Deep Vision

"To know what is where by looking"

- Can also understand actions
- Can anticipate future events

Applications

- Facial recognition
- Self-Driving Cars

What do computers "see"?

- Images are just another raw representation of data
- Computers "see" a 2D matrix

Tasks

- 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

- identify key features such as nose, eyes, mouth, wheel, color, car, etc

Manual Feature Extraction

- If operator has domain knowledge of object in picture, they can point it out

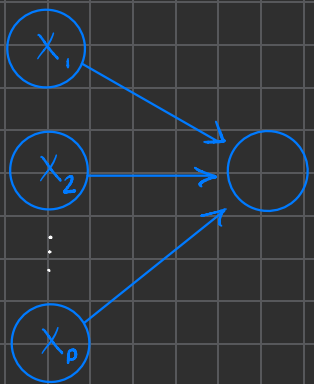
Domain Knowledge → Define Features → Detect Features to Classify

- How do you correctly identify unnamed features while not falsely detecting something?
- Neural networks <u>are</u> capable of learning hierarchical set of features

Using a Fully Connected Network

Input:

- 2D image
- Vector of pixel values

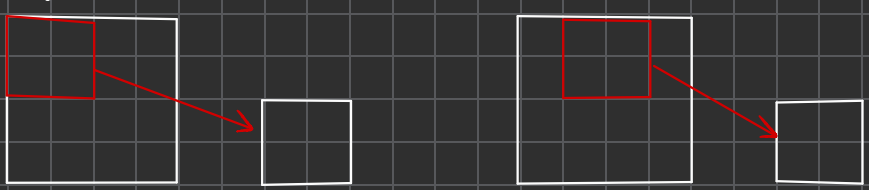


Fully Connected:

- Connect neuron in hidden layer to all neurons in input layer
- No spatial info!
- Many parameters

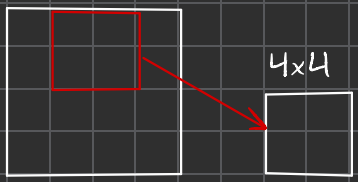
How can we maintain spatial info?

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 4×4 with 16 different weights
- Apply this same filter to 4×4 patches in input
- Shift by 2 pixels for next patch

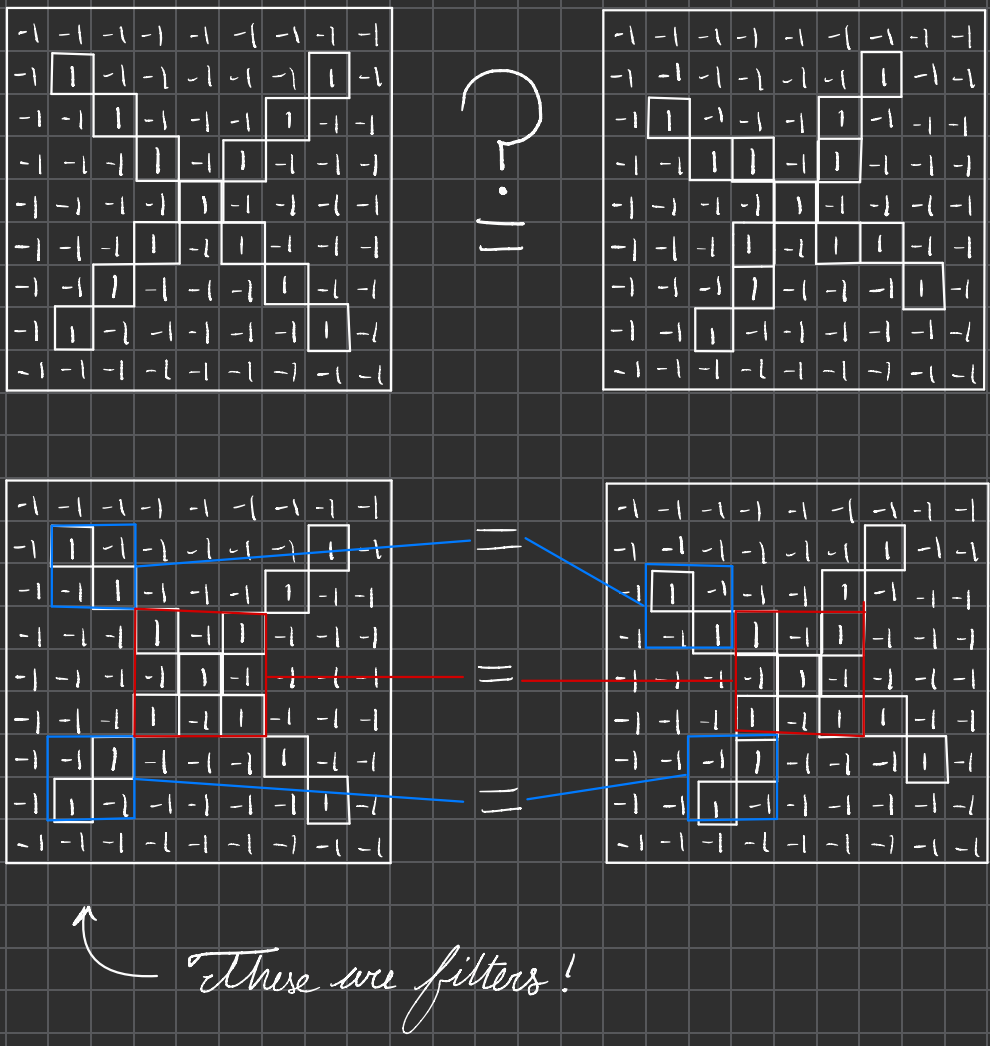
This "patchy" operation is called <u>convolution</u>

1.) Apply a set of weights (a filter) to extract local features

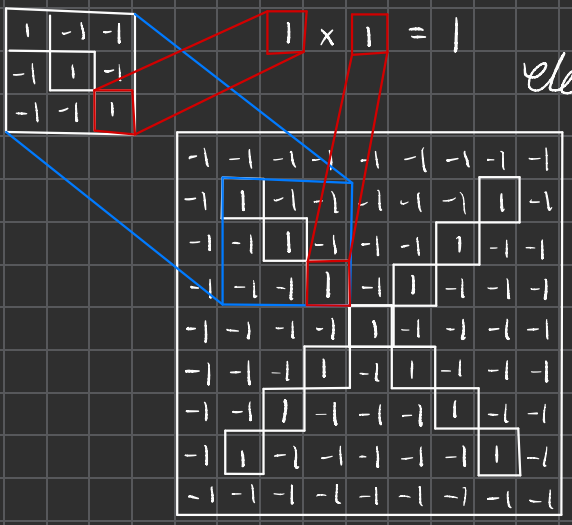
2.) Use multiple filters to extract different features

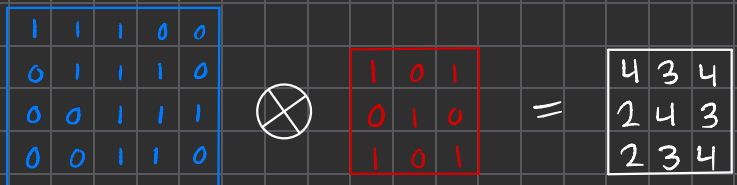
3.) Spatially share parameters of each feature

Why does convolution work?



The Convolution Operation





filter feature map

Feature maps show where features are in an image

Can we "learn" filters?

Can we build convolutional layers?

Convolutional Neural Networks (CNNs)

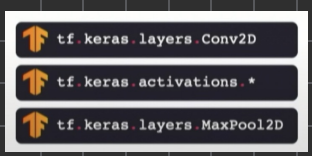
CNNs for classification

input image → Convolution → Maxpooling → Fully-Connected Layer

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

Convolutional Layers: Local Connectivity

For a neuron in a hidden layer:

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

} Convolution instead of dot product

Every single neuron only sees one patch

4×4 filter: matrix of weights w_{ij}

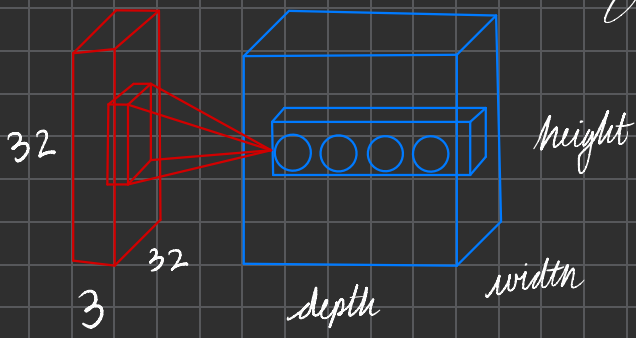
$$\sum_{i=1}^{4} \sum_{j=1}^{4} w_{ij} x_{i+p, j+q} + b$$

1.) apply a window of weights

2.) Compute linear combinations

3.) Activate with non-linear functions

CNNs: Spatial Arrangement of Output Volume



Layer dimensions:

$h \cdot w \cdot d$

where h and w are spatial dimensions

d (depth) = # of filters

Stride: Filter step size

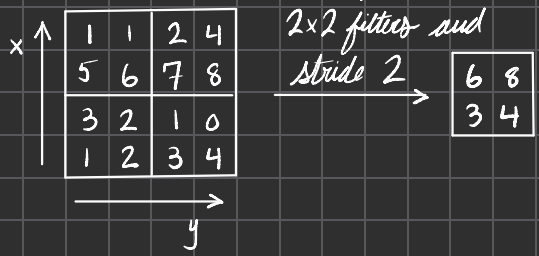
Receptive Field: Locations in input image that a node path is connected to.

```
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 with 0

Pooling




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

1.) Reduced dimensionality

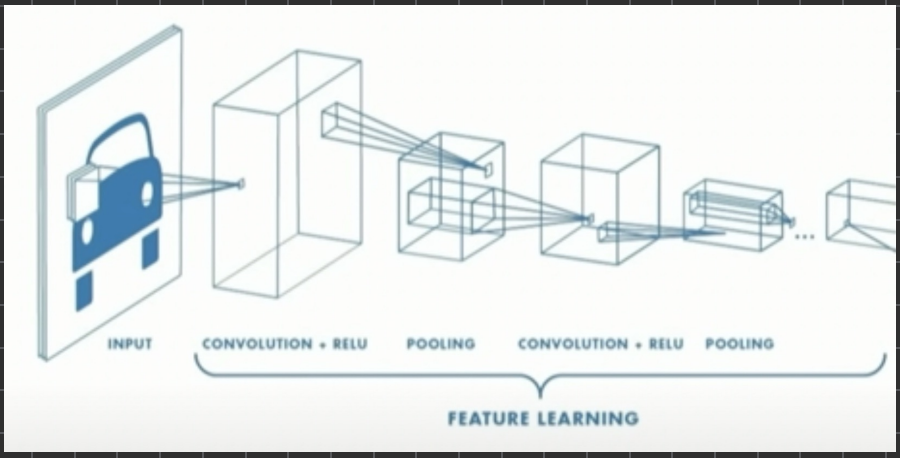
2.) Spatial invariance

Representation Learning in Deep CNNs

Low-Level Features → Mid-Level Features → High-Level Features

Convolution Layer 1 → Convolution Layer 2 → Convolution Layer 3

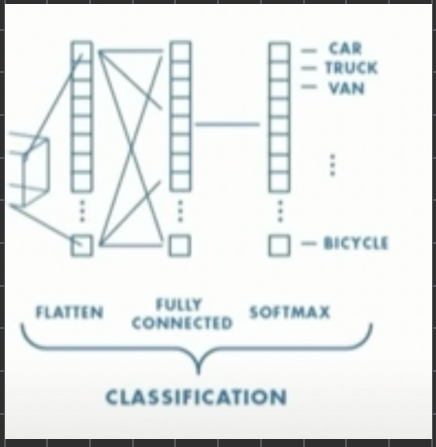
CNNs for Classification: Feature Learning



1.) Learn features in input image through convolution

2.) Introduce non-linearity through activation function

3.) Reduce dimensionality and preserve spatial invariance with pooling



1.) Convolution + pooling layers output high level features of input

2.) Fully connected layer uses these features for classifying input

3.) Express output as probability

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

End to End Network in Code

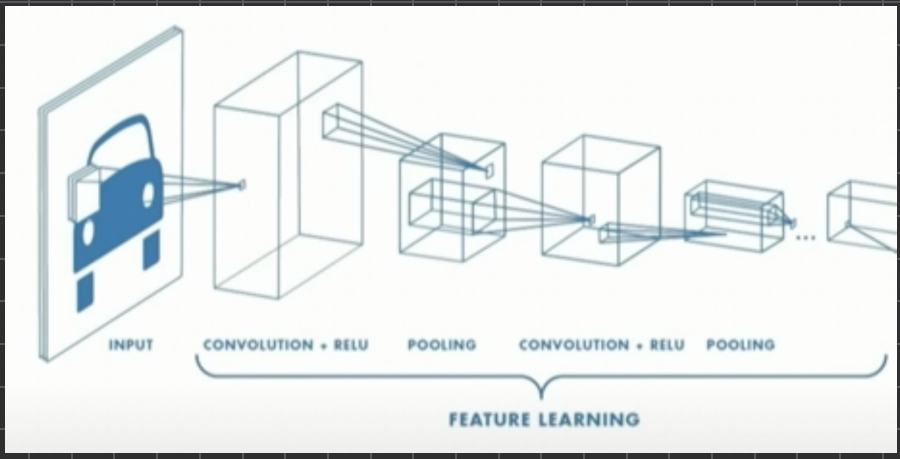
```python
import tensorflow as tf

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
```

An Architecture for Many Applications



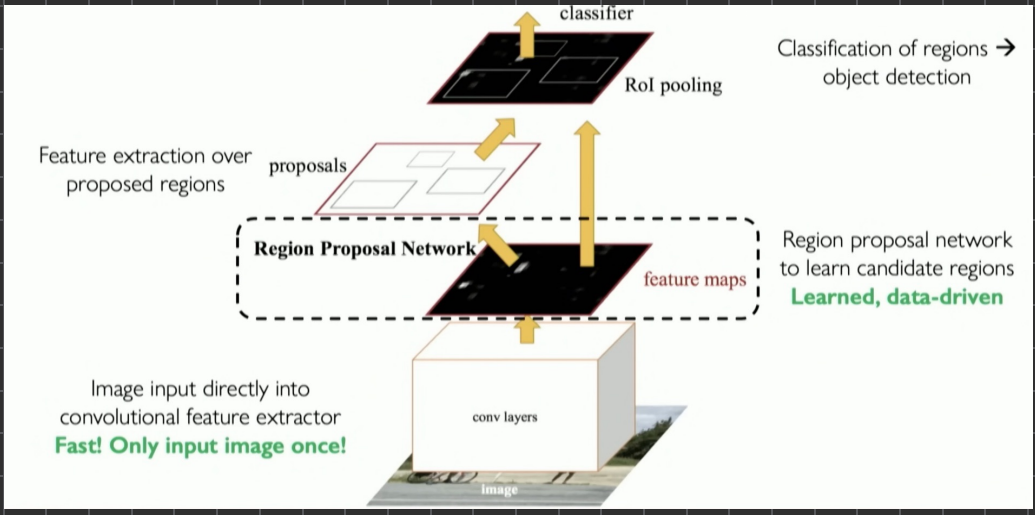
- Very flexible
- Can be used for:
  - Classification
  - Object detection
  - Segmentation
  - Probabilistic control

Classification: Breast Cancer Screening

Object Detection

- Find where an object is in an image
- Need to draw bounding boxes for arbitrarily many objects
- Naive solution: Random boxes in random places
  - Way too impractical
- R-CNN: Find regions that we think have objects
  1.) Input image
  2.) Extract region proposals (~2K)
  3.) Compute CNN features
  4.) Classify regions
  - Still slow and brittle

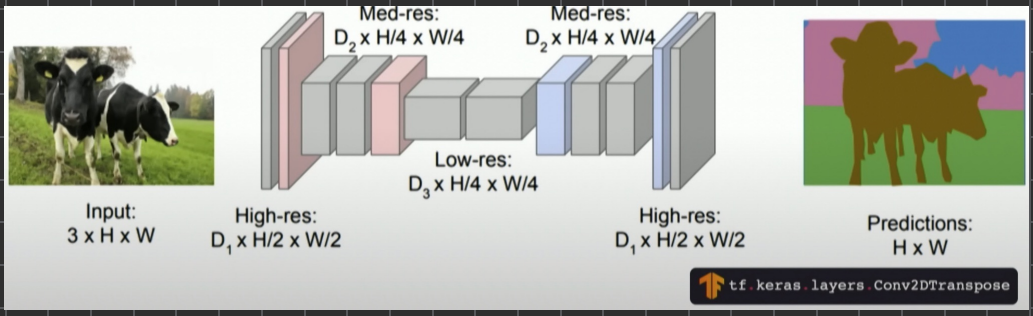
Faster R-CNN Learns Region Proposals



Semantic Segmentation: Fully Convolutional Networks

FCN: Fully Convolutional Network

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

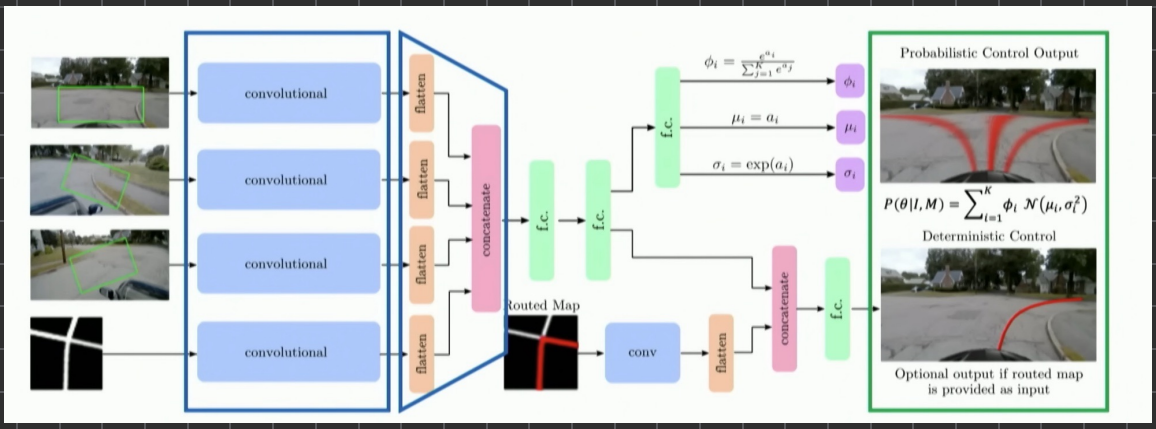


Semantic Segmentation

Good for:

- Biomedical Image Analysis
- Navigation from vision
  - Raw perceptions
  - Coarse maps
  - Possible control commands

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



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