Convolutional Neural Networks II

Figure from Convolutional Pose Machines, Wei et al. CVPR 2016

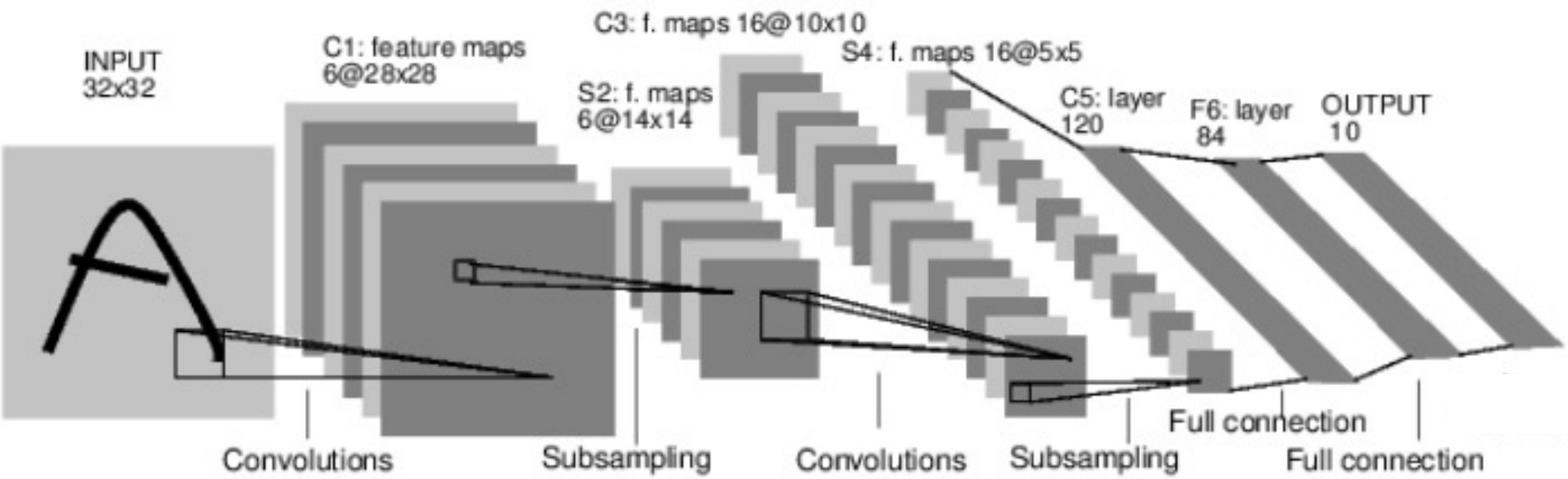
Input Image (a) Stage 1 (b) Stage 2 (c) Stage 3

CS194: Computer Vision and Comp. Photo
Angjoo Kanazawa/Alexei Efros, UC Berkeley, Fall 2021

Case Study: Lenet-5

Task: 10 digit classification

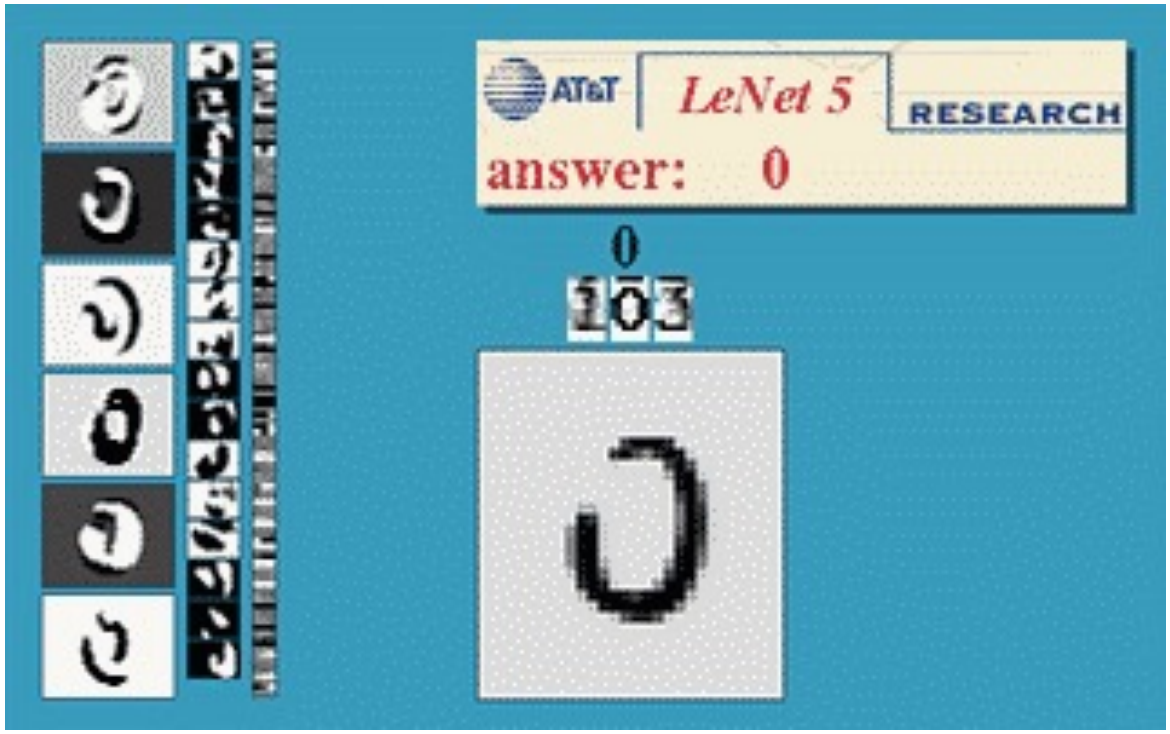
LeCun et al. 1998



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

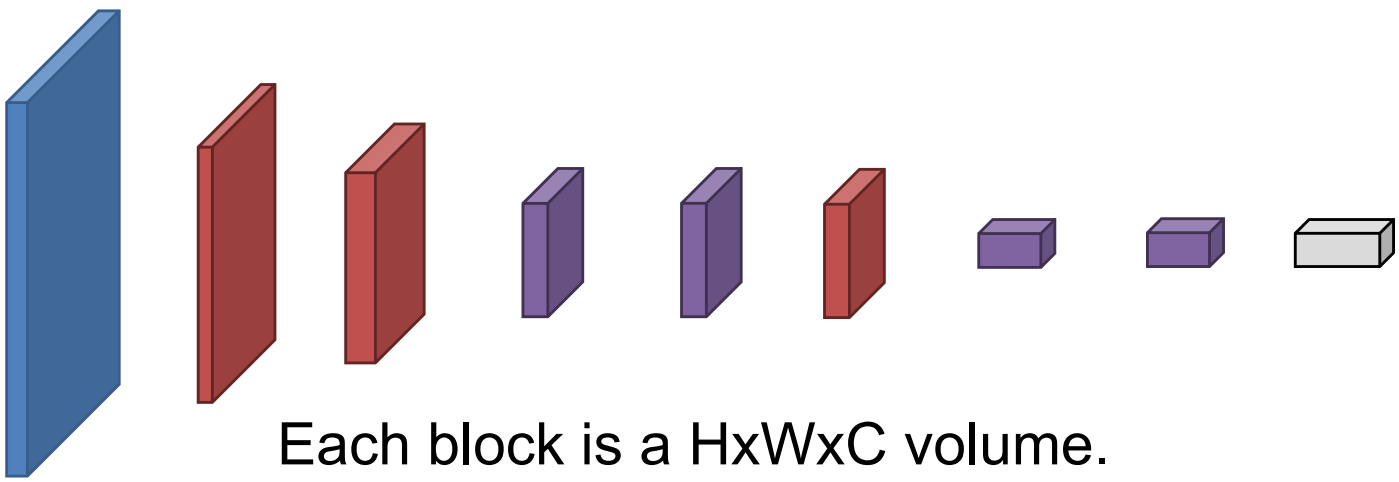


Case Study: AlexNet

[Krizhevsky, Sutskever, Hinton, NeurIPS 2012]

Task: ImageNet 1000-class classification

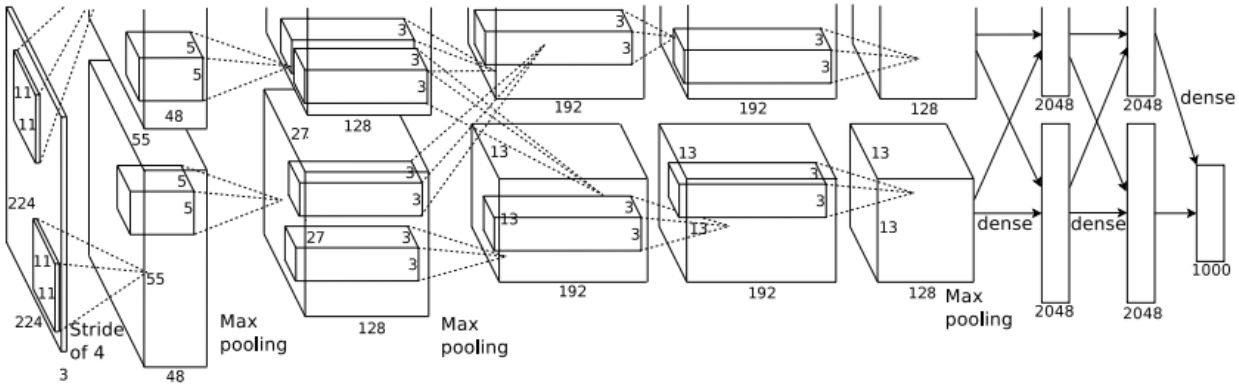
	Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
HxW	227x 227	55x55	27x27	13x13	13x13	13x13	1x1	1x1	1x1
C	3	96	256	384	384	256	4096	4096	1000



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: (N-K)/S + 1

(227-11)/4 + 1 = 55

Q: What is the output volume size after Conv1?

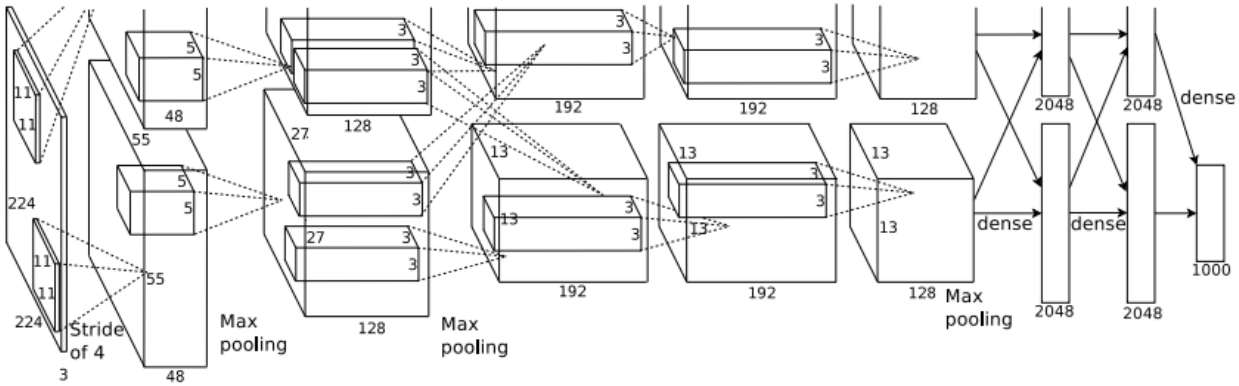
A: 55x55x96

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

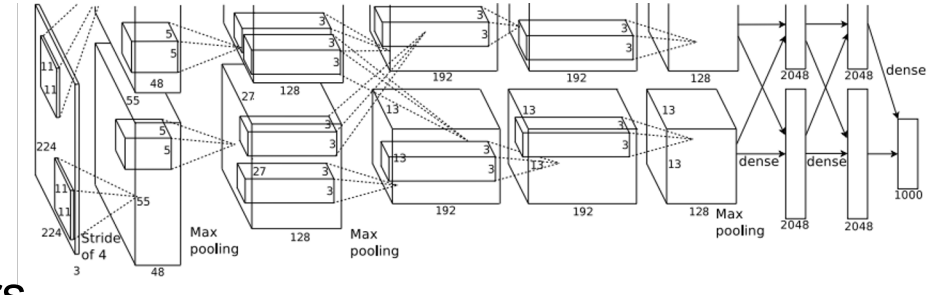
Q: What is the output volume size after Pool1?

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?

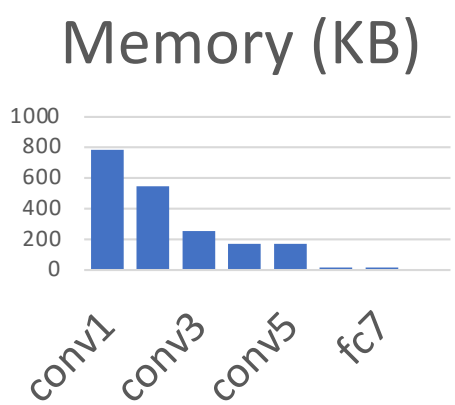
Most of the **memory usage** is in the early convolution layers

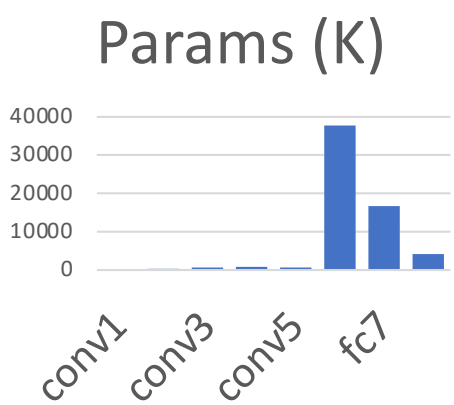
Large # of parameters?

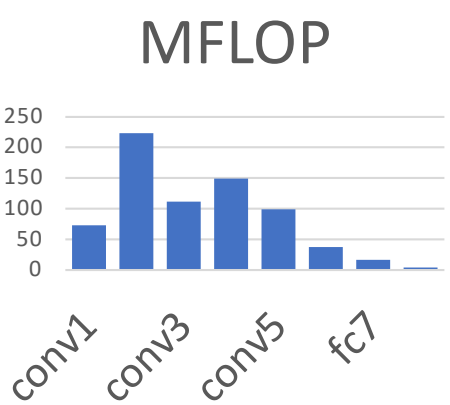
Nearly all **parameters** are in the fully-connected layers

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:

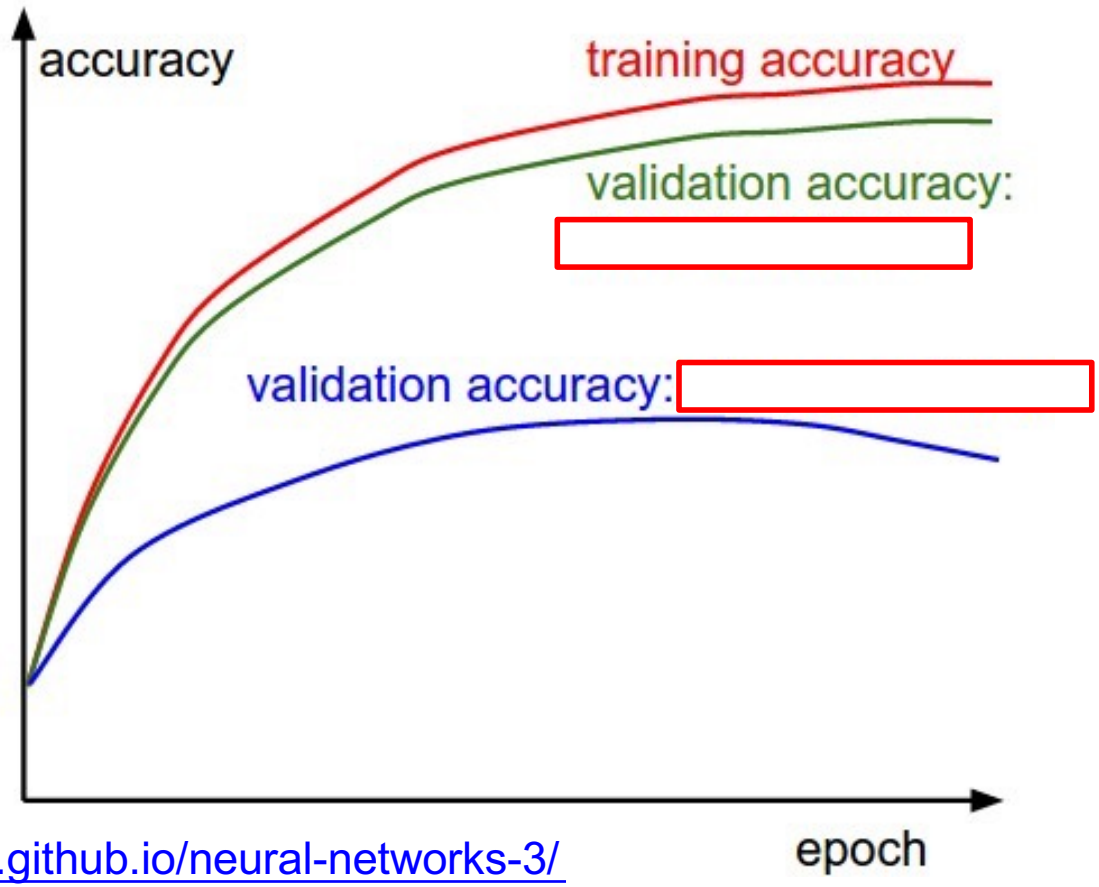
- **All kernel size 3x3**
- Always ReLu
- All max pool are 2x2, stride 2
- After pool, double the # of channels

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: $18C^2HW$

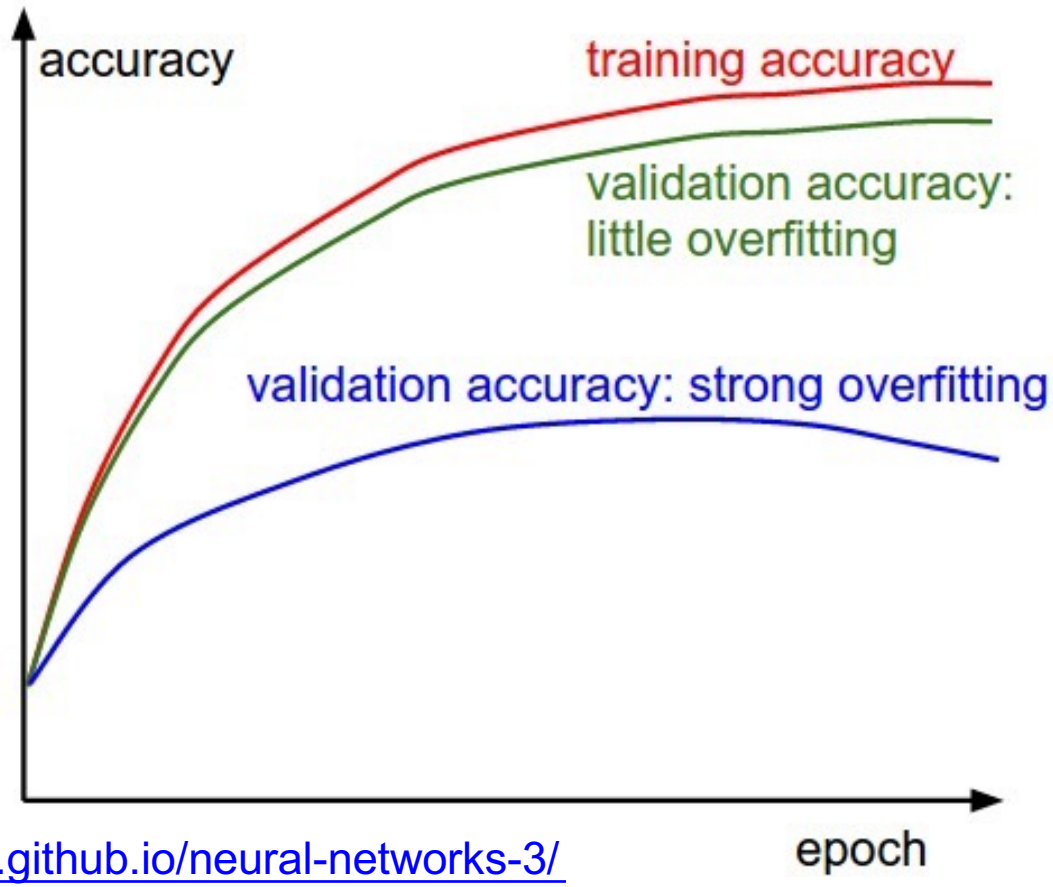
Slide adapted from Johnson & Fouhey

Measuring performance over train/val sets



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

Measuring performance over train/val sets



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

Naively adding more layers != better performance

Q: is this over fitting?

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

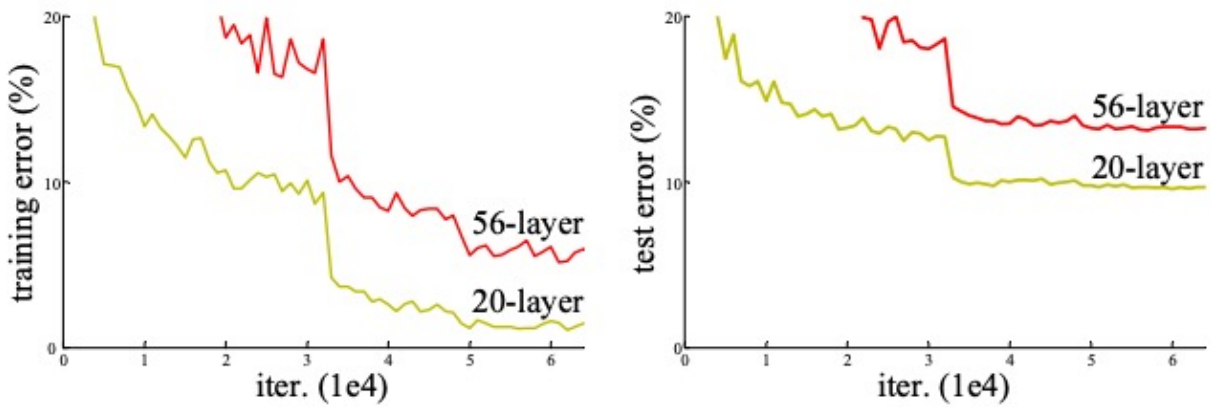
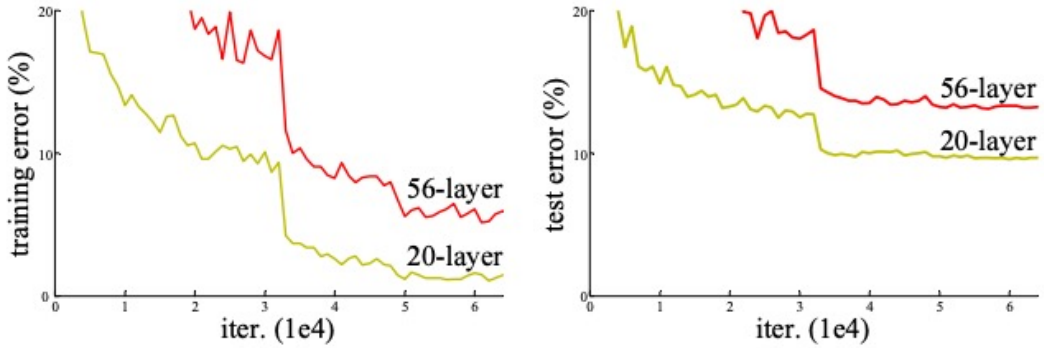


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

CVPR 2016]

Naively adding more layers != better performance



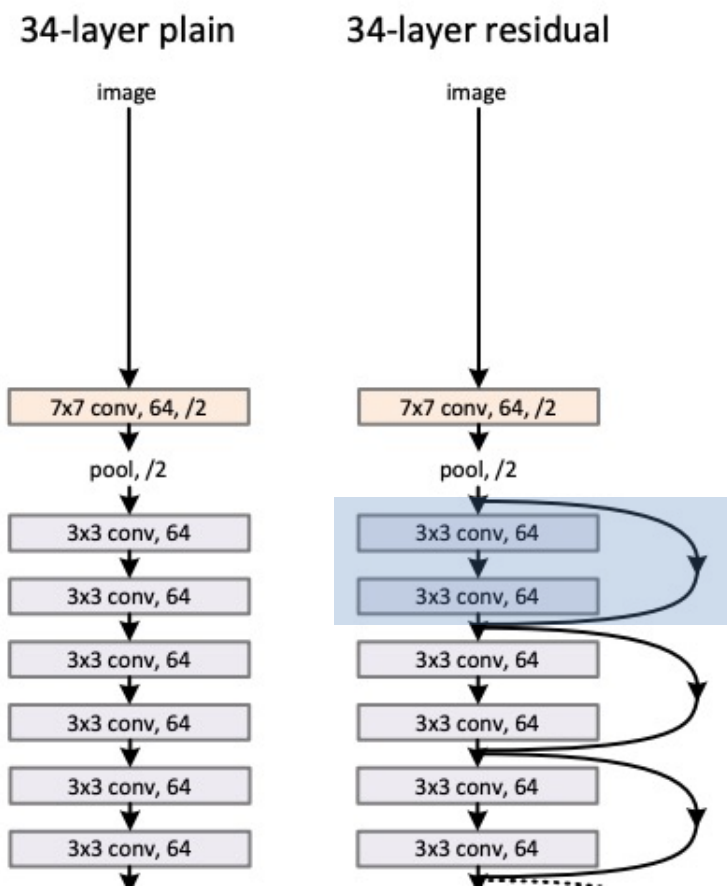
[He et al. CVPR 2016]

Deeper networks can 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]



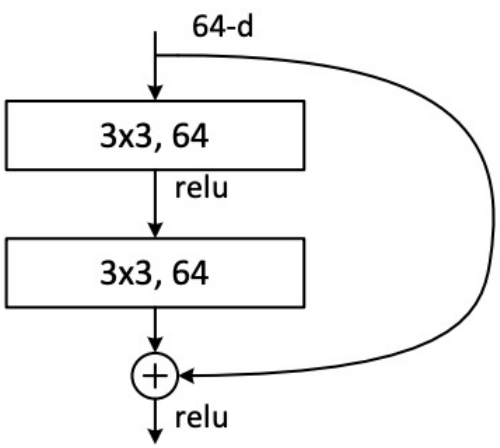
Input: 224x224x3

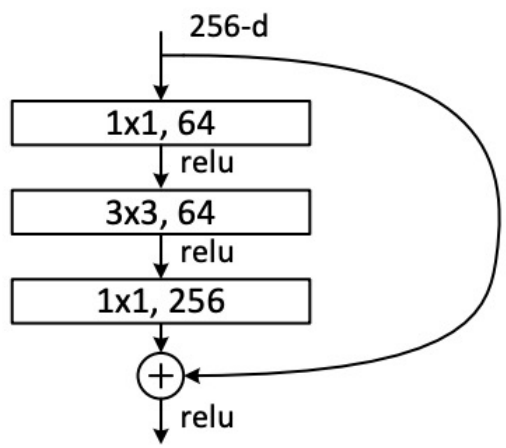
ResNet Block:

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

ResNet [He et al. CVPR 2016]

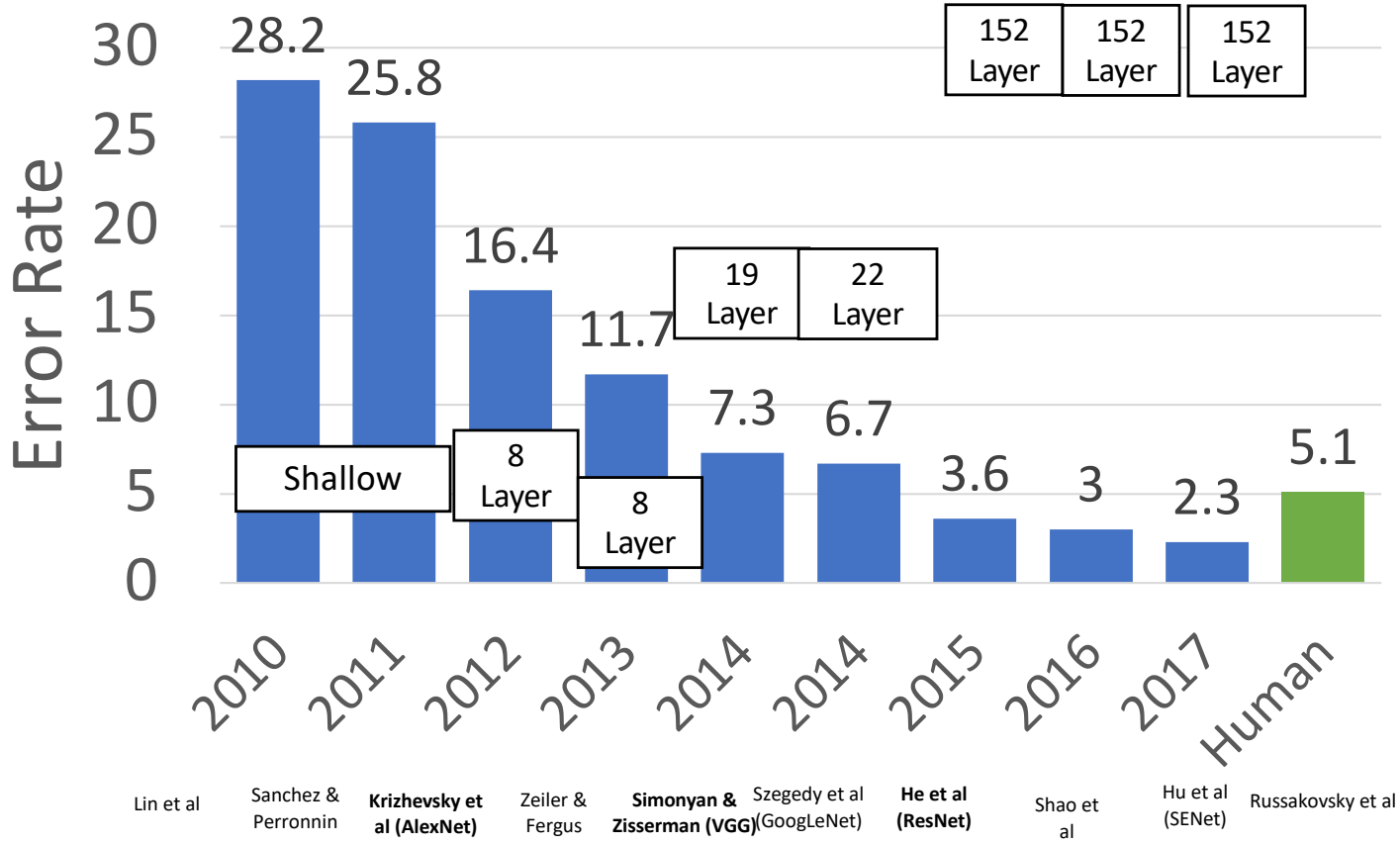
Can be made very deep: ResNet-50, 101, 152
And converge well





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

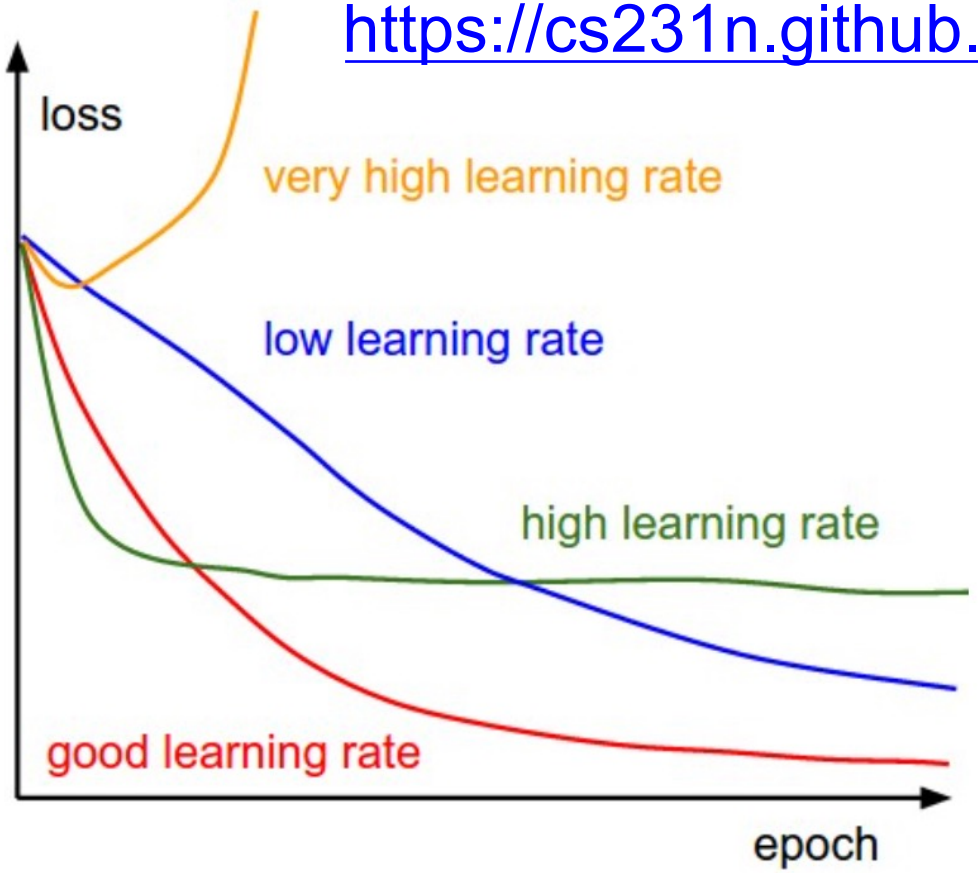
ImageNet Classification Challenge top-5 error

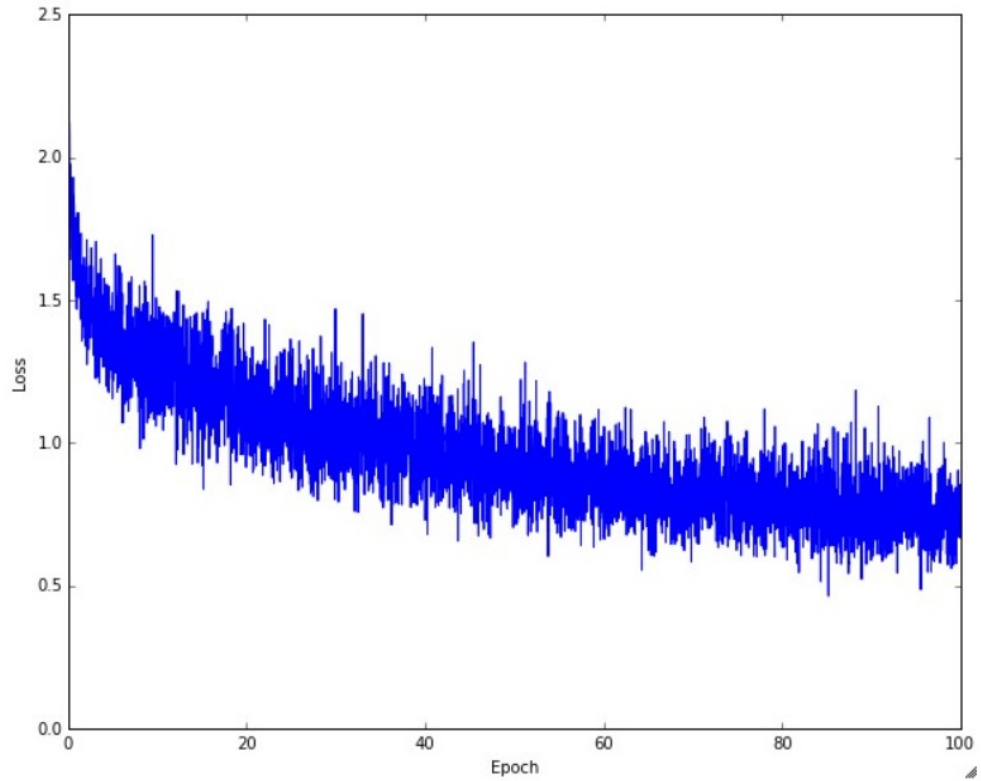


How to train your network

Training diagnosis

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





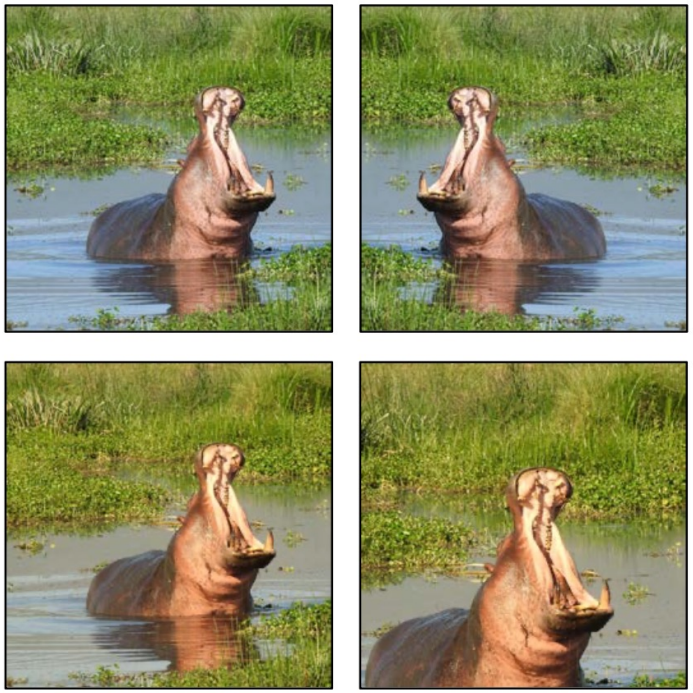
What if your data is not big enough?

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!!

Right eye Left eye

Defined wrt to the cat

Original

After flip + color aug

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

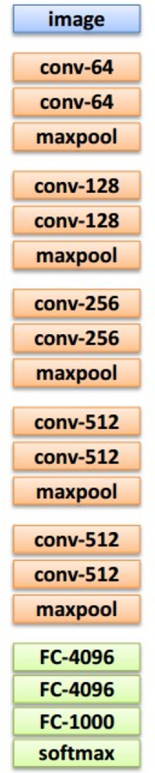
“You need a lot of a data if you want to train/use CNNs”

Transfer Learning

"You need a lot of a data if you want to train/use CNNs"

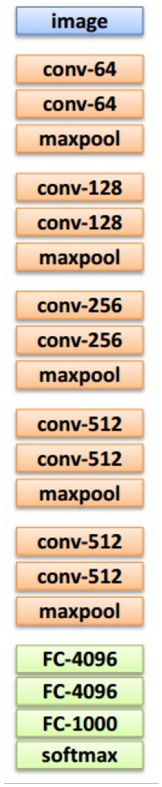
NOT ALWAYS

Transfer Learning with CNNs

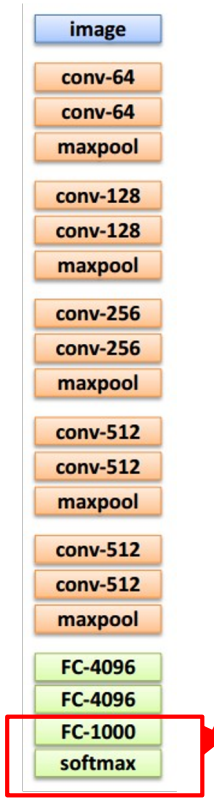


1. Train on Imagenet

Transfer Learning with CNNs



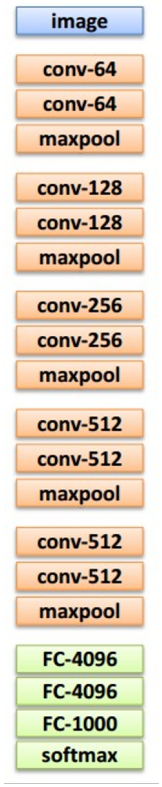
1. Train on Imagenet



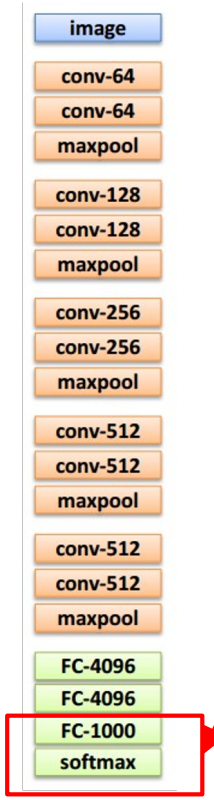
2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

i.e. swap the Softmax layer at the end

Transfer Learning with CNNs



1. Train on Imagenet



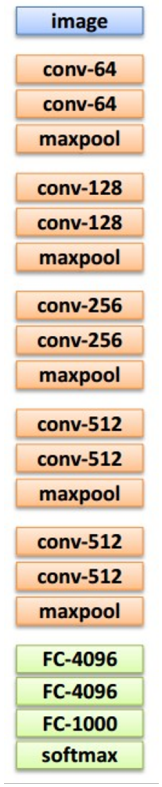
2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

i.e. swap the Softmax layer at the end

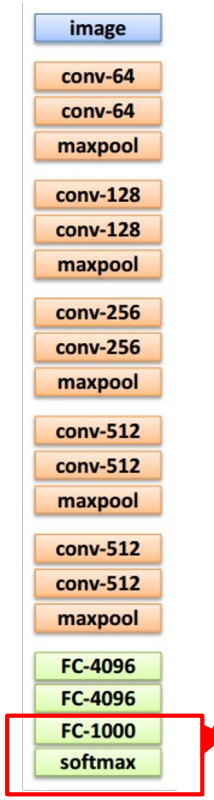
3. If you have medium sized dataset, **"finetune"** instead: use the old weights as initialization, train the full network or only some of the higher layers

retrain bigger portion of the network, or even all of it.

Transfer Learning with CNNs

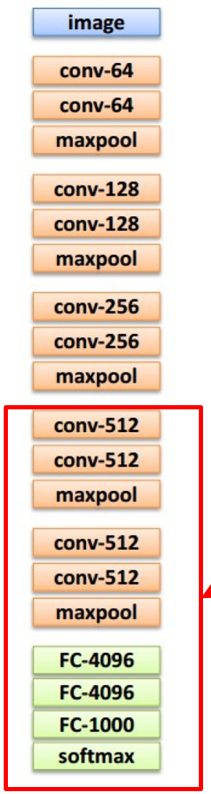


1. Train on Imagenet



2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

i.e. swap the Softmax layer at the end

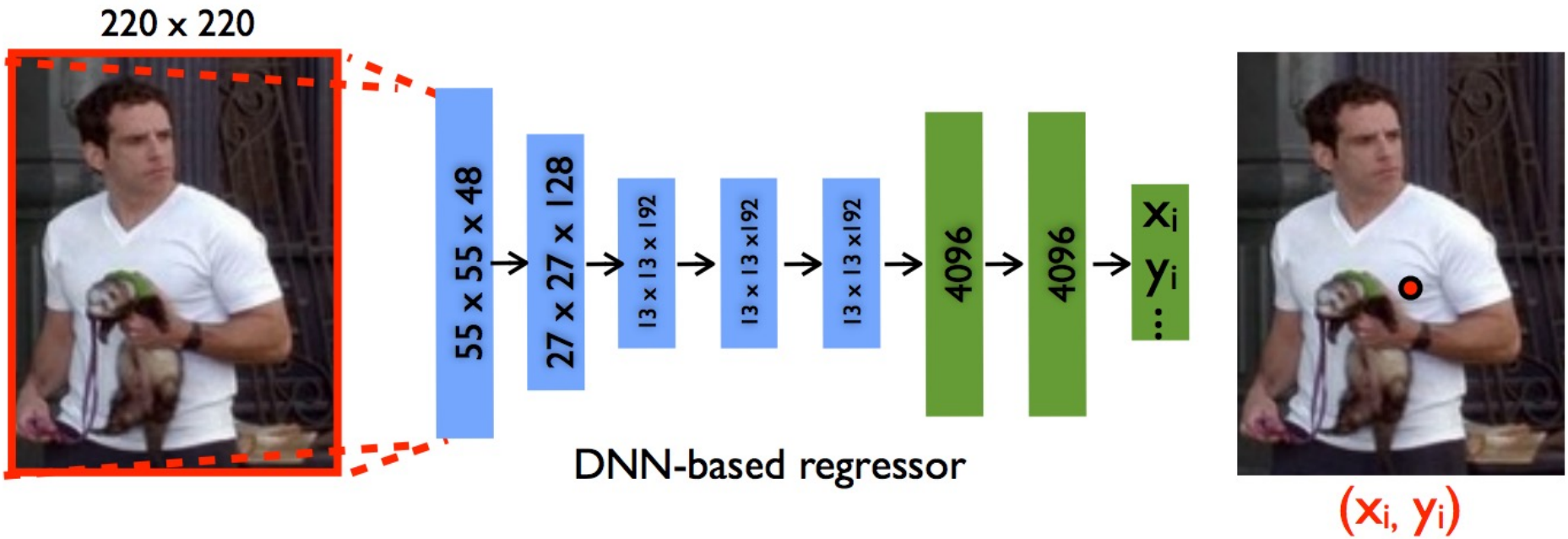


3. If you have medium sized dataset, **"finetune"** instead: use the old weights as initialization, train the full network or only some of the higher layers

retrain bigger portion of the network, or even all of it.

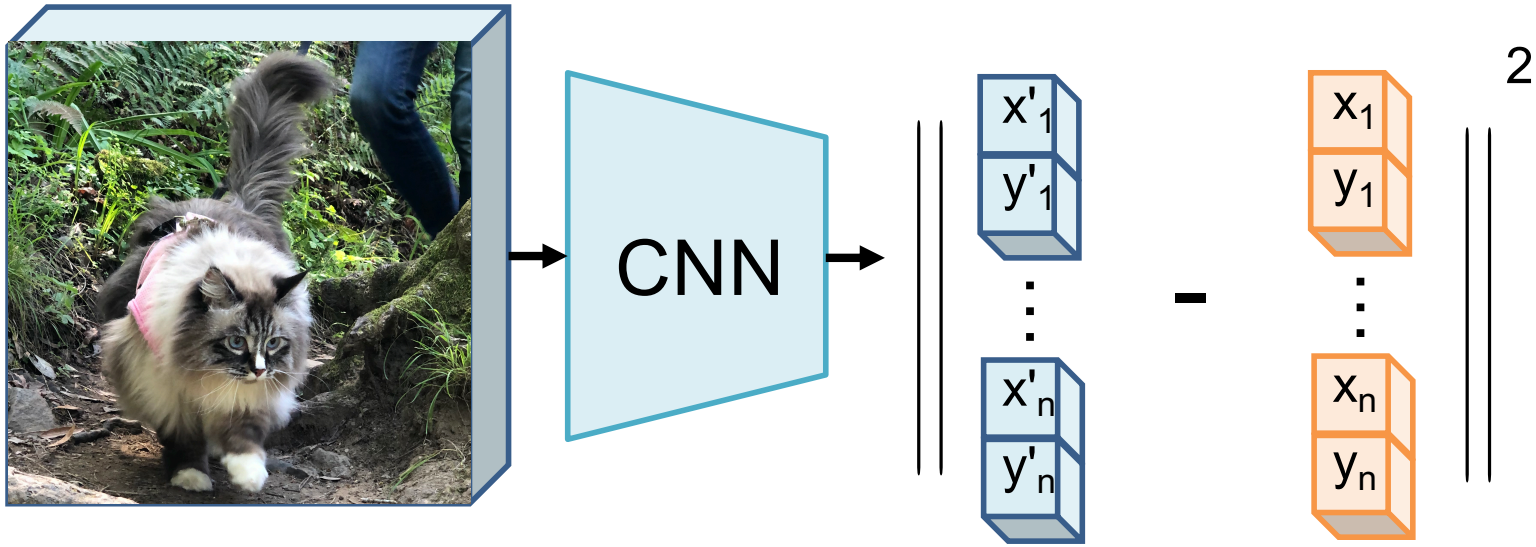
tip: use only ~1/10th of the original learning rate in finetuning to player, and ~1/100th on intermediate layers

Project 5 starting point (default)



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

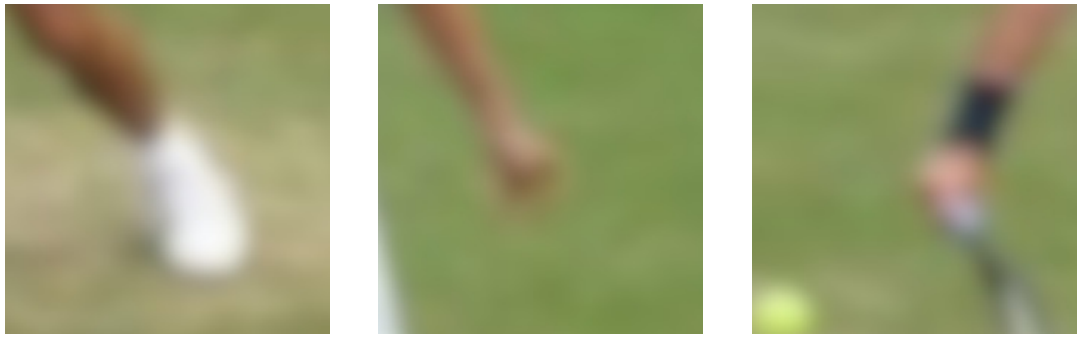
Regression Objective



This is what you should do before bells & whistles.
But this is not what the SoTA models do in practice

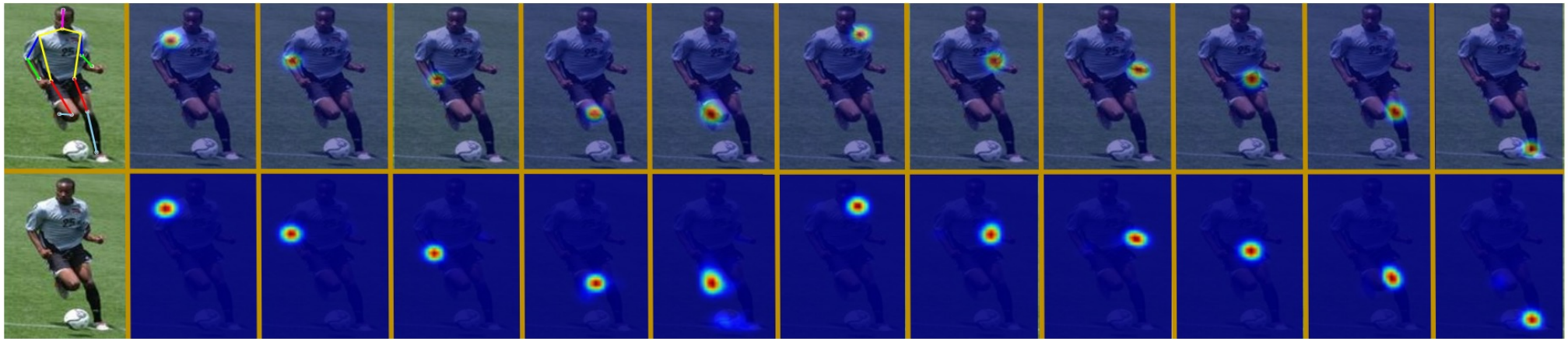
Downsides of regression objective

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.

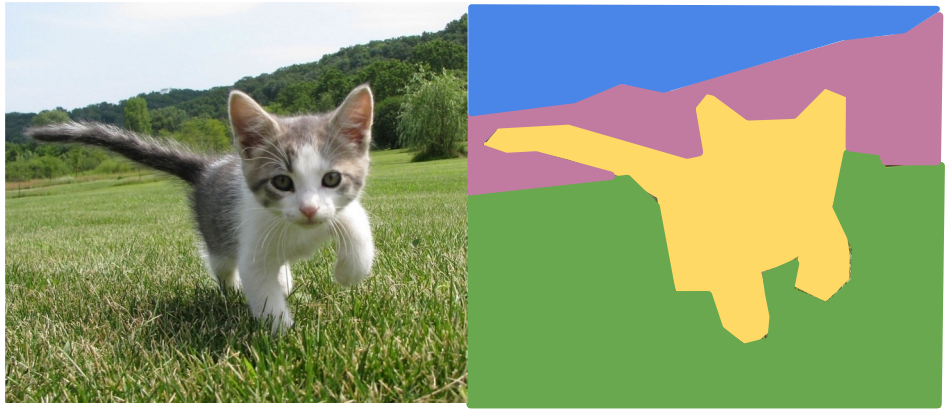
How do we do dense (per-pixel) predictions?

Dense Prediction

Needed for many things:

Image Synthesis!
(next lecture)

Semantic Segmentation

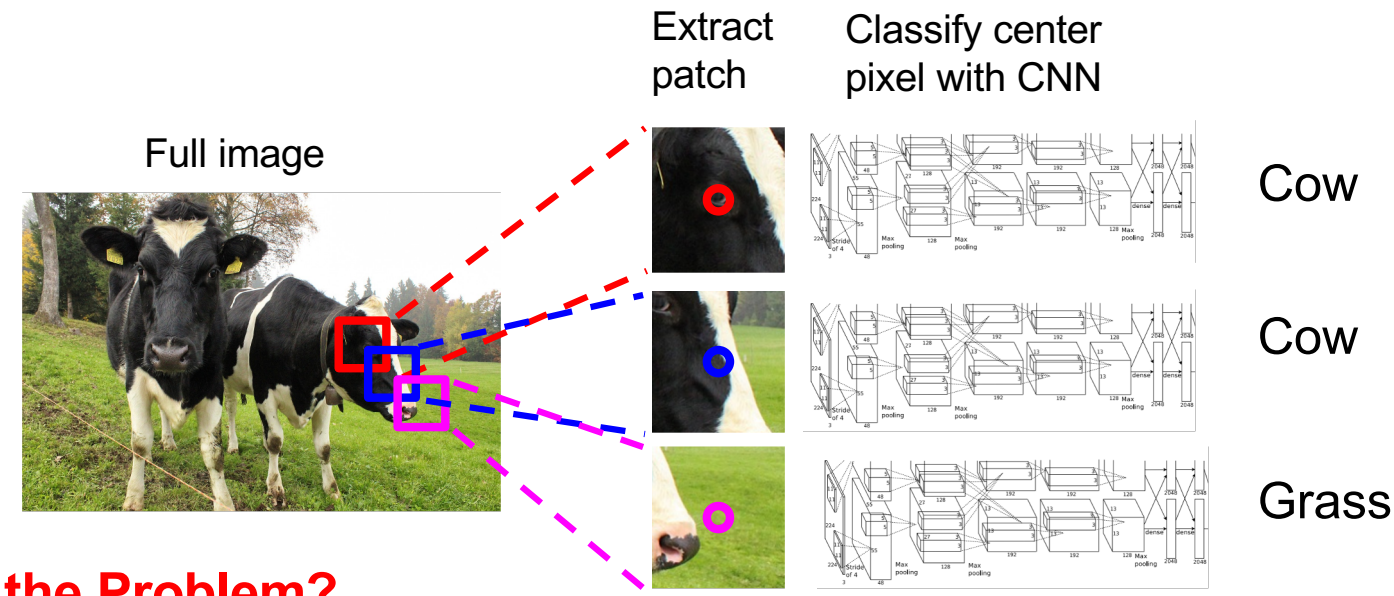


GRASS, **CAT**, **TREE**, **SKY**

Keypoint detection

Does this pixel contain right wrist?

Idea 1: Dense prediction by Sliding Window



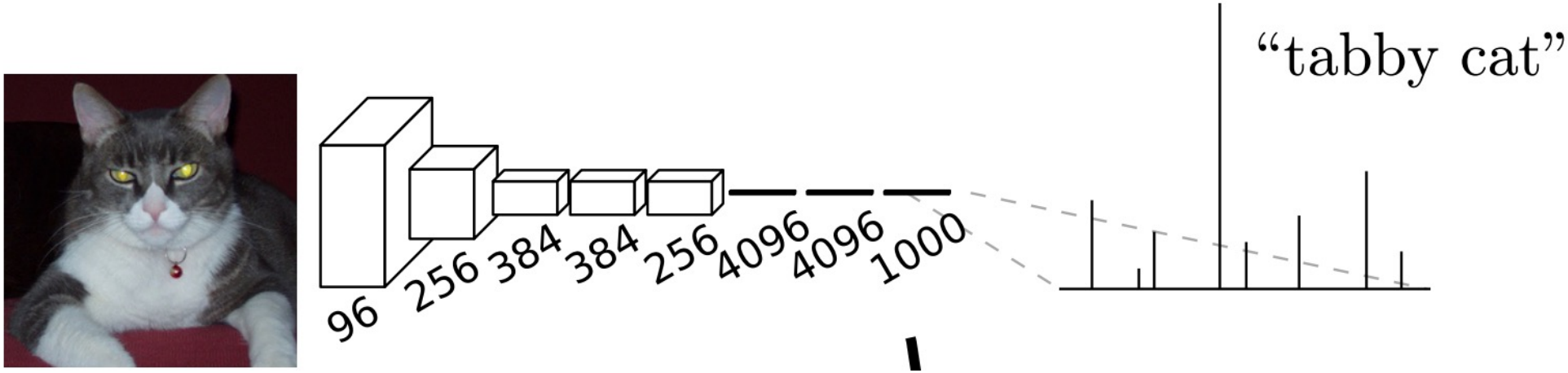
What is the Problem?

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!

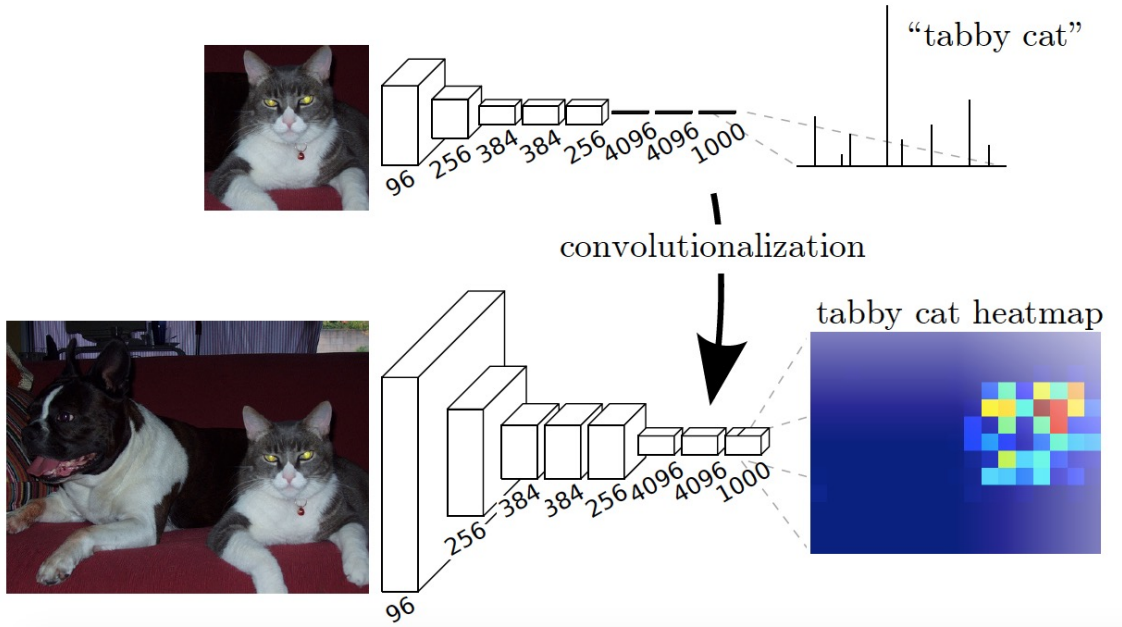
Think of the last few fully connected network as a convolutions with kernels (receptive field) that cover the entire image ➔ can turn it into 1x1 convs



Fully convolutional networks

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

Convert all FC into 1x1 convolutions

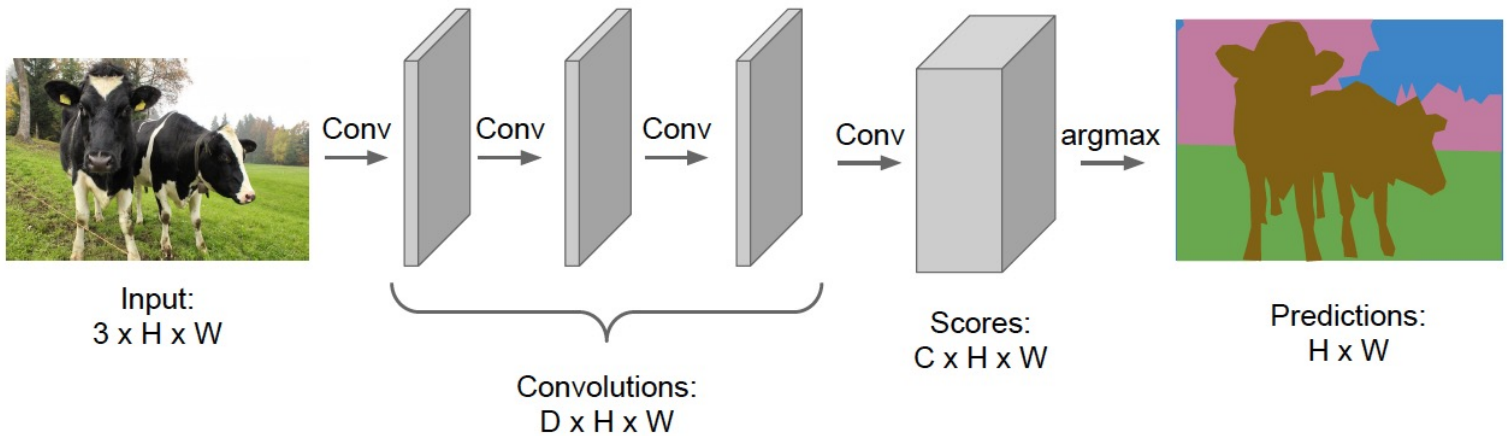


J. Long, E. Shelhamer, and T. Darrell, Fully Convolutional Networks for Semantic Segmentation, CVPR 2015

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

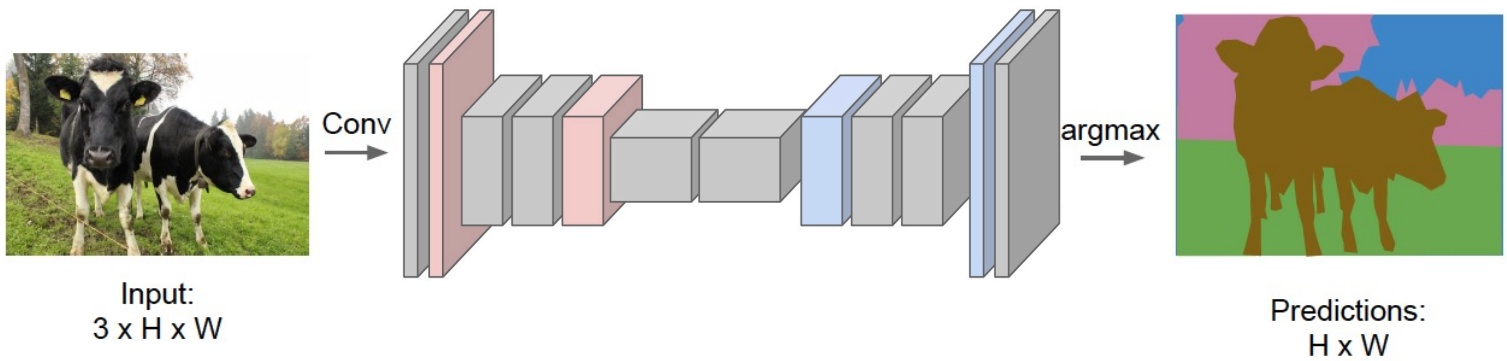


Slide courtesy of Lana Lezebnik

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

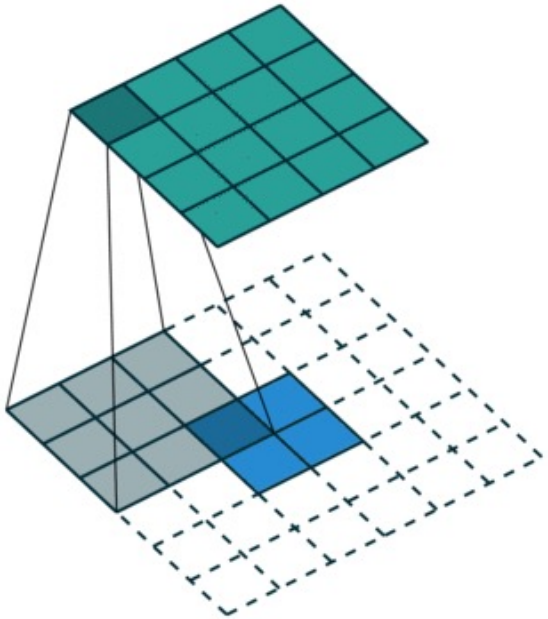
HOW TO UPSAMPLE WITH CONVNETS?

There are a few ways

- I will begin by the first thing people tried
- Then a much simpler, natural solution that’s more used in practice

Upsampling in a deep network

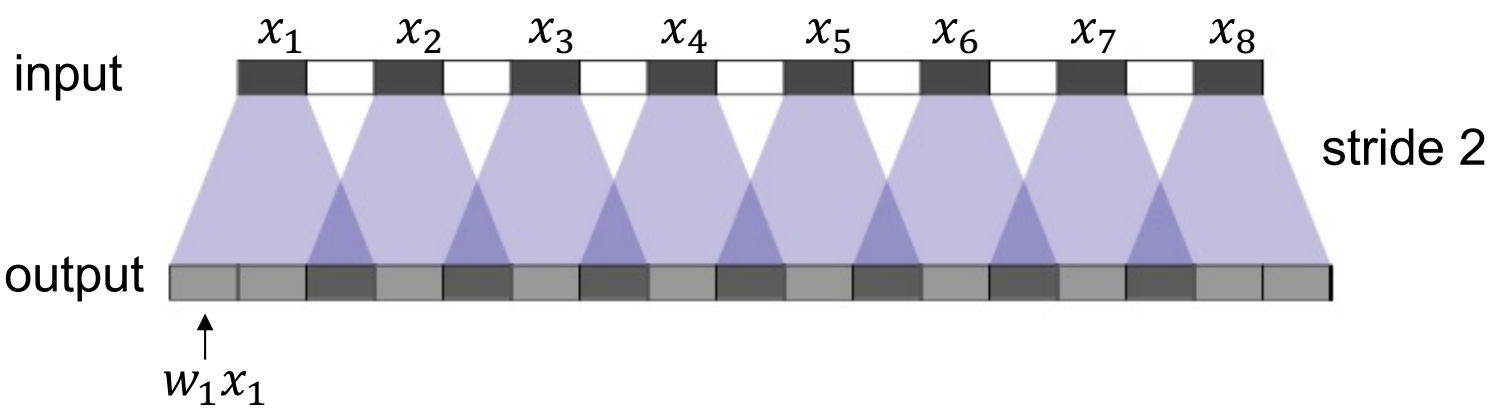
- *Transposed* or *backwards-strided* convolution: “paint” in the output feature map with the learned filter
  - Multiply input value by filter, place result in the output, sum overlapping value



Animation: https://github.com/vdumoulin/conv_arithmetic

Upsampling in a deep network

- Filter $[w_1, w_2, w_3]$, *output stride* = 2

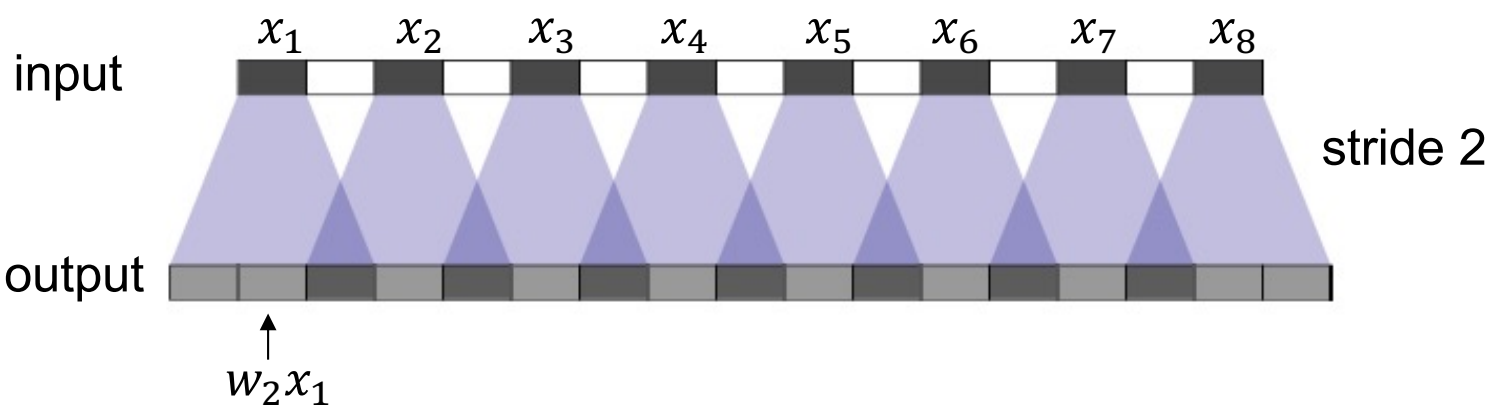


Animation: https://distill.pub/2016/deconv-checkerboard/

Slide courtesy of Lana Lezebnik

Upsampling in a deep network

- Filter $[w_1, w_2, w_3]$, *output stride* = 2

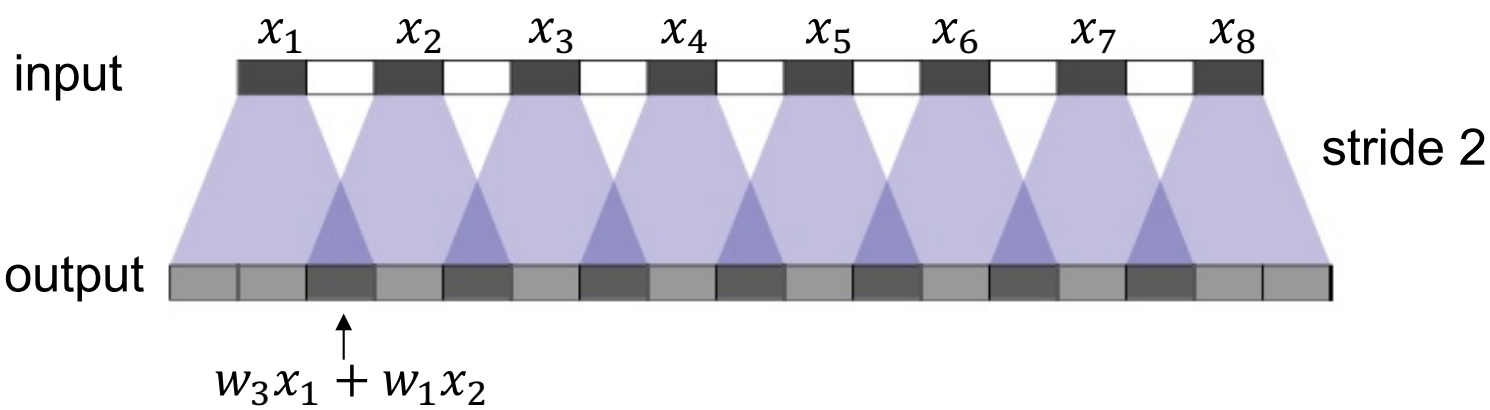


Animation: https://distill.pub/2016/deconv-checkerboard/

Slide courtesy of Lana Lezebnik

Upsampling in a deep network

- Filter $[w_1, w_2, w_3]$, *output stride* = 2

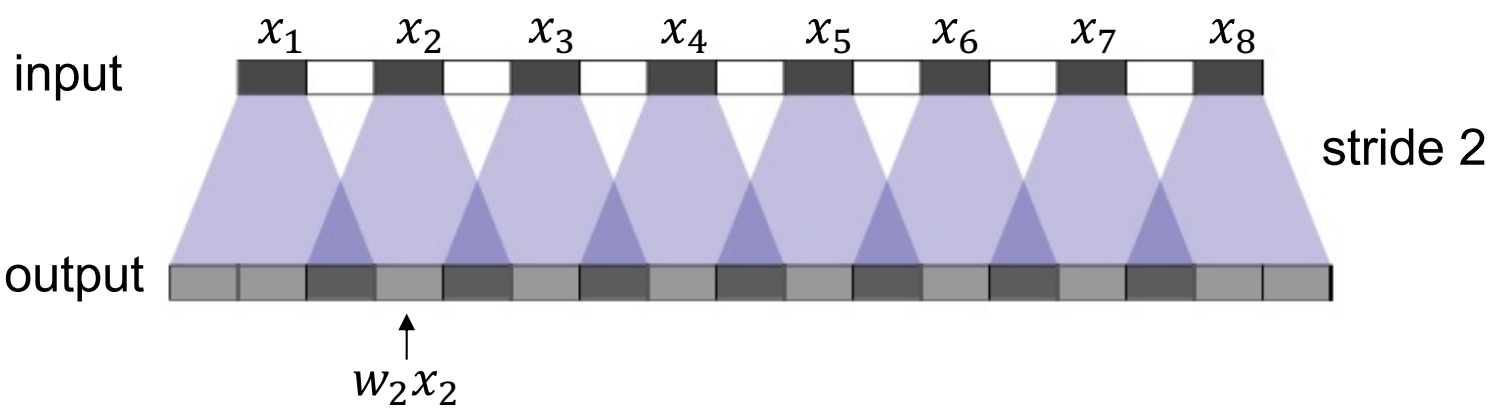


Animation: https://distill.pub/2016/deconv-checkerboard/

Slide courtesy of Lana Lezebnik

Upsampling in a deep network

- Filter $[w_1, w_2, w_3]$, *output stride* = 2

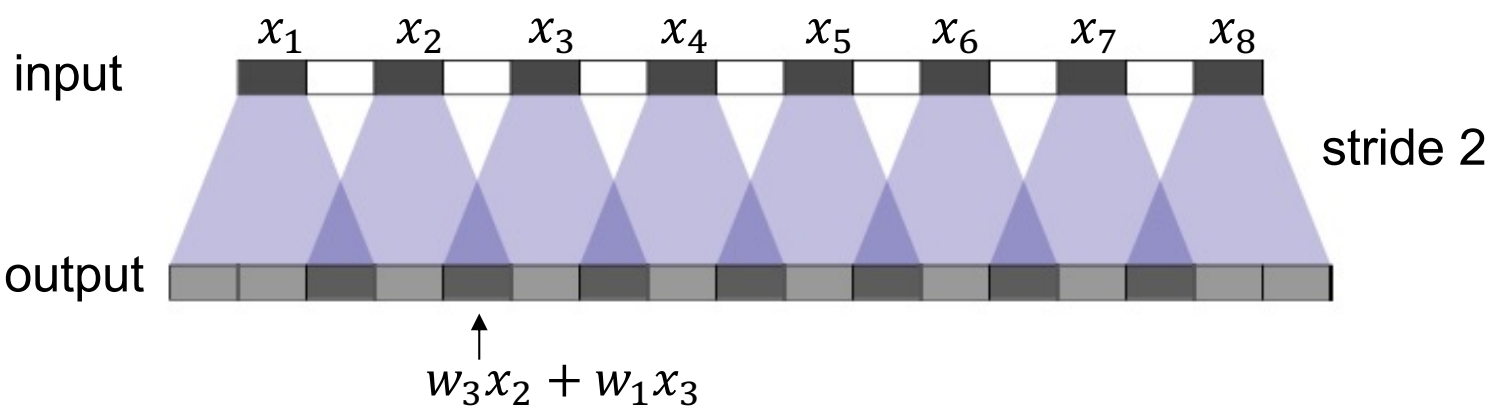


Animation: https://distill.pub/2016/deconv-checkerboard/

Slide courtesy of Lana Lezebnik

Upsampling in a deep network

- Filter $[w_1, w_2, w_3]$, *output stride* = 2



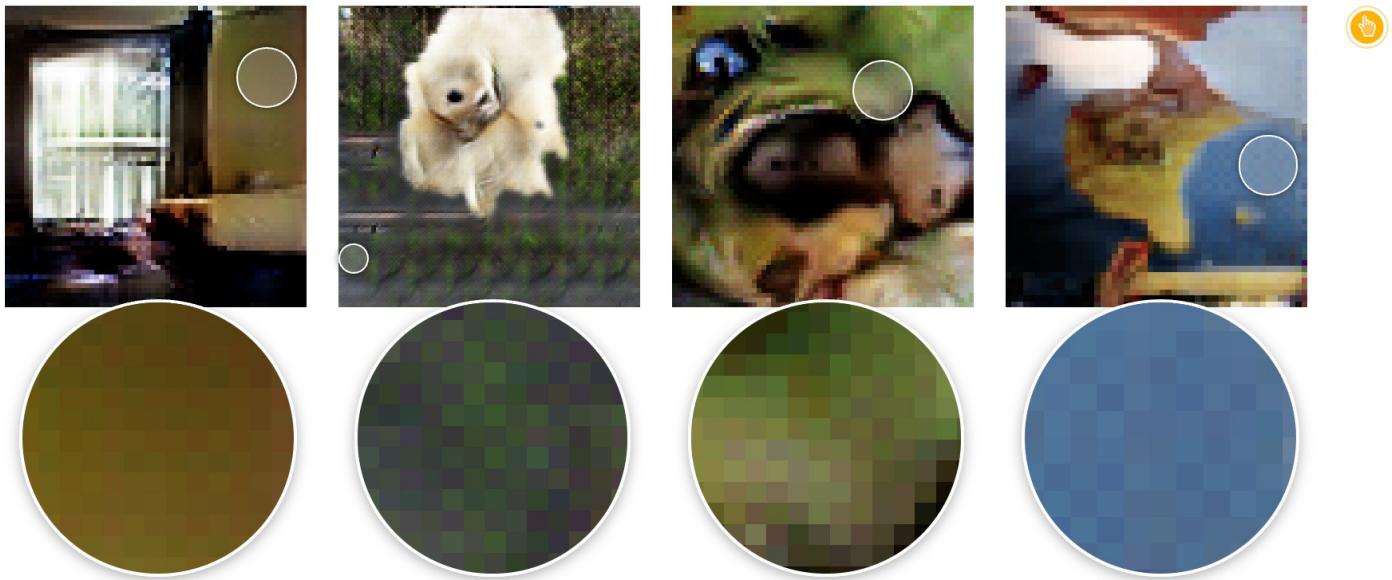
Animation: https://distill.pub/2016/deconv-checkerboard/

Slide courtesy of Lana Lezebnik

But this is generates checkerboard like artifacts!

Because of overlap.

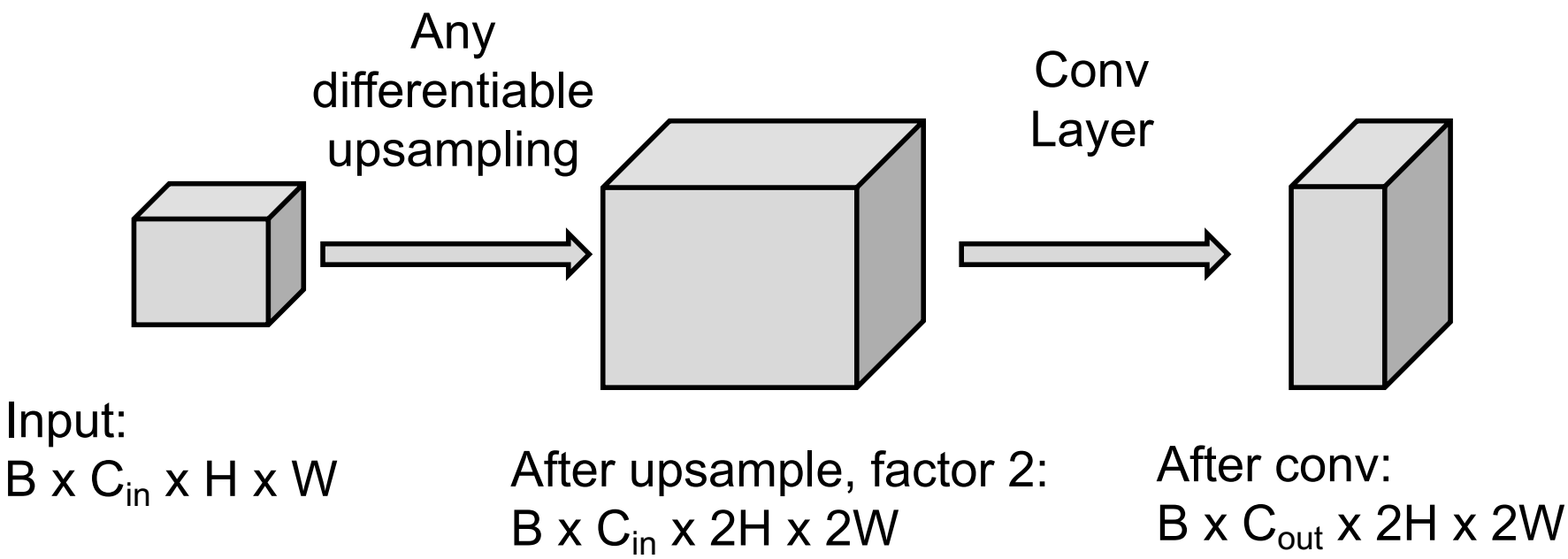
You can try to prevent overlap (play around with the distill page), but it gets complicated with multiple layers



Radford, et al., 2015 [1] Salimans et al., 2016 [2] Donahue, et al., 2016 [3] Dumoulin, et al., 2016 [4]

Simpler, effective solution (often used in practice)

- Upsample, followed by a regular Convolution



How to Upsample

Bed of Nails

1	2
3	4

Input
C x 2 x 2

→

1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

Output
C x 4 x 4

Nearest Neighbor

1	2
3	4

Input
C x 2 x 2

→

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Output
C x 4 x 4

Bilinear Interpolation

1	2
3	4

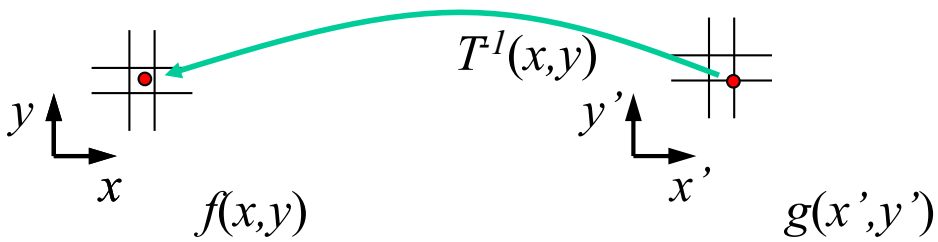
Input
C x 2 x 2

→

1.00	1.25	1.75	2.00
1.50	1.75	2.25	2.50
2.50	2.75	3.25	3.50
3.00	3.25	3.75	4.00

Output
C x 4 x 4

Recall from Morphing Lecture: Inverse warping



Get each pixel $g(x',y')$ from its corresponding location

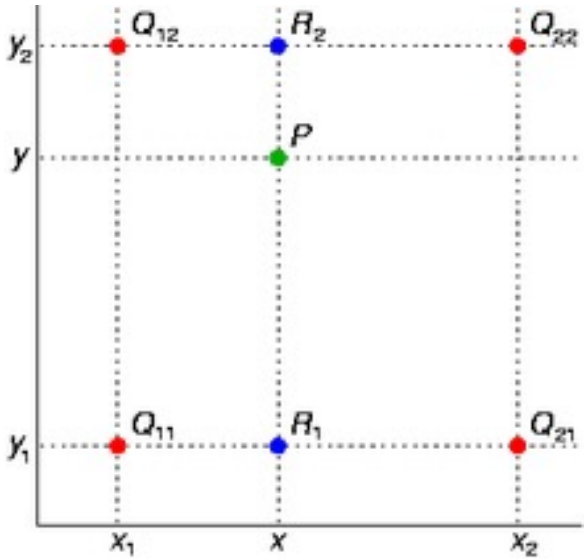
$(x,y) = T^{-1}(x',y')$ in the first image

Q: what if pixel comes from "between" two pixels?

A: *Interpolate* color value from neighbors

- nearest neighbor, bilinear, Gaussian, bicubic
- Check out `interp2` in Matlab / Python

Recall: Bilinear Interpolation



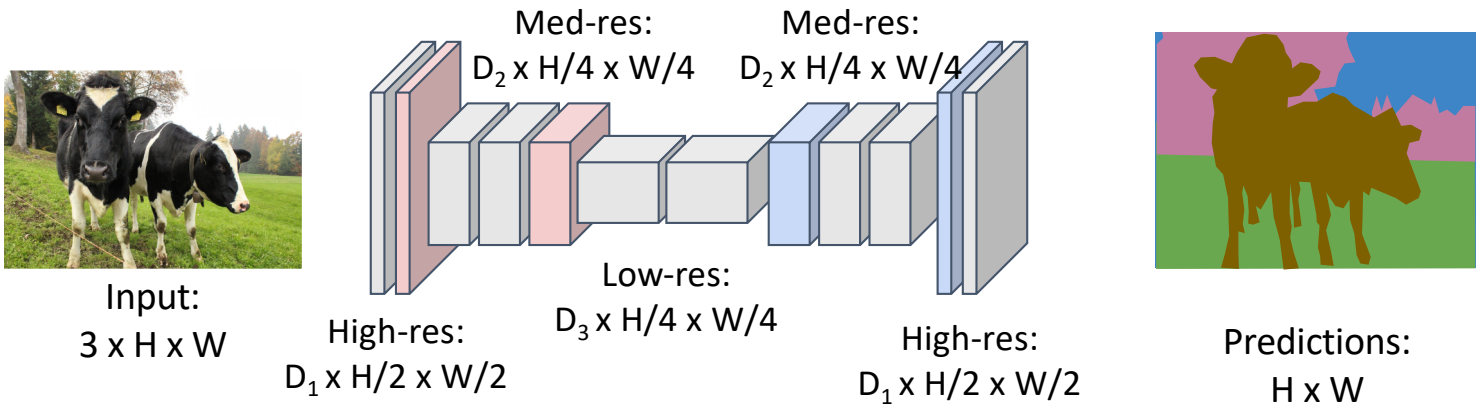
http://en.wikipedia.org/wiki/Bilinear_interpolation

```
Help interp2
```

Fully Convolutional Network

Rule of thumb: every upsample is a factor of 2

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

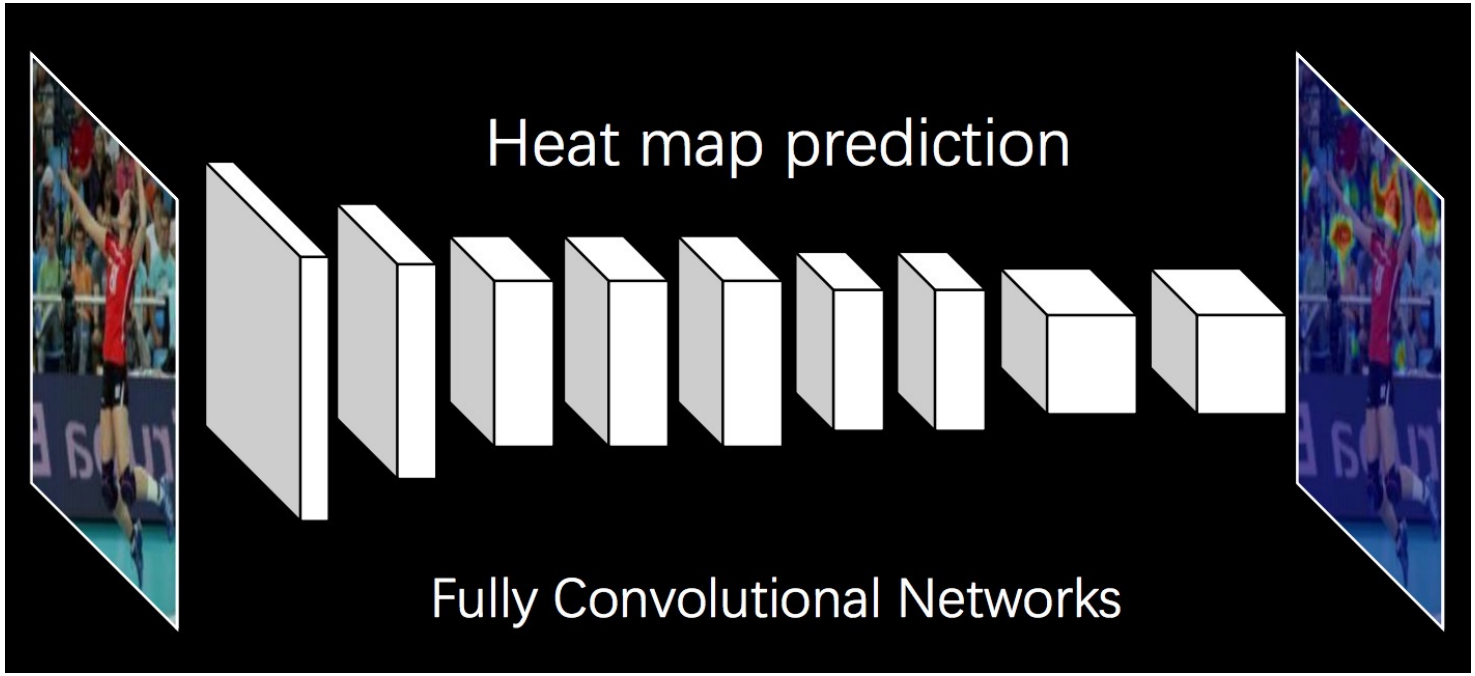


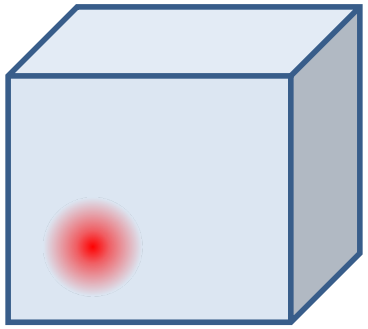
Downsampling:
Pooling, strided convolution

Upsampling:
Various choices

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Application to pose detection: Predict heat maps



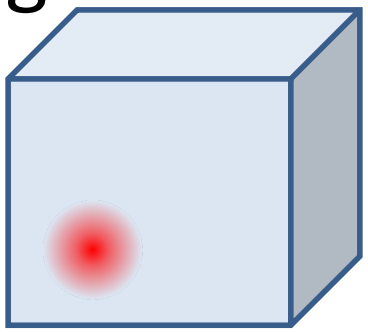


Target: K+1 x H x W Gaussian around (x,y) for k-th keypoint in the k-th channel

K+1 for K parts + background

Training Loss

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



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

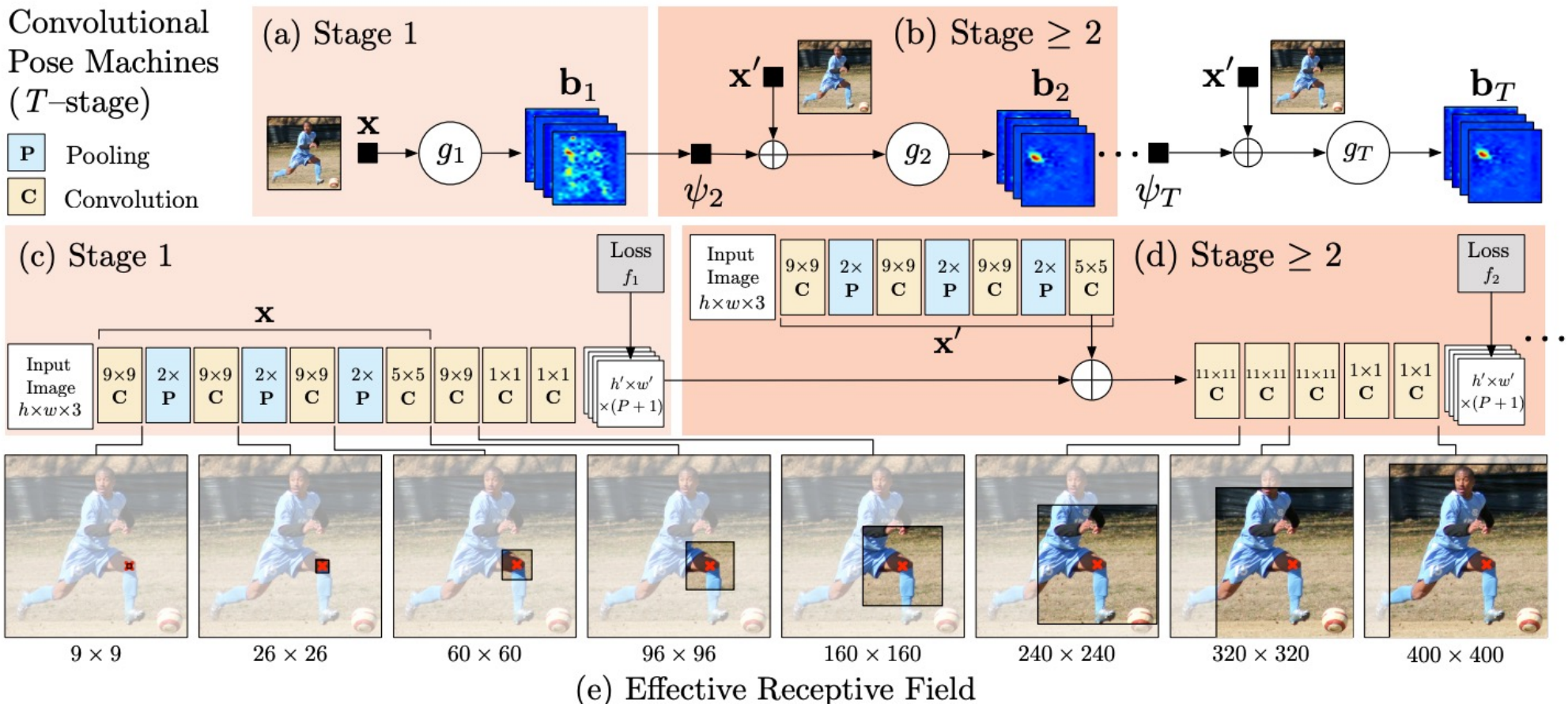
Target "belief map" : K+1 x H x W Gaussian around (x,y) for k-th keypoint in the k-th channel

- Mask RCNN applies softmax across a heatmap and trains with cross-entropy

[Wei et al CVPR 2016]

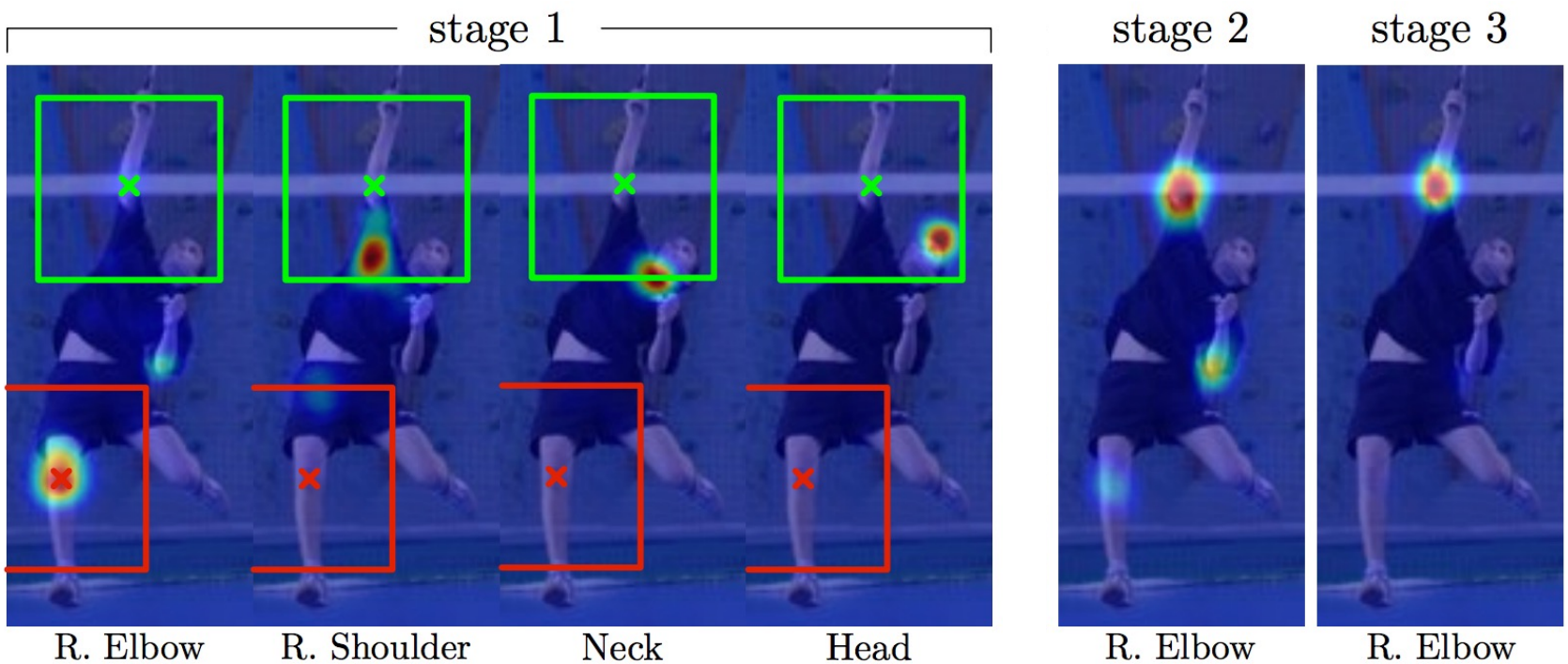
Convolutional Pose Machines

Base architecture for OpenPose



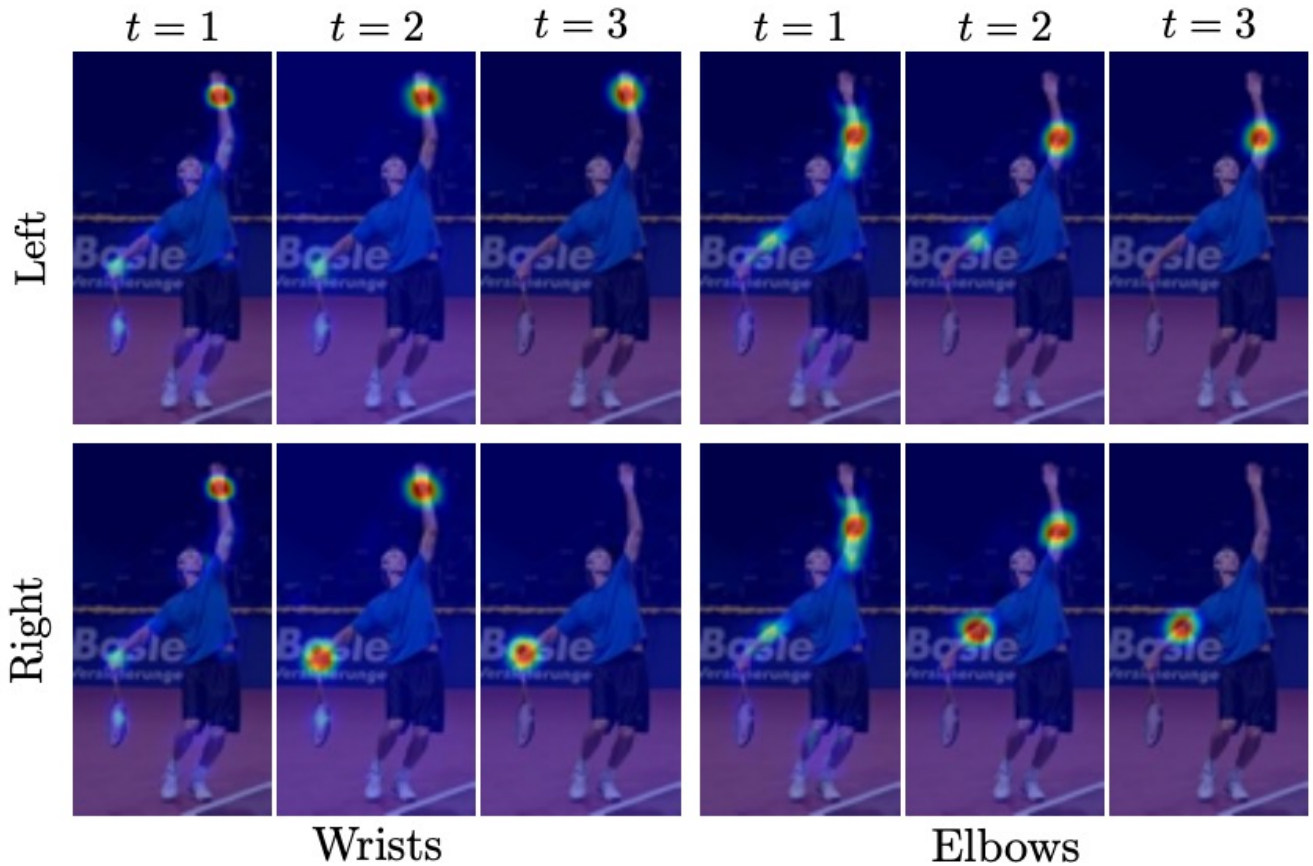
[Wei et al CVPR 201

Convolutional Pose Machines

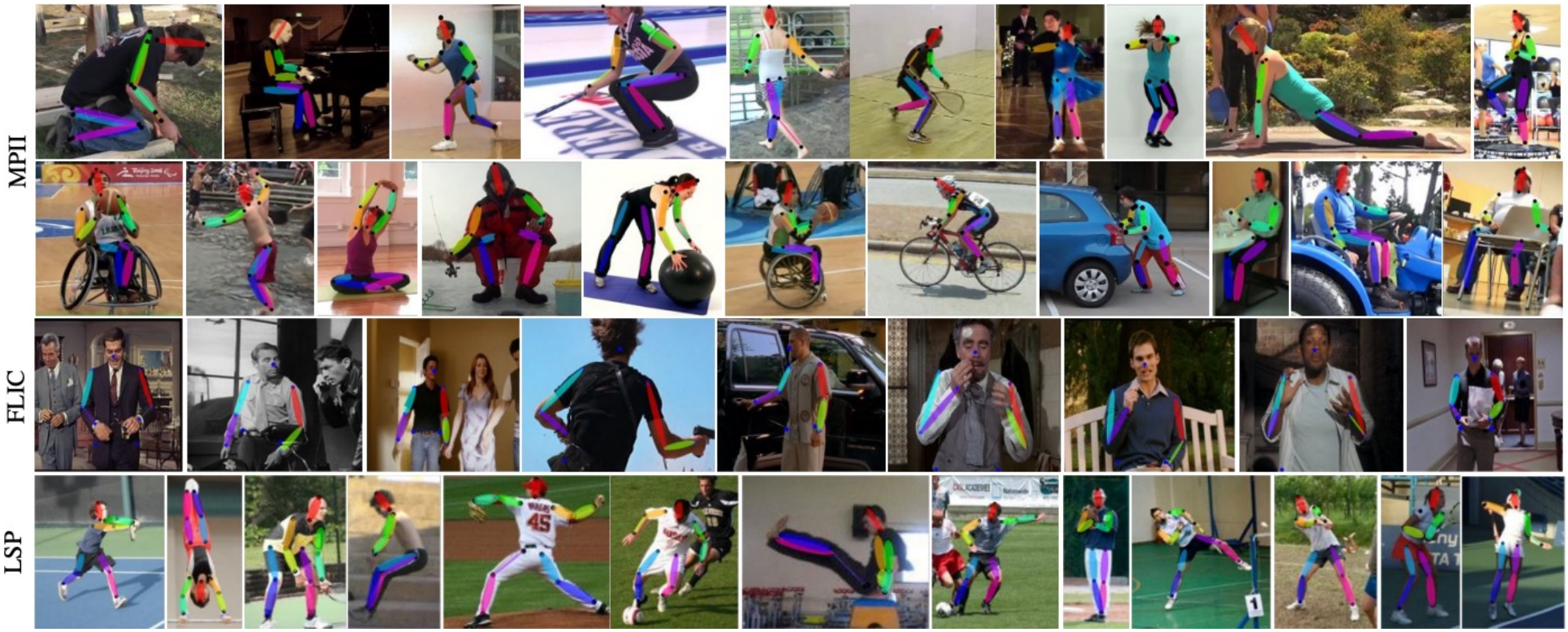


[Wei et al CVPR 2016]

Convolutional Pose Machines



Results

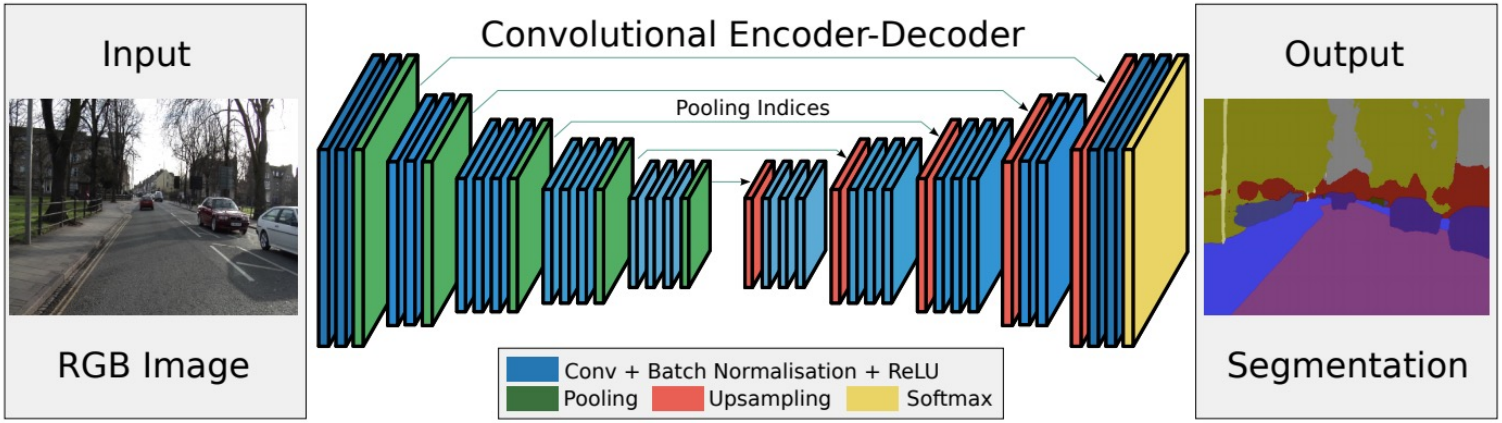


OpenPose

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

Examples for segmentation: SegNet

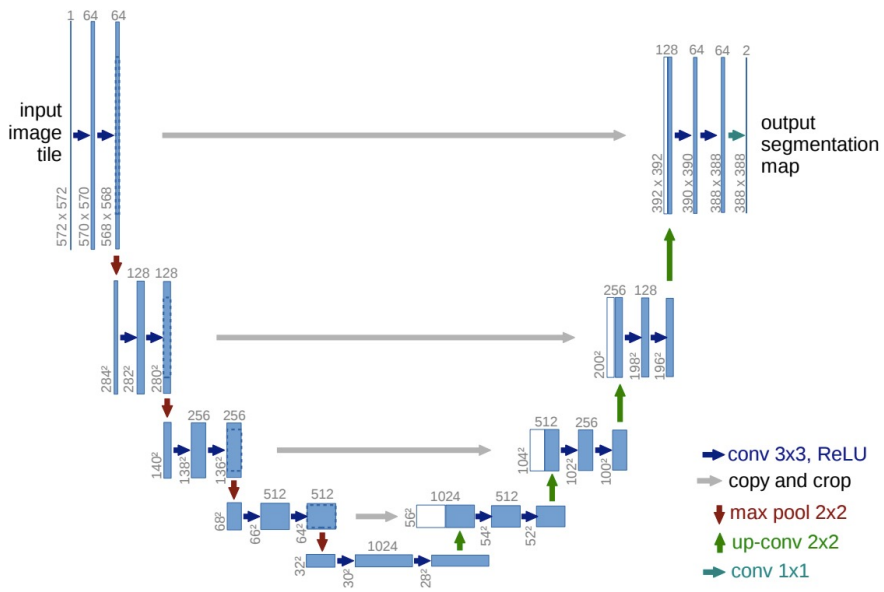


Drop the FC layers,
get better results

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

Other upsampling networks: U-Net

- Like FCN, fuse upsampled higher-level feature maps with higher-res, lower-level feature maps
- Unlike FCN, 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

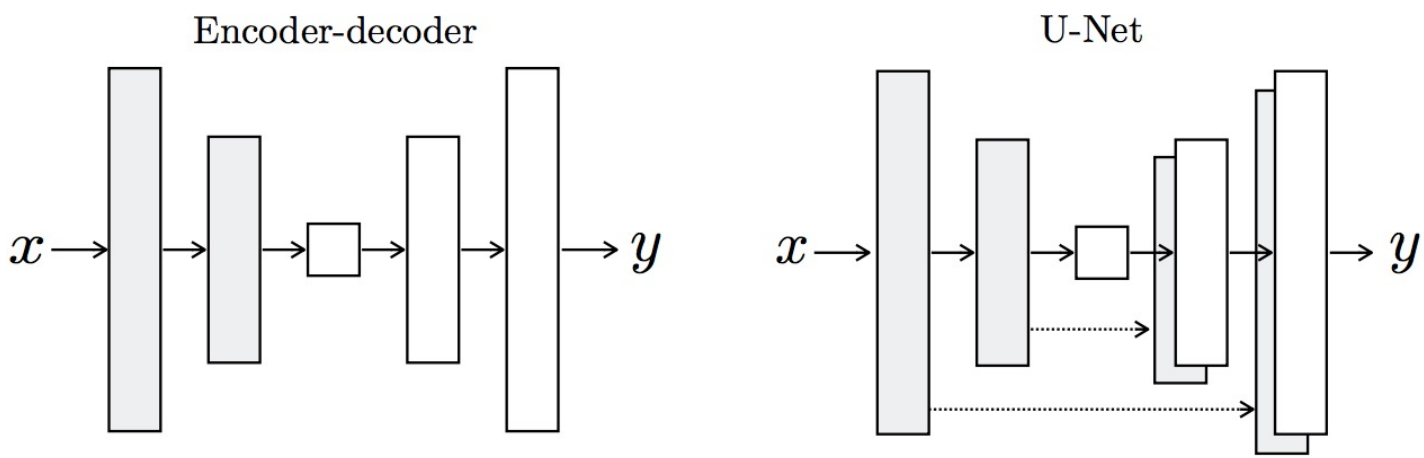


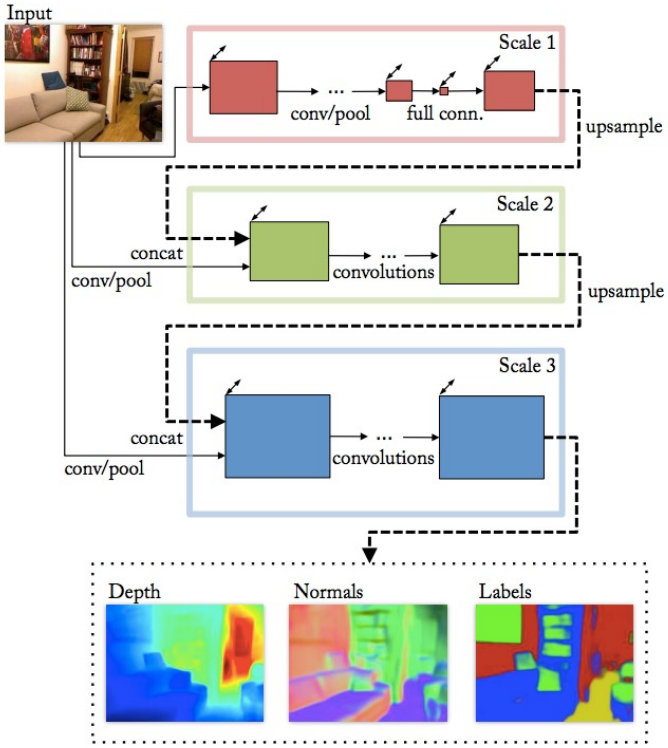
Figure source

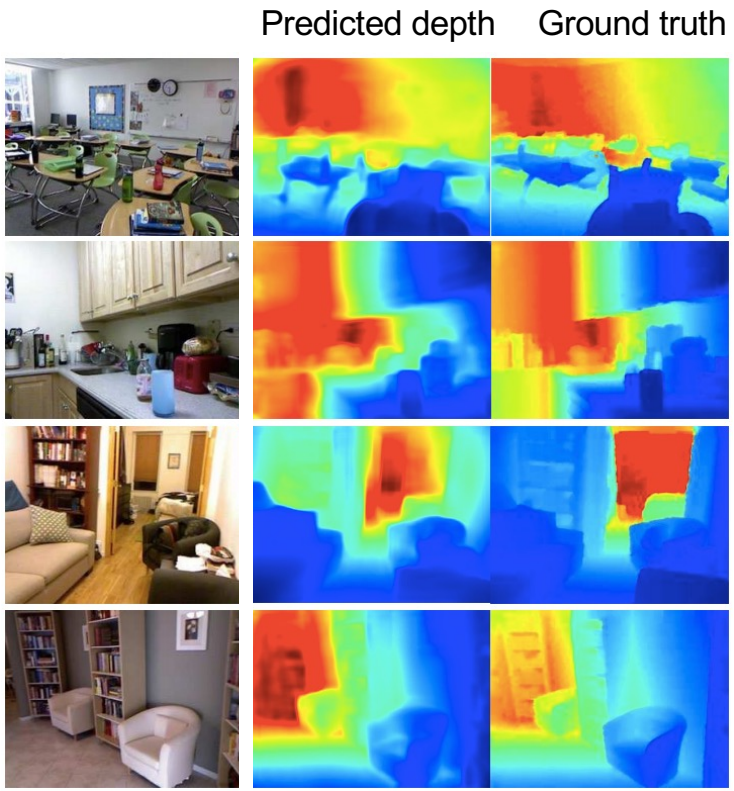
Slide courtesy of Lana Lezebnik

Other dense prediction tasks

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

Depth and normal estimation

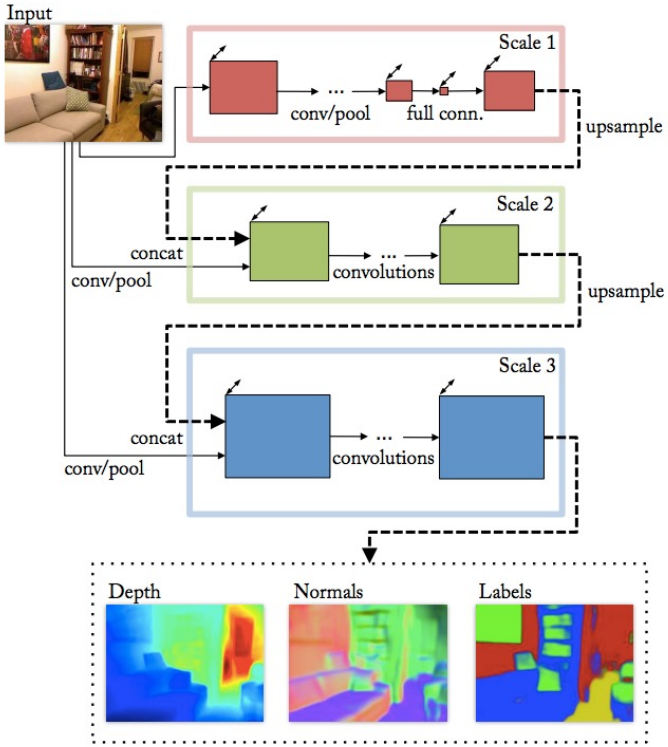


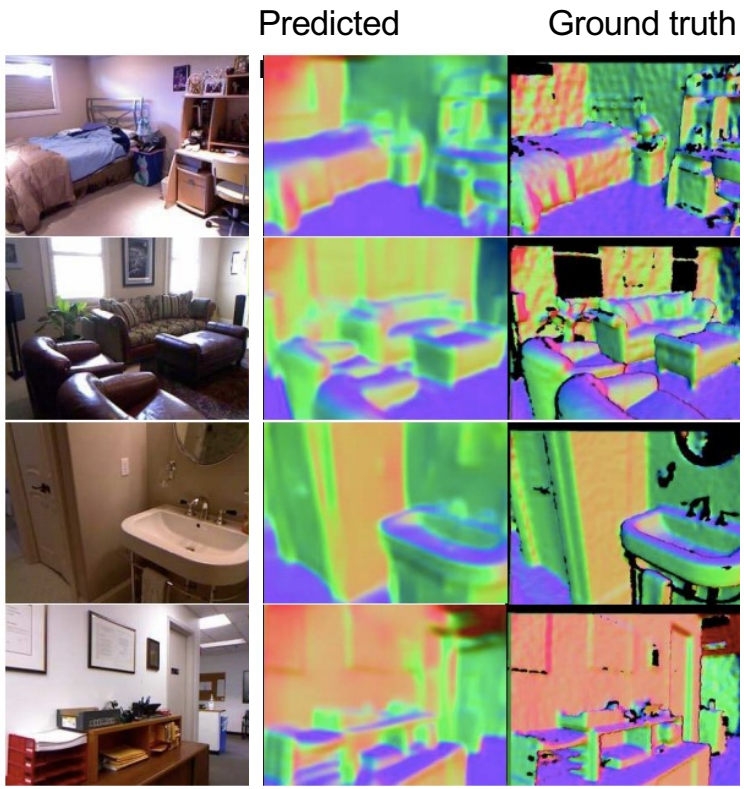


D. Eigen and R. Fergus, [Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture](), ICCV 2015

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

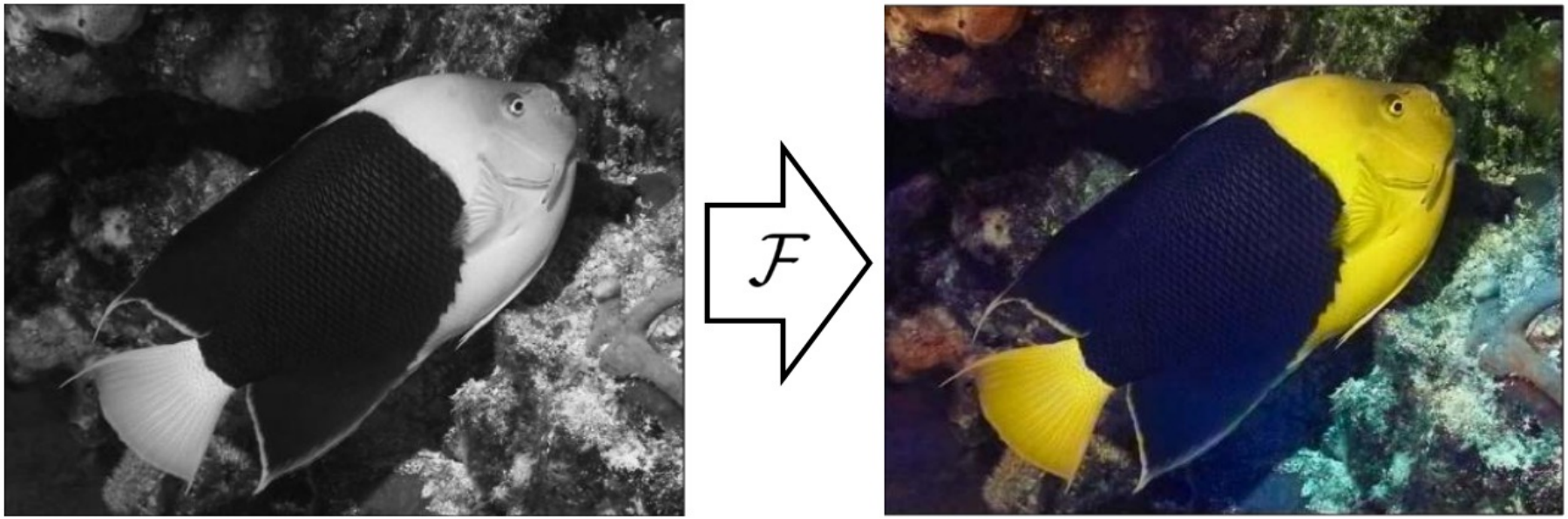
Slide courtesy of Lana Lezebnik

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)

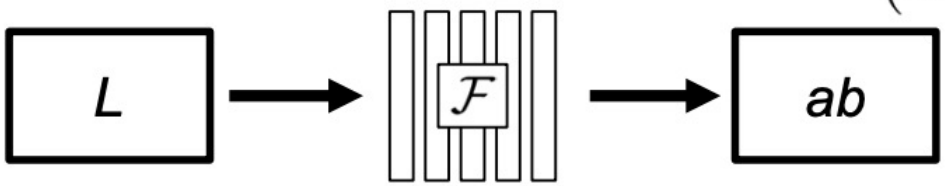


Grayscale image: *L* channel

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

Concatenate (*L*,*ab*) channels

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



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