

Lecture 3:

Convolutional Neural Networks

Prof. Manolis Kellis

Today: Convolutional Neural Networks (CNNs)

1. Scene understanding and object recognition for machines (and humans)

- Scene/object recognition challenge. Illusions reveal primitives, conflicting info
- Human neurons/circuits. Visual cortex layers==abstraction. General cognition

2. Classical machine vision foundations: features, scenes, filters, convolution

- Spatial structure primitives: edge detectors & other filters, feature recognition
- Convolution: basics, padding, stride, object recognition, architectures

3. CNN foundations: LeNet, *de novo* feature learning, parameter sharing

- Key ideas: *learn* features, hierarchy, re-use parameters, back-prop filter learning
- CNN formalization: representations(Conv+ReLU+Pool)*N layers + Fully-connected

4. Modern CNN architectures: millions of parameters, dozens of layers

- Feature invariance is hard: apply perturbations, learn for each variation
- ImageNet progression of best performers
- AlexNet: First top performer CNN, 60M parameters (from 60k in LeNet-5), ReLU
- VGGNet: simpler but deeper (8→19 layers), 140M parameters, ensembles
- GoogleNet: new primitive=inception module, 5M params, no FC, efficiency
- ResNet: 152 layers, vanishing gradients → fit residuals to enable learning

5. Countless applications: General architecture, enormous power

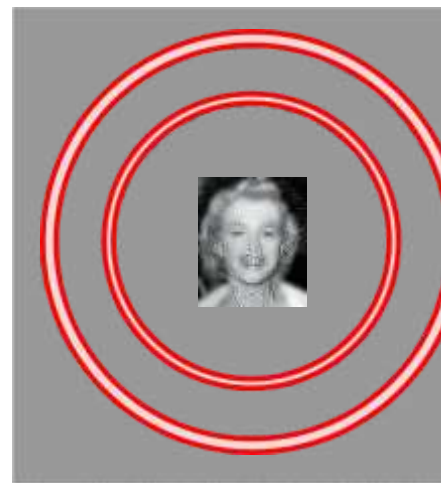
- Semantic segmentation, facial detection/recognition, self-driving, image colorization, optimizing pictures/scenes, up-scaling, medicine, biology, genomics

1a. What do you see, and how?
Can we teach machines to see?

What do *you* see?



How do you see?



How can we help computers see?



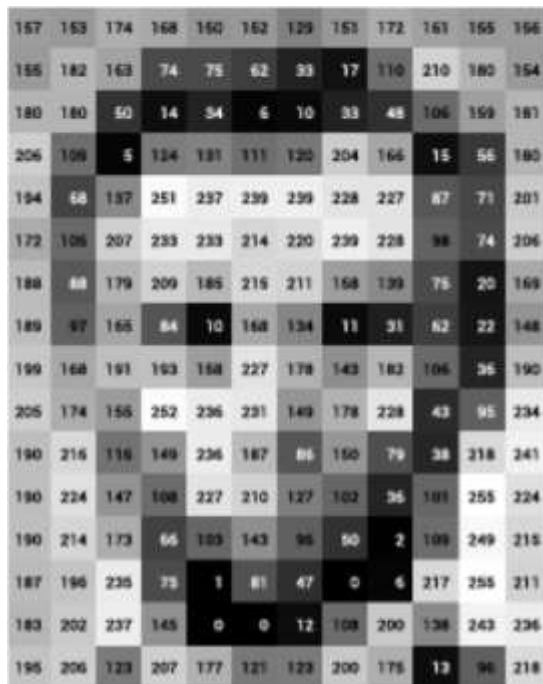
What computers 'see': Images as Numbers

What you see



Input Image

What you both see



Input Image + values

What the computer "sees"

| | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 157 | 153 | 174 | 168 | 150 | 152 | 129 | 151 | 172 | 161 | 155 | 156 |
| 155 | 182 | 163 | 74 | 75 | 62 | 33 | 17 | 110 | 210 | 180 | 154 |
| 180 | 180 | 50 | 14 | 34 | 6 | 10 | 33 | 48 | 106 | 159 | 181 |
| 206 | 109 | 5 | 124 | 131 | 111 | 120 | 204 | 166 | 15 | 56 | 180 |
| 194 | 58 | 137 | 251 | 237 | 239 | 239 | 228 | 227 | 87 | 71 | 201 |
| 172 | 106 | 207 | 233 | 233 | 214 | 220 | 239 | 228 | 98 | 74 | 206 |
| 188 | 88 | 179 | 209 | 185 | 215 | 211 | 158 | 139 | 75 | 20 | 169 |
| 189 | 97 | 165 | 84 | 10 | 168 | 134 | 11 | 31 | 62 | 22 | 148 |
| 199 | 168 | 191 | 193 | 158 | 227 | 178 | 143 | 182 | 106 | 36 | 190 |
| 205 | 174 | 155 | 252 | 236 | 231 | 149 | 178 | 228 | 43 | 95 | 234 |
| 190 | 216 | 116 | 149 | 236 | 187 | 86 | 150 | 79 | 38 | 218 | 241 |
| 190 | 224 | 147 | 108 | 227 | 210 | 127 | 102 | 36 | 101 | 255 | 224 |
| 190 | 214 | 173 | 66 | 103 | 143 | 96 | 90 | 2 | 109 | 249 | 215 |
| 187 | 196 | 235 | 75 | 1 | 81 | 47 | 0 | 6 | 217 | 255 | 211 |
| 183 | 202 | 237 | 145 | 0 | 0 | 12 | 108 | 200 | 138 | 243 | 236 |
| 195 | 206 | 123 | 207 | 177 | 121 | 123 | 200 | 175 | 13 | 96 | 218 |

Pixel intensity values
("pix-el"=picture-element)

An image is just a matrix of numbers $[0,255]$. i.e., $1080 \times 1080 \times 3$ for an RGB image.

Question: is this Lincoln? Washington? Jefferson? Obama?

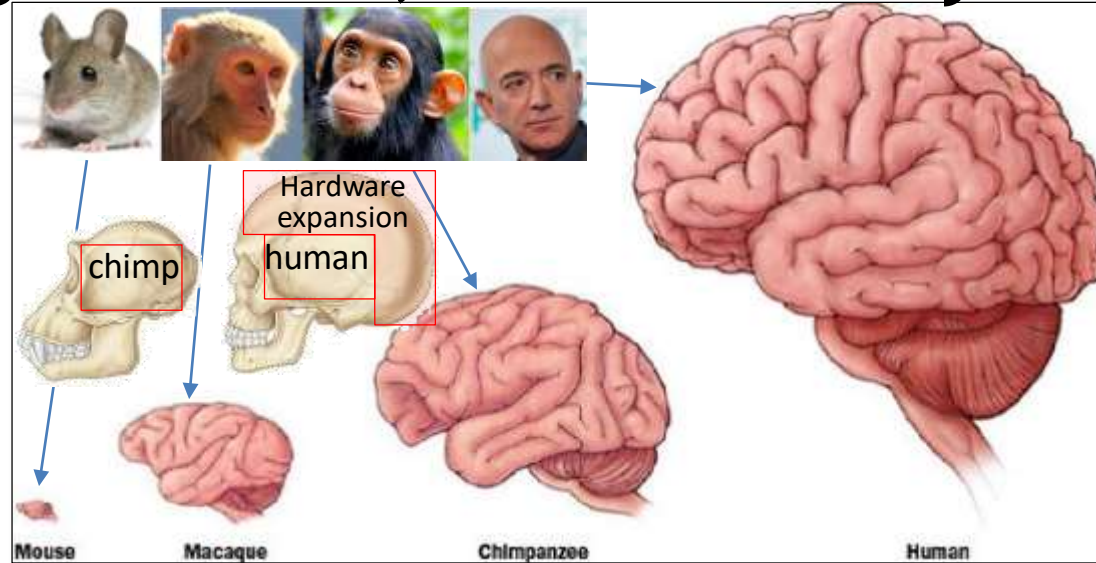
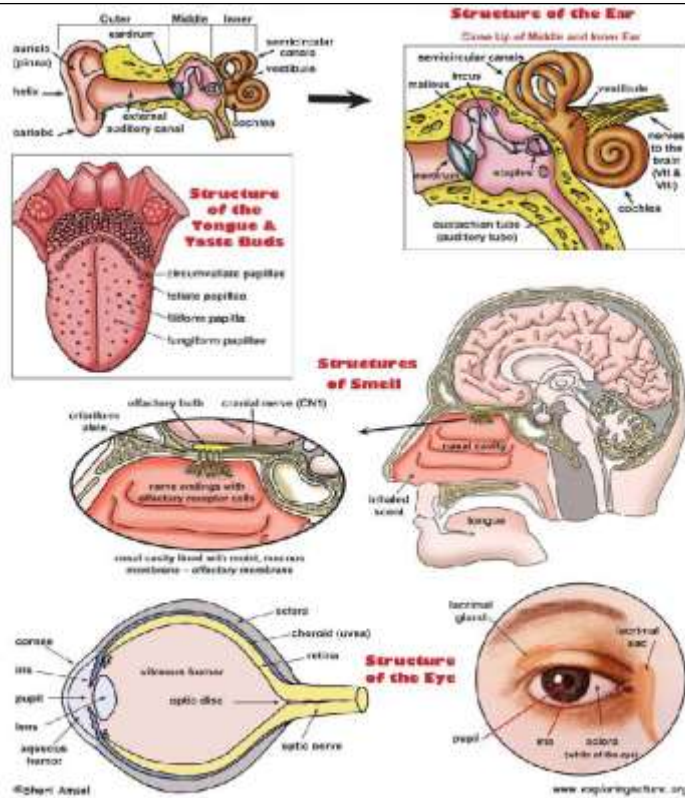
How can the computer answer this question?

Can I just do classification on the 1,166,400-long image vector directly?

No. Instead: exploit image spatial structure. Learn patches. Build them up

1b. Classical machine vision roots
in study of human/animal brains

General “learning machine”, reused widely

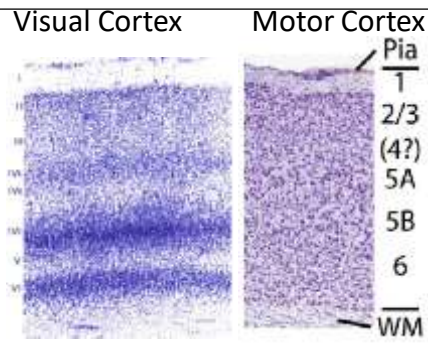


- Massive recent expanse of human brain has re-used a relatively simple but general learning architecture



- Not fully-general learning, but well-adapted to our world
- Humans co-opted this circuitry to many new applications
- Modern tasks accessible to any homo sapiens (<70k years)
- ML primitives not too different from animals: more to come?

- Hearing, taste, smell, sight, touch all re-use similar learning architecture



- Interchangeable circuitry
- Auditory cortex learns to ‘see’ if sent visual signals
- Injury area tasks shift to uninjured areas

Today: Convolutional Neural Networks (CNNs)

1. Scene understanding and object recognition for machines (and humans)

- Scene/object recognition challenge. Illusions reveal primitives, conflicting info
- Human neurons/circuits. Visual cortex layers==abstraction. General cognition

2. Classical machine vision foundations: features, scenes, filters, convolution

- Spatial structure primitives: edge detectors & other filters, feature recognition
- Convolution: basics, padding, stride, object recognition, architectures

3. CNN foundations: LeNet, *de novo* feature learning, parameter sharing

- Key ideas: *learn* features, hierarchy, re-use parameters, back-prop filter learning
- CNN formalization: representations(Conv+ReLU+Pool)*N layers + Fully-connected

4. Modern CNN architectures: millions of parameters, dozens of layers

- Feature invariance is hard: apply perturbations, learn for each variation
- ImageNet progression of best performers
- AlexNet: First top performer CNN, 60M parameters (from 60k in LeNet-5), ReLU
- VGGNet: simpler but deeper (8→19 layers), 140M parameters, ensembles
- GoogleNet: new primitive=inception module, 5M params, no FC, efficiency
- ResNet: 152 layers, vanishing gradients → fit residuals to enable learning

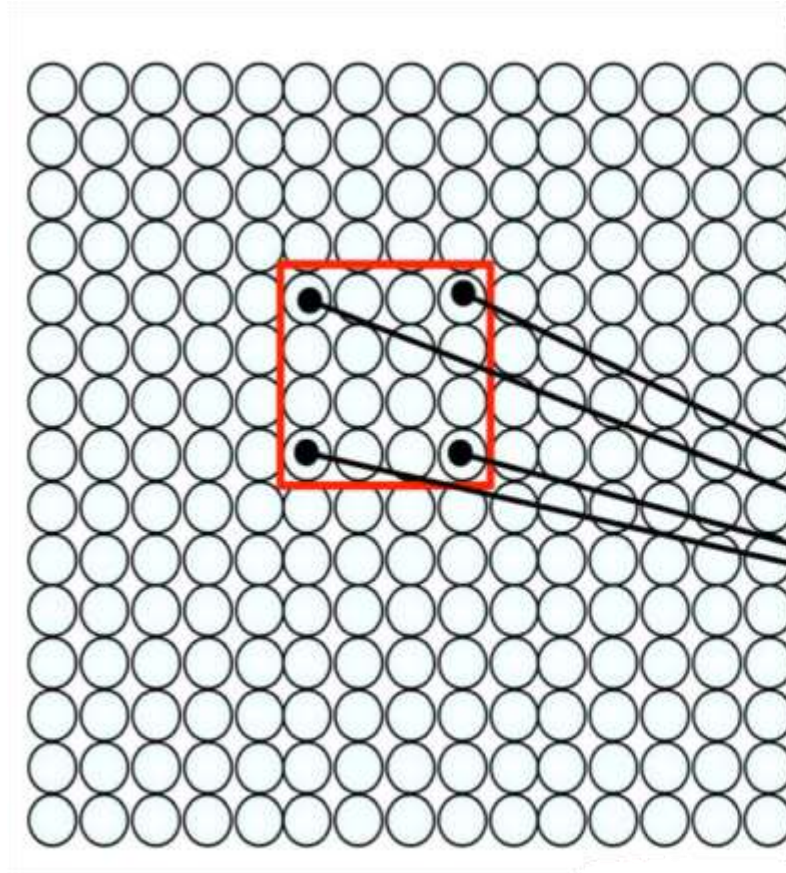
5. Countless applications: General architecture, enormous power

- Semantic segmentation, facial detection/recognition, self-driving, image colorization, optimizing pictures/scenes, up-scaling, medicine, biology, genomics

2a. Spatial structure for image recognition

Using Spatial Structure

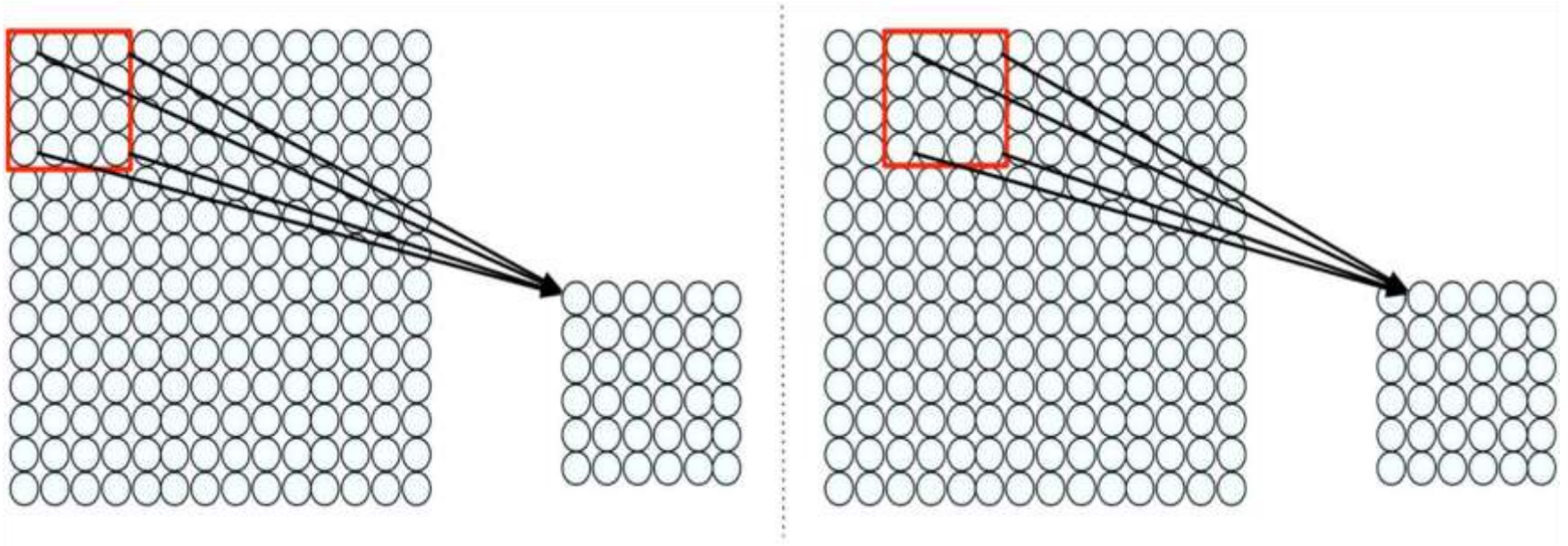
Input: 2D
image.
Array of pixel
values



Idea: connect
patches of input to
neurons in hidden
layer.

Neuron connected
to region of input.
Only “sees” these
values.

Using Spatial Structure

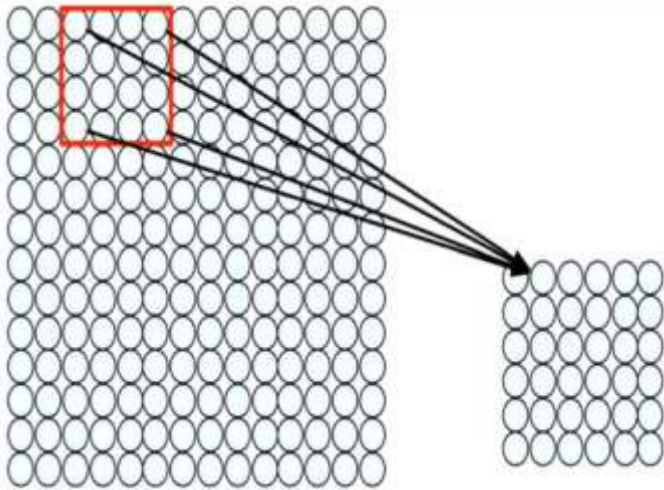


Connect patch in input layer to a single neuron in subsequent layer.

Use a sliding window to define connections.

How can we weight the patch to detect particular features?

Feature Extraction with Convolution



- Filter of size 4x4 :16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by 2 pixels for next patch

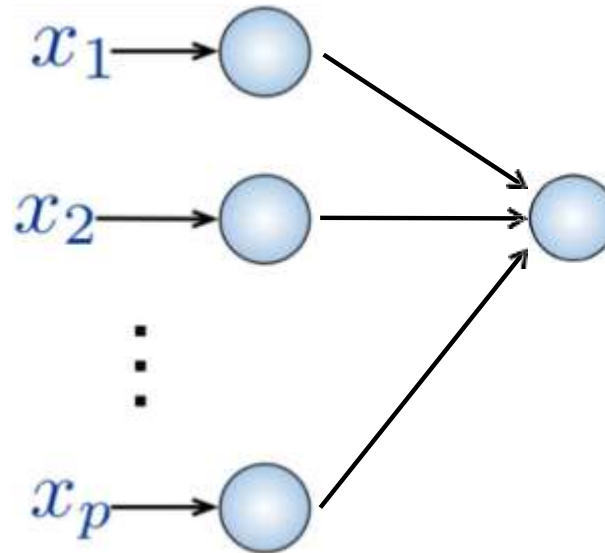
This “patchy” operation is **convolution**

- 1) Apply a set of weights - a filter - to extract **local features**
- 2) Use **multiple filters** to extract different features
- 3) **Spatially share** parameters of each filter

Fully Connected Neural Network

Input:

- 2D image
- Vector of pixel values



Fully Connected:

- Each neuron in hidden layer connected to all neurons in input layer
- No spatial information
- Many, many parameters

Key idea: Use spatial structure in input to inform architecture of the network

High Level Feature Detection

Let's identify key features in each image category



Nose, Eyes, Mouth

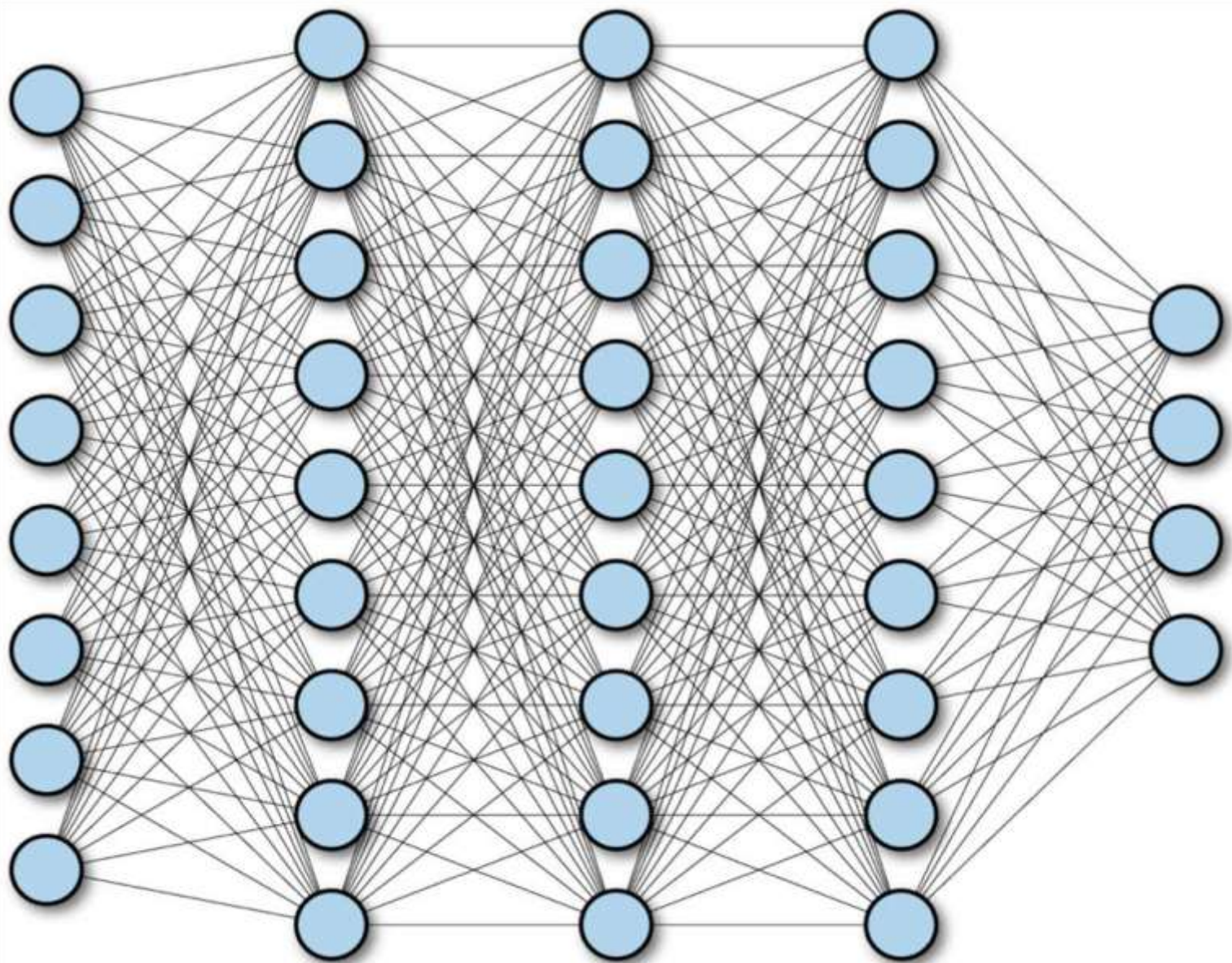


Wheels, License Plate,
Headlights



Door, Windows, Steps

Fully Connected Neural Network



2b. Convolutions and filters

Convolution operation is element wise multiply and add

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

Filter / Kernel

| | | | | |
|-----------------|-----------------|-----------------|---|---|
| 1 _{x1} | 1 _{x0} | 1 _{x1} | 0 | 0 |
| 0 _{x0} | 1 _{x1} | 1 _{x0} | 1 | 0 |
| 0 _{x1} | 0 _{x0} | 1 _{x1} | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

Image

| | | |
|---|--|--|
| 4 | | |
| | | |
| | | |

Convolved
Feature

Producing Feature Maps



Original



Sharpen



Edge Detect



“Strong” Edge
Detect

A simple pattern: Edges

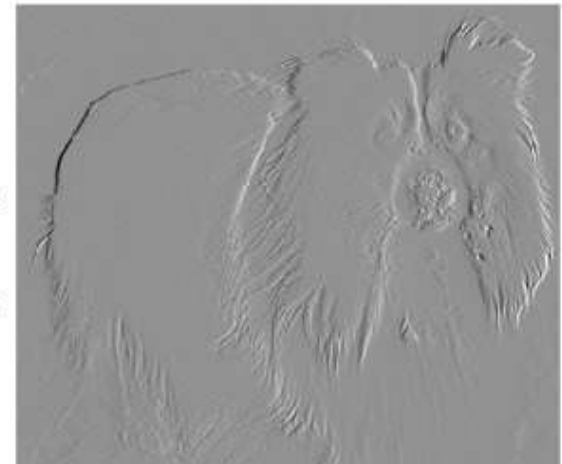
How can we detect edges with a kernel?



Input



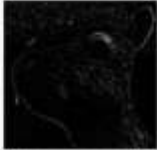




| | |
|---|----|
| 1 | -1 |
|---|----|

Filter



Output

Simple Kernels / Filters

| Operation | Filter | Convolved Image |
|----------------------------------|--|---|
| Identity | $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |  |
| Edge detection | $\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$ |  |
| | $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ |  |
| | $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ |  |
| Sharpen | $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ |  |
| Box blur (normalized) | $\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ |  |
| Gaussian blur (approximation) | $\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ |  |

X or X?

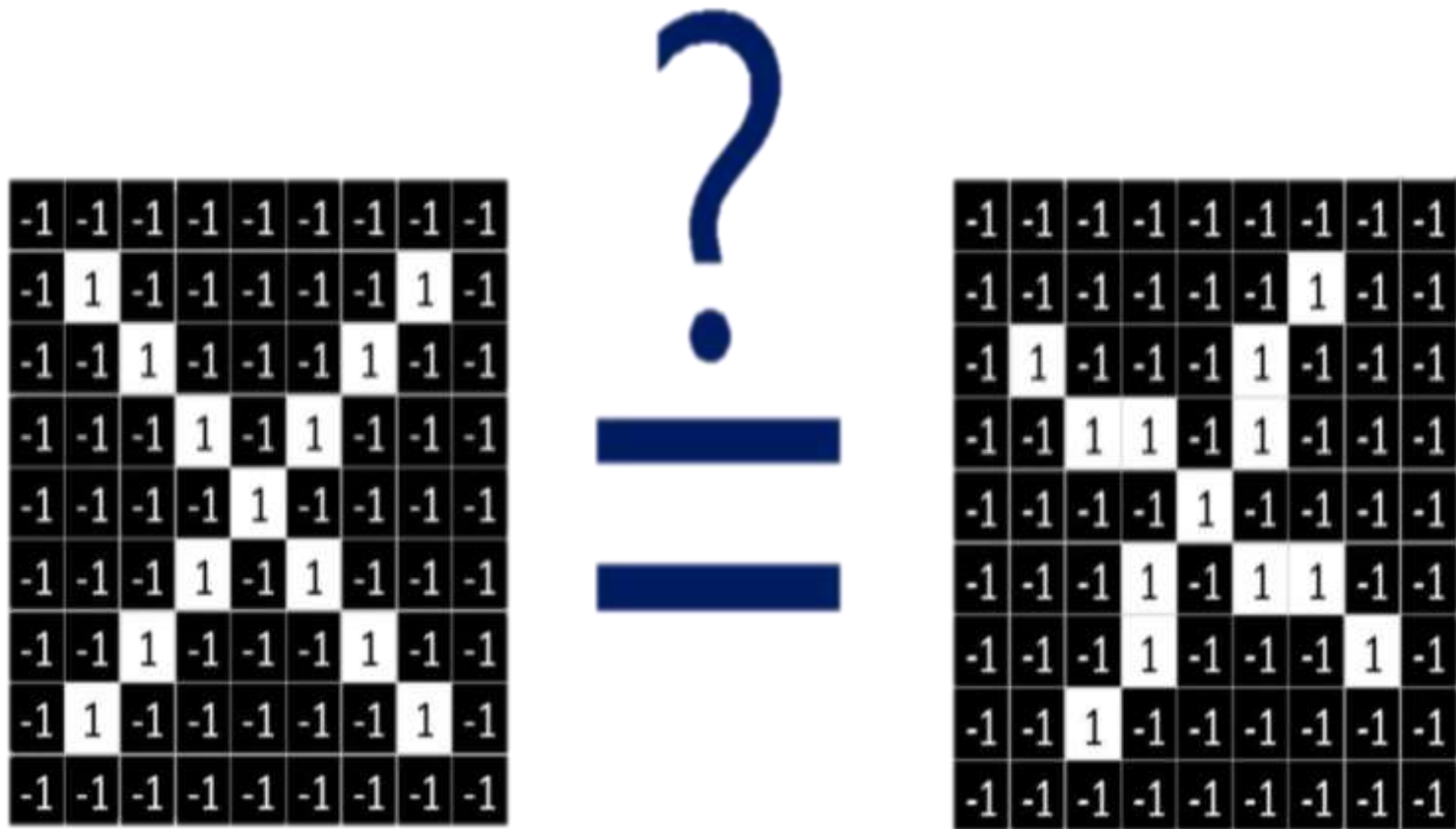


Image is represented as matrix of pixel values... and computers are literal!

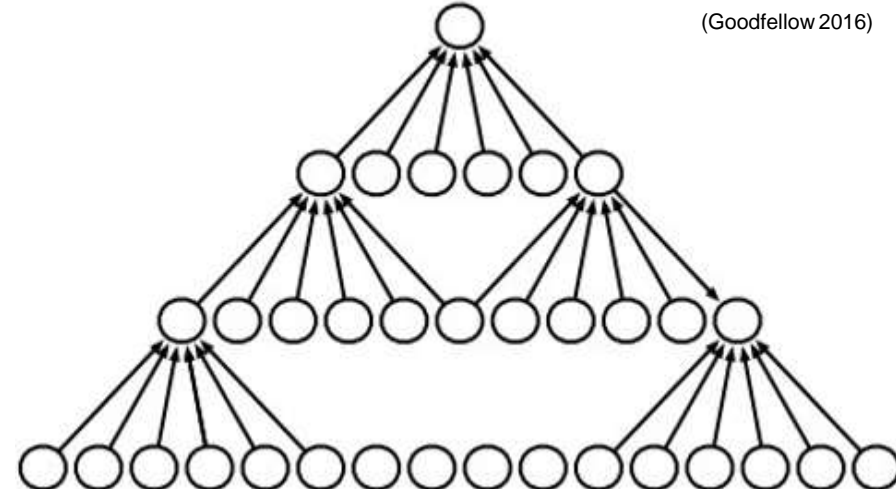
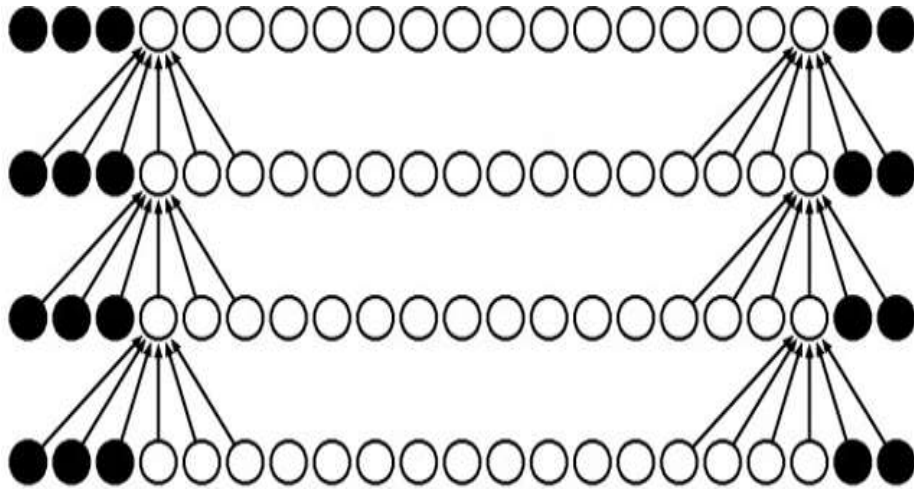
We want to be able to classify an X as an X even if it's shifted, shrunk, rotated, deformed.

There are three approaches to edge cases in convolution

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

Zero Padding Controls Output Size

(Goodfellow 2016)



- **Same convolution:** zero pad input so output is same size as input dimensions
- **Full convolution:** zero pad input so output is produced whenever an output value contains at least one input value (expands output)
- **Valid-only convolution:** output only when entire kernel contained in input (shrinks output)

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

```
x = tf.nn.conv2d(x, W, strides=[1, strides, strides, 1], padding='SAME')
```



- TF convolution operator takes stride and zero fill option as parameters
- Stride is distance between kernel applications in each dimension
- Padding can be SAME or VALID

Today: Convolutional Neural Networks (CNNs)

1. Scene understanding and object recognition for machines (and humans)

- Scene/object recognition challenge. Illusions reveal primitives, conflicting info
- Human neurons/circuits. Visual cortex layers==abstraction. General cognition

2. Classical machine vision foundations: features, scenes, filters, convolution

- Spatial structure primitives: edge detectors & other filters, feature recognition
- Convolution: basics, padding, stride, object recognition, architectures

3. CNN foundations: LeNet, *de novo* feature learning, parameter sharing

- Key ideas: *learn* features, hierarchy, re-use parameters, back-prop filter learning
- CNN formalization: representations(Conv+ReLU+Pool)*N layers + Fully-connected

4. Modern CNN architectures: millions of parameters, dozens of layers

- Feature invariance is hard: apply perturbations, learn for each variation
- ImageNet progression of best performers
- AlexNet: First top performer CNN, 60M parameters (from 60k in LeNet-5), ReLU
- VGGNet: simpler but deeper (8→19 layers), 140M parameters, ensembles
- GoogleNet: new primitive=inception module, 5M params, no FC, efficiency
- ResNet: 152 layers, vanishing gradients → fit residuals to enable learning

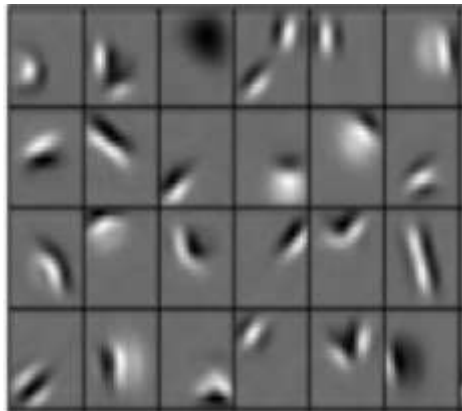
5. Countless applications: General architecture, enormous power

- Semantic segmentation, facial detection/recognition, self-driving, image colorization, optimizing pictures/scenes, up-scaling, medicine, biology, genomics

3a. Learning Visual Features *de novo*

Key idea:
learn hierarchy of features
directly from the data
(rather than hand-engineering them)

Low level features



Edges, dark spots

Mid level features



Eyes, ears, nose

High level features

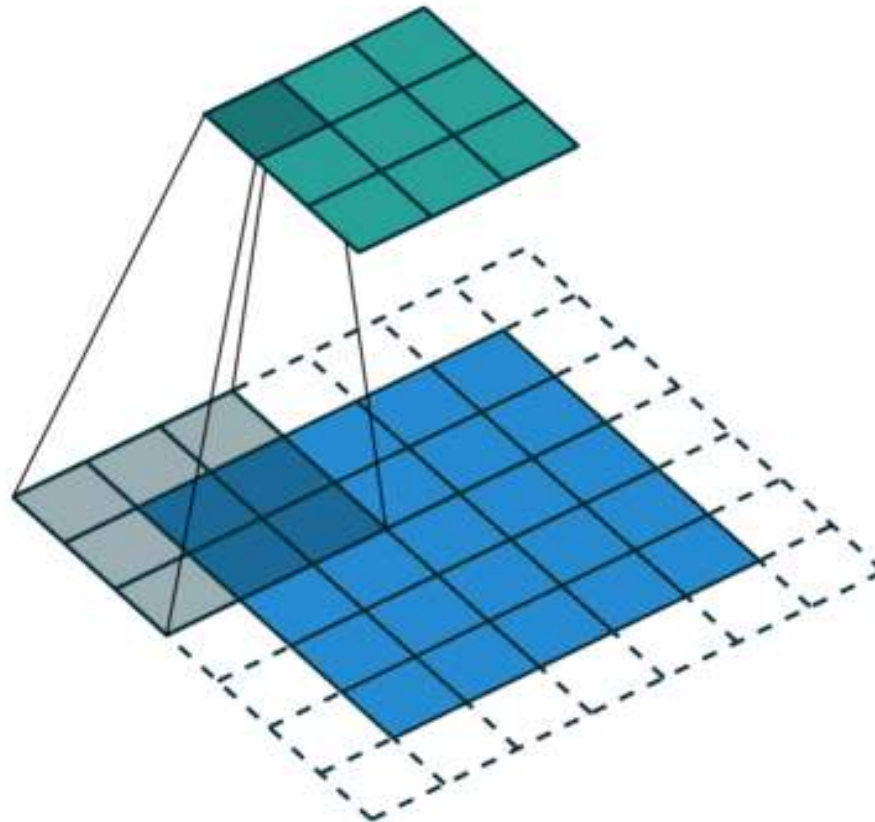


Facial structure

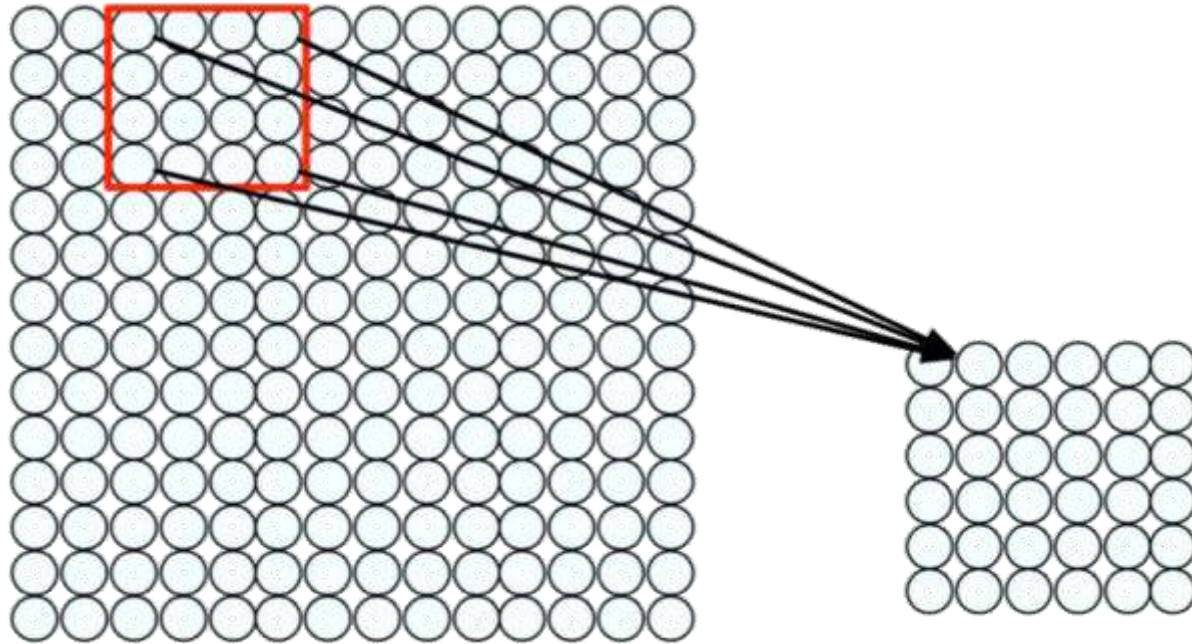
Key idea: re-use parameters

Convolution shares parameters

Example 3x3 convolution on a 5x5 image



Feature Extraction with Convolution



- 1) Apply a set of weights - a filter - to extract **local features**
- 2) Use **multiple filters** to extract different features
- 3) **Spatially share** parameters of each filter

LeNet-5

- *Gradient Based Learning Applied To Document Recognition* - Y. Lecun, L. Bottou, Y. Bengio, P. Haffner; 1998
- Helped establish how we use CNNs today
- Replaced manual feature extraction

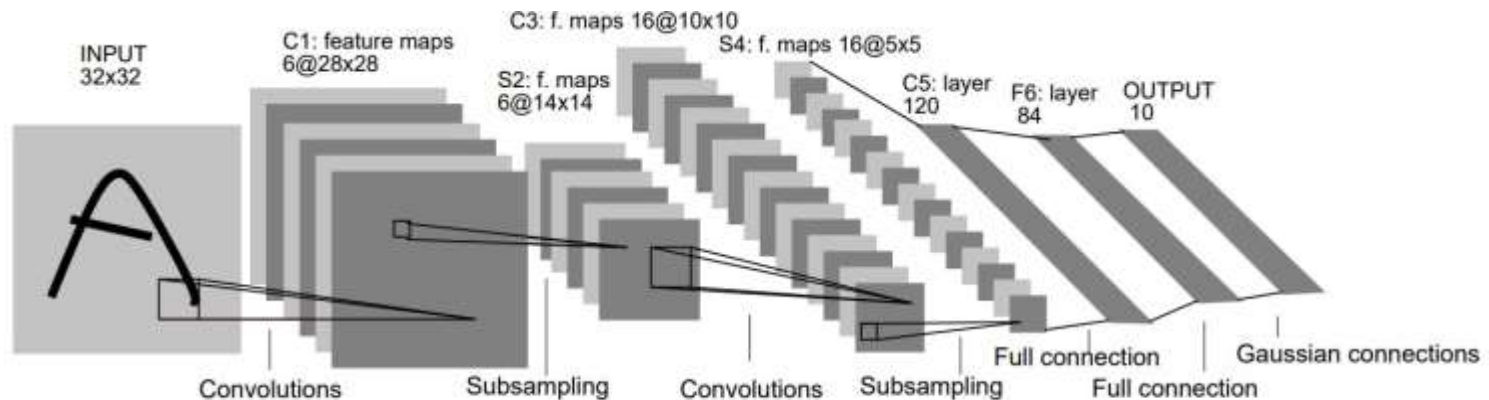
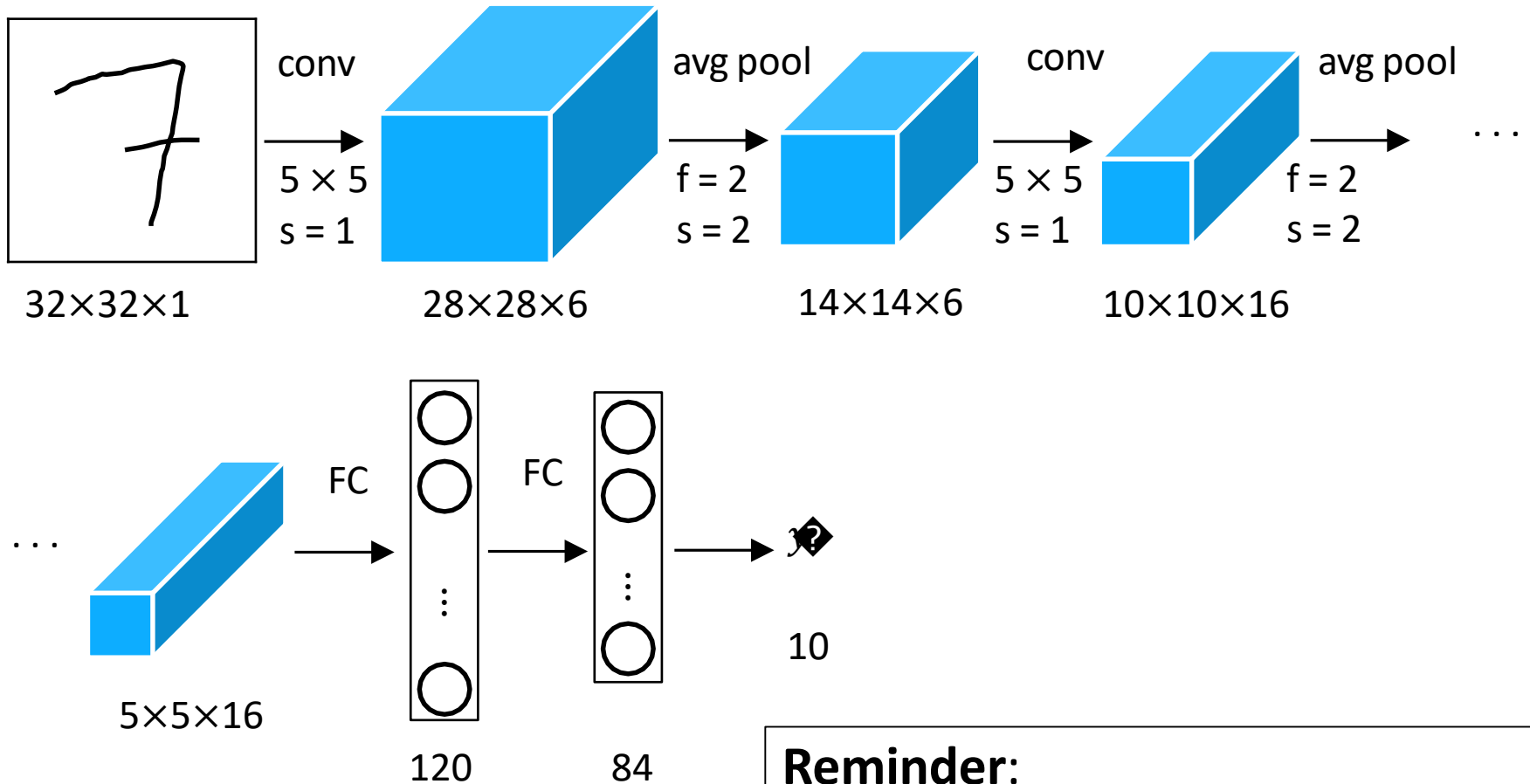


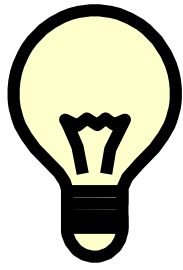
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet-5



Reminder:

$$\text{Output size} = (N + 2P - F) / \text{stride} + 1$$



LeNet-5

- Only 60K parameters
 - As we go deeper in the network: $N_H \downarrow$, $N_W \downarrow$, $N_C \uparrow$
 - General structure:
conv->pool->conv->pool->FC->FC->output
-

- Different filters look at different channels
- Sigmoid and Tanh nonlinearity

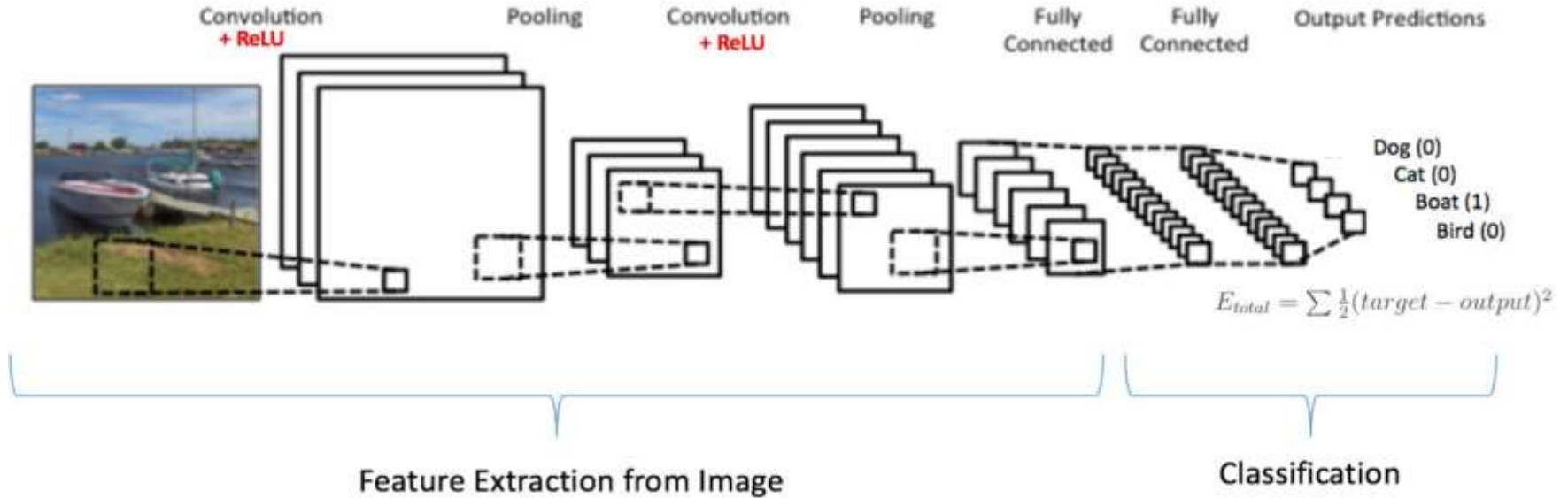
Backpropagation of convolution

$$\begin{array}{|c|c|} \hline O_{11} & O_{12} \\ \hline O_{21} & O_{22} \\ \hline \end{array} = \text{Convolution} \left(\begin{array}{|c|c|c|} \hline X_{11} & X_{12} & X_{13} \\ \hline X_{21} & X_{22} & X_{23} \\ \hline X_{31} & X_{32} & X_{33} \\ \hline \end{array}, \begin{array}{|c|c|} \hline F_{11} & F_{12} \\ \hline F_{21} & F_{22} \\ \hline \end{array} \right)$$

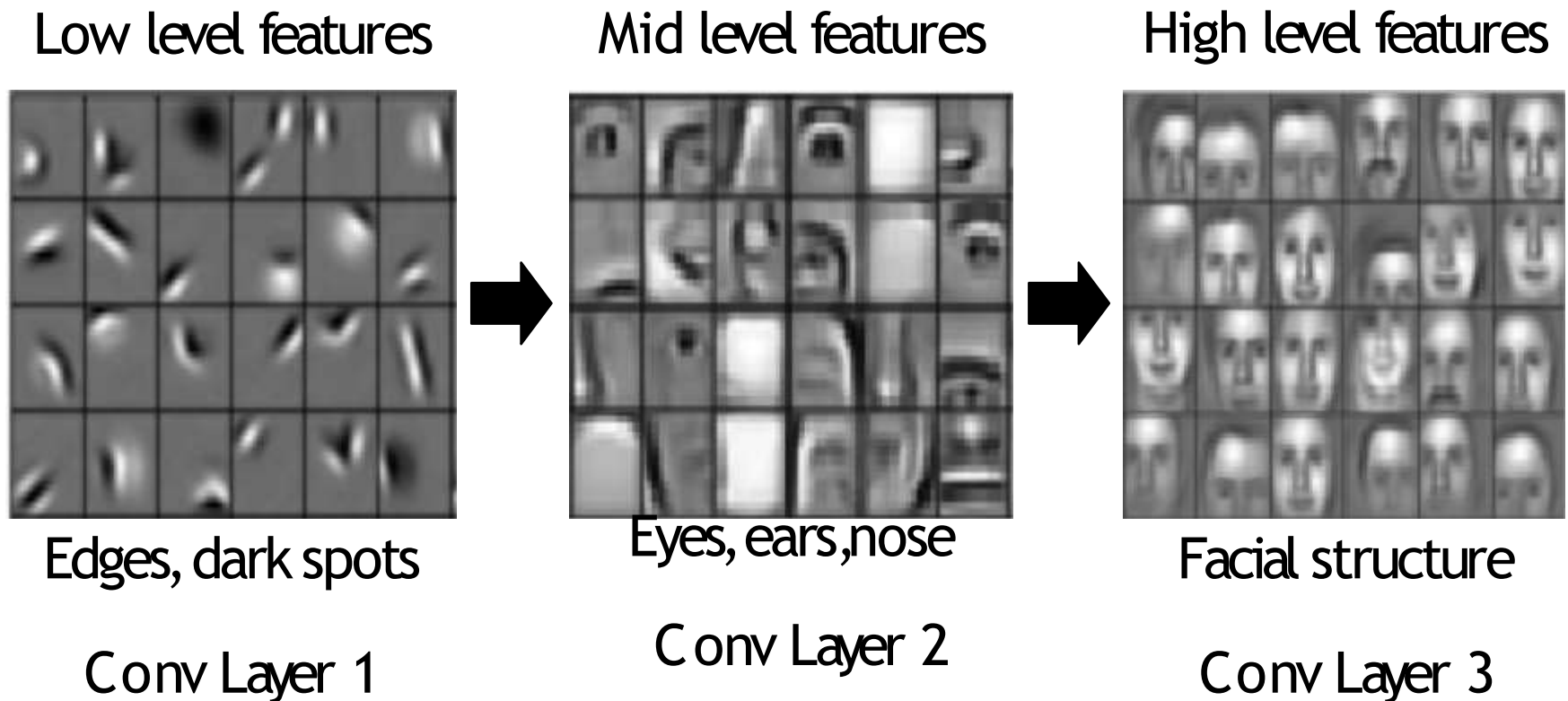
$$\begin{array}{|c|c|} \hline \partial E / \partial F_{11} & \partial E / \partial F_{12} \\ \hline \partial E / \partial F_{21} & \partial E / \partial F_{22} \\ \hline \end{array} = \text{Convolution} \left(\begin{array}{|c|c|c|} \hline X_{11} & X_{12} & X_{13} \\ \hline X_{21} & X_{22} & X_{23} \\ \hline X_{31} & X_{32} & X_{33} \\ \hline \end{array}, \begin{array}{|c|c|} \hline \partial E / \partial O_{11} & \partial E / \partial O_{12} \\ \hline \partial E / \partial O_{21} & \partial E / \partial O_{22} \\ \hline \end{array} \right)$$

3b. Convolutional Neural Networks (CNNs)

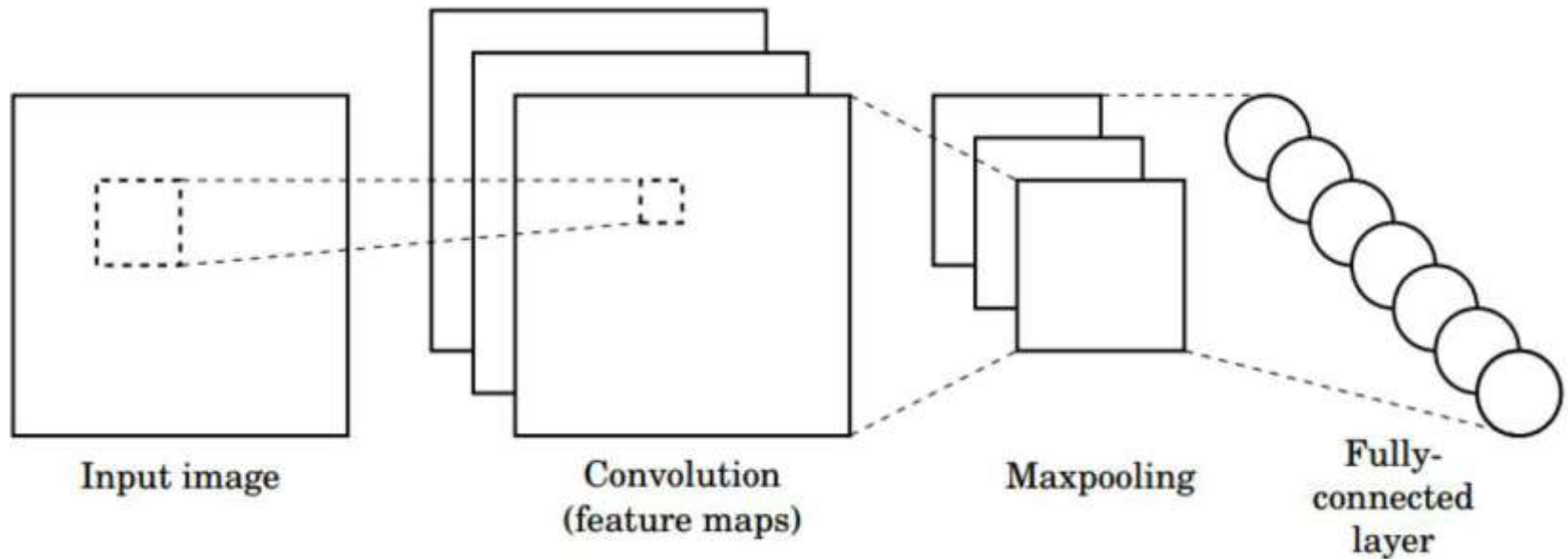
An image classification CNN



Representation Learning in Deep CNNs



CNNs for Classification



1. **Convolution:** Apply filters to generate feature maps.
2. **Non-linearity:** Often ReLU.
3. **Pooling:** Downsampling operation on each feature map.

Train model with image data.

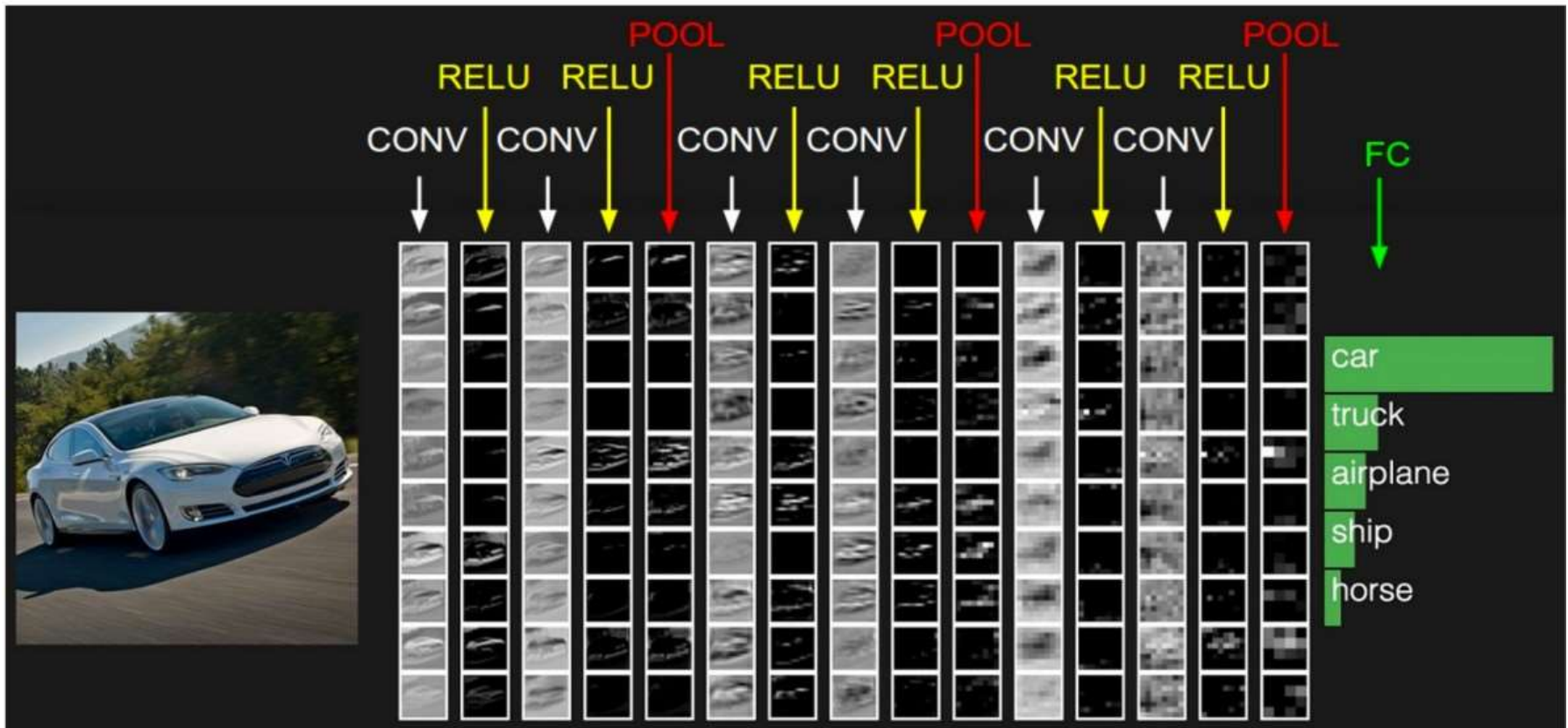
Learn weights of filters in convolutional layers.

```
tf.keras.layers.Conv2D
```

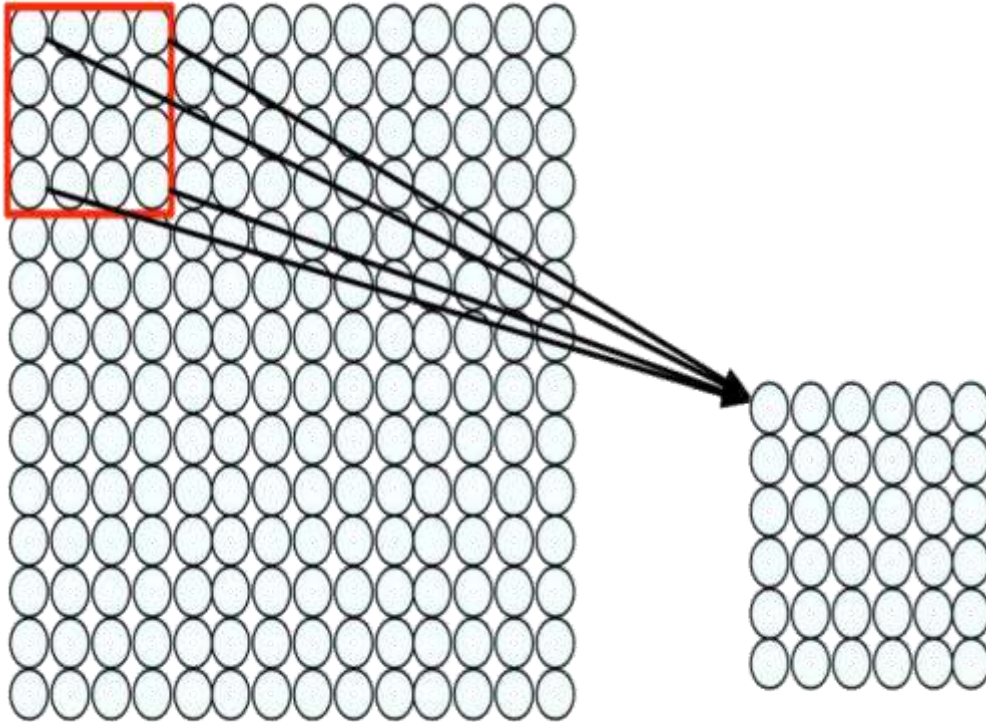
```
tf.keras.activations.*
```


```
tf.keras.layers.MaxPool2D
```

Example – Six convolutional layers



Convolutional Layers: Local Connectivity



 `tf.keras.layers.
Conv2D`

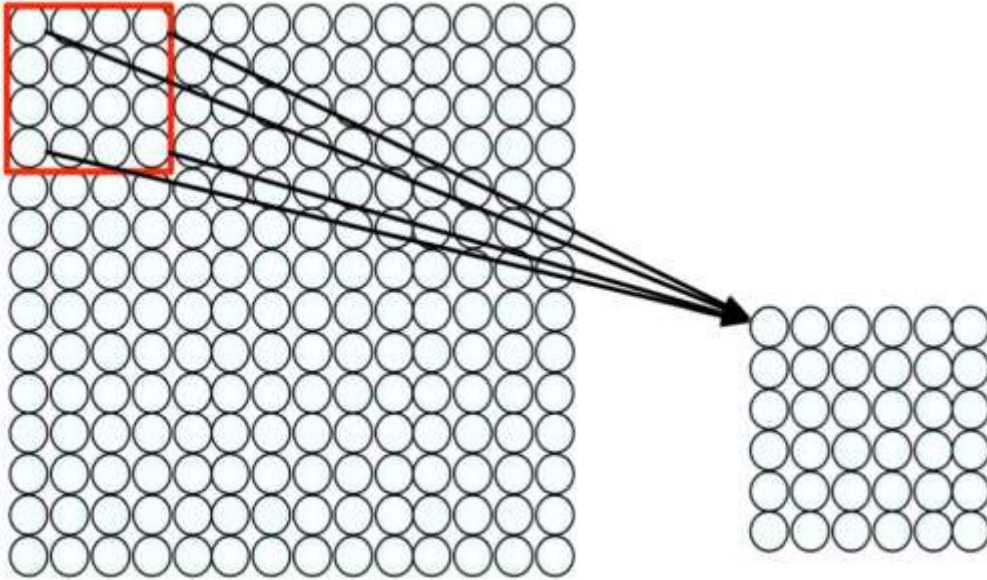
For a neuron in
hidden layer:

- Take inputs from patch
- Compute weighted sum
- Apply bias

Convolutional Layers: Local Connectivity



`tf.keras.layers.Conv2D`



For a neuron in hidden layer:

- Take inputs from patch
- Compute weighted sum
- Apply bias

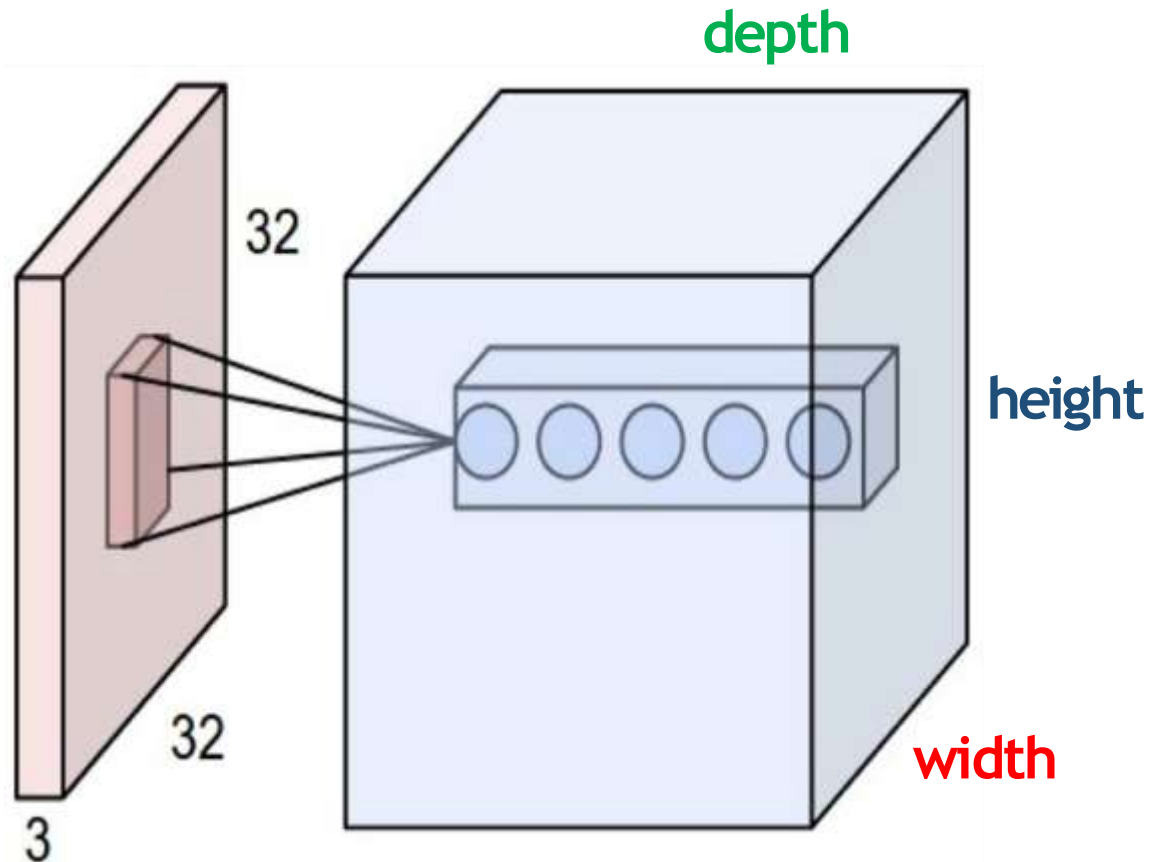
4x4 filter:
matrix of
weights w_{ij}

$$\sum_{i=1}^4 \sum_{j=1}^4 w_{ij} x_{i+p, j+q} + b$$

for neuron (p,q) in hidden layer

- 1) applying a window of weights
- 2) computing linear combinations
- 3) activating with non-linear function

CNNs: Spatial Arrangement of Output Volume



Layer Dimensions:

$$h \cdot w \cdot d$$

where h and w are spatial dimensions
 d (depth) = number of filters

Stride:

Filter step size

Receptive Field:

Locations in input image that a node is path connected to

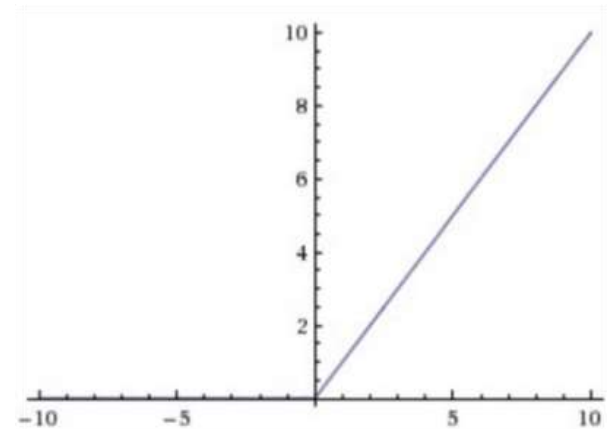
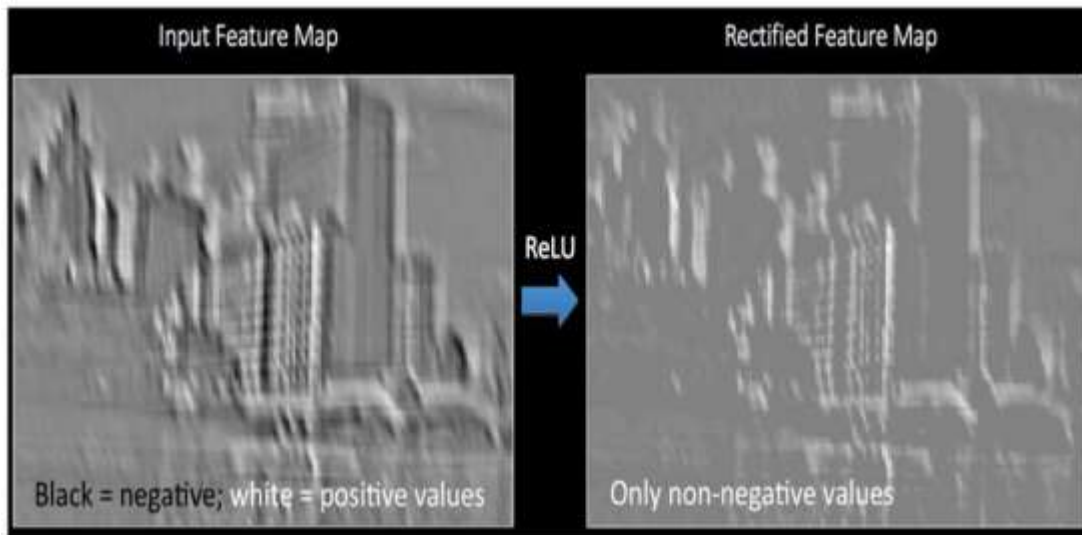


```
tf.keras.layers.Conv2D( filters=d, kernel_size=(h,w), strides=s )
```

Introducing Non-Linearity

- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero.
- **Non-linear operation**

Rectified Linear Unit (ReLU)

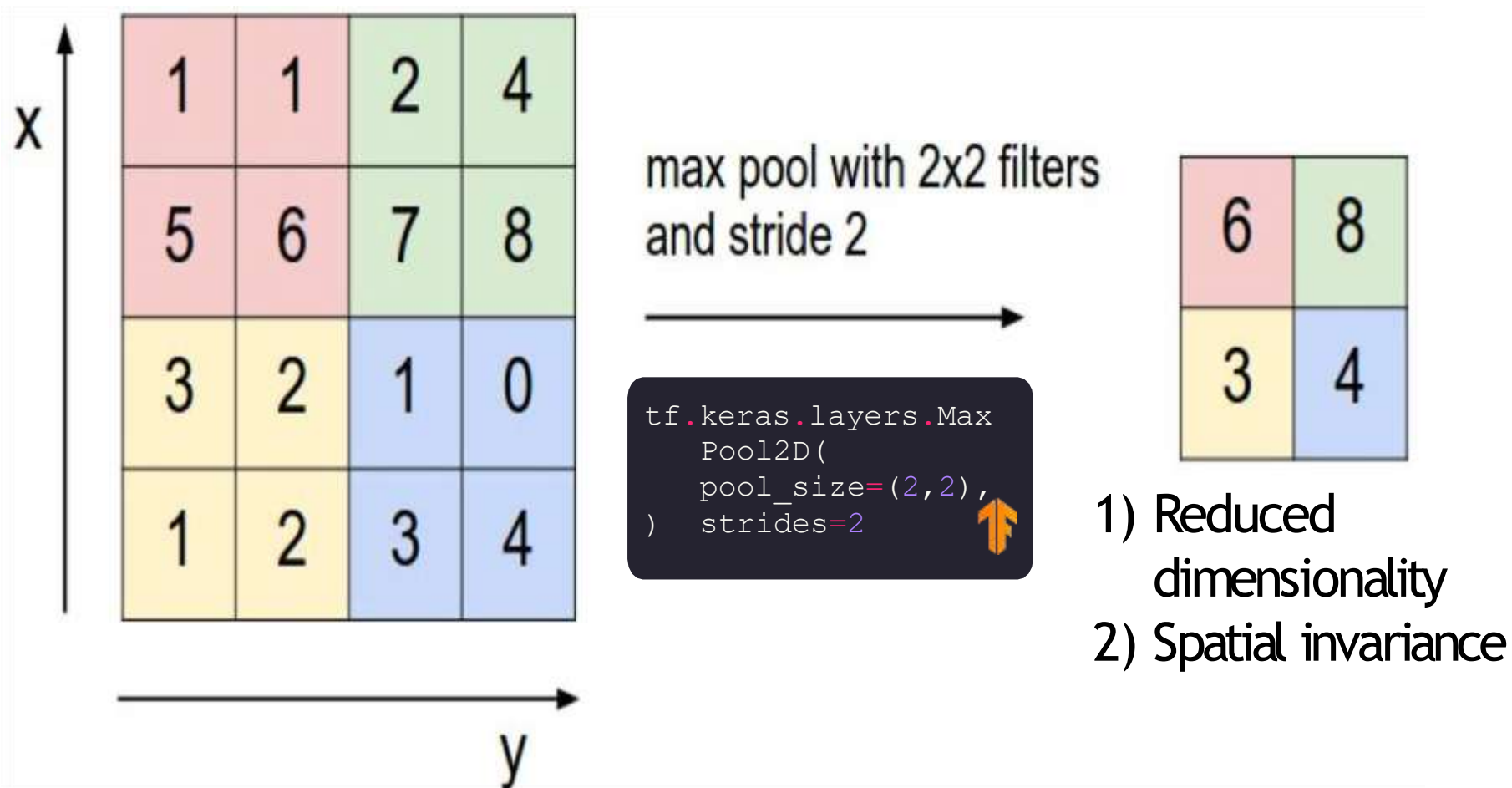


$$g(z) = \max(0, z)$$



`tf.keras.layers.ReLU`

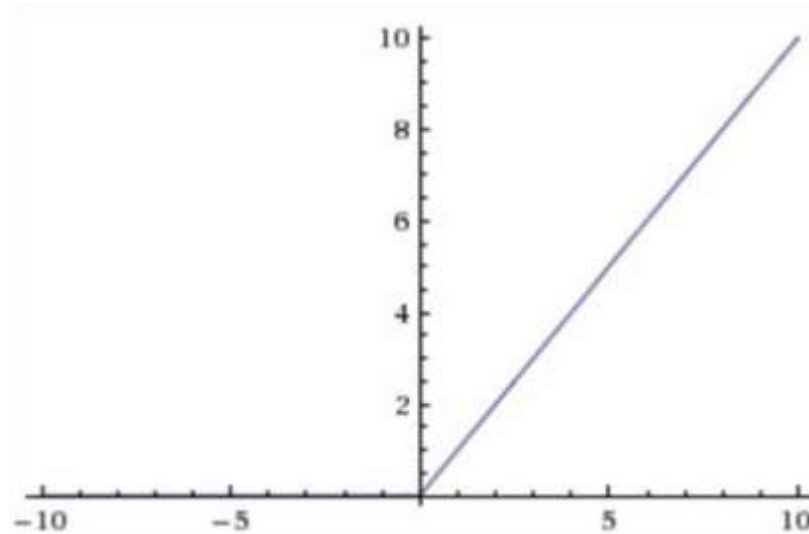
Pooling



Max Pooling, average pooling

The REctified Linear Unit (RELU) is a common non-linear **detector** stage after convolution

```
x = tf.nn.conv2d(x, W, strides=[1, strides, strides, 1], padding='SAME')  
x = tf.nn.bias_add(x, b)  
x = tf.nn.relu(x)
```



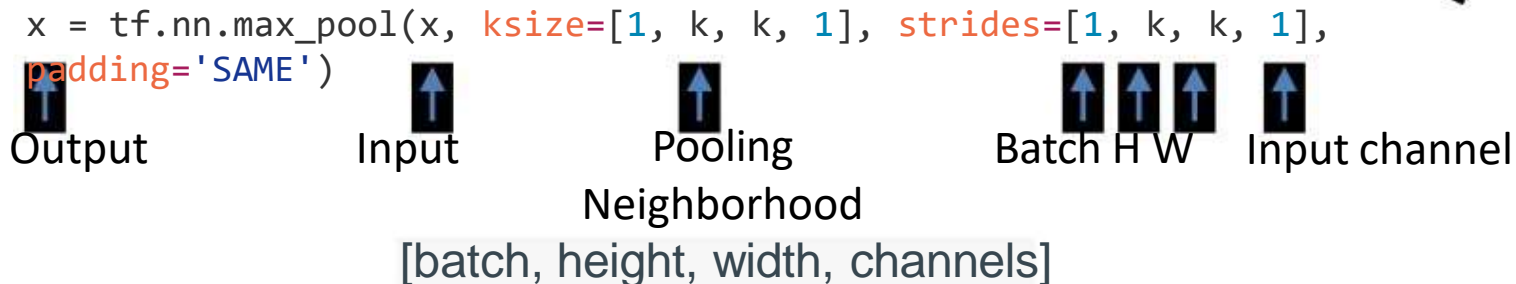
$$f(x) = \max(0, x)$$

When will we backpropagate through this?

Once it “dies” what happens to it?

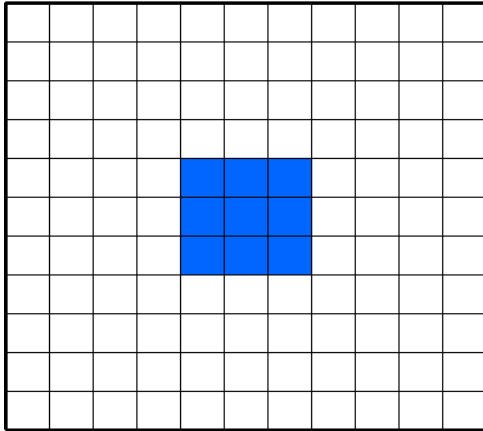
Pooling reduces dimensionality by giving up spatial location

- **max pooling** reports the maximum output within a defined neighborhood
- Padding can be SAME or VALID

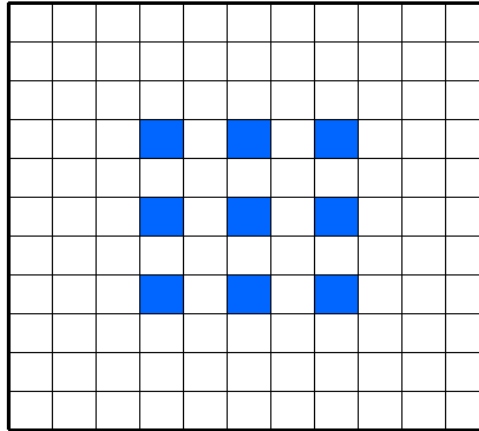


Dilated Convolution

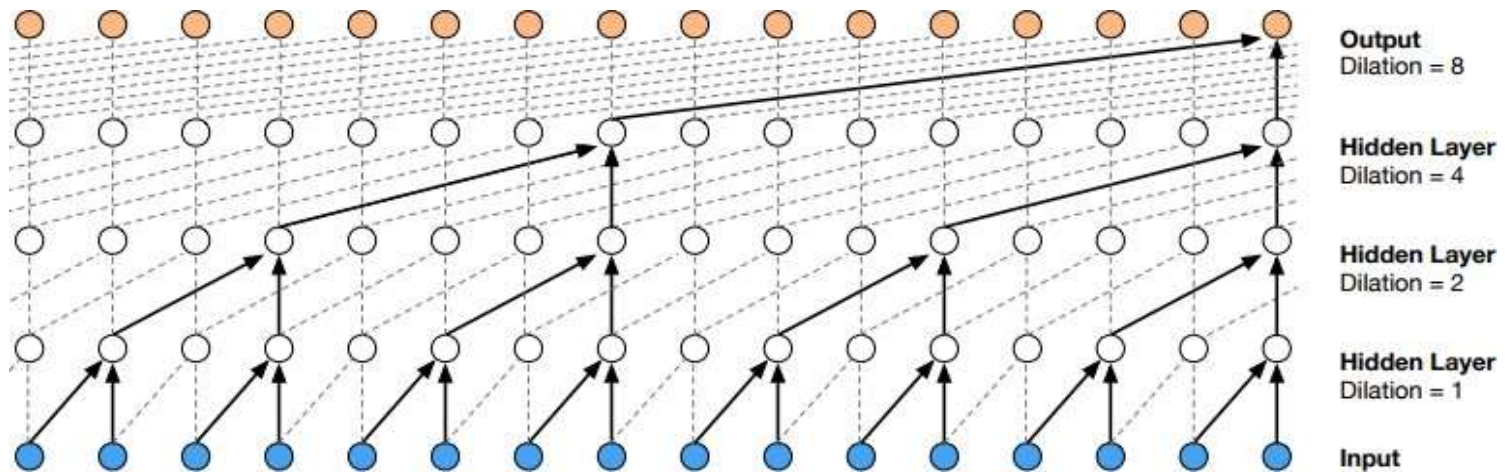
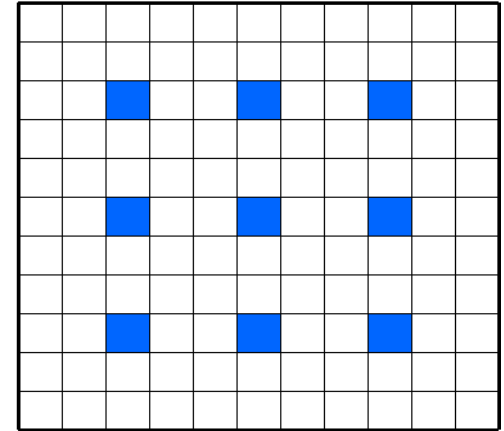
D = 1



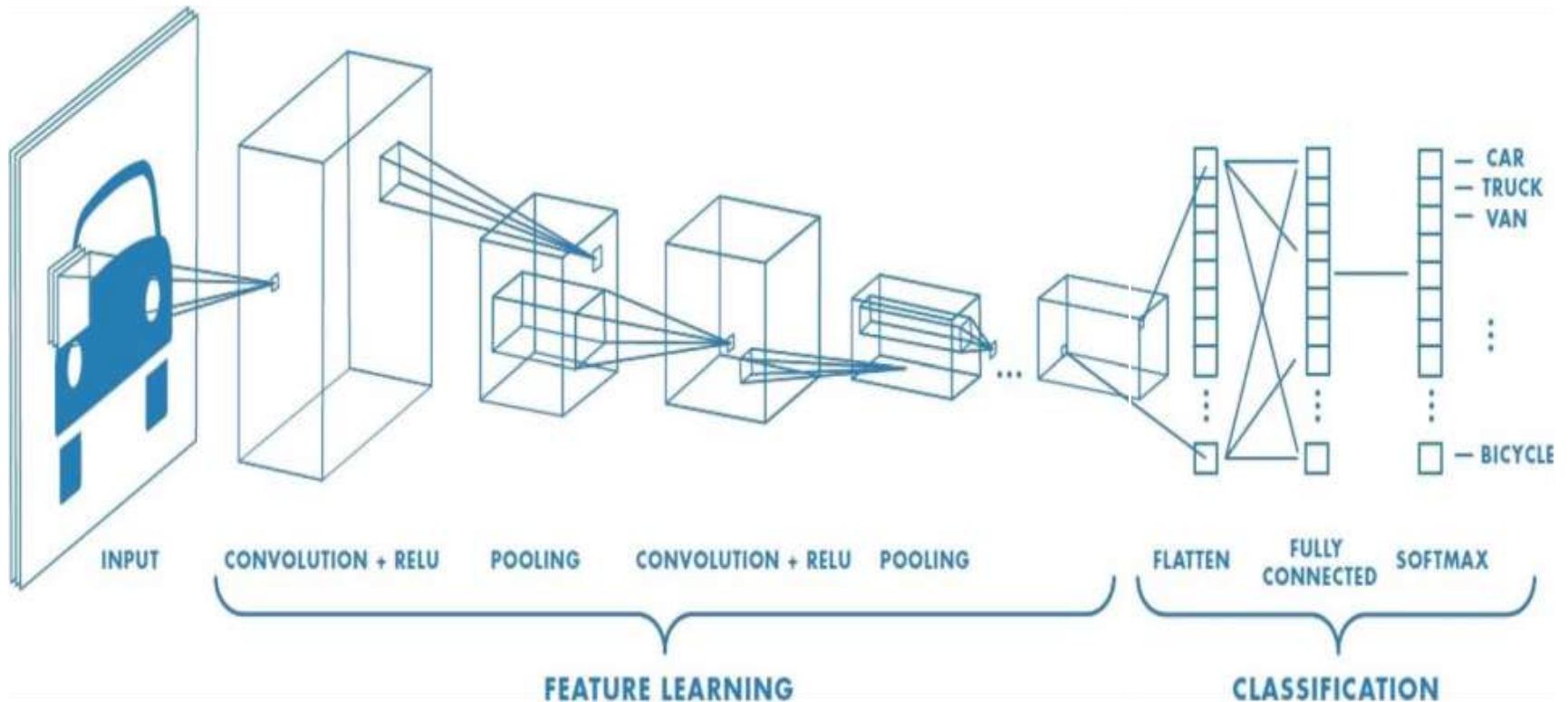
D = 2



D = 3

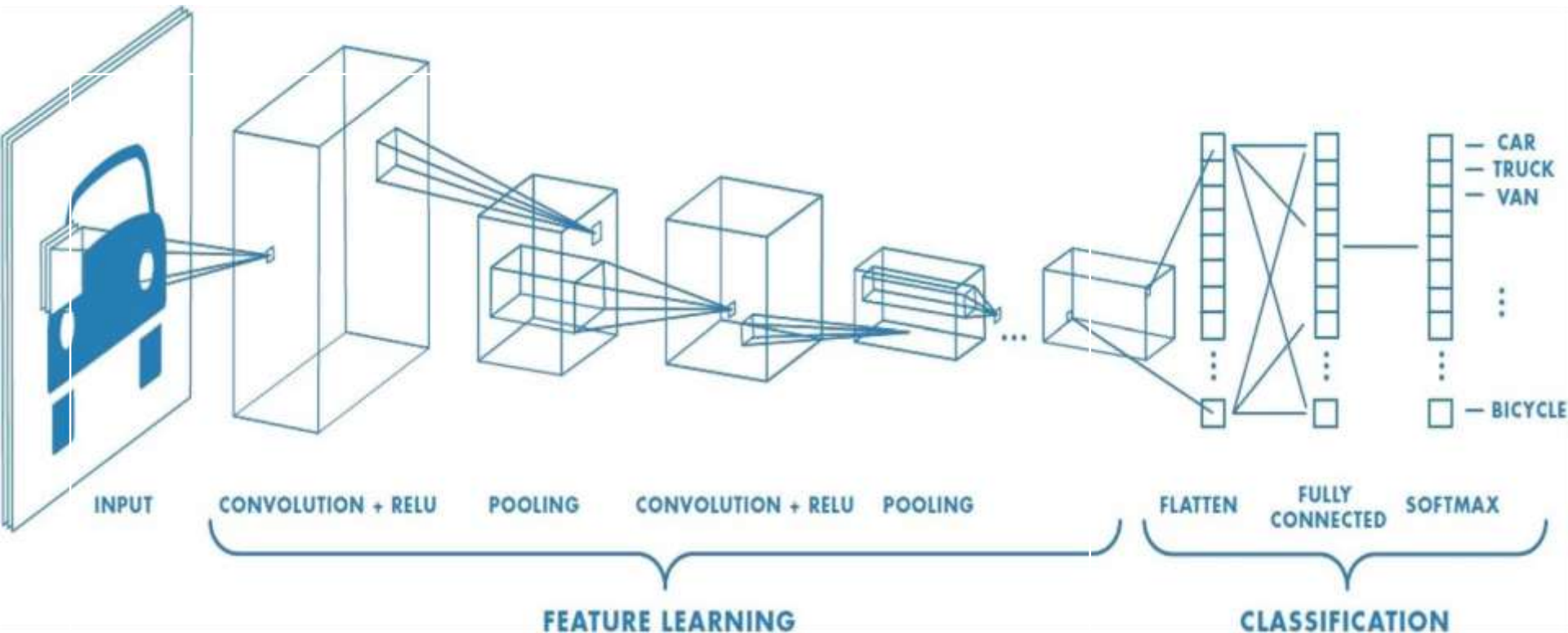


CNNs for Classification: Feature Learning



1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

CNNs for Classification: Class Probabilities



$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

Putting it all together

```
import tensorflow as tf
```

```
def generate_model():
```

```
    model = tf.keras.Sequential([
```

```
        # first convolutional layer
```

```
        tf.keras.layers.Conv2D(32, filter_size=3, activation='relu'),
```

```
        tf.keras.layers.MaxPool2D(pool_size=2, strides=2),
```

```
        # second convolutional layer
```

```
        tf.keras.layers.Conv2D(64, filter_size=3, activation='relu'),
```

```
        tf.keras.layers.MaxPool2D(pool_size=2, strides=2),
```

```
        # fully connected classifier
```

```
        tf.keras.layers.Flatten(),
```

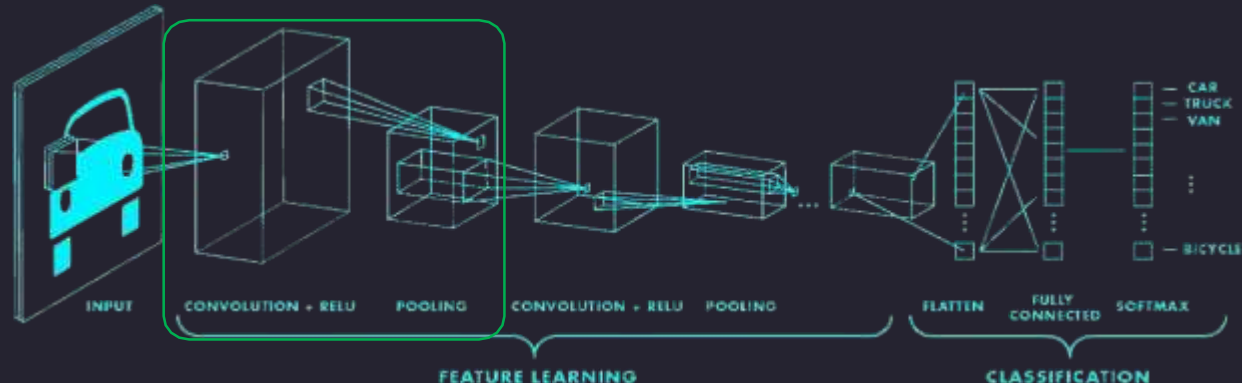
```
        tf.keras.layers.Dense(1024, activation='relu'),
```

```
        tf.keras.layers.Dense(10, activation='softmax')
```

```
        # 10 outputs
```

```
    ])
```

```
    return model
```



Today: Convolutional Neural Networks (CNNs)

1. Scene understanding and object recognition for machines (and humans)

- Scene/object recognition challenge. Illusions reveal primitives, conflicting info
- Human neurons/circuits. Visual cortex layers==abstraction. General cognition

2. Classical machine vision foundations: features, scenes, filters, convolution

- Spatial structure primitives: edge detectors & other filters, feature recognition
- Convolution: basics, padding, stride, object recognition, architectures

3. CNN foundations: LeNet, *de novo* feature learning, parameter sharing

- Key ideas: *learn* features, hierarchy, re-use parameters, back-prop filter learning
- CNN formalization: representations(Conv+ReLU+Pool)*N layers + Fully-connected

4. Modern CNN architectures: millions of parameters, dozens of layers

- Feature invariance is hard: apply perturbations, learn for each variation
- ImageNet progression of best performers
- AlexNet: First top performer CNN, 60M parameters (from 60k in LeNet-5), ReLU
- VGGNet: simpler but deeper (8→19 layers), 140M parameters, ensembles
- GoogleNet: new primitive=inception module, 5M params, no FC, efficiency
- ResNet: 152 layers, vanishing gradients → fit residuals to enable learning

5. Countless applications: General architecture, enormous power

- Semantic segmentation, facial detection/recognition, self-driving, image colorization, optimizing pictures/scenes, up-scaling, medicine, biology, genomics

4a. Real-world feature invariance is
hard

How can computers recognize objects?



How can computers recognize objects?



Challenge:

- Objects can be anywhere in the scene, in any orientation, rotation, color hue, etc.
- How can we overcome this challenge?

Answer:

- Learn a ton of features (millions) from the bottom up
- Learn the convolutional filters, rather than pre-computing them

Feature invariance to perturbation is hard

Viewpoint variation



Scale variation



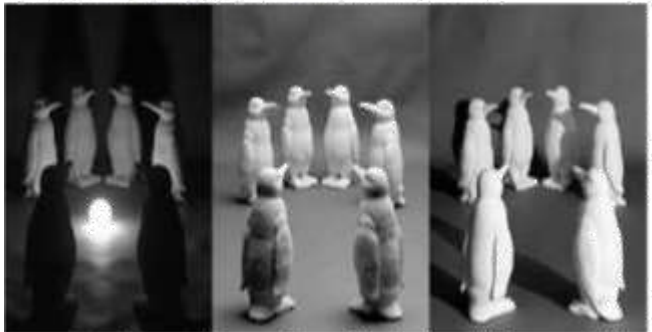
Deformation



Occlusion



Illumination conditions



Background clutter



Intra-class variation



Next-generation models
explode # of parameters

LeNet-5

- *Gradient Based Learning Applied To Document Recognition* - Y. Lecun, L. Bottou, Y. Bengio, P. Haffner; 1998
- Helped establish how we use CNNs today
- Replaced manual feature extraction

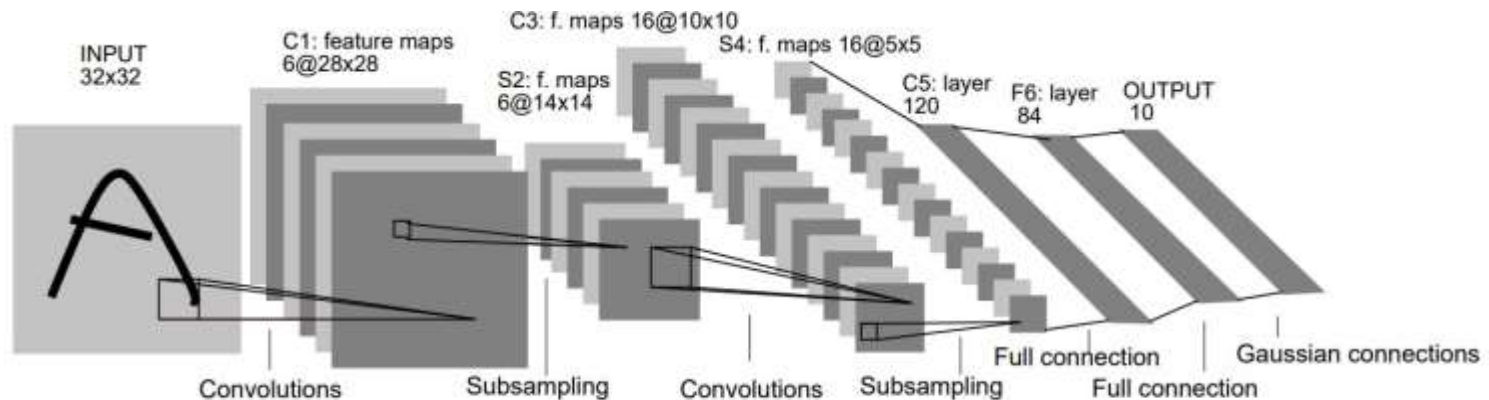
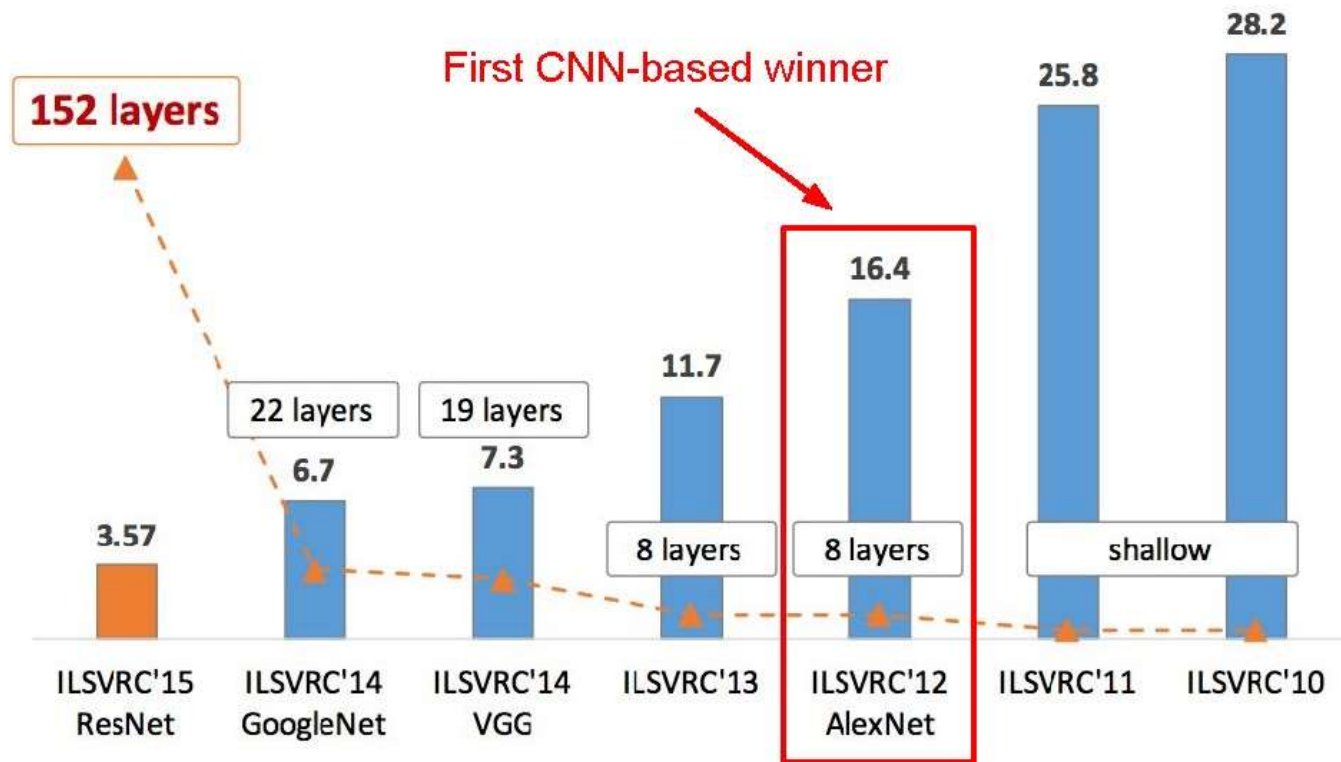


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



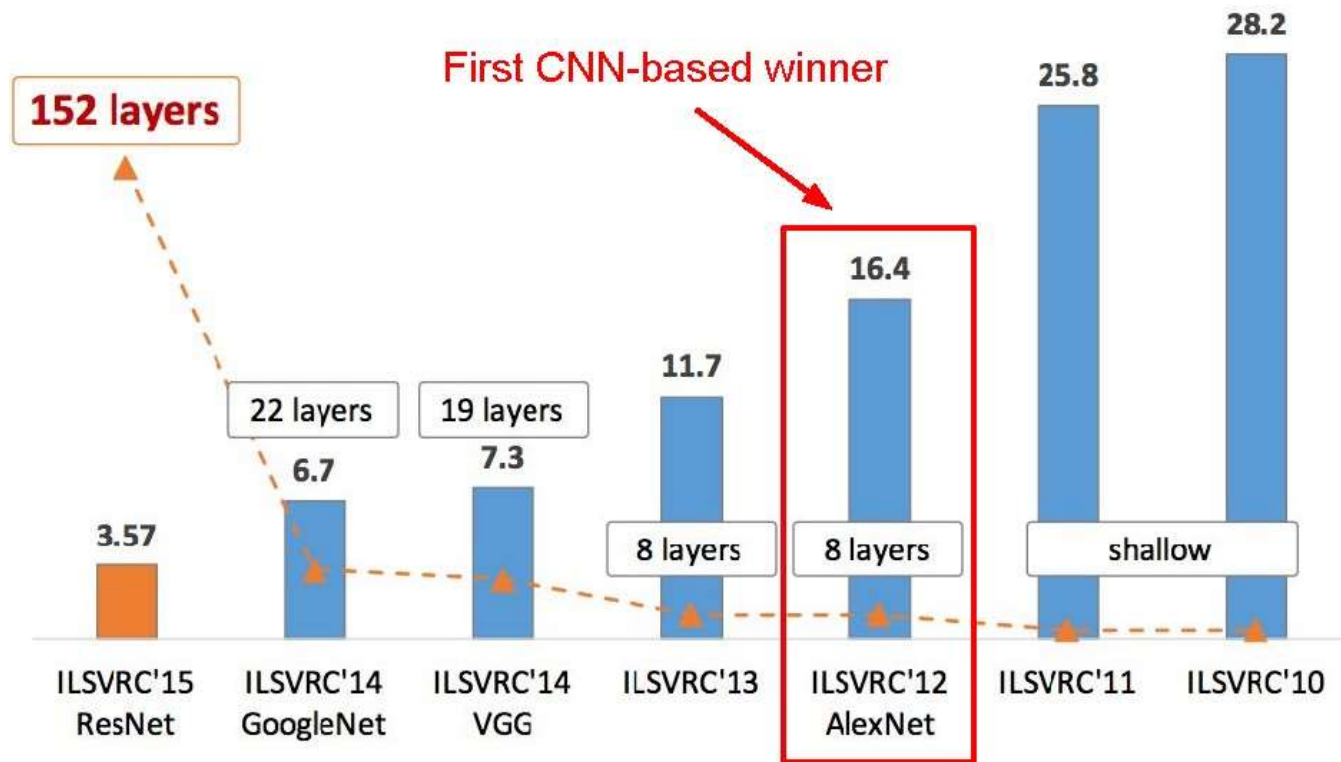
AlexNet

- *ImageNet Classification with Deep Convolutional Neural Networks - Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton; 2012*
- Facilitated by GPUs, highly optimized convolution implementation and large datasets (ImageNet)
- One of the largest CNNs to date
- Has 60 Million parameter compared to 60k parameter of LeNet-5

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- The annual “Olympics” of computer vision.
- Teams from across the world compete to see who has the best computer vision model for tasks such as classification, localization, detection, and more.
- **2012** marked **the first year where a CNN was used** to achieve a top 5 test error rate of 15.3%.
- The next best entry achieved an error of 26.2%.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



AlexNet

Architecture

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

CONV5

Max POOL3

FC6

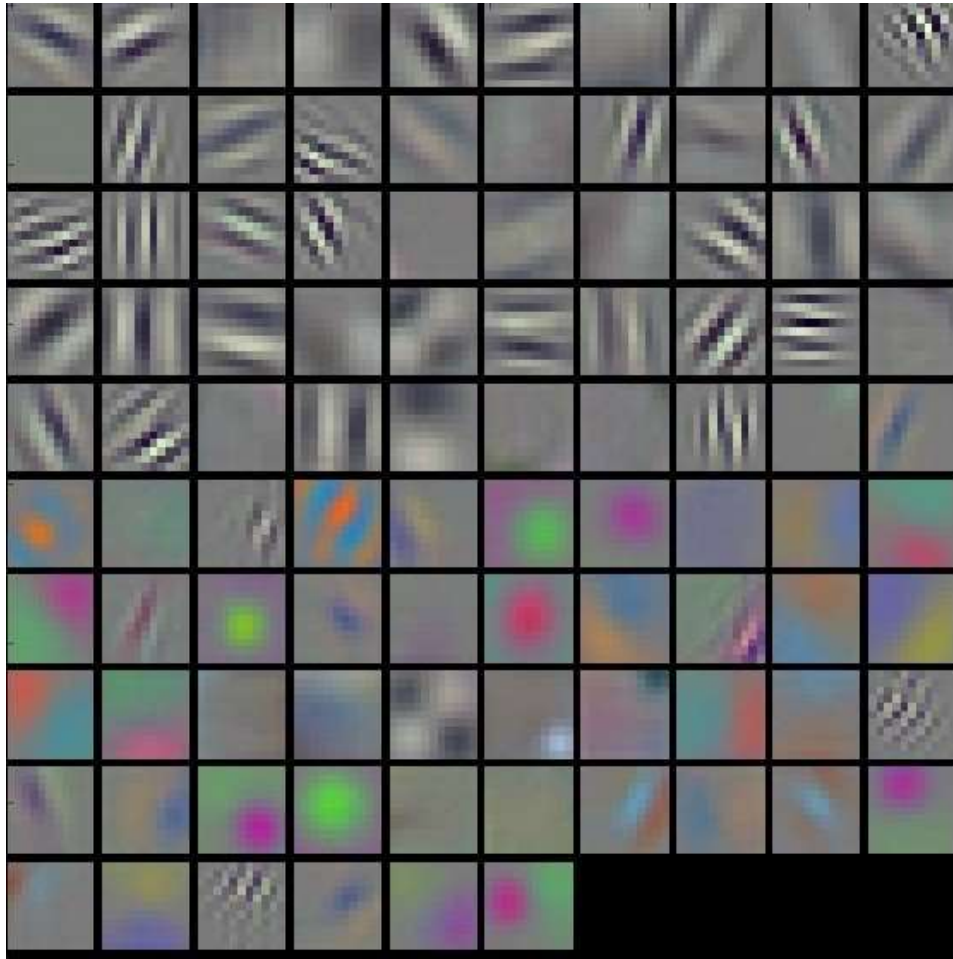
FC7

FC8

- Input: 227x227x3 images (224x224 before padding)
- First layer: 96 11x11 filters applied at stride 4
- **Output volume size?**
$$(N-F)/s+1 = (227-11)/4+1 = 55 \rightarrow$$

[55x55x96]
- **Number of parameters in this layer?**
$$(11*11*3)*96 = 35K$$

AlexNet



AlexNet

Architecture

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

CONV5

Max POOL3

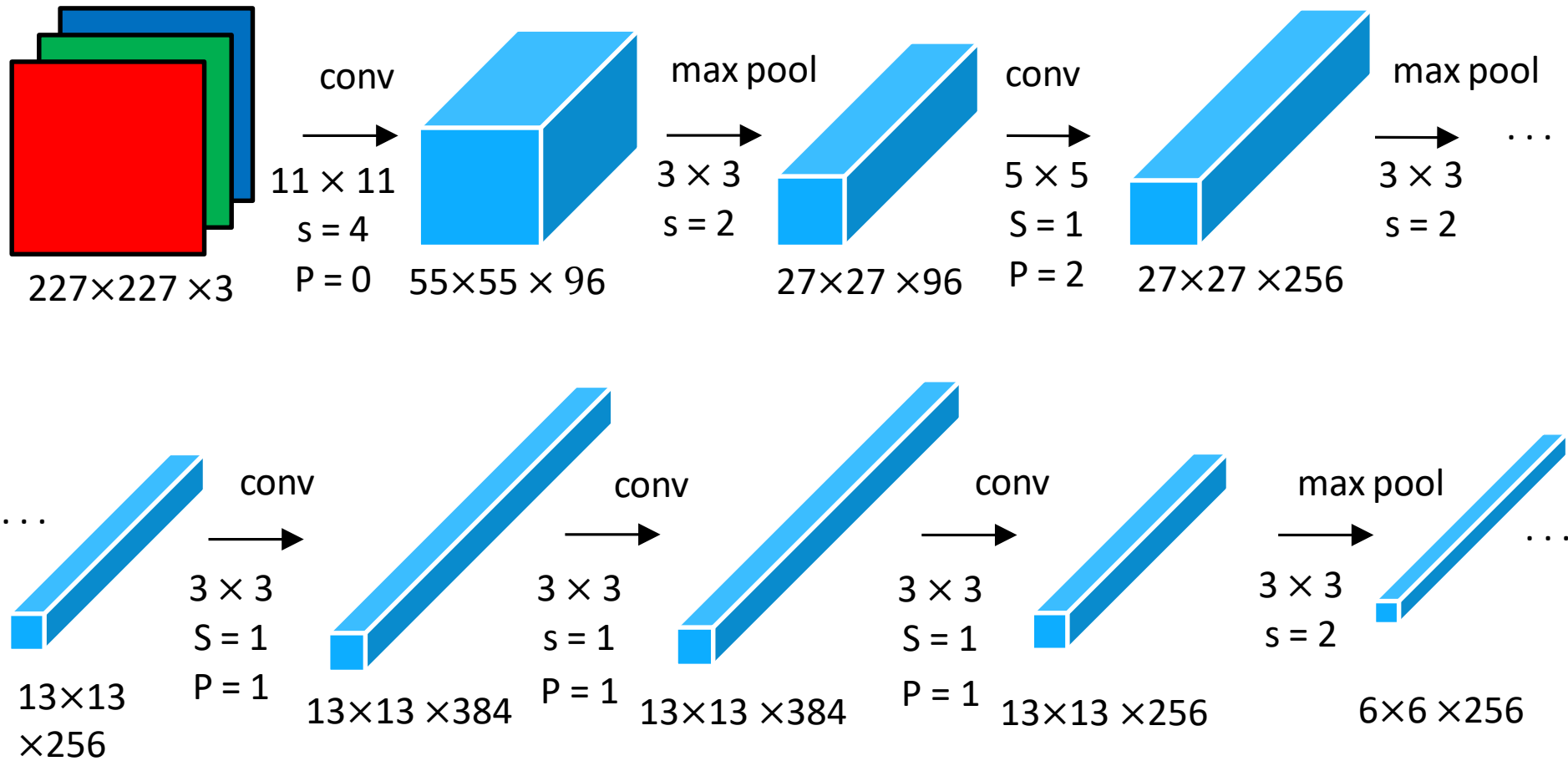
FC6

FC7

FC8

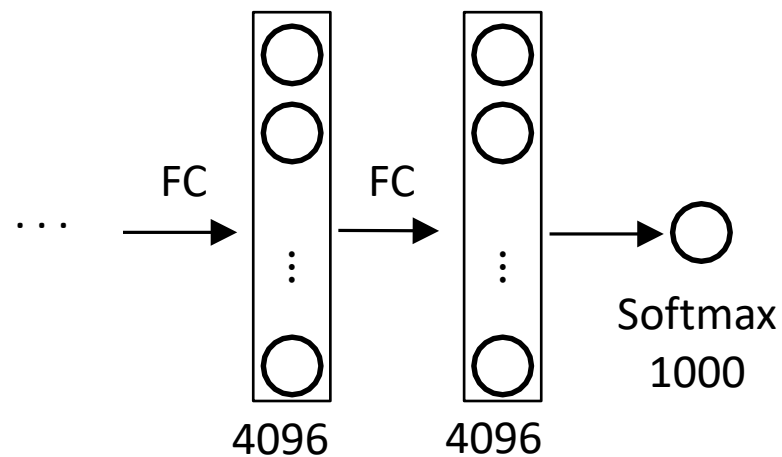
- Input: 227x227x3 images (224x224 before padding)
- After CONV1: 55x55x96
- Second layer: 3x3 filters applied at stride 2
- **Output volume size?**
$$(N-F)/s+1 = (55-3)/2+1 = 27 \rightarrow [27 \times 27 \times 96]$$
- **Number of parameters in this layer?**
0!

AlexNet



[Krizhevsky et al., 2012]

AlexNet



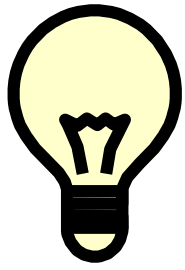
AlexNet

Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- 7 CNN ensemble

AlexNet

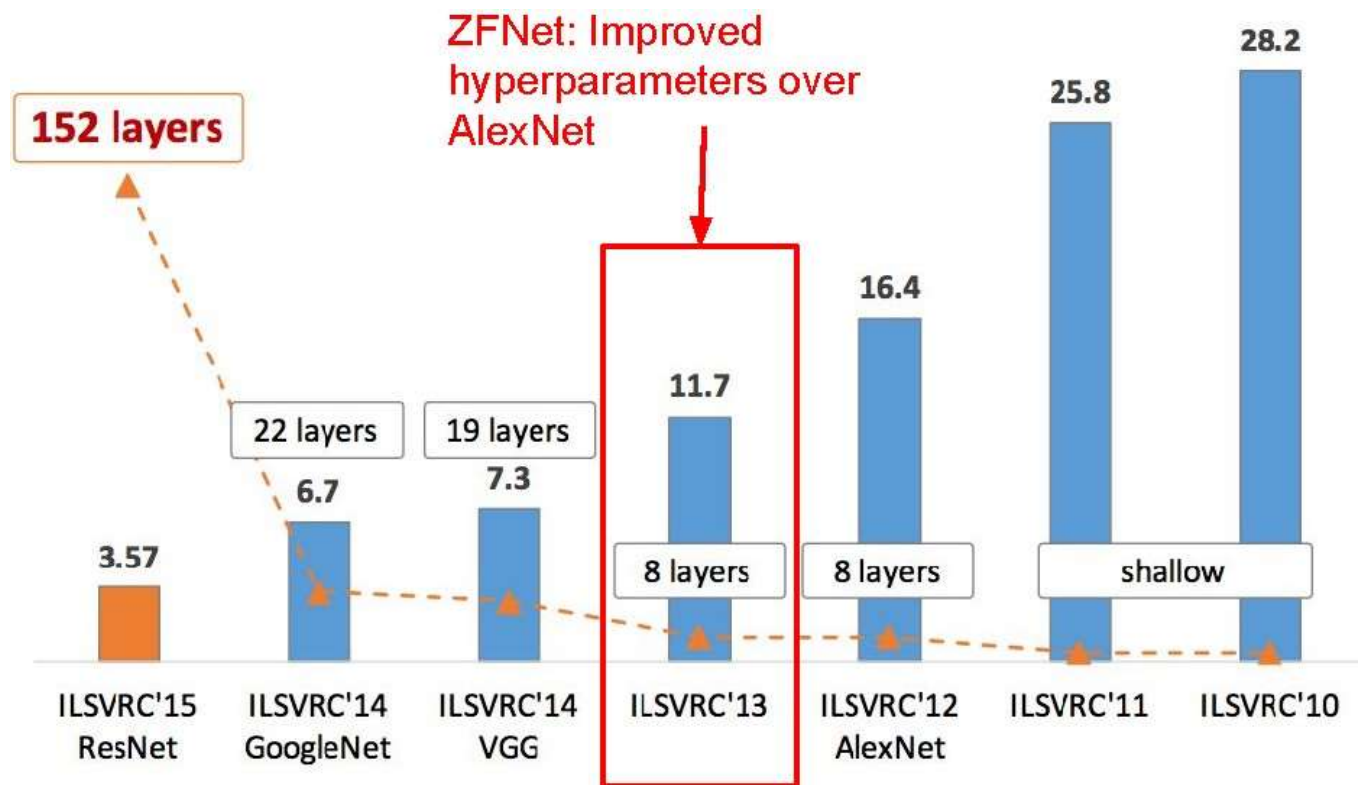
- Trained on GTX 580 GPU with only 3 GB of memory.
- Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.
- CONV1, CONV2, CONV4, CONV5:
Connections only with feature maps on same GPU.
- CONV3, FC6, FC7, FC8:
Connections with all feature maps in preceding layer, communication across GPUs.



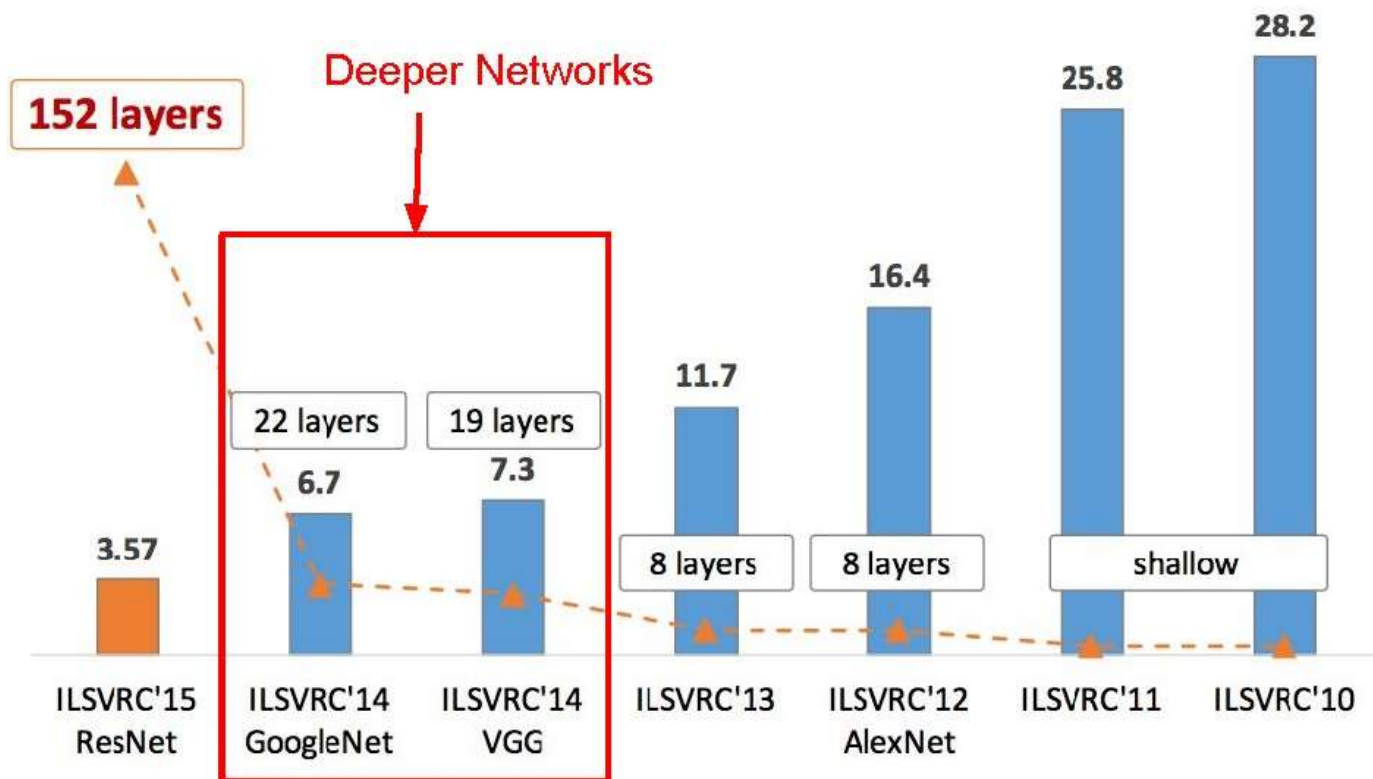
AlexNet

AlexNet was the coming out party for CNNs in the computer vision community. This was **the first time a model performed so well on a historically difficult ImageNet dataset**. This paper illustrated the benefits of CNNs and backed them up with record breaking performance in the competition.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



VGGNet

- *Very Deep Convolutional Networks For Large Scale Image Recognition - Karen Simonyan and Andrew Zisserman; 2015*
- The runner-up at the ILSVRC 2014 competition
- Significantly deeper than AlexNet
- 140 million parameters

VGGNet

- **Smaller filters**
Only 3x3 CONV filters, stride 1, pad 1
and 2x2 MAX POOL , stride 2
- **Deeper network**
AlexNet: 8 layers
VGGNet: 16 - 19 layers
- ZFNet: 11.7% top 5 error in ILSVRC'13
- VGGNet: 7.3% top 5 error in ILSVRC'14

Input

3x3 conv, 64

3x3 conv, 64

Pool 1/2

3x3 conv, 128

3x3 conv, 128

Pool 1/2

3x3 conv, 256

3x3 conv, 256

Pool 1/2

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Pool 1/2

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Pool 1/2

FC 4096

FC 4096

FC 1000

Softmax

VGGNet

- **Why use smaller filters? (3x3 conv)**

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

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

7x7

But deeper, more non-linearities

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

VGGNet

VGG16:

TOTAL memory: $24M * 4 \text{ bytes} \approx 96MB$ / image

TOTAL params: 138M parameters

Input

3x3 conv, 64

3x3 conv, 64

Pool

3x3 conv, 128

3x3 conv, 128

Pool

3x3 conv, 256

3x3 conv, 256

3x3 conv, 256

Pool

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Pool

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Pool

FC 4096

FC 4096

FC 1000

Softmax

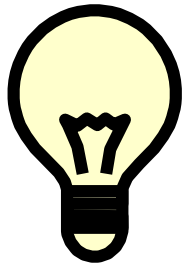
[Simonyan and Zisserman, 2014]

| | | |
|---------------|-----------------------------------|--------------------------------------|
| Input | memory: $224*224*3=150\text{K}$ | params: 0 |
| 3x3 conv, 64 | memory: $224*224*64=3.2\text{M}$ | params: $(3*3*3)*64 = 1,728$ |
| 3x3 conv, 64 | memory: $224*224*64=3.2\text{M}$ | params: $(3*3*64)*64 = 36,864$ |
| Pool | memory: $112*112*64=800\text{K}$ | params: 0 |
| 3x3 conv, 128 | memory: $112*112*128=1.6\text{M}$ | params: $(3*3*64)*128 = 73,728$ |
| 3x3 conv, 128 | memory: $112*112*128=1.6\text{M}$ | params: $(3*3*128)*128 =$ |
| 147,456 | | |
| Pool | memory: $56*56*128=400\text{K}$ | params: 0 |
| 3x3 conv, 256 | memory: $56*56*256=800\text{K}$ | params: $(3*3*128)*256 = 294,912$ |
| 3x3 conv, 256 | memory: $56*56*256=800\text{K}$ | params: $(3*3*256)*256 = 589,824$ |
| 3x3 conv, 256 | memory: $56*56*256=800\text{K}$ | params: $(3*3*256)*256 = 589,824$ |
| Pool | memory: $28*28*256=200\text{K}$ | params: 0 |
| 3x3 conv, 512 | memory: $28*28*512=400\text{K}$ | params: $(3*3*256)*512 = 1,179,648$ |
| 3x3 conv, 512 | memory: $28*28*512=400\text{K}$ | params: $(3*3*512)*512 = 2,359,296$ |
| 3x3 conv, 512 | memory: $28*28*512=400\text{K}$ | params: $(3*3*512)*512 = 2,359,296$ |
| Pool | memory: $14*14*512=100\text{K}$ | params: 0 |
| 3x3 conv, 512 | memory: $14*14*512=100\text{K}$ | params: $(3*3*512)*512 = 2,359,296$ |
| 3x3 conv, 512 | memory: $14*14*512=100\text{K}$ | params: $(3*3*512)*512 = 2,359,296$ |
| 3x3 conv, 512 | memory: $14*14*512=100\text{K}$ | params: $(3*3*512)*512 = 2,359,296$ |
| Pool | memory: $7*7*512=25\text{K}$ | params: 0 |
| FC 4096 | memory: 4096 | params: $7*7*512*4096 = 102,760,448$ |
| FC 4096 | memory: 4096 | params: $4096*4096 = 16,777,216$ |
| FC 1000 | memory: 1000 | params: $4096*1000 = 4,096,000$ |

VGGNet

Details/Retrospectives :

- ILSVRC'14 2nd in classification, 1st in localization
- Similar training procedure as AlexNet
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks
- Trained on 4 Nvidia Titan Black GPUs for **two to three weeks**.



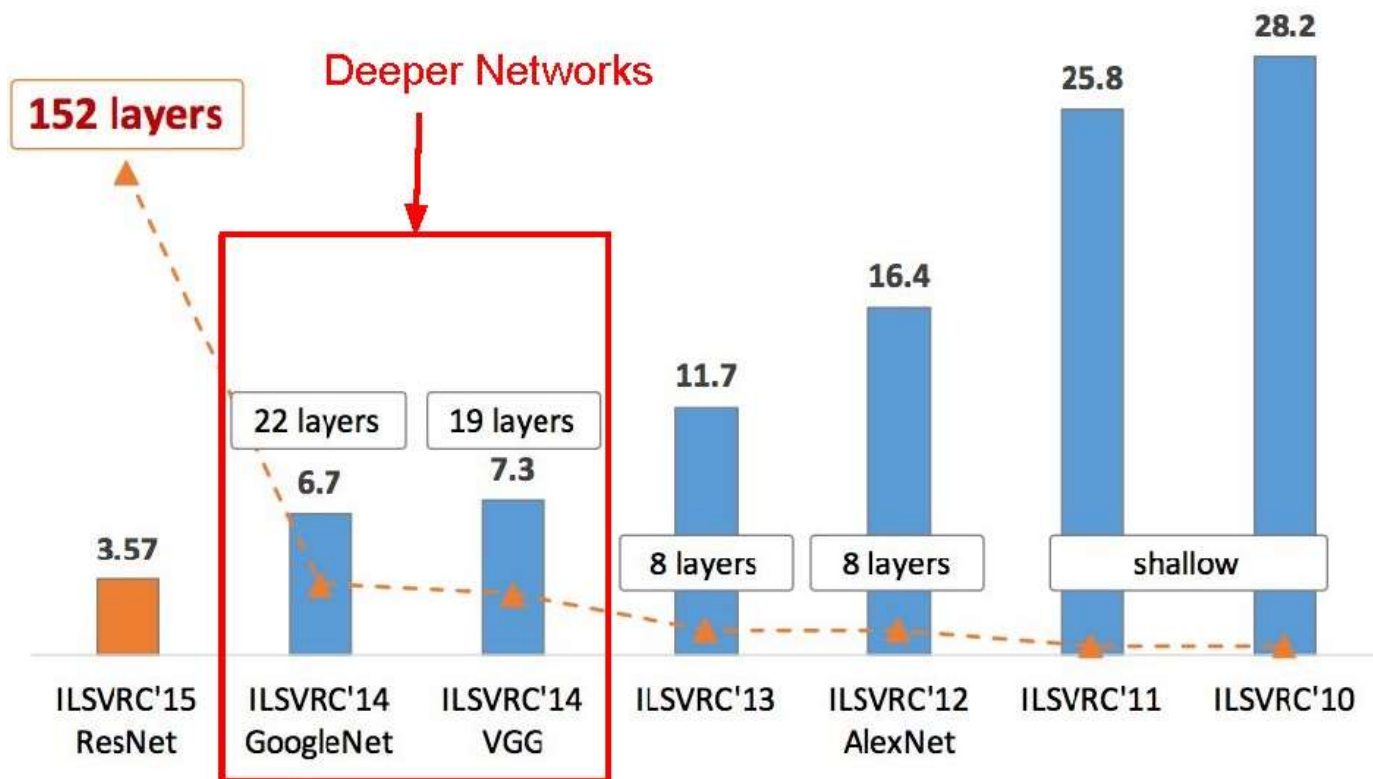
VGGNet

VGG Net reinforced the notion that **convolutional neural networks have to have a deep network of layers in order for this hierarchical representation of visual data to work.**

Keep it deep.

Keep it simple.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

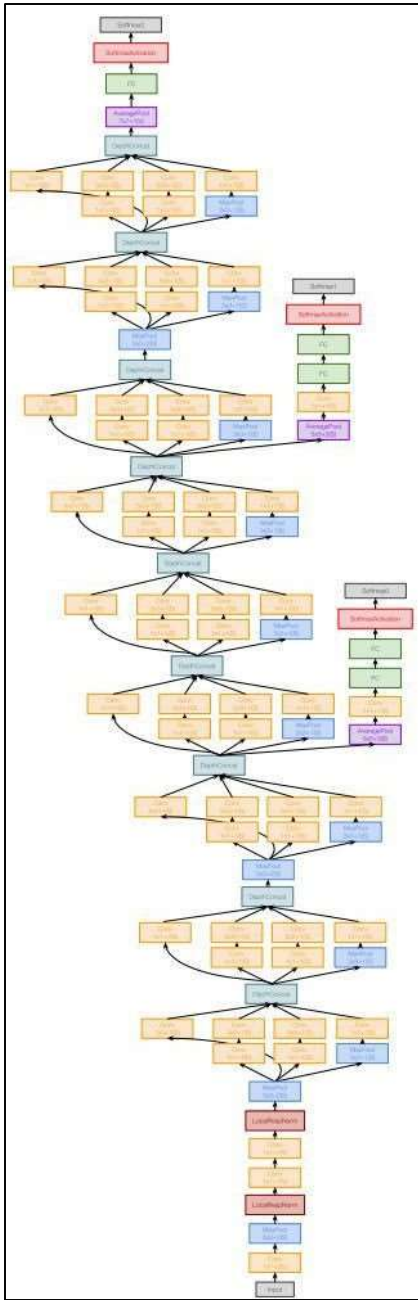


GoogleNet

- *Going Deeper with Convolutions - Christian Szegedy et al.; 2015*
- ILSVRC 2014 competition winner
- Also significantly deeper than AlexNet
- x12 less parameters than AlexNet
- Focused on computational efficiency

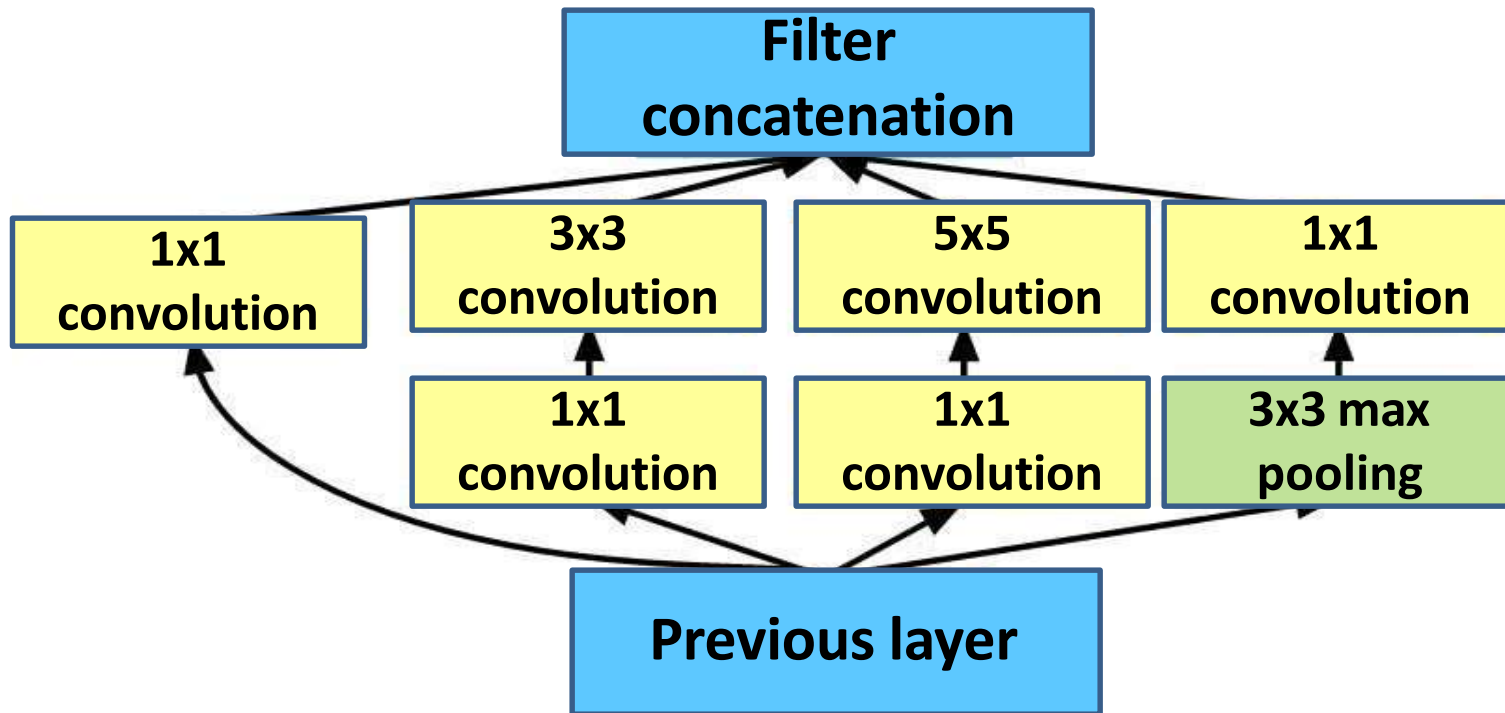
GoogleNet

- 22 layers
- Efficient **“Inception” module** - strayed from the general approach of simply stacking conv and pooling layers on top of each other in a sequential structure
- No FC layers
- Only 5 million parameters!
- ILSVRC’14 classification winner (6.7% top 5 error)



GoogleNet

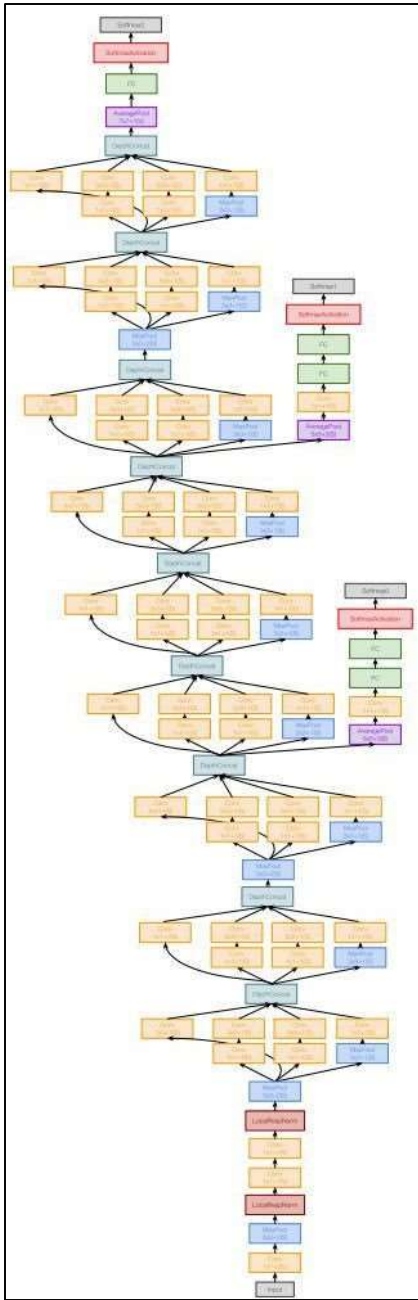
“Inception module”: design a good local network topology (network within a network) and then stack these modules on top of each other

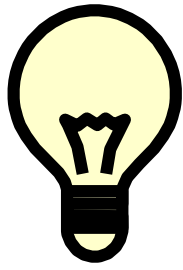


GoogleNet

Details/Retrospectives :

- Deeper networks, with computational efficiency
- 22 layers
- Efficient “Inception” module
- No FC layers
- 12x less params than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)

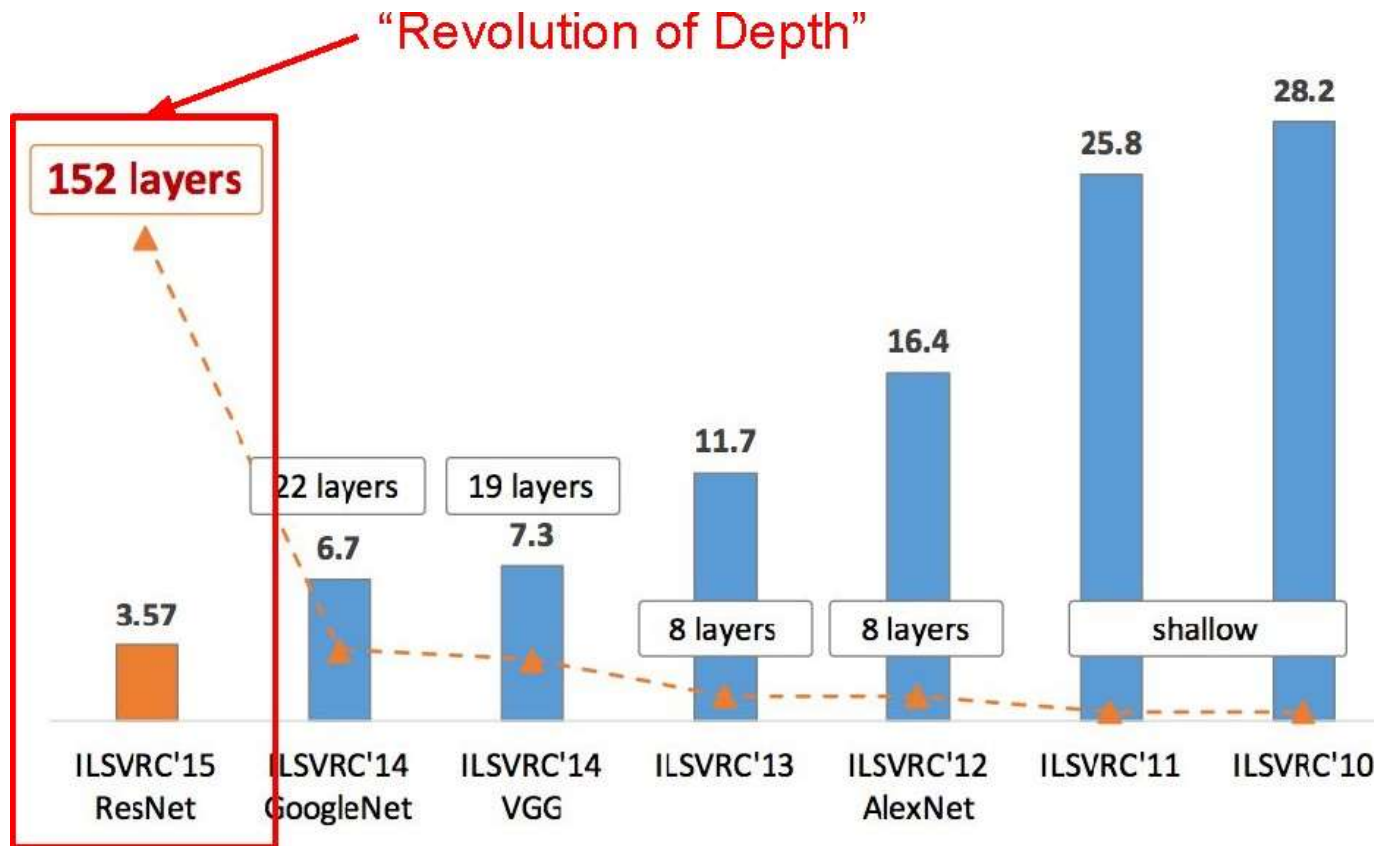




GoogleNet

Introduced the idea that CNN layers **didn't always have to be stacked up sequentially**. Coming up with the Inception module, the authors showed that a creative structuring of layers can lead to improved performance and **computationally efficiency**.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

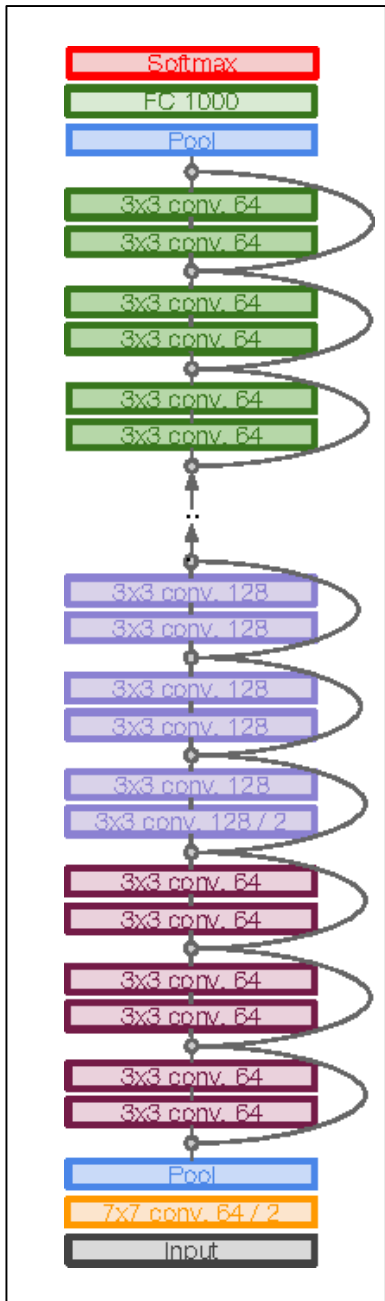


ResNet

- *Deep Residual Learning for Image Recognition - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun; 2015*
- Extremely deep network – 152 layers
- Deeper neural networks are more difficult to train.
- Deep networks suffer from vanishing and exploding gradients.
- Present a residual learning framework to ease the training of networks that are substantially deeper than those used previously.

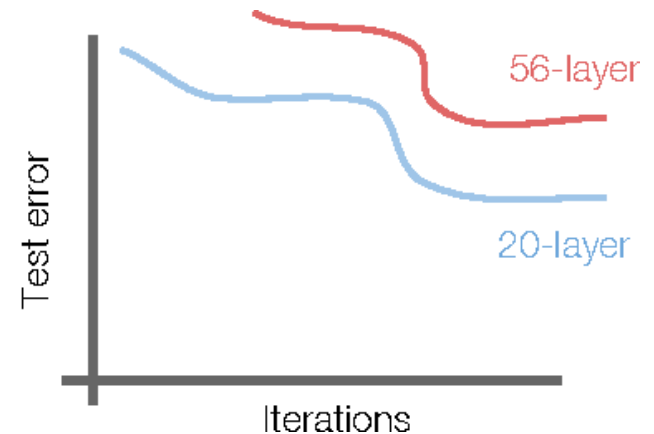
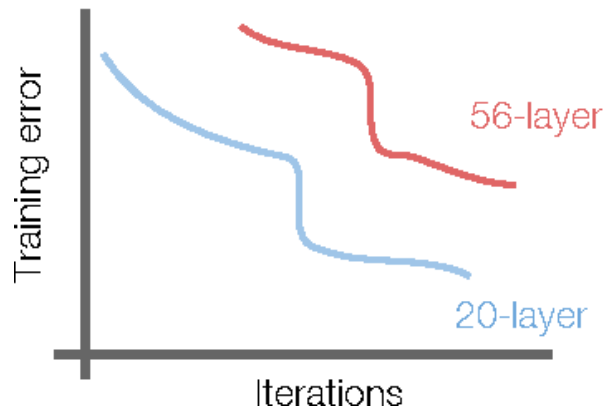
ResNet

- ILSVRC'15 classification winner (3.57% top 5 error, humans generally hover around a 5-10% error rate)
Swept all classification and detection competitions in ILSVRC'15 and COCO'15!



ResNet

- What happens when we continue stacking deeper layers on a convolutional neural network?



- 56-layer model performs worse on both training and test error
-> The deeper model performs worse (not caused by overfitting)!

ResNet

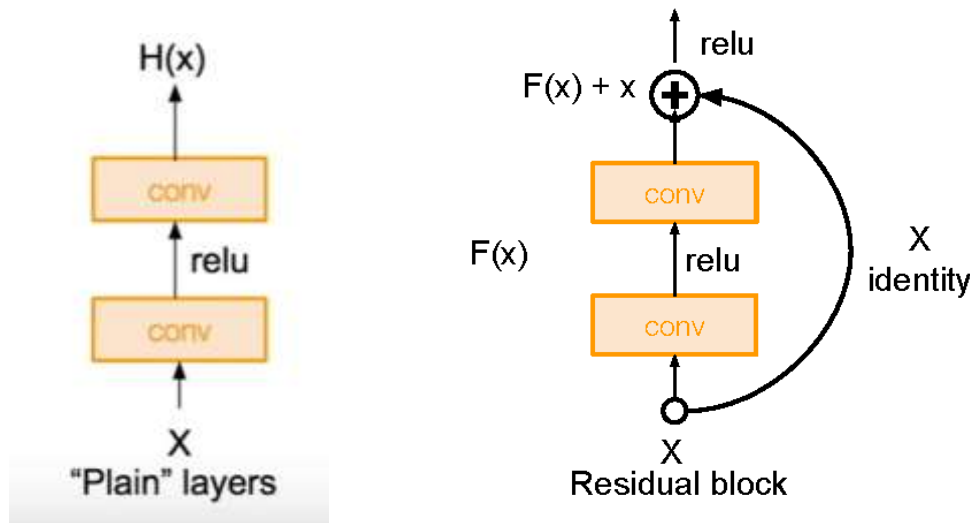
- **Hypothesis:** The problem is an optimization problem. Very deep networks are harder to optimize.
- **Solution:** Use network layers to fit residual mapping instead of directly trying to fit a desired underlying mapping.
- We will use **skip connections** allowing us to take the activation from one layer and feed it into another layer, much deeper into the network.
- Use layers to fit residual $F(x) = H(x) - x$ instead of $H(x)$ directly

ResNet

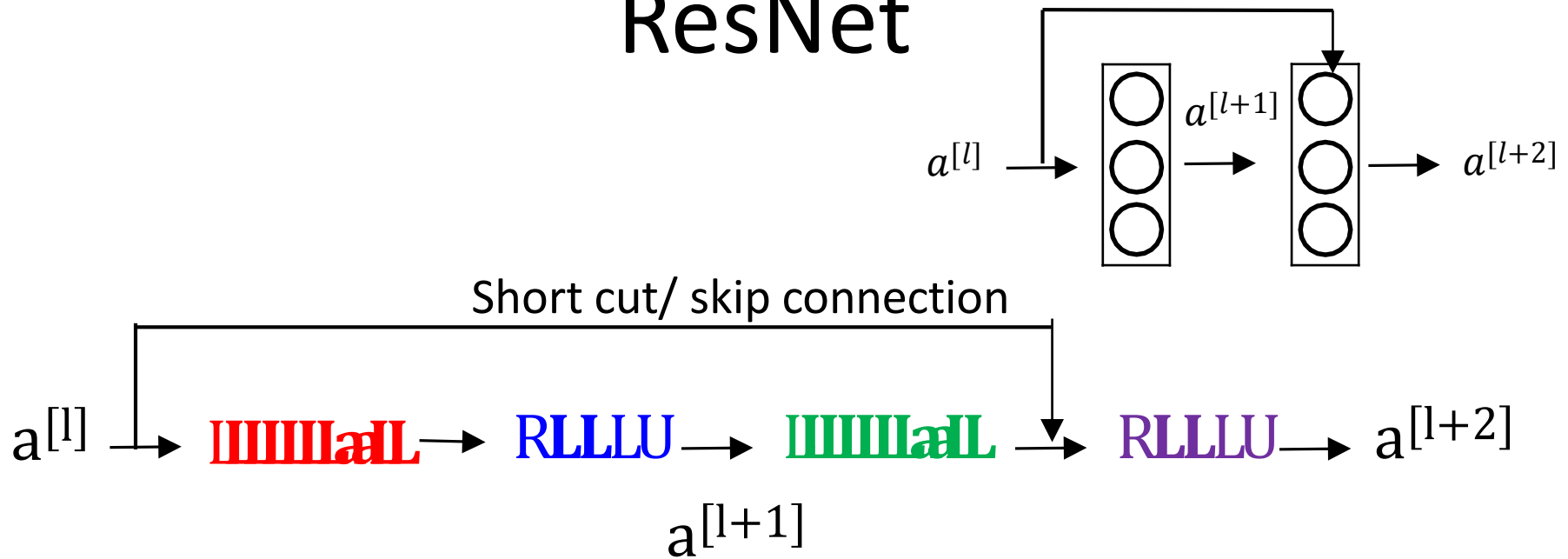
Residual Block

Input x goes through conv-relu-conv series and gives us $F(x)$. That result is then added to the original input x . Let's call that $H(x) = F(x) + x$.

In traditional CNNs, $H(x)$ would just be equal to $F(x)$. So, instead of just computing that transformation (straight from x to $F(x)$), we're computing the term that we have to *add*, $F(x)$, to the input, x .



ResNet



$$z^{[1]} = W^{[1]} a + b^{[1]}$$

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$$a^{[1+1]} = \text{gg}(z^{[1+1]})$$

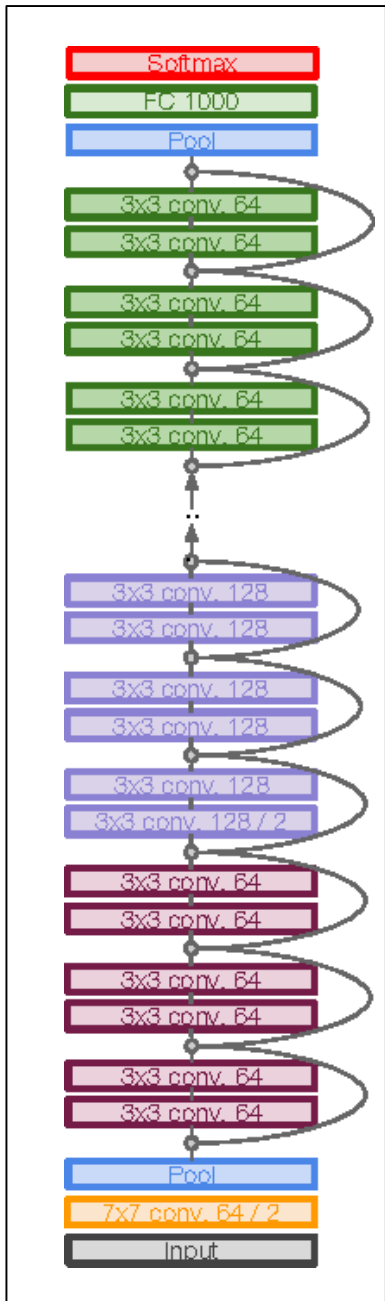
$$a^{[1+2]} = \text{gg}(z^{[1+2]})$$

$$a^{[2]} = \text{g} (z^{[2]} + a^{[1]}) = \text{g}(W^{[2]} a^{[1]} + b^{[2]} + a^{[1]}) \quad []$$

ResNet

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)



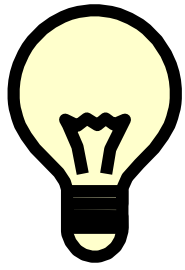
ResNet

- Total depths of 34, 50, 101, or 152 layers for ImageNet
- For deeper networks (ResNet-50+), use “bottleneck” layer to improve efficiency (similar to GoogLeNet)

ResNet

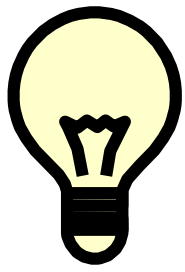
Experimental Results:

- Able to train very deep networks without degrading
- Deeper networks now achieve lower training errors as expected

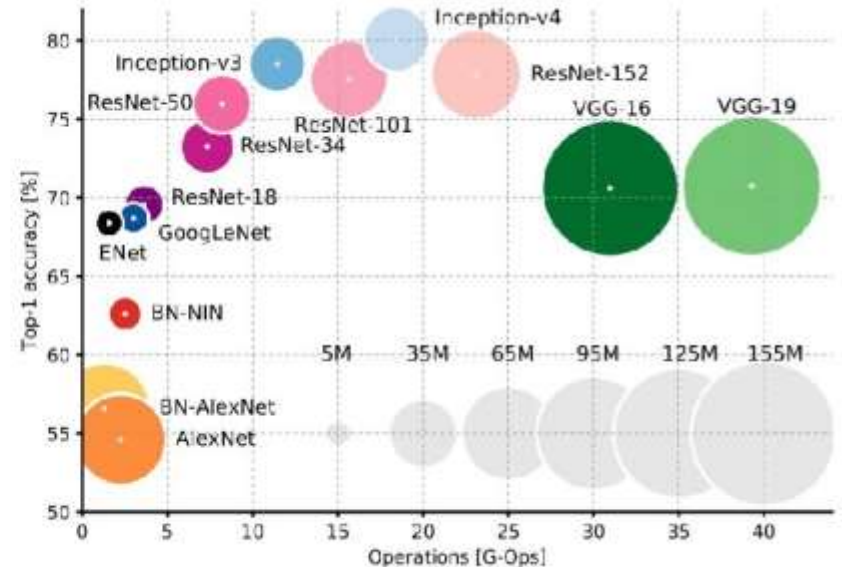
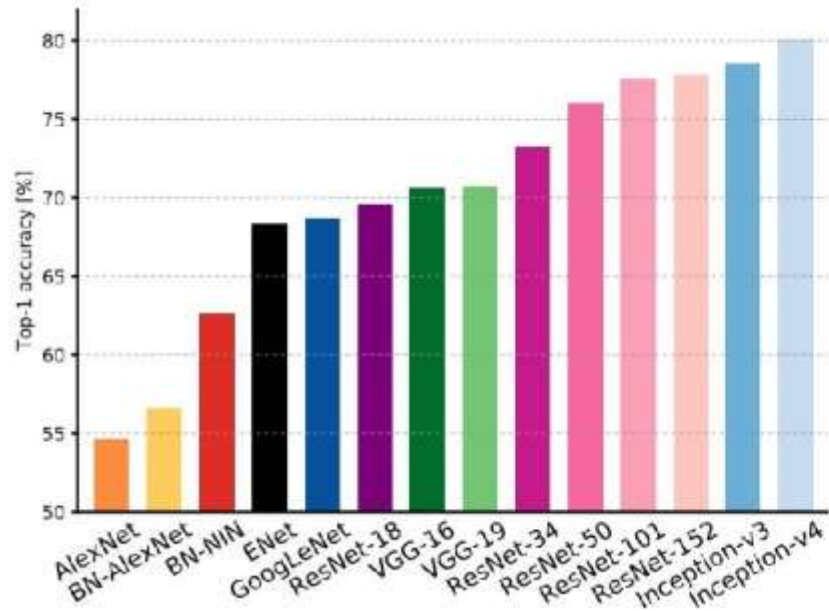


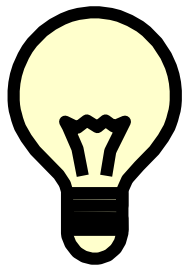
ResNet

The **best** CNN architecture that we currently have and is a great innovation for the idea of residual learning.
Even better than human performance!

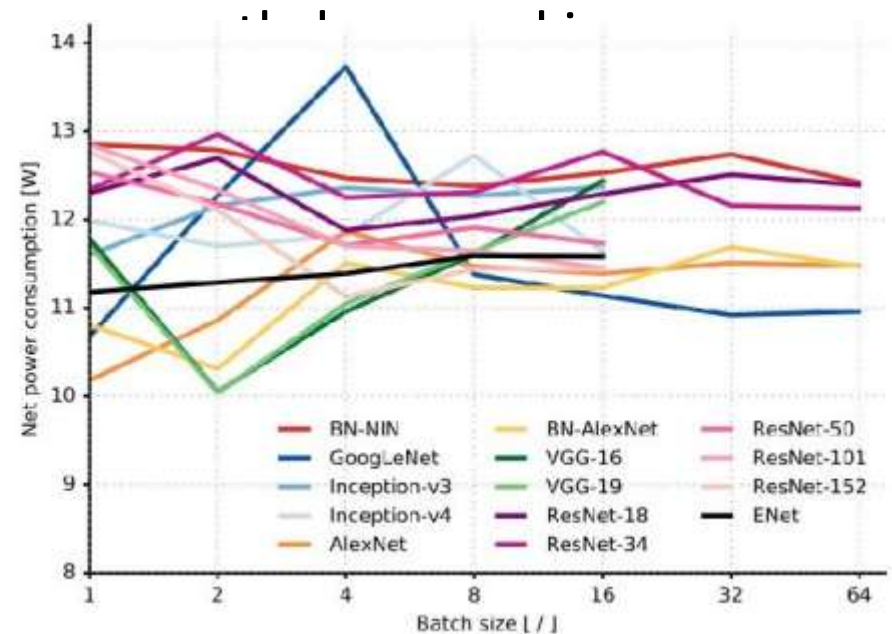
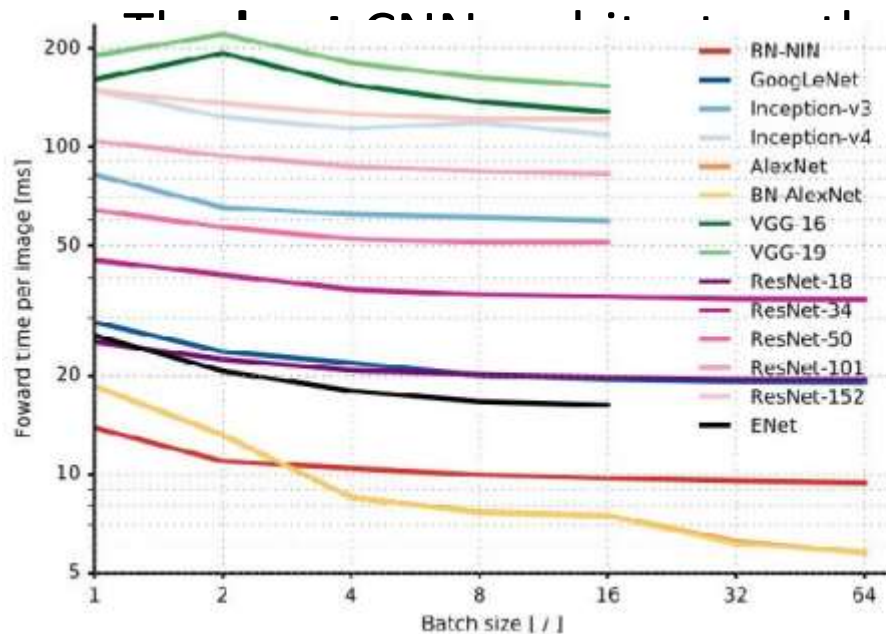


Accuracy comparison

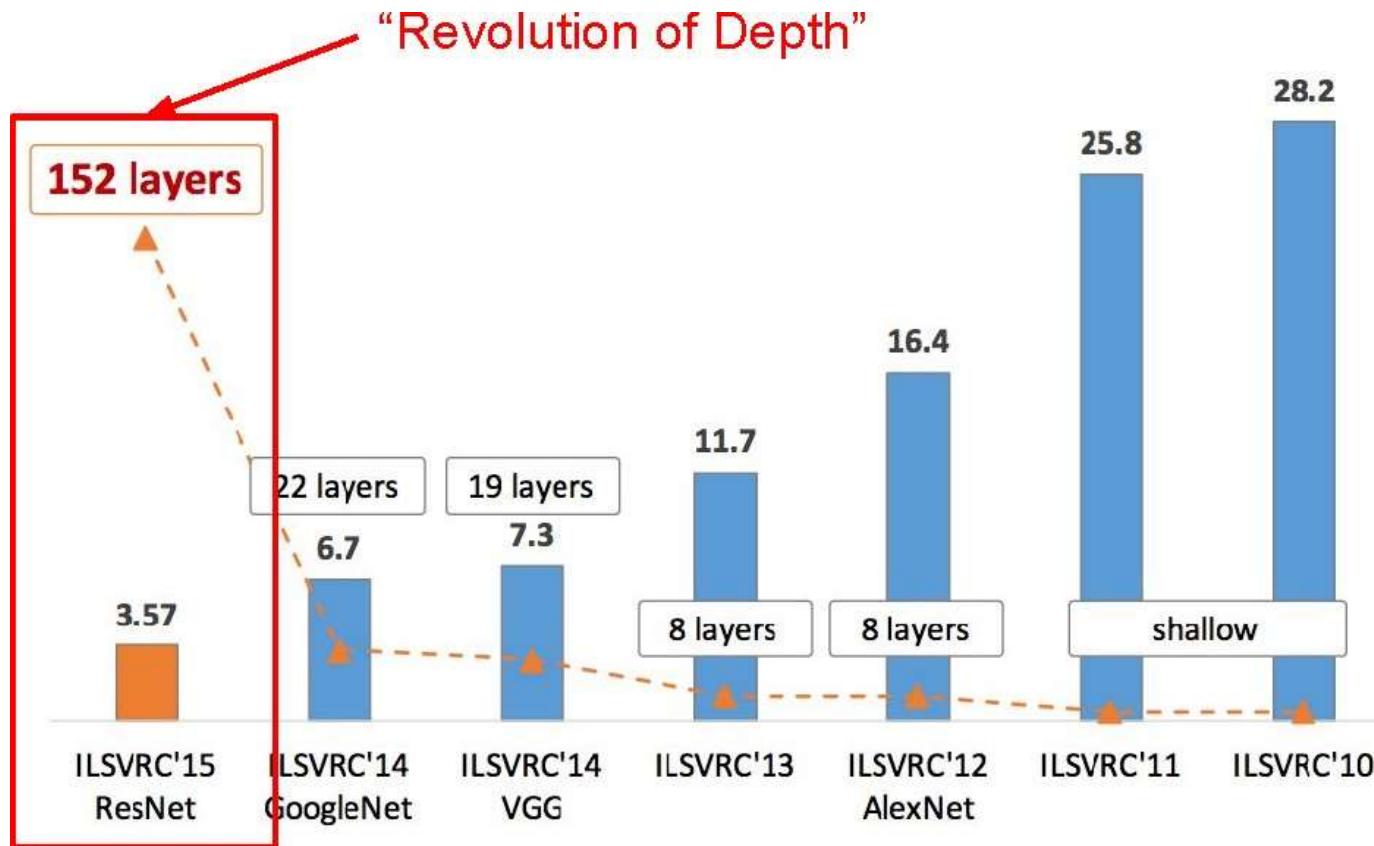




Forward pass time and power consumption



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Today: Convolutional Neural Networks (CNNs)

1. Scene understanding and object recognition for machines (and humans)

- Scene/object recognition challenge. Illusions reveal primitives, conflicting info
- Human neurons/circuits. Visual cortex layers==abstraction. General cognition

2. Classical machine vision foundations: features, scenes, filters, convolution

- Spatial structure primitives: edge detectors & other filters, feature recognition
- Convolution: basics, padding, stride, object recognition, architectures

3. CNN foundations: LeNet, *de novo* feature learning, parameter sharing

- Key ideas: *learn* features, hierarchy, re-use parameters, back-prop filter learning
- CNN formalization: representations(Conv+ReLU+Pool)*N layers + Fully-connected

4. Modern CNN architectures: millions of parameters, dozens of layers

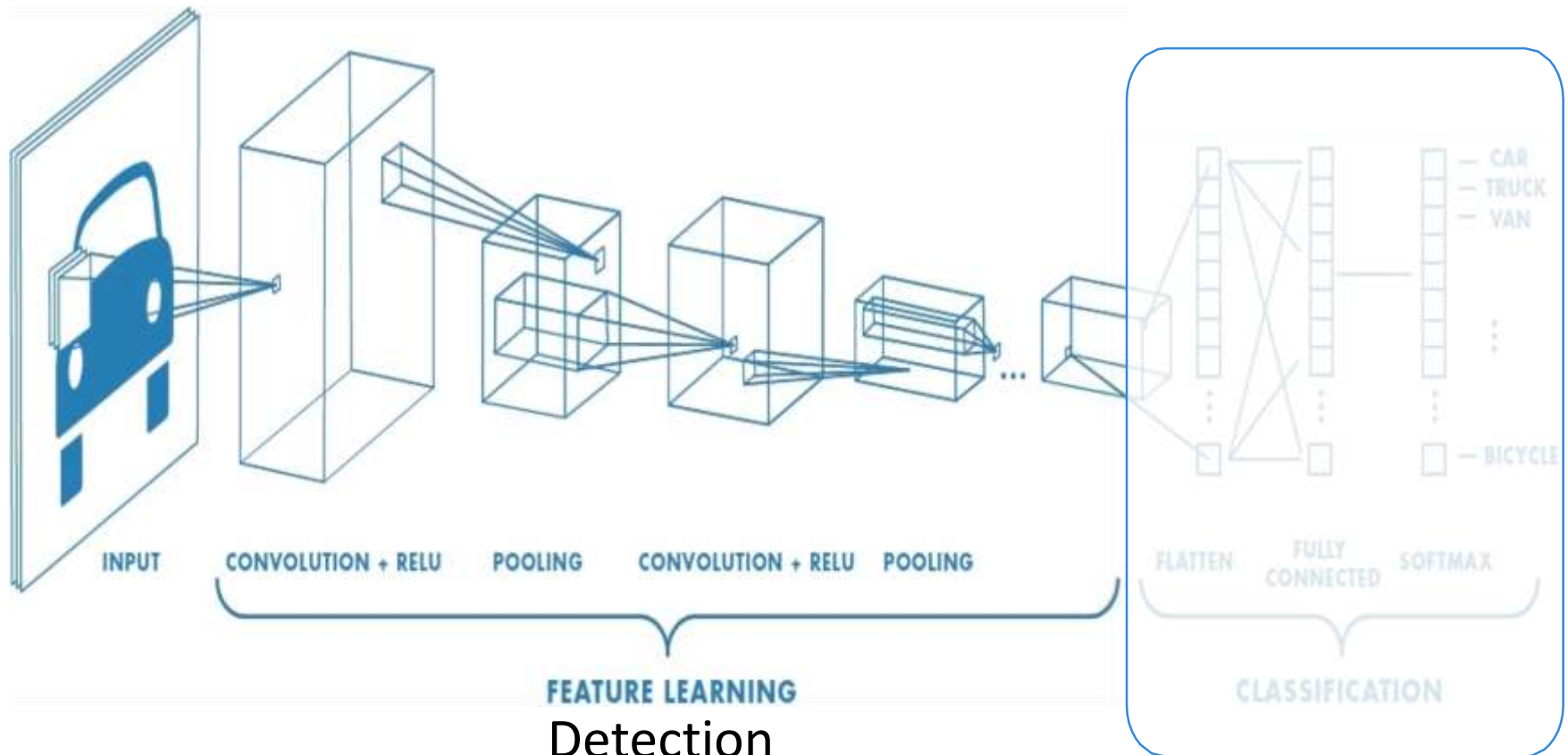
- Feature invariance is hard: apply perturbations, learn for each variation
- ImageNet progression of best performers
- AlexNet: First top performer CNN, 60M parameters (from 60k in LeNet-5), ReLU
- VGGNet: simpler but deeper (8→19 layers), 140M parameters, ensembles
- GoogleNet: new primitive=inception module, 5M params, no FC, efficiency
- ResNet: 152 layers, vanishing gradients → fit residuals to enable learning

5. Countless applications: General architecture, enormous power

- Semantic segmentation, facial detection/recognition, self-driving, image colorization, optimizing pictures/scenes, up-scaling, medicine, biology, genomics

Countless applications

An Architecture for Many Applications



Detection

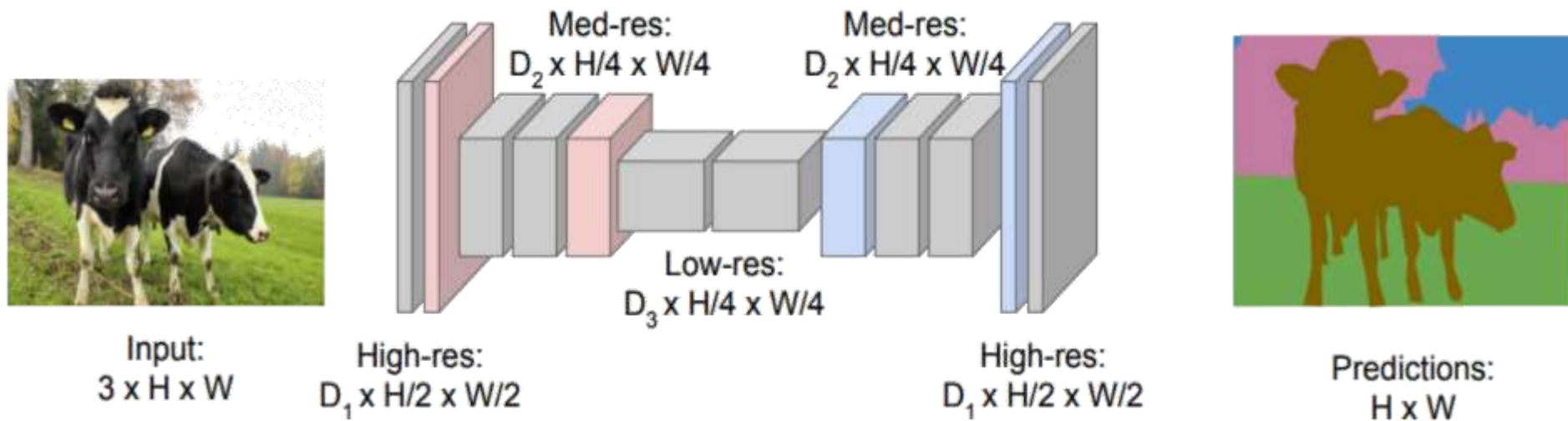
Semantic segmentation

End-to-end robotic control

Semantic Segmentation: Fully Convolutional Networks

FCN: Fully Convolutional Network.

Network designed with all convolutional layers, with **downsampling** and **upsampling** operations



`tf.keras.layers.Conv2DTranspose`