Deep Learning for Vision

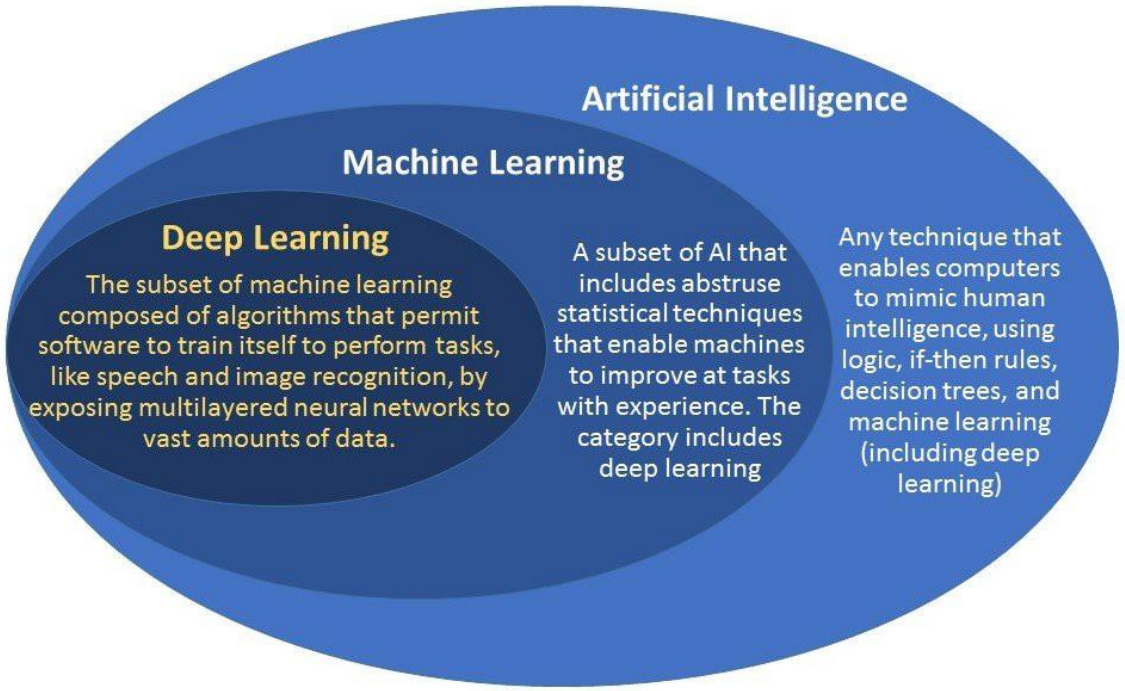
Deep Learning for Vision

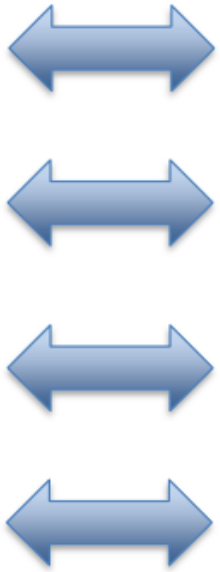
Computer Vision

Generative AI

Computer Vision

AI and Computer Vision

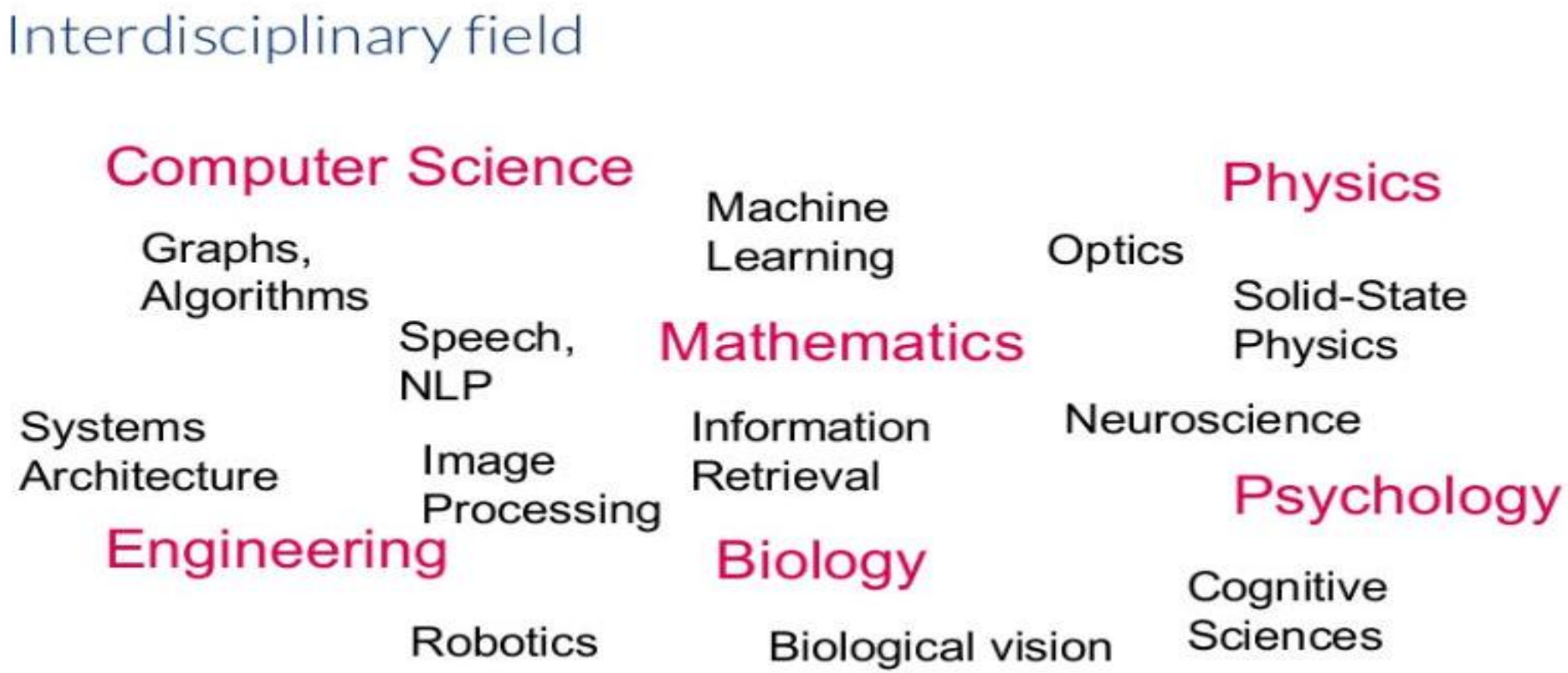




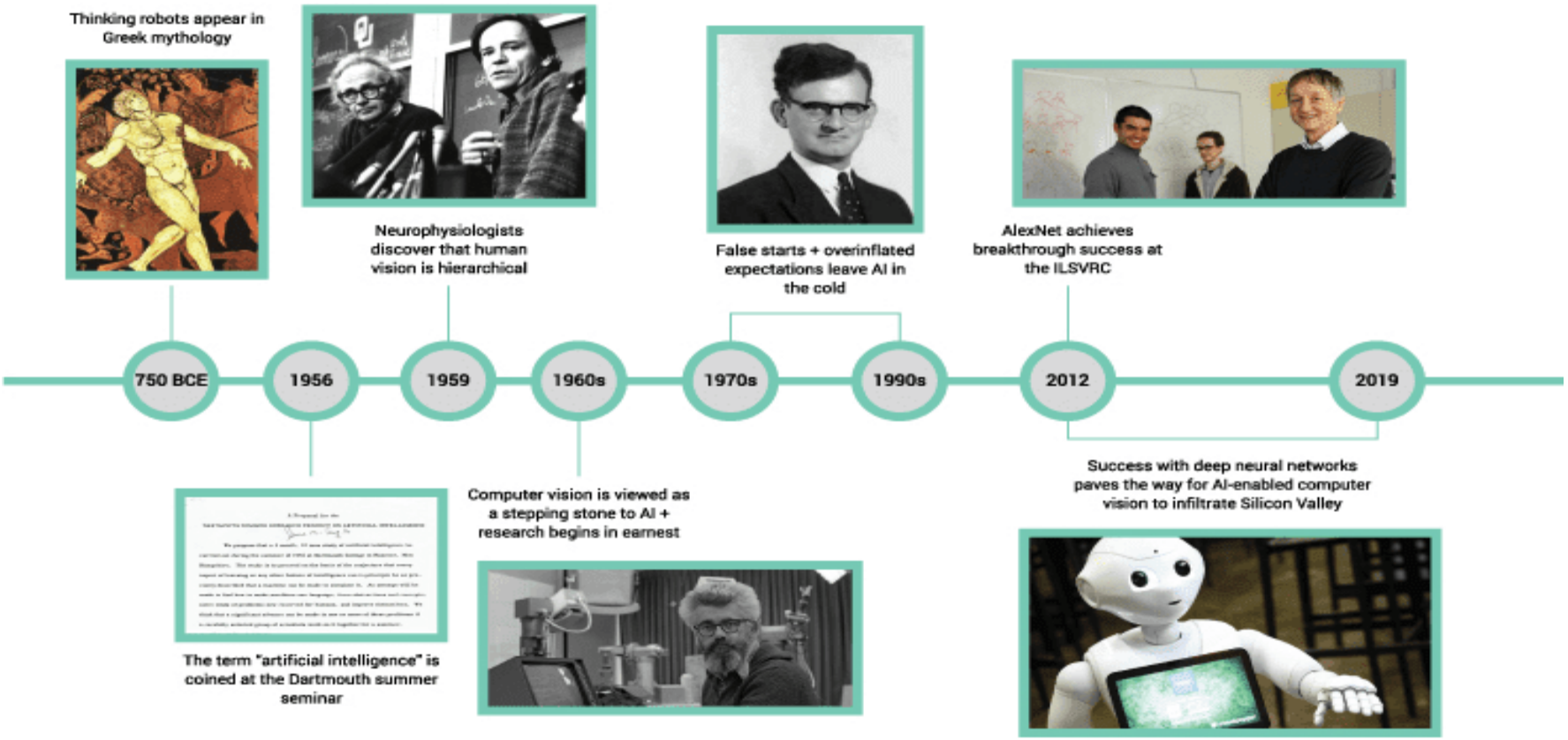
Computer Vision

- Object detection
- Object classification
- Scene understanding
- Semantic scene segmentation
- 3D reconstruction
- Object tracking
- Human pose estimation
- Activity recognition
- VQA
- ....

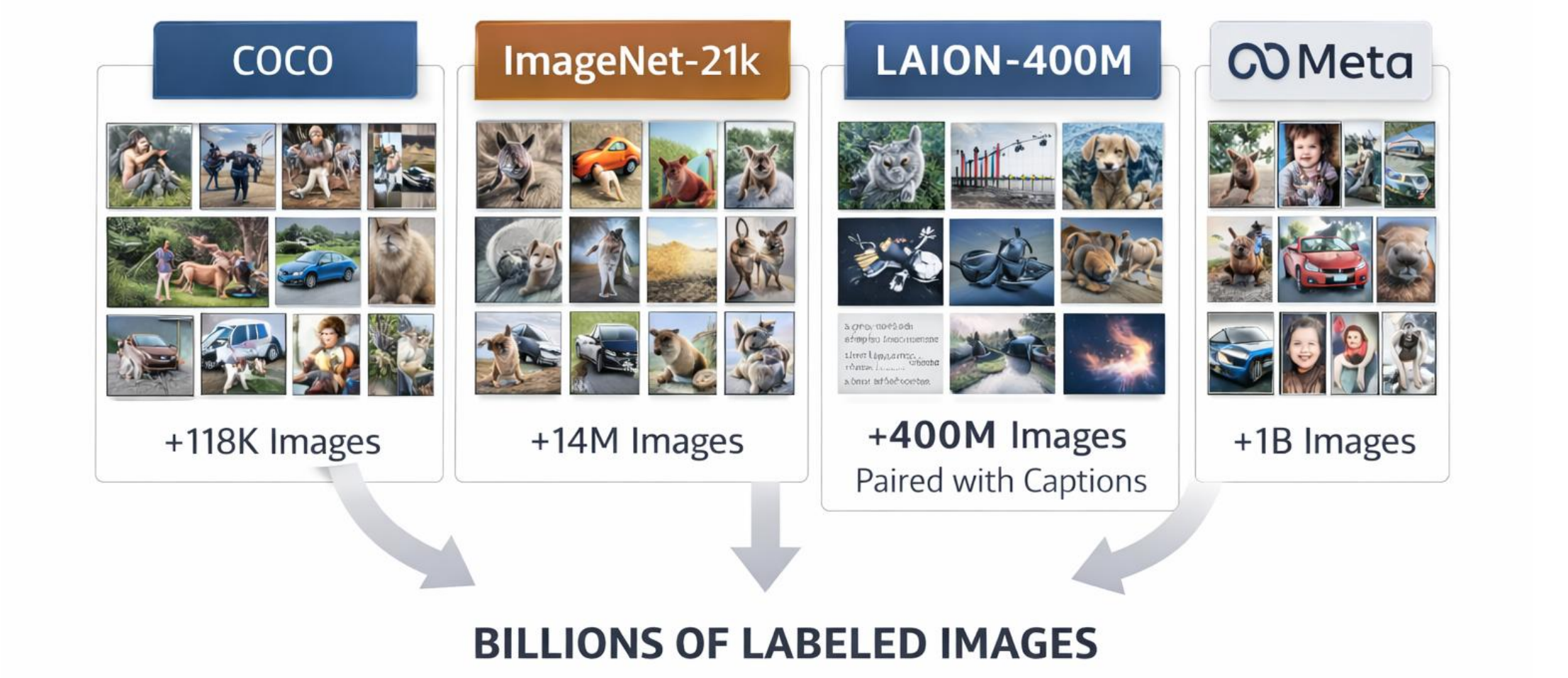
Computer Vision



Computer Vision History



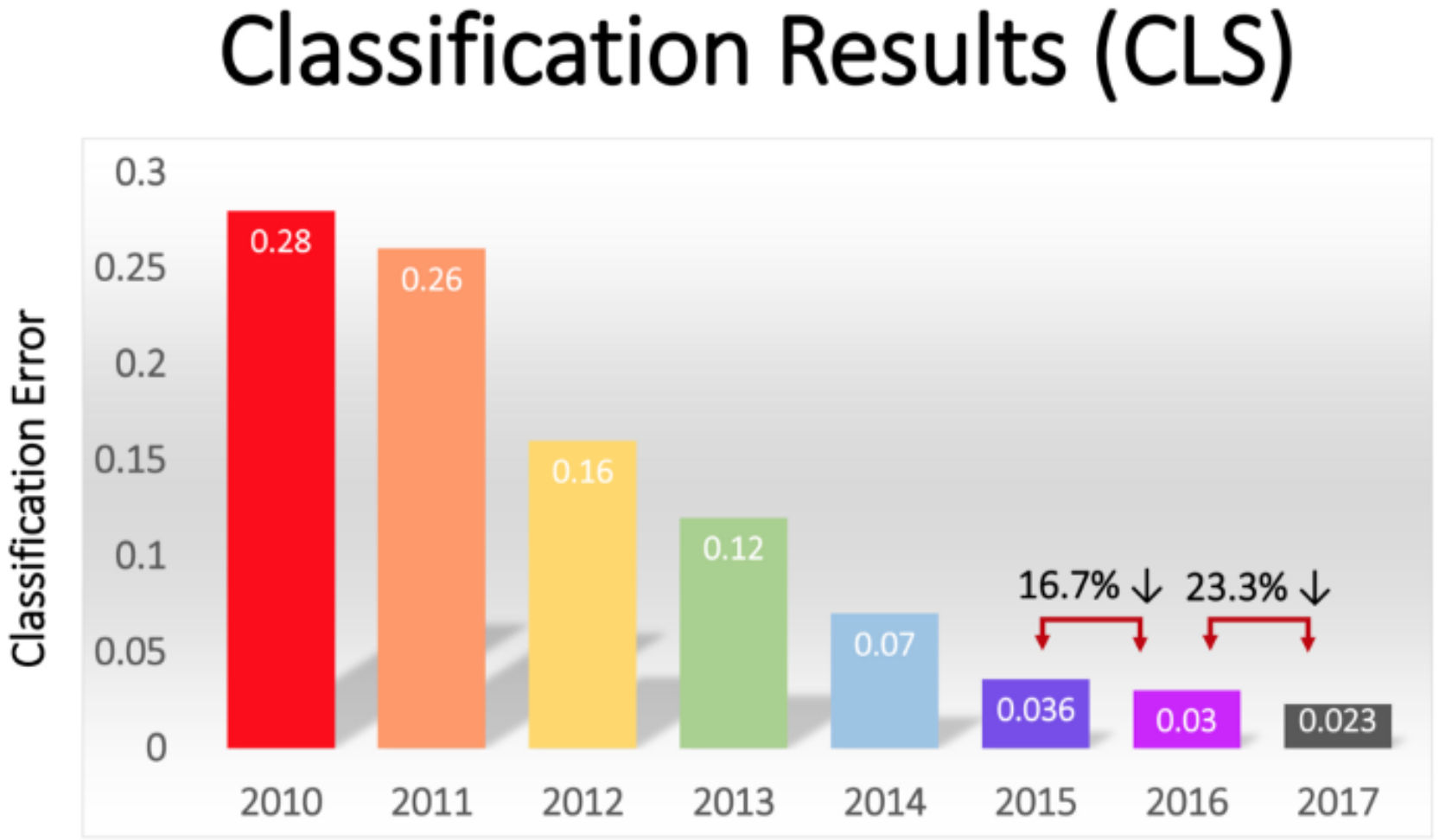
Deep Vision Models Require Large Datasets



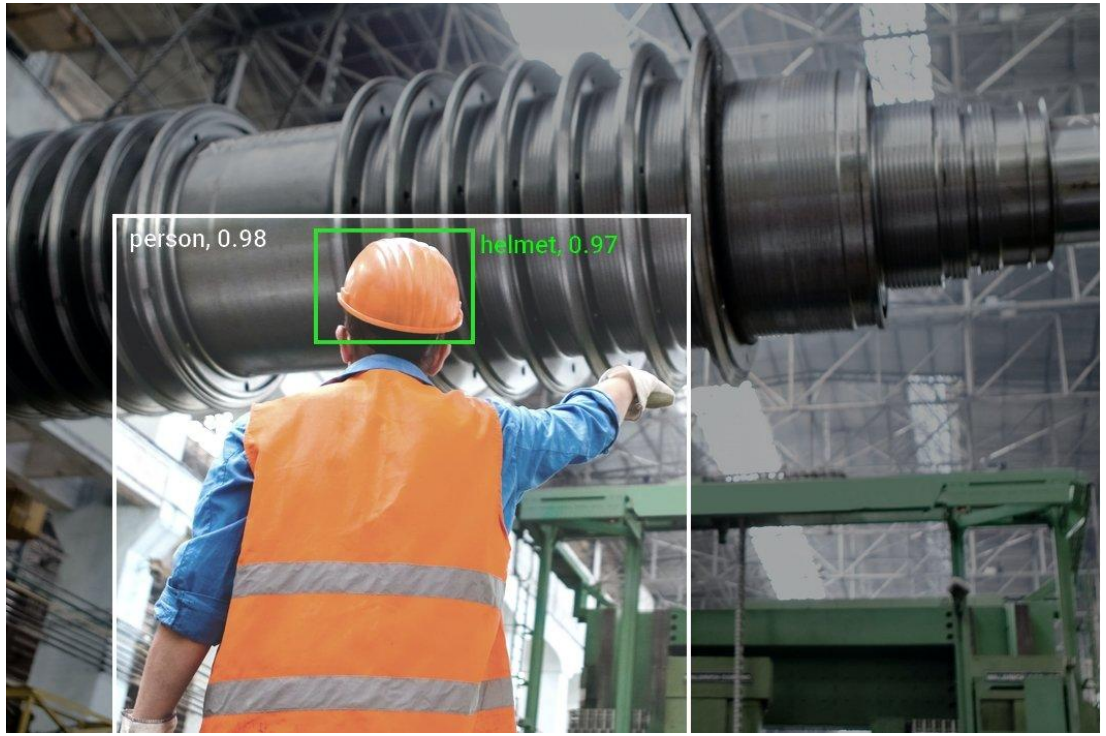
Detailed Labels

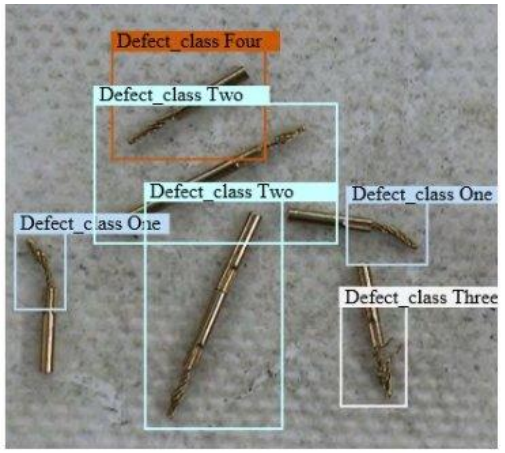
The Image Classification Challenge:
1,000 object classes
1,431,167 images

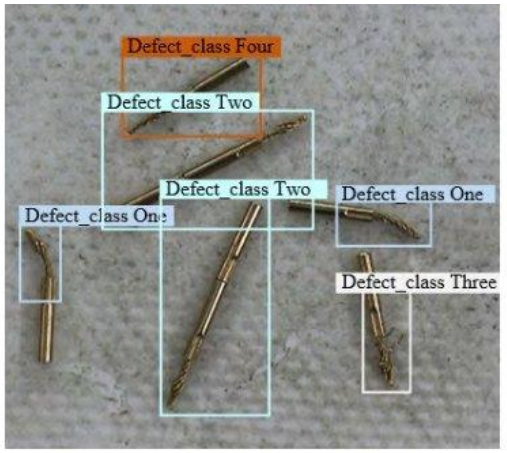
Imagenet Large Scale Visual Recognition Challenge

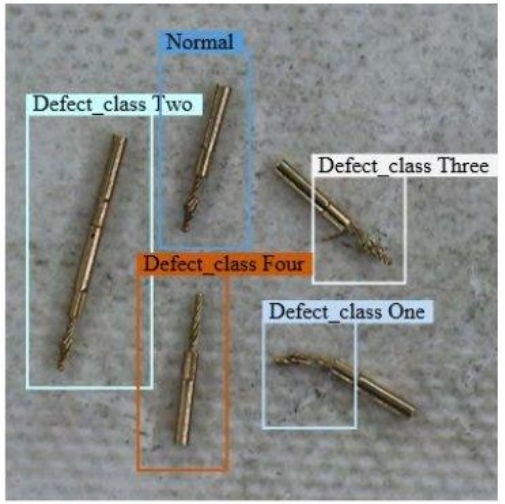


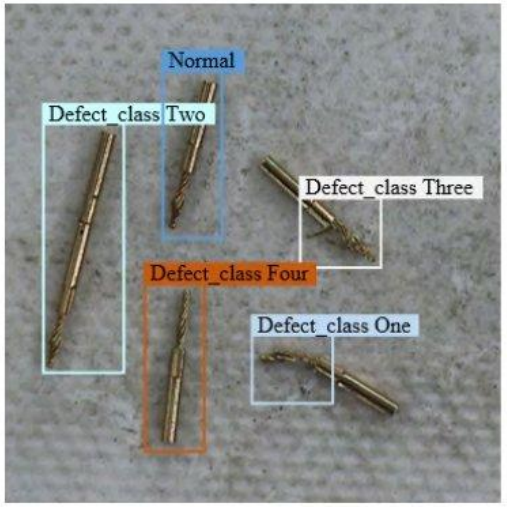
Applications



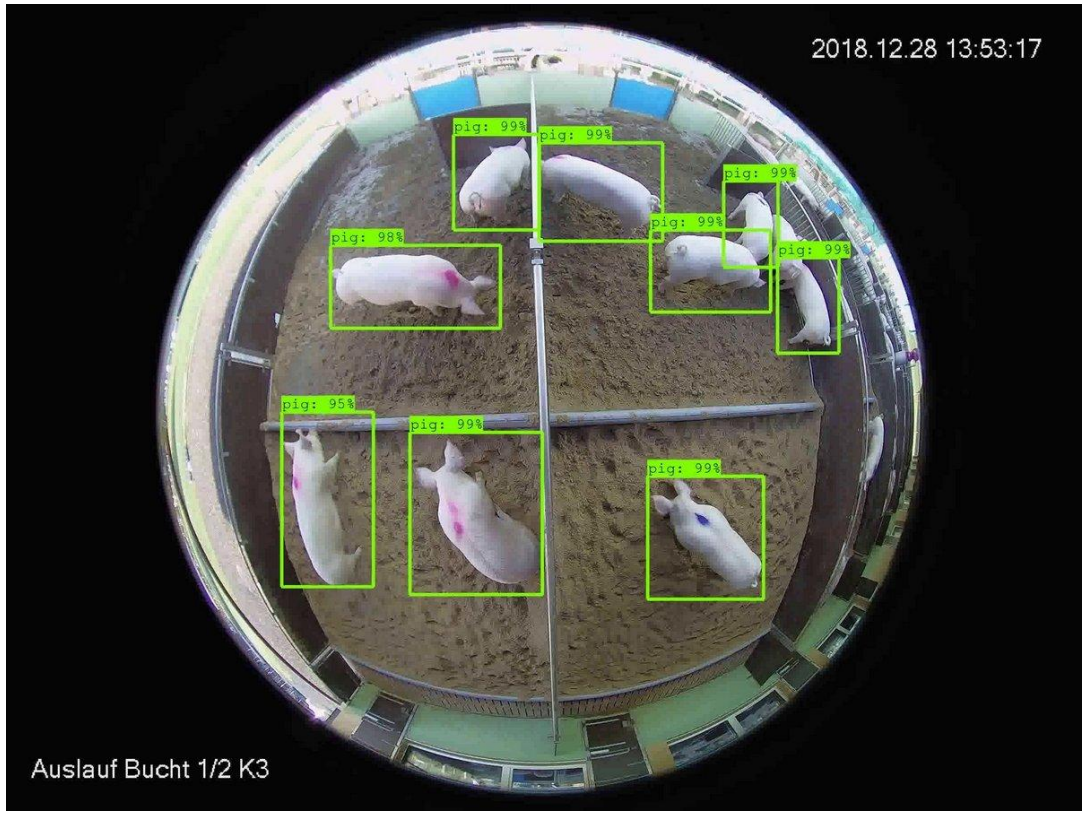


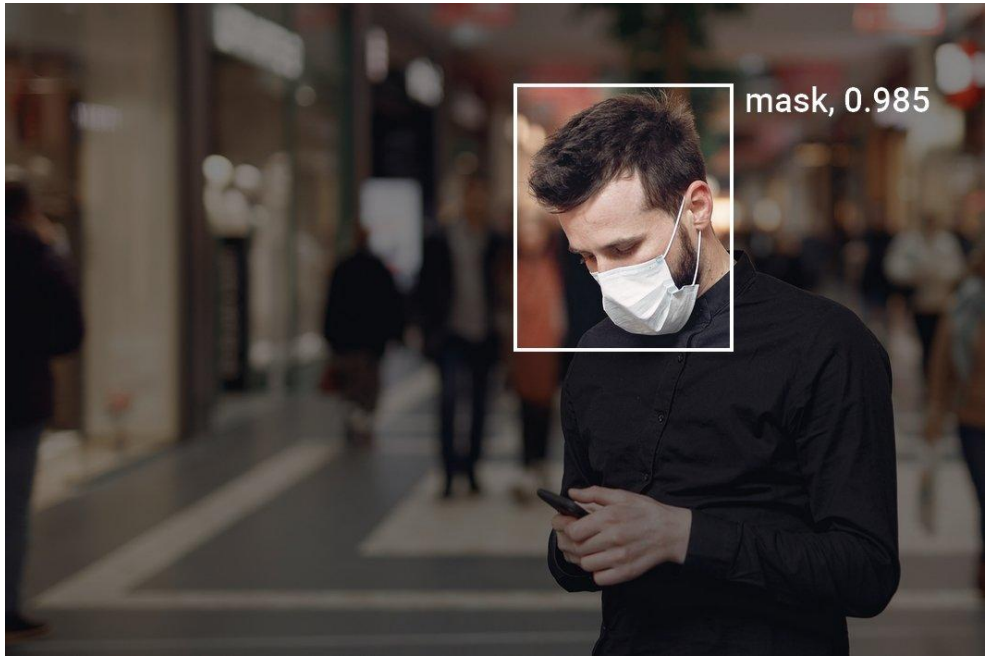






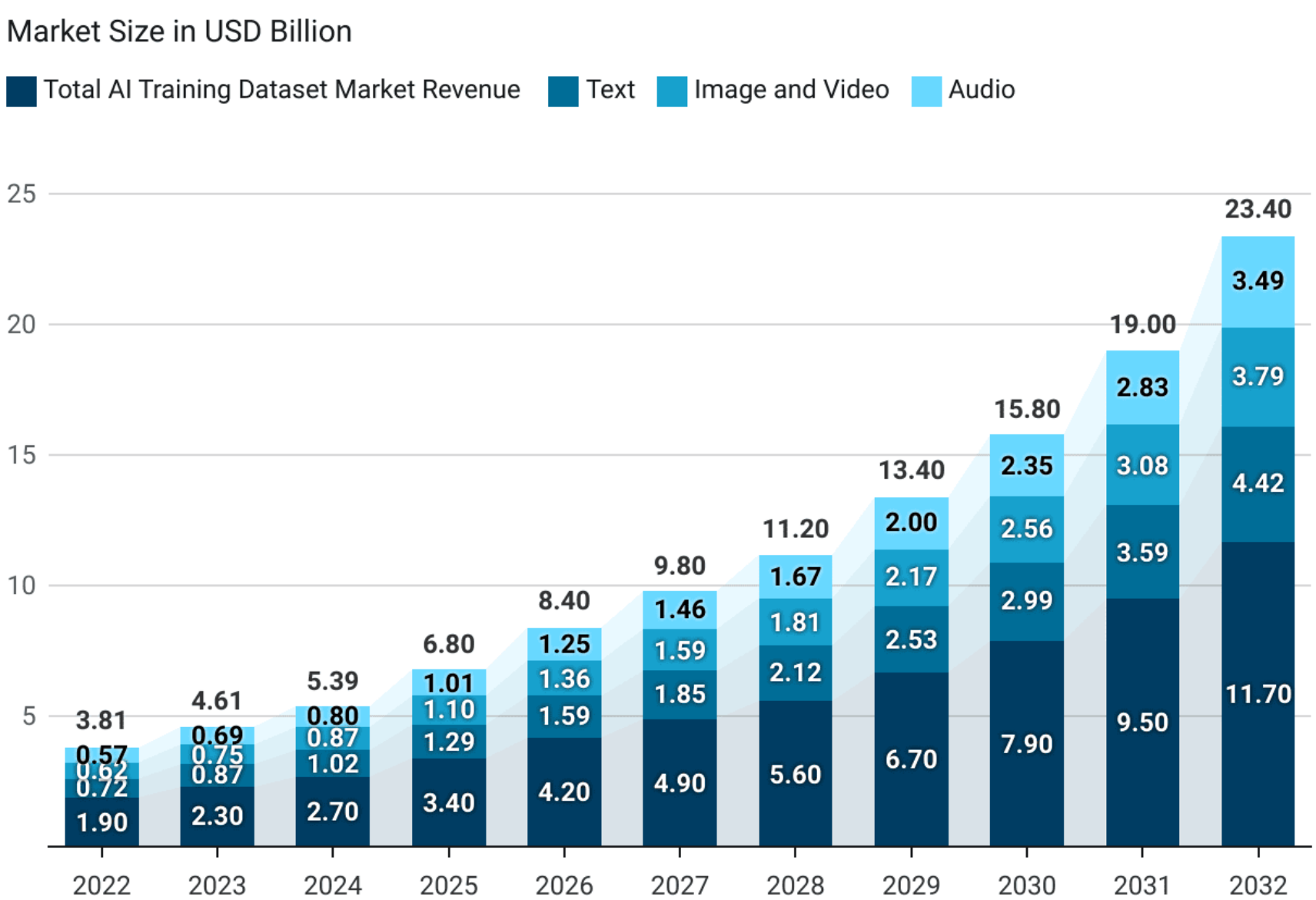
Applications





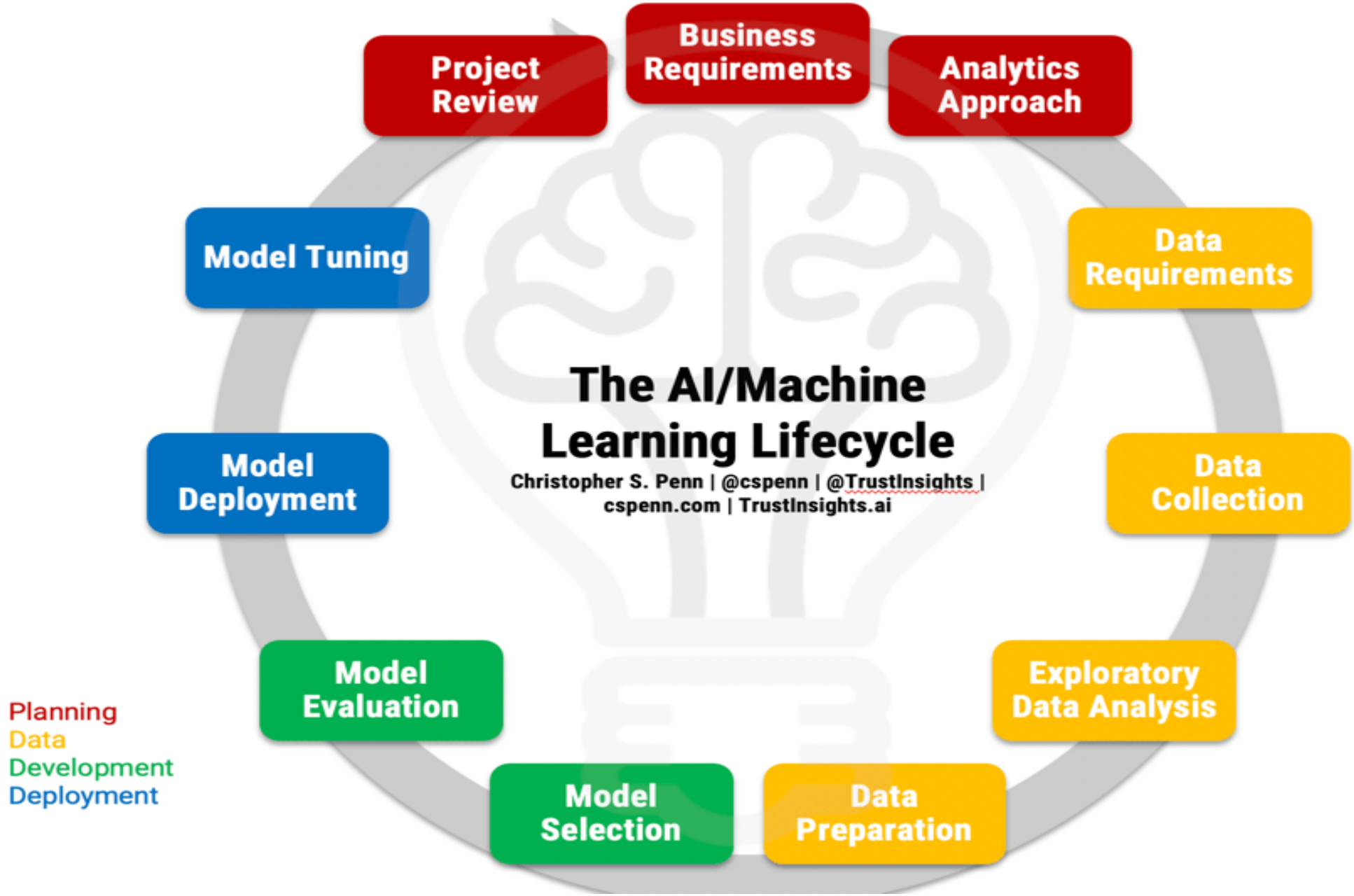
Applications

Global AI Training Dataset Market Size - By Type

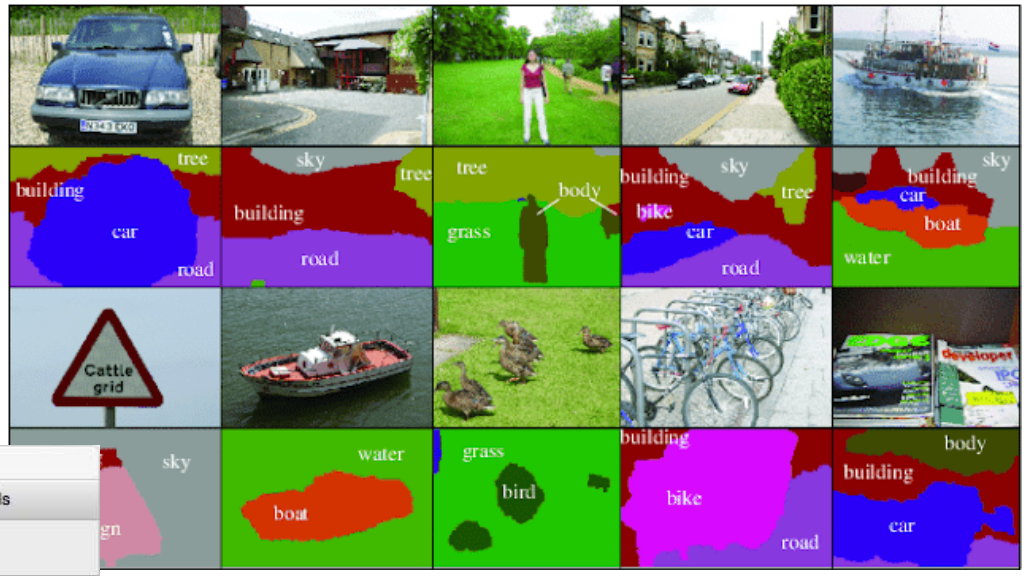


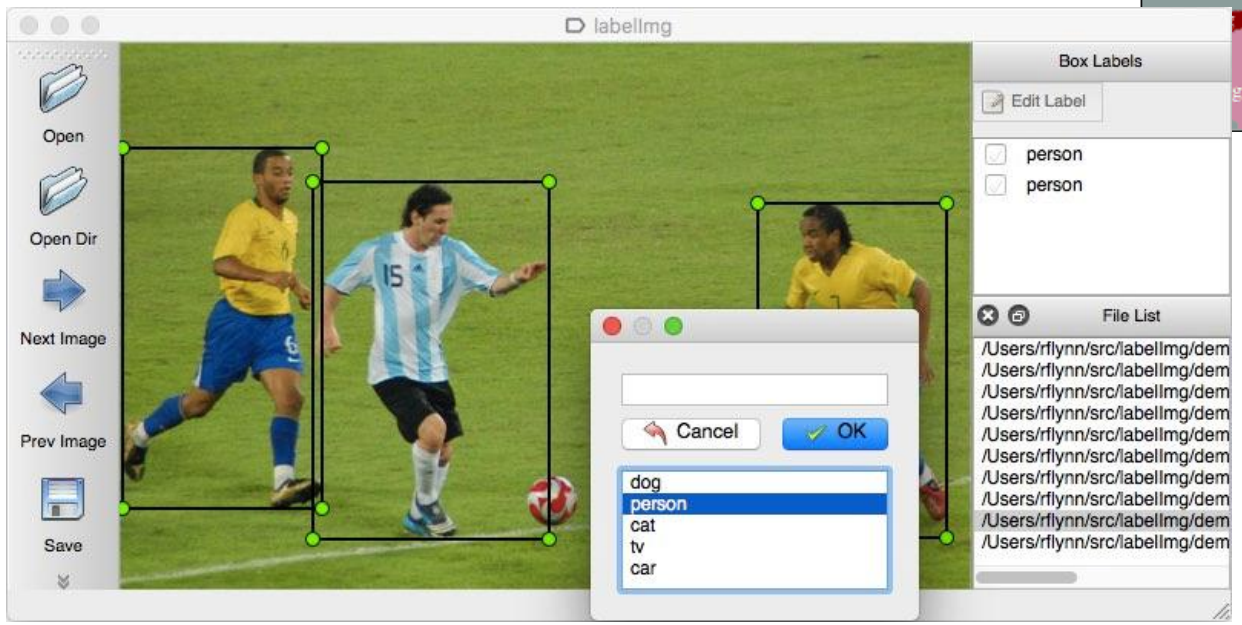
(Size in USD Billion)

Source: Market.us Scoop


Business Requirements
Analytics Approach
Data Requirements
Data Collection
Exploratory Data Analysis
Data Preparation
Model Selection
Model Evaluation
Model Deployment
Model Tuning
Project Review
The AI/Machine Learning Lifecycle
Christopher S. Penn | @cspenn | @TrustInsights | cspenn.com | TrustInsights.ai
Planning
Data
Development
Deployment

Data Collection





Computational Scene Modeling

Synthetic Data

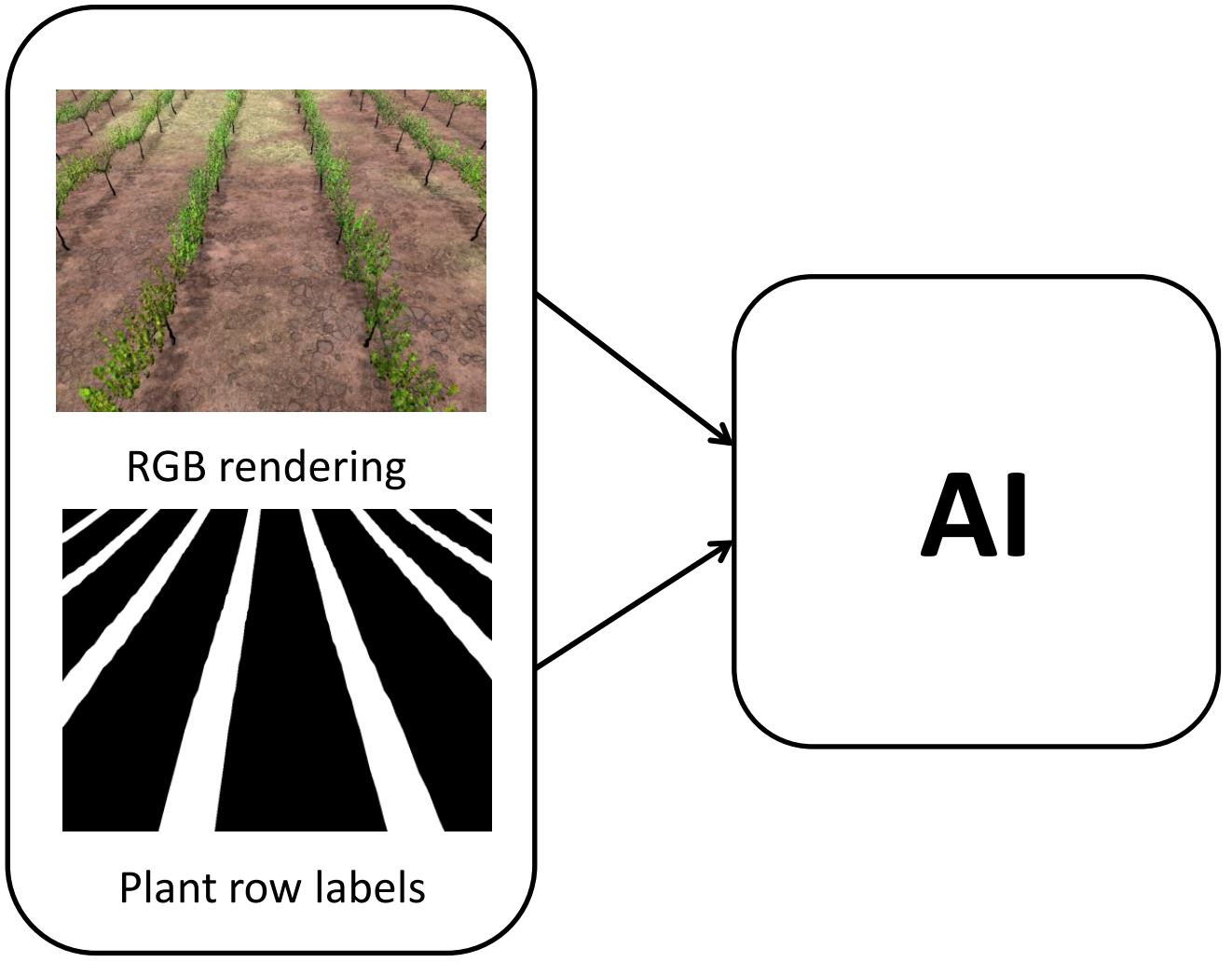
Rendering of 3D apple tree model

Semantic information is inherent

Vision Task Taxonomy

Task	Output Type	Example Metric	Synthetic Label Availability
Classification	 Class **Label**	Accuracy	✓
Detection	 Boxes + Classes	mAP	✓
Semantic Segmentation	 Pixel Mask	mIOU / Dice	✓
Pose Estimation	 Keypoints / 6D Pose	PCK / ADD(-S)	✓
Tracking	 Trajectories	MOTA / HOTA	✓

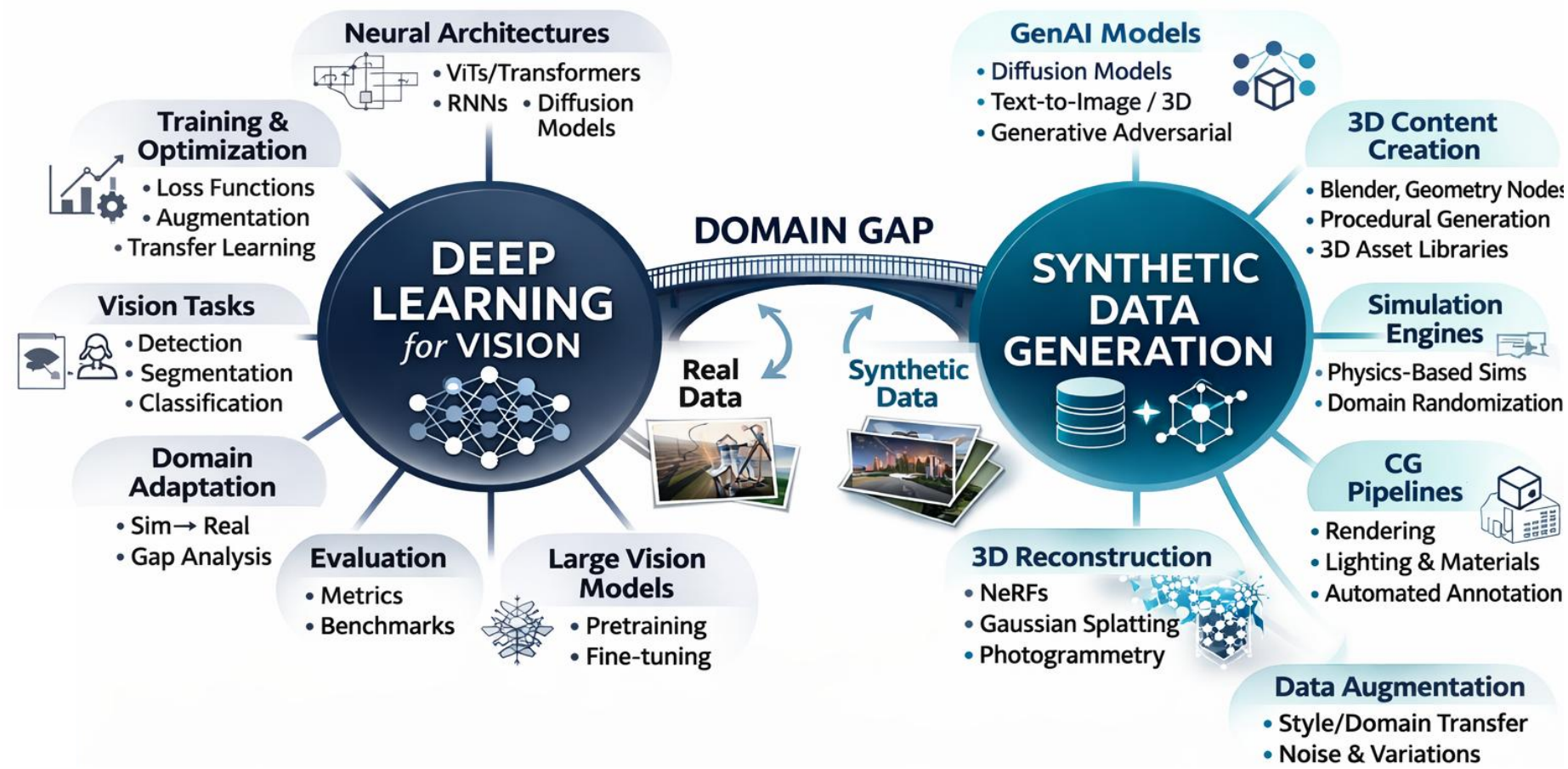
Synthetic Training Data



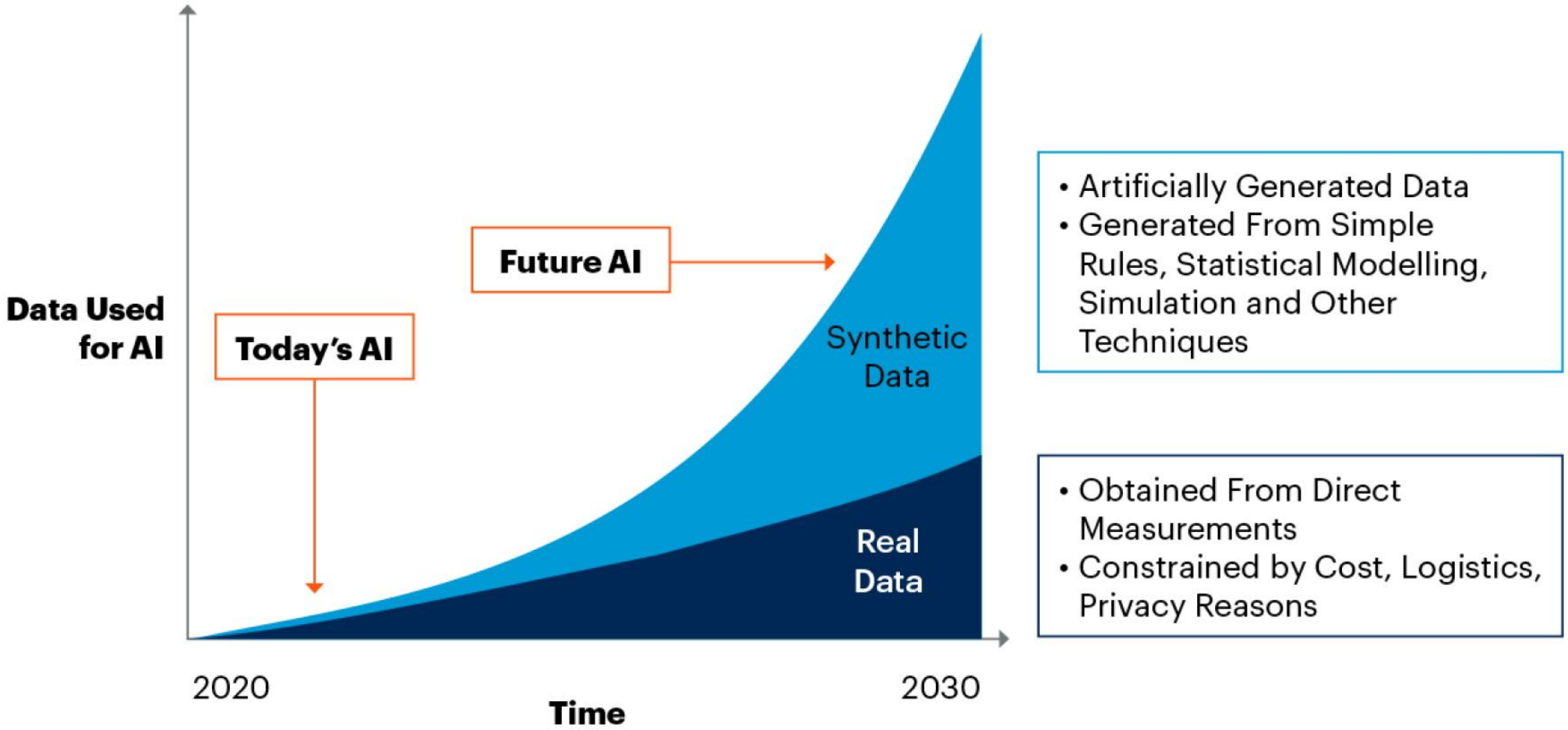
Scientific Questions

- Can synthetic data improve performance on a real benchmark?
- Under what conditions?
- Which randomization/augmentation/model choices matter?

Domain Gap


Neural Architectures
• ViTs/Transformers
• RNNs • Diffusion Models
Training & Optimization
• Loss Functions
• Augmentation
• Transfer Learning
Vision Tasks
• Detection
• Segmentation
• Classification
Domain Adaptation
• Sim→ Real
• Gap Analysis
Evaluation
• Metrics
• Benchmarks
Large Vision Models
• Pretraining
• Fine-tuning
DEEP LEARNING for VISION
DOMAIN GAP
Real Data
Synthetic Data
SYNTHETIC DATA GENERATION
GenAI Models
• Diffusion Models
• Text-to-Image / 3D
• Generative Adversarial
3D Content Creation
• Blender, Geometry Nodes
• Procedural Generation
• 3D Asset Libraries
Simulation Engines
• Physics-Based Sims
• Domain Randomization
CG Pipelines
• Rendering
• Lighting & Materials
• Automated Annotation
3D Reconstruction
• NeRFs
• Gaussian Splatting
• Photogrammetry
Data Augmentation
• Style/Domain Transfer
• Noise & Variations

By 2030, Synthetic Data Will Completely Overshadow Real Data in AI Models



Source: Gartner

Course Goals

- Introduction into **Computer Vision** and **Generative AI**
- Development of **Deep Neural Network Models** for vision tasks
- Development of an **Automatic Generative Application** for image-based synthetic datasets

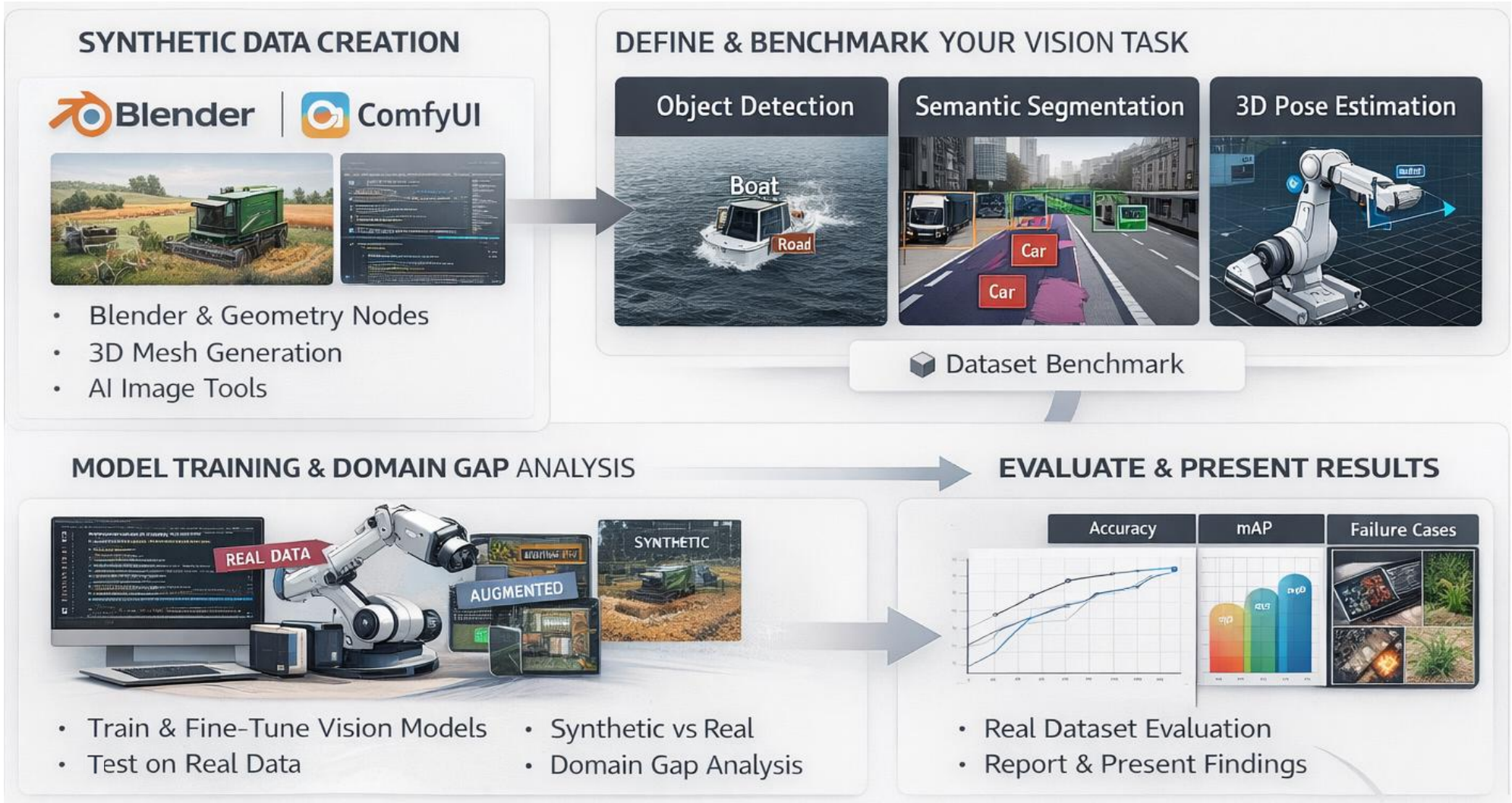
Timeline

- 1-5 lectures: **Computer Vision AI**
  - CNNs
  - Object detection
  - Semantic Segmentation
  - Transformers
- 6-8 lectures: **Hands-on Generative AI**
  - Intro to Blender
  - Geometry Nodes
  - Blender Python
- 9+ lectures: **Generative AI**
  - Diffusion Models
  - Gaussian Splatting
- All lecture slides available at: wp.faculty.wmi.amu.edu.pl/VC.html

Grading

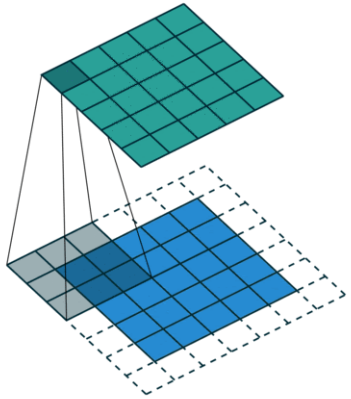
- **Written Exam:** Multiple choice about lecture material
- **5 Lab assignments 30%:** weekly lab assignment for first 5 weeks
- **End of Semester Project 70%**: Generate your own synthetic dataset with GenAI and computer graphics pipeline, train Vision Model on synthetic data, evaluate task performance on real dataset

End of Semester Project

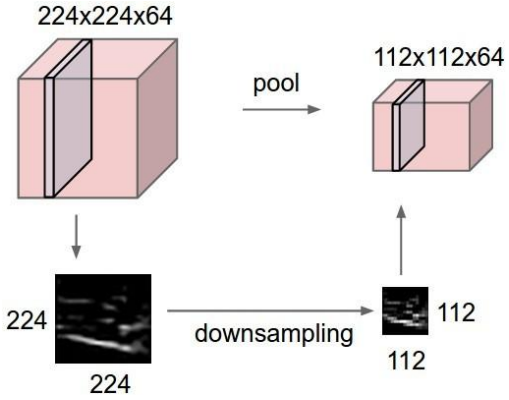


Components of a Convolutional Network

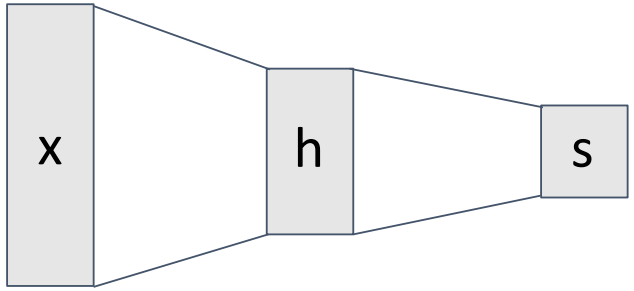
Convolution Layers



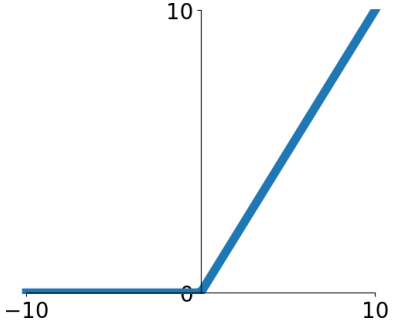
Pooling Layers



Fully-Connected Layers



Activation Function

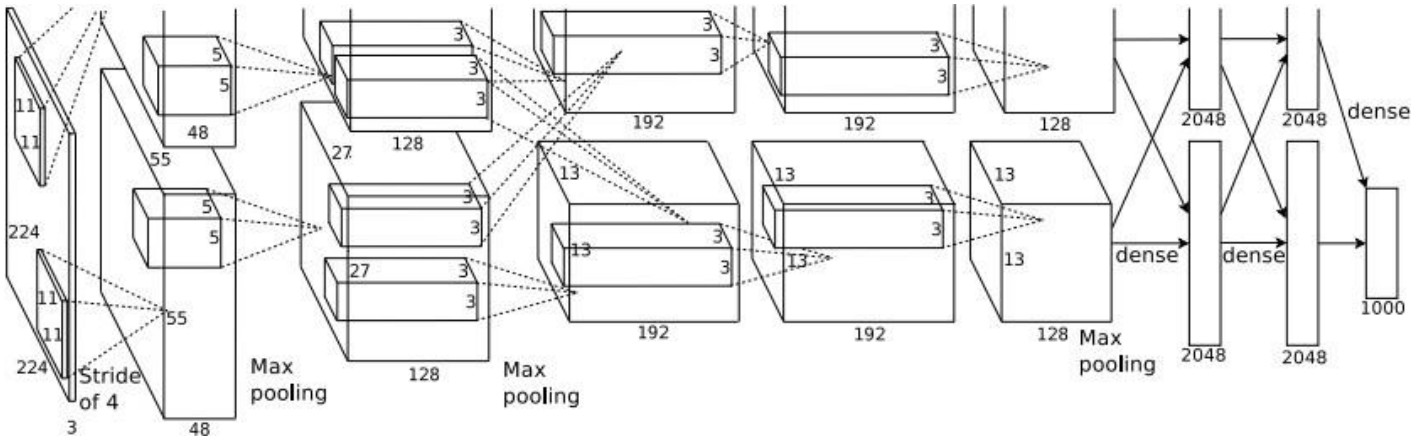


Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Inside Convolutional Networks

Input Image:
3 x 224 x 224

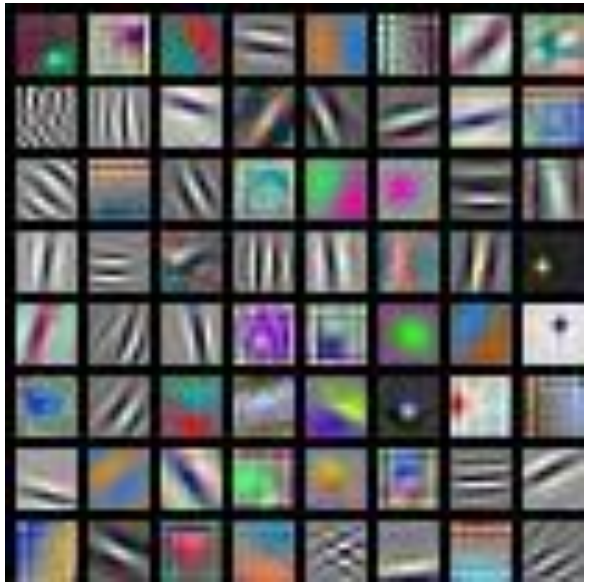


Class Scores:
1000 numbers

What are the intermediate features looking for?

Krizhevsky et al, “ImageNet Classification with Deep Convolutional Neural Networks”, NeurIPS 2012.

First Layer: Visualize Filters

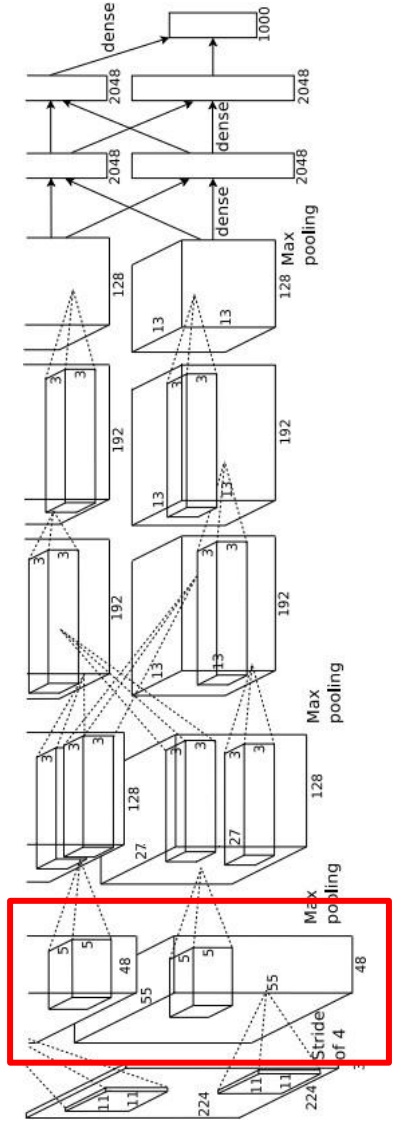


AlexNet:
64 x 3 x 11 x 11

ResNet-18:
64 x 3 x 7 x 7

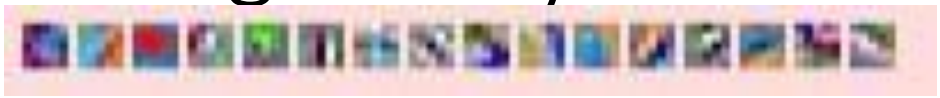
ResNet-101:
64 x 3 x 7 x 7

DenseNet-121:
64 x 3 x 7 x 7



Krizhevsky, “One weird trick for parallelizing convolutional neural networks”, arXiv 2014
He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016
Huang et al, “Densely Connected Convolutional Networks”, CVPR 2017

Higher Layers: Visualize Filters



First layer weights: 16 x 3 x 7 x 7

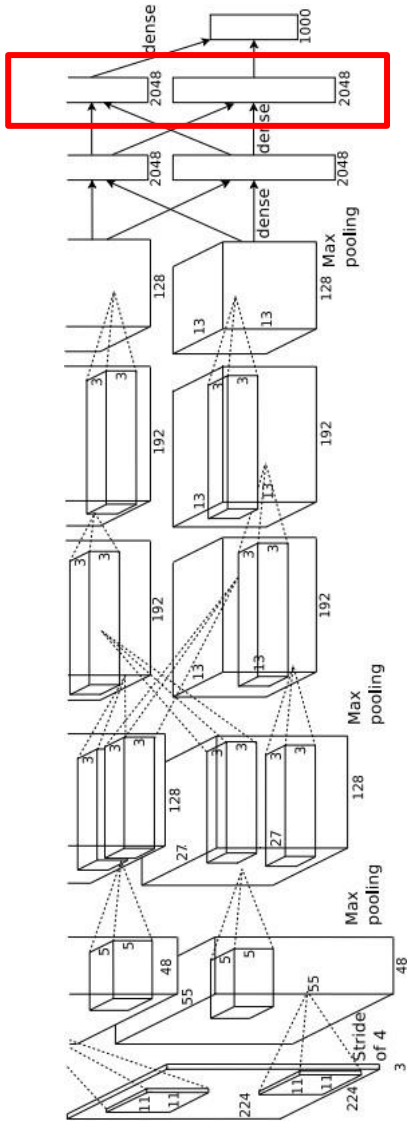
We can visualize filters at higher layers, but not that interesting

Second layer weights: 20 x 16 x 7 x 7

Third layer weights: 20 x 20 x 7 x 7

Last Layer

FC7 layer



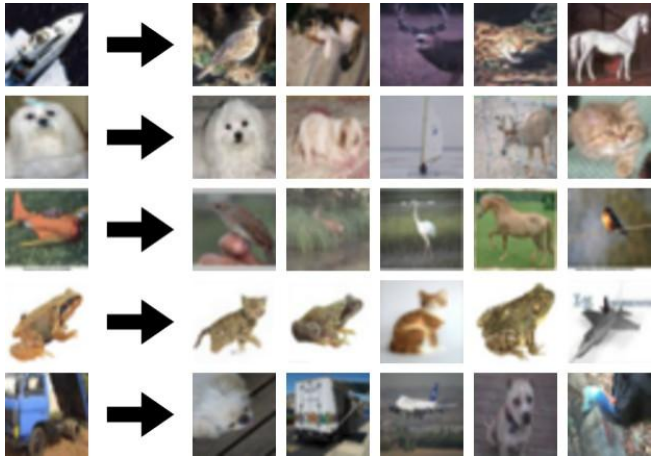
4096-dimensional feature vector for an image (layer immediately before the classifier)

Run the network on many images, collect the feature vectors

Krizhevsky et al, “ImageNet Classification with Deep Convolutional Neural Networks”, NeurIPS 2012.
Figures reproduced with permission.

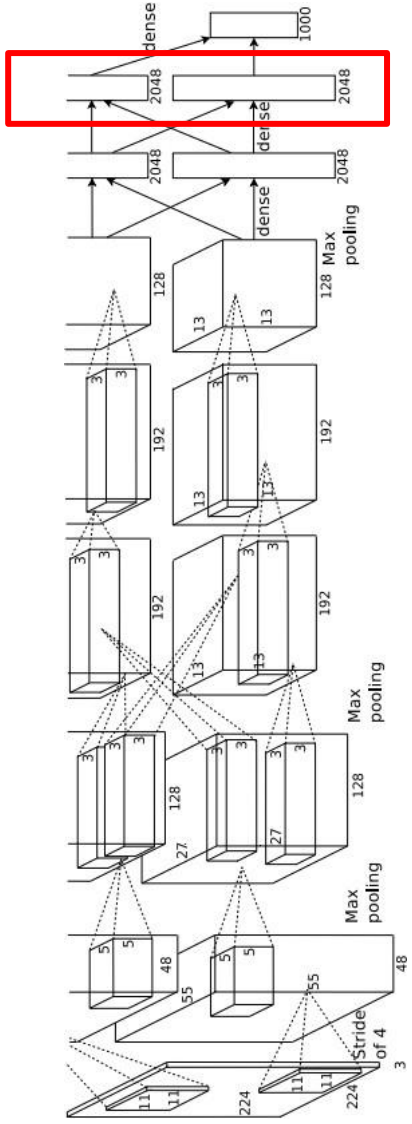
Last Layer: Nearest Neighbors

Recall: Nearest neighbors in <u>pixel</u> space



Test image

L2 Nearest neighbors in <u>feature</u> space



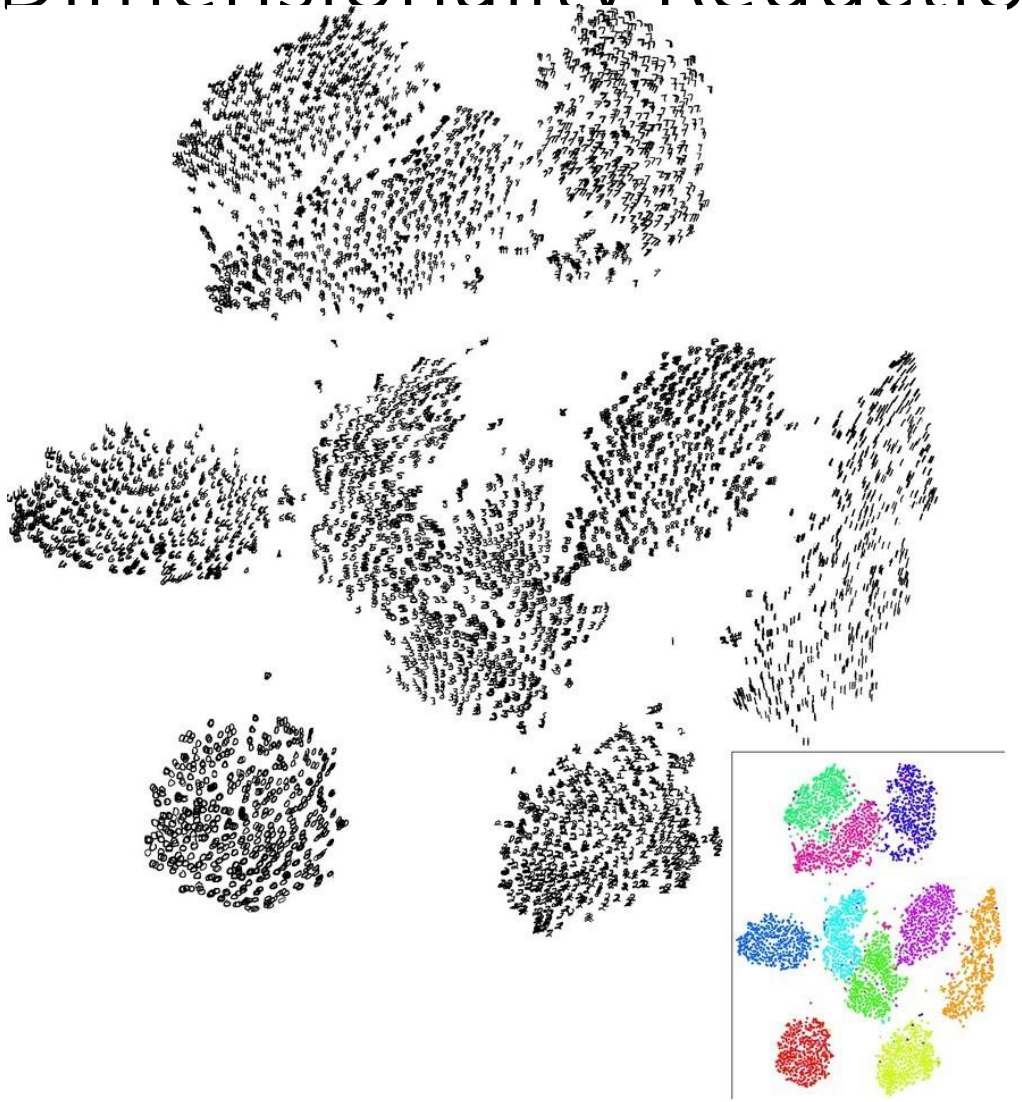
Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NeurIPS 2012.
Figures reproduced with permission.

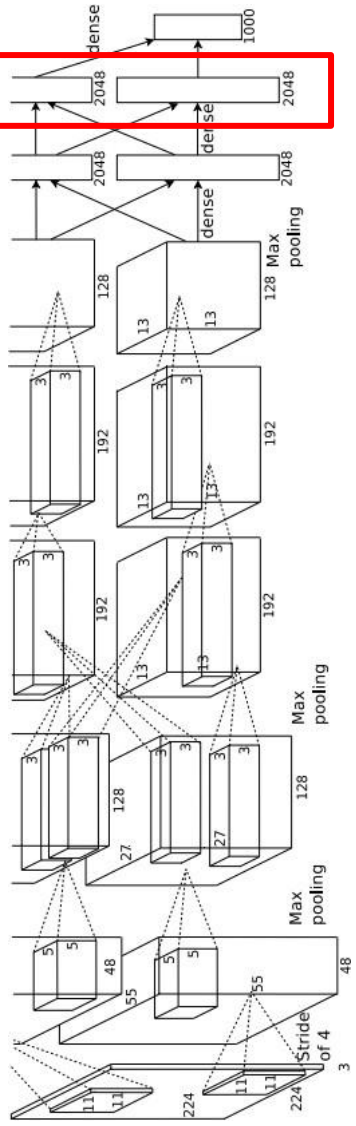
Last Layer: Dimensionality Reduction

Visualize the “space” of FC7 feature vectors by reducing dimensionality of vectors from 4096 to 2 dimensions

Simple algorithm: Principal Component Analysis (PCA)

More complex: **t-SNE**

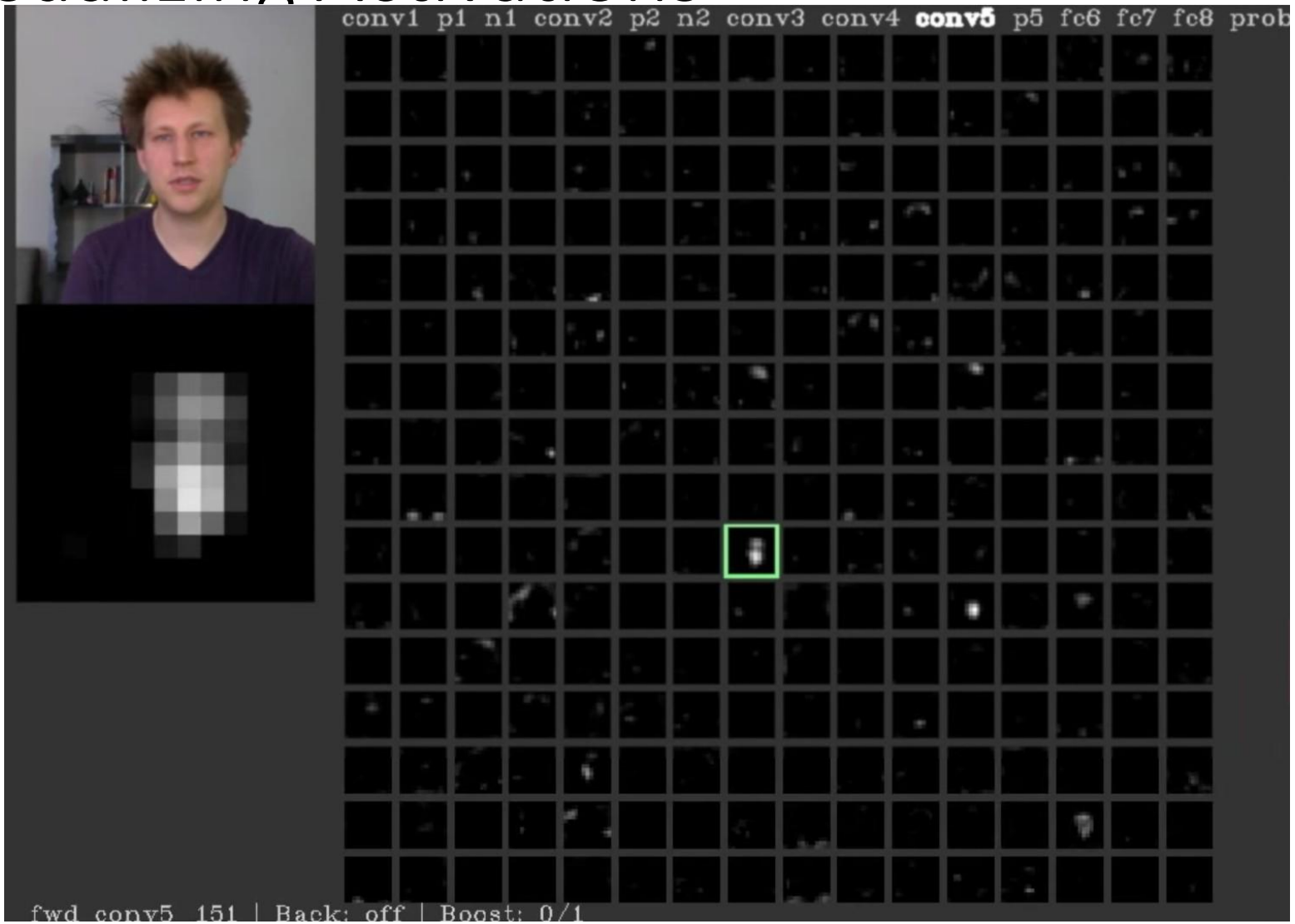




Van der Maaten and Hinton, “Visualizing Data using t-SNE”, JMLR 2008

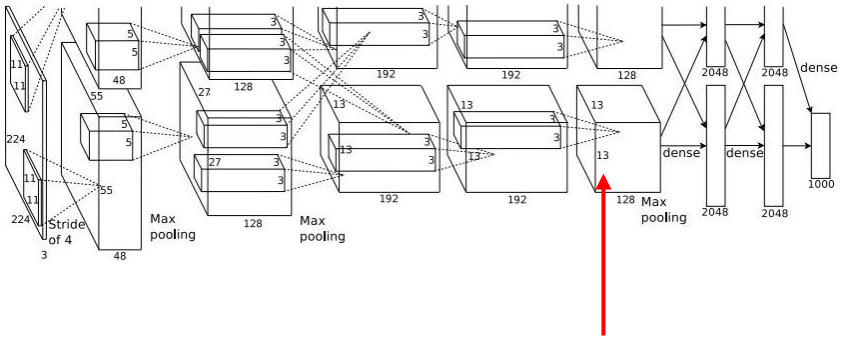
Visualizing Activations

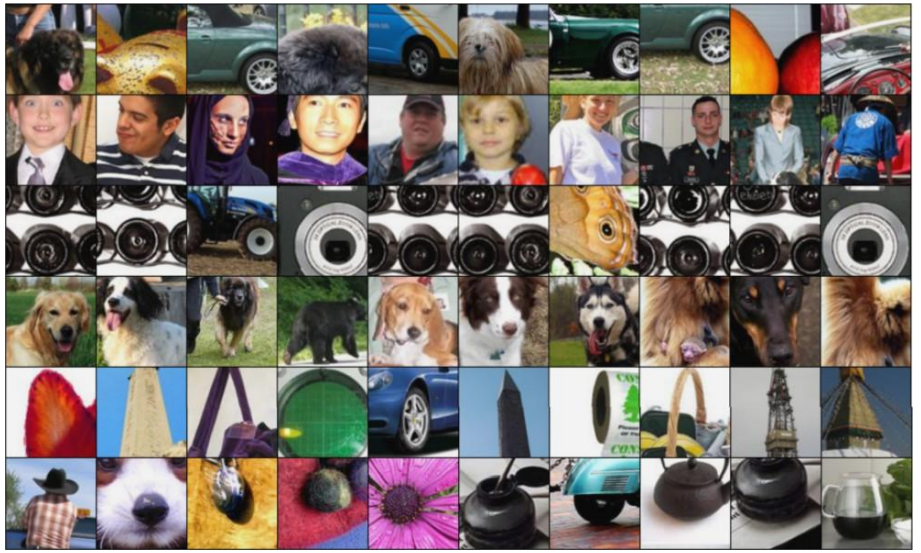
conv5 feature map is 128x13x13; visualize as 128 13x13 grayscale images



Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.

Maximally Activating





Springenberg et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015
Figure copyright Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, 2015; reproduced with permission.

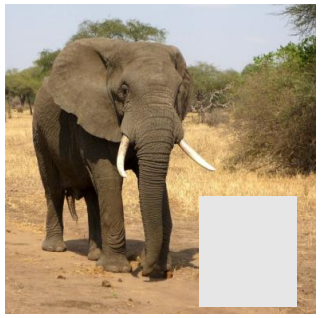
Pick a layer and a channel; e.g. conv5 is 128 x 13 x 13, pick channel 17/128

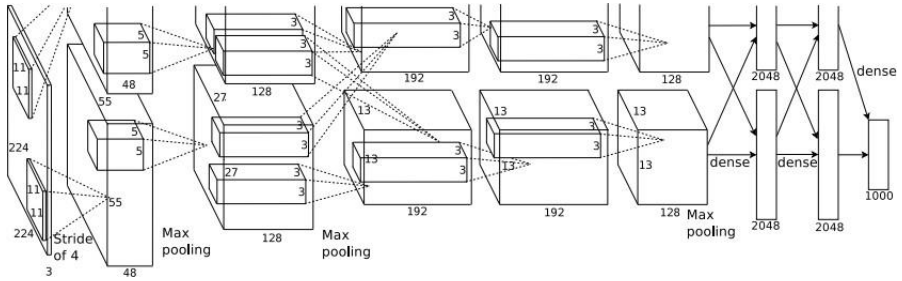
Run many images through the network, record values of chosen channel

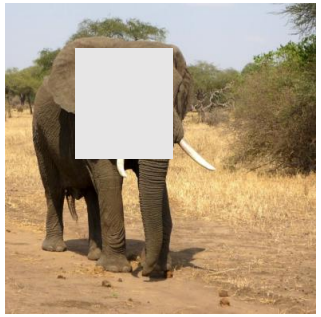
Visualize image patches that correspond to maximal activations

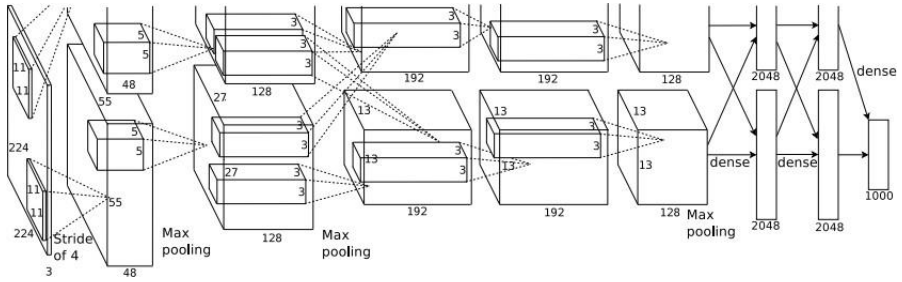
Which Pixels Matter? Occlusion

Mask part of the image before feeding to CNN, check how much predicted probabilities change





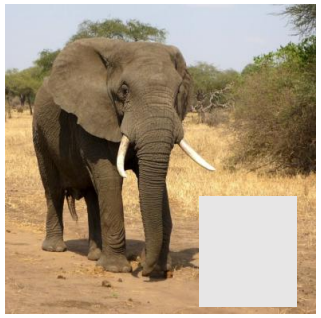


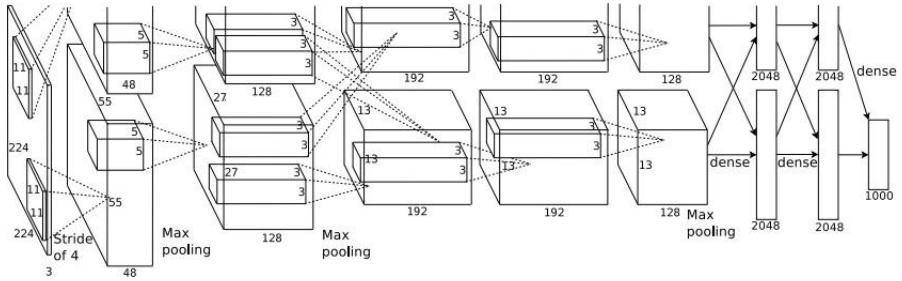


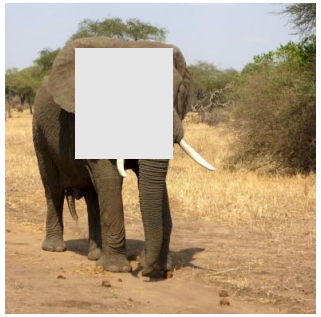
Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

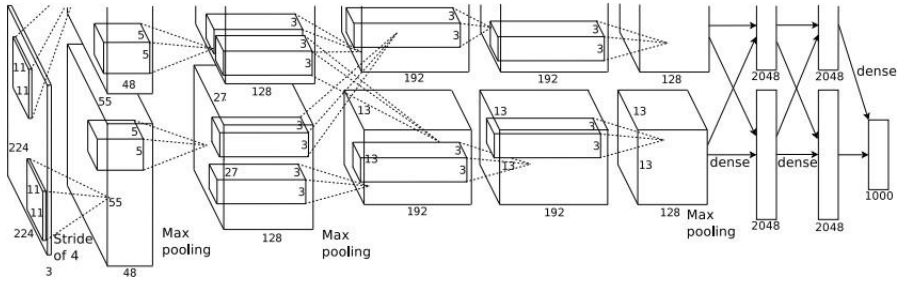
Which Pixels Matter? Occlusion

Mask part of the image before feeding to CNN, check how much predicted probabilities change

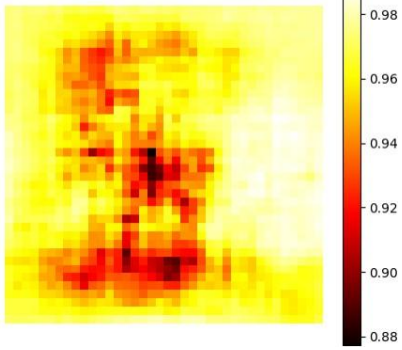




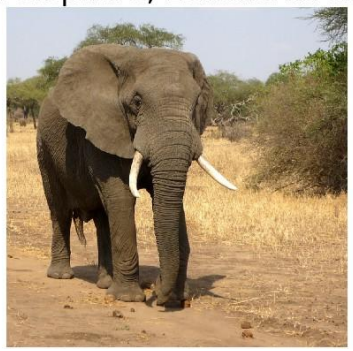


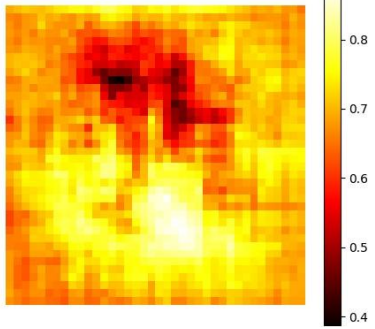


schooner

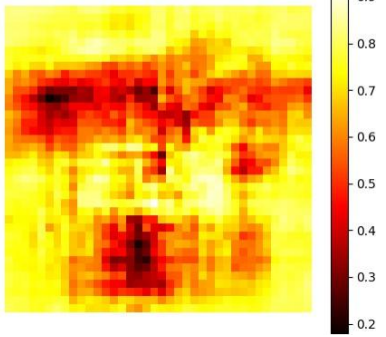


African elephant, Loxodonta africana



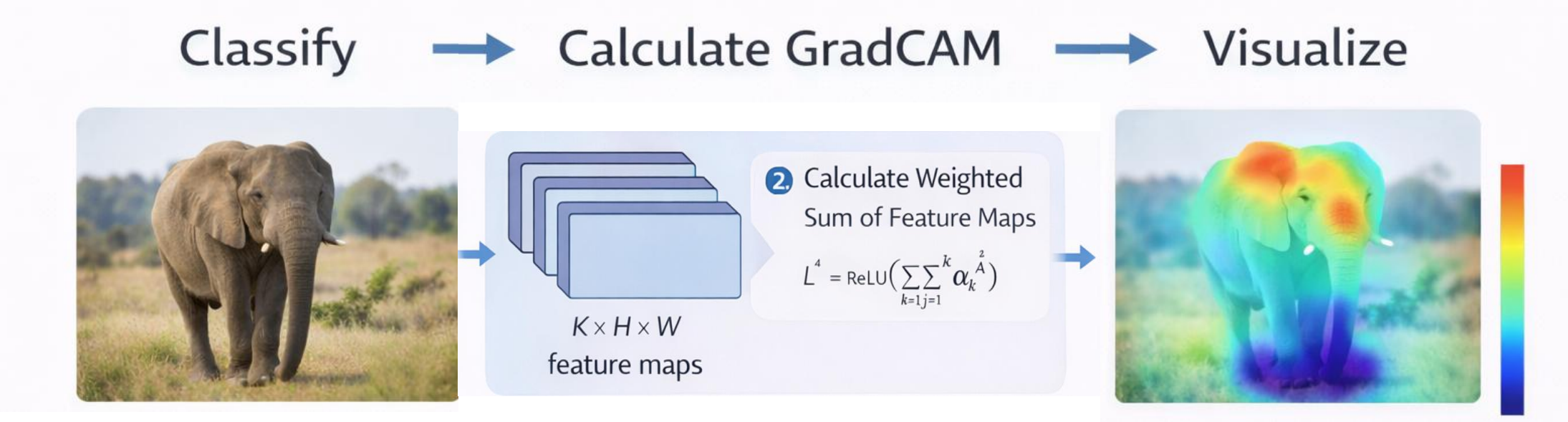


go-kart



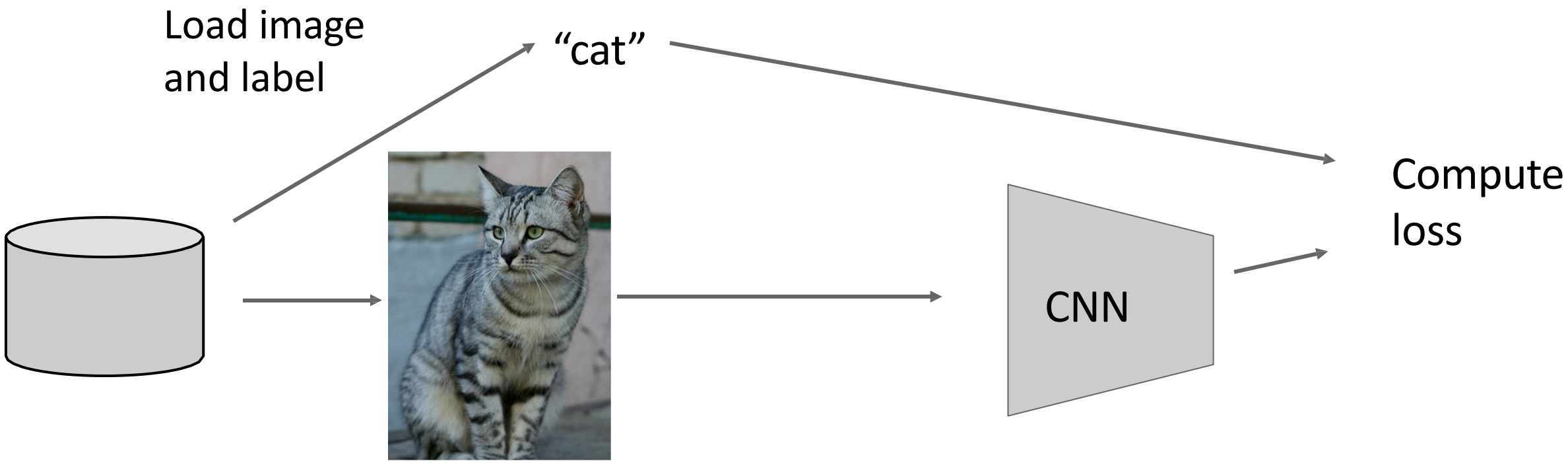
Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

Using Grad-CAM to visualize Predictions

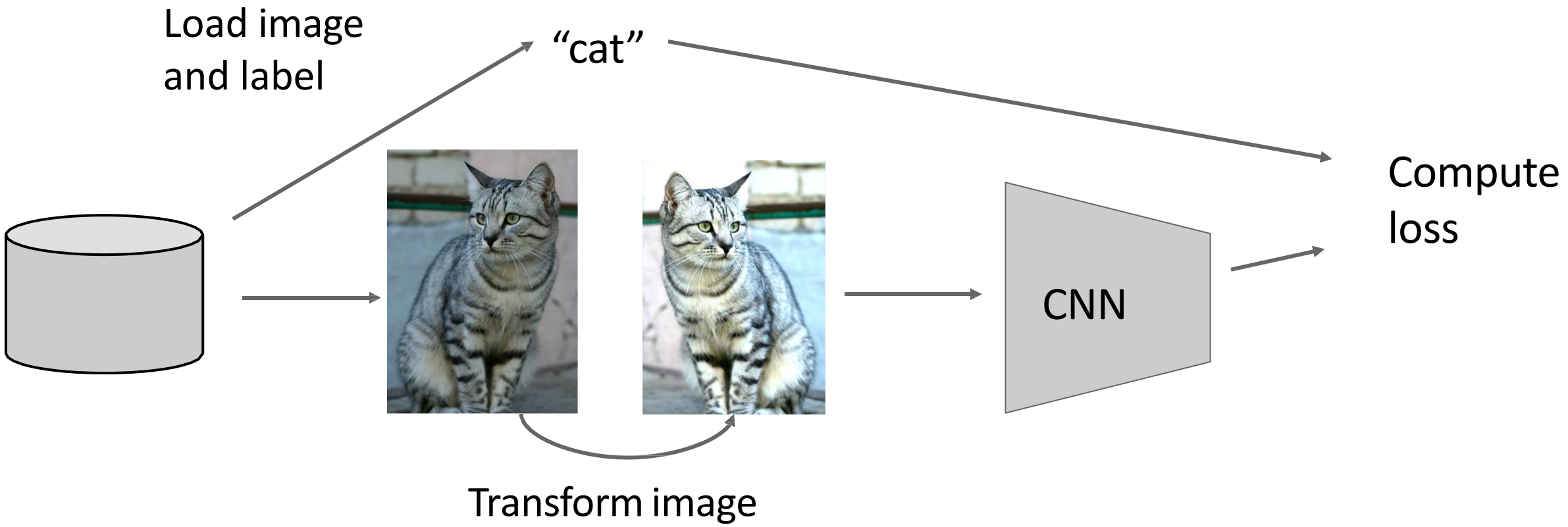


Data Augmentation

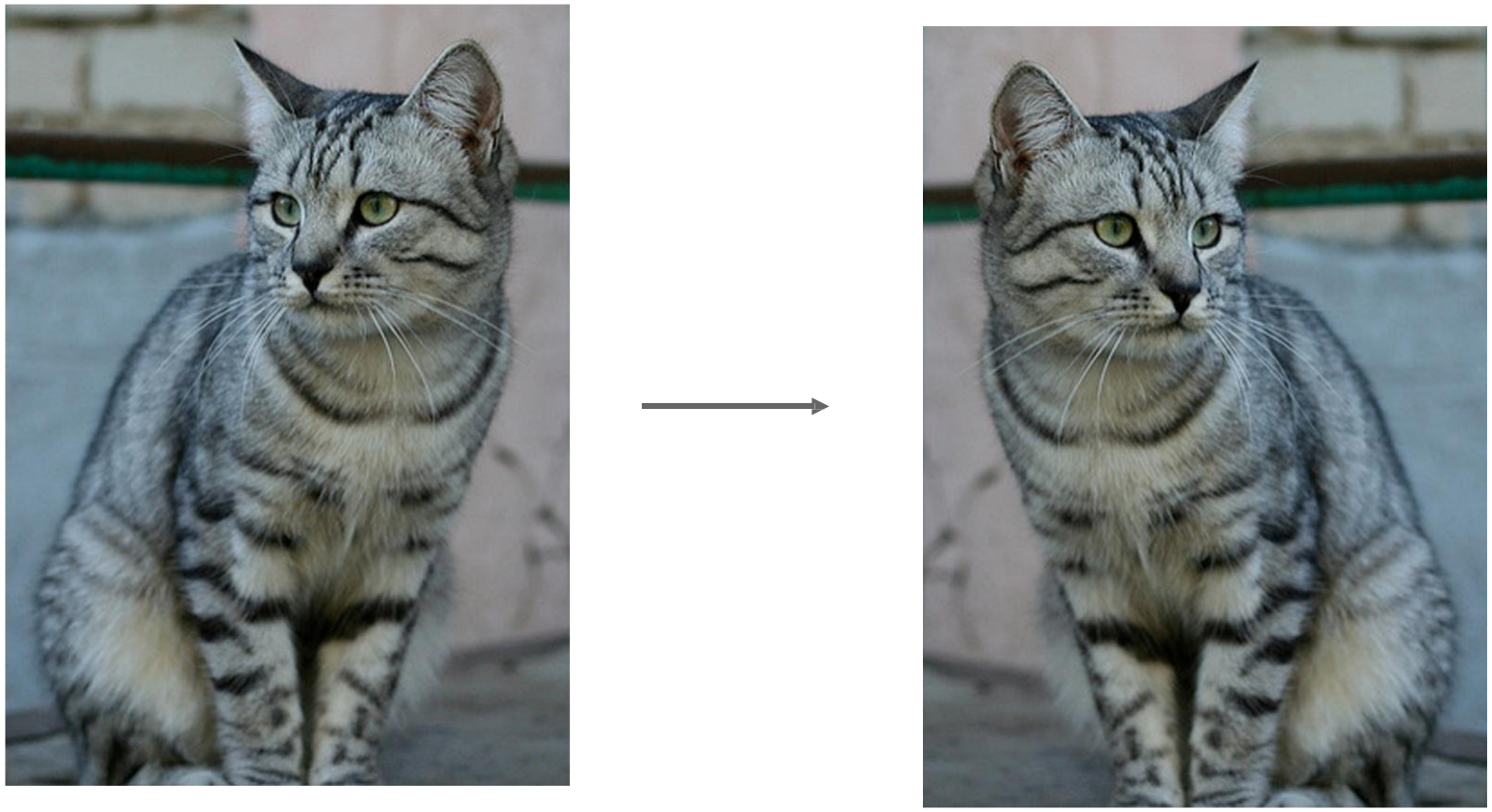
Data Augmentation



Data Augmentation



Data Augmentation: Horizontal Flips

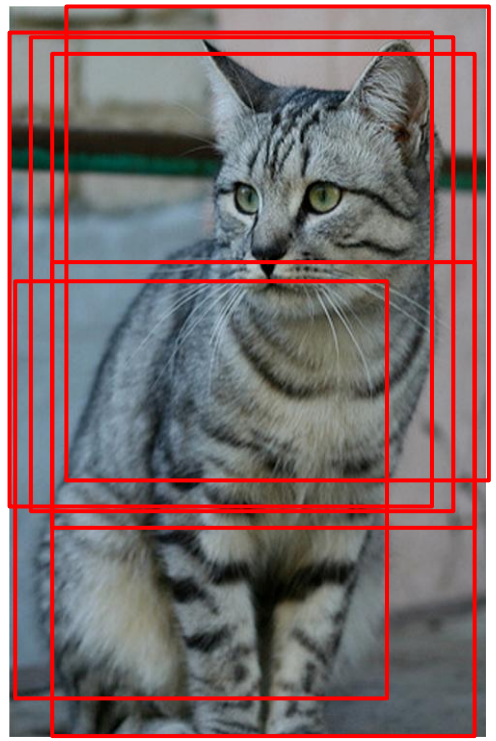


Data Augmentation: Random Crops and Scales

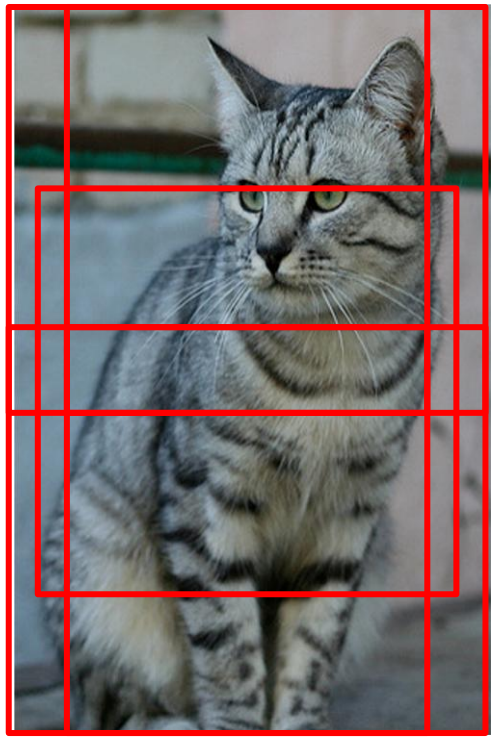
Training: sample random crops / scales

ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch



Data Augmentation: Random Crops and Scales



Training: sample random crops / scales

ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

Testing: average a fixed set of crops

ResNet:

1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224 x 224 crops

Data Augmentation: Color Jitter

Simple: Randomize contrast and brightness

More Complex:

Add noise to color intensity of pixels

Non-linear color intensity transformation

Histogram Matching

Data Augmentation : Cutout

Training: Train on random blends of images

Testing: Use original images

Examples:

Dropout

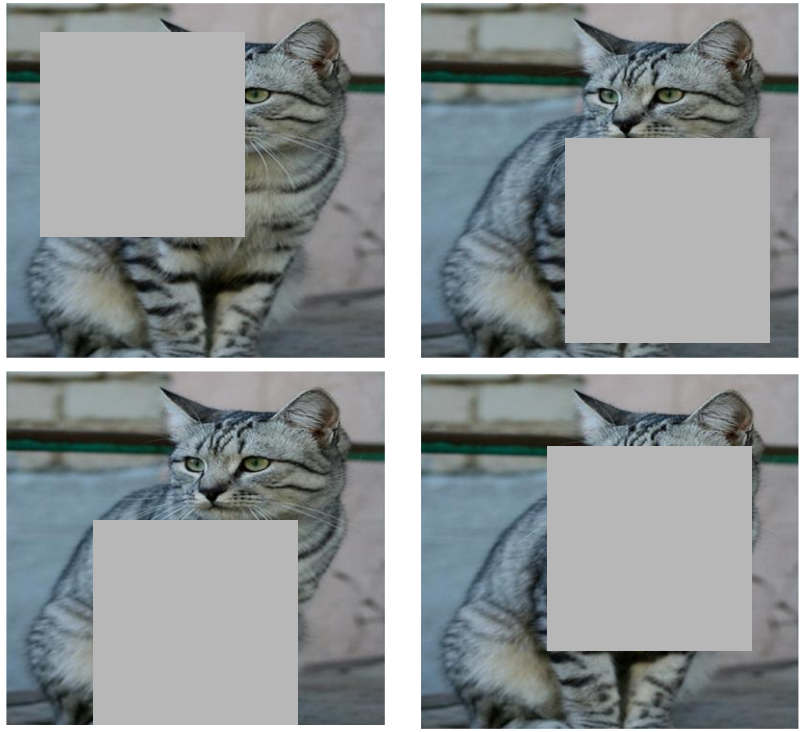
Batch Normalization

Data Augmentation

DropConnect

Stochastic Depth

Cutout



Works very well for small datasets like CIFAR, less common for large datasets like ImageNet

DeVries and Taylor, “Improved Regularization of Convolutional Neural Networks with Cutout”, arXiv 2017

Data Augmentation: Many Possibilities

Combinations of :

- translation
- rotation
- stretching
- shearing,
- lens distortions,
- compression, etc.

Regularization: Add term to the loss

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

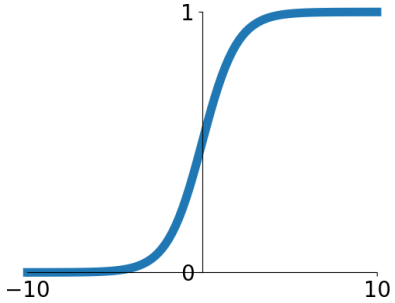
L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Activation Functions

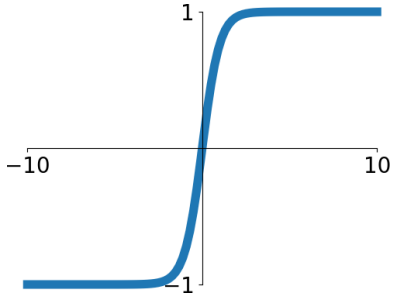
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



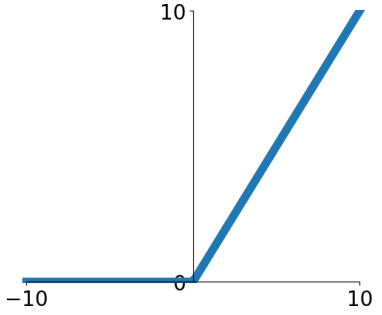
tanh

$$\tanh(x)$$



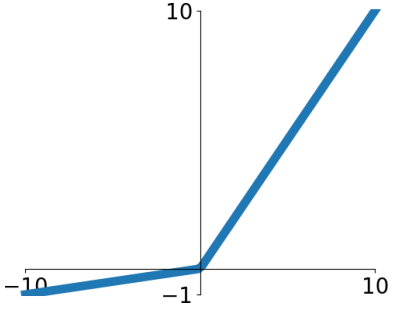
ReLU

$$\max(0, x)$$



Leaky ReLU

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

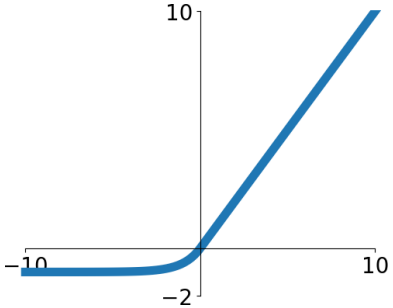


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

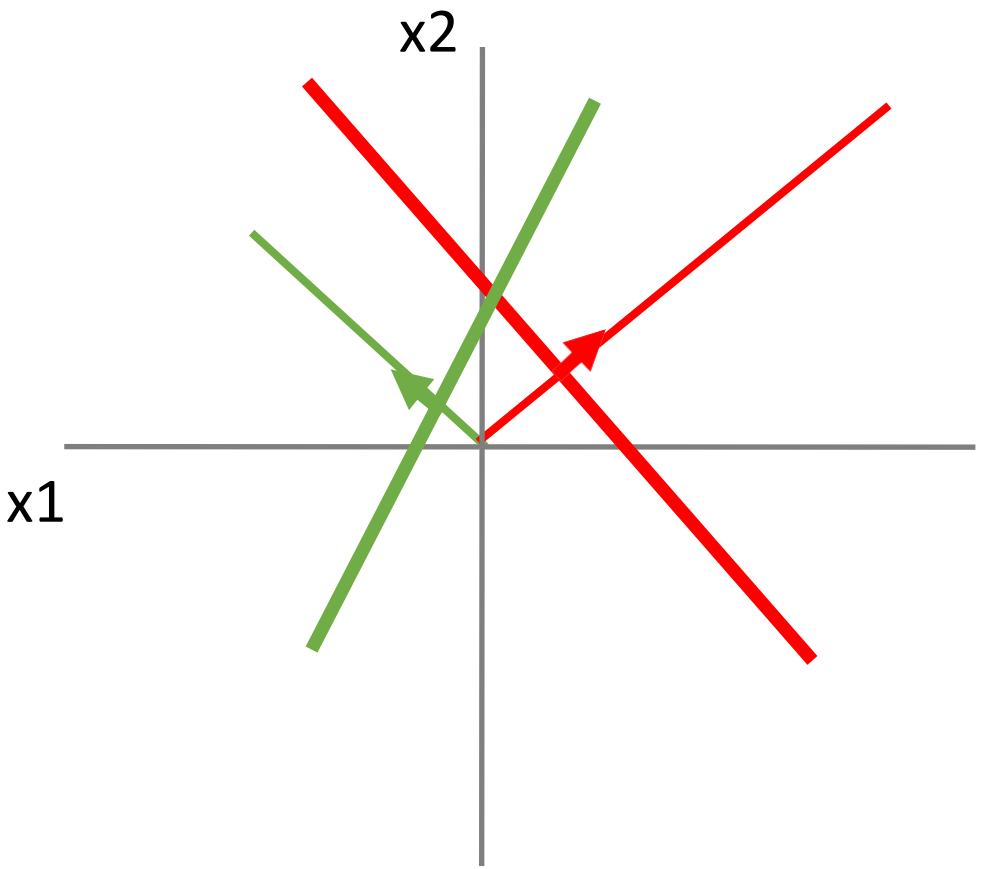
ELU

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



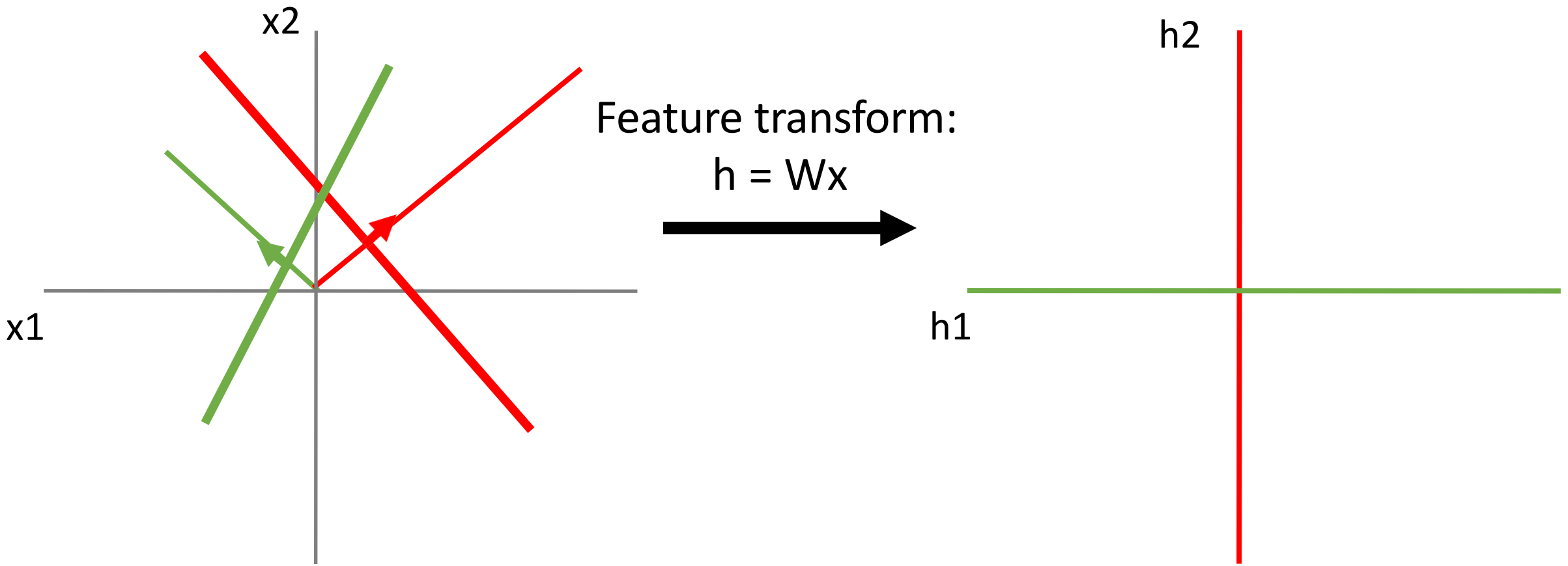
Feature Transform

Consider a linear transform: h = Wx
Where x, h are both 2-dimensional



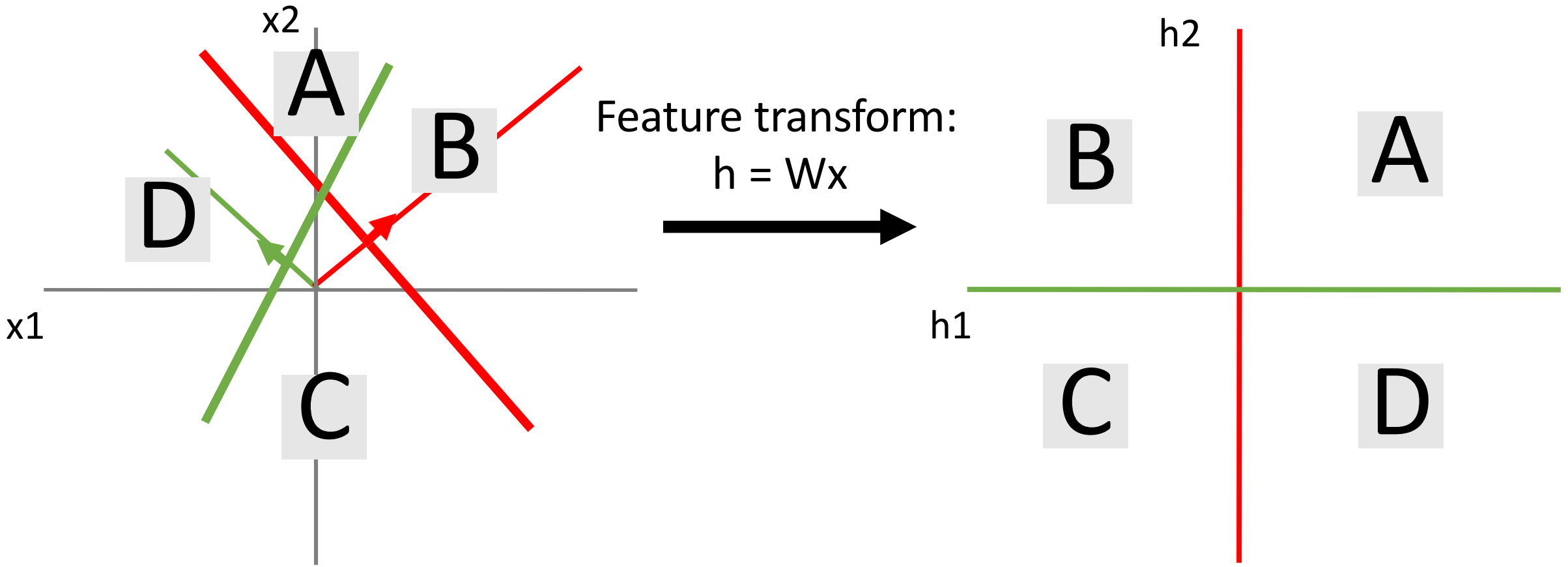
Feature Transform

Consider a linear transform: h = Wx
Where x, h are both 2-dimensional



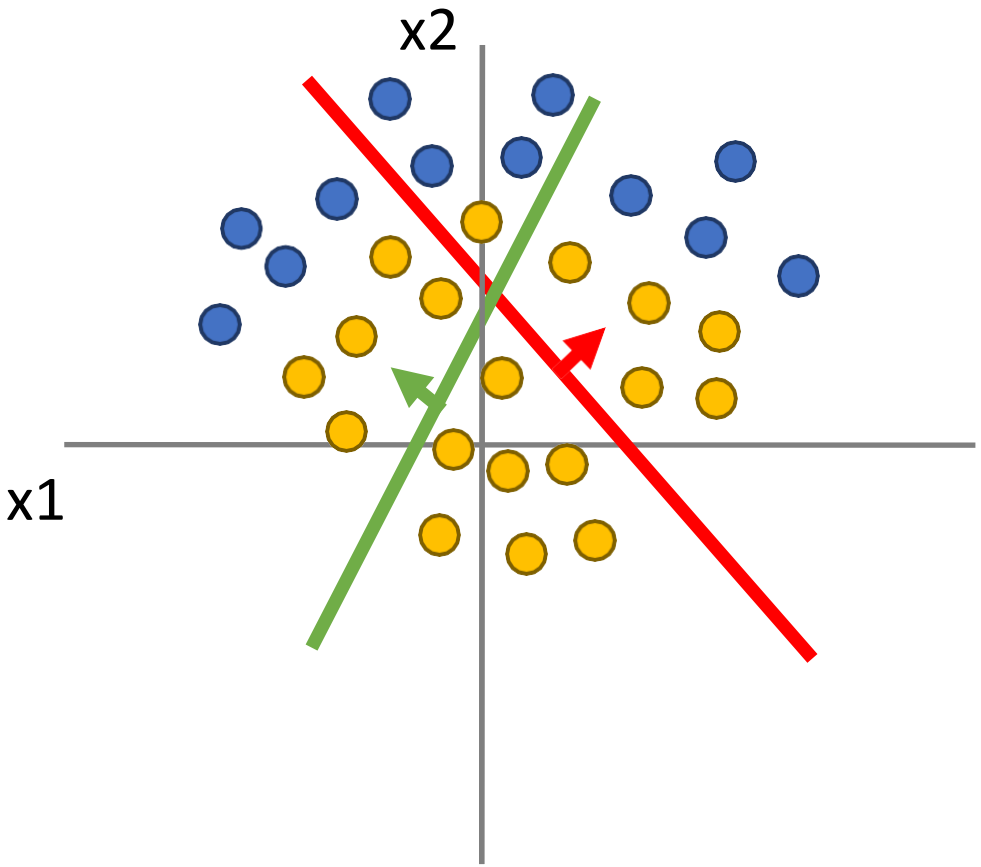
Feature Transform

Consider a linear transform: h = Wx
Where x, h are both 2-dimensional



Feature Transform

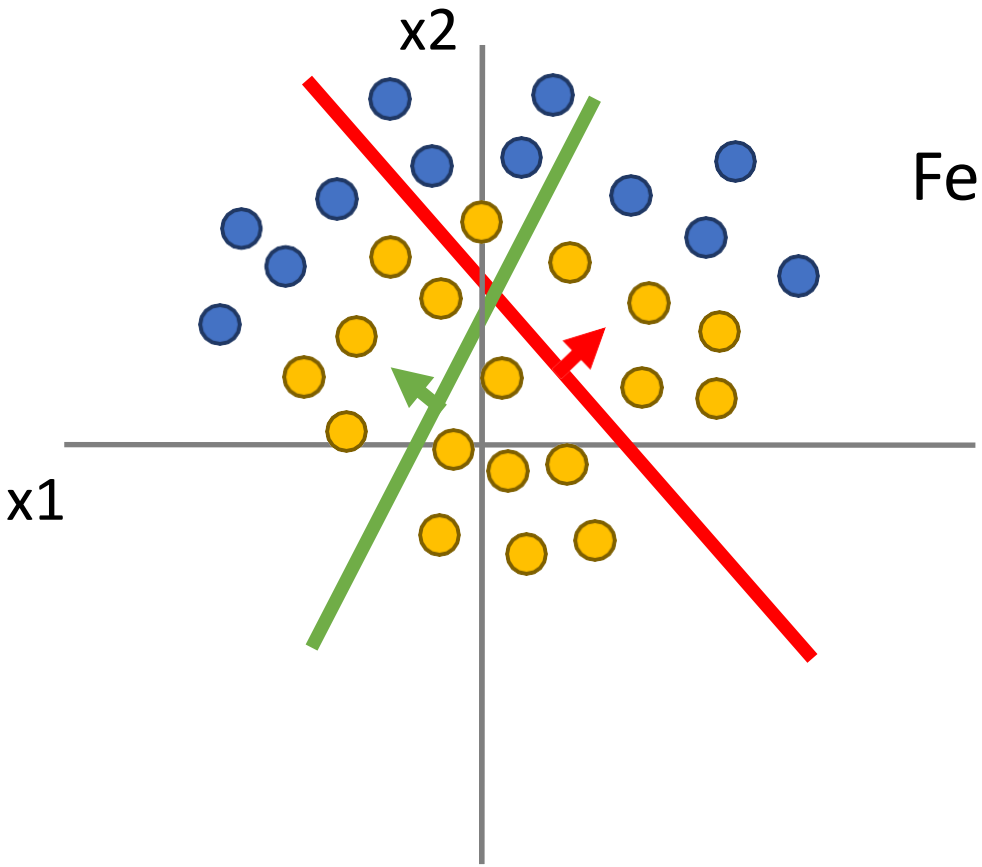
Points not linearly separable in original space



Consider a linear transform: h = Wx
Where x, h are both 2-dimensional

Feature Transform

Points not linearly separable in original space

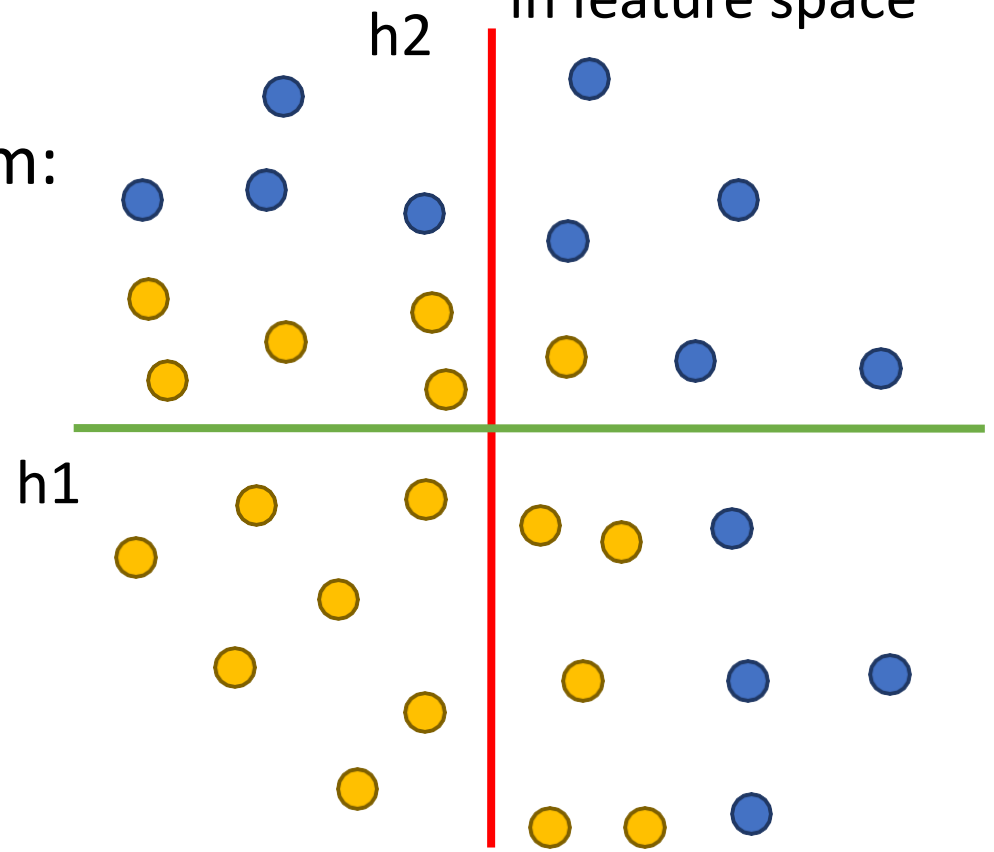


Consider a linear transform: h = Wx
Where x, h are both 2-dimensional

Feature transform:

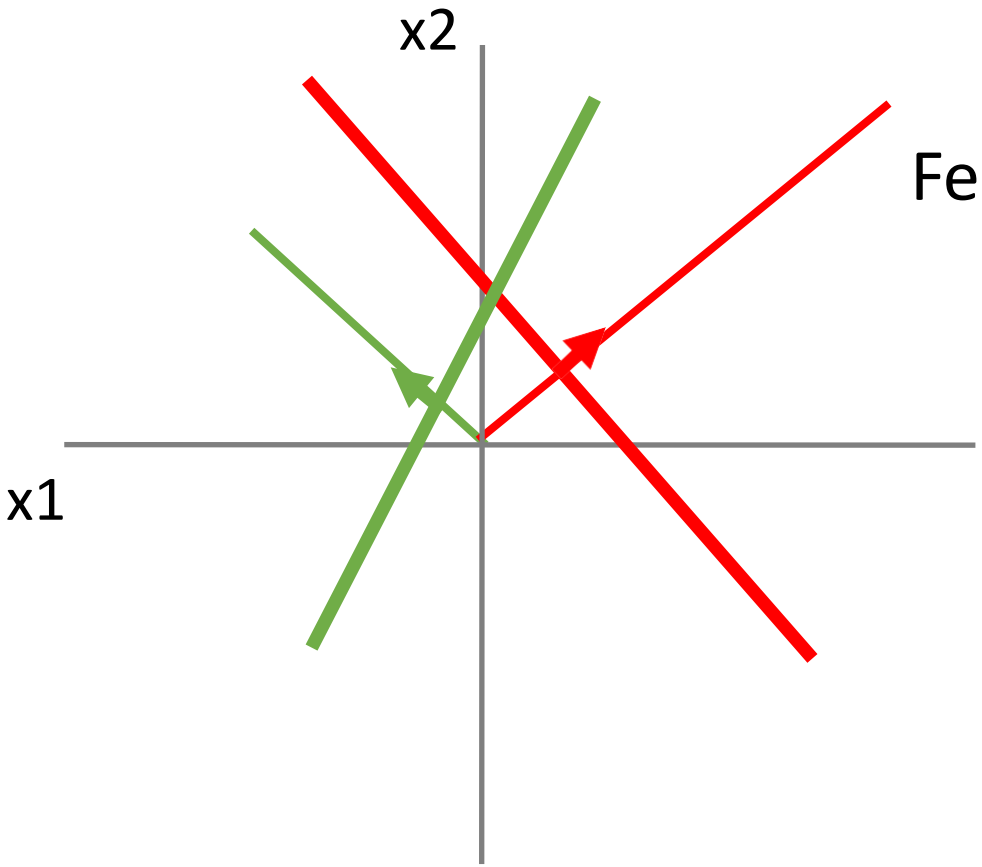
h = Wx

Not linearly separable in feature space



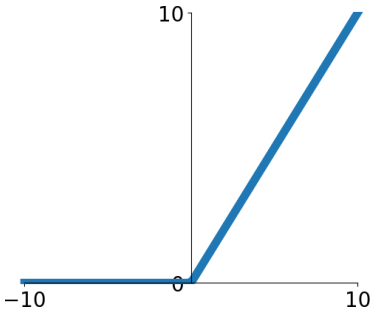
Feature Transform

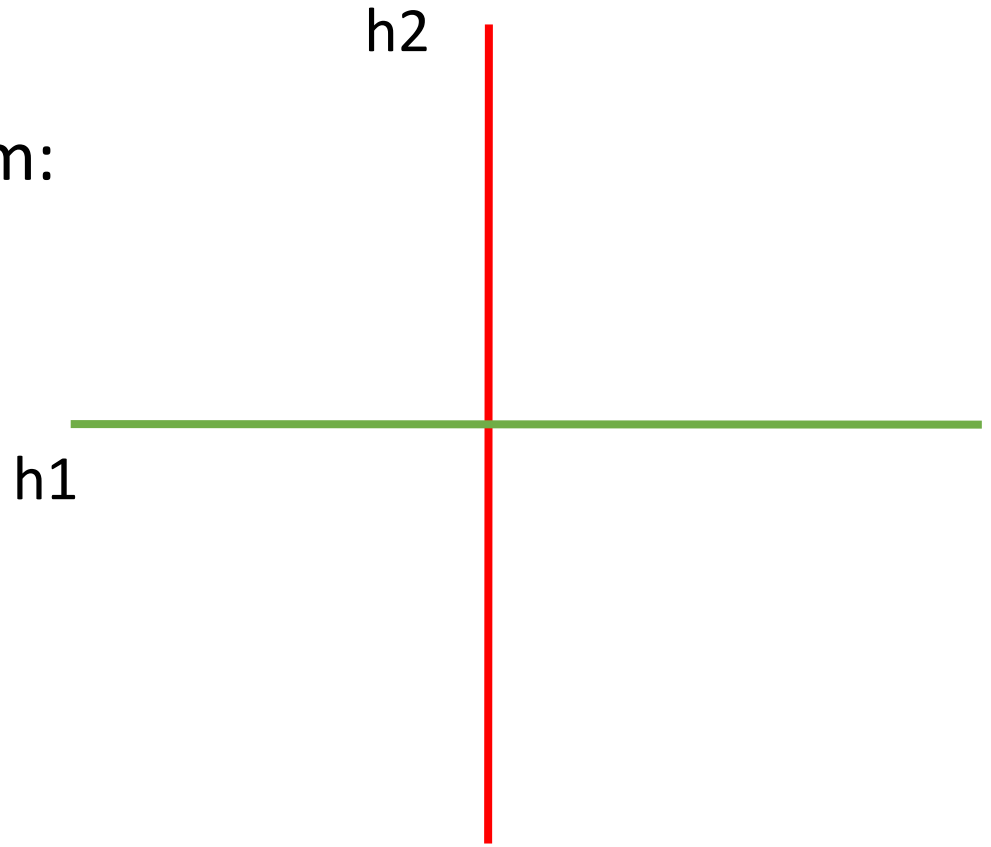
Consider a neural net hidden layer:
h = ReLU(Wx) = max(0, Wx)
Where x, h are both 2-dimensional



Feature transform:

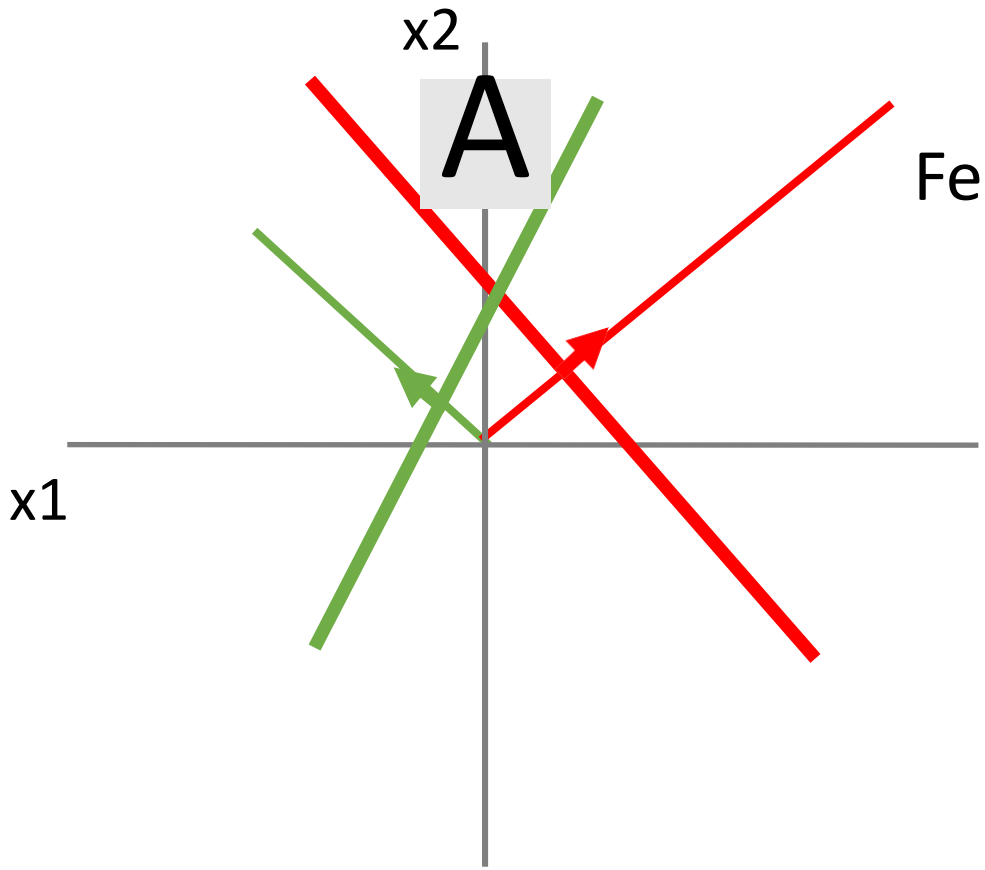
h = ReLU(Wx)



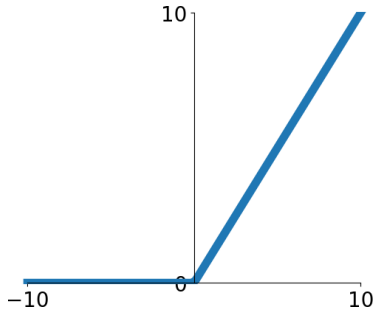


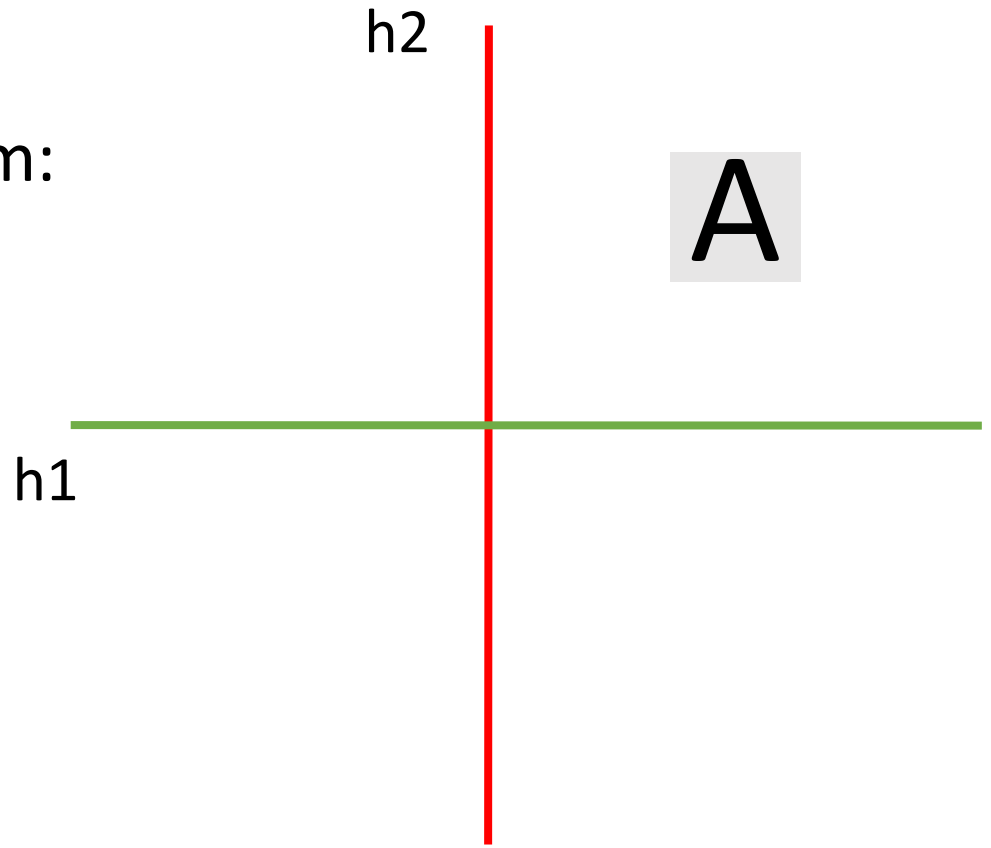
Feature Transform

Consider a neural net hidden layer:
h = ReLU(Wx) = max(0, Wx)
Where x, h are both 2-dimensional



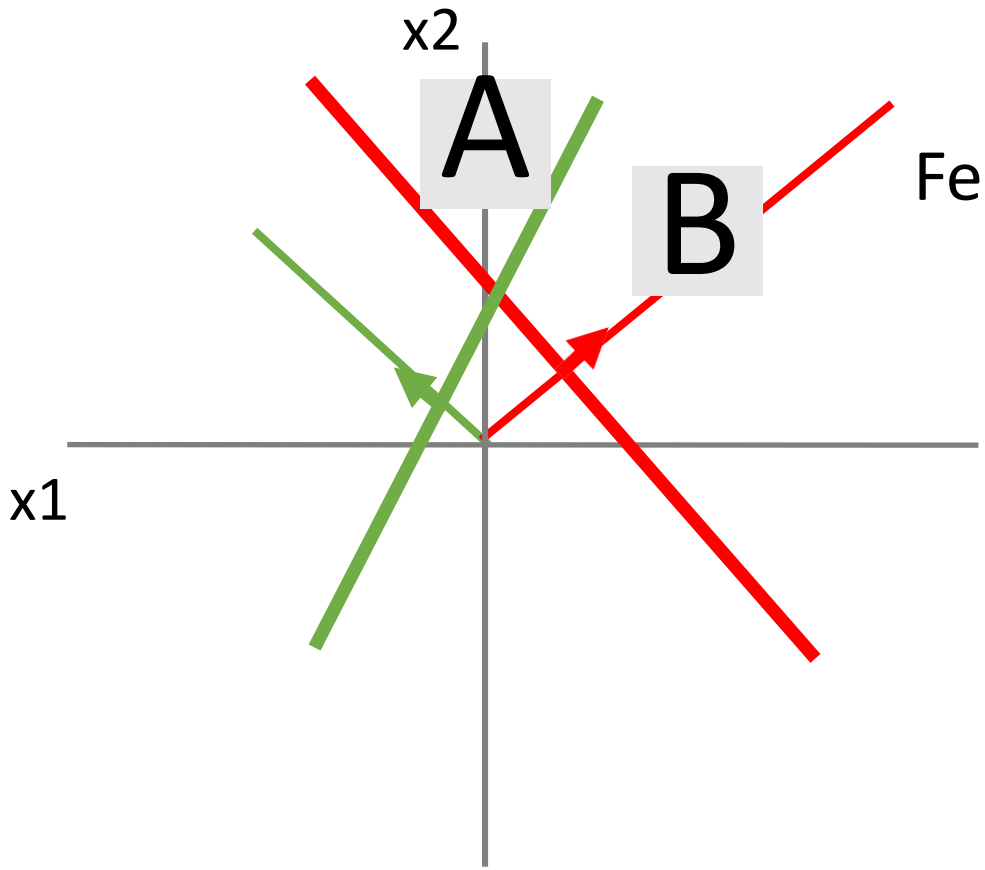
Feature transform:
h = ReLU(Wx)



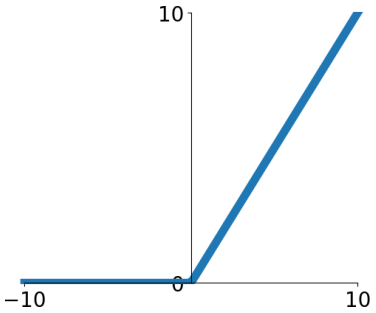


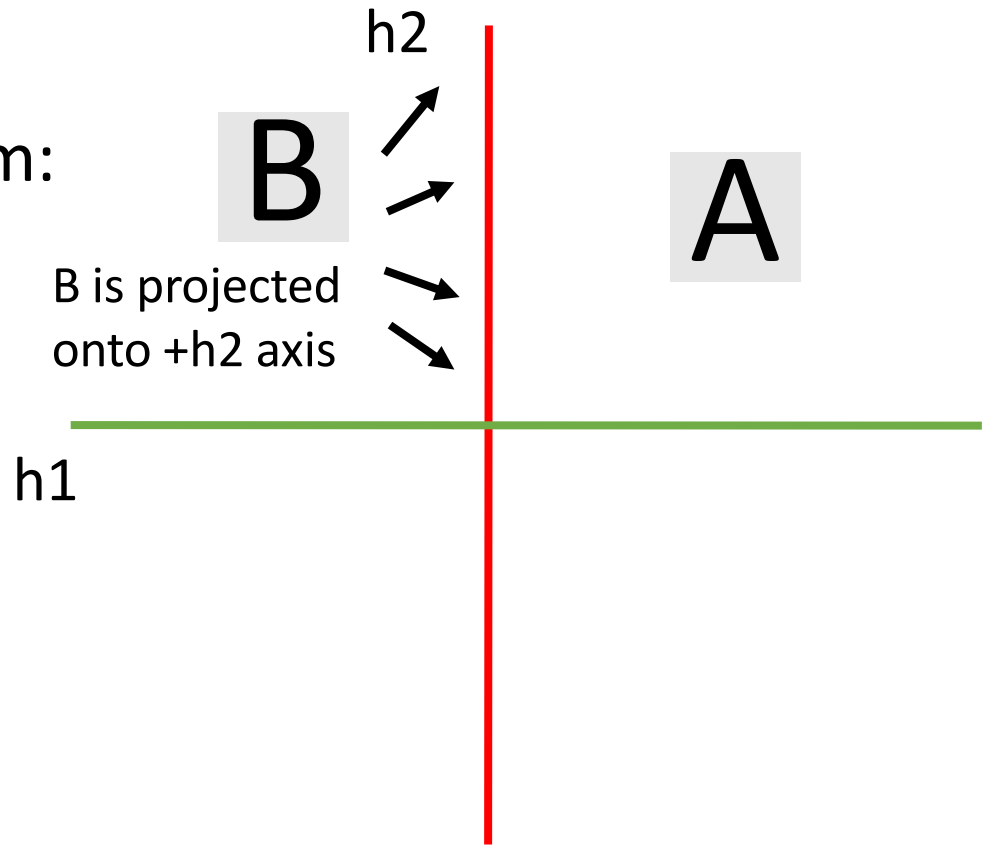
Feature Transform

Consider a neural net hidden layer:
h = ReLU(Wx) = max(0, Wx)
Where x, h are both 2-dimensional



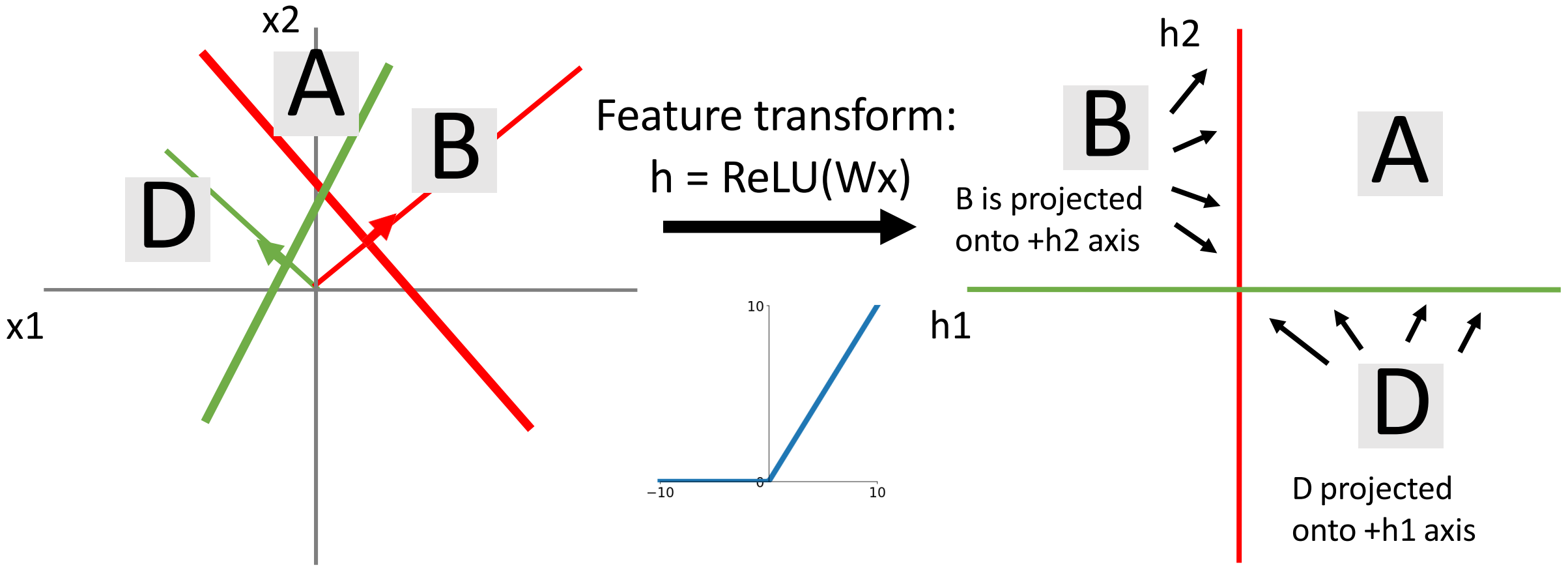
Feature transform:
h = ReLU(Wx)





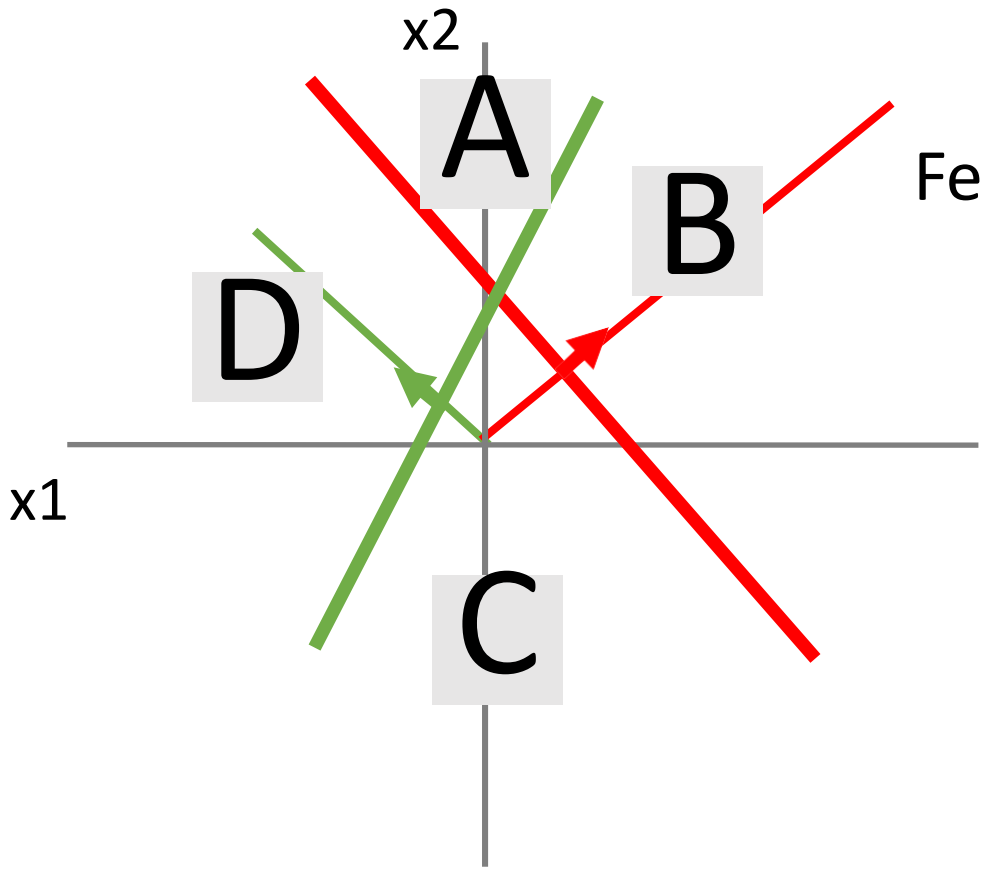
Feature Transform

Consider a neural net hidden layer:
h = ReLU(Wx) = max(0, Wx)
Where x, h are both 2-dimensional

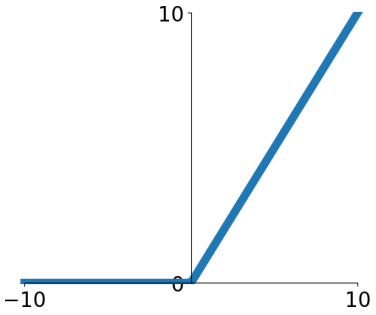


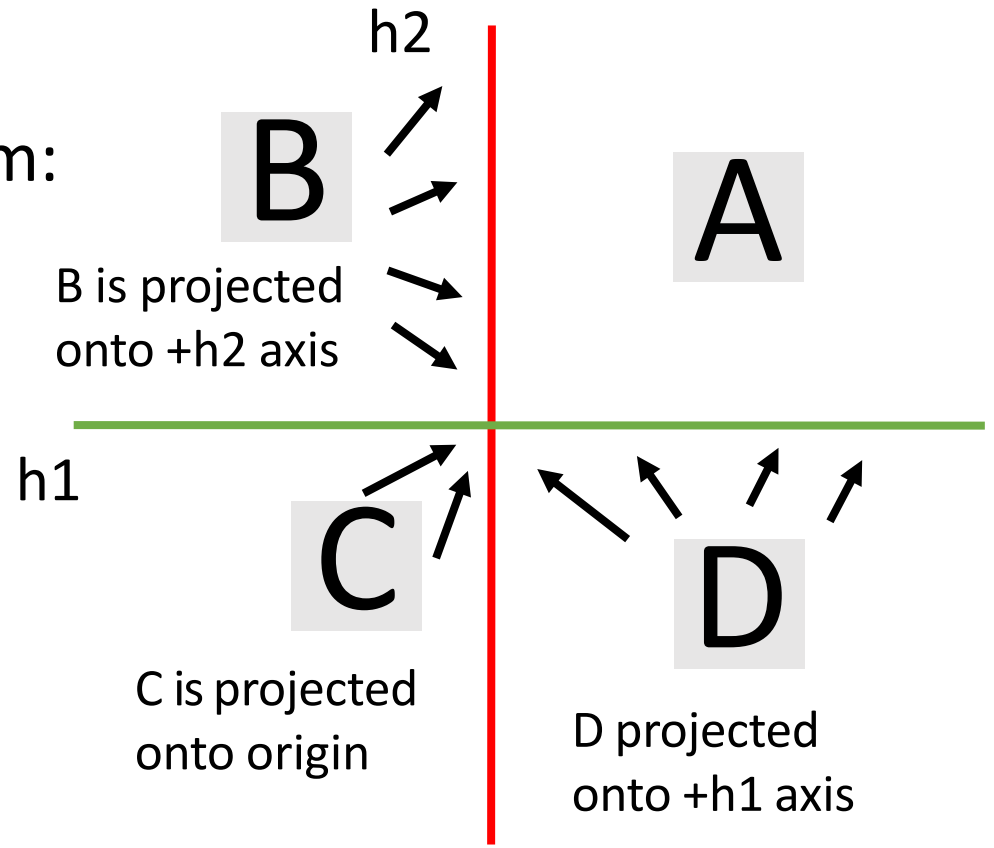
Feature Transform

Consider a neural net hidden layer:
h = ReLU(Wx) = max(0, Wx)
Where x, h are both 2-dimensional



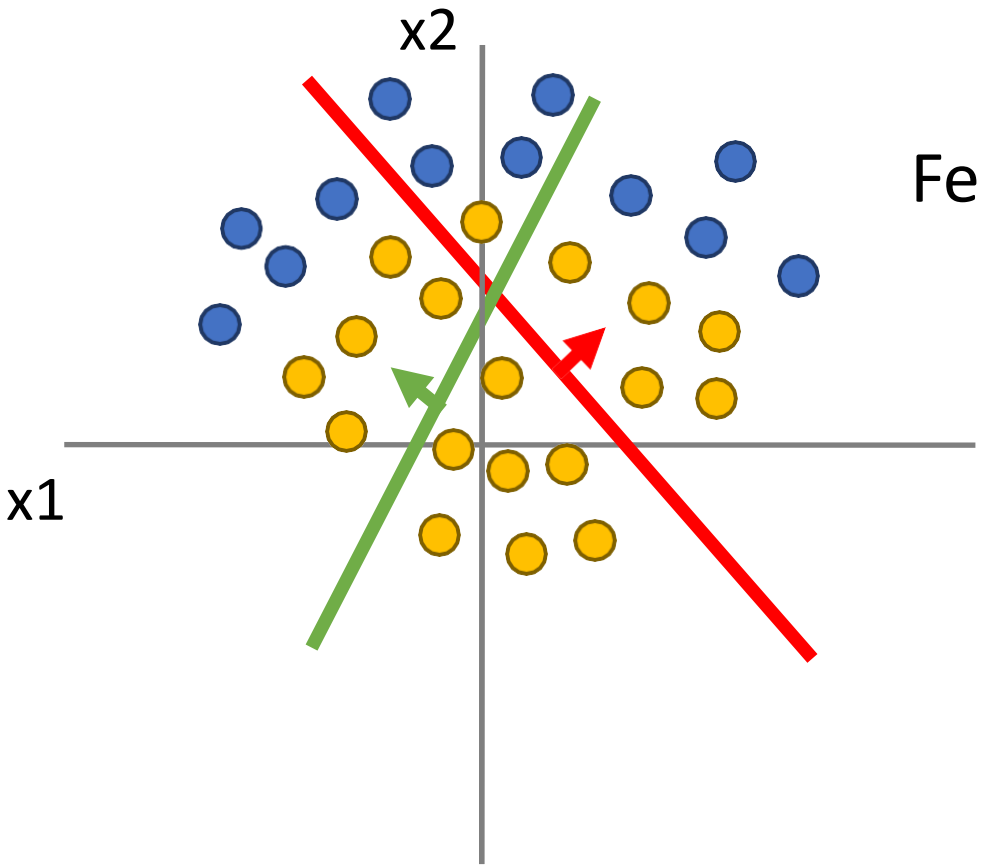
Feature transform:
h = ReLU(Wx)





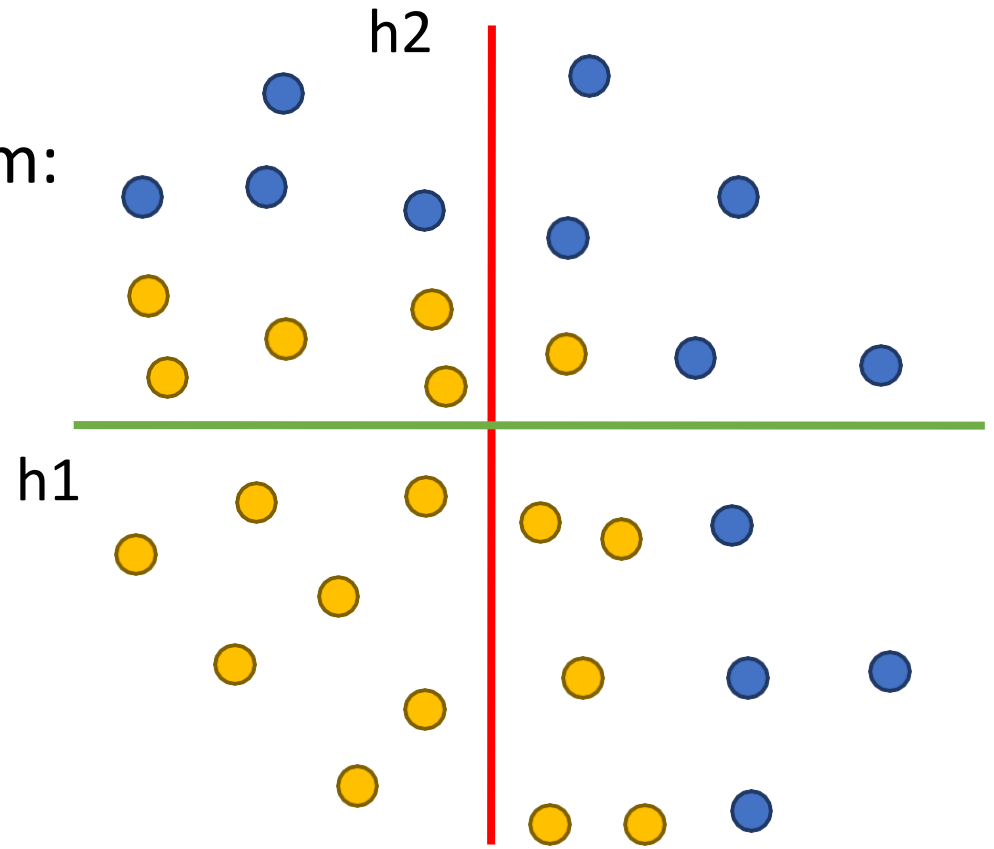
Feature Transform

Points not linearly separable in original space



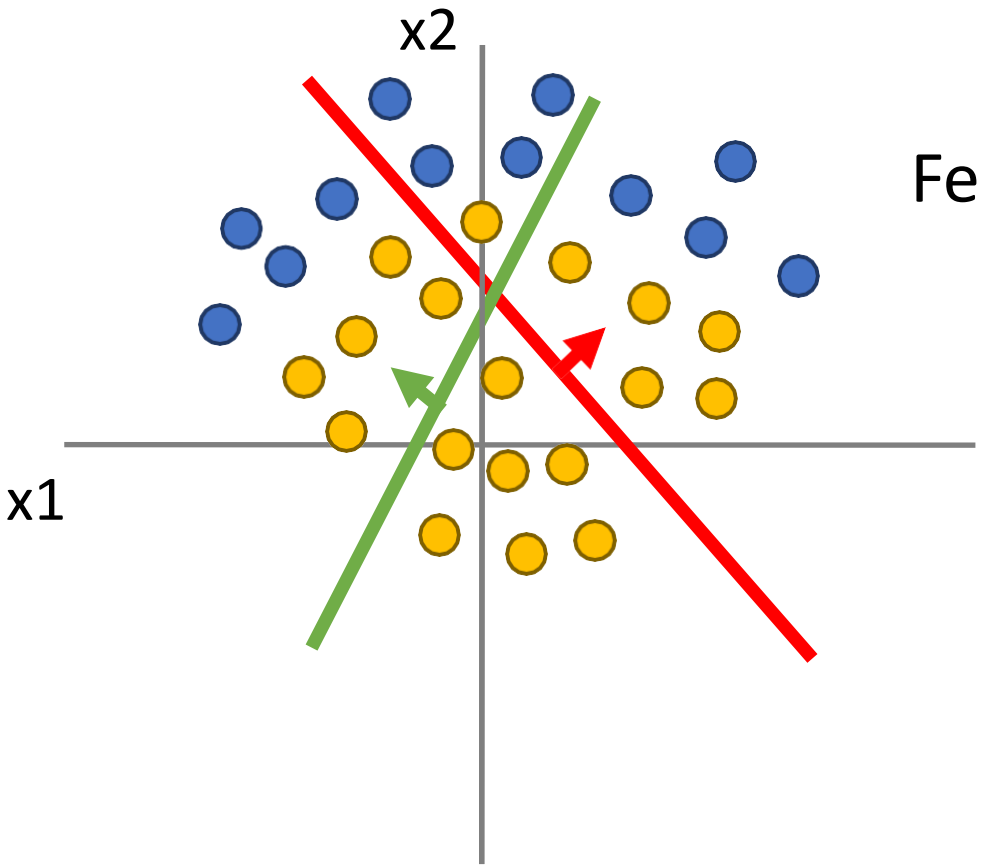
Consider a neural net hidden layer:
h = ReLU(Wx) = max(0, Wx)
Where x, h are both 2-dimensional

Feature transform:
h = Wx



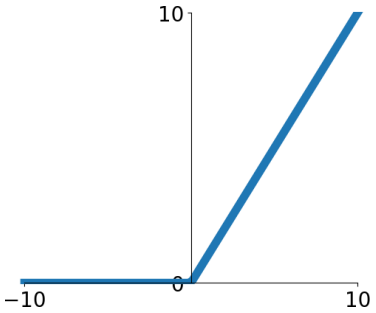
Feature Transform

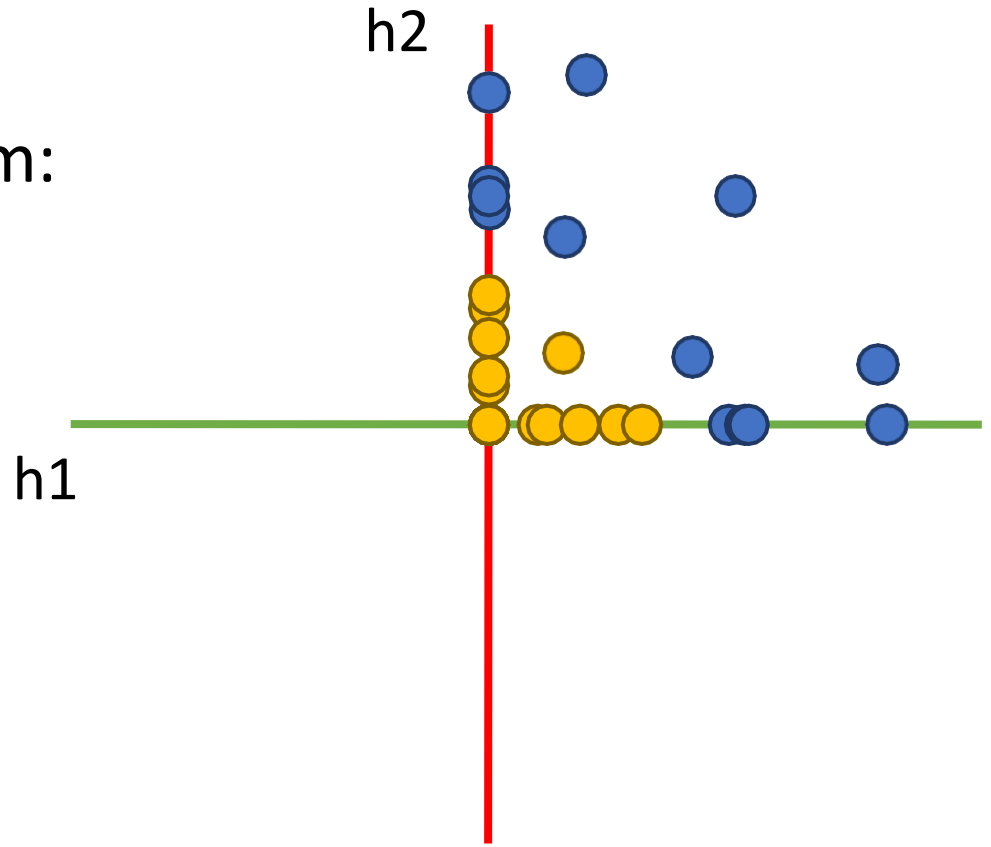
Points not linearly separable in original space



Consider a neural net hidden layer:
h = ReLU(Wx) = max(0, Wx)
Where x, h are both 2-dimensional

Feature transform:
h = ReLU(Wx)

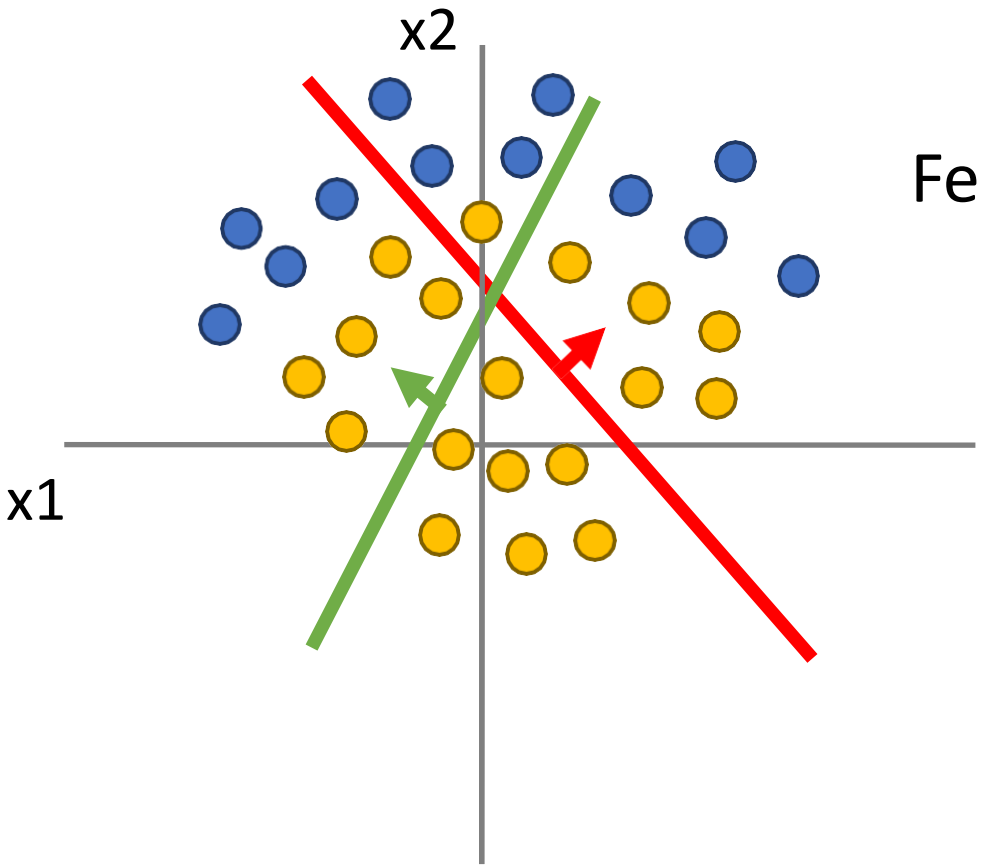




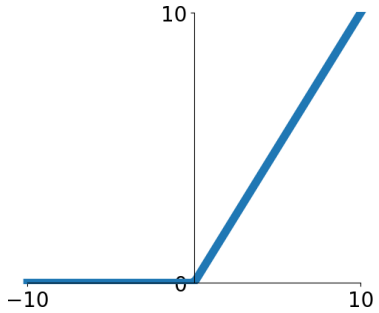
Feature Transform

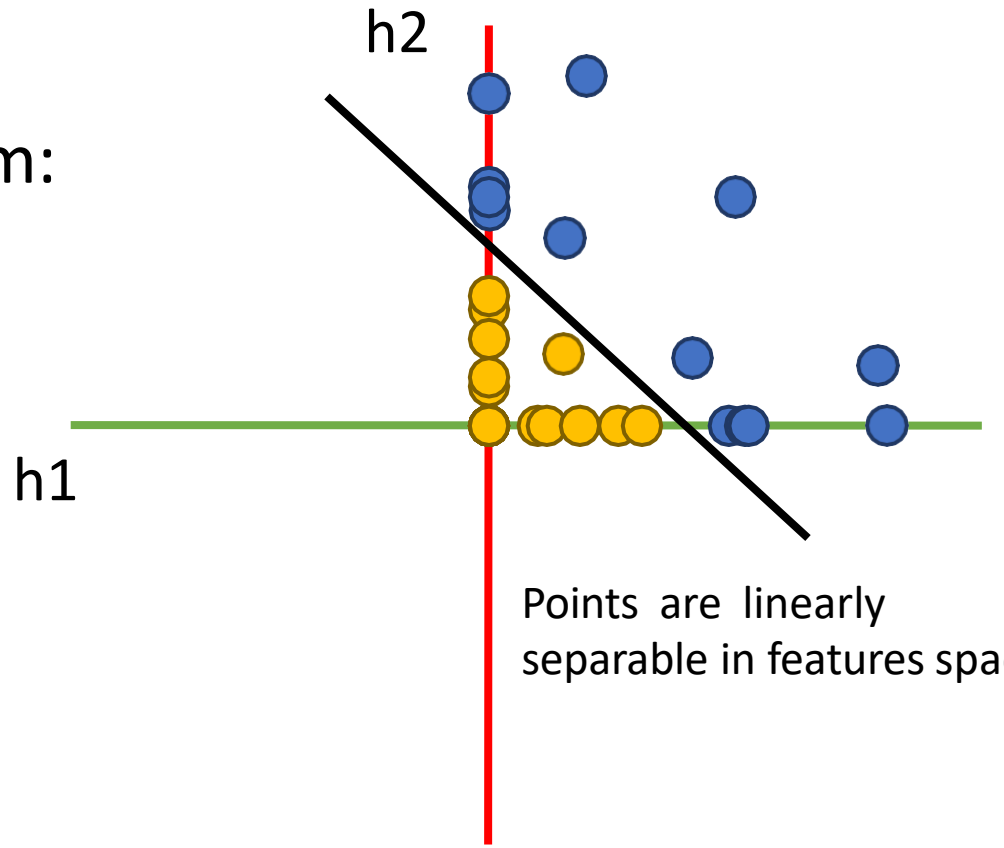
Consider a neural net hidden layer:
h = ReLU(Wx) = max(0, Wx)
Where x, h are both 2-dimensional

Points not linearly separable in original space



Feature transform:
h = ReLU(Wx)

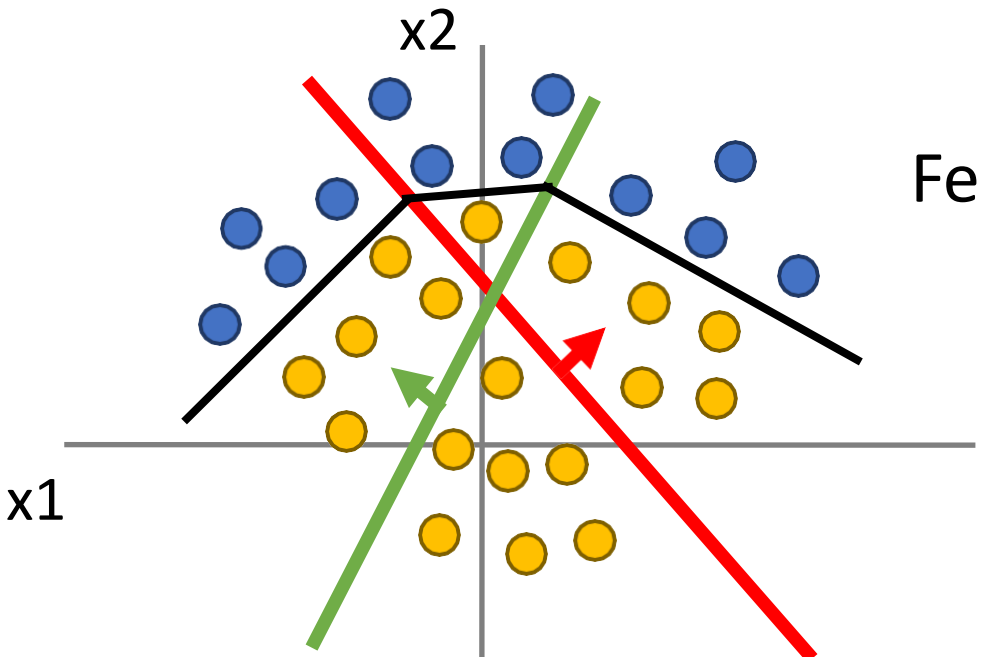




Points are linearly separable in features space!

Feature Transform

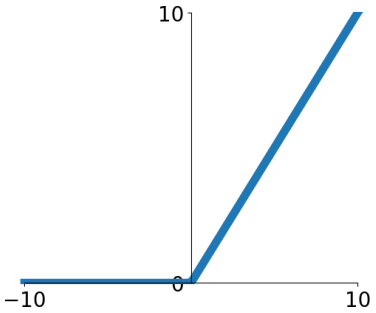
Points not linearly separable in original space

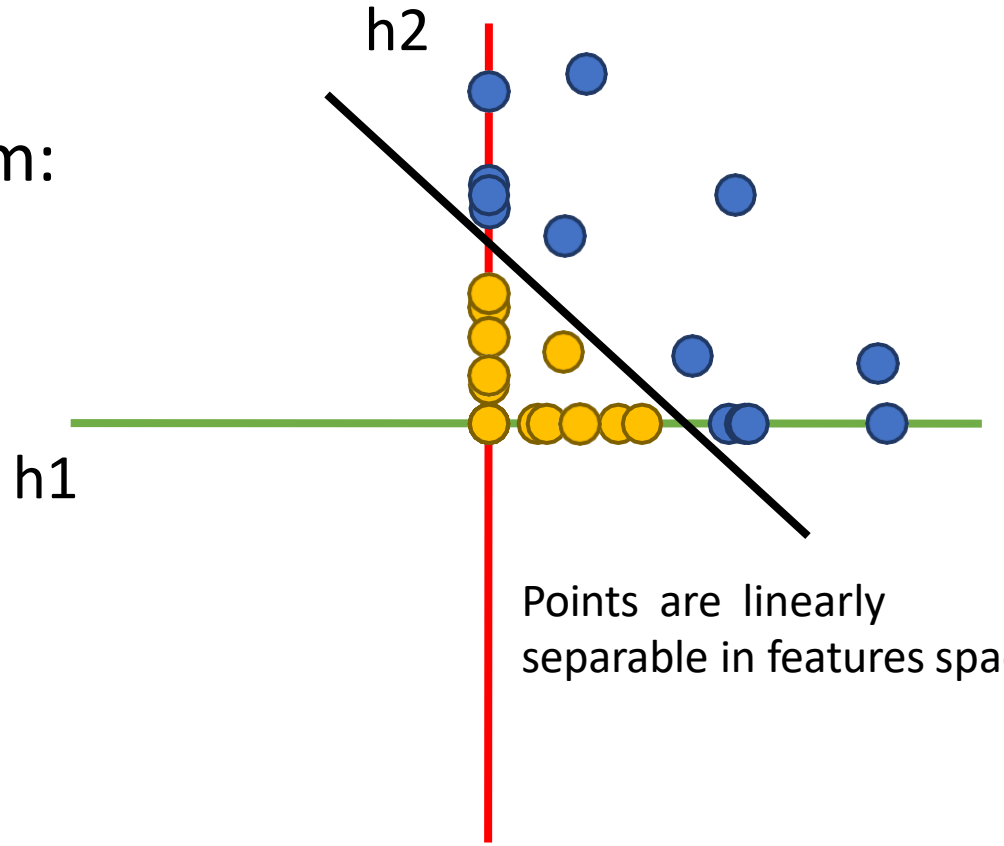


Linear classifier in feature space gives nonlinear classifier in original space

Consider a neural net hidden layer:
h = ReLU(Wx) = max(0, Wx)
Where x, h are both 2-dimensional

Feature transform:
h = ReLU(Wx)

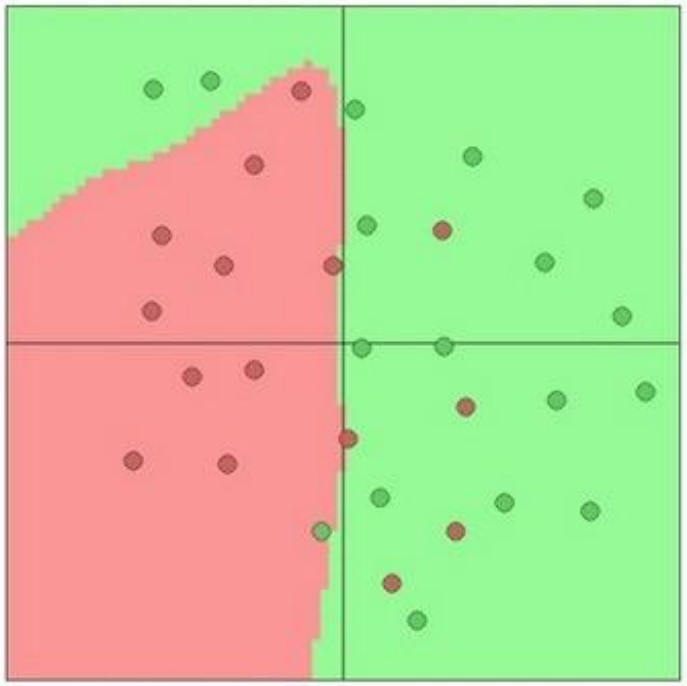




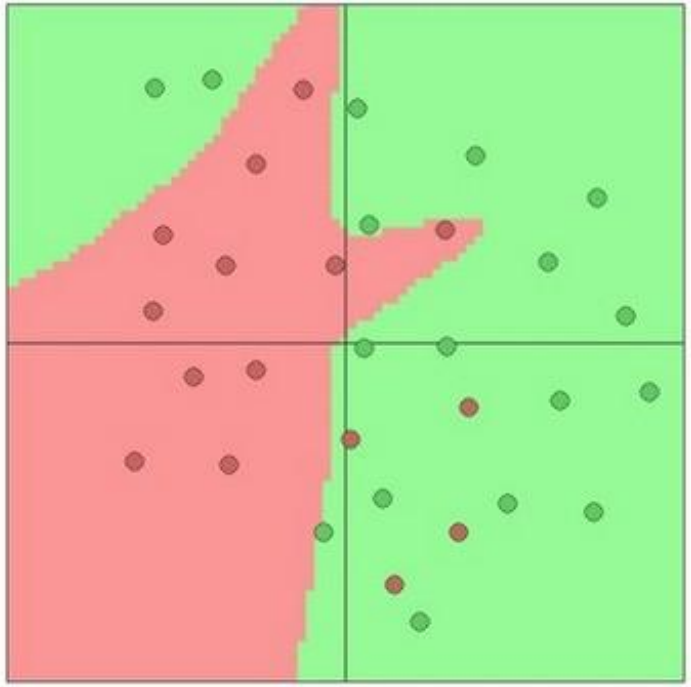
Points are linearly separable in features space!

Setting the number of layers and their sizes

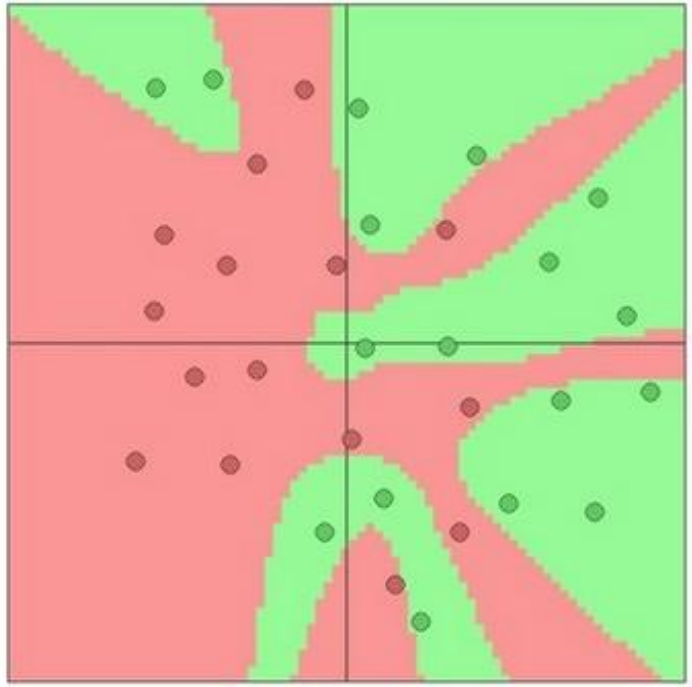
3 hidden units



6 hidden units



20 hidden units



More hidden units = more capacity

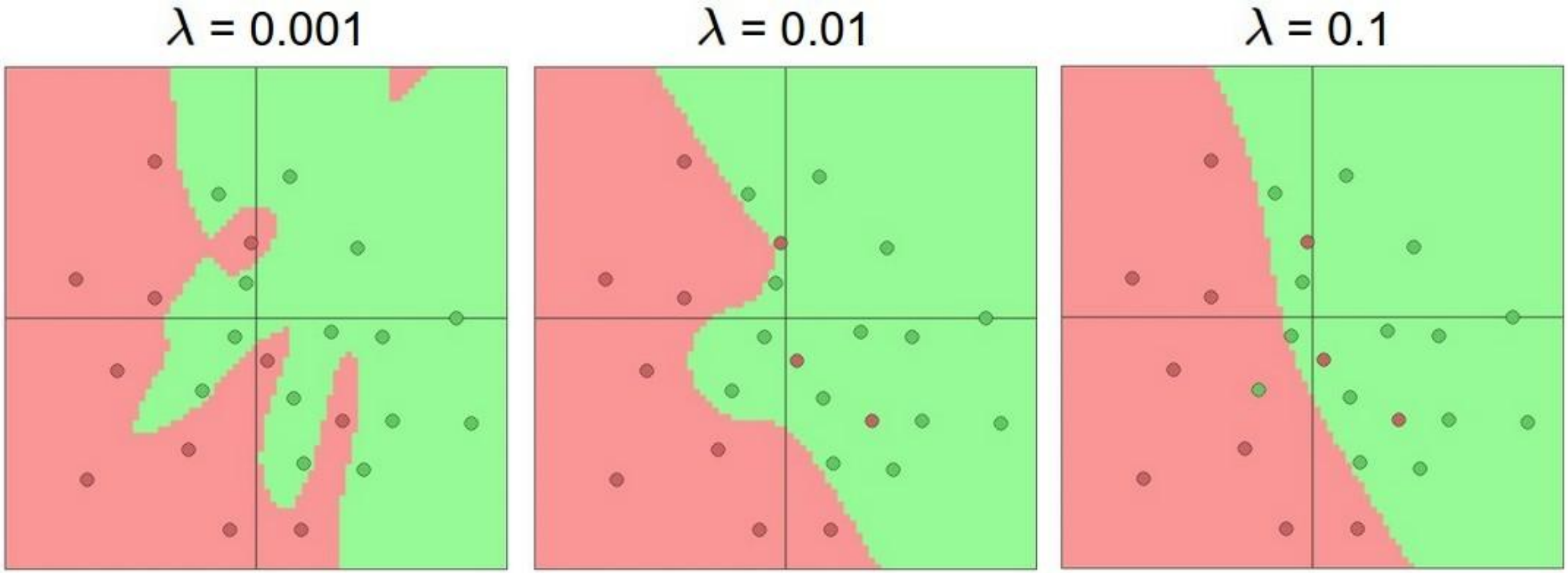
Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Regularization with constant number of layers

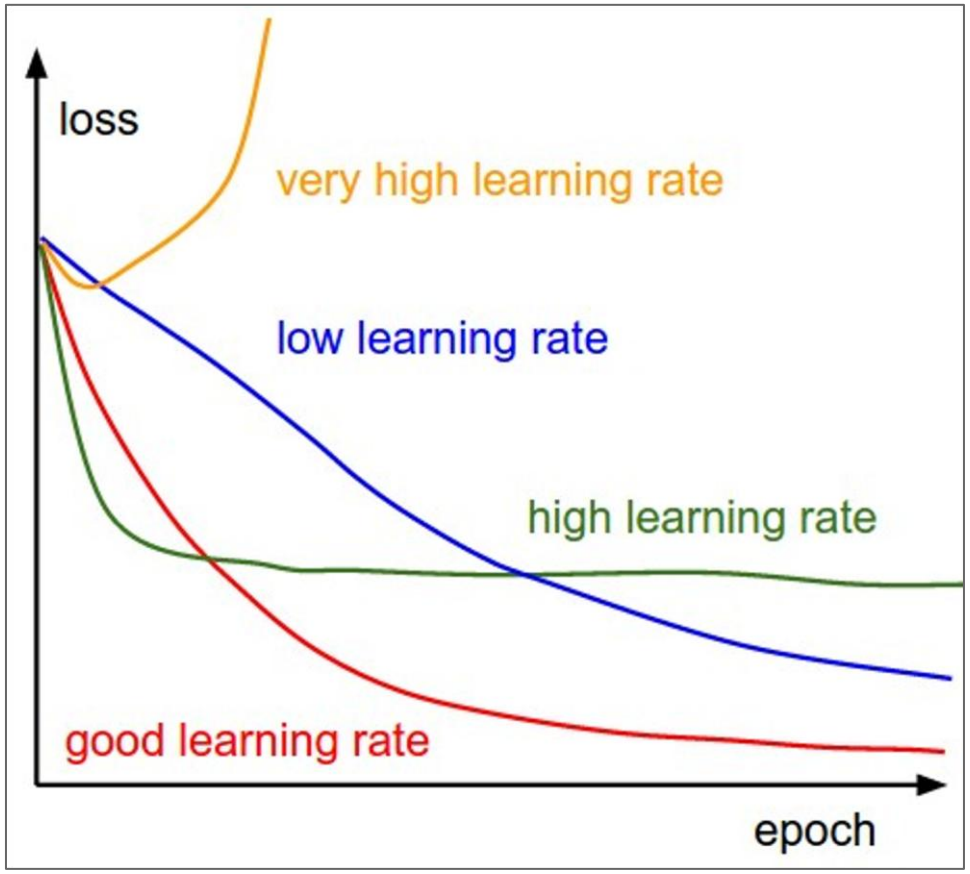


Overview

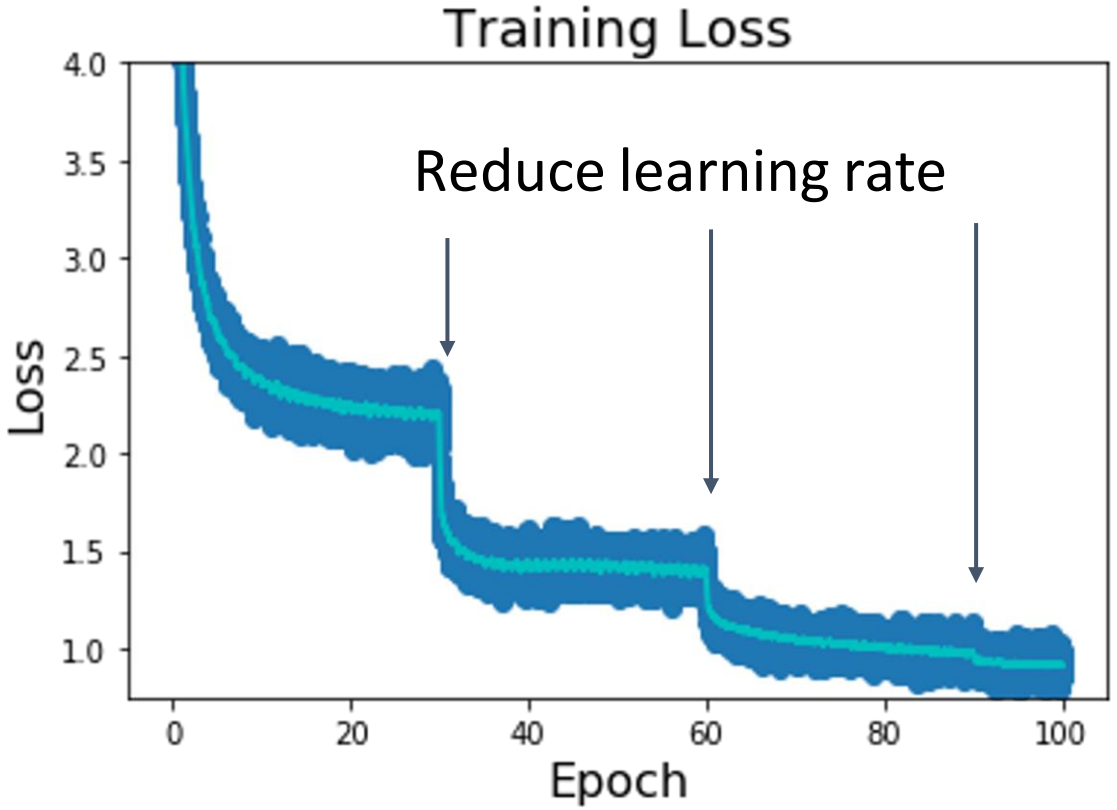
1. **One time setup**
   Activation functions, data preprocessing, weight initialization, regularization
2. **Training dynamics**
   Learning rate schedules; hyperparameter optimization
3. **After training**
   Model ensembles, transfer learning, large-batch training

Learning Rate Schedules

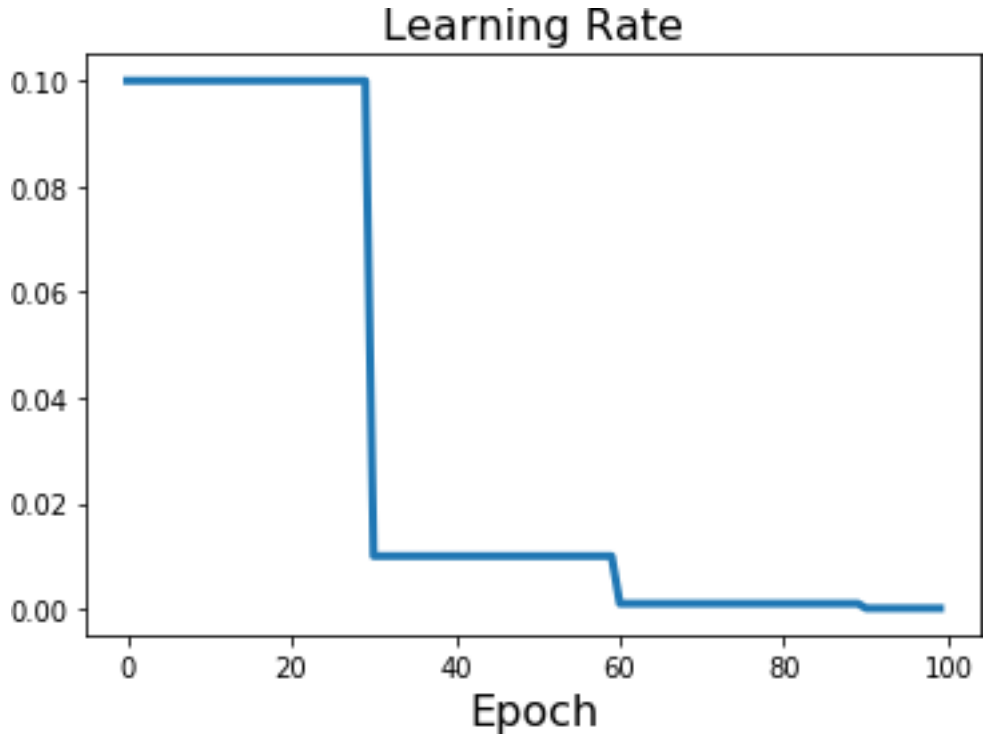
SGD, SGD+Momentum, Adagrad, RMSProp, Adam
all have **learning rate** as a hyperparameter.



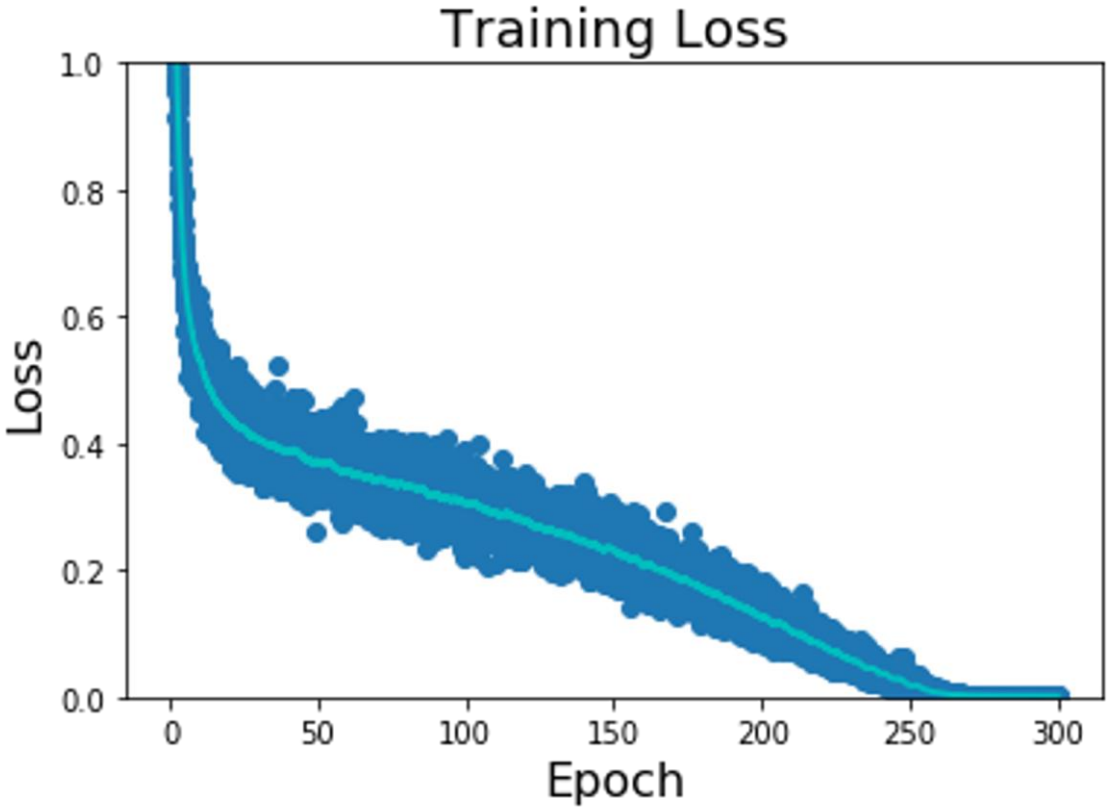
Learning Rate Decay: Step



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

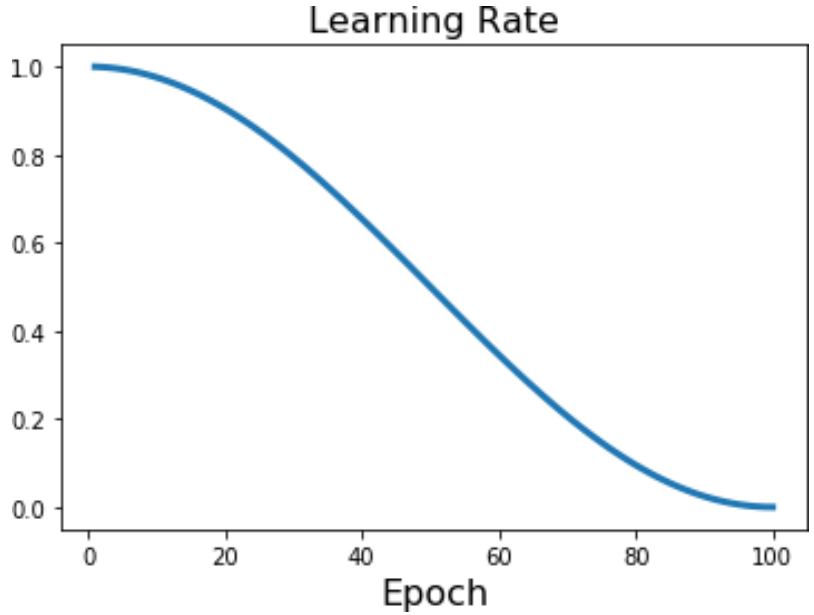


Learning Rate Decay: Cosine



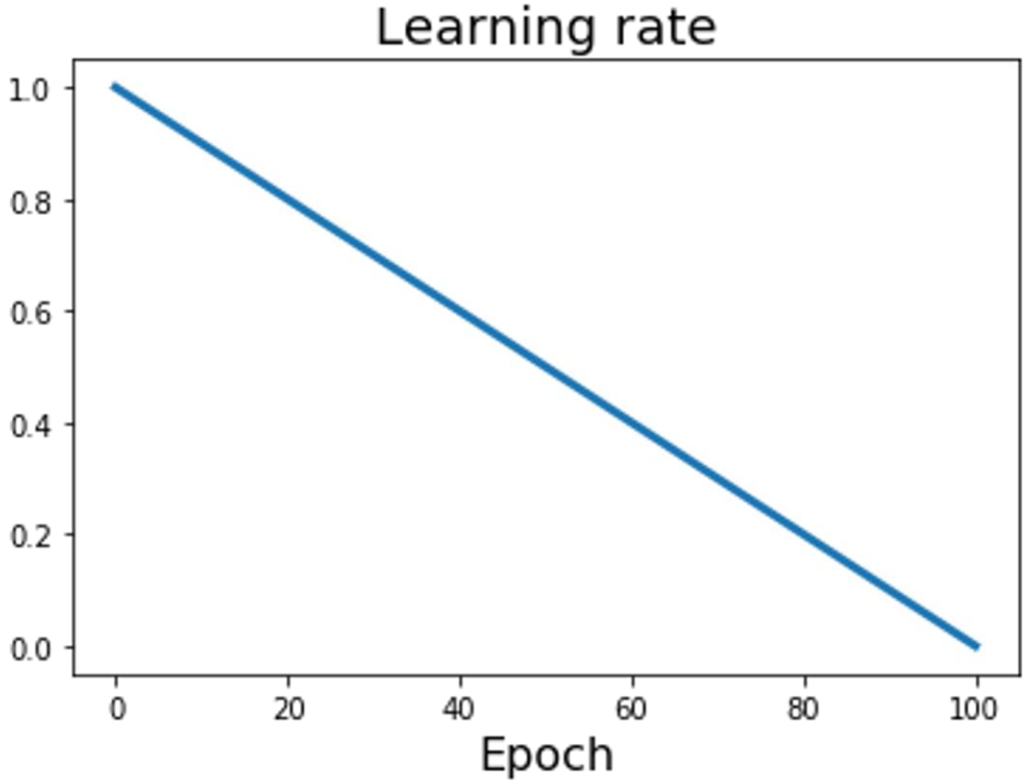
Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Cosine: $$\alpha_t = \frac{1}{2}\alpha_0\left(1 + \cos\left(\frac{t\pi}{T}\right)\right)$$



Loshchilov and Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts", ICLR 2017
Radford et al, "Improving Language Understanding by Generative Pre-Training", 2018
Feichtenhofer et al, "SlowFast Networks for Video Recognition", ICCV 2019
Radosavovic et al, "On Network Design Spaces for Visual Recognition", ICCV 2019
Child at al, "Generating Long Sequences with Sparse Transformers", arXiv 2019

Learning Rate Decay: Linear



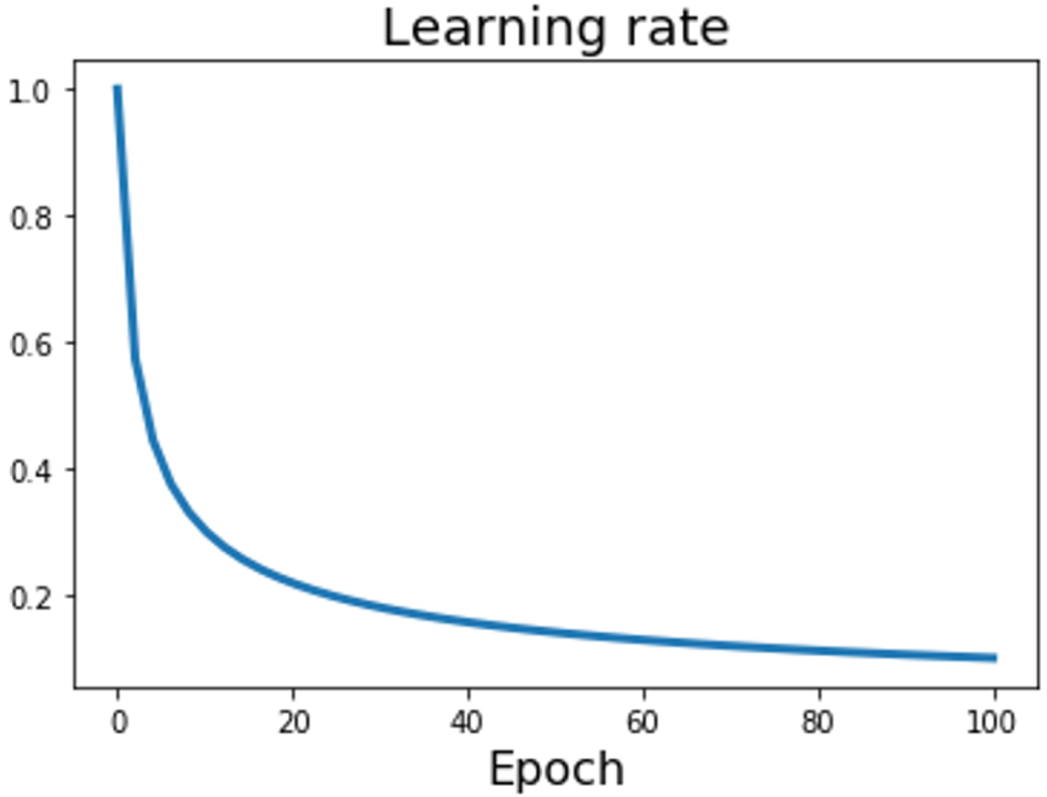
Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Cosine: $$\alpha_t = \frac{1}{2}\alpha_0\left(1 + \cos\left(\frac{t\pi}{T}\right)\right)$$

Linear: $$\alpha_t = \alpha_0\left(1 - \frac{t}{T}\right)$$

Devlin et al, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, NAACL 2018
Liu et al, “RoBERTa: A Robustly Optimized BERT Pretraining Approach”, 2019
Yang et al, “XLNet: Generalized Autoregressive Pretraining for Language Understanding”, NeurIPS 2019

Learning Rate Decay: Inverse Sqrt



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

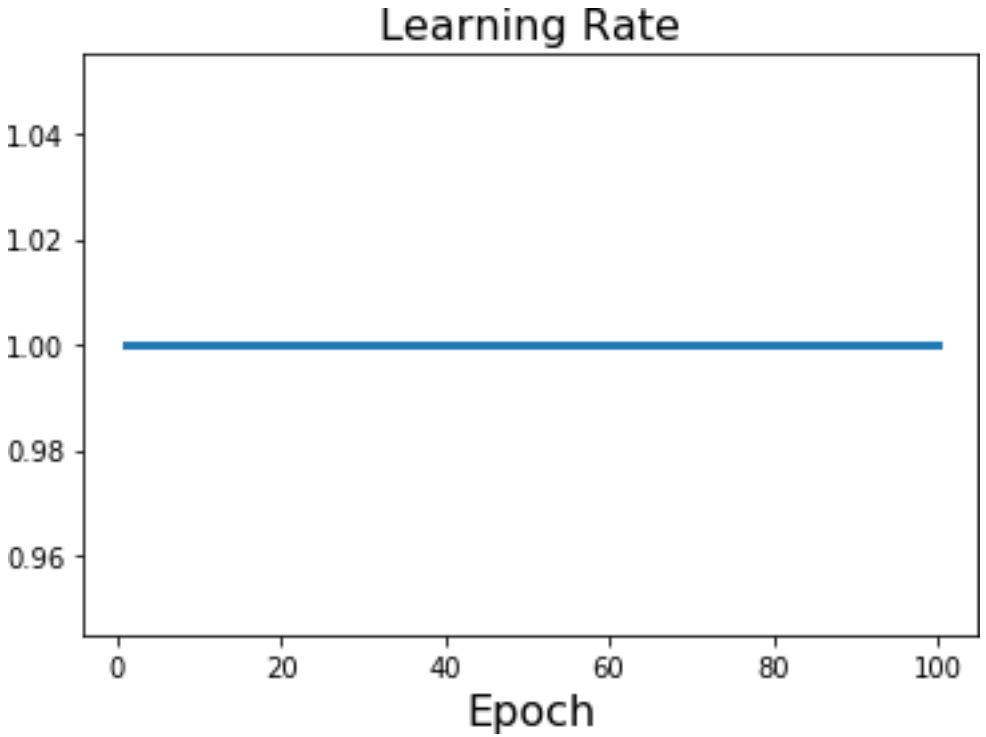
Cosine: $\alpha_t = \frac{1}{2}\alpha_0\left(1 + \cos\left(\frac{t\pi}{T}\right)\right)$

Linear: $\alpha_t = \alpha_0\left(1 - \frac{t}{T}\right)$

Inverse sqrt: $\alpha_t = \alpha_0/\sqrt{t}$

Vaswani et al, “Attention is all you need”, NIPS 2017

Learning Rate Decay: Constant



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Cosine: $\alpha_t = \frac{1}{2}\alpha_0\left(1 + \cos\left(\frac{t\pi}{T}\right)\right)$

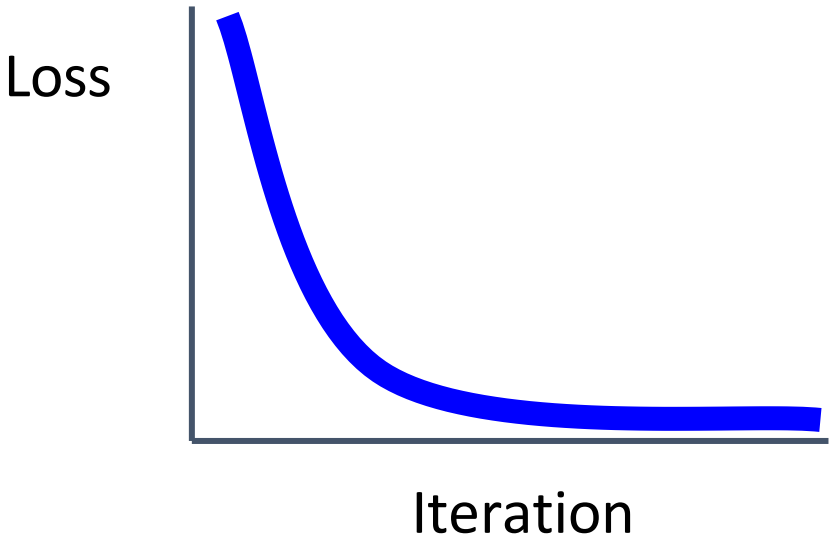
Linear: $\alpha_t = \alpha_0\left(1 - \frac{t}{T}\right)$

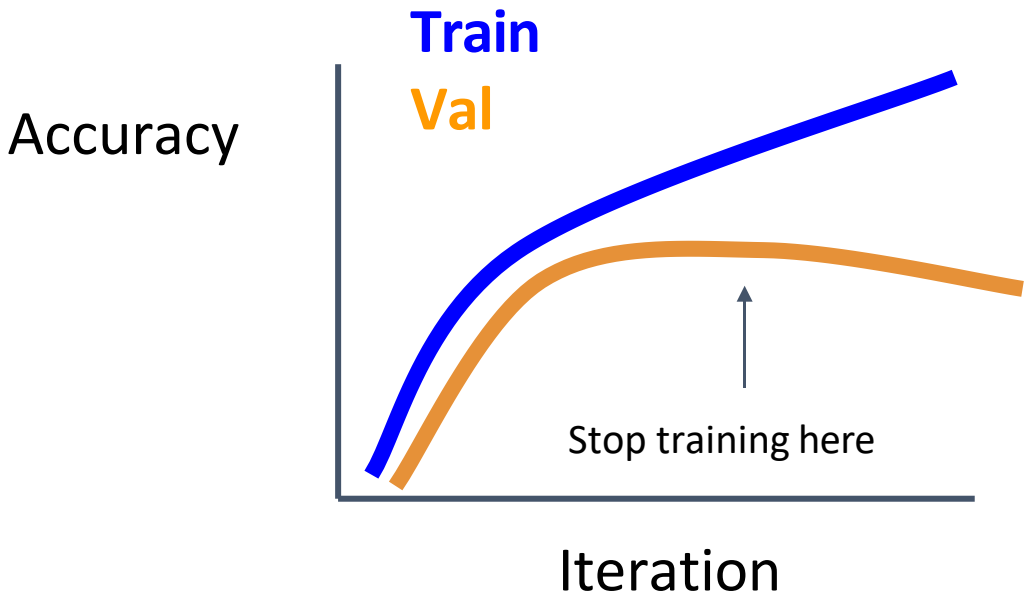
Inverse sqrt: $\alpha_t = \alpha_0/\sqrt{t}$

Constant: $\alpha_t = \alpha_0$

Brock et al, “Large Scale GAN Training for High Fidelity Natural Image Synthesis”, ICLR 2019
Donahue and Simonyan, “Large Scale Adversarial Representation Learning”, NeurIPS 2019

How long to train? Early Stopping





Stop training the model when accuracy on the validation set decreases
Or train for a long time, but always keep track of the model snapshot that worked **best** on val.

Choosing Hyperparameters

Choosing Hyperparameters: Grid Search

Choose several values for each hyperparameter
(Often space choices log-linearly)

Example:
Weight decay: [$1x10^{-4}$, $1x10^{-3}$, $1x10^{-2}$, $1x10^{-1}$]
Learning rate: [$1x10^{-4}$, $1x10^{-3}$, $1x10^{-2}$, $1x10^{-1}$]

Evaluate all possible choices on this
hyperparameter grid

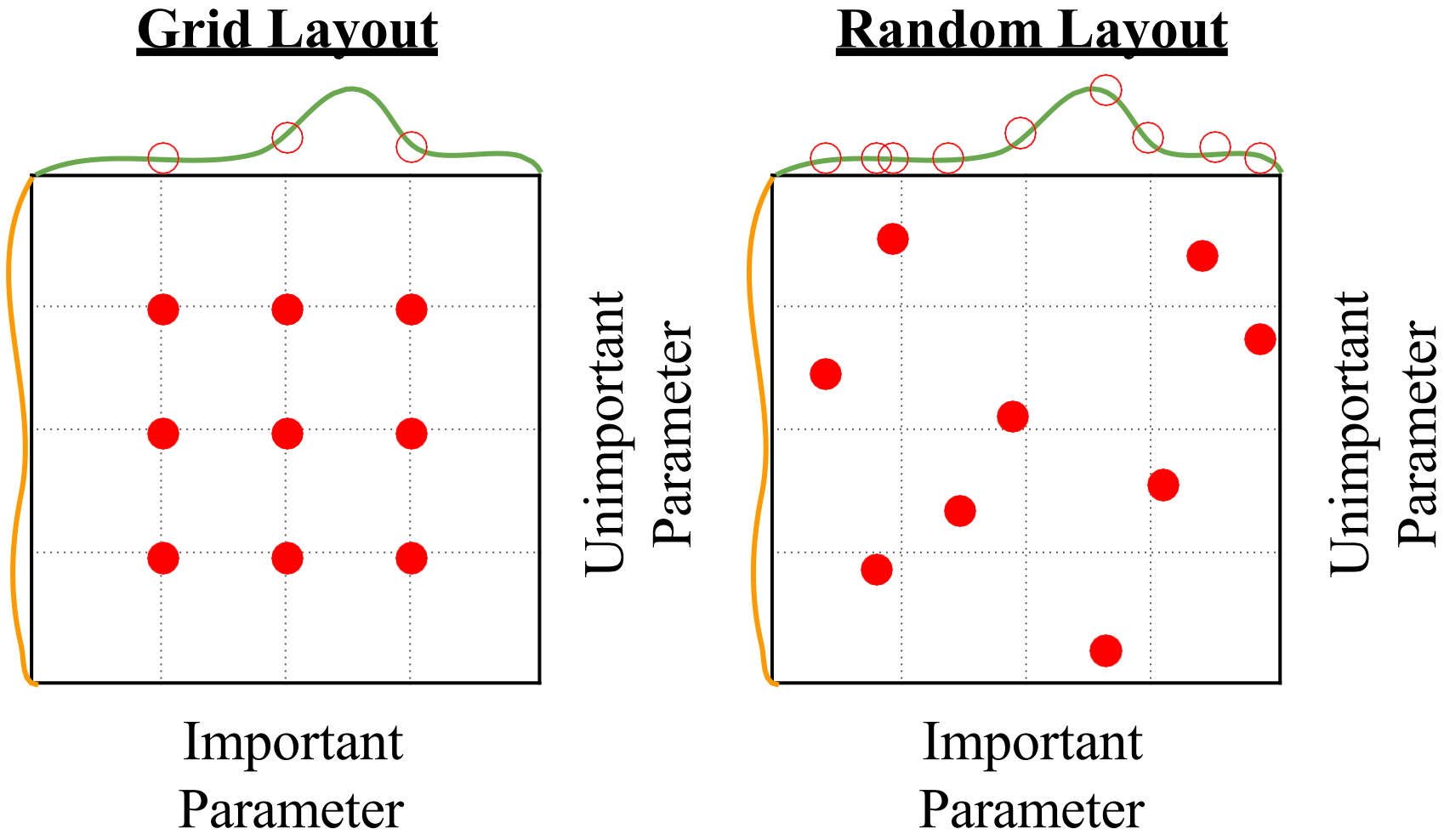
Choosing Hyperparameters: Random Search

Choose several values for each hyperparameter
(Often space choices log-linearly)

Example:
Weight decay: log-uniform on $[1x10^{-4}, 1x10^{-1}]$
Learning rate: log-uniform on $[1x10^{-4}, 1x10^{-1}]$

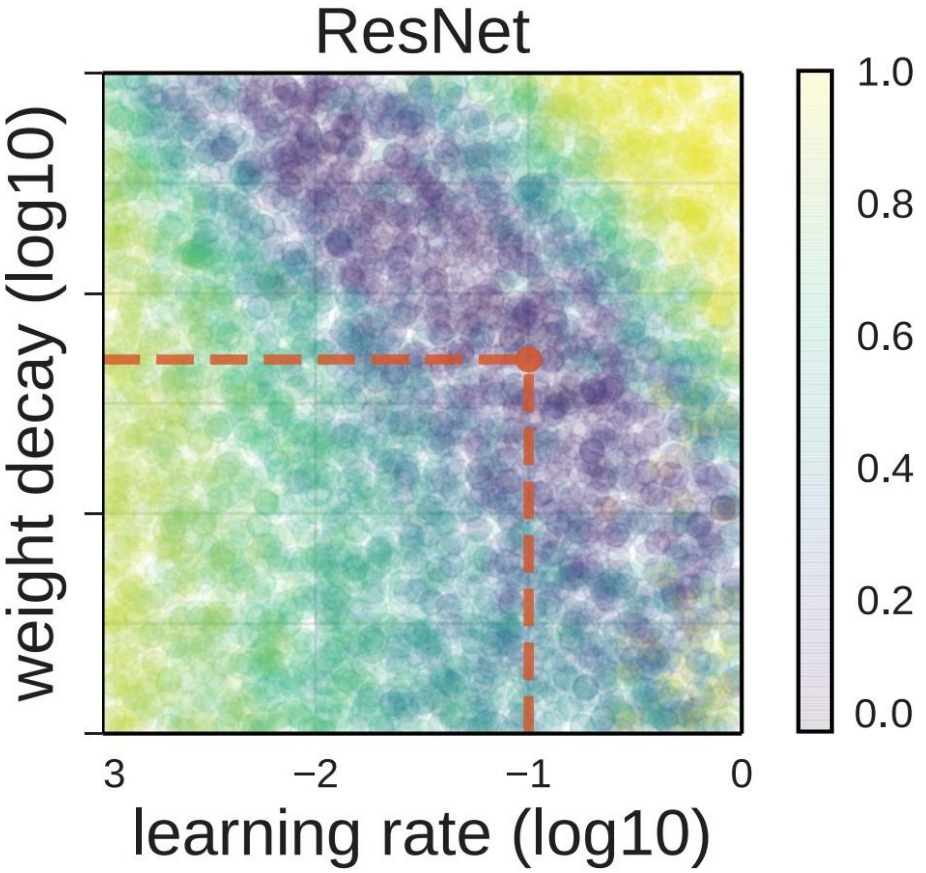
Run many different trials

Hyperparameters: Random vs Grid Search



Bergstra and Bengio, “Random Search for Hyper-Parameter Optimization”, JMLR 2012

Choosing Hyperparameters: Random Search



Radosavovic et al, “On Network Design Spaces for Visual Recognition”, ICCV 2019

Choosing Hyperparameters

(without significant amount of compute)

Choosing Hyperparameters

Step 1: Check initial loss

Turn off weight decay, sanity check loss at initialization
e.g. -log(1/C) for softmax with C classes

Choosing Hyperparameters

Step 1: Check initial loss
Step 2: Overfit a small sample

Try to train to 100% training accuracy on a small sample of training data (~5-10 minibatches); fiddle with architecture, learning rate, weight initialization. Turn off regularization.

Loss not going down? LR too low, bad initialization
Loss explodes to Inf or NaN? LR too high, bad initialization

Choosing Hyperparameters

Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down

Use the architecture from the previous step, use all training data, turn on small weight decay, find a learning rate that makes the loss drop significantly within ~100 iterations

Good learning rates to try: 1e-1, 1e-2, 1e-3, 1e-4

Choosing Hyperparameters

Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down
Step 4: Coarse grid, train for ~1-5 epochs

Choose a few values of learning rate and weight decay around what worked from Step 3, train a few models for ~1-5 epochs.

Good weight decay to try: 1e-4, 1e-5, 0

Choosing Hyperparameters

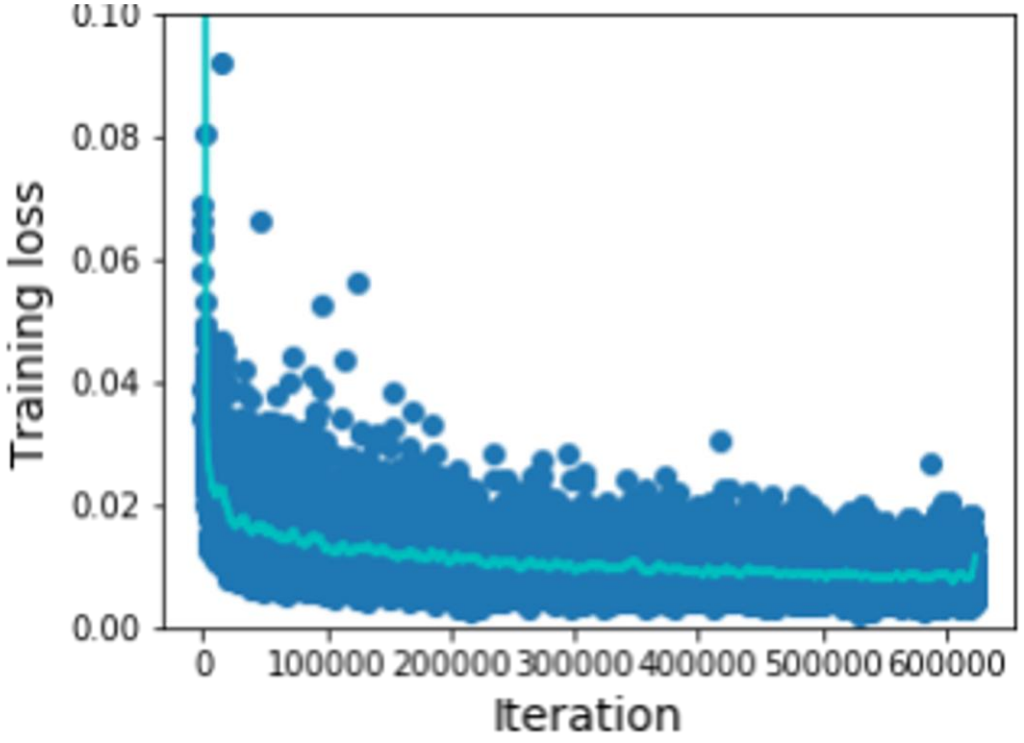
Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down
Step 4: Coarse grid, train for ~1-5 epochs
Step 5: Refine grid, train longer

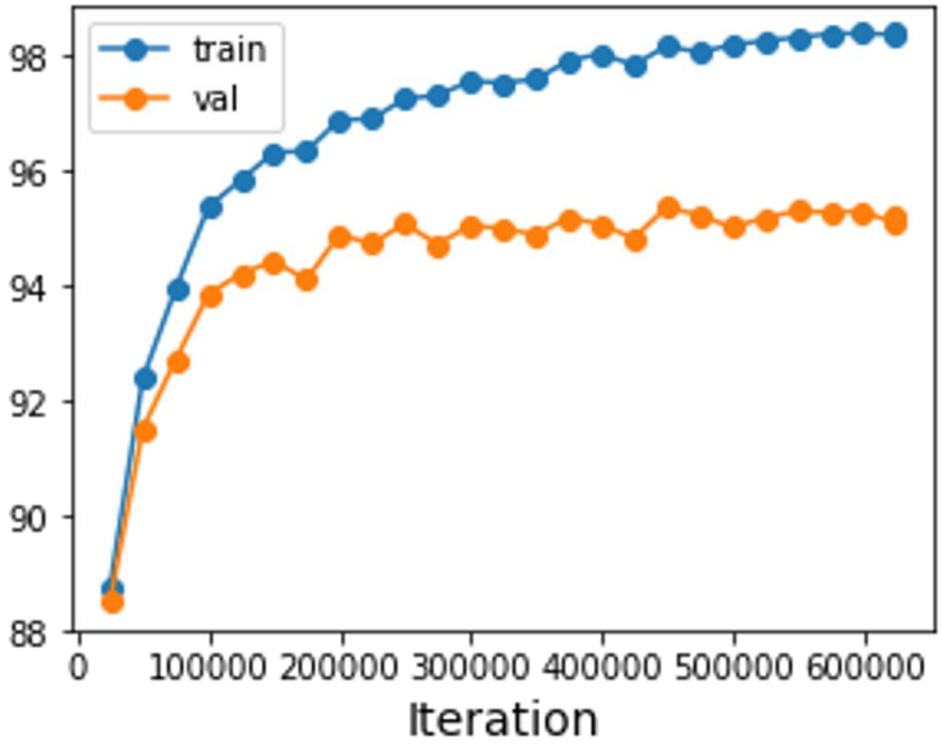
Pick best models from Step 4, train them for longer (~10-20 epochs) without learning rate decay

Choosing Hyperparameters

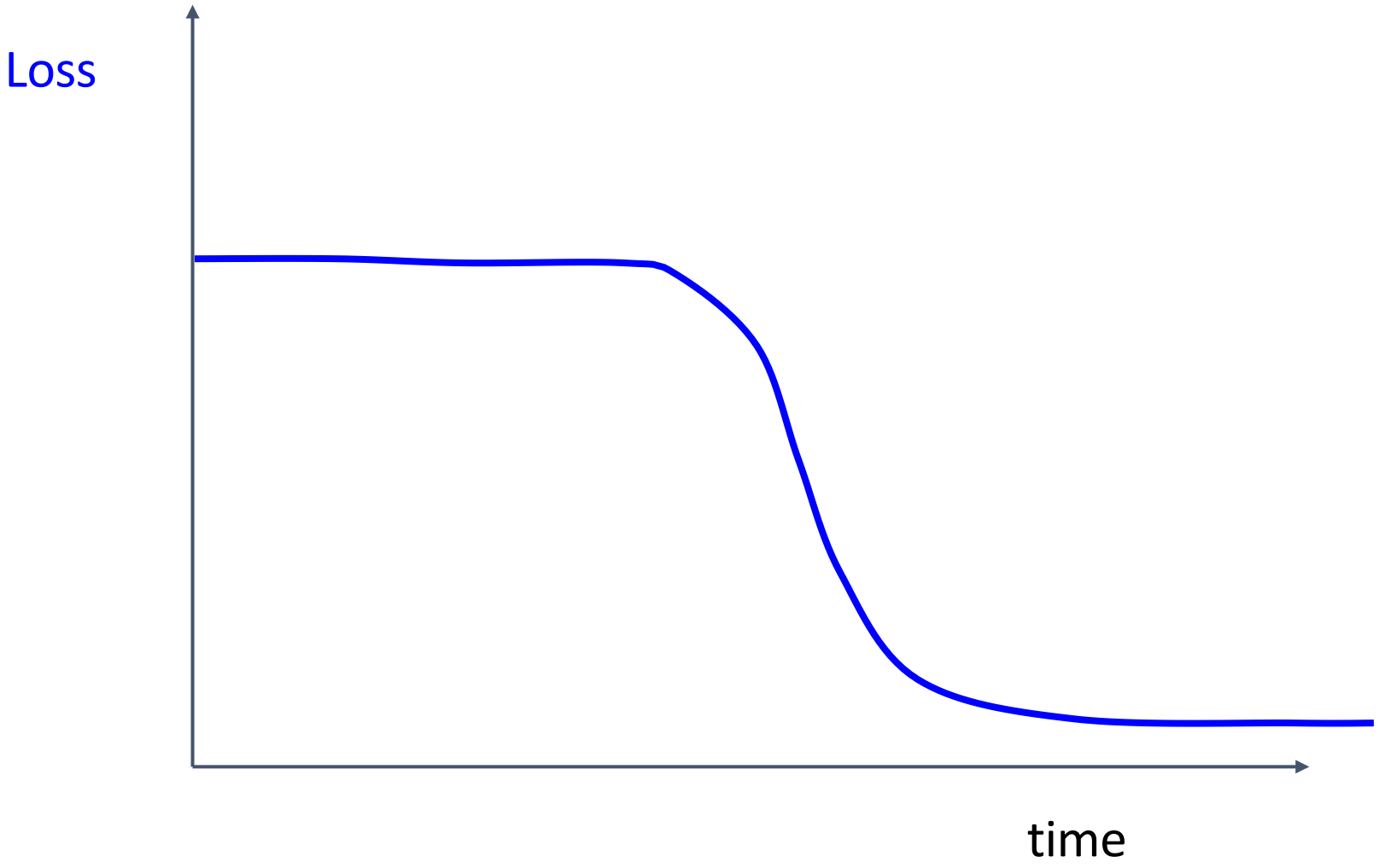
Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down
Step 4: Coarse grid, train for ~1-5 epochs
Step 5: Refine grid, train longer
Step 6: Look at learning curves

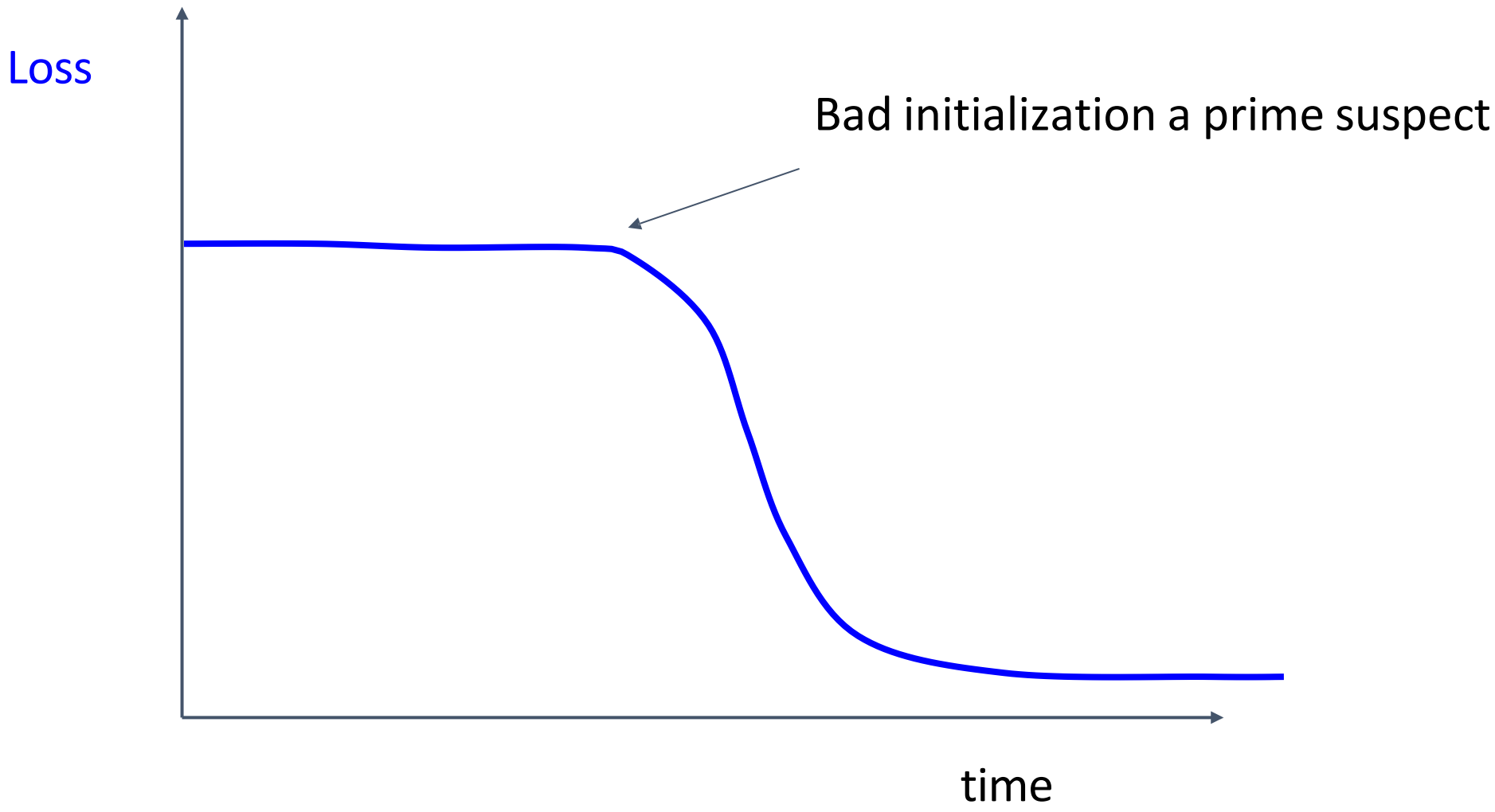
Look at Learning Curves

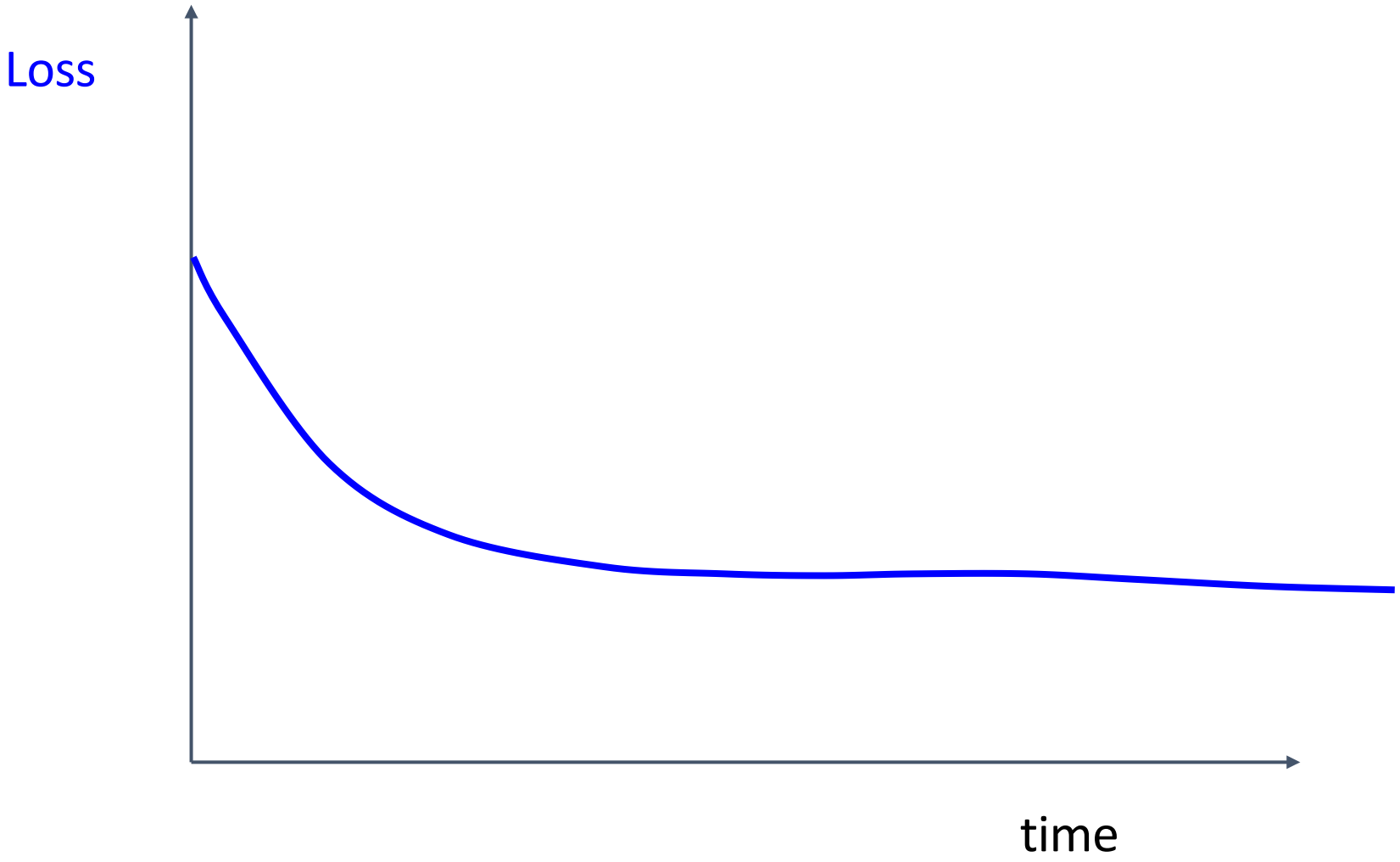


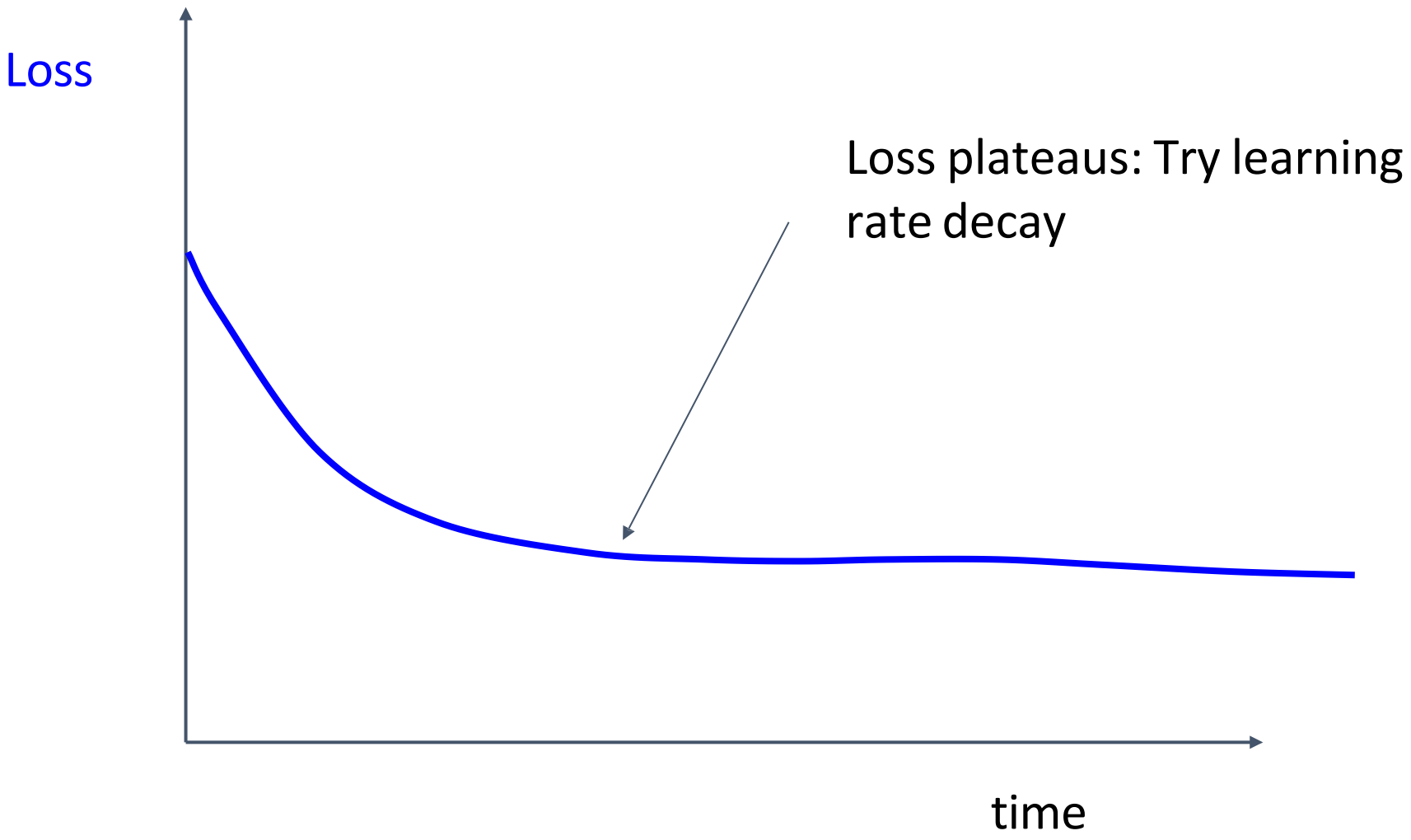


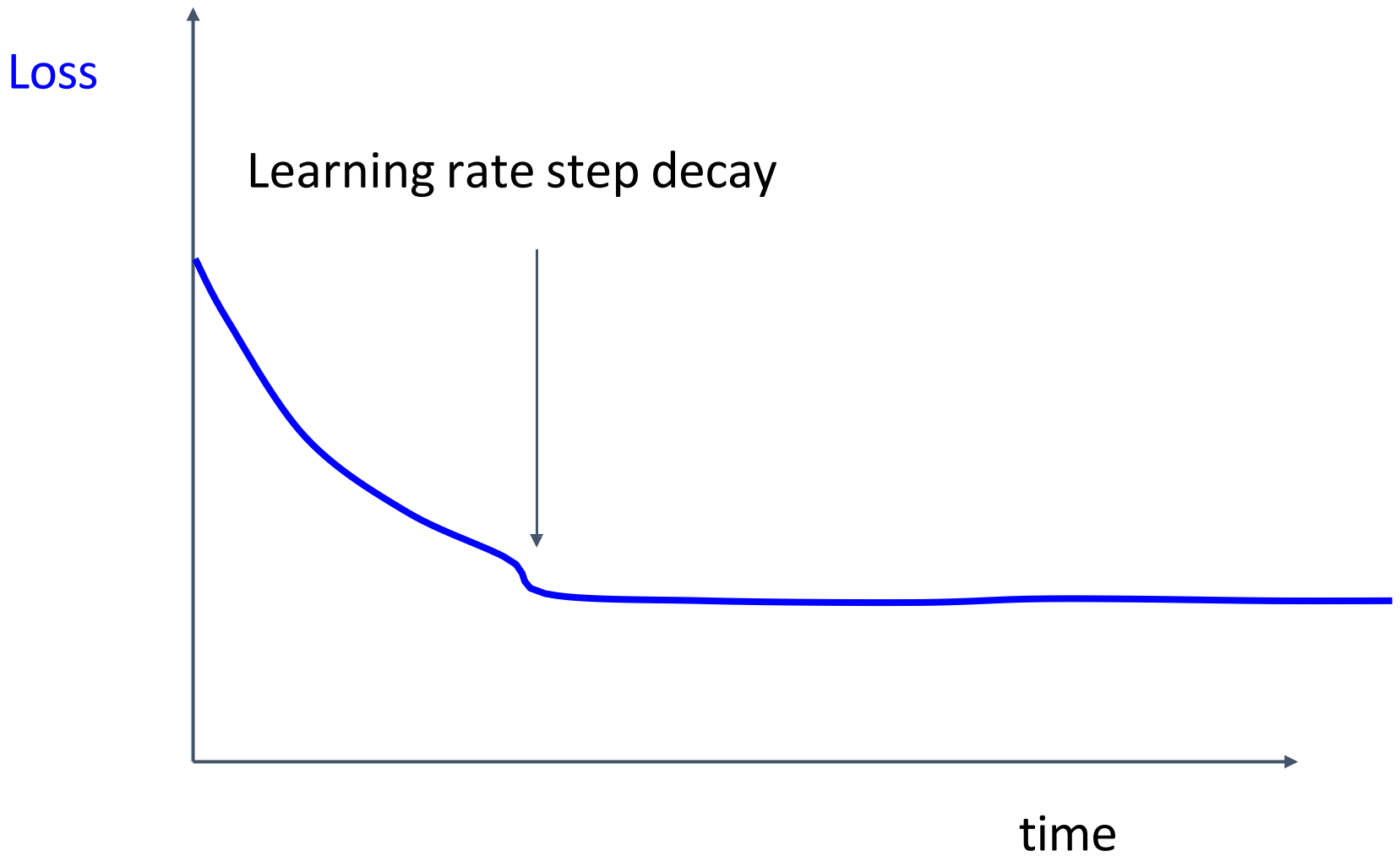
Losses may be noisy, use a scatter plot and also plot moving average to see trends better

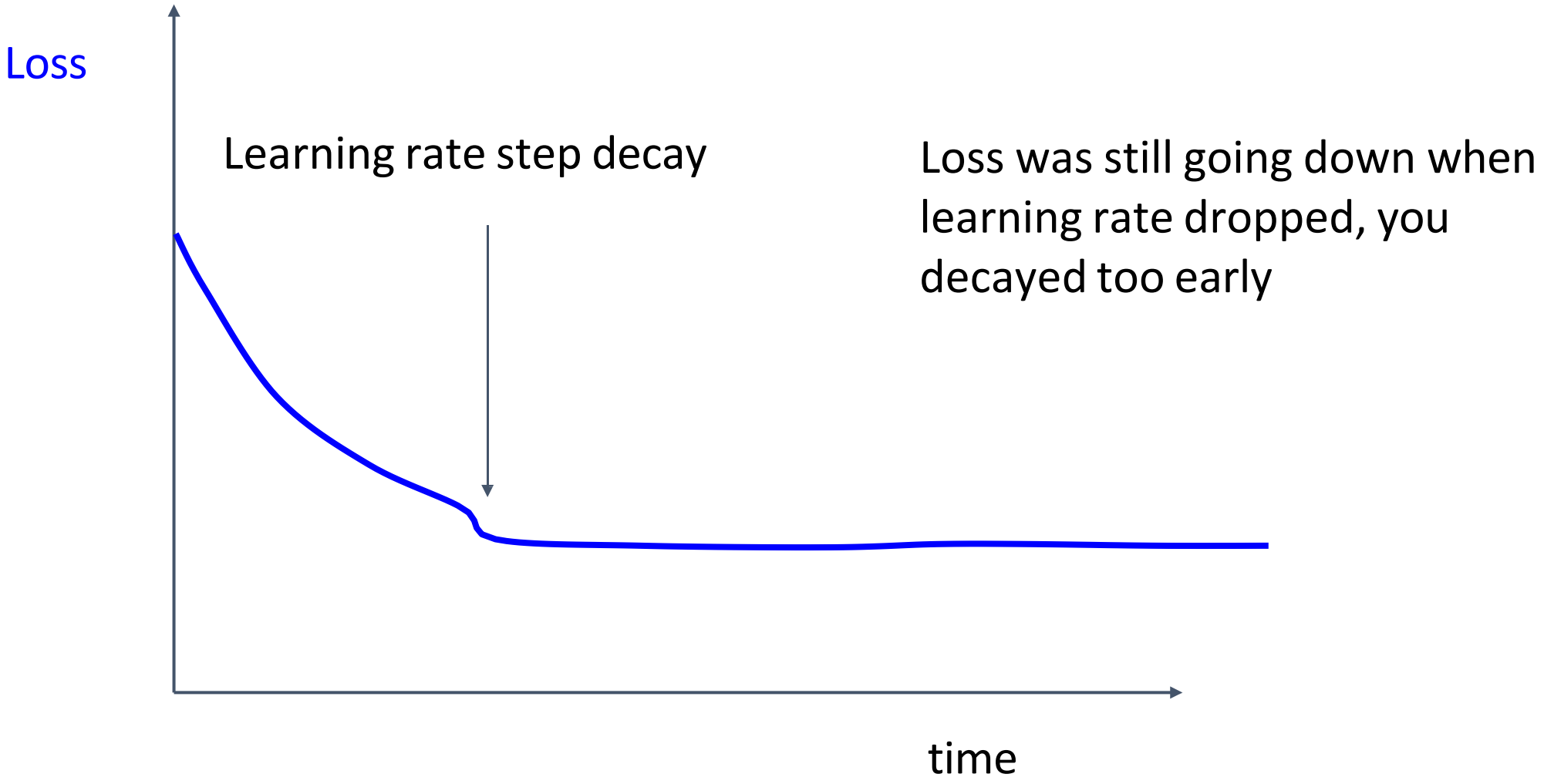

Loss
time

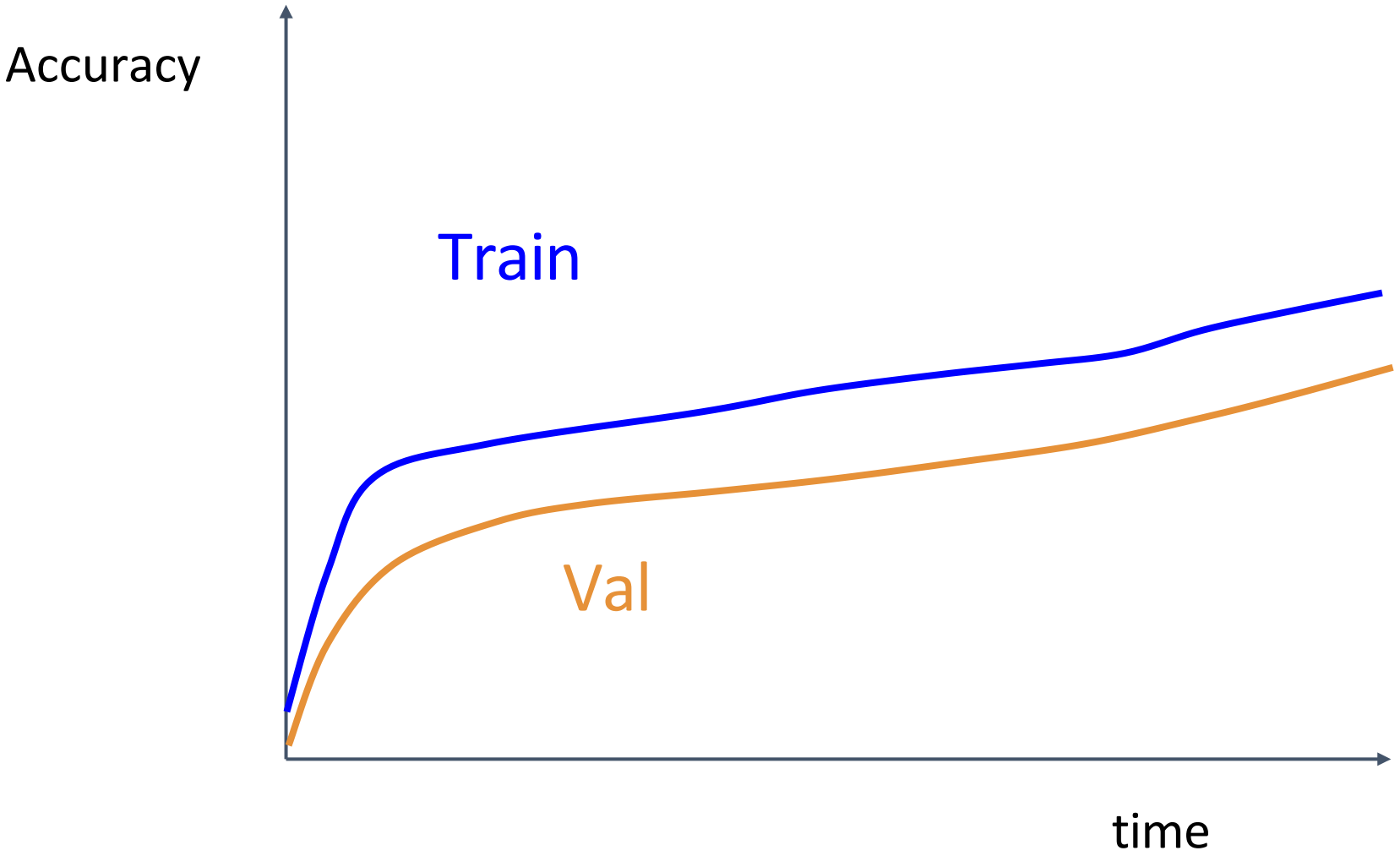

Loss
Bad initialization a prime suspect
time

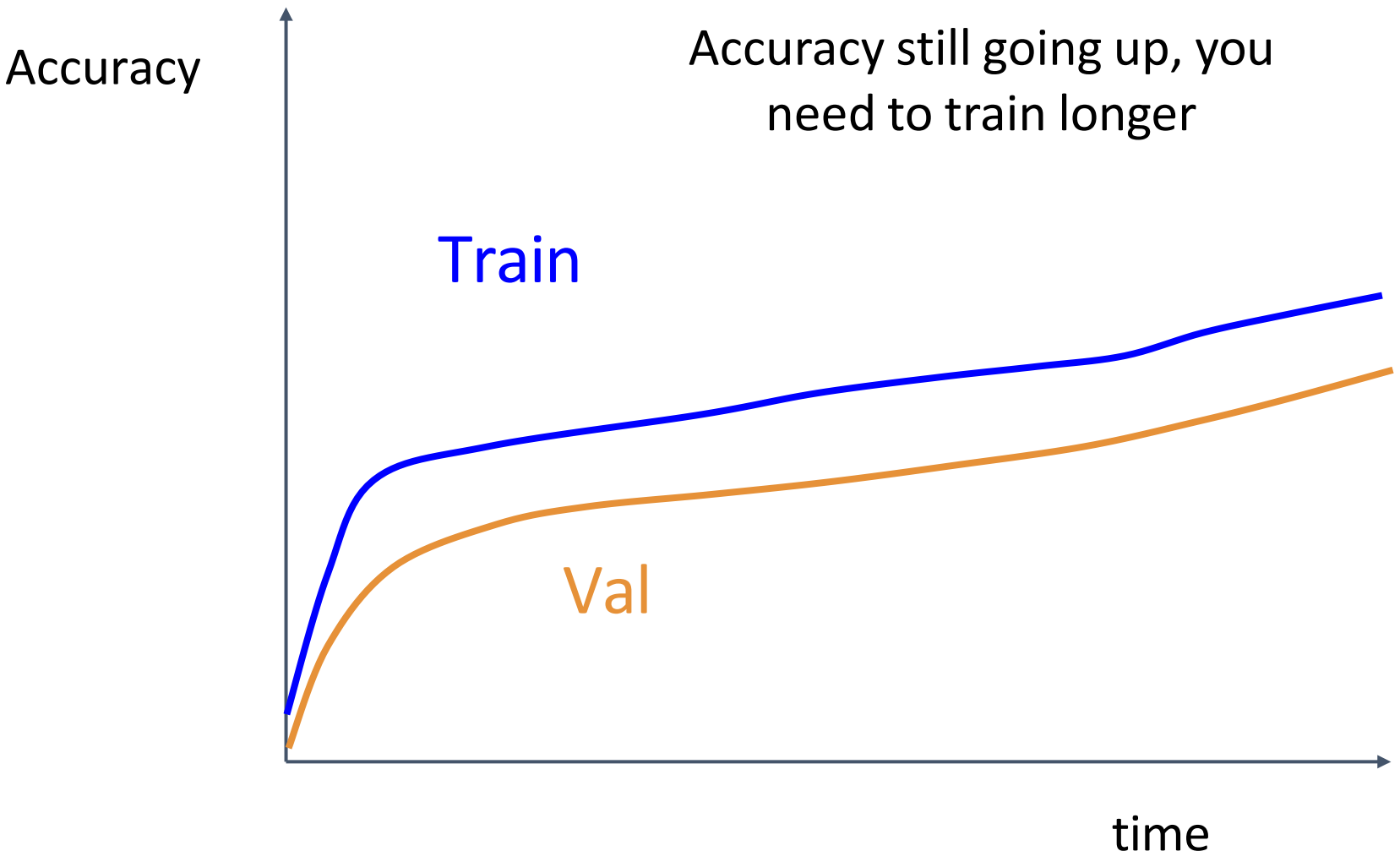

Loss
time

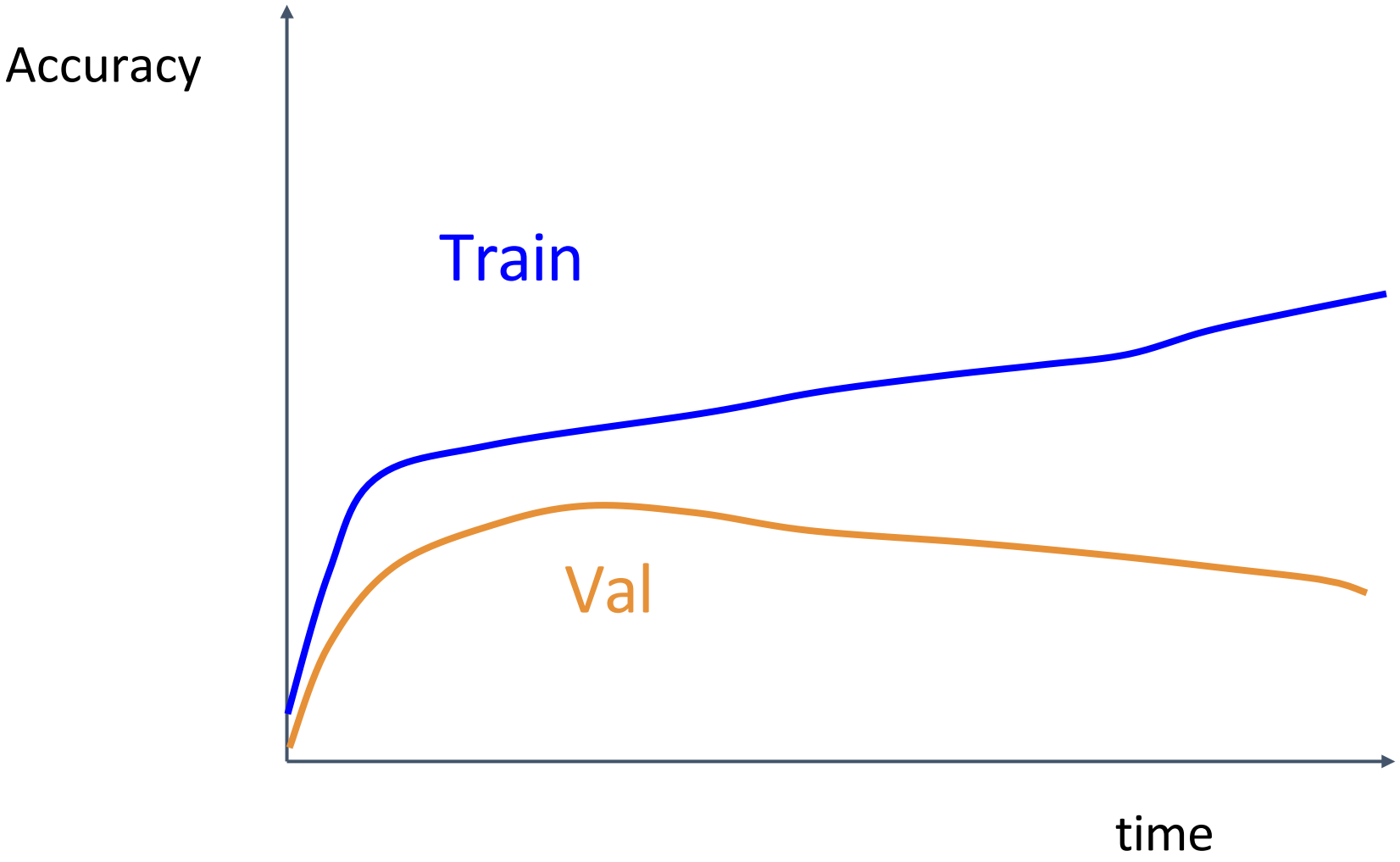

Loss
Loss plateaus: Try learning rate decay
time

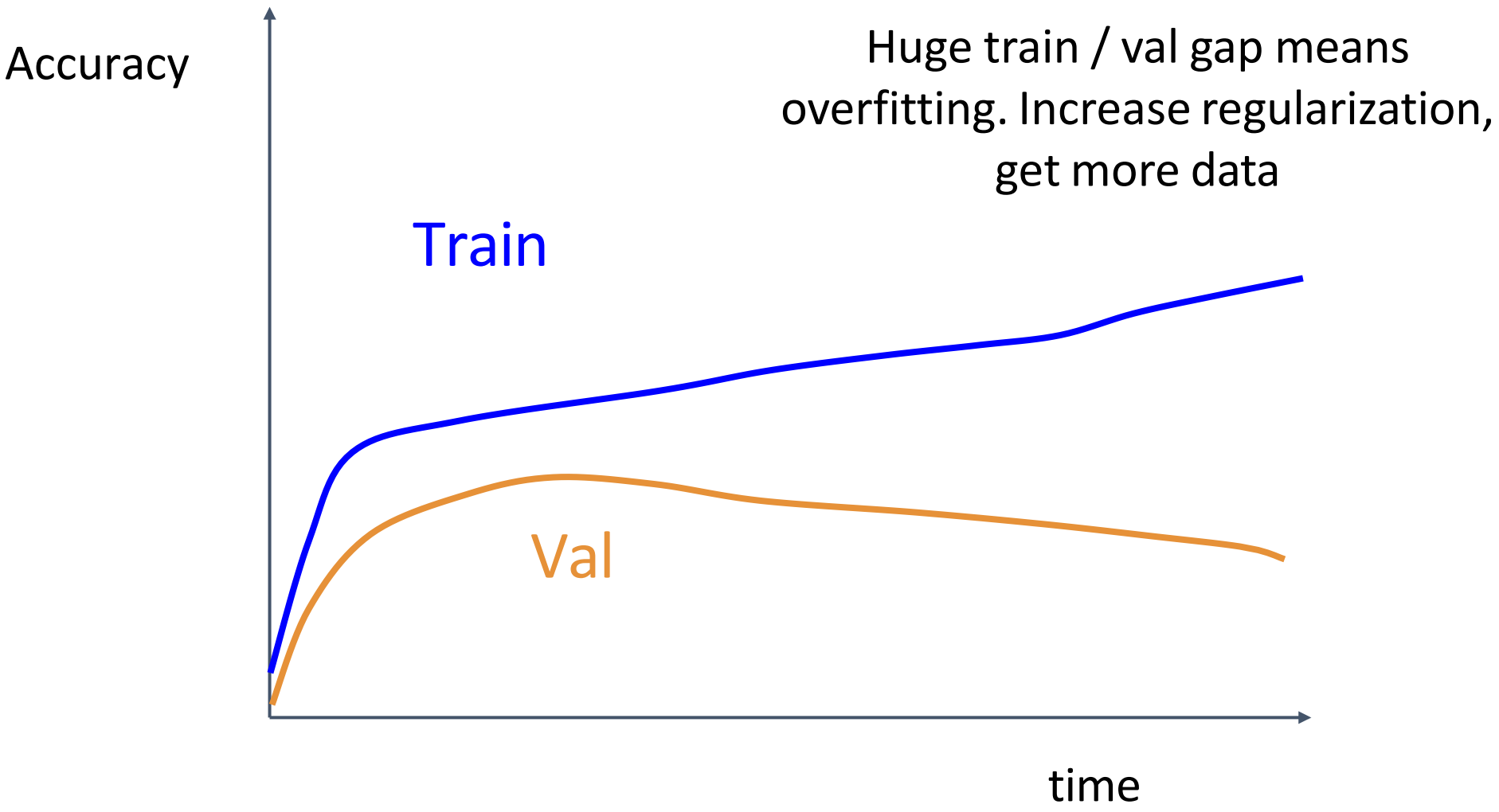

Loss
Learning rate step decay
time

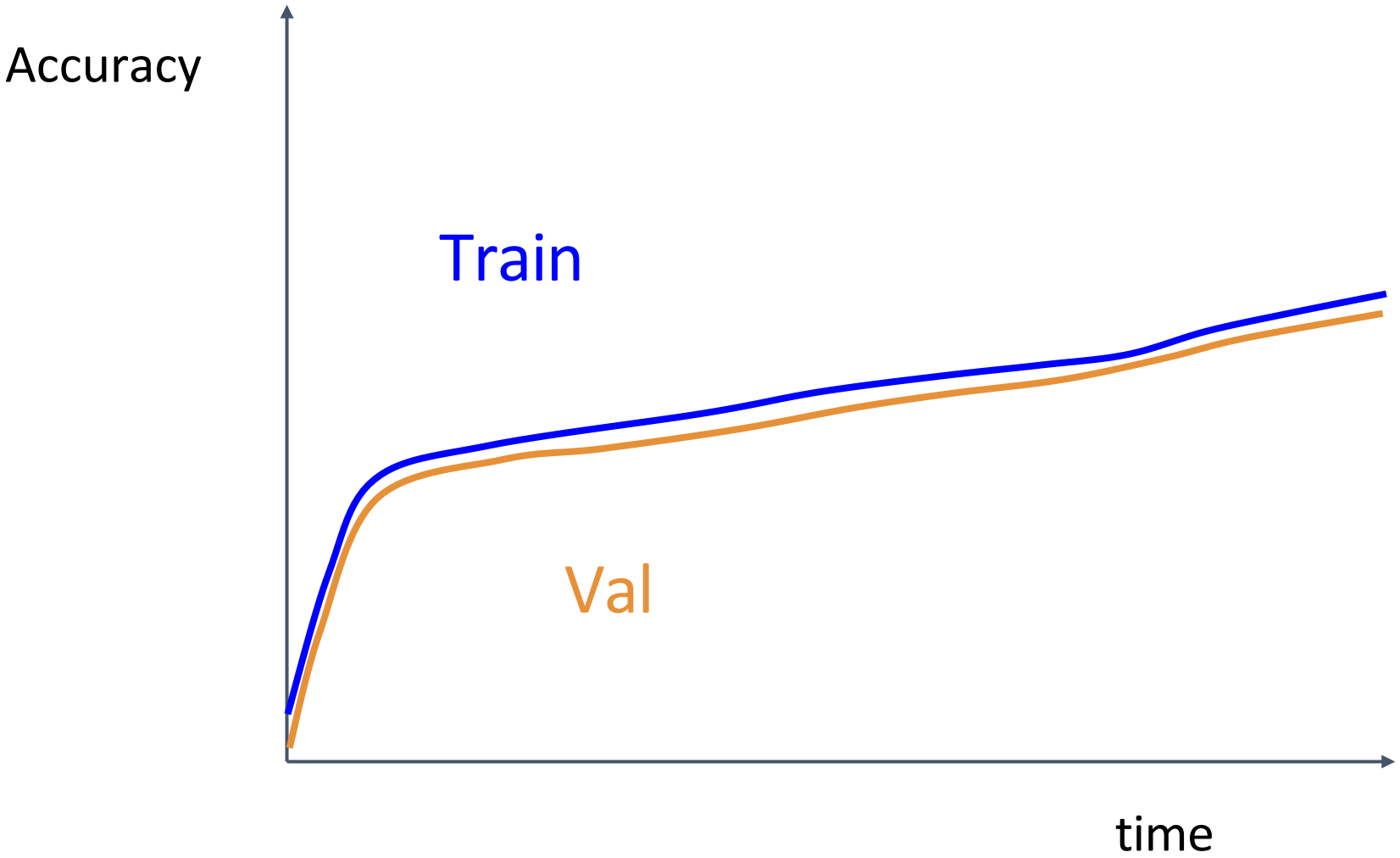

Loss
Learning rate step decay
Loss was still going down when learning rate dropped, you decayed too early
time

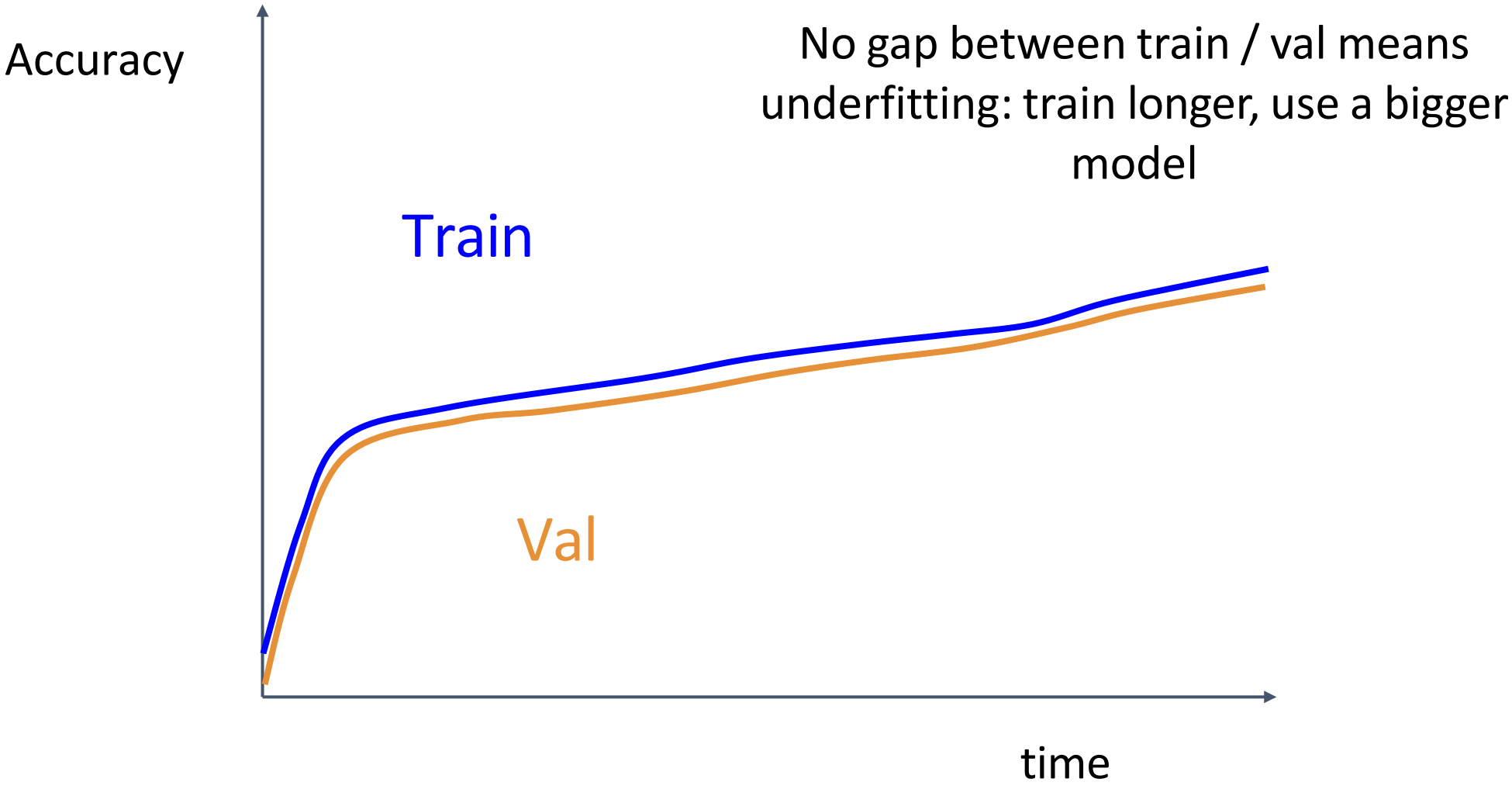

Accuracy
Train
Val
time


Accuracy
Accuracy still going up, you need to train longer
Train
Val
time


Accuracy
Train
Val
time


Accuracy
Huge train / val gap means overfitting. Increase regularization, get more data
Train
Val
time


Accuracy
Train
Val
time


Accuracy
No gap between train / val means underfitting: train longer, use a bigger model
Train
Val
time

Choosing Hyperparameters

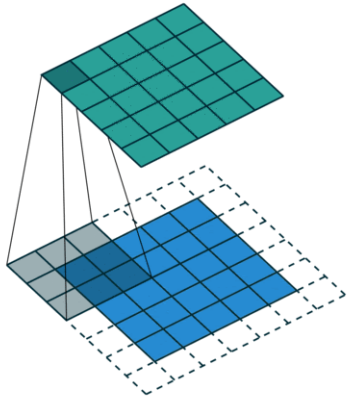
Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down
Step 4: Coarse grid, train for ~1-5 epochs
Step 5: Refine grid, train longer
Step 6: Look at loss curves
Step 7: GOTO step 5

Hyperparameters to play with:

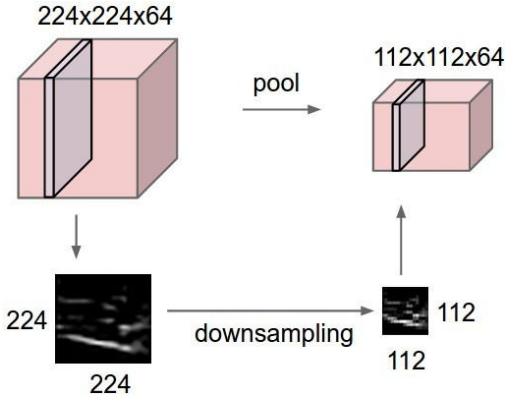
- network architecture
- learning rate, its decay schedule, update type
- regularization (L2/Dropout strength)

Components of a Convolutional Network

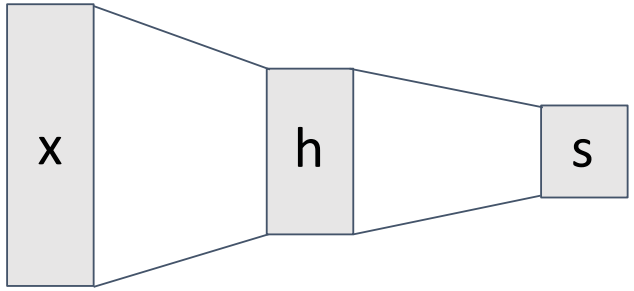
Convolution Layers



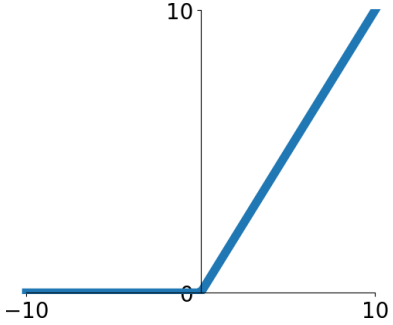
Pooling Layers



Fully-Connected Layers



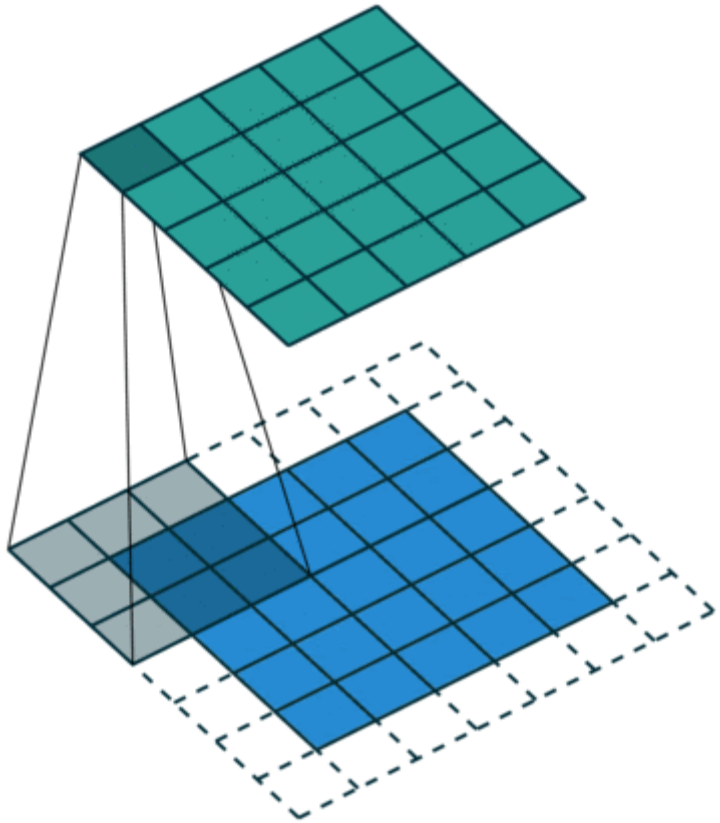
Activation Function



Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Convolution Layers



The Convolution operation

Assume a 6x6 matrix **M** as input. The 2D convolution of **M** with *filter (or kernel)* **F** and *stride* 1 is a 4x4 matrix **CM** (sometimes called *feature map*) computed as follows:

3	0	1	2	4	7
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

M

*

1	0	-1
1	0	-1
1	0	-1

F

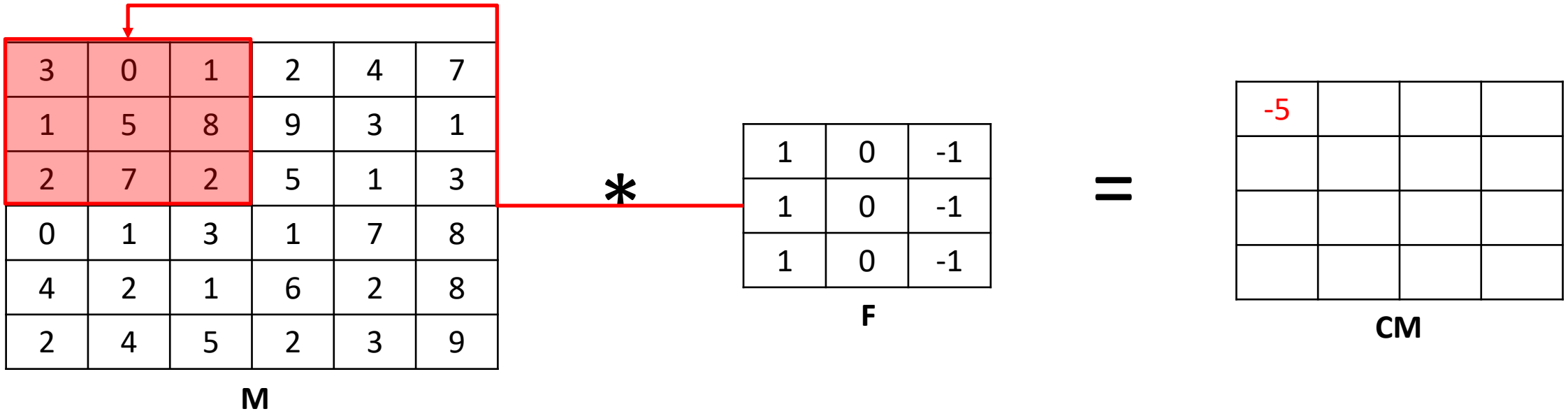
convolution operator

=

CM

The Convolution operation

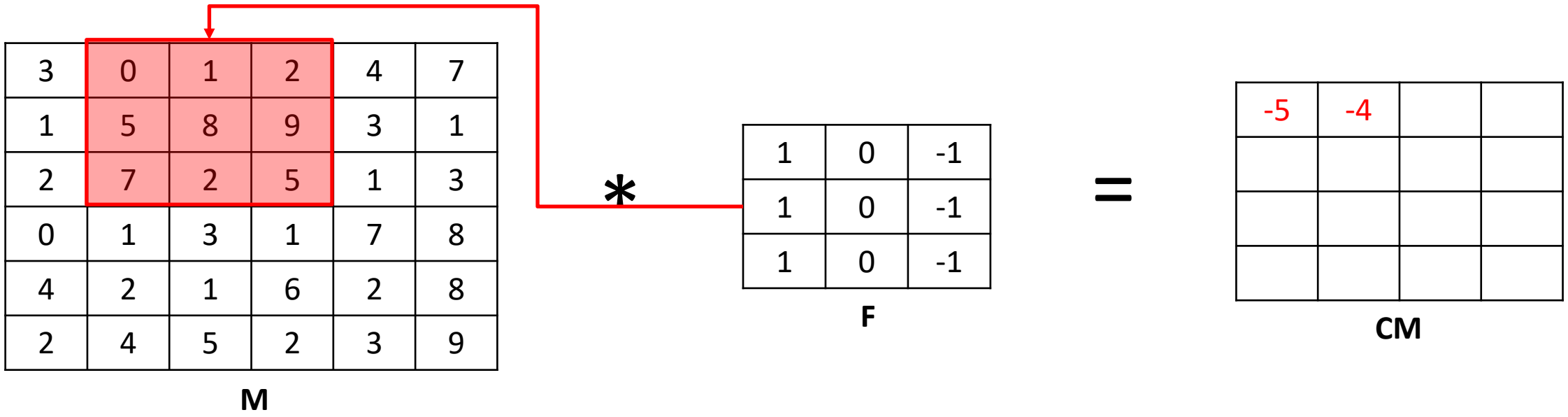
Assume a 6x6 matrix **M** as input. The 2D convolution of **M** with *filter (or kernel)* **F** and *stride* 1 is a 4x4 matrix **CM** (sometimes called *feature map*) computed as follows:



$$3*1 + 1*1 + 2*1 + 0*0 + 5*0 + 7*0 + 1*(-1) + 8*(-1) + 2*(-1)$$

The Convolution operation

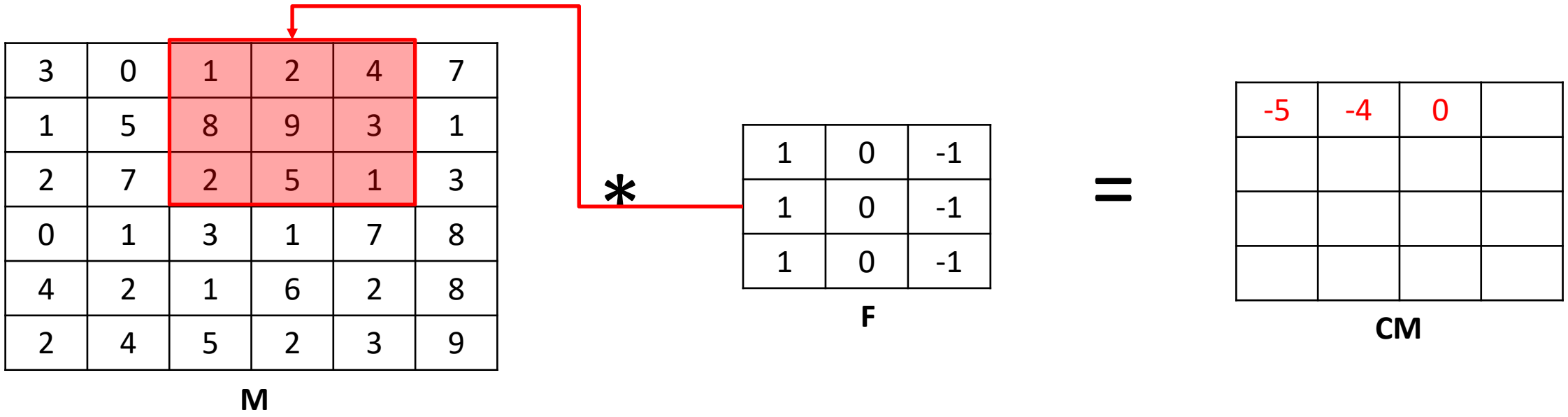
Assume a 6x6 matrix **M** as input. The 2D convolution of **M** with *filter (or kernel)* **F** and *stride* 1 is a 4x4 matrix **CM** (sometimes called *feature map*) computed as follows:



$$0*1 + 5*1 + 7*1 + 1*0 + 8*0 + 2*0 + 2*(-1) + 9*(-1) + 5*(-1)$$

The Convolution operation

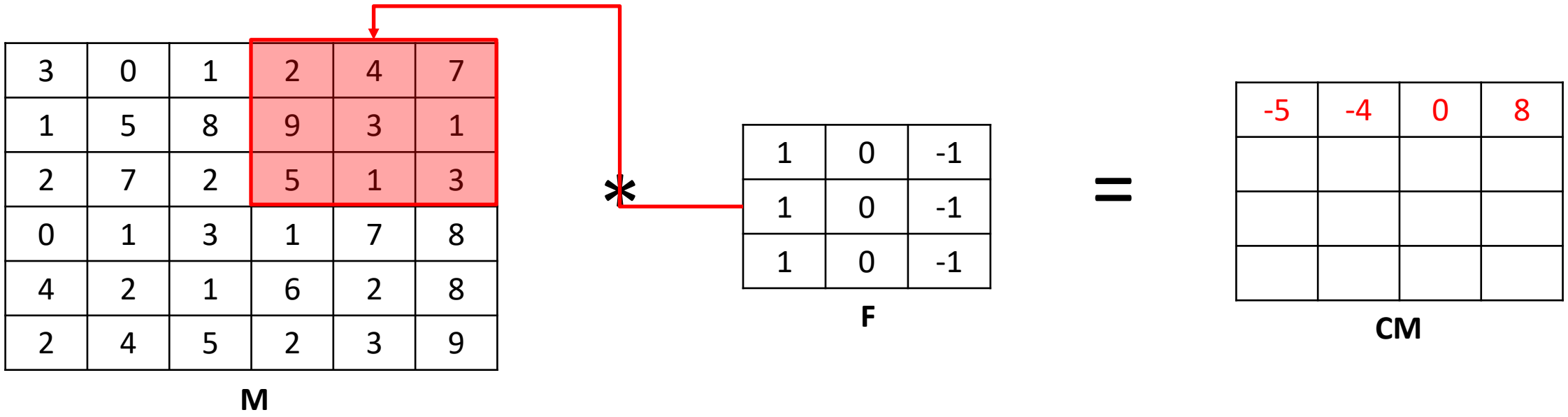
Assume a 6x6 matrix **M** as input. The 2D convolution of **M** with *filter (or kernel)* **F** and *stride* 1 is a 4x4 matrix **CM** (sometimes called *feature map*) computed as follows:



$$1*1 + 8*1 + 2*1 + 2*0 + 9*0 + 5*0 + 4*(-1) + 3*(-1) + 1*(-1)$$

The Convolution operation

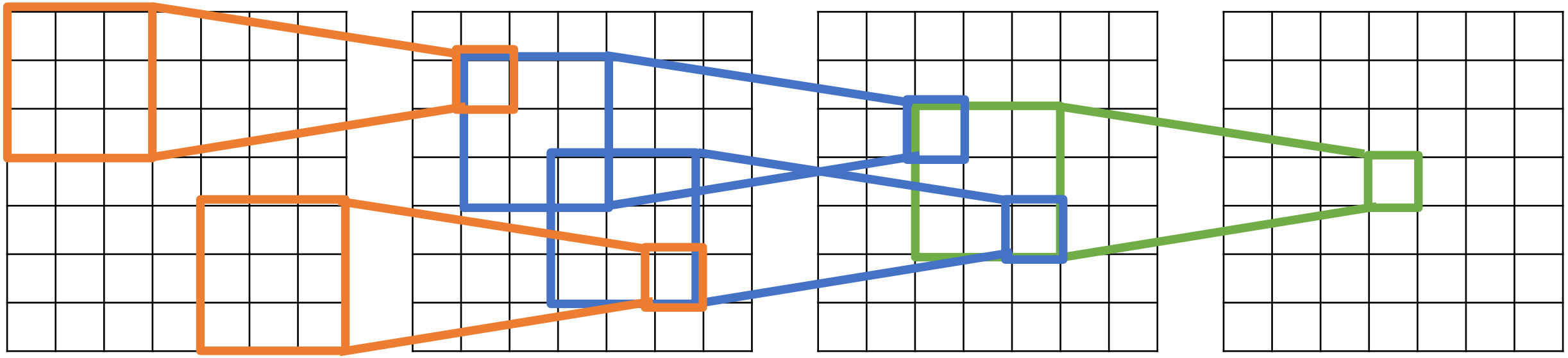
Assume a 6x6 matrix **M** as input. The 2D convolution of **M** with *filter (or kernel)* **F** and *stride* 1 is a 4x4 matrix **CM** (sometimes called *feature map*) computed as follows:



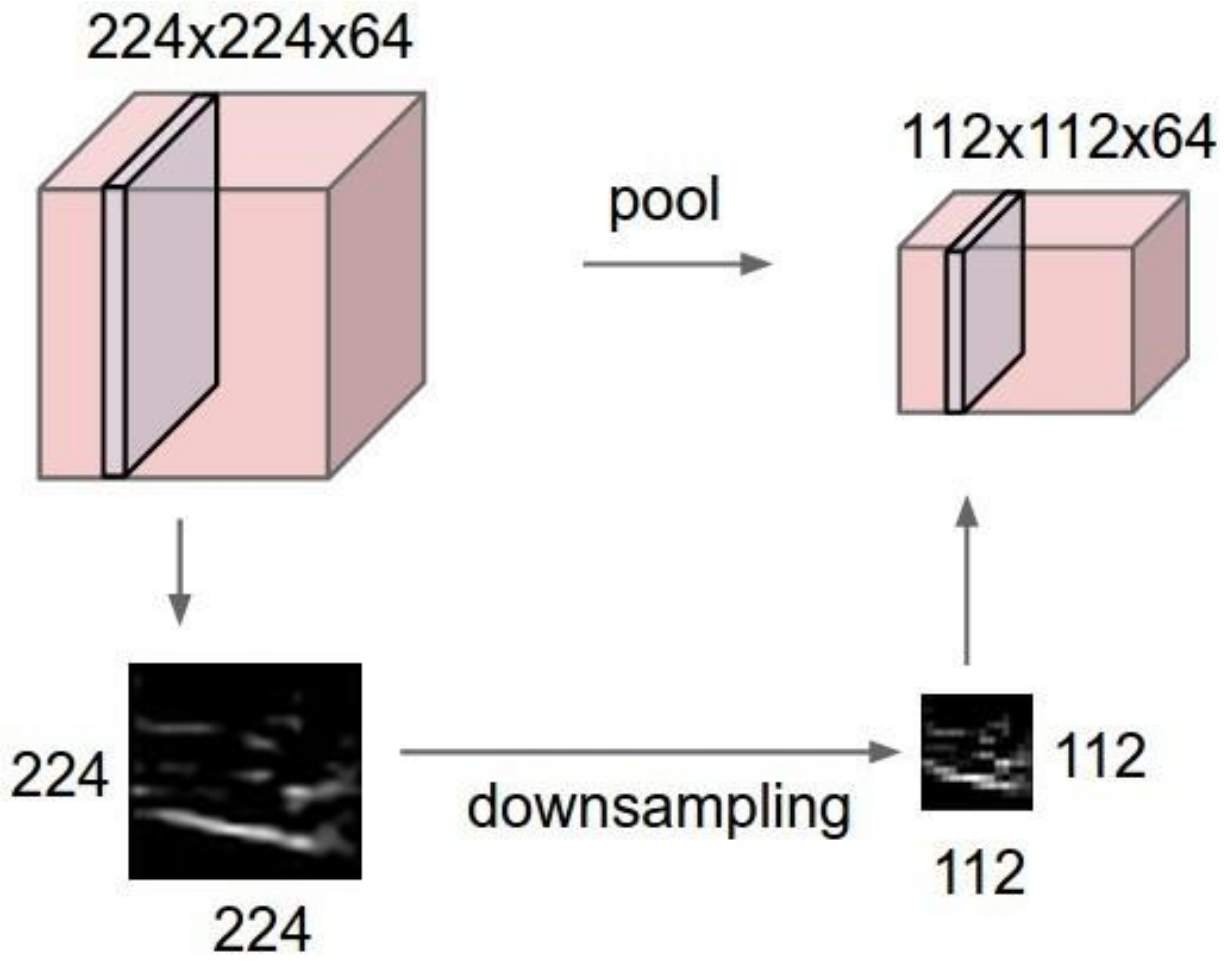
$$2*1 + 9*1 + 5*1 + 4*0 + 3*0 + 1*0 + 7*(-1) + 1*(-1) + 3*(-1)$$

Receptive Fields

Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)

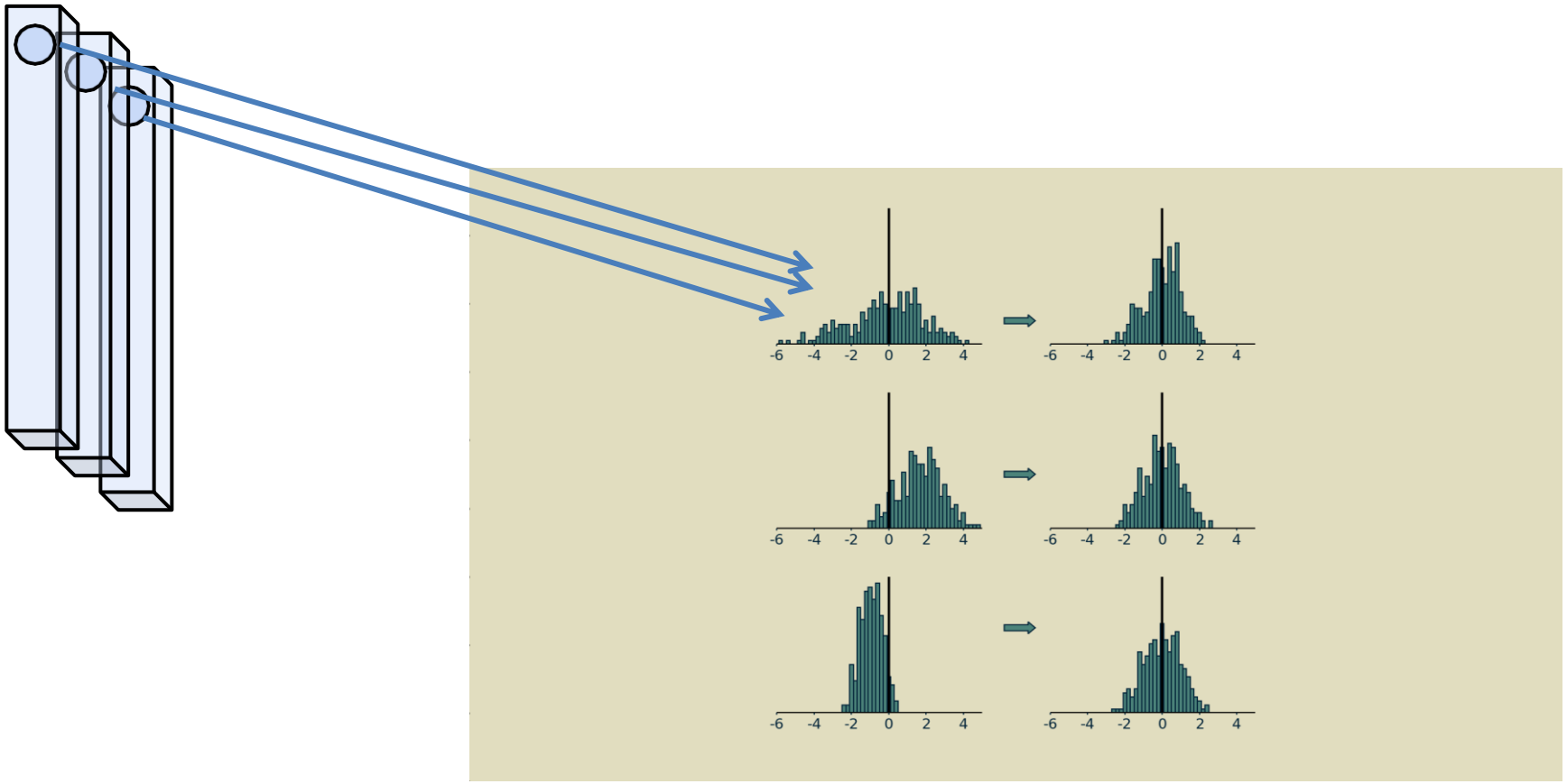


Pooling Layers: Another way to downsample



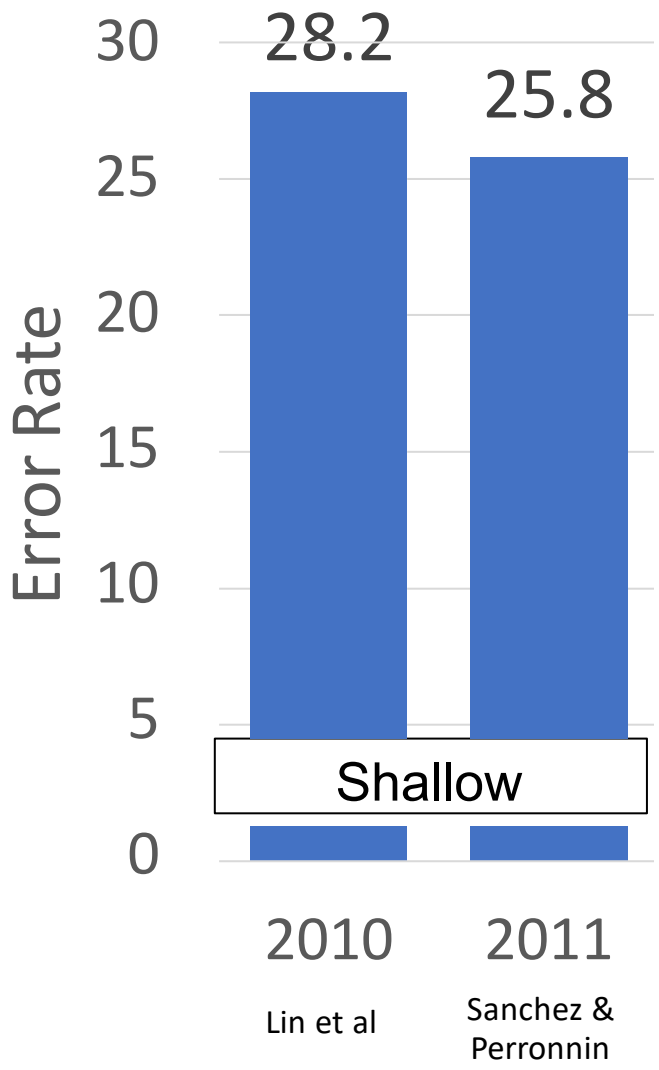
Hyperparameters:
Kernel Size
Stride
Pooling function

Batch Normalization

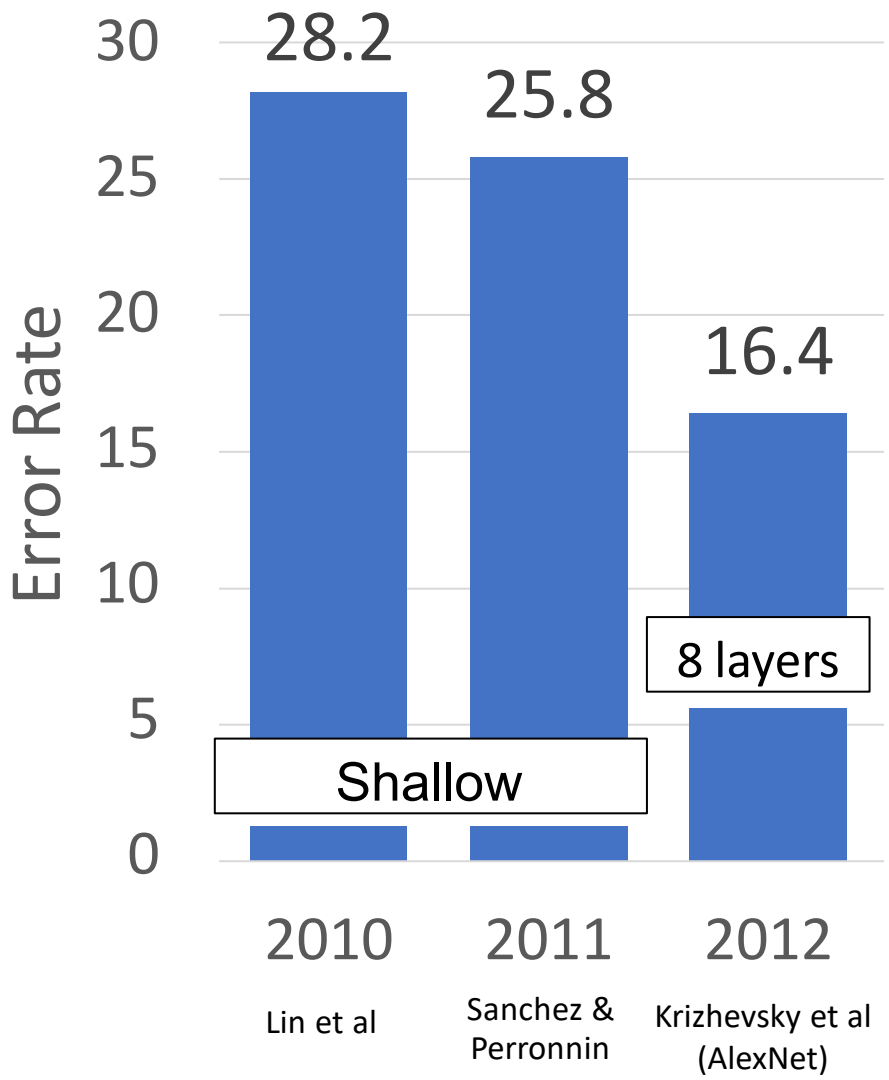


Normalize node activation across a batch

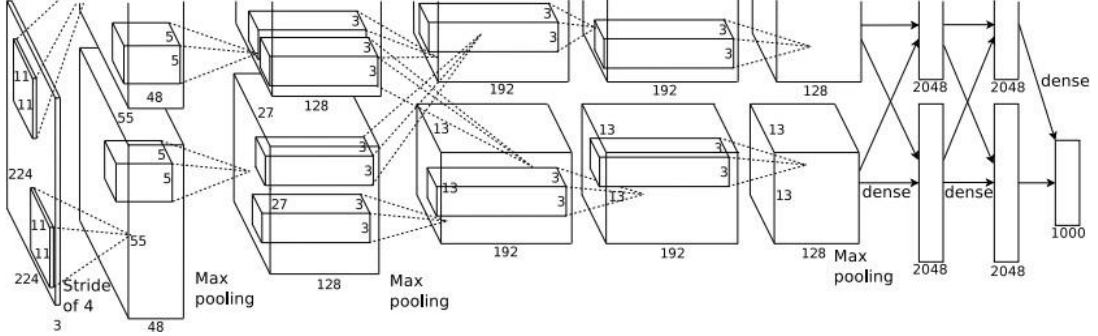
ImageNet Classification Challenge



ImageNet Classification Challenge



AlexNet



227 x 227 inputs
5 Convolutional layers
Max pooling
3 fully-connected layers
ReLU nonlinearities

AlexNet

	Input size		Layer				Output size	
Layer	C	H / W	filters	kernel	stride	pad	C	H / W
conv1	3	227	64	11	4	2		

AlexNet

	Input size		Layer				Output size	
Layer	C	H / W	filters	kernel	stride	pad	C	H / W
conv1	3	227	64	11	4	2	64	56

Recall: $W' = (W - K + 2P) / S + 1$

$= (227 - 11 + 2*2) / 4 + 1$

$= 220/4 + 1 = 56$

AlexNet

	Input size		Layer				Output size		
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)
conv1	3	227	64	11	4	2	64	56	

AlexNet

	Input size		Layer				Output size		
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)
conv1	3	227	64	11	4	2	64	56	784

Number of output elements = C * H' * W'

= 64*56*56 = 200,704

Bytes per element = 4 (for 32-bit floating point)

KB = (number of elements) * (bytes per elem) / 1024

= 200704 * 4 / 1024

= **784**

AlexNet

	Input size		Layer				Output size			
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)
conv1	3	227	64	11	4	2	64	56	784	

AlexNet

	Input size		Layer				Output size			
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)
conv1	3	227	64	11	4	2	64	56	784	23

Weight shape = C_{out} x C_{in} x K x K

= 64 x 3 x 11 x 11

Bias shape = C_{out} = 64

Number of weights = 64*3*11*11 + 64

= **23,296**

AlexNet

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	

AlexNet

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73

Number of floating point operations (multiply+add)

= (number of output elements) * (ops per output elem)

= (C_{out} x H' x W') * (C_{in} x K x K)

= (64 * 56 * 56) * (3 * 11 * 11)

= 200,704 * 363

= **72,855,552**

AlexNet

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0					

AlexNet

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27			

For pooling layer:

#output channels = #input channels = 64

W' = floor((W – K) / S + 1)

= floor(53 / 2 + 1) = floor(27.5) = **27**

AlexNet

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182		

#output elems = C_{out} x H' x W'

Bytes per elem = 4

KB = C_{out} * H' * W' * 4 / 1024

= 64 * 27 * 27 * 4 / 1024

= **182.25**

AlexNet

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	

Pooling layers have no learnable parameters!

AlexNet

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0

Floating-point ops for pooling layer

= (number of output positions) * (flops per output position)

= (C_{out} * H' * W') * (K * K)

= (64 * 27 * 27) * (3 * 3)

= 419,904

= **0.4 MFLOP**

AlexNet

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	112
conv4	384	13	256	3	1	1	256	13	169	885	145
conv5	256	13	256	3	1	1	256	13	169	590	100
pool5	256	13		3	2	0	256	6	36	0	0
flatten	256	6					9216		36	0	0

AlexNet

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	112
conv4	384	13	256	3	1	1	256	13	169	885	145
conv5	256	13	256	3	1	1	256	13	169	590	100
pool5	256	13		3	2	0	256	6	36	0	0
flatten	256	6					9216		36	0	0

Flatten output size = C_{in} x H x W

= 256 * 6 * 6

= **9216**

AlexNet

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	112
conv4	384	13	256	3	1	1	256	13	169	885	145
conv5	256	13	256	3	1	1	256	13	169	590	100
pool5	256	13		3	2	0	256	6	36	0	0
flatten	256	6					9216		36	0	0
fc6	9216		4096				4096		16	37,749	38

FC params $= C_{in} \times C_{out} + C_{out}$
$= 9216 * 4096 + 4096$
$= 37{,}725{,}832$

FC flops $= C_{in} \times C_{out}$
$= 9216 * 6409$
$= 37{,}748{,}736$

AlexNet

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	112
conv4	384	13	256	3	1	1	256	13	169	885	145
conv5	256	13	256	3	1	1	256	13	169	590	100
pool5	256	13		3	2	0	256	6	36	0	0
flatten	256	6					9216		36	0	0
fc6	9216		4096				4096		16	37,749	38

FC params $= C_{in} \times C_{out} + C_{out}$
$= 9216 * 4096 + 4096$
$= 37{,}725{,}832$

FC flops $= C_{in} \times C_{out}$
$= 9216 * 6409$
$= 37{,}748{,}736$

AlexNet

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	112
conv4	384	13	256	3	1	1	256	13	169	885	145
conv5	256	13	256	3	1	1	256	13	169	590	100
pool5	256	13		3	2	0	256	6	36	0	0
flatten	256	6					9216		36	0	0
fc6	9216		4096				4096		16	37,749	38
fc7	4096		4096				4096		16	16,777	17
fc8	4096		1000				1000		4	4,096	4

AlexNet

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	112
conv4	384	13	256	3	1	1	256	13	169	885	145
conv5	256	13	256	3	1	1	256	13	169	590	100
pool5	256	13		3	2	0	256	6	36	0	0
flatten	256	6					9216		36	0	0
fc6	9216		4096				4096		16	37,749	38
fc7	4096		4096				4096		16	16,777	17
fc8	4096		1000				1000		4	4,096	4

Determined by trial and error

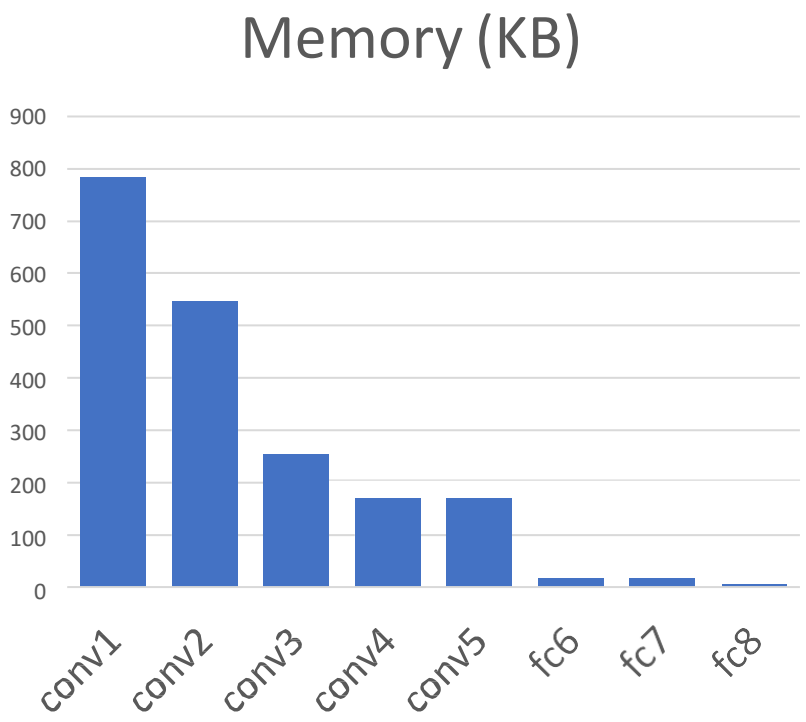
AlexNet

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	112
conv4	384	13	256	3	1	1	256	13	169	885	145
conv5	256	13	256	3	1	1	256	13	169	590	100
pool5	256	13		3	2	0	256	6	36	0	0
flatten	256	6					9216		36	0	0
fc6	9216		4096				4096		16	37,749	38
fc7	4096		4096				4096		16	16,777	17
fc8	4096		1000				1000		4	4,096	4

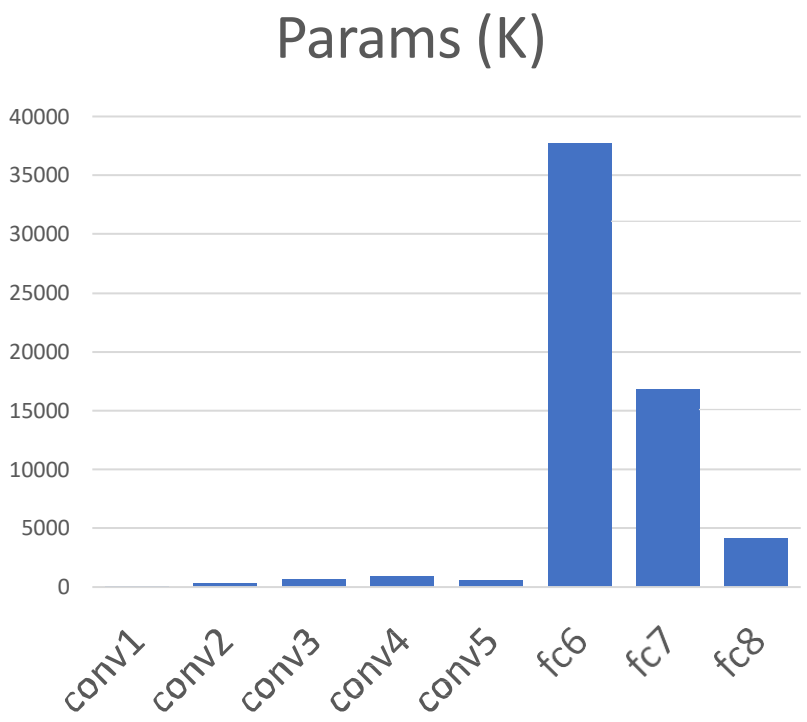
Max pooling inexpensive

AlexNet

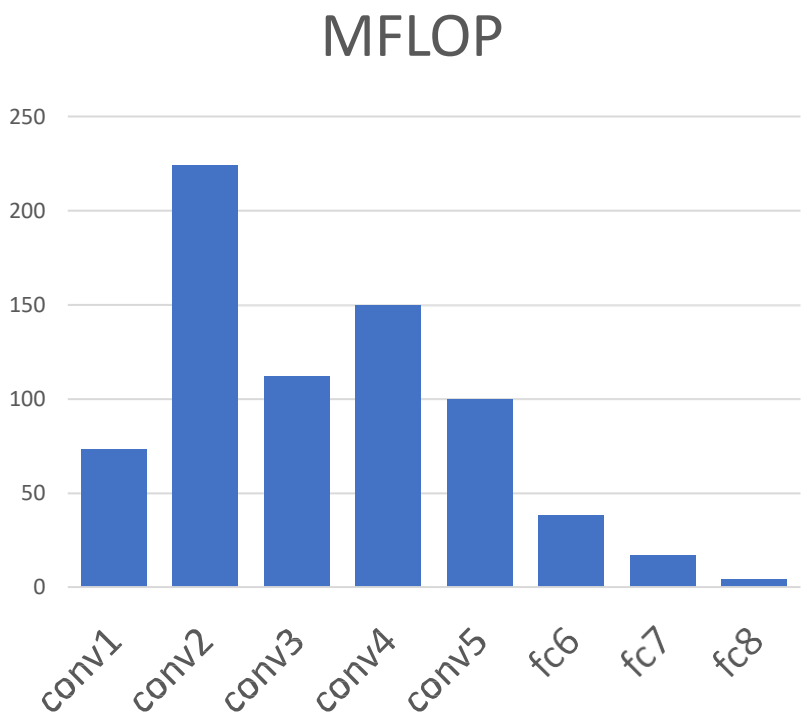
Most of the **memory usage** is in the early convolution layers



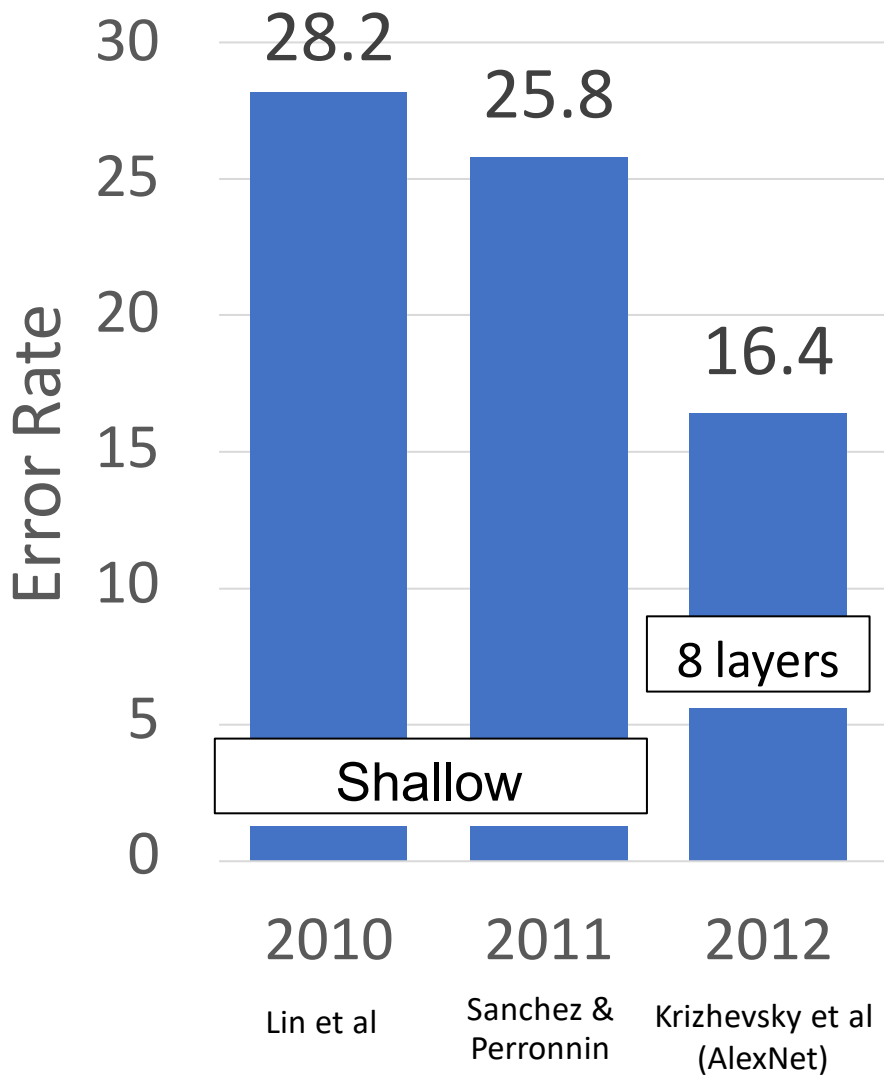
Nearly all **parameters** are in the fully-connected layers



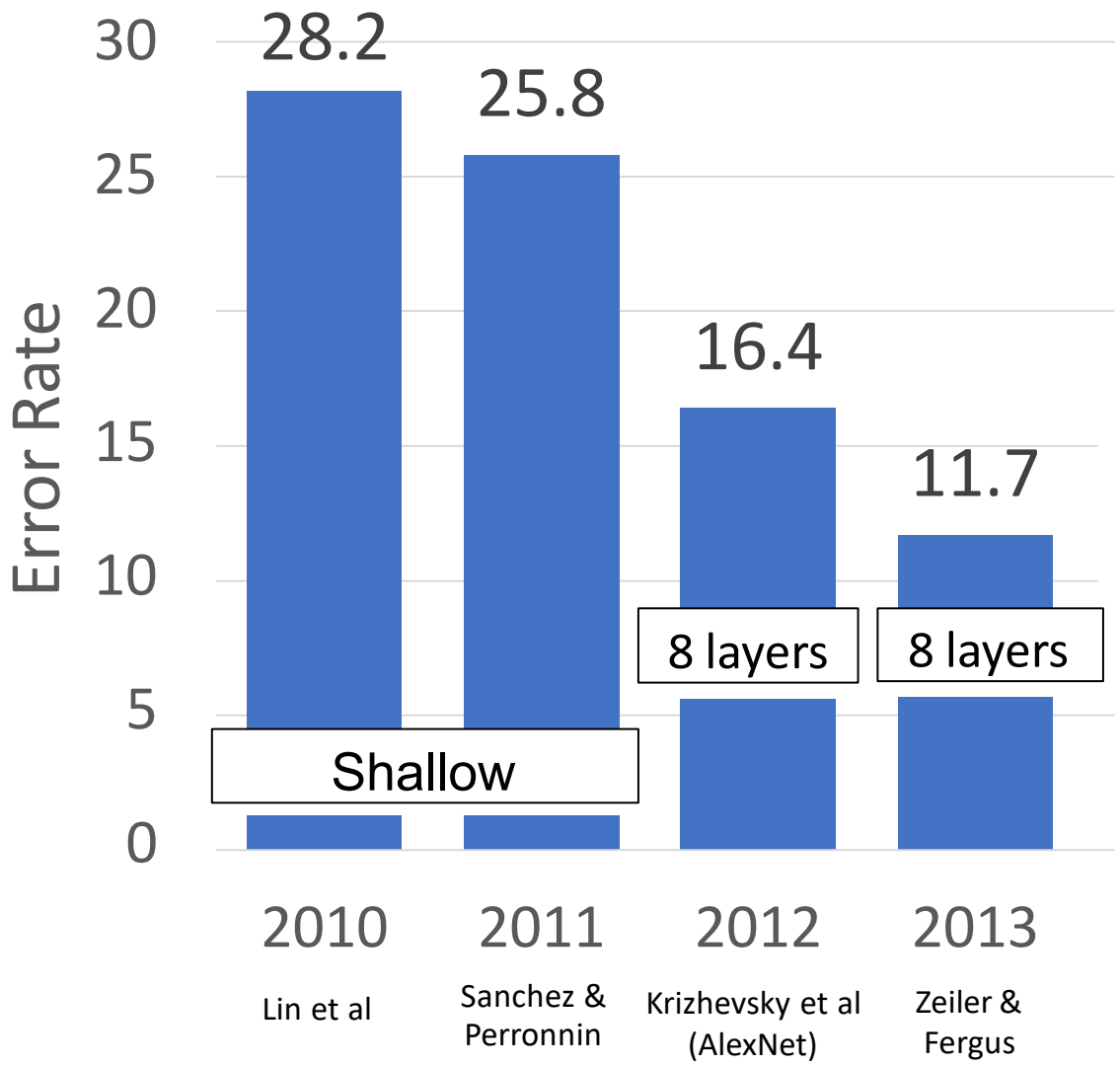
Most **floating-point ops** occur in the convolution layers



ImageNet Classification Challenge

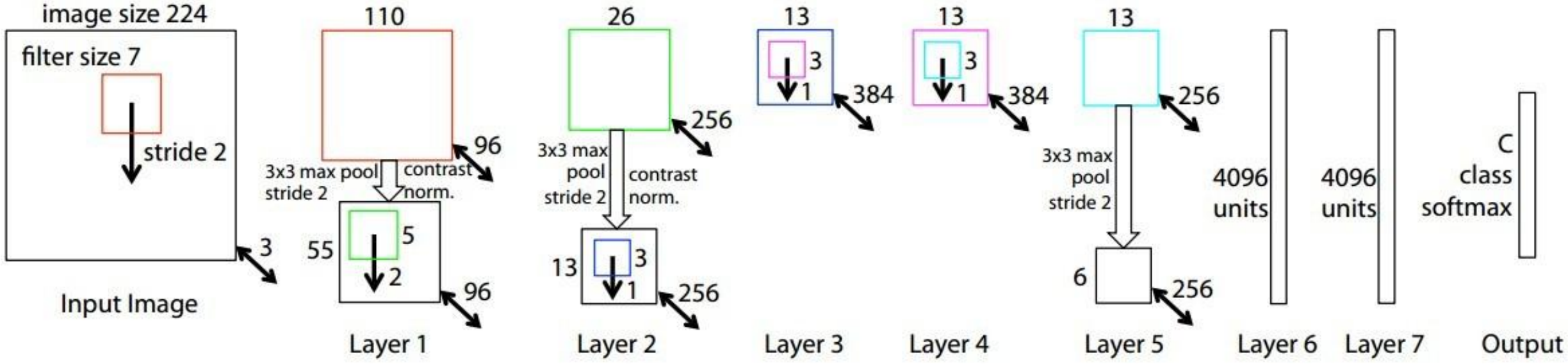


ImageNet Classification Challenge



ZFNet: A Bigger AlexNet

ImageNet top 5 error: 16.4% -> 11.7%



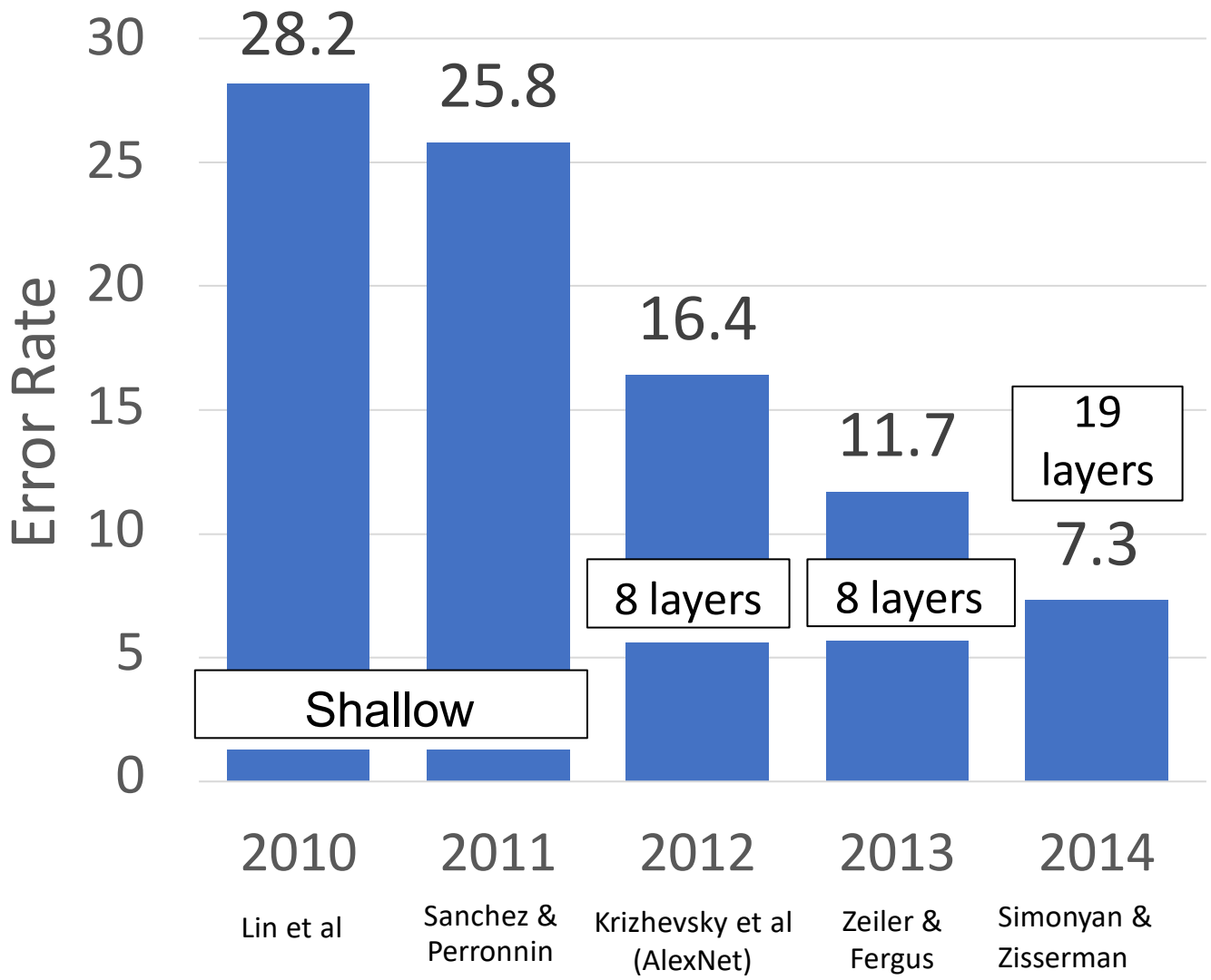
AlexNet but:

CONV1: change from (11x11 stride 4) to (7x7 stride 2)

CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

ImageNet Classification Challenge



VGG: Deeper Networks, Regular Design

Network has 5 convolutional **stages**:

Stage 1: conv-conv-pool

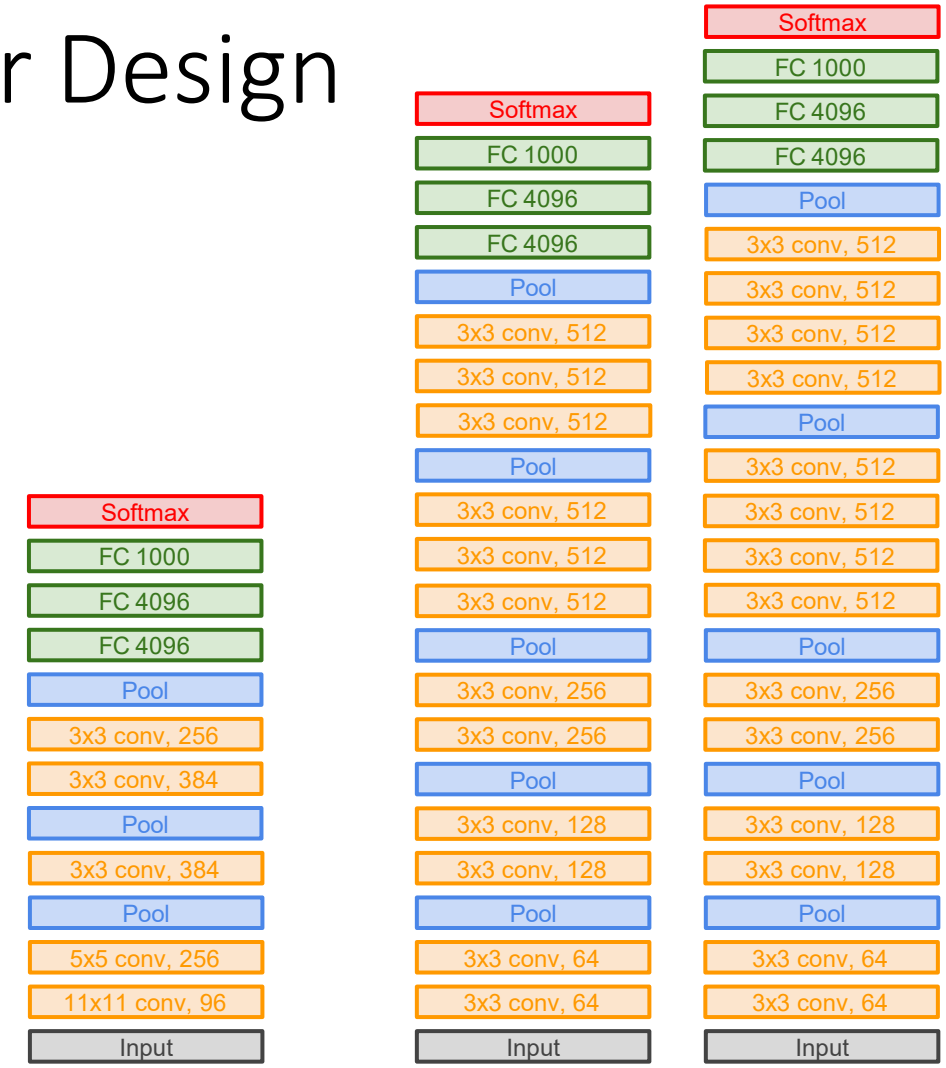
Stage 2: conv-conv-pool

Stage 3: conv-conv-pool

Stage 4: conv-conv-conv-[conv]-pool

Stage 5: conv-conv-conv-[conv]-pool

(VGG-19 has 4 conv in stages 4 and 5)



VGG: Deeper Networks, Regular Design

VGG Design rules:

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

Network has 5 convolutional **stages**:

Stage 1: conv-conv-pool

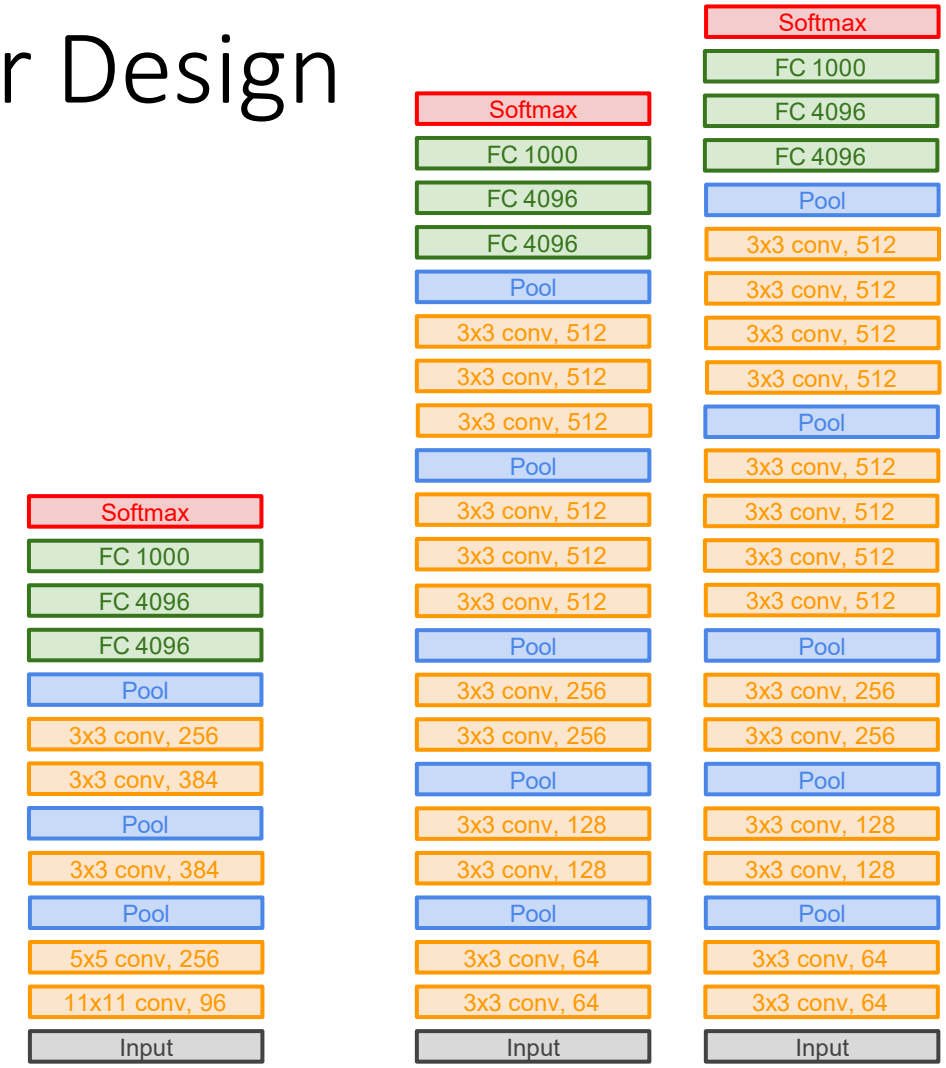
Stage 2: conv-conv-pool

Stage 3: conv-conv-pool

Stage 4: conv-conv-conv-[conv]-pool

Stage 5: conv-conv-conv-[conv]-pool

(VGG-19 has 4 conv in stages 4 and 5)



VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

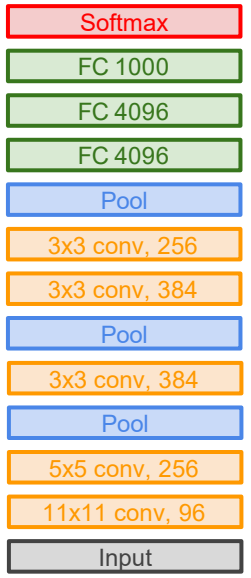
After pool, double #channels

<u>Option 1:</u>

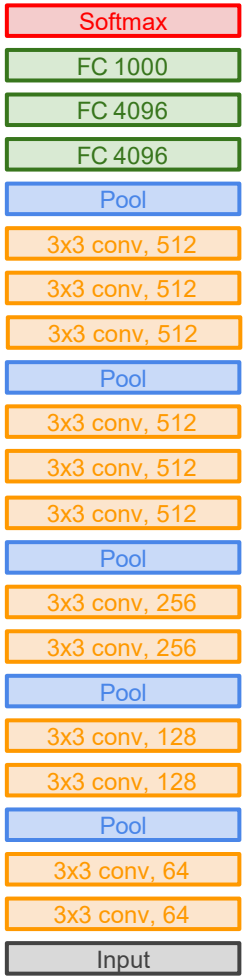
Conv(5x5, C -> C)

Params: $25C^2$

FLOPs: $25C^2HW$



AlexNet



VGG16

VGG19

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

<u>Option 1:</u>

Conv(5x5, C -> C)

Params: $25C^2$

FLOPs: $25C^2HW$

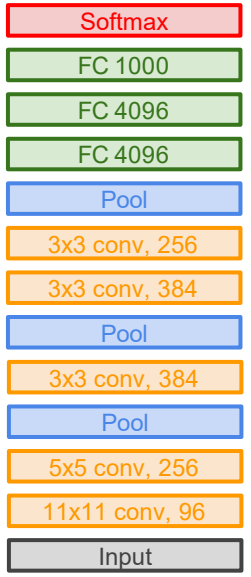
<u>Option 2:</u>

Conv(3x3, C -> C)

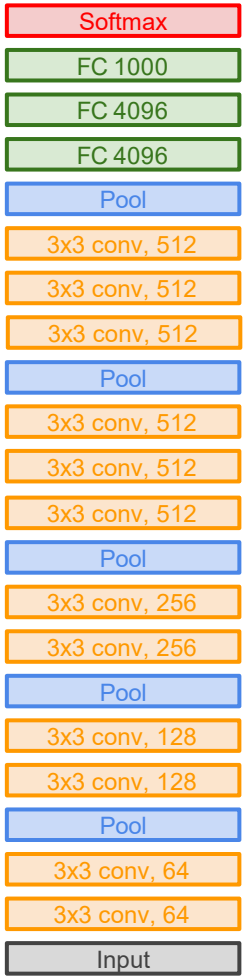
Conv(3x3, C -> C)

Params: $18C^2$

FLOPs: $18C^2HW$



AlexNet



VGG16

VGG19

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

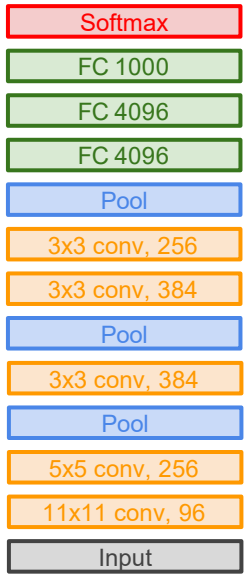
Input: C x 2H x 2W

Layer: Conv(3x3, C->C)

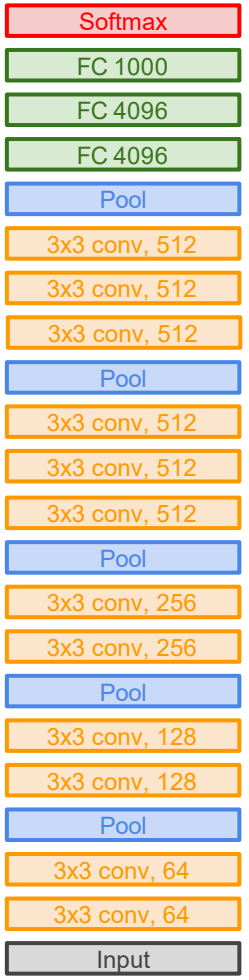
Memory: 4HWC

Params: $9C^2$

FLOPs: $36HWC^2$



AlexNet



VGG16

VGG19

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

Input: C x 2H x 2W

Layer: Conv(3x3, C->C)

Memory: 4HWC

Params: $9C^2$

FLOPs: $36HWC^2$

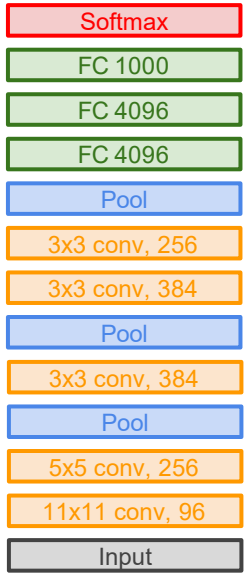
Input: 2C x H x W

Conv(3x3, 2C -> 2C)

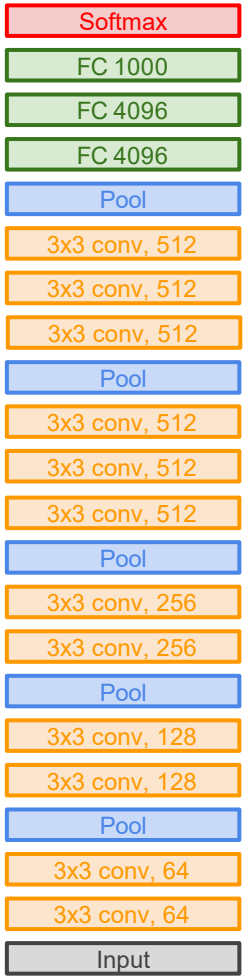
Memory: 2HWC

Params: $36C^2$

FLOPs: $36HWC^2$



AlexNet



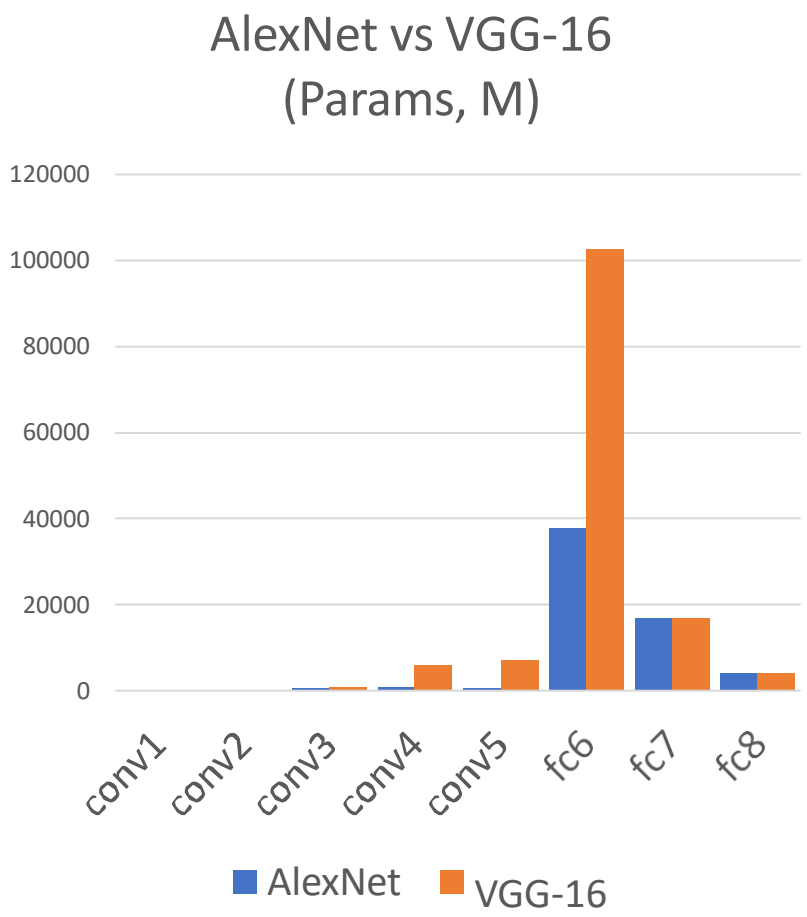
VGG16

VGG19

AlexNet vs VGG-16

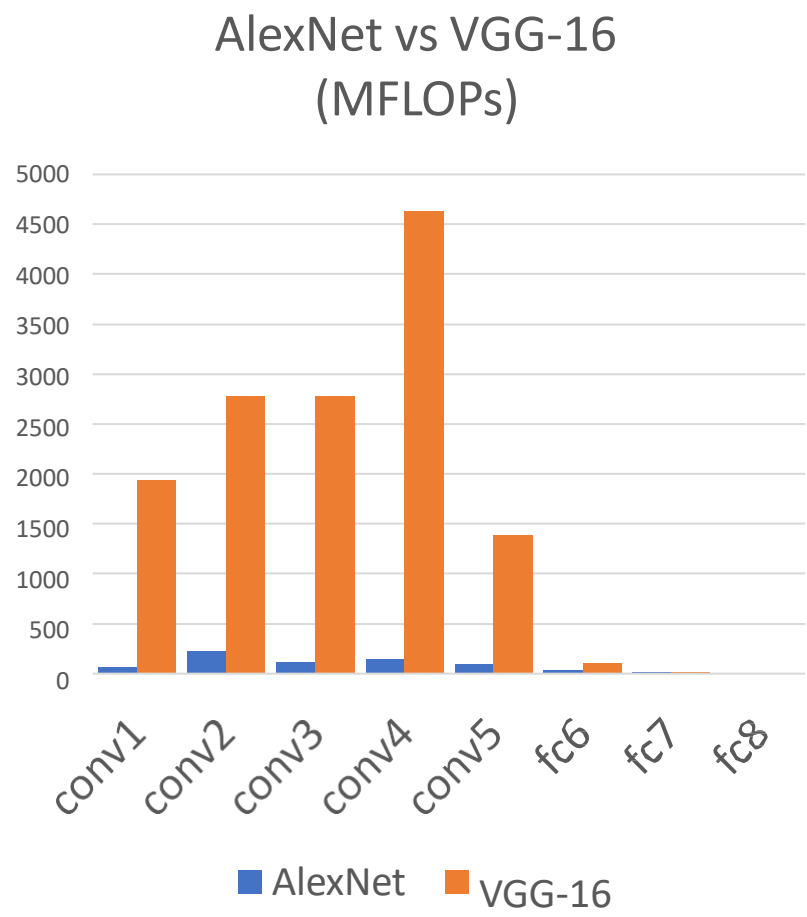
AlexNet total: 1.9 MB

VGG-16 total: 48.6 MB (25x)



AlexNet total: 61M

VGG-16 total: 138M (2.3x)

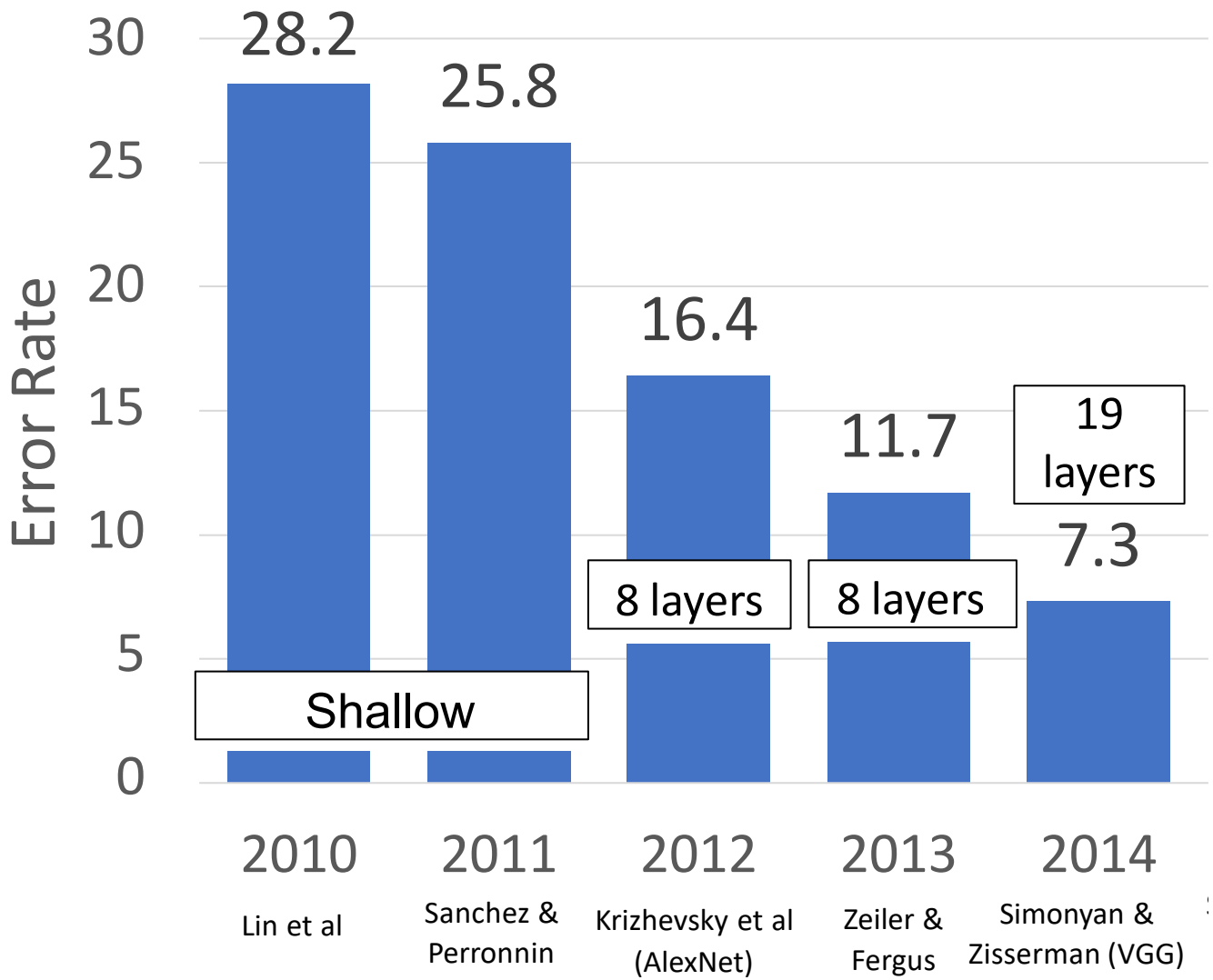


AlexNet total: 0.7 GFLOP

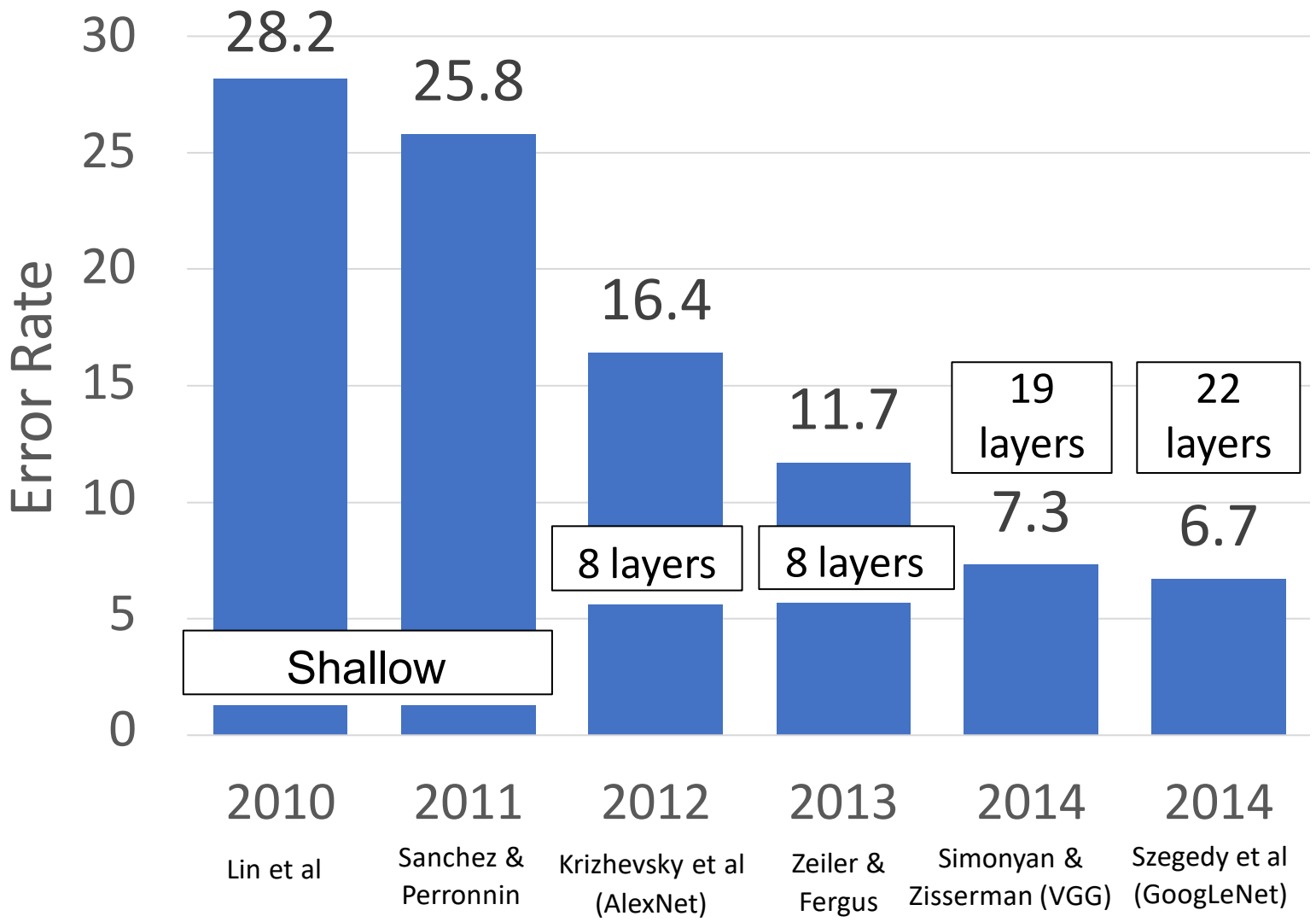
VGG-16 total: 13.6 GFLOP (19.4x)

Simonyan and Zissermann, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, ICLR 2015

ImageNet Classification Challenge



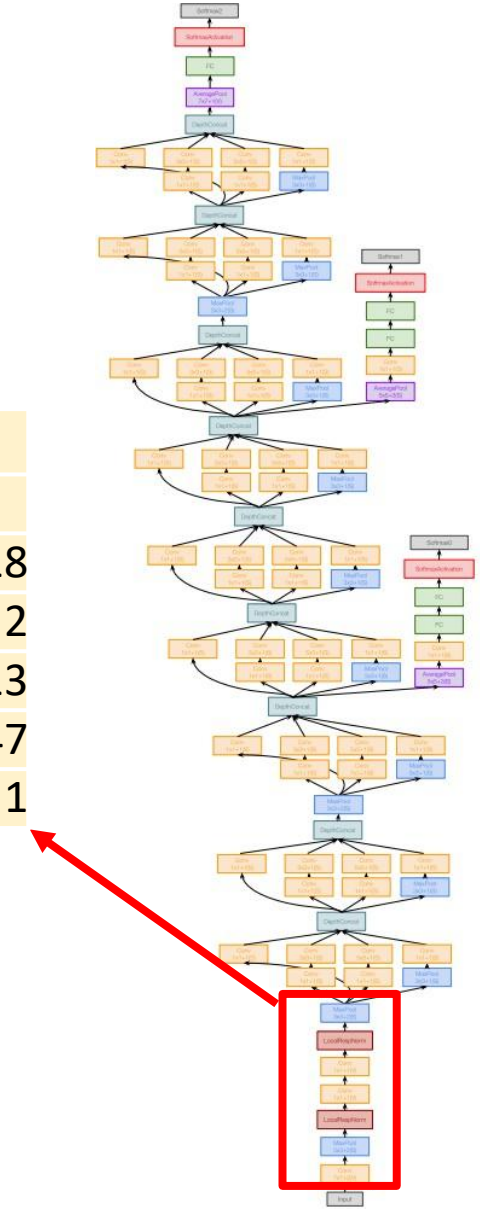
ImageNet Classification Challenge



GoogLeNet: Aggressive Stem

Stem network at the start aggressively downsamples input
(Recall in VGG-16: Most of the compute was at the start)

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (K)	flop (M)
conv	3	224	64	7	2	3	64	112	3136	9	118
max-pool	64	112		3	2	1	64	56	784	0	2
conv	64	56	64	1	1	0	64	56	784	4	13
conv	64	56	192	3	1	1	192	56	2352	111	347
max-pool	192	56		3	2	1	192	28	588	0	1

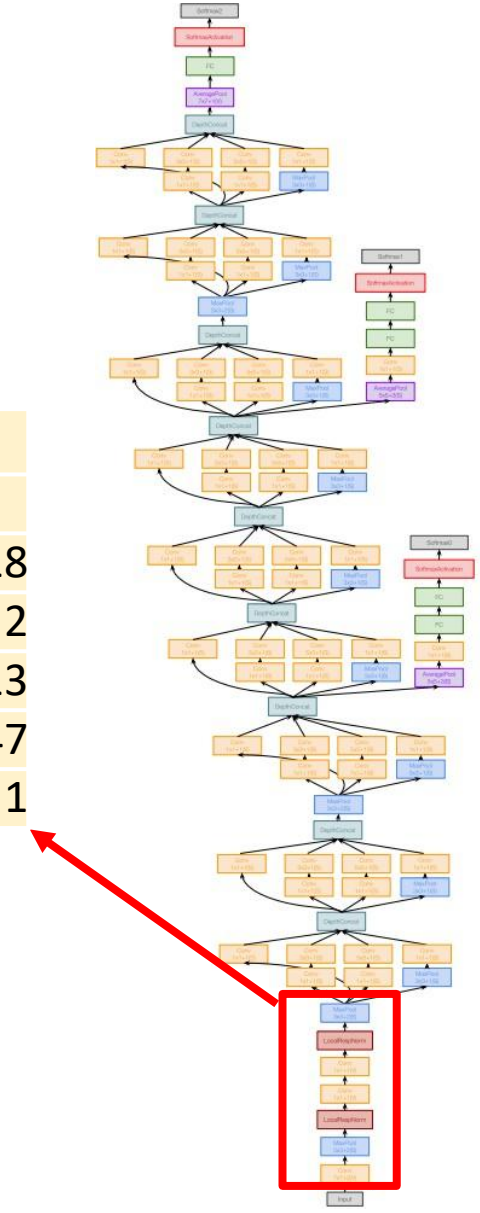


Total from 224 to 28 spatial resolution:
Memory: 7.5 MB
Params: 124K
MFLOP: 418

GoogLeNet: Aggressive Stem

Stem network at the start aggressively downsamples input
(Recall in VGG-16: Most of the compute was at the start)

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (K)	flop (M)
conv	3	224	64	7	2	3	64	112	3136	9	118
max-pool	64	112		3	2	1	64	56	784	0	2
conv	64	56	64	1	1	0	64	56	784	4	13
conv	64	56	192	3	1	1	192	56	2352	111	347
max-pool	192	56		3	2	1	192	28	588	0	1



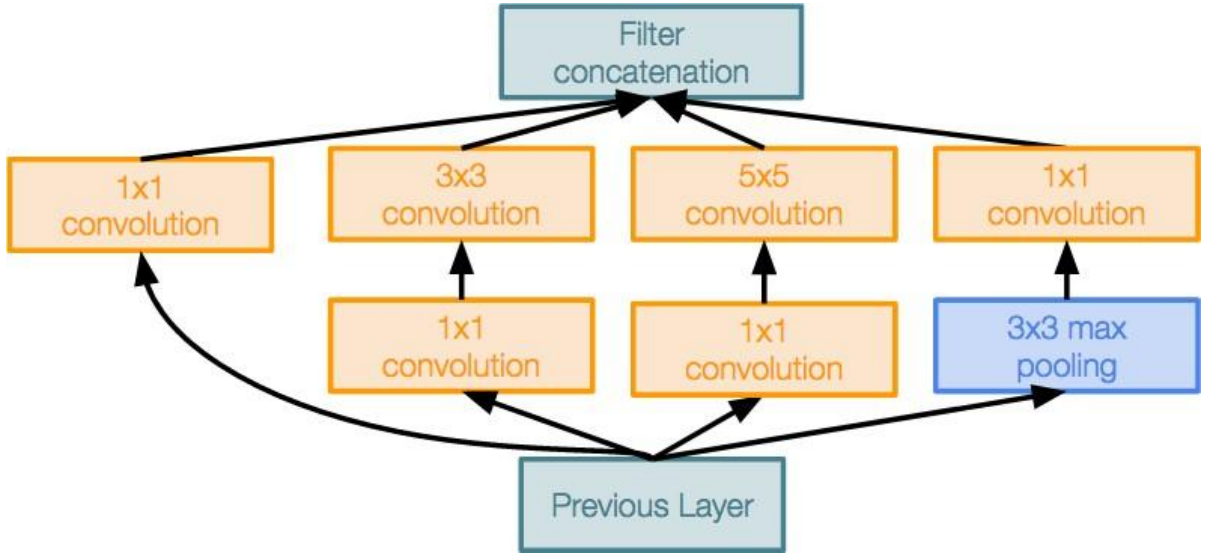
Total from 224 to 28 spatial resolution:
Memory: 7.5 MB
Params: 124K
MFLOP: 418

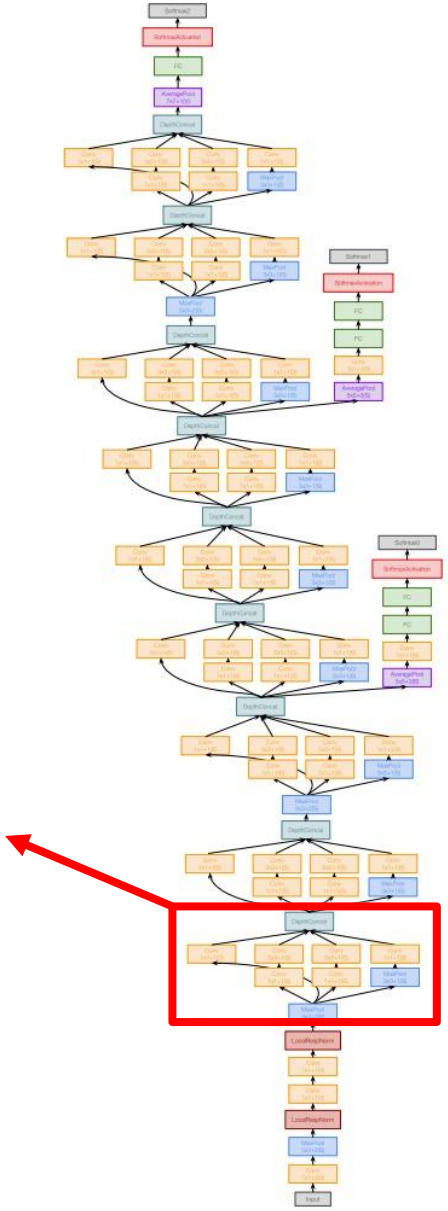
Compare VGG-16:
Memory: 42.9 MB (5.7x)
Params: 1.1M (8.9x)
MFLOP: 7485 (17.8x)

GoogLeNet: Inception Module

Inception module
Local unit with
parallel branches

Local structure repeated
many times throughout the
network





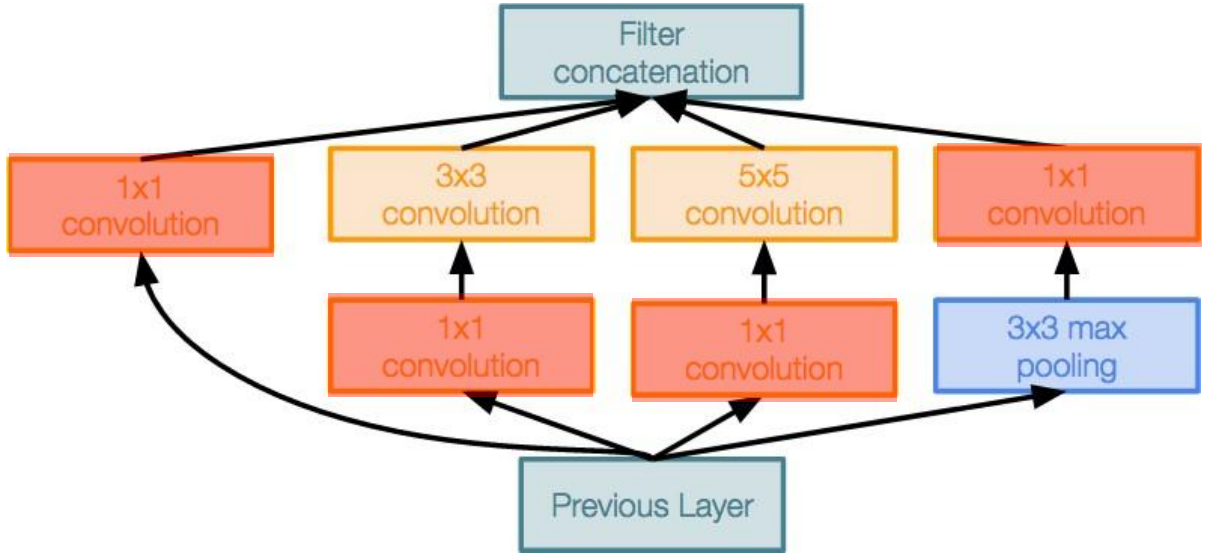
convolution filters of different sizes will handle
objects at multiple scales better

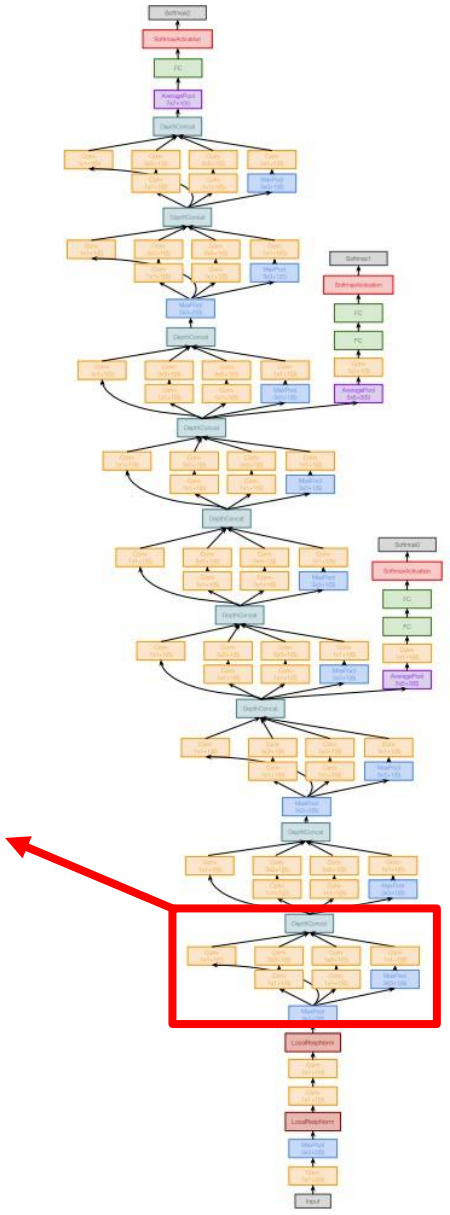
GoogLeNet: Inception Module

Inception module
Local unit with
parallel branches

Local structure repeated
many times throughout the
network

Uses 1x1 “Bottleneck”
layers to reduce channel
dimension

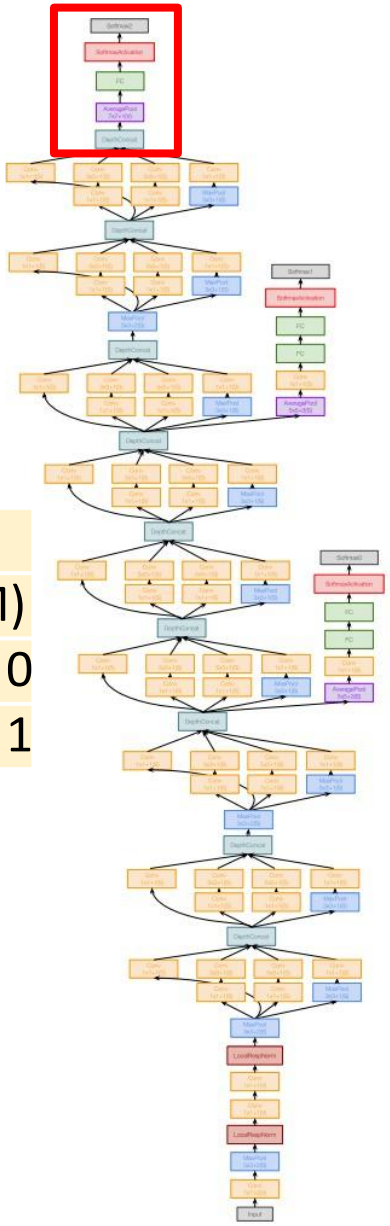




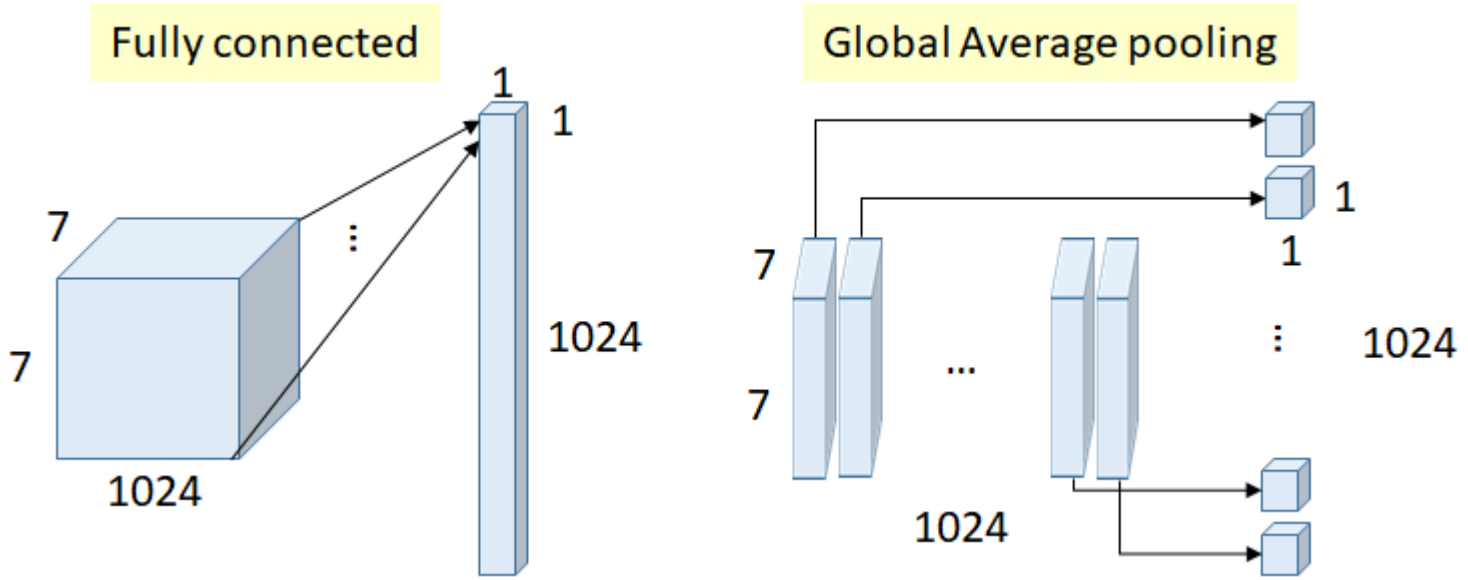
convolution filters of different sizes will handle
objects at multiple scales better

GoogLeNet: Global Average Pooling

Uses **global average pooling** to collapse spatial dimensions, and one linear layer to produce class scores (VGG-16: Most parameters were in FC layers)

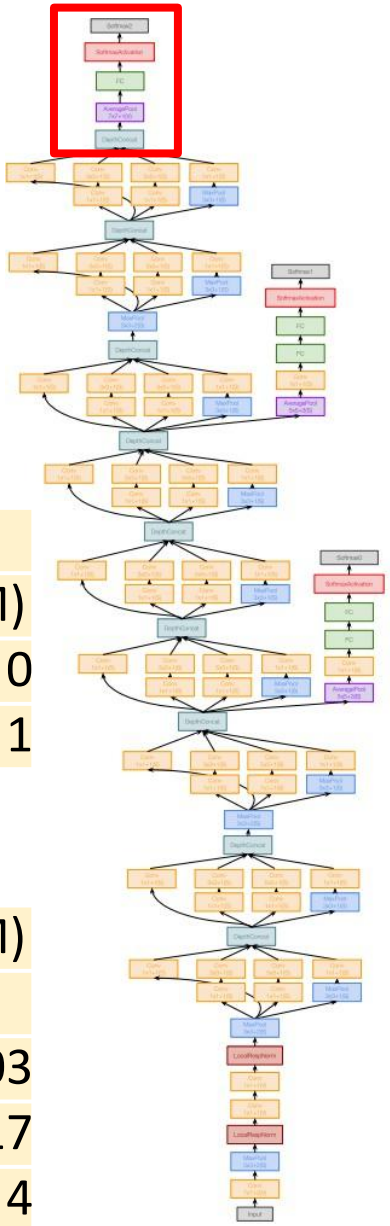


	Input size		Layer				Output size				
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (k)	flop (M)
avg-pool	1024	7		7	1	0	1024	1	4	0	0
fc	1024		1000				1000		0	1025	1



GoogLeNet: Global Average Pooling

Uses **global average pooling** to collapse spatial dimensions, and one linear layer to produce class scores (VGG-16: Most parameters were in FC layers)



	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 VGG16:

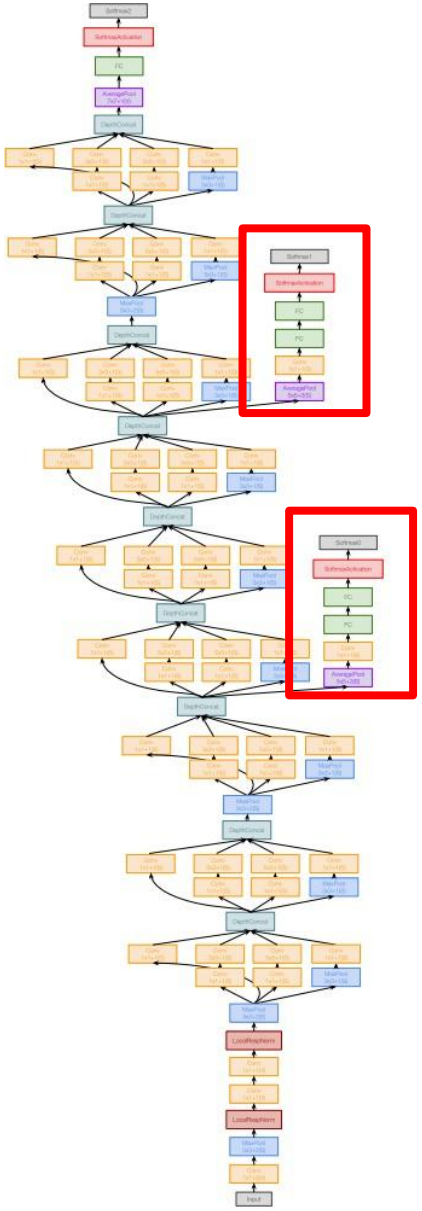
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (K)	flop (M)
flatten	512	7					25088		98		
fc6	25088			4096			4096		16	102760	103
fc7	4096			4096			4096		16	16777	17
fc8	4096			1000			1000		4	4096	4

GoogLeNet: Auxiliary Classifiers

Training using loss at the end of the network didn't work well:
Network is too deep, gradients don't propagate well

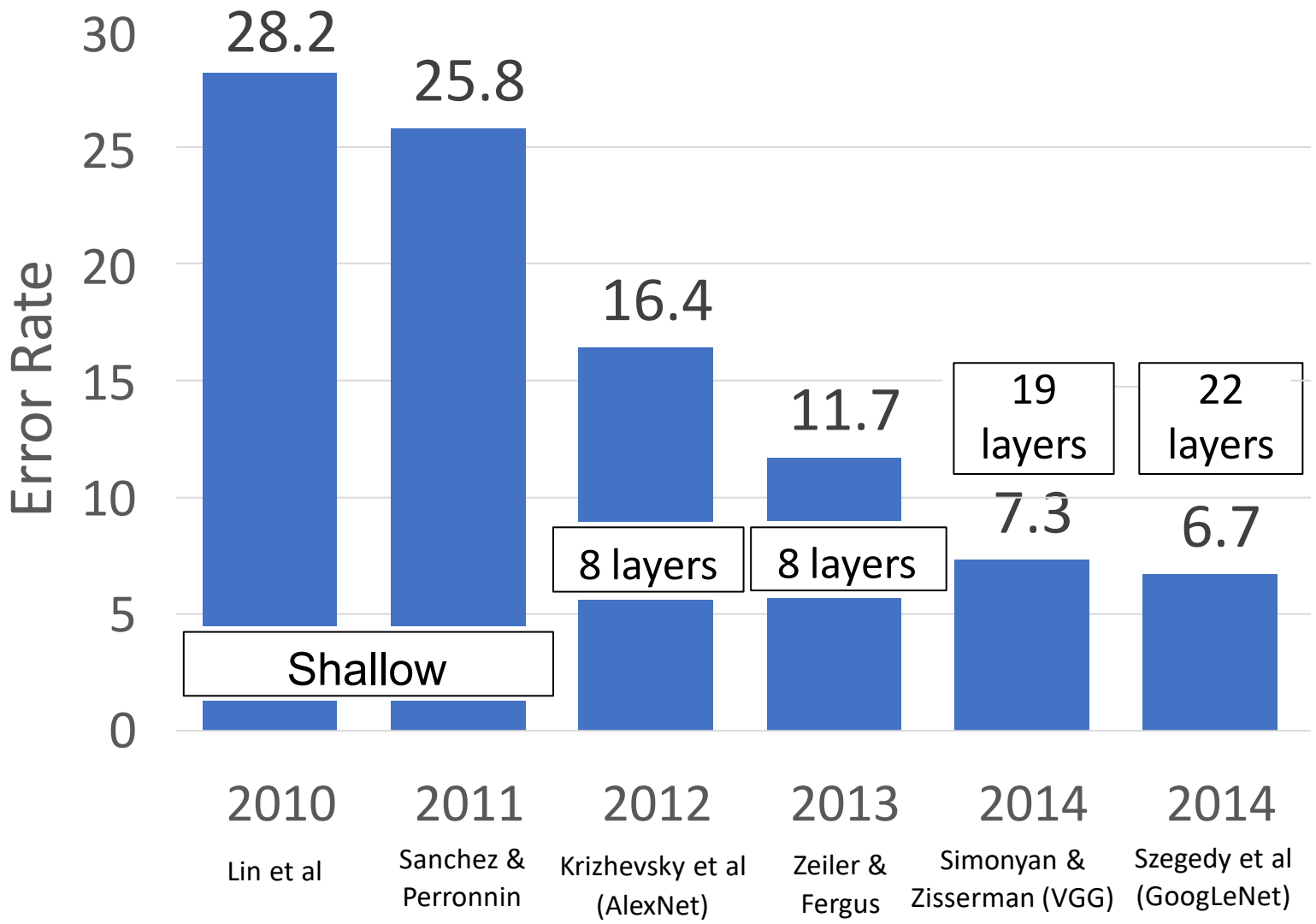
Attach "auxiliary classifiers" at several intermediate points in the network that also try to classify the image and receive loss (weighted with coefficient 0.3 only during training)

GoogLeNet was before batch normalization. With BatchNorm no longer need to use this method

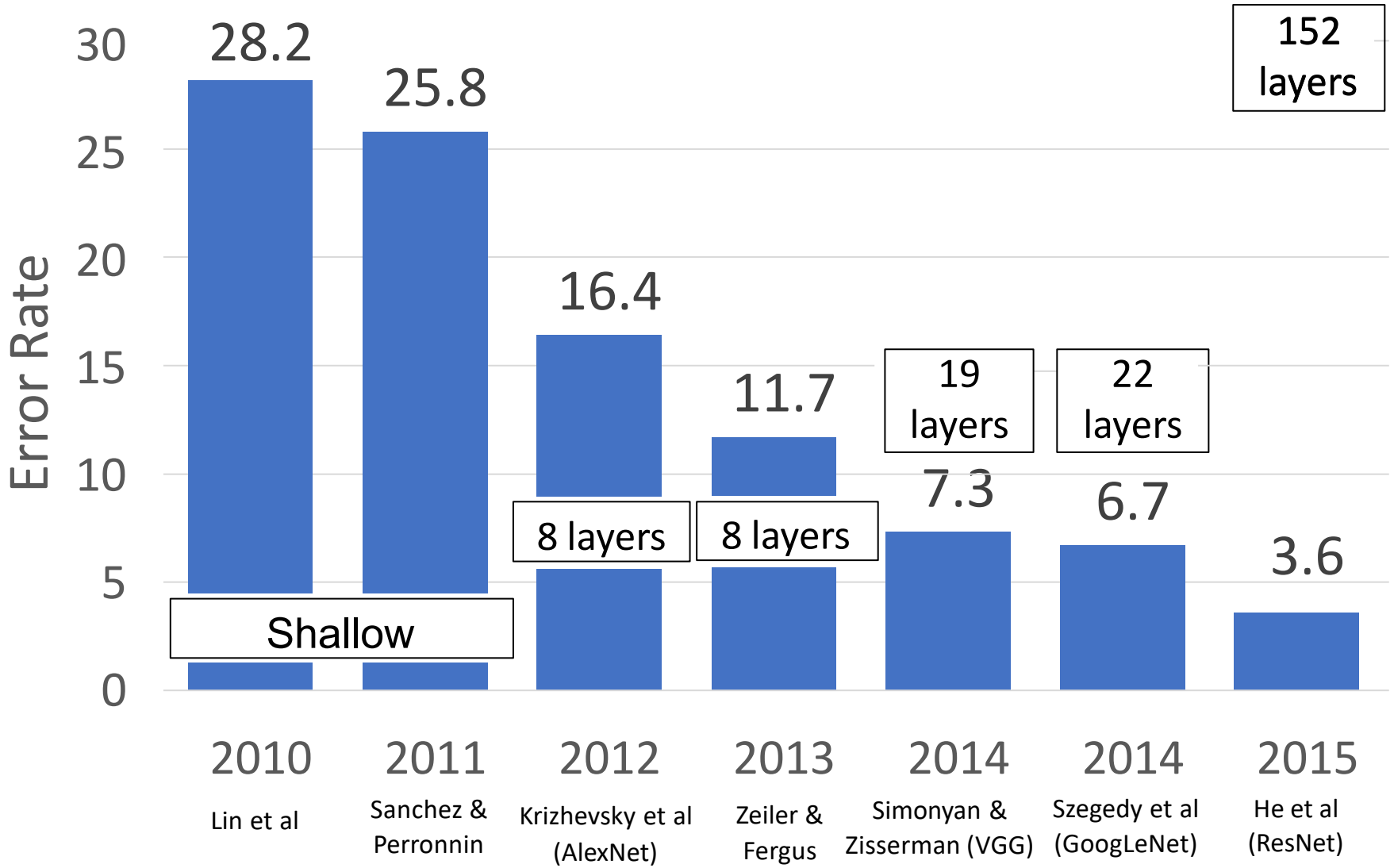


Szegedy et al, "Going deeper with convolutions", CVPR 2015

ImageNet Classification Challenge



ImageNet Classification Challenge



Residual Networks

Once we have Batch Normalization, we can train networks with 10+ layers.
Going deeper leads to **degradation problem**.

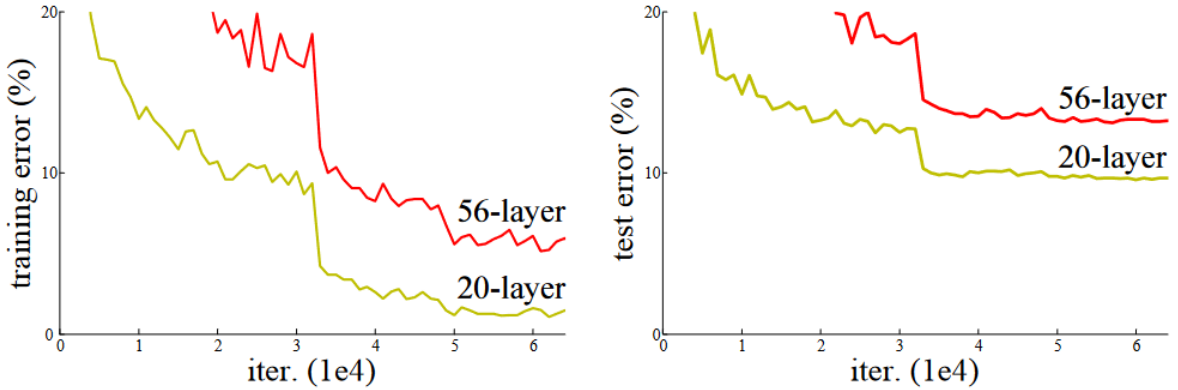
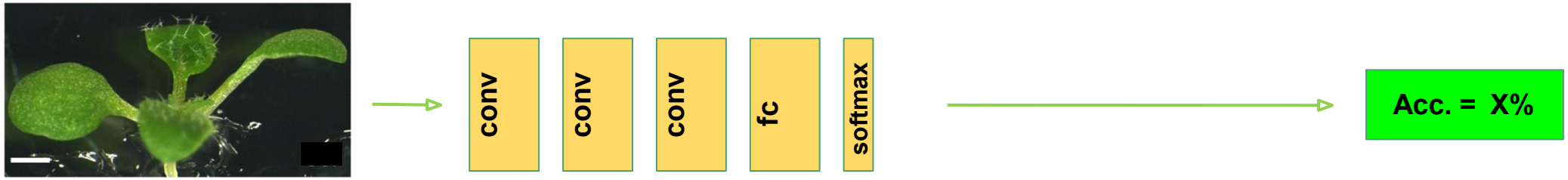


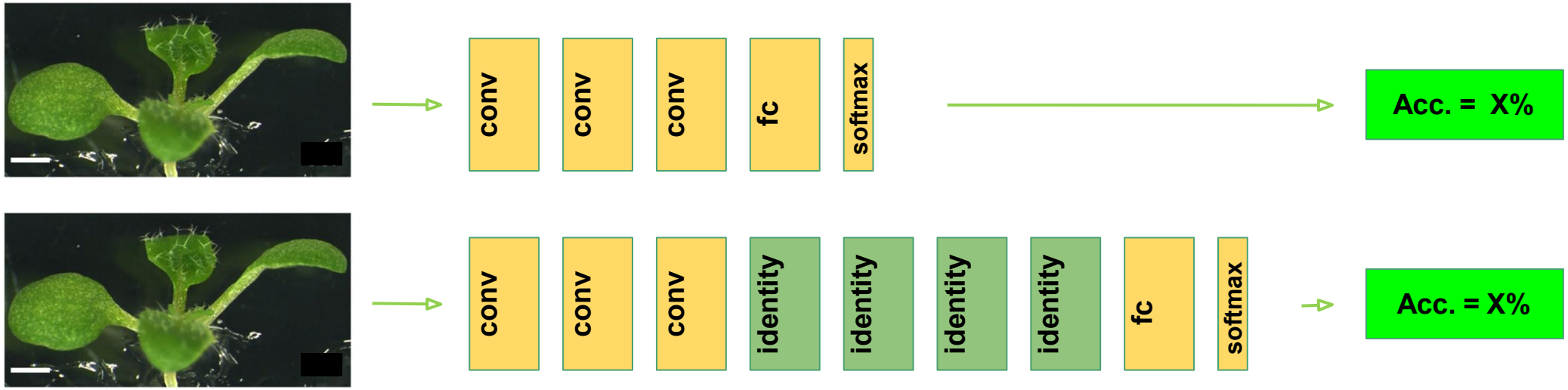
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

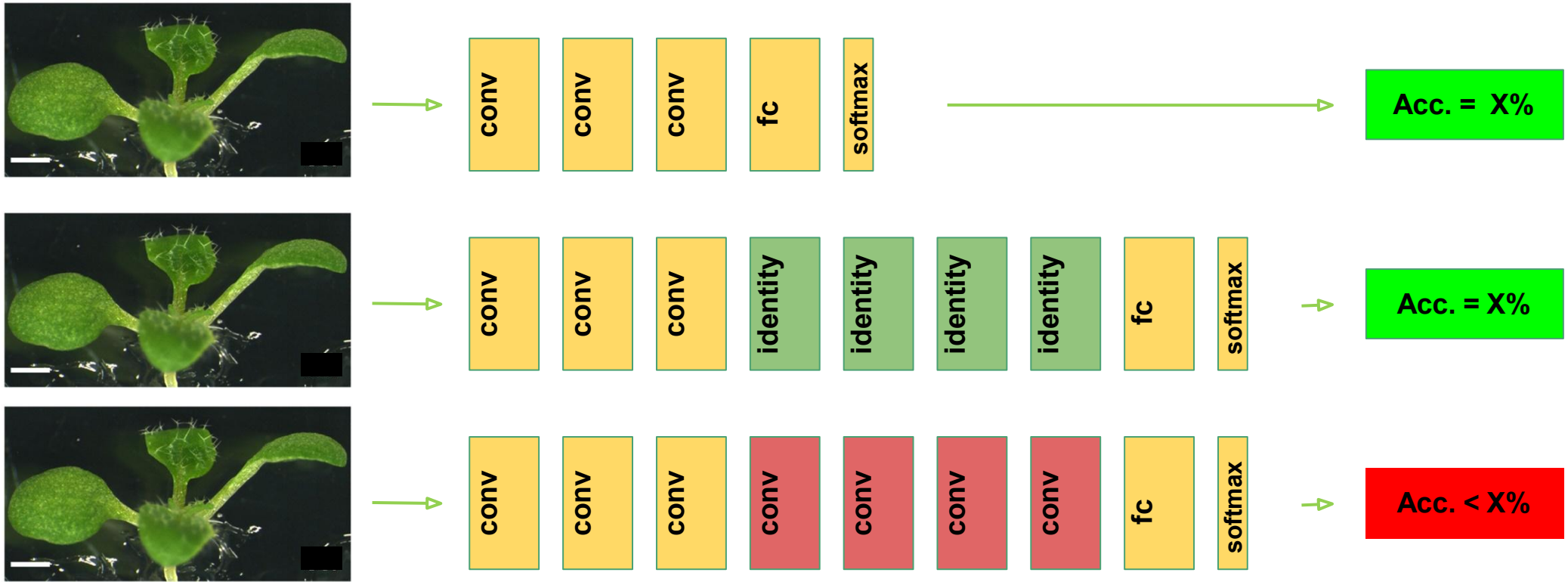
Degradation Problem



Degradation Problem



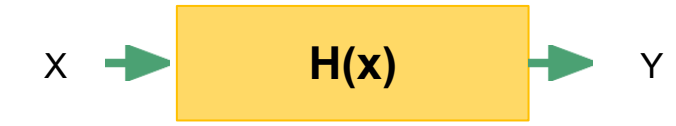
Degradation Problem

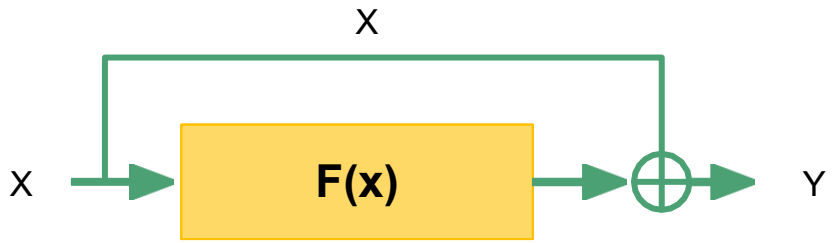


Residual Networks

H(x) is the true mapping function we want to learn
Let's define a function F(x), and learn it instead of H(x)

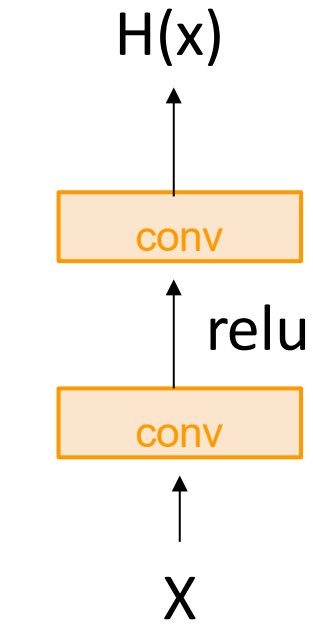
F(x) := H(x) - x



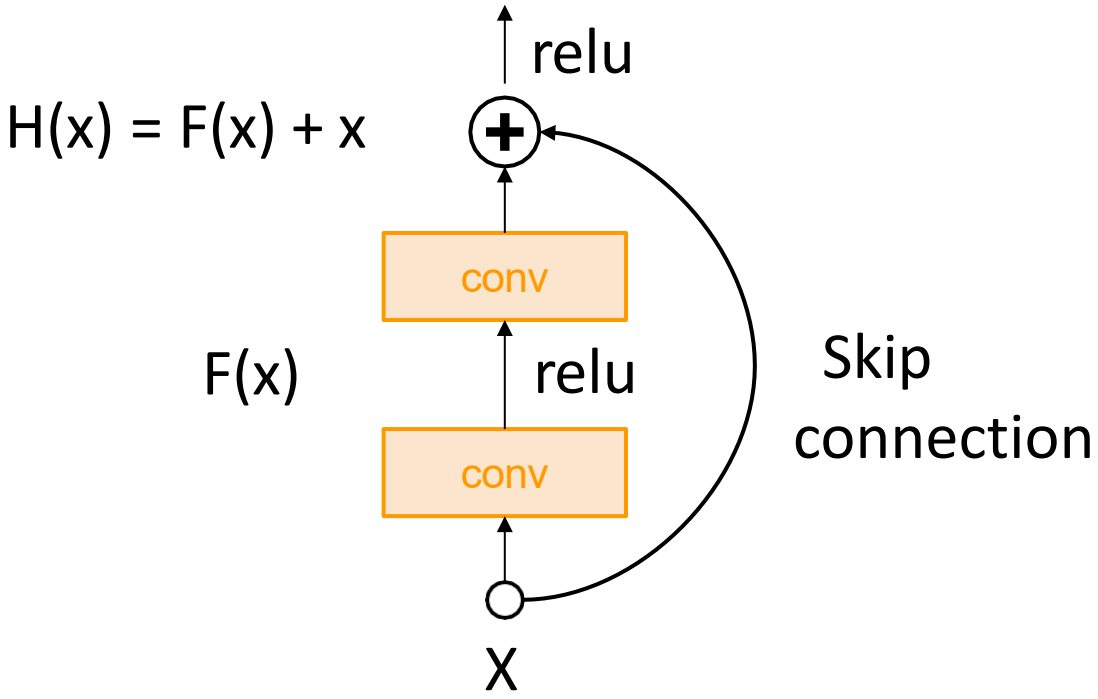


Residual Networks

Solution: Change the network to easier learn identity functions



"Plain" block



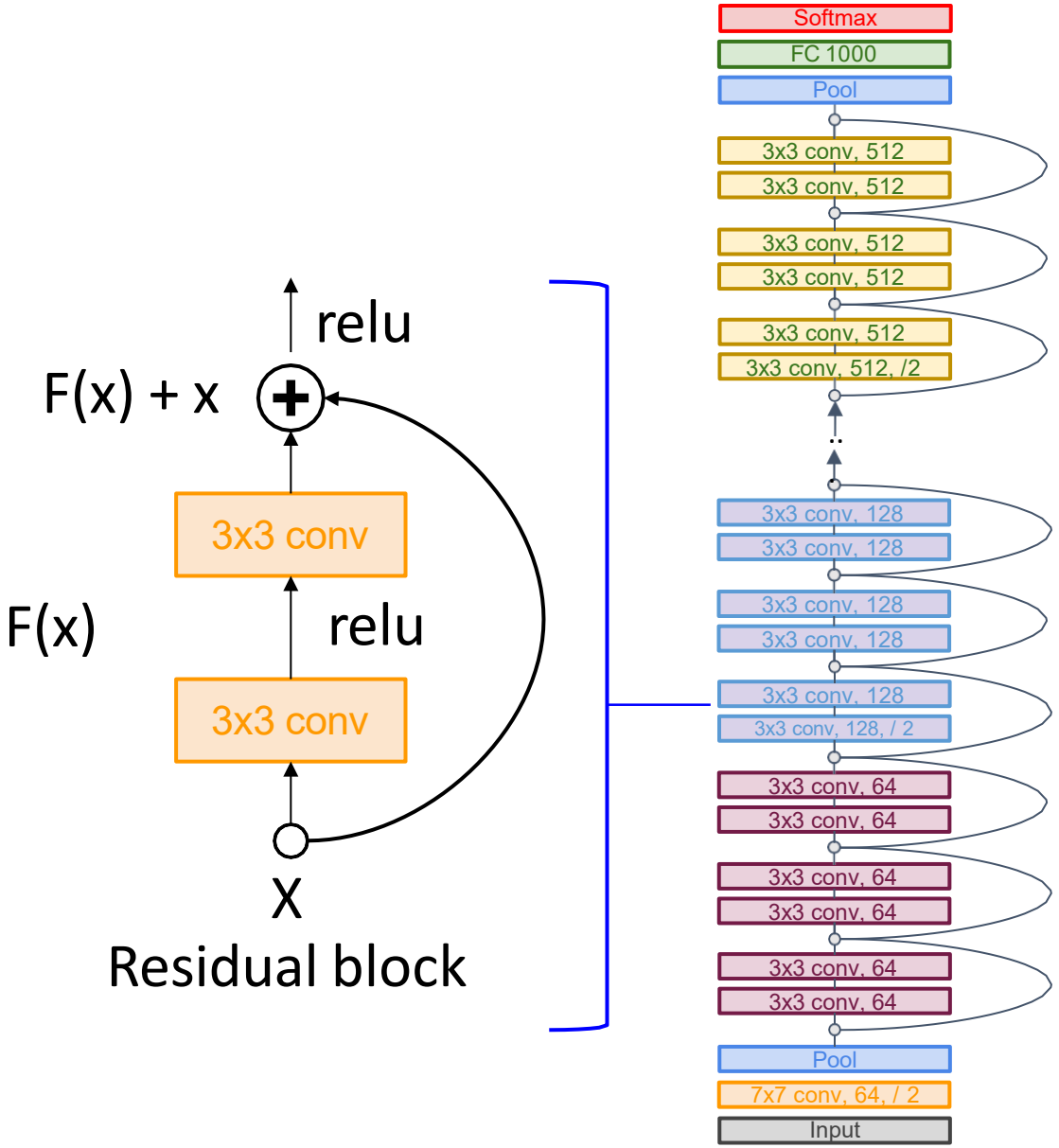
Residual Block

Residual Networks

A residual network is a stack of many residual blocks

Regular design, like VGG: each residual block has two 3x3 conv

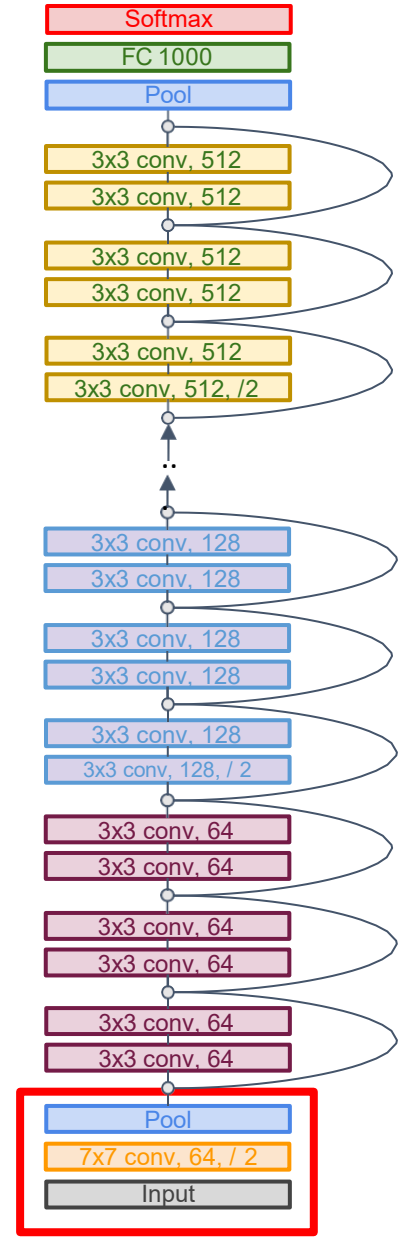
Network is divided into **stages**: the first block of each stage halves the resolution (with stride-2 conv) and doubles the number of channels



Residual Networks

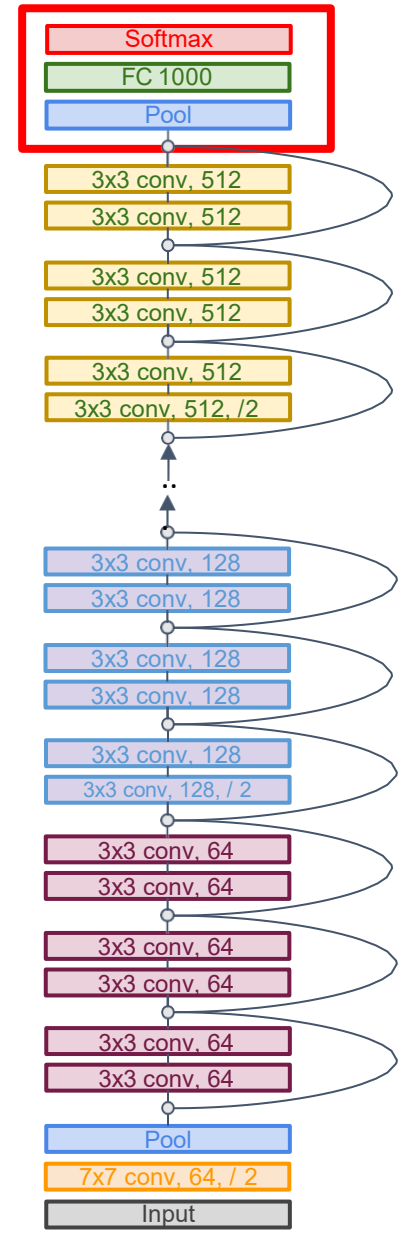
Uses the same **aggressive stem** as GoogleNet to downsample the input 4x before applying residual blocks:

	Input size		Layer				Output size				
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (k)	flop (M)
conv	3	224	64	7	2	3	64	112	3136	9	118
max-pool	64	112		3	2	1	64	56	784	0	2



Residual Networks

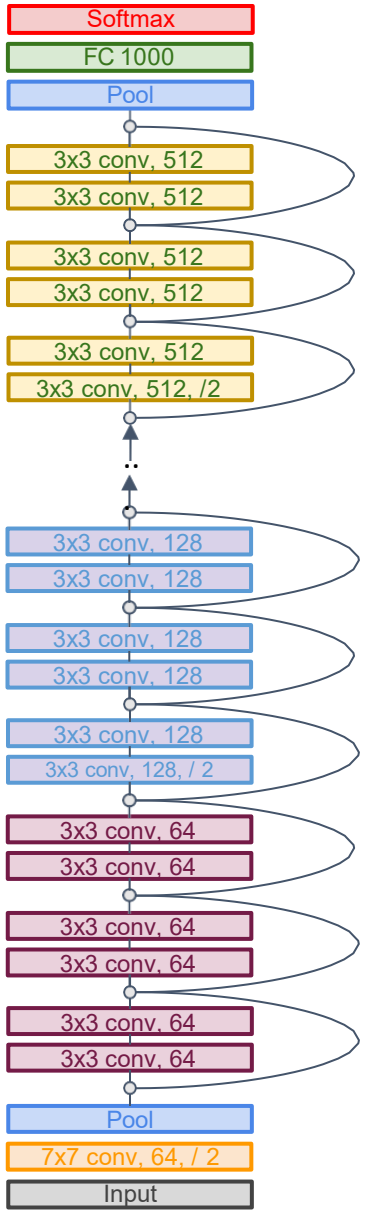
Like GoogLeNet, no big fully-connected-layers: instead use **global average pooling** and a single linear layer at the end



Residual Networks

ResNet-18:
Stem: 1 conv layer
Stage 1 (C=64): 2 res. block = 4 conv
Stage 2 (C=128): 2 res. block = 4 conv
Stage 3 (C=256): 2 res. block = 4 conv
Stage 4 (C=512): 2 res. block = 4 conv
Linear

ImageNet top-5 error: 10.92
GFLOP: 1.8



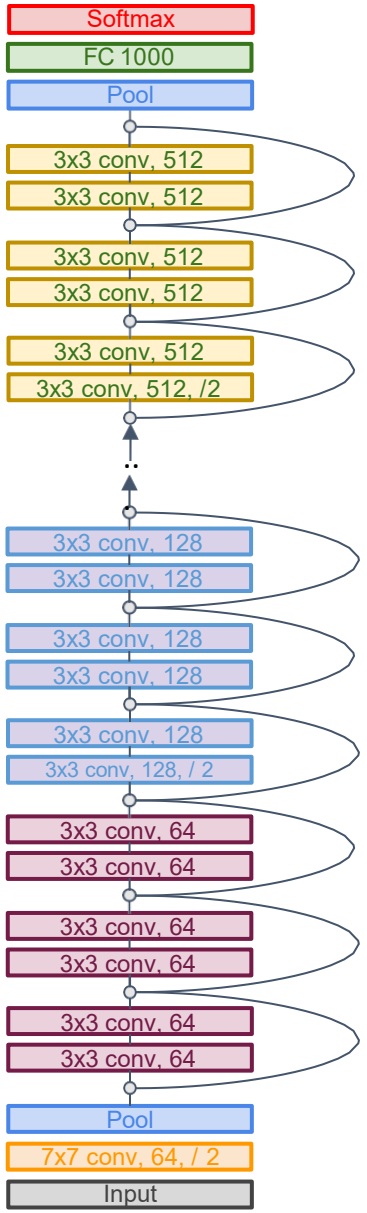
Residual Networks

ResNet-18:
Stem: 1 conv layer
Stage 1 (C=64): 2 res. block = 4 conv
Stage 2 (C=128): 2 res. block = 4 conv
Stage 3 (C=256): 2 res. block = 4 conv
Stage 4 (C=512): 2 res. block = 4 conv
Linear

ImageNet top-5 error: 10.92
GFLOP: 1.8

ResNet-34:
Stem: 1 conv layer
Stage 1: 3 res. block = 6 conv
Stage 2: 4 res. block = 8 conv
Stage 3: 6 res. block = 12 conv
Stage 4: 3 res. block = 6 conv
Linear

ImageNet top-5 error: 8.58
GFLOP: 3.6



Residual Networks

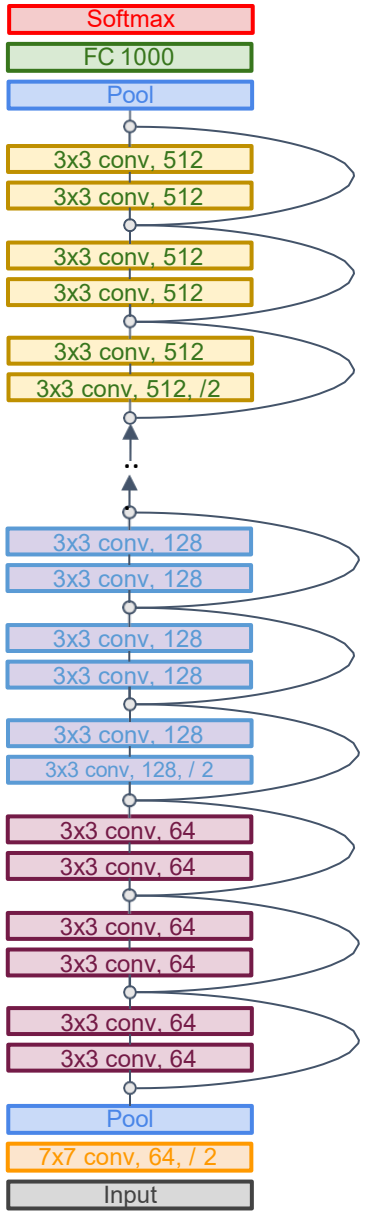
ResNet-18:
Stem: 1 conv layer
Stage 1 (C=64): 2 res. block = 4 conv
Stage 2 (C=128): 2 res. block = 4 conv
Stage 3 (C=256): 2 res. block = 4 conv
Stage 4 (C=512): 2 res. block = 4 conv
Linear

ImageNet top-5 error: 10.92
GFLOP: 1.8

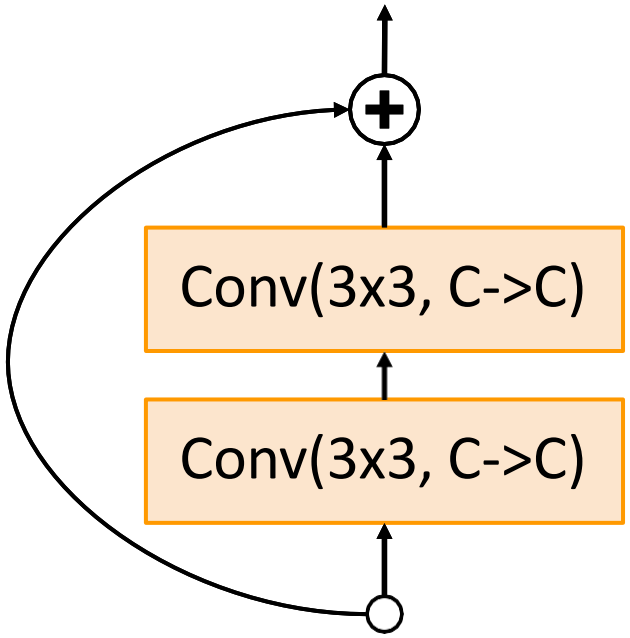
ResNet-34:
Stem: 1 conv layer
Stage 1: 3 res. block = 6 conv
Stage 2: 4 res. block = 8 conv
Stage 3: 6 res. block = 12 conv
Stage 4: 3 res. block = 6 conv
Linear

ImageNet top-5 error: 8.58
GFLOP: 3.6

VGG-16:
ImageNet top-5 error: 9.62
GFLOP: 13.6

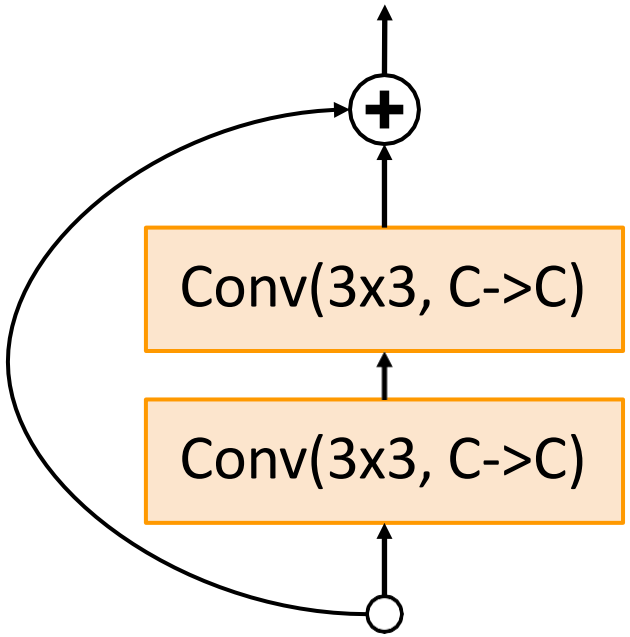


Residual Networks: Basic Block



“Basic”
Residual block

Residual Networks: Basic Block



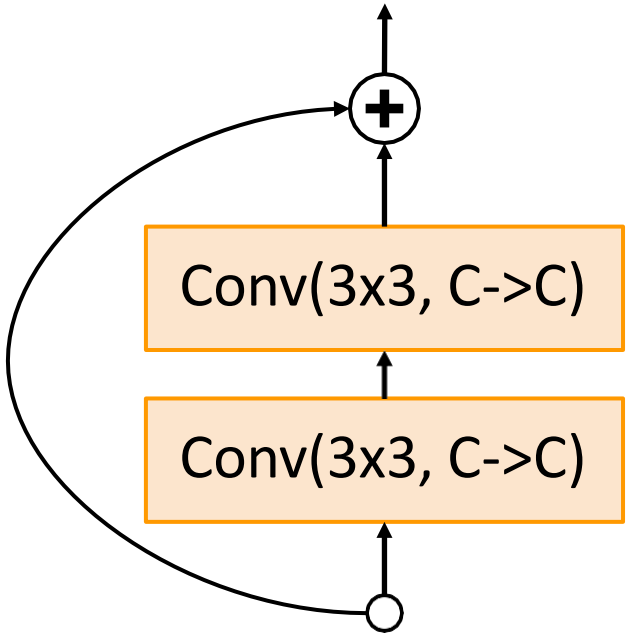
FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

"Basic"
Residual block

Total FLOPs:
$18HWC^2$

Residual Networks: Bottleneck Block

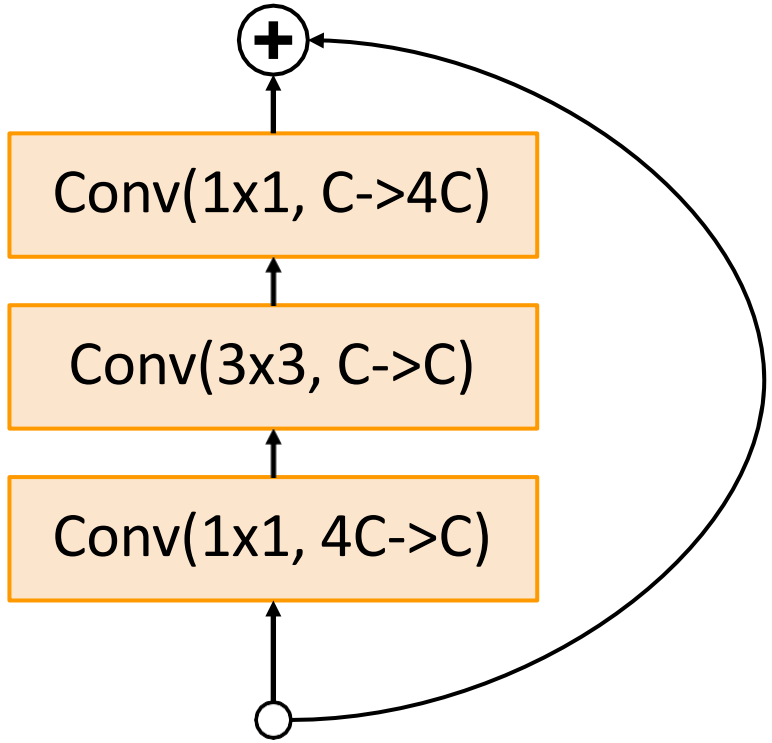


FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

"Basic"
Residual block

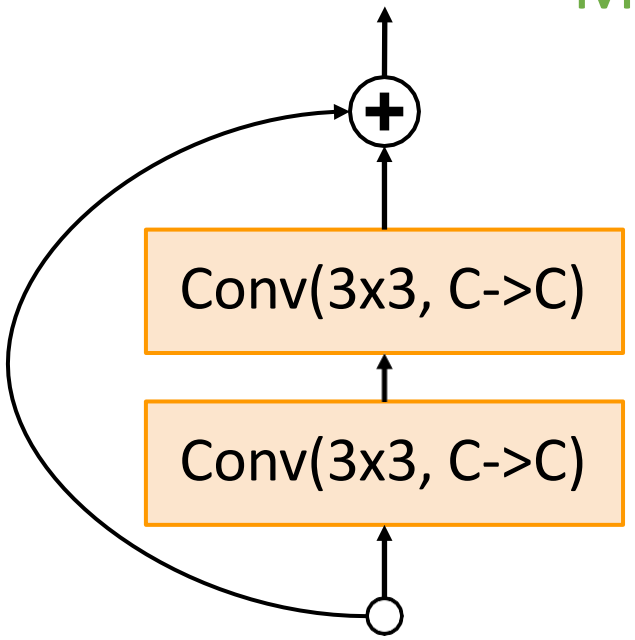
Total FLOPs:
$18HWC^2$



"Bottleneck"
Residual block

Residual Networks: Bottleneck Block

More layers, less computational cost



FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

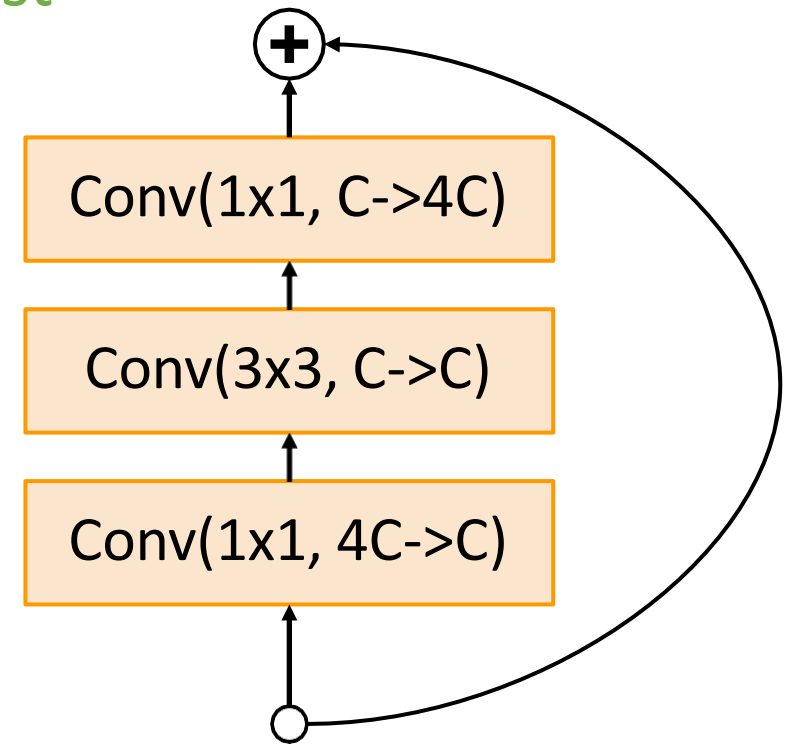
"Basic"
Residual block

Total FLOPs:
$18HWC^2$

FLOPs: $4HWC^2$

FLOPs: $9HWC^2$

FLOPs: $4HWC^2$

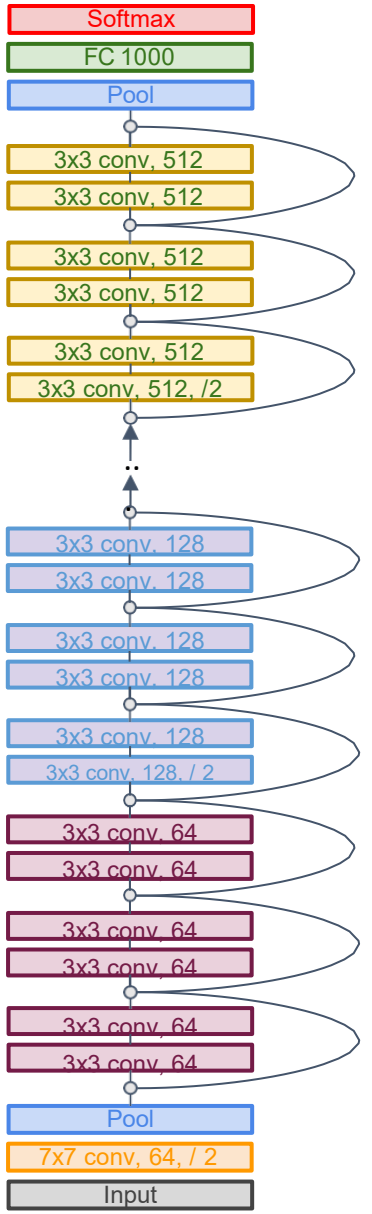


Total FLOPs:
$17HWC^2$

"Bottleneck"
Residual block

Residual Networks

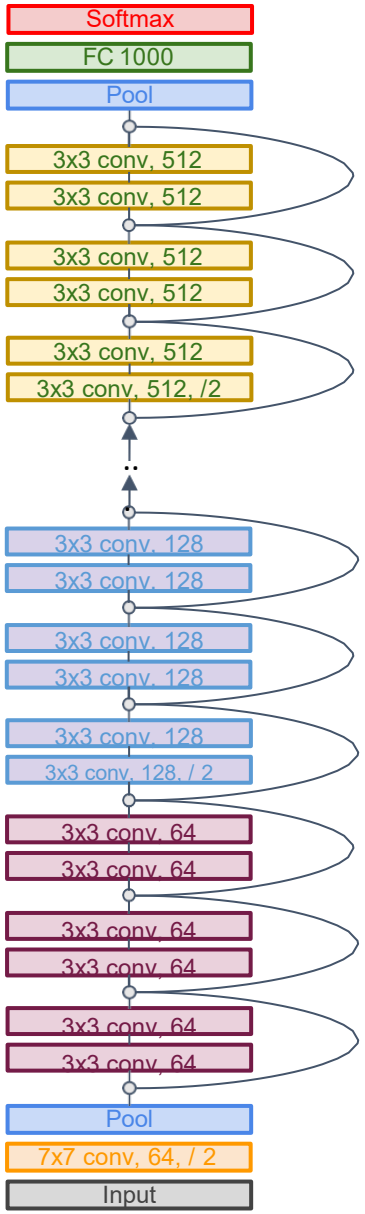
			Stage 1		Stage 2		Stage 3		Stage 4				
	Block type	Stem layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	FC layers	GFLOP	ImageNet top-5 error
ResNet-18	Basic	1	2	4	2	4	2	4	2	4	1	1.8	10.92
ResNet-34	Basic	1	3	6	4	8	6	12	3	6	1	3.6	8.58



Residual Networks

ResNet-50 is the same as ResNet-34, but replaces Basic blocks with Bottleneck Blocks.

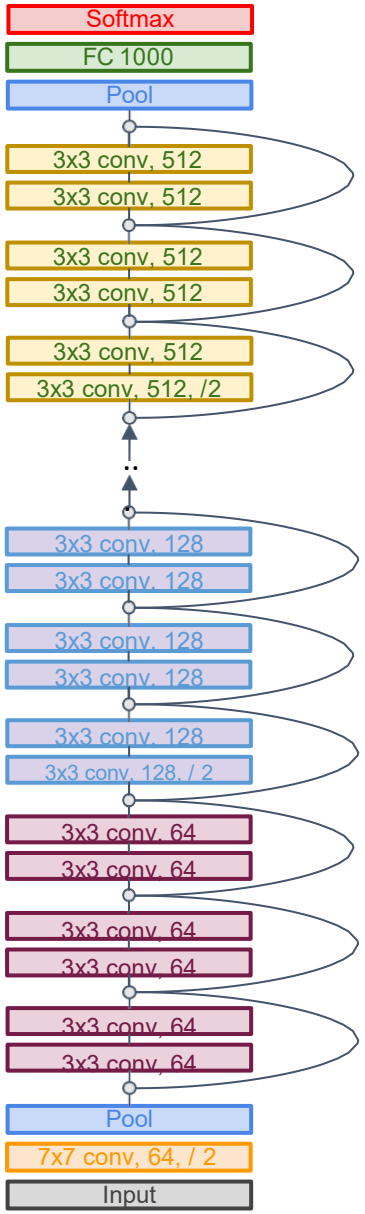
			Stage 1		Stage 2		Stage 3		Stage 4				
	Block type	Stem layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	FC layers	GFLOP	ImageNet top-5 error
ResNet-18	Basic	1	2	4	2	4	2	4	2	4	1	1.8	10.92
ResNet-34	Basic	1	3	6	4	8	6	12	3	6	1	3.6	8.58
ResNet-50	Bottle	1	3	9	4	12	6	18	3	9	1	3.8	7.13



Residual Networks

Deeper ResNet-101 and ResNet-152 models are more accurate, but also more computationally intensive

			Stage 1		Stage 2		Stage 3		Stage 4				
	Block type	Stem layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	FC layers	GFLOP	ImageNet top-5 error
ResNet-18	Basic	1	2	4	2	4	2	4	2	4	1	1.8	10.92
ResNet-34	Basic	1	3	6	4	8	6	12	3	6	1	3.6	8.58
ResNet-50	Bottle	1	3	9	4	12	6	18	3	9	1	3.8	7.13
ResNet-101	Bottle	1	3	9	4	12	23	69	3	9	1	7.6	6.44
ResNet-152	Bottle	1	3	9	8	24	36	108	3	9	1	11.3	5.94



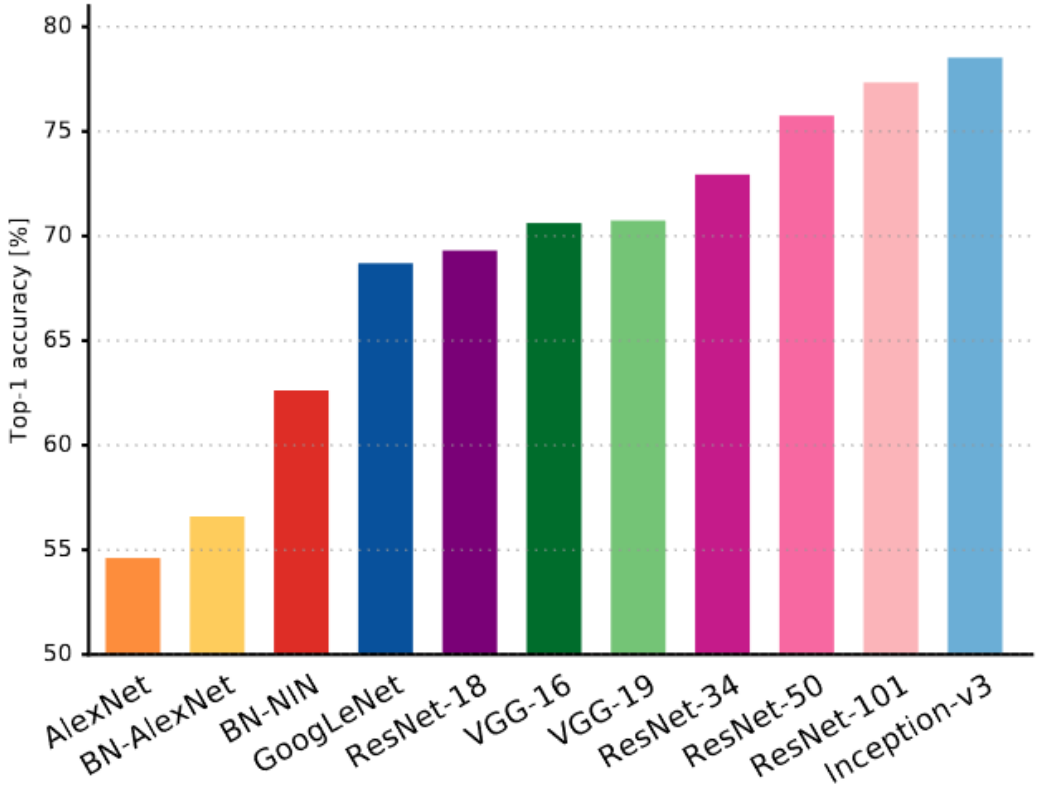
Residual Networks

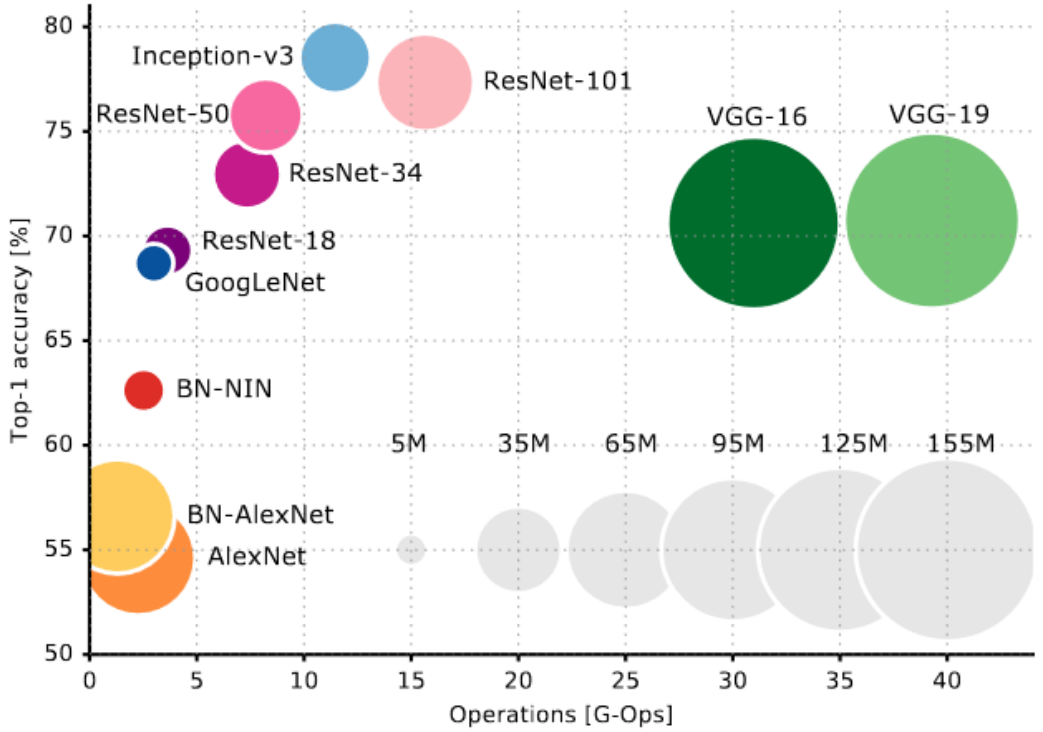
- Able to train very deep networks
- Deeper networks do better than shallow networks (as expected)
- Swept 1st place in all ILSVRC and COCO 2015 competitions
- Widely used today
- Over 100,000 citations

MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: "*Ultra-deep*" (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

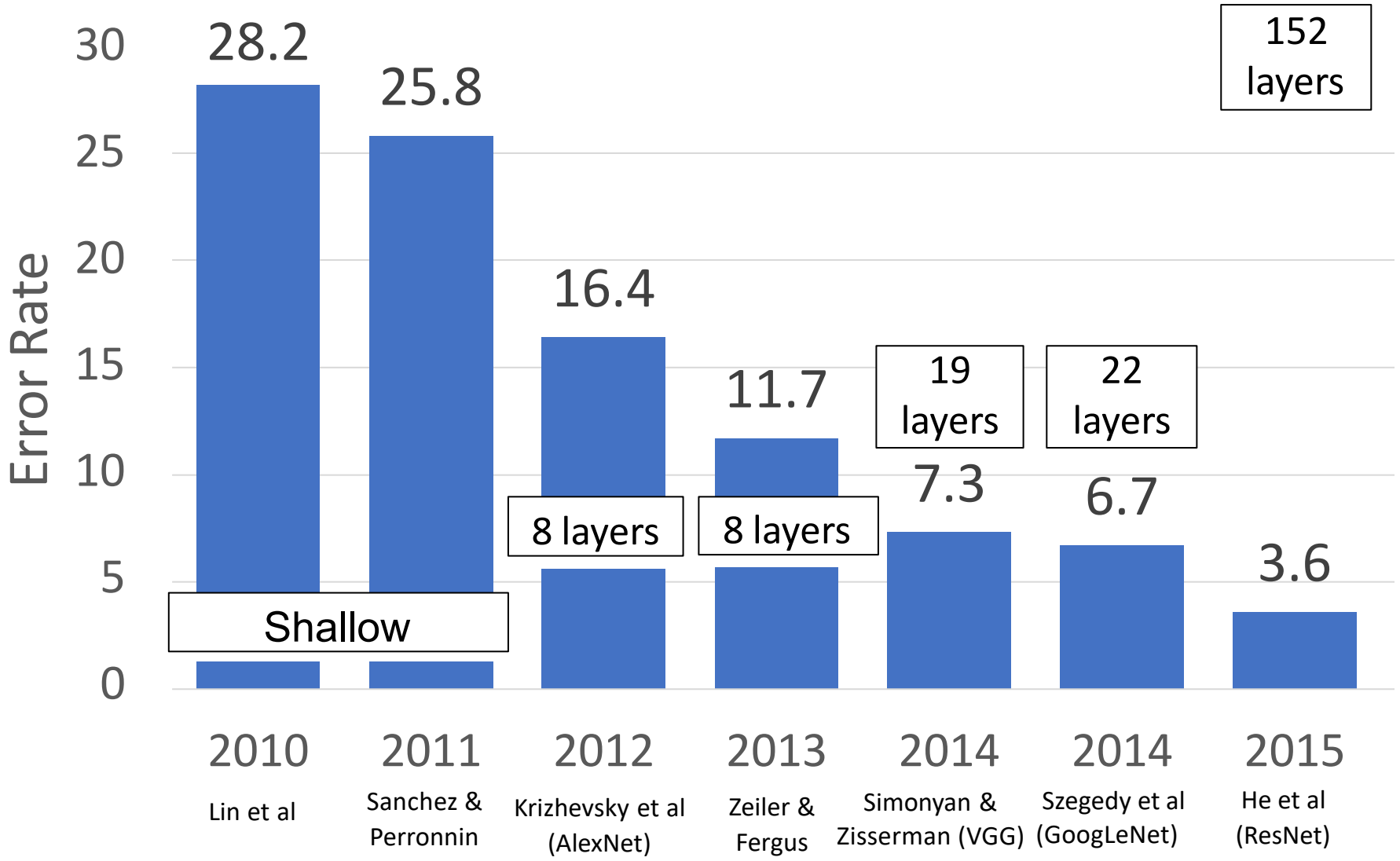
Comparing Complexity



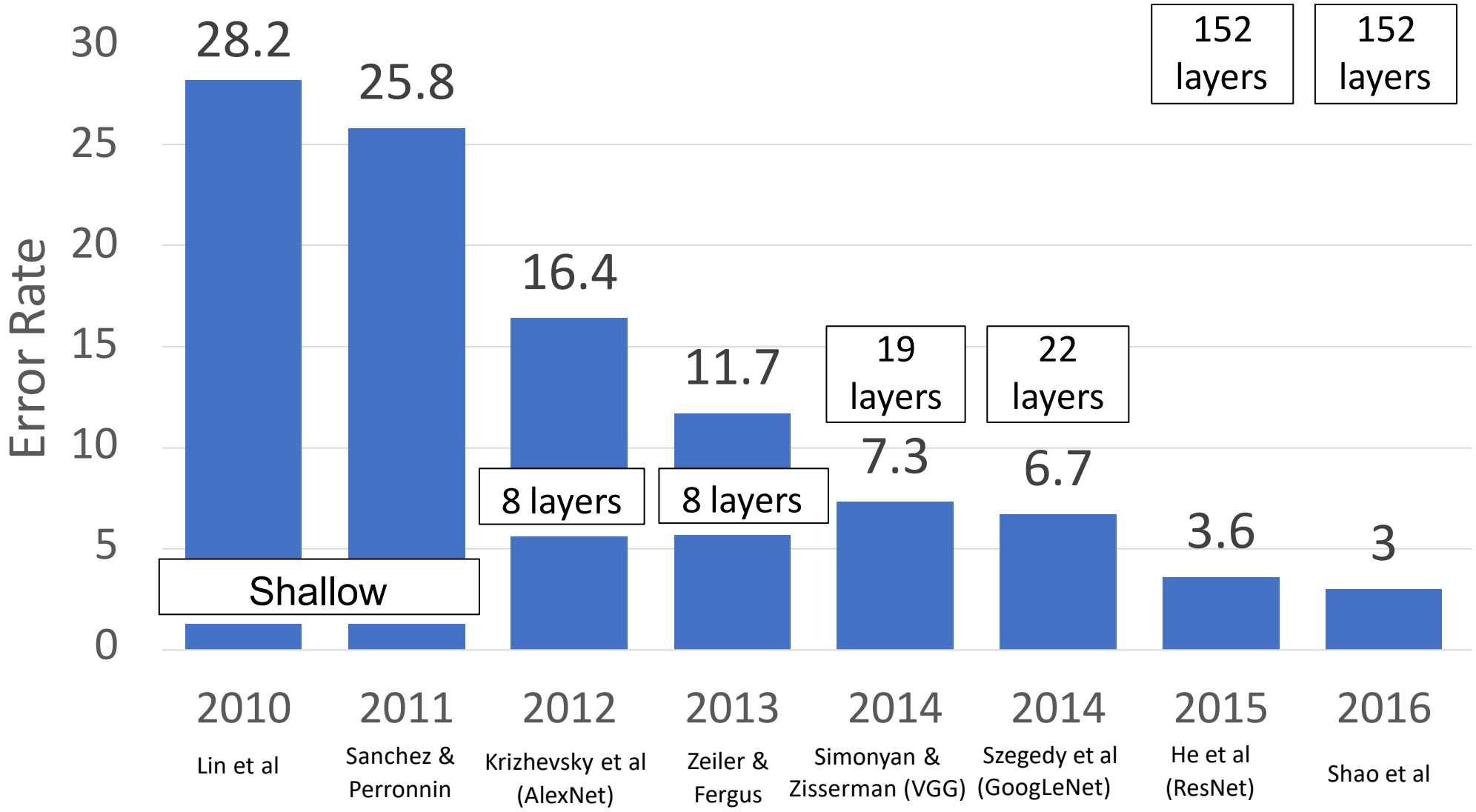


Canziani et al, “An analysis of deep neural network models for practical applications”, 2017

ImageNet Classification Challenge



ImageNet Classification Challenge



Overview

1. **One time setup**
   Activation functions, data preprocessing, weight initialization, regularization
2. **Training dynamics**
   Learning rate schedules; hyperparameter optimization
3. **After training**
   Model ensembles, transfer learning, large-batch training

Model Ensembles

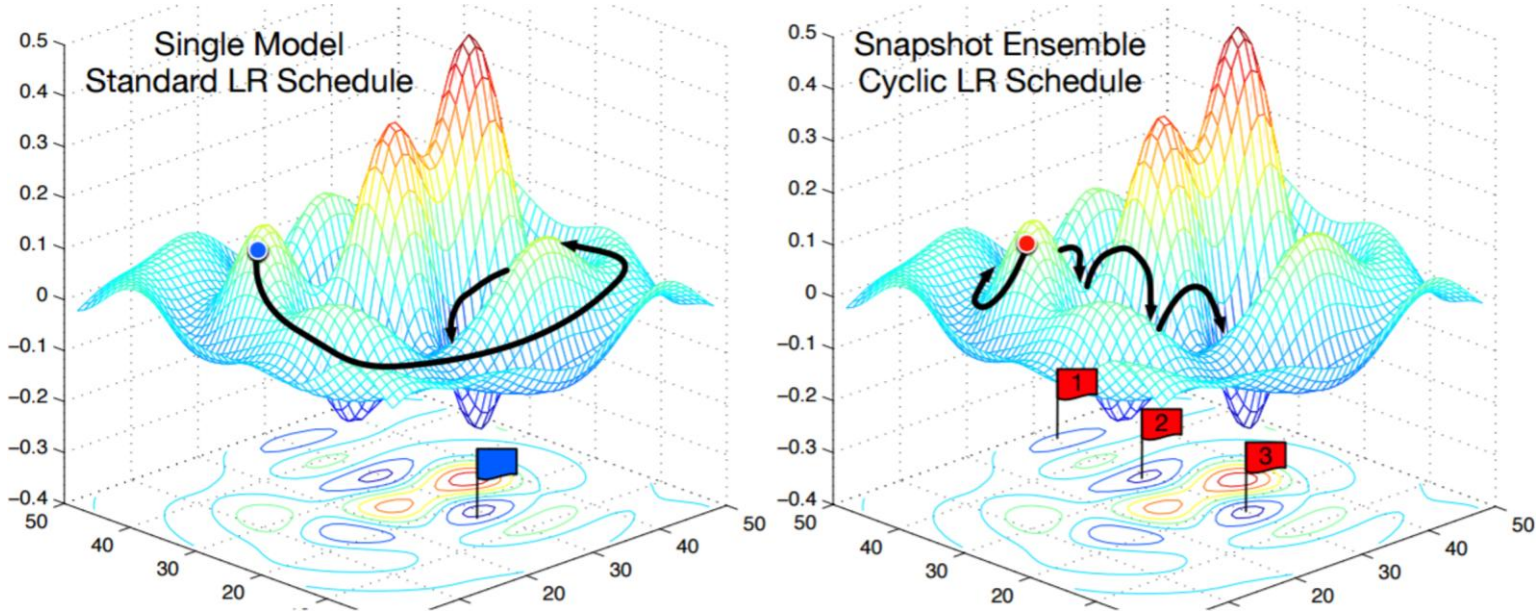
1. Train multiple independent models
2. At test time average their results

   (Take average of predicted probability distributions, then choose argmax)

Small amount of extra performance

Model Ensembles

Instead of training independent models, use multiple snapshots of a single model during training



Loshchilov and Hutter, “SGDR: Stochastic gradient descent with restarts”, arXiv 2016
Huang et al, “Snapshot ensembles: train 1, get M for free”, ICLR 2017

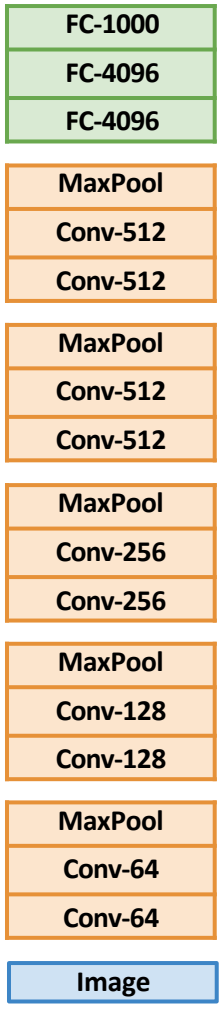
Transfer Learning

Transfer Learning

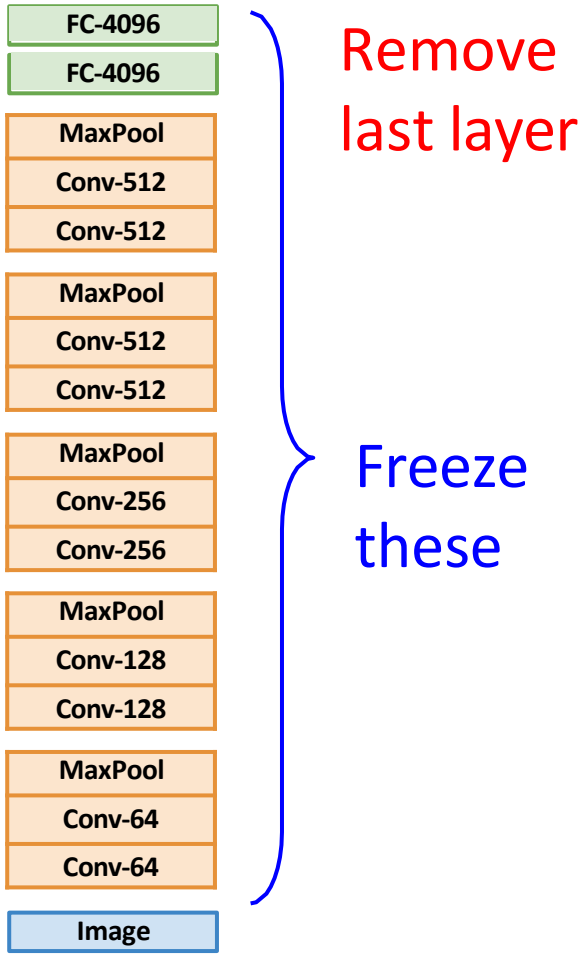
- Reuse AI model that has been trained on one task as a starting point to train for another
- Transfer learning is widely used because it dramatically reduces the time and/or data required to train in comparison to a model trained from scratch

Transfer Learning with CNNs

1. Train on Imagenet



2. Use CNN as a feature extractor



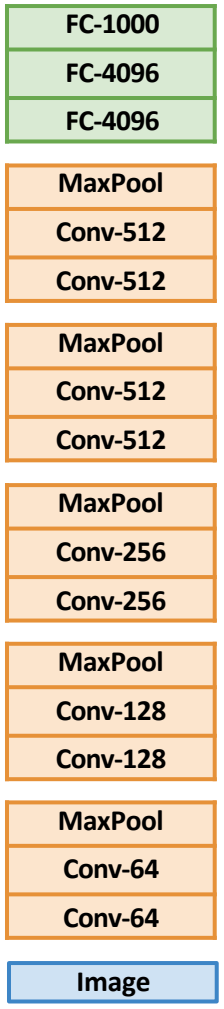
Remove last layer

Freeze these

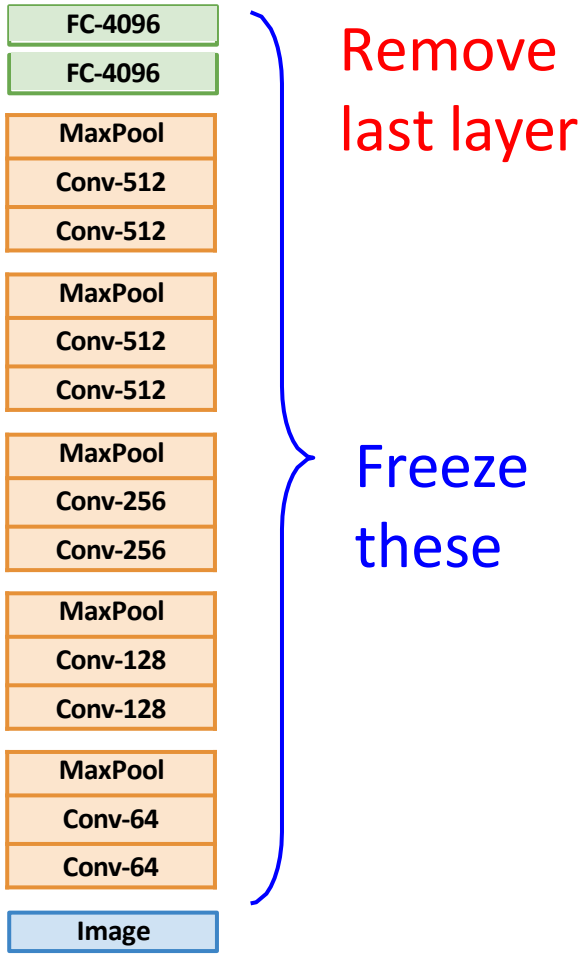
Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

Transfer Learning with CNNs

1. Train on Imagenet



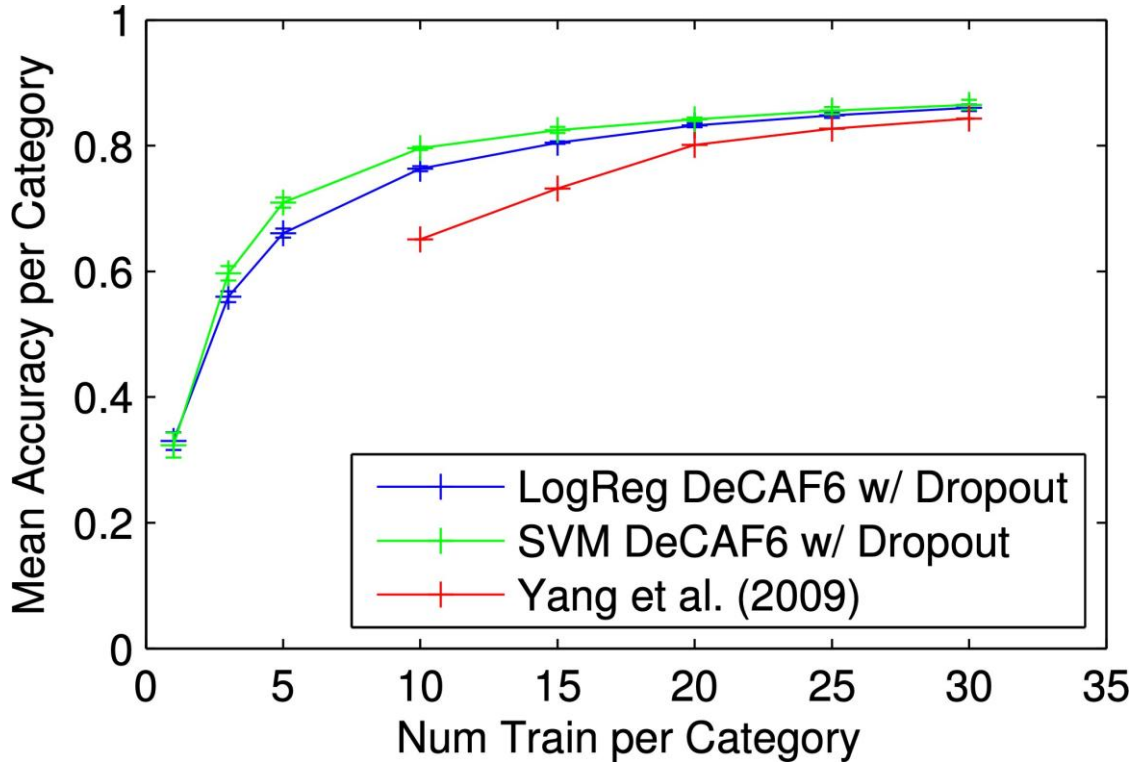
2. Use CNN as a feature extractor



Remove last layer

Freeze these

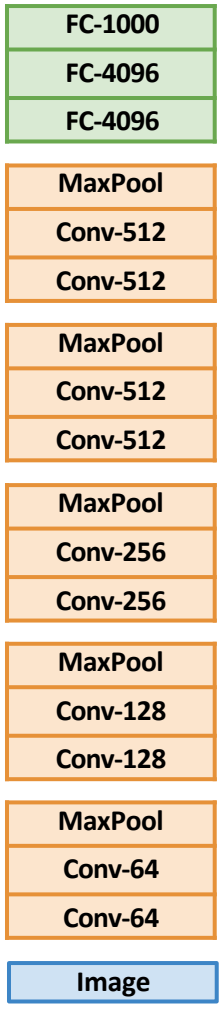
Classification on Caltech-101



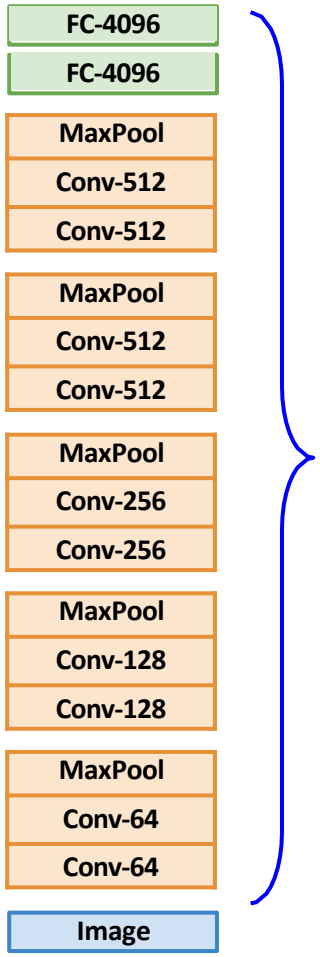
Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

Transfer Learning with CNNs

1. Train on Imagenet



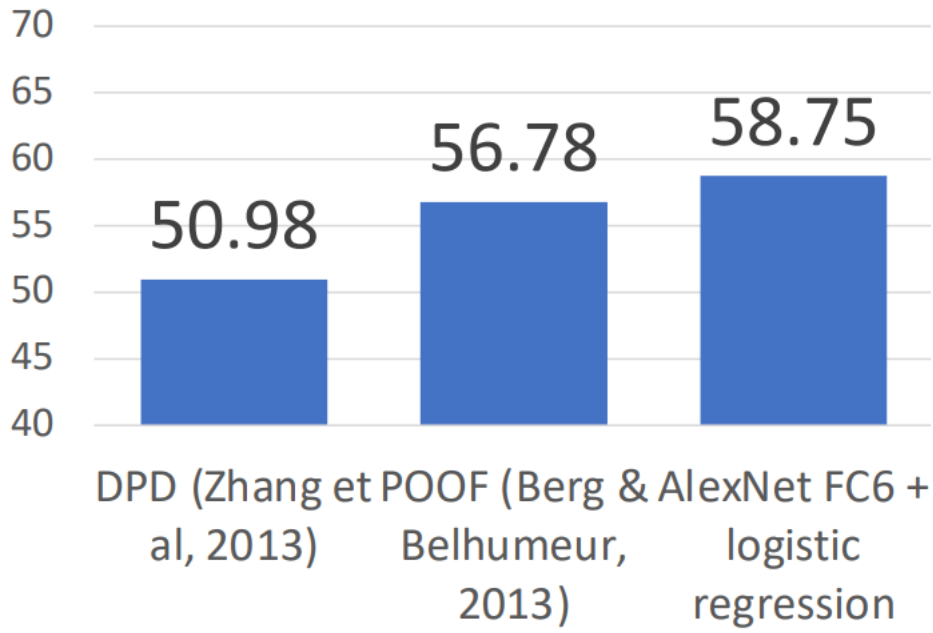
2. Use CNN as a feature extractor



Remove last layer

Freeze these

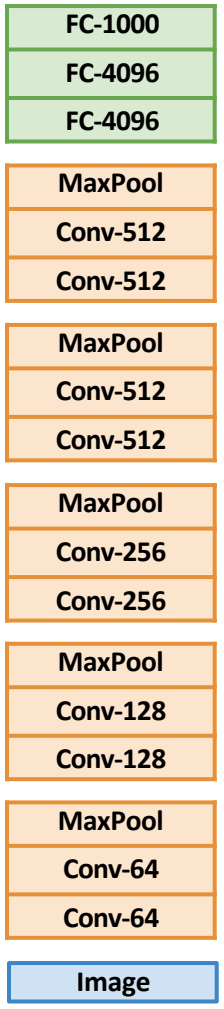
Bird Classification on Caltech-UCSD



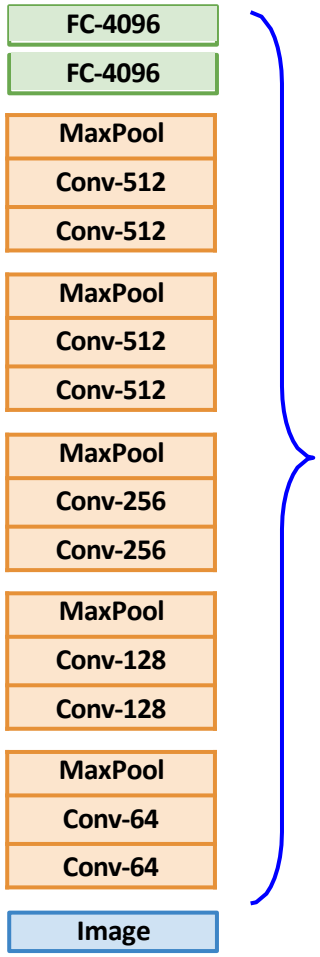
Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

Transfer Learning with CNNs

1. Train on Imagenet



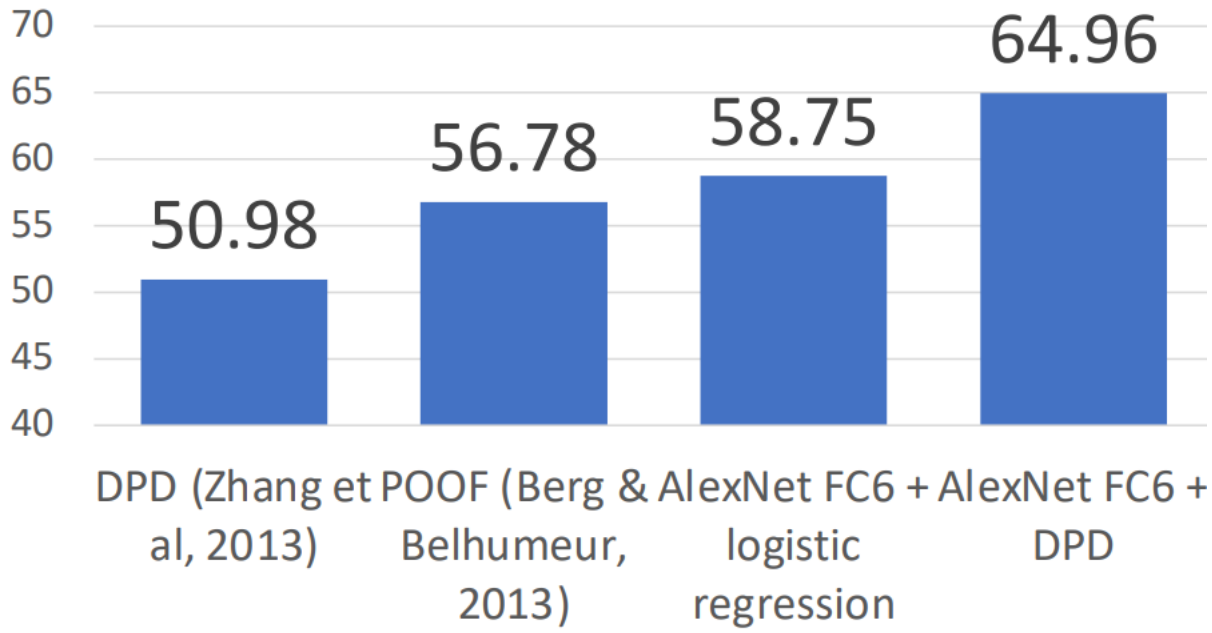
2. Use CNN as a feature extractor



Remove last layer

Freeze these

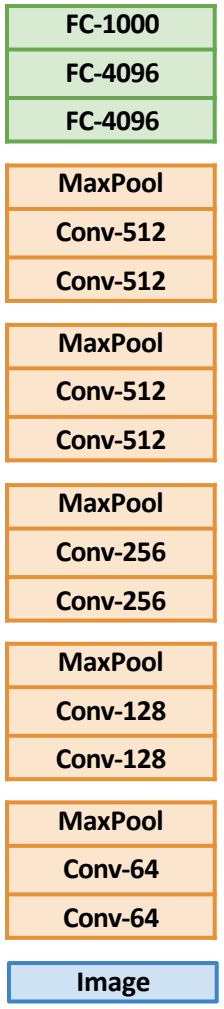
Bird Classification on Caltech-UCSD



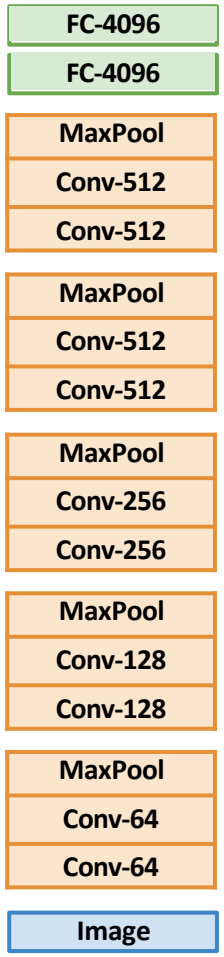
Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

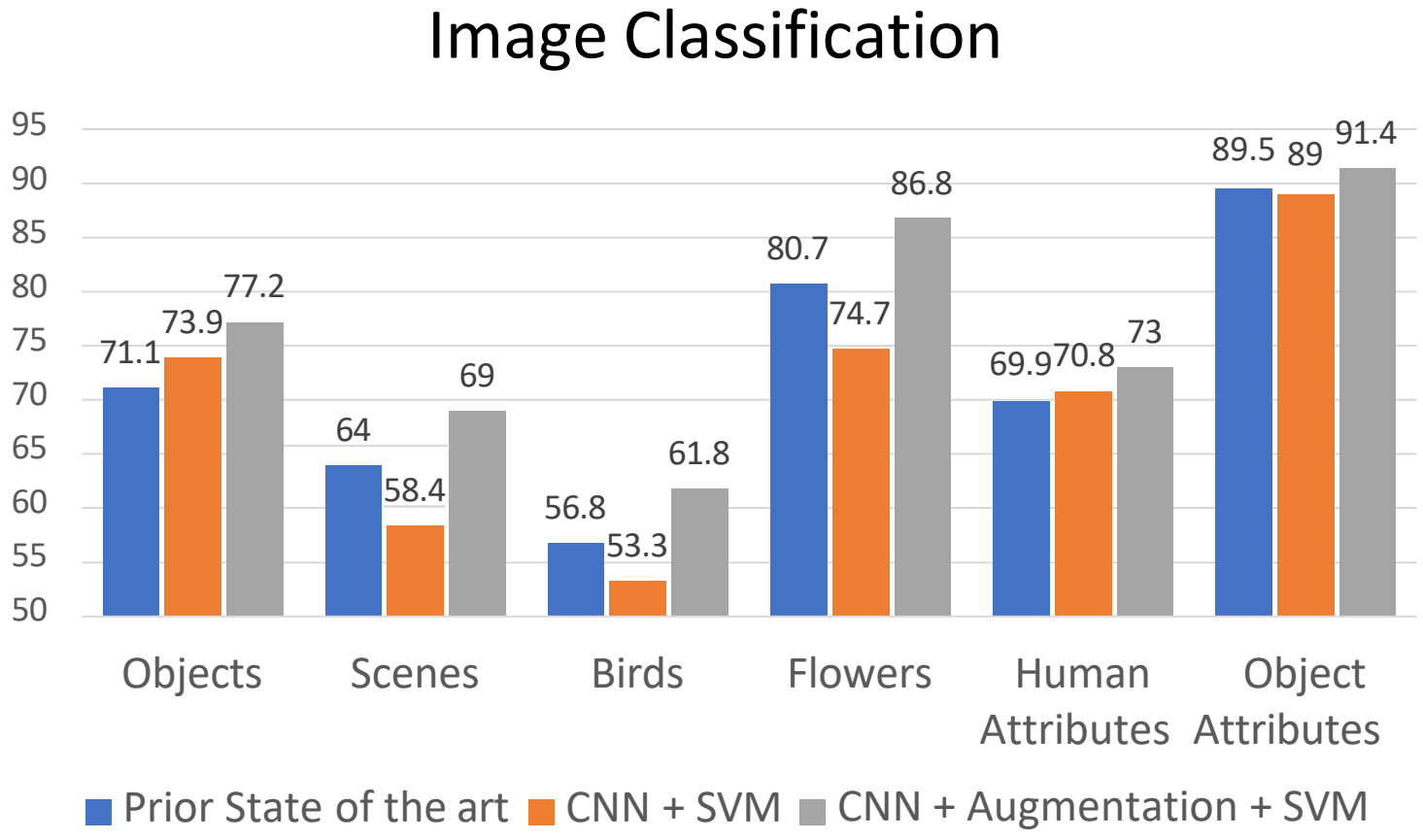
Transfer Learning with CNNs

1. Train on Imagenet



2. Use CNN as a feature extractor

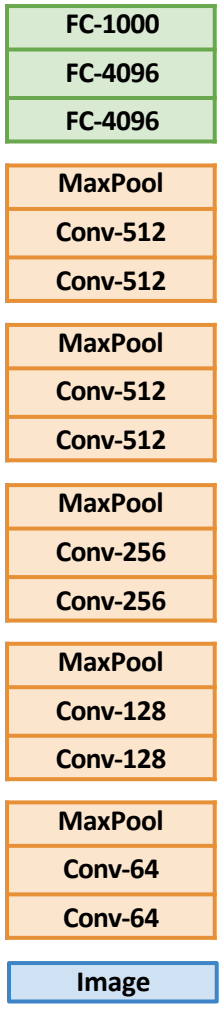




Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

Transfer Learning with CNNs

1. Train on Imagenet



2. Use CNN as a feature extractor

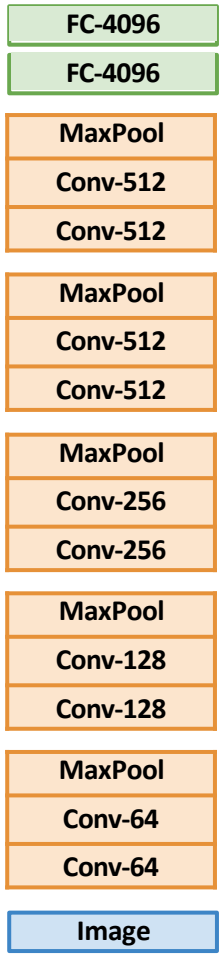
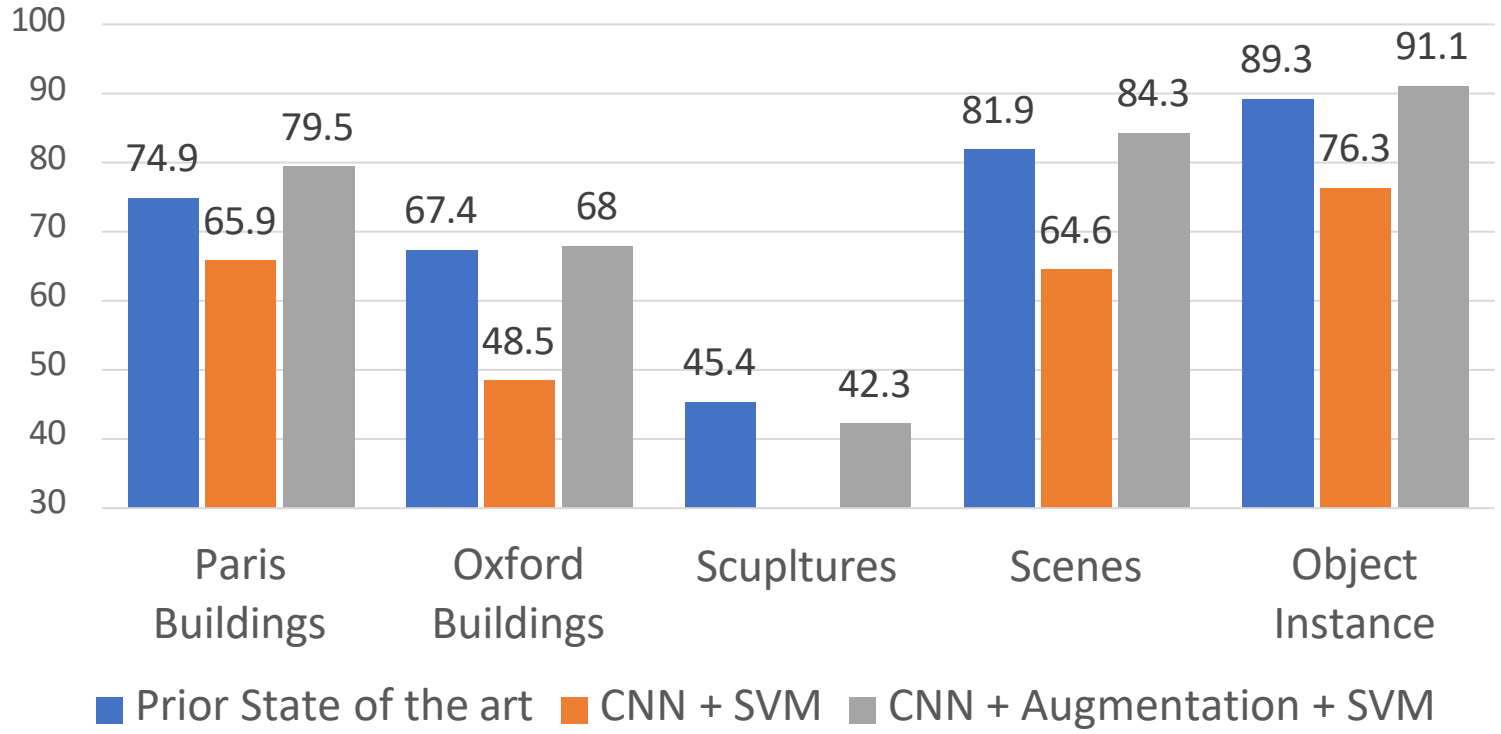


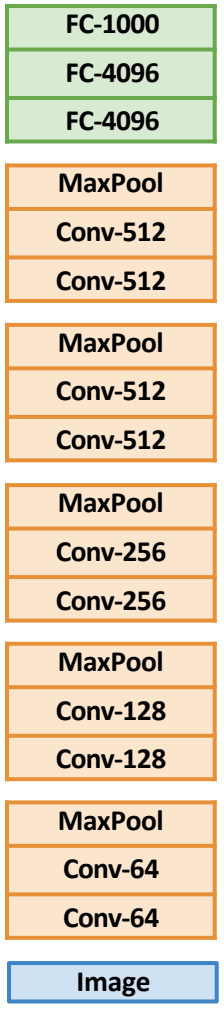
Image Retrieval: Nearest-Neighbor



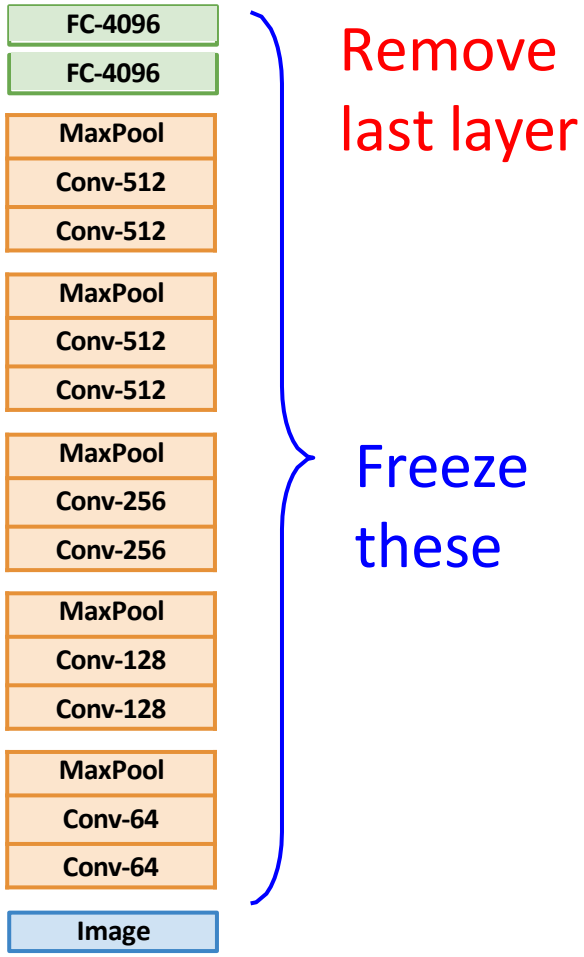
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

Transfer Learning with CNNs

1. Train on Imagenet



2. Use CNN as a feature extractor

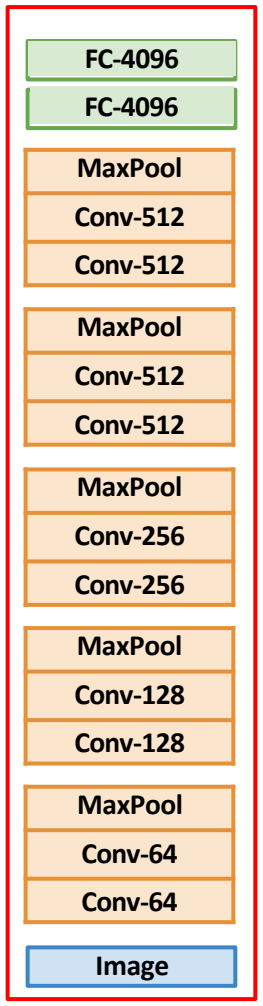


Remove last layer

Freeze these

3. Bigger dataset:

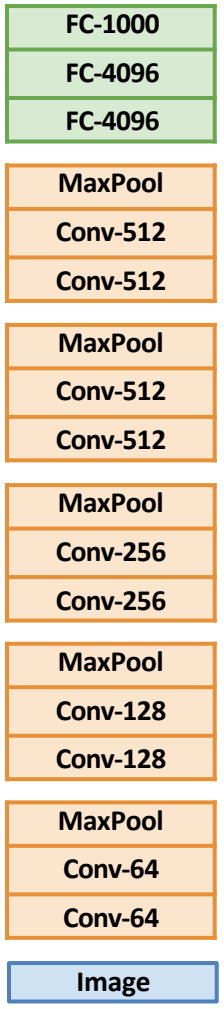
Fine-Tuning



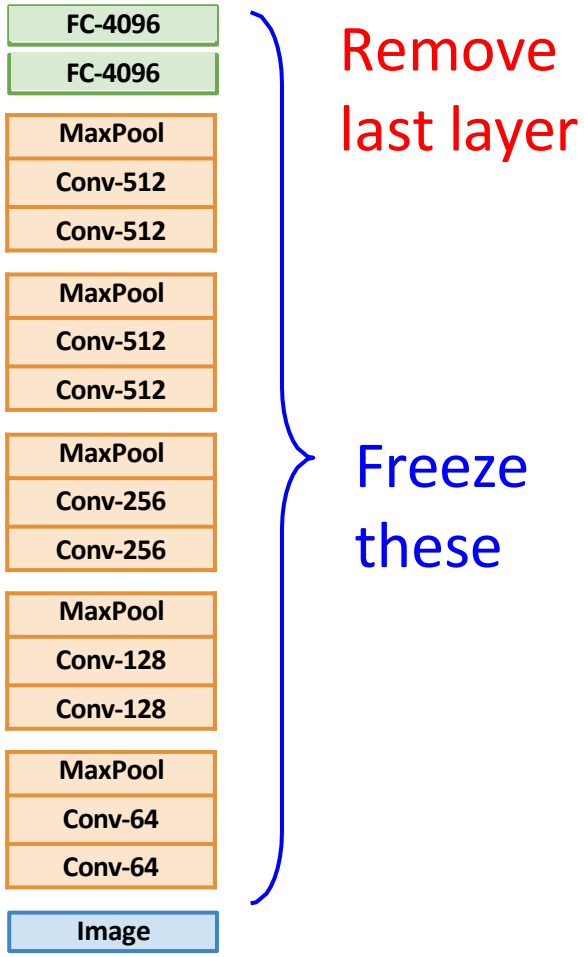
Continue training CNN for new task

Transfer Learning with CNNs

1. Train on Imagenet



2. Use CNN as a feature extractor

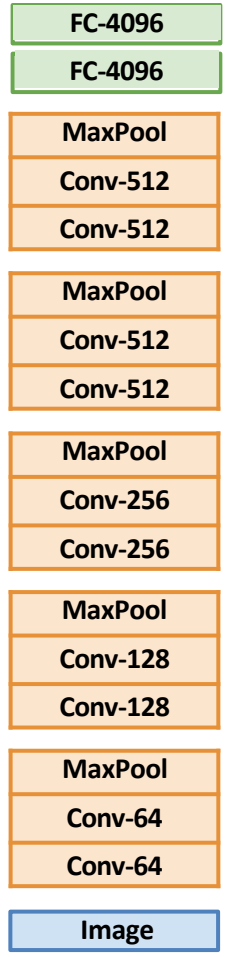


Remove last layer

Freeze these

3. Bigger dataset:

Fine-Tuning

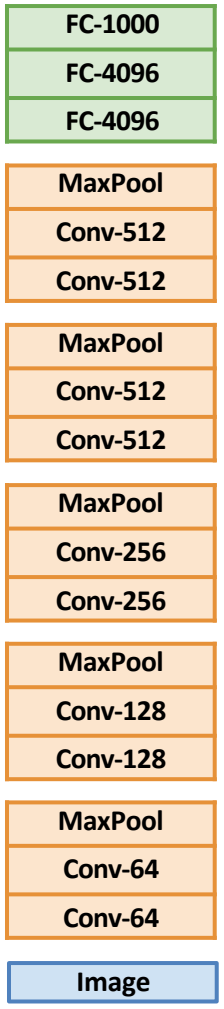


- Continue training CNN for new task

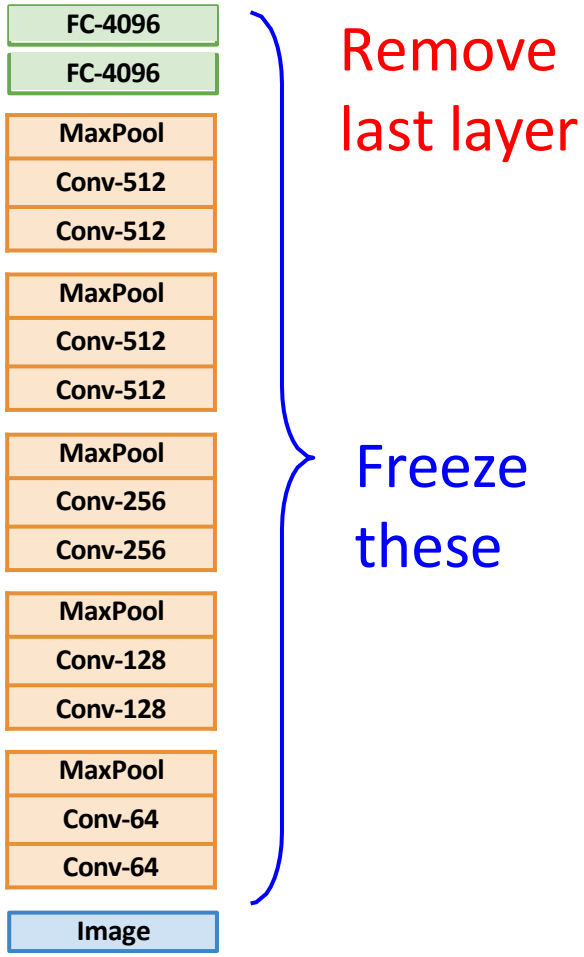
- Some tricks:
  - Lower the learning rate: use ~1/10 of LR used in original training
  - Sometimes freeze lower layers to save computation

Transfer Learning with CNNs

1. Train on Imagenet



2. Use CNN as a feature extractor

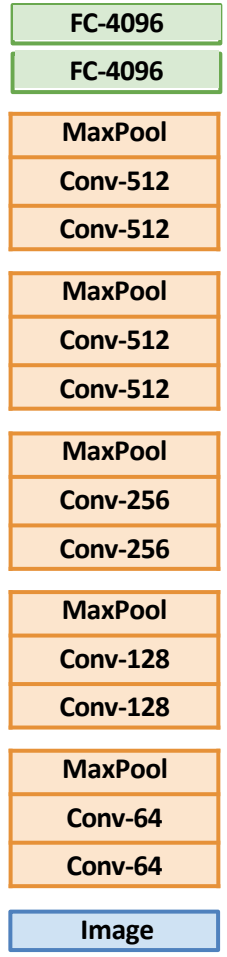


Remove last layer

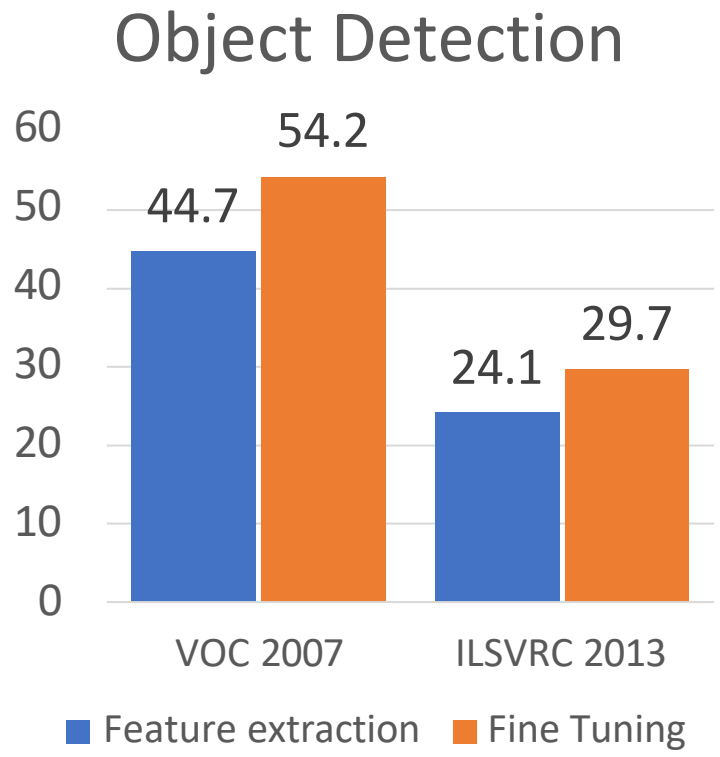
Freeze these

3. Bigger dataset:

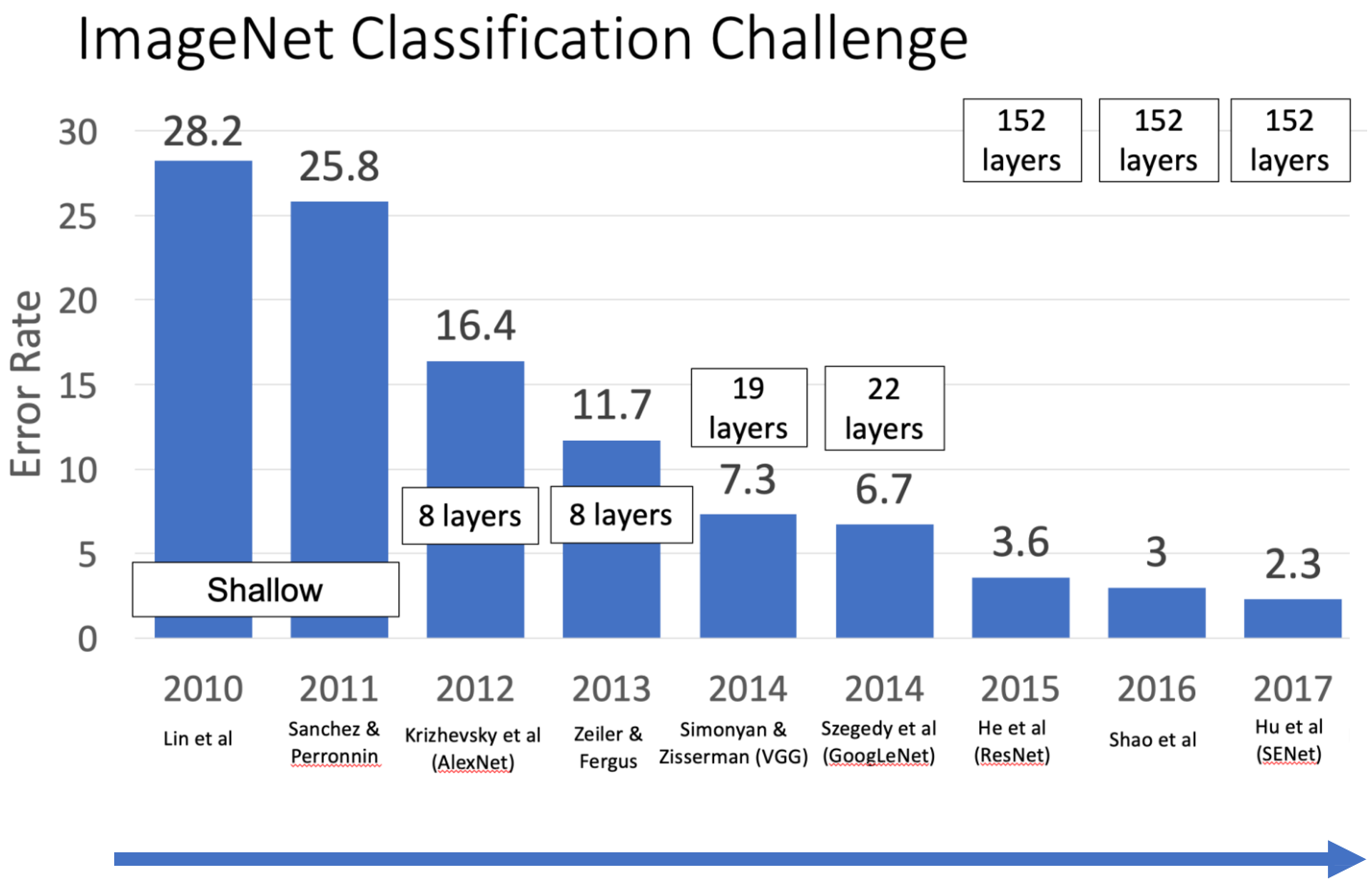
Fine-Tuning



Continue training CNN for new task

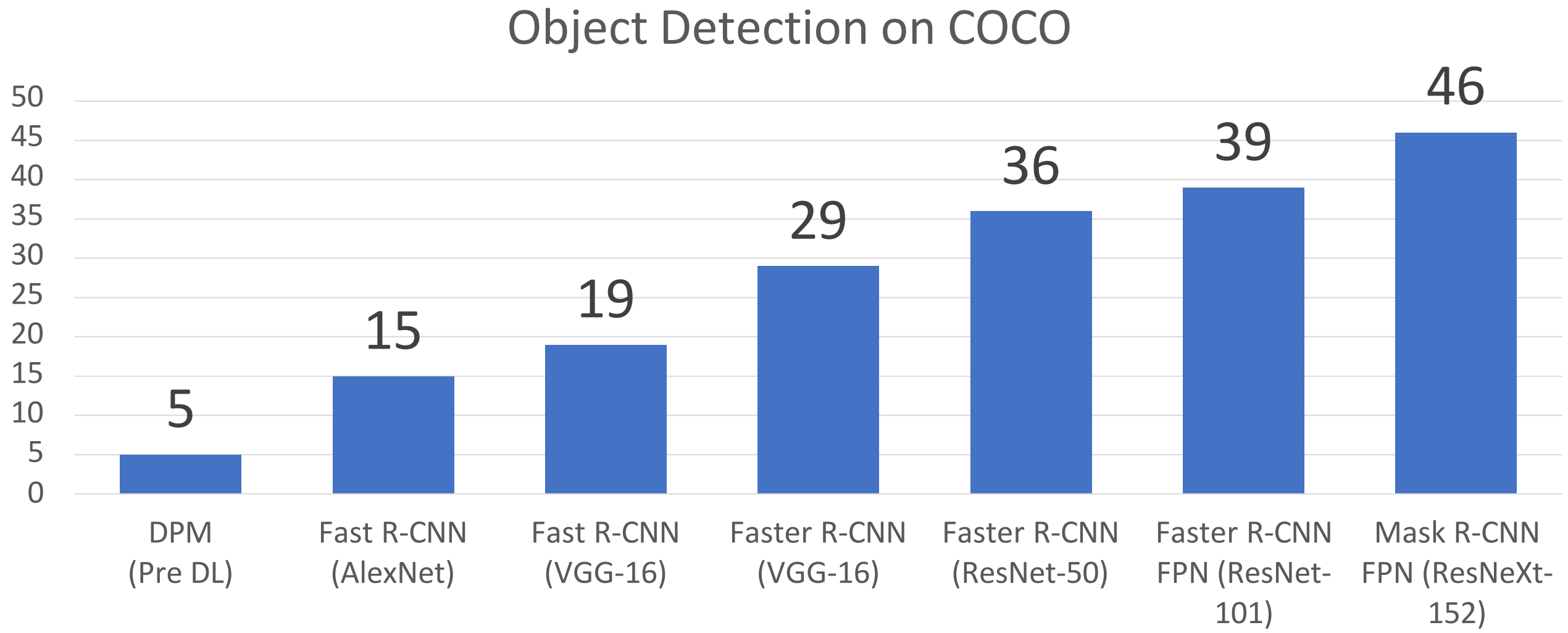


Transfer Learning with CNNs: Architecture Matters



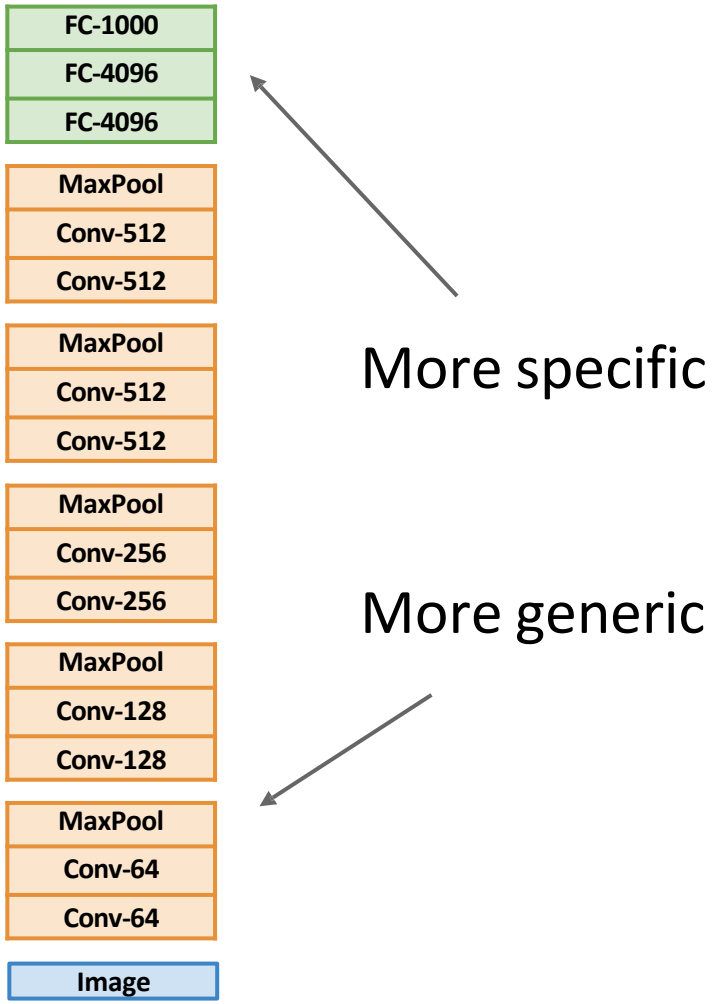
Improvements in CNN architectures lead to improvements in many downstream tasks due to transfer learning

Transfer Learning with CNNs: Architecture Matters



Ross Girshick, "The Generalized R-CNN Framework for Object Detection", ICCV 2017 Tutorial on Instance-Level Visual Recognition

Transfer Learning with CNNs

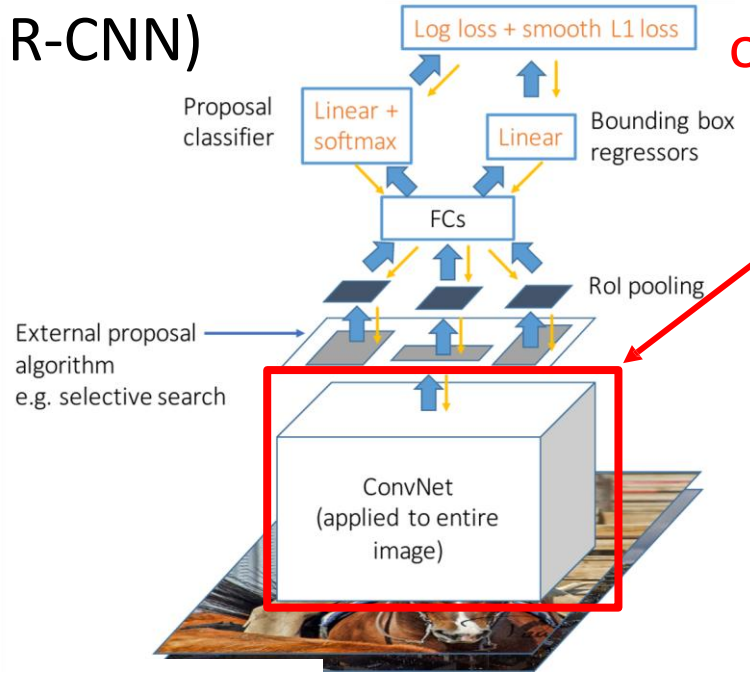


	Dataset similar to ImageNet	**Dataset very different from ImageNet**
very little data (10s to 100s)	Use Linear Classifier on top layer	Might not work well
quite a lot of data (100s to 1000s)	Finetune a few layers	Finetune a larger number of layers

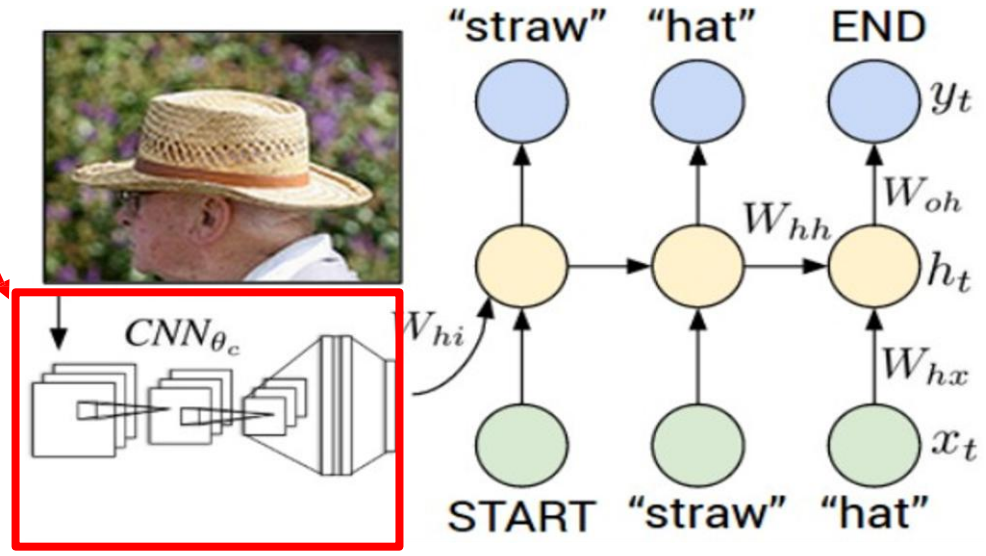
Transfer learning is pervasive

It's the norm, not the exception

Object Detection (Fast R-CNN)



CNN pretrained on ImageNet

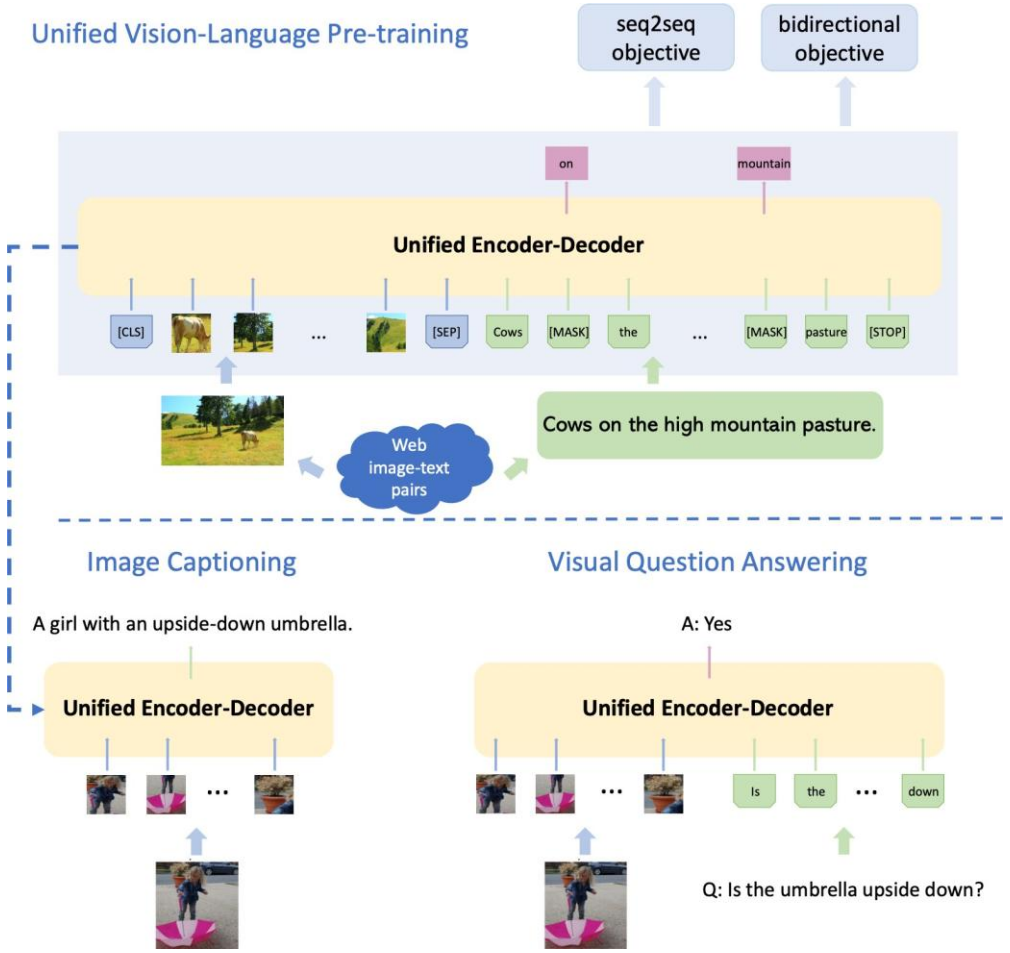


Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015

Transfer learning is pervasive

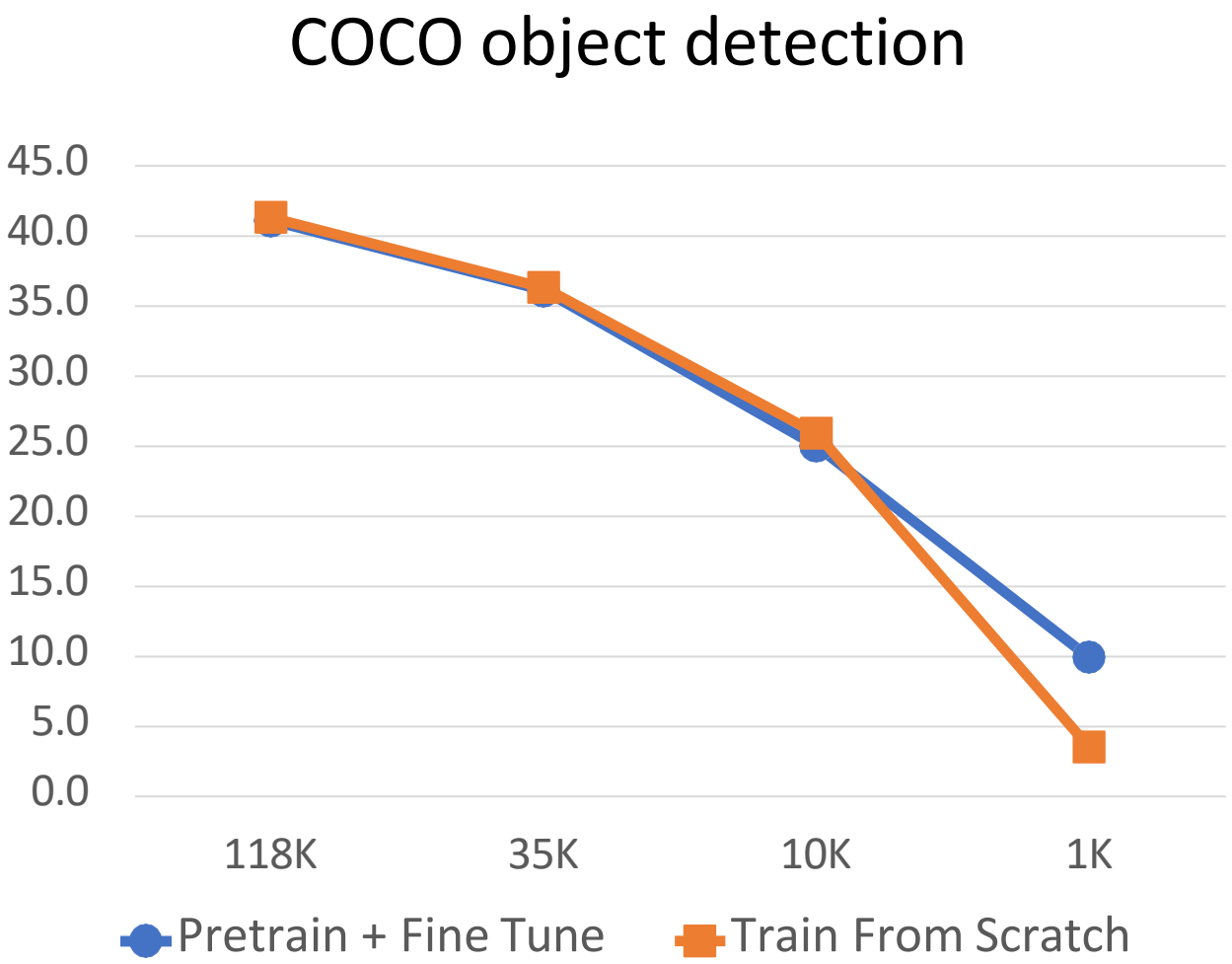
It's the norm, not the exception



1. Train CNN on ImageNet
2. Fine-Tune (1) for object detection on Visual Genome
3. Train BERT language model on lots of text
4. Combine (2) and (3), train for joint image / language modeling
5. Fine-tune (5) for image captioning, visual question answering, etc.

Zhou et al, "Unified Vision-Language Pre-Training for Image Captioning and VQA", arXiv 2019

Transfer learning is pervasive



Pretraining + Finetuning beats training from scratch when dataset size is very small

Collecting more data is more effective than pretraining

He et al, "Rethinking ImageNet Pre-Training", ICCV 2019