Training Deep Learning Models

Saeid Nahavandi
Distinguished Professor
snahavandi@swin.edu.au

Parham Kebria
Senior Research Scientist
parhamkebria@ieee.org

Advanced Study Institute: Artificial Intelligence for Disaster Management
Orlando, November 17 – 25, 2025

NATO OTAN
This activity is supported by: The NATO Science for Peace and Security Programme

SWIN BUR NE SWINBURNE UNIVERSITY OF TECHNOLOGY

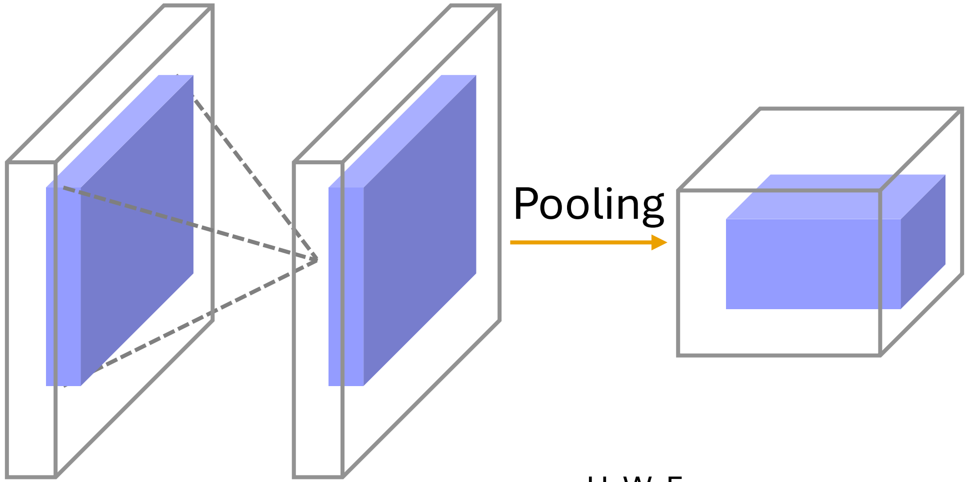
UCF

How to build and train *Deep* CNNs?

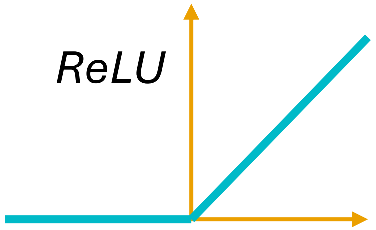
- Normalizations & Dropout
- Activation Functions
- Model Architectures
- Data Preprocessing
- Data Augmentation
- Transfer Learning
- Hyperparameter Tuning
- Code Practice

Training Deep Neural Networks

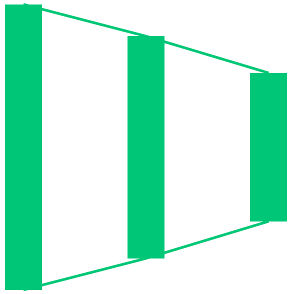
Convolution layers

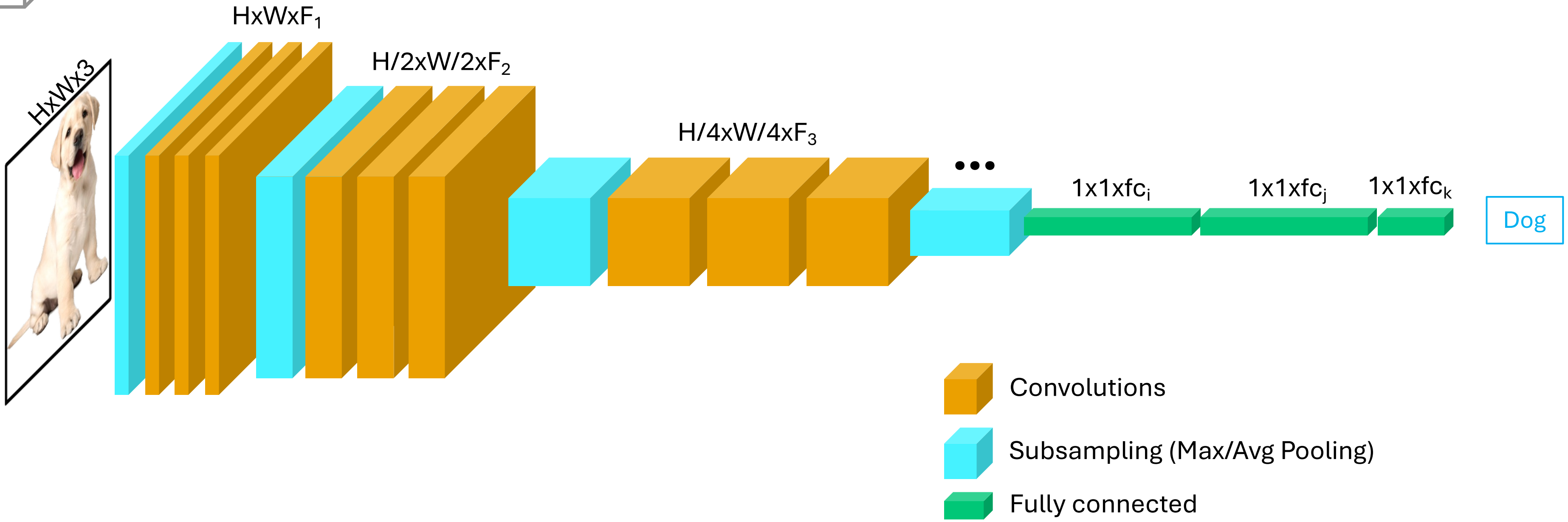


Activation function



Fully-Connected (FC)





Training Deep Neural Networks

- Normalize input tensors/features
- *Scale/shift* by trainable parameters

Normalize per batch

$x: N \times D$

$\mu, \sigma: \ N \times 1$

Trainable parameters

$\beta: \ 1 \times D$

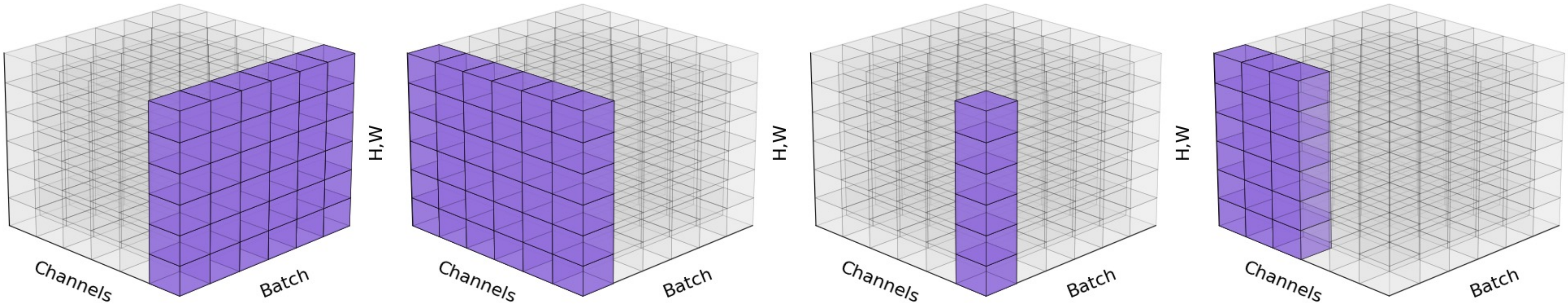
$$y = \frac{\gamma(x-\mu)}{\sigma} + \beta\gamma$$

Batch Normalization

Layer Normalization
Ba, Kiros, & Hinton, arXiv 2016

Instance Normalization

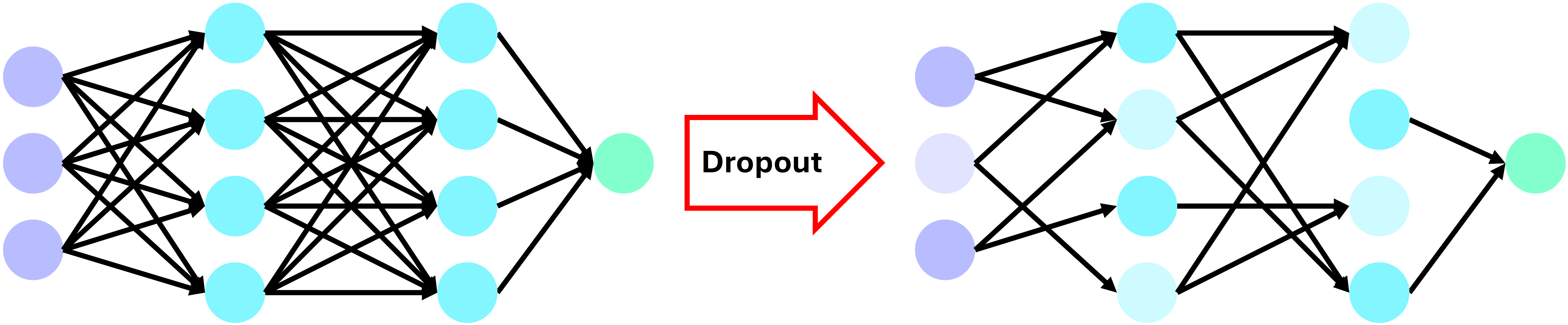
Group Normalization
Wu & He, ECCV 2018



Training Deep Neural Networks

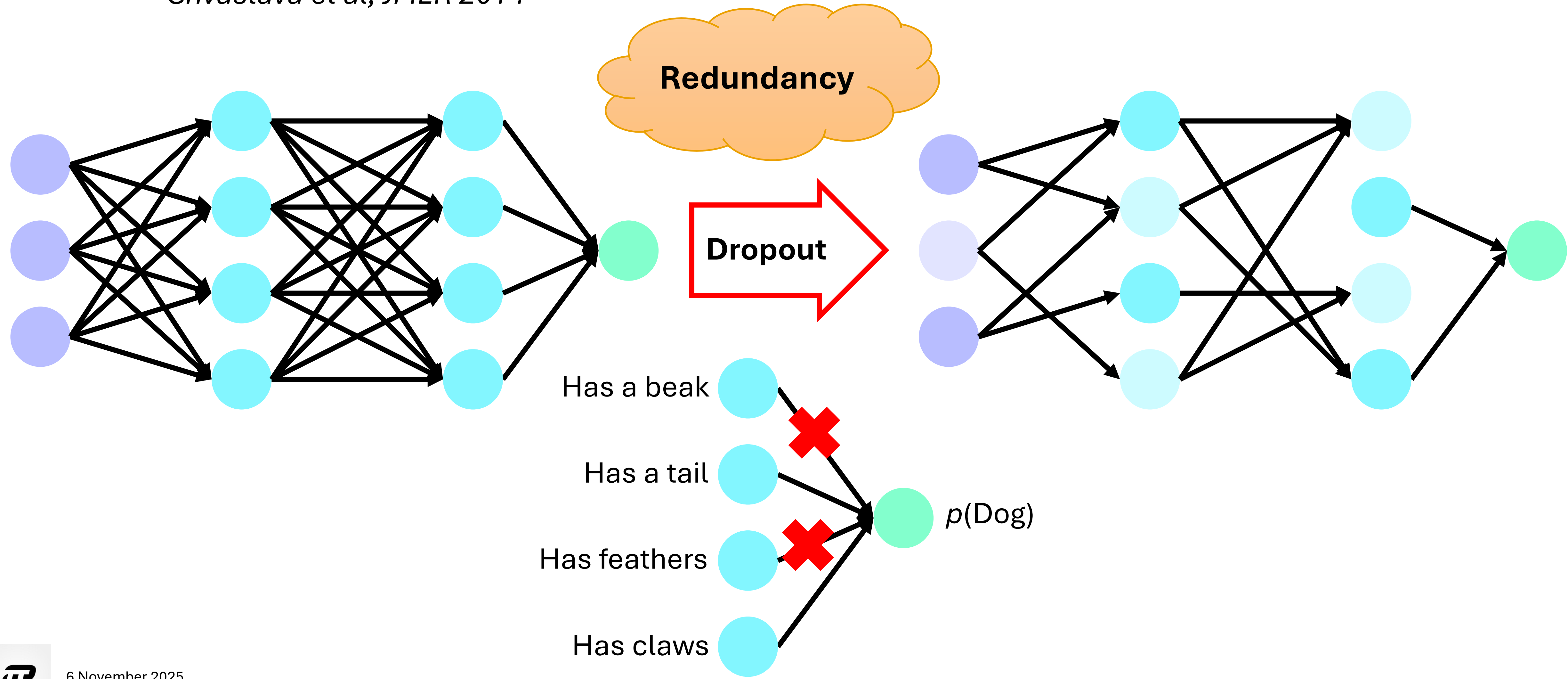
"Dropout: A simple way to prevent neural networks from overfitting",

Srivastava et al, JMLR 2014



Training Deep Neural Networks

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



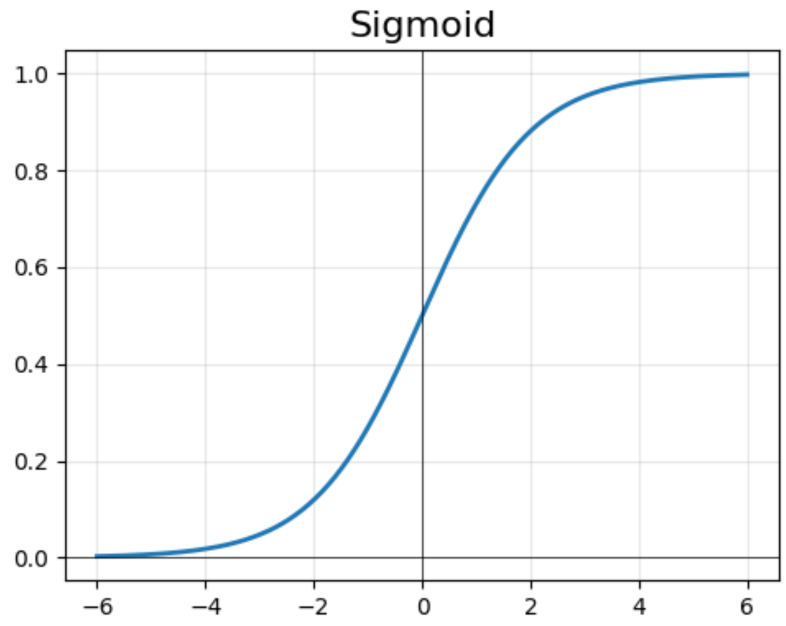
- Dropout: ensemble of several models that share parameters!
- At test (inference) all neurons and connections are active!
  - Each neuron's activation *MUST* be scaled!

```
def classify(x):
    H1 = torch.relu(torch.matmul(W1, x) + b1) * p
    H2 = torch.relu(torch.matmul(W2, H1) + b2) * p
    output = torch.matmul(W3, H2) + b3
    return output
```

Training Deep Neural Networks

- ✓ Squashes to range [0,1]
- ✓ Well-known as neuron firing rate
- x Large values ***kill*** the gradient
- x Too many layers of sigmoid results in ***gradient-vanishing***!

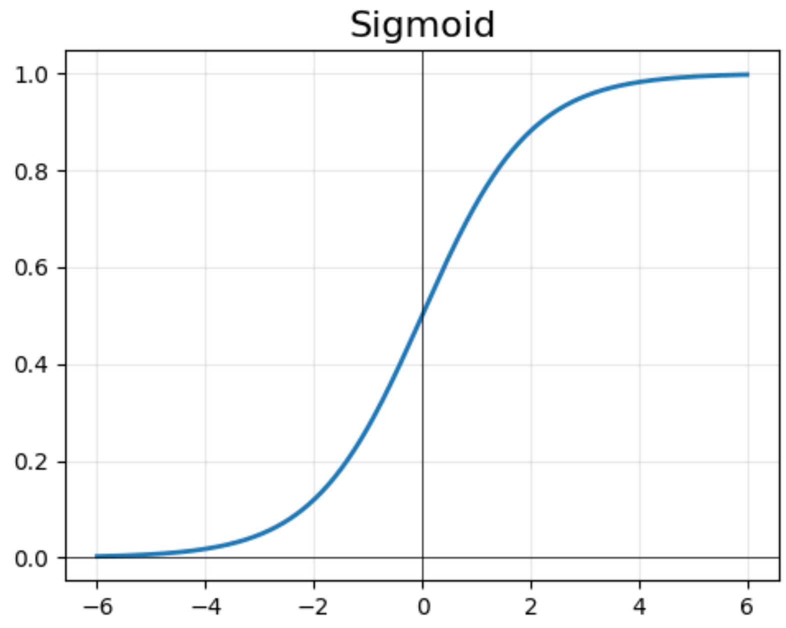
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Training Deep Neural Networks

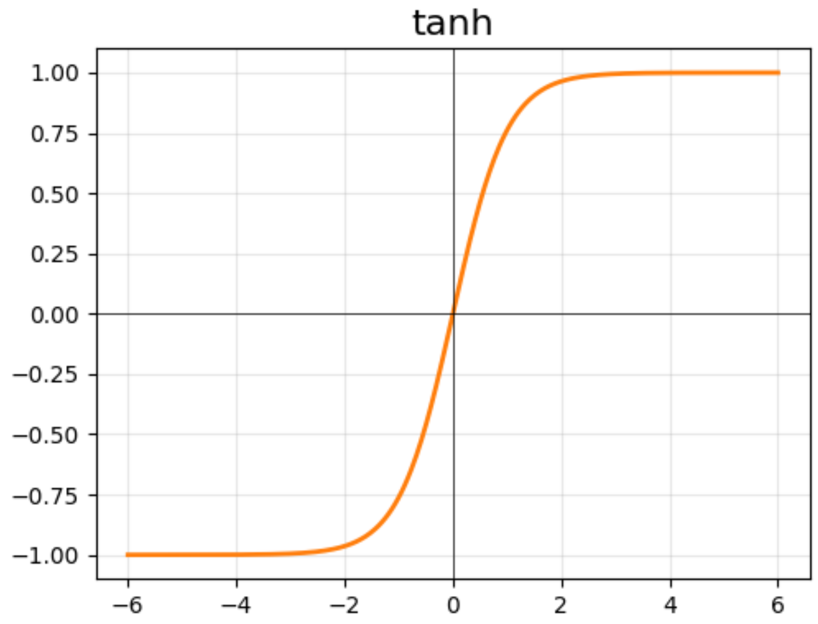
- ✓ Squashes to range [0,1]
- ✓ Well-known as neuron firing rate
- x Large values ***kill*** the gradient
- x Too many layers of sigmoid results in ***gradient-vanishing***!

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



- Between [-1,1] and zero-centered
- Slightly slower gradient-vanishing
- Still saturates like Sigmoid does

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



Training Deep Neural Networks

- ✓ Squashes to range [0,1]
- ✓ Well-known as neuron firing rate
- x Large values ***kill*** the gradient
- x Too many layers of sigmoid results in ***gradient-vanishing***!

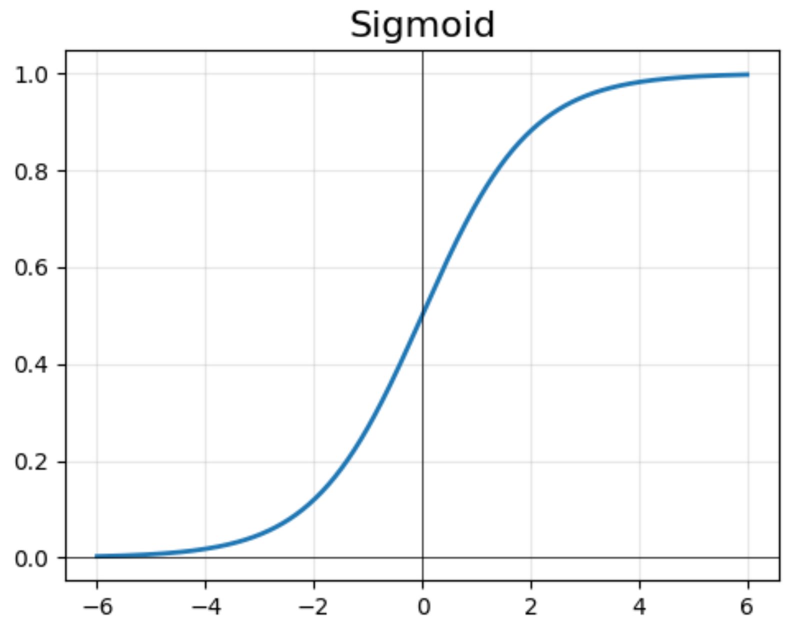
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

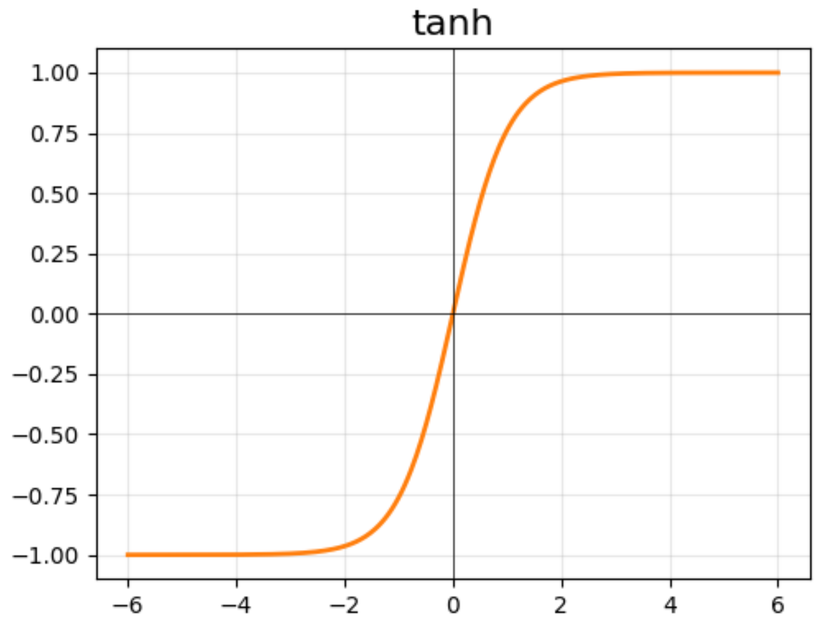
- Between [-1,1] and zero-centered
- Slightly slower gradient-vanishing
- Still saturates like Sigmoid does

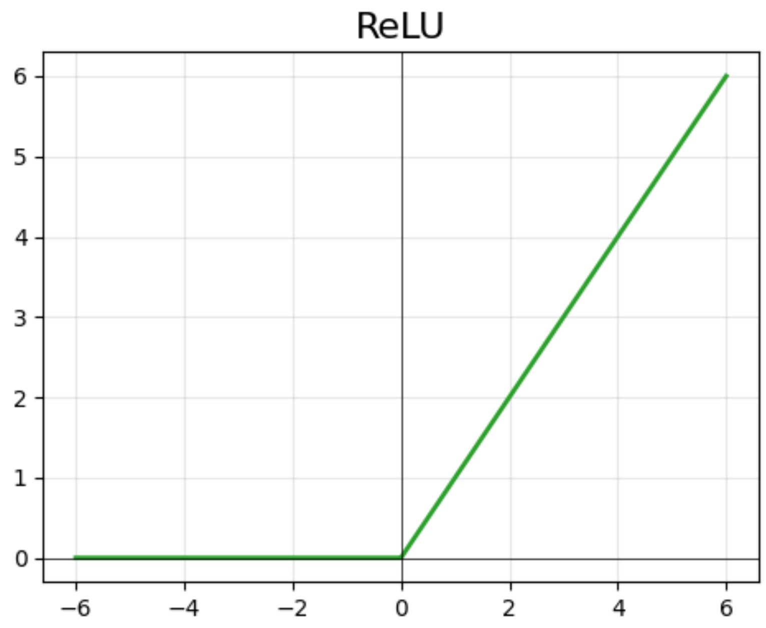
$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- ✓ Does not saturate
- ✓ Computationally efficient
- ✓ Converges much faster (~6x)
- x Not zero-centered
- x Dead zone for negatives

$$\mathrm{ReLU}(x) = \max(0, x)$$

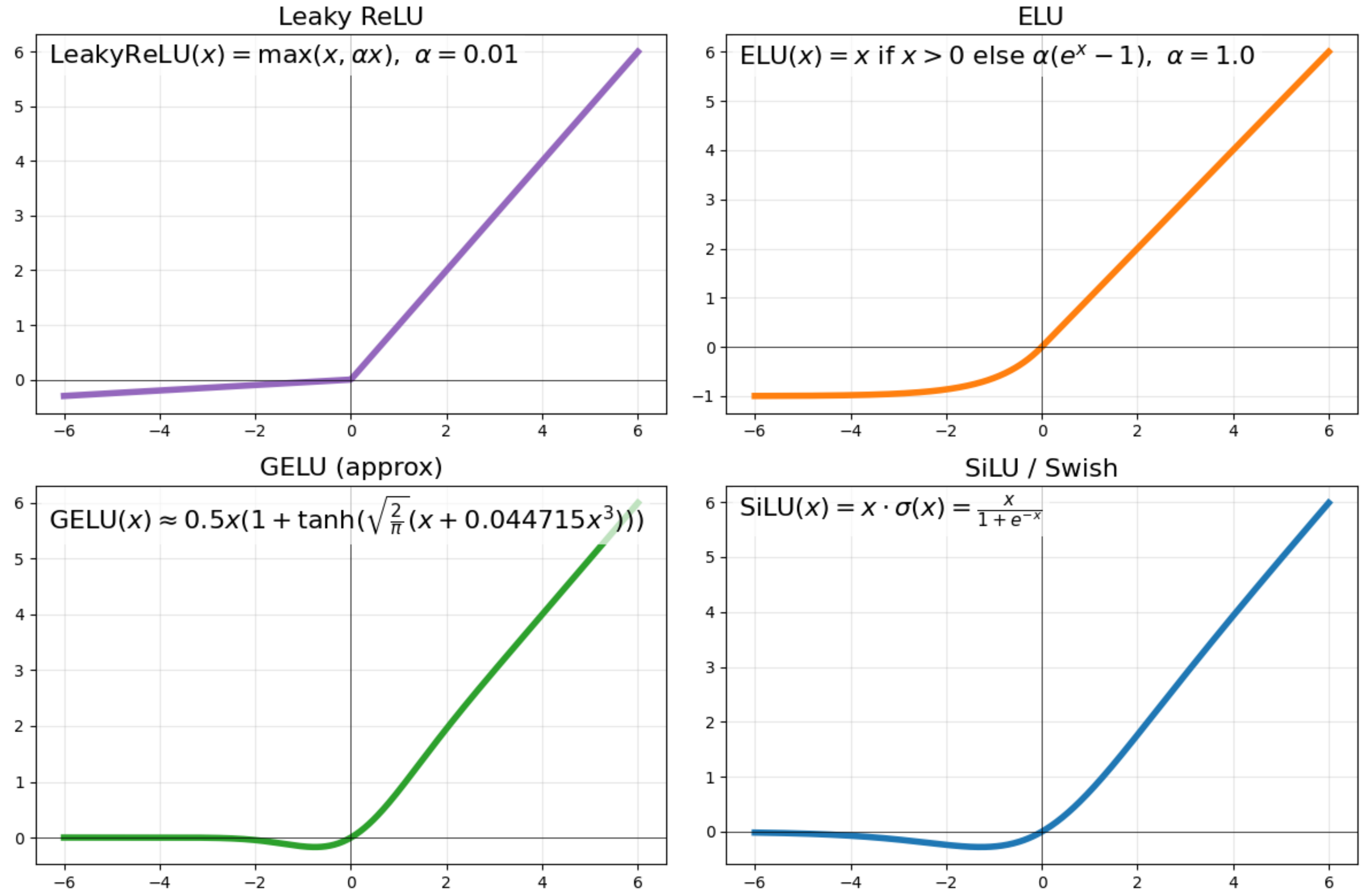






Training Deep Neural Networks

Activation Functions (Non-linearity)



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners (2012-present):
AlexNet, VGG, GoogleNet, ResNet, DenseNet, AllConvNet, etc.

Training Deep Neural Networks

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners (2012-present):
AlexNet, VGG, GoogleNet, ResNet, DenseNet, AllConvNet, etc.

AlexNet Krizhevsky et al. 2012	VGG (11, 13, 16, 19) Simonyan & Zisserman 2014	ResNet (18, 34, 50, 101, 152) He et al. 2015

Training Deep Neural Networks

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners (2012-present):
AlexNet, VGG, GoogleNet, ResNet, DenseNet, AllConvNet, etc.

AlexNet Krizhevsky et al. 2012	VGG (11, 13, 16, 19) Simonyan & Zisserman 2014	ResNet (18, 34, 50, 101, 152) He et al. 2015
227x227x3 **8** Layers: 5 Conv, 3 FC 96x11x11 256x5x5 384x3x3 (2) 256x3x3 4096 4096 1000 ReLU + Softmax Dropout 0.5 ~60M trainable 2 branches, 2 GPUs		

Training Deep Neural Networks

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners (2012-present):
AlexNet, VGG, GoogleNet, ResNet, DenseNet, AllConvNet, etc.

AlexNet Krizhevsky et al. 2012	VGG (11, 13, 16, 19) Simonyan & Zisserman 2014				ResNet (18, 34, 50, 101, 152) He et al. 2015
227x227x3 **8** Layers: 5 Conv, 3 FC 96x11x11 256x5x5 384x3x3 (2) 256x3x3 4096 4096 1000 ReLU + Softmax Dropout 0.5 ~60M trainable 2 branches, 2 GPUs	224x224x3 **11** Layers: 8 Conv, 3 FC 64x3x3 128x3x3 256x3x3 (2) 512x3x3 (2) 512x3x3 (2) 4096 4096 1000 ReLU, Softmax Dropout 0.5 <133M trainable	224x224x3 **13** Layers: 10 Conv, 3 FC 64x3x3 (2) 128x3x3 (2) 256x3x3 (2) 512x3x3 (2) 512x3x3 (2) 4096 4096 1000 ReLU, Softmax Dropout 0.5 >133M trainable	224x224x3 **16** Layers: 13 Conv, 3 FC 64x3x3 (2) 128x3x3 (2) 256x3x3 (3) 512x3x3 (3) 512x3x3 (3) 4096 4096 1000 ReLU, Softmax Dropout 0.5 ~138M trainable	224x224x3 **19** Layers: 16 Conv, 3 FC 64x3x3 (2) 128x3x3 (2) 256x3x3 (4) 512x3x3 (4) 512x3x3 (4) 4096 4096 1000 ReLU, Softmax Dropout 0.5 ~144M trainable	

Training Deep Neural Networks

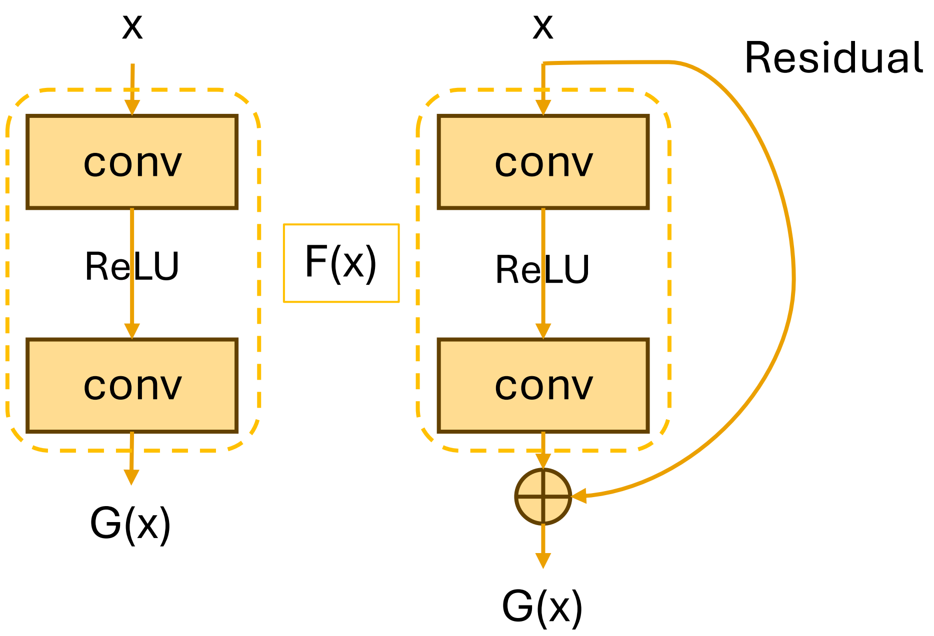
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners (2012-present):
AlexNet, VGG, GoogleNet, ResNet, DenseNet, AllConvNet, etc.

AlexNet Krizhevsky et al. 2012	VGG (11, 13, 16, 19) Simonyan & Zisserman 2014			
227x227x3	224x224x3	224x224x3	224x224x3	224x224x3
8 Layers:	**11** Layers:	**13** Layers:	**16** Layers:	**19** Layers:
5 Conv, 3 FC	8 Conv, 3 FC	10 Conv, 3 FC	13 Conv, 3 FC	16 Conv, 3 FC
96x11x11	64x3x3	64x3x3 (2)	64x3x3 (2)	64x3x3 (2)
256x5x5	128x3x3	128x3x3 (2)	128x3x3 (2)	128x3x3 (2)
384x3x3 (2)	256x3x3 (2)	256x3x3 (2)	256x3x3 (3)	256x3x3 (4)
256x3x3	512x3x3 (2)	512x3x3 (2)	512x3x3 (3)	512x3x3 (4)
4096	512x3x3 (2)	512x3x3 (2)	512x3x3 (3)	512x3x3 (4)
4096	4096	4096	4096	4096
1000	4096	4096	4096	4096
ReLU + Softmax	1000	1000	1000	1000
Dropout 0.5	ReLU, Softmax	ReLU, Softmax	ReLU, Softmax	ReLU, Softmax
~60M trainable	Dropout 0.5	Dropout 0.5	Dropout 0.5	Dropout 0.5
2 branches, 2 GPUs	<133M trainable	>133M trainable	~138M trainable	~144M trainable

ResNet (18, 34, 50, 101, 152)

He et al. 2015

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



Training Deep Neural Networks

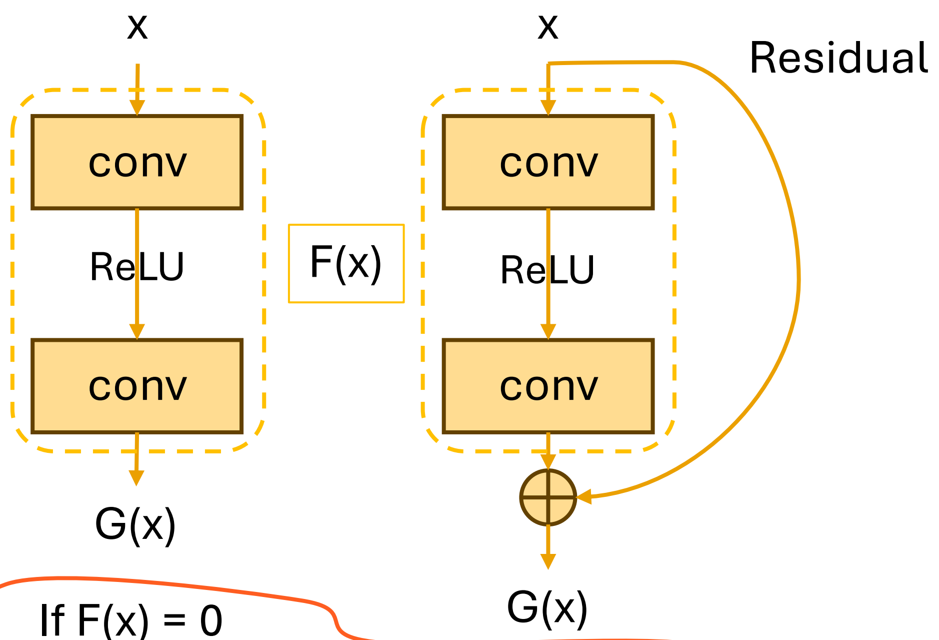
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners (2012-present):
AlexNet, VGG, GoogleNet, ResNet, DenseNet, AllConvNet, etc.

AlexNet Krizhevsky et al. 2012	VGG (11, 13, 16, 19) Simonyan & Zisserman 2014			
227x227x3	224x224x3	224x224x3	224x224x3	224x224x3
8 Layers:	**11** Layers:	**13** Layers:	**16** Layers:	**19** Layers:
5 Conv, 3 FC	8 Conv, 3 FC	10 Conv, 3 FC	13 Conv, 3 FC	16 Conv, 3 FC
96x11x11	64x3x3	64x3x3 (2)	64x3x3 (2)	64x3x3 (2)
256x5x5	128x3x3	128x3x3 (2)	128x3x3 (2)	128x3x3 (2)
384x3x3 (2)	256x3x3 (2)	256x3x3 (2)	256x3x3 (3)	256x3x3 (4)
256x3x3	512x3x3 (2)	512x3x3 (2)	512x3x3 (3)	512x3x3 (4)
4096	512x3x3 (2)	512x3x3 (2)	512x3x3 (3)	512x3x3 (4)
4096	4096	4096	4096	4096
1000	4096	4096	4096	4096
ReLU + Softmax	1000	1000	1000	1000
Dropout 0.5	ReLU, Softmax	ReLU, Softmax	ReLU, Softmax	ReLU, Softmax
~60M trainable	Dropout 0.5	Dropout 0.5	Dropout 0.5	Dropout 0.5
2 branches, 2 GPUs	<133M trainable	>133M trainable	~138M trainable	~144M trainable

ResNet (18, 34, 50, 101, 152)

He et al. 2015

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



If F(x) = 0
→ G(x) = 0

G(x) = F(x) + x

Image (input) Normalization: Center and scale for each channel

- Subtract per-channel mean and divide by per-channel std
- Requires pre-calculation of mean and std for each pixel channel

```
import numpy as np
for pixel in images: # dataset
    for i in range(pixel.shape[0]):
        for j in range(pixel.shape[1]):
            for c in range(pixel.shape[2]):
                norm_pixel[i,j,c] = (pixel[i,j,c] - np.mean(pixel[:,:,c])) / np.std(pixel[:,:,c])
```

Image (input) Normalization: Center and scale for each channel

- Subtract per-channel mean and divide by per-channel std
- Requires pre-calculation of mean and std for each pixel channel

```python
import numpy as np
for pixel in images: # dataset
    for i in range(pixel.shape[0]):
        for j in range(pixel.shape[1]):
            for c in range(pixel.shape[2]):
                norm_pixel[i,j,c] = (pixel[i,j,c] - np.mean(pixel[:,:,c])) / np.std(pixel[:,:,c])
```

Image (input) Resizing/Cropping:

- Deep learning models expect fixed input sizes: e.g. 224x224

```python
import torch
from torchvision import transforms
transforms.Resize((224, 224))
```

Regularization: A common pattern

- While training: add some randomness $y = fw(x, z)$
- At test: average out randomness $y = f(x) = E_z[f(x, z)] = \int p(x) f(x, z) dz$

Dropout: randomly drop activations

Regularization: A common pattern

- While training: add some randomness $y = fw(x, z)$
- At test: average out randomness $y = f(x) = E_z[f(x, z)] = \int p(x) f(x, z) dz$

Regularization: A common pattern

- While training: add some randomness $y = fw(x, z)$
- At test: average out randomness $y = f(x) = E_z[f(x, z)] = \int p(x) f(x, z) dz$

Dropout: randomly drop activations

Use all activations and average with *p*

Dropout: randomly drop activations

Regularization: A common pattern

- While training: add some randomness $y = fw(x, z)$
- At test: average out randomness $y = f(x) = E_z[f(x, z)] = \int p(x) f(x, z) dz$

Use all activations and average with *p*

Augmentation:

Training Deep Neural Networks

Dropout: randomly drop activations

Regularization: A common pattern

- While training: add some randomness $y = fw(x,z)$
- At test: average out randomness $y = f(x) = E_z[f(x,z)] = \int p(x) f(x,z) dz$

Use all activations and average with *p*

Augmentation:

- Random contract and brightness

Training Deep Neural Networks

Dropout: randomly drop activations

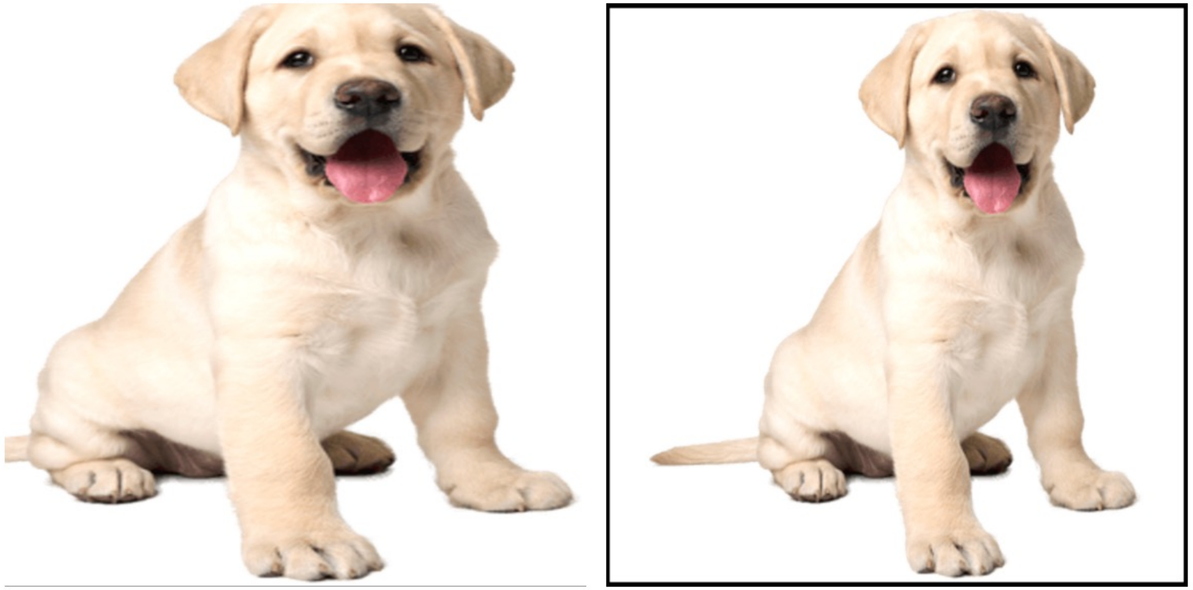
Regularization: A common pattern

- While training: add some randomness $y = fw(x, z)$
- At test: average out randomness $y = f(x) = E_z[f(x,z)] = \int p(x) f(x,z) dz$

Use all activations and average with *p*

Augmentation:

- Random contract and brightness
- Random cropping and scaling



Dropout: randomly drop activations

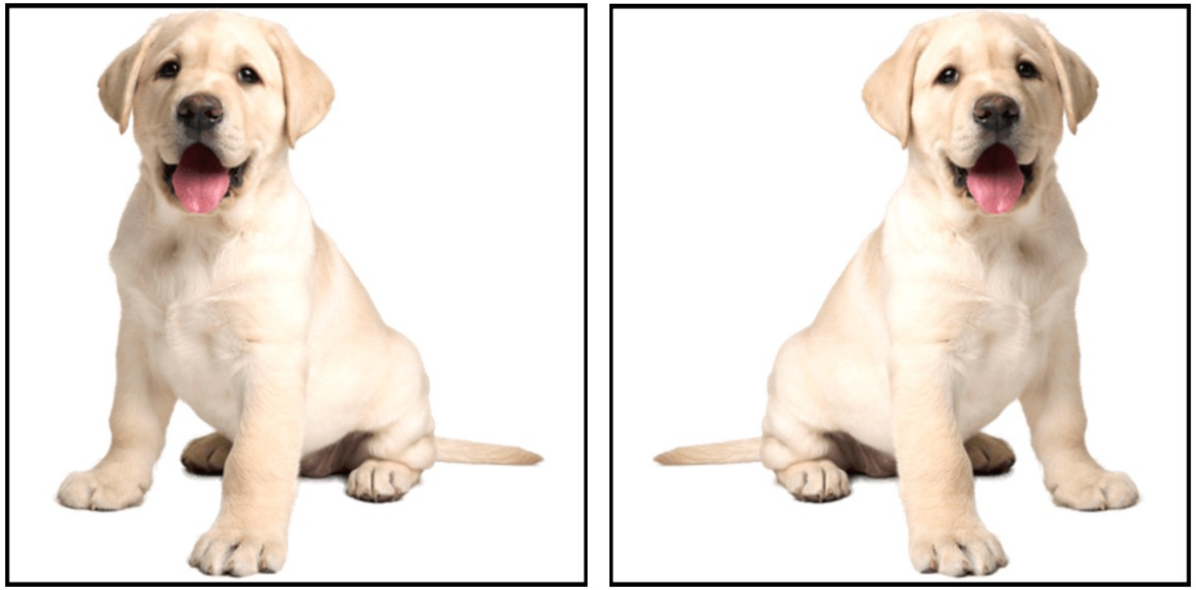
Regularization: A common pattern

- While training: add some randomness $y = fw(x,z)$
- At test: average out randomness $y = f(x) = E_z[f(x,z)] = \int p(x)f(x,z)dz$

Use all activations and average with *p*

Augmentation:

- Random contract and brightness
- Random cropping and scaling
- Random flipping



Regularization: A common pattern

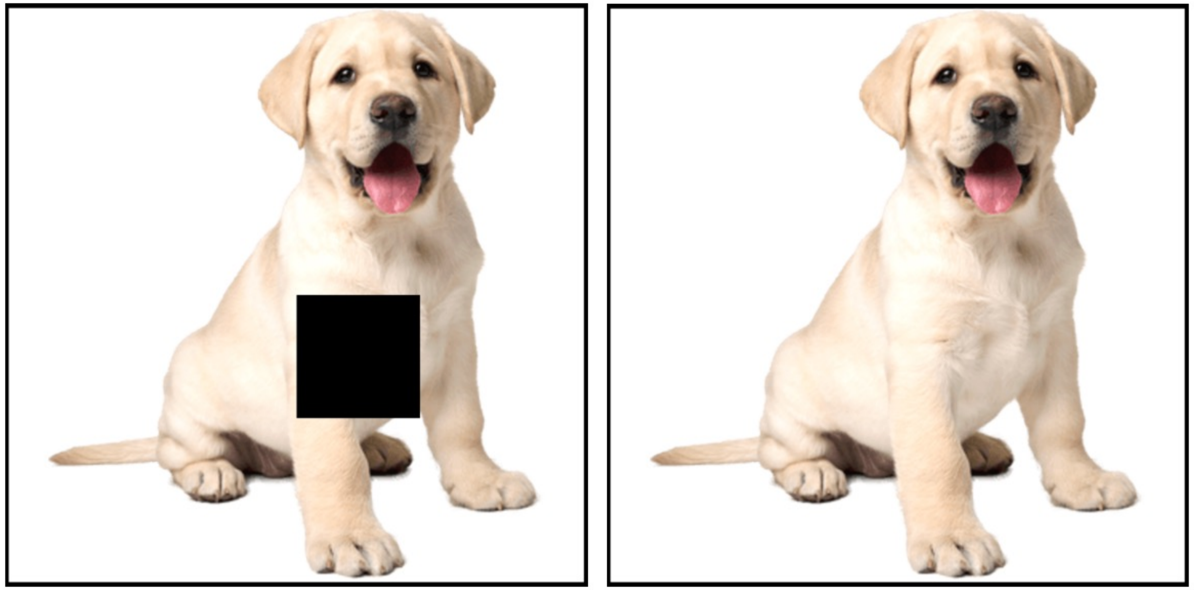
- While training: add some randomness $y = fw(x, z)$
- At test: average out randomness $y = f(x) = E_z[f(x,z)] = \int p(x) f(x,z) dz$

Dropout: randomly drop activations

Use all activations and average with *p*

Augmentation:

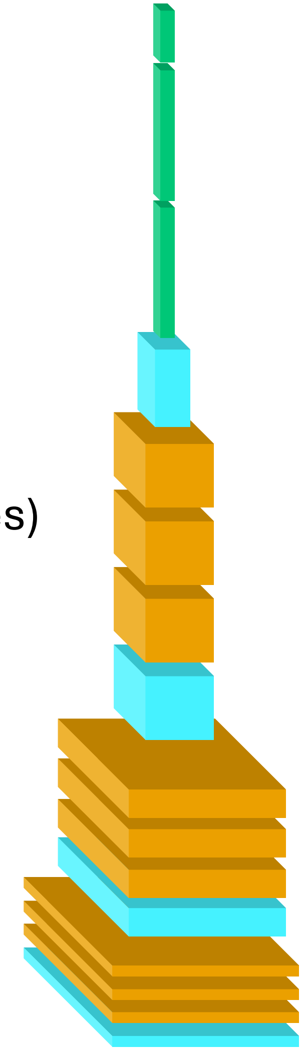
- Random contract and brightness
- Random cropping and scaling
- Random flipping
- Random cutouts (small datasets)



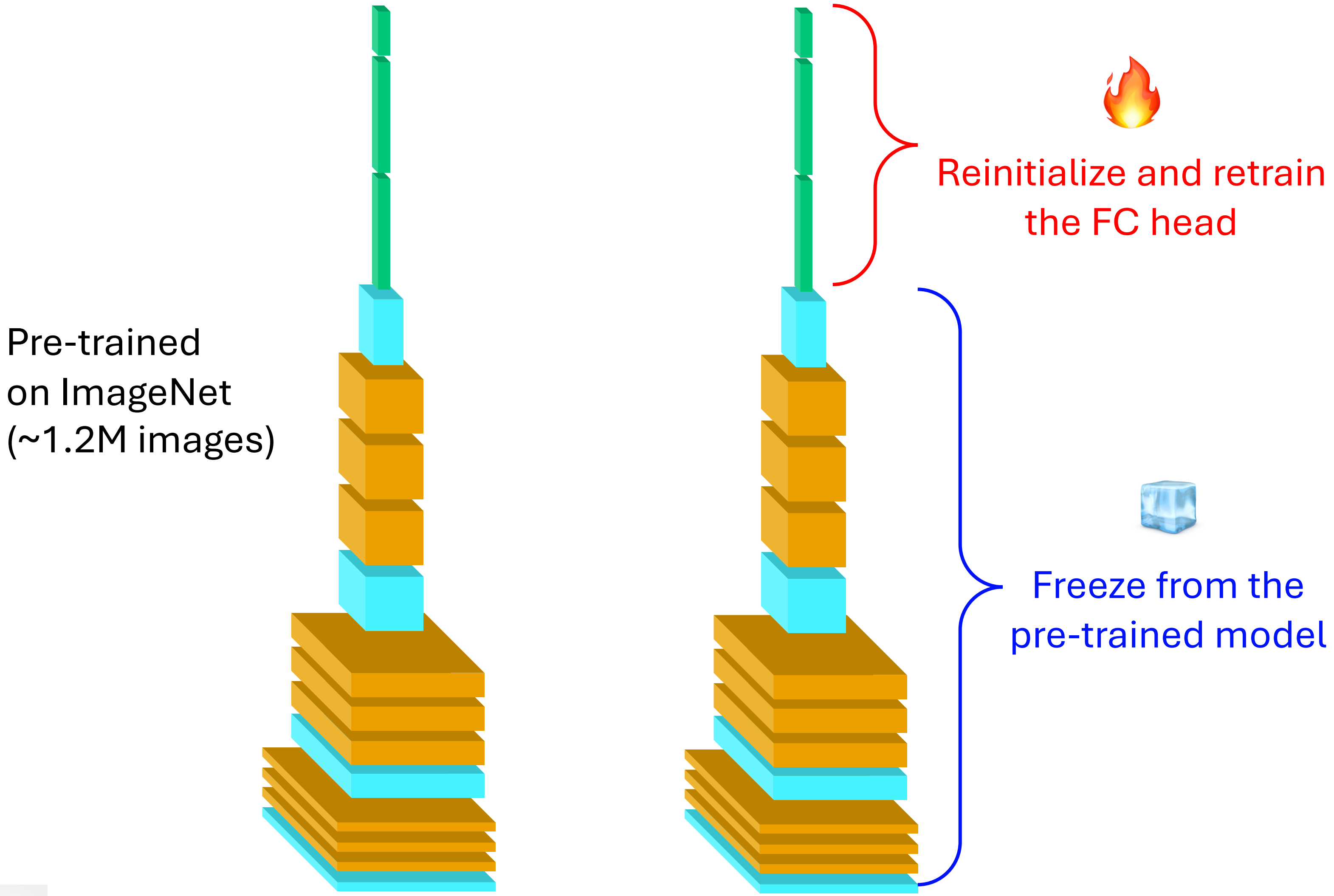
Training Deep Neural Networks

Training a ***deep*** network on a ***smaller*** dataset!

Pre-trained
on ImageNet
(~1.2M images)



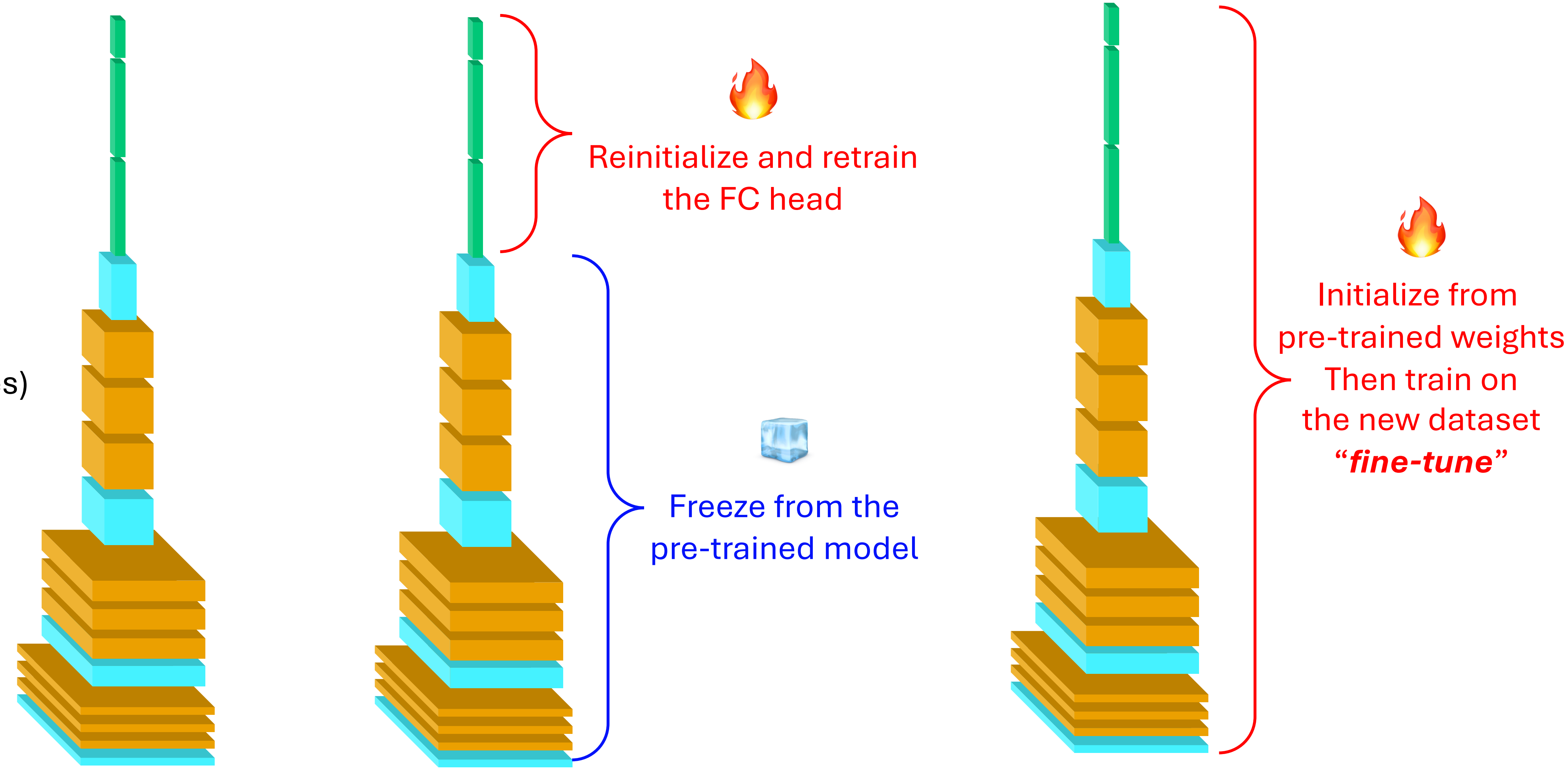
Training a <u>*deep*</u> network on a <u>*smaller*</u> dataset!



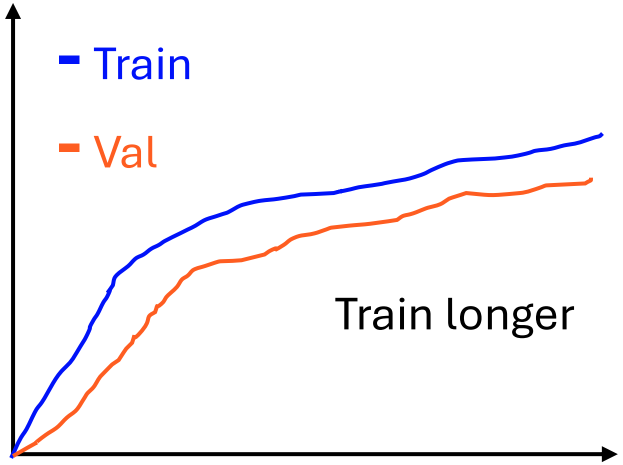
Training Deep Neural Networks

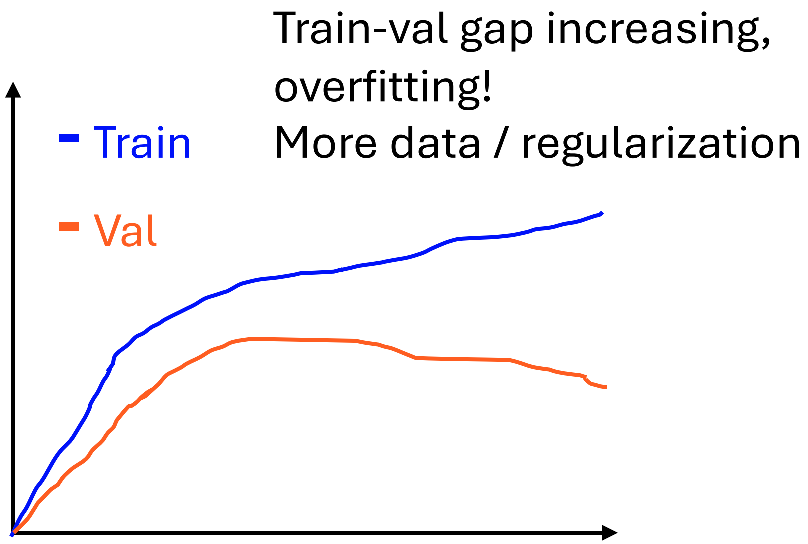
Training a *deep* network on a *smaller* dataset! But if a *large* dataset ...

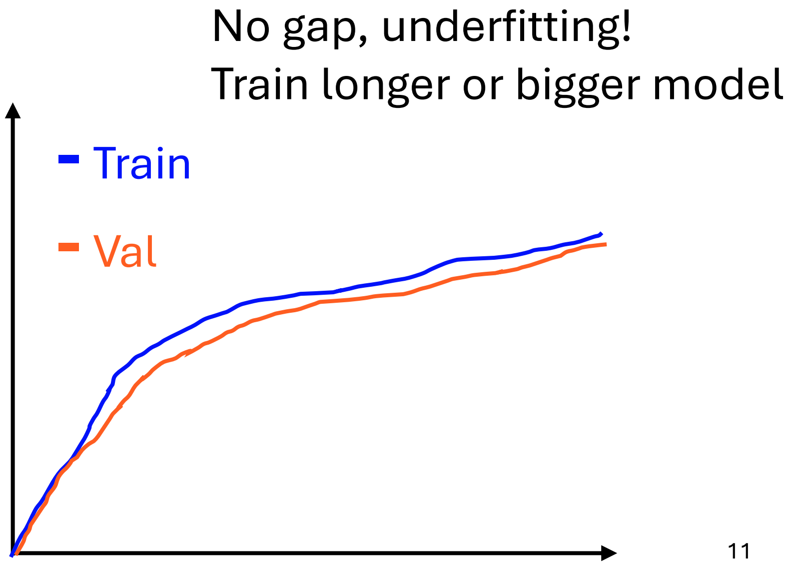
Pre-trained
on ImageNet
(~1.2M images)



1. Check initial loss
2. Overfit a small sample
3. *Find a learning-rate that makes loss decrease significantly*
    1. Common rates: 1e-1, … , 1e-5
4. Try a broad set of hyperparameters, and train for 1~5 epochs
5. Refine your selection set and train longer
6. Always keep an eye on loss and accuracy curves→Step 5.







Training Deep Neural Networks

(Q&A)