EECS 545: Machine Learning

Lecture 12: Optimization for Neural Networks, Convolutional Neural Network

Honglak Lee

02/19/2025

UNIVERSITY OF MICHIGAN

Logistics

- HW3 due 02/25/25. We highly recommend you start ASAP if you haven’t.

Outline

- Optimization
- CNN basics
- Examples of CNN Architectures
- Applications of CNN

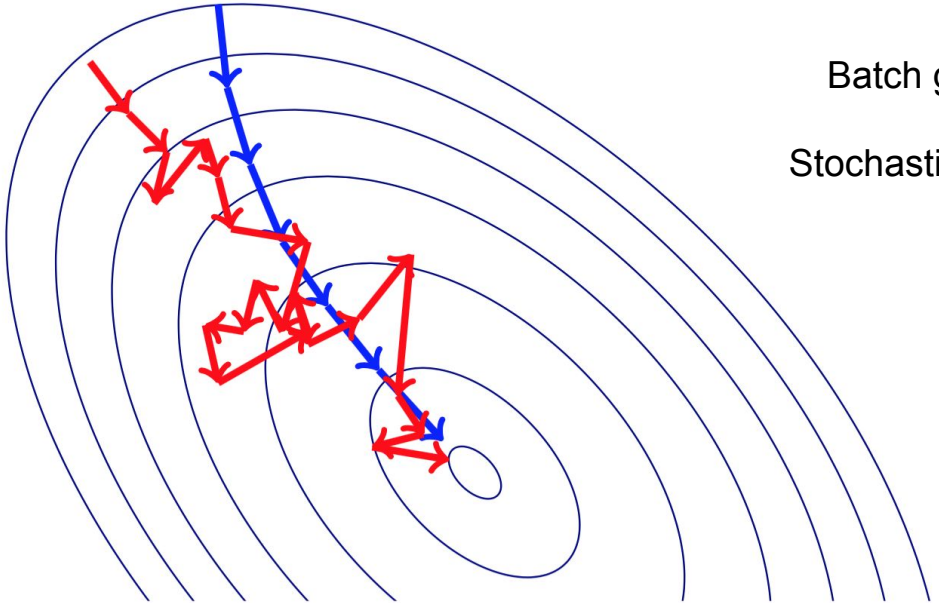
Optimization

- Stochastic Gradient Descent
- Momentum Method
- Adaptive Learning Methods
- (AdaGrad, RMSProp, Adam)

Broadly applicable for many ML methods

- Batch Normalization ← specific to neural nets

Stochastic Gradient Descent



Batch gradient descent
vs
Stochastic gradient descent

Slide credit: Shubhendu Trivedi, Risi Kondor

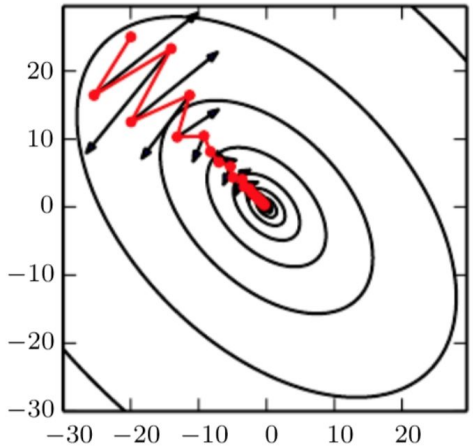
Limitations of GD and SGD

- GD and SGD suffer in the following scenarios:
  - Error surface has high curvature
  - We get small but consistent gradients
  - The stochastic gradients are very noisy (for SGD)

Slide credit: Shubhendu Trivedi, Risi Kondor

Momentum

- The Momentum method is a method to accelerate learning using GD or SGD

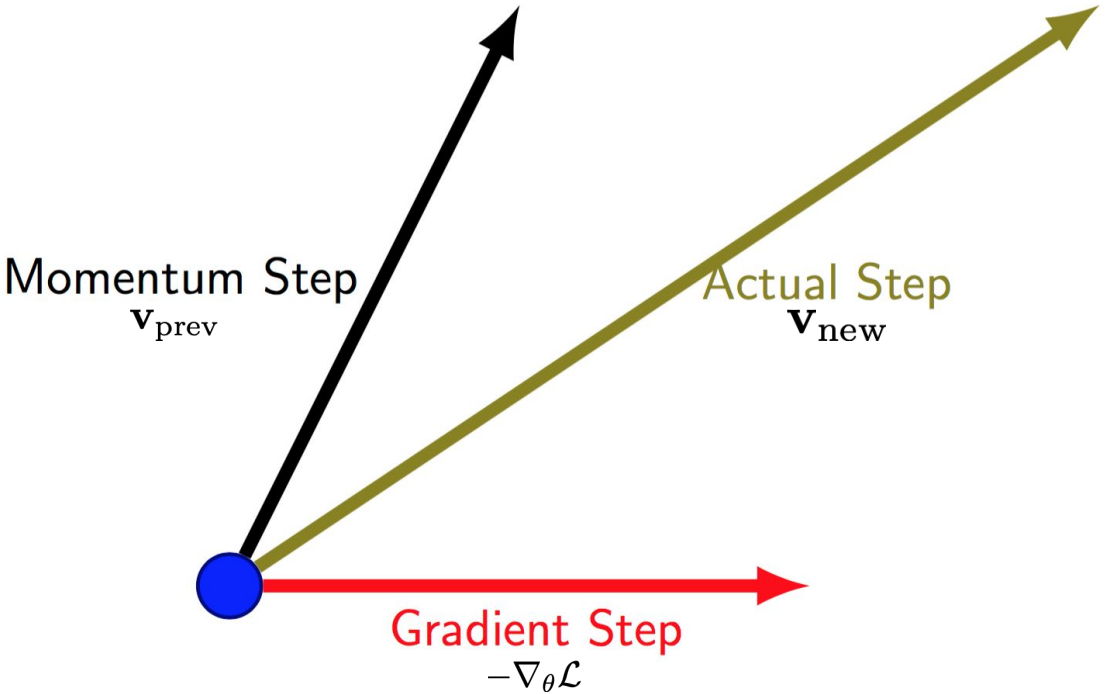


- Illustration of how momentum traverses such an error surface better compared to gradient descent
- Visualization of GD with momentum

Slide credit: Shubhendu Trivedi, Risi Kondor

Momentum

$$\mathbf{v}_{\text{new}} = \alpha \mathbf{v}_{\text{prev}} - \epsilon \nabla_\theta \left(\mathcal{L} \left(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)} \right) \right)$$



[Visualization of GD with momentum]

Slide credit: Shubhendu Trivedi, Risi Kondor

Adaptive Learning Methods: Motivation

- Until now we assigned the same learning rate to all features
  - If the features vary in importance and frequency, is this a good idea?
  - Probably not!
- Popular methods for adaptive learning rates:
  - AdaGrad, RMSProp, Adam, etc.
- High-level idea:
  - Discount the learning rate for each parameter by dividing with the "amplitude" (running average) of the gradient for that parameter.
  - Training is more stable and robust to noisy gradients and the choice of initial learning rates (e.g., even a reasonably large initial learning rate works)

Optimization with SGD and its variants

SGD: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

Momentum: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \hat{\mathbf{g}}$ then $\theta \leftarrow \theta + \mathbf{v}$

AdaGrad: $\mathbf{r} \leftarrow \mathbf{r} + \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$ then $\Delta\theta - \leftarrow \frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$ then $\theta \leftarrow \theta + \Delta\theta$

RMSProp: $\mathbf{r} \leftarrow \rho \mathbf{r} + (1-\rho) \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$ then $\Delta\theta - \leftarrow \frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$ then $\theta \leftarrow \theta + \Delta\theta$

ADAM: $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1-\rho_1) \hat{\mathbf{g}}$

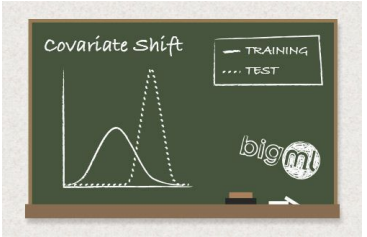
$\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1-\rho_2) \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$

$\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-\rho_1^t}, \hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1-\rho_2^t}$ then $\Delta\theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta}$ then $\theta \leftarrow \theta + \Delta\theta$

- Visualization:
  - Visualization of optimizers
  - Distill: why momentum really works

Slide credit: Shubhendu Trivedi, Risi Kondor

Batch Normalization



- **BN is specific to neural networks**
- We have a recipe to compute gradients (backpropagation) and update all parameters (SGD, adaptive learning rate methods, etc.)
- **Challenge: Internal Covariate Shift**
  - Implicit assumption during backpropagation: layer inputs remain unchanged.
  - Reality: Simultaneous updates across layers alter distributions.
  - Consequences:
    - Shifts in activation distributions across layers.
    - Training instability: unstable training, longer convergence times, suboptimal performance.
- **Insights: Addressing internal covariate shift can significantly enhance training efficiency.**

Batch normalization standardizes inputs to each layer…

- …which helps to stabilize training

- Consider standardizing the input to the input layer, i.e. our data
  - For minibatch mean μ_β and standard deviation σ_β^2, we can normalize each input x_i to

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

(ϵ added for numerical stability)

    which shifts our input to a distribution w/ mean = 0 and std dev = 1
  - Output of batch normalization (with γ, β being learnable parameters for the BN layer):

$$y_i \leftarrow \gamma \hat{x}_i + \beta$$

- Batch normalization extends this idea to the input of *every* layer, not just the input layer
  - But after going through previous layers and activations, the input does not necessarily reflect the original input distribution
  - To reflect the true distribution of the data, keep track of scale γ and shift β for each weight

Batch Normalization: forward prop

- Normalize distribution of each input feature in each layer across each minibatch to *N*(0, 1)
- Learn the scale and shift

- After training, at test time: Use running averages of μ and σ collected during training, use these for evaluating new input *x*

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Batch Normalization: backpropagation

- Differentiable via chain rule

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2} (\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

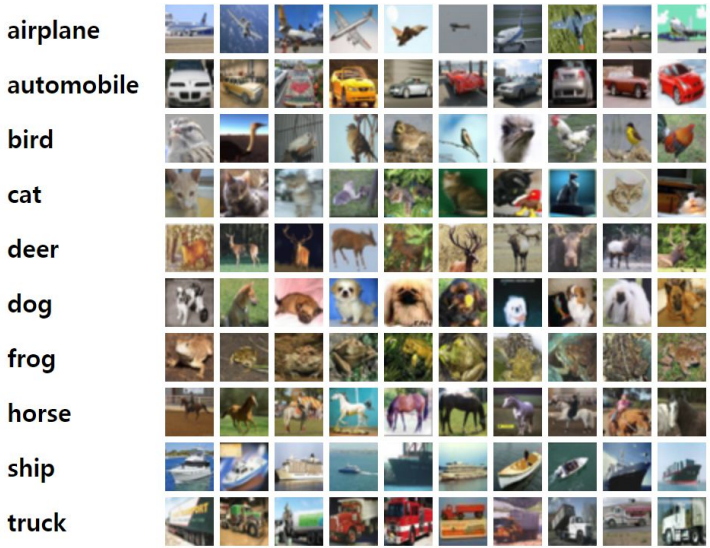
Outline

- Optimization
- CNN basics
- Examples of CNN Architectures
- Applications of CNN

Image Classification

Question: How can we categorize images?

Example: CIFAR-10 dataset



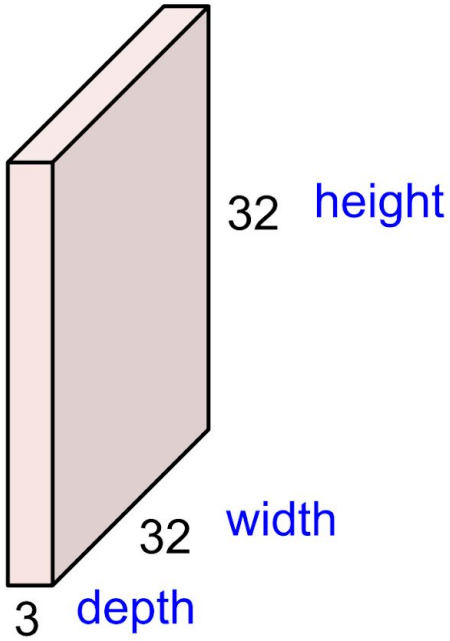
https://www.cs.toronto.edu/~kriz/cifar.html

Image Classification

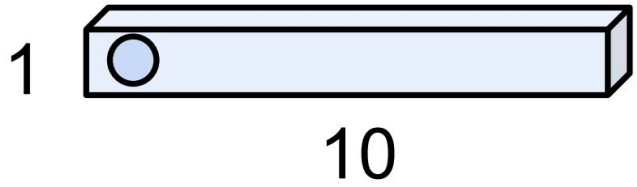
“airplane”

Image Classification

32x32x3 image



10-class output



Note: in this lecture, we use vectors visualized as rows

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Fully Connected Layer

Note: in this lecture, we use vectors visualized as rows

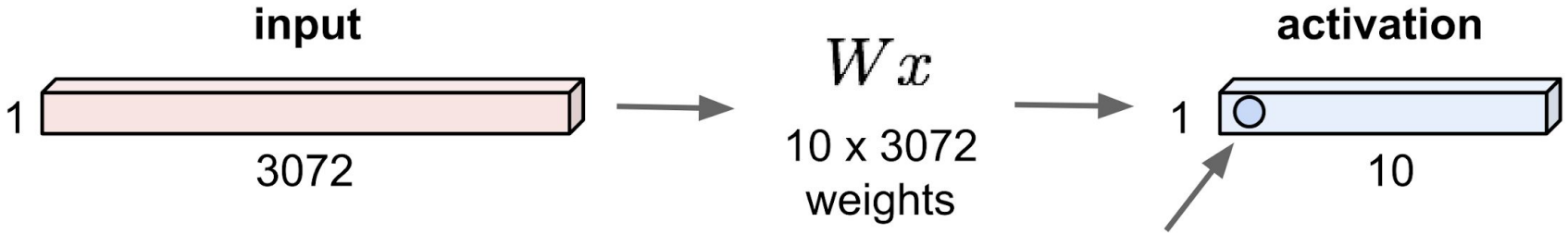
32x32x3 image -> stretch to 3072 x 1

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Fully Connected Layer

Note: in this lecture, we use vectors visualized as rows

32x32x3 image -> stretch to 3072 x 1

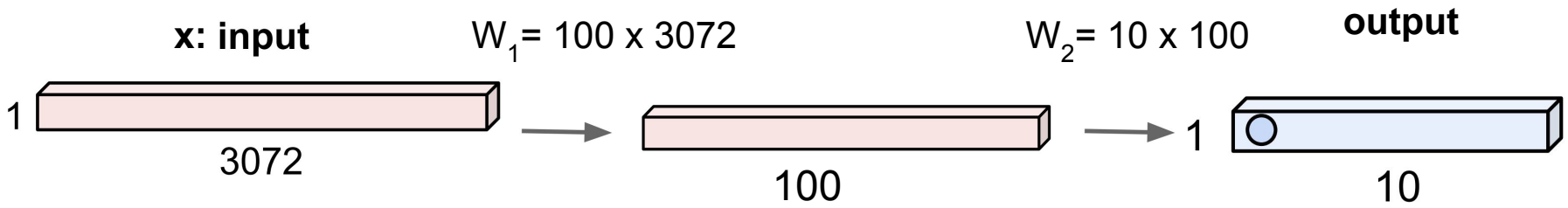


1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

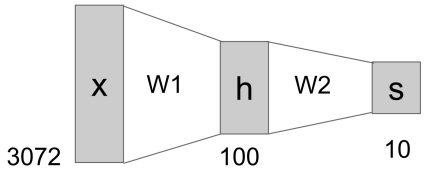
Note: k-th row of W is a transposed version of weight vector $\mathbf{w}_k \in \mathbb{R}^{3072}$ for the class k.

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Neural Network with 1 hidden layer (with Rectified Linear Units)



Alternatively (diagram):



Alternatively (short notation): $f = W_2 \max(0, W_1 x)$

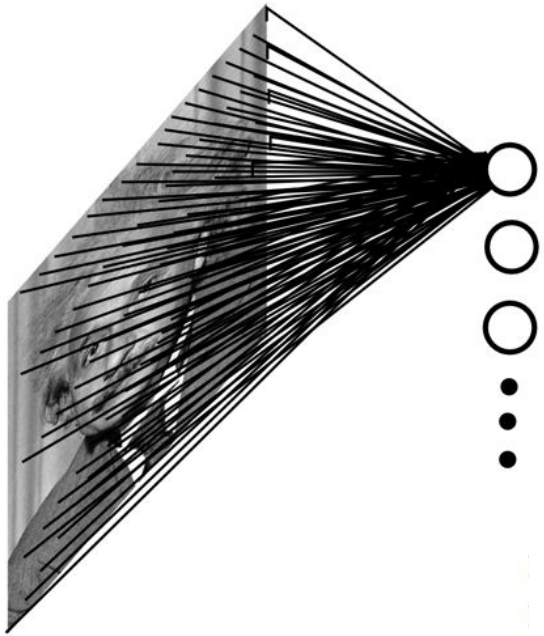
Deeper models: We can add more layers, etc.

Q. Have we solved the problem? Will this be applicable for larger-sized images?

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Scaling to larger-sized images

Example: 200x200 image, 40k hidden units: Fully connected neural network would have ~2B parameters!



Slide credit: Russ Salakhutdinov

Motivation: Convolutional Neural Networks

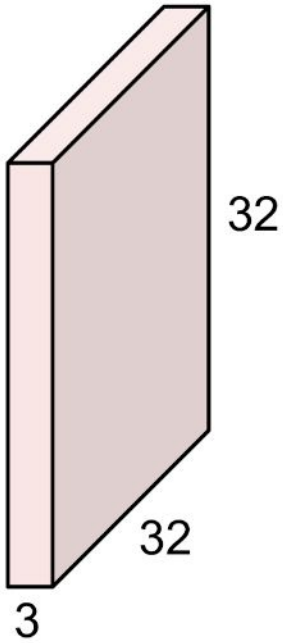
- How can we design neural networks that are specifically adapted for classifying large-sized images?
  - Must deal with very **high-dimensional inputs**: 150 x 150 pixels = 22500 inputs, or 3 x 22500 if RGB pixels
  - Can exploit the **2D topology** of pixels (or 3D for video data)
  - Can build in **invariance** to certain variations: translation, illumination, etc.
- **Convolutional networks** leverage these ideas
  - Local connectivity
  - Convolution, parameter sharing
  - Pooling / subsampling hidden units

Slide credit: Russ Salakhutdinov

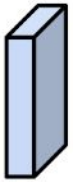
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



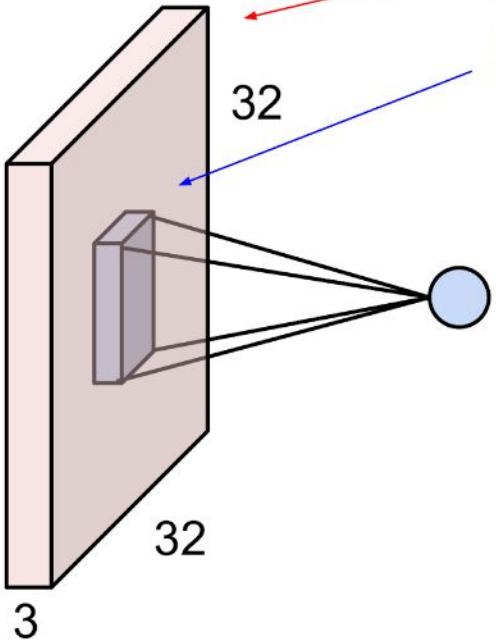
5x5x3 filter



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

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Convolution Layer



32x32x3 image

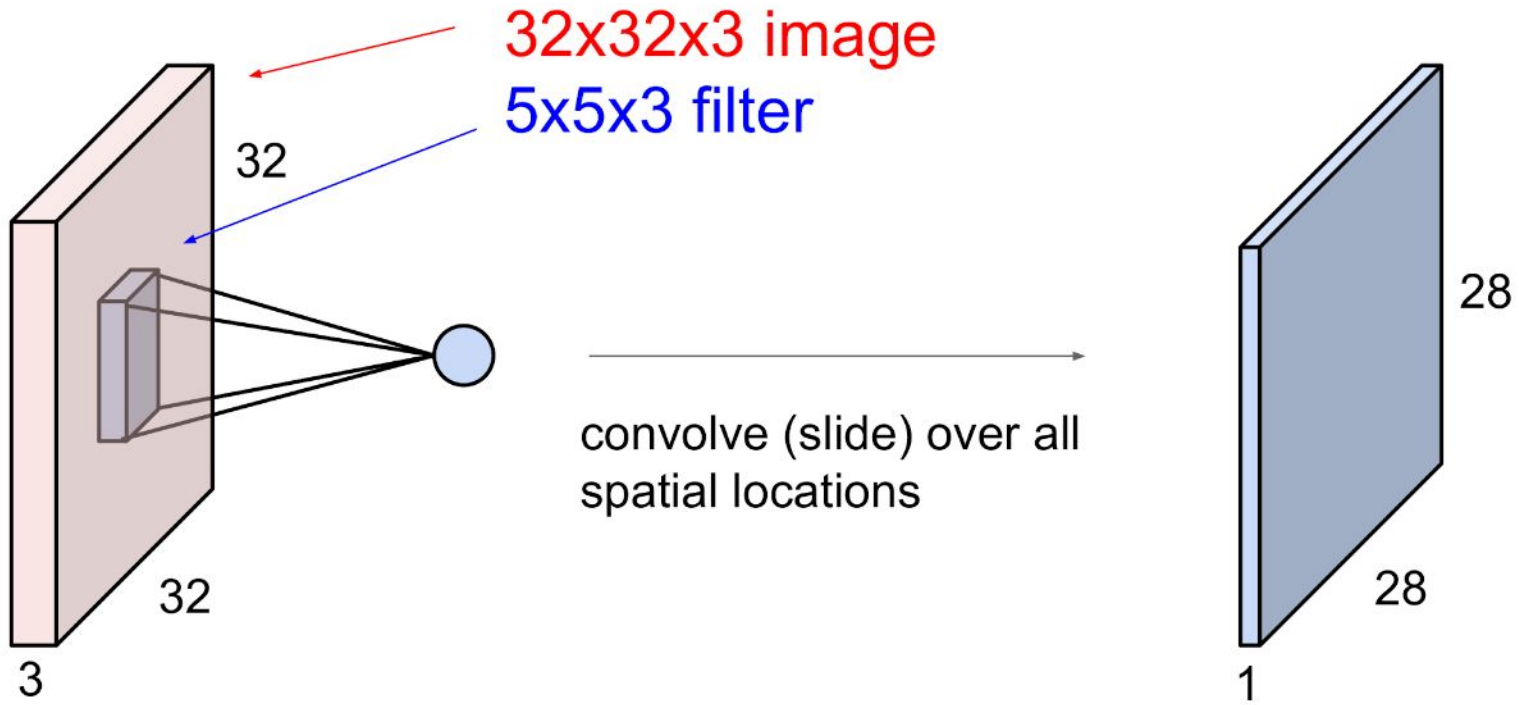
5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Convolution Layer



Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

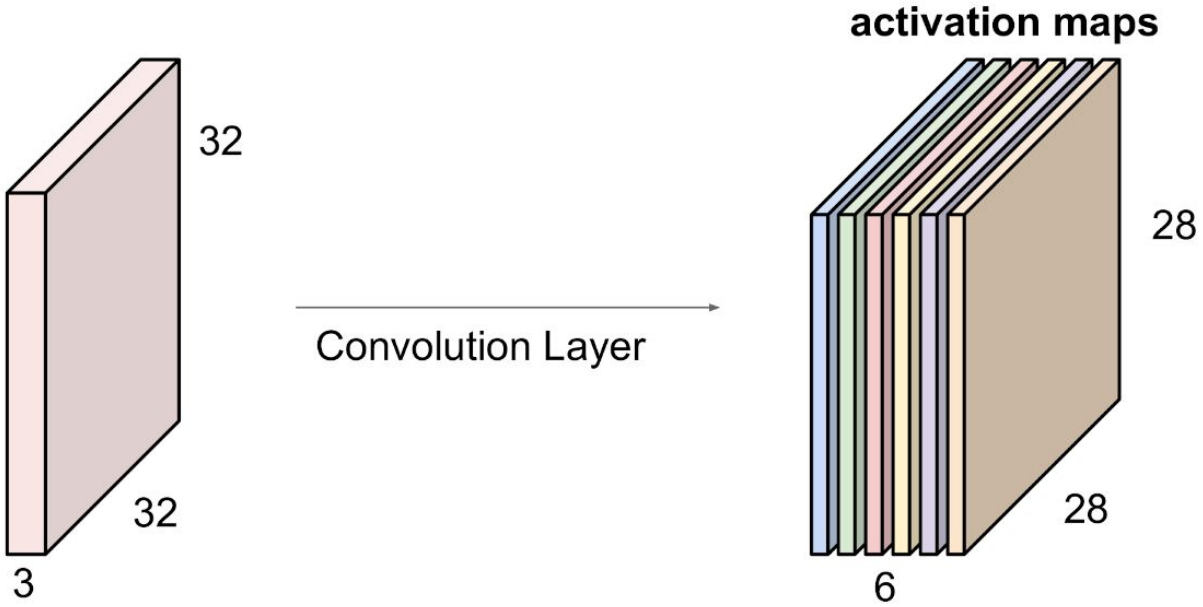
Convolution Layer

consider a second filter

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Convolution Layer

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

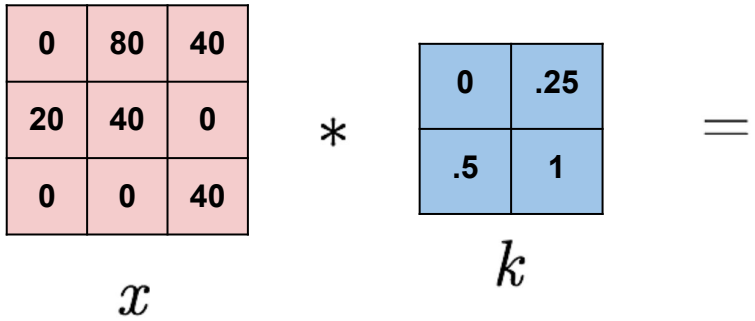
Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Discrete Convolution

- Applying filter to an image via “sliding window” can be efficiently computed as a convolution of an image x with kernel k as follows

$$(\mathbf{x} *_{\text{valid}} \mathbf{k})_{i,j} = \sum_{m=i}^{i+H_k-1} \sum_{n=j}^{j+W_k-1} x_{m,n} k_{i-m+H_k, j-n+W_k}$$

- Example

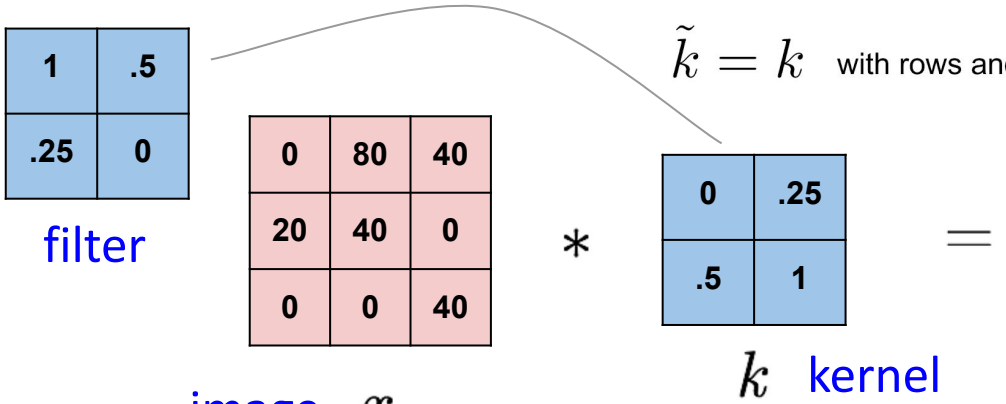


Discrete Convolution

- Applying filter to an image via “sliding window” can be efficiently computed as a convolution of an image x with kernel k as follows

$$(\mathbf{x} *_{\text{valid}} \mathbf{k})_{i,j} = \sum_{m=i}^{i+H_k-1} \sum_{n=j}^{j+W_k-1} x_{m,n} k_{i-m+H_k, j-n+W_k}$$

- Example

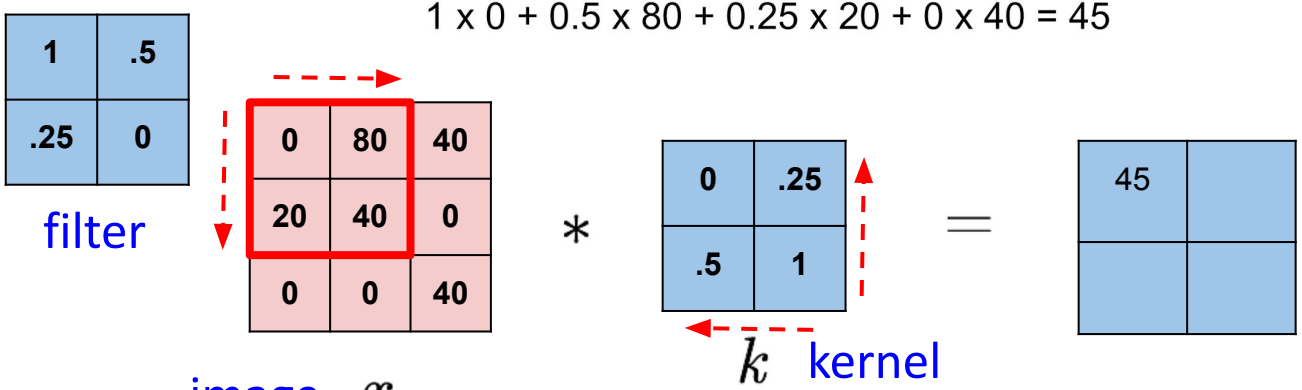


Discrete Convolution

- Applying filter to an image via “sliding window” can be efficiently computed as a convolution of an image x with kernel k as follows

$$(\mathbf{x} *_{\text{valid}} \mathbf{k})_{i,j} = \sum_{m=i}^{i+H_k-1} \sum_{n=j}^{j+W_k-1} x_{m,n} k_{i-m+H_k, j-n+W_k}$$

- Example

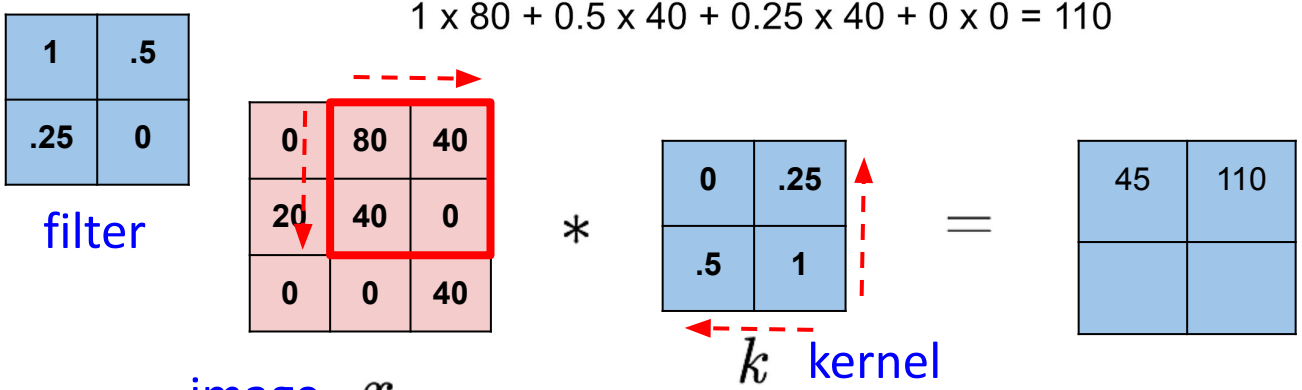


Discrete Convolution

- Applying filter to an image via “sliding window” can be efficiently computed as a convolution of an image x with kernel k as follows

$$(\mathbf{x} *_{\text{valid}} \mathbf{k})_{i,j} = \sum_{m=i}^{i+H_k-1} \sum_{n=j}^{j+W_k-1} x_{m,n} k_{i-m+H_k, j-n+W_k}$$

- Example



Discrete Convolution

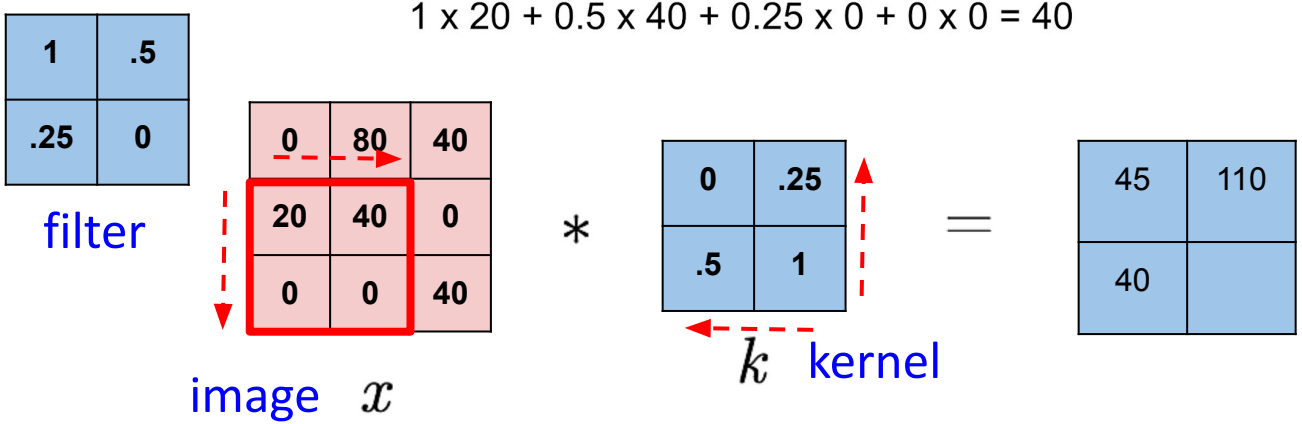
- Applying filter to an image via “sliding window” can be efficiently computed as a convolution of an image x with kernel k as follows

$$(\mathbf{x} *_{\text{valid}} \mathbf{k})_{i,j} = \sum_{m=i}^{i+H_k-1} \sum_{n=j}^{j+W_k-1} x_{m,n} k_{i-m+H_k,j-n+W_k}$$

- Example

1 x 20 + 0.5 x 40 + 0.25 x 0 + 0 x 0 = 40

kernel is a flipped version of a filter



Discrete Convolution

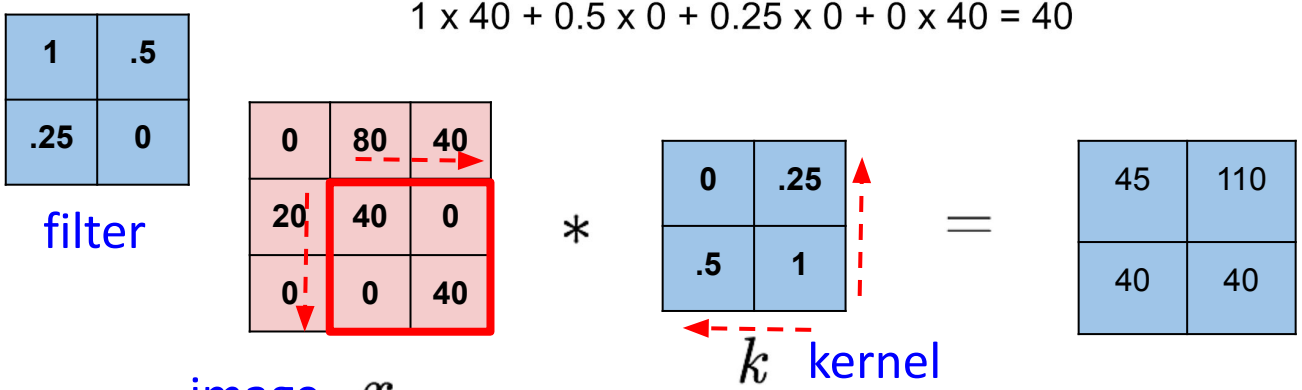
- Applying filter to an image via “sliding window” can be efficiently computed as a convolution of an image x with kernel k as follows

$$(\mathbf{x} *_{\text{valid}} \mathbf{k})_{i,j} = \sum_{m=i}^{i+H_k-1} \sum_{n=j}^{j+W_k-1} x_{m,n} k_{i-m+H_k,j-n+W_k}$$

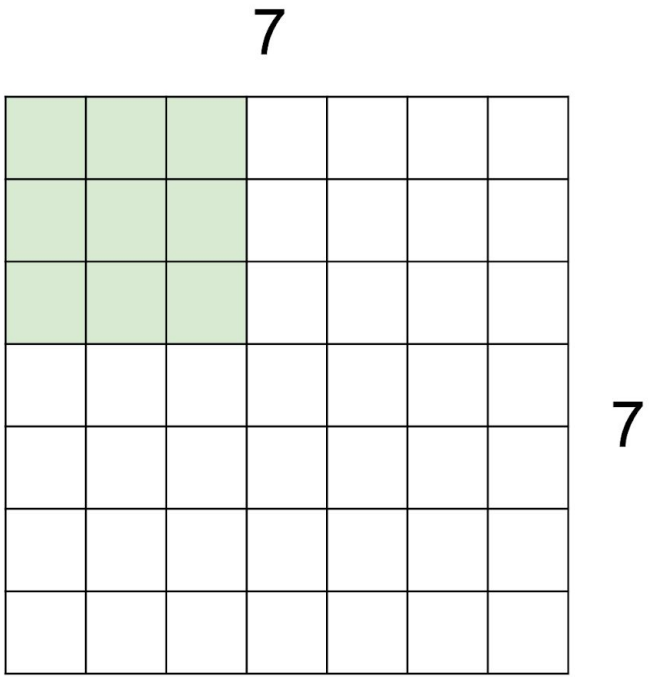
- Example

1 x 40 + 0.5 x 0 + 0.25 x 0 + 0 x 40 = 40

kernel is a flipped version of a filter



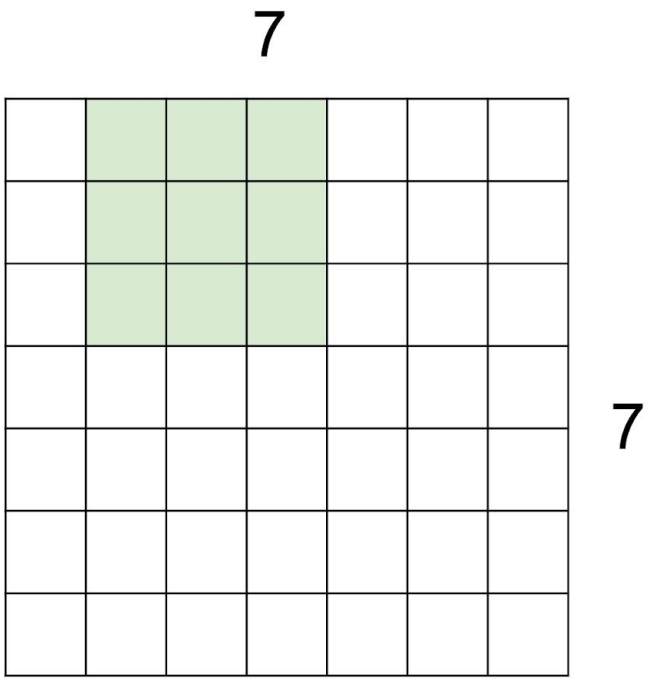
Closer look at spatial dimensions



7x7 input (spatially)
assume 3x3 filter

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Closer look at spatial dimensions

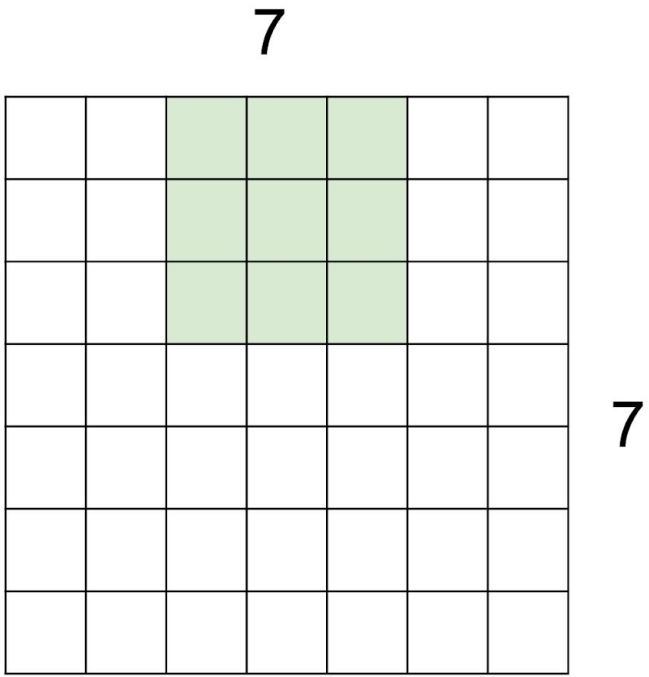


7x7 input (spatially)
assume 3x3 filter

Stride is convolutional "step size": here, = 1

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Closer look at spatial dimensions

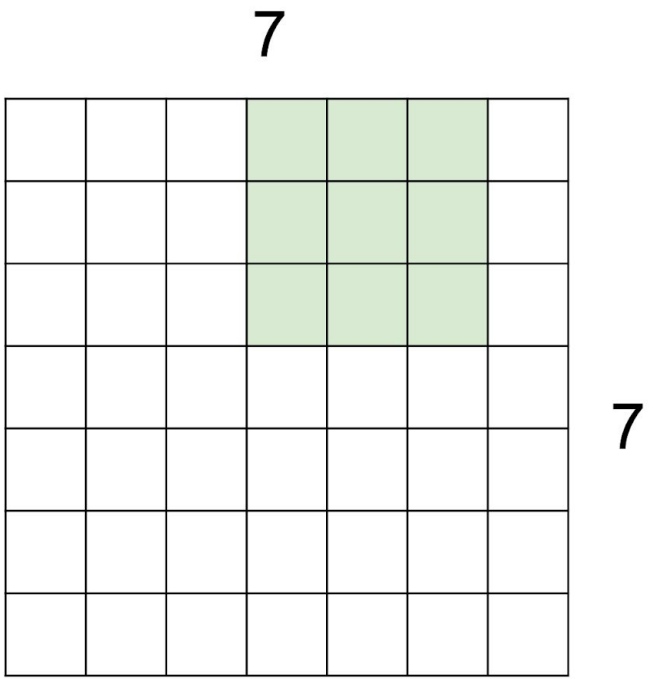


7x7 input (spatially)
assume 3x3 filter

Stride is convolutional "step size": here, = 1

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Closer look at spatial dimensions

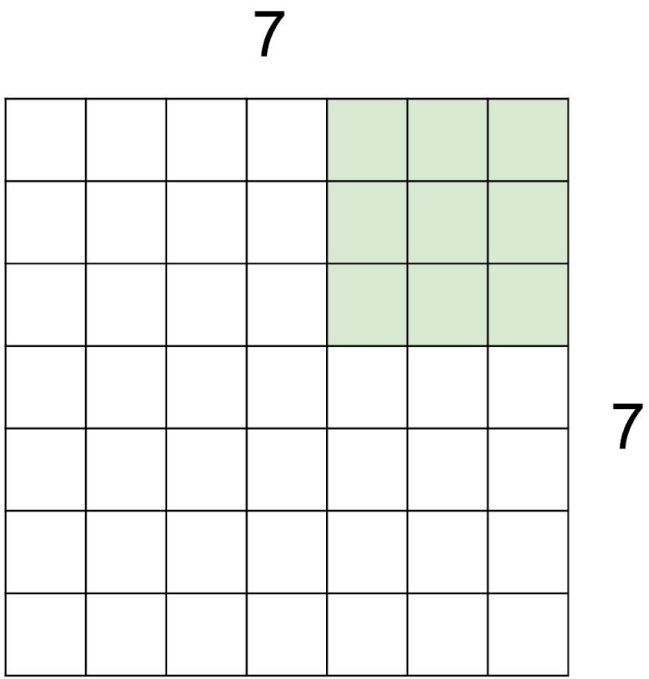


7x7 input (spatially)
assume 3x3 filter

Stride is convolutional "step size": here, = 1

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Closer look at spatial dimensions



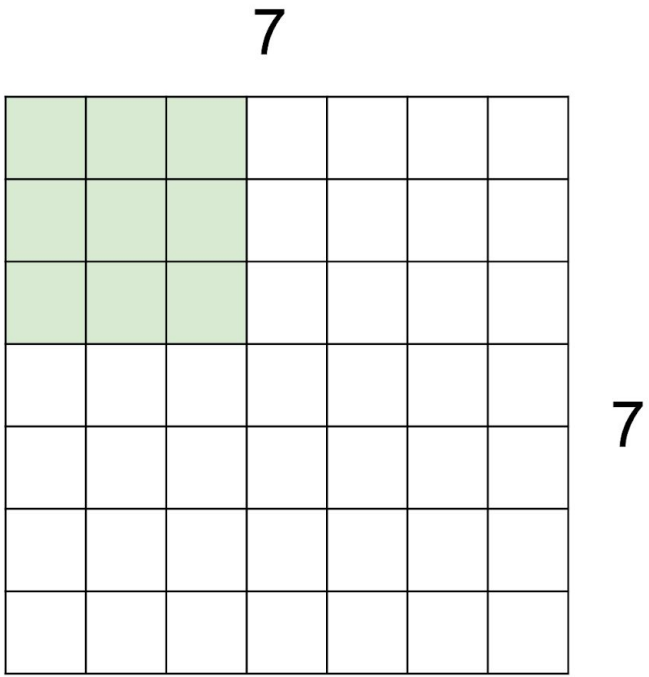
7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

Stride is convolutional "step size": here, = 1

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

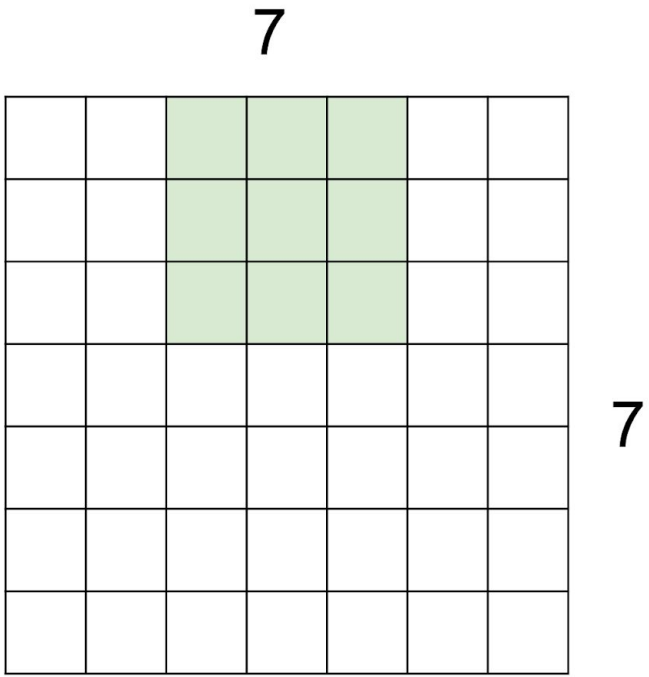
Closer look at spatial dimensions



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

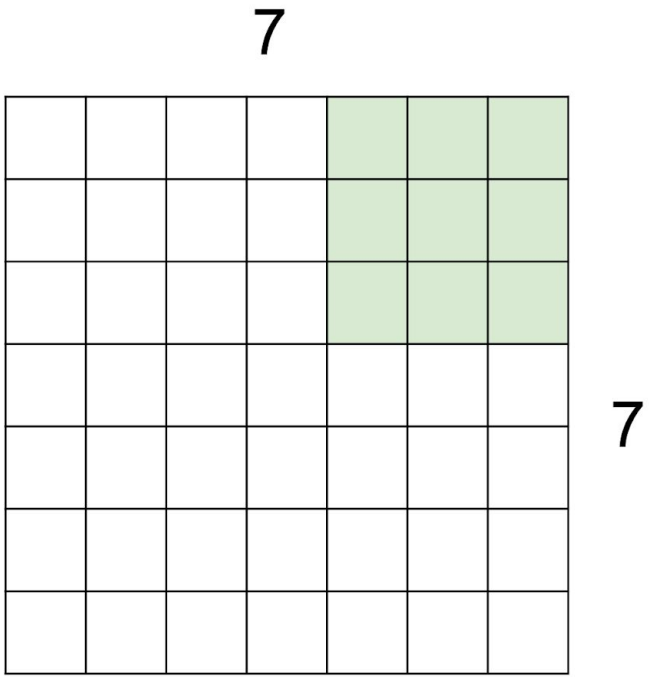
Closer look at spatial dimensions



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

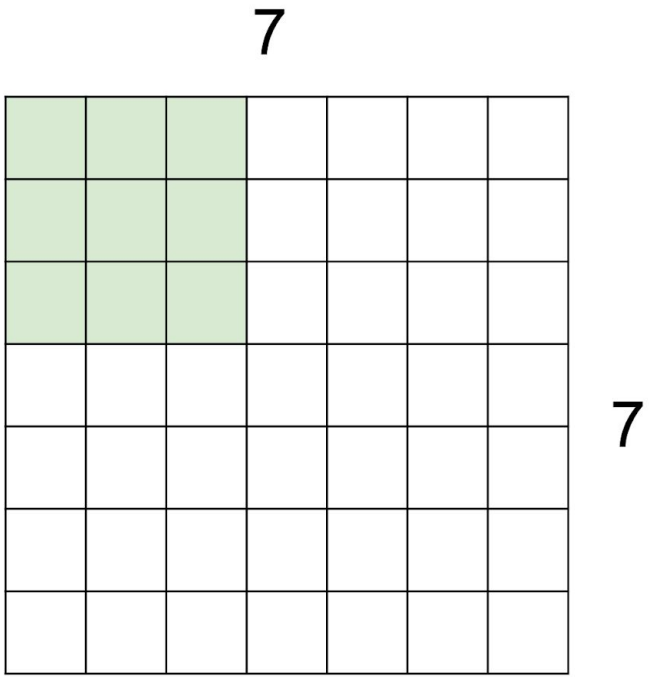
Closer look at spatial dimensions



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

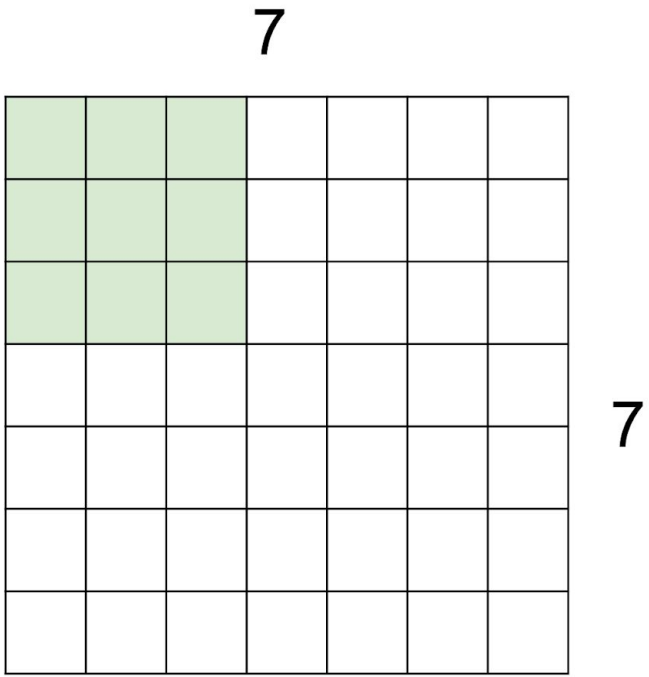
Closer look at spatial dimensions



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Closer look at spatial dimensions

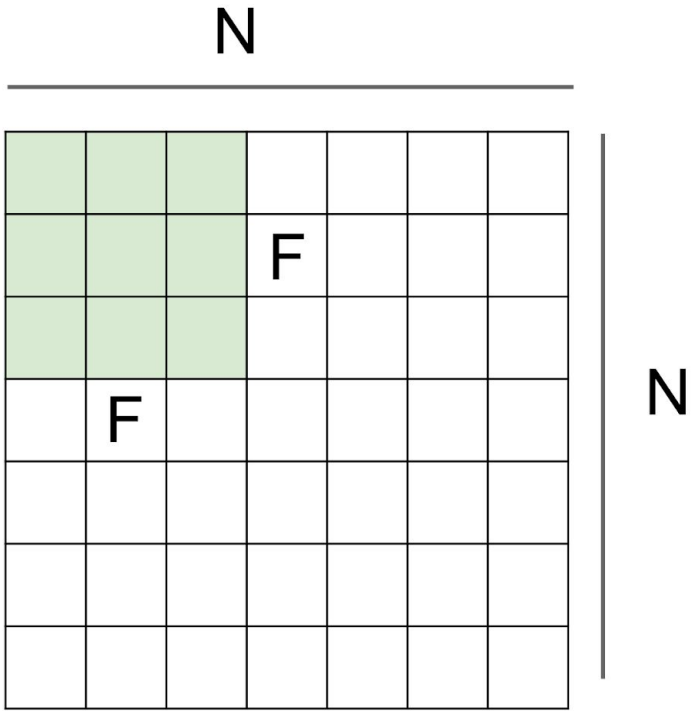


7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Closer look at spatial dimensions

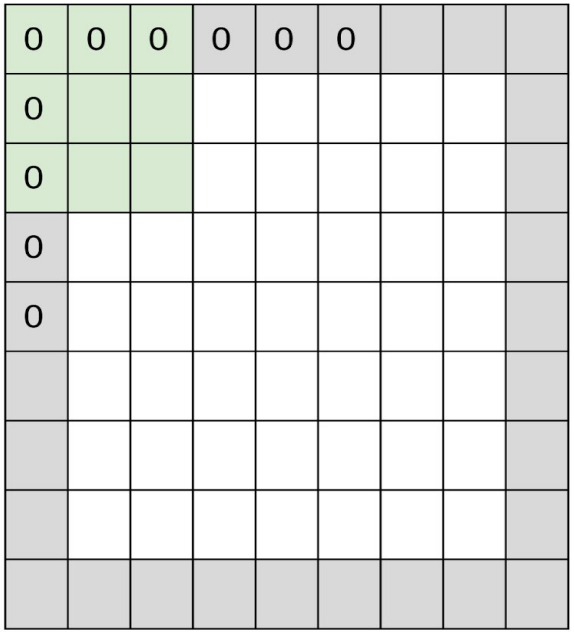


Output size:
(N - F) / stride + 1

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 :\

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

In practice: Common to zero pad the border

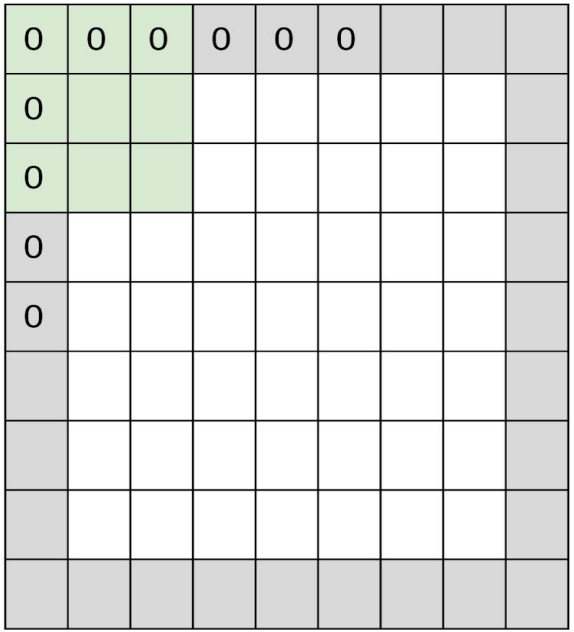


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

In practice: Common to zero pad the border



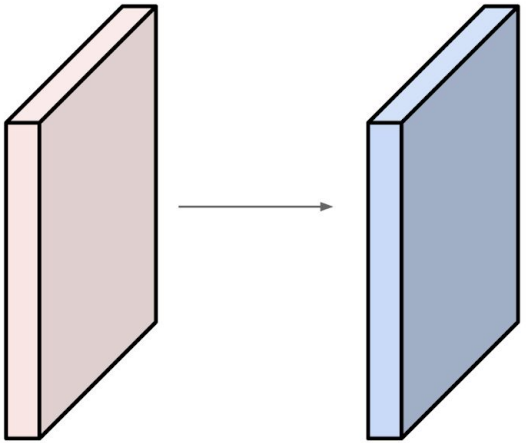
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Example

Convolution layer requires 4 hyper-parameters:
number of filters, their spatial dimension, stride, pad



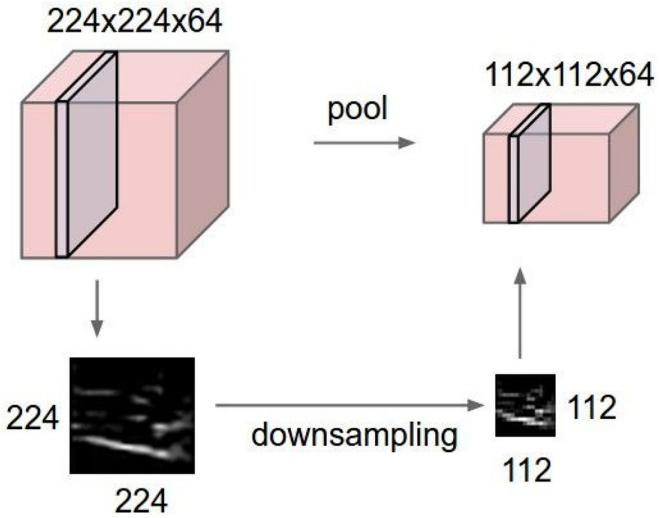
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

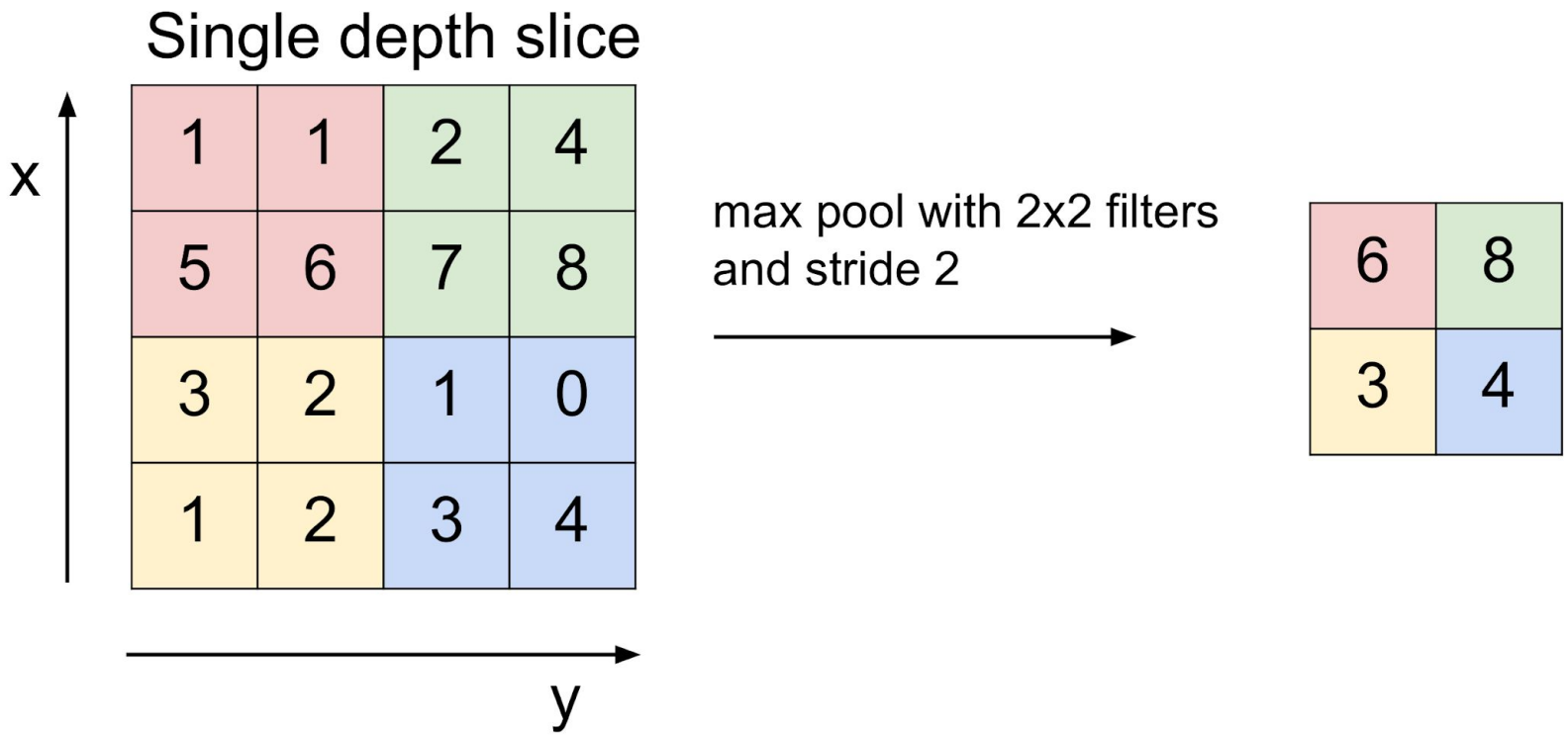
Pooling

- Makes the representations smaller and more manageable
- Operates over each activation map independently



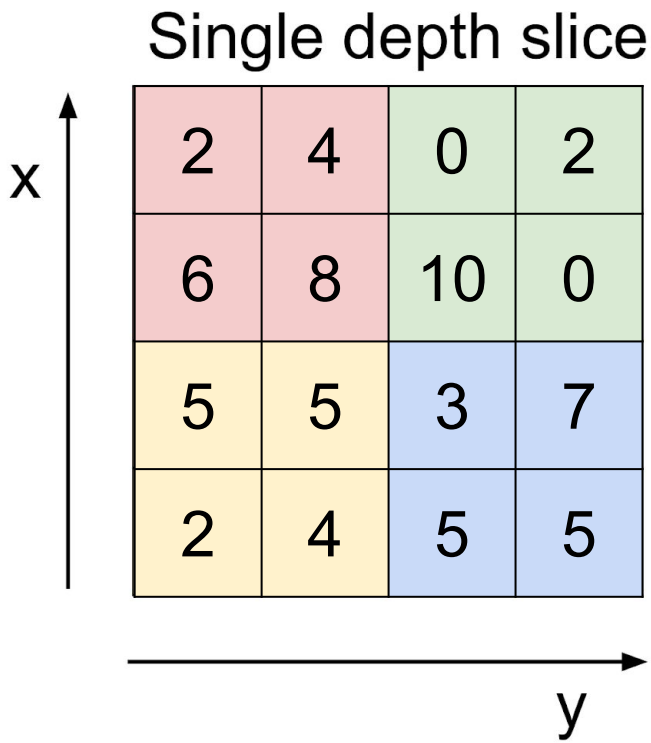
Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Max Pooling



Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Average Pooling



Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Classic ConvNet Architecture

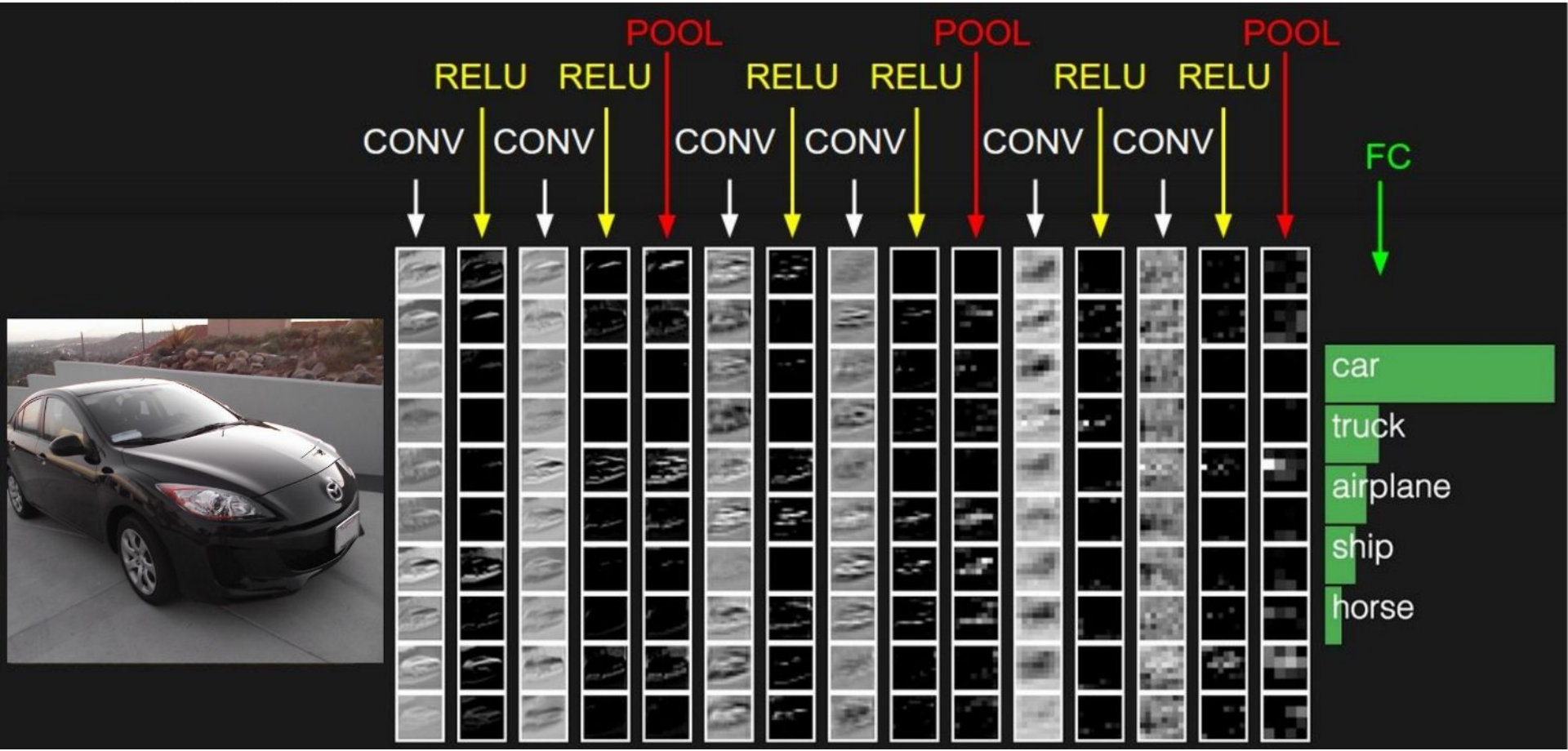
Input

Conv blocks

- Convolution + non-linear activation (relu)
- Convolution + non-linear activation (relu)
- ...
- Maxpooling 2x2

Output layer (e.g. for classification)

- Fully connected layers
- Softmax



Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

ConvNetJS demo: training on CIFAR-10

ConvNetJS CIFAR-10 demo

Description

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and verically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

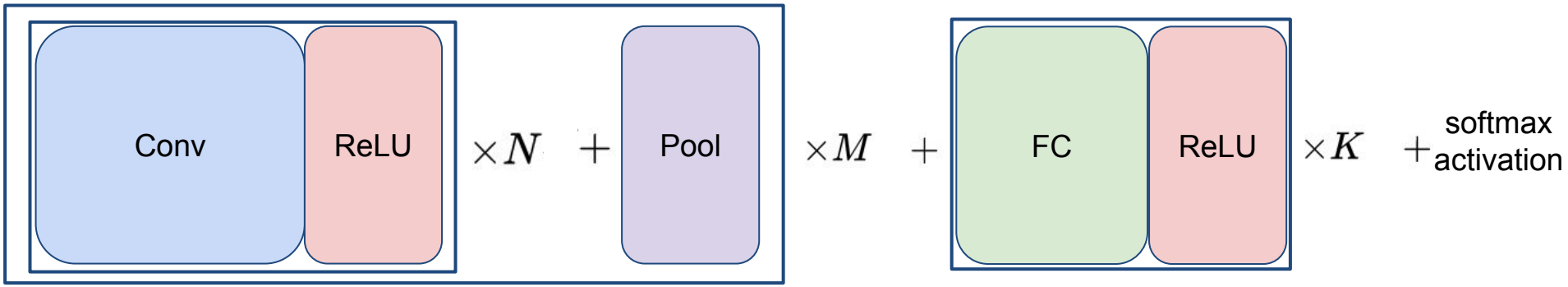
Report questions/bugs/suggestions to @karpathy.

http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Summary

- ConvNets stack CONV, POOL, FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like



where N is usually up to ~5, M is large, 0 <= K <= 2.

- but recent advances such as ResNet/GoogLeNet challenge this paradigm

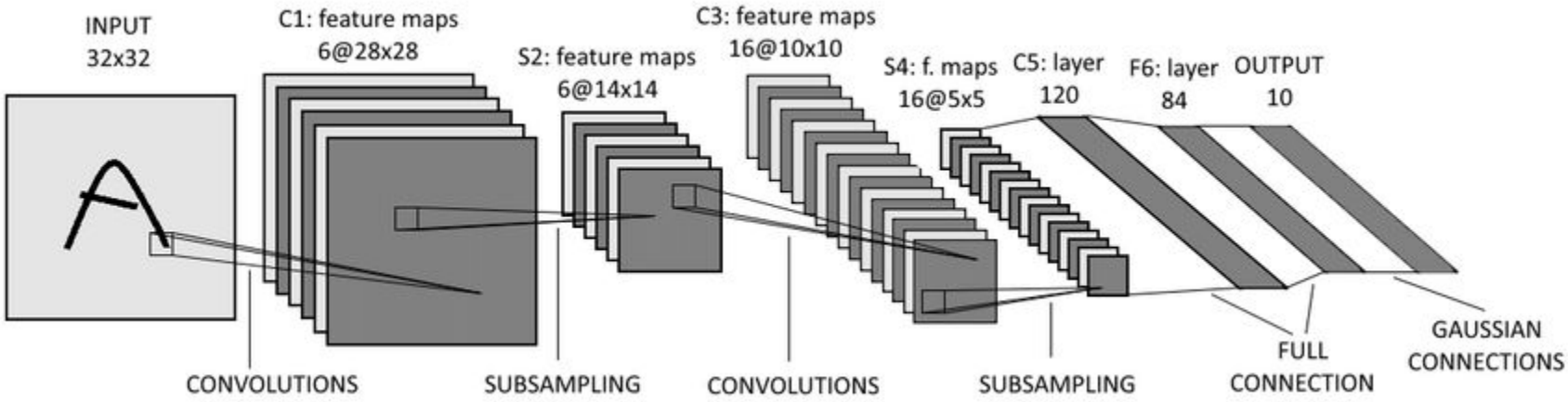
Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Outline

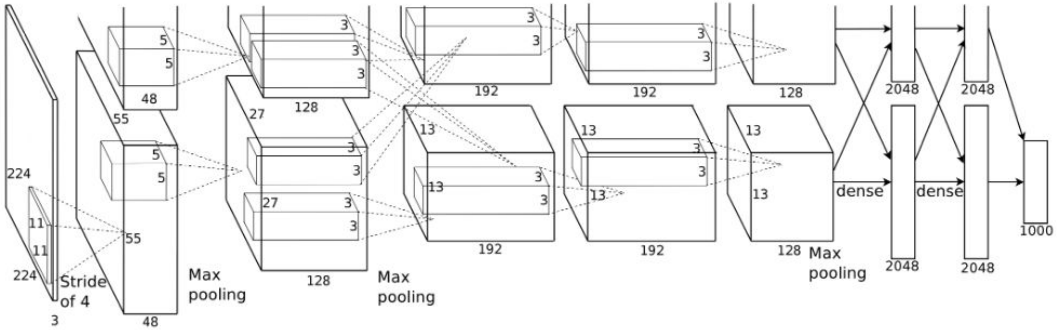
- Optimization
- CNN basics
- Examples of CNN Architectures
- Applications of CNN

LeCun et al. 1989

Gradient-based learning applied to document recognition [LeCun, Bottou, Bengio, Haffner, 1989]



AlexNet

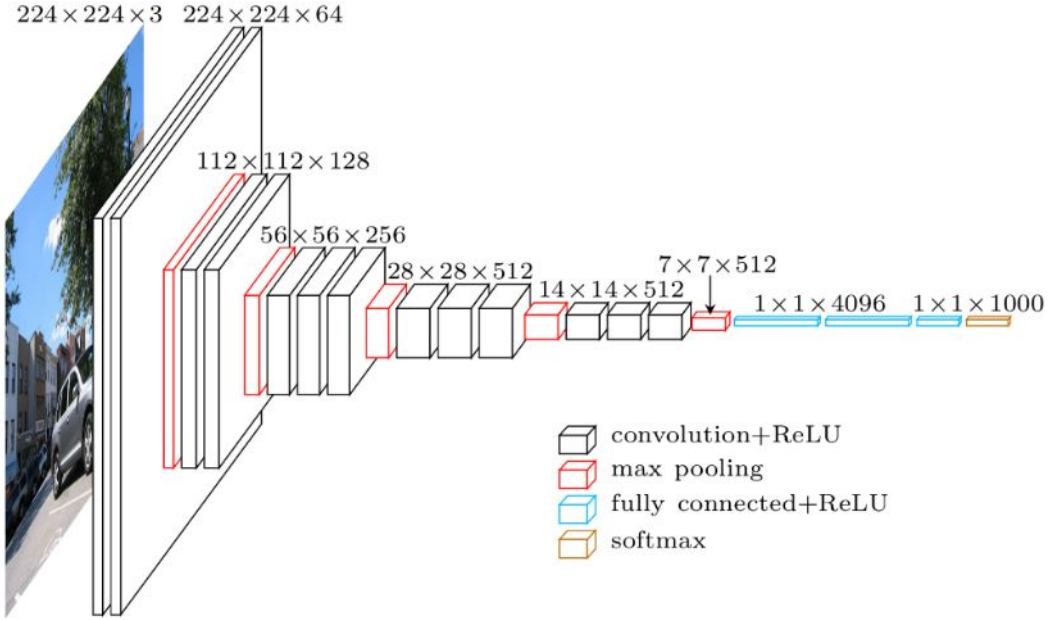


First conv layer: kernel 11x11x3x96 stride 4

- Kernel shape: (11,11,3,96)
- Output shape: (55,55,96)
- Number of parameters: 34,944
- Equivalent MLP parameters: 43.7 x 1e9

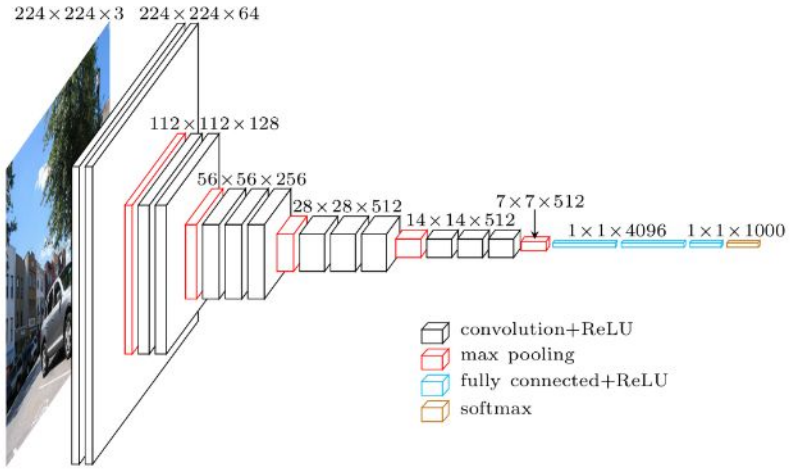
Simplified version of Krizhevsky, Alex, Sutskever, and Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012

VGG-16



Simonyan, Karen, and Zisserman. "Very deep convolutional networks for large-scale image recognition." (2014)

VGG-16



Lots of 3x3 convolution layers: more non-linearity than a single 7x7 conv layer

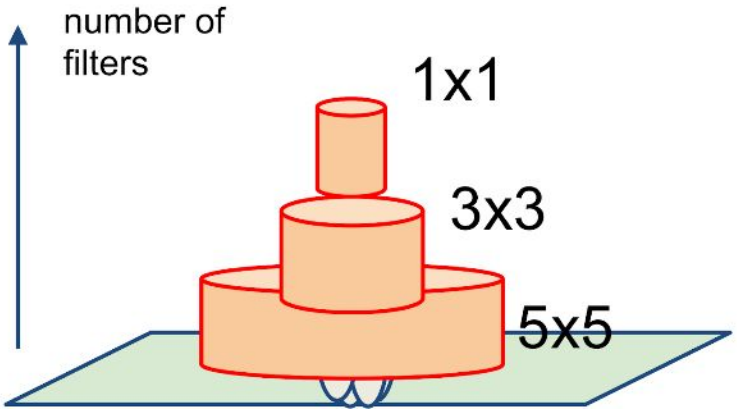
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	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					

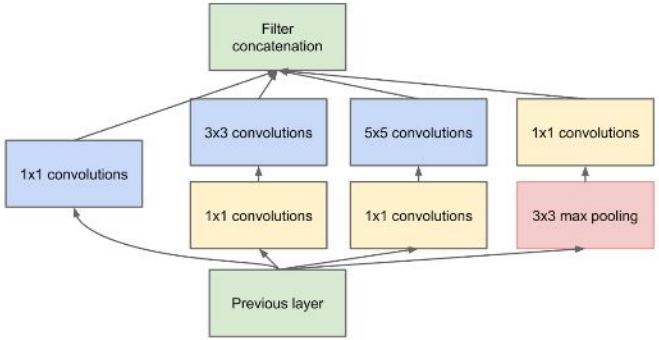
Very Deep Models

Going deep with convolutions, Szegedy et al.

GoogLeNet inception module:

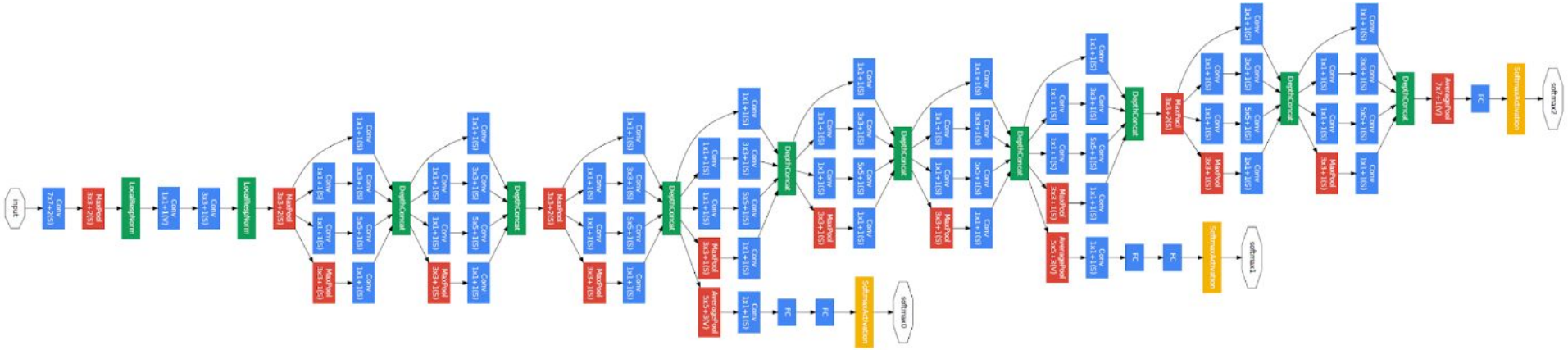
1. Multiple filter scales at each layer
2. Dimensionality reduction to keep computational requirements down





Very Deep Models

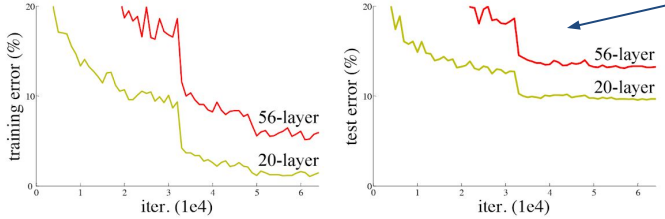
Going deep with convolutions, Szegedy et al.



Residual Networks

[He, Zhang, Ren, Sun, CVPR 2016]

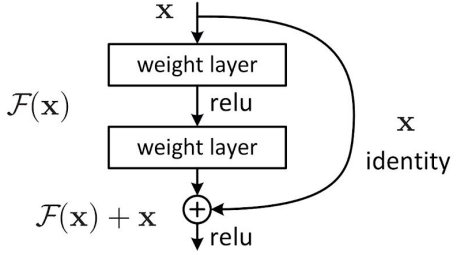
Really, really deep convnets don't train well, E.g. CIFAR10:



deeper networks perform worse due to vanishing gradient

Key idea: introduce "pass through" into each layer

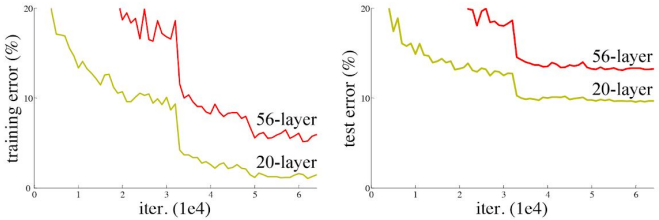
Thus only residual now needs to be learned



Residual Networks

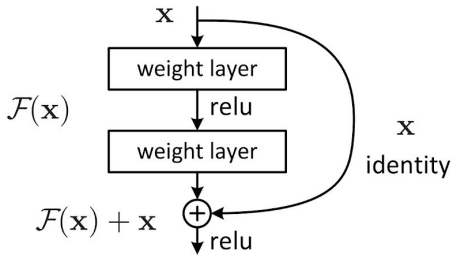
[He, Zhang, Ren, Sun, CVPR 2016]

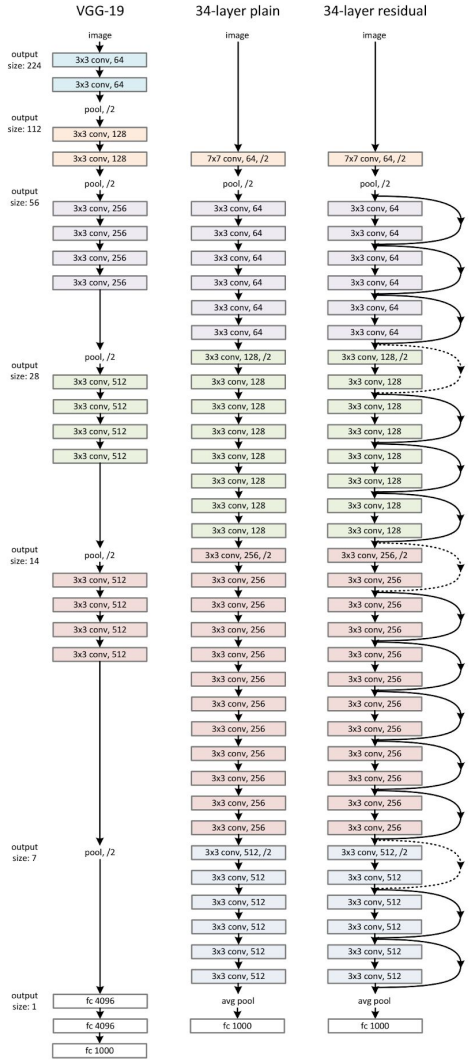
Really, really deep convnets don't train well,
E.g. CIFAR10:



Key idea: introduce "pass through" into each layer

Thus only residual now needs to be learned





Residual Networks

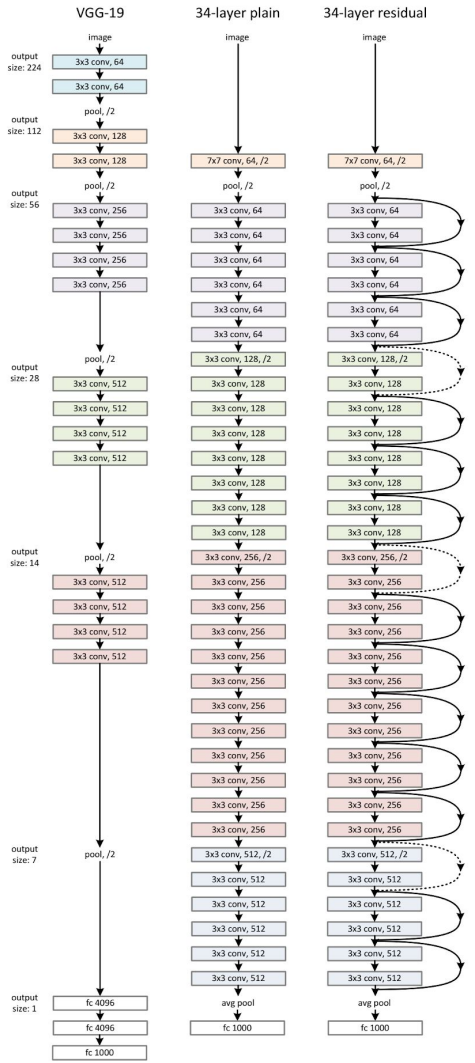
ResNet50 Compared to VGG:

- Superior accuracy in all vision tasks

  5.25% top-5 error vs 7.1%
- Less parameters

  25M vs 138M
- Computational complexity

  3.8B Flops vs 15.3B Flops
- Fully convolutional until the last layer



Deeper is better

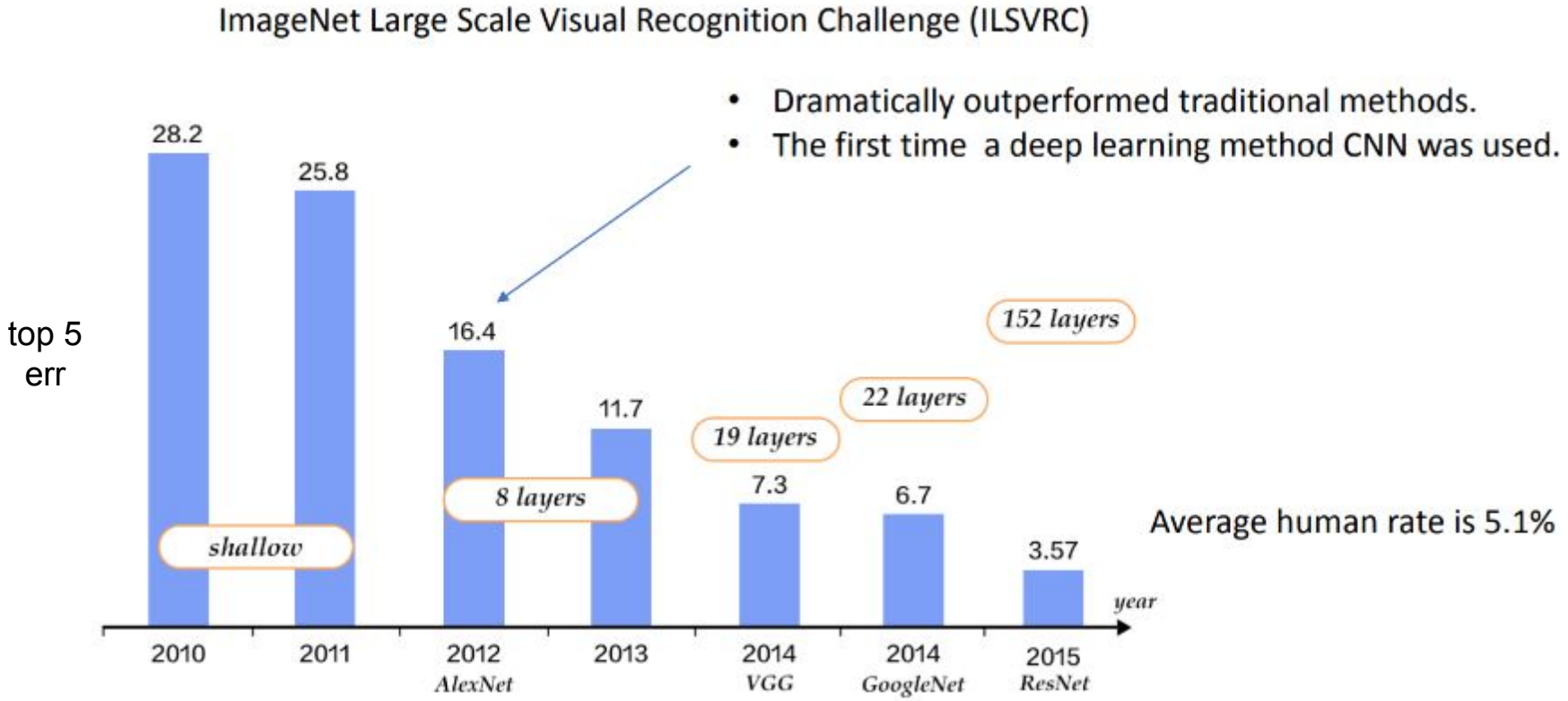
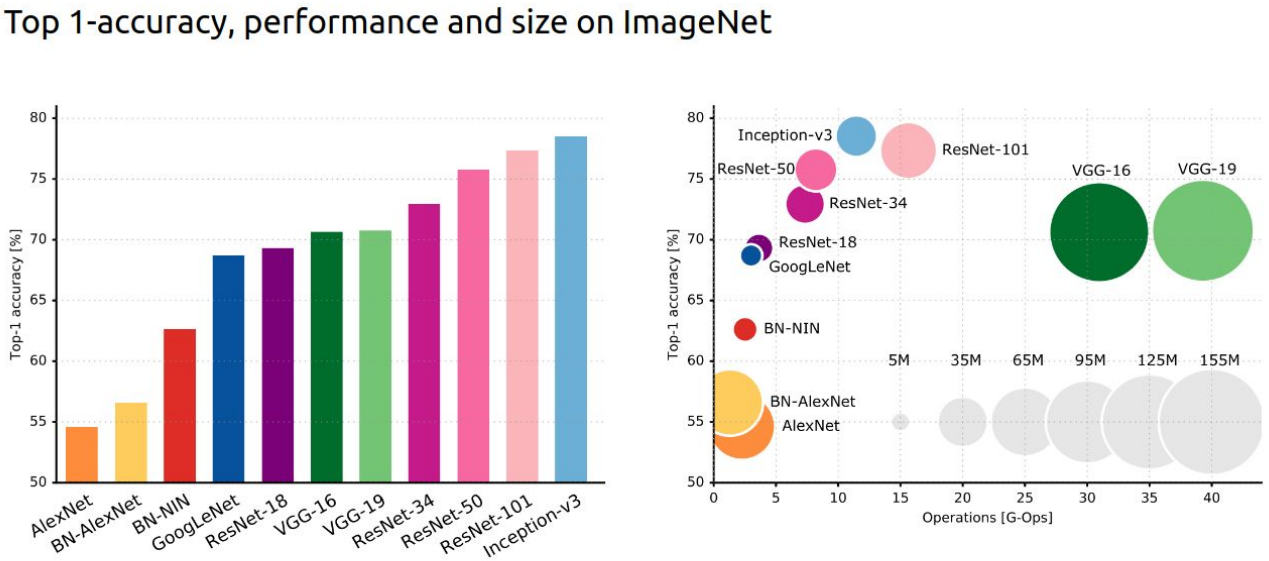


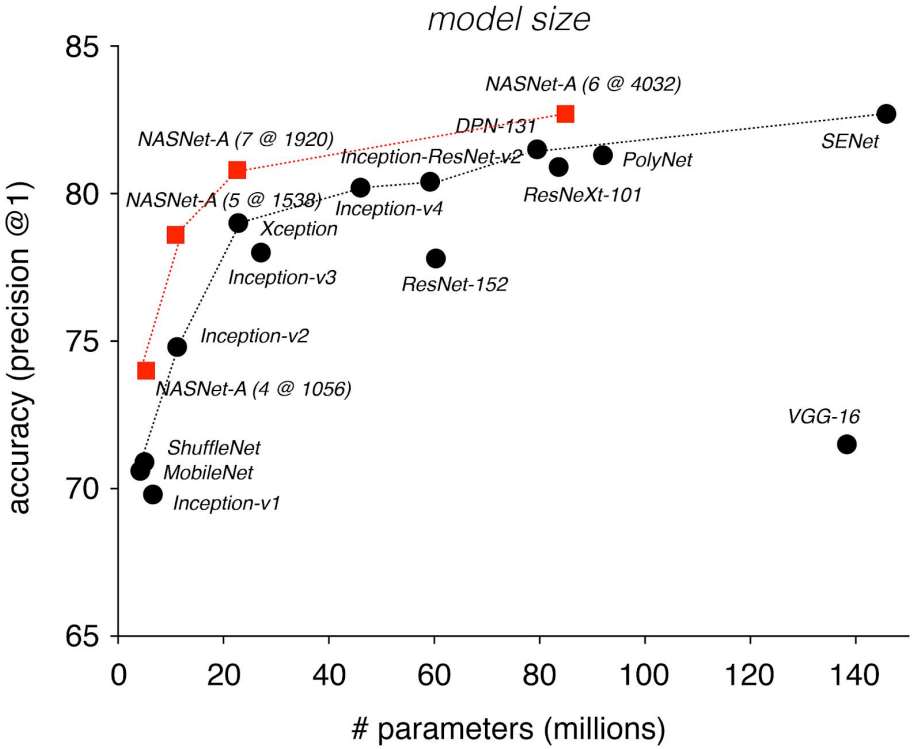
Figure source: http://www.rt-rk.uns.ac.rs/sites/default/files/materijali/predavanja/DL_5.pdf

ImageNet classification



Canziani, Paszke, and Culurciello. "An Analysis of Deep Neural Network Models for Practical Applications." (May 2016).

Learned architectures surpass human-invented



Red: architectures found by Neural Architecture Search

Black: Human-designed neural network architectures

Learning Transferable Architectures for Scalable Image Recognition
B Zoph, V Vasudevan, J Shlens, Q Le (2017)

Outline

- Optimization
- CNN basics
- Examples of CNN Architectures
- Applications of CNN

Pre-trained models

- Training a model on ImageNet from scratch takes **days or weeks**.
- Many models trained on ImageNet and their weights are publicly available!

Transfer learning

- Use pre-trained weights, remove last layers to compute representations of images
- Train a classification model from these features on a new classification task
- The network is used as a generic feature extractor
- Better than handcrafted feature extraction on natural images

Pre-trained models

- Training a model on ImageNet from scratch takes **days or weeks**.
- Many models trained on ImageNet and their weights are publicly available!

Fine-tuning

Retraining the (some) parameters of the network (given enough data)

- Truncate the last layer(s) of the pre-trained network
- Freeze the remaining layers weights
- Add a (linear) classifier on top and train it for a few epochs
- Then fine-tune the whole network or the few deepest layers
- Use a smaller learning rate when fine tuning

Applications of CNN

- **Case study 1: image classification**
- Case study 2: object detection
- Case study 3: segmentation

Object classification

- Given an input image, classify the object within it
- Input? 
- Output? “Cat”
- Loss? Multi-class cross entropy

Object classification

- Given an input image, classify the object within it
- Input?

- Output? “Cat” (class probability): $\hat{y} = (\hat{p_1}, \hat{p_2}, \cdots, \hat{p_C})$
- Loss? $\mathcal{L}(y, \hat{y}) = -y \log(\hat{y}) = -\sum_{c=1}^{C} p_c \log(\hat{p_c})$

Object classification

- Imagenet dataset
- Very large dataset of many object classes
- Enable learning models capable to identify many object features

Object Classification: Deeper is better

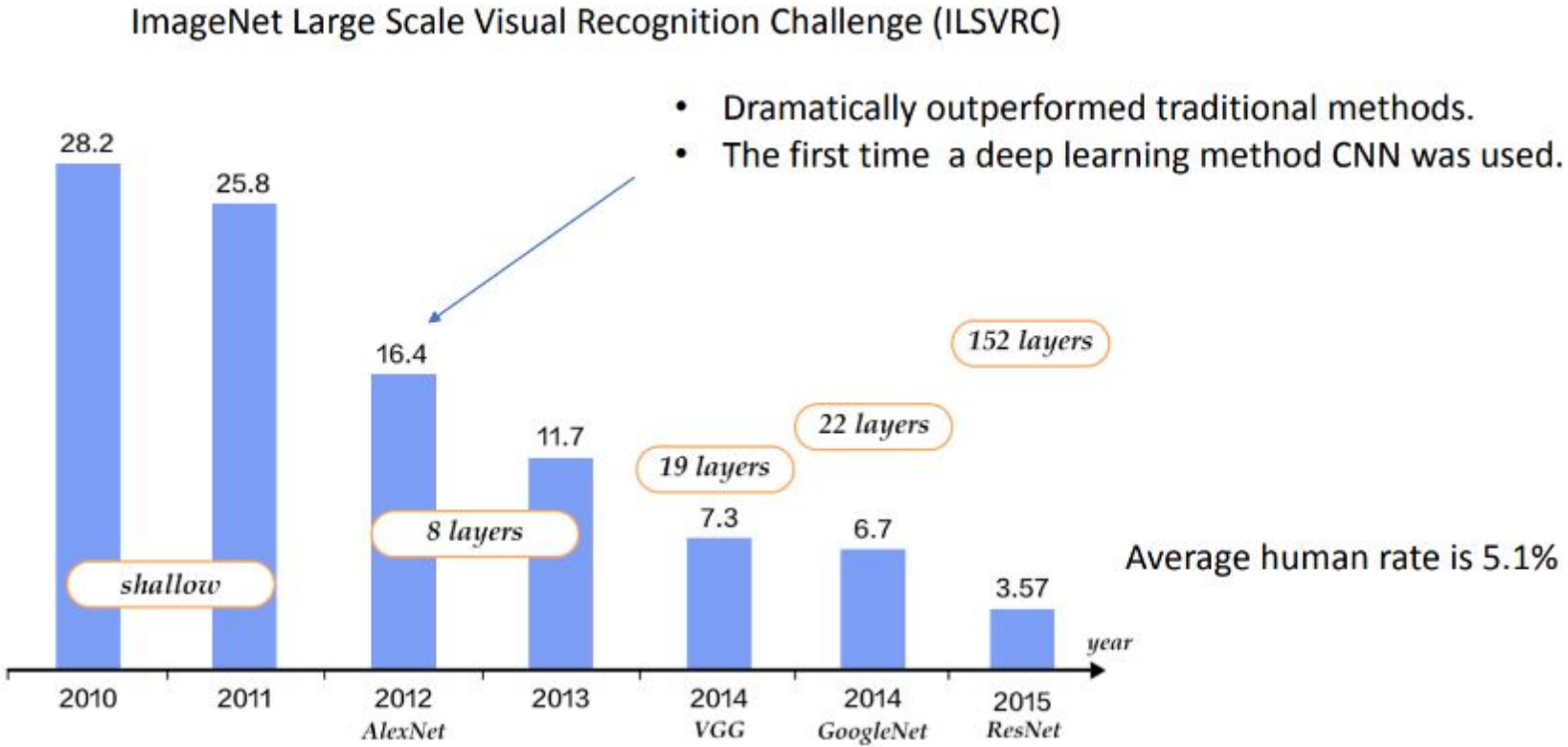
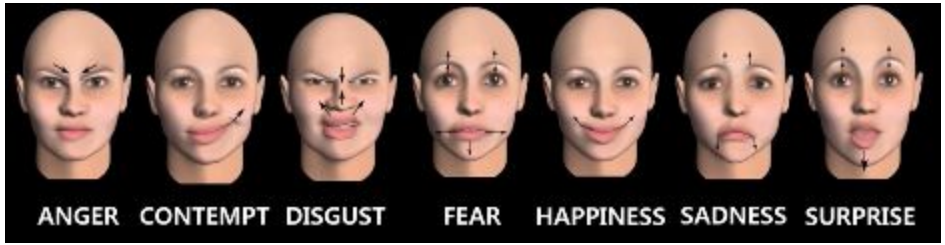


Figure source: http://www.rt-rk.uns.ac.rs/sites/default/files/materijali/predavanja/DL_5.pdf

Emotion Classification

- Given an input human face, label the emotion
- Input?

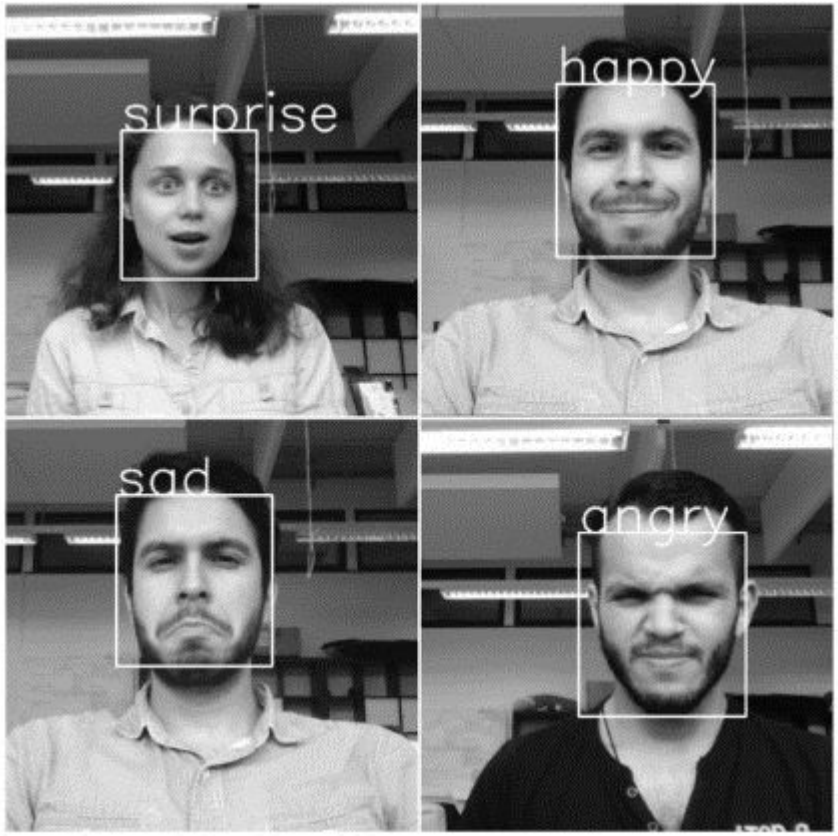


Examples of emotion categories

- Output? “fear”
- Loss? Multi-class cross entropy

Figure from: https://www.youtube.com/watch?v=wlaR5F30hiU

Emotion Classification



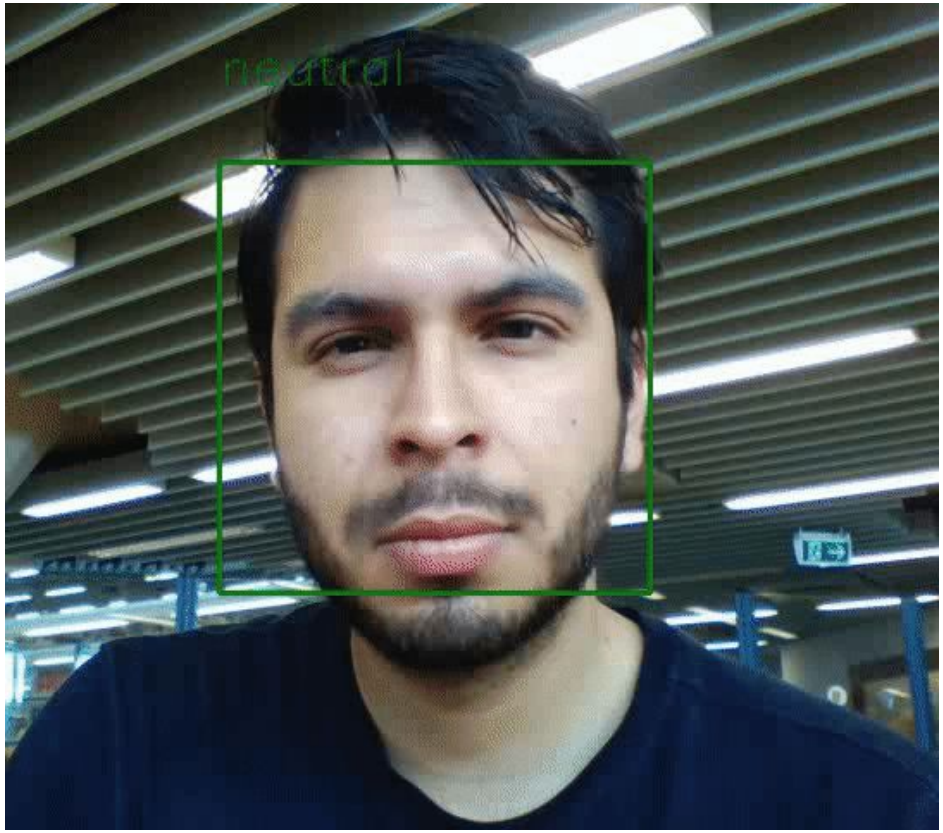


Figure credit: https://github.com/oarriaga/face_classification

Emotion Classification

- Pipeline:
  - Input preprocessing (alignment, normalization, etc)
  - Given the preprocessed data, pass it to a CNN and get output
  - Output label with highest probability is the assigned emotion

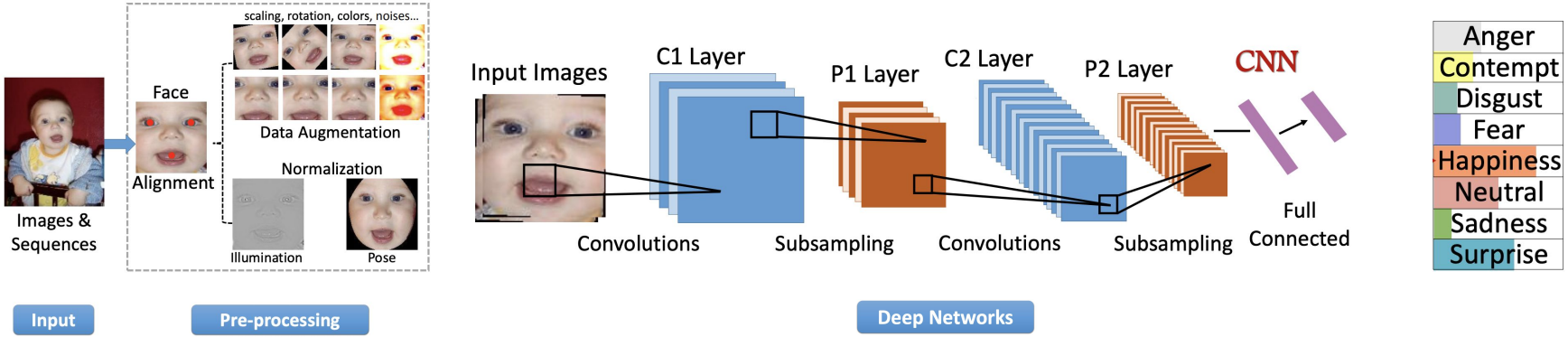


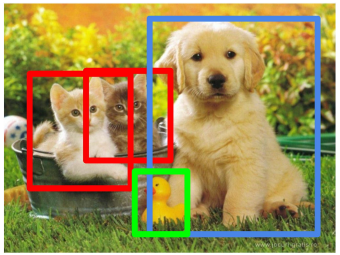
Figure credit: Shan Li and Weihong Deng. Deep Facial Expression Recognition: A Survey.

Applications of CNN

- Case study 1: Image Classification
- **Case study 2: Object Detection**
- Case study 3: Segmentation

Object Detection

- Given an image, detect all objects and classify
- Input?
- Output?

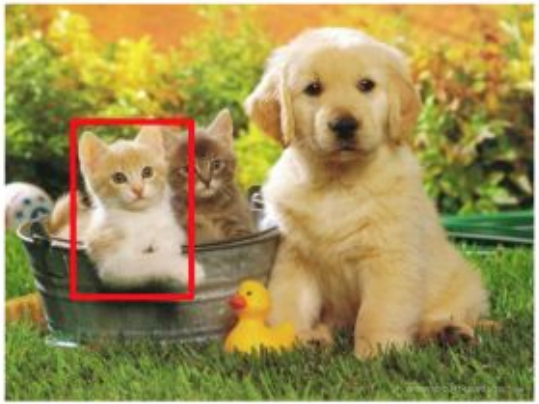


CAT, DOG, DUCK

- Loss?

Error in bounding box position?
Bounding box scale? Class label?

Object Detection



Classes = [cat, dog, duck]

Cat ? YES

Dog ? NO

Duck? NO

Slide credit: Amaia Salvador

Object Detection

Classes = [cat, dog, duck]

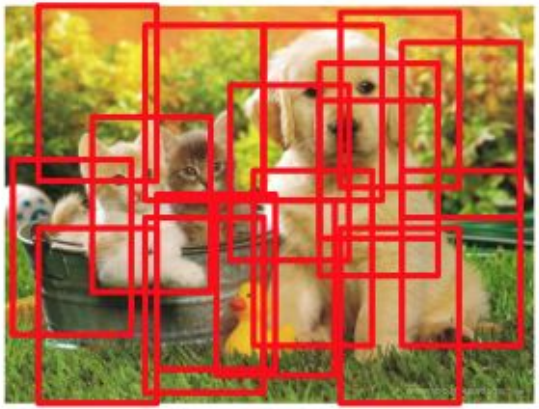
Cat ? NO

Dog ? NO

Duck? NO

Slide credit: Amaia Salvador

Object Detection

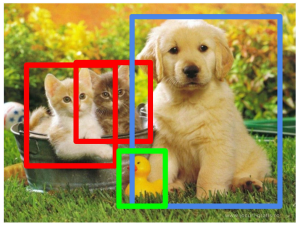


Problem:
Too many positions & scales to test

Solution: If your classifier is fast enough, go for it

Slide credit: Amaia Salvador

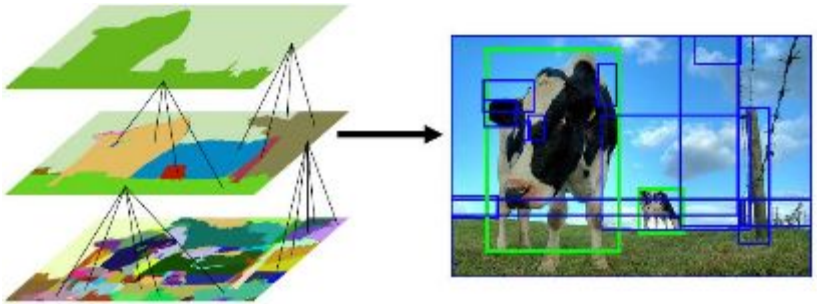
Object Detection



- Two sub-problems:
  - Object locations/size: localize each object with a tight bounding box in the image
  - Object classification: classify the object in each bounding box
- Typically solved by **“proposing candidate bounding boxes”** and **“classifying them”**
- Two main families:
  - A grid in the image where each cell is a proposal (YOLO)
  - Region proposal (Fast RCNN, Faster RCNN)

Bounding Box Proposals

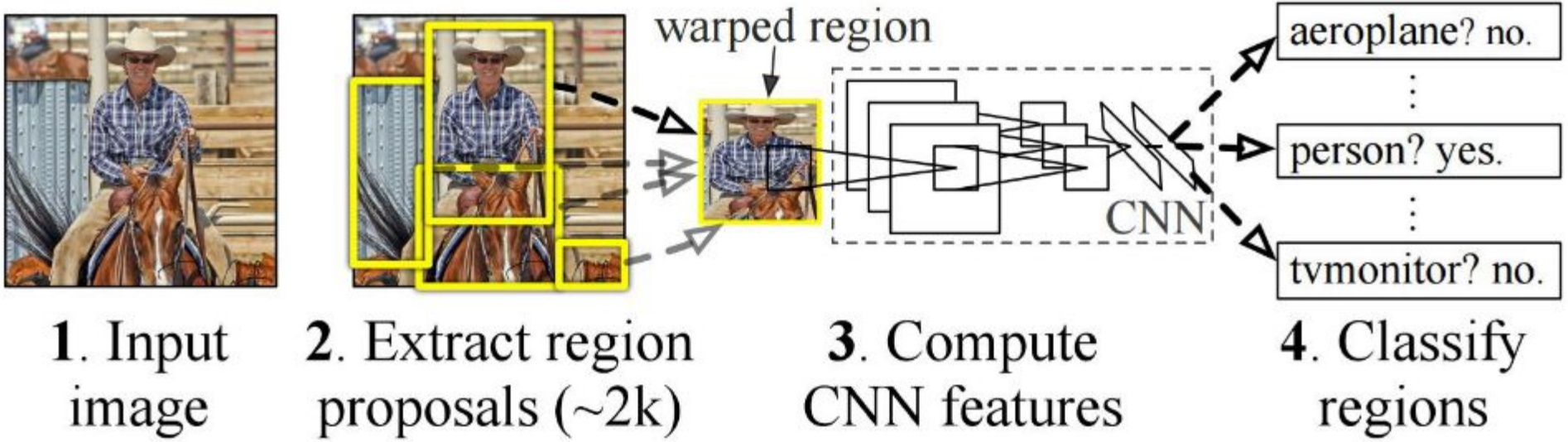
- Instead of having a predefined set of box proposals, find them on the image:
  - Selective Search (from pixels, not learnt)
  - Region Proposal Network (RPN, learnt)



Uijlings, J. R., Van De Sande, K. E., Gevers, T., & Smeulders, A. W. (2013). Selective search for object recognition. *IJCV*, *104*(2), 154-171.

Region-CNN (RCNN)

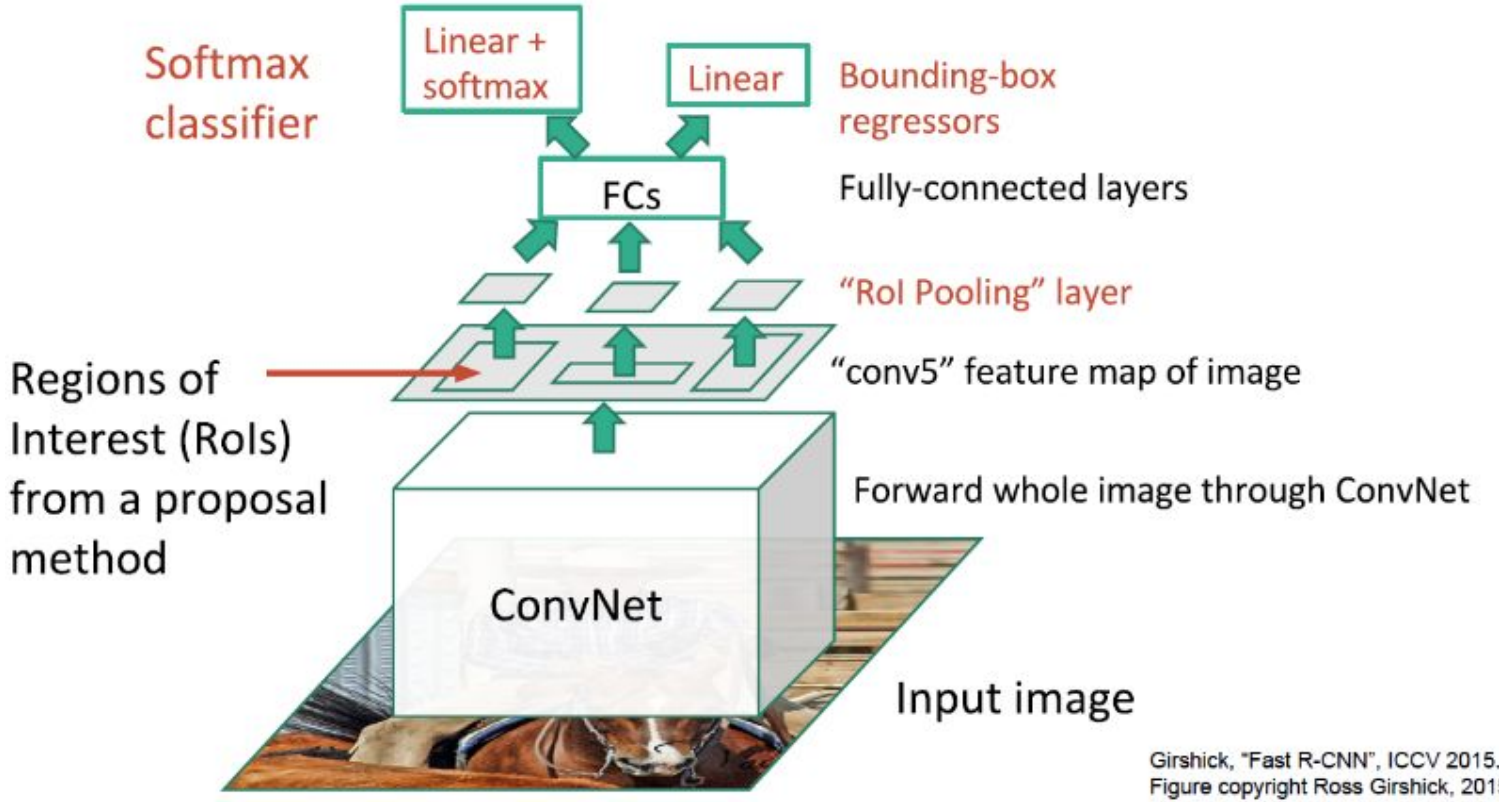
For each region proposal, use CNN to predict the categories (with confidence)



Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 580-587).

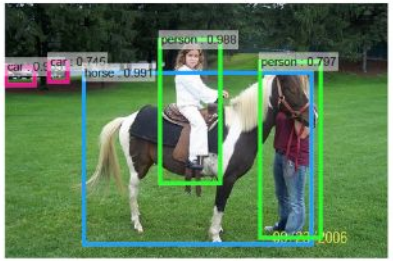
Fast RCNN

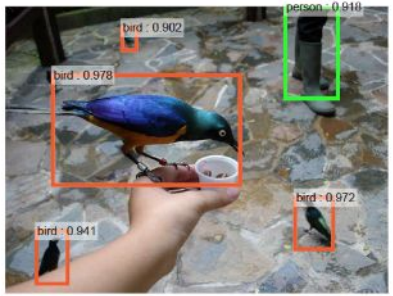
Main idea: calculate convolutional feature map only once and reuse it!



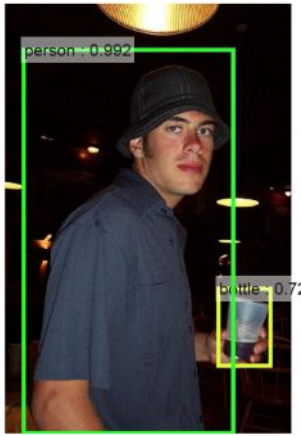
Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015;

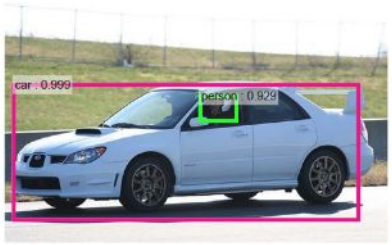
Faster RCNN

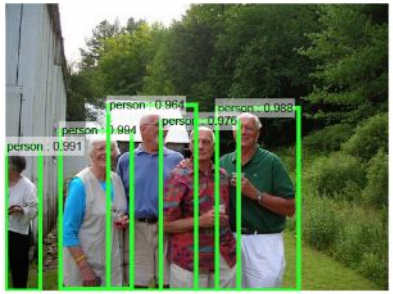

car : 0.746
horse : 0.991
person : 0.988
person : 0.797

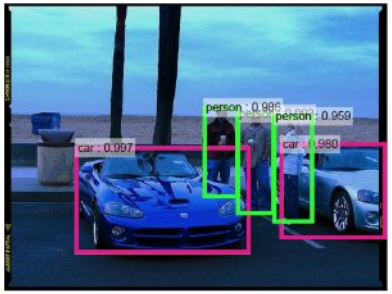

bird : 0.902
person : 0.818
bird : 0.978
bird : 0.972
bird : 0.941

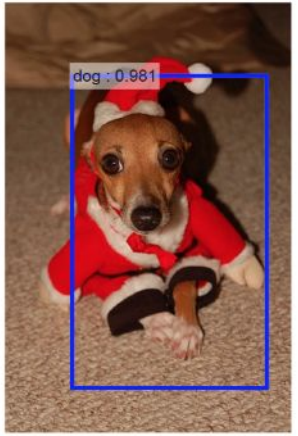

cow : 0.995
cow : 0.998


person : 0.992

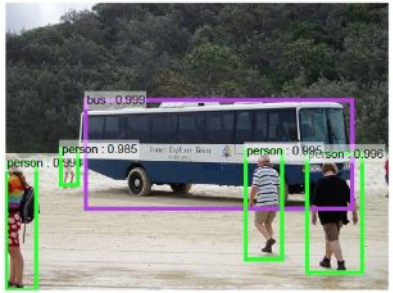

car : 0.999
person : 0.929

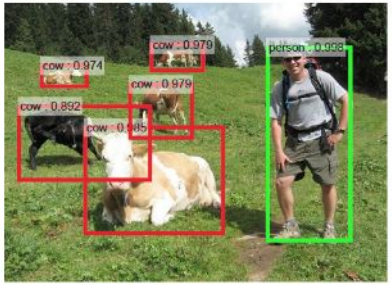

person : 0.964
person : 0.994
person : 0.988
person : 0.976
person : 0.991


person : 0.986
person : 0.959
car : 0.997
car : 0.980


dog : 0.981


person : 0.961
person : 0.960
person : 0.757
dog : 0.697


bus : 0.999
person : 0.985
person : 0.996


cow : 0.979
cow : 0.974
cow : 0.979
cow : 0.892
cow : 0.985

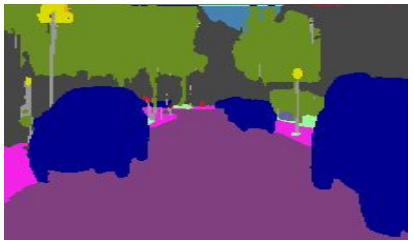
Applications of CNN

- Case study 1: Image Classification
- Case study 2: Object Detection
- **Case study 3: Segmentation**

Semantic Segmentation

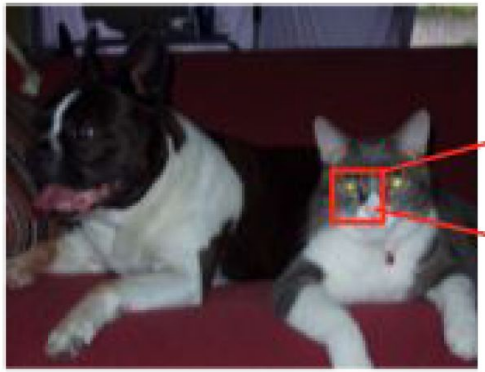
- Given an image, partition the image into coherent parts
- Input?

- Output? Pixel-level class map

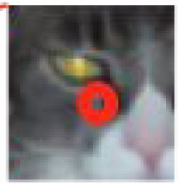


- Loss? Pixel level classification loss

From Classification to Semantic Segmentation



extract a patch



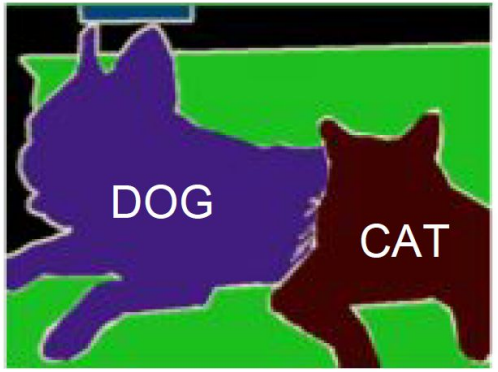
run through a CNN trained for image classification

CNN

classify the center pixel

CAT

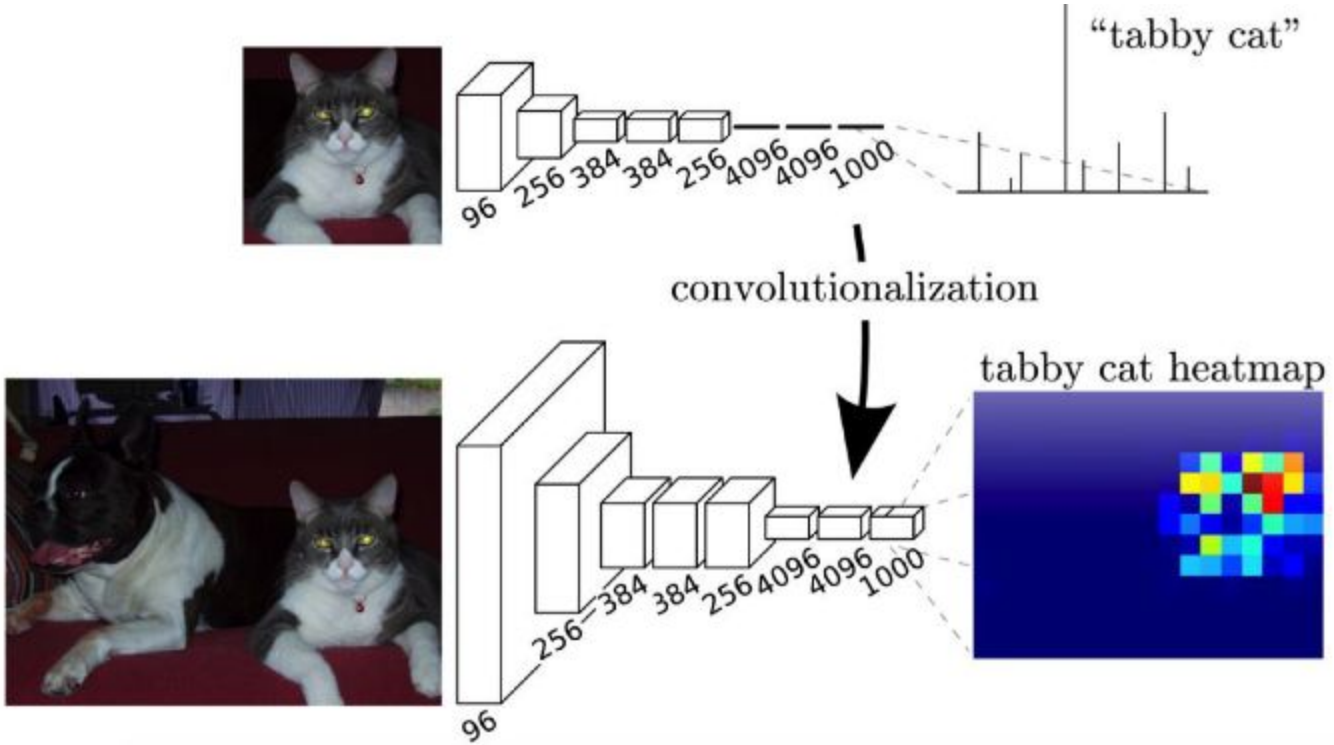
repeat for every pixel



Long, Jonathan, et al. "Fully convolutional networks for semantic segmentation." CVPR 2015

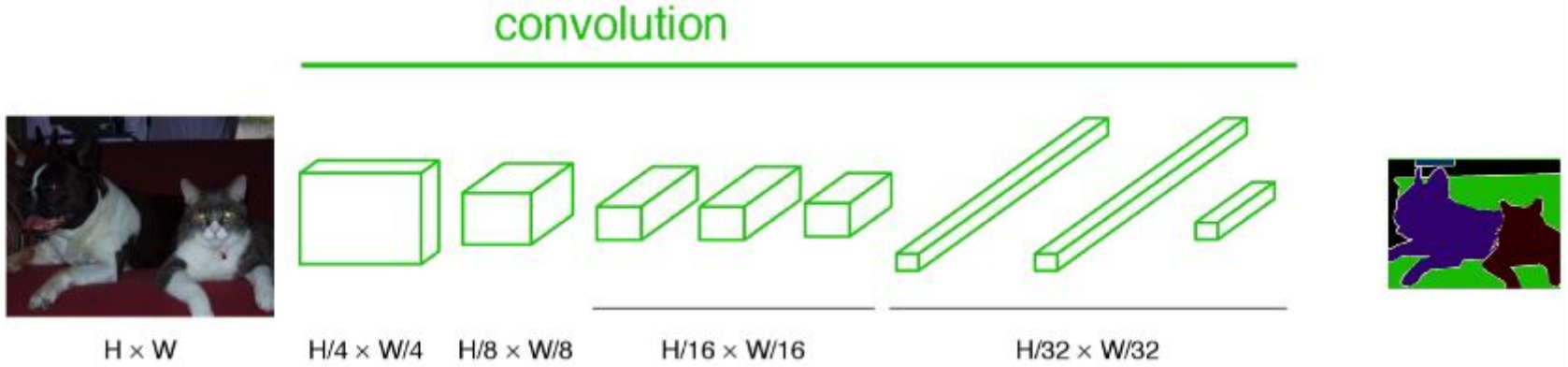
From Classification to Semantic Segmentation

- A classification network becoming fully convolutional



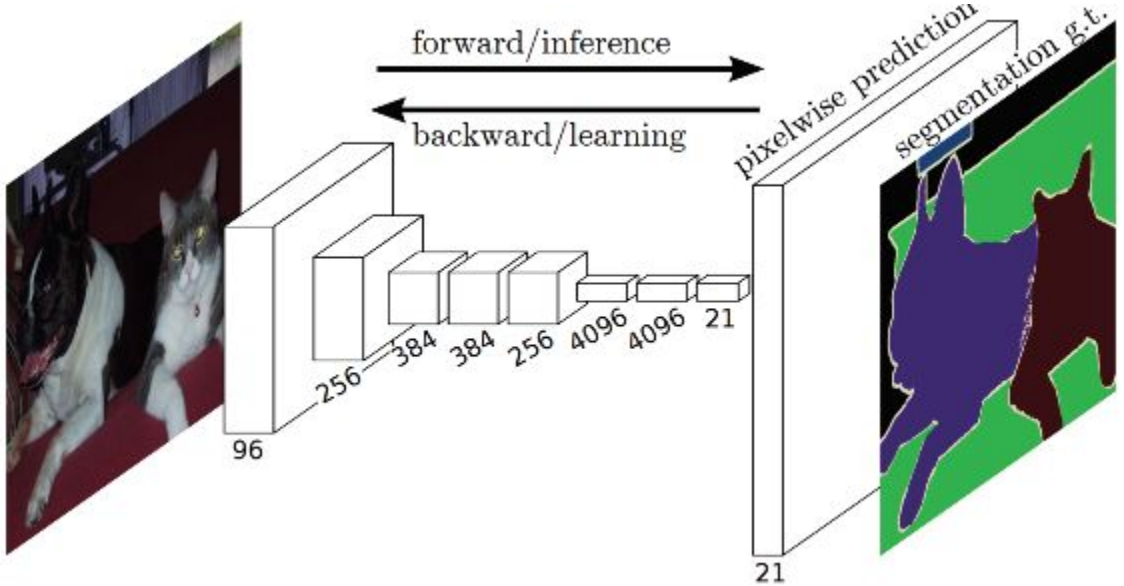
From Classification to Semantic Segmentation

- Dense prediction: fully convolutional, end to end, pixel-to-pixel network
- Problems
  - Output is smaller than input → Add upsampling layers



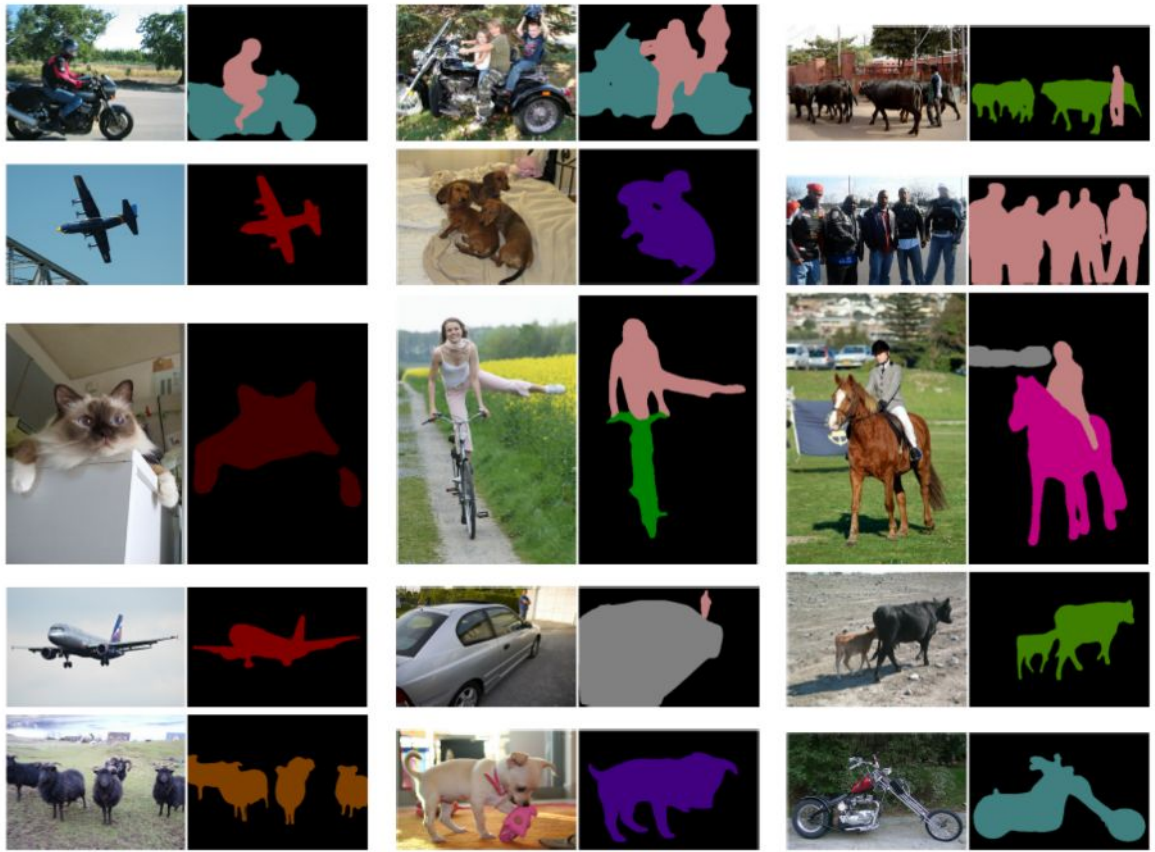
From Classification to Semantic Segmentation

- Dense prediction: fully convolutional, end to end, pixel-to-pixel network
- Learnable upsampling



Long, Jonathan, et al. "Fully convolutional networks for semantic segmentation." CVPR 2015

Example results



Chen, L. C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2018). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, *40*(4), 834-848.

Any feedback (about lecture, slide, homework, project, etc.)?

(via **anonymous** google form: https://forms.gle/fpYmiBtG9Me5qbP37)

Change Log of lecture slides:
https://docs.google.com/document/d/e/2PACX-1vSSIHJjkIypK7rKFSR1-5GYXyBCEW8UPtpSfCR9AR6M1l7K9ZQEmxfFwaWaW7kLDxusthsF8WlCyZJ-/pub