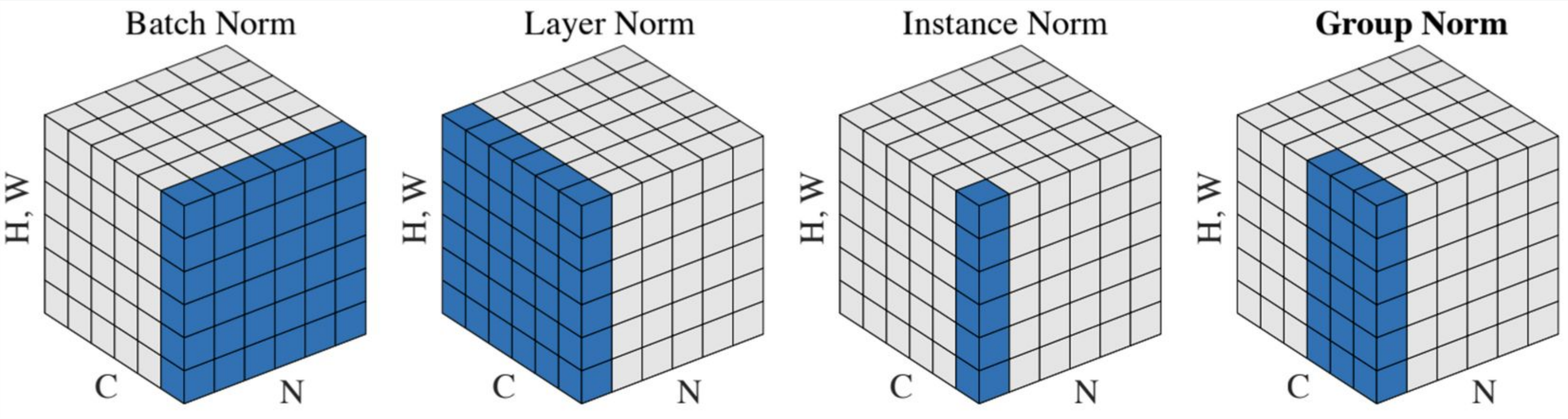
Normalize

$$\mu, \sigma : N \times C \times 1 \times 1$$

$$\gamma, \beta : 1 \times C \times 1 \times 1$$

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Group Normalization

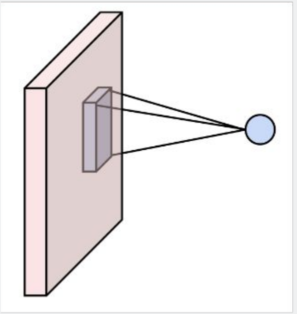


Wu and He, "Group Normalization," ECCV 2018
https://openaccess.thecvf.com/content_ECCV_2018/papers/Yuxin_Wu_Group_Normalization_ECCV_2018_paper.pdf

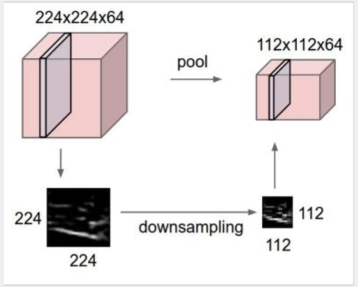
ROBOTICS

Summary: Components of Convolutional Networks

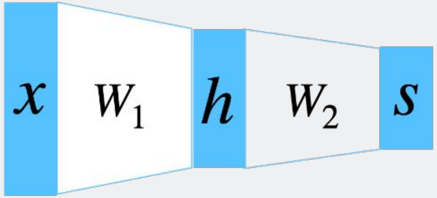
Convolution Layers



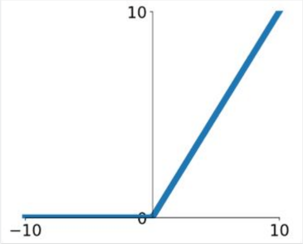
Pooling Layers



Fully-Connected Layers



Activation Function



Normalization

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

Next Up

Question: How should we put them together?

Due dates

Canvas Assignment: (reminder)

Scored - individual (as part of in-class activity points)

20250129 BackProp quiz - Due Feb. 3, 2025 (tonight)
20250203 Conv layer quiz - Due Feb. 5, 2025 (Wednesday)

P2 (ConvNet)

5 submissions per day - Start today!!!

Due Feb. 16, 2025

Due dates

Reminder: For tomorrow (Tuesday, Feb.4) discussion section, in Visualization Studio in Duderstadt Center
https://xr.engin.umich.edu/visualization-studio/

NOT in CSRB!!

Capacity - 30
Zoom link will be available