

Computer Vision

Fundamentals of Artificial Intelligence

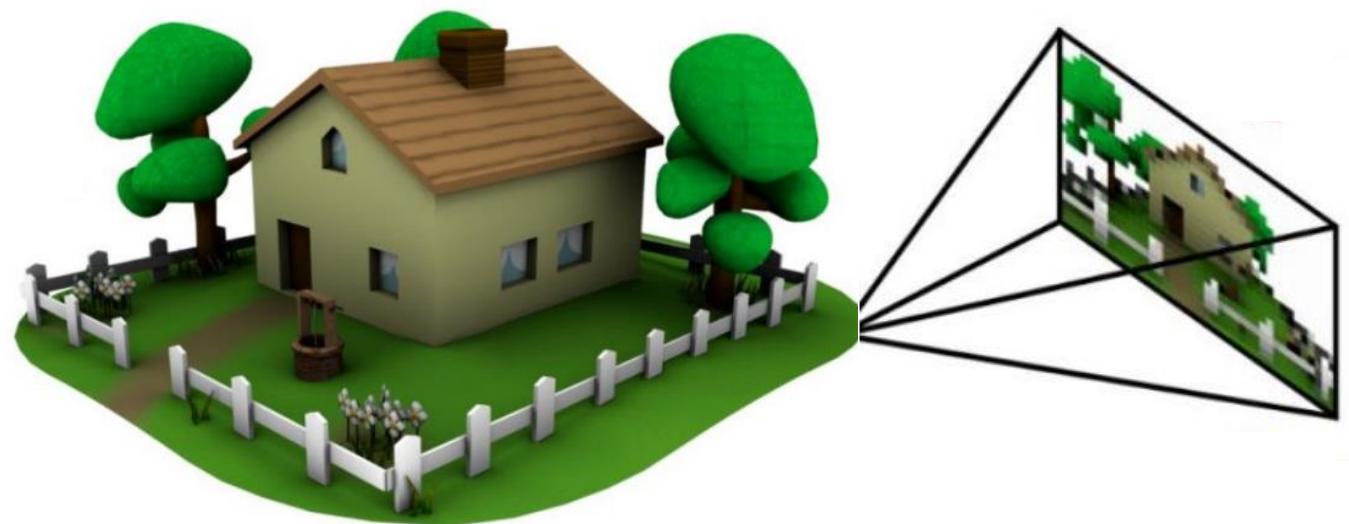
Goal of Computer Vision

extract semantic information from digital image data

to be used for decision making support or automated systems

challenging problem:

images are only 2D projections
of the 3D world



nowadays heavily powered by artificial intelligence (AI), especially machine learning (ML)

Applications of Computer Vision

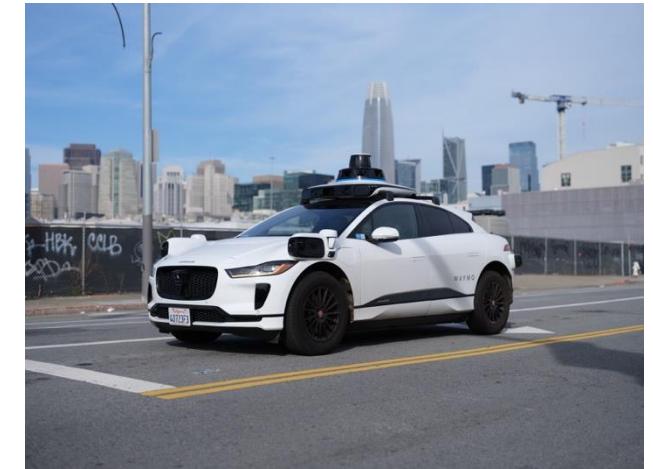
facial recognition



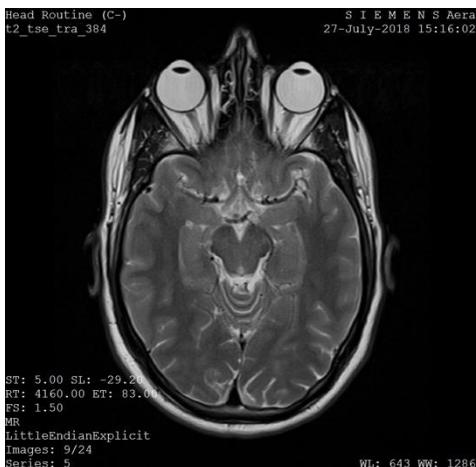
automated inspection



autonomous driving



medical imaging



optical character recognition

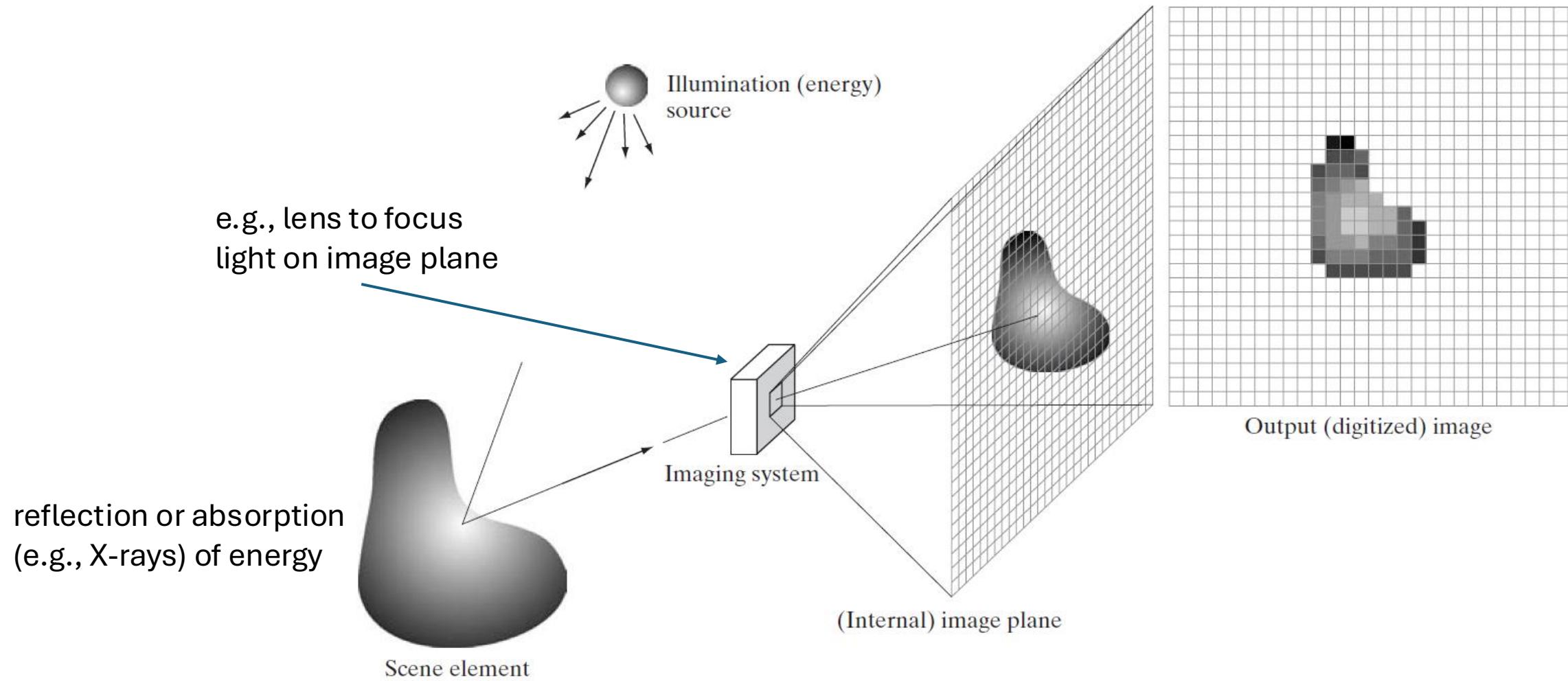


augmented reality



and many more ...

Digital Image Acquisition Process



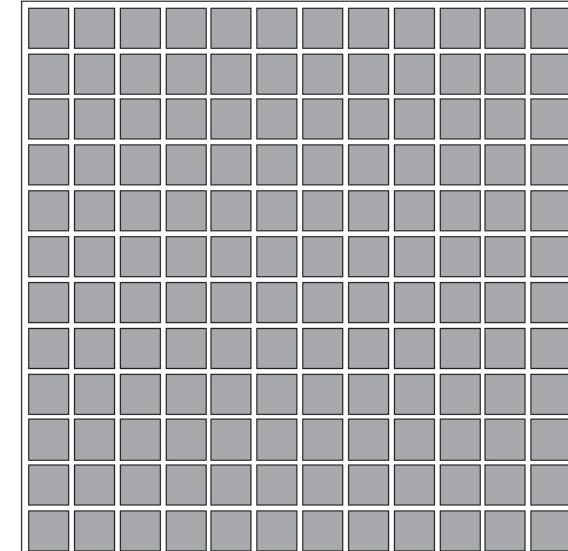
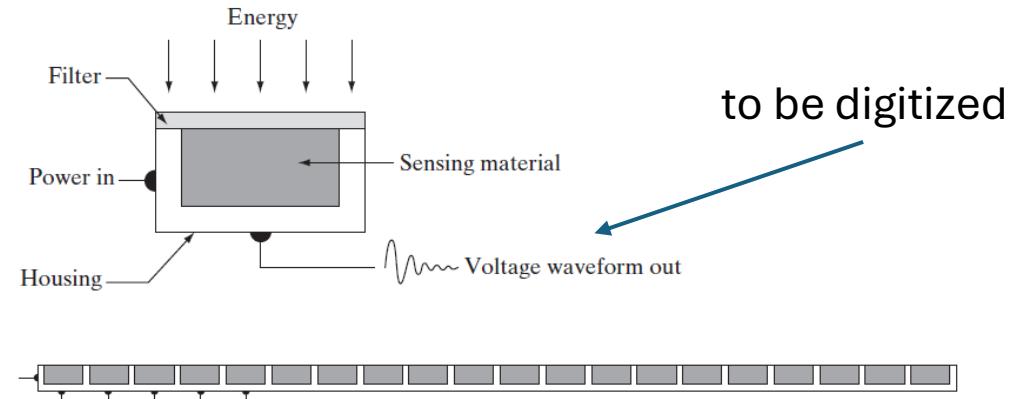
Photons to Electrons

largely replaced chemical and analog imaging

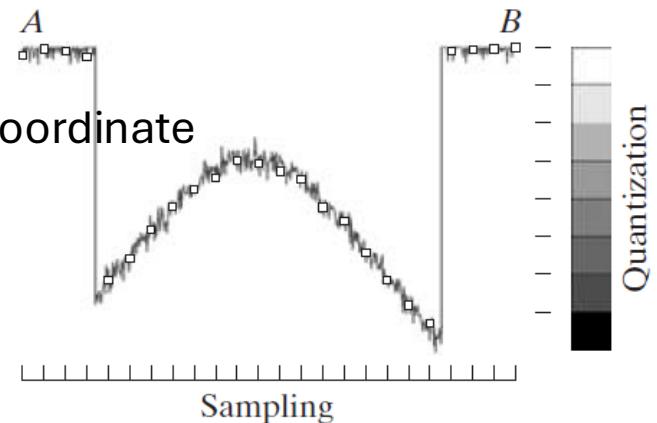
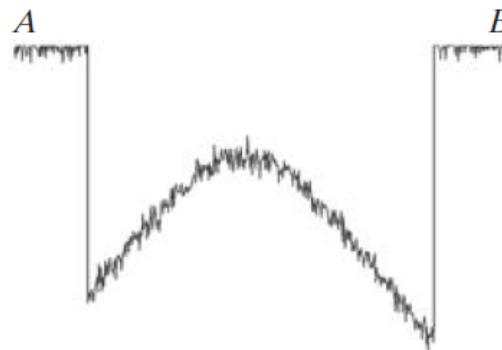
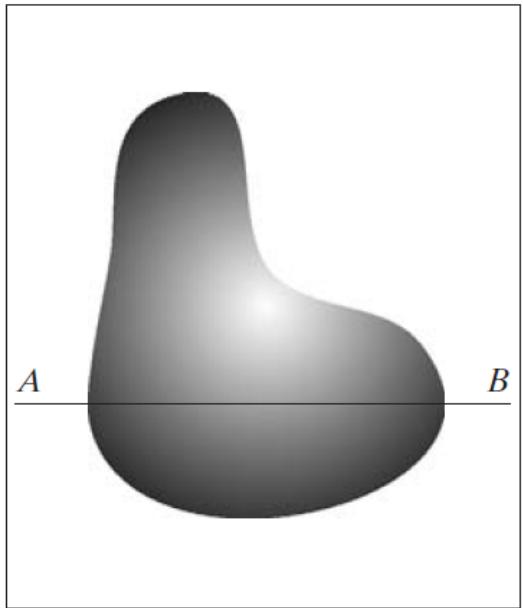
single sensor: photodiode

sensor strips: motion perpendicular to strip for imaging in other direction, used in CT

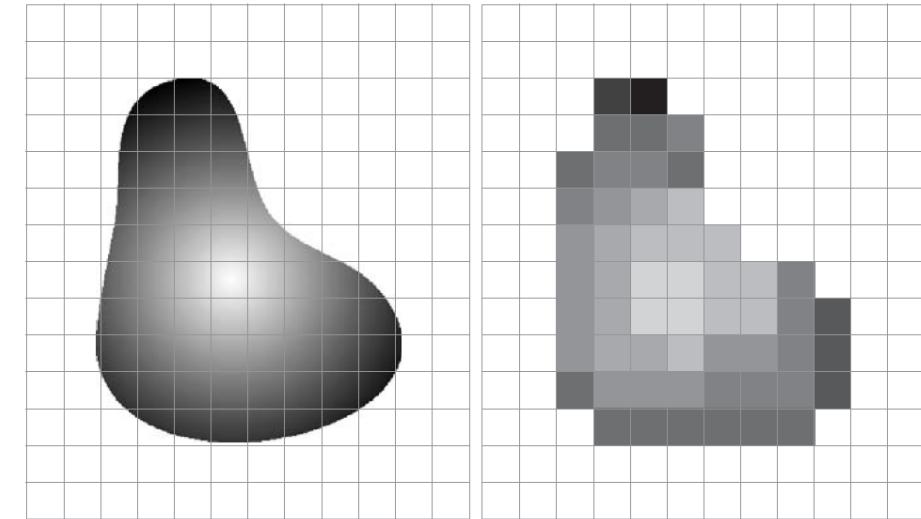
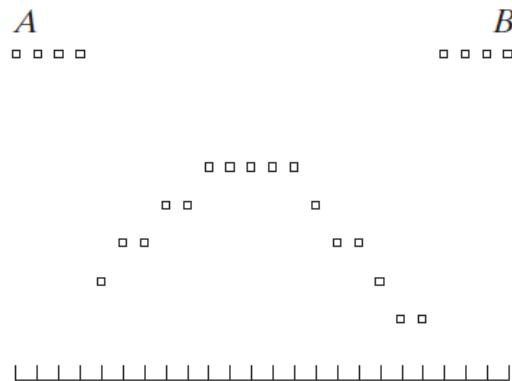
sensor arrays: used in digital cameras (in focal plane, outputs proportional to integral of light received at each sensor)



Digitizing: Sampling & Quantization



digitizing the coordinate
values: pixels



digitizing the amplitude values:
discrete intensity levels

Discrete Intensity Levels

image data: 2D grid (matrix) of pixels with different intensity values

intensity resolution:

common to use one byte (2^8) to express possible intensity values (8-bit image)

→ 0 = black, 255 = white

storage requirement for, e.g., 1024×1024 8-bit image (grayscale): ~1 MB
($\times 3$ for RGB color images)

0	2	15	0	0	11	10	0	0	0	9	9	0	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	249	255	240	255	129	0	5	
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

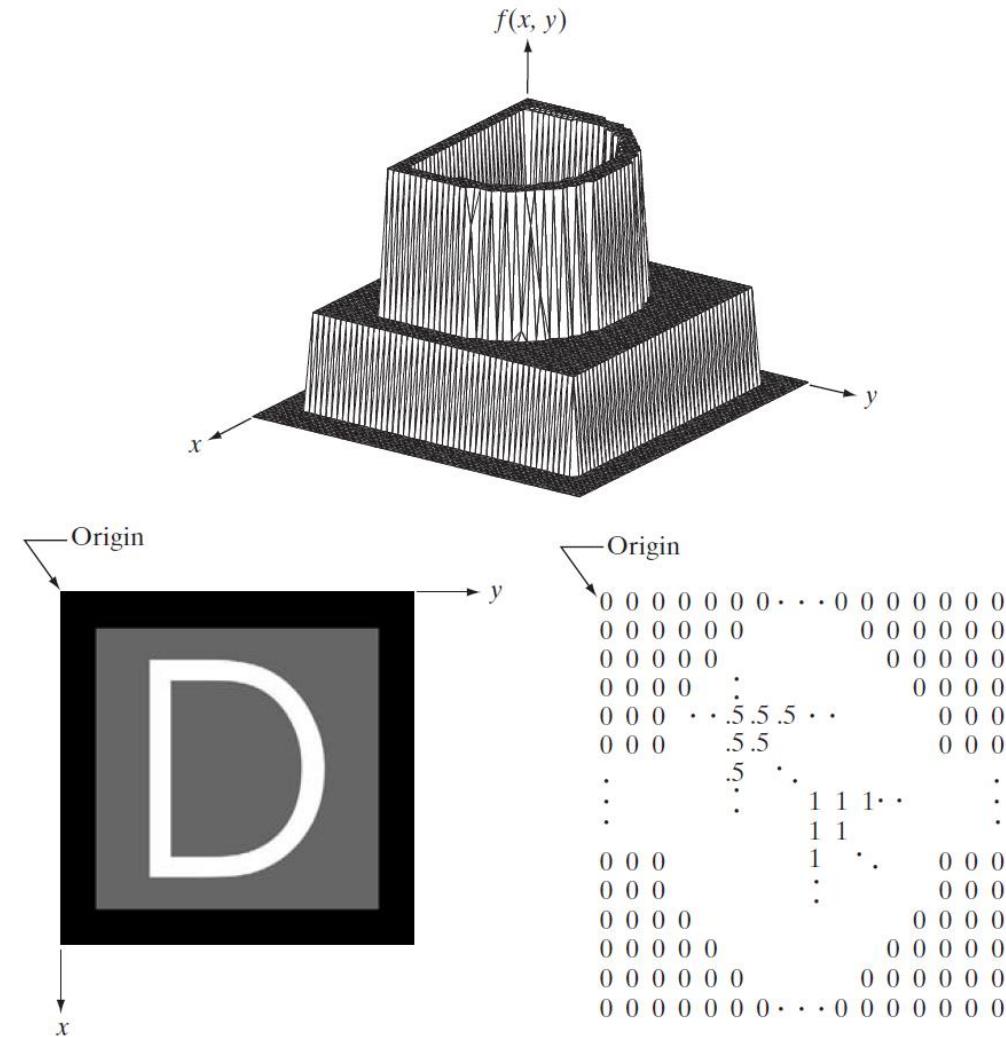
[source](#)

Image as Two-Dimensional Function

image can be seen as function $f(x, y)$:
intensity at position (x, y)

digital image: discrete (sampled and quantized) version of this function

$$f(x, y) = \begin{bmatrix} f(0, 0) & f(0, 1) & \cdots & f(0, N - 1) \\ f(1, 0) & f(1, 1) & \cdots & f(1, N - 1) \\ \vdots & \vdots & & \vdots \\ f(M - 1, 0) & f(M - 1, 1) & \cdots & f(M - 1, N - 1) \end{bmatrix}$$

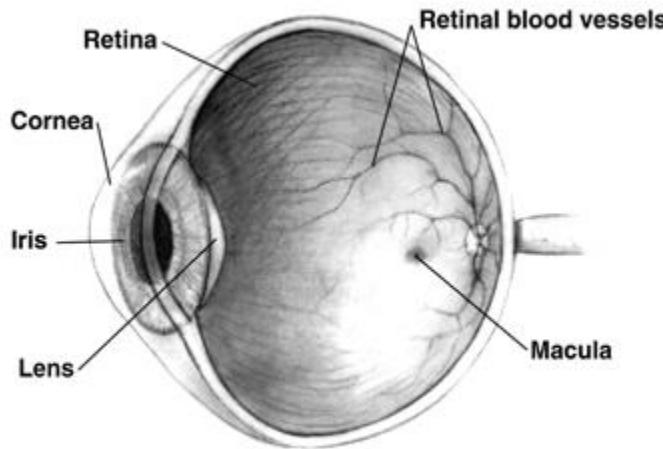
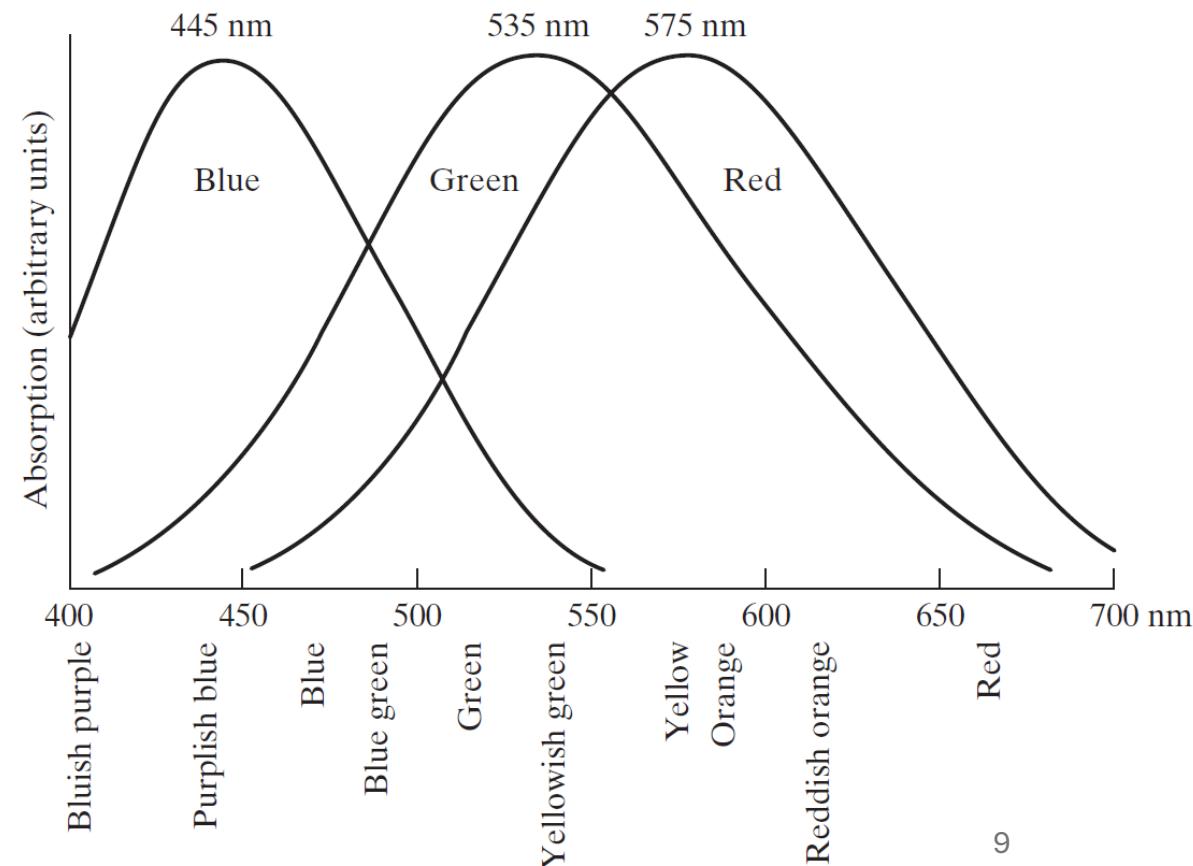


Colors in the Human Eye

retina: image sensor of human eye

two kinds of light receptors on surface of retina: rods (monochromatic vision) and cones (color-sensitive cells)

three types of cones: red, green, blue
→ combinations of red, green, and blue create perceived colors



Primary Colors

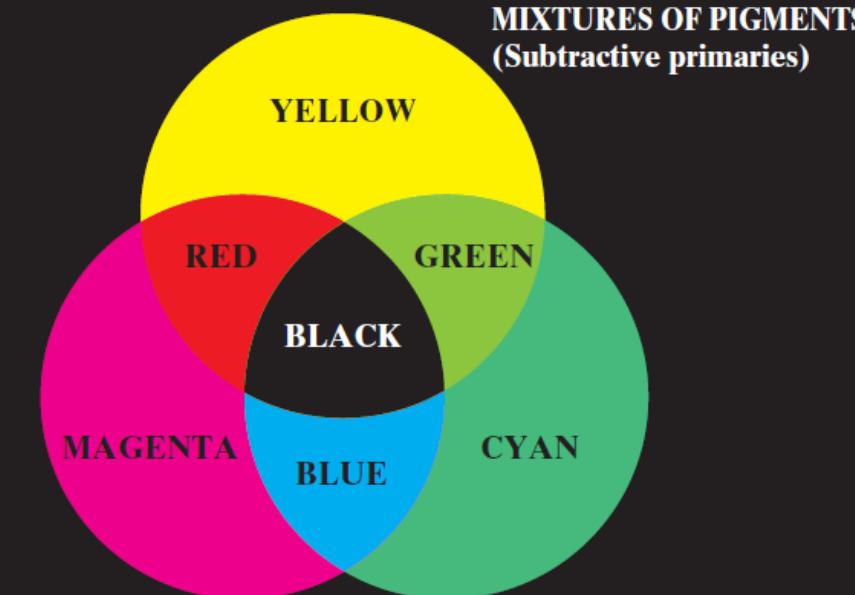
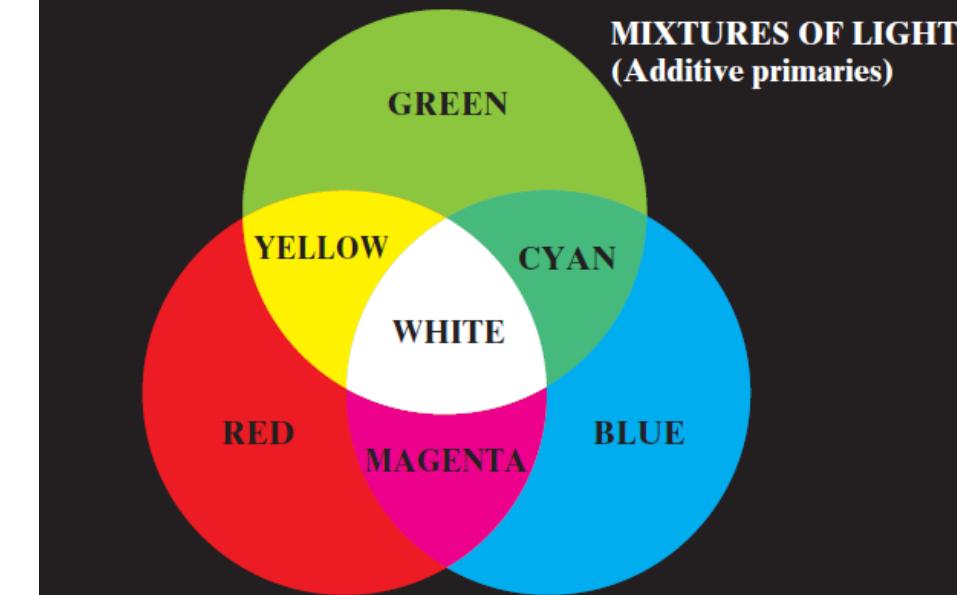
standardization:

use of specific wavelengths for three primary colors red, green, and blue

→ can reproduce (almost) all visible colors by mixing with different intensities

for printing:

primary colors of pigments each absorb one primary color of light

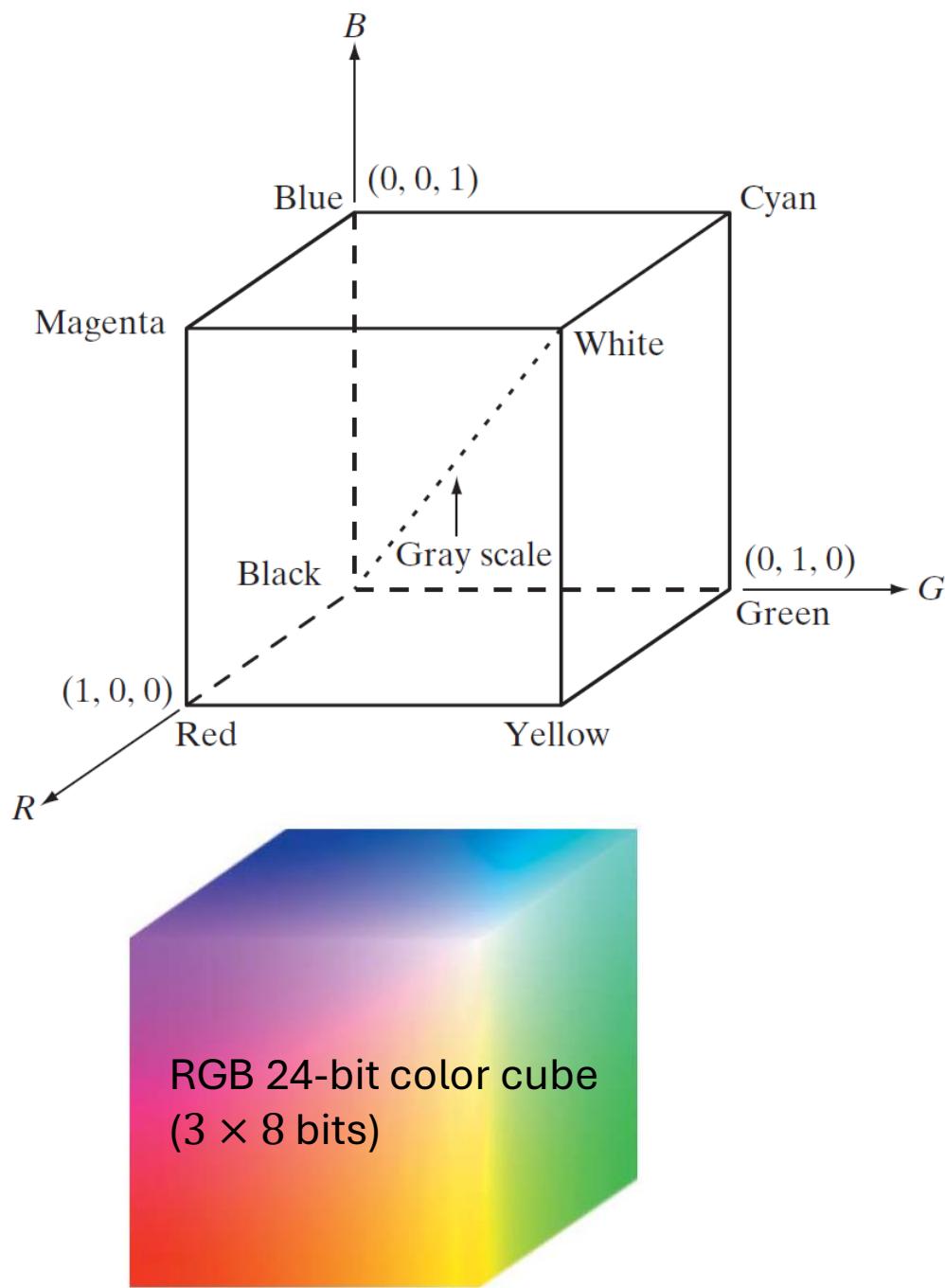
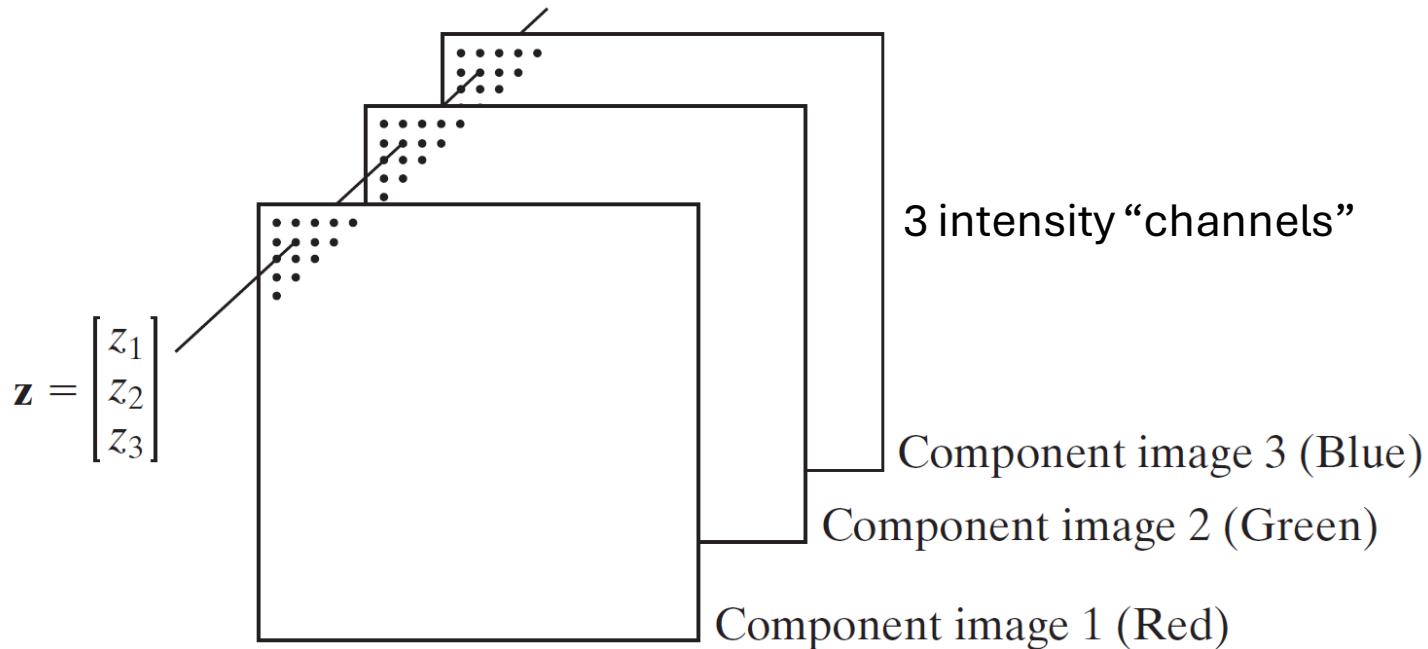


PRIMARY AND SECONDARY COLORS
OF LIGHT AND PIGMENT

RGB Color Model

color model for monitors and cameras
(other color models: CMYK, HSV, ...)

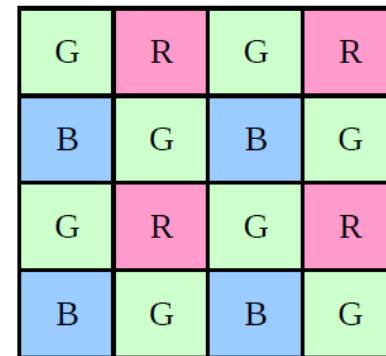
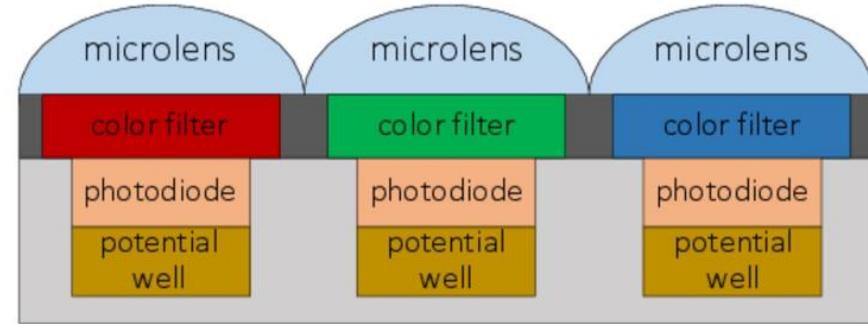
color pixels:



Color Sensing

single image sensor with one of three color filters in front of each pixel

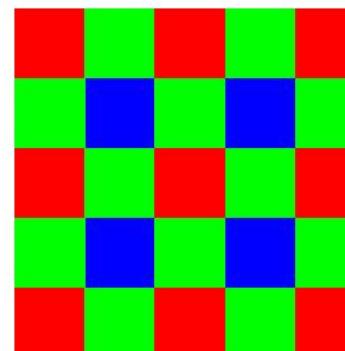
estimation of missing color values for each pixel by interpolation from neighboring pixels
→ full-color image



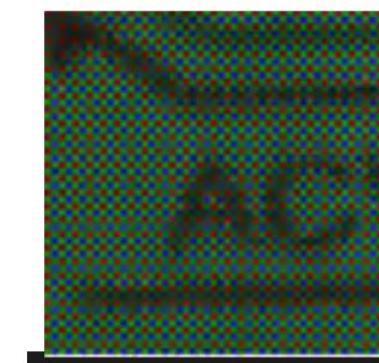
Bayer RGB Pattern

rGb	Rgb	rGb	Rgb
rgB	rGb	rgB	rGb
rGb	Rgb	rGb	Rgb
rgB	rGb	rgB	rGb

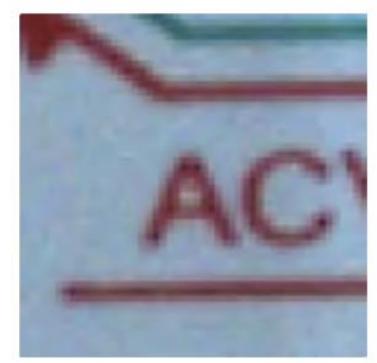
Interpolated Pixels



Bayer Pattern
(Color Filter Mosaic)



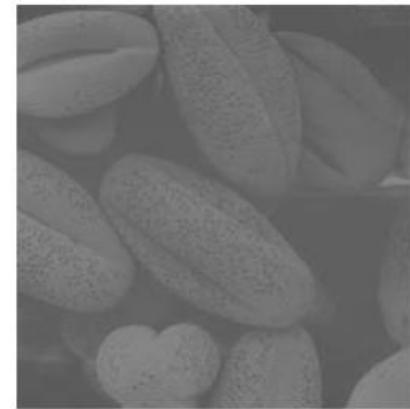
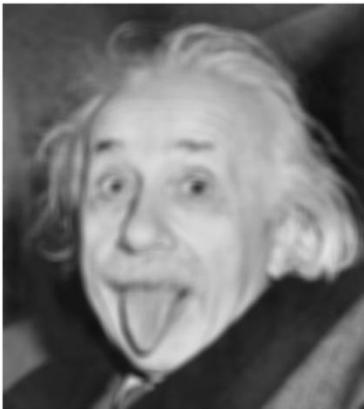
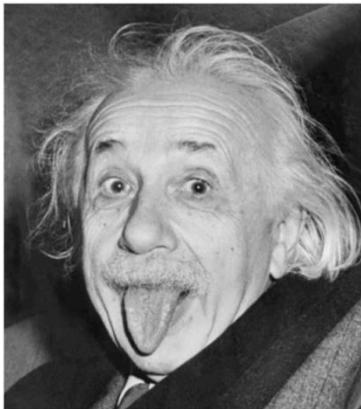
Raw Image



Interpolated Image

Image Processing

transformations from image to image
(such as scaling, smoothing, sharpening, or contrast stretching)

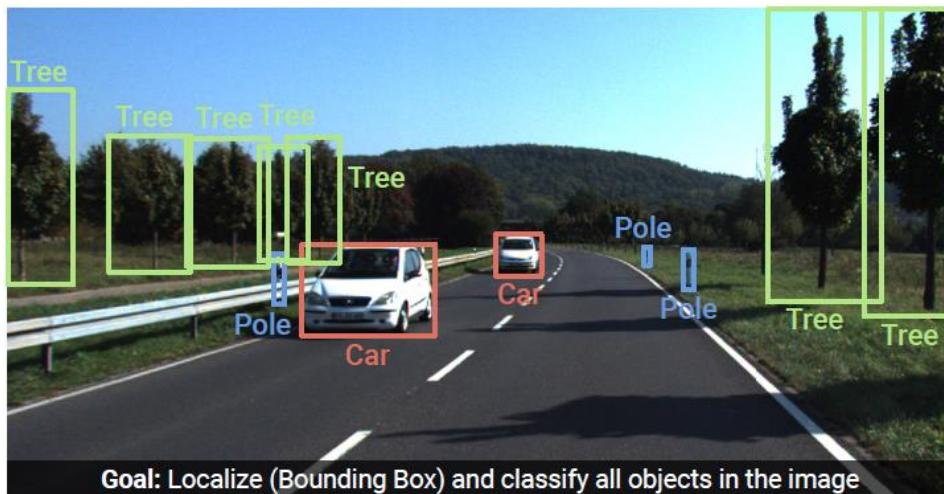


to facilitate either machine perception or just human interpretation

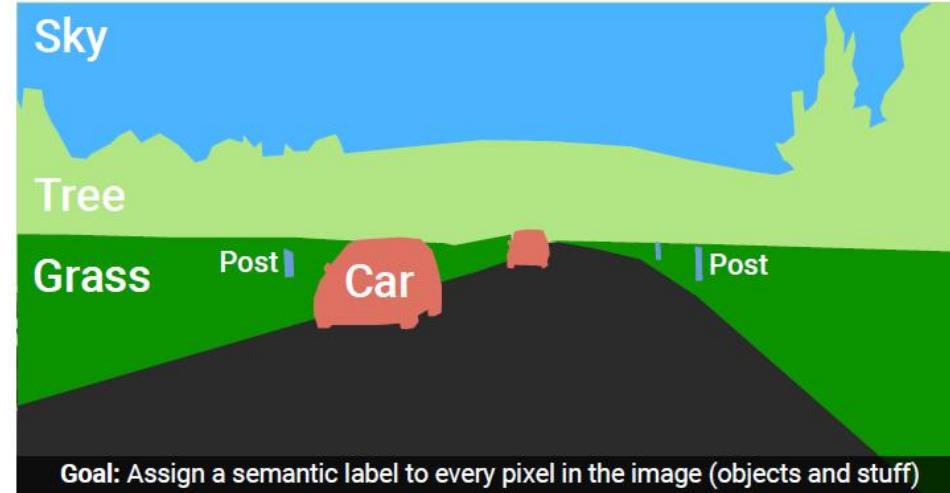
Image Understanding (Recognition)



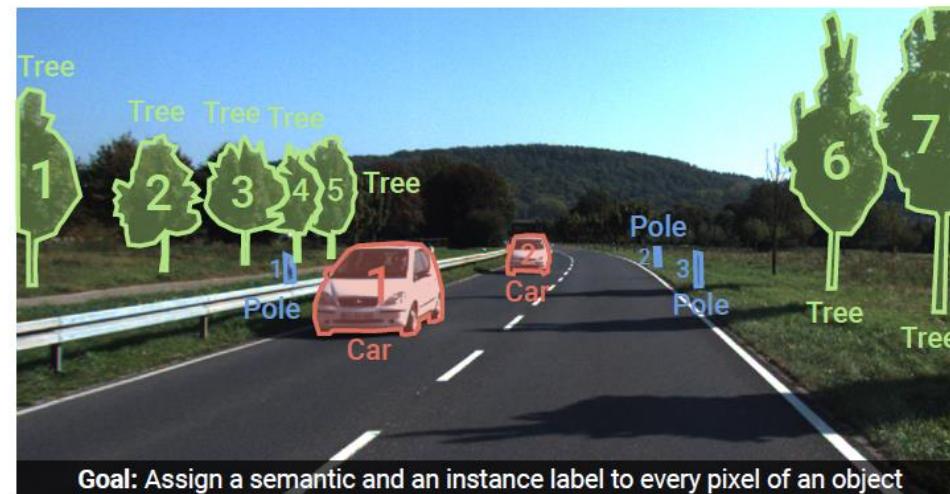
Image Classification



Object Detection



Semantic Segmentation



Instance Segmentation

Need for ML

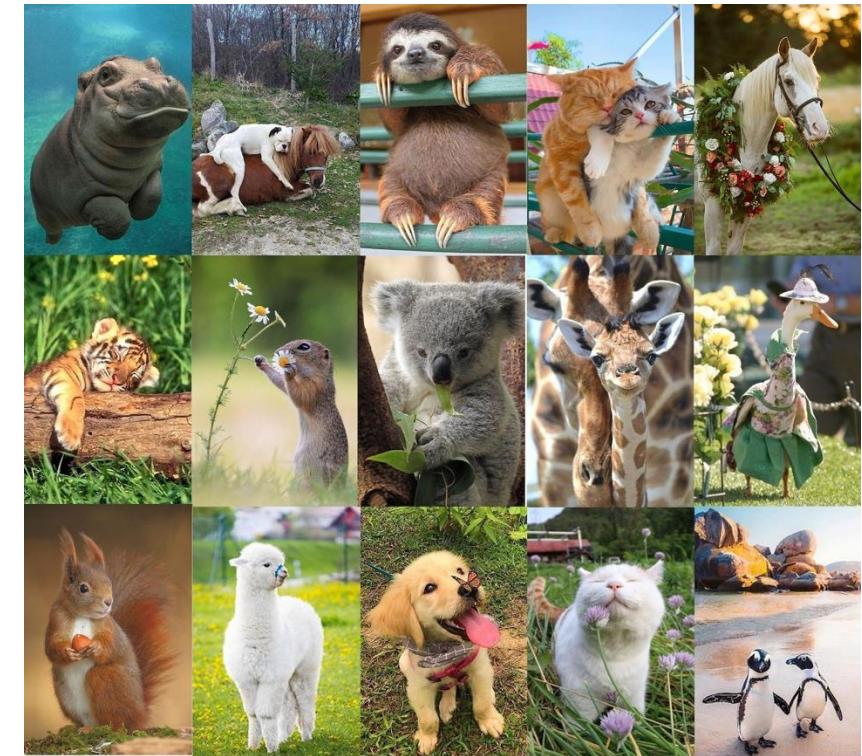
prime example (and also foundation for detection and segmentation):

image classification (whole-image class recognition) according to generic object categories (e.g., cat)

plain keypoint-feature matching only really works for specific instances of a class

→ need to compare with generic objects (e.g., kind of “abstract cat”)

Let's learn it from data ...

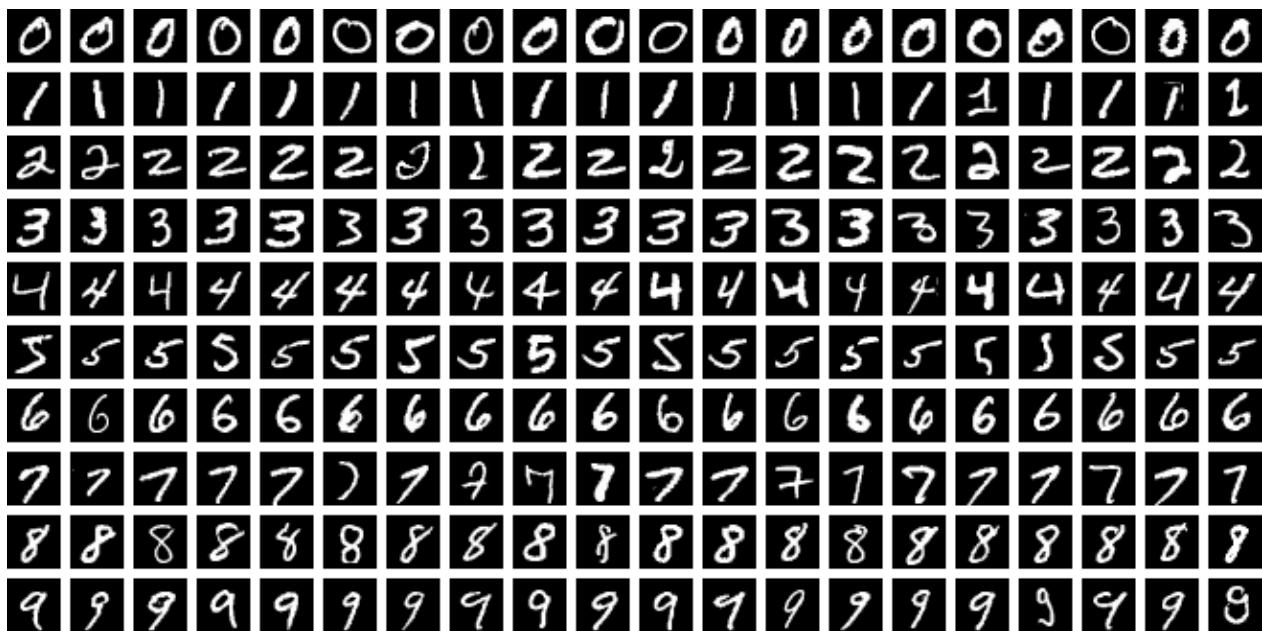


What if you haven't seen this very cat before?

Image Classification

MNIST

- Modified National Institute of Standards and Technology
- handwritten digits (10 classes)
- black and white images
- 28×28 pixels
- 60k training and 10k test images



CIFAR-10

- Canadian Institute for Advanced Research
- 10 different labeled object classes
- color images
- 32×32 pixels
- 50k training and 10k test images

airplane



automobile



bird



cat



deer



dog



frog



horse



ship

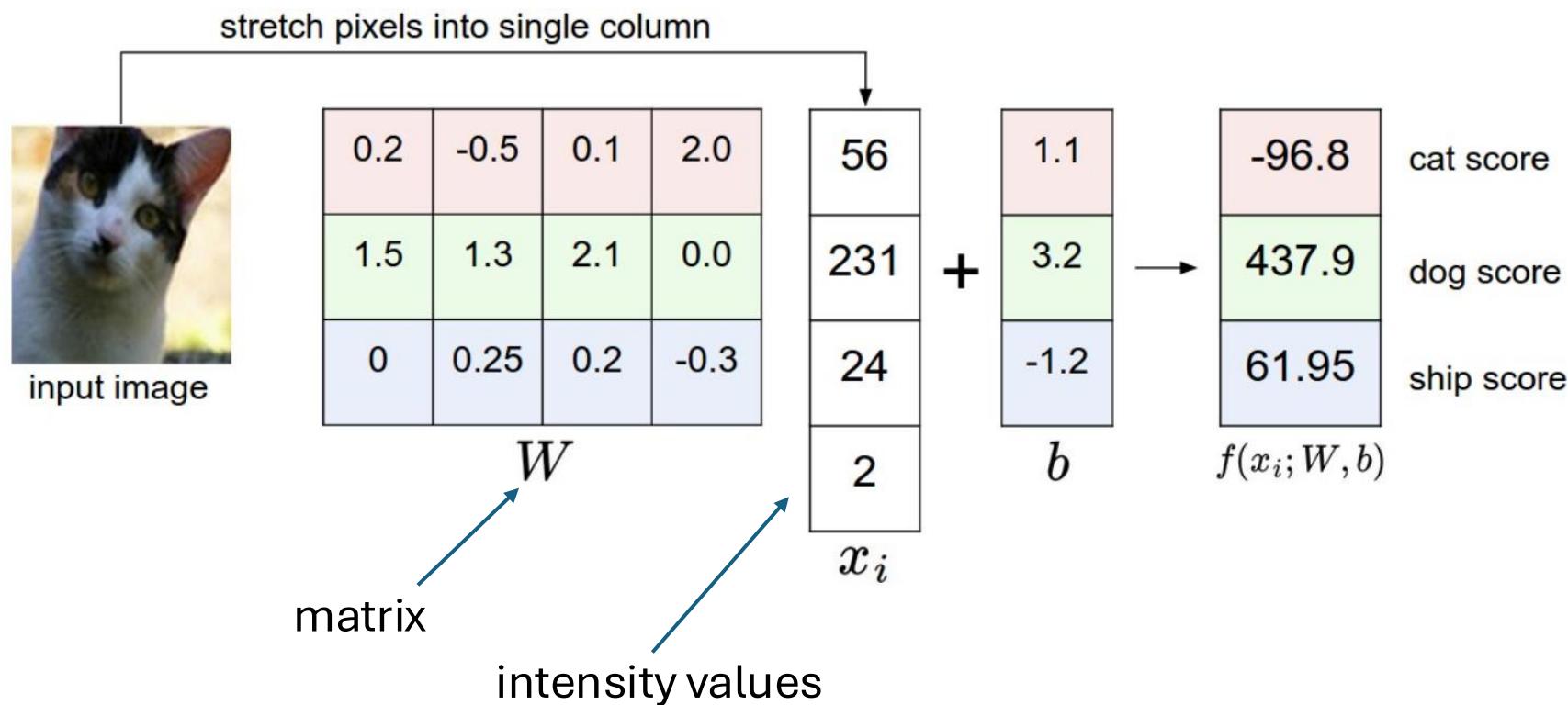


truck



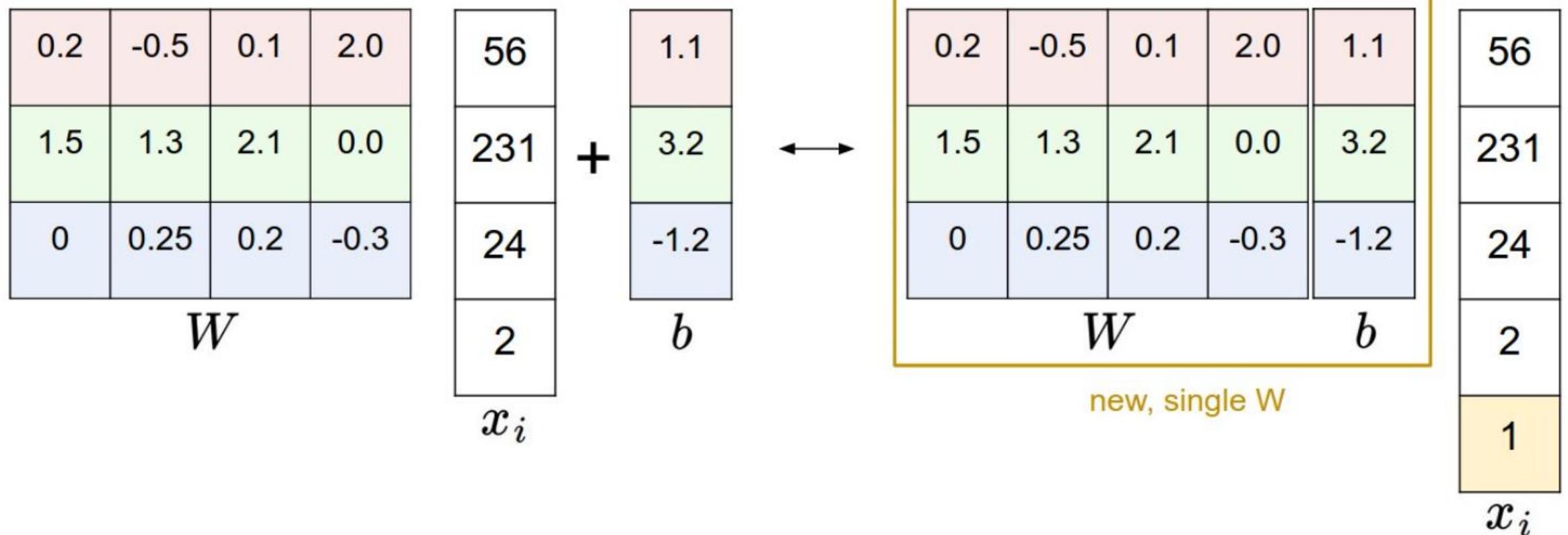
Image Classification with Linear Regression

simplified example: 4-pixel image, 3 classes

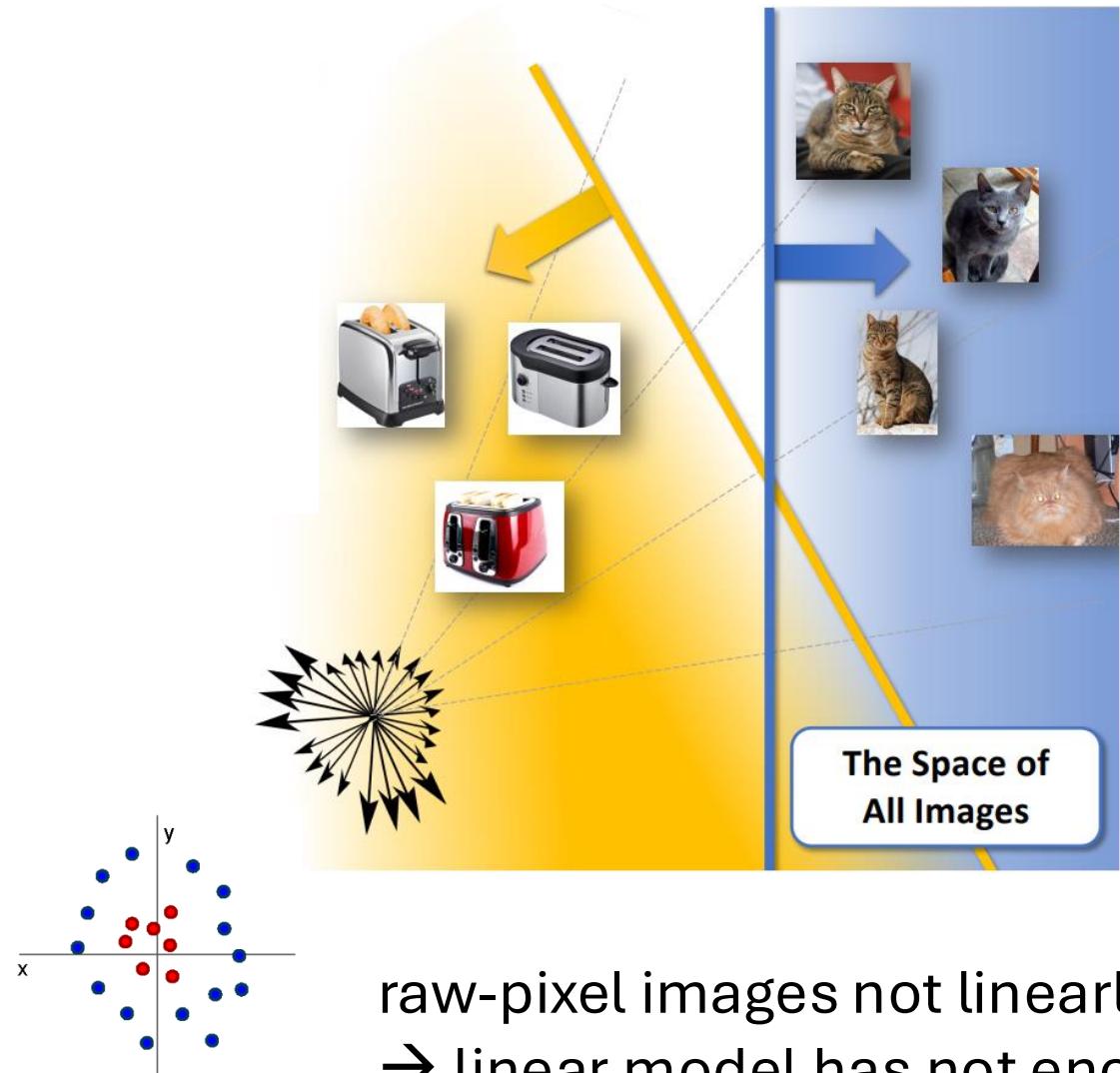


need for one common image size per model

Simplified Matrix Multiplication: Bias Trick



geometric interpretation: separating hyperplanes



matching interpretation: generic class templates

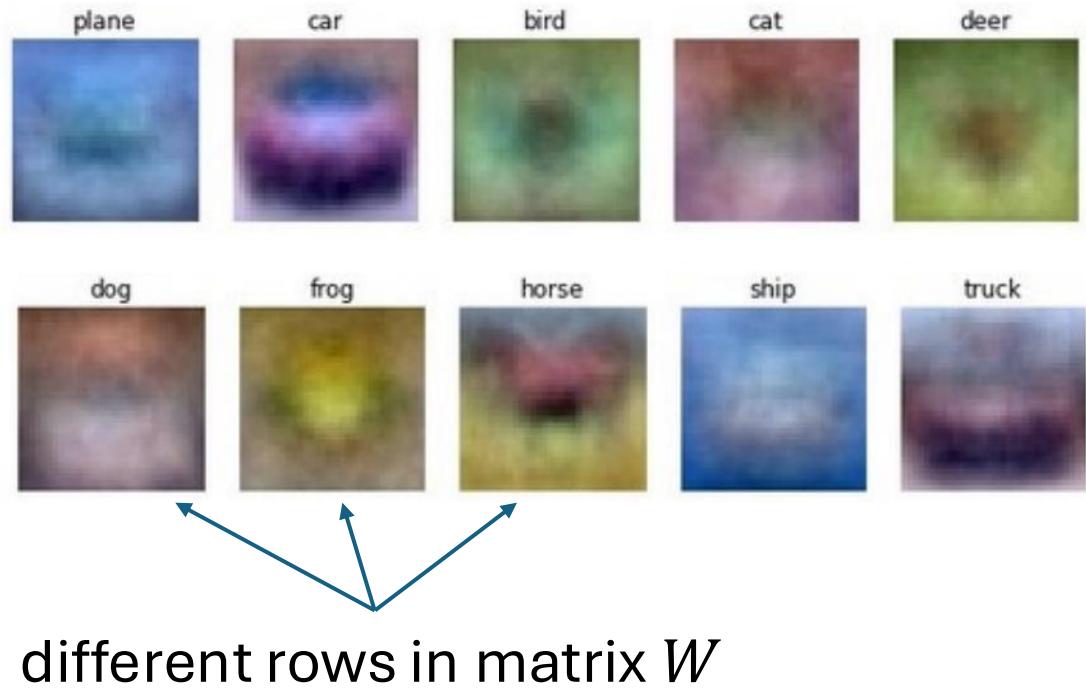


Image Classification with kNN

choose an image distance, e.g., L1 distance:

$$d(I_1, I_2) = \sum_p |I_1(p) - I_2(p)|$$

test image				training image				pixel-wise absolute value differences				→ 456
56	32	10	18	10	20	24	17	46	12	14	1	
90	23	128	133	8	10	89	100	82	13	39	33	
24	26	178	200	12	16	178	170	12	10	0	30	
2	0	255	220	4	32	233	112	2	32	22	108	

Image Classification with kNN

training examples CIFAR-10 data set

10 classes

airplane	
automobile	
bird	
cat	
deer	
dog	
frog	
horse	
ship	
truck	

10 nearest neighbors to some test images

The figure illustrates the k-Nearest Neighbors (kNN) classification process. On the left, a grid of 400 images represents the training set for 10 different classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each class is represented by a 4x10 grid of images. On the right, a series of 10 rows shows the results of a search for the 10 nearest neighbors for specific test images. Each row starts with a small image followed by an arrow pointing to a 10x10 grid of images. These grids represent the top 10 similar images found in the training set for each respective test image.

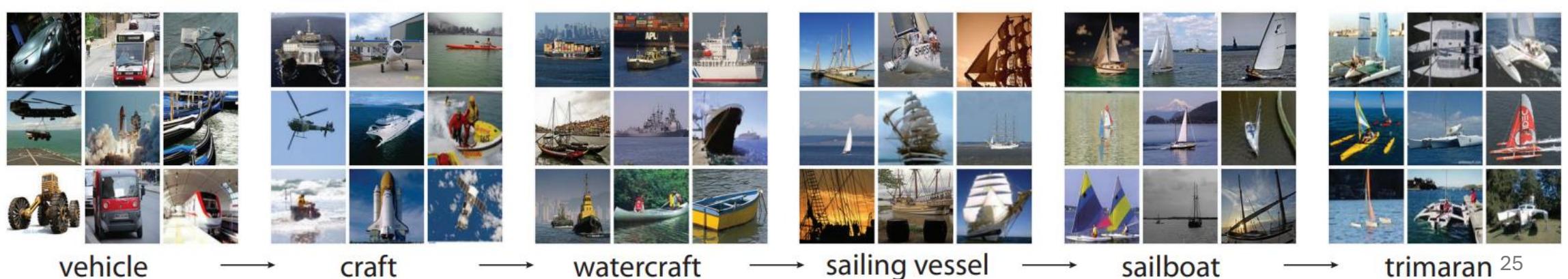
better than random guessing, but also not very impressive

In Reality: Lots of Categories



ImageNet

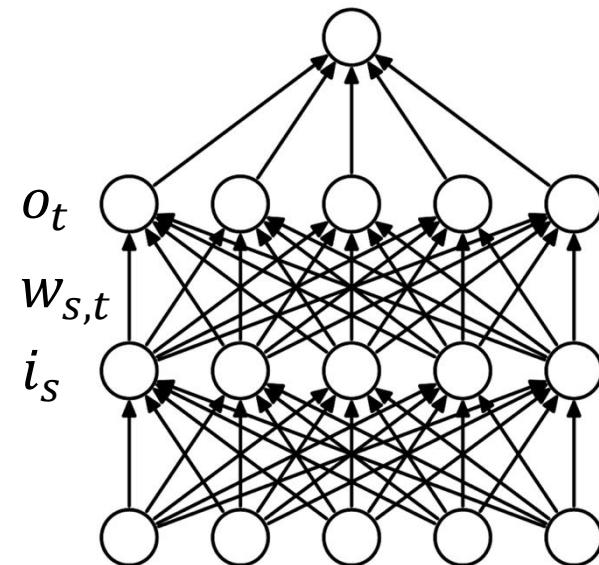
- more than 14 million color images with varying sizes
- more than 20k labeled categories



Recap: Feed-Forward Neural Networks

computation in usual feed-forward network:
matrix multiplication of scalar inputs and weights

$$o_t = \sum_s w_{s,t} i_s$$



dropped dimension of different training observations
(in mini-batch) in this view

Inductive Bias

plain feed-forward networks very inefficient for image classification:
make no use of spatial structure (locality of objects)
→ need to learn it from scratch

better way: introduce it right in the architecture of the ML method
(called inductive bias)
→ convolution with spatial filters (**learned** rather than handcrafted)

Convolutional Neural Networks (CNN or ConvNet)

Grid-Like Data

image data: 2-D grid of pixels (spatial structure)

convolutional networks:

neural networks using learned convolution kernels as weights (in place of general matrix multiplications)

→ highly regularized feed-forward networks

scalar values (like in usual feed-forward network)

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0

source

Convolution Operation

2D convolution

actually, correlation (convolution would have – here):

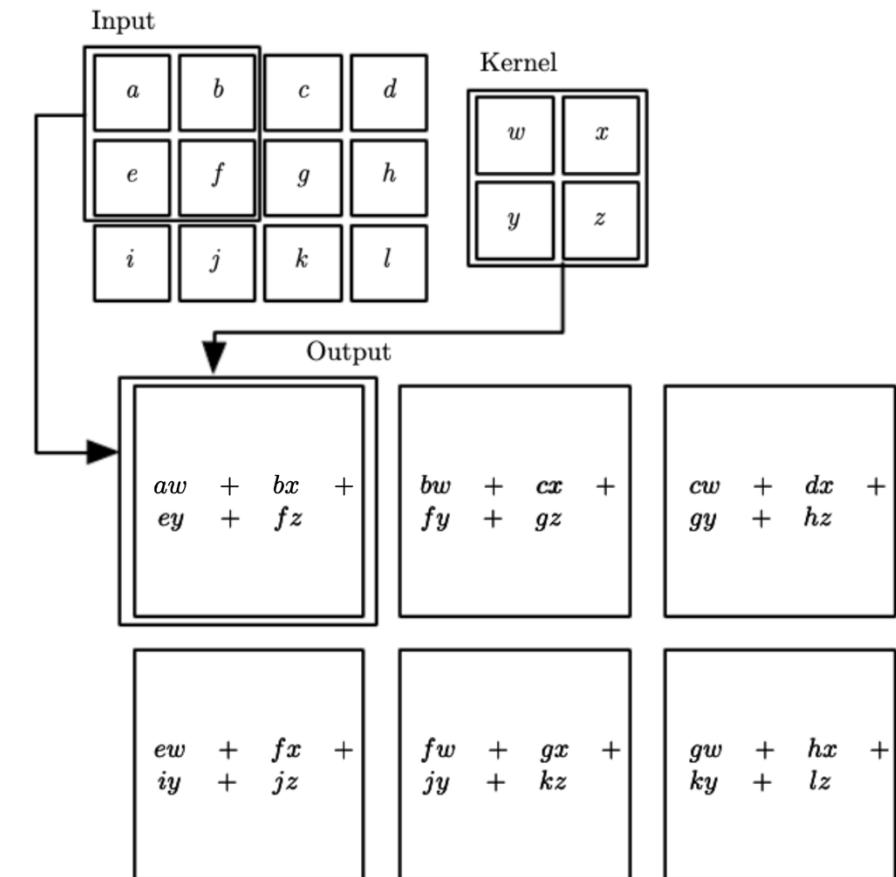
$$o_{x,y} = \sum_{s,t} w_{s,t} i_{x+s,y+t}$$

feature map
(transformed image)

kernel
(learned)

image
(input or feature map)

(again, dropping dimension of different training observations)



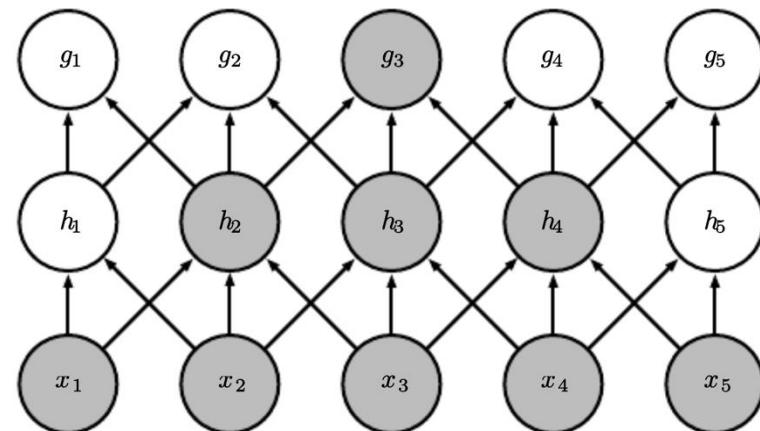
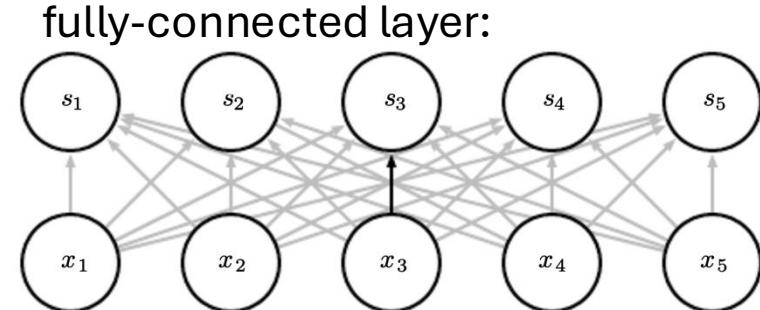
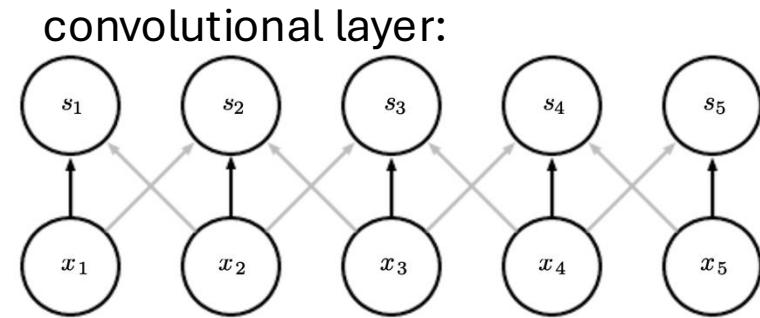
[source](#)

Regularization Effects

- sparse interactions: much less weights
- parameter sharing: use same weights for different connections

effect of receptive field over several layers:

- consider only locally restricted number of input values from previous layer
- grows for earlier layers (indirect interactions)
→ hierarchical patterns from simple building blocks (many aspects of nature hierarchical)



source

Channel Mixing

several input channels c (RGB) and several output channels f (different feature maps):

$$o_{f,x,y} = \sum_{c,s,t} w_{f,c,s,t} i_{c,x+s,y+t}$$

0	0	0	0	0	0	0	...
0	156	155	156	158	158	158	...
0	153	154	157	159	159	159	...
0	149	151	155	158	159	159	...
0	146	146	149	153	158	158	...
0	145	143	143	148	158	158	...
...

Input Channel #1 (Red)

0	0	0	0	0	0	0	...
0	167	166	167	169	169	169	...
0	164	165	168	170	170	170	...
0	160	162	166	169	170	170	...
0	156	156	159	163	168	168	...
0	155	153	153	158	168	168	...
...

Input Channel #2 (Green)

0	0	0	0	0	0	0	...
0	163	162	163	165	165	165	...
0	160	161	164	166	166	166	...
0	156	158	162	165	166	166	...
0	155	155	158	162	167	167	...
0	154	152	152	157	167	167	...
...

Input Channel #3 (Blue)

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

Kernel Channel #1



158

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

Kernel Channel #2



-14

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



653 + 1 = 798

-25	466	466	475	...
295	787	798
...
...

Bias = 1

one feature map

Striding and Padding

need to define how to stride over image

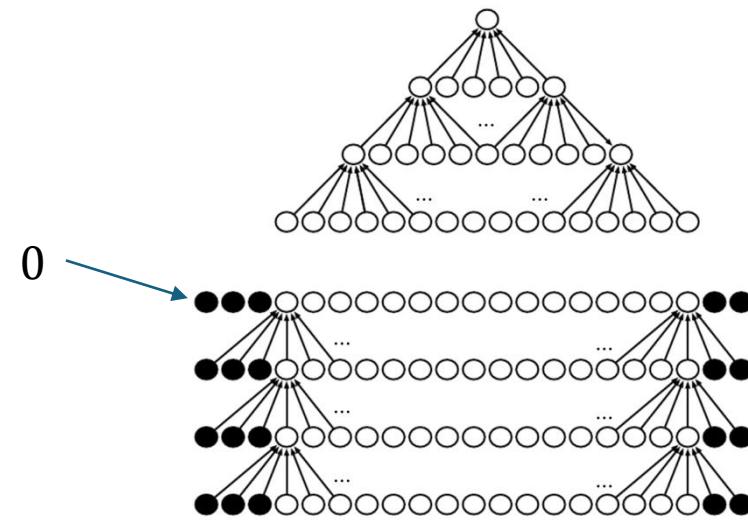
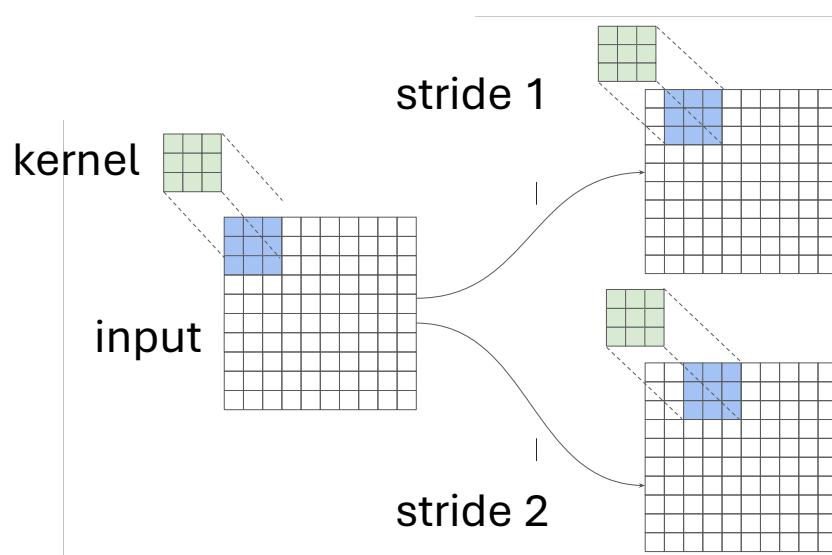
stride > 1 corresponds to down-sampling

→ fewer nodes after convolutional layer

zero-padding of input to make it wider:

otherwise shrinking of representation with each layer (depending on kernel size)

→ needed for large kernels and several layers



[source](#)

Another Ingredient: Pooling

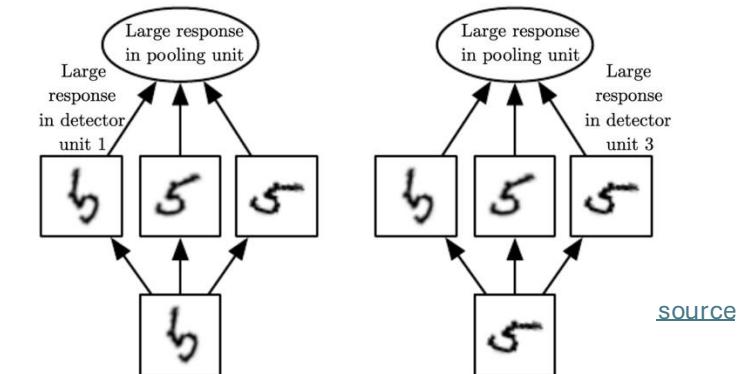
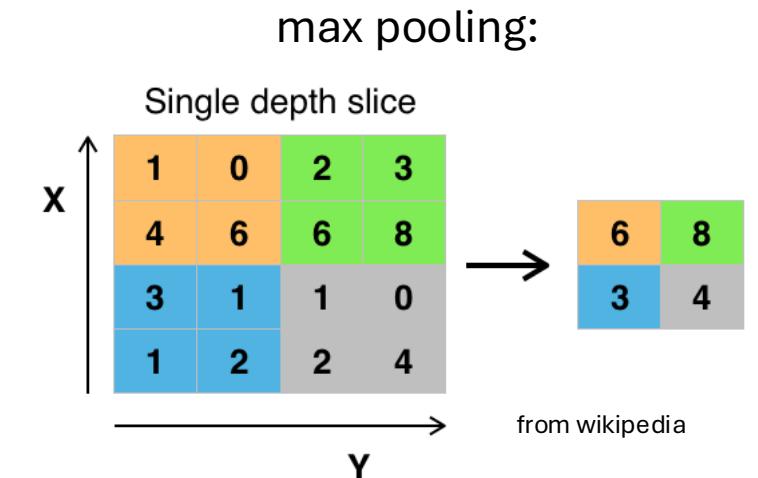
replacing outputs of neighboring nodes with summary statistic (e.g., maximum or average value of nodes)

→ non-linear downsampling (regularization)

pooling is translation invariant: no interest in exact position of, e.g., maximum value

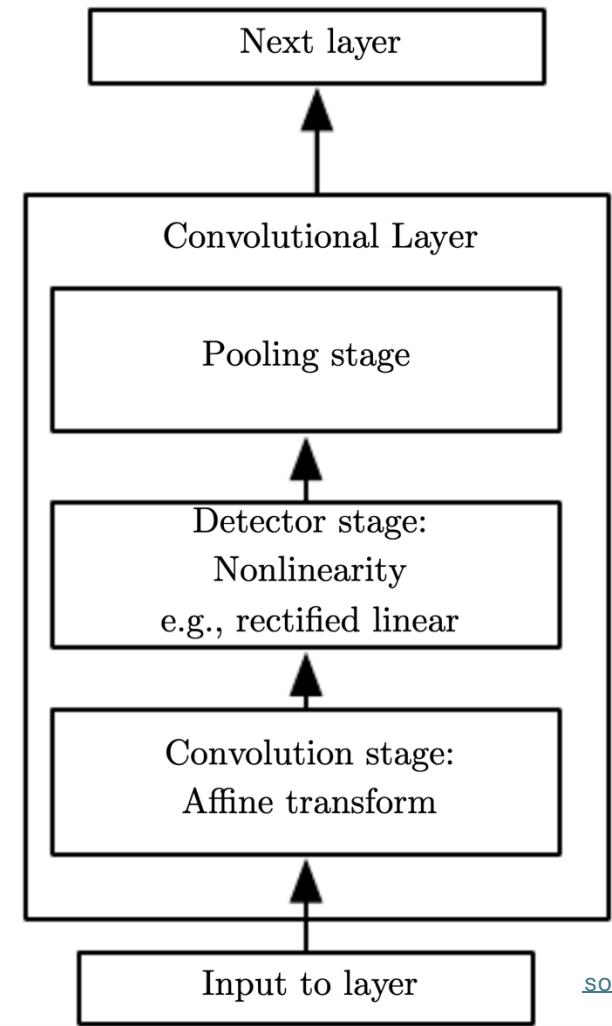
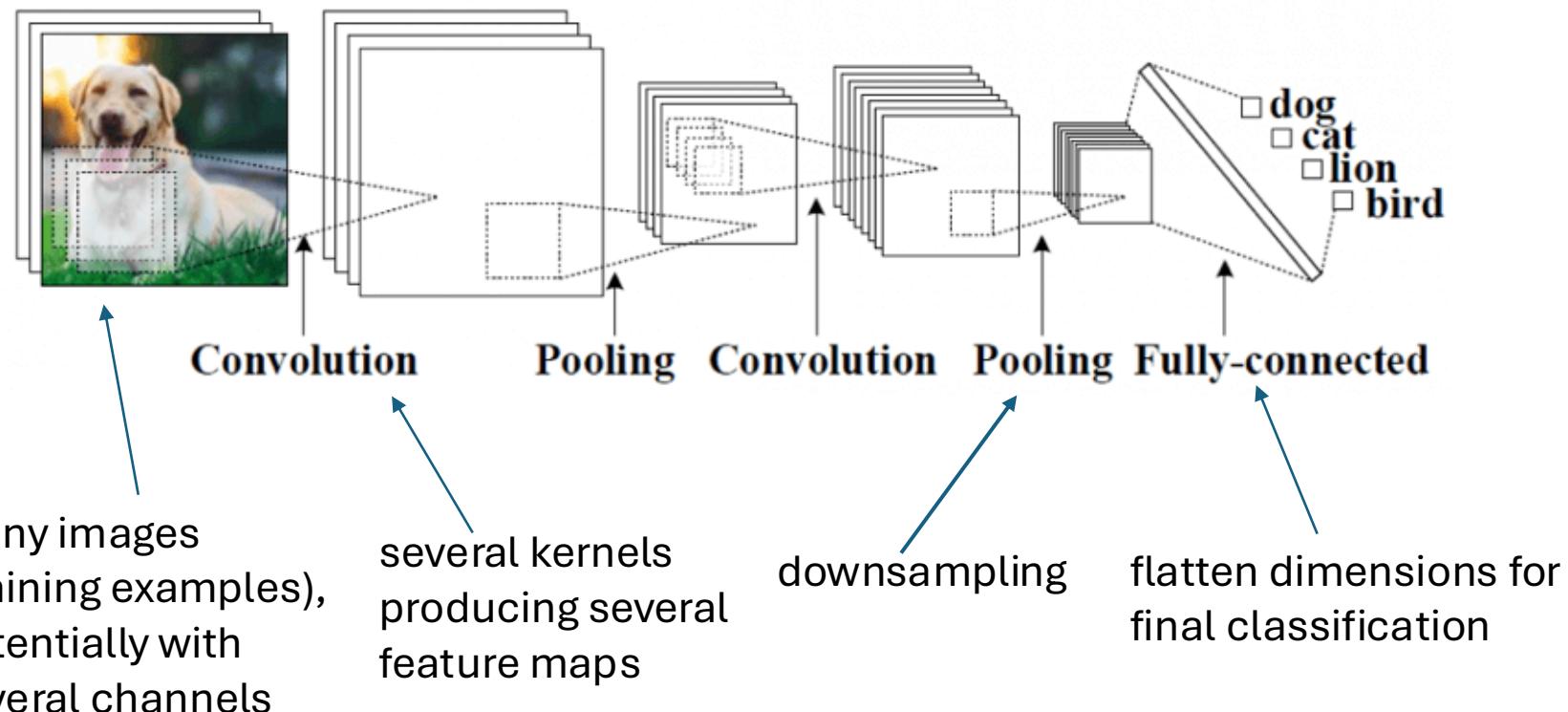
pooling over features learned by separate kernels (cross-channel pooling) can also learn other transformation invariances, like rotation or scale

(convolutions can detect same translated motif across entire image, but not rotated or scaled versions of it)



Putting It All Together

convolutional neural networks in short:
local connections, shared weights, pooling, many layers

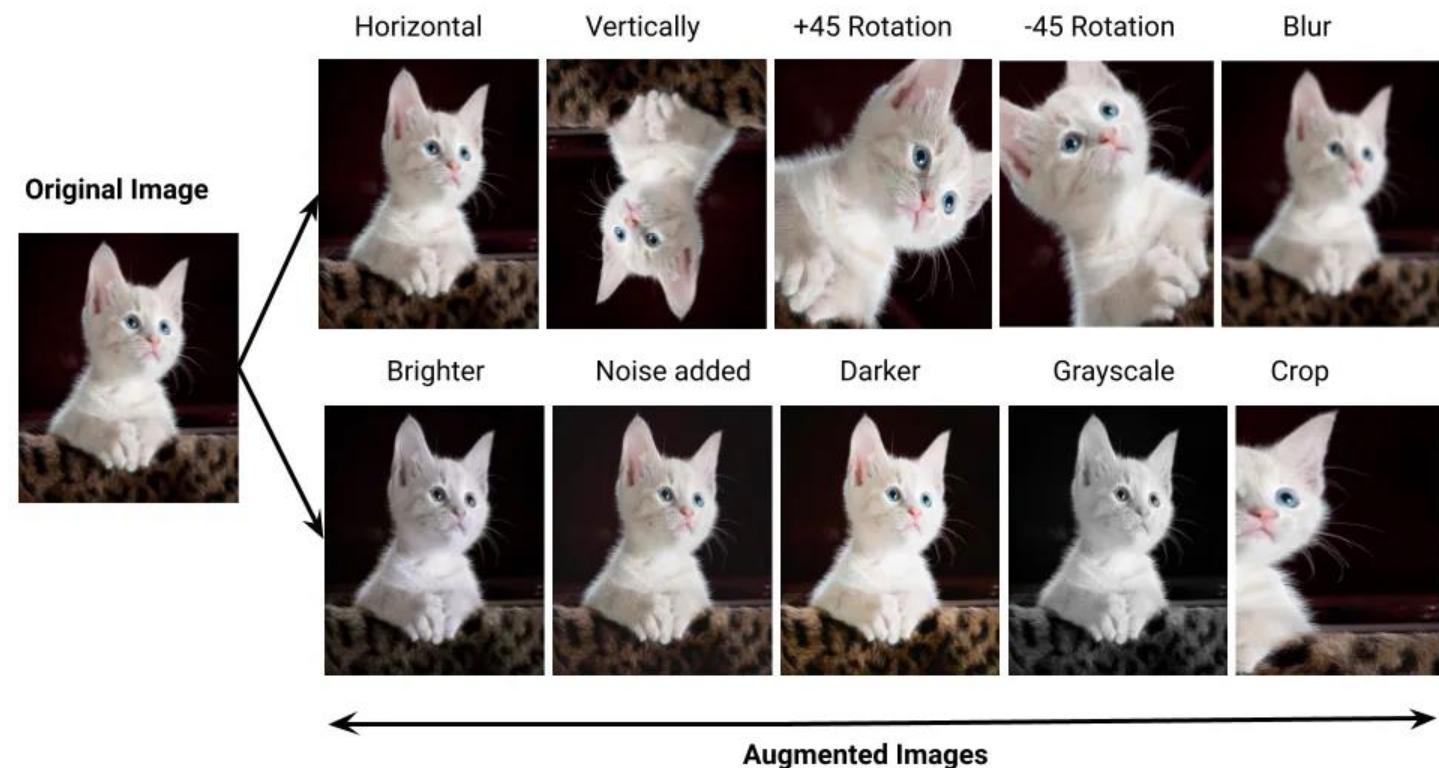


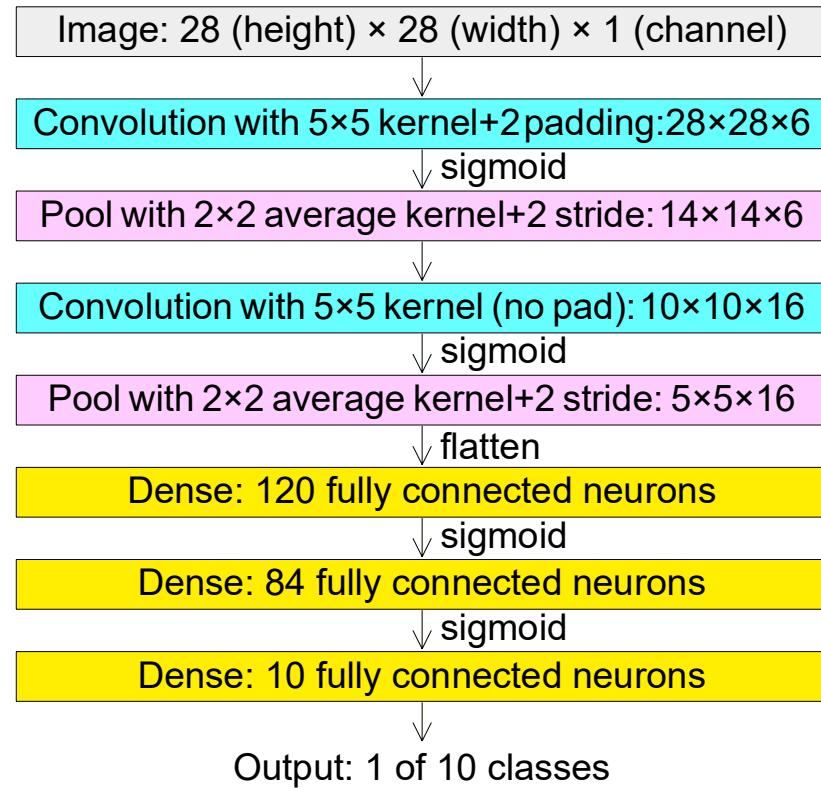
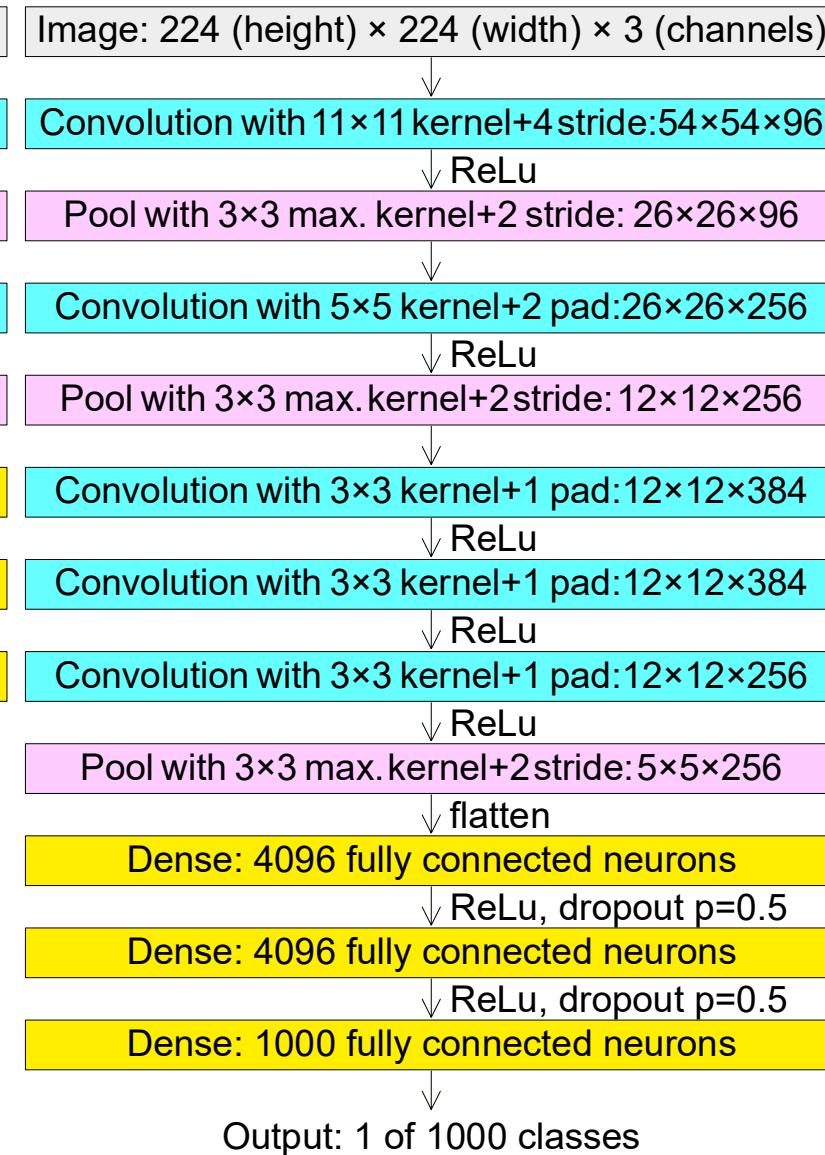
[source](#)

Data Augmentation

adding different variations
of input data to training

idea: supporting the model
in maintaining desired
invariances



LeNet**AlexNet**

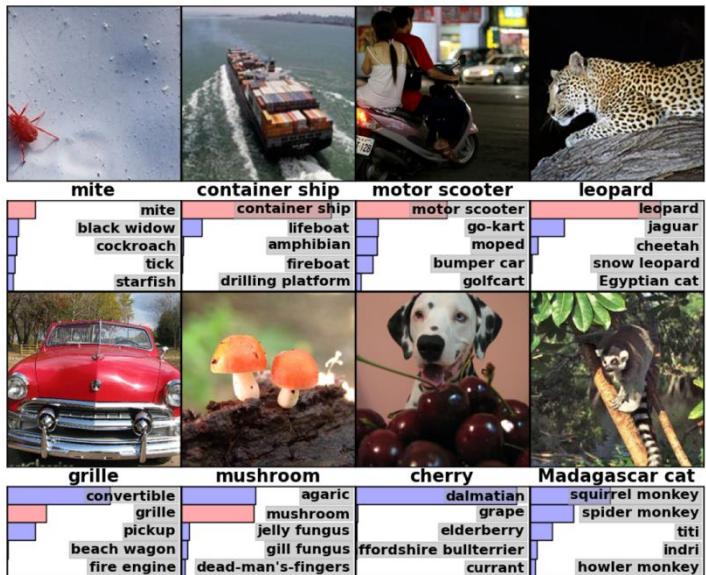
AlexNet finally started the deep learning hype.
(winning the ImageNet challenge in 2012)

Rise of Deep Learning

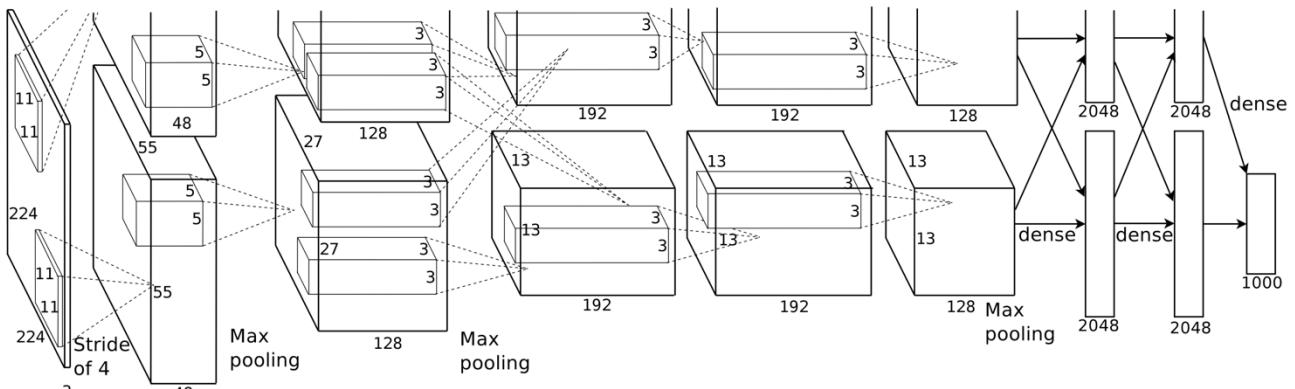
a little bit oversimplified:

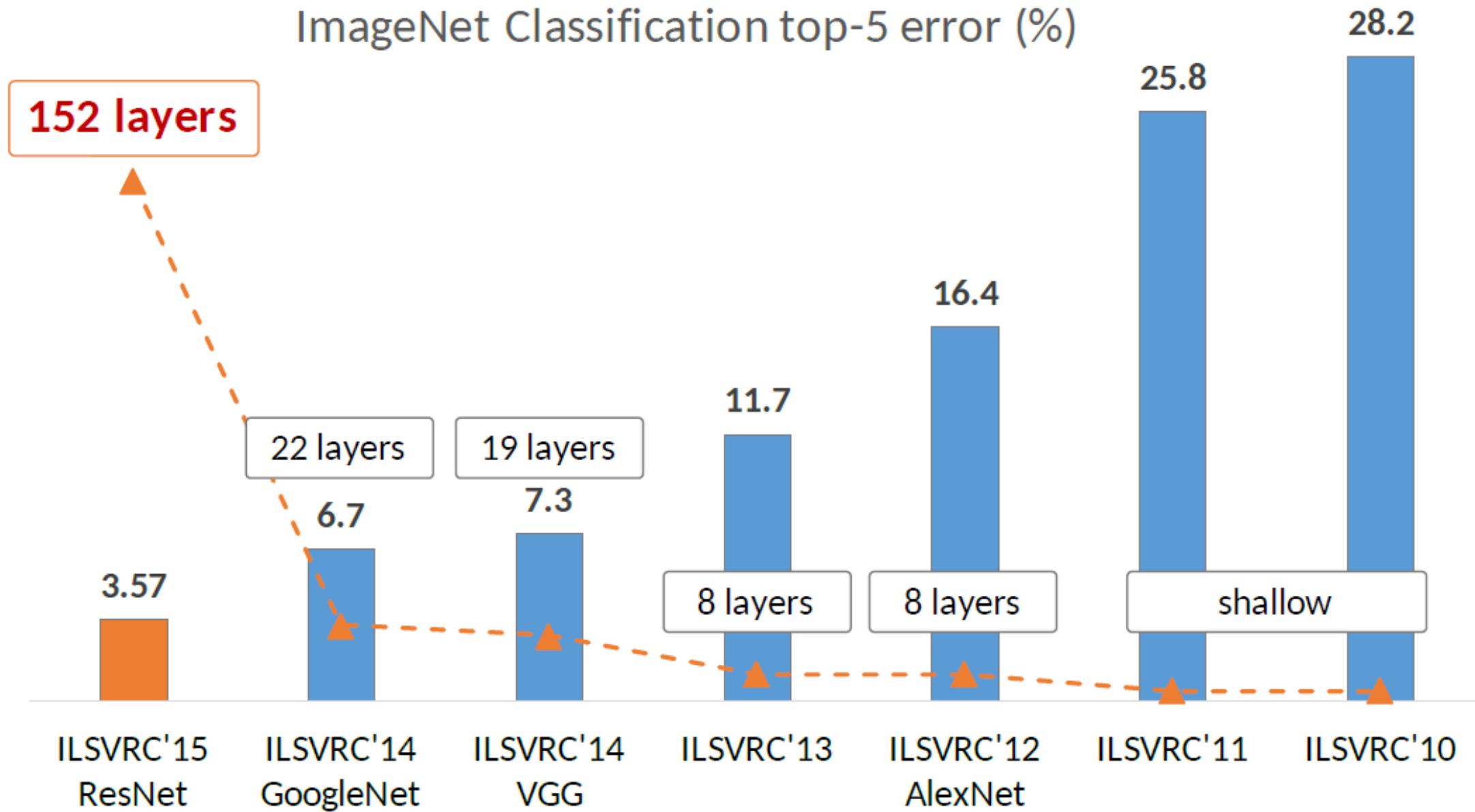
deep learning = lots of training data + parallel computation + smart algorithms

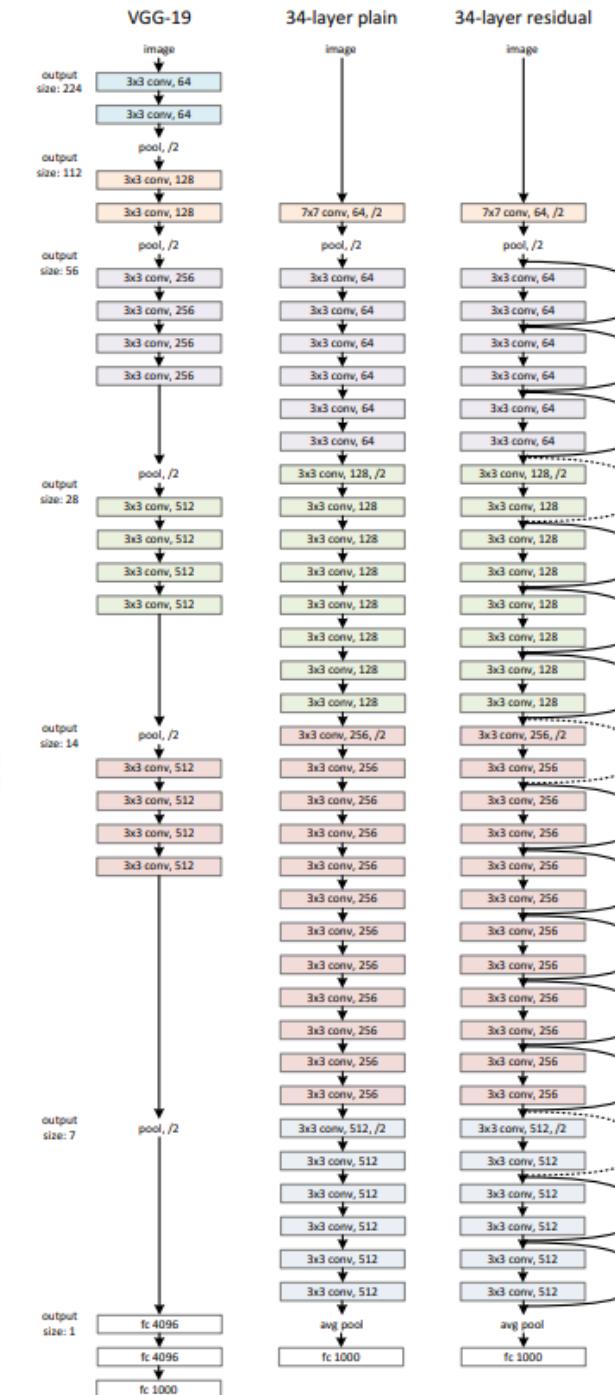
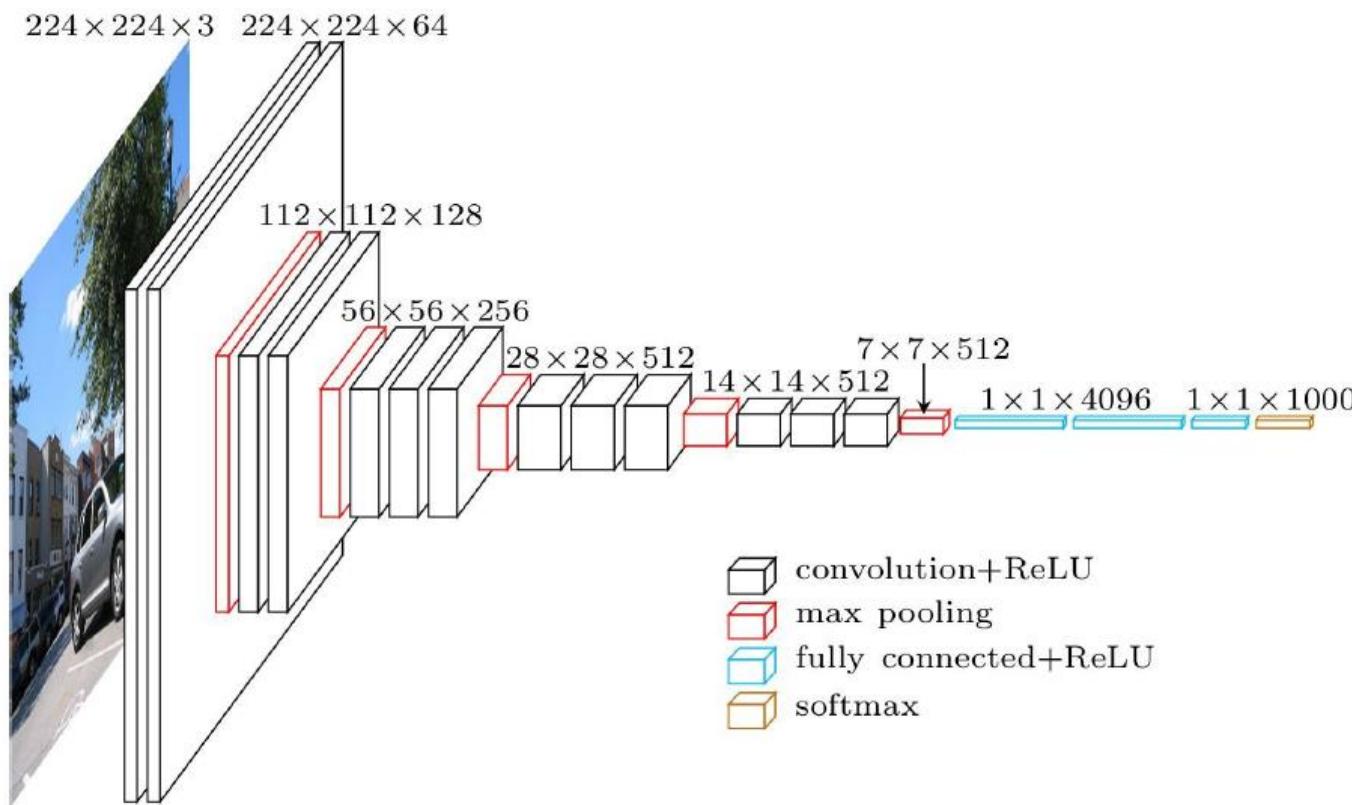
AlexNet: ImageNet (with data augmentation) + GPUs + ReLU, dropout, SGD



[source](#)







ResNet

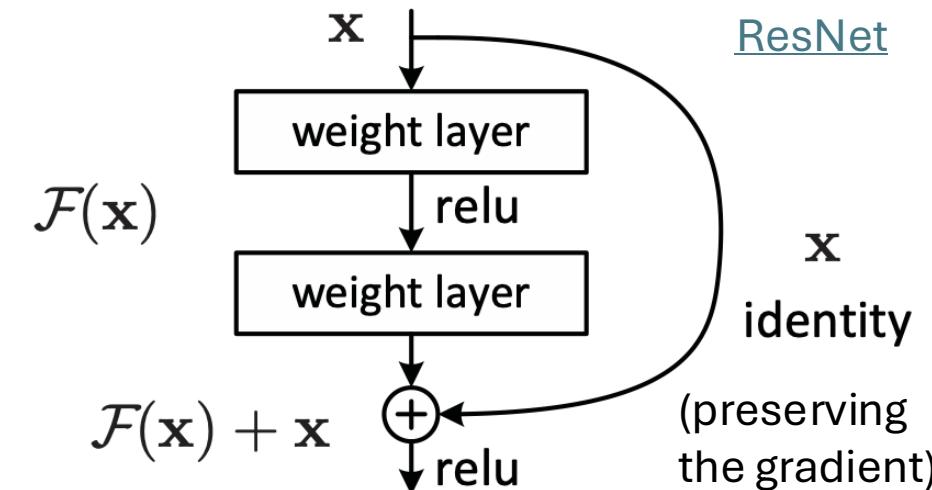
Skip Connections

issue: degradation of training and test errors when adding more and more layers → not due to overfitting (but reason controversial)

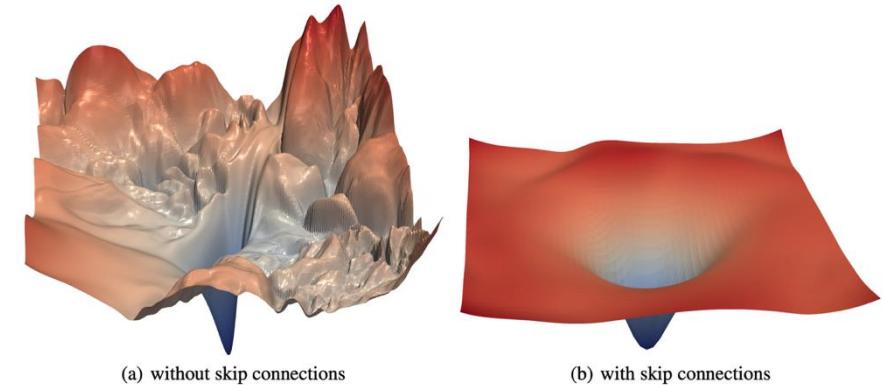
solution: learning of residuals by means of skip connections (resulting in combination of different paths through computational graph)
→ produces loss functions that train easier

together with batch normalization (avoiding exploding gradients), skip connections enable extremely deep networks (>1000 layers) without degradation

residual mapping (special kind of skip/shortcut connections):



loss surface:



[source](#)

Foundation Models

problem with ConvNets trained from scratch (in fact, with all ML methods): only suited for domain of training data

idea of transfer learning:

pretrain a big model (called foundation model) on a broad data set, and then use these learnings for subsequent trainings on specific (typically, narrow) data

two forms of transfer learning: feature extraction and finetuning

Feature Extraction

approach:

- get a pretrained ConvNet as foundation model
- remove final fully-connected layer (its outputs are the class scores from the pretraining task)
- pass training data (for the task at hand) through the network
- for each image, extract vector of last hidden layer's activations (e.g., 4096-dimensional for AlexNet)
- use the extracted values as features to train another model (to be adjusted to the task at hand)

(same idea as used before for SIFT features)

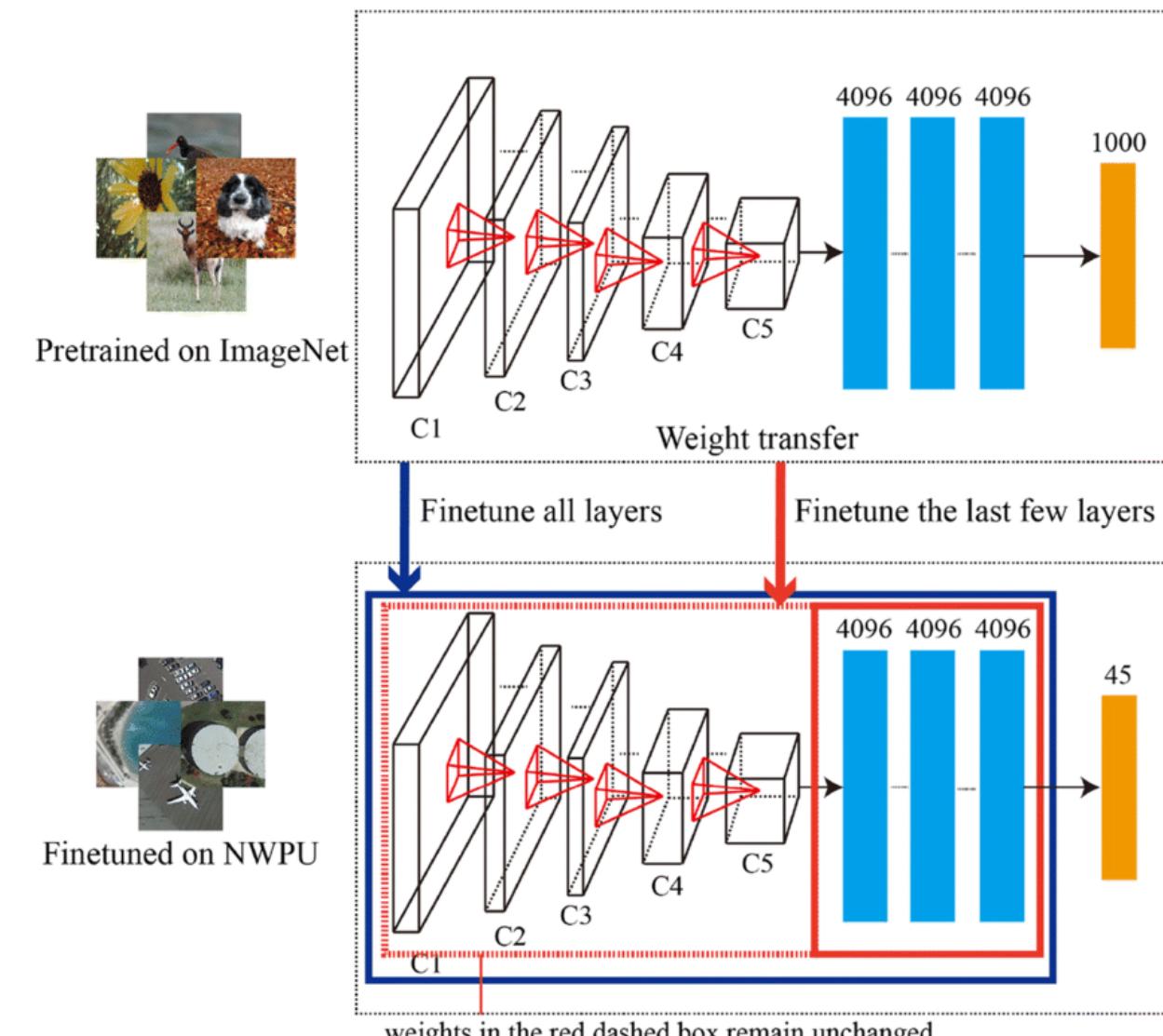
Finetuning

approach:

- get pretrained ConvNet as foundation model
- continue (starting with weights after pretraining) training with data for task at hand

typically, need to change architecture of final layer:
different number of classes

options: finetune all weights or just some



Finetuning Scenarios

finetuning gives better adjustment to new task than feature extraction
(important if new task differs a lot from pretraining task)

→ coming along with danger of overfitting

ConvNet features more generic in early layers (e.g., edges) and more specific to the data set of pretraining in later layers

→ for small finetuning data sets, typically keep weights of early layers fixed to avoid overfitting

for large finetuning data sets (less danger of overfitting), finetuning all layers can be beneficial

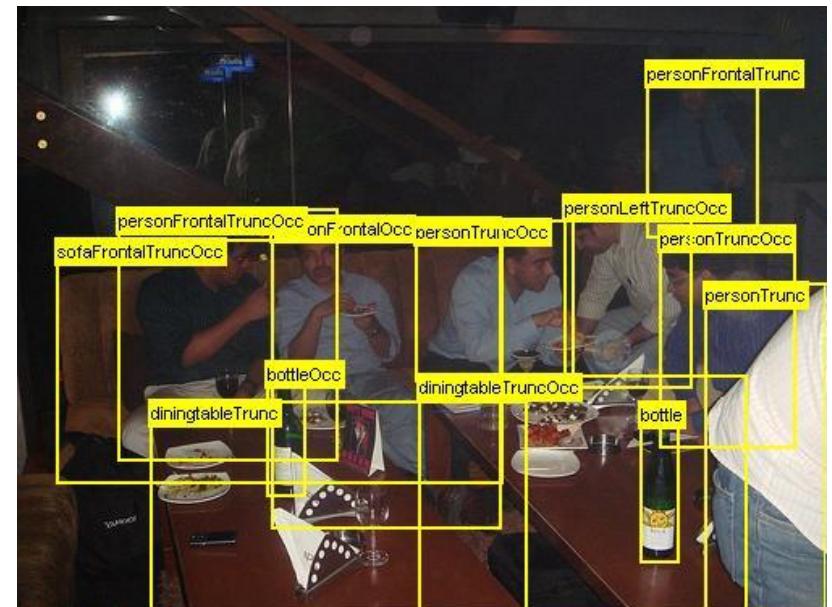
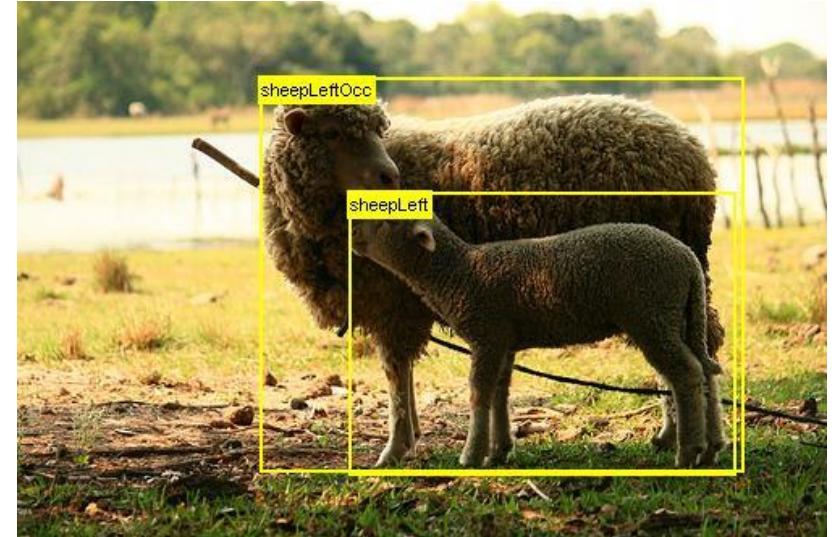
Different Image Classification Techniques

- Hough transform: looking for predefined shapes such as lines or circles (feature-based recognition)
- feature description & matching: identify specific object instances (typically used for instance rather than class recognition)
- image features as input to classic ML method: generalization from data
- plain feed-forward network on raw image data: even more generalization without handcrafted components
- convolutional neural networks: use of spatial structure (inductive bias)
- pretraining & finetuning (or feature extraction): transfer learning

Semantic Segmentation

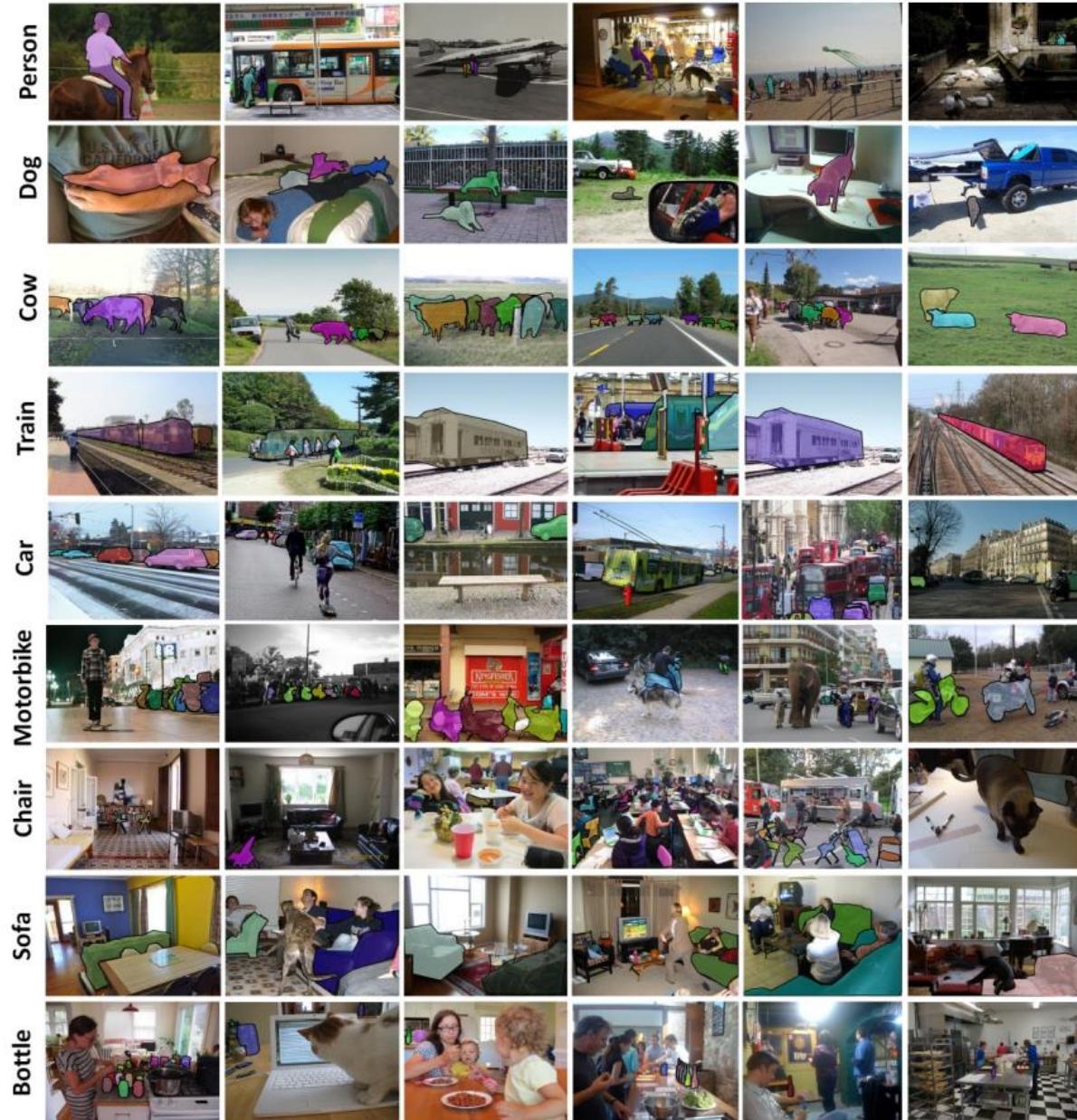
PASCAL VOC Data Set

- PASCAL Visual Object Class challenge
- widely used as benchmark for object detection and semantic segmentation
- 20 object categories such as person, sofa, sheep, car, ...
- 11530 annotated images
- available annotations: pixel-level segmentation, bounding boxes, object classes



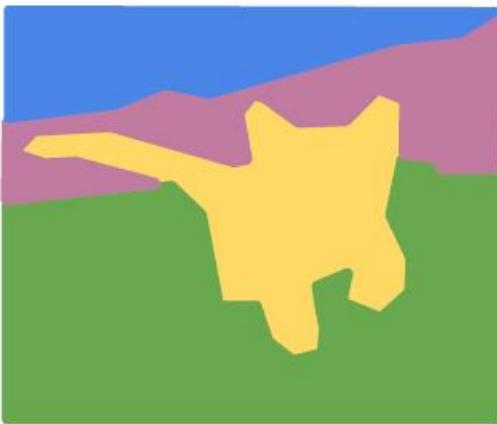
MS COCO Data Set

- Microsoft Common Objects in Context
- images of complex everyday scenes containing common objects in their natural context
- 91 objects types
- 2.5 million annotated instances in 328k images → instance segmentation



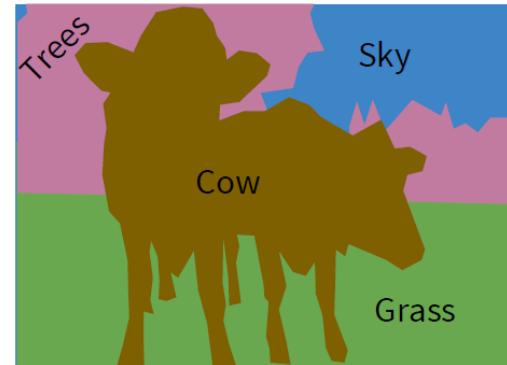
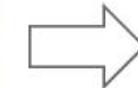
Classification of Each Pixel

segmentation: no objects, just pixels



GRASS, CAT, TREE,
SKY, ...

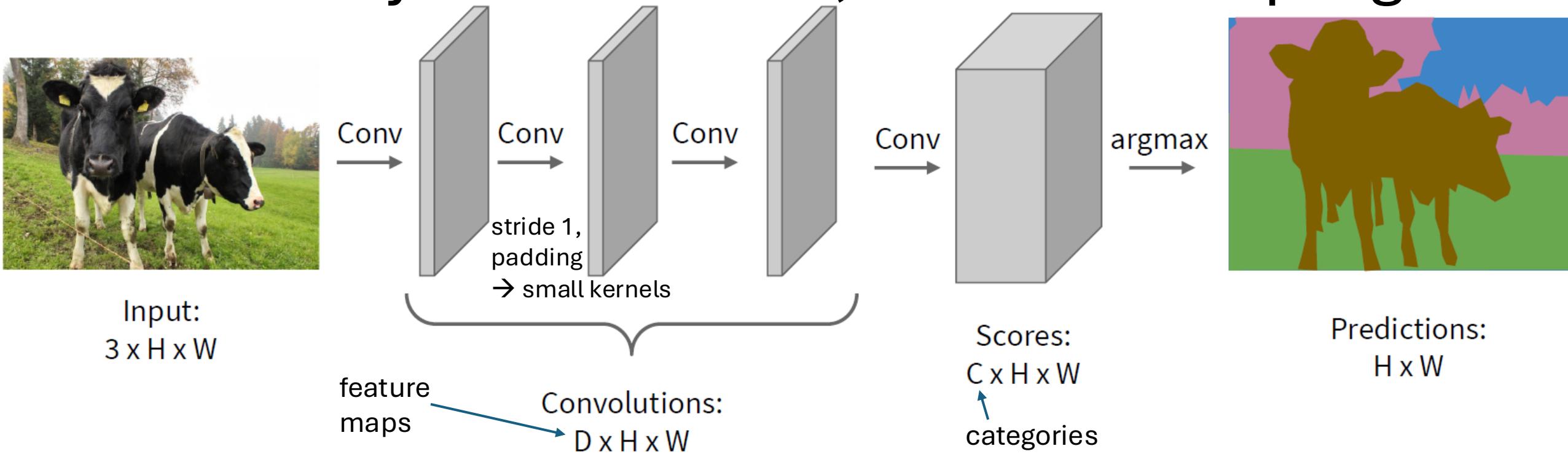
Paired training data: for each training image,
each pixel is labeled with a semantic category.



At test time, classify each pixel of a new image.

minimize sum over classification losses
(cross-entropy loss at every output pixel)

Idea: Fully Convolutional, No Downsampling



replace flattened, fully-connected classification layers with 1×1 convolutions
→ maintain spatial relationships and enable pixel-wise classification:
conversion of feature maps into classification heat maps (one for each class)

but no downsampling means small receptive field and no hierarchical learning

Upsampling to the Rescue

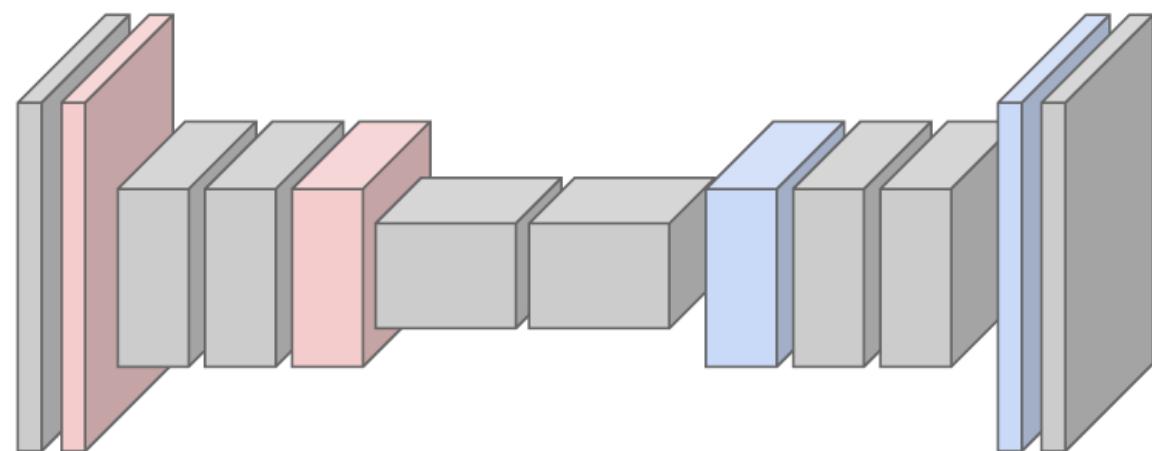
typical spatial feature extraction: convolution and pooling layers → downsampling



Input:
 $3 \times H \times W$

upsampling layers to restore original image size

pixel-wise classification



$C \times H \times W$

Predictions:
 $H \times W$



different options for down- (pooling, strided convolution) and upsampling ...

Reverse Pooling

resampling (no learned parameters)

Nearest Neighbor

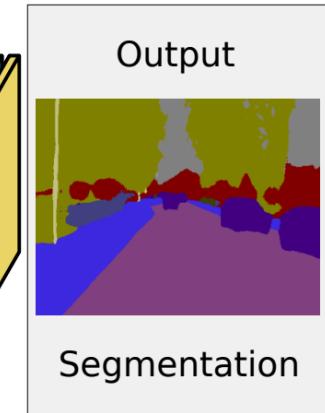
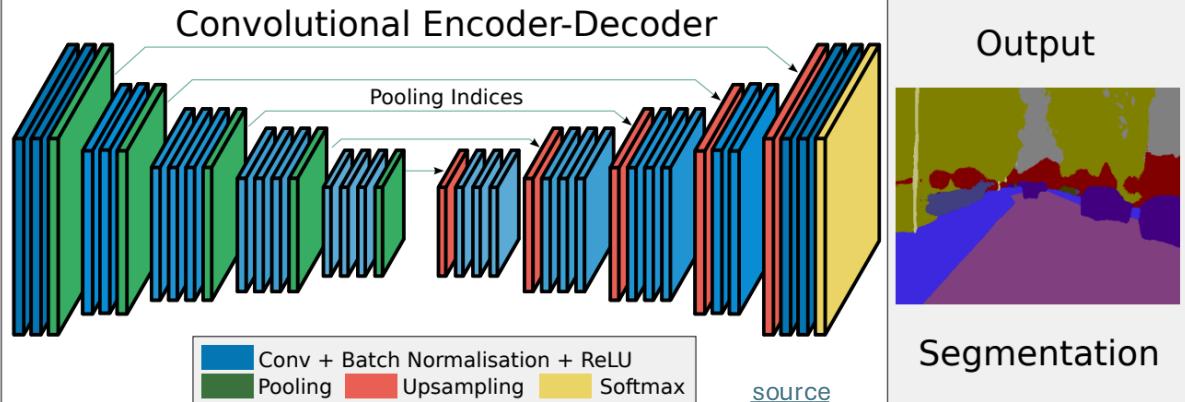
1	2
1	2
3	4

Input: 2 x 2

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Output: 4 x 4

unpooling (recording max positions from pooling)



Max Pooling
Remember which element was max!

1	2	6	3
3	5	2	1
1	2	2	1
7	3	4	8

Input: 4 x 4

5	6
7	8

Output: 2 x 2

Rest of the network

Max Unpooling
Use positions from
pooling layer

1	2
3	4

Input: 2 x 2

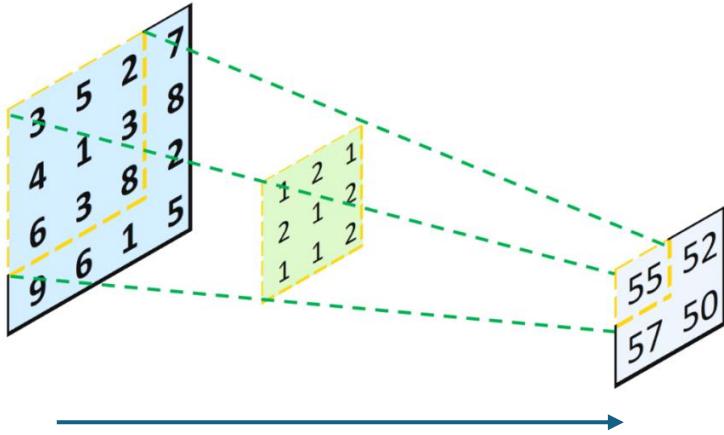
0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

Output: 4 x 4

filled with learned parameters in subsequent convolution

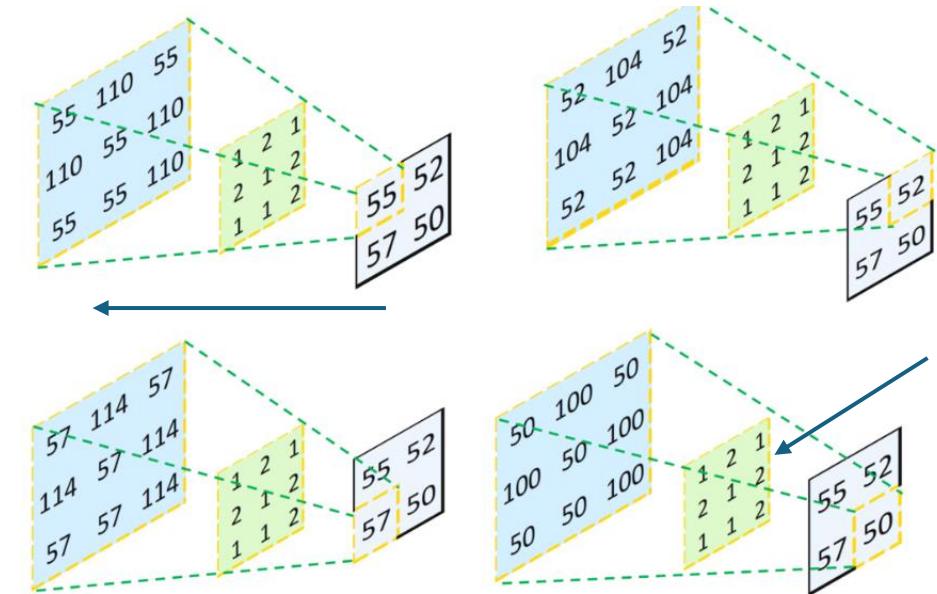
Reverse Convolution

convolution

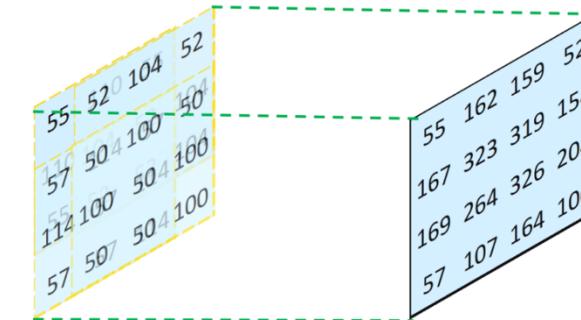


not inverse convolution
→ additional learned parameters

transposed convolution



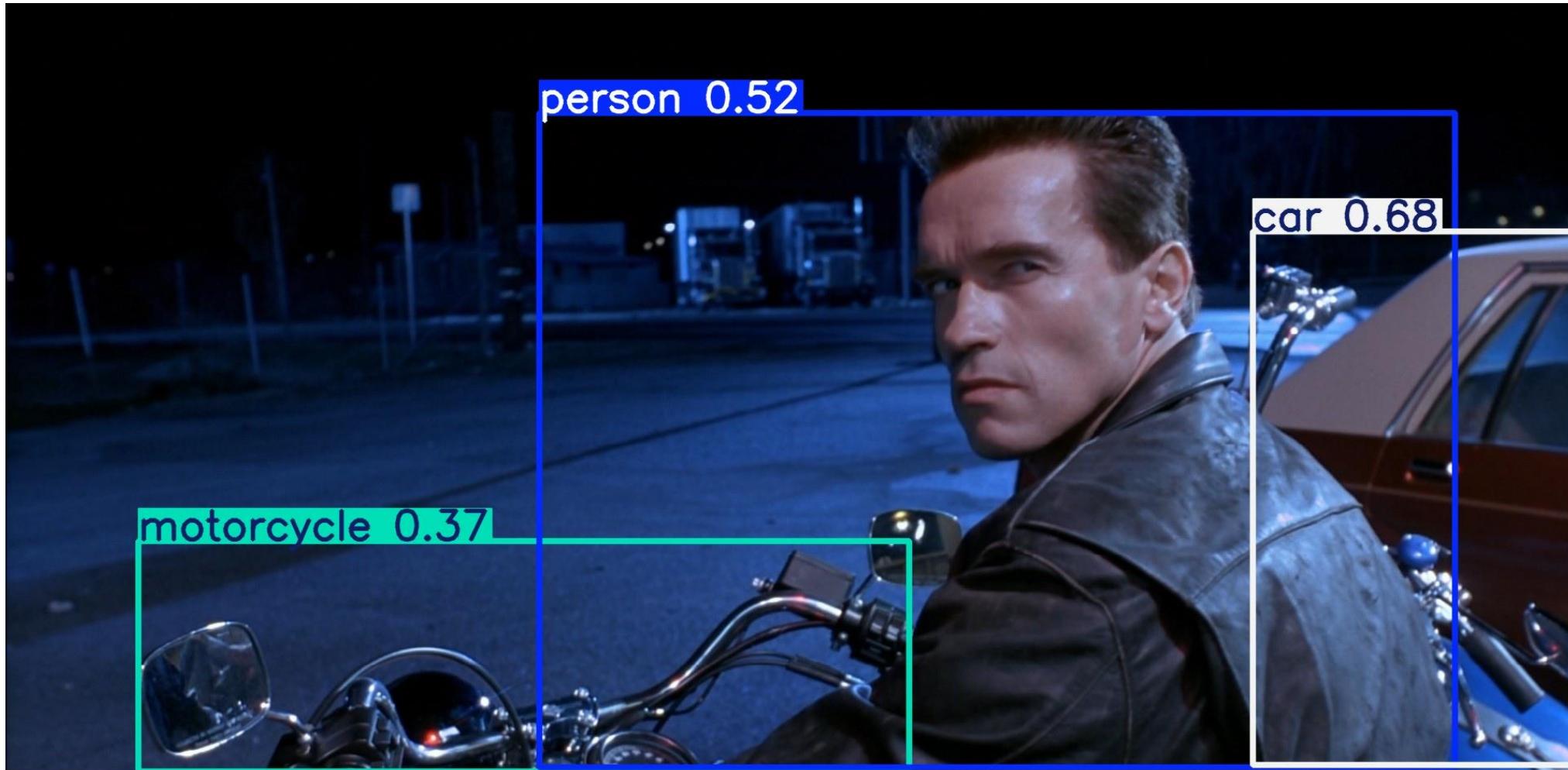
combine and sum overlaps:



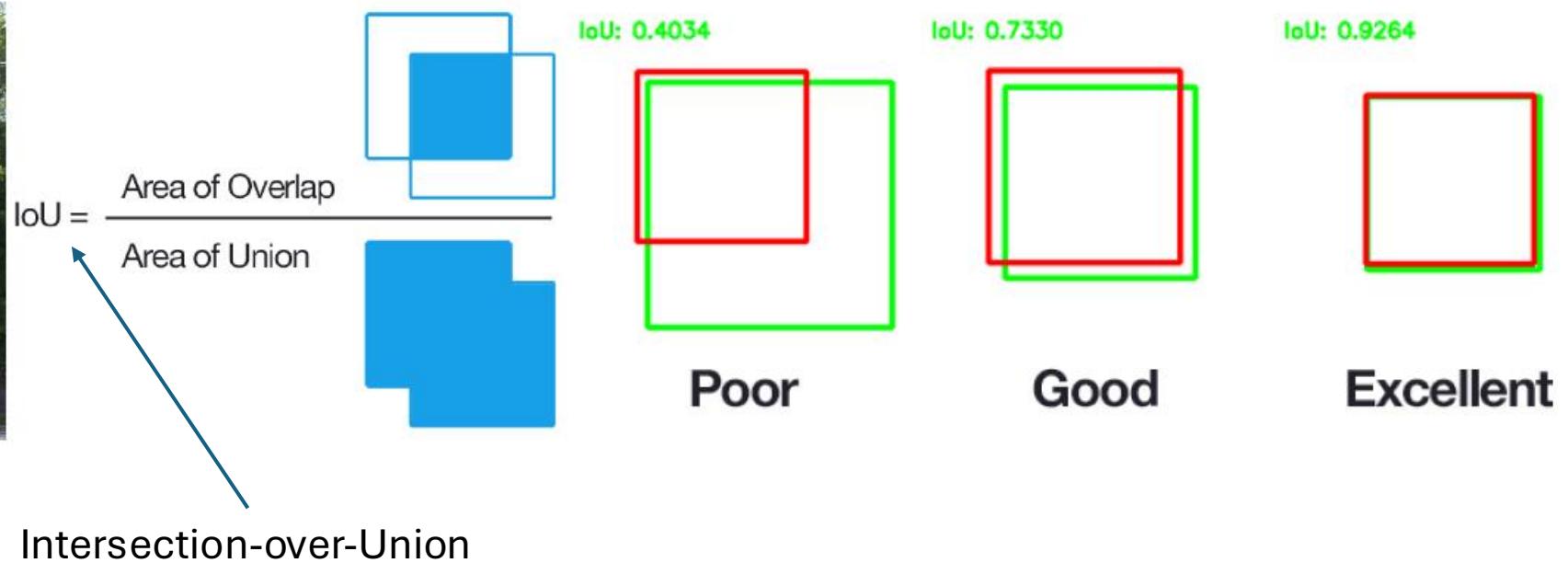
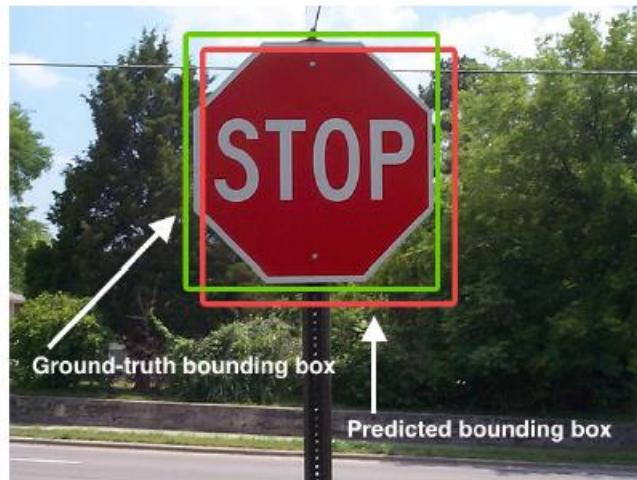
source

Object Detection

output: bounding boxes with category label and confidence



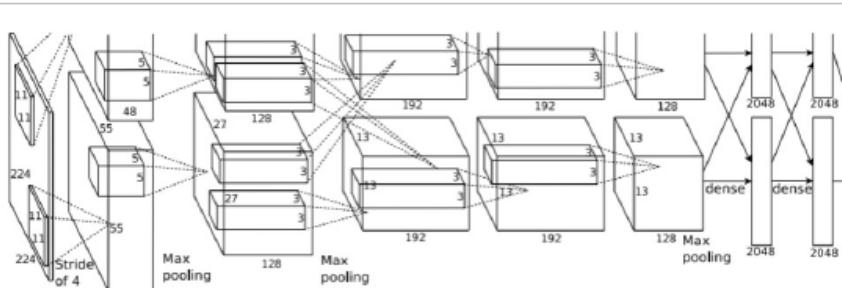
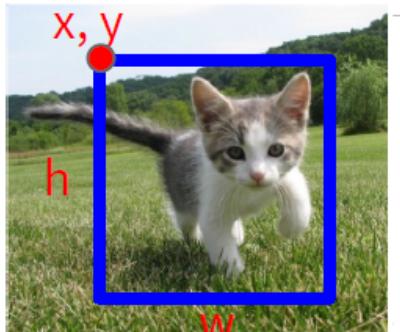
Performance Evaluation of Localization



images with multiple objects: number of detections dependent on used confidence threshold

Object Detection: Single Object

(Classification + Localization)



Treat localization as a regression problem!

too complicated for multiple objects

Correct label:
Cat

Localization for Multiple-Object Images

idea: classify many different crops of the image as object or background
(crops: sliding window over image, iterated at multiple window sizes)



Dog? No
Cat? No
Background? Yes



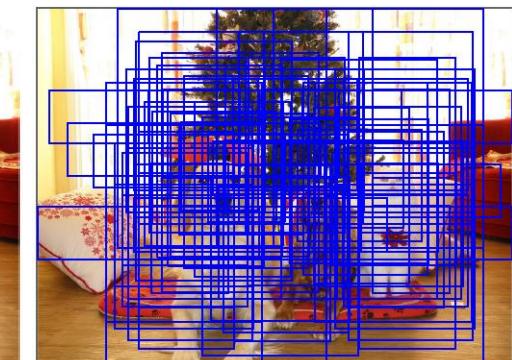
Dog? Yes
Cat? No
Background? No



Dog? Yes
Cat? No
Background? No



Dog? No
Cat? Yes
Background? No



issue: too many possible crops

→ need for region proposals

Region-Based Convolutional Neural Network (R-CNN)

Per-image computation

Per-region computation for each $r_i \in r(I)$

two stages

task-specific heads:

object classification

Linear
classifier

4

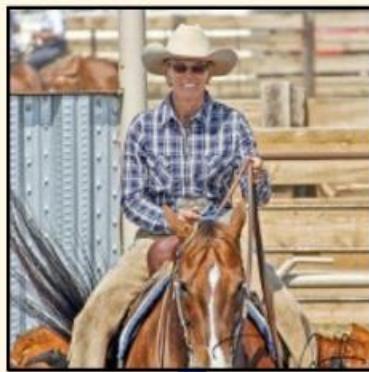
Box regressor

5

feature representation
(forward-pass through
pretrained CNN)

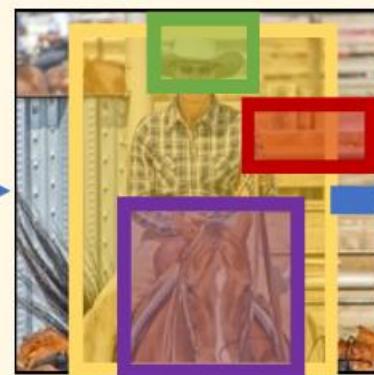
bounding-box refinement

$I:$



Selective search,
Edge Boxes,
MCG, ...

1



Crop &
warp

2



ConvNet(r_i)

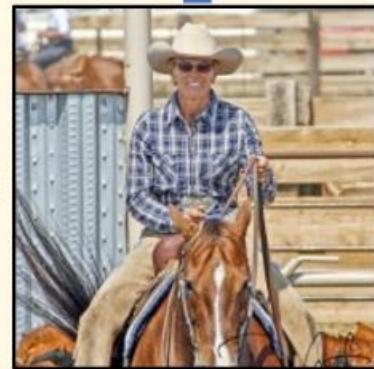
3



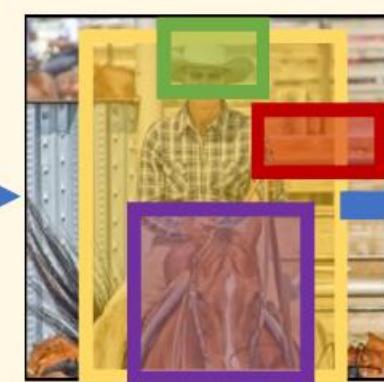
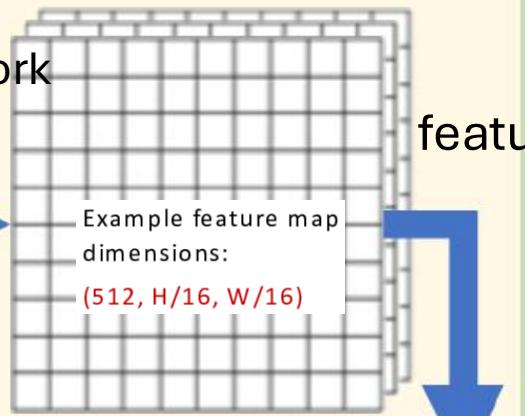
region proposals by “classic” method (~2k)

Fast R-CNN: Crop ConvNet Features

Per-image computation
Fully Convolutional Network
(pretrained, e.g., VGG-16)



Selective search,
Edge Boxes,
MCG, ...



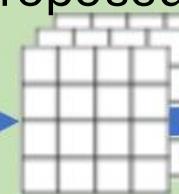
Per-region computation for each $r_i \in r(I)$

feature sharing

fixed-dimensional
representation for
each proposed region

2

RoIPool



MLP

or CNN

more lightweight

Region of Interest pooling:

- map proposed RoI coordinates to feature map
- divide RoI on feature map into fixed bins (e.g., 7x7 grid)
- max pooling in each bin

end-to-end trainable

Softmax clf.

3

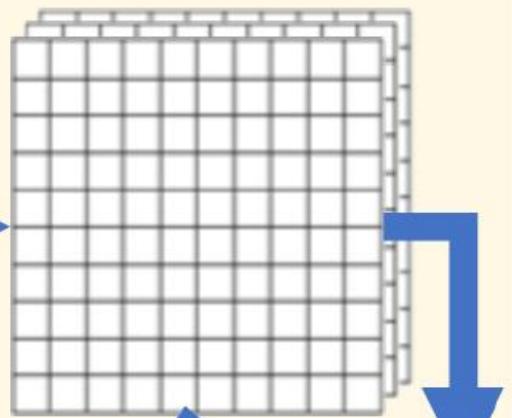
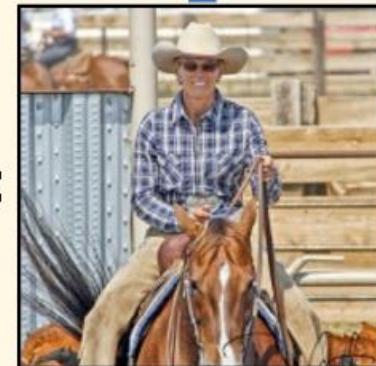
Box regressor

4

Faster R-CNN: Use Region Proposal Network

Per-image computation

$$f_I = \text{FCN}(I)$$

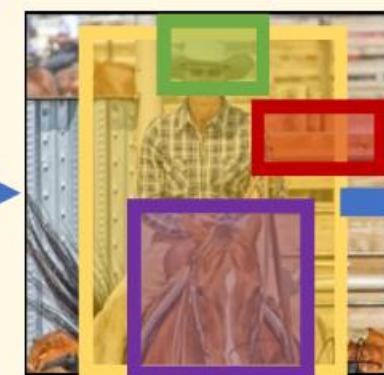


Per-region computation for each $r_i \in r(I)$

two RPN losses combined with two detection head losses
during end-to-end training

I :

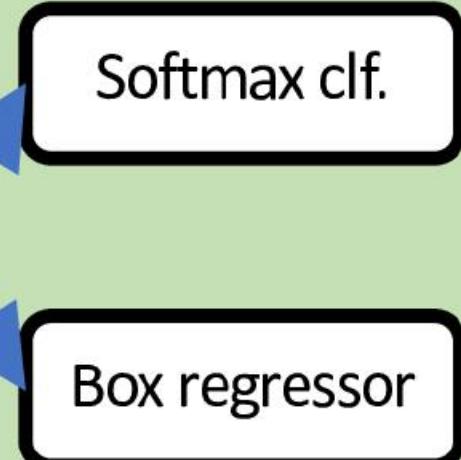
$$\text{RPN}(f_I)$$



RoIPool



MLP



Softmax clf.

Box regressor

Learned proposals

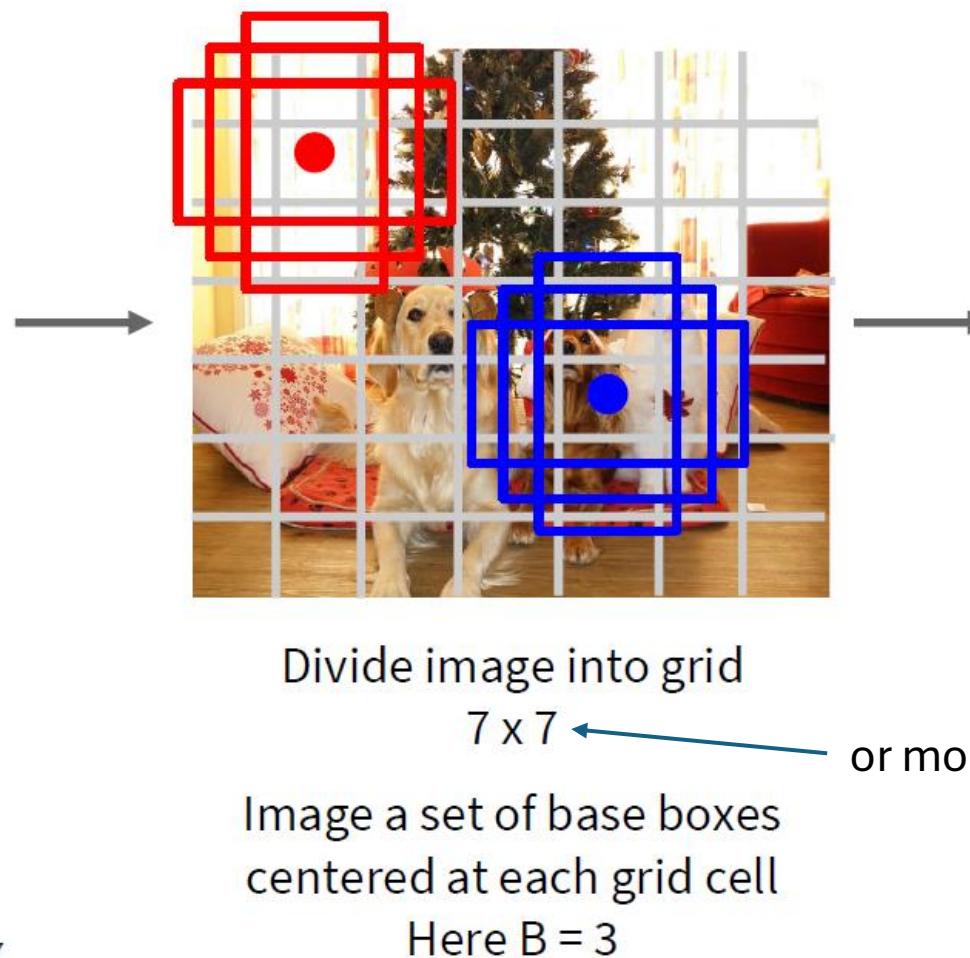
Shares computation with whole-image network

binary classification (object or not) with sliding window (e.g., 3x3) over feature map, refining predefined anchor boxes

Single-Stage Detectors: Drop Per-Region Computation



Input image
 $3 \times H \times W$



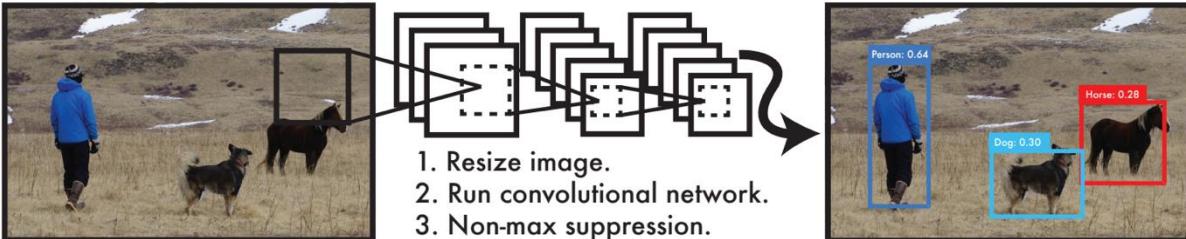
- Within each grid cell:
- Regress from each of the B base boxes to a final box with 5 numbers:
(dx , dy , dh , dw , confidence)
 - Predict scores for each of C classes (including background as a class)
 - Looks a lot like RPN, but category-specific!

Output:
 $7 \times 7 \times (5 * B + C)$

faster, but usually worse performance

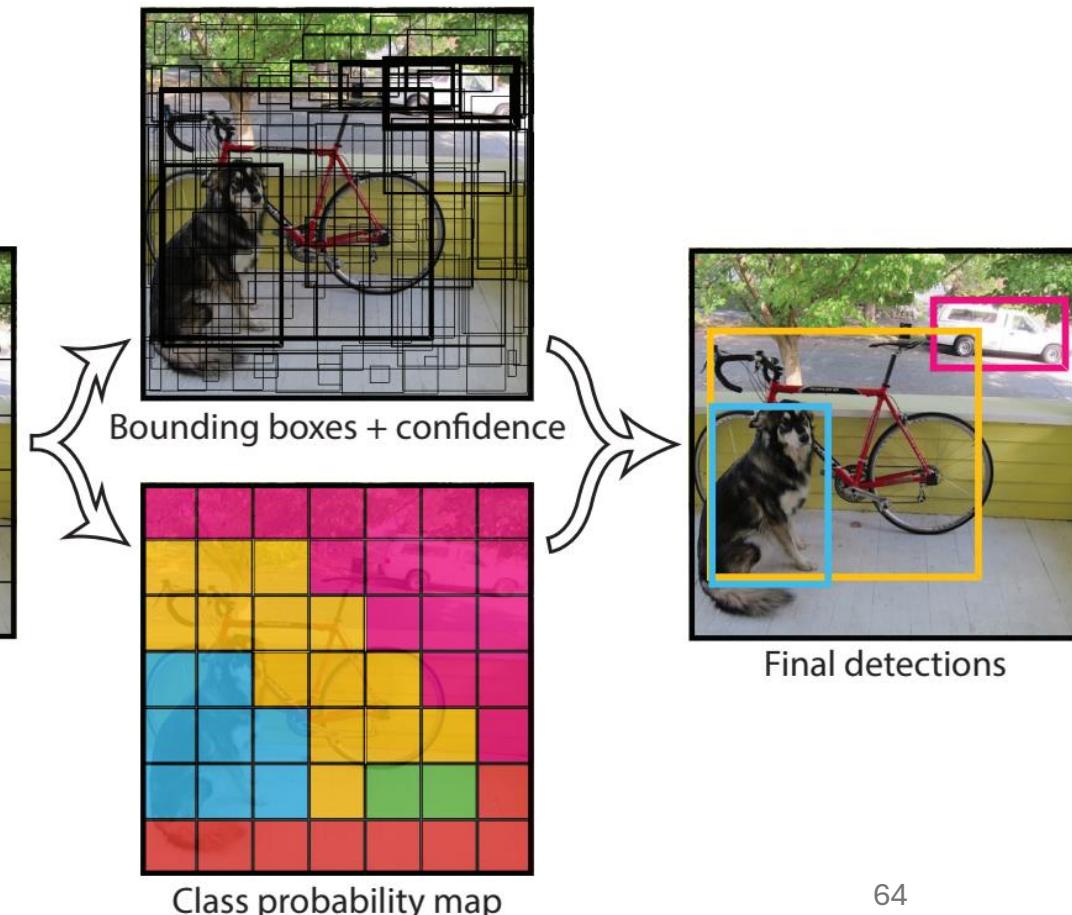
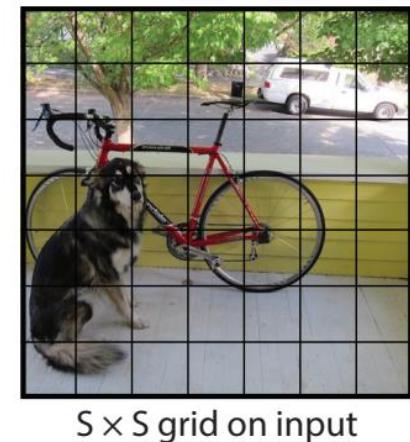
Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016
Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

YOLO: Real-Time Object Detection



anchor-free approach
in newer YOLO versions

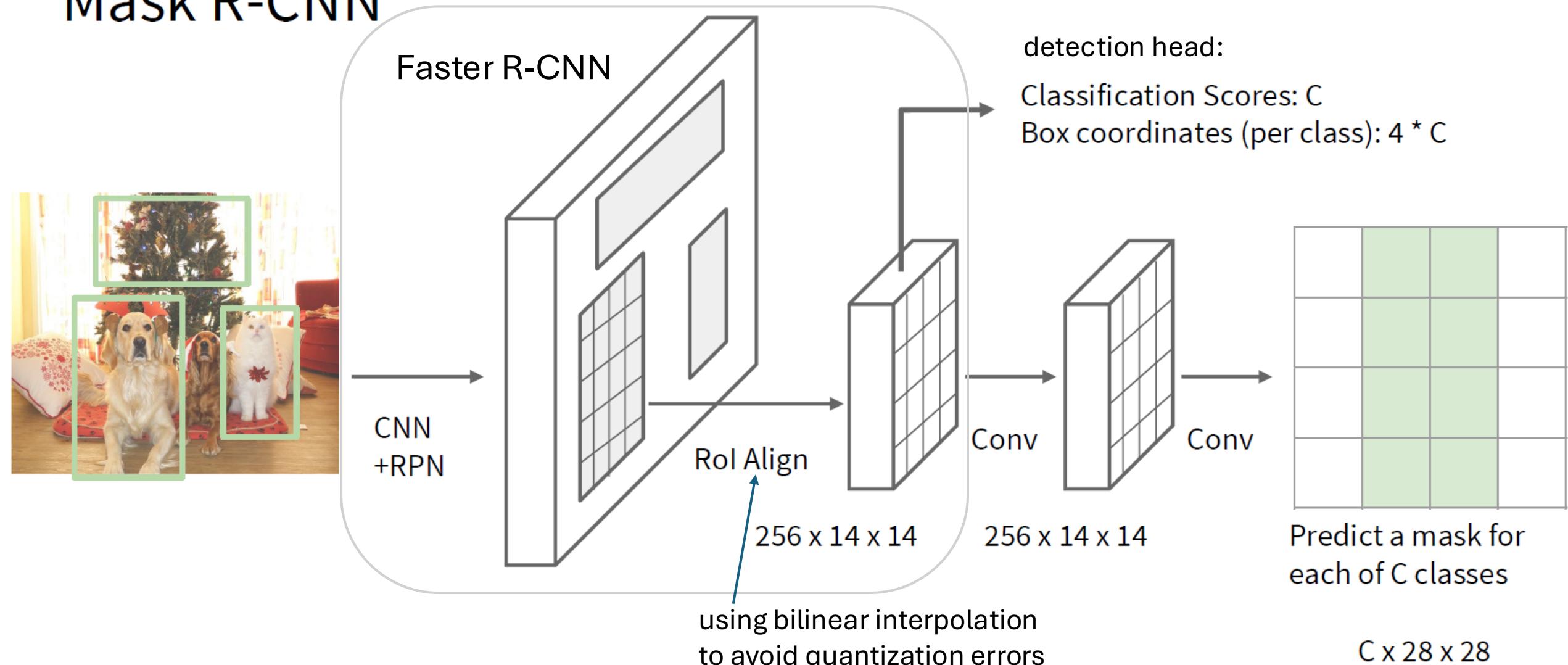
You Only Look Once:
prediction of bounding boxes
and class probabilities in one go



Instance Segmentation

Mask R-CNN

→ total loss is sum of RPN, detection-head, and mask-head losses



1. object detection (boundary boxes)
2. prediction of separate binary masks for each detected object (specific instances)⁶⁶

image with training proposal



28×28 mask target

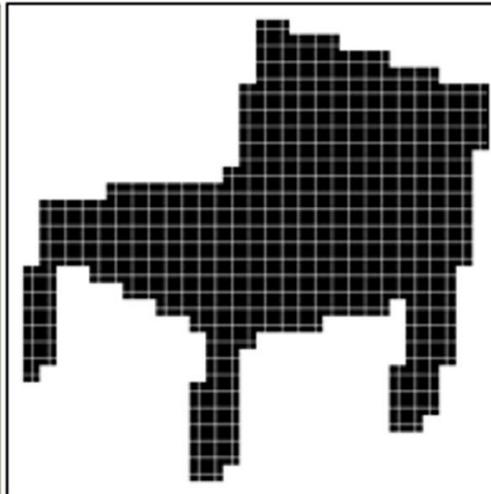
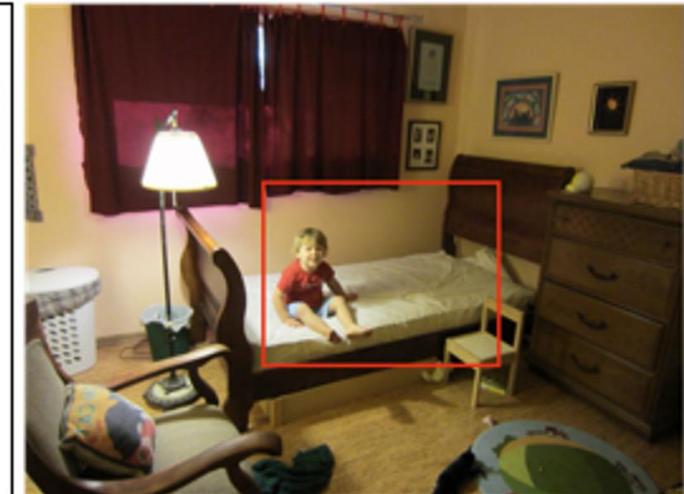
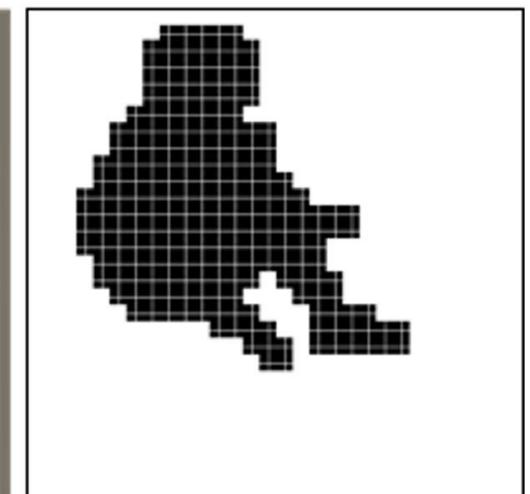
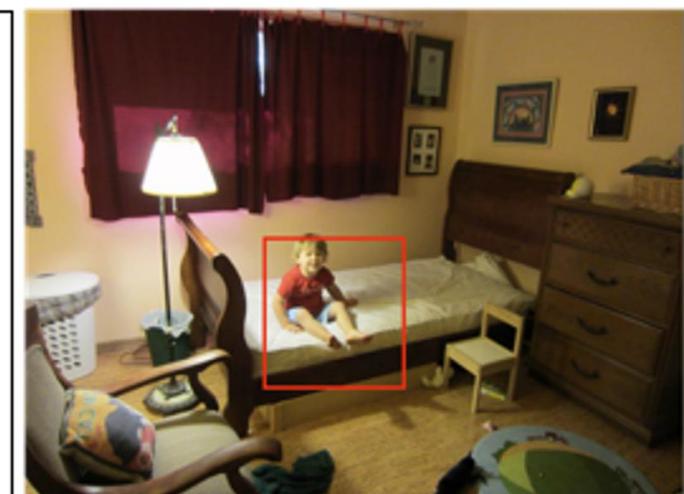
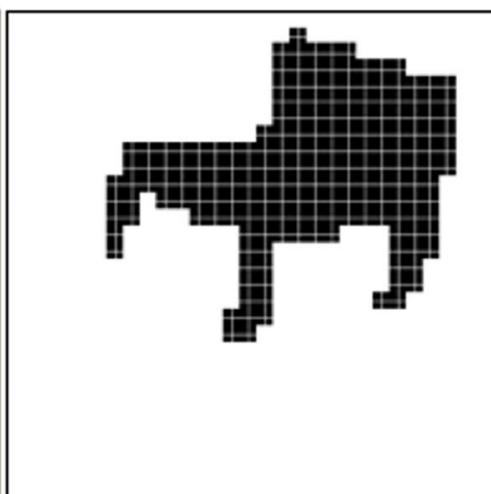
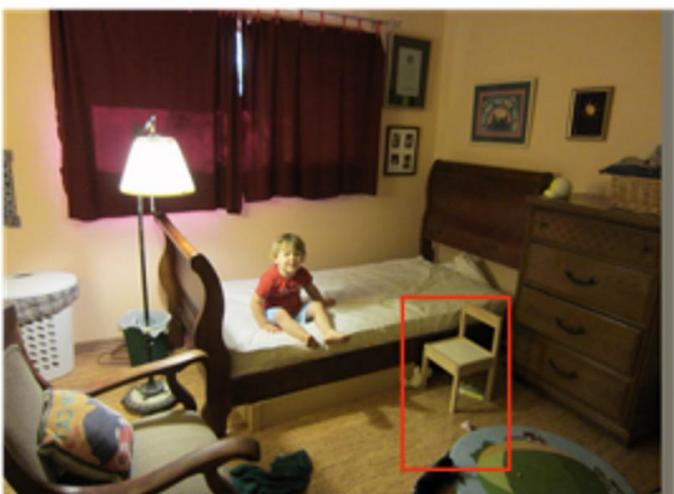
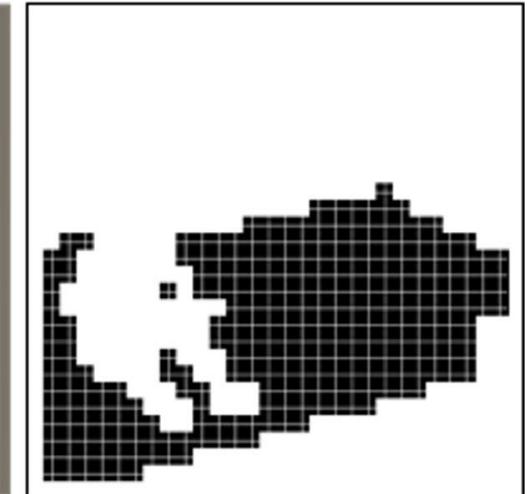


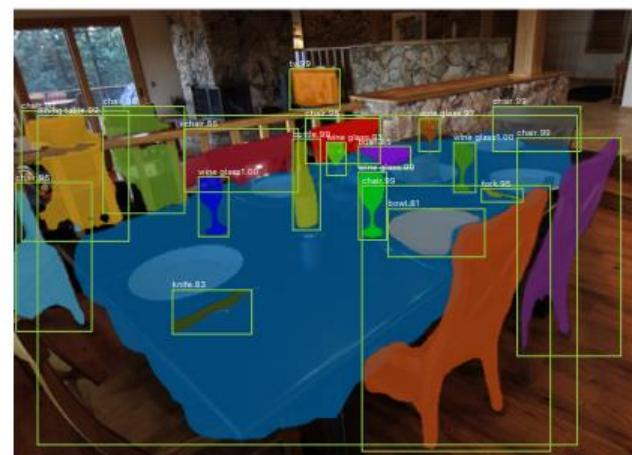
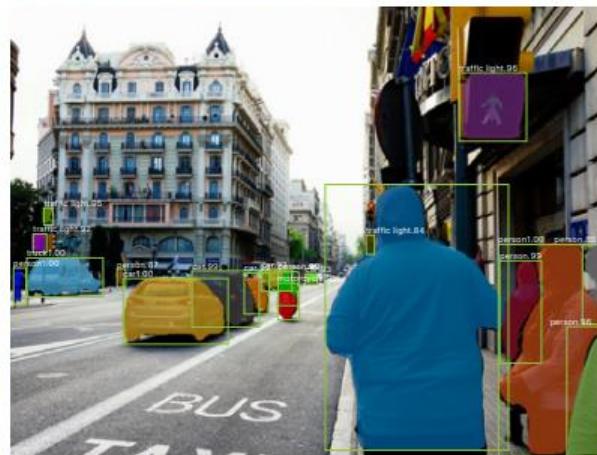
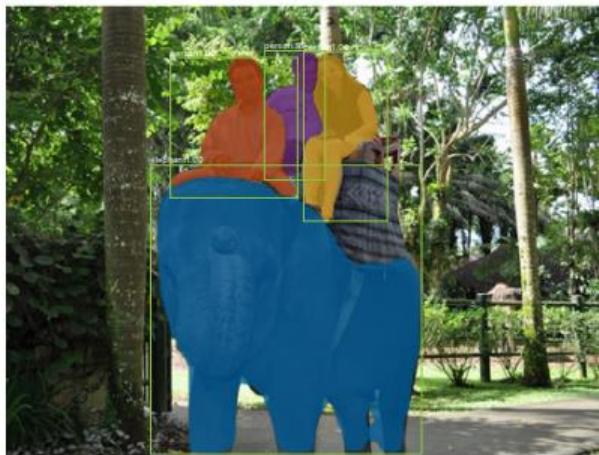
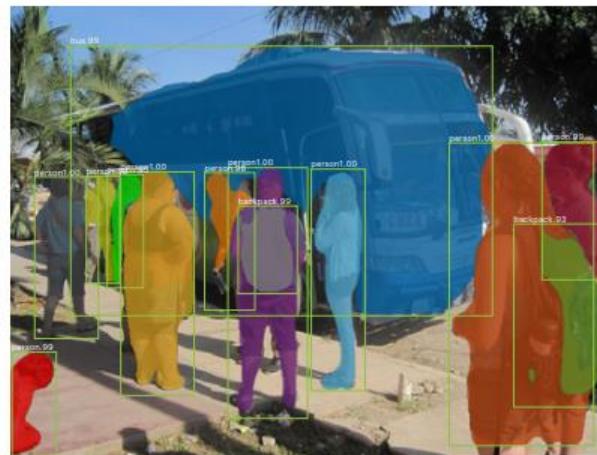
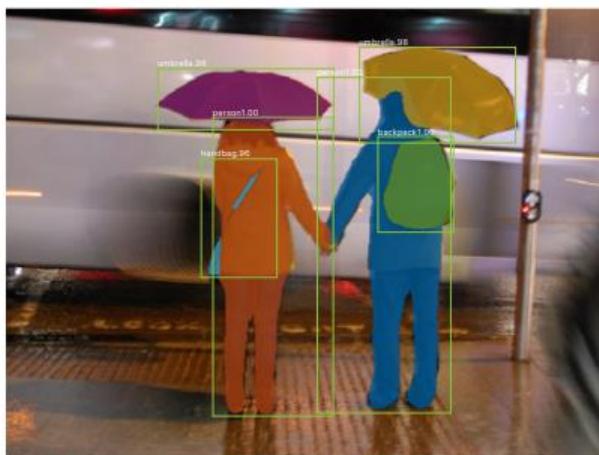
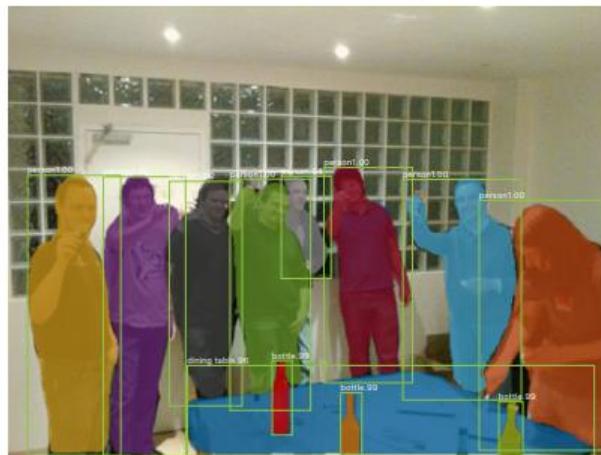
image with training proposal



28×28 mask target



results on MS COCO data set



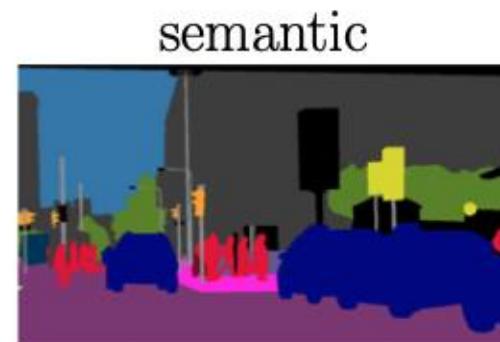
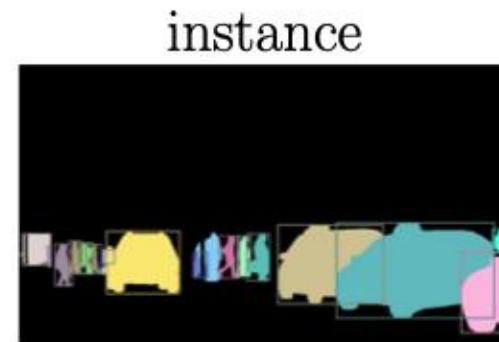
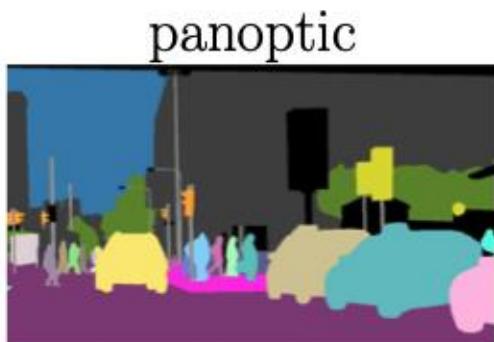
Panoptic Segmentation

unification of semantic segmentation (labeling all pixels) and instance segmentation (identifying individual objects)

each pixel is assigned

- a class label (e.g., road, car)
- an instance ID, if it belongs to a “thing” class (e.g., car #1, car #2)

covering both “thing” classes (countable objects) and “stuff” classes (uncountable regions like sky, grass, road)



e.g., with combination
of Mask R-CNN and
fully-convolutional head
(or DETR variants such
as Mask2Former)

Promptable Segmentation with Transformers

Segment Anything Model ([SAM](#))

[SAM2](#) includes also video segmentation

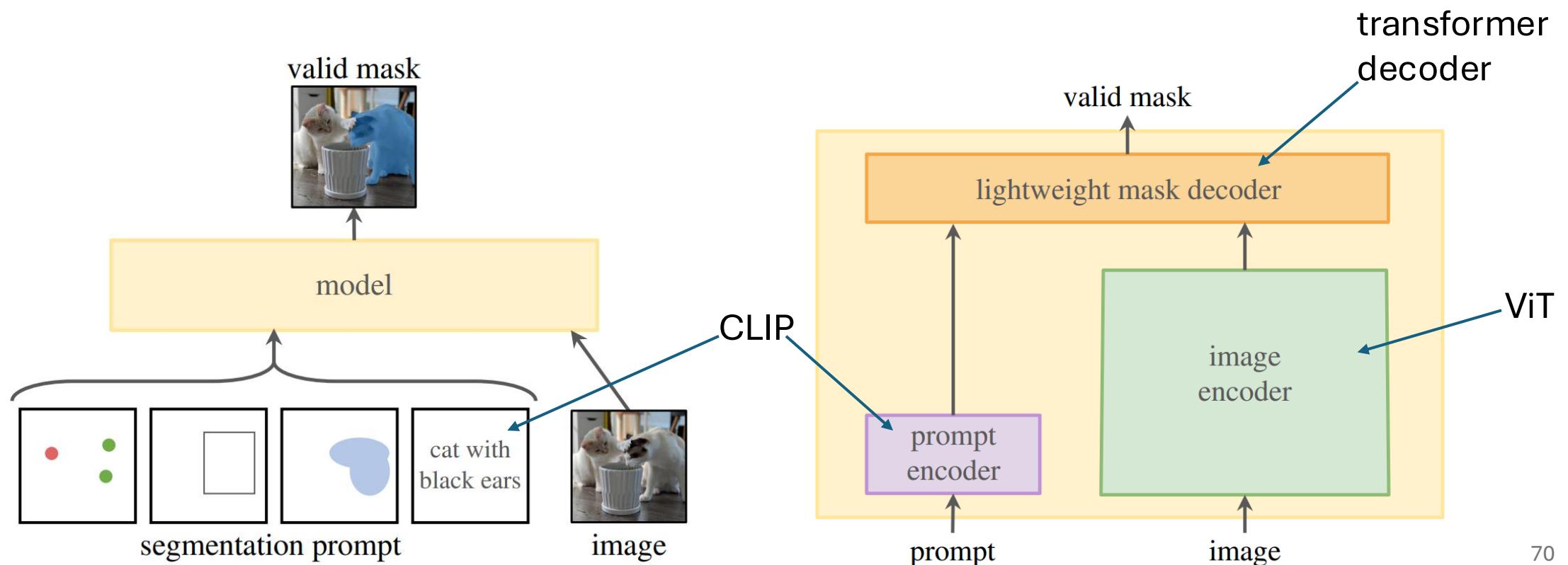


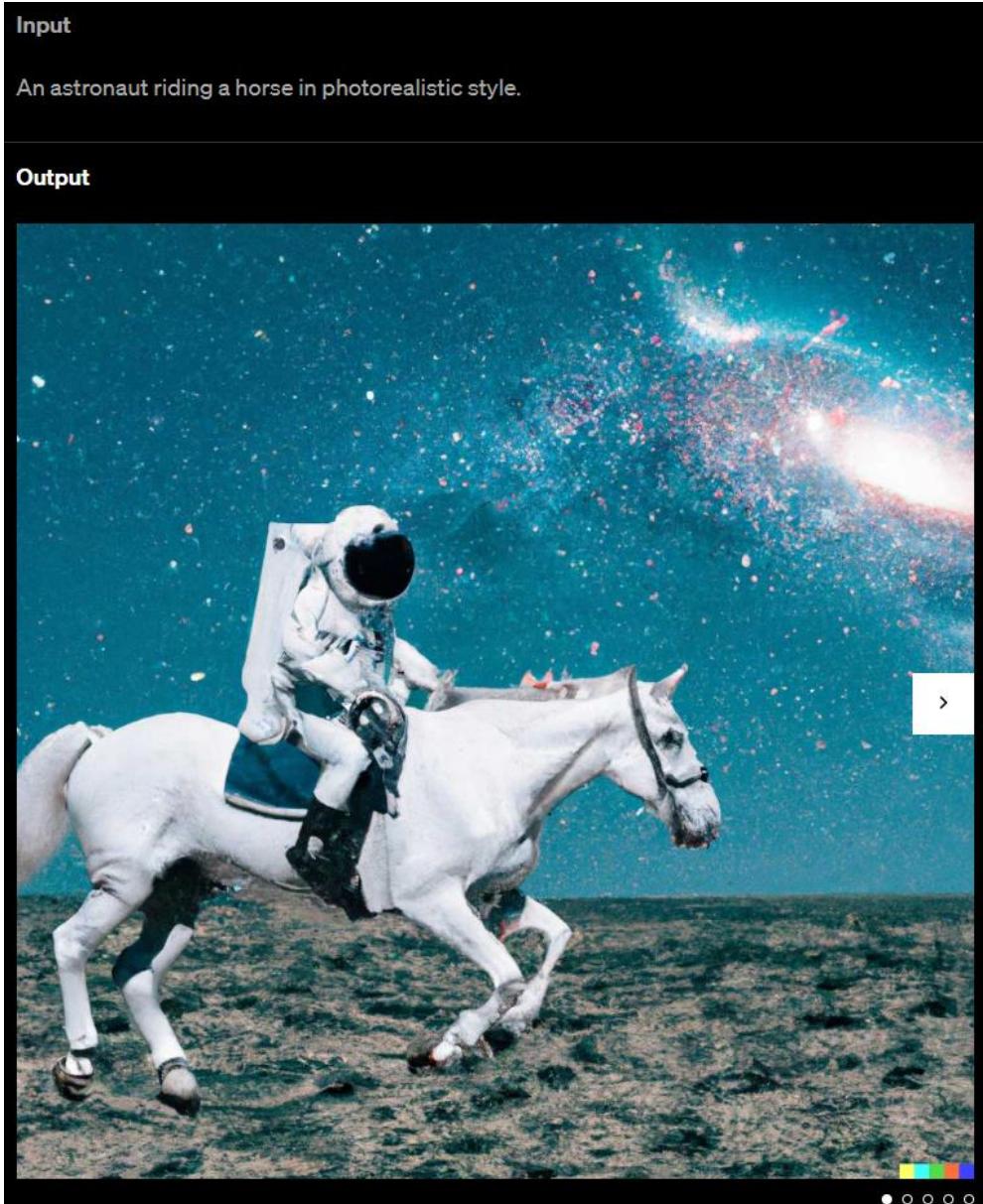
Image Synthesis

example: DALL-E 2

idea: generate new images as variations of training data (same distribution)

usually conditioned on text (prompt)

compared to text generation, additional mechanism needed (e.g., diffusion) due to more complex image structures



plenty of products: [DALL-E](#), [Stable Diffusion](#), [ImageGen](#), [Midjourney](#), ...

Stable Diffusion:



A scenic view of mountains at sunset, digital art

← prompt

“zebras roaming in the field”



source

Generative vs Predictive/Discriminative Models

discriminative models:

predict conditional probability $P(Y|X)$

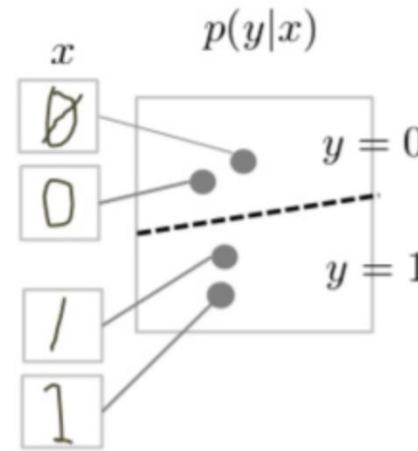
generative models:

predict joint probability $P(Y, X)$

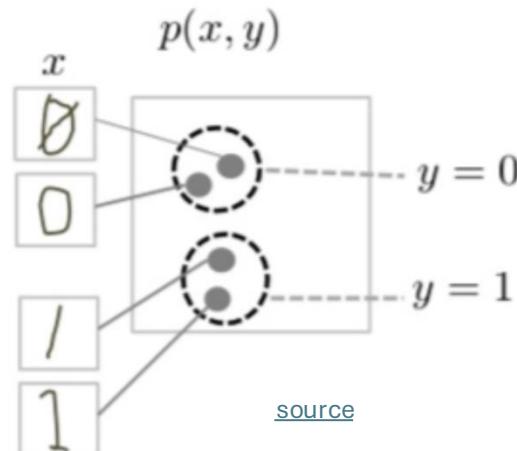
(or just $P(X) \rightarrow$ unsupervised learning)

\rightarrow allow to generate new data samples

discriminative model



generative model



task of generative models more difficult: need to model full data distribution rather than merely find patterns in inputs to distinguish outputs

Generative models can be used for predictive tasks (Bayes theorem).
But predictive models are usually better at it.

Classic ML vs Deep Learning

Generative vs Predictive Models

text generation

```
done = False
total_reward = 0
while not done:
    state = torch.tensor(state, dtype=torch.float32)
    logits = policy_network(state)
    action = torch.argmax(logits).item()
    next_state, reward, done, _ = env.step(action)
    total_reward += reward
    state = next_state
    total_rewards.append(total_reward)

# Print average reward
print("Average Reward:", np.mean(total_rewards))
```

This code uses PyTorch to implement the policy gradient method (REINFORCE algorithm) to solve the CartPole problem. The policy network is defined as a simple feedforward neural network, and the training loop updates the policy network parameters to maximize the expected reward. Finally, it evaluates the learned policy by running it for 100 episodes and prints the average reward. Adjust hyperparameters and network architecture as needed for better performance.

Can you extend this to an actor-critic method?

Certainly! Here's an extension of the previous code using the actor-critic method to solve the CartPole problem:

Message ChatGPT

ChatGPT can make mistakes. Check important info.

ChatGPT

image synthesis



Prompt: Epic anime artwork of a wizard atop a mountain at night casting a cosmic spell into the dark sky that says "Stable Diffusion 3" made out of colorful energy

[Stable Diffusion 3 — Stability AI](#)

text-to-video



Prompt: A stylish woman walks down a Tokyo street filled with warm glowing neon and animated city signage. She wears a black leather jacket, a long red dress, and black boots, and carries a black purse. She wears sunglasses and red lipstick. She +

[Sora | OpenAI](#)

BERT family



dmrc
XGBoost

tabular data

computer vision

Classify



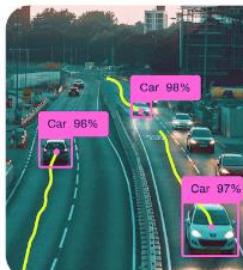
Detect



Segment



Track



YOLO

Deep Learning for Generative AI

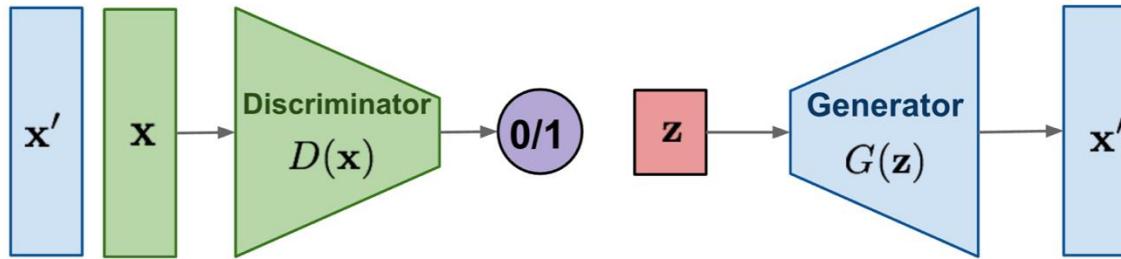
Depending on the application, there are currently two dominant approaches for generative AI:

- text generation: LLMs
- image synthesis: diffusion models

note the difference between image synthesis and multimodal understanding in LLMs
(images as additional input sequences to transformer, tokenized by splitting into patches)

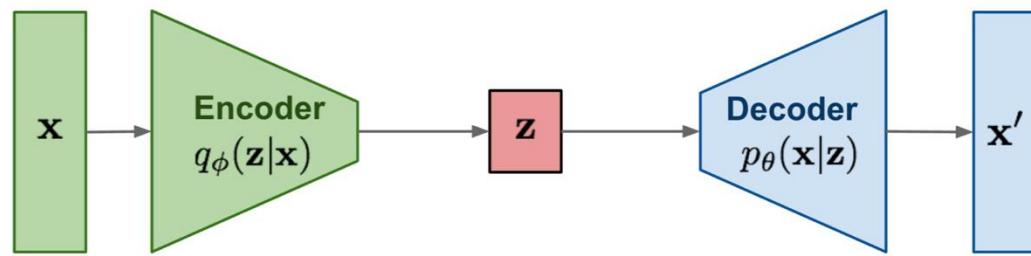
Different Model Types for Image Synthesis

GAN: Adversarial training



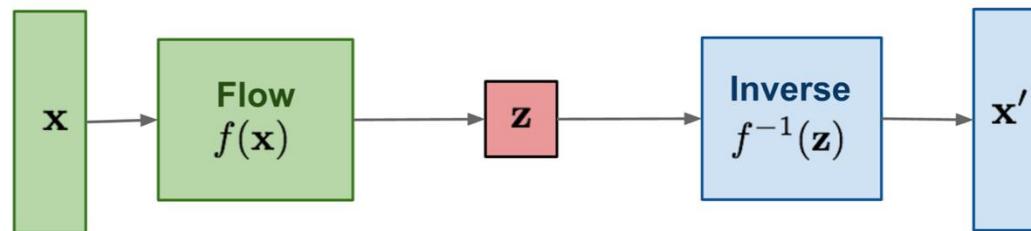
two neural networks playing a zero-sum game

VAE: maximize variational lower bound



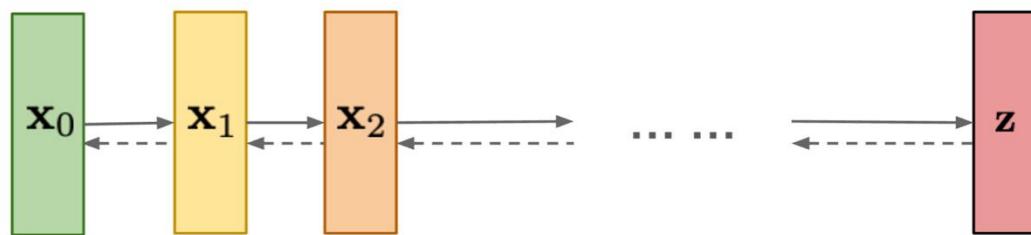
learn variational distribution (not just replicating inputs)

Flow-based models:
Invertible transform of distributions



more complex distributions by applying change-of-variable technique (need for specialized architecture)

Diffusion models:
Gradually add Gaussian noise and then reverse

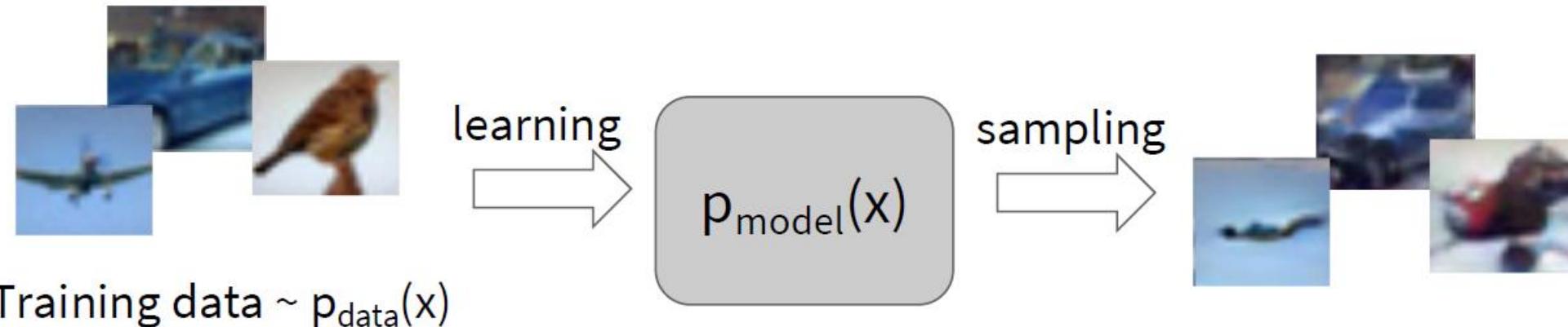


chain of denoising autoencoders

[source](#)

→ generalization: [flow matching](#)

Generative Modeling



Objectives:

1. Learn $p_{\text{model}}(x)$ that approximates $p_{\text{data}}(x)$
2. Sampling new x from $p_{\text{model}}(x)$

Explicit density estimation: explicitly define and solve for $p_{\text{model}}(x)$

Implicit density estimation: learn model that can sample from $p_{\text{model}}(x)$ without explicitly defining it.