Week 3

Convolutional Neural Networks

[ECEA0649/ECE40049] Deep Learning for Image Processing | Spring 2026

Kihyun Na (Research Professor)

BK21 AI Project Group & Institute for Information and Communication Technology,
Handong Global University

HANDONG GLOBAL UNIVERSITY

Curriculum

** Adjusted based on survey results*

Wk 1	OT + Introduction
Wk 2	DL Fundamentals Review
Wk 3	CNN (today)
Wk 4	RNN + Attention Mechanism
Wk 5	Transformer + ViT
Wk 6	Detection + Segmentation
Wk 7	Self-Supervised Learning
Wk 8	Review Literacy + Role Explanation
Wk 9	Foundation Models (CLIP, SAM) + Paper #1
Wk 10	Diffusion Models + Paper #2
Wk 11	Conditional Generation + Paper #3
Wk 12	Vision-Language Models + Paper #4
Wk 13	VLM Applications + Paper #5
Wk 14	Video Understanding + Paper #6
Wk 15	Embodied AI & Robot Vision + Paper #7
Wk 16	Miniconference (Final Project)

Lecture Lecture + Paper Miniconference

Today's Agenda

01

From Pixels to Features

MLP limitations, CNN inductive biases, hand-crafted → learned features

15 min

02

The Convolution Operation

Filters, feature maps, stride/padding, pooling, normalization, separable conv

35 min

03

CNN Architectures

AlexNet → VGG → ResNet → DenseNet/EfficientNet → ConvNeXt

30 min

04

What CNNs Learn & Transfer Learning

Feature visualization, CAM, Grad-CAM, pretrain → fine-tune

10 min

Part 1

From Pixels to Features

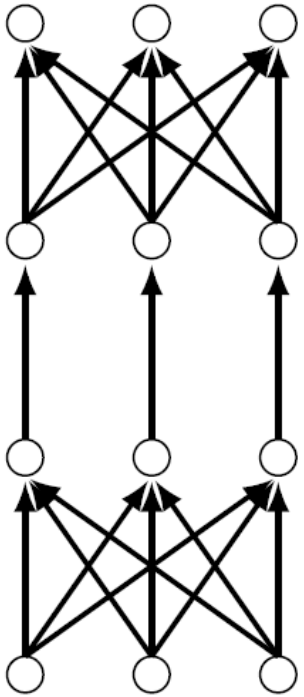
Why fully-connected networks aren't enough for images.

Multilayer Perceptron

linear comb. of neurons ▷

neuron-wise nonlinearity ▷

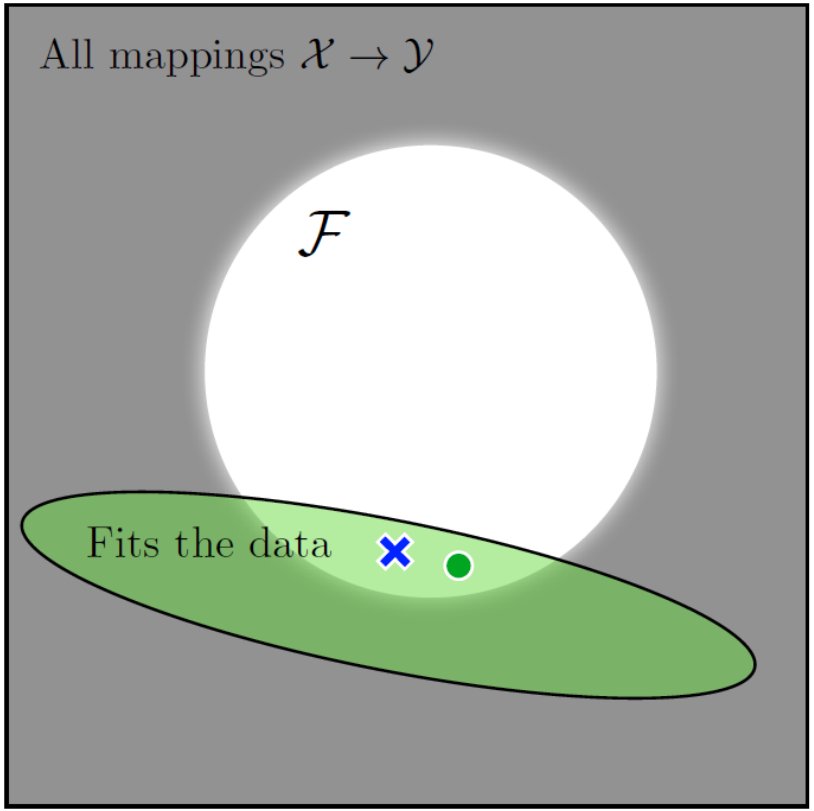
linear comb. of neurons ▷



+ Universal
+ Simple (elegant theory)
+ Embarrassingly parallel

- Weak inductive biases
- Sample inefficient / data hungry
- Dense (fully-connected) linear layers take a lot of compute

Why use other architectures?



$\mathcal{F}$ – Hypothesis space

● True solution

✖ Learned solution

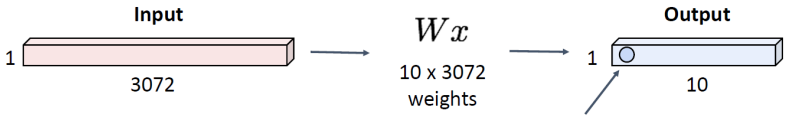
Less data, better architecture.

We can pin down truth *either* by adding more data, or by using a more constrained architecture.

The Problem with FC Layers

Flatten → FC

32x32x3 image -> stretch to 3072 x 1



1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

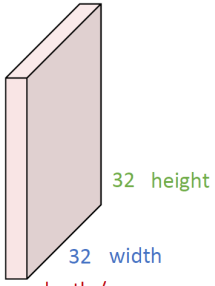
Spatial structure is destroyed.

32×32×3 image → 3,072-dim vector
224×224×3 → 150,528-dim vector

Parameters explode with image size.
No notion of "nearby" pixels.

Solution: Convolution

3x32x32 image



3x5x5 filter

Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

Preserve spatial structure.

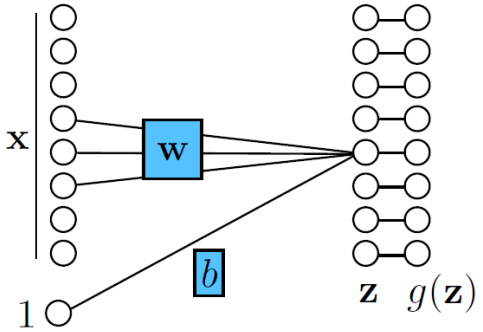
Process local regions with small filters.
Share parameters across the image.
Build hierarchical features.

→ Fewer parameters, better performance.

Why CNNs Work: Inductive Biases

An architecture encodes assumptions about the data. CNN's assumptions match images perfectly.

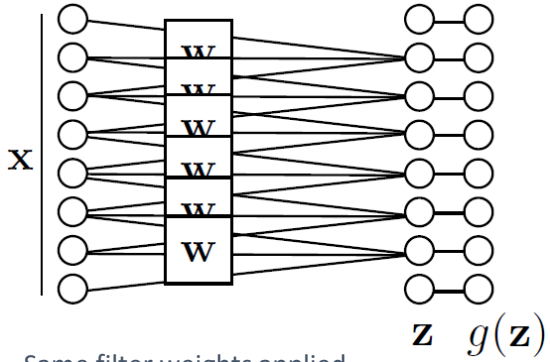
Local Connectivity



Each neuron sees only a small local region (receptive field), not the entire image.

Pixels that are close together are more related than distant ones.

Parameter Sharing



Same filter weights applied across all spatial positions.

A feature detector useful in one part of the image is likely useful elsewhere too.

Translation Equivariance

$$f(\texttt{translate}(x)) = \texttt{translate}(f(x))$$

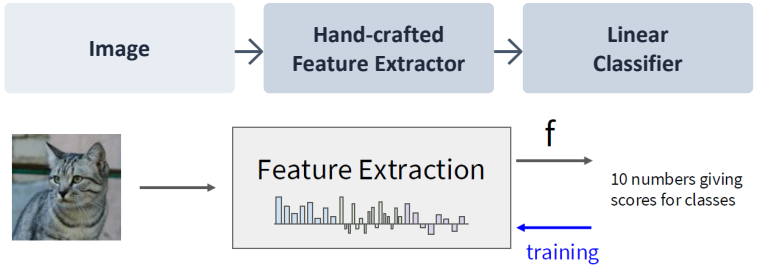
If the input shifts, the output shifts correspondingly.

A cat is a cat regardless of where it appears in the image.

Ref: MIT 6.7960 — "Architecture gives Inductive Bias". This perspective connects CNN → GNN → Transformer.

From Hand-Crafted to Learned Features

Before Deep Learning (~2012)

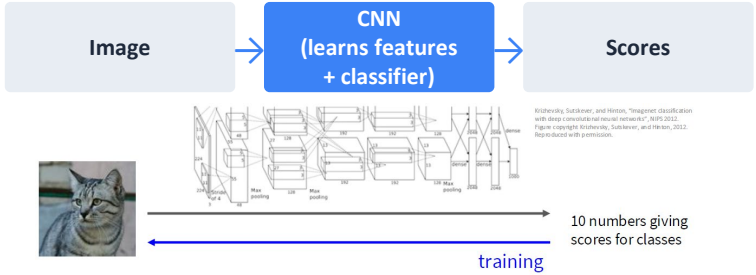


HoG, SIFT, Color Histograms, Bag of Words

Humans design the features.
Classifier only learns the final mapping.

"Feature engineering" was the bottleneck.

With CNNs (2012→)



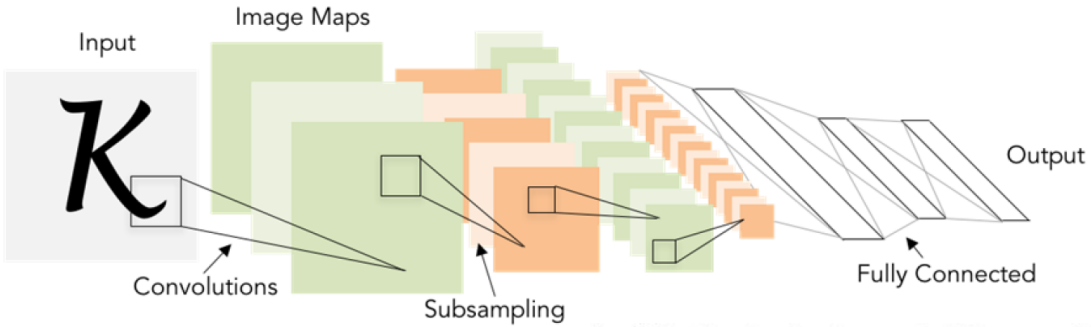
End-to-end learning.

The network learns both features AND the classifier jointly from data.

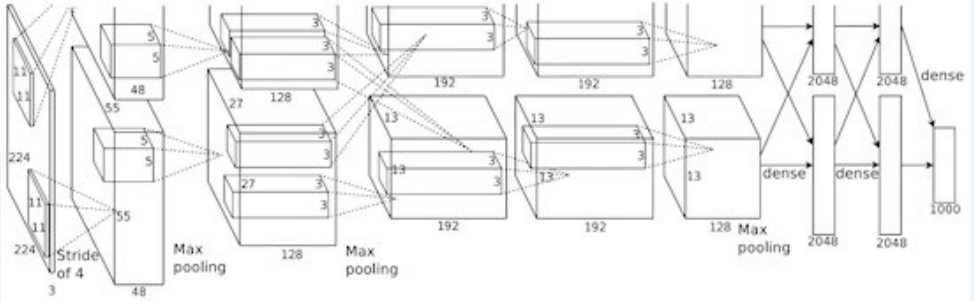
No manual feature design needed.
Backprop optimizes the entire pipeline.

LeNet (1998), AlexNet (2012)

Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]

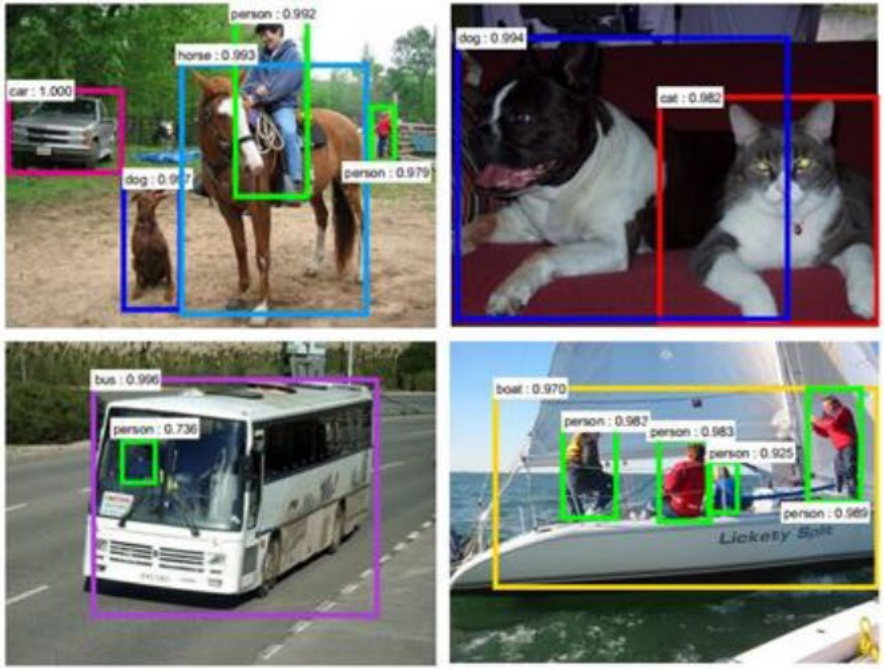


ImageNet Classification with Deep Convolutional Neural Networks
[Krizhevsky, Sutskever, Hinton, 2012]



~2020: Convnets dominate all vision tasks

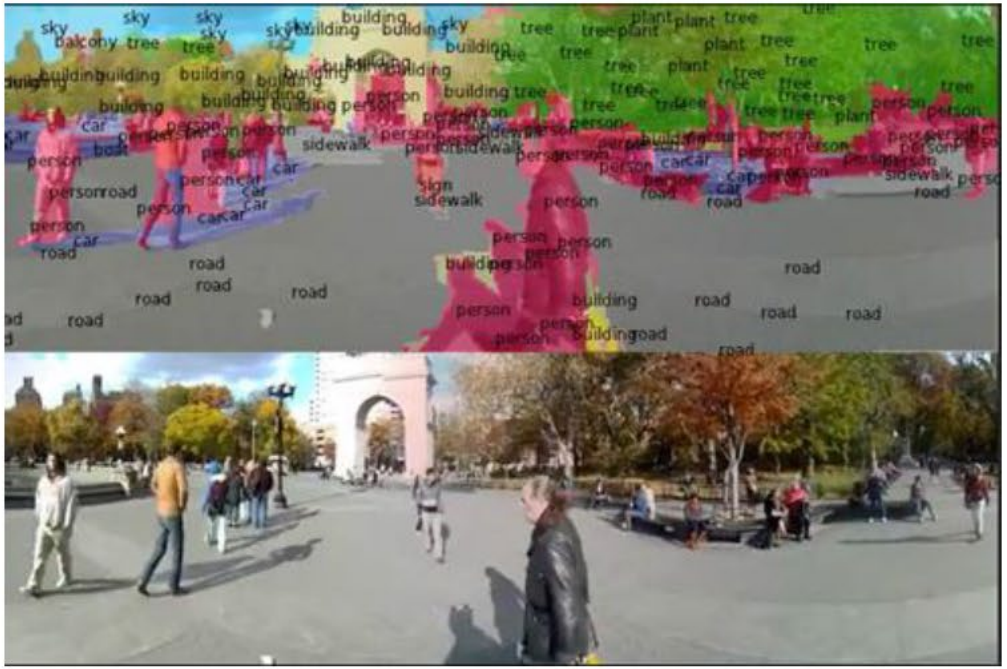
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Detection



Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

~2020: Convnets dominate all vision tasks

Image Captioning

A white teddy bear sitting in the grass

A man in a baseball uniform throwing a ball

A woman is holding a cat in her hand

A man riding a wave on top of a surfboard

A cat sitting on a suitcase on the floor

A woman standing on a beach holding a surfboard

All images are CC0 Public domain:
https://pixabay.com/en/luggage-antique-cat-1643010/
https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/
https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/
https://pixabay.com/en/woman-female-model-portrait-adult-983967/
https://pixabay.com/en/handstand-lake-meditation-496008/
https://pixabay.com/en/baseball-player-shortstop-infield-1045263/

Captions generated by Justin Johnson using Neuraltalk2

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

~2020: Convnets dominate all vision tasks

Text-to-Image Generation

Rombach et al, "High-Resolution Image Synthesis with Latent Diffusion Models", CVPR 2022

A zombie in the style of Picasso

An image of a half mouse half octopus

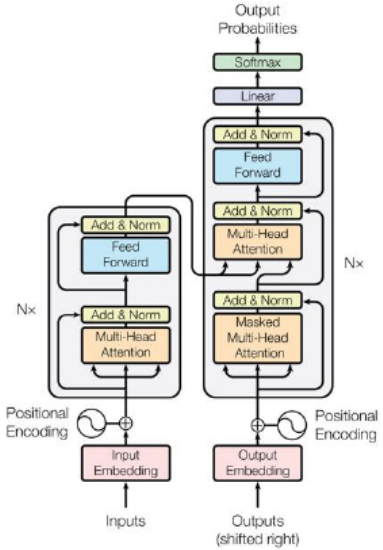
A painting of a squirrel eating a burger

A watercolor painting of a chair that looks like an octopus

A shirt with the inscription: "I love generative models!"

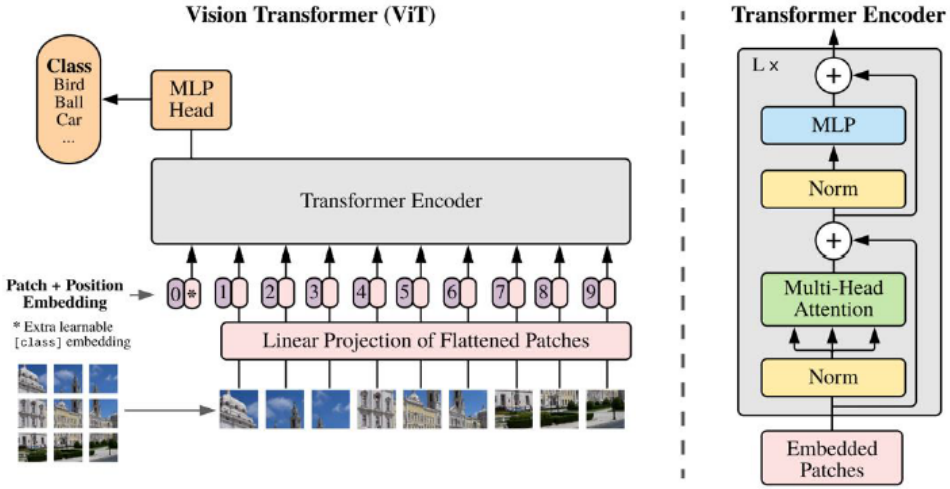
2020~: Transformers have taken over

2017: Transformers for language tasks



Vaswani et al, "Attention is all you need", NeurIPS 2017

2021: Transformers for vision tasks



Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

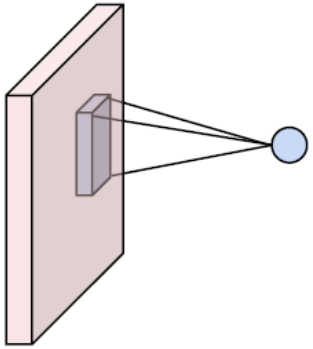
Part 2

The Convolution Operation

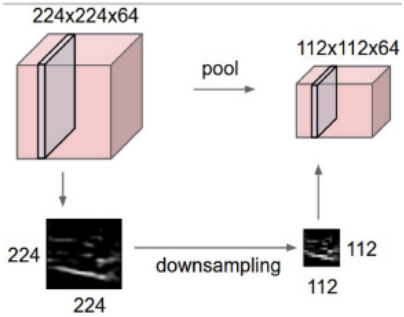
Sliding filters, feature maps, and spatial dimensions.

Components of CNNs

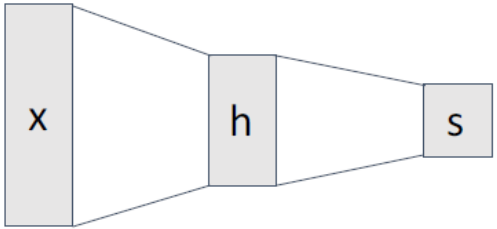
Convolution Layers



Pooling Layers



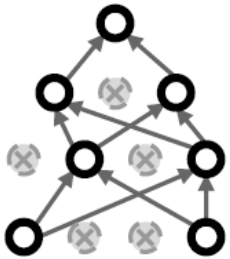
Fully-Connected Layers



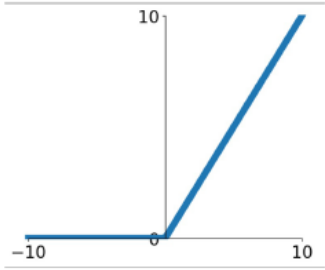
Normalization Layers

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

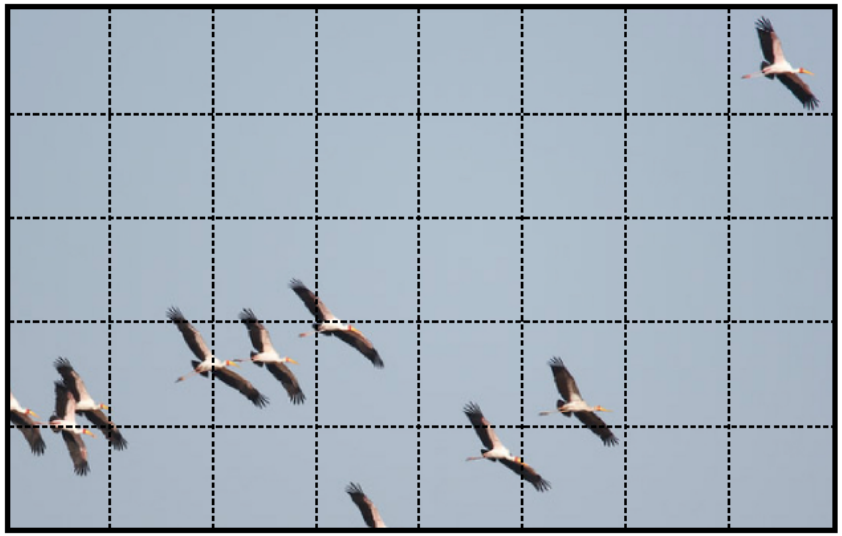
Dropout (sometimes)



Activation Functions



Convolution: Core Idea



f

sky	sky	sky	sky	sky	sky	sky	bird
sky	sky	sky	sky	sky	sky	sky	sky
sky	sky	sky	sky	sky	sky	sky	sky
bird	bird	bird	bird	sky	bird	sky	sky
sky	sky	sky	bird	bird	sky	sky	sky

Problem:

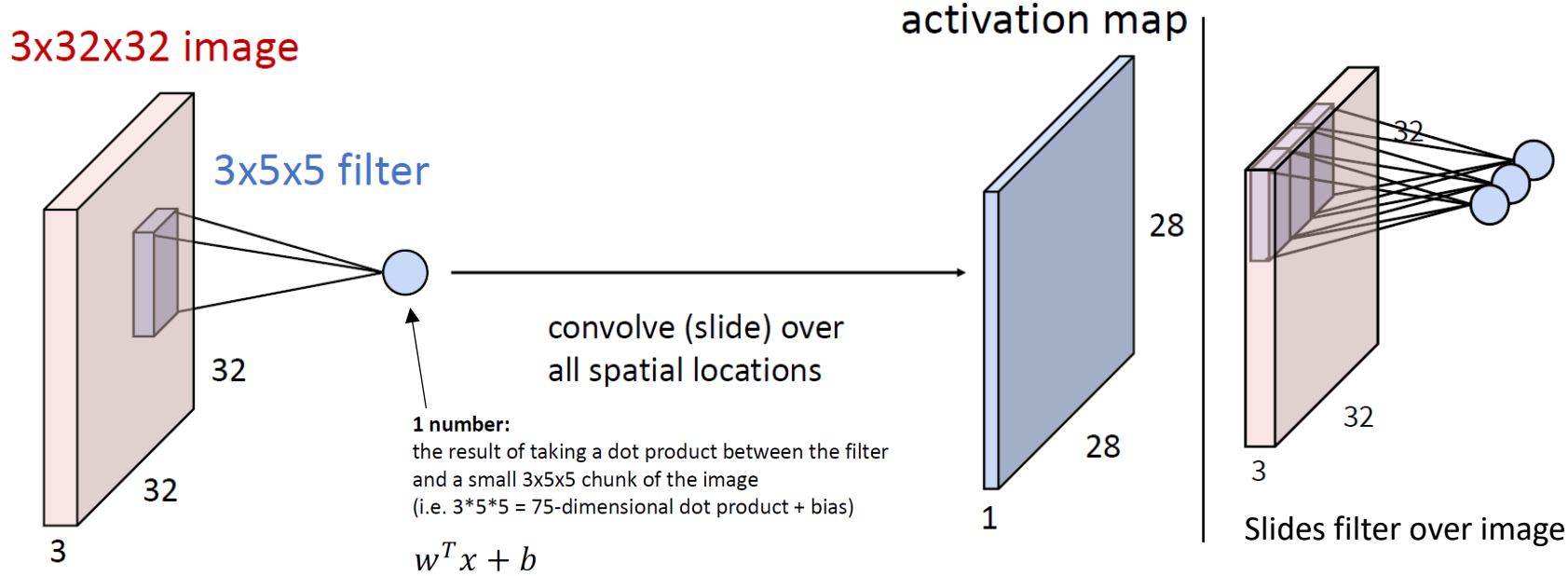
- What if objects don't fit neatly into these patches?
- How to increase the resolution of the output map?

Smaller patches increase resolution, but not easy to recognize content in each small patch

Instead: Use large but overlapping patches

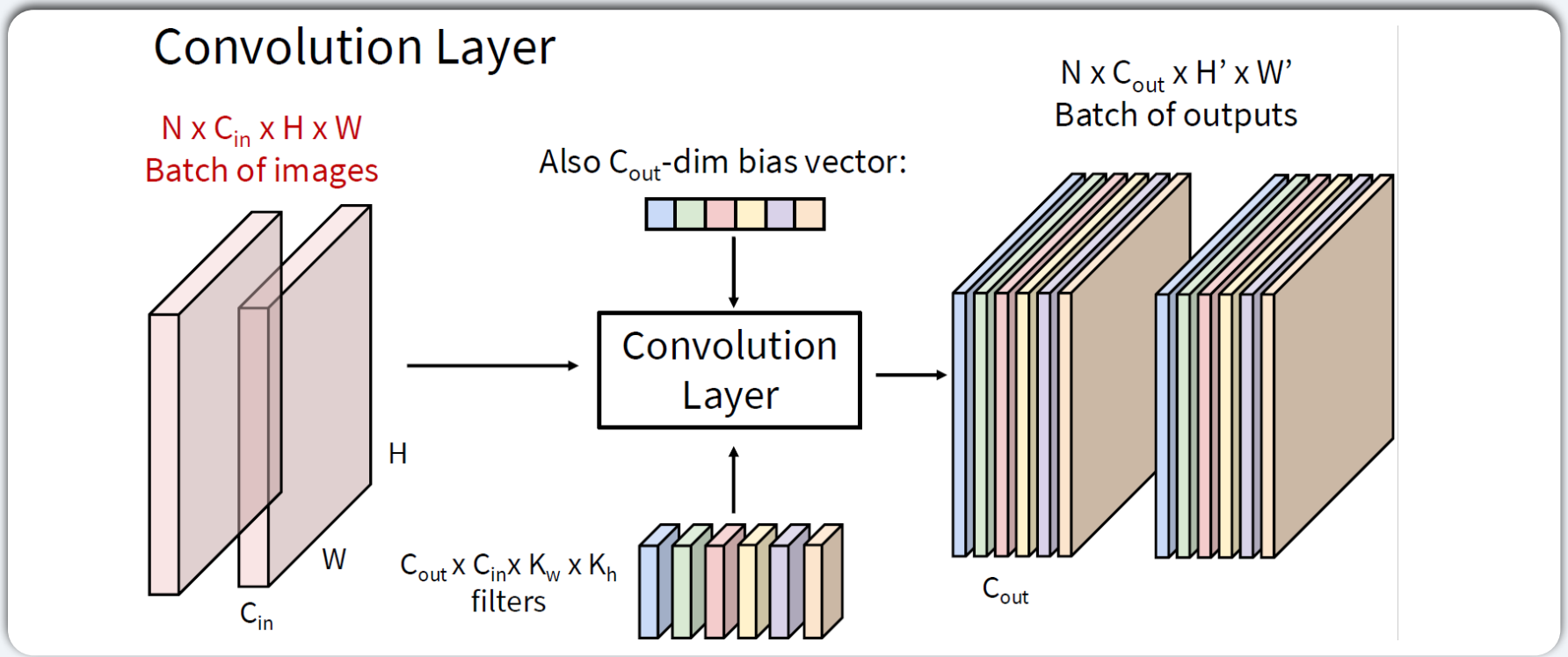
Convolution: Core Idea

Slide a small filter over the image, computing dot products at each position.



Key Filter always extends full depth of input (all channels). One filter → one activation map.
Each position: dot product + bias = one number.

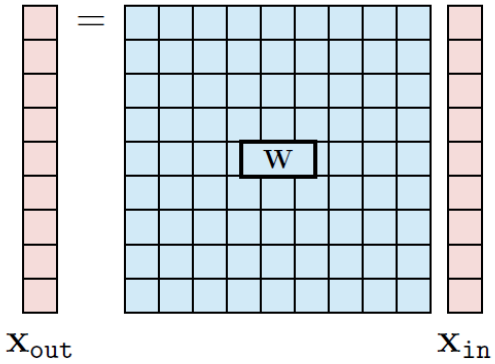
Multiple Filters → Feature Maps



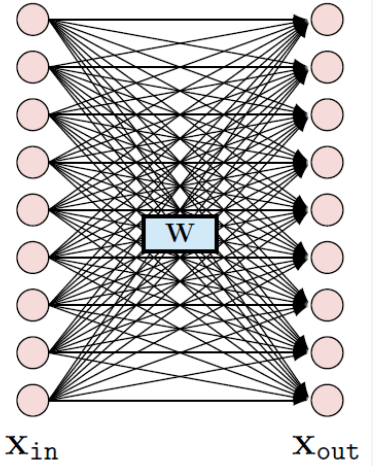
$$Input: N \times C\_in \times H \times W \quad Filters: C\_out \times C\_in \times K \times K \quad Output: N \times C\_out \times H' \times W'$$

Each filter detects a different feature. Output channels = number of filters.

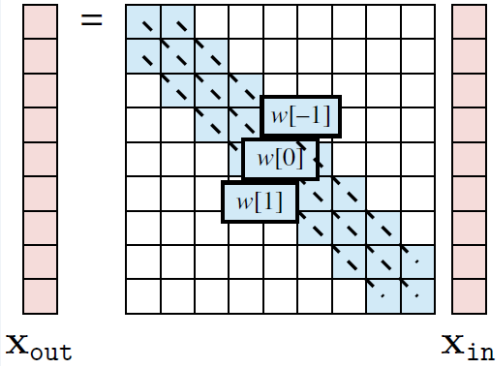
Fully-connected to Locally connected



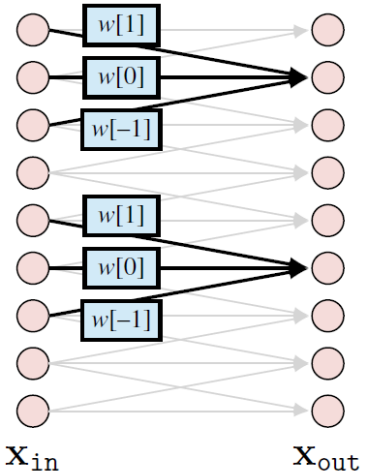
$\Longleftrightarrow$



$$\mathbf{x}_{\text{out}} = \mathbf{W}\mathbf{x}_{\text{in}} + \mathbf{b}$$



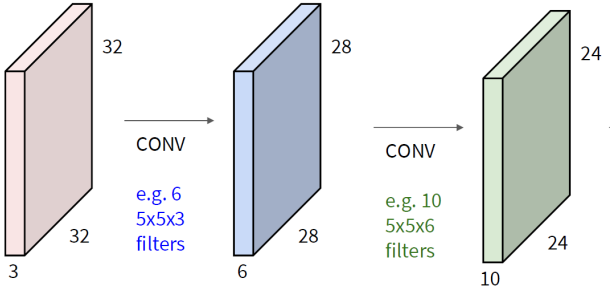
$\Longleftrightarrow$



$$\mathbf{x}_{\text{out}} = \mathbf{w} \star \mathbf{x}_{\text{in}} + b$$

Stacking Convolutions

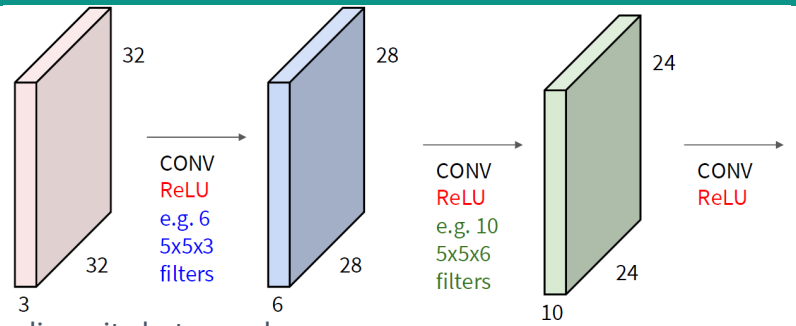
Conv → Conv (no activation)



Two linear operations composed
= still one linear operation!

$$W_2(W_1x) = (W_2W_1)x$$

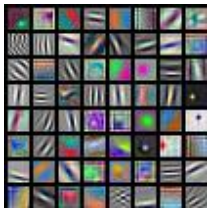
Conv → ReLU → Conv → ReLU

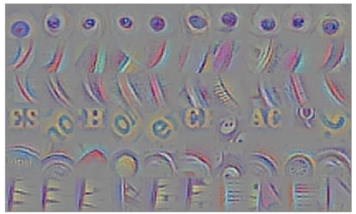


Non-linearity between layers
= can learn complex features!
Each layer builds on the previous.
Deeper = more abstract features.

What do filters learn at each depth?

Linear classifier: One template per class

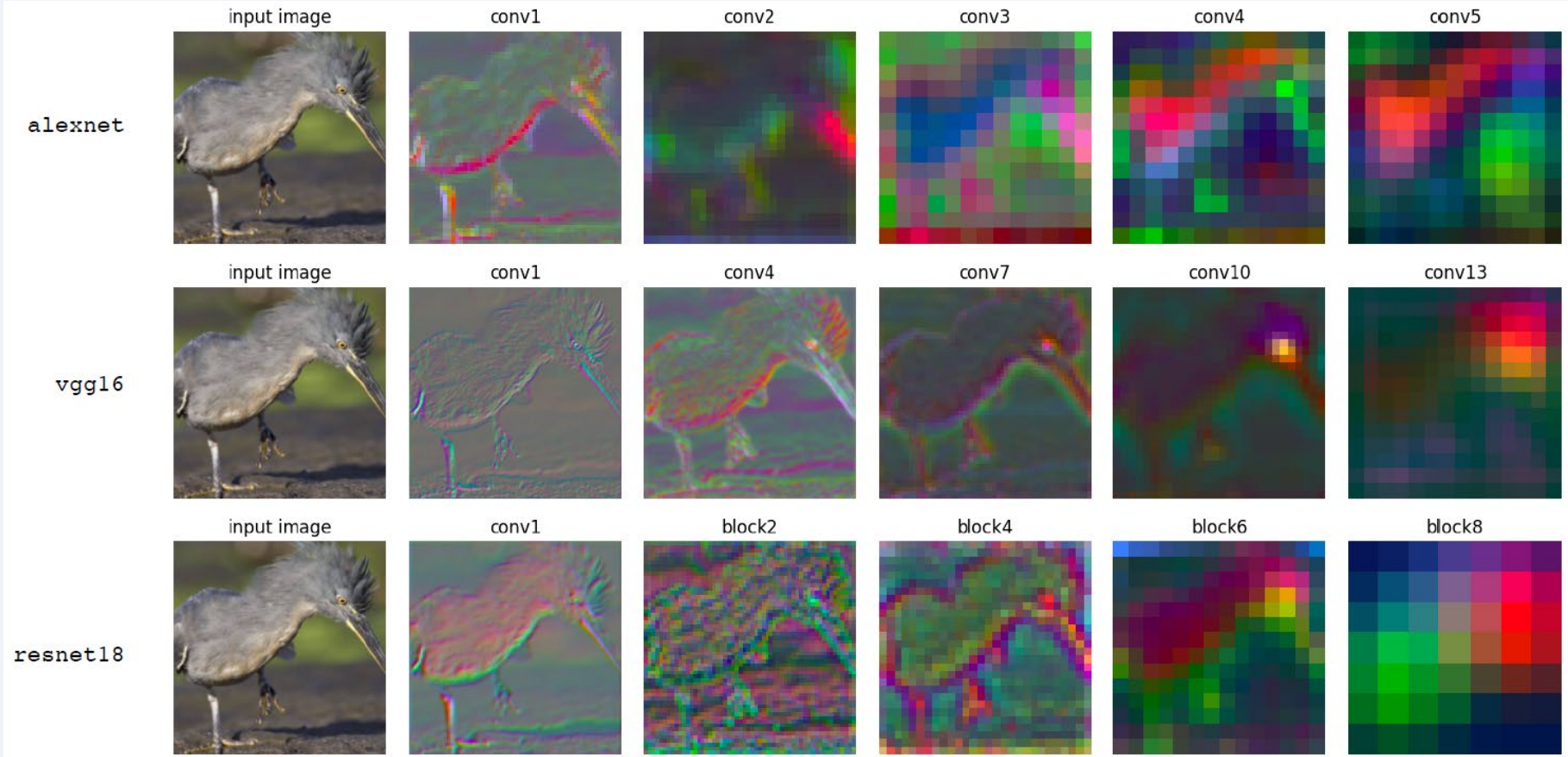




Layer 1 Local image templates (edges, colors)

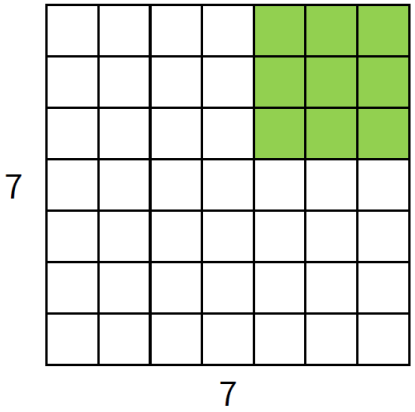
Deep layers Learn larger structures

Stacking Convolutions



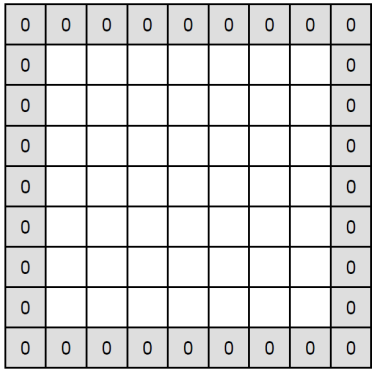
Spatial Dimensions: Padding

$$Output\ Size = W - K + 2P + 1$$



Input: 7x7
Filter: 3x3
Output: 5x5

In general
Input: W
Filter: K
Output: W – K + 1



Input: 7x7
Filter: 3x3
Output: 5x5

Problem: Feature maps shrink with each layer!

Solution: Add **padding** around the input before sliding the filter

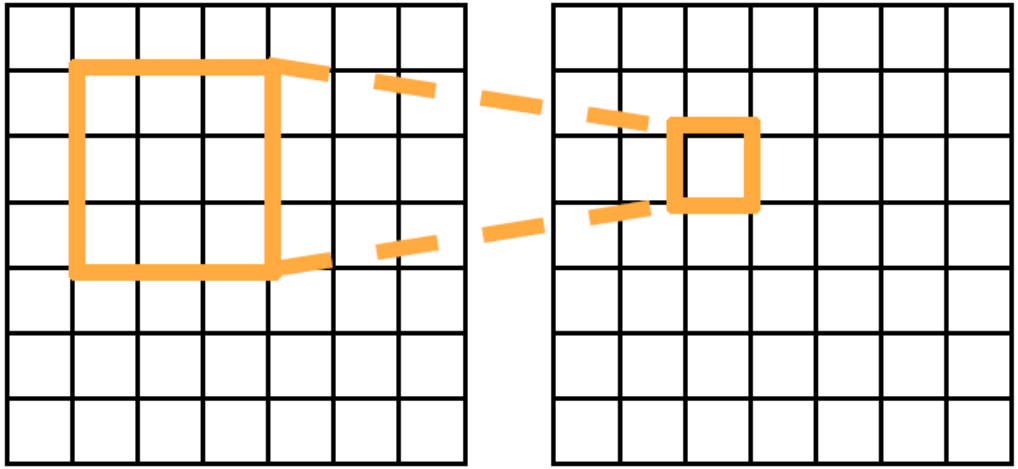
In general
Input: W
Filter: K
Padding: P
Output: W – K + 1 + 2P

$W = Input\ spatial\ size, \quad K = Filter\ (kernel) size, \quad S = Stride\ (step\ size\ of\ filter)$

Common setting:
P = (K – 1) / 2
Means output has same size as input

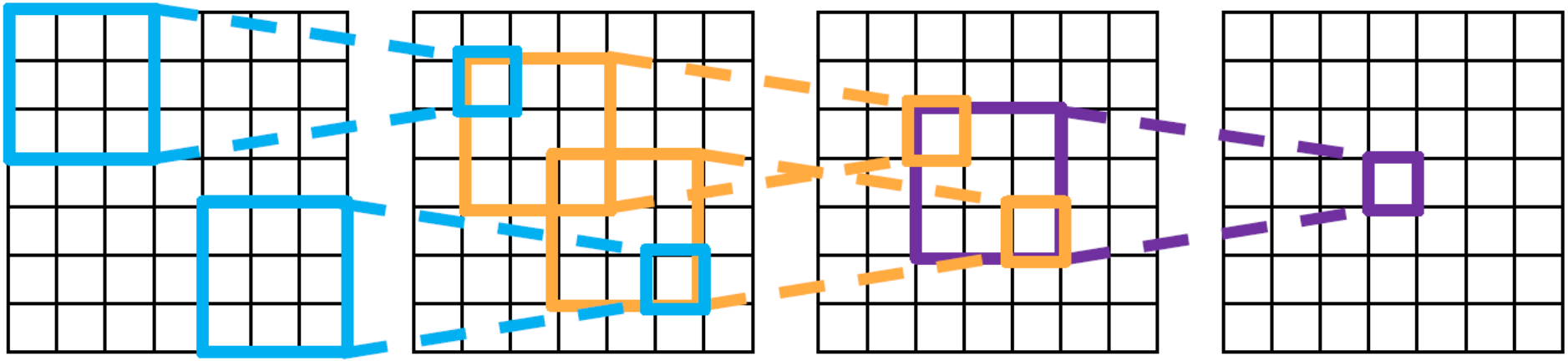
Receptive Fields

For convolution with **kernel size K**, each element in the output depends on a K x K receptive field in the input



Receptive Fields

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



Input

Output

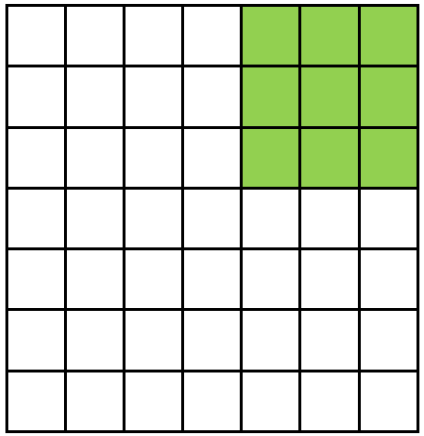
Problem: For large images we need many layers for each output to "see" the whole image image

Solution: Downsample inside the network

Slide inspiration: Justin Johnson

Spatial Dimensions: Stride

$$Output\ Size = (W - K + 2P) / S + 1$$



Input: 7x7
Filter: 3x3
Stride: 2
Output: 3x3

In general:
Input: W
Filter: K
Padding: P
Stride: S

→ Output:
(W – K + 2P) / S + 1

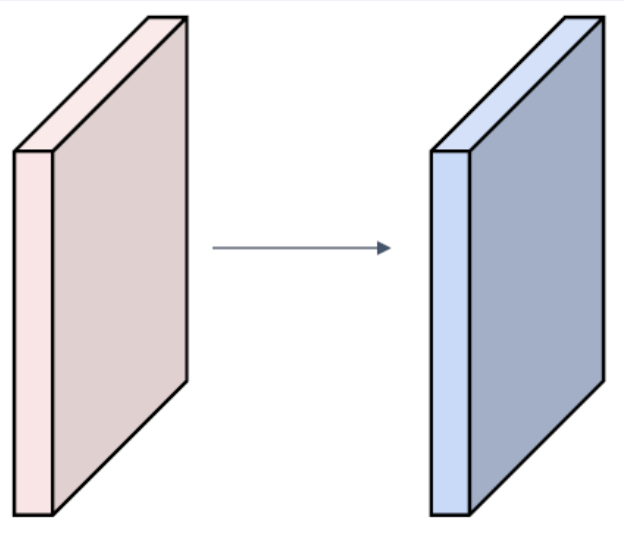
$W = Input\ spatial\ size,$ $K = Filter\ (kernel) size,$ $P = Padding,$ $S = Stride\ (step\ size\ of\ filter)$

Convolution Example: Conv2d

Input volume: 3 x 32 x 32

10 5x5 filters with stride 1, pad 2

Output volume size: ?



Convolution Example: Conv2d

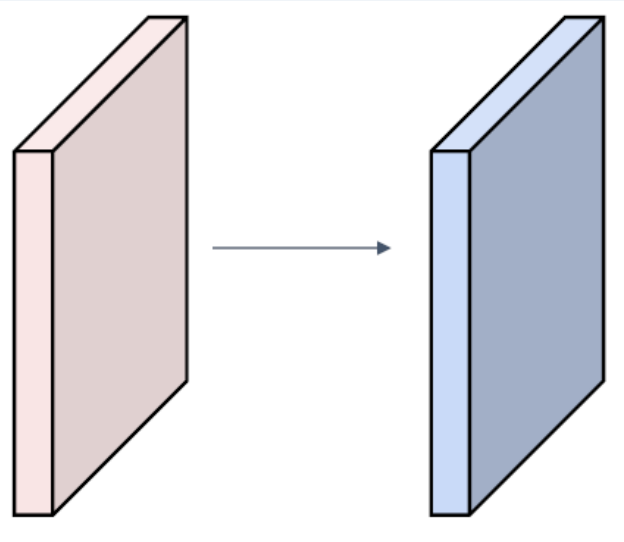
Input volume: 3 x 32 x 32

10 5x5 filters with stride 1, pad 2

Output volume size:

(32+2*2-5)/1+1 = 32 spatially, so

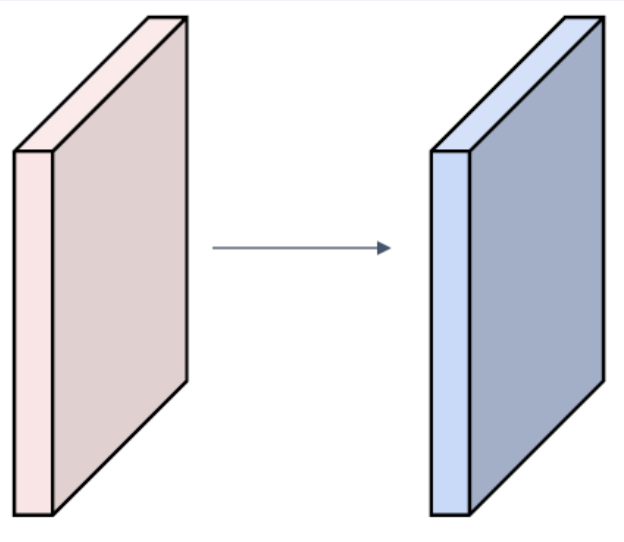
10 x 32 x 32



Convolution Example: Conv2d

Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

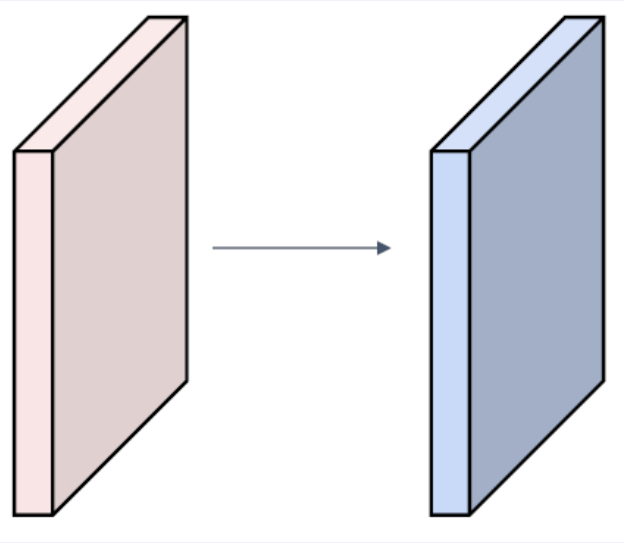
Output volume size: 10 x 32 x 32
Number of learnable parameters: ?



Convolution Example: Conv2d

Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

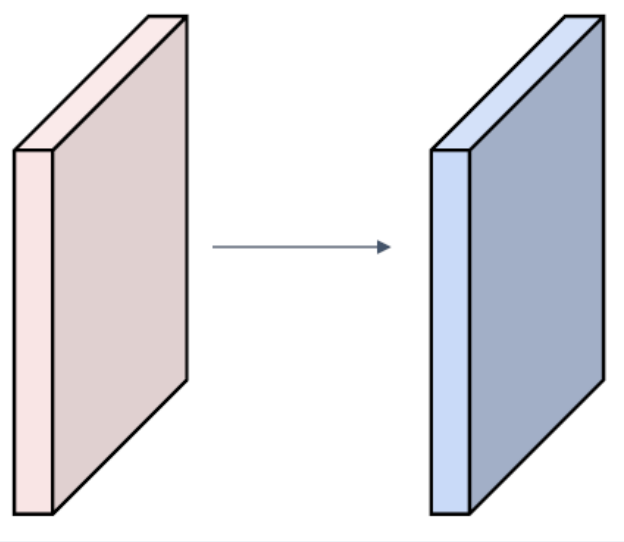
Output volume size: 10 x 32 x 32
Number of learnable parameters: **760**
Parameters per filter: 3*5*5 + 1 (for bias) = 76
10 filters, so total is 10 * 76 = **760**



Convolution Example: Conv2d

Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

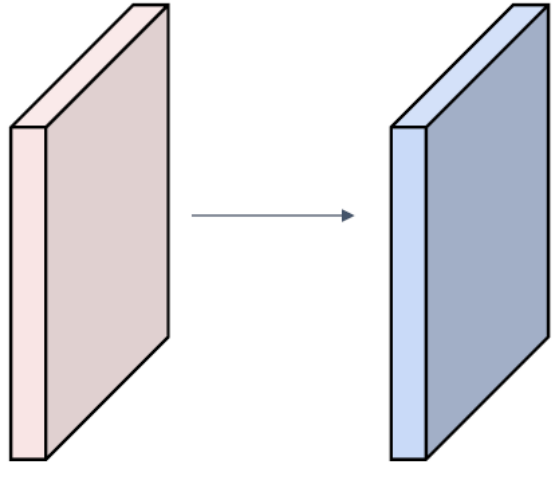
Output volume size: 10 x 32 x 32
Number of learnable parameters: 760
Number of multiply-add operations: ?



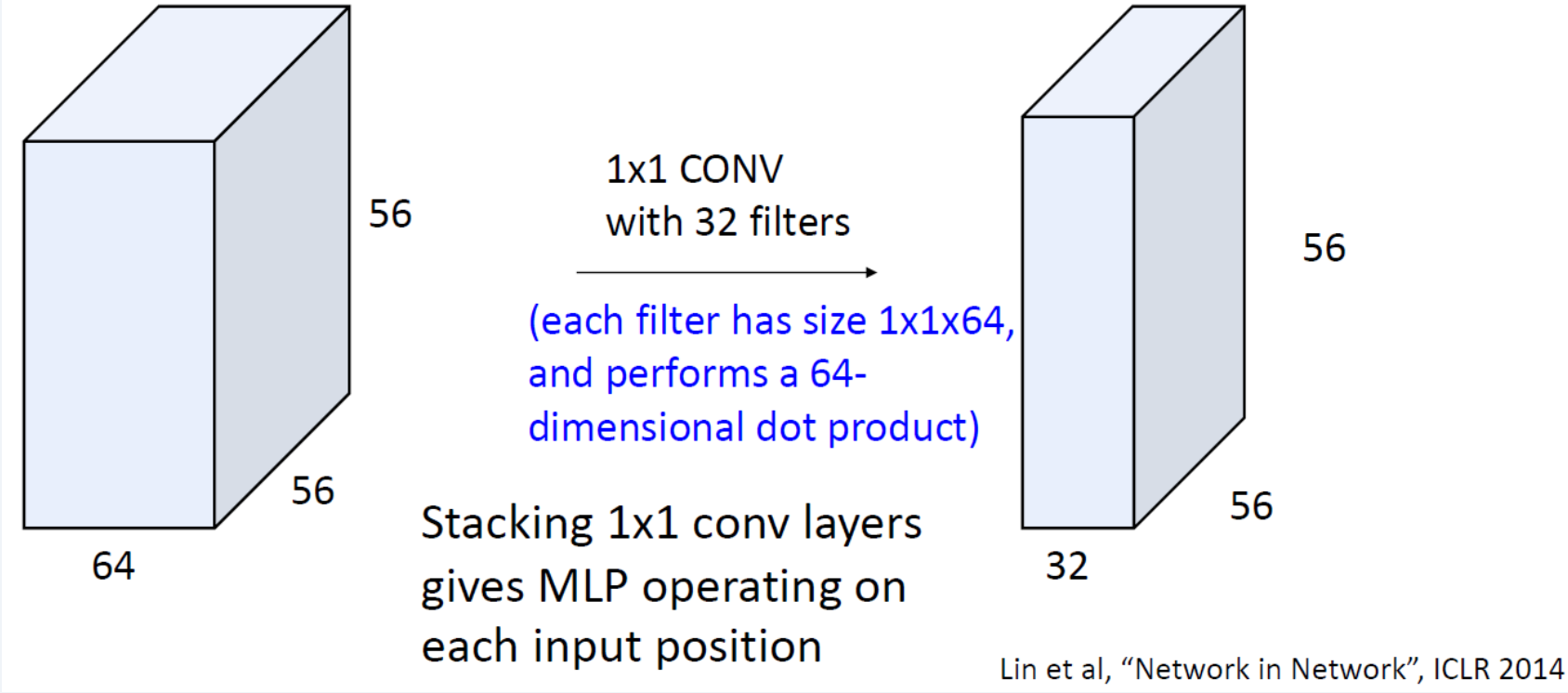
Convolution Example: Conv2d

Input volume: **3** x 32 x 32
10 **5x5** filters with stride 1, pad 2

Output volume size: **10 x 32 x 32**
Number of learnable parameters: 760
Number of multiply-add operations: **768,000**
10*32*32 = 10,240 outputs; each output is the inner product of two **3x5x5** tensors (75 elems); total = 75*10240 = **768K**



Convolution Example: 1x1 Convolution



Lin et al, "Network in Network", ICLR 2014

Convolution Summary

Input: C_{in} x H x W

Hyperparameters:

- **Kernel size**: K_H x K_W
- **Number filters**: C_{out}
- **Padding**: P
- **Stride**: S

Weight matrix: C_{out} x C_{in} x K_H x K_W
giving C_{out} filters of size C_{in} x K_H x K_W

Bias vector: C_{out}

Output size: C_{out} x H' x W' where:

- H' = (H – K + 2P) / S + 1
- W' = (W – K + 2P) / S + 1

Common settings:

$K_H = K_W$ (Small square filters)

P = (K – 1) / 2 ("Same" padding)

C_{in}, C_{out} = 32, 64, 128, 256 (powers of 2)

K = 3, P = 1, S = 1 (3x3 conv)

K = 5, P = 2, S = 1 (5x5 conv)

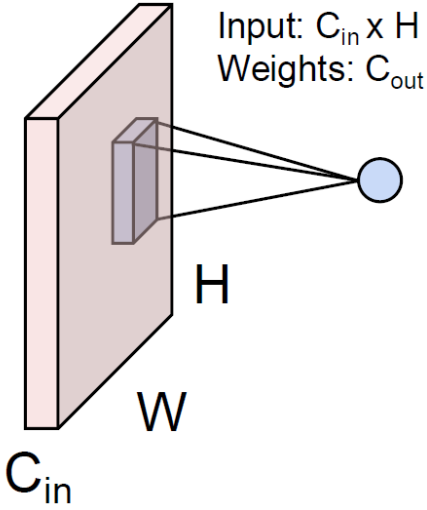
K = 1, P = 0, S = 1 (1x1 conv)

K = 3, P = 1, S = 2 (Downsample by 2)

Convolution Summary

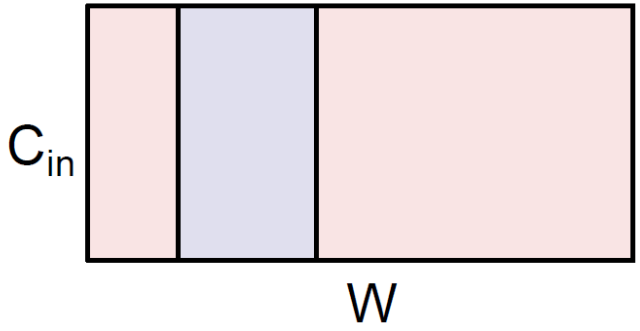
So far: 2D Convolution

Input: C_{in} x H x W
Weights: C_{out} x C_{in} x K x K



1D Convolution

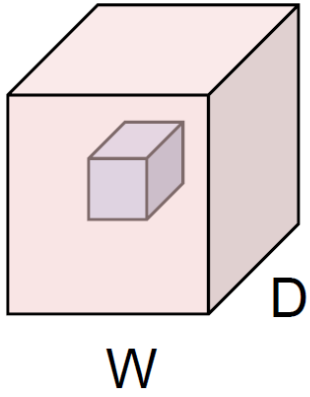
Input: C_{in} x W
Weights: C_{out} x C_{in} x K

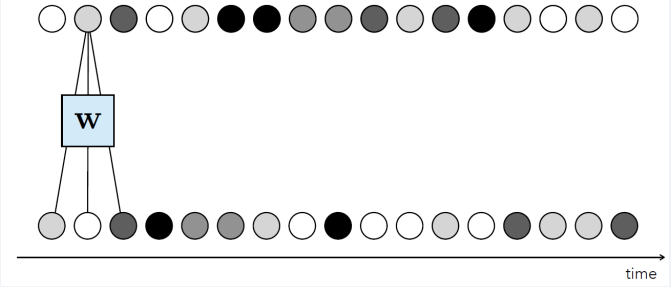


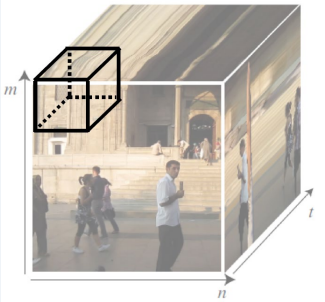
C_{in}-dim vector at each point in the volume

3D Convolution

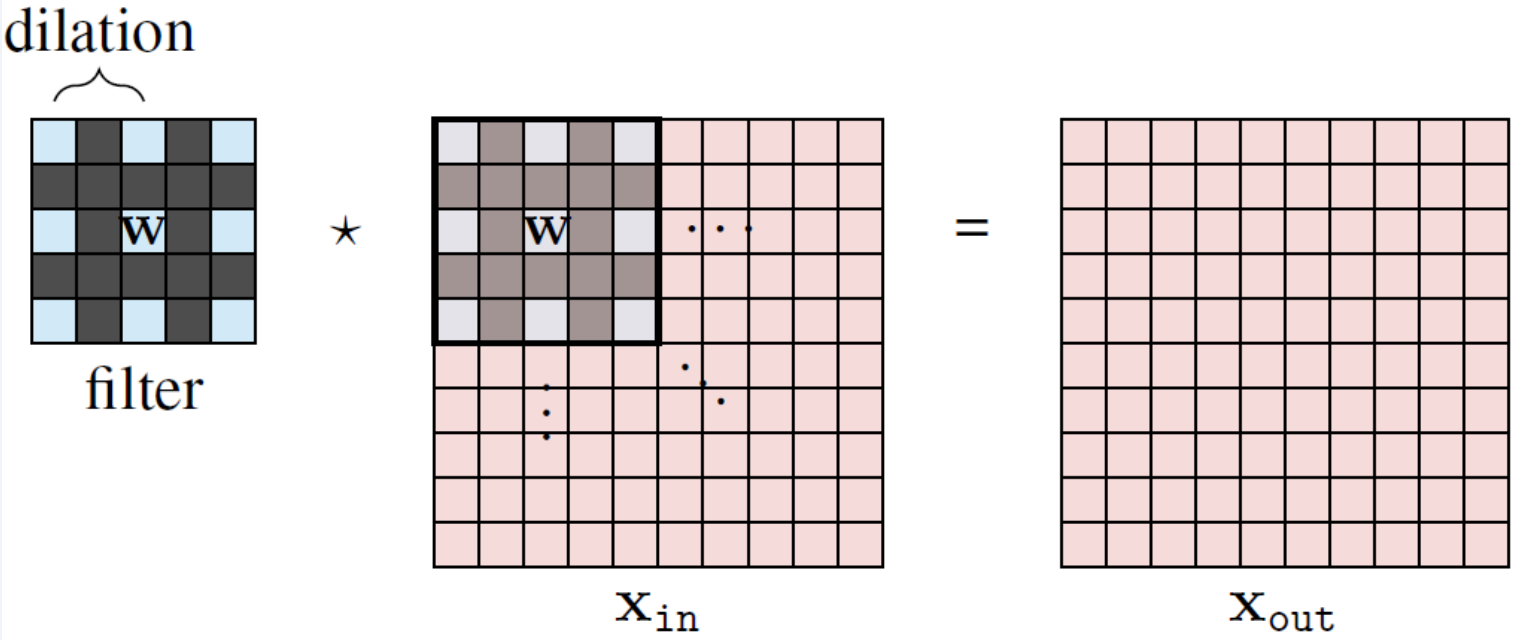
Input: C_{in} x H x W x D
Weights: C_{out} x C_{in} x K x K x K





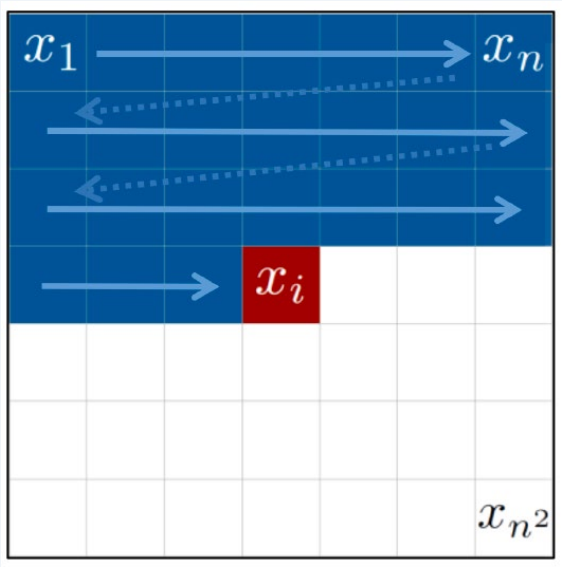


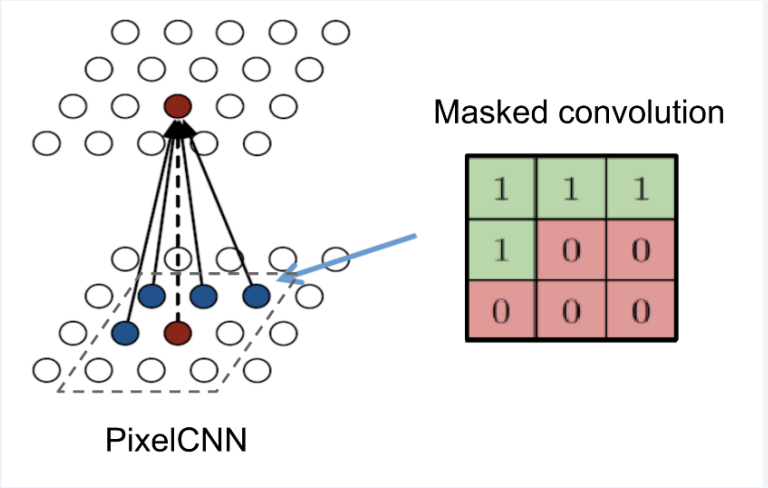
Dilated Convolution



Covers a large receptive field with fewer parameters.

Masked Convolution





Raster Scan Order

- Pixels generated left→right, top→bottom
- $p(x) = \prod p(x_i \,|x_1, \ldots, x_{i-1})$

PixelCNN with Masked Convolution

- Conv filter masked to see only previous pixels
- Zero out future pixel weights in the kernel
- Enables parallel training via convolutions

Separable Convolution

Factor a standard conv into two cheaper operations. Used in MobileNet, EfficientNet, ConvNeXt.

Standard Convolution

Input
$C_{in} \times H \times W$

→

Output
$C_{out} \times H' \times W'$

Cost: $C_{in} \times C_{out} \times K^2 \times H' \times W'$

Depthwise Separable Convolution

Step 1: Depthwise Conv

Each channel gets its own $K \times K$ filter.
No mixing between channels.

C_{in} filters, each $1 \times K \times K$
→ Output: $C_{in} \times H' \times W'$

Spatial info only

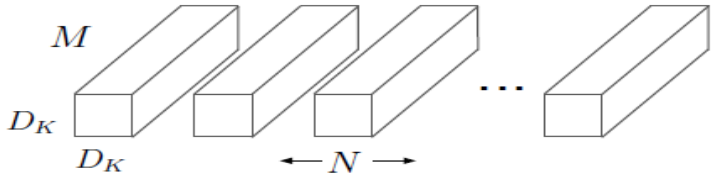
→

Step 2: Pointwise Conv (1×1)

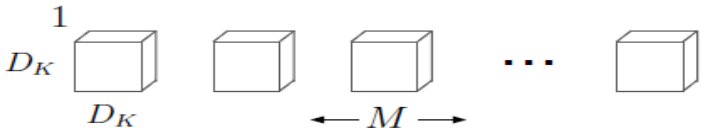
1×1 conv mixes information across channels.

C_{out} filters, each $C_{in} \times 1 \times 1$
→ Output: $C_{out} \times H' \times W'$

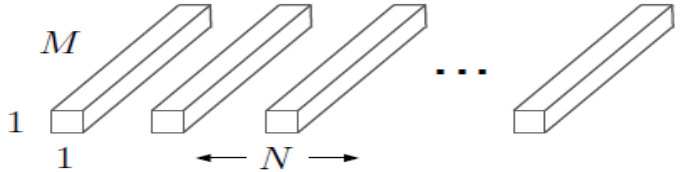
Channel info only



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters

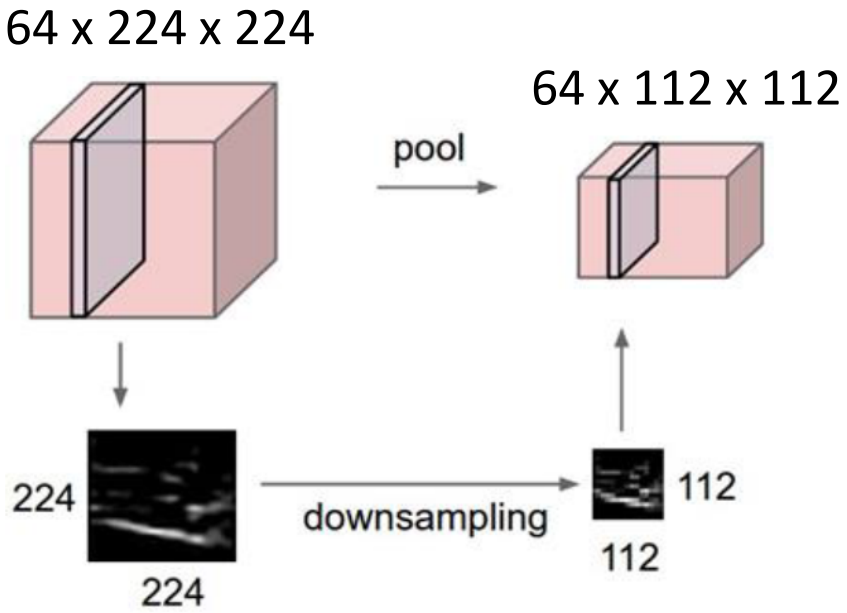


(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Cost reduction Standard: $C_{in} \cdot C_{out} \cdot K^2$ vs Separable: $C_{in} \cdot K^2 + C_{in} \cdot C_{out}$ → **~8–9× cheaper** for $K = 3, C_{out} = 256$

Pooling Layers

Downsample feature maps to reduce computation and build translation invariance.



Given an input C x H x W,
downsample each 1 x H x W plane

Hyperparameters:
Kernel Size
Stride
Pooling function

Pooling Layers

Downsample feature maps to reduce computation and build translation invariance.

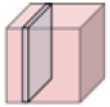
Single depth slice

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

x

y

64 x 224 x 224



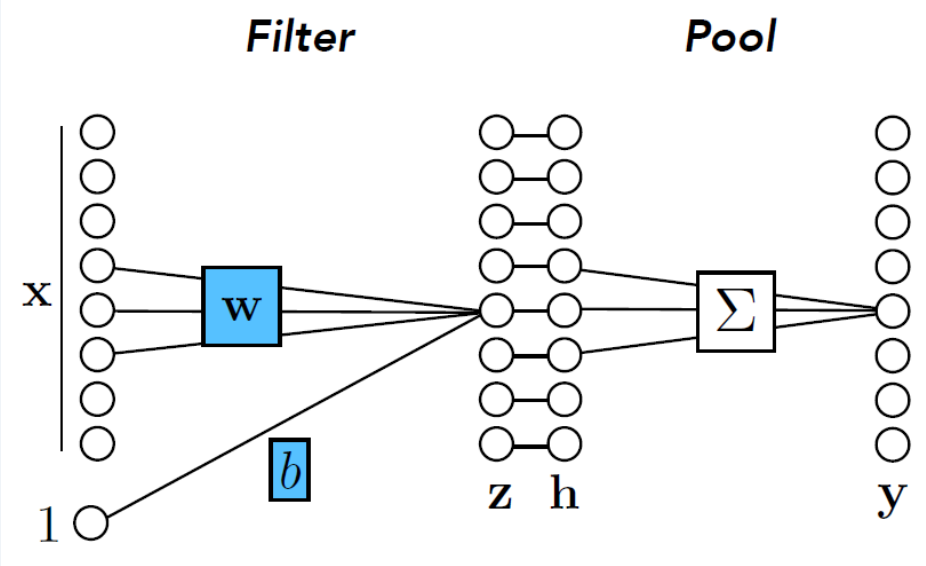
Max pooling with 2x2 kernel size and stride 2

6	8
3	4

Gives **invariance** to small spatial shifts. No learnable parameters.

Pooling Layers

Downsample feature maps to reduce computation and build translation invariance.



Average Pooling

$$y_j = \frac{1}{|\mathcal{N}|} \sum_{j \in \mathcal{N}(j)} h_j$$

Takes the mean value in each window.

Global Average Pooling (GAP) is used at the end of modern architectures → replaces FC layer before classifier.

Max Pooling

$$y_j = \max_{j \in \mathcal{N}(j)} h_j$$

Take the maximum value in each window.
Most common: 2×2, stride 2
→ halves spatial size.
Retains strongest activations.

Note Pooling has no learnable parameters. It's a fixed operation. Some modern architectures skip pooling entirely, using strided convolutions instead.

Normalization Layers

Stabilize training by normalizing activations. The standard CNN block: Conv → BN → ReLU.

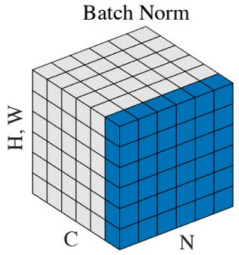
Batch Normalization

CNNs (standard)

Normalize across batch for each channel.

$$y = \gamma \hat{x} + \beta \hat{x} = \frac{(x - \mu_B)}{\sigma_B}$$

Stabilizes training.
Allows higher learning rates.
Slight regularization effect.



Depends on batch size.
Behavior differs train vs test.

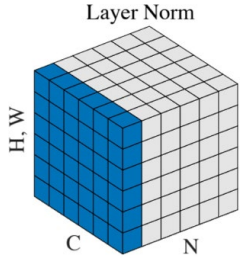
Layer Normalization

Transformers (standard)

Normalize across all channels for each sample.

$$y = \gamma \hat{x} + \beta \hat{x} = \frac{(x - \mu_L)}{\sigma_L}$$

Independent of batch size.
Works with variable-length sequences.



Default in Transformer / ViT.
We'll revisit in Week 5.

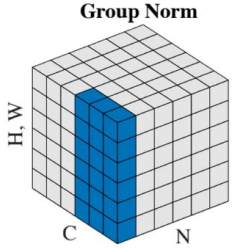
Group Normalization

Detection / Segmentation

Normalize across groups of channels per sample.

Groups, normalize each.
Split C channels into G

Robust to small batches.
Useful when batch=1–2 (e.g., high-res images).



Used in Mask R-CNN, etc.
We'll see this in Week 6.

Batch Normalization to Layer Normalization

Layer Normalization

Batch Normalization for
fully-connected networks

$$x : N \times D$$

Normalize ↓

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

$$\gamma, \beta : 1 \times D$$

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

Layer Normalization for fully-connected networks
Same behavior at train and test!
Used in RNNs, Transformers

$$x : N \times D$$

Normalize ↓

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

$$\gamma, \beta : 1 \times D$$

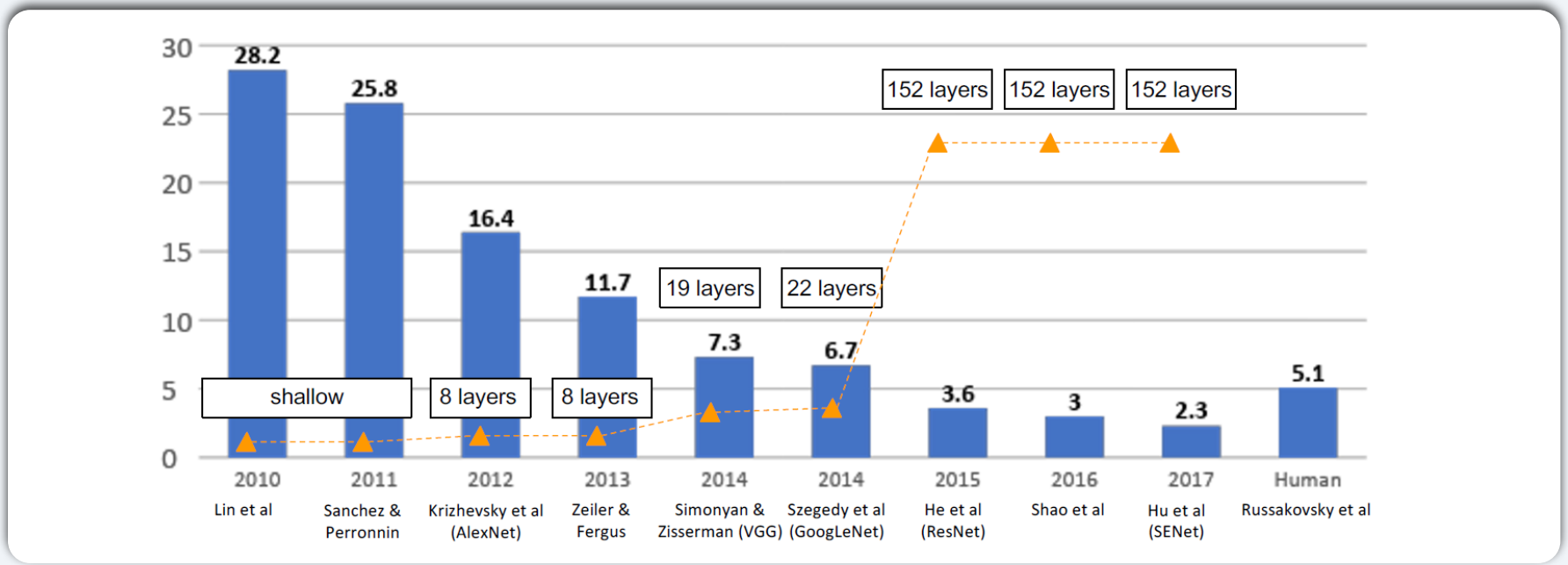
$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Part 3

CNN Architectures

From AlexNet to ResNet — the ideas that shaped modern vision.

ImageNet & The ILSVRC Challenge



2012

AlexNet

CNN revolution

2014

VGG

Deeper, smaller filters

2014

GoogLeNet

Inception module

2015

ResNet

Skip connections

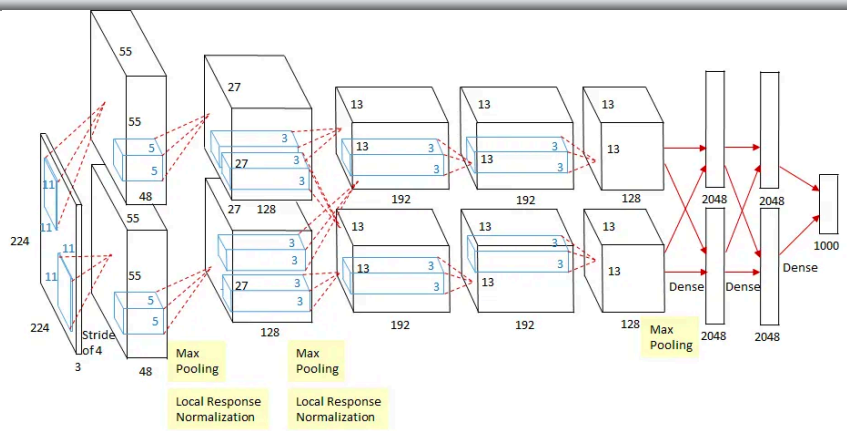
2022

ConvNeXt

CNN fights back

AlexNet (2012)

Krizhevsky, Sutskever, Hinton — The paper that started the deep learning revolution.



ReLU

First large-scale use.
Faster than sigmoid/tanh.

Dropout

Regularization for
FC layers. Novel at the time.

GPU Training

Split across 2 GPUs.
Made deep nets practical.

Data Aug

Random crops, flips,
PCA color augmentation.

VGGNet (2014)

Simonyan & Zisserman — "Deeper is better, but keep it simple."

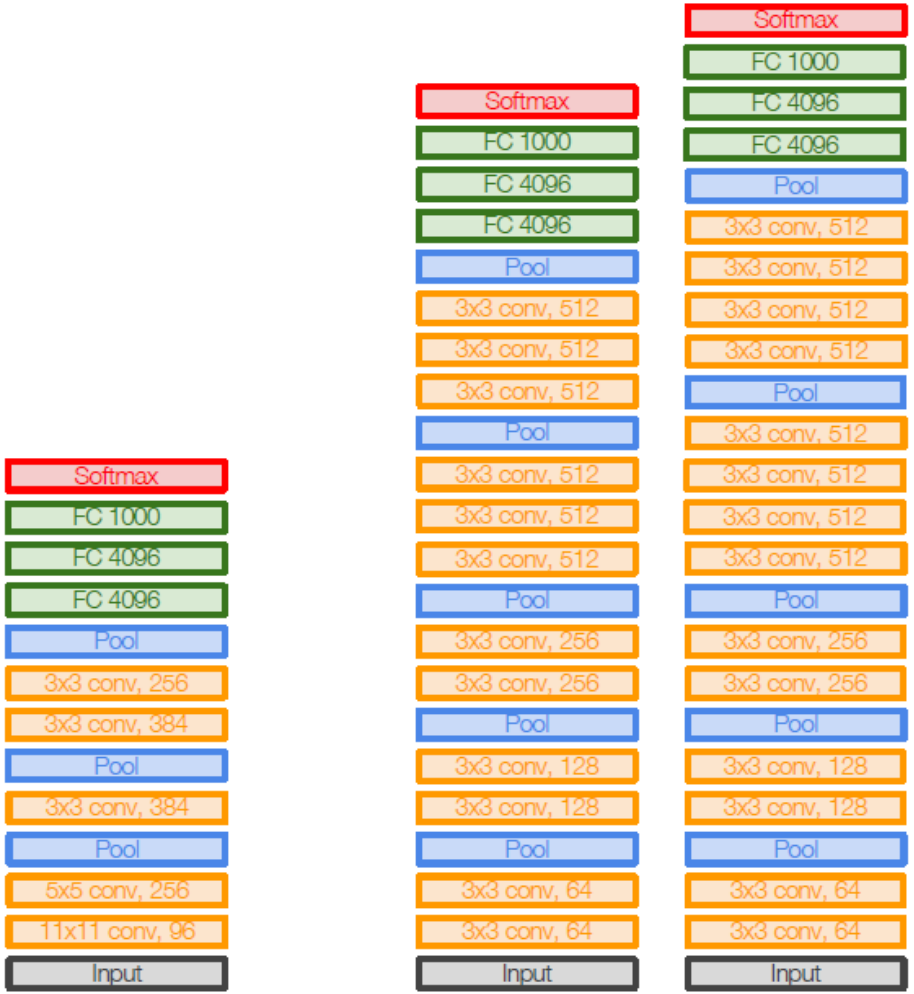
Core Insight

Small filters, Deeper networks

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

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

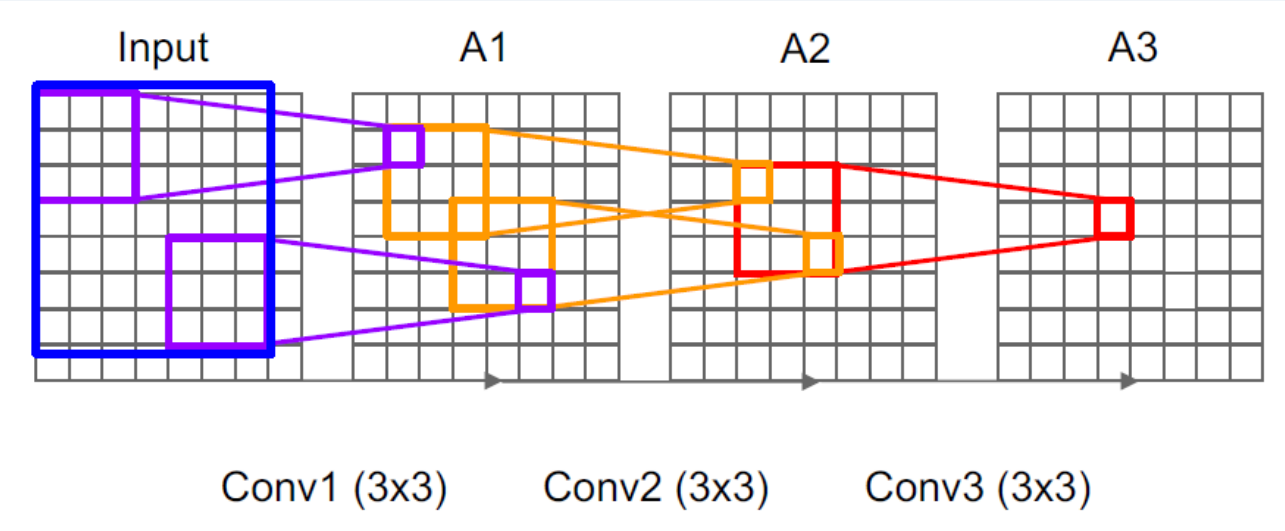
11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14



VGGNet (2014)

Simonyan & Zisserman — "Deeper is better, but keep it simple."

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



VGGNet (2014)

Simonyan & Zisserman — "Deeper is better, but keep it simple."

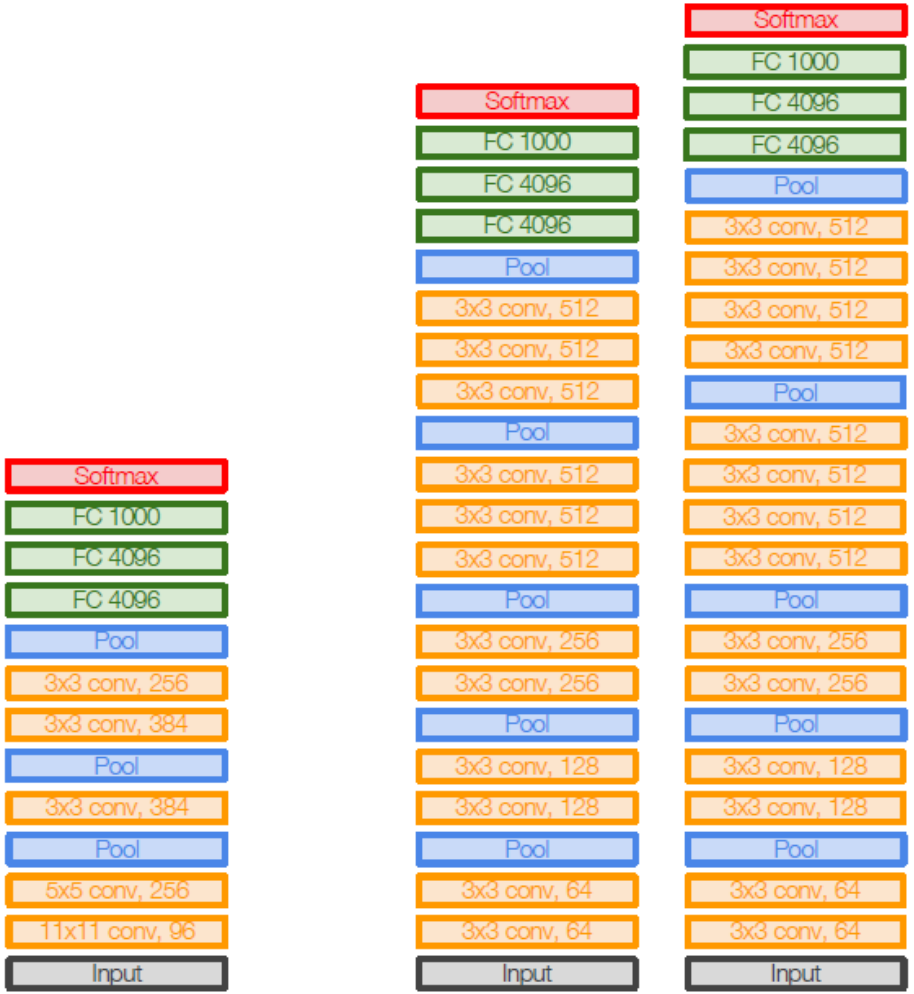
Core Insight

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

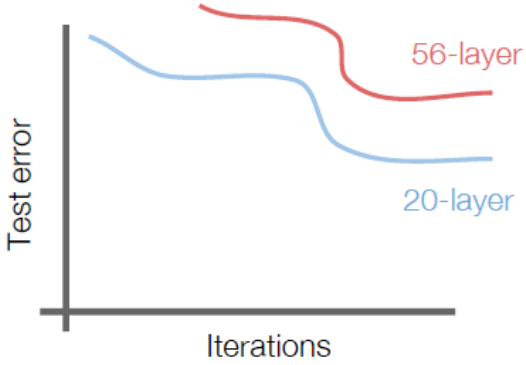
But deeper, more non-linearities

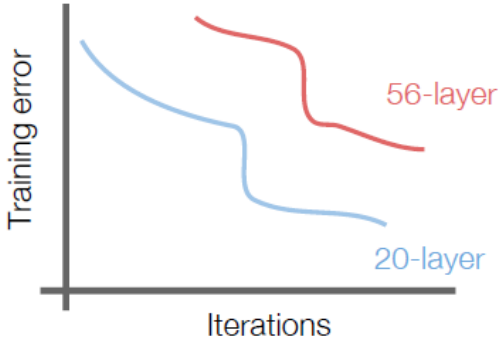
And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer



ResNet (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!

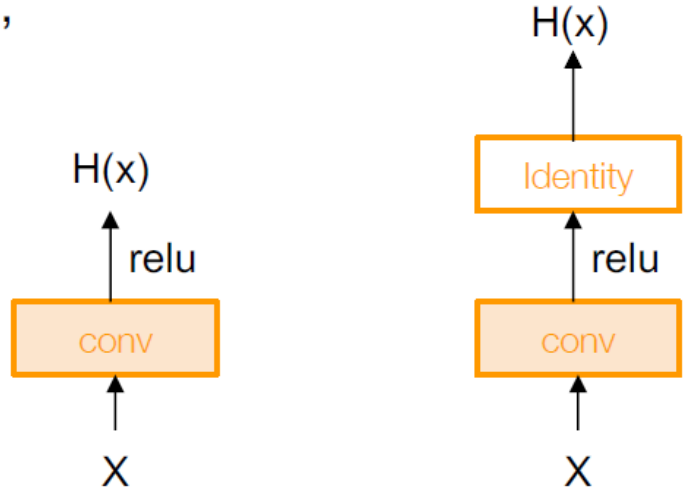
ResNet (2015)

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

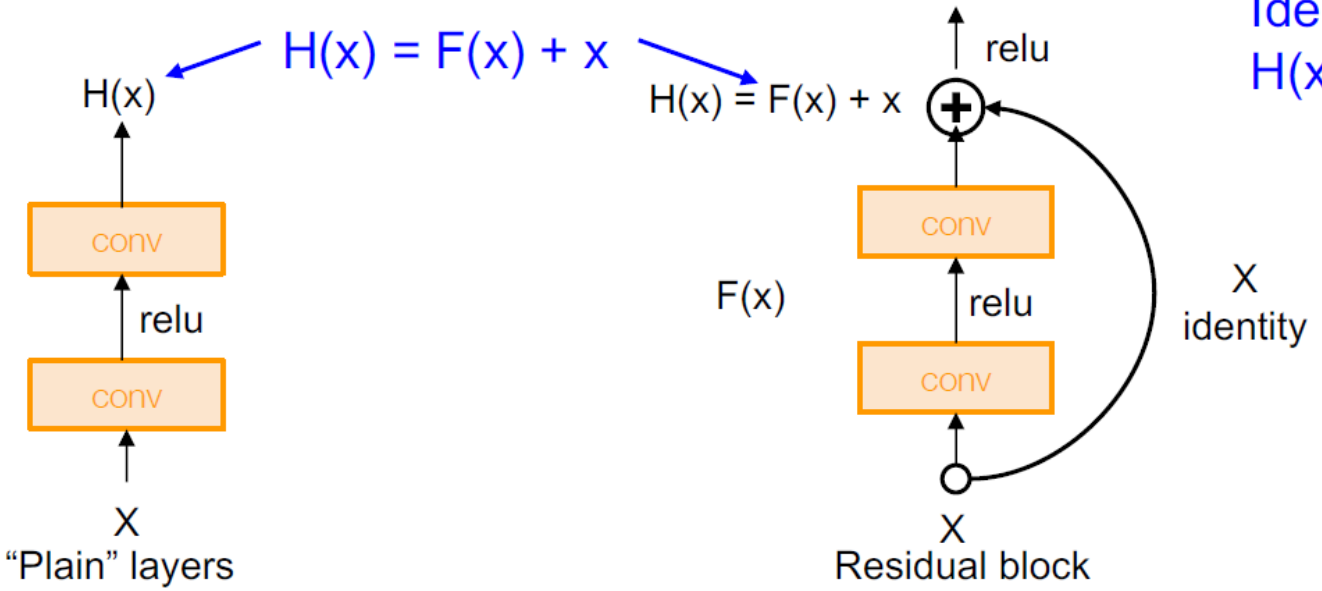
What should the deeper model learn to be at least as good as the shallower model?

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



ResNet (2015)

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

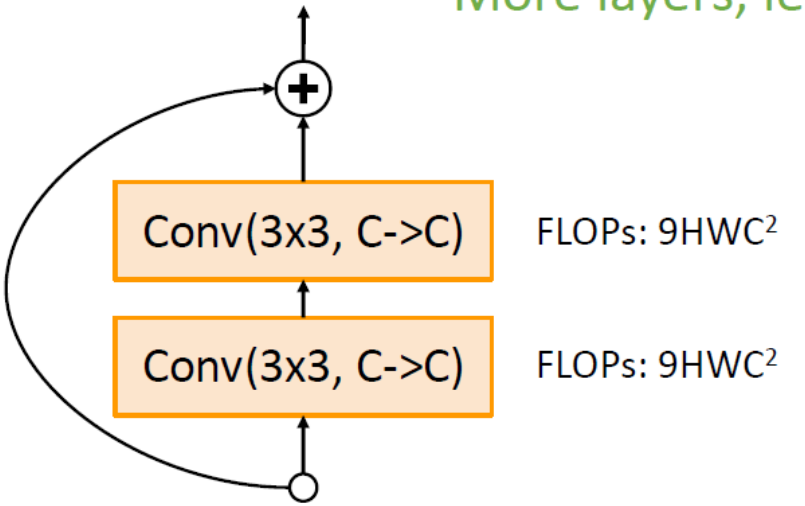


Identity mapping:
H(x) = x if F(x) = 0

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

ResNet: Basic vs Bottleneck Block

More layers, less computational cost!

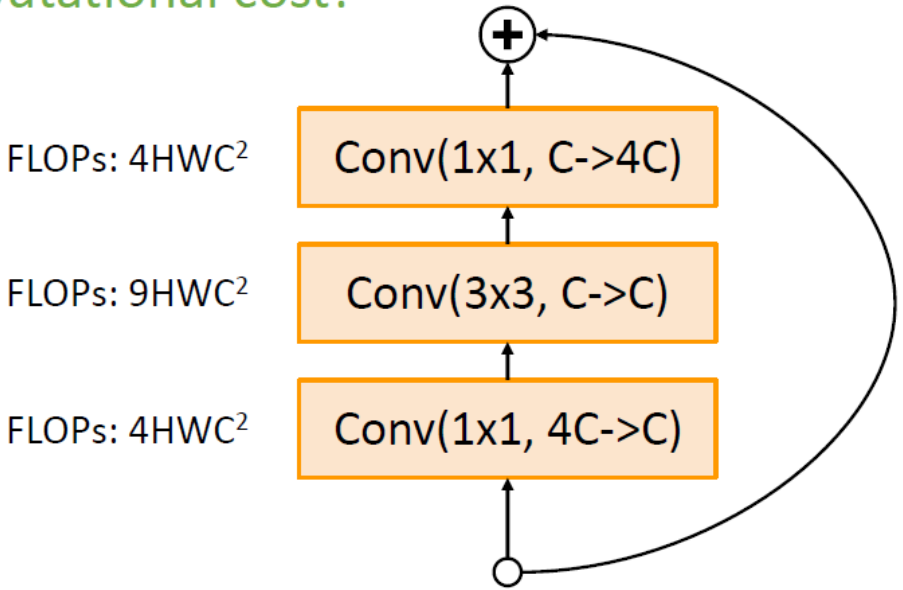


FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

"Basic" Residual block

Total FLOPs: $18HWC^2$



FLOPs: $4HWC^2$

FLOPs: $9HWC^2$

FLOPs: $4HWC^2$

Total FLOPs: $17HWC^2$

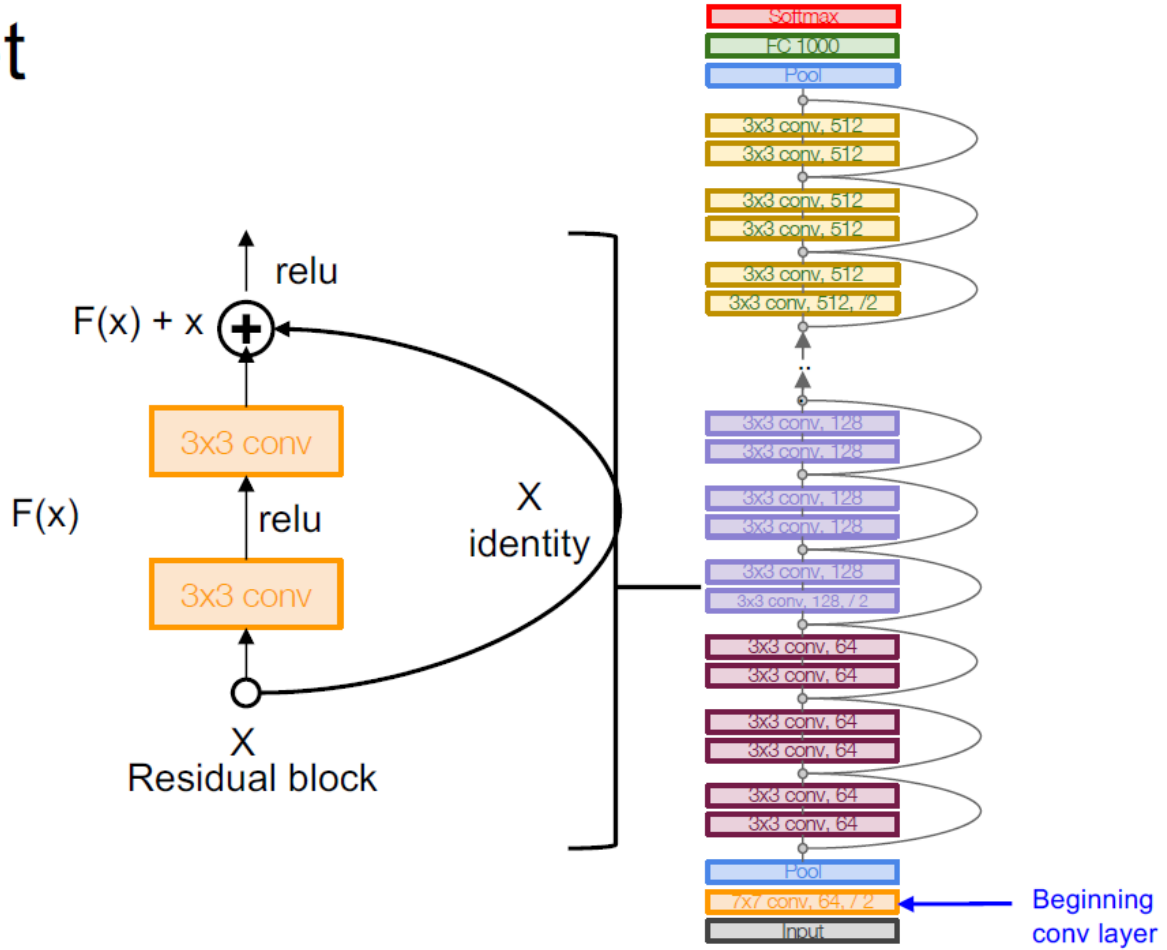
"Bottleneck" Residual block

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)

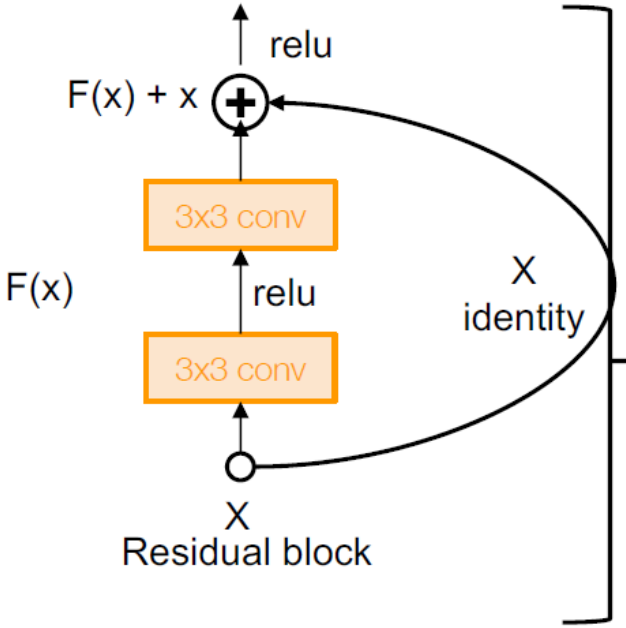


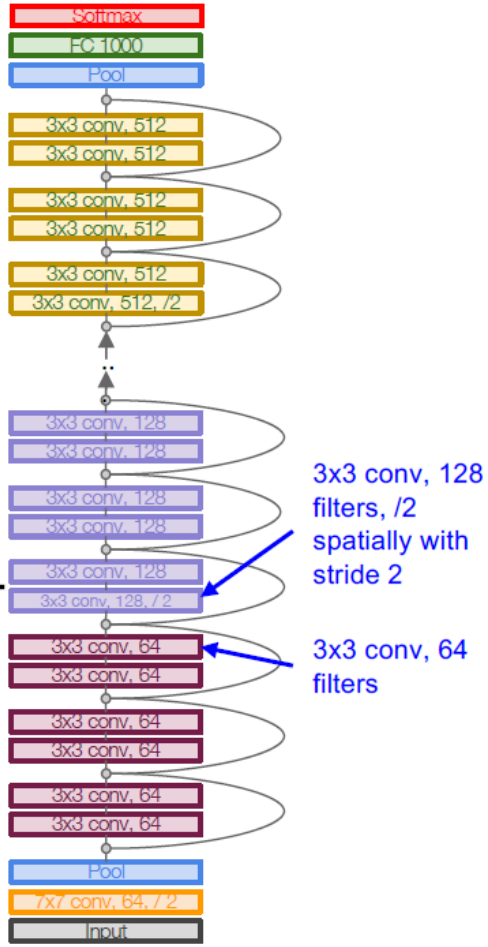
Case Study: ResNet

[He et al., 2015]

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.

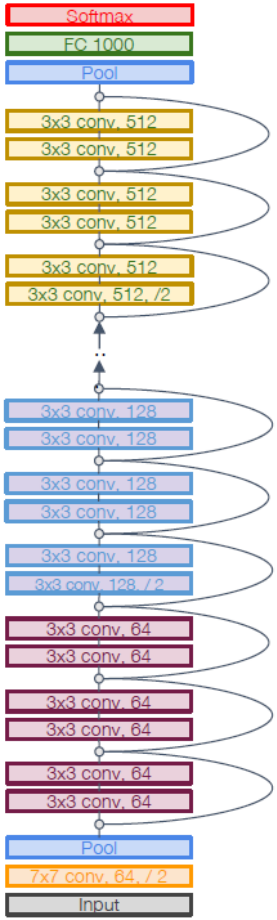




Residual Networks

ResNet-50 is the same as ResNet-34, but replaces Basic blocks with Bottleneck Blocks.
This is a great baseline architecture for many tasks even today!

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

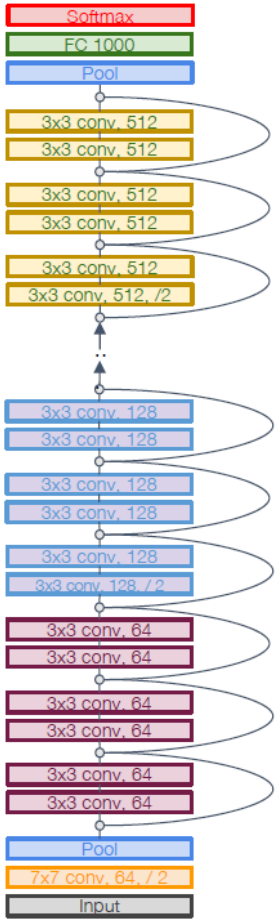


He et al, "Deep Residual Learning for Image Recognition", CVPR 2016
Error rates are 224x224 single-crop testing, reported by torchvision

Residual Networks

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

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



He et al, "Deep Residual Learning for Image Recognition", CVPR 2016
Error rates are 224x224 single-crop testing, reported by torchvision

After ResNet: Deeper, Wider, Efficient

Natural evolution of the ResNet paradigm — still the CNN era.

ResNeXt (2017)

"Split-Transform-Merge"

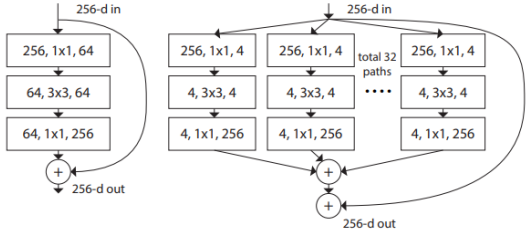


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

Replace one big conv with multiple parallel pathways (grouped convolutions).

More capacity at the same parameter count.

Cardinality > Depth or Width

DenseNet (2017)

"Connect everything"

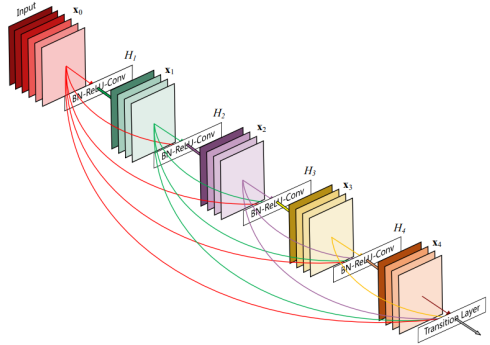


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Every layer receives input from ALL previous layers (concatenation, not addition)

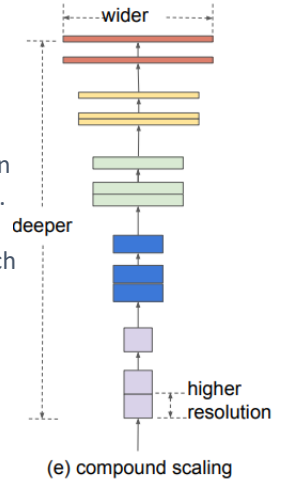
Dense connections + Growth rate

EfficientNet (2019)

"Scale smartly"

Compound scaling: increase width × depth × resolution together with fixed ratios.

Neural Architecture Search finds optimal base network.

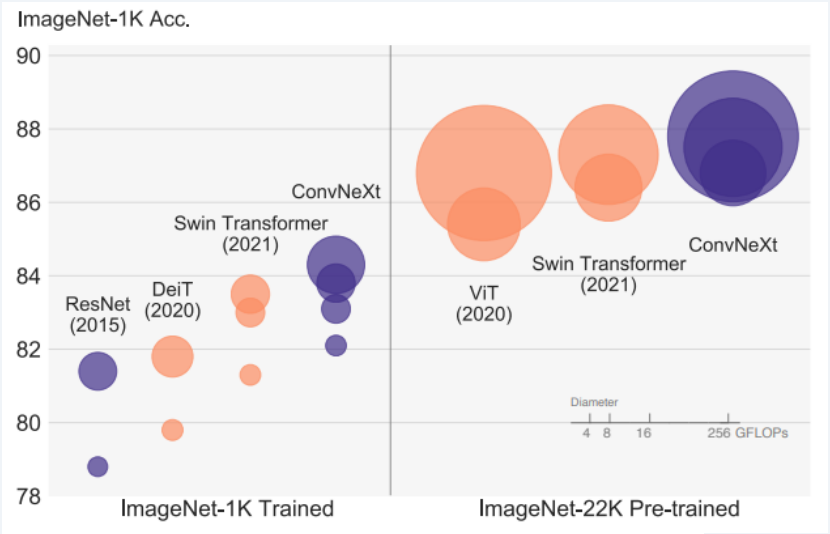


Better accuracy/FLOPS trade-off

All three build on ResNet's skip connection. The core paradigm is the same: convolution + residual.

ConvNeXt (2022)

Liu et al. — "A ConvNet for the 2020s". What if we apply ViT's design choices to a pure CNN?



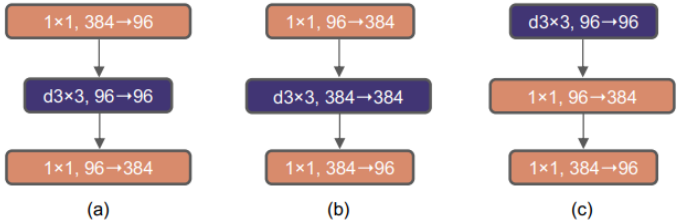
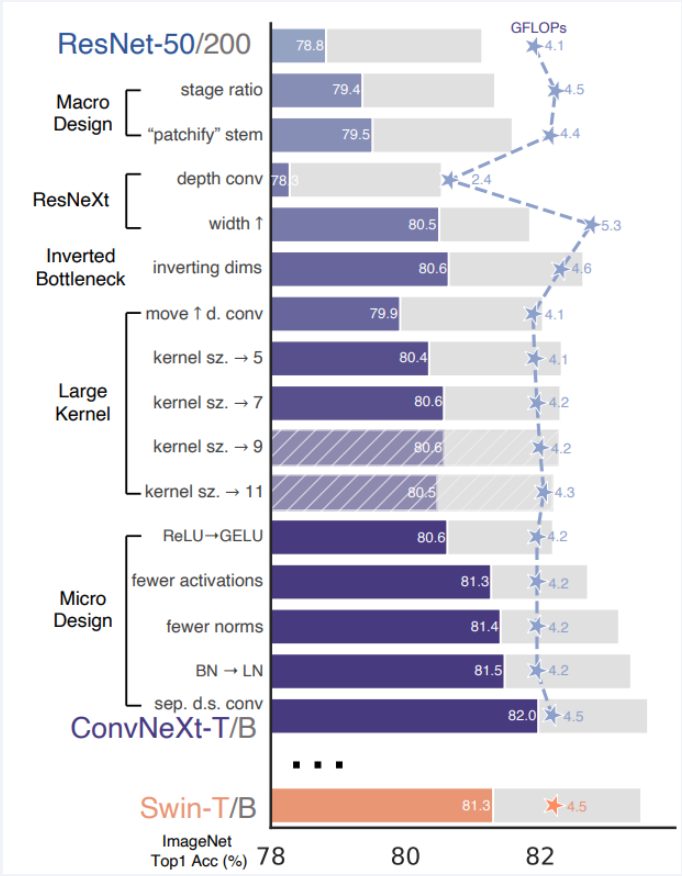


Figure 3. **Block modifications and resulted specifications. (a)** is a ResNeXt block; in **(b)** we create an inverted bottleneck block and in **(c)** the position of the spatial depthwise conv layer is moved up.



ConvNeXt (2022)

Liu et al. — "A ConvNet for the 2020s". What if we apply ViT's design choices to a pure CNN?

The Question

2020: ViT shows pure Transformers beat CNNs.

But was it the attention mechanism?
Or the modern training recipe?

ConvNeXt answer: Start from ResNet-50,
gradually adopt ViT's design choices
(training tricks, macro design, micro design)
→ match ViT performance with pure convolutions.

Key Changes (ResNet → ConvNeXt)

ResNet training	→	**ViT training recipe (300 epochs, AdamW, augmentation)**
3×3 conv	→	**7×7 depthwise conv (larger kernel like ViT's patch)**
BatchNorm	→	**LayerNorm**
ReLU	→	**GELU**
Bottleneck block	→	**Inverted bottleneck (wide → narrow → wide)**

Takeaway

CNN vs Transformer is not about convolution vs attention. It's about training recipes, scaling, and design choices.
→ Week 5: We'll learn what ViT actually does, and why these design choices matter.

Part 4

What CNNs Learn & Transfer Learning

Peeking inside the black box, and leveraging pretrained knowledge.

Feature Inversion

"What does the network remember at each layer?" — Reconstruct the image from features only.

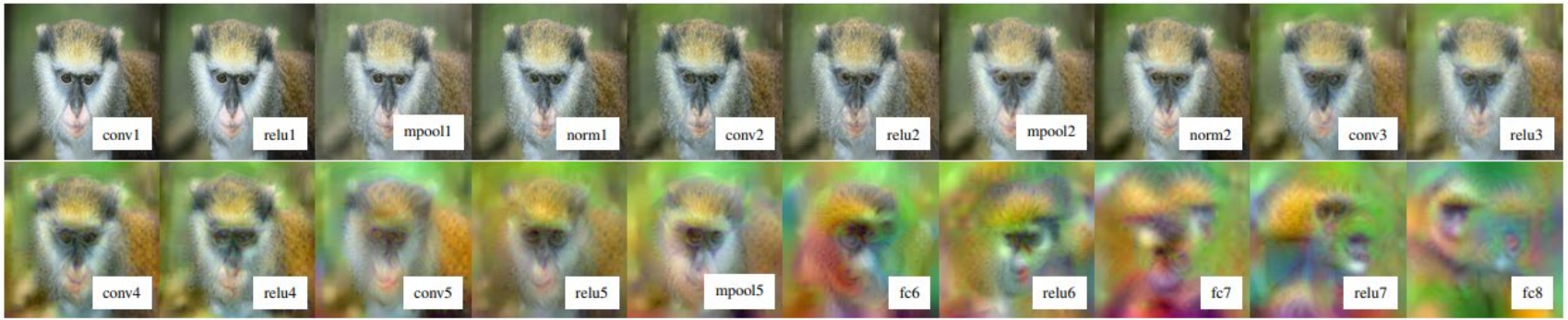


Figure 6. **CNN reconstruction.** Reconstruction of the image of Fig. 5.a from each layer of CNN-A. To generate these results, the regularization coefficient for each layer is chosen to match the highlighted rows in table 3. This figure is best viewed in color/screen.

Early Layers

Almost perfect reconstruction.
The network retains nearly all
pixel-level information.

Middle Layers

Textures and local structures
survive, but exact pixel
positions are lost.

Deep Layers

Only high-level semantic
information remains.
Colors and details are gone.

Insight: Deeper layers progressively discard spatial detail and keep semantic meaning. This is why transfer learning works.

CAM: Class Activation Mapping

Zhou et al. (2016) — Where is the network looking to make its prediction?

How It Works

1. Last conv layer produces feature maps (C × H × W)
2. Global Average Pooling (GAP) → one value per channel
3. FC layer weights w_c connect GAP to class score
4. $CAM = \Sigma\, w_c \times feature_{map_c}$
   $(weighted\ sum\ of\ feature\ maps)$

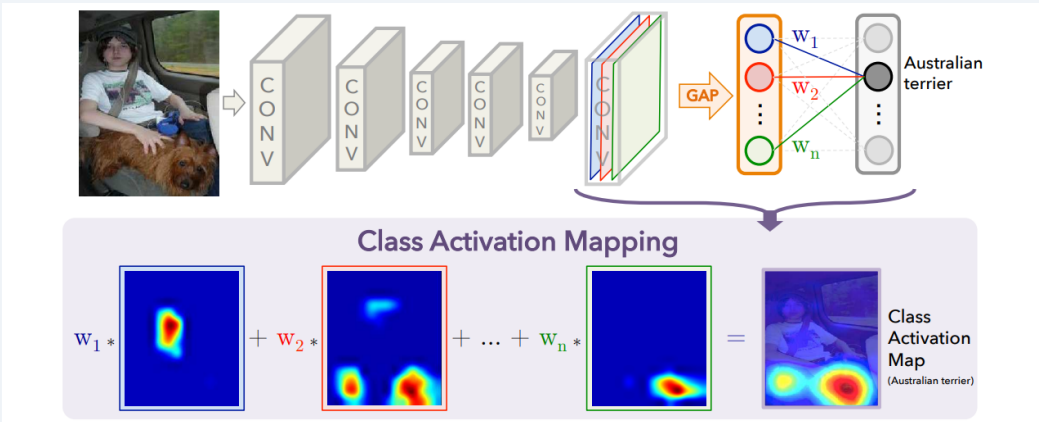


Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.

Limitation

Requires Global Average Pooling (GAP) before the final FC layer. Only works with architectures that have this specific structure (e.g., GoogLeNet, ResNet). Cannot be applied to models with multiple FC layers (e.g., VGG) or arbitrary layers.

Grad-CAM: Generalized CAM

Selvaraju et al. (2017) — Use gradients instead of GAP weights. Works with any CNN.

Key Idea

CAM uses FC weights to weight feature maps.
Grad-CAM uses gradients instead:

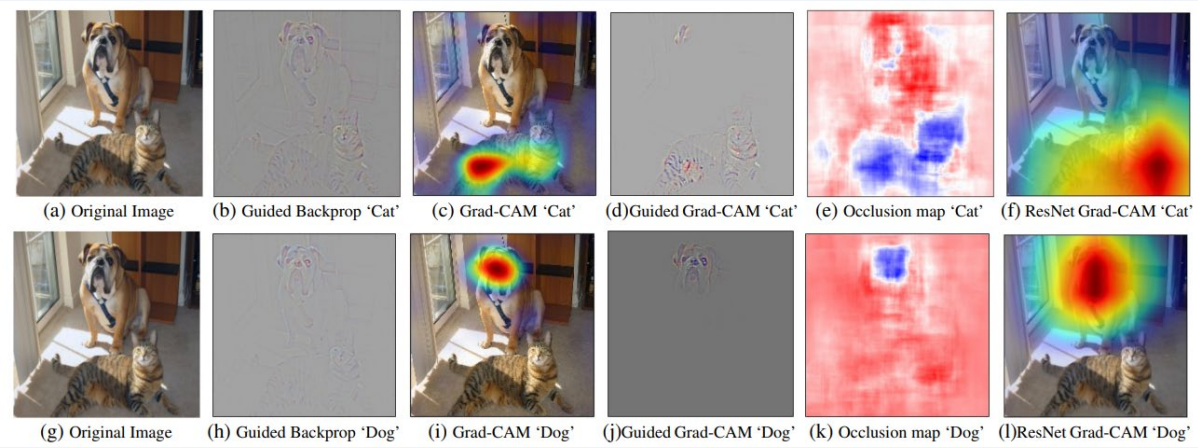
$$\alpha_c = GAP\left(\frac{\partial y_{class}}{\partial A_c}\right) \quad \leftarrow \text{importance of channel c}$$

$$L = ReLU\left(\Sigma\, \alpha_c \times A_c\right) \leftarrow \text{weighted sum + ReLU}$$

No architectural constraint. Any conv layer, any model.

CAM vs Grad-CAM

	CAM	Grad-CAM
Weight source	FC weights	Gradients
Requires GAP?	Yes	No
Any layer?	Last only	Any conv layer
Any architecture?	Limited	Universal



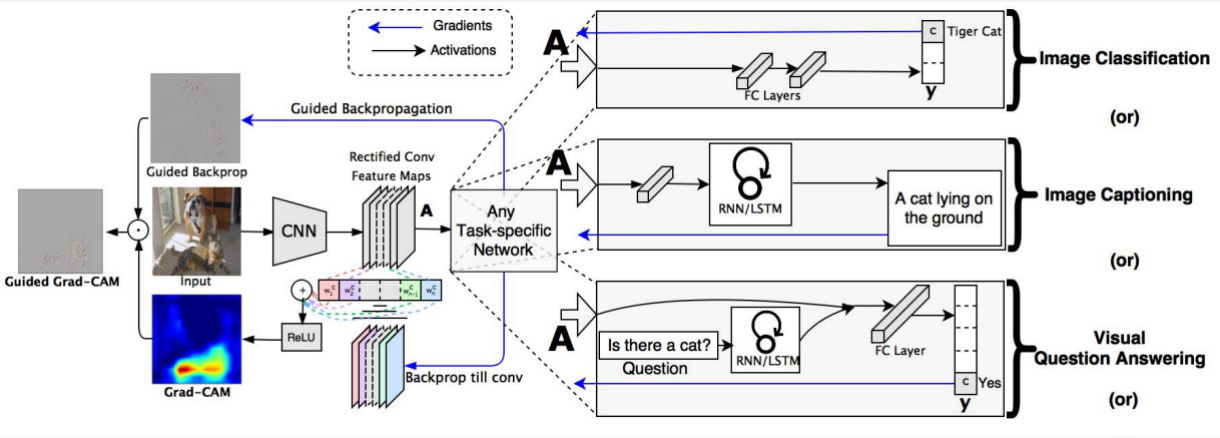
Grad-CAM: Where Does the Model Look?

Selvaraju et al. (2017) — Visual explanations from any CNN-based model.

How it works

1. Forward pass → get class score
2. Backward pass → get gradients at last conv layer
3. Weighted combination of activation maps
4. ReLU → heatmap showing important regions for that class

No architecture modification needed.



Why it matters Model debugging (is it looking at the right thing?), trustworthiness, failure analysis. Essential tool for your challenge projects.

Transfer Learning

Don't train from scratch. Start from a pretrained model and adapt.

1. Pretrain on ImageNet

- Train on 1.28M images, 1000 classes
- Learn general features

2. Freeze Backbone

- Keep pretrained conv layers
- Remove final FC layer

3. Fine-tune on Your Data

- Add new classifier head
- Train with small lr
- Optionally unfreeze layers

When to use which?

Small dataset, similar domain

→ Freeze backbone, train classifier only.
Low risk of overfitting.

Large dataset or different domain

→ Fine-tune entire network with small lr.
More flexibility, needs more data.

Summary & Resources

What We Covered

MLP limitations → CNN inductive biases
Convolution: filters, feature maps, stacking
Stride, padding, pooling, output size
AlexNet → VGG → ResNet evolution
ConvNeXt: CNN still competitive
Visualization: features, CAM, Grad-CAM
Transfer learning: pretrain → fine-tune

Further Reading & Self-Study

Michigan EECS 498 (YouTube):
- L7 Convolutional Networks
- L8 CNN Architectures

CS231n Notes: cs231n.github.io
- Module 2: Convolutional Neural Networks

Key Papers:
- AlexNet (Krizhevsky 2012)
- VGG (Simonyan 2014)
- ResNet (He 2015)
- ConvNeXt (Liu 2022)

Next Week Week 4: RNN + Attention Mechanism — From sequence modeling to the birth of attention

Acknowledgments

Some slides and figures in this course are adapted from or inspired by the following open resources.

Stanford CS231n: Deep Learning for Computer Vision (Spring 2025)

Fei-Fei Li, Ehsan Adeli, et al. | cs231n.stanford.edu

UMich EECS 498/598: Deep Learning for Computer Vision (Winter 2022)

Justin Johnson | web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

MIT 6.8300: Advances in Computer Vision (Spring 2025)

Vincent Sitzmann | scenerepresentations.org/courses/2025/spring/advances-in-cv/

MIT 6.7960: Deep Learning (Fall 2024)

Phillip Isola, Sara Beery, Jeremy Bernstein | ocw.mit.edu/courses/6-7960-deep-learning-fall-2024/

CMU 16-824: Visual Learning and Recognition (Fall 2025)

Jun-Yan Zhu | visual-learning.cs.cmu.edu