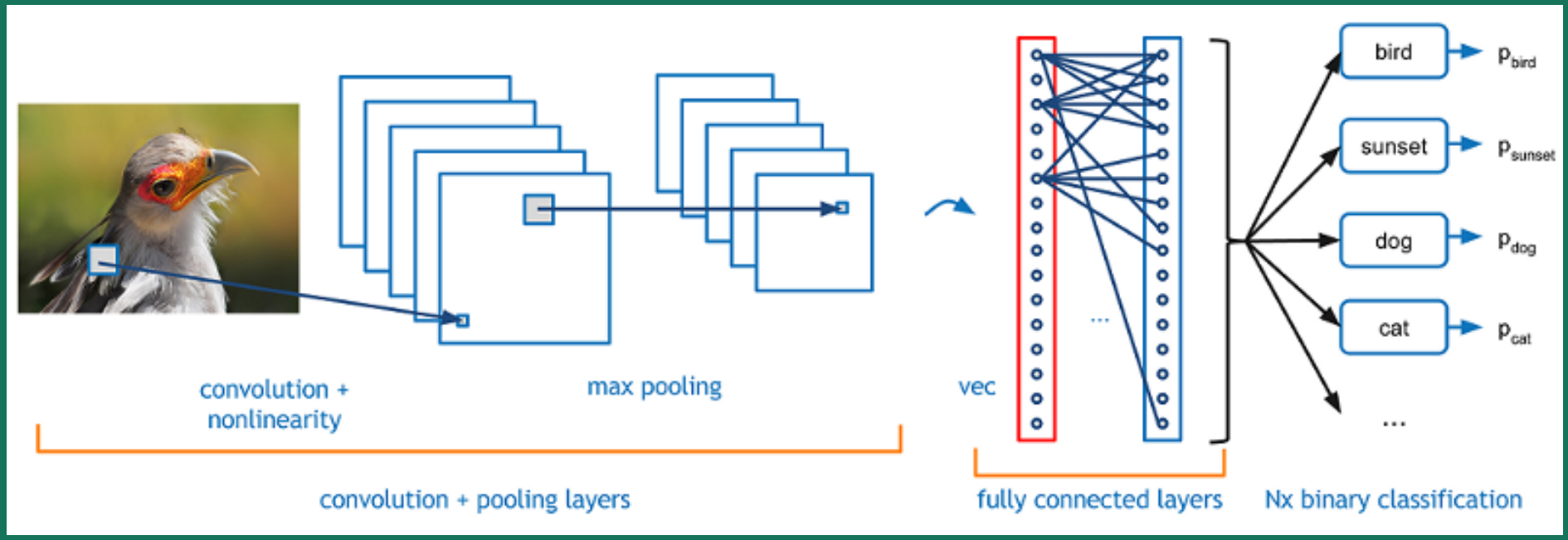
Convolutional Neural Networks (CNN) – Part 2

EE 599 Deep Learning

Spring 2019

Brandon Franzke

Convnets are Everywhere!



Reminder: slides show mostly vision (input) + classification (output).
Adjust input / output layers for other tasks (e.g. MSE for regression)

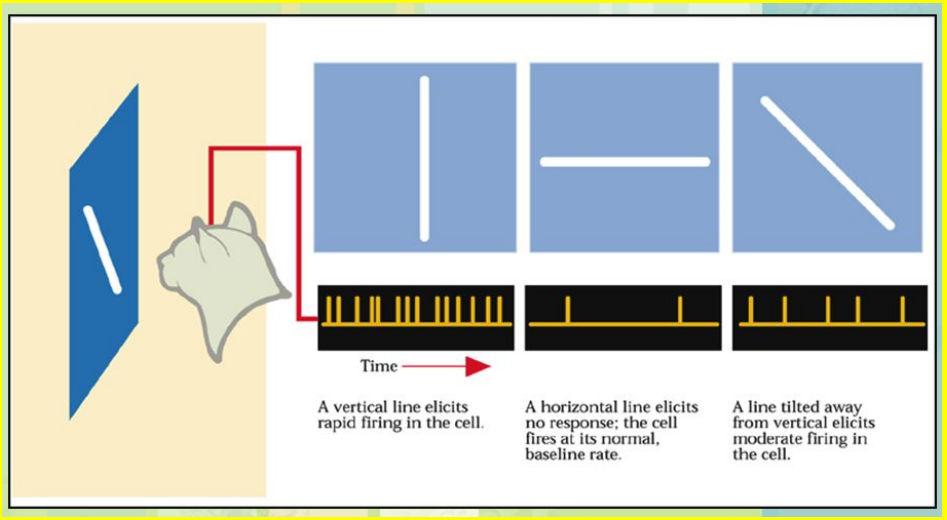
Convnets are the most influential machine learning advancement in the past decade.

This week we cover

- Convolution layer
  - Dimension, stride, channels and depth, padding
- Pooling layers
- Backpropagation
- Architectures
  - VGGNet, AlexNet, GoogLeNet (inception), ResNet (residual)
  - MobileNet (“depthwise” convolution)

CNNs are based on *award-winning* research from the 1960s…

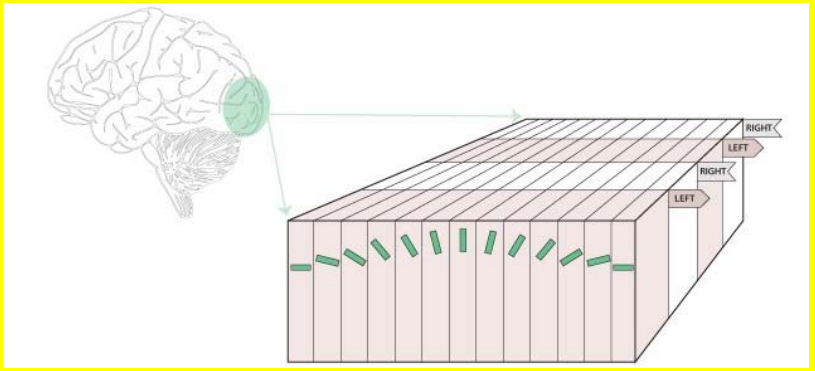
Hubel and Wiesel



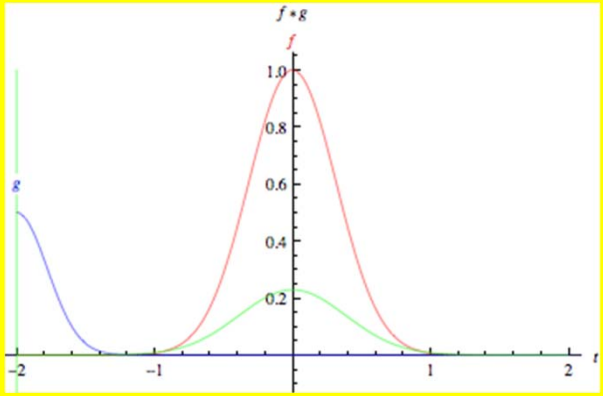
<u>The visual cortex is a topographical map.</u>

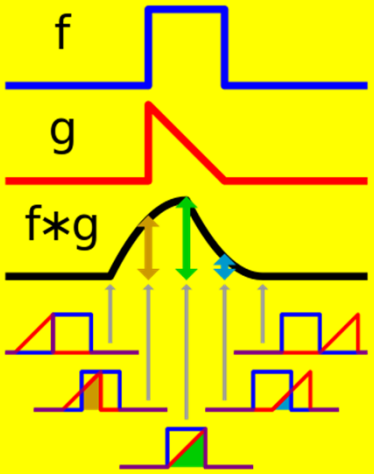
Columns in the lower visual processing units (V1) activated by stimulus in:

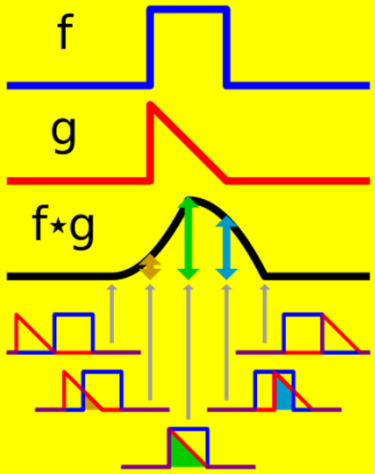
- particular areas of the retina.
- specific orientations.



Correlation = convolution without time reversal



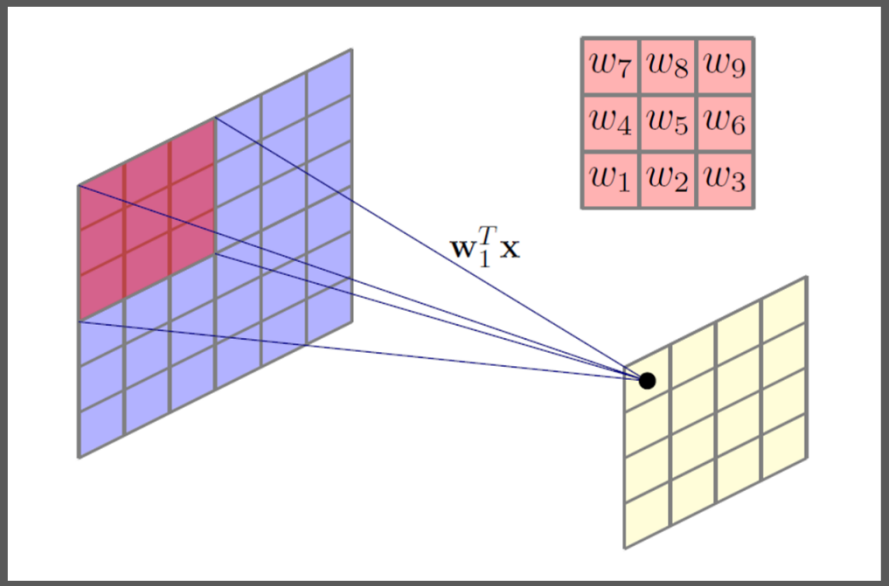


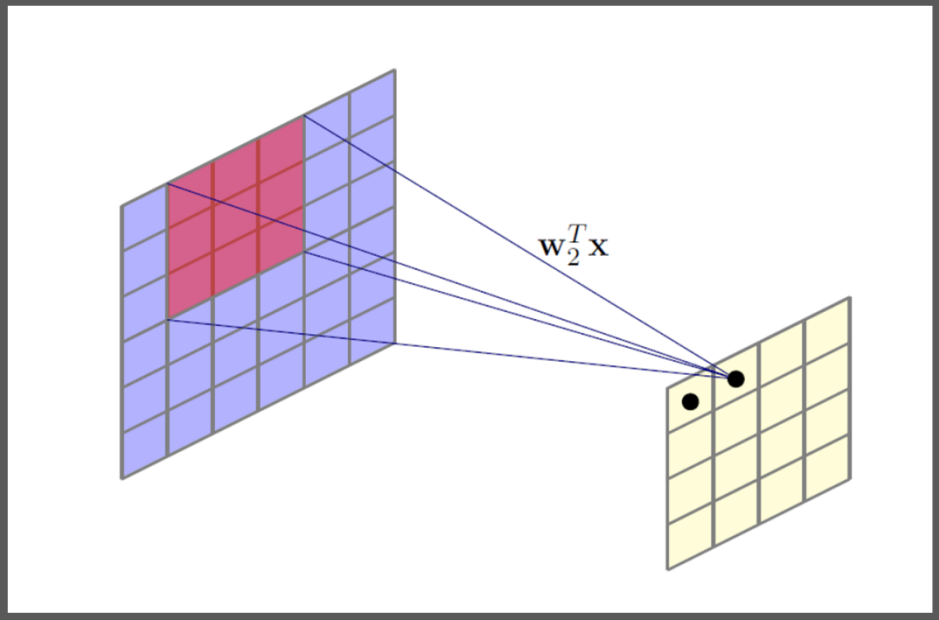


$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau) d\tau$$

CNN = Skip the time-reversal. Use simpler correlation instead. Why does it work?

CNN insight: sliding window dot product

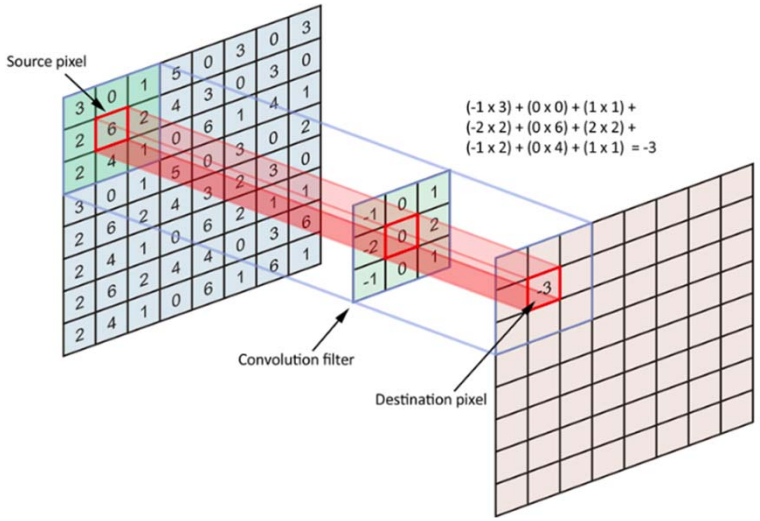




$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)\, K(m,n)$$

Apply a small set of learned parameters over the entire input

Discrete “convolution” is a local dot product.



$$S(i,j) = (I * K)(i,j)$$

$$= \sum_m \sum_n I(i+m, j+n)\, K(m,n)$$

K has small dimension $\Longrightarrow \therefore$ fast compute

This is what you learn!

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

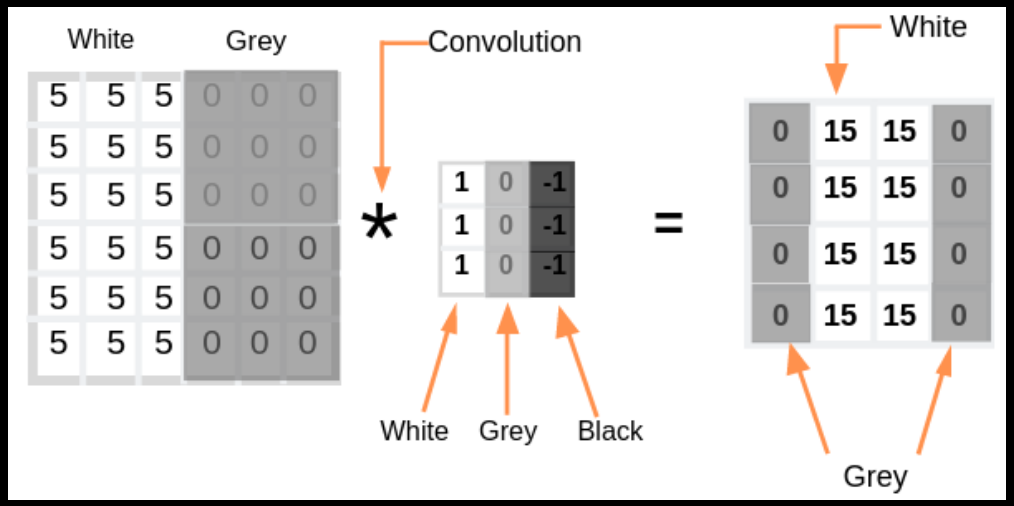
Kernel

0	-1	0
-1	5	-1
0	-1	0

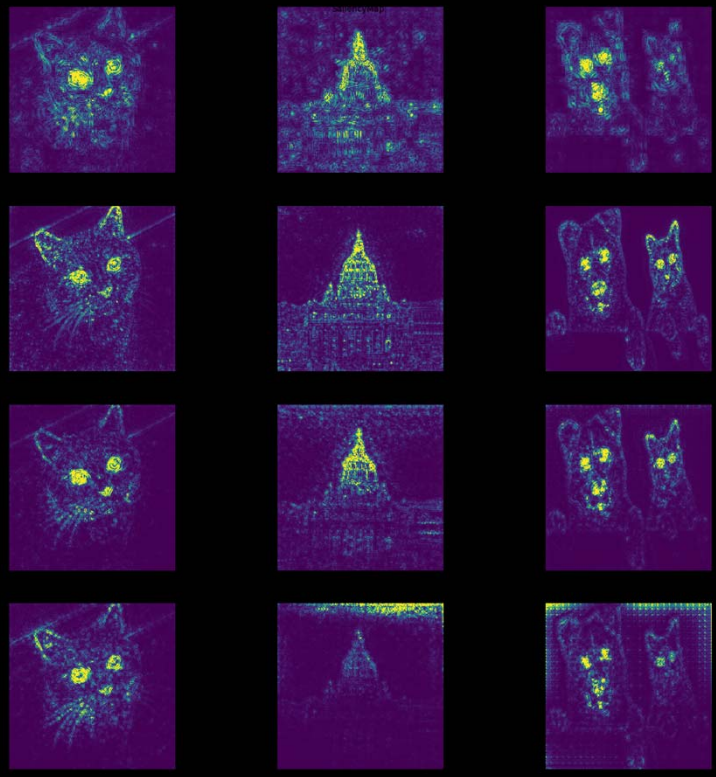
114				

Convolution kernels are feature detectors

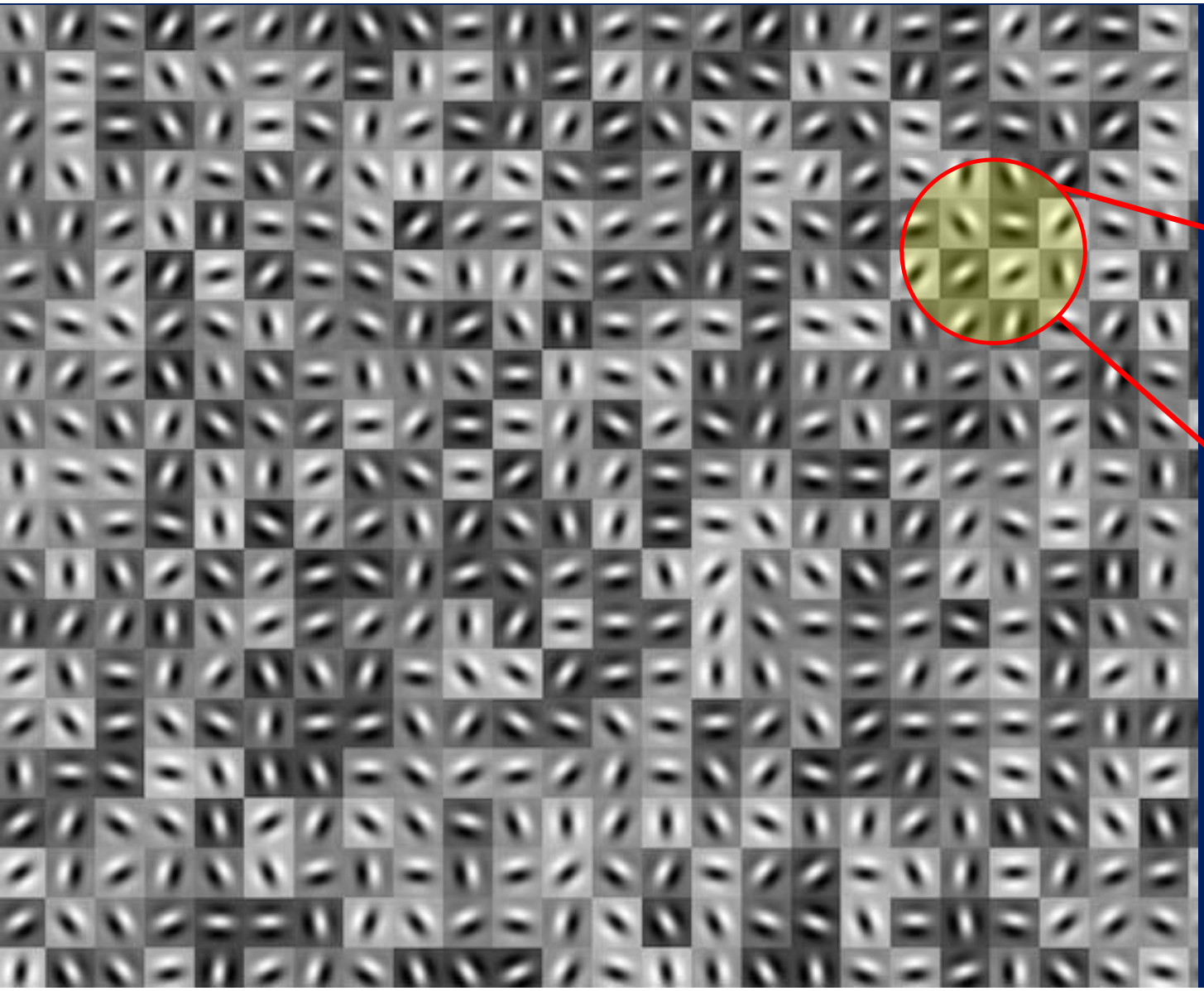
Detect edges in layer 1



Abstract later layers

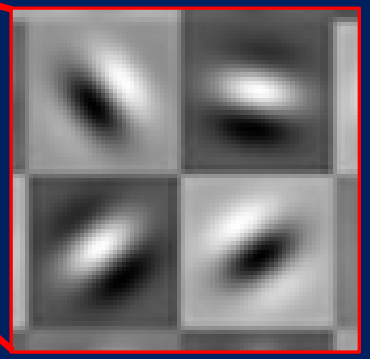


Early layers identify simple features. Later layers identify complex patterns.



Lots of filters

(think: *Gabor*)



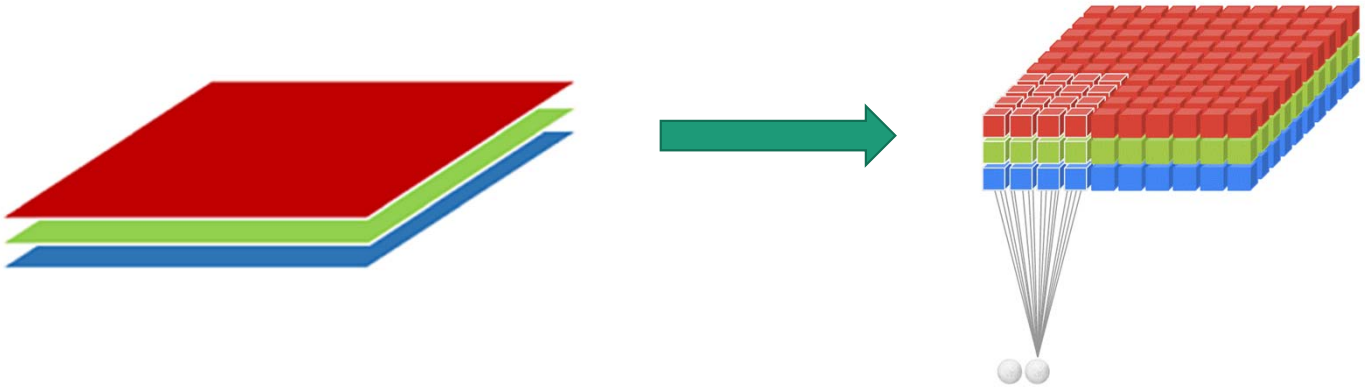
line edge

Multiply filter with input

dark = 0 and light = 1

Convolutional layers capture local structure

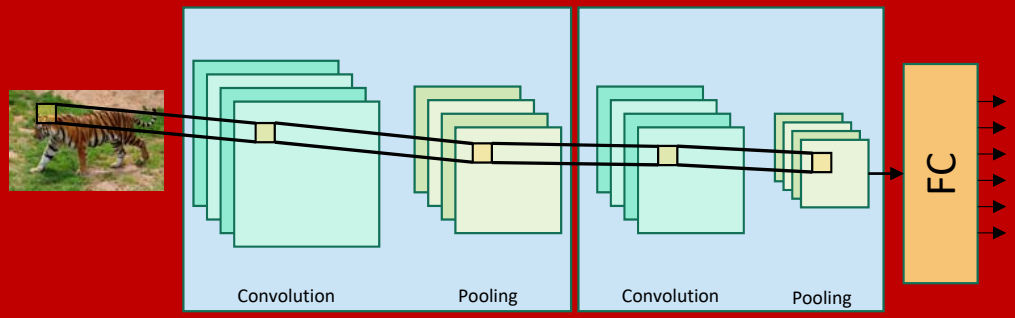
Convolve filters across channels *(usually)*

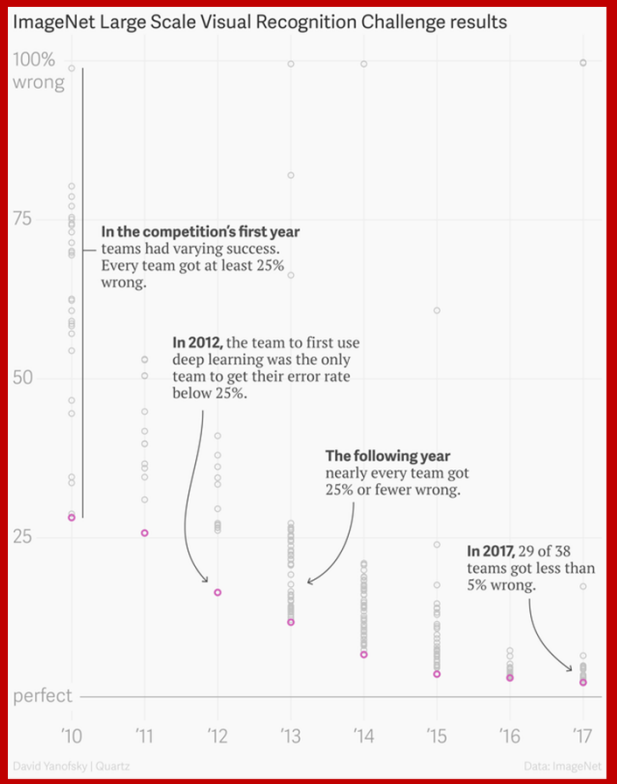


Convolve yields dot product across all channels

Why convolutions?

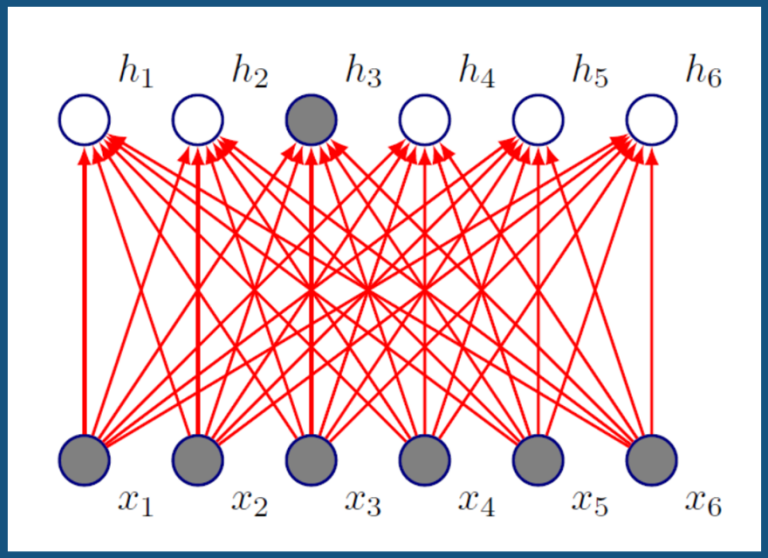
- Sparse interactions
- Parameter sharing
- Equivariant representations
- Variable input size

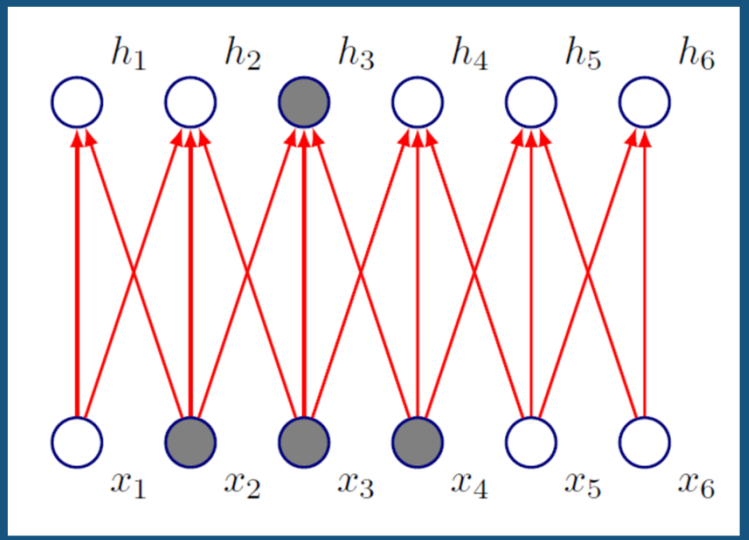




Convolutions are a good tradeoff between feature detectors and trainable parameter count

Sparse interactions reduce computation (space and time)

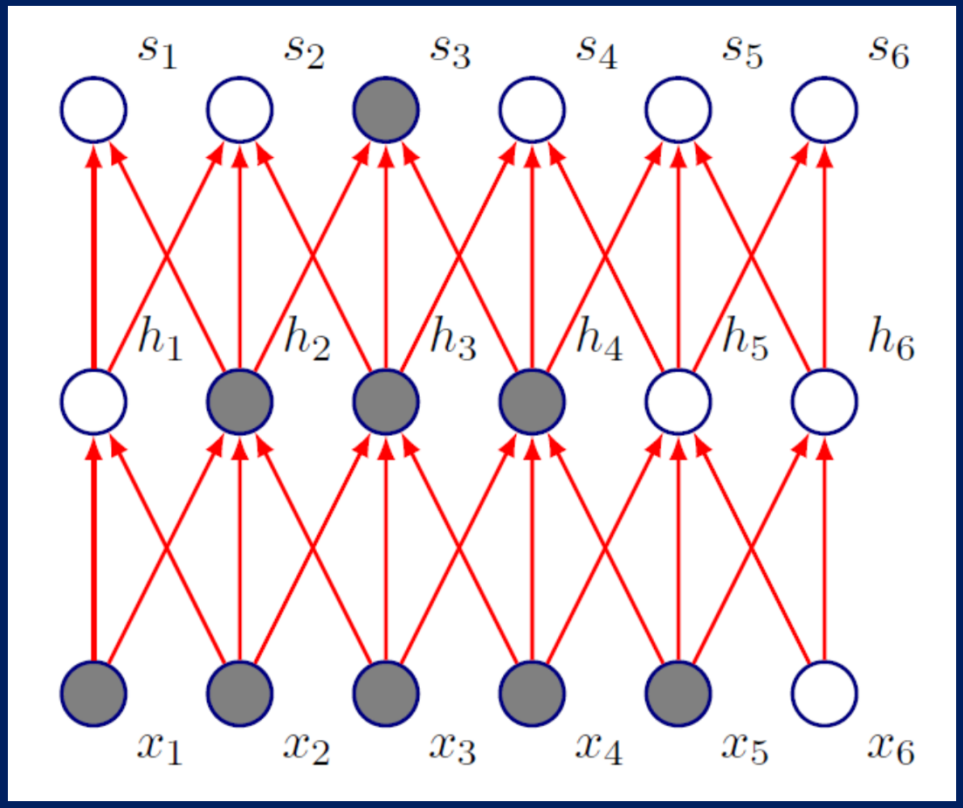




Fully connected MLP: matrix multiplication $y = \boldsymbol{W}x$ to compute activations for every layer (every output depends on every input)

CNN: *sparse interactions* since kernel is smaller than input. Need to store fewer parameters. ($O(m \times n)$ vs $O(k \times n)$)

Deeper layers handle non-locality



Deeper layers depend on progressively larger **receptive fields**

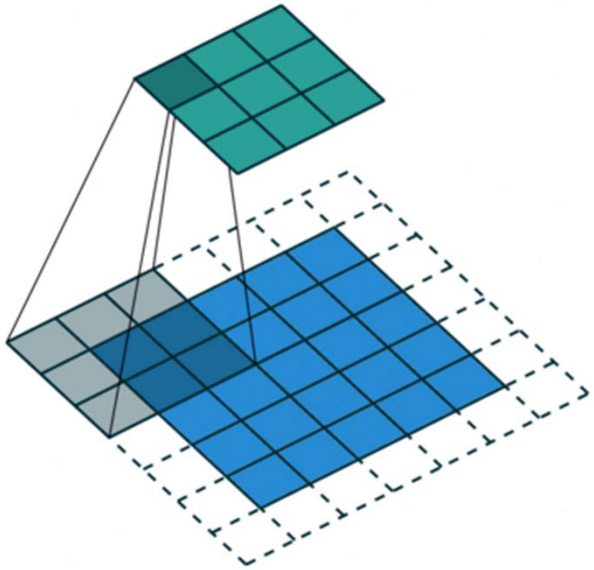
Parameter sharing reduces parameter count

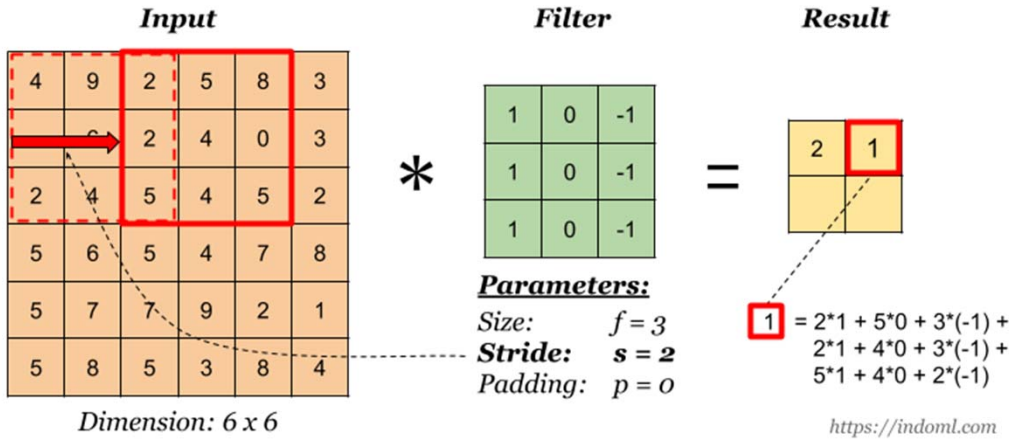
- Fully connected MLP: each element of $\boldsymbol{W}$ is used exactly once in the output layer product.
- CNN: same parameter is used over entire image (via convolution). Learn a smaller set of parameters.

Parameter equivariance robustifies network to translation

- Fully connected MLP: shifts in input features result in very different output response.
- CNN: translation of input (time or space) does not affect the output. Shifted outputs lead to the same feature representations in the output. Leads to re-usable local functions (e.g. edge detectors).

Stride reduces convolution overlap in feature map.





$$S(i,j) = (I * K)(i,j)$$

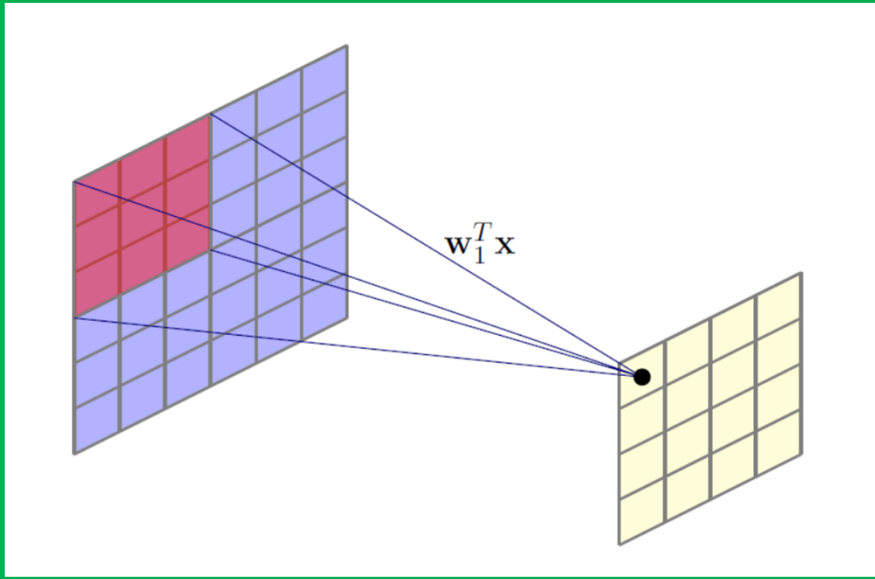
$$= \sum_m \sum_n I(i + s \cdot m, j + s \cdot n)\, K(m,n)$$

Stride *usually* equal in all dimensions (e.g. 2D, 3D).

Stride shrinks computing footprint (fewer FLOPS).

Does it reduce parameter count (model memory footprint)?

Convolution reduces output size



3x3 kernel: Input: 6x6 ⟹ output: 4x4

ConvNet output size (1D):

$$O = \frac{I - K}{S} + 1$$

O: output size
I: input size
K: kernel size
S: stride

Example: stride = 1

$$O = I - K + 1 < I$$

Problem: deep convolution networks trivialize the input dimension.

Use zero-padding to preserve size

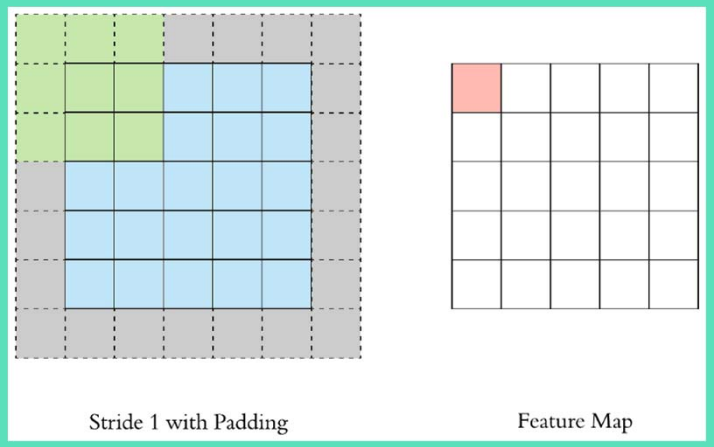
0	0	0	0	0	0	0	0
0							0
0							0
0							0
0							0
0							0
0							0
0	0	0	0	0	0	0	0

ConvNet output size (1D):

P: input padding

$$O = \frac{I - K + 2P}{S} + 1$$

Example: zero pad $P = 1$



$$O = \frac{(I + 2) - K}{S} + 1$$

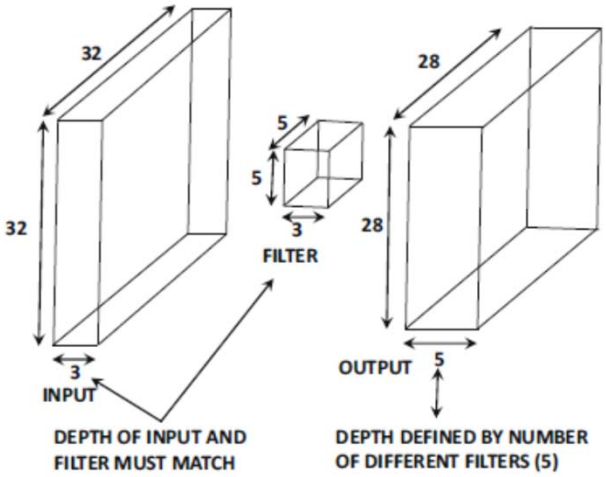
$K = 3$
$S = 1$

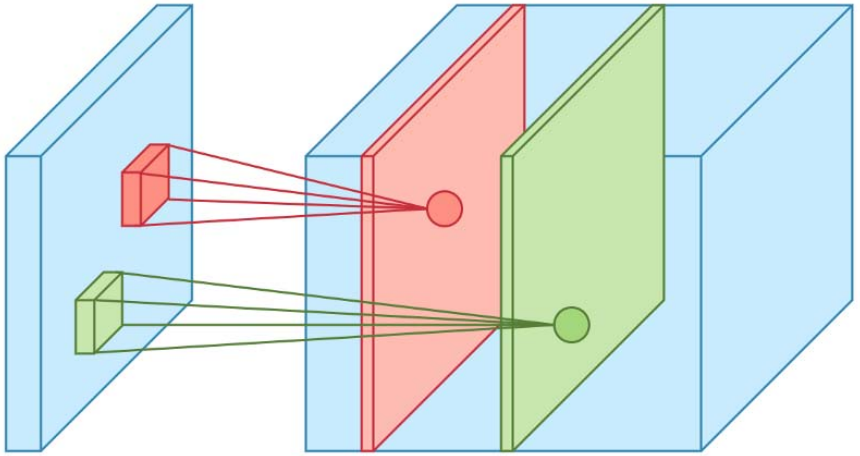
$\therefore O = I$

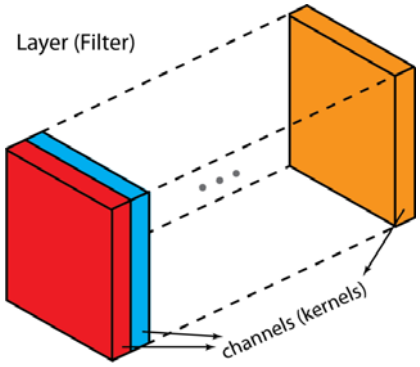
Most ConvNet packages (Keras) default to zero padding to preserve input size.

But know your gear!

Stack filters and then apply *"independently"*





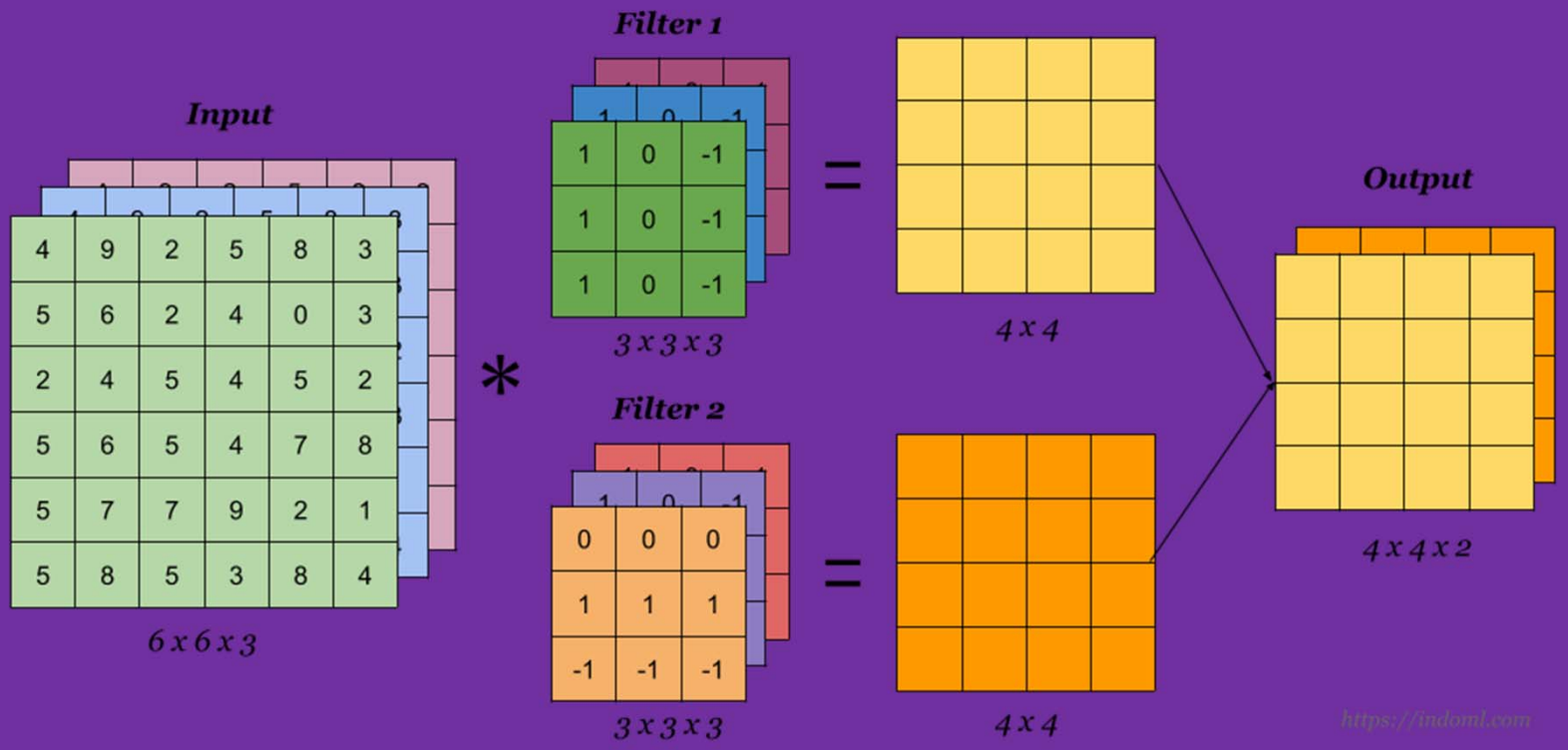


Training goal:

Learn the *filter bank* at each level

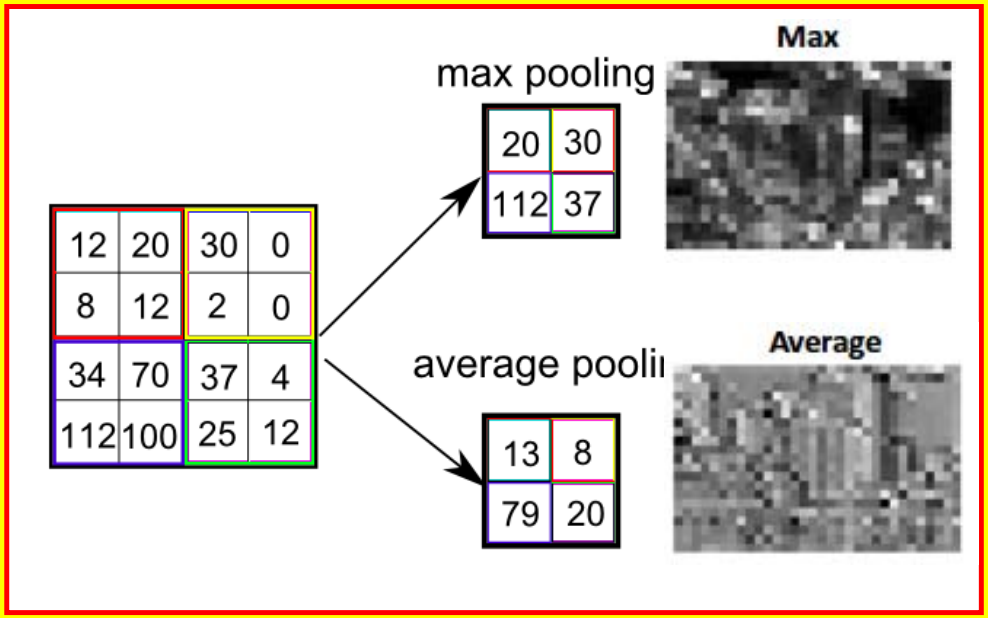
Allows for substantial speed-up by parallelizing (e.g. across GPU cores)

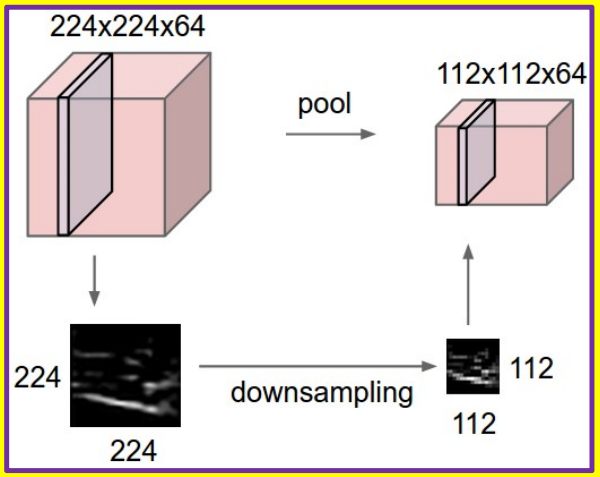
Stack filter responses to construct multi-channel output



Stacked filter responses usually Pooled (or to more Conv layers)

Use pooling to "*down-sample*" input



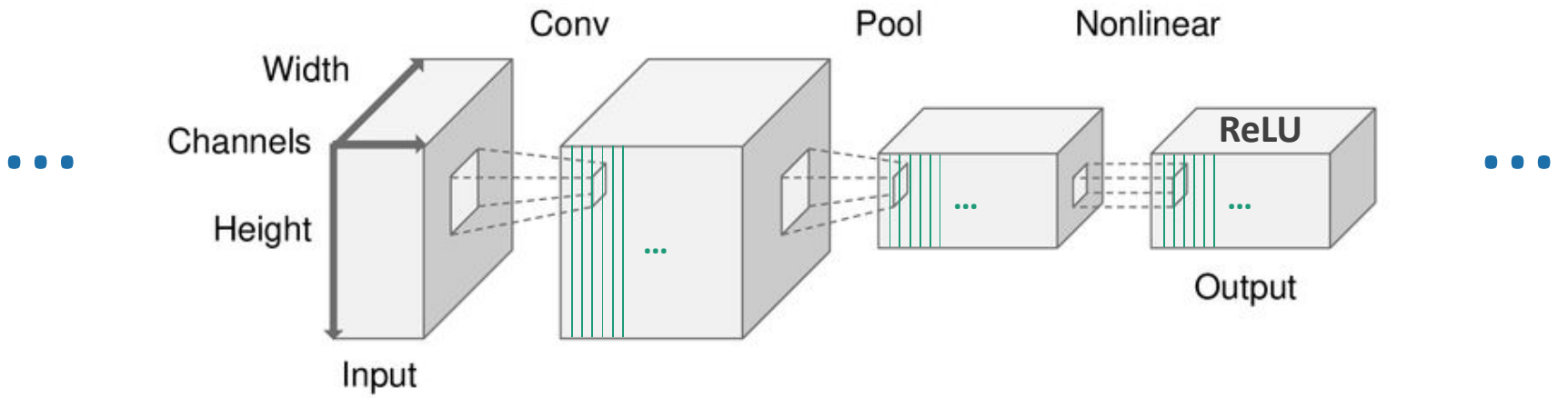


Apply pooling to each convolution output

Also: drop-out pooling (not covered here)

MaxPool "*concentrates*" (feature detection). AveragePool reduces variance.

CNNs use block templates



Block N-1

Block N

Block N+1

New blocks drive state of the art results

ConvNets have lots of parameters. Let's count.

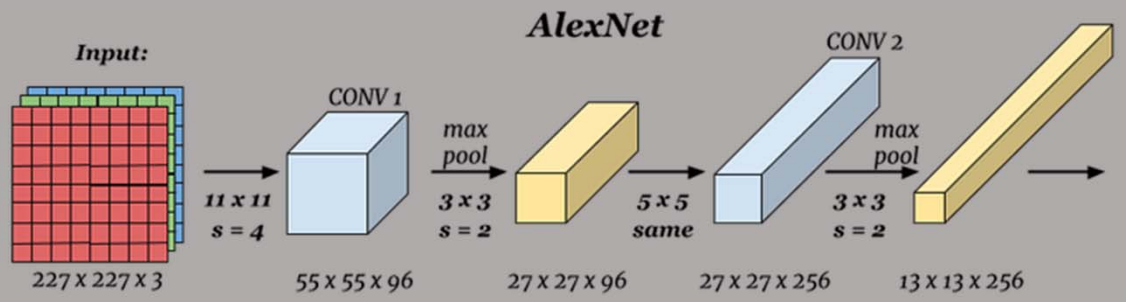
$$W_C = K^2 \times C \times N$$

$$B_C = N$$

$$P_C = W_C + B_C$$

C: input channels
N: number of filter (output channels)

Example:



- 3 input channels (R,G,B)
- $K = 11 \times 11$
- $N = 96$

$W_{C1} = 11^2 \times 3 \times 96$

$B_{C1} = 96$

$\boldsymbol{P_{C1} = 34,944}$

$P_{C2} = 614{,}656$
$P_{C3} = 885{,}120$
$P_{C4} = 1{,}327{,}488$
$P_{C5} = 884{,}992$

ConvNets efficiently scale from ≈ 154k inputs to feature detectors

First FC layer uses the most parameters

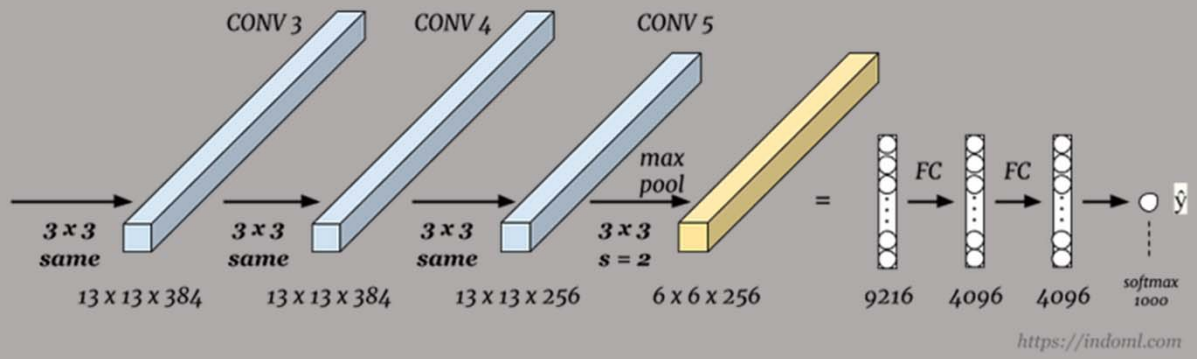
$$W_{Cf} = O^2 \times N \times F$$

$$B_{Cf} = F$$

$$P_{Cf} = W_{Cf} + B_{Cf}$$

F: number of FC neurons

Example (AlexNet):



- $O = 6 \times 6$
- $N = 256$
- $F = 4096$

$$W_{Cf} = 6^2 \times 256 \times 4096$$

$$B_{Cf} = 4096$$

$$\boldsymbol{P_{Cf} = 37,752,832}$$

ConvLayer to FC takes 10x parameters as all ConvLayers!

Classification layers need a lot of training

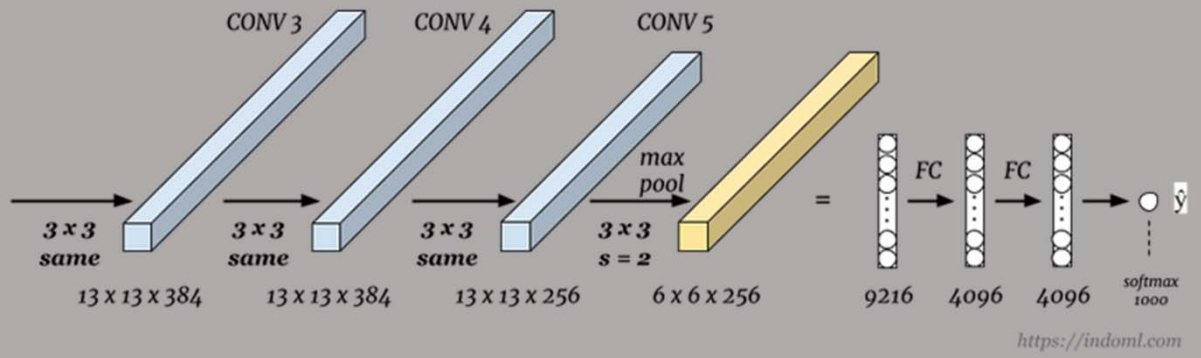
$$W_{ff} = F_{-1} \times F$$

$$B_{ff} = F$$

$$P_{ff} = W_{ff} + B_{ff}$$

F_{-1}: number of FC neurons in previous layer

Example (AlexNet)



- $F_1 = 4096$
- $F_2 = 4096$

$W_{ff} = 4096 \times 4096$

$B_{ff} = 4096$

$\boldsymbol{P_{ff} = 16,781,312}$

- $F_2 = 4096$
- $F_3 = 1000$

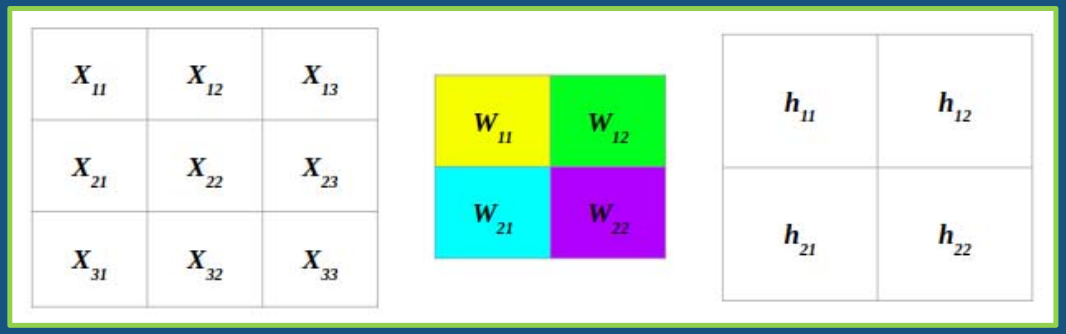
$W_{ff} = 4096 \times 1000$

$B_{ff} = 4096$

$\boldsymbol{P_{ff} = 4,097,000}$

Fully connected layers comprise >90% of parameters

ConvNet backpropagation = dot product accounting

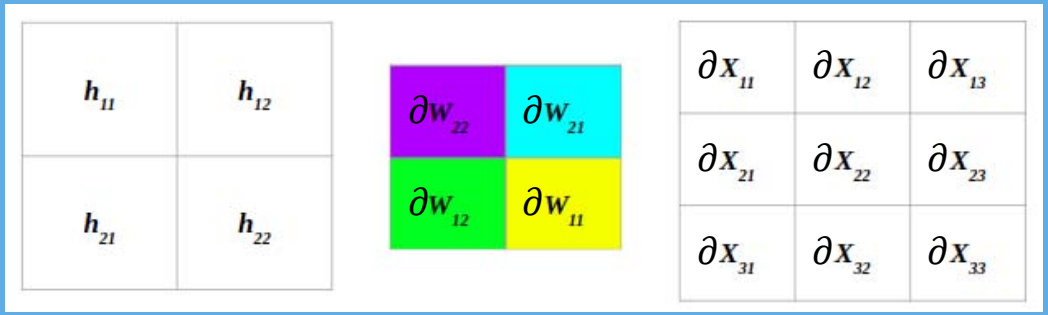


$$h_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{21}x_{21} + w_{22}x_{22}$$
$$h_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{21}x_{22} + w_{22}x_{23}$$
$$h_{21} = w_{11}x_{21} + w_{12}x_{22} + w_{21}x_{31} + w_{22}x_{32}$$
$$h_{22} = w_{11}x_{22} + w_{12}x_{23} + w_{21}x_{32} + w_{22}x_{33}$$

$$\partial w_{11} = x_{11}\partial h_{11} + x_{12}\partial h_{12} + x_{21}\partial h_{21} + x_{22}\partial h_{22}$$
$$\partial w_{12} = x_{12}\partial h_{11} + x_{13}\partial h_{12} + x_{22}\partial h_{21} + x_{23}\partial h_{22}$$
$$\partial w_{21} = x_{21}\partial h_{11} + x_{22}\partial h_{12} + x_{31}\partial h_{21} + x_{32}\partial h_{22}$$
$$\partial w_{22} = x_{22}\partial h_{11} + x_{23}\partial h_{12} + x_{32}\partial h_{21} + x_{33}\partial h_{22}$$

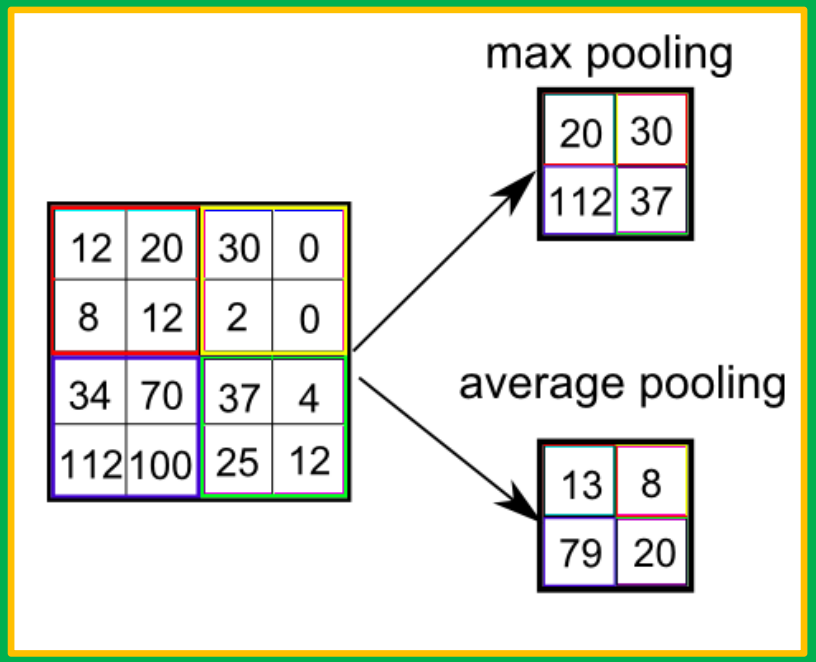
Identifying the *forward set* is the key to back-propagating through a convolution

Backpropagating errors is a “flipped” convolution



Optimizing convolution mechanics means optimizing back propagation as well!

Back-propagating over pooling layers scales error



$$\delta^l = \begin{cases} \delta_j^{l+1} & \text{argmax}_i \, z_i^l \\ 0 & else \end{cases}$$

Assign all δ to the winning input

$$\delta^l = \delta_j^{l+1}/n$$

Spread δ over all inputs

Pooling layers disperse error over the input field

Backpropagation: (you know the rest)

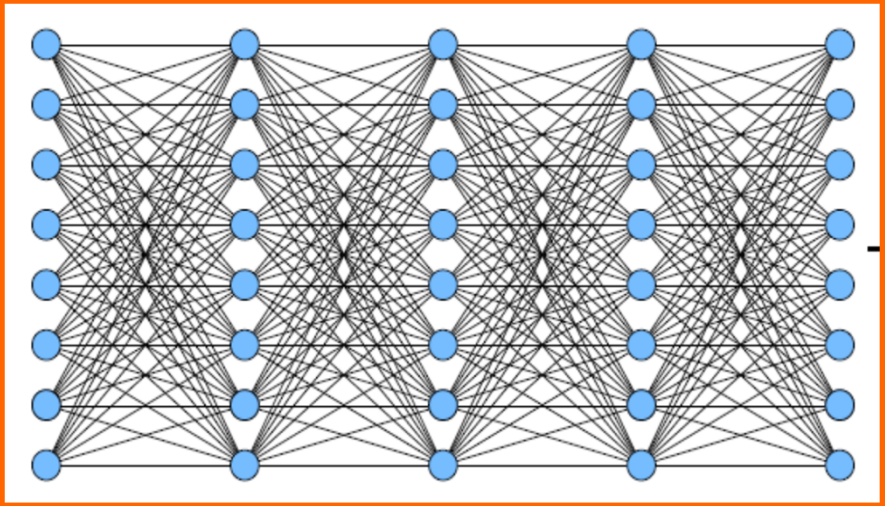
Softmax

$$\delta^{(L)} = a^{(L)} - y^{(L)}$$

ReLU

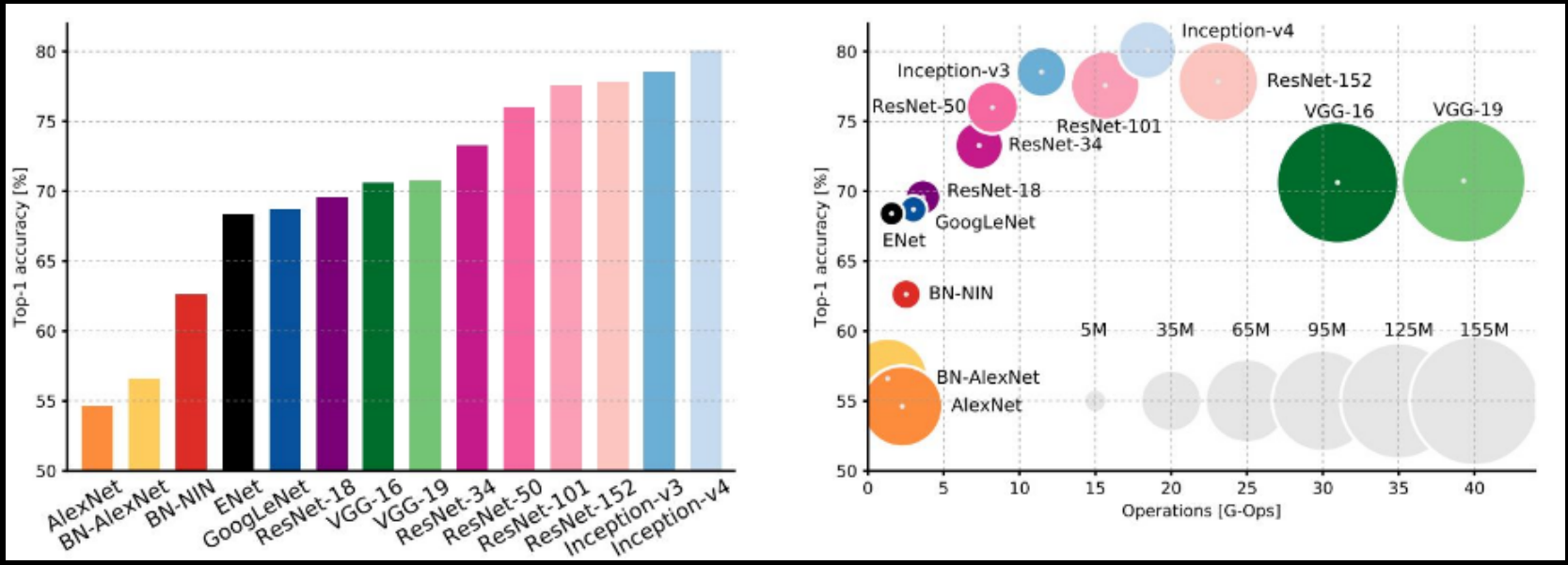
$$\delta^{(L)} = \begin{cases} \delta^{(L+1)} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Fully connected

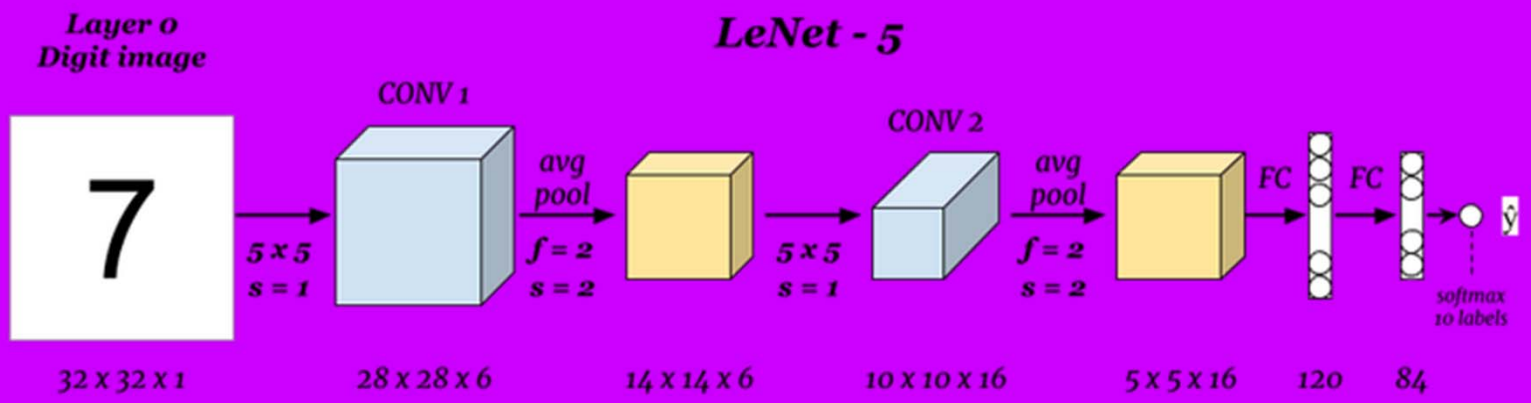


Convolutional Neural Networks

"Case Studies"



LeNet-5 – 1998 (Yan LeCun, AT&T "Bell" Lab)



 # parameters ≈ 60k

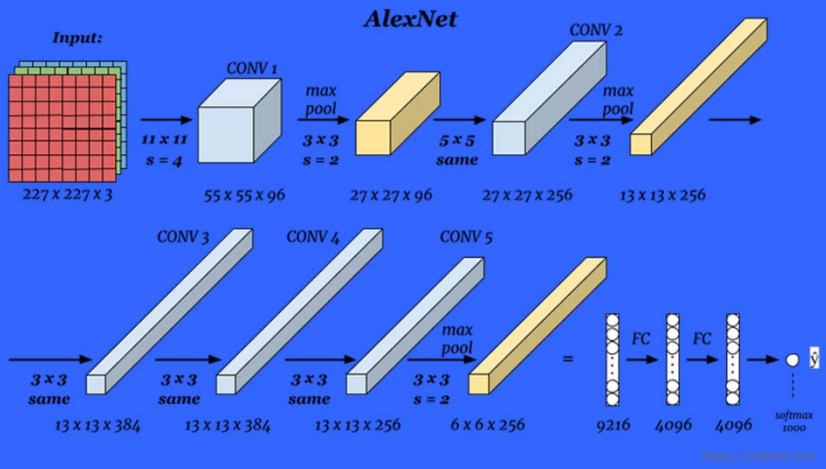
10 Euclidean RBF output neurons

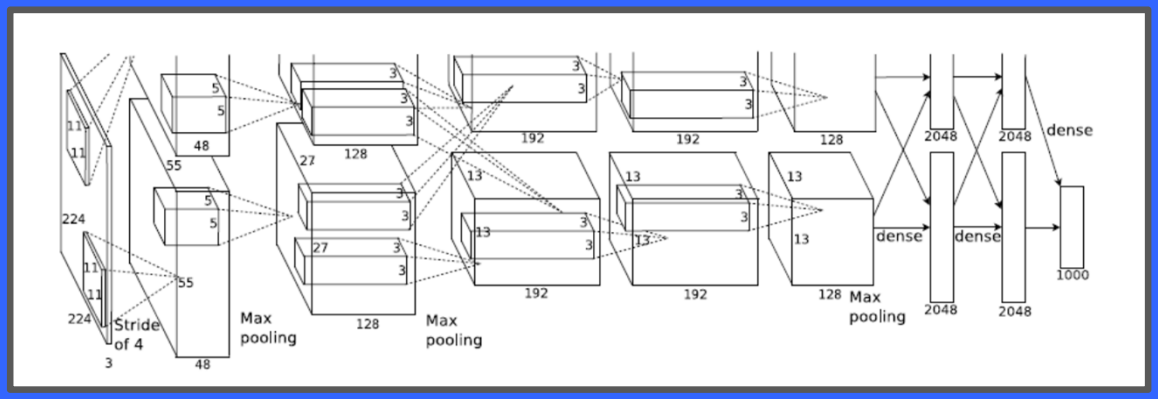
Compute L2 distance between feature vector and **manually predefined** weights vector

< 1% error on MNIST
matched state of the art

Main ideas: convolution, local receptive fields, shared weights, special subsampling

AlexNet – 2012 (Alex Krizhevsky, Univ. of Toronto)





 # parameters ≈ 60M

ReLU decreased training time by a factor of 6x over tanh neurons.

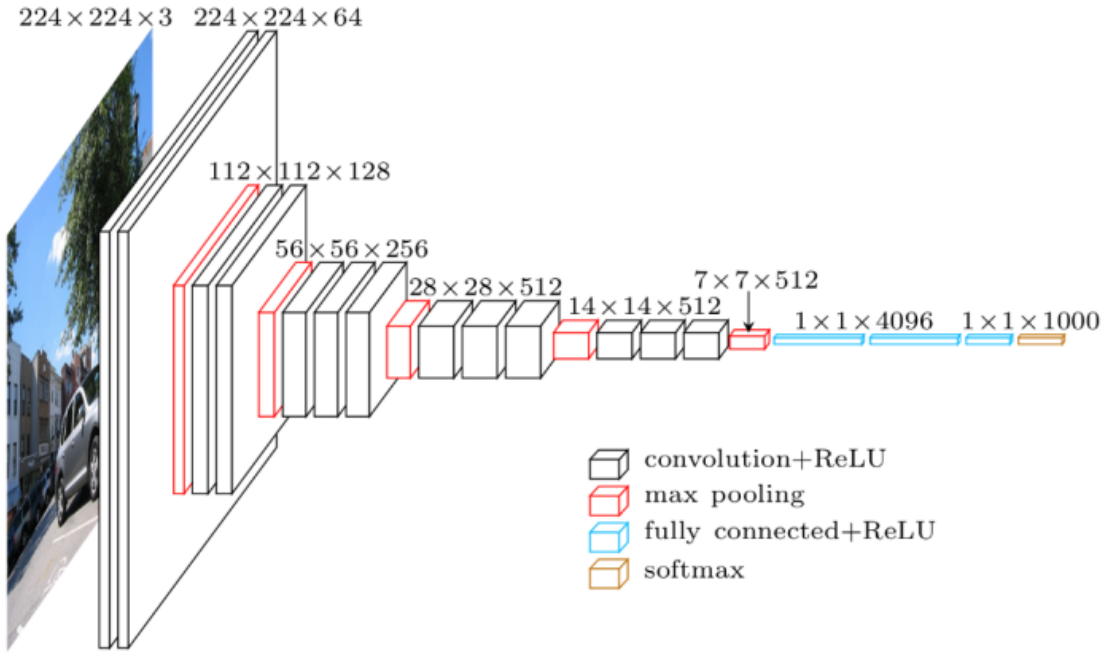
Dropout reduced "complex co-adaptations" of neurons.

Rank	Name	Accuracy
1.	AlexNet	16.4%
2.	SIFT + FVs	26.2%

CNN 10% better than state of the art!

Main ideas: multiple GPU, ReLU nonlinearity, local normalization, overlapping pools, data augmentation, dropout

VGGNet – 2014 (Oxford Visual Geometry Group)



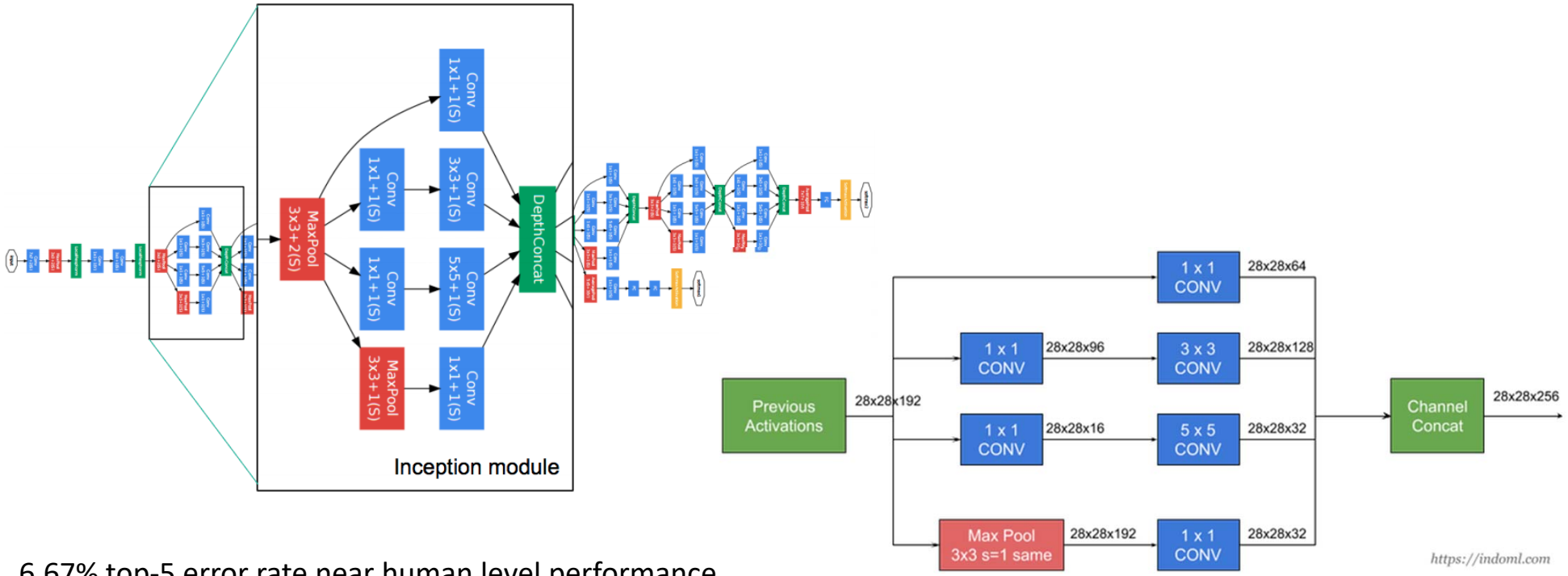
Name:	A	A-LRN	B	C	D	E
# Layers	11	11	13	16	16	19
	C3D64	C3D64	C3D64	C3D64	C3D64	C3D64
		LRN	C3D64	C3D64	C3D64	C3D64
	M	M	M	M	M	M
	C3D128	C3D128	C3D128	C3D128	C3D128	C3D128
			C3D128	C3D128	C3D128	C3D128
	M	M	M	M	M	M
	C3D256	C3D256	C3D256	C3D256	C3D256	C3D256
	C3D256	C3D256	C3D256	C3D256	C3D256	C3D256
				C1D256	C3D256	C3D256
						C3D256
	M	M	M	M	M	M
	C3D512	C3D512	C3D512	C3D512	C3D512	C3D512
	C3D512	C3D512	C3D512	C3D512	C3D512	C3D512
				C1D512	C3D512	C3D512
						C3D512
	M	M	M	M	M	M
	C3D512	C3D512	C3D512	C3D512	C3D512	C3D512
	C3D512	C3D512	C3D512	C3D512	C3D512	C3D512
				C1D512	C3D512	C3D512
						C3D512
	M	M	M	M	M	M
	FC4096	FC4096	FC4096	FC4096	FC4096	FC4096
	FC4096	FC4096	FC4096	FC4096	FC4096	FC4096
	FC1000	FC1000	FC1000	FC1000	FC1000	FC1000
	S	S	S	S	S	S

Original training: 3 weeks on 4 GPUs

 # parameters ≈ 138M

Main ideas: uniform architecture, lots of filters

GoogLeNet (inception unit) – 2015

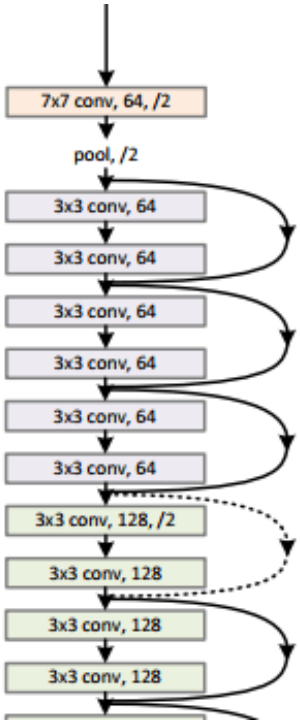


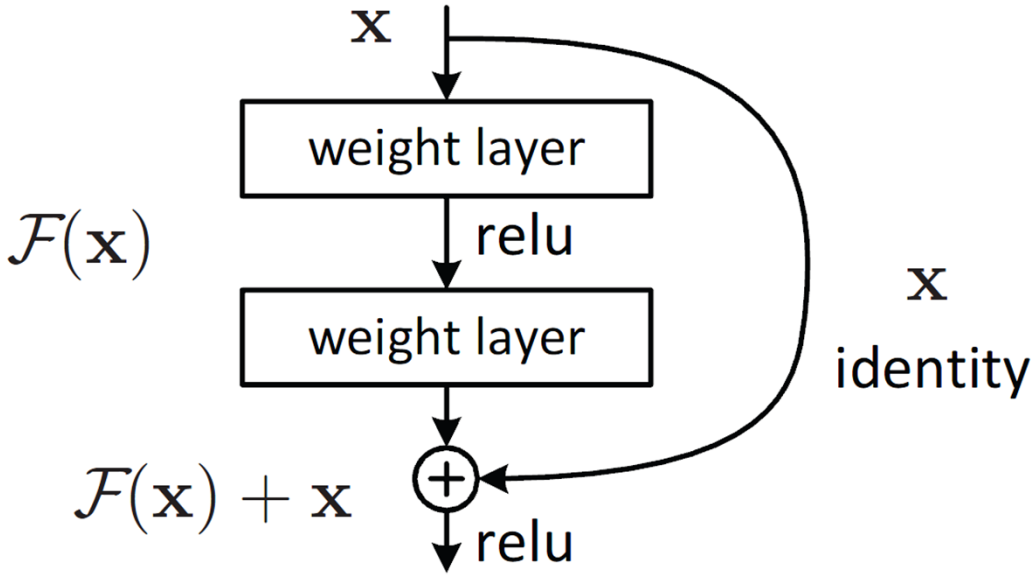
6.67% top-5 error rate near human level performance.

22 layers with only 4 million parameters (compare AlexNet 60M)

Main ideas: small convolutions to reduce parameter count

ResNet (Residual network) – 2015





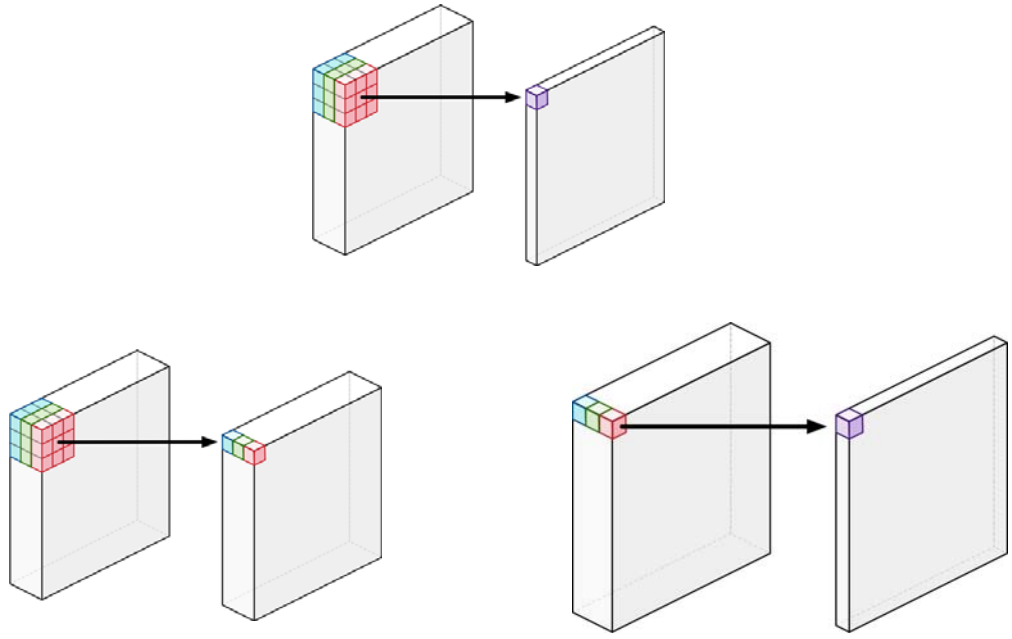
ResNet-152 has 152 layers!

3.57% top-5 error rate beats human level performance.

Main ideas: “skip connections” (gated units) and heavy batch normalization

MobileNet (Google) – 2017

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

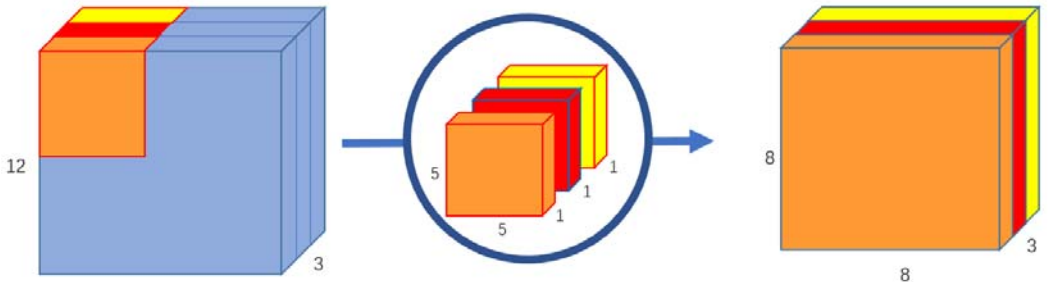


Main ideas: compromise accuracy for computational efficiency, depthwise separable convolution

Depthwise separable convolutions means real time video on a phone

Step 1

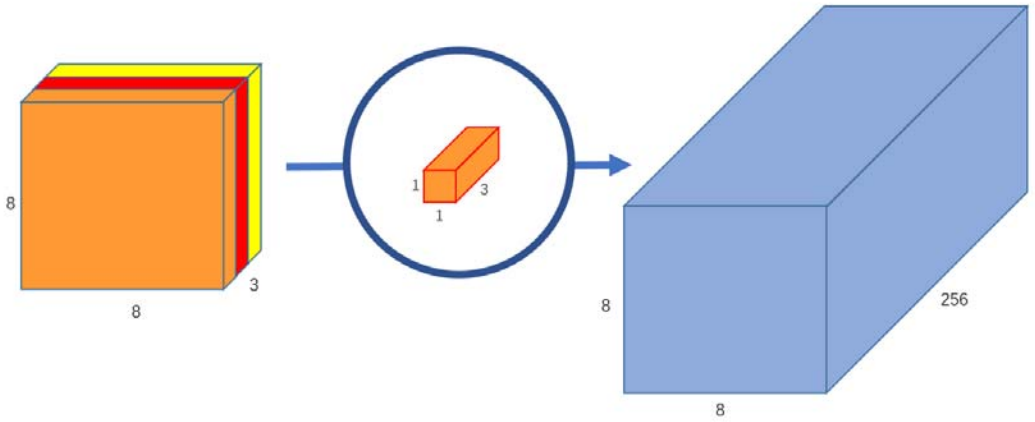
Depthwise convolution



(not all channels!)

Step 2

Pointwise convolution



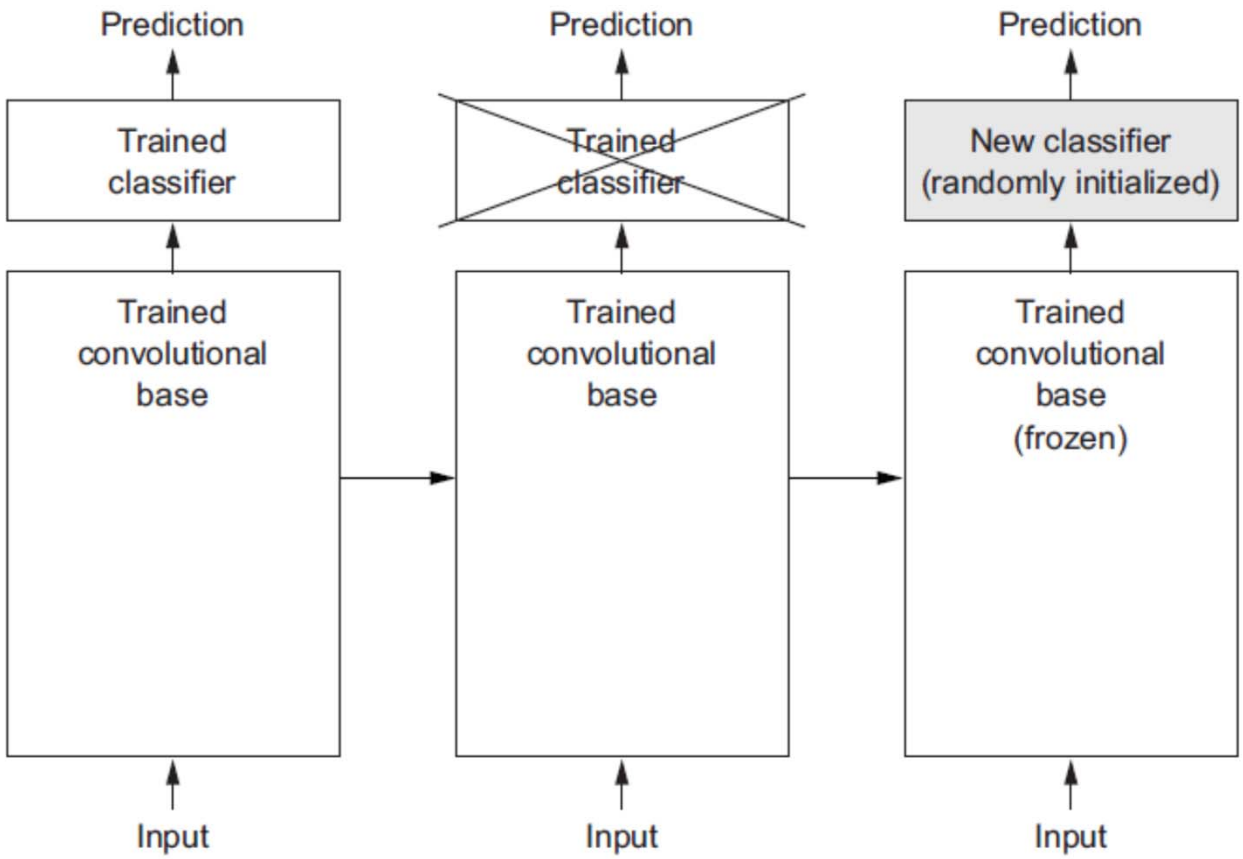
256 $1 \times 1 \times 3$ kernels

"Normal": 256 $5 \times 5 \times 3$ give 8×8 output = 1,228,800 multiplications

Step 1: 3 $5 \times 5 \times 1$ give 8×8 ouput = 4800
Step 2: 256 $1 \times 1 \times 3$ give 8×8 ouput = 49,152

} **DW+S: 52,952 multiplications!**

Pretrained models include useful feature detectors



Replace classifier to learn domain space. Use methodical training to retain learned features