Image Recognition with Convolutional Networks

Karen Simonyan

Lecture Format

Outline

- The foundations of image recognition with neural networks.
- Stacking the layers: convolutional networks.
- Going deeper: the challenges and how to solve them.
- Image recognition beyond classification.
- Visualising ConvNets.

The foundations of image recognition with neural networks

Neural Networks for Image Data

- Previous lecture
  - generic neural networks (Linear layer)
  - generic tensor inputs

- This lecture
  - Neural Networks for image inputs -- Convolutional Networks (ConvNets)
  - ConvNets also applicable to other signals: video, audio, text

Locality and Translation Invariance

Properties

- **Locality**: objects tend to have a local spatial support
- **Translation invariance**: object appearance is independent of location
- Can define these properties since an image lies on a grid/lattice
  - ConvNet machinery applicable to other data with such properties, e.g. audio/text



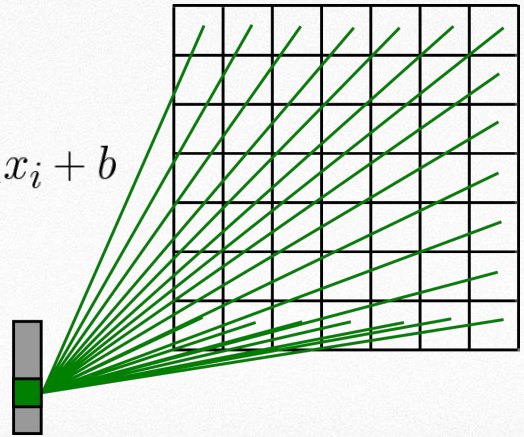
The bird occupies a local area and looks the same in different parts of an image.
We should construct neural nets which exploit these properties!

Incorporating Assumptions

Locality

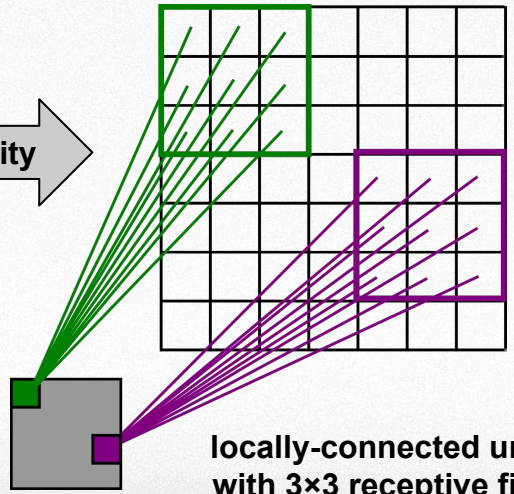
- Make **fully**-connected layer **locally**-connected
- Each unit/neuron is connected to a local rectangular area – receptive field
- Different units connected to different locations
  - output (“**feature map**”) lies on a grid itself
- Rarely used, beneficial only in specific cases (face recognition)

$$y = \sum_{i \in image} w_i x_i + b$$



fully-connected unit

locality



locally-connected units with 3×3 receptive field

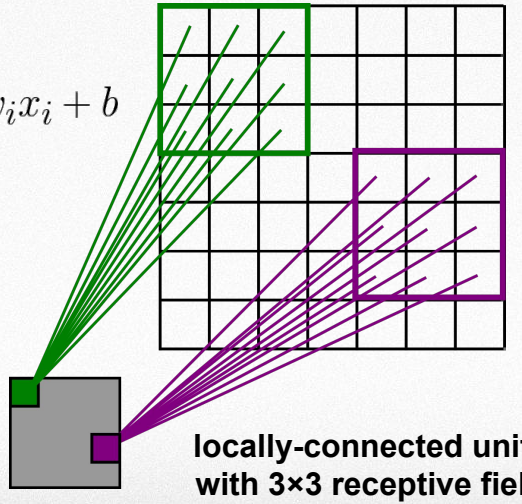
$$y = \sum_{i \in receptive\ field} w_i x_i + b$$

Incorporating Assumptions

Translation Invariance

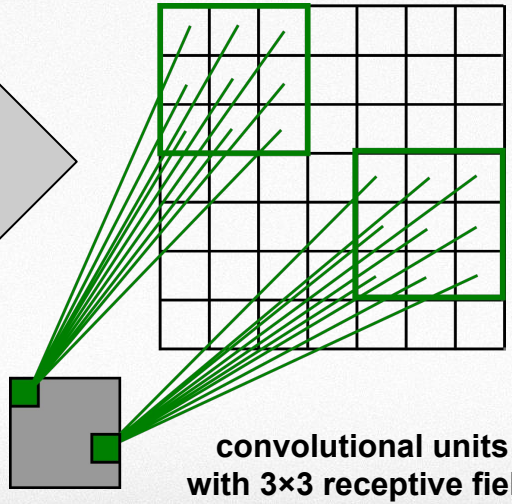
- **Weight sharing**
  - units connected to different locations have the same weights
  - equivalently, each unit is applied to all locations
- *Convolutional layer – locally-connected layer with weight sharing (translation invariance)*
- The weights are invariant, the output is equivariant

$$y = \sum_{i \in receptive\ field} w_i x_i + b$$



locally-connected units with 3×3 receptive field

weight sharing



$$y = w * x + b$$

convolutional units with 3×3 receptive field

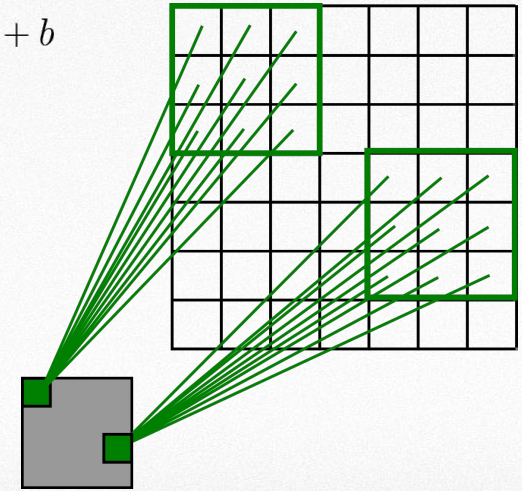
Conv Layer Mechanics

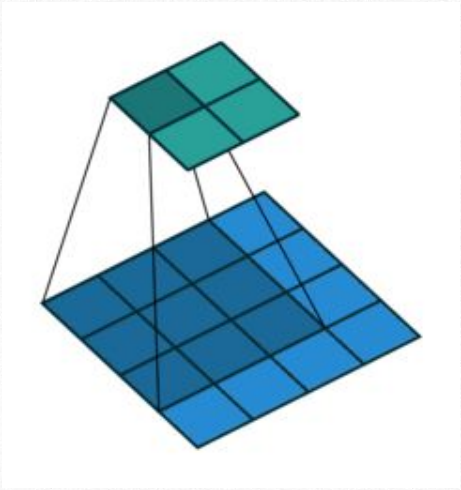
single-channel scenario

- Weight matrix of conv layer is called conv kernel (or filter)
- To compute the output feature map
  - slide the receptive field of the filter over the input and compute dot products
  - receptive field size == filter size

$$y = \sum_{i \in receptive\ field} w_i x_i + b$$

$$y = w * x + b$$





filter has 3*3=9 weights

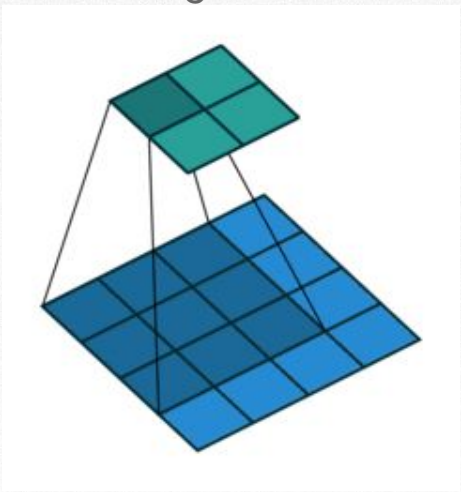
evaluation of a single conv filter with 3×3 receptive field on 4×4 input produces 2×2 output

Conv Layer Mechanics

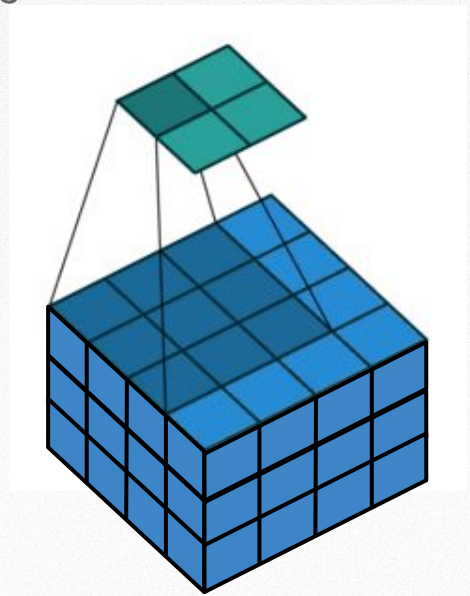
multi-channel scenario

- Conv layer input and output can have multiple channels
  - e.g. 3-channel RGB image or 16-channel feature map

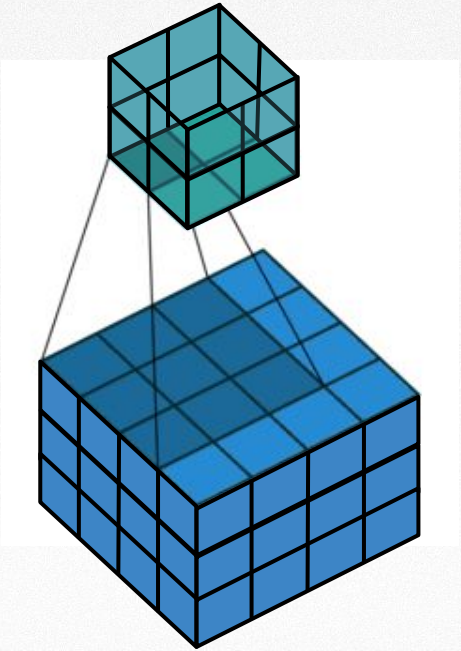
feature maps are 3-D tensors
height × width × channels



4×4×1 input, 2×2×1 output
3*3=9 filter weights
projecting from 9-D onto 1-D



4×4×3 input, 2×2×1 output
3*3*3=27 filter weights
projecting from 27-D onto 1-D

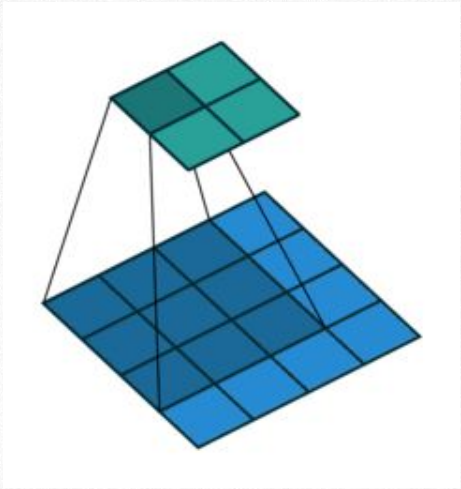


4×4×3 input, 2×2×2 output
3*3*3*2=54 filter weights
projecting from 54-D onto 2-D

Conv Layer Mechanics

Output size

- We'll assume multi-channel input & output from now on
- For N×N input and kernel size k×k the output size is M = N - k + 1
- We consider all receptive fields lying fully within the input: known as 'VALID' convolution

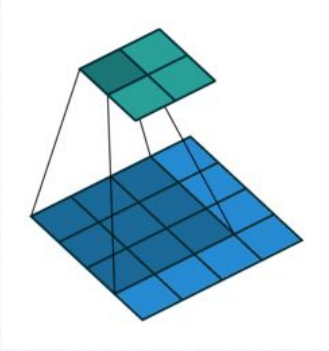


4×4×cin input
2×2×cout output

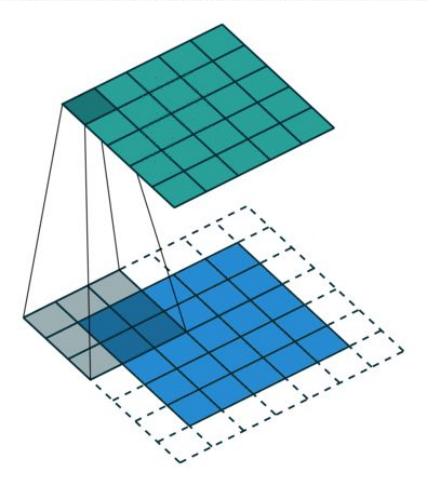
Conv Layer Variants

Padded Convolution

- Increase (pad) the input with p zeros on both sides
  - sometimes implemented as a separate padding layer
- Purpose: control output resolution (e.g. preserve resolution)
- Common settings
  - ‘VALID’: p=0
  - ‘SAME’: p = (k - 1)/2 on each side for kernel size k
    - receptive fields go beyond the original input
    - output has the same spatial size as the input



4×4×cin input, 2×2×cout output
‘VALID’ padding



5×5×cin input, 5×5×cout output
‘SAME’ padding

Conv Layer Variants

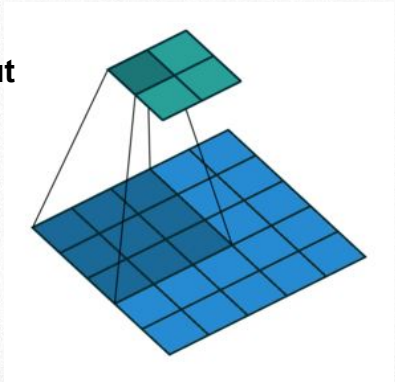
Strided Convolution

- Conv filter is applied with a step (“**stride**”) between receptive fields
- Purposes
  - reduce spatial resolution for faster processing
  - achieve invariance to local translation

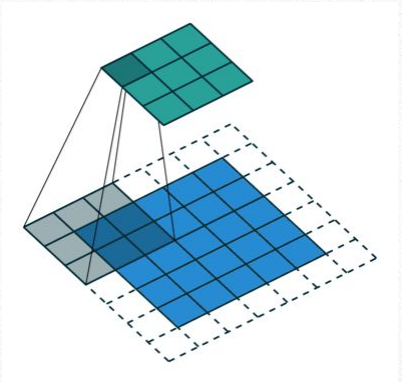
- Output size: $M = \left\lfloor \frac{N + 2p - k}{s} \right\rfloor + 1$ for input size N, kernel size k, padding p, and stride s

M×M output

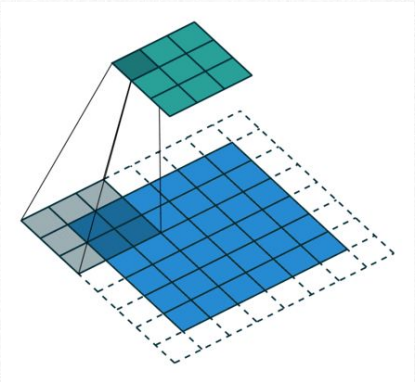
N×N input



N=5, k=3, p=0, s=2 ⇒ M=2



N=5, k=3, p=1, s=2 ⇒ M=3

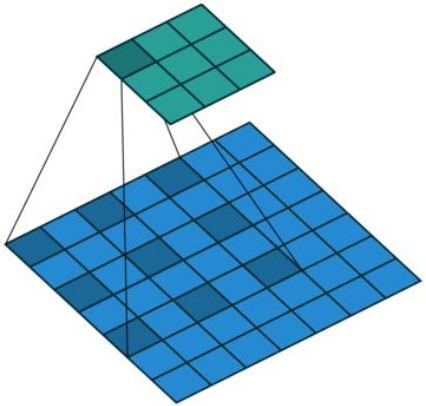


N=6, k=3, p=1, s=2 ⇒ M=3

Conv Layer Variants

Dilated Convolution

- Conv filter is applied with a step (“**dilation rate”)** between kernel elements
  - k×k kernel dilated to size k’=k+(k-1)(r-1), where r is dilation rate
- Purposes
  - large receptive field with a small kernel
  - fast alternative to large kernels
- Output size
  - computed based on the dilated kernel size k’
  - $M = \left\lfloor \frac{N + 2p - k'}{s} \right\rfloor + 1$



M×M output

N×N input

k=3, r=2 ⇒ k’=5
N=7, k’=5, p=0, s=1 ⇒ M=3

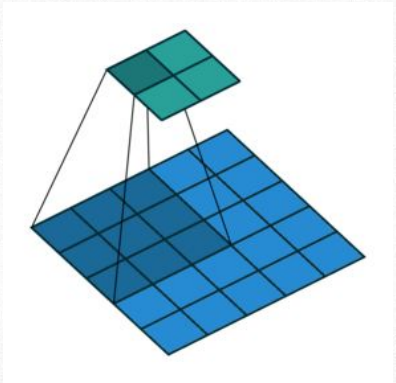
Conv Layer Variants

Transposed Convolution

- Also known as: up-convolution, de-convolution, fractionally-strided convolution
- Purpose: **increase** the resolution
- Does the opposite of strided convolution
  - implemented by swapping forward and backward operations of standard convolution
- Output size $N = s(M-1) + k - 2p$

M×M output

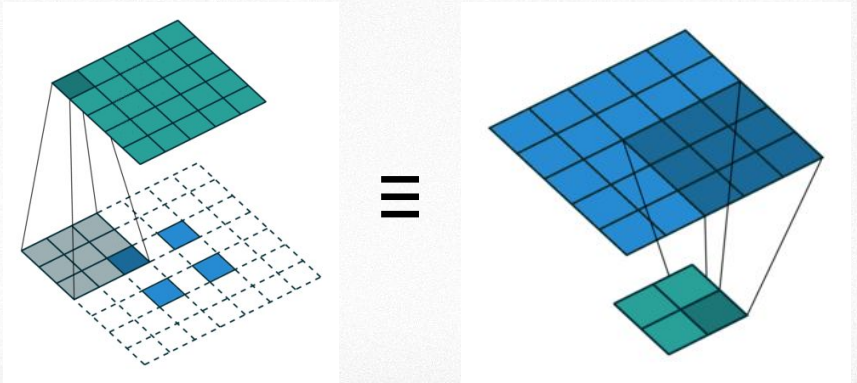
N×N input



strided convolution

N×N output

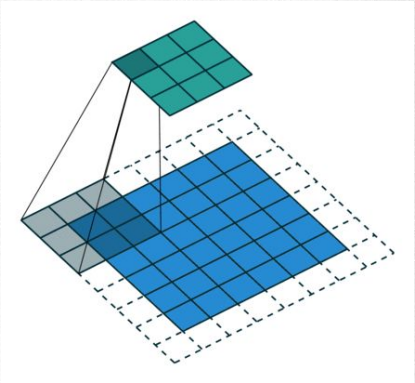
M×M input



transposed convolution

Pooling Layer

- Purposes (same as strided convolution)
  - reduce spatial resolution for faster processing
  - achieve invariance to local translation
- Average pooling
  - computes the average input over the receptive field
  - same as k×k strided convolution with weights fixed to 1/(k*k)
- Max pooling
  - computes the max input over the receptive field
- Global pooling
  - pooling with the whole input as the receptive field
  - gets rid of spatial dimensions, full invariance to location
  - can be average or max



Summary

The foundations of image recognition with neural networks

Convolutional layer – main building block for image recognition NNs

- main properties: locality and translation invariance
- various spatial connectivity patterns
- main parameters: filter size, number of channels, stride, padding

QUESTIONS?

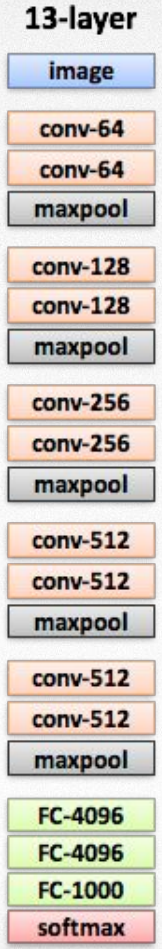
Convolutional networks

stacking the layers together

Convolutional Networks

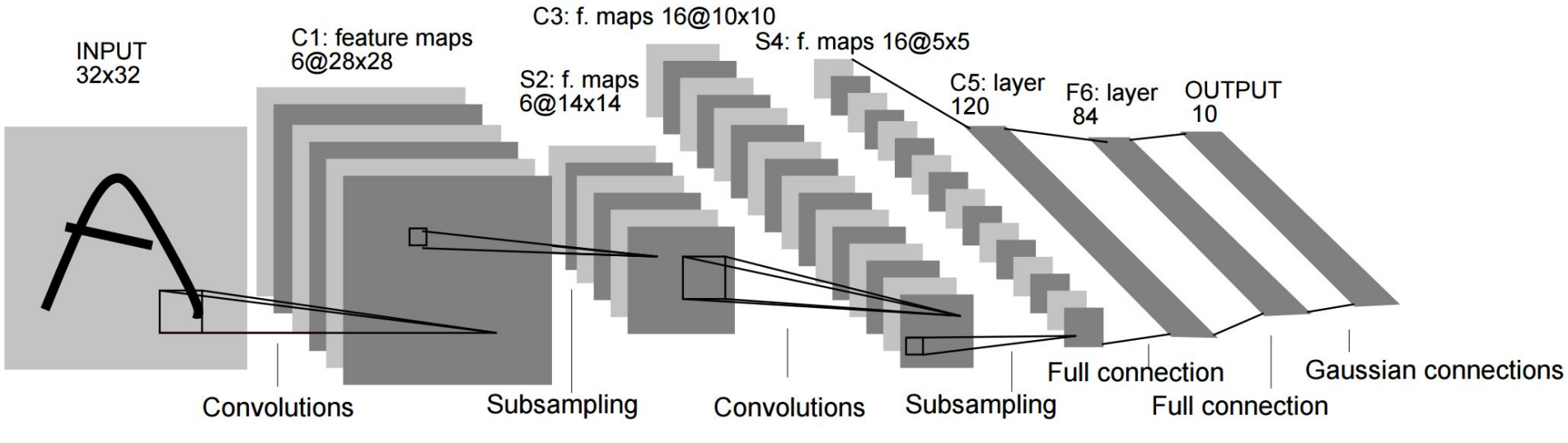
for image classification

- Now we are ready to build an image classification network using conv layers
- Activation function: RELU(x) = max(x, 0)
- Typical structure for image classification
  - image → [[conv →] * M → pool] * N → [linear] * K → softmax



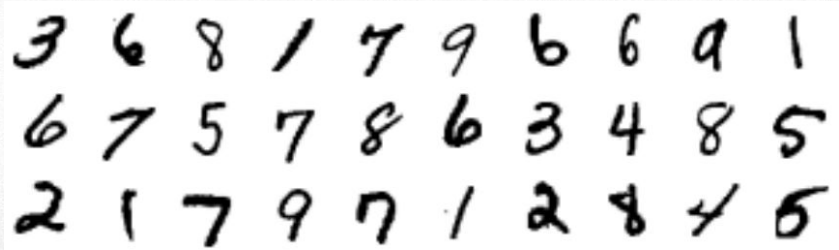
Case study 1: MNIST classification

LeNet-5 [LeCun et al., 1998]



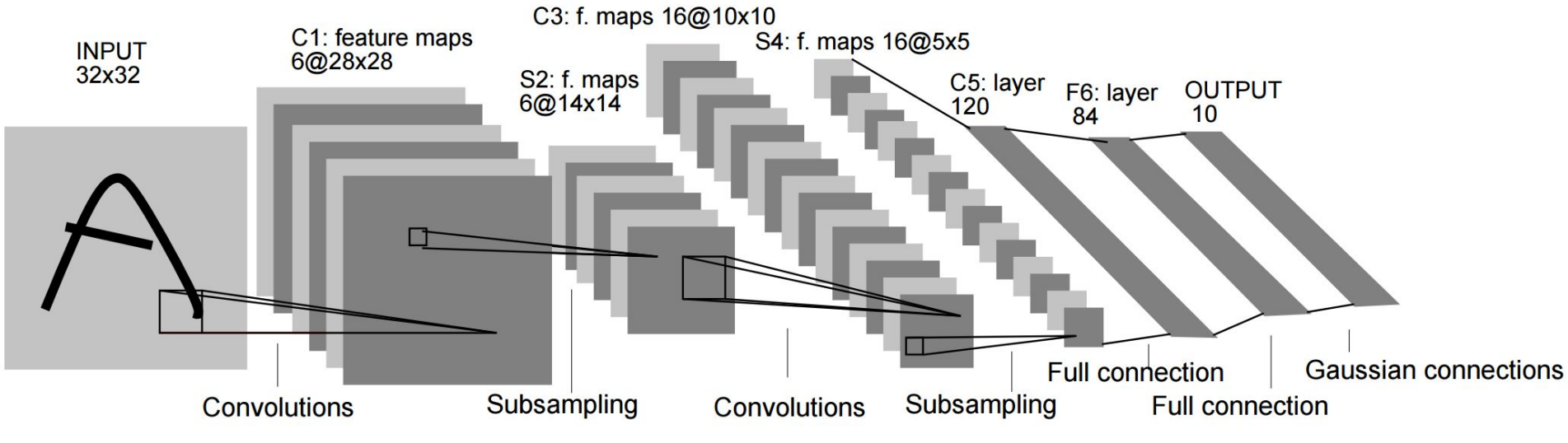
Task

- hand-written digit classification
- 10 classes



Case study 1: MNIST classification

LeNet-5 [LeCun et al., 1998]

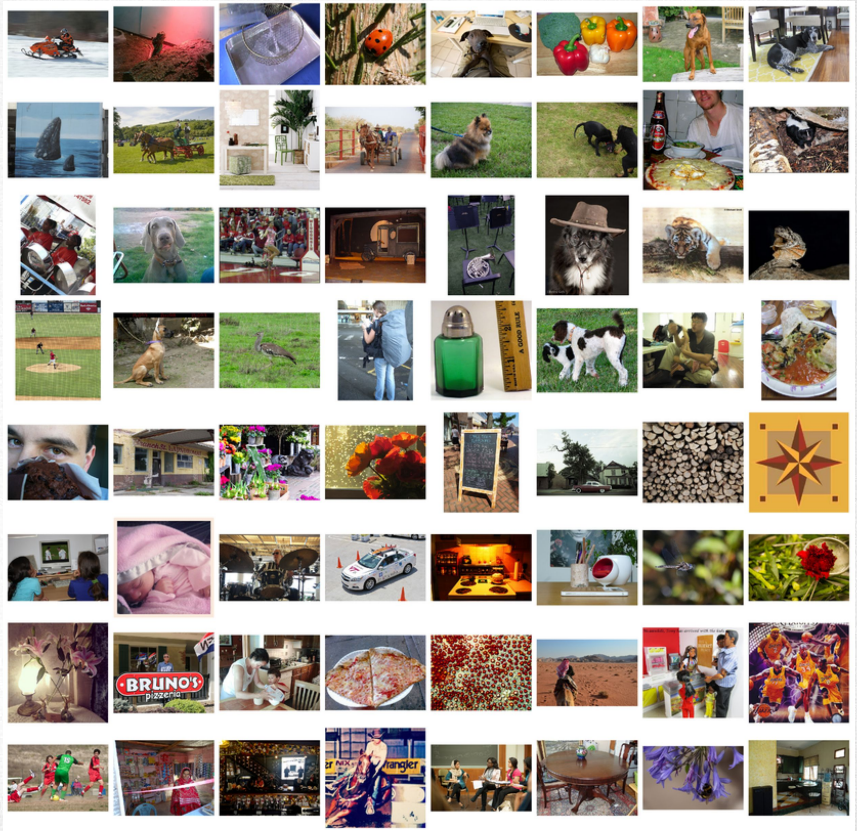


Layer configuration:

- 5×5 conv, stride=1, ‘VALID’ padding, sigmoid activation
- 2×2 average pool, stride=2

ImageNet Challenge

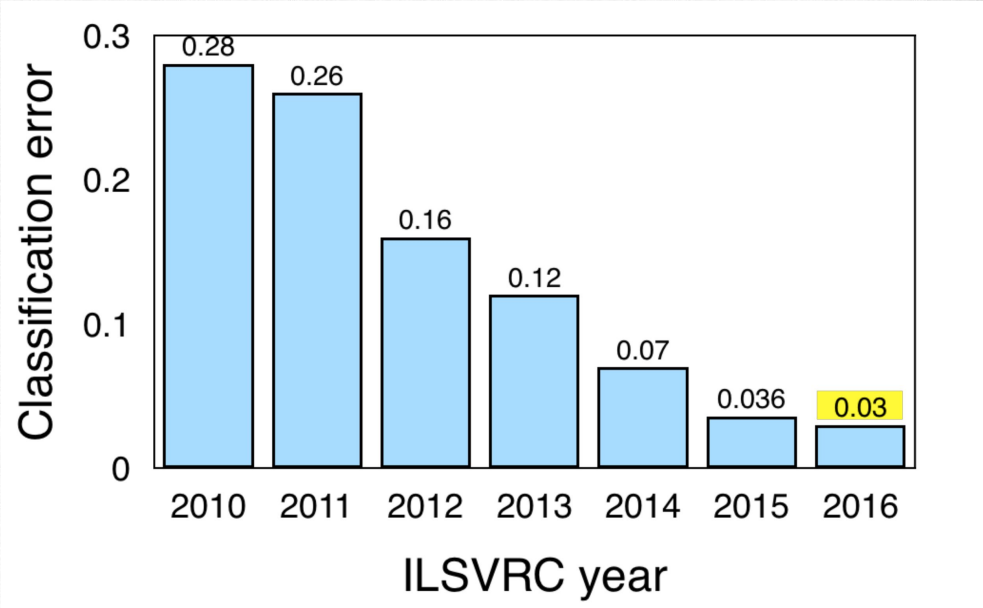
- Large-scale image recognition challenge
  - Major computer vision benchmark
  - Running since 2010 (Stanford, UNC)
  - http://www.image-net.org/challenges/LSVRC/
- 1.4M images, 1000 classes
- Main tasks
  - classify an image into 1 of the classes
    - top-1 error
      - predicted class should be correct
    - top-5 error
      - predict 5 classes, the correct one should be among them
  - detect all objects in an image



ImageNet Challenge

Overview of the classification task

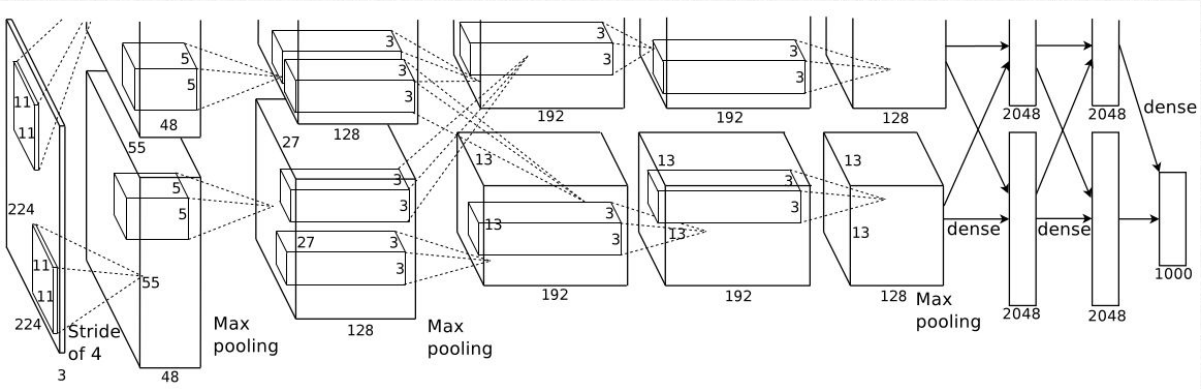
- 2010-11: hand-crafted computer vision pipelines
- 2012-2016: ConvNets
  - 2012: AlexNet
    - major deep learning success
  - 2013: ZFNet
    - improvements over AlexNet
  - 2014
    - VGGNet: deeper, simpler
    - InceptionNet: deeper, faster
  - 2015
    - ResNet: even deeper
  - 2016
    - ensembled networks, results have saturated



Case study 2: AlexNet

Krizhevsky et al., 2012

224x224x3 RGB input



1000 class likelihoods

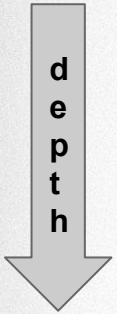
- 8-layer ConvNet: 5 conv layers, 3 fc layers
- Ingredients for success
  - Architecture
    - ReLU non-linearities
    - regularisation: dropout, weight decay (L_2 penalty)
  - Infrastructure
    - large dataset with random augmentation
    - two GPUs (model split across GPUs), 6 days of training

two important components
of successful deep learning models:
architecture and **infrastructure**

Case study 2: AlexNet

Krizhevsky et al., 2012

layer	output size
input image	224x224x3
conv-11x11x96/4	56x56x96
maxpool/2	28x28x96
conv-5x5x256	28x28x256
maxpool/2	14x14x256
conv-3x3x384	14x14x384
conv-3x3x384	14x14x384
conv-3x3x256	14x14x256
maxpool/2	7x7x256
fc-4096	4096
fc-4096	4096
fc-1000	1000



With depth: higher-level representations, more spatial invariance

- spatial resolution is reduced
- #channels is increased

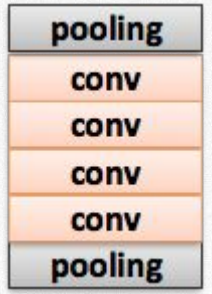
Linear layers at the bottom of AlexNet contain a lot of parameters

Deeper is Better

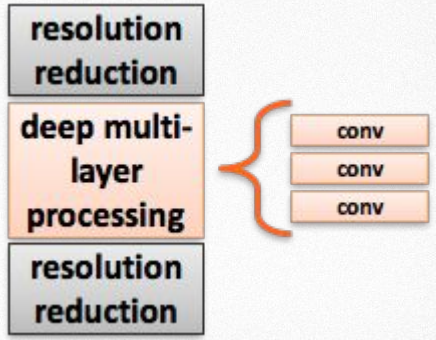
- Each weight layer performs a linear operation, followed by non-linearity
  - layer can be seen as a linear classifier itself
- More layers – more non-linearities
  - leads to a more discriminative (more powerful) model
- What limits the number of layers in ConvNets?
  - early ConvNet models used pooling after each conv. layer
    - input image resolution sets the limit: log(N) for N×N input
  - computational complexity

Building Very Deep ConvNets

- Stack several conv. layers between pooling
  - #conv. layers >> #pooling layers
  - #conv. layers will not affect resolution if each layer preserves spatial resolution
  - stride = 1 & input padding ('SAME' convolution)

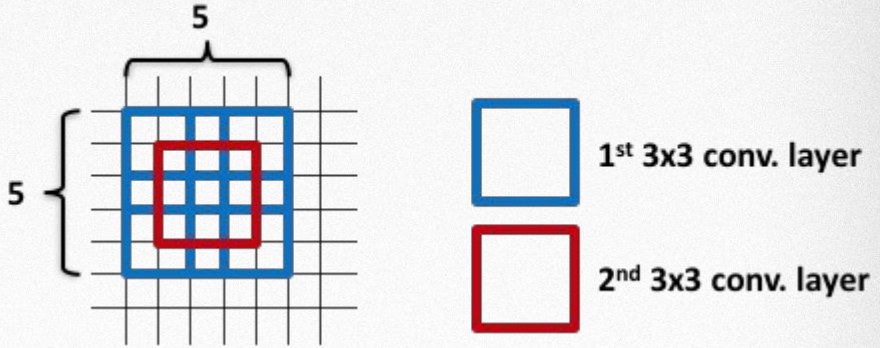


- More generally, interleave deep multi-layer blocks with resolution reduction layers
  - strided conv instead of pooling

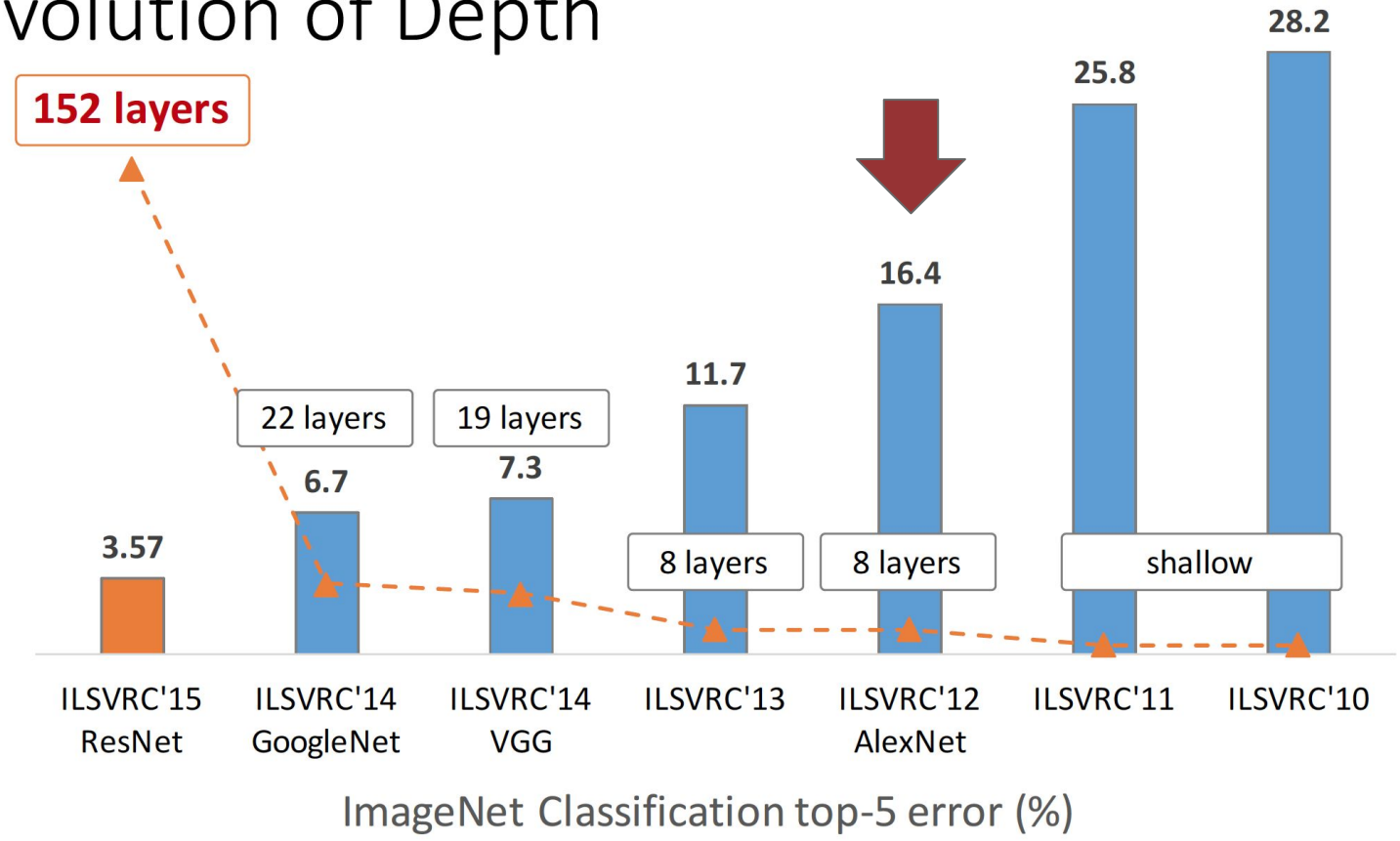


Building Very Deep ConvNets

- Use stacks of small (3×3) conv. layers
  - in most cases, the only kernel size you need
  - a cheap way of building a deep ConvNet
- Stacks have a large receptive field
  - two 3×3 layers – 5×5 field
  - three 3×3 layers – 7×7 field
- Less parameters than a single layer with a large kernel

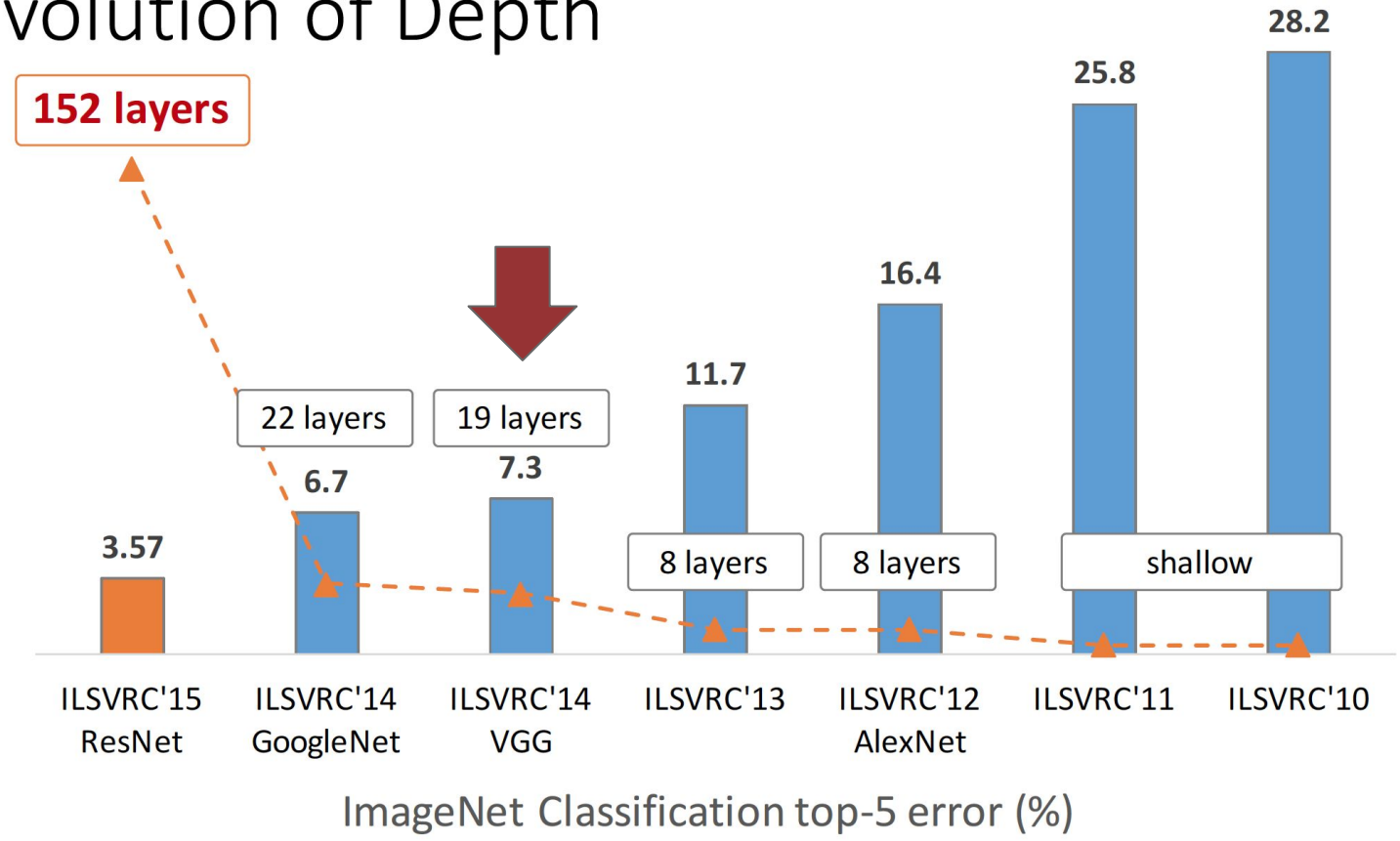


Revolution of Depth



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Revolution of Depth

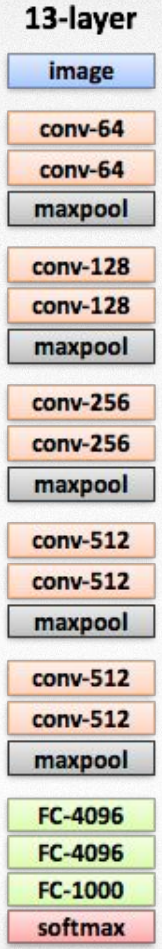


Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

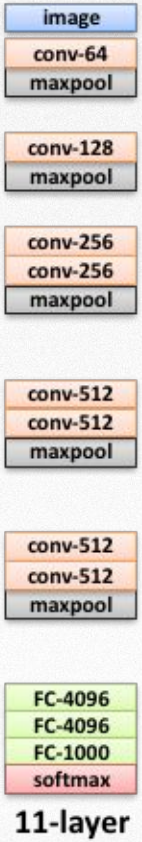
Case study 3: VGGNet

Simonyan & Zisserman, 2014

- Straightforward implementation of very deep nets:
  - stacks of conv. layers followed by max-pooling
  - 3x3 conv. kernels, stride=1
  - ReLU non-linearities
  - regularisation: dropout, weight decay (L2 penalty)
- A family of architectures
  - derived by injecting more conv. layers
- Infrastructure
  - trained on 4 GPUs (training data split across GPUs)
  - 2-3 weeks

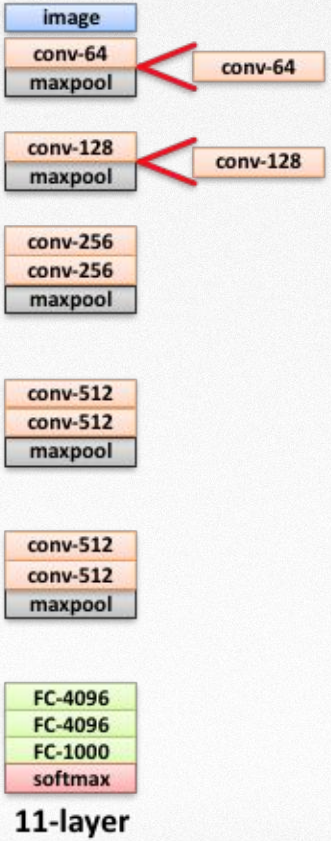


VGGNet Incarnations



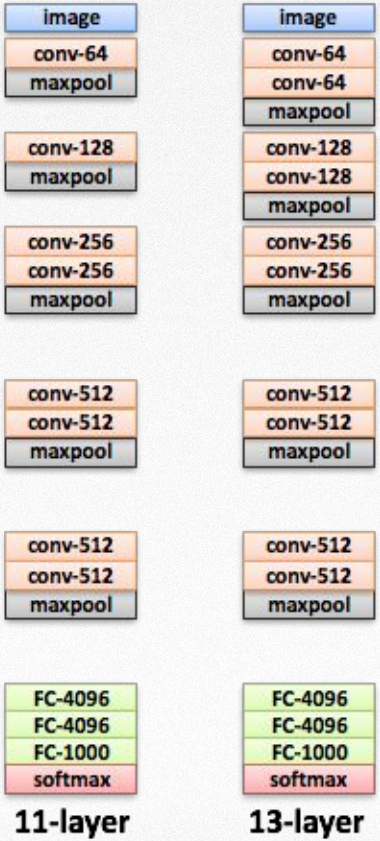
- Started from 11 layers

VGGNet Incarnations

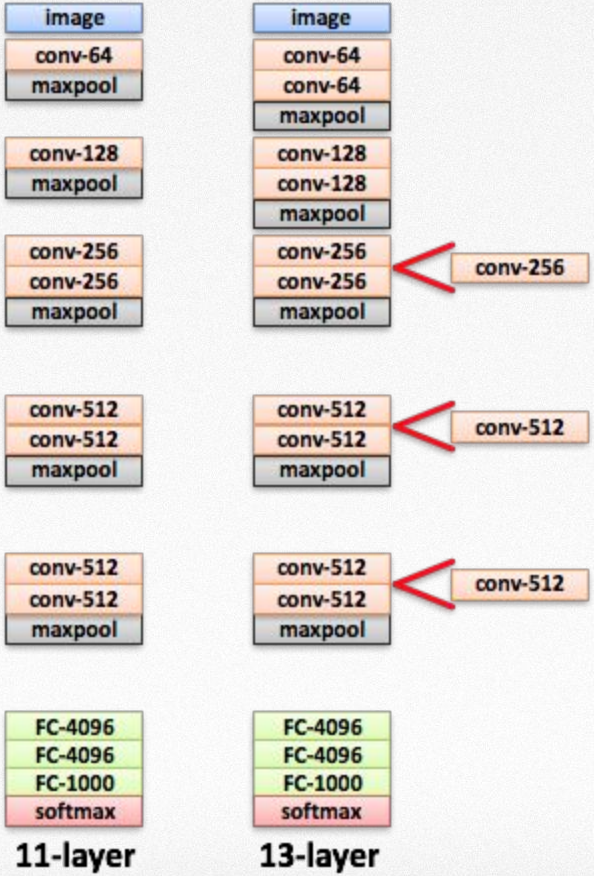


- Started from 11 layers & injected more conv. layers

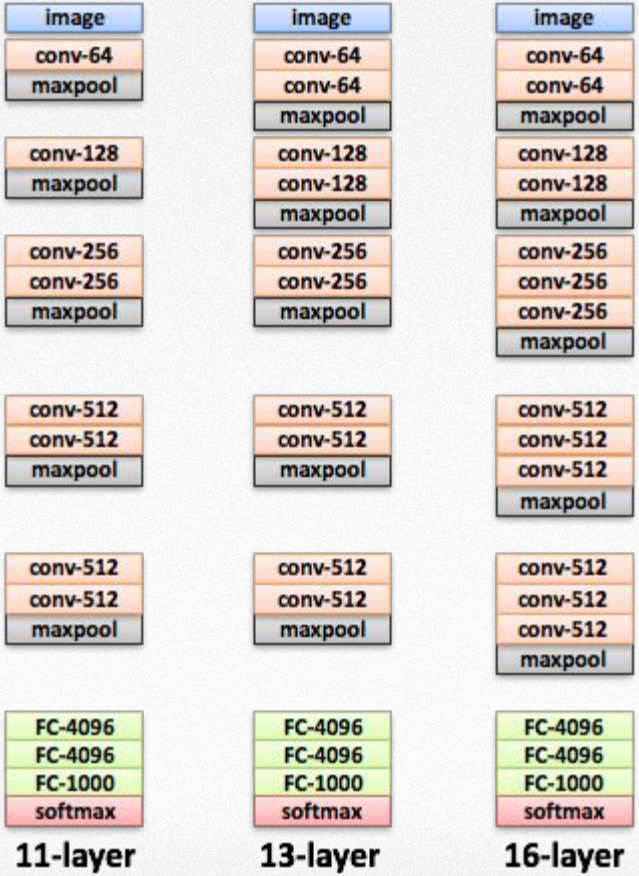
VGGNet Incarnations



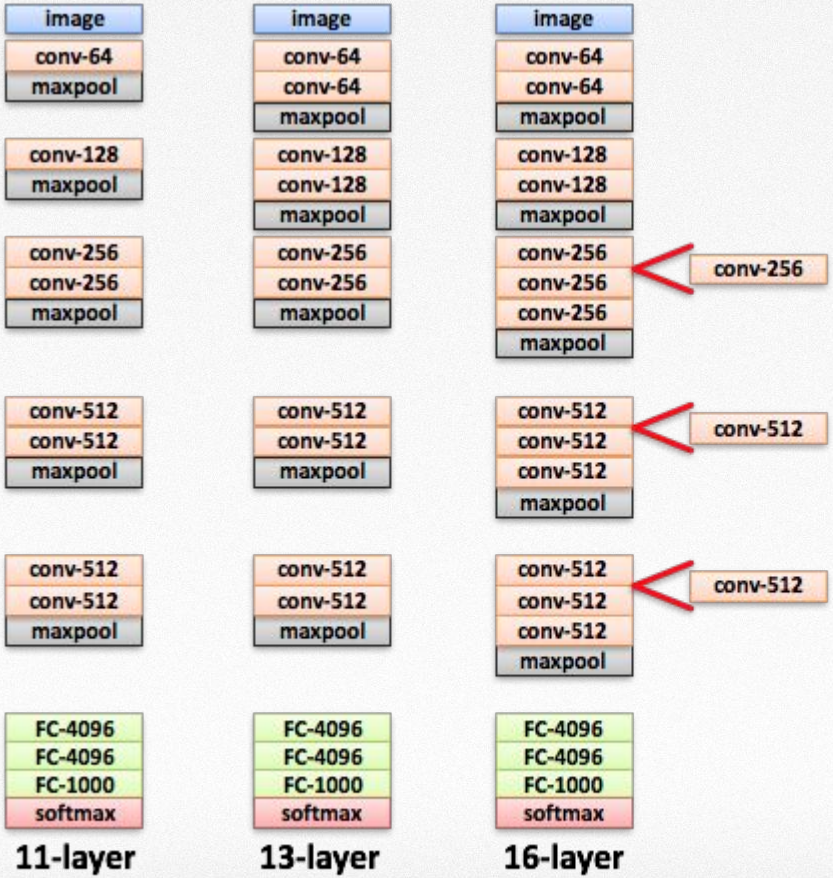
VGGNet Incarnations



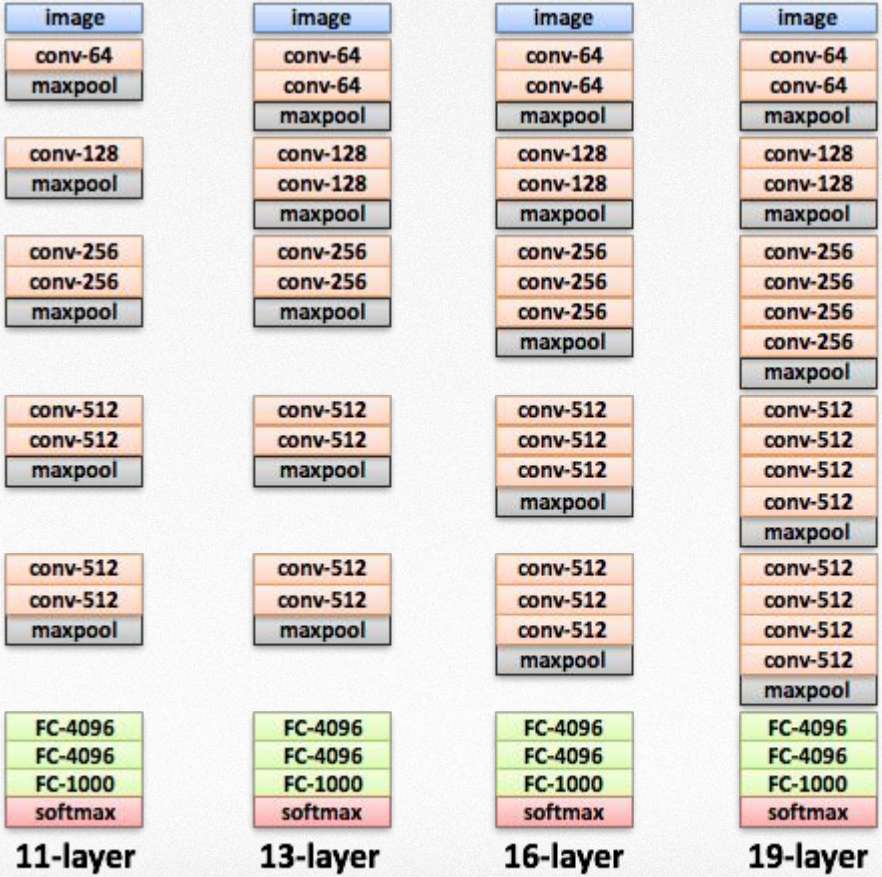
VGGNet Incarnations



VGGNet Incarnations



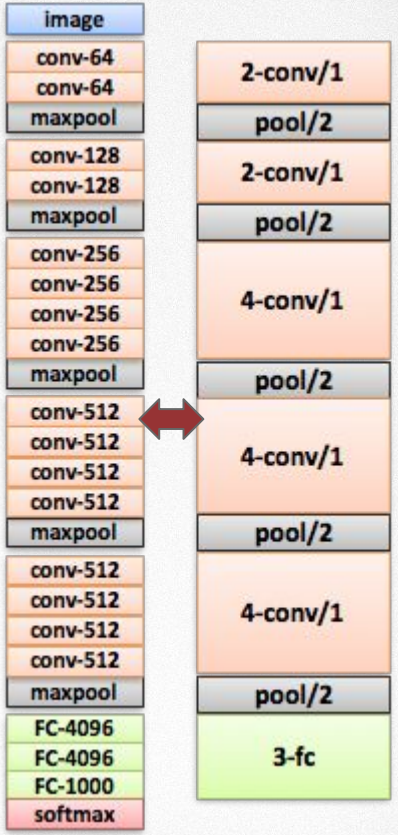
VGGNet Incarnations



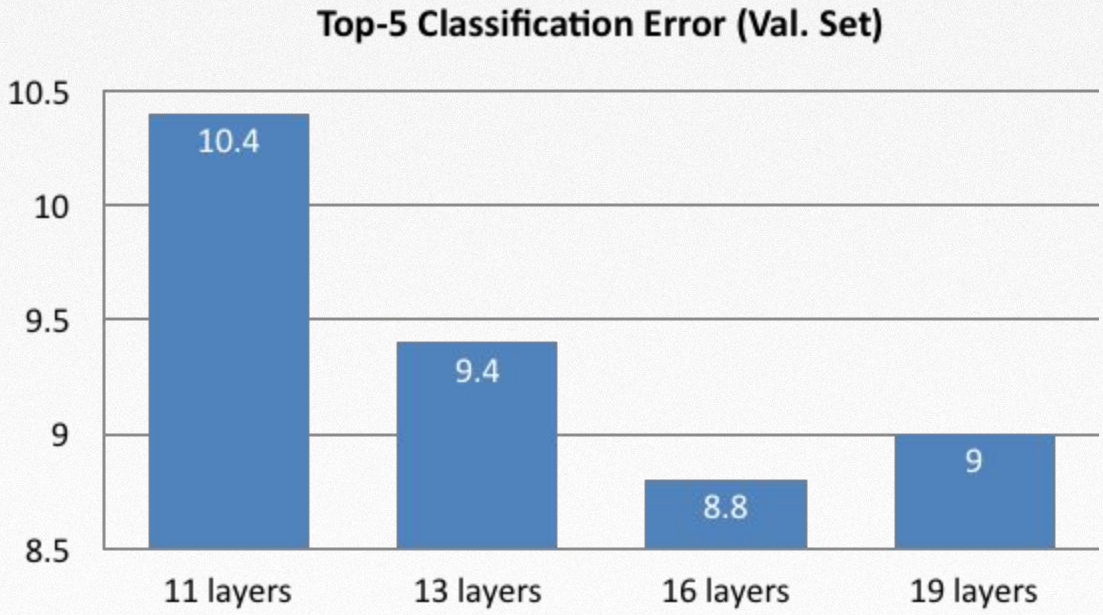
VGGNet Layer Pattern

- Multi-layer stacks (conv. layers, stride=1) interleaved with resolution reduction (max-pooling, stride=2)

- Other very deep nets (discussed later) follow a similar pattern



VGGNet Error vs Depth



- Error reduces with depth
- Plateaus after 16 layers
  - we'll discuss how to fix that

Summary

Convolutional networks

- image classification ConvNets
  - stacked conv. layers
  - occasional pooling (or strided conv)
  - fully-connected layers in the end
- 3×3 conv. layer - main workhorse of very deep ConvNets

QUESTIONS?

BREAK

Going Deeper

challenges of training very deep ConvNets and how to solve them

Challenges of training very deep ConvNets

- We have seen that depth is important
- Why not to keep adding layers to VGGNet?

Two main reasons:

- computational complexity
  - ConvNet will be too slow to train and evaluate
- optimisation
  - we won’t be able to train such nets

Optimisation

- Model optimisation (cf. James Martens' talk) is important
  - some architectures are hard to train – in particular very deep nets
- A plethora of gradient-based optimisation methods
  - in common
    - end-to-end training with cross-entropy ("softmax") loss for classification
    - weight gradients computed with back-prop
  - weight update rules are different: SGD, rms-prop, Adam, etc.
  - SGD with momentum – typical choice for ConvNets
- Major problem: gradient instability
  - when we backprop through many layers, compute a product of weights
  - if the weights are small, the gradients vanish (get too small)
  - if the weights are large, the gradients explode (get too large)

How to fix gradient instability?

No universal solution, but several ways to address it:

- Careful weight initialisation
  - start from the weights which lead to stable training

- Network design
  - add modules which ensure that the gradient doesn’t vanish

Weight Initialisation

- Sample from zero-mean normal distribution
- How to set the variance?
- Same variance for all layers, e.g. 0.01
  - works fine for shallow nets (e.g. AlexNet)
  - deeper nets suffer from vanishing gradient
- Adaptively choose variance for each layer
  - preserve gradient magnitude [Glorot & Bengio, 2010]: 1/sqrt(fan_in)
  - FC layers: fan_in = #input channels
  - conv. layers: fan_in = #input channels × size × size
  - works fine for VGGNets (up to 20 layers), but not sufficient for deeper nets

Batch Normalisation

- Motivation: the distribution of activations changes during training, making it harder
- Batchnorm layer normalises the input to zero mean and unit variance
- Can be placed anywhere in the network
  - typically after each conv layer before activation

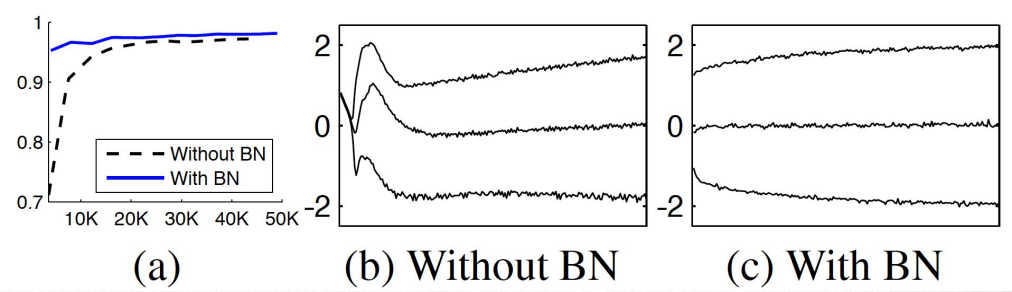


Figure 1: (a) *The test accuracy of the MNIST network trained with and without Batch Normalization, vs. the number of training steps. Batch Normalization helps the network train faster and achieve higher accuracy.* (b, c) *The evolution of input distributions to a typical sigmoid, over the course of training, shown as* $\{15, 50, 85\}$*th percentiles. Batch Normalization makes the distribution more stable and reduces the internal covariate shift.*

Batch Normalisation (2)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

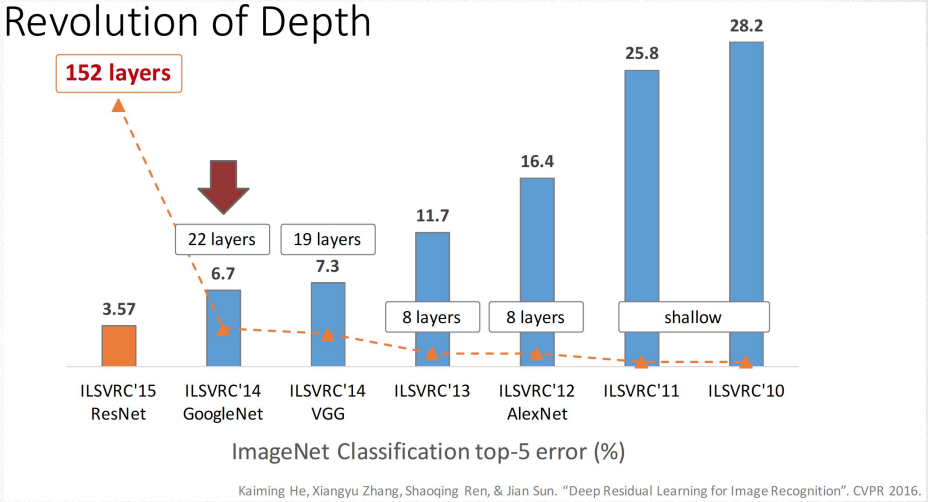
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- Requires batched training
- Batchnorm is differentiable
- Means and variances are (slightly) different for different batches
  - adds randomness, which is a good regulariser
  - nets with batchnorm need less regularisation, dropout is rarely needed
- Less sensitive to initialisation, can use N(0, 0.01)

Case Study 4: InceptionNet

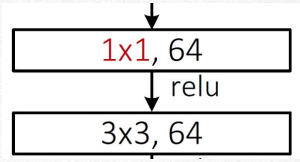
Szegedy et al., 2014-16

- A family of models
  - several versions (v1-v4)
  - v1 won the classification task of ImageNet-2014
- Objectives
  - high accuracy
  - high speed and low #parameters
- Achieved by careful hand-crafting of layer configuration
- Starting from v2, uses batchnorm to facilitate training of deeper models



1x1 Convolution

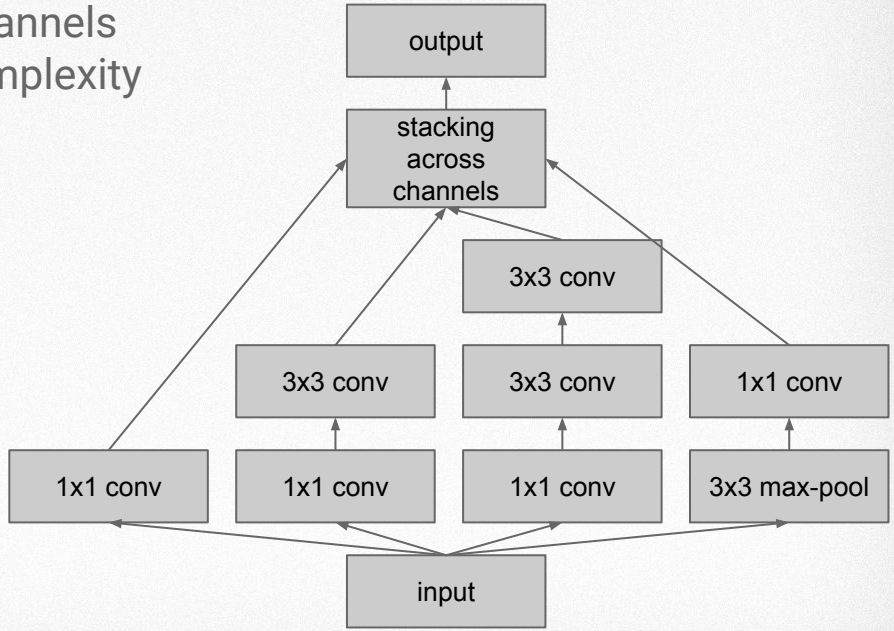
- “Bottleneck” layer
- Doesn’t capture spatial context, only operates across channels
  - linear projection of channels for each pixel
- Reduces #channels before slower layers, e.g. 3×3 conv
  - network is faster, has less parameters
- Also increases the depth
  - network is more discriminative
- Very useful trick used in many architectures (Inception, ResNet, etc.)



Inception Module

building block of InceptionNet

- Recall: a neural net can take the form of a DAG (directed acyclic graph)
- Inception module contains several branches
  - each with a different #layers and #channels
  - carefully balanced to have similar complexity
  - more channels for 1×1 conv
  - less channels for 3×3 conv
- Branches are combined by stacking
- “Bottleneck” 1×1 conv

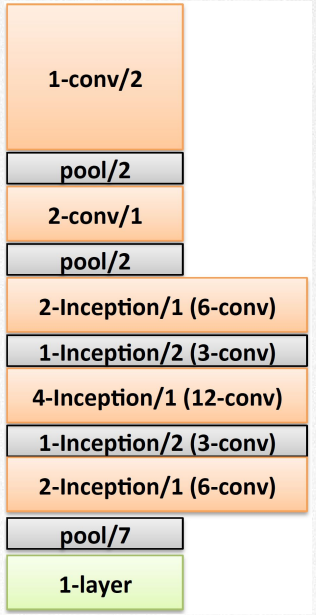


Inception module v2

Inception Net v2

- 4.8% top-5 error, 25.2% top-1 error
- 10 Inception modules (34 layers deep)
- Network structure
  - spatial downsampling by strided conv layers
  - followed by more complex Inception modules
  - global average pooling
  - single linear layer - reduction in #parameters

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	double #3×3 reduce	double #3×3	Pool +proj
convolution*	7×7/2	112×112×64	1						
max pool	3×3/2	56×56×64	0						
convolution	3×3/1	56×56×192	1		64	192			
max pool	3×3/2	28×28×192	0						
inception (3a)		28×28×256	3	64	64	64	64	96	avg + 32
inception (3b)		28×28×320	3	64	64	96	64	96	avg + 64
inception (3c)	stride 2	28×28×576	3	0	128	160	64	96	max + pass through
inception (4a)		14×14×576	3	224	64	96	96	128	avg + 128
inception (4b)		14×14×576	3	192	96	128	96	128	avg + 128
inception (4c)		14×14×576	3	160	128	160	128	160	avg + 128
inception (4d)		14×14×576	3	96	128	192	160	192	avg + 128
inception (4e)	stride 2	14×14×1024	3	0	128	192	192	256	max + pass through
inception (5a)		7×7×1024	3	352	192	320	160	224	avg + 128
inception (5b)		7×7×1024	3	352	192	320	192	224	max + 128
avg pool	7×7/1	1×1×1024	0						

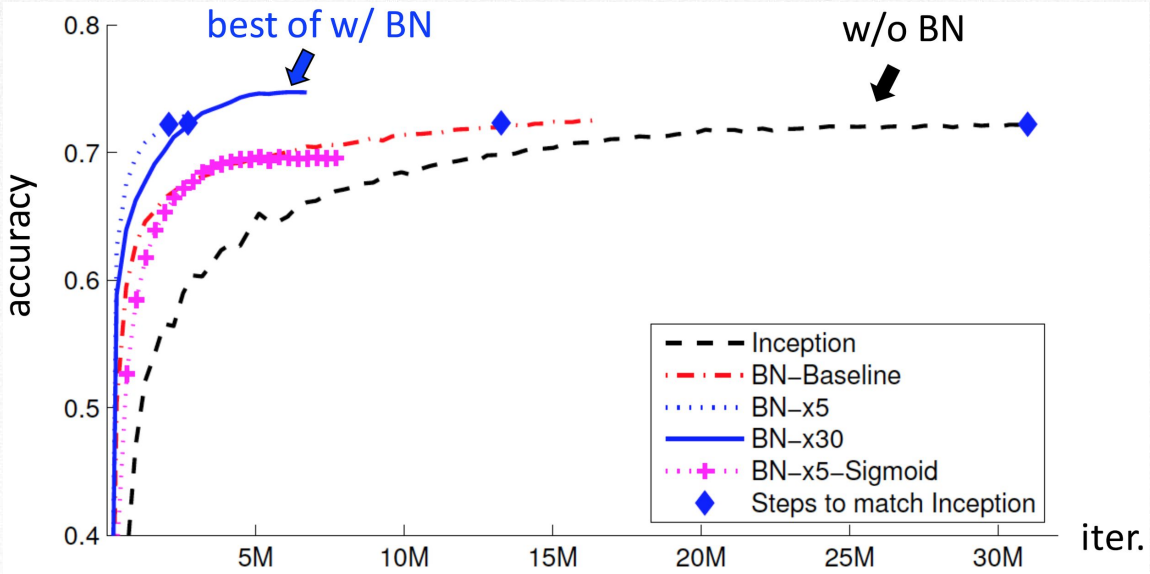


Inception v2

Inception Net v2

Importance of batch normalisation

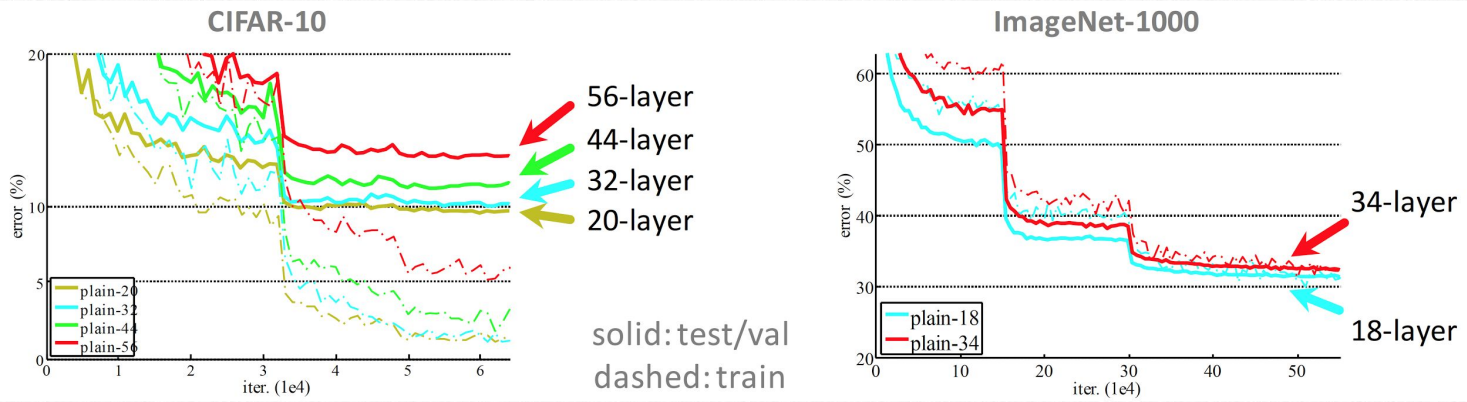
- Higher accuracy and faster training with batchnorm



Residual Connections

Motivation

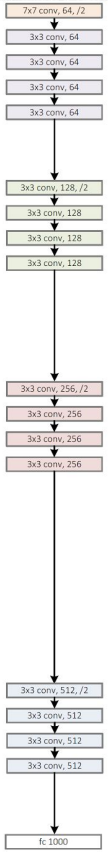
- Construction to facilitate training of ultra deep nets (100-1000 layers)
  - complementary to batchnorm
- Motivation: after certain depth, deeper nets have higher **training** error

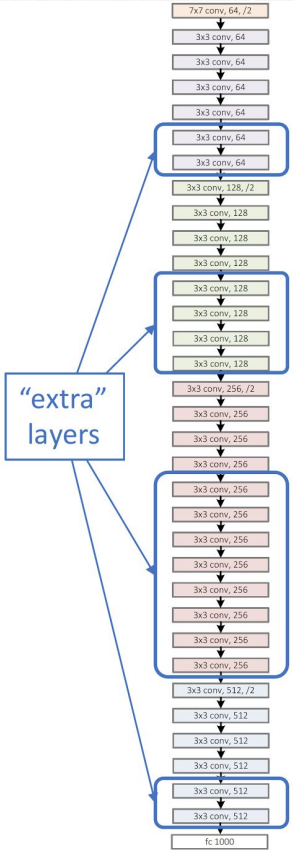


error curves for VGG-like nets (3×3 conv throughout)
with batchnorm

Residual Connections

a shallower model (18 layers)



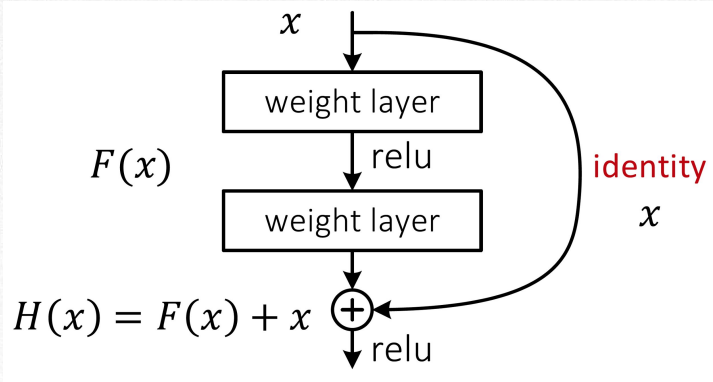


a deeper counterpart (34 layers)

- Richer solution space
- A deeper model should not have **higher training error**
- A solution *by construction*:
  - original layers: copied from a learned shallower model
  - extra layers: set as identity
  - at least the same training error
- Optimization difficulties: solvers cannot find the solution when going deeper…

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Residual Connections



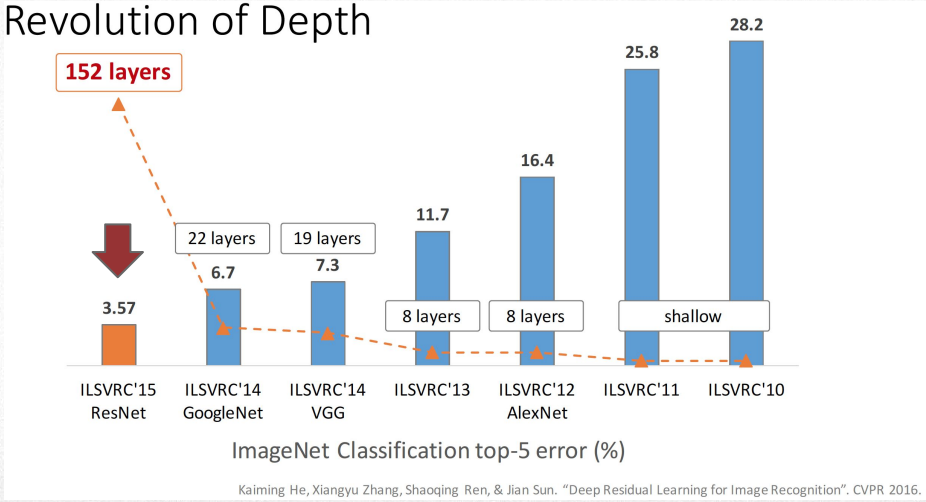
$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial H}\frac{\partial H}{\partial x} = \frac{\partial L}{\partial H}\left(\frac{\partial F}{\partial x} + 1\right)$$

- Identity connection which **skips** a few layers
- We only need to learn the **residual**
- Becomes easier to learn identity, if need to
  - just set the weights to 0
- Backprop perspective
  - gradient skips weight layers – no vanishing
  - improves gradient flow through layers

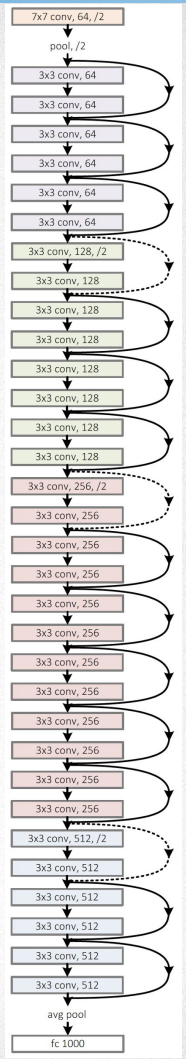
Case Study 5: ResNet

He at al., 2015-16

- A family of models
  - won the classification task of ImageNet-2015
- Simple network design
  - inspired by VGGNet, but **10x deeper**
  - **residual connections** & batchnorm

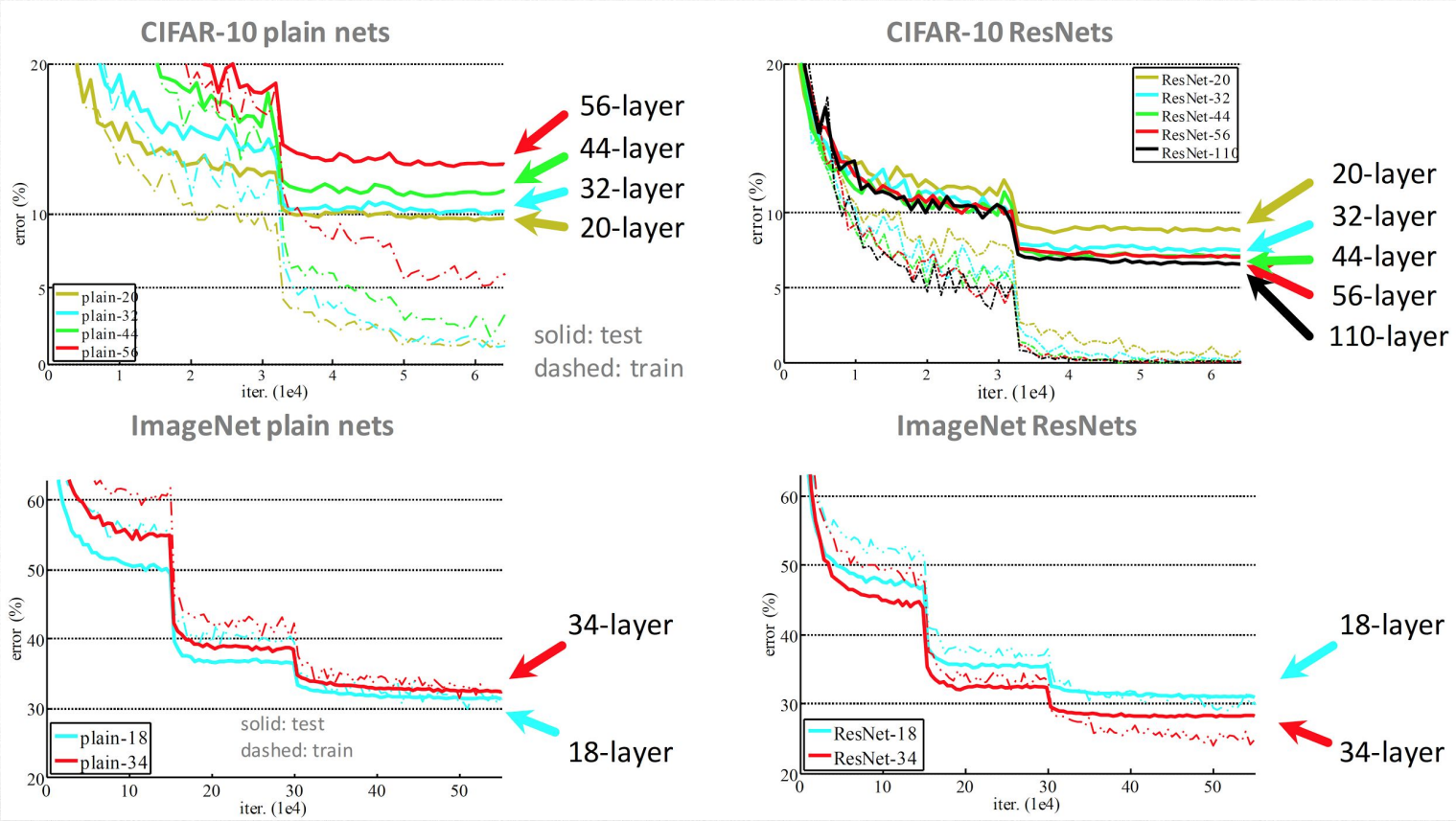


Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.



ResNets

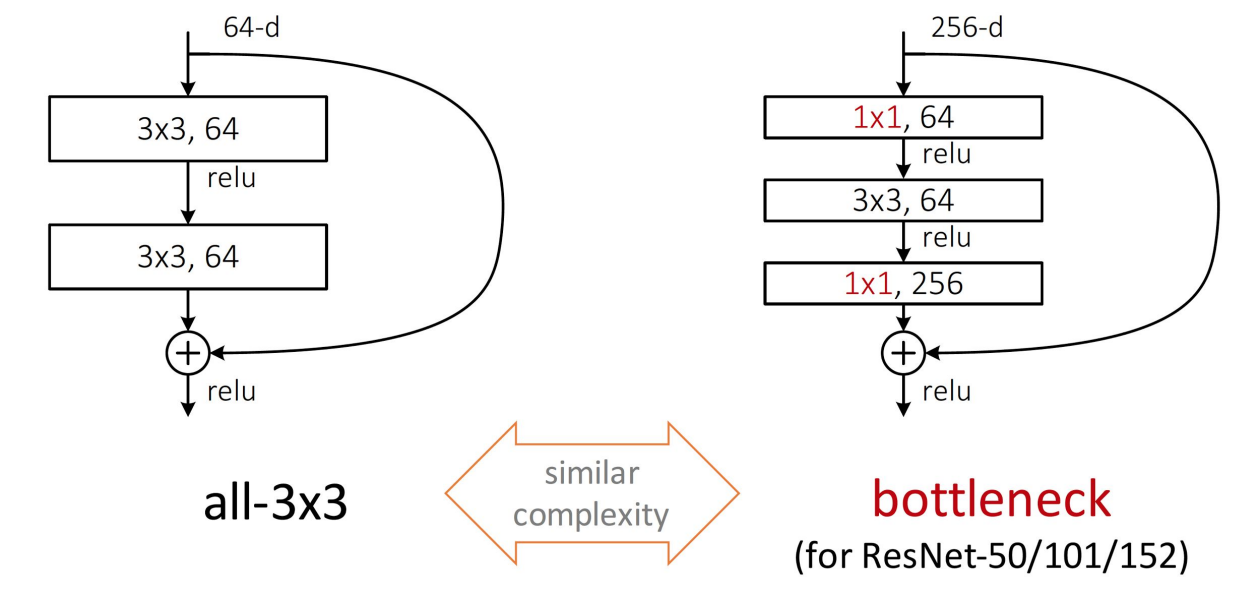
Deeper ResNets have lower training and test errors



Bottleneck Residual Block

Tackling computational complexity

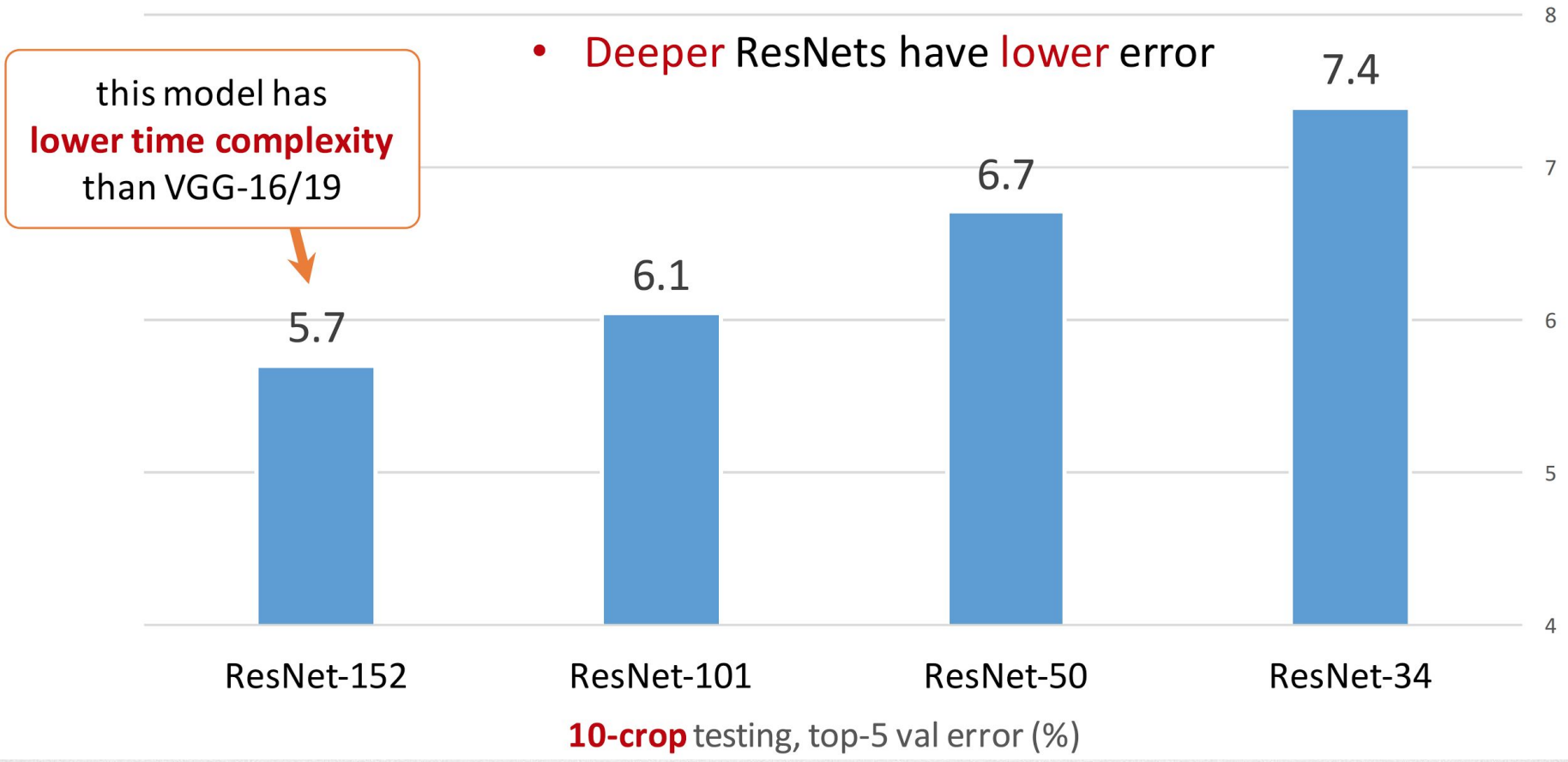
- Recall: bottleneck (1x1 conv) layers in Inception
- Also used in ultra deep ResNets before 3x3 conv
- Increases the depth w/o complexity increase



ResNet Layer Configuration

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

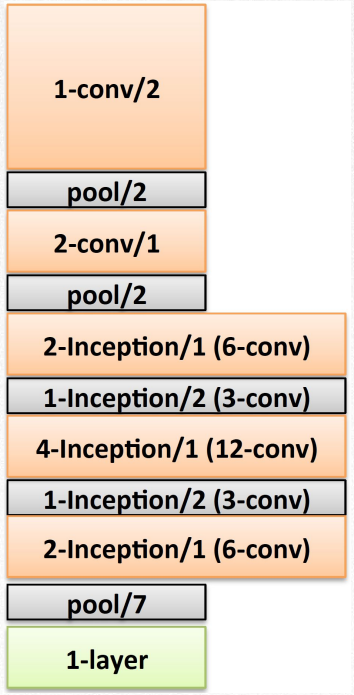
ResNet ImageNet Results



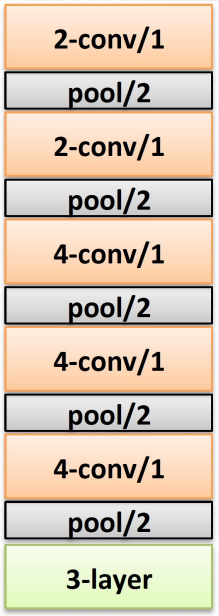
Architecture Comparison

Inception vs ResNet vs VGGNet

- Inception & ResNet
  - deeper than VGGNet
  - less deep at high resolution
  - deeper at low resolution
  - strided convolution instead of pooling
    - learnt pooling
  - global pooling in the end
    - 1 linear layer vs 3 in VGG
    - less parameters
  - both use batchnorm
- Inception vs ResNet
  - ResNet has residual connections
  - much simpler (VGGNet-like) structure



Inception v2



VGG-19

Summary

challenges of training very deep ConvNets

Main challenges and their solutions:

- computational complexity
  - use 3×3 conv (or even 3×1 and 1×3)
  - use 1×1 conv bottlenecks (also increase depth)
- optimisation
  - batch normalisation after each weight layer
  - residual connections to skip 2-3 weight layers

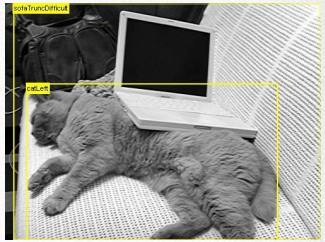
QUESTIONS?

Beyond ImageNet Classification

Pretraining

- Training large models on large datasets takes a lot of time
  - several weeks on multiple GPUs for ImageNet ConvNets
- Network trained on a large diverse dataset (ImageNet) should be useful for other data if there is a large semantic overlap
  - similar classes
  - similar domain (photos)

ImageNet cat



PASCAL VOC cat

- Make use of trained models instead of training from scratch:
  - Take a pretrained model
  - Plug into anto another network
  - Train remaining layers
  - Keep pre-trained weights fixed or slowly updated (fine-tuning)

Pretraining of Image Classification Nets

- Train ImageNet classification ConvNet
- Replace the last layer with a new one
  - #outputs = #classes in a new dataset
- Train the last layer

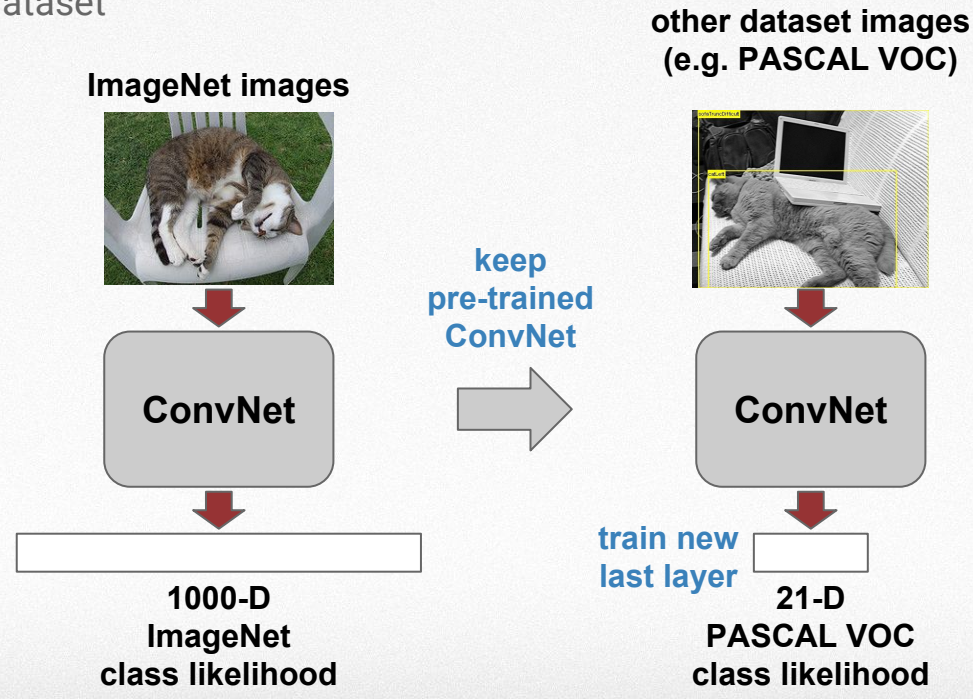
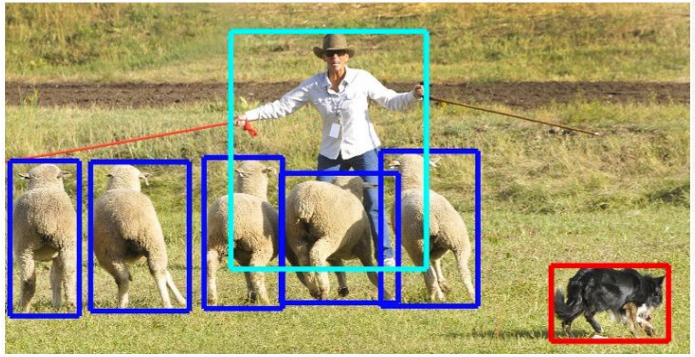


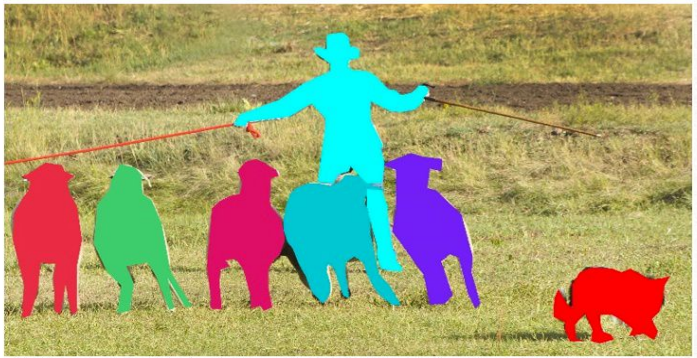
Image Recognition Tasks

(a) Image classification



(b) Object localisation/detection

(c) Semantic segmentation

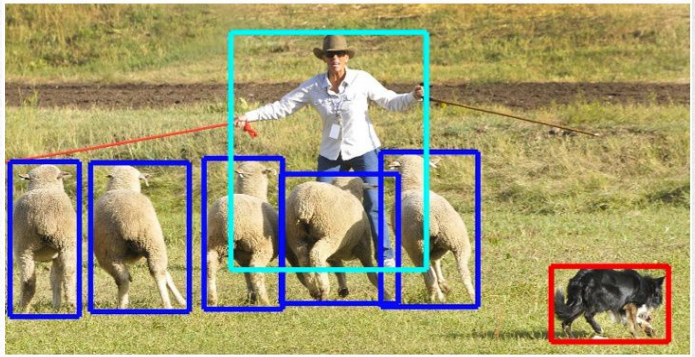


(d) Instance segmentation

Object Detection with ConvNets

Popular Approaches

- Detection as classification - classify all bounding boxes
  - too slow if done separately, but can be optimised
  - the same object might be detected multiple times



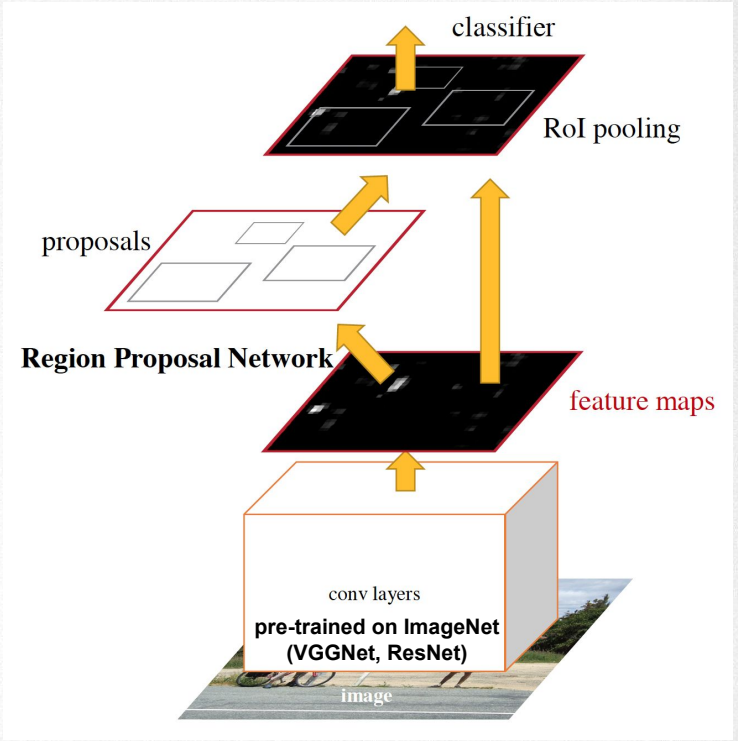
- Detection as bounding box prediction
  - regress 4 numbers - box coordinates (MSE loss)
  - the number of boxes is unknown

- Bounding box proposal
  - separate module (which can also be a ConvNet) predicts which boxes might contain objects
  - proposals are then passed through a classifier

Faster R-CNN: Overview

State of the art object detection

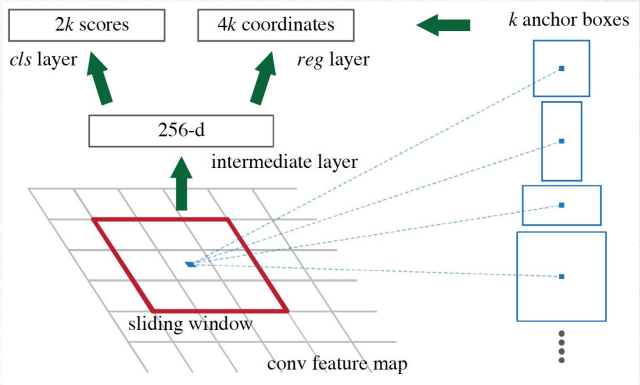
- Proposal-based object detection
- Conv layers pre-trained on ImageNet
- Proposal stage
  - region proposal sub-network produces possible object locations
- Final stage
  - classifies proposals into classes
  - further refines their location



Faster R-CNN: Overview

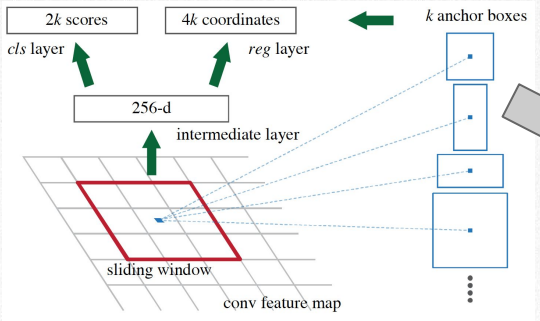
Region proposal network

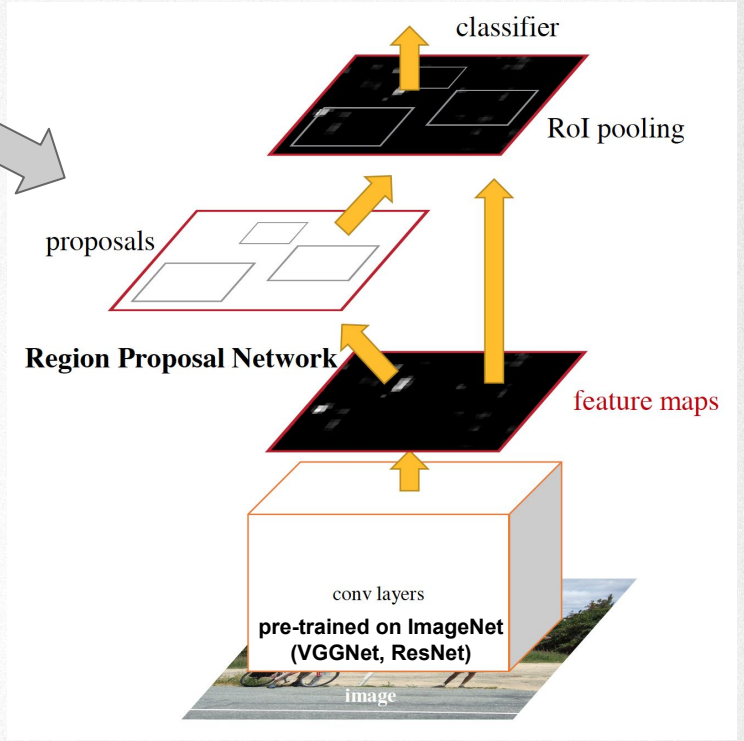
- Objective
  - for each location, for each of the k=9 anchors
    - predict proposal box location wrt the anchor
    - predict if the proposal contains *some* object
- Configuration: two conv layers
  - first: 3×3 conv with 256 channels
  - second: 1×1 conv with 6k output channels
    - 6 numbers for each of the k proposals



Faster R-CNN: Overview

State of the art object detection





- Trained end-to-end
  - cf. Raia Hadsell's talk
- Pre-trained conv layers are fine-tuned with a lower learning rate

Semantic Segmentation with ConvNets

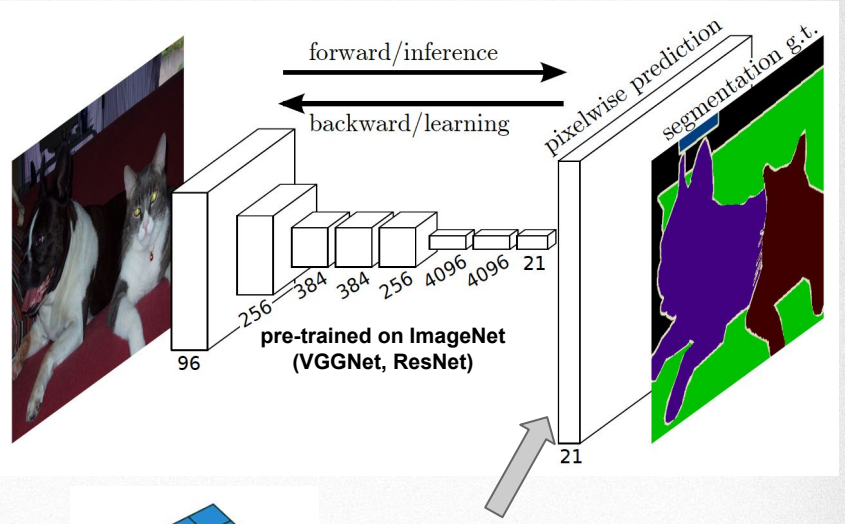
- Label each pixel into one of the object classes
- Popular approach: classify each pixel
- Network outputs a full resolution map S with #classes channels
  - $S_{i,j,c}$ - likelihood of pixel (i, j) having class c

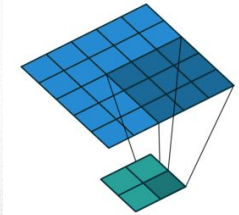
- Many architectural variations
- Popular resolution-preserving building blocks
  - pooling → transposed conv
  - conv/dilated conv, stride=1

Fully Convolutional Networks

from classification to segmentation

- ConvNet w/o linear layers (“fully convolutional”)
  - pre-trained on ImageNet classification
- Penultimate conv layer has 21 channel
  - 20 classes & background
- ConvNet contains pooling layers
  - which reduce resolution
  - compensated by transposed conv in the end



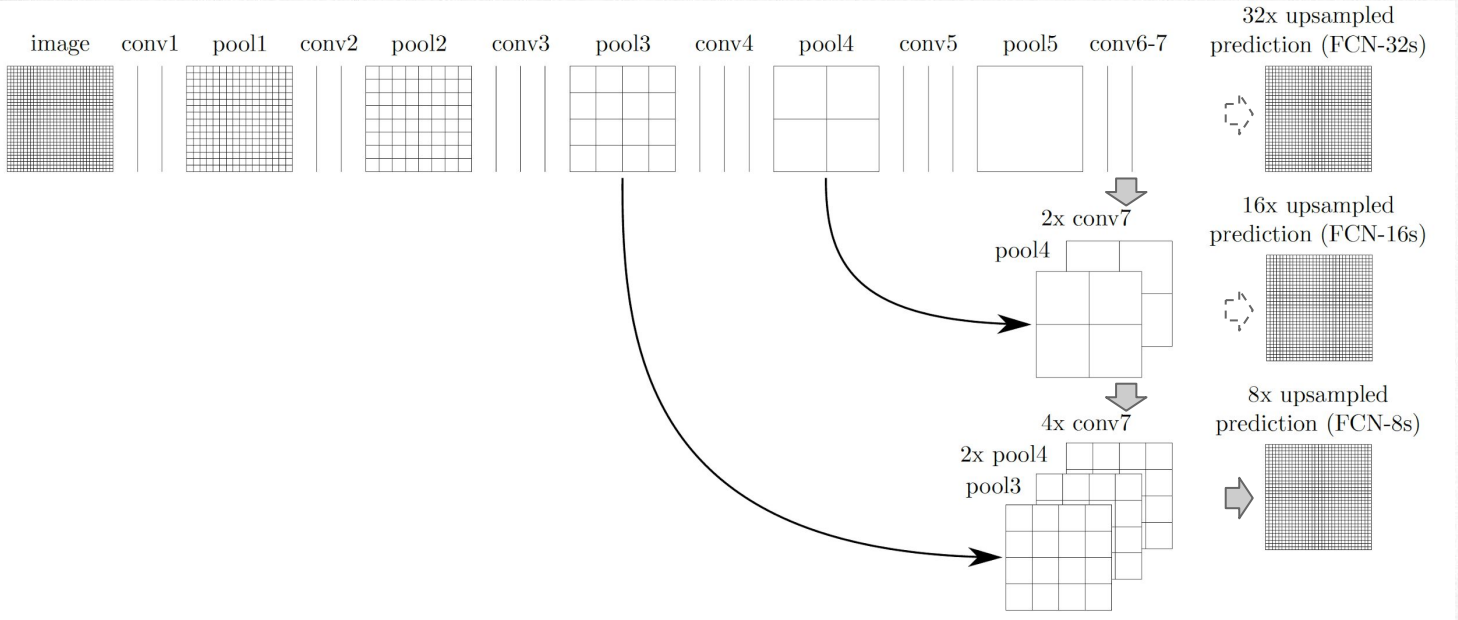


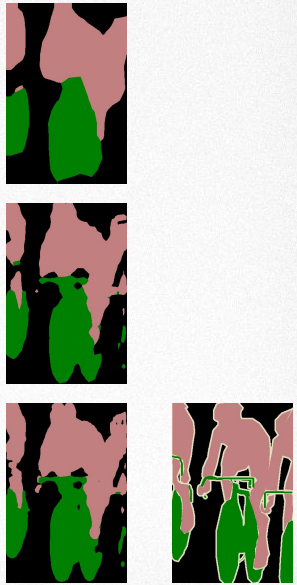
transposed conv increases spatial resolution

Fully Convolutional Networks

Combining high- and low-level cues

- skip connections from shallow (hi-res) layers to deep (low-res) layers
- fuse information from different layers



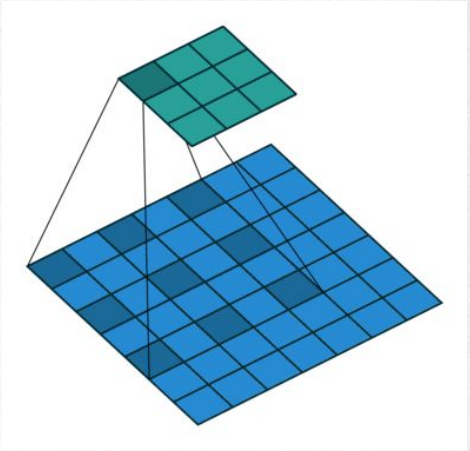


ground truth

Image Segmentation

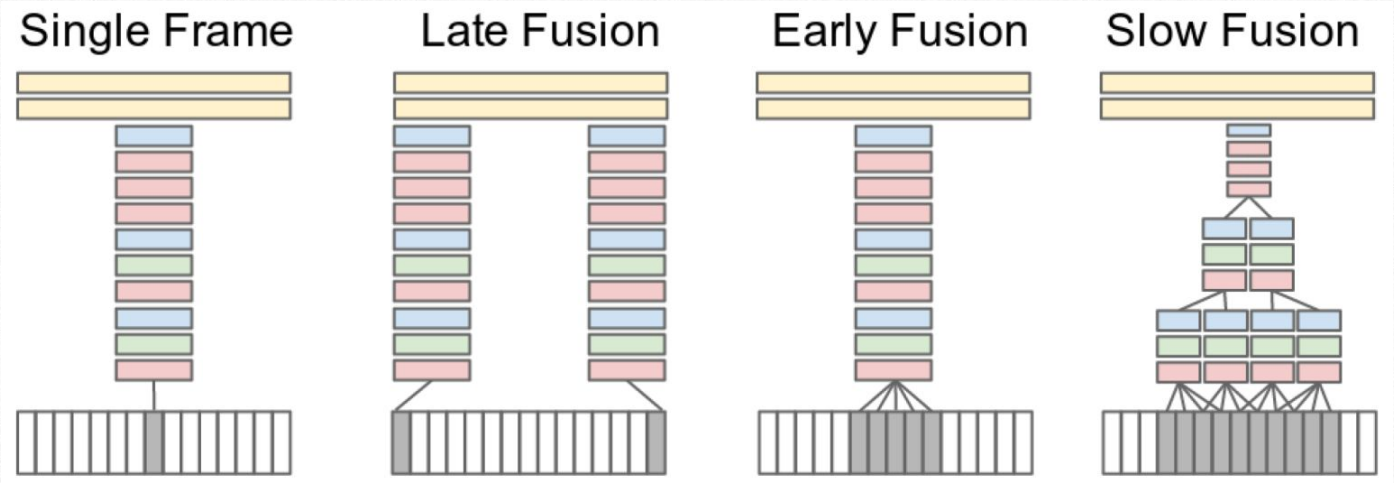
with dilated convolutions [Yu & Koltun, 2015]

- Recall: dilated convolution expands conv kernel
  - large receptive field
- ConvNet w/o linear layers
  - pre-trained on ImageNet classification (again!)
  - pooling removed
  - conv replaced with dilated conv
    - same receptive field
    - no resolution decrease
- Simpler and higher accuracy than the previous method



Video Classification with ConvNets

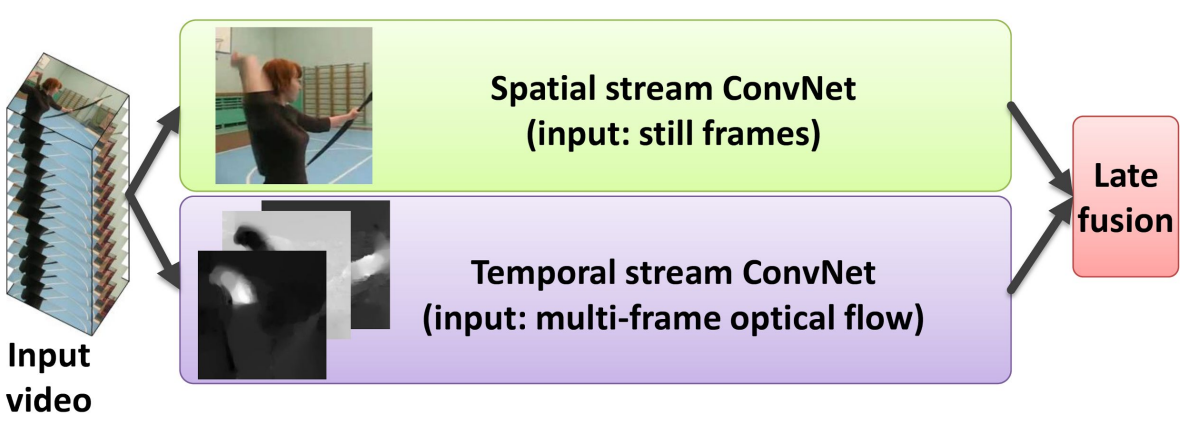
- ConvNets are applicable not only to images, but to other structured domains
- Two ways to incorporate temporal component
  - stack multiple frames into a volume (Karpathy et al., 2014)
  - process motion separately from appearance



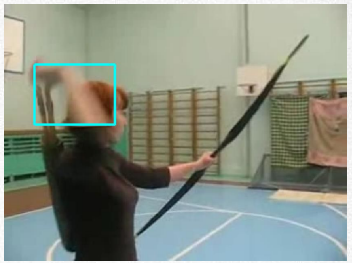
fusing information over multiple frames

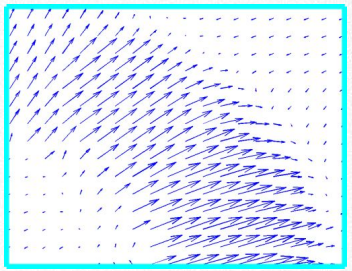
Two-Stream ConvNet for Video

Simonyan & Zisserman, 2014



- Appearance and motion are processed separately
- Spatial stream ConvNet
  - input: RGB frame
- Temporal stream ConvNet
  - input: motion vector field between several frames
- Each ConvNet can be pre-trained (again!)





optical flow

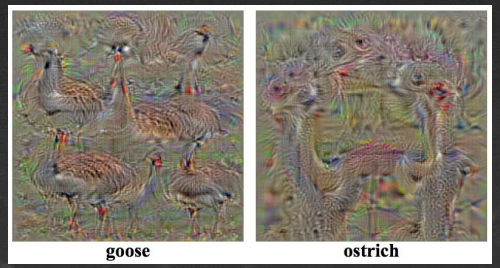
Summary

beyond image classification

- Pre-training on large datasets (ImageNet) gives a reusable core for various visual recognition networks
- Object detection nets
  - predict bounding box location and class
  - complex multi-stage / multi-output graphs
- Semantic segmentation nets
  - predict a class for each pixel
  - building blocks: dilated conv or transposed conv
- Video recognition nets
  - either stack frames or
  - separately process frames and optical flow, then combine

QUESTIONS?

Visualising ConvNets



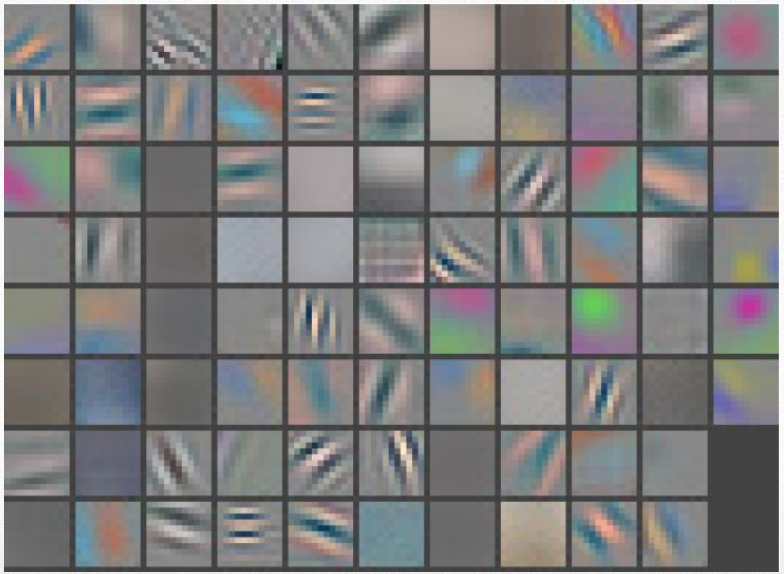
Visualising ConvNets

- So far we treated ConvNets as a blackbox
  - input: image
  - output: prediction

- Several ways to understand the model better
  - visualise weights
  - change the input, observe how it affects the output
  - synthesize inputs to activate particular parts

ConvNet Filter Visualisation

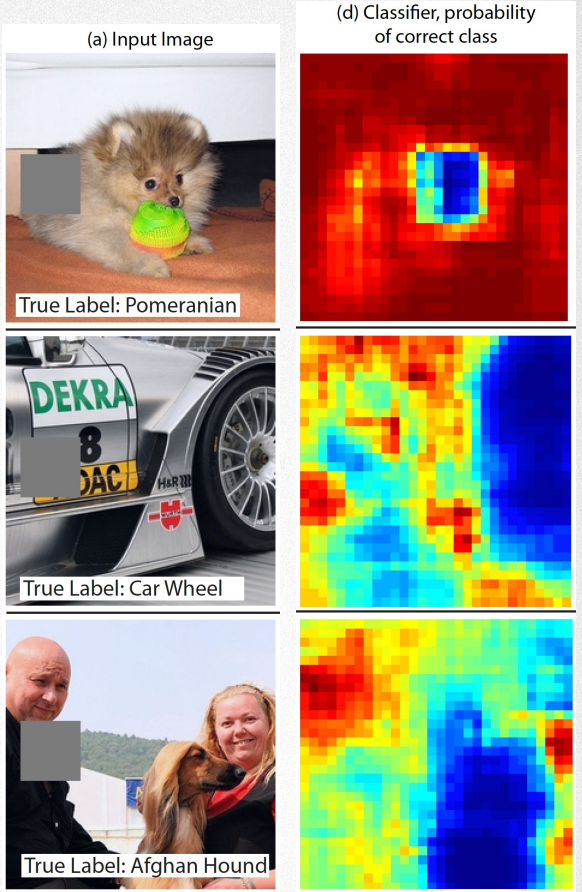
- The first layer operates on the input image
  - AlexNet: 96 filters, 11x11x3 each
- Can be visualised in the RGB domain as 11x11 patches
- Resembles Gabor filters



Sensitivity Analysis

Zeiler & Fergus, 2013

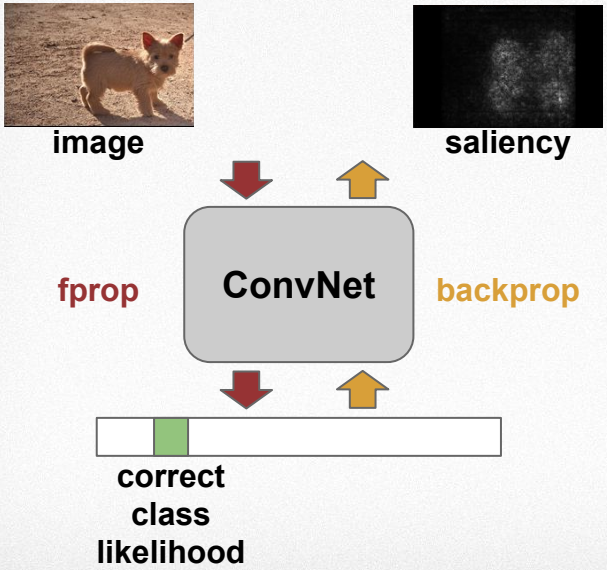
- Cover a part of an image with a grey square
- Move the square, observe how it affects the probability of correct image class
- The resulting map shows which image areas are important for classification
- Problems
  - re-evaluation is slow
  - map is coarse

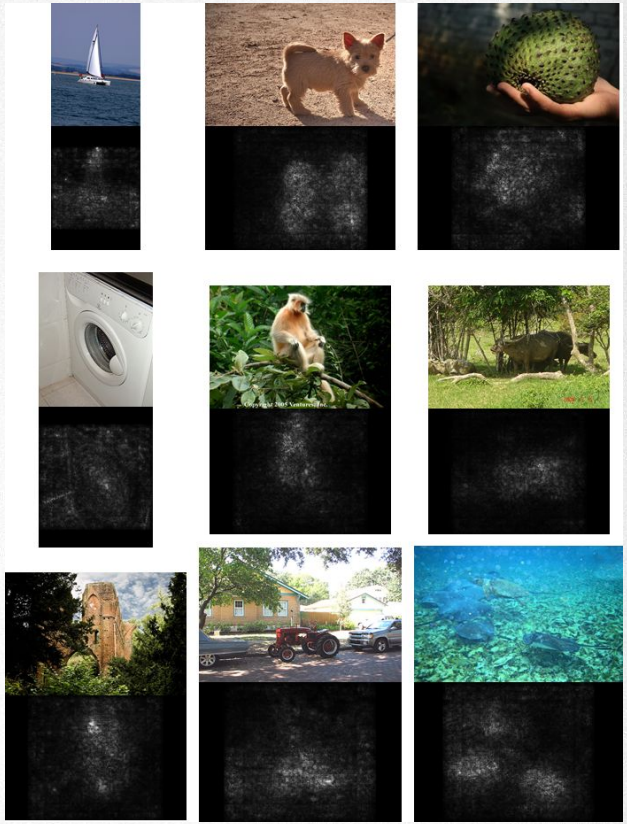


Visualisation using Backprop

Simonyan et al., 2013

- Class saliency: determine which image pixels affect class likelihood by back-propagation of the likelihood wrt **the image**

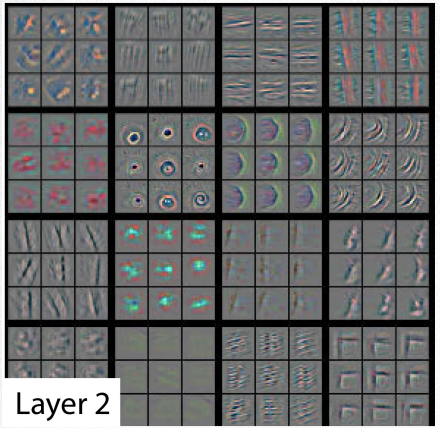


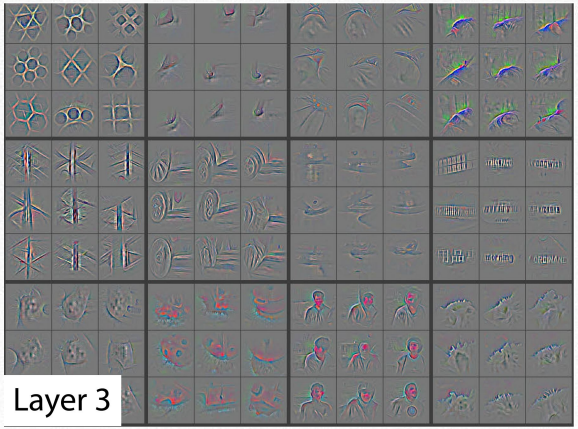


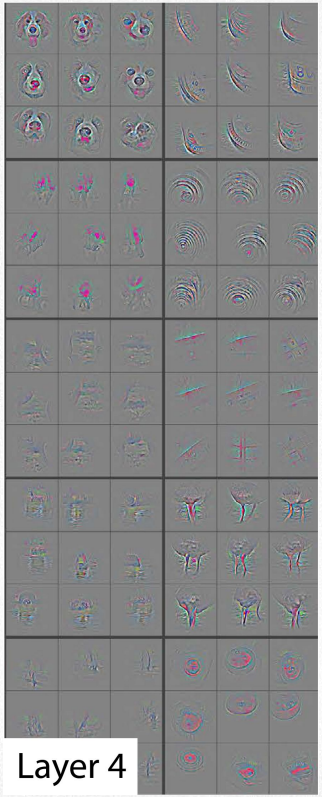
Visualisation using Backprop

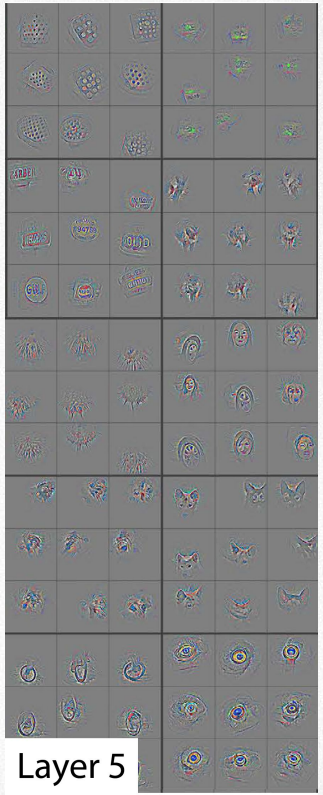
Zeiler & Fergus, 2013

- Works for hidden layers too
- Pick a high-magnitude activation and backprop wrt the image





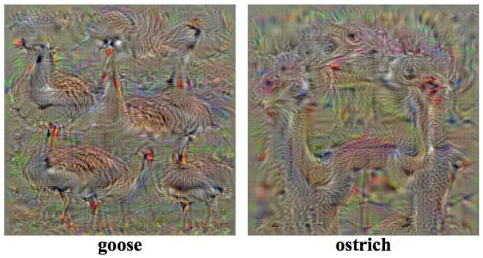


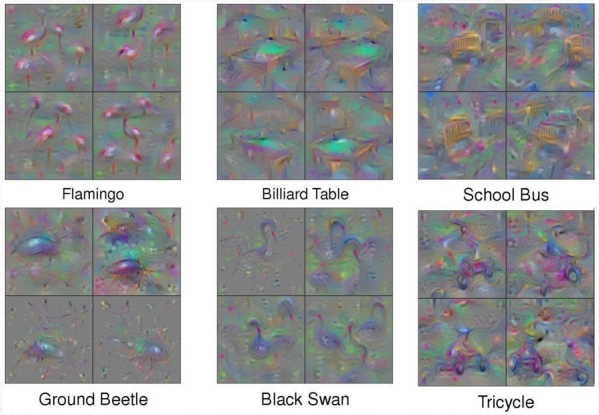


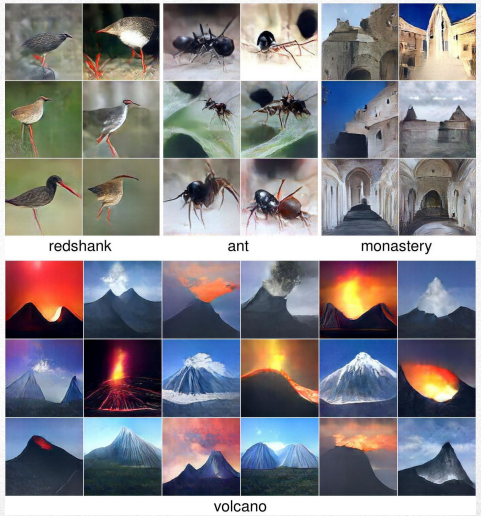
Visualisation using Backprop

[Simonyan et al., 2013], [Yosinski et al., 2015], [Nguyen et al., 2016]

- Canonical class images
- Gradient ascent wrt the image to find a canonical image of a class
  - when training neural nets, we optimise weights
  - here, keep weights fixed and optimise the input
- Important to regularise the image to make it look realistic





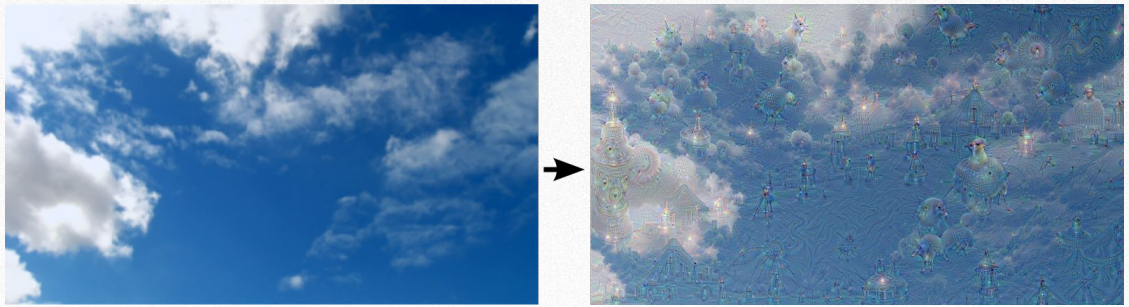


different regularisers, from simple to complex

Visualisation using Backprop

DeepDream (Inceptionism) [Mordvintsev et al., 2015]

- Start with an image
- Find high-magnitude activations
- Make them even more active
  - gradient ascent of activation magnitude wrt the image
- Repeat



"Admiral Dog!" "The Pig-Snail" "The Camel-Bird" "The Dog-Fish"

Summary

Visualising ConvNets

- First-layer filters looking Gabor-like is a good sanity check
- Backprop wrt the input is a powerful visualisation tool
  - class saliency
  - canonical images
  - works not only for image ConvNets, but for other neural nets too

QUESTIONS?