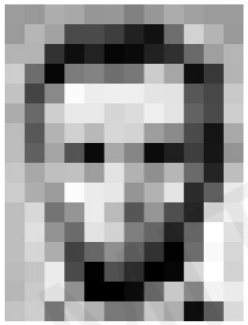
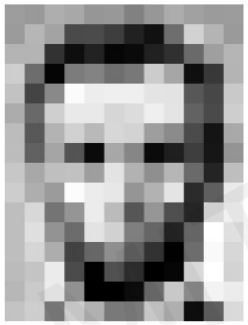
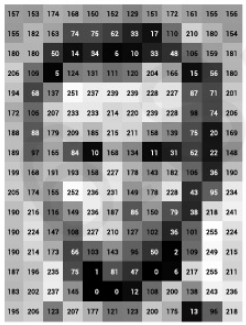
What Computers "See"

Images are Numbers

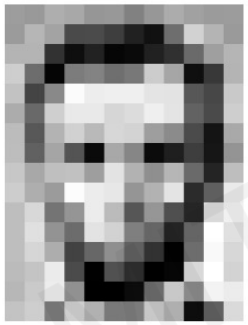


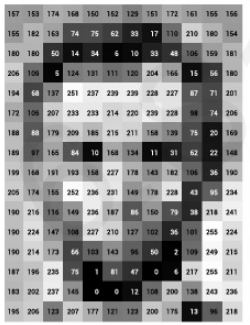
Images are Numbers



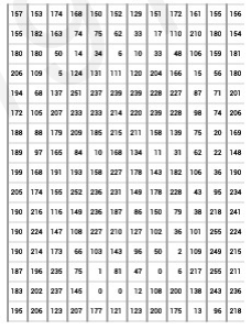


Images are Numbers



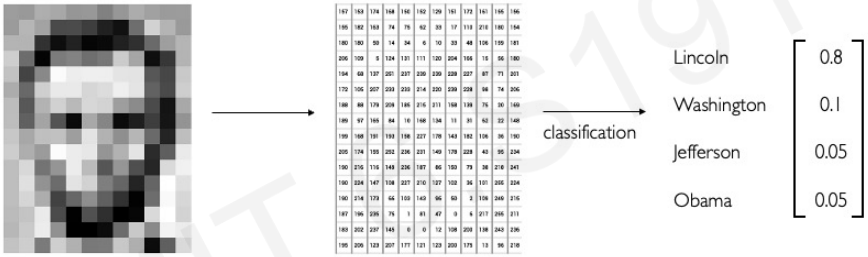


What the computer sees



An image is just a matrix of numbers [0,255]!
i.e., 1080x1080x3 for an RGB image

Tasks in Computer Vision



- **Regression**: output variable takes continuous value
- **Classification**: output variable takes class label. Can produce probability of belonging to a particular class

High Level Feature Detection

Let's identify key features in each image category

Nose,
Eyes,
Mouth

Wheels,
License Plate,
Headlights

Door,
Windows,
Steps

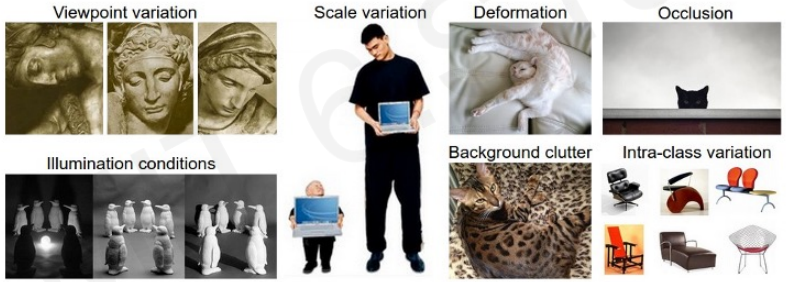
Manual Feature Extraction

Domain knowledge → Define features → Detect features to classify

Problems?

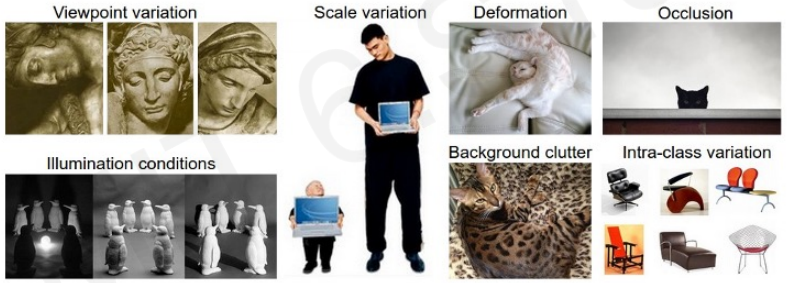
Manual Feature Extraction

Domain knowledge → Define features → Detect features to classify



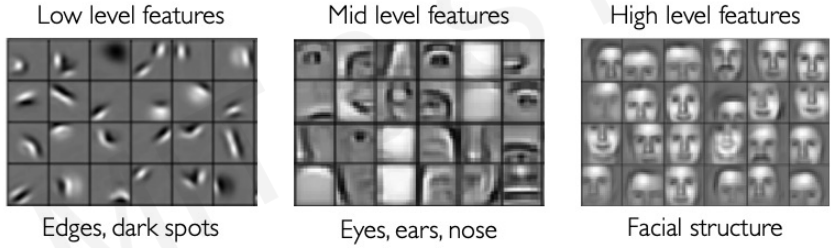
Manual Feature Extraction

Domain knowledge → Define features → Detect features to classify



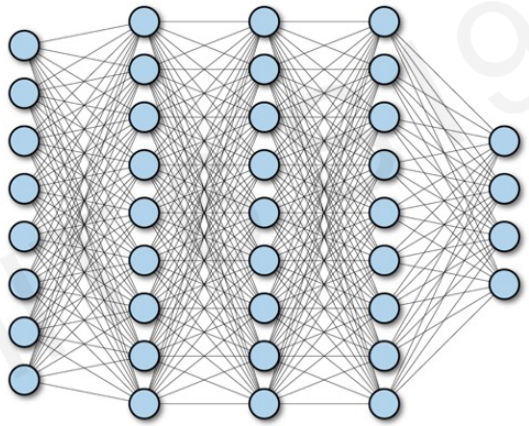
Learning Feature Representations

Can we learn a **hierarchy of features** directly from the data instead of hand engineering?



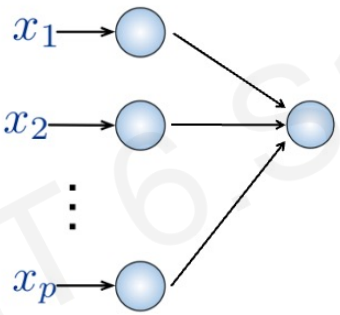
Learning Visual Features

Fully Connected Neural Network



Fully Connected Neural Network

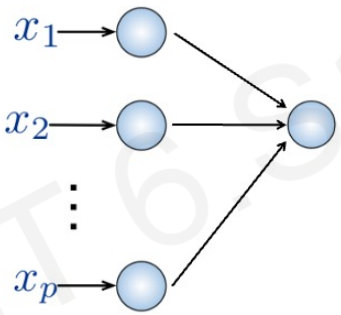
Input:
- 2D image
- Vector of pixel values



Fully Connected:
- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

Fully Connected Neural Network

Input:
- 2D image
- Vector of pixel values



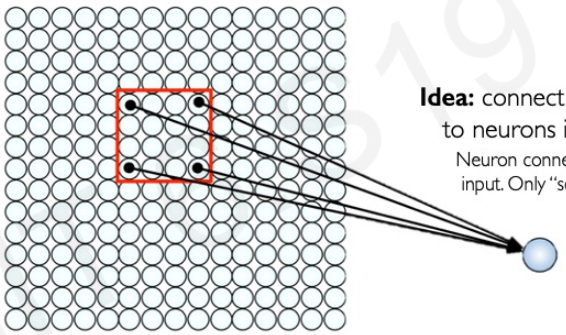
Fully Connected:
- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

How can we use **spatial structure** in the input to inform the architecture of the network?

Massachusetts Institute of Technology
MIT Introduction to Deep Learning
introtodeeplearning.com @MITDeepLearning
1/7/25

Using Spatial Structure

Input: 2D image.
Array of pixel values

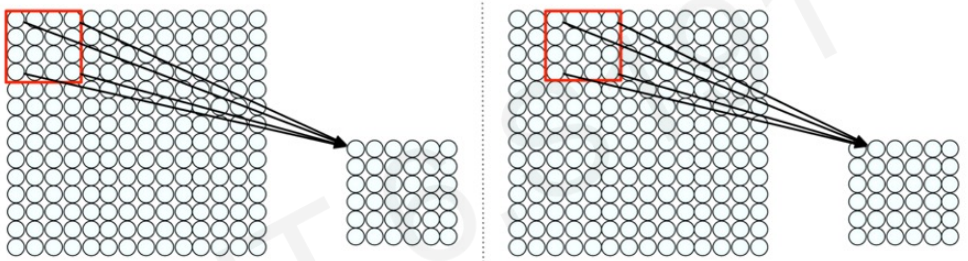


Idea: connect patches of input to neurons in hidden layer.

Neuron connected to region of input. Only "sees" these values.

Massachusetts Institute of Technology
MIT Introduction to Deep Learning
introtodeeplearning.com @MITDeepLearning
1/7/25

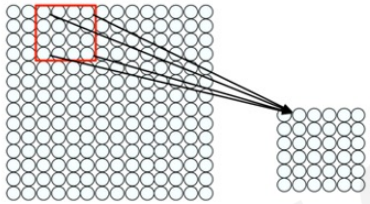
Using Spatial Structure



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?*

Massachusetts Institute of Technology
MIT Introduction to Deep Learning
introtodeeplearning.com @MITDeepLearning
1/7/25

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

Feature Extraction and Convolution
A Case Study

X or X?

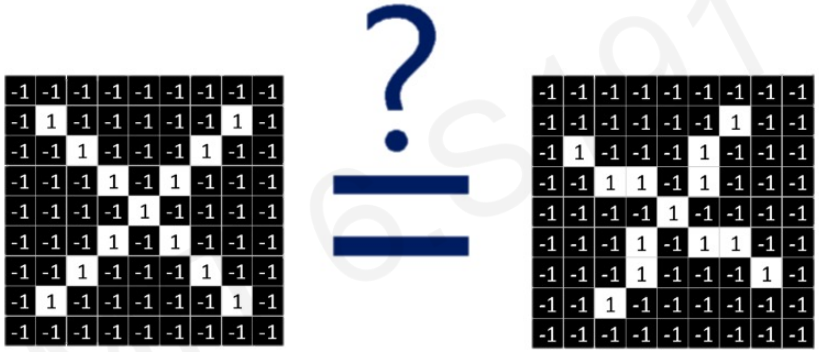
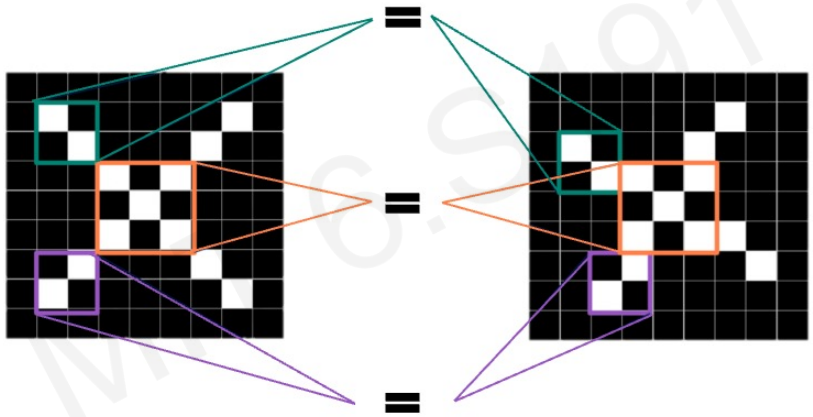
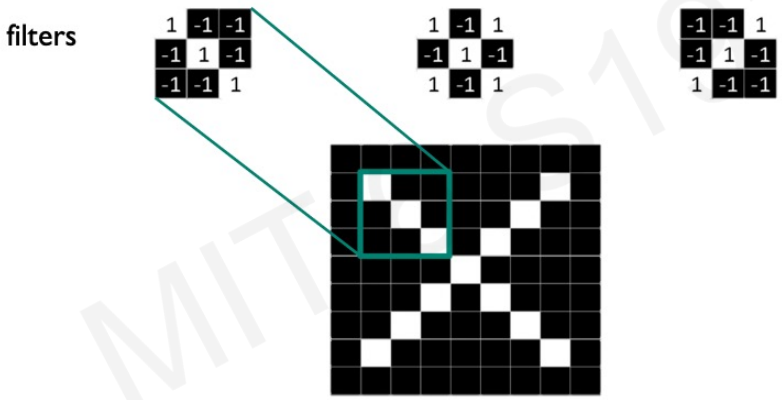


Image is represented as matrix of pixel values… and computers are literal!
We want to be able to classify an X as an X even if it's shifted, shrunk, rotated, deformed.

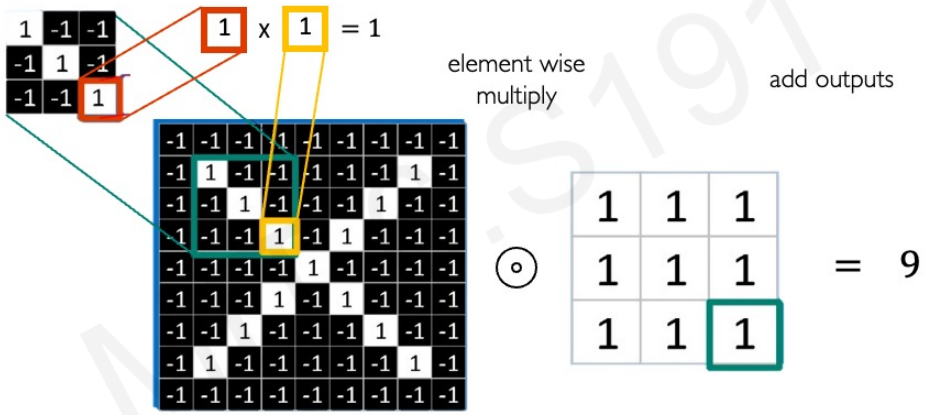
Features of X



Filters to Detect X Features



The Convolution Operation



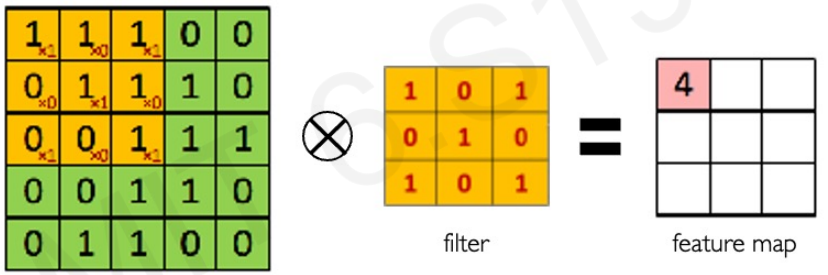
The Convolution Operation

Suppose we want to compute the convolution of a 5x5 image and a 3x3 filter:

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs…

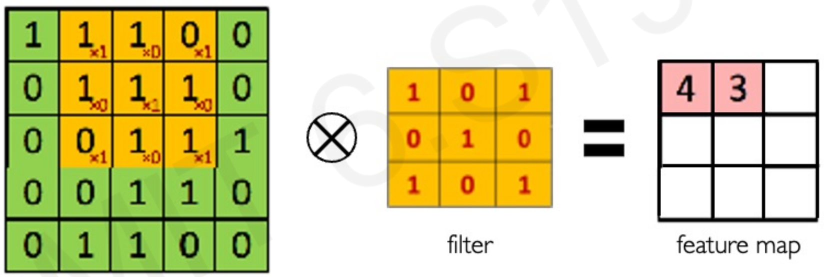
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



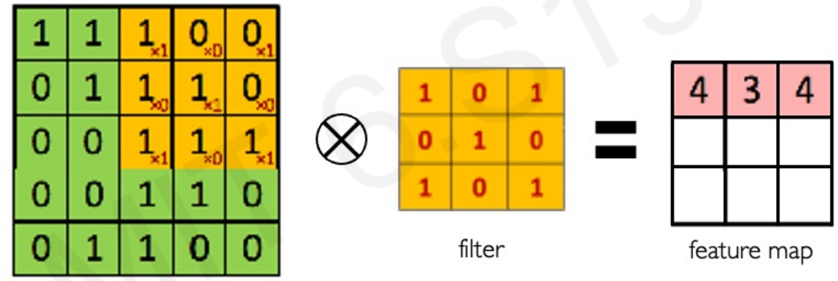
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



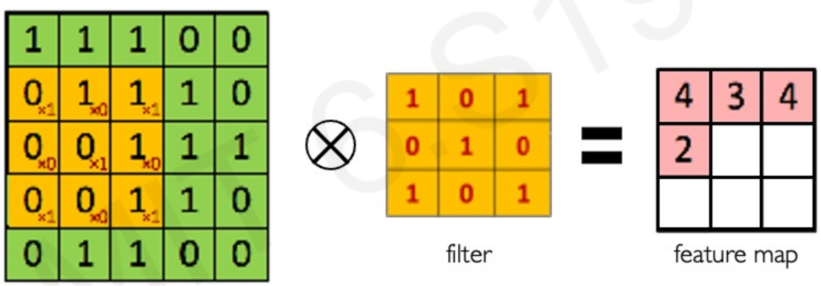
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



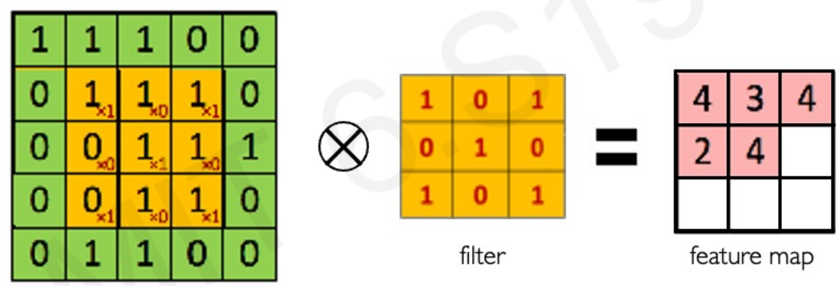
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



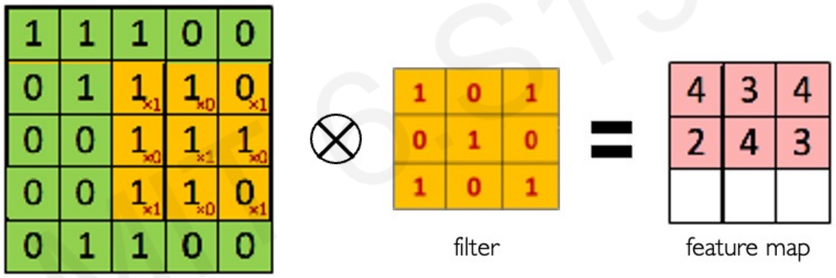
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



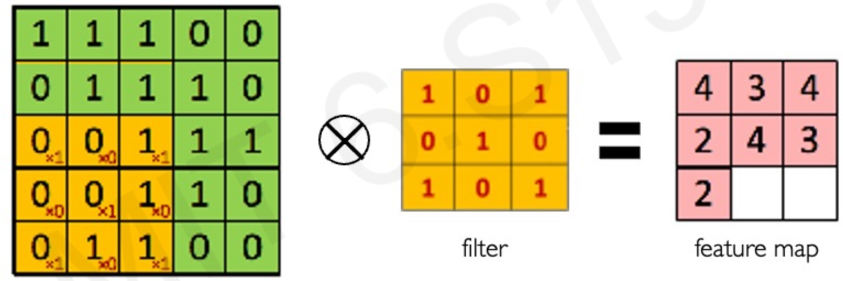
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



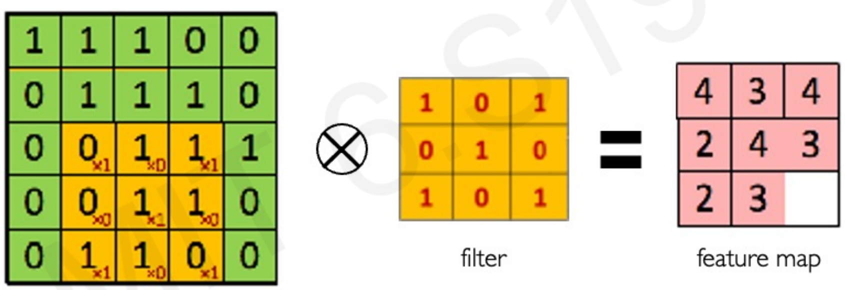
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



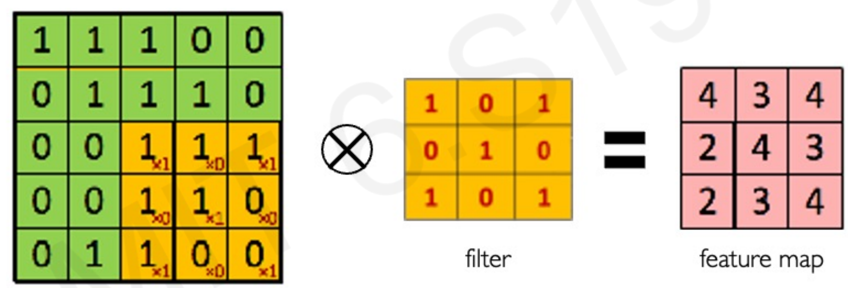
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



The Convolution Operation

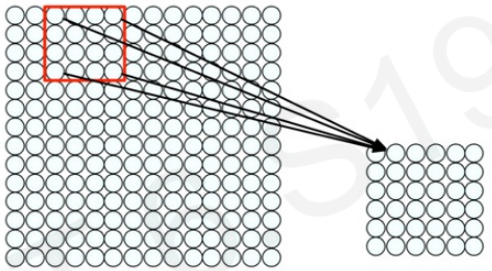
We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



Producing Feature Maps

Original | Sharpen | Edge Detect | "Strong" Edge Detect

Feature Extraction with Convolution



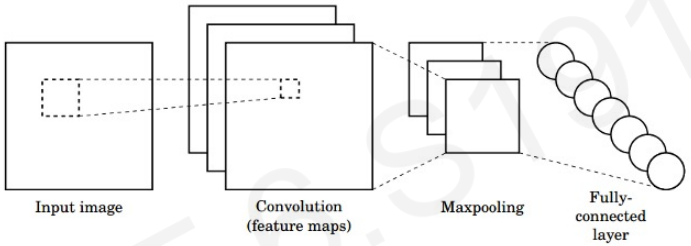
1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) **Spatially share** parameters of each filter

Convolutional Neural Networks (CNNs)

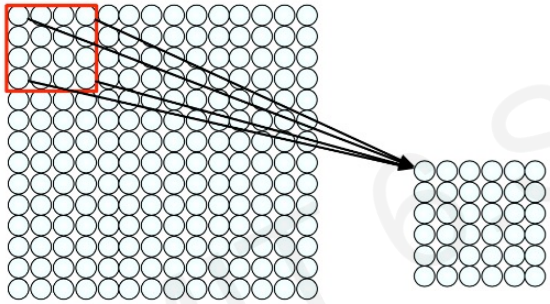
CNNs for Classification



`tf.keras.layers.Conv2D`	**1. Convolution**: Apply filters to generate feature maps.	`torch.nn.Conv2d`
`tf.keras.activations.*`	**2. Non-linearity**: Often ReLU.	`torch.nn.ReLU,…`
`tf.keras.layers.MaxPool2D`	**3. Pooling**: Downsampling operation on each feature map.	`torch.nn.MaxPool2d`

Train model with image data.
Learn weights of filters in convolutional layers.

Convolutional Layers: Local Connectivity

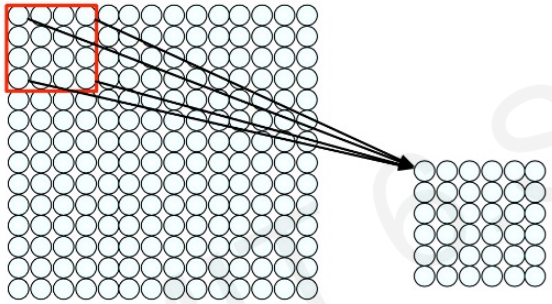


`tf.keras.layers.Conv2D`

`torch.nn.Conv2d`

For a neuron in hidden layer:
- Take inputs from patch
- Compute weighted sum
- Apply bias

Convolutional Layers: Local Connectivity



`tf.keras.layers.Conv2D`

`torch.nn.Conv2d`

For a neuron in hidden layer:
- Take inputs from patch
- Compute weighted sum
- Apply bias

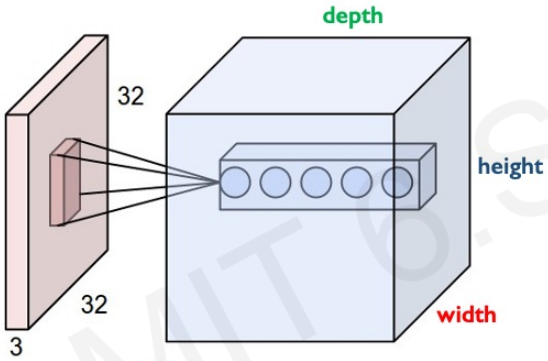
4x4 filter: matrix of weights w_{ij}

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

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:

$h \times w \times d$

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

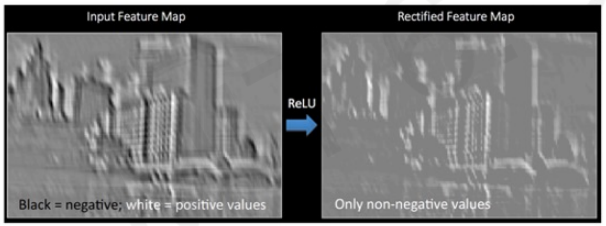
```
tf.keras.layers.Conv2D( filters=d, kernel_size=(h,w), strides=s )
```

```
torch.nn.Conv2d( in_channels=3, out_channels=d, kernel_size=(h,w), stride=s )
```

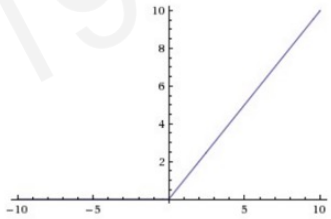
Need to specify input dimensionality!

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)

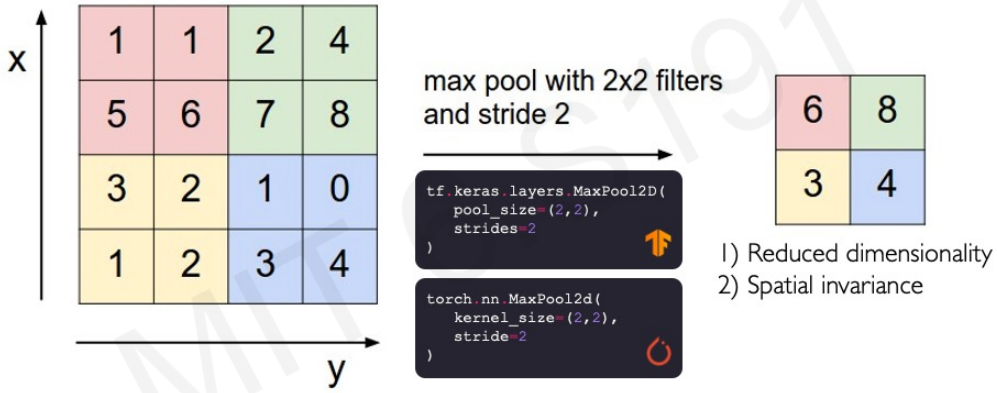


$$g(z) = \max(0, z)$$

`tf.keras.layers.ReLU`

`torch.nn.ReLU`

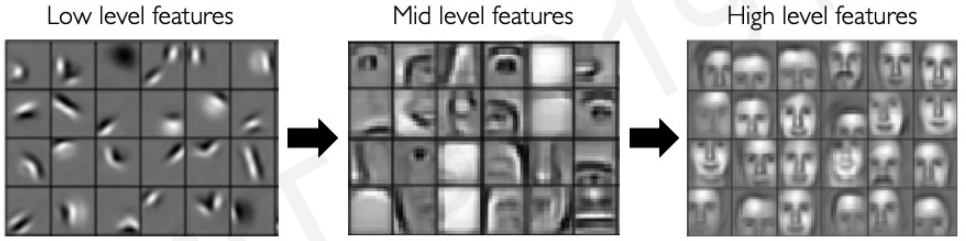
Pooling



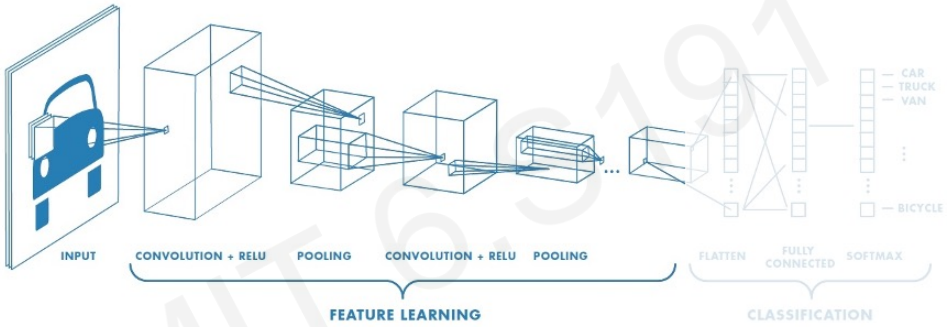
1) Reduced dimensionality
2) Spatial invariance

How else can we downsample and preserve spatial invariance?

Representation Learning in Deep CNNs

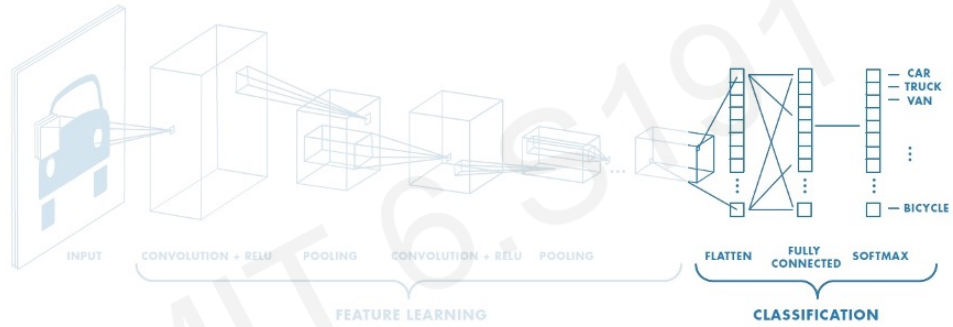


CNNs for Classification: Feature Learning



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

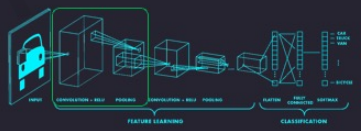


- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

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

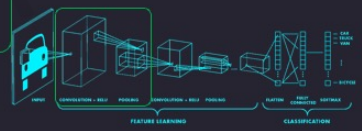
Putting it all together: CNN in TensorFlow

```python
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
```



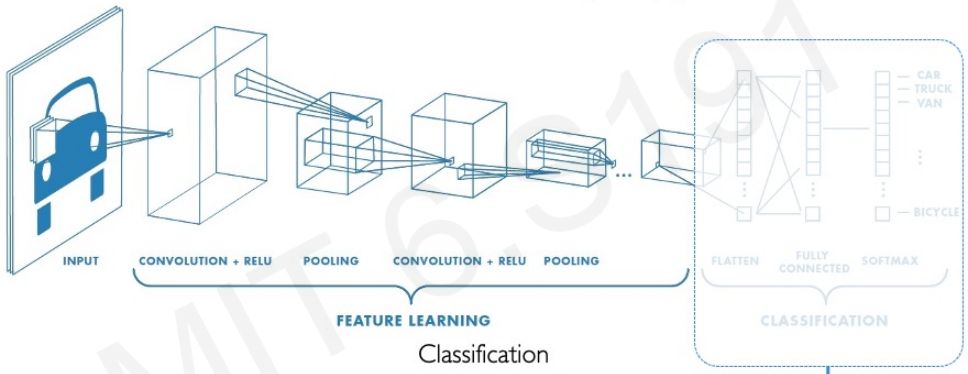
Putting it all together: CNN in PyTorch

```python
import torch

def generate_model():
    model = nn.Sequential([
        # first and second convolutional layer
        torch.nn.Conv2d(in_channels=3,out_channels=32, filter_size=3),
        torch.nn.ReLU(),
        torch.nn.MaxPool2d(kernel_size=2, stride=2),

        torch.nn.Conv2d(in_channels=32, out_channels=64, filter_size=3),
        torch.nn.ReLU(),
        torch.nn.MaxPool2d(kernel_size=2, stride=2),

        # fully connected classifier
        torch.nn.Flatten(),
        torch.nn.Linear(64*6*6, 1024), # flattened dim after 2 conv layers
        torch.nn.ReLU(),
        torch.nn.Linear(1024, 10),  # 10 outputs
    ])
    return model
```



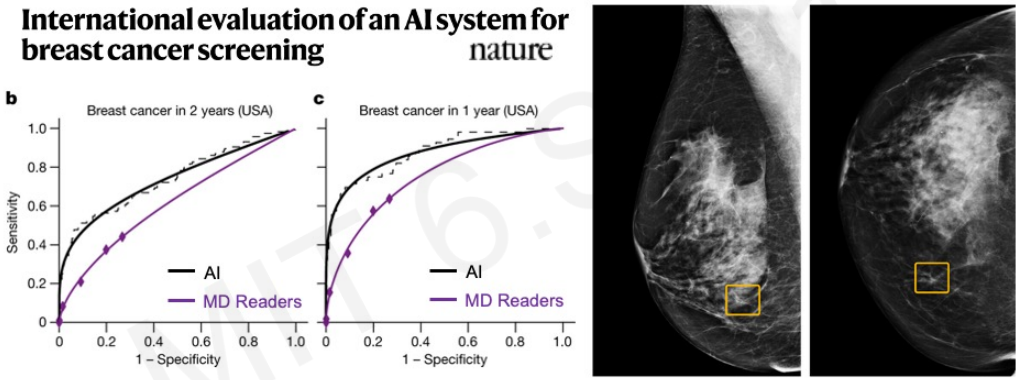
An Architecture for Many Applications

An Architecture for Many Applications



Classification
Object detection
Segmentation
Probabilistic control

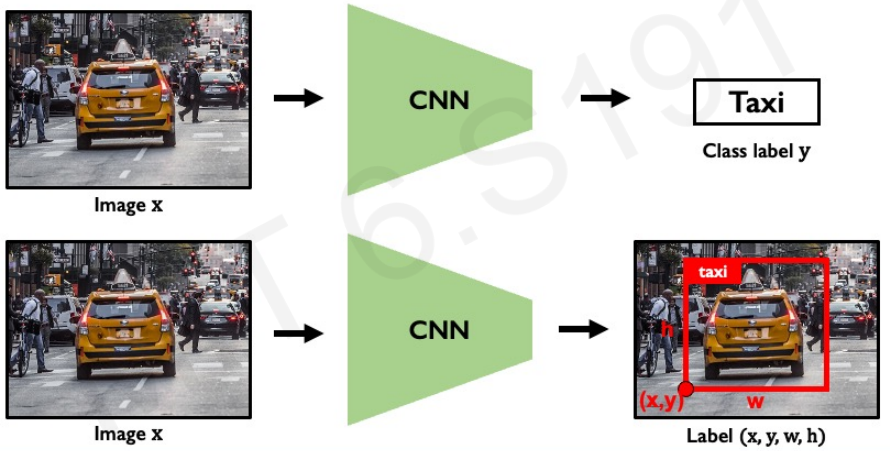
Classification: Breast Cancer Screening



CNN-based system outperformed expert radiologists at detecting breast cancer from mammograms

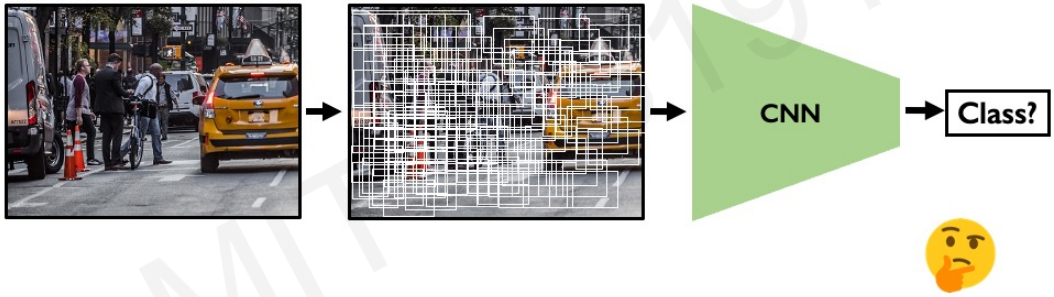
Breast cancer case missed by radiologist but detected by AI

Object Detection



Object Detection

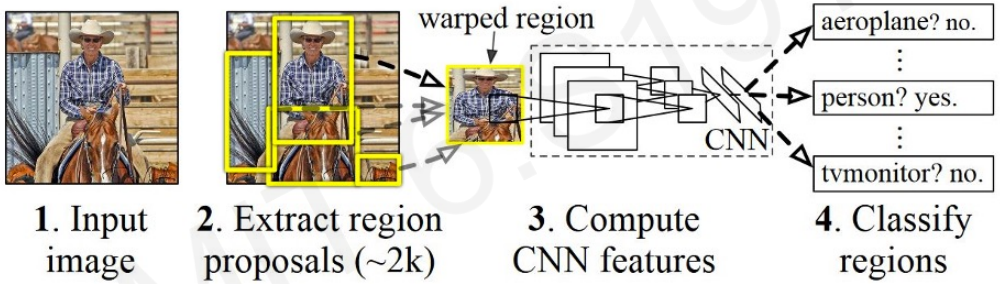
Naïve Solution to Object Detection



Problem: Way too many inputs! This results in too many scales, positions, sizes!

Object Detection with R-CNNs

R-CNN algorithm: Find regions that we think have objects. Use CNN to classify.



Problems: 1) Slow! Many regions; time intensive inference.
2) Brittle! Manually defined region proposals.

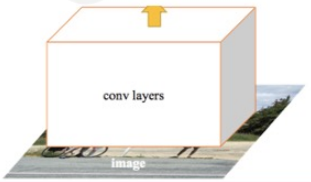
Faster R-CNN Learns Region Proposals

Classification of regions → object detection

Feature extraction over proposed regions

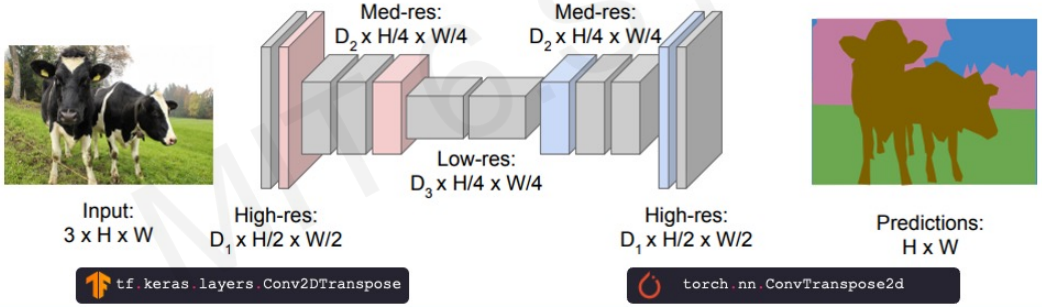
Region proposal network to learn candidate regions
Learned, data-driven

Image input directly into convolutional feature extractor
Fast! Only input image once!



Semantic Segmentation: Fully Convolutional Networks

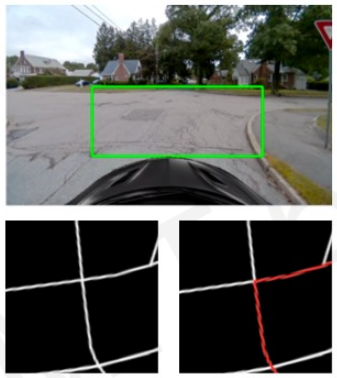
FCN: Fully Convolutional Network.
Network designed with all convolutional layers, with **downsampling** and **upsampling** operations



Continuous Control: Navigation from Vision

Raw Perception *I* (ex. camera)

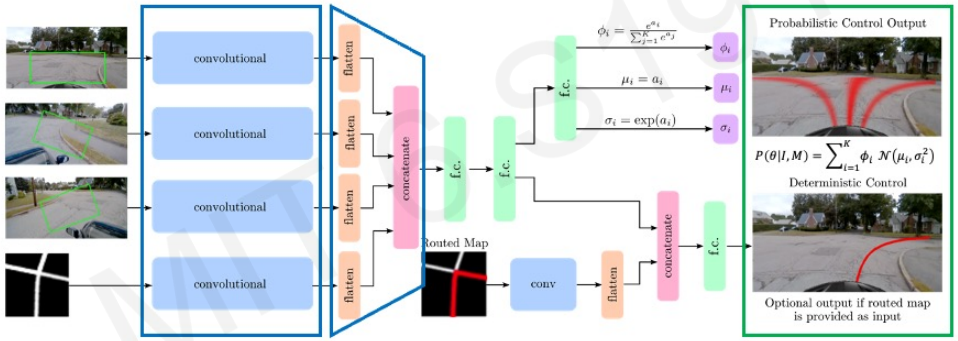
Coarse Maps *M* (ex. GPS)



Possible Control Commands

End-to-End Framework for Autonomous Navigation

Entire model is trained end-to-end **without any human labelling or annotations**



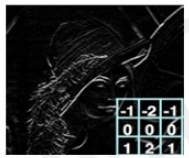
$$L = -\log(P(\theta|I,M))$$

Deep Learning for Computer Vision: Impact

Deep Learning for Computer Vision: Summary

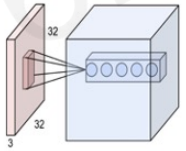
Foundations

- Why computer vision?
- Representing images
- Convolutions for feature extraction



CNNs

- CNN architecture
- Application to classification
- ImageNet



Applications

- Segmentation, image captioning, control
- Security, medicine, robotics