

# Intelligent Systems

**Susana M. Vieira**

Universidade de Lisboa, Instituto Superior Técnico

IS4, Center of Intelligent Systems, IDMEC, LAETA, Portugal

[{susana.vieira}@tecnico.ulisboa.pt](mailto:{susana.vieira}@tecnico.ulisboa.pt)

# Deep Learning

SI8 – Deep Learning for Computer Vision

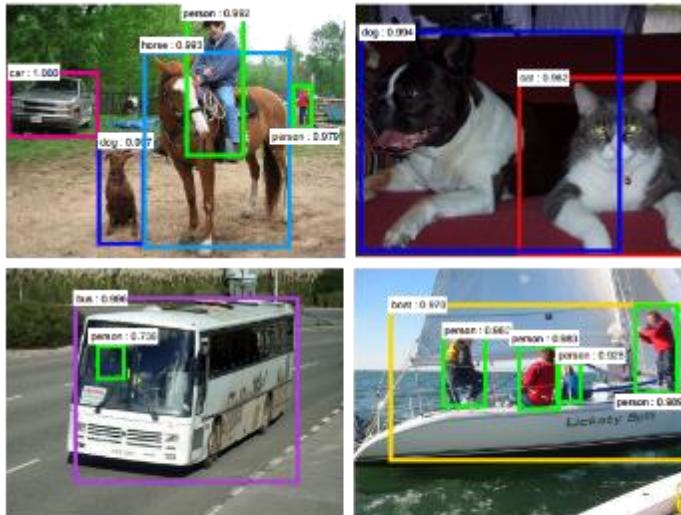
**Reading:**

- Ian Goodfellow. *Deep Learning*. MIT Press, 2016.
- François Chollet. *Deep Learning With Python*. 2<sup>nd</sup> Edition, 2017.

# Deep Learning for Computer Vision

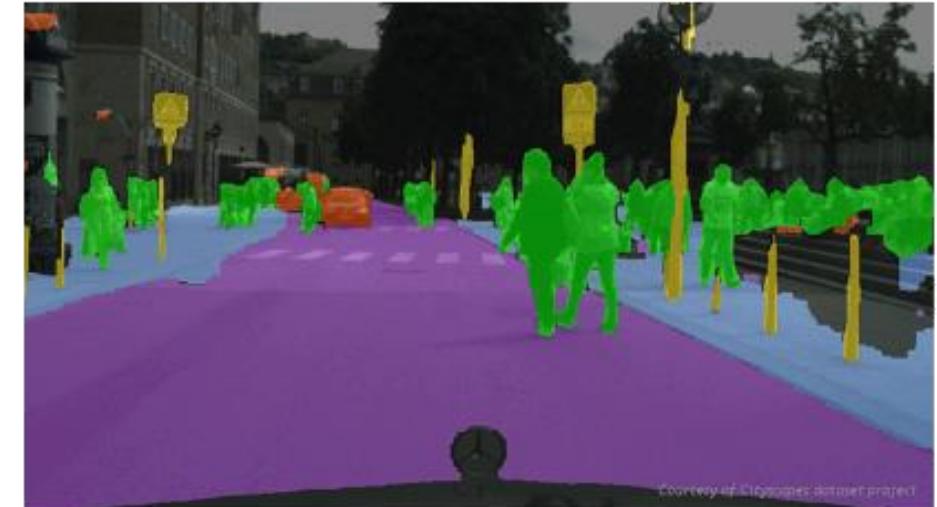
1. Understanding convolutional neural networks (convnets)
2. Using data augmentation to mitigate overfitting
3. Using a pretrained convnet to do feature extraction
4. Fine-tuning a pretrained convnet
5. Visualizing what convnets learn and how they make classification decisions

# Convolutional Networks



[Faster R-CNN - Ren 2015]

[Krizhevsky 2012]



[NVIDIA dev blog]



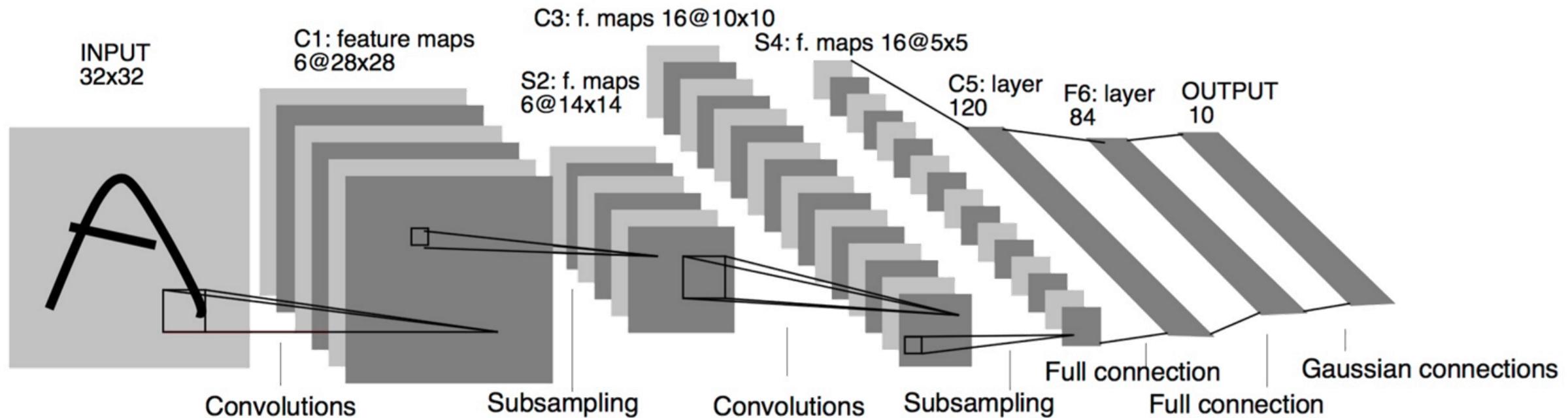
[Ciresan et al. 2013]

# Convolutional Networks

- Other applications
  - Speech recognition & speech synthesis
  - Natural Language Processing
  - Protein/DNA binding prediction
  - Any problem with a spatial (or sequential) structure

# ConvNets for image classification

- CNN = Convolutional Neural Networks = ConvNet



LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition.

# Convolutions: Motivation

- Standard Dense Layer for an image input:

```
x = Input((640, 480, 3), dtype='float32')
# shape of x is: (None, 640, 480, 3)
x = Flatten()(x)
# shape of x is: (None, 640 x 480 x 3)
z = Dense(1000)(x)
```

- How many parameters in the Dense layer?

$$640 \times 480 \times 3 \times 1000 + 1000 = 922M!$$

Spatial organization of the input is destroyed by Flatten

We never use Dense layers directly on large images. Most standard solution is convolution layers

# Fully Connected Network: MLP

```
input_image = Input(shape=(28, 28, 1))
x = Flatten()(input_image)
x = Dense(256, activation='relu')(x)
x = Dense(10, activation='softmax')(x)
mlp = Model(inputs=input_image, outputs=x)
```

# Convolutional Network:

```
input_image = Input(shape=(28, 28, 1))
x = Conv2D(32, 5, activation='relu')(input_image)
x = MaxPool2D(2, strides=2)(x)
x = Conv2D(64, 3, activation='relu')(x)
x = MaxPool2D(2, strides=2)(x)
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dense(10, activation='softmax')(x)
convnet = Model(inputs=input_image, outputs=x)
```

2D spatial organization of features preserved until Flatten.

# Convolutional Networks

- Scale up neural networks to process very large images /video sequences
  - Sparse connections
  - Parameter sharing
- Automatically generalize across spatial translations of inputs
- Applicable to any input that is laid out on a grid (1-D, 2-D,
- 3-D, ...)

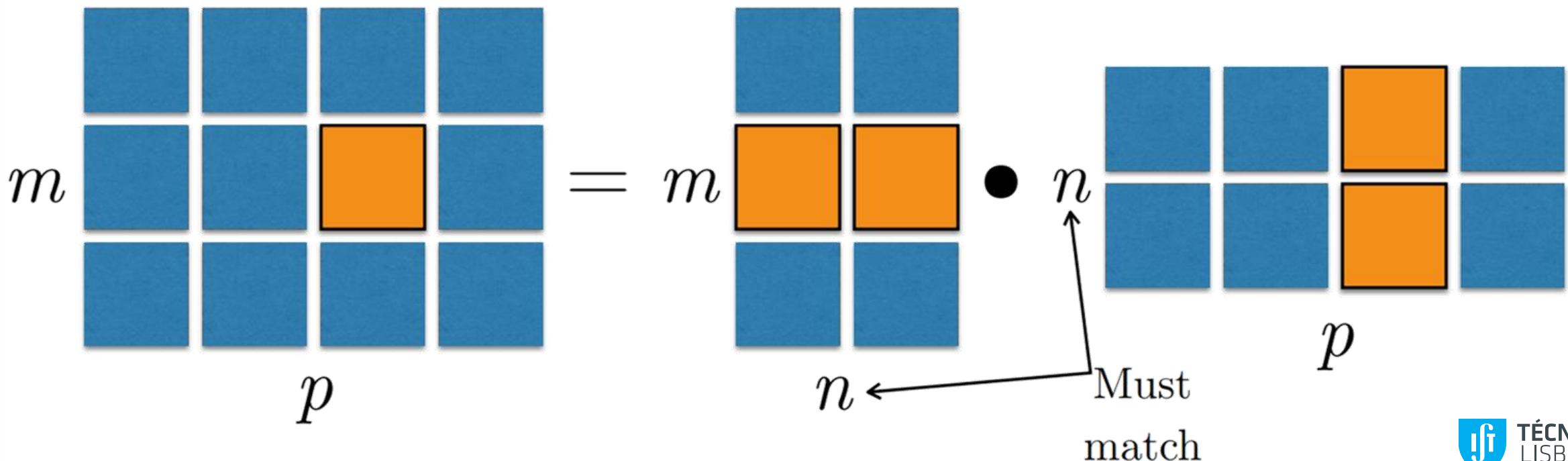
# Key Idea

- Replace matrix multiplication in neural nets with convolution
- Everything else stays the same
  - Maximum likelihood
  - Back-propagation
  - etc

# Matrix (Dot) Product

$$C = AB$$

$$C_{i,j} = \sum_k A_{i,k} B_{k,j}$$



# Matrix Transpose

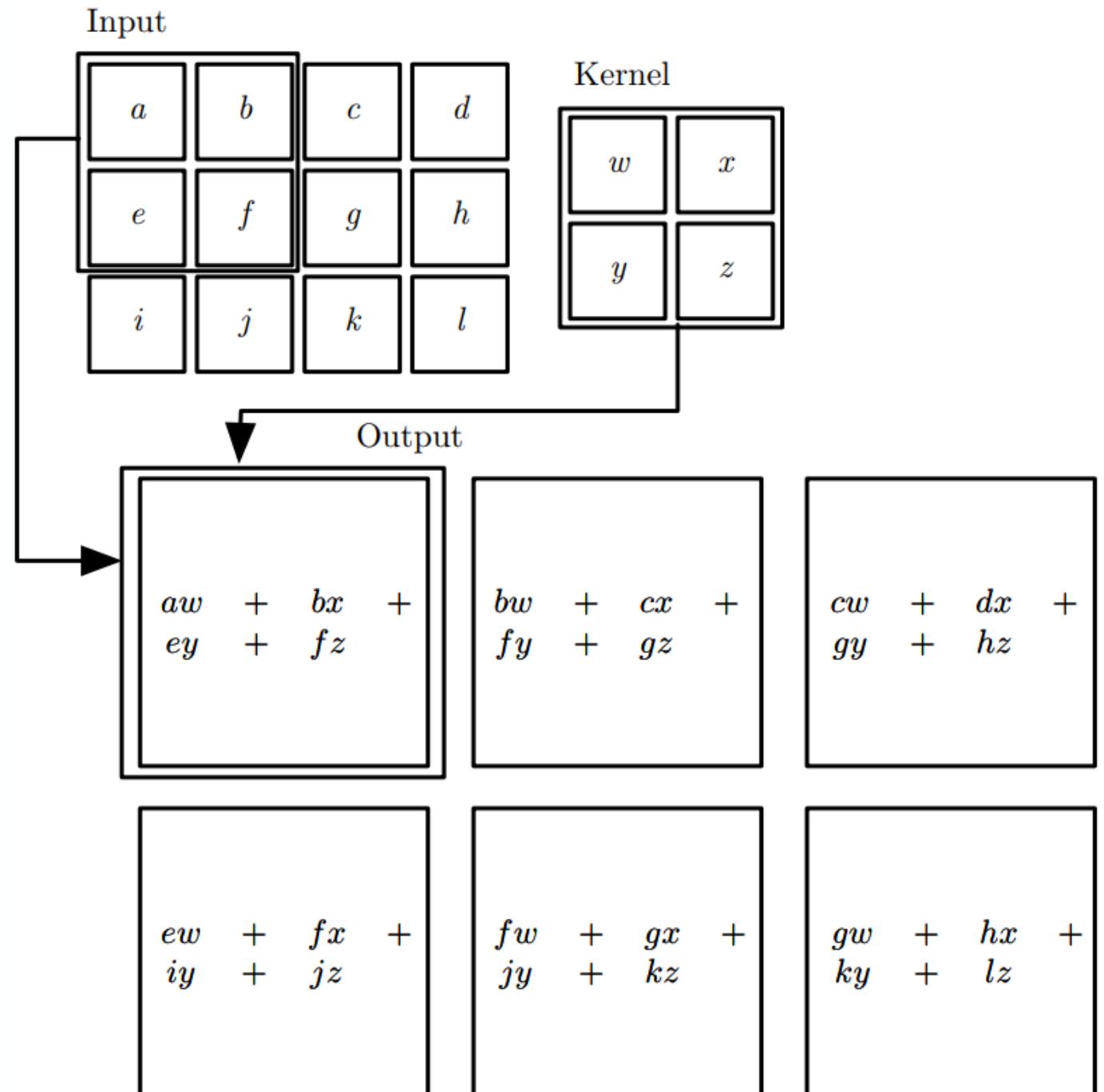
$$(A^T)_{i,j} = A_{j,i}$$

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \rightarrow A^T = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$

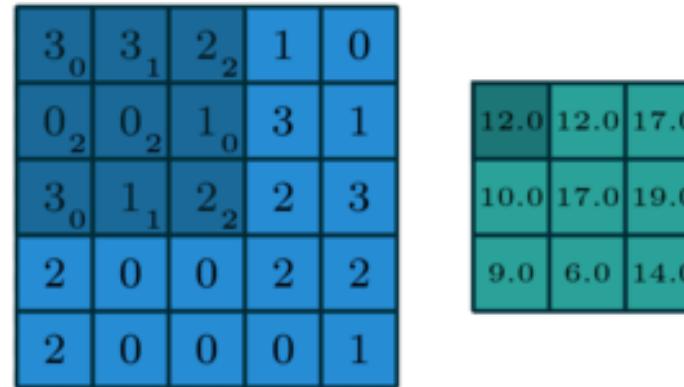
The transpose of the matrix can be thought of as a mirror image across the main diagonal

$$(AB)^T = B^T A^T.$$

# 2D Convolution



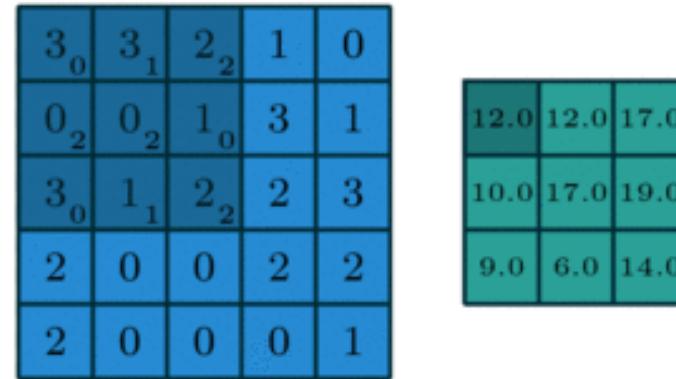
# 2D Convolution



- $x$  is a  $3 \times 3$  chunk (dark area) of the image (blue array)
- Each output neuron is parametrized with the  $3 \times 3$  weight matrix  $w$  (kernel: small numbers)

These slides extensively use convolution visualization by V. Dumoulin available at  
[https://github.com/vdumoulin/conv\\_arithmetic](https://github.com/vdumoulin/conv_arithmetic)

# 2D Convolution



- $x$  is a  $3 \times 3$  chunk (dark area) of the image (blue array)
- Each output neuron is parametrized with the  $3 \times 3$  weight matrix  $w$  (kernel: small numbers)
- The activation obtained by sliding the  $3 \times 3$  window and computing:

$$Z(x) = \text{relu}(\mathbf{w}^T x + b)$$

# Why Convolution

Discrete convolution (actually cross-correlation) between two functions  $f$  and  $g$ :

$$(f \star g)(x) = \sum_{a+b=x} f(a) \cdot g(b) = \sum_a f(a) \cdot g(x+a)$$

2D-convolutions (actually 2D cross-correlation):

$$(f \star g)(x, y) = \sum_n \sum_m f(n, m) \cdot g(x+n, y+m)$$

$f$  is a convolution kernel or filter applied to the 2-d map  $g$  (our image)

# Example: Convolution Image

- Image:  $im$  of dimensions  $5 \times 5$
- Kernel:  $k$  of dimensions  $3 \times 3$

$$(k \star im)(x, y) = \sum_{n=0}^2 \sum_{m=0}^2 k(n, m) \cdot im(x + n - 1, y + m - 1)$$

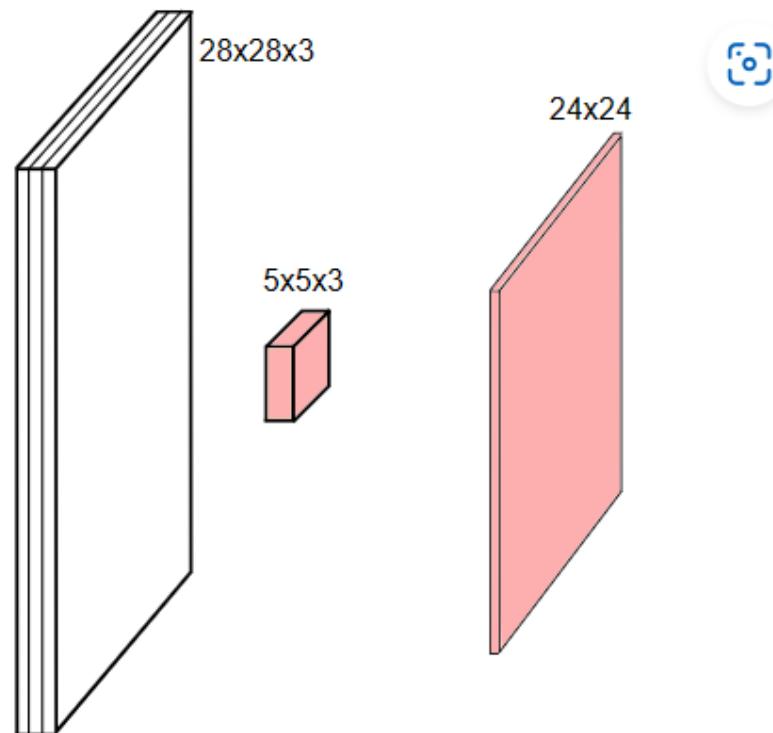
$3_0$	$3_1$	$2_2$	1	0
$0_2$	$0_2$	$1_0$	3	1
$3_0$	$1_1$	$2_2$	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

# Channels

Colored image = tensor of shape (height, width, channels)

Convolutions are usually computed for each channel and summed:



$$(k * im^{color}) = \sum_{c=0}^2 k^c * im^c$$

# Three Operations

- **Convolution:** like matrix multiplication
  - Take an input, produce an output (hidden layer)
- **Deconvolution:** like multiplication by transpose of a matrix
  - Used to back-propagate error from output to input
  - Reconstruction in autoencoder / RBM
- **Weight gradient computation**
  - Used to backpropagate error from output to weights
  - Accounts for the parameter sharing

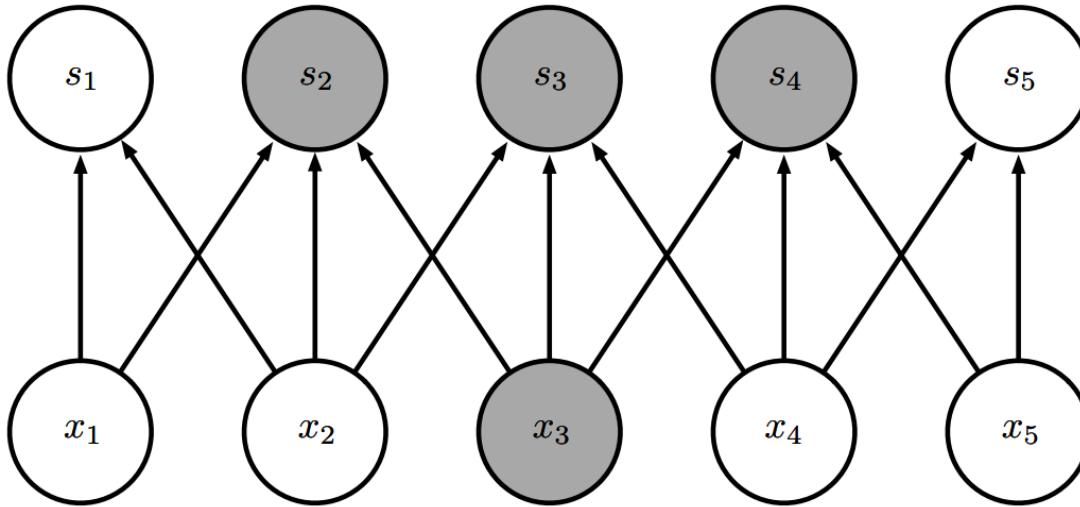
# Sparse Connectivity

- **Local connectivity**
  - A neuron depends only on a few local input neurons
  - Translation invariance
- **Comparison to Fully connected**
  - Parameter sharing: reduce overfitting
  - Make use of spatial structure: **strong prior** for vision!
- **Animal Vision Analogy**

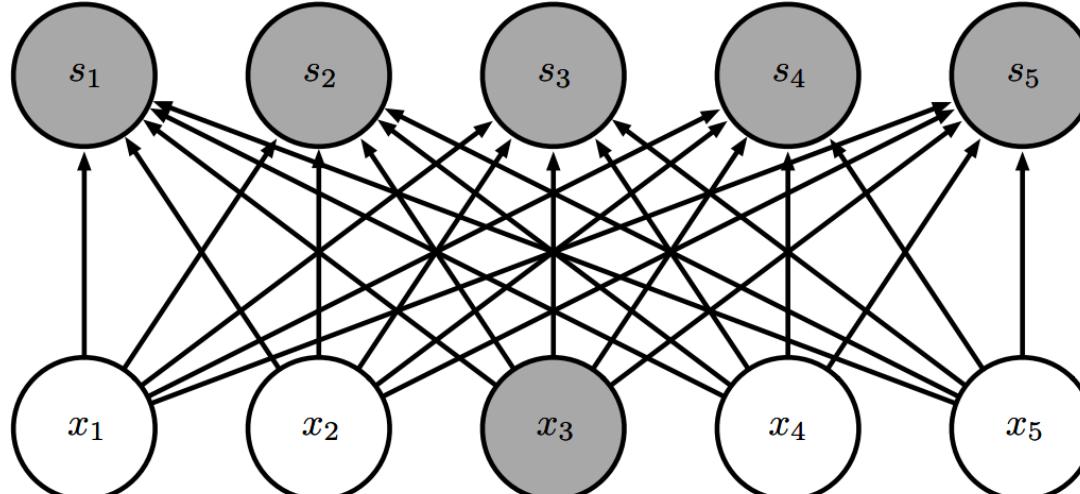
Hubel & Wiesel, RECEPTIVE FIELDS OF SINGLE NEURONES IN THE CAT'S STRIATE CORTEX (1959)

# Sparse Connectivity

Sparse  
connections due  
to small  
convolution kernel

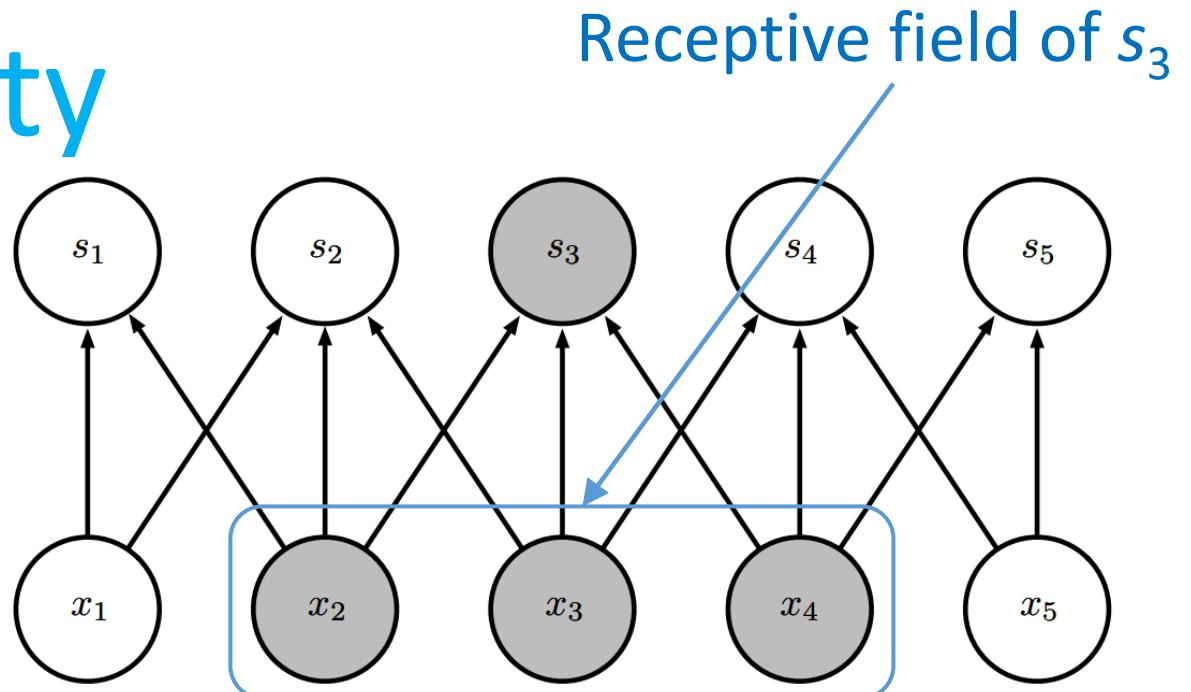


Dense  
connections

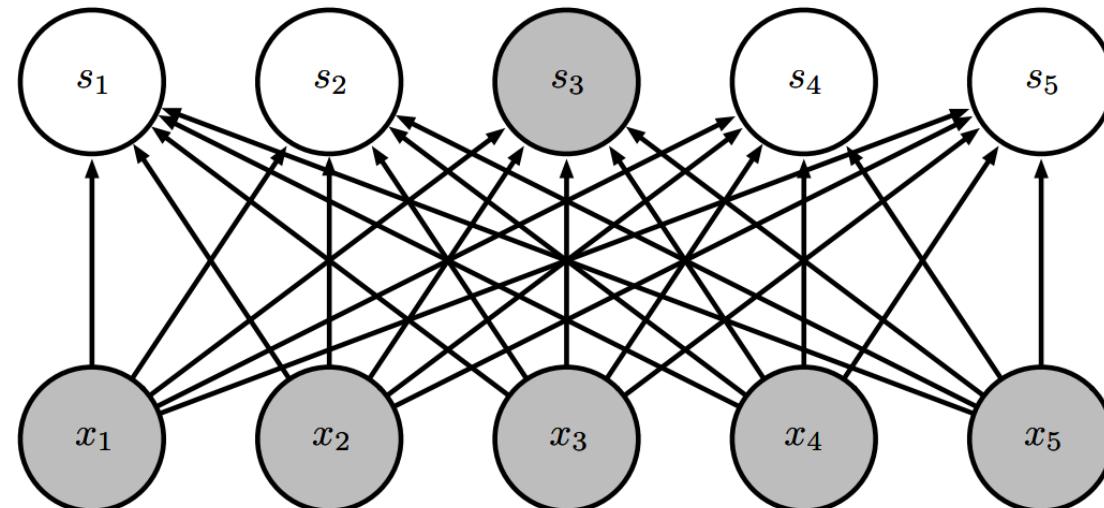


# Sparse Connectivity

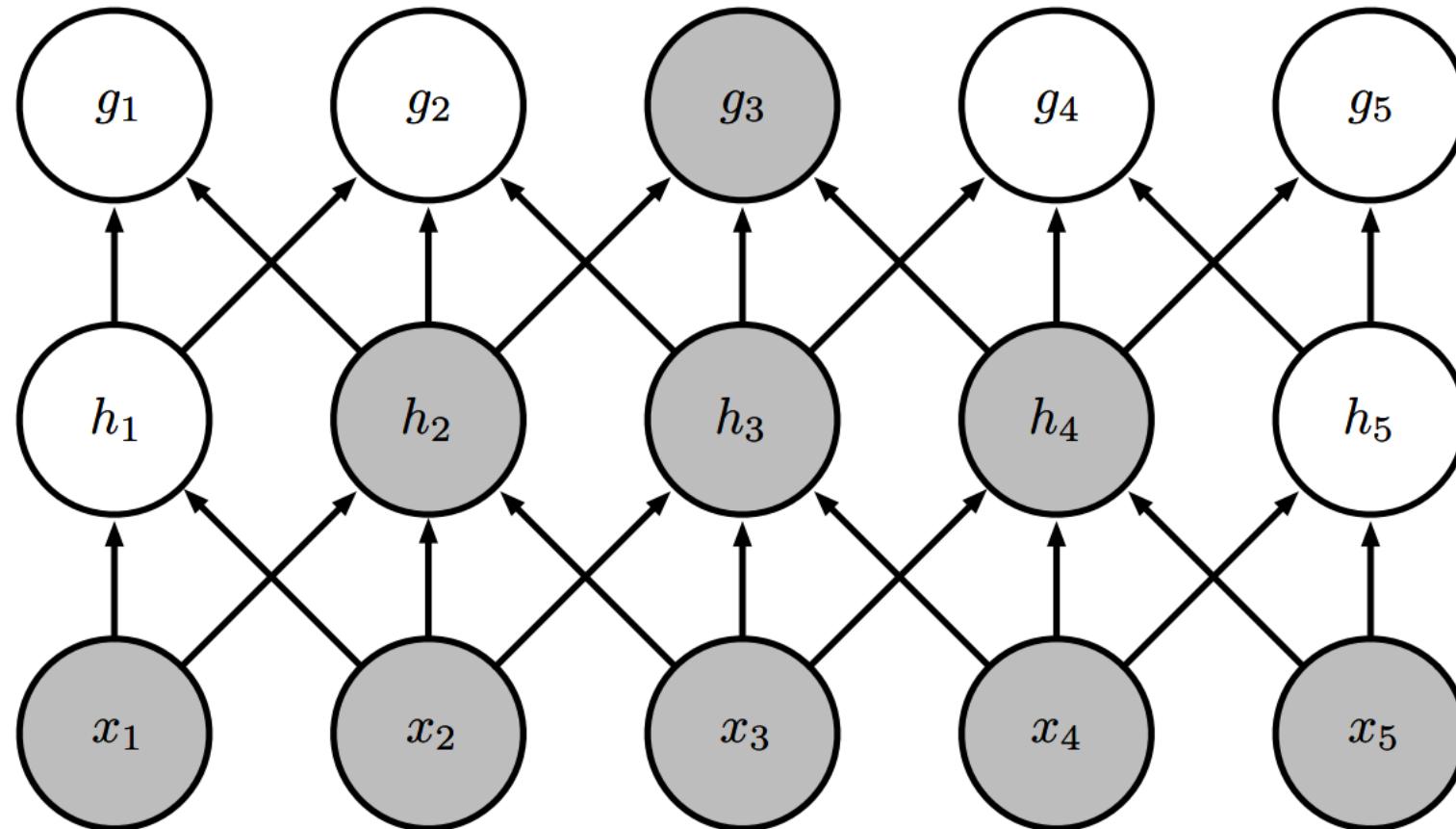
Sparse  
connections due  
to small  
convolution kernel



Dense  
connections

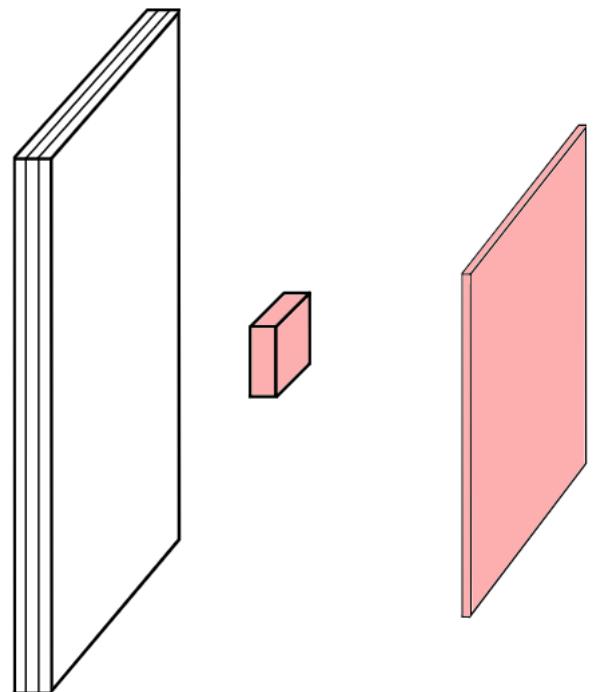


# Growing Receptive Fields

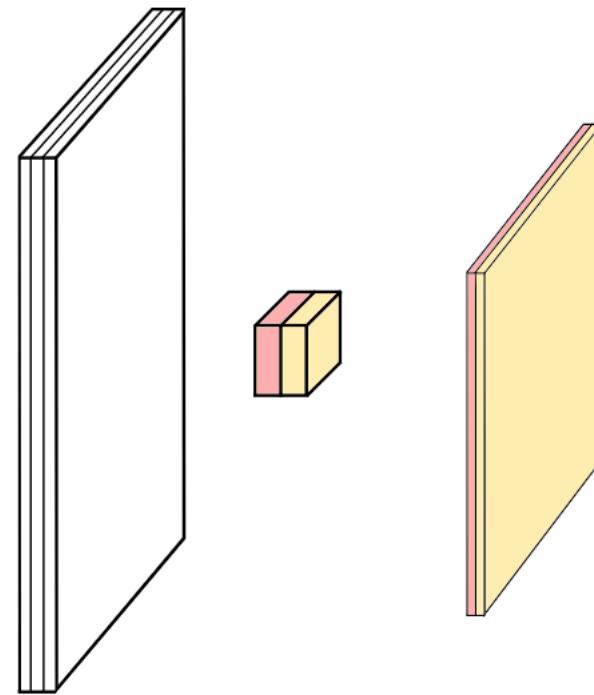


Receptive field of units in deeper layers of a convnet is larger than of units in shallow layers

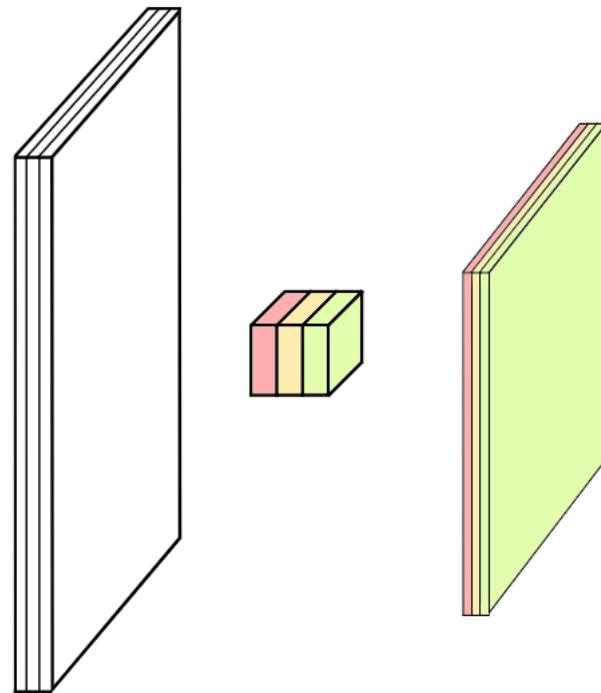
# Multiple Convolutions



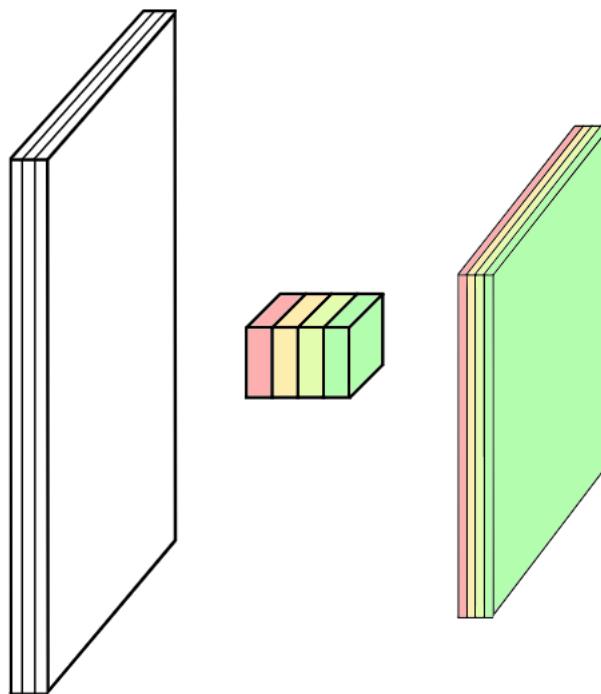
# Multiple Convolutions



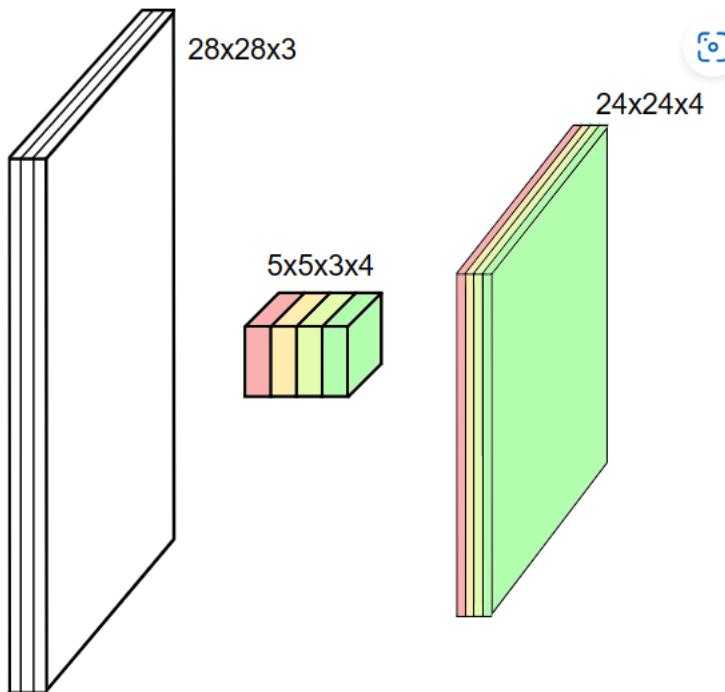
# Multiple Convolutions



# Multiple Convolutions



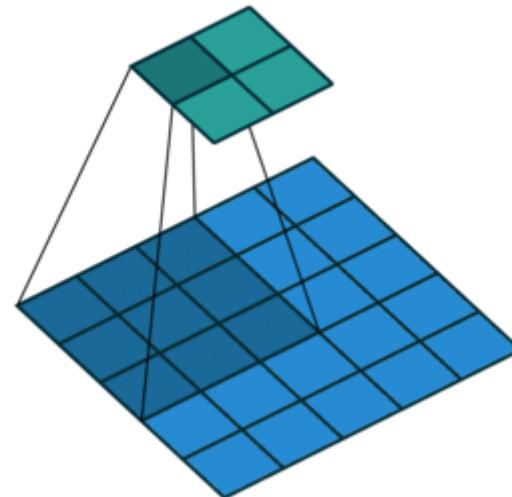
# Multiple Convolutions



- Kernel size aka receptive field (usually 1, 3, 5, 7, 11)
- Output dimension:  $\text{length} - \text{kernel\_size} + 1$

# Stride

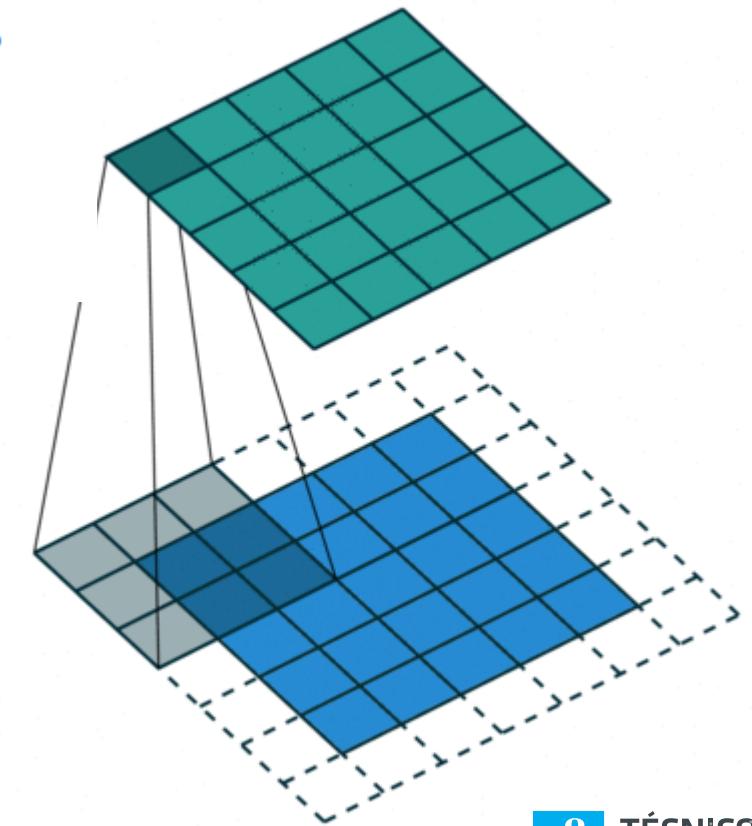
- Strides: increment step size for the convolution operator
- Reduces the size of the output map



Example with kernel size  $3 \times 3$  and a stride of 2 (image in blue)

# Padding

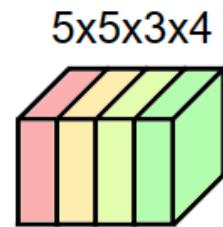
- Padding: artificially fill borders of image
- Useful to keep spatial dimension constant across filters
- Useful with strides and large receptive fields
- Usually: fill with 0s



# Dealing with Shapes

**Kernel or Filter shape**  $(F, F, C^i, C^o)$

- $F \times F$  kernel size,
- $C^i$  input channels
- $C^o$  output channels



Number of parameters:  $(F \times F \times C^i + 1) \times C^o$

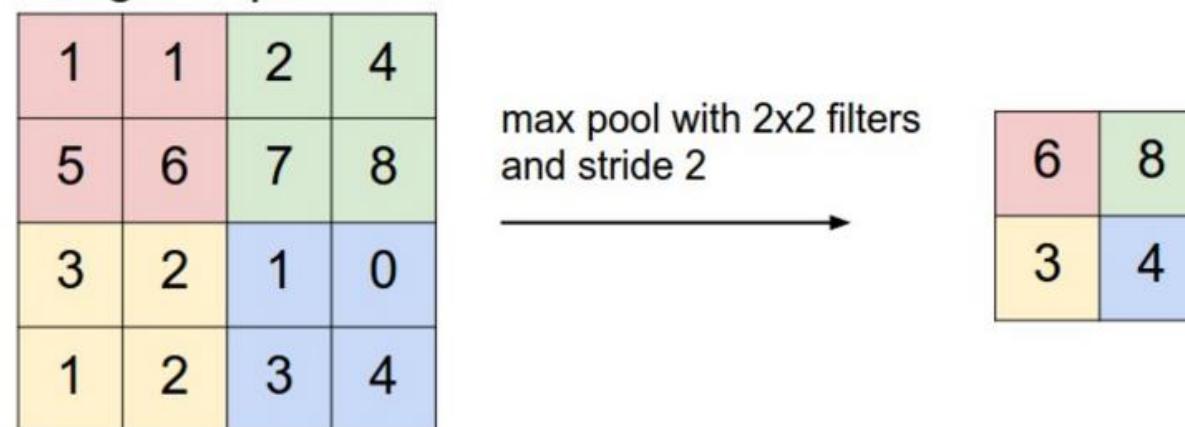
**Activations or Feature maps shape:**

- Input  $(W^i, H^i, C^i)$
- Output  $(W^o, H^o, C^o)$

$$W^o = (W^i - F + 2P)/S + 1$$

# Pooling

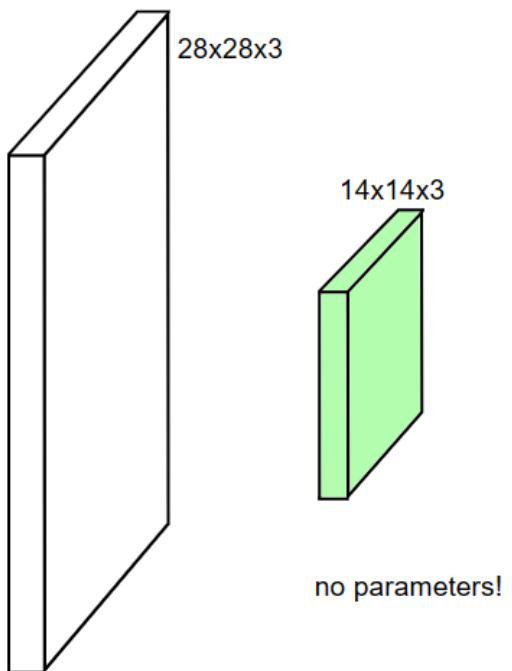
- Spatial dimension reduction
- Local invariance
- No parameters: max or average of 2x2 units



Schematic from Stanford <http://cs231n.github.io/convolutional-networks>

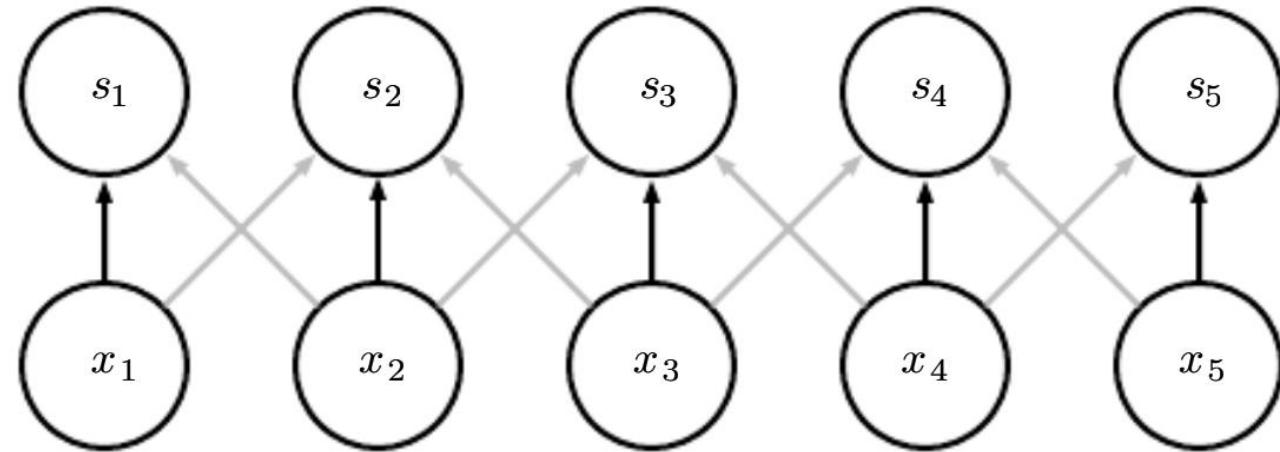
# Pooling

- Spatial dimension reduction
- Local invariance
- No parameters: max or average of  $2 \times 2$  units

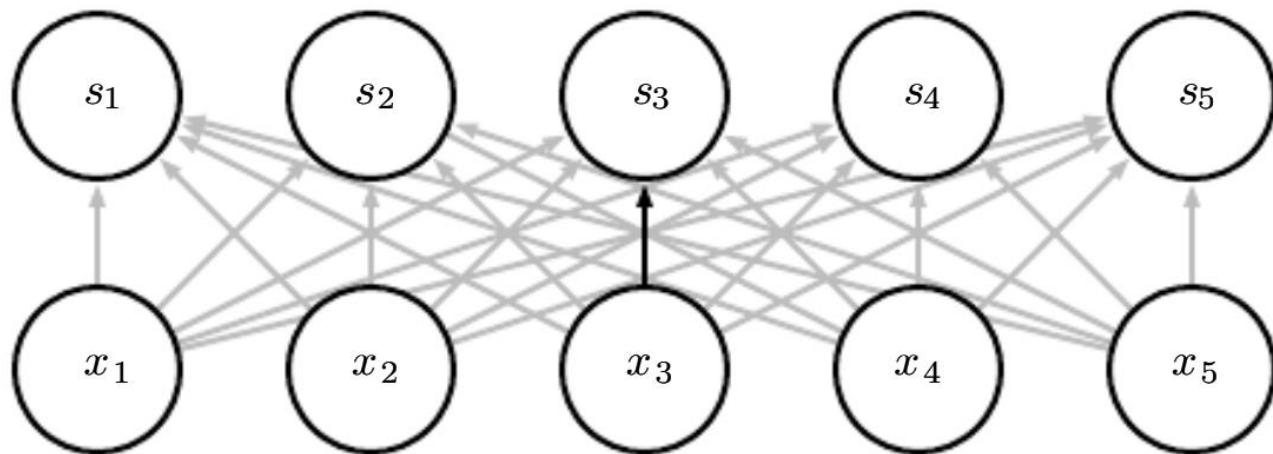


# Parameter Sharing

Convolution shares the same parameters across all spatial locations



Traditional matrix multiplication does not share any parameters

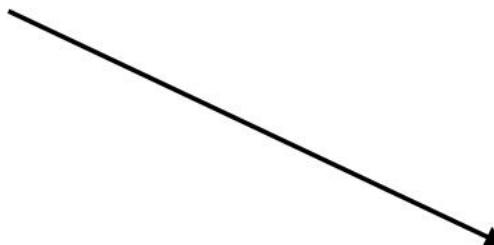


The black arrows indicate uses of the central element of a 3-element kernel in a convolutional model

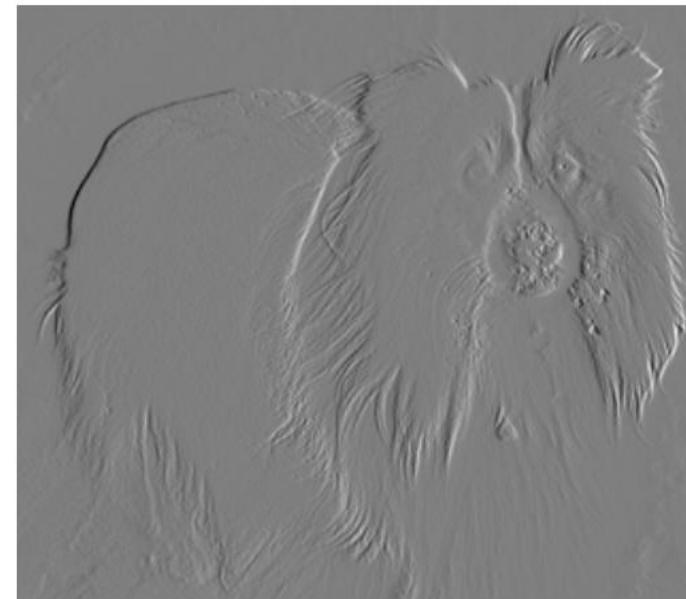
# Edge Detection by Convolution



Input



1	-1
---	----



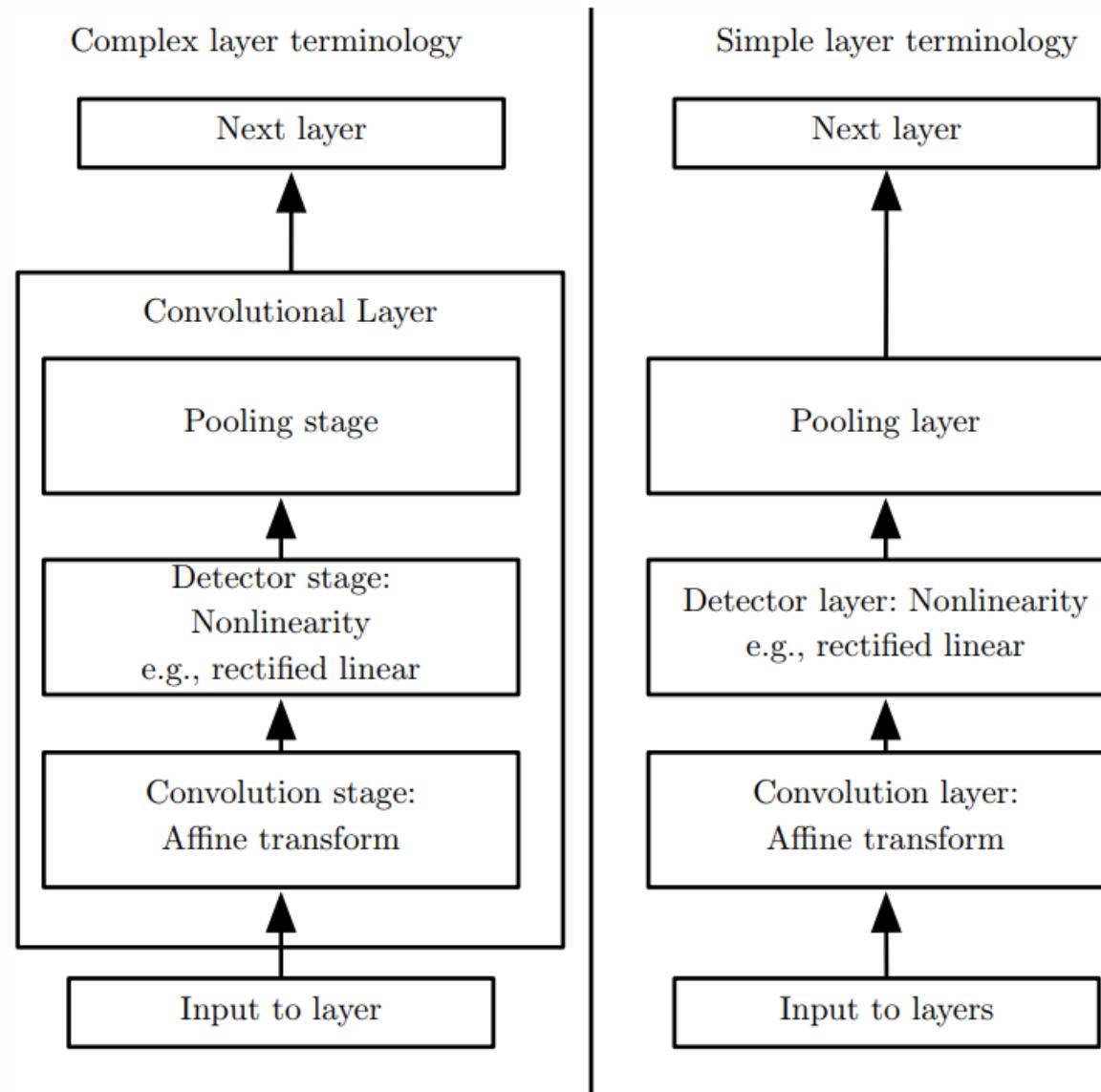
Output

# Efficiency of Convolution

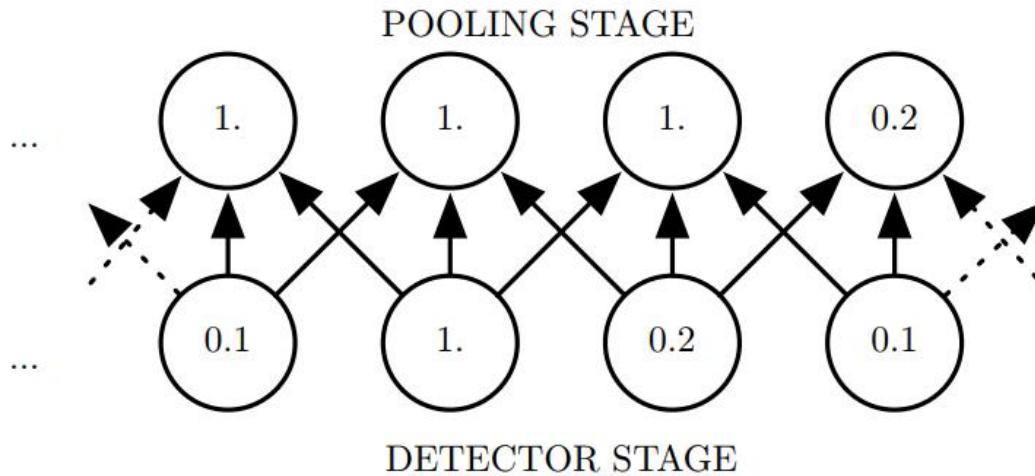
- Input size: 320 by 280
- Kernel size: 2 by 1
- Output size: 319 by 280

	Convolution	Dense matrix	Sparse matrix
Stored floats	2	$319*280*320*280$ $> 8e9$	$2*319*280 =$ 178,640
Float muls or adds	$319*280*3 =$ 267,960	$> 16e9$	Same as convolution (267,960)

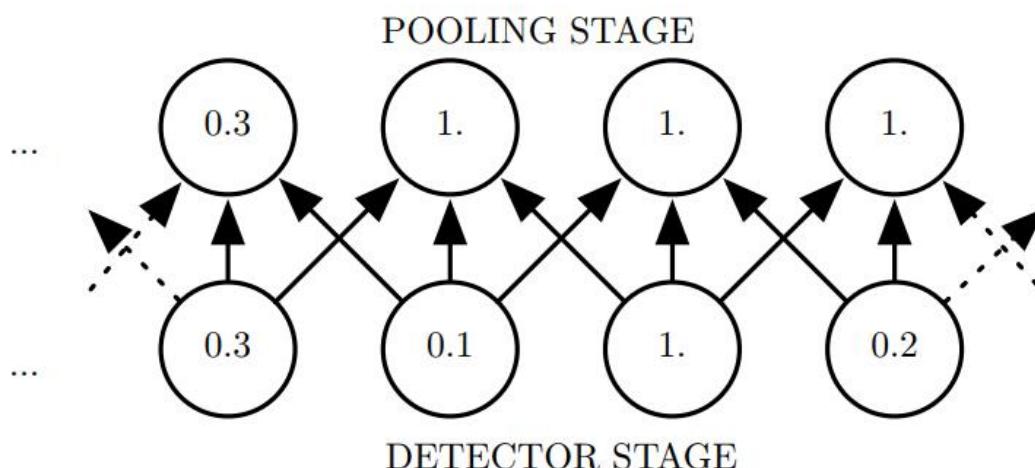
# Convolutional Network Components



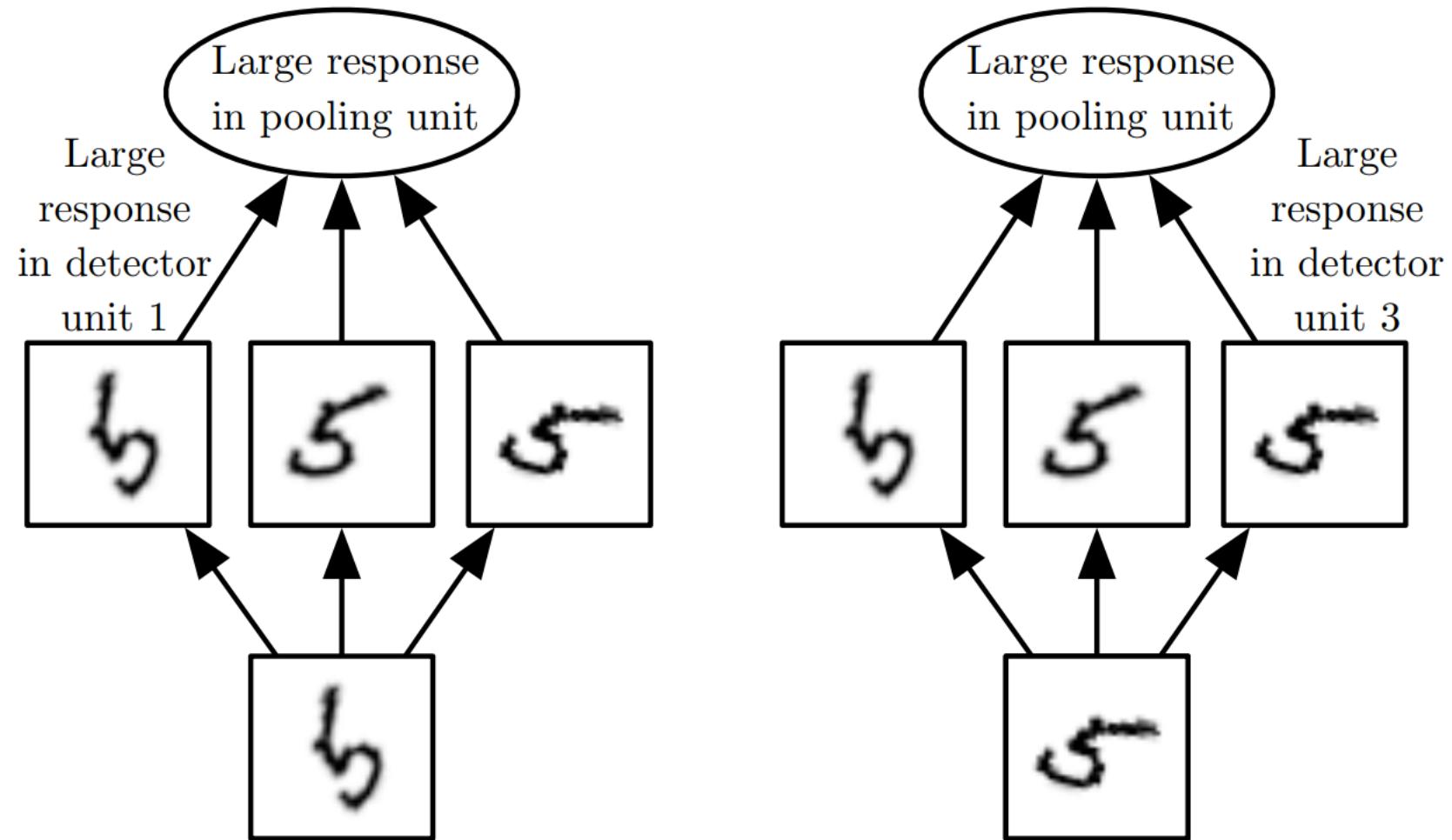
# Max Pooling and Invariance to Translation



A view of the middle of the output of a convolutional layer. The bottom row shows outputs of the nonlinearity. The top row shows the outputs of **max pooling**, with a stride of one pixel between pooling regions and a pooling region width of three pixels.



# Cross-Channel Pooling and Invariance to Learned Transformations



# Classic ConvNet Architecture

Input

Conv blocks

- Convolution + activation (relu)
- Convolution + activation (relu)
- ...
- Maxpooling 2x2

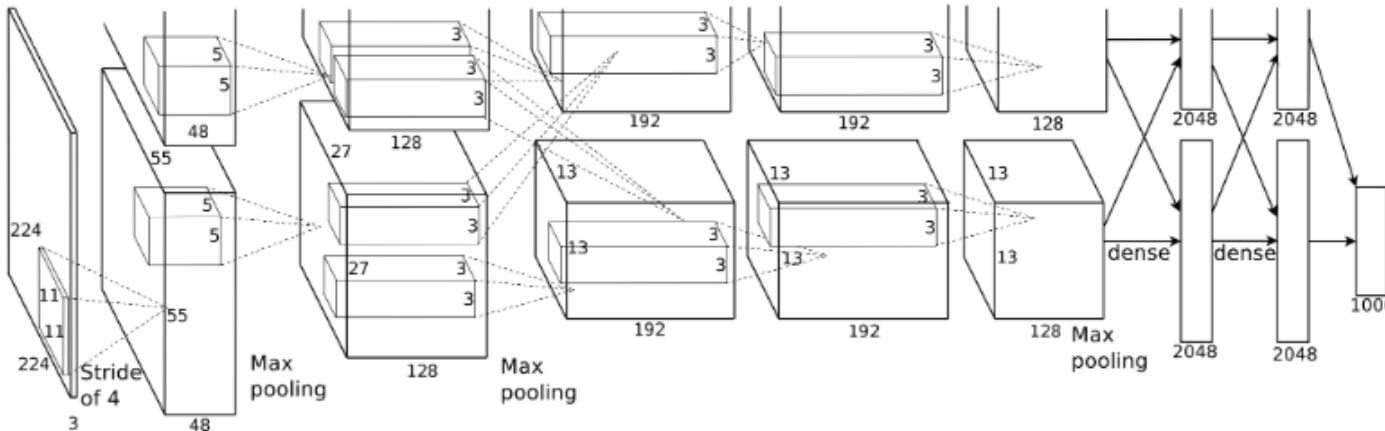
Output

- Fully connected layers
- Softmax

# Major Architectures

- **Spatial Transducer Net:** input size scales with output size, all layers are convolutional
- **All Convolutional Net:** no pooling layers, just use strided convolution to shrink representation size
- **Inception:** complicated architecture designed to achieve high accuracy with low computational cost
- **ResNet:** blocks of layers with same spatial size, with each layer's output added to the same buffer that is repeatedly updated. Very many updates = very deep net, but without vanishing gradient.

# AlexNet

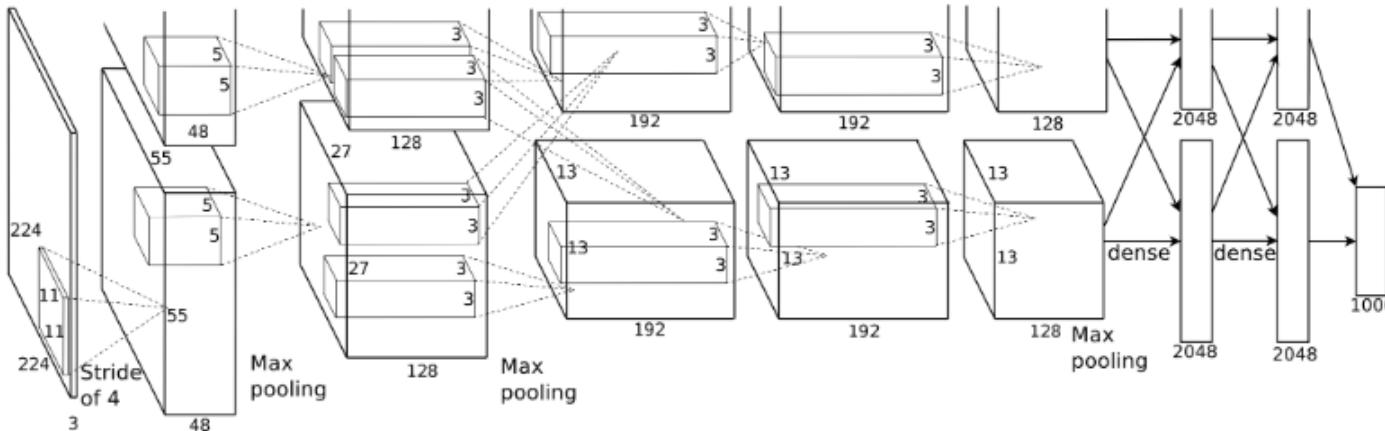


First conv layer: kernel 11x11x3x96 stride 4

- Kernel shape: (11, 11, 3, 96)
- Output shape: (55, 55, 96)
- Number of parameters: 34,944
- Equivalent MLP parameters:  $43.7 \times 1e9$

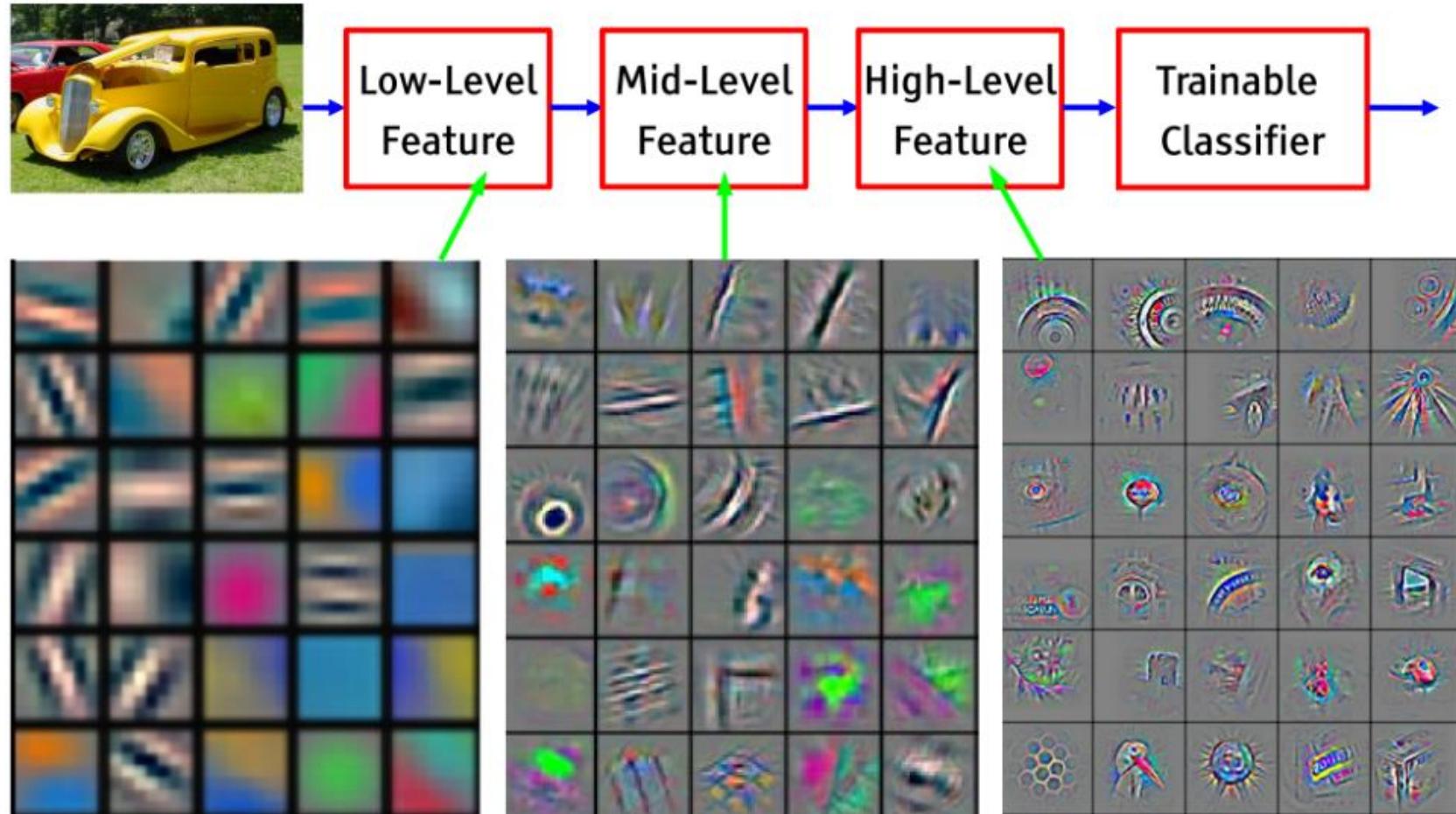
Simplified version of Krizhevsky, Alex, Sutskever, and Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012

# AlexNet



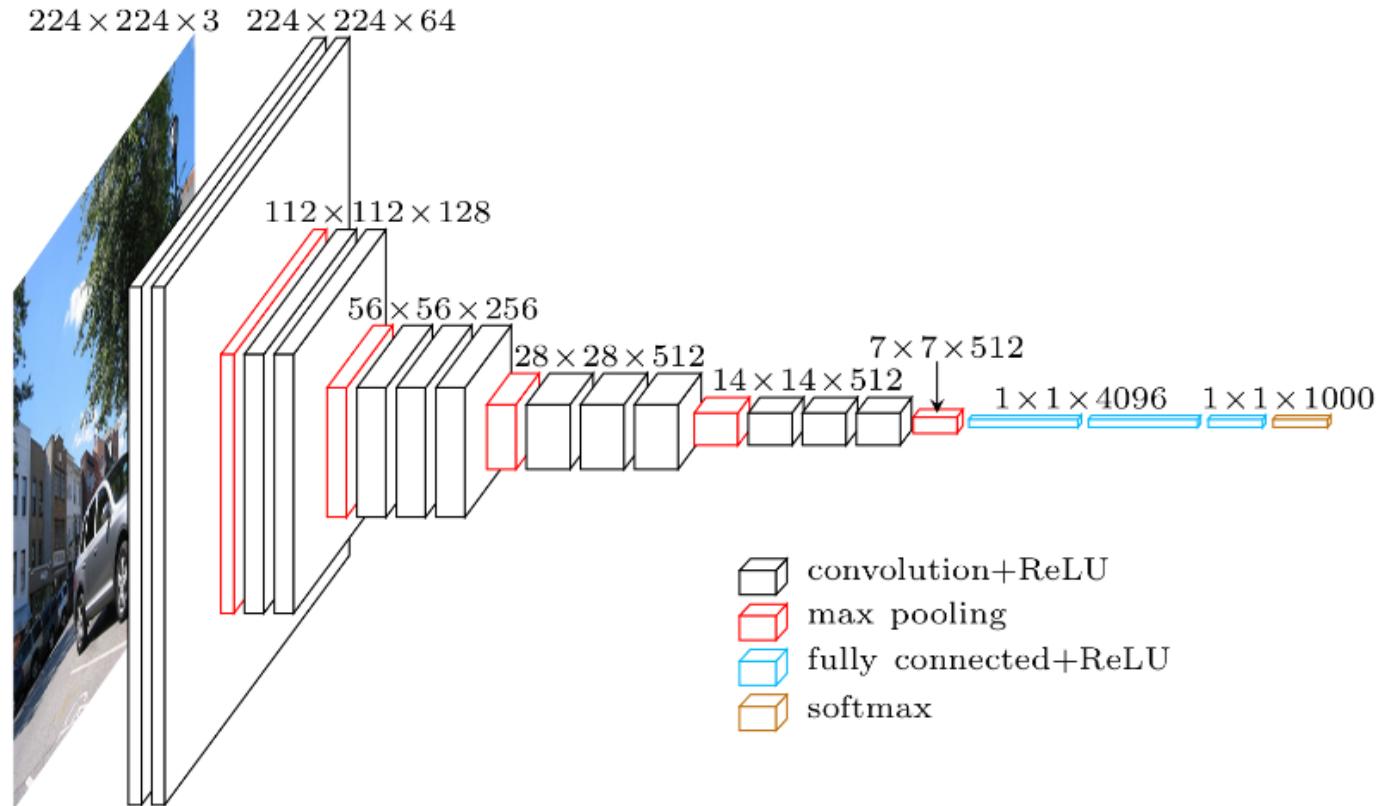
INPUT: [227x227x3]  
CONV1: [55x55x96] 96 11x11 filters at stride 4, pad 0  
MAX POOL1: [27x27x96] 3x3 filters at stride 2  
CONV2: [27x27x256] 256 5x5 filters at stride 1, pad 2  
MAX POOL2: [13x13x256] 3x3 filters at stride 2  
CONV3: [13x13x384] 384 3x3 filters at stride 1, pad 1  
CONV4: [13x13x384] 384 3x3 filters at stride 1, pad 1  
CONV5: [13x13x256] 256 3x3 filters at stride 1, pad 1  
MAX POOL3: [6x6x256] 3x3 filters at stride 2  
FC6: [4096] 4096 neurons  
FC7: [4096] 4096 neurons  
FC8: [1000] 1000 neurons (softmax logits)

# Hierarchical Representation



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

# VGG-16



Simonyan, Karen, and Zisserman. "Very deep convolutional networks for large-scale image recognition." (2014)

# VGG in Keras

```
model.add(Convolution2D(64, 3, 3, activation='relu',input_shape=(3,224,224)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1000, activation='softmax'))
```

# Memory and Parameters

	Activation maps	Parameters	
INPUT:	[224x224x3] = 150K	0	
CONV3-64:	[224x224x64] = 3.2M	(3x3x3)x64 = 1,728	
CONV3-64:	[224x224x64] = 3.2M	(3x3x64)x64 = 36,864	
POOL2:	[112x112x64] = 800K	0	
CONV3-128:	[112x112x128] = 1.6M	(3x3x64)x128 = 73,728	
CONV3-128:	[112x112x128] = 1.6M	(3x3x128)x128 = 147,456	
POOL2:	[56x56x128] = 400K	0	
CONV3-256:	[56x56x256] = 800K	(3x3x128)x256 = 294,912	
CONV3-256:	[56x56x256] = 800K	(3x3x256)x256 = 589,824	
CONV3-256:	[56x56x256] = 800K	(3x3x256)x256 = 589,824	
POOL2:	[28x28x256] = 200K	0	
CONV3-512:	[28x28x512] = 400K	(3x3x256)x512 = 1,179,648	
CONV3-512:	[28x28x512] = 400K	(3x3x512)x512 = 2,359,296	
CONV3-512:	[28x28x512] = 400K	(3x3x512)x512 = 2,359,296	
POOL2:	[14x14x512] = 100K	0	
CONV3-512:	[14x14x512] = 100K	(3x3x512)x512 = 2,359,296	
CONV3-512:	[14x14x512] = 100K	(3x3x512)x512 = 2,359,296	
CONV3-512:	[14x14x512] = 100K	(3x3x512)x512 = 2,359,296	
POOL2:	[7x7x512] = 25K	0	
FC:	[1x1x4096] = 4096	7x7x512x4096 = 102,760,448	
FC:	[1x1x4096] = 4096	4096x4096 = 16,777,216	
FC:	[1x1x1000] = 1000	4096x1000 = 4,096,000	
TOTAL activations: 24M x 4 bytes ~ 93MB / image (x2 for backward)			
TOTAL parameters: 138M x 4 bytes ~ 552MB (x2 for plain SGD, x4 for Adam)			

# ResNet

Even deeper models:

34, 50, 101, 152 layers

A block learns the residual w.r.t.  
identity

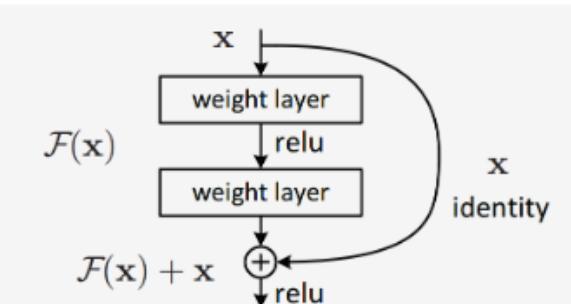
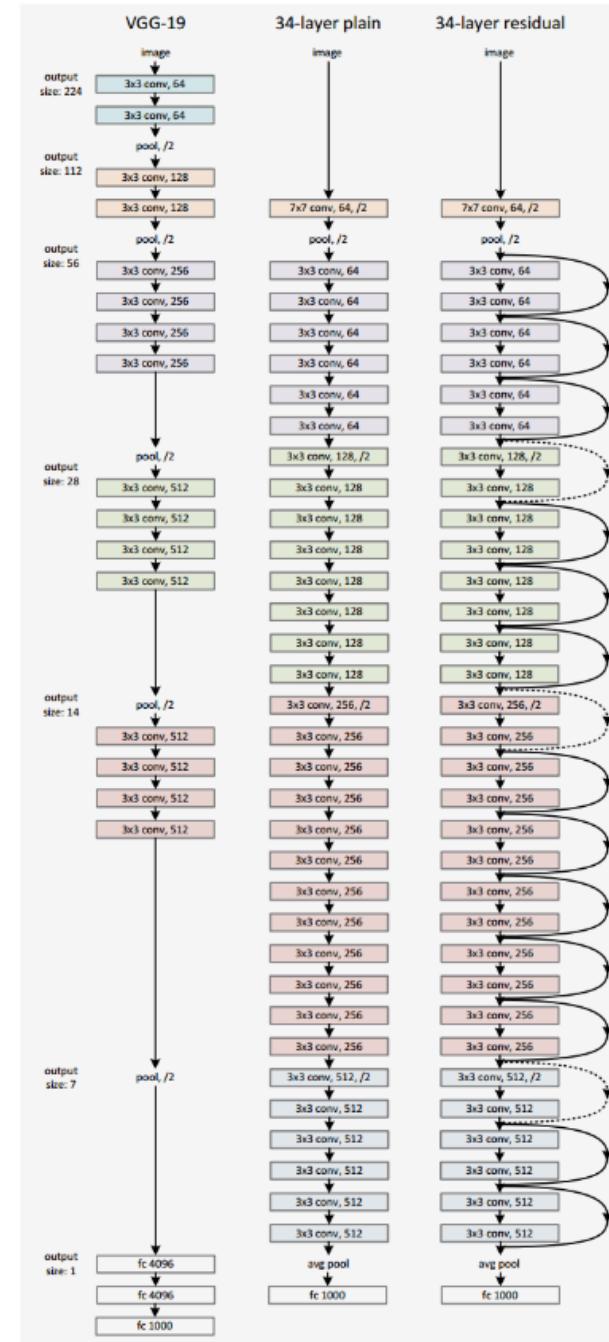


Figure 2. Residual learning: a building block.

- Good optimization properties

He, Kaiming, et al. "Deep residual learning for image recognition." CVPR.  
2016.



# ResNet

ResNet50 Compared to VGG:

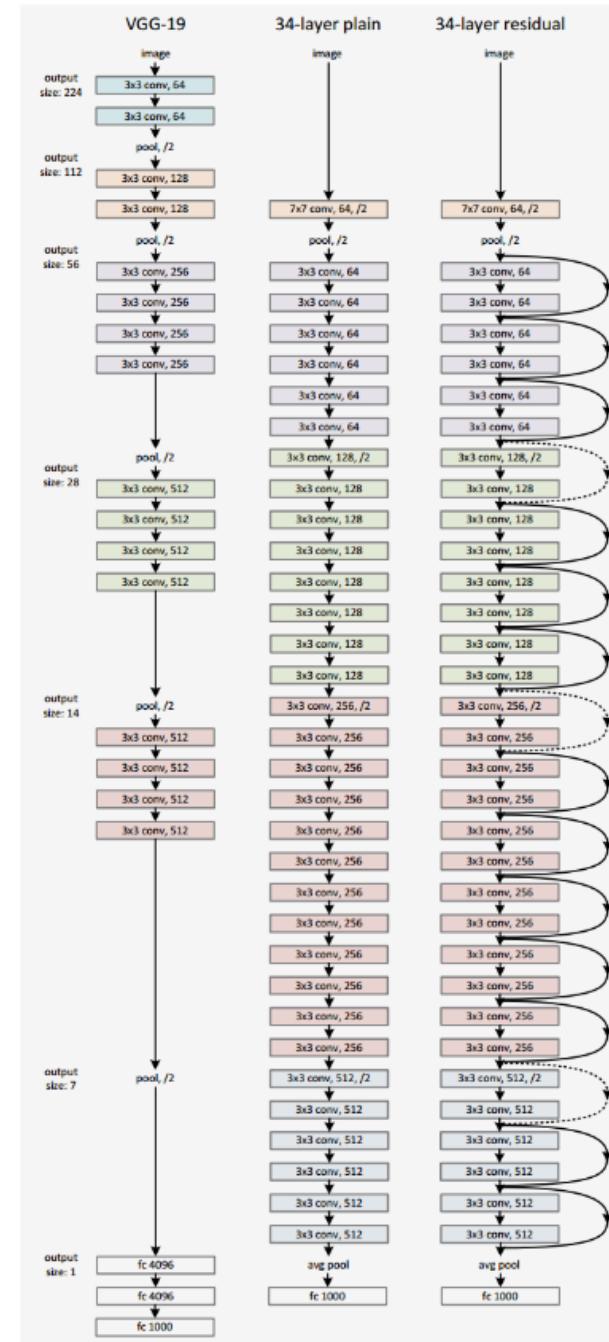
Superior accuracy in all vision tasks  
**5.25% top-5 error vs 7.1%**

Less parameters  
**25M vs 138M**

Computational complexity  
**3.8B Flops vs 15.3B Flops**

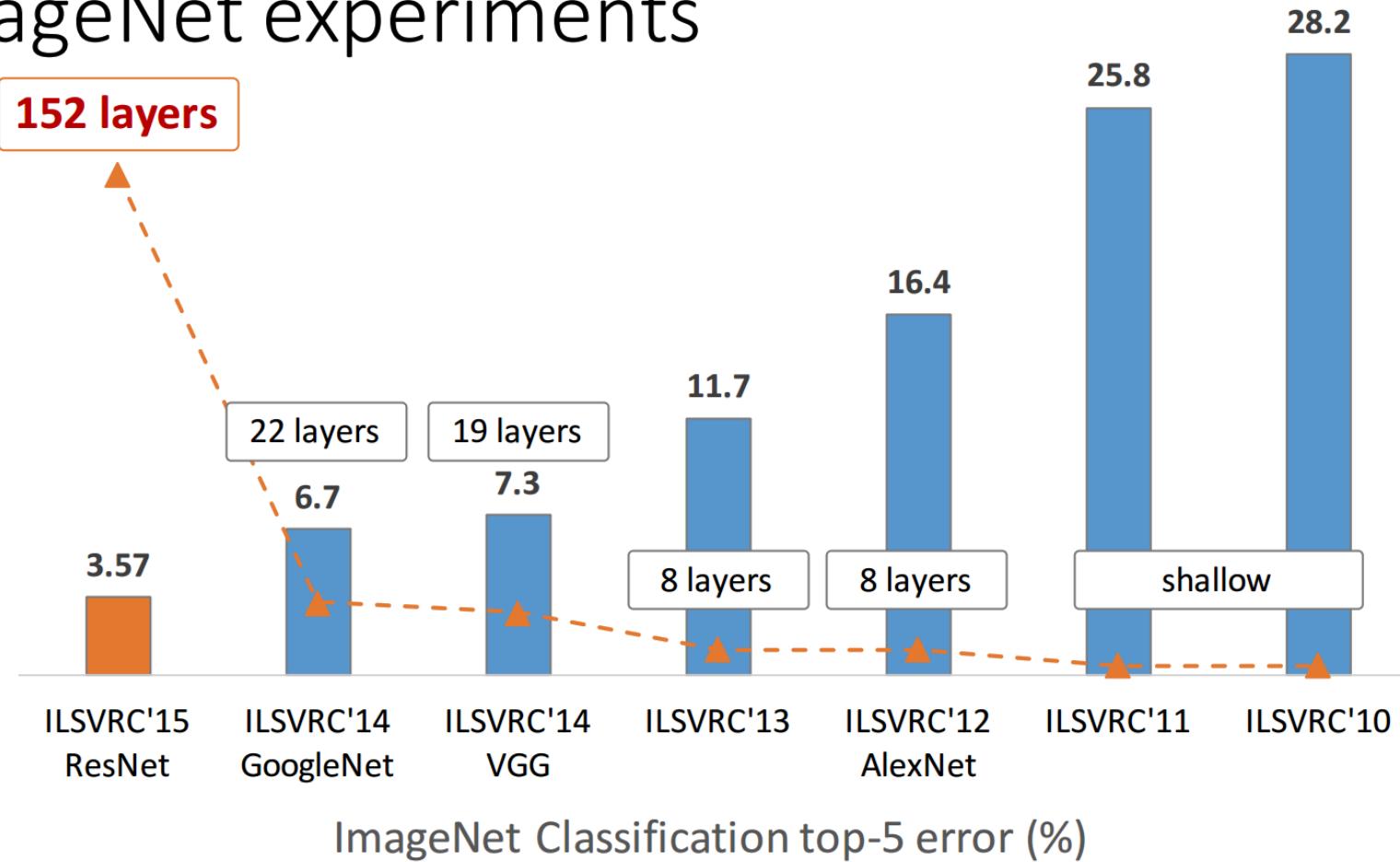
Fully Convolutional until the last layer

He, Kaiming, et al. "Deep residual learning for image recognition." CVPR. 2016.



# Deeper is Better

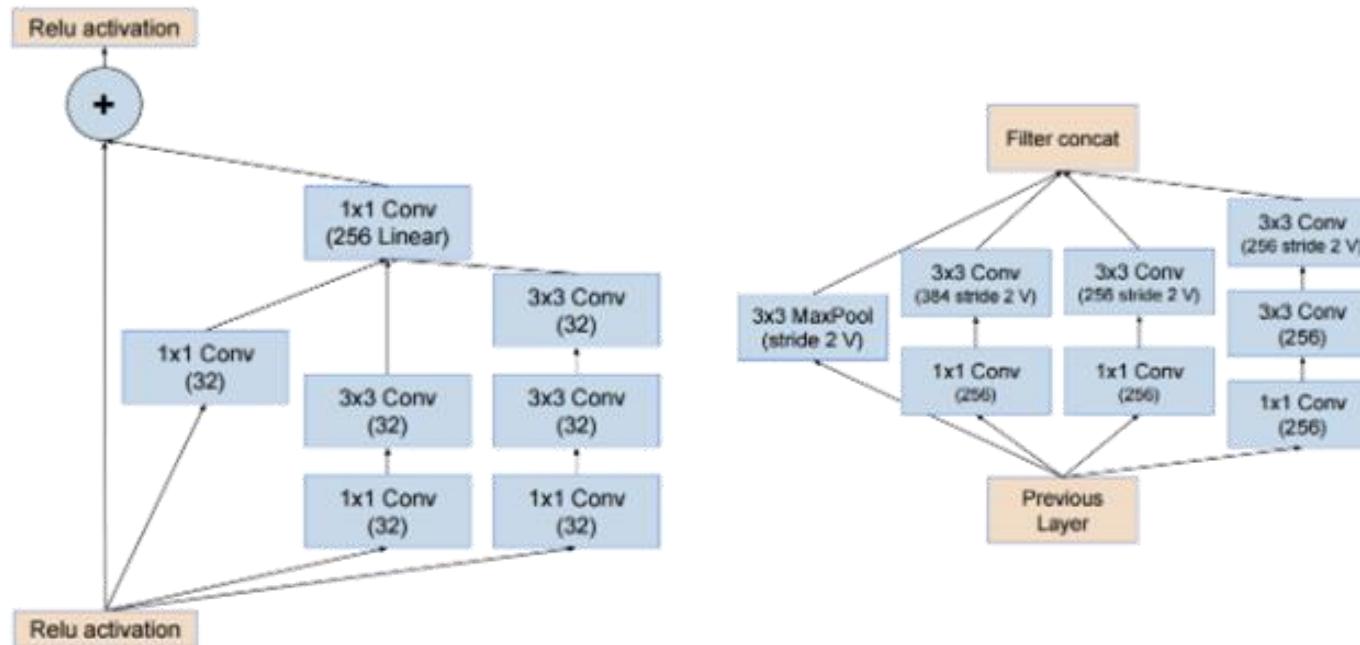
## ImageNet experiments



from Kaiming He slides "Deep residual learning for image recognition." ICML. 2016.

# State of the Art

- Finding right architectures: Active area or research

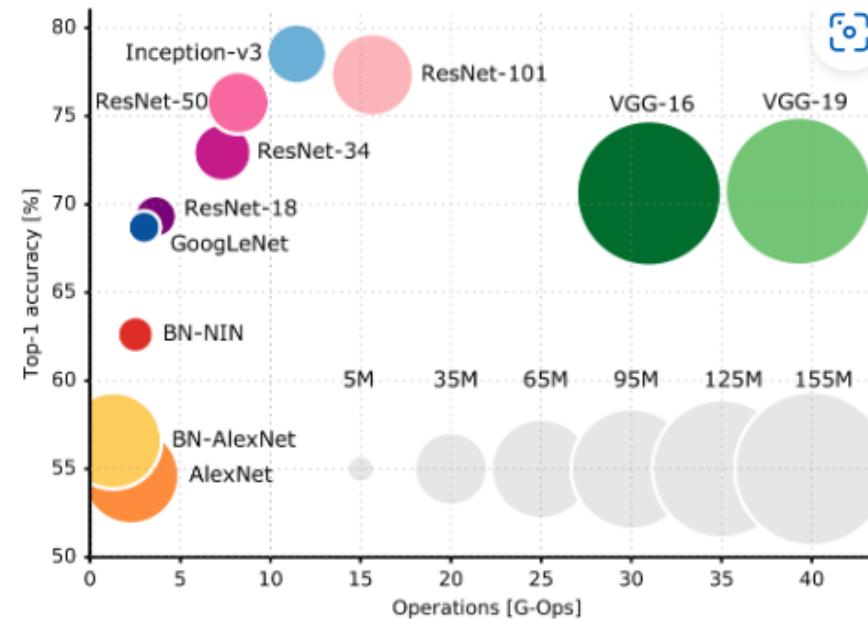
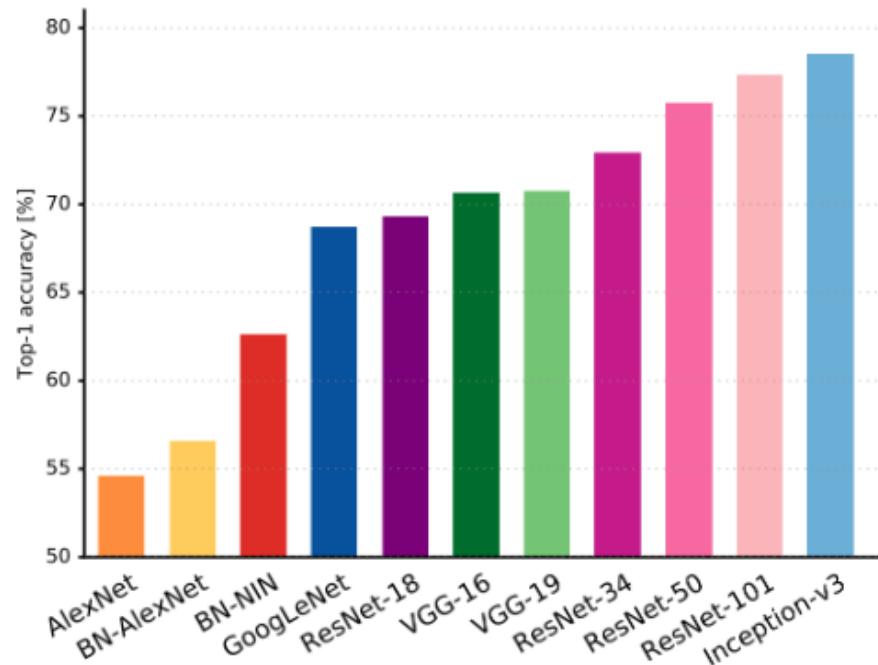


Modular building blocks engineering

see also DenseNets, Wide ResNets, Fractal ResNets, ResNeXts,  
Pyramidal ResNets

# State of the Art

Top 1-accuracy, performance and size on ImageNet



See also: <https://paperswithcode.com/sota/image-classification-on-imagenet>

Canziani, Paszke, and Culurciello. "An Analysis of Deep Neural Network Models for Practical Applications." (May 2016).

# State of the Art...

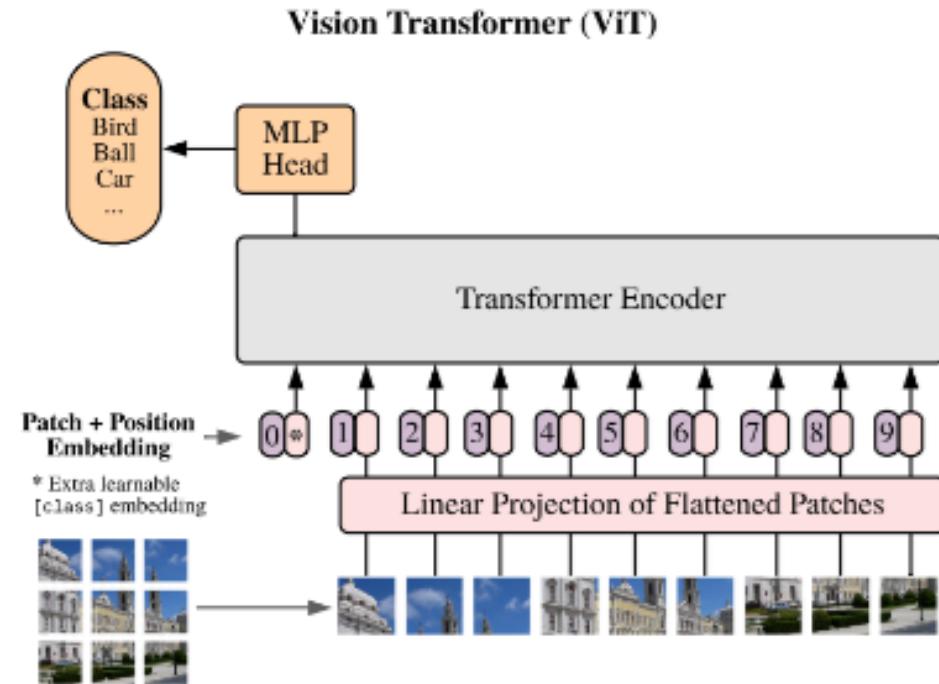
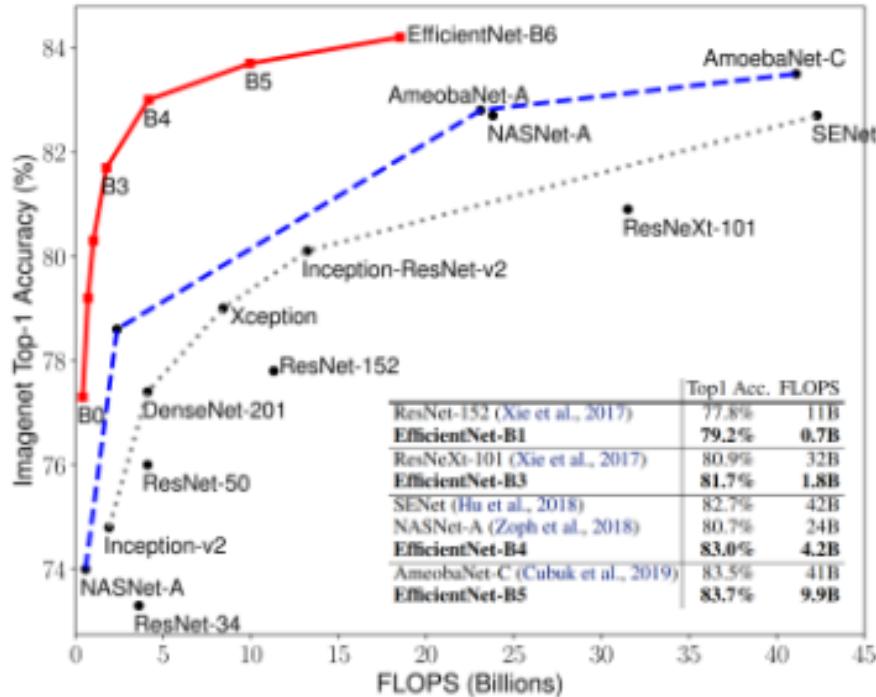


Figure 5. FLOPS vs. ImageNet Accuracy – Similar to Figure 1 except it compares FLOPS rather than model size.

- Mingxing Tan, Quoc V. Le, [EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks](#), ICML 2019.
- Irwan Bello, [LambdaNetworks: Modeling long-range Interactions without Attention](#), ICLR 2021
- Dosovitskiy A. et al, [An Image is worth 16X16 Words: Transformers for Image Recognition at Scale](#), ICLR 2021

# State of the Art...

Method	# Params	Extra Data	ImageNet		ImageNet-Real
			Top-1	Top-5	Precision@1
ResNet-50 [24]	26M	—	76.0	93.0	82.94
ResNet-152 [24]	60M	—	77.8	93.8	84.79
DenseNet-264 [28]	34M	—	77.9	93.9	—
Inception-v3 [62]	24M	—	78.8	94.4	83.58
Xception [11]	23M	—	79.0	94.5	—
Inception-v4 [61]	48M	—	80.0	95.0	—
Inception-resnet-v2 [61]	56M	—	80.1	95.1	—
ResNeXt-101 [78]	84M	—	80.9	95.6	85.18
PolyNet [87]	92M	—	81.3	95.8	—
SENet [27]	146M	—	82.7	96.2	—
NASNet-A [90]	89M	—	82.7	96.2	82.56
AmoebaNet-A [52]	87M	—	82.8	96.1	—
PNASNet [39]	86M	—	82.9	96.2	—
AmoebaNet-C + AutoAugment [12]	155M	—	83.5	96.5	—
GPipe [29]	557M	—	84.3	97.0	—
EfficientNet-B7 [63]	66M	—	85.0	97.2	—
EfficientNet-B7 + FixRes [70]	66M	—	85.3	97.4	—
EfficientNet-L2 [63]	480M	—	85.5	97.5	—
ResNet-50 Billion-scale SSL [79]	26M	3.5B labeled Instagram	81.2	96.0	—
ResNeXt-101 Billion-scale SSL [79]	193M	3.5B labeled Instagram	84.8	—	—
ResNeXt-101 WSL [42]	829M	3.5B labeled Instagram	85.4	97.6	88.19
FixRes ResNeXt-101 WSL [69]	829M	3.5B labeled Instagram	86.4	98.0	89.73
Big Transfer (BiT-L) [33]	928M	300M labeled JFT	87.5	98.5	90.54
Noisy Student (EfficientNet-L2) [77]	480M	300M unlabeled JFT	88.4	98.7	90.55
Noisy Student + FixRes [70]	480M	300M unlabeled JFT	88.5	98.7	—
Vision Transformer (ViT-H) [14]	632M	300M labeled JFT	88.55	—	90.72
EfficientNet-L2-NoisyStudent + SAM [16]	480M	300M unlabeled JFT	88.6	98.6	—
Meta Pseudo Labels (EfficientNet-B6-Wide)	390M	300M unlabeled JFT	90.0	98.7	<b>91.12</b>
Meta Pseudo Labels (EfficientNet-L2)	480M	300M unlabeled JFT	<b>90.2</b>	<b>98.8</b>	91.02

Meta Pseudo Labels, Hieu Pham et al. (Jan 2021)

# Pre-trained Models

SI8 – Deep Learning for Computer Vision

**Reading:**

- Ian Goodfellow. *Deep Learning*. MIT Press, 2016.
- François Chollet. *Deep Learning With Python*. 2<sup>nd</sup> Edition, 2017.

# Pre-trained Models

- Training a model on ImageNet from scratch takes days or weeks.
- Many models trained on ImageNet and their weights are publicly available!

## Transfer learning – Feature Extraction

- Use pre-trained weights, remove last layers to compute representations of images
- Train a classification model from these features on a new classification task
- The network is used as a generic feature extractor
- Better than handcrafted feature extraction on natural images

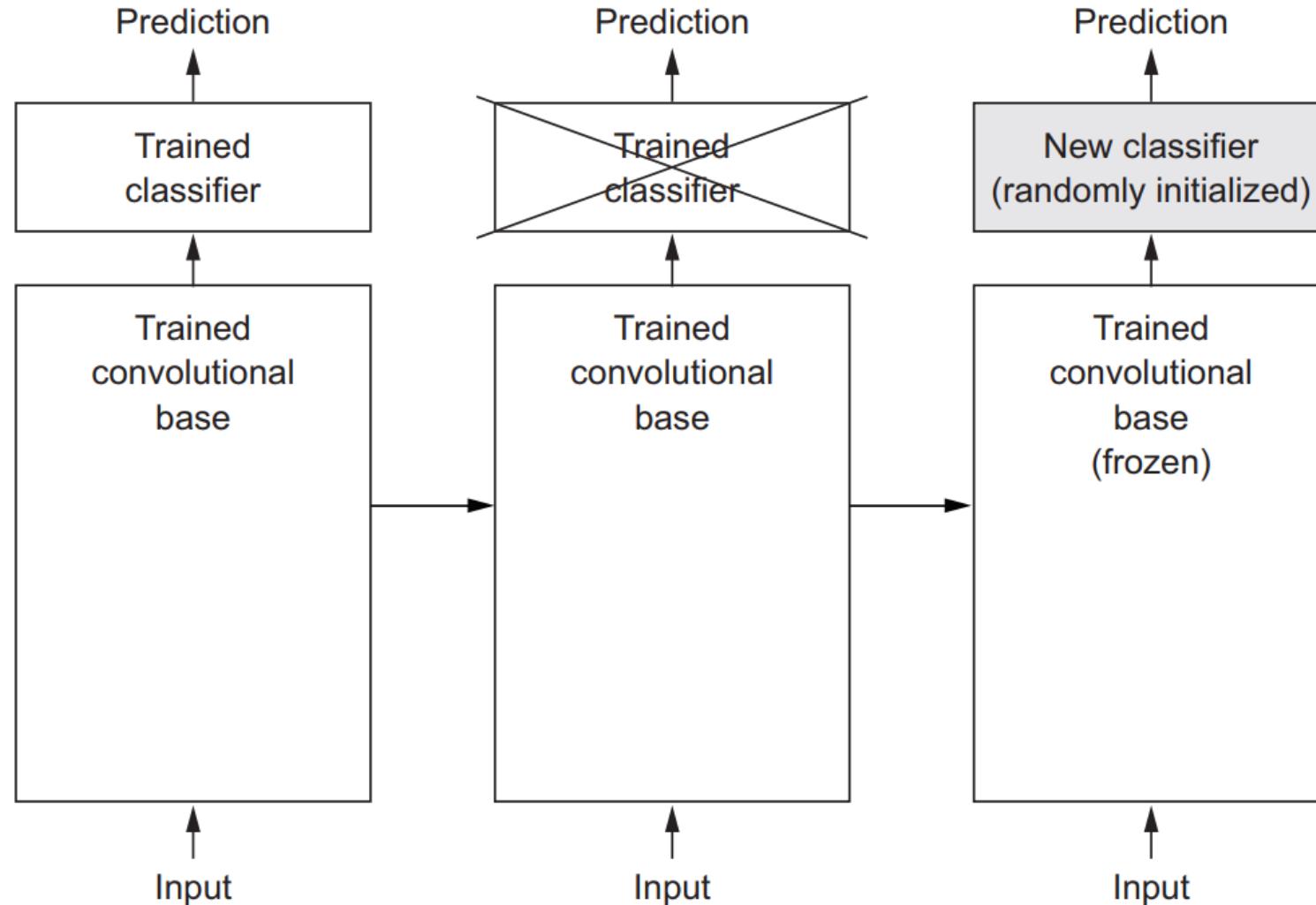
# Pre-trained Models

- Training a model on ImageNet from scratch takes days or weeks.
- Many models trained on ImageNet and their weights are publicly available!

## Fine-tuning

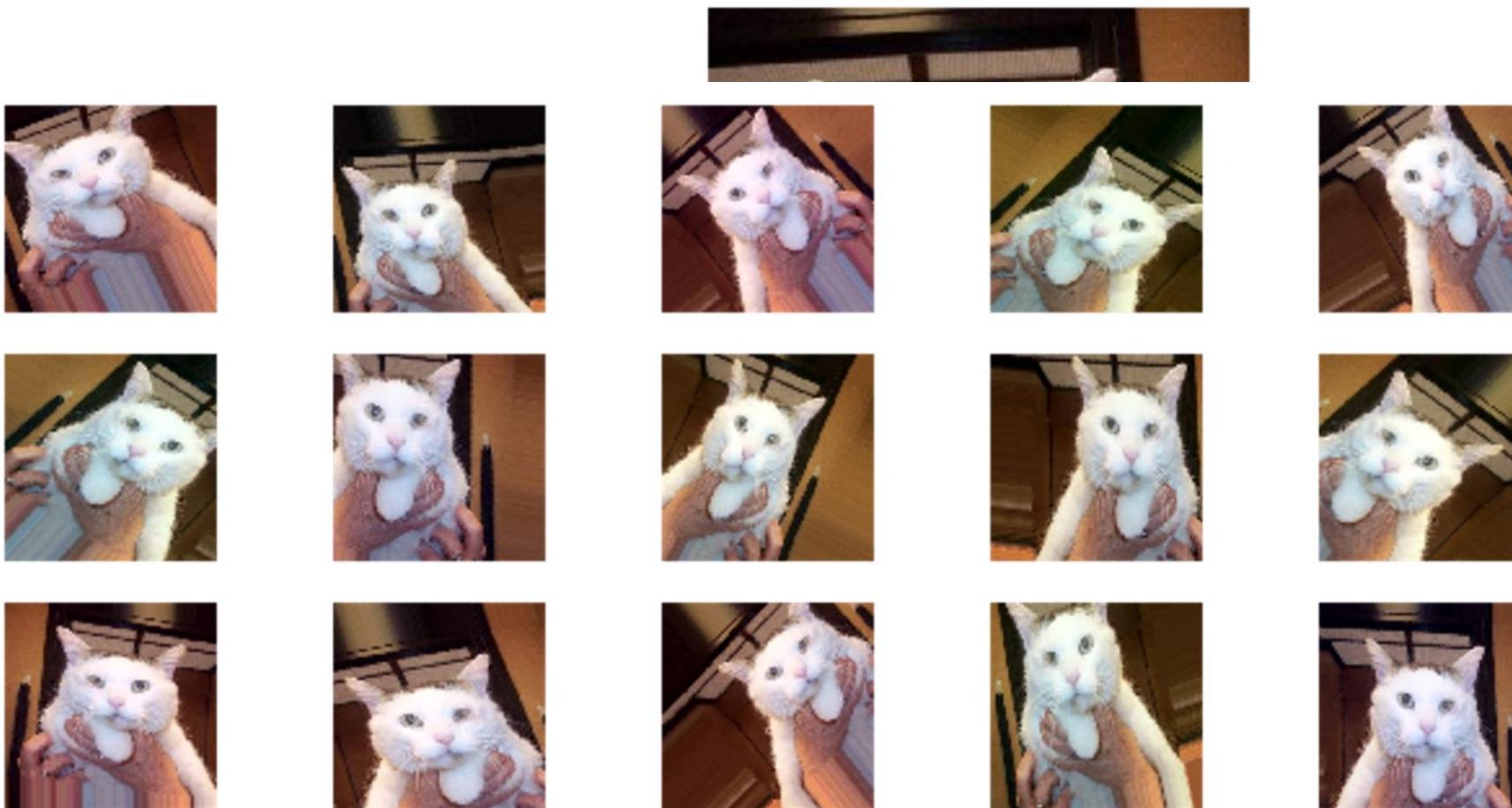
- Retraining the (some) parameters of the network (given enough data)
- Truncate the last layer(s) of the pre-trained network
- Freeze the remaining layers weights
- Add a (linear) classifier on top and train it for a few epochs
- Then fine-tune the whole network or the few deepest layers
- Use a smaller learning rate when fine tuning

# Pre-trained Models





# Data Augmentation



See also: [RandAugment](#) and [Unsupervised Data Augmentation for Consistency Training](#).

# Data Augmentation

- With Keras:

```
from keras.preprocessing.image import ImageDataGenerator

image_gen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    channel_shift_range=9,
    fill_mode='nearest'
)

train_flow = image_gen.flow_from_directory(train_folder)
model.fit_generator(train_flow, train_flow.n)
```

# References

- Ian Goodfellow and Yoshua Bengio and Aaron Courville,  
*Deep Learning*, 2016, MIT Press



TÉCNICO LISBOA