

EECS 545: Machine Learning

Lecture 12: Optimization for Neural Networks, Convolutional Neural Network

Honglak Lee and Michał Dereziński

02/16/2022



Logistics

- Project progress report due 03/08/22 – the week after spring break
 - Start now!
 - See below for things to think about if you haven't started thinking about them already
- See [project information document](#) for detailed info – summary on next slide

Additional tips: Recommended content in the report

- Dataset and/or data collection
- Pre-processing
- Feature representation
- Pipeline of the method
- Initial results
 - Tables
 - Figures
 - Visualization of results/models
- Hyperparameters and how they were tuned
 - make it clear how you performed cross validation (hold-out CV, or K-fold CV, etc.)
 - For hold-out CV, specify whether you separated training, validation, and test set
 - For K-fold CV, specify K value.
- Error analysis
- Summary and future work

In your project progress report, please include the following:

- Project Title
 - Please put a specific and informative title (that describes what your project is about)
 - For example, "545 Project Progress Report" is not a specific and informative title
- Team members (* Please put these in the beginning of your report)
 - Please specify the full name and email addresses in uniqueusername@umich.edu
 - You can specify the names and emails as the following format
 - author1, author2, author3, ..., authorN
 - {email1, email2, ..., emailN}@umich.edu
- Abstract (1 paragraph)
- Section 1. Introduction:
 - Problem description and motivation.
 - Why do you want to solve this problem?
 - What's the impact if you can solve this problem?
- Section 2. Proposed method:
 - How are you going to solve this problem?
 - Why should the proposed method work?
 - Provide explanation/rationale of why you chose this specific method
 - Provide technical details of your approach if you are proposing a novel method.
 - Description of the pipeline. Including figure (e.g., block diagram) that explains the pipeline will be very helpful
- Section 3. Related work:
 - What are existing methods?
 - What are the state-of-the-art methods for this problem?
 - How is your approach similar to other existing work?
 - How is your approach different from the related work?
- Section 4. Experimental results:
 - Milestones achieved so far (add all relevant experimental results).
 - How do these results support your claim?
- Section 5. Future milestones: Dates and sub-goals (please set sub-goals on a weekly basis so that they can be done in a week)
- Section 6. Conclusion: Summary of your progress and your final expected goal (what do you expect to achieve or demonstrate for the final project?)
- Author Contributions
 - Please see the guideline "Contribution of each team member" below
- References

Outline

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

Optimization

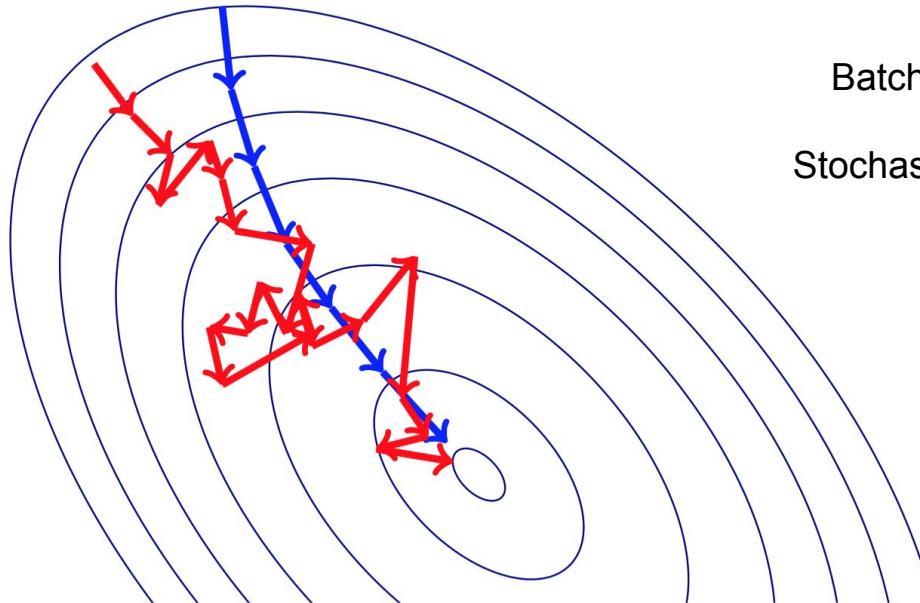
- Stochastic Gradient Descent
- Momentum Method
- Adaptive Learning Methods
(AdaGrad, RMSProp, Adam)



Broadly applicable for
many ML methods

- Batch Normalization ← specific to neural nets

Stochastic Gradient Descent



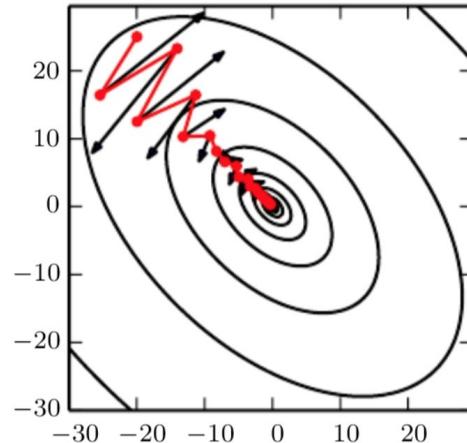
Batch gradient descent
vs
Stochastic gradient descent

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)

Momentum

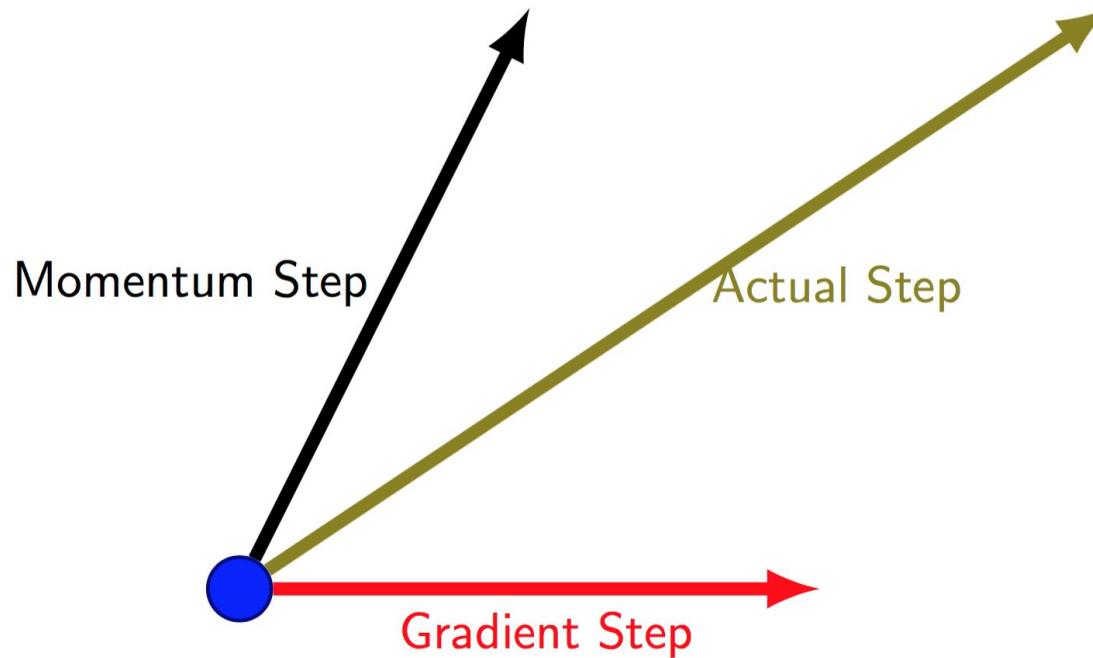
- The Momentum method is a method to accelerate learning using GD or SGD



- Illustration of how momentum traverses such an error surface better compared to gradient descent
- [Visualization of GD with momentum](#)

Momentum

$$\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_{\theta} \left(L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)}) \right)$$



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

$$\begin{aligned}\mathbf{s} &\leftarrow \rho_1 \mathbf{s} + (1 - \rho_1) \hat{\mathbf{g}} \\ \mathbf{r} &\leftarrow \rho_2 \mathbf{r} + (1 - \rho_2) \hat{\mathbf{g}} \odot \hat{\mathbf{g}}\end{aligned}$$
$$\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \rho_1^t}, \hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1 - \rho_2^t} \text{ then } \Delta\theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta} \text{ then } \theta \leftarrow \theta + \Delta\theta$$

- Visualization:

- [Visualization of optimizers](#)

- [Distill: why momentum really works](#)

Slide credit: Shubhendu Trivedi, Risi Kondor

Batch Normalization: motivation

- **Specific to neural networks**
- We have a recipe to compute gradients (backpropagation) and update every parameter (GD, SGD, adaptive learning rate methods, etc.)
- During backpropagation, we implicitly assume other layers don't change
 - This isn't actually the case, because we update all layers simultaneously
 - Can lead to unstable training \Rightarrow longer time to convergence

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 $\mu_{\mathcal{B}}$ and standard deviation $\sigma_{\mathcal{B}}^2$, we can normalize each input x_i to

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad (\epsilon \text{ added for numerical stability})$$

- which shifts our input to a distribution w/ mean = 0 and std dev = 1
- 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: $\mathcal{B} = \{x_1 \dots m\}$;
Parameters to be learned: γ, β

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

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

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

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

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

Batch Normalization: Backpropagation

- Differentiable via chain rule

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

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

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

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

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

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

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

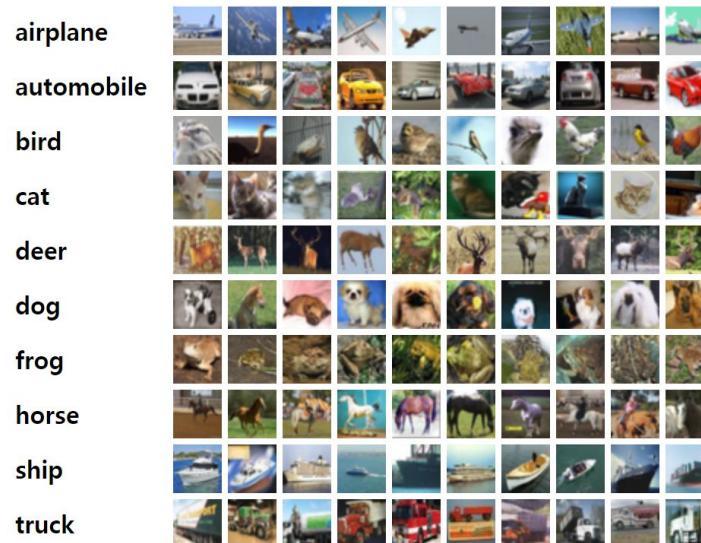


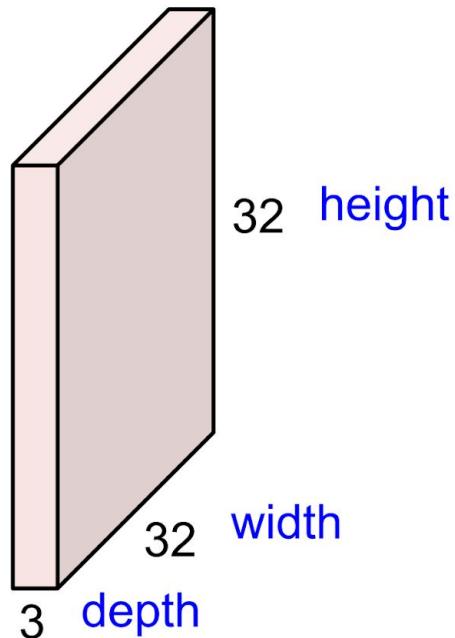
Image Classification



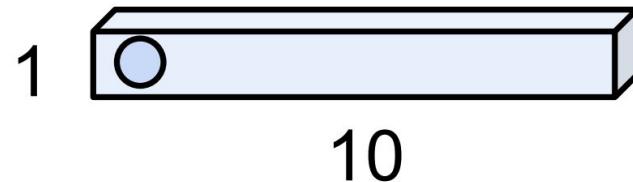
“airplane”

Image Classification

32x32x3 image

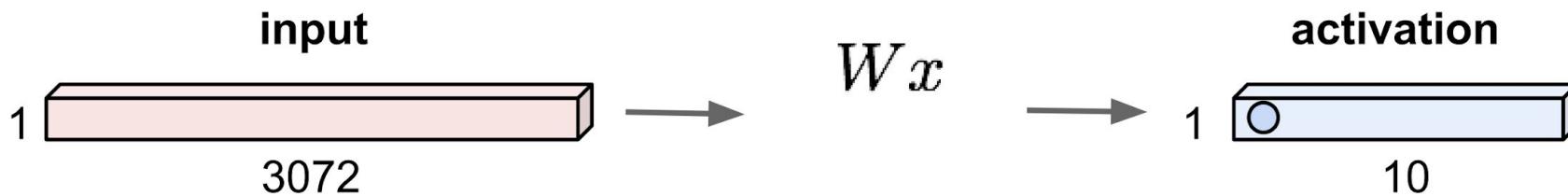


10-class output



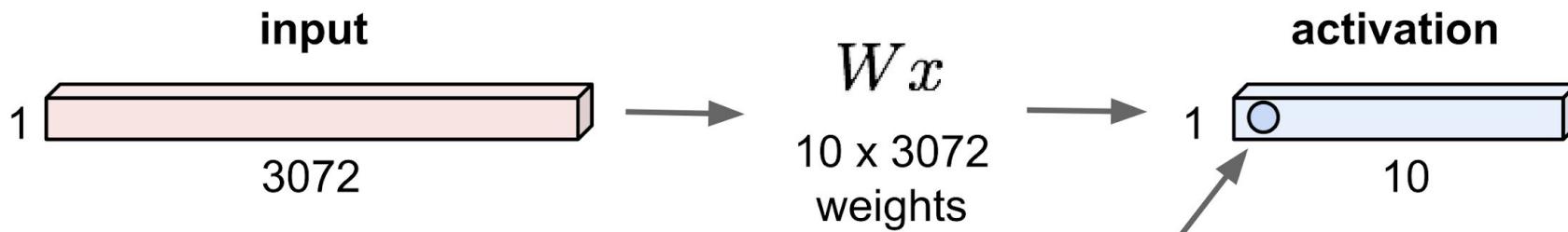
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Fully Connected Layer

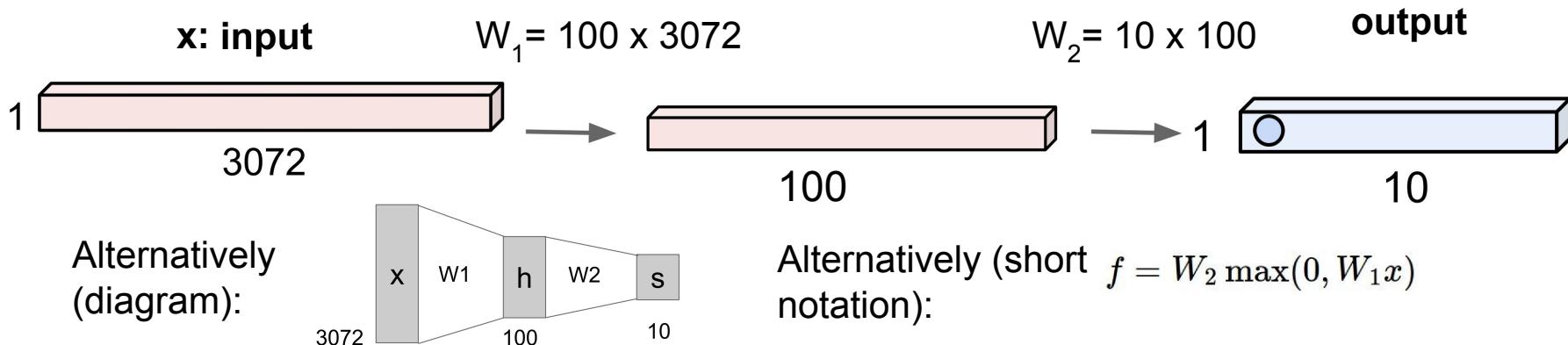
32x32x3 image -> stretch to 3072 x 1



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 $w_k \in \mathbb{R}^{3072}$ for the class k.

Neural Network with 1 hidden layer (with ReLu)

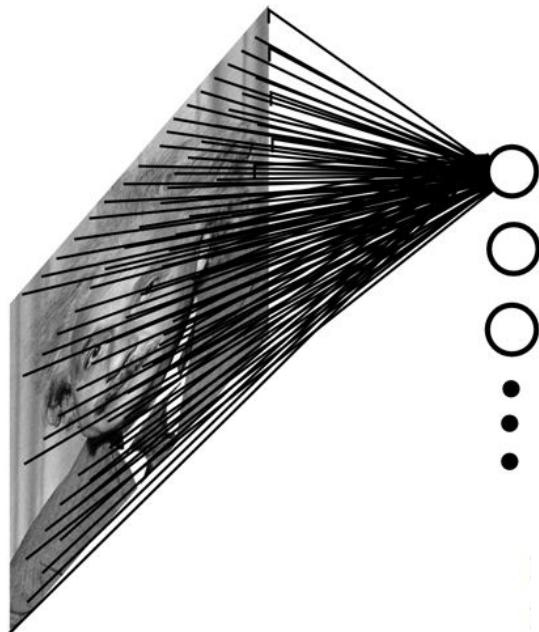


Deeper models: We can add more layers, etc.

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

Scaling to larger-sized images

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

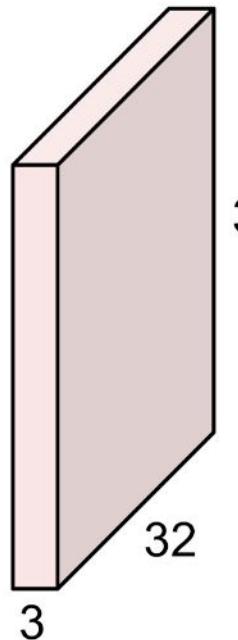


Motivation: Convolutional Neural Networks

- How can we design neural networks that are specifically adapted for classifying large-sized images?
 - Must deal with very **high-dimensional inputs**: 150×150 pixels = 22500 inputs, or 3×22500 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

Convolution Layer

32x32x3 image



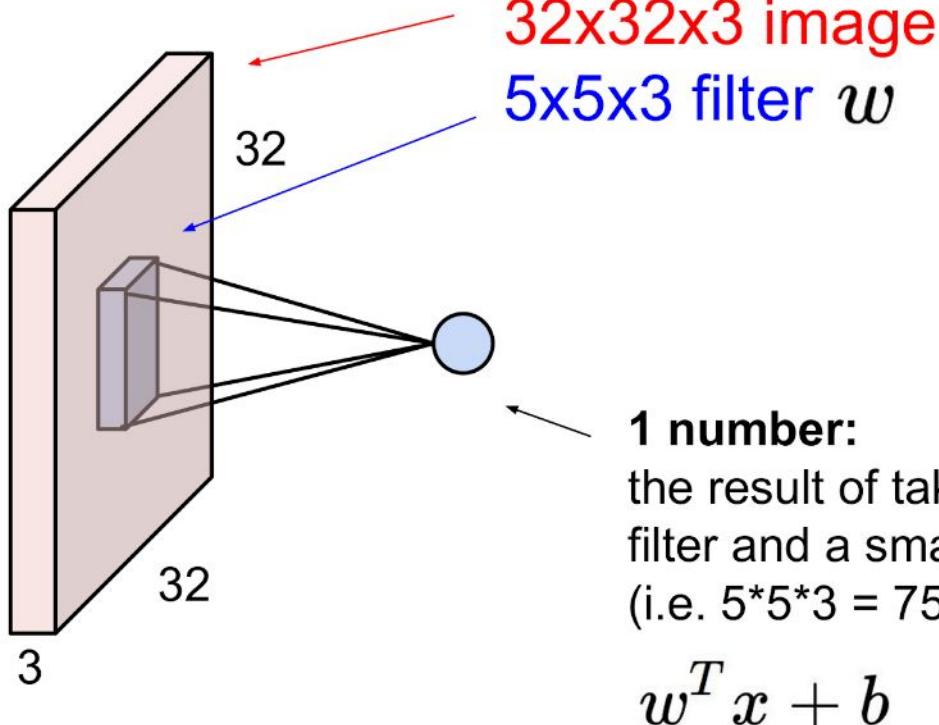
5x5x3 filter



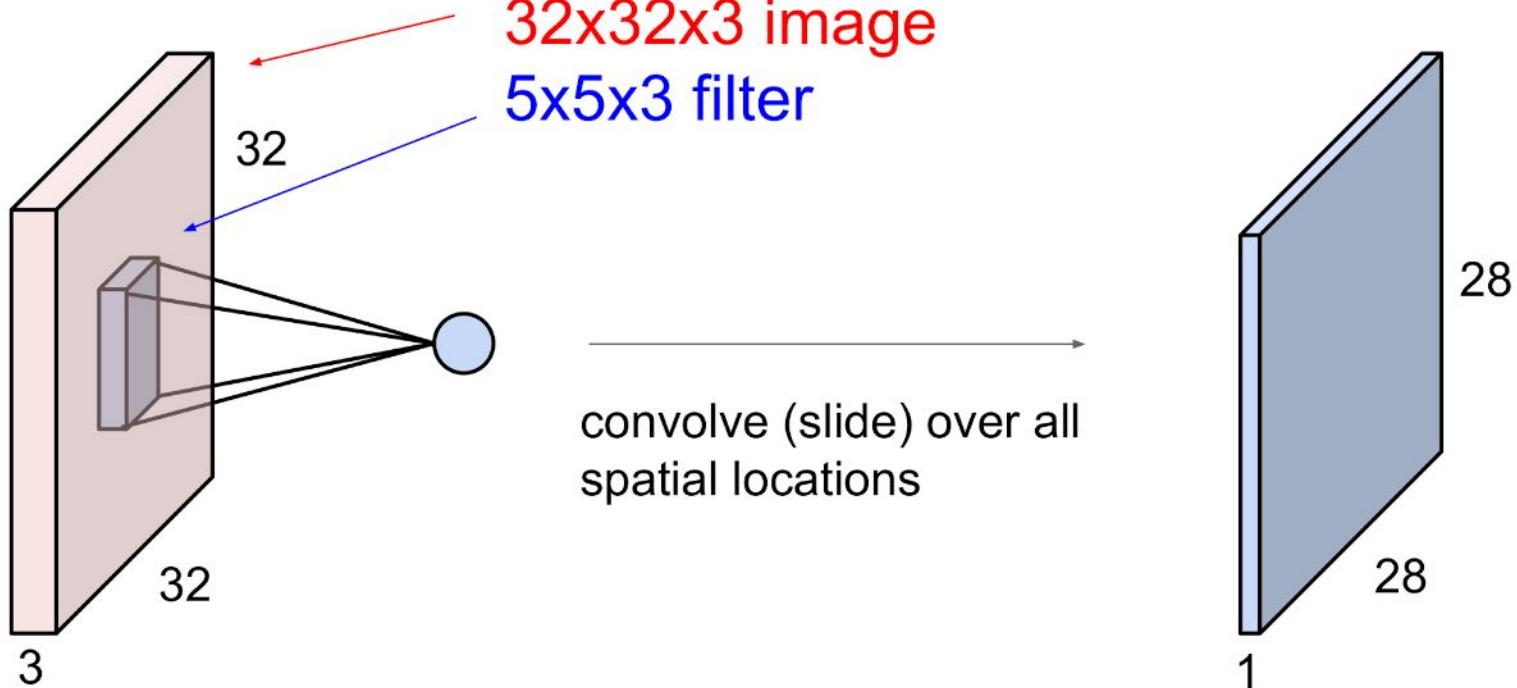
Filters always extend the full depth of the input volume

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

Convolution Layer



Convolution Layer

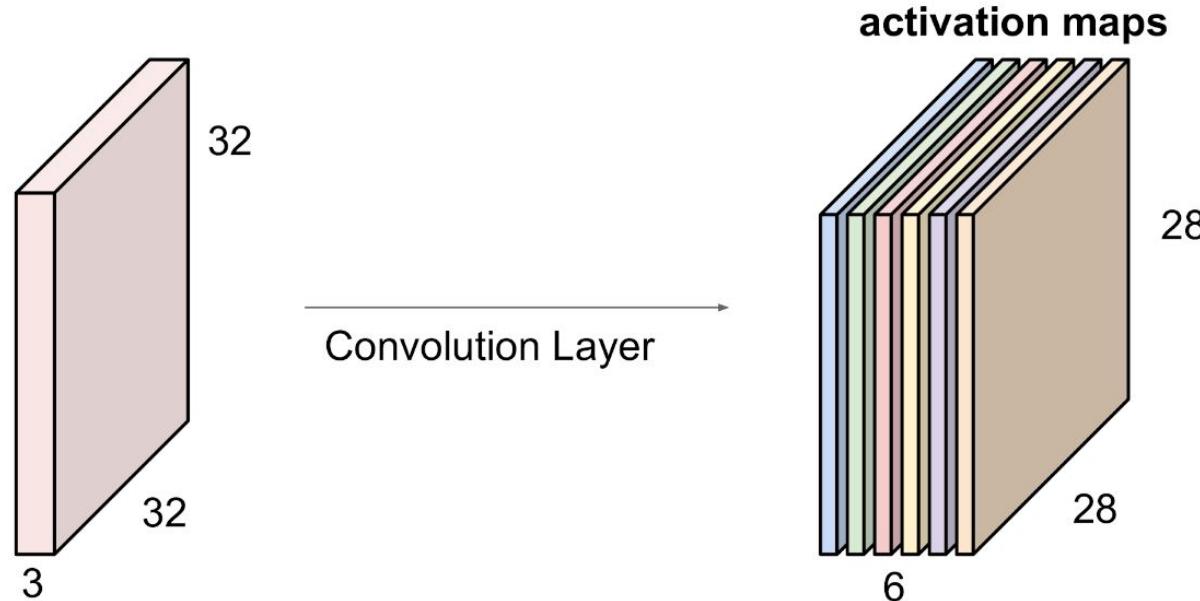


Convolution Layer



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!

Discrete Convolution

- The convolution of an image x with kernel k is computed as follows

$$(\mathbf{a} *_{\text{valid}} \mathbf{b})_{i,j} = \sum_{m=i}^{i+H_b-1} \sum_{n=j}^{j+W_b-1} a_{m,n} b_{i-m+H_b, j-n+W_b}$$

- Example

0	80	40
20	40	0
0	0	40

x

*

0	.25
.5	1

k

=

Slide credit: Russ Salakhutdinov

Discrete Convolution

- The convolution of an image x with kernel k is computed as follows

$$(\mathbf{a} *_{\text{valid}} \mathbf{b})_{i,j} = \sum_{m=i}^{i+H_b-1} \sum_{n=j}^{j+W_b-1} a_{m,n} b_{i-m+H_b, j-n+W_b}$$

- Example

1	.5
.25	0

0	80	40
20	40	0
0	0	40

x

$$\tilde{k} = k \text{ with rows and columns flipped}$$

*

0	.25
.5	1

k

=

Discrete Convolution

- The convolution of an image x with kernel k is computed as follows

$$(\mathbf{a} *_{\text{valid}} \mathbf{b})_{i,j} = \sum_{m=i}^{i+H_b-1} \sum_{n=j}^{j+W_b-1} a_{m,n} b_{i-m+H_b, j-n+W_b}$$

- Example

$$1 \times 0 + 0.5 \times 80 + 0.25 \times 20 + 0 \times 40 = 45$$

1	.5
.25	0

0	80	40
20	40	0
0	0	40

x

*

0	.25
.5	1

k

=

45	

Slide credit: Russ Salakhutdinov

Discrete Convolution

- The convolution of an image x with kernel k is computed as follows

$$(\mathbf{a} *_{\text{valid}} \mathbf{b})_{i,j} = \sum_{m=i}^{i+H_b-1} \sum_{n=j}^{j+W_b-1} a_{m,n} b_{i-m+H_b, j-n+W_b}$$

- Example

$$1 \times 80 + 0.5 \times 40 + 0.25 \times 40 + 0 \times 0 = 110$$

1	.5
.25	0

0	80	40
20	40	0
0	0	40

x

*

0	.25
.5	1

k

=

45	110

Discrete Convolution

- The convolution of an image x with kernel k is computed as follows

$$(\mathbf{a} *_{\text{valid}} \mathbf{b})_{i,j} = \sum_{m=i}^{i+H_b-1} \sum_{n=j}^{j+W_b-1} a_{m,n} b_{i-m+H_b, j-n+W_b}$$

- Example

$$1 \times 20 + 0.5 \times 40 + 0.25 \times 0 + 0 \times 0 = 40$$

1	.5
.25	0

0	80	40
20	40	0
0	0	40

x

*

0	.25
.5	1

k

=

45	110
40	

Discrete Convolution

- The convolution of an image x with kernel k is computed as follows

$$(\mathbf{a} *_{\text{valid}} \mathbf{b})_{i,j} = \sum_{m=i}^{i+H_b-1} \sum_{n=j}^{j+W_b-1} a_{m,n} b_{i-m+H_b, j-n+W_b}$$

- Example

$$1 \times 40 + 0.5 \times 0 + 0.25 \times 0 + 0 \times 40 = 40$$

1	.5
.25	0

0	80	40
20	40	0
0	0	40

x

*

0	.25
.5	1

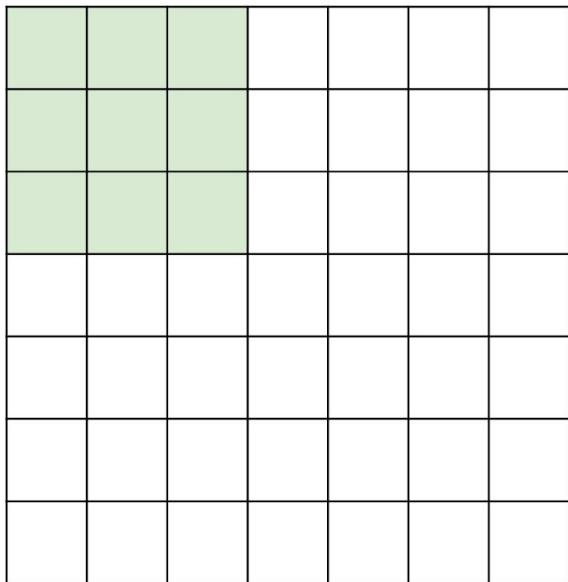
k

=

45	110
40	40

Closer look at spatial dimensions

7

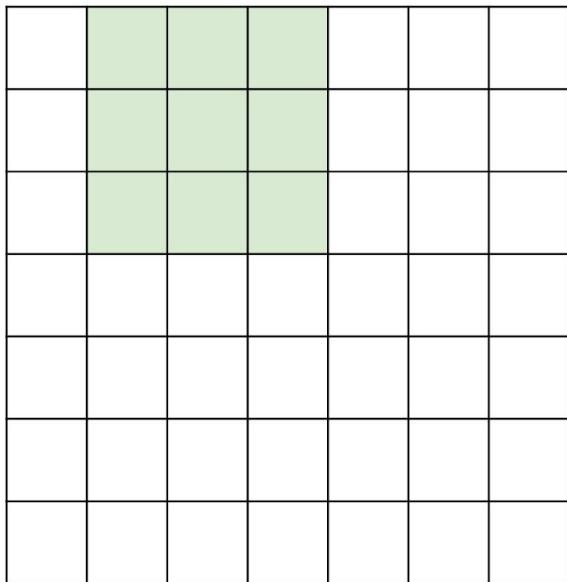


7x7 input (spatially)
assume 3x3 filter

7

Closer look at spatial dimensions

7



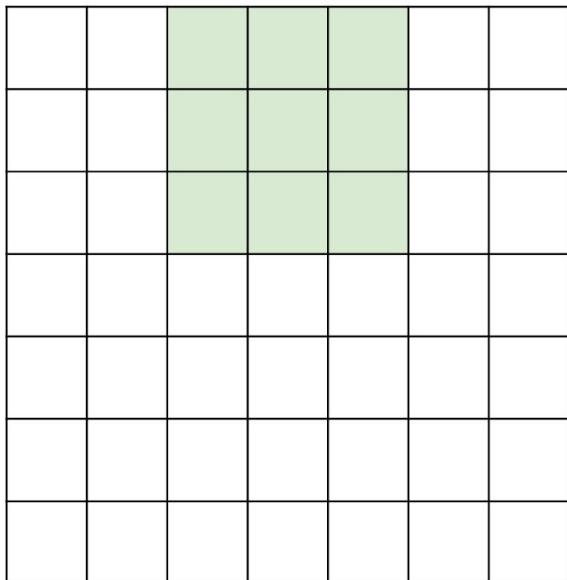
7x7 input (spatially)
assume 3x3 filter

7

Stride is convolutional
“step size”: here, = 1

Closer look at spatial dimensions

7



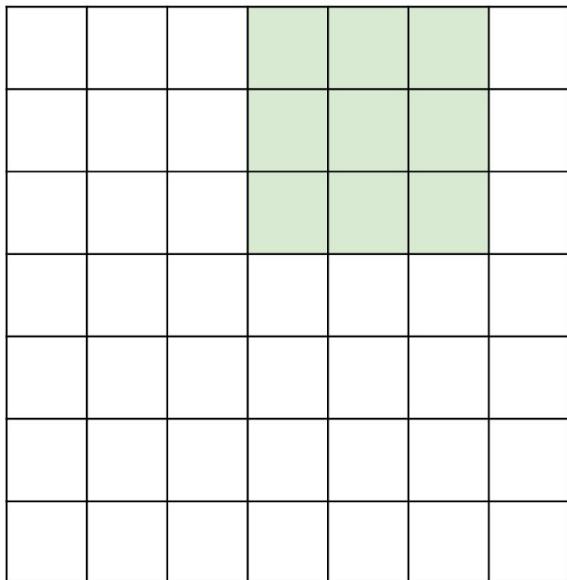
7x7 input (spatially)
assume 3x3 filter

7

Stride is convolutional
“step size”: here, = 1

Closer look at spatial dimensions

7



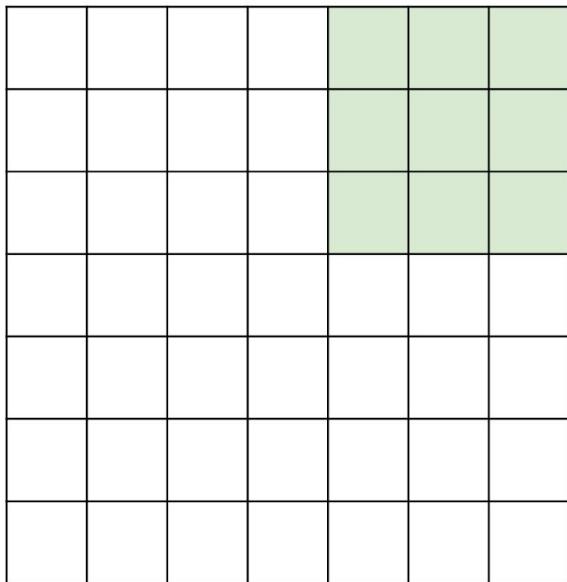
7x7 input (spatially)
assume 3x3 filter

7

Stride is convolutional
“step size”: here, = 1

Closer look at spatial dimensions

7



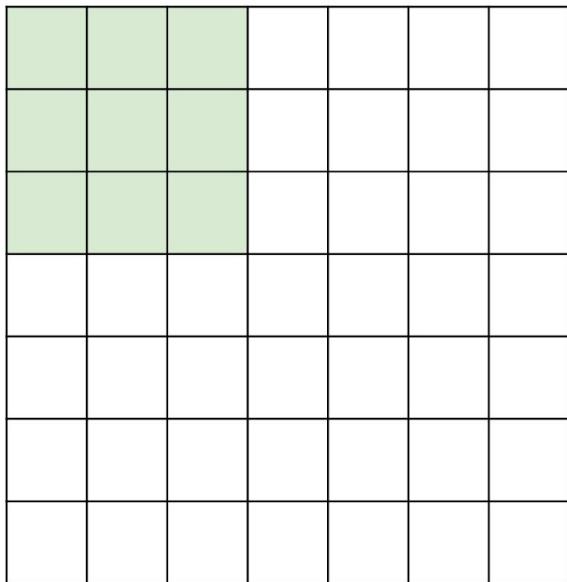
7x7 input (spatially)
assume 3x3 filter

Stride is convolutional
“step size”: here, = 1

=> 5x5 output

Closer look at spatial dimensions

7

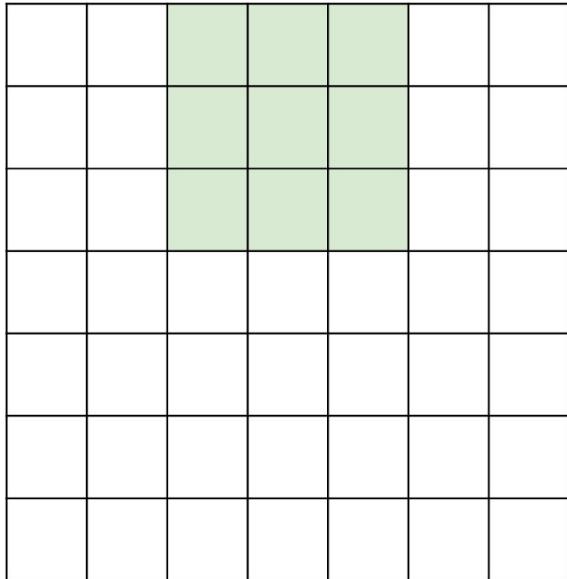


7

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

Closer look at spatial dimensions

7

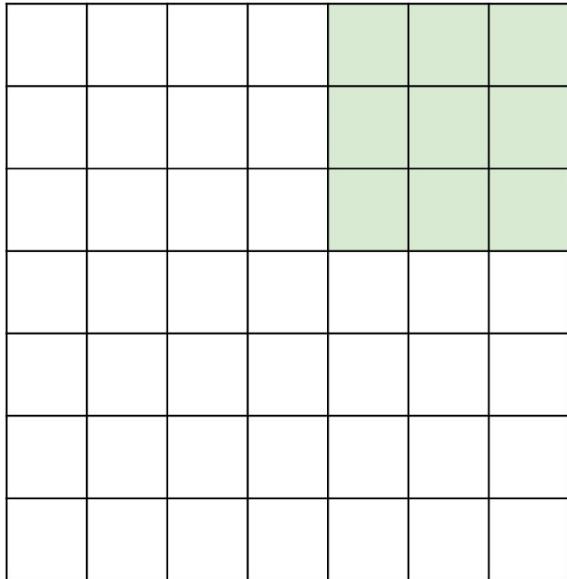


7

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

Closer look at spatial dimensions

7

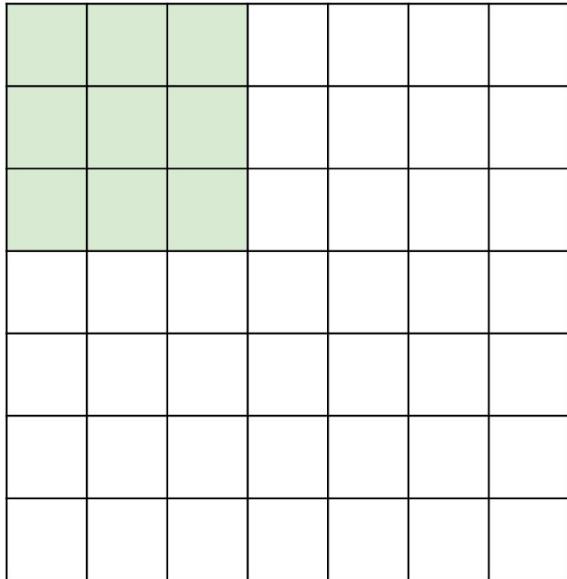


7

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

Closer look at spatial dimensions

7

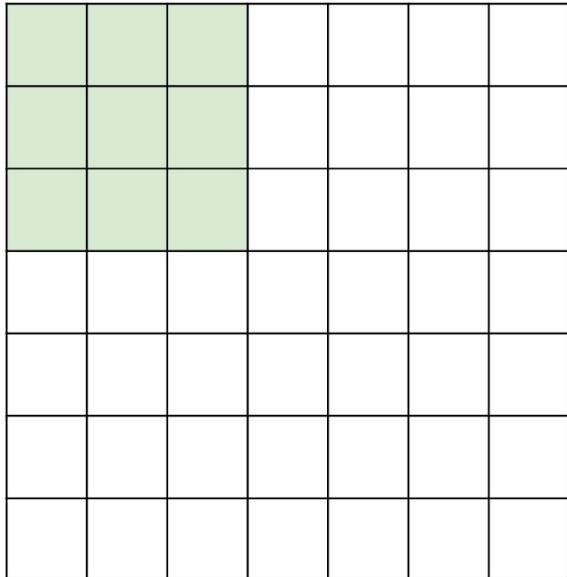


7

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

Closer look at spatial dimensions

7



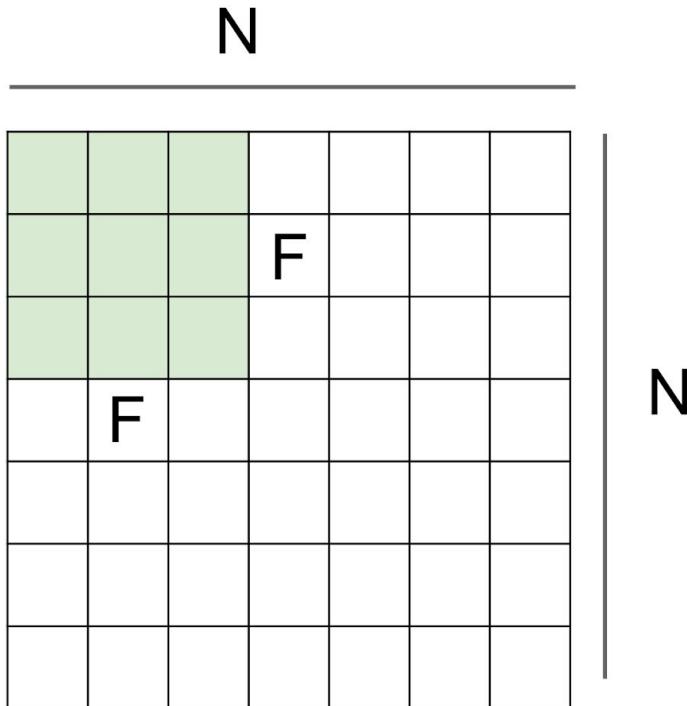
7

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

doesn't fit!

cannot apply 3x3 filter on
7x7 input with stride 3.

Closer look at spatial dimensions



Output size:
 $(N - F) / \text{stride} + 1$

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

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

3x3 filter, applied with stride 1

pad with 1 pixel border => what is the output?

(recall:)

$$(N - F) / \text{stride} + 1$$

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

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. $F = 3 \Rightarrow$ zero pad with 1

$F = 5 \Rightarrow$ zero pad with 2

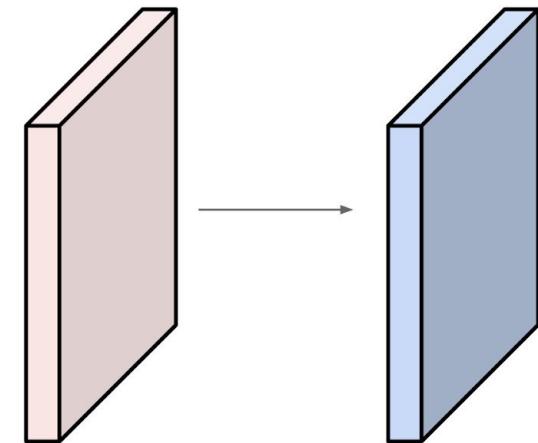
$F = 7 \Rightarrow$ zero pad with 3

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**



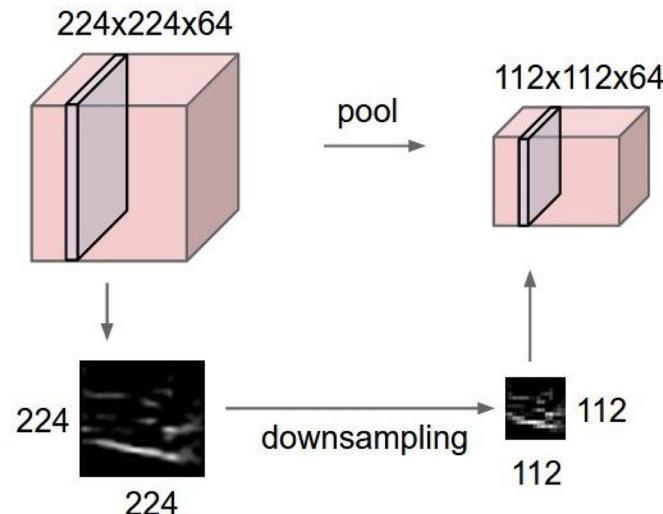
Output volume size:

$(32+2*2-5)/1+1 = 32$ spatially, so

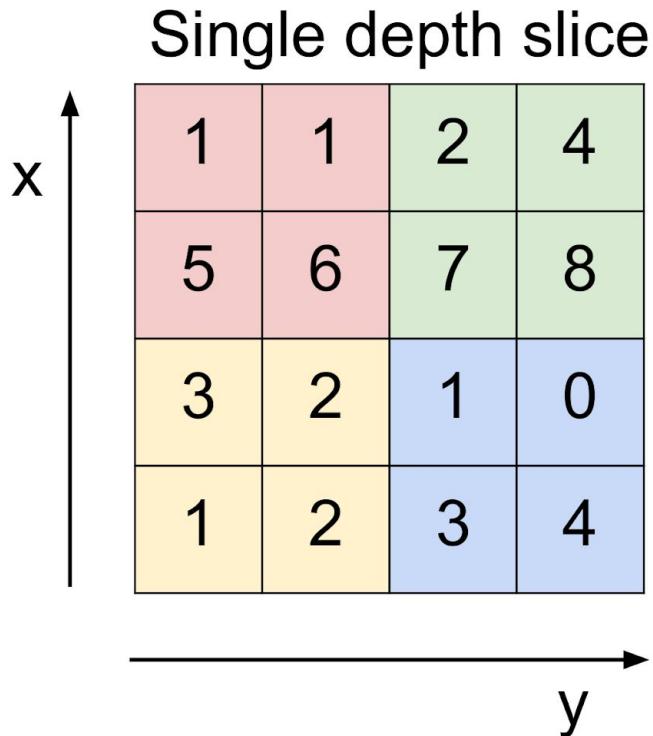
32x32x10

Pooling

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



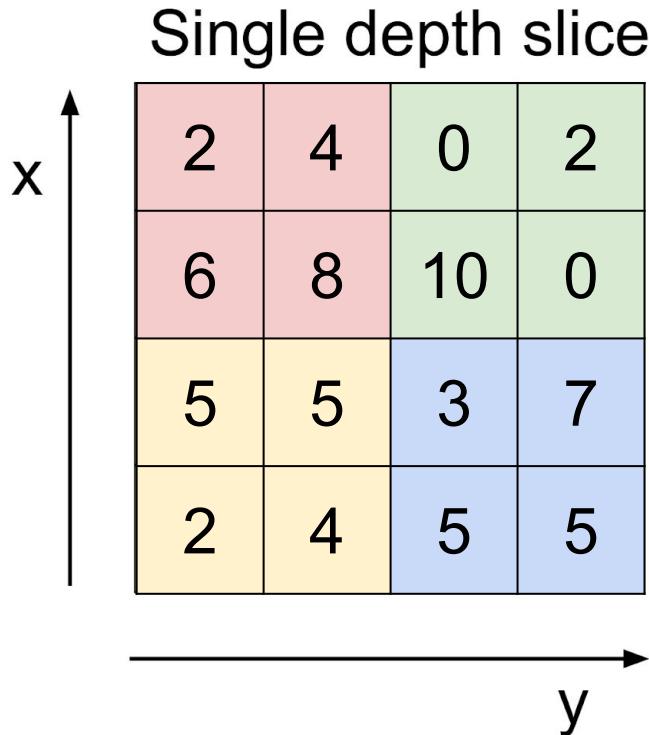
Max Pooling



max pool with 2x2 filters
and stride 2

6	8
3	4

Average Pooling



max pool with 2x2 filters
and stride 2

5	3
4	5

Classic ConvNet Architecture

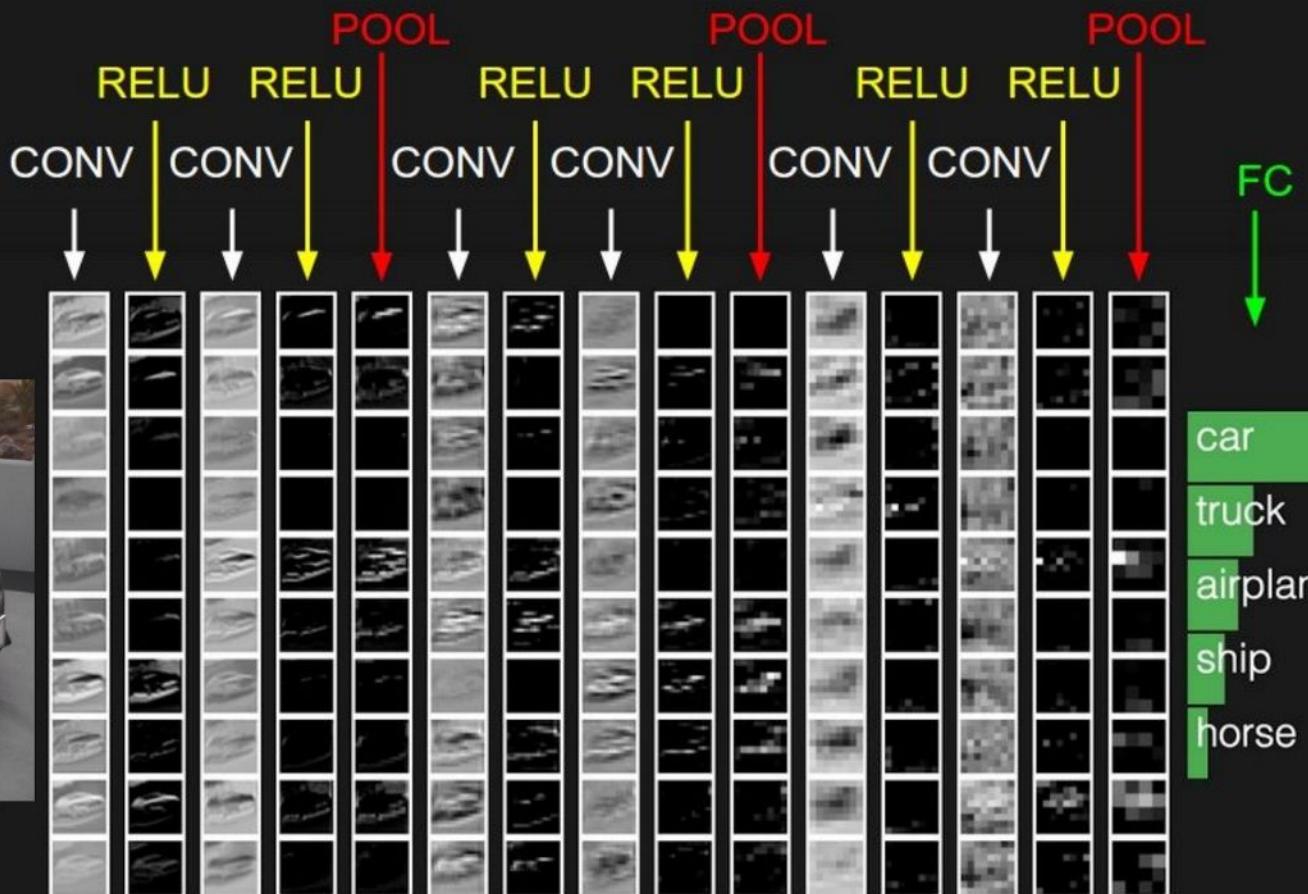
Input

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

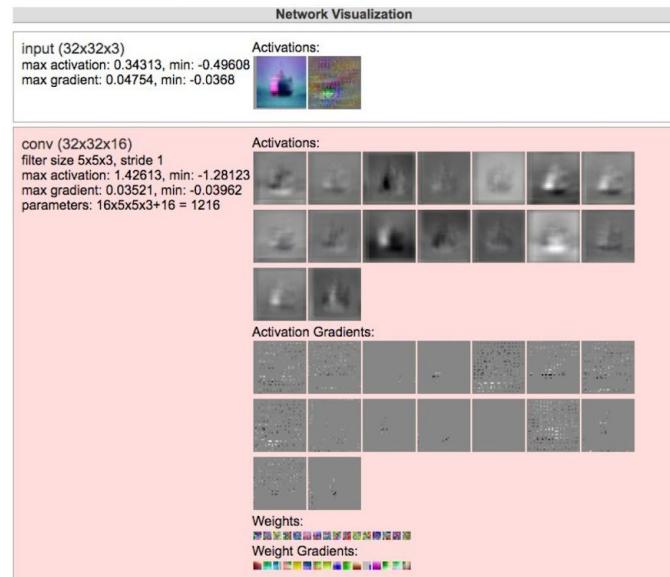
Description

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 vertically.

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.

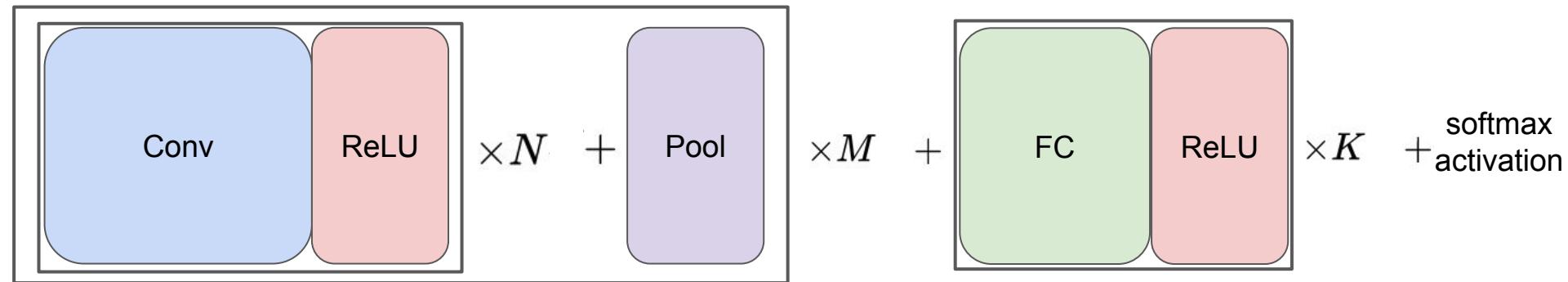
Report questions/bugs/suggestions to [@karpathy](#).



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

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, $0 \leq K \leq 2$.

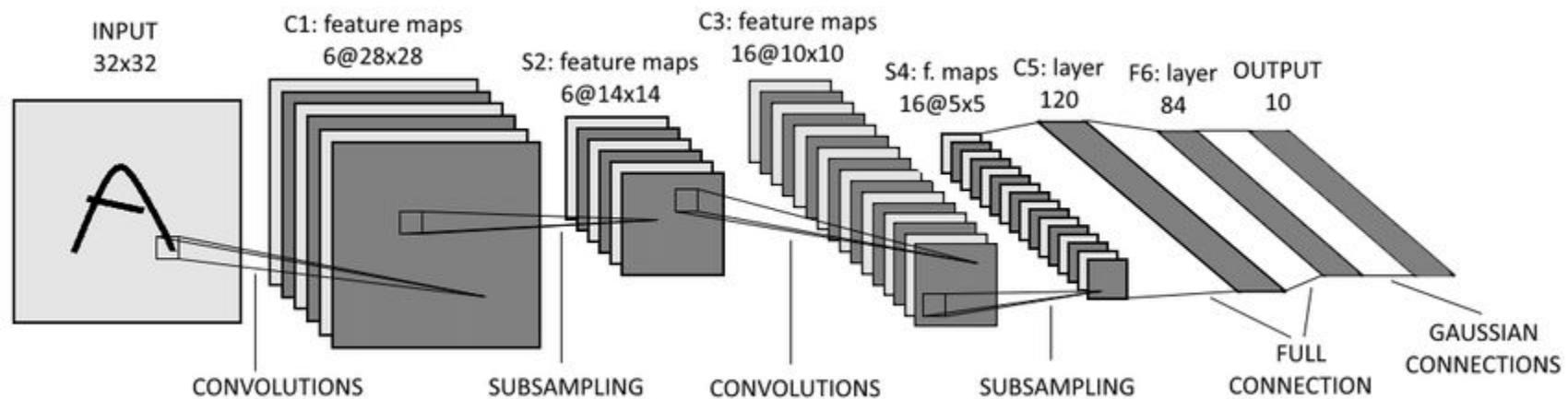
- but recent advances such as ResNet/GoogLeNet challenge this paradigm

Outline

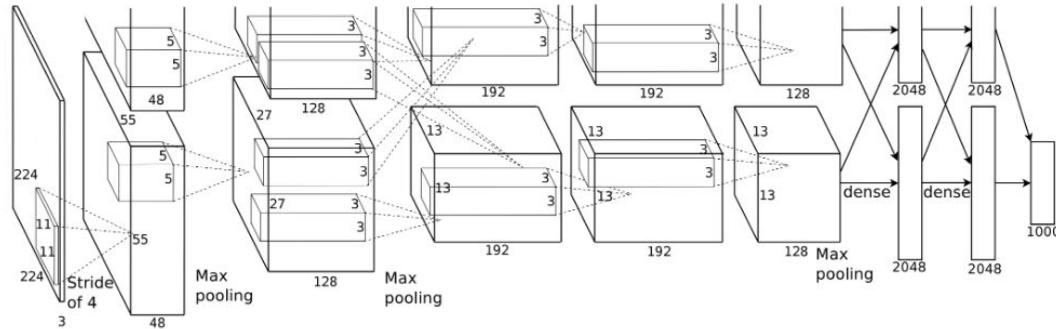
- Optimization
- CNN basics
- Examples of CNN Architectures
- Applications of CNN

LeCun et al. 1989

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



AlexNet



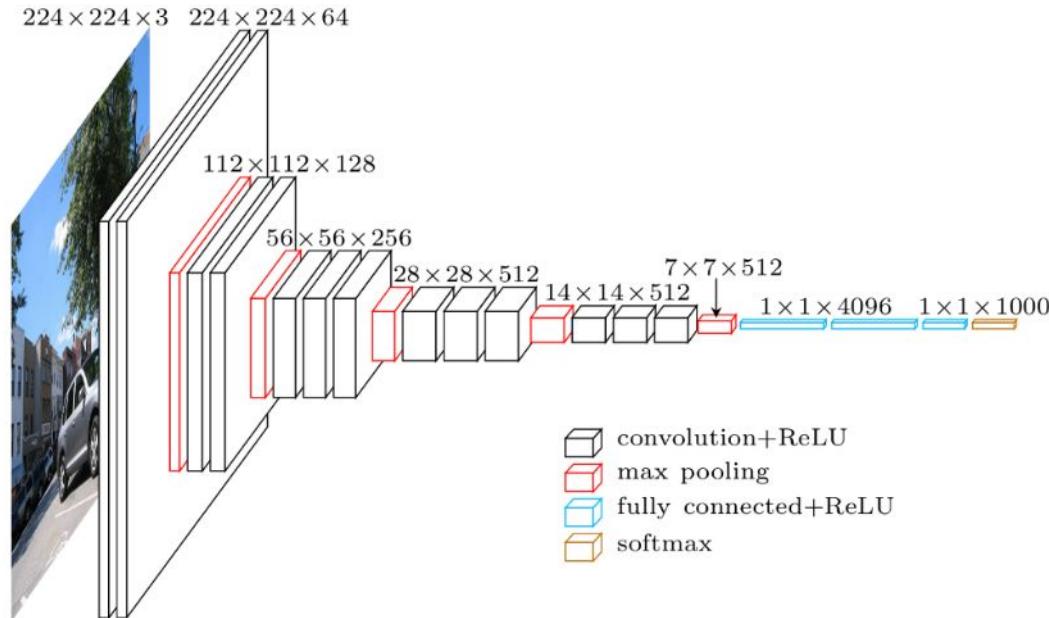
First conv layer: kernel 11x11x3x96 stride 4

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

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

Slide credit: Charles Ollion - Olivier Grisel

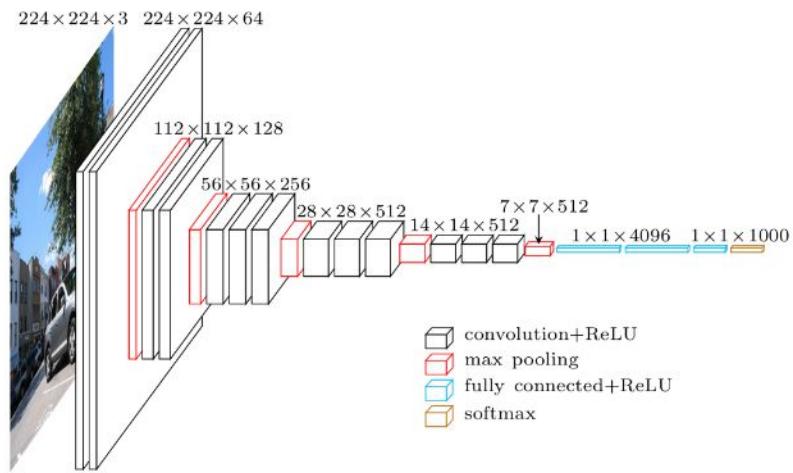
VGG-16



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

Slide credit: Charles Ollion - Olivier Grisel

VGG-16



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
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
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
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

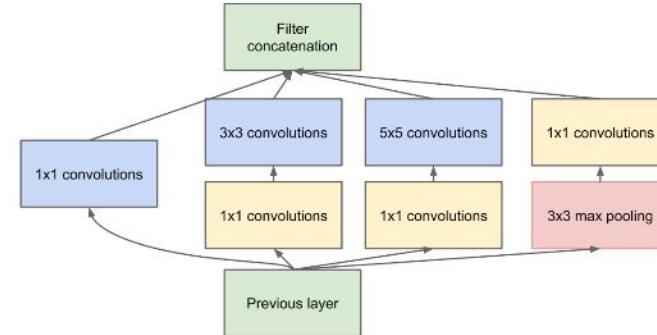
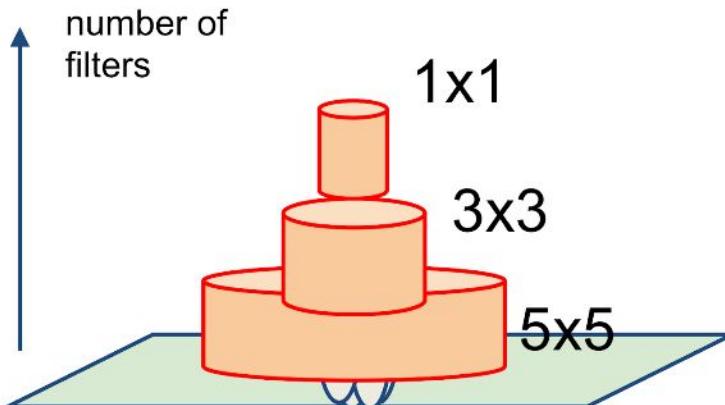
Lots of 3x3 convolution layers: more non-linearity than a single 7x7 conv layer

Very Deep Models

Going deep with convolutions, Szegedy et al.

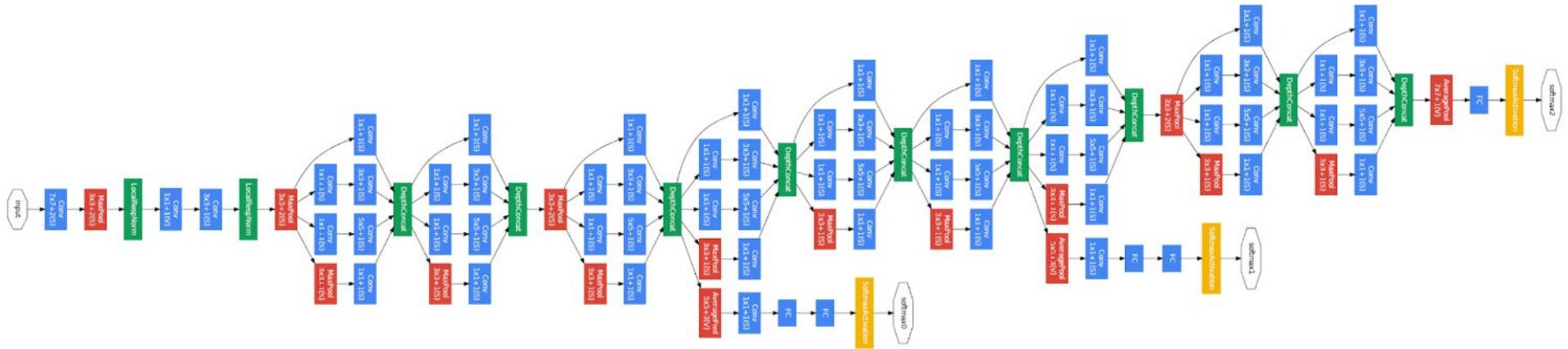
GoogLeNet inception module:

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.

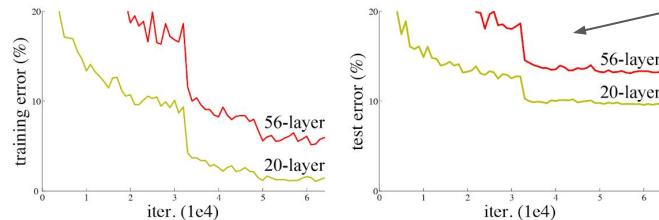


Slide credit: Rob Fergus

Residual Networks

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

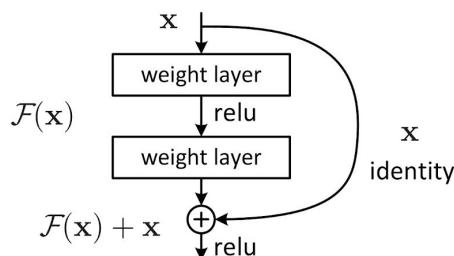
Really, really deep convnets don't train well,
E.g. CIFAR10:



deeper networks
perform worse due
to vanishing
gradient

Key idea: introduce “pass
through” into each layer

Thus only residual now
needs to be learned

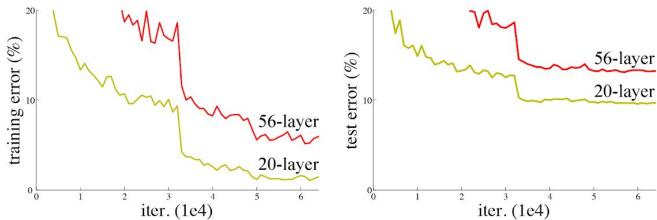


Slide credit
Rob Fergus

Residual Networks

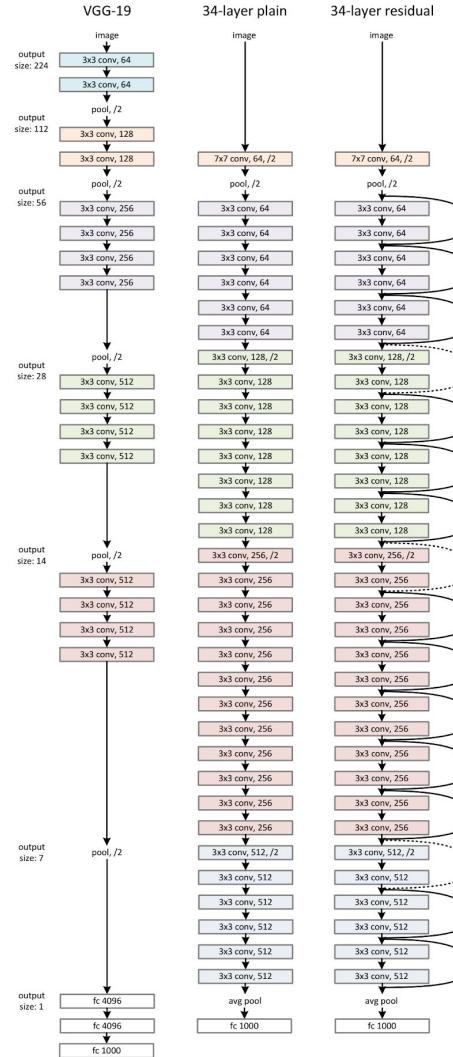
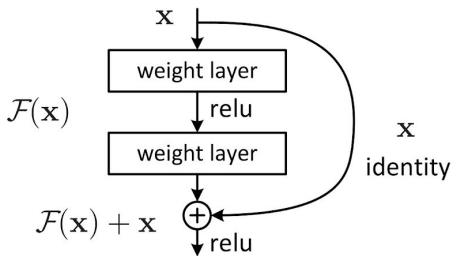
[He, Zhang, Ren, Sun, CVPR 2016]

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

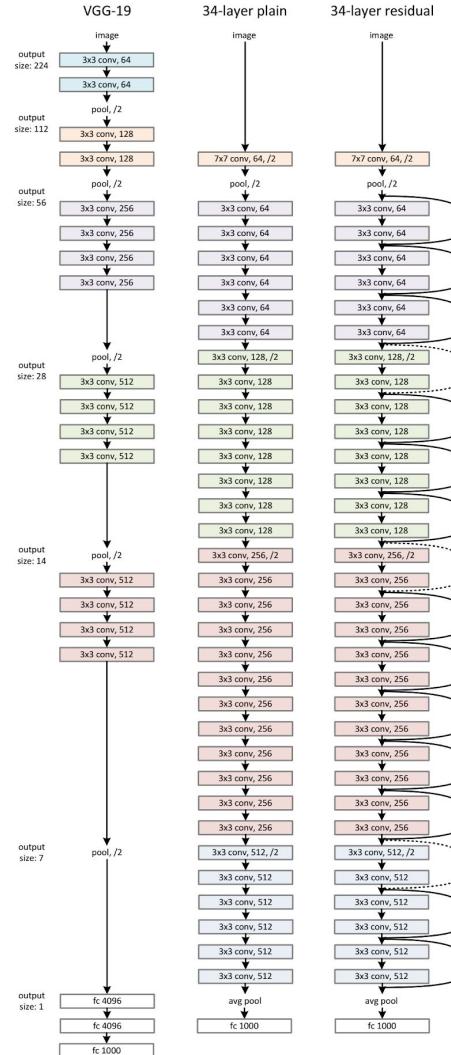


Slide credit
Rob Fergus

Residual Networks

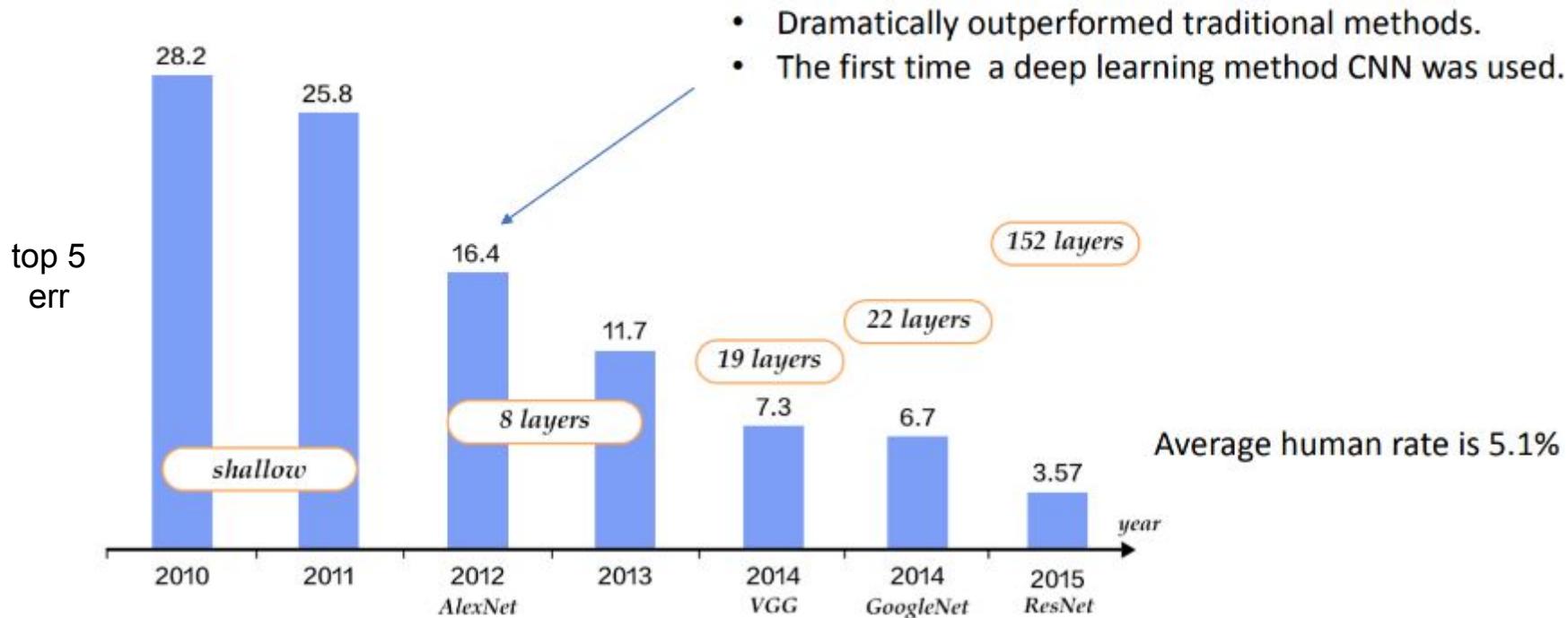
ResNet50 Compared to VGG:

- 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



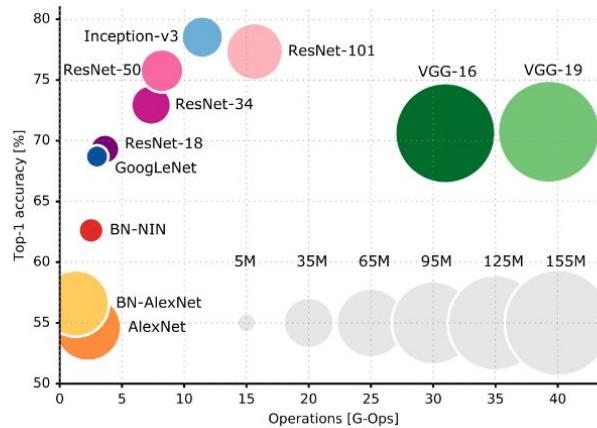
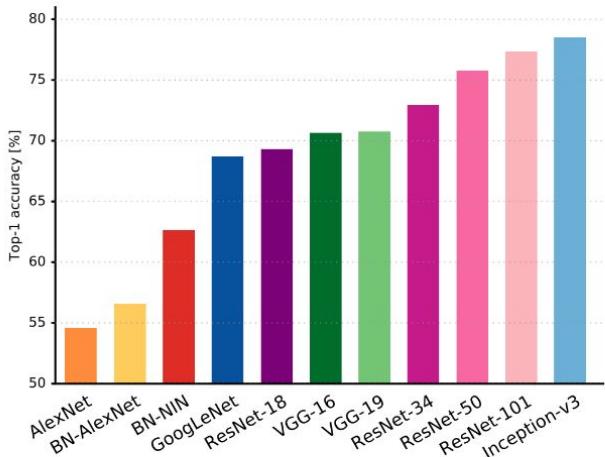
Deeper is better

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

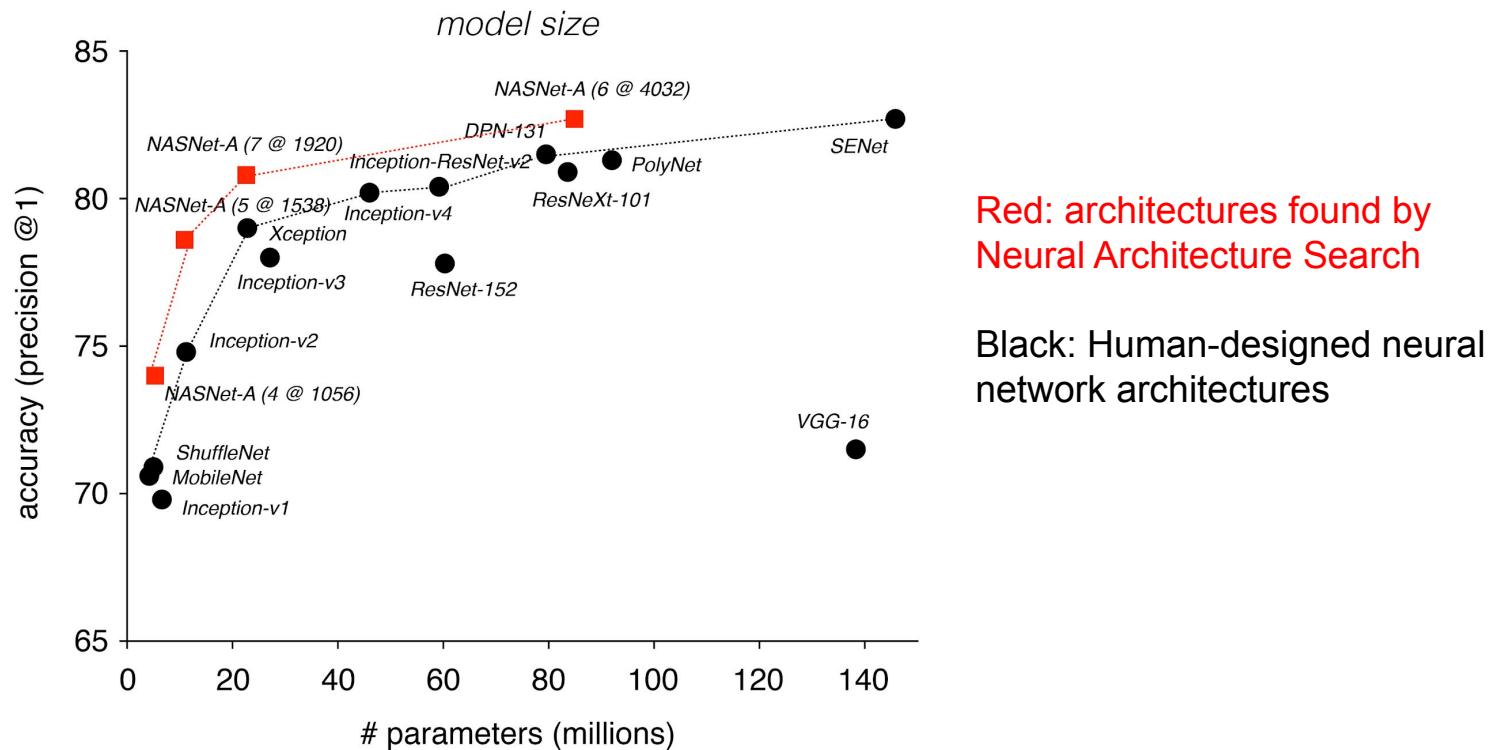


ImageNet classification

Top 1-accuracy, performance and size on ImageNet



Learned architectures surpass human-invented



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): $\hat{y} = (\hat{p}_1, \hat{p}_2, \dots, \hat{p}_C)$

- Loss? $\mathcal{L}(y, \hat{y}) = -y \log(\hat{y}) = -\sum_{c=1}^C p_c \log(\hat{p}_c)$

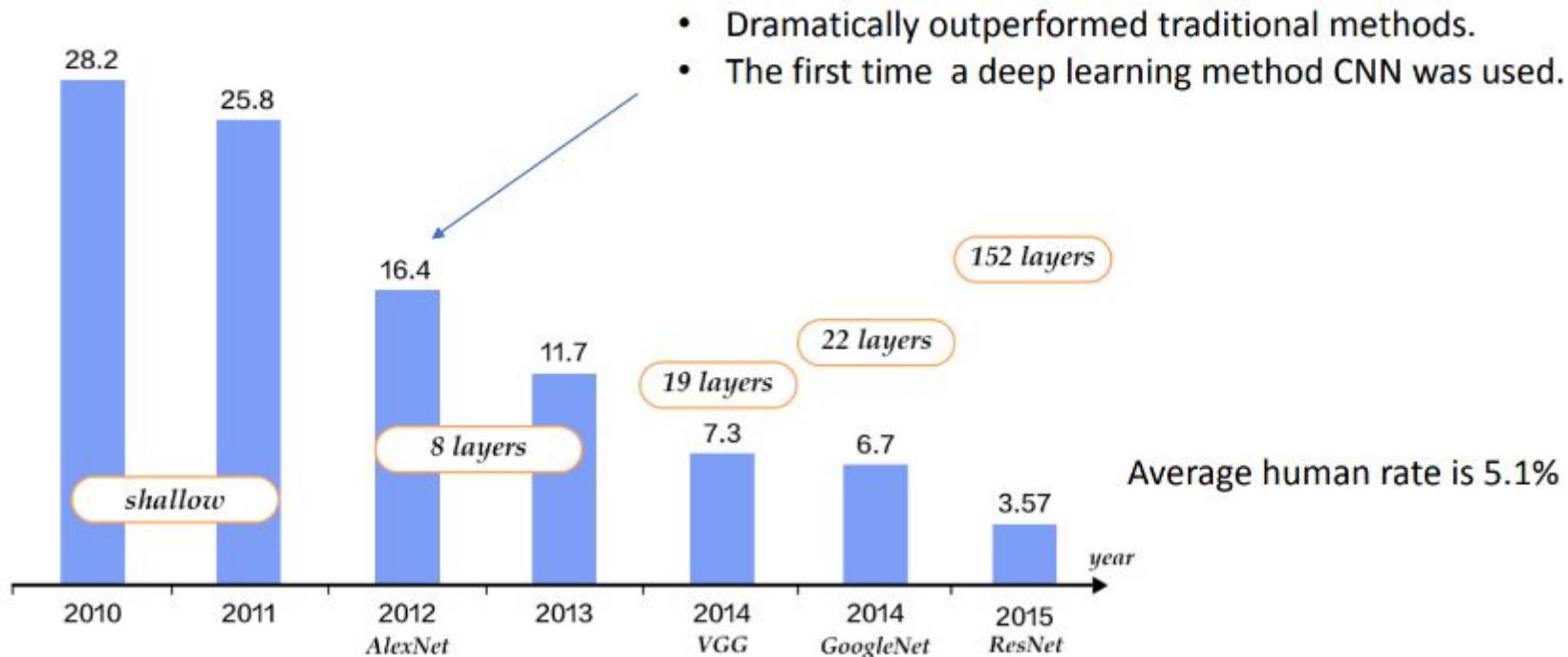
Object classification

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



Object Classification: Deeper is better

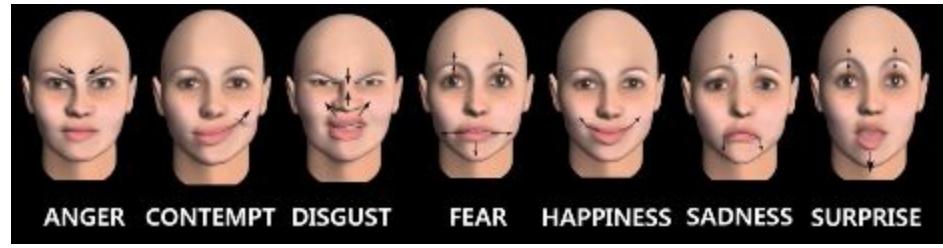
ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



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

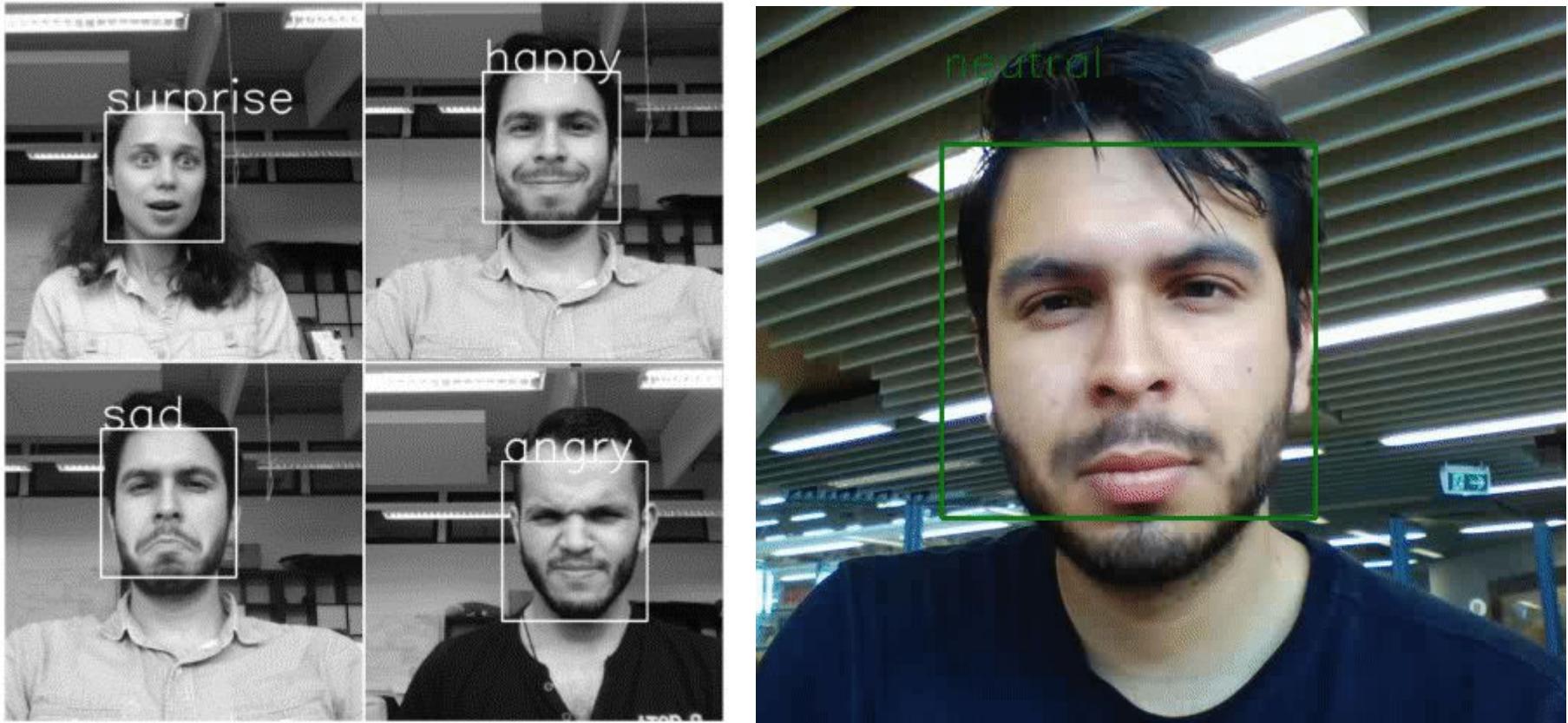


Figure credit: https://github.com/oarriaga/face_classification

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

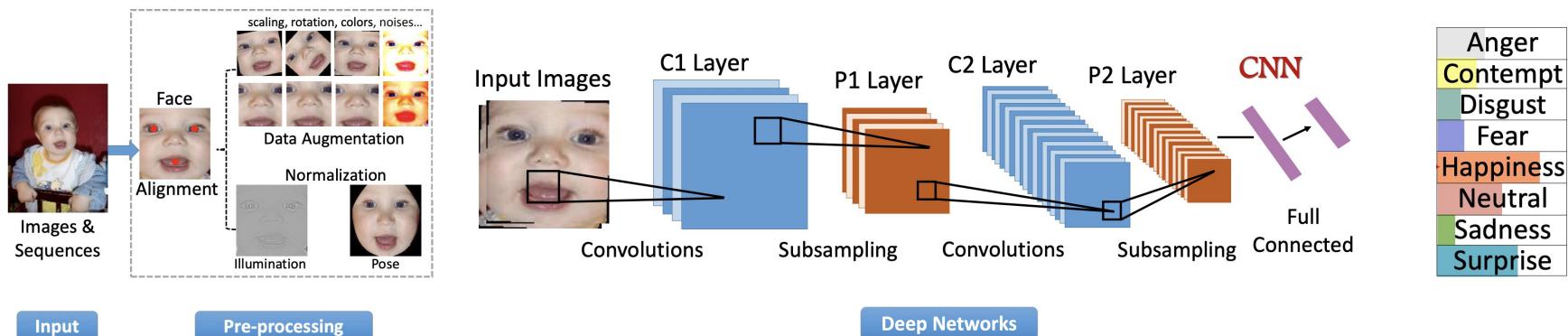
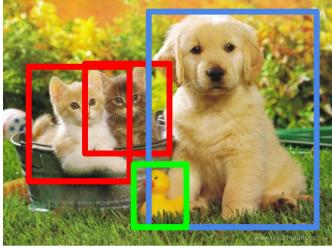


Figure credit: Shan Li and Weihong Deng. Deep Facial Expression Recognition: A Survey.

Applications of CNN

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

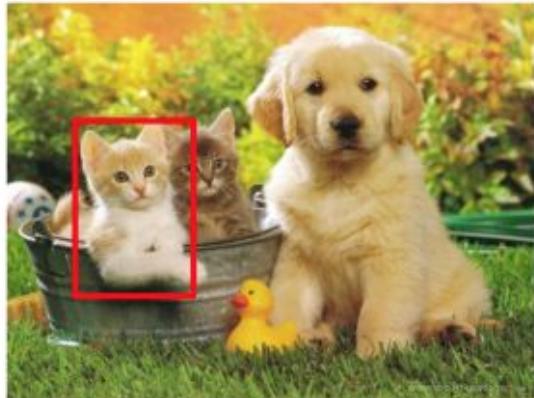
Object Detection

- 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]



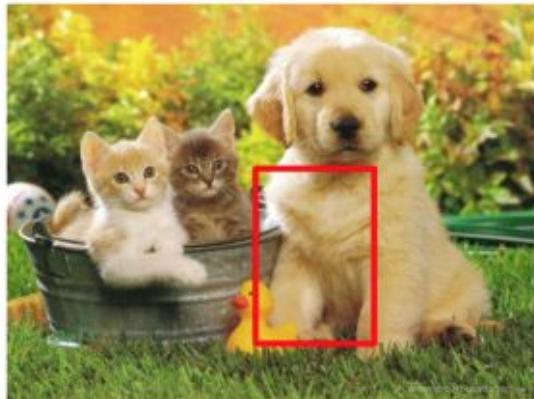
Cat ? YES

Dog ? NO

Duck? NO

Object Detection

Classes = [cat, dog, duck]

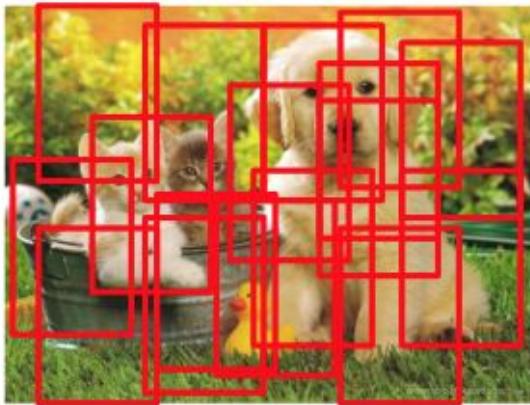


Cat ? NO

Dog ? NO

Duck? NO

Object Detection

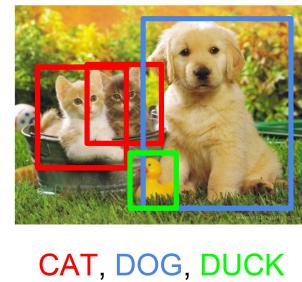


Problem:
Too many positions & scales to test

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

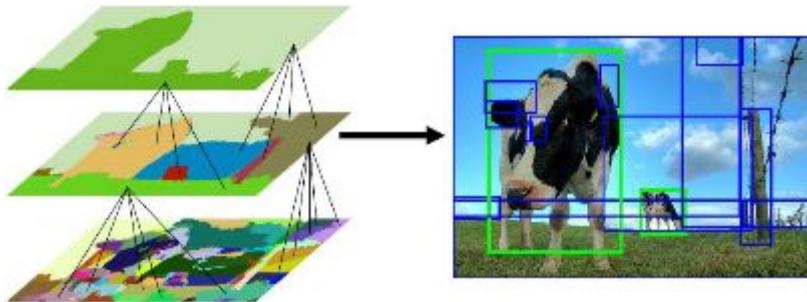
Object Detection

- 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)



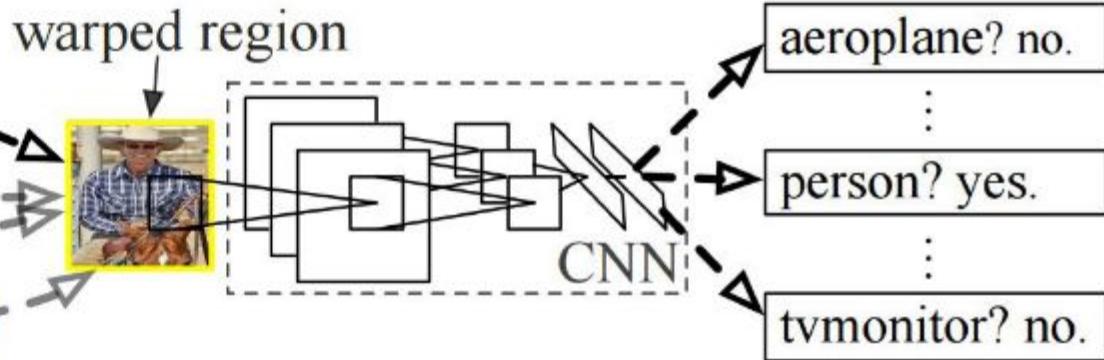
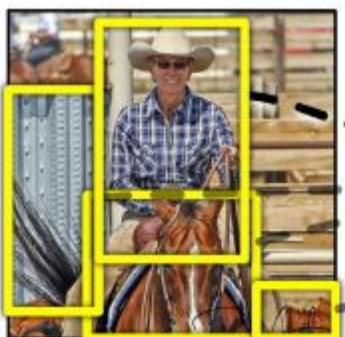
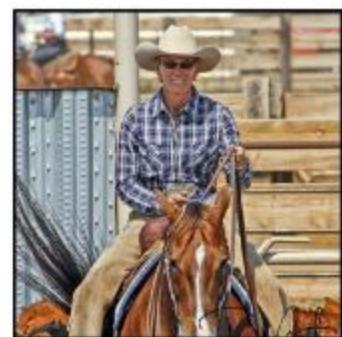
Bounding Box Proposals

- 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)



Region-CNN (RCNN)

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



1. Input image

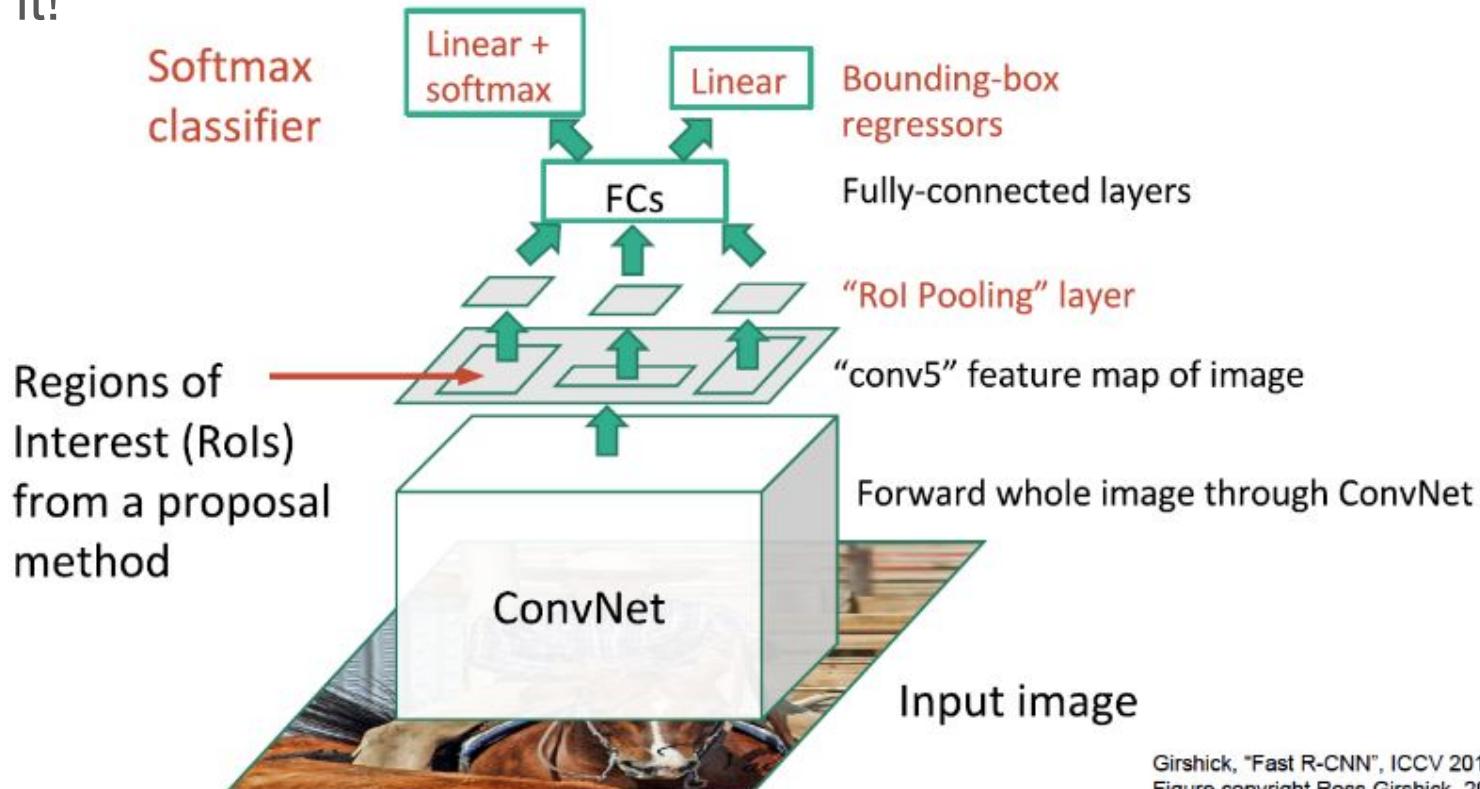
2. Extract region proposals (~2k)

3. Compute CNN features

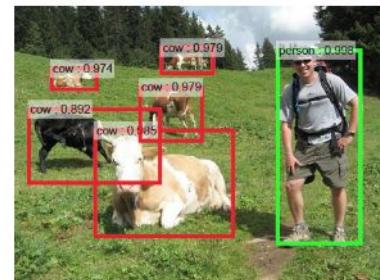
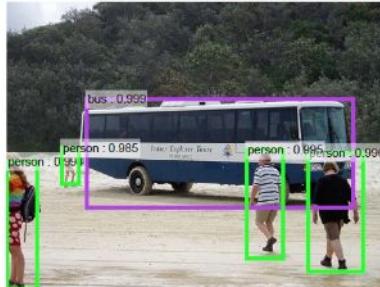
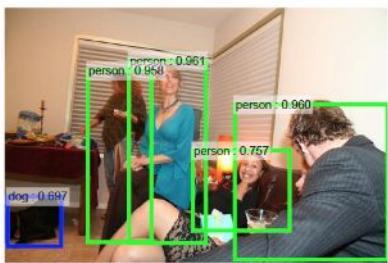
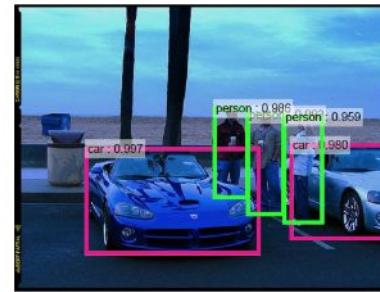
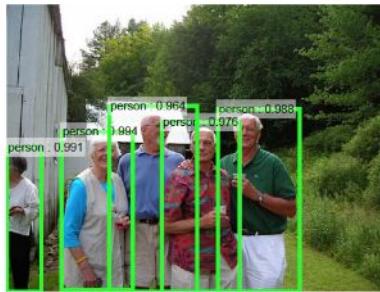
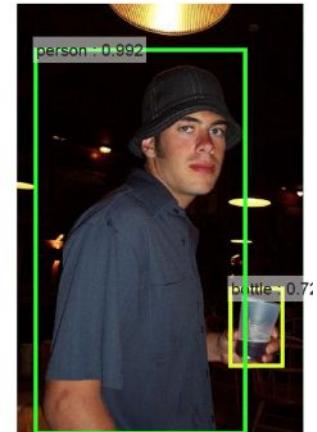
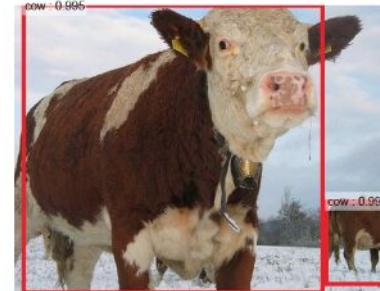
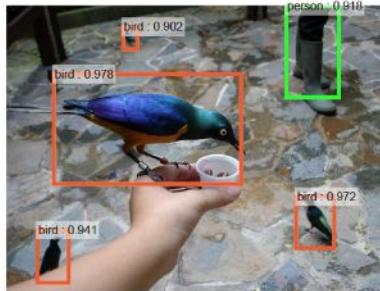
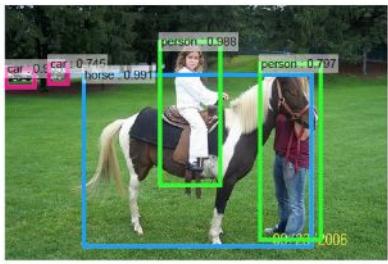
4. Classify regions

Fast RCNN

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



Faster RCNN



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?

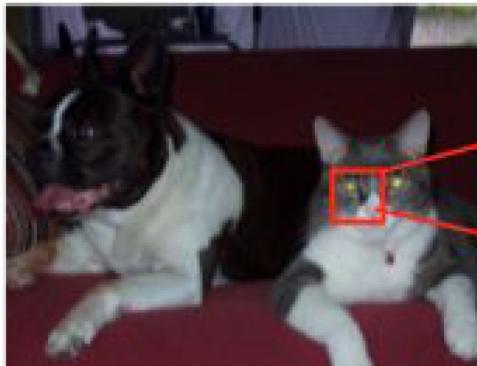


- Output? Pixel-level class map

- Loss?

Pixel level classification loss

From Classification to Semantic Segmentation



extract a
patch



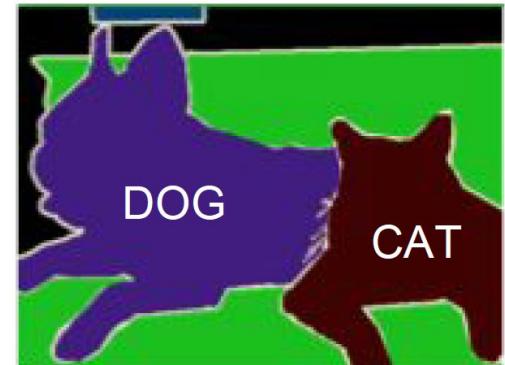
run through a CNN
trained for image
classification



classify the
center pixel

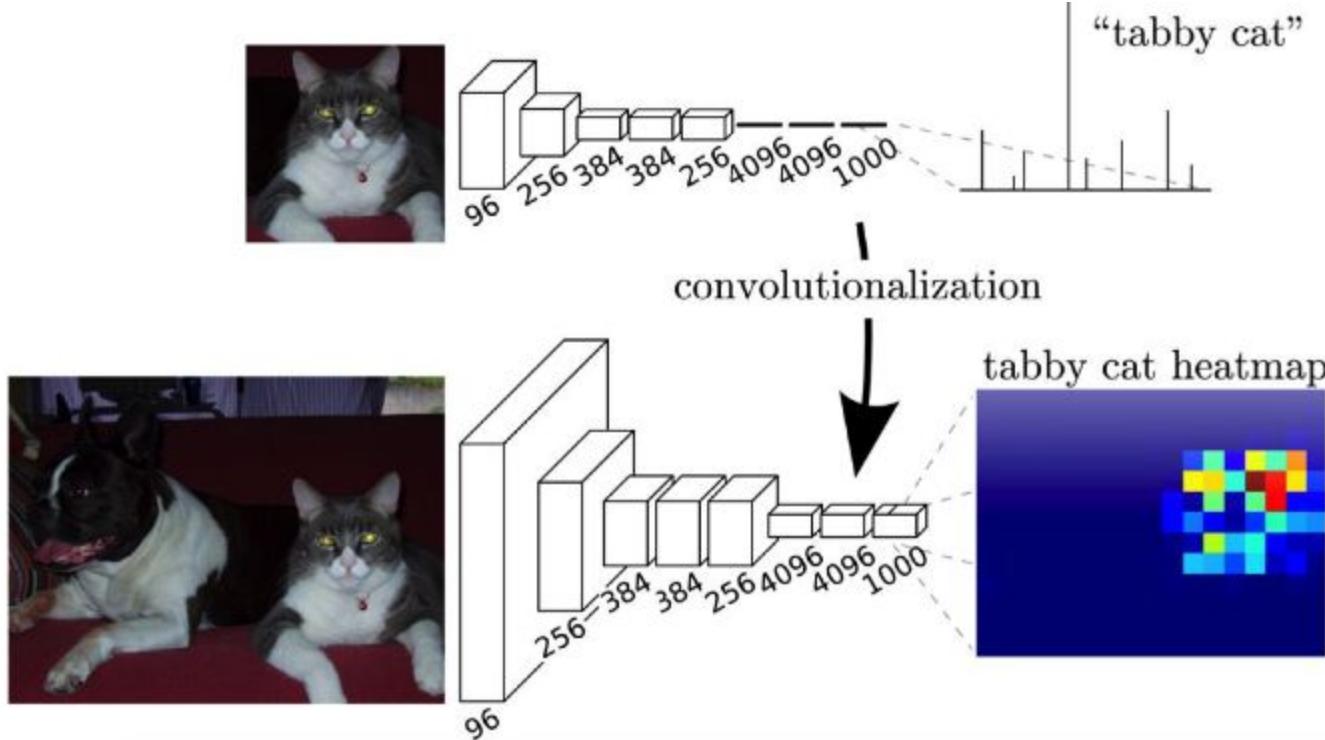
CAT

repeat for every pixel



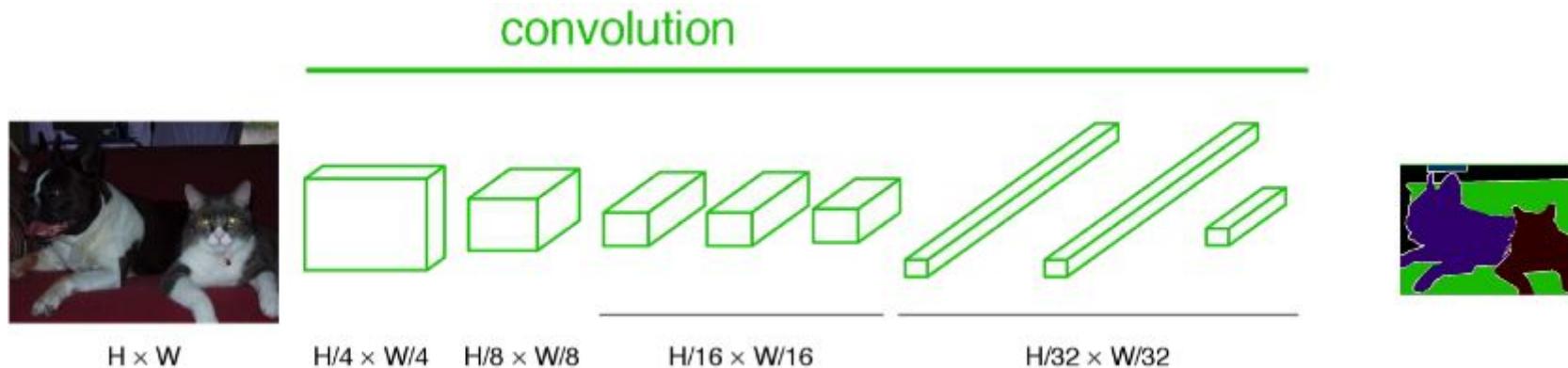
From Classification to Semantic Segmentation

- A classification network becoming fully convolutional



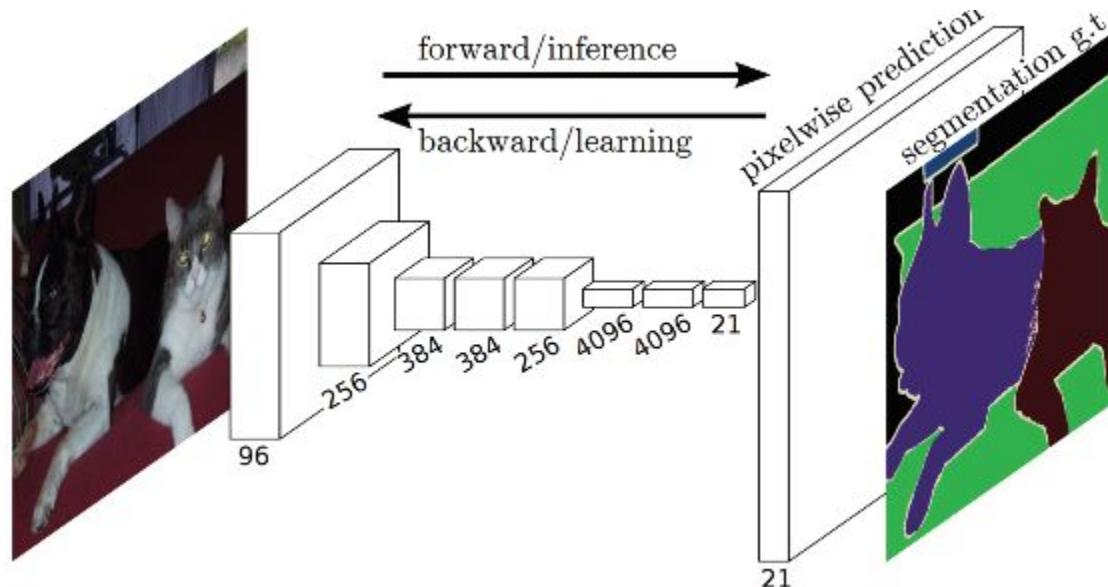
From Classification to Semantic Segmentation

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

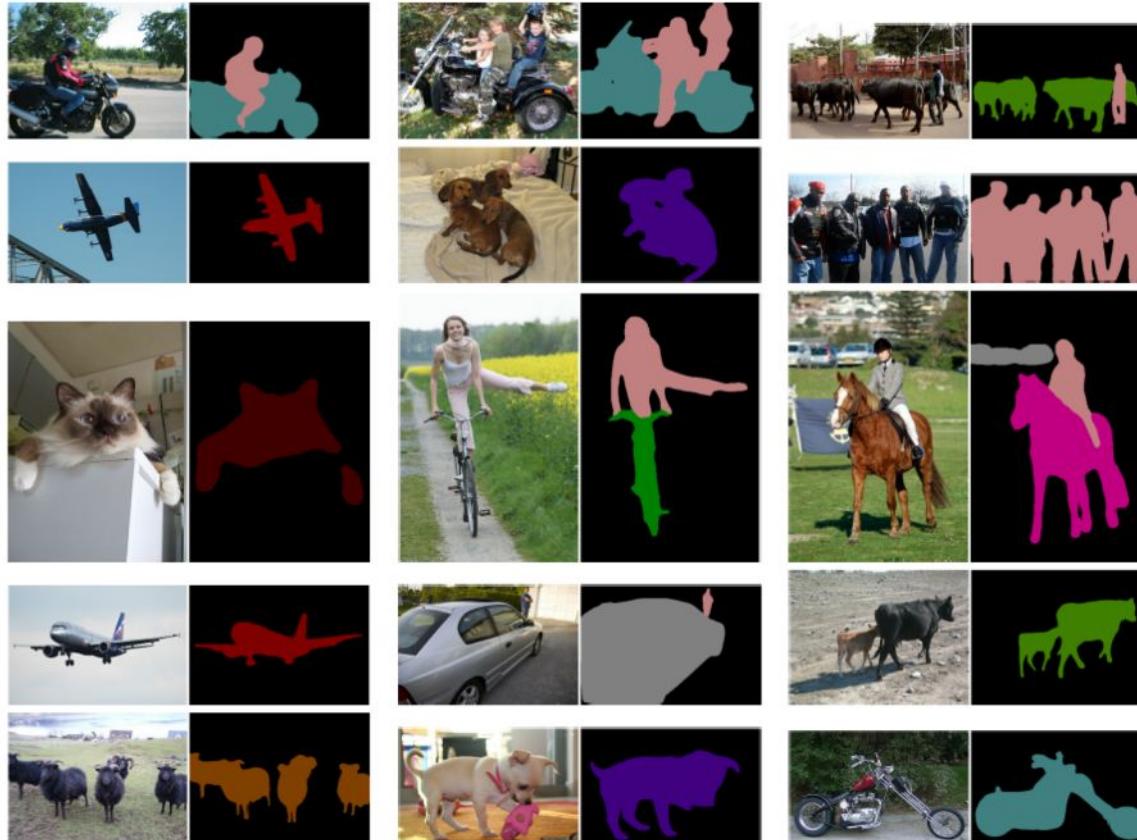


From Classification to Semantic Segmentation

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



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.

Quiz

<https://forms.gle/KRxi3qqLooQBJj1c9>

