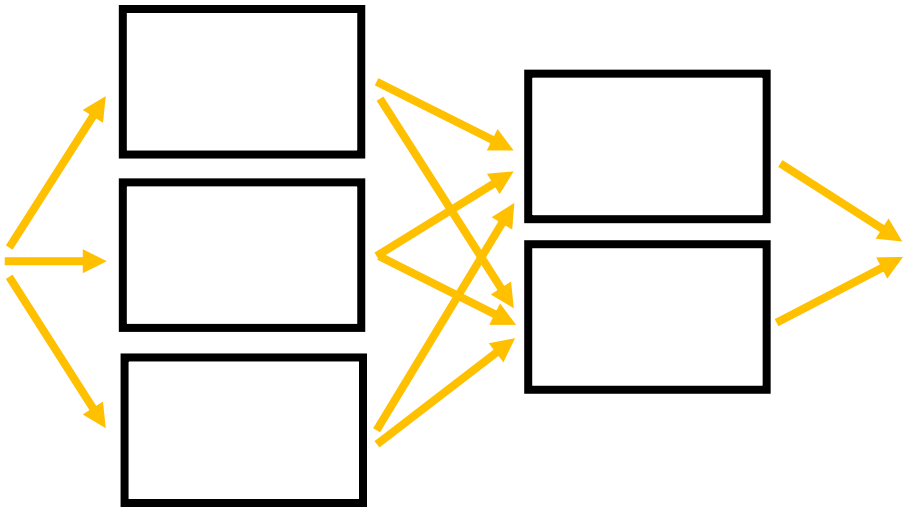
Convolutional Neural Networks



CS194: Computer Vision and Comp. Photo

Angjoo Kanazawa/Alexei Efros, UC Berkeley, Fall 2022

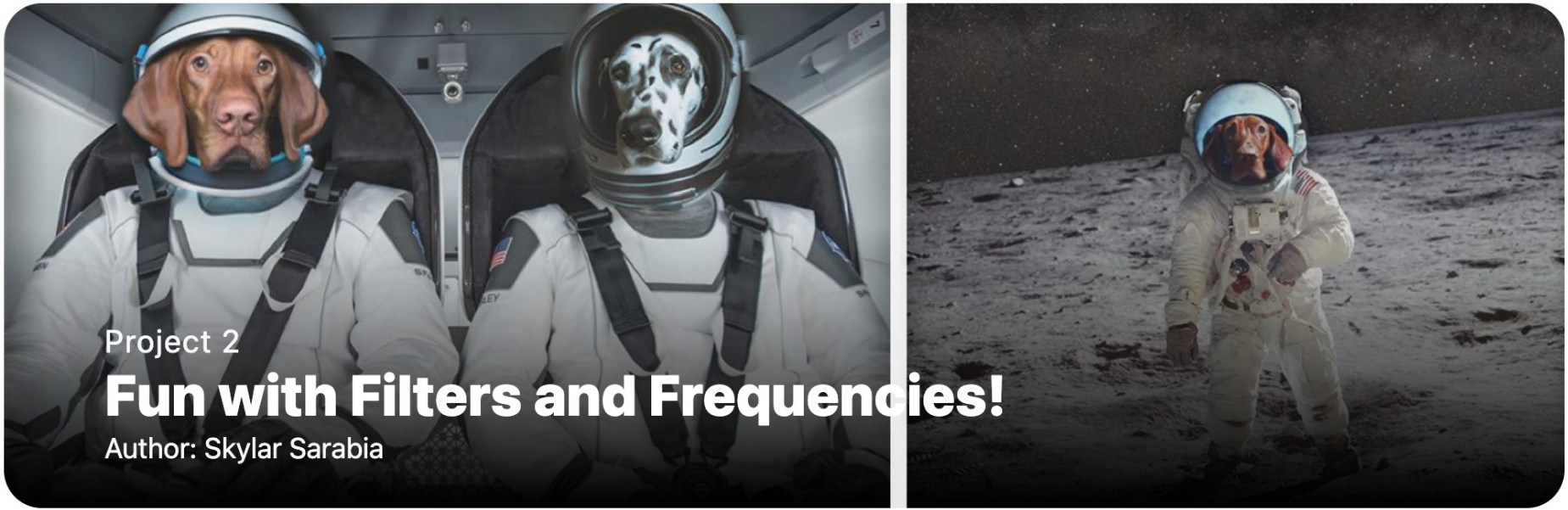
Project 4

- Part A was due last Thursday → checkpoint graded on completion
  - Update: slip days won't be applied on part A
- Part B due next *Monday*
- Final submission includes both A & B
- if you have a bug in part A do fix it for Part B

Project 5

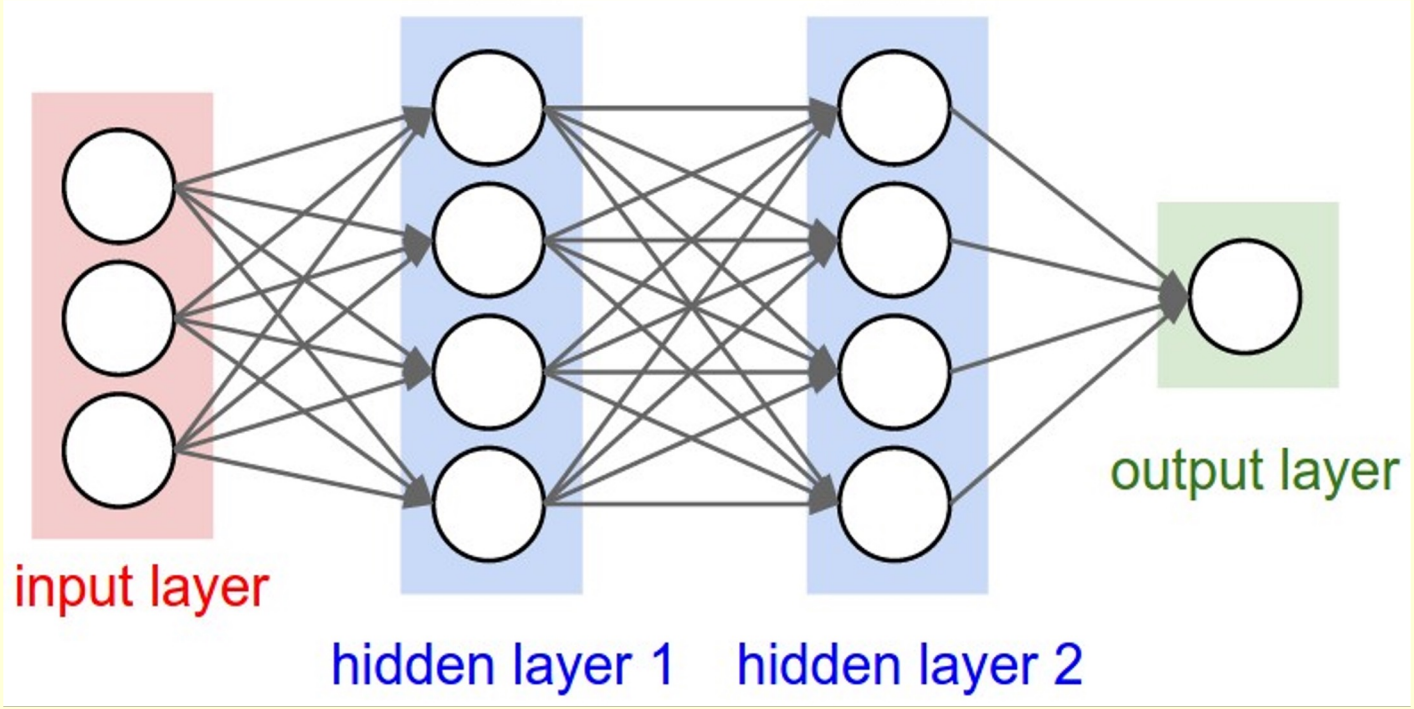
- Will be released Wednesday after Project 4 B is due.
- Go to this Thursday's OH for PyTorch review!!!
- Very important that you go to this if you have never done anything with PyTorch.

Project 2 Class Choice Award Winner!



Runner ups: Anik Gupta Erich Liang Andrew Zhang

Vanilla (fully-connected) Neural Networks

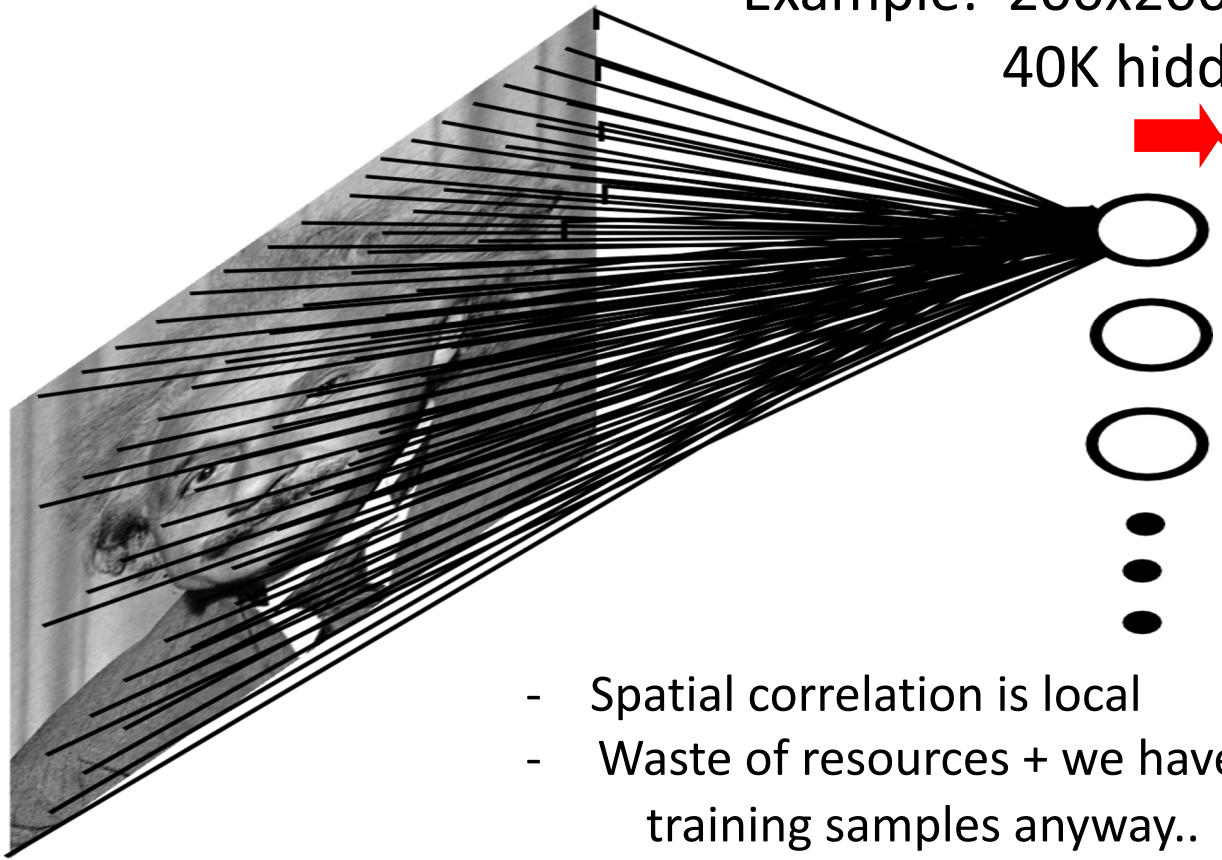


Fully Connected Layer

Example: 200x200 image

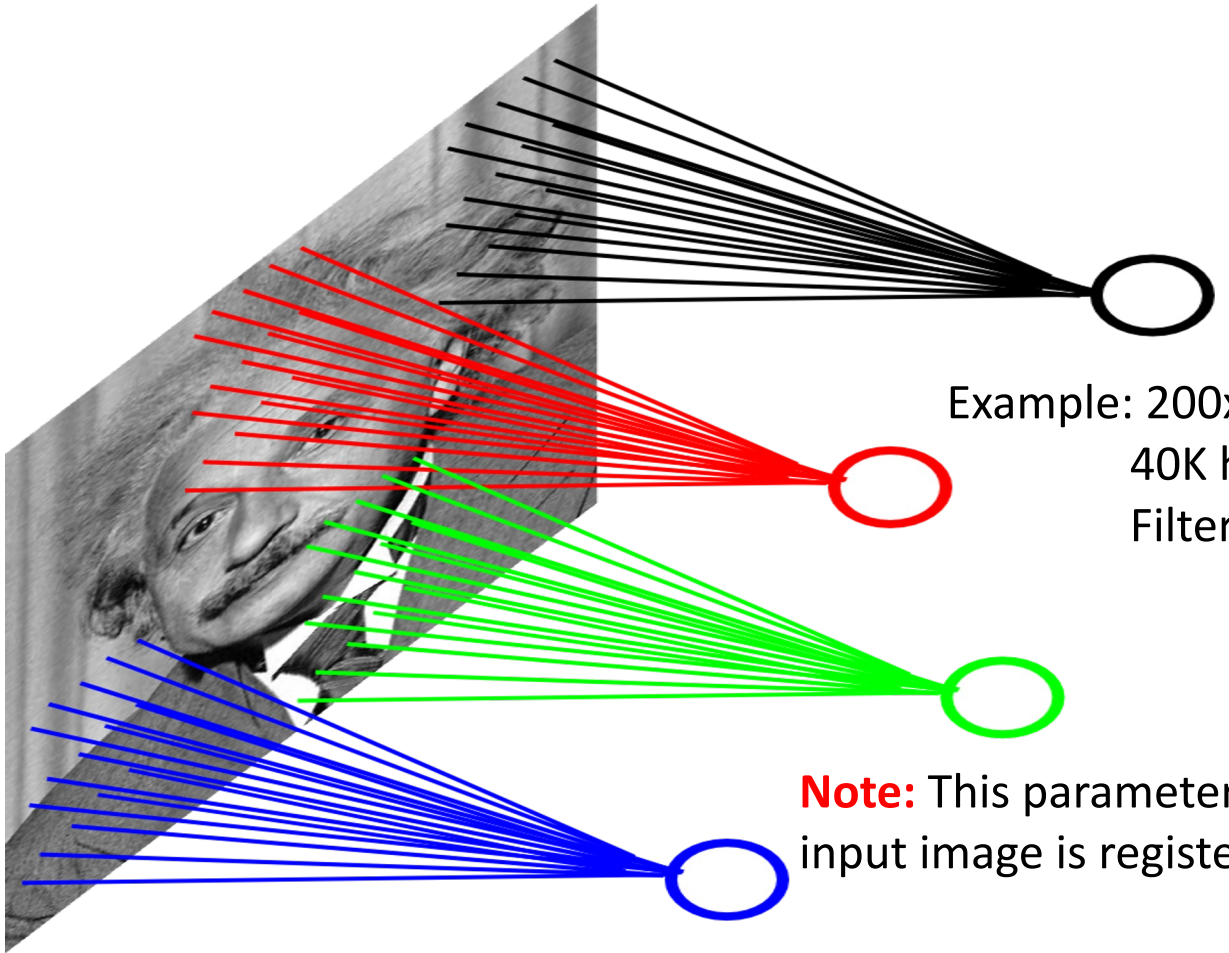
40K hidden units

~2B parameters!!!



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

Locally Connected Layer



Example: 200x200 image

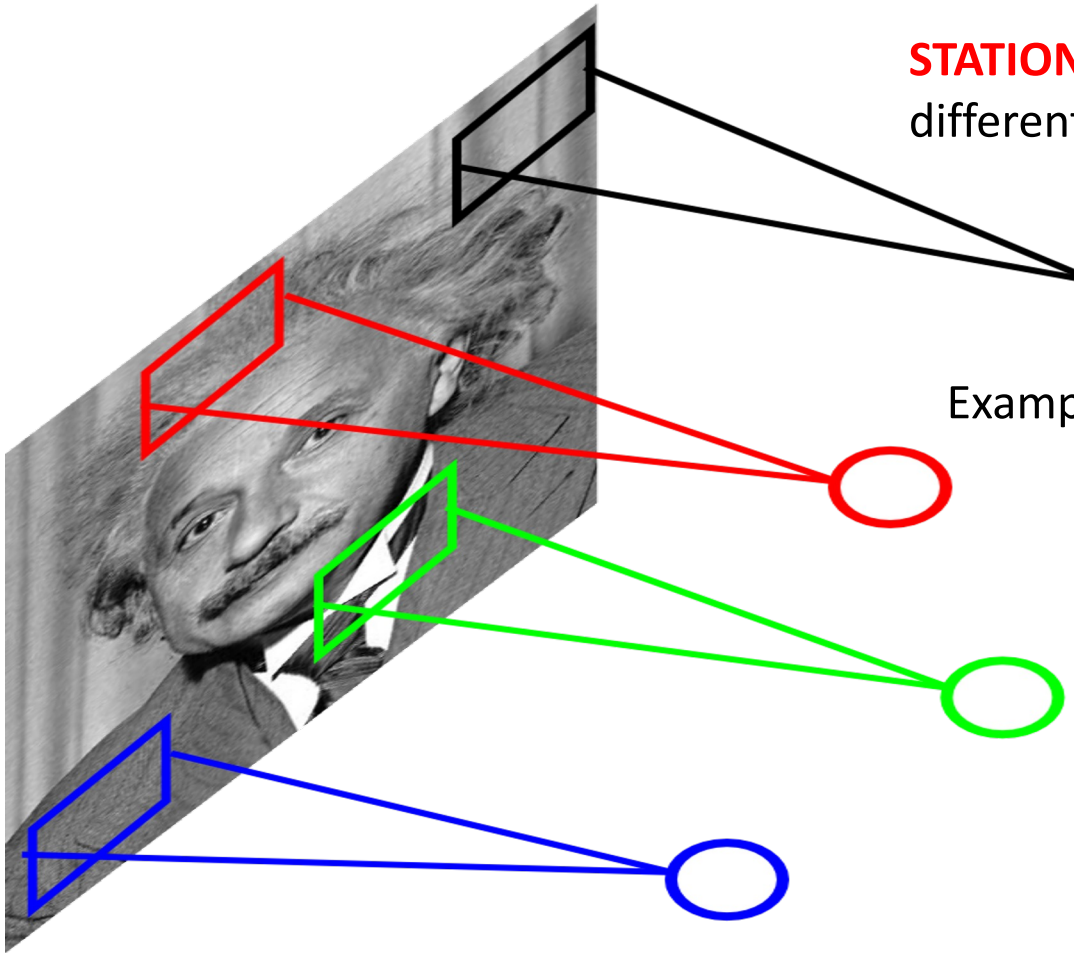
40K hidden units

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

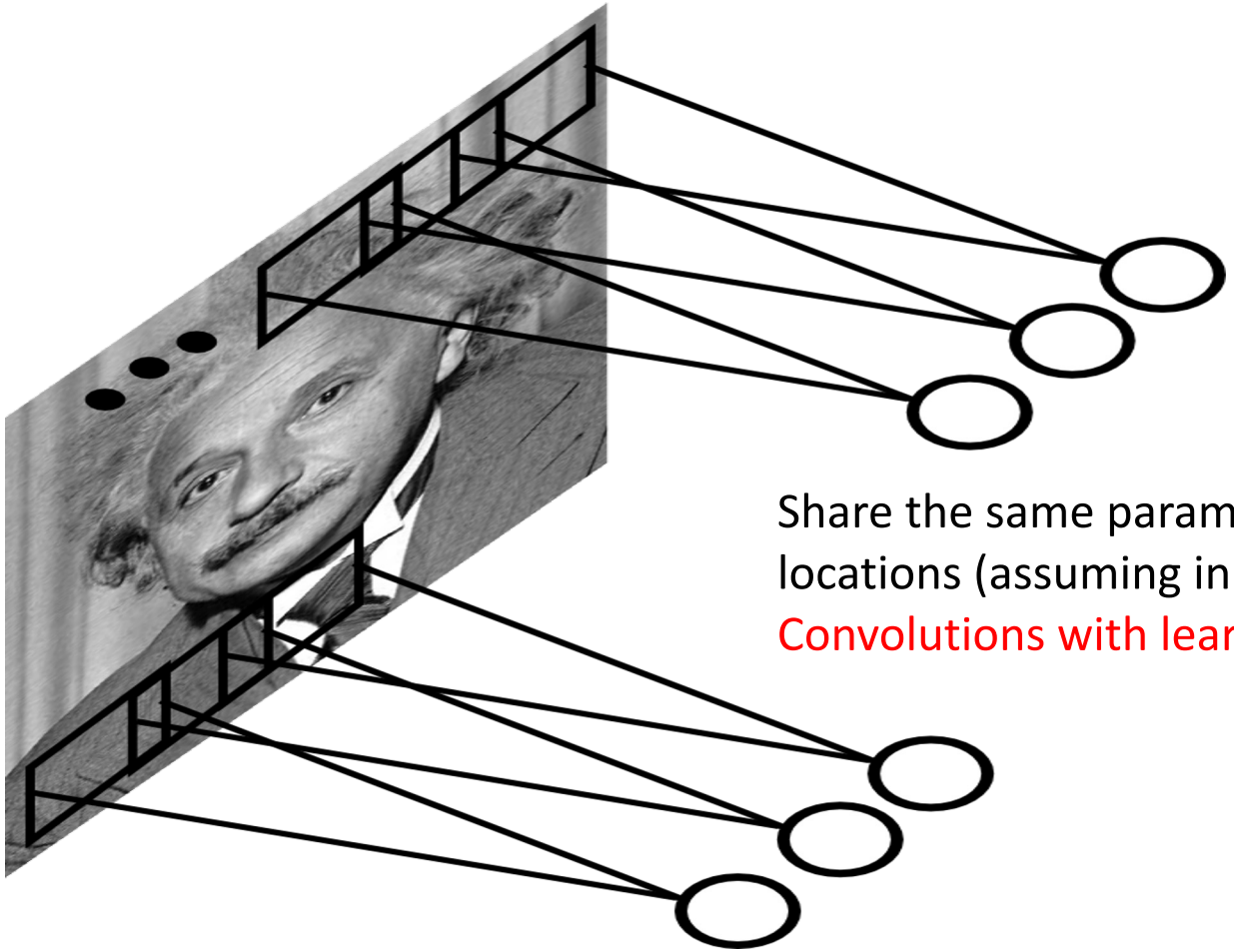
Example: 200x200 image

40K hidden units

Filter size: 10x10

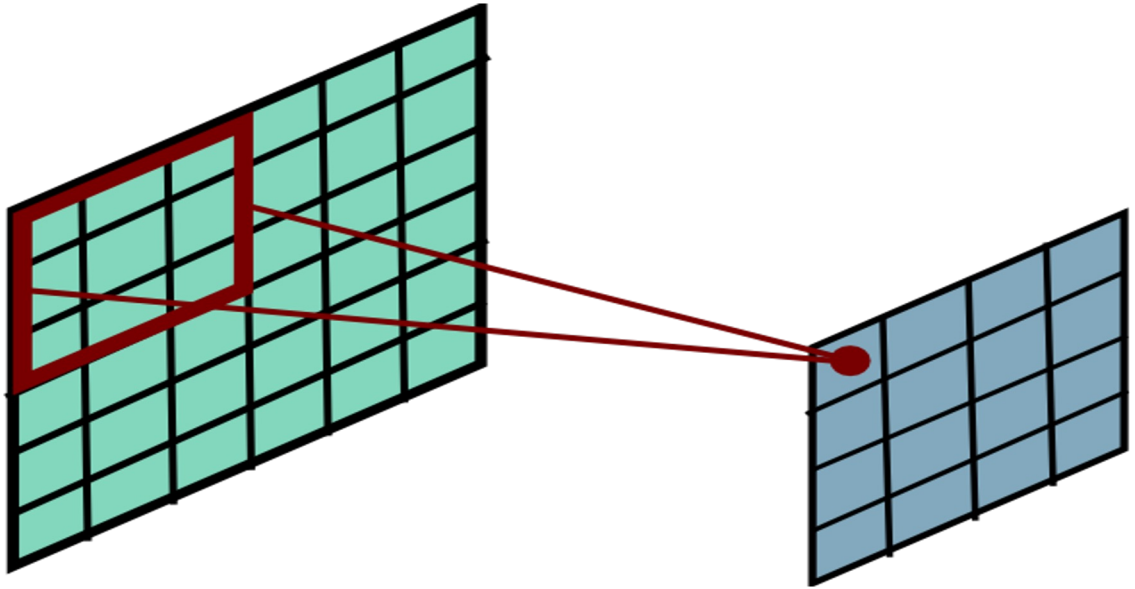
4M parameters

Convolutional Layer

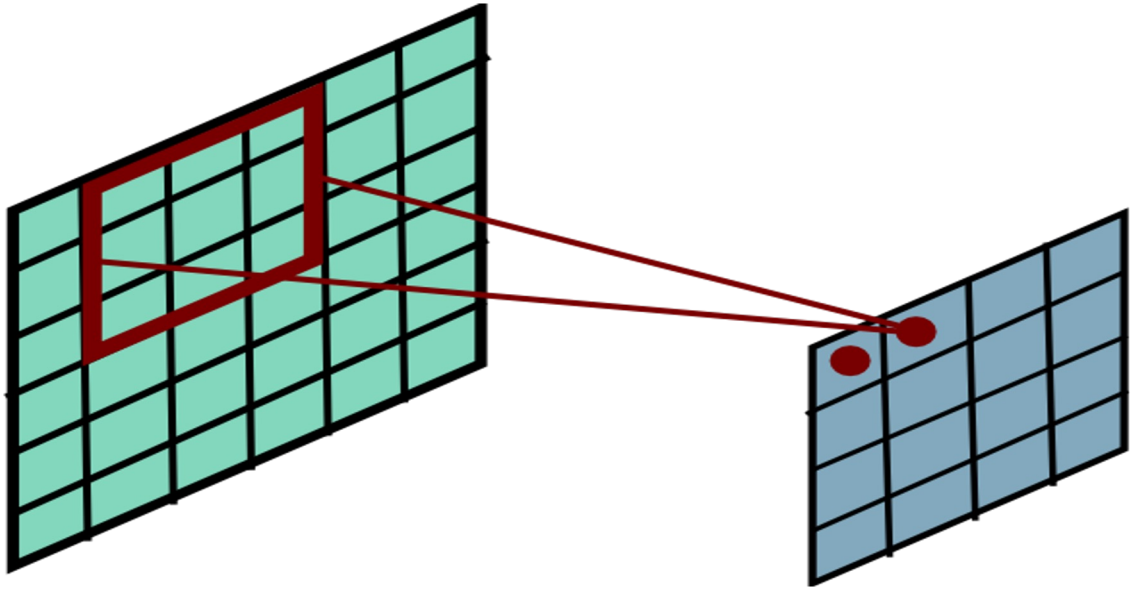


Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

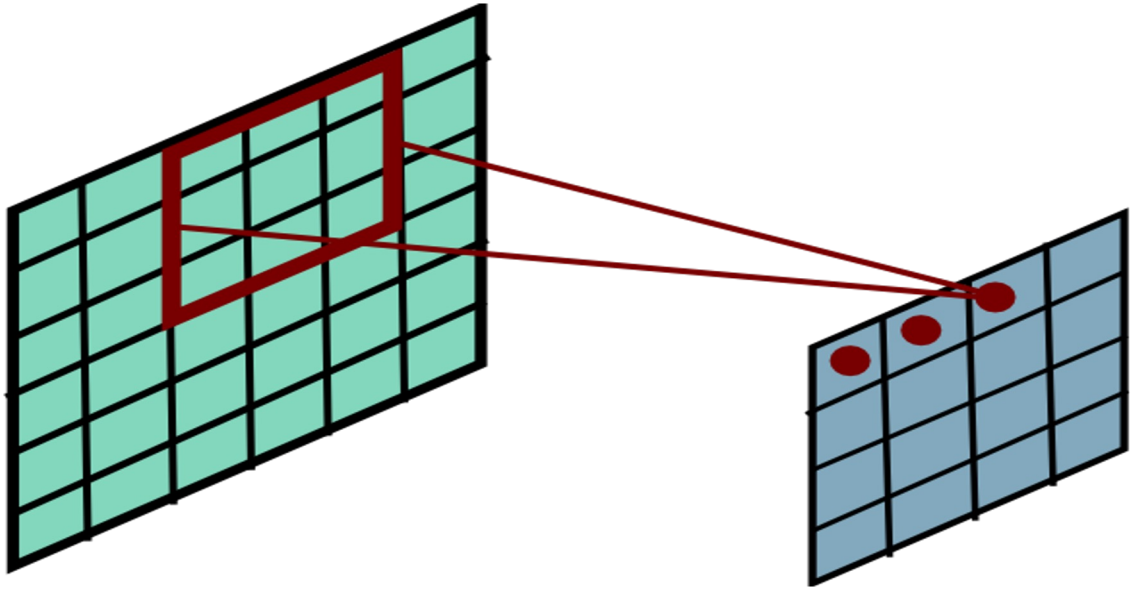
Convolutional Layer



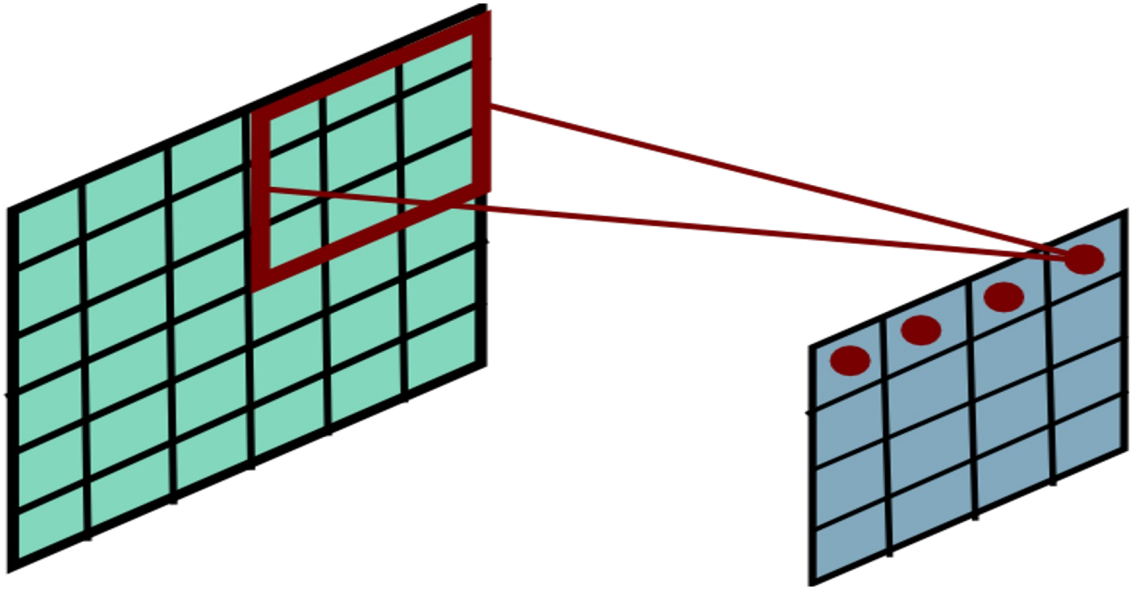
Convolutional Layer



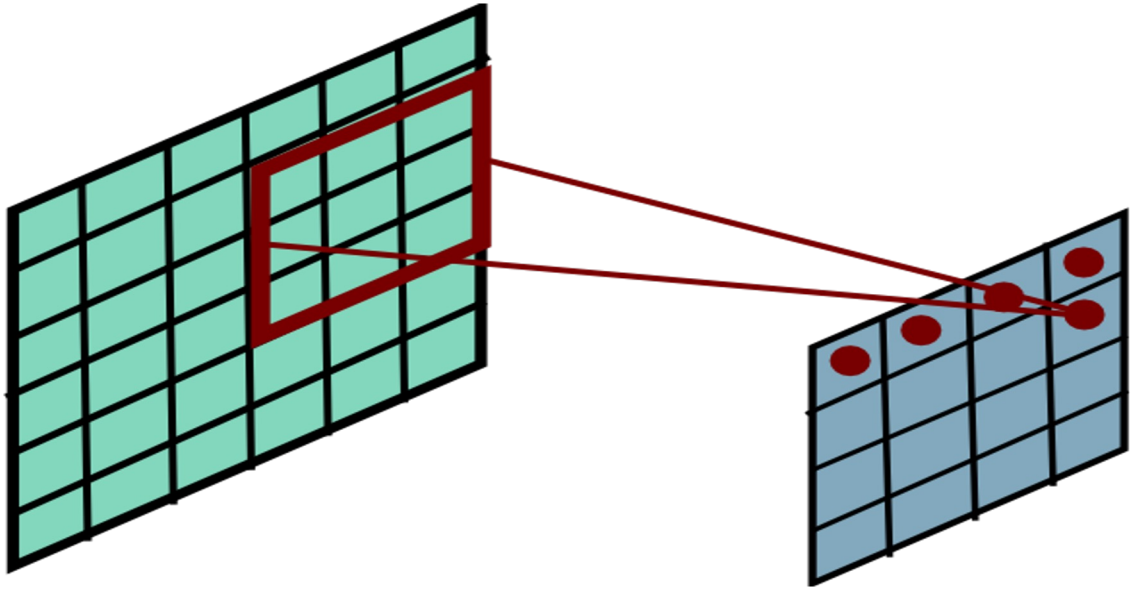
Convolutional Layer



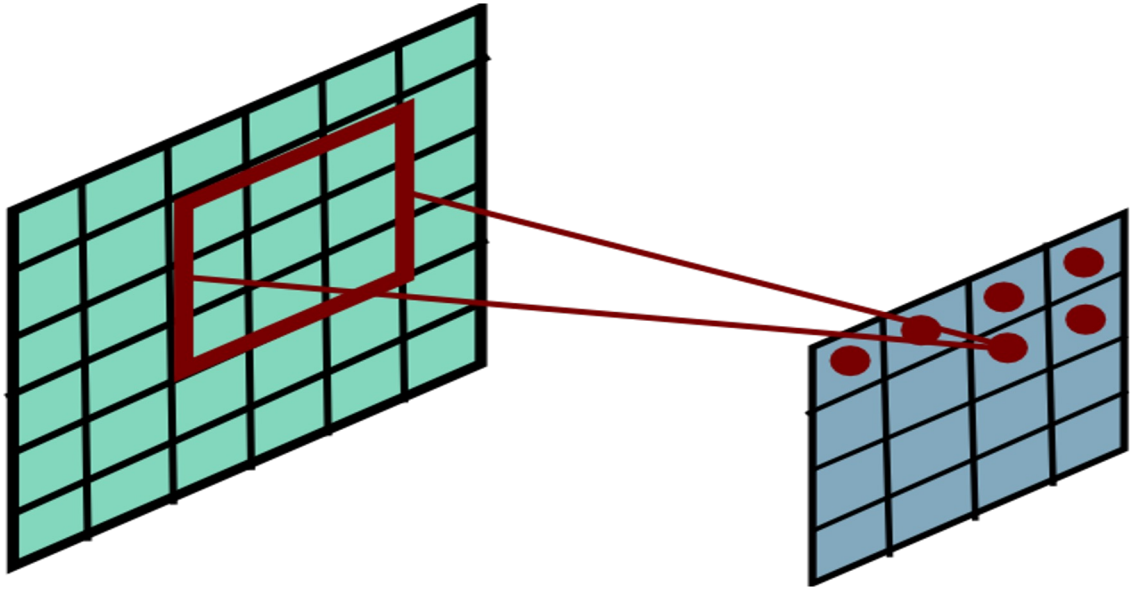
Convolutional Layer



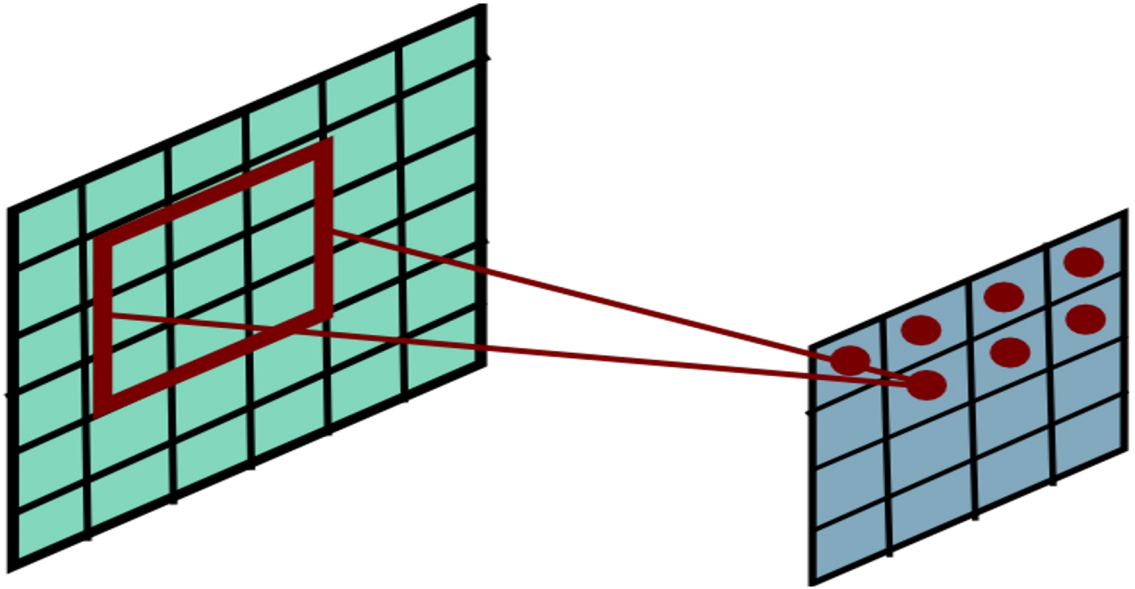
Convolutional Layer



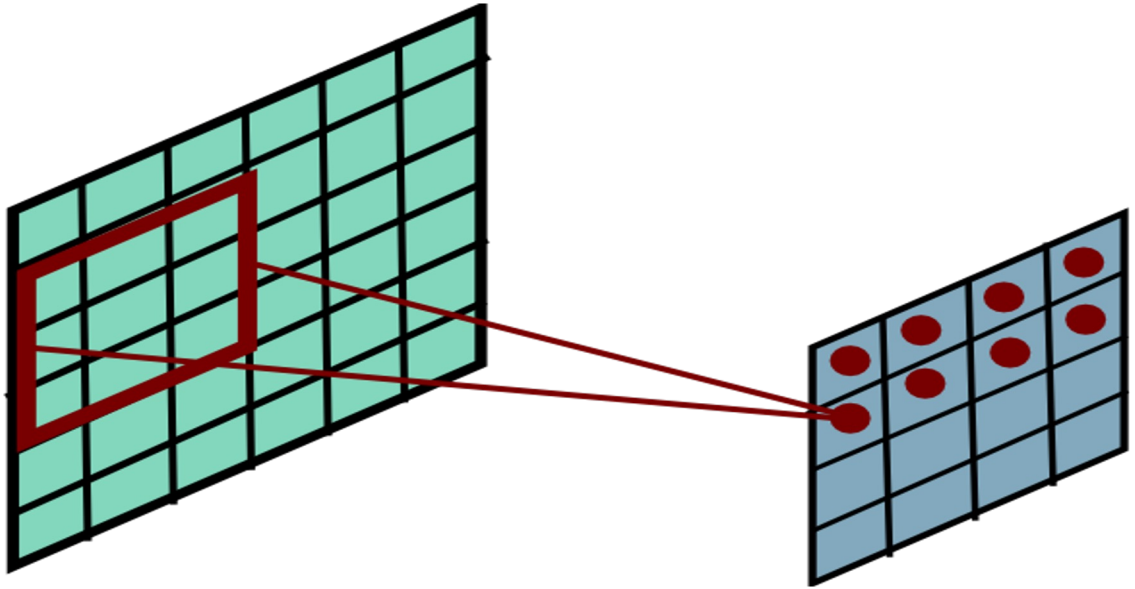
Convolutional Layer



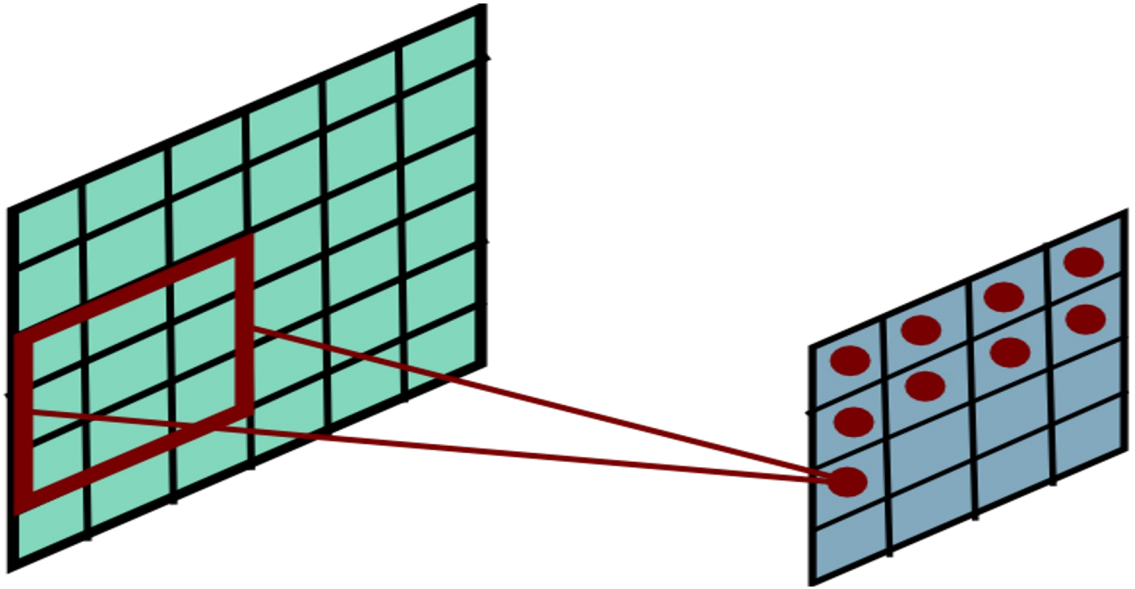
Convolutional Layer



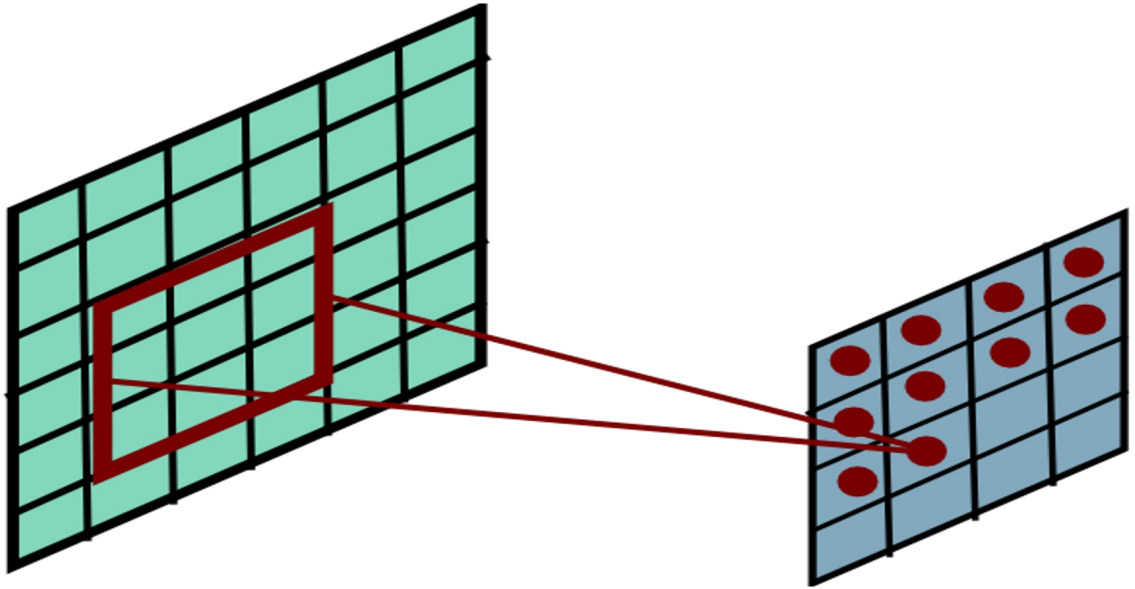
Convolutional Layer



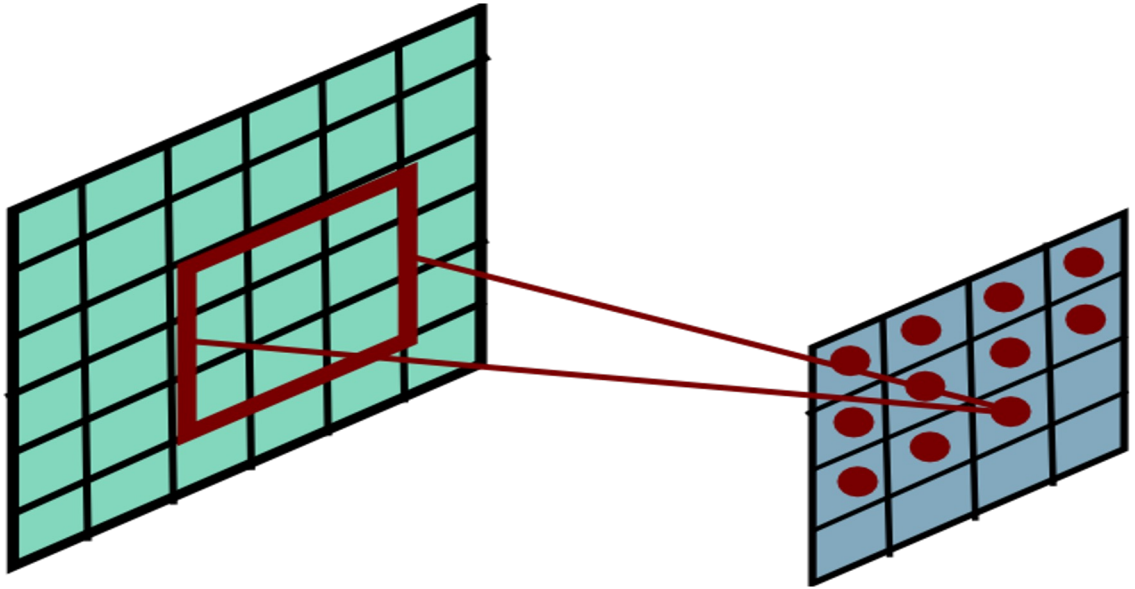
Convolutional Layer



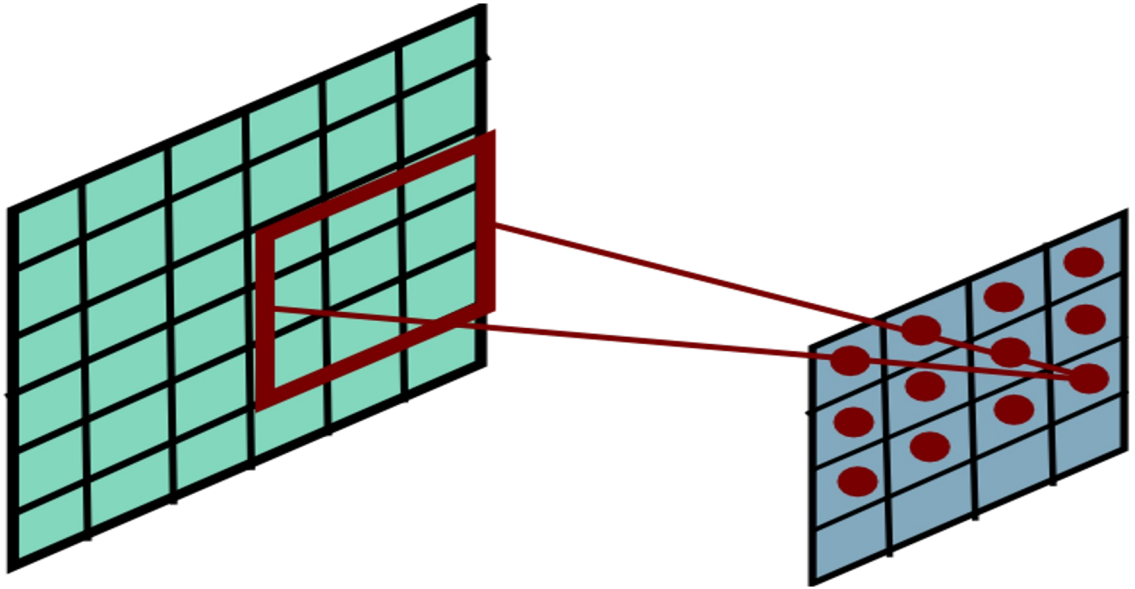
Convolutional Layer



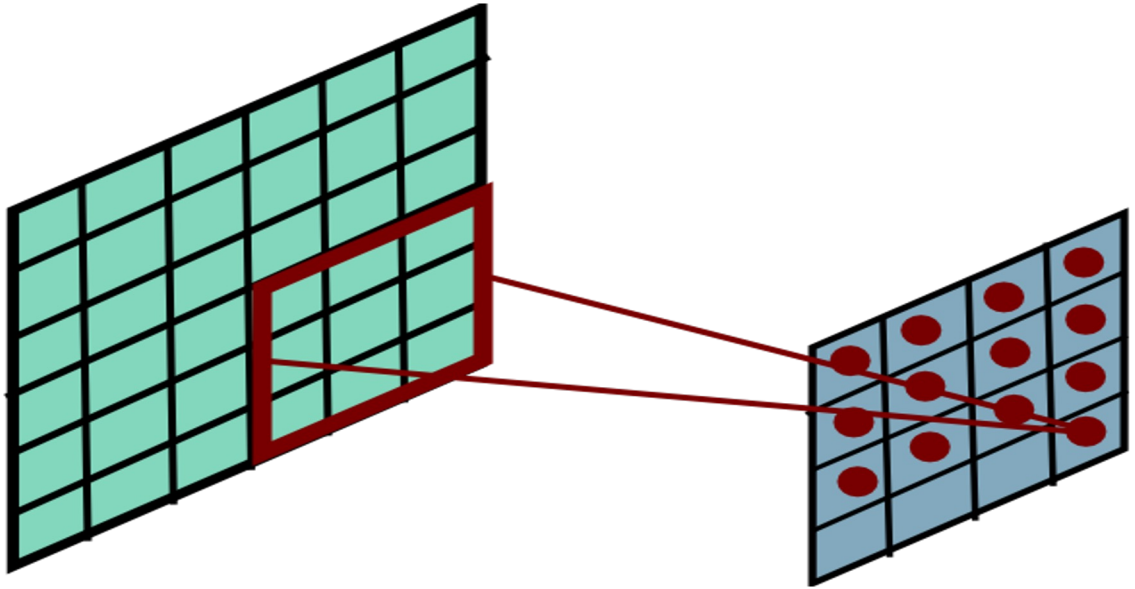
Convolutional Layer



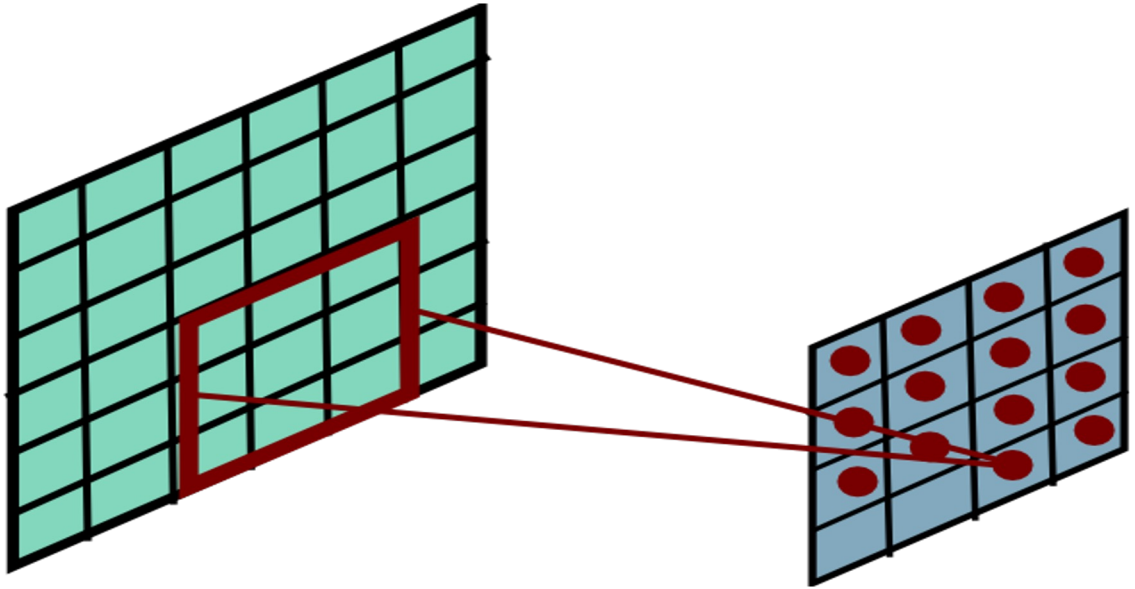
Convolutional Layer



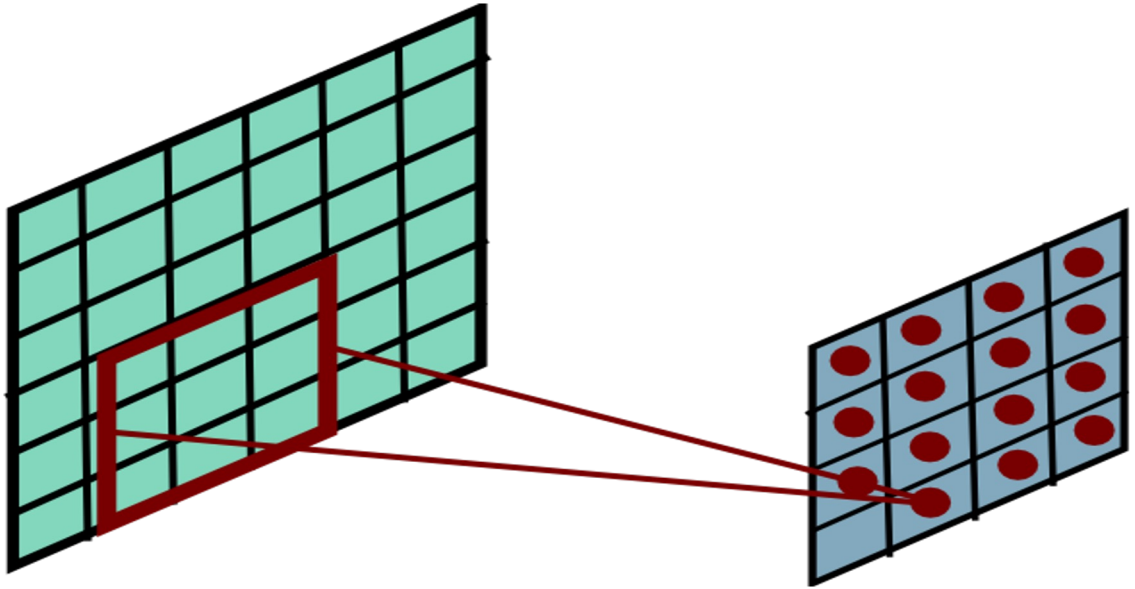
Convolutional Layer



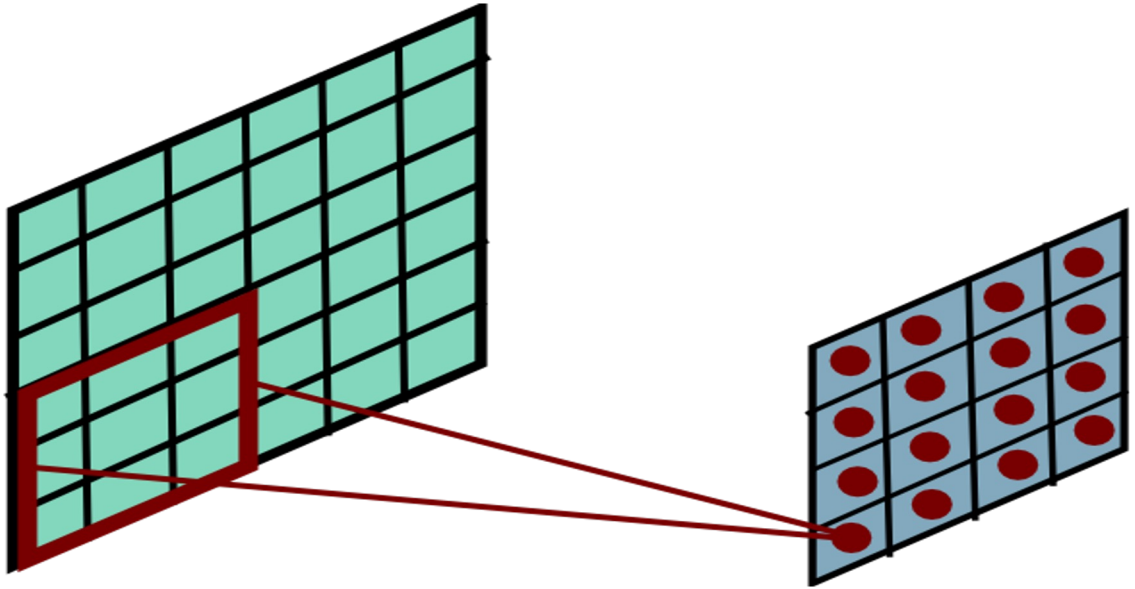
Convolutional Layer



Convolutional Layer

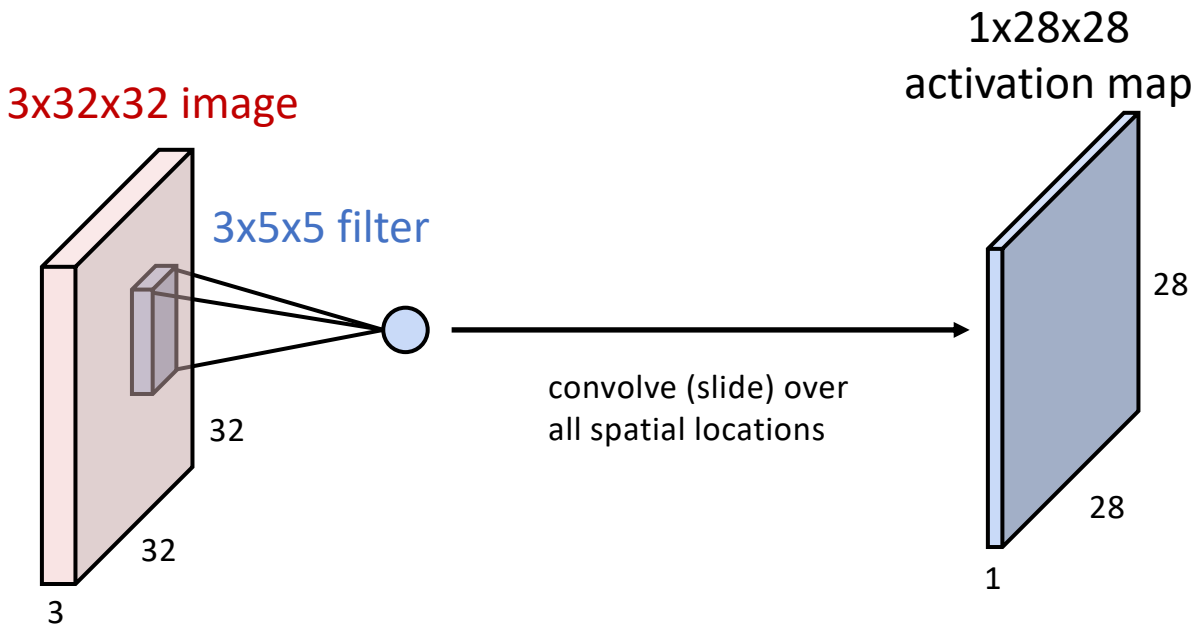


Convolutional Layer



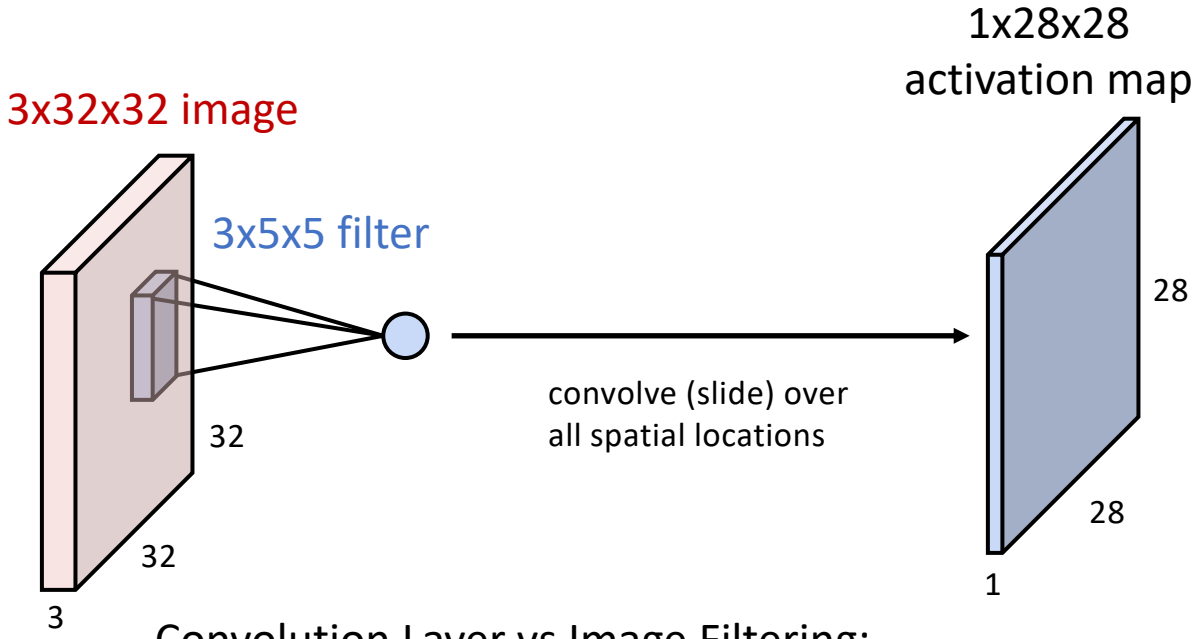
Convolution Layer

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:



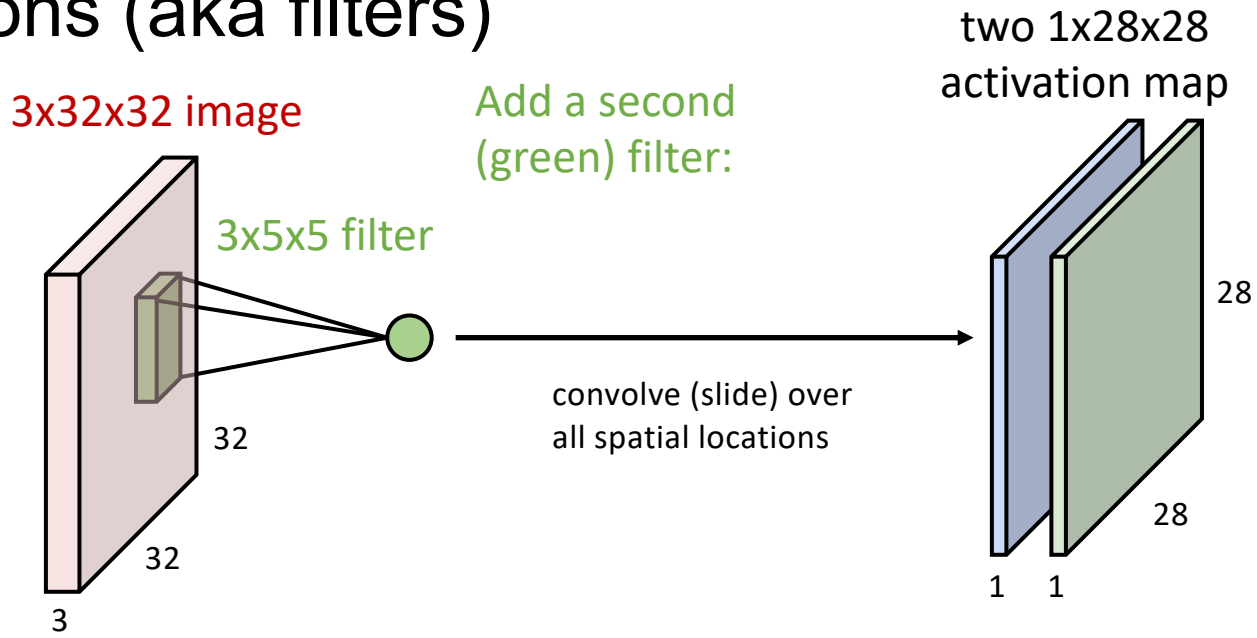
Convolution Layer vs Image Filtering:

>1 input and output channels

Forget about convolution vs cross-correlation (Why?)

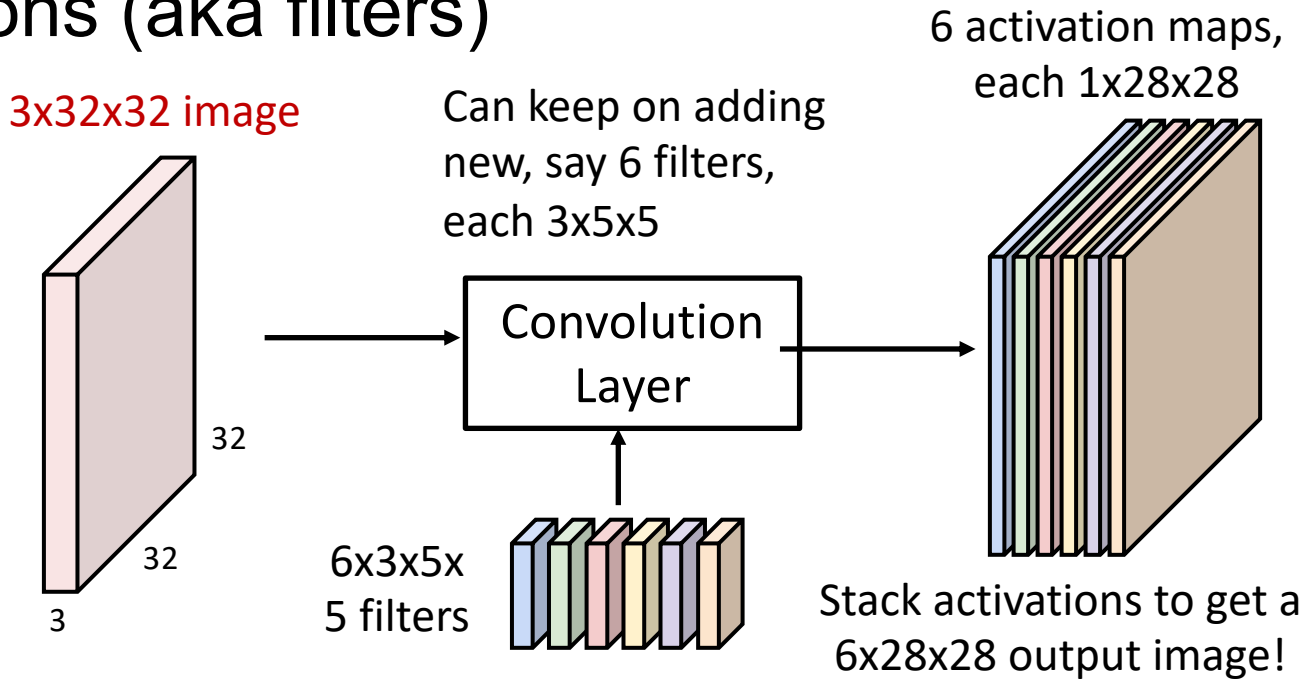
Convolution Layer:

2 neurons (aka filters)

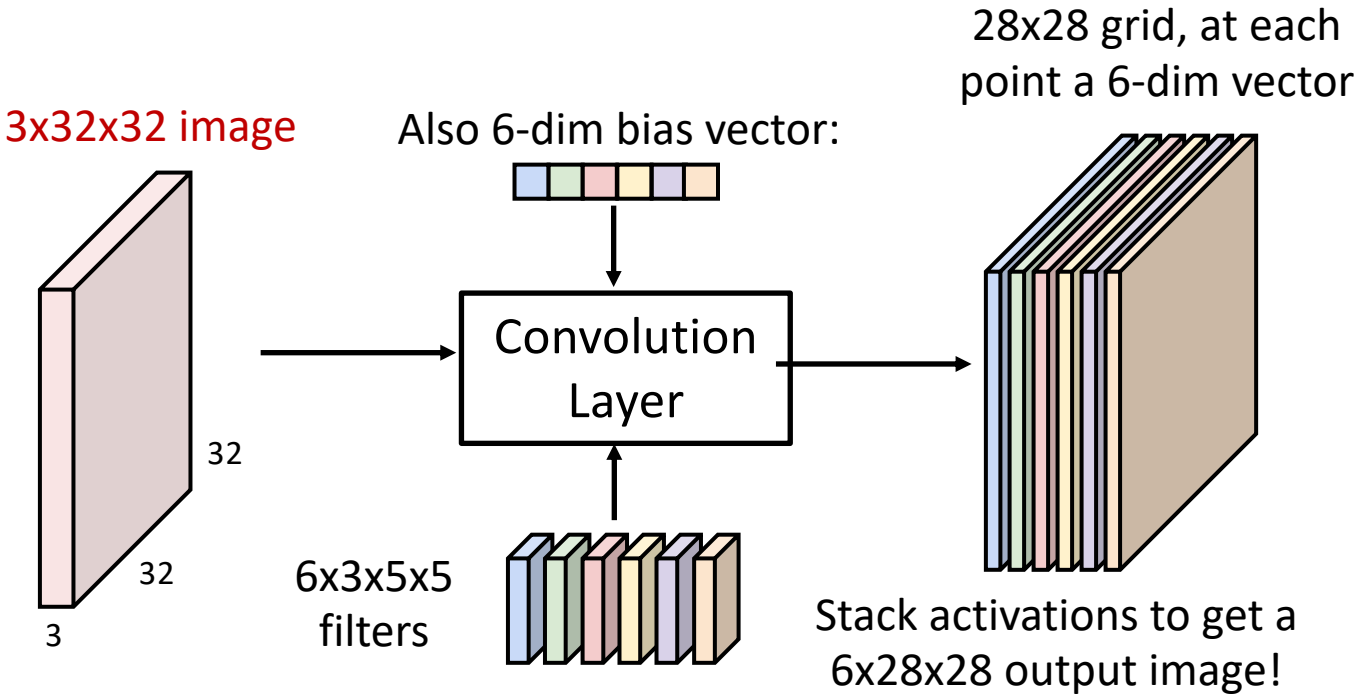


Convolution Layer

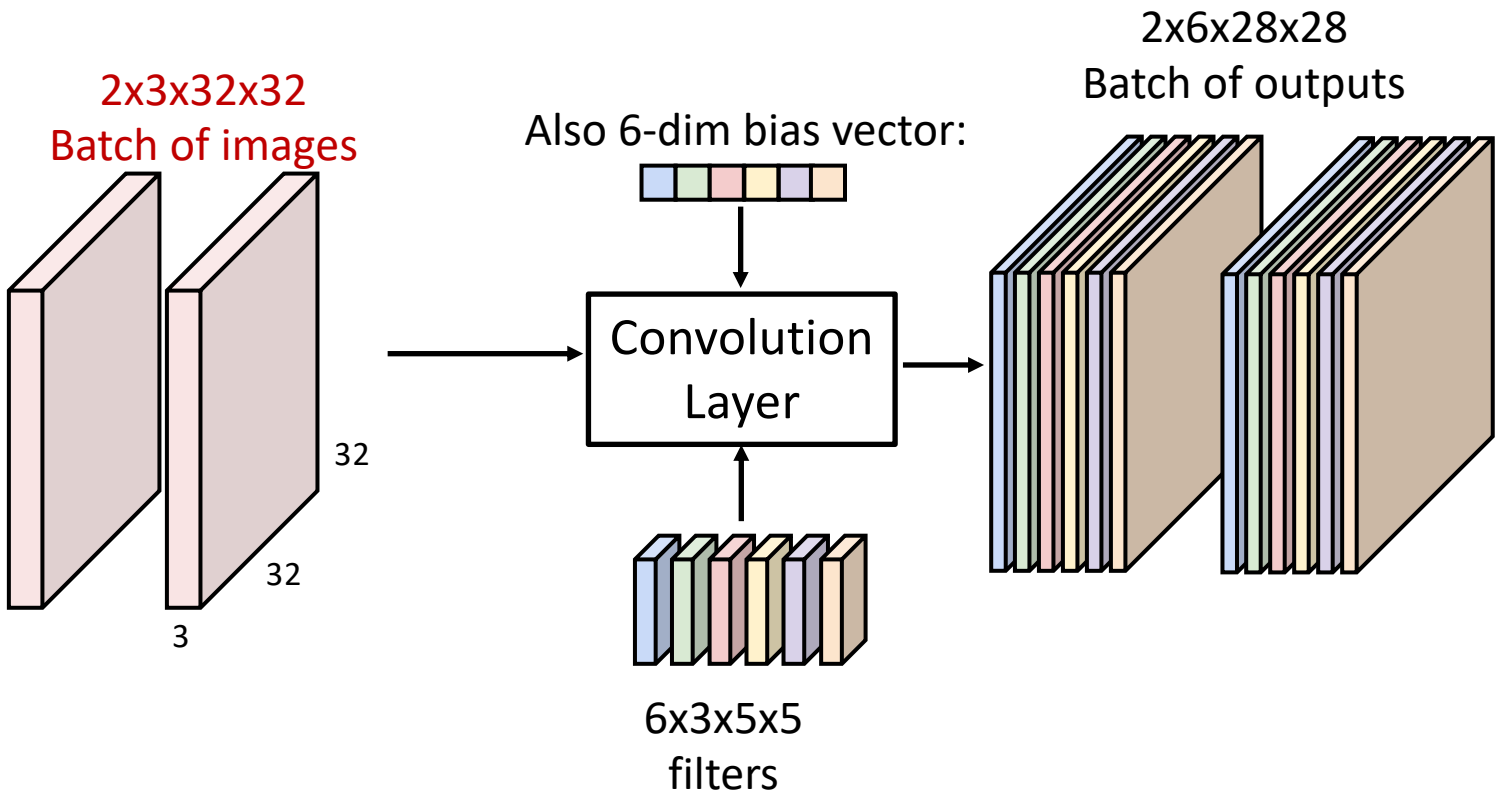
6 neurons (aka filters)



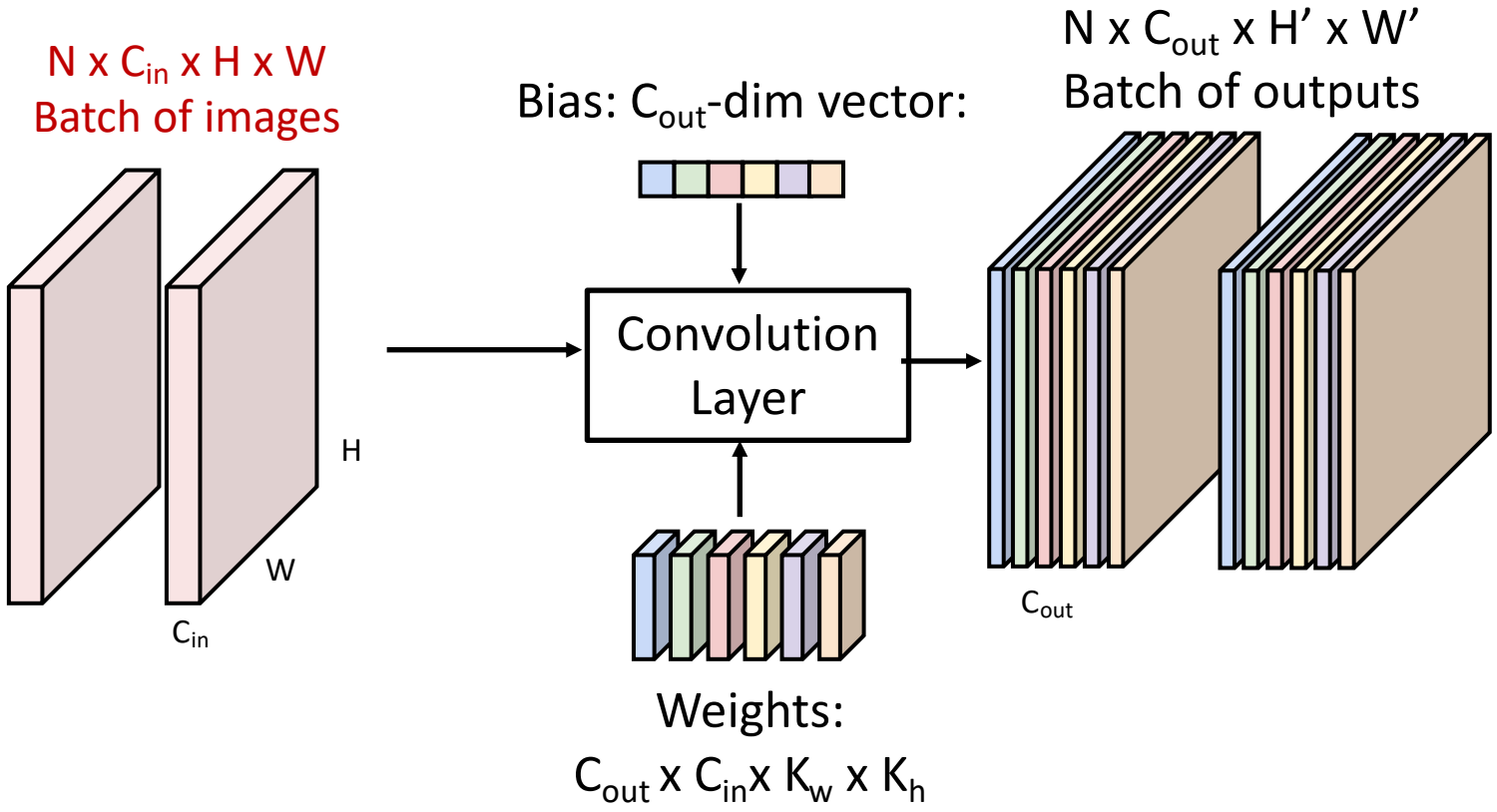
Convolution Layer



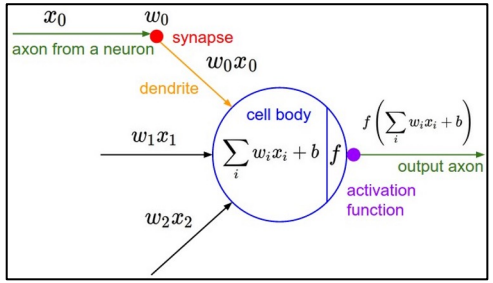
Convolution Layer: With many images

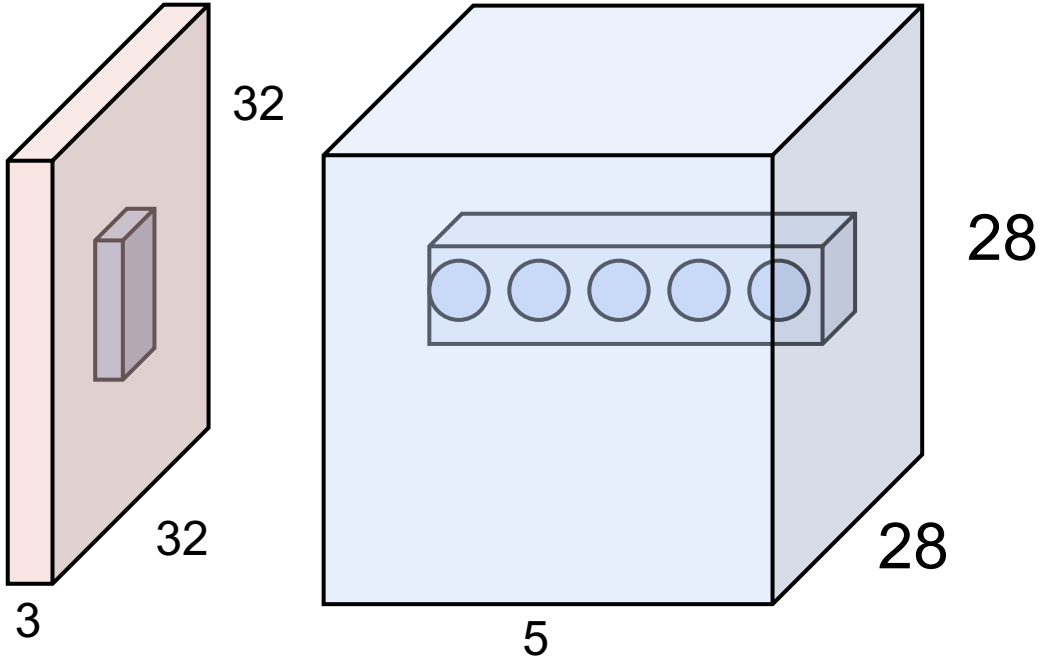


Convolution Layer



The brain/neuron view of CONV Layer



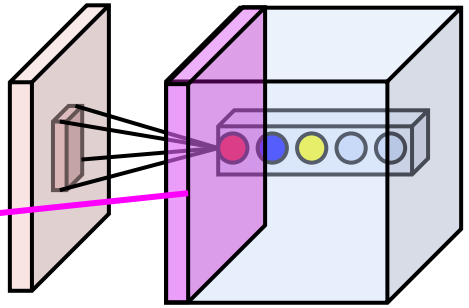


E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

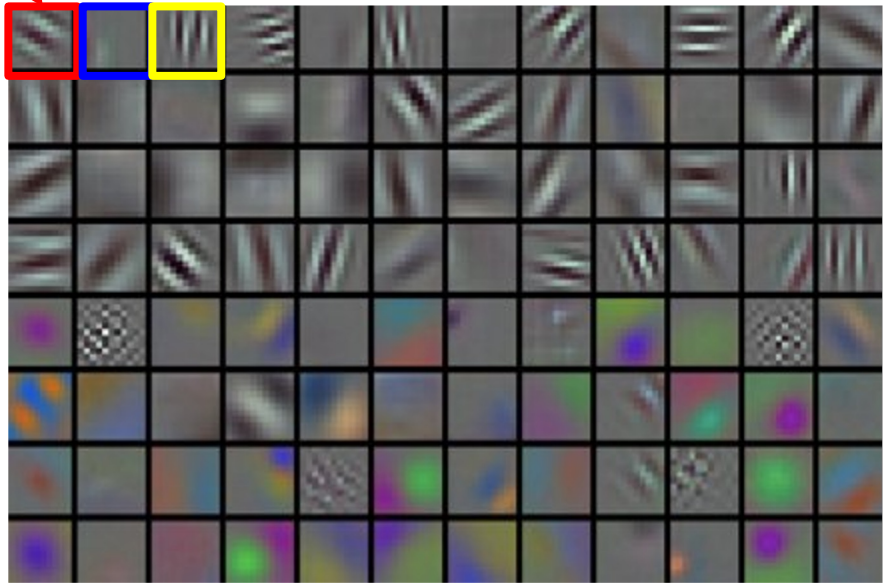
There will be 5 different neurons all looking at the same region in the input volume

Activations:

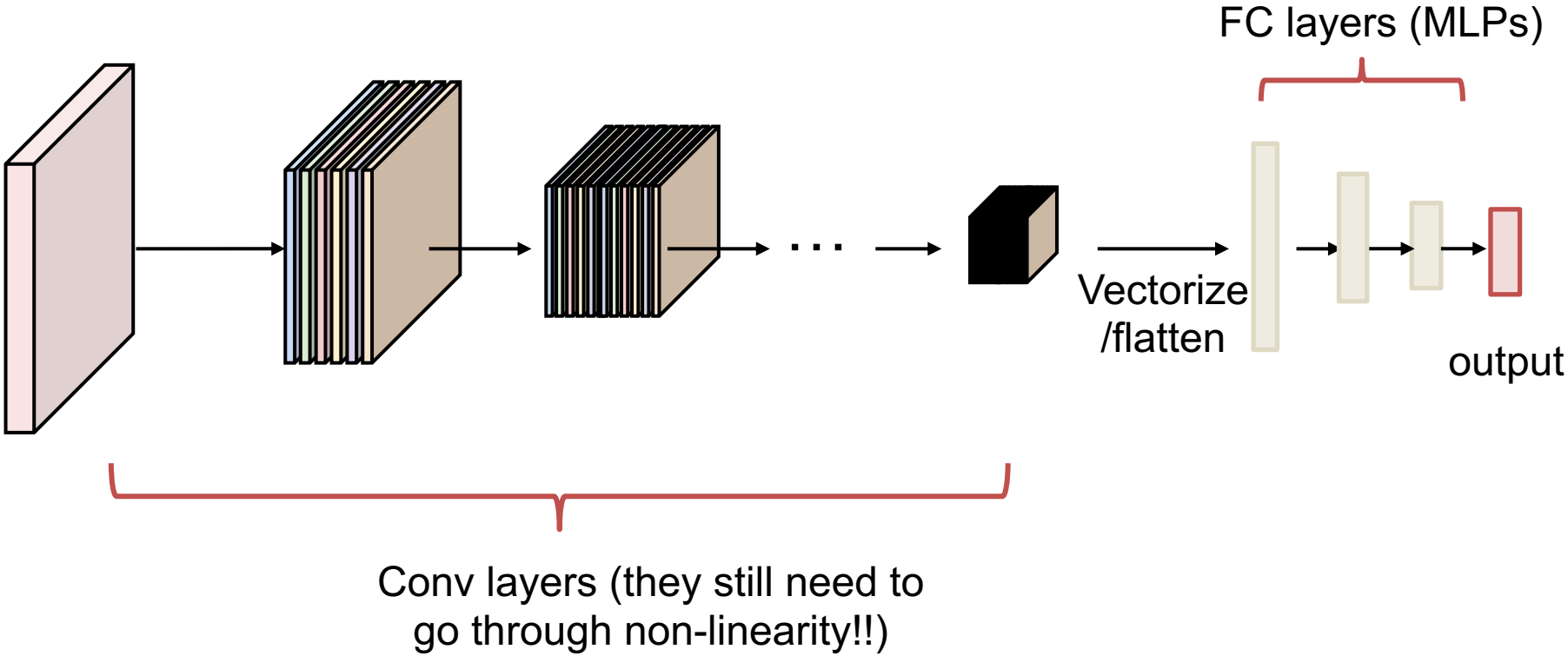
convolving the first filter in the input gives the first slice of depth in output volume



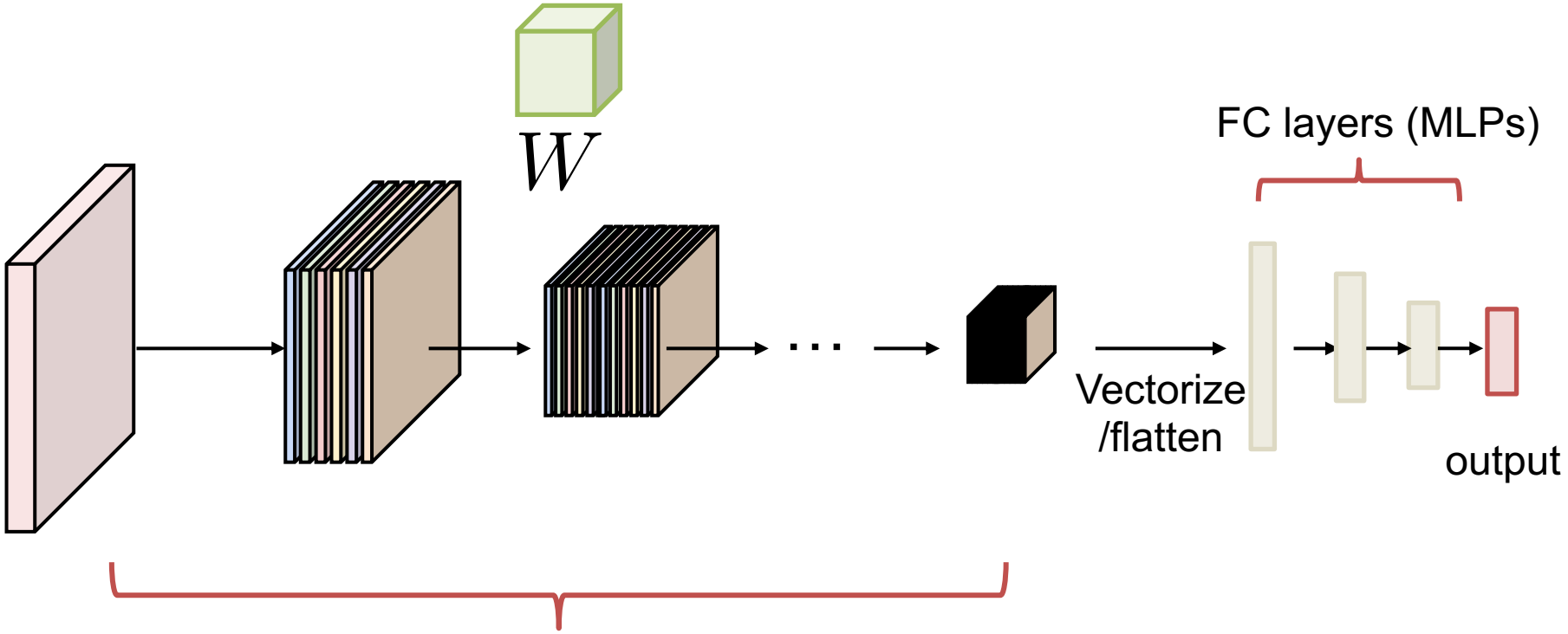
Activations:



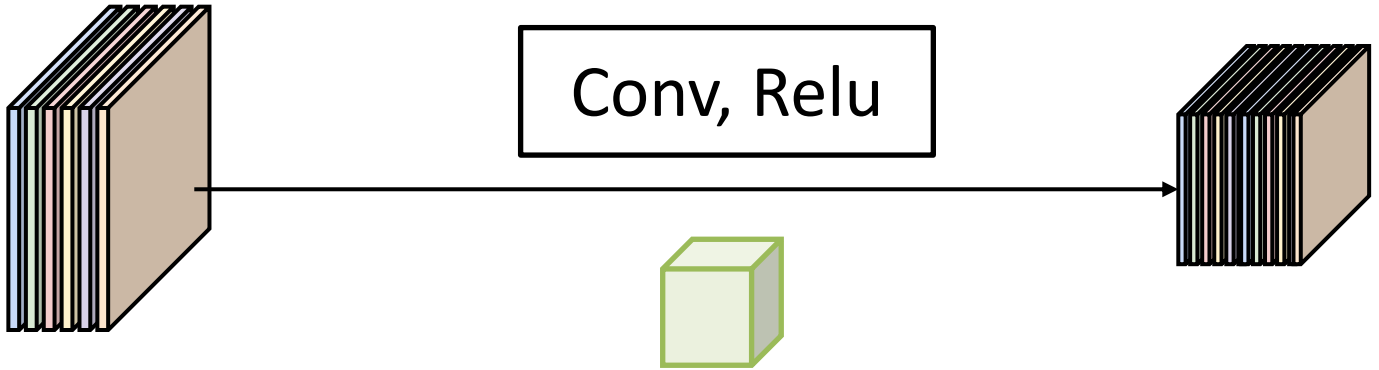
CNNs in practice



What needs to be learned?



What needs to be learned?



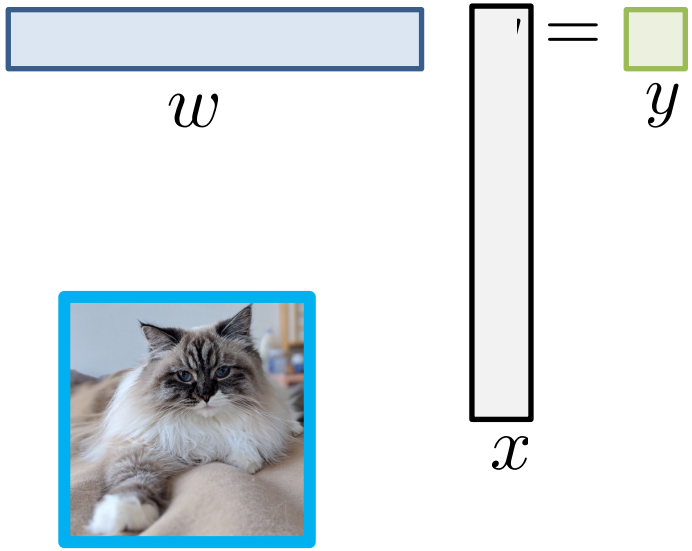
Input:
B, C_in, H, W

W
C_out, C_in, K, K

Output
B, C_out, H, W

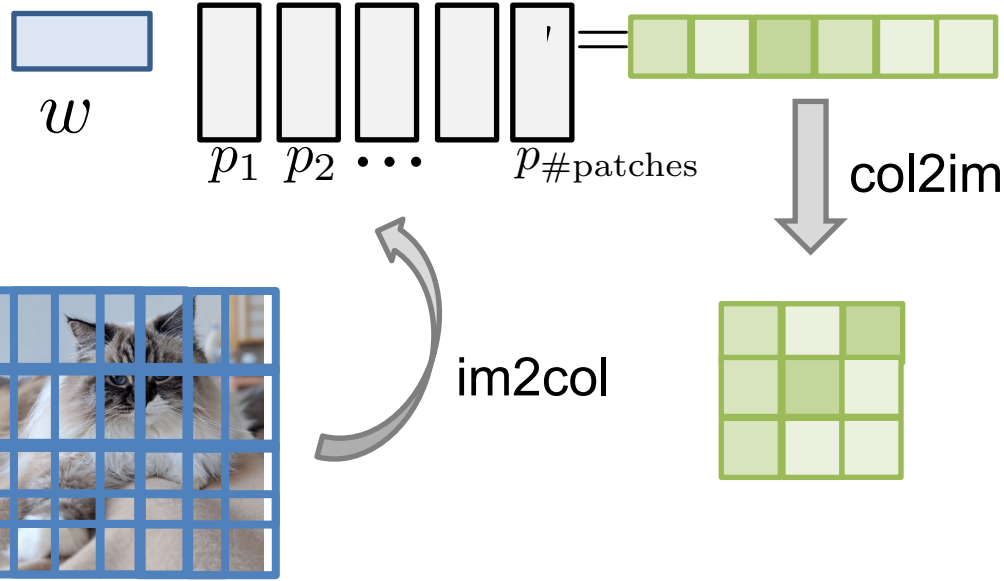
We're still doing matrix multiplications, just localized & shared

Recall one neuron in FC layer:



w takes the entire image!

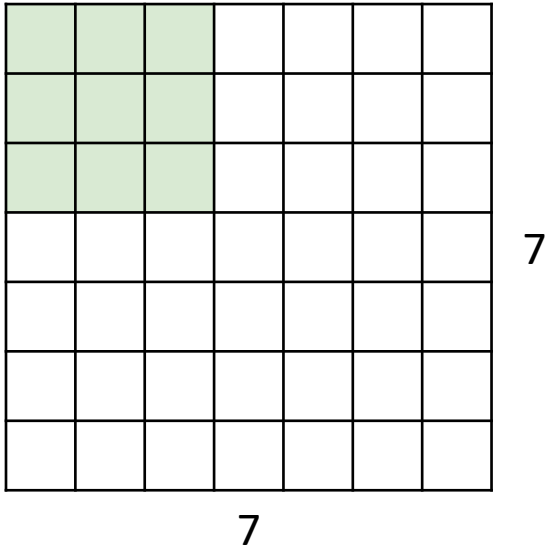
With Conv layer:



Now w takes (overlapping) patches

CONVNET NITTY GRITTIES

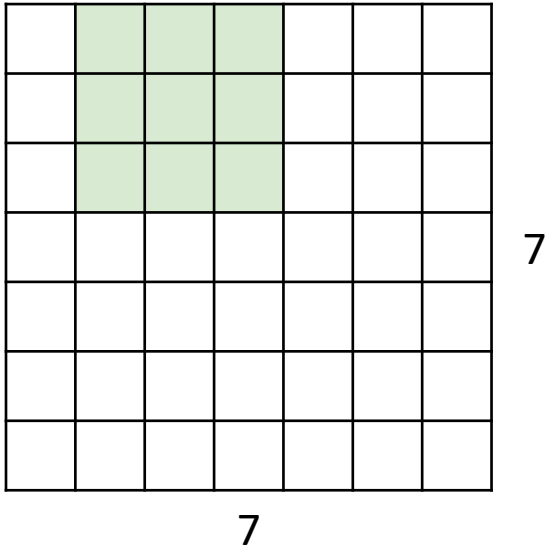
Convolution Spatial Dimensions



Input: 7x7
Filter: 3x3

Q: How big is output?

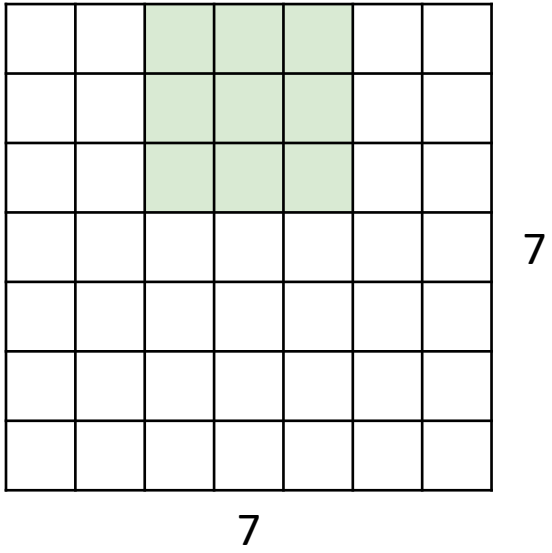
Convolution Spatial Dimensions



Input: 7x7
Filter: 3x3

Q: How big is output?

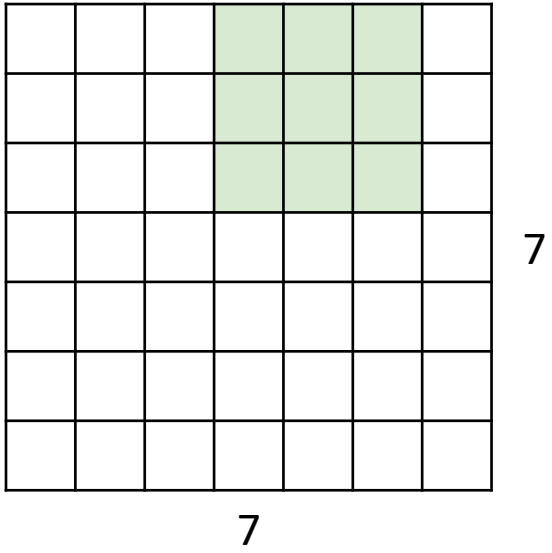
Convolution Spatial Dimensions



Input: 7x7
Filter: 3x3

Q: How big is output?

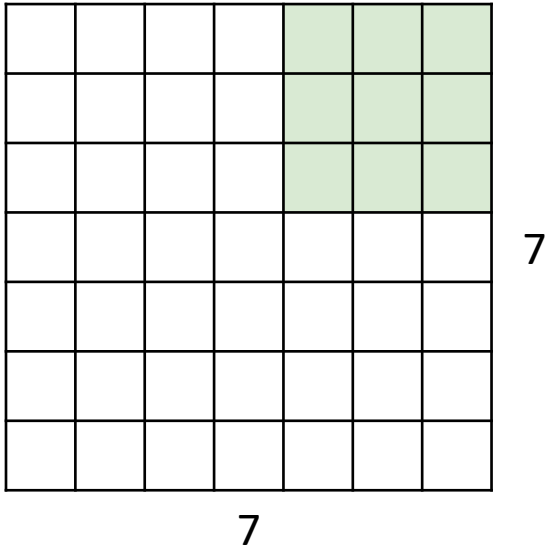
Convolution Spatial Dimensions



Input: 7x7
Filter: 3x3

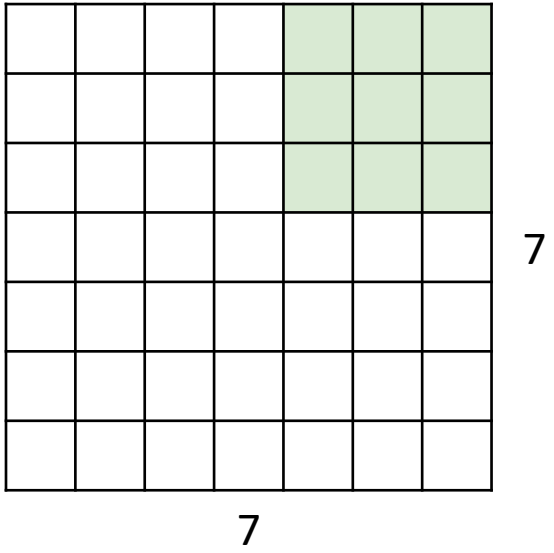
Q: How big is output?

Convolution Spatial Dimensions



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

Convolution Spatial Dimensions

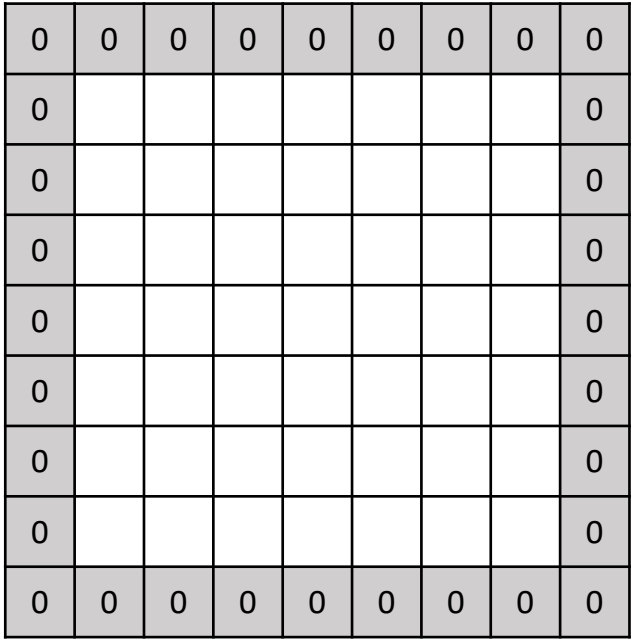


Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Output: W – K + 1

Problem:
Feature maps
“shrink” with
each layer!

Convolution Spatial Dimensions



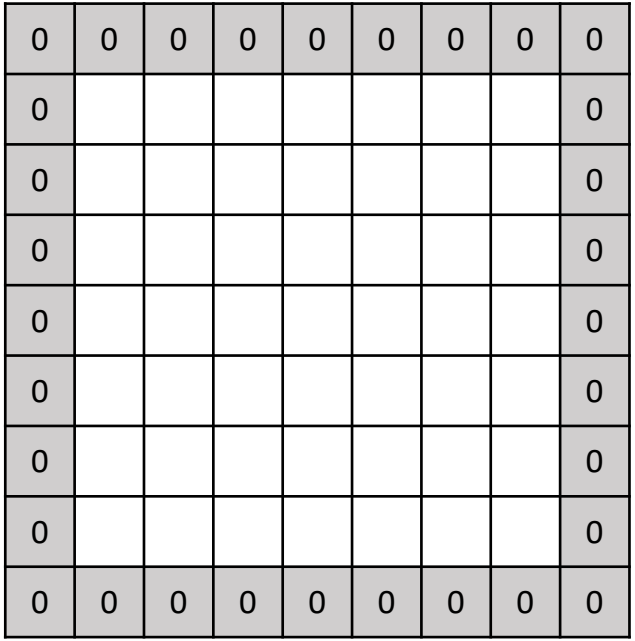
Input: 7x7
Filter: 3x3
Output: 7x7

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



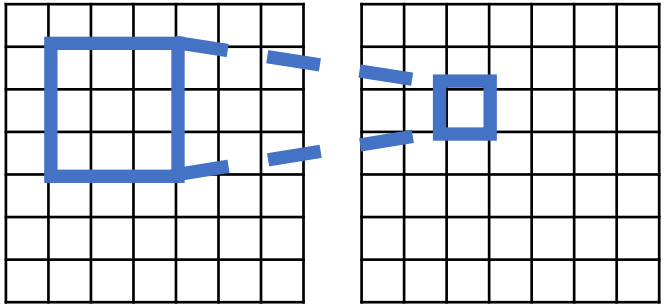
Input: 7x7
Filter: 3x3
Output: 5x5

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

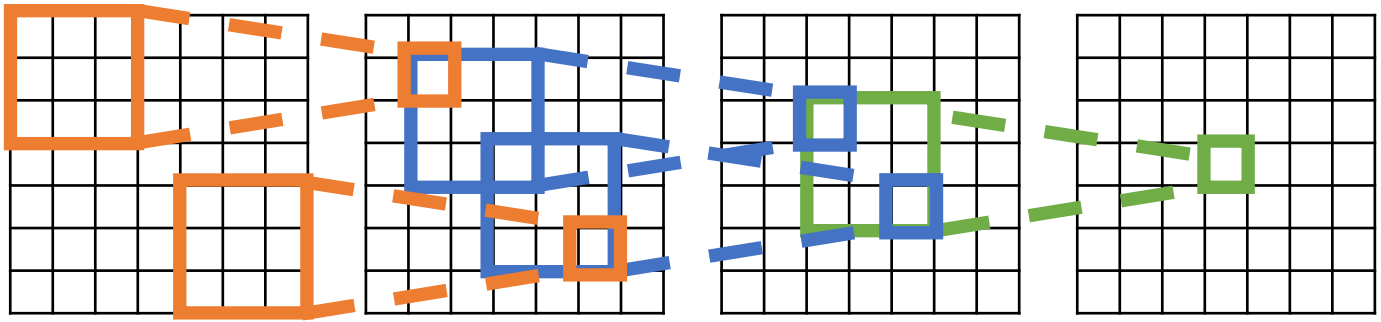
For convolution with kernel size K, each element in the output depends on a K x K **receptive field** in the input



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)



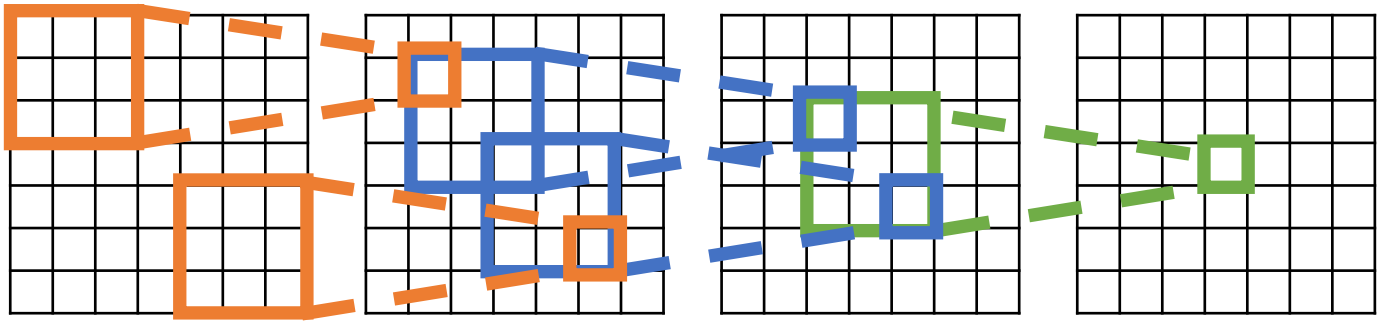
Input

Output

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

Receptive Fields

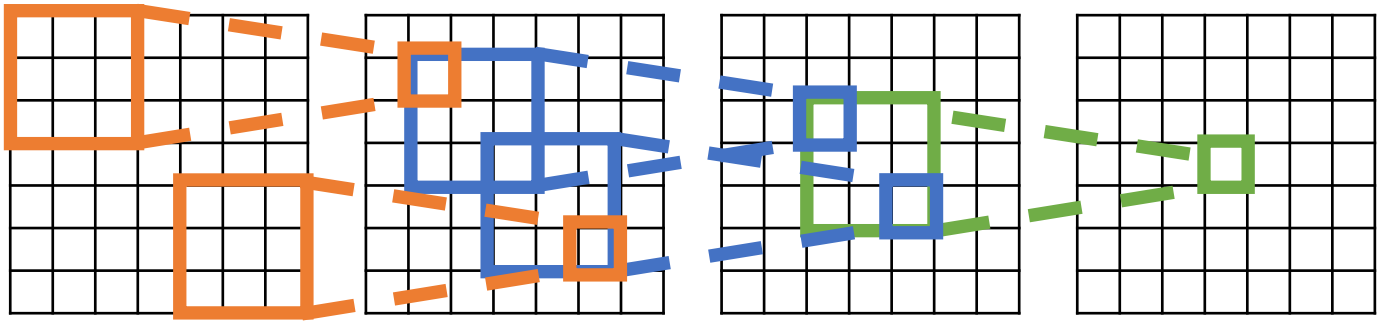
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



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

Receptive Fields

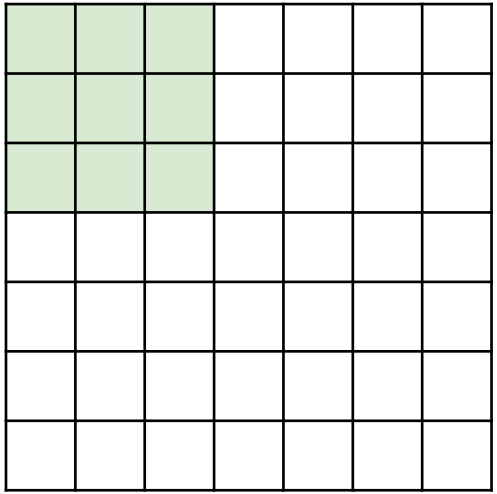
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



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

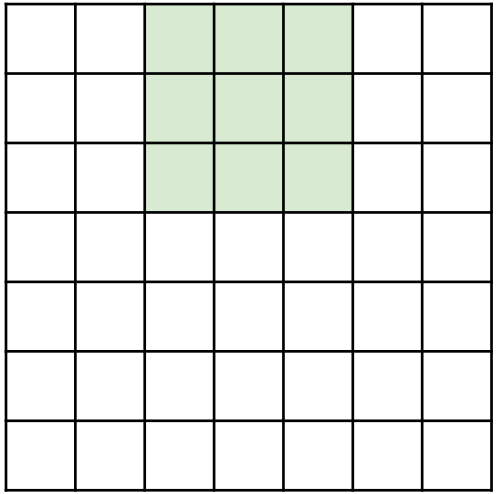
Solution: Downsample inside the network

Strided Convolution



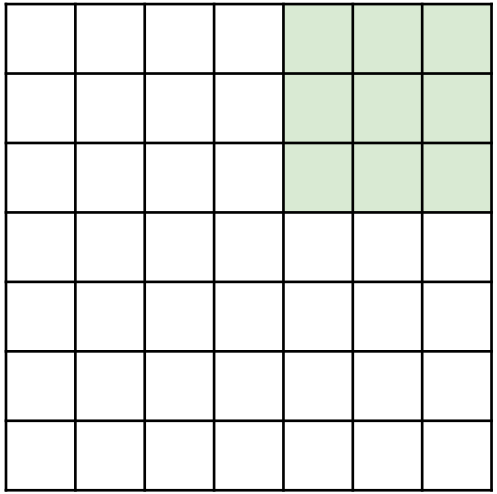
Input: 7x7
Filter: 3x3
Stride: 2

Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2

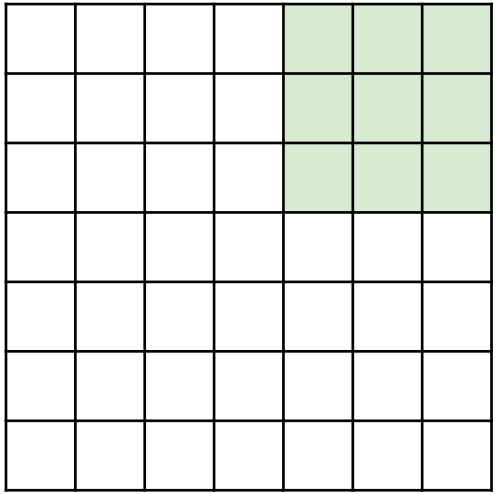
Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2

Output: 3x3

Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2

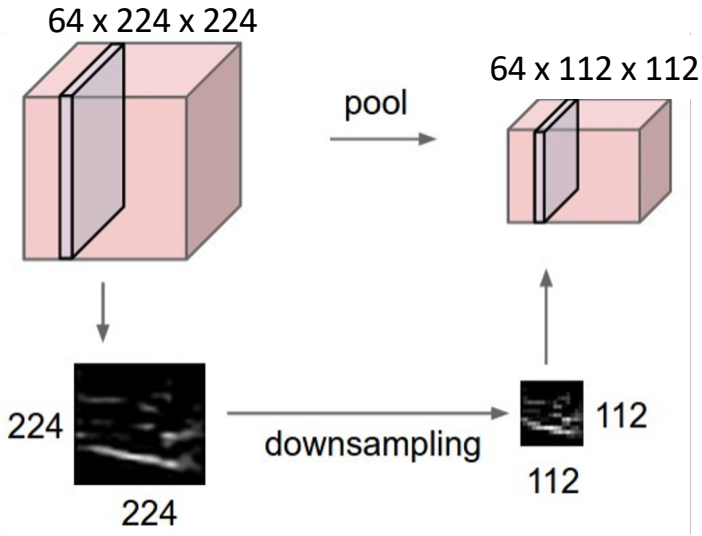
Output: 3x3

In general:
Input: N
Filter: K
Stride: S
With padding: P
Output: (N – K + 2P) / S + 1

Output: (N-K) / S + 1

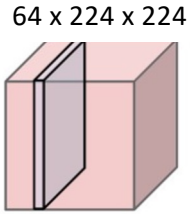
Pooling Layers: Downsampling

Another way to reduce size while aggregating(pooling) information

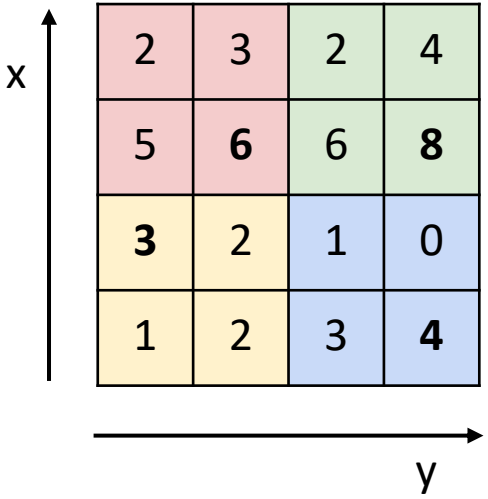


Has a choice of:
Kernel Size
Stride
Pooling function

Max Pooling



Single depth slice



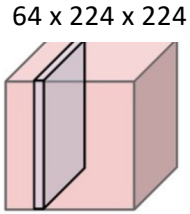
Max pooling with 2x2 kernel size and stride 2

6	8
3	4

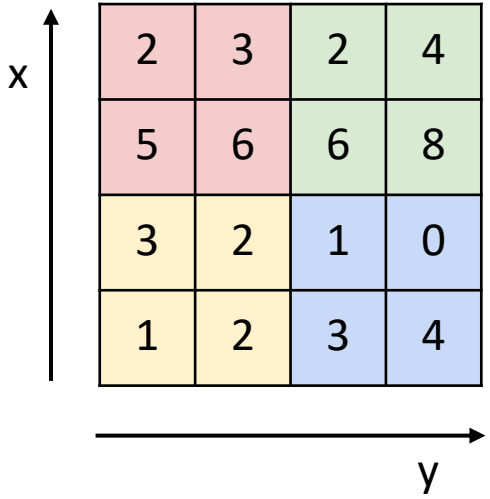
Introduces **invariance** to small spatial shifts

No learnable parameters!

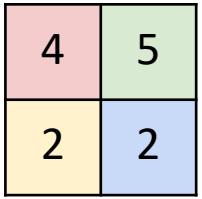
Average Pooling



Single depth slice



Avg pooling with 2x2 kernel size and stride 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)

Normalization

Why do we need to normalize our data?

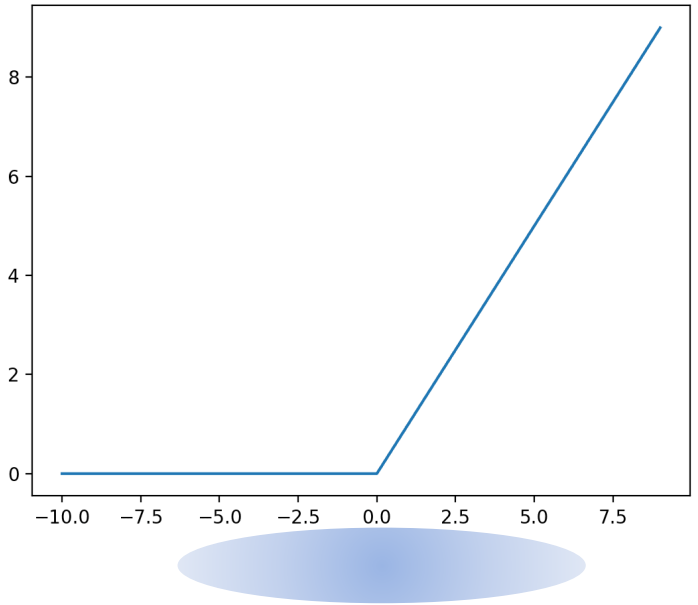
Extreme example:

- 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!

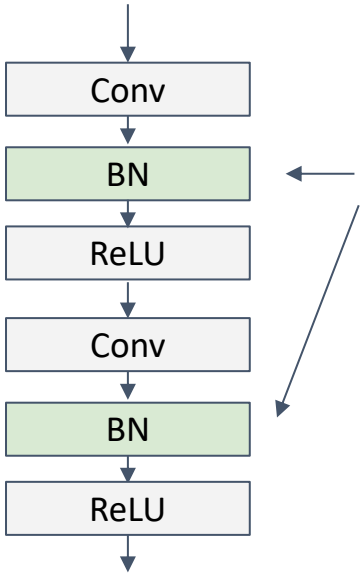
But now the data is always zero mean, unit variance…

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)

Normalization



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

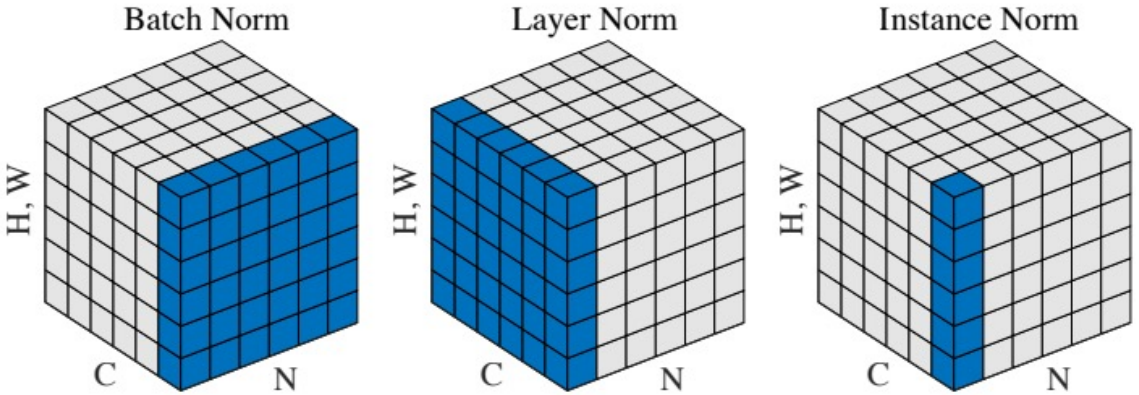
$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}$$

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

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

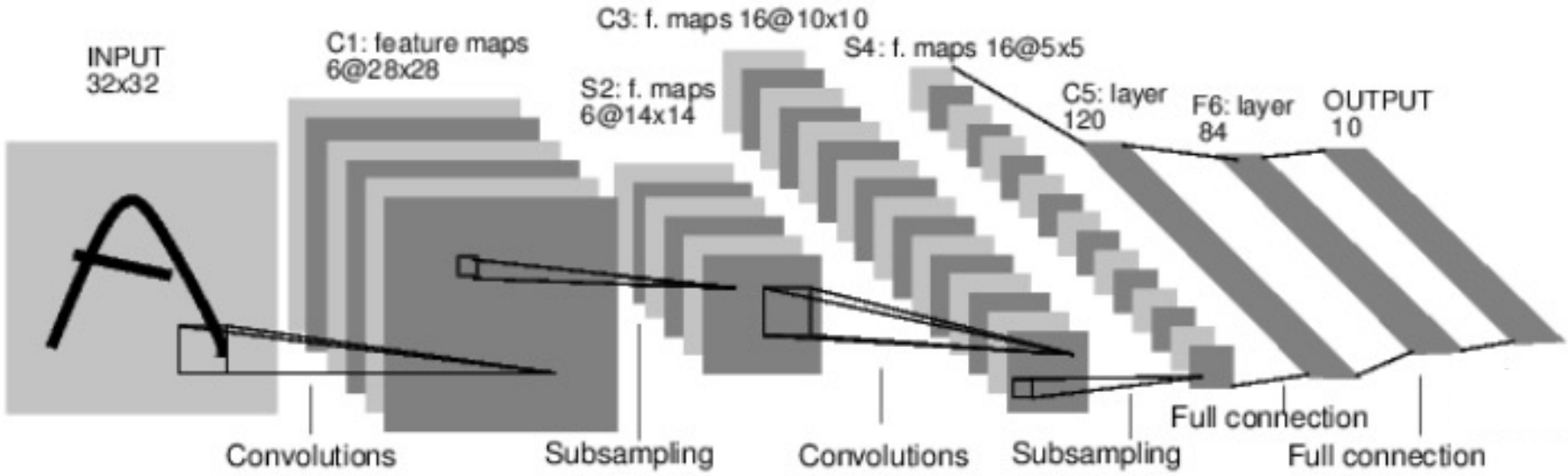
$$\gamma, \beta : 1 \times C \times 1 \times 1$$

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

Case Study: Lenet-5

Task: 10 digit classification

LeCun et al. 1998



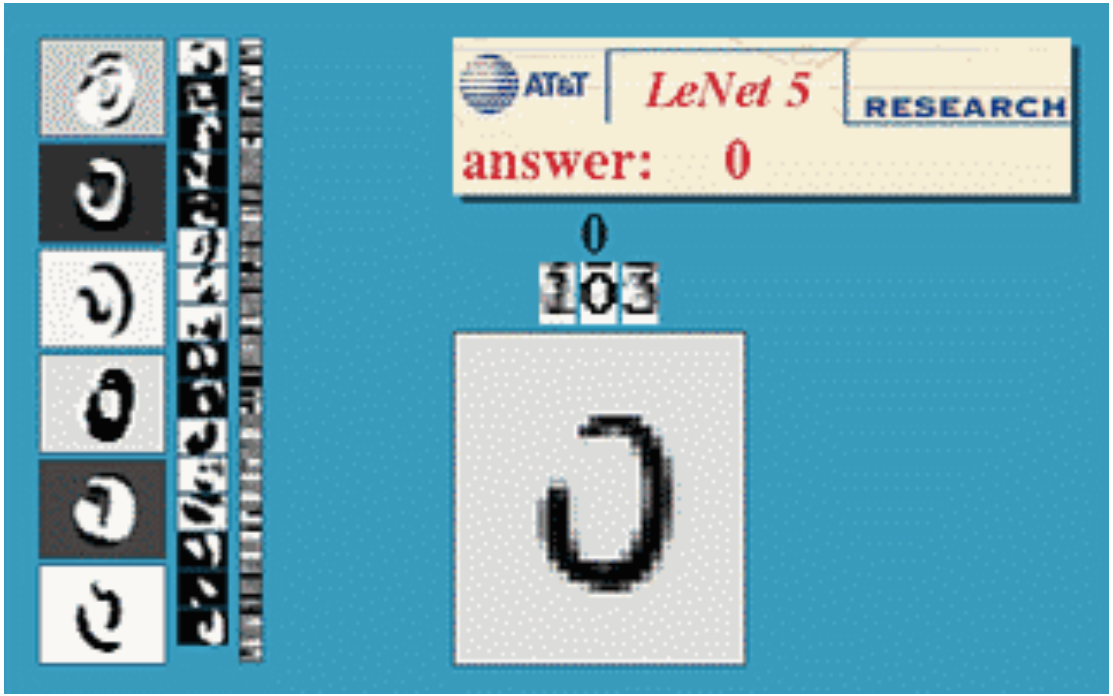
Conv filters were 5x5, applied at stride 1

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

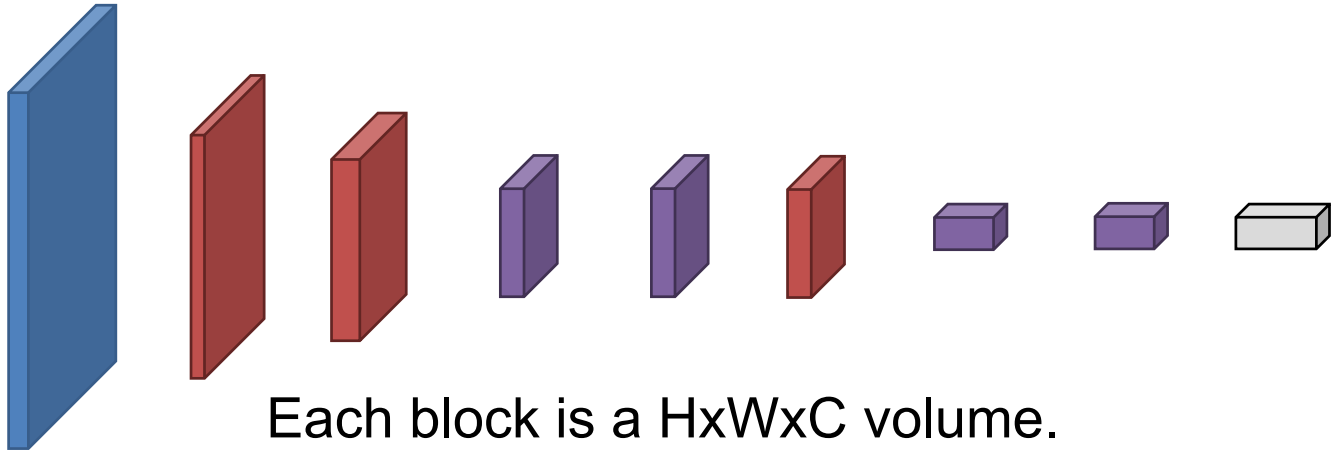


Case Study: AlexNet

[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

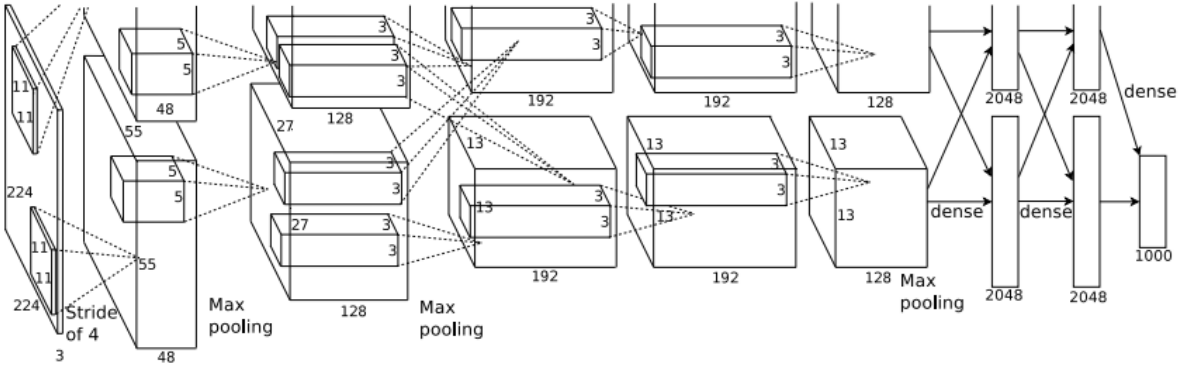


Each block is a HxWxC volume.

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?

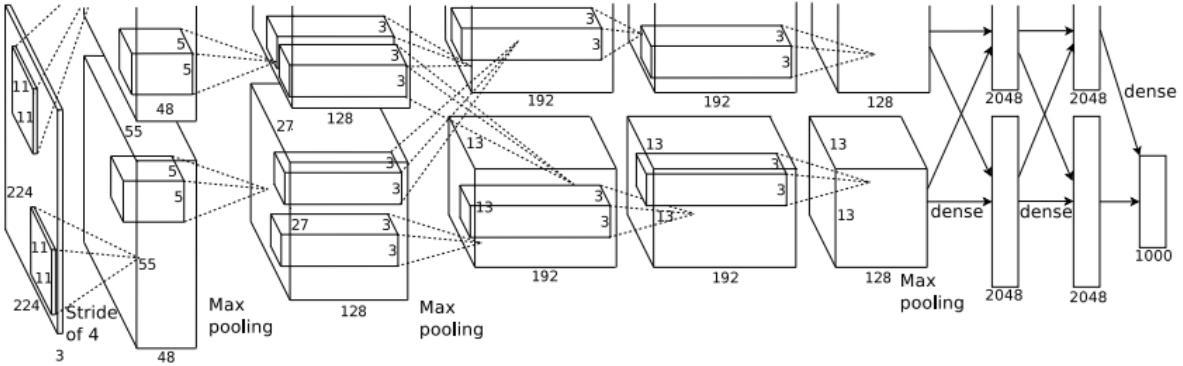
A: 55x55x96

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

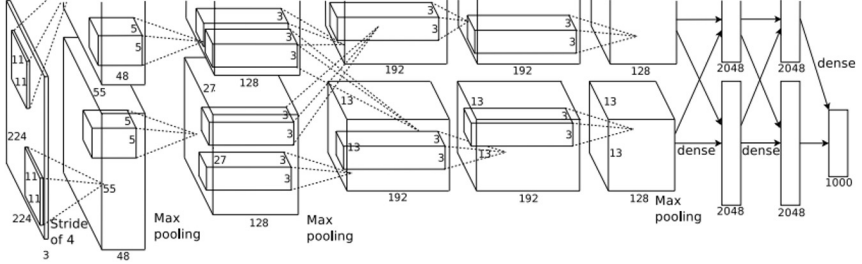
Q: What is the output volume size after Pool1?

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

Q: how many parameters in this layer?

A: 0!!!

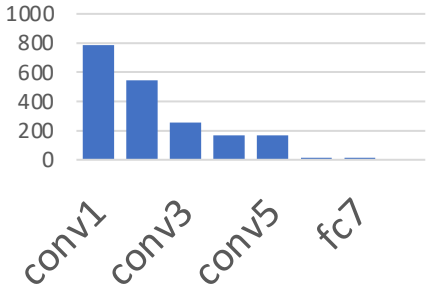
AlexNet



Q: Which part of the network incurs

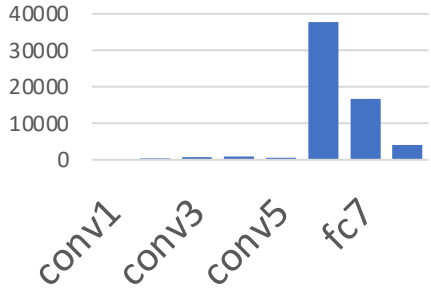
high memory usage?

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



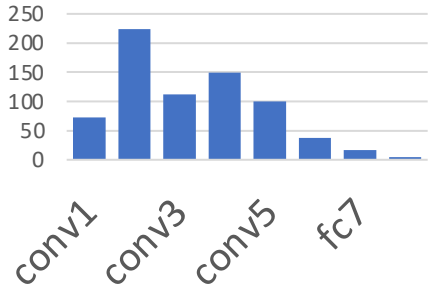
Large # of parameters?

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

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

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

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					

VGGNet [Simonyan and Zisserman 2015]

Input: 224x224x3

Simplified design rules:

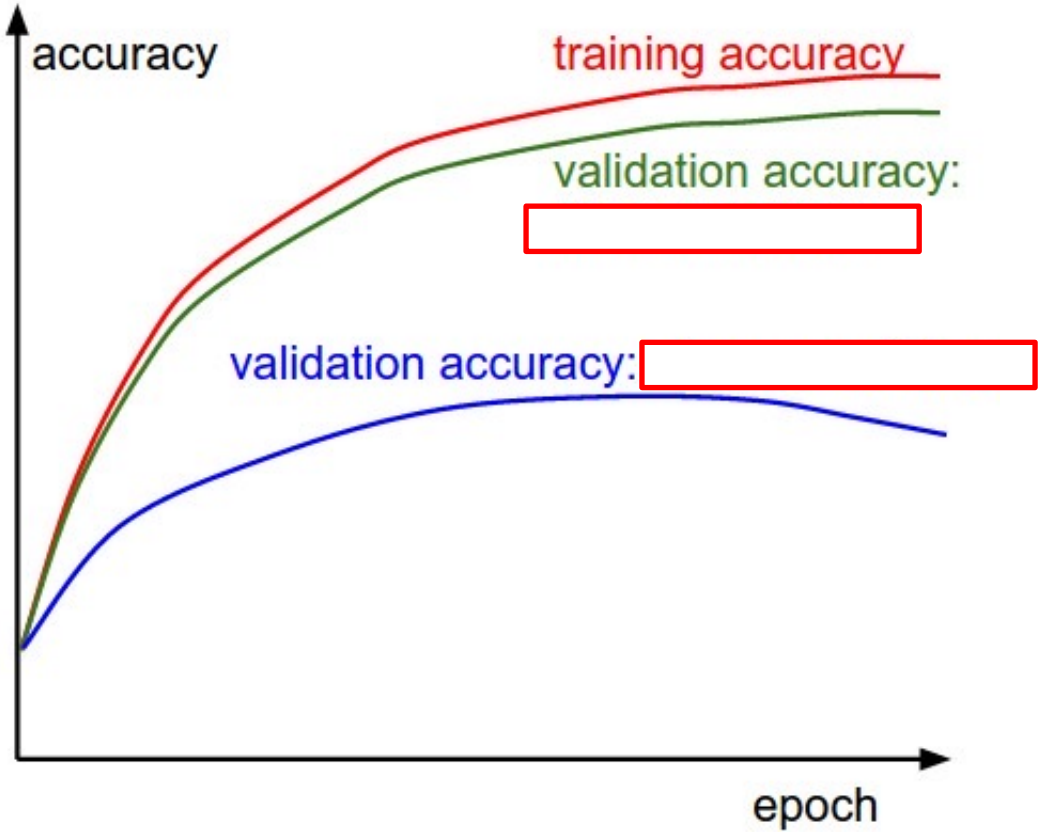
- **All kernel size 3x3**
- Always ReLu
- All max pool are 2x2, stride 2
- After pool, double the # of channels

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

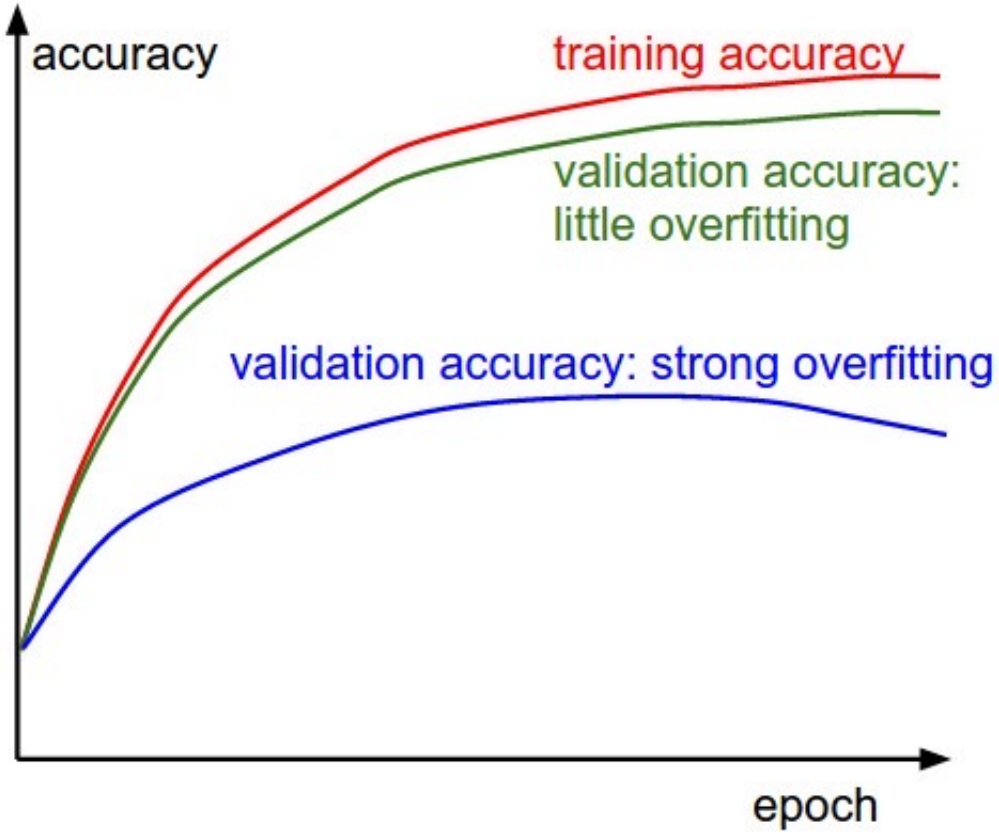
Option 1:	Option 2:
Conv(5x5, C -> C)	Conv(3x3, C -> C) Conv(3x3, C -> C)
Params: $25C^2$	Params: $18C^2$
FLOPs: $25C^2HW$	FLOPs: $18C^2HW$

Slide adapted from Johnson & Fouhey

Measuring performance over train/val sets



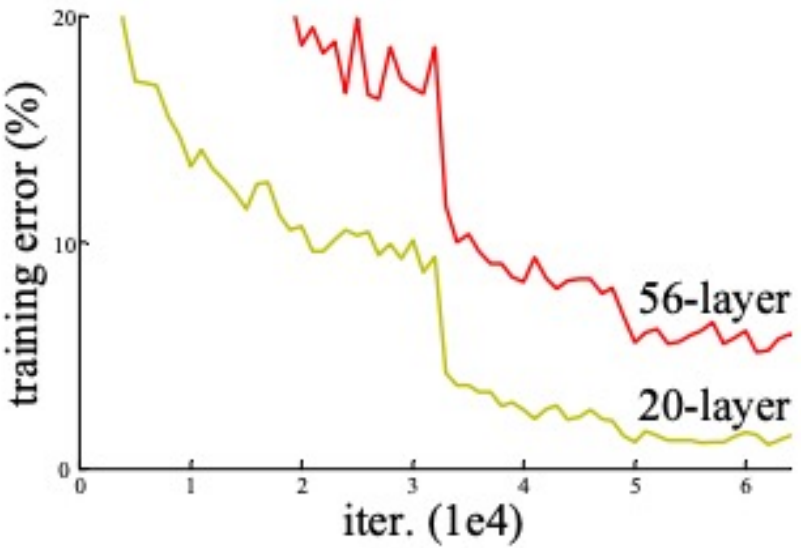
Measuring performance over train/val sets

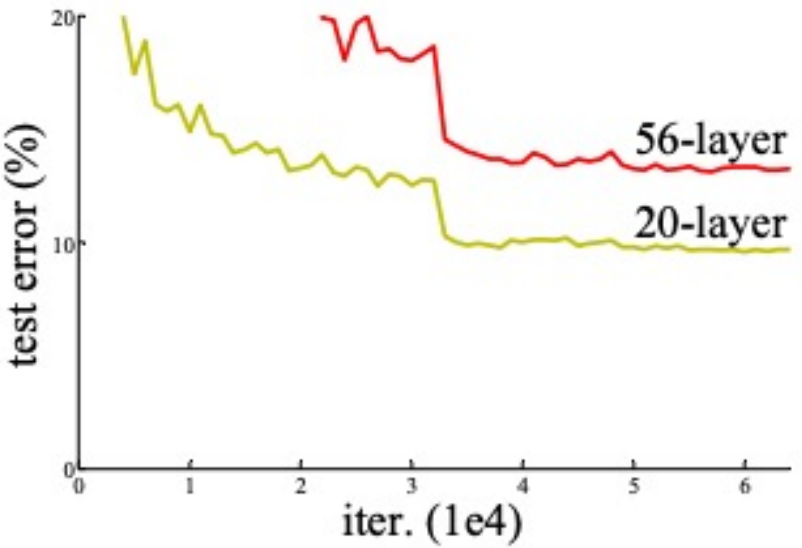


Deeper network != Better performance

Q: is this over fitting?

No! it's a matter of optimization

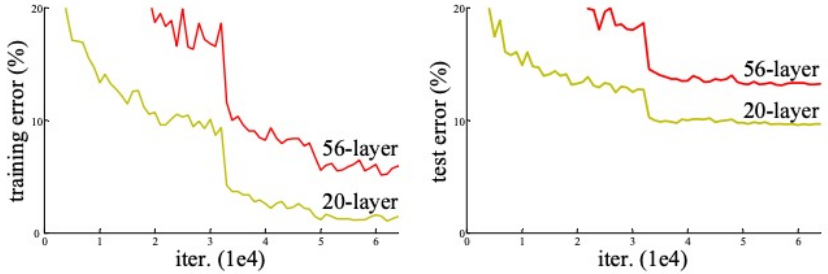




ResNet [He et al. CVPR 2016]

Naively adding more layers != better performance

[He et al. CVPR 2016]

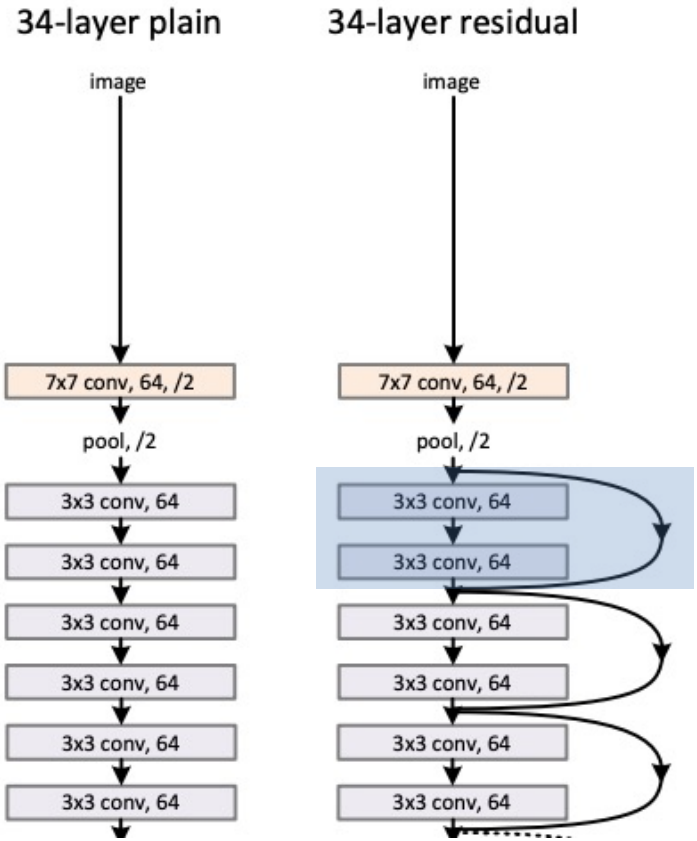


Deeper networks *should be* able to emulate shallow networks.

How? By making the extra layers to learn the identity function

But they don’t seem to learn identity by default

ResNet [He et al. CVPR 2016]



Input: 224x224x3

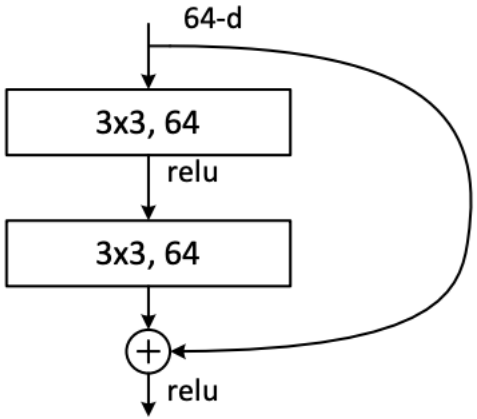
ResNet: “Residual Learning”

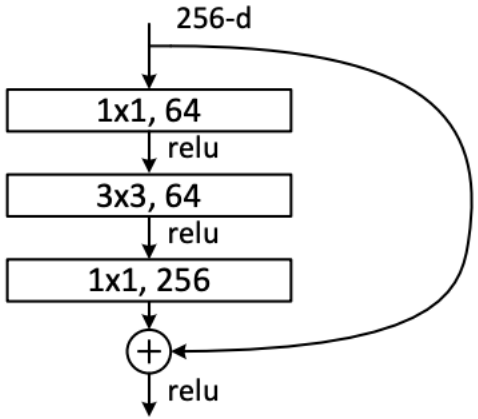
ResNet Block:

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

ResNet [He et al. CVPR 2016]

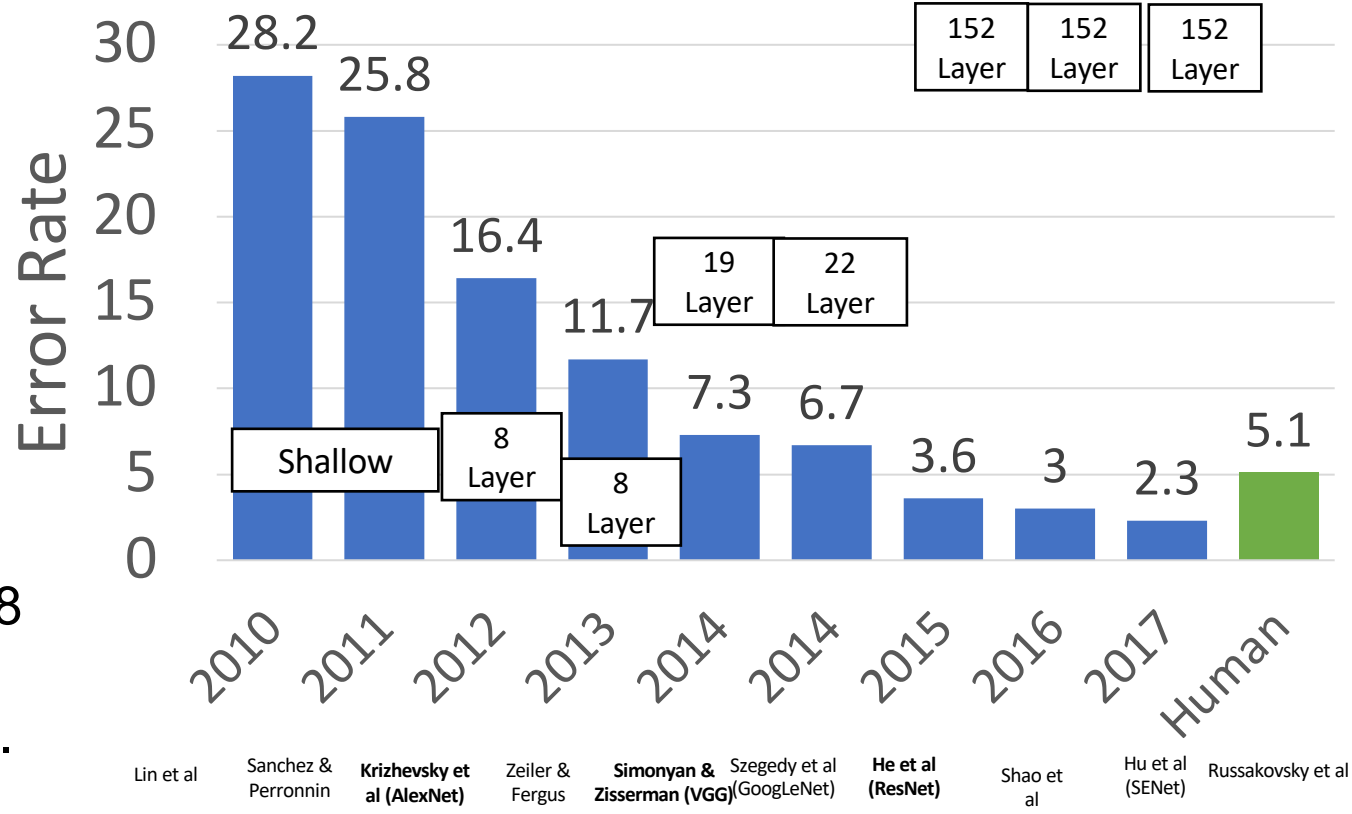
Can be made very deep: ResNet-50, 101, 152
And converge well





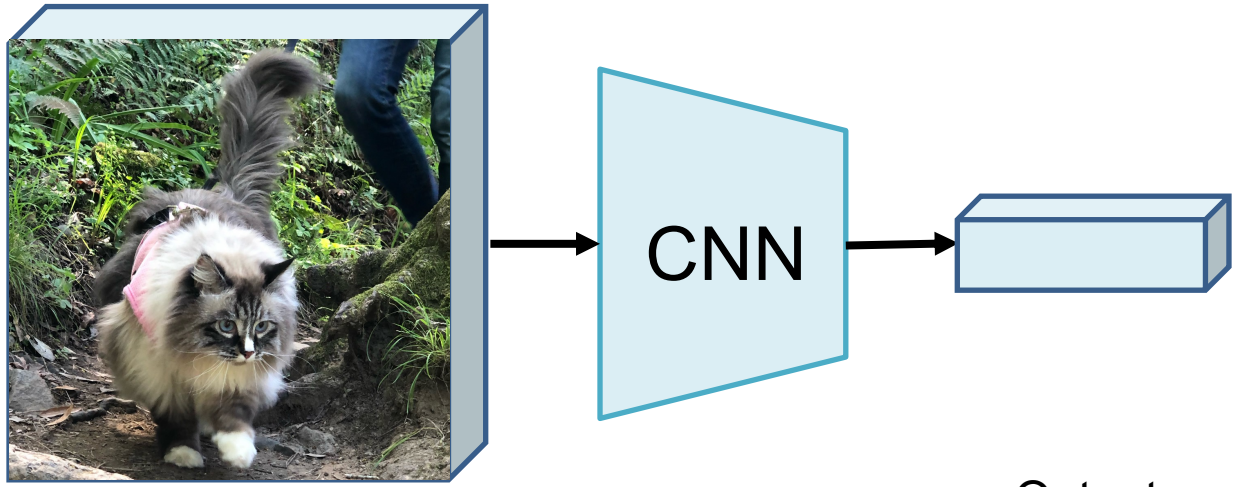
Bottleneck design:
Reduce by 1x1
Conv by 3x3
Restore by 1x1

ImageNet Classification Challenge top-5 error



In 2021, Top-5 is .98 Florence [Yuan et al. 2021]

Bottom line: CNN workflow



Input:
HxWxC

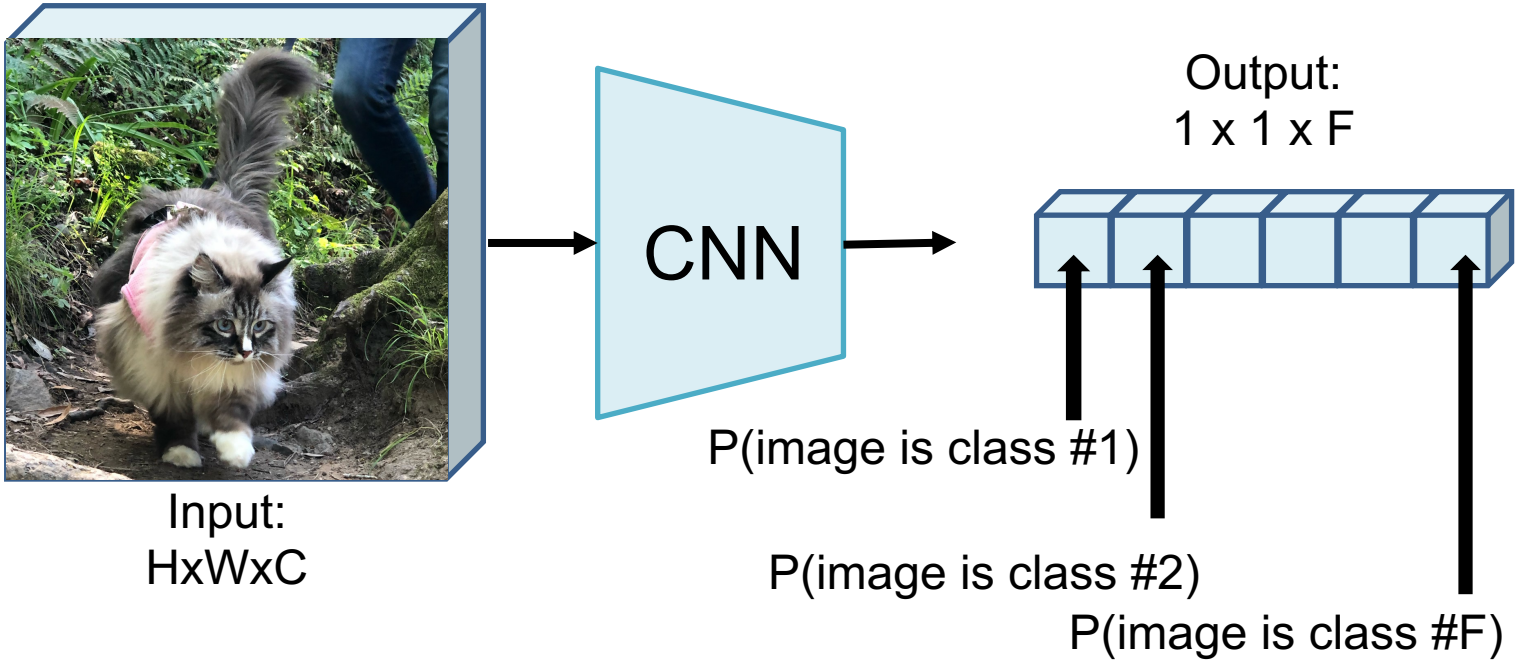
Output:
1 x 1 x F

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=5*2 = 10)

Slide modified from David Fouhey

Example: Image classification



Slide modified from David Fouhey

Example: Keypoint detection

Input:
HxWxC

CNN

Output:
1 x 1 x 2K

X coord of keypoint #1

Y coord of keypoint #1

X coord of keypoint #K

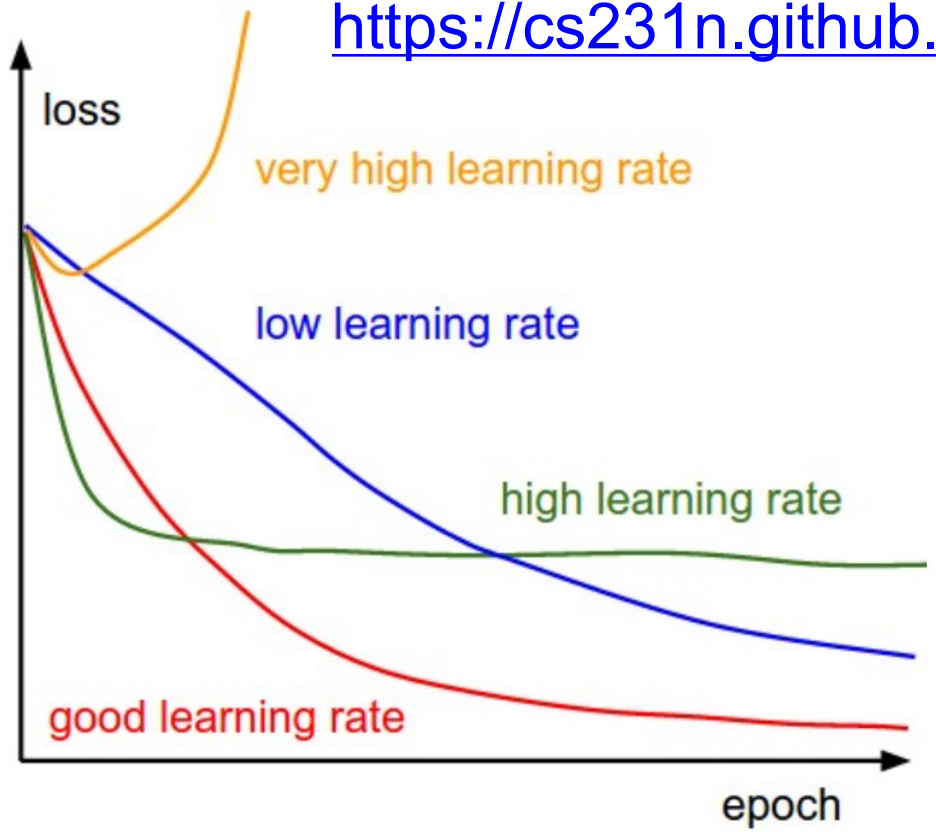
Y coord of keypoint #K

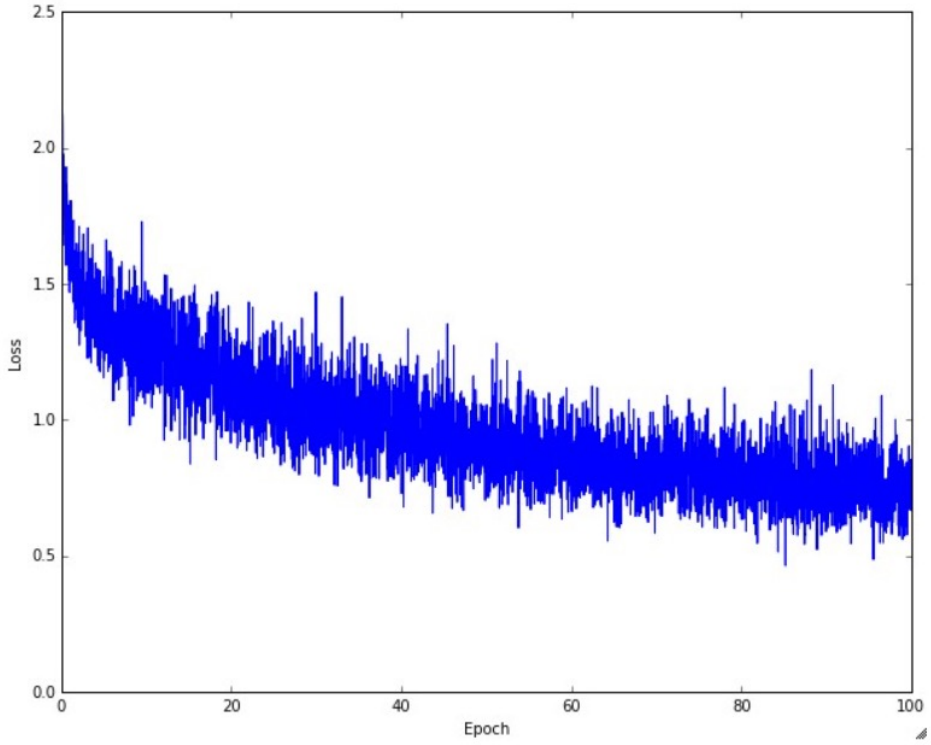
Slide modified from David Fouhey

How to train your network

Training diagnosis

https://cs231n.github.io/neural-networks-3/





Basic setup

1. Get your dataset
2. Design your CNN architecture (start by reusing (or simplified version of) AlexNet/ResNet)
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

What if your data is not big enough?

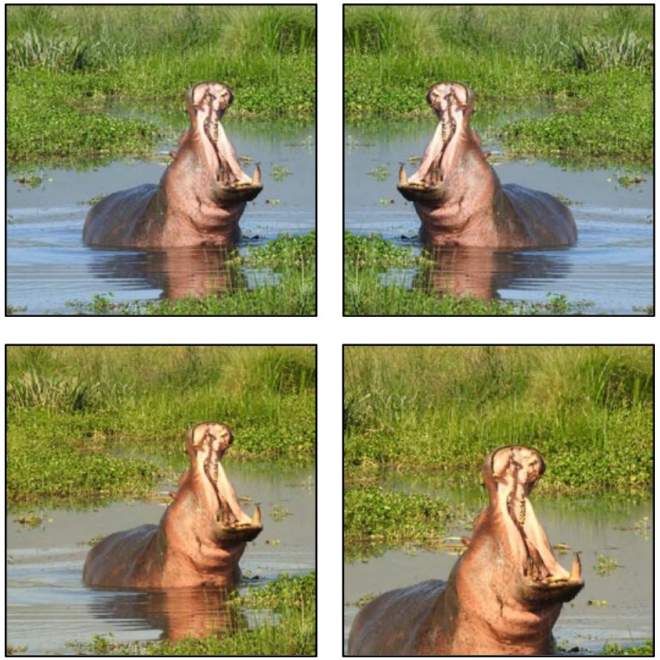
Horizontal Flip

Color Jitter

Image Cropping

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

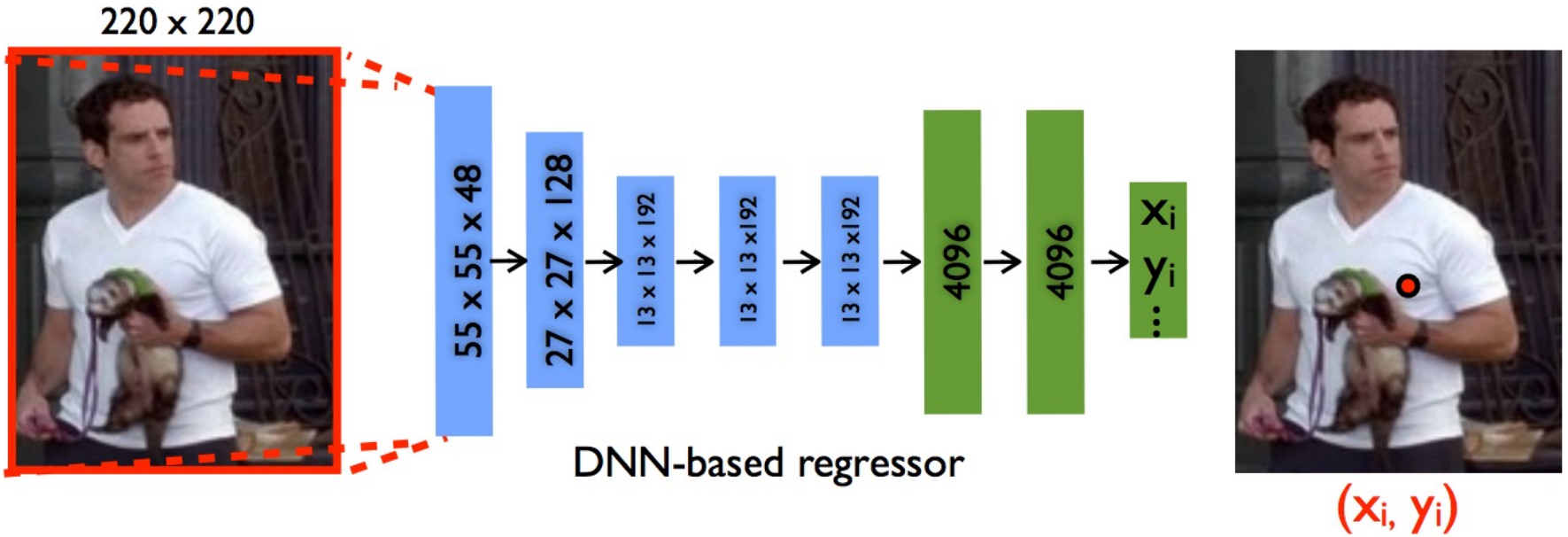
Defined wrt to the cat

Original

After flip + color aug

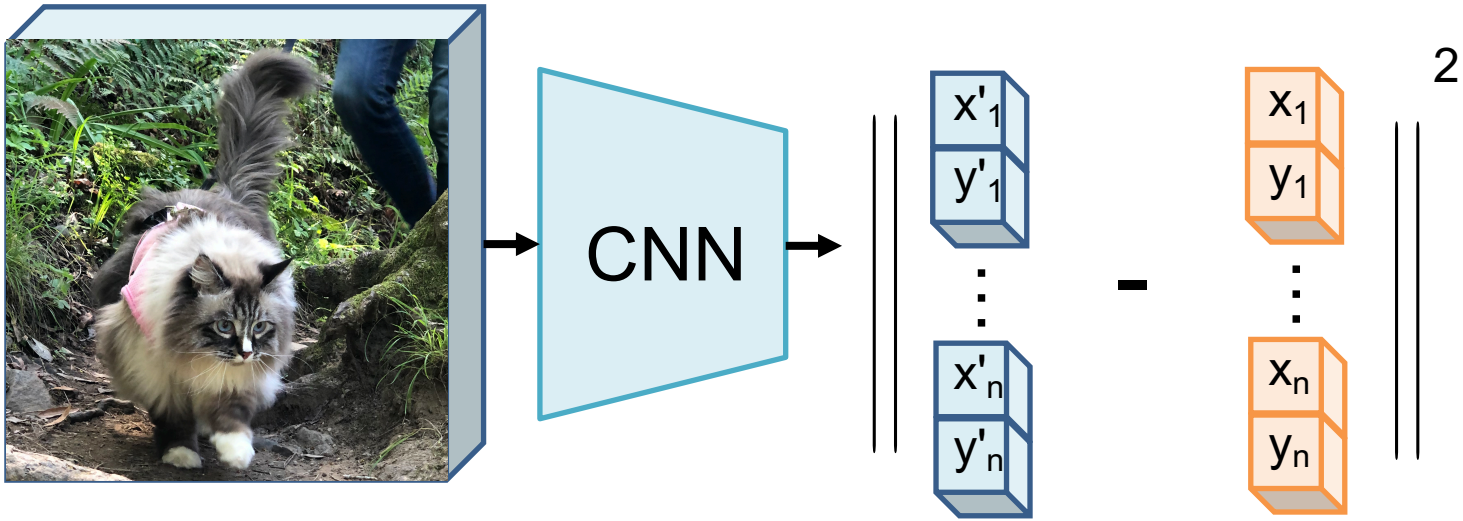
Correct

A good starting point



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

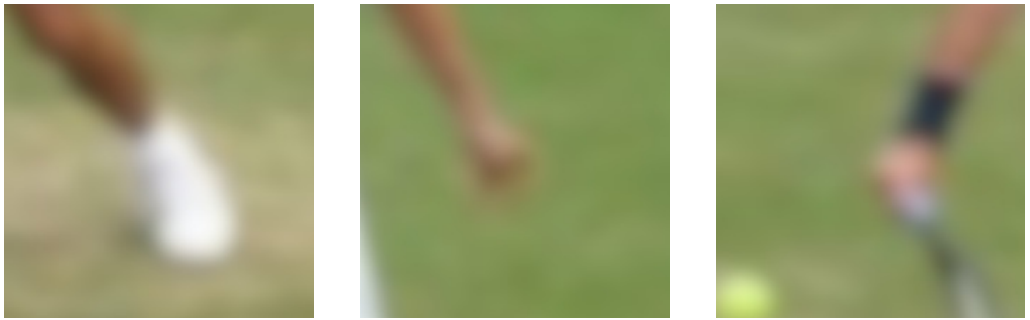
Regression Objective



This is what you implement in part 1. But this is not what the SoTA models do in practice

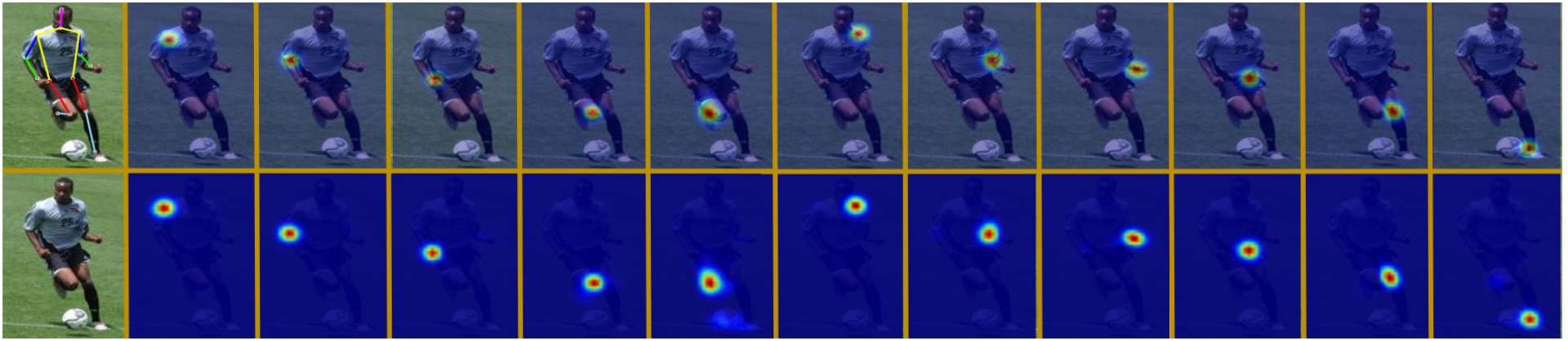
Downsides of regression objective

Locally a lot of things look similar!!



With regression objective you have to commit to ONE location and only get one training signal on how correct that location was.

Belief/Confidence map formulation



[Thompson et al. NeurIPS 2014, CVPR 2015, Convolutional Pose Machine, Wei et al. CVPR 2016, (figure credit: Ning et al TMM 2017)…]

For K keypoints, train model to predict K many sheets (h x w) of scores of how likely the pixel is k-th keypoint

Problem

So far, we've only seen examples that output a vector representation out of an image.

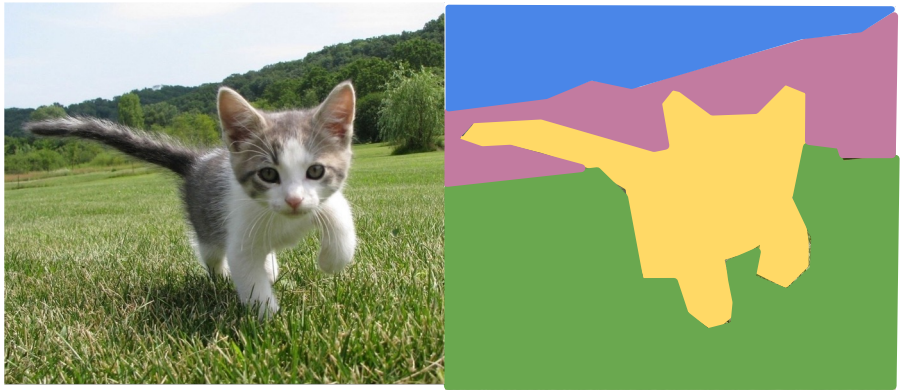
How do we do dense (per-pixel) predictions?

Dense Prediction

Needed for many things:

Image Synthesis!
(next lecture)

Semantic Segmentation

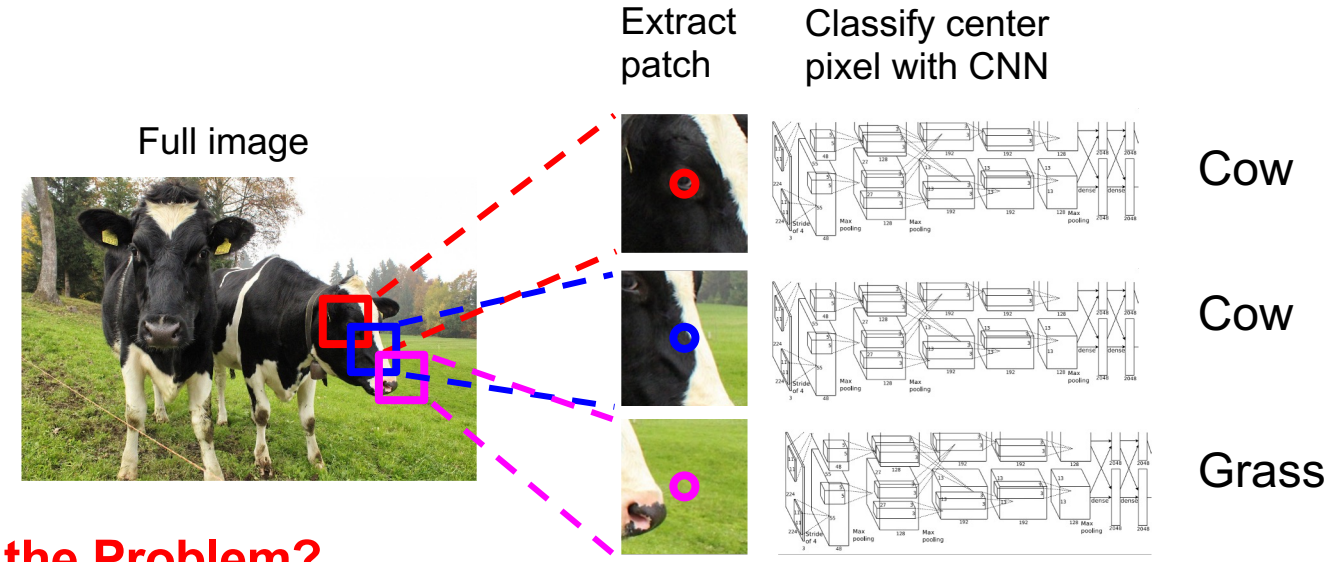


GRASS, CAT, TREE, SKY

Keypoint detection

Does this pixel contain right wrist?

Idea 1: Dense prediction by Sliding Window



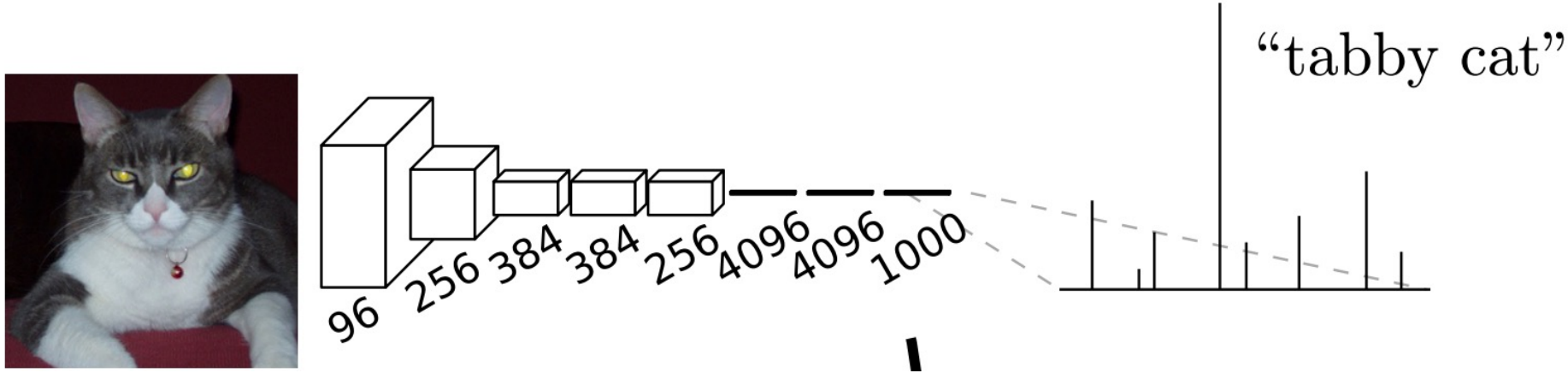
What is the Problem?

Very inefficient! Not reusing shared features between overlapping patches

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

Instead, make the whole network convolutional!

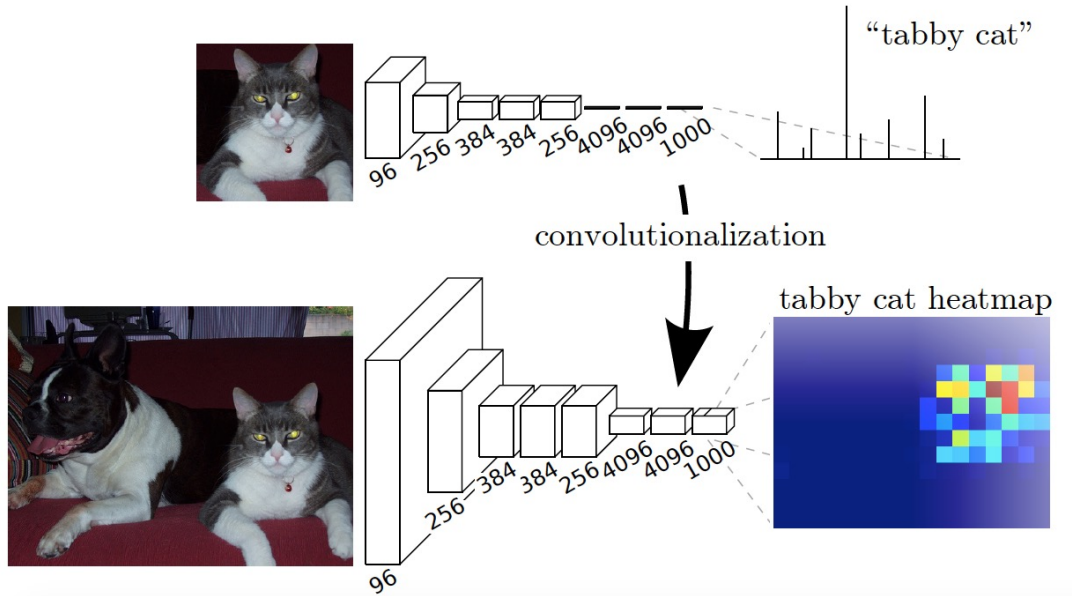
Think of the last few fully connected network as a convolutions with kernels (receptive field) that cover the entire image ➔ can turn it into 1x1 convs



Fully convolutional networks

- Design a network with only convolutional layers, make predictions for all pixels at once

Convert all FC into 1x1 convolutions

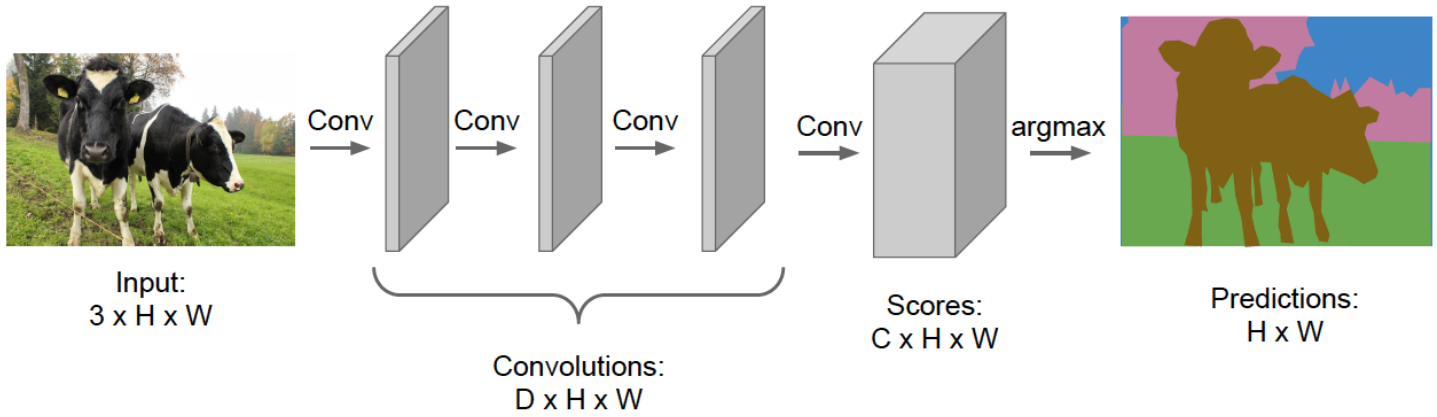


J. Long, E. Shelhamer, and T. Darrell, Fully Convolutional Networks for Semantic Segmentation, CVPR 2015

Slide courtesy of Lana Lezebnik

Fully convolutional networks

- Design a network with only convolutional layers, make predictions for all pixels at once
- Ideally, we want convolutions at full image resolution, but implementing that naively is too expensive

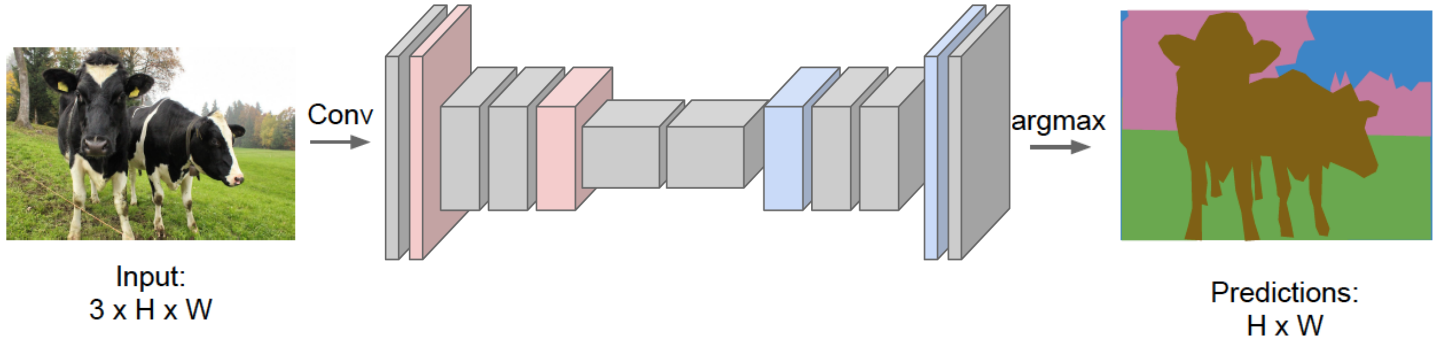


Slide courtesy of Lana Lezebnik

Source: Stanford CS231n

Fully convolutional networks

- Design a network with only convolutional layers, make predictions for all pixels at once
- Ideally, we want convolutions at full image resolution, but implementing that naively is too expensive
  - Solution: first downsample, then **upsample**



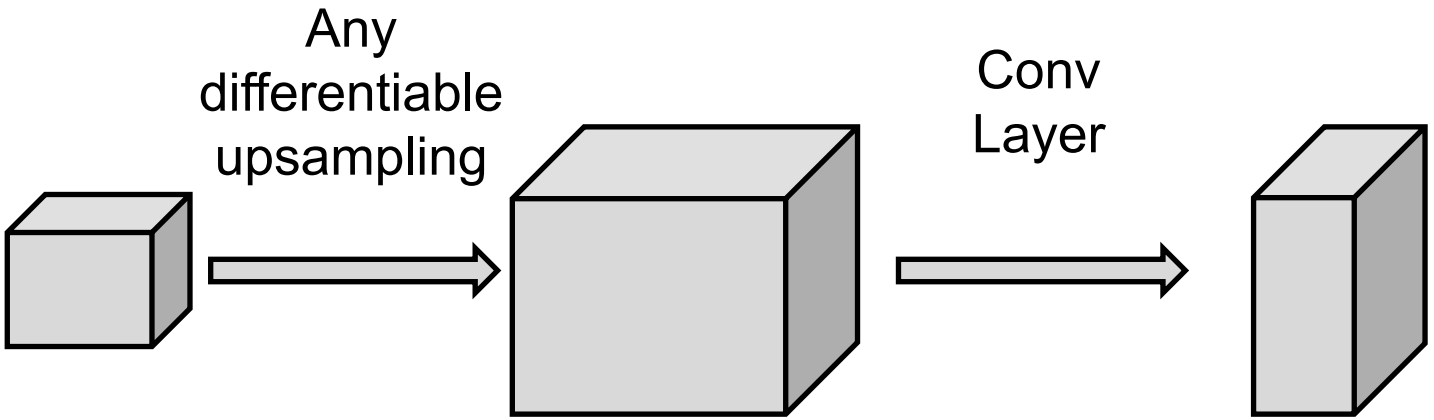
Slide courtesy of Lana Lezebnik

Source: Stanford CS231n

HOW TO UPSAMPLE WITH CONVNETS?

Simple solution

- Upsample, followed by a regular Convolution

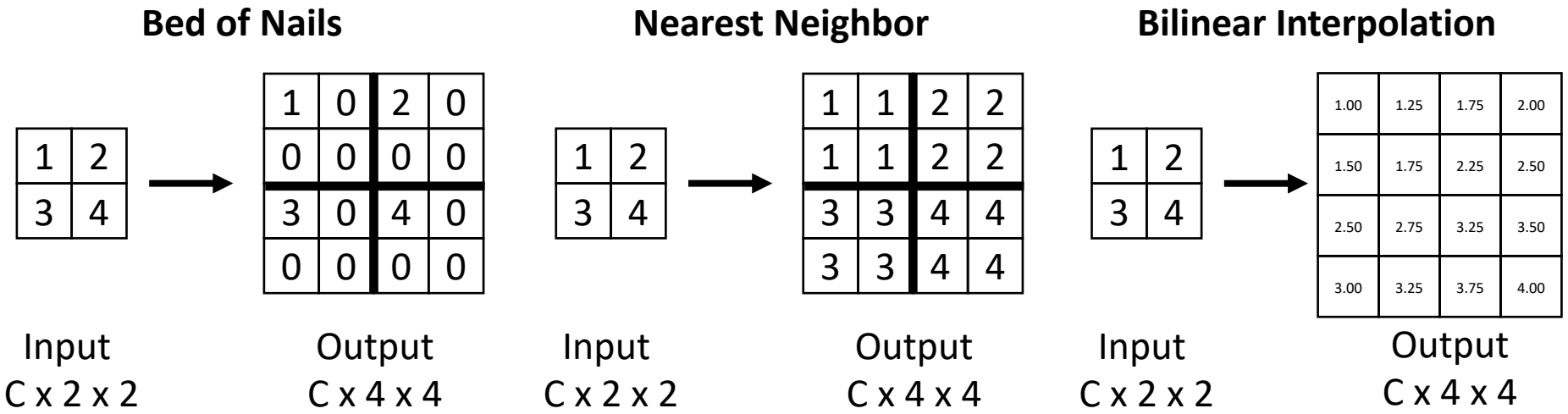


Input:
B x C_{in} x H x W

After upsample, factor 2:
B x C_{in} x 2H x 2W

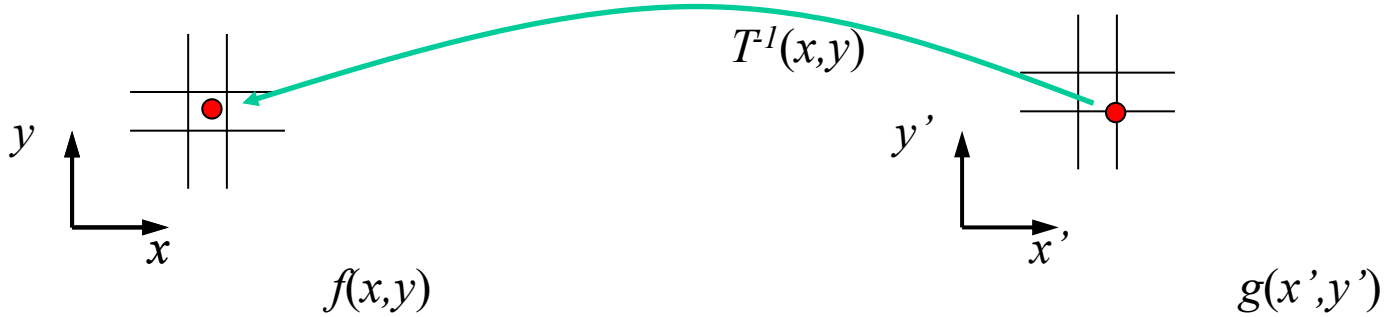
After conv:
B x C_{out} x 2H x 2W

How to Upsample

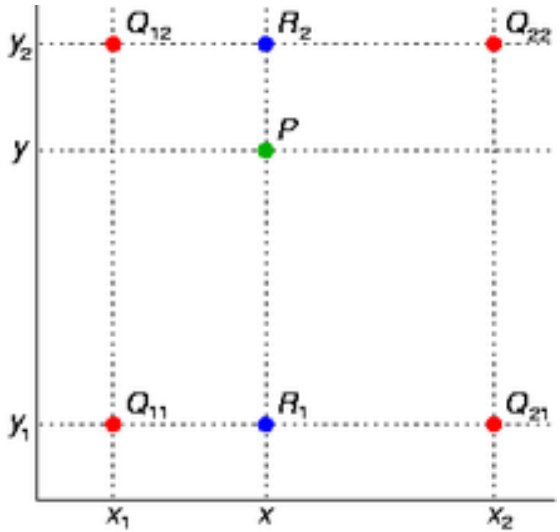


Recall from Morphing Lecture: Inverse warping

Don’t splat! Do inverse warping.
You know this from project 3



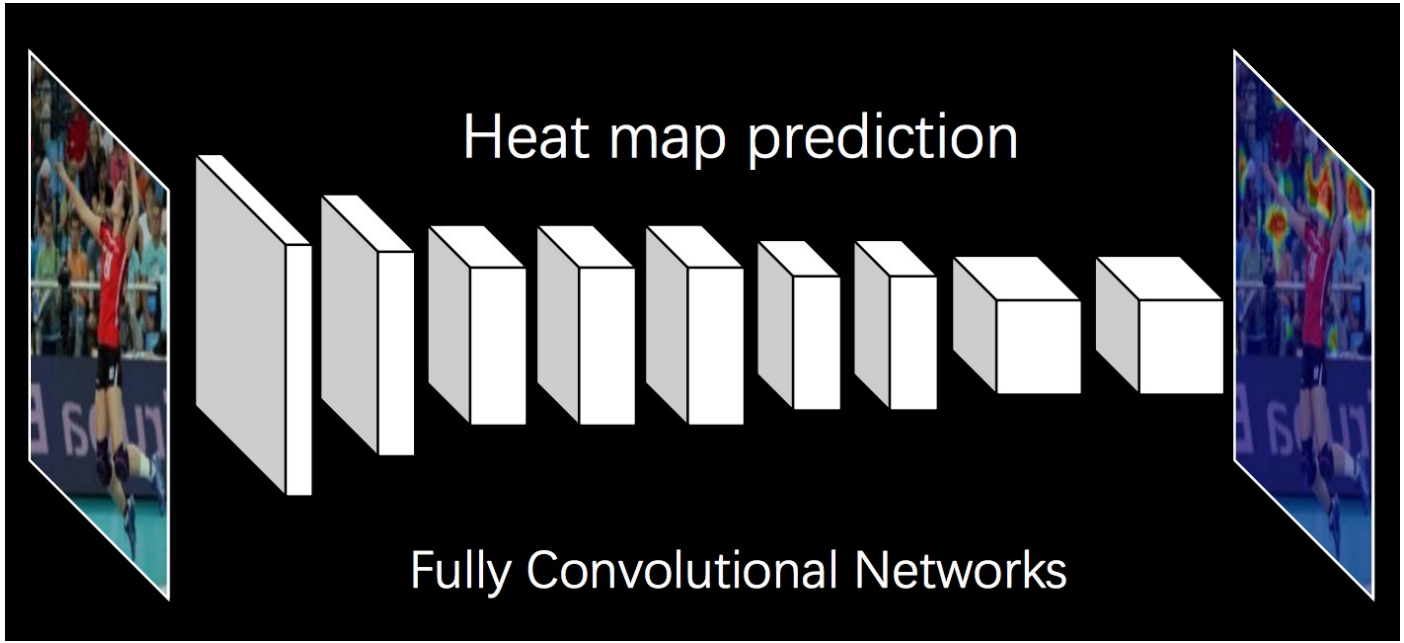
Recall: Bilinear Interpolation

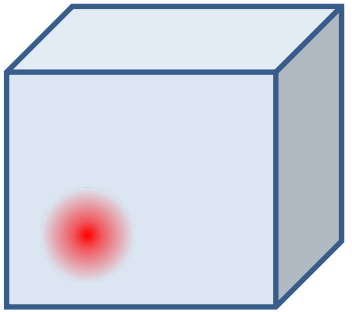


http://en.wikipedia.org/wiki/Bilinear_interpolation

```
Help interp2
```

Application to pose detection: Predict heat maps





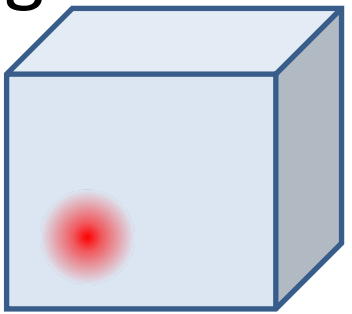
Target: K+1 x H x W Gaussian around (x,y) for k-th keypoint in the k-th channel

K+1 for K parts + background

You will implement this in project 5!

L2 Training Loss

- L2 loss on the target heatmap (peaky gaussian around the gt keypoint)



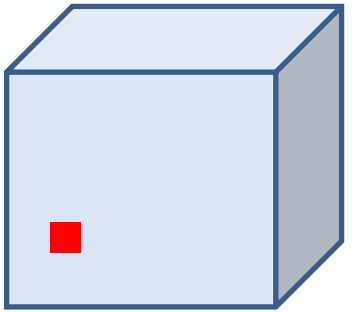
$$L = \sum_{k=1}^{K+1} \sum_{(x,y)} ||b^k(x,y) - b^k_*(x,y)||$$

Target “belief map” : K+1 x H x W Gaussian around (x,y) for k-th keypoint in the k-th channel

You will implement this in project 5!

Log Loss Training Loss

- Log loss (or cross entropy loss) on the target heatmap probabilities
- The target must also sum to 1
- Mask RCNN just uses 1 at the target, 0 everywhere else.
- Experiment

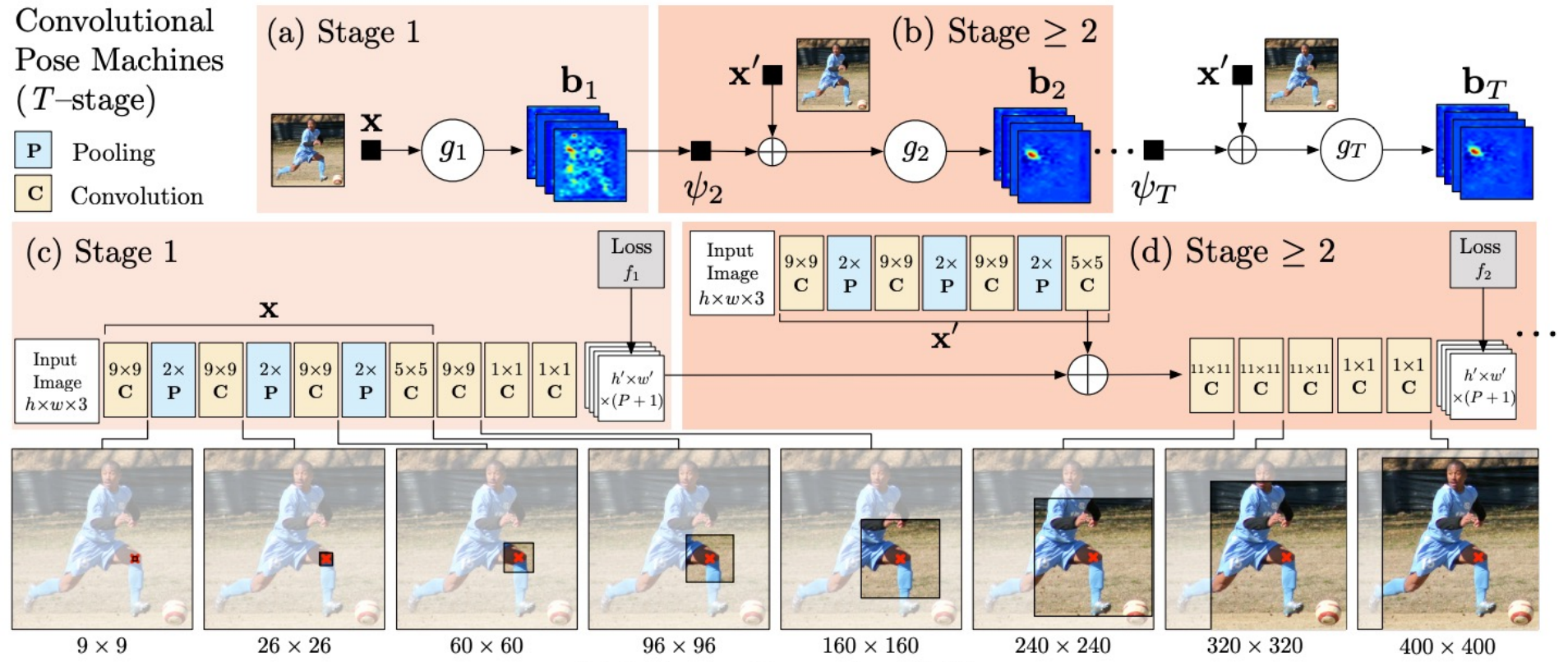


Target “belief map” : K+1 x H x W
1 at Grount truth location (x,y) for k-th keypoint in the k-th channel

[Wei et al CVPR 2016]

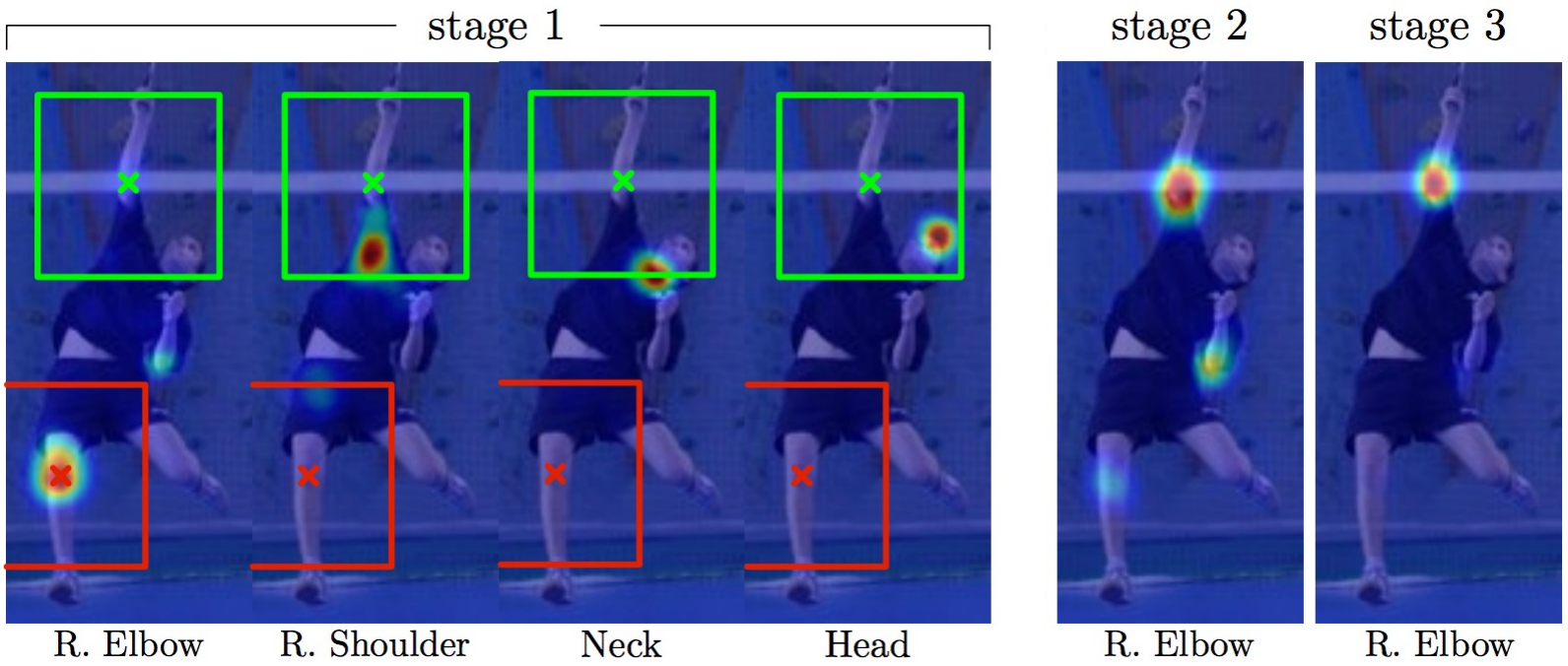
Convolutional Pose Machines

Base architecture for OpenPose



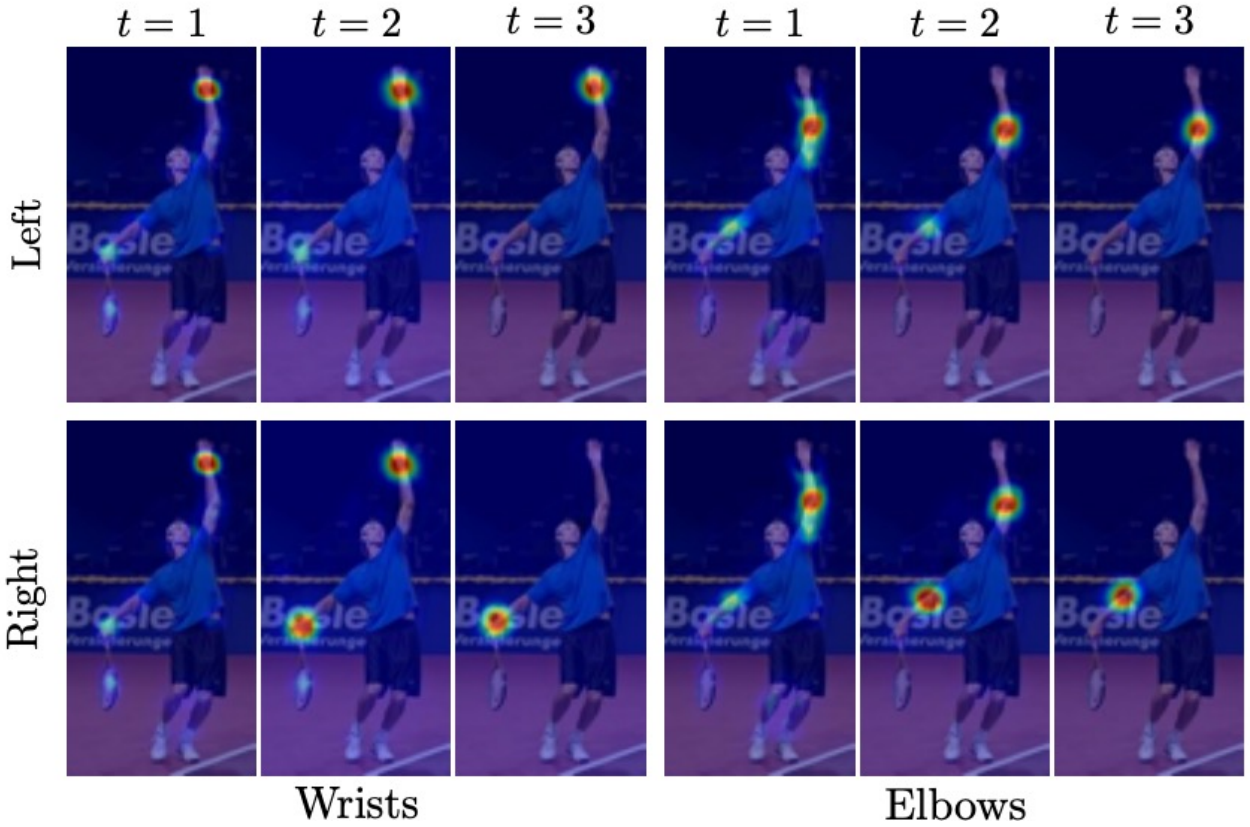
[Wei et al CVPR 201

Convolutional Pose Machines

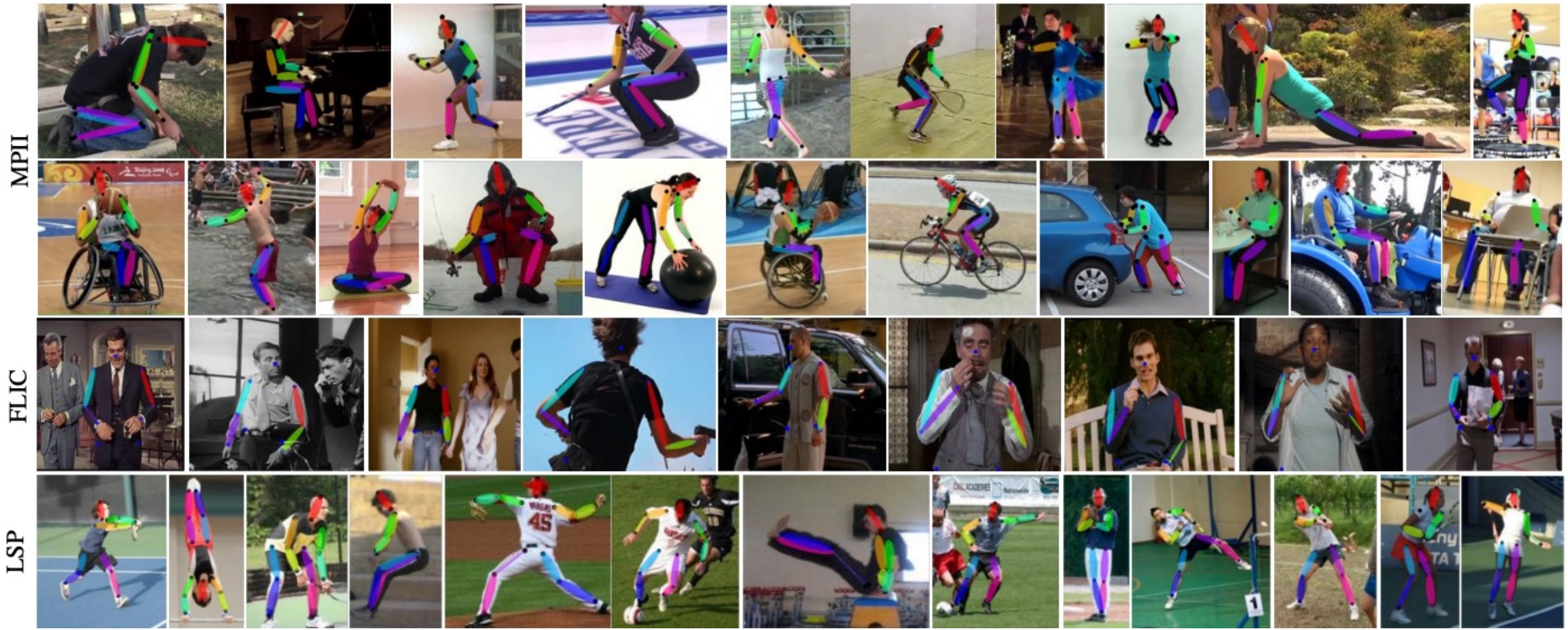


[Wei et al CVPR 2016]

Convolutional Pose Machines



Results



OpenPose

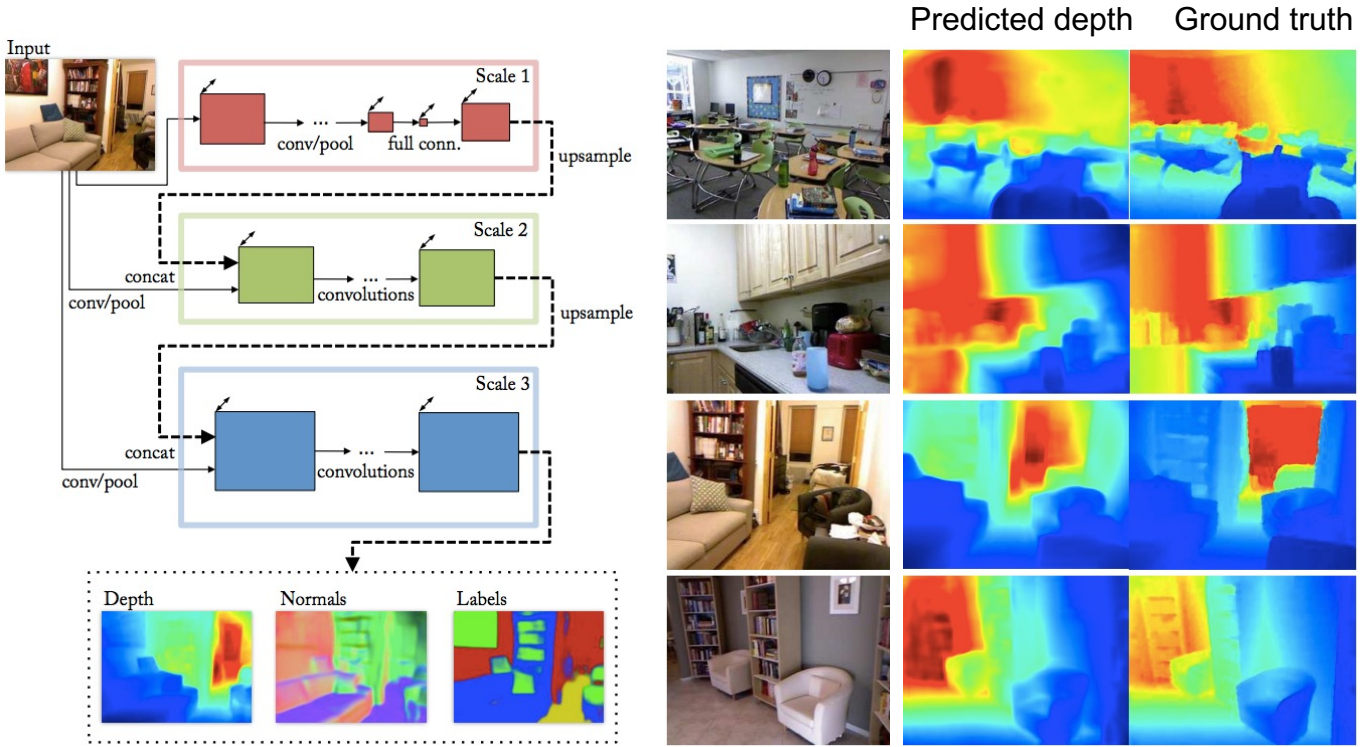
Great opensource tool, builds on convolutional pose machine architecture, adapted to multiple people

Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, Yaser Sheikh ‘16-17

Other dense prediction tasks

- Depth estimation
- Surface normal estimation
- Colorization
- Image synthesis (next week)
- ….

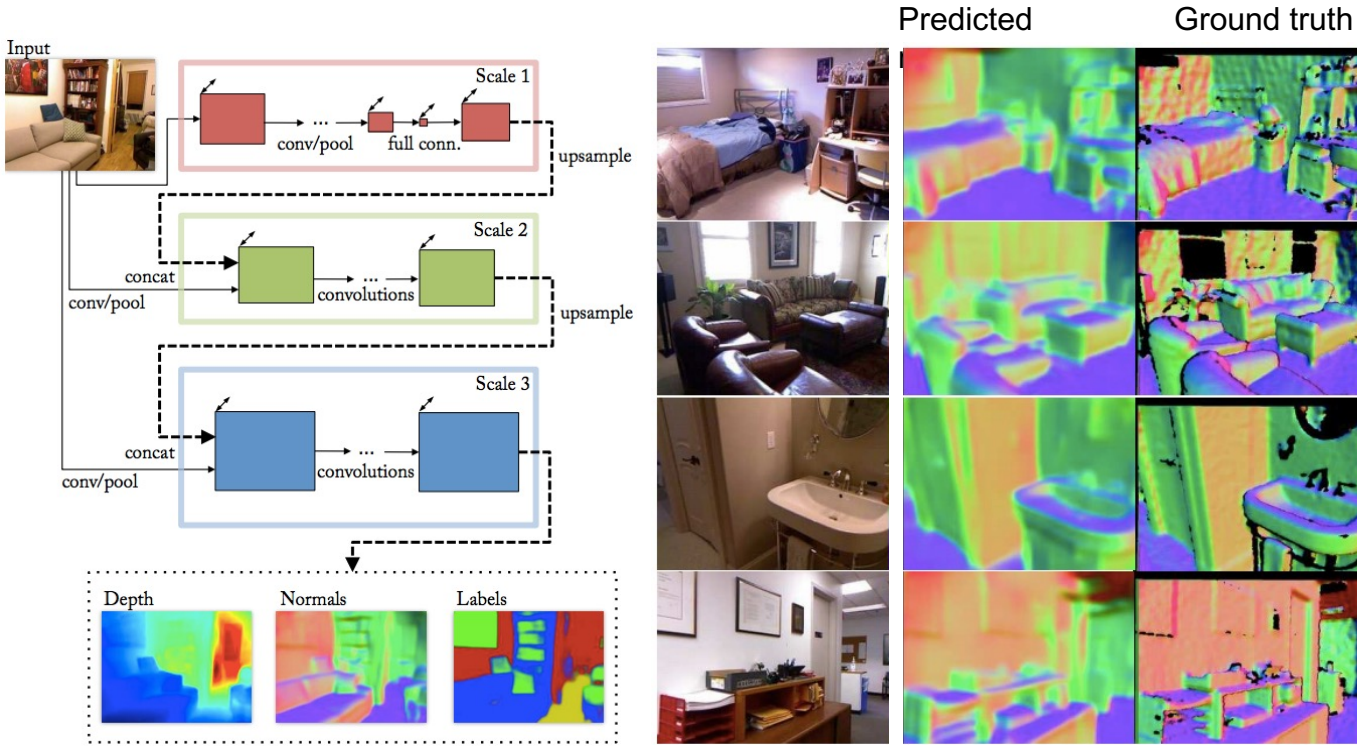
Depth and normal estimation



D. Eigen and R. Fergus, [Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture](), ICCV 2015

Slide courtesy of Lana Lezebnik

Depth and normal estimation



D. Eigen and R. Fergus, Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture, ICCV 2015

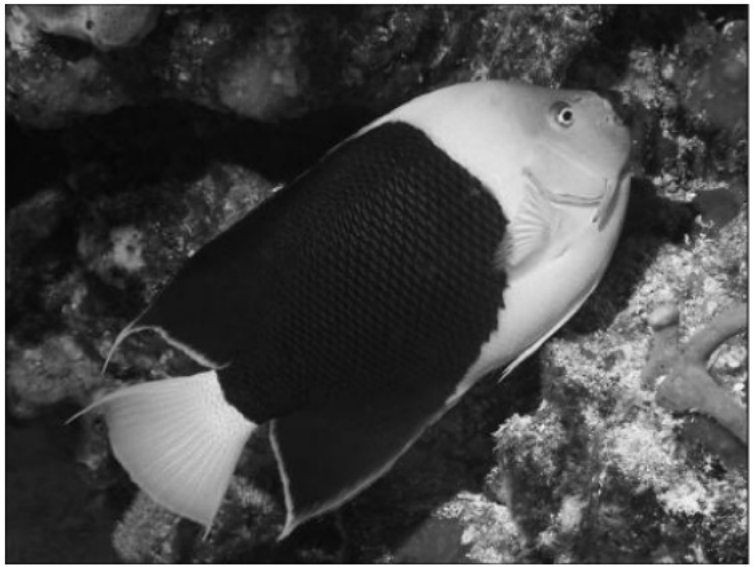
Slide courtesy of Lana Lezebnik

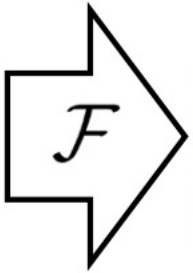
Latest: Midas, DPT

Great tool for final project!

Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer
René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, Vladlen Koltun, PAMI 2020

Colorization (self-supervised learning)



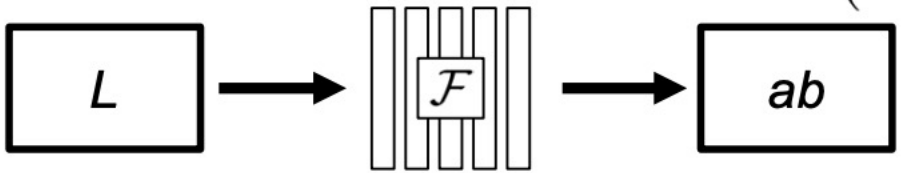


Grayscale image: *L* channel

$\mathbf{X} \in \mathbb{R}^{H \times W \times 1}$

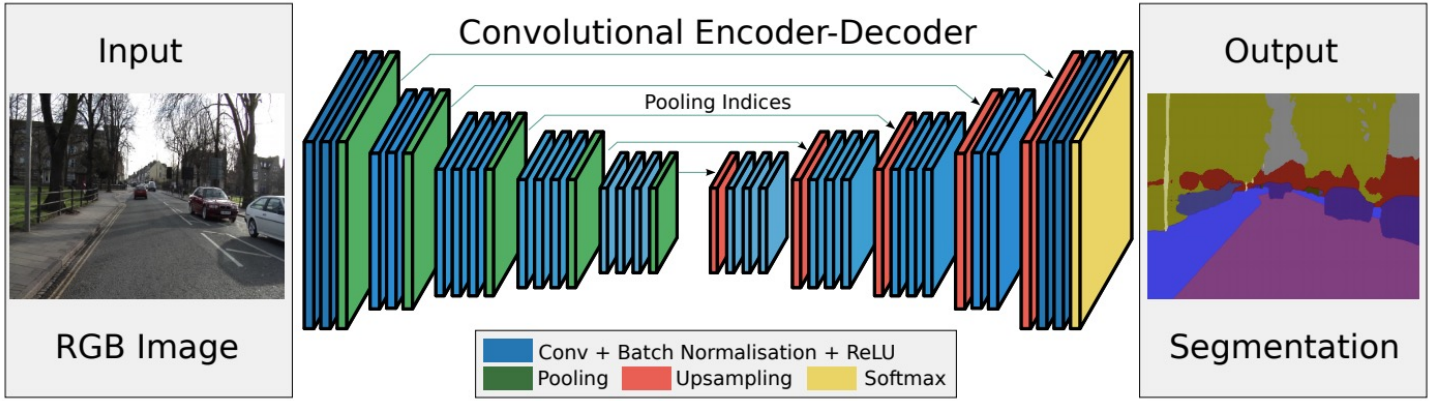
Concatenate (*L*,*ab*) channels

$(\mathbf{X}, \widehat{\mathbf{Y}})$



Zhang, Isola, Efros. ***Colorful Image Colorization.*** In *ECCV*, 2016.

Examples for segmentation: SegNet

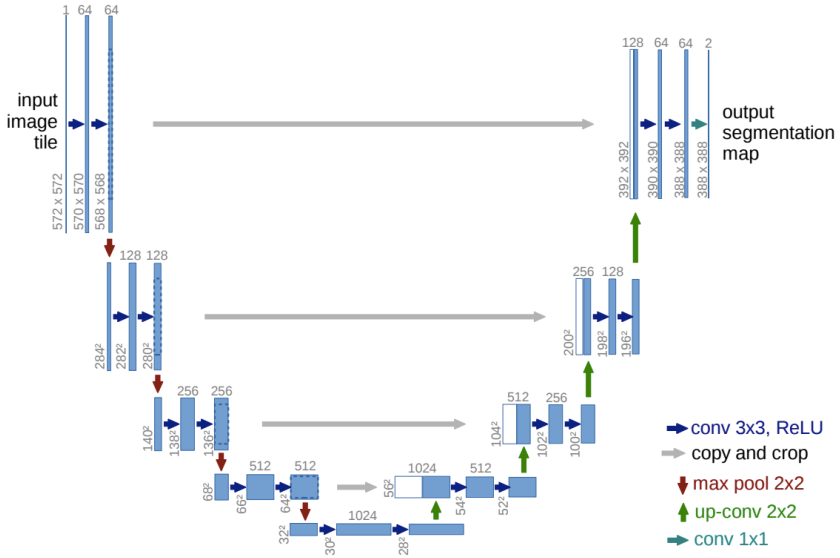


Drop the FC layers,
get better results

V. Badrinarayanan, A. Kendall and R. Cipolla, SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation, PAMI 2017

Other upsampling networks: U-Net

- Fuse upsampled higher-level feature maps with lower-level feature maps (via upsampling)
- Fuse by concatenation, predict at the end



O. Ronneberger, P. Fischer, T. Brox U-Net: Convolutional Networks for Biomedical Image Segmentation, MICCAI 2015

Slide courtesy of Lana Lezebnik

Summary of upsampling architectures

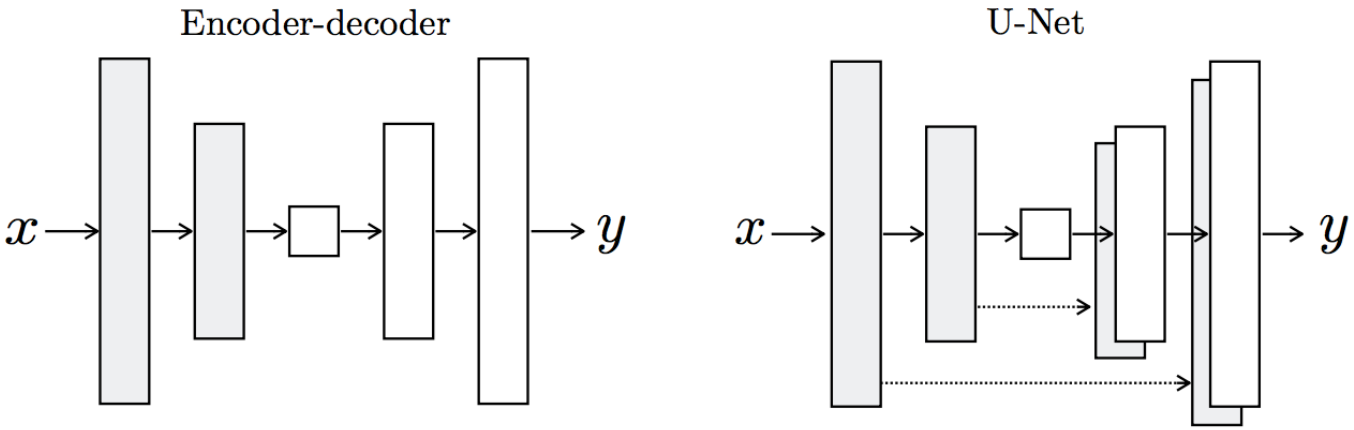


Figure source

Slide courtesy of Lana Lezebnik