

Topics:

- CNNs Continued
- Transfer Learning

Regularization

CS 4644-DL / 7643-A

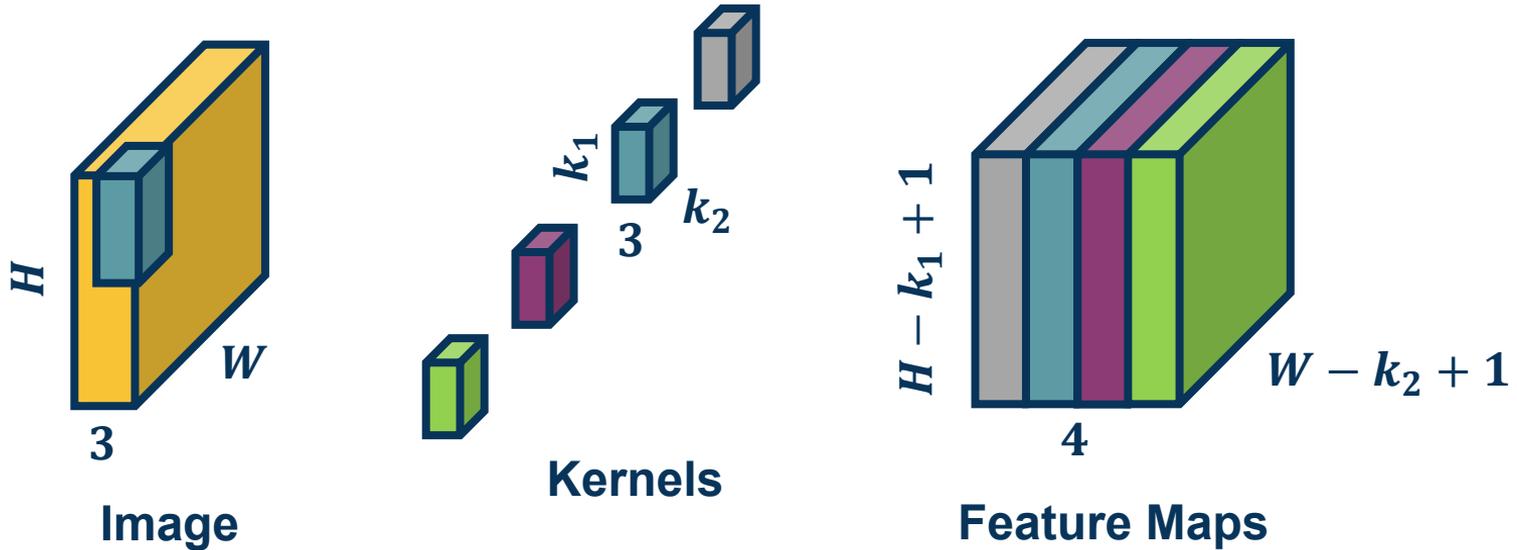
ZSOLT KIRA

- **Assignment 2 – Due Feb. 22nd**
 - Implement convolutional neural networks
 - Resources (in addition to lectures):
 - [DL book: Convolutional Networks](#)
 - CNN notes https://www.cc.gatech.edu/classes/AY2022/cs7643_spring/assets/L10_cnns_notes.pdf
 - Backprop notes https://www.cc.gatech.edu/classes/AY2023/cs7643_spring/assets/L10_cnns_backprop_notes.pdf
- **GPU resources: PACE-ICE announced**
 - Google Cloud coming soon

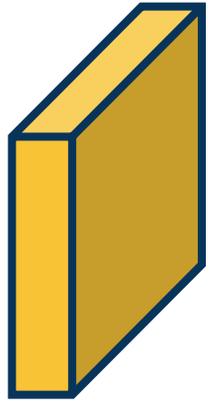
Number of parameters with N filters is: $N * (k_1 * k_2 * 3 + 1)$

Example:

$k_1 = 3, k_2 = 3, N = 4$ input channels = 3, then $(3 * 3 * 3 + 1) * 4 = 112$



Number of Parameters



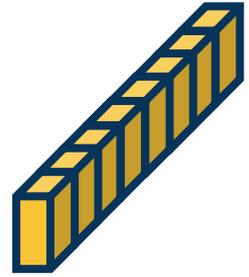
Image



Convolution +
Non-Linear
Layer



Pooling
Layer



Convolution +
Non-Linear
Layer



Fully
Connected
Layers

Loss

Adding a Fully Connected Layer

Full (simplified) AlexNet architecture:

[224x224x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

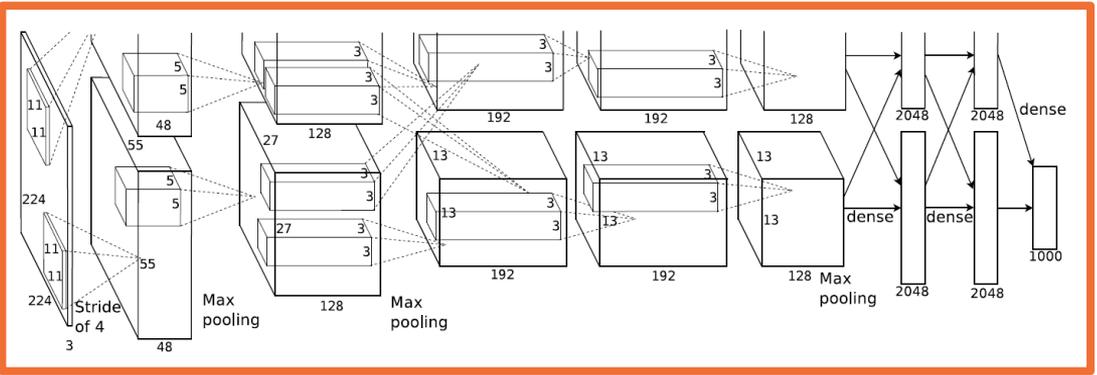
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)



Key aspects:

- ReLU instead of sigmoid or tanh
- Specialized normalization layers
- PCA-based data augmentation
- Dropout
- Ensembling

From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Small filters, Deeper networks

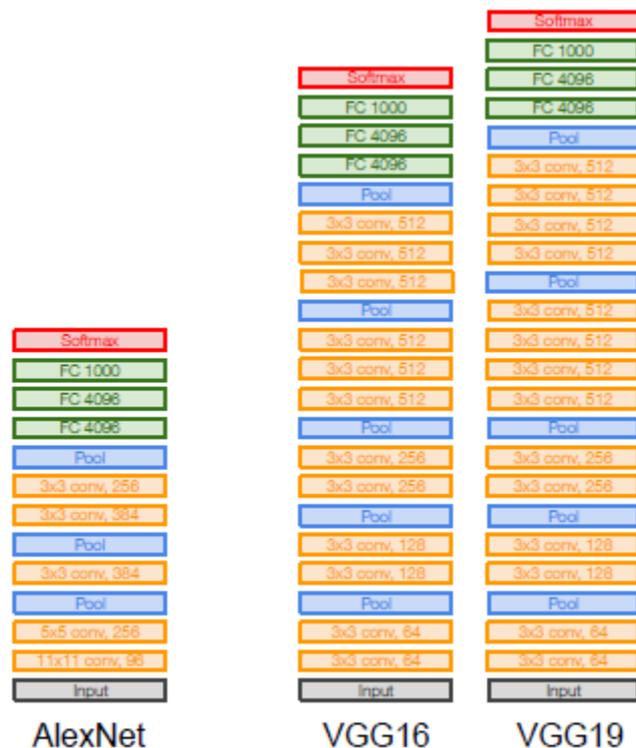
8 layers (AlexNet)

-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13 (ZFNet)

-> 7.3% top 5 error in ILSVRC'14

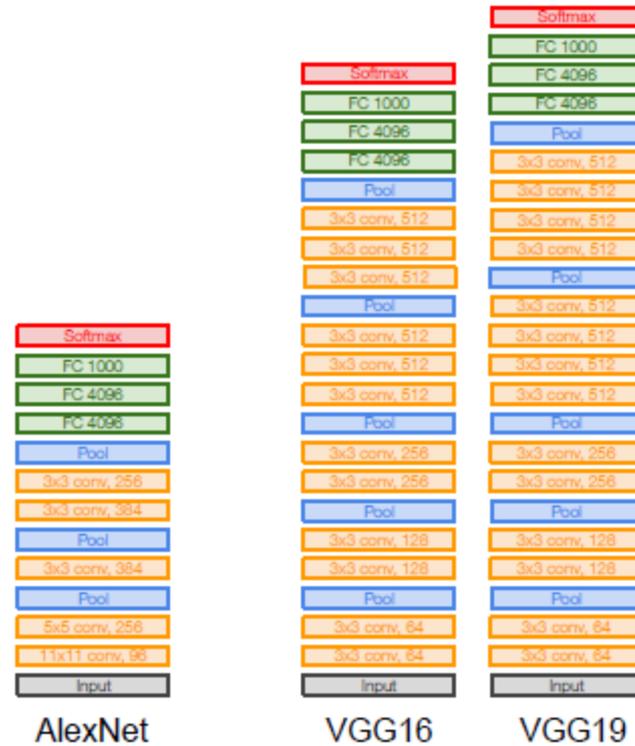


From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

INPUT: [224x224x3] memory: $224*224*3=150K$ params: 0 (not counting biases)

CONV3-64: [224x224x64] memory: $224*224*64=3.2M$ params: $(3*3*3)*64 = 1,728$

CONV3-64: [224x224x64] memory: $224*224*64=3.2M$ params: $(3*3*64)*64 = 36,864$

POOL2: [112x112x64] memory: $112*112*64=800K$ params: 0

CONV3-128: [112x112x128] memory: $112*112*128=1.6M$ params: $(3*3*64)*128 = 73,728$

CONV3-128: [112x112x128] memory: $112*112*128=1.6M$ params: $(3*3*128)*128 = 147,456$

POOL2: [56x56x128] memory: $56*56*128=400K$ params: 0

CONV3-256: [56x56x256] memory: $56*56*256=800K$ params: $(3*3*128)*256 = 294,912$

CONV3-256: [56x56x256] memory: $56*56*256=800K$ params: $(3*3*256)*256 = 589,824$

CONV3-256: [56x56x256] memory: $56*56*256=800K$ params: $(3*3*256)*256 = 589,824$

POOL2: [28x28x256] memory: $28*28*256=200K$ params: 0

CONV3-512: [28x28x512] memory: $28*28*512=400K$ params: $(3*3*256)*512 = 1,179,648$

CONV3-512: [28x28x512] memory: $28*28*512=400K$ params: $(3*3*512)*512 = 2,359,296$

CONV3-512: [28x28x512] memory: $28*28*512=400K$ params: $(3*3*512)*512 = 2,359,296$

POOL2: [14x14x512] memory: $14*14*512=100K$ params: 0

CONV3-512: [14x14x512] memory: $14*14*512=100K$ params: $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14*14*512=100K$ params: $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14*14*512=100K$ params: $(3*3*512)*512 = 2,359,296$

POOL2: [7x7x512] memory: $7*7*512=25K$ params: 0

FC: [1x1x4096] memory: 4096 params: $7*7*512*4096 = 102,760,448$

FC: [1x1x4096] memory: 4096 params: $4096*4096 = 16,777,216$

FC: [1x1x1000] memory: 1000 params: $4096*1000 = 4,096,000$

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: Number of parameters (in millions).

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

From: Simonyan & Zimmerman, Very Deep Convolutional Networks for Large-Scale Image Recognition
 From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

INPUT: [224x224x3] memory: $224*224*3=150K$ params: 0 (not counting biases)

CONV3-64: [224x224x64] memory: $224*224*64=3.2M$ params: $(3*3*3)*64 = 1,728$

CONV3-64: [224x224x64] memory: $224*224*64=3.2M$ params: $(3*3*64)*64 = 36,864$

POOL2: [112x112x64] memory: $112*112*64=800K$ params: 0

CONV3-128: [112x112x128] memory: $112*112*128=1.6M$ params: $(3*3*64)*128 = 73,728$

CONV3-128: [112x112x128] memory: $112*112*128=1.6M$ params: $(3*3*128)*128 = 147,456$

POOL2: [56x56x128] memory: $56*56*128=400K$ params: 0

CONV3-256: [56x56x256] memory: $56*56*256=800K$ params: $(3*3*128)*256 = 294,912$

CONV3-256: [56x56x256] memory: $56*56*256=800K$ params: $(3*3*256)*256 = 589,824$

CONV3-256: [56x56x256] memory: $56*56*256=800K$ params: $(3*3*256)*256 = 589,824$

POOL2: [28x28x256] memory: $28*28*256=200K$ params: 0

CONV3-512: [28x28x512] memory: $28*28*512=400K$ params: $(3*3*256)*512 = 1,179,648$

CONV3-512: [28x28x512] memory: $28*28*512=400K$ params: $(3*3*512)*512 = 2,359,296$

CONV3-512: [28x28x512] memory: $28*28*512=400K$ params: $(3*3*512)*512 = 2,359,296$

POOL2: [14x14x512] memory: $14*14*512=100K$ params: 0

CONV3-512: [14x14x512] memory: $14*14*512=100K$ params: $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14*14*512=100K$ params: $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14*14*512=100K$ params: $(3*3*512)*512 = 2,359,296$

POOL2: [7x7x512] memory: $7*7*512=25K$ params: 0

FC: [1x1x4096] memory: 4096 params: $7*7*512*4096 = 102,760,448$

FC: [1x1x4096] memory: 4096 params: $4096*4096 = 16,777,216$

FC: [1x1x1000] memory: 1000 params: $4096*1000 = 4,096,000$

Most memory usage in convolution layers

Most parameters in FC layers

From: Simonyan & Zimmerman, Very Deep Convolutional Networks for Large-Scale Image Recognition
 From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

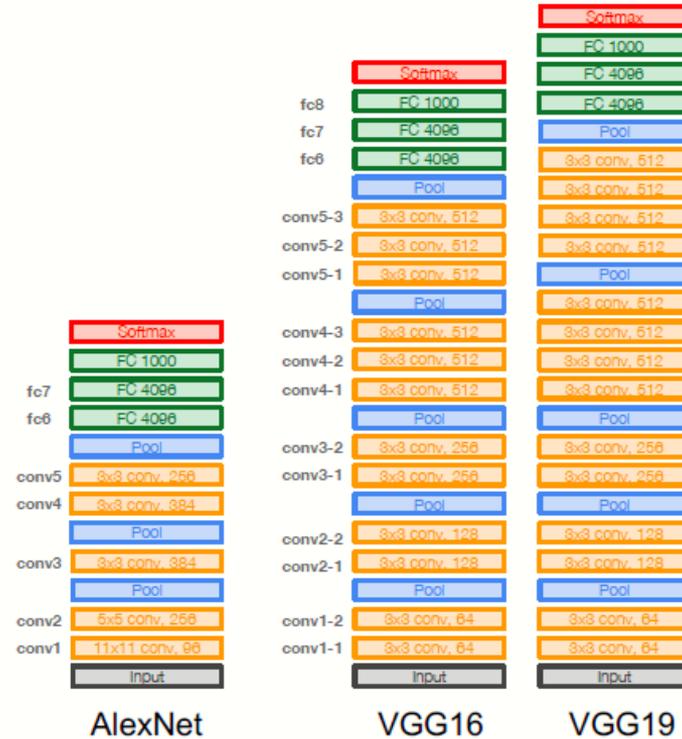
Details:

- ILSVRC'14 2nd in classification, 1st in localization
- Similar training procedure as Krizhevsky 2012
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks

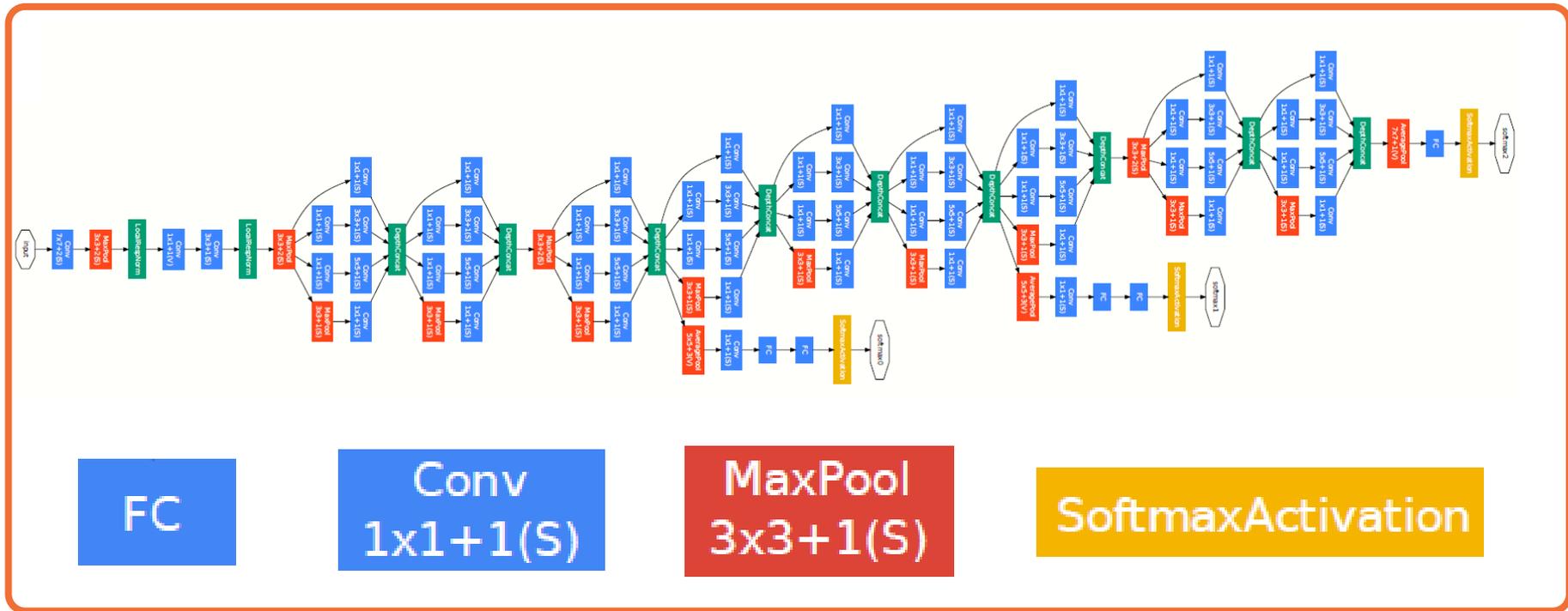
Still very expensive!

TOTAL memory: $24M * 4 \text{ bytes} \approx 96MB / \text{image}$
(only forward! $\sim *2$ for bwd)

TOTAL params: 138M parameters



But have become **deeper and more complex**



From: Szegedy et al. *Going deeper with convolutions*

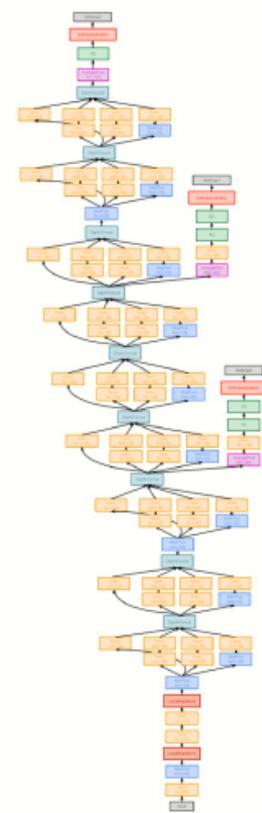
Inception Architecture

Case Study: GoogLeNet

[Szegedy et al., 2014]

Deeper networks, focus on
computational efficiency

- ILSVRC'14 classification winner
(6.7% top 5 error)
- 22 layers
- Only 5 million parameters!
12x less than AlexNet
27x less than VGG-16
- Efficient "Inception" module
- No FC layers



From: Szegedy et al. Going deeper with convolutions

Inception Architecture

Case Study: GoogLeNet

[Szegedy et al., 2014]

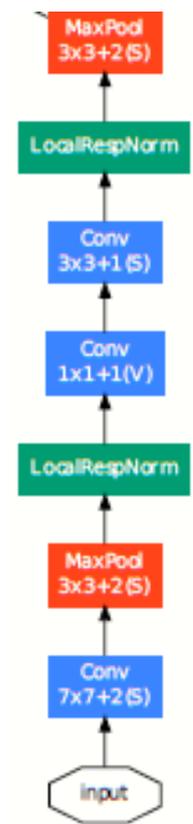
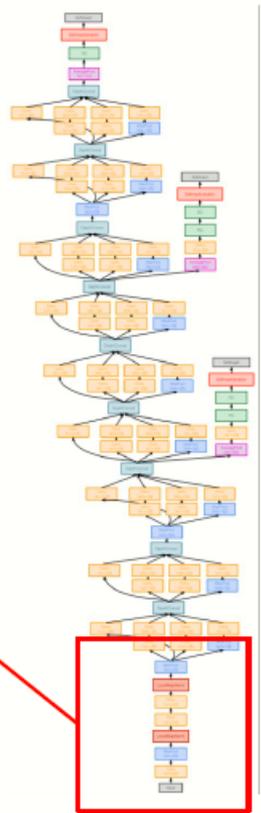
Deeper networks, focus on computational efficiency

- ILSVRC'14 classification winner (6.7% top 5 error)
- 22 layers
- Only 5 million parameters!
12x less than AlexNet
27x less than VGG-16
- Efficient "Inception" module
- No FC layers

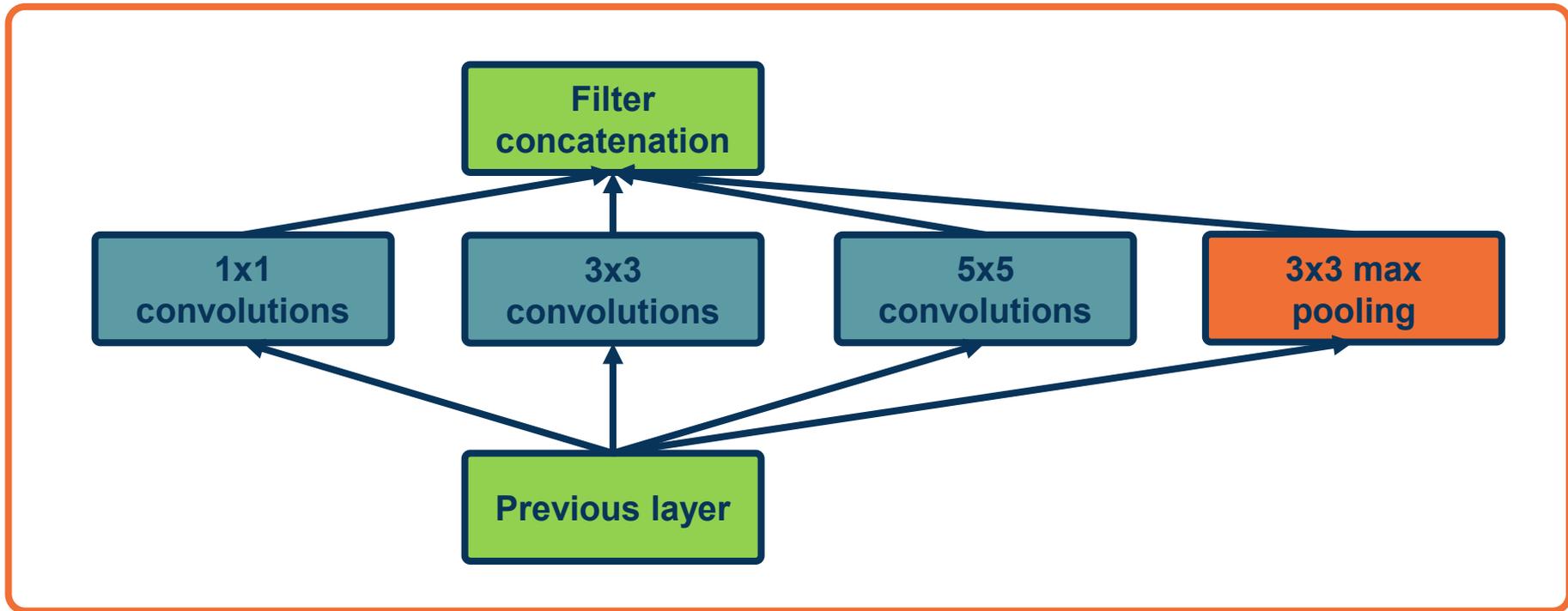
Stem Network: aggressively reduce the input feature volume

- Conv 7 x 7 x 64 with stride 2
- MaxPool
- Conv 1 x 1 x 64
- Conv 3 x 3 x 192
- MaxPool

Reduce 224 x 224 spatial resolution to 28 x 28 with just 418 MFLOP! (Comparing to 7485 MFLOP of VGG)



Key idea: Repeated blocks and multi-scale features



From: Szegedy et al. Going deeper with convolutions

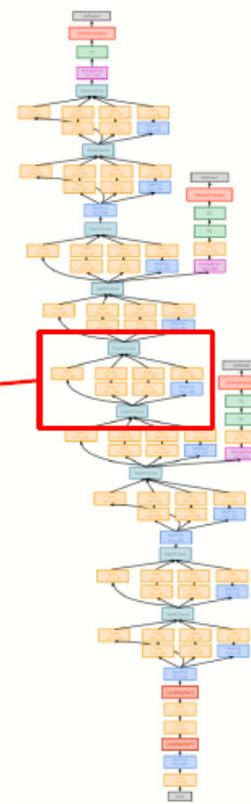
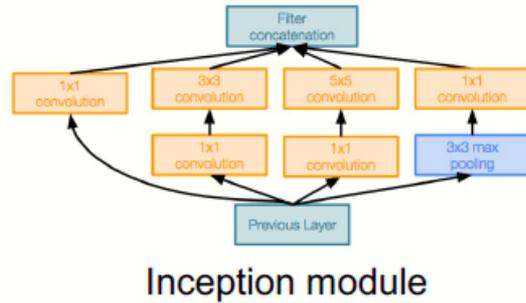
Inception Module

Case Study: GoogLeNet

[Szegedy et al., 2014]

“Inception module”: design a good local network topology (network within a network) and then stack these modules on top of each other

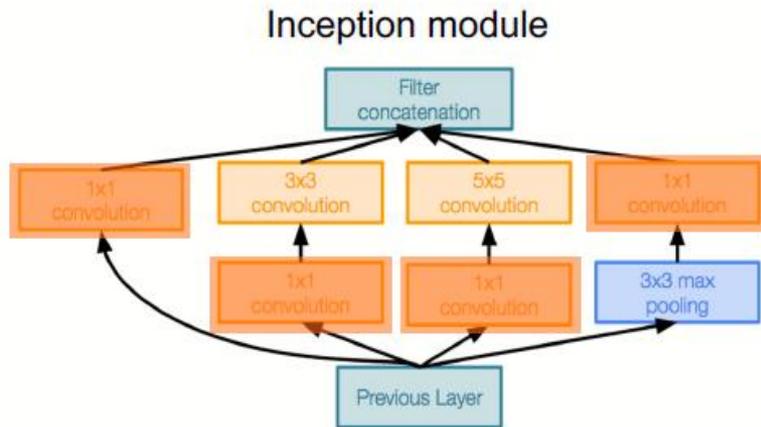
Multiple conv filter size diversifies learned features



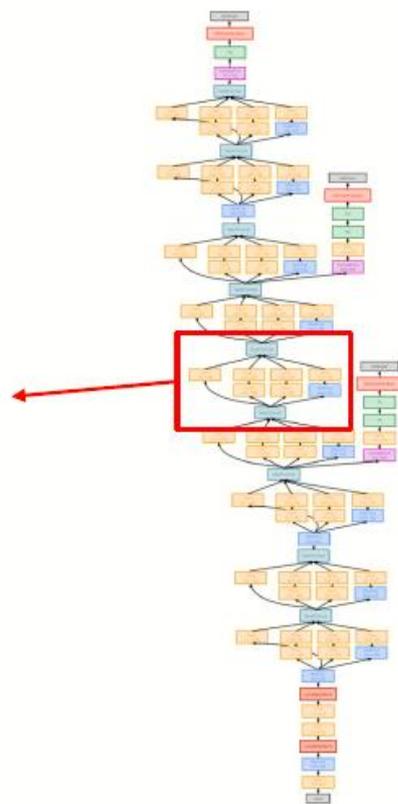
From: slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Case Study: GoogLeNet

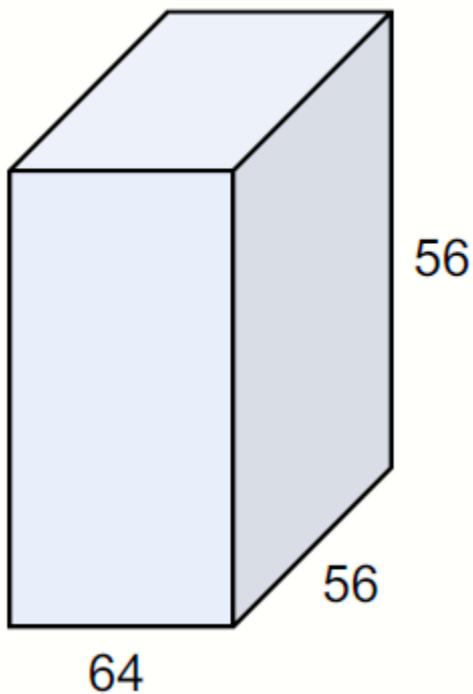
[Szegedy et al., 2014]



Uses 1x1 “Bottleneck” layers to reduce channel dimension before expensive conv (we will revisit this with ResNet!)



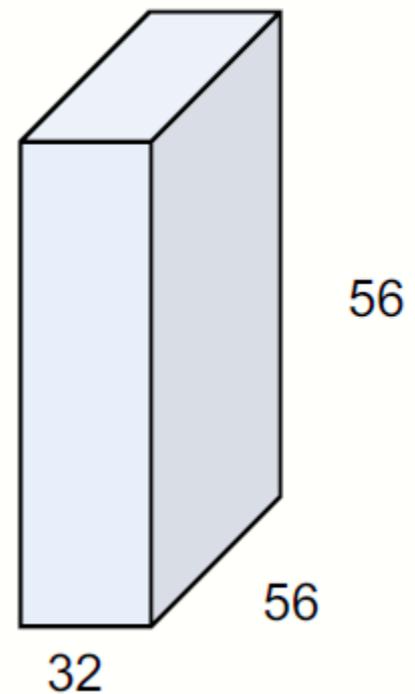
From: slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n



1x1 CONV
with 32 filters

→

(each filter has size
1x1x64, and performs a
64-dimensional dot
product)

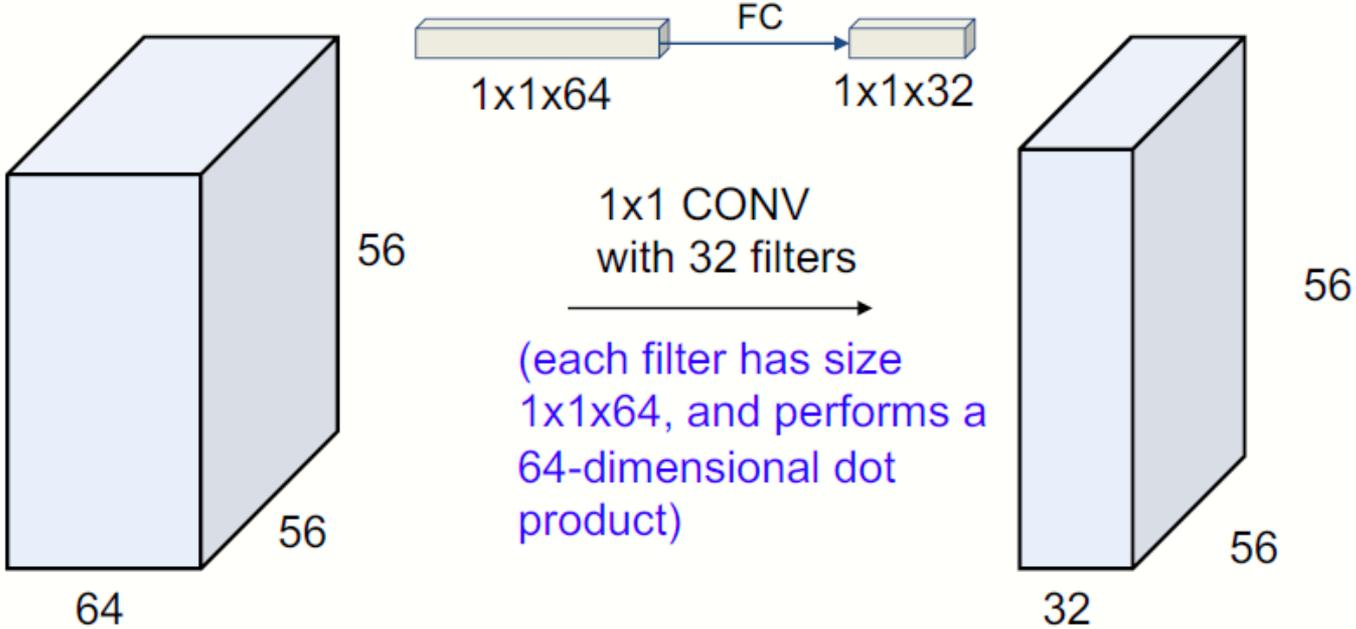


From: slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

1x1 Convolutions



Alternatively, interpret it as applying the same FC layer on each input pixel



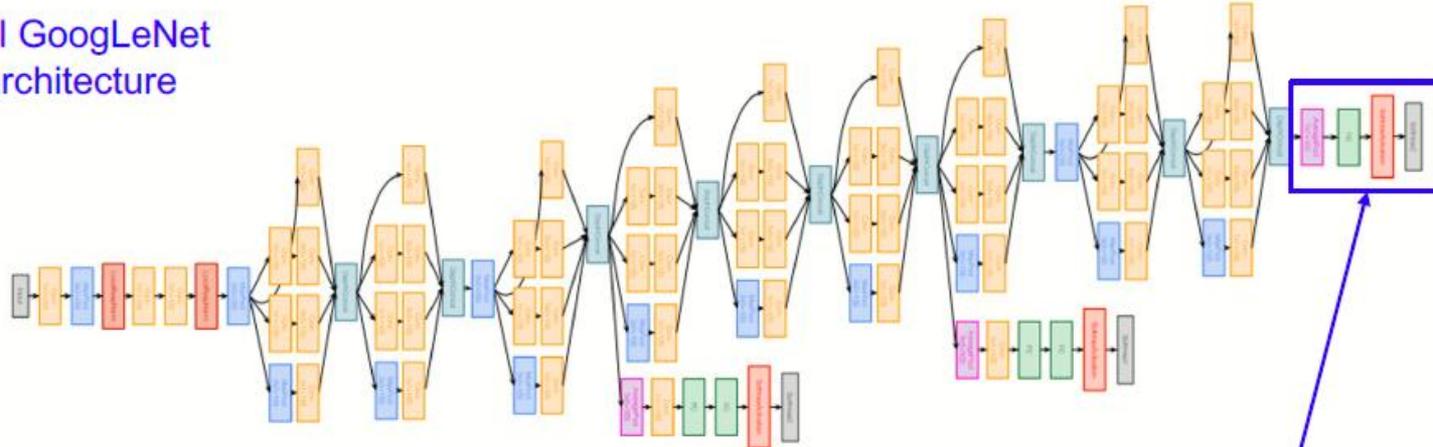
From: slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

1x1 Convolutions

Case Study: GoogLeNet

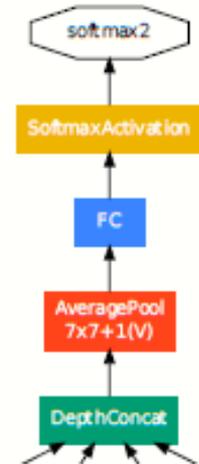
[Szegedy et al., 2014]

Full GoogLeNet
architecture



Note: after the last convolutional layer, a global average pooling layer is used that **spatially** averages across each feature map, before final FC layer. No longer multiple expensive FC layers!
(Also used in ResNet)

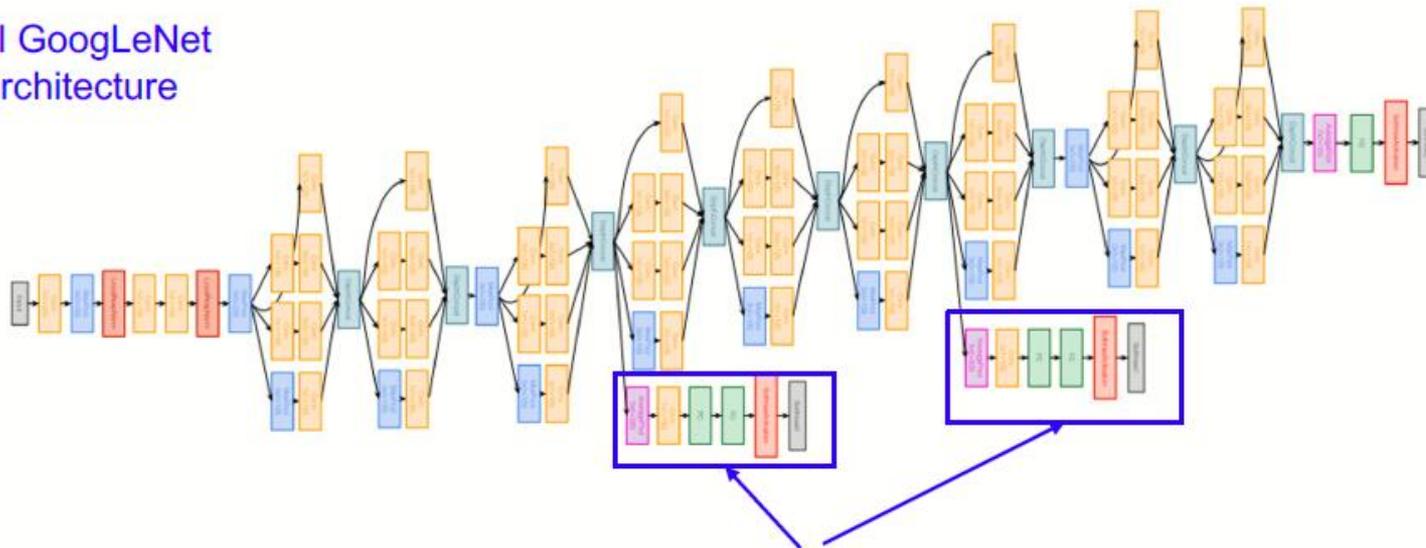
Classifier output



Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet
architecture



Auxiliary classification outputs to inject additional gradient at lower layers (AvgPool-1x1Conv-FC-FC-Softmax)

Why?

From: slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

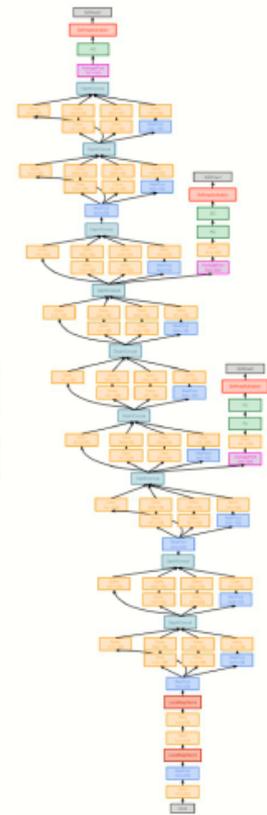
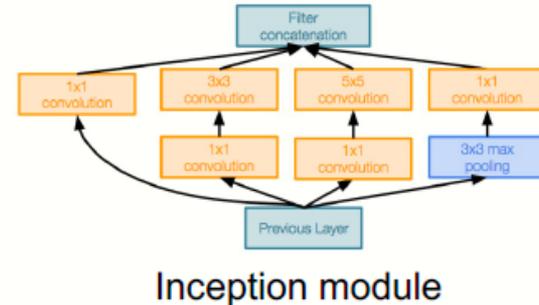
1x1 Convolutions

Case Study: GoogLeNet

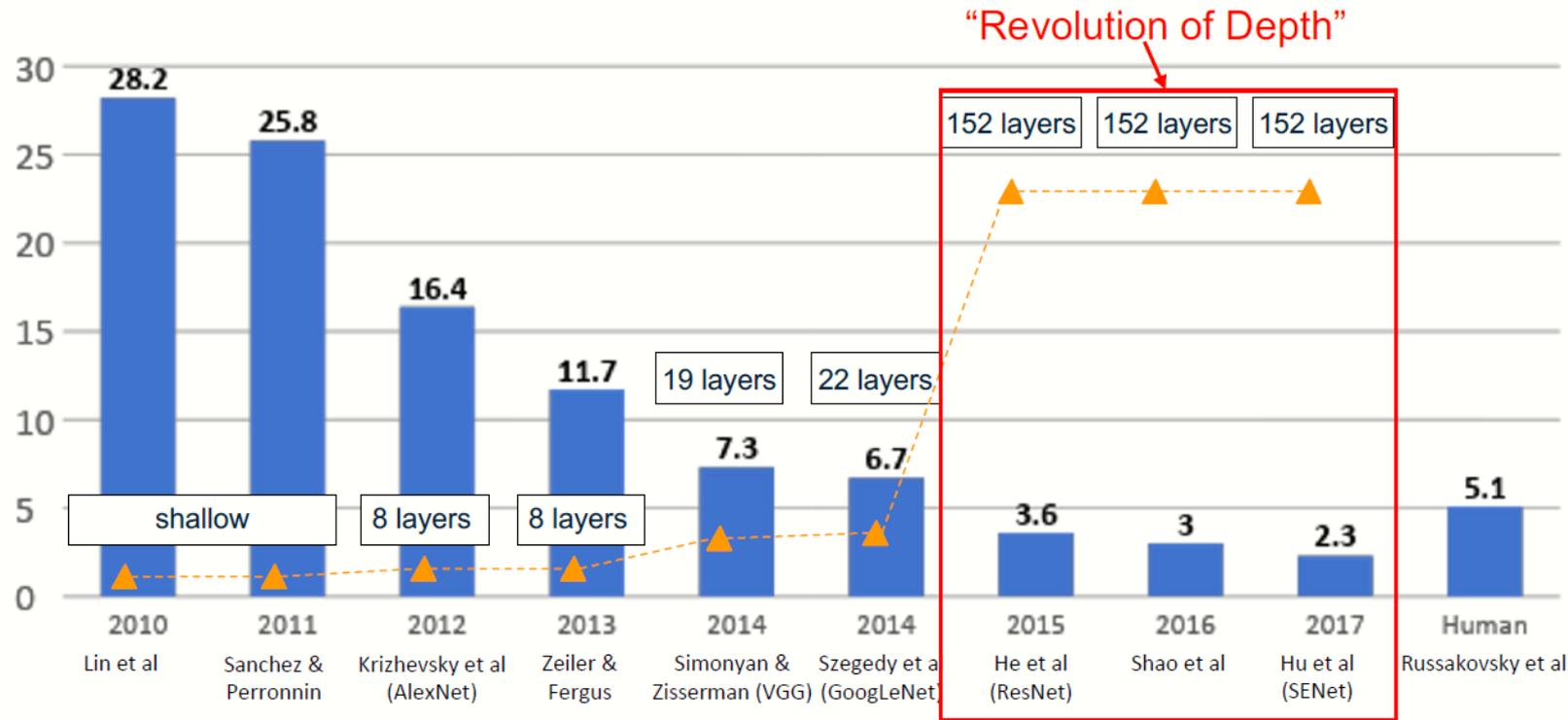
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- Avoids expensive FC layers
- 12x less params than AlexNet
- 27x less params than VGG-16
- ILSVRC’14 classification winner (6.7% top 5 error)

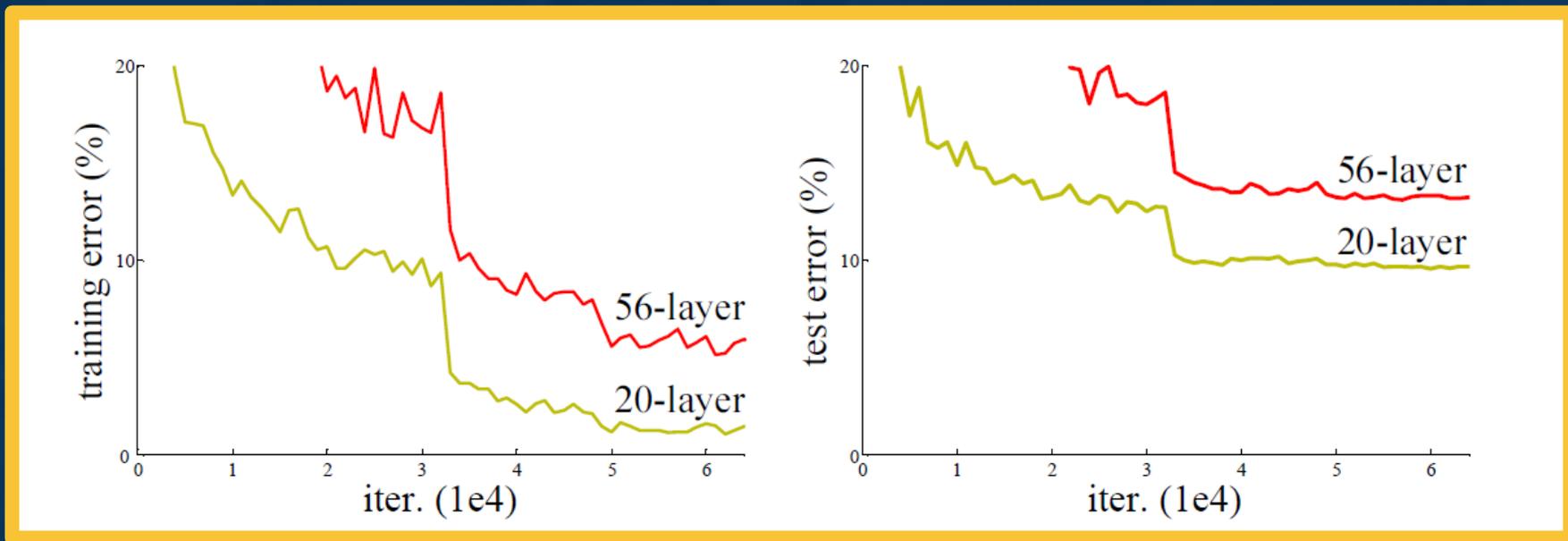


ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Revolution of Depth

The Challenge of Depth



From: He et al., Deep Residual Learning for Image Recognition

Optimizing very deep networks is challenging!

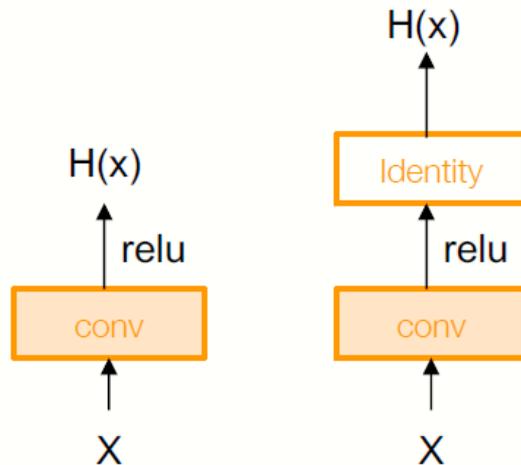
[He et al., 2015]

A deeper model can **emulate** a shallower model: copy layers from shallower model, set extra layers to identity

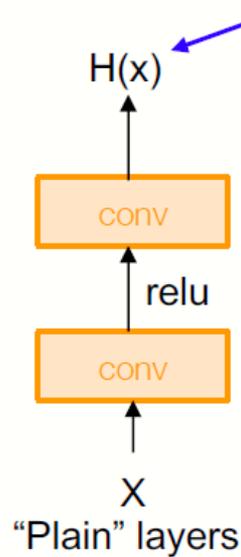
Thus deeper models should do at least as good as shallow models

Deeper models are harder to optimize. They don't learn identity functions (no-op) to emulate shallow models

Solution: Change the network so learning identity functions (no-op) as extra layers is easy

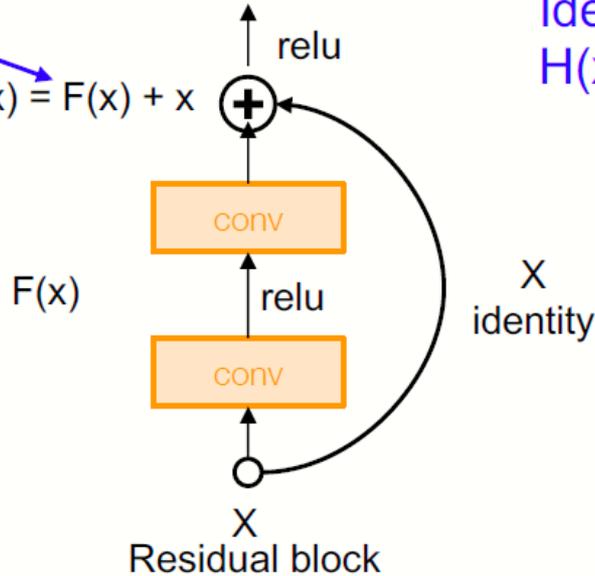


Solution: Change the network so learning identity functions as extra layers is easy



$$H(x) = F(x) + x$$

$$H(x) = F(x) + x$$



Identity mapping:
 $H(x) = x$ if $F(x) = 0$

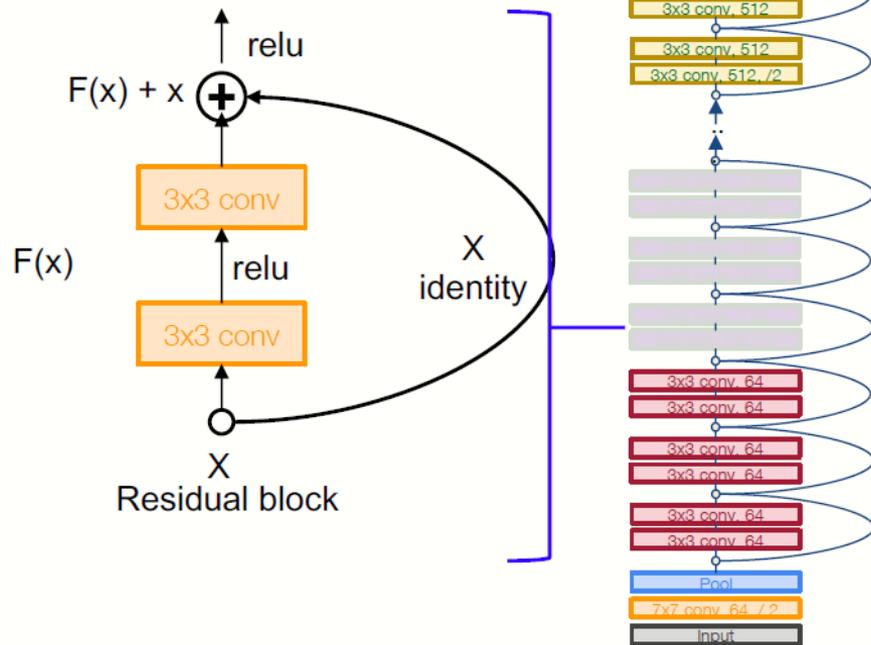
Use layers to fit **residual**
 $F(x) = H(x) - x$
instead of
 $H(x)$ directly

Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers

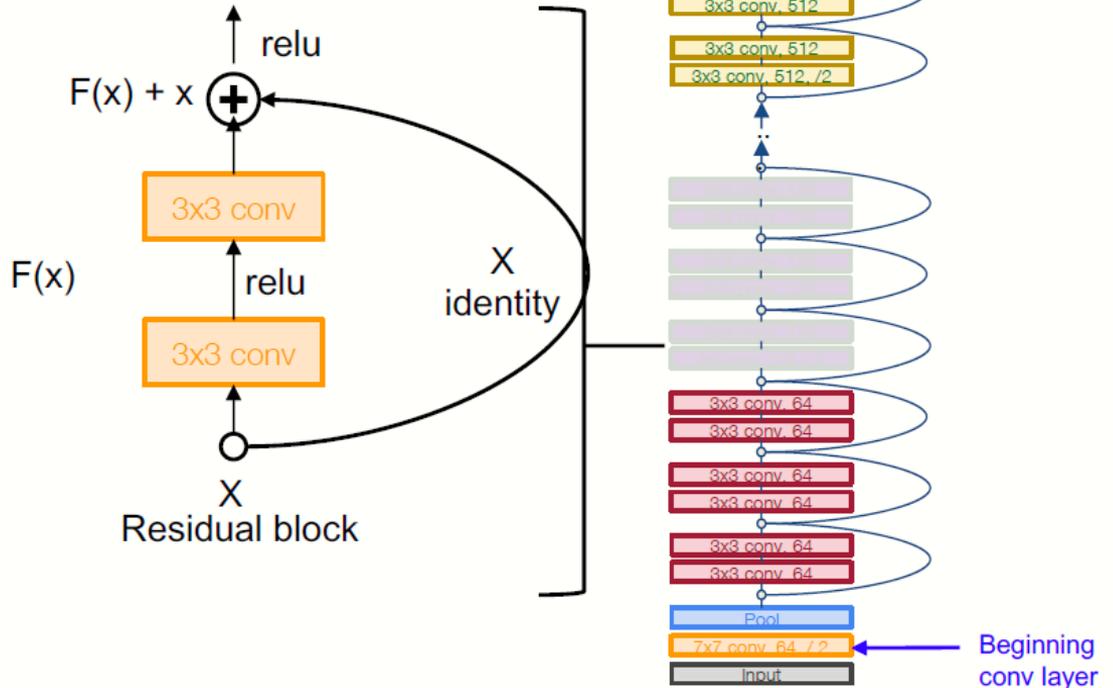


Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

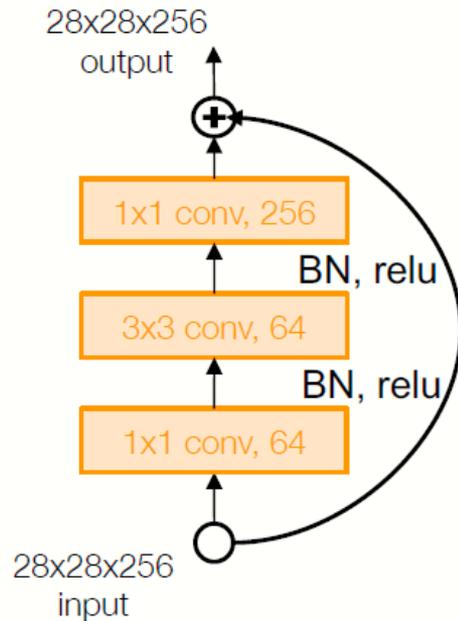
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Reduce the activation volume by half.
- Additional conv layer at the beginning (stem)



Skip Connections

[He et al., 2015]

For deeper networks (ResNet-50+), use “bottleneck” layer to improve efficiency (similar to GoogLeNet)



Bottleneck Layers

```

>>> import torch
>>> from torchvision.models import resnet18
>>> model = resnet18()
>>> summary(model, (3, 224, 224), device='cpu')

```

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 112, 112]	9,408
BatchNorm2d-2	[-1, 64, 112, 112]	128
ReLU-3	[-1, 64, 112, 112]	0
MaxPool2d-4	[-1, 64, 56, 56]	0
Conv2d-5	[-1, 64, 56, 56]	36,864
BatchNorm2d-6	[-1, 64, 56, 56]	128
ReLU-7	[-1, 64, 56, 56]	0
Conv2d-8	[-1, 64, 56, 56]	36,864
BatchNorm2d-9	[-1, 64, 56, 56]	128
ReLU-10	[-1, 64, 56, 56]	0
BasicBlock-11	[-1, 64, 56, 56]	0
Conv2d-12	[-1, 64, 56, 56]	36,864
BatchNorm2d-13	[-1, 64, 56, 56]	128
ReLU-14	[-1, 64, 56, 56]	0
Conv2d-15	[-1, 64, 56, 56]	36,864
BatchNorm2d-16	[-1, 64, 56, 56]	128
ReLU-17	[-1, 64, 56, 56]	0
BasicBlock-18	[-1, 64, 56, 56]	0
Conv2d-19	[-1, 128, 28, 28]	73,728
BatchNorm2d-20	[-1, 128, 28, 28]	256
ReLU-21	[-1, 128, 28, 28]	0
Conv2d-22	[-1, 128, 28, 28]	147,456
BatchNorm2d-23	[-1, 128, 28, 28]	256

layer name	output size	18-layer	34-layer
conv1	112×112		
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$
	1×1		average
FLOPs		1.8×10^9	3.6×10^9

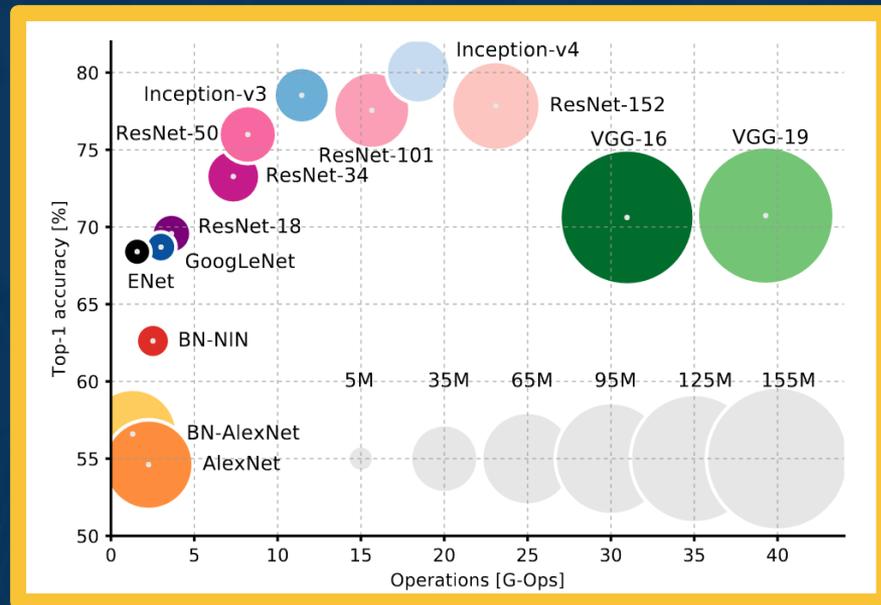
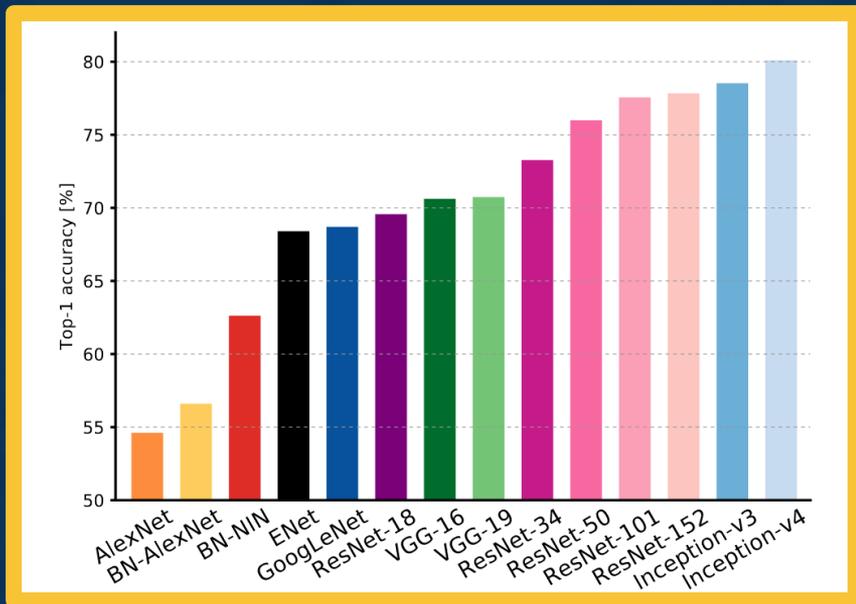


ResNet Details

Training ResNet in practice:

- Batch Normalization after every CONV layer
- Xavier initialization from He et al.
- SGD + Momentum
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of $1e-5$
- No dropout used

Computational Complexity

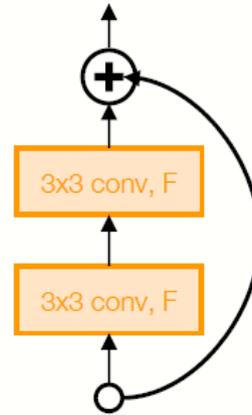


From: *An Analysis Of Deep Neural Network Models For Practical Applications*

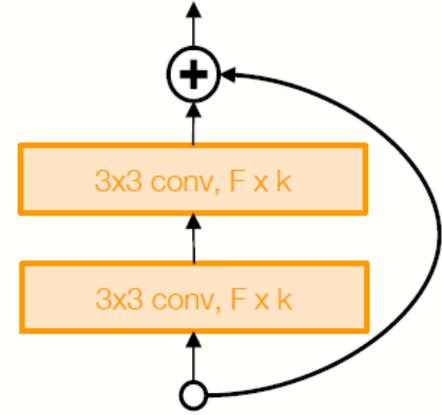
Wide Residual Networks

[Zagoruyko et al. 2016]

- Argues that residuals are the important factor, not depth
- Use wider residual blocks ($F \times k$ filters instead of F filters in each layer)
- 50-layer wide ResNet outperforms 152-layer original ResNet
- Increasing width instead of depth more computationally efficient (parallelizable)



Basic residual block

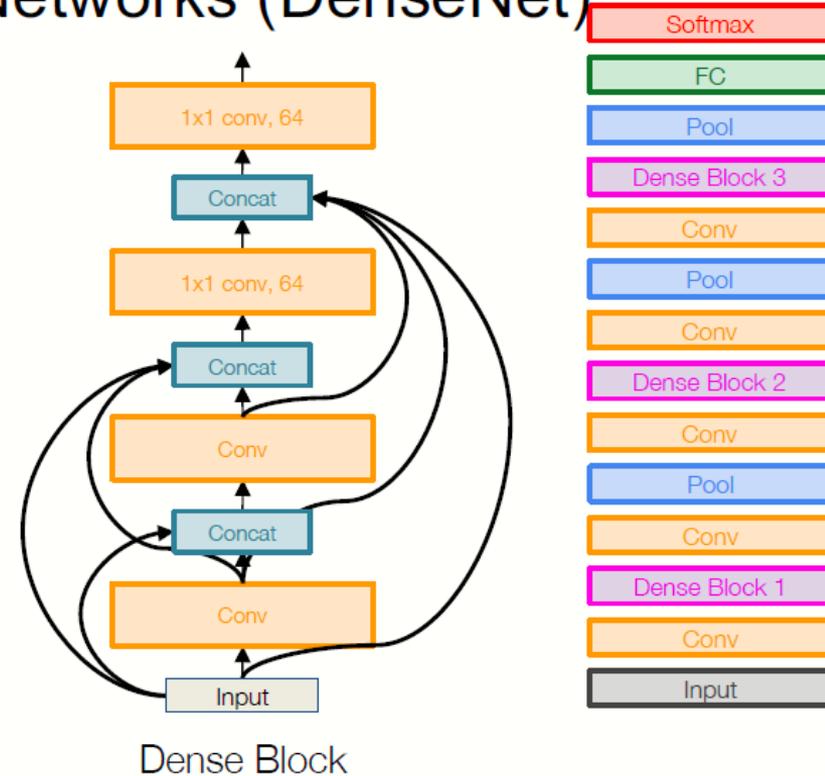


Wide residual block

Densely Connected Convolutional Networks (DenseNet)

[Huang et al. 2017]

- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse
- Showed that shallow 50-layer network can outperform deeper 152 layer ResNet

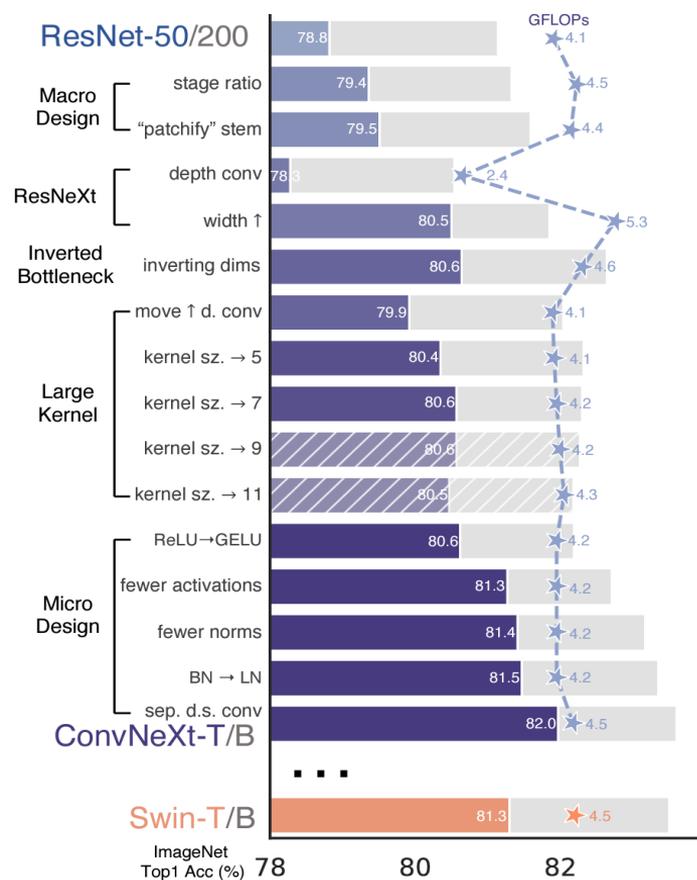


ConvNeXt (2022)

- ***To bridge the gap between the Conv Nets and Vision Transformers (ViT)***
 - ViT, Swin Transformer has been the SOTA visual model backbone
 - Is convolutional networks really not as good as transformer models?
- Investigation
 - The author start with ResNet-50 and reimplement the CNN networks with modern designs
 - The results showing that ConvNeXt achieves beat the ViT models, again.

ConvNeXt (2022)

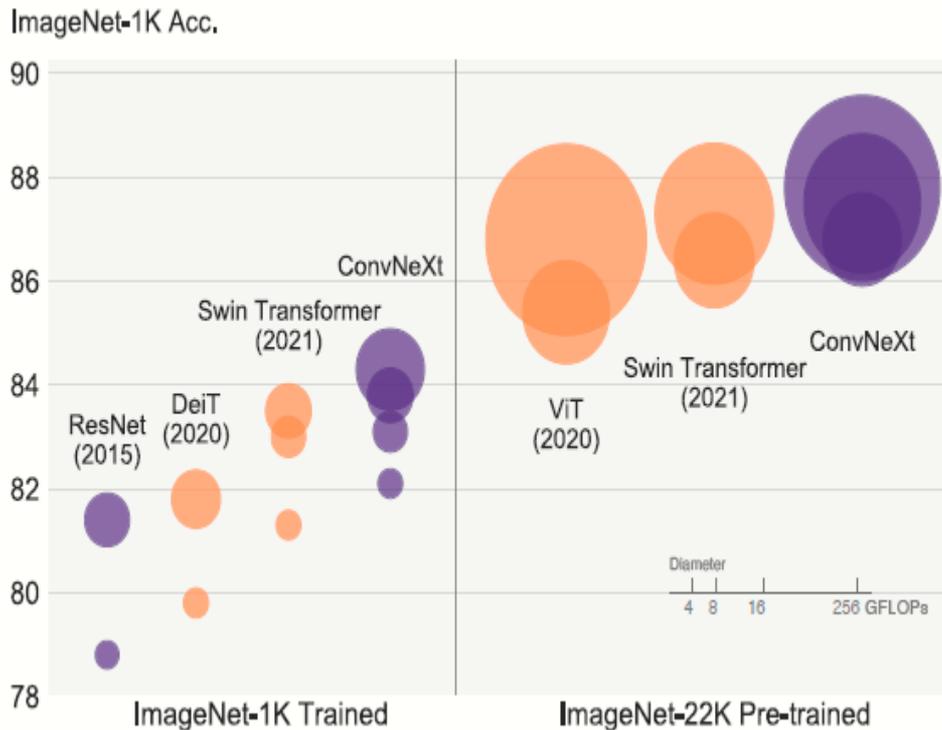
- Modern designs added:
- Use ResNeXt
- Apply Inverted Bottleneck
- Use larger kernel size
- Training strategy:
 - 90 epochs -> 300 epochs
 - AdamW optimizer
 - Data augmentation like Mixup, CutMix
 - Regularization Schemes like label smoothing
 - ...



ConvNeXt (2022)

- **Modern designs added:**
- Macro Design
 - Changing stage compute ratio
 - Changing stem to “patchify”
- Micro Design
 - ReLU -> GELU
 - Fewer activation functions
 - Fewer normalization layers
 - BatchNorm -> LayerNorm
 - Separate downsampling layers

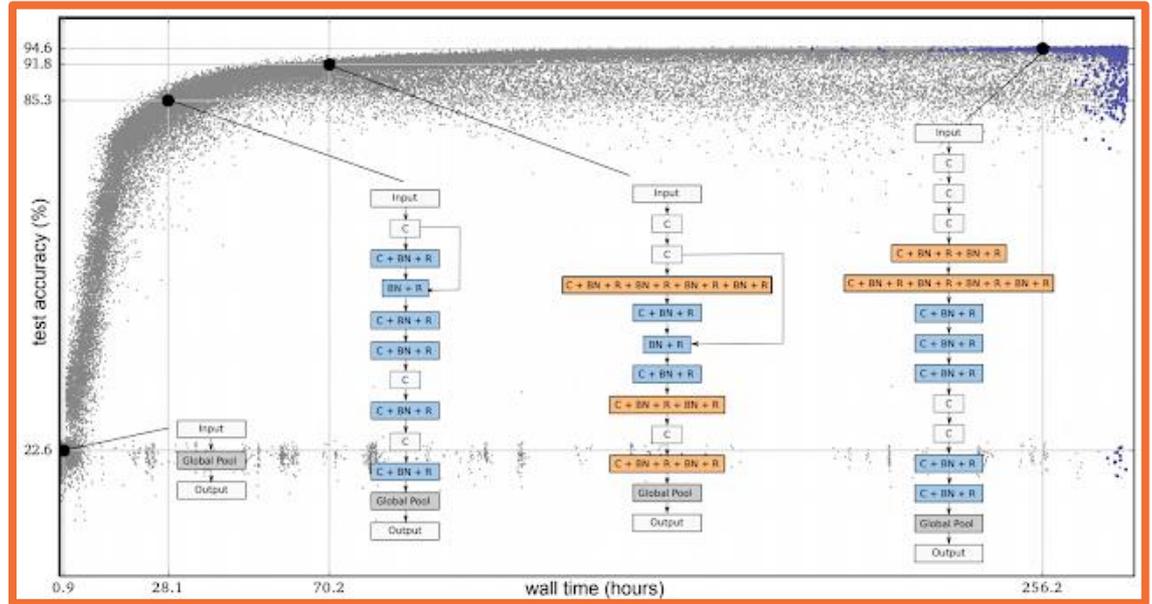
Nonlinearities



$$f(x) = 0.5x \left(1 + \tanh \left(\sqrt{\frac{2}{\pi}} (x + 0.044715x^3) \right) \right)$$

Several ways to learn architectures:

- Evolutionary learning and reinforcement learning
- Prune over-parameterized networks
- Learning of repeated blocks typical

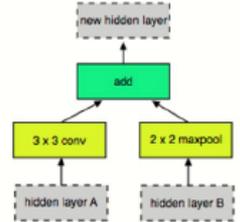
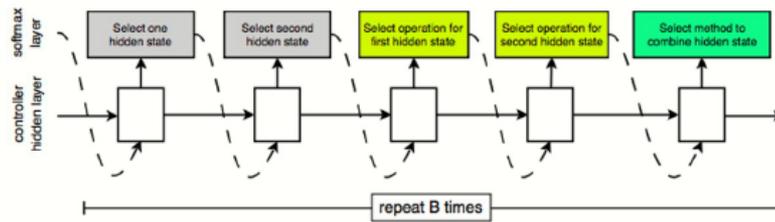
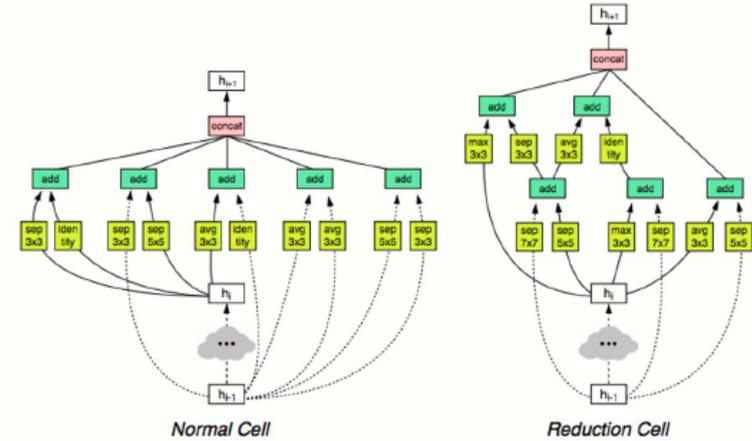


From: <https://ai.googleblog.com/2018/03/using-evolutionary-automl-to-discover.html>

Learning Transferable Architectures for Scalable Image Recognition

[Zoph et al. 2017]

- Applying neural architecture search (NAS) to a large dataset like ImageNet is expensive
- Design a search space of building blocks (“cells”) that can be flexibly stacked
- NASNet: Use NAS to find best cell structure on smaller CIFAR-10 dataset, then transfer architecture to ImageNet
- Many follow-up works in this space e.g. AmoebaNet (Real et al. 2019) and ENAS (Pham, Guan et al. 2018)



- ◆ Convolutional neural networks (CNNs) stack pooling, convolution, non-linearities, and fully connected (FC) layers

- ◆ Feature engineering => architecture engineering!
 - ◆ Tons of small details and tips/tricks
 - ◆ Considerations: Memory, compute/FLO, dimensionality reduction, diversity of features, number of parameters/capacity, etc.

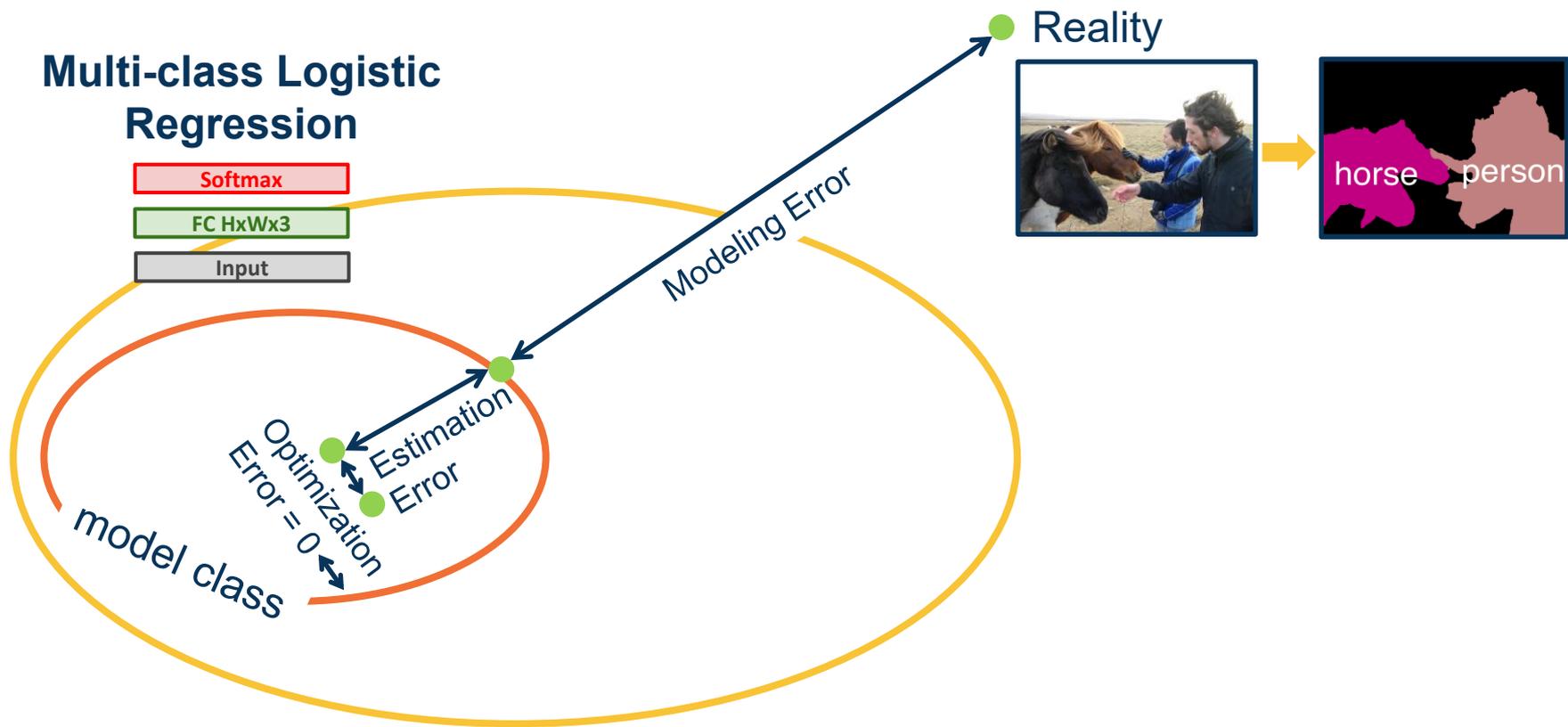
Transfer Learning & Generalization

Multi-class Logistic Regression

Softmax

FC HxWx3

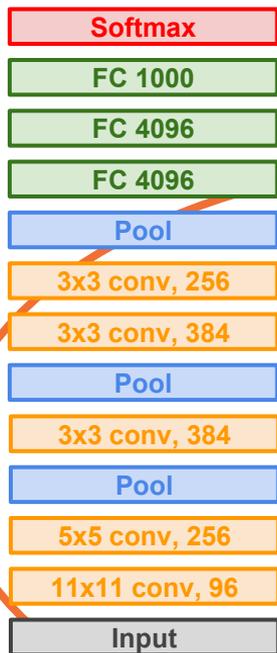
Input



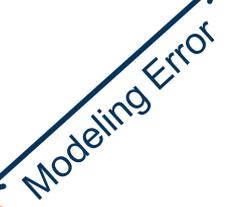
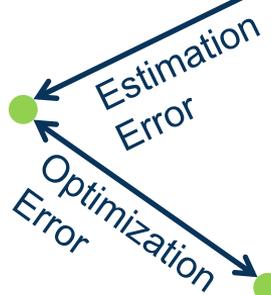
From: slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Generalization

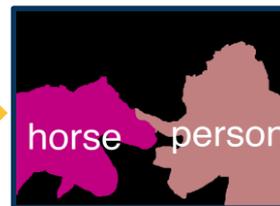
AlexNet



model class



Reality

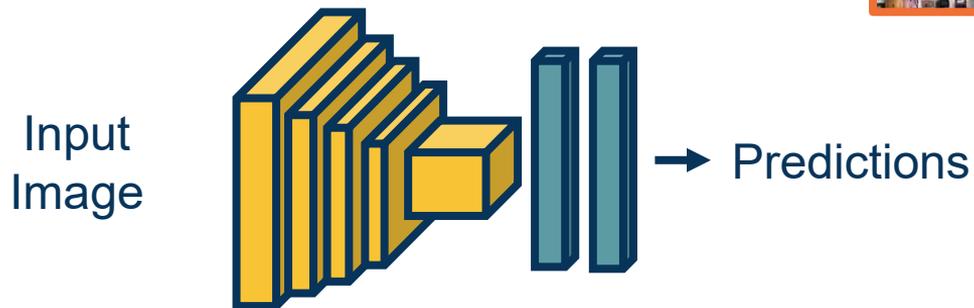


From: slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Generalization

What if we don't have enough data?

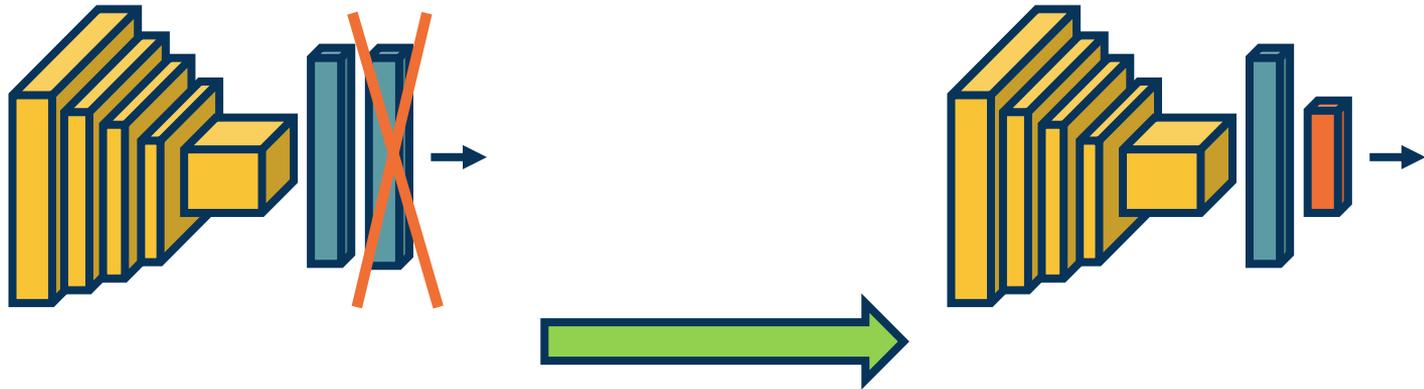
Step 1: Train on large-scale dataset



Convolutional Neural Networks

Transfer Learning – Training on Large Dataset

Step 2: Take your custom data and **initialize** the network with weights trained in Step 1



Replace last layer with new fully-connected for output nodes per new category

Initializing with Pre-Trained Network

Step 3: (Continue to) train on new dataset

◆ **Finetune:** Update all parameters

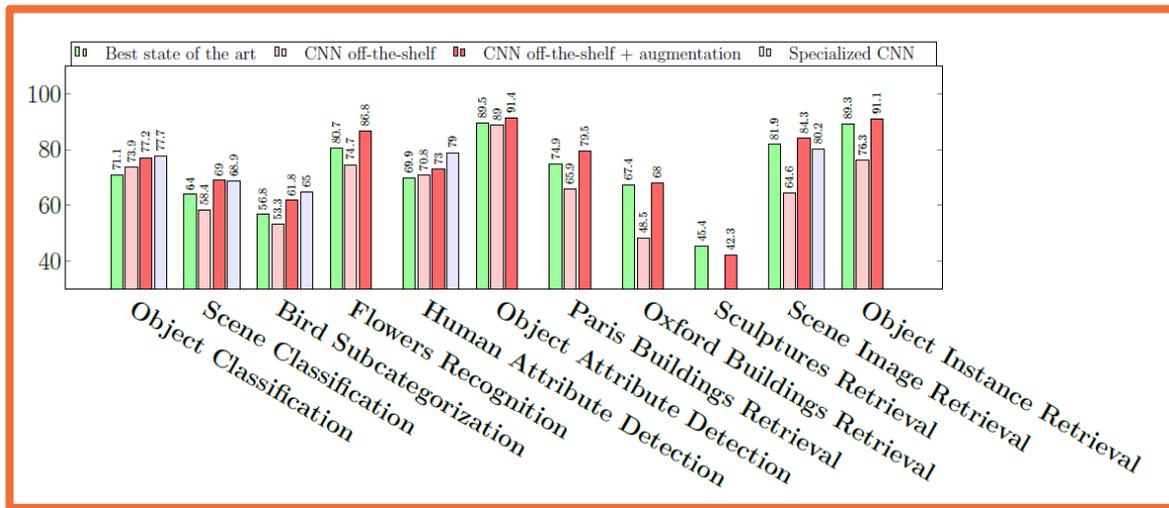
◆ **Freeze** feature layer: Update only last layer weights (used when not enough data)



Replace last layer with new fully-connected for output nodes per new category

This works extremely well! It was surprising upon discovery.

- Features learned for 1000 object categories will work well for 1001st!
- Generalizes even across tasks (classification to object detection)



From: Razavian et al., CNN Features off-the-shelf: an Astounding Baseline for Recognition

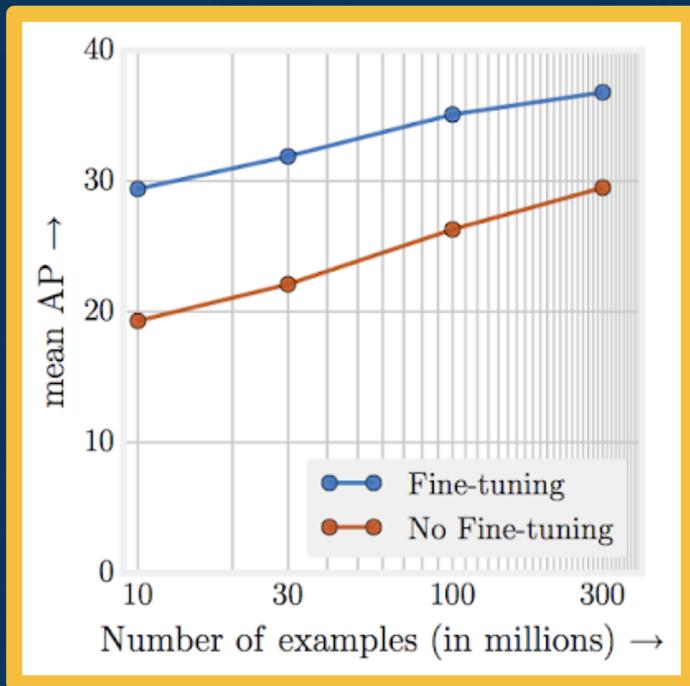
Learning with Less Labels

But it doesn't always work that well!

- ◆ If the **source** dataset you train on is very different from the **target** dataset, transfer learning is not as effective
- ◆ If you have enough data for the target domain, it just results in faster convergence
 - ◆ See He et al., “Rethinking ImageNet Pre-training”



Effectiveness of More Data



From: *Revisiting the Unreasonable Effectiveness of Data*
<https://ai.googleblog.com/2017/07/revisiting-unreasonable-effectiveness.html>

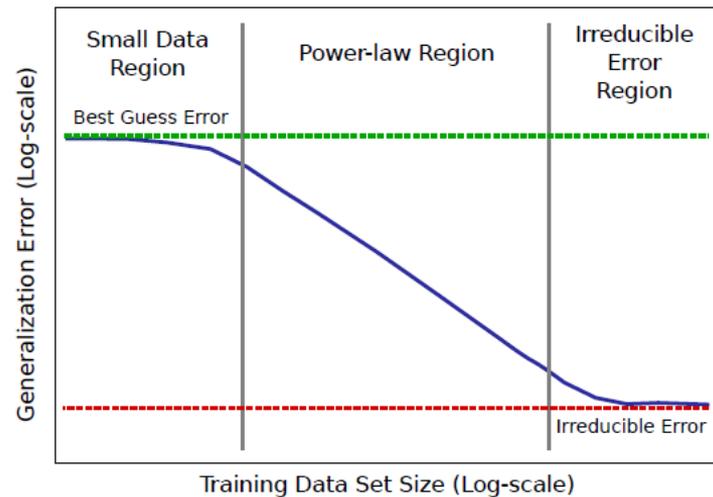


Figure 6: Sketch of power-law learning curves

From: *Hestness et al., Deep Learning Scaling Is Predictable*

There is a large number of different low-labeled settings in DL research

Setting	Source	Target	Shift Type
Semi-supervised	Single labeled	Single unlabeled	None
Domain Adaptation	Single labeled	Single unlabeled	Non-semantic
Domain Generalization	Multiple labeled	Unknown	Non-semantic
Cross-Task Transfer	Single labeled	Single unlabeled	Semantic
Few-Shot Learning	Single labeled	Single few-labeled	Semantic
Un/Self-Supervised	Single unlabeled	Many labeled	Both/Task

Non-Semantic Shift



Semantic Shift



Regularization

Many **standard regularization methods** still apply!

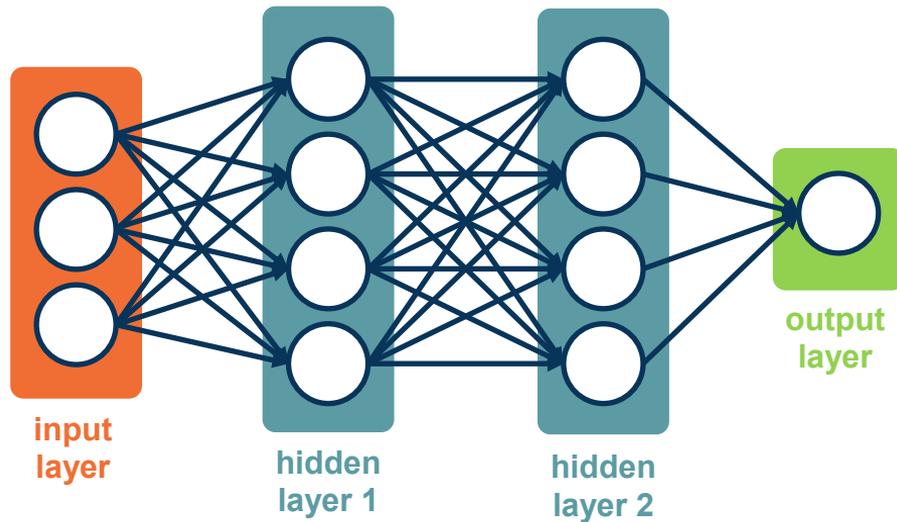
L1 Regularization

$$L = |y - Wx_i|^2 + \lambda|W|$$

where $|W|$ is element-wise

Example regularizations:

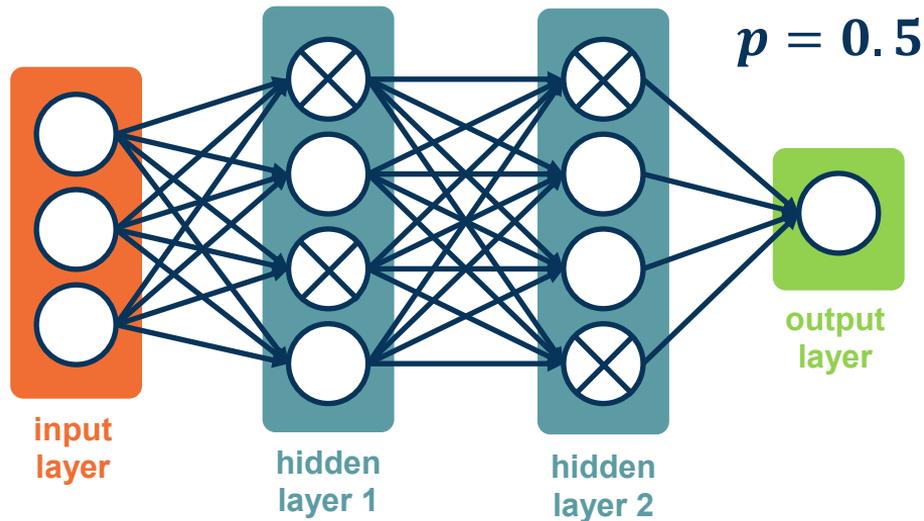
- ◆ L1/L2 on weights (encourage small values)
- ◆ L2: $L = |y - Wx_i|^2 + \lambda|W|^2$ (weight decay)
- ◆ Elastic L1/L2: $|y - Wx_i|^2 + \alpha|W|^2 + \beta|W|$



Problem: Network can learn to rely strong on a few features that work really well

- ◆ May cause **overfitting** if not representative of test data

From: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al.



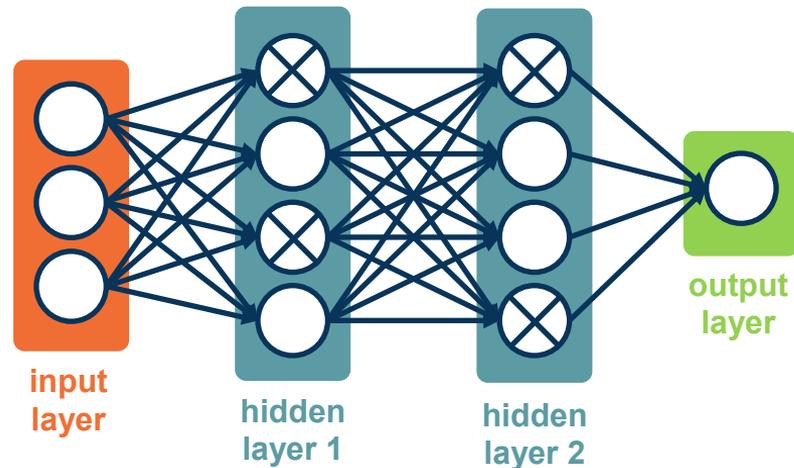
An idea: For each node, keep its output with probability p

- ◆ Activations of deactivated nodes are essentially zero

Choose whether to mask out a particular node **each iteration**

From: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al.

- In practice, implement with a **mask** calculated each iteration
- During testing, no nodes are dropped



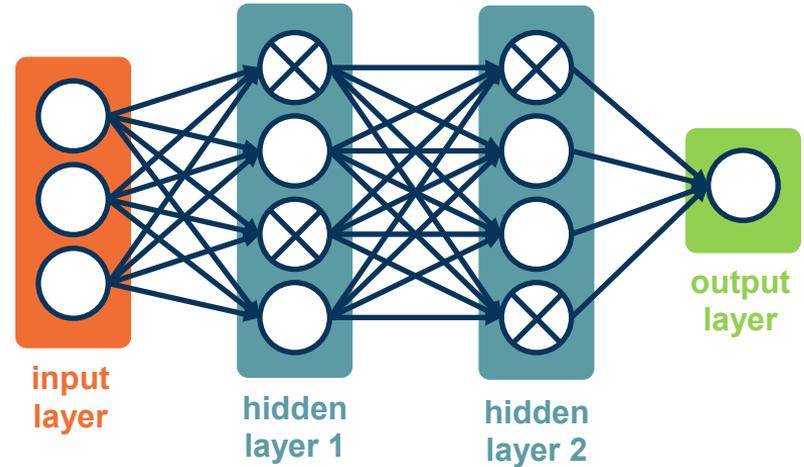
$$\begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{41} \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

From: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al.

- During training, each node has an expected $p * fan_in$ nodes
- During test all nodes are activated
- Principle:** Always try to have similar train and test-time input/output distributions!

Solution: During test time, scale outputs (or equivalently weights) by p

- i.e. $W_{test} = pW$
- Alternative: Scale by $\frac{1}{p}$ at train time

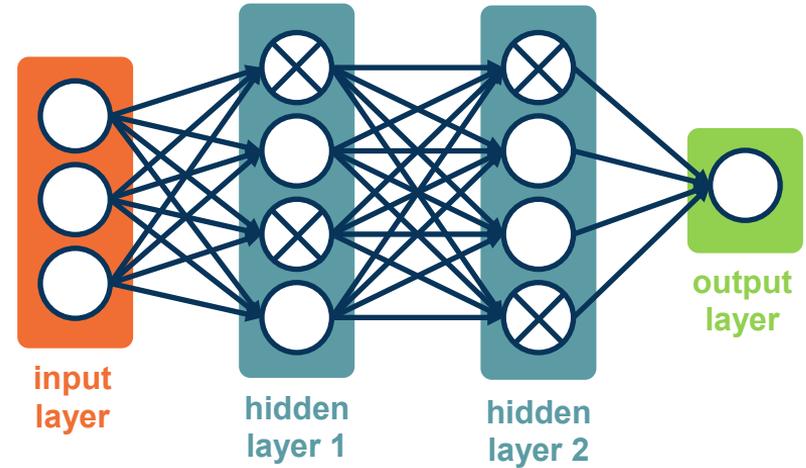


$$\begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{41} \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

From: *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, Srivastava et al.

Interpretation 1: The model should not rely too heavily on particular features

- If it does, it has probability $1 - p$ of losing that feature in an iteration



From: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al.

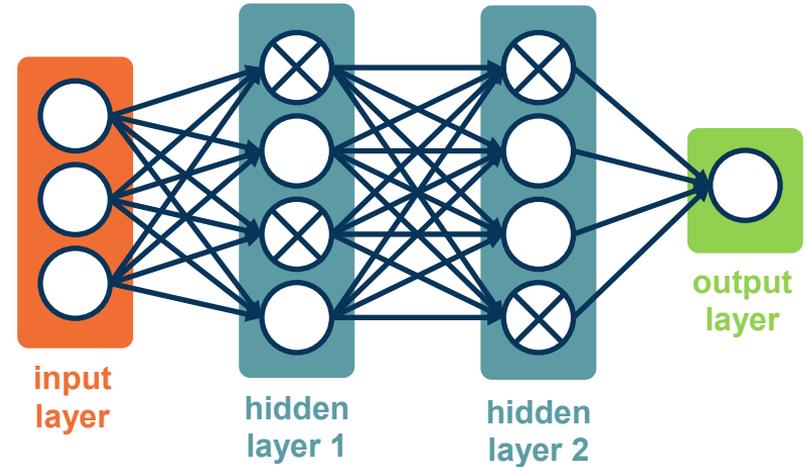
Why Dropout Works

Interpretation 1: The model should not rely too heavily on particular features

- ◆ If it does, it has probability $1 - p$ of losing that feature in an iteration

Interpretation 2: Training 2^n networks:

- ◆ Each configuration is a network
- ◆ Most are trained with 1 or 2 mini-batches of data



From: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al.