Lecture 7: Convolutional Networks (CNNs)

COMPSCI/DATA 182: Deep Learning

09/19/2024

Today’s menu

- A new architecture: Convolutional Neural Networks (aka CNNs)
- Walk through some simple PyTorch code
  - Illustrating using the torch library to build CNN models
- References
  - Chapter 10: Convolutional Networks of *Bishop Book*

Structure in Data

- Data: unstructured vs structured data in machine learning
- Auto-regressive
- Images, pixels: spatial relationships

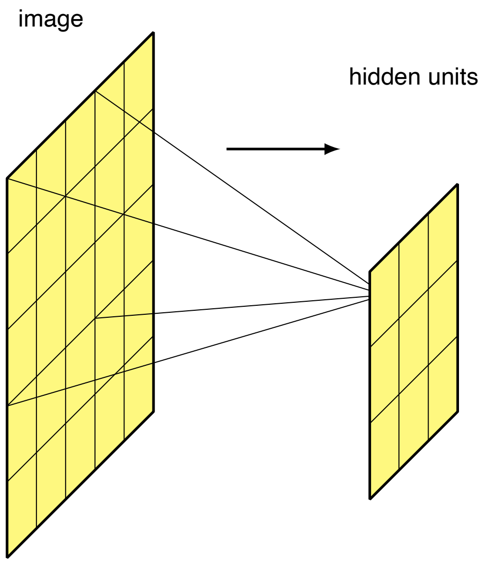
The Convolutional Network Architecture

- Sparsely connected
- Parameter sharing
- Computer vision:
  - History
  - Applications

Image Analysis

- 2D/3D; pixels/voxels
- Local relationships → Generalization !
- REDUCTION in parameters

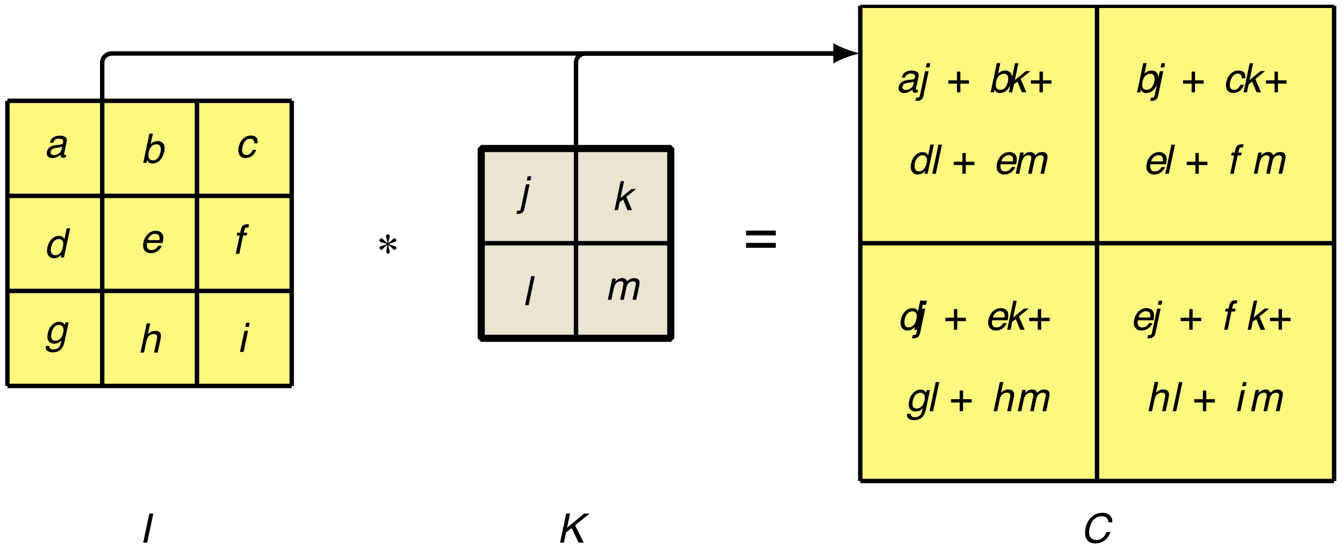
The Convolution Filter



0.4	1.7	0.9
2.3	−2.1	4.0
−1.4	0.7	2.1

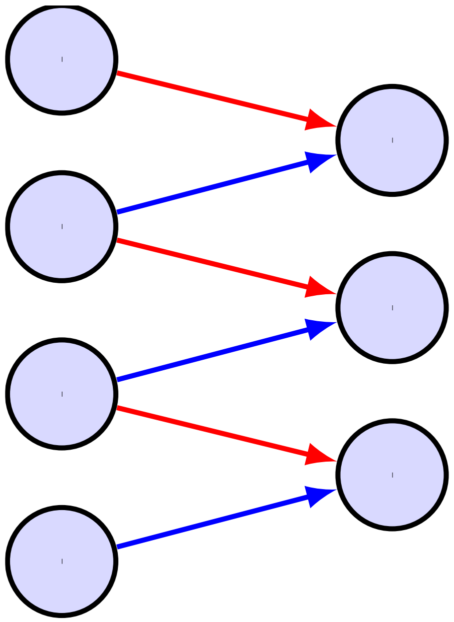
- $z = \mathrm{ReLU}(w_T x + w_0)$
- the *Kernel*

Convolution

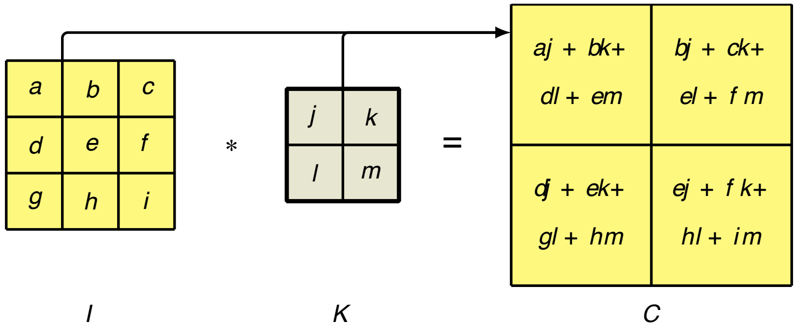


Translation equivariance

- Hidden layer units → *feature map*
- $C(j,k) = \sum_l \sum_m I(j+k, l+m)\, K(l,m)$
- **C = I * K**



Convolution



- Horizontal and vertical edge detection filters
  - Examples
- Generalizability and Scalability advantages of CNN filters

-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
0	0	0
1	1	1

Example

-1	0	1
-1	0	1
-1	0	1

0	50	50	0	0	0
0	50	50	0	0	0
0	50	50	0	0	0
0	50	50	0	0	0
0	50	50	0	0	0

150	-150	-150	0
150	-150	-150	0
150	-150	-150	0

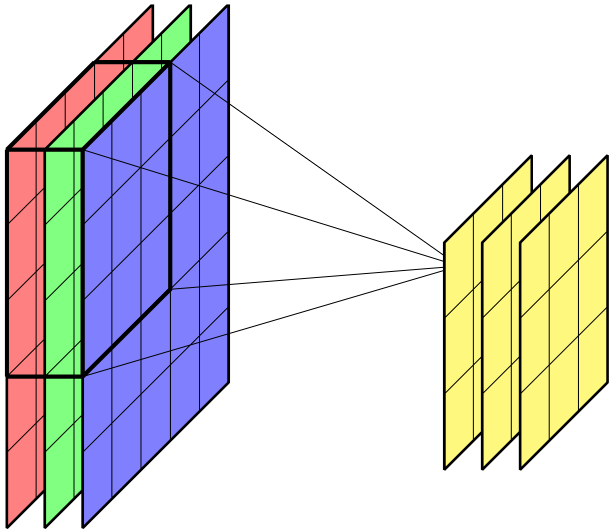
(a) (b) (c)

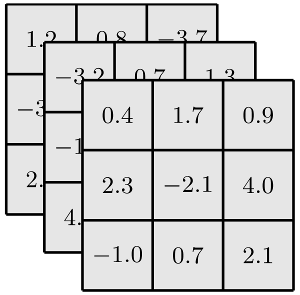
Padding, Strided Convolutions

- Image: $J\ X\ K$ pixels
- Conv filter: $M\ X\ M$
- Feature map: $(J\text{-}M\text{+}1)\ X\ (K\text{-}M\text{+}1)$
- Padding: $(J\ \text{+}2P\text{-}M\text{+}1)\ X\ (K\ \text{+}2P\text{-}M\text{+}1)$
- Strides: $\left\lfloor \frac{J+2P-M}{S} - 1 \right\rfloor \times \left\lfloor \frac{K+2P-M}{S} - 1 \right\rfloor$

0	0	0	0	0	0
0	X_{11}	X_{12}	X_{13}	X_{14}	0
0	X_{21}	X_{22}	X_{23}	X_{24}	0
0	X_{31}	X_{32}	X_{33}	X_{34}	0
0	X_{41}	X_{42}	X_{43}	X_{44}	0
0	0	0	0	0	0

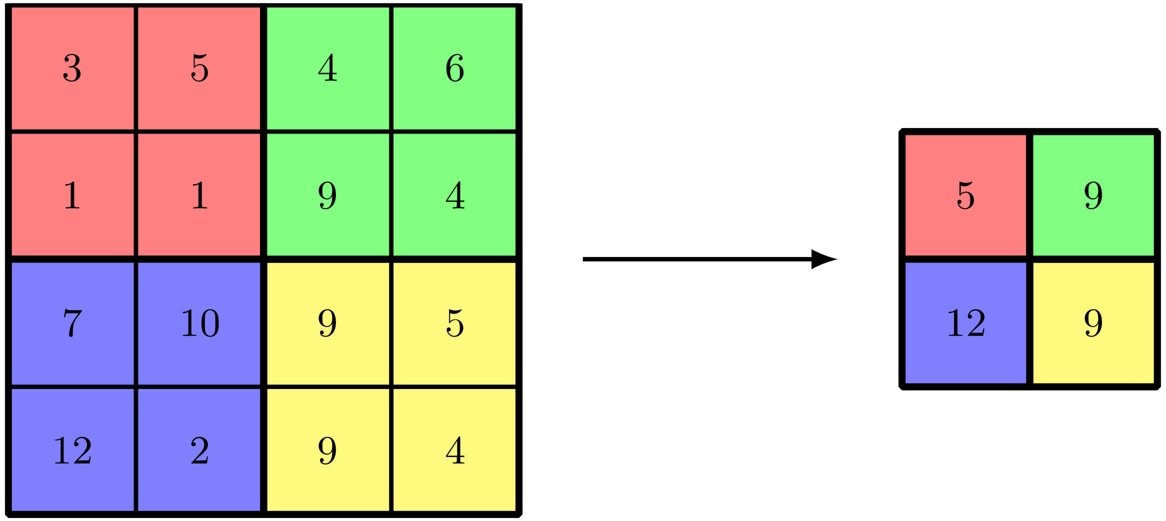
Multi-dimensional Convolutions



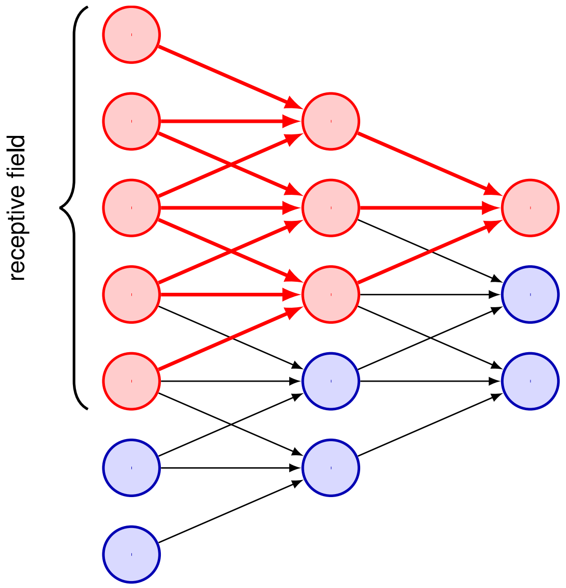


- Channels
- Filter Tensor dimensionality: $M\ X\ M\ X\ C\ X\ C_{OUT}$
- Parameters: $(M^2C+1)C_{OUT}$

Pooling



Multilayer Convolutions



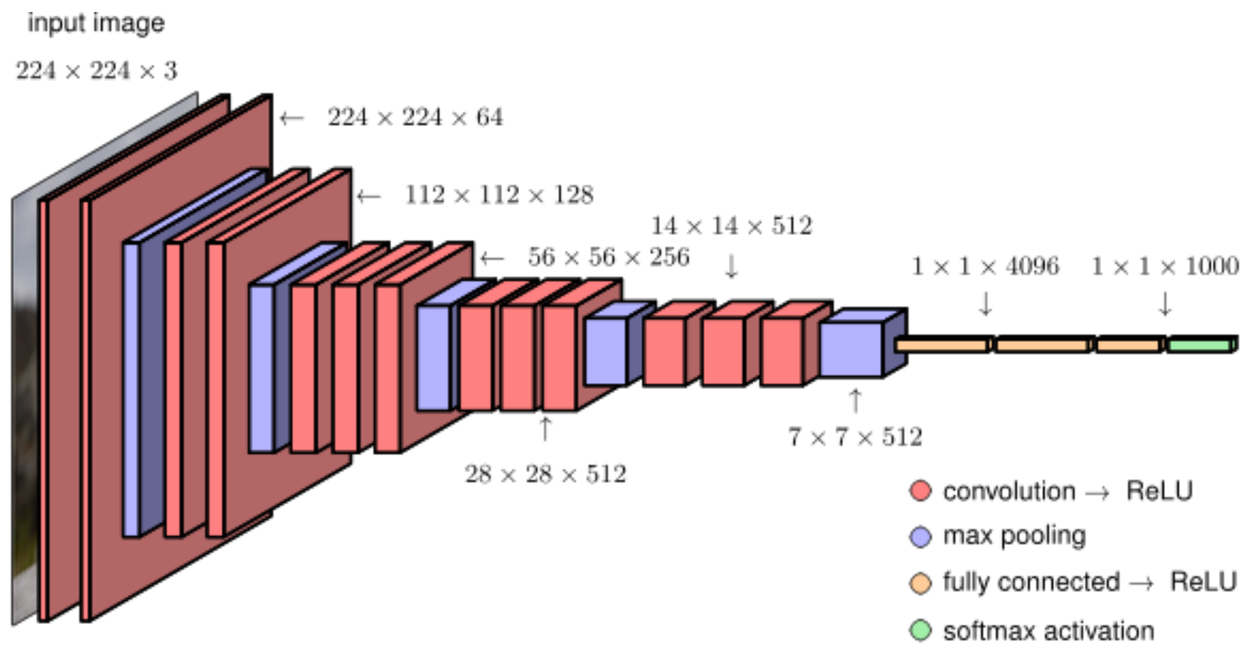
- *Receptive field*

Example Architectures

Example architectures

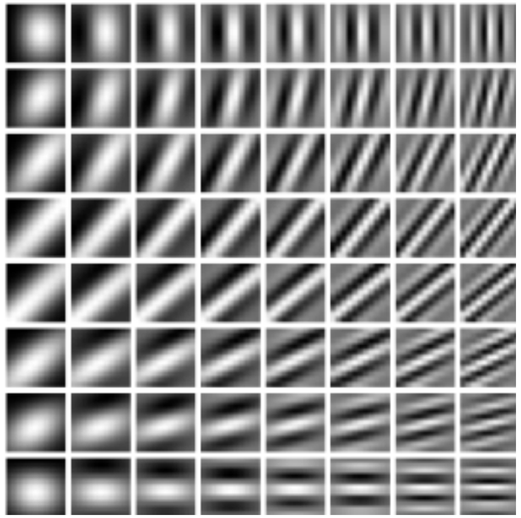
- LeNet
- ImageNet
- AlexNet

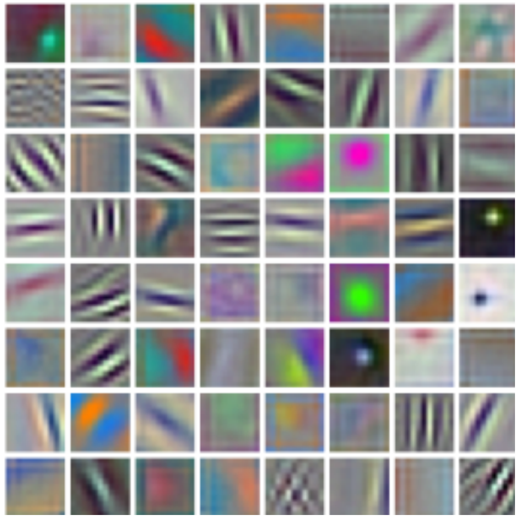
VGG16



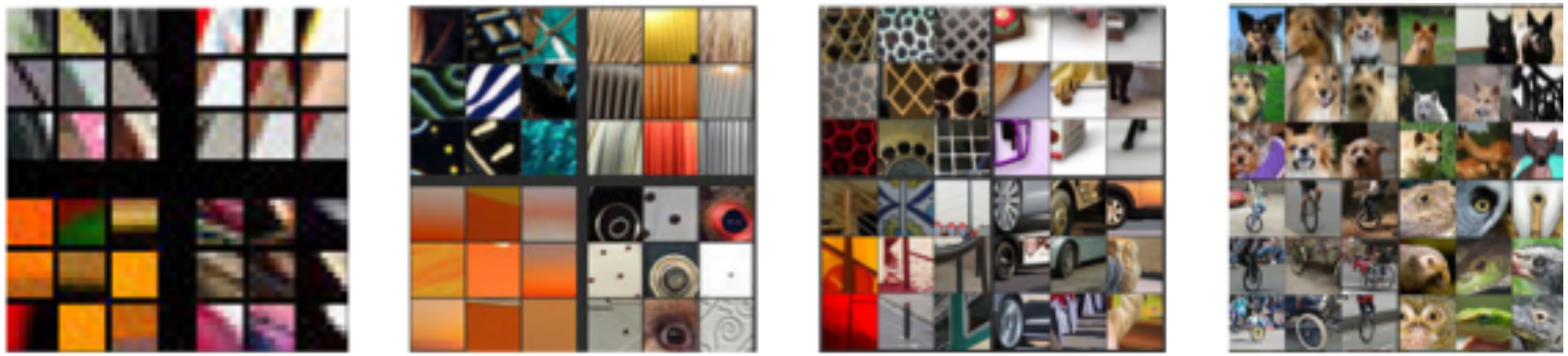
input image
224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution → ReLU
max pooling
fully connected → ReLU
softmax activation

Visual Cortex and Gabor filters



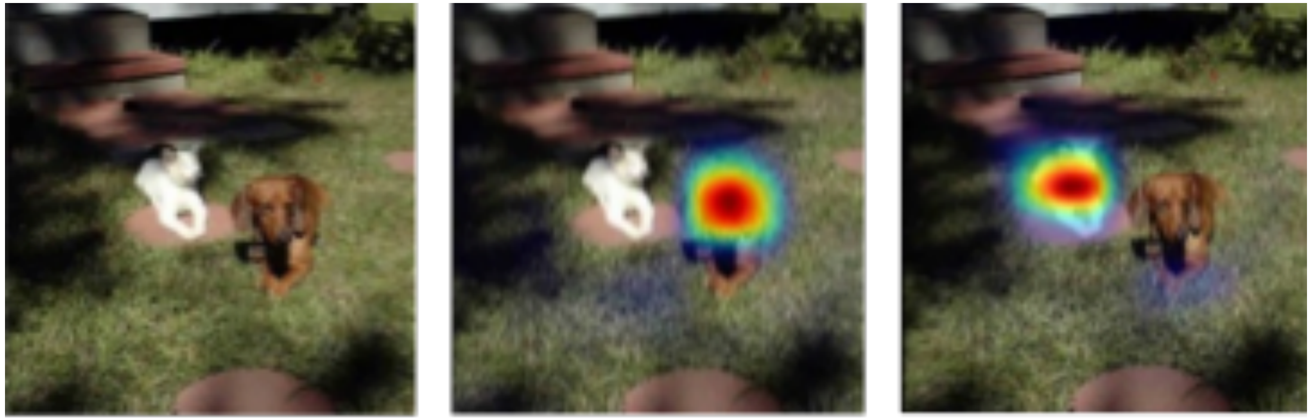


Visualizing Trained Filters



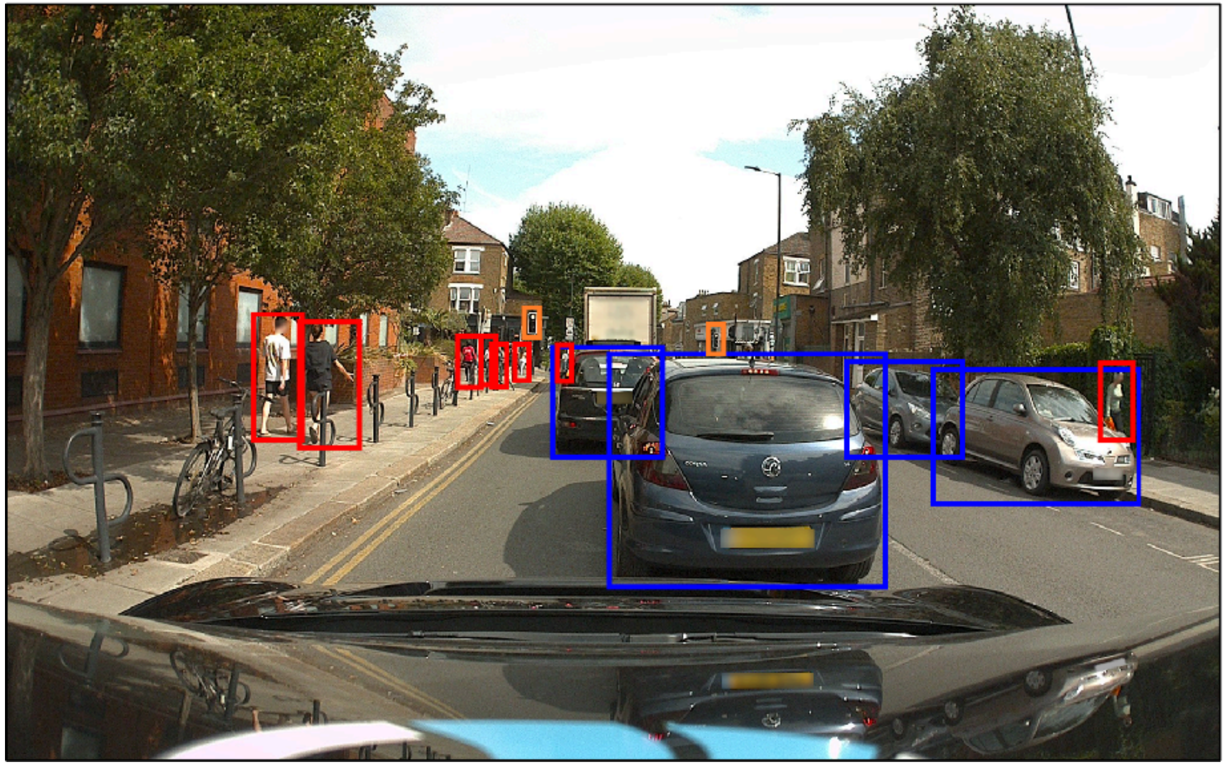
Layer 1 Layer 2 Layer 3 Layer 5

- Saliency maps



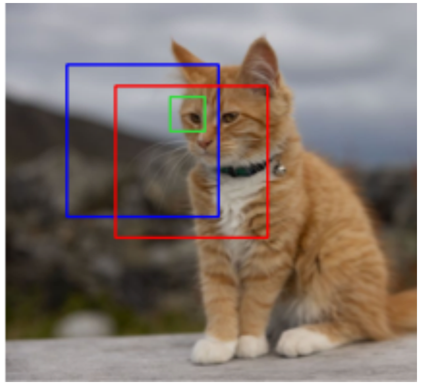
Original image Saliency map for 'dog' Saliency map for 'cat'

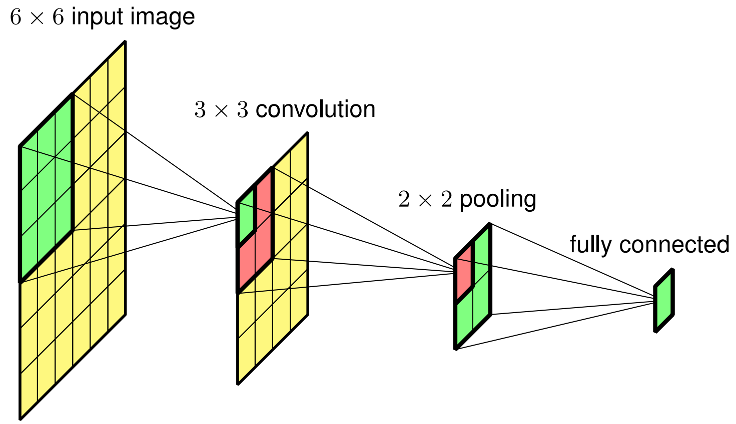
Object detection

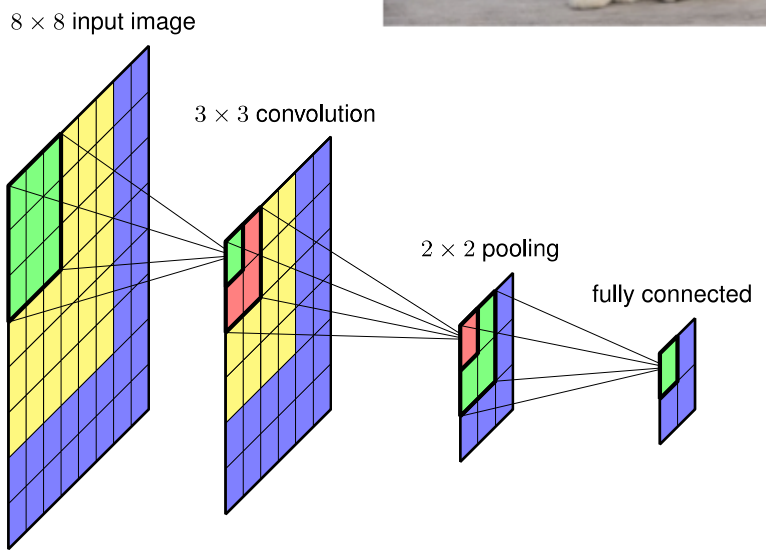


- Bounding Boxes

Sliding windows







- Efficient object detection

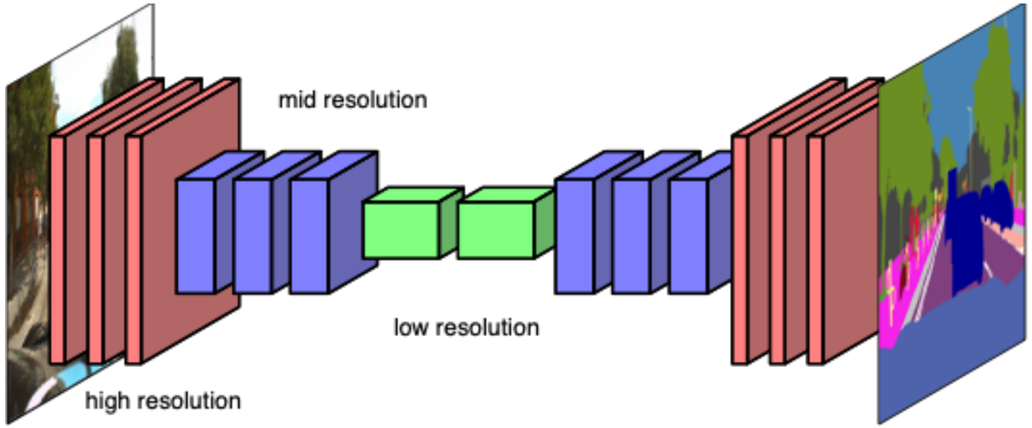
Detection across scales

(a)

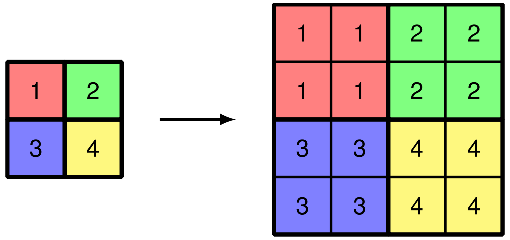
(b)

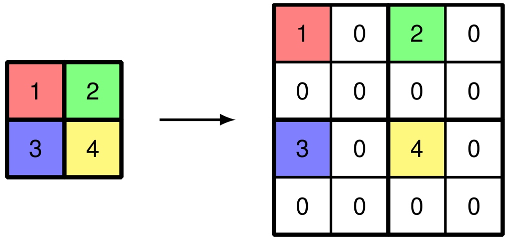
(c)

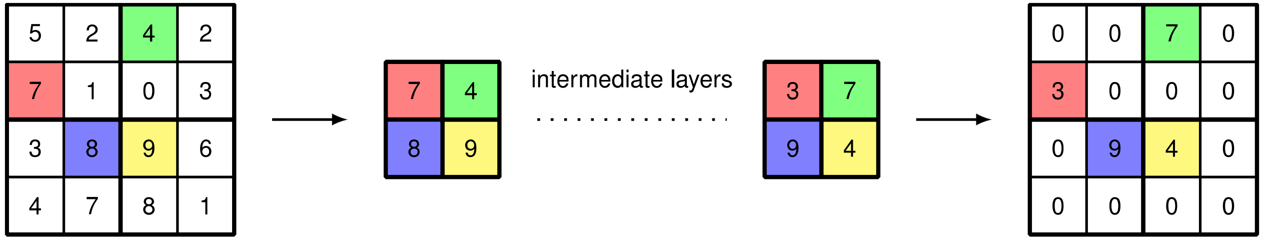
Upsampling

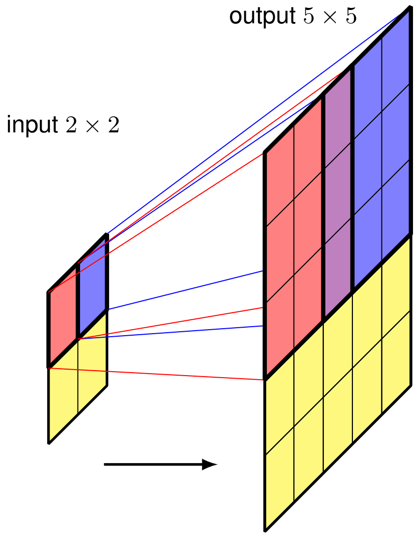


Fully Convolutional, Transpose convolution (Deconvolution)









Transpose convolution

Input matrix:

$$\begin{bmatrix} 1 & 0 \\ 2 & 3 \end{bmatrix}$$

Kernel matrix:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

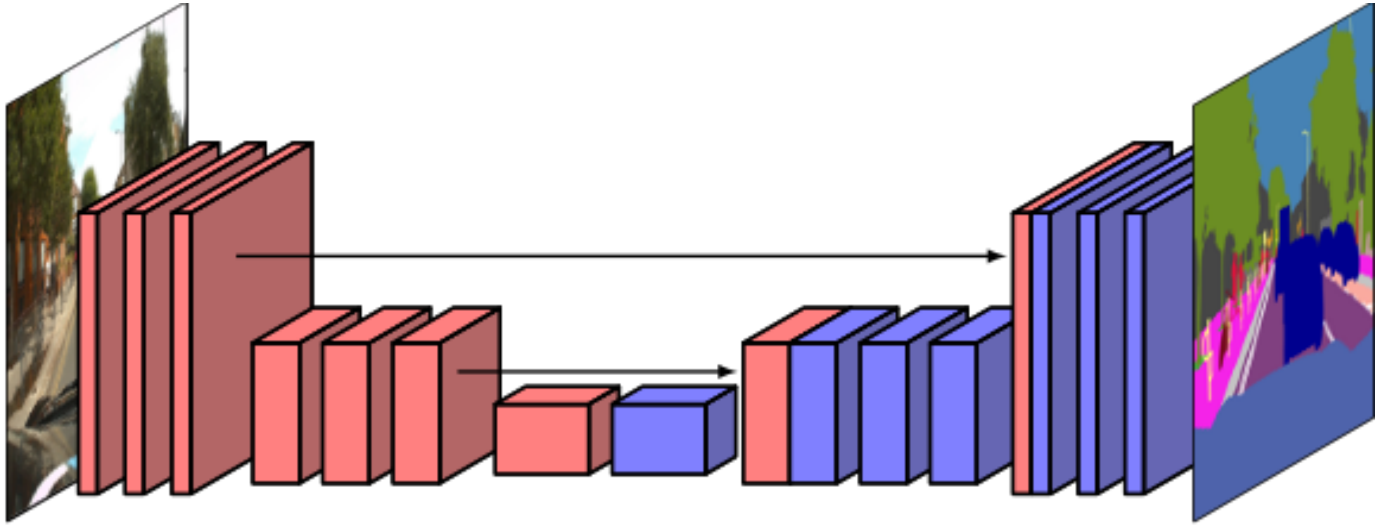
Upsampled input (after zero-insertion):

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Final output after transpose convolution:

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 2 & 2 & 3 & 0 \\ 0 & 3 & 0 & 0 \end{bmatrix}$$

U-Net



- *Corresponding* up and down sampling layers

Key Applications

- Autonomous driving
- Security
- Robotics

Example time

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

# Define a simple CNN using Conv2d
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        # Define a convolutional layer
        # Parameters: 1 input channel, 6 output channels, 3x3 kernel
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=3, stride=1, padding=1)
        # Another convolutional layer
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=3, stride=1, padding=1)
        # Define a fully connected layer
        self.fc1 = nn.Linear(16*7*7, 120)  # Adjusted for the 7x7 image size after pooling
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # First convolutional layer + ReLU + max pooling
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2)  # 2x2 max pooling
        # Second convolutional layer + ReLU + max pooling
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2)
        # Flatten for the fully connected layers
        x = x.view(-1, 16*7*7)  # Reshape the tensor for the fully connected layer
        # Fully connected layers
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# Create a random 1x1x28x28 image (1 batch, 1 channel, 28x28 image)
input_image = torch.randn(1, 1, 28, 28)

# Initialize the CNN and forward the input
model = SimpleCNN()
output = model(input_image)

print(output)
```