Lecture 6:
Training CNNs and CNN Architectures

Course Logistics

- Assignment 1 is due **next Wednesday** (4/23) at 11:59PM!
- Project proposal deadline is due on **Friday next week** (4/25)

Lecture Overview – Two Broad Sets of Topics

How to build CNNs?

- **Layers in CNNs**
- **Activation Functions**
- **CNN Architectures**
- **Weight Initialization**

How to train CNNs?

- **Data Preprocessing**
- **Data augmentation**
- **Transfer Learning**
- **Hyperparameter Selection**

Lecture Overview – Two Broad Sets of Topics

How to build CNNs?

- **Layers in CNNs**
- **Activation Functions**
- **CNN Architectures**
- **Weight Initialization**

How to train CNNs?

- **Data Preprocessing**
- **Data augmentation**
- **Transfer Learning**
- **Hyperparameter Selection**

Lecture Overview – Two Broad Sets of Topics

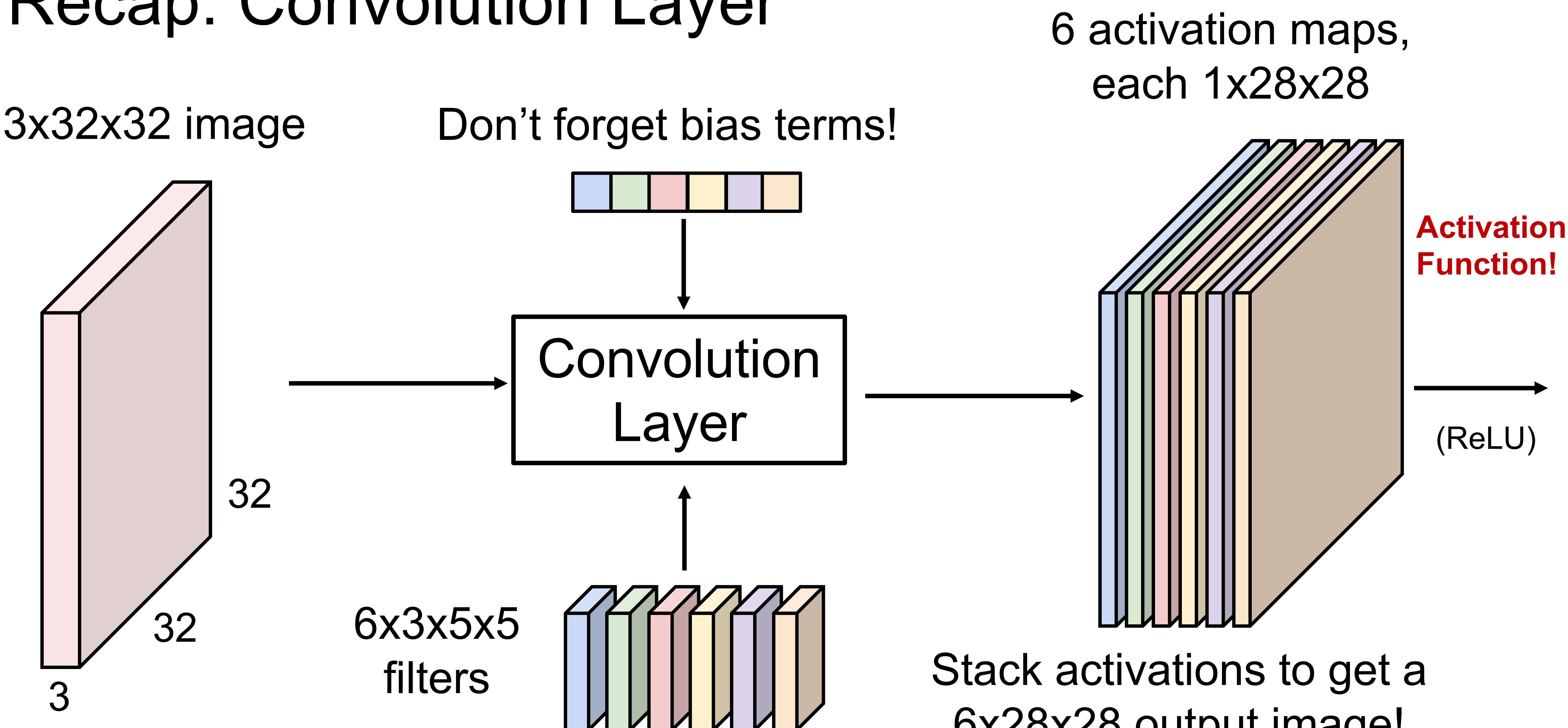
How to build CNNs?

- **Layers in CNNs**
- **Activation Functions**
- **CNN Architectures**
- **Weight Initialization**

How to train CNNs?

- **Data Preprocessing**
- **Data augmentation**
- **Transfer Learning**
- **Hyperparameter Selection**

Recap: Convolution Layer



Slide inspiration: Justin Johnson

Recap: Pooling Layer

Single depth slice

x

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

y

pool with 2x2 filters and stride 2

6	8
3	4

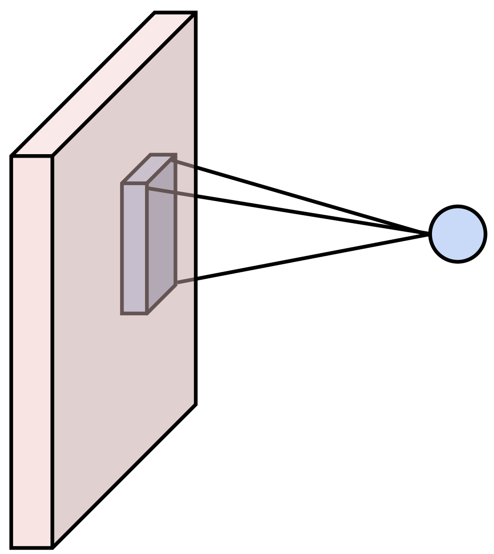
Max Pooling

3.25	5.25
2	2

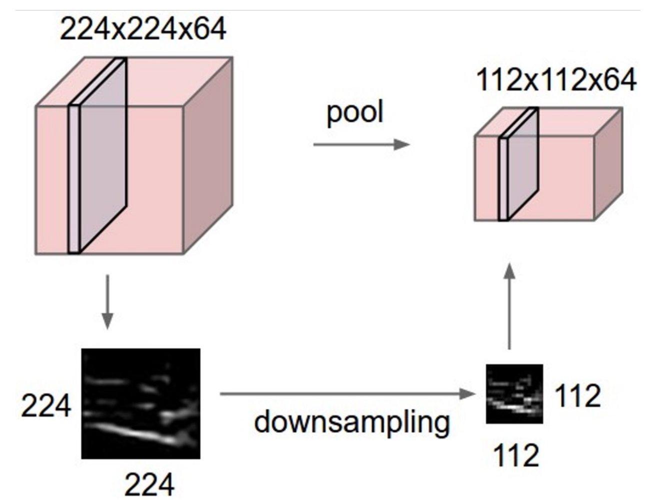
Average Pooling

Components of CNNs

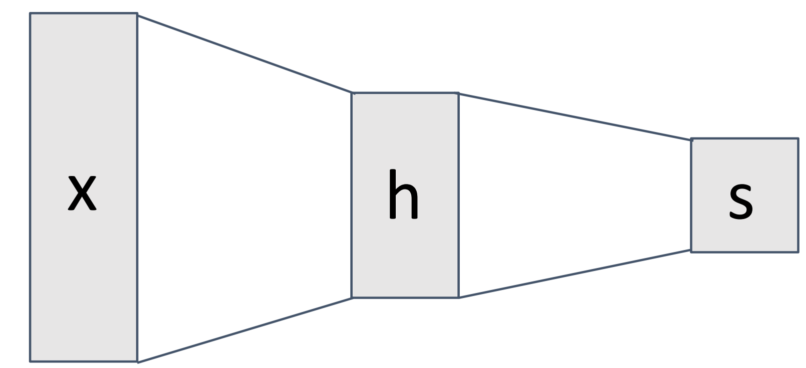
Convolution Layers



Pooling Layers



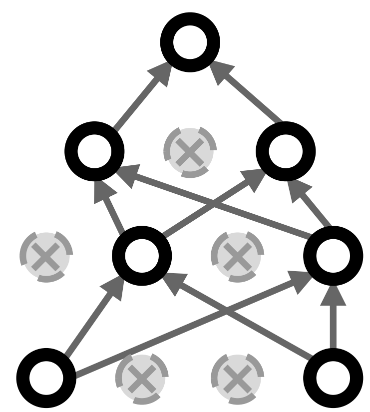
Fully-Connected Layers



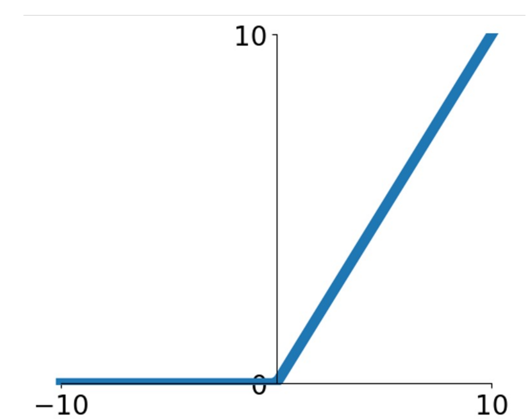
Normalization Layers

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

Dropout (sometimes)

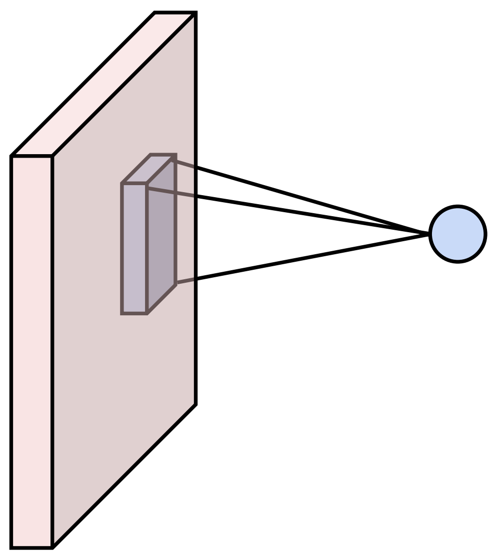


Activation Functions

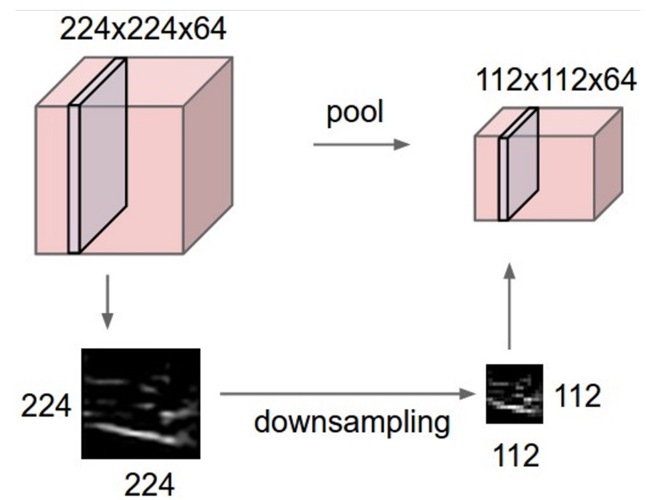


Components of CNNs

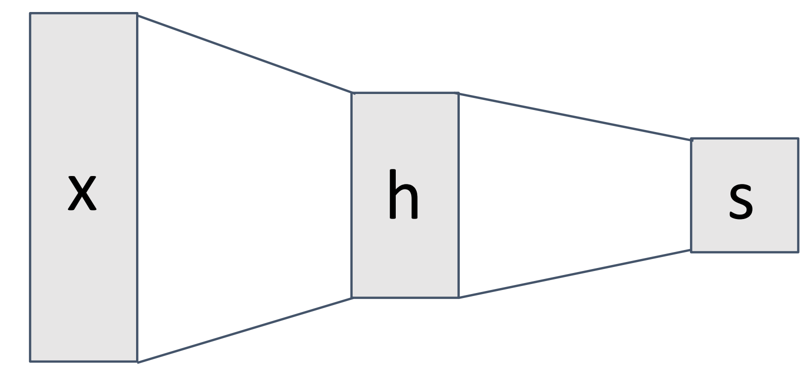
Convolution Layers



Pooling Layers



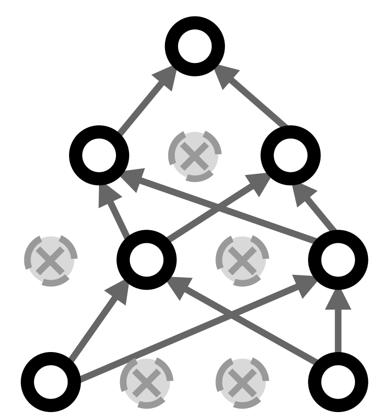
Fully-Connected Layers



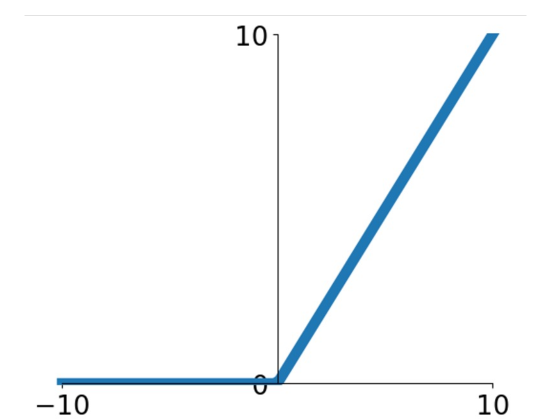
Normalization Layers

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

Dropout (sometimes)



Activation Functions



Example Normalization Layer: LayerNorm

High-level Idea: Learn parameters that let us scale / shift the input data

1. Normalize input data
2. Scale / shift using learned parameters

Ba, Kiros, and Hinton, "Layer Normalization", arXiv 2016

Example Normalization Layer: LayerNorm

High-level Idea: Learn parameters that let us scale / shift the input data

1. Normalize input data
2. Scale / shift using learned parameters

$$\mathbf{x}: N \times D$$

Normalize

Statistics calculated per batch →

$$\boldsymbol{\mu}, \boldsymbol{\sigma}: N \times 1$$

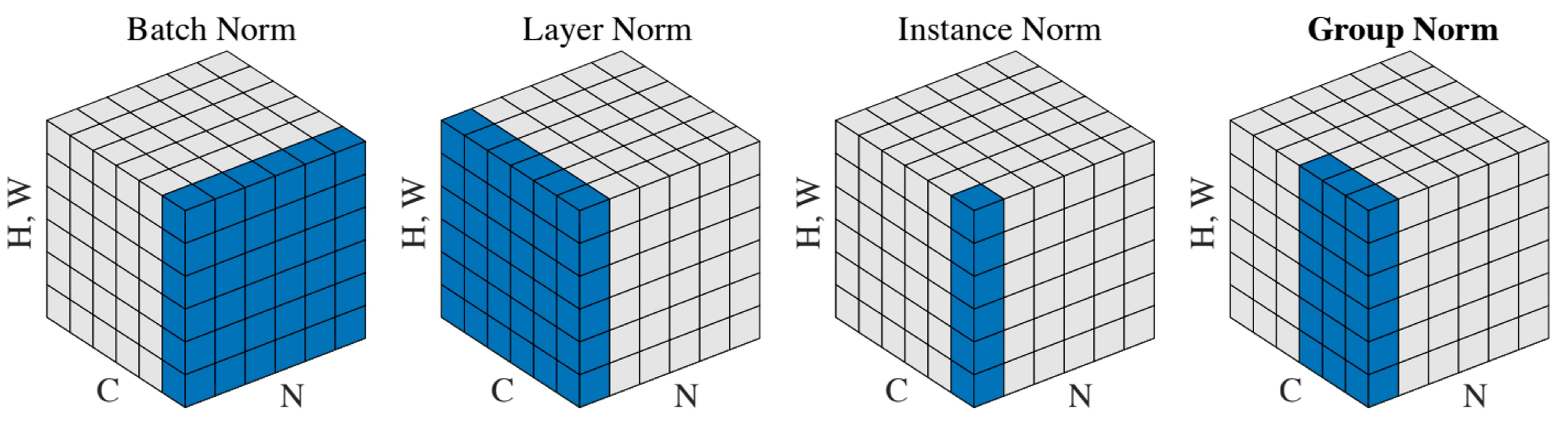
Learned parameters applied to each sample →

$$\boldsymbol{\gamma}, \boldsymbol{\beta}: 1 \times D$$

$$\mathbf{y} = \boldsymbol{\gamma}(\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \boldsymbol{\beta}$$

Ba, Kiros, and Hinton, “Layer Normalization”, arXiv 2016

Other Normalization Layers

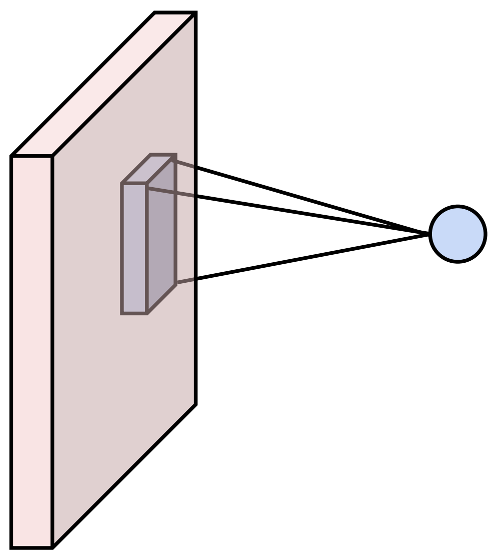


You will implement some of these in assignment 2!

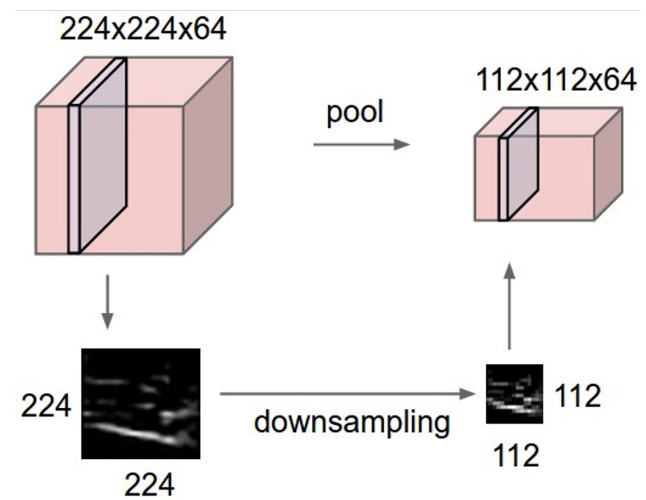
Wu and He, "Group Normalization", ECCV 2018

Components of CNNs

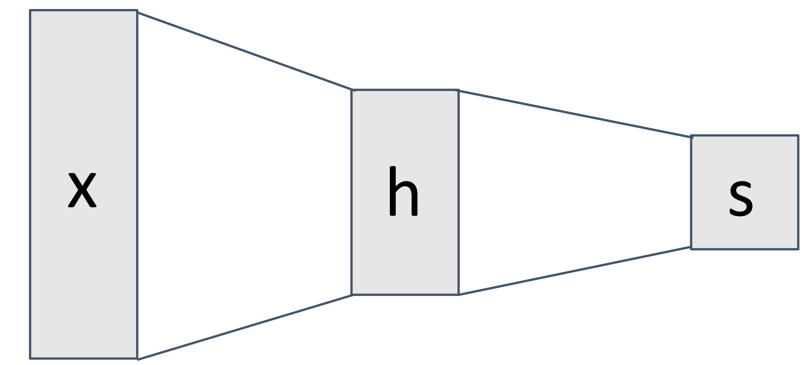
Convolution Layers



Pooling Layers



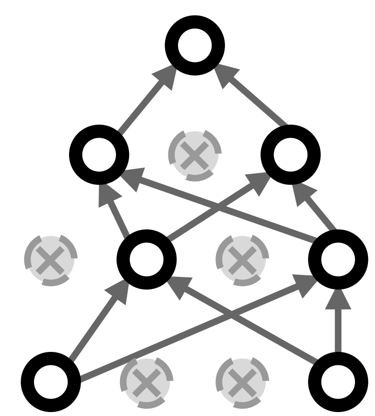
Fully-Connected Layers



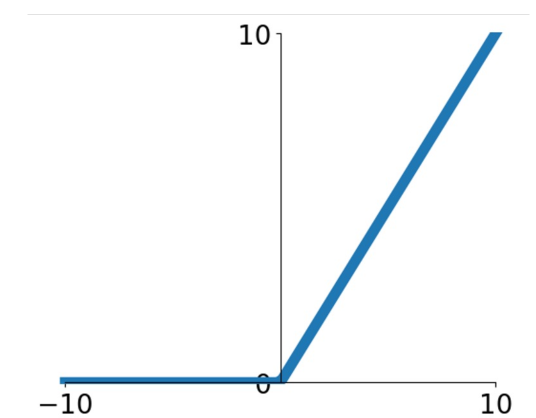
Normalization Layers

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

Dropout (sometimes)

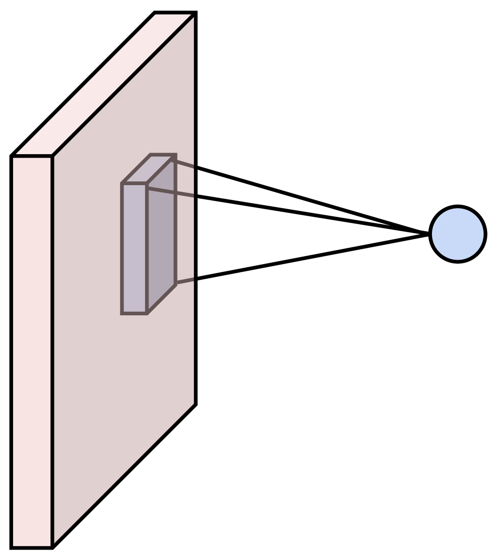


Activation Functions

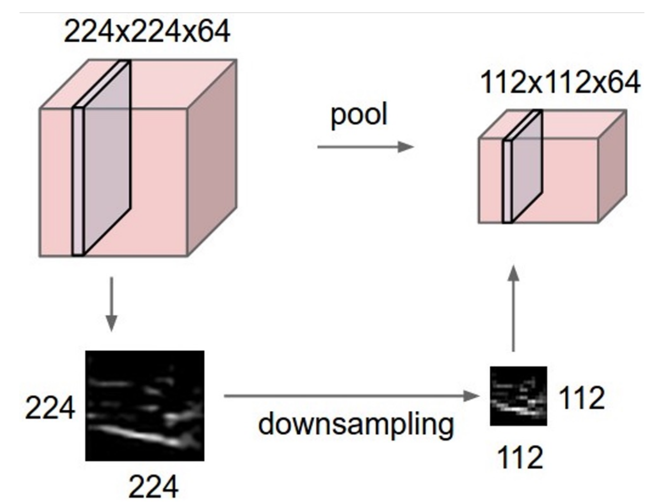


Components of CNNs

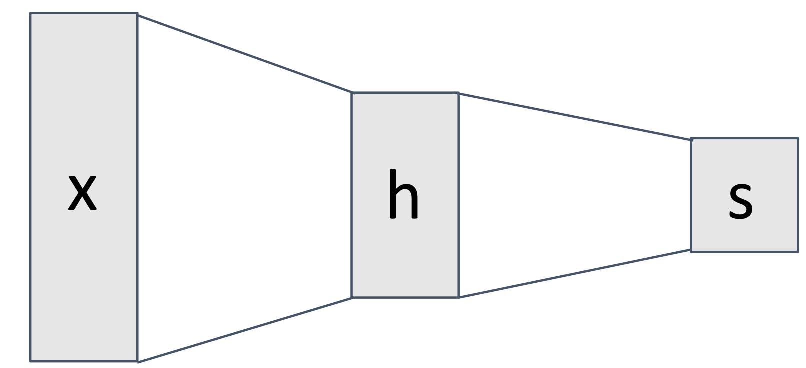
Convolution Layers



Pooling Layers



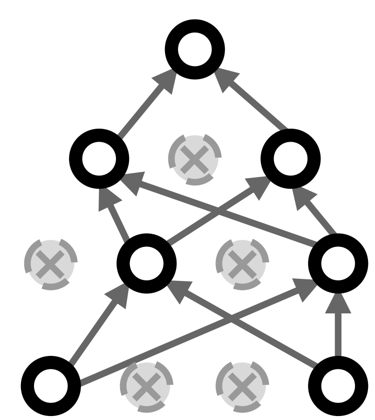
Fully-Connected Layers



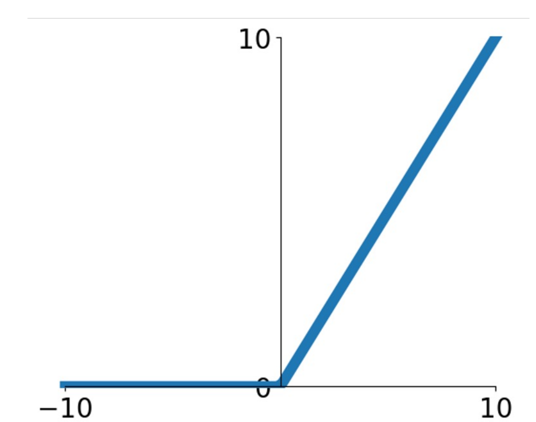
Normalization Layers

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

Dropout (sometimes)

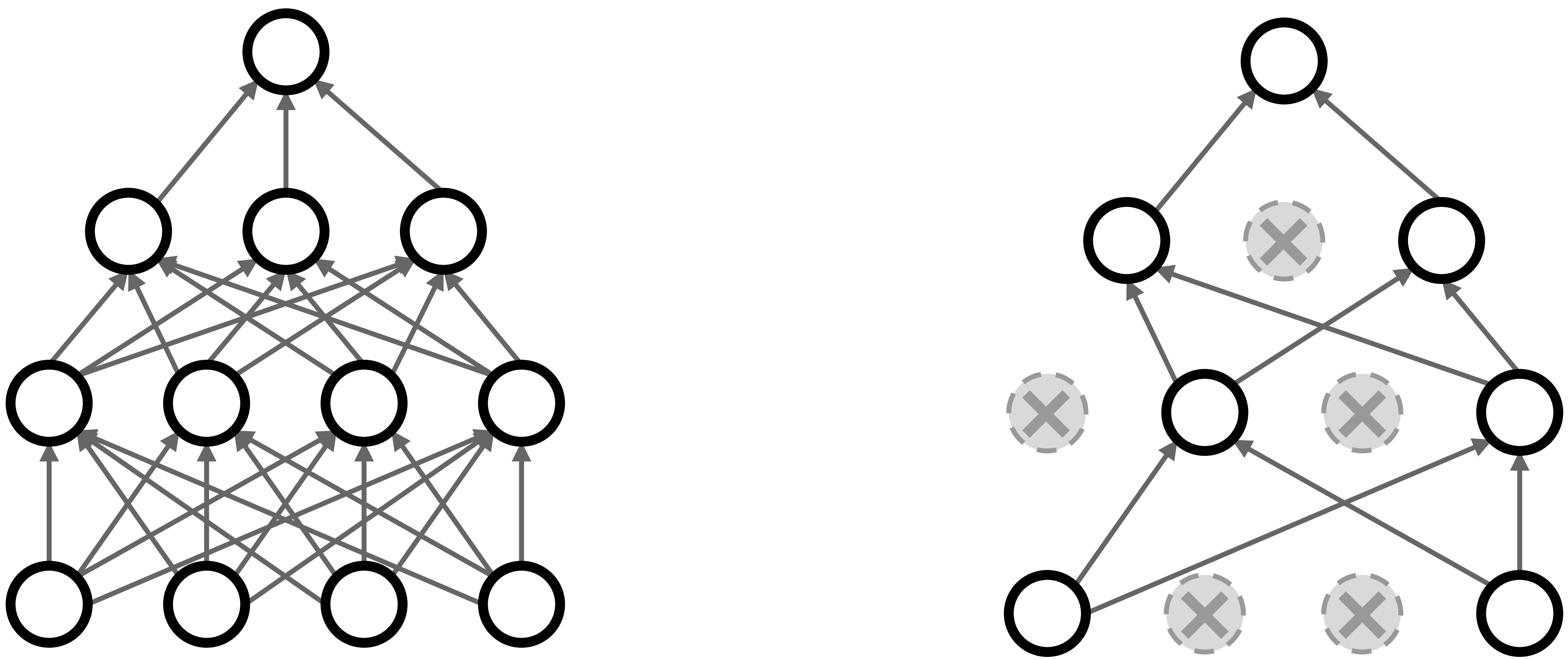


Activation Functions



Regularization: Dropout

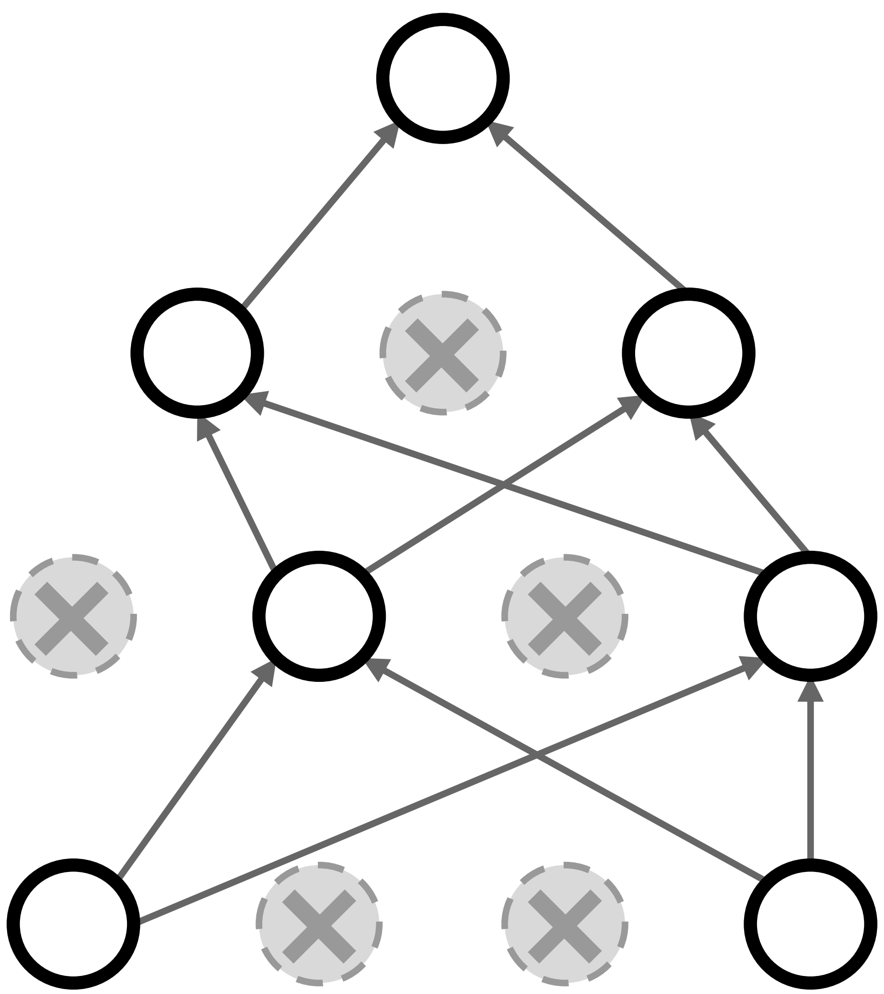
In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common



Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

Regularization: Dropout

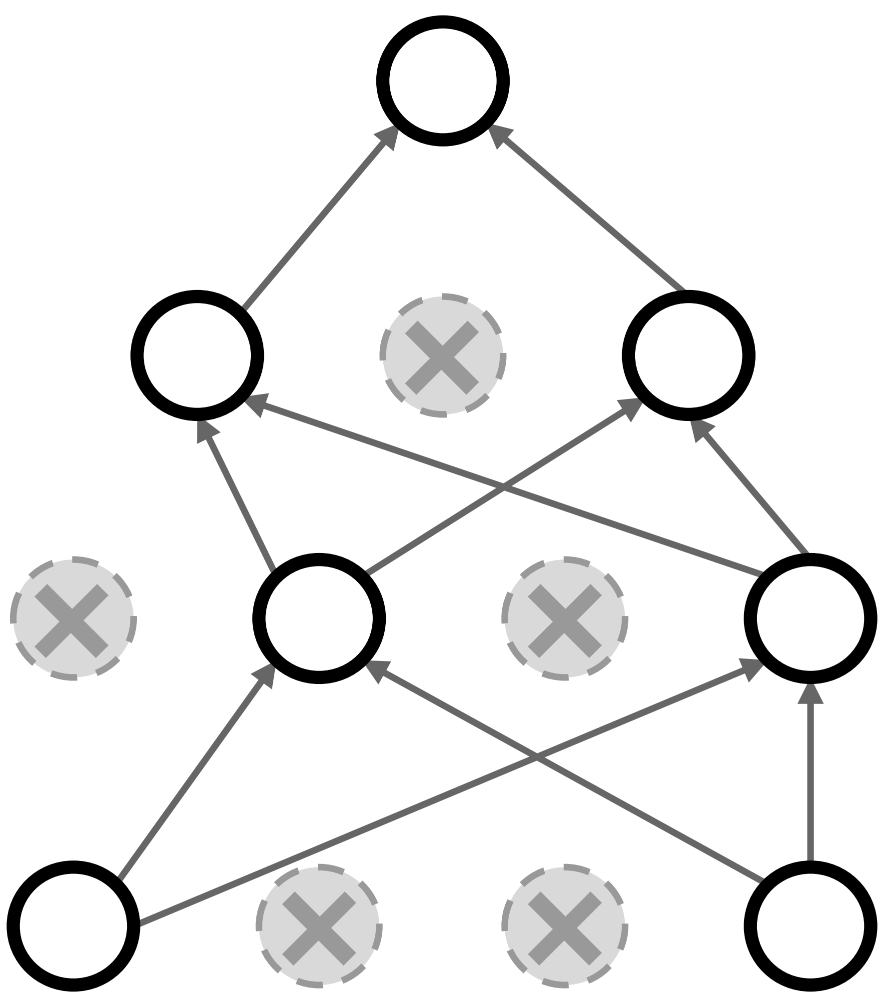
How can this possibly be a good idea?



Forces the network to have a redundant representation;
Prevents co-adaptation of features

Regularization: Dropout

How can this possibly be a good idea?



Another interpretation:

Dropout is training a large **ensemble** of models (that share parameters).

Each binary mask is one model

An FC layer with 4096 units has $2^{4096} \sim 10^{1233}$ possible masks!
Only $\sim 10^{82}$ atoms in the universe...

Dropout: Test time

```python
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

At test time all neurons are active always
=> We must scale the activations so that for each neuron:
<u>output at test time</u> = <u>expected output at training time</u>

Dropout Summary

```
""" Vanilla Dropout: Not recommended implementation (see notes below) """

p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

drop in train time

scale at test time

Lecture Overview – Two Broad Sets of Topics

How to build CNNs?

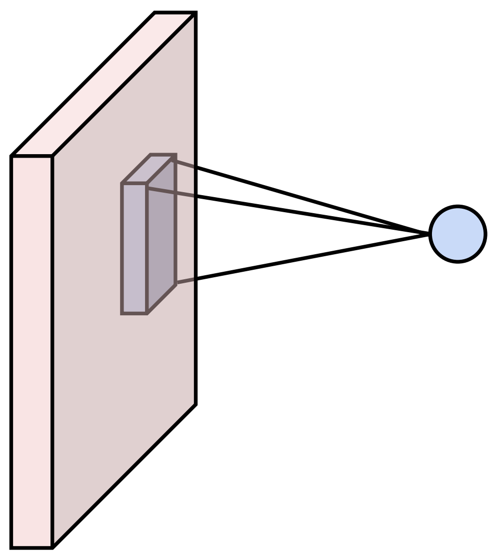
- **Layers in CNNs**
- **Activation Functions**
- **CNN Architectures**
- **Weight Initialization**

How to train CNNs?

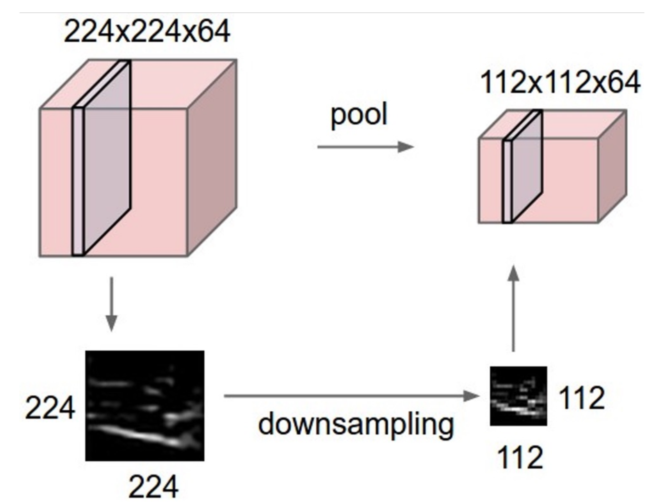
- **Data Preprocessing**
- **Data augmentation**
- **Transfer Learning**
- **Hyperparameter Selection**

Components of CNNs

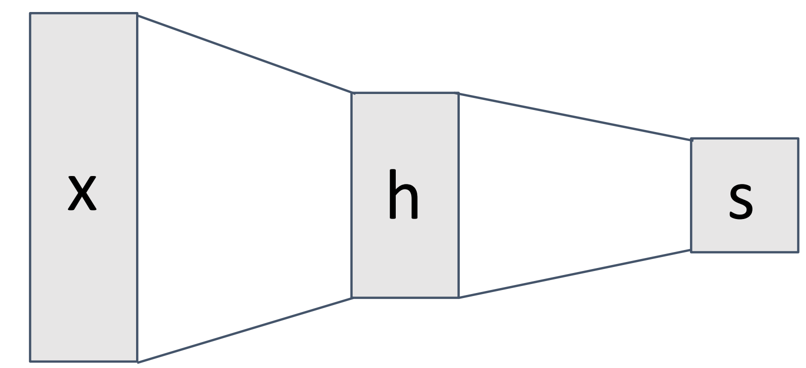
Convolution Layers



Pooling Layers



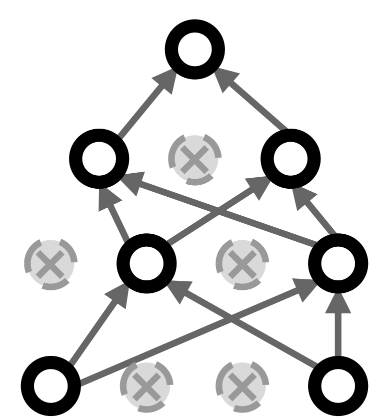
Fully-Connected Layers



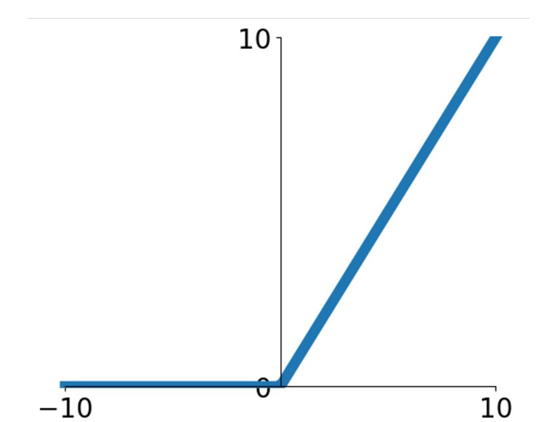
Normalization Layers

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

Dropout (sometimes)

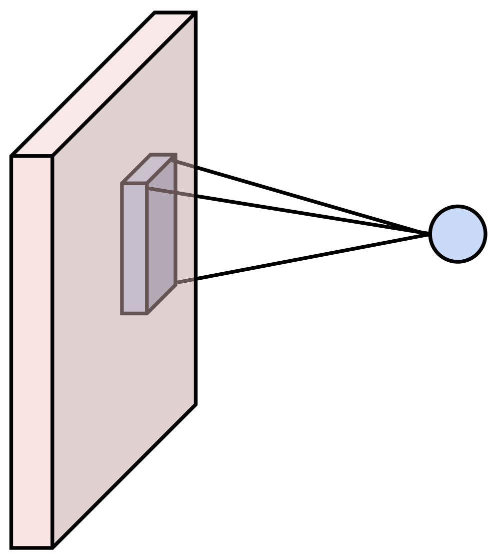


Activation Functions

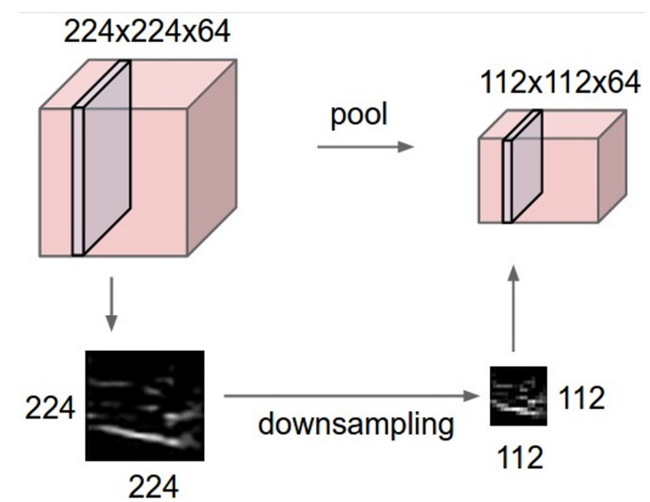


Components of CNNs

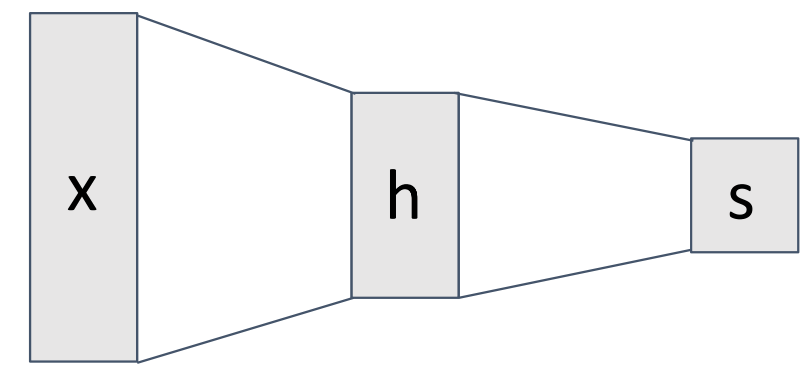
Convolution Layers



Pooling Layers



Fully-Connected Layers

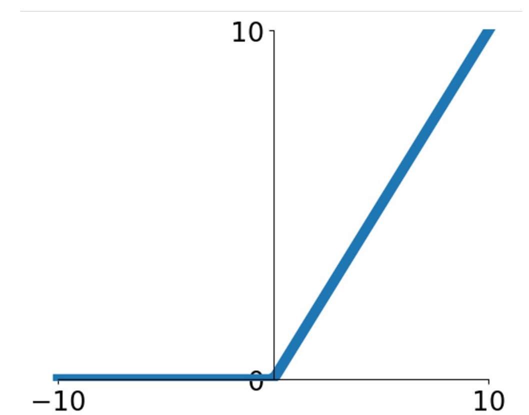


Normalization Layers

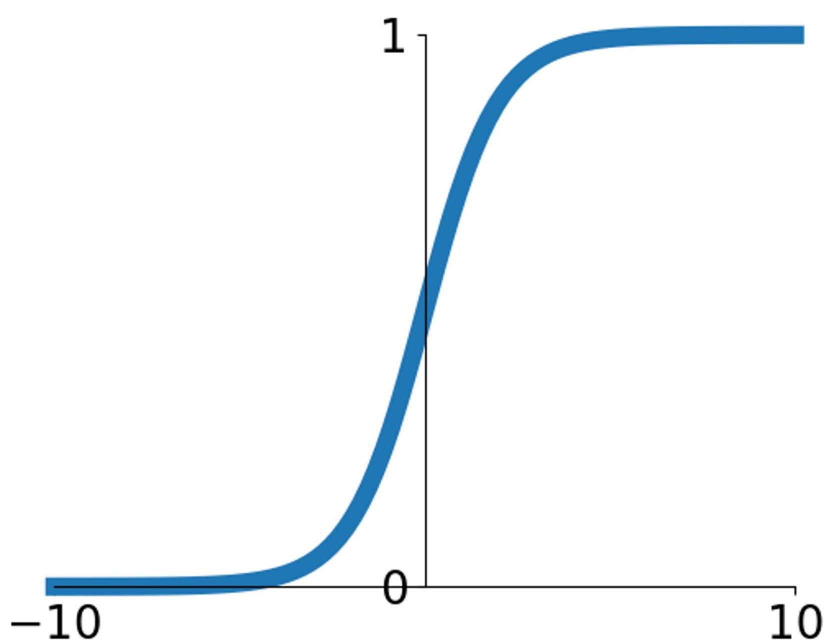
Dropout (sometimes)

Goal: Introduce non-linearities to our model!

Activation Functions



Activation Functions



Sigmoid

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

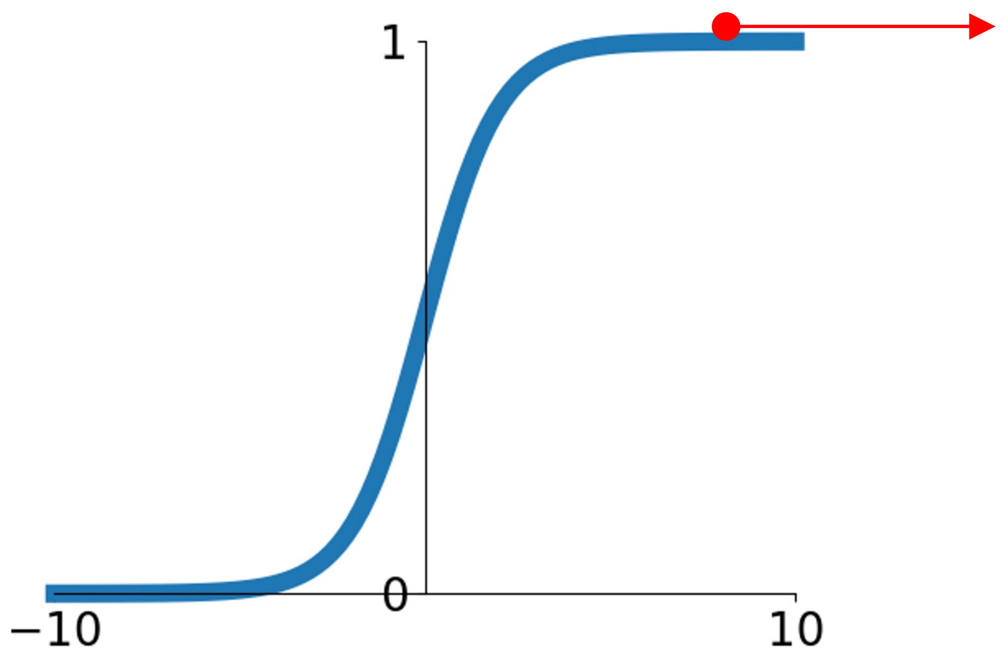
- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

Key problem:

Many layers of sigmoids → smaller and smaller gradients.

Q: In which regions does sigmoid have a small gradient?

Activation Functions



Sigmoid

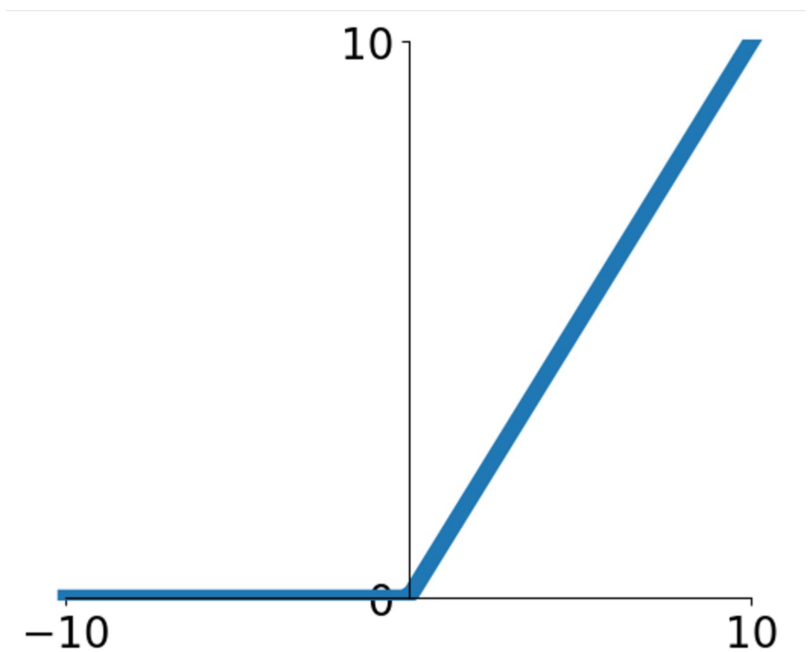
$$\sigma(x) = 1/(1+e^{-x})$$

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

Key problem:

Large positive or negative values can "kill" the gradients. Many layers of sigmoids → smaller and smaller gradients in practice

Activation Functions



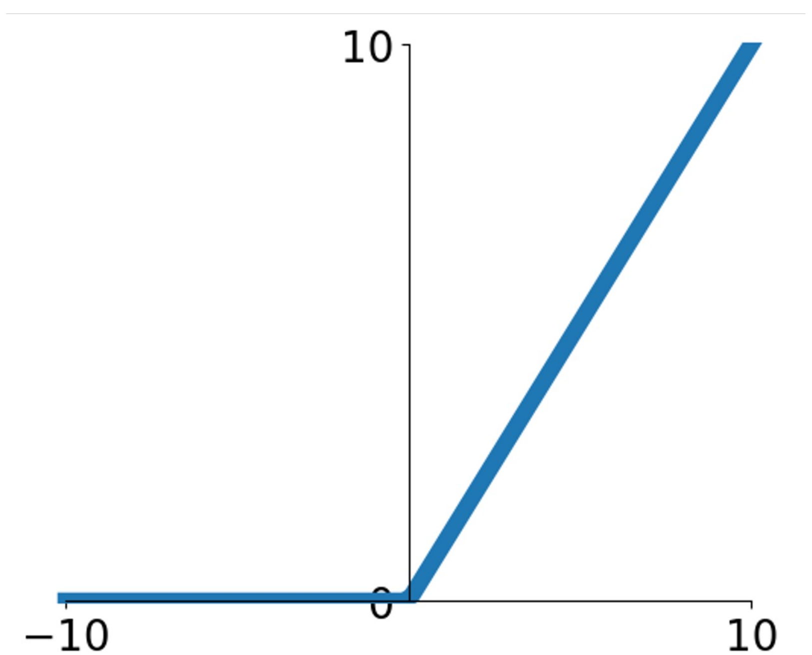
ReLU
(Rectified Linear Unit)

- Computes **f(x) = max(0,x)**

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid in practice (e.g. 6x)

[Krizhevsky et al., 2012]

Activation Functions



ReLU
(Rectified Linear Unit)

- Computes **f(x) = max(0,x)**

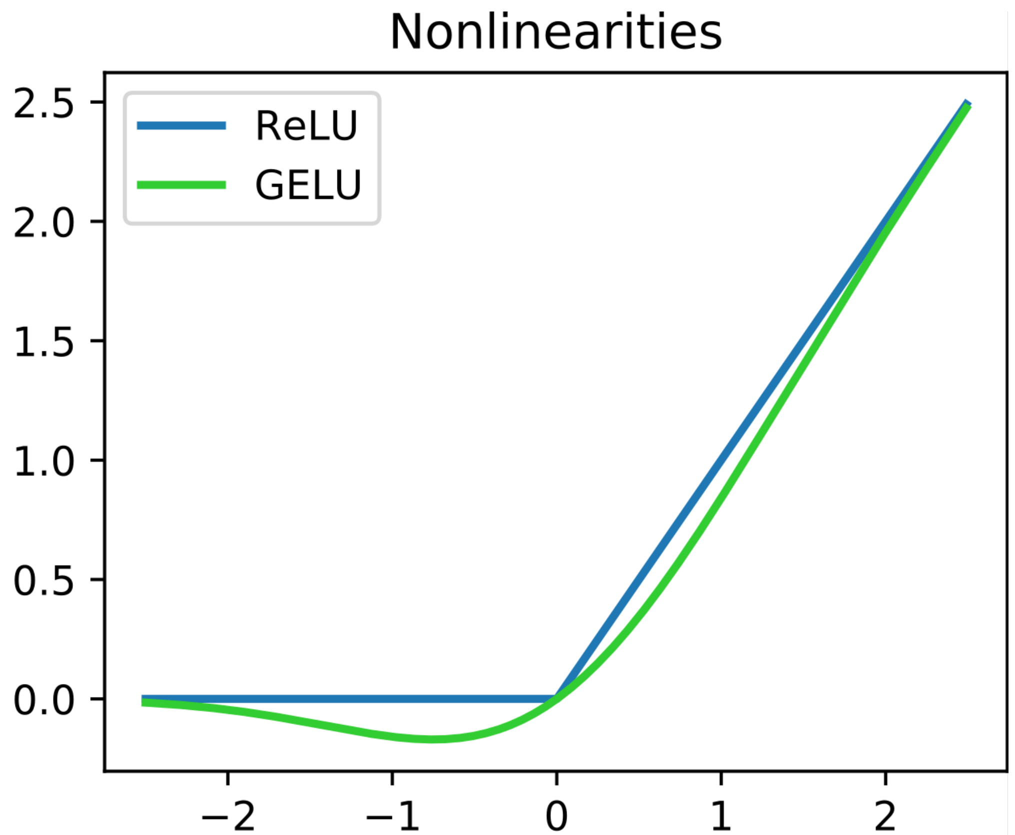
- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid in practice (e.g. 6x)

- Not zero-centered output
- An annoyance:

Dead ReLUs when x < 0!

Activation Functions

[Hendrycks et al., 2016]



Source: https://en.m.wikipedia.org/wiki/File:ReLU_and_GELU.svg

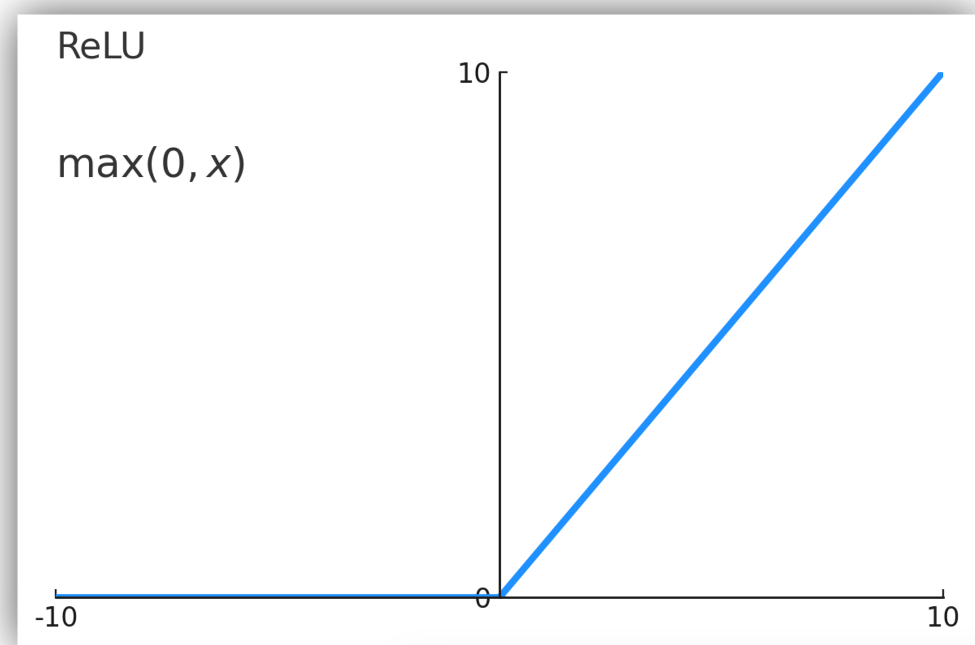
GELU
(Gaussian Error Linear Unit)

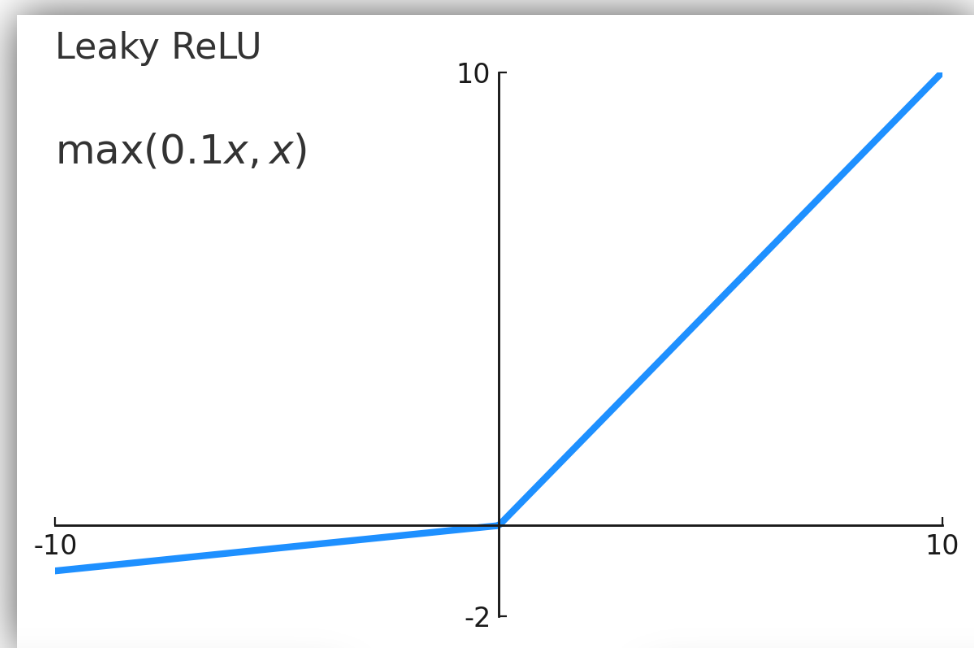
- Computes **f(x) = x*Φ(x)**

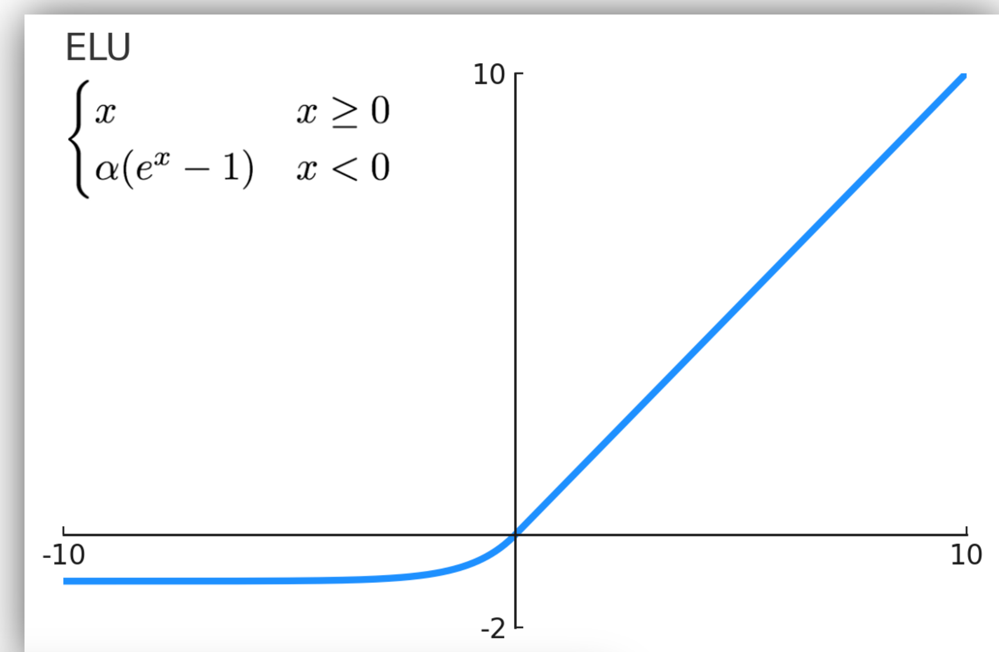
- Very nice behavior around 0
- Smoothness facilitates training in practice

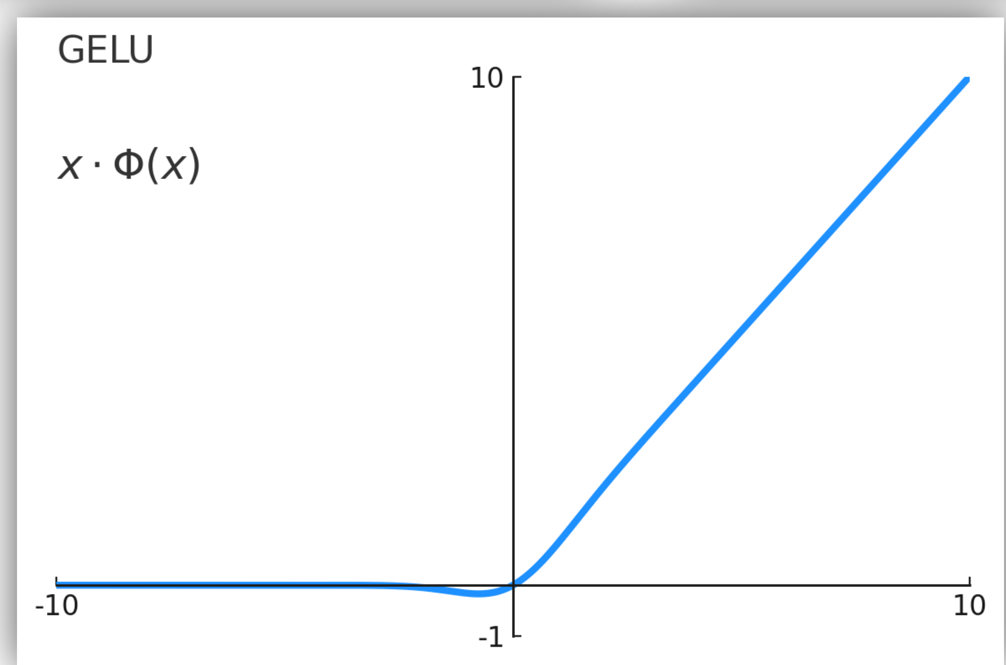
- Higher computational cost than ReLU
- Large negative values can still have gradient → 0

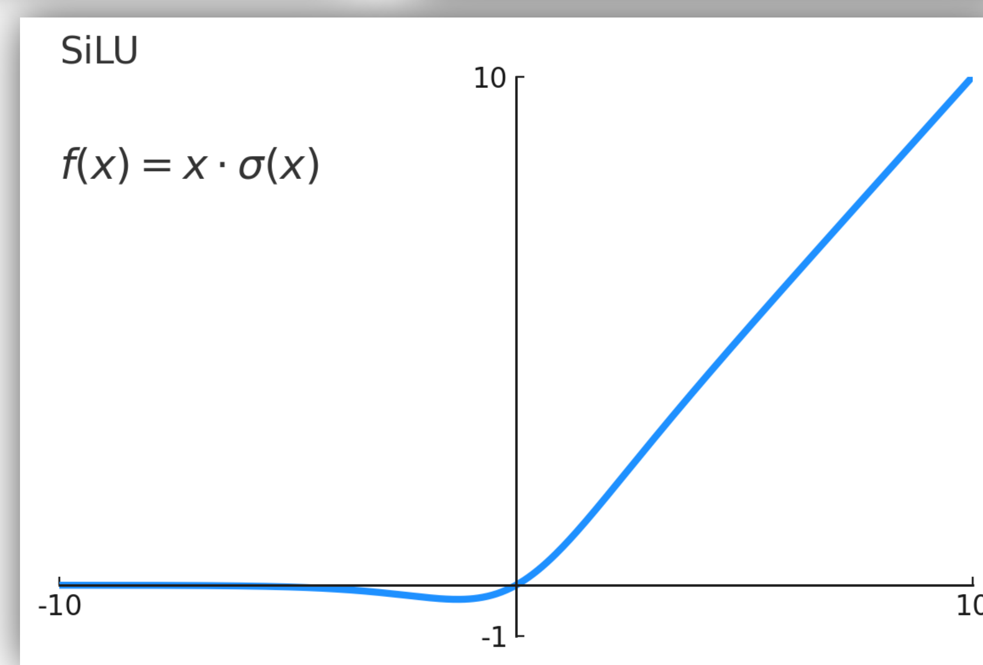
Activation Function Zoo





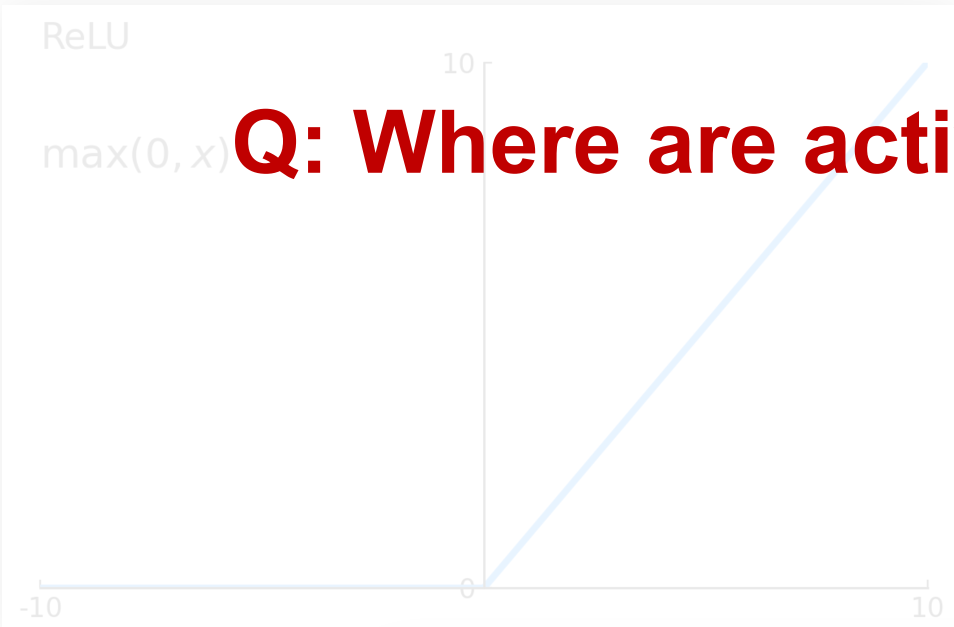


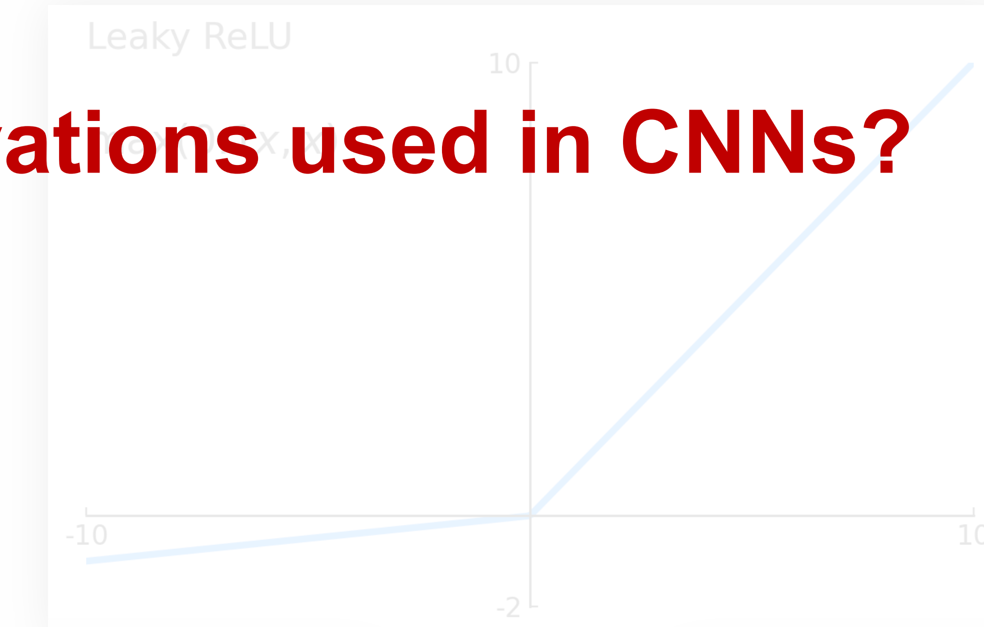


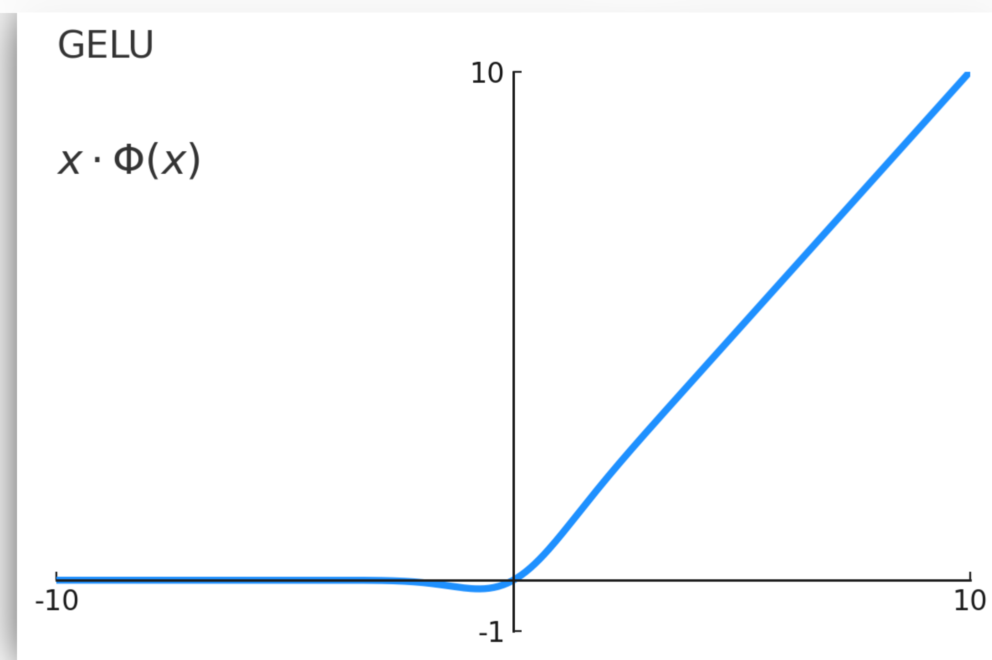


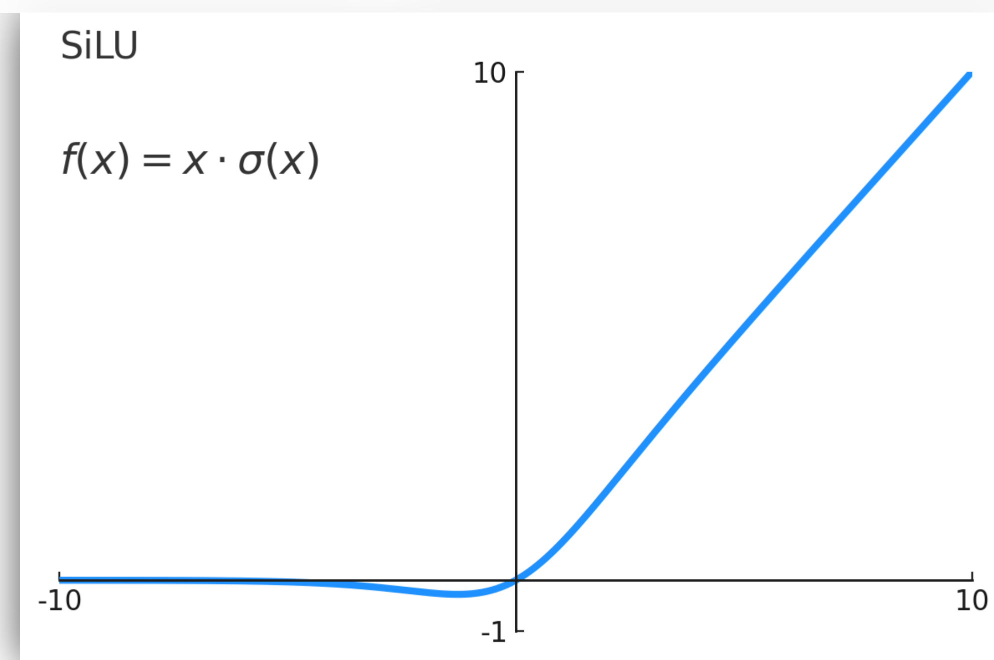
Activation Function Zoo

Q: Where are activations used in CNNs?





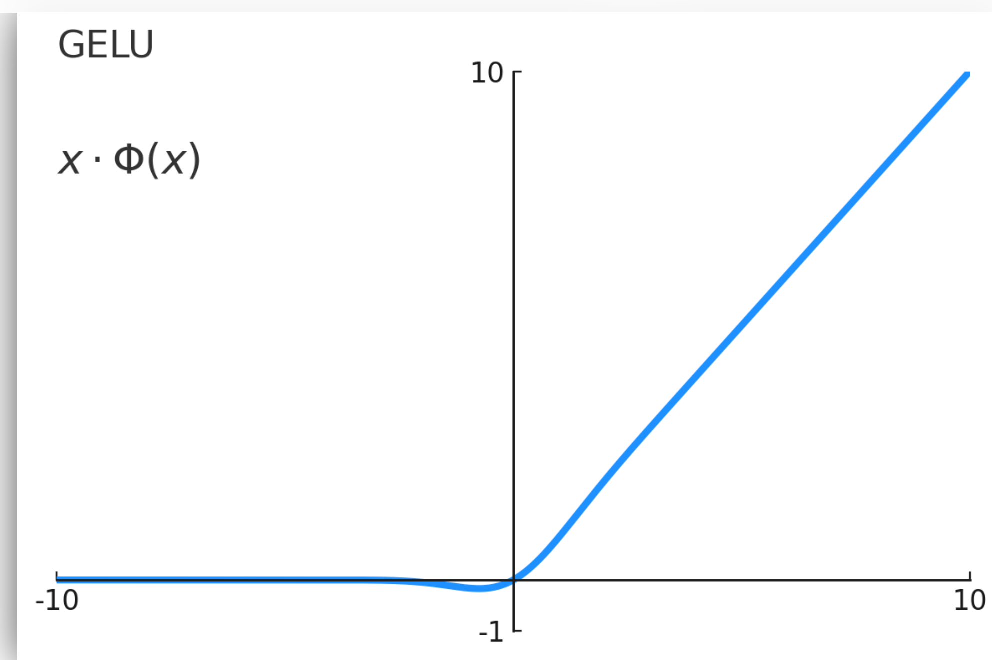


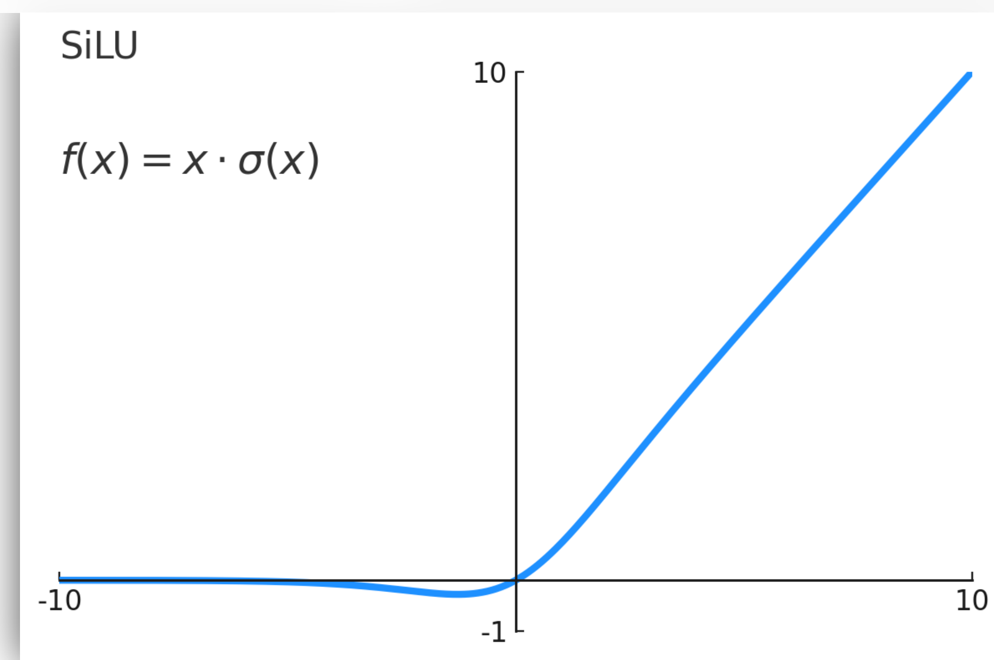


Activation Function Zoo

Q: Where are activations used in CNNs?

A: Generally placed after linear operators (feedforward/linear layer, convolutional layer, etc.)





Lecture Overview – Two Broad Sets of Topics

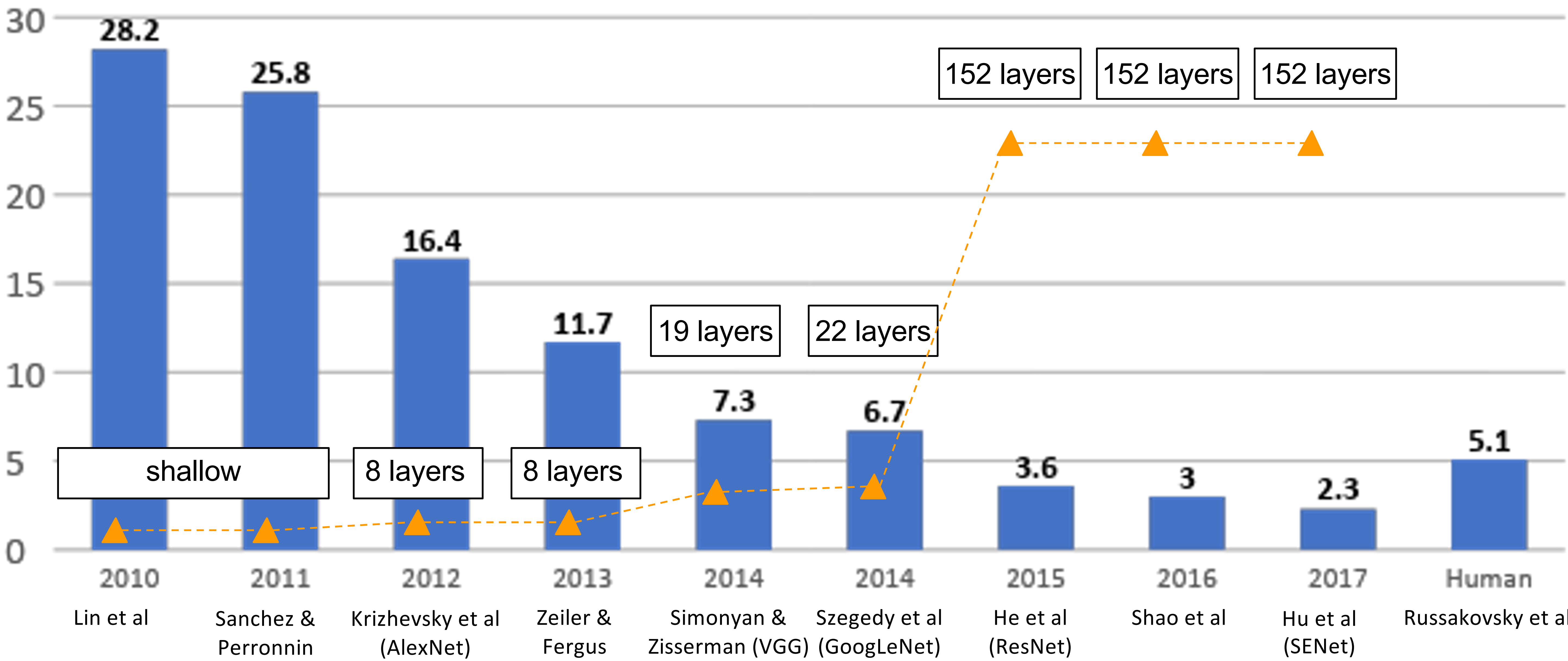
How to build CNNs?

- **Layers in CNNs**
- **Activation Functions**
- **CNN Architectures**
- **Weight Initialization**

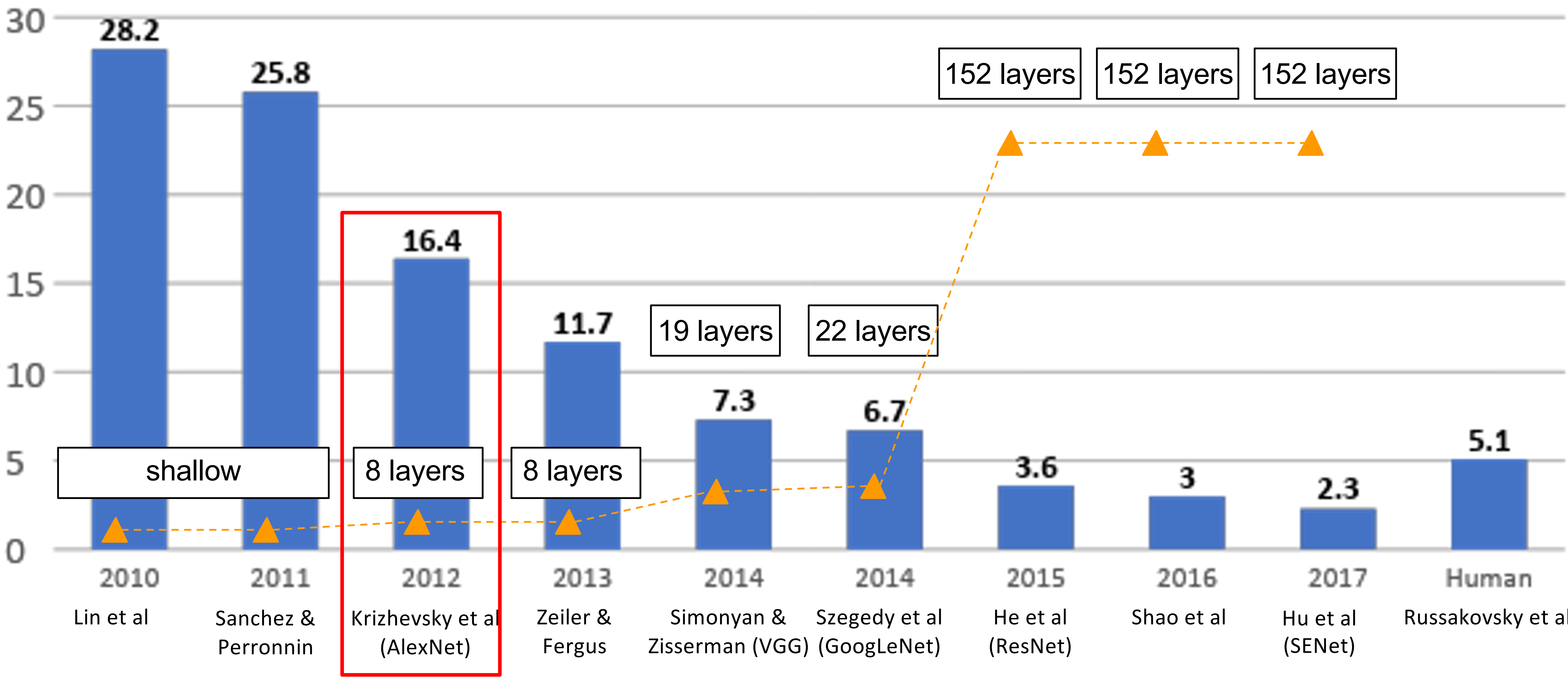
How to train CNNs?

- **Data Preprocessing**
- **Data augmentation**
- **Transfer Learning**
- **Hyperparameter Selection**

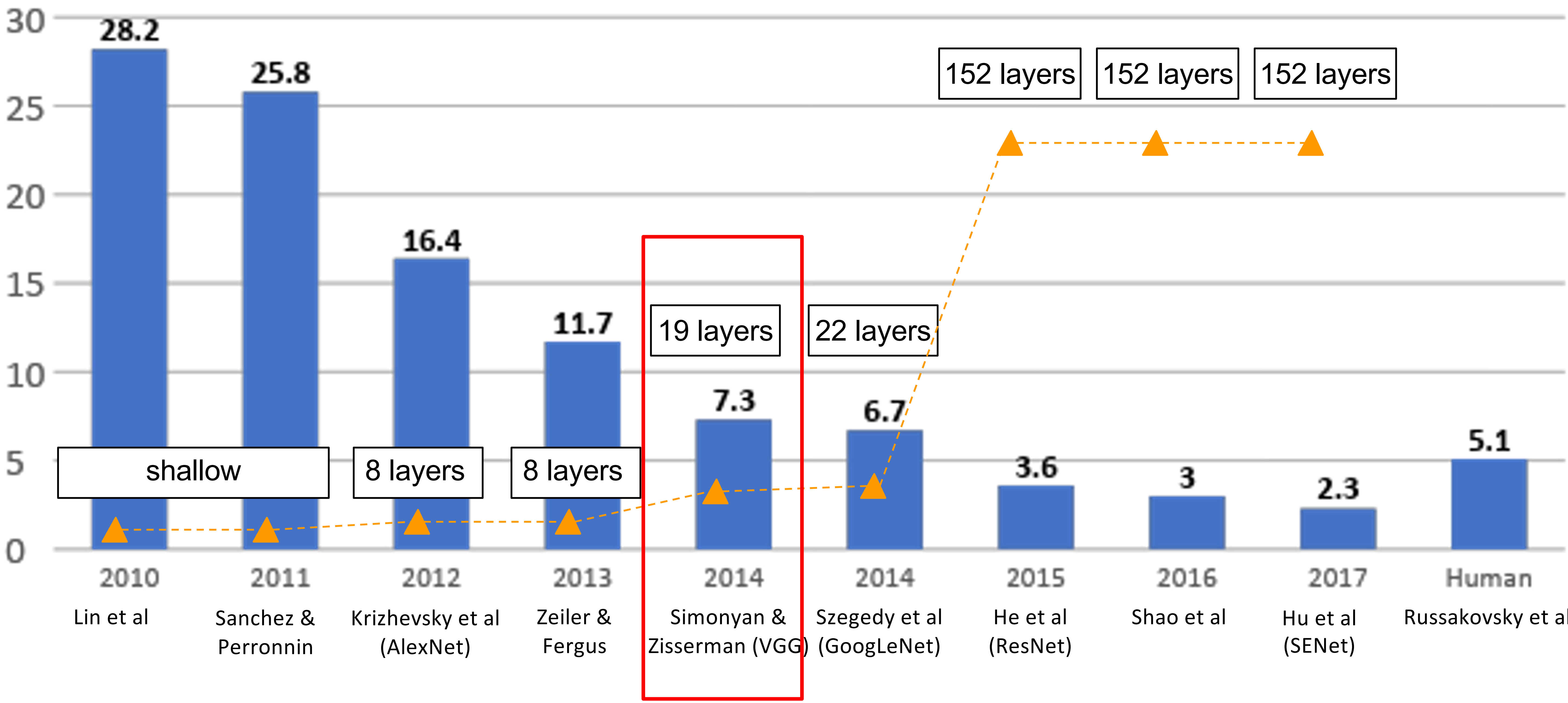
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: VGGNet

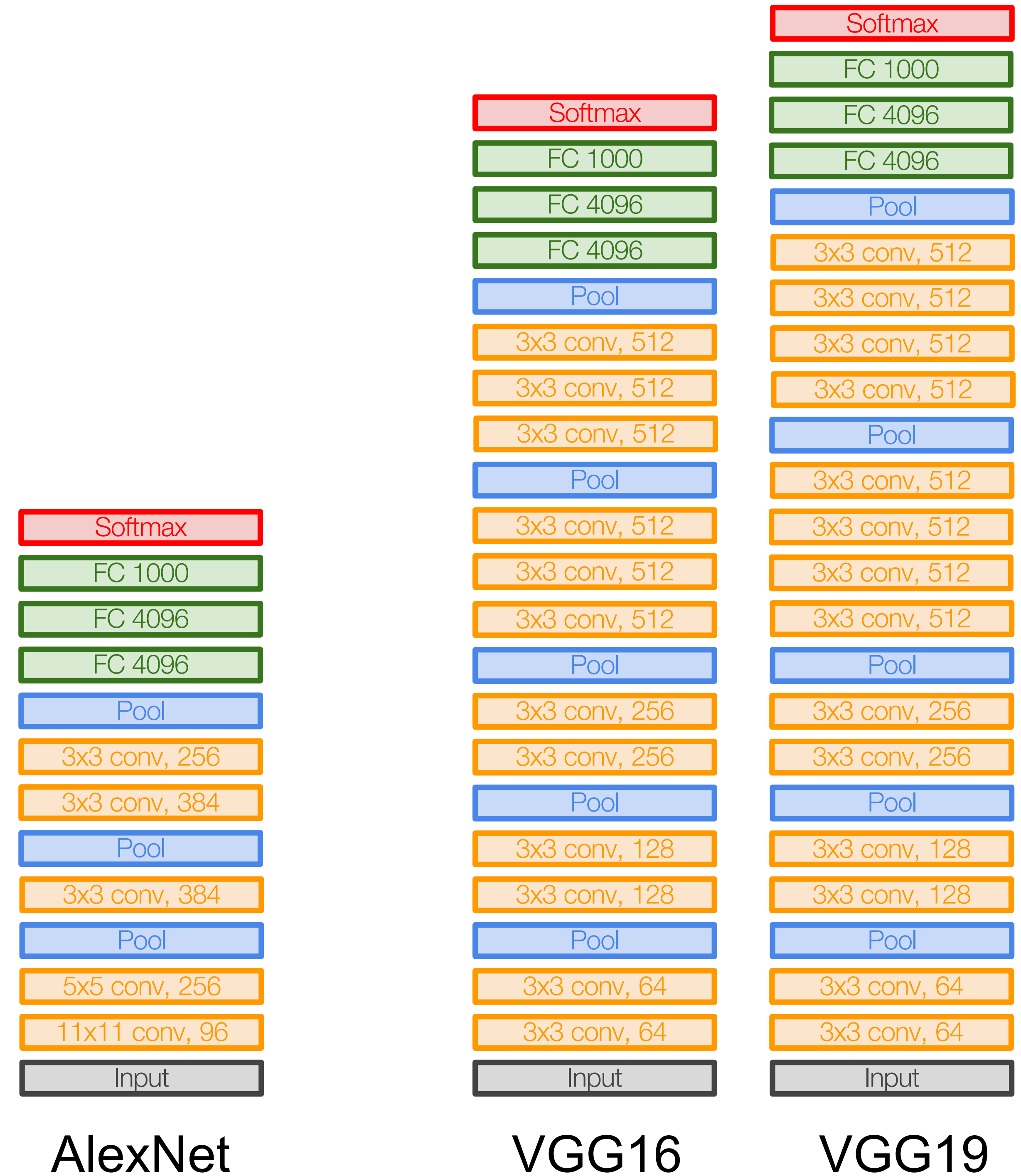
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

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

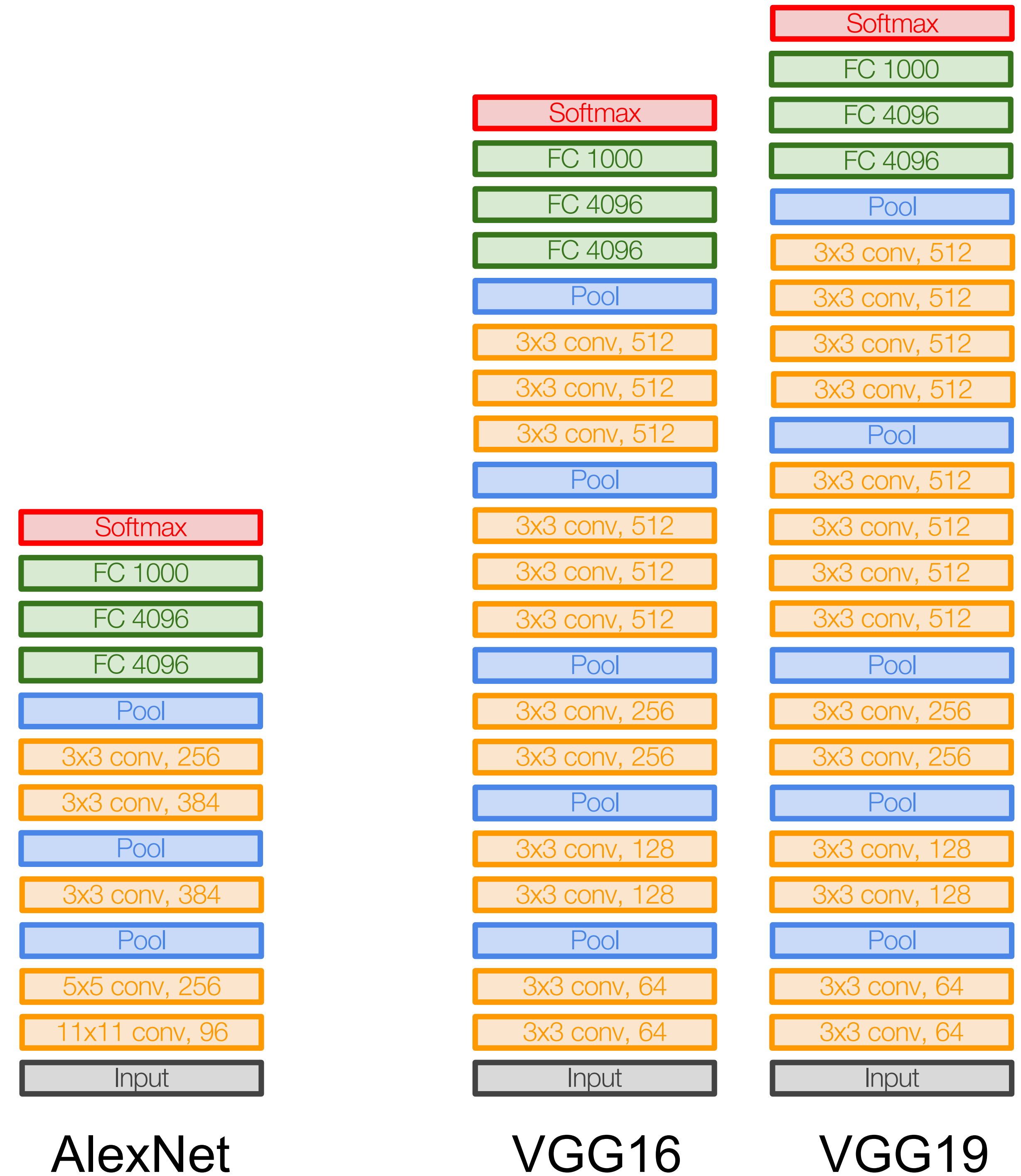
11.7% top 5 error in ILSVRC’13
(ZFNet)
-> 7.3% top 5 error in ILSVRC’14



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

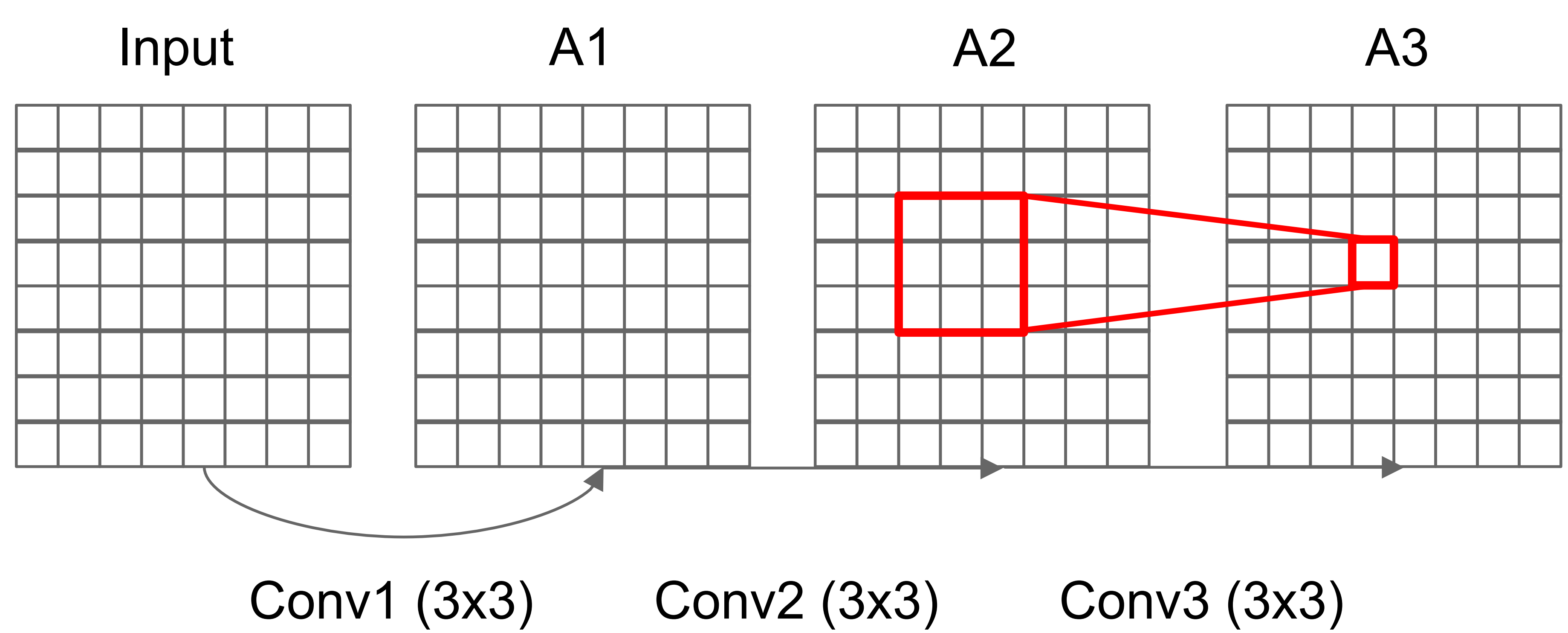
Q: Why use smaller filters? (3x3 conv)



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

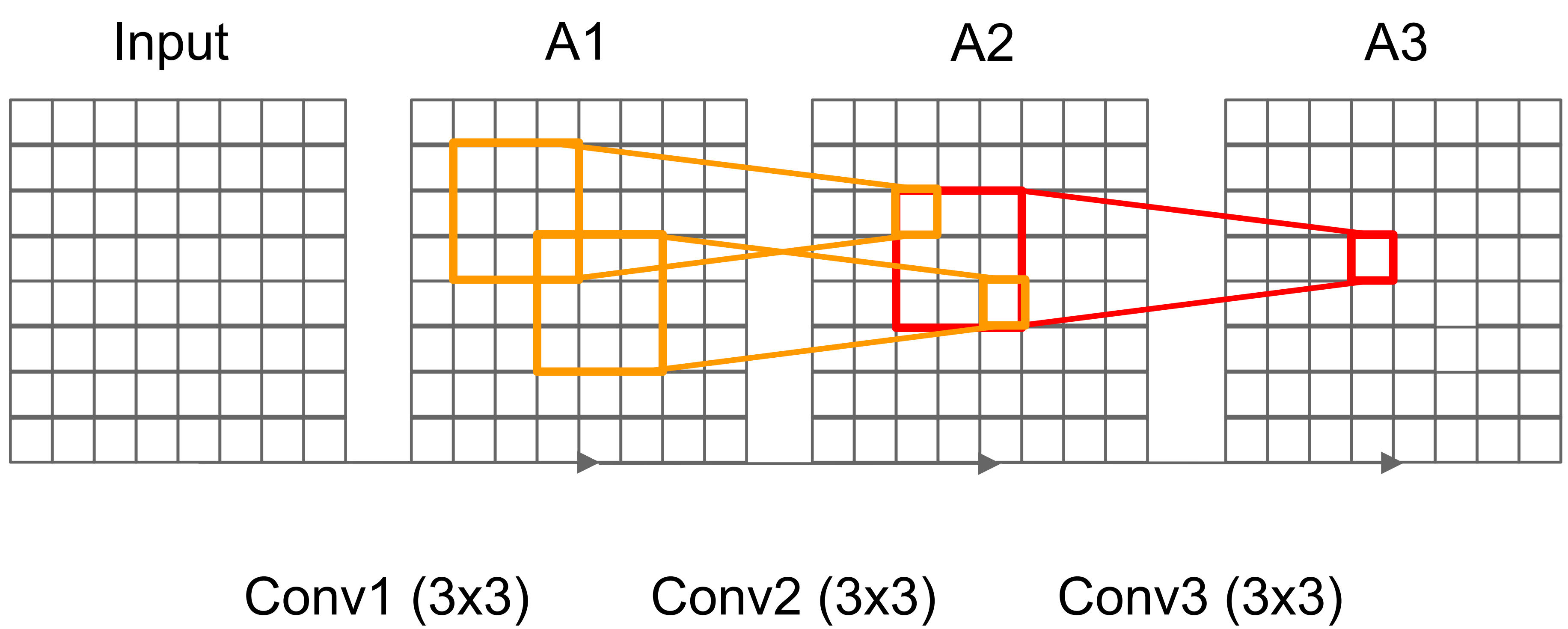
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

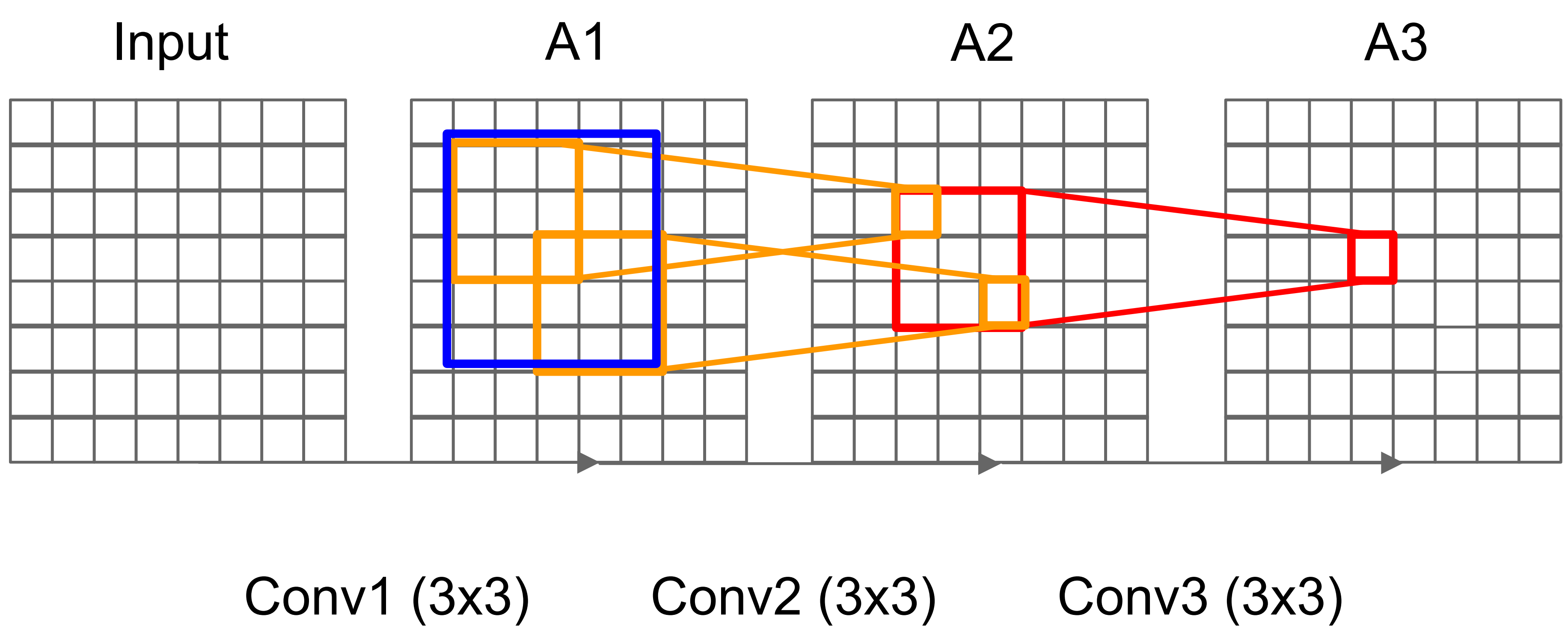
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

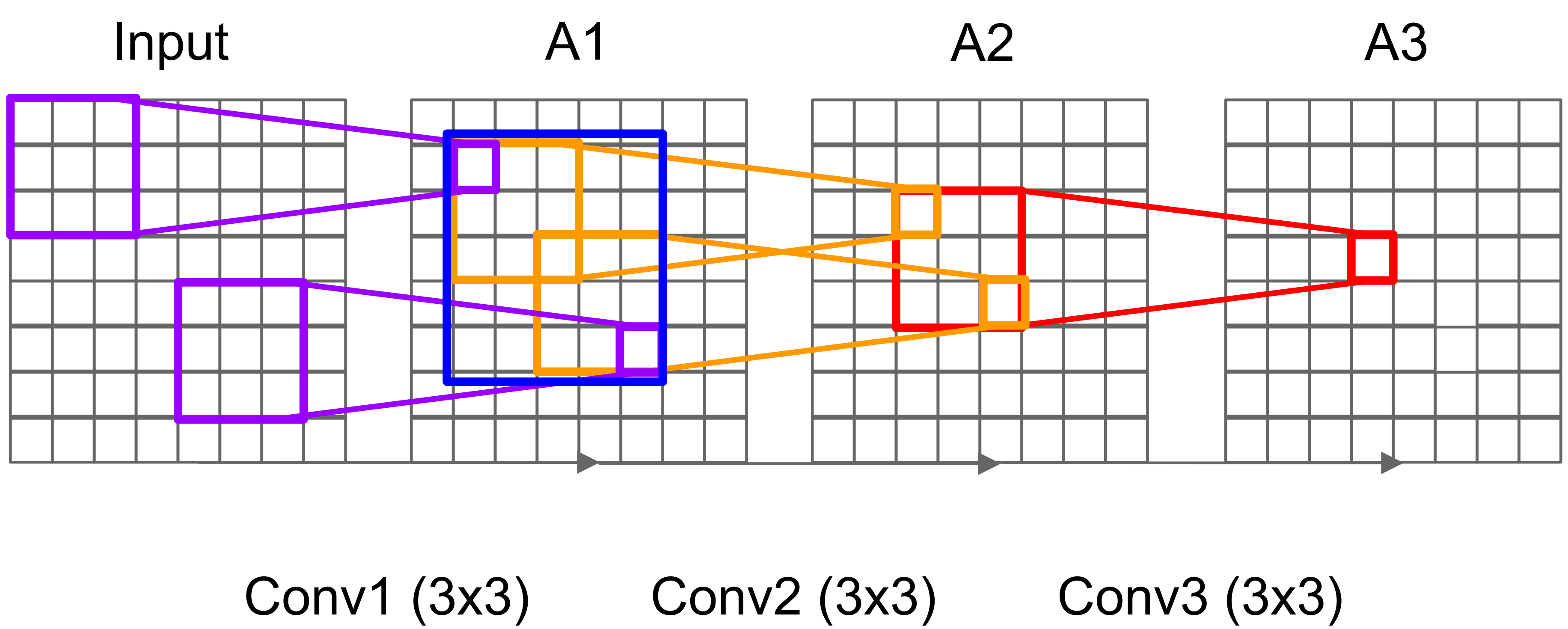
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

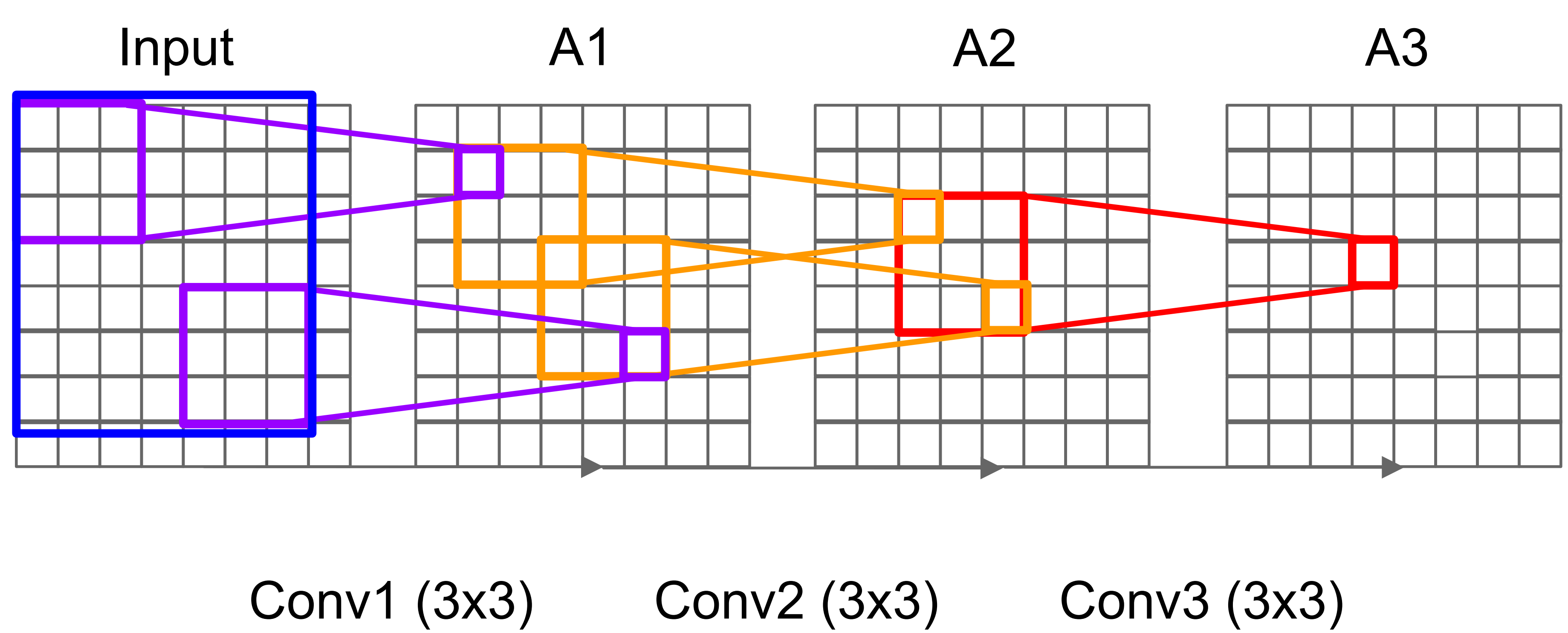
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

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



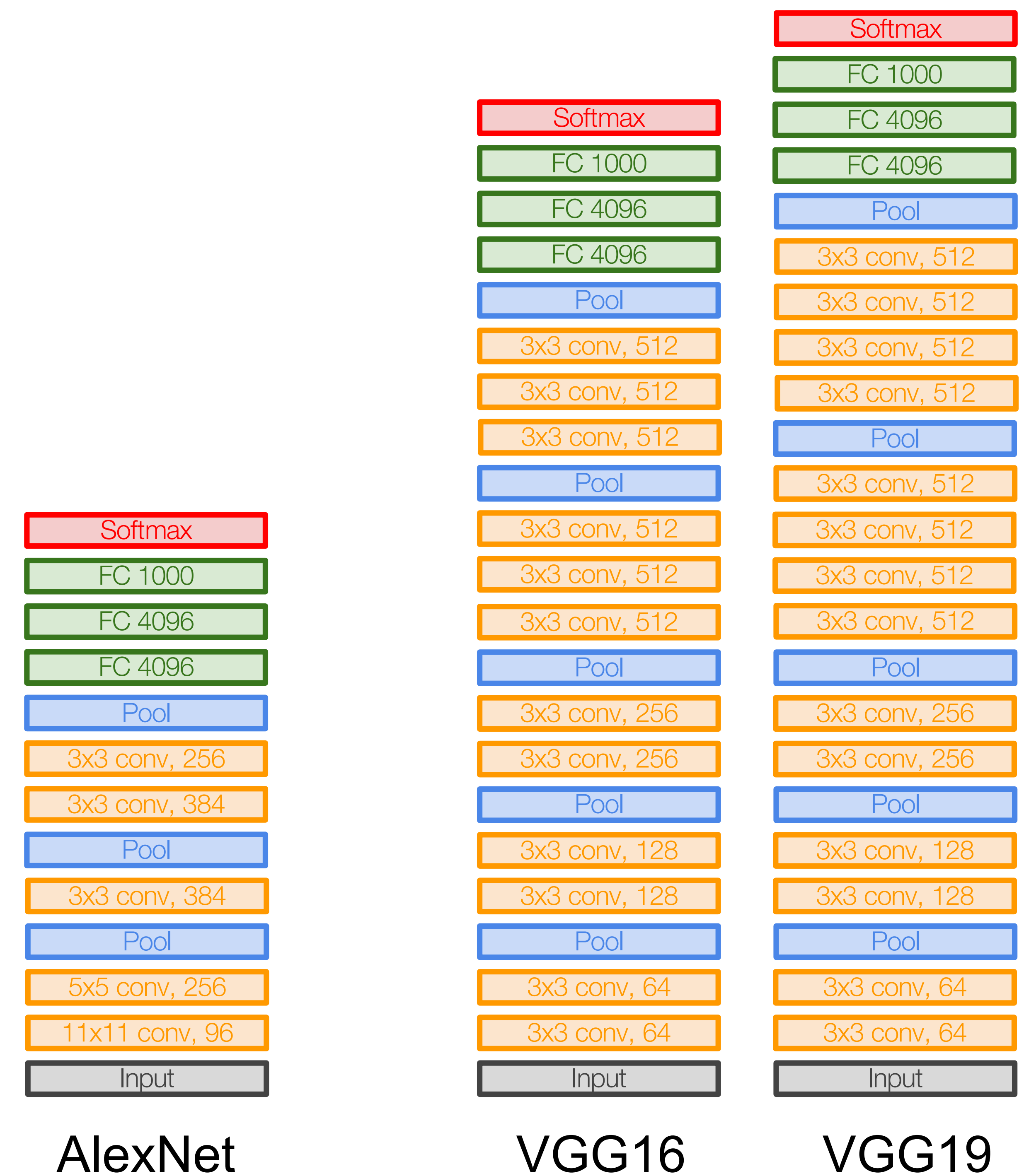
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

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

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

[7x7]



Case Study: VGGNet

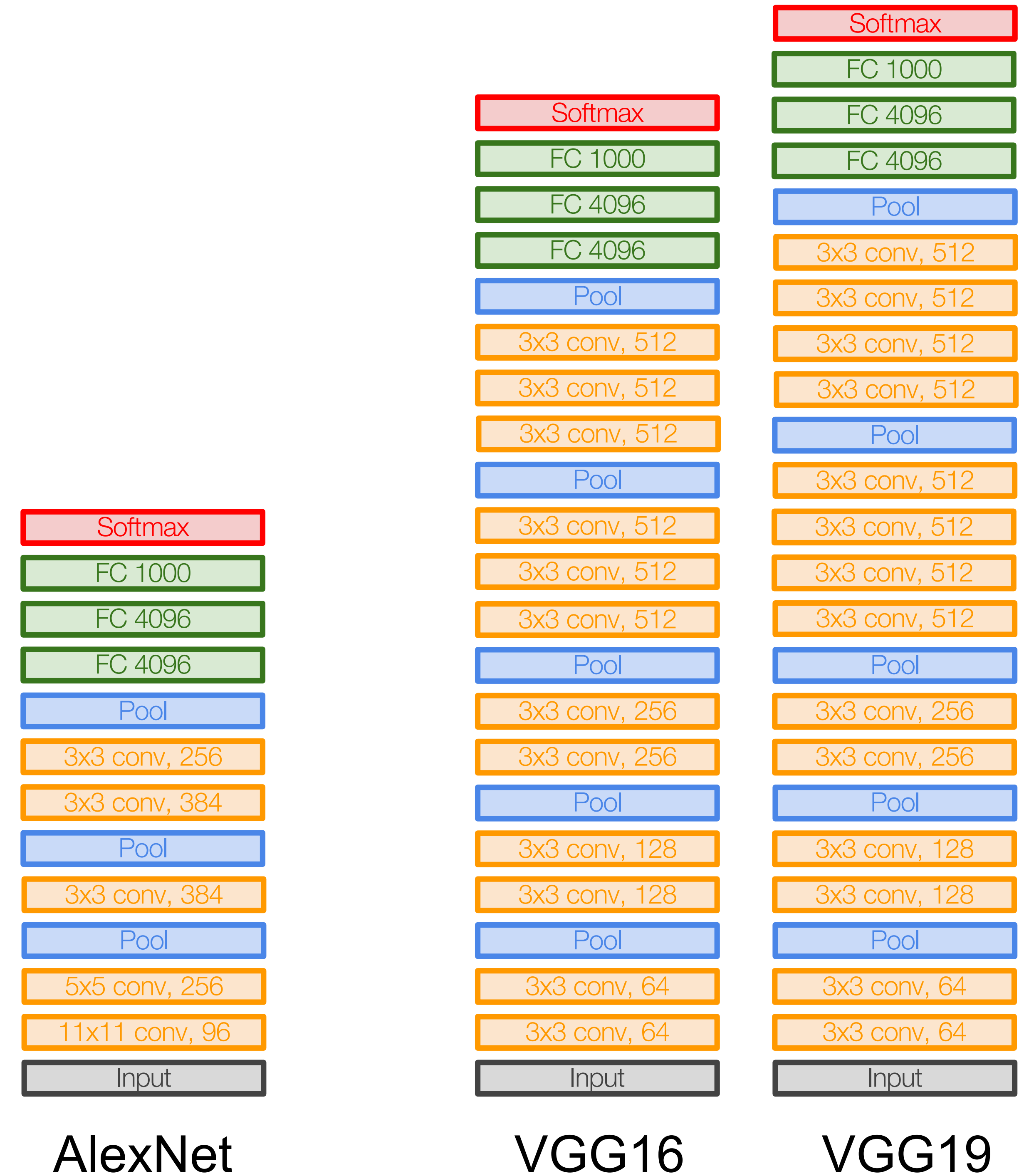
[Simonyan and Zisserman, 2014]

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

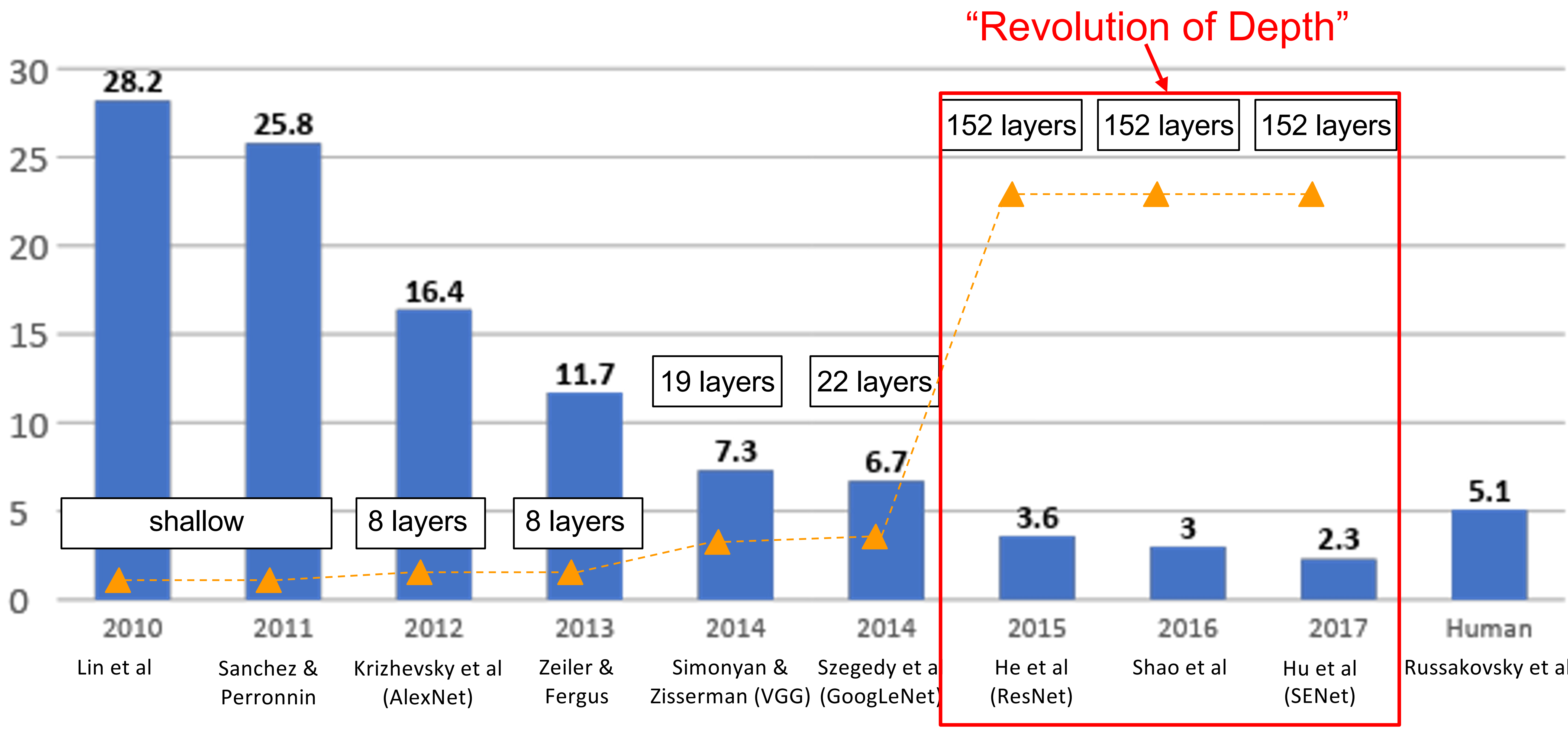
Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

But deeper, more non-linearities

And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: ResNet

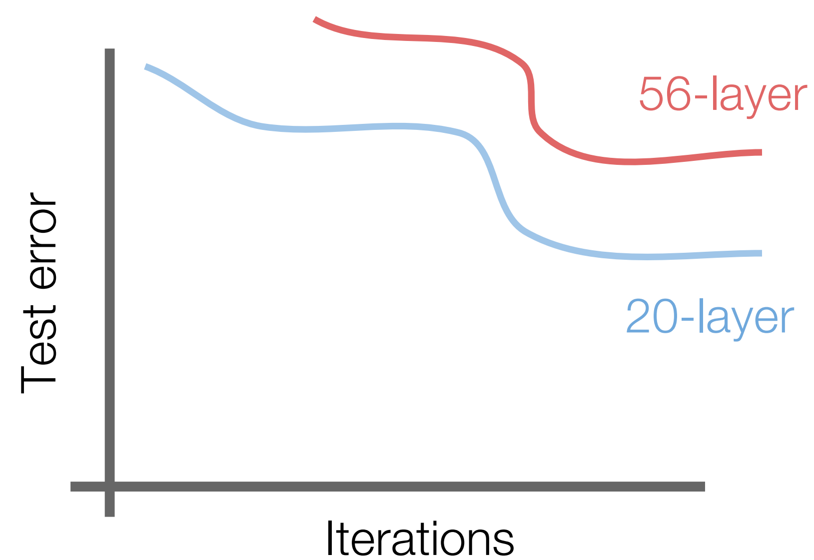
[He et al., 2015]

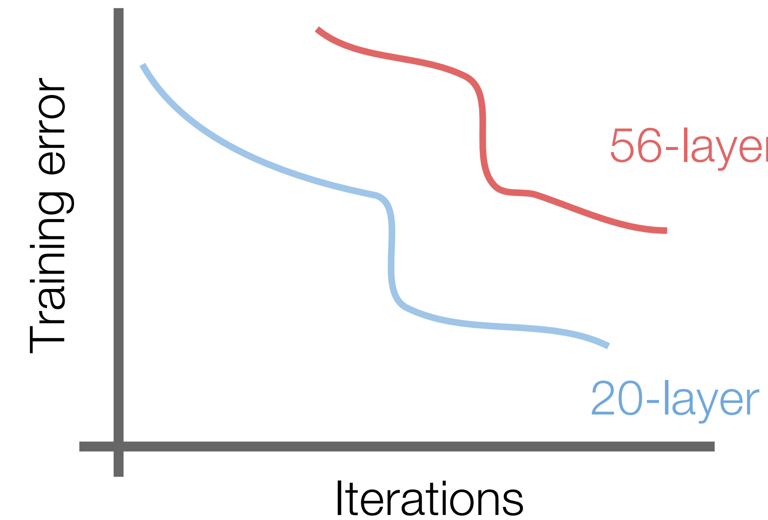
What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

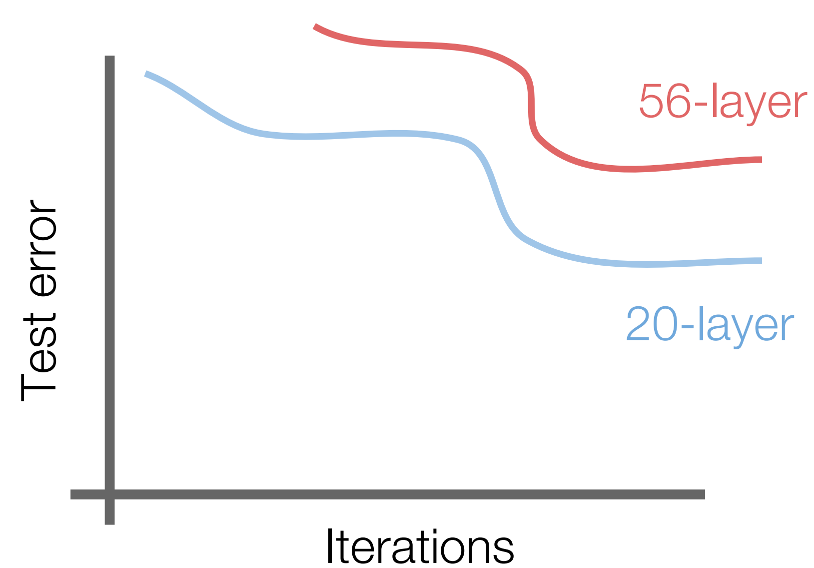


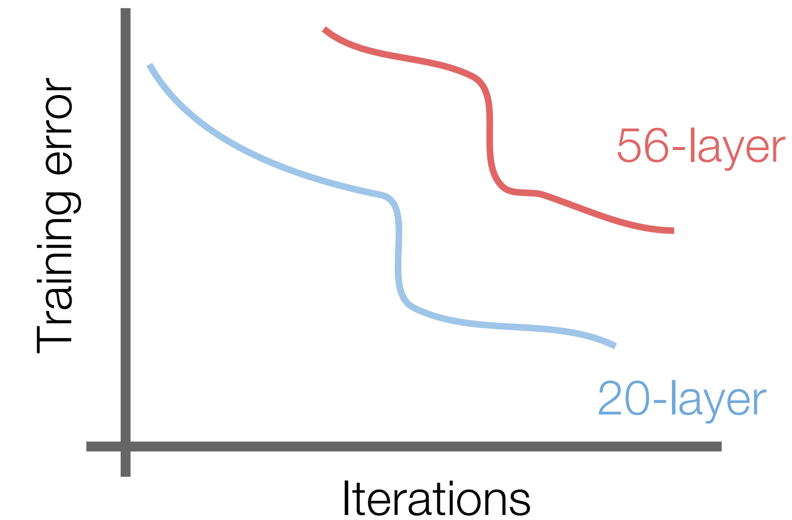


Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?





56-layer model performs worse on both test and training error
-> The deeper model performs worse, but it’s not caused by overfitting!

Case Study: ResNet

[He et al., 2015]

Fact: Deep models have more representation power (more parameters) than shallower models.

Hypothesis: the problem is an *optimization* problem, **deeper models are harder to optimize**

Case Study: ResNet

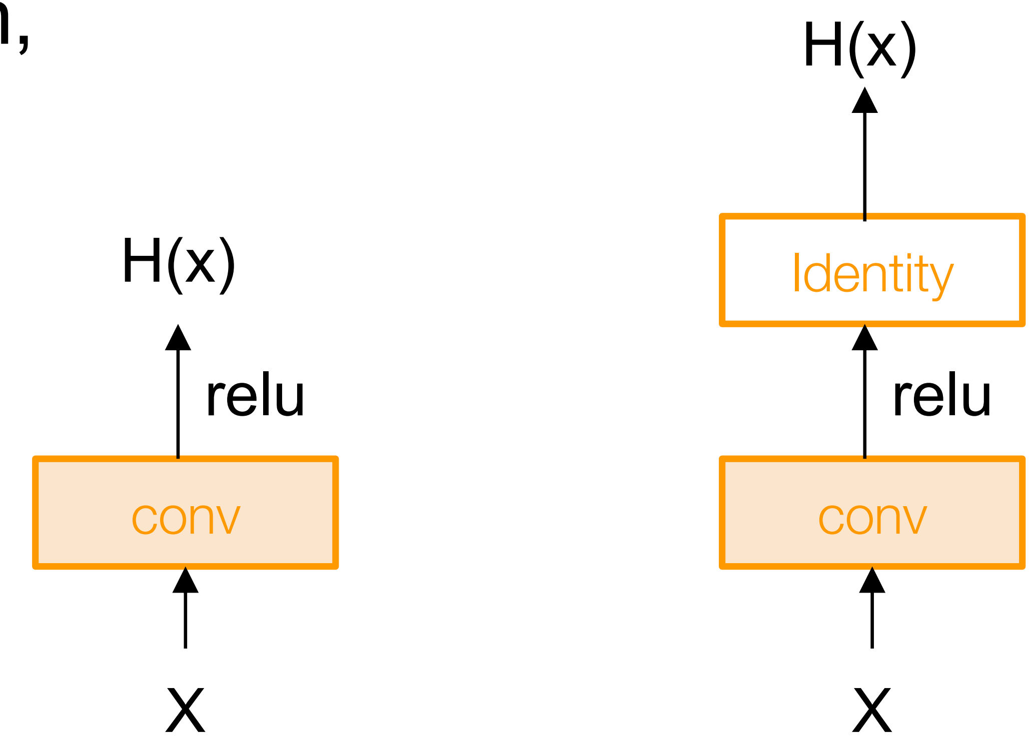
[He et al., 2015]

Fact: Deep models have more representation power (more parameters) than shallower models.

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

What should the deeper model learn to be at least as good as the shallower model?

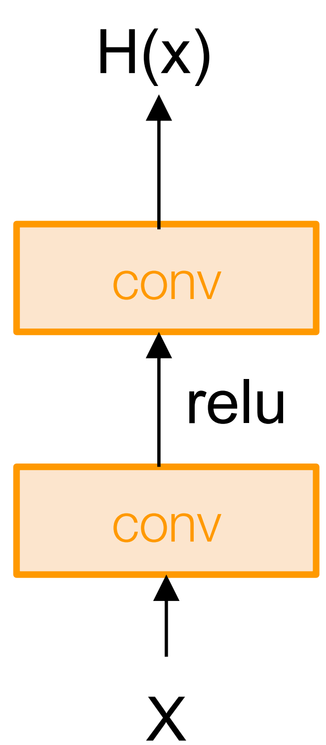
A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.



Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

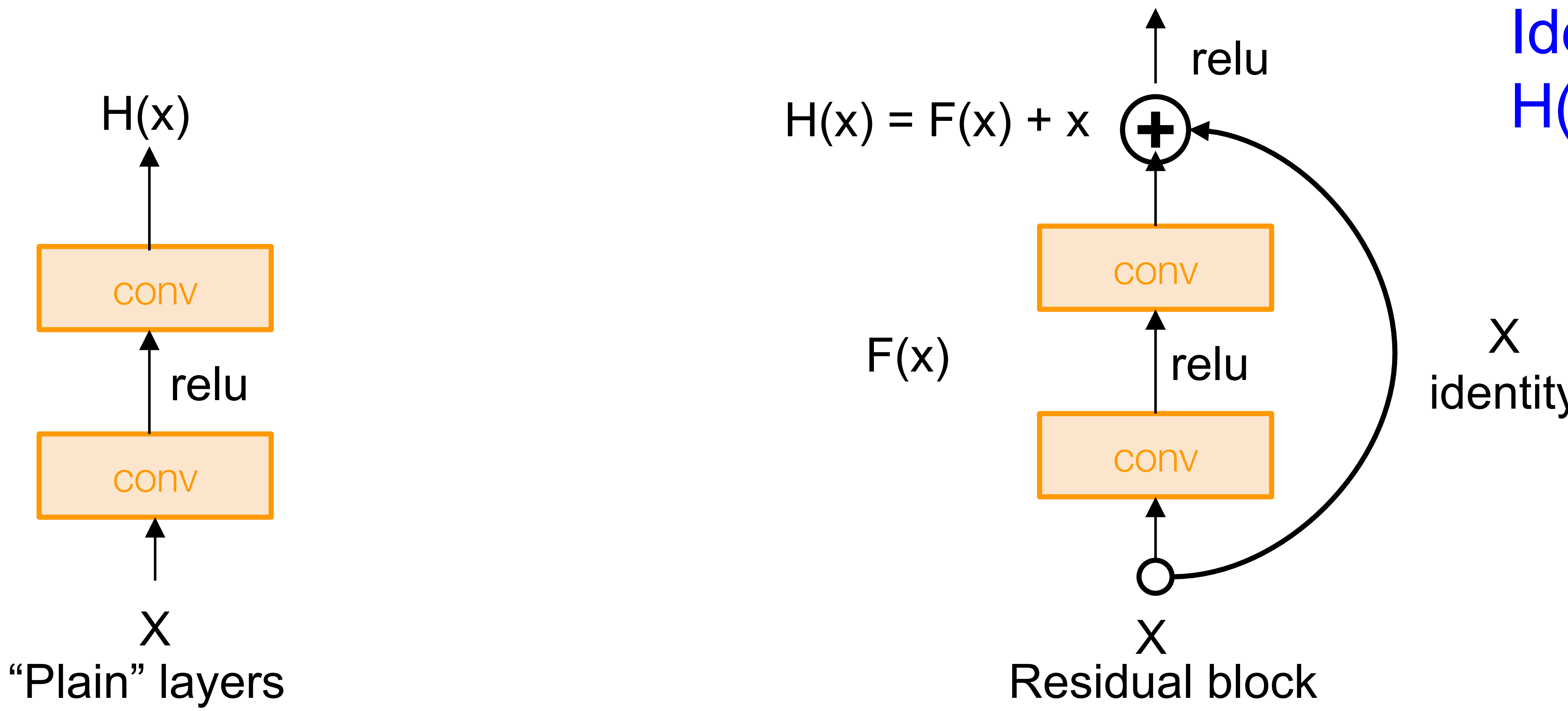


“Plain” layers

Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

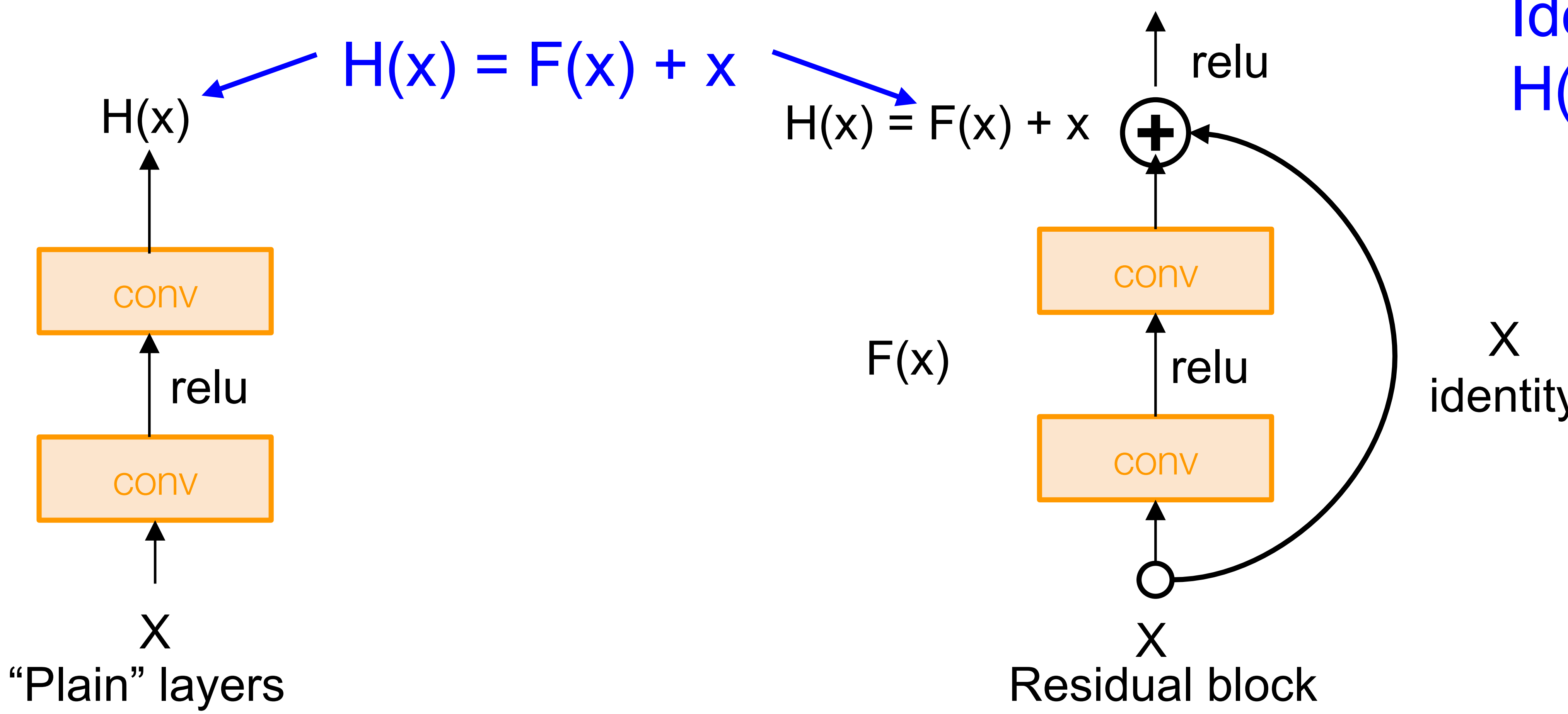


Identity mapping:
$H(x) = x$ if $F(x) = 0$

Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



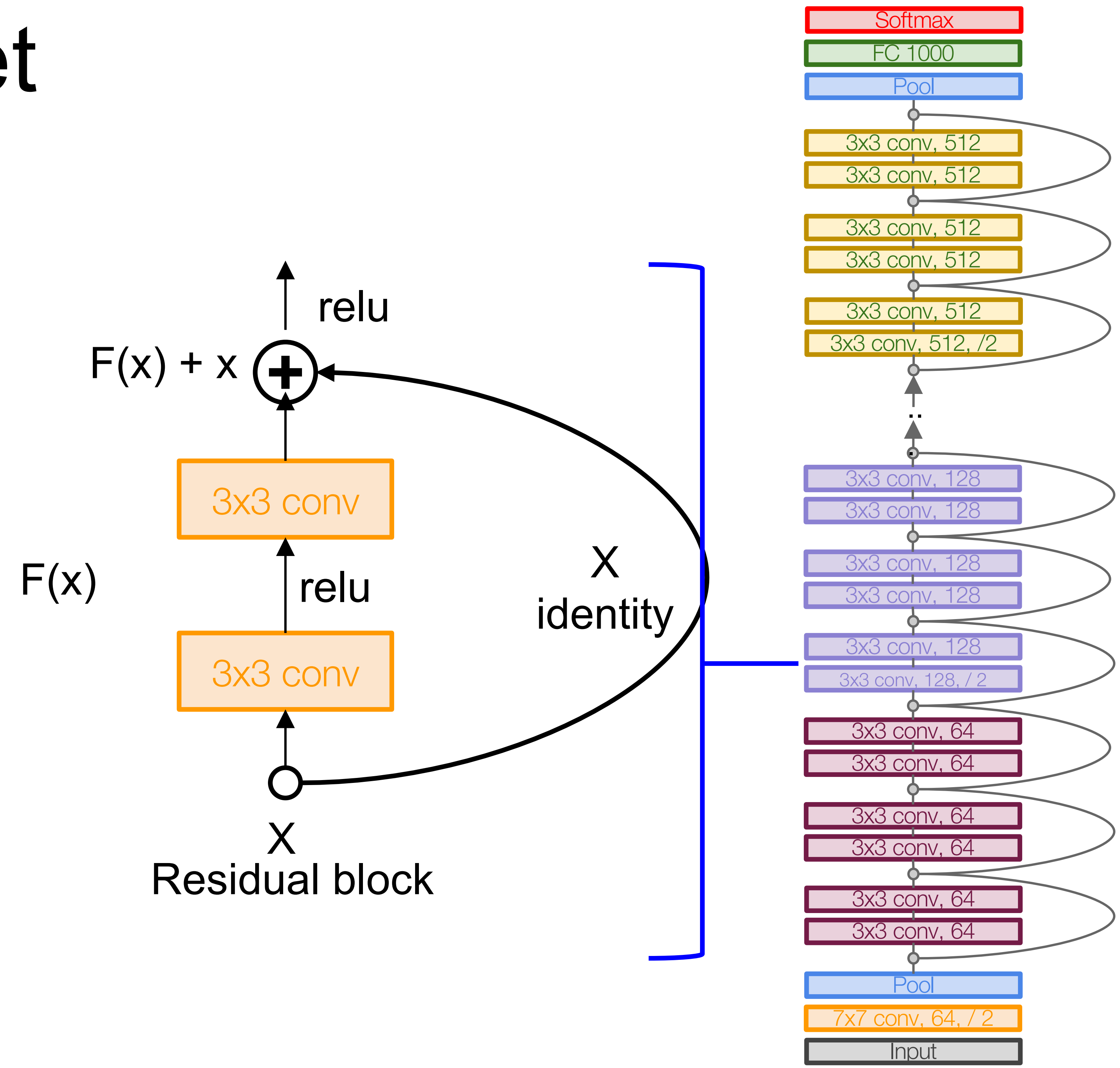
Identity mapping:
H(x) = x if F(x) = 0

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

Case Study: ResNet

[He et al., 2015]

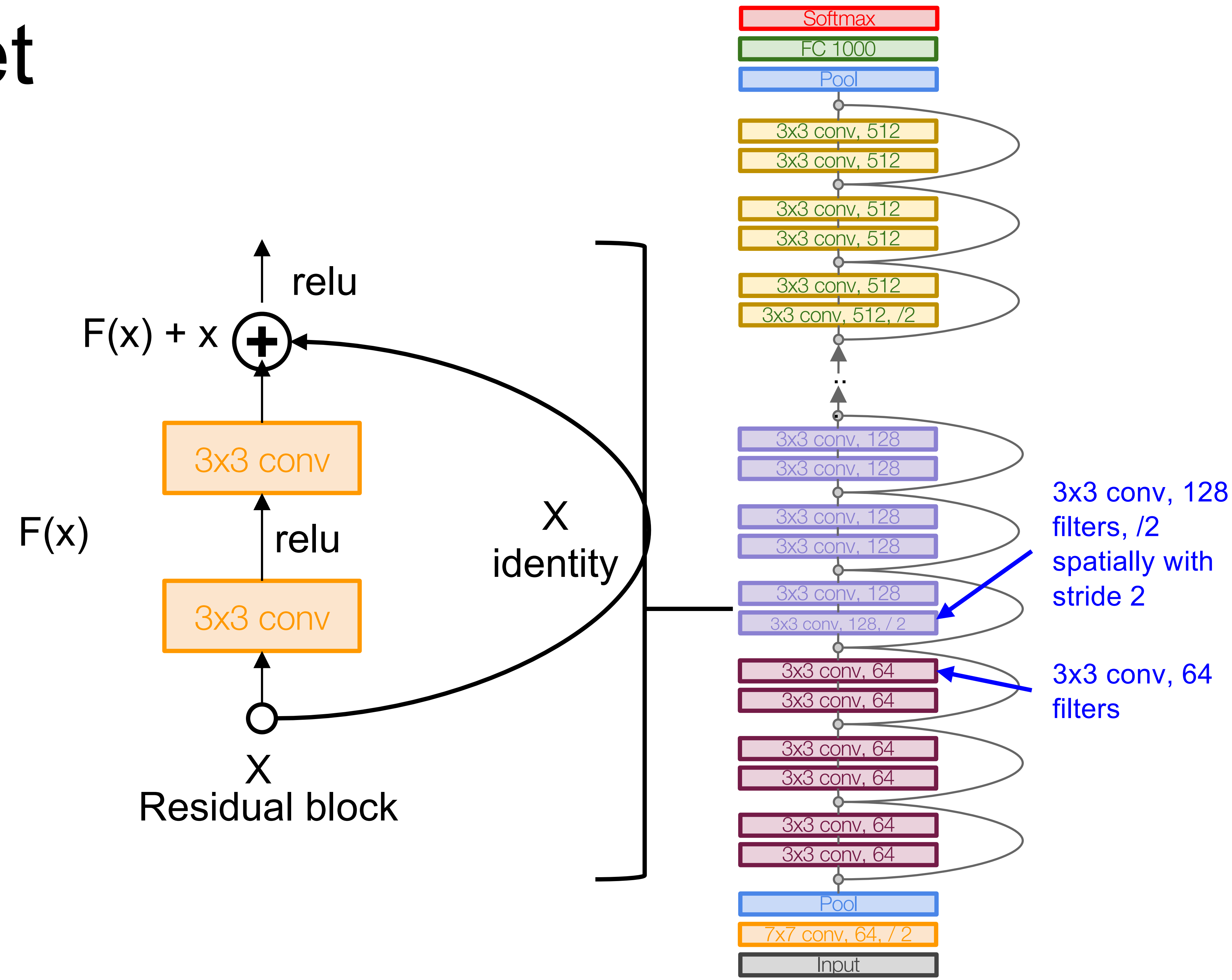
Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers



Case Study: ResNet

[He et al., 2015]

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

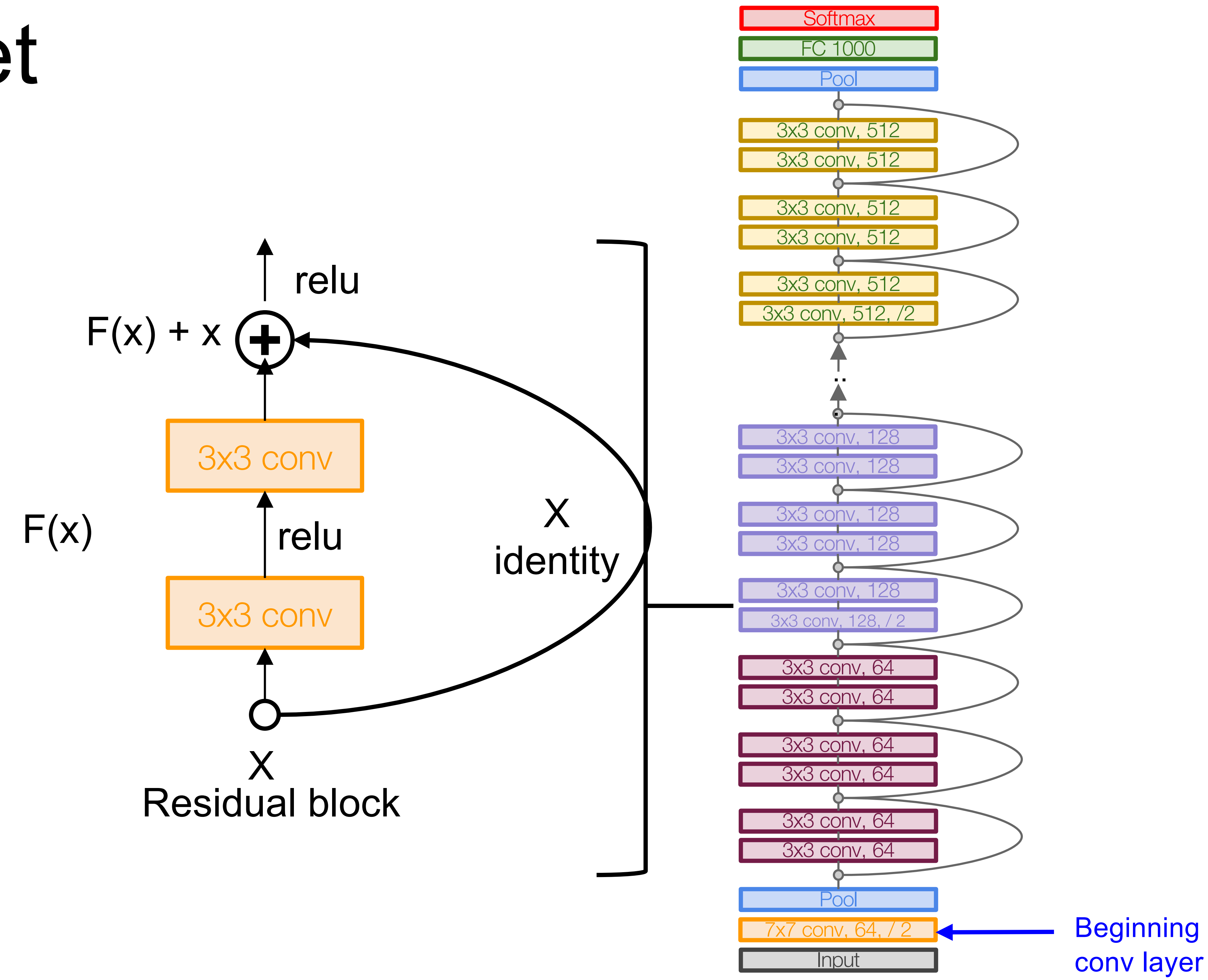


Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

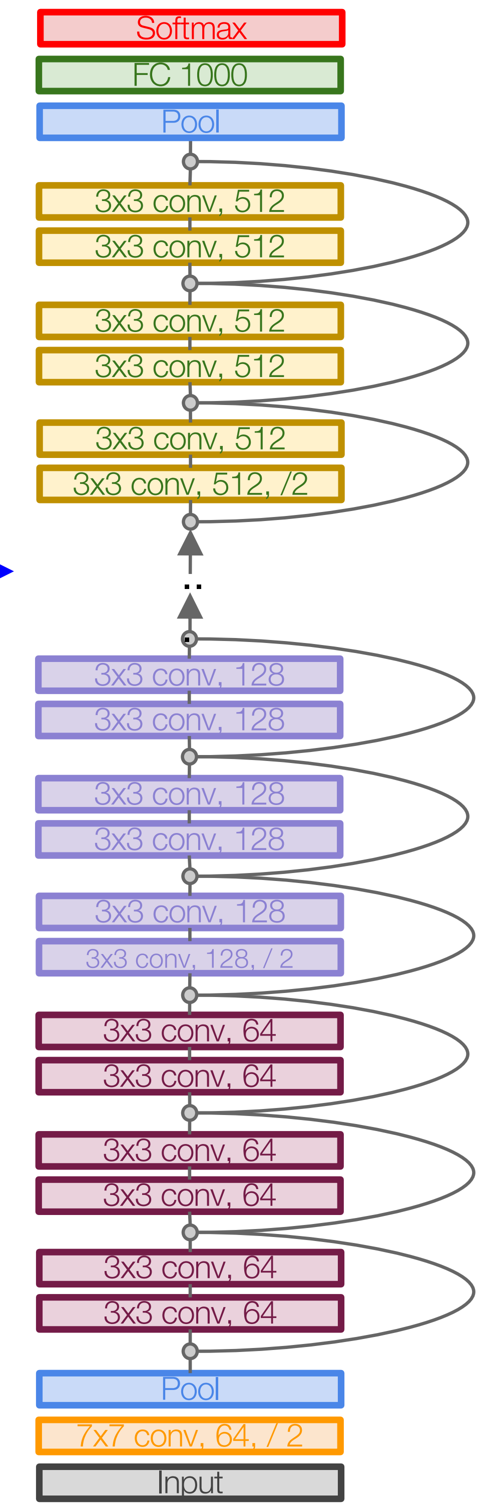
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning (stem)



Case Study: ResNet

[He et al., 2015]

Total depths of 18, 34, 50, 101, or 152 layers for ImageNet

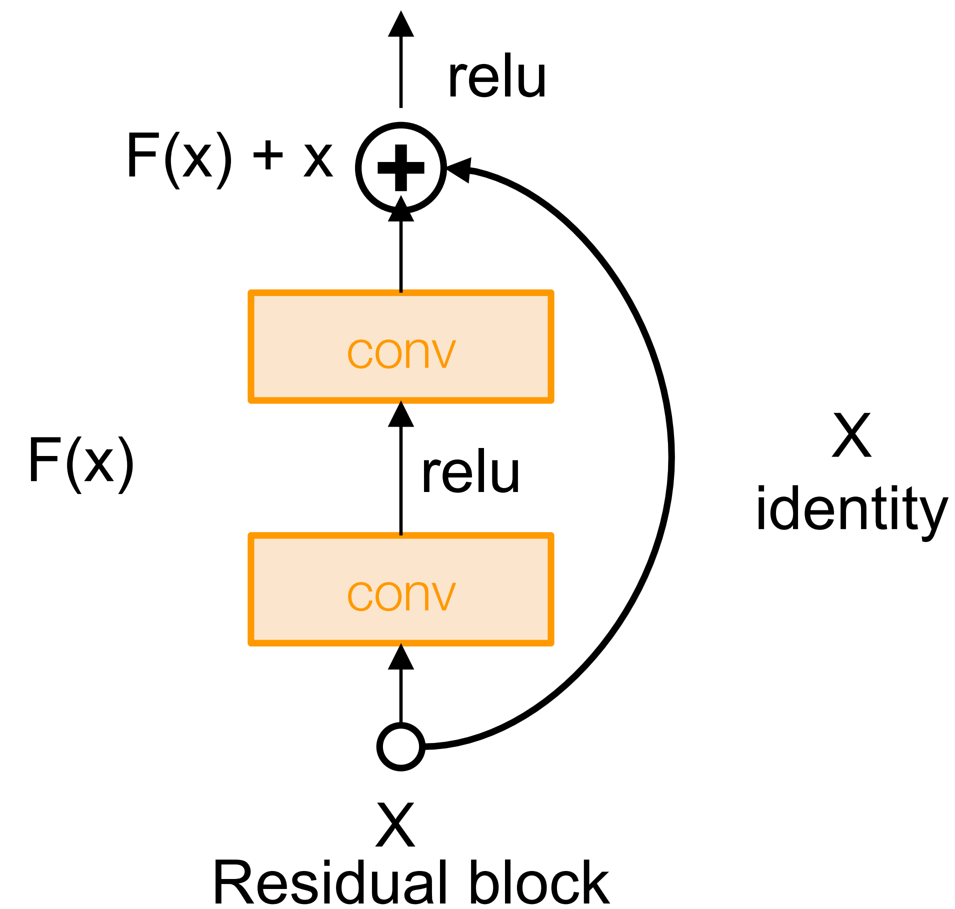


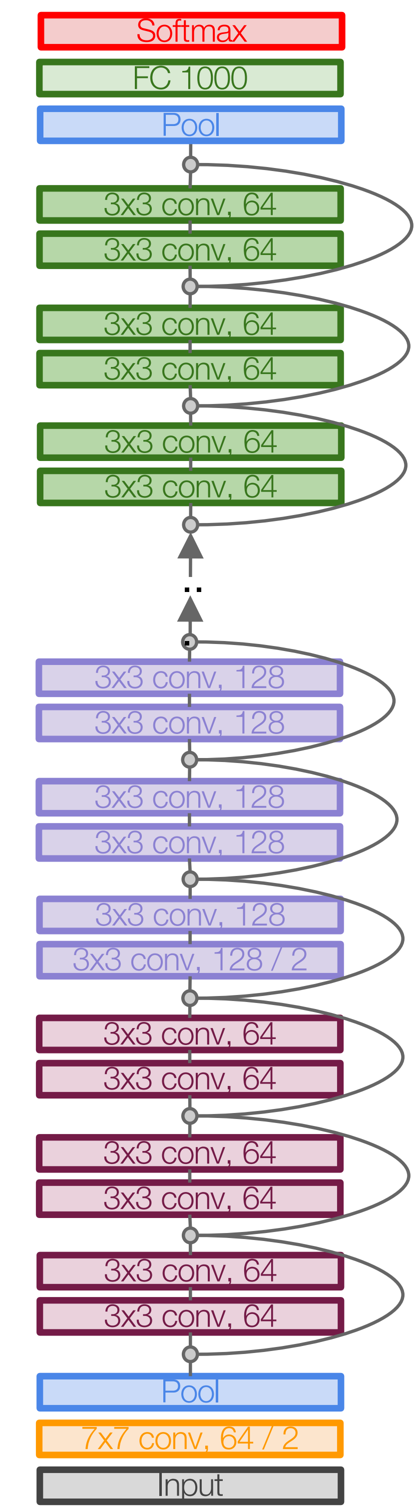
Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!





Lecture Overview – Two Broad Sets of Topics

How to build CNNs?

- **Layers in CNNs**
- **Activation Functions**
- **CNN Architectures**
- **Weight Initialization**

How to train CNNs?

- **Data Preprocessing**
- **Data augmentation**
- **Transfer Learning**
- **Hyperparameter Selection**

How to initialize weights in neural network layers?

Weight Initialization Case: Values too small

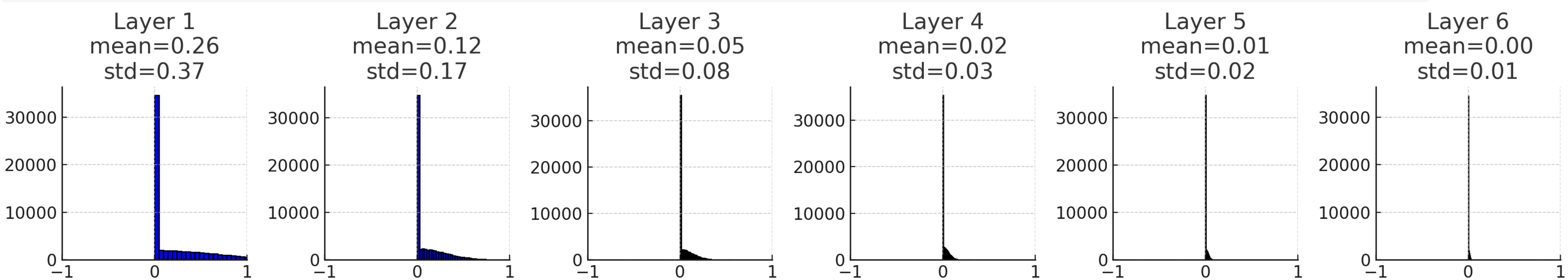
```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
# Forward pass with ReLU activation
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.01 * np.random.randn(Din, Dout)  # Small weight init
    x = np.maximum(0, x.dot(W))  # ReLU activation
    hs.append(x)
```

Forward pass for a 6-layer net with hidden size 4096

Weight Initialization Case: Values too small

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
# Forward pass with ReLU activation
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.01 * np.random.randn(Din, Dout) # Small weight init
    x = np.maximum(0, x.dot(W))  # ReLU activation
    hs.append(x)
```

All activations tend to zero for deeper network layers



Weight Initialization Case: Values too large

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
# Forward pass with ReLU activation
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.05 * np.random.randn(Din, Dout)  # Small weight init
    x = np.maximum(0, x.dot(W))  # ReLU activation
    hs.append(x)
```

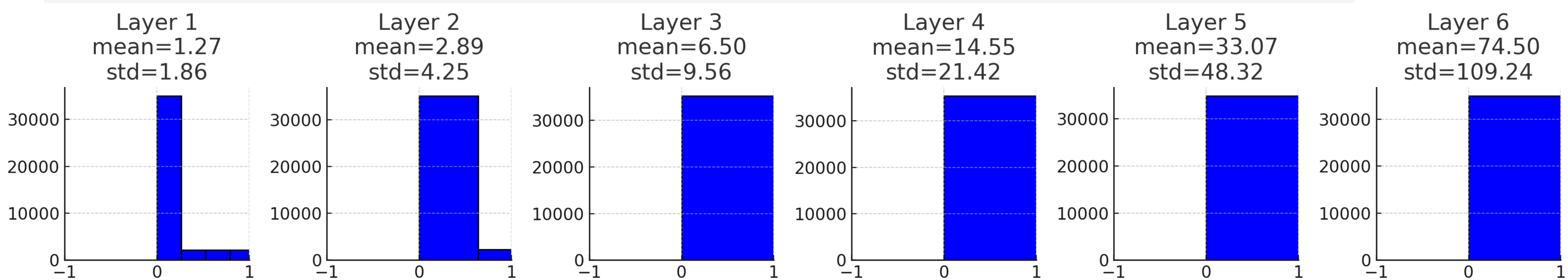
Increase std of initial weights from 0.01 to 0.05

Weight Initialization Case: Values too large

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
# Forward pass with ReLU activation
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.05 * np.random.randn(Din, Dout)  # Small weight init
    x = np.maximum(0, x.dot(W))  # ReLU activation
    hs.append(x)
```

Activations blow up quickly

Increase std of initial weights from 0.01 to 0.05



How to fix this? Depends on the size of the layer

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) * np.sqrt(2/Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

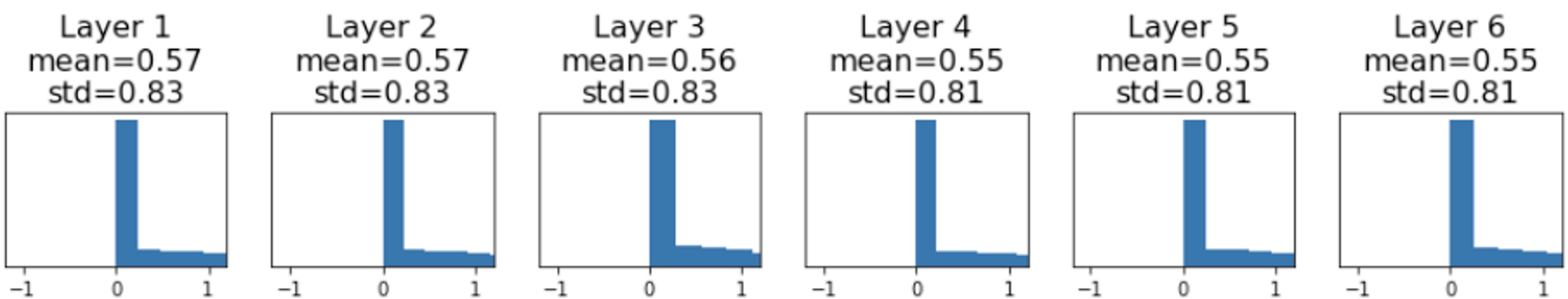
He et al, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”, ICCV 2015

One solution: Kaiming / MSRA Initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) * np.sqrt(2/Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

ReLU correction: std = sqrt(2 / Din)

"Just right": Activations are nicely scaled for all layers!



He et al, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", ICCV 2015

Lecture Overview – Two Broad Sets of Topics

How to build CNNs?

- **Layers in CNNs**
- **Activation Functions**
- **CNN Architectures**
- **Weight Initialization**

How to train CNNs?

- **Data Preprocessing**
- **Data augmentation**
- **Transfer Learning**
- **Hyperparameter Selection**

TLDR for Image Normalization: center and scale for each channel

- Subtract per-channel mean and
  Divide by per-channel std (almost all modern models)
  (stats along each channel = 3 numbers)
- Requires pre-computing means and std for each pixel channel (given your dataset)

```
norm_pixel[i,j,c] = (pixel[i,j,c] - np.mean(pixel[:,:,c])) / np.std(pixel[:,:,c])
```

Lecture Overview – Two Broad Sets of Topics

How to build CNNs?

- **Layers in CNNs**
- **Activation Functions**
- **CNN Architectures**
- **Weight Initialization**

How to train CNNs?

- **Data Preprocessing**
- **Data augmentation**
- **Transfer Learning**
- **Hyperparameter Selection**

Regularization: A common pattern

Training: Add some kind of randomness

$$y = f_W(x, z)$$

Testing: Average out randomness (sometimes approximate)

$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

Regularization: A common pattern

Training: Add some kind of randomness

$$y = f_W(x, z)$$

Testing: Average out randomness (sometimes approximate)

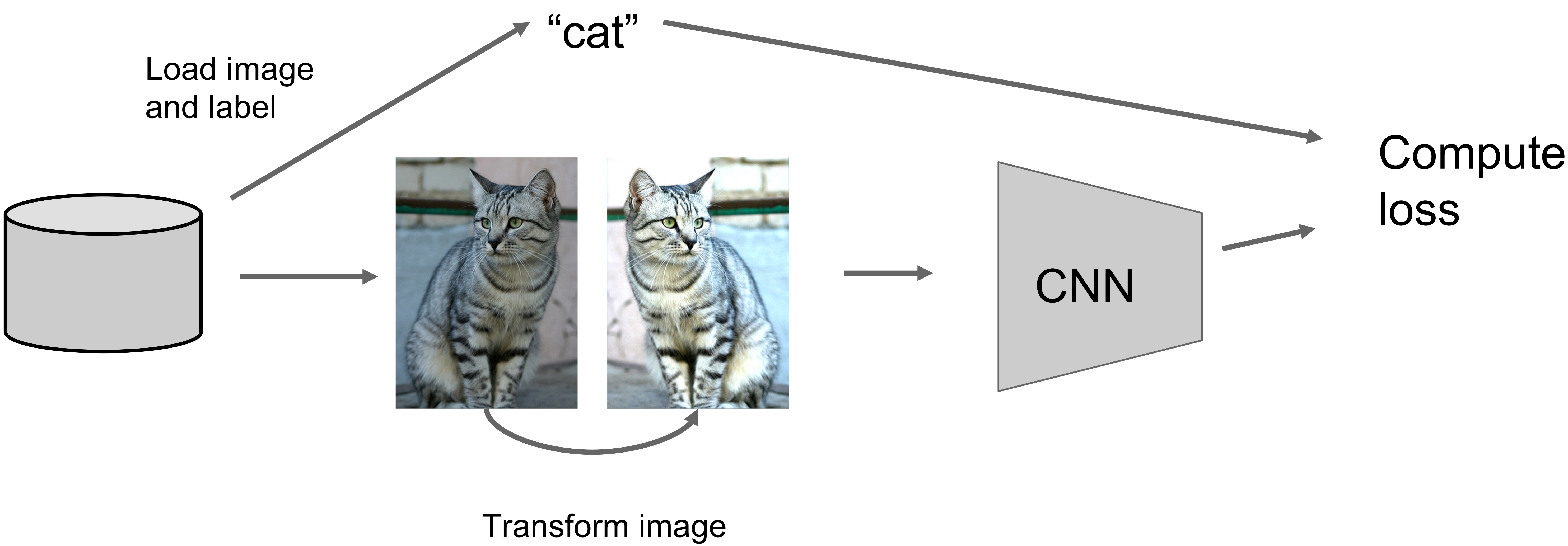
$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

Example: Dropout

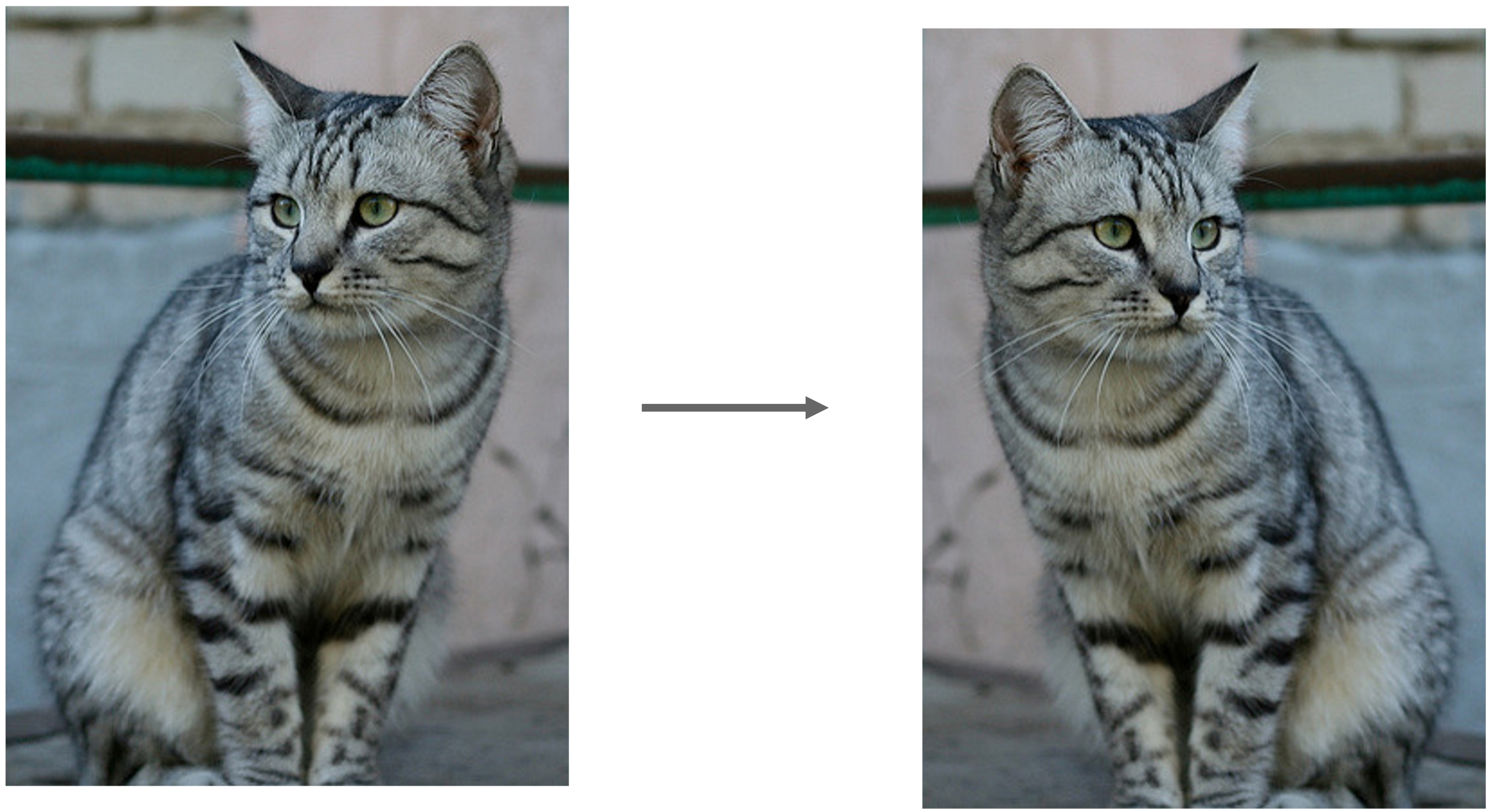
Training: Randomly drop activations

Testing: Use all activations and average values with p

Regularization: Data Augmentation



Data Augmentation
Horizontal Flips



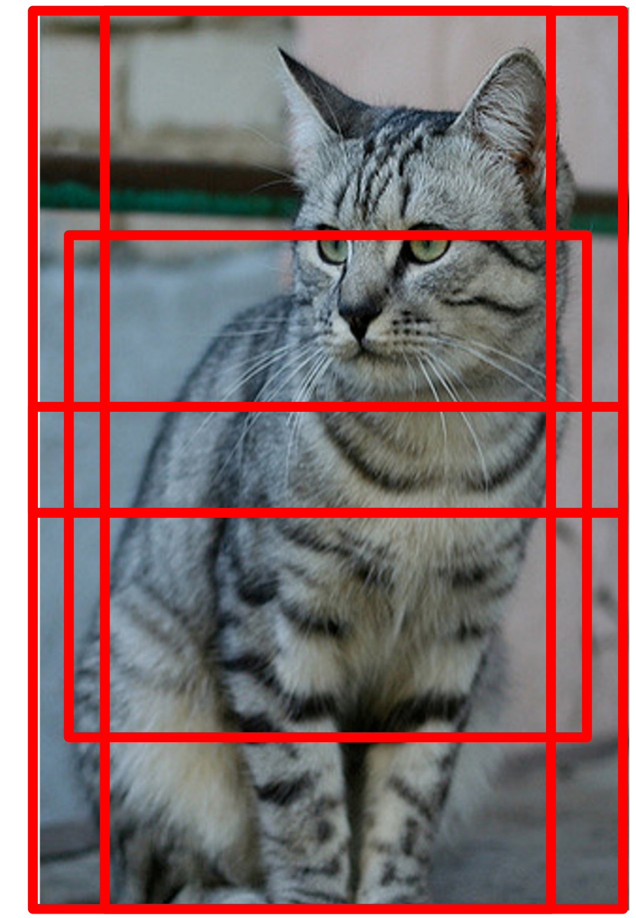
Data Augmentation

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



Test Time Augmentation: 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: 4 corners + center, + flips

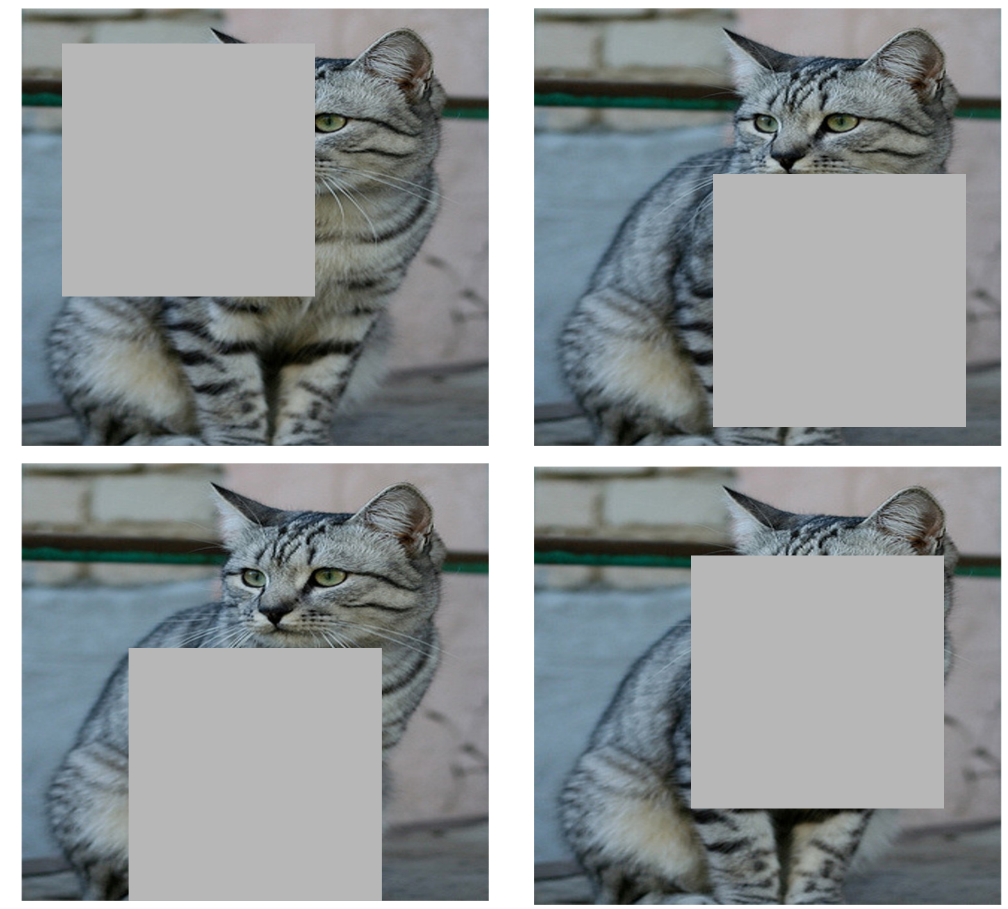
Data Augmentation
Color Jitter

Simple: Randomize
contrast and brightness

Regularization: Cutout

Training: Set random image regions to zero

Testing: Use full image



Examples:

Dropout

Data Augmentation

Cutout / Random Crop

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

Lecture Overview – Two Broad Sets of Topics

How to build CNNs?

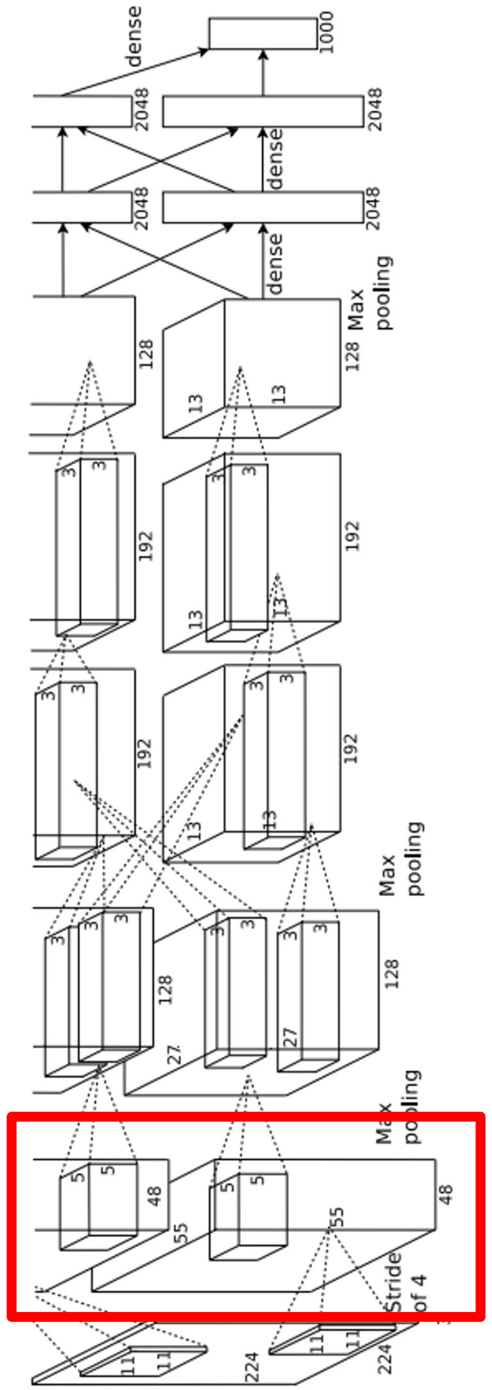
- **Layers in CNNs**
- **Activation Functions**
- **CNN Architectures**
- **Weight Initialization**

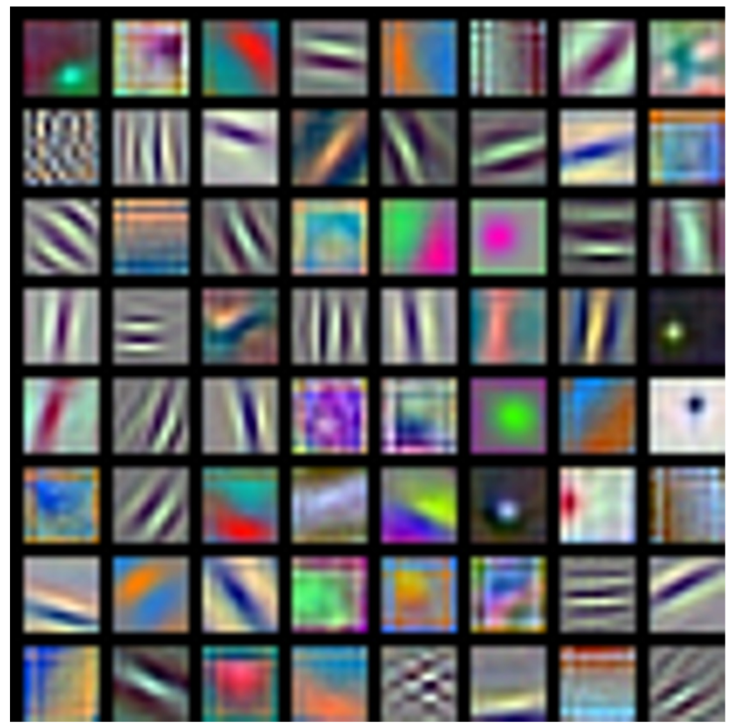
How to train CNNs?

- **Data Preprocessing**
- **Data augmentation**
- **Transfer Learning**
- **Hyperparameter Selection**

What if you don’t have a lot of data? Can you still train CNNs?

Transfer Learning with CNNs

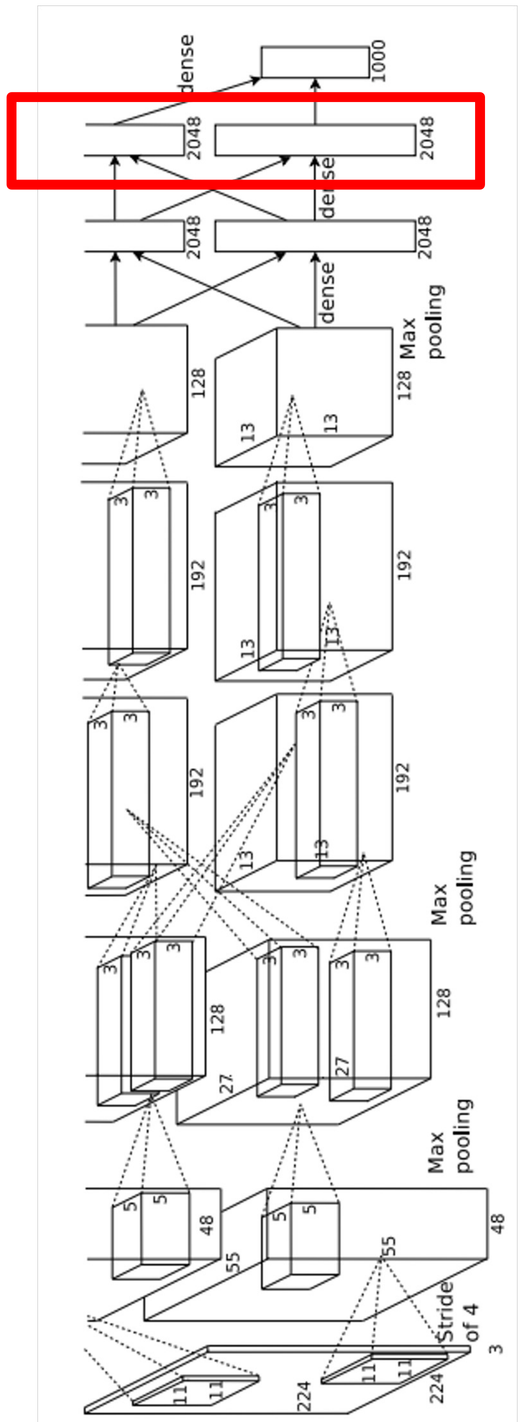




AlexNet:
64 x 3 x 11 x 11

(More on this in Lecture 13)

Transfer Learning with CNNs



Test image   L2 Nearest neighbors in feature space

(More on this in Lecture 13)

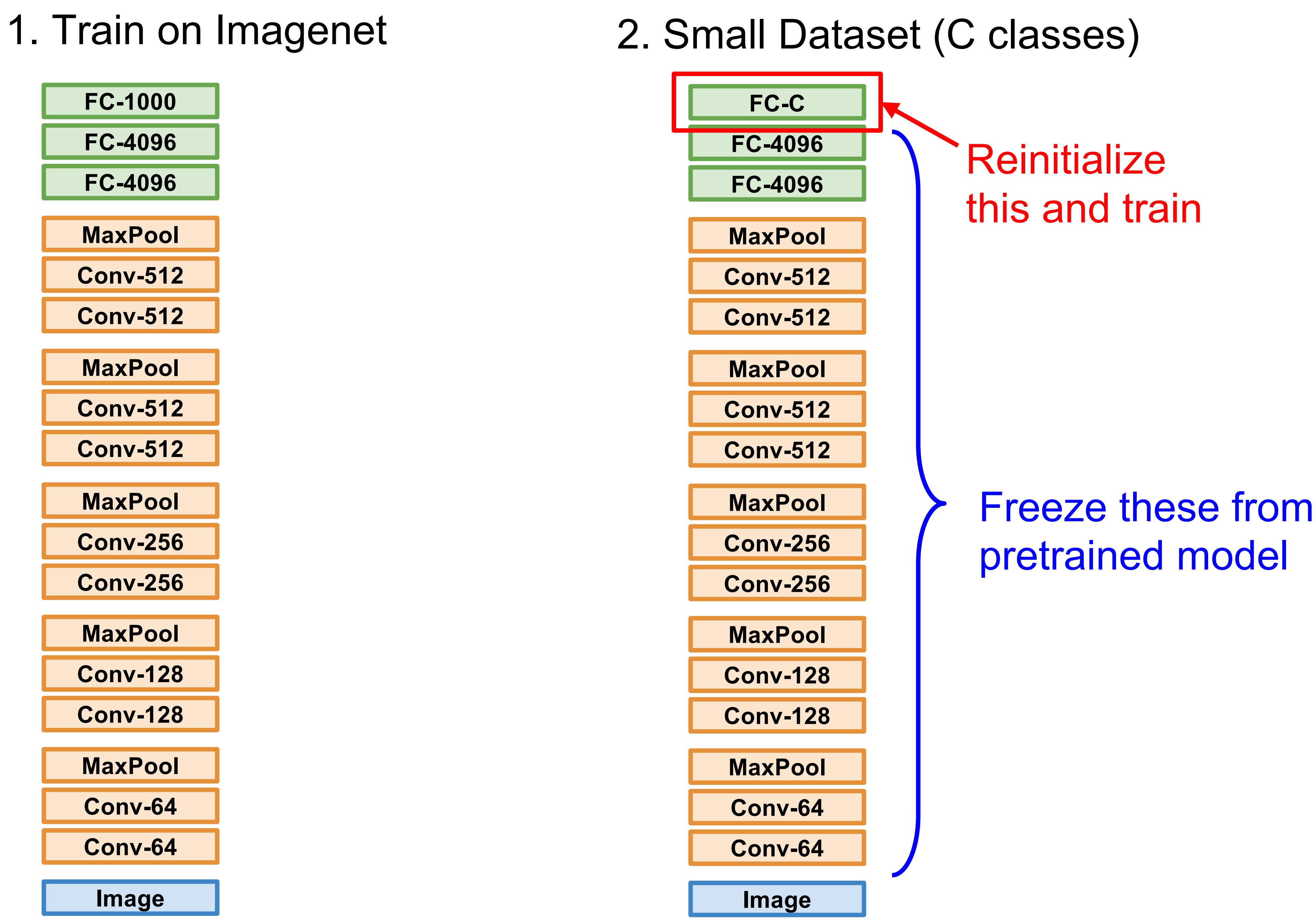
Transfer Learning with CNNs

Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014
Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

1. Train on Imagenet (or internet scale data)

Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014



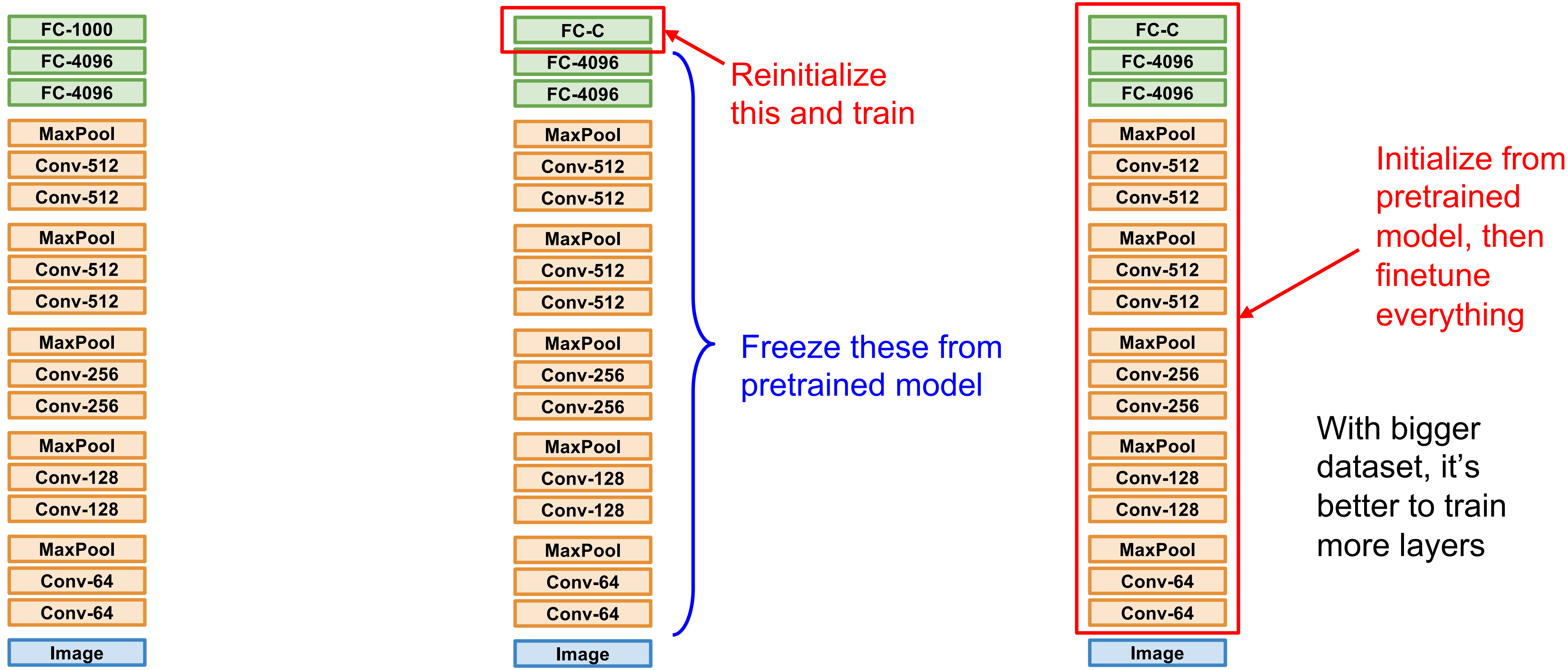
Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014
Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

Transfer Learning with CNNs

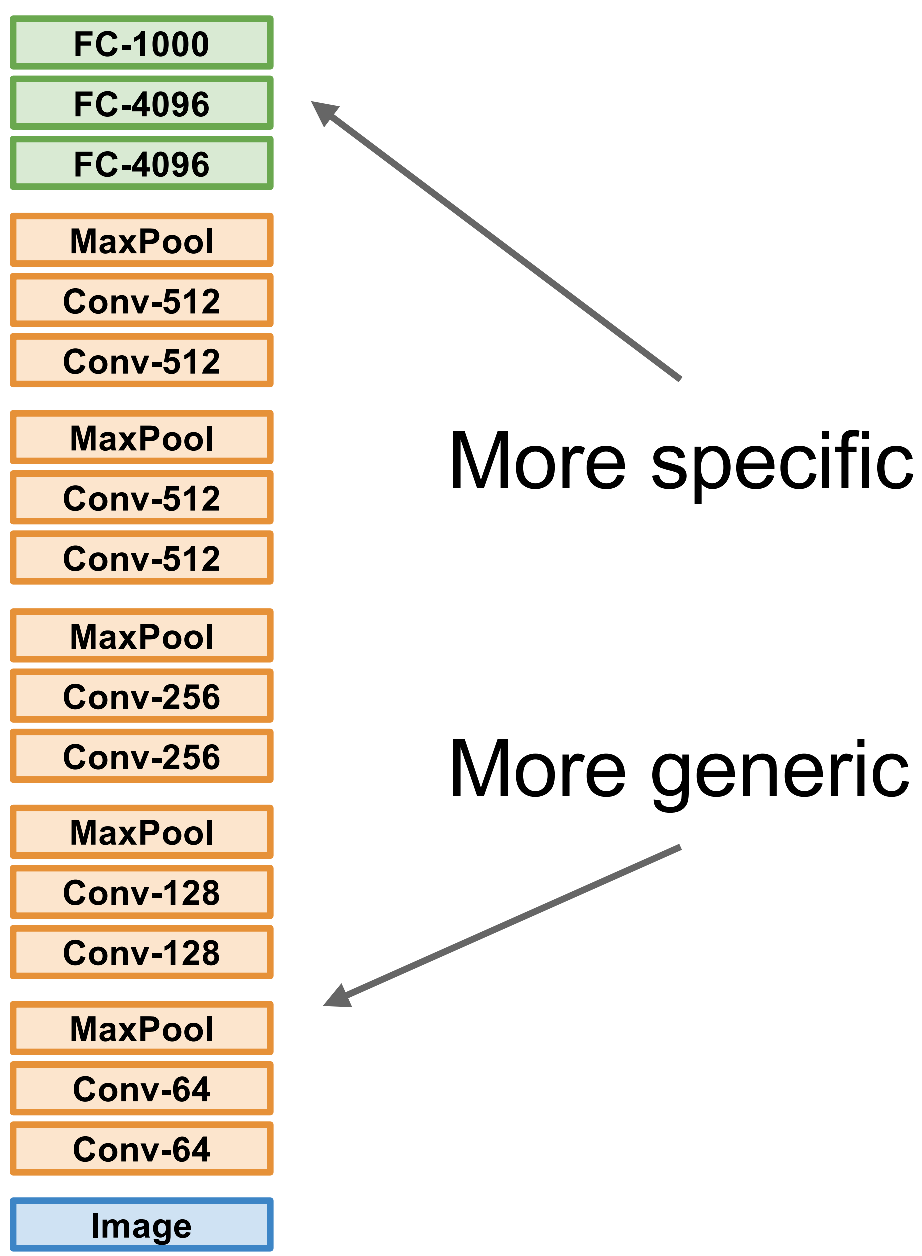
1. Train on Imagenet

2. Small Dataset (C classes)

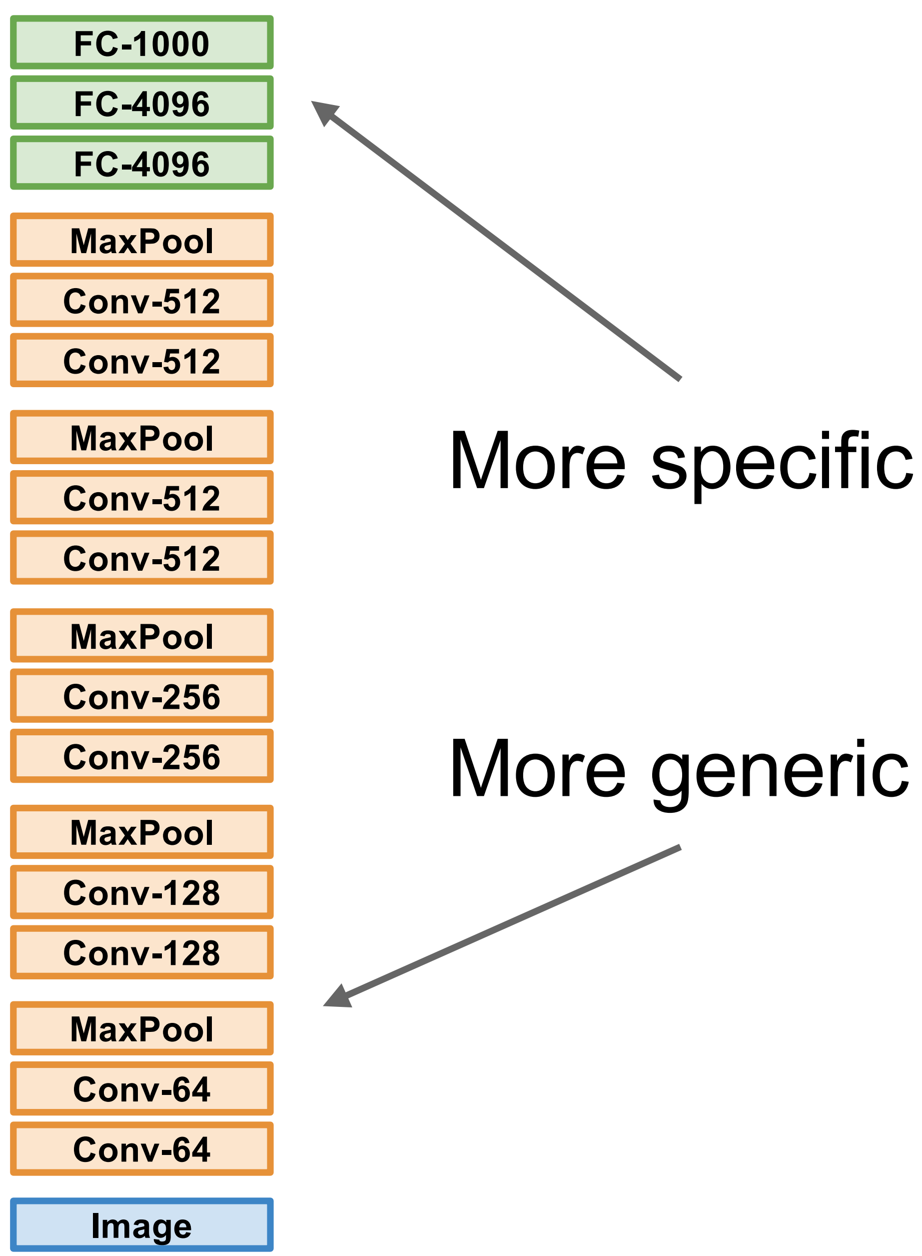
3. Bigger dataset



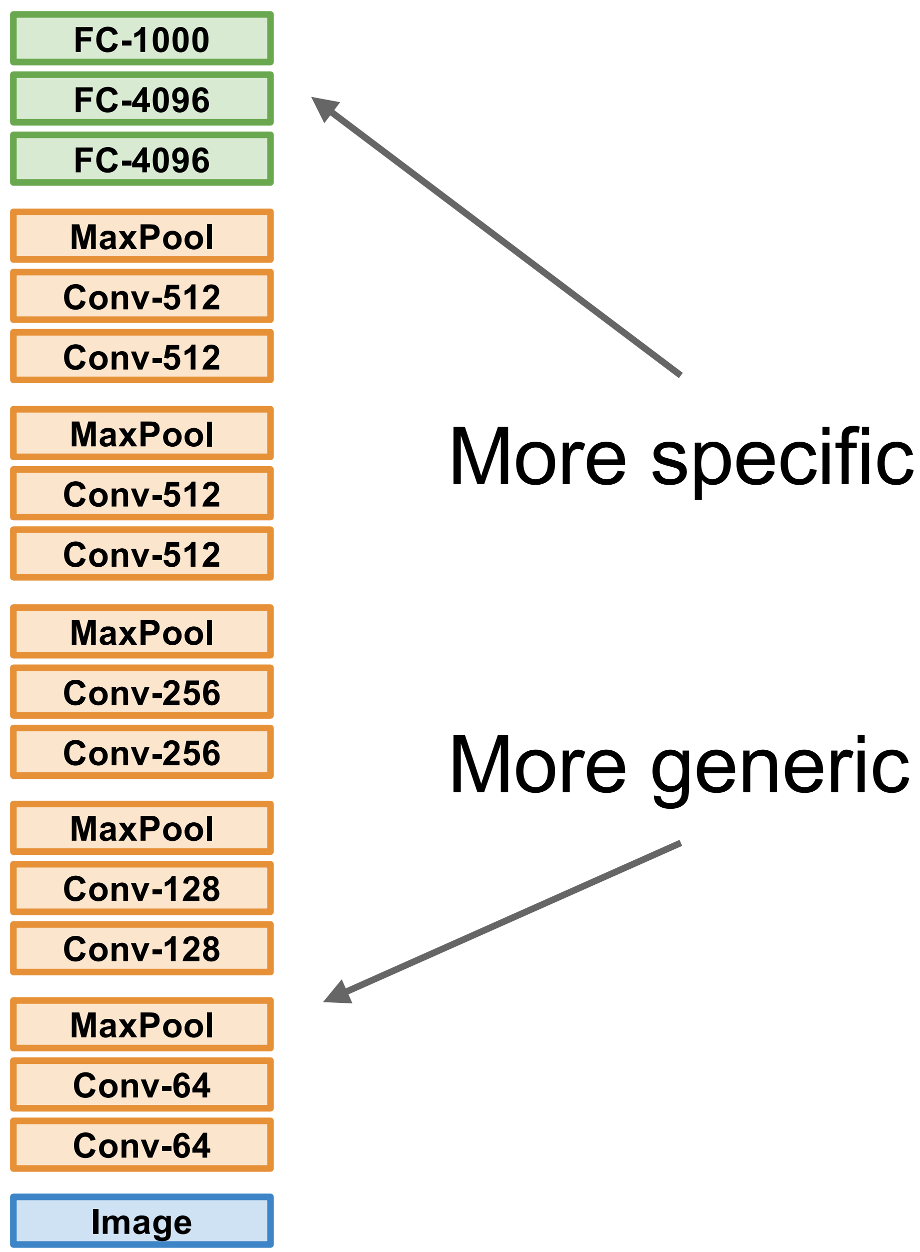
With bigger dataset, it’s better to train more layers



	very similar dataset	**very different dataset**
very little data	?	?
quite a lot of data	?	?



	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on final layer	?
quite a lot of data	Finetune all model layers	?



	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on final layer	Try another model or collect more data ☹
quite a lot of data	Finetune all model layers	Either finetune all model layers or train from scratch!

Takeaway for your projects and beyond:

Have some dataset of interest but it has < ~1M images?

1. Find a very large dataset that has similar data, train a big model there
2. Transfer learn to your dataset

Deep learning frameworks provide a "Model Zoo" of pretrained models so you don't need to train your own

PyTorch: https://github.com/pytorch/vision
Huggingface: https://github.com/huggingface/pytorch-image-models

Lecture Overview – Two Broad Sets of Topics

How to build CNNs?
- **Layers in CNNs**
- **Activation Functions**
- **CNN Architectures**
- **Weight Initialization**

How to train CNNs?
- **Data Preprocessing**
- **Data augmentation**
- **Transfer Learning**
- **Hyperparameter Selection**

Choosing Hyperparameters

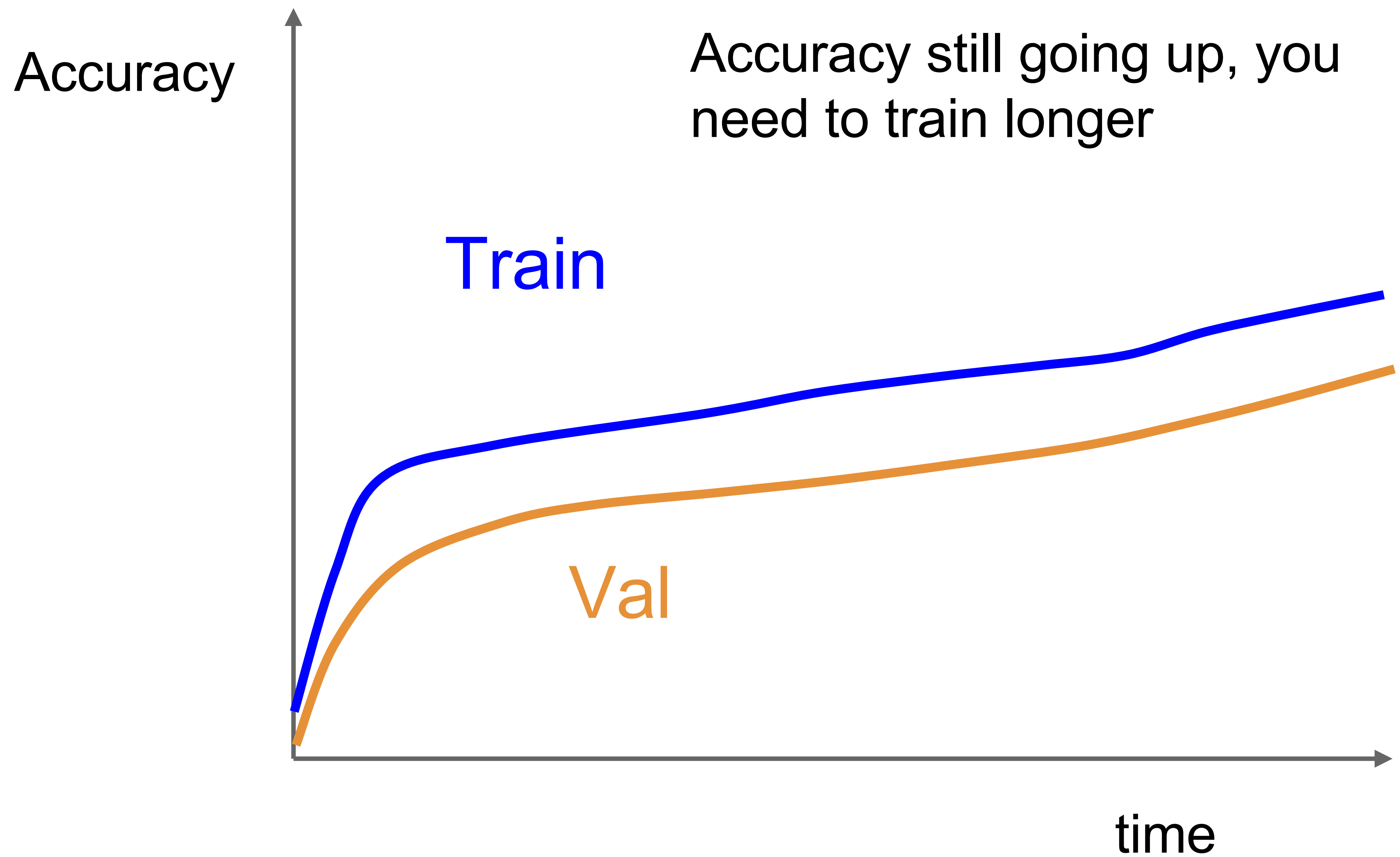
Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down

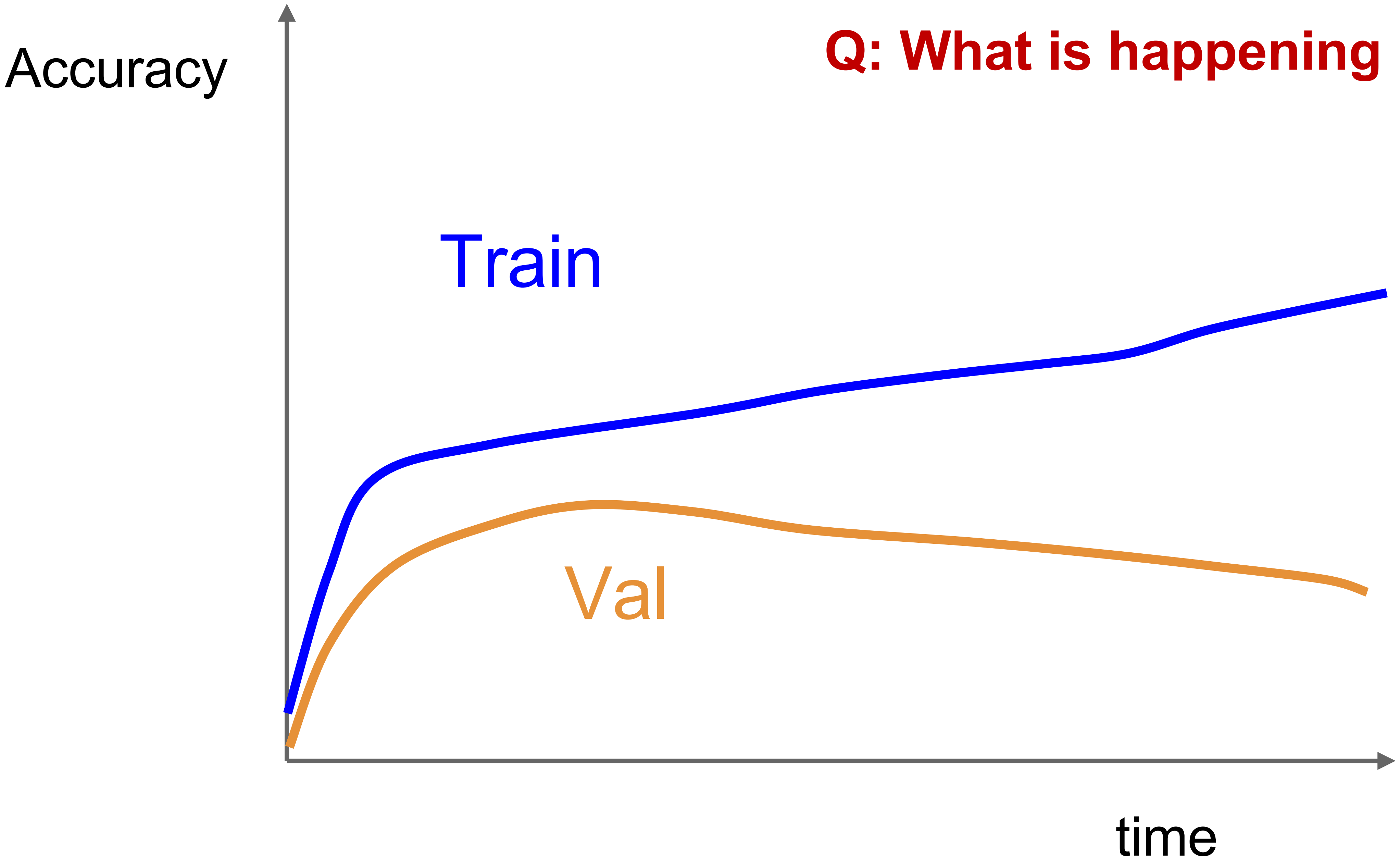
Use the architecture from the previous step, use all training data, turn on small weight decay, find a learning rate that makes the loss drop significantly within ~100 iterations

Good learning rates to try: 1e-1, 1e-2, 1e-3, 1e-4, 1e-5

Choosing Hyperparameters

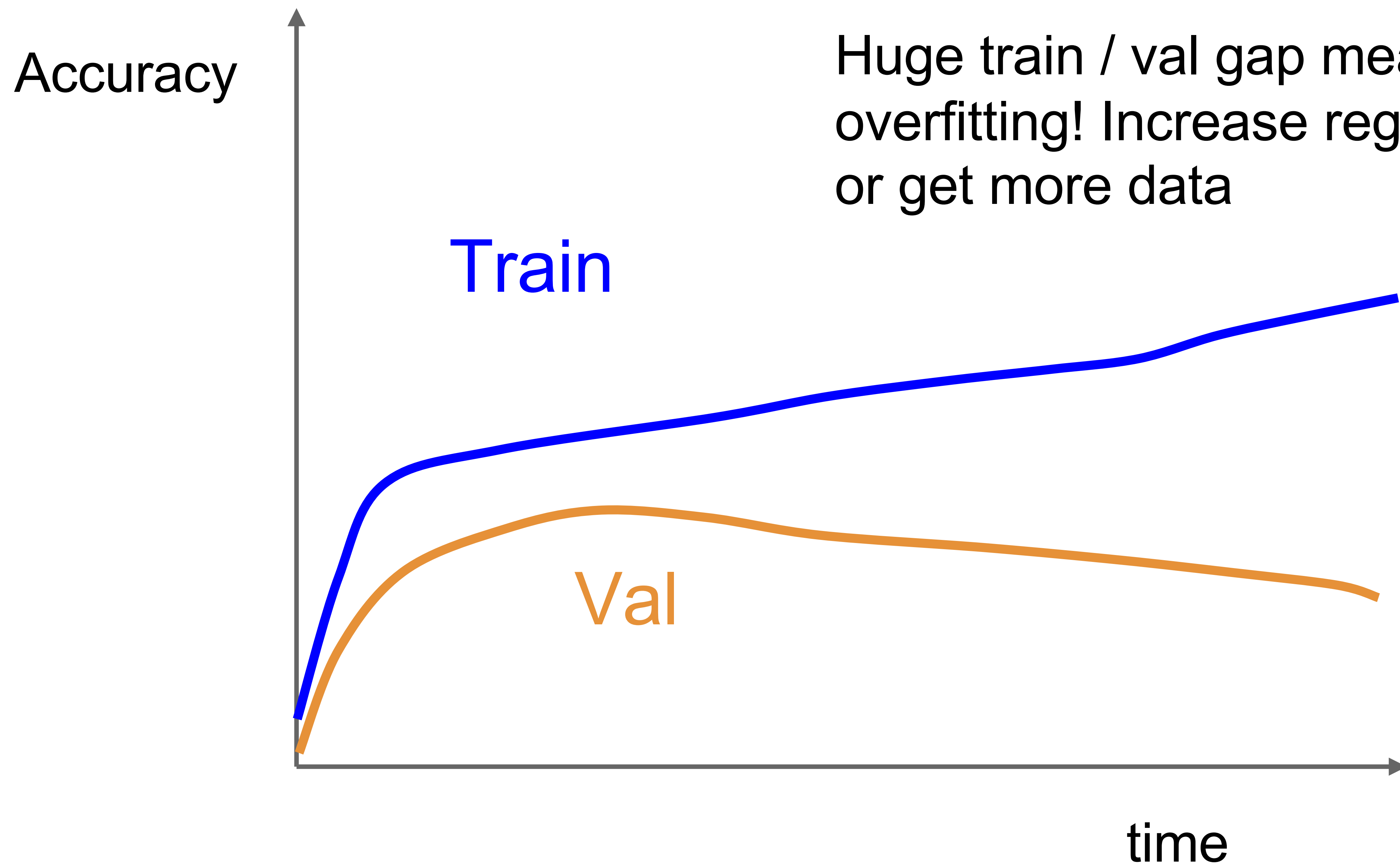
Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down
Step 4: Coarse grid of hyperparams, train for ~1-5 epochs
Step 5: Refine grid, train longer
Step 6: Look at loss and accuracy curves (next slides)

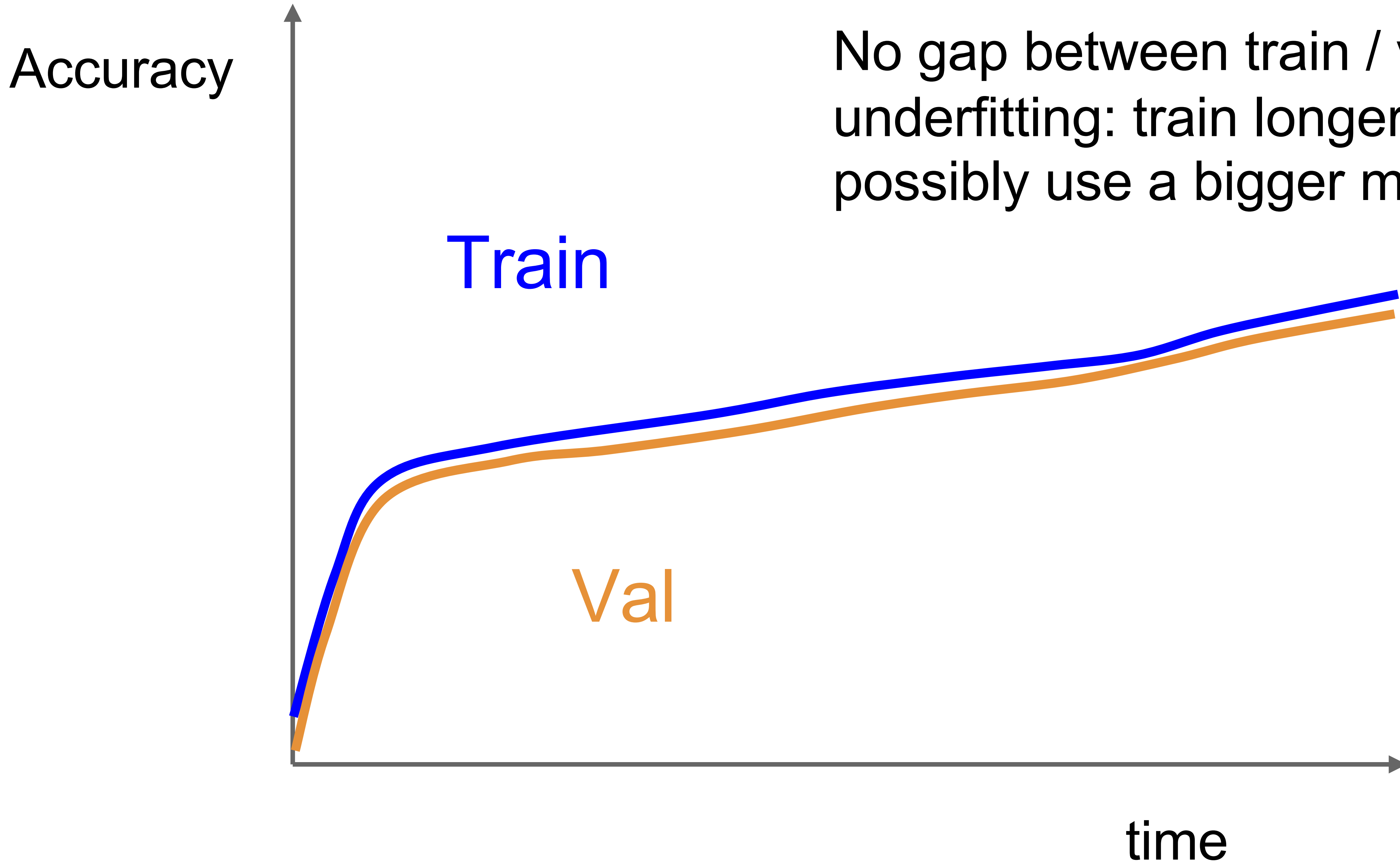

Accuracy
Accuracy still going up, you need to train longer
Train
Val
time



Q: What is happening here?

Huge train / val gap means overfitting! Increase regularization or get more data





No gap between train / val means underfitting: train longer, can possibly use a bigger model

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 and accuracy curves
Step 7: GOTO step 5

Random Search vs. Grid Search

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

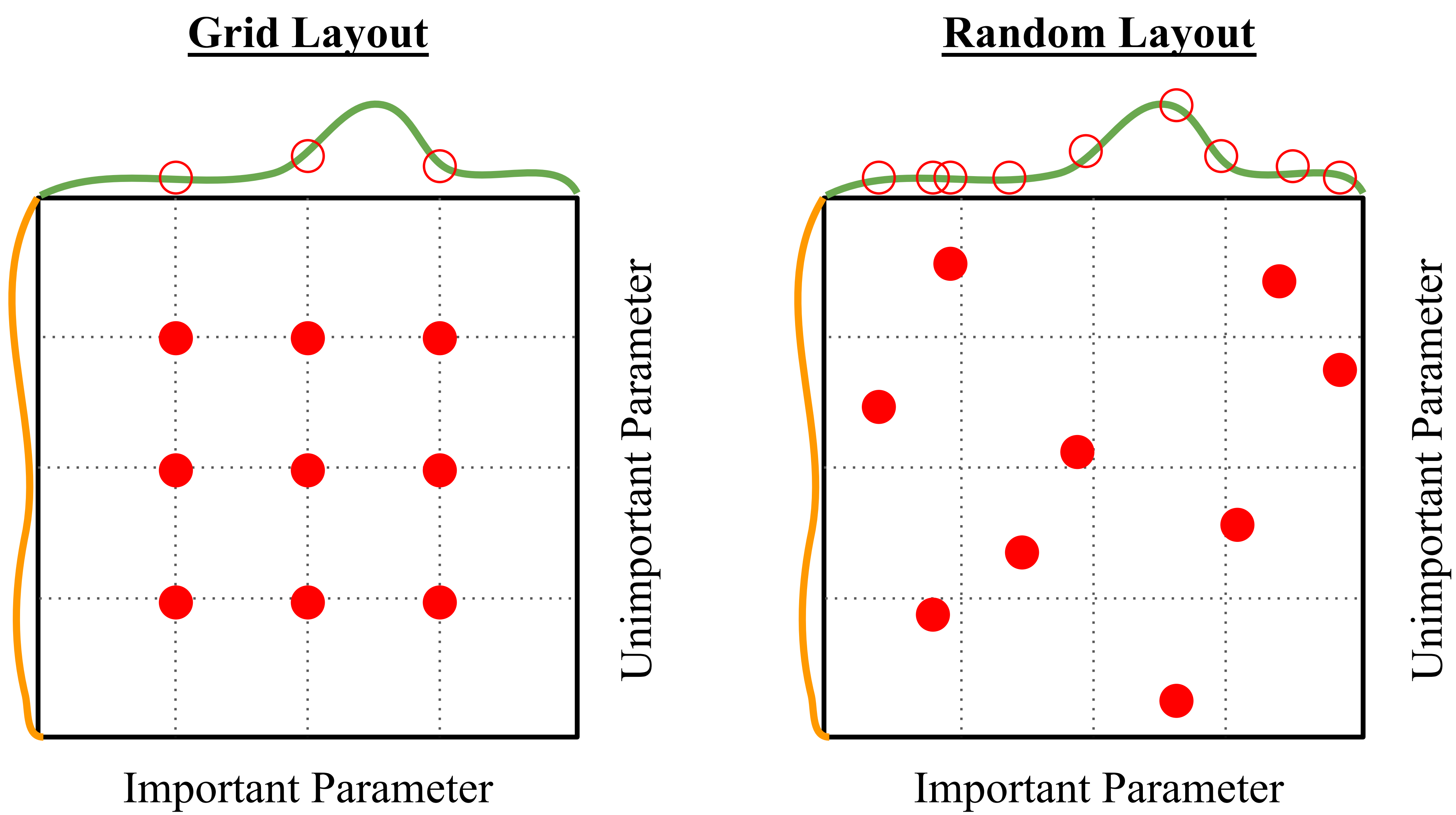


Illustration of Bergstra et al., 2012 by Shayne Longpre, copyright CS231n 2017

Summary

We reviewed 8 topics at a high level:

1. Layers in CNNs (Conv, FC, Norm, Dropout)
2. Activation Functions in NNs (ReLU, GELU, etc.)
3. CNN Architectures (VGG, ResNets)
4. Weight Initialization (Maintain Activation Distribution)

Summary

We reviewed 8 topics at a high level:

5. Data Preprocessing (subtract mean, divide std)
6. Data augmentation (cropping, jitter)
7. Transfer Learning (train on ImageNet first)
8. Hyperparameter (Checking Losses + Random Search)