INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

IIIT Hyderabad

Deep Learning-III

Robotics Research Center

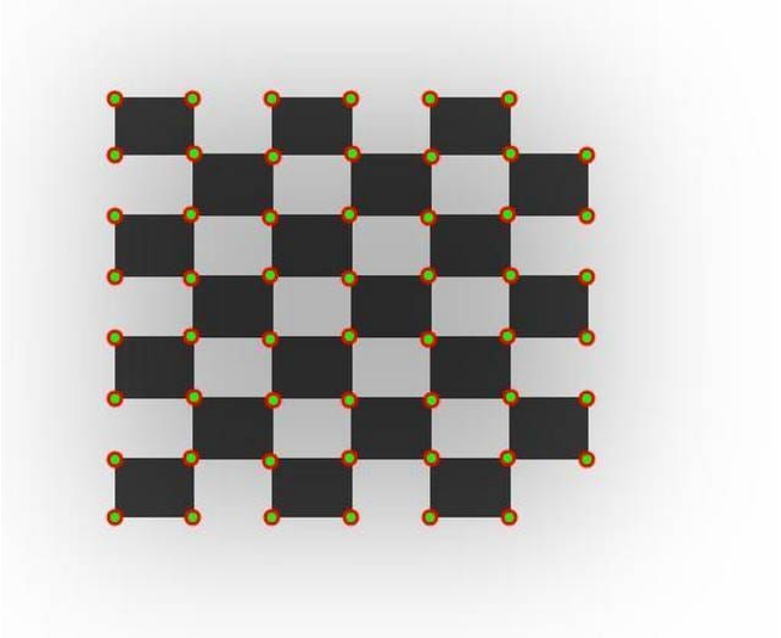
Summer School'25

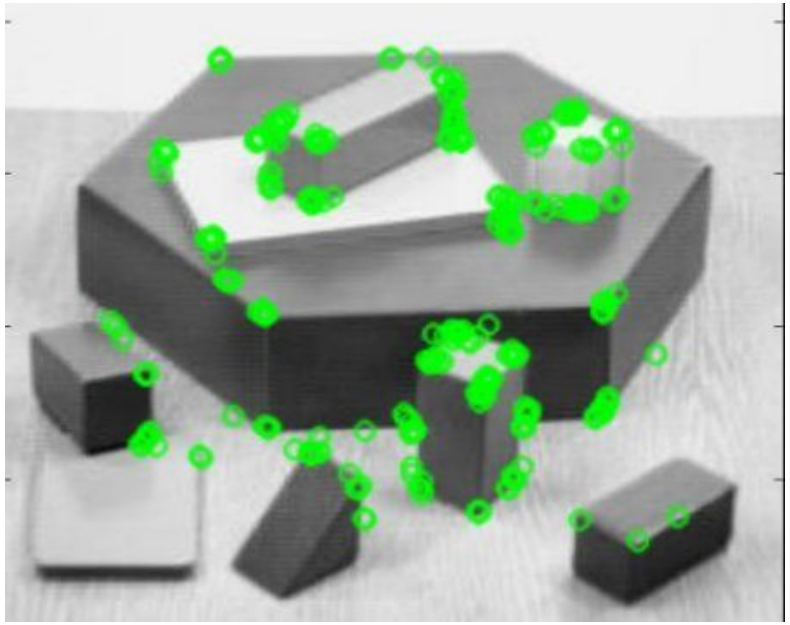
Image Recognition

Basics

1. Region selection (keypoints)

Distinct, repeatable locations in an image: such as corners, blobs, or edges, where local image structure exhibits high variation in multiple directions.





Basics

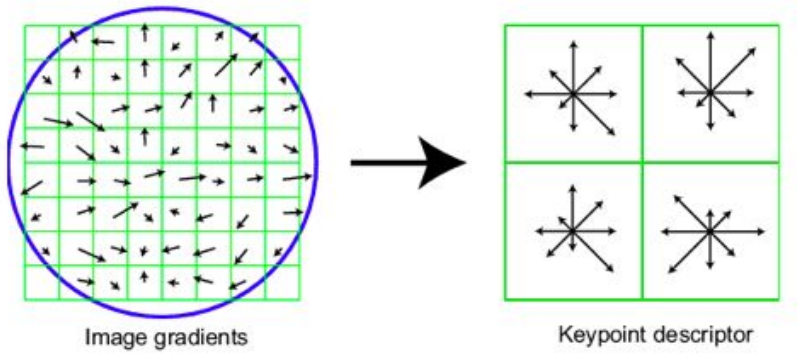
2. Feature descriptor calculation

A numerical vector that encodes the appearance or shape of the image patch around a keypoint. Classical methods include: SIFT, SURF, ORB

SIFT descriptor

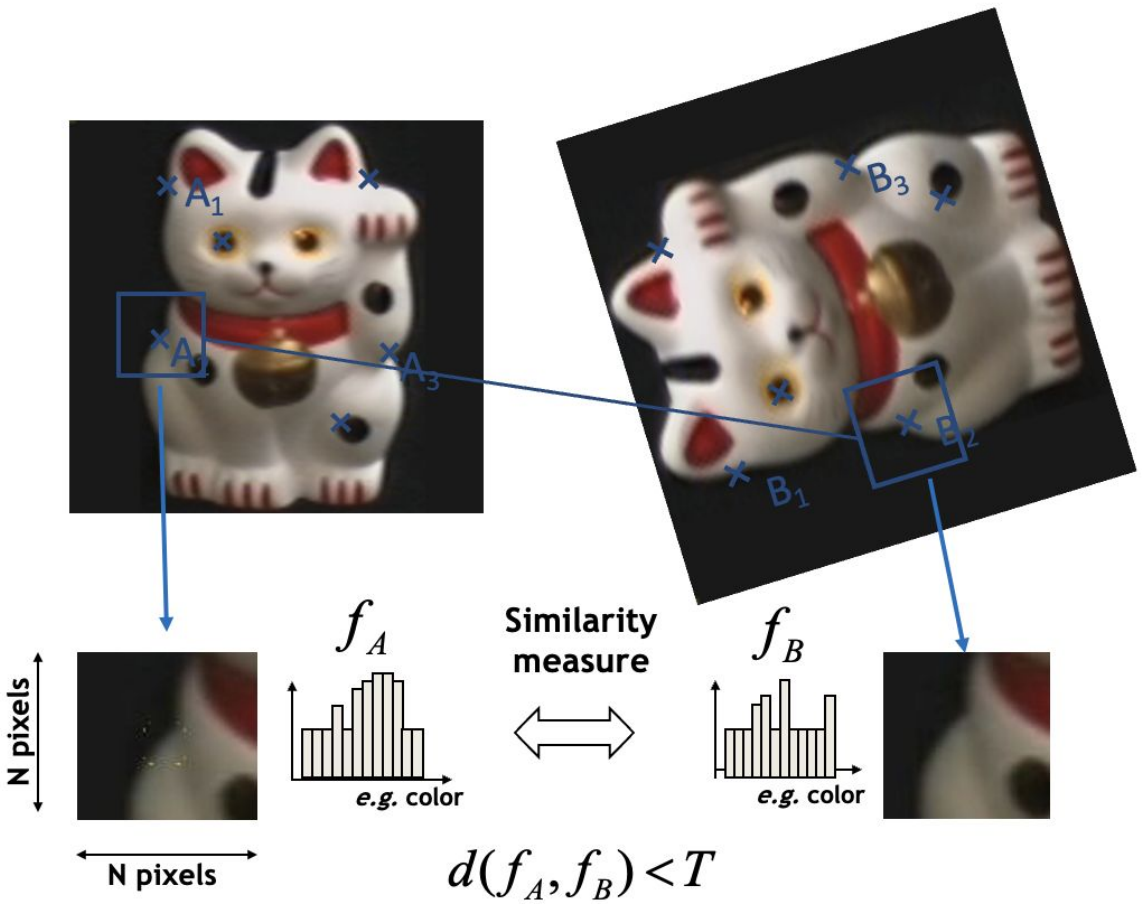
Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor



Applications ?

Image Matching through local descriptors



1. Find a set of distinctive key-points
2. Define a region around each keypoint
3. Extract and normalize the region content
4. Compute a local descriptor from the normalized region
5. Match local descriptors

Identifying correspondences between two images !!

Slide credit: Bastian Leibe

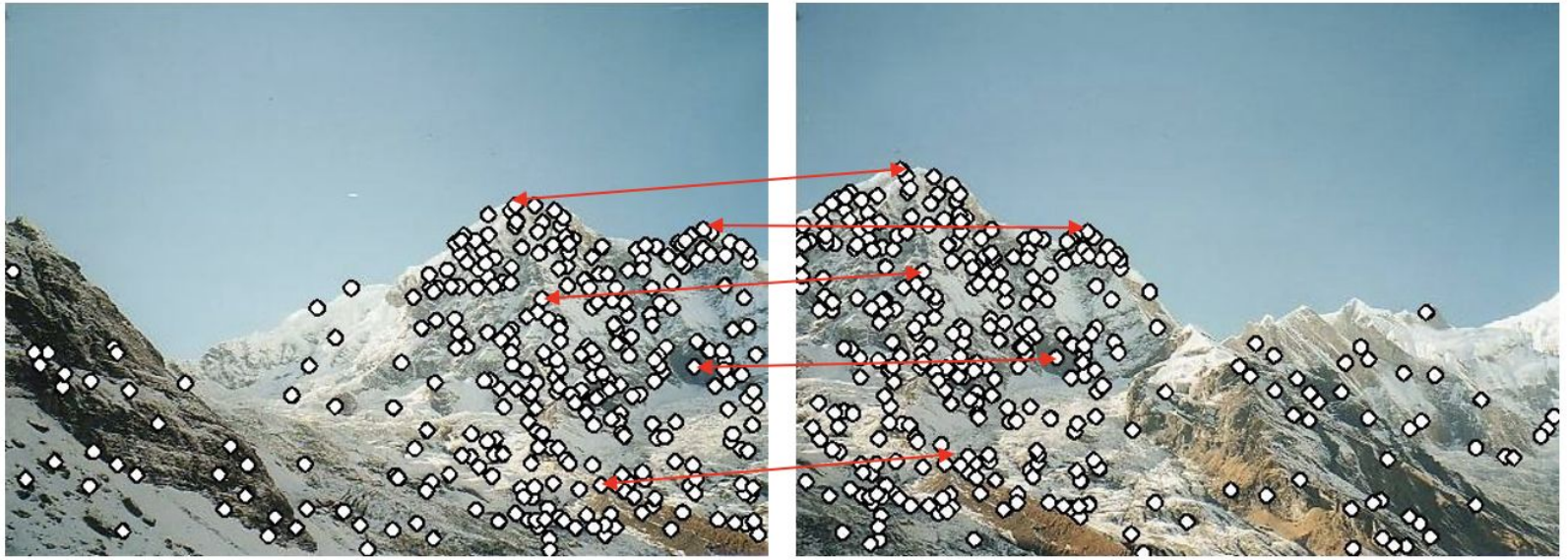
Wide-baseline matching (large-viewpoint change)

Segment Matching: matching different segments across two images using SegMASt3R.

(By leveraging a denser representation: segment-level descriptors -- uplifting local features.)

PS: Some cool work done on this by Rohit and us last semester :) -- SegMASt3R (will be up on arxiv soon)

Image Stitching



- Procedure:
  - Detect feature points in both images
  - Find corresponding pairs

Slide credit: Darya Frolova, Denis Simakov

Image Stitching

- Procedure:
  - Detect feature points in both images
  - Find corresponding pairs
  - Use these pairs to align the images

Slide credit: Darya Frolova, Denis Simakov

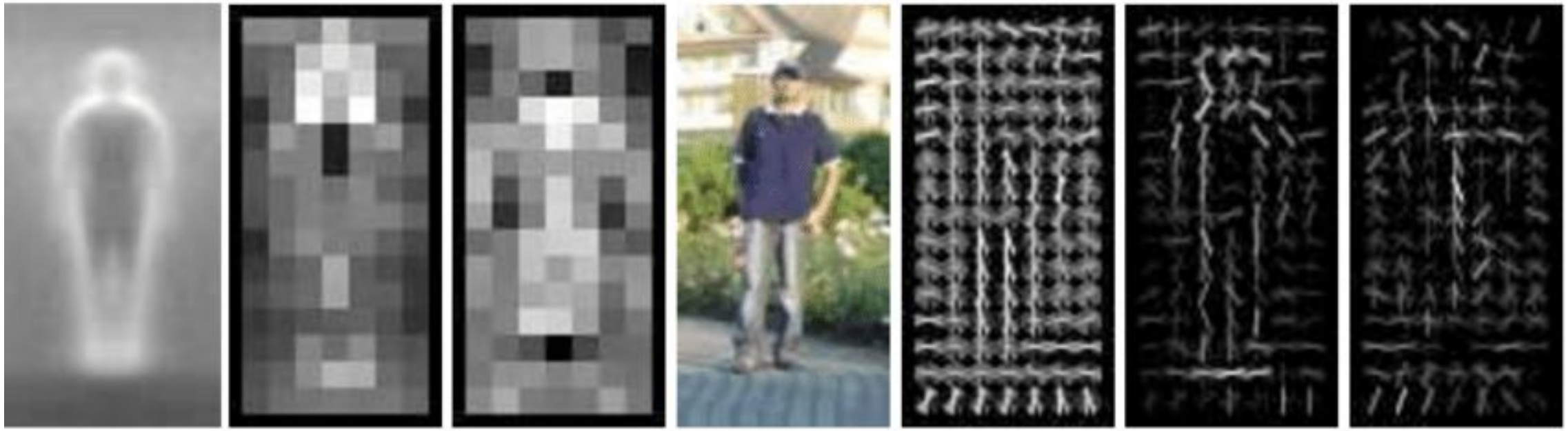
More on this in Multi-View Geometry (MVG) :)

Pre-Neural Network Era

NOTE: Slides adapted from CS7.505 CV'25.
Credits: Makarand Tapaswi, Ravikiran Sarvadevabhatla

Feature Extraction

- Scale Invariant Feature Transform (SIFT): Please go through the paper !!
  - One of the fundamental techniques which you should know
- Histogram of Oriented Gradients (HOG)



Old school image recognition

1. Feature extraction
2. Classification

0. Normalization / Preprocessing
1b. Dimensionality reduction

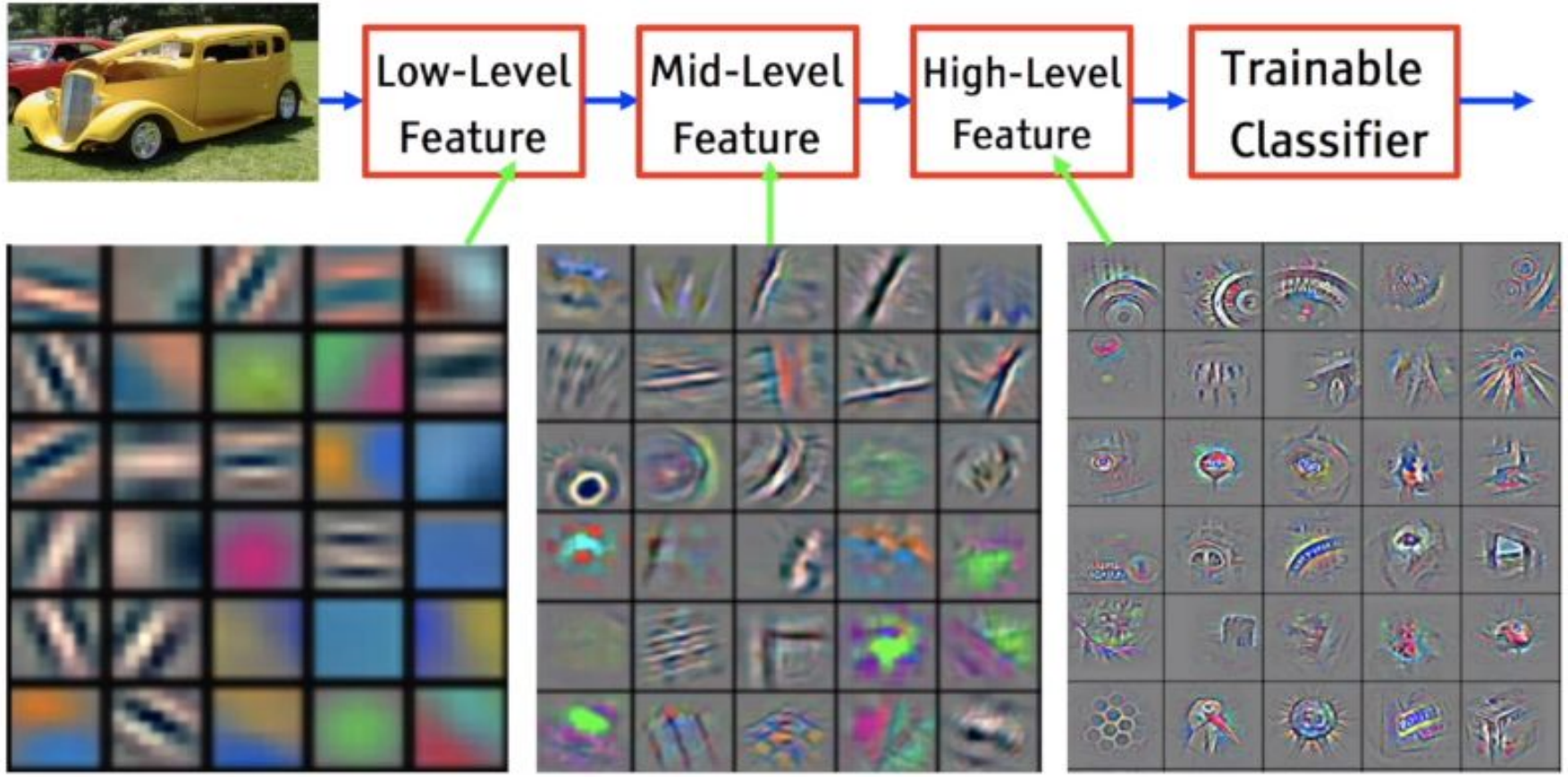
Preprocess □ Extract Features □ Reduce Dimension □ Classify

Recall how to:

- Reduce dimensions ?
  - PCA
  - tSNE
  - Learnable linear layer
  - …
- Classify ?
  - K-Nearest Neighbors
  - Multinomial logistic regression
  - SVMs
  - …

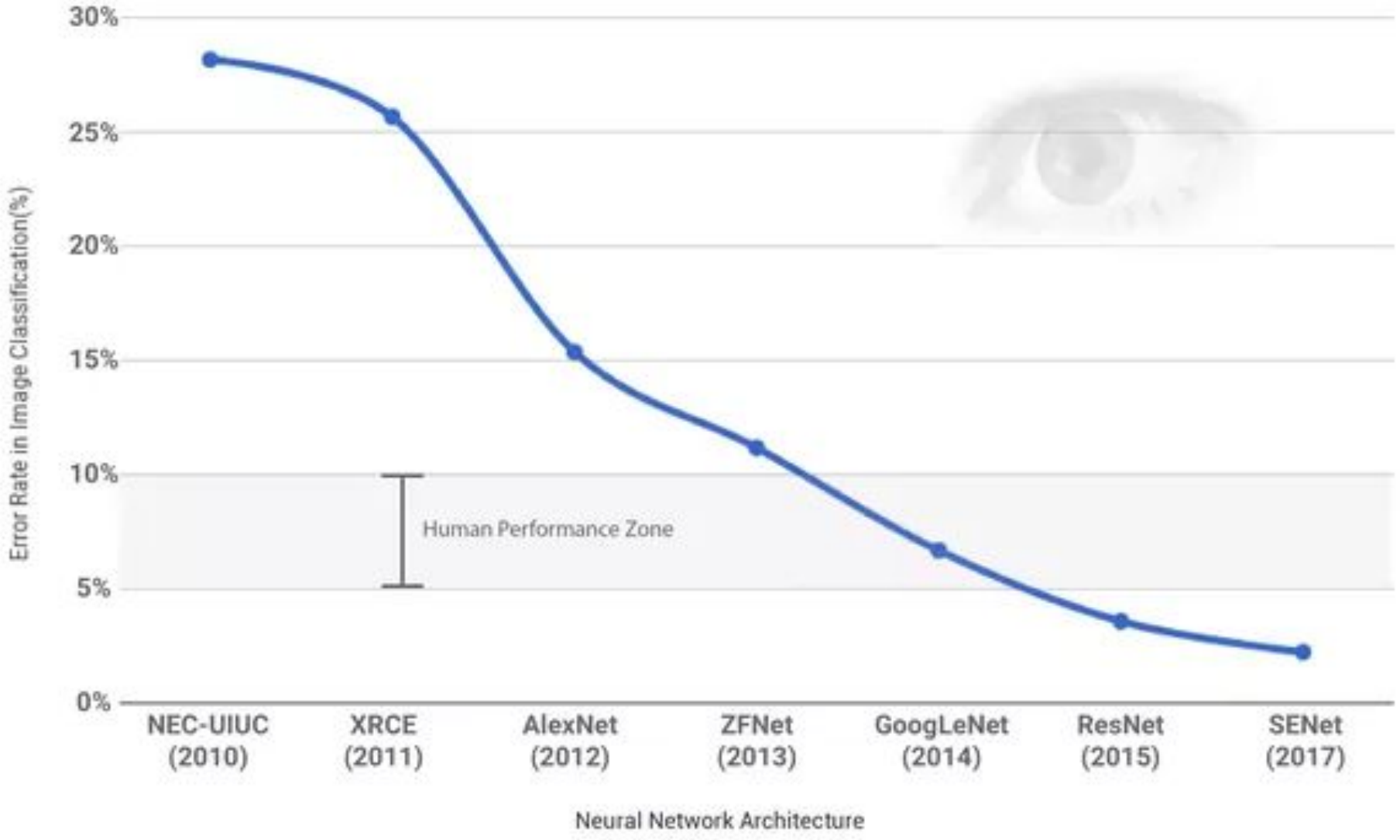
“Deep” Learning

Local to Global



Source: Yann LeCun's presentations on ConvNets

Convolutional Networks FTW!



ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

- Dataset of over a million images
- A total of 1000 categories
- Over 120+ breeds of dogs! (fine-grained)
- Really challenging, even for humans
  - What I learned from competing against a ConvNet on ImageNet
  - http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

Seems like CNNs are amazing!
But can an MLP learn as well?

Let’s do some simple math

- Recall, a four hidden layer MLP:
  - Input dimension = 100
  - Hidden dimension = 256
  - Output dimension = 10
- HW: What are the # Parameters of the following MLP ?

```
nn.Sequential(
    nn.Linear(100, 256), nn.ReLU(),
    nn.Linear(256, 256), nn.ReLU(),
    nn.Linear(256, 256), nn.ReLU(),
    nn.Linear(256, 256), nn.ReLU(),
    nn.Linear(256, 10),
)
```

Larger images?

- 100 dimensions allows for a 10 x 10 sized image
- A 50 x 50 image will require 2500 input dimensions
- An HD image (1920x1080) is about 2 million inputs!
  - (this is why it's called 2 Mega pixels)
- Do you see the problem?

Not easy for MLPs to learn *invariance*

Changes to the input **do not** affect the output

Image A

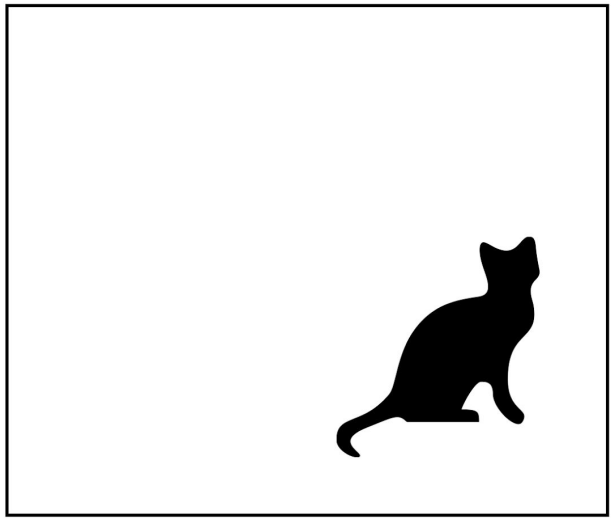
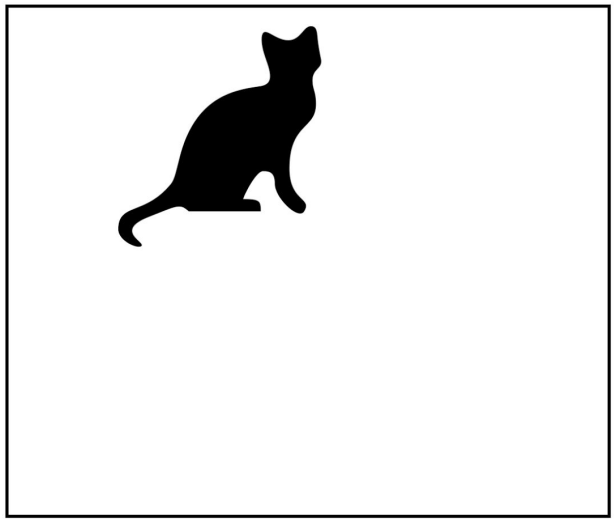


Image B

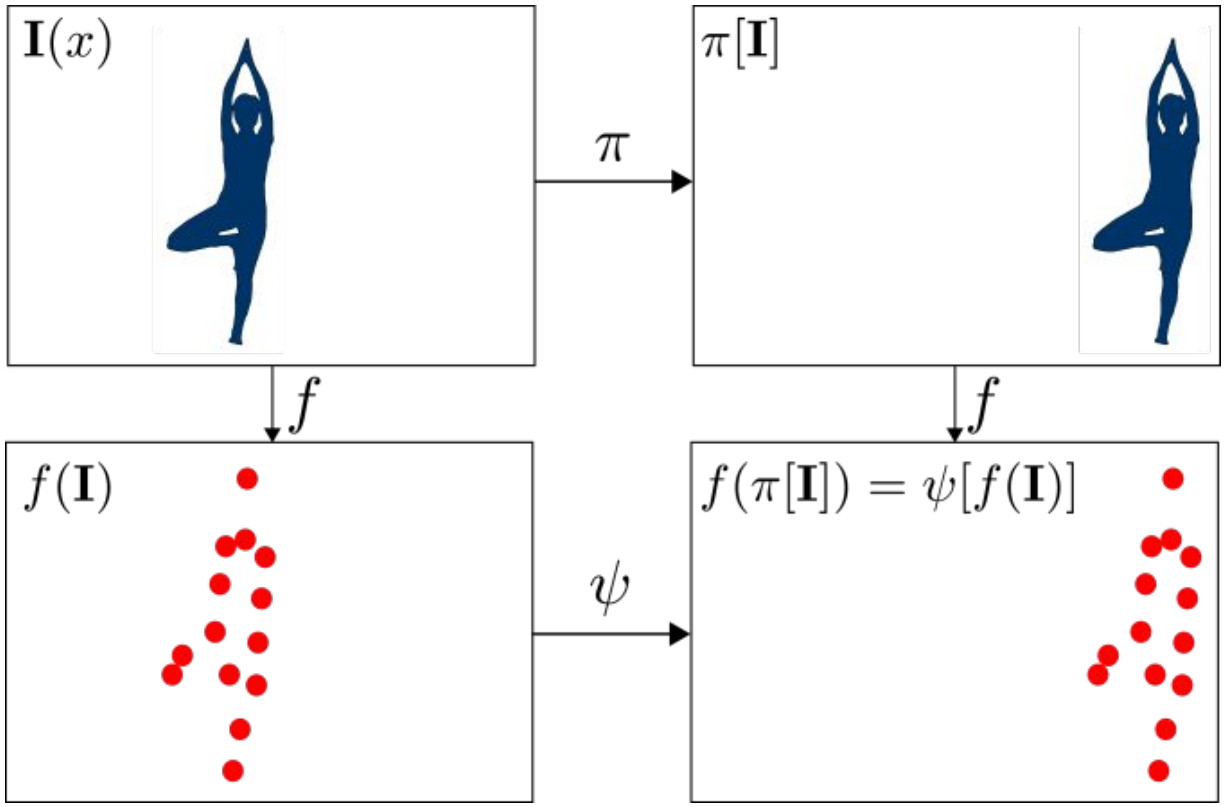


We want a model that predicts “cat” for both images!

Source: https://penkovsky.com/neural-networks/day5/

… even harder to learn *equivariance*

Changes to the input **predictably** (equivalently) affect the output



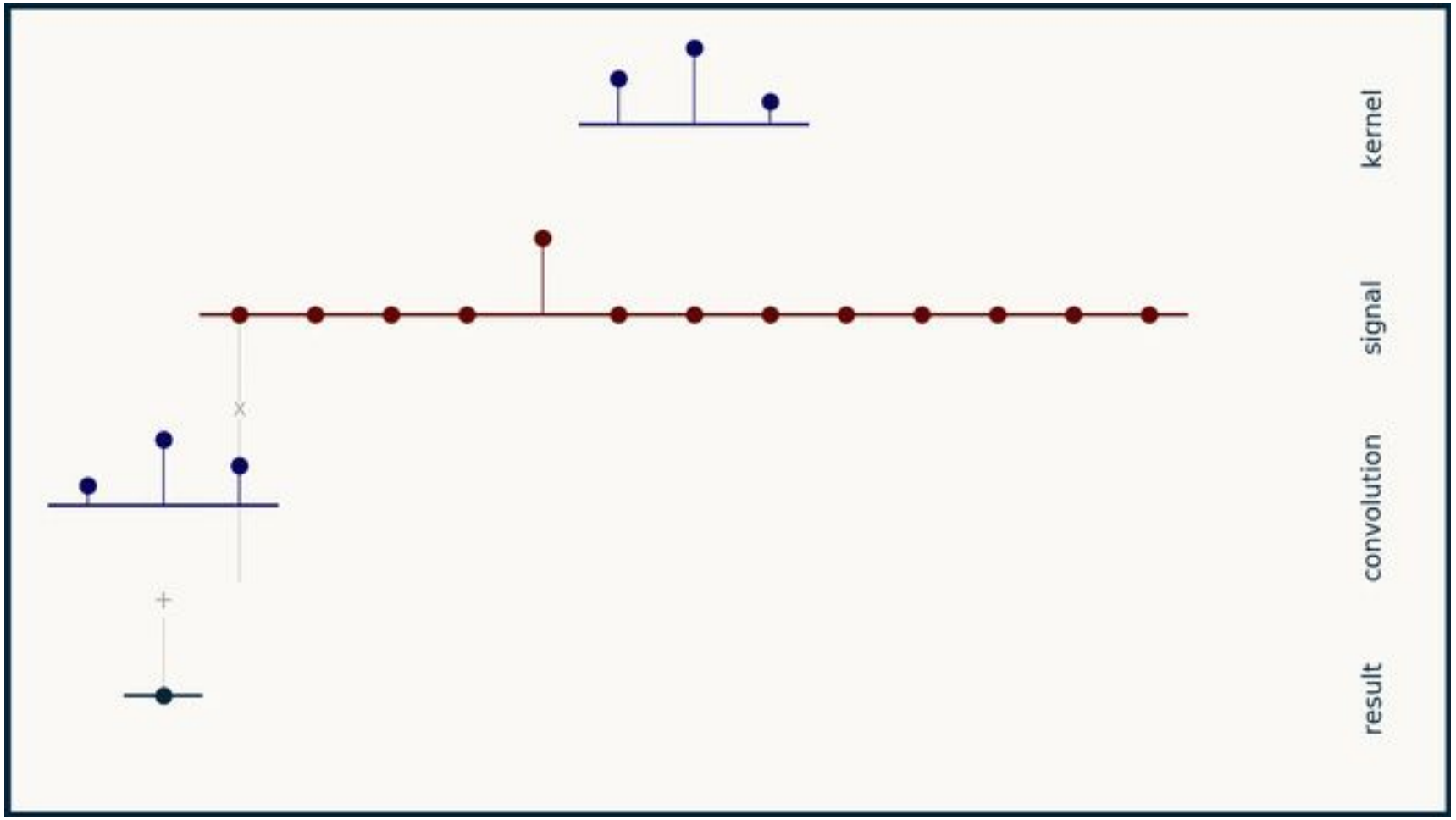
Source: https://towardsdatascience.com/translational-invariance-vs-translational-equivariance-f9fbc8fca63a

First, what does a 1D convolution do?

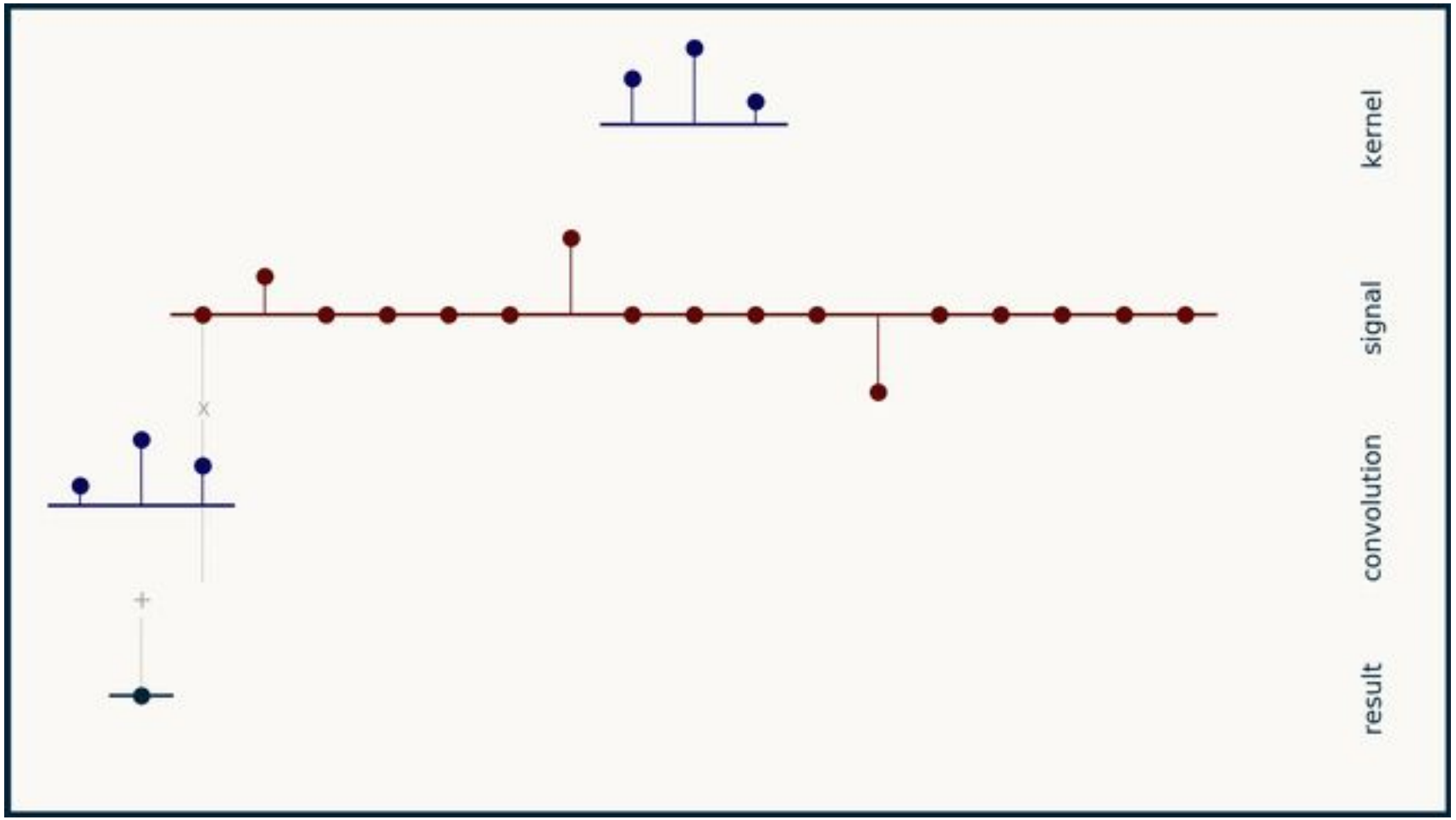
(remember your signal processing?)

Source: Brandon Rohrer https://e2eml.school/convolution_one_d.html

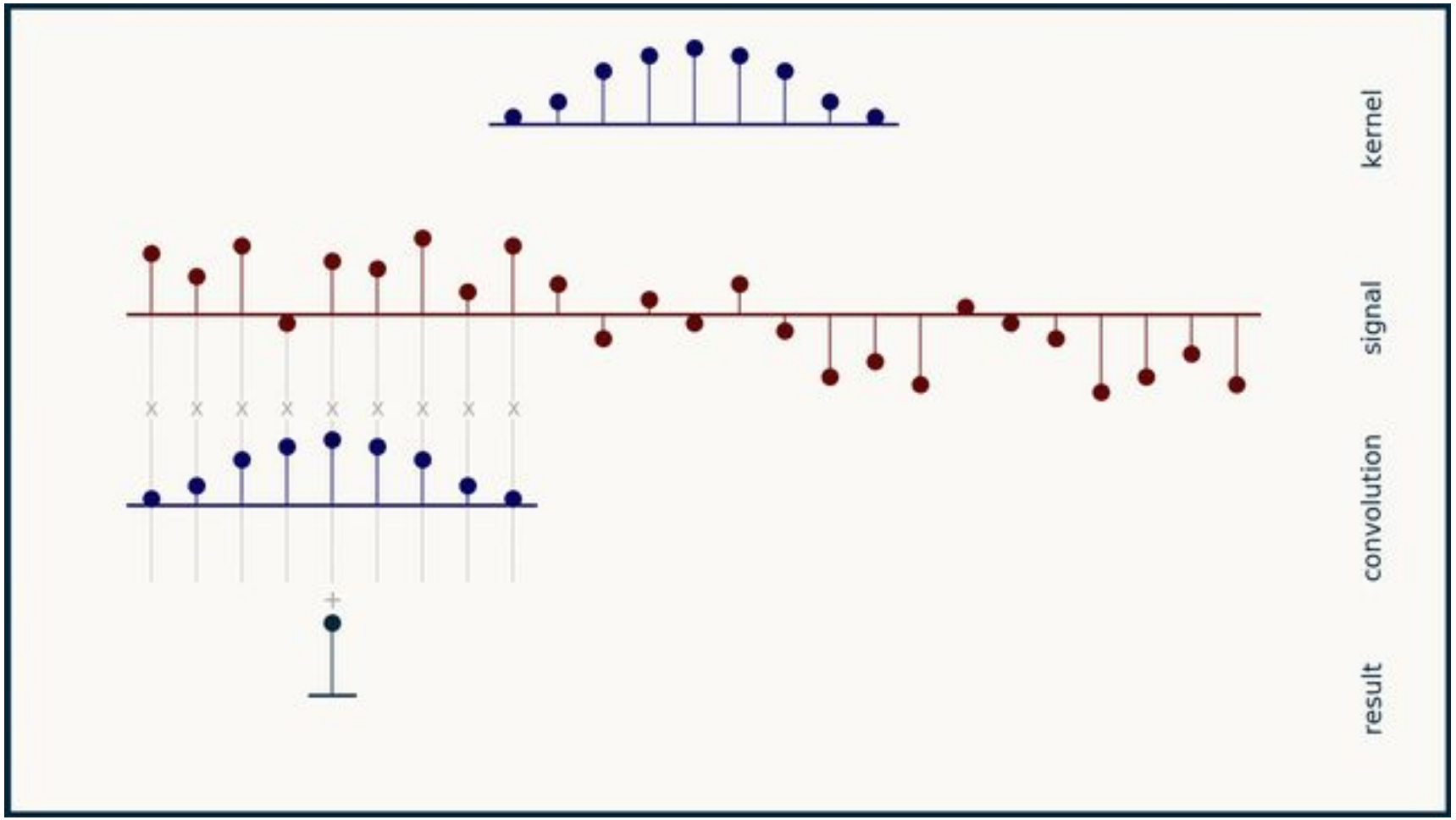
Convolutions in 1D: Impulse Response



Convolutions in 1D: Multiple Kronecker deltas



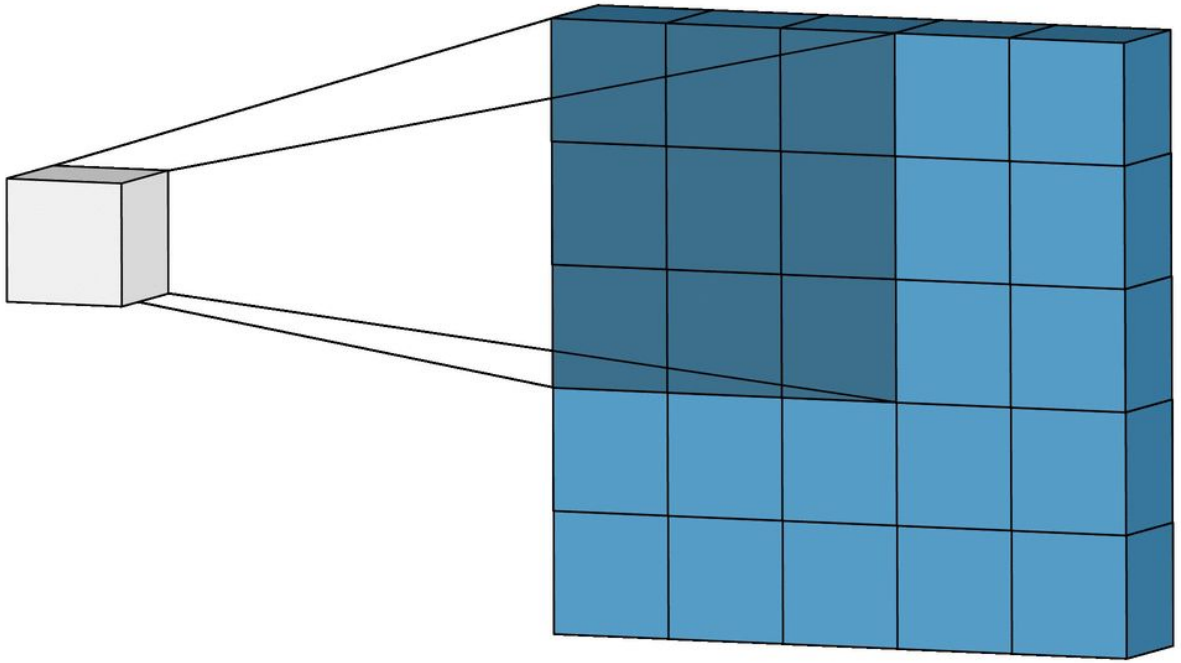
Convolutions in 1D: Gaussian averaging



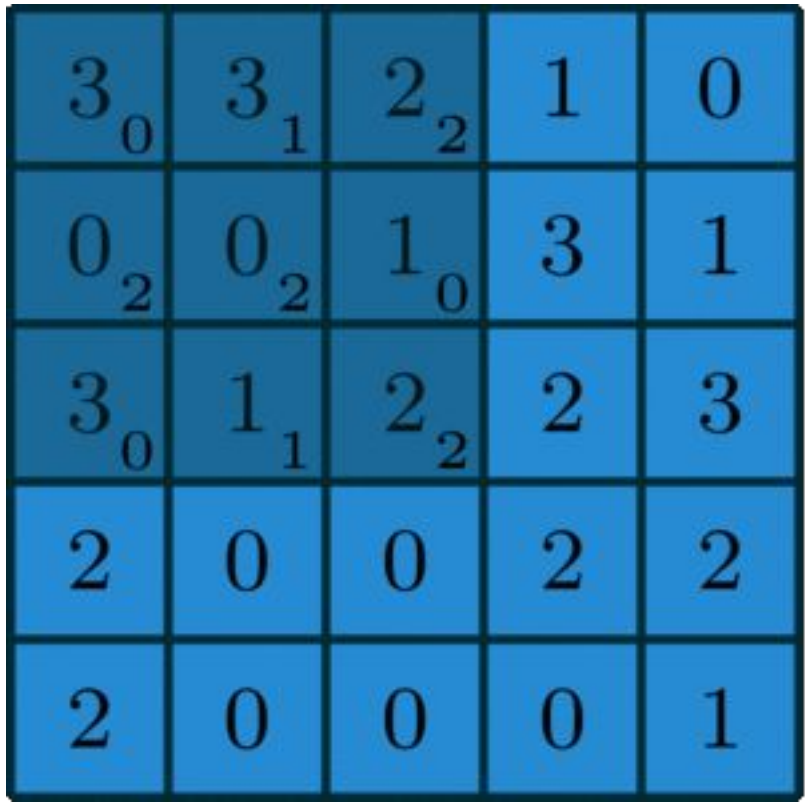
What does a 2D convolution do?

Source: Intuitively Understanding Convolutions for Deep Learning, Irhum Shafkat
https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

MLP like operation, now on 2D image patches

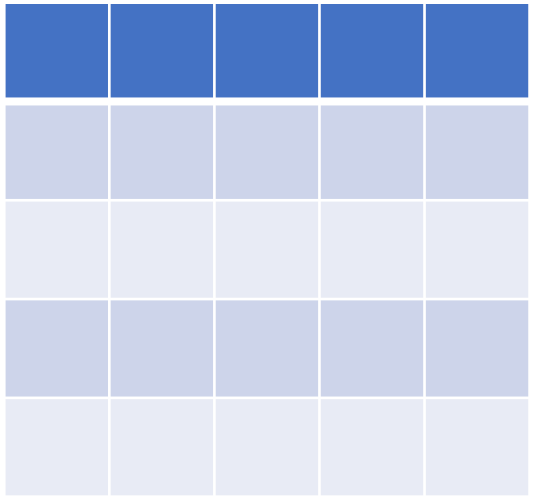


Example: 2D Convolution, with numbers

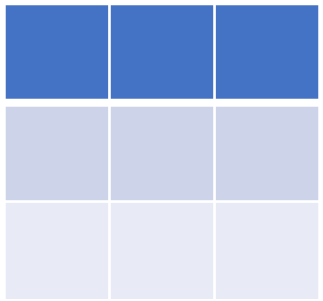


Let's generalize 2D Convolutions

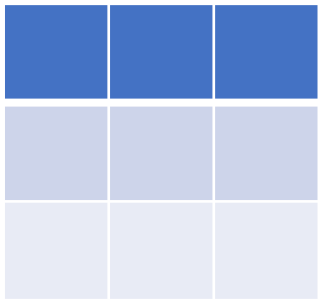
Input Data x



Conv. Kernel w (flipped)



Output o



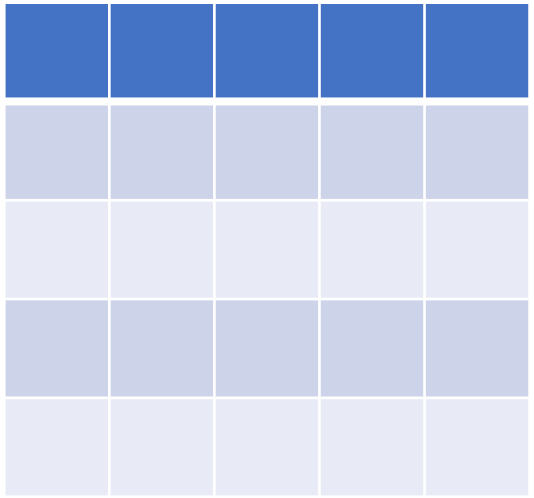
Quiz

$$o_{22} = w_{11}x_{11} + w_{12}x_{12} + \cdots + w_{33}x_{33}$$
$$o_{33} = w_{11}x_{22} + w_{12}x_{23} + \cdots + w_{33}x_{44}$$
$$o_{43} = w_{11}x_{32} + w_{12}x_{33} + \cdots + w_{33}x_{54}$$

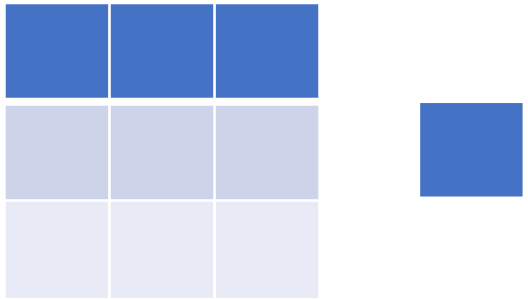
There is something missing here, can you identify it?

A proper Conv2D layer has a weight **and bias**

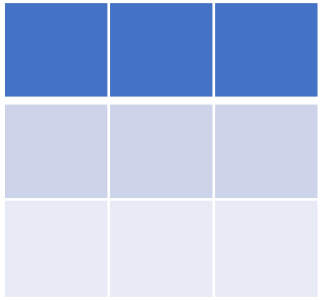
Input Data x



Conv. Kernel w (flipped)



Output o



$$o_{22} = w_{11}x_{11} + w_{12}x_{12} + \cdots + w_{33}x_{33} + b$$
$$o_{33} = w_{11}x_{22} + w_{12}x_{23} + \cdots + w_{33}x_{44} + b$$
$$o_{43} = w_{11}x_{32} + w_{12}x_{33} + \cdots + w_{33}x_{54} + b$$

For “brevity” we’ll be dropping the biases from here on out, but always assume they exist unless mentioned otherwise!

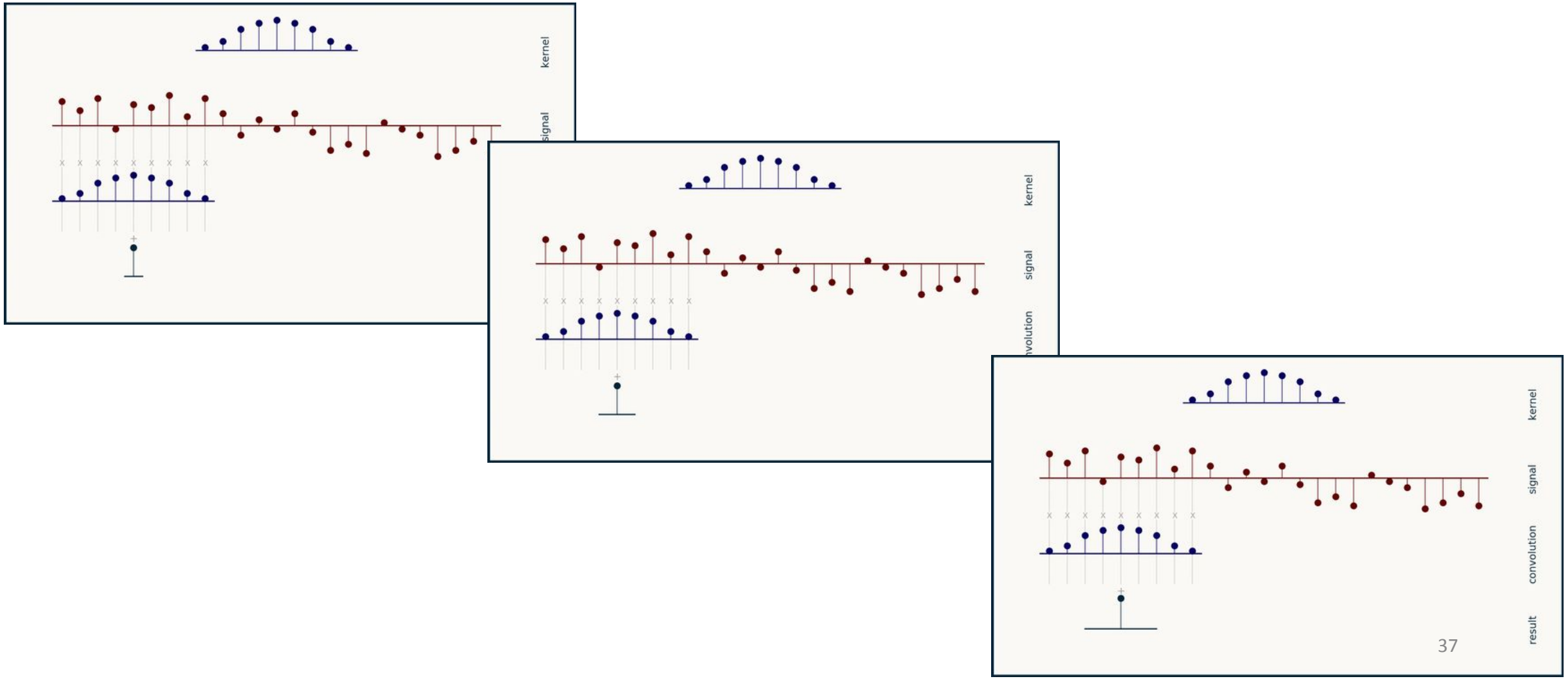
Do we now understand convolutions?

Sources:
1D conv images from Brandon Rohrer https://e2eml.school/convolution_one_d.html
2D conv images from https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

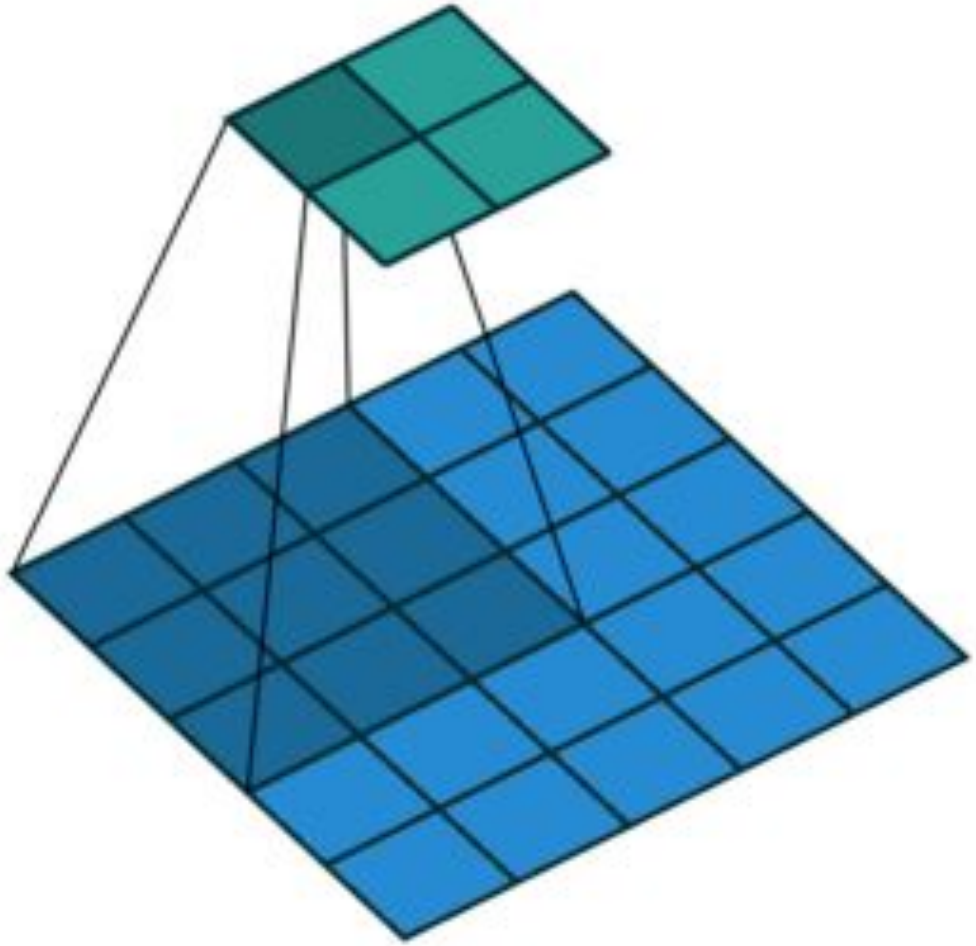
Can we predict the output size?

- Input size
- Kernel size
- Other things?
- Stride
- Padding
- Multiple channels

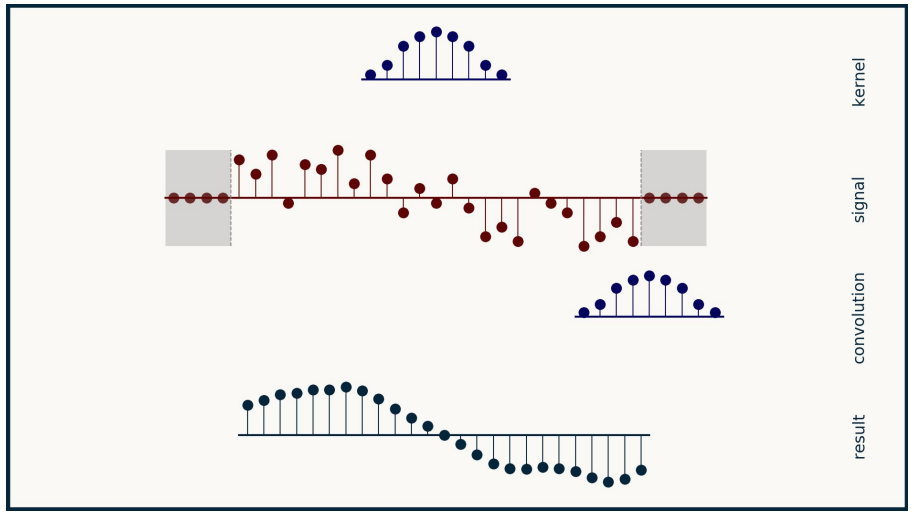
1D Convolutions, stride > 1

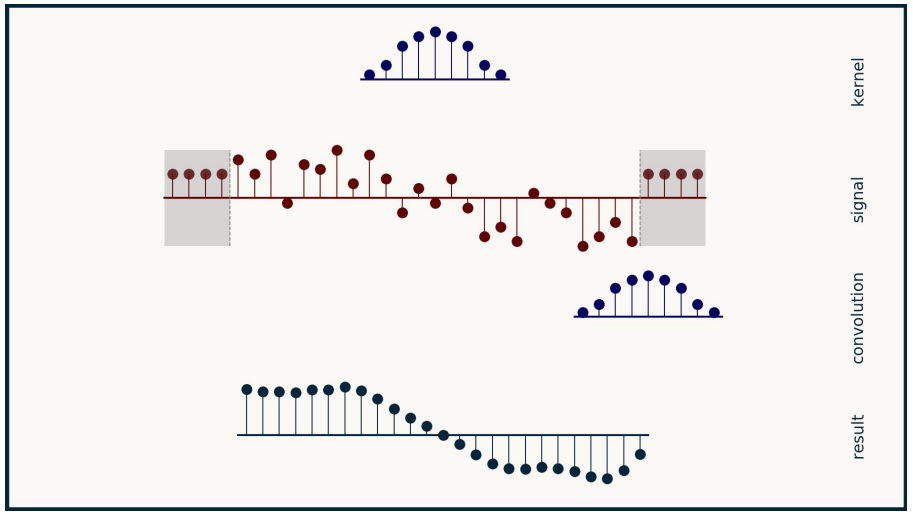


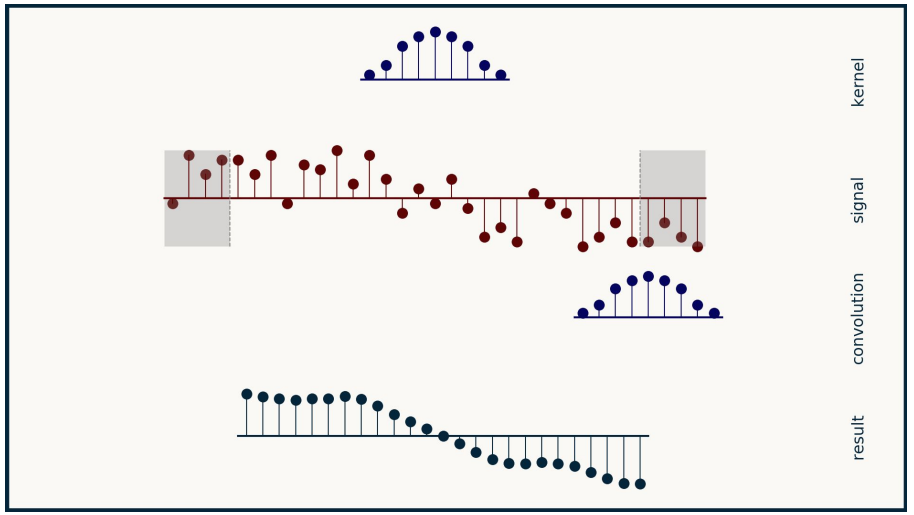
2D Convolutions, stride=2



1D Convolutions, padding



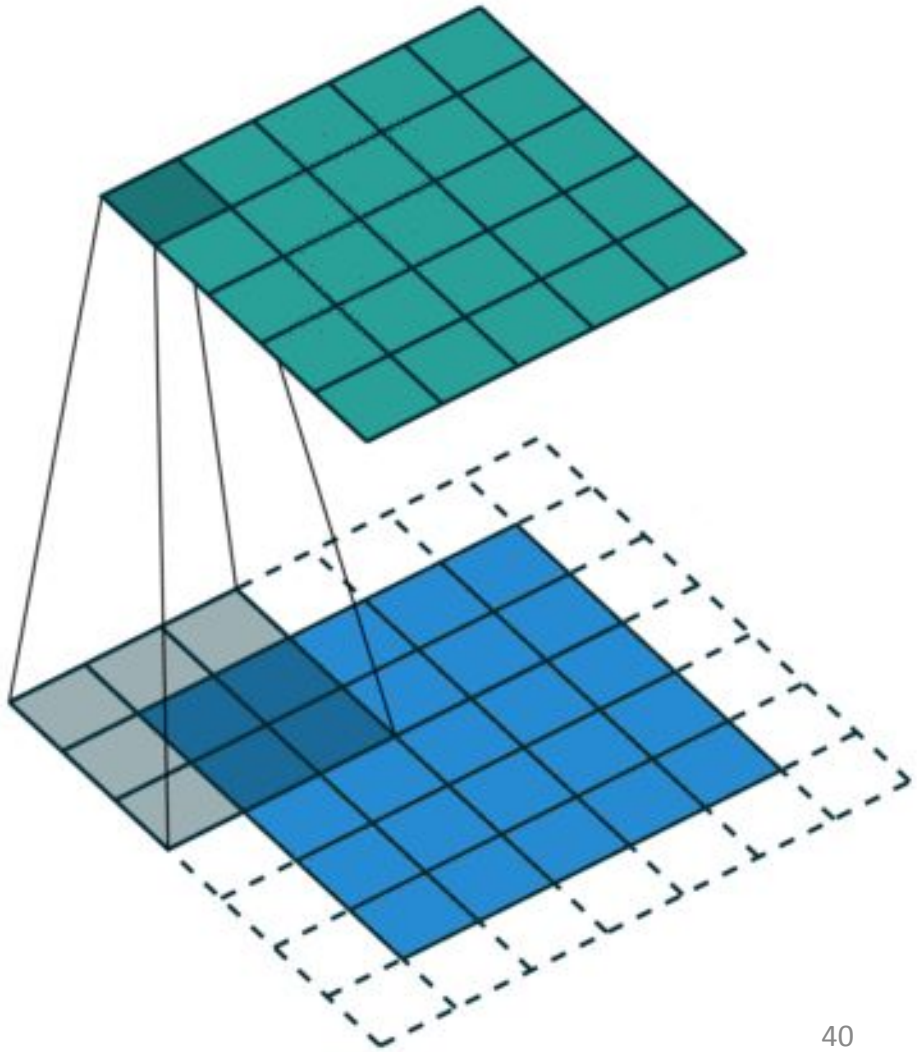




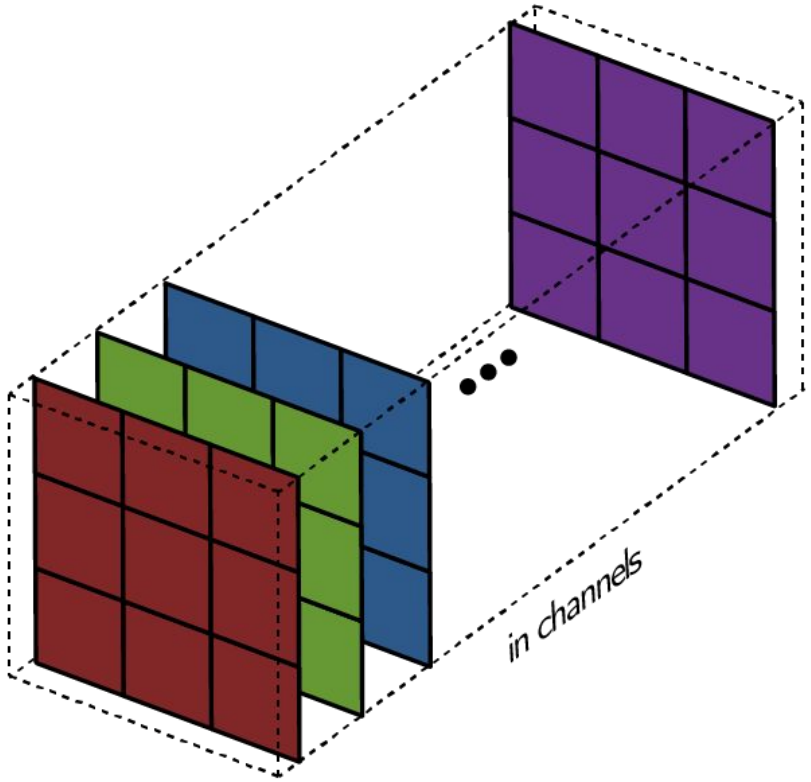
2D Convolutions, padding=1

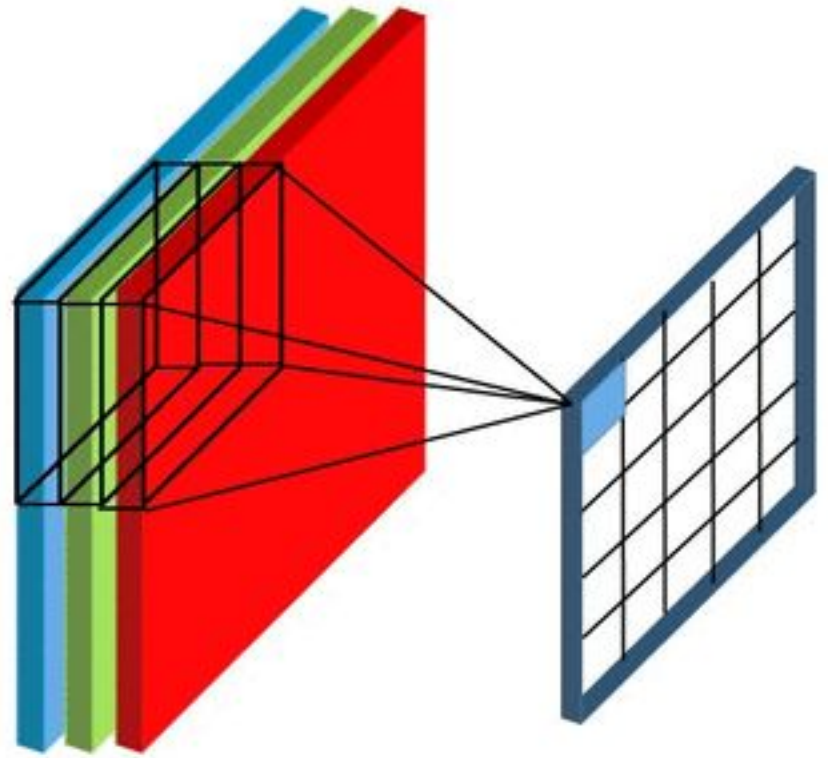
HW:

For a kernel of size K, what value of padding is necessary for output size to be equal to input? (assume stride=1)



2D Convolutions, multiple channels





Now, can we predict the output size?

Input	Kernel	Stride	Padding	Output
15x15	2x2	1	0	
15x15	3x3	1	0	
15x15	3x3	1	1	
15x15	5x5	1	1	
15x15	5x5	2	1	
15x15	5x5	1	2	
15x15	5x5	2	2	

Homework: Derive a formula to compute output size given the 4 pieces of information.

Inputs and Outputs

- Input image: $B \times C_{in} \times H_{in} \times W_{in}$
  - Multi-channel processing C_{in}
  - Input data is a batch of B samples
- Convolution filter
  - Weight parameters: $C_{out} \times C_{in} \times K_H \times K_W$
  - Bias parameters: C_{out}
- Output activation: $B \times C_{out} \times H_{out} \times W_{out}$
- C_{out} filters of shape $C_{in} \times K_H \times K_W$ operate on each activation

Non-linearities

Vision is non-linear!

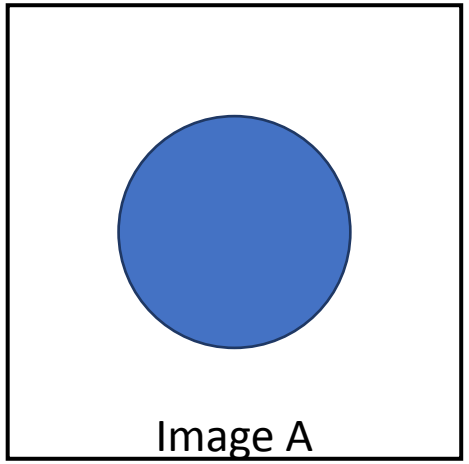


Image A

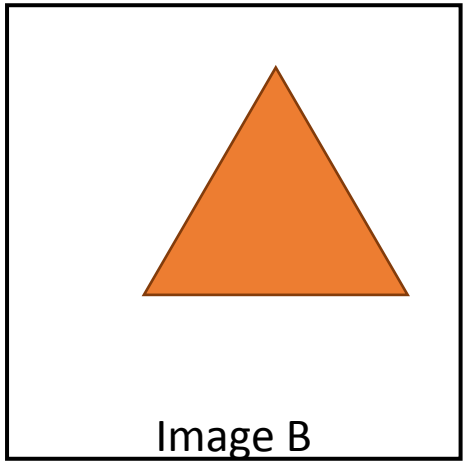
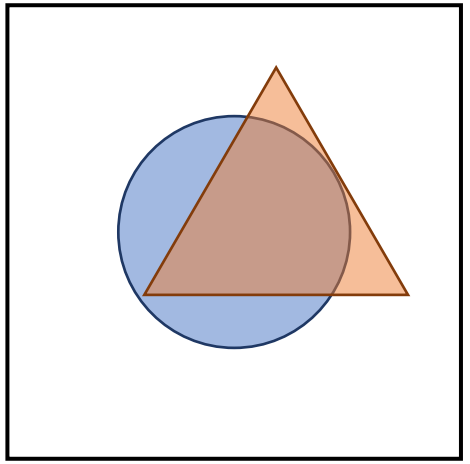
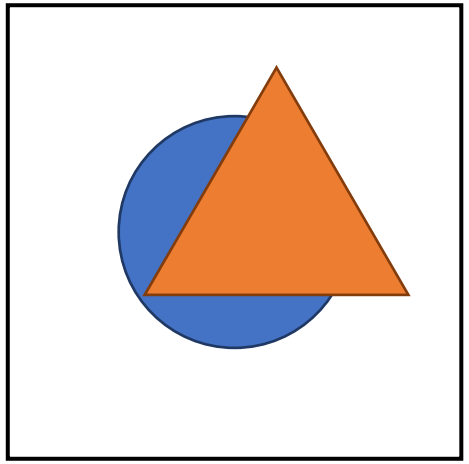


Image B



Linear combination
A+B



Actual combination
A+B

No non-linearities, what happens?

Without non-linearities

- $h_1 = W_1 x + b_1$
- $o = W_2 h_1 + b_2$

What's the problem here?

- $o = W_2 W_1 x + (W_2 b_1 + b_2)$
- $o = W^* x + b^*$

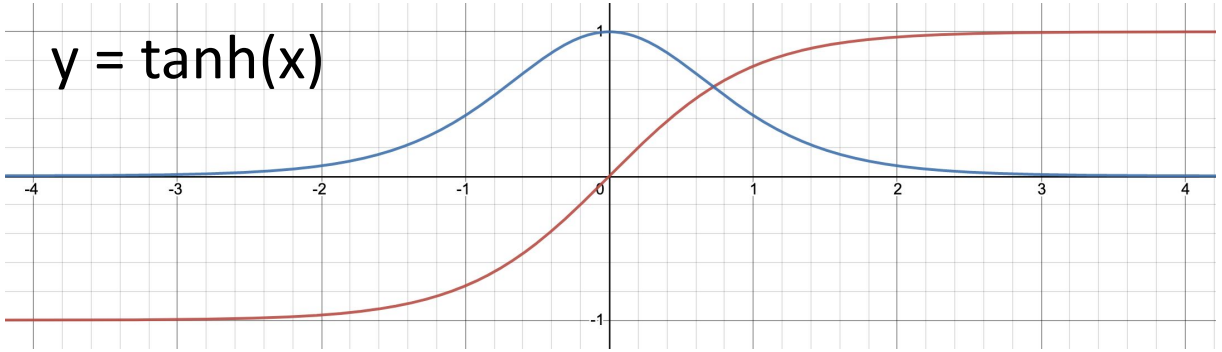
There is only one layer!

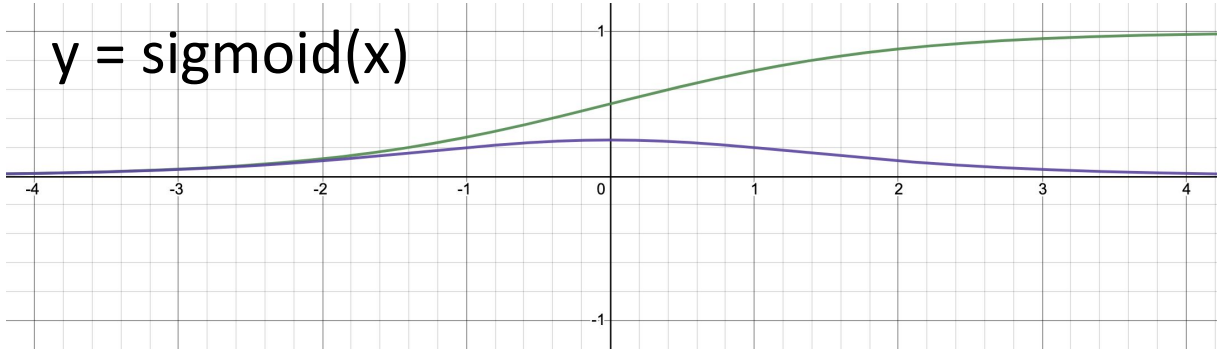
With non-linearities

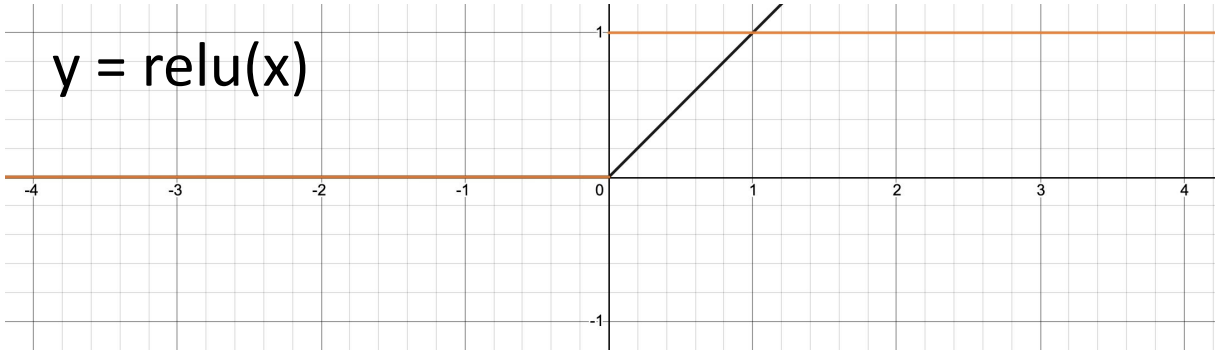
- $h_1 = \phi(W_1 x + b_1)$
- $o = W_2 h_1 + b_2$

There is meaning in having two layers now!

Simple non-linearities are often sufficient







Graphs created using https://www.desmos.com/calculator

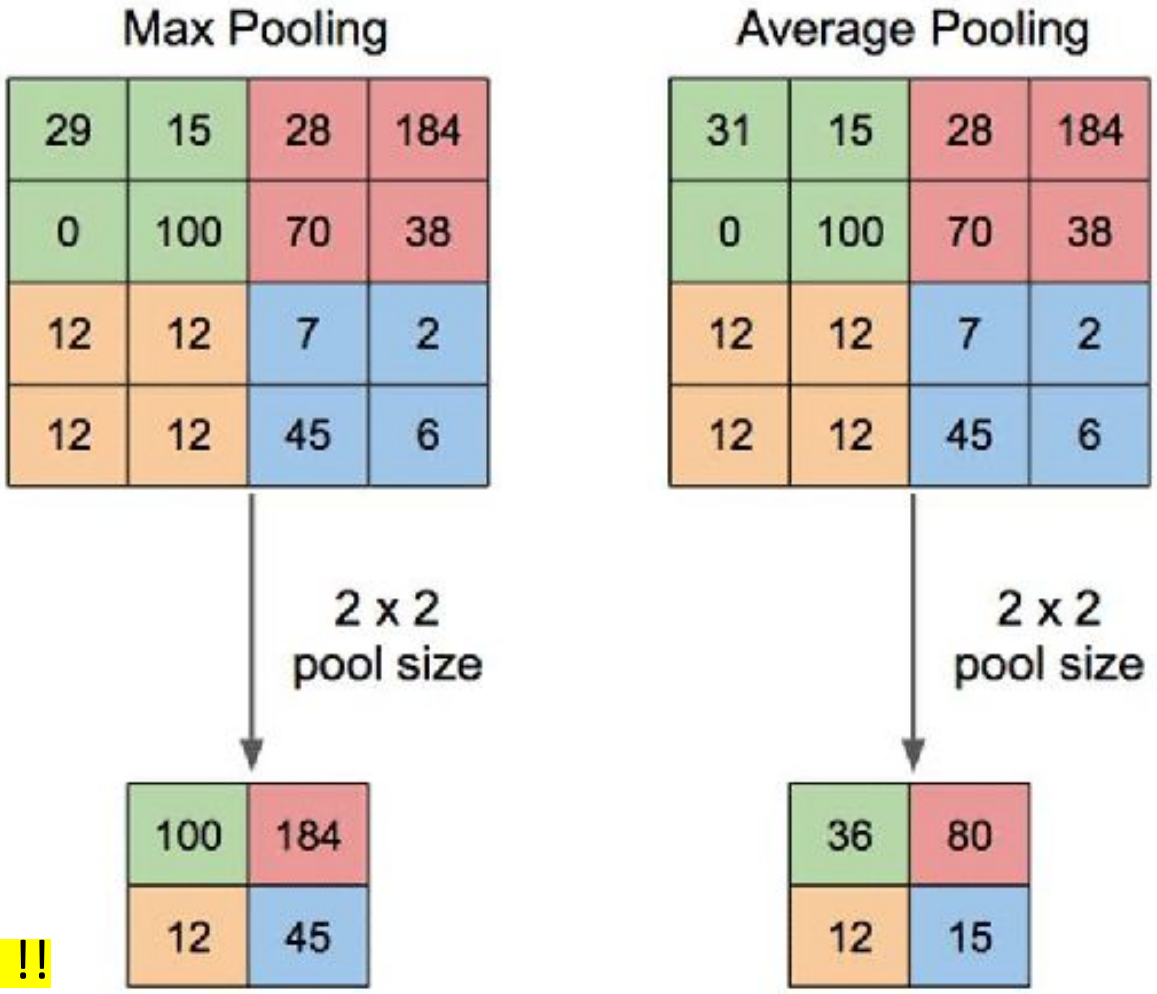
Pooling Op

Pooling

Why do we need pooling in the first place ?

Pooling “forgets” the exact pixel location in favor of a summary statistic, so small translations or deformations inside its window don’t change the network’s response. It is also an easy way to downsample :)

Recall that this is the definition of invariance !!



Are deeper models always better ?

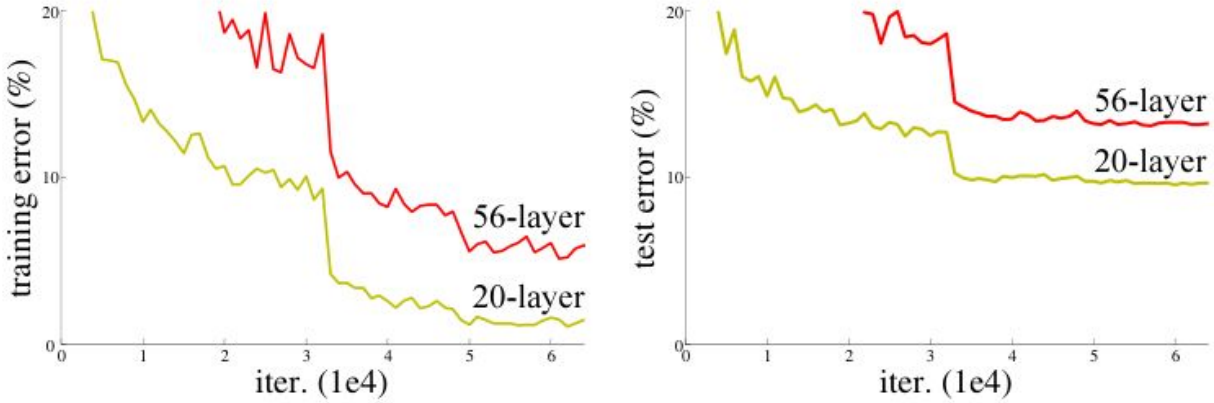


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

The degradation (of training accuracy) indicates that not all systems are similarly easy to optimize. Let us consider a shallower architecture and its deeper counterpart that adds more layers onto it. There exists a solution *by construction* to the deeper model: the added layers are *identity* mapping, and the other layers are copied from the learned shallower model. The existence of this constructed solution indicates that a deeper model should produce no higher training error than its shallower counterpart. But experiments show that our current solvers on hand are unable to find solutions that

Residual connections are very powerful!

- Solves the problem of vanishing gradients.

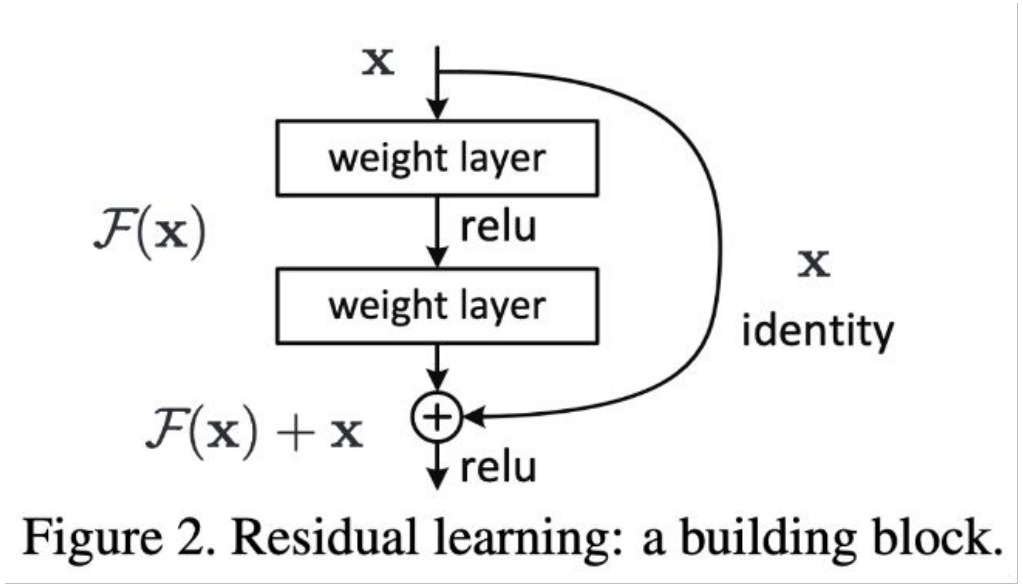
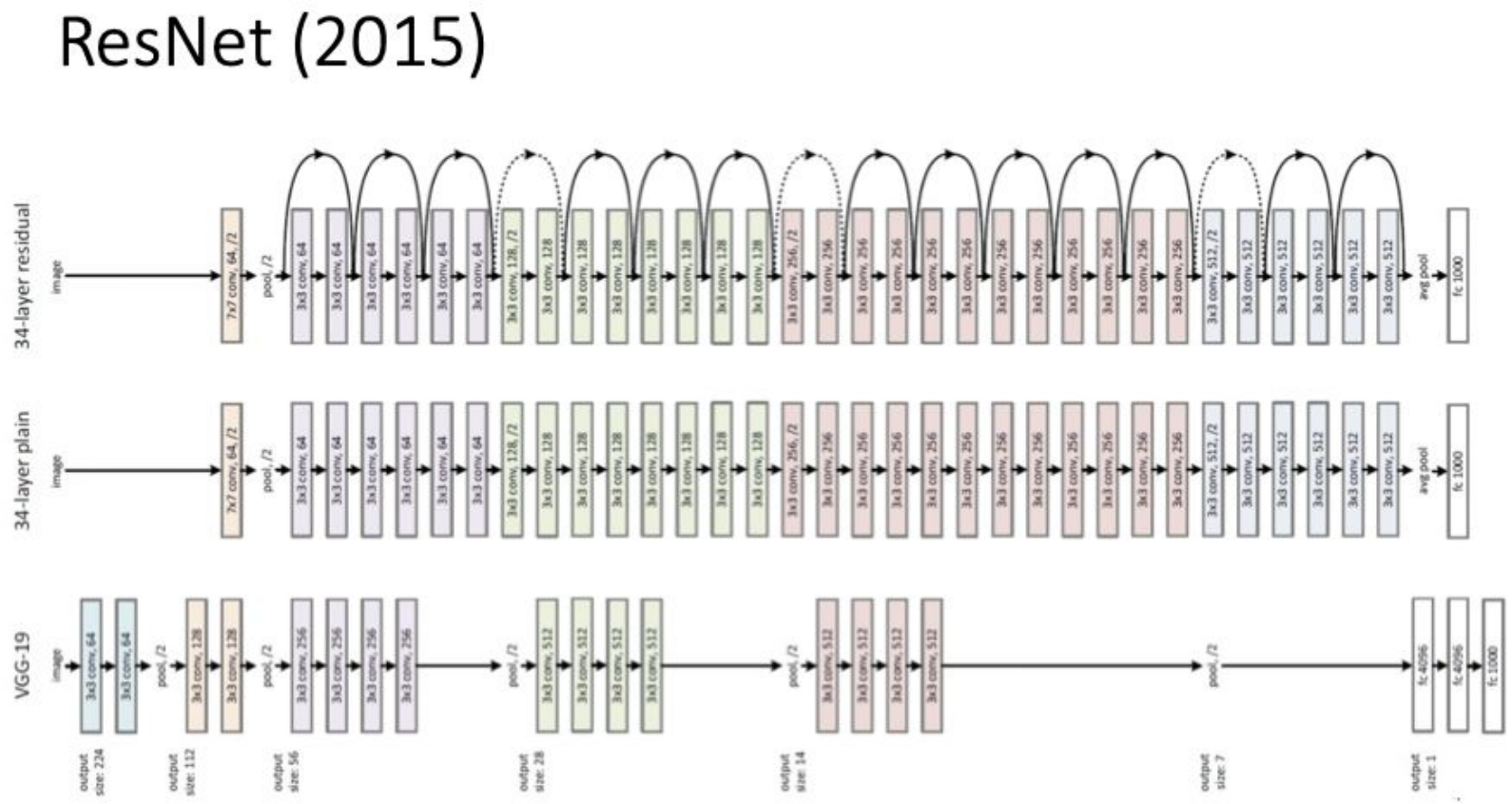


Figure 2. Residual learning: a building block.

Ok, maybe not exactly at 3, but you get the point

Residual Connections



Efficacy of Residual Connections

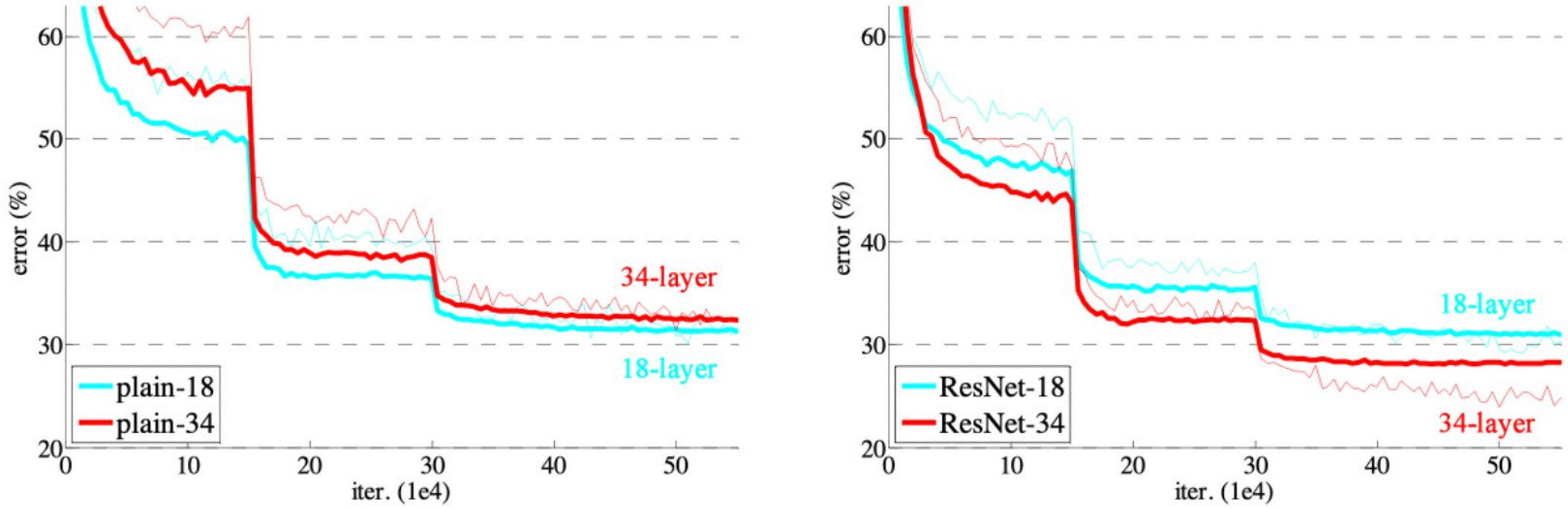


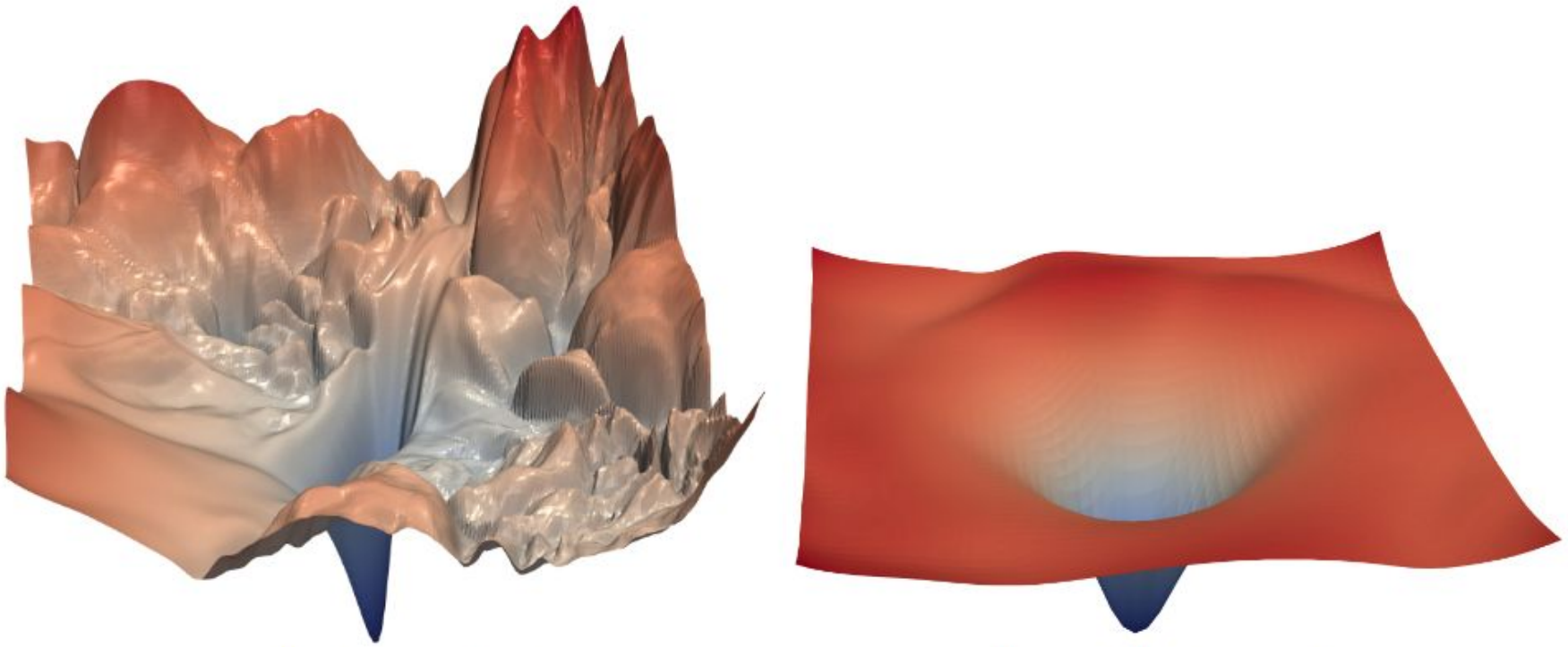
Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

Source: https://arxiv.org/abs/1512.03385

ResNet-18 Modules

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Why are residual networks easier to optimize ?



(a) without skip connections (b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

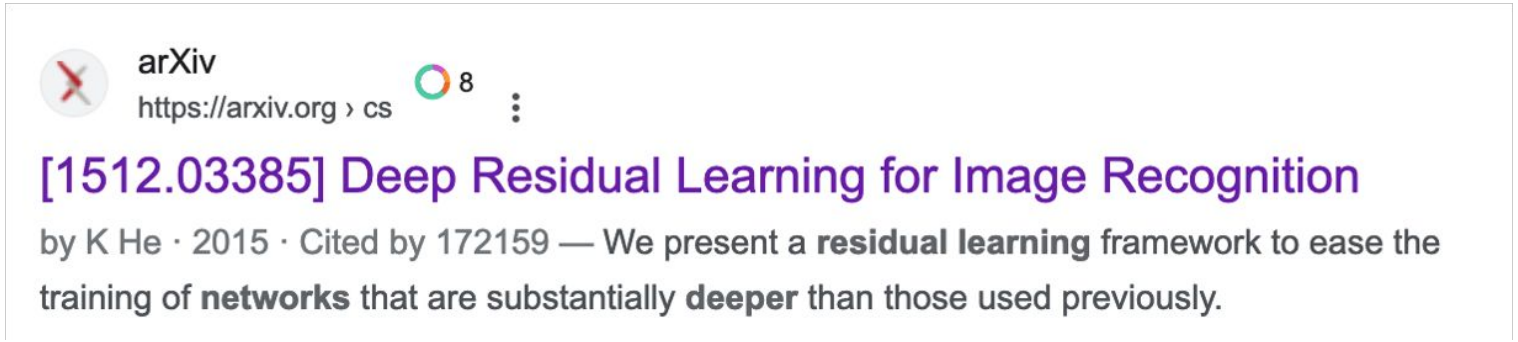
Source : [Visualizing the Loss Landscape of Neural Nets]

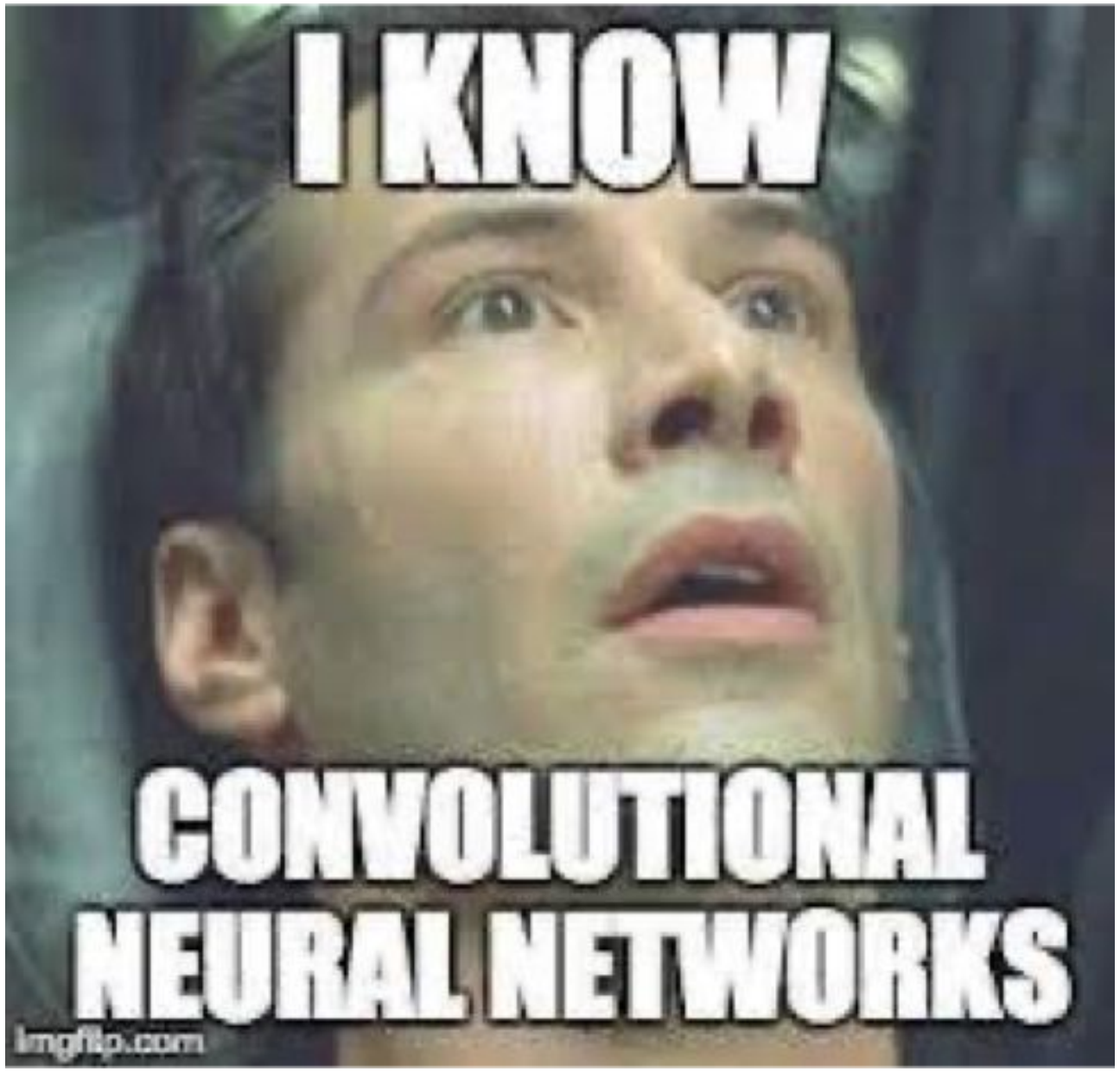
ResNet

- Add residual connections between **basic blocks**
- Easy gradient flow for deep models
- Enable very deep models
  - from ~20 layers to 150 and even 1000 layers!
- “Fully convolutional”: No linear layers except the classifier!

Please read this paper: https://arxiv.org/abs/1512.03385

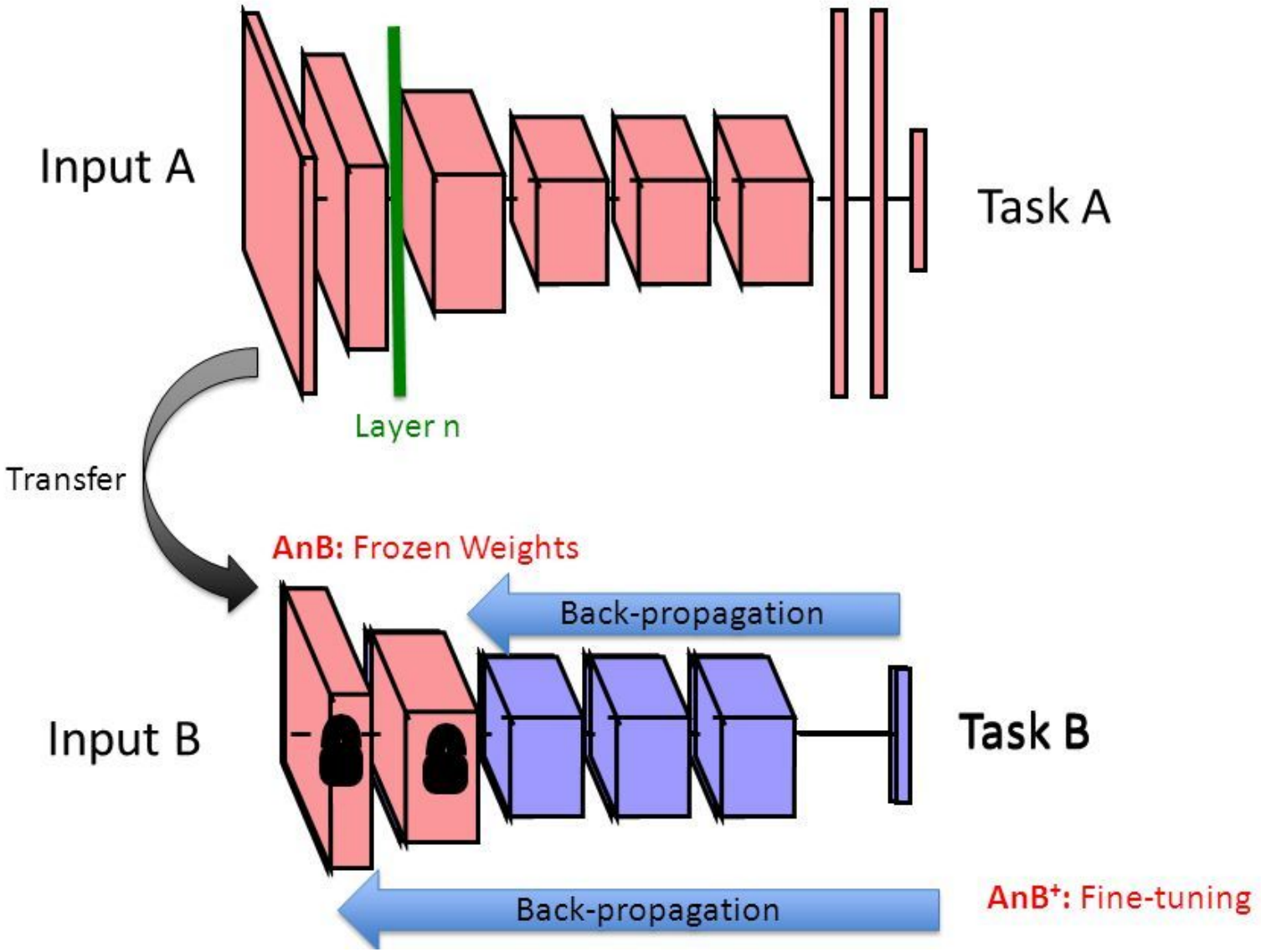
- Lots of insights + fundamental to CV research.




I KNOW
CONVOLUTIONAL
NEURAL NETWORKS
imgflip.com

Transfer Learning

Transfer Learning Overview



Transfer Learning: Feature Extraction

1. Train on ImageNet

2. Extract features with CNN, train linear model

Remove last layer

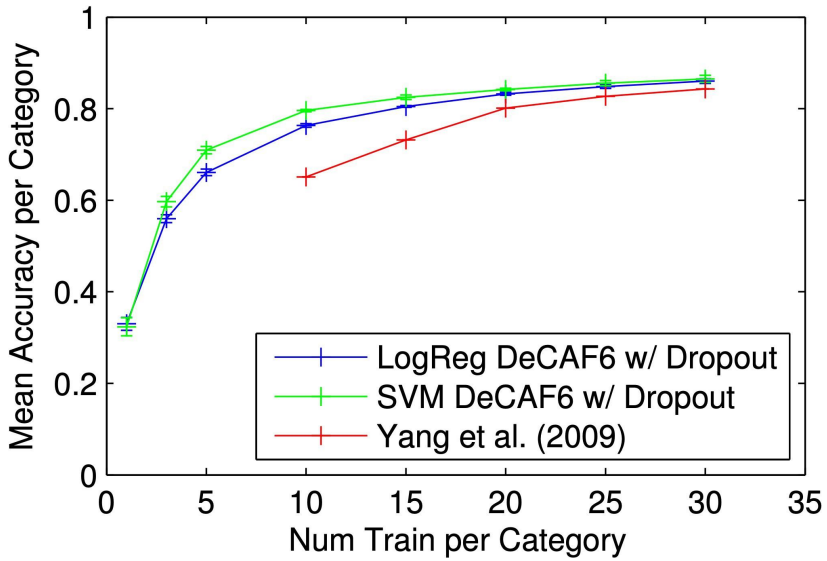
Freeze these

Transfer Learning: Feature Extraction

1. Train on ImageNet

2. Extract features with CNN, train linear model

Classification on Caltech-101



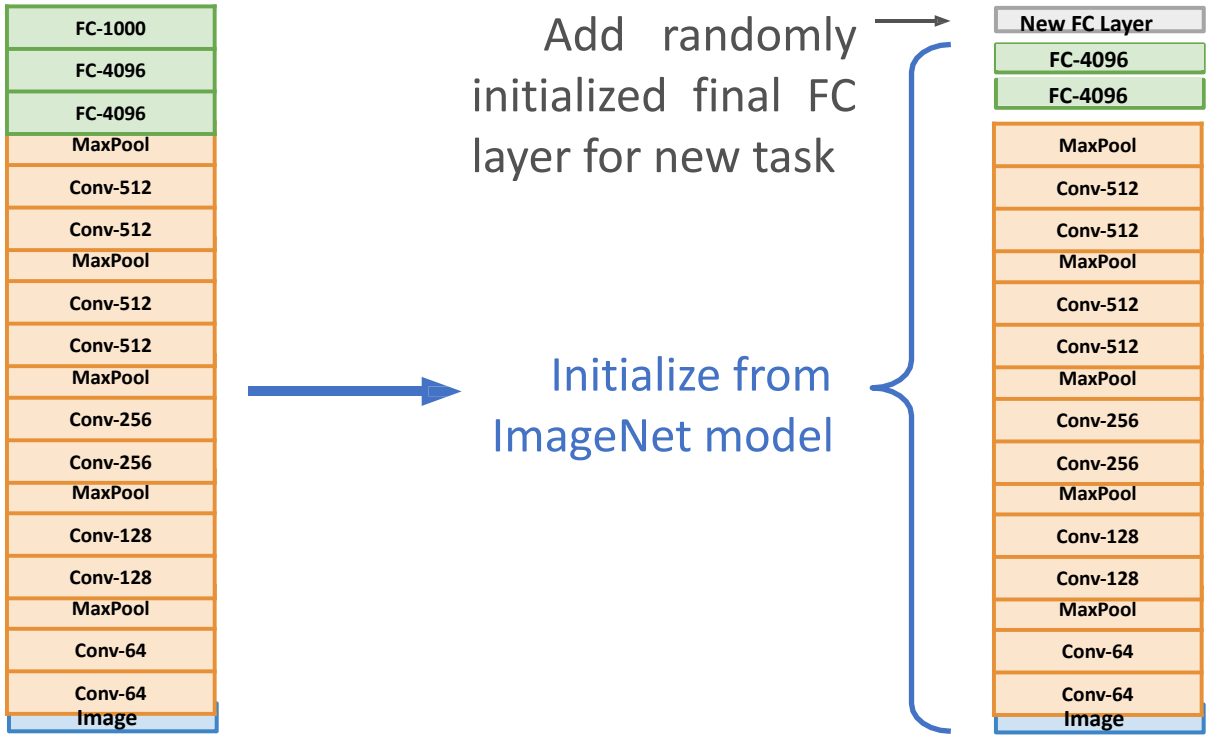
Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

Transfer Learning: Fine-Tuning

1. Train on ImageNet

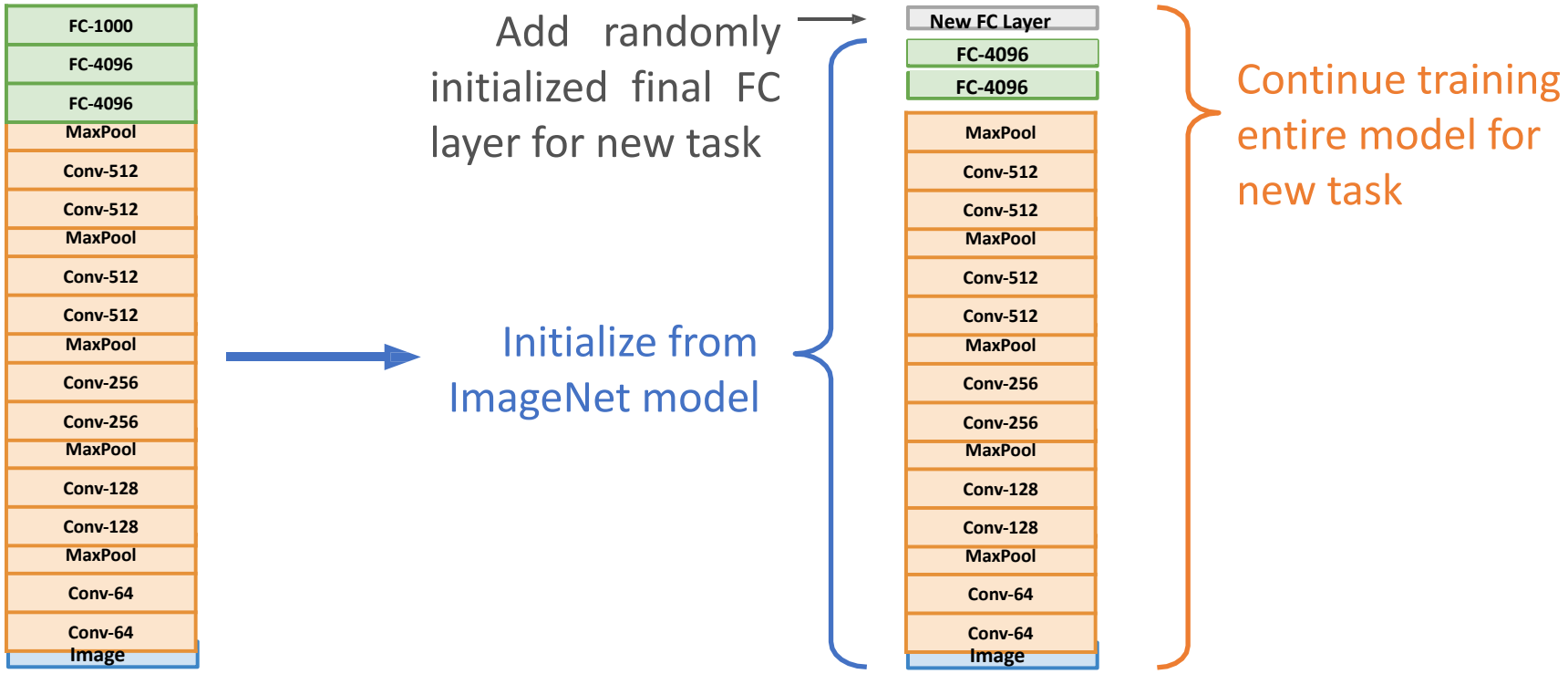
Transfer Learning: Fine-Tuning

1. Train on ImageNet



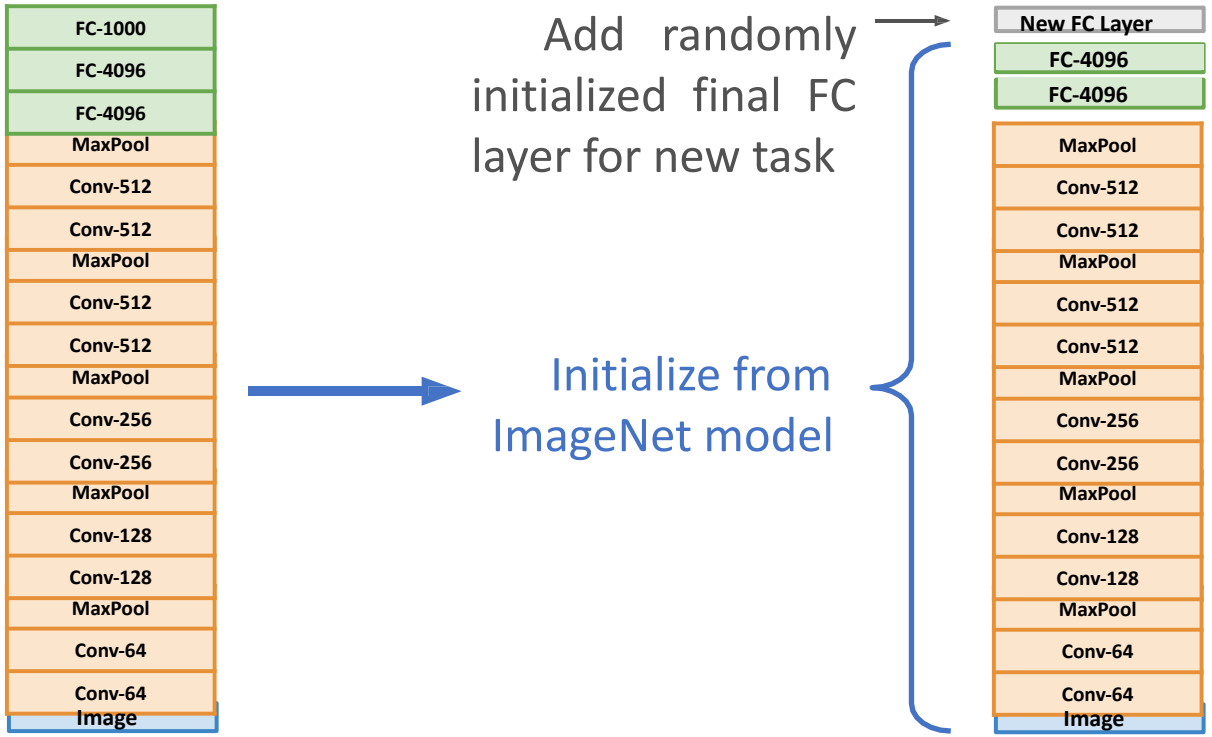
Transfer Learning: Fine-Tuning

1. Train on ImageNet



Transfer Learning: Fine-Tuning

1. Train on ImageNet

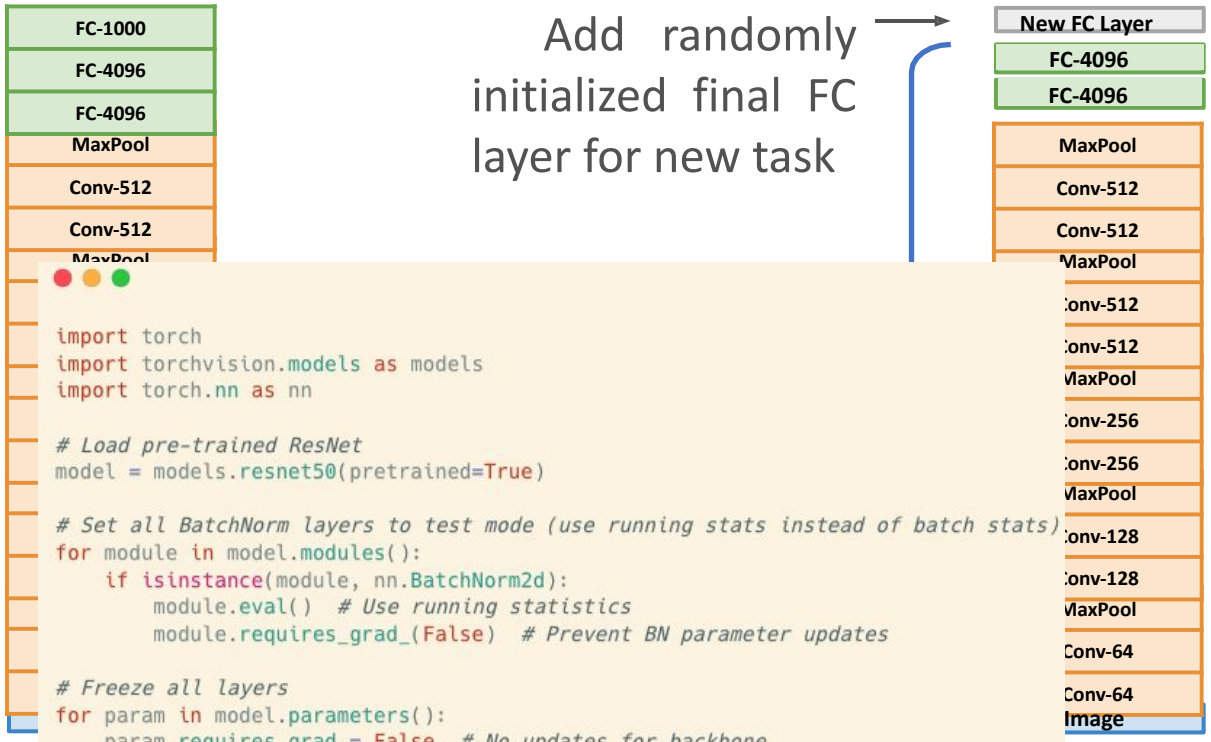


Continue training entire model for new task

```python
import torchvision.models as models
import torch.nn as nn
import torch.optim as optim

# Load pre-trained AlexNet
model = models.alexnet(pretrained=True)

# Replace the last layer in the classifier
num_features = model.classifier[6].in_features  # Get input size of last FC layer
model.classifier[6] = nn.Linear(num_features, 10)  # Replace for 10 output classes

# Unfreeze all layers (train the whole model)
for param in model.parameters():
    param.requires_grad = True  # Allow updates for all layers

# Set different learning rates: lower for pretrained layers, higher for new layer
base_lr = 1e-4  # Lower LR for old layers
new_layer_lr = 1e-3  # Higher LR for new classifier layer

optimizer = optim.Adam([
    {'params': model.features.parameters(), 'lr': base_lr},  # CNN feature extractor
    {'params': model.classifier[:-1].parameters(), 'lr': base_lr},  # Old FC layers
    {'params': model.classifier[6].parameters(), 'lr': new_layer_lr}  # Newly added FC layer
])
```

Transfer Learning: Fine-Tuning

1. Train on ImageNet



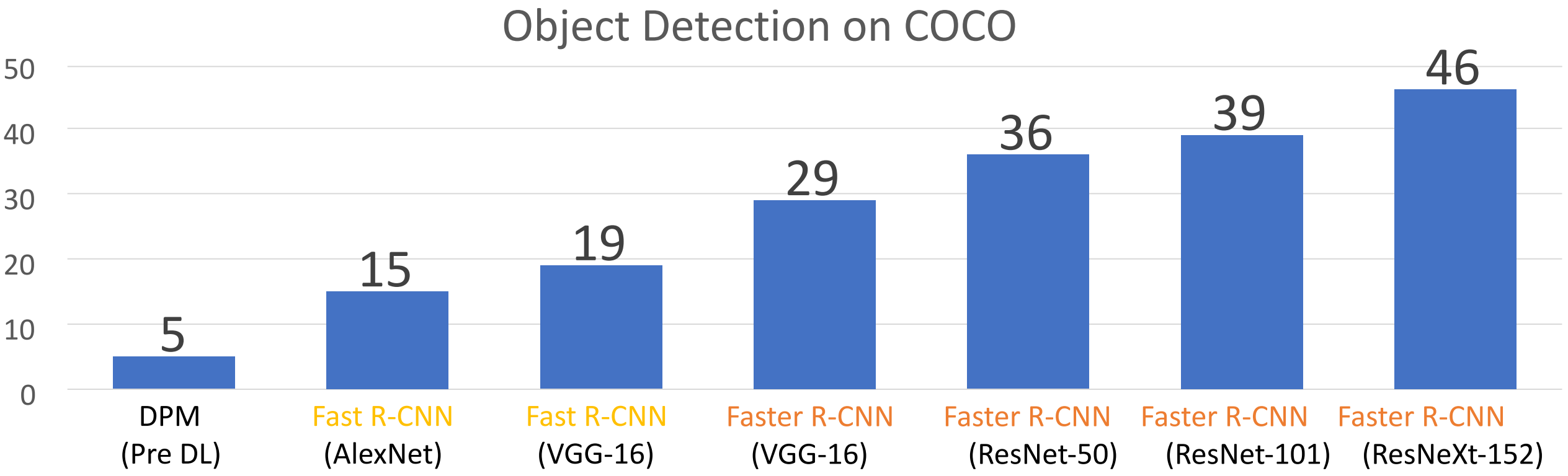

```python
import torch
import torchvision.models as models
import torch.nn as nn

# Load pre-trained ResNet
model = models.resnet50(pretrained=True)

# Set all BatchNorm layers to test mode (use running stats instead of batch stats)
for module in model.modules():
    if isinstance(module, nn.BatchNorm2d):
        module.eval()  # Use running statistics
        module.requires_grad_(False)  # Prevent BN parameter updates

# Freeze all layers
for param in model.parameters():
    param.requires_grad = False  # No updates for backbone

# Replace the last layer (fully connected) for new task
num_features = model.fc.in_features  # Get input size of last FC layer
model.fc = nn.Linear(num_features, 10)  # Modify output layer for 10 classes

# Enable training for only the last layer
for param in model.fc.parameters():
    param.requires_grad = True  # Fine-tune classifier
```

Continue training entire model for new task

Some tricks:

- Train with feature extraction first before fine-tuning
- Lower the learning rate: use ~1/10 of LR used in original training
- Sometimes freeze lower layers to save computation
- Train with BatchNorm in “test” mode

Transfer Learning: Architecture Matters!



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

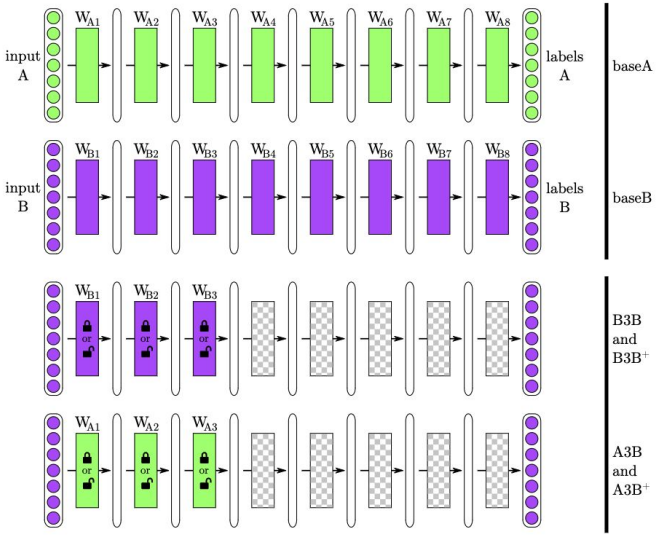
When and how to fine-tune ?

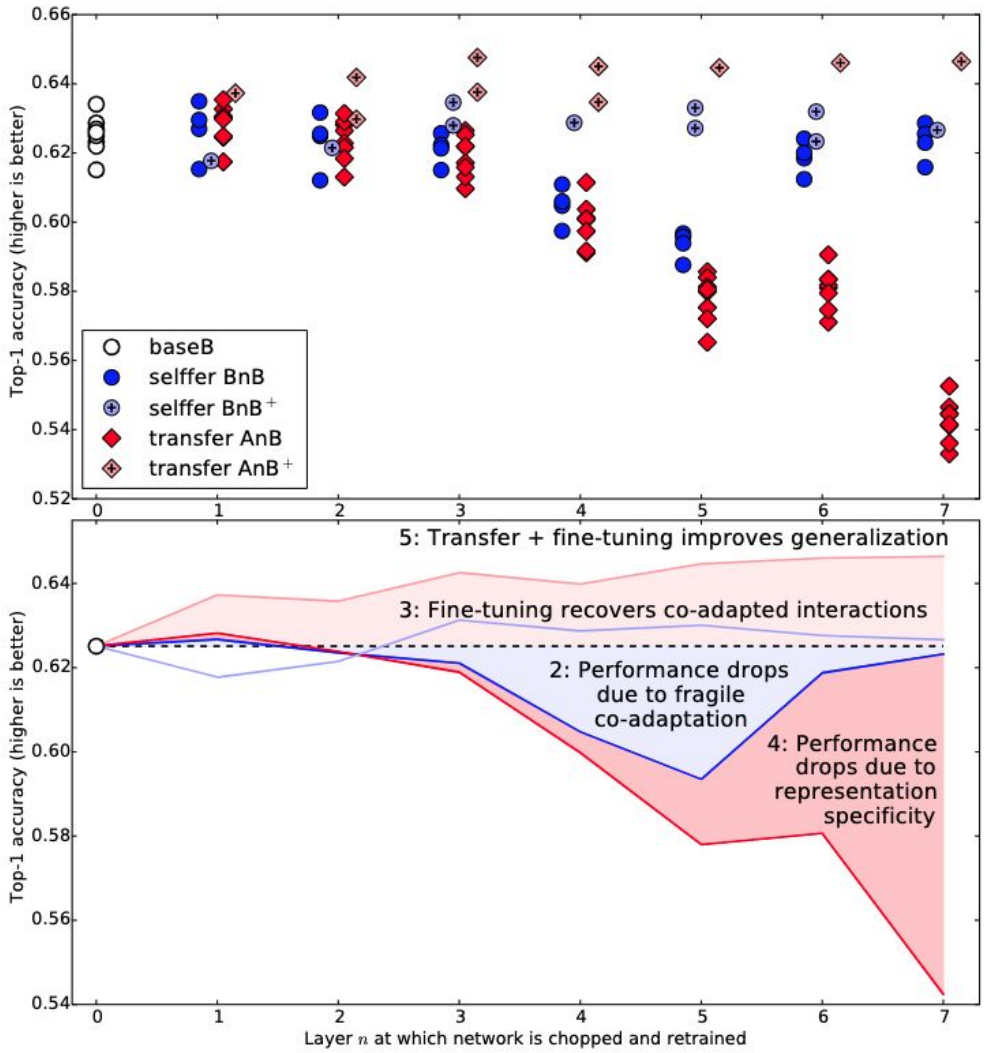
Suppose we have model A, trained on dataset A Q: How do we apply transfer learning to dataset B to create model B?

Dataset size	Dataset similarity	Recommendation
Large	Very different	Train model B from scratch, initialize weights from model A
Large	Similar	OK to fine-tune (less likely to overfit)
Small	Very different	Train classifier using the earlier layers (later layers won't help much)
Small	Similar	Don't fine-tune (overfitting). Train a linear classifier

https://cs231n.github.io/transfer-learning/

Which layers to re-train ?



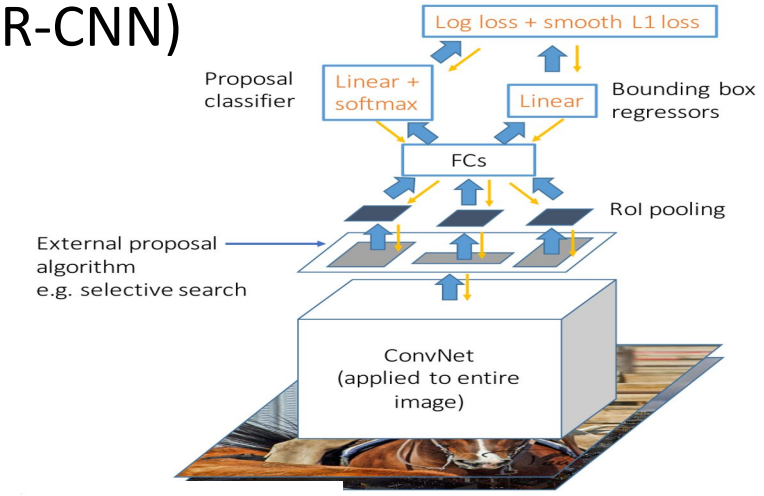


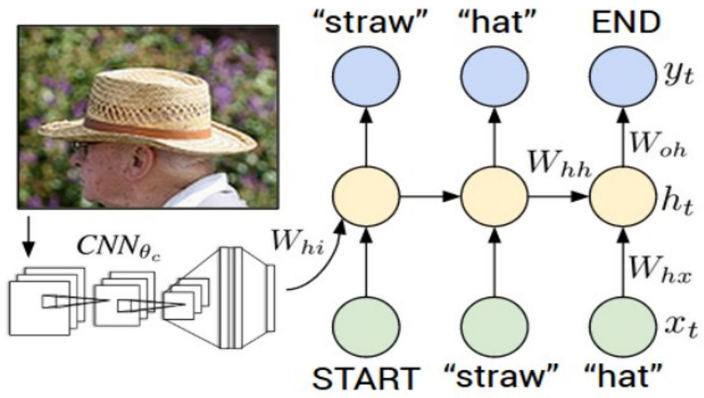
The light red AnB$^+$ diamonds show a particularly surprising effect: that transferring features and then fine-tuning them results in networks that generalize better than those trained directly on the target dataset. Previously, the reason one might want to transfer learned features is to enable training without overfitting on small target datasets, but this new result suggests that transferring features will boost generalization performance even if the target dataset is large. Note that this effect should not be attributed to the longer total training time (450k base iterations + 450k fine-tuned iterations for AnB$^+$ vs. 450k for baseB), because the BnB$^+$ networks are also trained for the same longer length of time and do not exhibit this same performance improvement. Thus, a plausible explanation is that even after 450k iterations of fine-tuning (beginning with completely random top layers), the effects of having seen the base dataset still linger, boosting generalization performance. It is surprising that this effect lingers through so much retraining. This generalization improvement seems not to depend much on how much of the first network we keep to initialize the second network: keeping anywhere from one to seven layers produces improved performance, with slightly better performance as we keep more layers. The average

How transferable are features in deep neural networks? https://arxiv.org/pdf/1411.1792

Transfer learning is pervasive! It's the norm, not the exception

Object Detection
(Fast R-CNN)



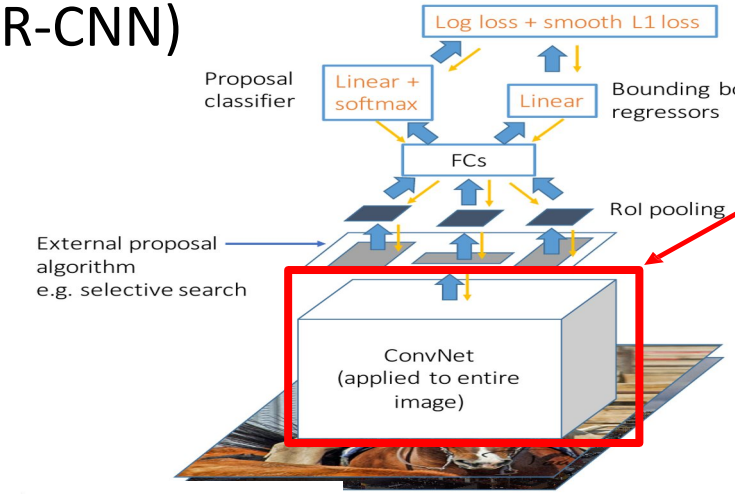


Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

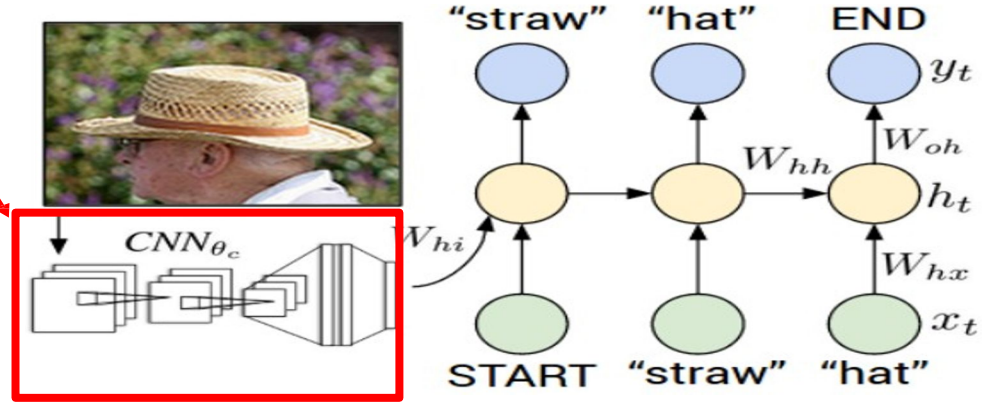
Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015

Transfer learning is pervasive! It's the norm, not the exception

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