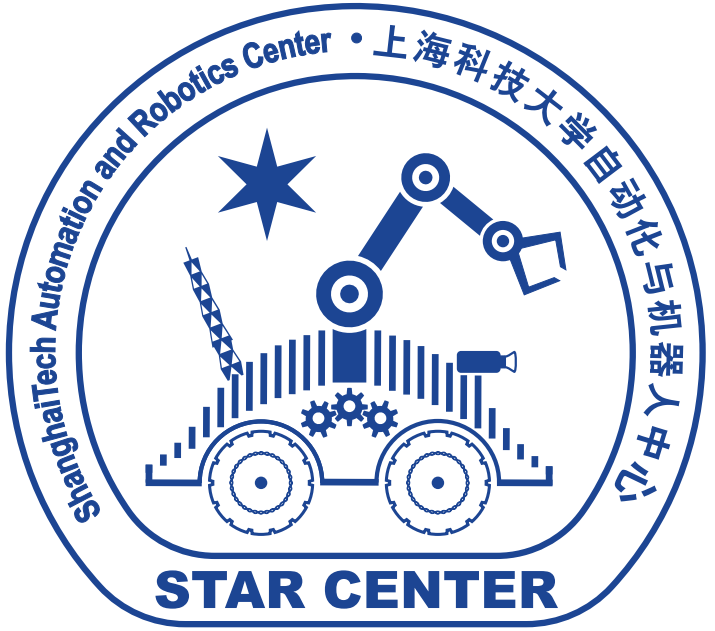
上海科技大学

ShanghaiTech University

CS289: **Mobile Manipulation Fall 2025**

Sören Schwertfeger

ShanghaiTech University



Outline

- Simulation
- Perception

SIMULATION

Types of Simulators

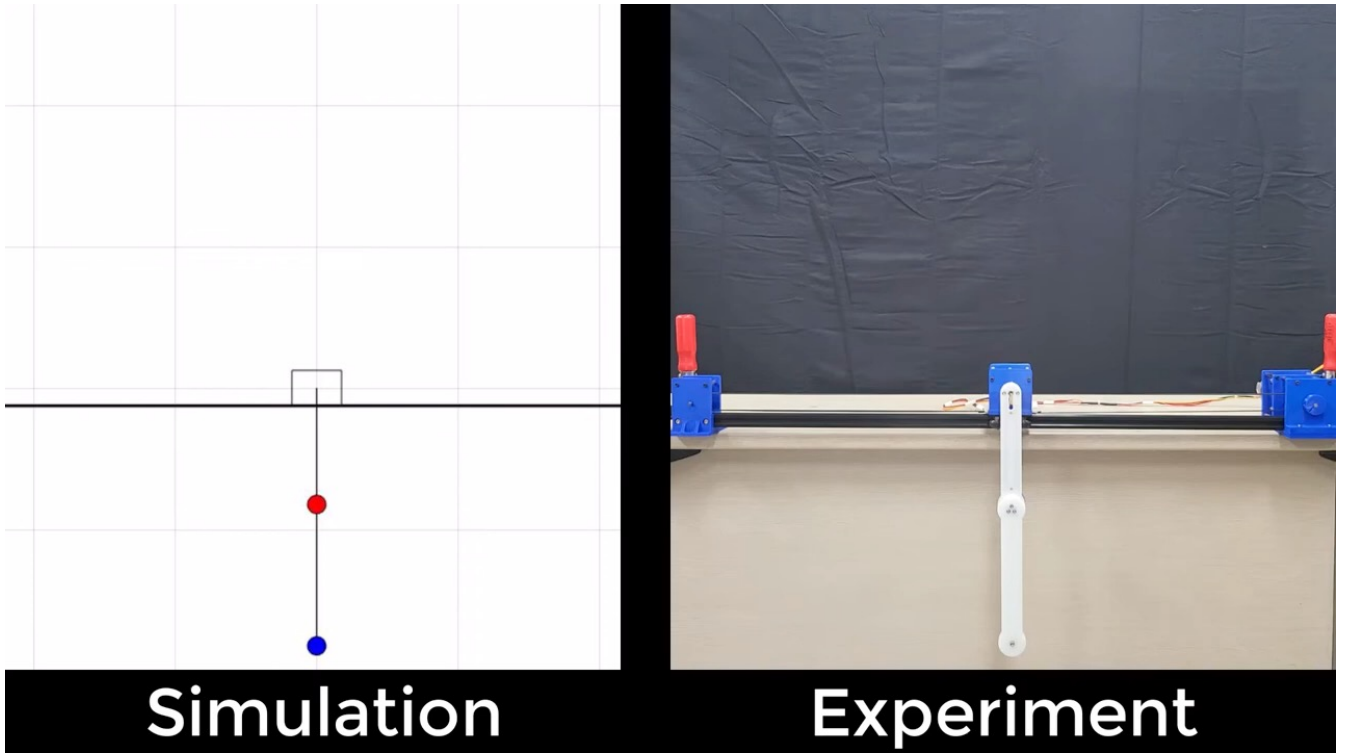
- Dynamics Simulation
- 3D Games
- Photo-realistic rendering
- Mission simulators
- 2D simulators
- 3D simulators

Dynamics Simulation (Control)

- Model a dynamic system
  - Kinematics (joints & links)
  - Dynamics (mass, inertia)
- Often used for research on control

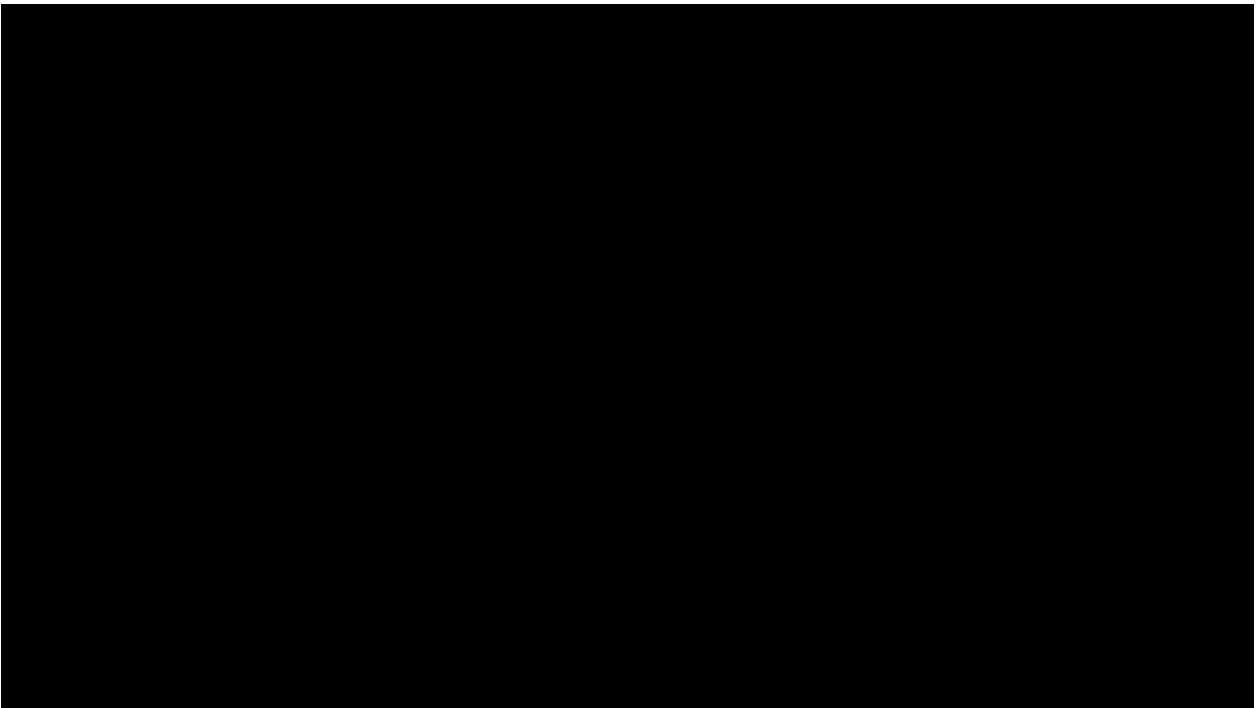
- Example:
  - Double inverted pendulum
  - E.g. via Model Predictive Control (MPC)
  - 2D simulation only

https://www.youtube.com/watch?v=e7669NPbENY



3D Games

- Simulate 3D world:
  - Kinematics and Dynamics (typically very simple)
  - RGB camera view
  - Sound



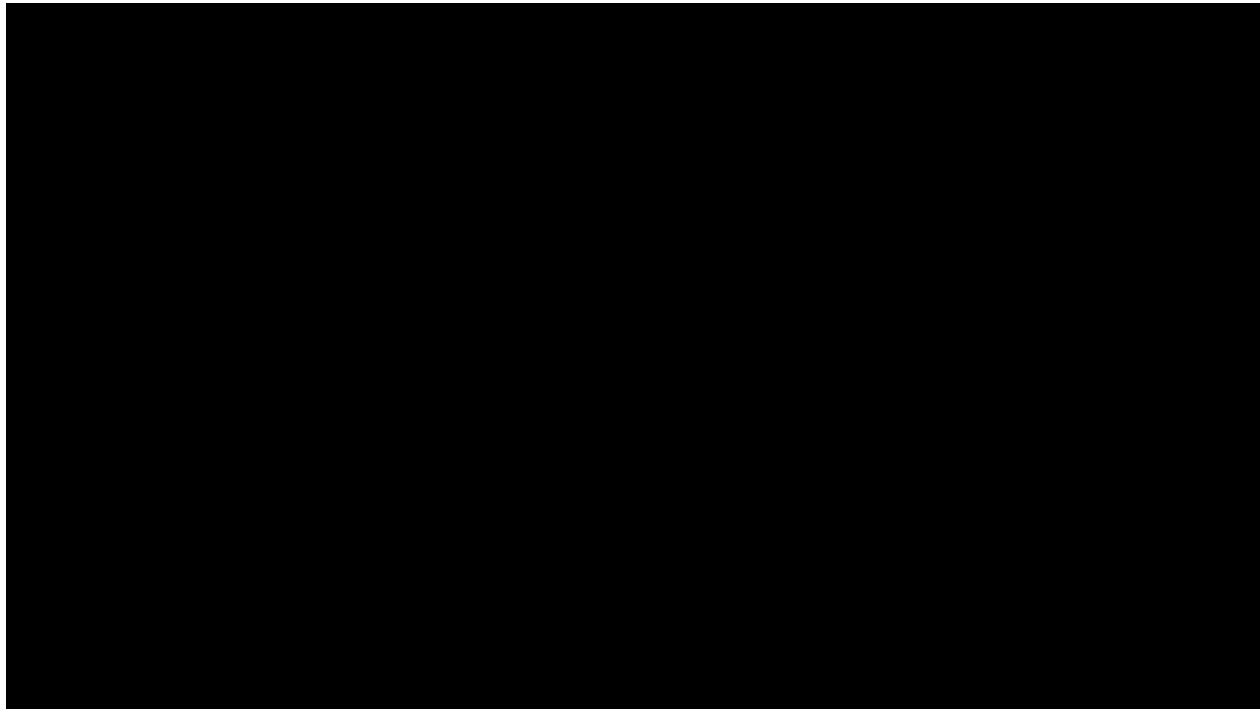
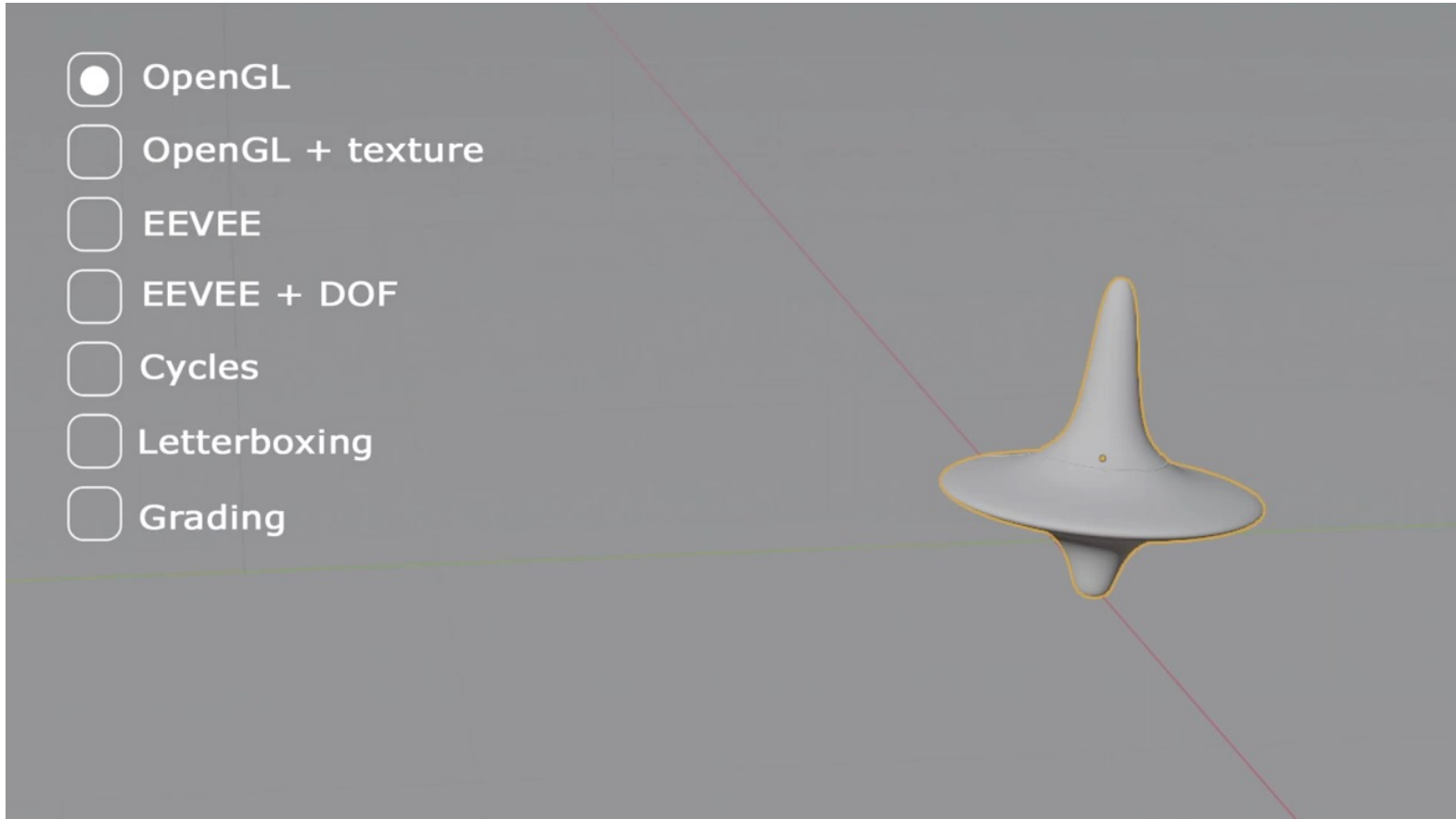


Photo-realistic rendering

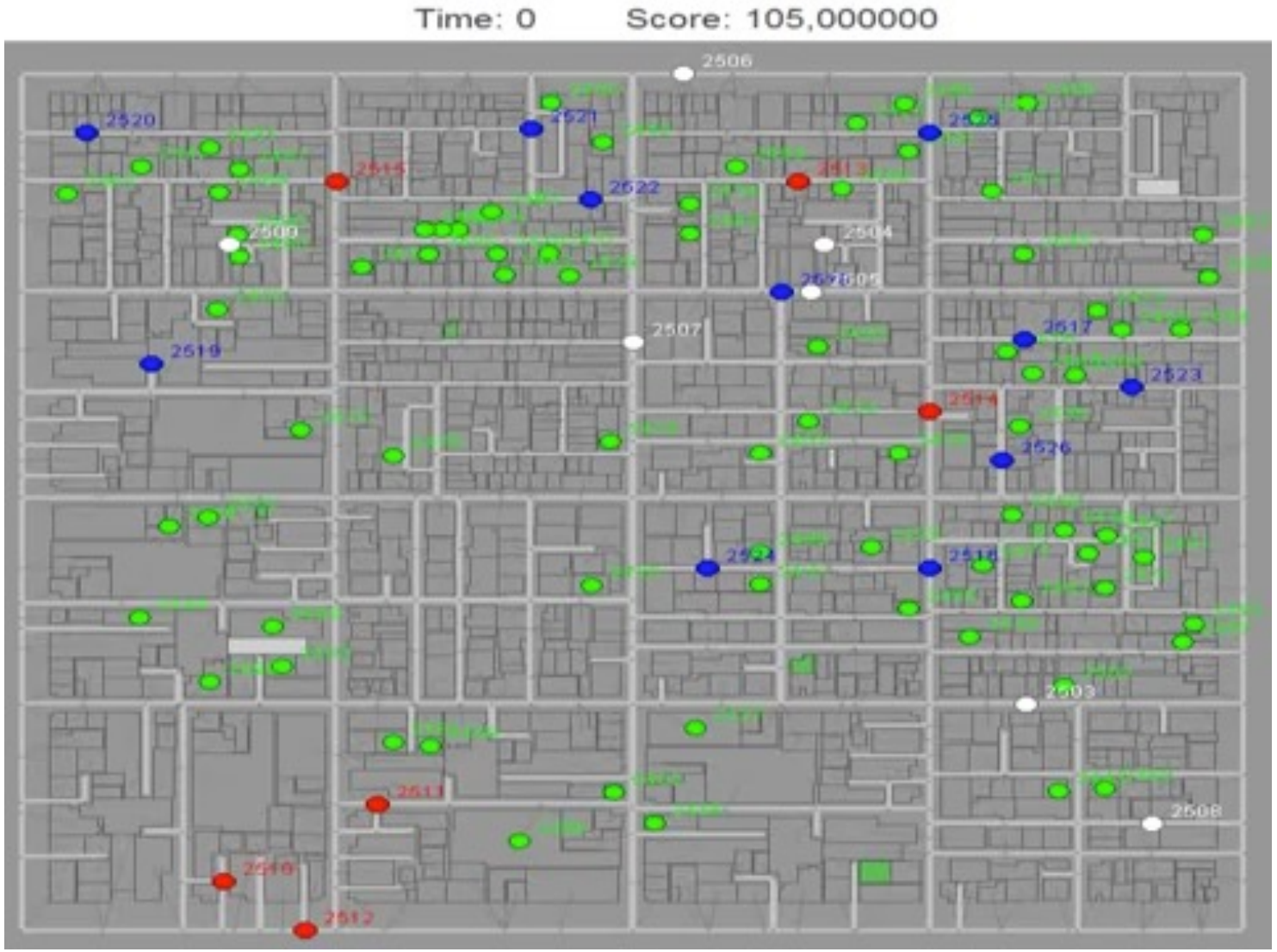
- 3D game engine photos not realistic (at least in the past)
  - Problem for computer vision algorithms
  - Especially for training computer vision algorithms (sim-to-real)
- Photo-realistic rendering:
  - Time consuming
  - Especially lights, shadows, transparence, hair, etc.
  - Often: non-real time
  - Blender: open source renderer:



Next slide: Unrecord trailer
https://www.youtube.com/watch?v=otu_iFTivQw

Robot mission task simulator: RoboCup Rescue

- Simulate deployment of (robotic) units like police, fire, ambulance
- All entities are agents
- Communication between agents
- Simulate environment, e.g.:
  - Spreading of fire



3D Robot Simulation

Why?

- Training & education
  - (when you don't have a robot)
- Ease of experiments
  - Don't need to deal with the hardware (e.g. battery charging, weather, robot breaking, bringing robot back to start, …)
- Multi-robot simulation
  - Simulate multiple (100s!?) of robots -> low cost in simulation
- Collect ground truth data
  - Know exactly where the robot is & its state
  - Know exactly where surrounding objects are
- Collect training data (see above)
- Train robot online (e.g. reinforcement learning)

Why not?

- Extra work
  - Need to setup simulated robot & simulated world
- Simulated data differs from real data
  - Non-photo realistic rendering
  - Non-realistic noise on simulated sensor data
  - Non-realistic physics (e.g. tracked locomotion)


Activities
rviz2
7月 28 02:49
AWSIM
/home/mackie/workspace/GitHub/tier4/autoware/install/autoware_launch/share/autoware_launch/rviz/autoware....
File
Panels
Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Goal Pose
-0.4deg
2.90km/h
limited
72km/h
occlusion spot
occupancy
4d6a
,d,v,m)=(-0.85 , -3.74 , 3.88 , 7.22
82e4
Time Scale x 1.00
AWSIM
(prototype ver.)
Time
ROS Time: 63.06
ROS Elapsed: 53.01
Wall Time: 1658944181.38
Wall Elapsed: 53.02
Experimental
Reset
21 fps

Waiting for the next game...

3rd place match: 12:00


Untitled

Simulators Overview

- Many different simulation platforms available…

Tselegkaridis, Sokratis, and Theodosios Sapounidis. "Simulators in educational robotics: A review." Education Sciences 11.1 (2021): 11.

Name	Develop Year	Type of Robot	Users' Age	Programming Language	Scope	Used	User's Level	Based on a Real Robot	Development Platform	Operating System
RoSoS *	2016	Wheeled	6~7	Processing	Competition	1, 2	Beginner	No	Processing	W, L, M
RoboSim	2017	Modular	10~18	C/C++	Competition	1	Intermediate	Yes	-	W, M
Robot One *	2019	Multiple	>18	Many	Education	2	Expert	No	Unity3D	W, L
Tactode	2019	Multiple	6~11	Puzzle/Tangible	Education	-	Beginner	No	-	n/s
Pololu *	2018	Wheeled	n/s	C/C++	Education	2	Beginner	Yes	-	W
EUROPA	2019	Wheeled	>12	Python	Education	3	Intermediate	Yes	Gazebo	n/s
ADS *	2019	Wheeled	>18	n/s	Competition	1, 2	Intermediate	Yes	Gazebo	L
MLPNN	2020	Wheeled	>18	MATLAB	Research	-	Expert	No	MATLAB	W, L
Drone Simulator	2019	Drone	>12	MATLAB	Research	1, 3	Expert	No	MATLAB/Gazebo	W, L
PiBot *	2018	Wheeled	>12	Python	Education	4	Intermediate	Yes	Gazebo	W, L, M
AlphaBot2 *	2019	Wheeled	>12	n/s	Research	-	Intermediate	Yes	Gazebo	W, L
MSRP	2017	Wheeled	>18	n/s	Education	-	Expert	Yes	Unity3D	n/s
KheperaIV	2016	Wheeled	n/s	C/C++	Education	-	Intermediate	Yes	V-REP	n/s
OBR simulator	2018	Wheeled	6~18	Flowchart	Competition	1, 5	Intermediate	Yes	V-REP	W, L
MiniSim *	2014	Wheeled	6~10	Python/Block	Education	-	Beginner	No	-	W, L
SRM	2018	Arm	>18	MATLAB	Education	2	Expert	No	MATLAB/V-REP	W, L
LaRoCS + Unesp	2018	Drone	n/s	n/s	Competition	-	Intermediate	Yes	V-REP	W, L, M

Note: 1: Competition, 2: University, 3: Secondary School, 4: Robotics workshops for teachers, 5: Technical high school; W: Windows, L: Linux, M: MacOS; *: Open access; n/s: not specified.

Most popular professional/ research Robot Simulators

Farley, Andrew, Jie Wang, and Joshua A. Marshall. "How to pick a mobile robot simulator: A quantitative comparison of CoppeliaSim, Gazebo, MORSE and Webots with a focus on accuracy of motion." Simulation Modelling Practice and Theory 120 (2022): 102629.

Summary of the simulator qualitative features evaluation results, after completing the setup of each simulation environment.

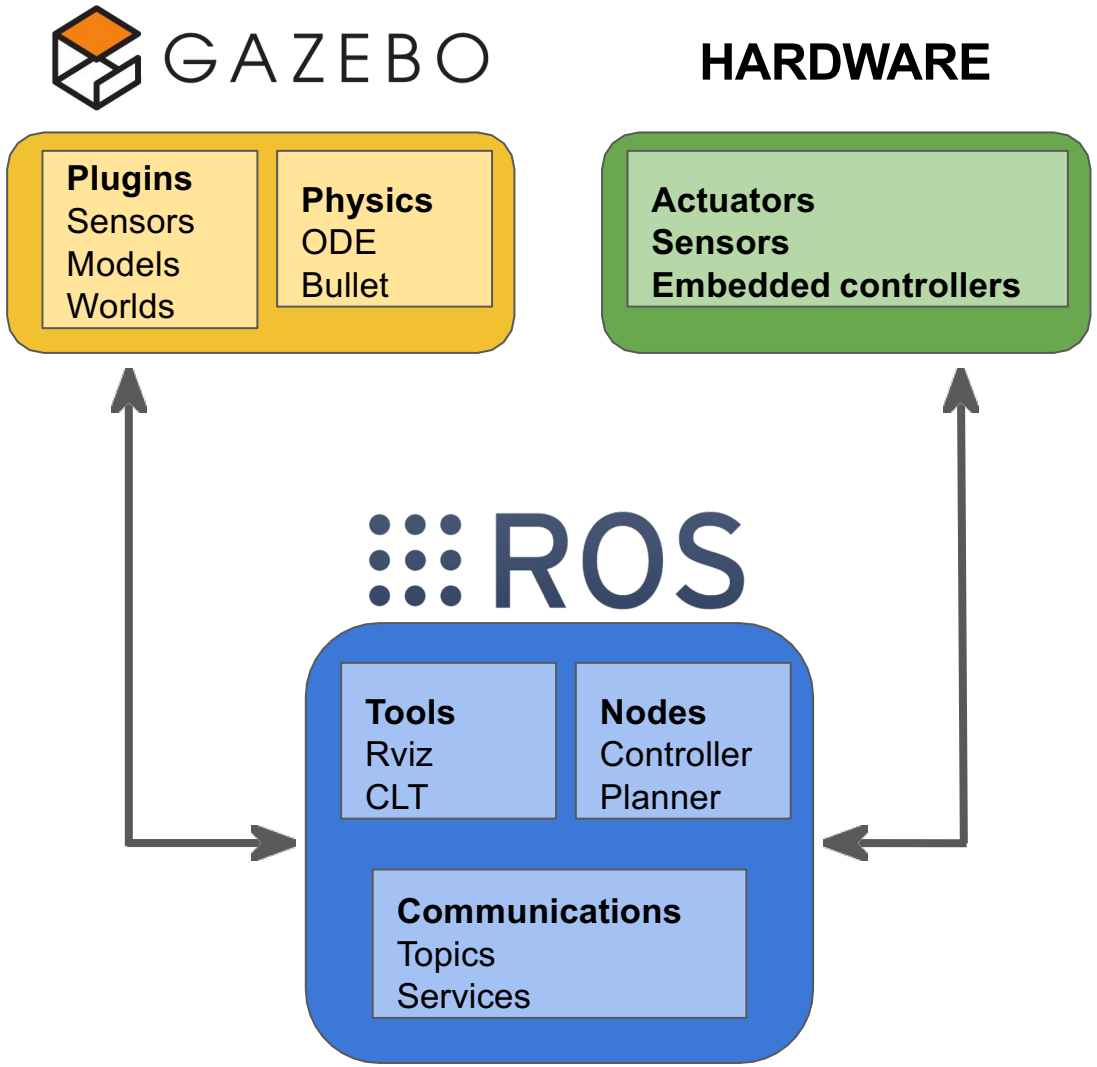
Metric name	CoppeliaSim	Gazebo	MORSE	Webots
Free to use	True	True	True	True
Open source	False	True	True	True
ROS compatibility	A built-in plugin provided	Out of the box	Out of the box	A built-in plugin provided
Programming languages	C/C++, Python, Lua, MATLAB, Java, Octave	C/C++, Python	Python	C/C++, Python, Java, MATLAB
UI functionality	Full functionality	Full functionality	Visualization only	Full functionality
Model format support	URDF, SDF, Stl, Obj, Dxf, Collada	URDF, SDF, Stl, Obj, Collada	Blend	Proto Nodes
Physics engine support	Bullet, ODE, Vortex, Newton	Bullet, ODE, DART, Simbody	Bullet	ODE

Simulation for Reinforcement Learning

- Emphasis on physics simulation and RL, less on extrinsic sensors nor complete robot simulation
- GPU acceleration for massive parallel simulation
- MuJoCo (Multi-Joint dynamics with Contact)
  - By Google DeepMind
  - Limited LiDAR simulation
  - No official ROS integration
- Isaac Lab
  - By Nvidia
  - Needs Nvidia GPU
  - Supports hardware ray-tracing (for LiDAR)
  - Not very good support for ROS
  - Big software packages (including Omniverse)

Gazebo Robot Simulator

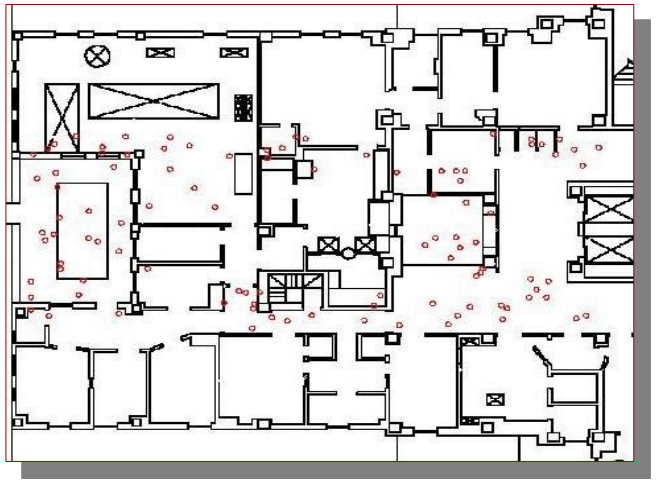
- Simulators mimic the real world, to a certain extent
  - Simulates robots, sensors, and objects in a 3-D dynamic environment
  - Generates realistic sensor feedback and physical interactions between objects



2D vs 3D Simulation

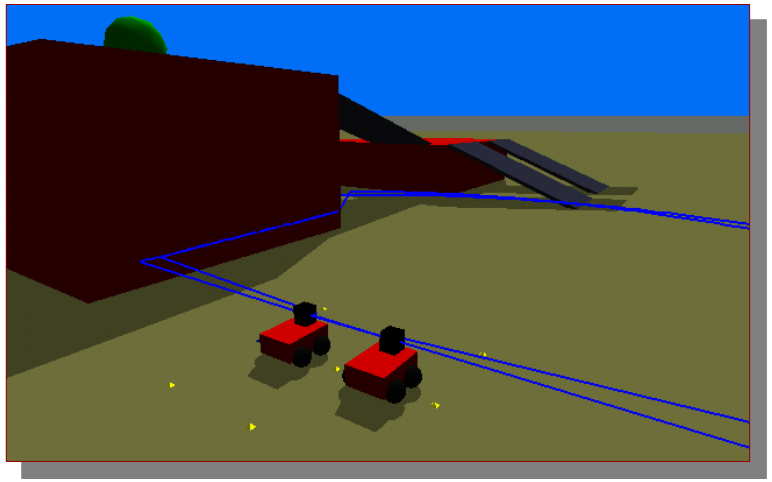
Stage

- 2D
- Sensor-based
- Player interface
- Kinematic
- O(1) ~ O(n)
- Large teams (100's)



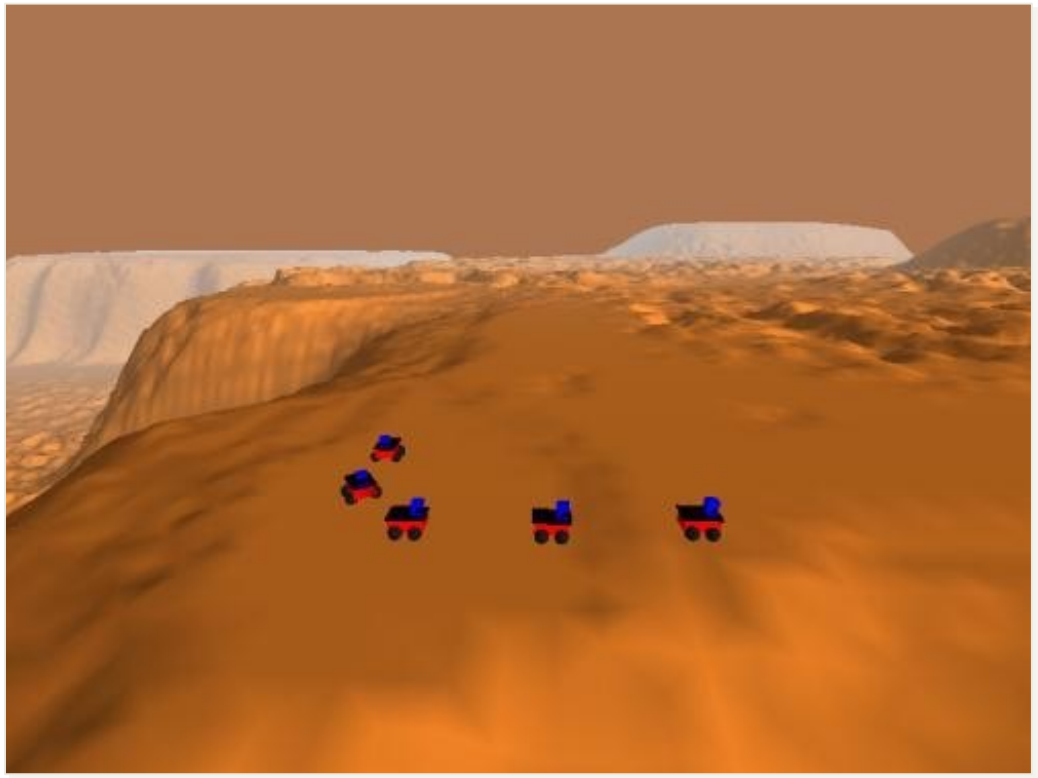
Gazebo

- 3D
- Sensor-based
- Player
- Dynamic
- O(n) ~ $O(n^3)$
- Small teams (10's)

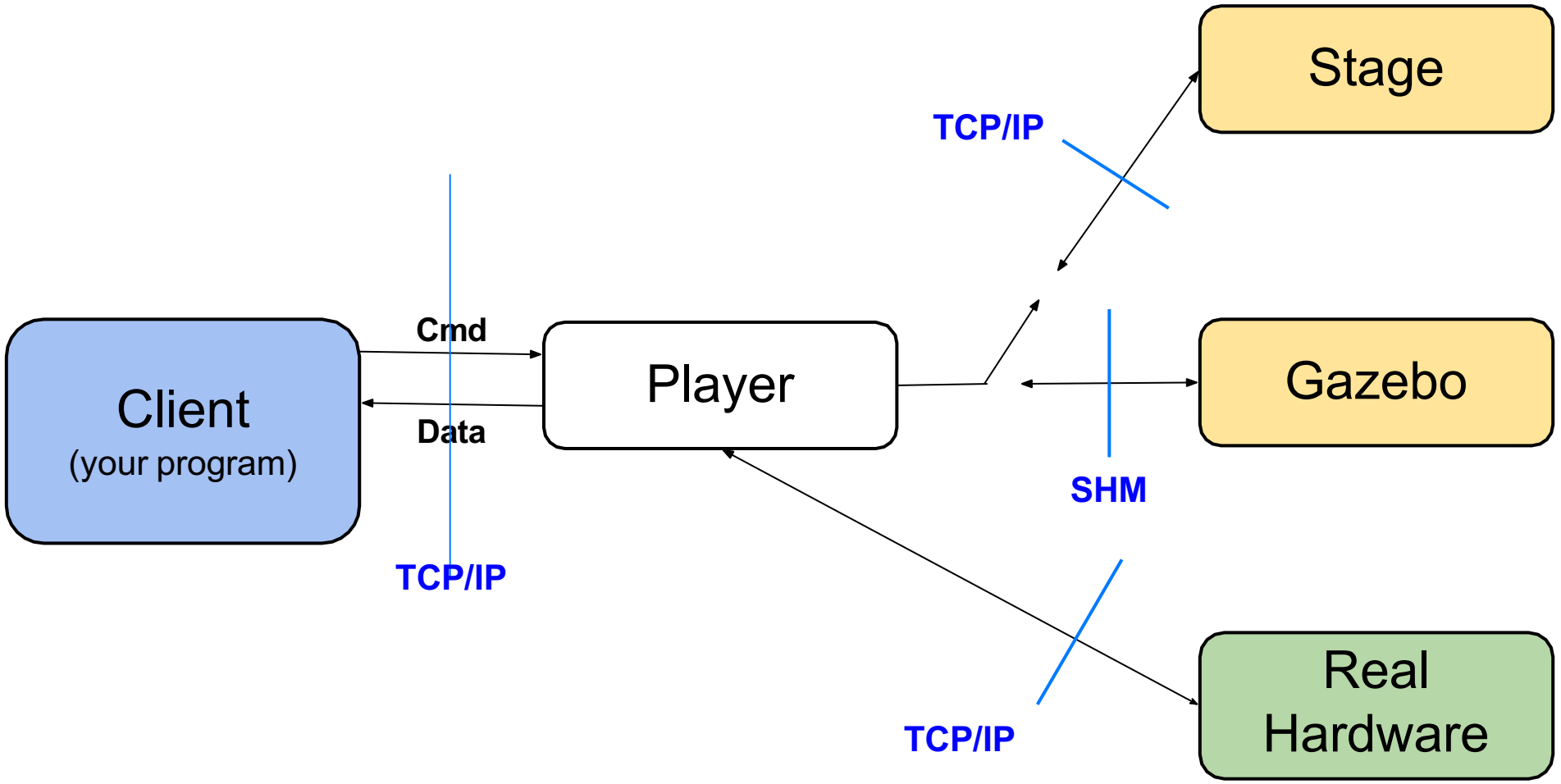


Gazebo

- Simulates robots, sensors, and objects in a 3-D dynamic environment
- Generates realistic sensor feedback and physical interactions between objects



Simulation Architecture



Simulation Architecture

Gazebo runs two processes:

- **Server**: Runs the physics loop and generates sensor data.
  - Executable: *gzserver*
  - Libraries: Physics, Sensors, Rendering, Transport
- **Client**: Provides user interaction and visualization of a simulation.
  - Executable: *gzclient*
  - Libraries: Transport, Rendering, GUI

Run Gazebo server and client separately:

```
$ gzserver
$ gzclient
```

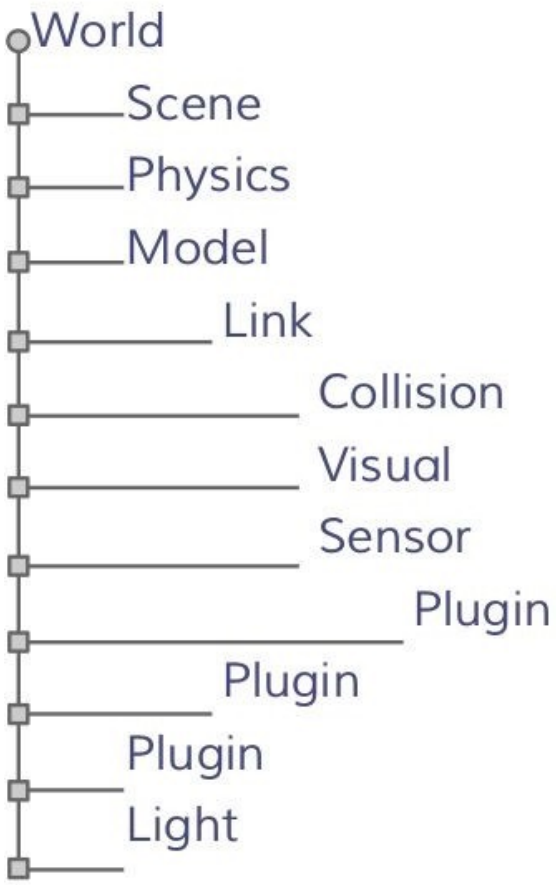
Run Gazebo server and client simultaneously:

```
$ gazebo
```

Elements within Simulation

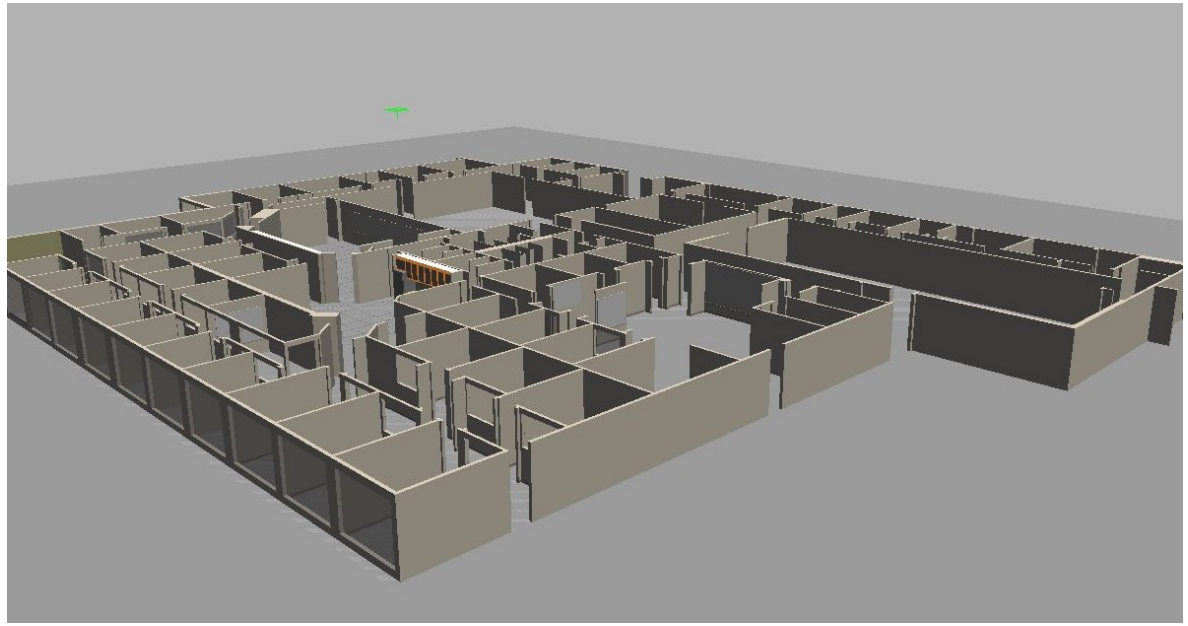
- World
  - Collection of models, lights,plugins and global properties
- Models
  - Collection of links, joints,sensors, and plugins
- Links
  - Collection of collision and visual objects
- Collision Objects
  - Geometry that defines a colliding surface
- Visual Objects
  - Geometry that defines visual representation
- Joints
  - Constraints between links
- Sensors
  - Collect, process, and output data
- Plugins
  - Code attached to a World, Model, Sensor, or the simulator itself

Element Hierarchy



World

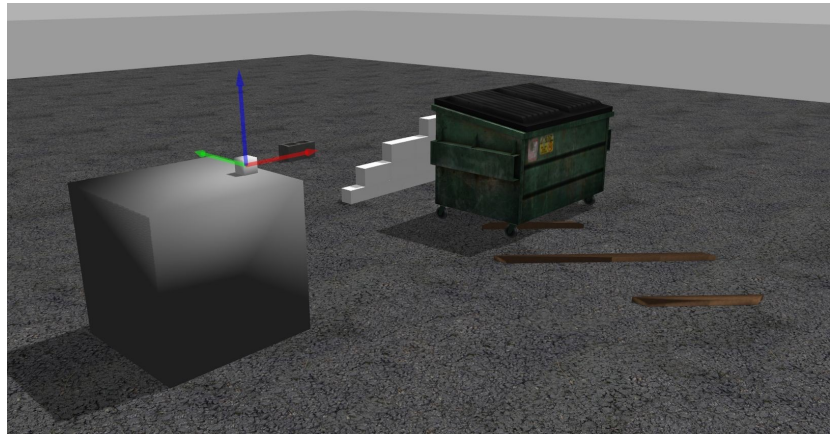
- A world is composed of a model hierarchy
- The Gazebo server (*gzserver*) reads the world file to generate and populate a world
  - This file is formatted using SDF (Simulation Description format) or URDF (Unified Robot Description Format)
  - Has a *".world"* extension
  - Contains all the elements in a simulation, including robots, lights, sensors, and static objects



Willow Garage World

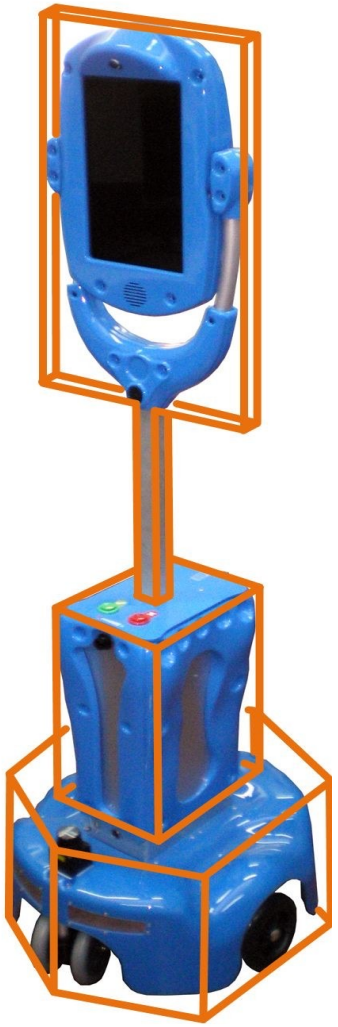
Models

- Each model contains a few key properties:
  - **Physical presence** (optional):
    - Body: sphere, box, composite shapes
    - Kinematics: joints, velocities
    - Dynamics: mass, friction, forces
    - Appearance: color, texture
  - **Interface** (optional):
    - Control and feedback interface (libgazebo)



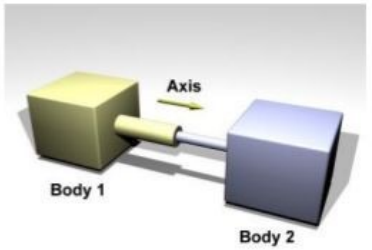
Element Types

- Collision and Visual Geometries
  - Simple shapes: sphere, cylinder, box, plane
  - Complex shapes: heightmaps, meshes

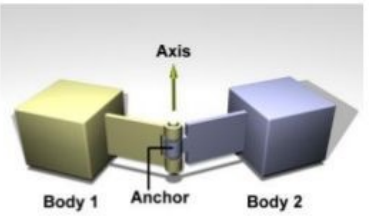


Element Types

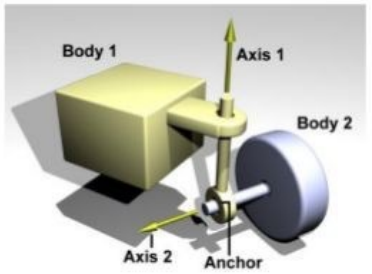
- Collision and Visual Geometries
  - Simple shapes: sphere, cylinder, box, plane
  - Complex shapes: heightmaps, meshes
- Joints
  - Prismatic: 1 DOF translational
  - Revolute: 1 DOF rotational
  - Revolute2: Two revolute joints in series
  - Ball: 3 DOF rotational
  - Universal: 2 DOF rotational
  - Screw: 1 DOF translational, 1 DOF rotational



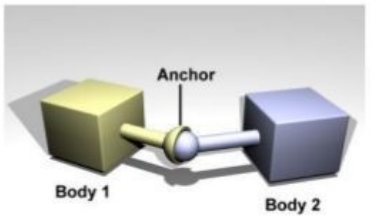
Prismatic



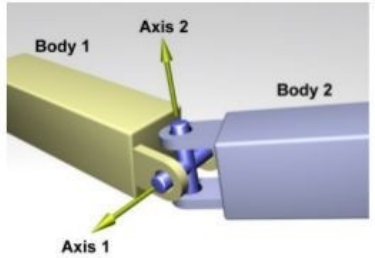
Revolute



Revolute 2



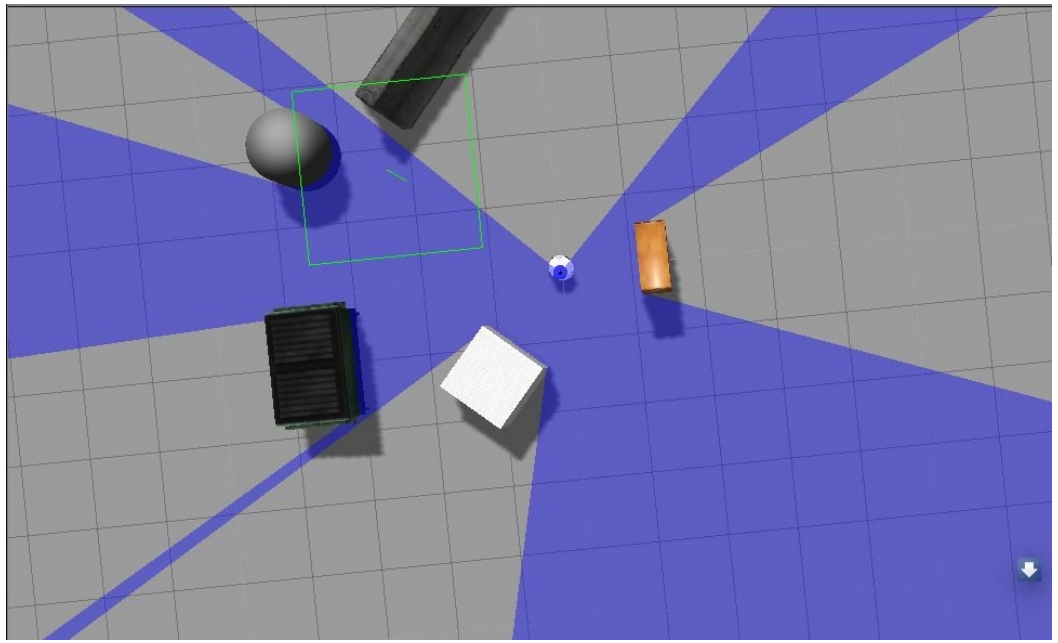
Ball



Universal

Element Types

- Sensors
  - Ray: produces range data
  - Camera (2D and 3D): produces image and/or depth data
  - Contact: produces collision data
  - RFID: detects RFID tags
- Lights
  - Point: omni-directional light source, a light bulb
  - Spot: directional cone light, a spot light
  - Directional: parallel directional light, sun

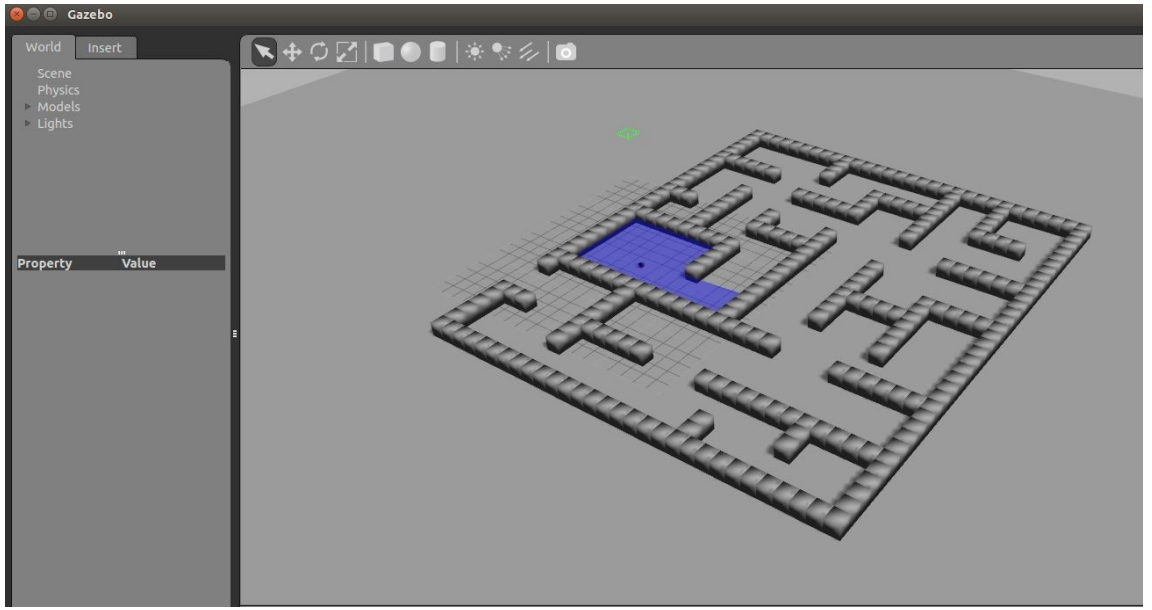


LiDAR sensor in Gazebo

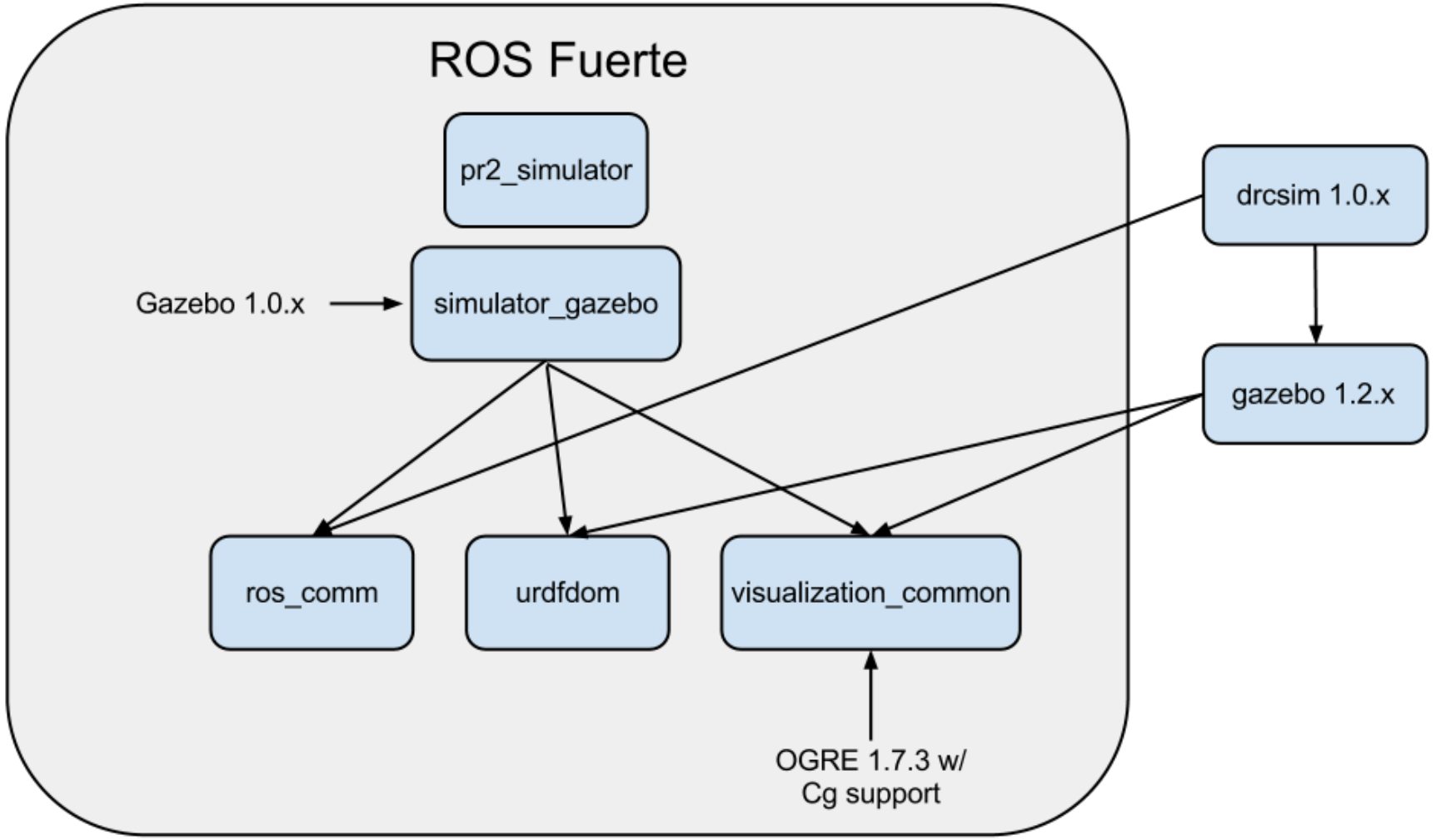
How to use Gazebo to simulate your robot?

Steps:

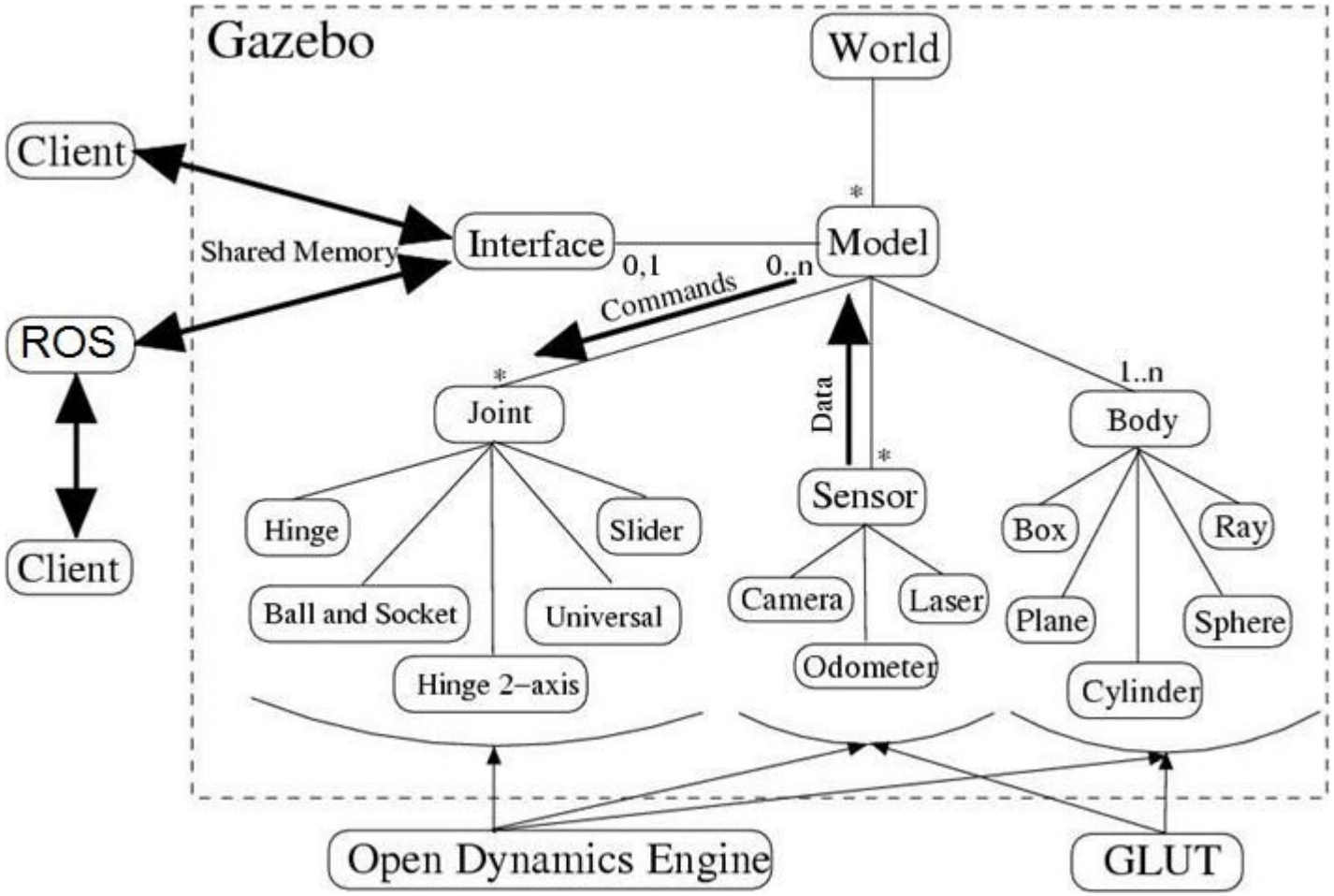
1. load a world
2. load the description of the robot
3. spawn the robot in the world
4. publish joints states
5. publish robot states
6. run rviz



ROS Integration Overview



Gazebo Architecture



World File

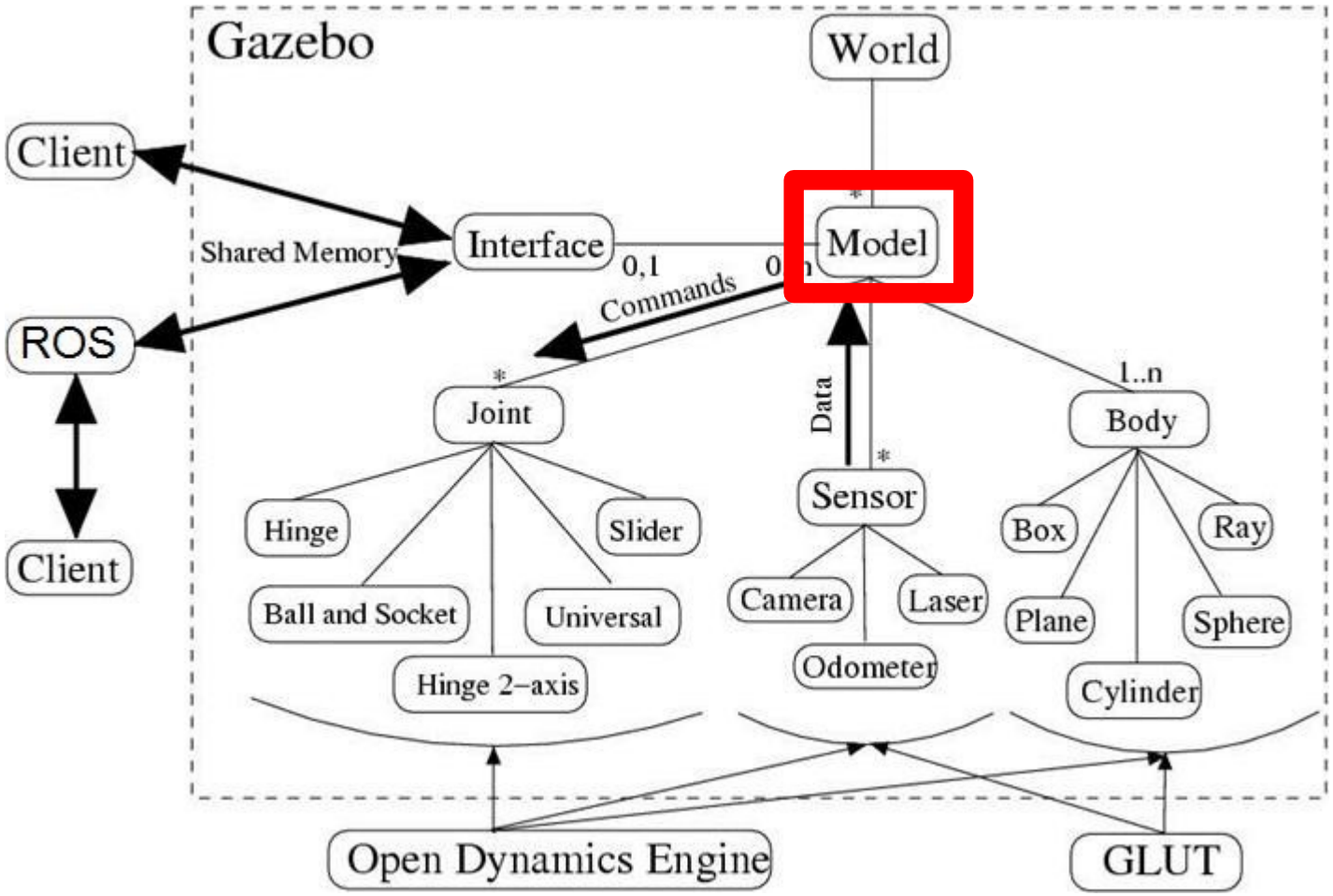
- A world is composed of a model hierarchy
  - Models define simulated devices
  - Models can be nested
    - Specifies how models are physically attached to one another
    - Think of it as “bolting” one model to another

Worldfile snippet:

- Pioneer with a sick laser attached
- Sick’s <xyz> relative location to Pioneer

```
<model:Pioneer2AT>
  <id>robot1</id>
  <model:SickLMS200>
    <id>laser1</id>
    <xyz>0.10.0 0.2</xyz>
  </model:SickLMS200>
</model:Pioneer2AT>
```

Gazebo Architecture



Models

- Each model contains a few key properties:
  - Physical presence (optional):
    - Body: sphere, box, composite shapes
    - Kinematics: joints, velocities
    - Dynamics: mass, friction, forces
    - Appearance: color, texture
  - Interface (optional):
    - Control and feedback interface (libgazebo)

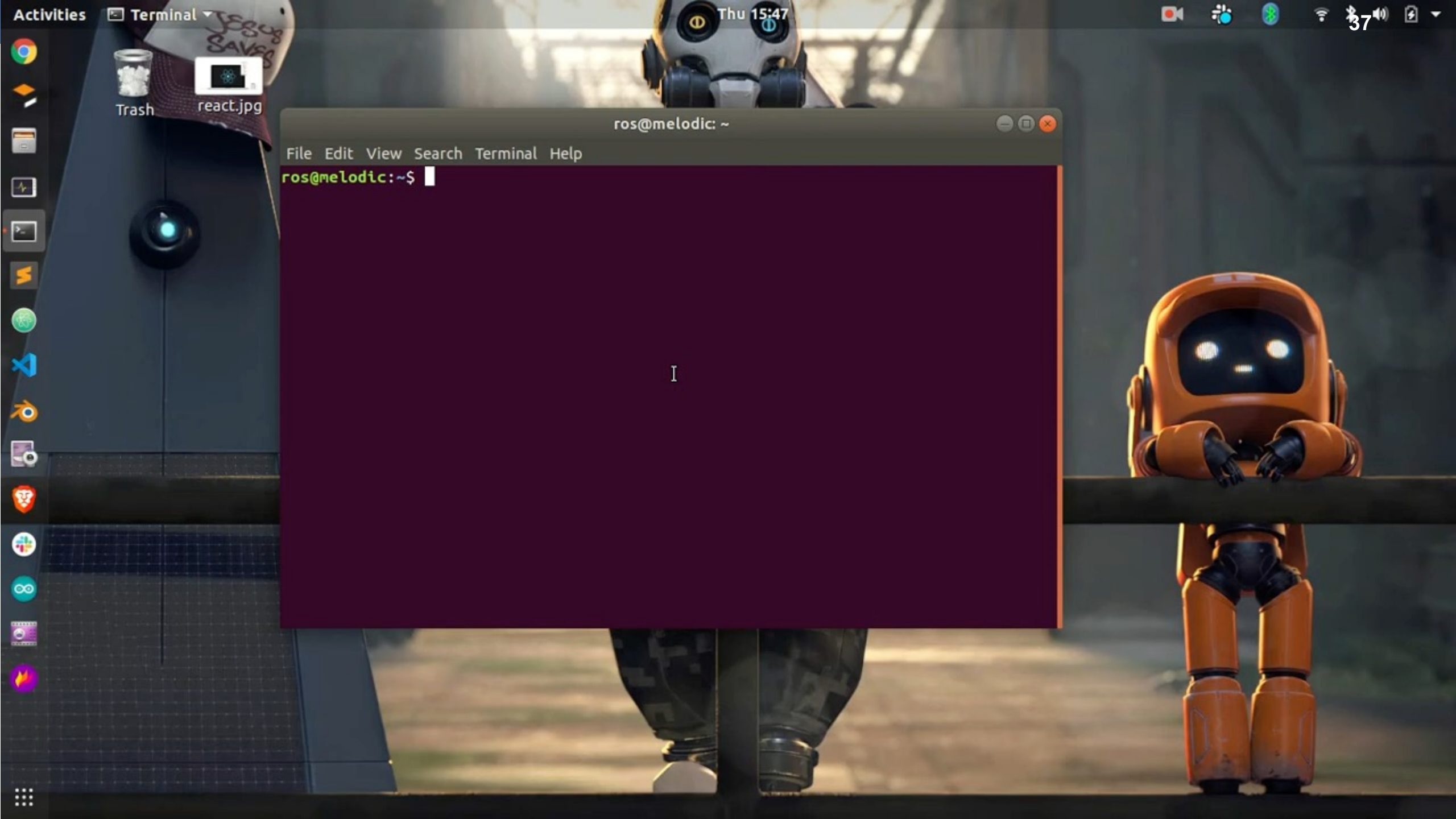
Components of a Model

- **Links**: an object may consist of multiple links and can define following properties, e.g. wheel
  - **Visual**: For visualization
  - **Collision**: Encapsulate geometry for collision checking
  - **Inertial**: Dynamic properties of a link e.g. mass, inertia
  - **Sensors**: To collect data from world for plugins
- **Joints**: connect links using a parent-child relationship
- **Plugins**: Library to control model

Example Model File

```
<model name="my_model">
 <pose>0 0 0.5 0 0 0</pose>
 <static>true</static>
  <link name="link">
   <inertial>
    <mass>1.0</mass>
     <ixx>0.083</ixx>     <!-- for a box: ixx = 0.083 * mass * (y*y + z*z) -->
     <ixy>0.0</ixy>       <!-- for a box: ixy = 0 -->
     <ixz>0.0</ixz>       <!-- for a box: ixz = 0 -->
     <iyy>0.083</iyy>     <!-- for a box: iyy = 0.083 * mass * (x*x + z*z) -->
     <iyz>0.0</iyz>       <!-- for a box: iyz = 0 -->
     <izz>0.083</izz>     <!-- for a box: izz = 0.083 * mass * (x*x + y*y) -->
    </inertia>
   </inertial>
.....
```

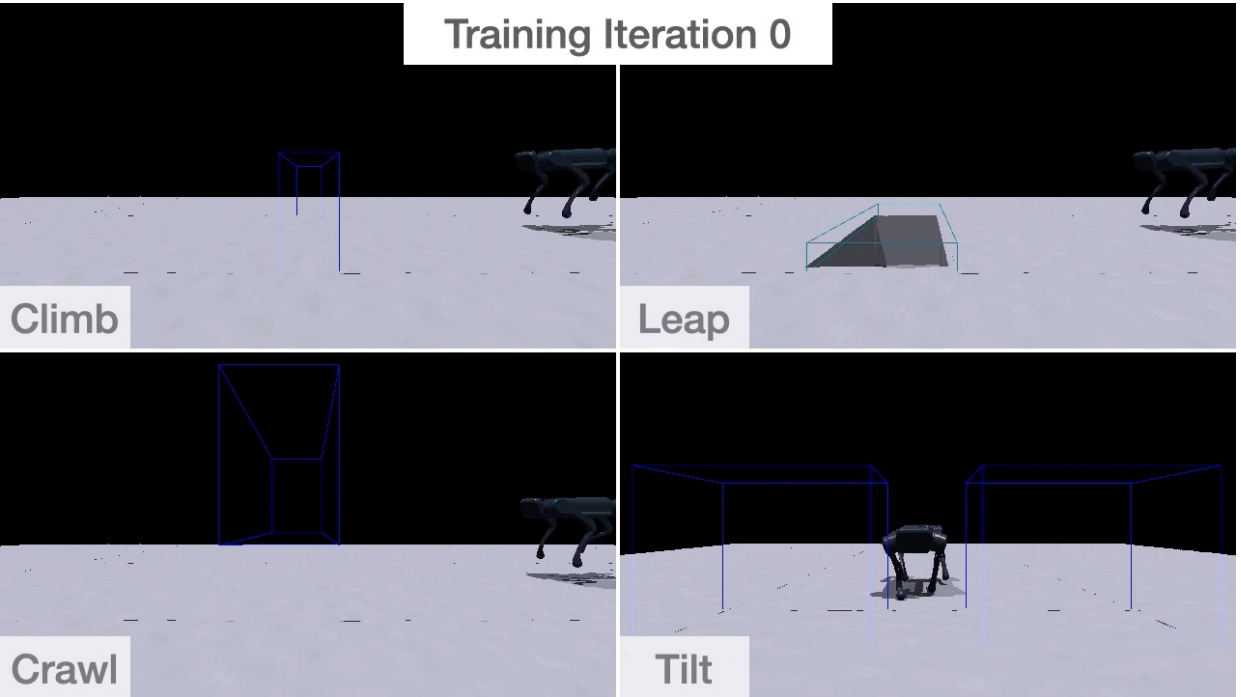
```
.....
 <collision name="collision">
      <geometry>
       <box>
        <size>1 1 1</size>
       </box>
      </geometry>
 </collision>
 <visual name="visual">
      <geometry>
       <box>
        <size>1 1 1</size>
       </box>
      </geometry>
  </visual>
 </link>
</model>
```

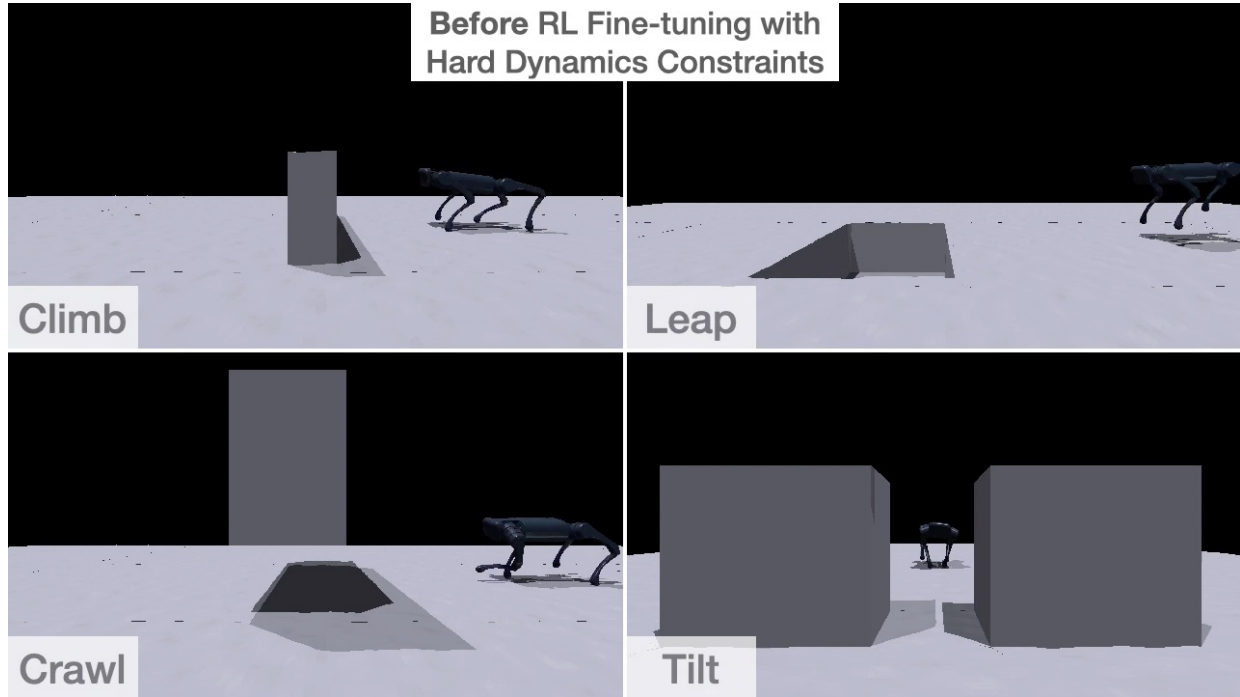

Activities
Terminal
Thu 15:47
37
Trash
react.jpg
ros@melodic: ~
File Edit View Search Terminal Help
ros@melodic:~$


Marrobamba
Tampico

Reinforcement learning example: Robot parkour learning

- Zhuang, Z., Fu, Z., Wang, J., Atkeson, C., Schwertfeger, S., Finn, C., & Zhao, H. (2023). Robot parkour learning. arXiv preprint arXiv:2309.05665.





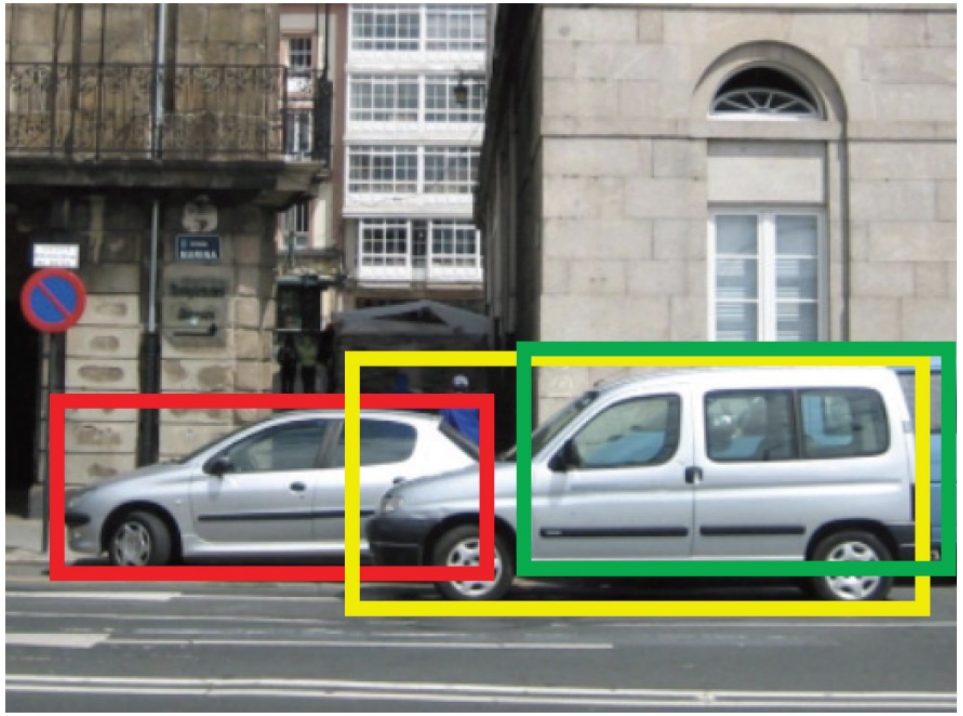
Robot Parkour Learning

twitter: @ziwenzhuang_leo
@zipengfu

PERCEPTION

Overview

- Motivation
- Image Classification
- Object Detection
- Object Segmentation
- Vision for Manipulation

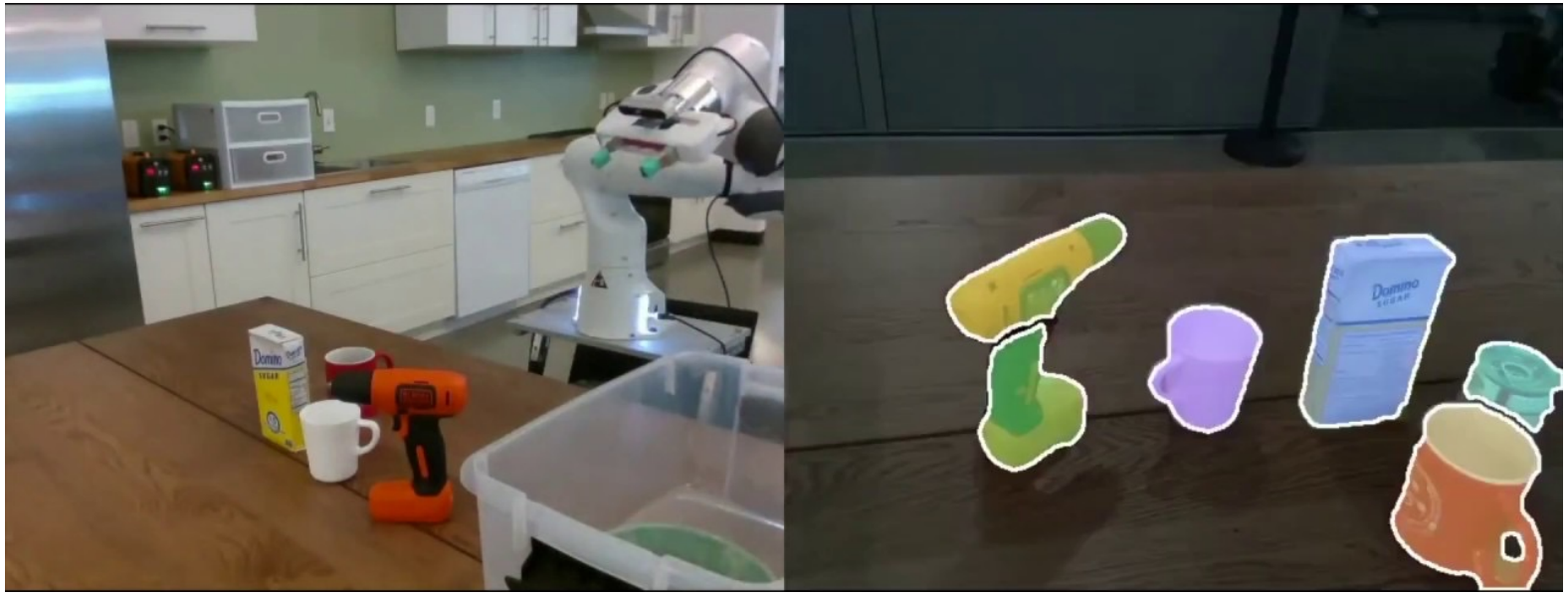


Material from:

- Vincent Sitzmann; Bill Freeman; Mina Konaković Luković MIT CSAIL Advances in Computer Vision Spring 2023 http://6.8300.csail.mit.edu/sp23/schedule.html
- Tushar B. Kute: http://tusharkute.com/

Motivation

- Need to find objects & humans
- Need to get 3D locations
- Need to get shape of objects
- Need to get object properties, such as color, material, weight
- Need to describe objects with natural language
- Need to identify objects based on language



Object Recognition

- Image classification:
  - Image with single object ->
  - Generate one label
- Object Localization:
  - Image with multiple objects ->
  - Generate multiple bounding boxes around objects
- Object Detection:
  - Assign class to bounding box – with confidence
- Object Segmentation:
  - Mark all pixels belonging to one class
  - Instance-level segmentation:
    Distinguish the different instances (e.g. 2 cars) with different color pixels

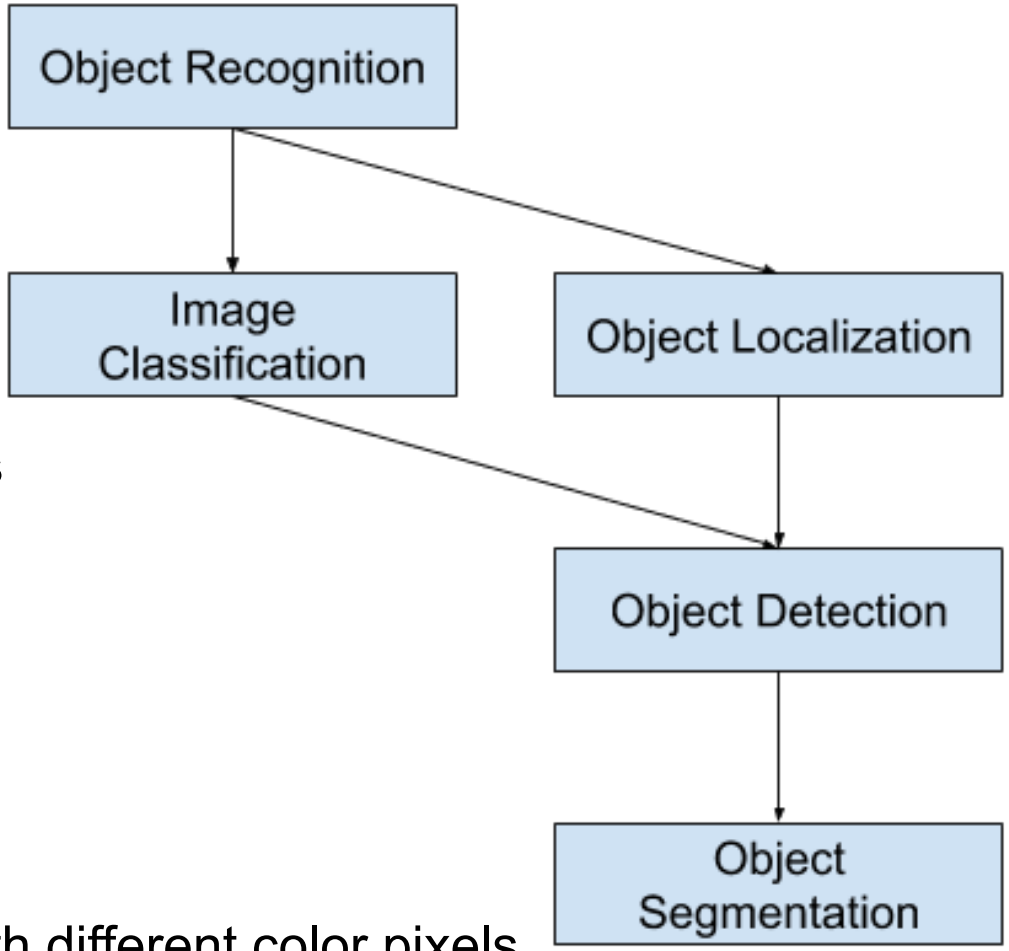


Image Classification: shortcomings

What is a car?

The data represents the formulation of the problem.

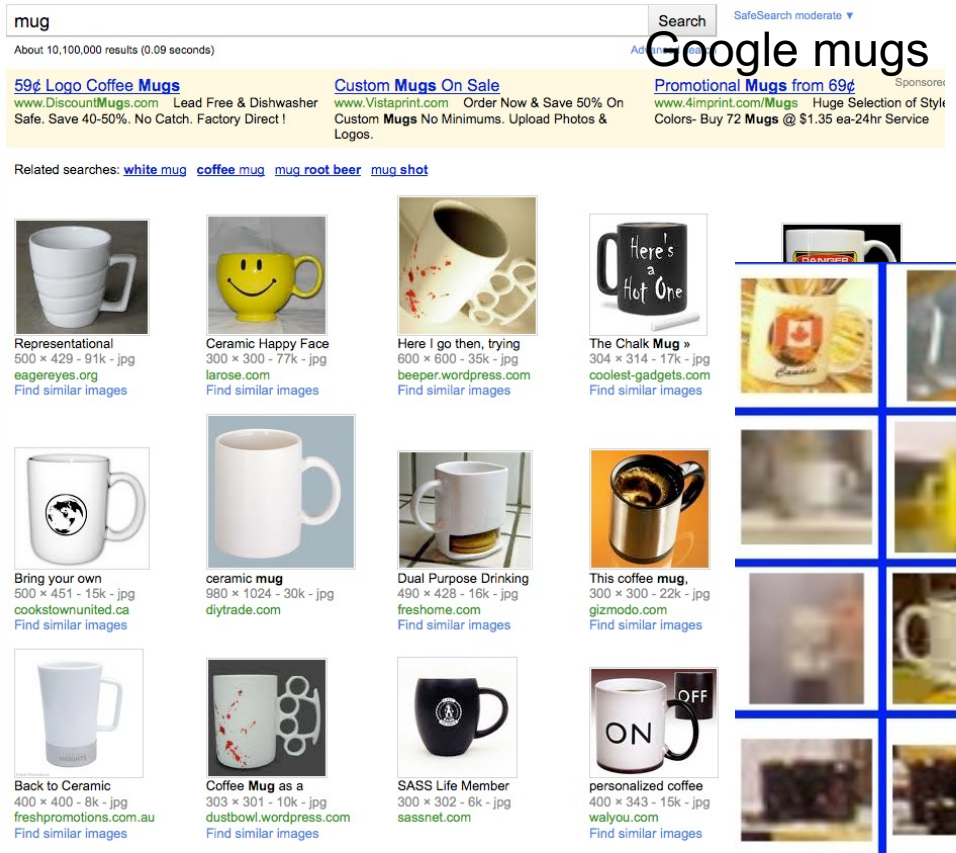
There is ambiguity even for very familiar concepts

View typicality -> canonical perspective

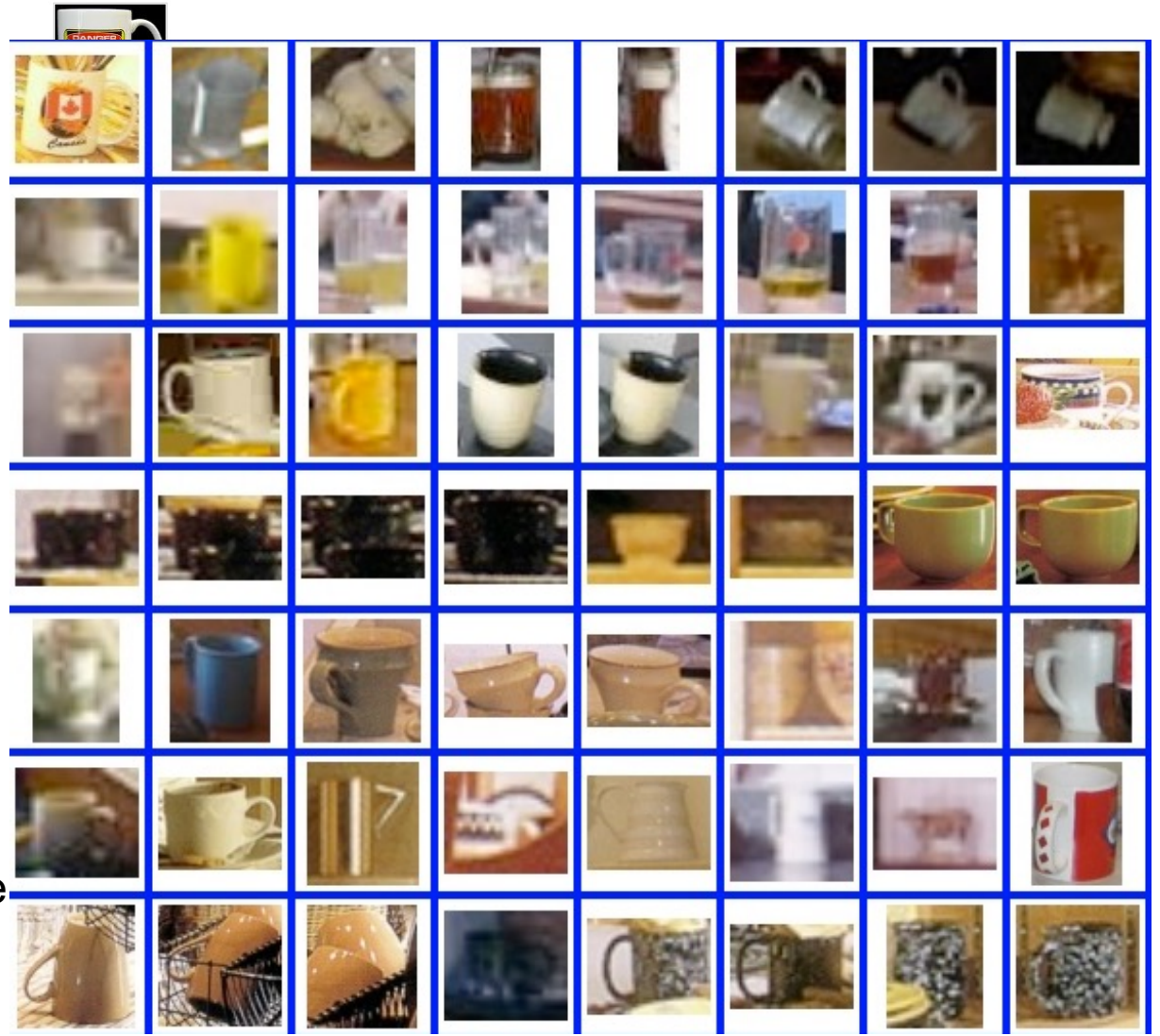
Despite we can categorize all three pictures as being views of a horse, the three pictures do not look as being equally typical views of horses. And they do not seem to be recognizable with the same easiness.

Dataset biases

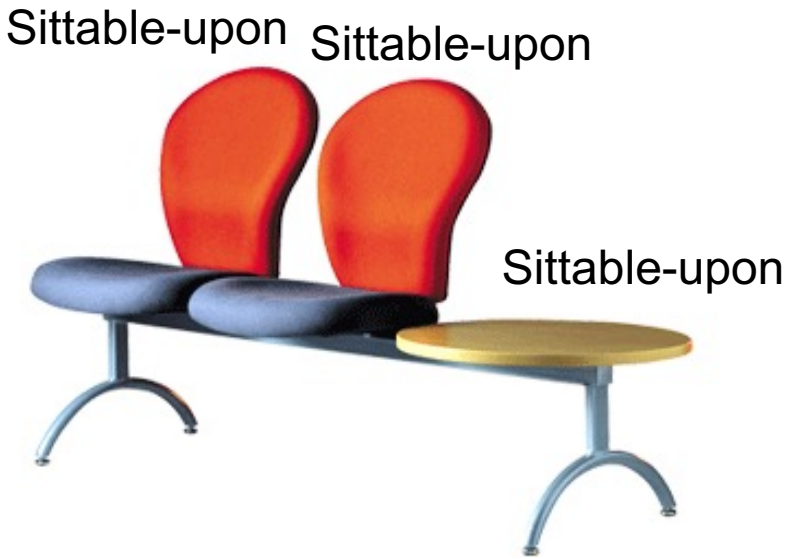
Google mugs



Mugs from LabelMe



What is a chair?



It does not seem easy to sit-upon this…

Some aspects of an object function can be perceived directly

- Functional form: Some forms clearly indicate to a function (“sittable-upon”, container, cutting device, …)

Image classification

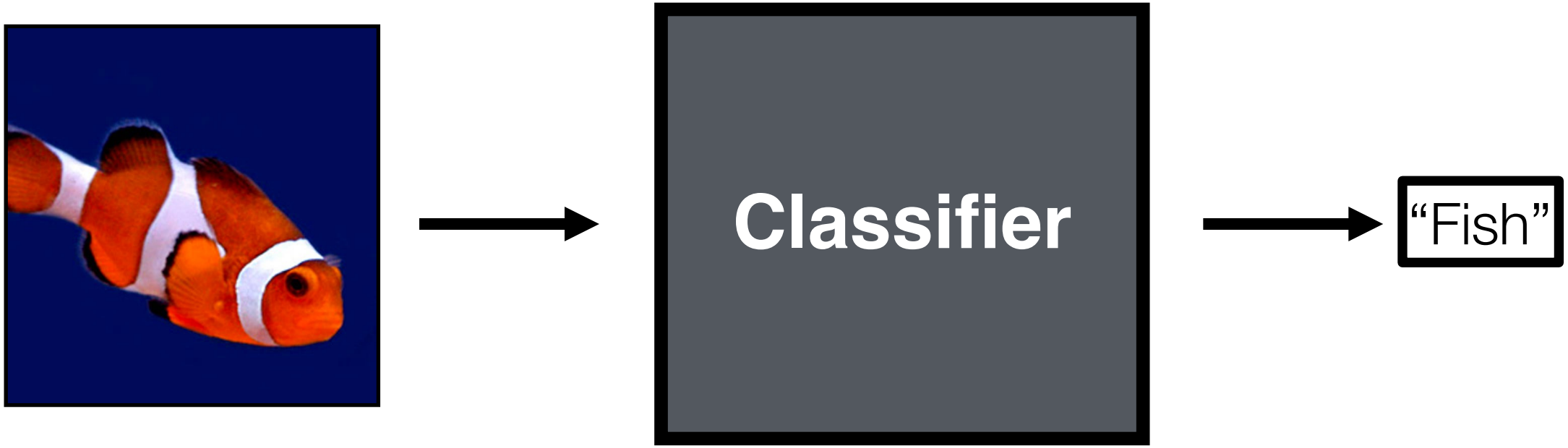
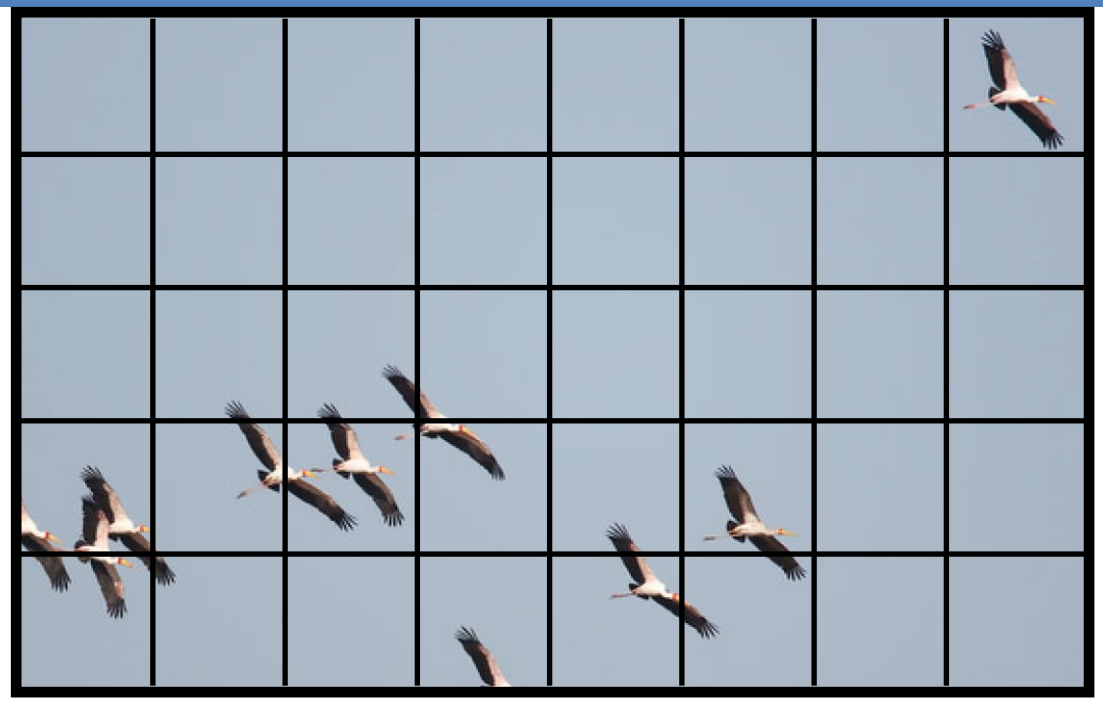
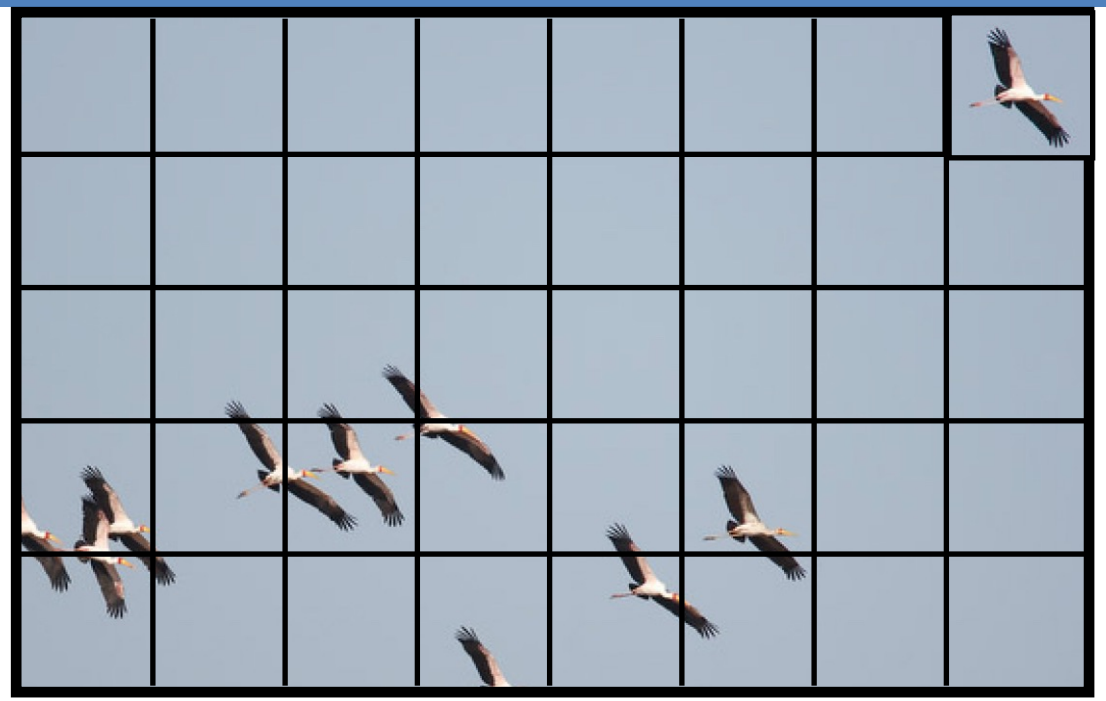


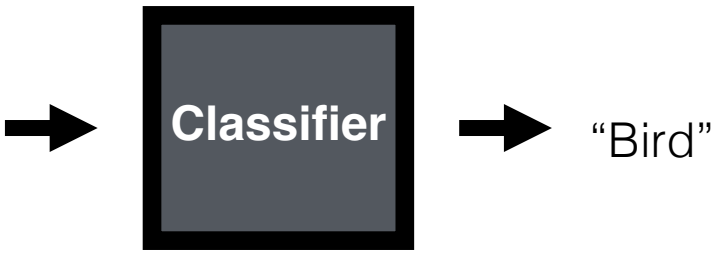
image **x**

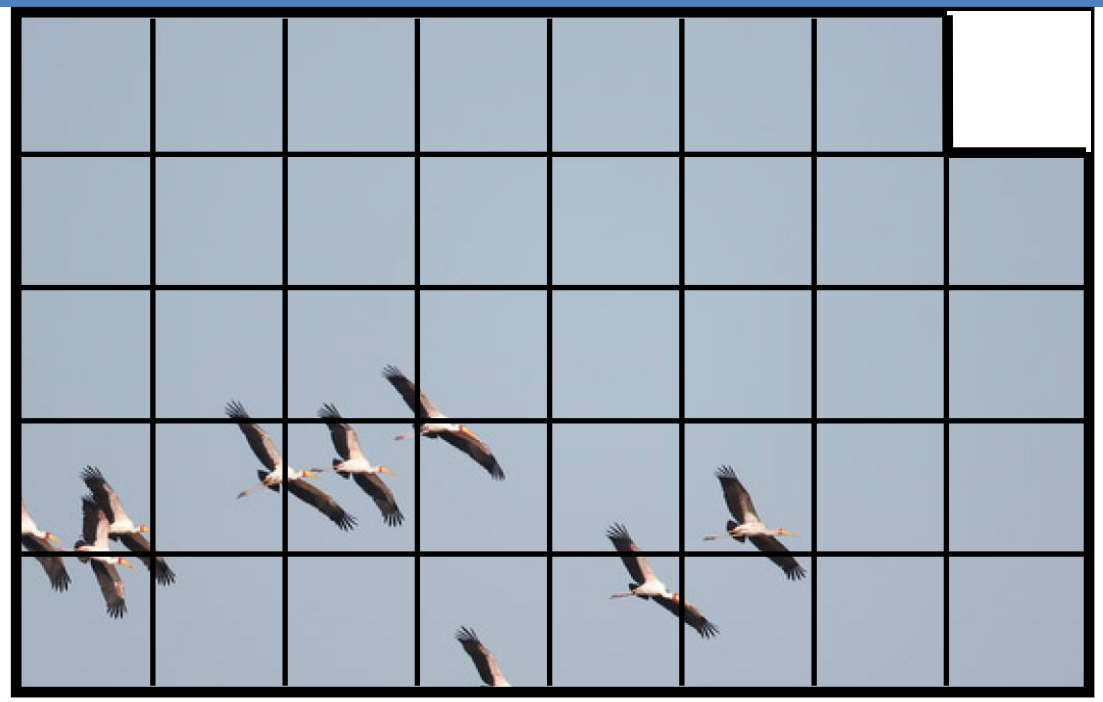
label y

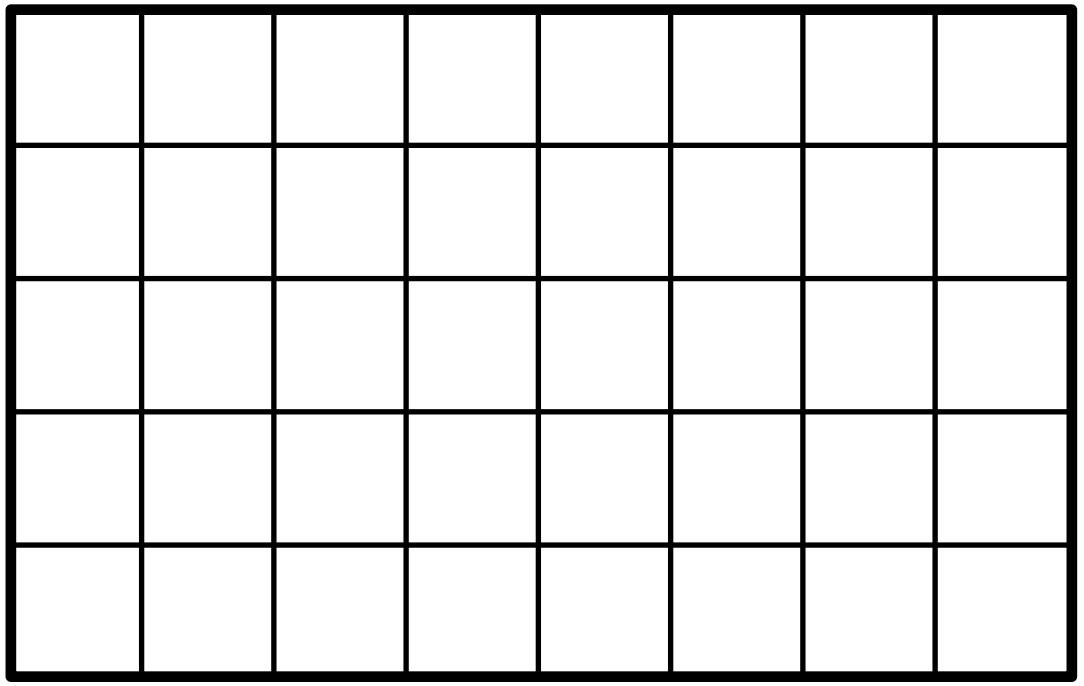
Photo credit: Fredo Durand



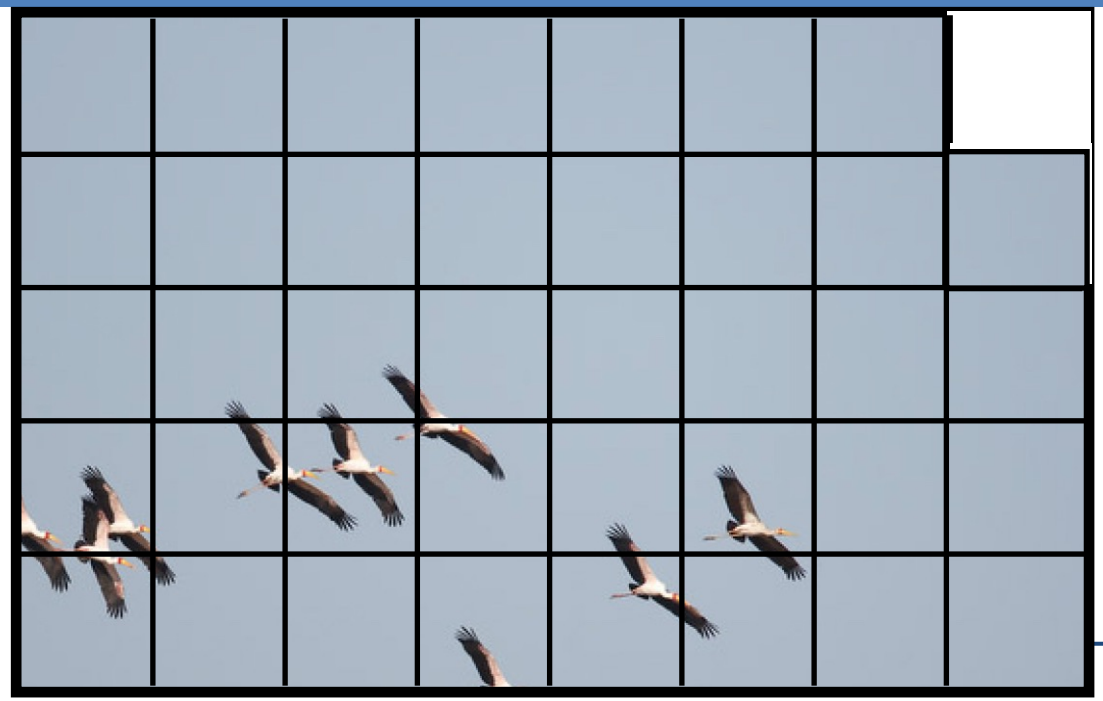


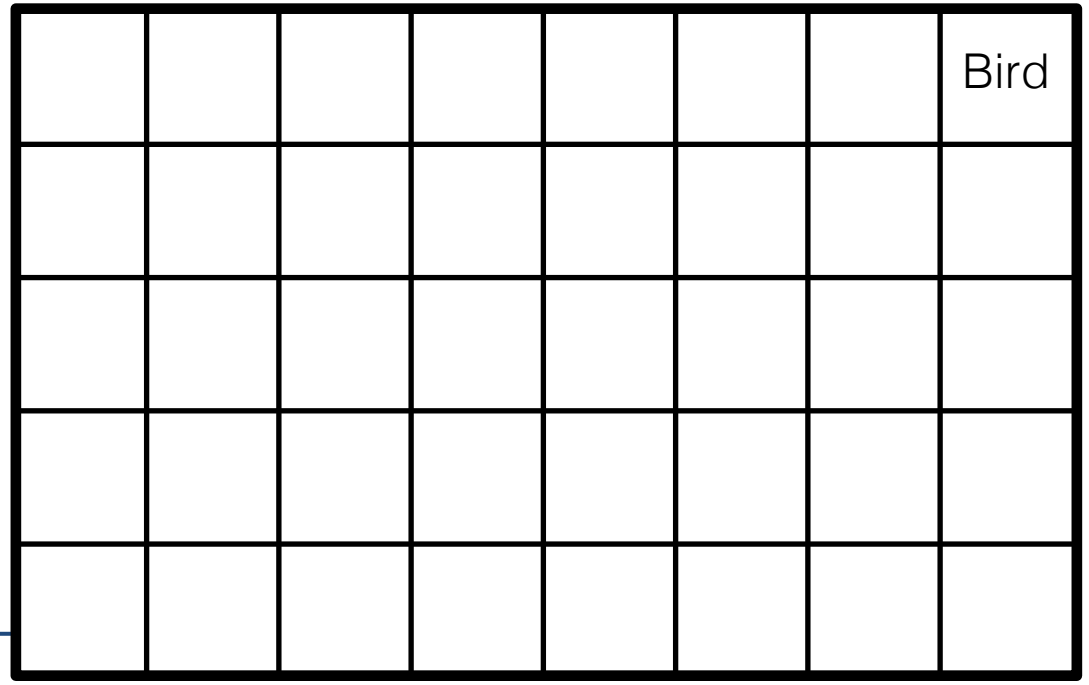

Classifier
“Bird”



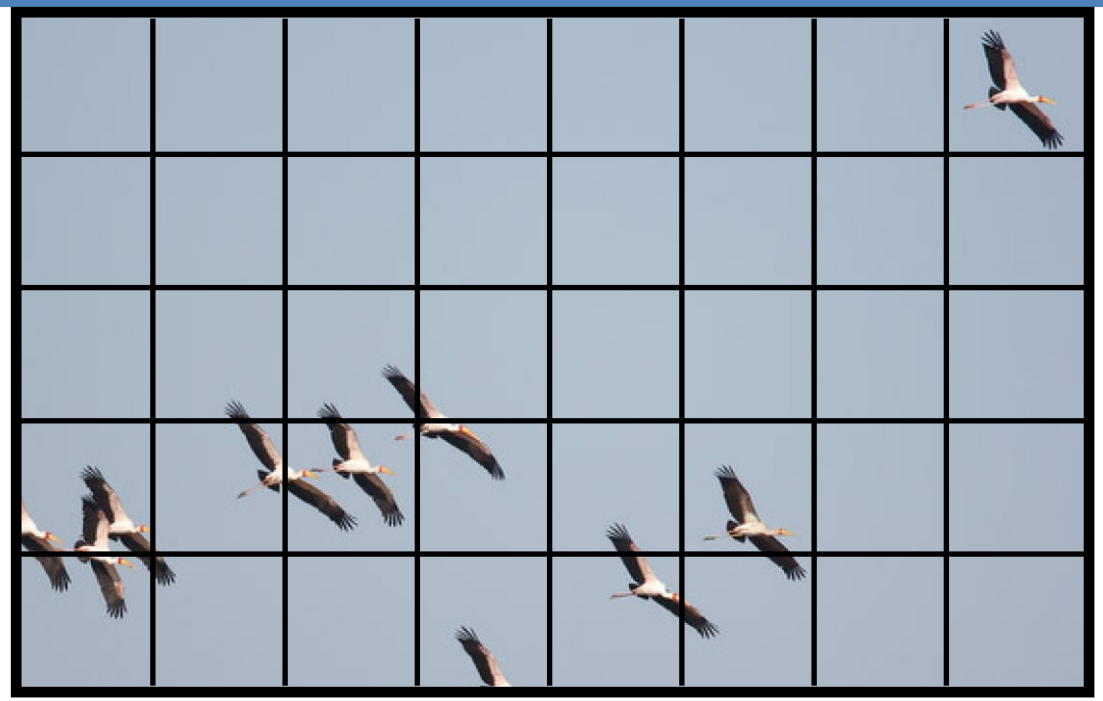



Classifier
“Bird”

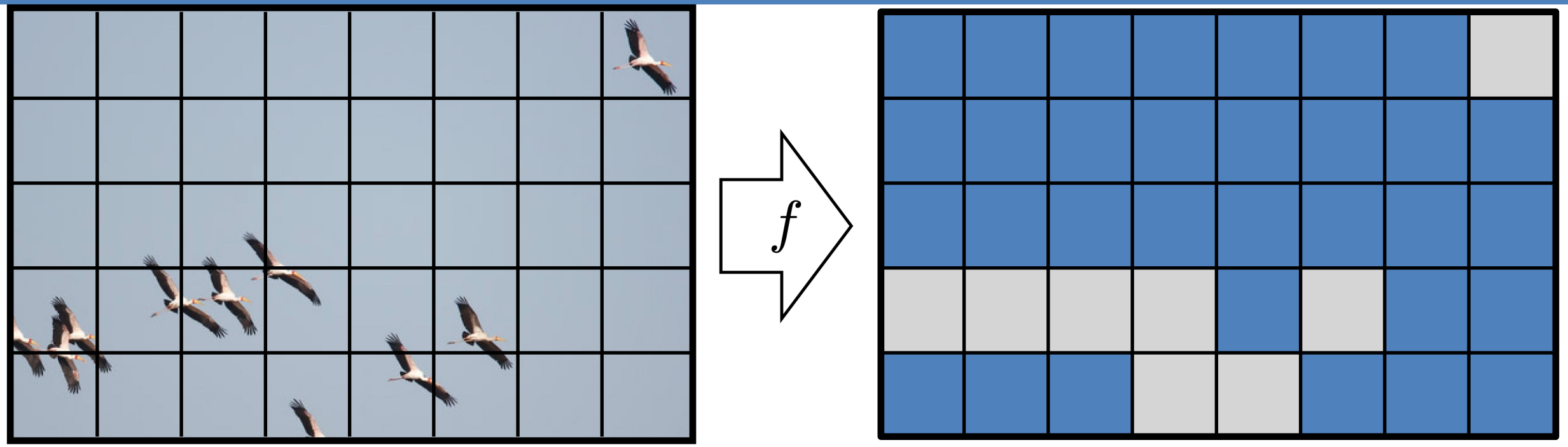



Bird


Classifier
“Sky”



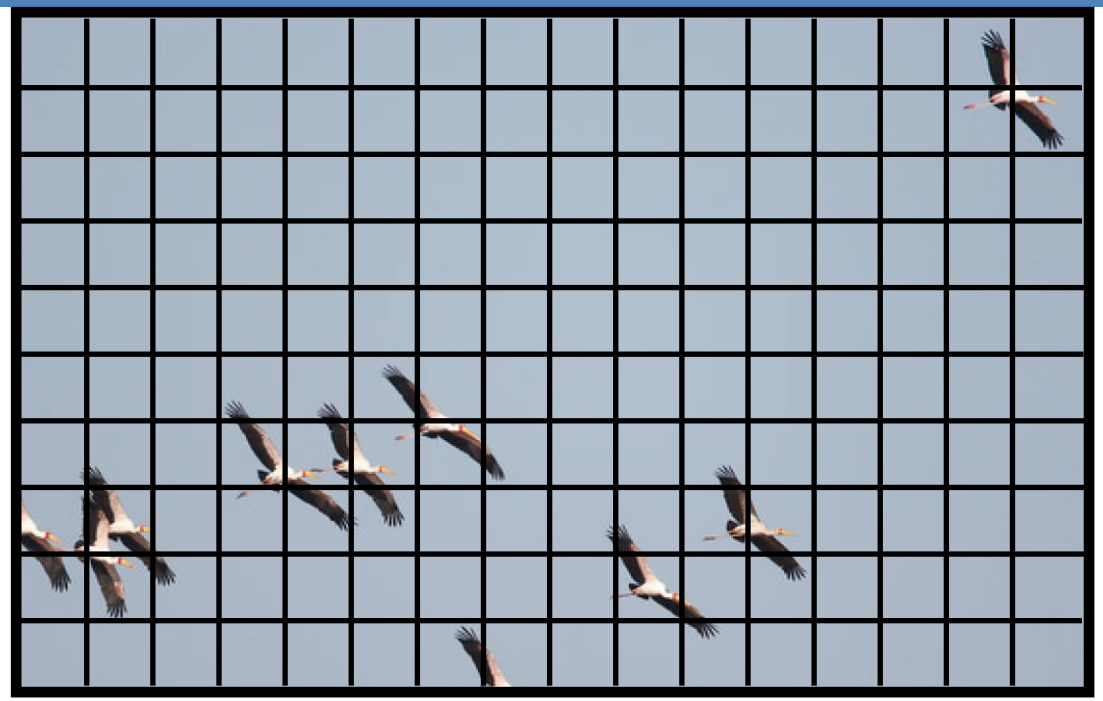
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Bird
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Sky
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Sky
Bird	Bird	Bird	Sky	Bird	Sky	Sky	Sky
Sky	Sky	Sky	Bird	Sky	Sky	Sky	Sky



Problem:

What happens to objects that are bigger?

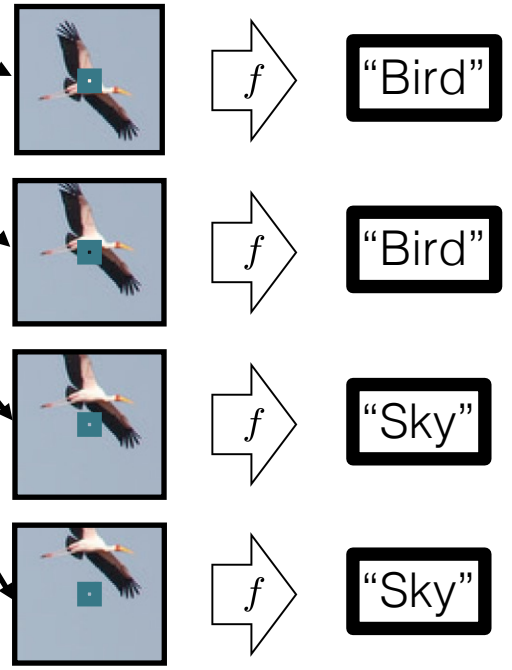
What if an object crosses multiple cells?

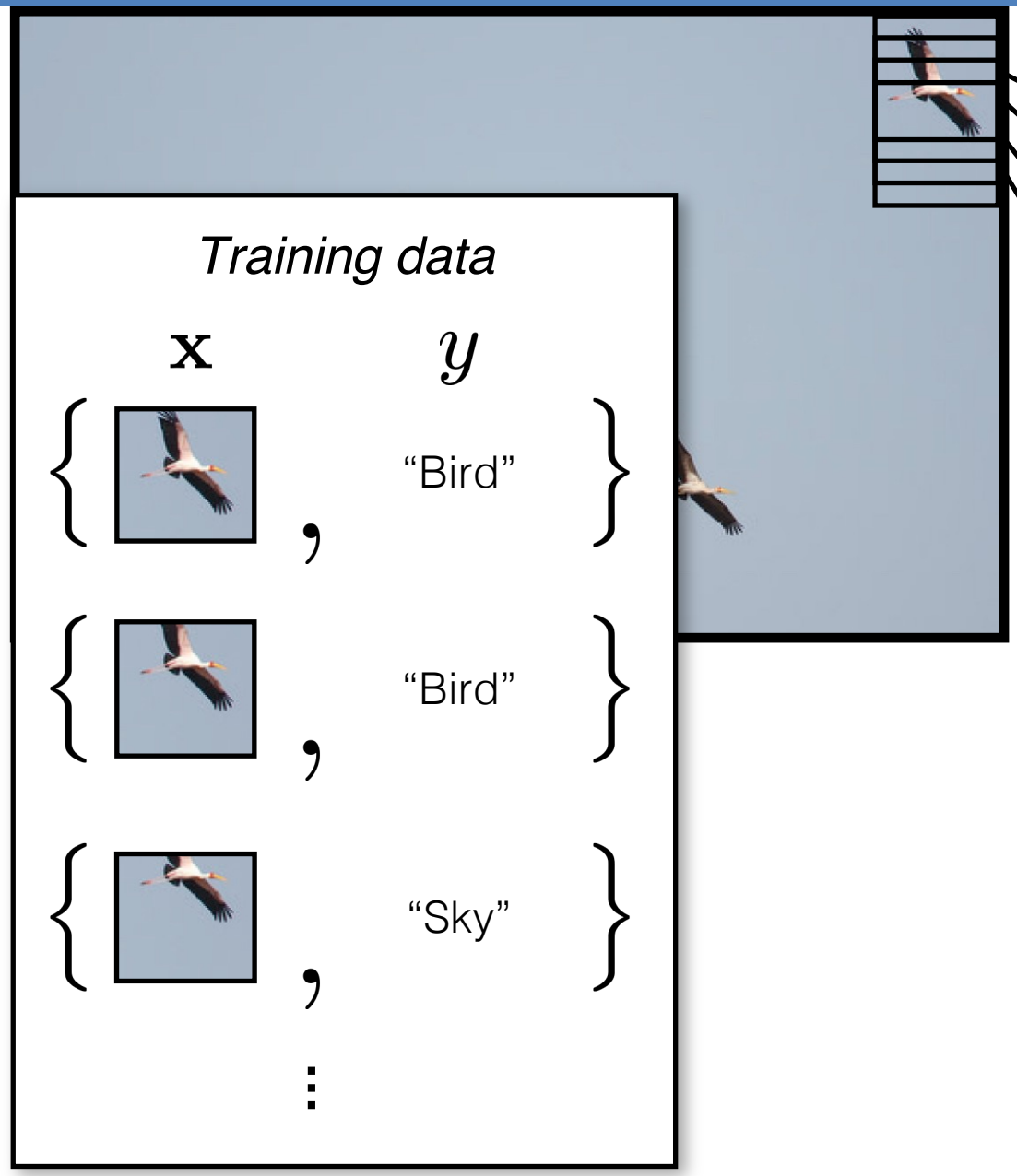


“Cell”-based approach is limited.

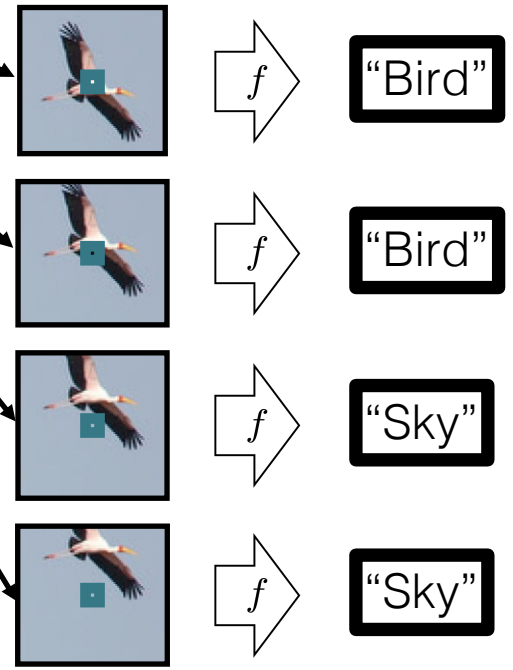
What can we do instead?

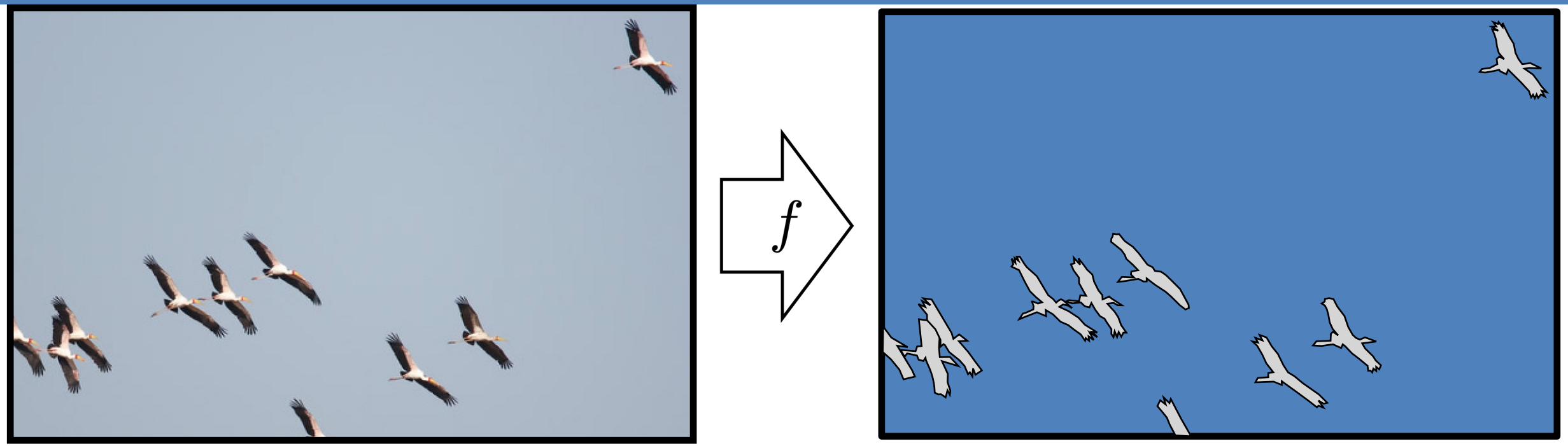
What's the object class of the center pixel?





What’s the object class of the center pixel?

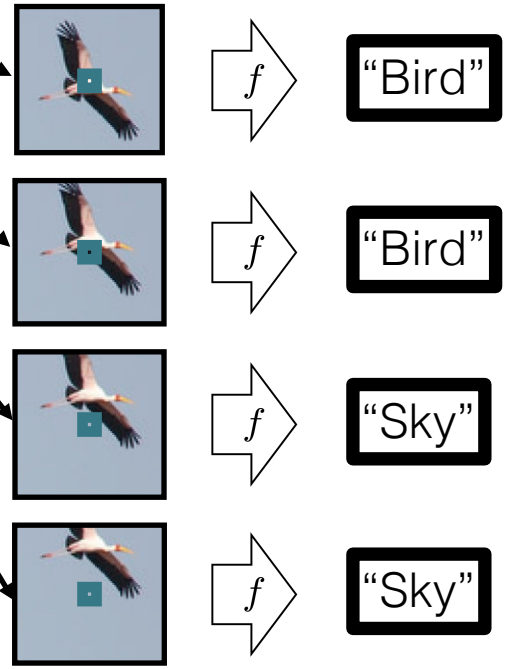




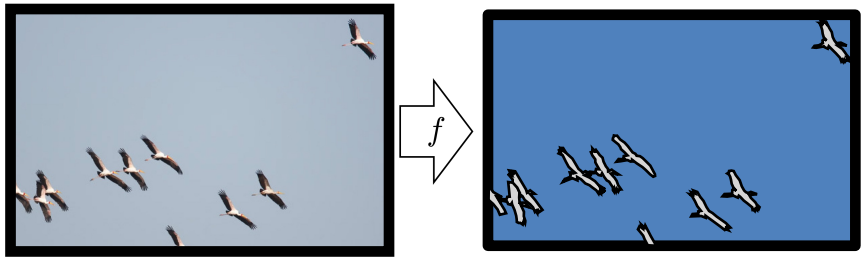
(Colors represent one-hot codes)

This problem is called **semantic segmentation**

What's the object class of the center pixel?



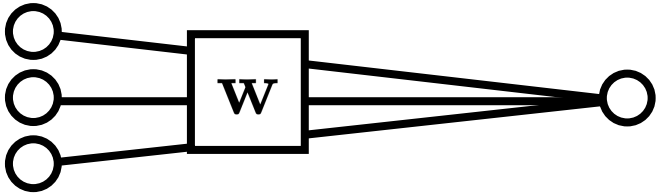
Translation invariance: process each patch in the same way.

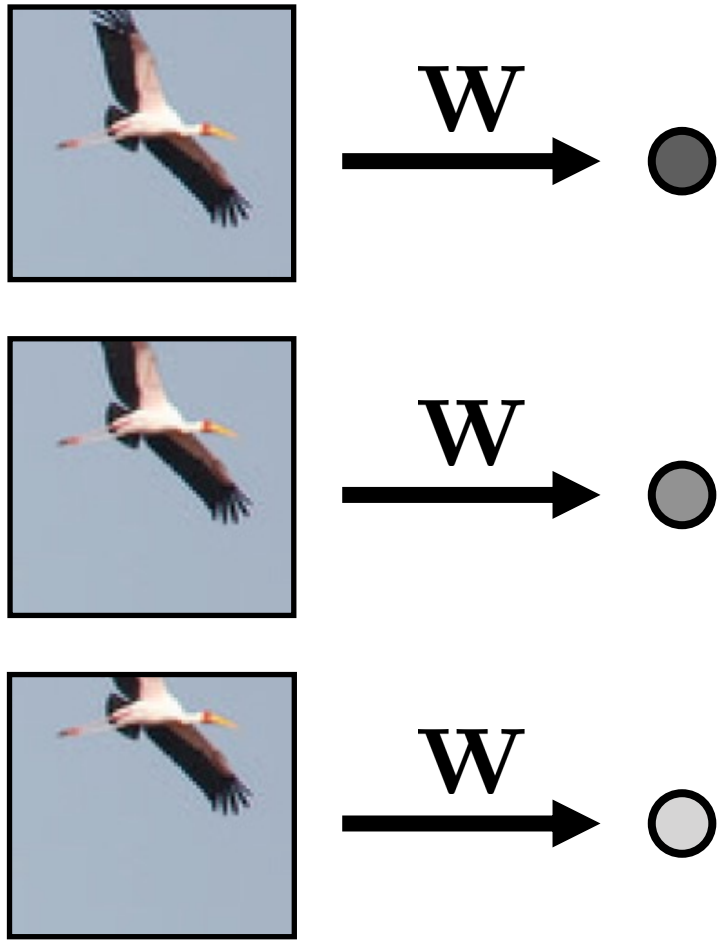


An *equivariant* mapping:

$$f(\texttt{translate}(x)) = \texttt{translate}(f(x))$$

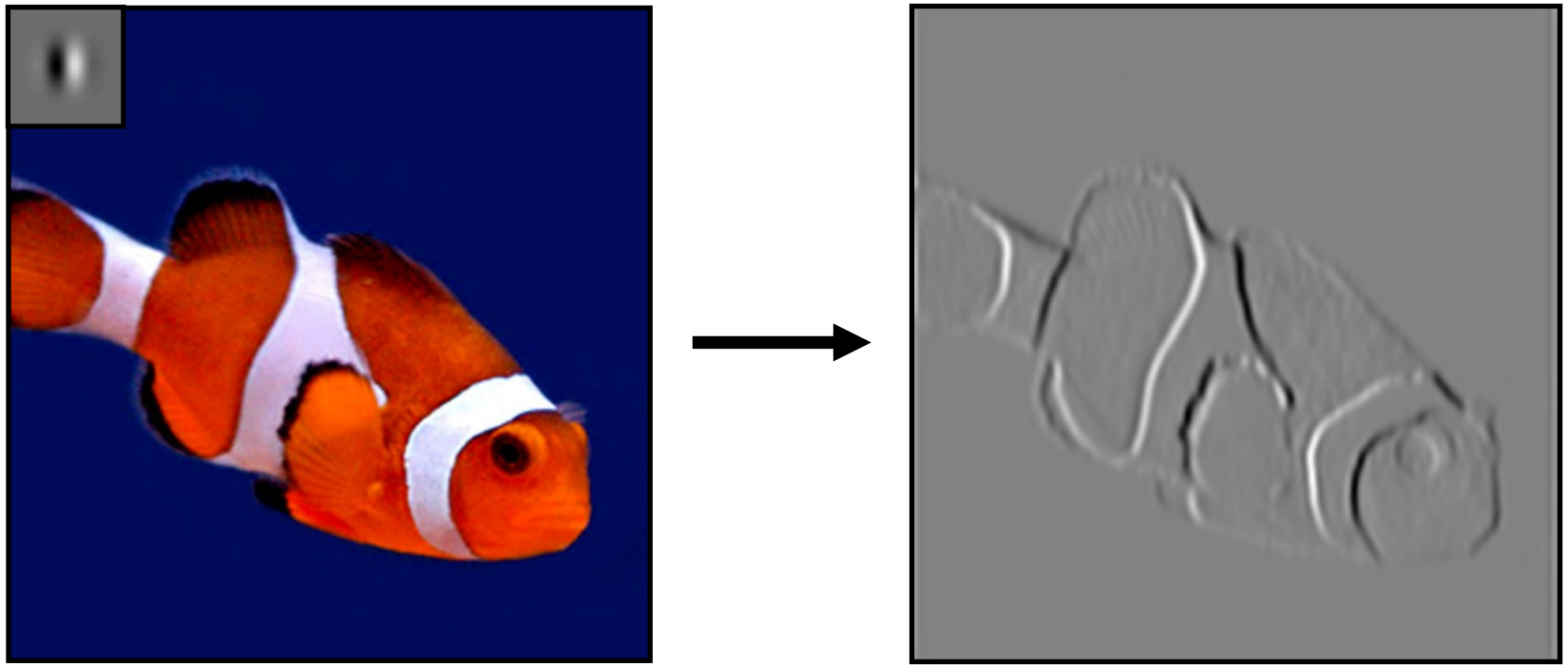
W computes a weighted sum of all pixels in the patch



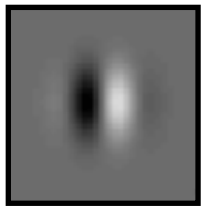


W is a convolutional kernel applied to the full image!

Convolution

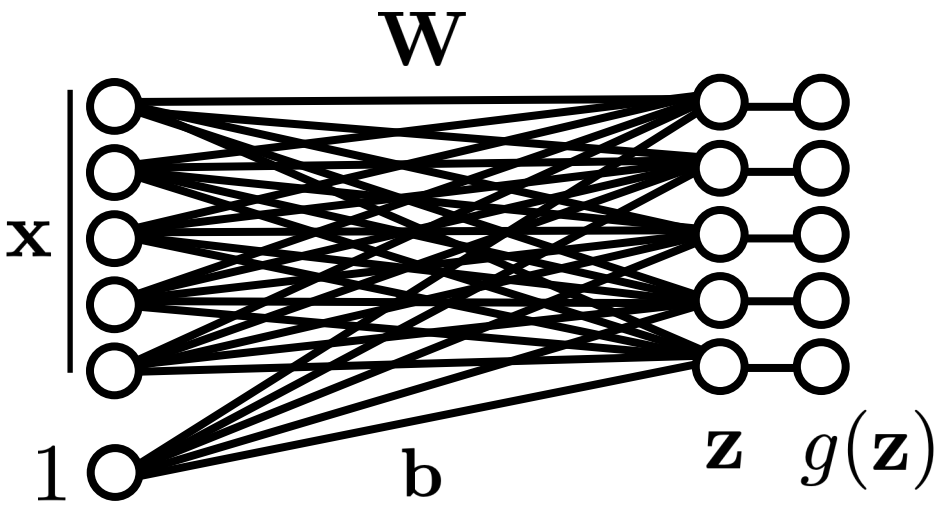


filter

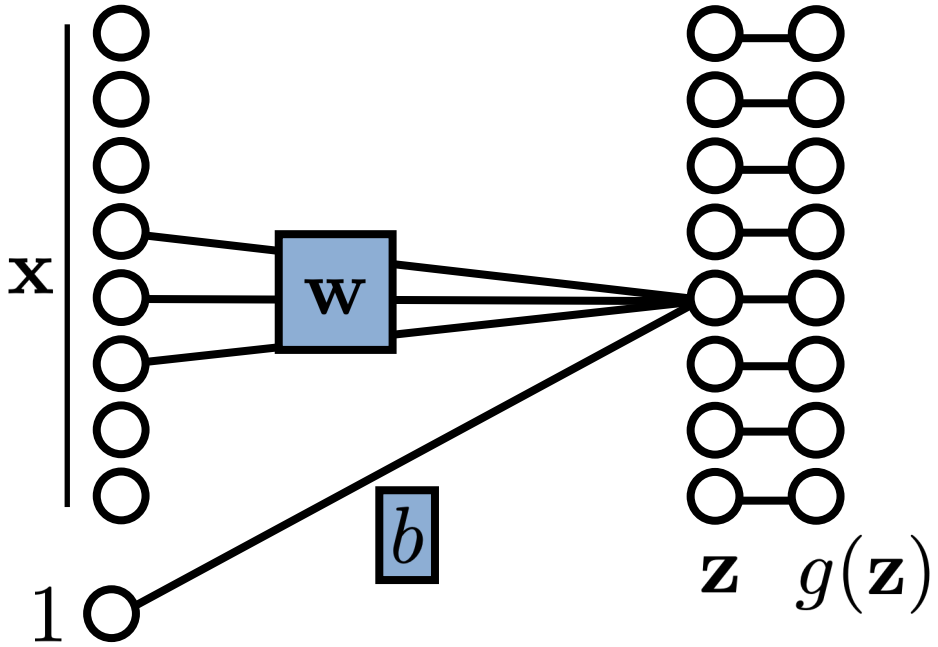


Fully-connected network

Fully-connected (fc) layer



Locally connected network

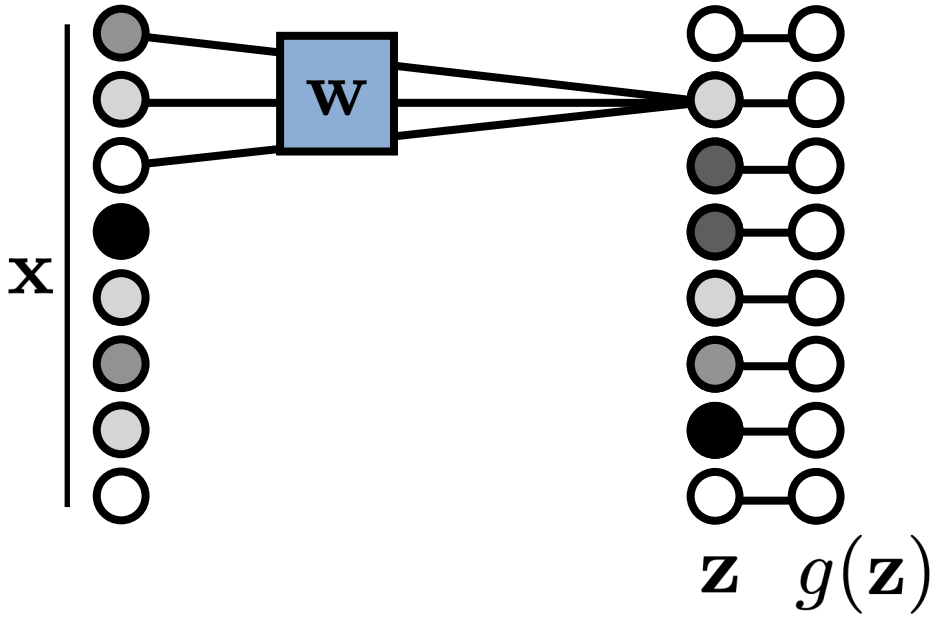


Often, we assume output is a **local** function of input.

If we use the same weights (**weight sharing**) to compute each local function, we get a convolutional neural network.

Convolutional neural network

Conv layer



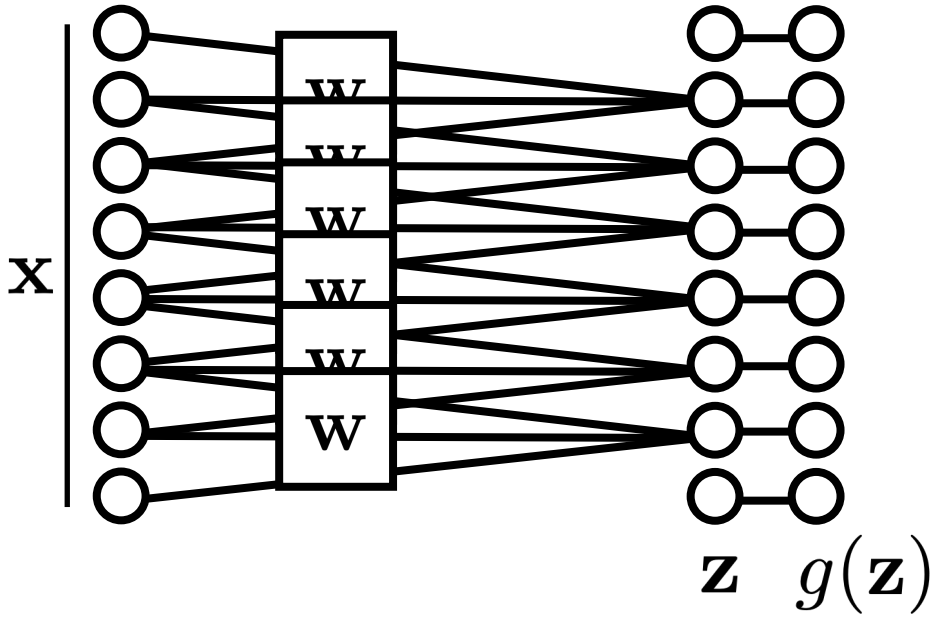
$$\mathbf{z} = \mathbf{w} \circ \mathbf{x} + \mathbf{b}$$

Often, we assume output is a **local** function of input.

If we use the same weights (**weight sharing**) to compute each local function, we get a convolutional neural network.

Weight sharing

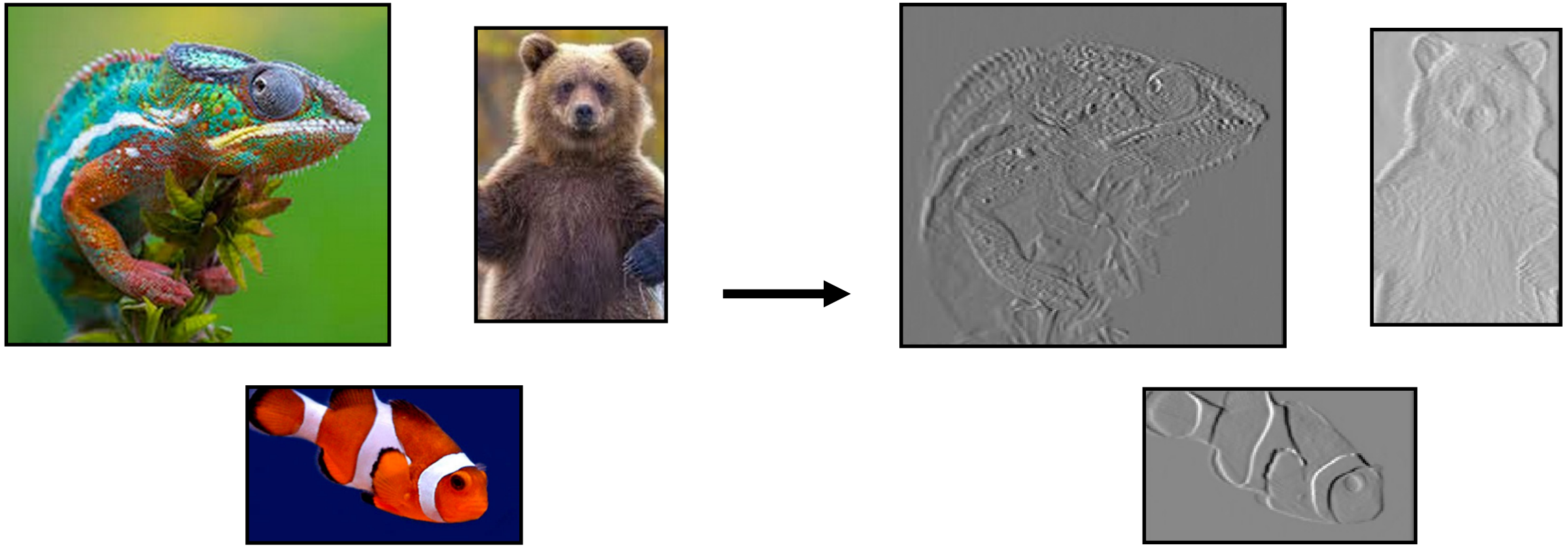
Conv layer



$$\mathbf{z} = \mathbf{w} \circ \mathbf{x} + \mathbf{b}$$

Often, we assume output is a **local** function of input.

If we use the same weights (**weight sharing**) to compute each local function, we get a convolutional neural network.

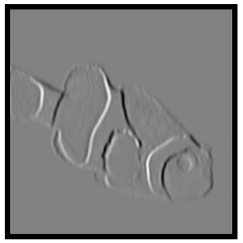


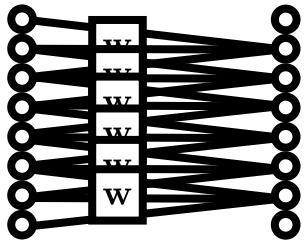
Conv layers can be applied to arbitrarily-sized inputs

Five views on convolutional layers

1. Equivariant with translation $f(\texttt{translate}(x)) = \texttt{translate}(f(x))$

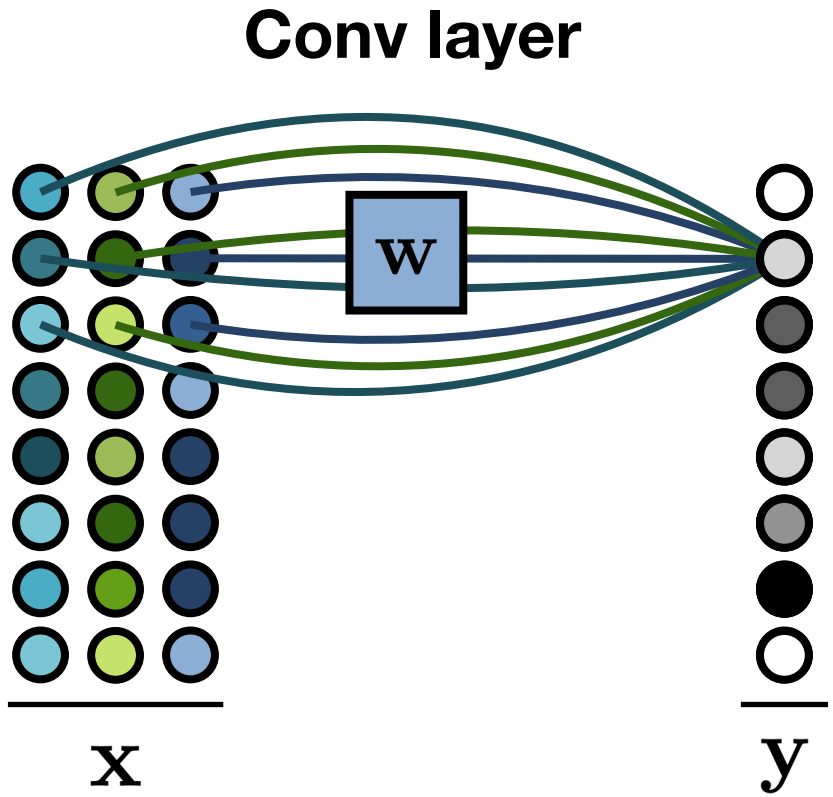
2. Patch processing

3. Image filter 

4. Parameter sharing 

5. A way to process variable-sized tensors

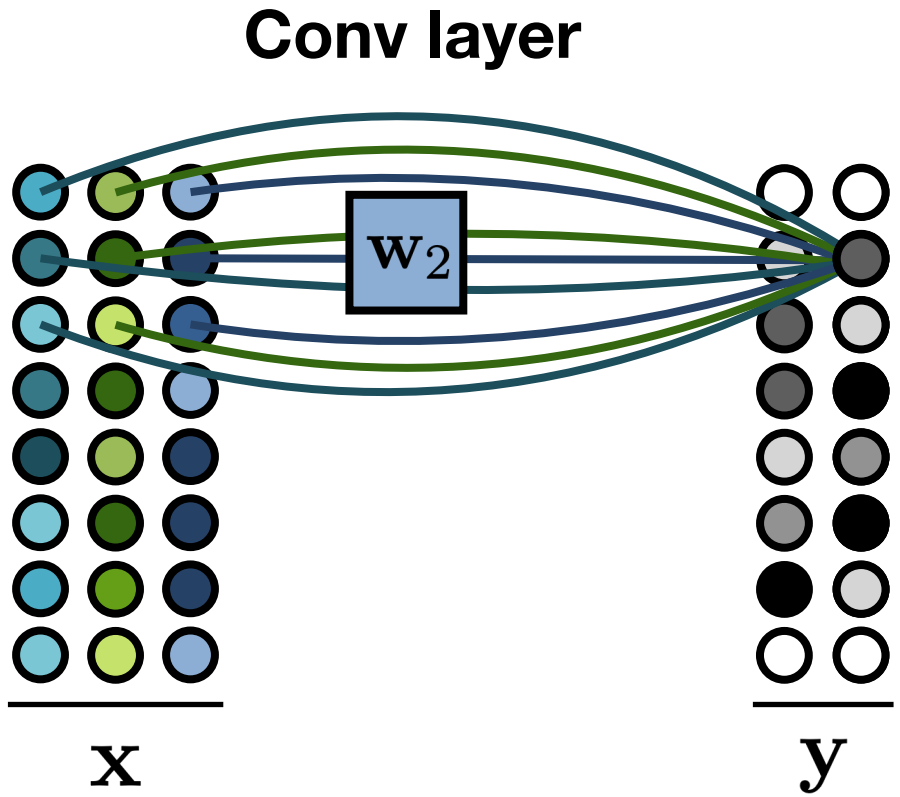
Multiple channel inputs



$$\mathbf{y} = \sum_c \mathbf{w}_c \circ \mathbf{x}_c$$

$$\mathbb{R}^{N \times C} \rightarrow \mathbb{R}^{N \times 1}$$

Multiple channel *outputs*

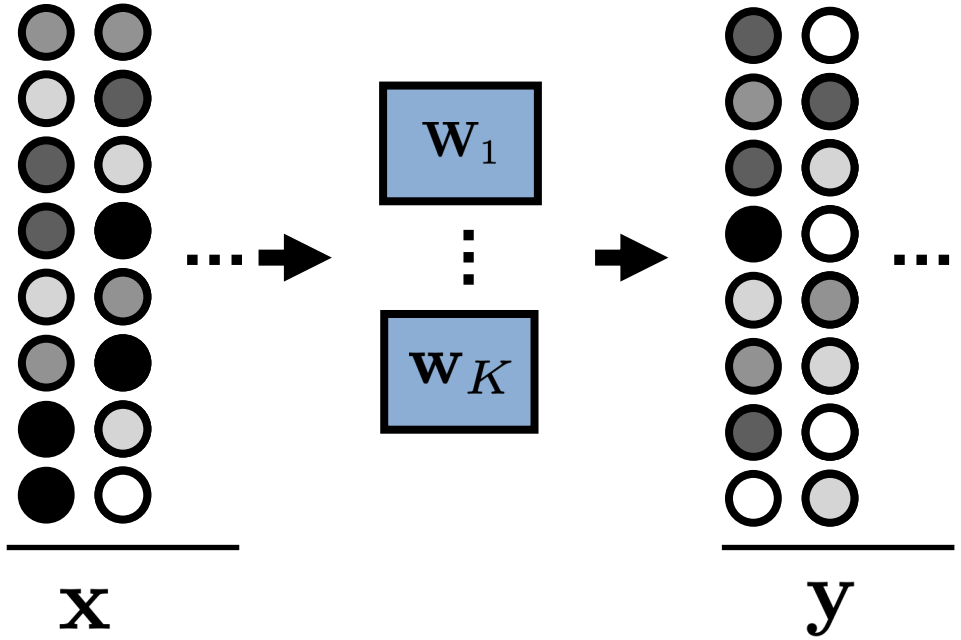


$$\mathbf{y}_k = \sum_c \mathbf{w}_{k_c} \circ \mathbf{x}_c$$

$$\mathbb{R}^{N \times C} \rightarrow \mathbb{R}^{N \times K}$$

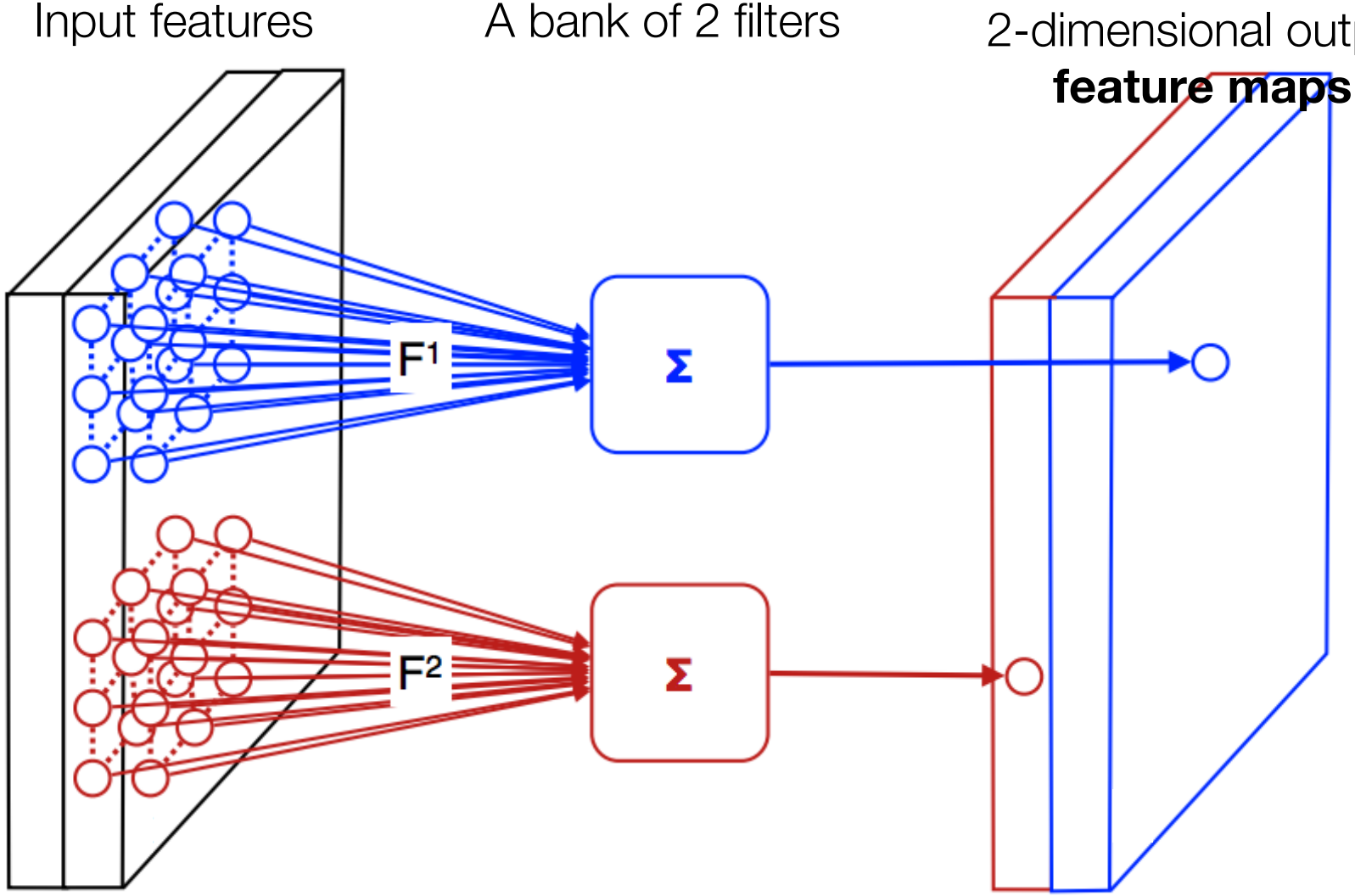
Multiple channels

Conv layer



$$\mathbf{y}_k = \sum_c \mathbf{w}_{k_c} \circ \mathbf{x}_c$$

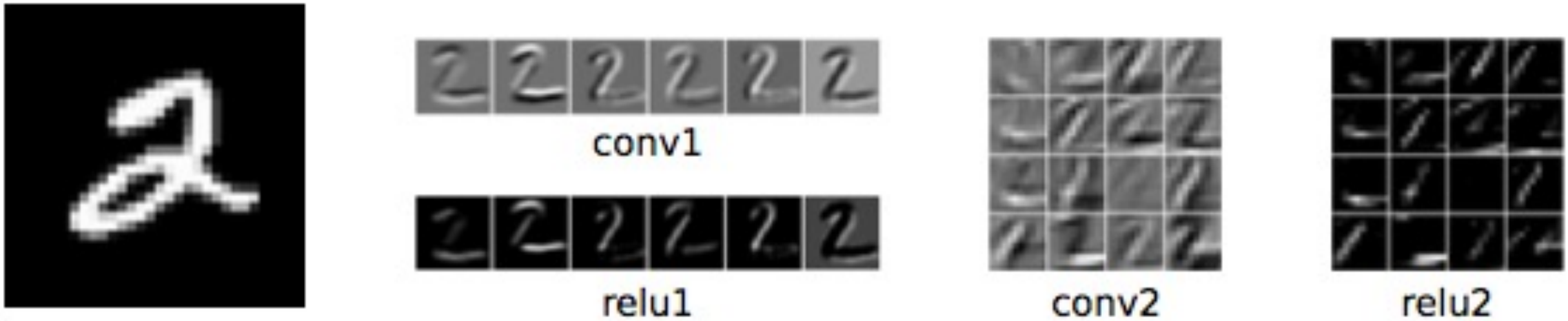
$$\mathbb{R}^{N \times C} \rightarrow \mathbb{R}^{N \times K}$$



$$\mathbf{x}_l \in \mathbb{R}^{H \times W \times C_l} \quad \rightarrow \quad \mathbf{x}_{(l+1)} \in \mathbb{R}^{H \times W \times C_{(l+1)}}$$

[Figure modified from Andrea Vedaldi]

Feature maps



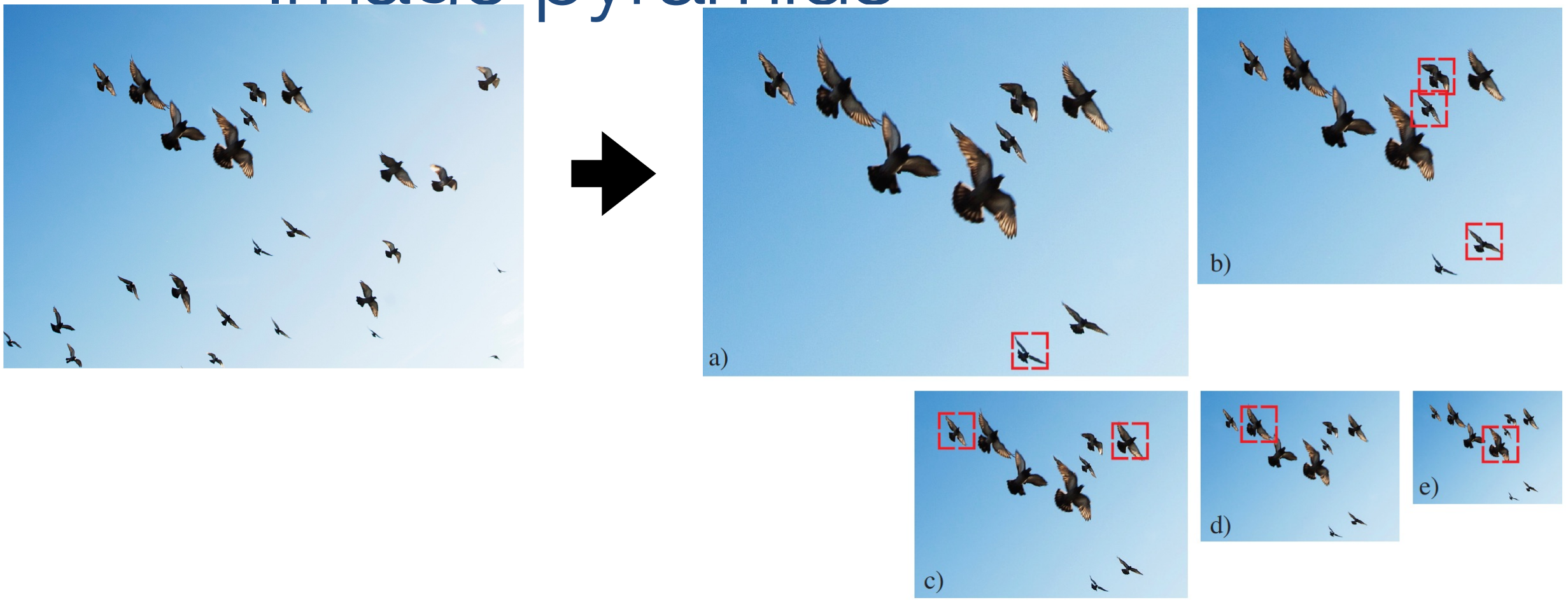
- Each layer can be thought of as a set of C **feature maps** aka **channels**
- Each feature map is an NxM image

Image source: https://stackoverflow.com/questions/45678473/convolution-neural-networks-all-feature-maps-are-blackpixel-value-is-0

Pooling and downsampling

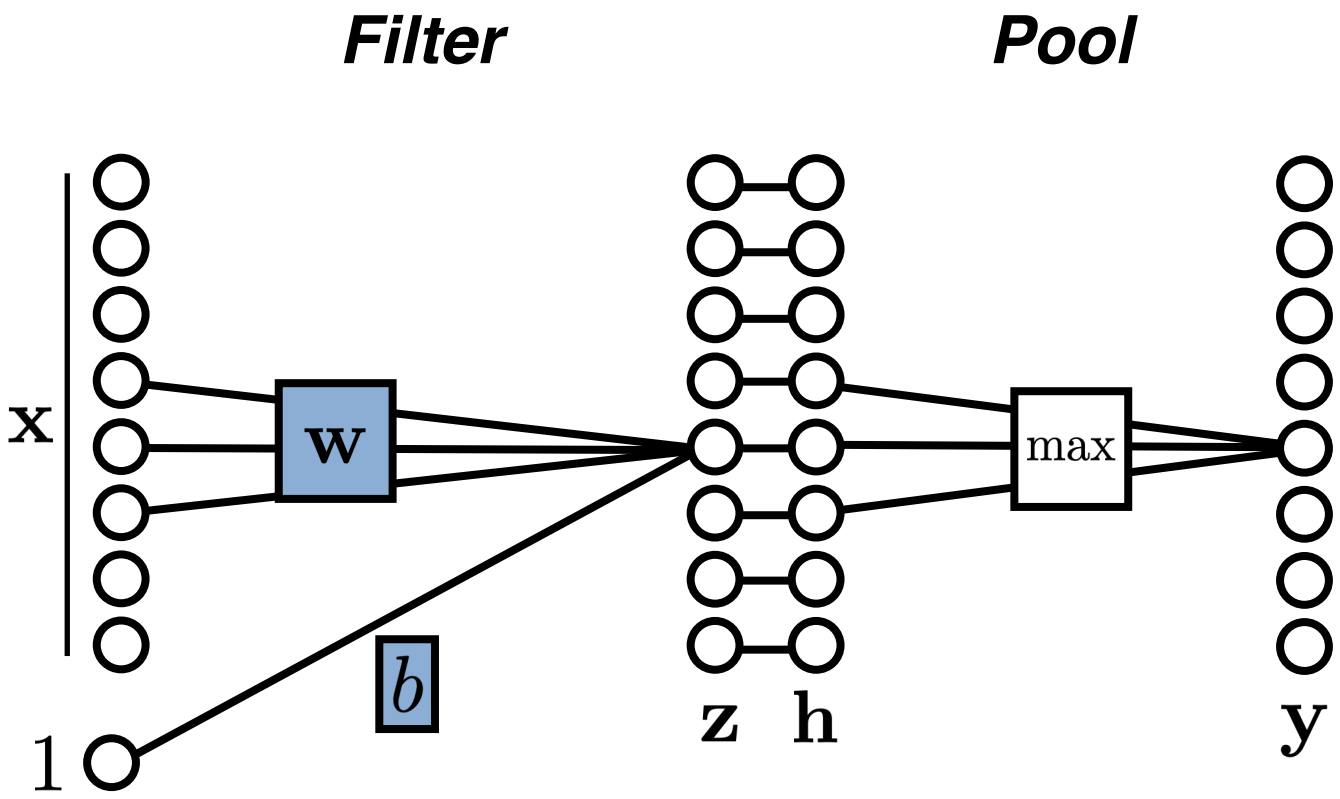
We need translation and **scale** invariance

Image pyramids



Gaussian Pyramid

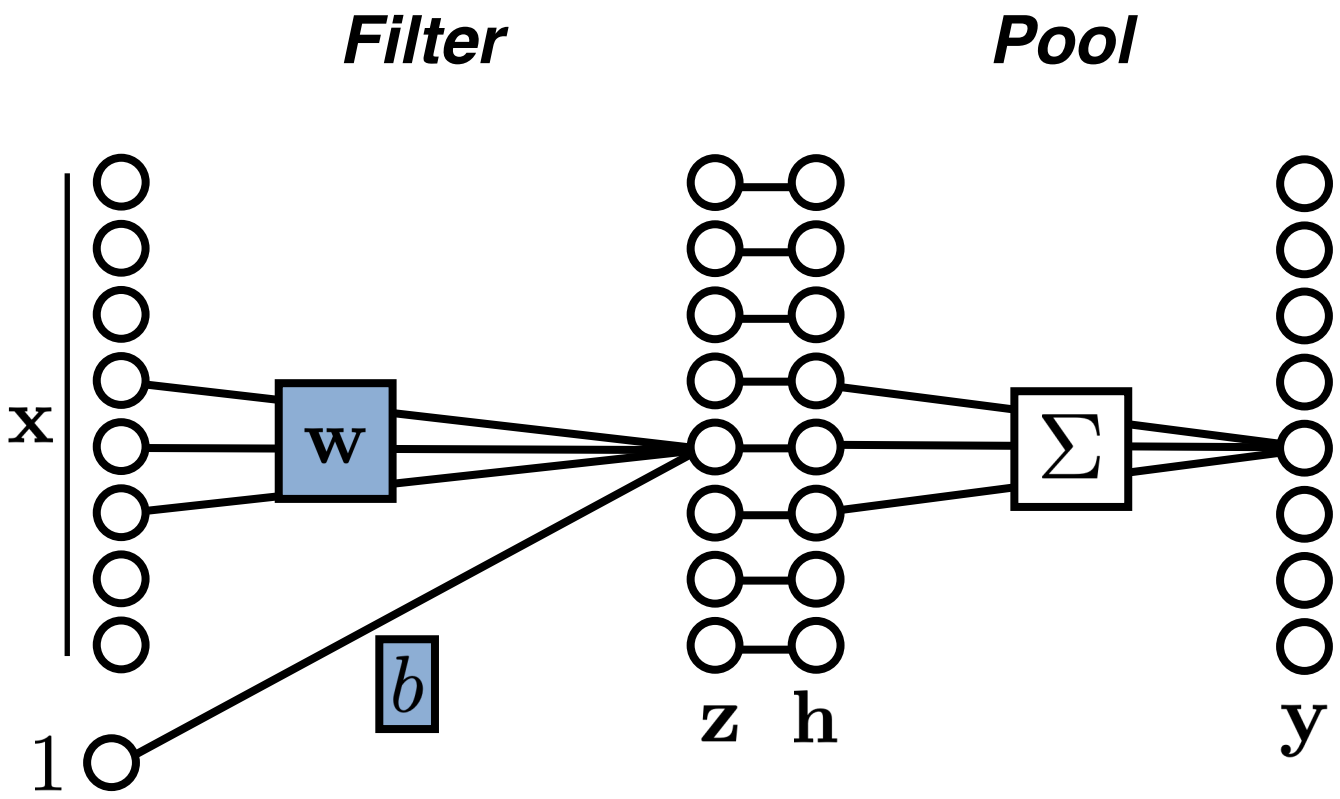
Pooling



Max pooling

$$y_j = \max_{j \in \mathcal{N}(j)} h_j$$

Pooling



Max pooling

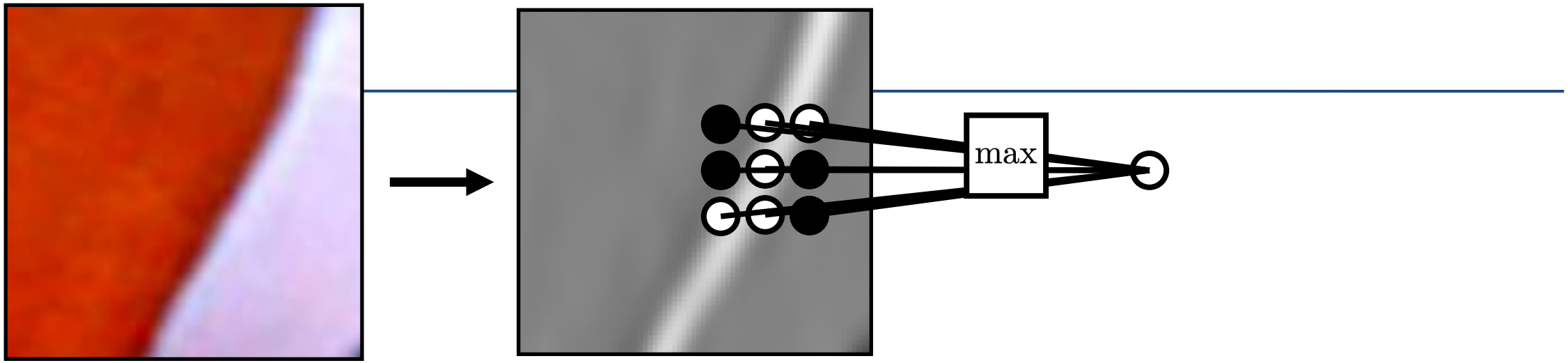
$$y_j = \max_{j \in \mathcal{N}(j)} h_j$$

Mean pooling

$$y_j = \frac{1}{|\mathcal{N}|} \sum_{j \in \mathcal{N}(j)} h_j$$

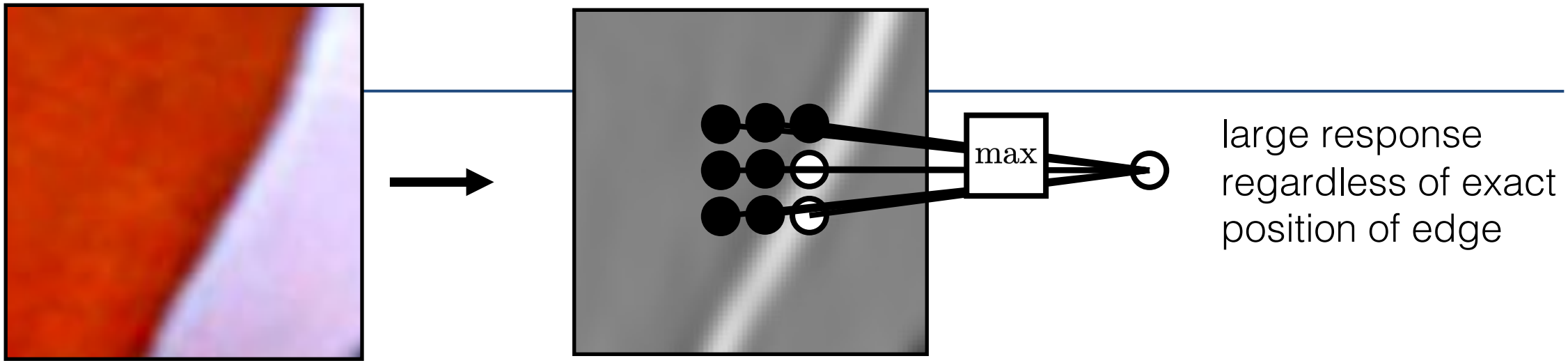
Pooling — Why?

Pooling across spatial locations achieves stability w.r.t. small translations:



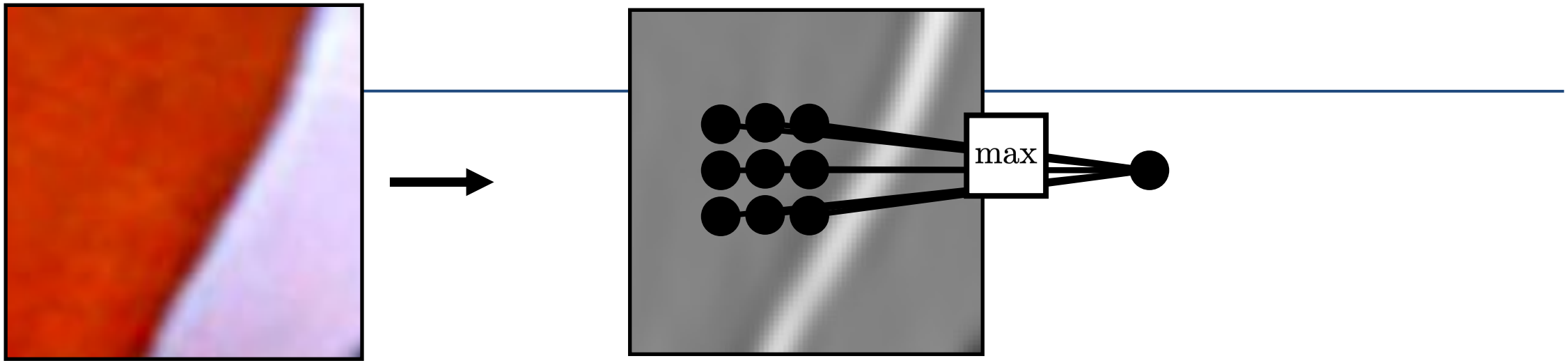
Pooling — Why?

Pooling across spatial locations achieves stability w.r.t. small translations:



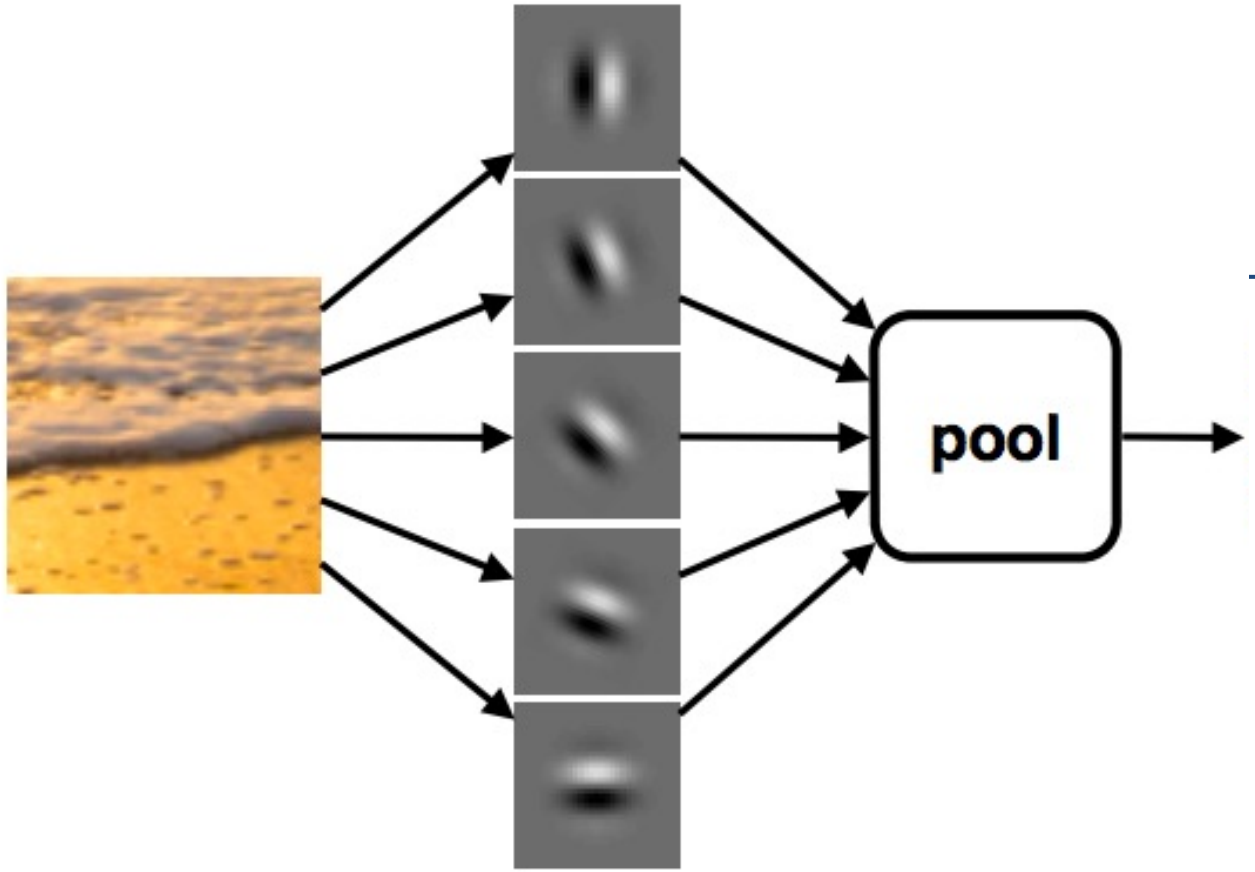
Pooling — Why?

Pooling across spatial locations achieves stability w.r.t. small translations:



Pooling *across channels* — Why?

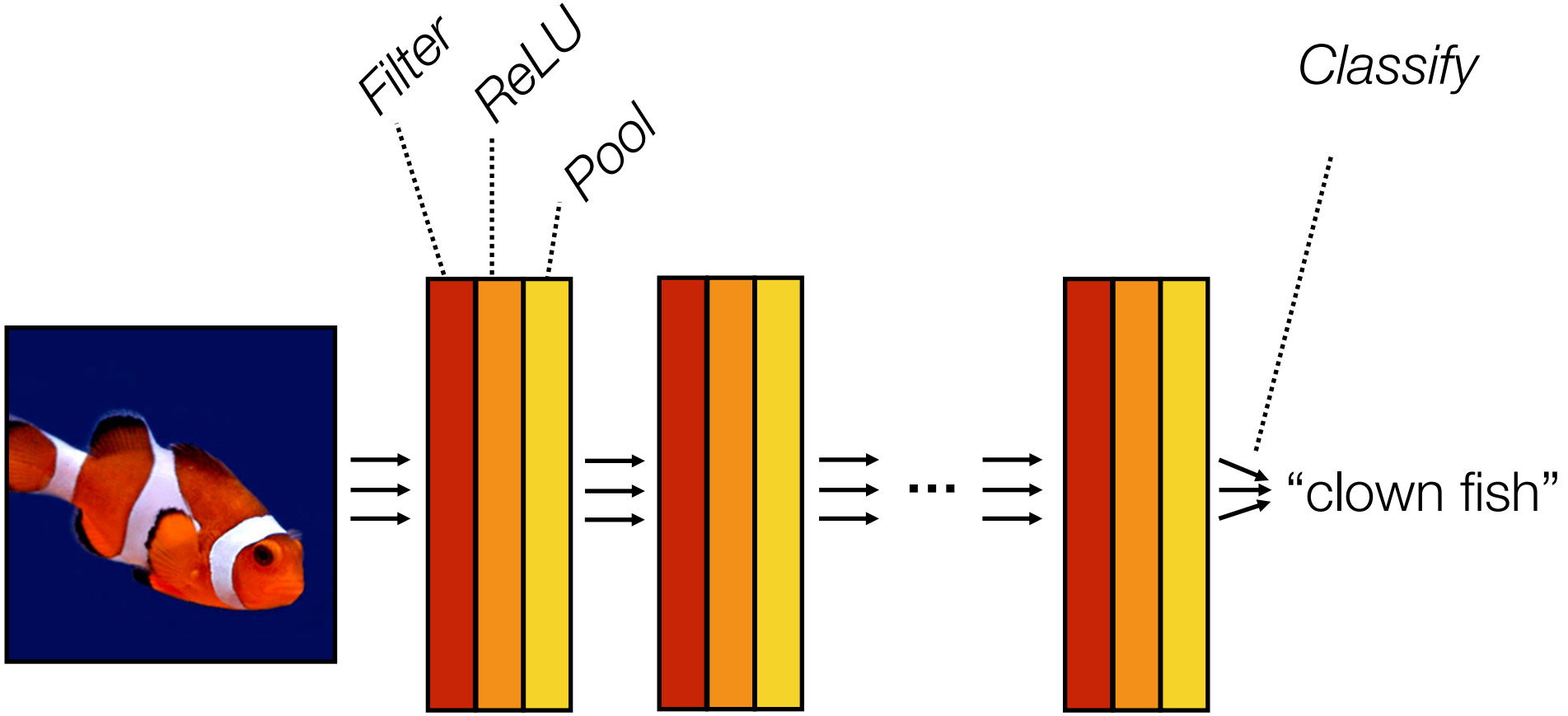
Pooling across feature channels (filter outputs) can achieve other kinds of invariances:



large response for any edge, regardless of its orientation

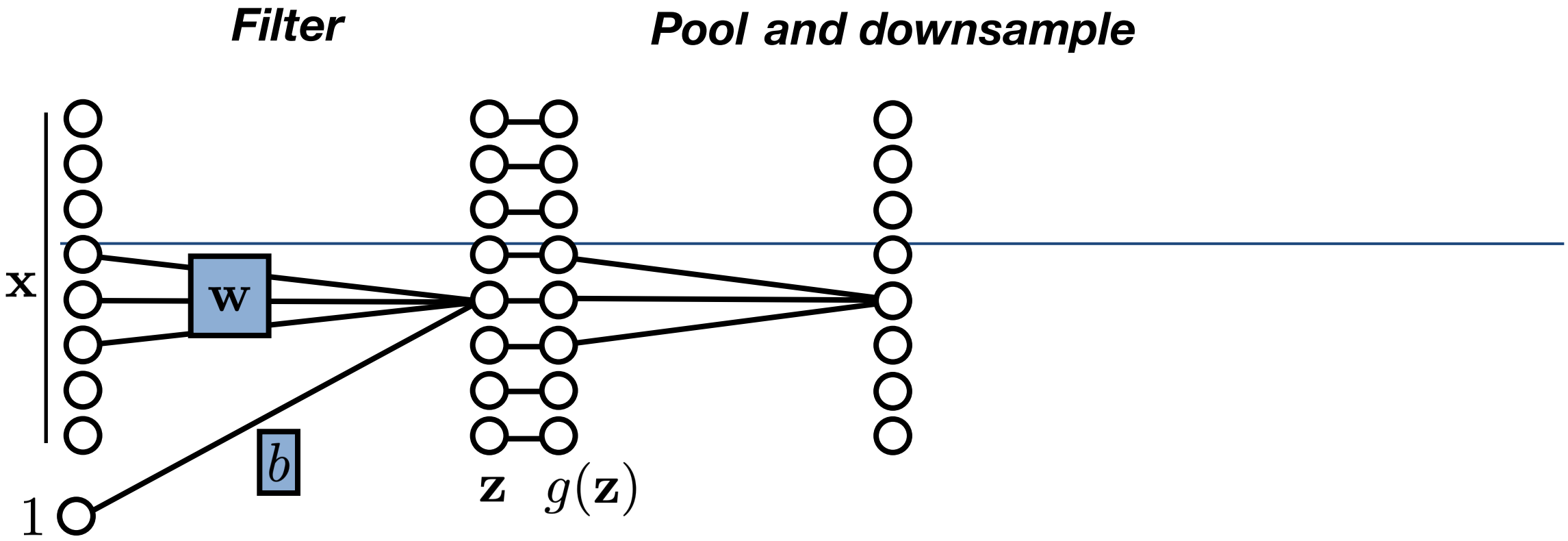
[Derived from slide by Andrea Vedaldi]

Computation in a neural net

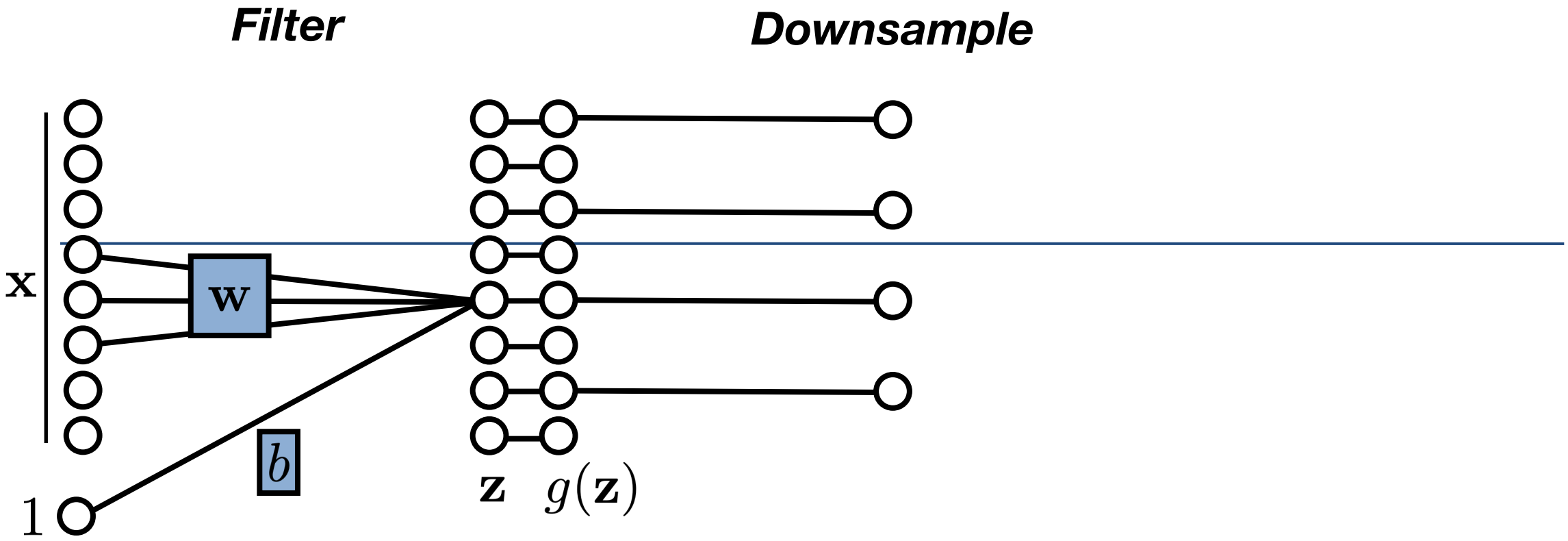


$$f(\mathbf{x}) = f_L(\ldots f_2(f_1(\mathbf{x})))$$

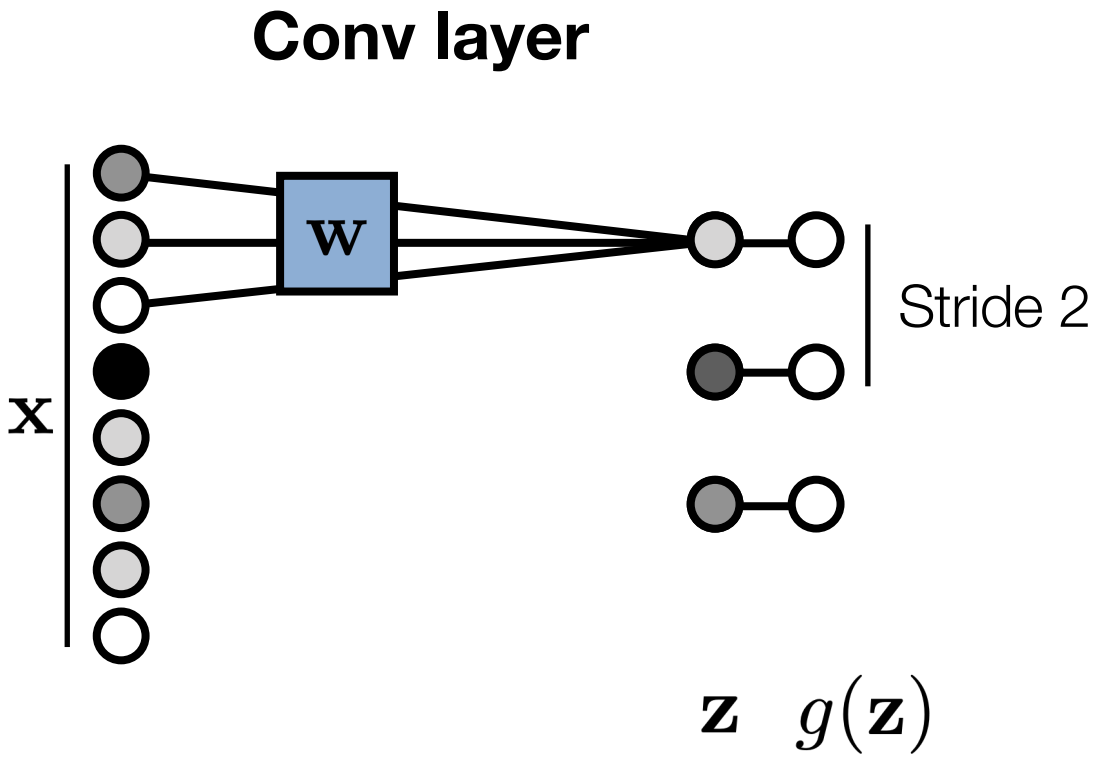
Downsampling



Downsampling

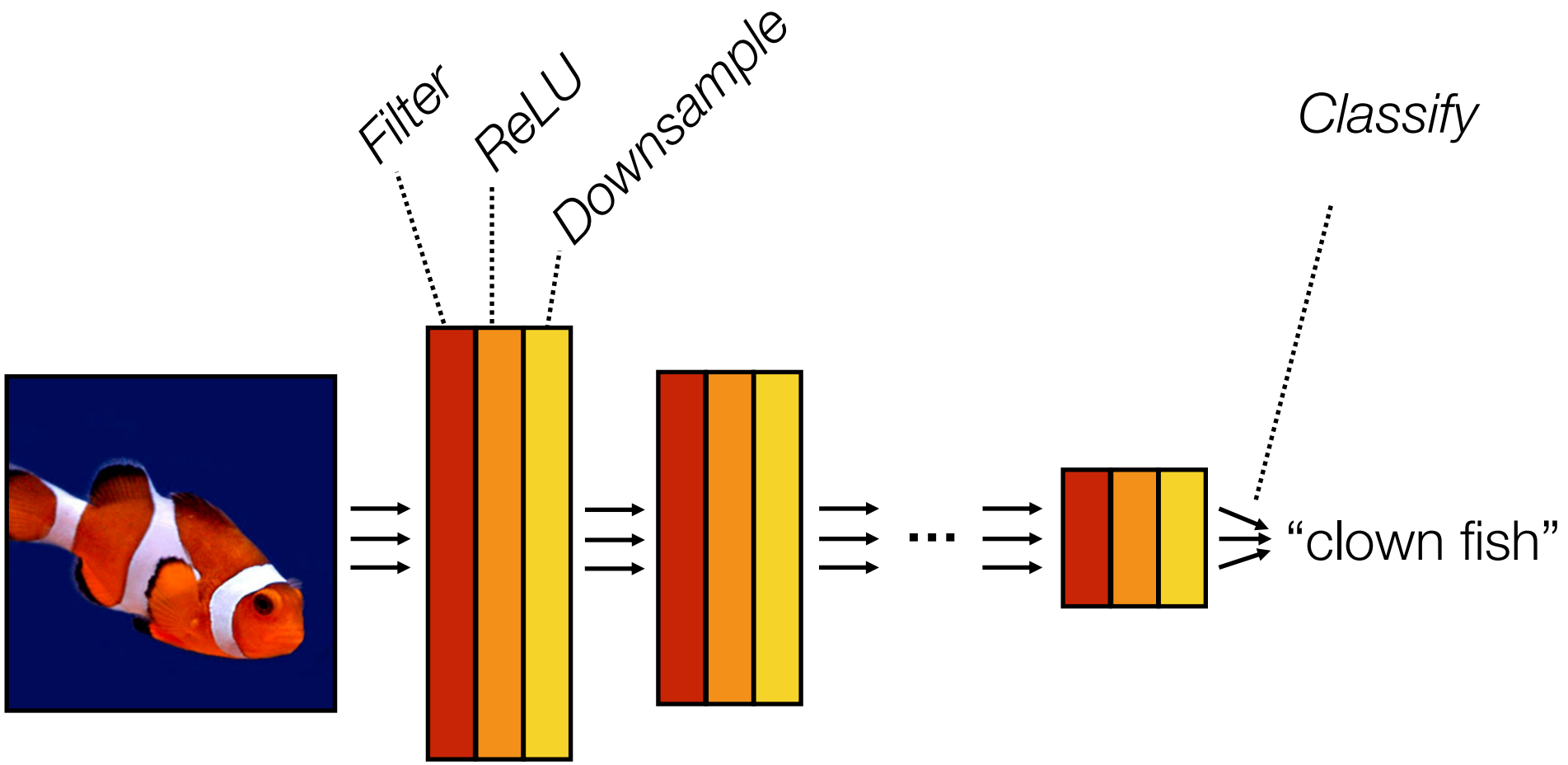


Strided operations



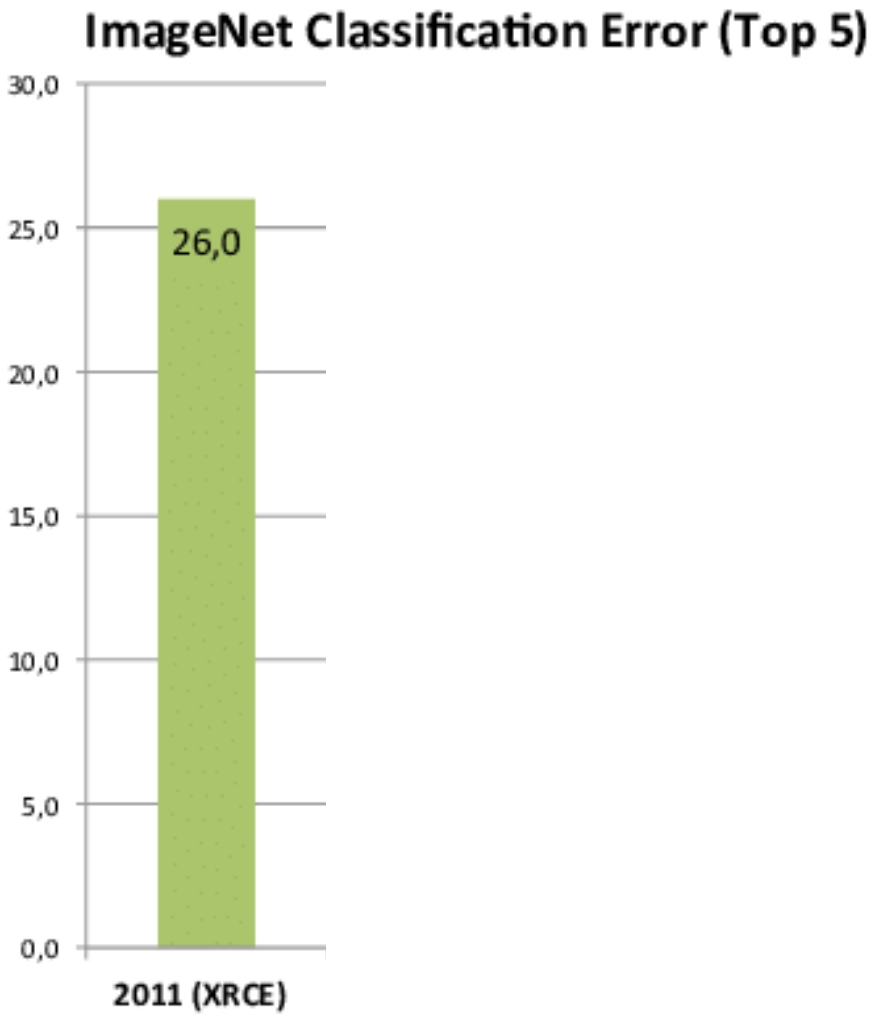
Strided operations combine a given operation (convolution or pooling) and downsampling into a single operation.

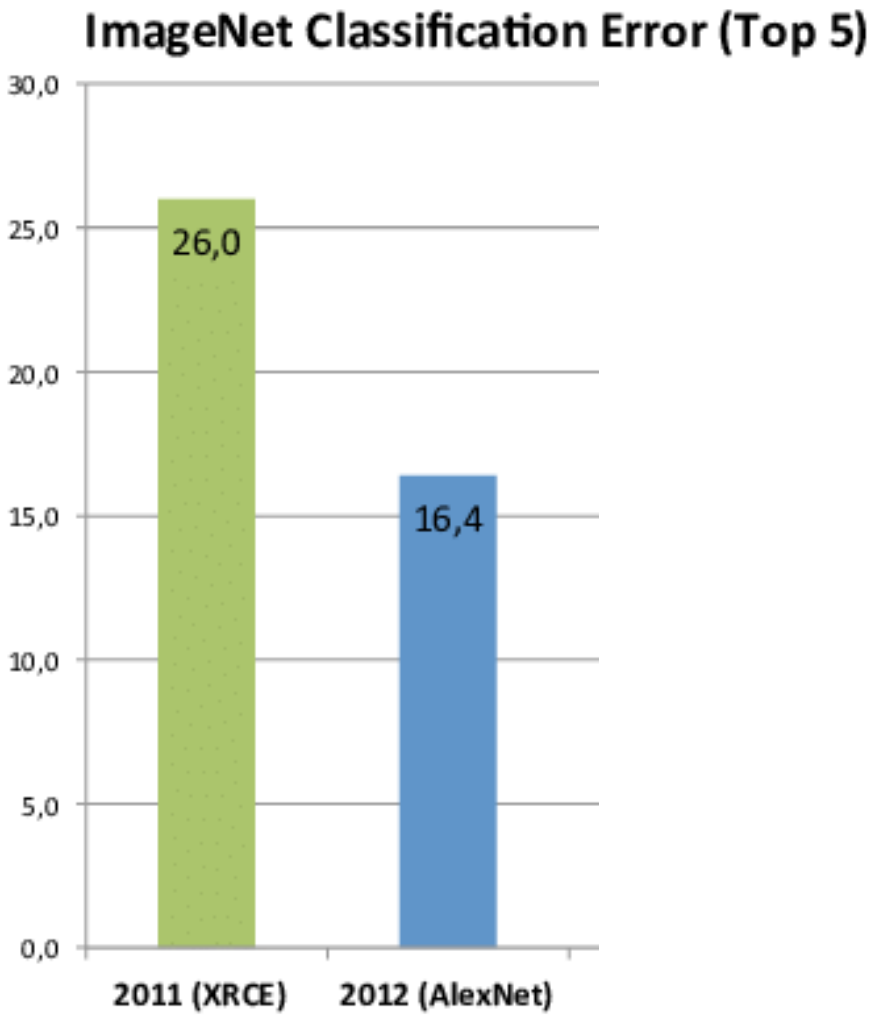
Computation in a neural net



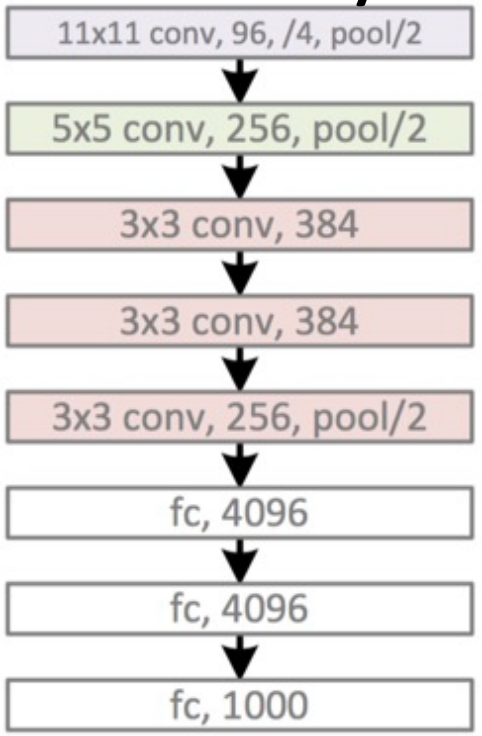
$$f(\mathbf{x}) = f_L(\ldots f_2(f_1(\mathbf{x})))$$

SOME NETWORKS


ImageNet Classification Error (Top 5)
30,0
25,0
20,0
15,0
10,0
5,0
0,0
26,0
2011 (XRCE)



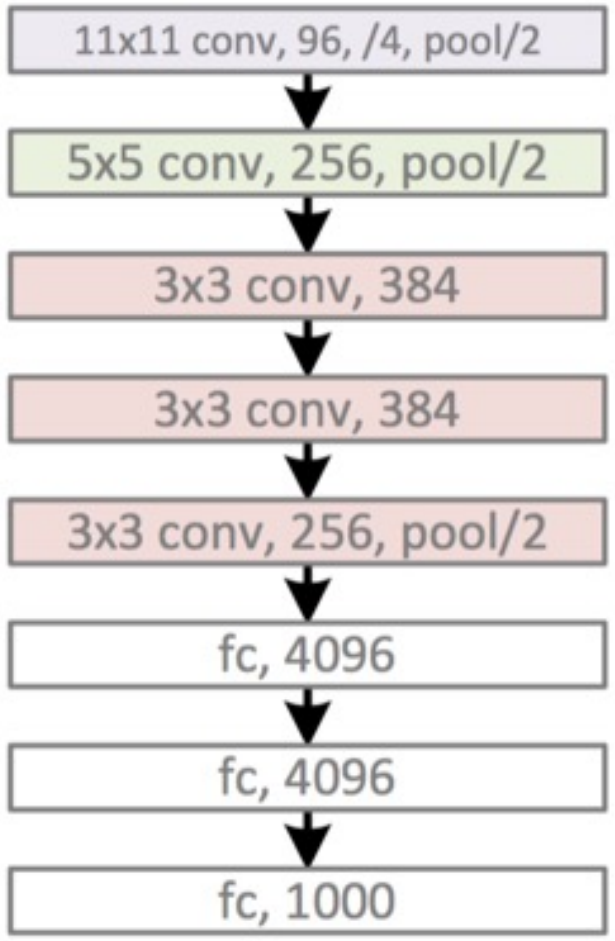
2012: AlexNet
5 conv. layers



Error: 16.4%

[Krizhevsky et al: ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012]

Alexnet — [Krizhevsky et al. NIPS 2012]



[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

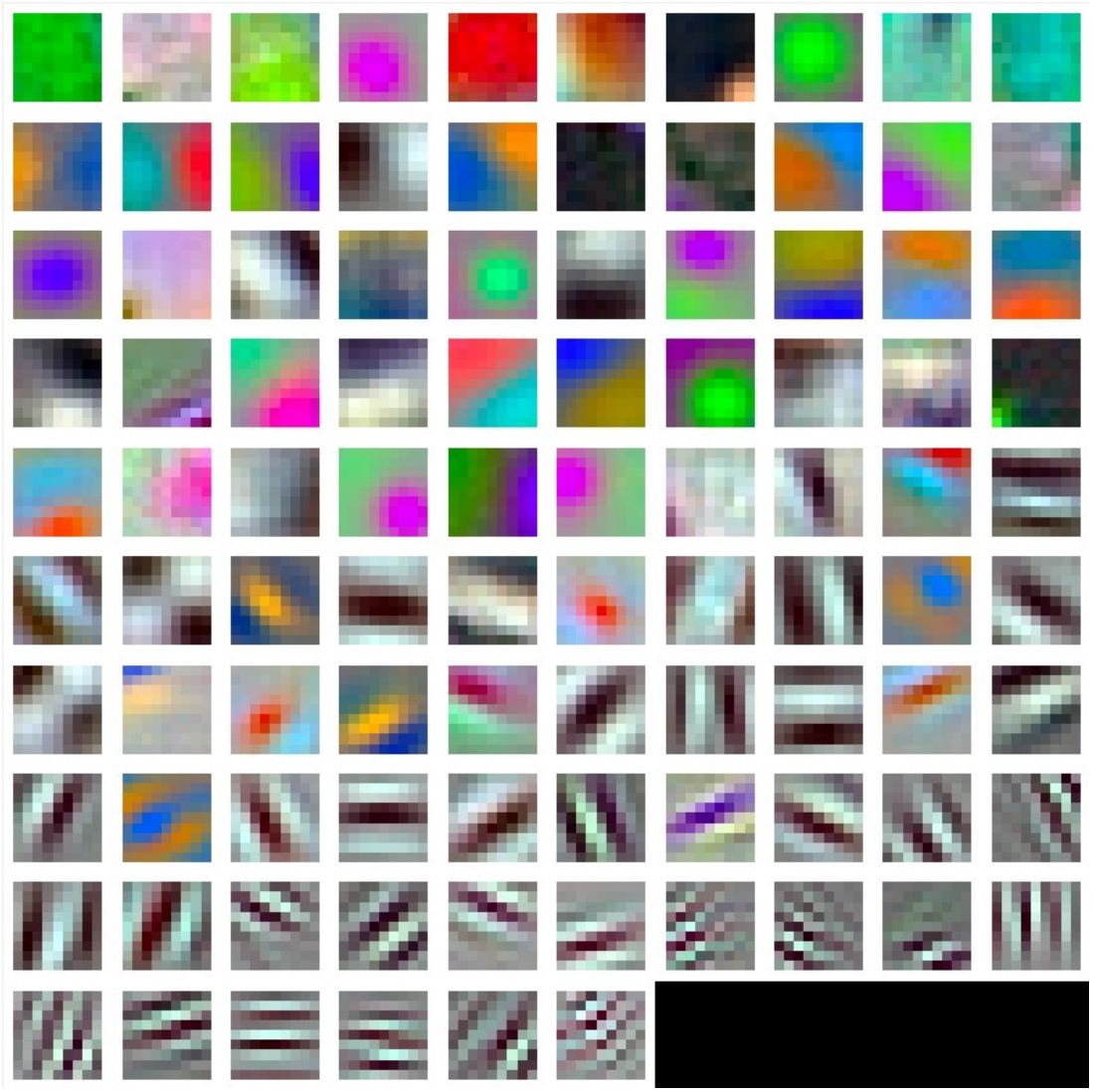
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

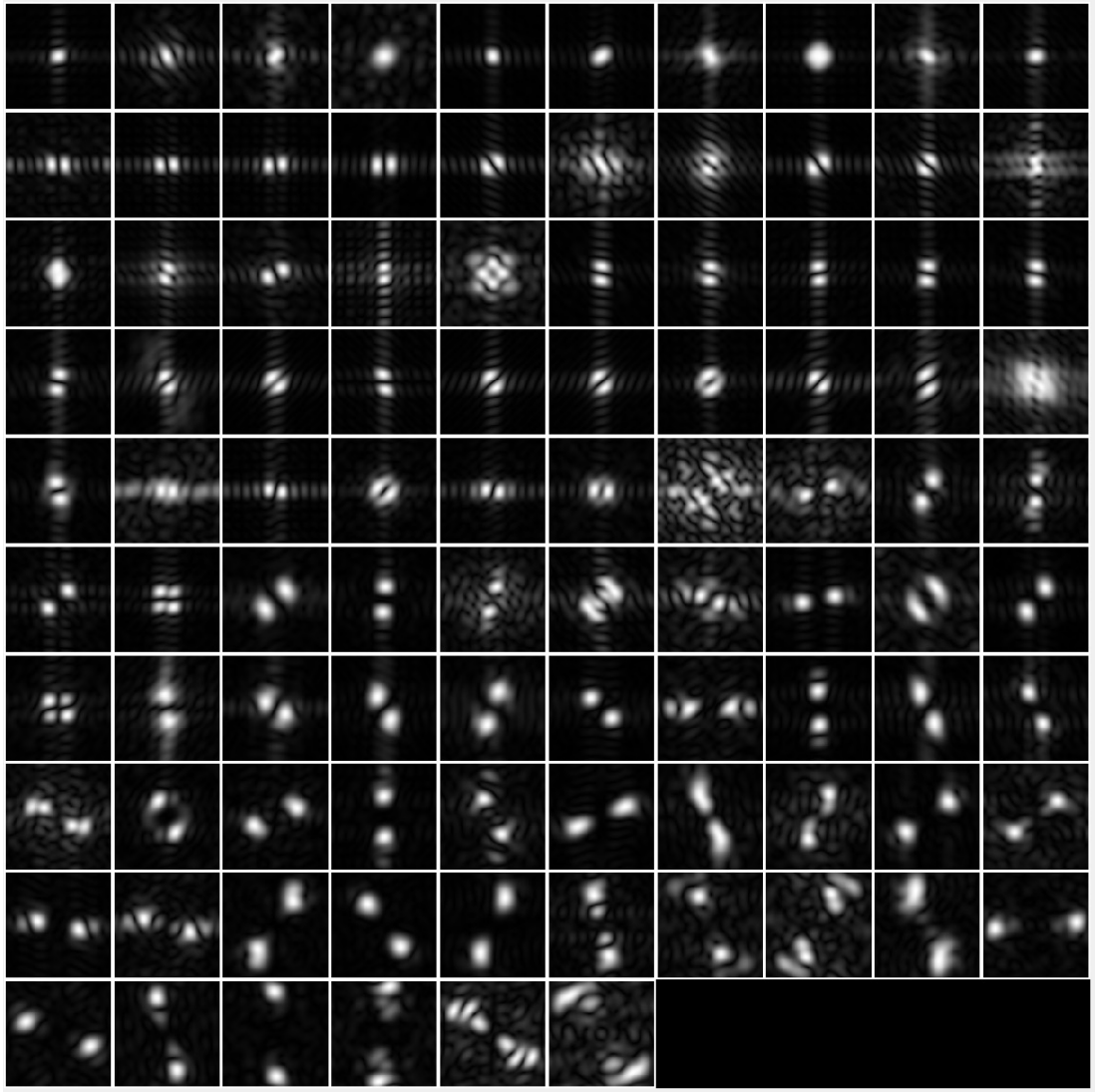
[4096] FC7: 4096 neurons

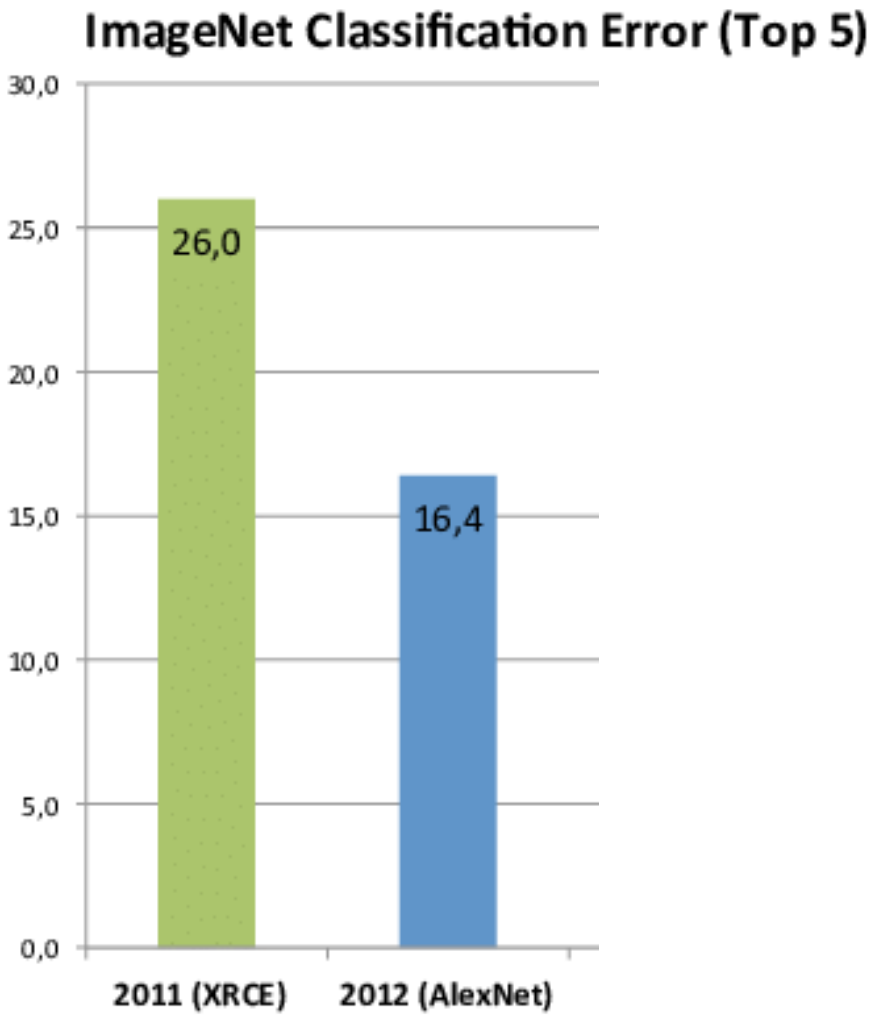
[1000] FC8: 1000 neurons (class scores)

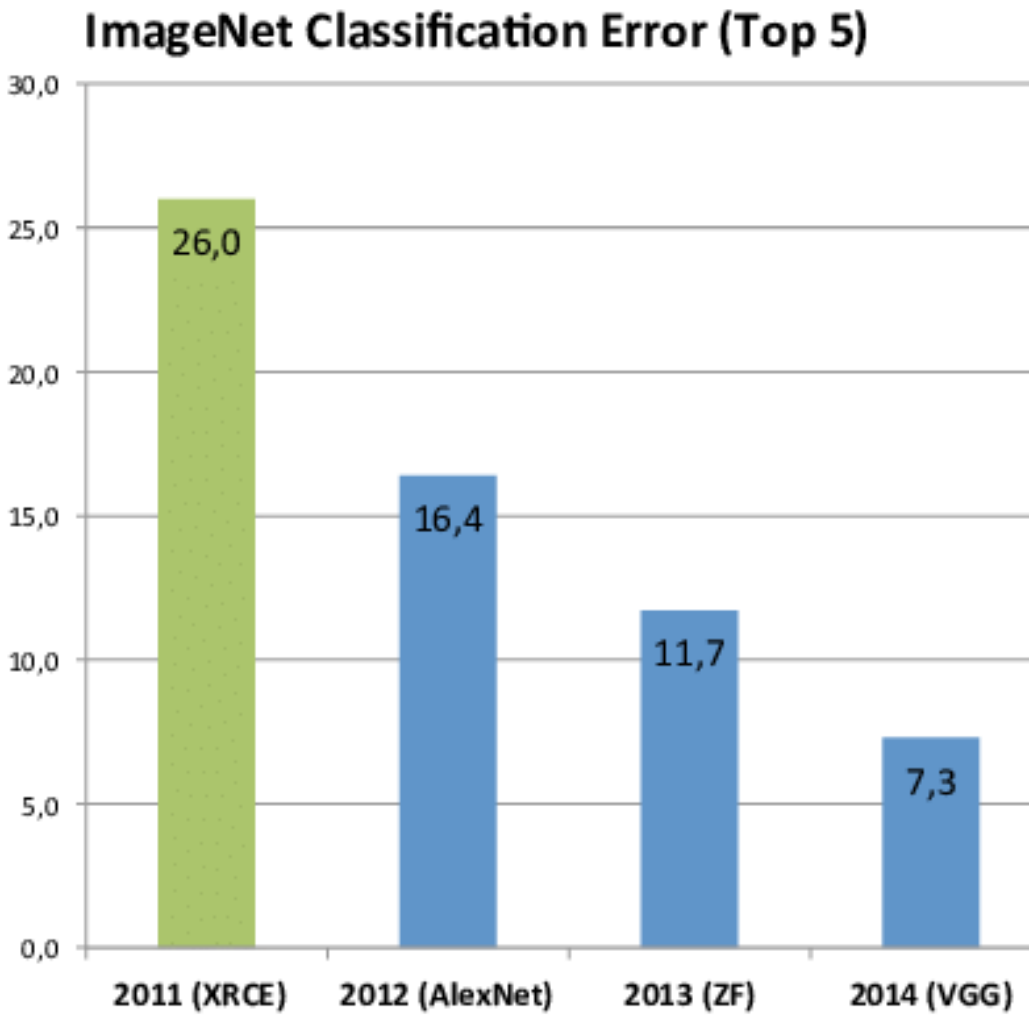
Get to know your units



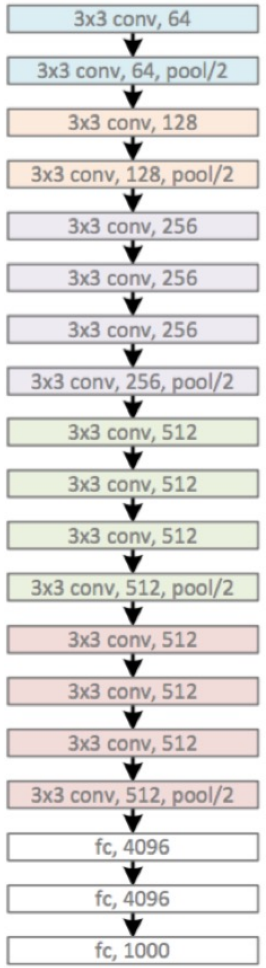
96 Units in conv1




ImageNet Classification Error (Top 5)
30,0
25,0
20,0
15,0
10,0
5,0
0,0
26,0
16,4
2011 (XRCE)
2012 (AlexNet)

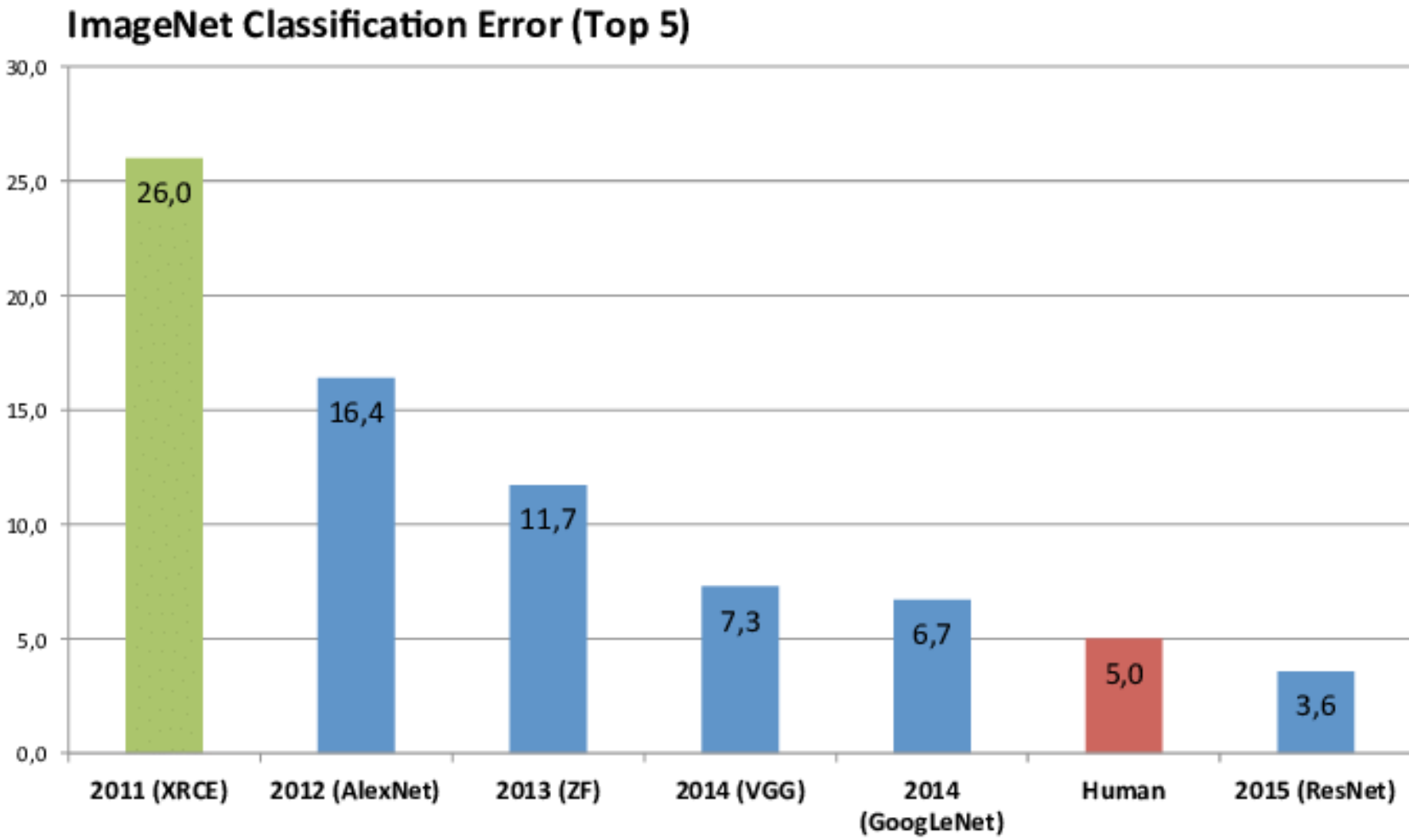


2014: VGG
16 conv. layers



Error: 7.3%

[Simonyan & Zisserman: Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015]



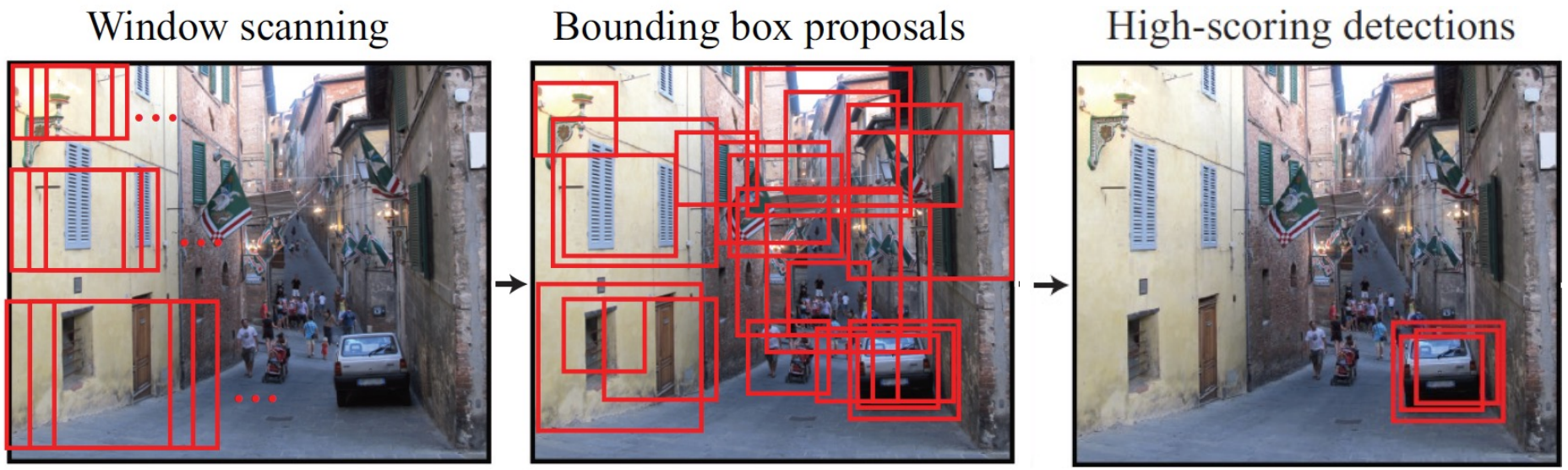
2016: ResNet
>100 conv. layers

Error: 3.6%

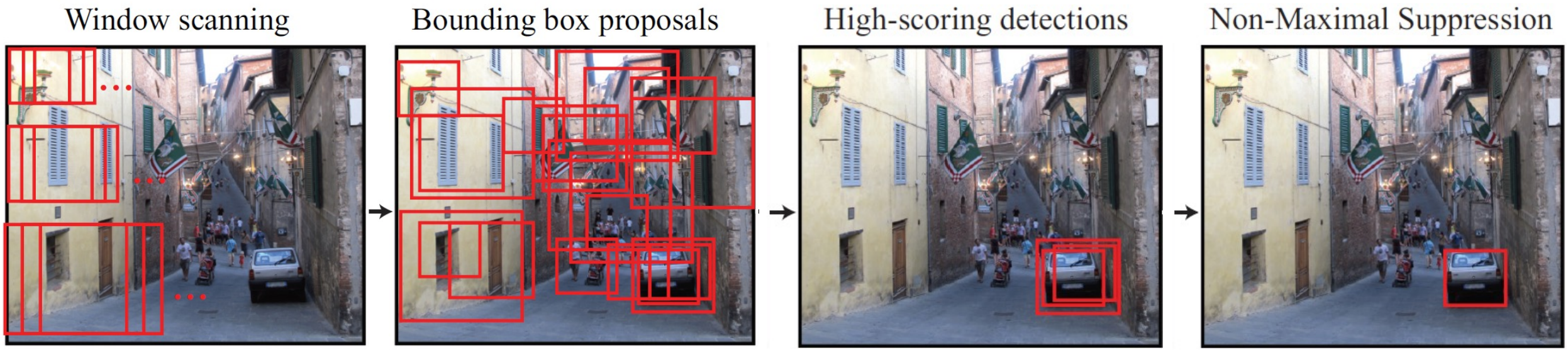
[He et al: Deep Residual Learning for Image Recognition, CVPR 2016]

OBJECT DETECTION

Localization to Classification



Non-maximal suppression



1. Take the highest confidence bounding box from the set S and add it to the final set S*

2. Remove from S the selected bounding box and all the bounding boxes with an IoU larger than a threshold.

3. go to step 1 until S is empty.

R-CNN, Fast R-CNN, Faster R-CNN

Rich feature hierarchies for accurate object detection and semantic segmentation

Tech report (v5)

Ross Girshick Jeff Donahue Trevor Darrell Jitendra Malik
UC Berkeley
{rbg,jdonahue,trevor,malik}@eecs.berkeley.edu

Abstract

Object detection performance, as measured on the canonical PASCAL VOC dataset, has plateaued in the last few years. The best-performing methods are complex ensemble systems that typically combine multiple low-level image features with high-level context. In this paper, we propose a simple and scalable detection algorithm that improves mean average precision (mAP) by more than 30% relative to the previous best result on VOC 2012—achieving a mAP of 53.3%. Our approach combines two key insights: (1) one can apply high-capacity convolutional neural networks (CNNs) to bottom-up region proposals in order to localize and segment objects and (2) when labeled training data is scarce, supervised pre-training for an auxiliary task, followed by domain-specific fine-tuning, yields a significant performance boost. Since we combine region proposals with CNNs, we call our method R-CNN: *Regions with CNN features. We also compare R-CNN to OverFeat, a recently proposed sliding-window detector based on a similar CNN architecture. We find that R-CNN outperforms OverFeat by a large margin on the 200-class ILSVRC2013 detection dataset. Source code for the complete system is available at* http://www.cs.berkeley.edu/~rbg/rcnn.

1. Introduction

Features matter. The last decade of progress on various visual recognition tasks has been based considerably on the use of SIFT [29] and HOG [7]. But if we look at performance on the canonical visual recognition task, PASCAL VOC object detection [15], it is generally acknowledged that progress has been slow during 2010-2012, with small gains obtained by building ensemble systems and employing minor variants of successful methods.

SIFT and HOG are blockwise orientation histograms, a representation we could associate roughly with complex cells in V1, the first cortical area in the primate visual pathway. But we also know that recognition occurs several stages downstream, which suggests that there might be hierarchical, multi-stage processes for computing features that are even more informative for visual recognition.

Fukushima's "neocognitron" [19], a biologically-inspired hierarchical and shift-invariant model for pattern recognition, was an early attempt at just such a process. The neocognitron, however, lacked a supervised training algorithm. Building on Rumelhart et al. [33], LeCun et al. [26] showed that stochastic gradient descent via back-propagation was effective for training convolutional neural networks (CNNs), a class of models that extend the neocognitron.

CNNs saw heavy use in the 1990s (e.g., [27]), but then fell out of fashion with the rise of support vector machines. In 2012, Krizhevsky et al. [25] rekindled interest in CNNs by showing substantially higher image classification accuracy on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [9, 10]. Their success resulted from training a large CNN on 1.2 million labeled images, together with a few twists on LeCun's CNN (e.g., $\max(x, 0)$ rectifying non-linearities and "dropout" regularization).

The significance of the ImageNet result was vigorously

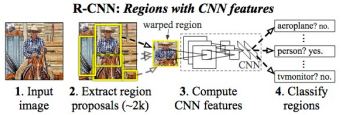


Figure 1: Object detection system overview. Our system (1) takes an input image, (2) extracts around 2000 bottom-up region proposals, (3) computes features for each proposal using a large convolutional neural network (CNN), and then (4) classifies each region using class-specific linear SVMs. R-CNN achieves a mean average precision (mAP) of **53.7% on PASCAL VOC 2010**. For comparison, [39] reports 35.1% mAP using the same region proposals, but with a spatial pyramid and bag-of-visual-words approach. The popular deformable part models perform at 33.4%. On the 200-class **ILSVRC2013 detection dataset, R-CNN's mAP is 31.4%**, a large improvement over OverFeat [34], which had the previous best result at 24.3%.

https://arxiv.org/pdf/1311.2524.pdf

Fast R-CNN

Ross Girshick
Microsoft Research
rbg@microsoft.com

Abstract

This paper proposes a Fast Region-based Convolutional Network method (Fast R-CNN) *for object detection. Fast R-CNN builds on previous work to efficiently classify object proposals using deep convolutional networks. Compared to previous work, Fast R-CNN employs several innovations to improve training and testing speed while also increasing detection accuracy. Fast R-CNN trains the very deep VGG16 network 9× faster than R-CNN, is 213× faster at test-time, and achieves a higher mAP on PASCAL VOC 2012. Compared to SPPnet, Fast R-CNN trains VGG16 3× faster, tests 10× faster, and is more accurate. Fast R-CNN is implemented in Python and C++ (using Caffe) and is available under the open-source MIT License at* https://github.com/rbgirshick/fast-rcnn.

1. Introduction

Recently, deep ConvNets [14, 16] have significantly improved image classification [14] and object detection [9, 19] accuracy. Compared to image classification, object detection is a more challenging task that requires more complex methods to solve. Due to this complexity, current approaches (*e.g.*, [9, 11, 19, 25]) train models in multi-stage pipelines that are slow and inelegant.

Complexity arises because detection requires the accurate localization of objects, creating two primary challenges. First, numerous candidate object locations (often called "proposals") must be processed. Second, these candidates provide only rough localization that must be refined to achieve precise localization. Solutions to these problems often compromise speed, accuracy, or simplicity.

In this paper, we streamline the training process for state-of-the-art ConvNet-based object detectors [9, 11]. We propose a single-stage training algorithm that jointly learns to classify object proposals and refine their spatial locations.

The resulting method can train a very deep detection network (VGG16 [20]) 9× faster than R-CNN [9] and 3× faster than SPPnet [11]. At runtime, the detection network processes images in 0.3s (excluding object proposal time) while achieving top accuracy on PASCAL VOC 2012 [7] with a mAP of 66% (vs. 62% for R-CNN).[1]

1.1. R-CNN and SPPnet

The Region-based Convolutional Network method (R-CNN) [9] achieves excellent object detection accuracy by using a deep ConvNet to classify object proposals. R-CNN, however, has notable drawbacks:

1. **Training is a multi-stage pipeline.** R-CNN first fine-tunes a ConvNet on object proposals using log loss. Then, it fits SVMs to ConvNet features. These SVMs act as object detectors, replacing the softmax classifier learnt by fine-tuning. In the third training stage, bounding-box regressors are learned.
2. **Training is expensive in space and time.** For SVM and bounding-box regressor training, features are extracted from each object proposal in each image and written to disk. With very deep networks, such as VGG16, this process takes 2.5 GPU-days for the 5k images of the VOC07 trainval set. These features require hundreds of gigabytes of storage.
3. **Object detection is slow.** At test-time, features are extracted from each object proposal in each test image. Detection with VGG16 takes 47s / image (on a GPU).

R-CNN is slow because it performs a ConvNet forward pass for each object proposal, without sharing computation. Spatial pyramid pooling networks (SPPnets) [11] were proposed to speed up R-CNN by sharing computation. The SPPnet method computes a convolutional feature map for the entire input image and then classifies each object proposal using a feature vector extracted from the shared feature map. Features are extracted for a proposal by max-pooling the portion of the feature map inside the proposal into a fixed-size output (*e.g.*, 6 × 6). Multiple output sizes are pooled and then concatenated as in spatial pyramid pooling [15]. SPPnet accelerates R-CNN by 10 to 100× at test time. Training time is also reduced by 3× due to faster proposal feature extraction.

[1] All timings use one Nvidia K40 GPU overclocked to 875 MHz.

https://arxiv.org/pdf/1504.08083.pdf

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun

Abstract—State-of-the-art object detection networks depend on region proposal algorithms to hypothesize object locations. Advances like SPPnet [1] and Fast R-CNN [2] have reduced the running time of these detection networks, exposing region proposal computation as a bottleneck. In this work, we introduce a *Region Proposal Network* (RPN) that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals. An RPN is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position. The RPN is trained end-to-end to generate high-quality region proposals, which are used by Fast R-CNN for detection. We further merge RPN and Fast R-CNN into a single network by sharing their convolutional features—using the recently popular terminology of neural networks with "attention" mechanisms, the RPN component tells the unified network where to look. For the very deep VGG-16 model [3], our detection system has a frame rate of 5fps (*including all steps*) on a GPU, while achieving state-of-the-art object detection accuracy on PASCAL VOC 2007, 2012, and MS COCO datasets with only 300 proposals per image. In ILSVRC and COCO 2015 competitions, Faster R-CNN and RPN are the foundations of the 1st-place winning entries in several tracks. Code has been made publicly available.

Index Terms—Object Detection, Region Proposal, Convolutional Neural Network.

1 Introduction

Recent advances in object detection are driven by the success of region proposal methods (*e.g.*, [4]) and region-based convolutional neural networks (R-CNNs) [5]. Although region-based CNNs were computationally expensive as originally developed in [5], their cost has been drastically reduced thanks to sharing convolutions across proposals [1], [2]. The latest incarnation, Fast R-CNN [2], achieves near real-time rates using very deep networks [3], *when ignoring the time spent on region proposals*. Now, proposals are the test-time computational bottleneck in state-of-the-art detection systems.

Region proposal methods typically rely on inexpensive features and economical inference schemes. Selective Search [4], one of the most popular methods, greedily merges superpixels based on engineered low-level features. Yet when compared to efficient detection networks [2], Selective Search is an order of magnitude slower, at 2 seconds per image in a CPU implementation. EdgeBoxes [6] currently provides the best tradeoff between proposal quality and speed, at 0.2 seconds per image. Nevertheless, the region proposal step still consumes as much running time as the detection network.

- *S. Ren is with University of Science and Technology of China, Hefei, China. This work was done when S. Ren was an intern at Microsoft Research. Email: sqren@mail.ustc.edu.cn*
- *K. He and J. Sun are with Visual Computing Group, Microsoft Research. E-mail: {kahe,jiansun}@microsoft.com*
- *R. Girshick is with Facebook AI Research. The majority of this work was done when R. Girshick was with Microsoft Research. E-mail: rbg@fb.com*

One may note that fast region-based CNNs take advantage of GPUs, while the region proposal methods used in research are implemented on the CPU, making such runtime comparisons inequitable. An obvious way to accelerate proposal computation is to re-implement it for the GPU. This may be an effective engineering solution, but re-implementation ignores the down-stream detection network and therefore misses important opportunities for sharing computation.

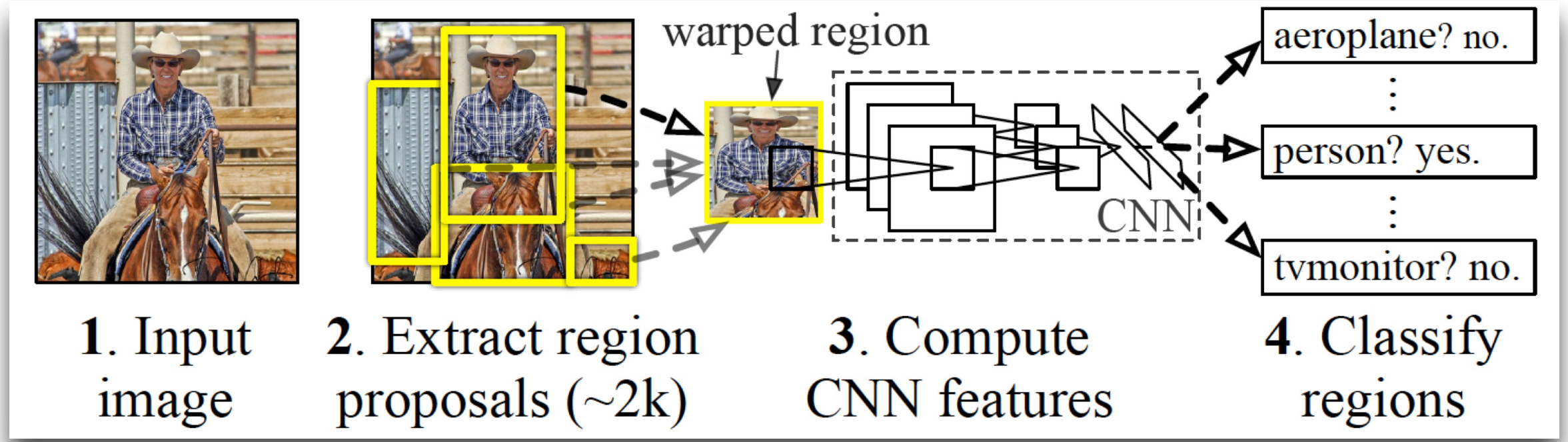
In this paper, we show that an algorithmic change—computing proposals with a deep convolutional neural network—leads to an elegant and effective solution where proposal computation is nearly cost-free given the detection network's computation. To this end, we introduce novel *Region Proposal Networks* (RPNs) that share convolutional layers with state-of-the-art object detection networks [1], [2]. By sharing convolutions at test-time, the marginal cost for computing proposals is small (*e.g.*, 10ms per image).

Our observation is that the convolutional feature maps used by region-based detectors, like Fast R-CNN, can also be used for generating region proposals. On top of these convolutional features, we construct an RPN by adding a few additional convolutional layers that simultaneously regress region bounds and objectness scores at each location on a regular grid. The RPN is thus a kind of fully convolutional network (FCN) [7] and can be trained end-to-end specifically for the task for generating detection proposals.

RPNs are designed to efficiently predict region proposals with a wide range of scales and aspect ratios. In contrast to prevalent methods [8], [9], [1], [2] that use

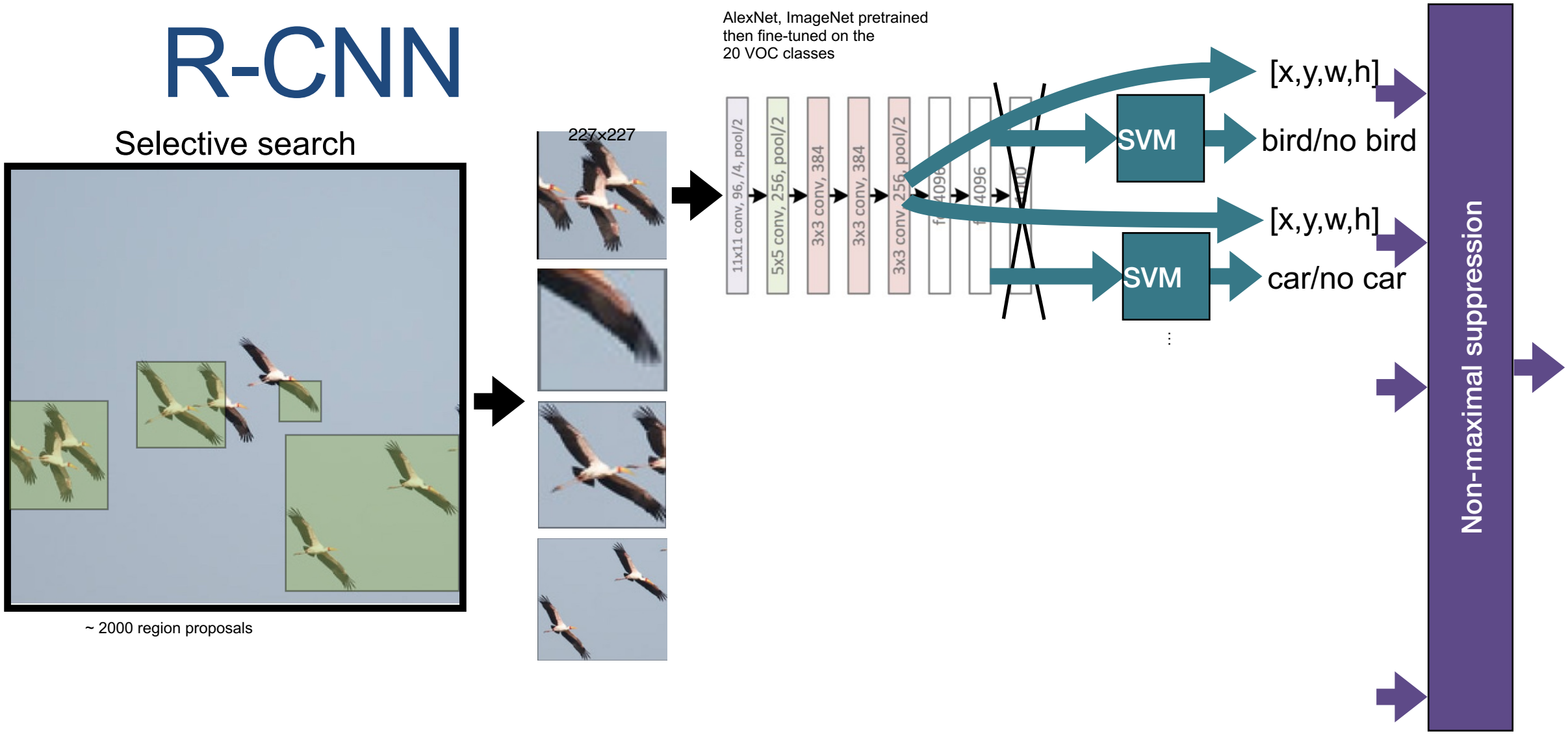
https://arxiv.org/pdf/1506.01497.pdf

R-CNN



Girshick, R., J. Donahue, T. Darrell, and J. Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation." *CVPR '14 Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*. Pages 580-587. 2014

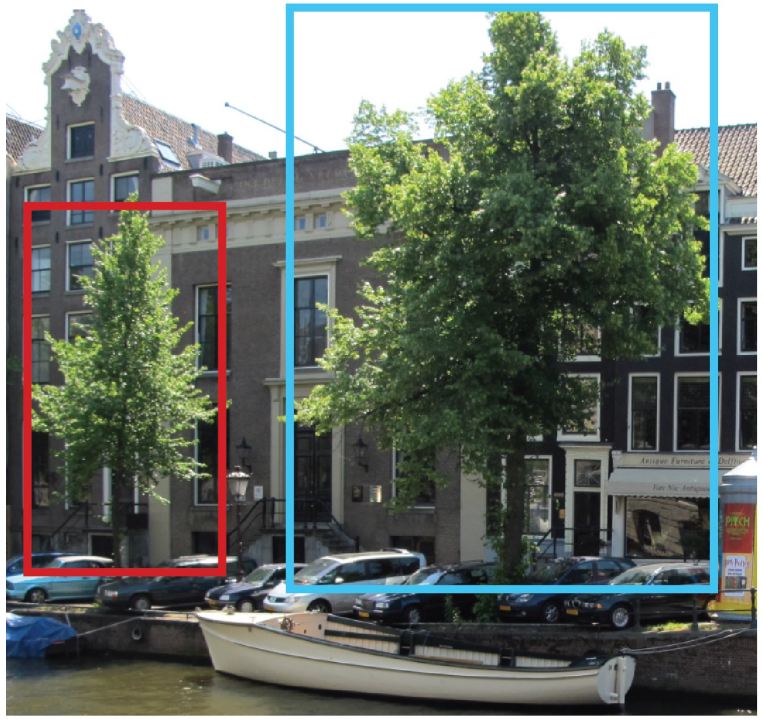
R-CNN

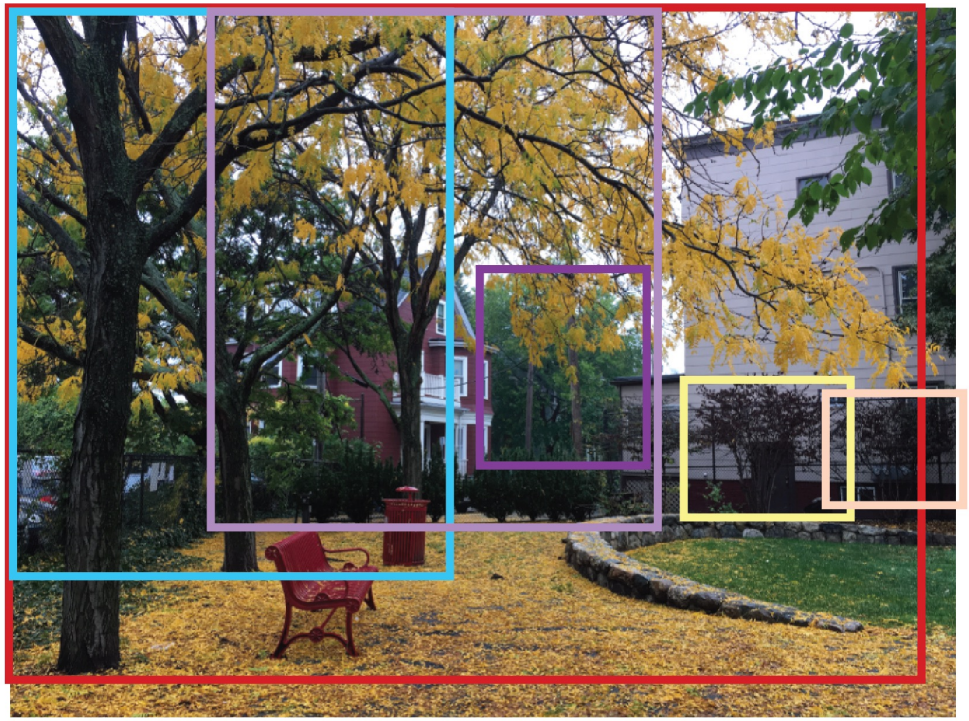


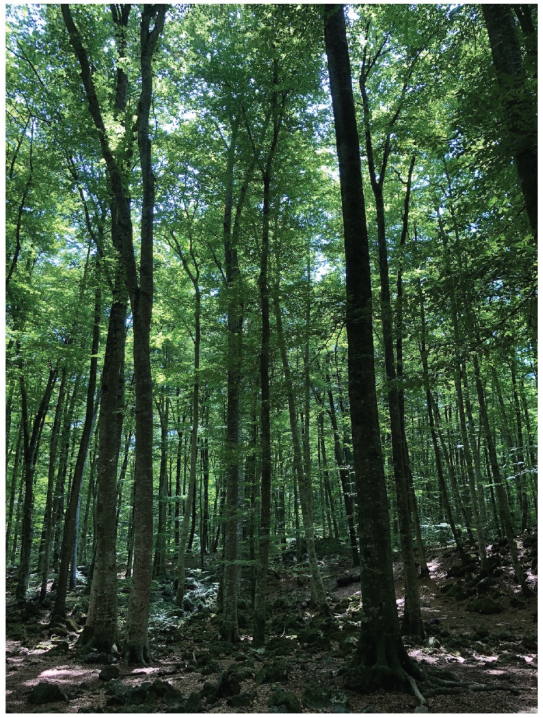
Girshick, R., J. Donahue, T. Darrell, and J. Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation."
CVPR '14 Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition. Pages 580-587. 2014

Bounding box localization: shortcomings

It inherits all the problems of classification plus:





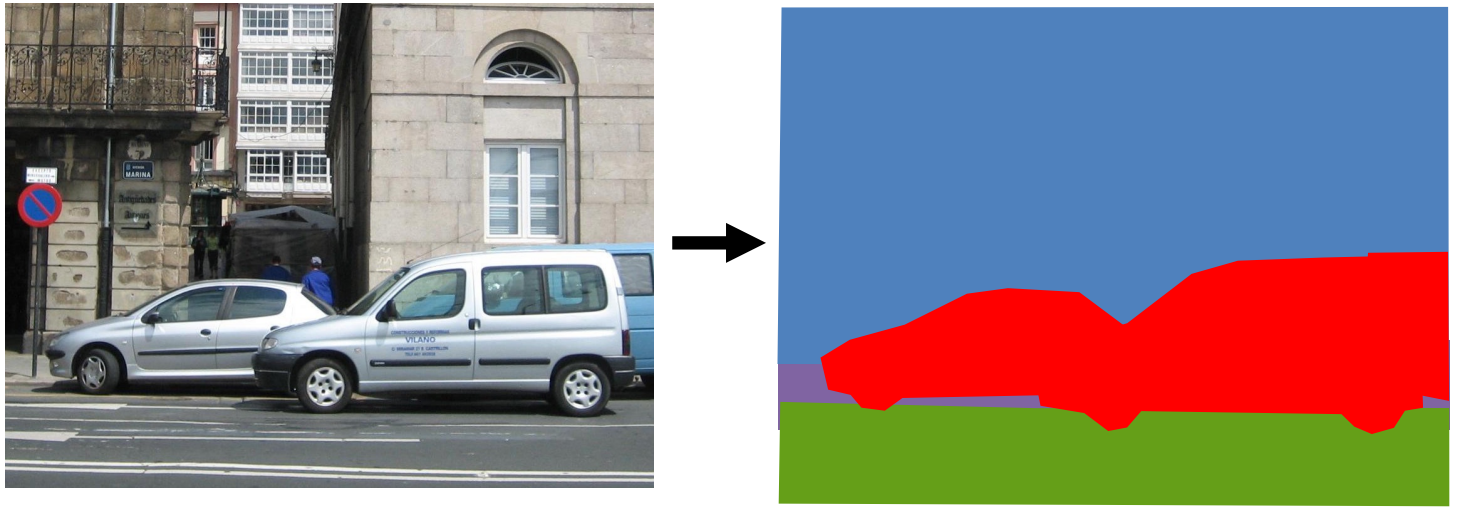


OBJECT SEGMENTATION

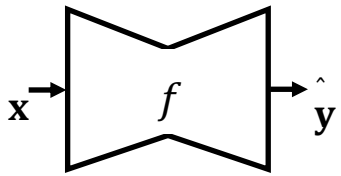
Object segmentation

We can try to classify each pixel in an image with an object class. Per-pixel classification of object labels is referred to as semantic segmentation.

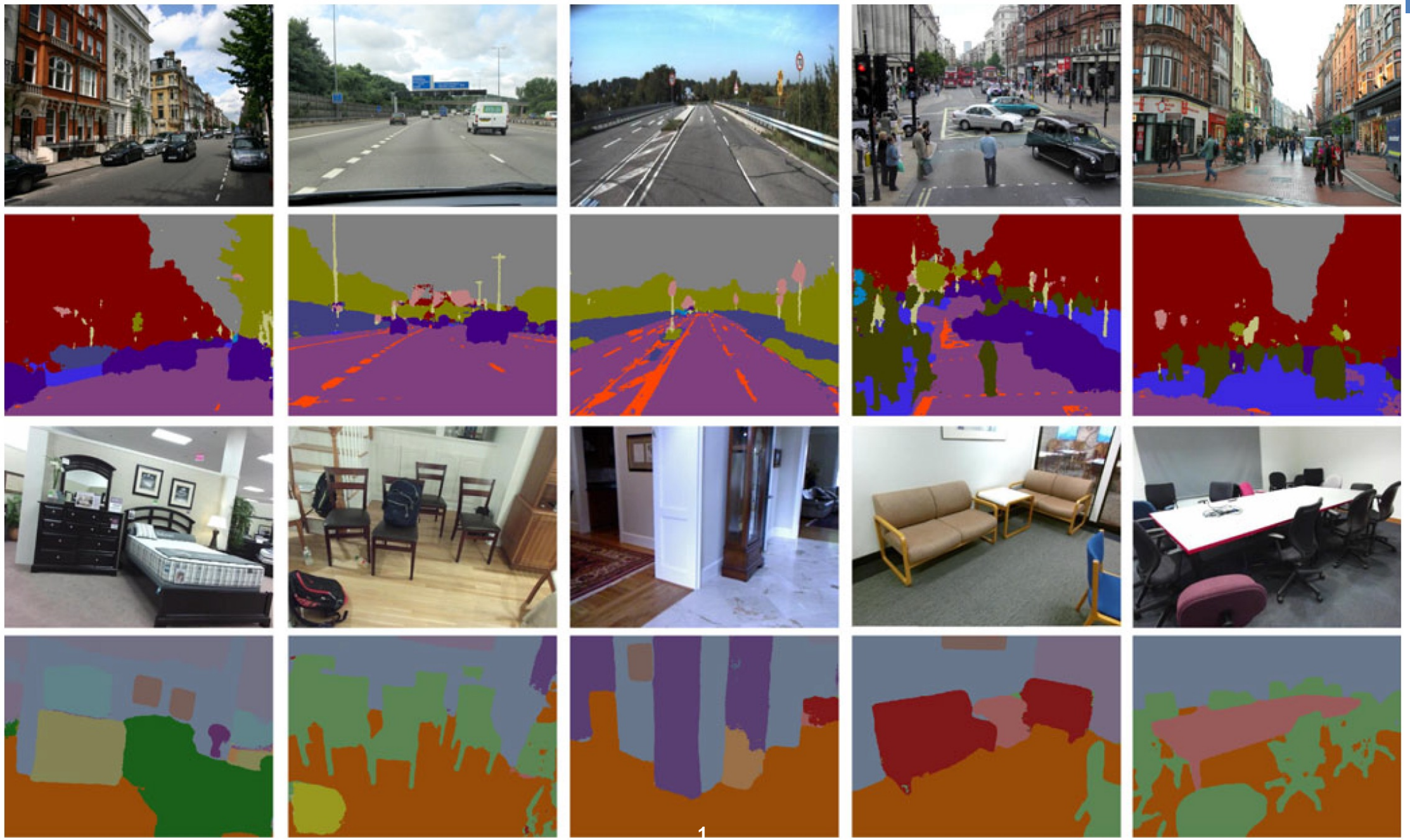
Is object class c present in the pixel [n,m]?



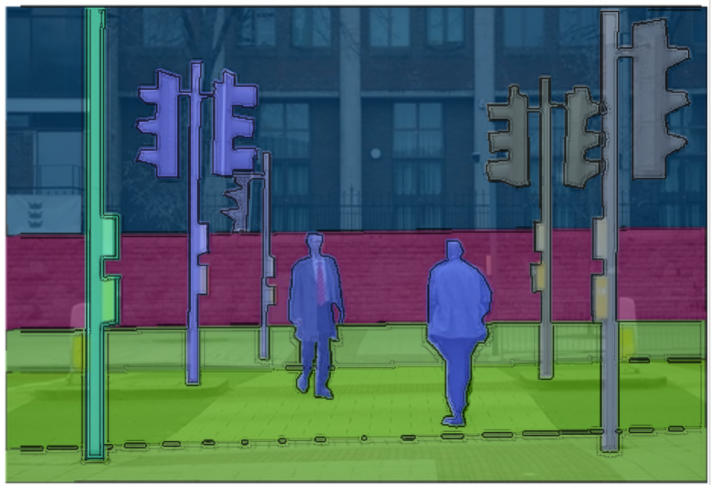
Input is an array (image)



Output is an array (segmentation)



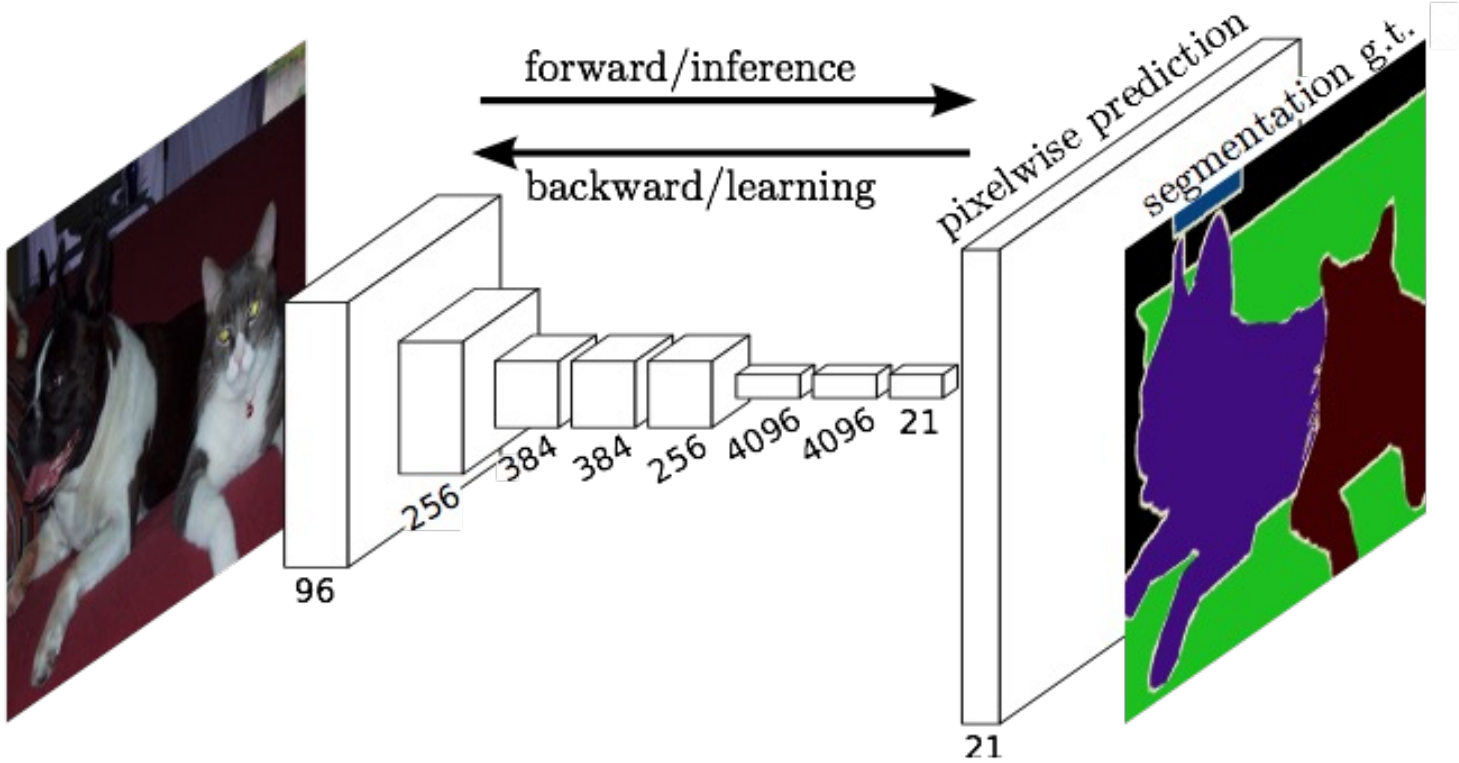
Datasets: COCO ADE20K



	Images	Obj. inst.	Obj. classes	Part inst.	Part classes	Obj. classes per image
COCO	123,287	886,284	91	0	0	3.5
ImageNet*	476,688	534,309	200	0	0	1.7
NYU Depth V2	1,449	34,064	894	0	0	14.1
Cityscapes	25,000	N/A	30	0	0	N/A
SUN	16,873	313,884	4,479	0	0	9.8
OpenSurfaces	22,214	71,460	160	0	0	N/A
PascalContext	10,103	∼104,398**	540	181,770	40	5.1
ADE20K	22,000	415,099	2,944	171,148	354	10.5

* has only bounding boxes (no pixel-level segmentation). Sparse annotations.
** PascalContext dataset does not have instance segmentation. In order to estimate the number of instances, we find connected components (having at least 150pixels) for each class label.

Fully Convolutional Networks



Fully Convolutional Networks for Semantic Segmentation

Jonathan Long* Evan Shelhamer* Trevor Darrell
UC Berkeley
{jonlong,shelhamer,trevor}@cs.berkeley.edu

Abstract

Convolutional networks are powerful visual models that yield hierarchies of features. We show that convolutional networks by themselves, trained end-to-end, pixels-to-pixels, exceed the state-of-the-art in semantic segmentation. Our key insight is to build "fully convolutional" networks that take input of arbitrary size and produce correspondingly-sized output with efficient inference and learning. We define and detail the space of fully convolutional networks, explain their application to spatially dense prediction tasks, and draw connections to prior models. We adapt contemporary classification networks (AlexNet [22], the VGG net [34], and GoogLeNet [35]) into fully convolutional networks and transfer their learned representations by fine-tuning [5] to the segmentation task. We then define a skip architecture that combines semantic information from a deep, coarse layer with appearance information from a shallow, fine layer to produce accurate and detailed segmentations. Our fully convolutional network achieves state-of-the-art segmentation of PASCAL VOC (20% relative improvement to 62.2% mean IU on 2012), NYUDv2, and SIFT Flow, while inference takes less than one fifth of a second for a typical image.

1. Introduction

Convolutional networks are driving advances in recognition. Convnets are not only improving for whole-image classification [22, 34, 35], but also making progress on local tasks with structured output. These include advances in bounding box object detection [32, 12, 19], part and keypoint prediction [42, 26], and local correspondence [26, 10].

The natural next step in the progression from coarse to fine inference is to make a prediction at every pixel. Prior approaches have used convnets for semantic segmentation [30, 3, 9, 31, 17, 15, 11], in which each pixel is labeled with the class of its enclosing object or region, but with shortcomings that this work addresses.

*Authors contributed equally

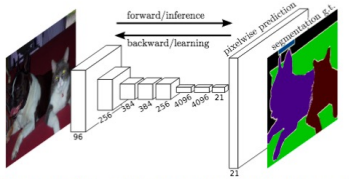


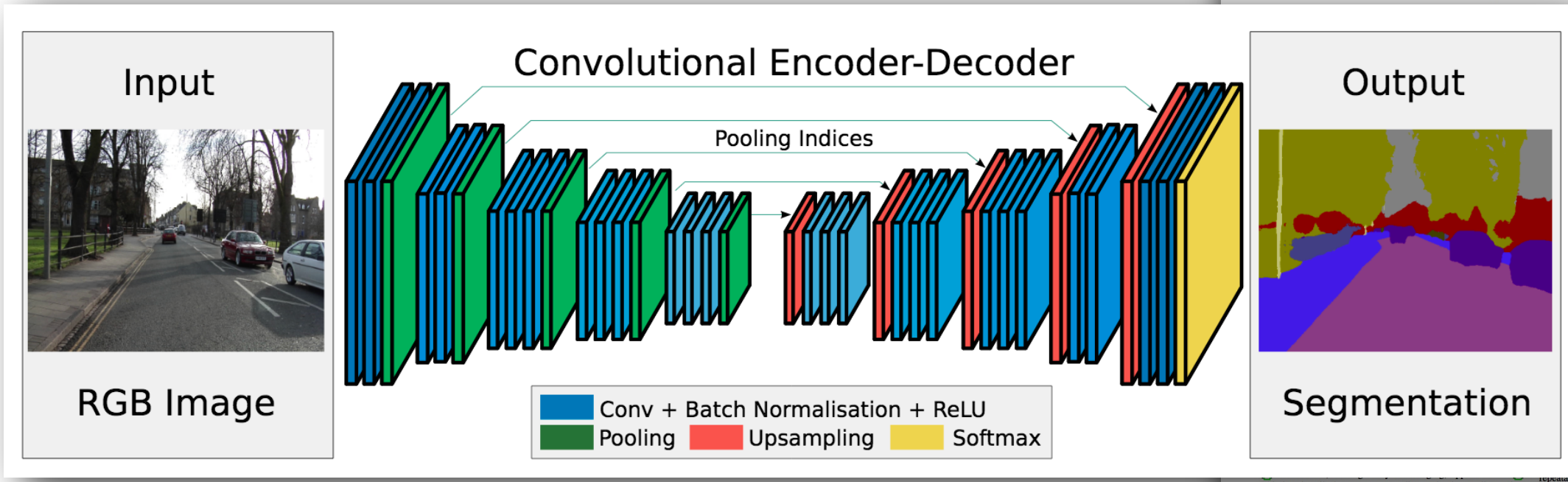
Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

We show that a fully convolutional network (FCN) trained end-to-end, pixels-to-pixels on semantic segmentation exceeds the state-of-the-art without further machinery. To our knowledge, this is the first work to train FCNs end-to-end (1) for pixelwise prediction and (2) from supervised pre-training. Fully convolutional versions of existing networks predict dense outputs from arbitrary-sized inputs. Both learning and inference are performed whole-image-at-a-time by dense feedforward computation and backpropagation. In-network upsampling layers enable pixelwise prediction and learning in nets with subsampled pooling.

This method is efficient, both asymptotically and absolutely, and precludes the need for the complications in other works. Patchwise training is common [30, 3, 9, 31, 11], but lacks the efficiency of fully convolutional training. Our approach does not make use of pre- and post-processing complications, including superpixels [9, 17], proposals [17, 15], or post-hoc refinement by random fields or local classifiers [9, 17]. Our model transfers recent success in classification [22, 34, 35] to dense prediction by reinterpreting classification nets as fully convolutional and fine-tuning from their learned representations. In contrast, previous works have applied small convnets without supervised pre-training [9, 31, 30].

Semantic segmentation faces an inherent tension between semantics and location: global information resolves what while local information resolves where. Deep feature hierarchies encode location and semantics in a nonlinear

Encoder-decoder architectures



SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation

Cipolla, *Senior Member, IEEE,*

work architecture for semantic pixel-wise segmentation
r network, a corresponding decoder network followed
pologically identical to the 13 convolutional layers in the
n encoder feature maps to full input resolution feature
which the decoder upsamples its lower resolution input
max-pooling step of the corresponding encoder to
le. The upsampled maps are sparse and are then
r proposed architecture with the widely adopted FCN [2]
es. This comparison reveals the memory versus

is designed to be efficient both in terms of memory and
r of trainable parameters than other competing
We also performed a controlled benchmark of SegNet
gmentation tasks. These quantitative assessments
nd most efficient inference memory-wise as compared
web demo at http://mi.eng.cam.ac.uk/projects/segnet/.

mentation, Indoor Scenes, Road Scenes, Encoder,

ns) and understand the spatial-relationship (context) be-
fferent classes such as road and side-walk. In typical road
he majority of the pixels belong to large classes such
building and hence the network must produce smooth
tions. The engine must also have the ability to delineate
ased on their shape despite their small size. Hence it is
to retain boundary information in the extracted image
ation. From a computational perspective, it is necessary
network to be efficient in terms of both memory and
ion time during inference. The ability to train end-to-end
to jointly optimise all the weights in the network using
nt weight update technique such as stochastic gradient
SGD) [17] is an additional benefit since it is more easily
repeatable. The design of SegNet arose from a need to match these criteria.

This is primarily because max pooling and sub-sampling reduce feature map resolution. Our motivation to design SegNet arises from this need to map low resolution features to input resolution for pixel-wise classification. This mapping must produce features which are useful for accurate boundary localization.

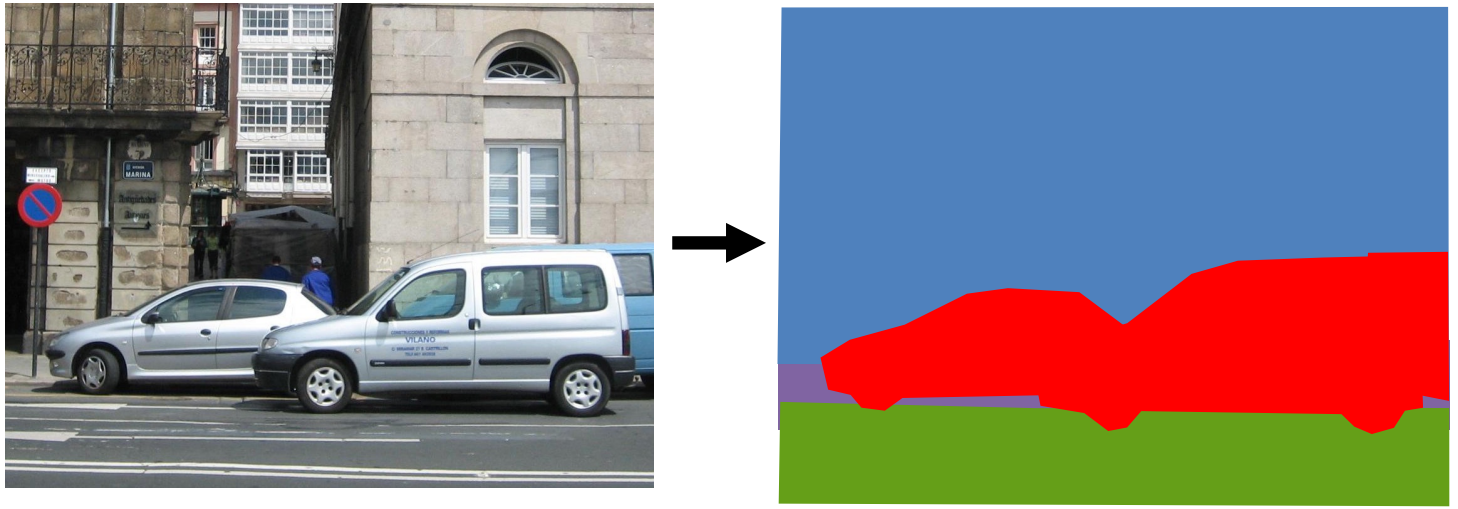
Our architecture, SegNet, is designed to be an efficient architecture for pixel-wise semantic segmentation. It is primarily motivated by road scene understanding applications which require the ability to model appearance (road, building), shape (cars,

The encoder network in SegNet is topologically identical to the convolutional layers in VGG16 [1]. We remove the fully connected layers of VGG16 which makes the SegNet encoder network significantly smaller and easier to train than many other recent architectures [2], [4], [11], [18]. The key component of SegNet is the decoder network which consists of a hierarchy of decoders one corresponding to each encoder. Of these, the appropriate decoders use the max-pooling indices received from the corresponding encoder to perform non-linear upsampling of their input feature maps. This idea was inspired from an architecture designed for unsupervised feature learning [19]. Reusing max-pooling indices in the decoding process has several practical

• *V. Badrinarayanan, A. Kendall, R. Cipolla are with the Machine Intelligence Lab, Department of Engineering, University of Cambridge, UK. E-mail: vb292,agk34,cipolla@eng.cam.ac.uk*

Object segmentation: shortcomings

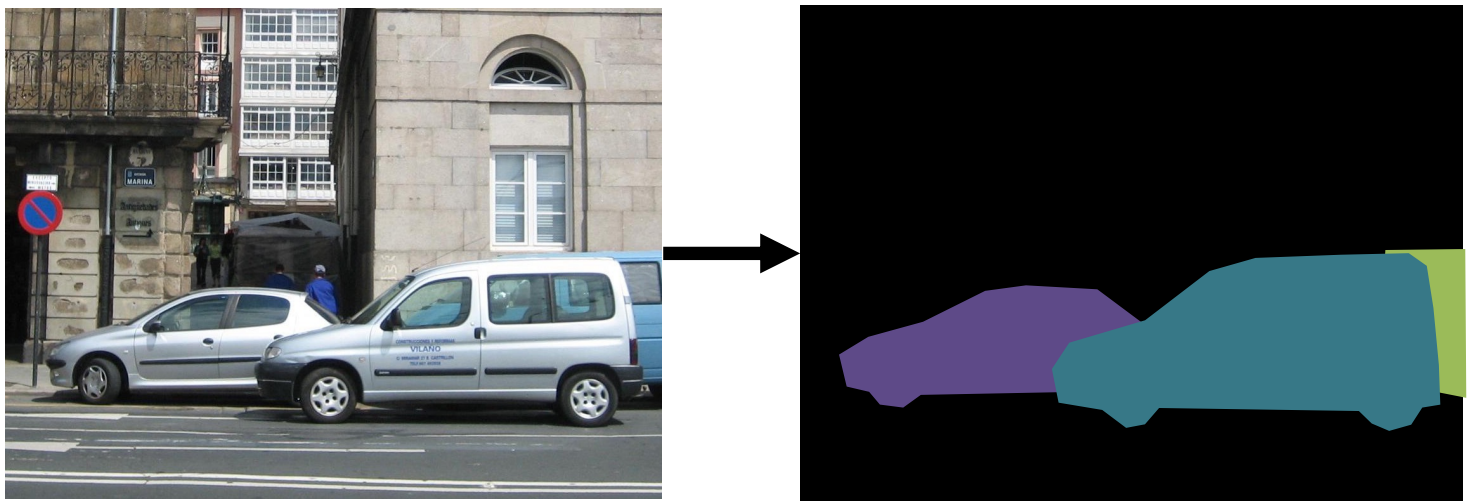
We can not count objects!

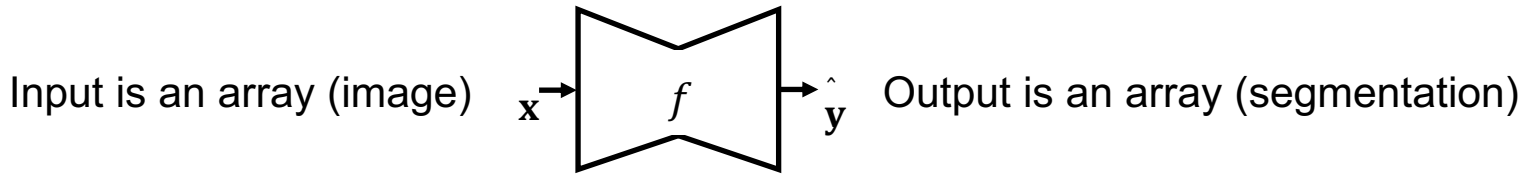


Instance segmentation

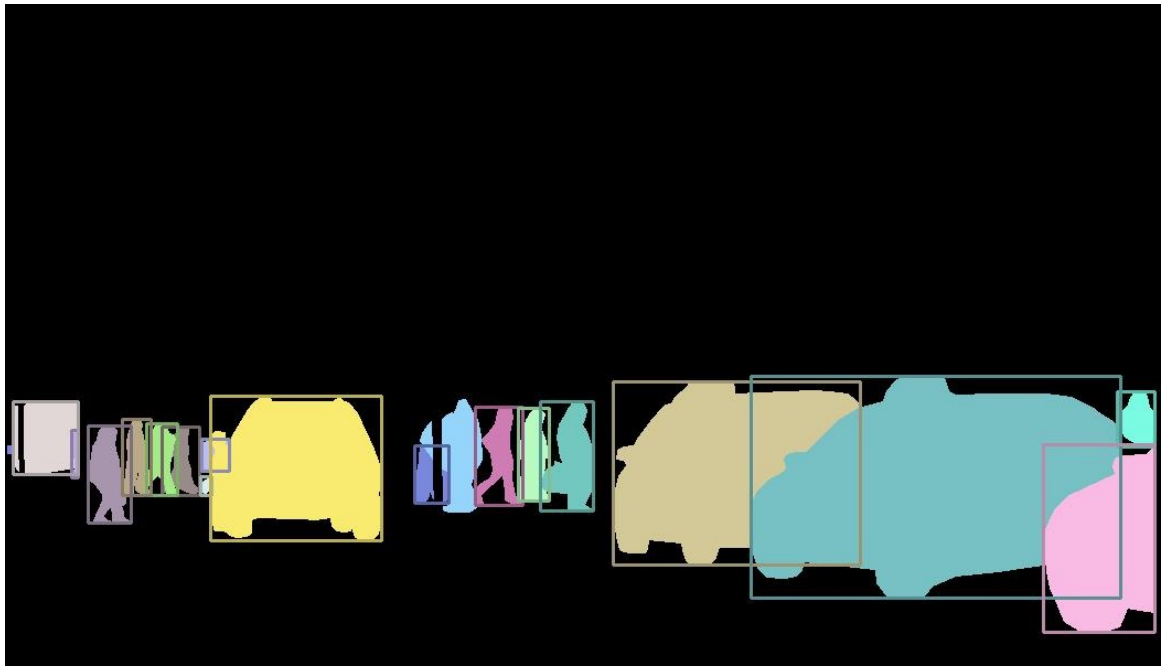
We can try to classify each pixel in an image with an object class. Per-pixel classification of object labels is referred to as semantic segmentation.

Is instance ***i*** of object class ***c*** present in the pixel [n,m]?



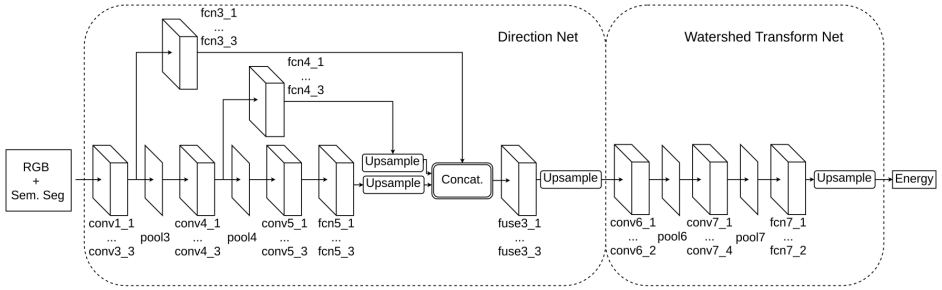


Instance segmentation

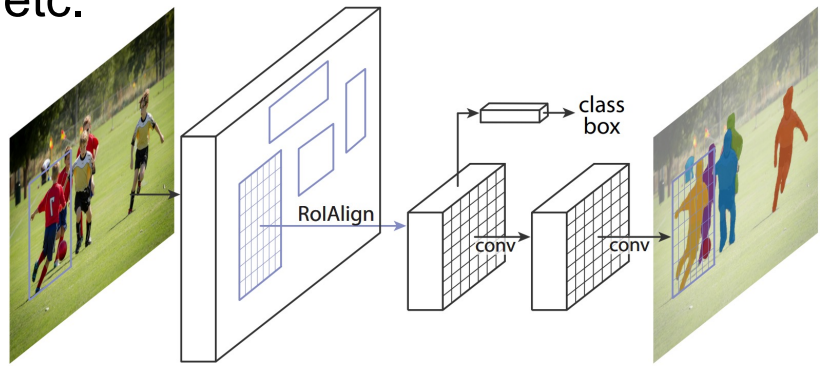


Approaches

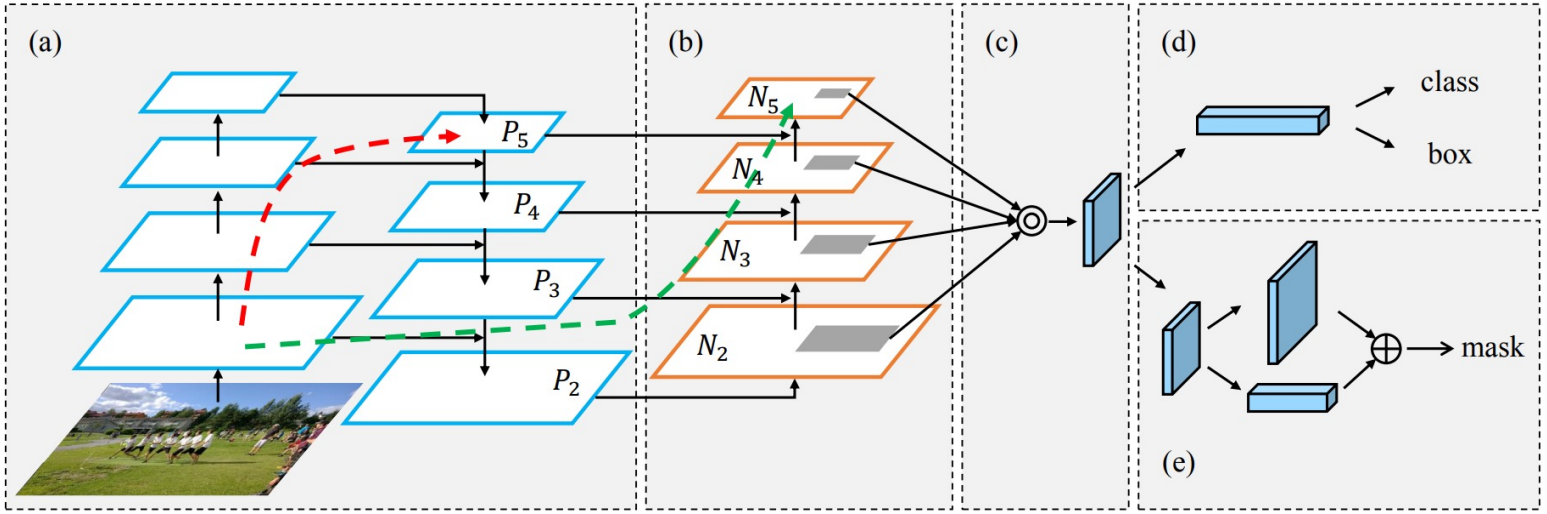
InstanceCut, DWT, SAIS, DIN, FCIS, SGN, Mask-RCNN, PANet etc.



DWT [Bai et al, CVPR'17]



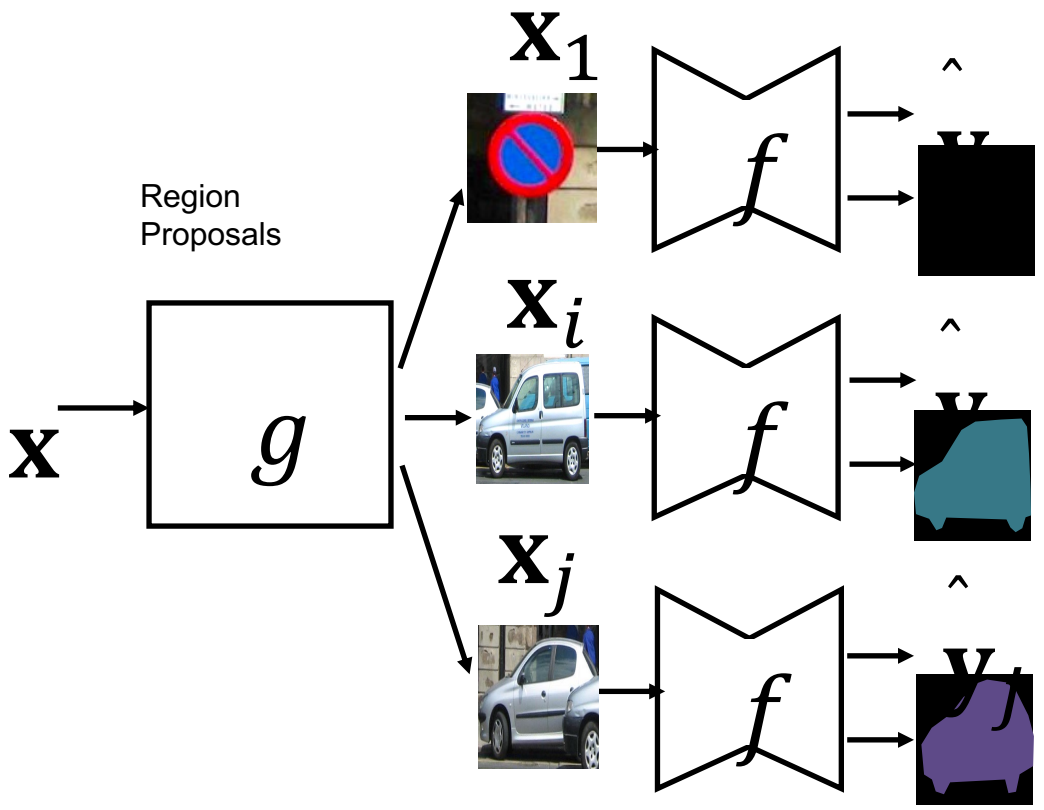
Mask-RCNN [He et al, ICCV'17]



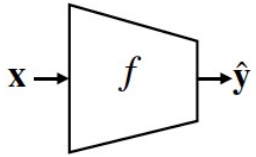
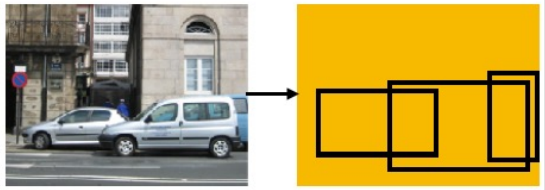
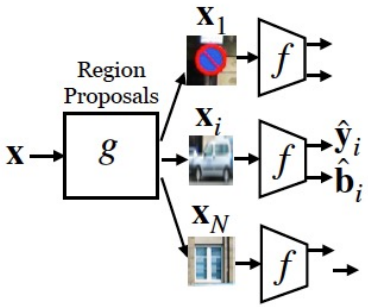
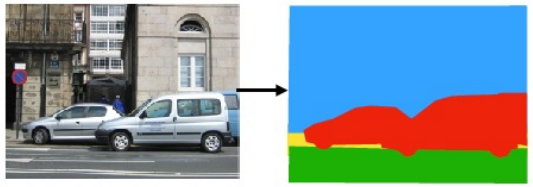
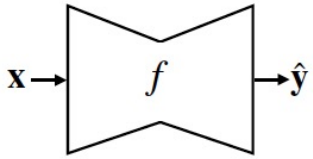
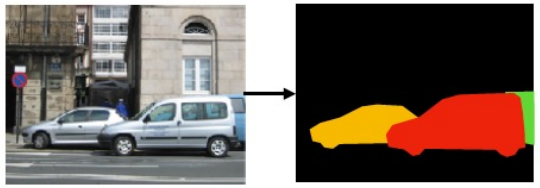
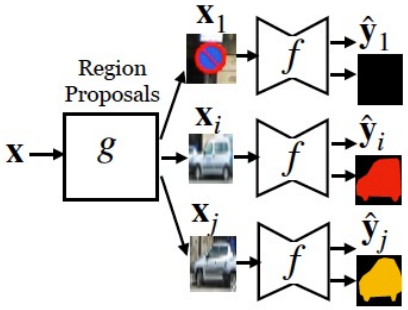
PANet [Liu et al, CVPR'18]

Solution:

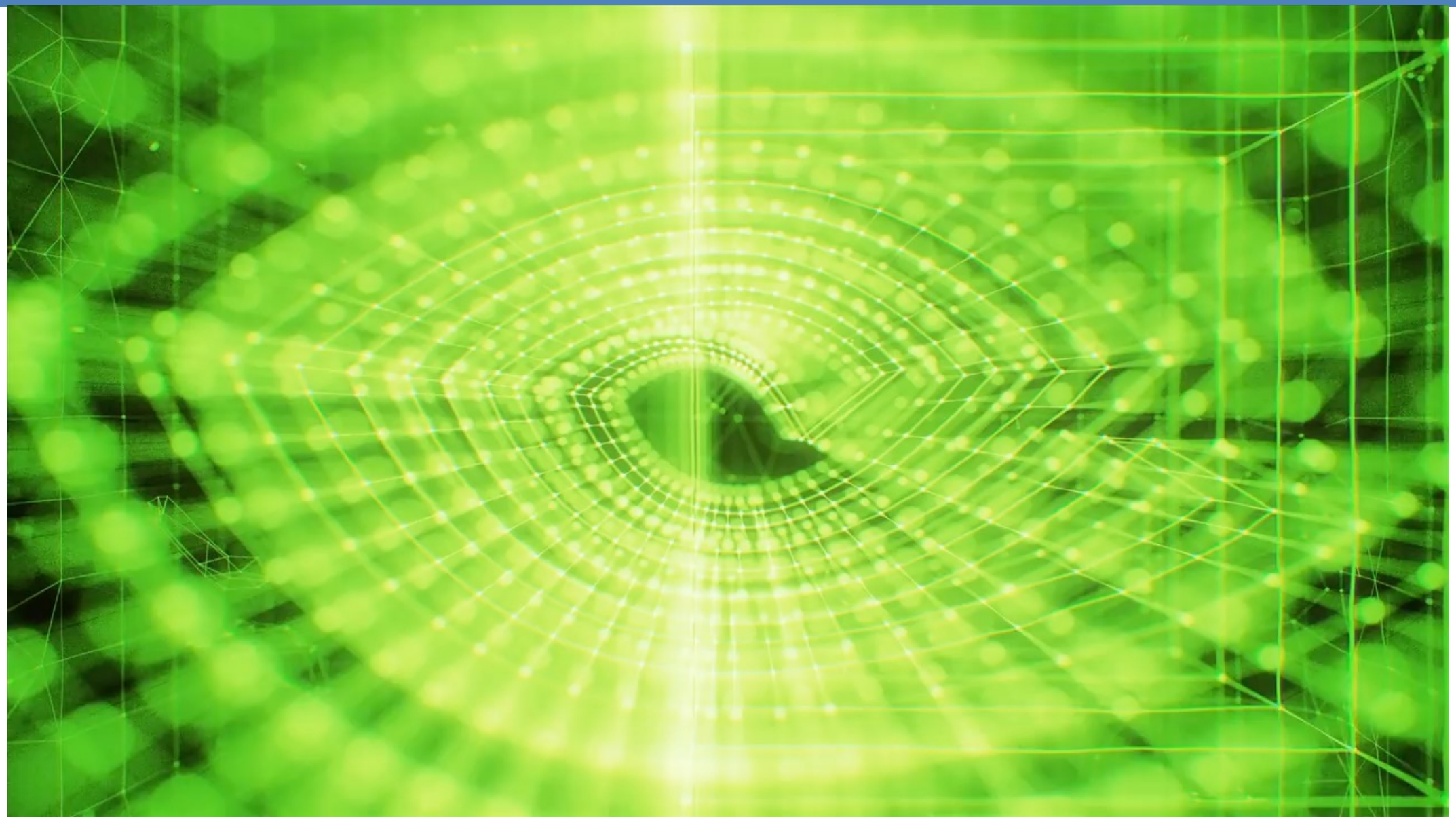
Combine classification, regions proposal and segmentation



Summary

	Question	Mapping	Architecture	Functional form
Classification	Is object class c present in the image?	 → `Car` `Building` `Road`		$\hat{\mathbf{y}} = f(\mathbf{x})$
Detection	What is the location of all the instances of class c?			$\{\mathbf{x}_i\}_{i=1}^{N} = g(\mathbf{x})$ $(\hat{\mathbf{y}}_i, \hat{\mathbf{b}}_i) = f(\mathbf{x}_i)$
Segmentation	Is object class c present in the pixel [n,m]?			$\hat{\mathbf{y}}[n,m] = f(\mathbf{x})$
Instance segmentation	Is instance *i* of object class *c* present in the pixel [n,m]?			$\{\mathbf{x}_i\}_{i=1}^{N} = g(\mathbf{x})$ $(\hat{\mathbf{y}}_i, \hat{\mathbf{s}}_i[n,m]) = f(\mathbf{x}_i)$

VISION FOR MANIPULATION

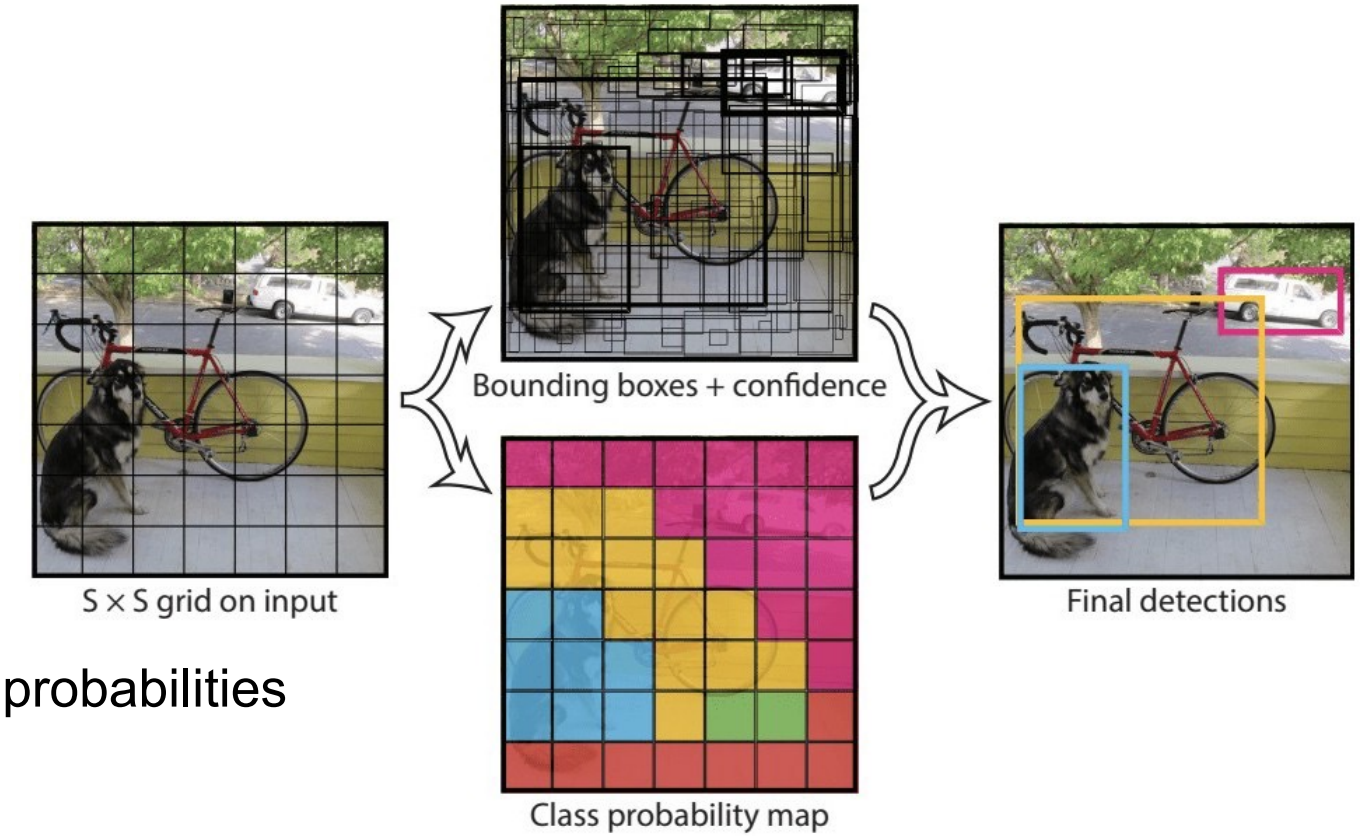


https://www.youtube.com/watch?v=yVGViBqWtBI

Tremblay, J., To, T., Sundaralingam, B., Xiang, Y., Fox, D., & Birchfield, S. (2018). Deep object pose estimation for semantic robotic grasping of household objects. *arXiv preprint arXiv:1809.10790*.

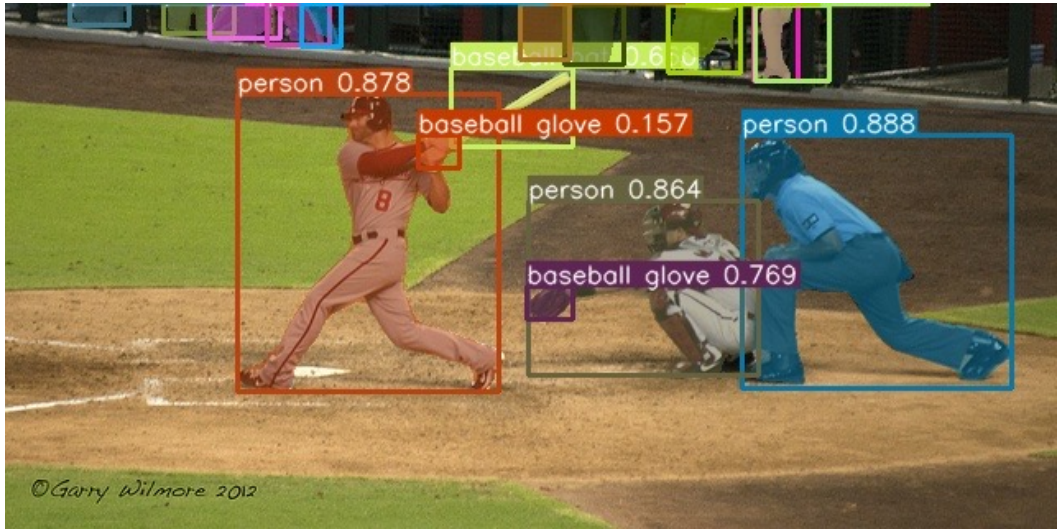
Yolo

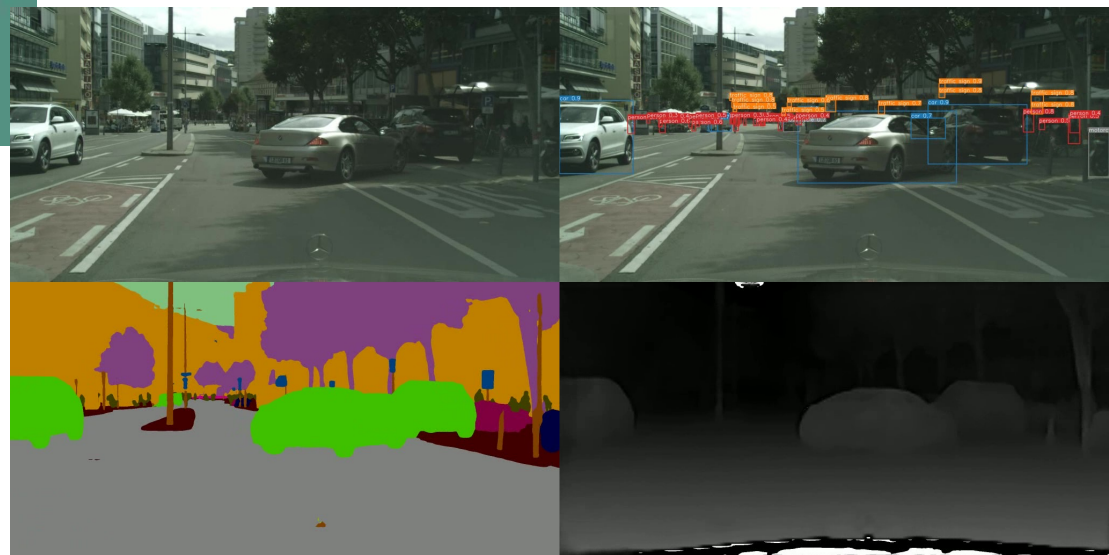
- You Only Look Once
- Much faster than R-CNN:
  - Yolo works in real time
- Approach:
  - Divide image into grids
  - Predict bounding boxes and class probabilities for each cell
  - Non-max suppression



YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors

- Pose estimation
- Instance Segmentation
- Semantic Segmentation
- Caption:
  - A group of people playing a game of tennis





https://github.com/WongKinYiu/yolov7

OnePose++: Keypoint-Free One-Shot Object Pose Estimation without CAD Models

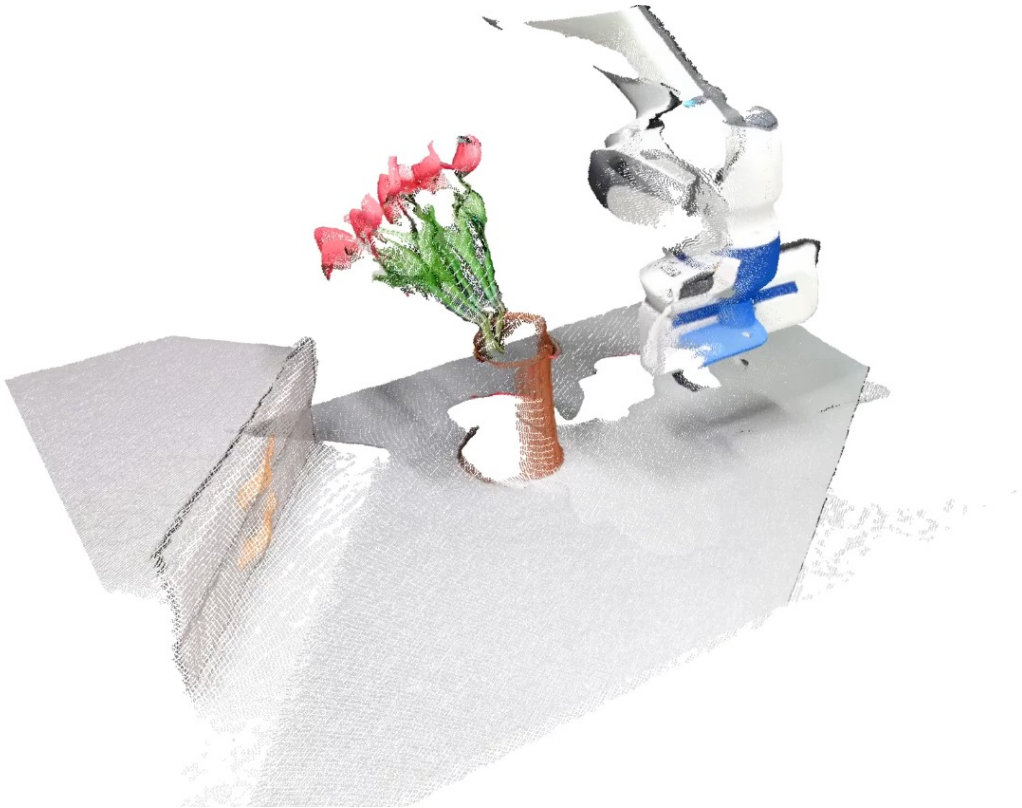
Xingyi He* Jiaming Sun* Yu'ang Wang Di Huang Hujun Bao Xiaowei Zhou

NeurIPS 2022

https://www.youtube.com/watch?v=x5CL5JfIHzw

He, X., Sun, J., Wang, Y., Huang, D., Bao, H., & Zhou, X. (2022). Onepose++: Keypoint-free one-shot object pose estimation without CAD models. Advances in Neural Information Processing Systems, 35, 35103-35115.

VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models



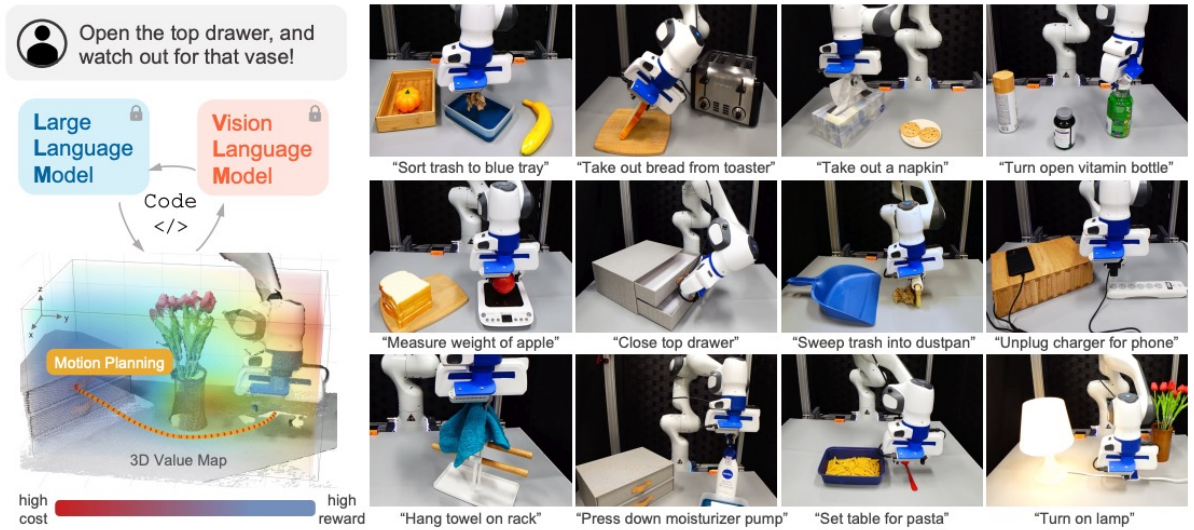


Figure 1: VoxPoser extracts language-conditioned **affordances** and **constraints** from LLMs and grounds them to the perceptual space using VLMs, using a code interface and without additional training to either component. The composed map is referred to as a 3D value map, which enables **zero-shot** synthesis of trajectories for large varieties of everyday manipulation tasks with an **open-set of instructions** and an **open-set of objects**.

https://arxiv.org/abs/2307.05973