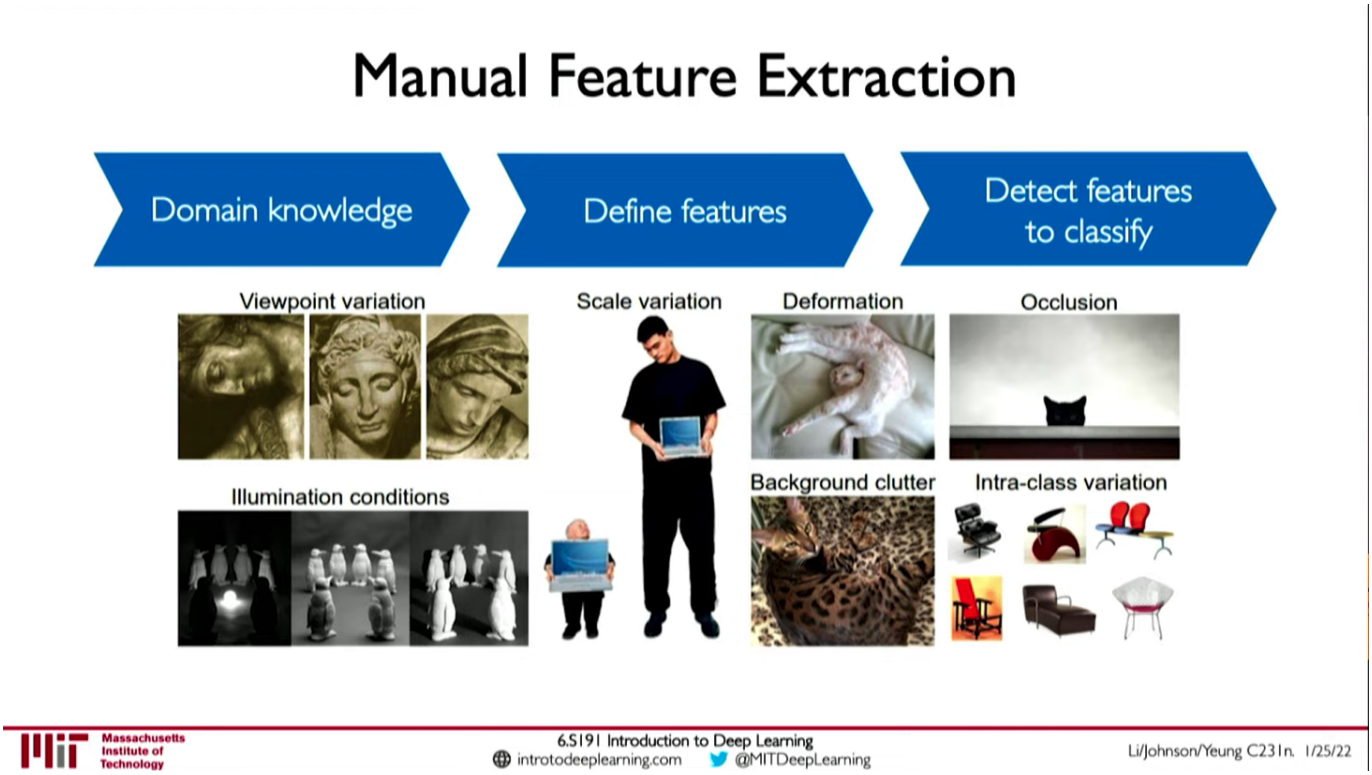
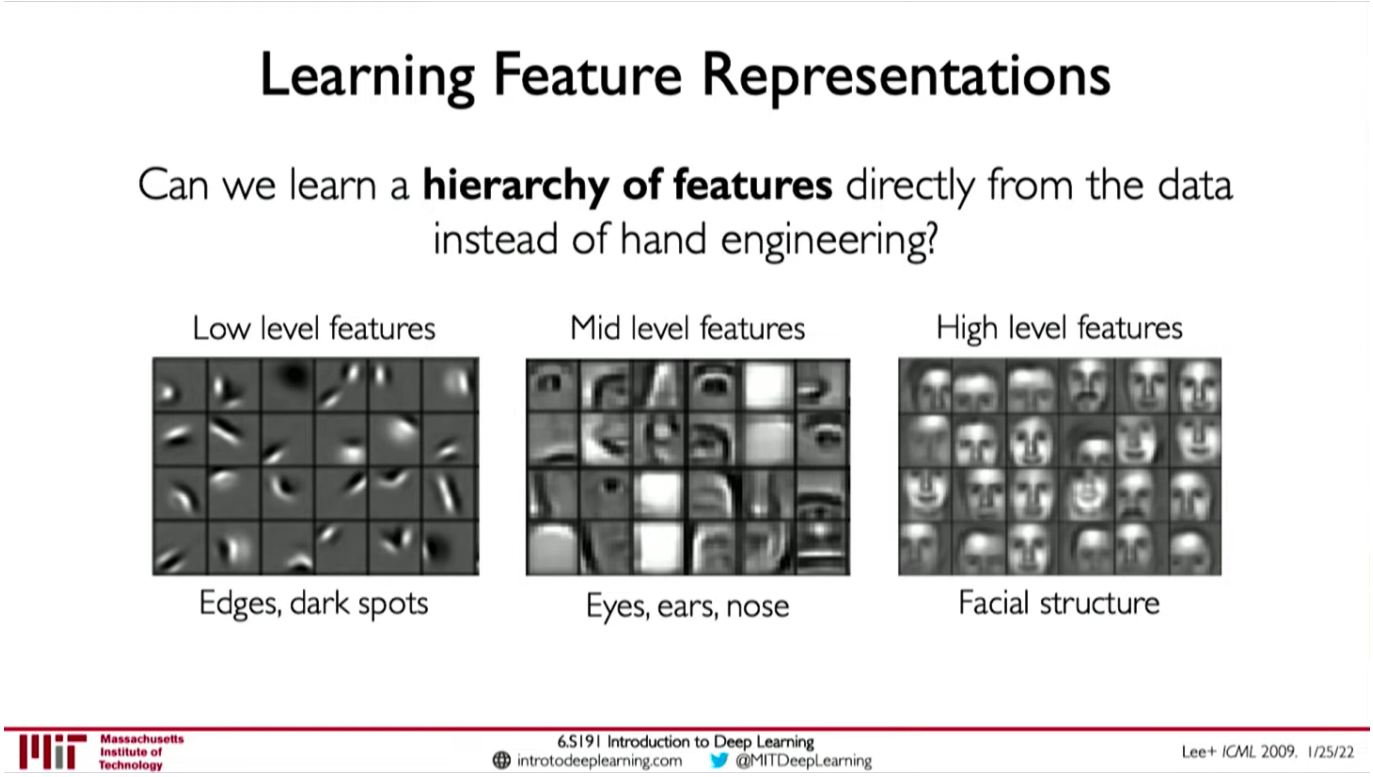
Convolutional Neural Network

Convolutional Neural Networks (CNN) *are a neural network architecture developed for computer vision tasks (i.e., regression and classification) that utilizes **kernel convolutions***.

Feature Extraction

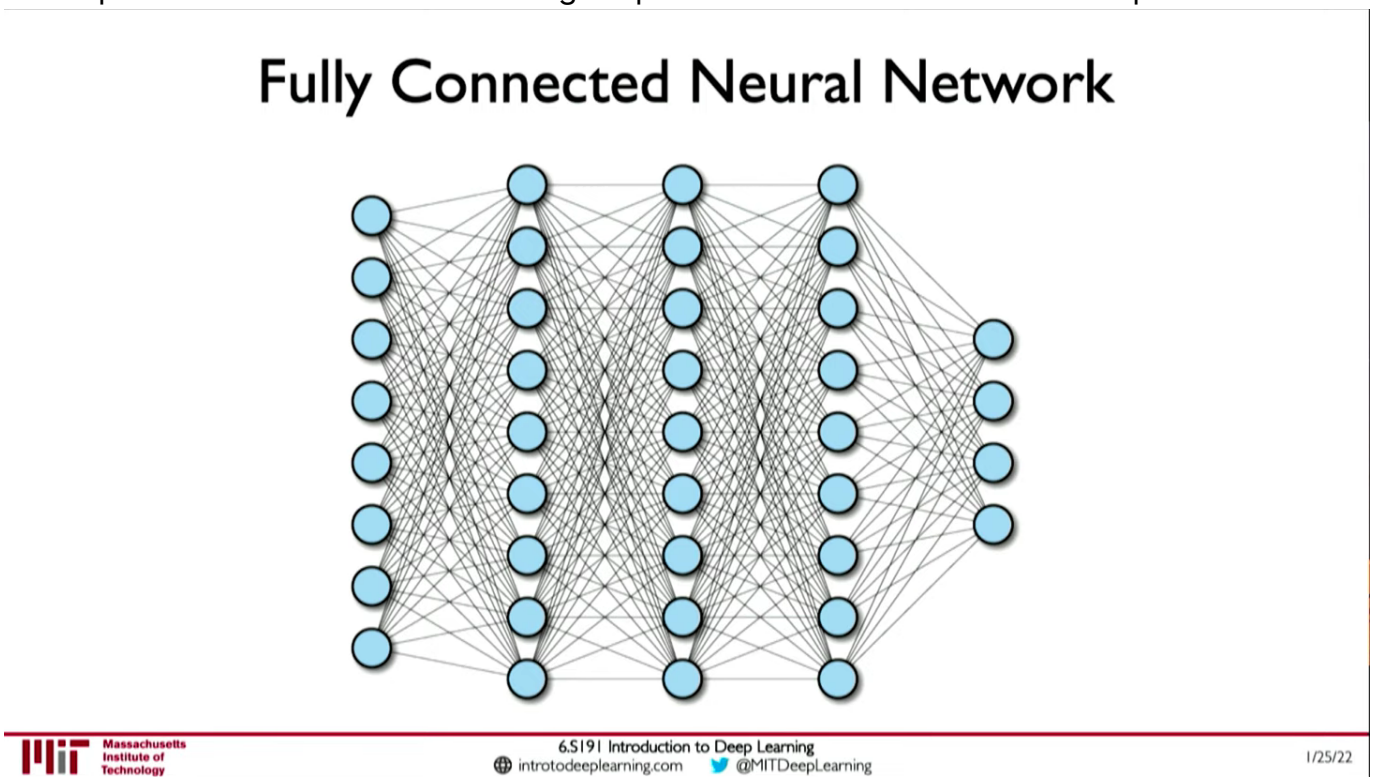


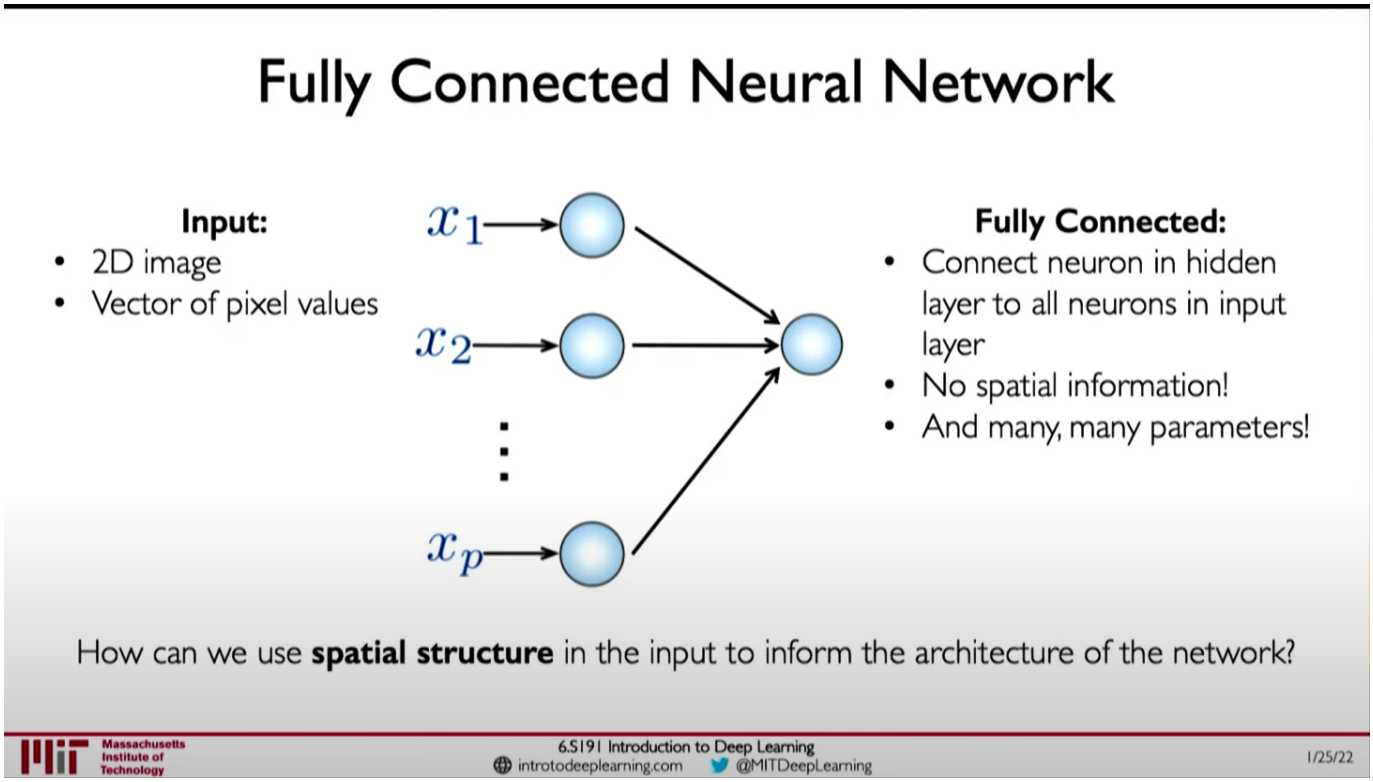
Learning Feature Representations



Fully Connected Neural Network

The input to this network is a 2D image represented as a column vector of pixels





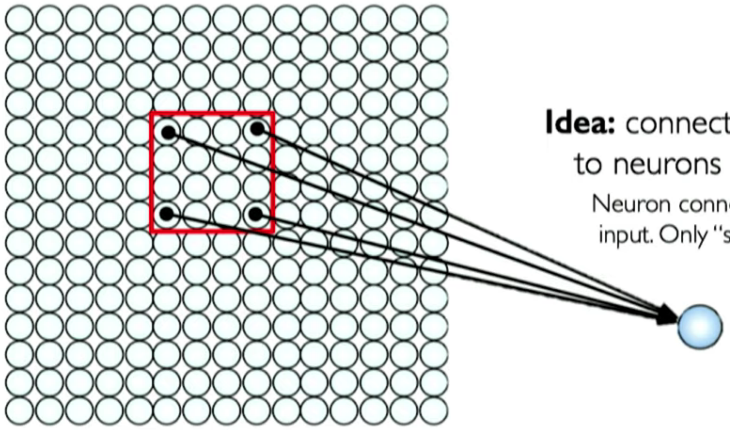
Convolution Operation

The convolution operation is a mathematical operation that performs a **moving weighted average** (cumulative average or sum of elementwise multiplications between a filter ($f^{n\times n}$) and pixels of a 2D image ($x^{l\times l}$). The filter can be larger than the image, but in practice its size is a subset of the image's (e.g., $f^{4\times 4} * x^{8\times 8}$):

$$\sum_{i=1}^{4}\sum_{j=1}^{4} f_{ij}x_{ij}$$

Using Spatial Structure

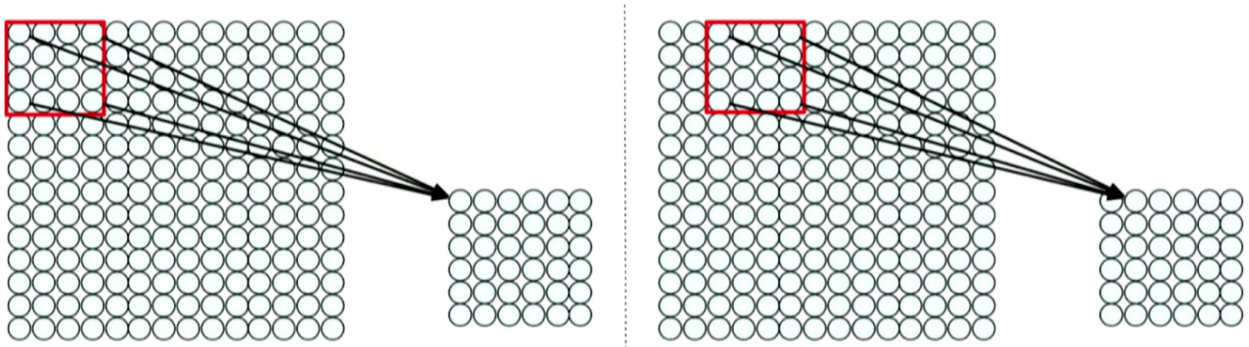
Input: 2D image.
Array of pixel values



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

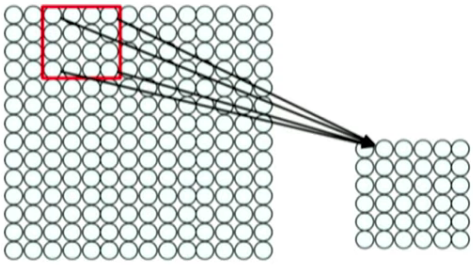
Neuron connected to region of input. Only "sees" these values.

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

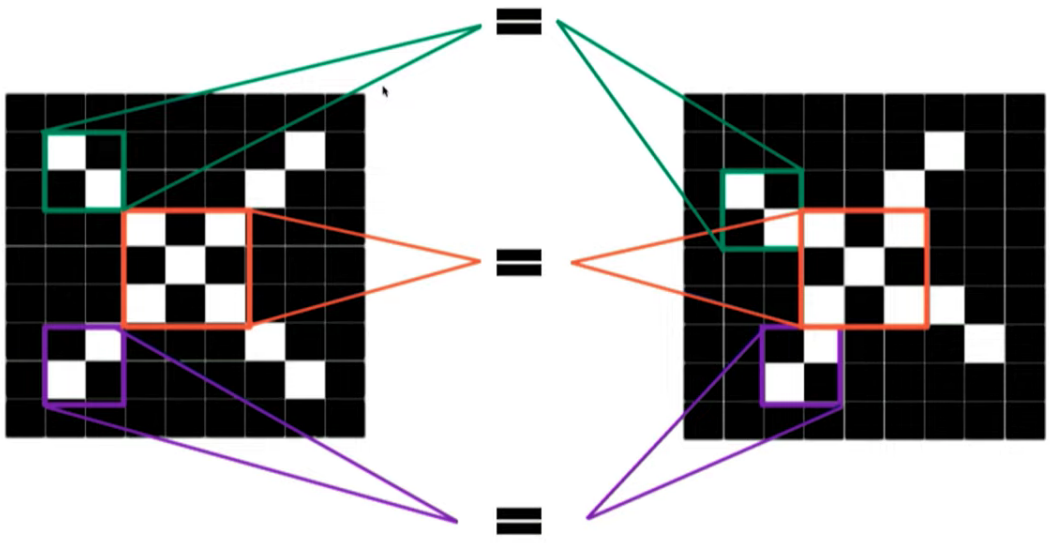
This "patchy" operation is **convolution**

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

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

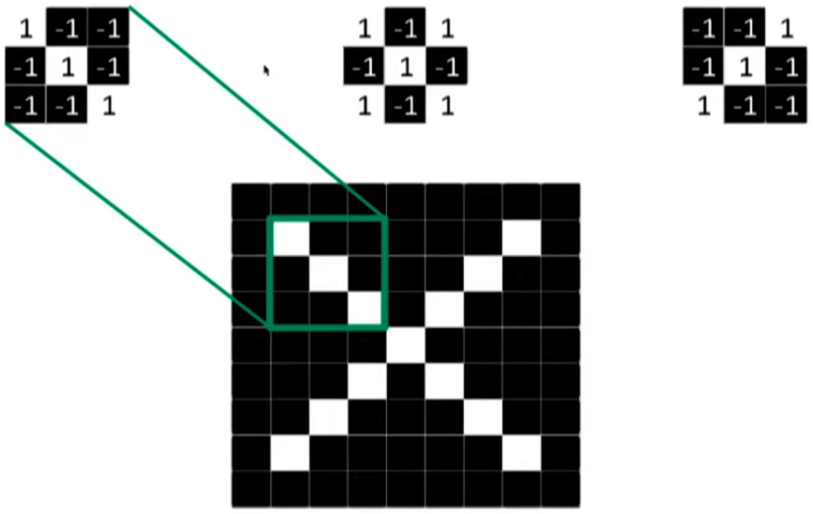
3) **Spatially share** parameters of each filter

Features of X



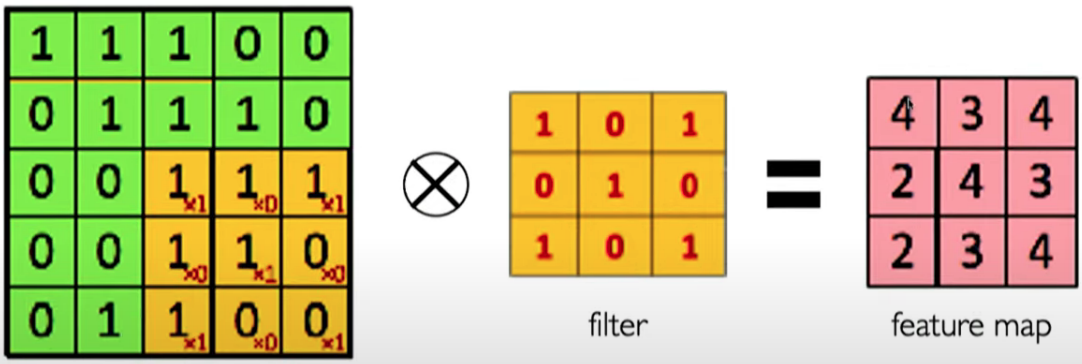
Filters to Detect X Features

filters

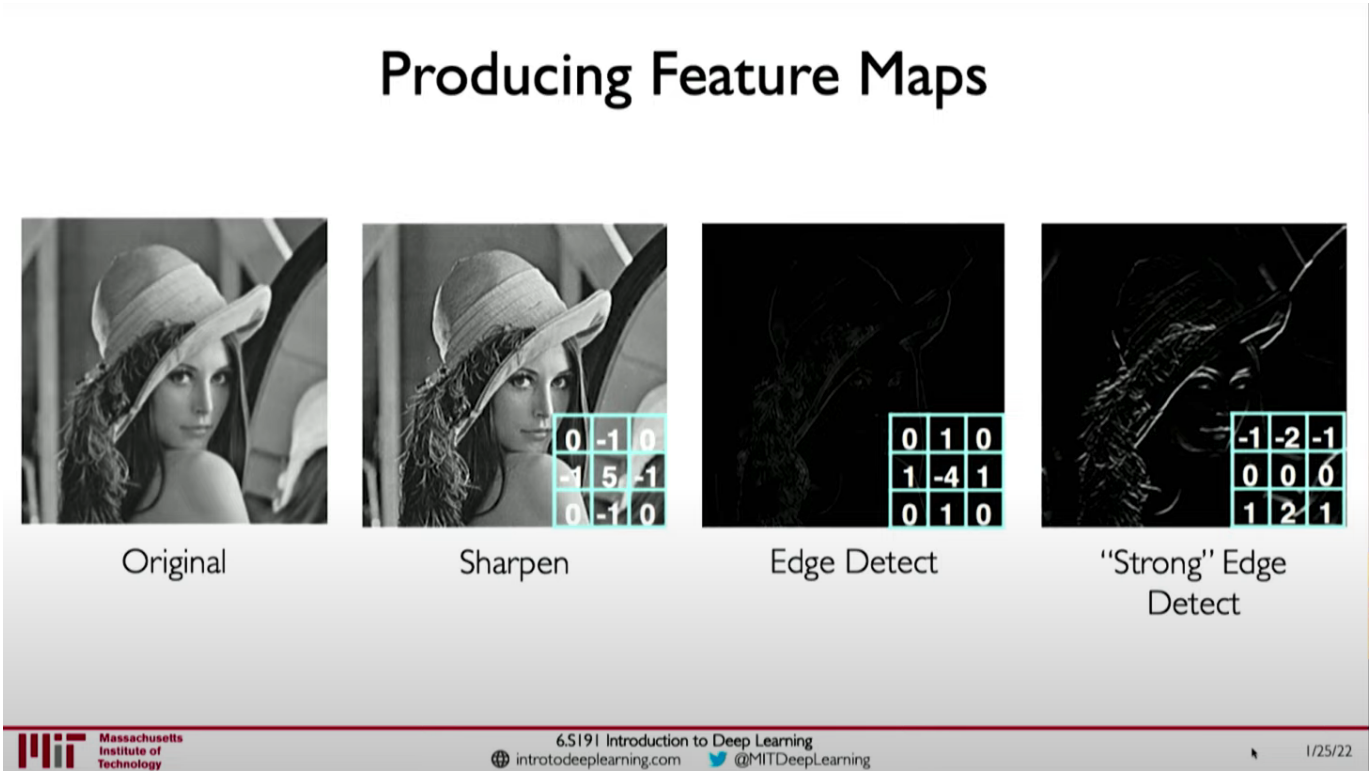


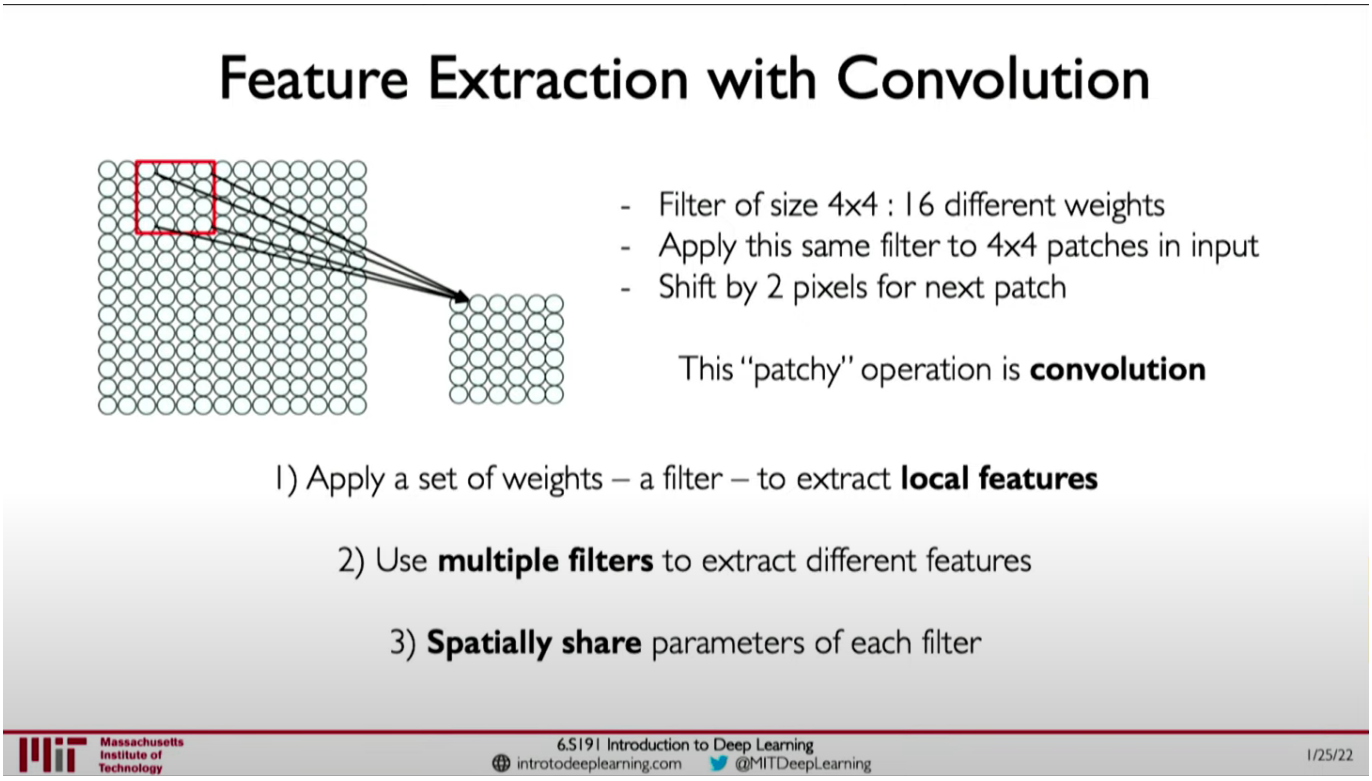
The Convolution Operation

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



filter feature map

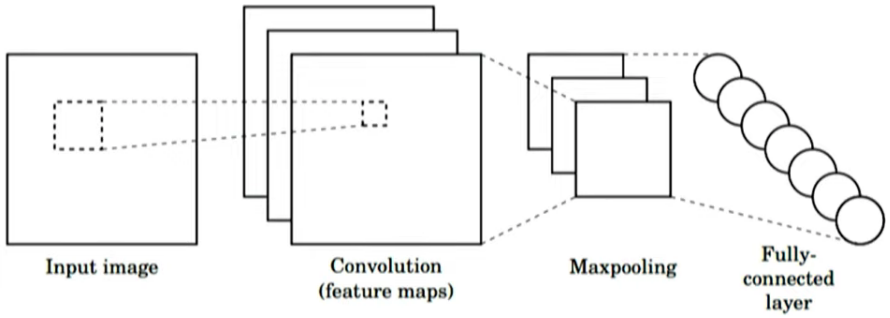




Building CNNs

Goal: Learn features from image data and use said features to identify certain properties of images of a paticular type to inform some vision task.

CNNs for Classification

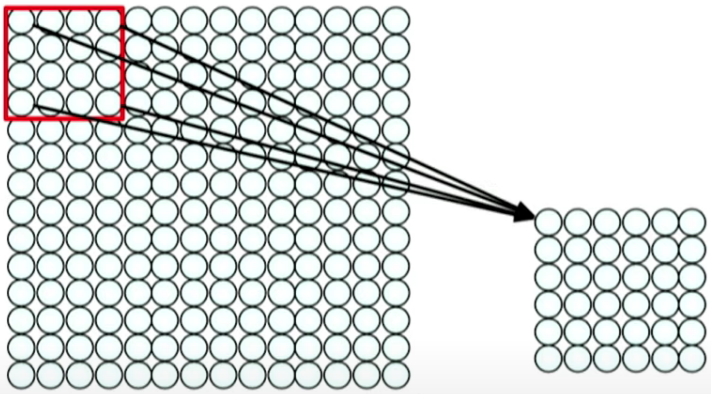


1. **Convolution**: Apply filters to generate feature maps. `tf.keras.layers.Conv2D`
2. **Non-linearity**: Often ReLU. `tf.keras.activations.*`
3. **Pooling**: Downsampling operation on each feature map. `tf.keras.layers.MaxPool2D`

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

Building 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

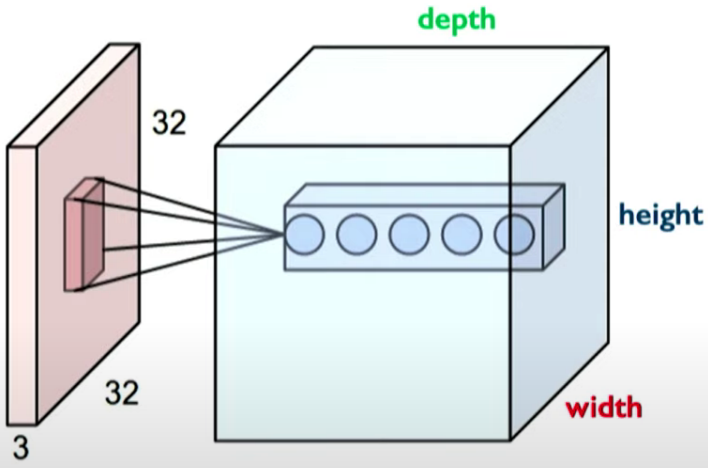
4x4 filter: matrix of weights w_{ij}

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

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

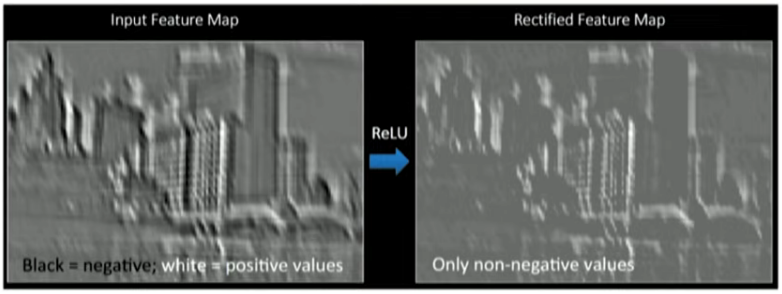
Receptive Field:
Locations in input image that a node is path connected to

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

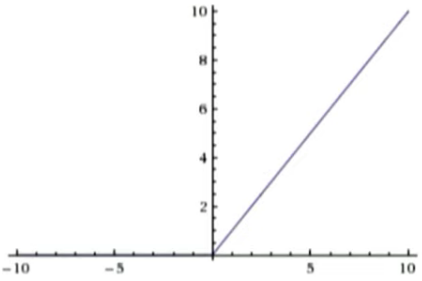

Non-Linearity

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)

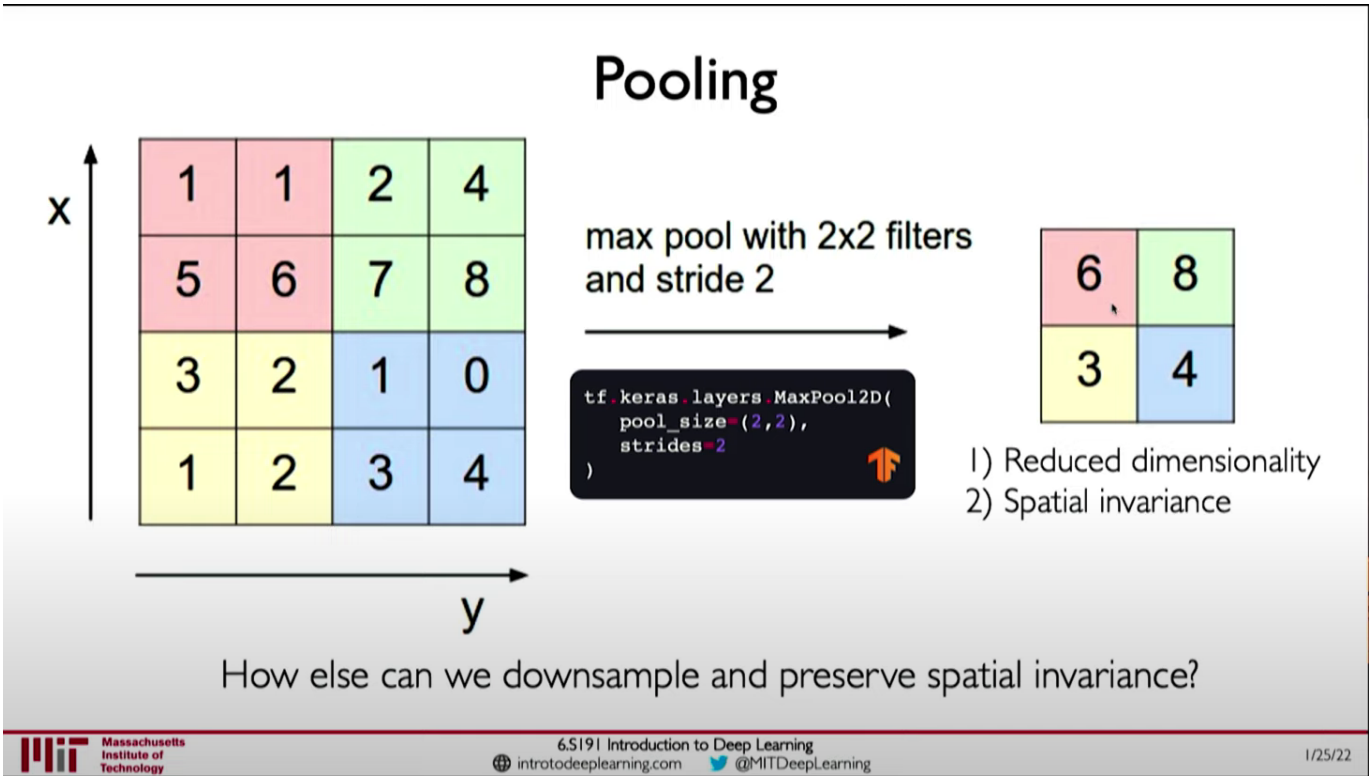


$$g(z) = \max(0, z)$$

```
tf.keras.layers.ReLU
```

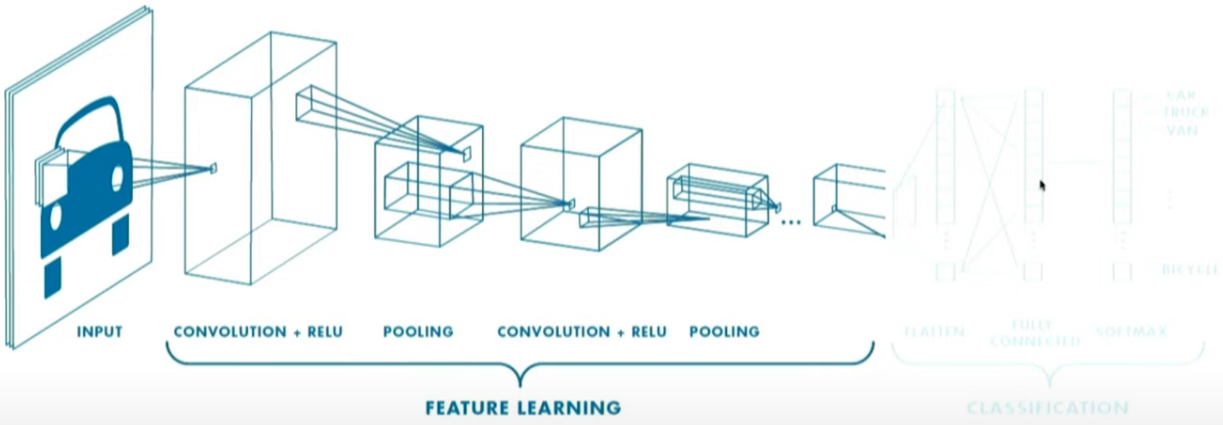

Pooling

A way to donsample and preserve spatial invariance



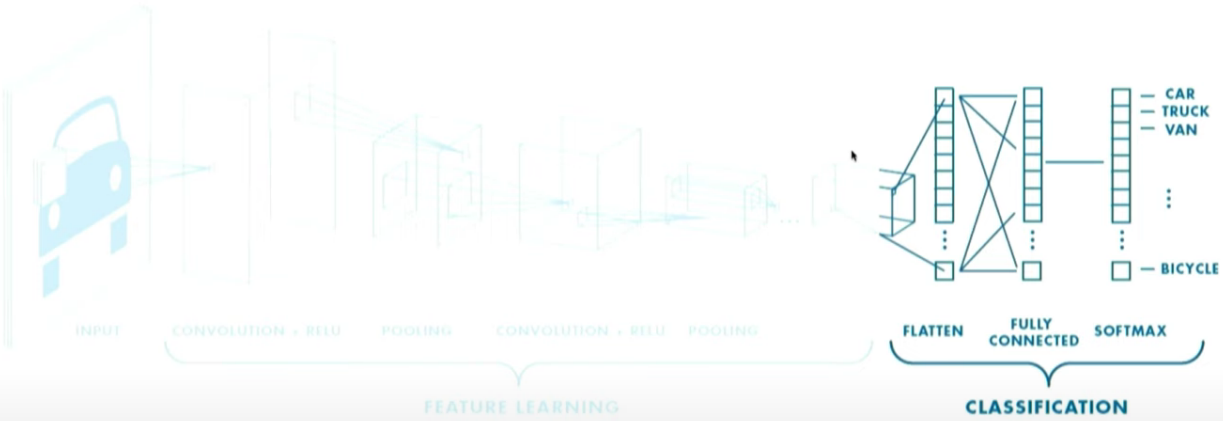
CNN Pipeline

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

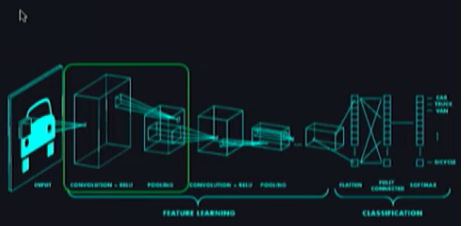


- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

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

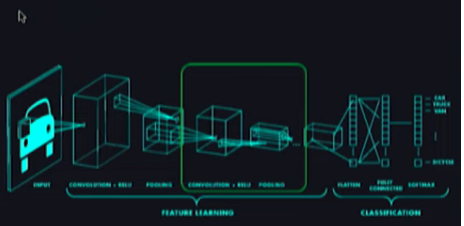
Putting it all together

```
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

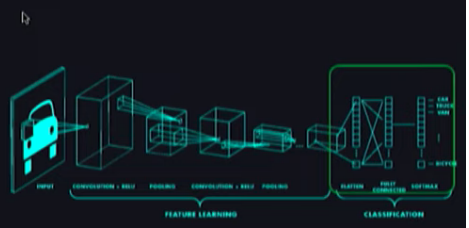
```
import tensorflow as tf

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
```



Massachusetts Institute of Technology
6.S191 Introduction to Deep Learning
introtodeeplearning.com @MITDeepLearning
1/25/22

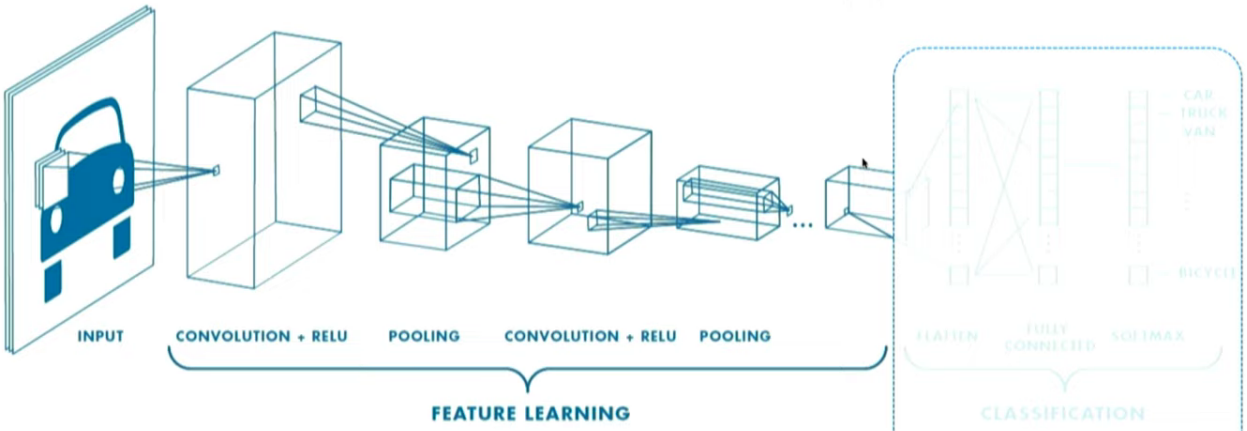
Putting it all together

```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
```



CNN Applications

An Architecture for Many Applications



Classification
Object detection
Segmentation
Probabilistic control

Object Detection

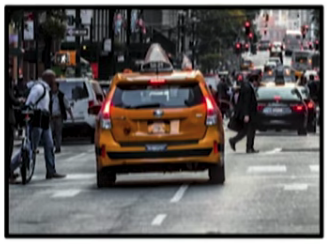


Image x

CNN

Taxi

Class label y

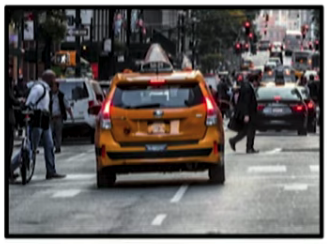
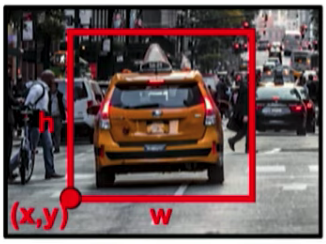


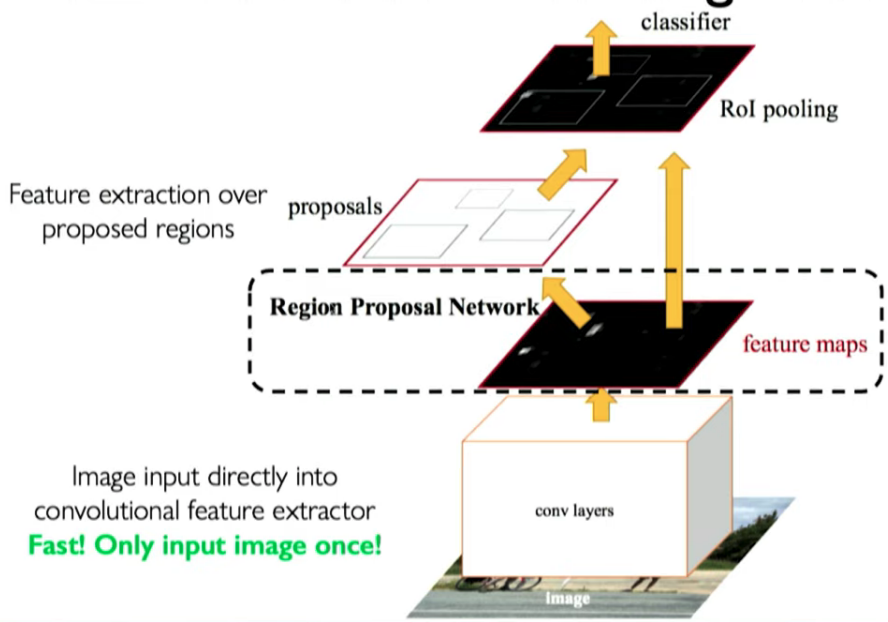
Image x

CNN



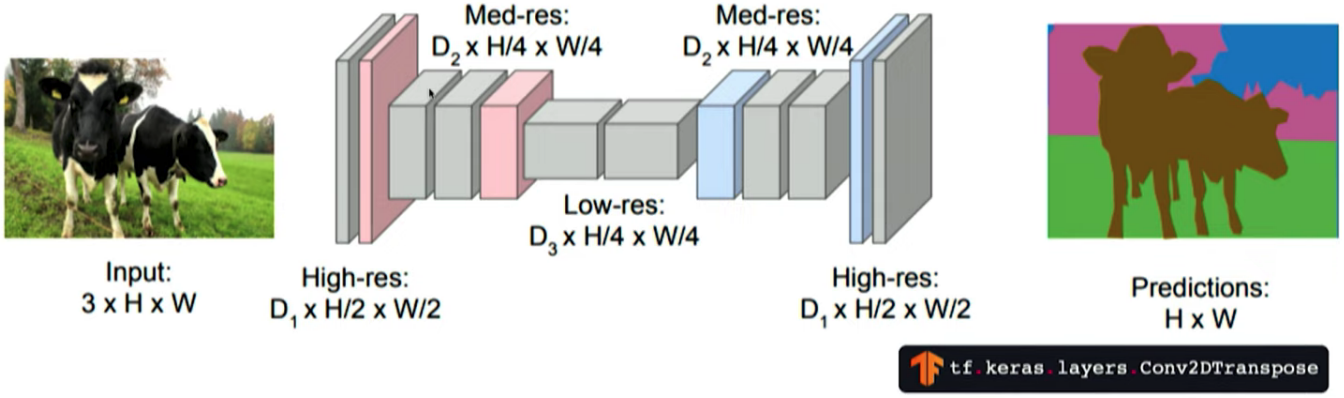
Label (x, y, w, h)

Faster R-CNN Learns Region Proposals



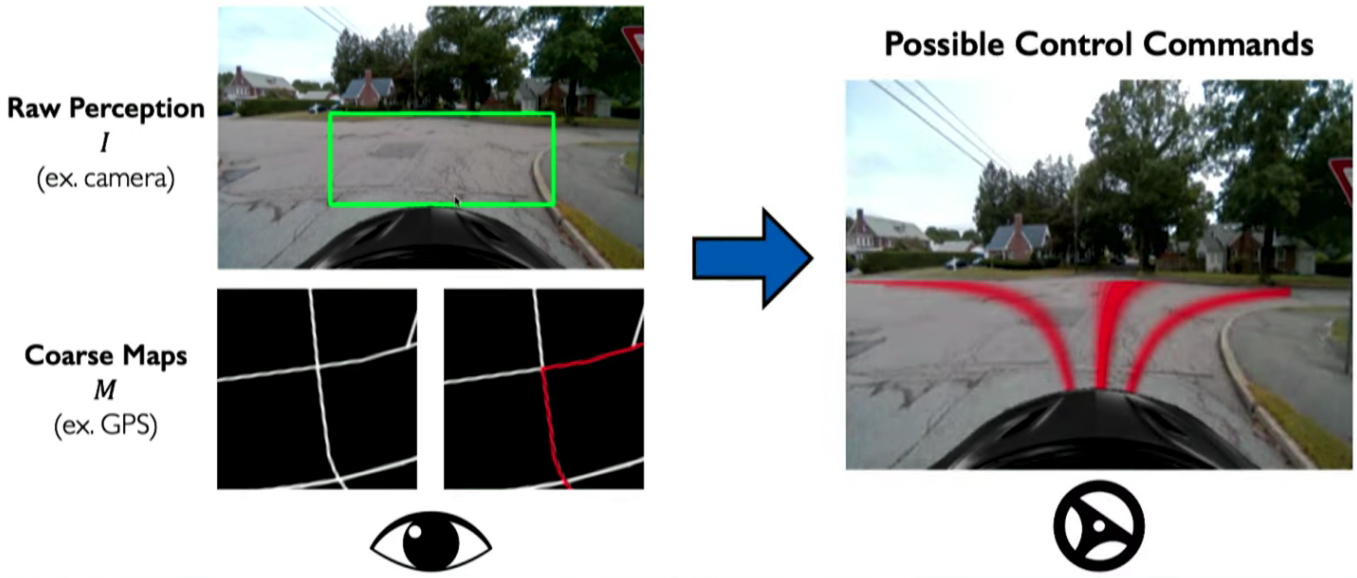
Semantic Segmentation: Fully Convolutional Networks

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



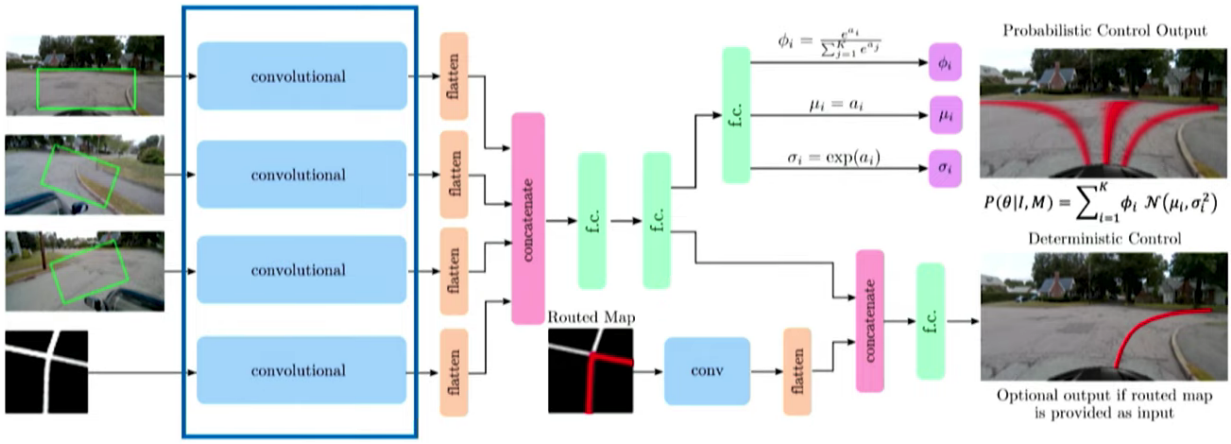
`tf.keras.layers.Conv2DTranspose`

Continuous Control: Navigation from Vision



End-to-End Framework for Autonomous Navigation

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



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

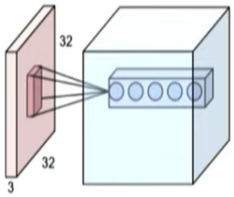
Deep Learning for Computer Vision: Summary

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