Lecture 7: CNN Architectures & Advanced Training (Part 2)

Olexandr Isayev

Department of Chemistry, CMU

olexandr@cmu.edu

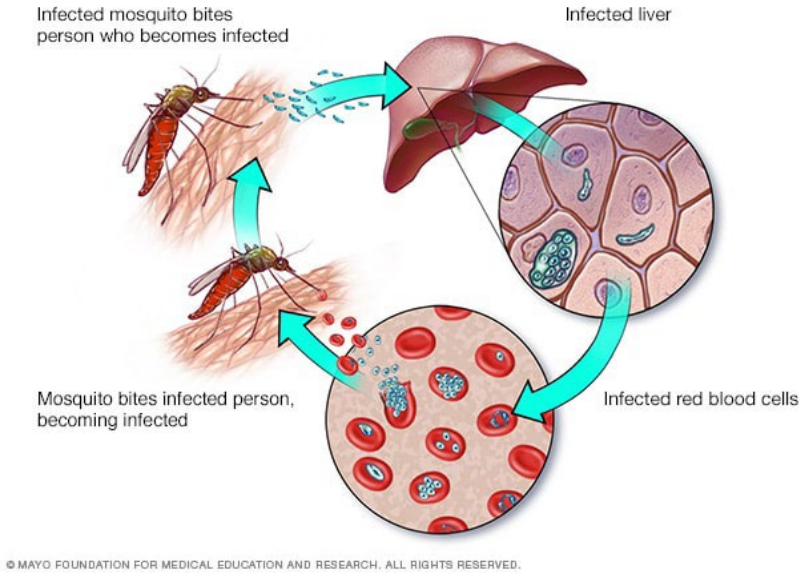
In Class Projects

Please select a problem and dataset

- Upload title and one paragraph summary to canvas.
- I encourage you to use your **domain-specific dataset**
- If not, I am happy to give you one. Just send email or see after class.
- Deadline: March 3

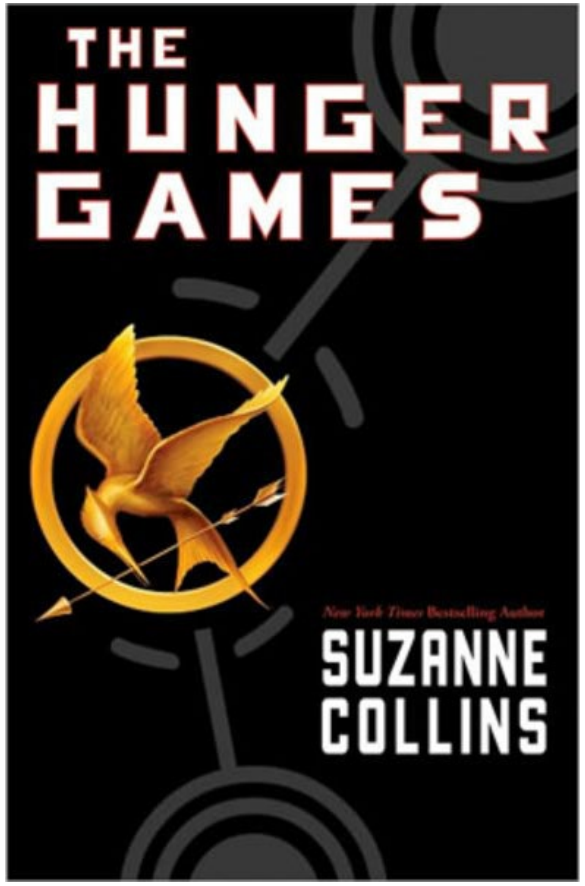
HW2: Fighting Malaria with CNNs!

Malaria, a disease caused by protozoan parasites of the genus Plasmodium, is not only an acute life threat in many developing countries but also a significant burden on the healthcare system worldwide.



In this homework, you are required to <u>implement a convolution neural network-based machine learning model</u> to predict whether the cell in the given picture is parasitized by the genus Plasmodium or not.

Your Mission



Kaggle competition

Get data from kaggle.com
Solve the problem
Submit solution for autograding to Kaggle
Submit IPYNB file to canvas

Deadline: **March 11**

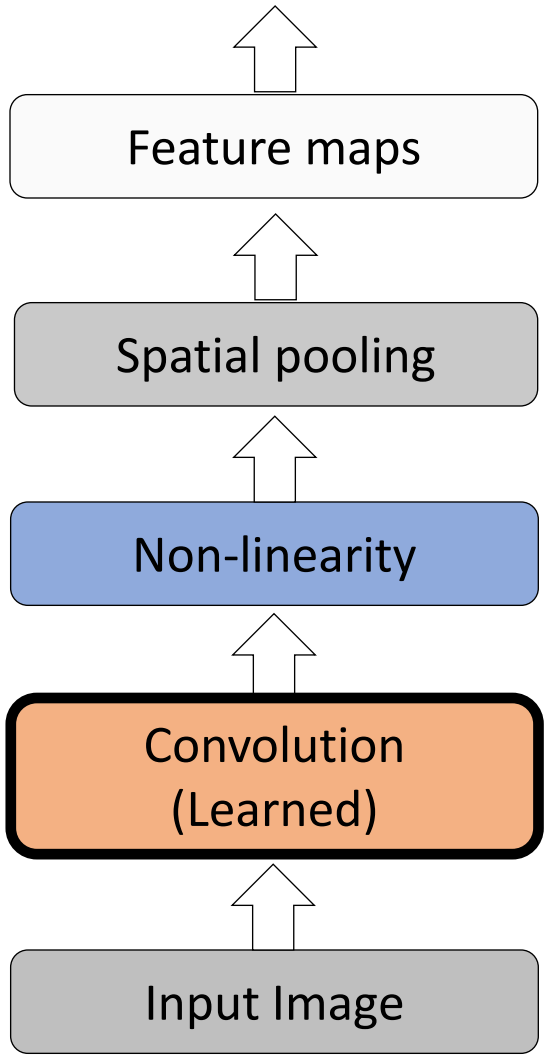
Sign-up Link:

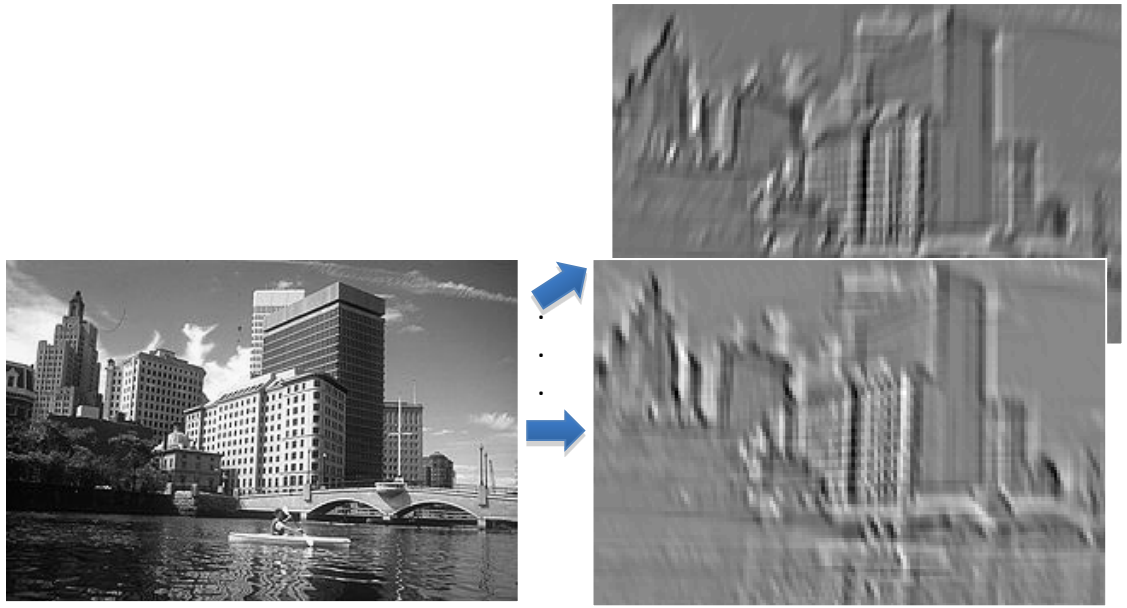
https://www.kaggle.com/t/b7af9038607d45afadb8d22bbdfc0c5d

Convolutional Neural Network (CNN)

- Convolutional Neural Networks (CNN) is the most successful Deep Learning method used to process *multiple arrays*, e.g., 1D for signals, 2D for images, 3D for video.
- CNN consists of a list of Neural Network layers that transform the input data into an output (class/prediction).
- Great success in ImageNet competition in 2012 and later.
- Efficient use of GPUs (NVIDIA), ReLUs (10-20 layers, billions of connections), and dropout for regularization.
- Development of specialized CNN chips (by NVIDIA, Intel, Samsung, etc.) for real-time applications in smartphones, cameras, robots, self-driving cars, etc.

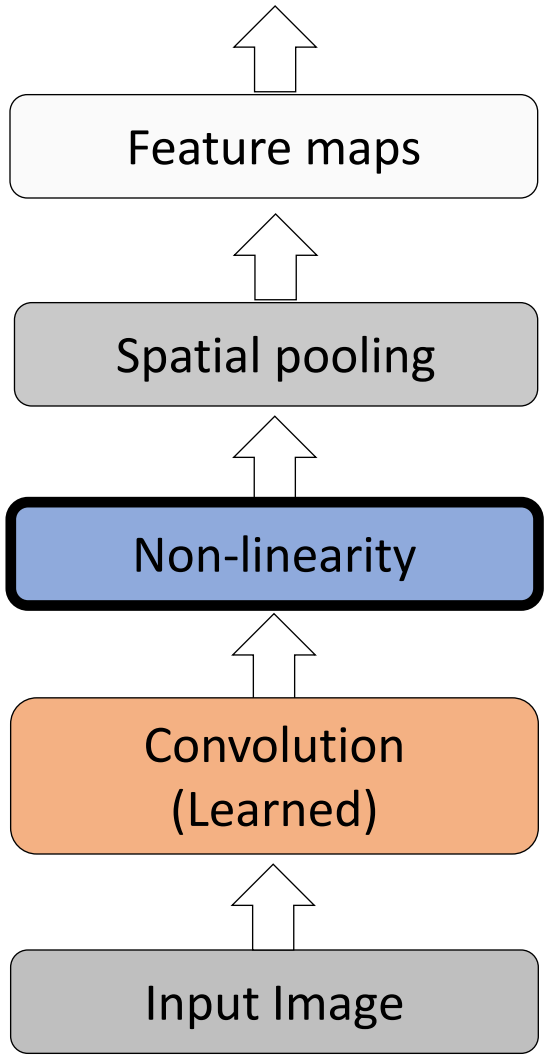
Summary: CNN pipeline





Source: R. Fergus, Y. LeCun

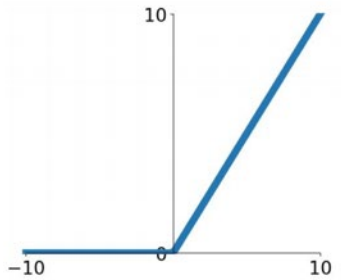
Summary: CNN pipeline



Source: R. Fergus, Y. LeCun

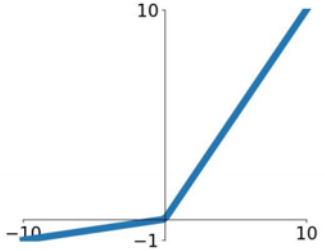
ReLU

$$\max(0, x)$$



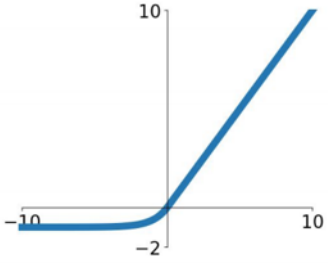
Leaky ReLU

$$\max(0.1x, x)$$



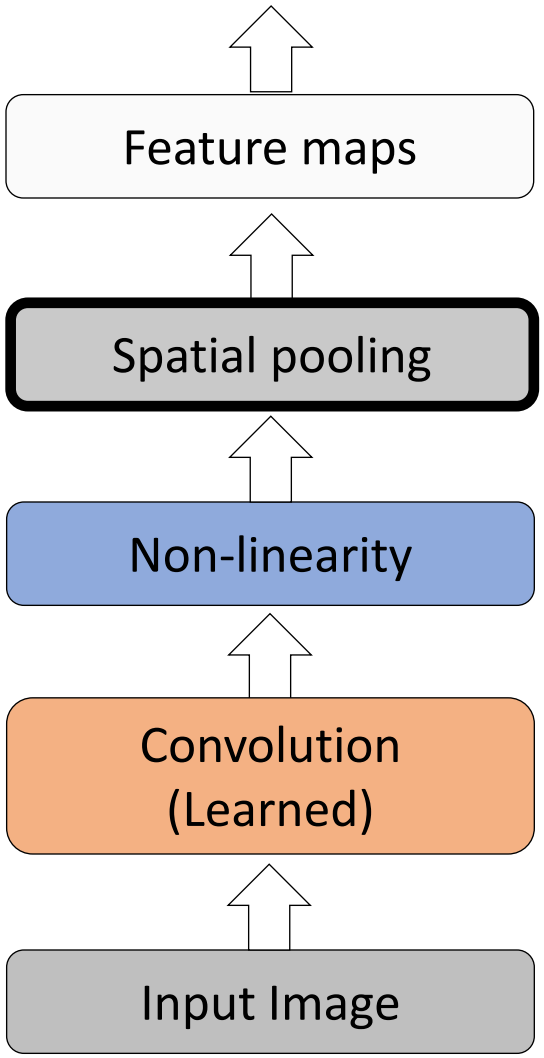
ELU

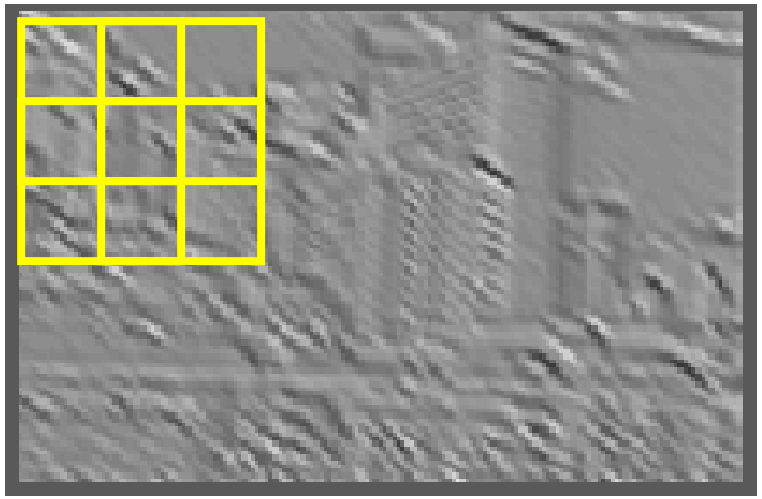
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



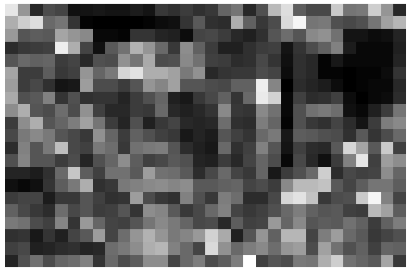
Source: Stanford 231n

Summary: CNN pipeline





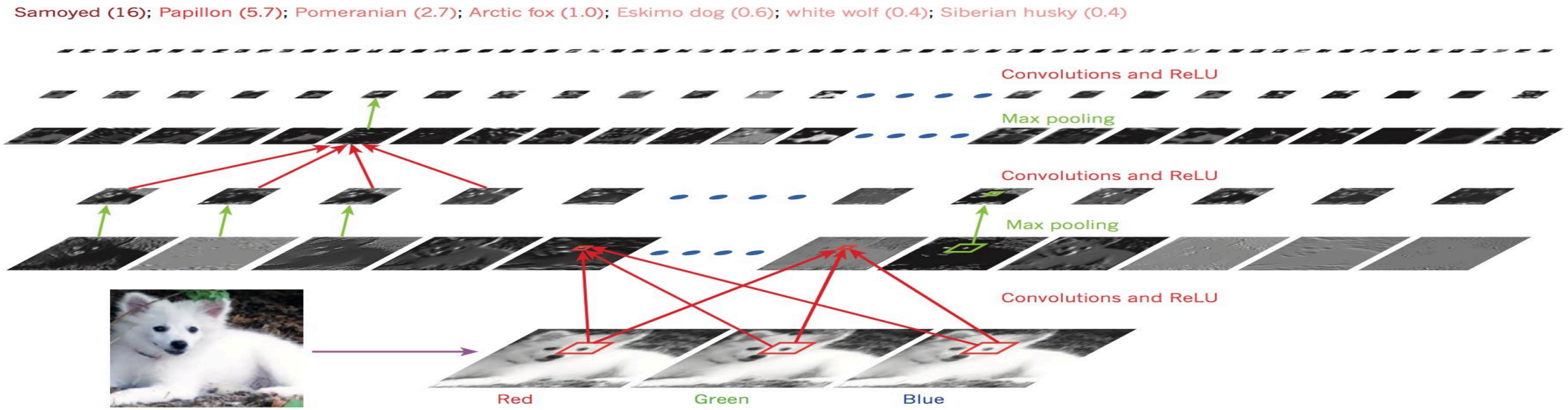
Max
(or Avg)



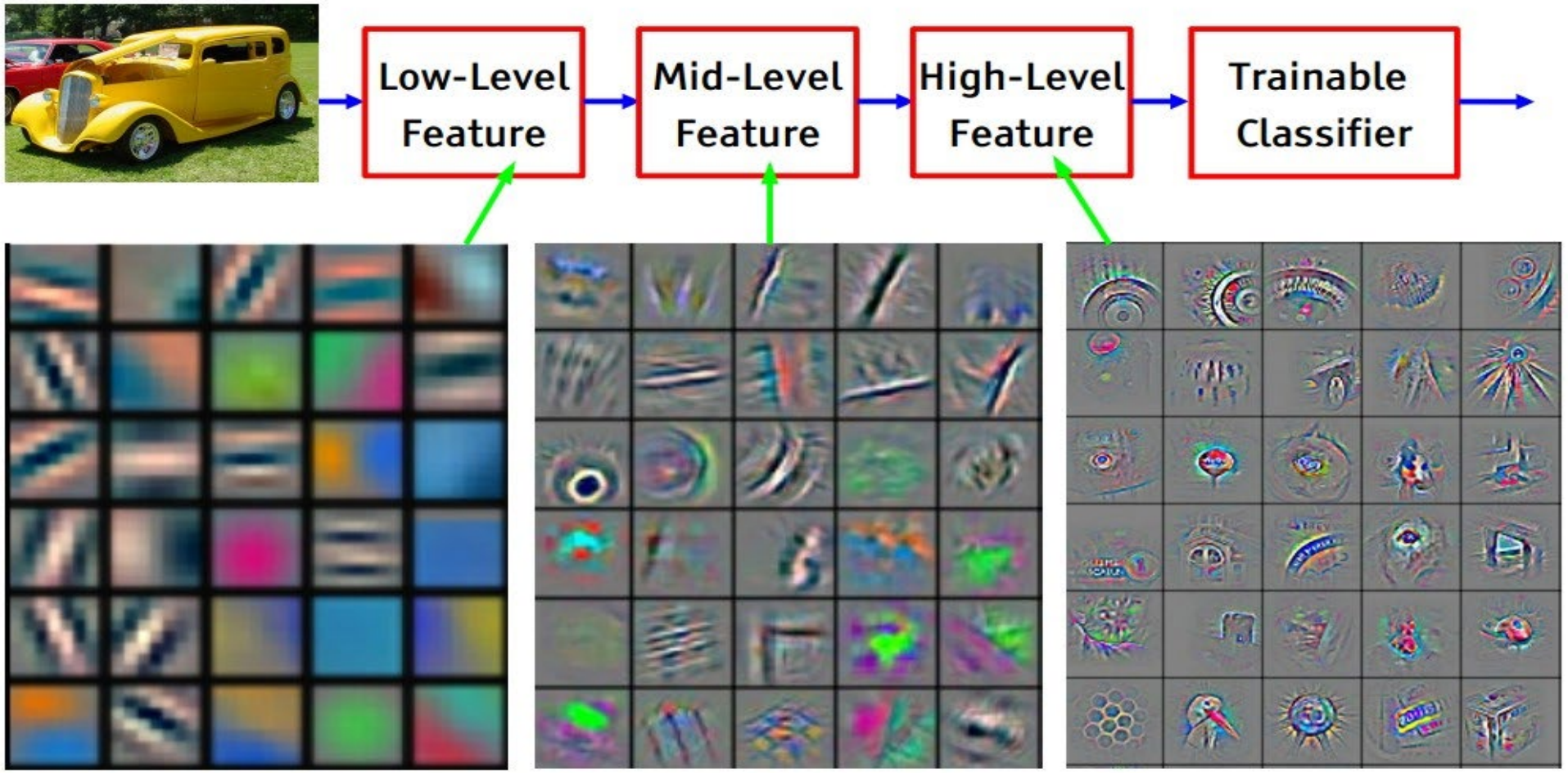
Source: R. Fergus, Y. LeCun

CNN Architecture

- Intuition: Neural network with specialized connectivity structure
  - Stacking multiple layers of feature extractors, low-level layers extract local features, and high-level layers extract learn global patterns.
- There are a few distinct types of layers:
  - **Convolutional Layer**: detecting local features through filters (discrete convolution)
  - **Non-linear Layer**: normalization via Rectified Linear Unit (ReLU)
  - **Pooling Layer**: merging similar features



A Deep Classification Network



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Initial ideas

The first piece of research proposing something similar to a Convolutional Neural Network was authored by Kunihiko Fukushima in 1980, and was called the NeoCognitron[1].

Inspired by discoveries on visual cortex of mammals.

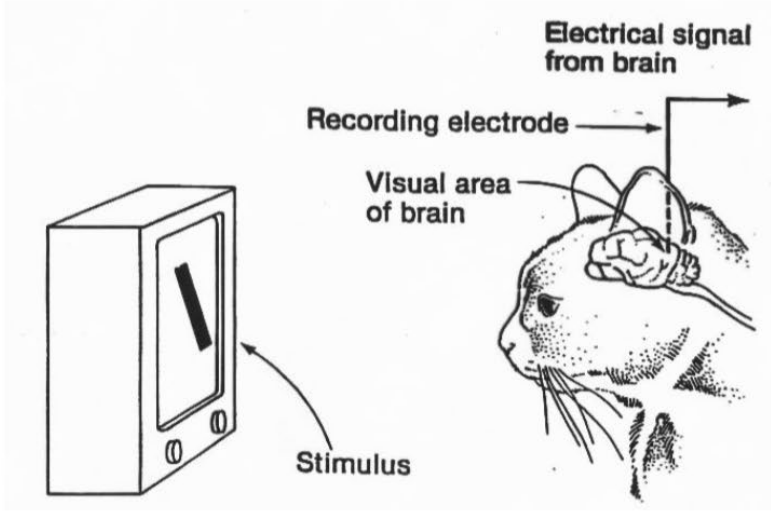
Fukushima applied the NeoCognitron to hand-written character recognition.

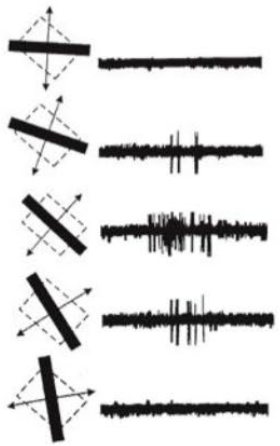
End of the 80's: several papers advanced the field

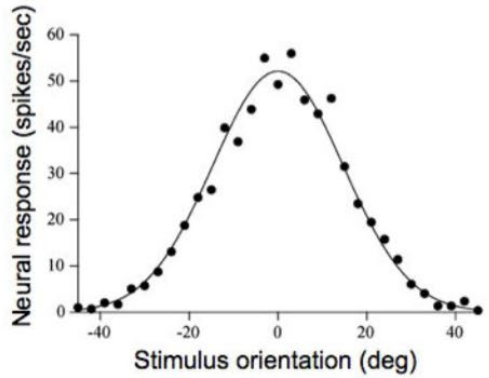
- Backpropagation published in French by Yann LeCun in 1985 (independently discovered by other researchers as well)
- TDNN by Waiber et al., 1989 - Convolutional-like network trained with backprop.
- Backpropagation applied to handwritten zip code recognition by LeCun et al., 1989

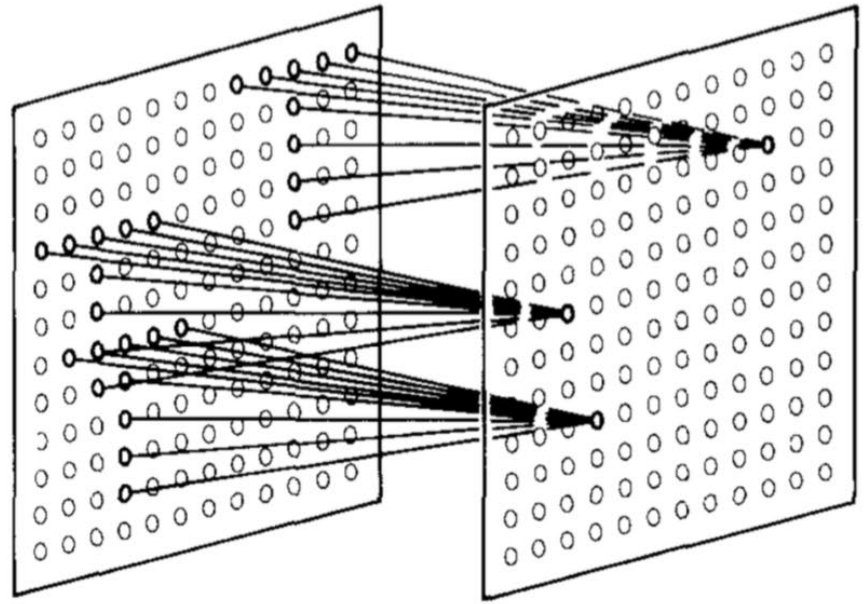
---

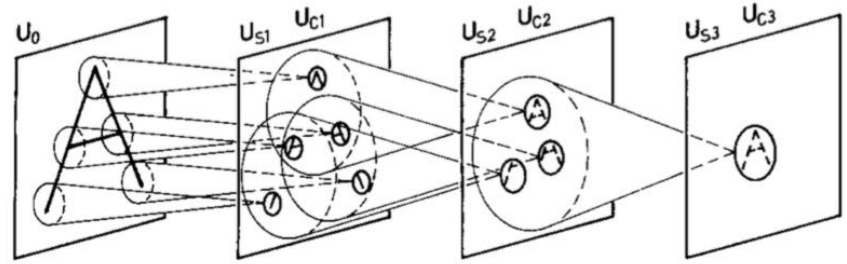
[1] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, 36(4): 93-202, 1980.


Electrical signal
from brain
Recording electrode
Visual area
of brain
Stimulus



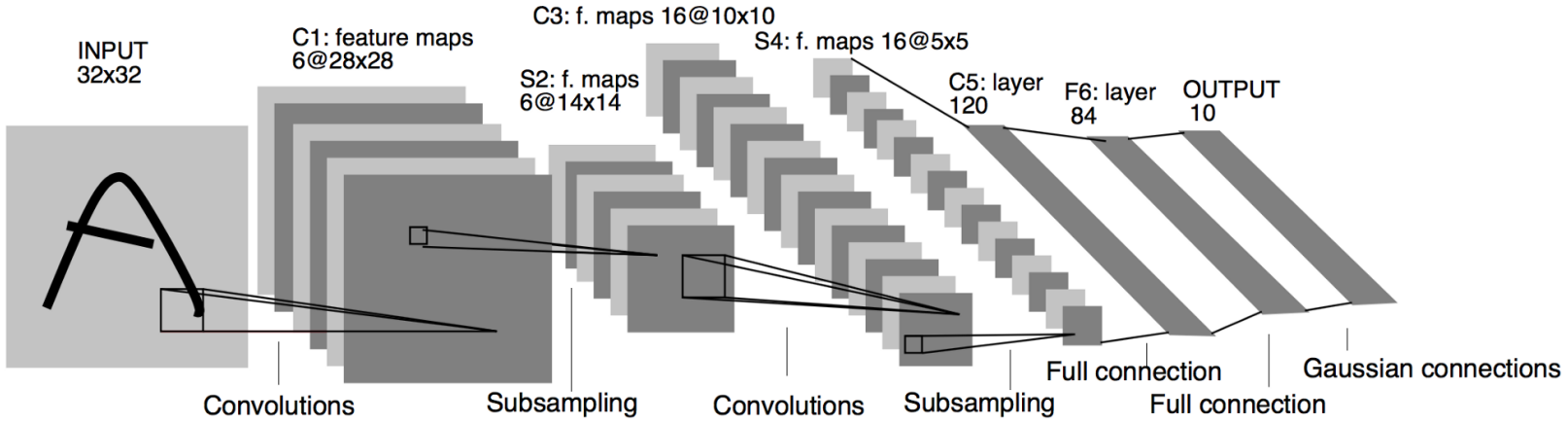

Neural response (spikes/sec)
60
50
40
30
20
10
0
-40
-20
0
20
40
Stimulus orientation (deg)




U0
US1
UC1
US2
UC2
US3
UC3

LeNet

- November 1998: LeCun publishes one of his most recognized papers describing a “modern” CNN architecture for document recognition, called LeNet[1].
- Not his first iteration, this was in fact LeNet-5, but this paper is the commonly cited publication when talking about LeNet.



[1] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.

Architectures

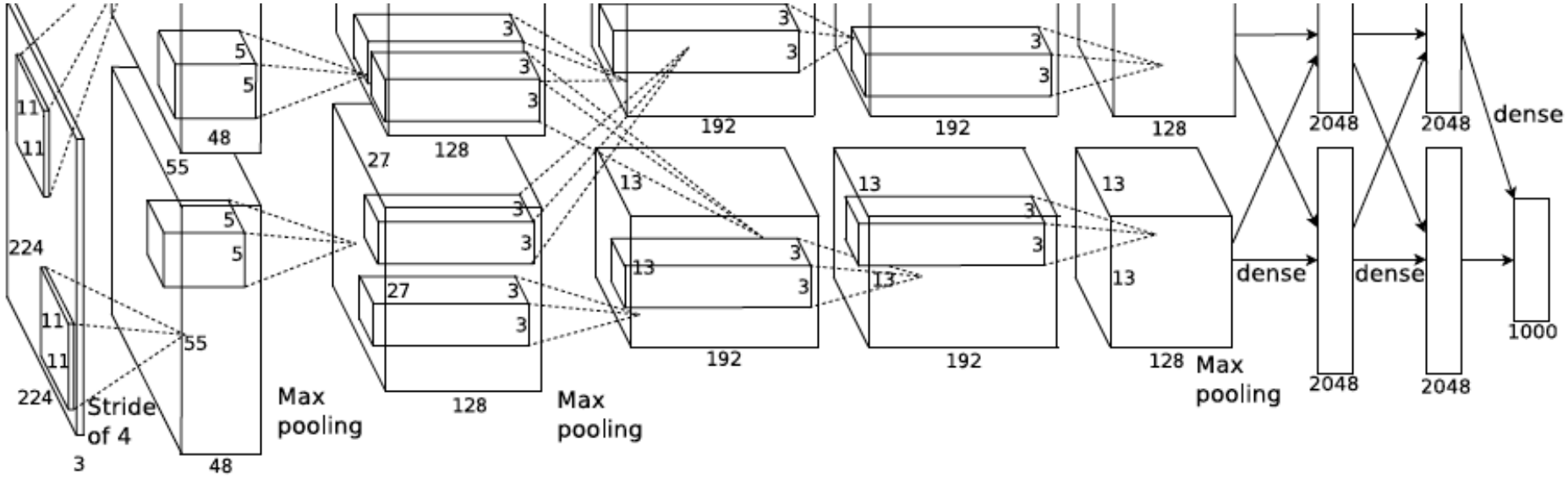
1st generation (2012-2013): AlexNet

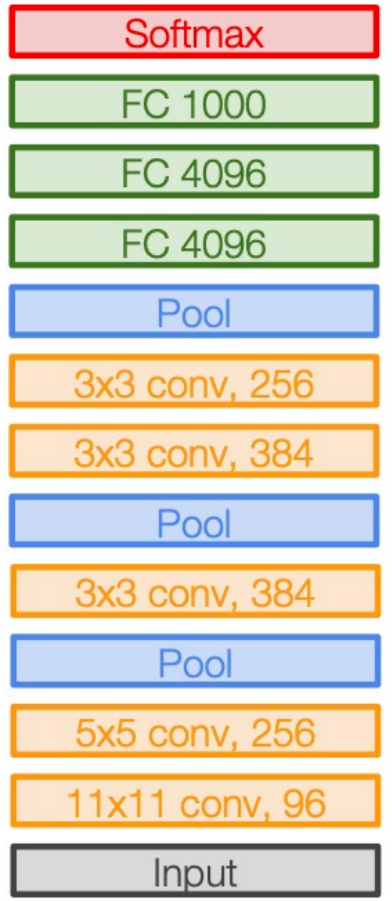
2nd generation (2014): VGGNet, GoogLeNet

3rd generation (2015): ResNet

4th generation (2016): Wide ResNet, ResNeXt, DenseNet

AlexNet: ImageNet 2012 winner





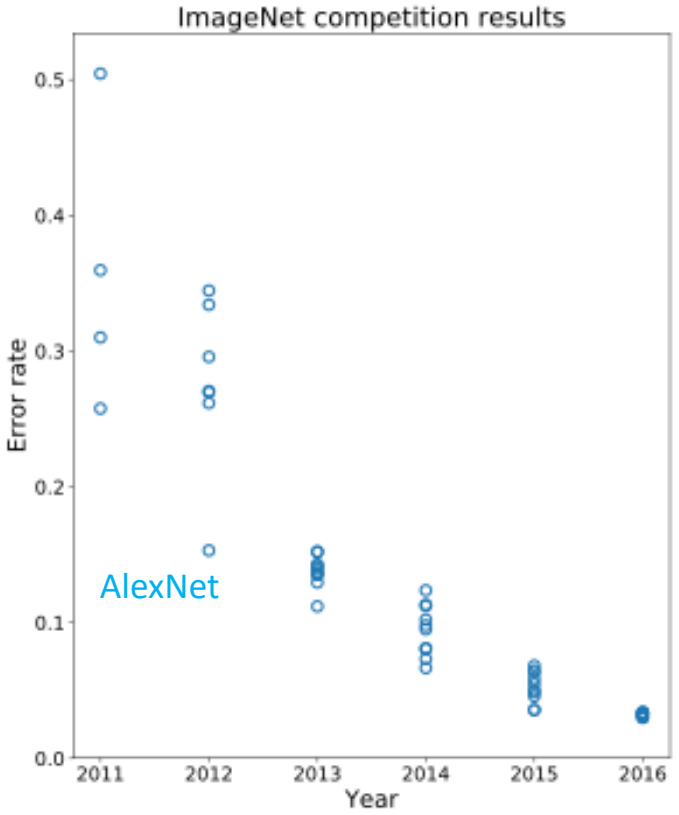
Successor of LeNet-5 with a few crucial changes:

- Max pooling, ReLU nonlinearity
- Dropout regularization
- More data and bigger model (7 hidden layers, 650K units, 60M params)
- GPU implementation (50x speedup over CPU)
  - Trained on two GPUs for a week

A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

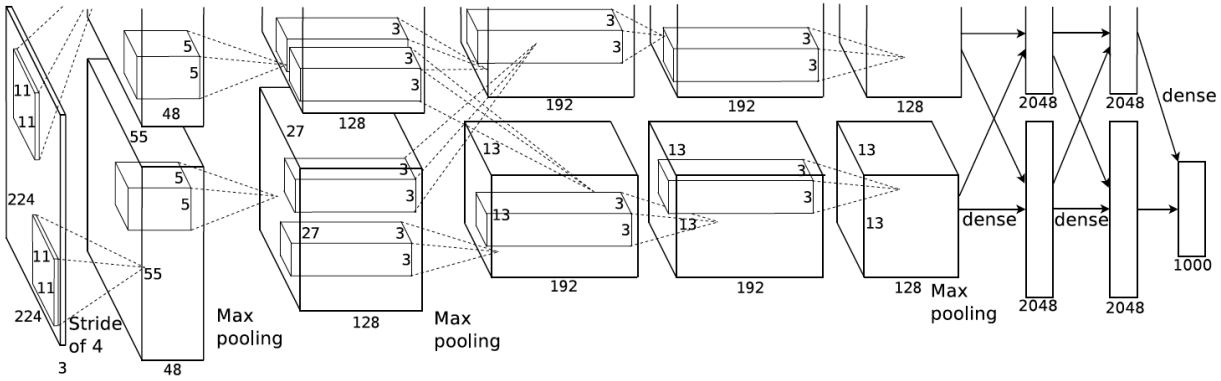
AlexNet

- Developed by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton at Utoronto in 2012. More than 25000 citations.
- Destroyed the competition in the 2012 ImageNet Large Scale Visual Recognition Challenge. Showed benefits of CNNs and kickstarted AI revolution.
- top-5 error of 15.3%, more than 10.8 percentage points lower than runner-up.

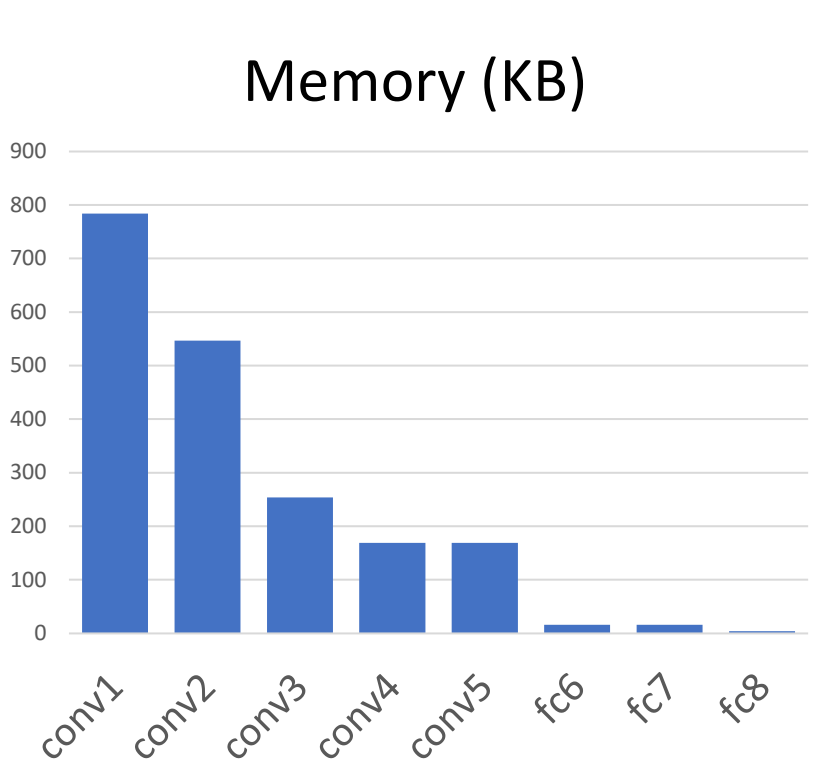


Main contributions:

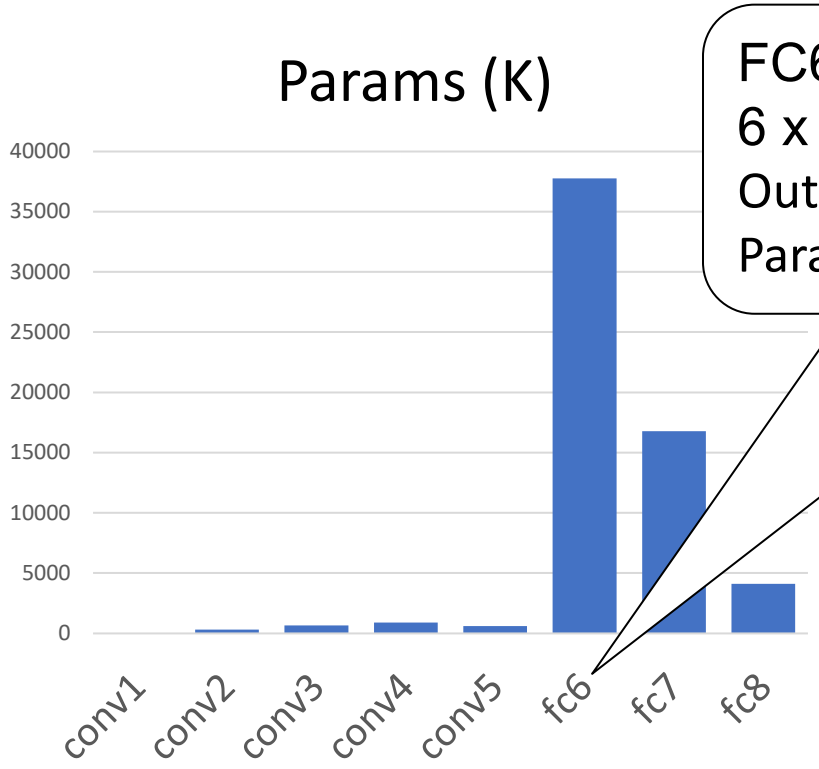
- Trained on ImageNet with data augmentation
- Increased depth of model, GPU training (*five to six days*)
- Smart optimizer and Dropout layers
- ReLU activation!



AlexNet (modified): Analysis

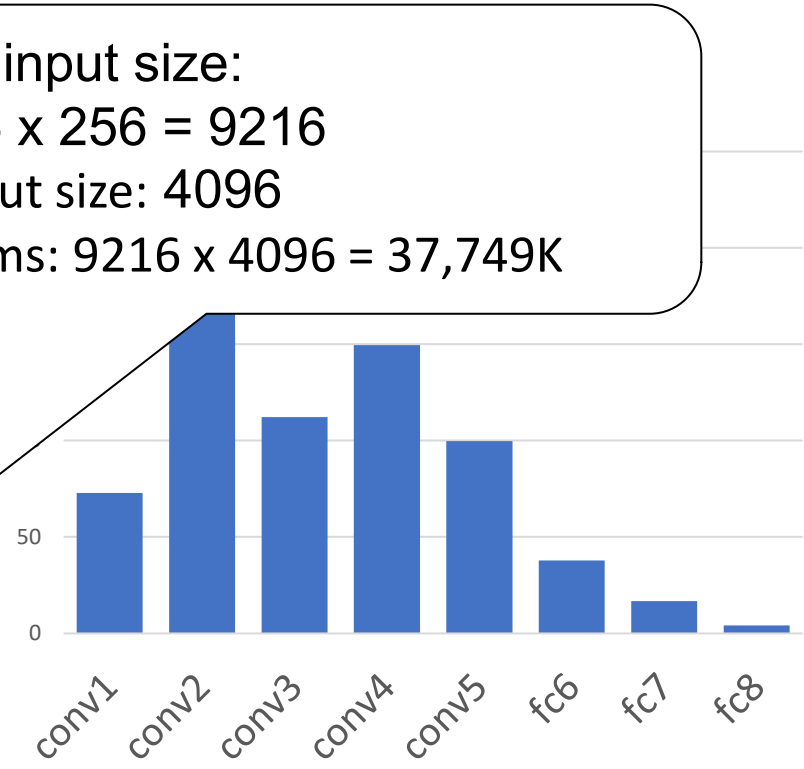


Most of the memory usage is in the early convolution layers



Nearly all parameters are in the fully-connected layers

FC6 input size:
6 x 6 x 256 = 9216
Output size: 4096
Params: 9216 x 4096 = 37,749K



Most floating-point ops occur in the convolution layers

Source: J. Johnson

Clarifai or ZFNet: ImageNet 2013 winner

- R

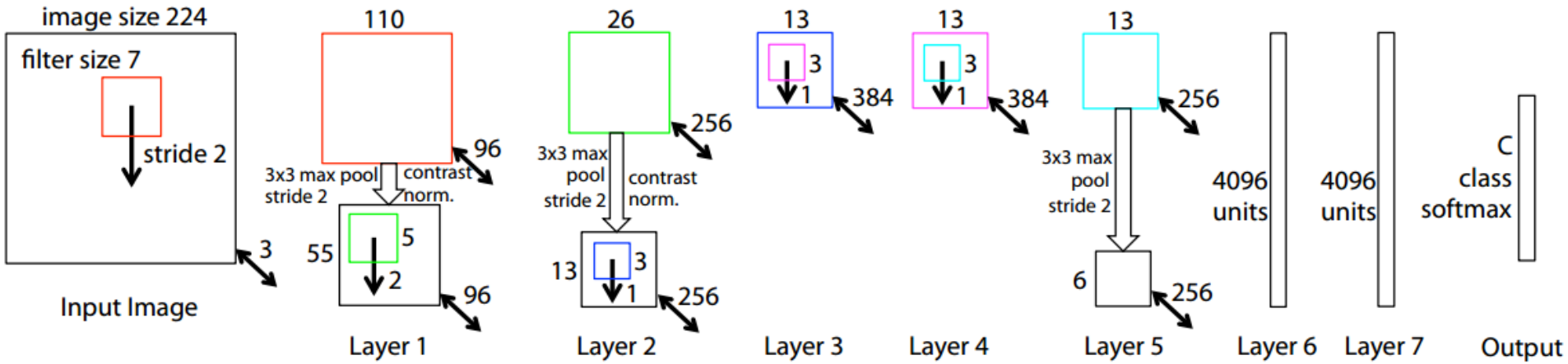
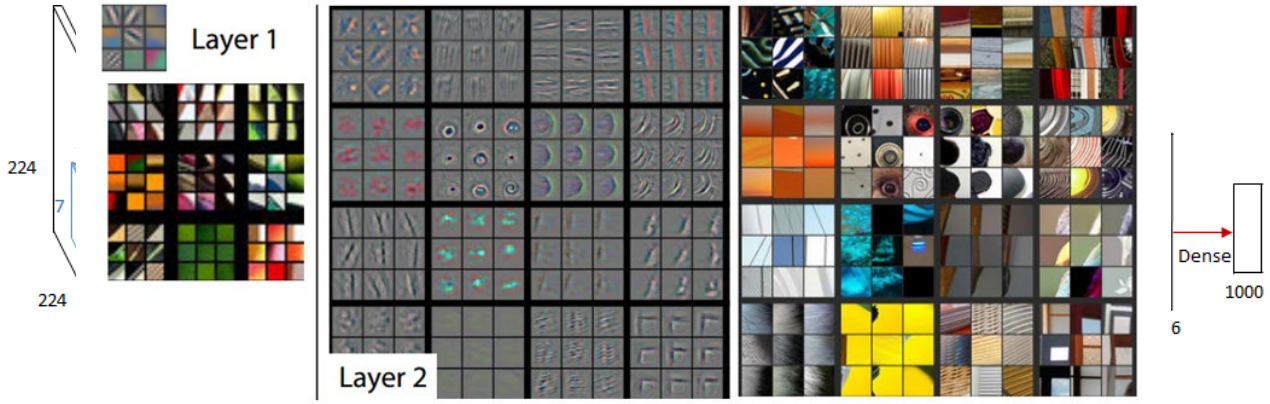


Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through a rectified linear function (not shown), (ii) pooled (max within 3x3 regions, using stride 2) and (iii) contrast normalized across feature maps to give 96 different 55 by 55 element feature maps. Similar operations are repeated in layers 2,3,4,5. The last two layers are fully connected, taking features from the top convolutional layer as input in vector form ($6 \cdot 6 \cdot 256 = 9216$ dimensions). The final layer is a C-way softmax function, C being the number of classes. All filters and feature maps are square in shape.

M. Zeiler and R. Fergus, Visualizing and Understanding Convolutional Networks,
ECCV 2014 (Best Paper Award winner)

ZFNet

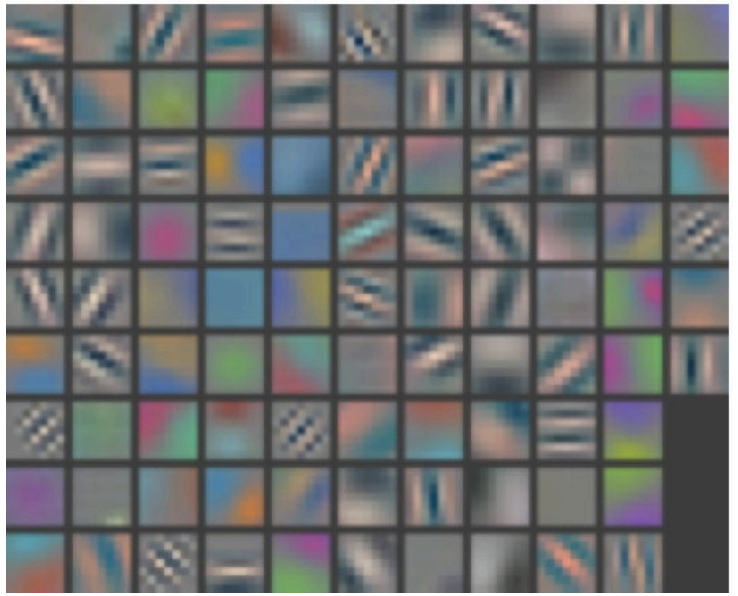
- Introduced by Matthew Zeiler and Rob Fergus from NYU, won ILSVRC 2013 with 11.2% error rate. Decreased sizes of filters.
- Trained for 12 days.
- Paper presented a visualization technique named Deconvolutional Network, which helps to examine different feature activations and their relation to the input space.



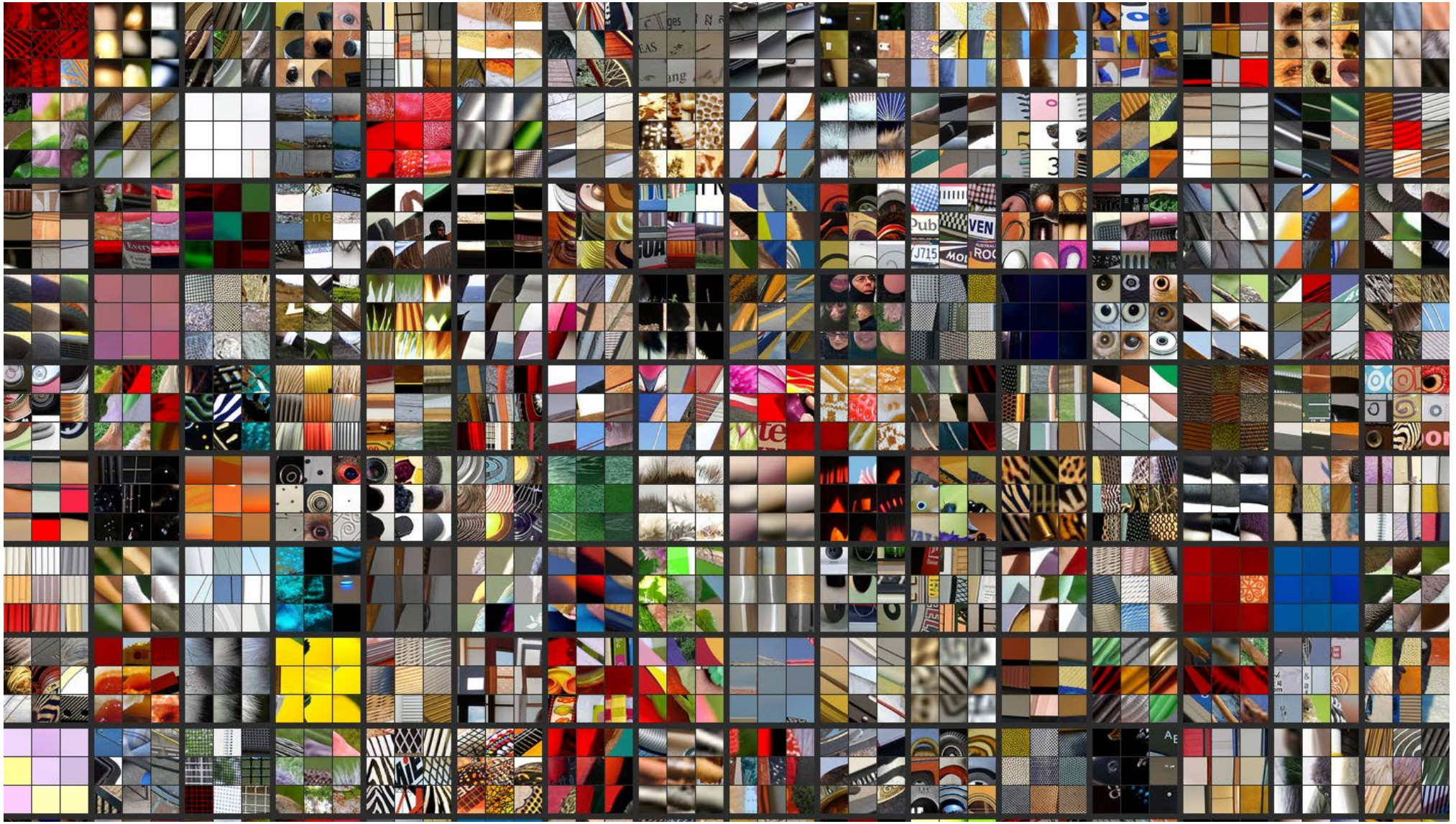
Visualizations of Layer 1 and 2. Each layer illustrates 2 pictures, one which shows the filters themselves and one that shows what part of the image are most strongly activated by the given filter. For example, in the space labled Layer 2, we have representations of the 16 different filters (on the left)

Layer 1: Top-9 Patches

Layer 1 filters



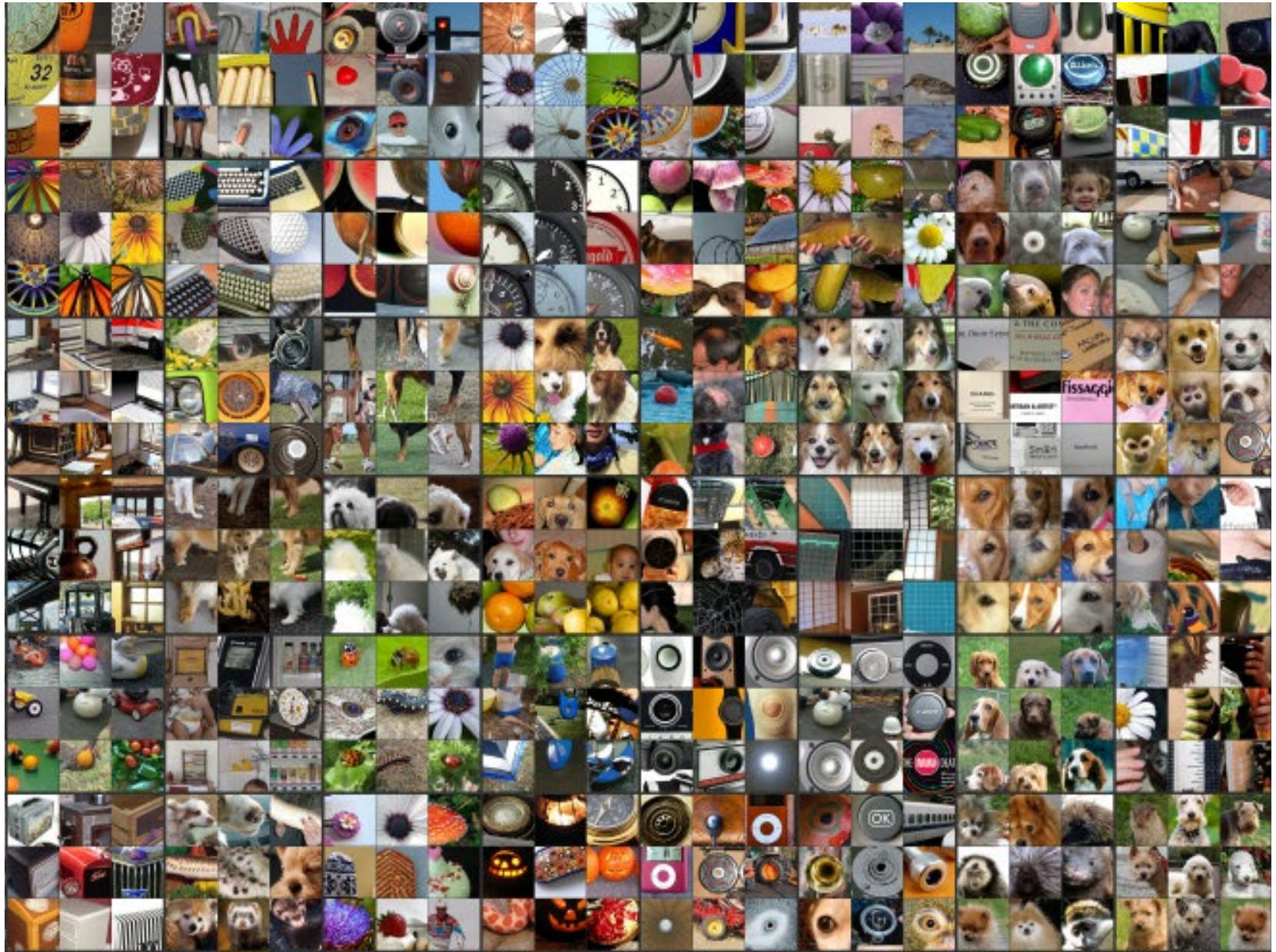
Layer 2: Top-9 Patches



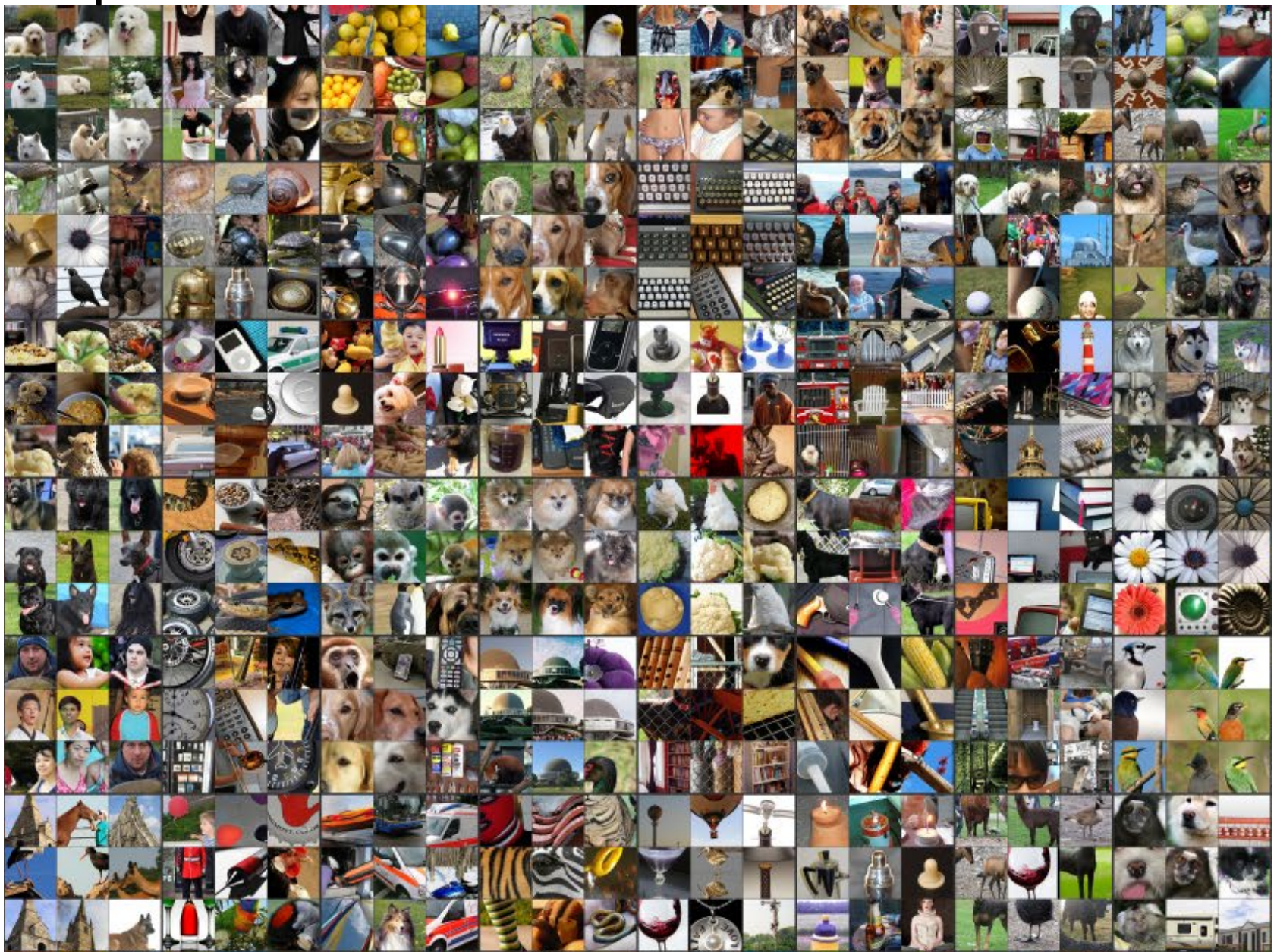
- Patches from validation images that give maximal activation of a given feature map

Layer 3: Top-9 Patches

Layer 4: Top-9 Patches

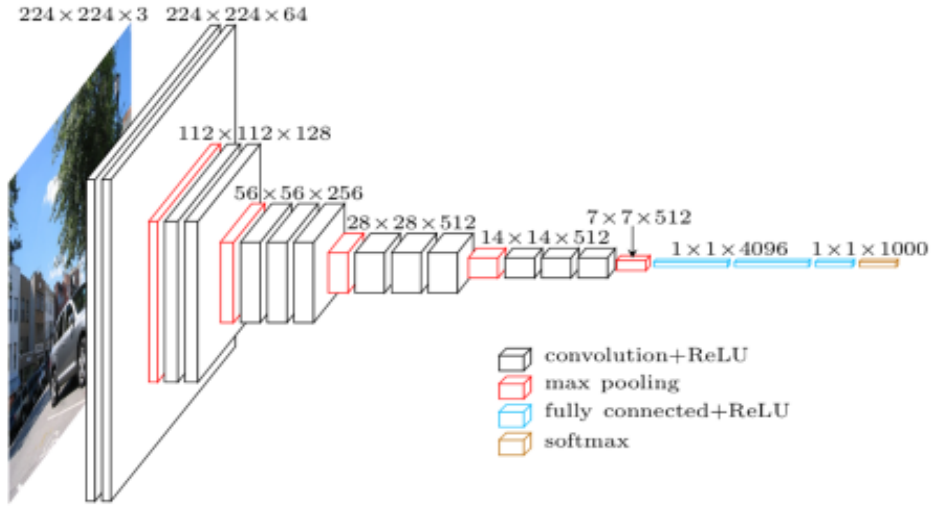


Layer 5: Top-9 Patches



VGG

- Introduced by Simonyan and Zisserman (Oxford) in 2014
- Simplicity and depth as main points. Used 3x3 filters exclusively and 2x2 MaxPool layers with stride 2.
- Showed that two 3x3 filters have an effective receptive field of 5x5.
- As spatial size decreases, depth increases.
- Trained for *two to three weeks*.
- Still used as of today.



VGGNet vs. AlexNet

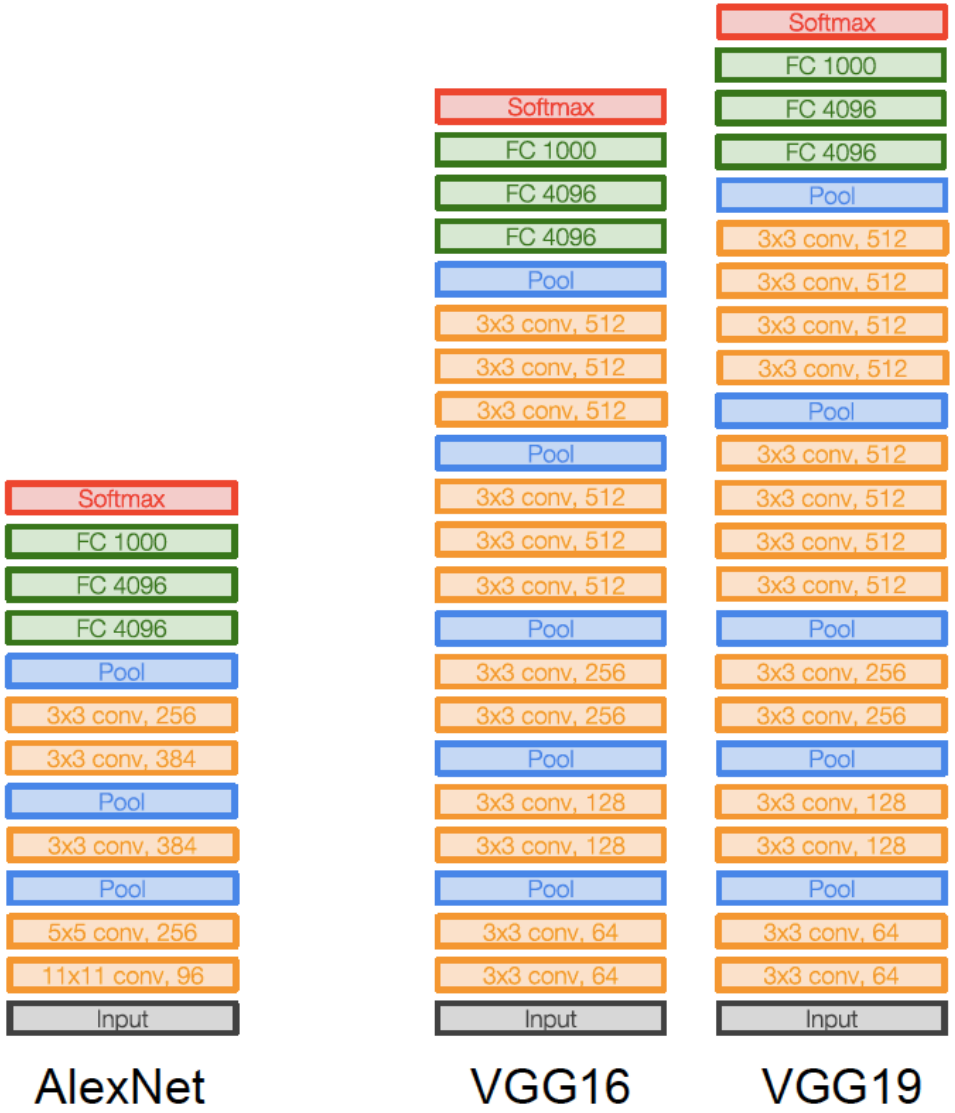
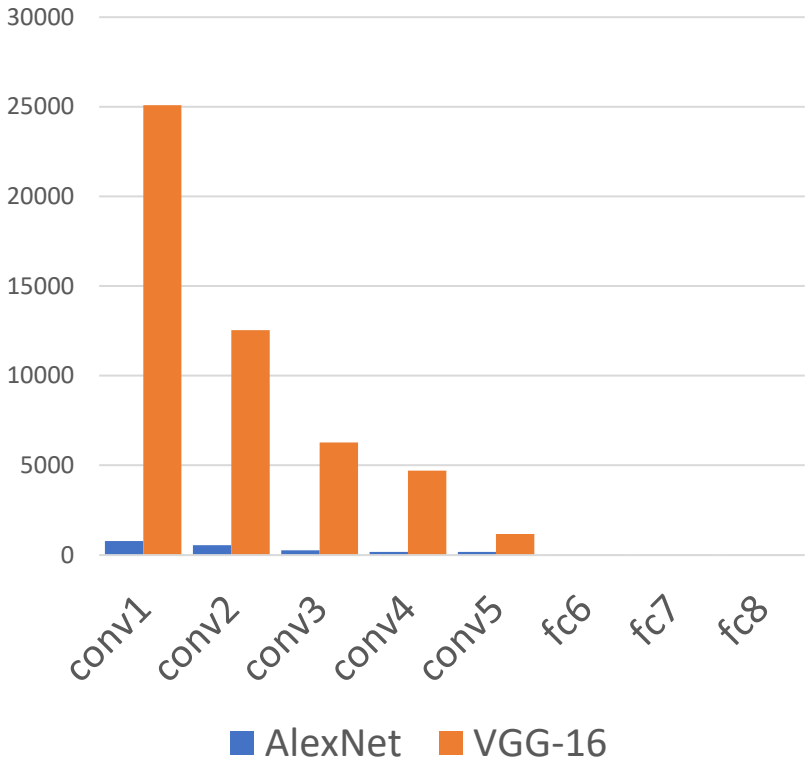


Image source

K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015

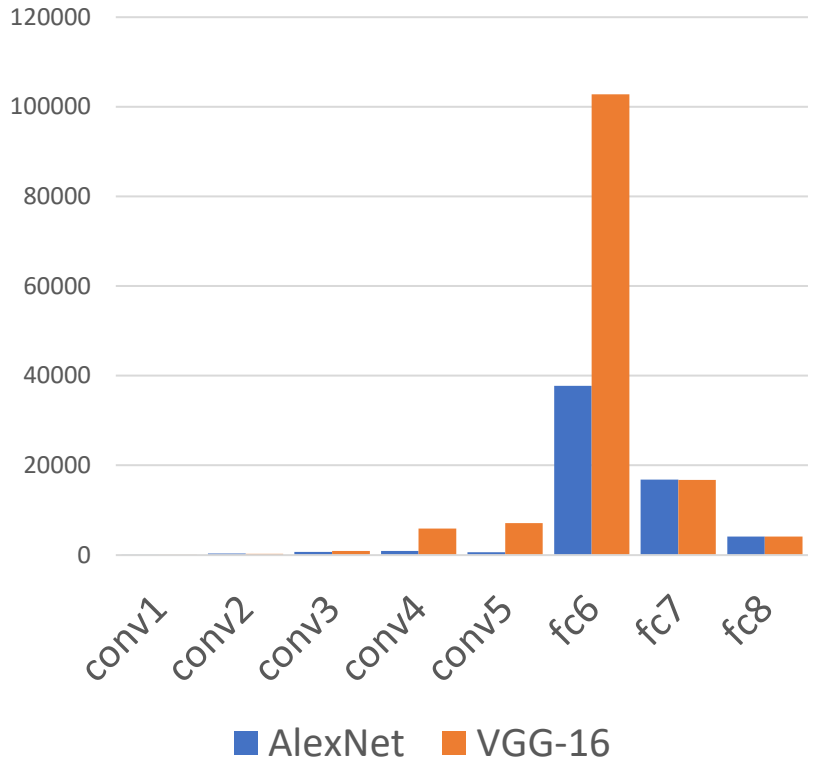
VGGNet vs. AlexNet

AlexNet vs VGG-16
(Memory, KB)



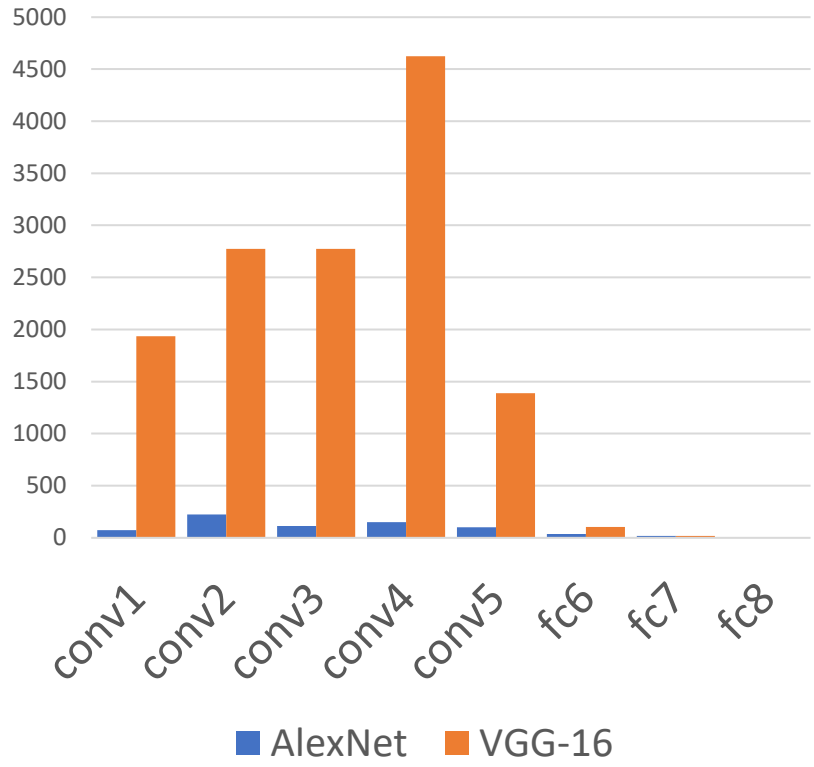
AlexNet total: 1.9 MB
VGG-16 total: 48.6 MB (25x)

AlexNet vs VGG-16
(Params, M)



AlexNet total: 61M
VGG-16 total: 138M (2.3x)

AlexNet vs VGG-16
(MFLOPs)

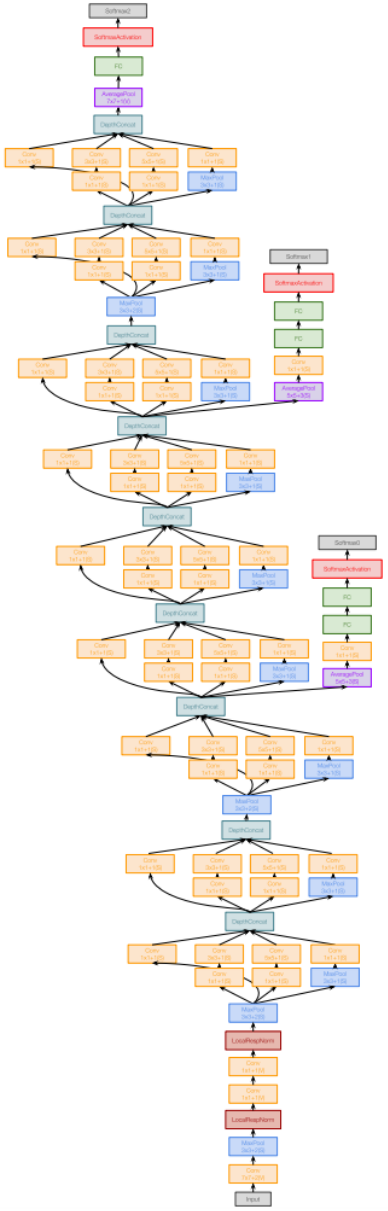


AlexNet total: 0.7 GFLOP
VGG-16 total: 13.6 GFLOP (19.4x)

Source: J. Johnson

GoogLeNet: ImageNet 2014 winner

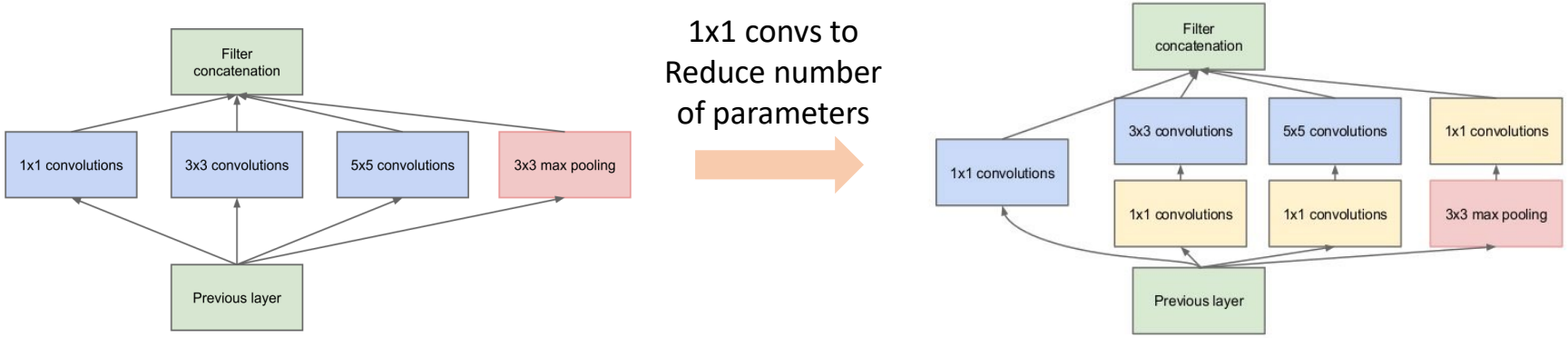
http://knowyourmeme.com/memes/we-need-to-go-deeper



C. Szegedy et al., Going deeper with convolutions, CVPR 2015

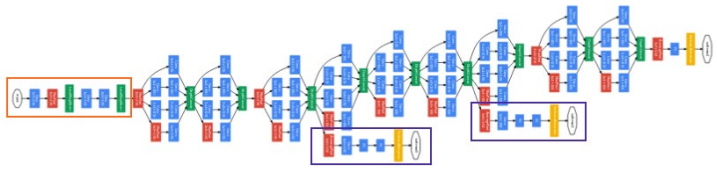
GoogLeNet (Inception-v1)

- Introduced by Szegedy et al. (Google), 2014. Winners of ILSVRC 2014.
- Introduces inception module: parallel conv. layers with different filter sizes. Motivation: we don't know which filter size is best – let the network decide. Key idea for future archs.
- No fully connected layer at the end. AvgPool instead. 12x fewer params than AlexNet.



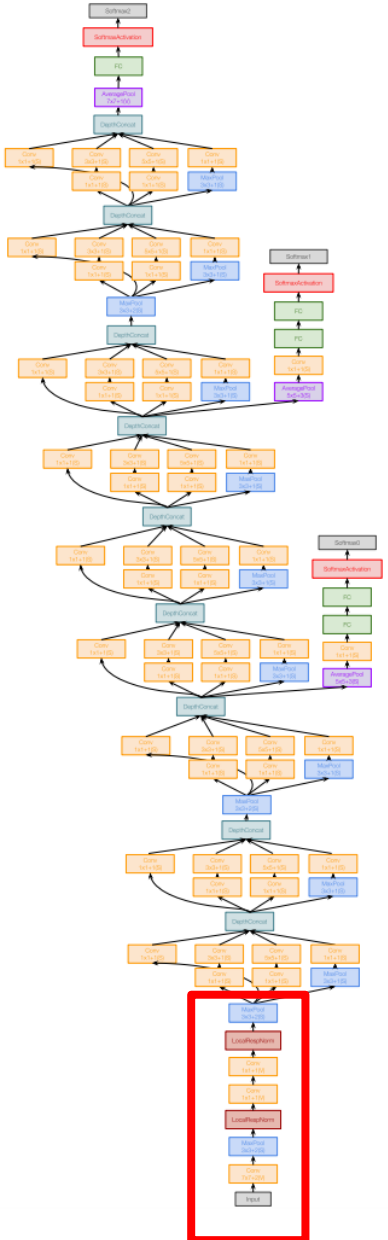
Proto Inception module

Inception module



GoogLeNet: Aggressive stem

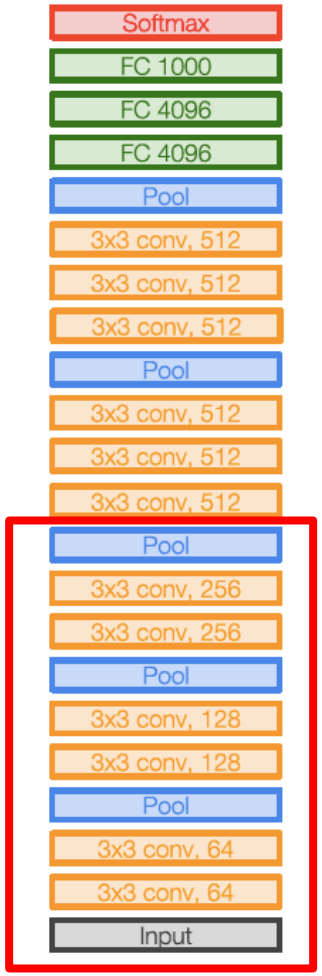
- **Stem network** at the start aggressively downsamples input



Source: J. Johnson

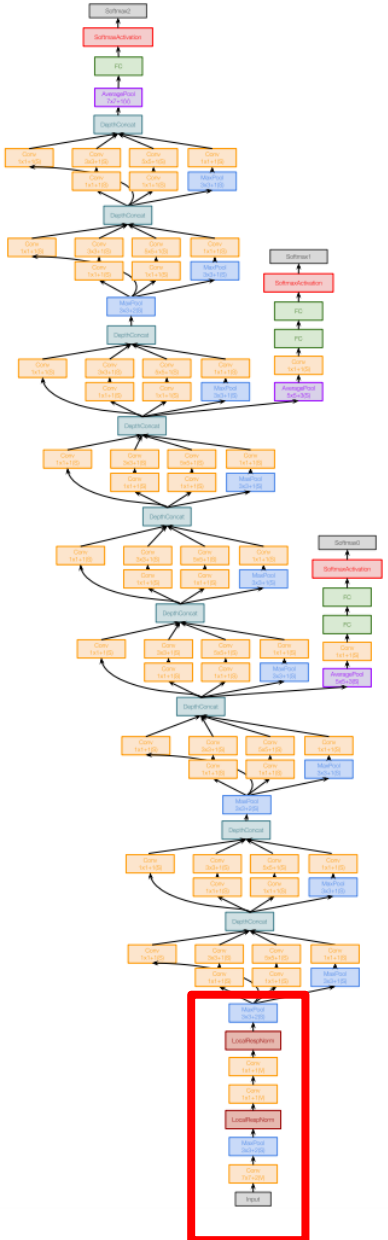
GoogLeNet: Aggressive stem

Compare VGG-16:
Memory: 42.9 MB (5.7x)
Params: 1.1M (8.9x)
MFLOP: 7485 (17.8x)



VGG16

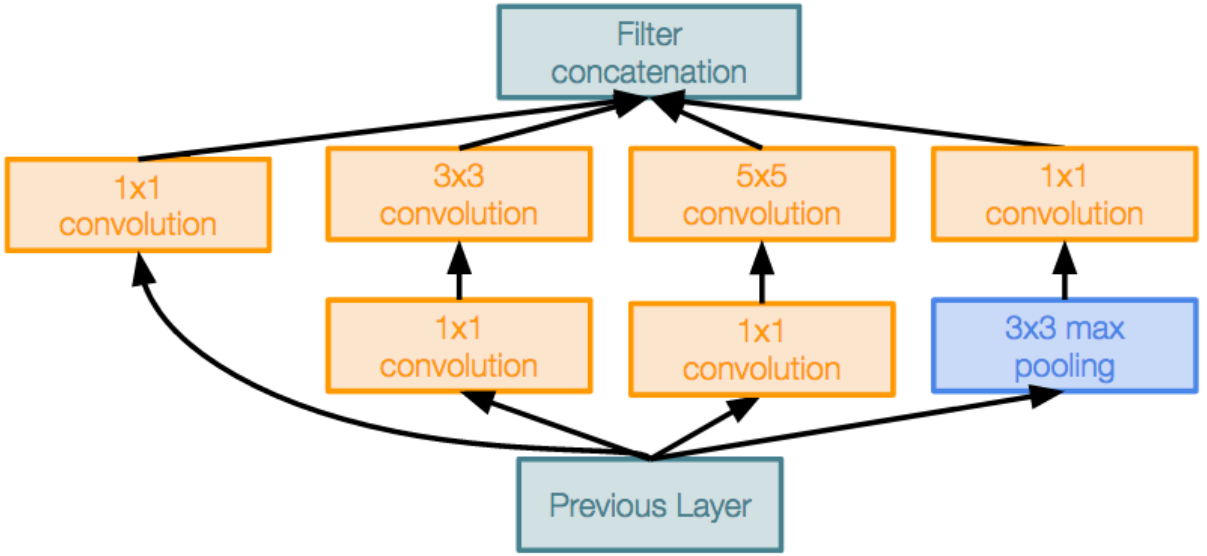
Total from 224 to 28 resolution:
Memory: 7.5 MB
Params: 124K
MFLOP: 418

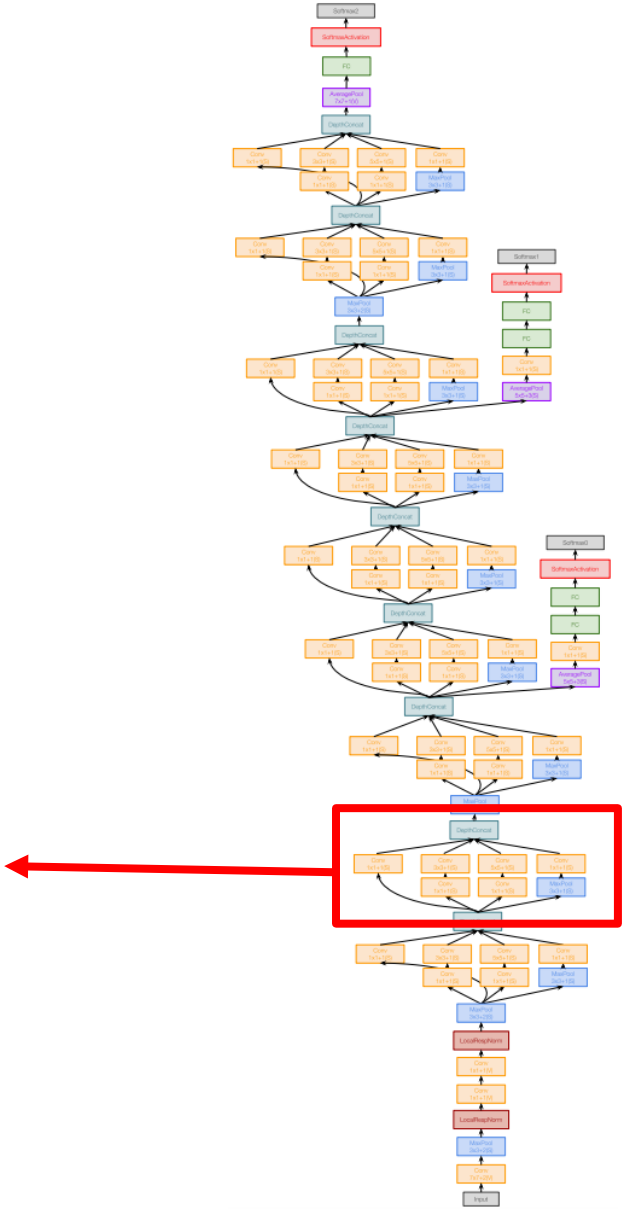


Source: J. Johnson

GoogLeNet: Inception module

- Parallel paths with different receptive field sizes and operations are meant to capture sparse patterns of correlations in the stack of feature maps
- Use 1×1 convolutions for dimensionality reduction before expensive convolutions





Source: J. Johnson

Why 1x1 convolutions?

- **Option 1:** 3×3 conv layer with 256 channels at input and output
  - $256 \times 256 \times 3 \times 3 \approx 600{,}000$ operations (per location)

- **Option 2: "bottleneck module"**

  1×1 conv layer, $256 \rightarrow 64$ channels

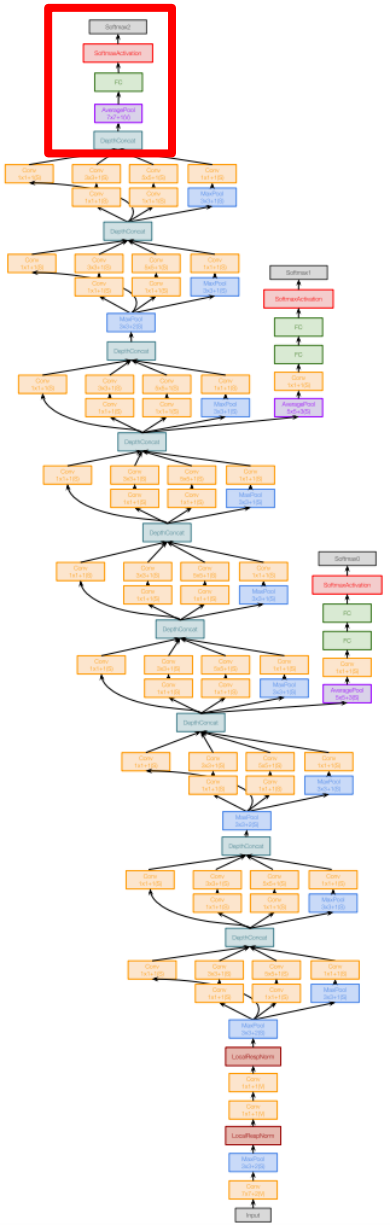
  3×3 conv layer, $64 \rightarrow 64$ channels

  1×1 conv layer, $64 \rightarrow 256$ channels

  - $256 \times 64 \times 1 \times 1 \approx 16{,}000$
  - $64 \times 64 \times 3 \times 3 \approx 36{,}000$
  - $64 \times 256 \times 1 \times 1 \approx 16{,}000$
  - Total $\approx 70{,}000$ operations

GoogLeNet: Global average pooling

- Instead of large FC layers at the end, use *global average pooling* to collapse spatial dimensions, followed by linear layer to produce class scores
  - Recall VGG-16: Most parameters were in the FC layers!

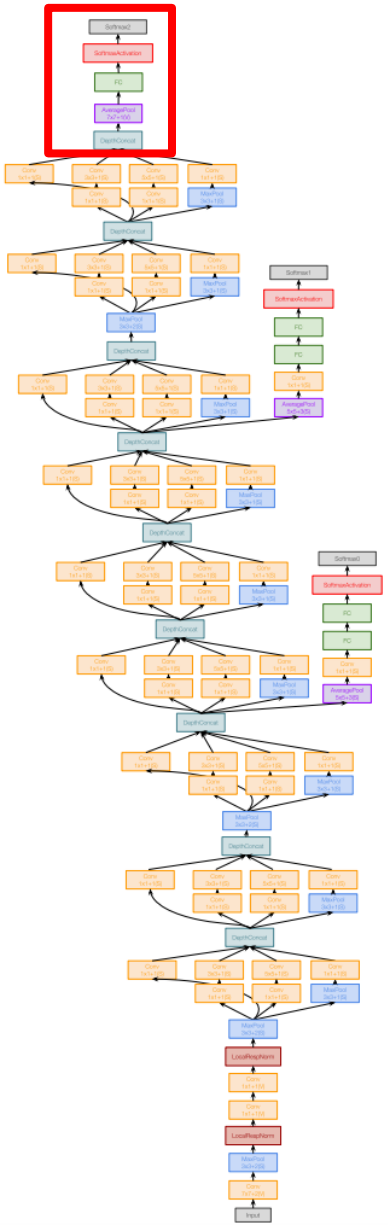


Source: J. Johnson

GoogLeNet: Global average pooling

GoogLeNet head:

	Input size		Layer				Output size				
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (k)	flop (M)
avg-pool	1024	7		7	1	0	1024	1	4	0	0
fc	1024		1000				1000		0	1025	1



Source: J. Johnson

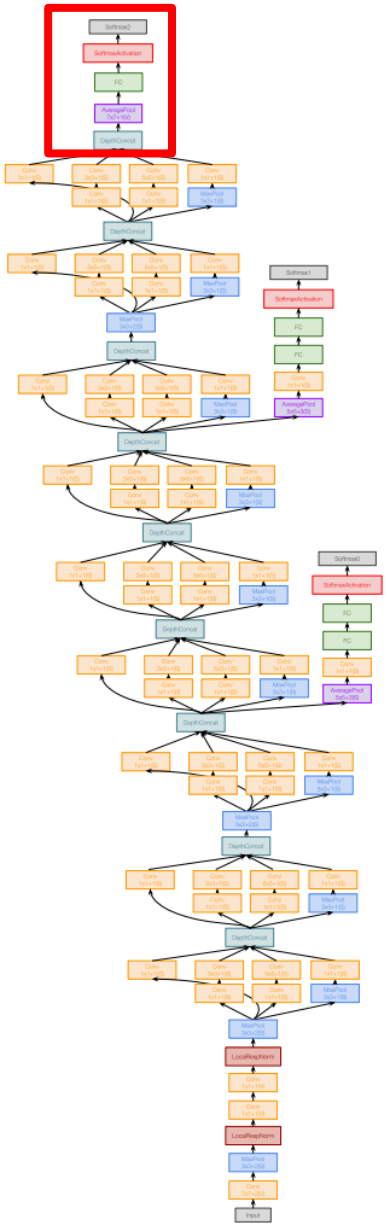
GoogLeNet: Global average pooling

GoogLeNet head:

	Input size		Layer				Output size				
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (k)	flop (M)
avg-pool	1024	7		7	1	0	1024	1	4	0	0
fc	1024		1000				1000		0	1025	1

Compare with VGG-16:

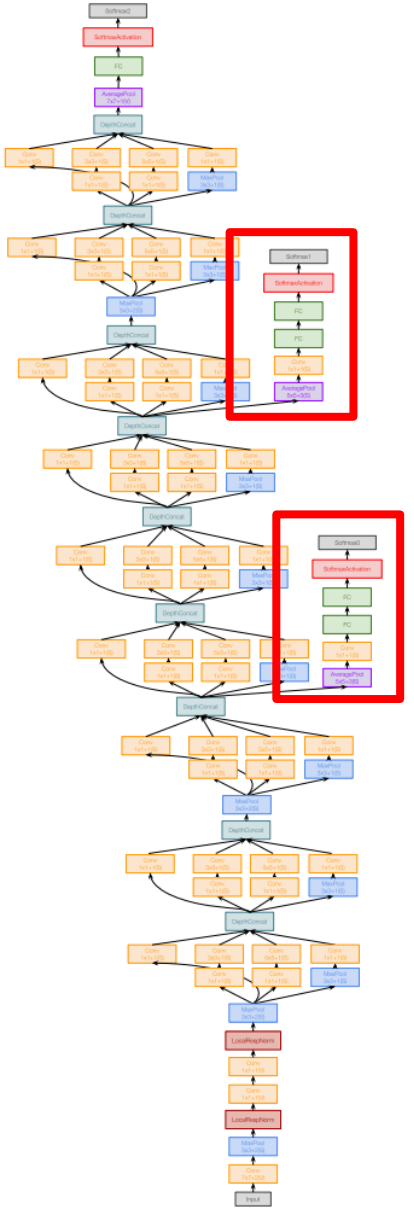
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (k)	flop (M)
flatten	512	7					25088		98		
fc6	25088			4096			4096		16	102760	103
fc7	4096			4096			4096		16	16777	17
fc8	4096			1000			1000		4	4096	4



Source: J. Johnson

GoogLeNet: Auxiliary classifiers

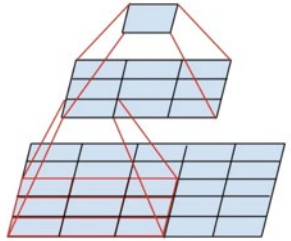
- Training using loss at the end of the network didn't work well: network is too deep, gradients don't propagate cleanly
- As a hack, attach "auxiliary classifiers" at several intermediate points in the network that also try to classify the image and receive loss
- GoogLeNet was before batch normalization! With BatchNorm no longer need to use this trick

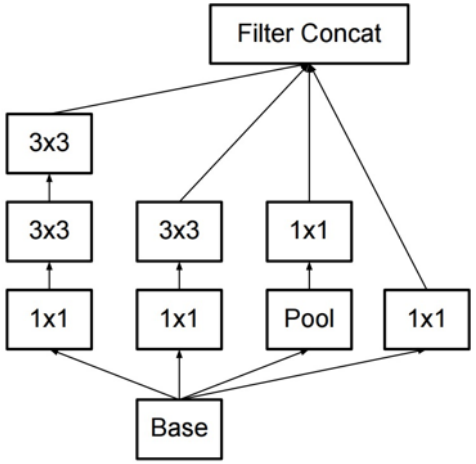


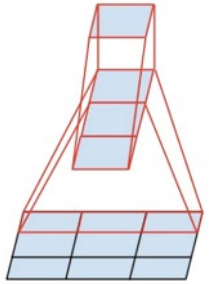
Source: J. Johnson

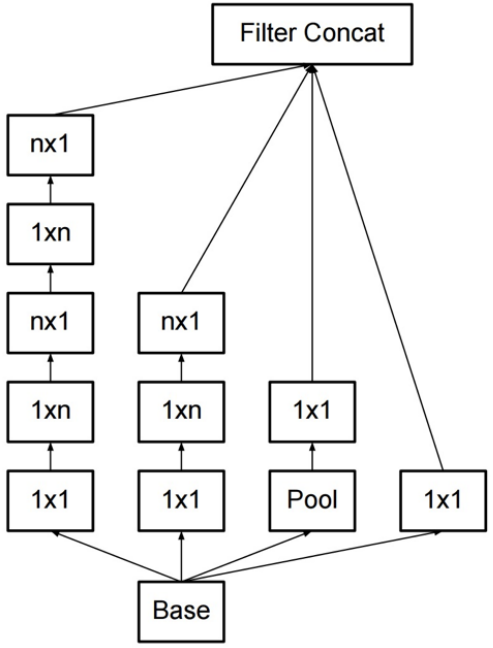
Inception v2, v3

- Improve training with batch normalization, eliminating need for auxiliary classifiers
- More variants of inception modules with aggressive factorization of filters





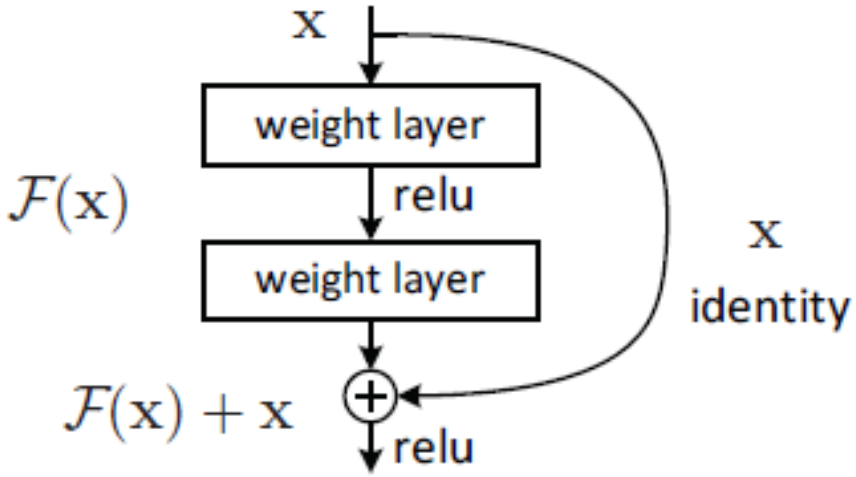




C. Szegedy et al., Rethinking the inception architecture for computer vision, CVPR 2016

ResNet

- Presented by He et al. (Microsoft), 2015. Won ILSVRC 2015 in multiple categories.
- Main idea: Residual block. Allows for extremely deep networks.
- Authors believe that it is easier to optimize the residual mapping than the original one. Furthermore, residual block can decide to “shut itself down” if needed.



Residual Block

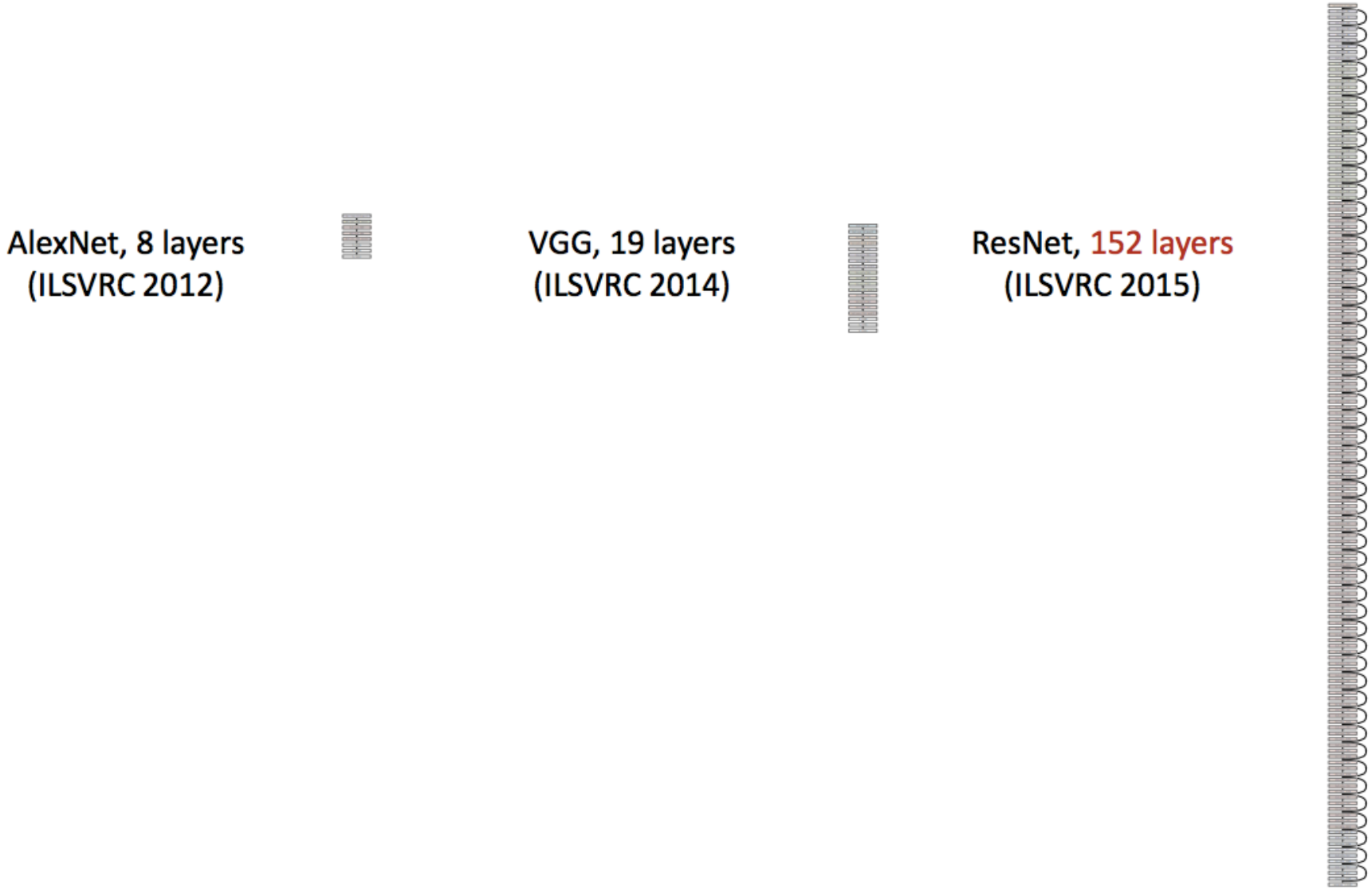
ResNet: ImageNet 2015 winner

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition,
CVPR 2016 (Best Paper)

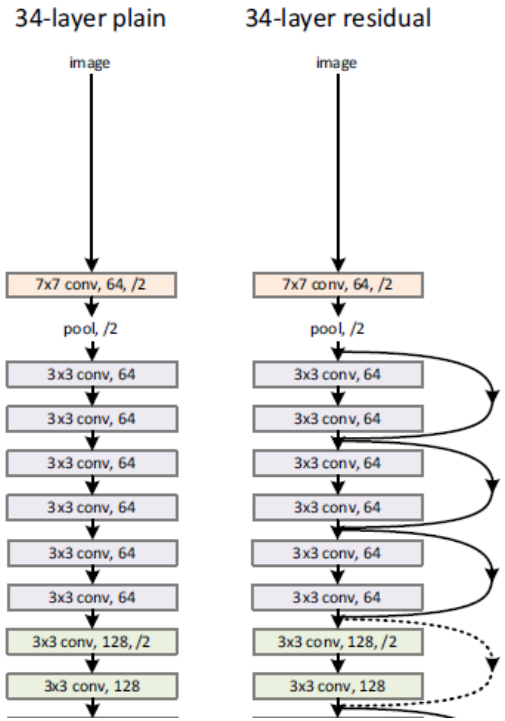
ResNet: ImageNet 2015 winner

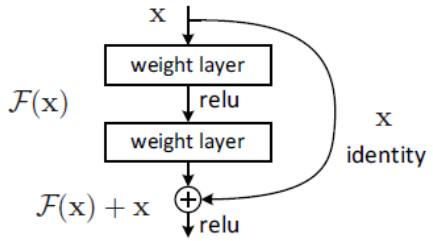


K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

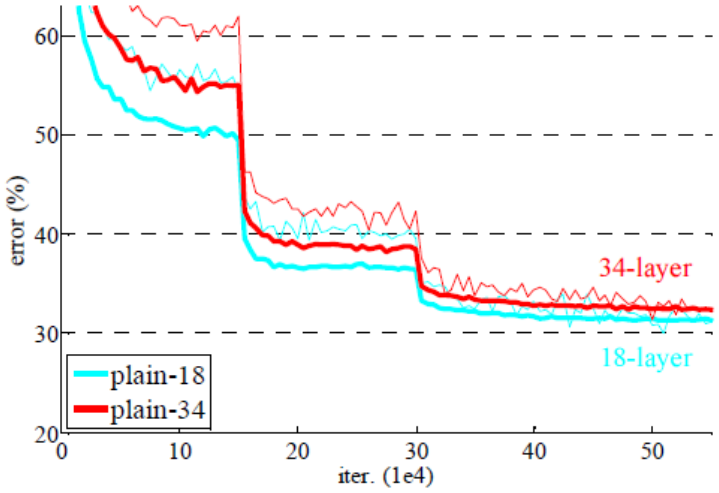
ResNet

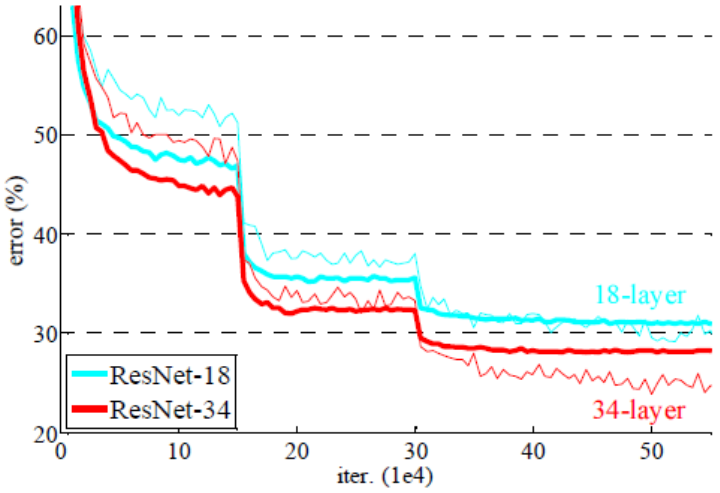
- Presented by He et al. (Microsoft), 2015. Won ILSVRC 2015 in multiple categories.
- Main idea: Residual block. Allows for extremely deep networks.
- Authors believe that it is easier to optimize the residual mapping than the original one. Furthermore, residual block can decide to “shut itself down” if needed.





Residual Block

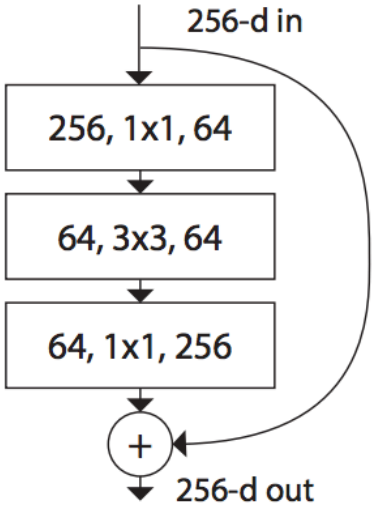




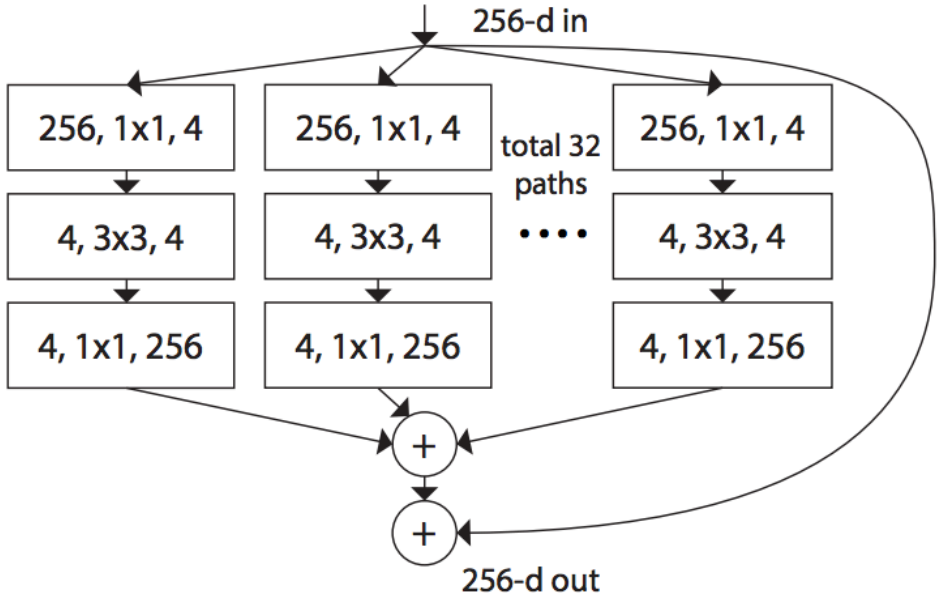
Beyond ResNet: ResNeXt

- Propose “cardinality” as a new factor in network design, apart from depth and width
- Claim that increasing cardinality is a better way to increase capacity than increasing depth or width

ResNet



ResNeXt, roughly same complexity



S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, Aggregated Residual Transformations for Deep Neural Networks, CVPR 2017

DenseNet

- Proposed by Huang et al., 2016. Radical extension of ResNet idea.
- Each block uses every previous feature map as input.
- Idea: n computation of redundant features. All the previous information is available at each point.
- Counter-intuitively, it reduces the number of parameters needed.

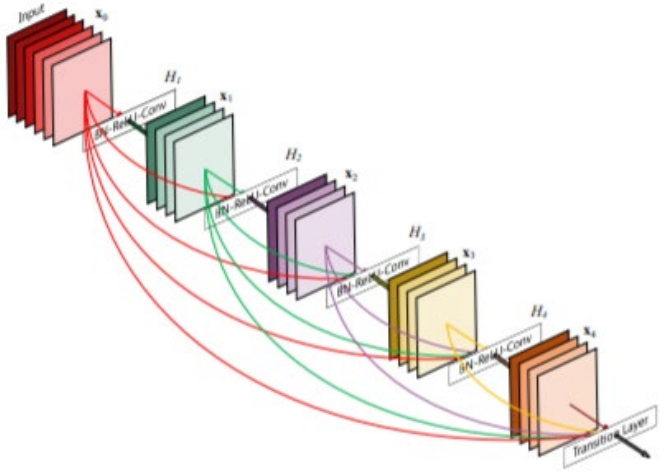


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

DenseNet

- Proposed by Huang et al., 2016. Radical extension of ResNet idea.
- Each block uses every previous feature map as input.
- Idea: n computation of redundant features. All the previous information is available at each point.
- Counter-intuitively, it reduces the number of parameters needed.

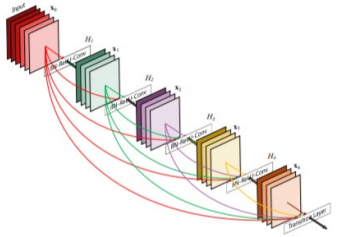
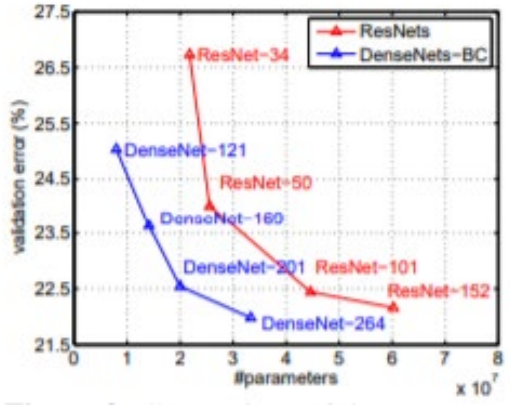
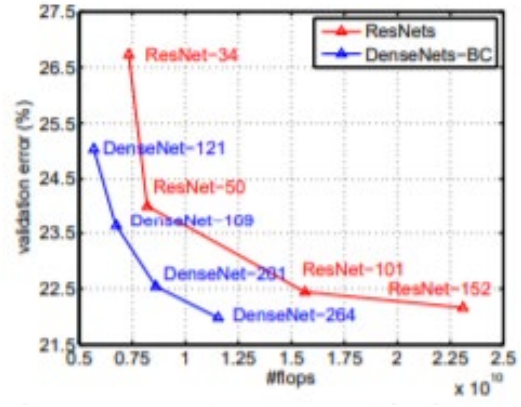
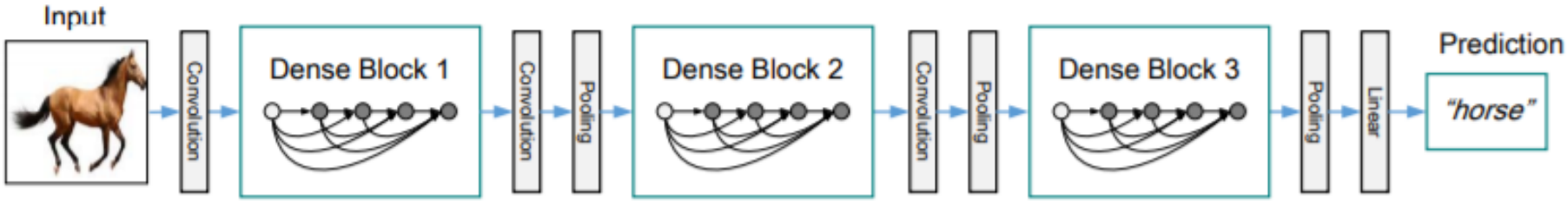


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

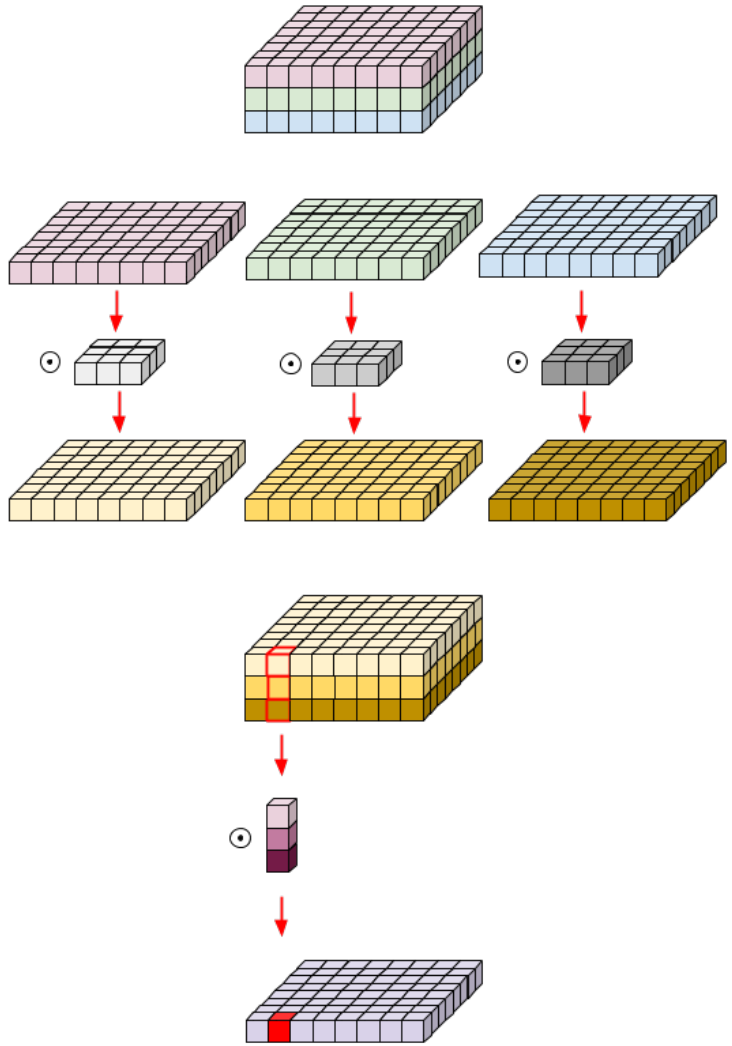






MobileNet

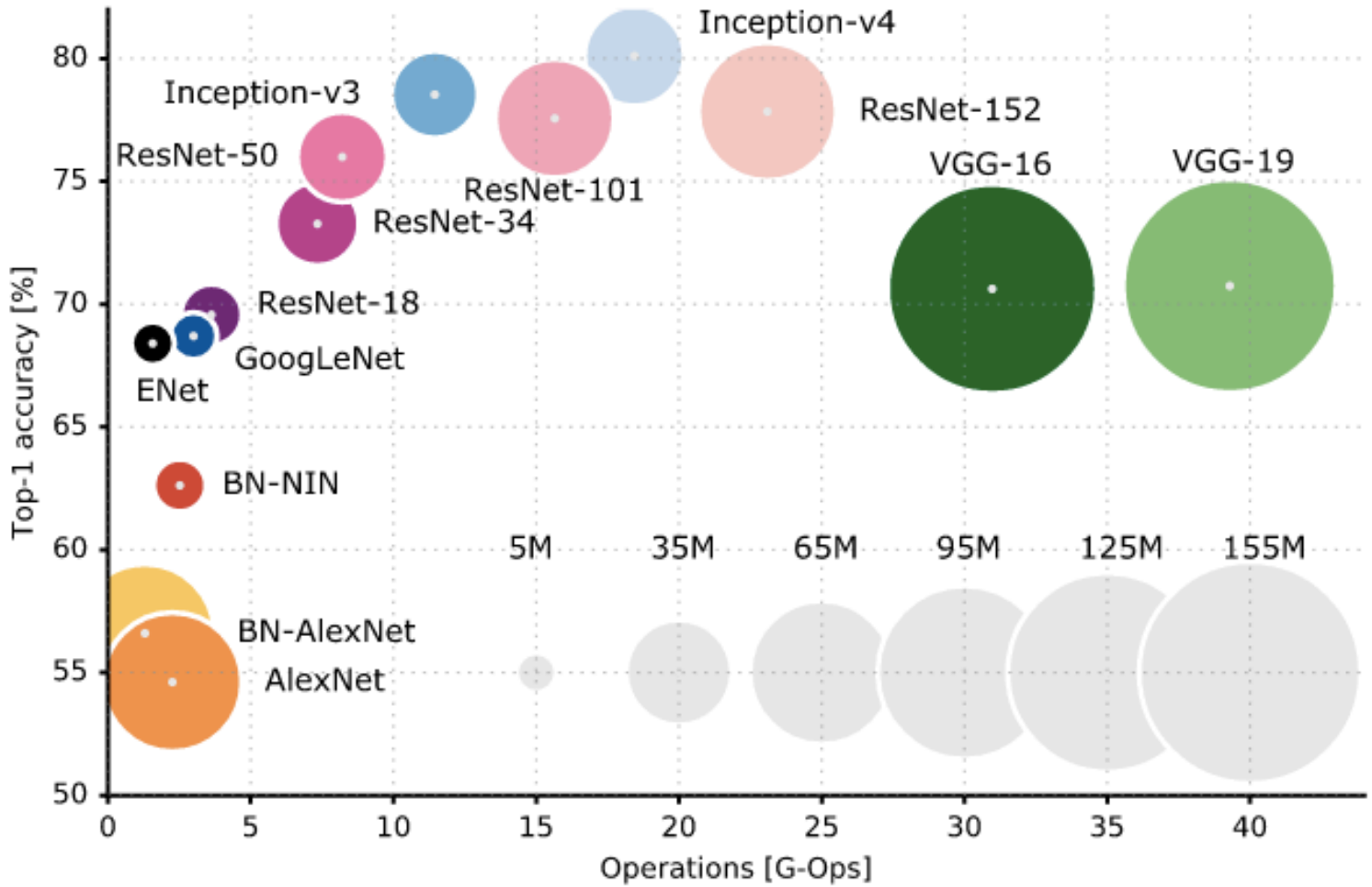
- Published by Howard et al., 2017.
- Extremely efficient network with decent accuracy.
- Main concept: depthwise-separable convolutions. Convolve each feature maps with a kernel, then use a 1x1 convolution to aggregate the result.
- This approximates vanilla convolutions without having to convolve large kernels through channels.



Beyond

- MobileNetV2 (https://arxiv.org/abs/1801.04381)
- Inception-Resnet, v1 and v2 (https://arxiv.org/abs/1602.07261)
- Wide-Resnet (https://arxiv.org/abs/1605.07146)
- Xception (https://arxiv.org/abs/1610.02357)
- ResNeXt (https://arxiv.org/pdf/1611.05431)
- ShuffleNet, v1 and v2 (https://arxiv.org/abs/1707.01083)
- Squeeze and Excitation Nets (https://arxiv.org/abs/1709.01507)

Comparing architectures



https://culurciello.github.io/tech/2016/06/04/nets.html

Techniques necessary for state-of-the-art performance

- Training tricks and details: initialization, regularization, normalization
- Training data augmentation
- Averaging classifier outputs over multiple crops/flips
- Ensembles of networks

Neural network training: Beyond the basics

Learning rate decay

- Decay formulas
  - Exponential: $\eta_t = \eta_0 e^{-kt}$, where η_0 and k are hyperparameters, t is the iteration or epoch number
  - Inverse: $\eta_t = \eta_0/(1 + kt)$
  - Inverse sqrt: $\eta_t = \eta_0/\sqrt{t}$
  - Linear: $\eta_t = \eta_0(1 - t/T)$, where *T* is the total number of epochs
  - Cosine: $\eta_t = \frac{1}{2}\eta_0(1 + \cos(t\pi/T))$

A typical phenomenon

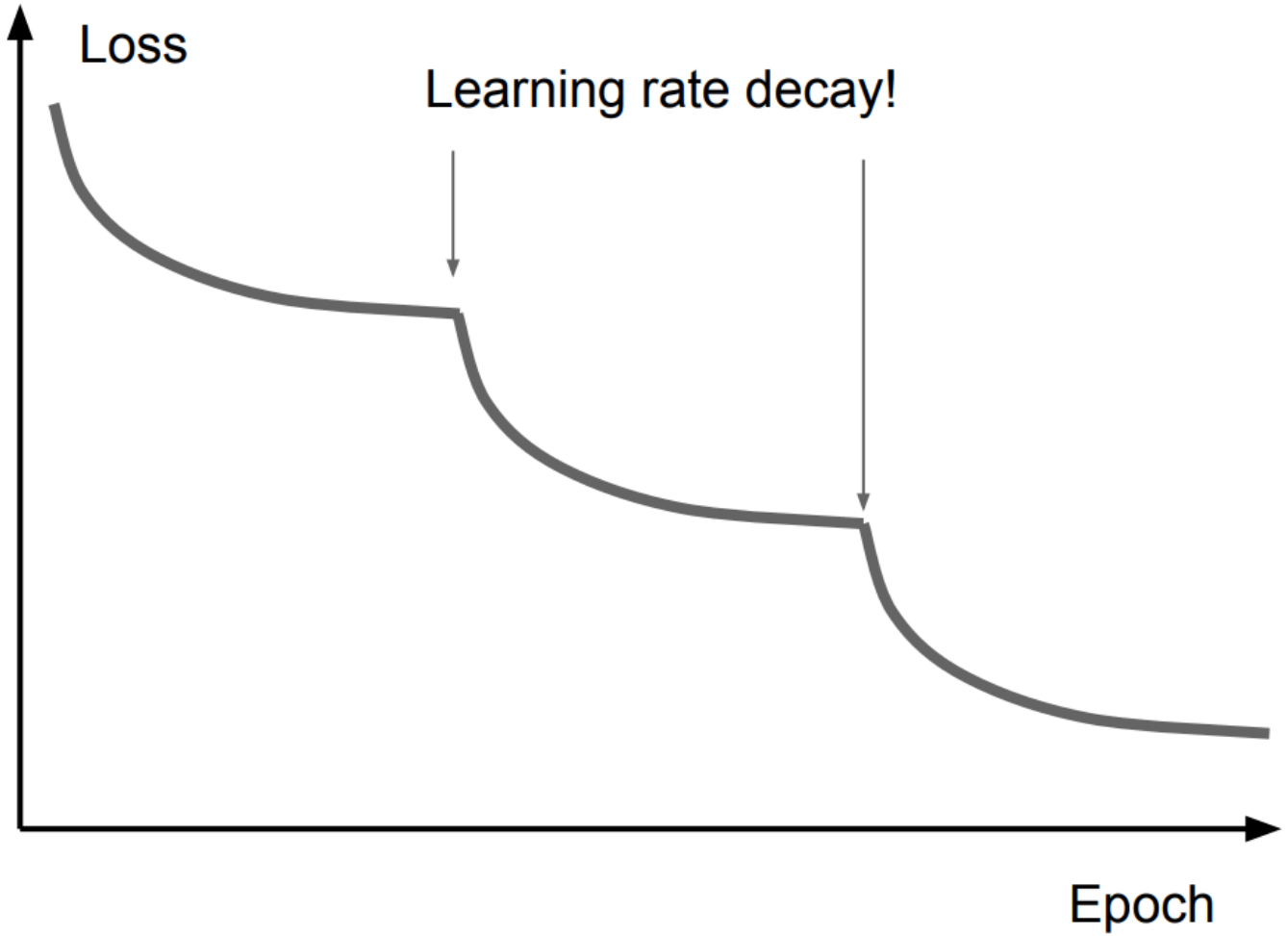
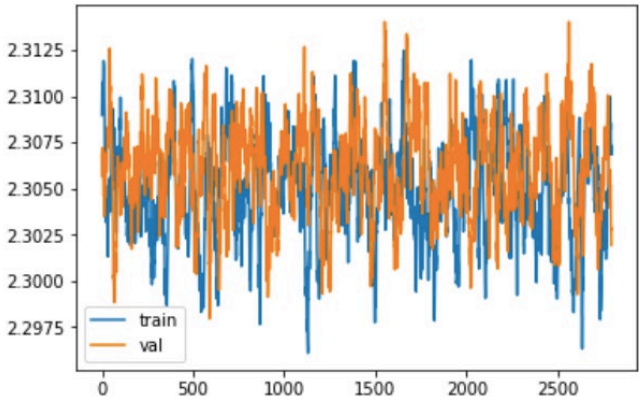


Image source: Stanford CS231n

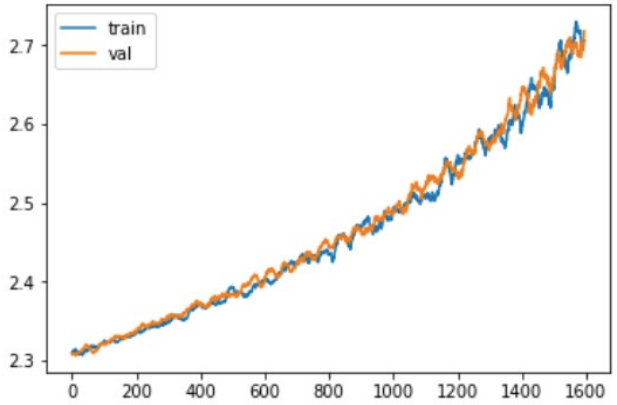
Learning rate decay

- Decay formulas
- Most common in practice:
  - **Step decay:** reduce rate by a constant factor every few epochs, e.g., by 0.5 every 5 epochs, 0.1 every 20 epochs
  - **Manual:** watch validation error and reduce learning rate whenever it stops improving
    - "Patience" hyperparameter: number of epochs without improvement before reducing learning rate
- **Warmup:** train with a low learning rate for a first few epochs, or linearly increase learning rate before transitioning to normal decay schedule (Goyal et al., 2018)

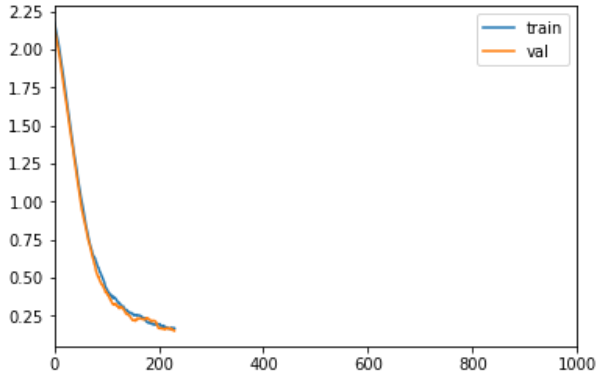
Diagnosing learning curves: Obvious problems



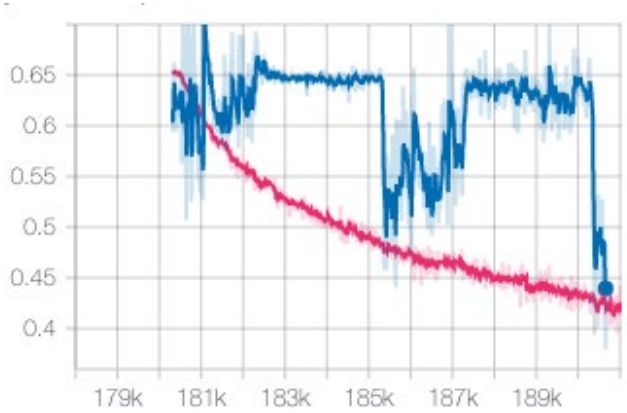
Not training
Bug in update calculation?



Error increasing
Bug in update calculation?



Get NaNs in the loss after a number of iterations:
Numerical instability



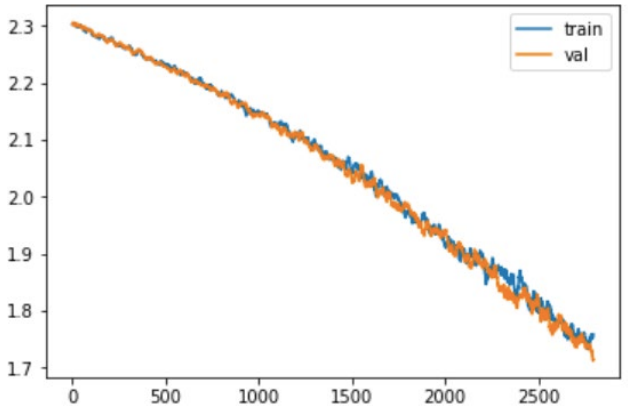
Shuffling off

Shuffling on

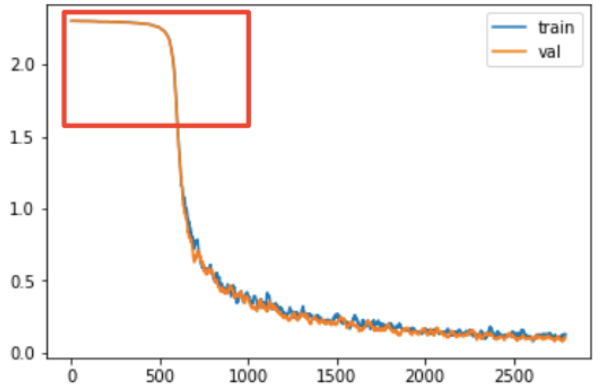
Weird cyclical patterns in loss:
Data not shuffled

Source: Stanford CS231n

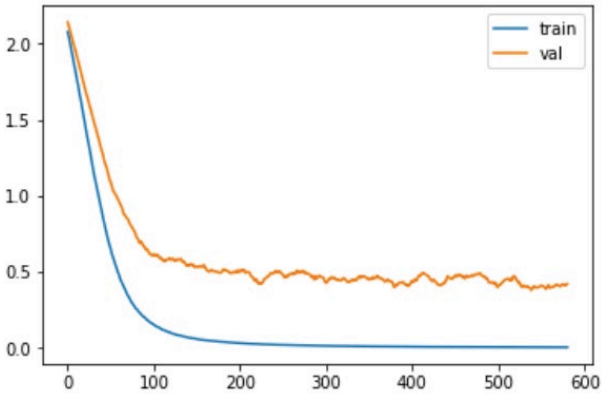
Diagnosing learning curves: Subtler behaviors



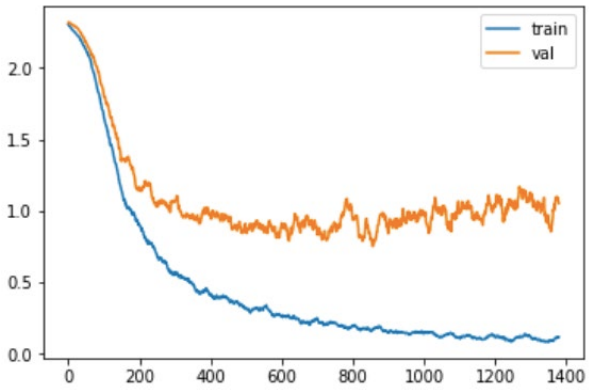
Not converged yet
Keep training, possibly increase learning rate



Slow start
Bad initialization?



Possible overfitting



Definite overfitting

Source: Stanford CS231n

When to stop training?

- Monitor validation error to decide when to stop
  - "Patience" hyperparameter: number of epochs without improvement before stopping
  - *Early stopping* can be viewed as a kind of regularization

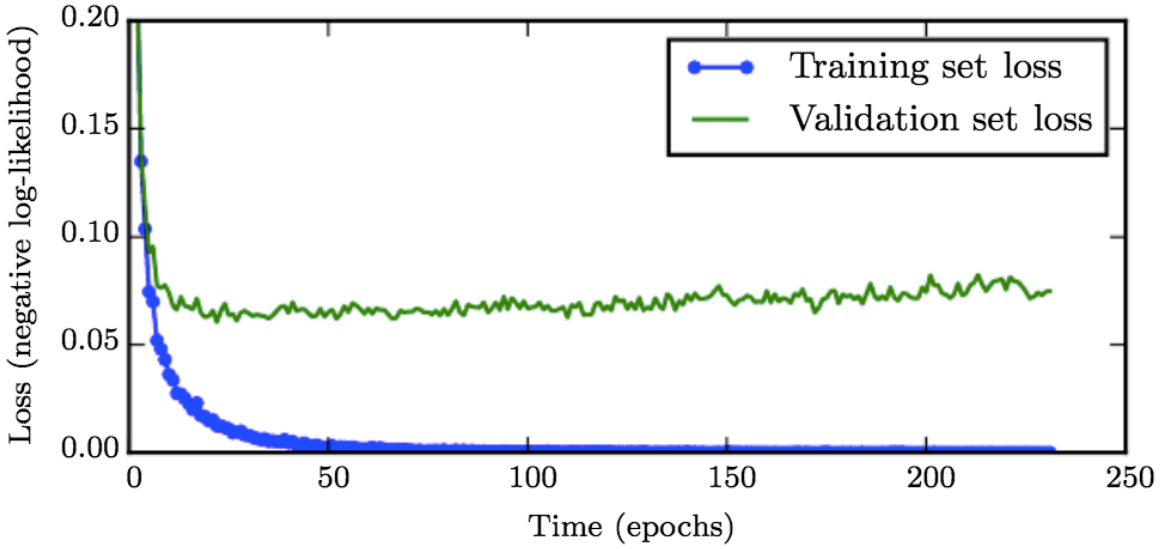
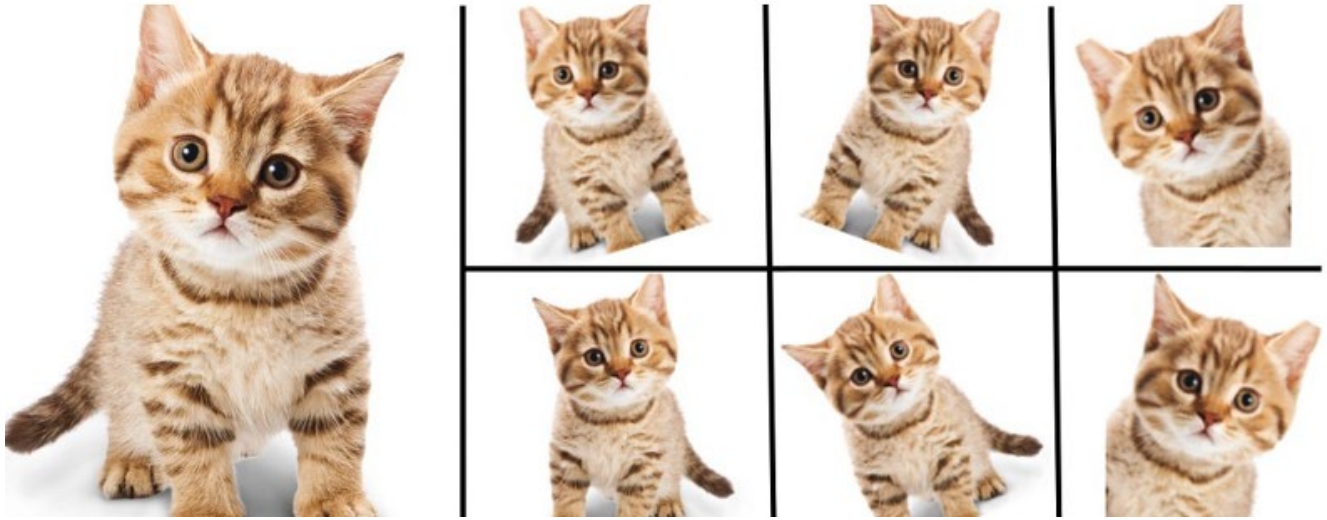


Figure from Deep Learning Book

Data augmentation

Introduce transformations not adequately sampled in the training data

- Geometric: flipping, rotation, shearing, multiple crops



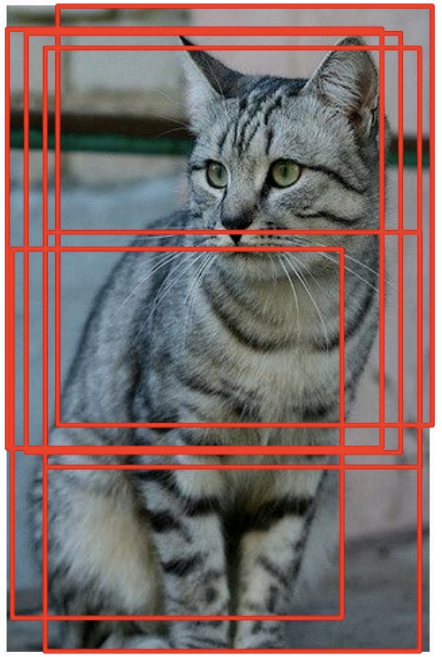
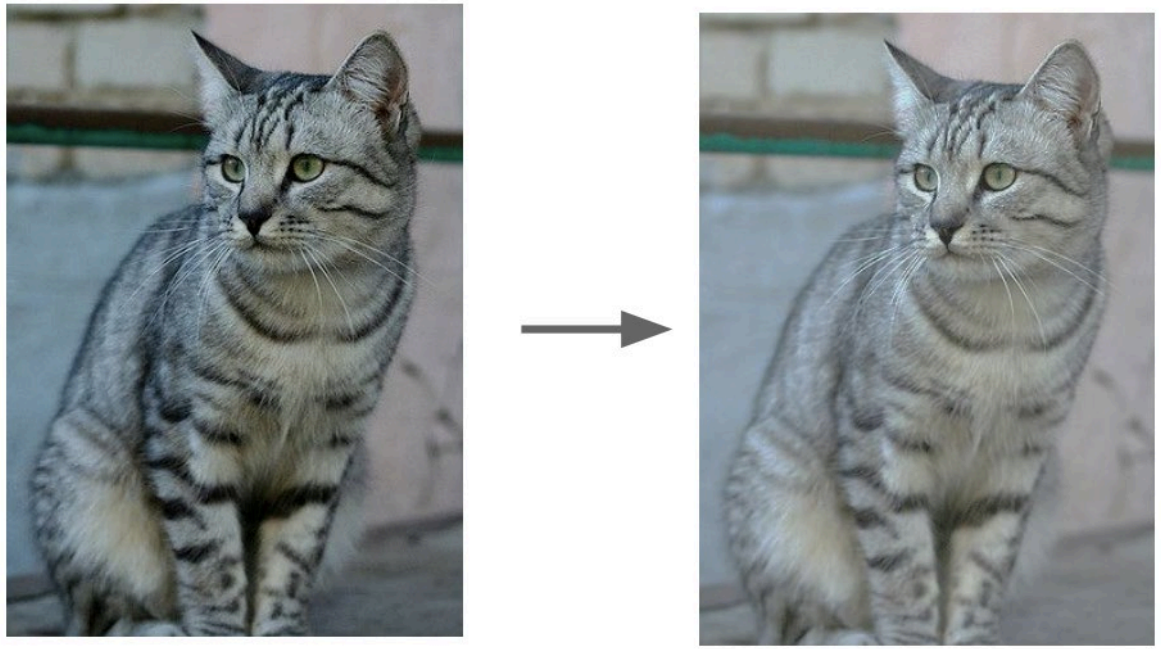


Image source

Data augmentation

Introduce transformations not adequately sampled in the training data

- Geometric: flipping, rotation, shearing, multiple crops
- Photometric: color transformations



Data augmentation

Introduce transformations not adequately sampled in the training data

- Geometric: flipping, rotation, shearing, multiple crops
- Photometric: color transformations
- Other: add noise, compression artifacts, lens distortions, etc.

Data augmentation

Introduce transformations not adequately sampled in the training data

- Limited only by your imagination and time/memory constraints!
- Avoid introducing artifacts

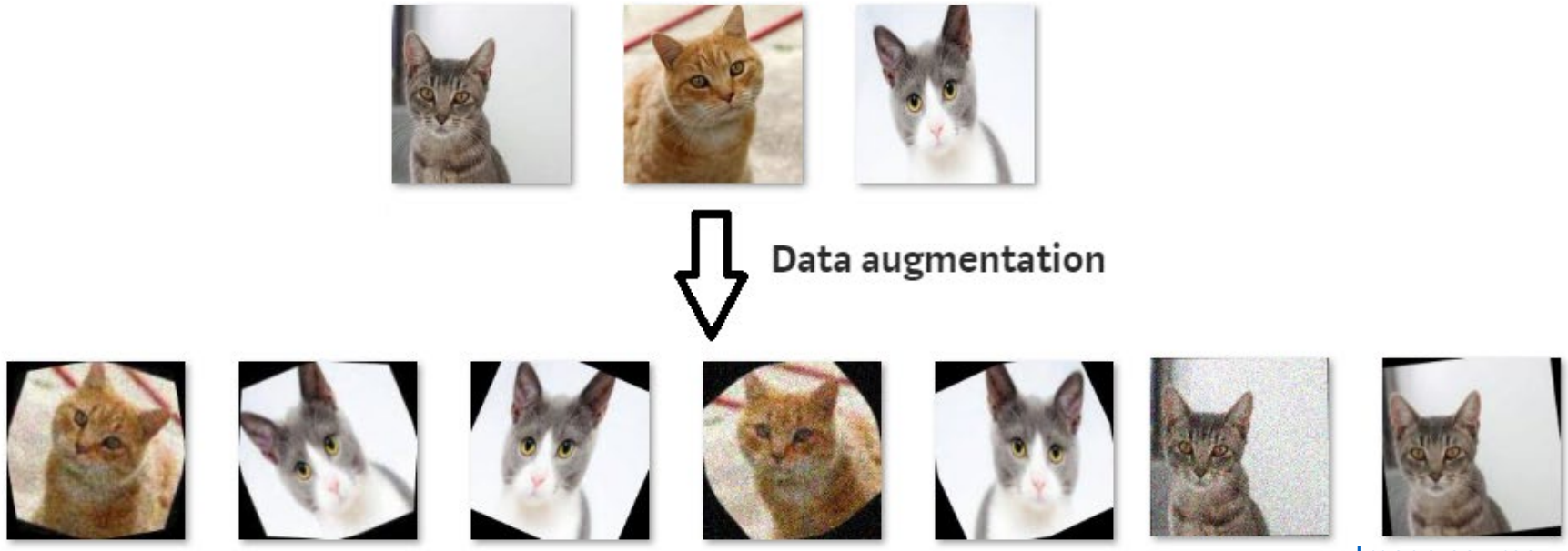


Image source

Data preprocessing

- Zero centering
  - Subtract *mean image* – all input images need to have the same resolution
  - Subtract *per-channel means* – images don't need to have the same resolution
- Optional: rescaling – divide each value by (per-pixel or per-channel) standard deviation

- Be sure to apply the same transformation at training and test time!
  - Save training set statistics and apply to test data

Weight initialization

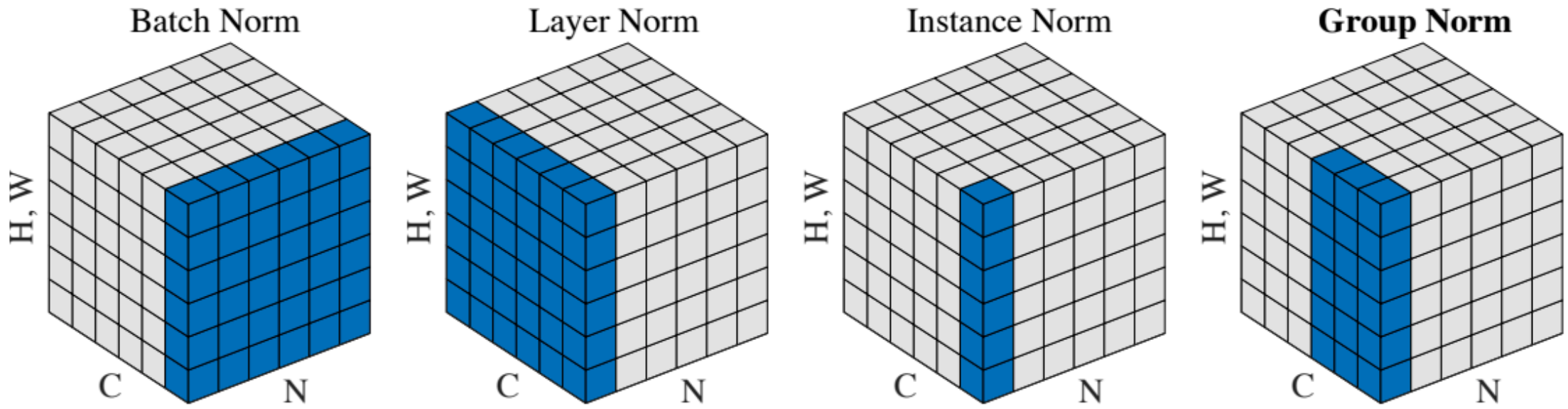
What’s wrong with initializing all weights to the same number (e.g., zero)?

Batch normalization

- Benefits
  - Prevents exploding and vanishing gradients
  - Keeps most activations away from saturation regions of non-linearities
  - Accelerates convergence of training
  - Makes training more robust w.r.t. hyperparameter choice, initialization
- Pitfalls
  - Behavior depends on composition of mini-batches, can lead to hard-to-catch bugs if there is a mismatch between training and test regime (example)
  - Doesn’t work well for small mini-batch sizes
  - Cannot be used in recurrent models

Other types of normalization

- Layer normalization (Ba et al., 2016)
- Instance normalization (Ulyanov et al., 2017)
- Group normalization (Wu and He, 2018)
- Weight normalization (Salimans et al., 2016)



Y. Wu and K. He, Group Normalization, ECCV 2018

Regularization

- Techniques for controlling the capacity of a neural network to prevent overfitting – short of explicit reduction of the number of parameters
- Recall: classic regularization: L1, L2

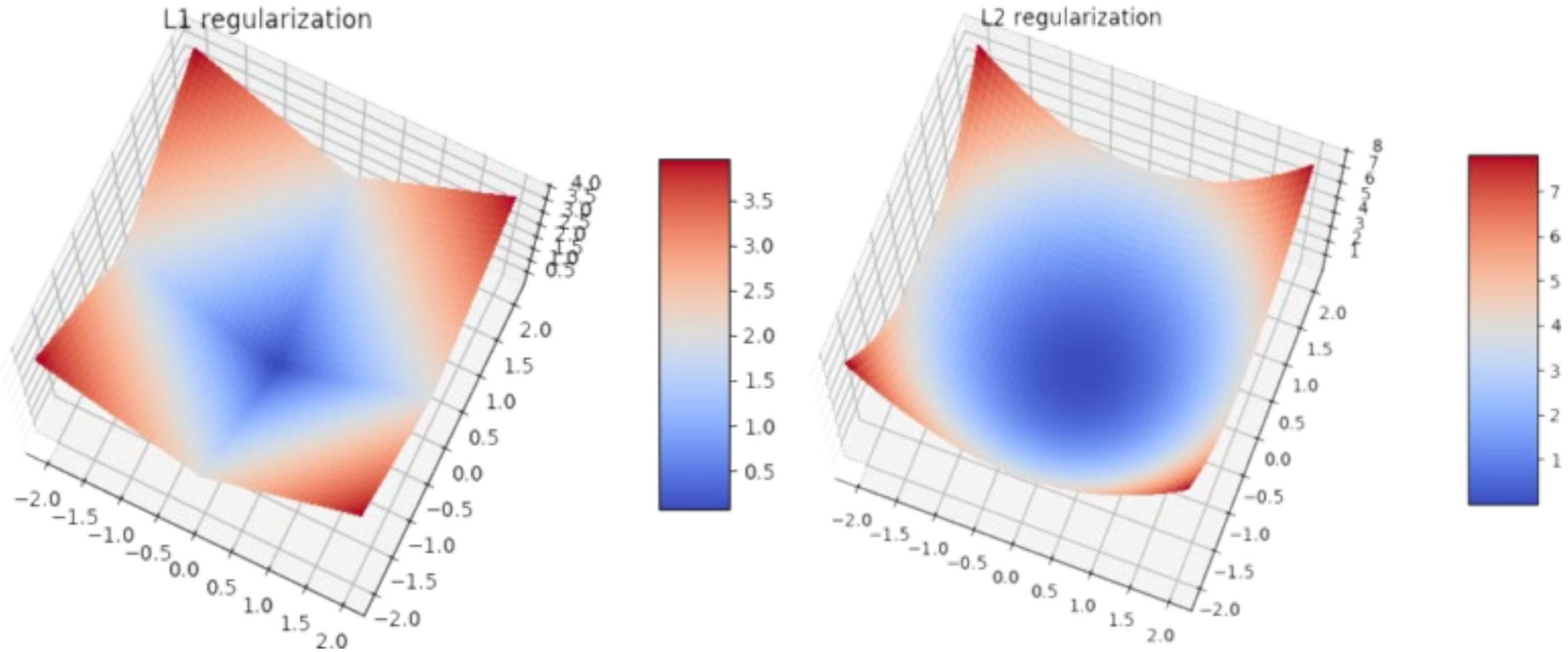


Image source

Weight decay

- Generic optimization step:

$$L(w) = L_{\text{data}}(w) + L_{\text{reg}}(w)$$
$$g_t = \nabla L(w_t)$$
$$s_t = \text{optimizer}(g_t)$$
$$w_{t+1} = w_t - \eta s_t$$

- SGD with L2 regularization:

$$L(w) = L_{\text{data}}(w) + \frac{\lambda}{2}\|w\|^2$$
$$g_t = \nabla L_{\text{data}}(w_t) + \lambda w$$
$$w_{t+1} = w_t - \eta g_t$$
$$= (1 - \eta\lambda) w_t - \eta \nabla L_{\text{data}}(w_t)$$

- Optimization with weight decay:

$$L(w) = L_{\text{data}}(w)$$
$$g_t = \nabla L(w_t)$$
$$s_t = \text{optimizer}(g_t)$$
$$w_{t+1} = (1 - \eta\lambda) w_t - \eta s_t$$

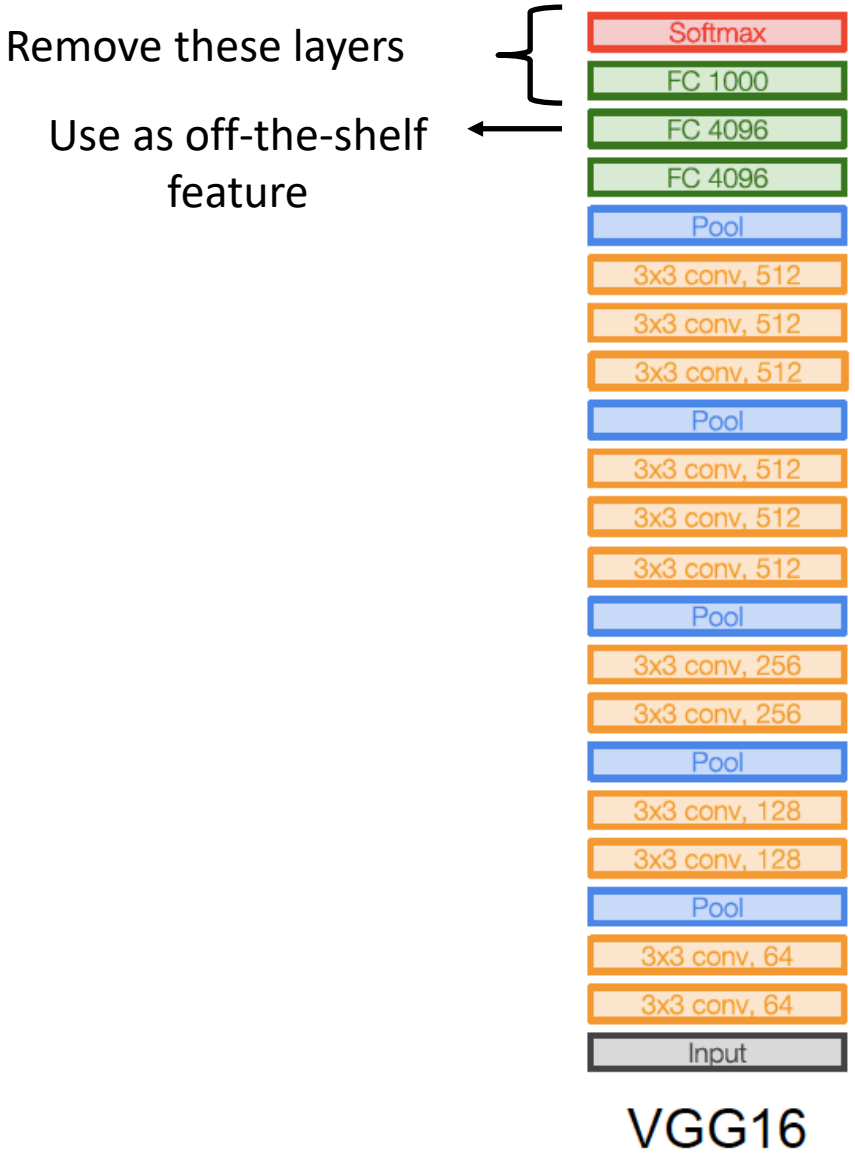
I. Loshchilov and F. Hutter, Decoupled Weight Decay Regularization, ICLR 2019

Adapted from J. Johnson

Other types of regularization

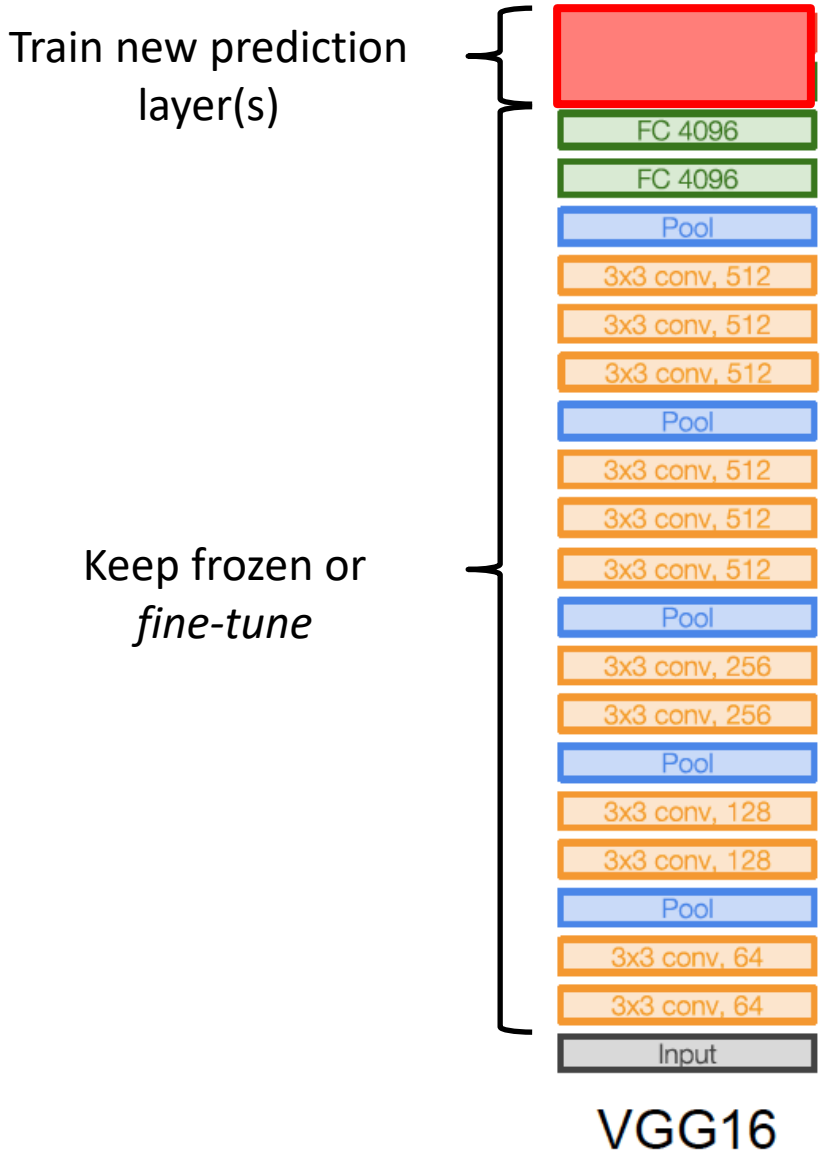
- Adding noise to the inputs
  - Recall motivation of max margin criterion
  - In simple scenario (linear model, quadratic loss, Gaussian noise), this is equivalent to weight decay
  - Data augmentation is a more general form of this
- Adding noise to the weights
- Label smoothing
  - Recall: when using softmax loss, replace hard 1 and 0 prediction targets with "soft" targets of $1 - \epsilon$ and $\frac{\epsilon}{C-1}$

How to use a pre-trained network for a new task?



- Strategy 1: Use as feature extractor

How to use a pre-trained network for a new task?



- Strategy 2: Transfer learning

Distillation

1. Train a *teacher* network on initial labeled dataset
2. Save the softmax outputs the teacher network for each training example
3. Train a *student* network with cross-entropy loss using the softmax outputs of the teacher network as targets

- Many uses
  - Compressing a larger model (or even an ensemble) into a smaller one
  - "Copying" a black-box teacher model (e.g., network you can only access via an API)
  - Extending a network to additional tasks without "forgetting" old tasks (Li and Hoiem, 2017)

G. Hinton, O. Vinyals, J. Dean. Distilling the knowledge in a neural network. arXiv 2015

Some take-aways

- Training neural networks is still a black art
- Process requires close “babysitting”
- For many techniques, the reasons why, when, and whether they work are in active dispute – read everything but don’t trust anything
- It all comes down to (principled) trial and error
- Further reading: A. Karpathy, [A recipe for training neural networks]