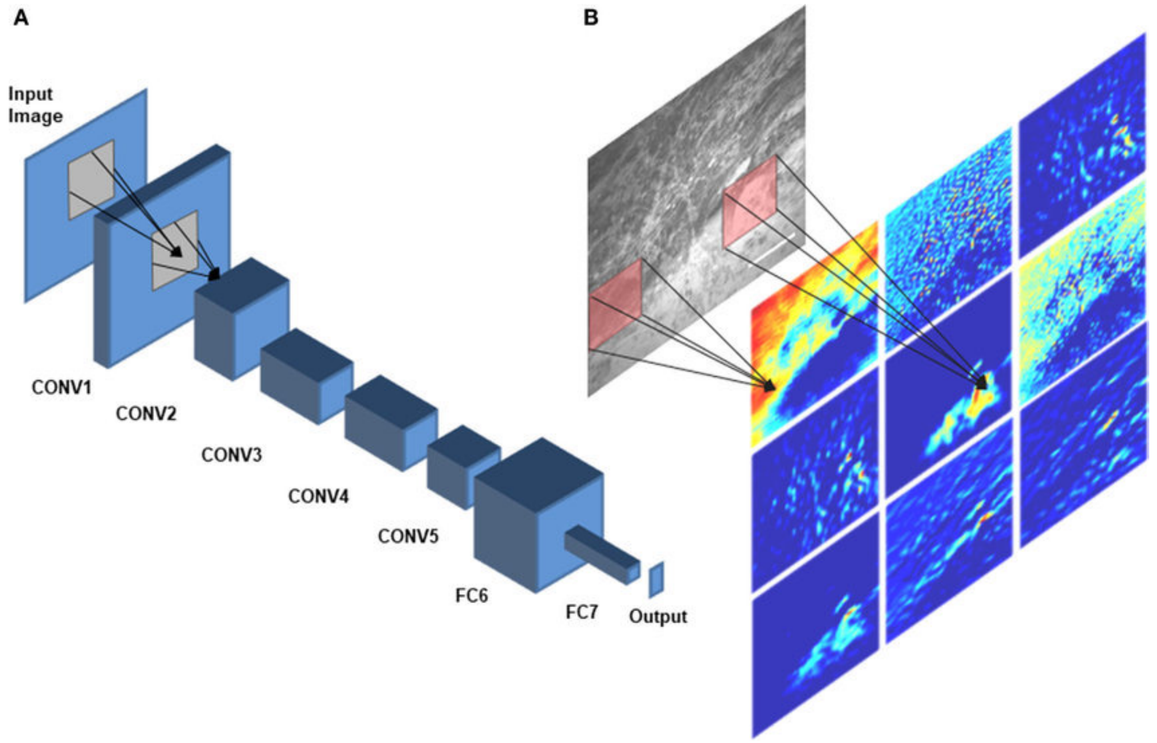
HSLU Lucerne University of Applied Sciences and Arts

# Convolutional Neural Networks for Image Classification




10. November 2023

# Classification

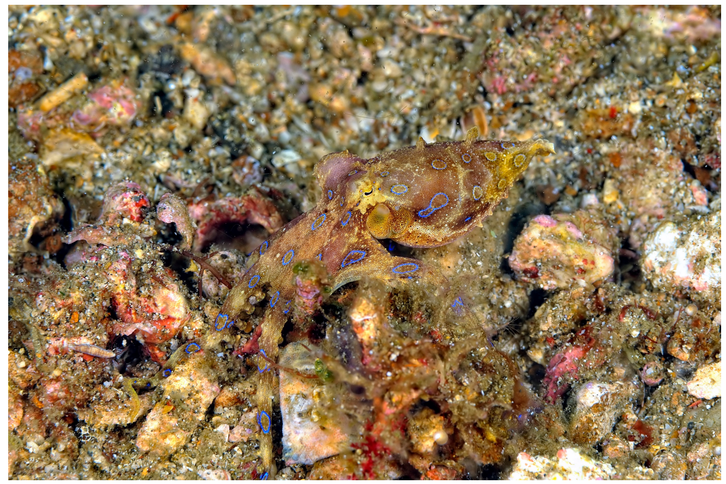
f(  ) = “cat”

We would like to have a function where we can give the image as input and it will return the class of the image.

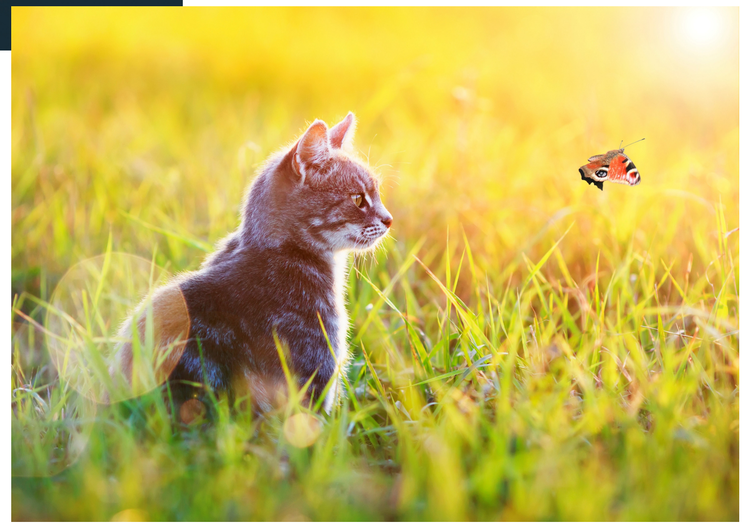
```
def classify(image) -> int:
    # what to do here

    return class_label
```

It is not obvious how to program an algorithm for recognizing different object classes!

# Challenges: Background and Clutter



# Challenges: Illumination

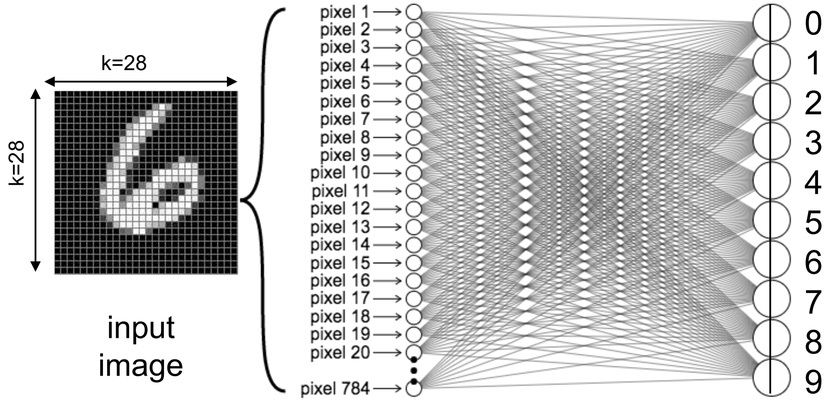


# Challenges: Occlusion

# Challenges: Deformation

# Challenges: Interclass variation

# Image Classification by Multinomial Logistic Regression




$$z_j = \sum_i \left( x_i \cdot w_{ij} \right)$$

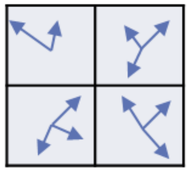
$$\mathrm{p}_j = \frac{e^{z_j}}{\sum_{k=0}^{9} e^{z_k}}$$

Problems:

- Large number of parameters (linear in pixels and classes) even in this simple example
- Can be sensitive to small changes in pixel values

# Hand-designed Features

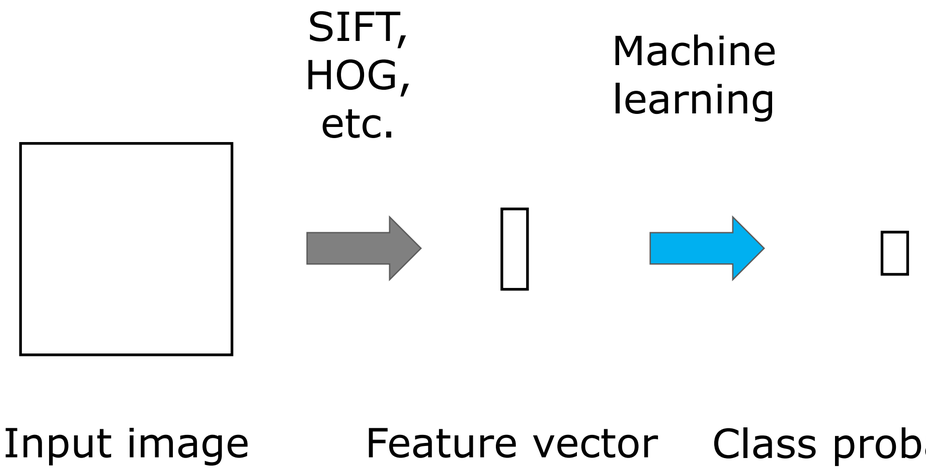
- Features calculate properties of an image or an image region
- Examples
  - Hand-designed filters (difference of Gaussians)
  - Histograms of pixel values
  - SIFT or HOG
- Goal:
  - Feature vector as a robust (e.g., to deformations or illumination) low-dimensional description of image



(0.2, 0.7, 0.3, ...)

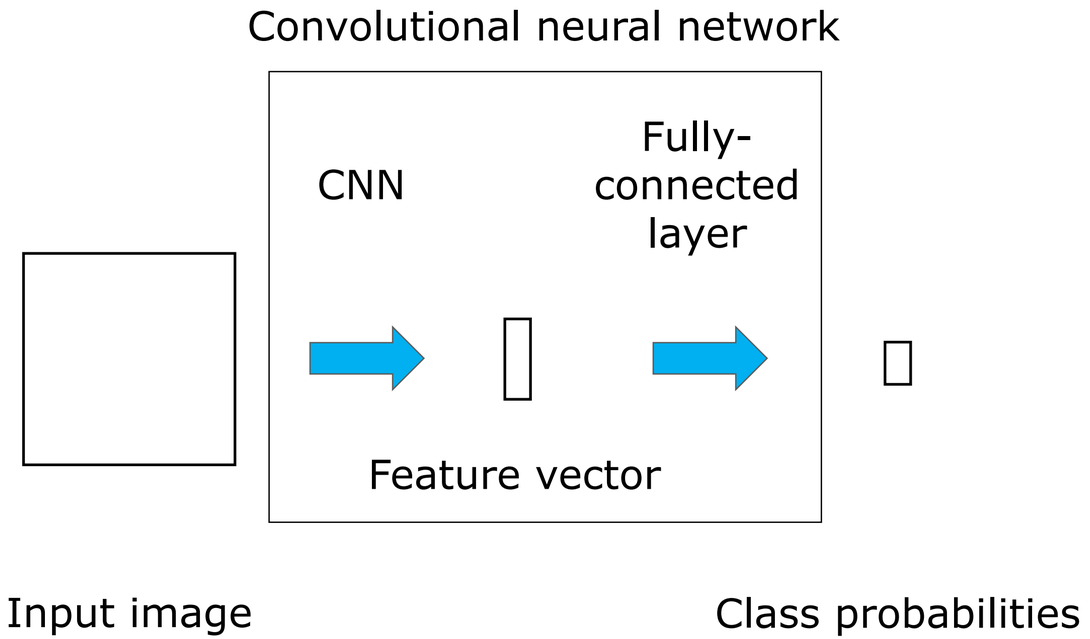
Feature Vector

# Image Classification with Hand-designed Features



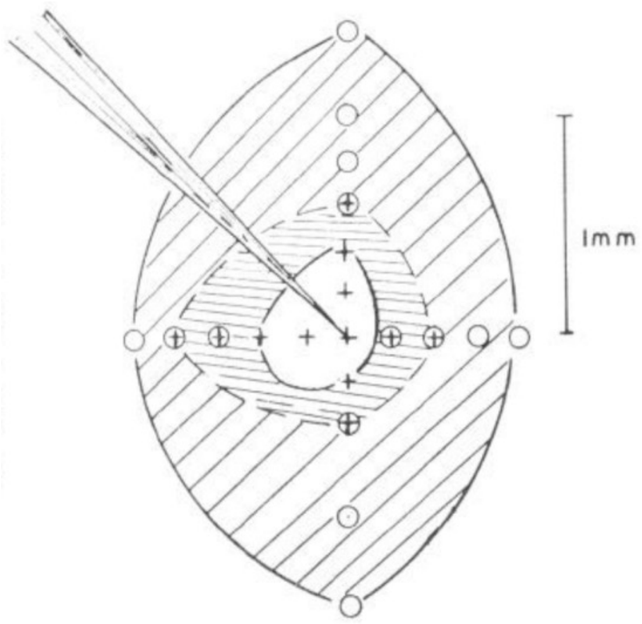

- Pro: More robust than pixel-based image classification
- Pro: Machine learning model has fewer parameters and requires less training data
- Con: Poor features result in poor image classification
- Con: Features must be designed / selected specifically for each classification problem. This requires domain knowledge.

# Convolutional Neural Networks (CNN) for Image Classification



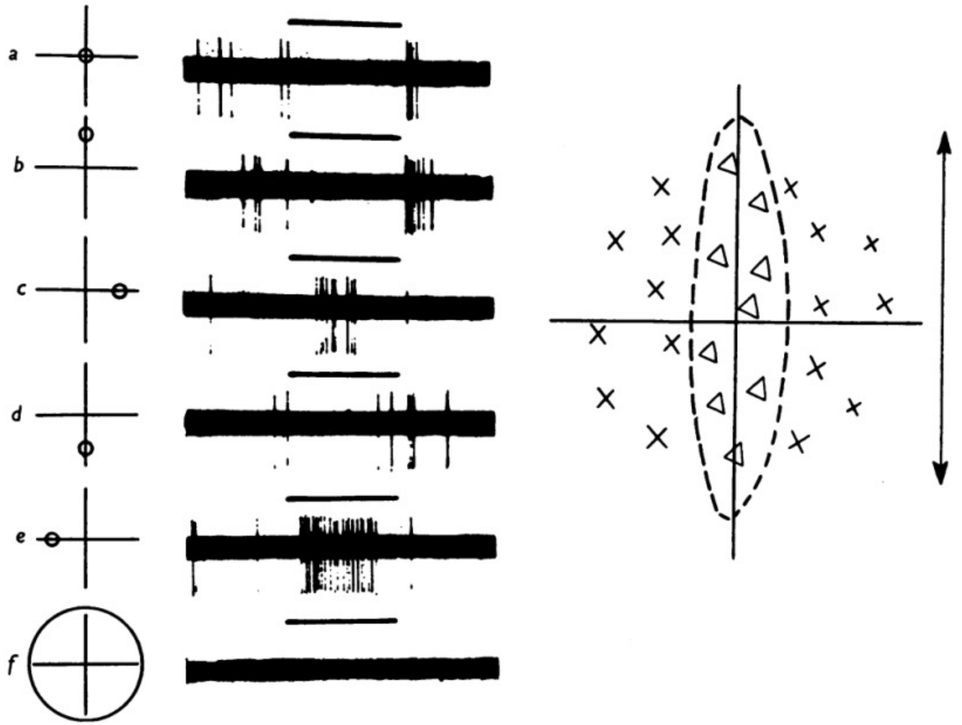
# Hierarchical organization of visual processing in the brain

**Retina**



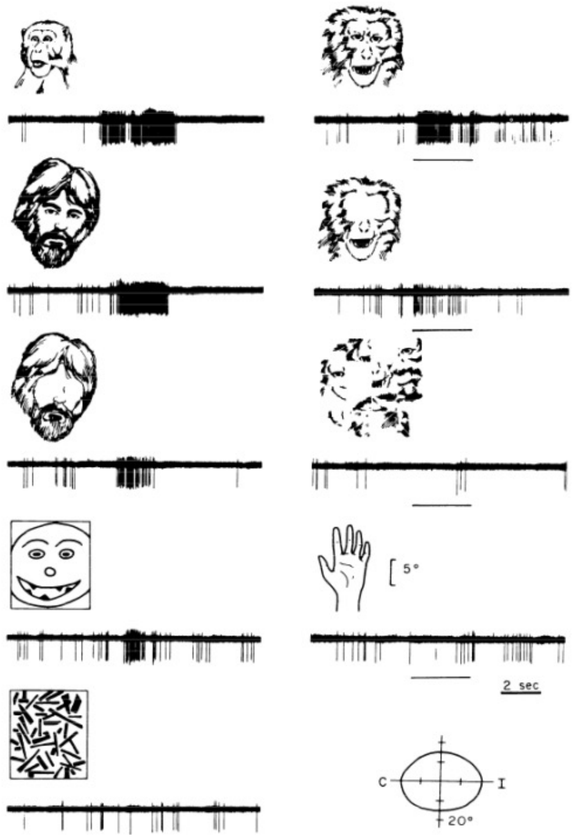

Kuffler 1953
J Neurophysiol

**Visual cortex**



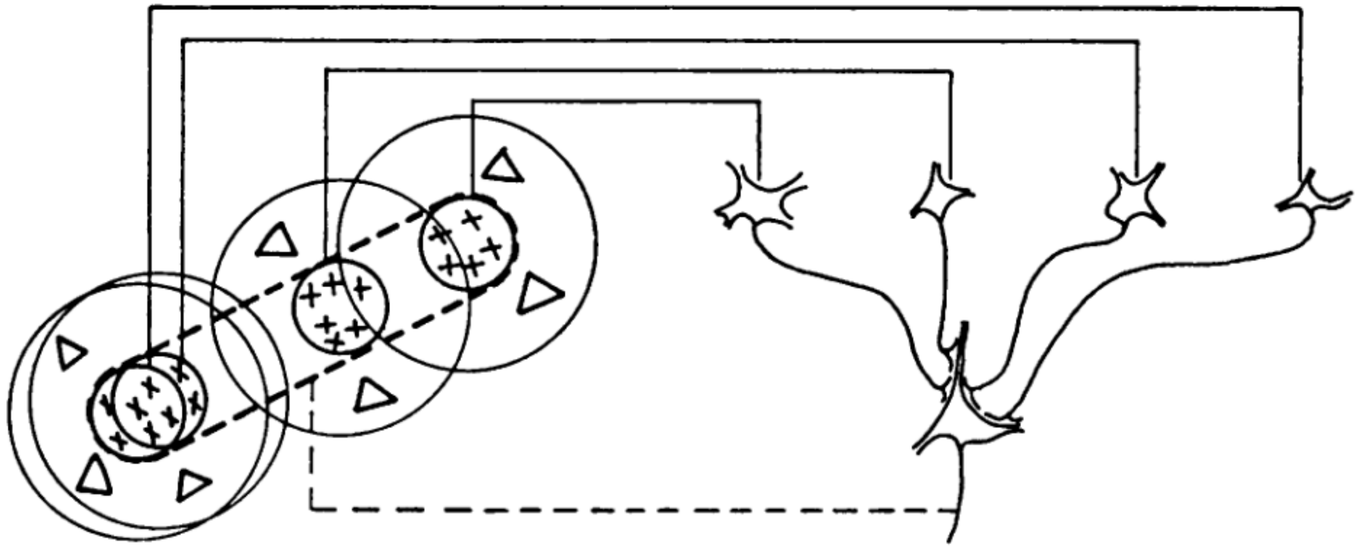

Hubel and Wiesel
1959 J Physiol

**"Higher-order" cortex**




Bruce, Desimone, Gross
1981 J Neurophysiol

# Hierarchical organization of visual processing in the brain





Hubel and Wiesel
1962 J Physiol

# From the neocognitron to convolutional neural networks (CNNs)

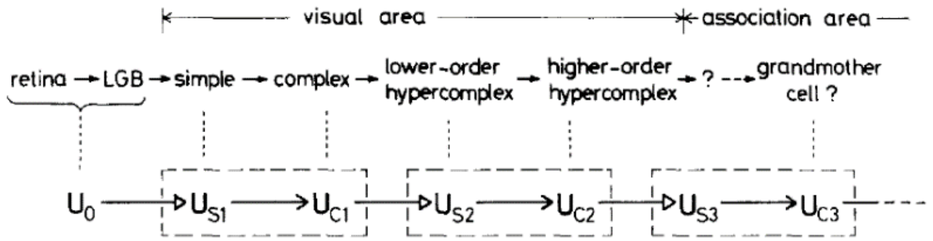



Fig. 1. Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron

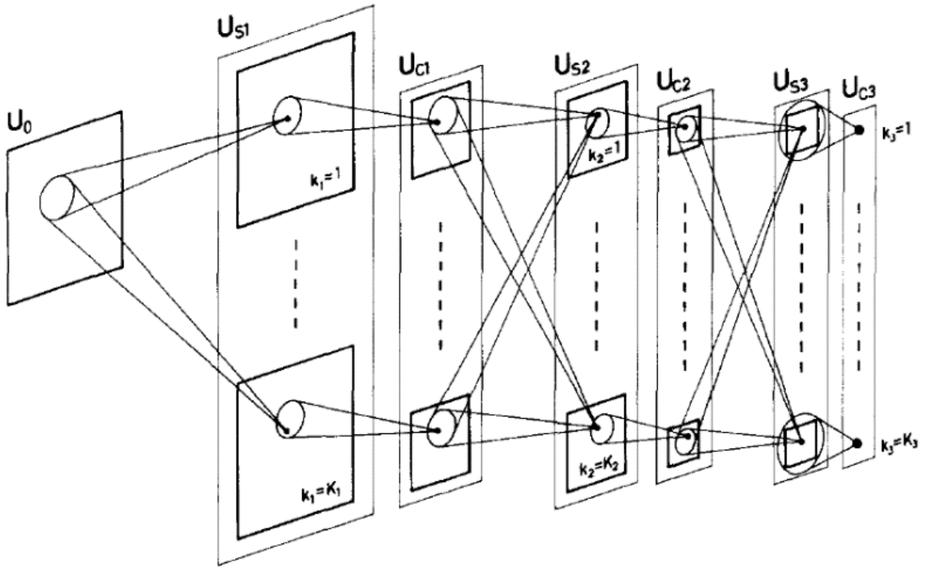



Fig. 2. Schematic diagram illustrating the interconnections between layers in the neocognitron

Error back-propagation
Rumelhart, Hinton, Williams 1986 Nature

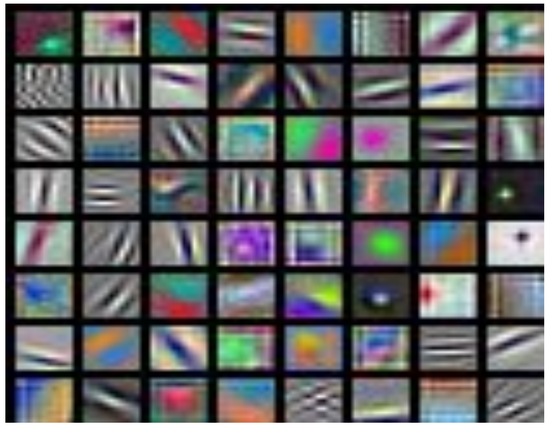
Supervised training of convolutions (weight sharing):
LeCun et al. 1989 Neural Comput

AlexNet (2012), ResNet, .., U-Net, Vision Transformers, ConvNeXt, …



Fukushima 1980 Biol Cybernetics

# Today : Convolutional Neural Networks

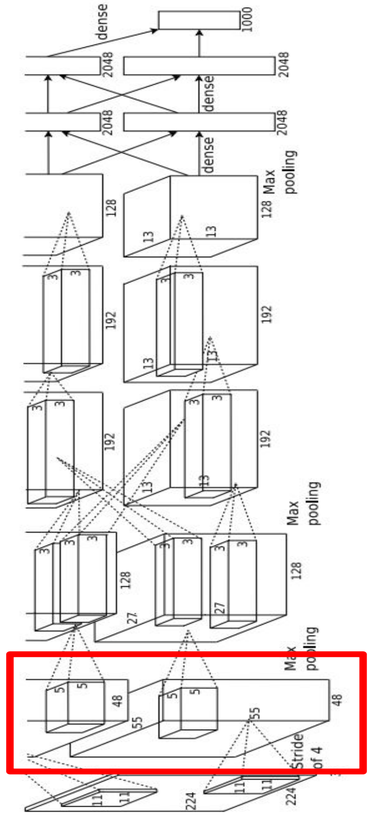


AlexNet:
64 x 3 x 11 x 11

ResNet-18:
64 x 3 x 7 x 7

ResNet-101:
64 x 3 x 7 x 7

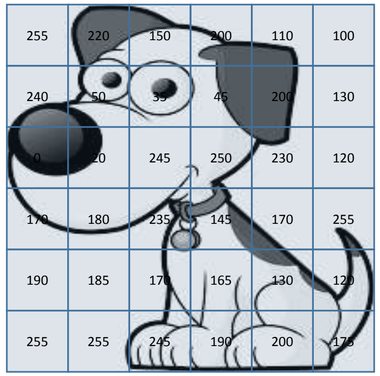
DenseNet-121:
64 x 3 x 7 x 7




Krizhevsky, “One weird trick for parallelizing convolutional neural networks”, arXiv 2014
He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016
Huang et al, “Densely Connected Convolutional Networks”, CVPR 2017

# What are filters doing?

Input image 6x6x1



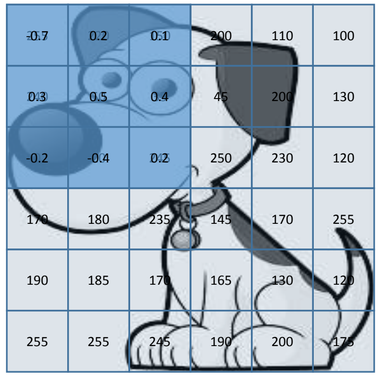

| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

$$z_j = \sum_i \left( x_i \cdot w_{ij} \right)$$

# What are filters doing?

Input image 6x6x1




Feature map
4x4x1

| |
|---|
| 32.5 |

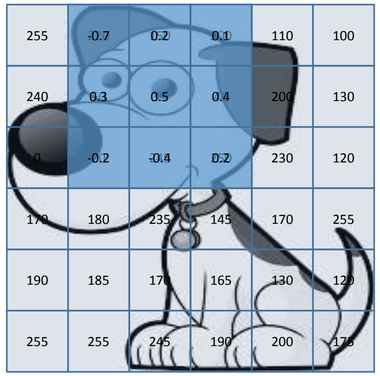
| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

$$z_j = \sum_i \left( x_i \cdot w_{ij} \right)$$

# What are filters doing?

Input image 6x6x1




Feature map
4x4x1

| | |
|---|---|
| 32.5 | -105.5 |

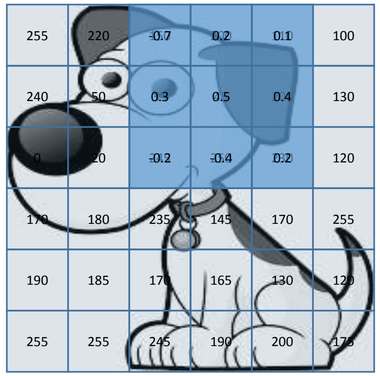
| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

$$z_j = \sum_i \left( x_i \cdot w_{ij} \right)$$

# What are filters doing?

Input image 6x6x1




Feature map
4x4x1

| | | |
|---|---|---|
| 32.5 | -105.5 | 185.5 |

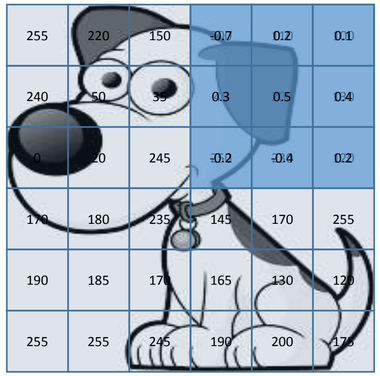
| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

$$z_j = \sum_i \left( x_i \cdot w_{ij} \right)$$

# What are filters doing?

Input image 6x6x1




Feature map
4x4x1

| | | | |
|---|---|---|---|
| 32.5 | -105.5 | 185.5 | 54 |

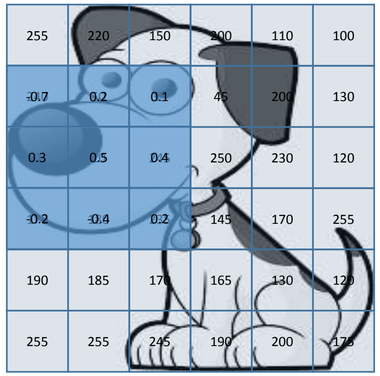
| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

$$z_j = \sum_i \left( x_i \cdot w_{ij} \right)$$

# What are filters doing?

Input image 6x6x1




Feature map
4x4x1

| | | | |
|---|---|---|---|
| 32.5 | -105.5 | 185.5 | 54 |
| -105.5 | | | |

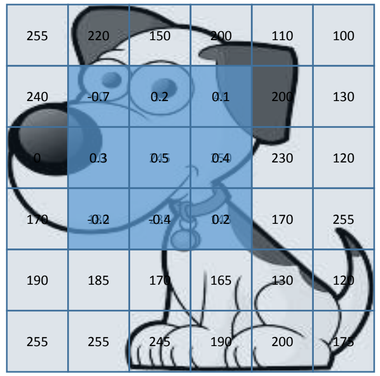
| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

$$z_j = \sum_i \left( x_i \cdot w_{ij} \right)$$

# What are filters doing?

Input image 6x6x1




Feature map
4x4x1

| | | | |
|---|---|---|---|
| 32.5 | -105.5 | 185.5 | 54 |
| -105.5 | 104 | | |

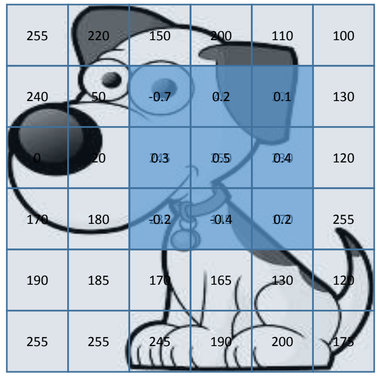
| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

$$z_j = \sum_i \left( x_i \cdot w_{ij} \right)$$

# What are filters doing?

Input image 6x6x1




Feature map
4x4x1

| | | | |
|---|---|---|---|
| 32.5 | -105.5 | 185.5 | 54 |
| -105.5 | 104 | 217.5 | |

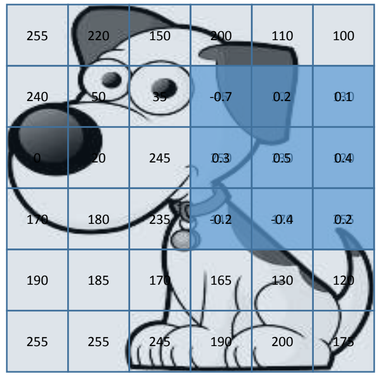
| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

$$z_j = \sum_i \left( x_i \cdot w_{ij} \right)$$

# What are filters doing?

Input image 6x6x1




Feature map
4x4x1

| | | | |
|---|---|---|---|
| 32.5 | -105.5 | 185.5 | 54 |
| -105.5 | 104 | 217.5 | 31 |

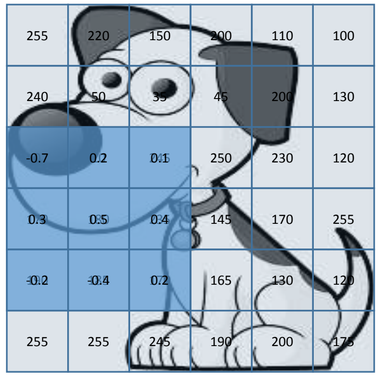
| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

$$z_j = \sum_i \left( x_i \cdot w_{ij} \right)$$

# What are filters doing?

Input image 6x6x1




Feature map
4x4x1

| | | | |
|---|---|---|---|
| 32.5 | -105.5 | 185.5 | 54 |
| -105.5 | 104 | 217.5 | 31 |
| -44 | | | |

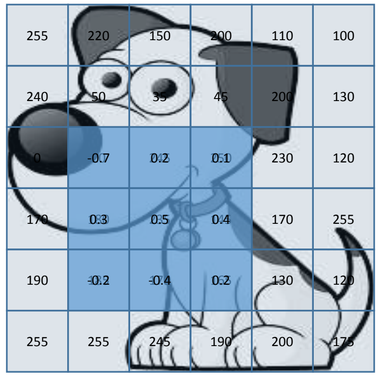
| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

$$z_j = \sum_i \left( x_i \cdot w_{ij} \right)$$

# What are filters doing?

Input image 6x6x1




Feature map
4x4x1

| | | | |
|---|---|---|---|
| 32.5 | -105.5 | 185.5 | 54 |
| -105.5 | 104 | 217.5 | 31 |
| -44 | 224 | | |

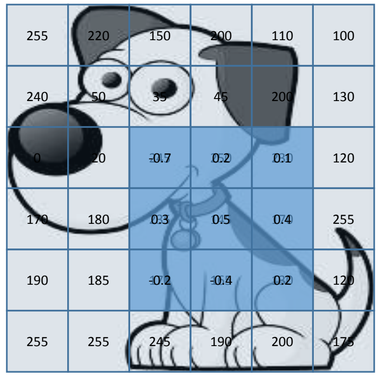
| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

$$z_j = \sum_i \left( x_i \cdot w_{ij} \right)$$

# What are filters doing?

Input image 6x6x1




Feature map
4x4x1

| | | | |
|---|---|---|---|
| 32.5 | -105.5 | 185.5 | 54 |
| -105.5 | 104 | 217.5 | 31 |
| -44 | 224 | 38.5 | |

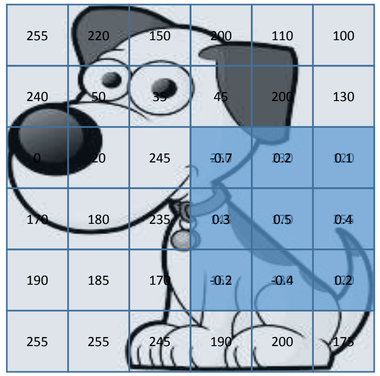
| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

$$z_j = \sum_i \left( x_i \cdot w_{ij} \right)$$

# What are filters doing?

Input image 6x6x1




Feature map
4x4x1

| | | | |
|---|---|---|---|
| 32.5 | -105.5 | 185.5 | 54 |
| -105.5 | 104 | 217.5 | 31 |
| -44 | 224 | 38.5 | -18 |

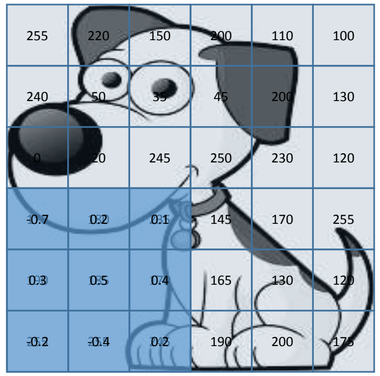
| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

$$z_j = \sum_i \left( x_i \cdot w_{ij} \right)$$

# What are filters doing?

Input image 6x6x1




Feature map
4x4x1

| | | | |
|---|---|---|---|
| 32.5 | -105.5 | 185.5 | 54 |
| -105.5 | 104 | 217.5 | 31 |
| -44 | 224 | 38.5 | -18 |
| -60.5 | | | |

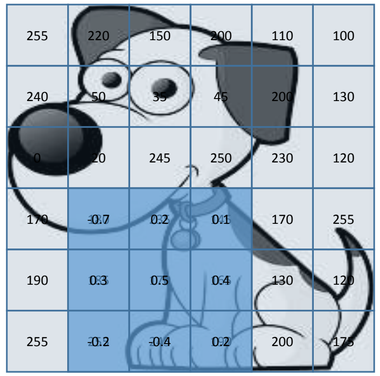
| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

$$z_j = \sum_i \left( x_i \cdot w_{ij} \right)$$

# What are filters doing?

Input image 6x6x1




Feature map
4x4x1

| | | | |
|---|---|---|---|
| 32.5 | -105.5 | 185.5 | 54 |
| -105.5 | 104 | 217.5 | 31 |
| -44 | 224 | 38.5 | -18 |
| -60.5 | 213.5 | | |

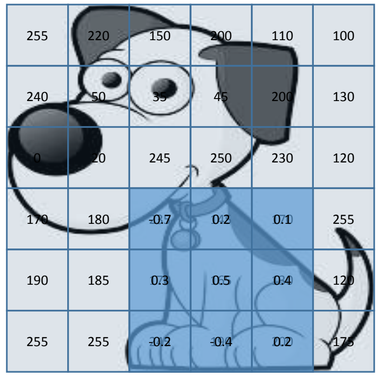
| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

$$z_j = \sum_i \left( x_i \cdot w_{ij} \right)$$

# What are filters doing?

Input image 6x6x1




Feature map
4x4x1

| | | | |
|---|---|---|---|
| 32.5 | -105.5 | 185.5 | 54 |
| -105.5 | 104 | 217.5 | 31 |
| -44 | 224 | 38.5 | -18 |
| -60.5 | 213.5 | 52.5 | |

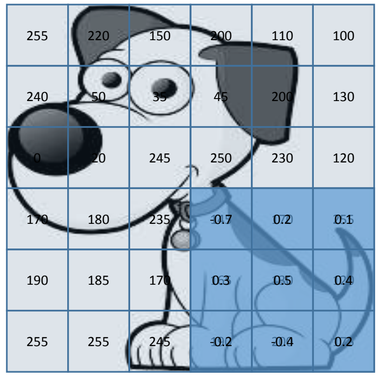
| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

$$z_j = \sum_i \left( x_i \cdot w_{ij} \right)$$

# What are filters doing?

Input image 6x6x1




Feature map
4x4x1

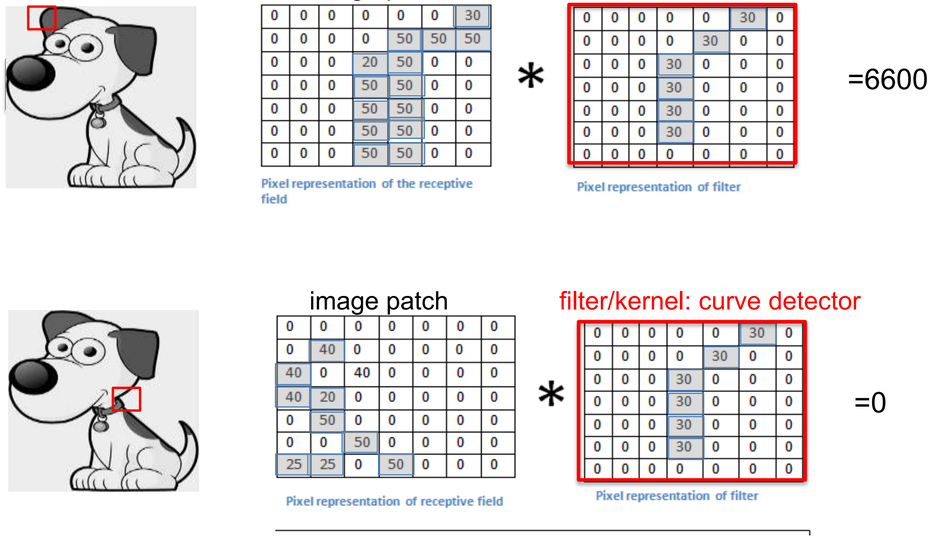
| | | | |
|---|---|---|---|
| 32.5 | -105.5 | 185.5 | 54 |
| -105.5 | 104 | 217.5 | 31 |
| -44 | 224 | 38.5 | -18 |
| -60.5 | 213.5 | 52.5 | 37.5 |

| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

# What are filters doing?

**One kernel or filter searches for specific local feature**

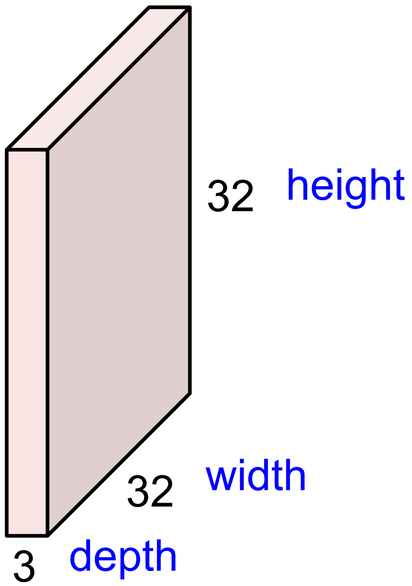



We get a large resulting value if the filter resembles the pattern in the image patch on which the filter was applied.

credits: https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

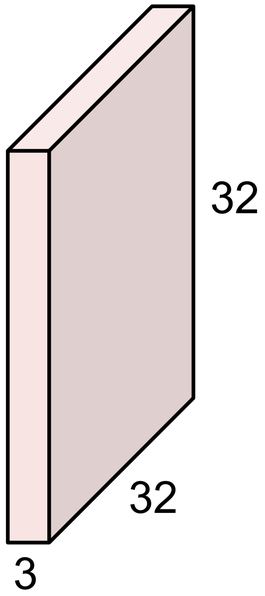
# Convolutional Layer

32x32x3 image -> preserve spatial structure

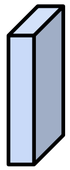


# Convolutional Layer

32x32x3 image

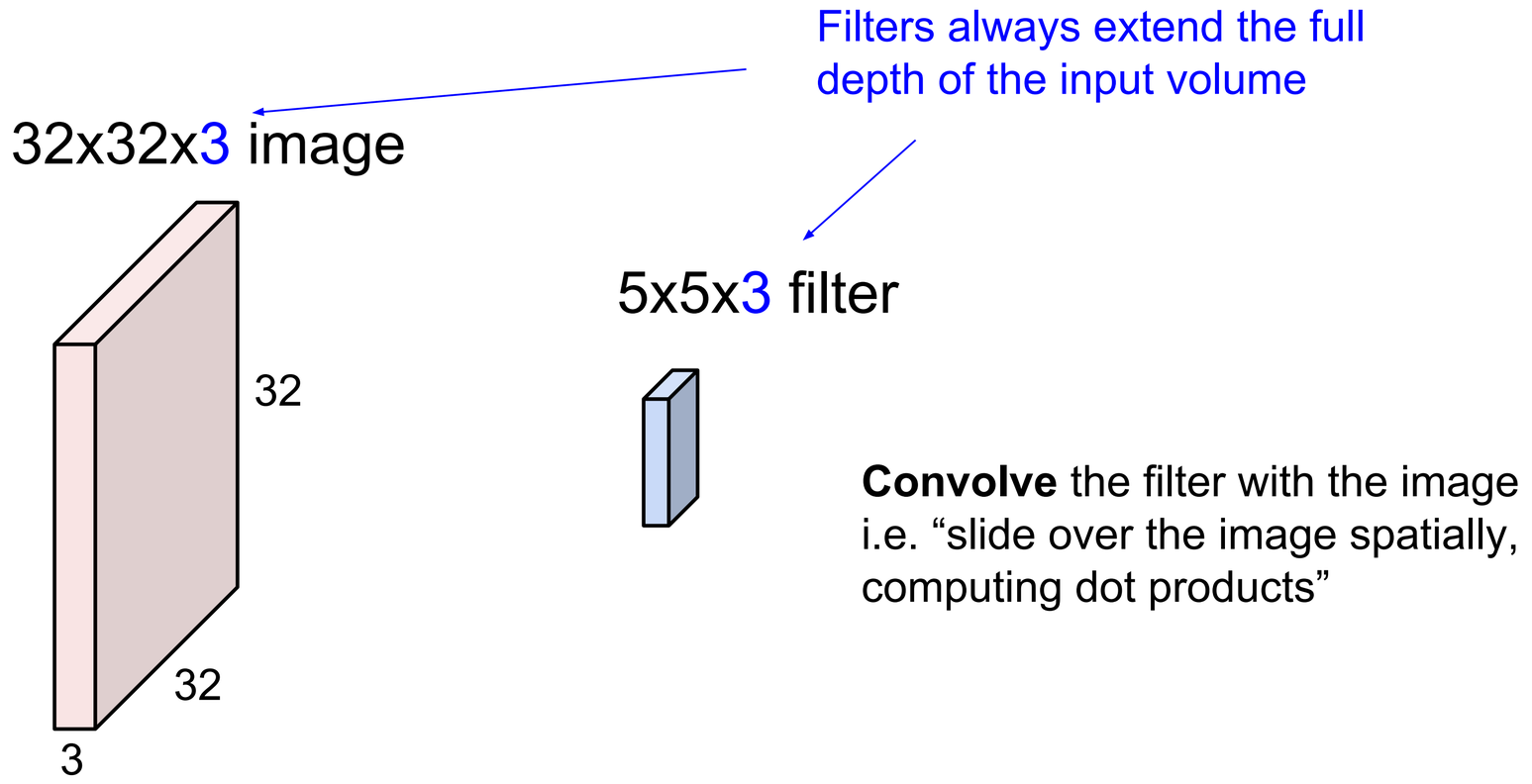



5x5x3 filter

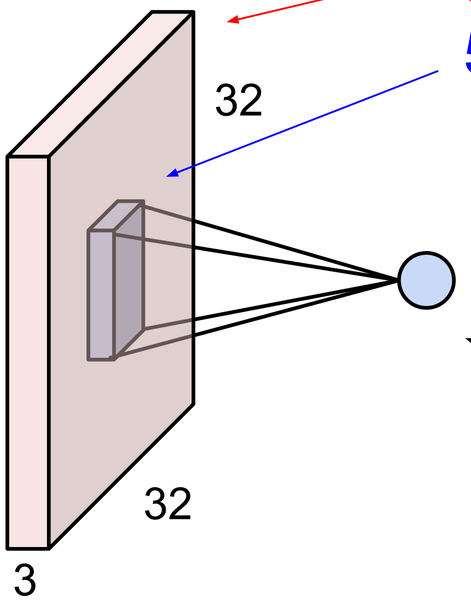


**Convolve** the filter with the image
i.e. “slide over the image spatially,
computing dot products”

# Convolutional Layer



# Convolutional Layer



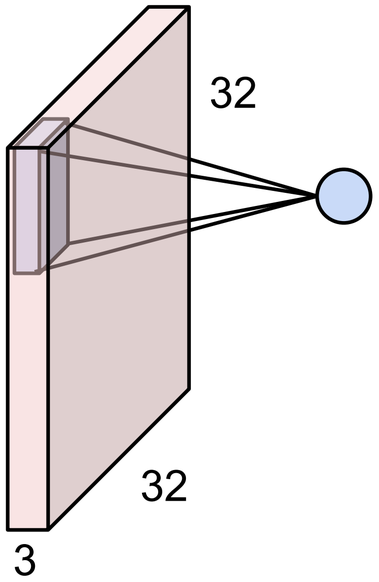

32x32x3 image

5x5x3 filter $w$

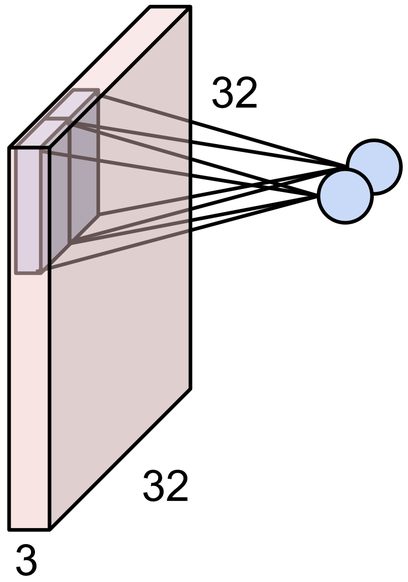
**1 number:**
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

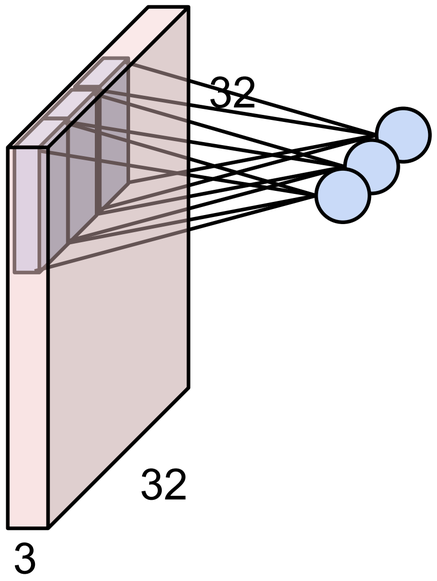
# Convolutional Layer



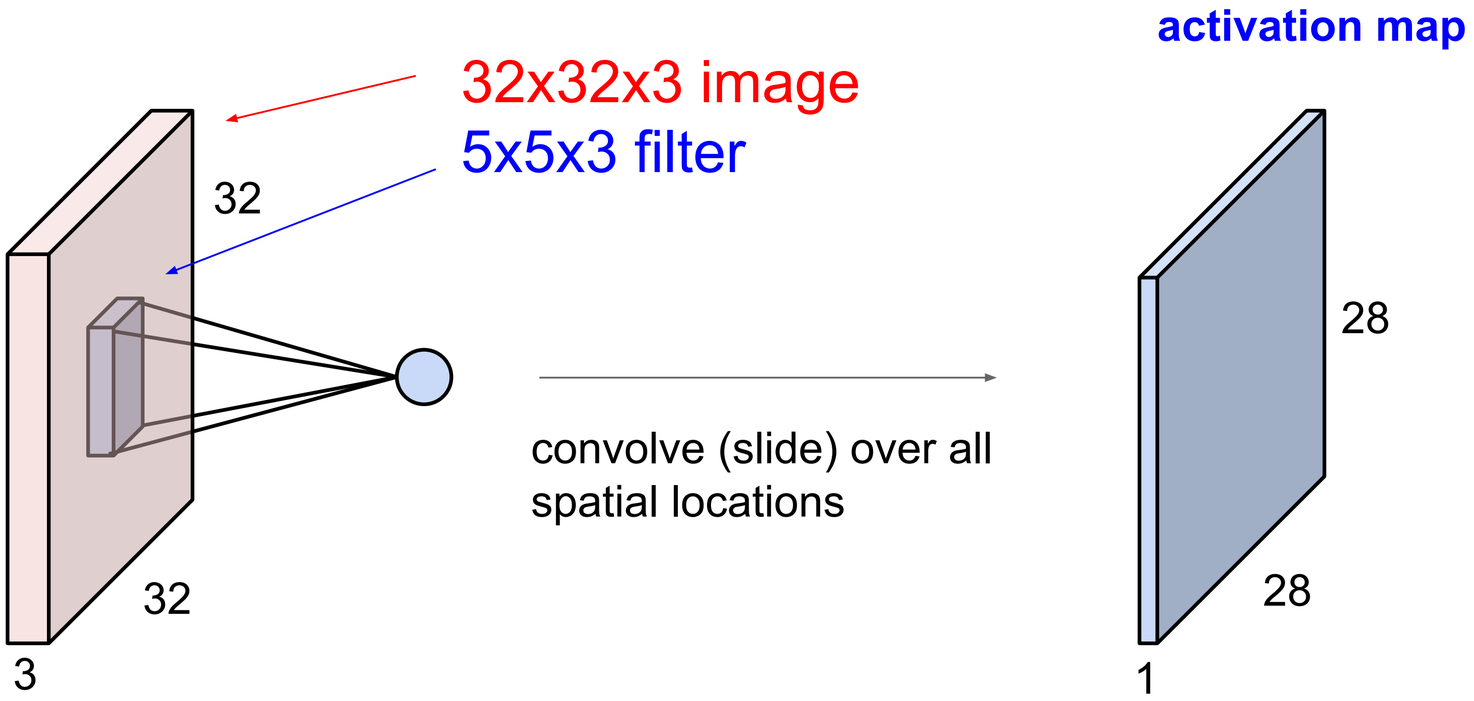
# Convolutional Layer



# Convolutional Layer

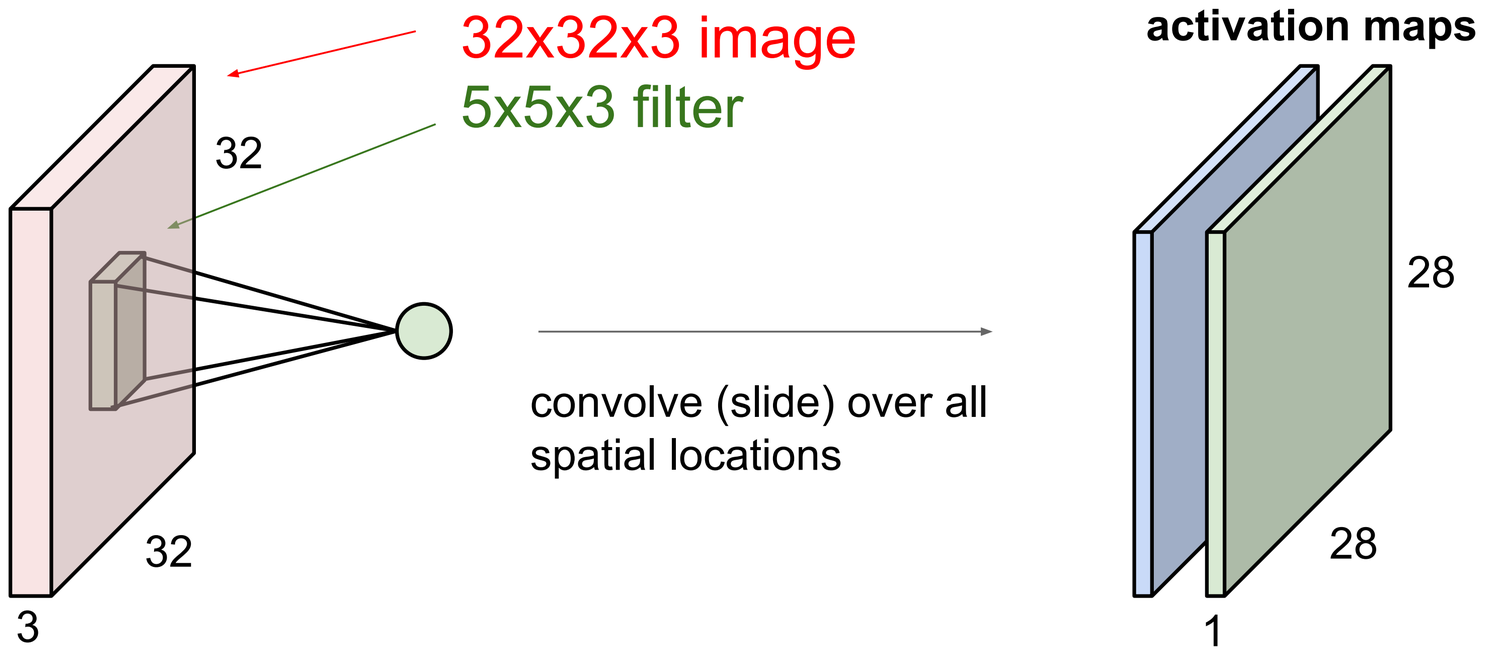


# Convolutional Layer



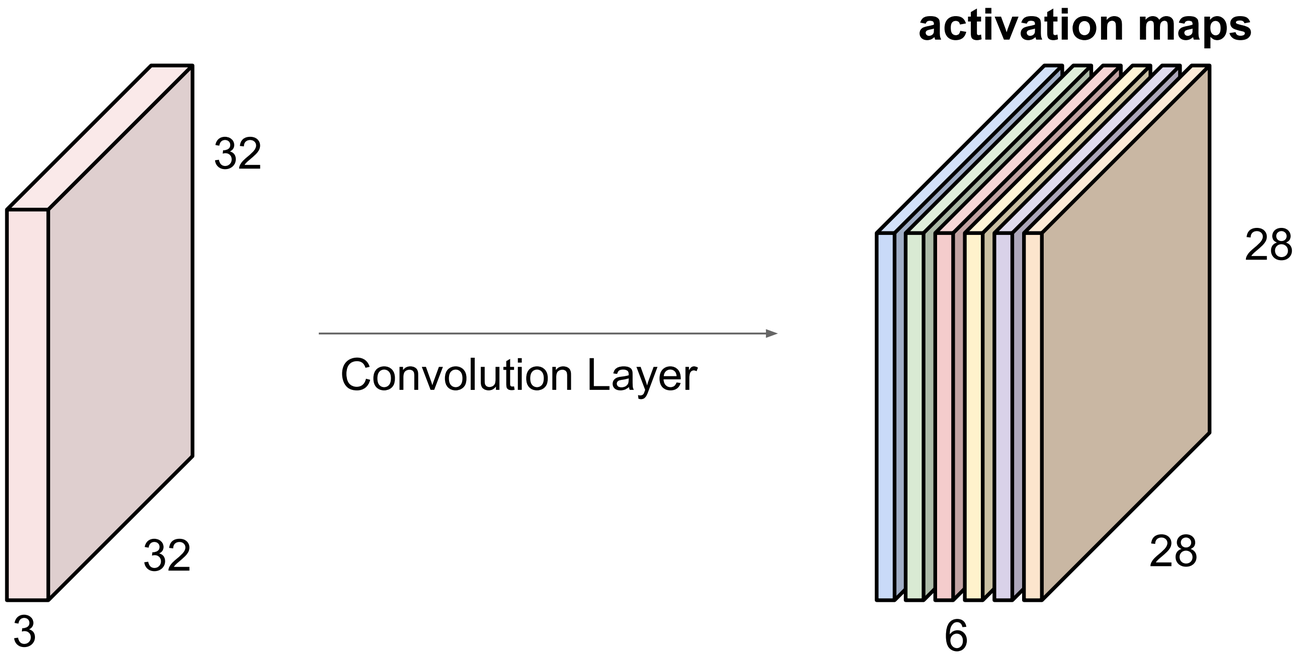
# Convolutional Layer

consider a second, green filter



# Convolutional Layer

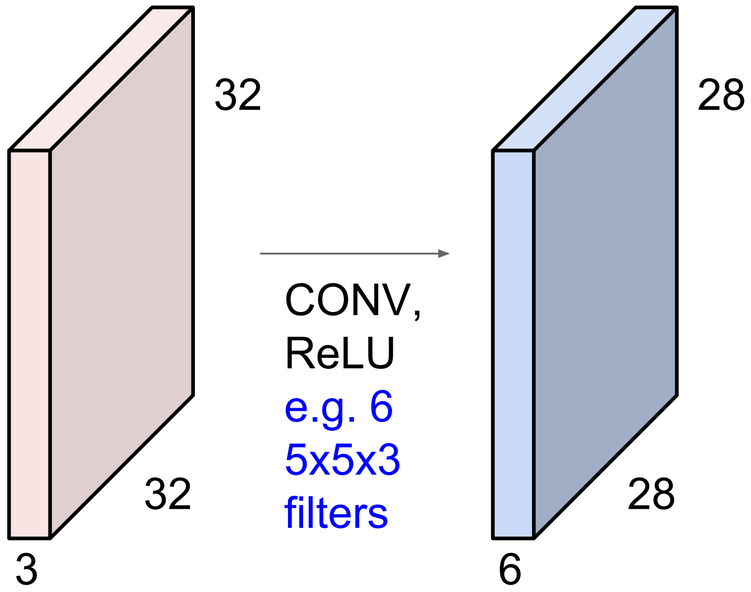
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:




We stack these up to get a "new image" of size 28x28x6!

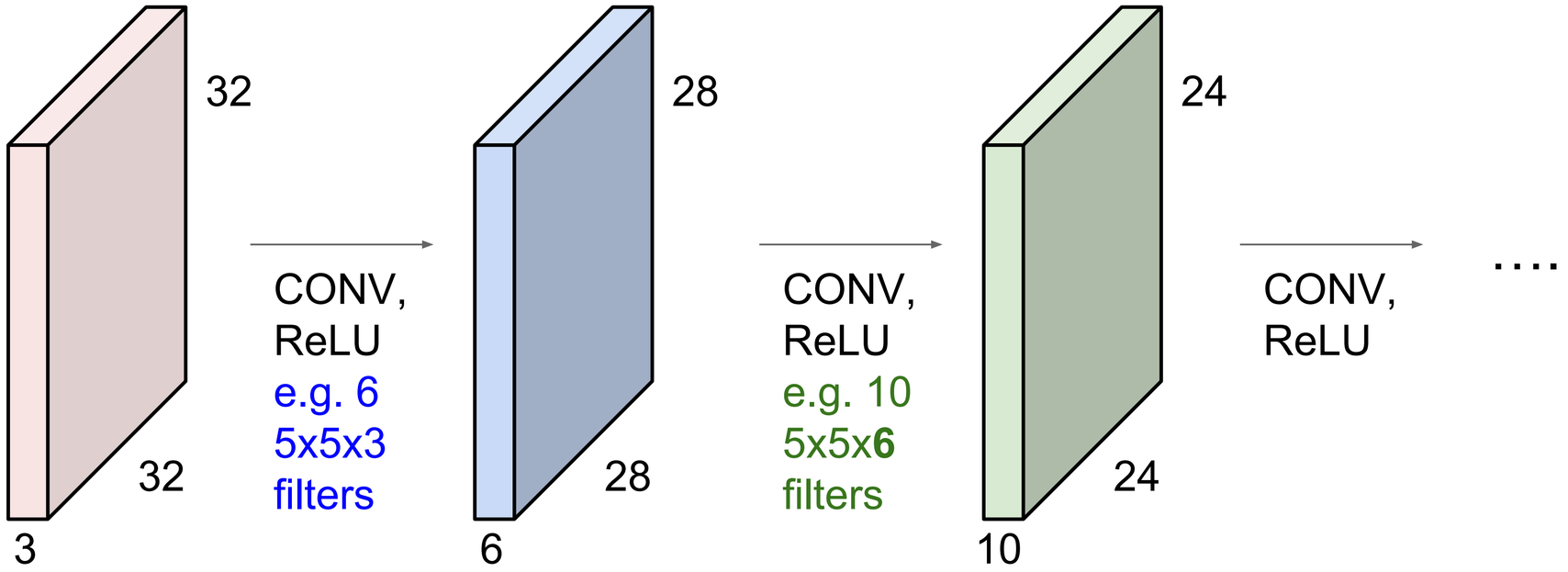
# Convolutional Layer

**Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions

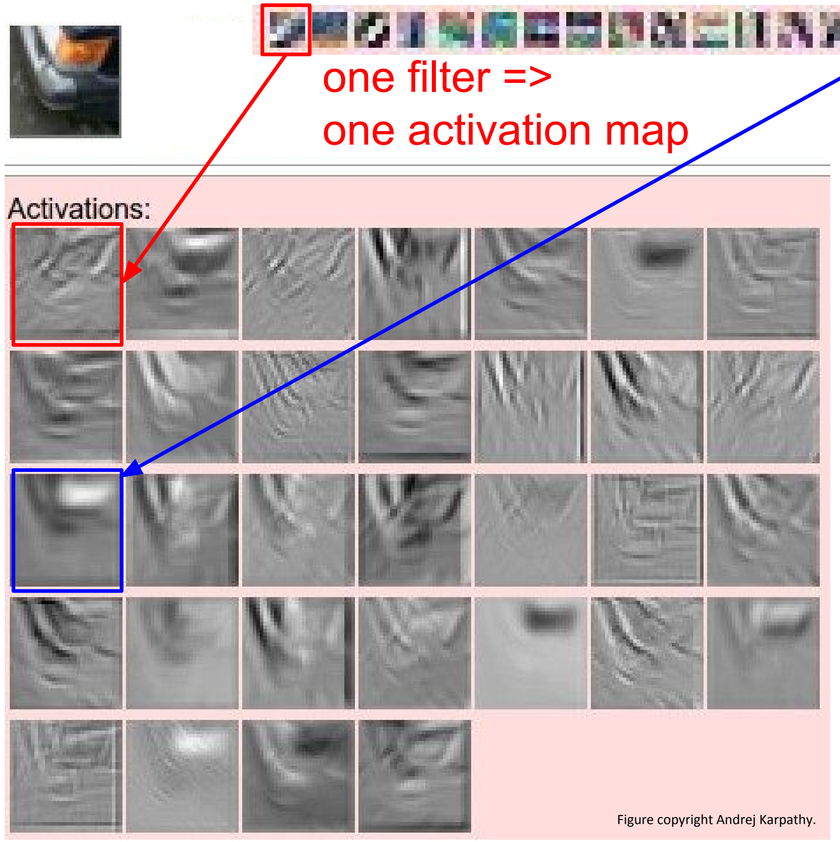


# Convolutional Layer

**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



# Convolutional Layer




example 5x5 filters
(32 total)

We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

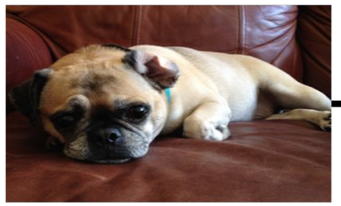
elementwise multiplication and sum of a filter and the signal (image)

# Convolutional Layer

**Preview**

*[Zeiler and Fergus 2013]*

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].

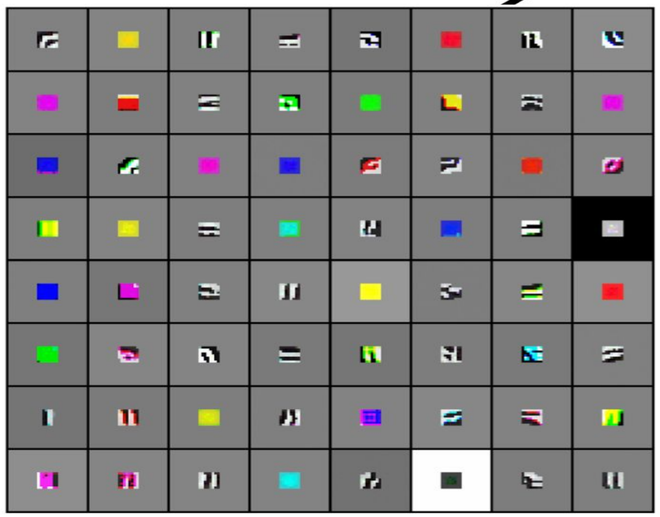


Low-level features

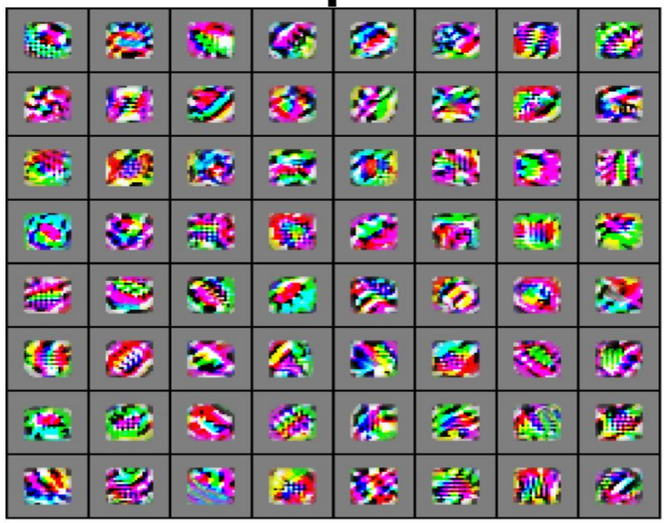
Mid-level features

High-level features

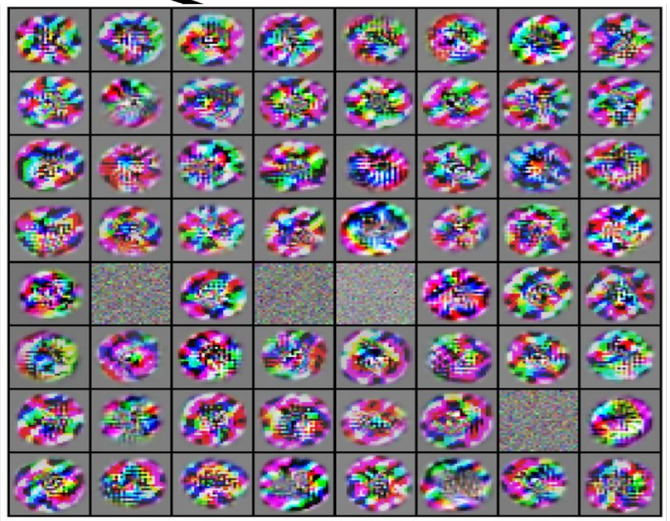
Linearly separable classifier



VGG-16 Conv1_1



VGG-16 Conv3_2

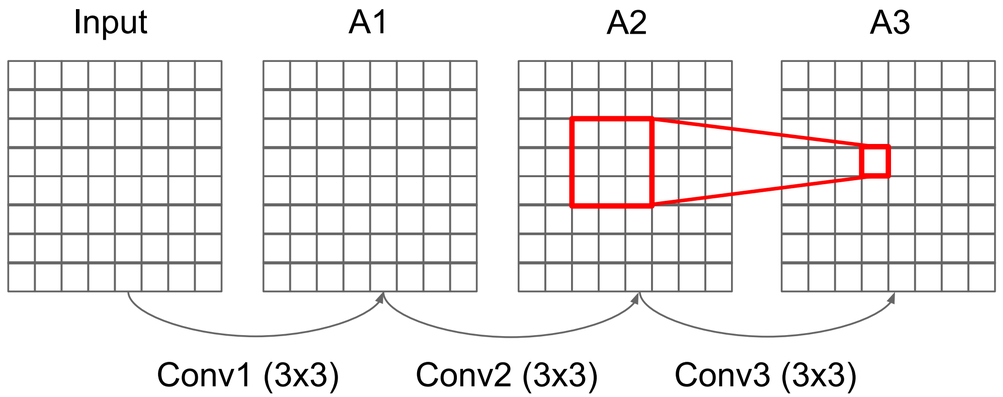


VGG-16 Conv5_3

# Receptive Field

*[Simonyan and Zisserman, 2014]*

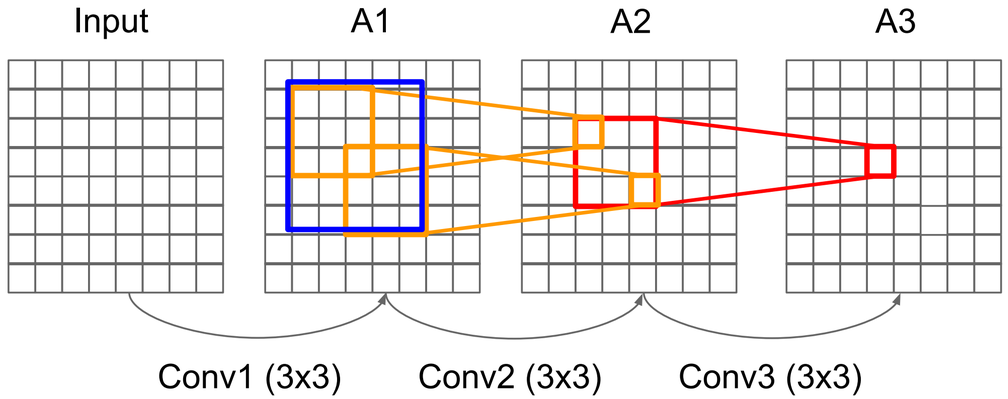
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



# Receptive Field

*[Simonyan and Zisserman, 2014]*

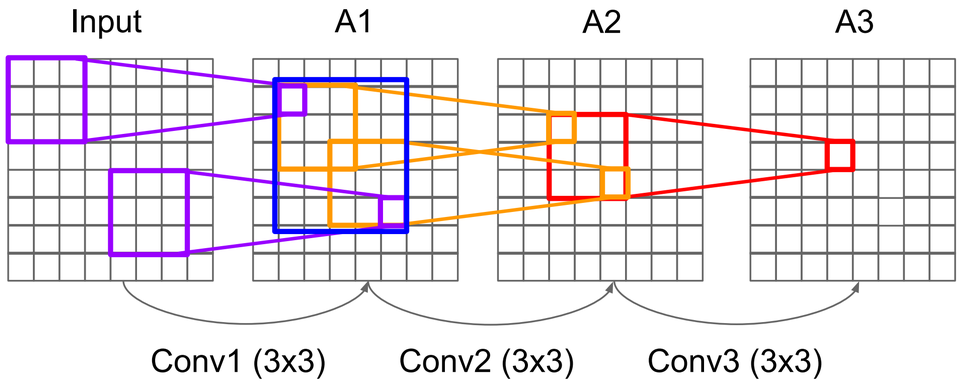
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



# Receptive Field

*[Simonyan and Zisserman, 2014]*

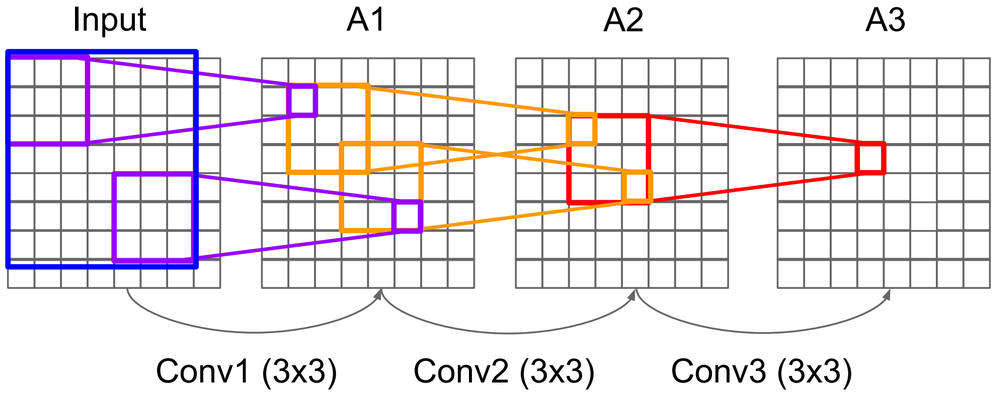
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



# Receptive Field

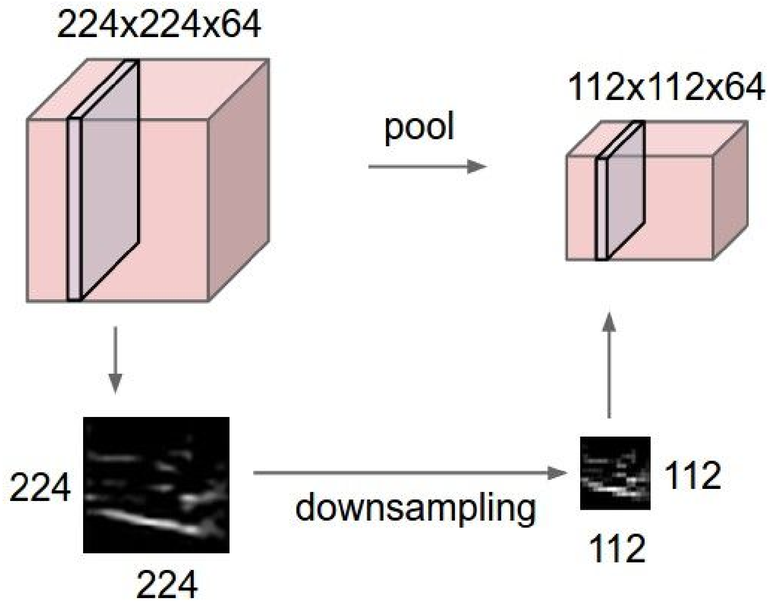
*[Simonyan and Zisserman, 2014]*

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

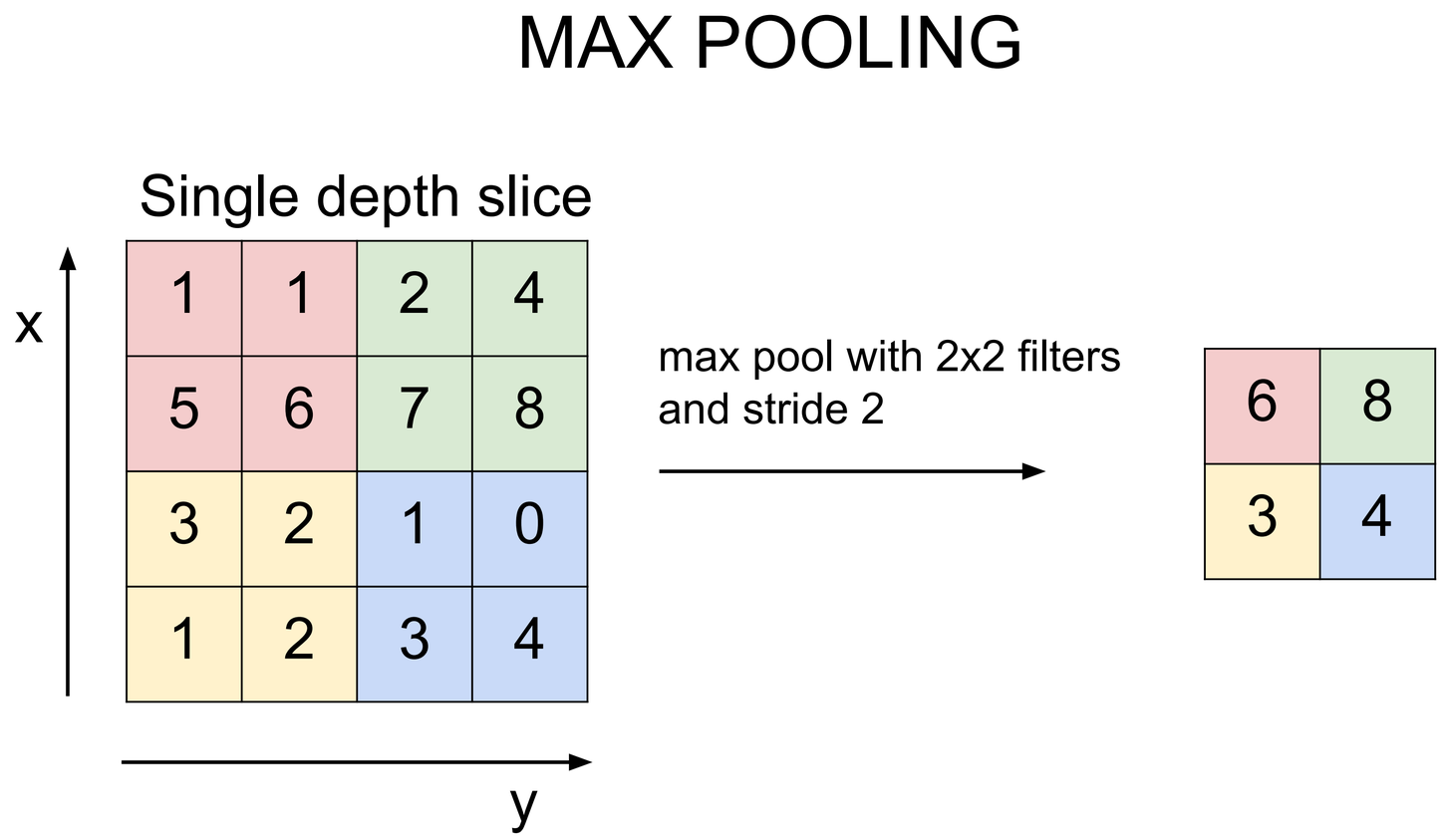


# Pooling Layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



# Pooling Layer



# Summary: From Convolutional Layer to Max Pooling

Input image 6x6x1

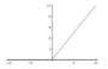
| | | | | | |
|---|---|---|---|---|---|
| 255 | 220 | 150 | 200 | 110 | 100 |
| 240 | 50 | 35 | 45 | 200 | 130 |
| 0 | 20 | 245 | 250 | 230 | 120 |
| 170 | 180 | 235 | -0.7 | 0.2 | 0.1 |
| 190 | 185 | 170 | 0.3 | 0.5 | 0.4 |
| 255 | 255 | 245 | -0.2 | -0.4 | 0.2 |

| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

Feature map 4x4x1

| | | | |
|---|---|---|---|
| 32.5 | -105.5 | 185.5 | 54 |
| -105.5 | 104 | 217.5 | 31 |
| -44 | 224 | 38.5 | -18 |
| -60.5 | 213.5 | 52.5 | 37.5 |



Relu

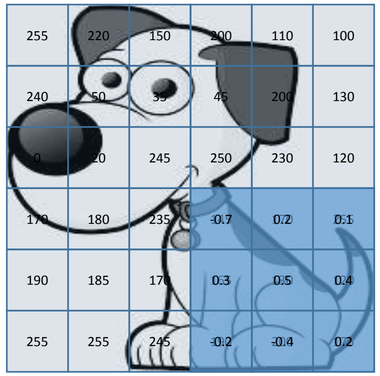
| | | | |
|---|---|---|---|
| 32.5 | 0 | 185.5 | 54 |
| 0 | 104 | 217.5 | 31 |
| 0 | 224 | 38.5 | 0 |
| 0 | 213.5 | 52.5 | 37.5 |

Maxpool (2x2x1)

| |
|---|
| 104 |

# Summary: From Convolutional Layer to Max Pooling

Input image 6x6x1

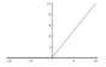



| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

Feature map 4x4x1

| | | | |
|---|---|---|---|
| 32.5 | -105.5 | 185.5 | 54 |
| -105.5 | 104 | 217.5 | 31 |
| -44 | 224 | 38.5 | -18 |
| -60.5 | 213.5 | 52.5 | 37.5 |



Relu

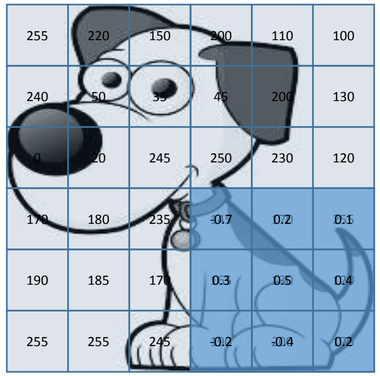
| | | | |
|---|---|---|---|
| 32.5 | 0 | 185.5 | 54 |
| 0 | 104 | 217.5 | 31 |
| 0 | 224 | 38.5 | 0 |
| 0 | 213.5 | 52.5 | 37.5 |

Maxpool (2x2x1)

| | |
|---|---|
| 104 | 217.5 |

# Summary: From Convolutional Layer to Max Pooling

Input image 6x6x1




| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

Feature map 4x4x1

| | | | |
|---|---|---|---|
| 32.5 | -105.5 | 185.5 | 54 |
| -105.5 | 104 | 217.5 | 31 |
| -44 | 224 | 38.5 | -18 |
| -60.5 | 213.5 | 52.5 | 37.5 |

Relu

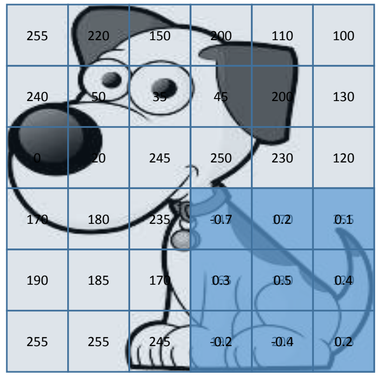
| | | | |
|---|---|---|---|
| 32.5 | 0 | 185.5 | 54 |
| 0 | 104 | 217.5 | 31 |
| 0 | 224 | 38.5 | 0 |
| 0 | 213.5 | 52.5 | 37.5 |

Maxpool (2x2x1)

| | |
|---|---|
| 104 | 217.5 |
| 224 | |

# Summary: From Convolutional Layer to Max Pooling

Input image 6x6x1




| | | |
|---|---|---|
| -0.7 | 0.2 | 0.1 |
| 0.3 | 0.5 | 0.4 |
| -0.2 | -0.4 | 0.2 |

3x3 filter

Feature map 4x4x1

| | | | |
|---|---|---|---|
| 32.5 | -105.5 | 185.5 | 54 |
| -105.5 | 104 | 217.5 | 31 |
| -44 | 224 | 38.5 | -18 |
| -60.5 | 213.5 | 52.5 | 37.5 |

Relu

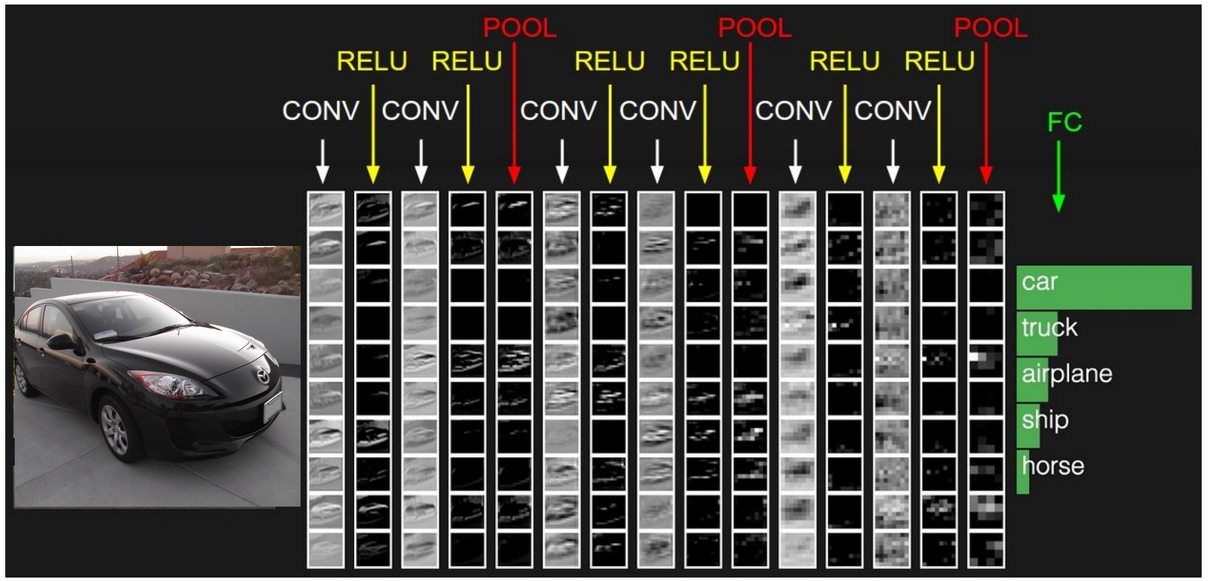
| | | | |
|---|---|---|---|
| 32.5 | 0 | 185.5 | 54 |
| 0 | 104 | 217.5 | 31 |
| 0 | 224 | 38.5 | 0 |
| 0 | 213.5 | 52.5 | 37.5 |

Maxpool (2x2x1)

| | |
|---|---|
| 104 | 217.5 |
| 224 | 52.5 |

# Final Layer: Fully Connected Layer

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



# CNN Architectures

Typical architectures look like

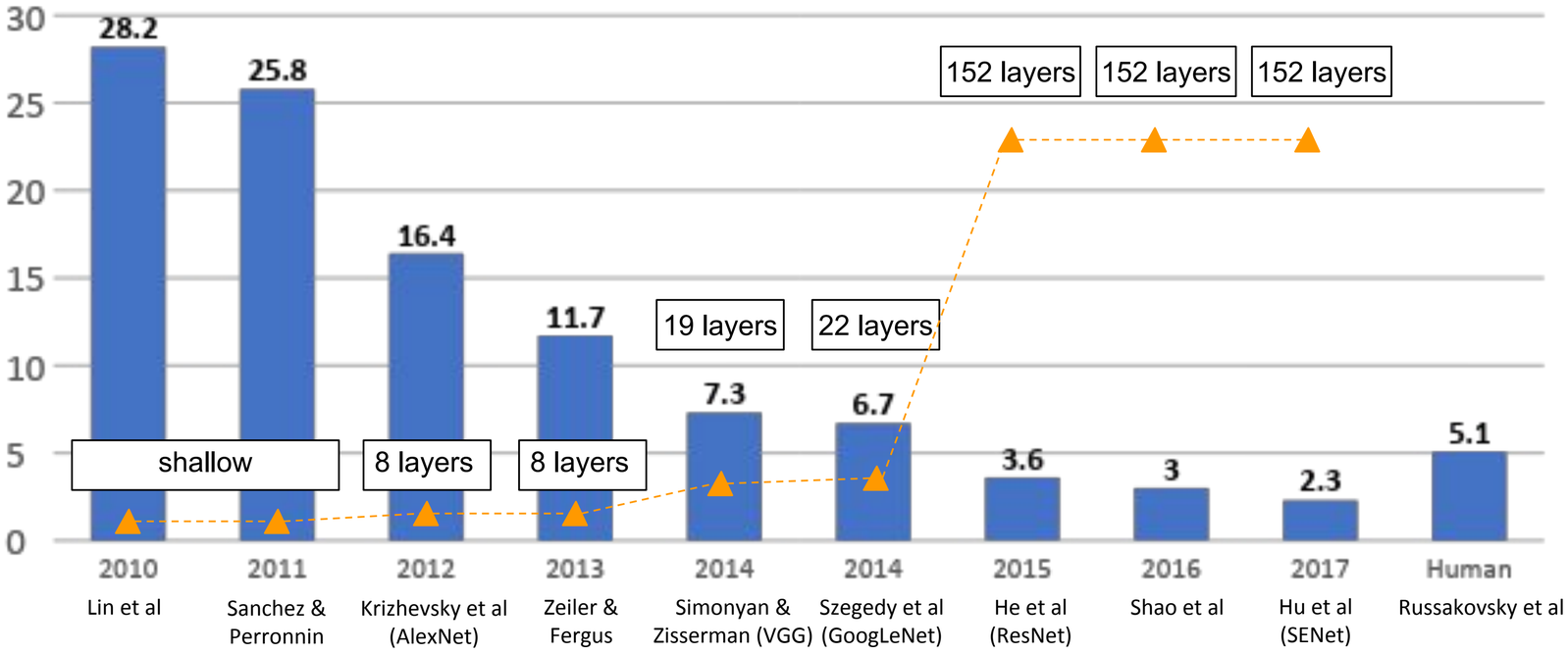
**[(CONV → RELU)*N → POOL]*M → (FC → RELU)*K → SOFTMAX**

N is usually up to ~5, M is large, 0 <= K <= 2.

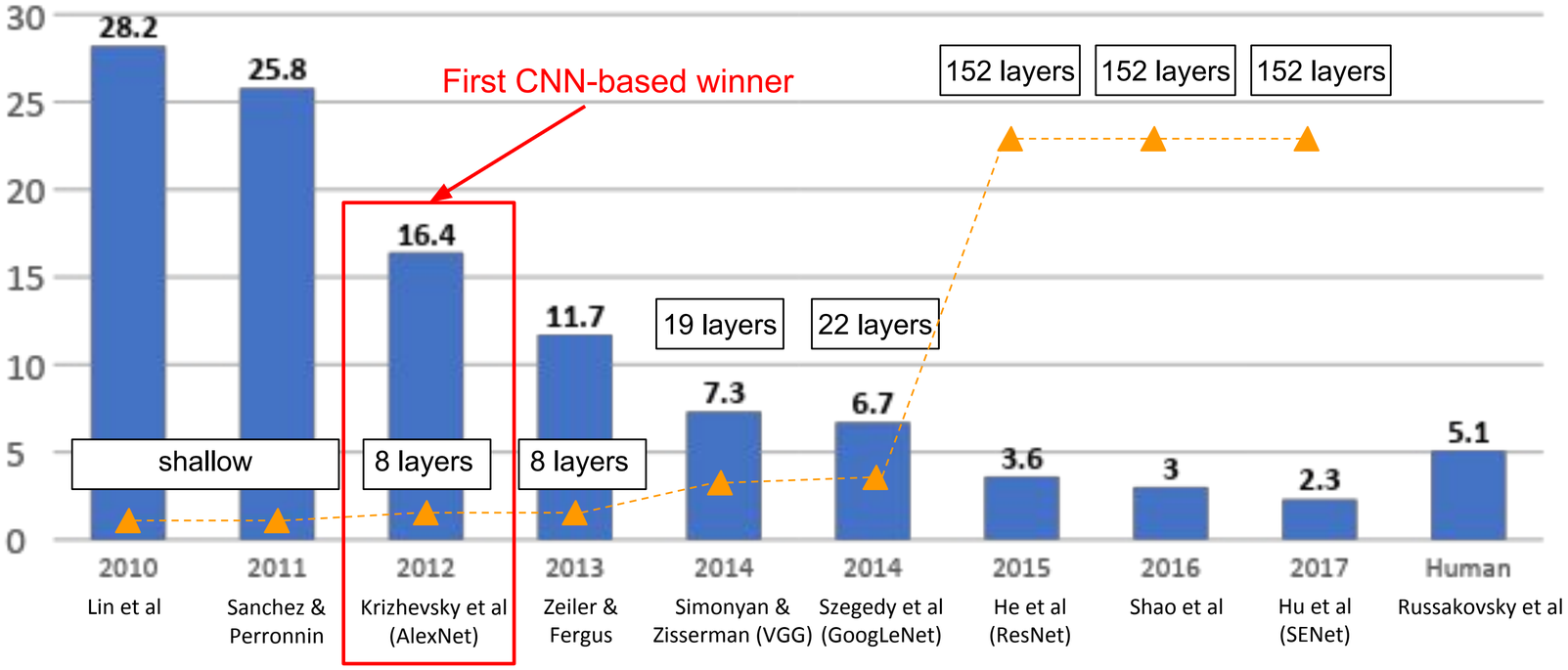
but recent advances such as ResNet/GoogLeNet challenge this paradigm

# Exercises

# ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



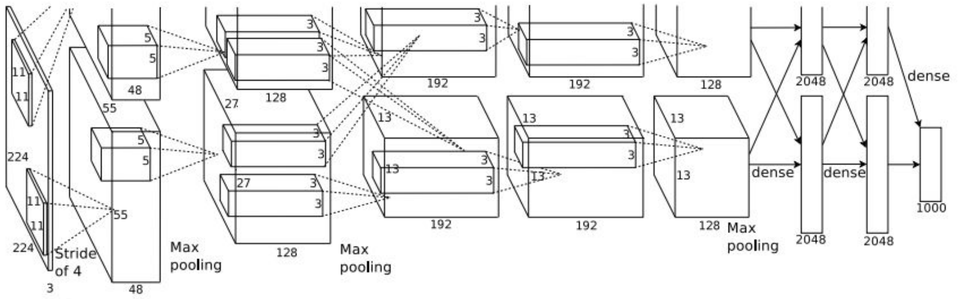
# ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



# CNN Architecture : AlexNet

## Case Study: AlexNet

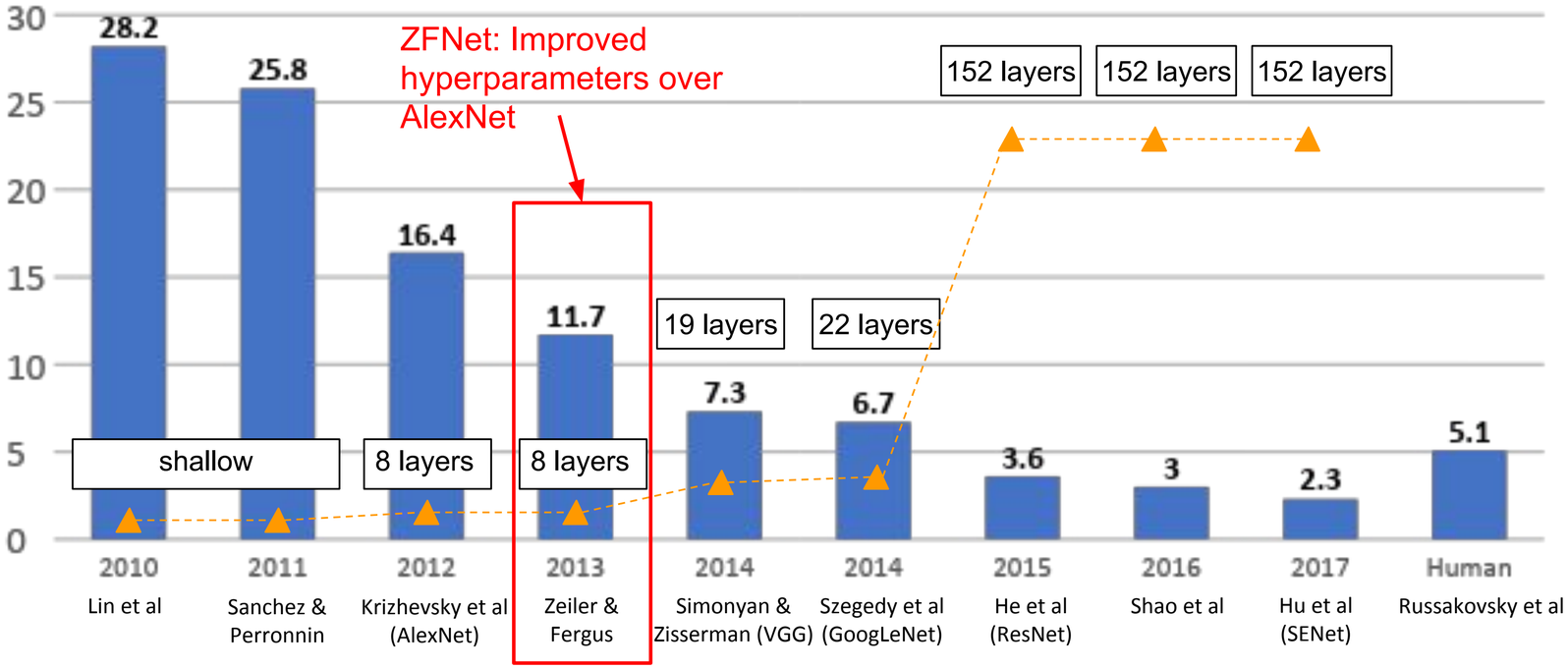
*[Krizhevsky et al. 2012]*



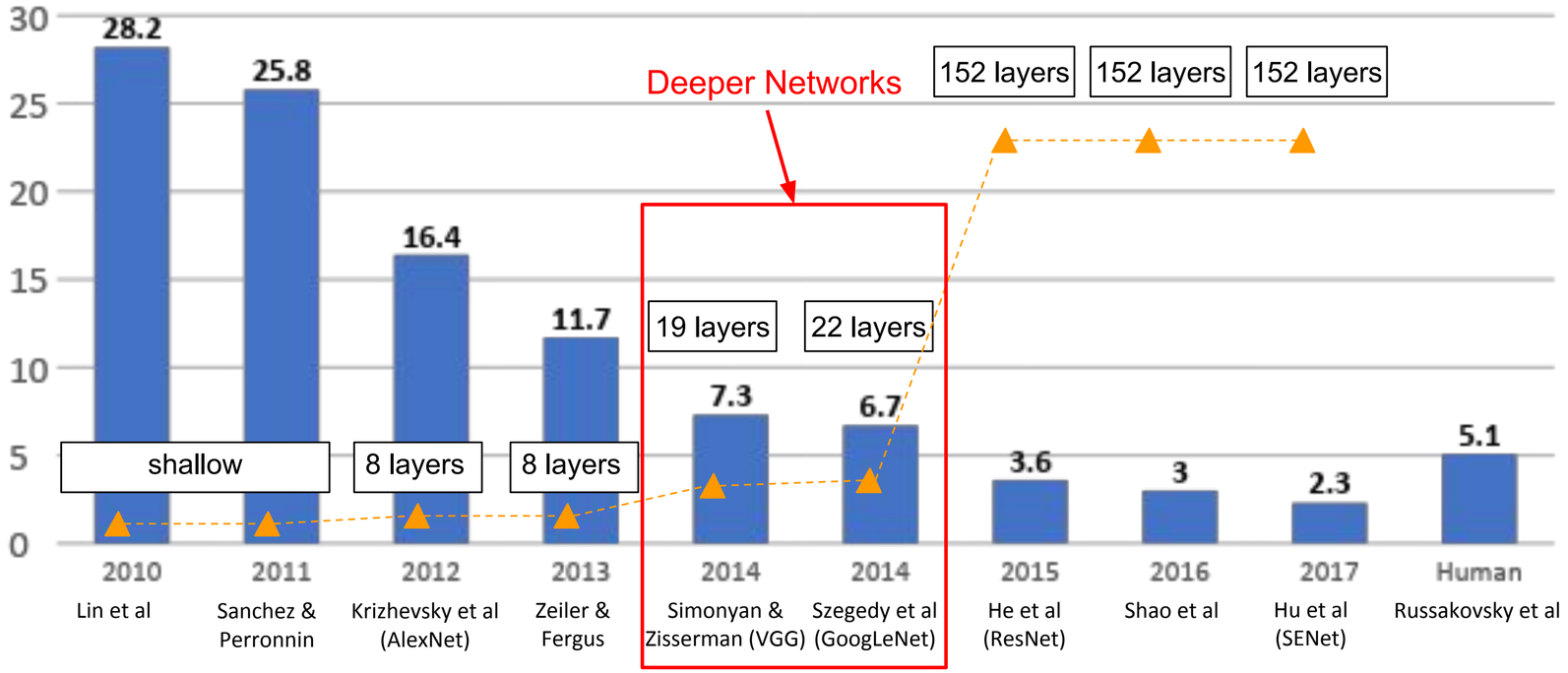

Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

**Details/Retrospectives:**
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

# CNN Architecture : ZF Net



# CNN Architecture : VGG16 and GoogLeNet



# CNN Architecture : VGG16

## Case Study: VGGNet

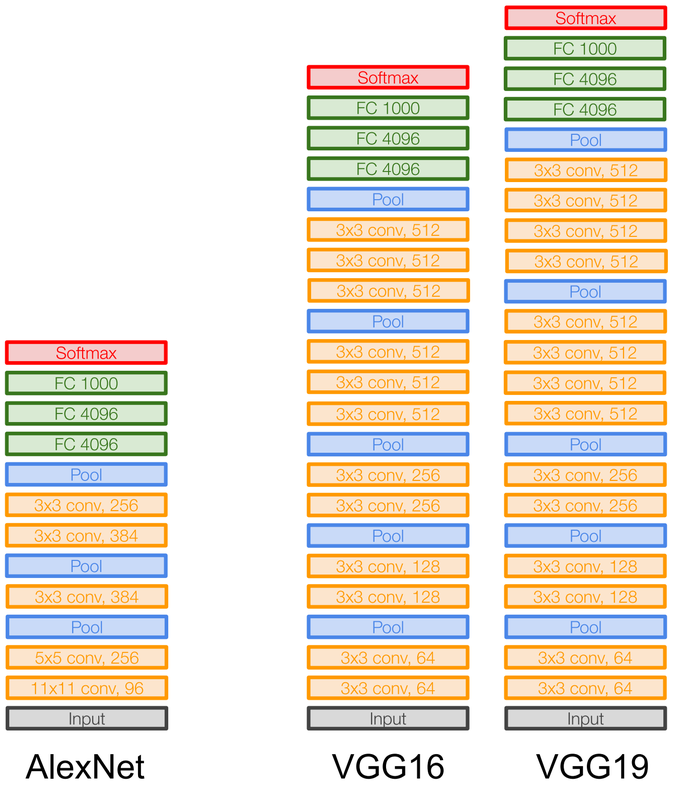
*[Simonyan and Zisserman, 2014]*

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



# CNN Architecture : VGG16

*[Simonyan and Zisserman, 2014]*

Q: Why use smaller filters? (3x3 conv)

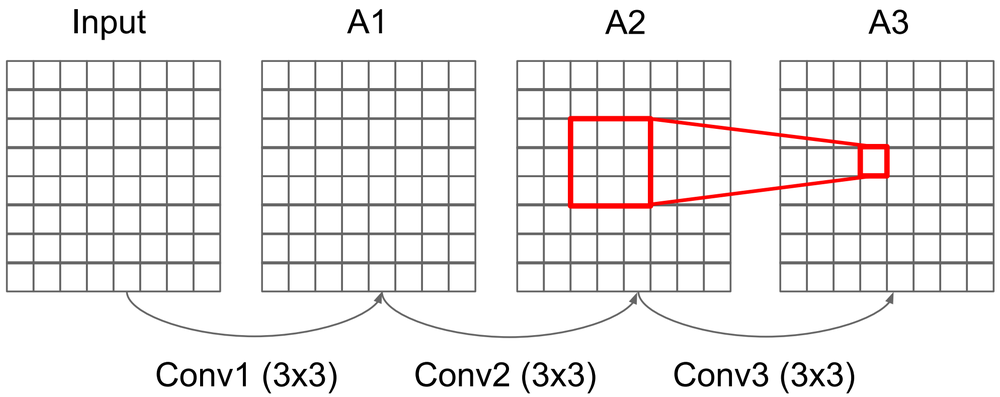
Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

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

# CNN Architecture : VGG16

*[Simonyan and Zisserman, 2014]*

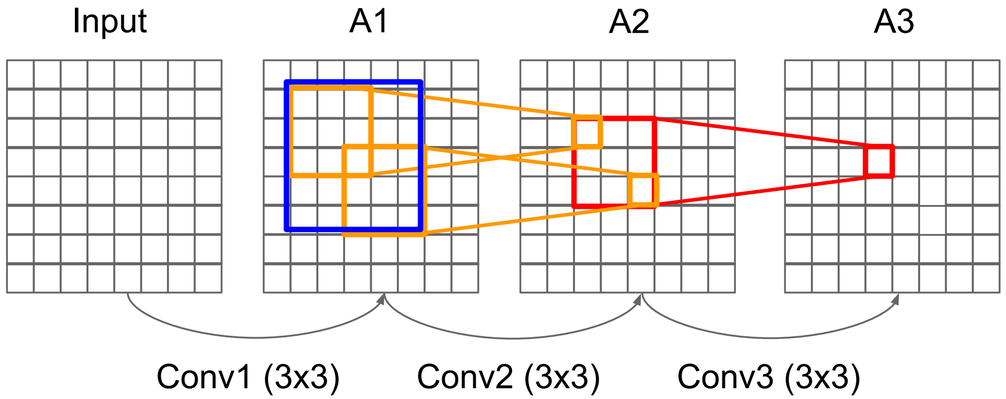
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?




# CNN Architecture : VGG16

*[Simonyan and Zisserman, 2014]*

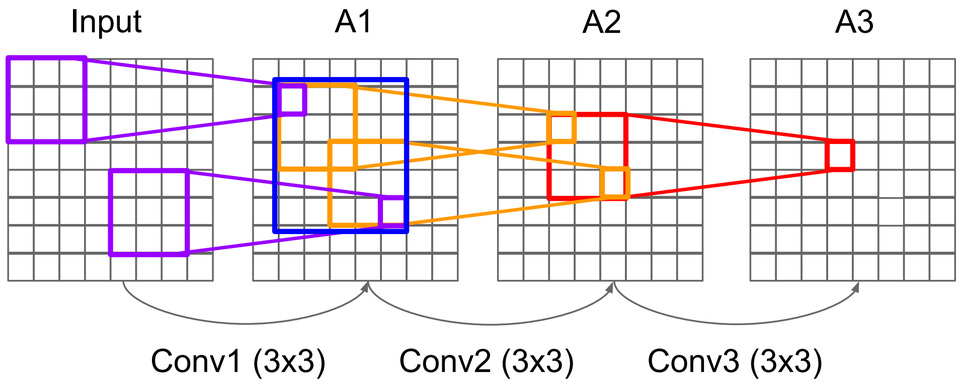
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?




# CNN Architecture : VGG16

*[Simonyan and Zisserman, 2014]*

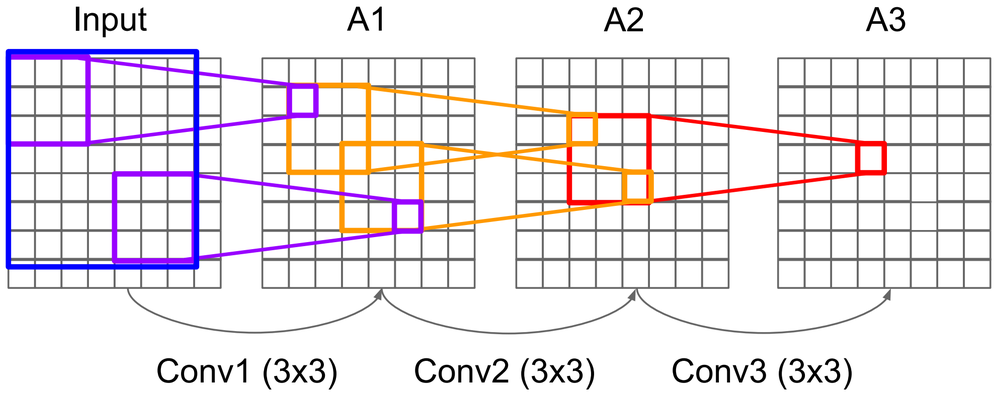
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



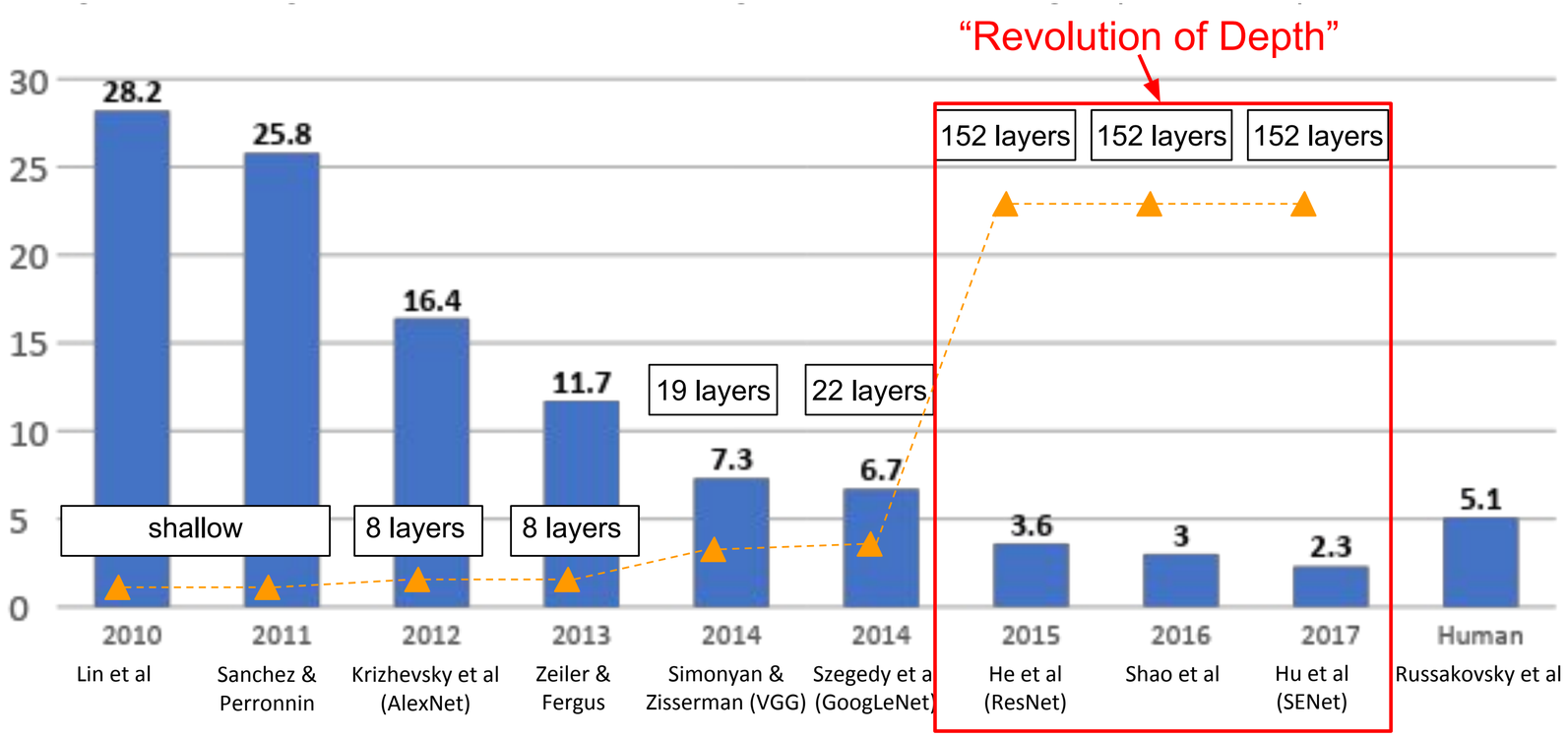

# CNN Architecture : VGG16

*[Simonyan and Zisserman, 2014]*

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




# CNN Architecture : ResNet



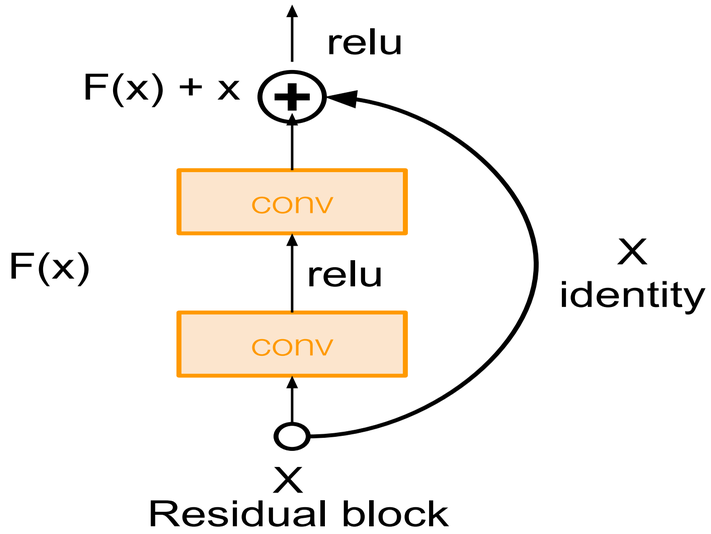
# CNN Architecture : ResNet

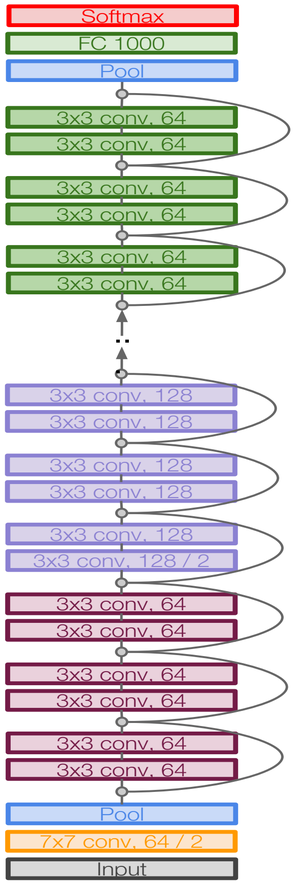
## Case Study: ResNet

*[He et al., 2015]*

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!



# Convolutional Neural Networks in Practice: Fine-tuning existing models

The discussed models were trained on the **ImageNet dataset**, which contains more than **14 million labeled images**!

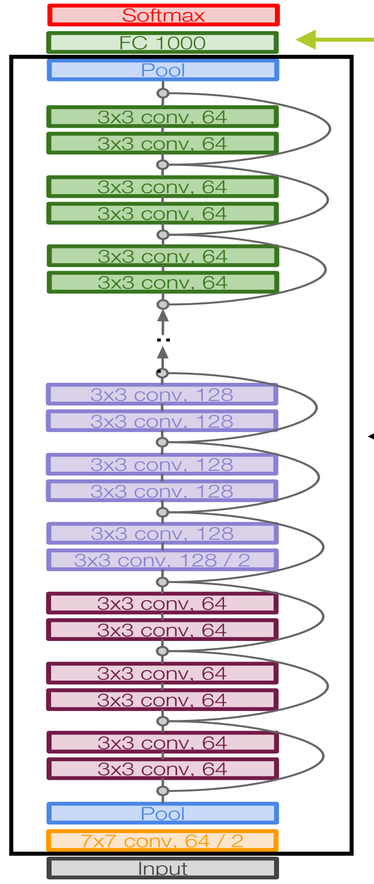
In practice, you rarely have access to labeled datasets of this size.

What to do?

- Fine-tuning an existing model

**Available models**

| Model | Size | Top-1 Accuracy | Top-5 Accuracy | Parameters | Depth |
|---|---|---|---|---|---|
| Xception | 88 MB | 0.790 | 0.945 | 22,910,480 | 126 |
| VGG16 | 528 MB | 0.713 | 0.901 | 138,357,544 | 23 |
| VGG19 | 549 MB | 0.713 | 0.900 | 143,667,240 | 26 |
| ResNet50 | 98 MB | 0.749 | 0.921 | 25,636,712 | - |
| ResNet101 | 171 MB | 0.764 | 0.928 | 44,707,176 | - |
| ResNet152 | 232 MB | 0.766 | 0.931 | 60,419,944 | - |
| ResNet50V2 | 98 MB | 0.760 | 0.930 | 25,613,800 | - |
| ResNet101V2 | 171 MB | 0.772 | 0.938 | 44,675,560 | - |
| ResNet152V2 | 232 MB | 0.780 | 0.942 | 60,380,648 | - |
| InceptionV3 | 92 MB | 0.779 | 0.937 | 23,851,784 | 159 |
| InceptionResNetV2 | 215 MB | 0.803 | 0.953 | 55,873,736 | 572 |
| MobileNet | 16 MB | 0.704 | 0.895 | 4,253,864 | 88 |
| MobileNetV2 | 14 MB | 0.713 | 0.901 | 3,538,984 | 88 |
| DenseNet121 | 33 MB | 0.750 | 0.923 | 8,062,504 | 121 |
| DenseNet169 | 57 MB | 0.762 | 0.932 | 14,307,880 | 169 |
| DenseNet201 | 80 MB | 0.773 | 0.936 | 20,242,984 | 201 |
| NASNetMobile | 23 MB | 0.744 | 0.919 | 5,326,716 | - |
| NASNetLarge | 343 MB | 0.825 | 0.960 | 88,949,818 | - |
| EfficientNetB0 | 29 MB | - | - | 5,330,571 | - |
| EfficientNetB1 | 31 MB | - | - | 7,856,239 | - |
| EfficientNetB2 | 36 MB | - | - | 9,177,569 | - |
| EfficientNetB3 | 48 MB | - | - | 12,320,535 | - |
| EfficientNetB4 | 75 MB | - | - | 19,466,823 | - |
| EfficientNetB5 | 118 MB | - | - | 30,562,527 | - |
| EfficientNetB6 | 166 MB | - | - | 43,265,143 | - |
| EfficientNetB7 | 256 MB | - | - | 66,658,687 | - |




Retrain only this layer with your data

- Learning few parameters requires few training samples
- Assumption: the ”stem” of the network learned to extract generally useful features

**Reuse as-is**

# Convolutional Neural Networks in Practice: Augmenting your Dataset

The discussed models were trained on the **ImageNet dataset**, which contains more than **14 million labeled images**!

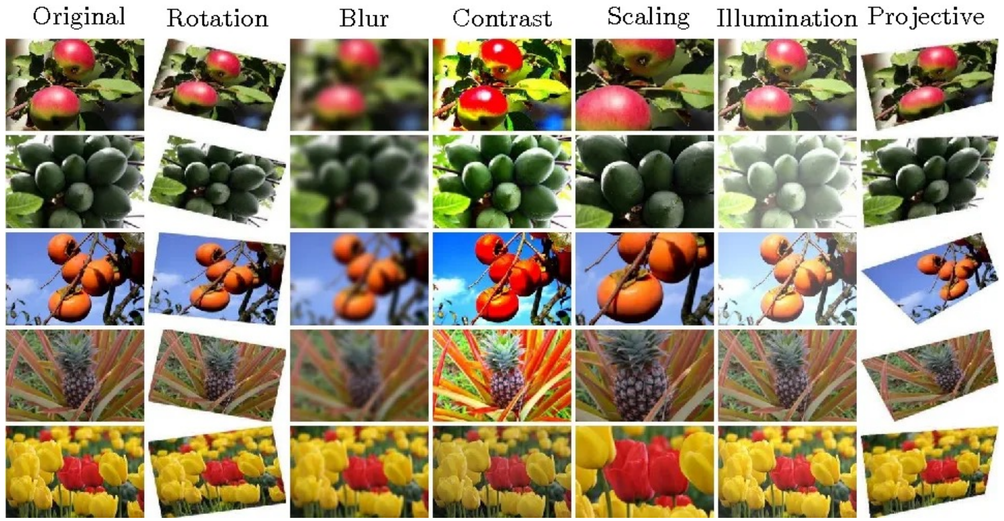
In practice, you rarely have access to labeled datasets of this size.

What to do?

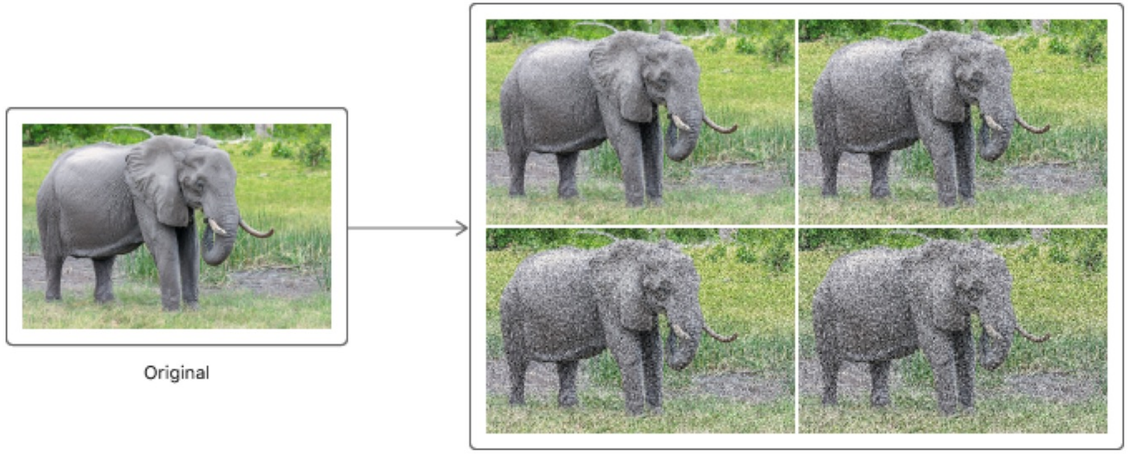
- Fine-tuning an existing model and/or
- Augmenting your Dataset

As long as a human can classify the image, the CNN should be able to do too!

It also makes the CNN more robust!




Source: https://medium.com/analytics-vidhya/data-augmentation-is-it-really-necessary-b3cb12ab3c3f




Source: https://developer.apple.com/documentation/createml/mlimageclassifier/imageaugmentationoptions/noise

# Contrastive Learning of Visual Representations (e.g., SimCLR; Chen et al. 2020)

**Self-supervised learning**
Learn image representations without label data!

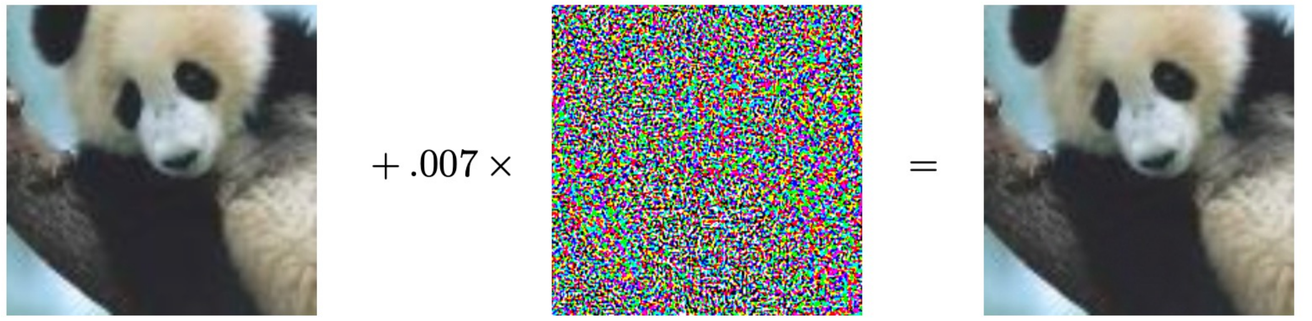
Goals:
- Augmented patches from **same image** should produce **similar representations**
- Patches from **different images** should produce **dissimilar representations**

**Semi-supervised learning**
Use representations from self-supervised learning step as input to a neural network that is trained with supervised learning.

- If the learned representations are good, only little training data is required for that!
- Like fine-tuning, but representations are learned without labels.

Source: https://blog.research.google/2020/04/advancing-self-supervised-and-semi.html

# Generating Adversarial Images




Adding a humanly imperceptible change to the input image is sufficient to confuse the CNN

How? Use backpropagation to compute **how the image should be changed** to **worsen the output**.

Note: During training we compute how the parameter should be changed to improve the output.

This is a "**white box**" attack (as opposed to black box): We need to have access to the model weights to backpropagate through it.