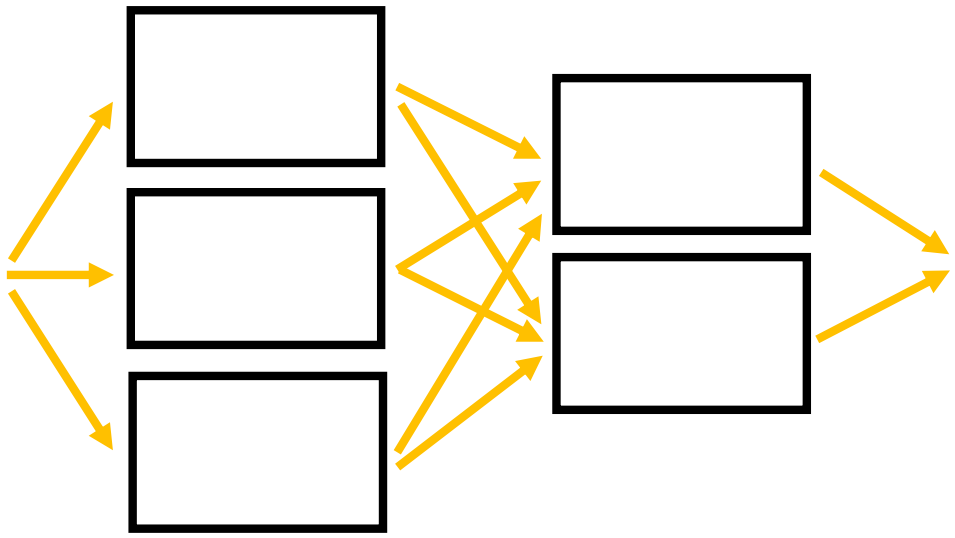
Convolutional Neural Networks



CS194: Computer Vision and Comp. Photo

Angjoo Kanazawa/Alexei Efros, UC Berkeley, Fall 2021

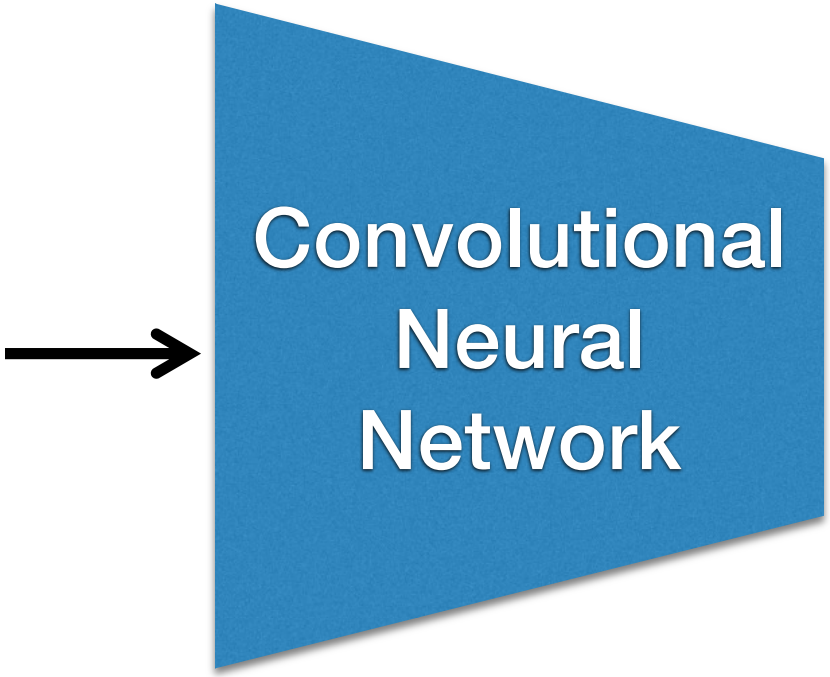
Project 5

Facial Keypoint Detection with Neural Networks

- Will be released today, in any moment
- You need to implement things in pytorch
- If you have NOT done pytorch before, you **must** go to the discussion by Tim/Vickie on Thursday 1-2pm

Neural Nets: a particularly useful Black Box

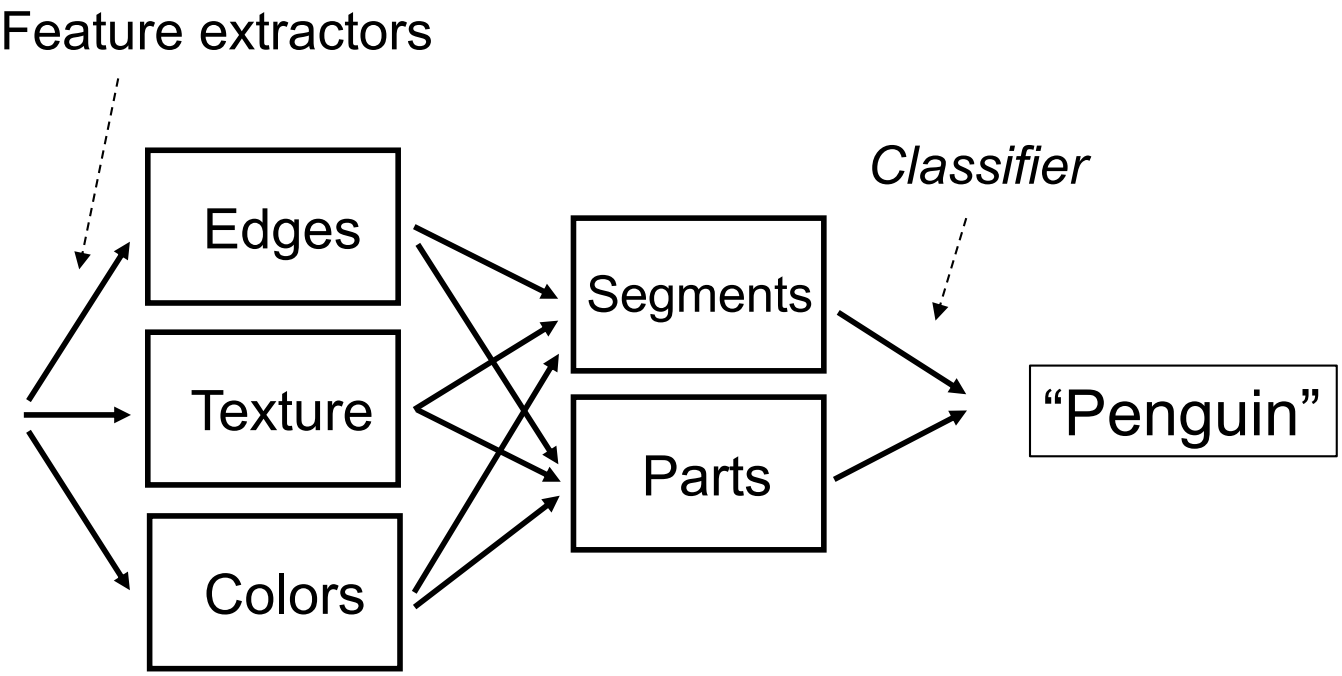
image *X*



label *Y*

Classic Object Recognition

image *X*

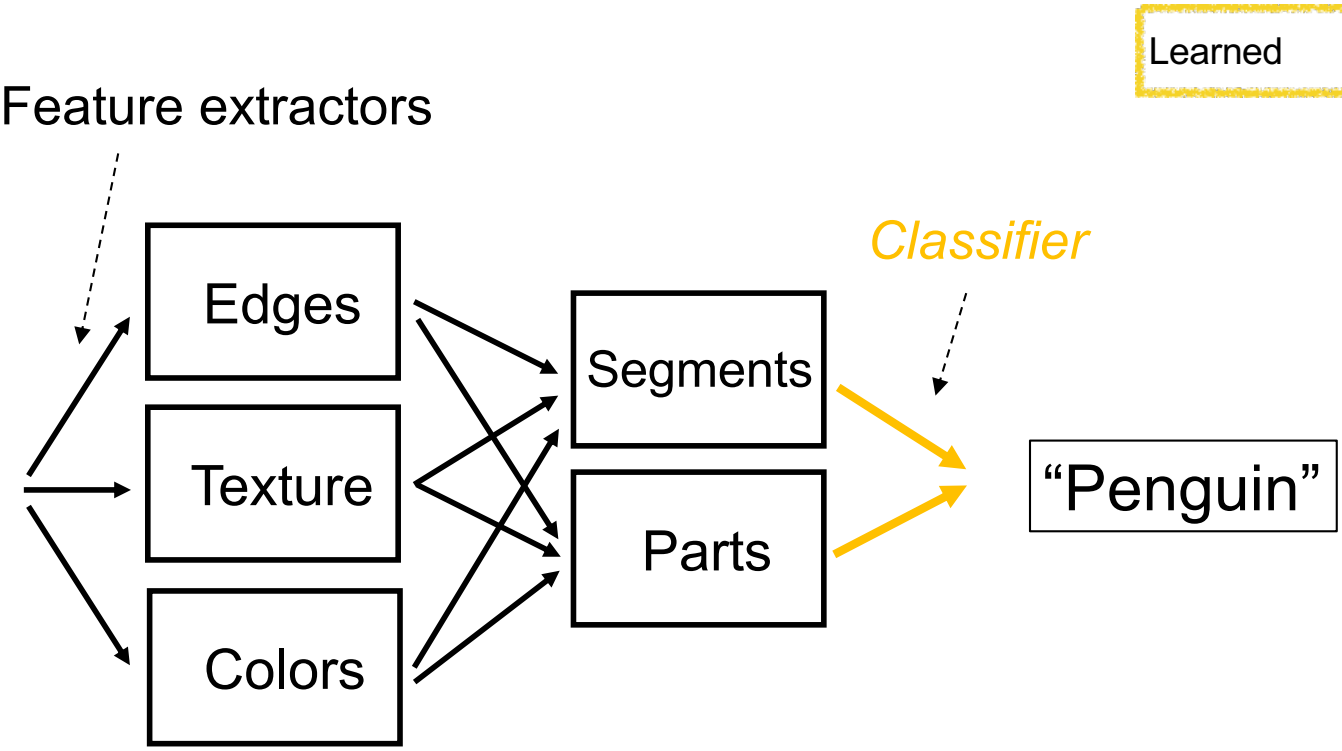


label *Y*

Slide by Philip Isola

Classic Object Recognition

image *X*



label *Y*

Slide by Philip Isola

Learning Features

Learned

Feature extractors

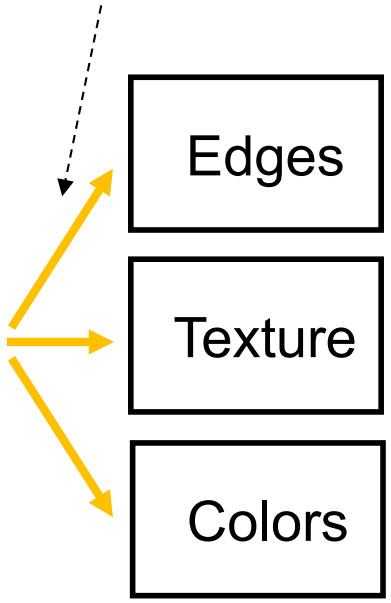


image *X*

label *Y*

Neural Network:
algorithm + feature + data!

Learned

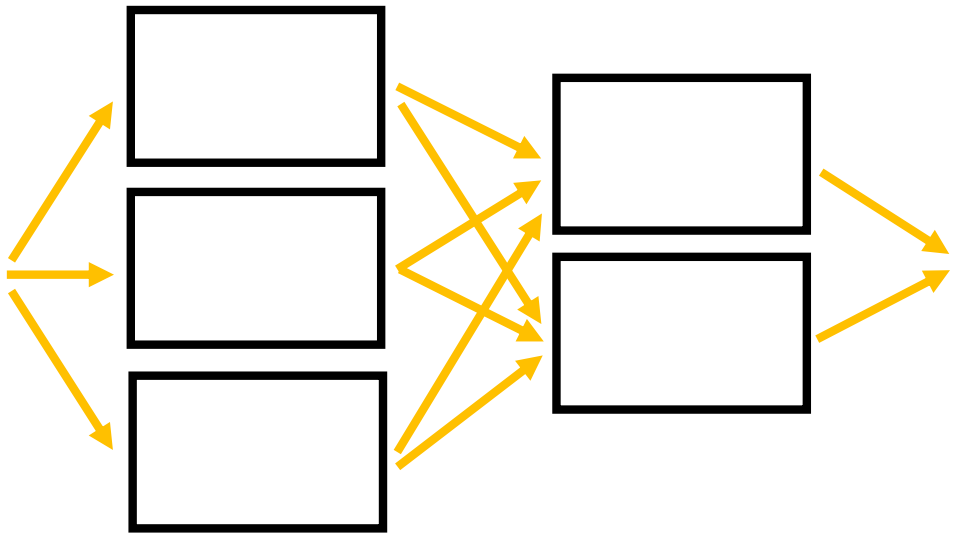
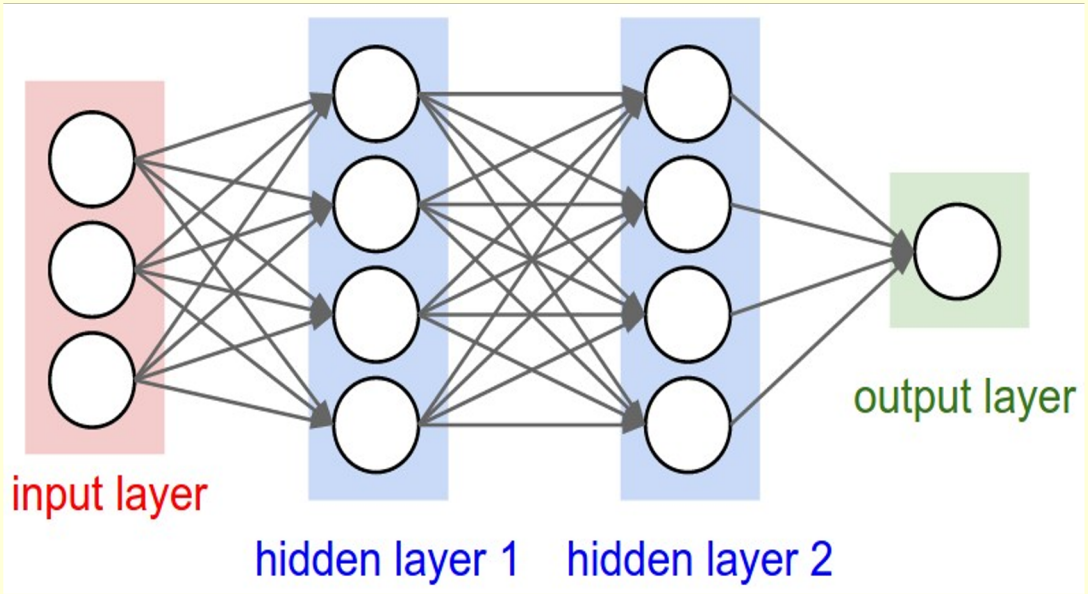


image *X*

label *Y*

Slide by Philip Isola

Vanilla (fully-connected) Neural Networks

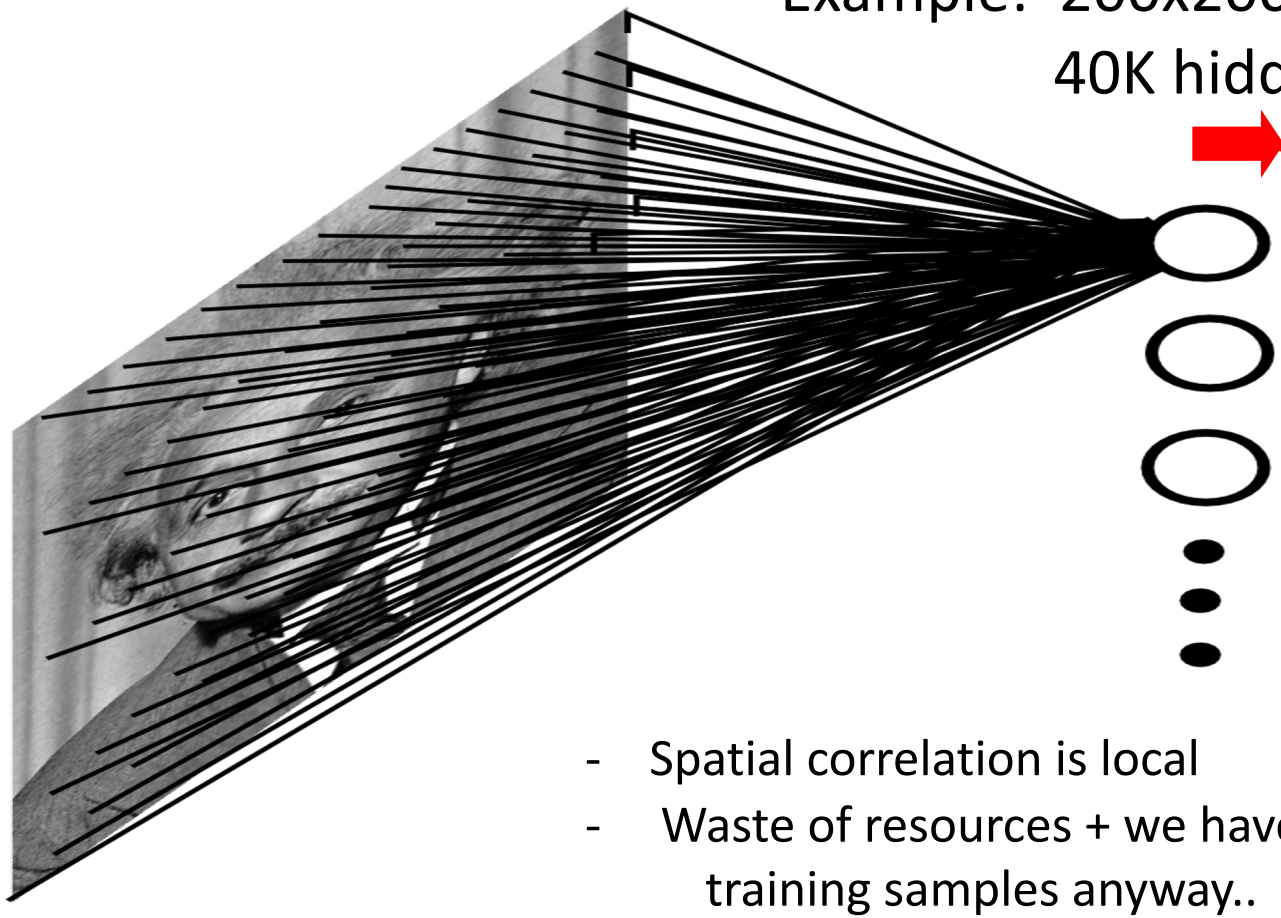


Fully Connected Layer

Example: 200x200 image

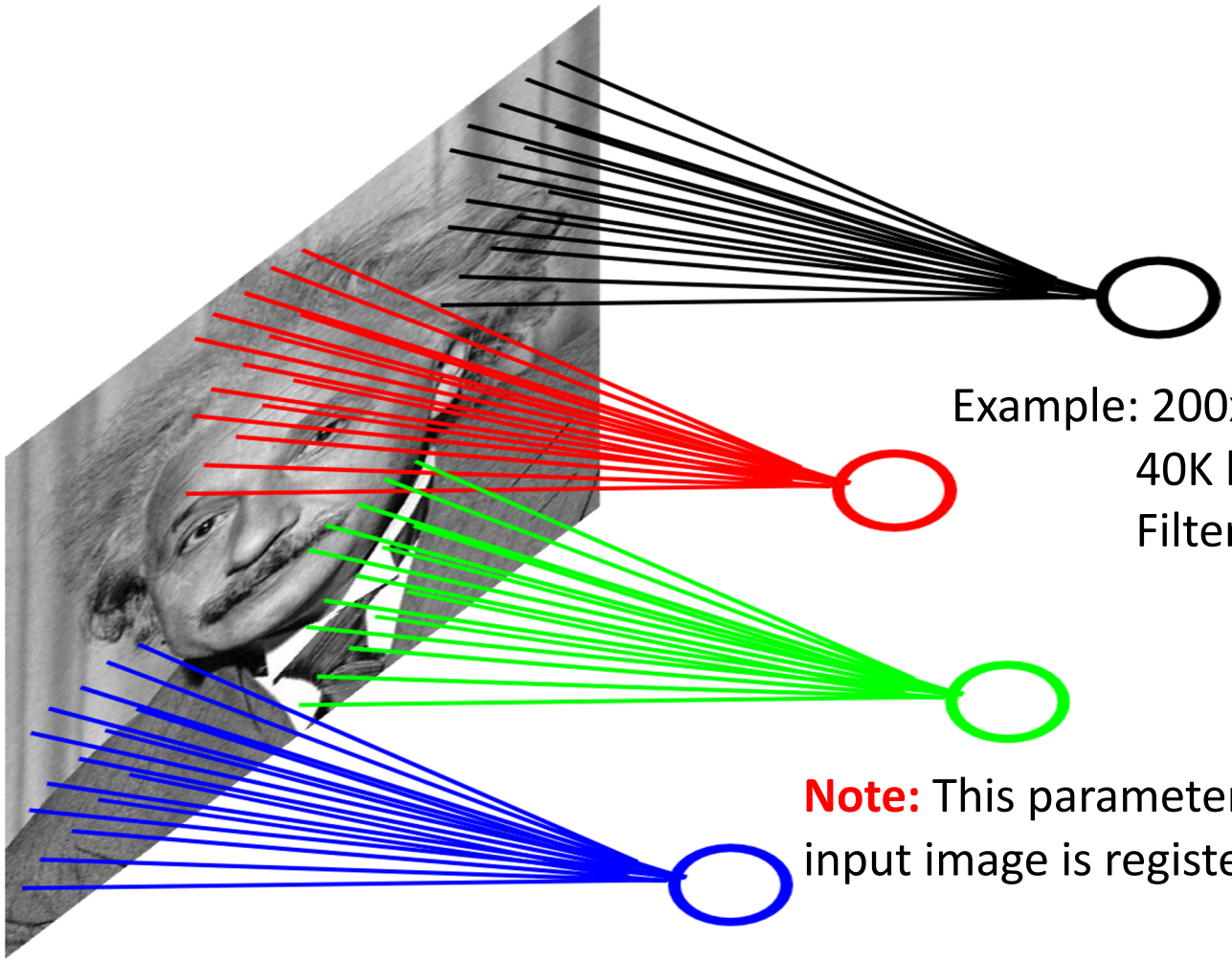
40K hidden units

2B parameters!!!



- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

Locally Connected Layer



Example: 200x200 image

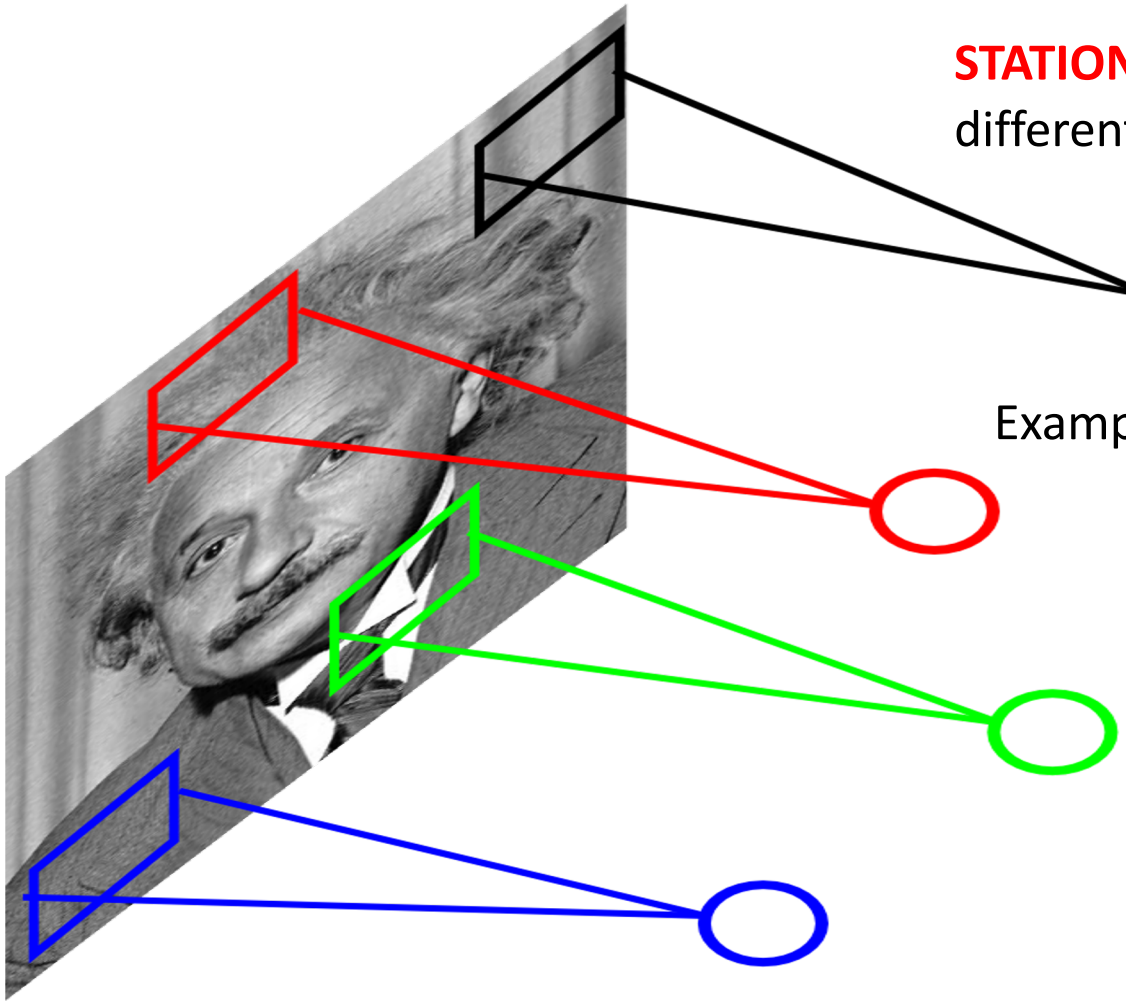
40K hidden units

Filter size: 10x10

4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

Locally Connected Layer



STATIONARITY? Statistics is similar at different locations

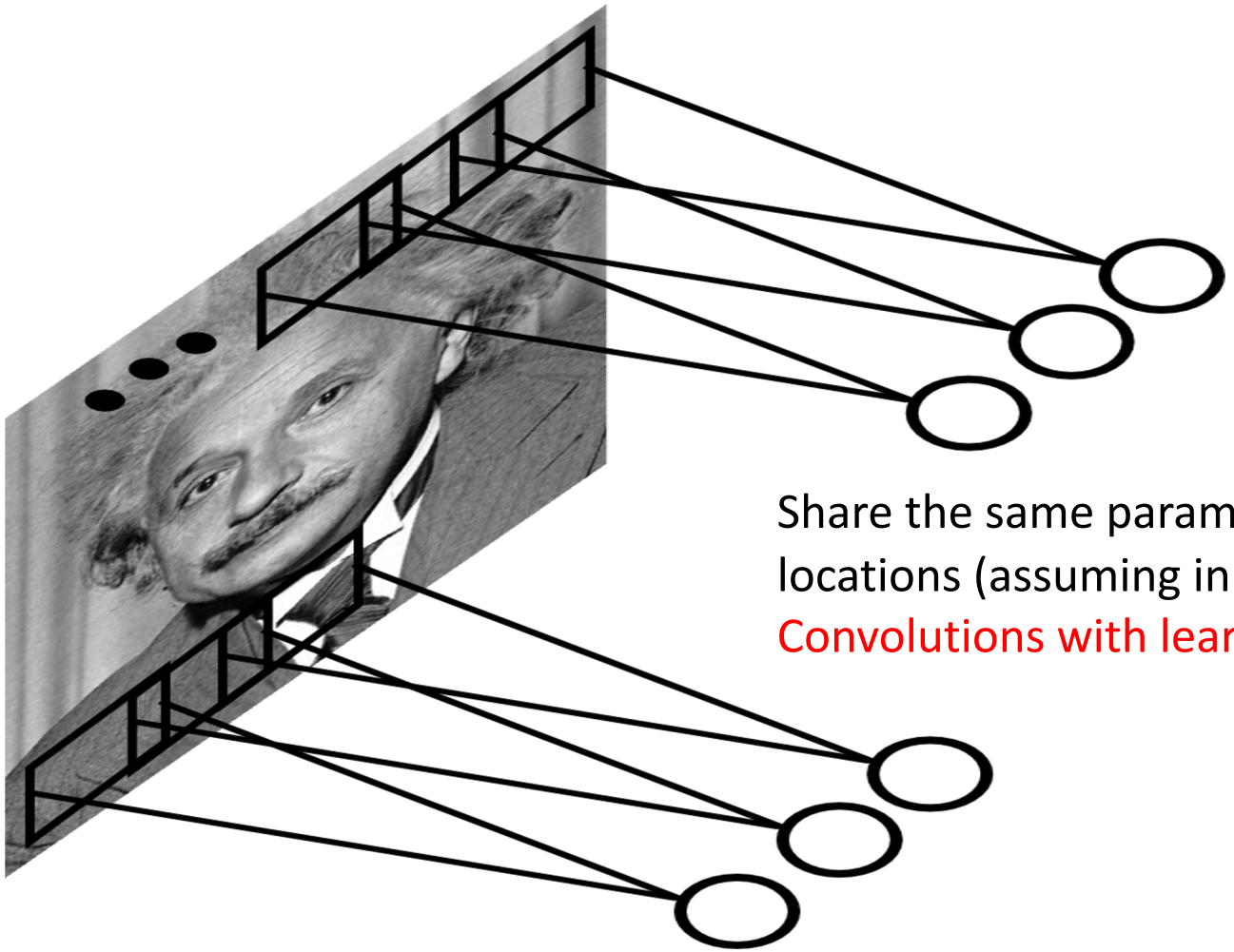
Example: 200x200 image

40K hidden units

Filter size: 10x10

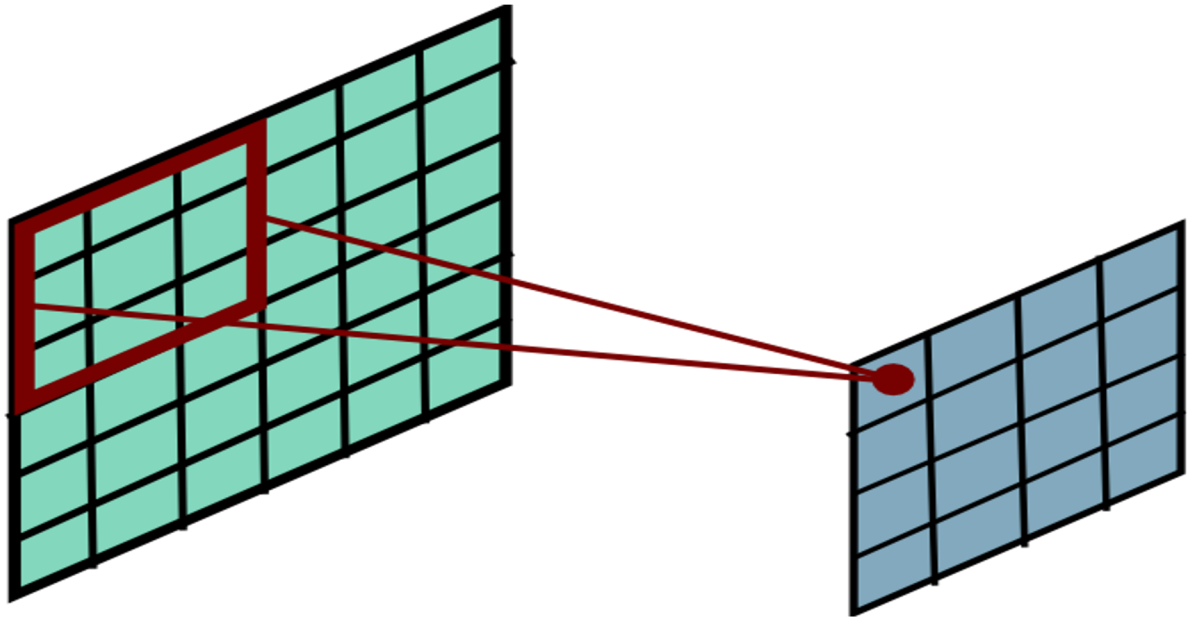
4M parameters

Convolutional Layer



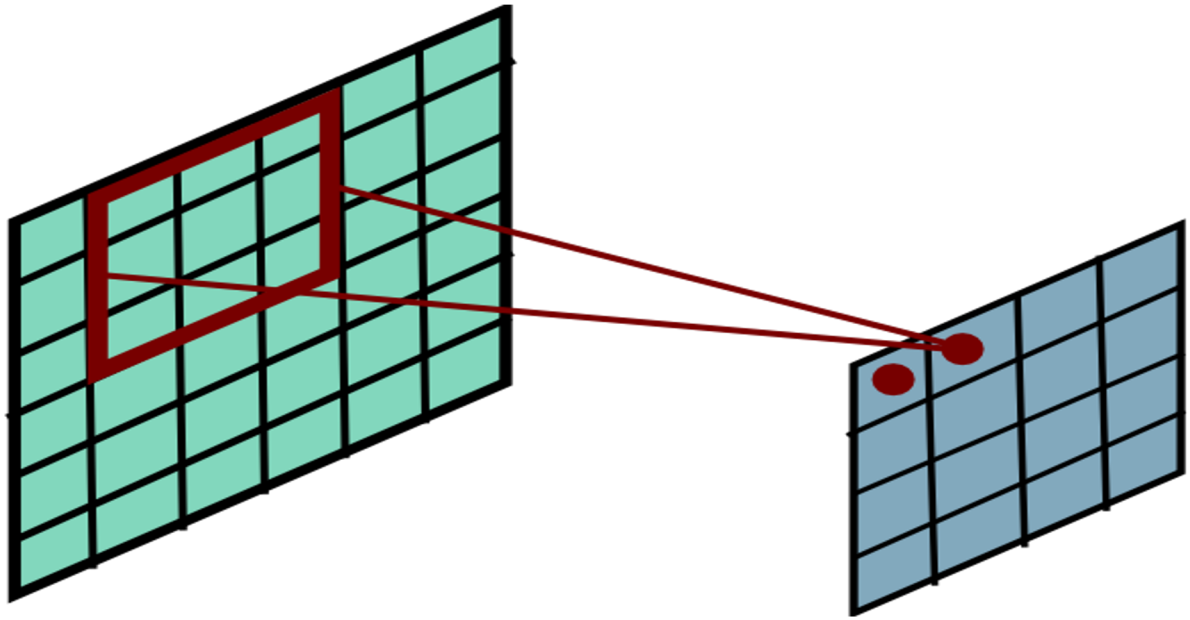
Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

Convolutional Layer



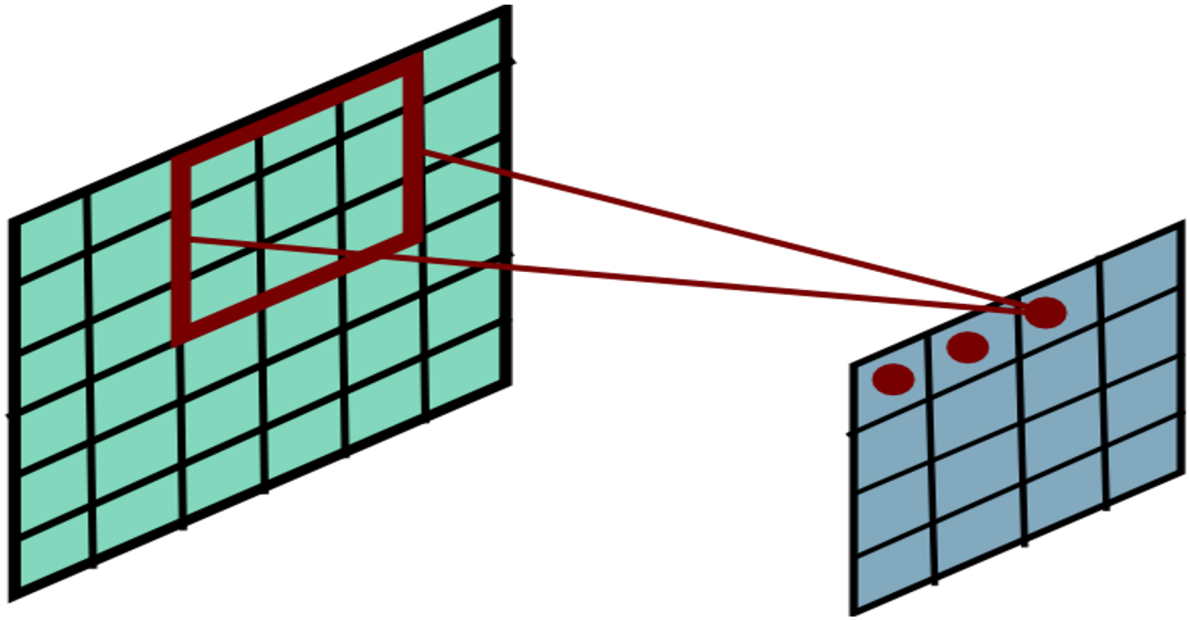
Ranzato

Convolutional Layer



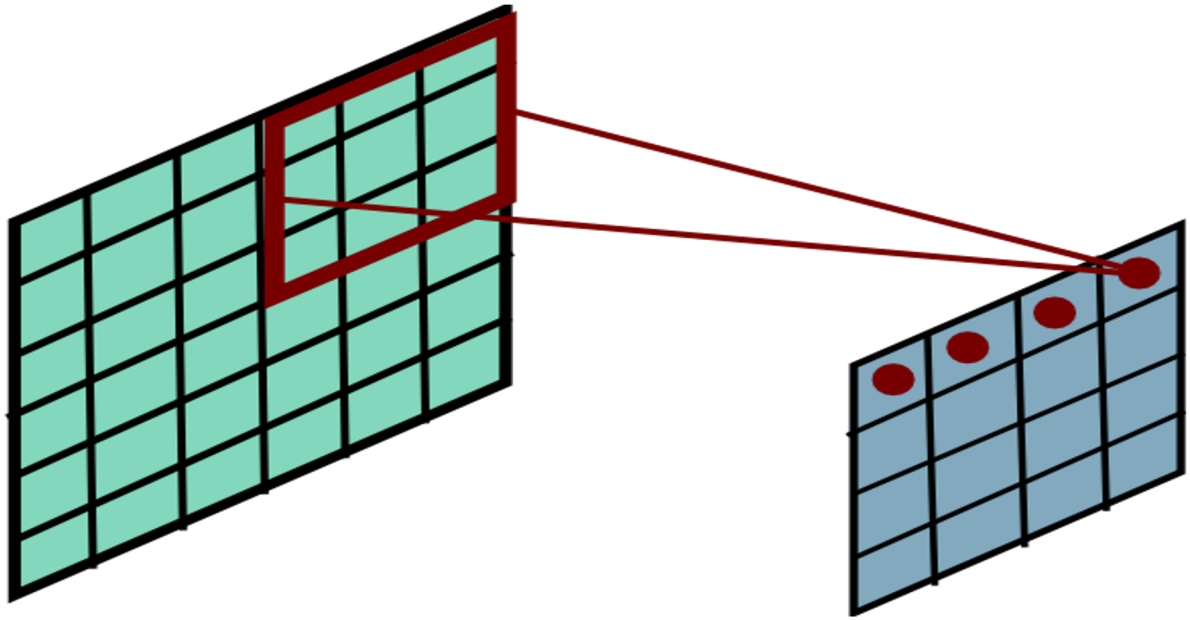
Ranzato

Convolutional Layer



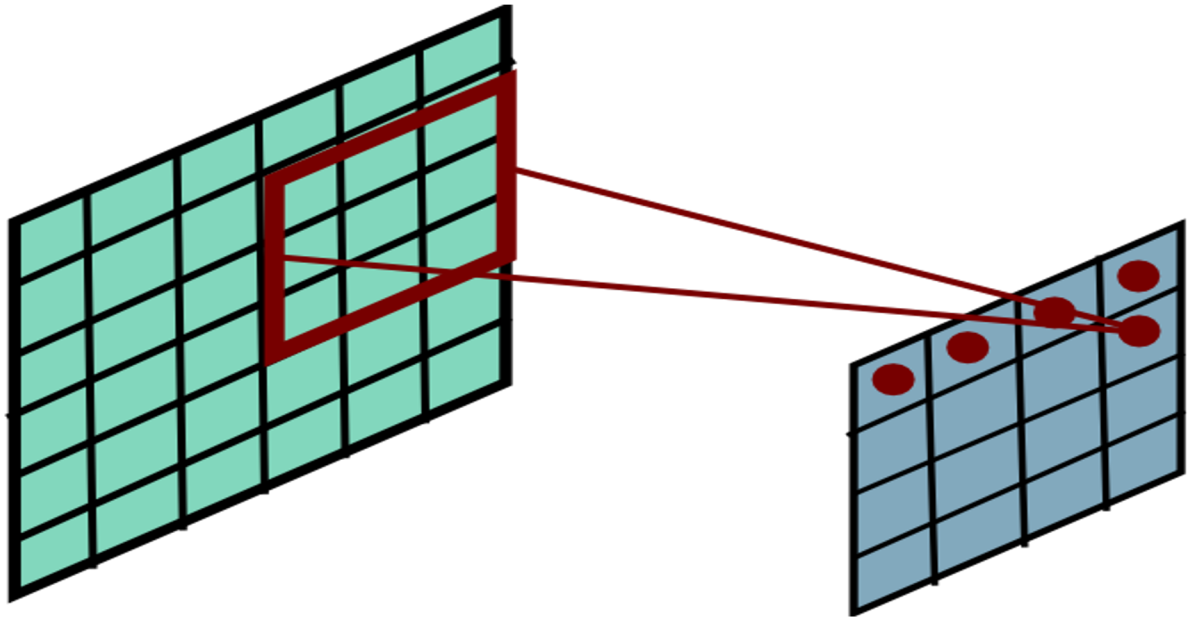
Ranzato

Convolutional Layer



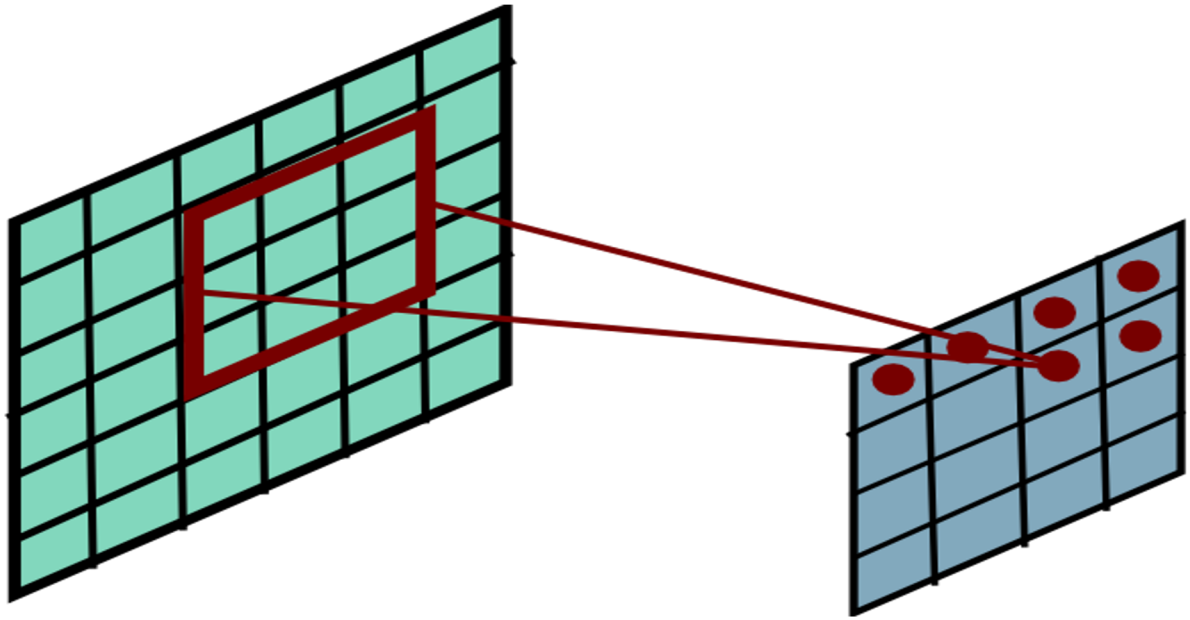
Ranzato

Convolutional Layer



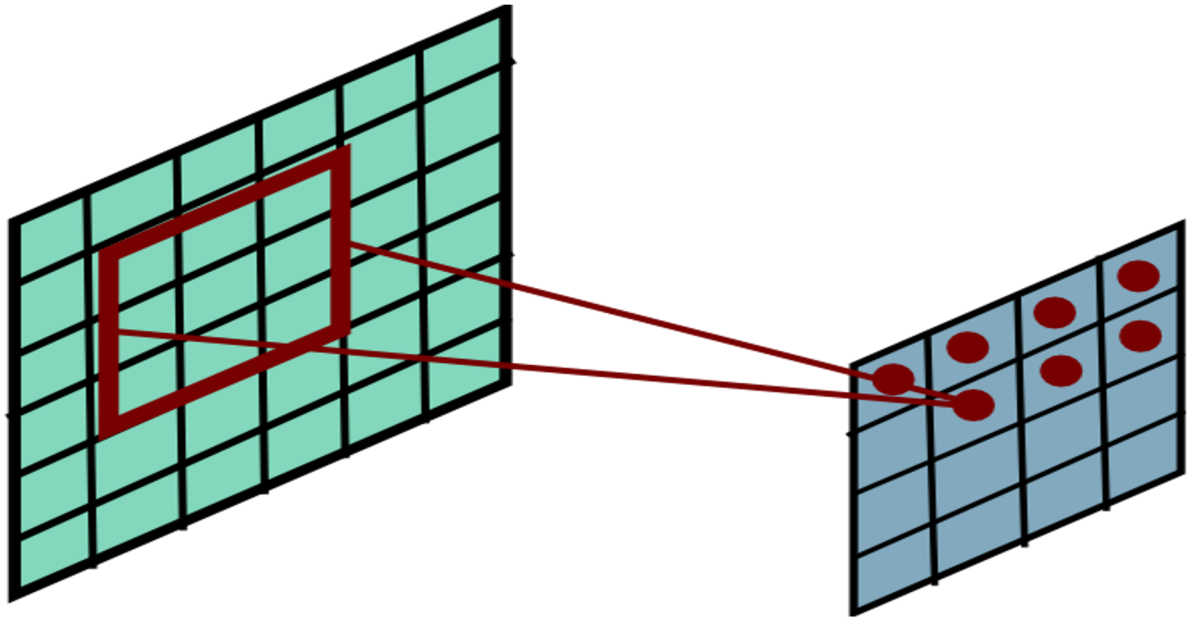
Ranzato

Convolutional Layer



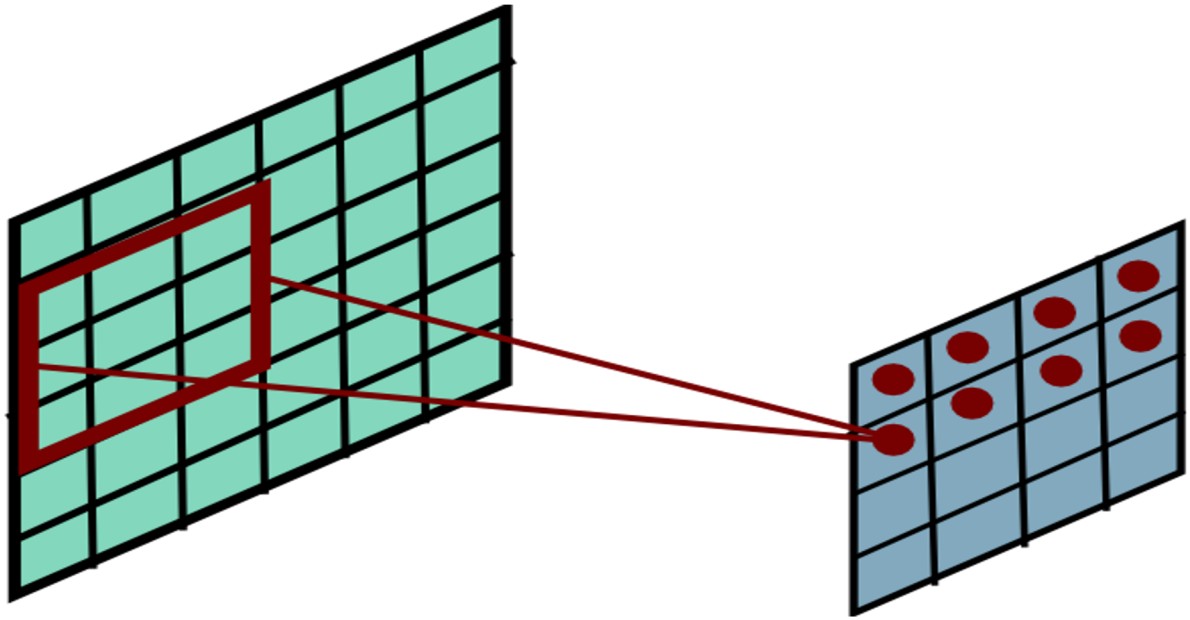
Ranzato

Convolutional Layer



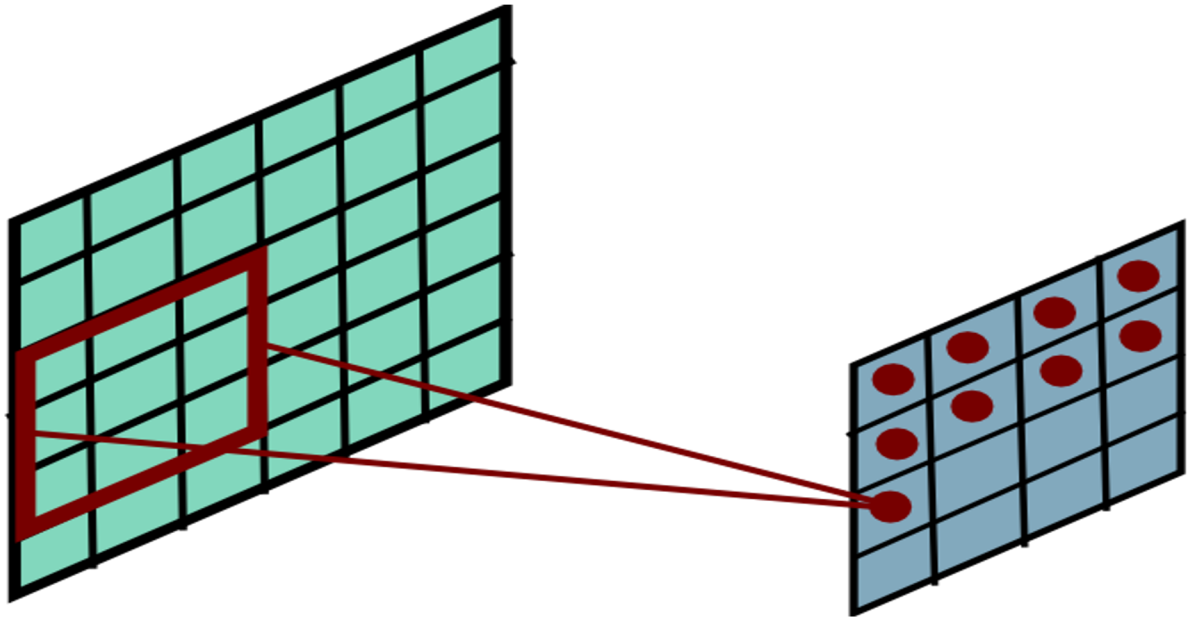
Ranzato

Convolutional Layer



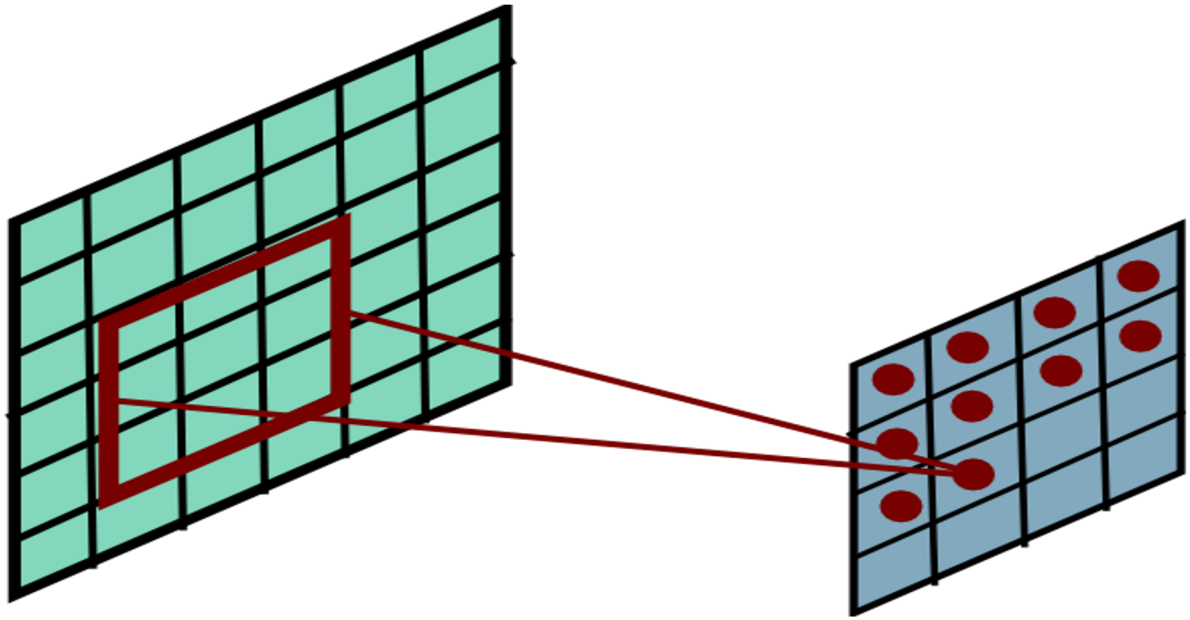
Ranzato

Convolutional Layer



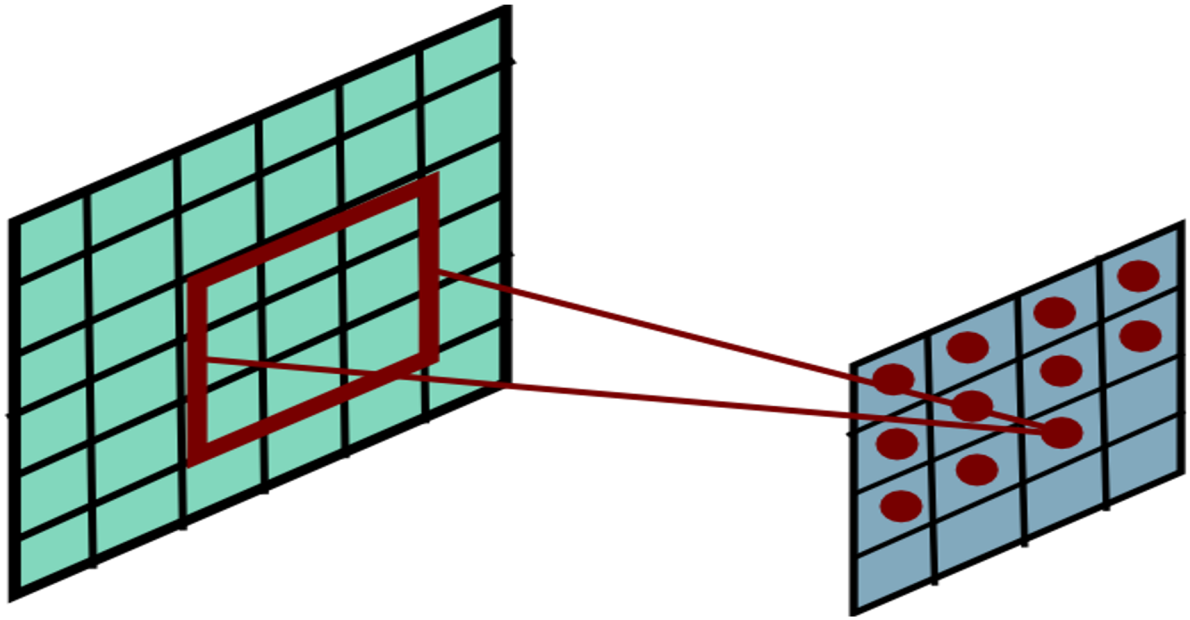
Ranzato

Convolutional Layer



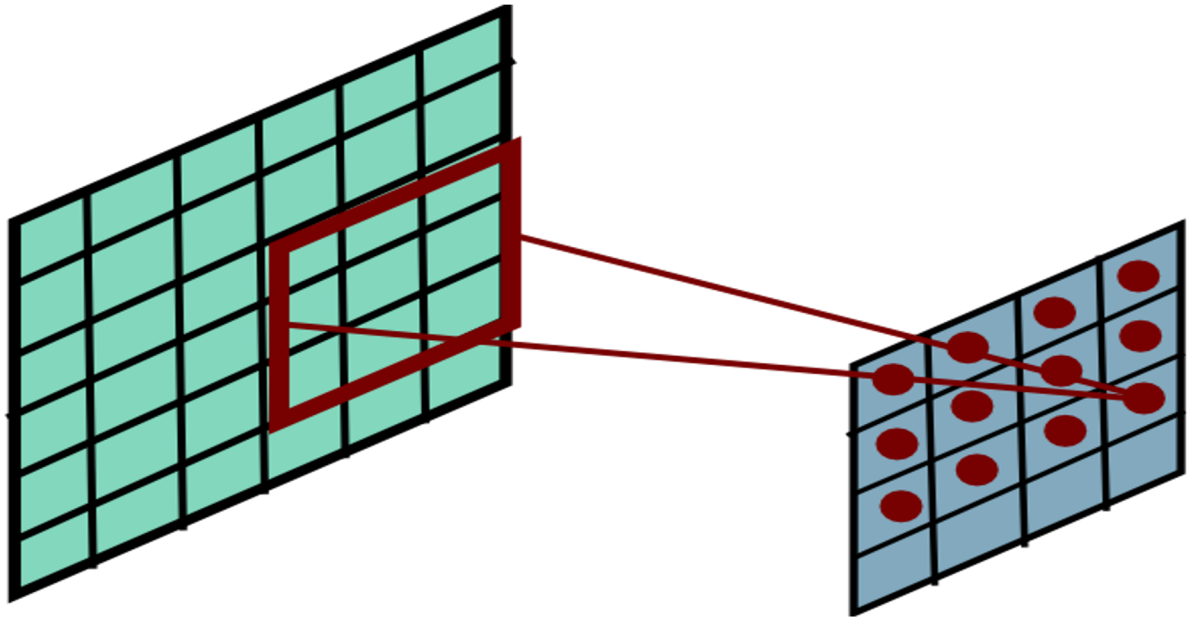
Ranzato

Convolutional Layer



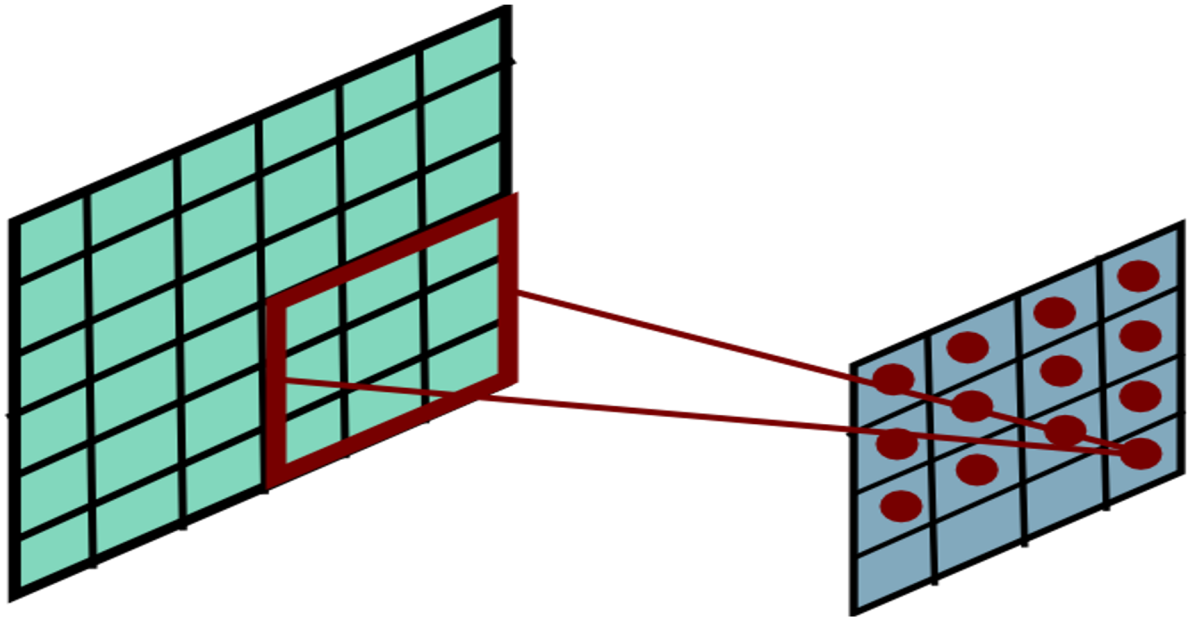
Ranzato

Convolutional Layer



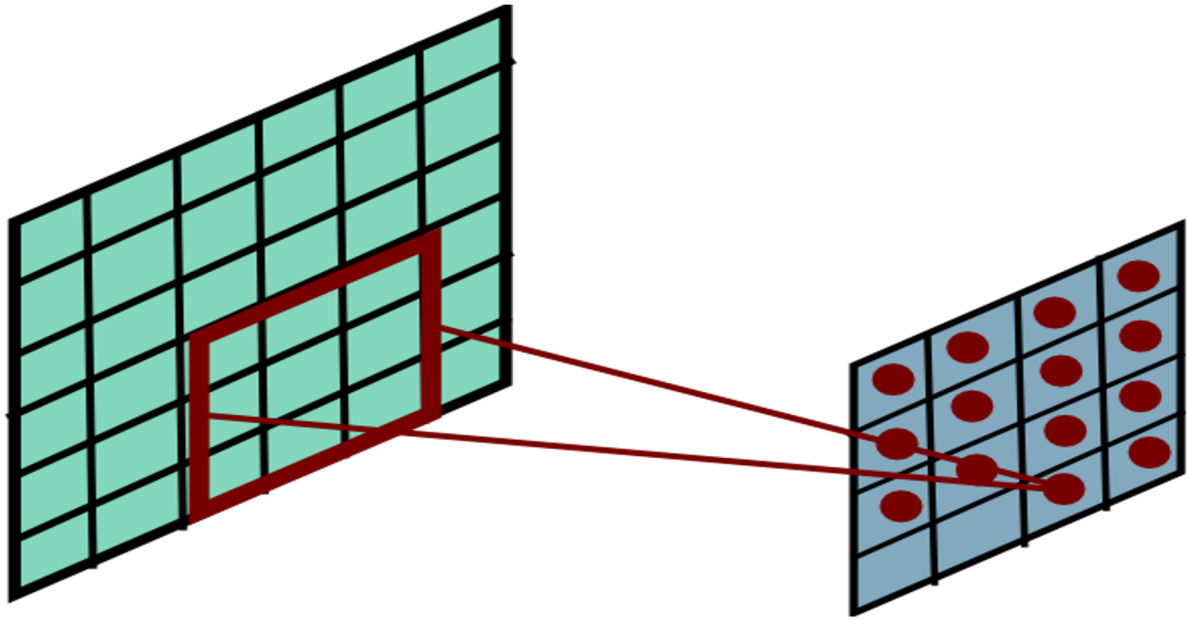
Ranzato

Convolutional Layer



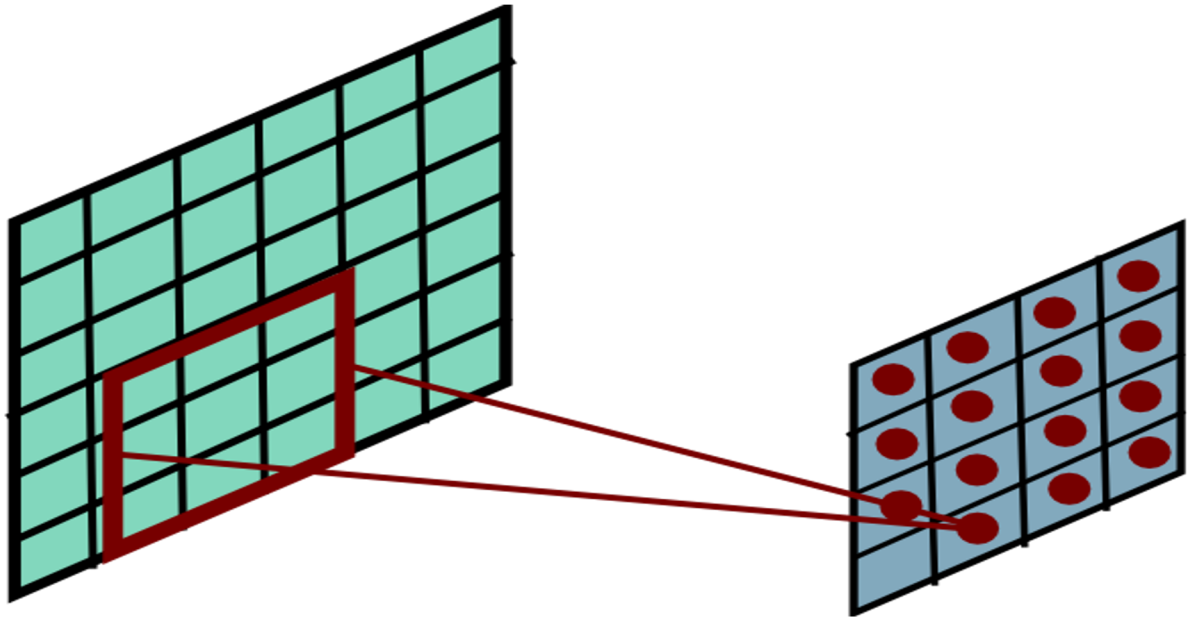
Ranzato

Convolutional Layer



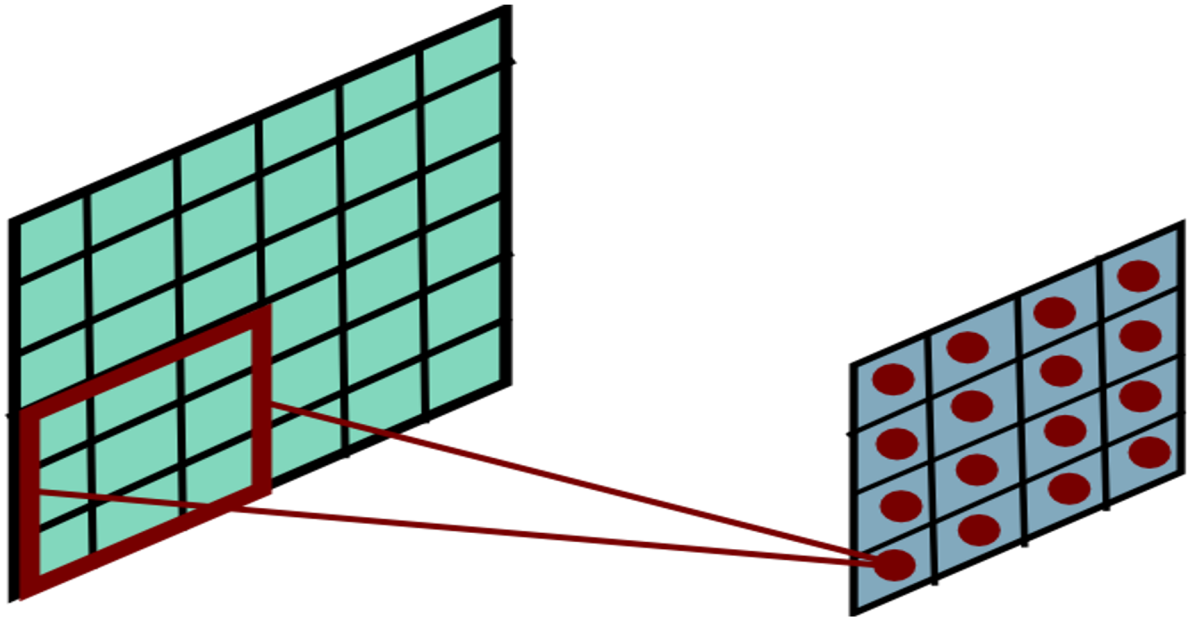
Ranzato

Convolutional Layer



Ranzato

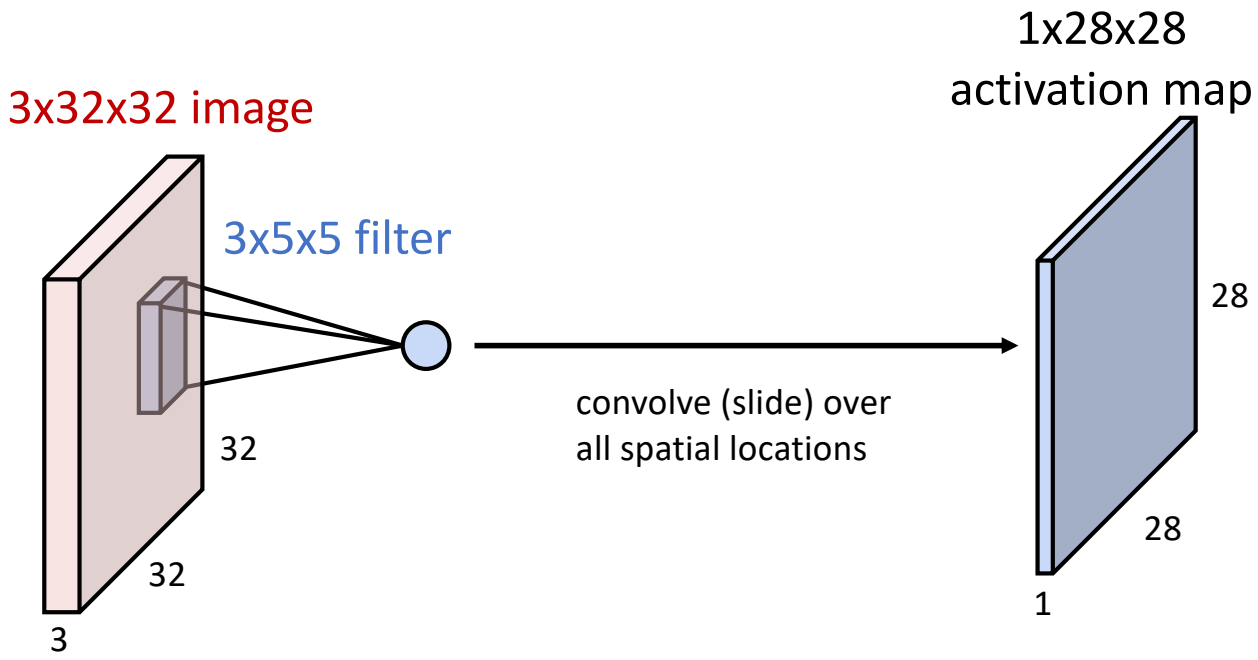
Convolutional Layer



Ranzato

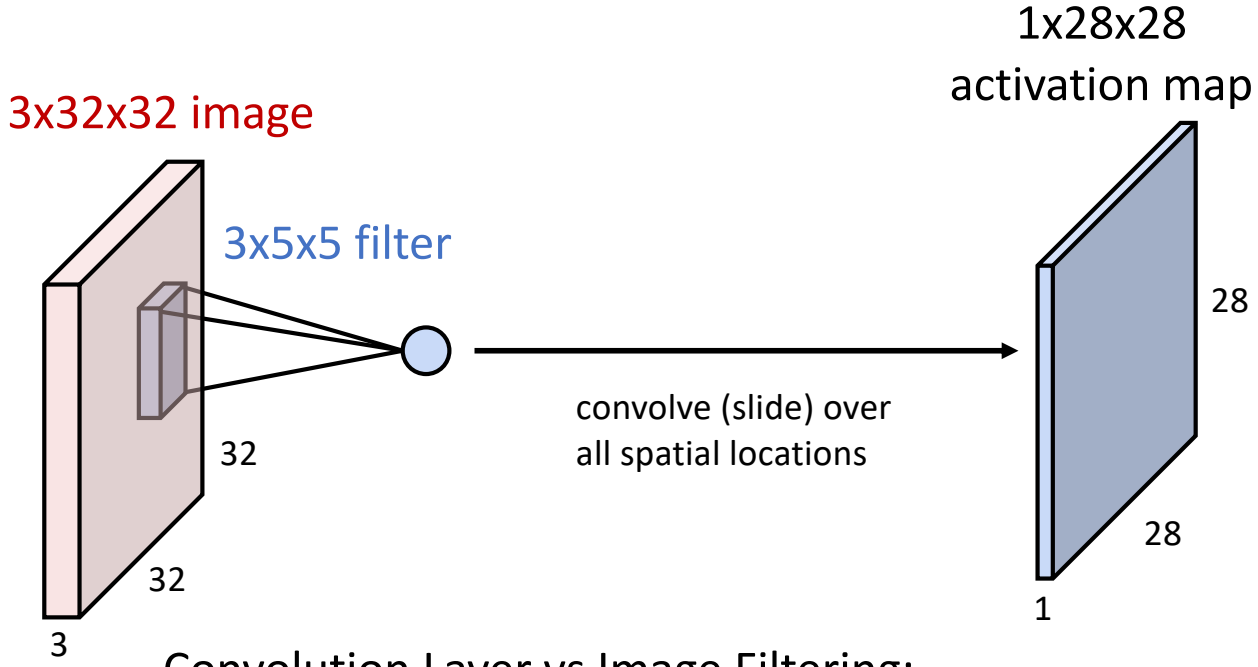
Convolution Layer

One neuron, that looks at 5x5 region and outputs a sheet of activation map



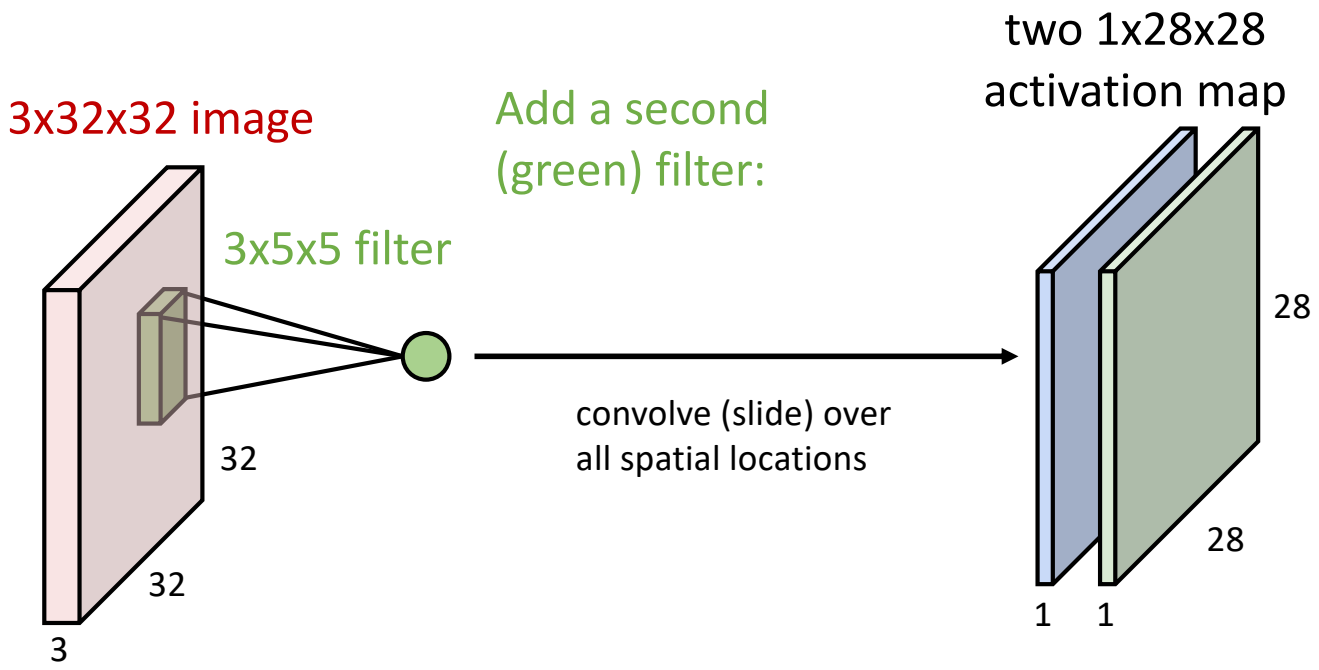
Convolution Layer

One neuron, that looks at 5x5 region and outputs a sheet of activation map

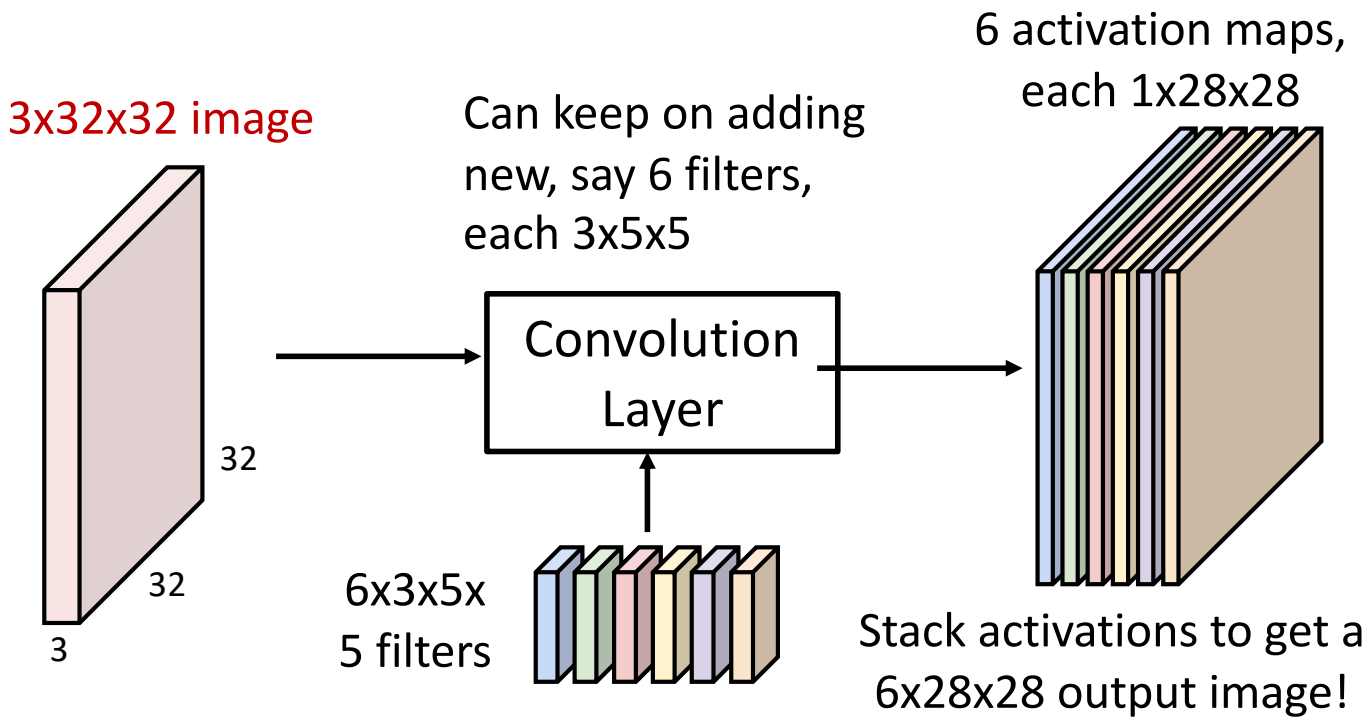


Convolution Layer vs Image Filtering:
>1 input and output channels
Forget about convolution vs cross-correlation (Why?)

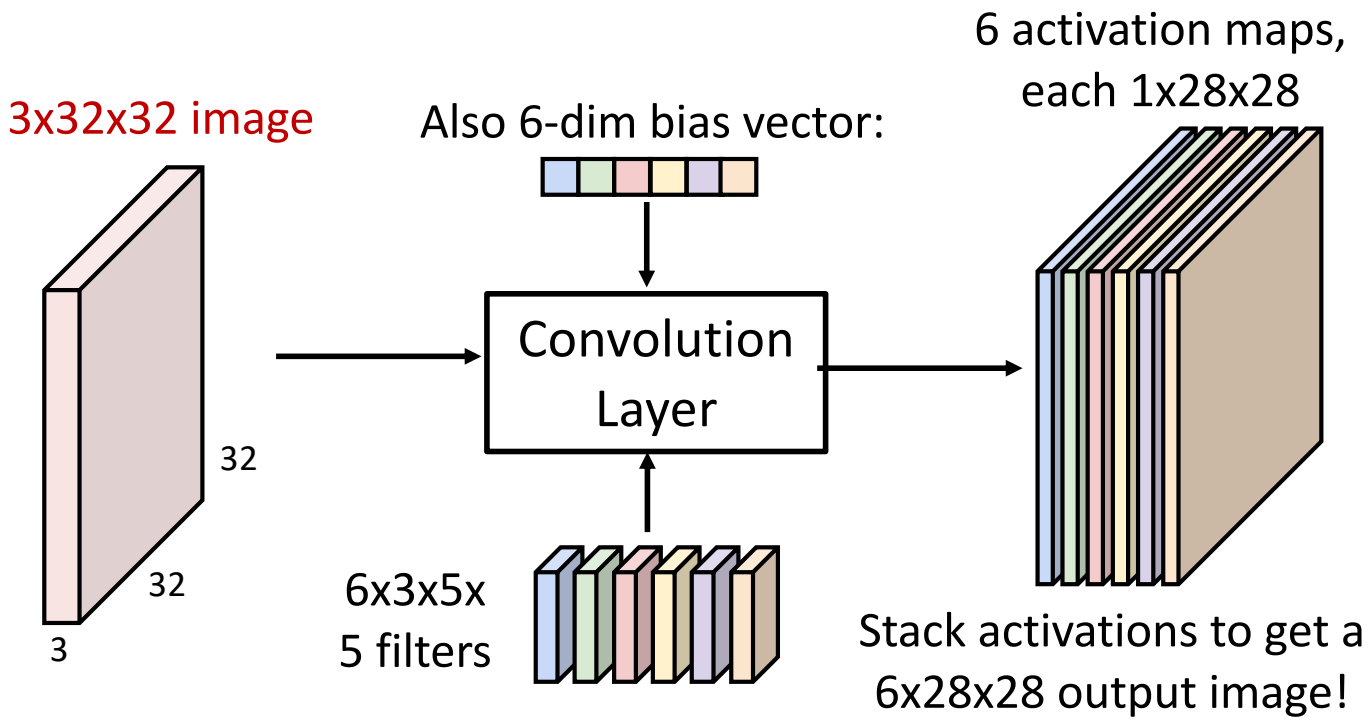
Convolution Layer:



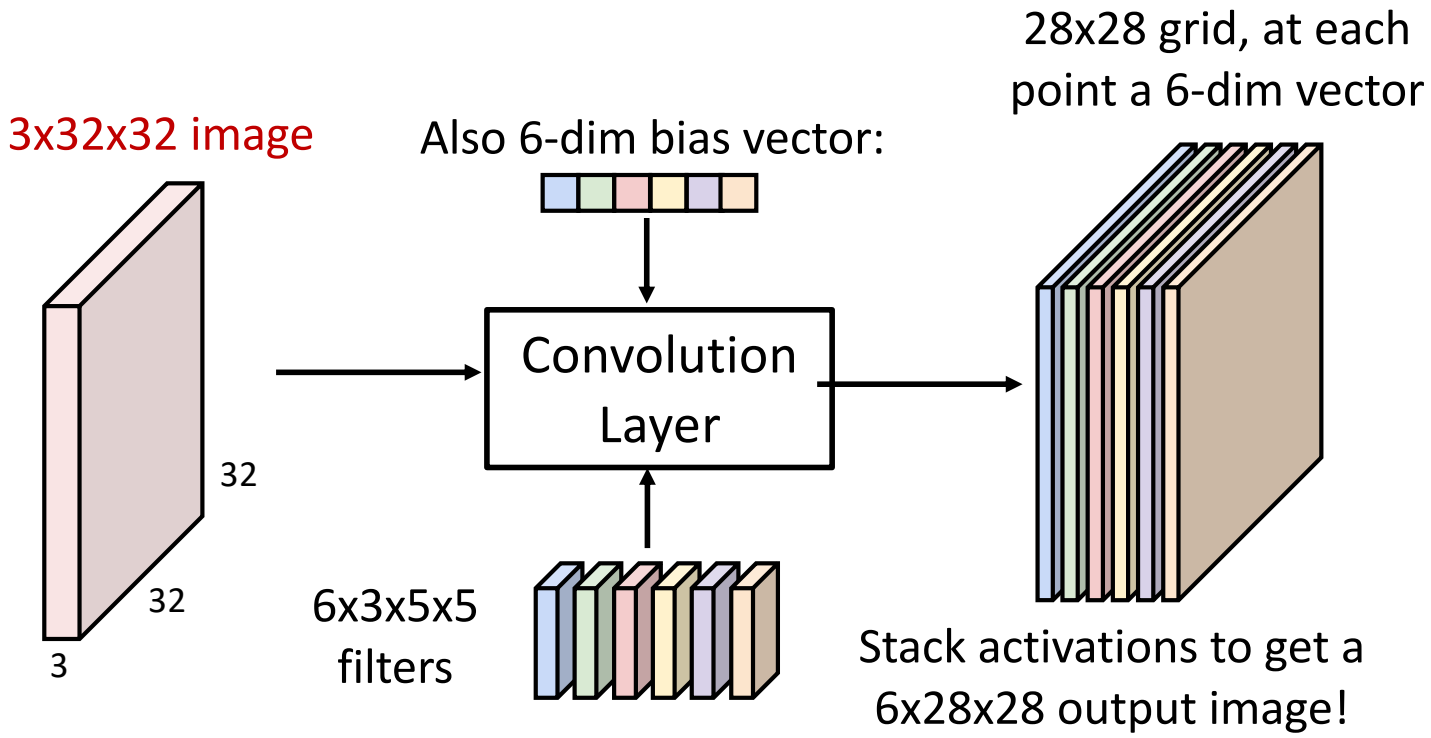
Convolution Layer



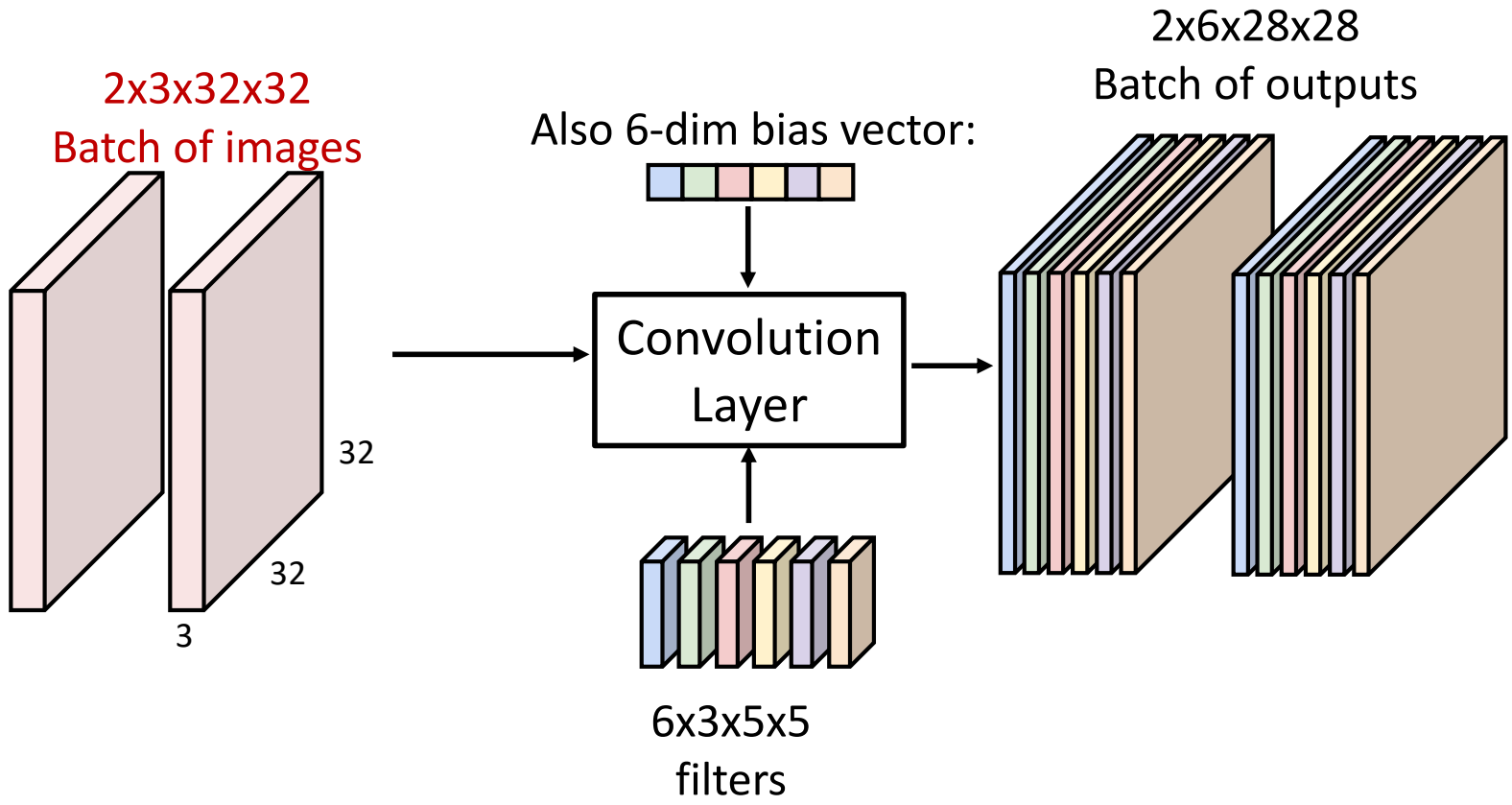
Convolution Layer



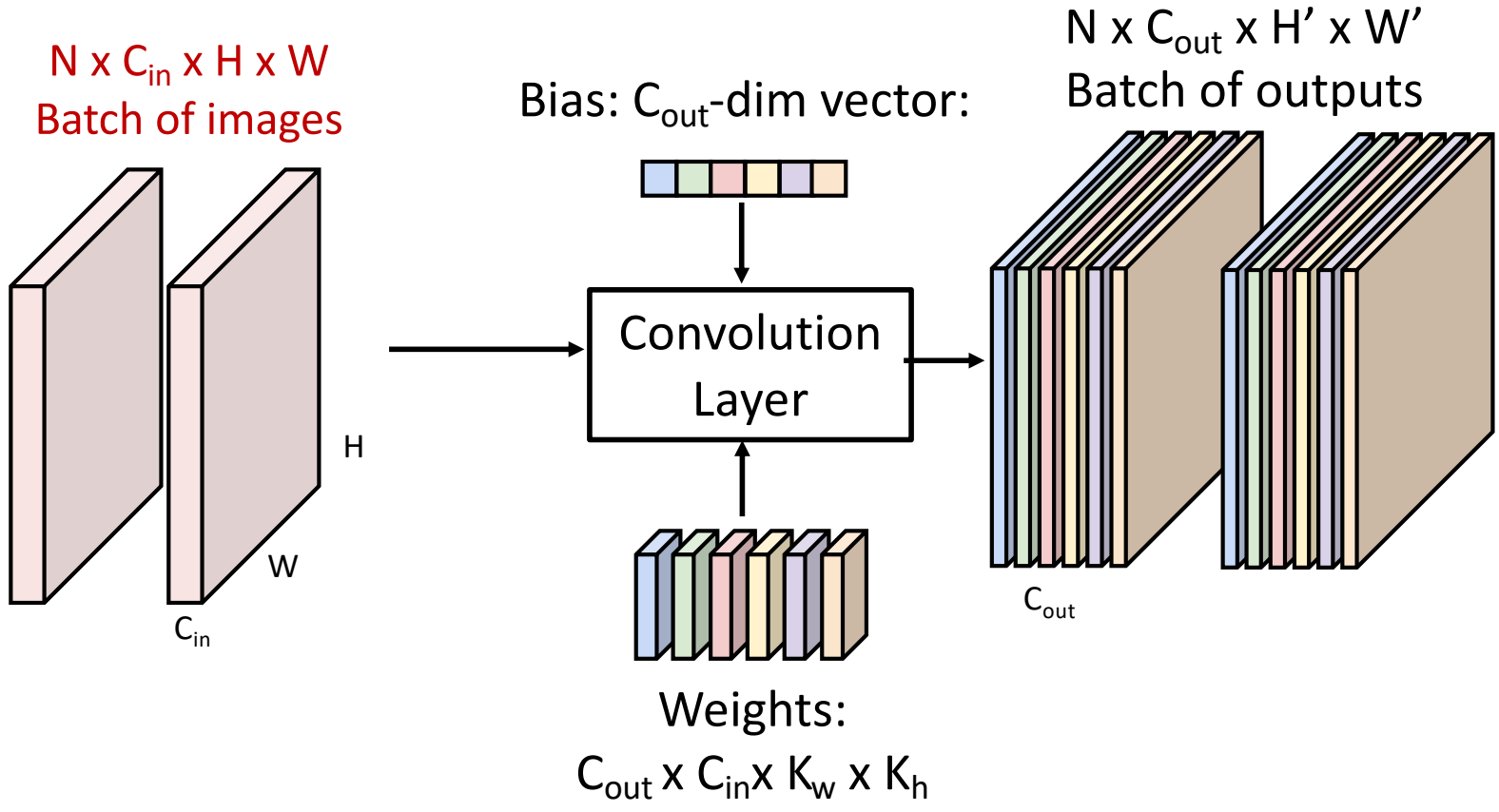
Convolution Layer



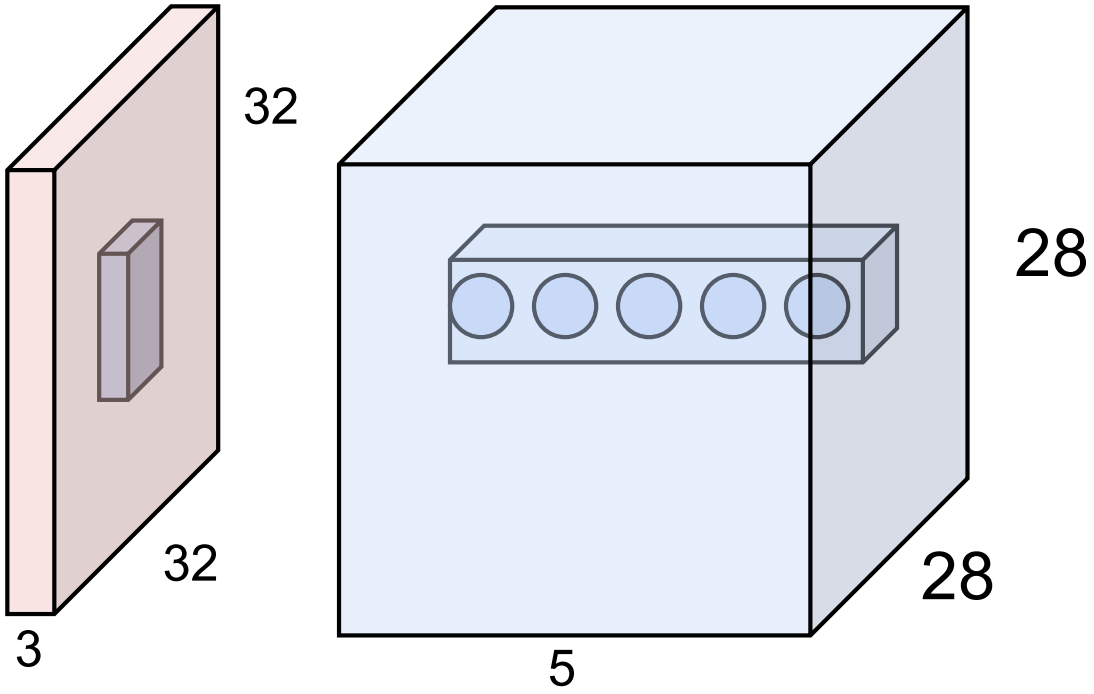
Convolution Layer: With many images

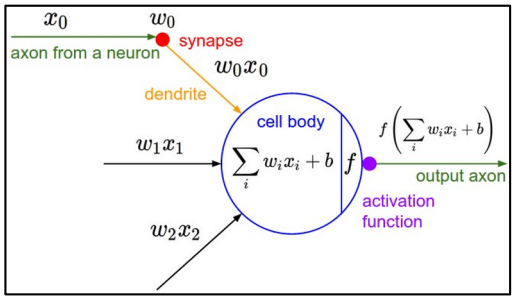


Convolution Layer



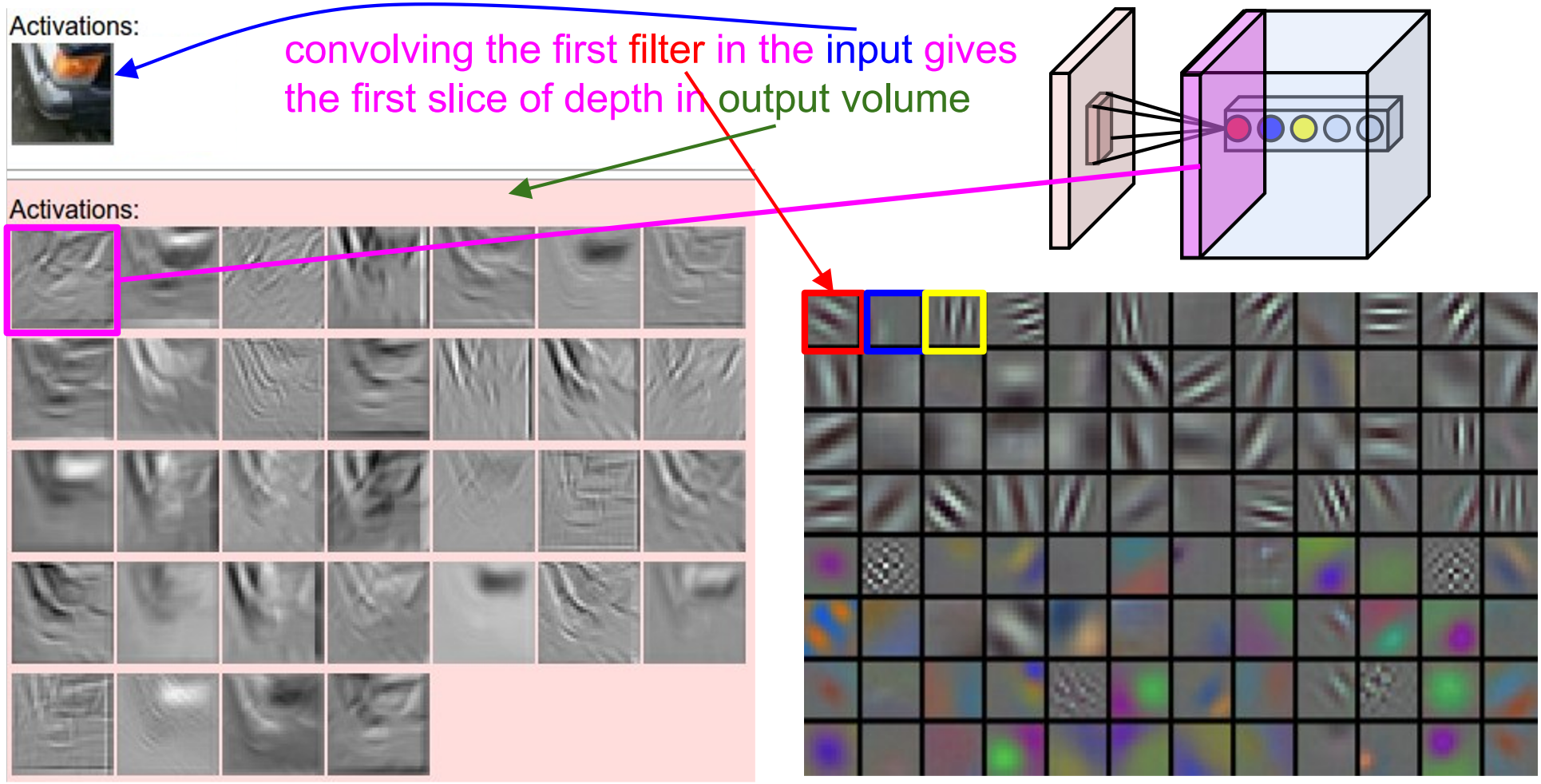
The brain/neuron view of CONV Layer



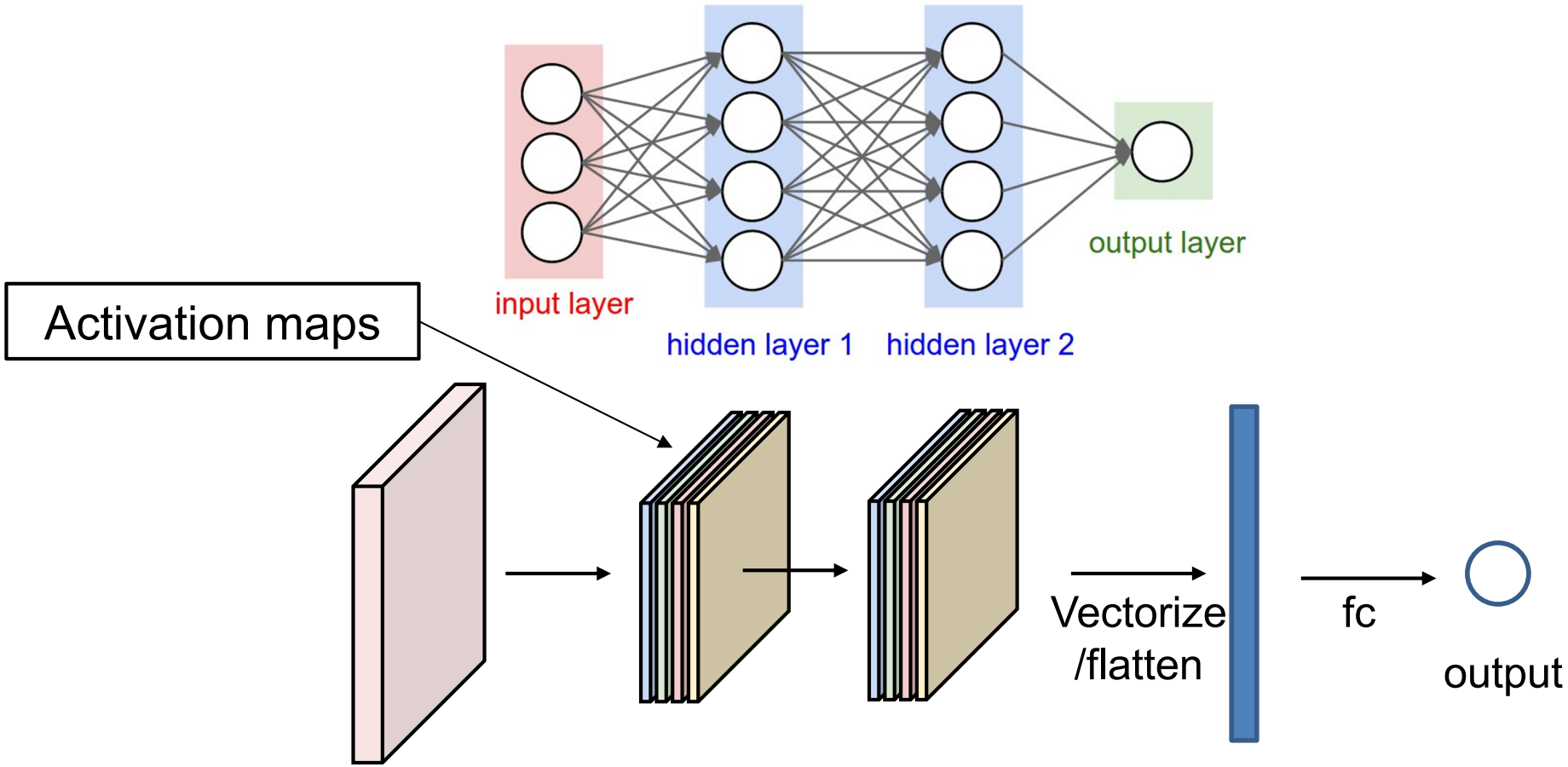


E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

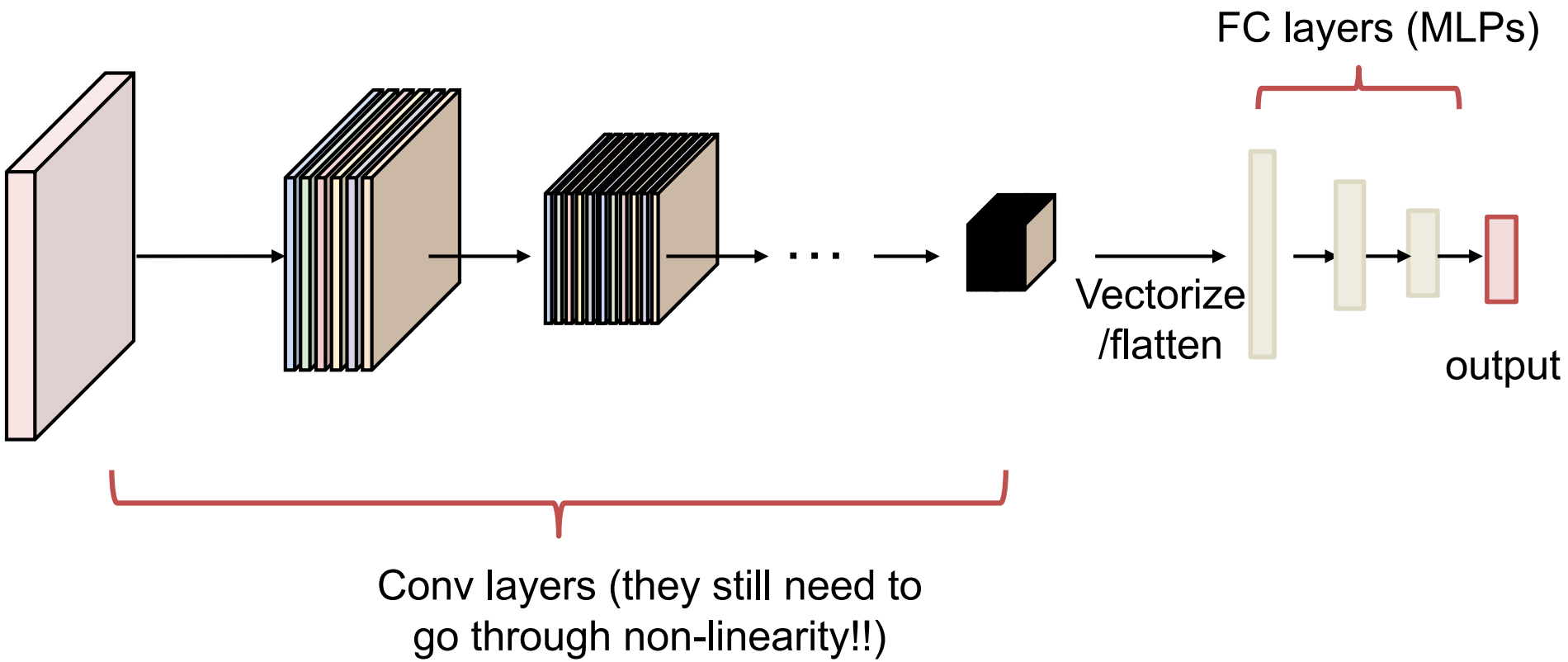
There will be 5 different neurons all looking at the same region in the input volume


Activations:
convolving the first filter in the input gives
the first slice of depth in output volume
Activations:

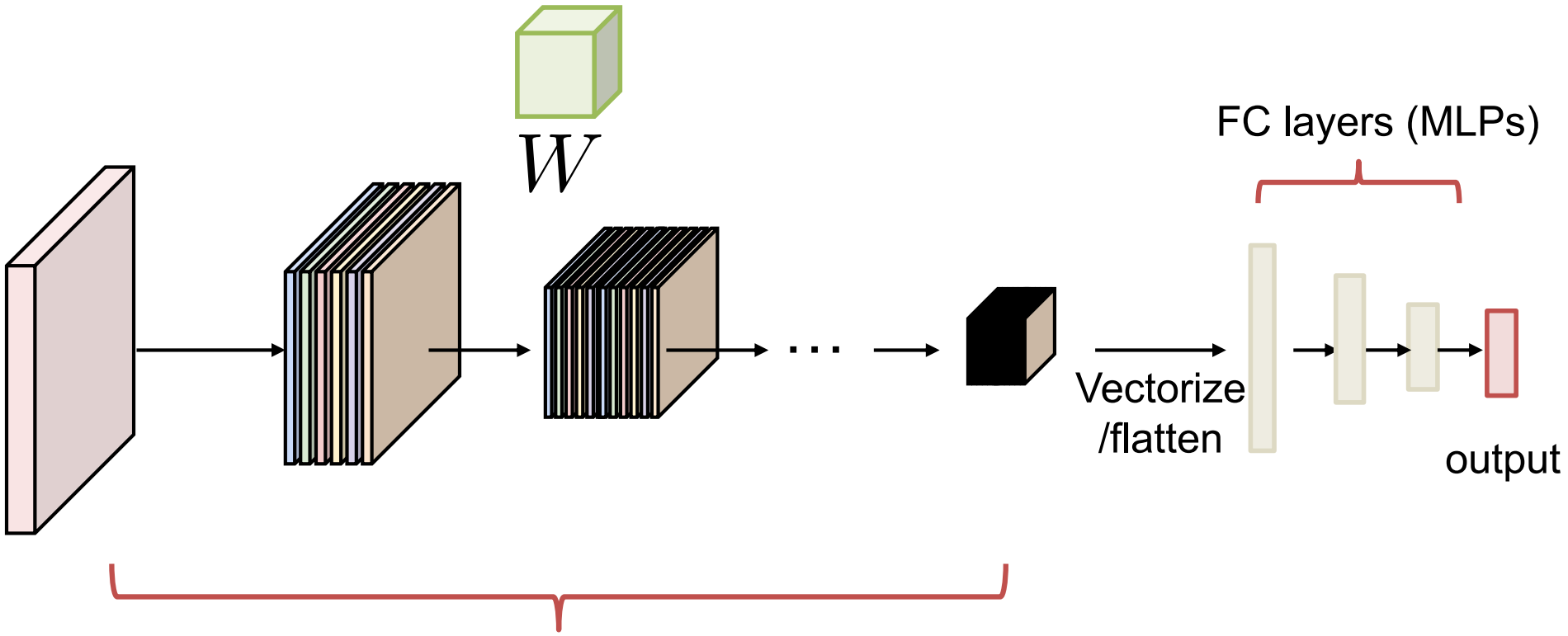
Compared with MLPs



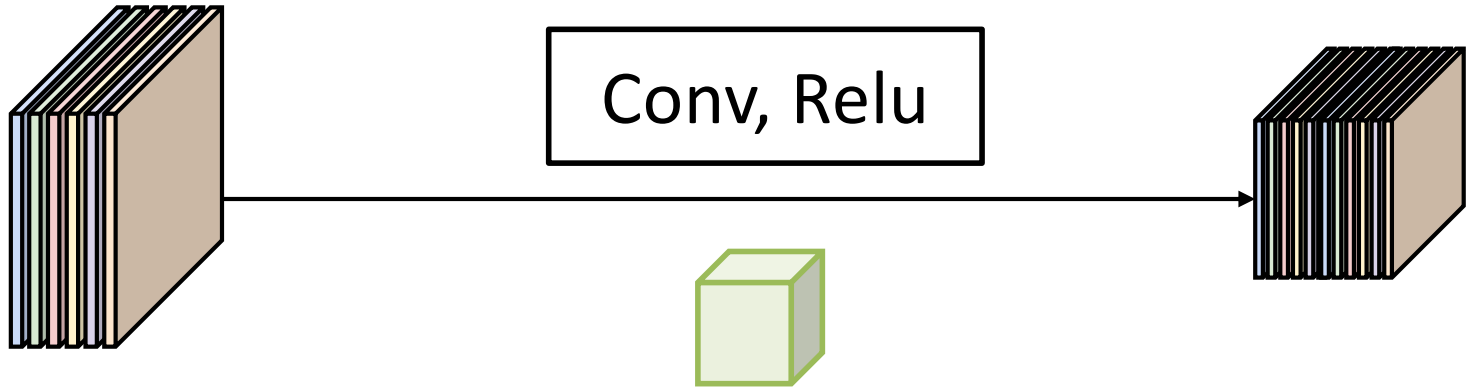
CNNs in practice



What needs to be learned?



What needs to be learned?



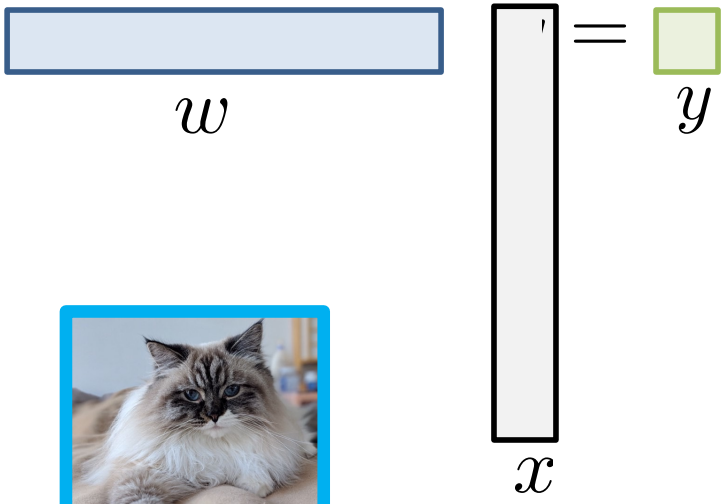
Input:
B, C_in, H, W

W
C_out, C_in, K, K

Output
B, C_out, H, W

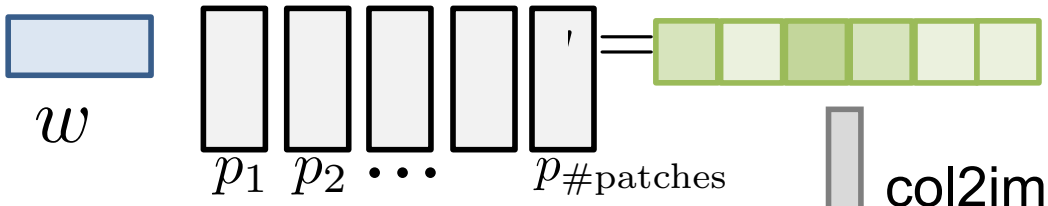
We're still doing matrix multiplications, just localized & shared

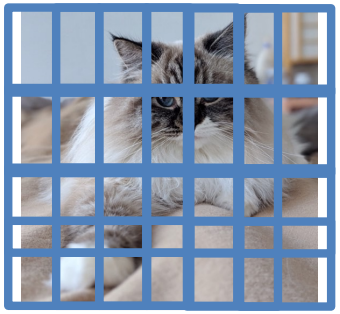
Recall one neuron in FC layer:



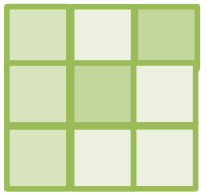
w takes the entire image!

With Conv layer:





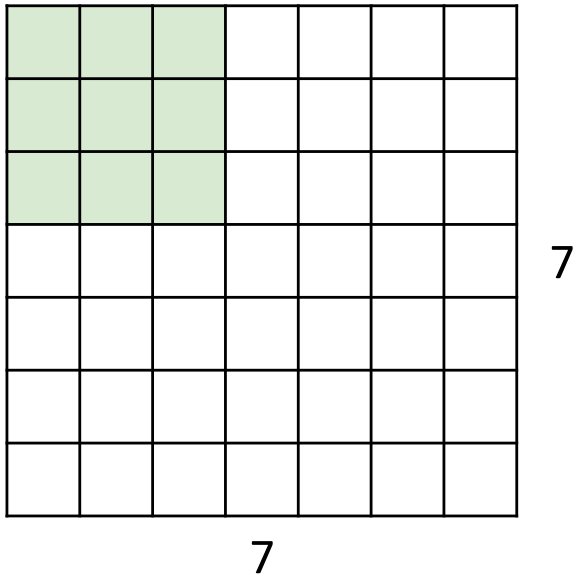
im2col



Now w takes (overlapping) patches

CONVNET NITTY GRITTIES

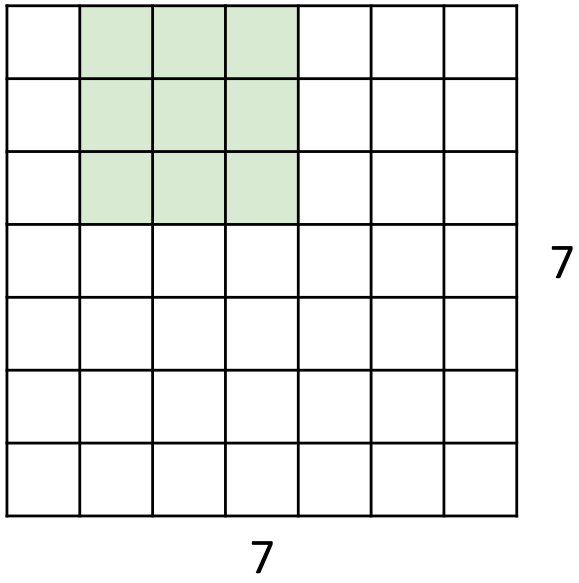
Convolution Spatial Dimensions



Input: 7x7
Filter: 3x3

Q: How big is output?

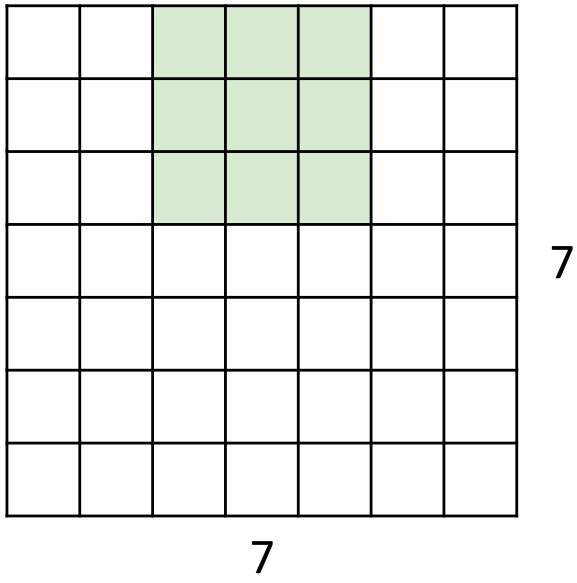
Convolution Spatial Dimensions



Input: 7x7
Filter: 3x3

Q: How big is output?

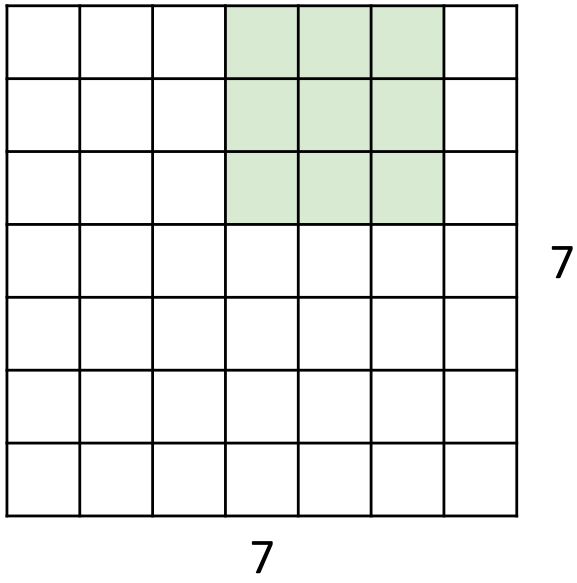
Convolution Spatial Dimensions



Input: 7x7
Filter: 3x3

Q: How big is output?

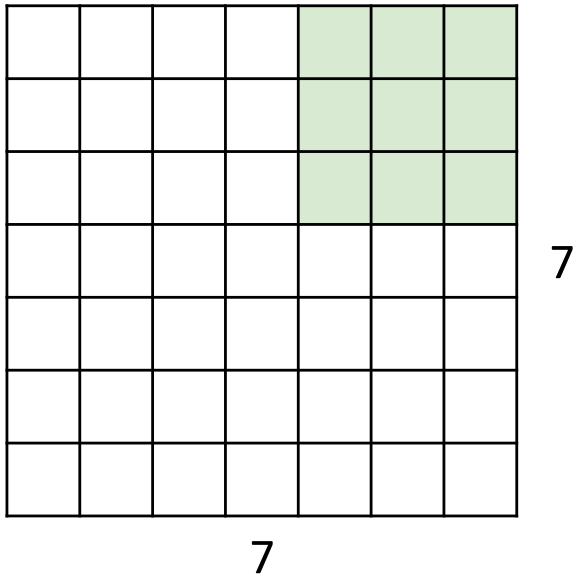
Convolution Spatial Dimensions



Input: 7x7
Filter: 3x3

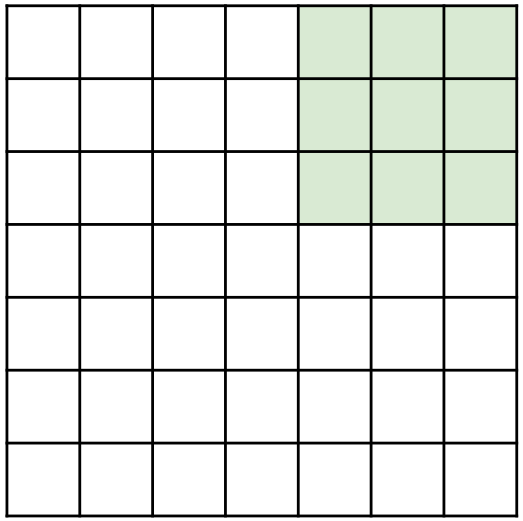
Q: How big is output?

Convolution Spatial Dimensions



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

Convolution 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!

Convolution 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

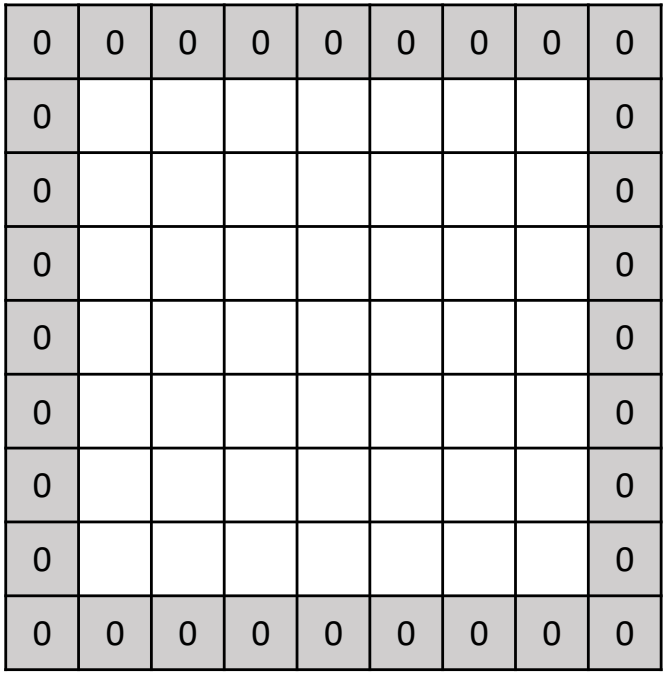
Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Padding: P

Problem:
Feature maps "shrink" with each layer!

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

Convolution Spatial Dimensions



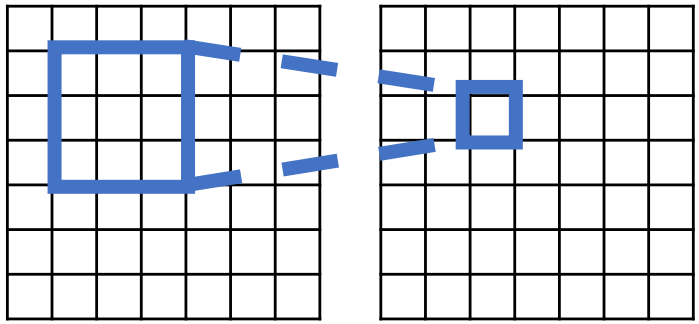
Input: 7x7
Filter: 3x3
Output: 5x5

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

Very common: “same padding”
Set P = (K – 1) / 2
Then output size = input size

Receptive Fields

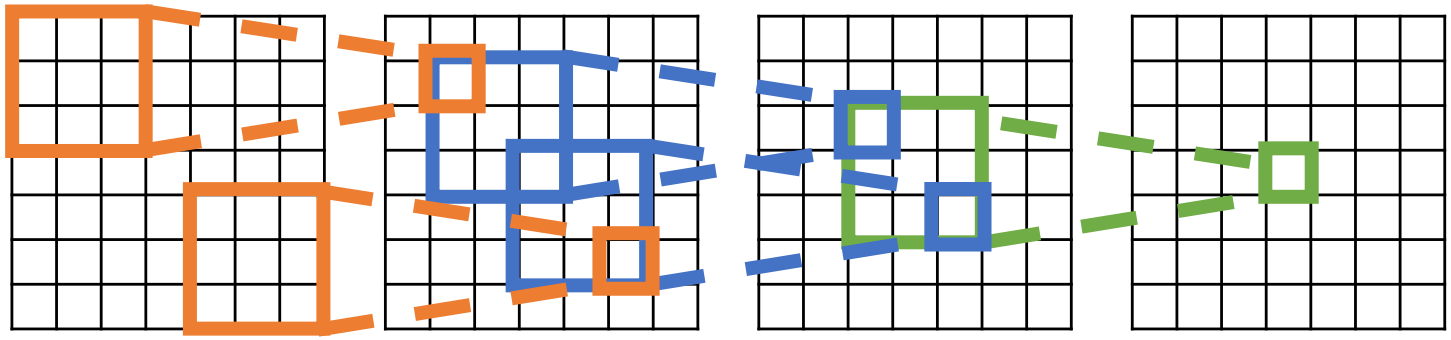
For convolution with kernel size K, each element in the output depends on a K x K **receptive field** in the input



Input Output

Receptive Fields

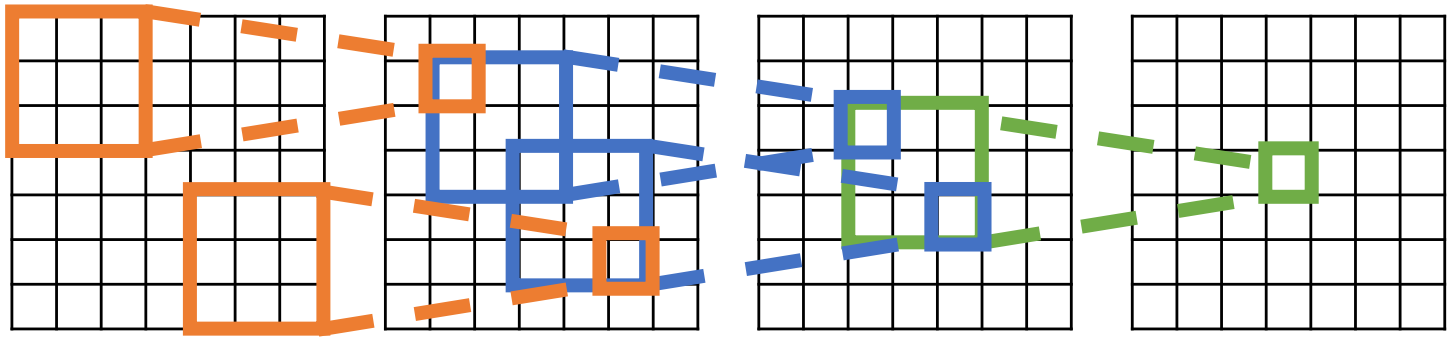
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



Careful – ”receptive field wrt to the input”
vs “receptive field wrt the previous layer”

Receptive Fields

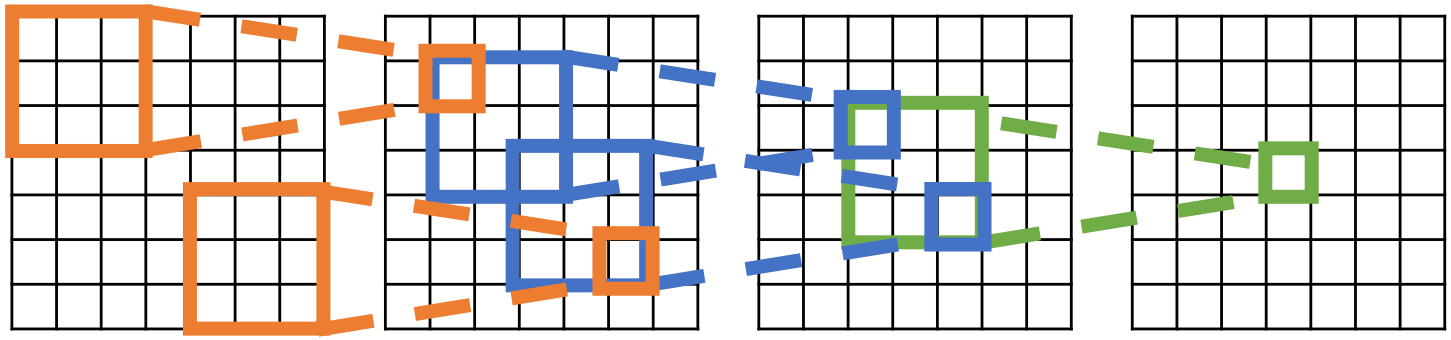
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



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

Receptive Fields

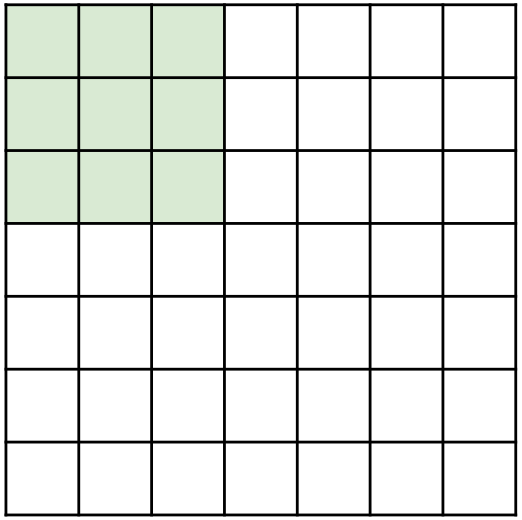
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



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

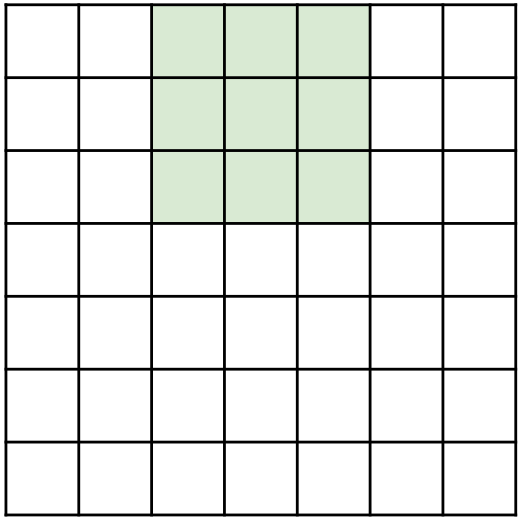
Solution: Downsample inside the network

Strided Convolution



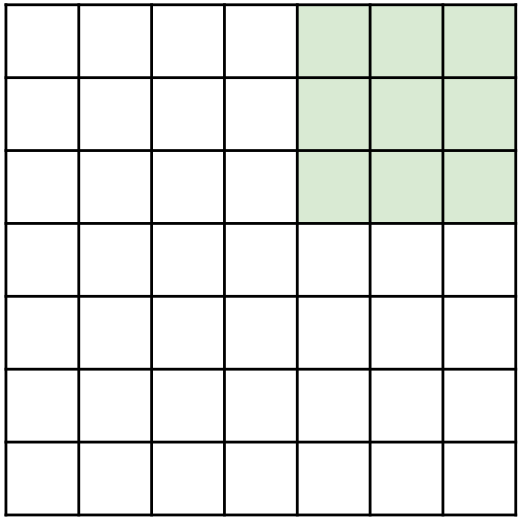
Input: 7x7
Filter: 3x3
Stride: 2

Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2

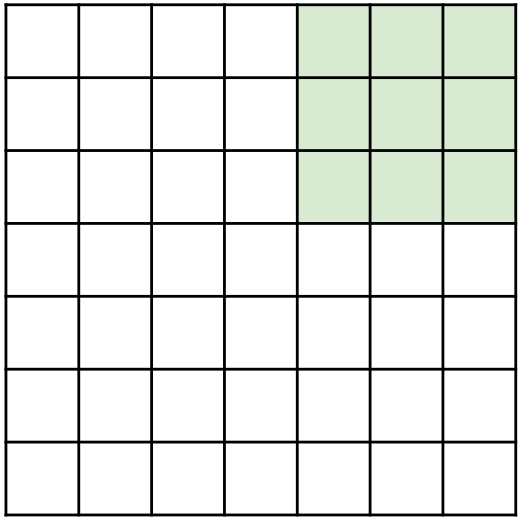
Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2

Output: 3x3

Strided Convolution

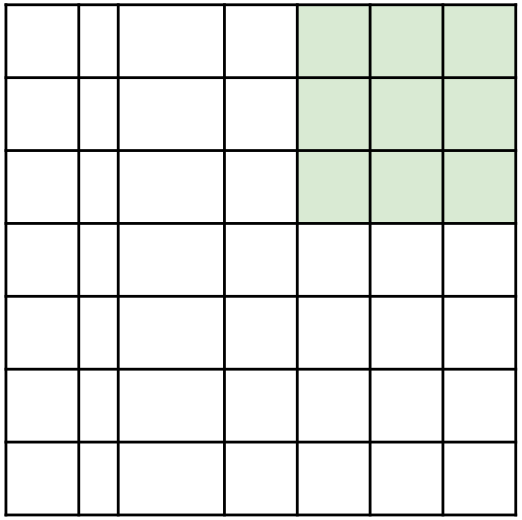


Input: 7x7
Filter: 3x3
Stride: 2

Output: 3x3

In general:
Input: W
Filter: K
Padding: P
Stride: S
Output: (W – K + 2P) / S + 1

Can we do stride 3?



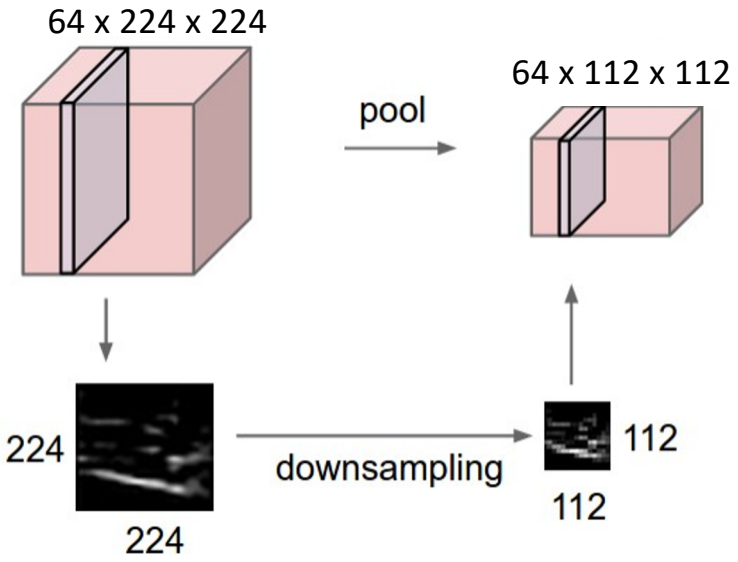
Input: 7x7
Filter: 3x3
Stride: 3?

Output: 3x3

Output: (N-F) / S + 1

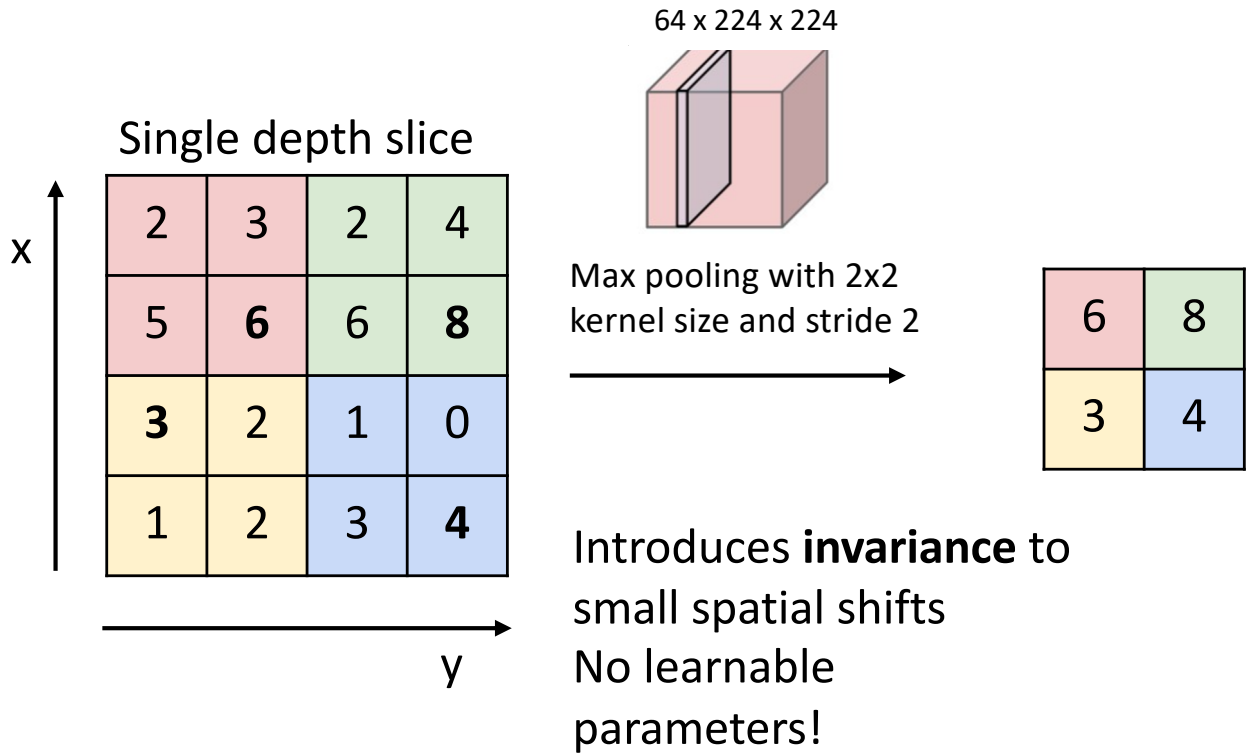
e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 ☹

Pooling Layers: Downsampling



Hyperparameters:
Kernel Size
Stride
Pooling function

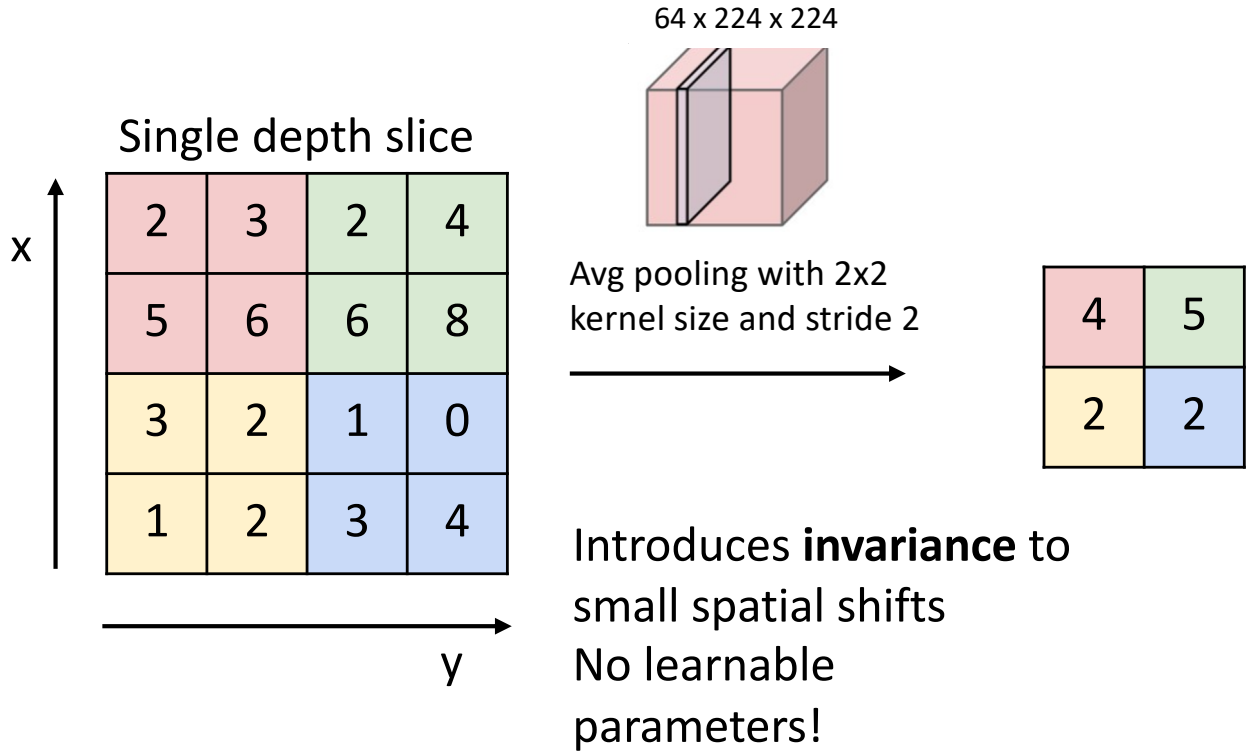
Max Pooling



Introduces **invariance** to small spatial shifts

No learnable parameters!

Average Pooling



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)

Normalization

Why do we need to normalize our data?

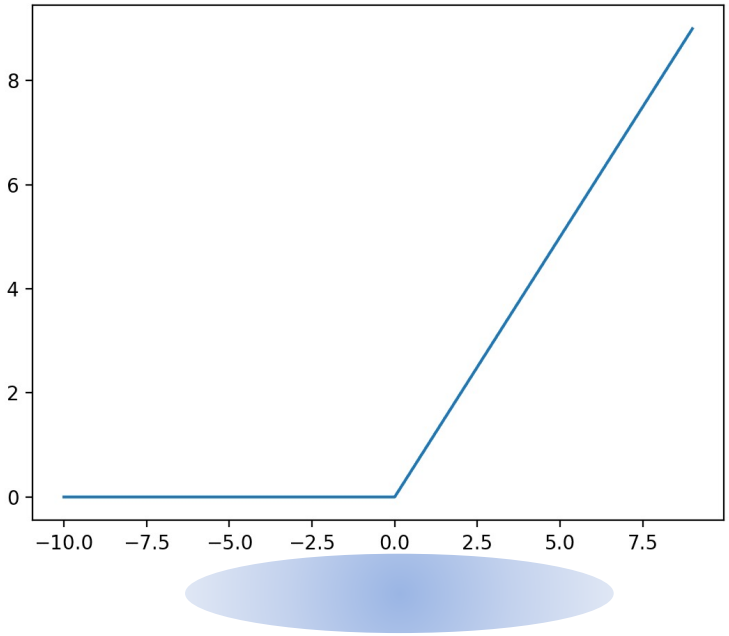
Extreme example:

- sometimes pixel range is [0, 255], sometimes it's [0, 1]

What is the problem?

Network activations will be completely different!!

You want the inputs to be in a similar range → low variance



How to normalize

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

Is this differentiable?

Yes! so we can use it as an operator in our networks and backprop through it!

But now the data is always zero mean, unit variance…

Sol: Add scale and shift parameters: γ, β

$$y = \gamma\hat{x} + \beta$$

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

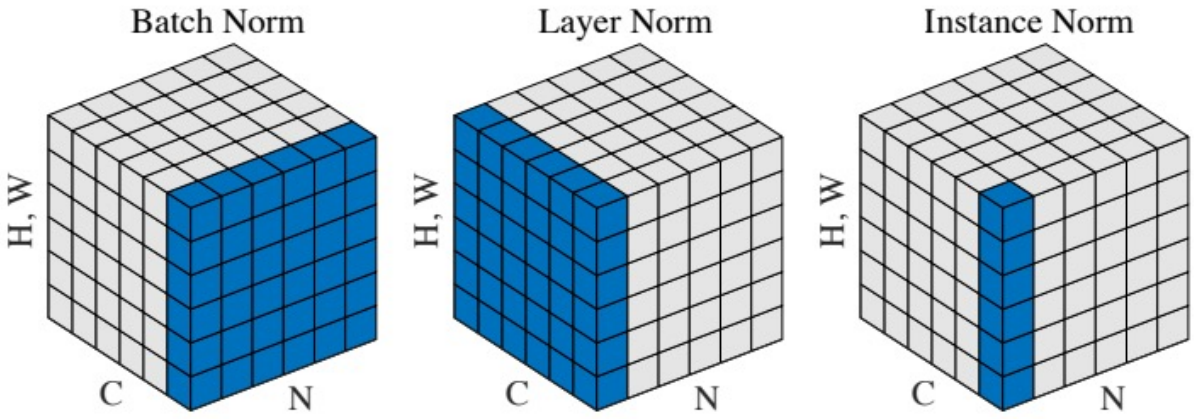
Slide modified from David Fouhey

How to normalize

Next Q: from **what** do you compute the mean & variance?

- BatchNorm computes mean and var from each batch
- Depends on the batch, so need to keep a running average and store these.

- LayerNorm/InstanceNorm do not need this.



Batch Normalization for ConvNets

Batch Normalization for
fully-connected networks

$$x : N \times C$$

Normalize ↓

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

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

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

Batch Normalization for
convolutional networks
(Spatial Batchnorm, BatchNorm2D)

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

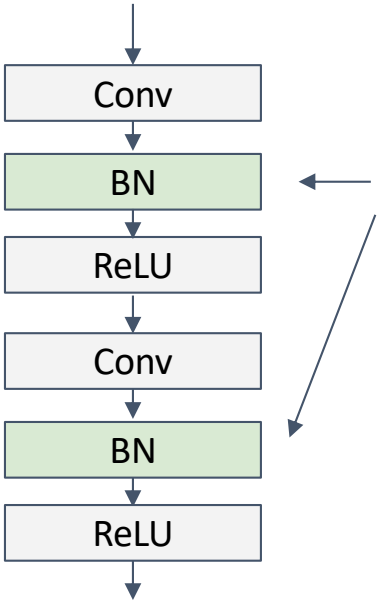
Normalize ↓ ↓ ↓

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

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

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

Normalization



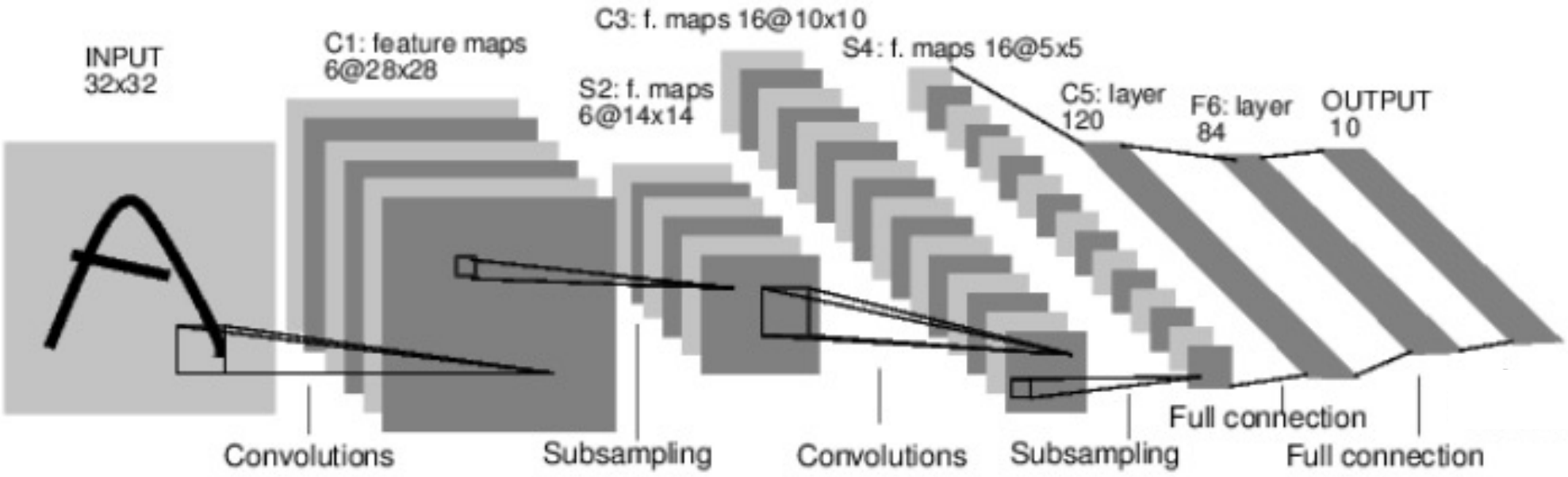
Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

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

Case Study: Lenet-5

Task: 10 digit classification

LeCun et al. 1998



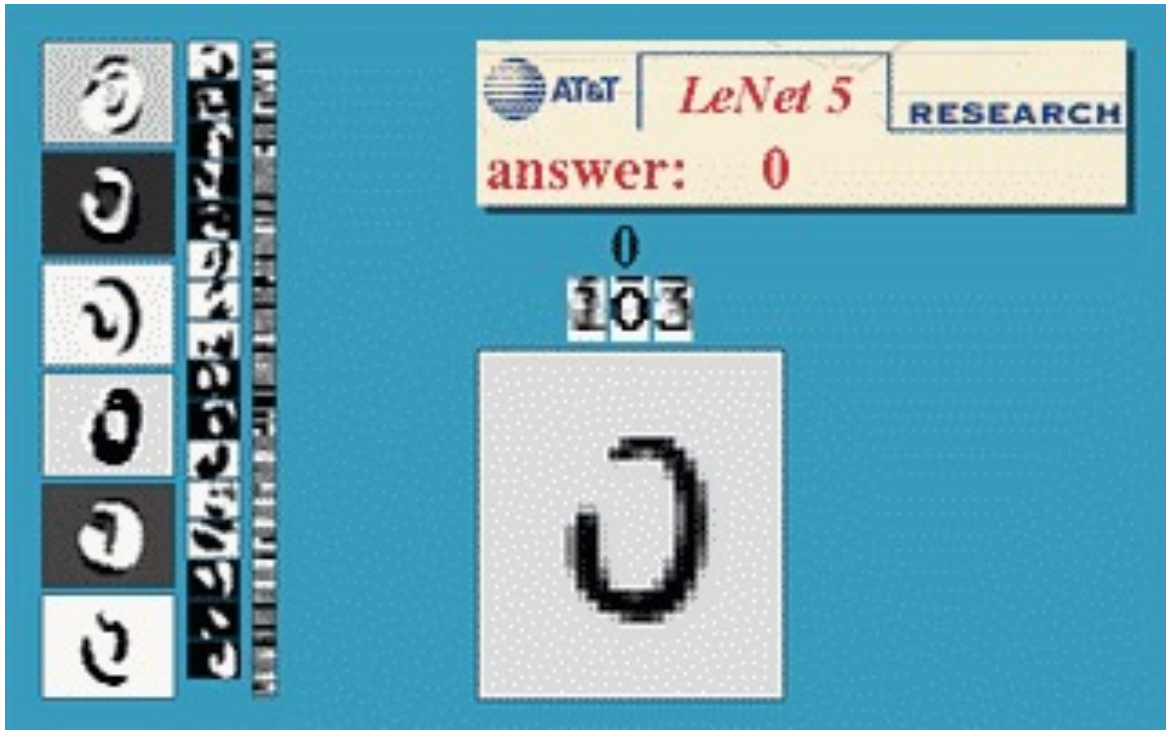
Conv filters were 5x5, applied at stride 1

Subsampling (Pooling) layers were 2x2 applied at stride 2

i.e. architecture is [CONV-POOL-CONV-POOL-CONV-flatten-FC-output]

LeNet5 demo

LeCun et al. 1998

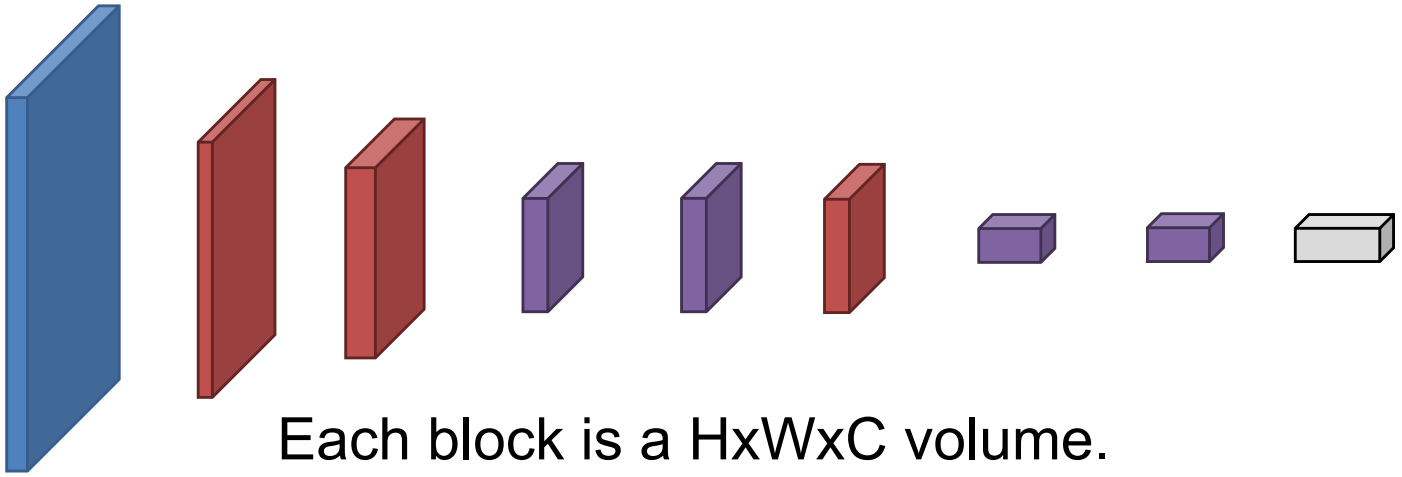


Case Study: AlexNet

[Krizhevsky, Sutskever, Hinton, NeurIPS 2012]

Task: ImageNet 1000-class classification

	Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
HxW	227x 227	55x55	27x27	13x13	13x13	13x13	1x1	1x1	1x1
C	3	96	256	384	384	256	4096	4096	1000

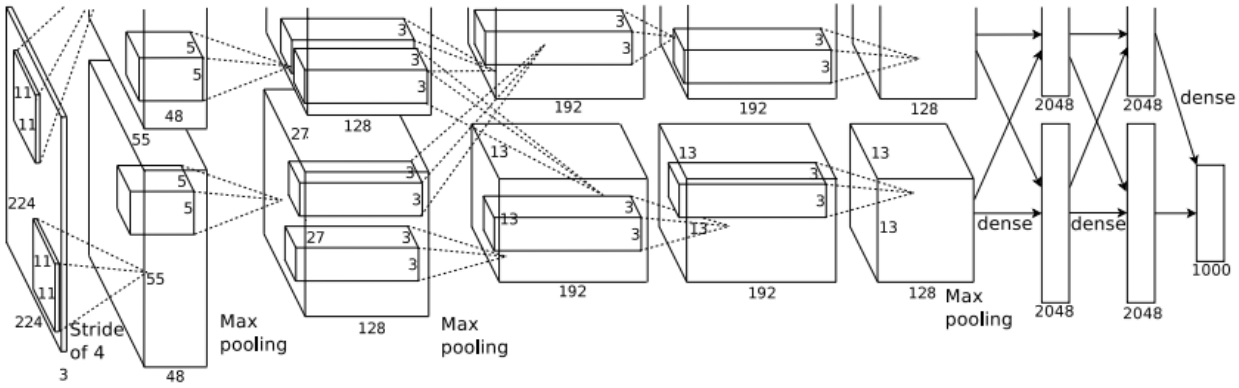


Each block is a HxWxC volume.

You transform one volume to another with convolution

Case Study: AlexNet

[Krizhevsky, Sutskever, Hinton, NeurIPS 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

Hint:
(227-11)/4 + 1 = 55

Q: What is the output volume size after Conv1?

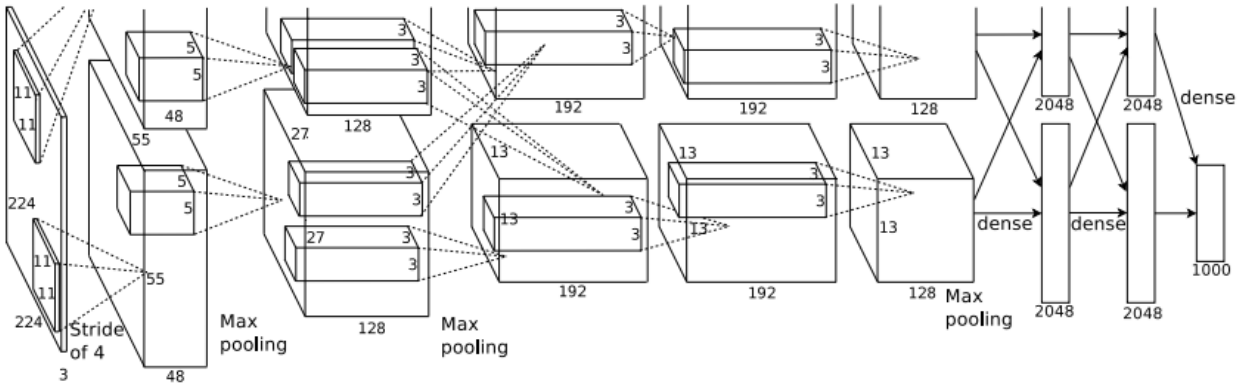
A: 55x55x96

Q: how many parameters in this layer?

A: (3*11*11) * 96

Case Study: AlexNet

[Krizhevsky, Sutskever, Hinton, NeurIPS 2012]



Input: 227x227x3 images
After Conv1: 55x55x96
Second layer: Pool1: 3x3 at stride 2

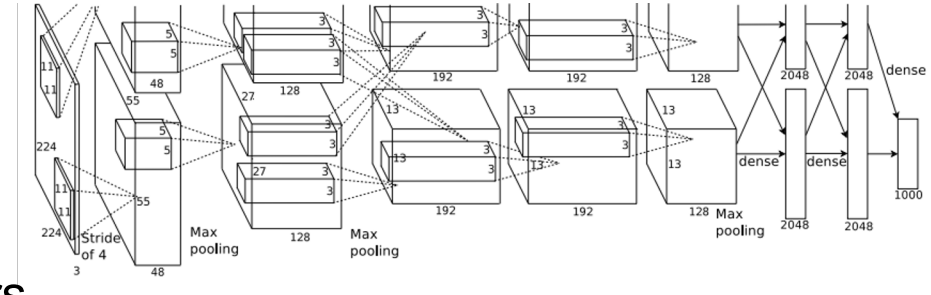
Q: What is the output volume size after Pool1?

A: (55-3)/2 + 1 = 27
27x27x96

Q: how many parameters in this layer?

A: 0!!!

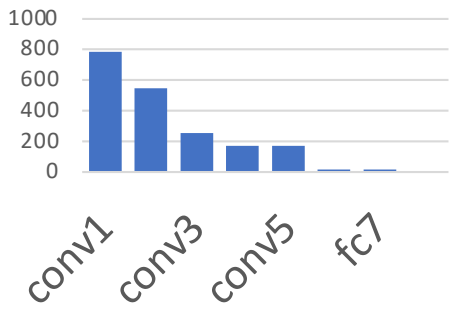
AlexNet



Q: Which part of the network incurs

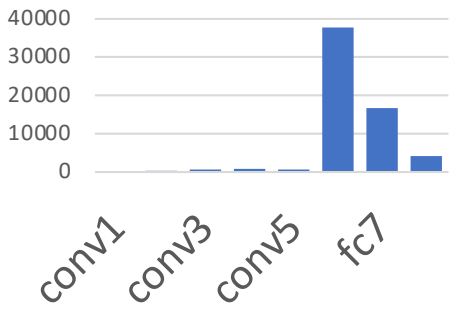
high memory usage?

Most of the **memory usage** is in the early convolution layers



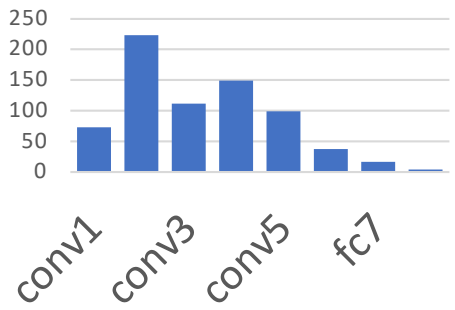
Large # of parameters?

Nearly all **parameters** are in the fully-connected layers



high FLOPs?

Most **floating-point ops** occur in the convolution layers



In pytorch

https://github.com/pytorch/vision/blob/main/torchvision/models/alexnet.py#L18

```
18  class AlexNet(nn.Module):
19      def __init__(self, num_classes: int = 1000, dropout: float = 0.5) -> None:
20          super(AlexNet, self).__init__()
21          _log_api_usage_once(self)
22          self.features = nn.Sequential(
23              nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
24              nn.ReLU(inplace=True),
25              nn.MaxPool2d(kernel_size=3, stride=2),
26              nn.Conv2d(64, 192, kernel_size=5, padding=2),
27              nn.ReLU(inplace=True),
28              nn.MaxPool2d(kernel_size=3, stride=2),
29              nn.Conv2d(192, 384, kernel_size=3, padding=1),
30              nn.ReLU(inplace=True),
31              nn.Conv2d(384, 256, kernel_size=3, padding=1),
32              nn.ReLU(inplace=True),
33              nn.Conv2d(256, 256, kernel_size=3, padding=1),
34              nn.ReLU(inplace=True),
35              nn.MaxPool2d(kernel_size=3, stride=2),
36          )
37          self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
38          self.classifier = nn.Sequential(
39              nn.Dropout(p=dropout),
40              nn.Linear(256 * 6 * 6, 4096),
41              nn.ReLU(inplace=True),
42              nn.Dropout(p=dropout),
43              nn.Linear(4096, 4096),
44              nn.ReLU(inplace=True),
45              nn.Linear(4096, num_classes),
46          )
47
```

VGGNet [Simonyan and Zisserman 2015]

Input: 224x224x3

Simplified design rules:

- All kernel size 3x3
- Always ReLu
- All max pool are 2x2, stride 2
- After pool, double the # of channels

The ReLU activation function is not shown for brevity.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

VGGNet [Simonyan and Zisserman 2015]

Input: 224x224x3

Simplified design rules:

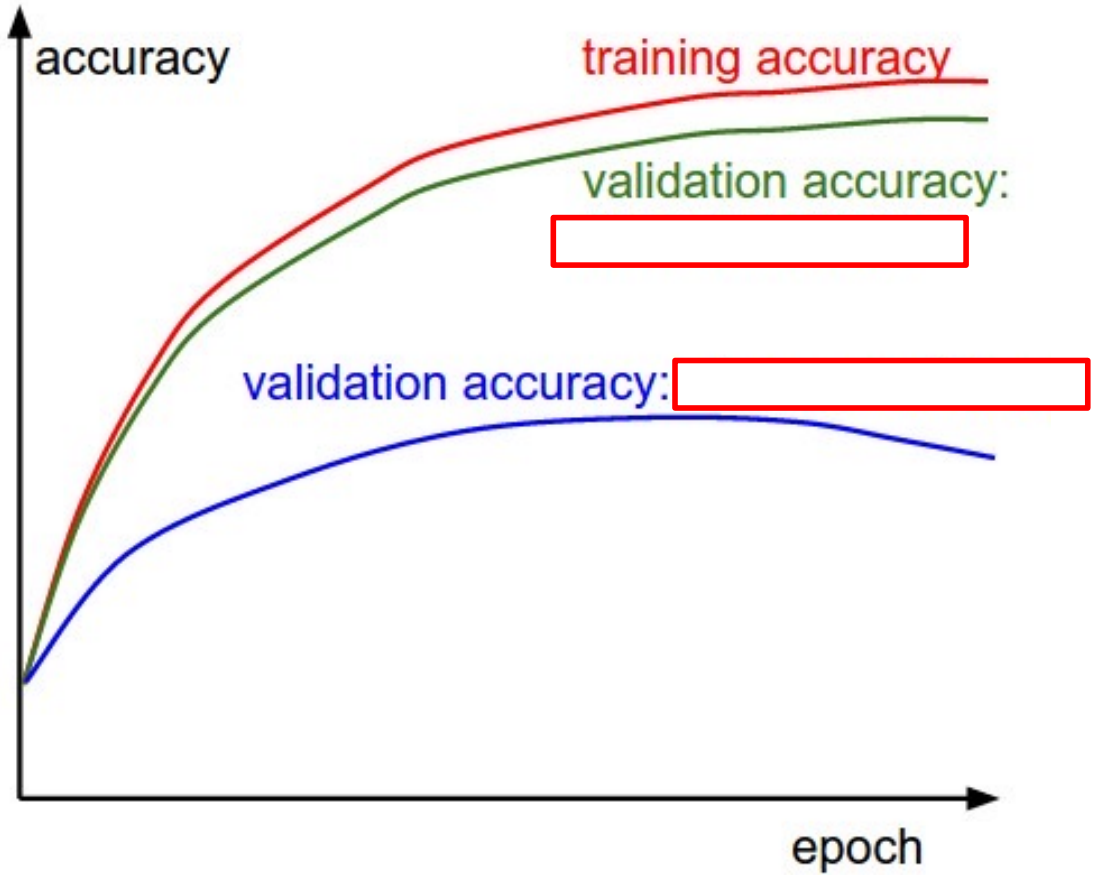
- **All kernel size 3x3**
- Always ReLu
- All max pool are 2x2, stride 2
- After pool, double the # of channels

Two 3x3 conv has same receptive field as a single 5x5 conv, but has fewer parameters and takes less computation!

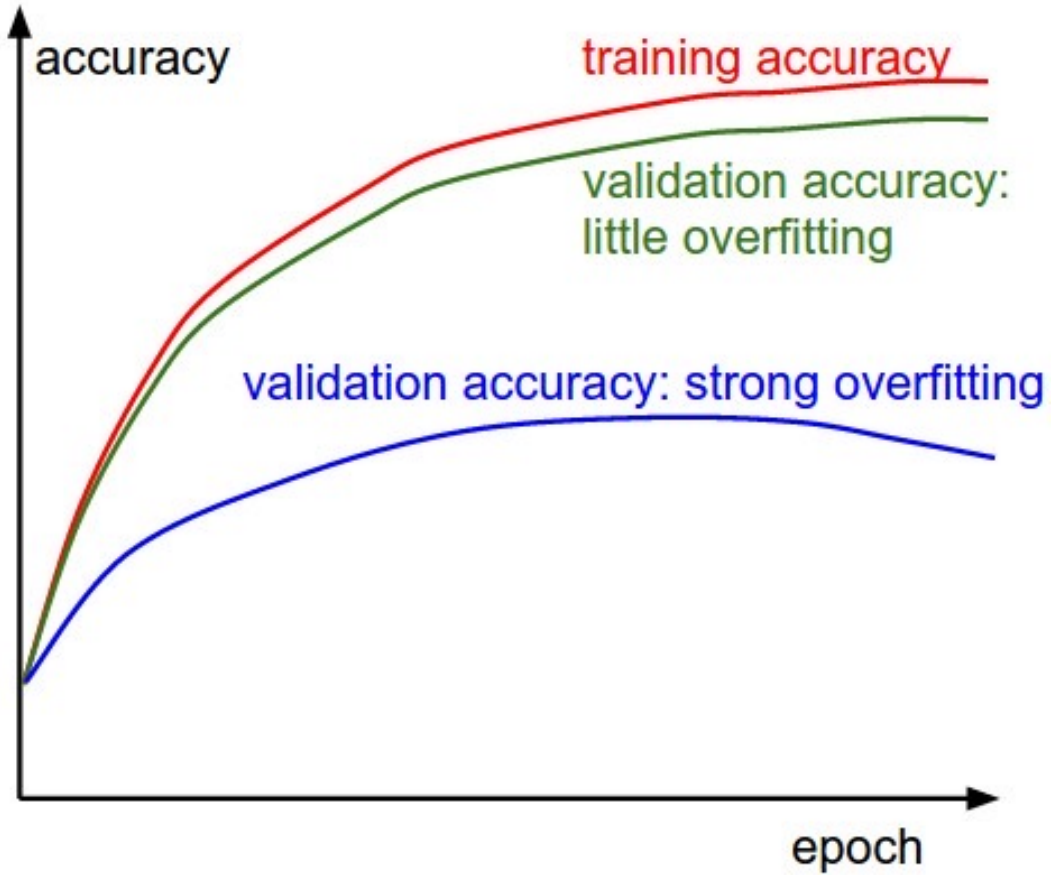
Option 1:	Option 2:
Conv(5x5, C -> C)	Conv(3x3, C -> C) Conv(3x3, C -> C)
Params: $25C^2$	Params: $18C^2$
FLOPs: $25C^2HW$	FLOPs: $18C^2HW$

Slide adapted from Johnson & Fouhey

Measuring performance over train/val sets



Measuring performance over train/val sets



Naively adding more layers != better performance

Q: is this over fitting?

No! (hypothesis) it's a matter of optimization

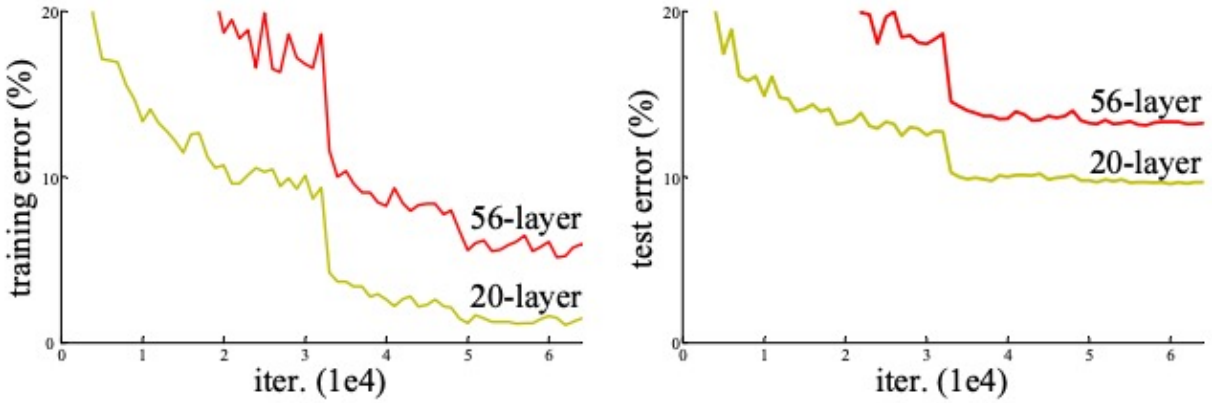


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

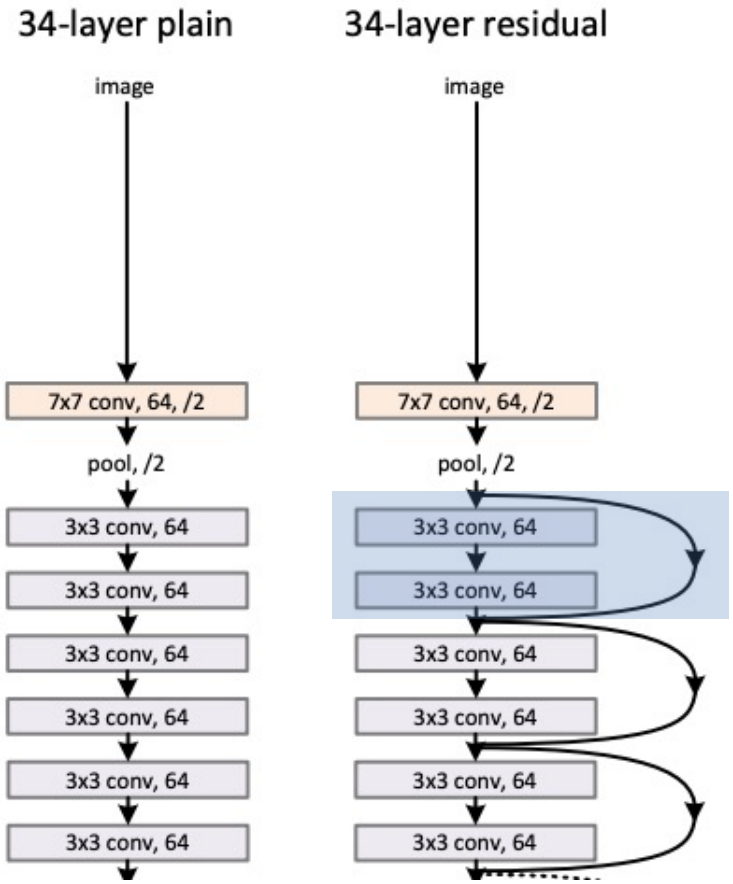
CVPR 2016]

Naively adding more layers != better performance

Deeper networks can emulate shallow networks.
How?
By making the extra layers to model the identity function.
But it's difficult to do this as y= sig(w*x + b)

[He et al. CVPR 2016]

ResNet [He et al. CVPR 2016]



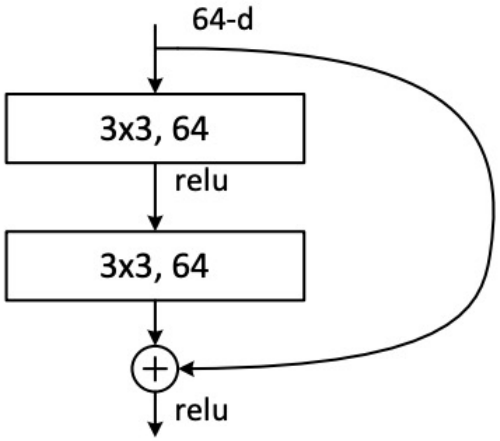
Input: 224x224x3

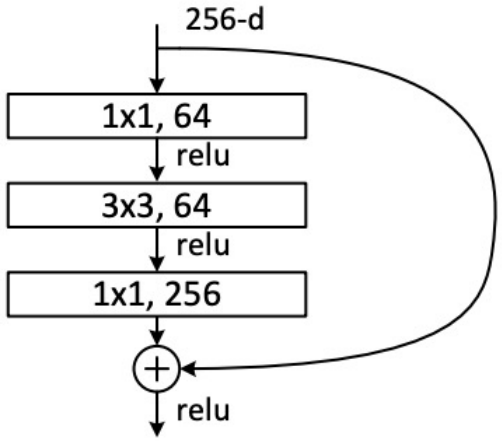
ResNet Block:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}.$$

ResNet [He et al. CVPR 2016]

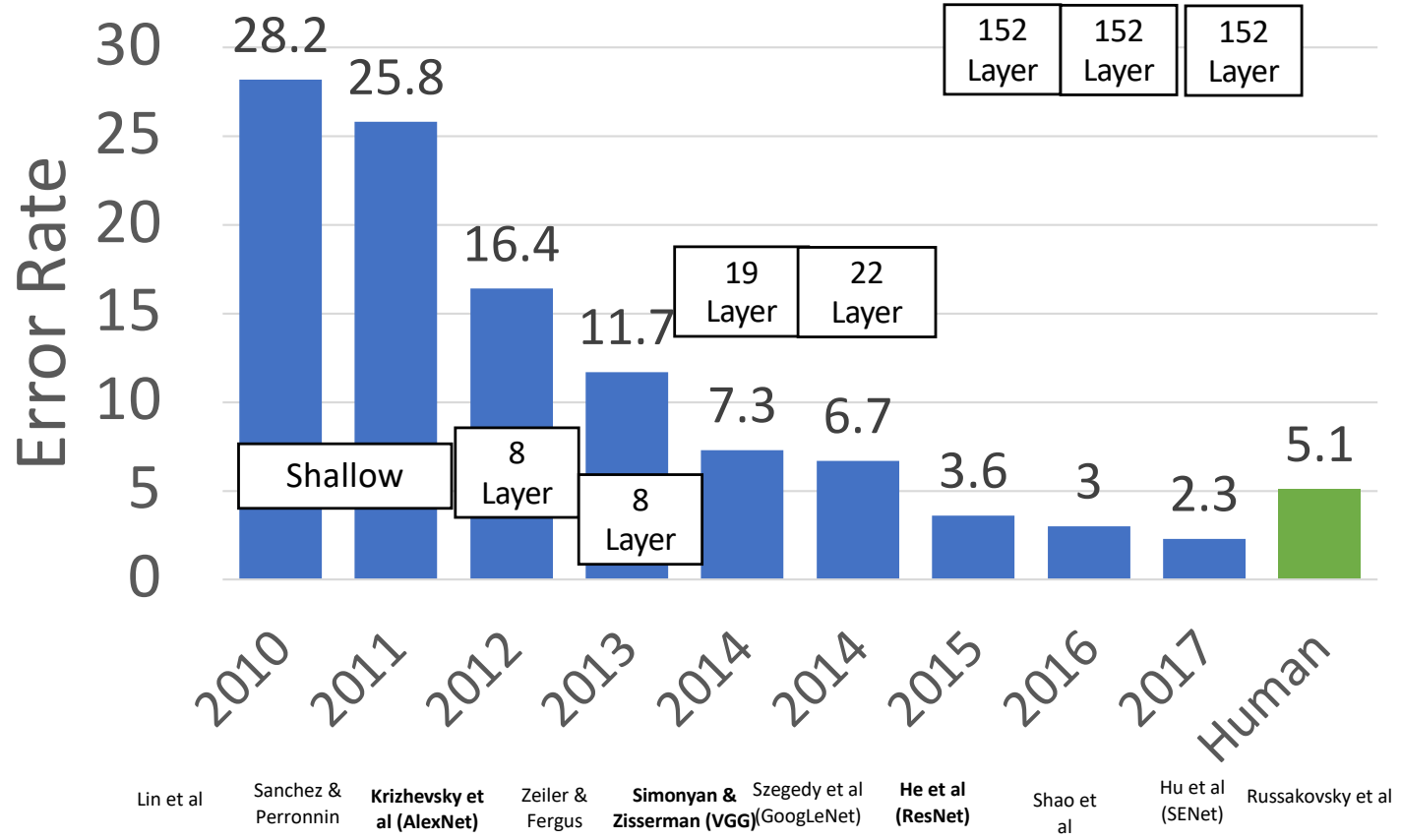
Can be made very deep: ResNet-50, 101, 152
And converge well



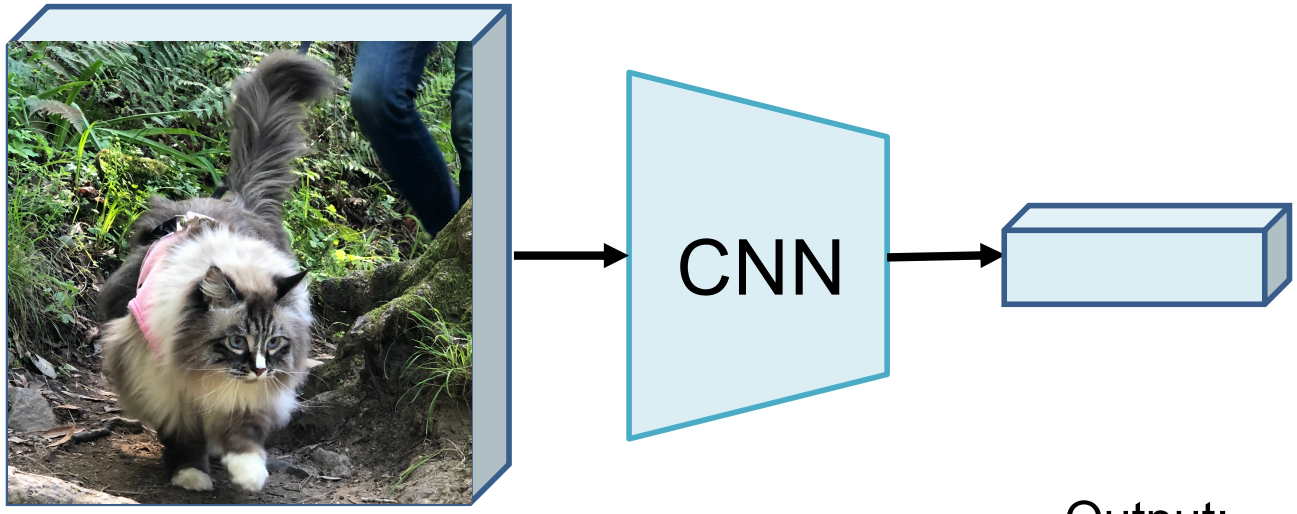


Bottleneck design:
Reduce by 1x1
Conv by 3x3
Restore by 1x1

ImageNet Classification Challenge top-5 error



Bottom line: CNN workflow



Input:
HxWxC

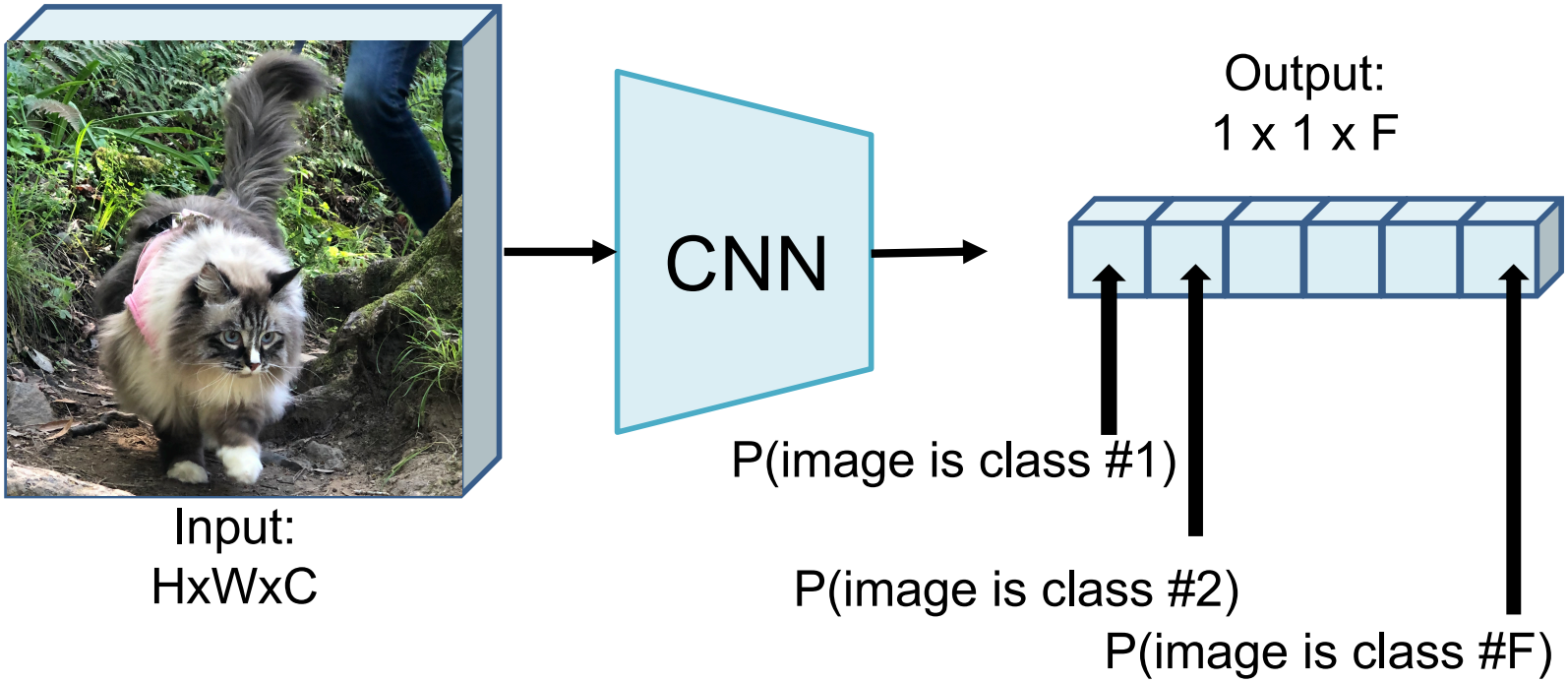
Output:
1 x 1 x F

Convert HxW image into a F-dimensional vector

- What's the probability this image has a cat? (F=1)
- Which of 1000 categories is this image? (F=1000)
- Where are the X,Y coordinates of 5 cat face keypoints? (F=28*2 = 56)

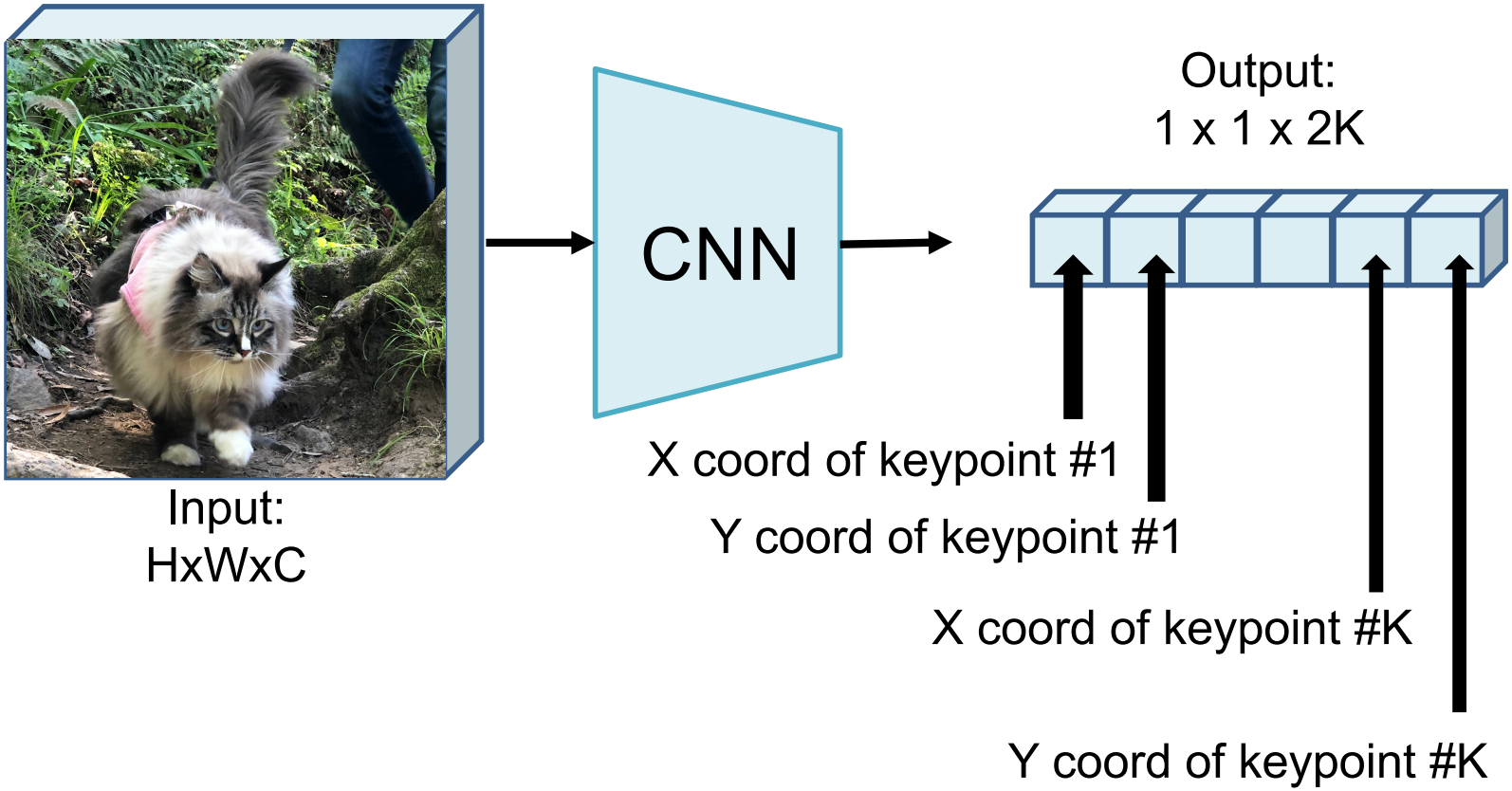
Slide modified from David Fouhey

Example: Image classification



Slide modified from David Fouhey

Example: Keypoint detection



Slide modified from David Fouhey

What if your data is not big enough?

Horizontal Flip

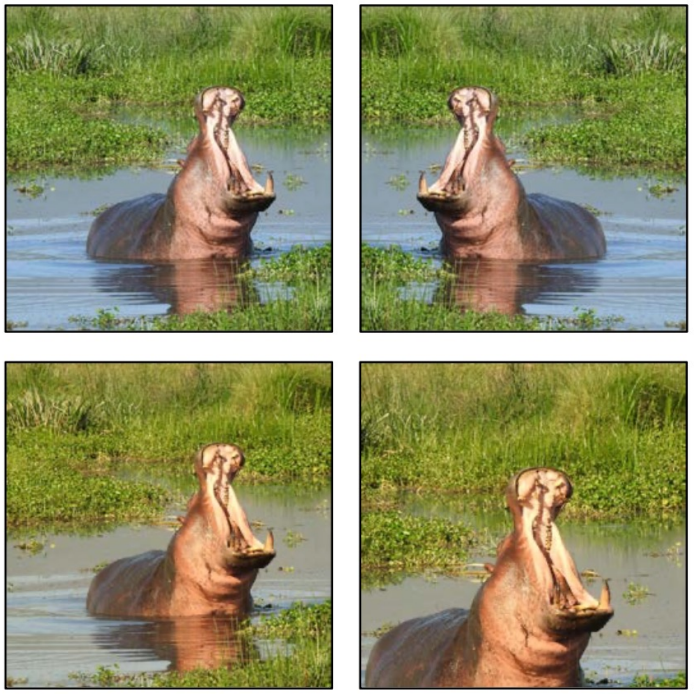
Color Jitter

Image Cropping

Slide by David Fouhey

Data Augmentation

- Apply transformations (on the fly) to increase training data
- Can mix multiple transformations at once
- Make sure you don't change the meaning of the output



Slide modified from David Fouhey

Sometimes labels need to be transformed too

For certain tasks, make sure to transform the output accordingly!!

Original

After flip + color aug

Right eye Left eye

Defined wrt to the cat

Sometimes labels need to be transformed too

For certain tasks, make sure to transform the output accordingly!!

Original

After flip + color aug

Right eye Left eye

Defined wrt to the cat

Very common bug

For certain tasks, make sure to transform the output accordingly!!

Right eye Left eye

Defined wrt to the cat

Original

After flip + color aug

WRONG! Flipped without updating the label!!

Very common bug

For certain tasks, make sure to transform the output accordingly!!

Right eye Left eye

Defined wrt to the cat

Original

After flip + color aug

Correct

How to train your network

Basic setup

1. Get your dataset
2. Design your CNN architecture
3. Train + update weights with pytorch/tf…

Sanity Check #1

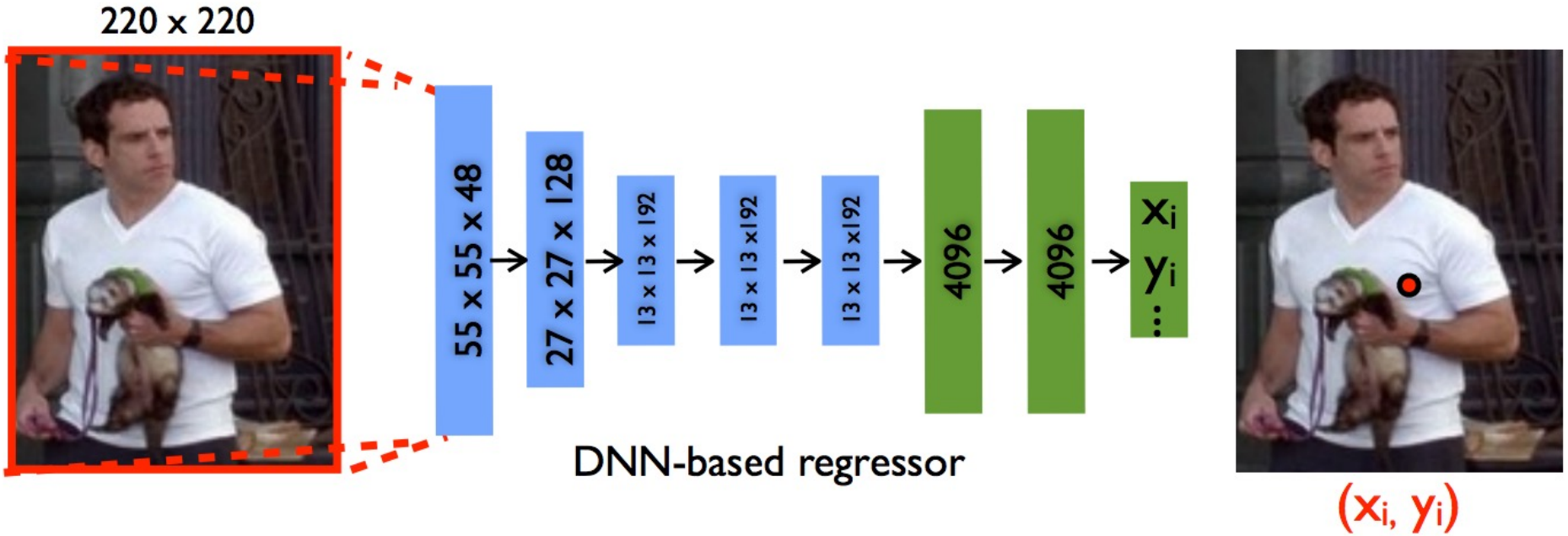
Make sure you can memorize a single data point: (x_i, y_i).

- Train your CNN on only 1 image
- It **must** be able to get very low training loss

If you can't do this, it means something is wrong in your loss function, label, network/pytorch setup

It should also overfit to a single data fairly quickly. If it takes many iterations to do this that's also bad news bears

A good starting point



DeepPose: Human Pose Estimation via Deep Neural Networks
[Toshev and Szegedy 2014]