Computer Vision

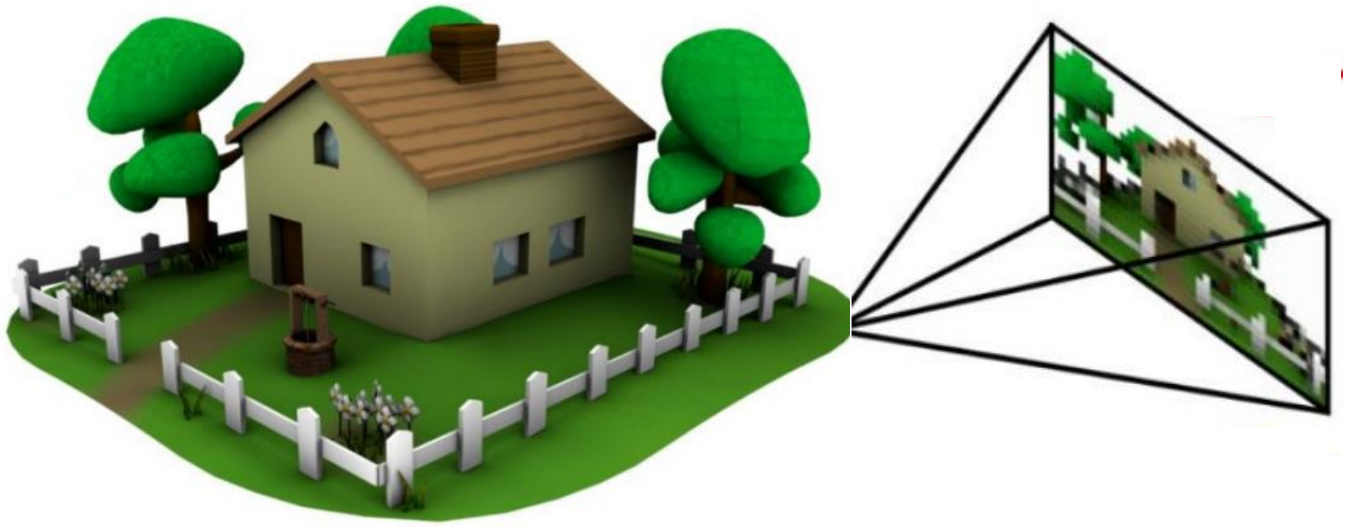
Fundamentals of Artificial Intelligence

Goal of Computer Vision

extract semantic information from digital image data

to be used for decision making support or automated systems

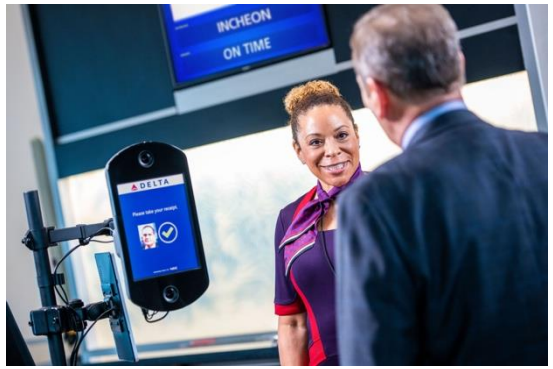
challenging problem:
images are only 2D projections
of the 3D world



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

Applications of Computer Vision

facial recognition



automated inspection

autonomous driving

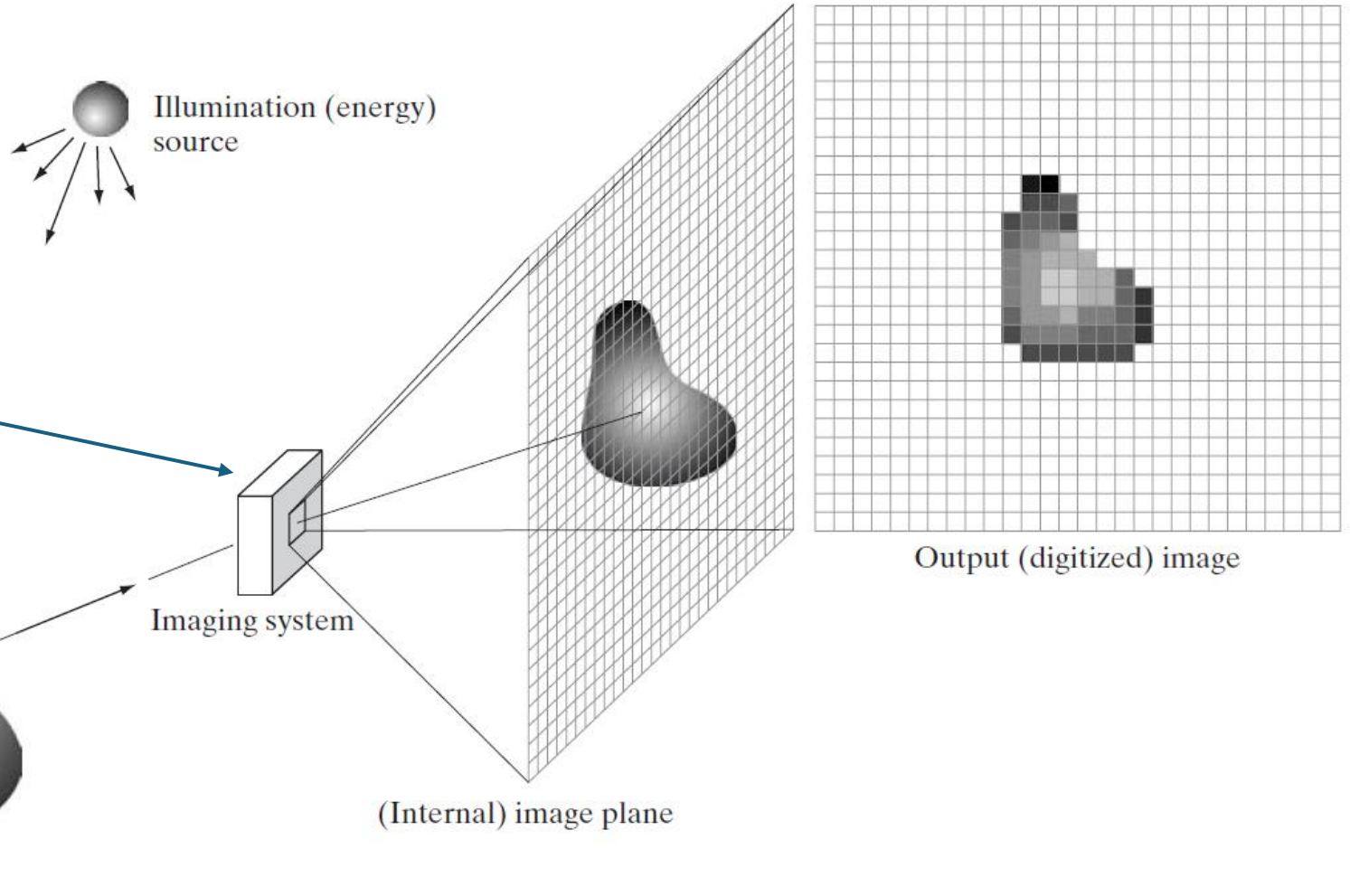
medical imaging

optical character recognition

augmented reality

and many more ...

Digital Image Acquisition Process



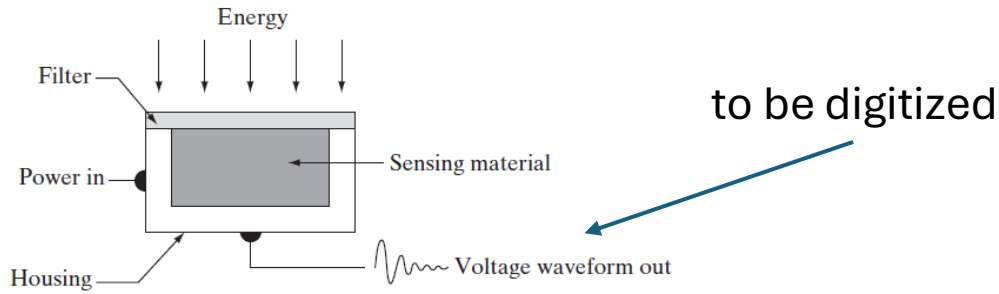
Photons to Electrons

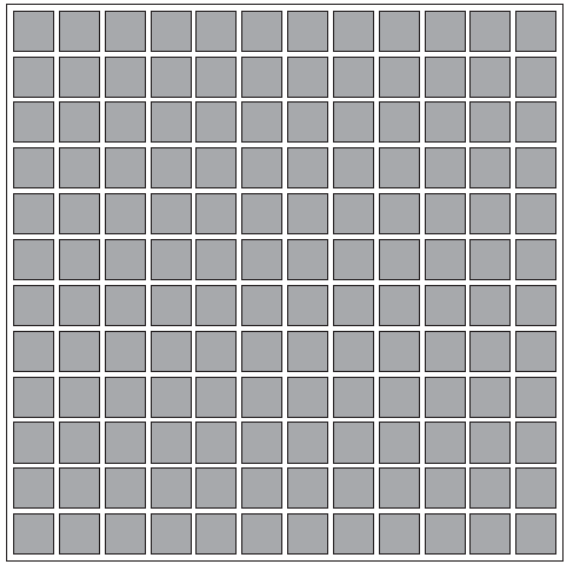
largely replaced chemical and analog imaging

single sensor: photodiode

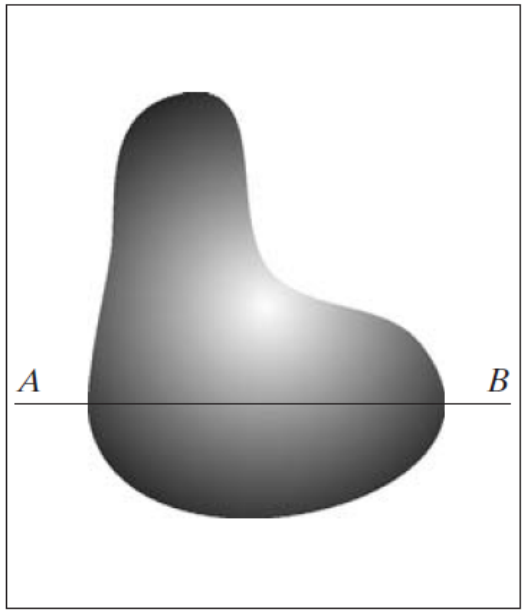
sensor strips: motion perpendicular to strip for imaging in other direction, used in CT

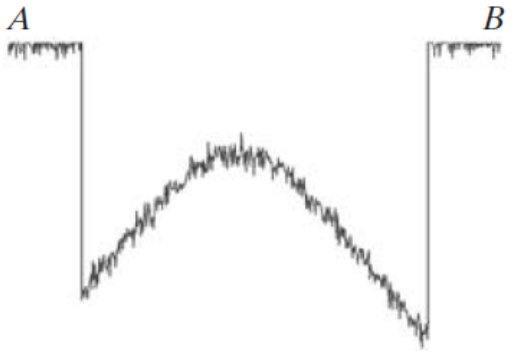
sensor arrays: used in digital cameras (in focal plane, outputs proportional to integral of light received at each sensor)

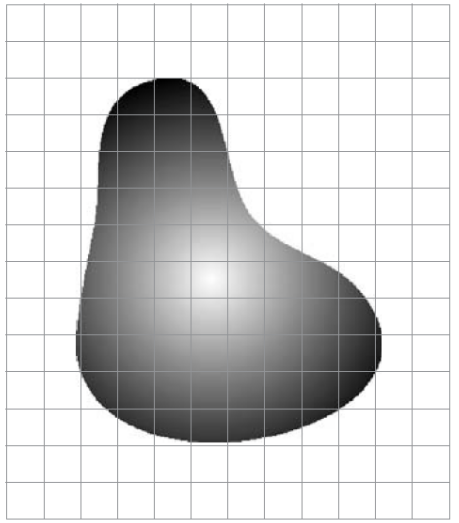


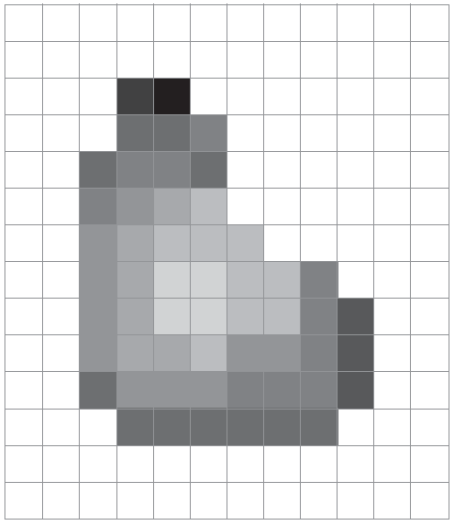


Digitizing: Sampling & Quantization

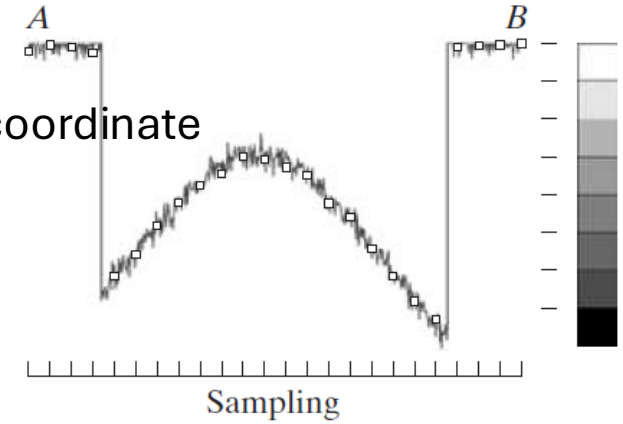


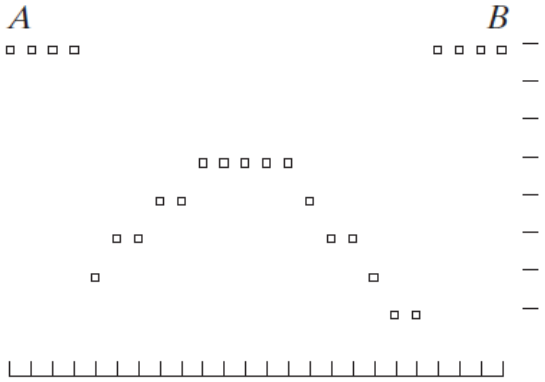






digitizing the coordinate values: pixels





digitizing the amplitude values: discrete intensity levels

Discrete Intensity Levels

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

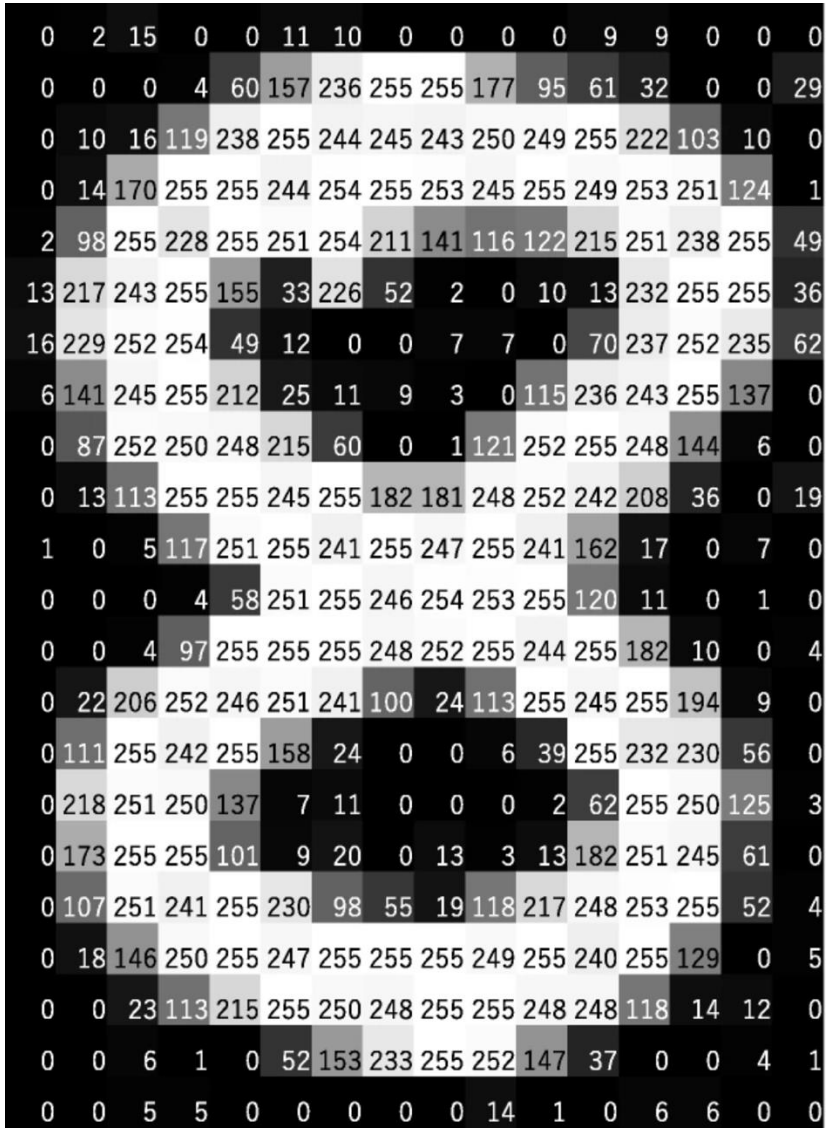
intensity resolution:

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)



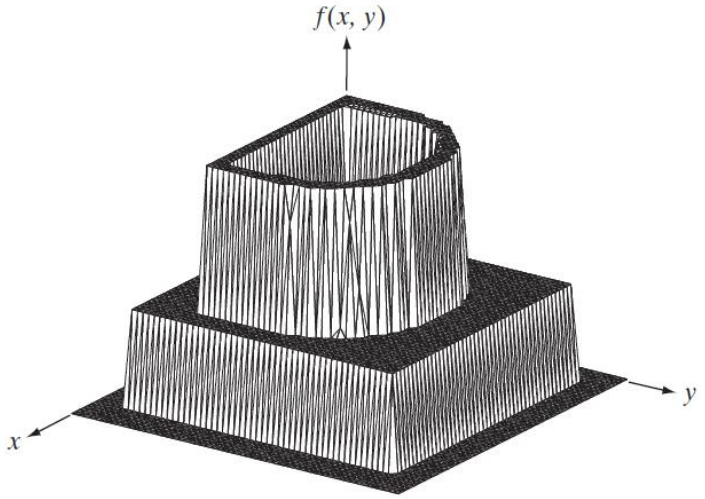
source

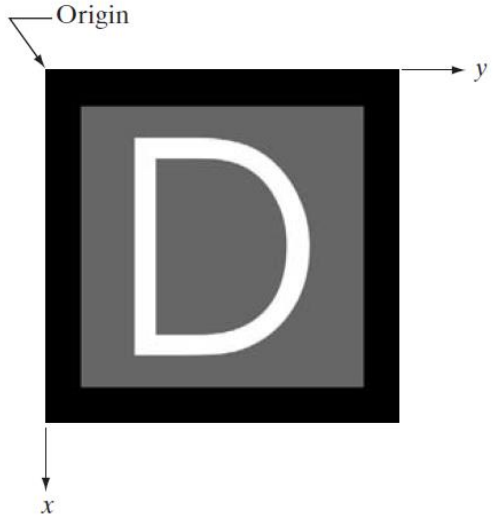
Image as Two-Dimensional Function

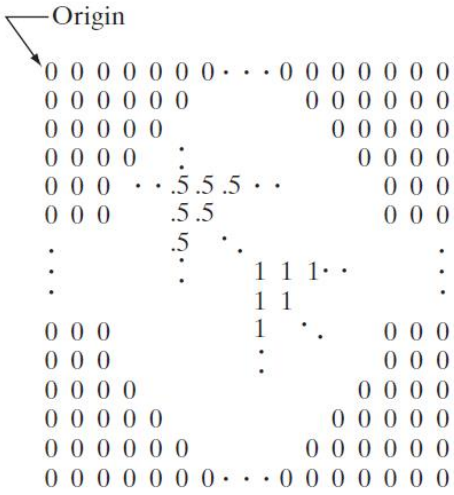
image can be seen as function $f(x, y)$: intensity at position (x, y)

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

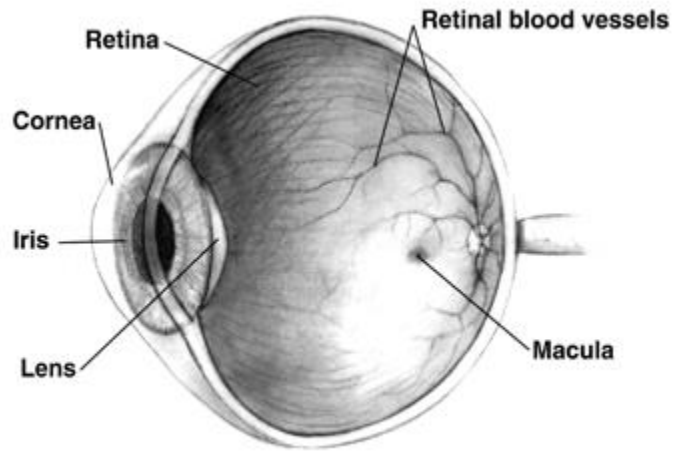
$$f(x, y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}$$







Colors in the Human Eye

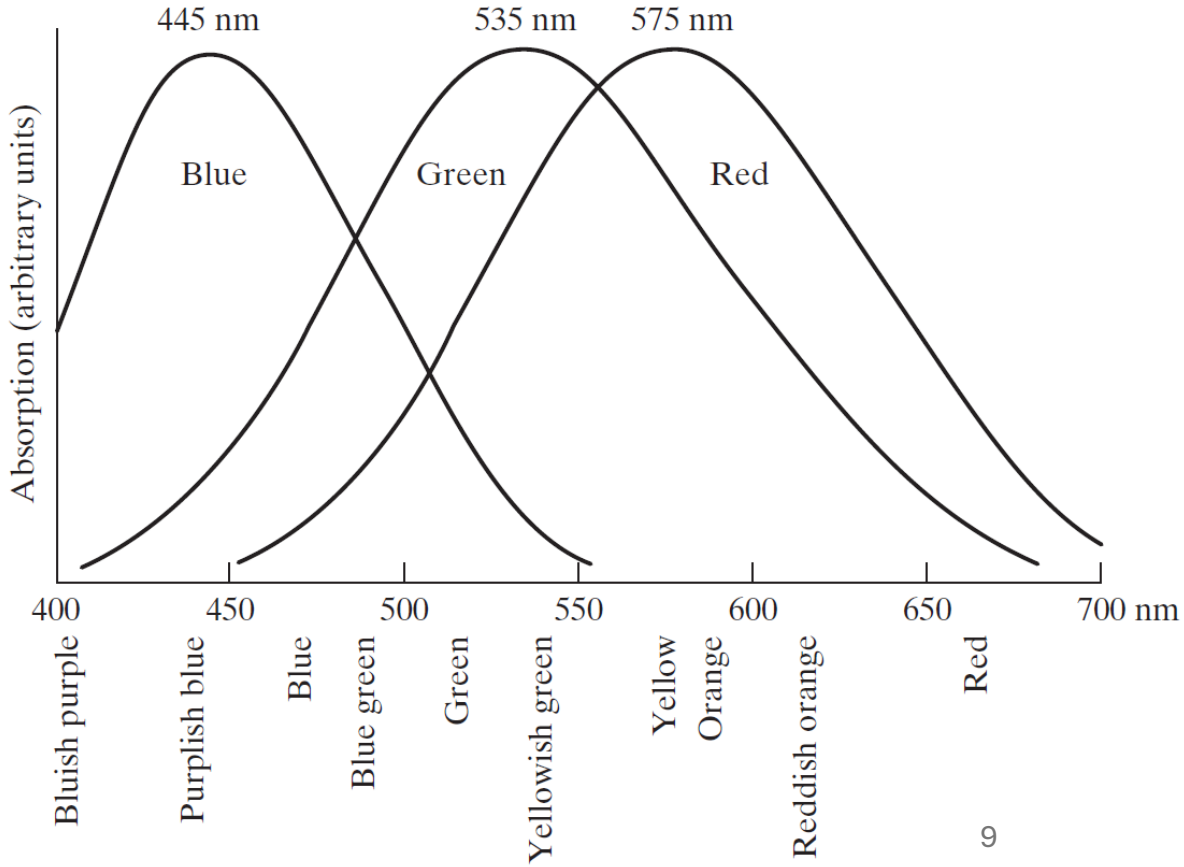


retina: image sensor of human eye

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

three types of cones: red, green, blue

→ combinations of red, green, and blue create perceived colors



Primary Colors

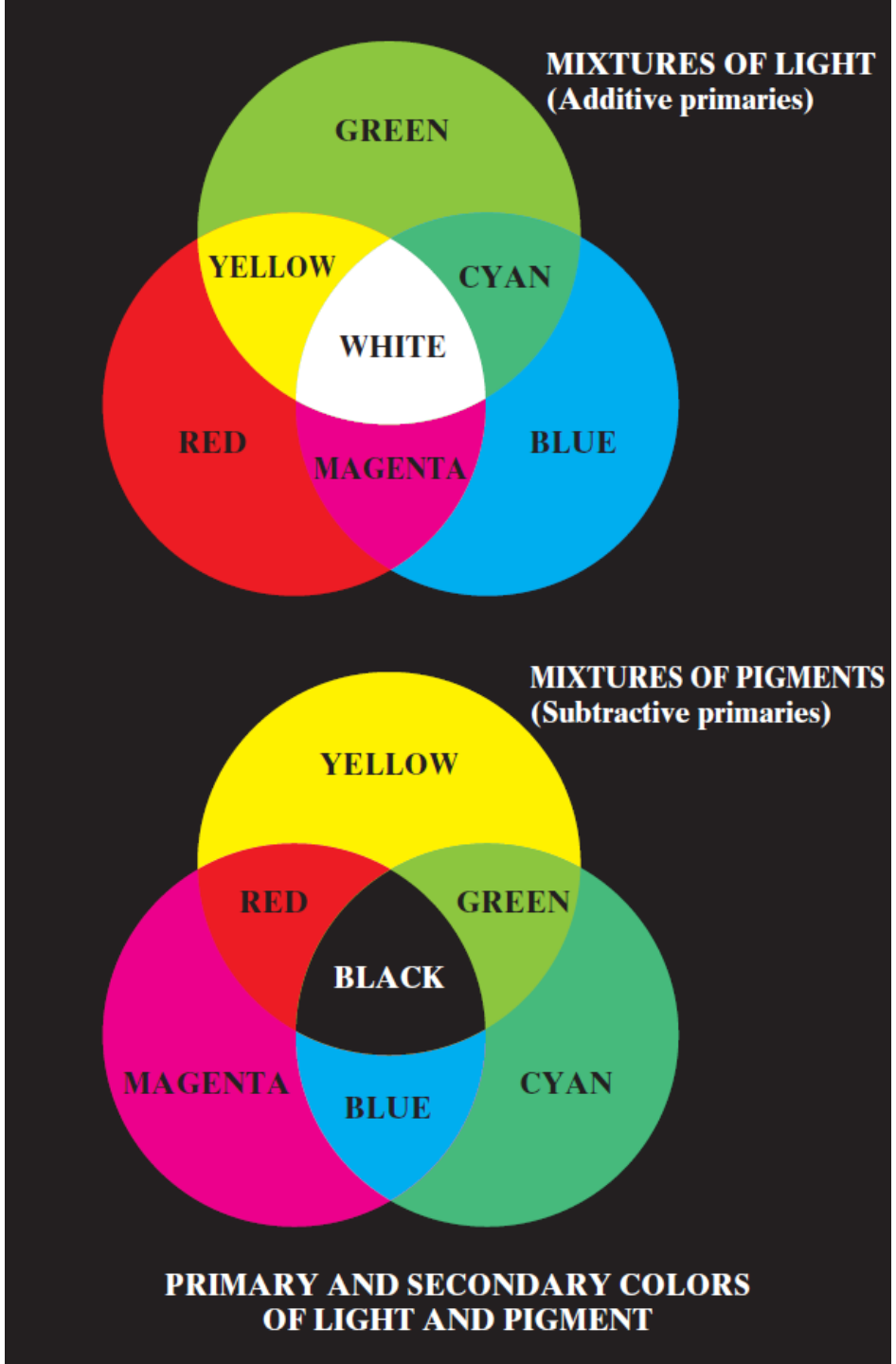
standardization:

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

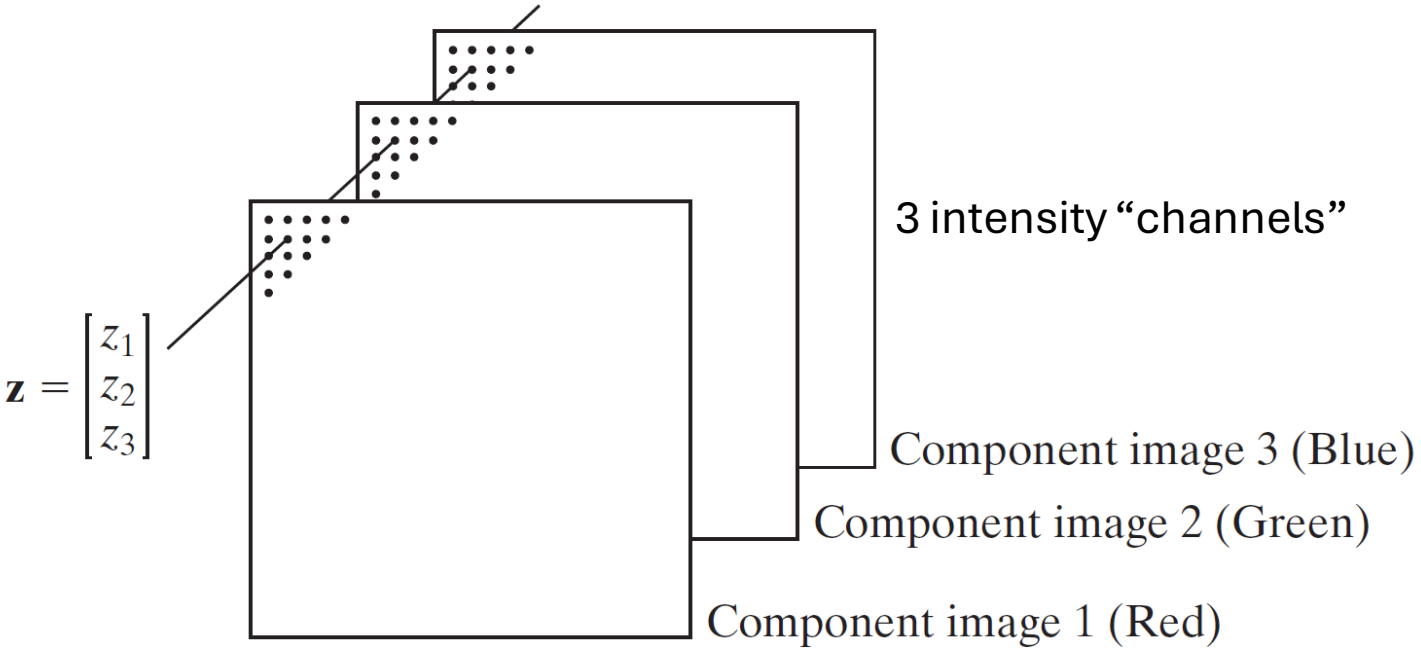


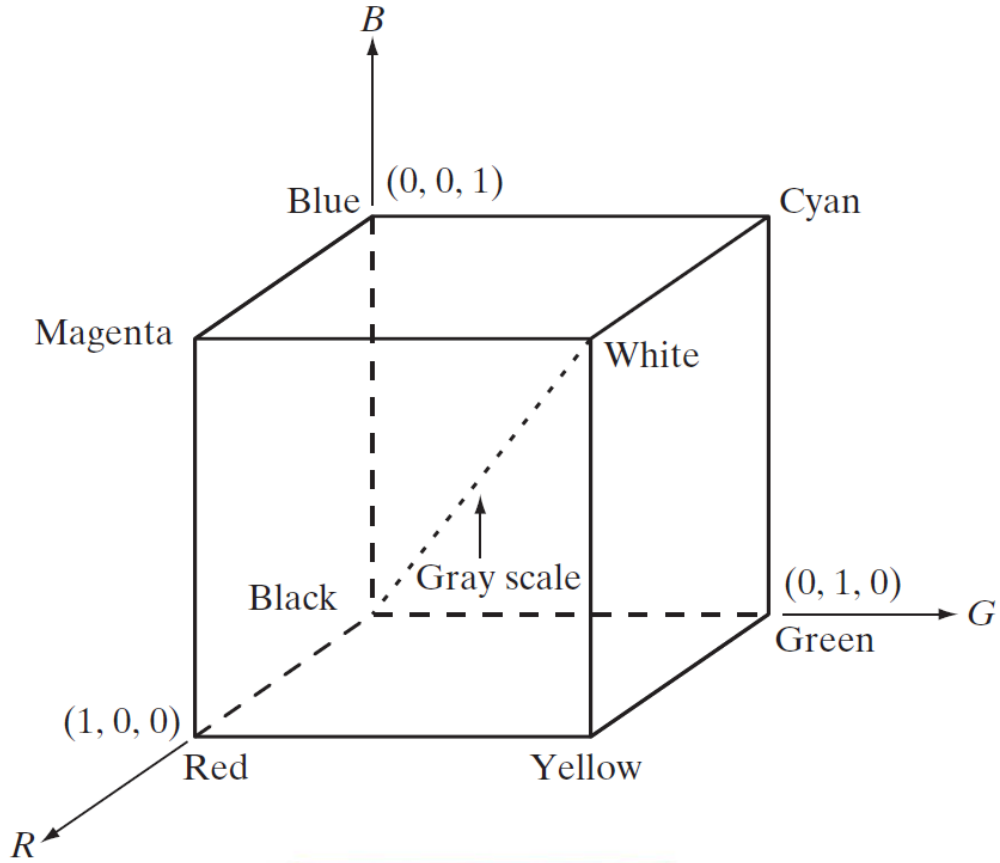
RGB Color Model

color model for monitors and cameras

(other color models: CMYK, HSV, …)

color pixels:



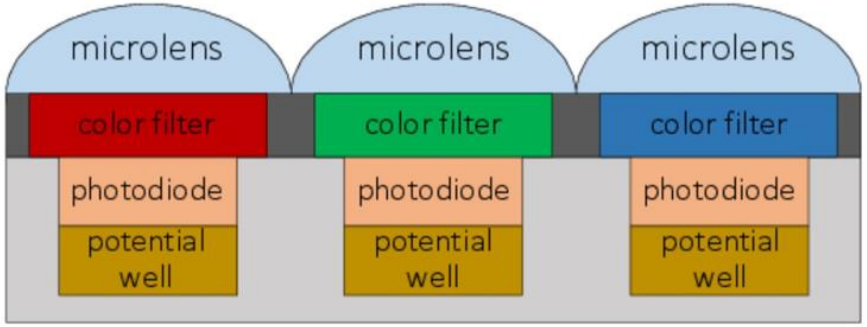


Color Sensing

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

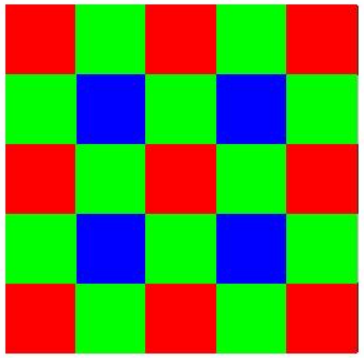


G	R	G	R
B	G	B	G
G	R	G	R
B	G	B	G

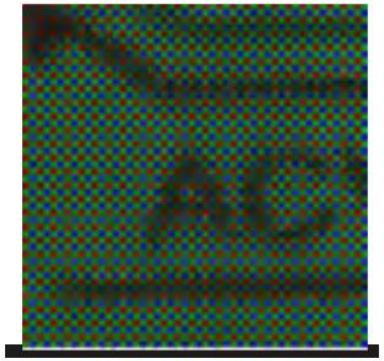
Bayer RGB Pattern

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

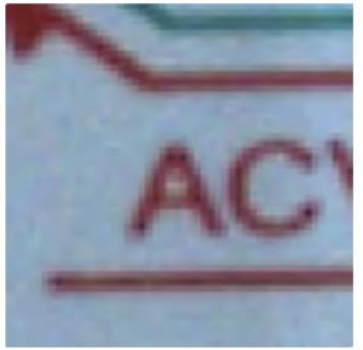
Interpolated Pixels



Bayer Pattern
(Color Filter Mosaic)



Raw Image

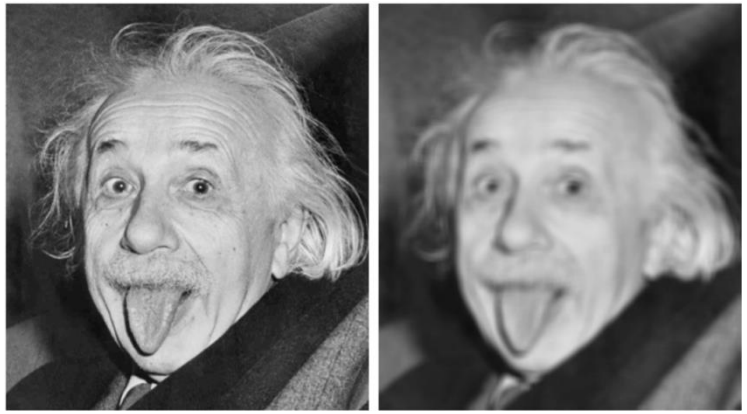


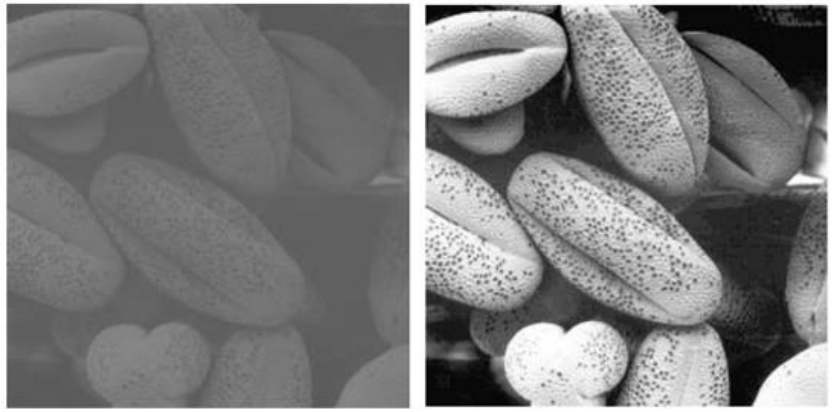
Interpolated Image

Image Processing

transformations from image to image

(such as scaling, smoothing, sharpening, or contrast stretching)

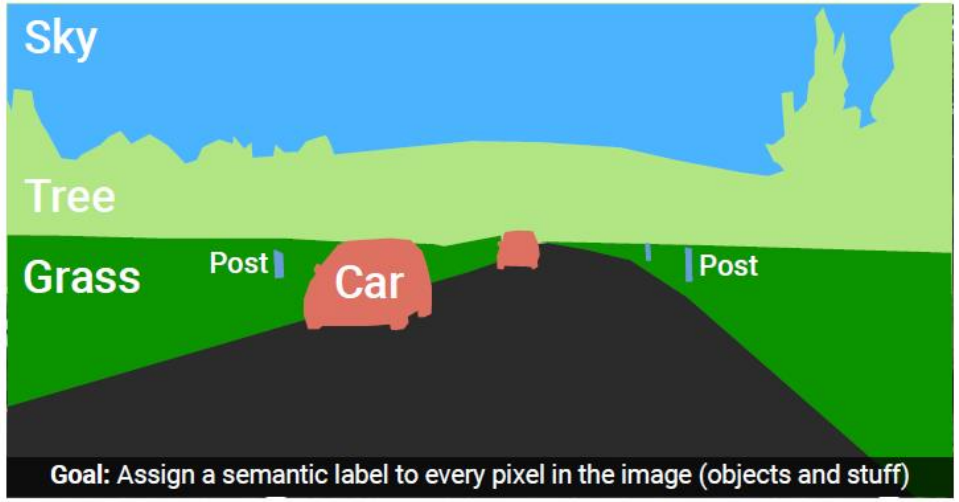




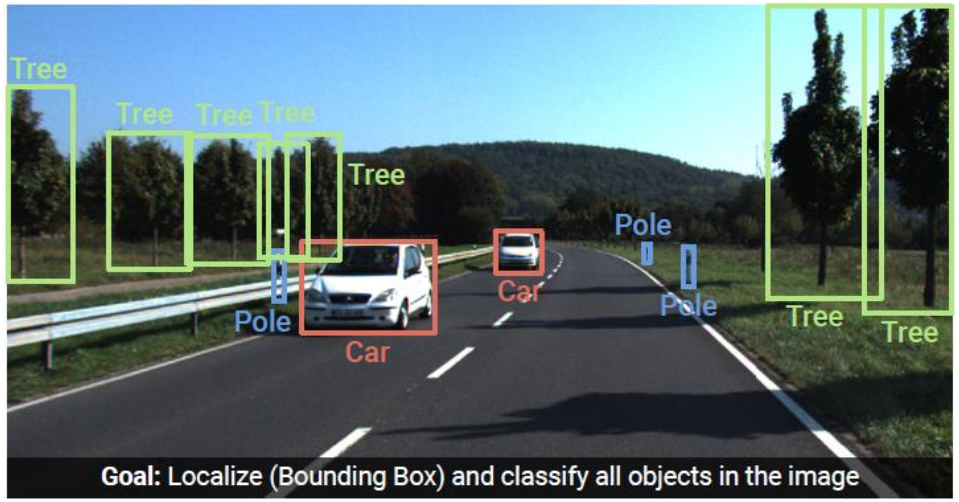
to facilitate either machine perception or just human interpretation

Image Understanding (Recognition)

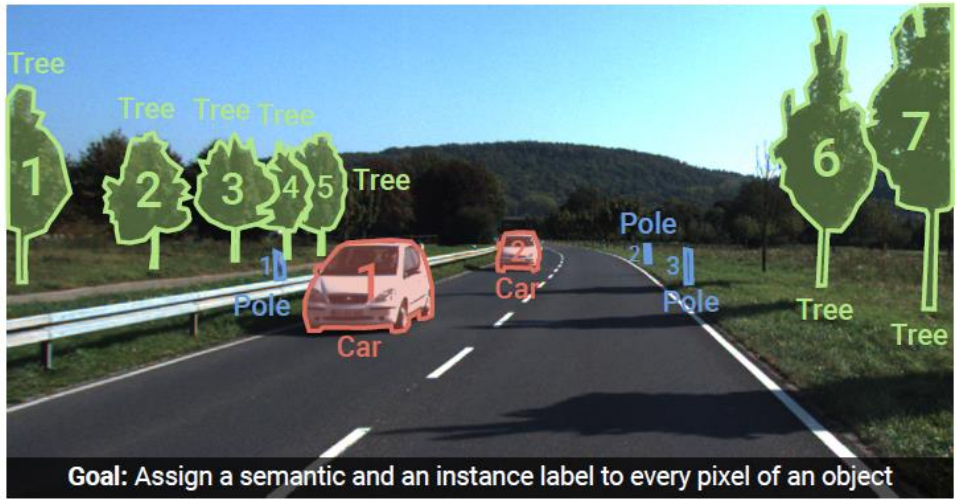
Image Classification



Semantic Segmentation



Object Detection



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”)

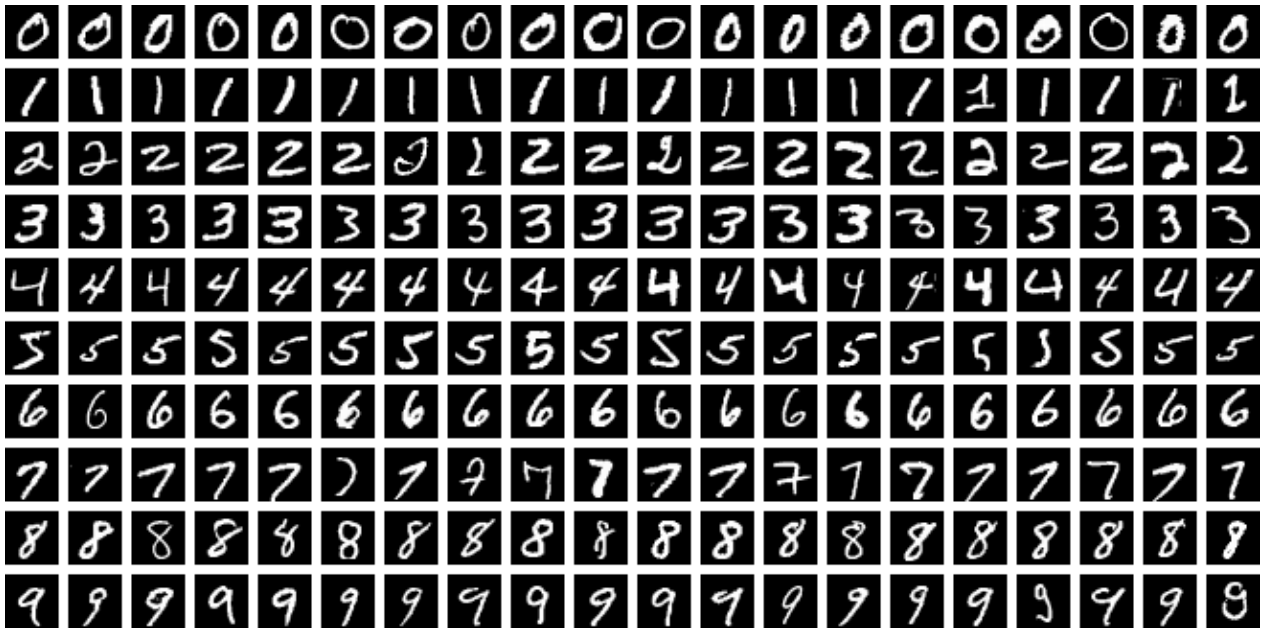
Let’s learn it from data …

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

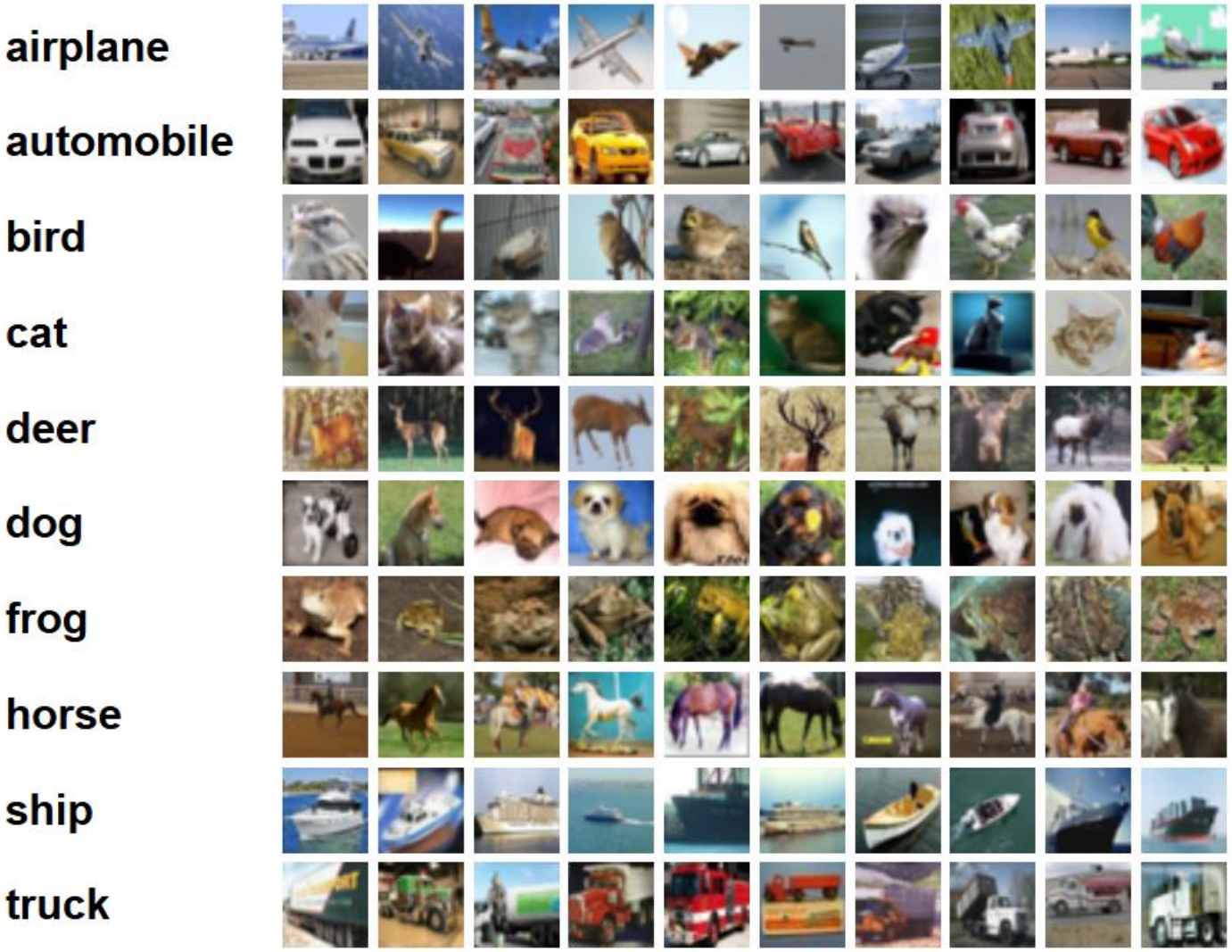
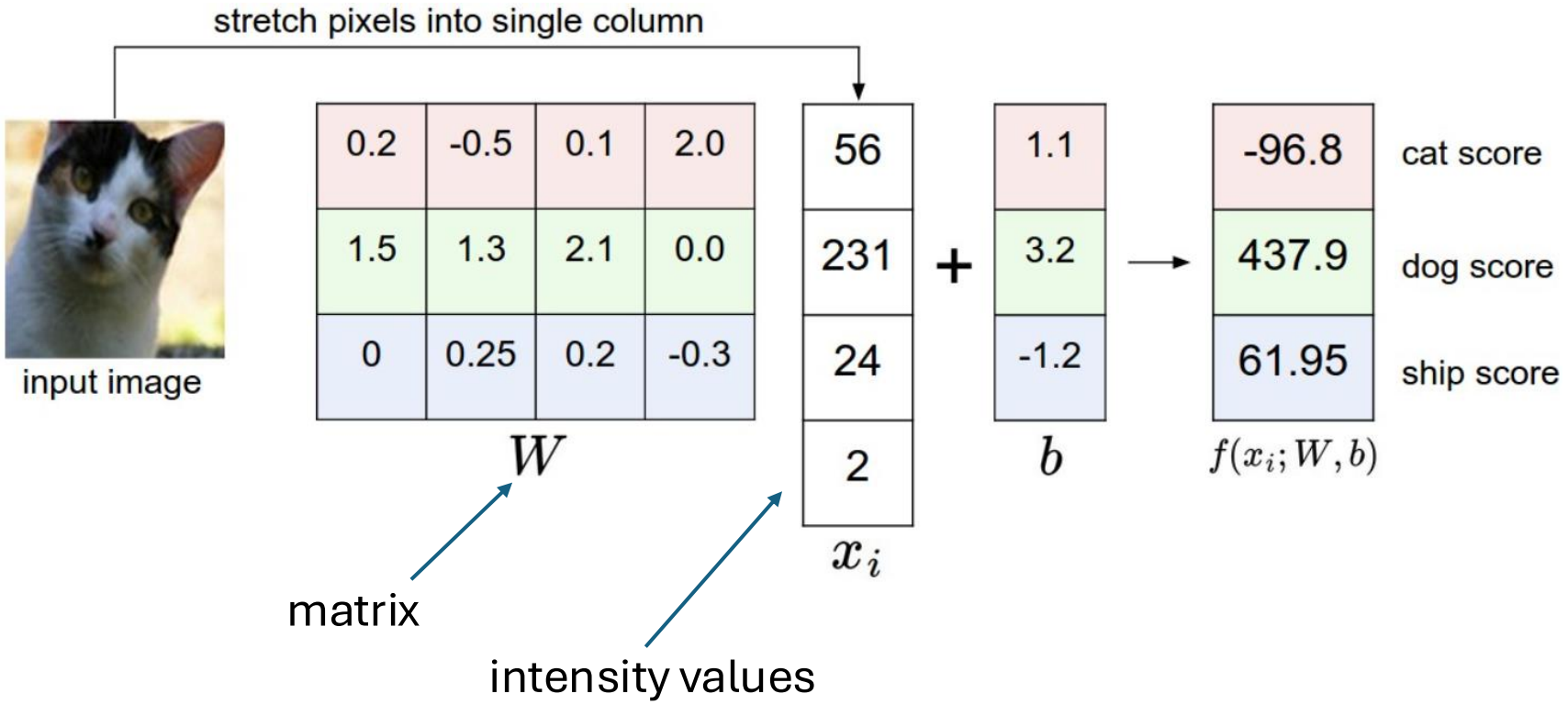


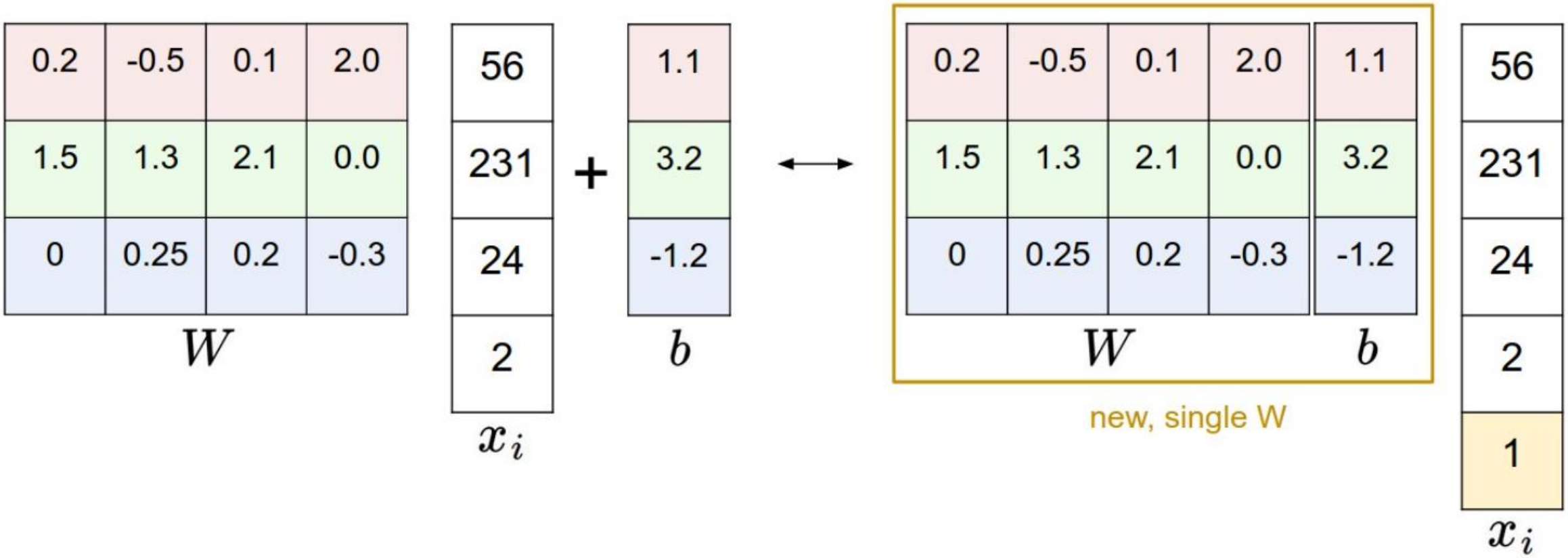
Image Classification with Linear Regression

simplified example: 4-pixel image, 3 classes

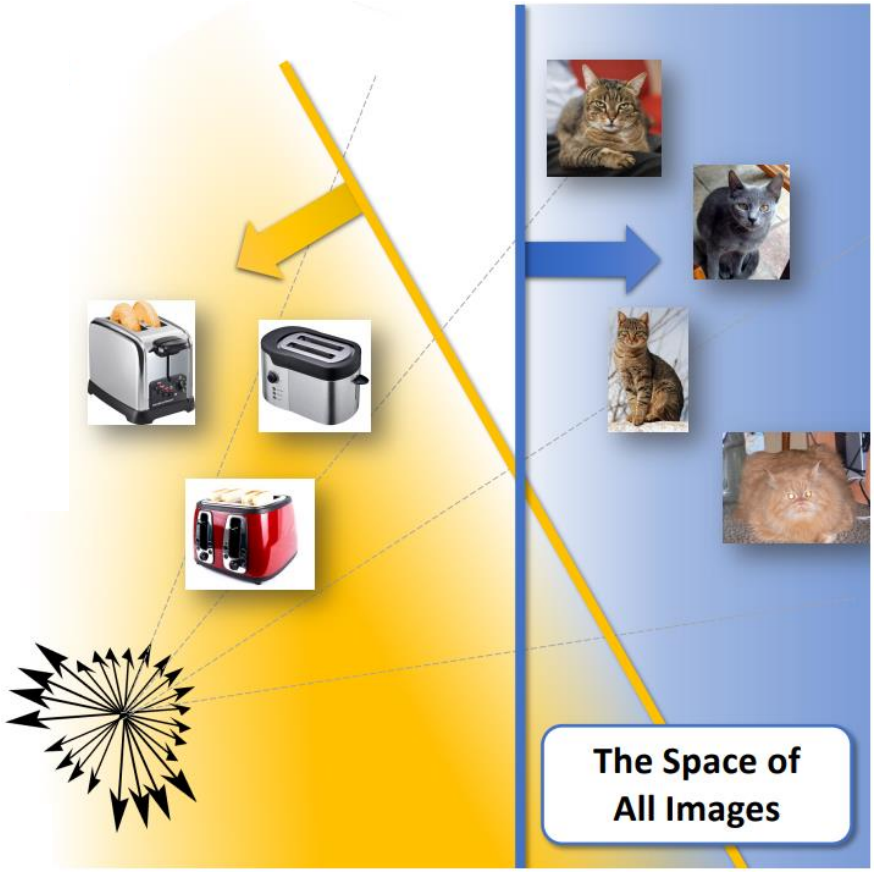


need for one common image size per model

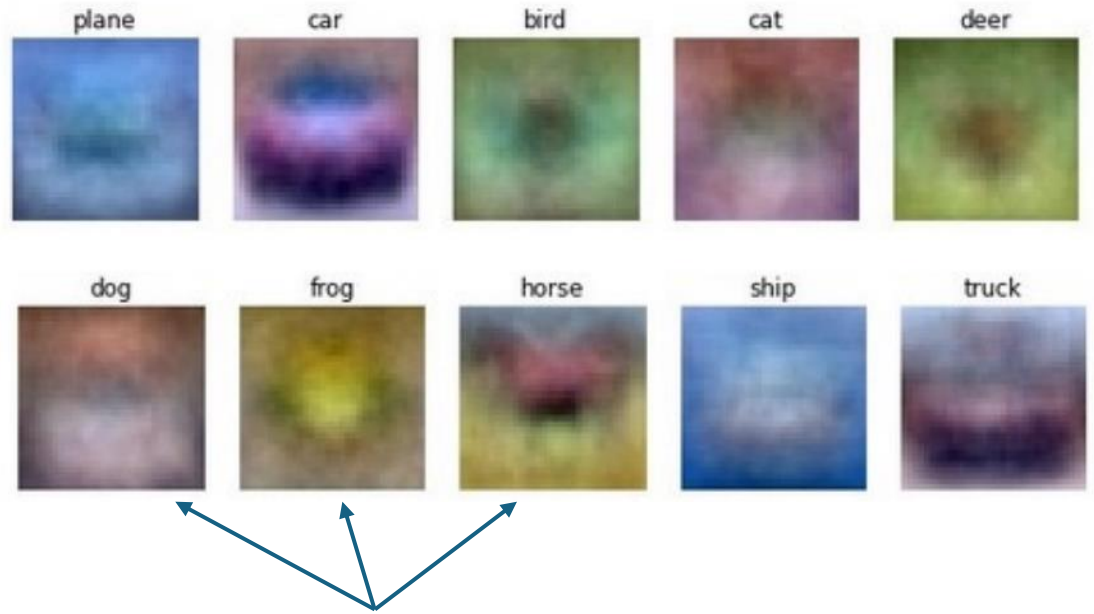
Simplified Matrix Multiplication: Bias Trick



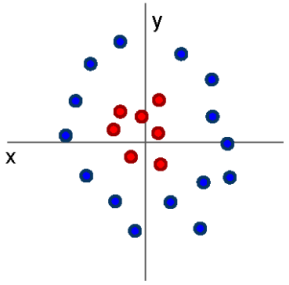
geometric interpretation:
separating hyperplanes



matching interpretation:
generic class templates



different rows in matrix W



Cannot separate red and blue points with linear classifier

raw-pixel images not linearly separable
→ linear model has not enough representational power

Image Classification with kNN

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

$$d(\boldsymbol{I}_1, \boldsymbol{I}_2) = \sum_p |I_1(p) - I_2(p)|$$

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

-

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

=

pixel-wise absolute value differences

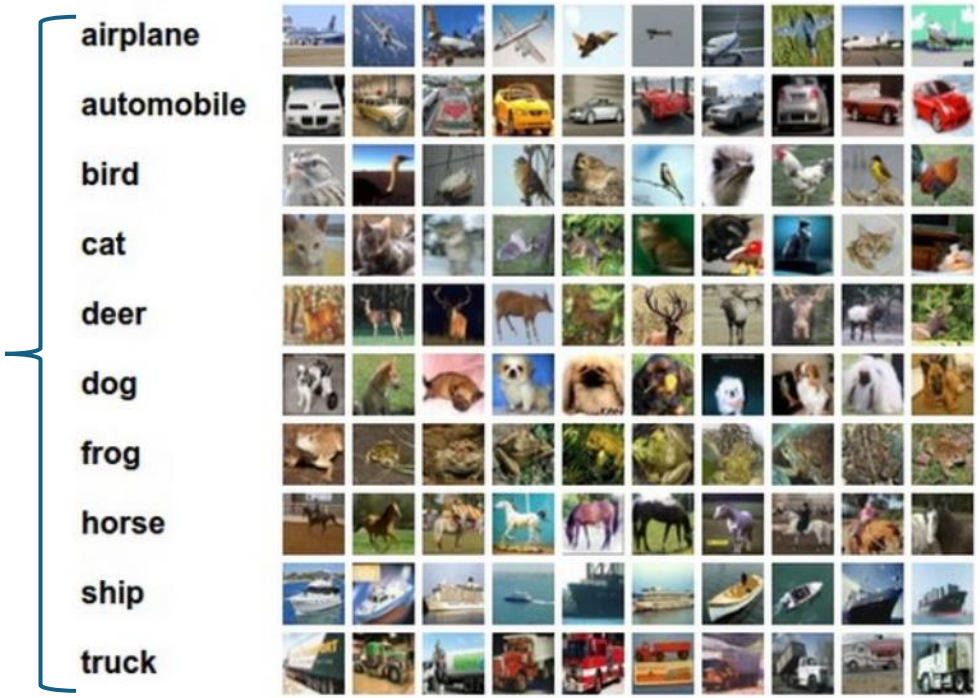
46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

→ 456

Image Classification with kNN

training examples CIFAR-10 data set

10 classes



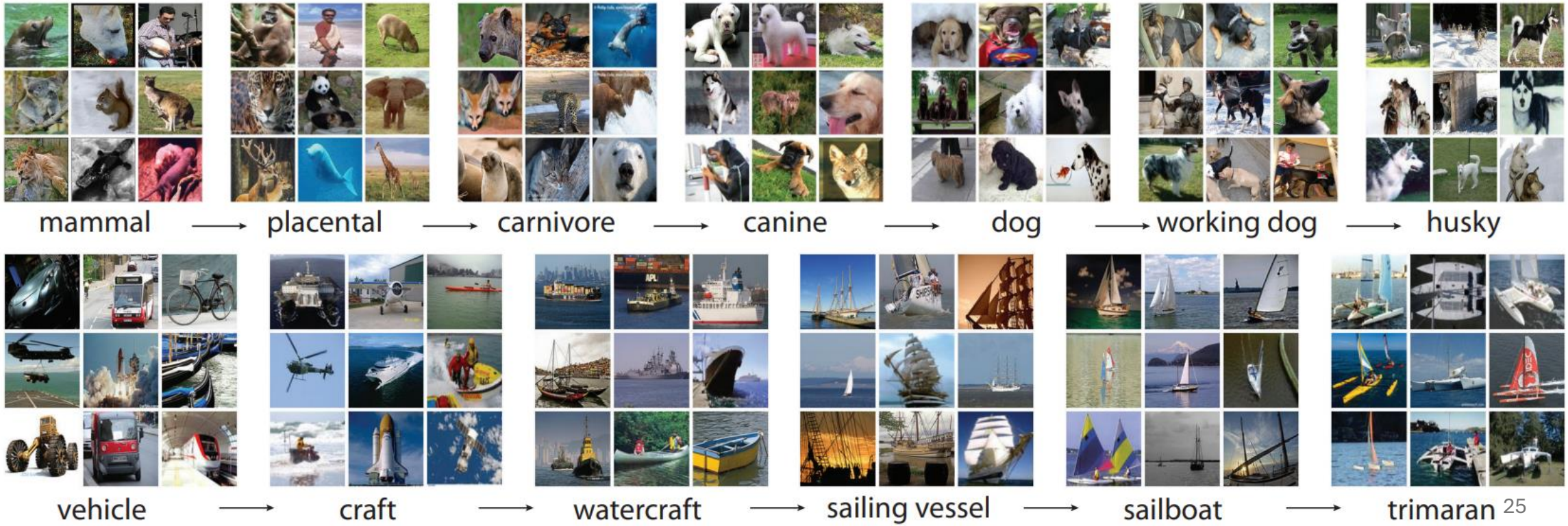
10 nearest neighbors to some test images

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

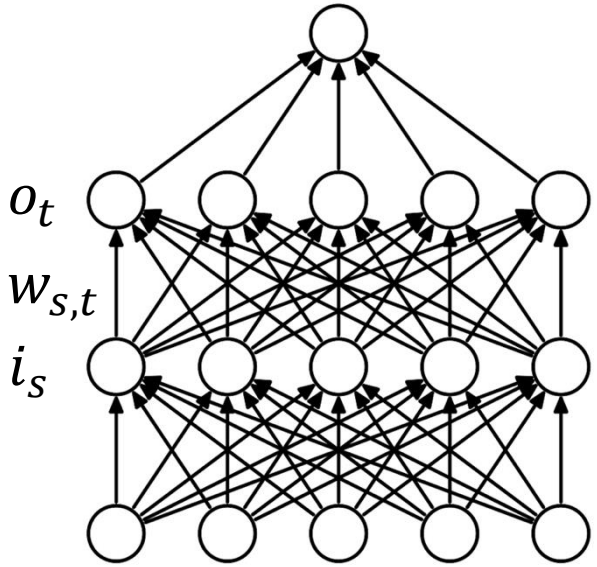


Recap: Feed-Forward Neural Networks

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

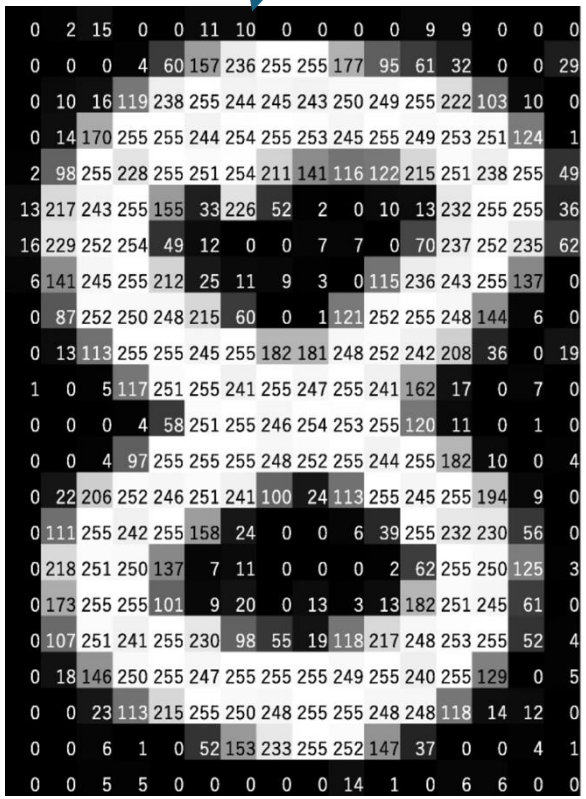
image data: 2-D grid of pixels (spatial structure)

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)



source

Convolution Operation

actually, correlation (convolution would have – here):

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

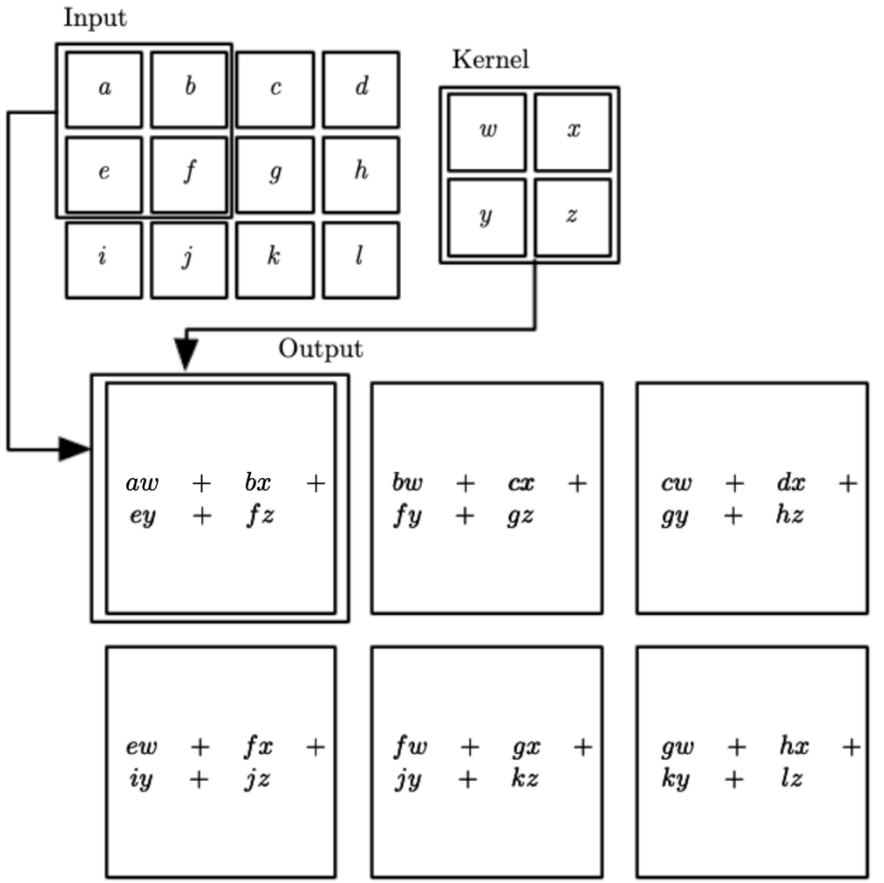
feature map
(transformed image)

kernel
(learned)

image
(input or feature map)

(again, dropping dimension of different training observations)

2D convolution

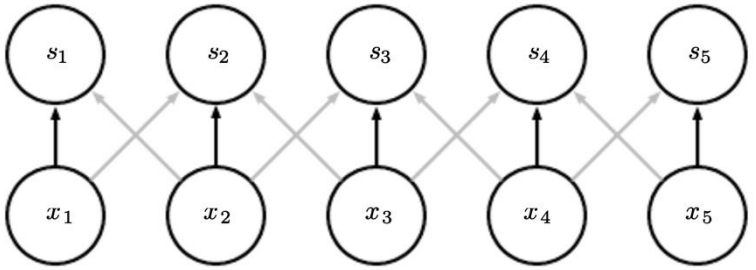


source

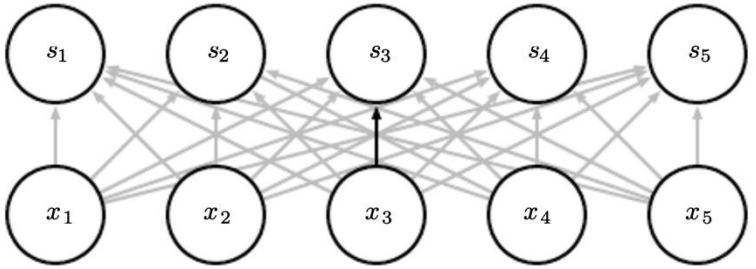
Regularization Effects

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

convolutional layer:



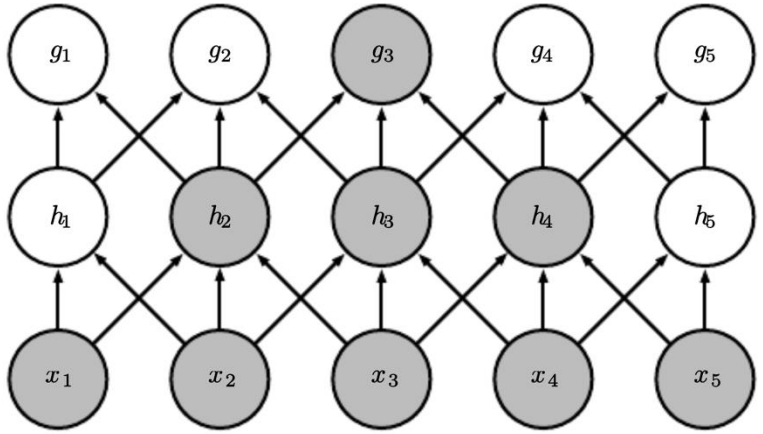
fully-connected layer:



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)

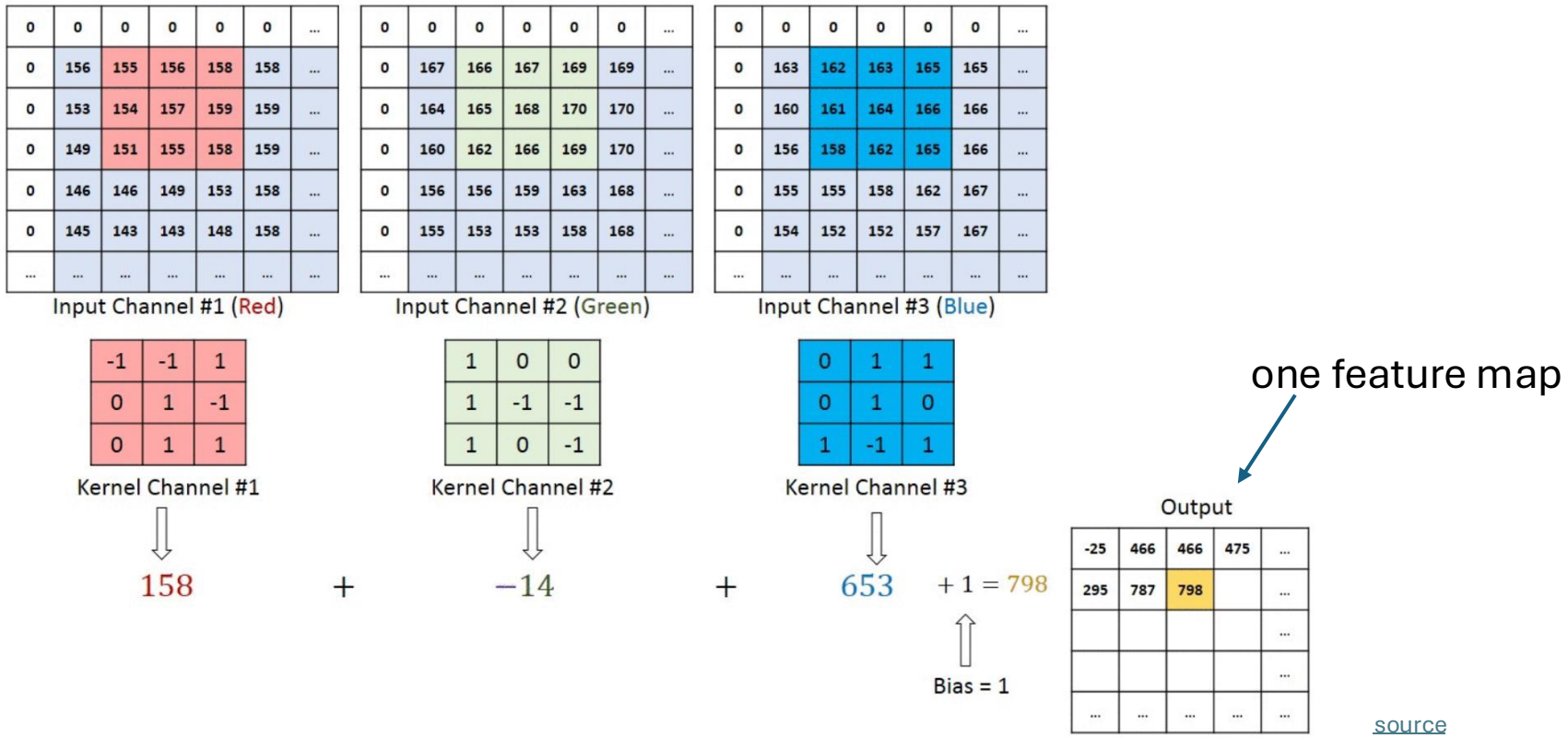


source

Channel Mixing

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

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



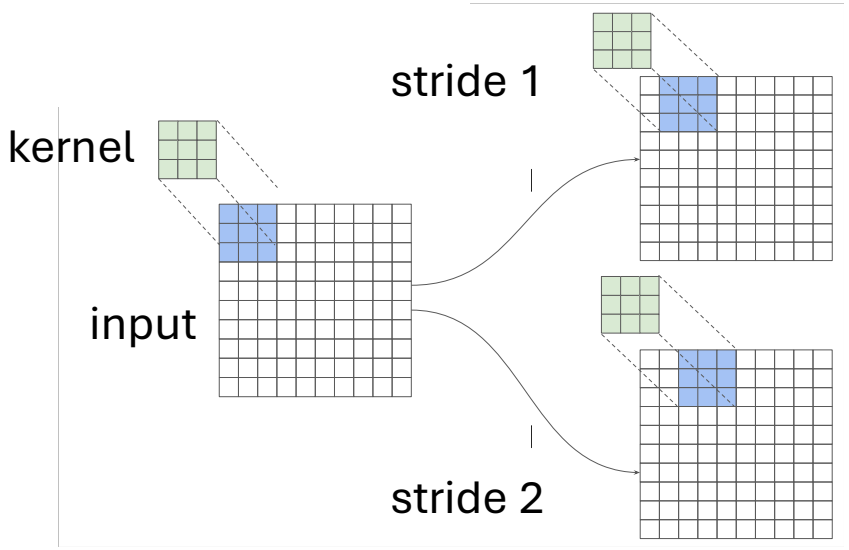
source

Striding and Padding

need to define how to stride over image

stride > 1 corresponds to down-sampling

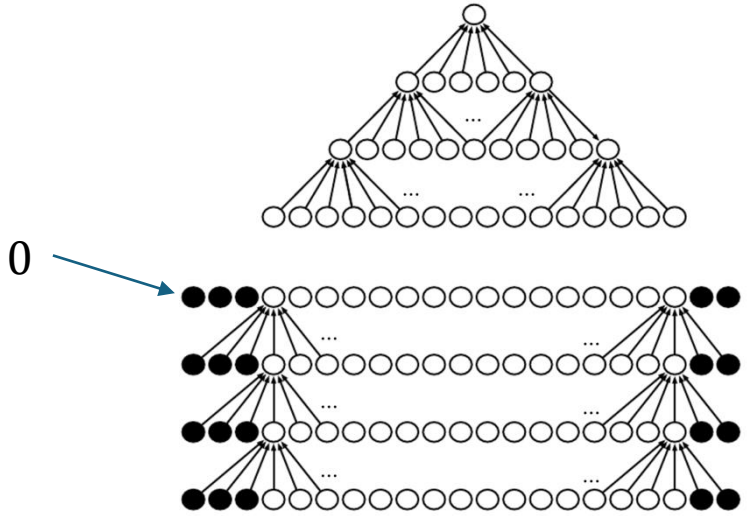
→ fewer nodes after convolutional layer



zero-padding of input to make it wider:

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

→ needed for large kernels and several layers



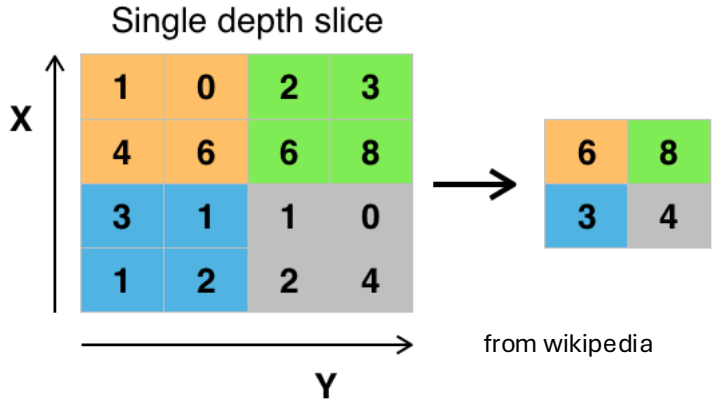
source

Another Ingredient: Pooling

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

→ non-linear downsampling (regularization)

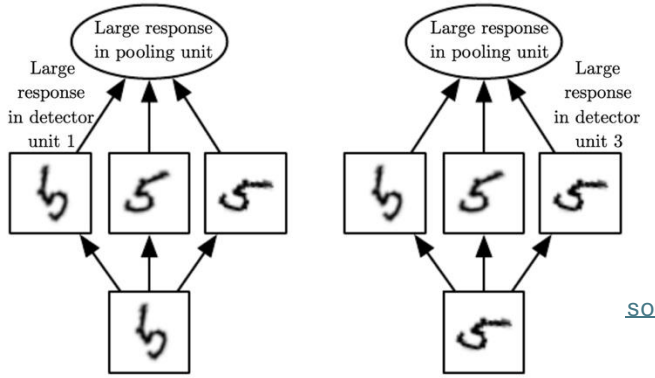
max pooling:



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

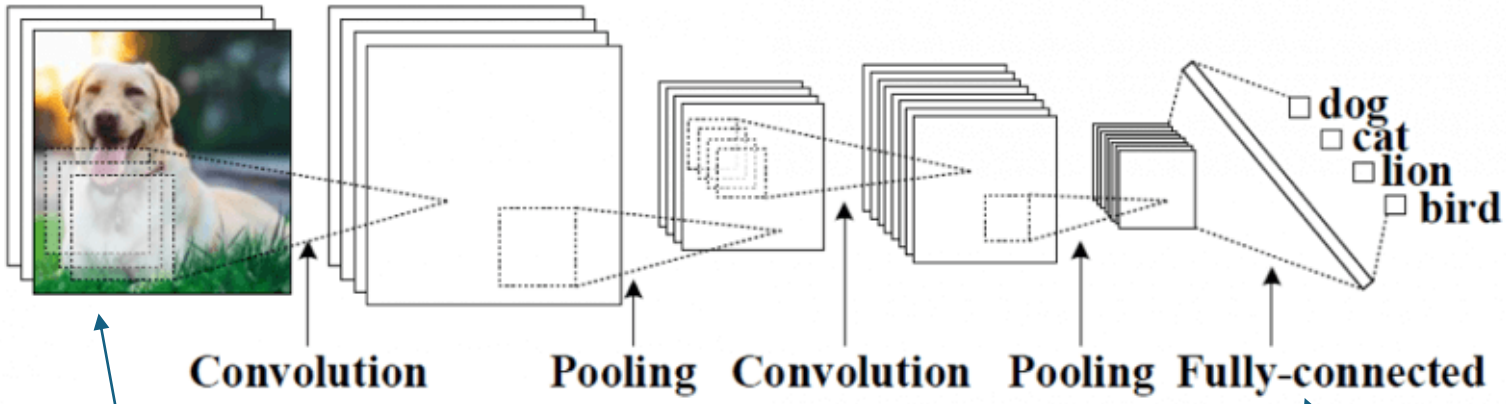
(convolutions can detect same translated motif across entire image, but not rotated or scaled versions of it)



source

Putting It All Together

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

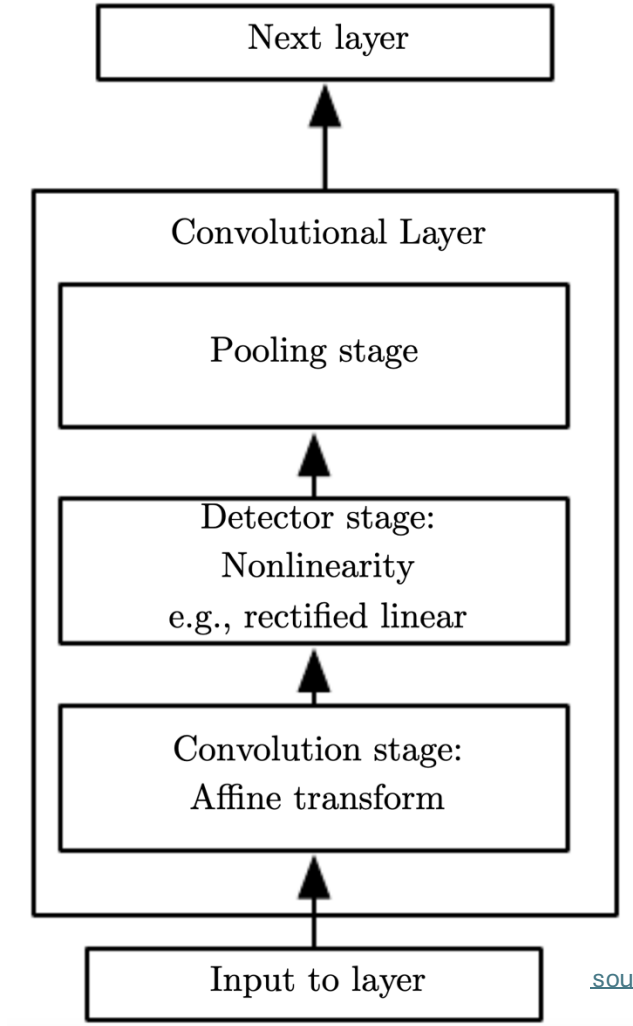


many images (training examples), potentially with several channels

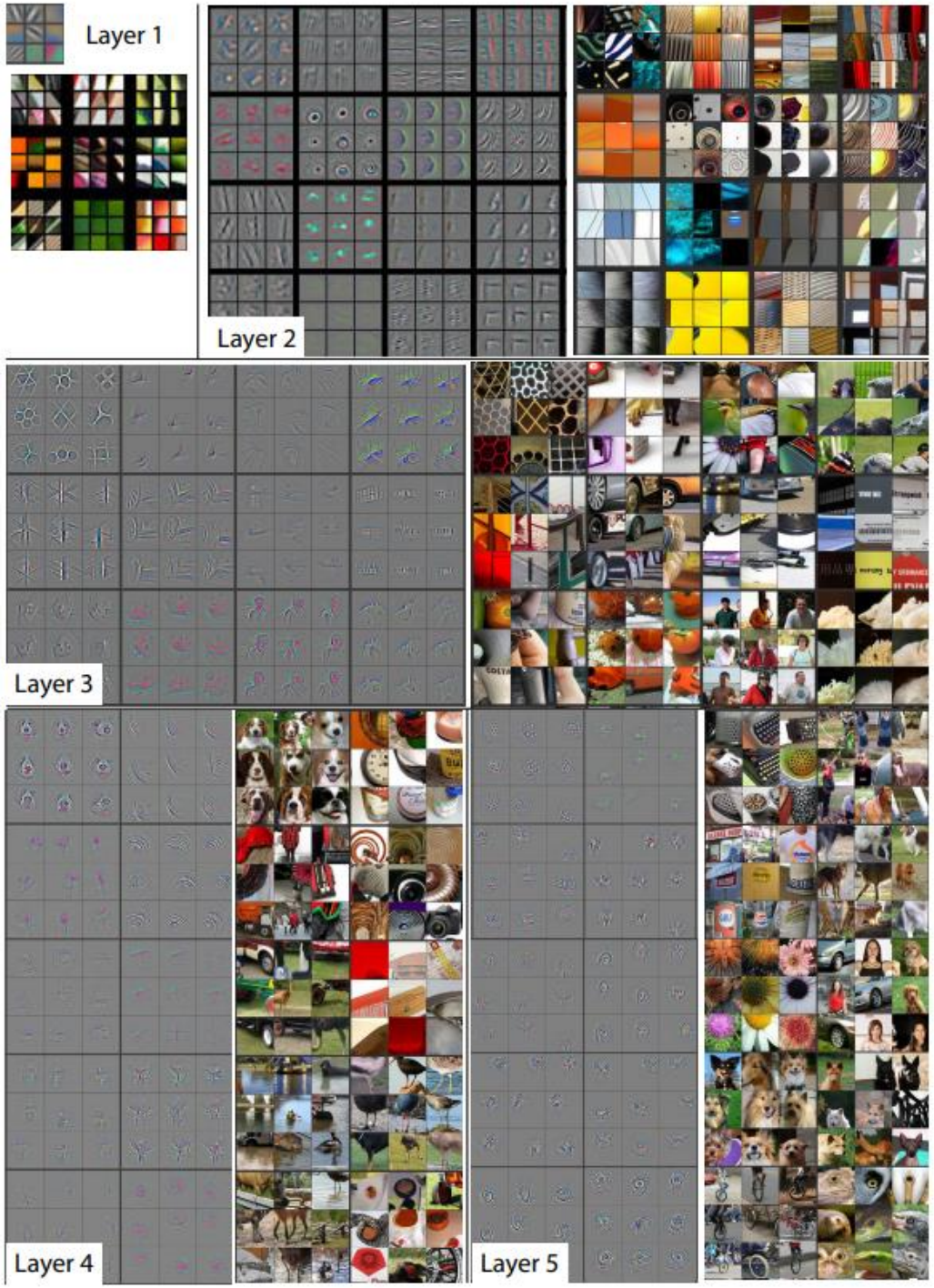
several kernels producing several feature maps

downsampling

flatten dimensions for final classification



source



Hierarchical Learning

shown here: top 9 activations of some random feature maps on test images, together with corresponding image patches
(projected down to pixel space using deconvolutional network)

some insights:

- strong groupings in feature maps, with exaggeration of discriminative image parts
- more global activations at higher layers
- greater invariance at higher layers

source

t-SNE Visualization

approach:
for every image, embed high-dimensional vector of last hidden layer's activations (right before final fully-connected classification layer) into two-dimensional vector (while preserving distances)

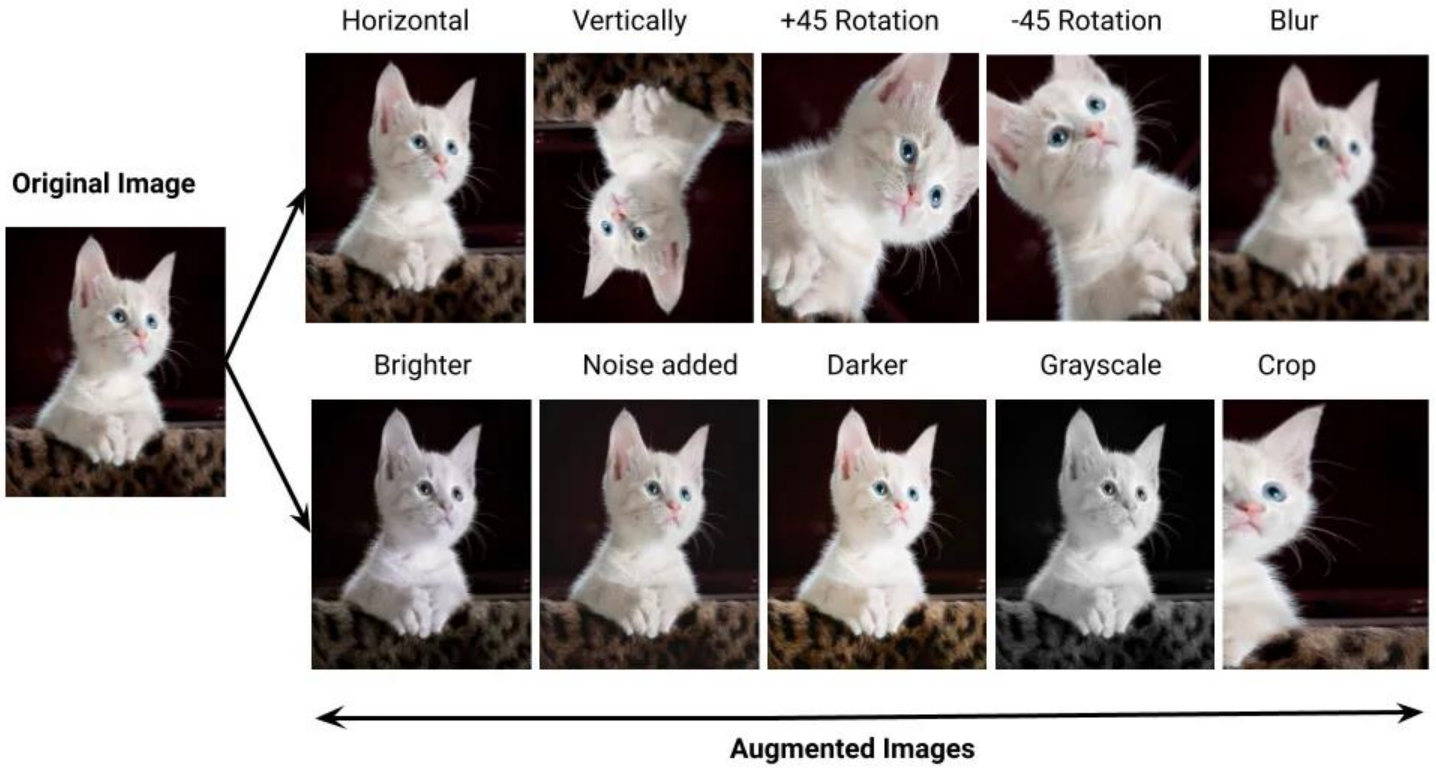
→ semantic similarities

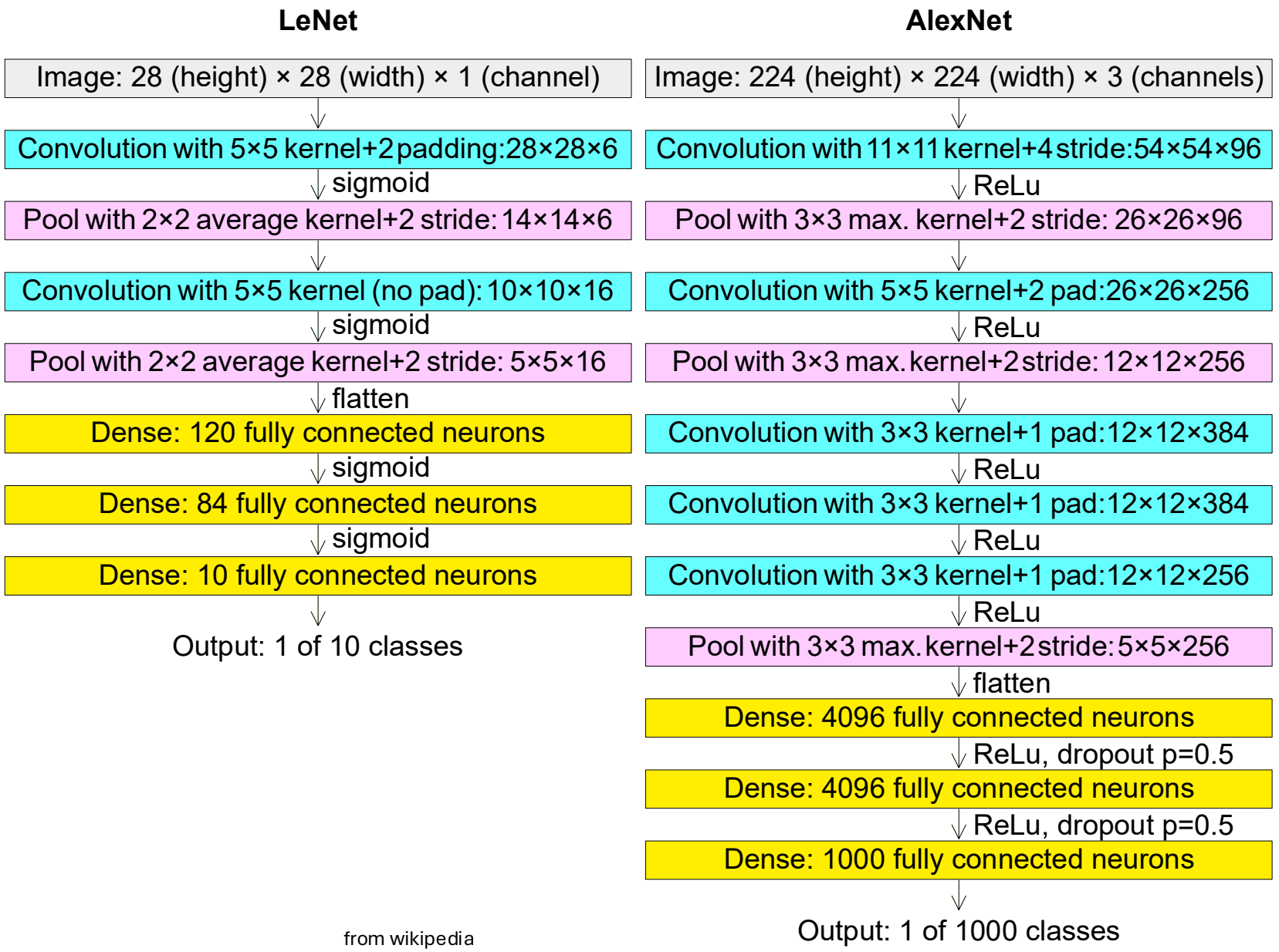
source

Data Augmentation

adding different variations of input data to training

idea: supporting the model in maintaining desired invariances





from wikipedia

AlexNet finally started the deep learning hype.

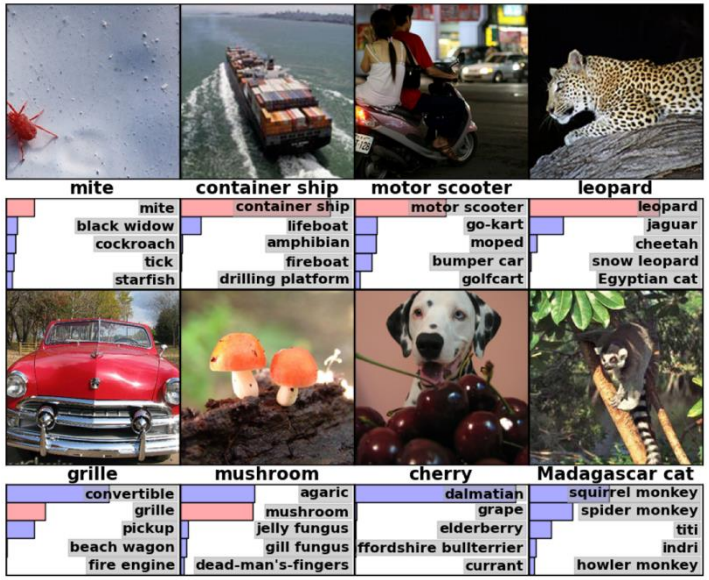
(winning the ImageNet challenge in 2012)

Rise of Deep Learning

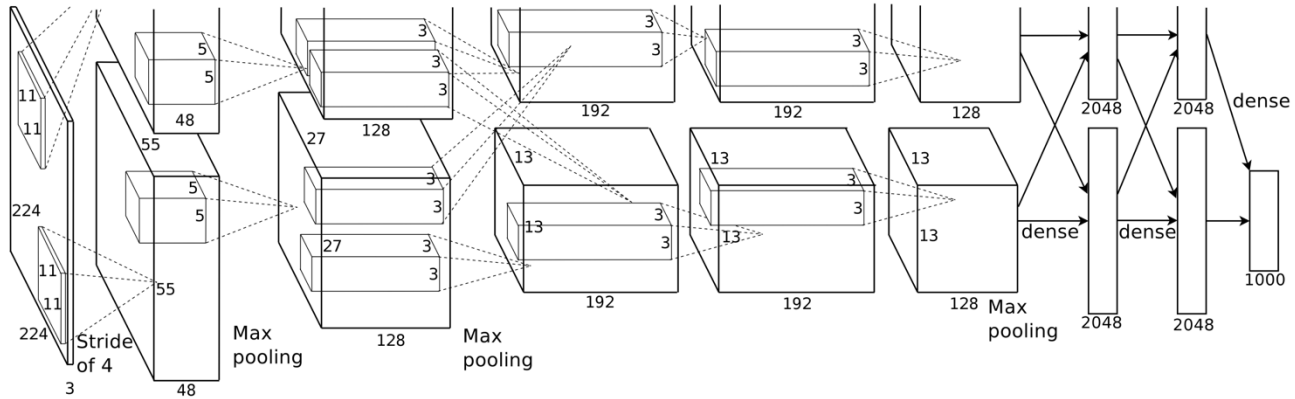
a little bit oversimplified:

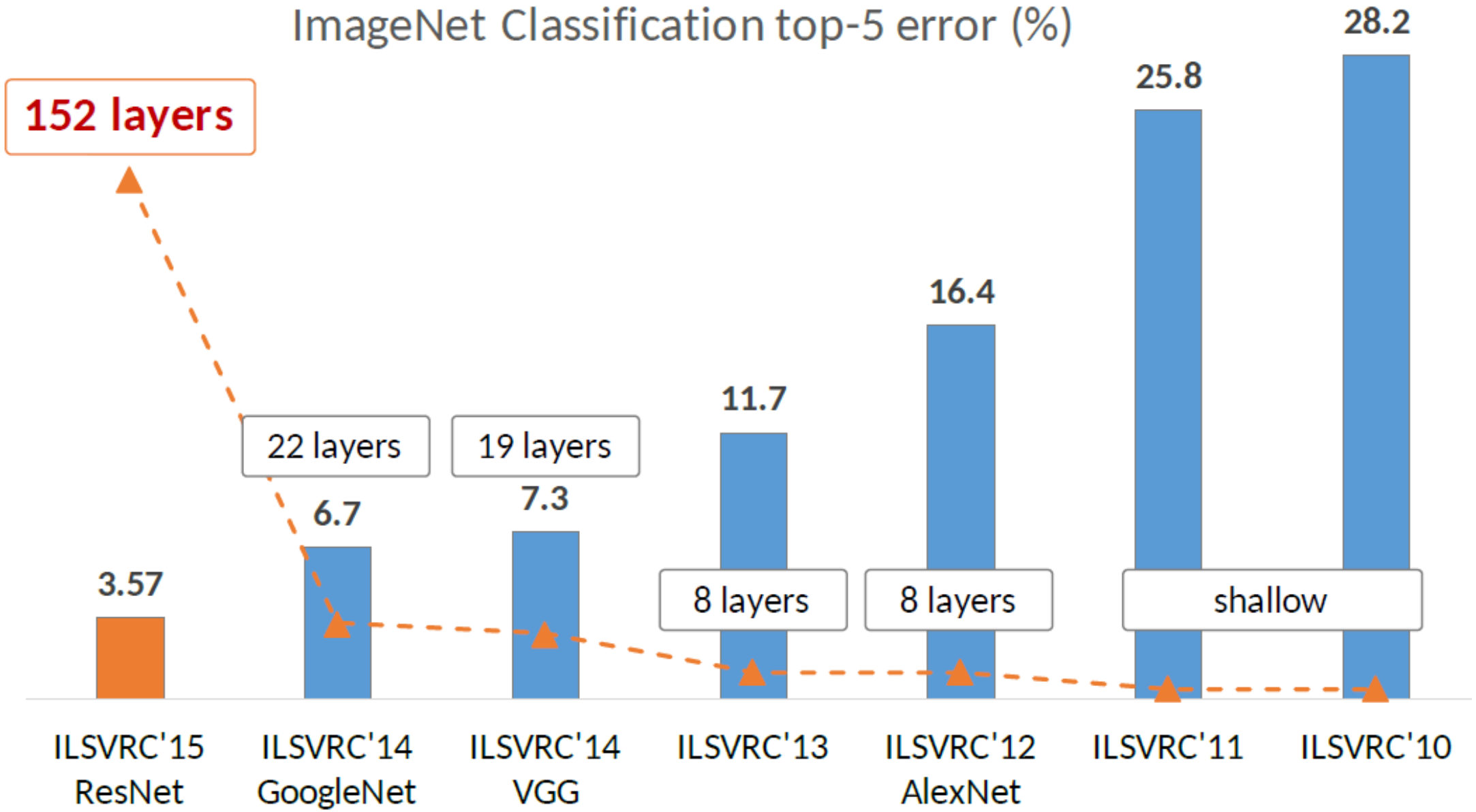
deep learning = lots of training data + parallel computation + smart algorithms

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

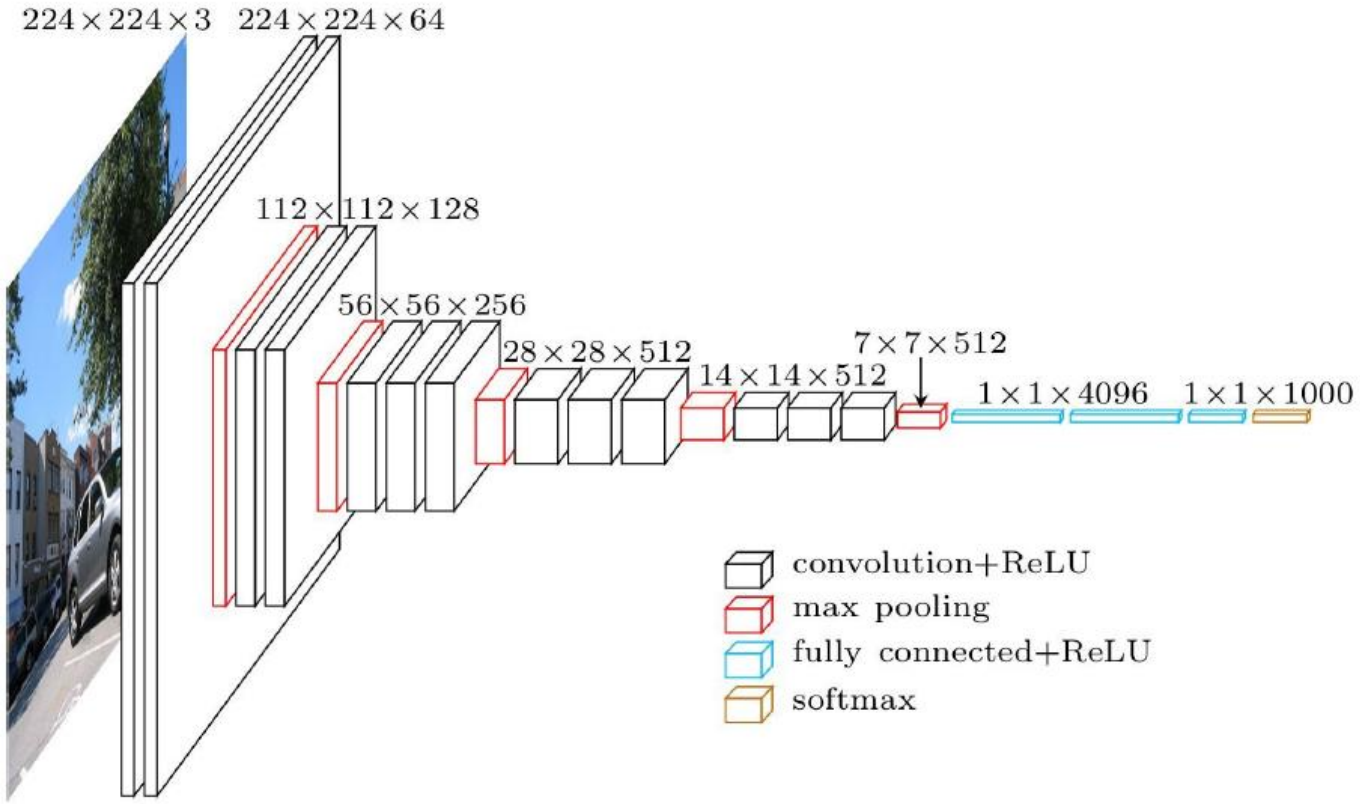


source

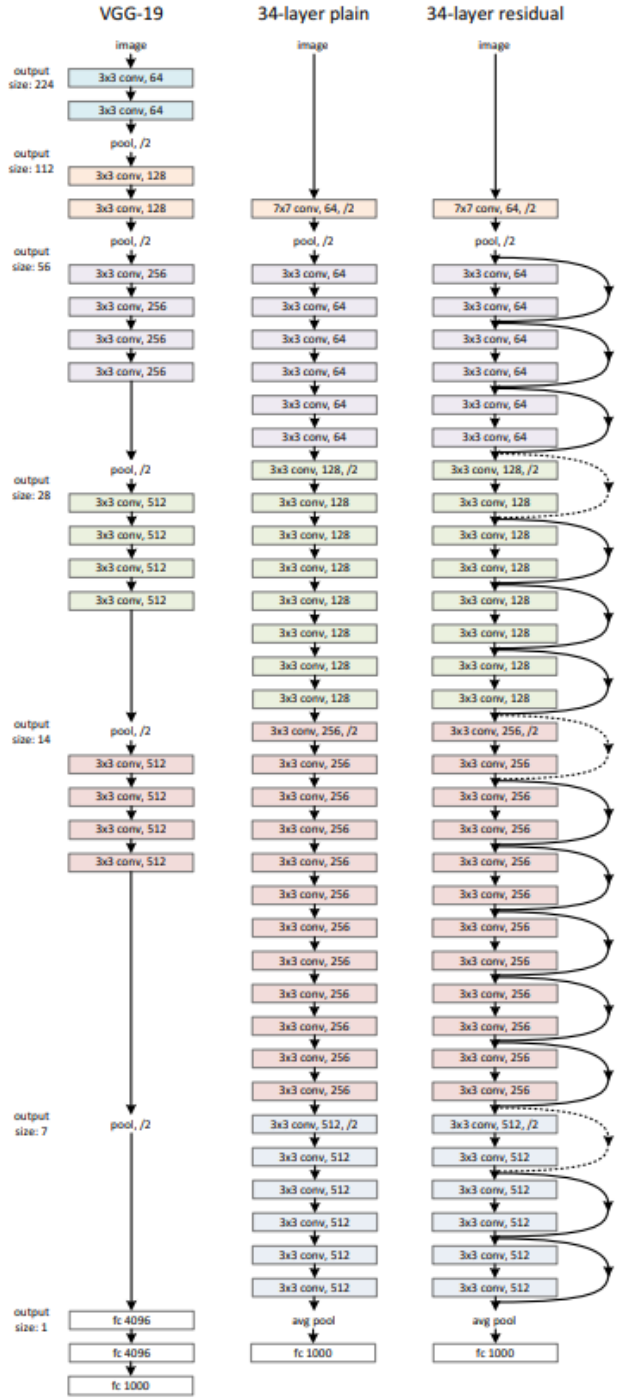



ImageNet Classification top-5 error (%)
152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10

VGG


224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax

ResNet


VGG-19
34-layer plain
34-layer residual
image
output size: 224
3x3 conv, 64
pool, /2
output size: 112
3x3 conv, 128
output size: 56
3x3 conv, 256
output size: 28
3x3 conv, 512
output size: 14
output size: 7
fc 4096
fc 1000
7x7 conv, 64, /2
3x3 conv, 128, /2
3x3 conv, 256, /2
3x3 conv, 512, /2
avg pool
output size: 1

Skip Connections

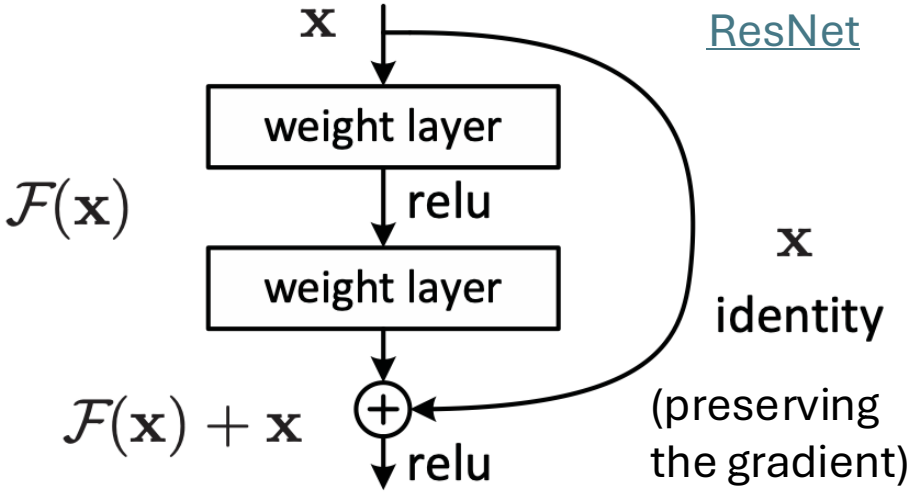
issue: degradation of training and test errors when adding more and more layers → not due to overfitting (but reason controversial)

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

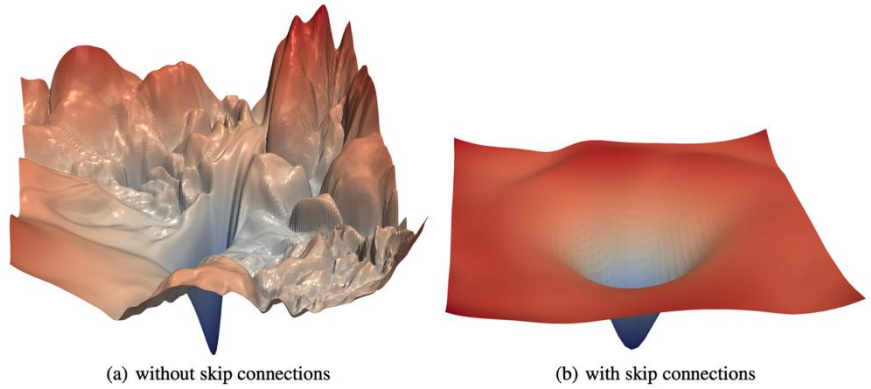
residual mapping (special kind of skip/shortcut connections):



ResNet

(preserving the gradient)

loss surface:



(a) without skip connections (b) with skip connections

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)

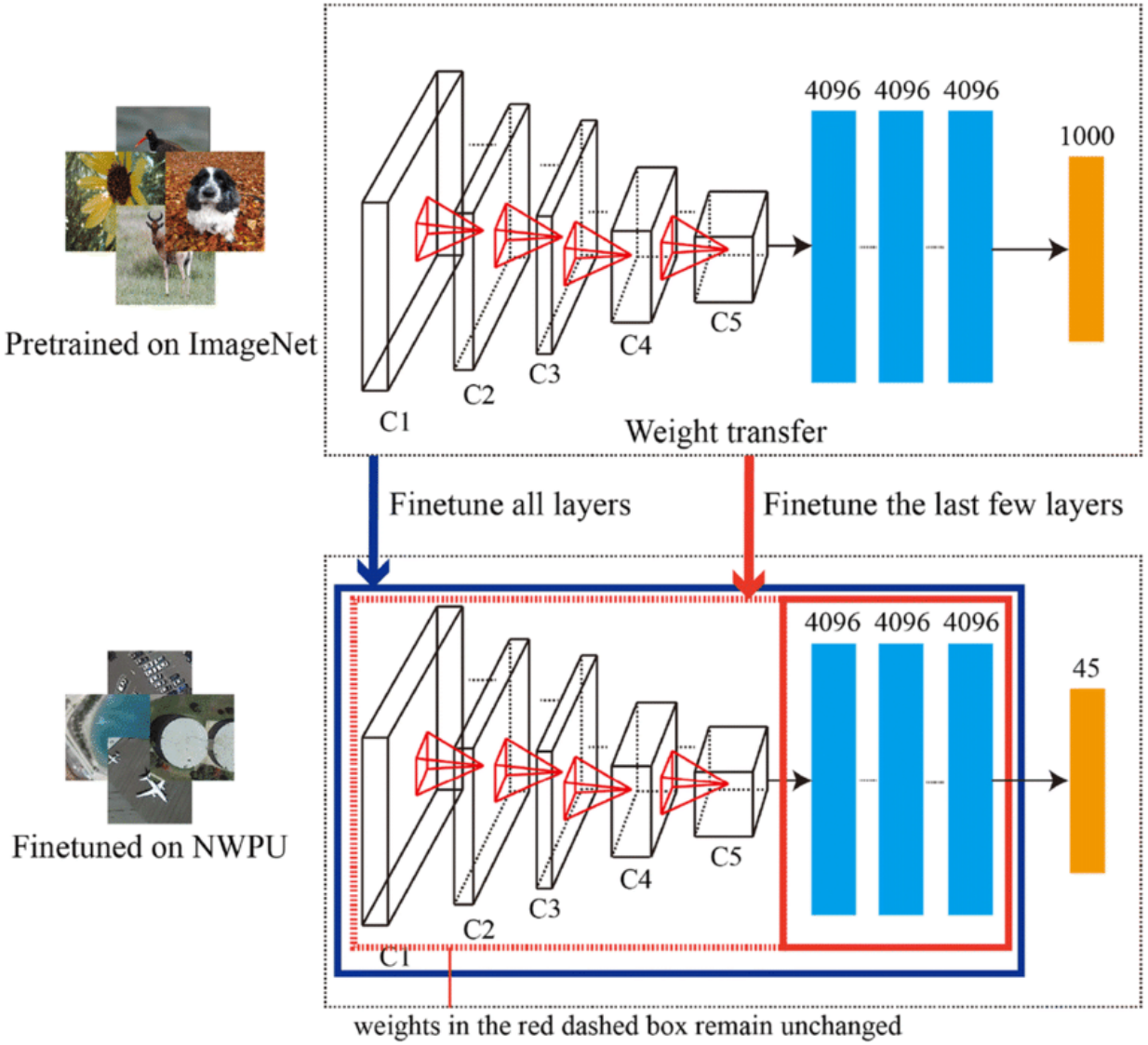
Finetuning

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



source

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

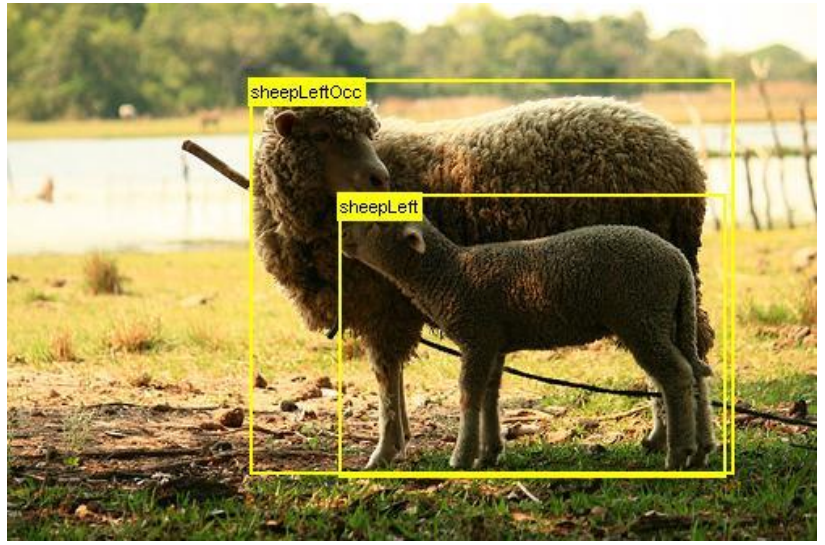
Different Image Classification Techniques

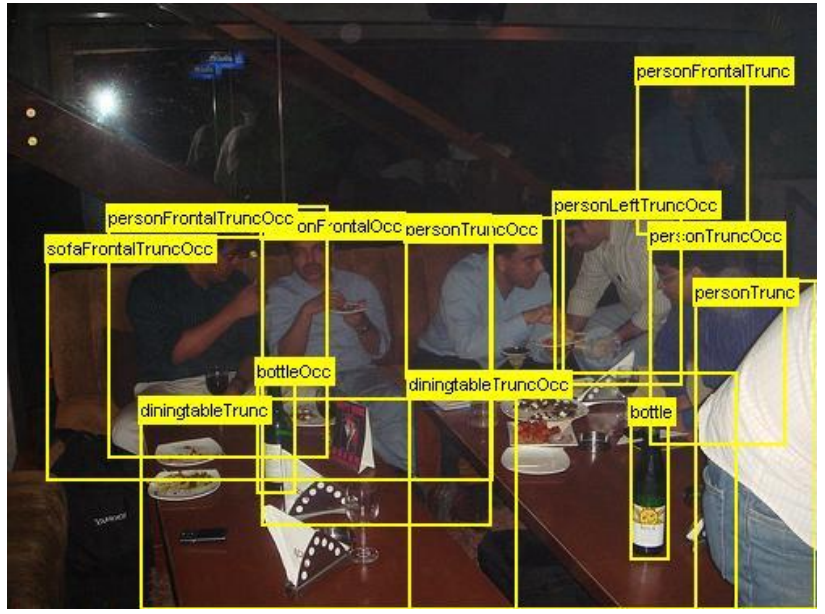
- Hough transform: looking for predefined shapes such as lines or circles (feature-based recognition)
- feature description & matching: identify specific object instances (typically used for instance rather than class recognition)
- image features as input to classic ML method: generalization from data
- 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

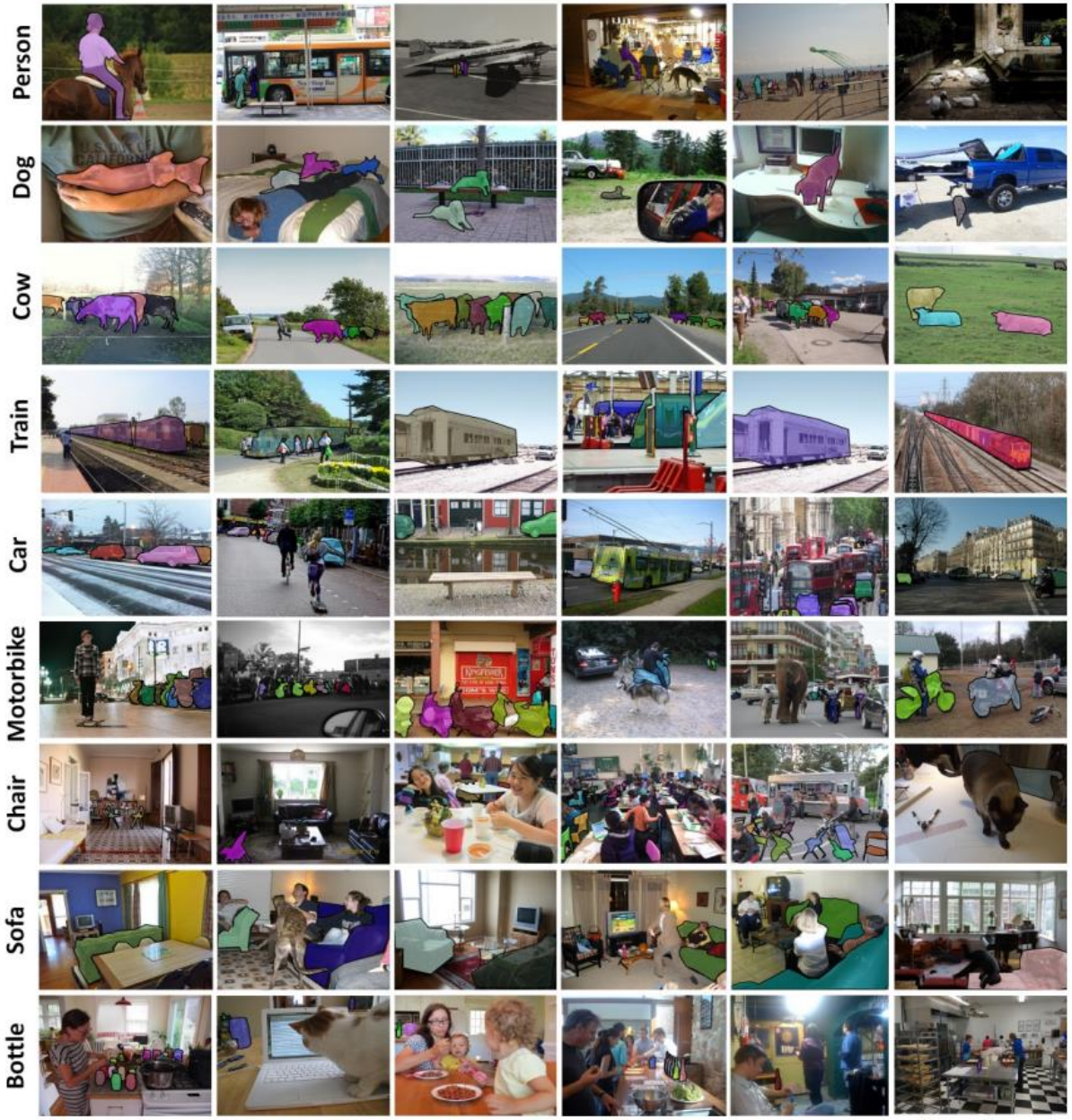
- PASCAL Visual Object Class challenge
- widely used as benchmark for object detection and semantic segmentation
- 20 object categories such as person, sofa, sheep, car, ...
- 11530 annotated images
- available annotations: pixel-level segmentation, bounding boxes, object classes





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



Object Segmentation from DINO

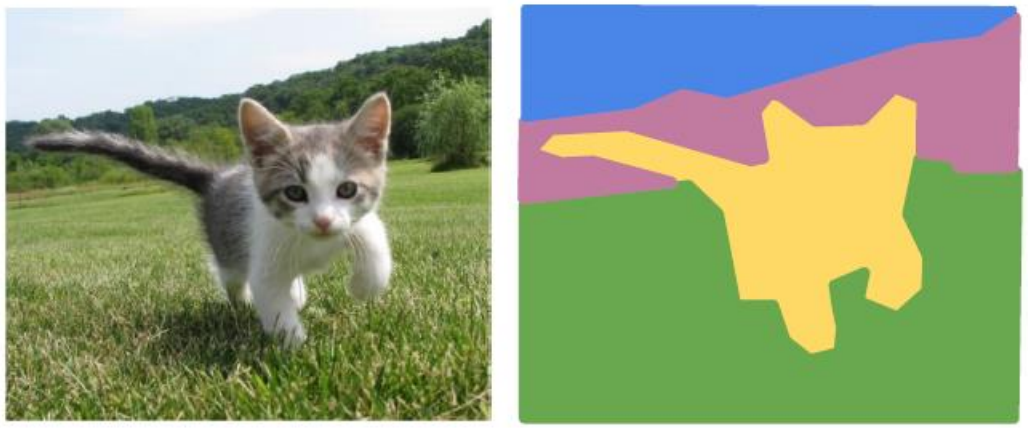
thresholding self-attention map of last layer:

source

not a full segmentation mask though …

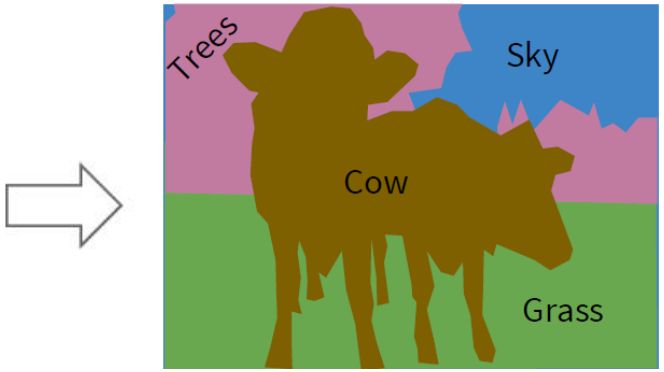
Classification of Each Pixel

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

Input:
3 x H x W

Conv → Conv → Conv → Conv → argmax

stride 1, padding
→ small kernels

feature maps → Convolutions:
D x H x W

Scores:
C x H x W

categories

Predictions:
H x W

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)

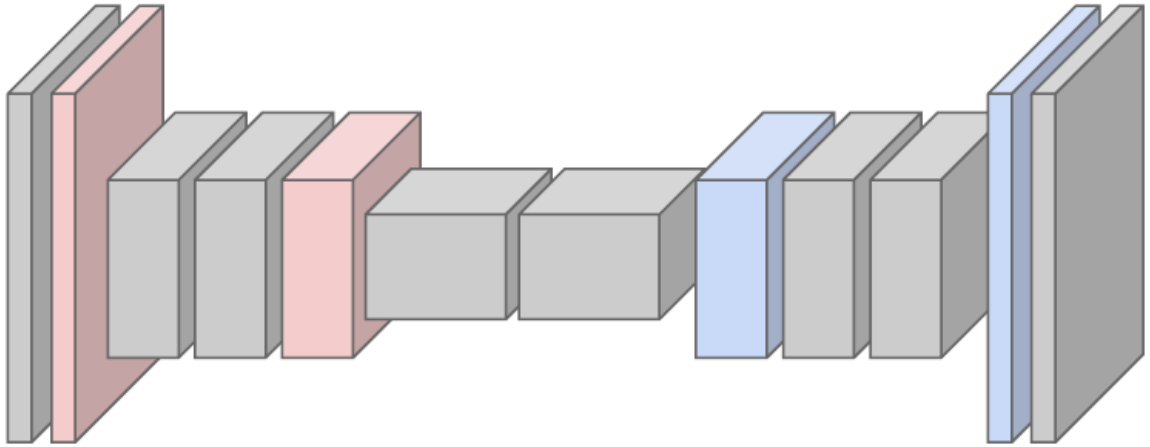
but no downsampling means small receptive field and no hierarchical learning

Upsampling to the Rescue

typical spatial feature extraction: convolution and pooling layers → downsampling

upsampling layers to restore original image size

Input:
3 x H x W



C x H x W

pixel-wise classification

Predictions:
H x W

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

Reverse Pooling

resampling (no learned parameters)

Nearest Neighbor

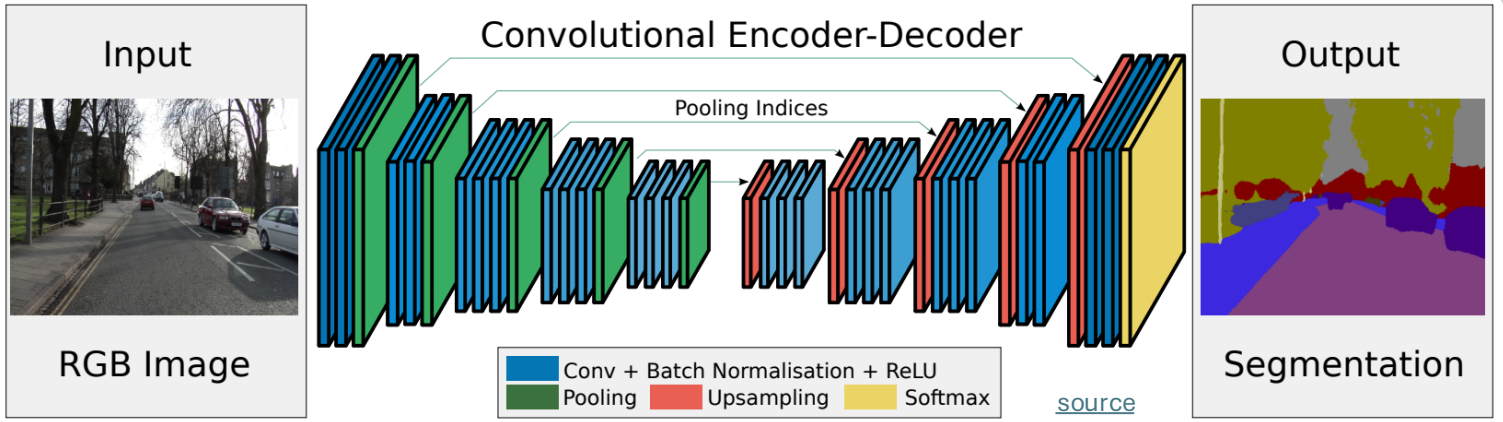
1	2
3	4

→

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

Input: 2 x 2 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

→

5	6
7	8

→ • • • → Rest of the network

Input: 4 x 4 Output: 2 x 2

Max Unpooling
Use positions from pooling layer

1	2
3	4

→

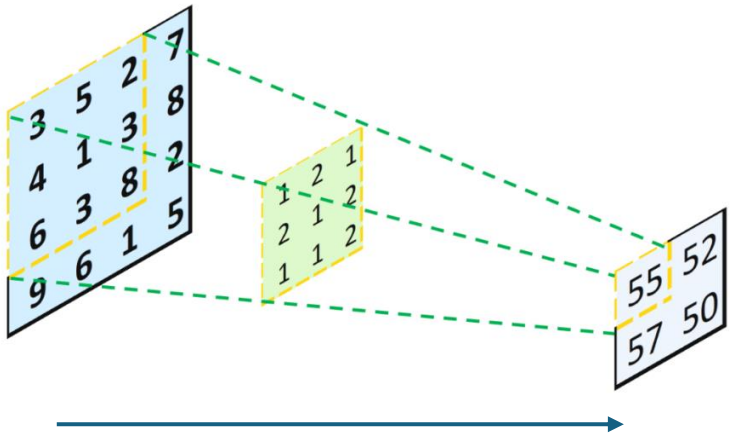
0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

Input: 2 x 2 Output: 4 x 4

filled with learned parameters in subsequent convolution

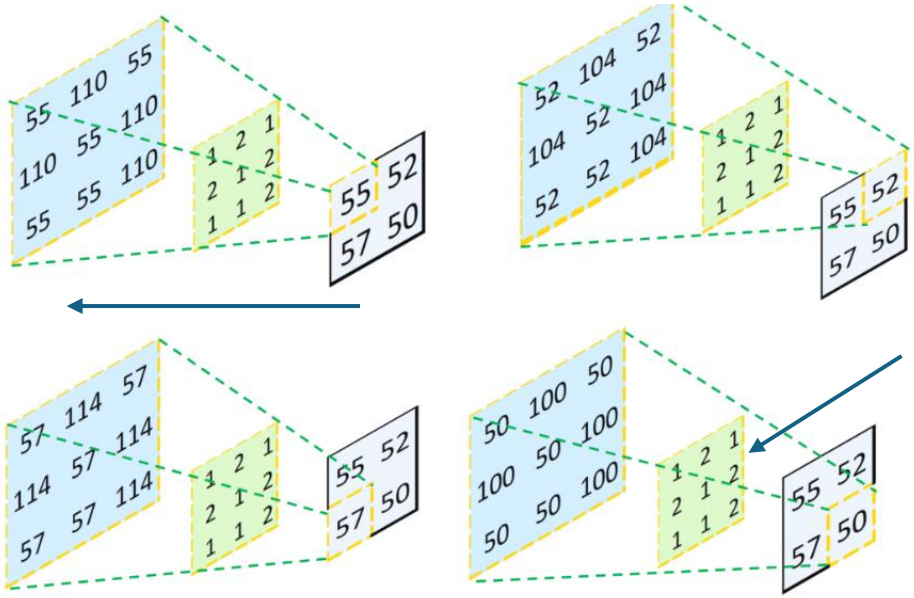
Reverse Convolution

convolution

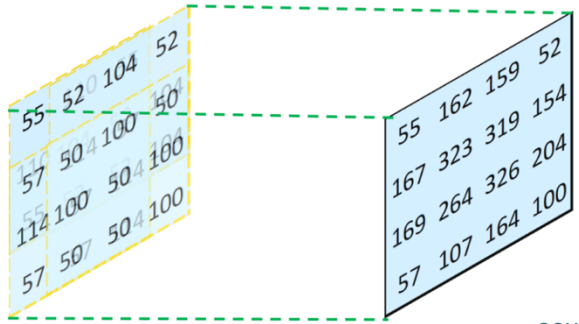


not inverse convolution
→ additional learned parameters

transposed convolution

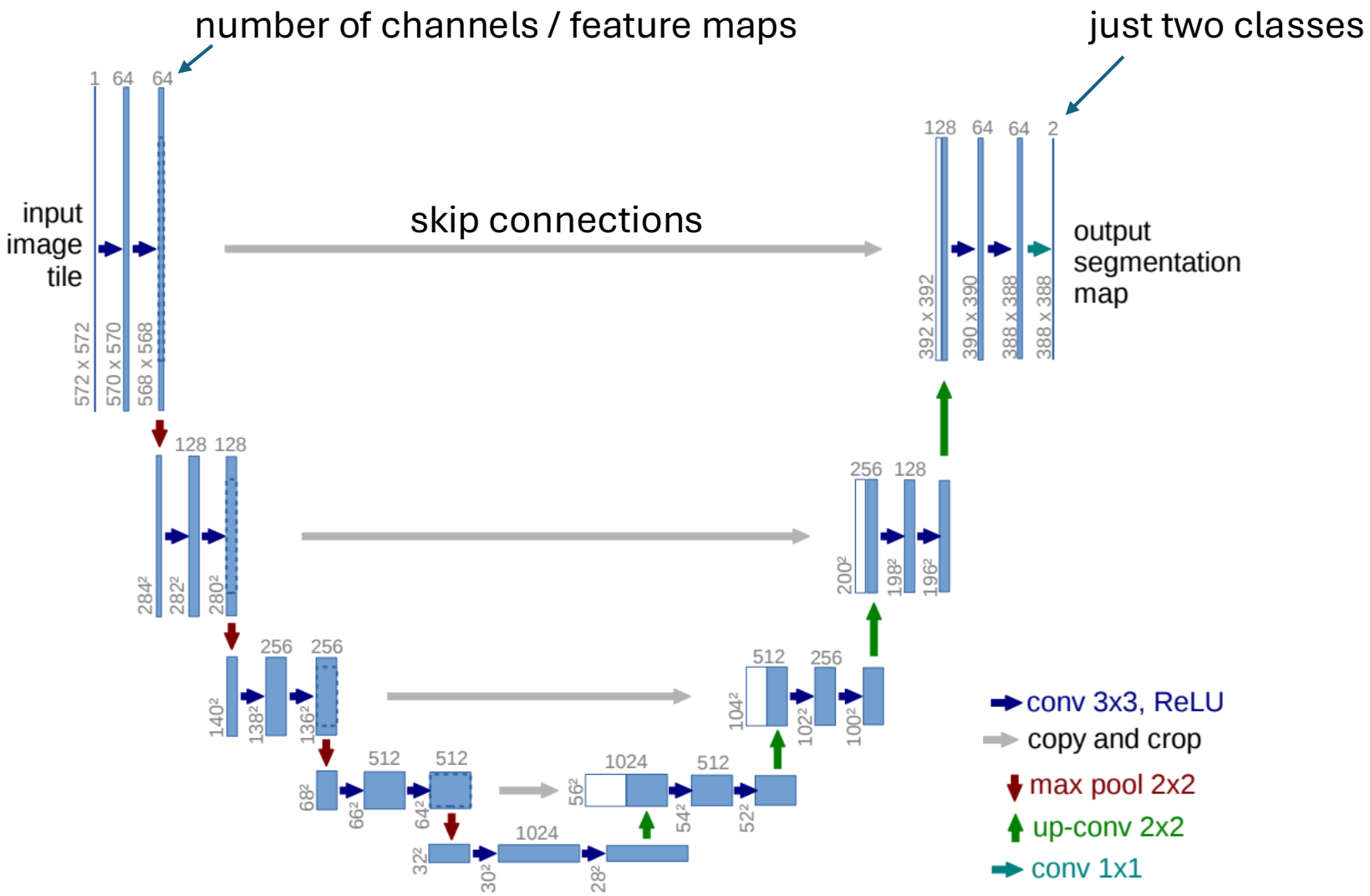


combine and sum overlaps:



source

U-Net



also used for learning of depth maps, image synthesis (diffusion), ...

Aside: Autoencoders

Autoencoder

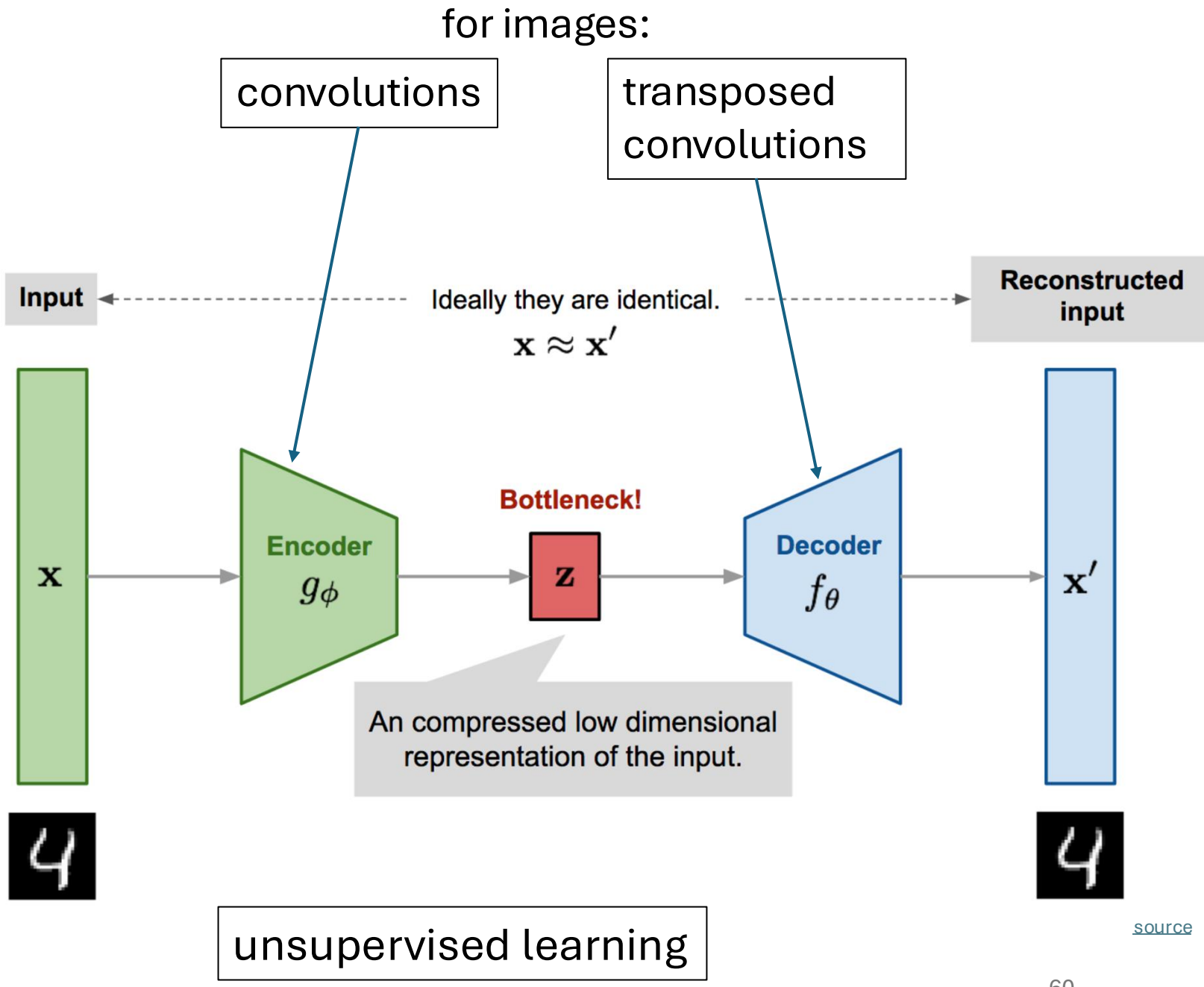
(deep) encoder network

(deep) decoder network

learned together by minimizing differences between original input and reconstructed input (expressed as losses)

compressed intermediate representation: dimensionality reduction

(alternative to PCA)



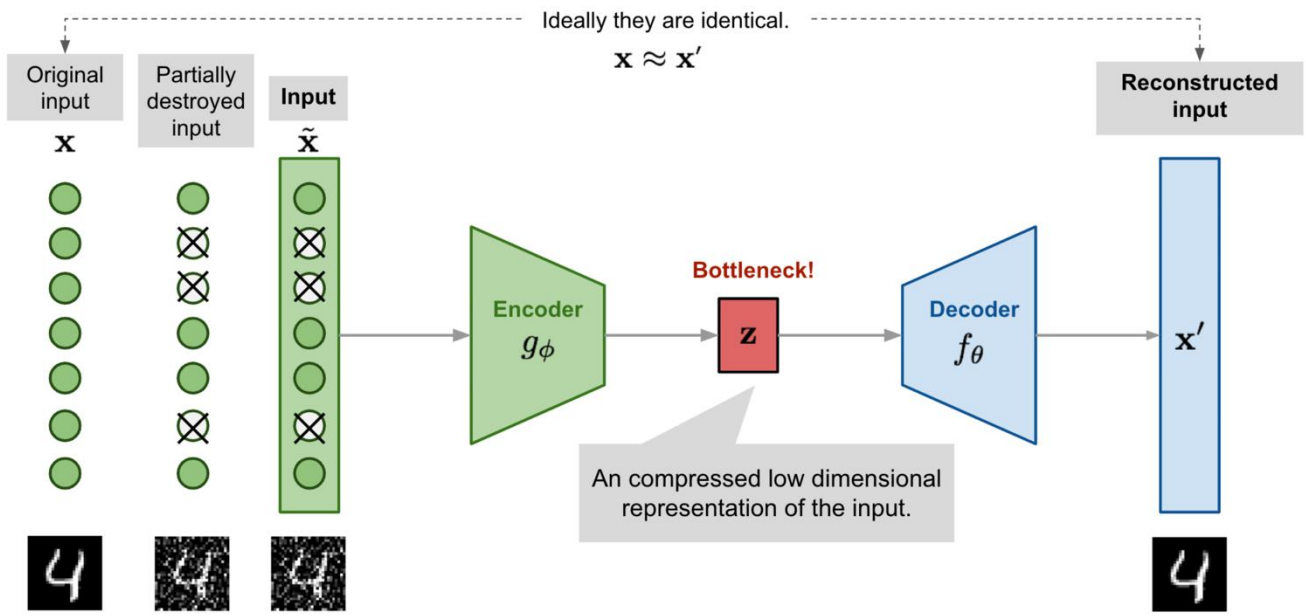
source

Denoising Autoencoder

goal: avoid overfitting and improve robustness of plain autoencoder

learn to remove noise of distorted input $\tilde{x}$ → restore original input x

similar to dropout

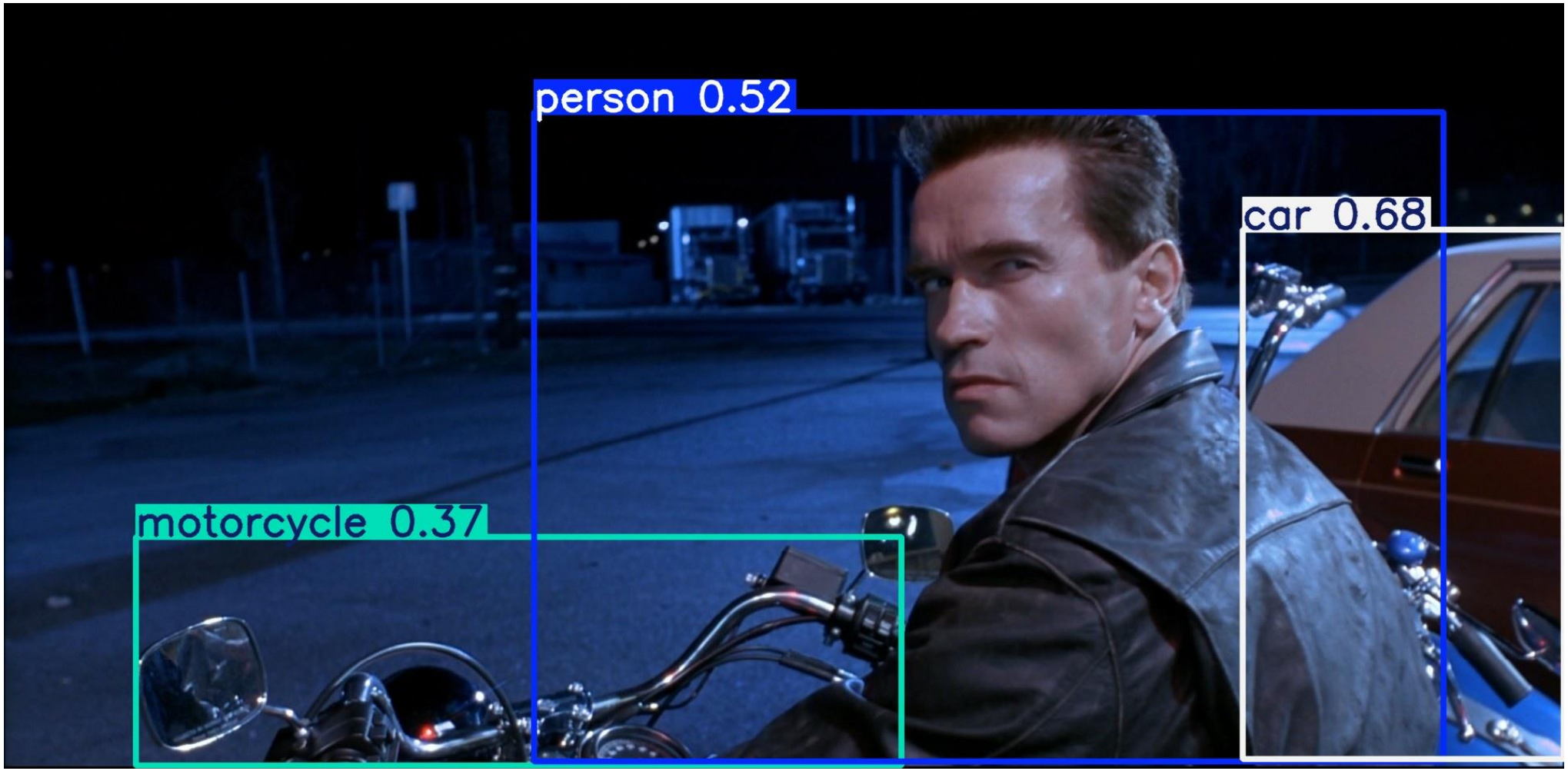


source

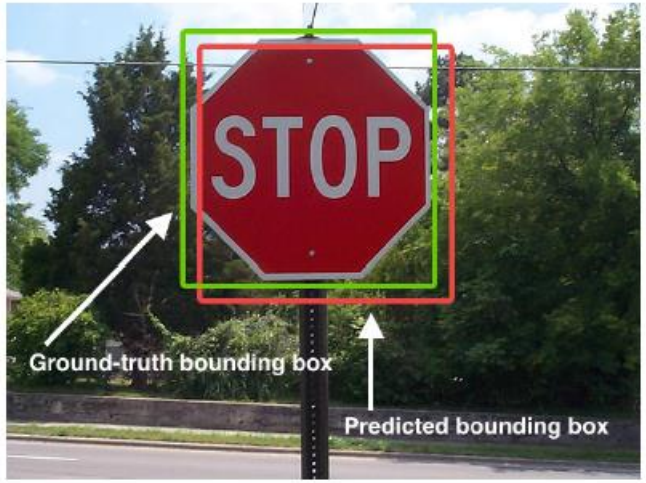
alternative to deconvolution (image restoration)

Object Detection

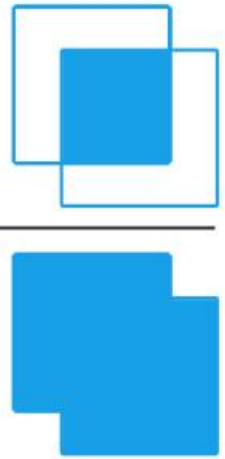
output: bounding boxes with category label and confidence



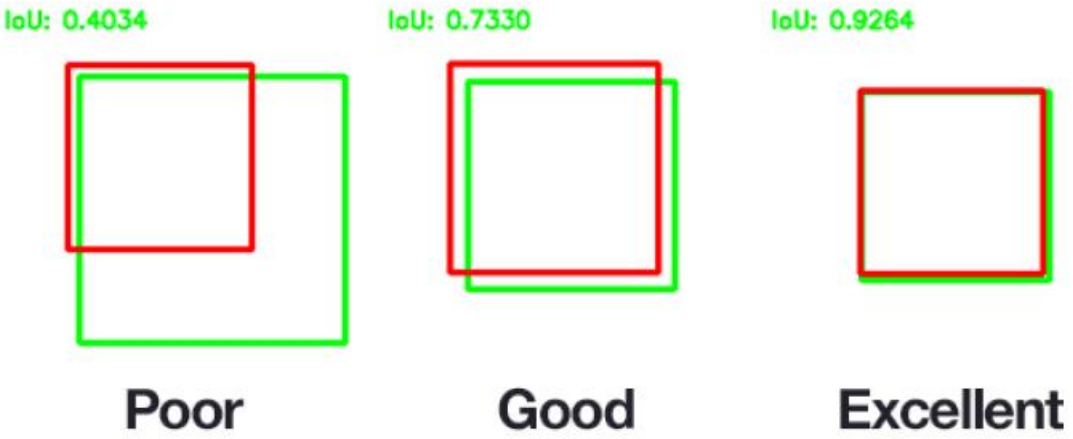
Performance Evaluation of Localization



$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



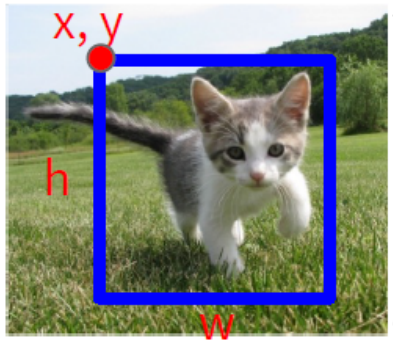
Intersection-over-Union

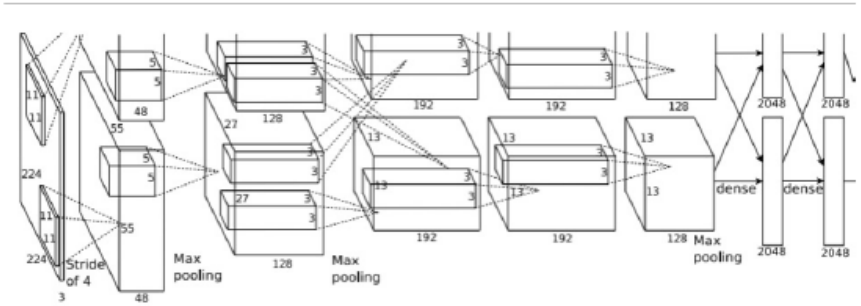


images with multiple objects: number of detections dependent on used confidence threshold

Object Detection: Single Object

(Classification + Localization)





Correct label: Cat

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Softmax Loss

Fully Connected: 4096 to 1000

Multitask Loss

+ → Loss

Vector: 4096

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

L2 Loss

Correct box: (x', y', w', h')

Treat localization as a regression problem!

too complicated for multiple objects

Localization for Multiple-Object Images

idea: classify many different crops of the image as object or background

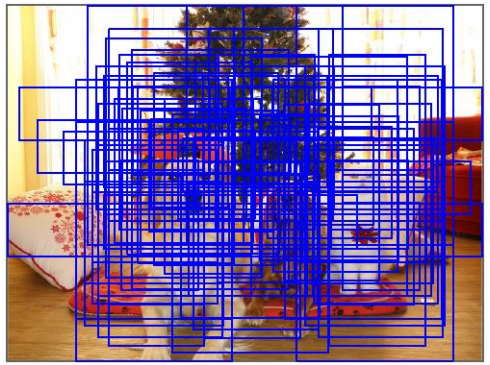
(crops: sliding window over image, iterated at multiple window sizes)

Dog? No
Cat? No
Background? Yes

Dog? Yes
Cat? No
Background? No

Dog? Yes
Cat? No
Background? No

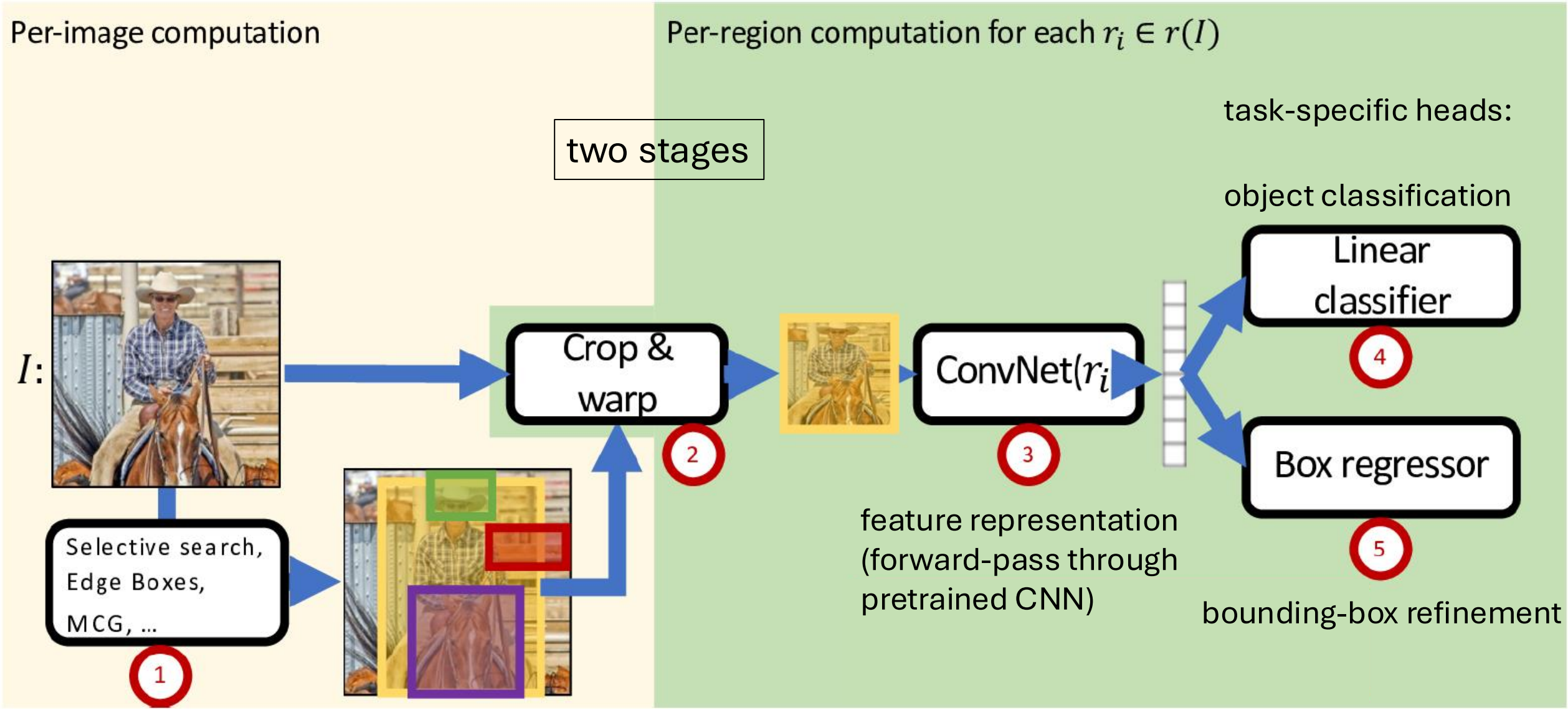
Dog? No
Cat? Yes
Background? No



issue: too many possible crops

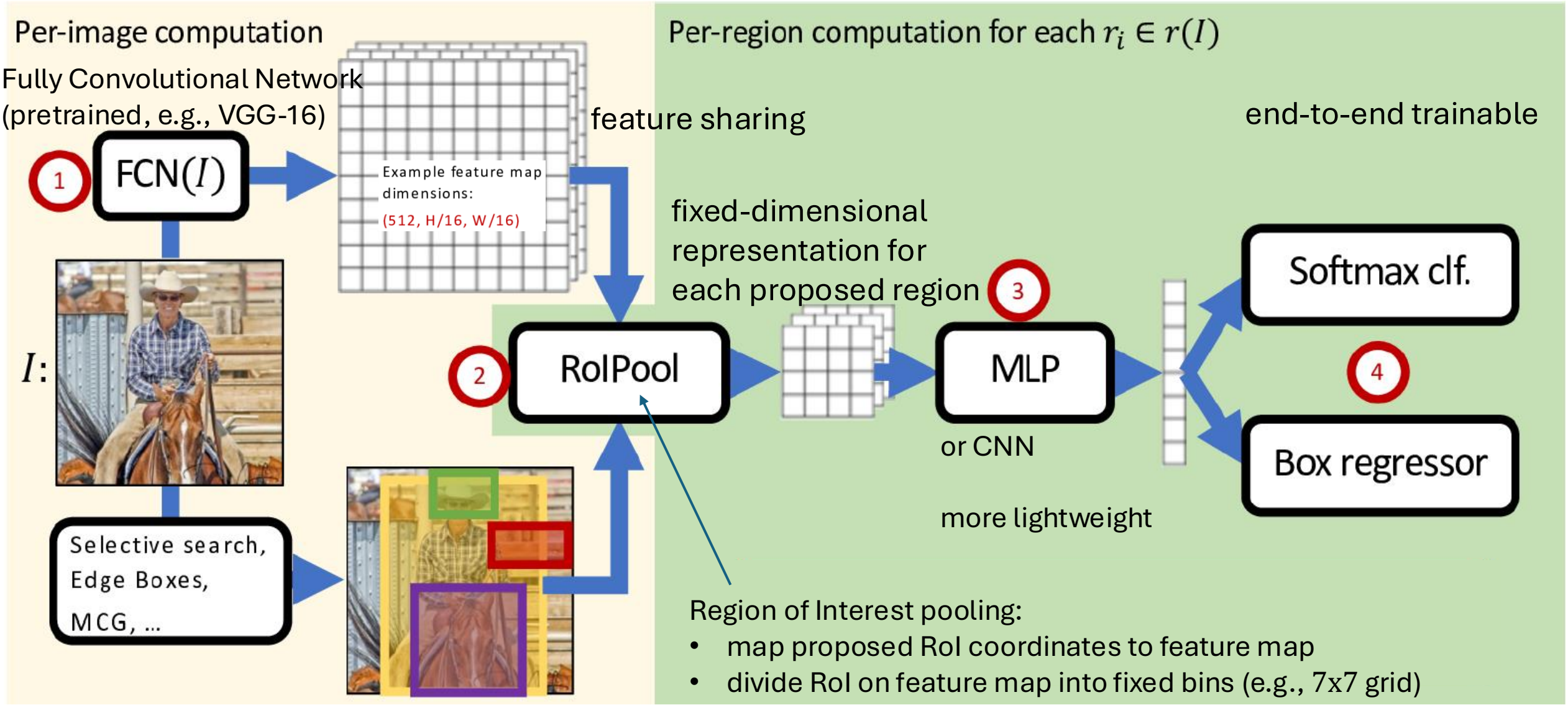
→ need for region proposals

Region-Based Convolutional Neural Network (R-CNN)

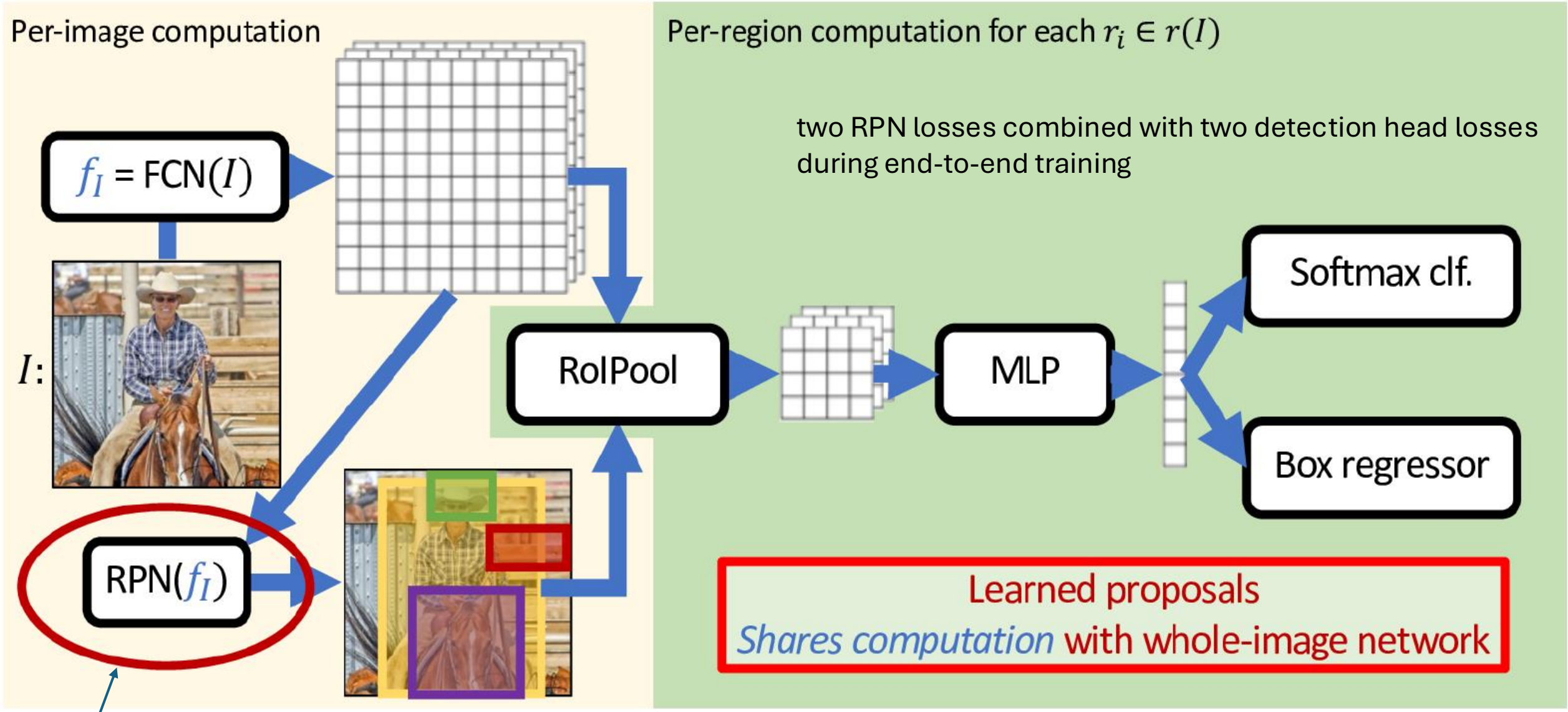


region proposals by "classic" method (~2k)

Fast R-CNN: Crop ConvNet Features



Faster R-CNN: Use Region Proposal Network



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

Feature Pyramid Network (FPN)

different intermediate layers of same backbone CNN

Per-image computation

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

high-resolution features improve detection of small objects

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

I:

RPN(f_I)

RoIPool

MLP

Softmax clf.

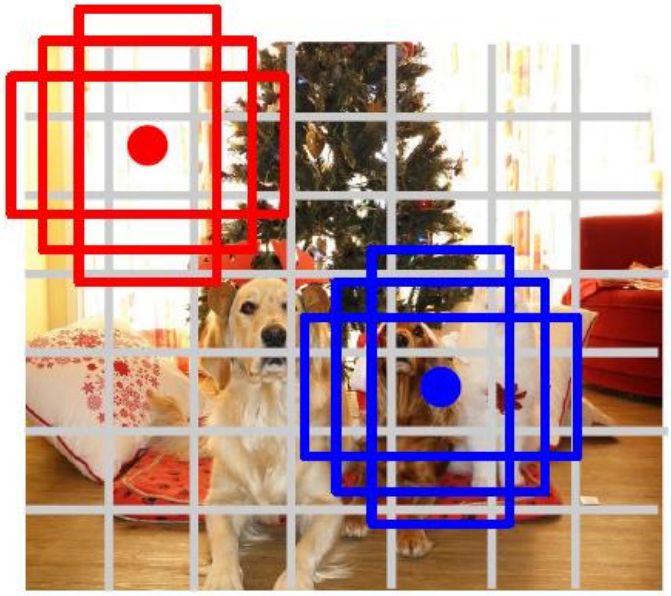
Box regressor

The whole-image feature representation can be improved by making it *multi-scale*

using different anchor sizes for different pyramid levels

Single-Stage Detectors: Drop Per-Region Computation

Input image
3 x H x W



Divide image into grid
7 x 7 ← or more

Image a set of base boxes
centered at each grid cell
Here B = 3

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!

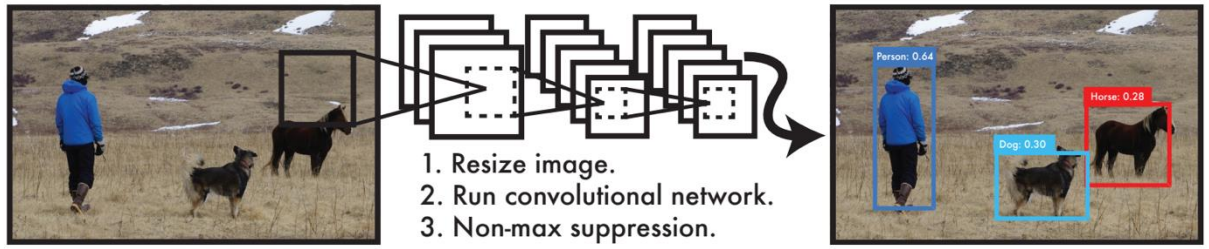
Output:
7 x 7 x (5 * B + C)

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

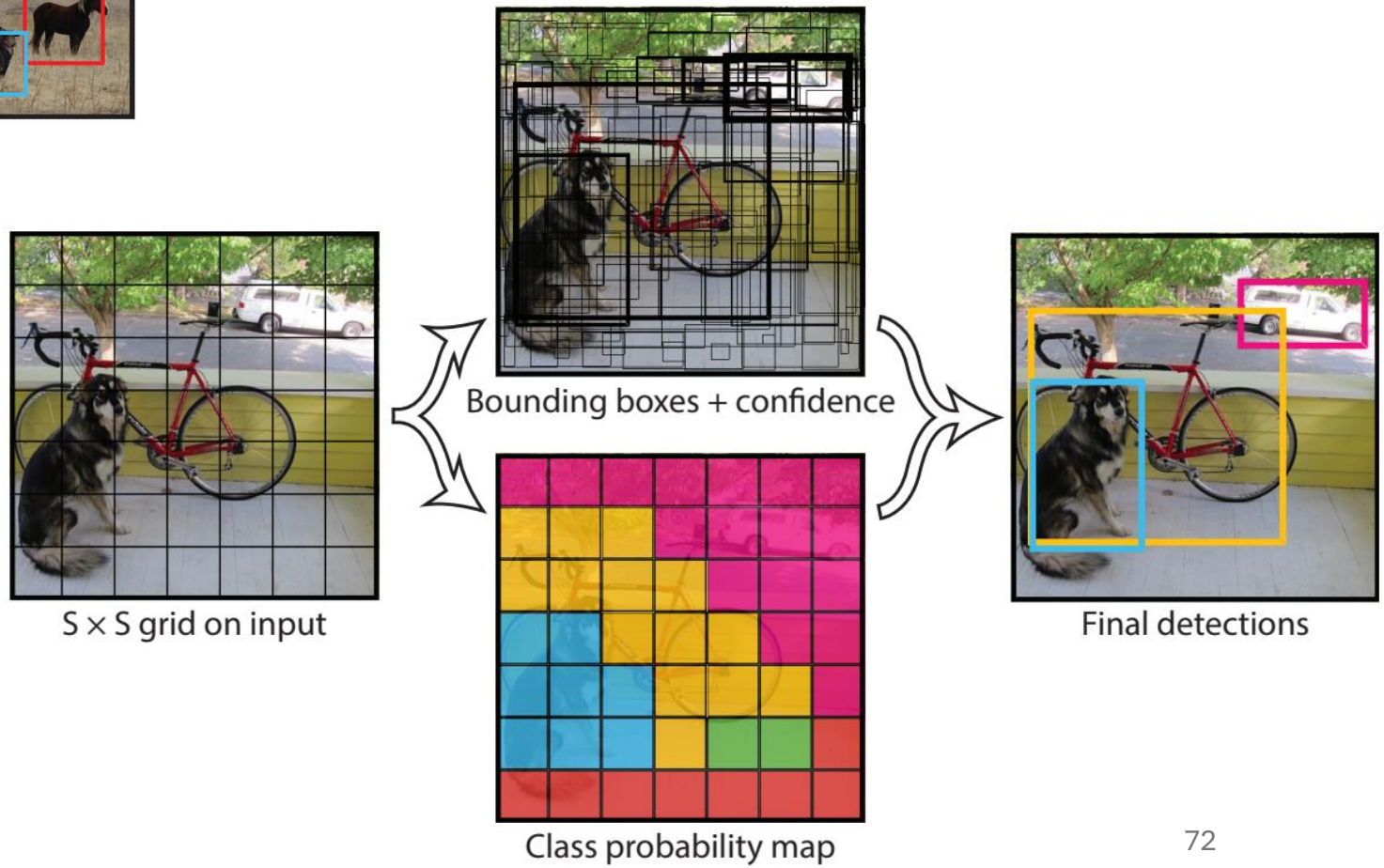
faster, but usually worse performance

YOLO: Real-Time Object Detection

anchor-free approach
in newer YOLO versions



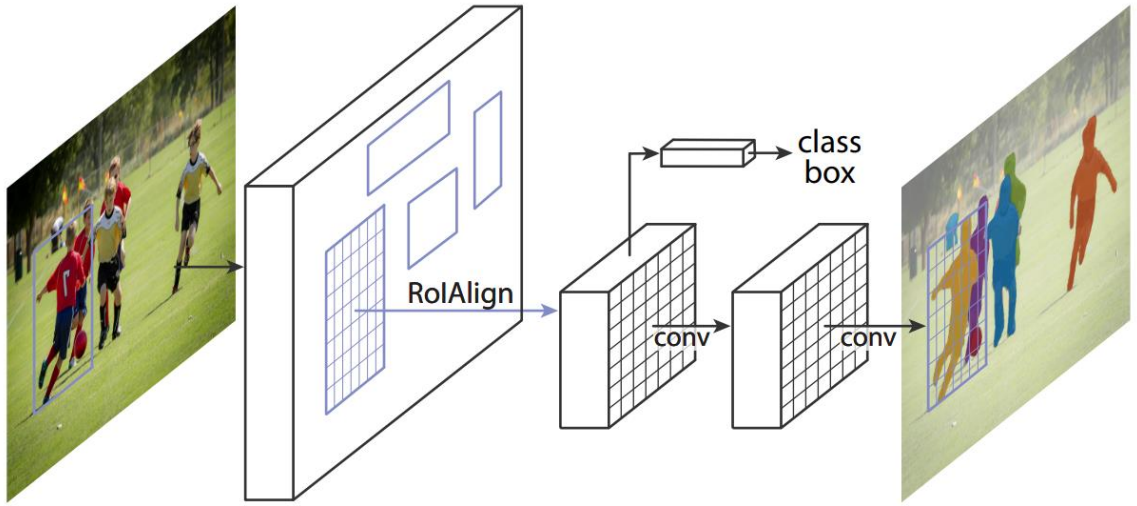
You Only Look Once:
prediction of bounding boxes
and class probabilities in one go



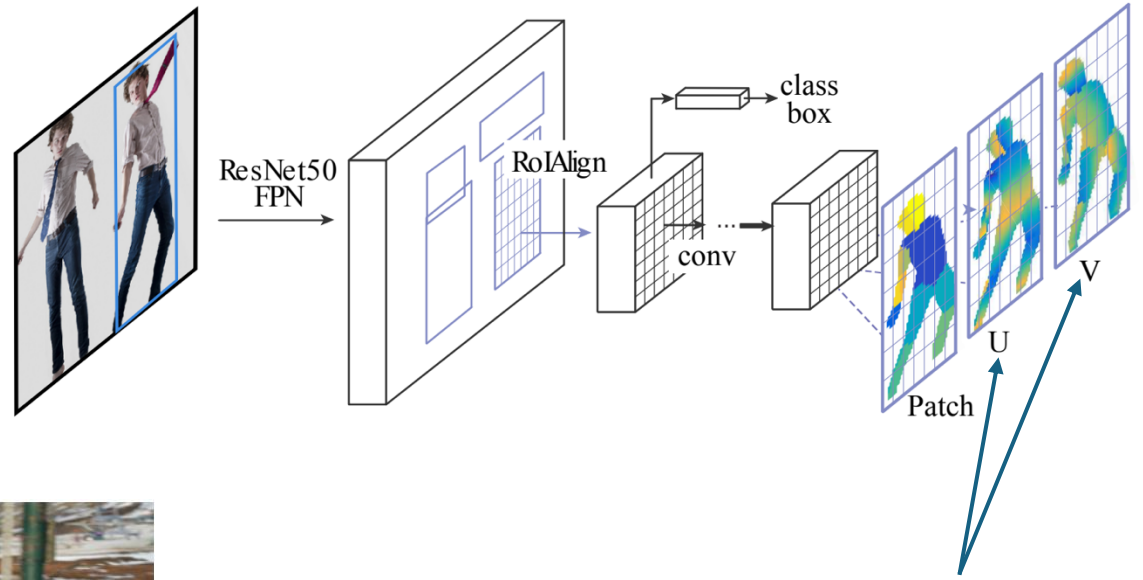
Instance Segmentation

Add Additional Network Heads to R-CNN

instance segmentation (Mask R-CNN):



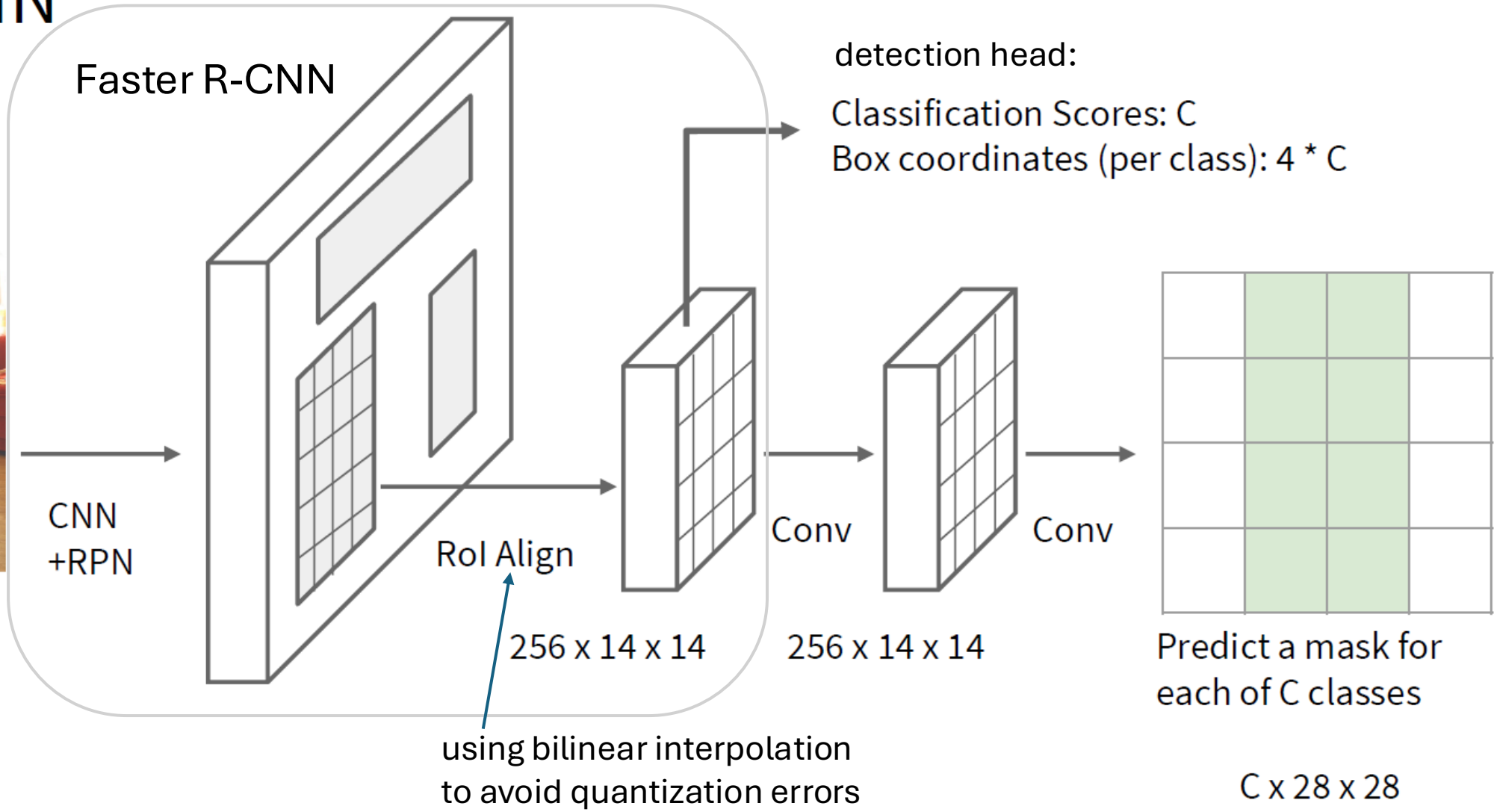
pose estimation (DensePose):



3D model's surface to 2D

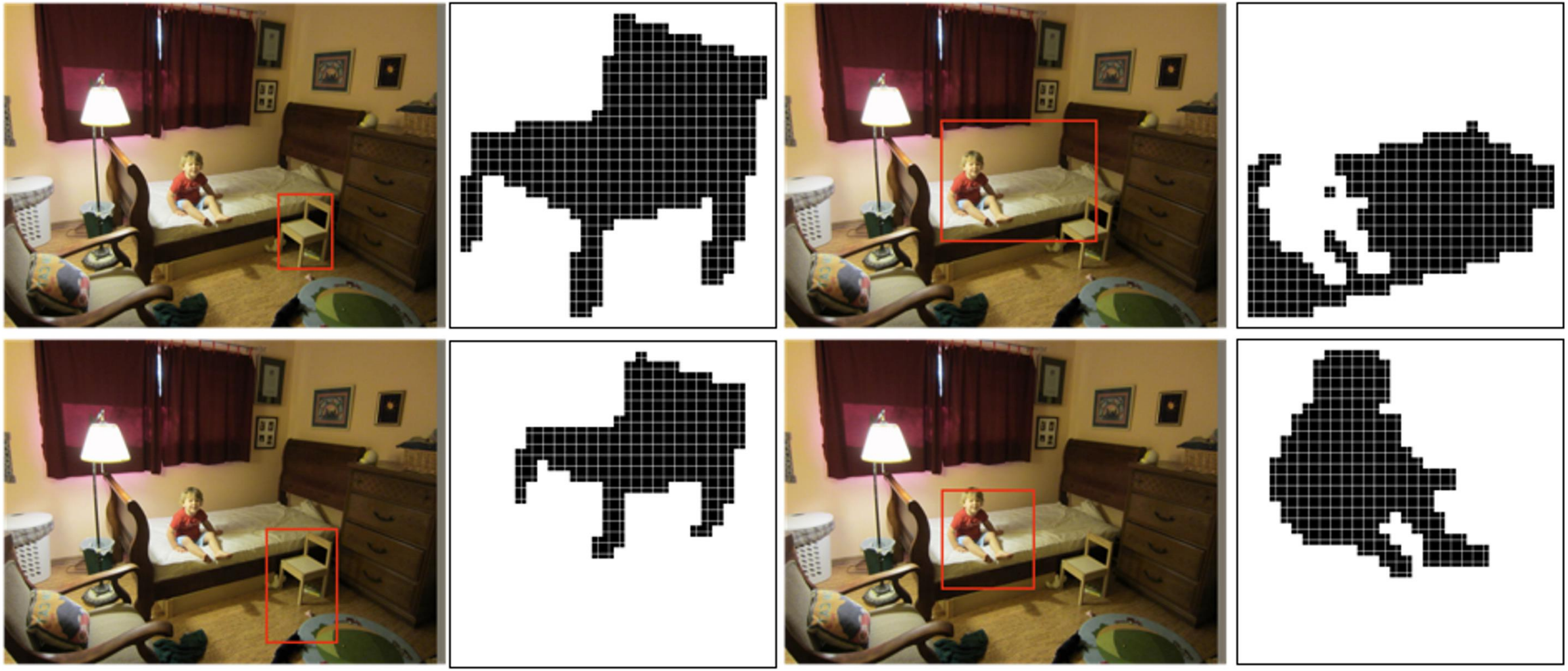
→ total loss is sum of RPN, detection-head, and mask-head losses

Mask R-CNN

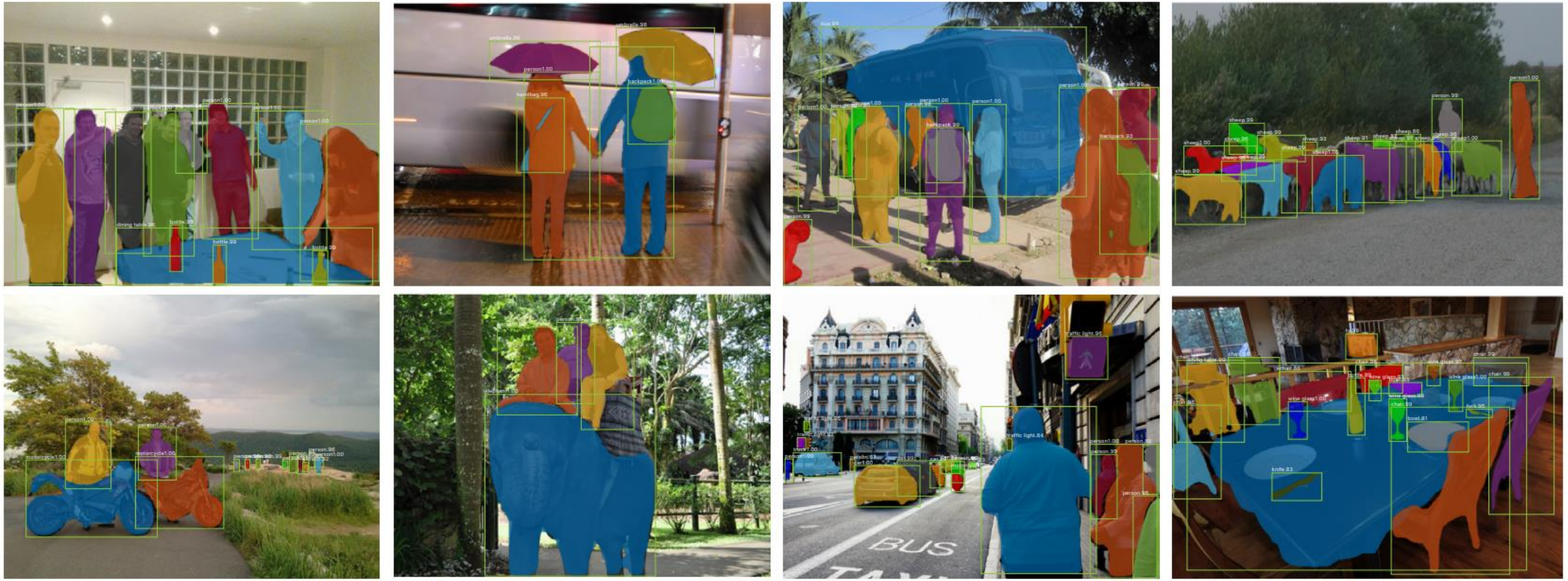


1. object detection (boundary boxes)
2. prediction of separate binary masks for each detected object (specific instances)

image with training proposal | 28×28 mask target | image with training proposal | 28×28 mask target



results on MS COCO data set



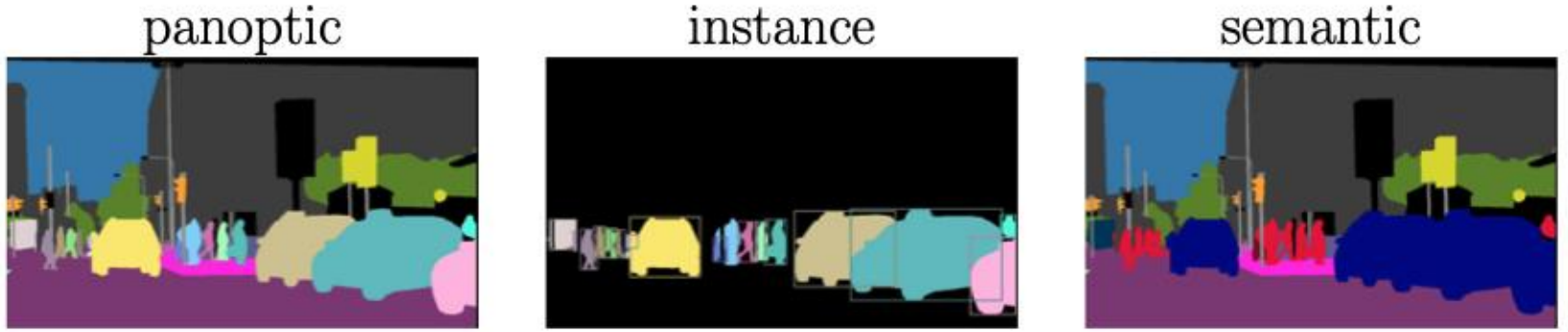
Panoptic Segmentation

unification of semantic sementation (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)

source

Promptable Segmentation with Transformers

Segment Anything Model (SAM)

SAM2 includes also video segmentation

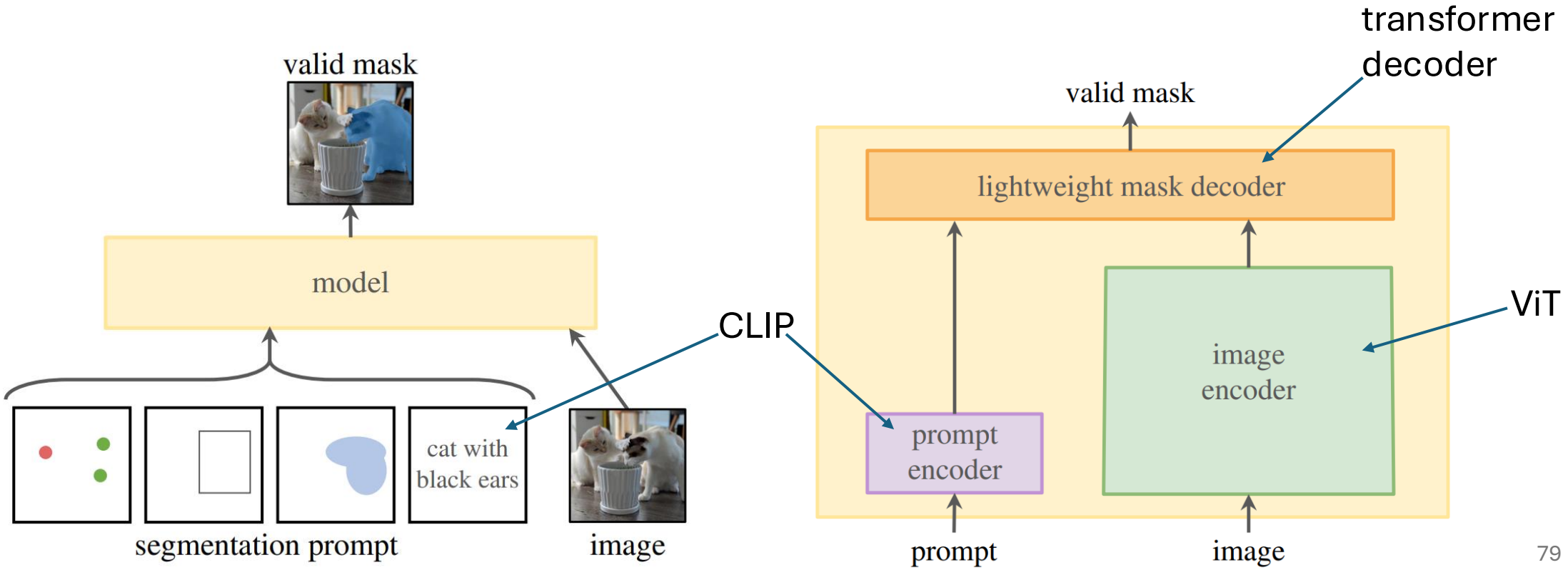


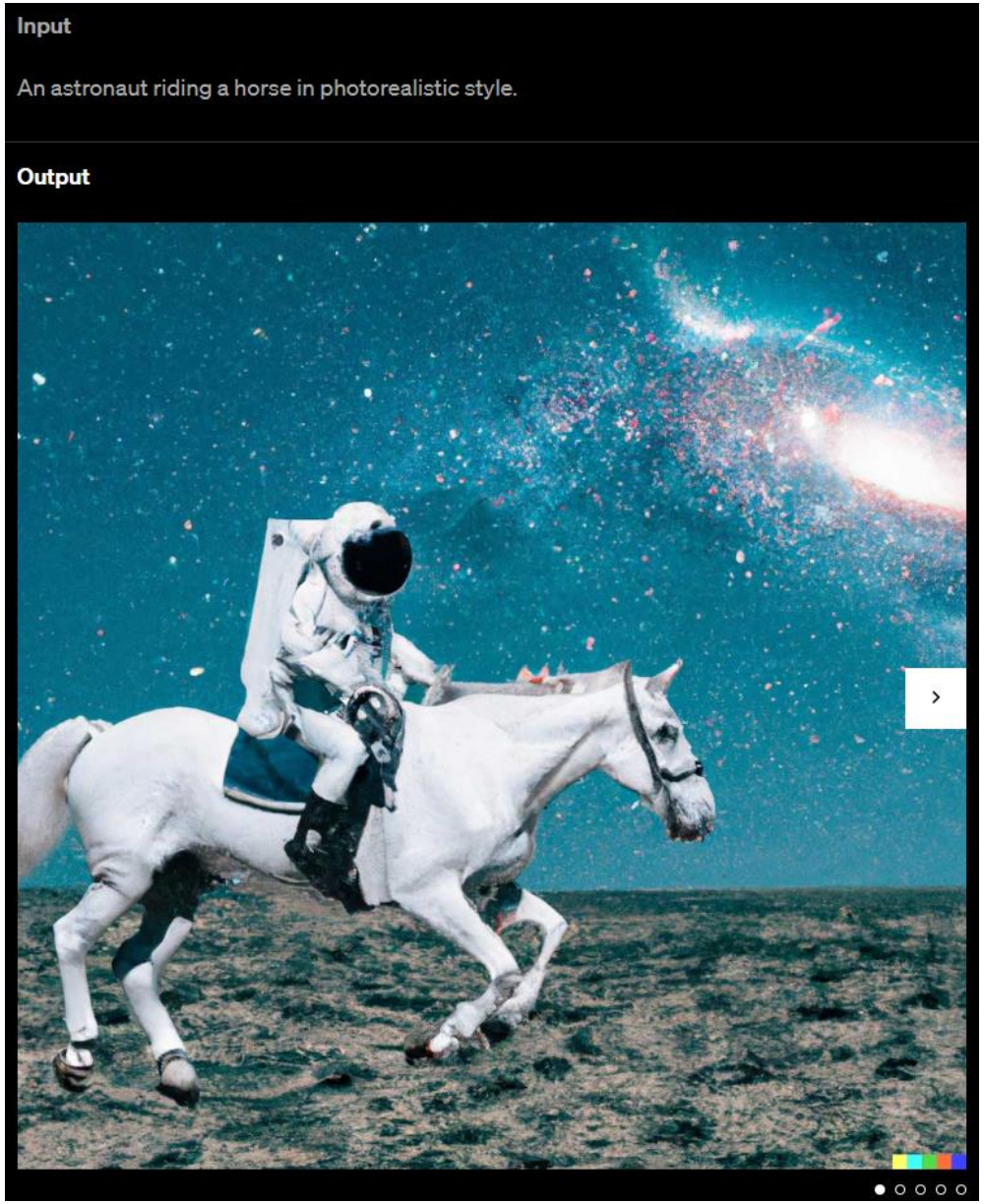
Image Synthesis

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

example: DALL-E 2



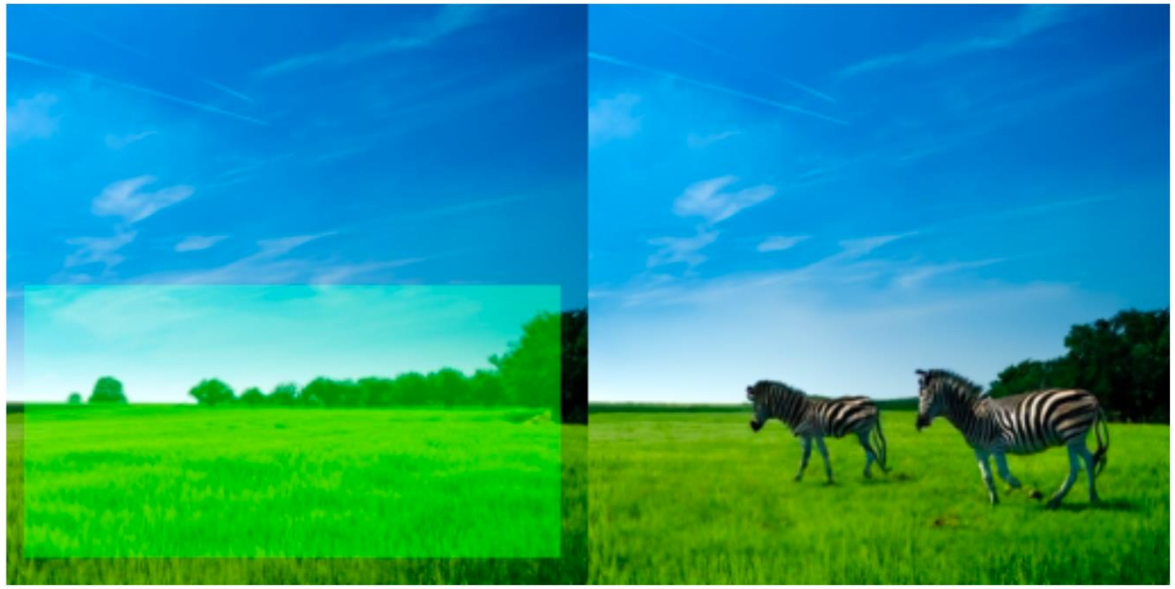
plenty of products: DALL-E, Stable Diffusion, ImageGen, Midjourney, …

Stable Diffusion:

A scenic view of mountains at sunset, digital art

prompt

inpainting example (GLIDE):



"zebras roaming in the field"

source

Generative vs Predictive/Discriminative Models

discriminative models:

predict conditional probability $P(Y|\boldsymbol{X})$

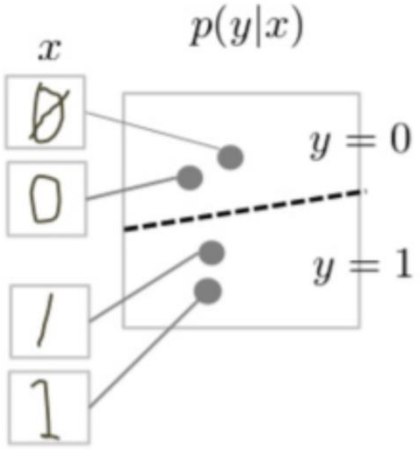
generative models:

predict joint probability $P(Y, \boldsymbol{X})$

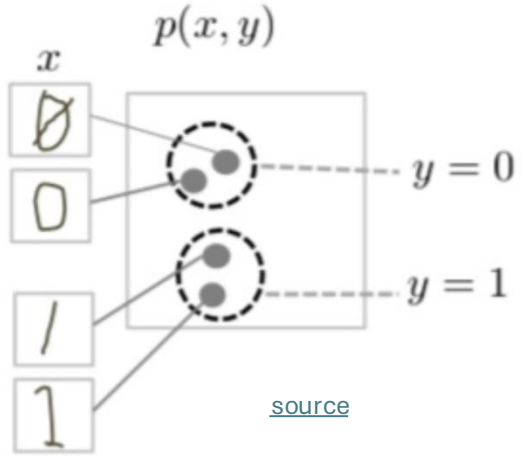
(or just $P(\boldsymbol{X})$ → unsupervised learning)

→ allow to generate new data samples

discriminative model



generative model



source

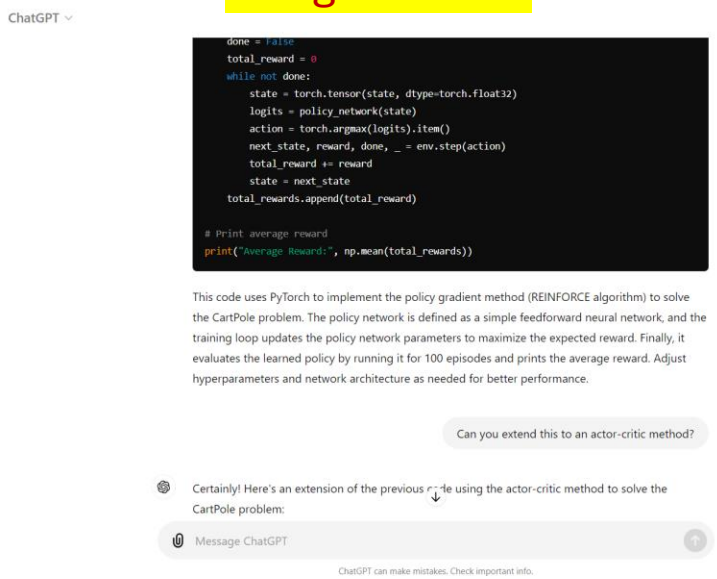
task of generative models more difficult: need to model full data distribution rather than merely find patterns in inputs to distinguish outputs

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



[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]

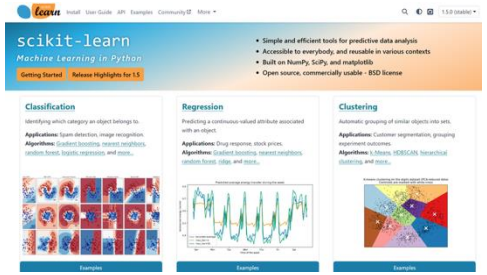
text-to-video

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]

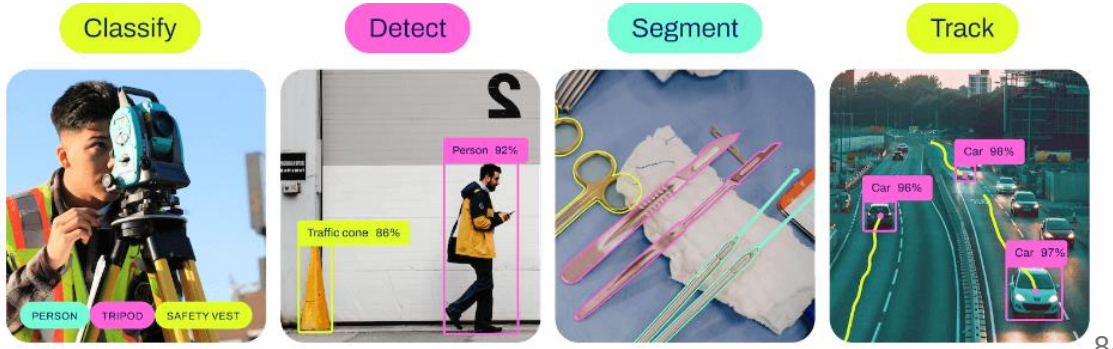
BERT family

tabular data



dmlc XGBoost

computer vision



[YOLO]

Deep Learning for Generative AI

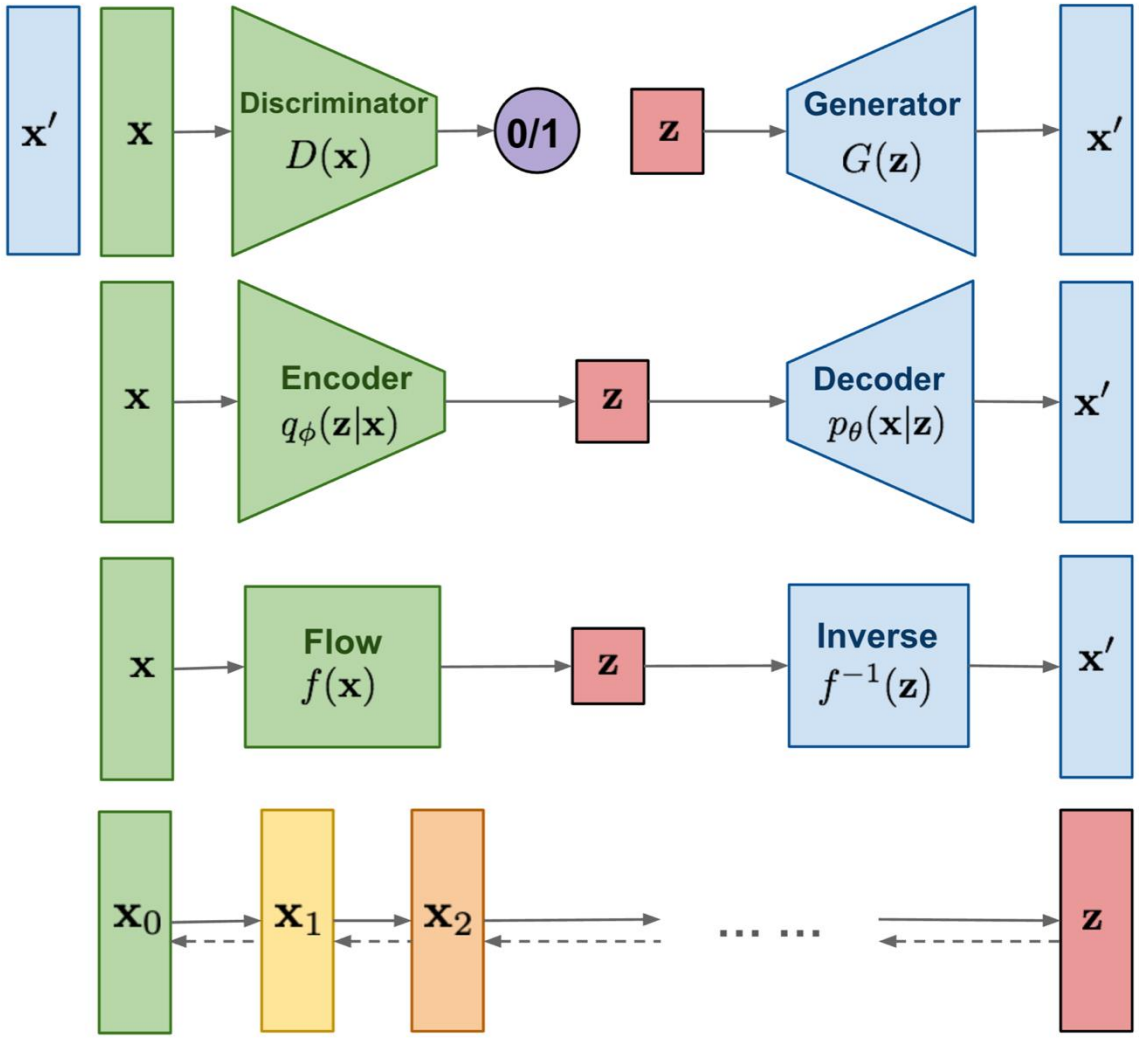
Depending on the application, there are currently two dominant approaches for generative AI:

- 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



two neural networks playing a zero-sum game

learn variational distribution (not just replicating inputs)

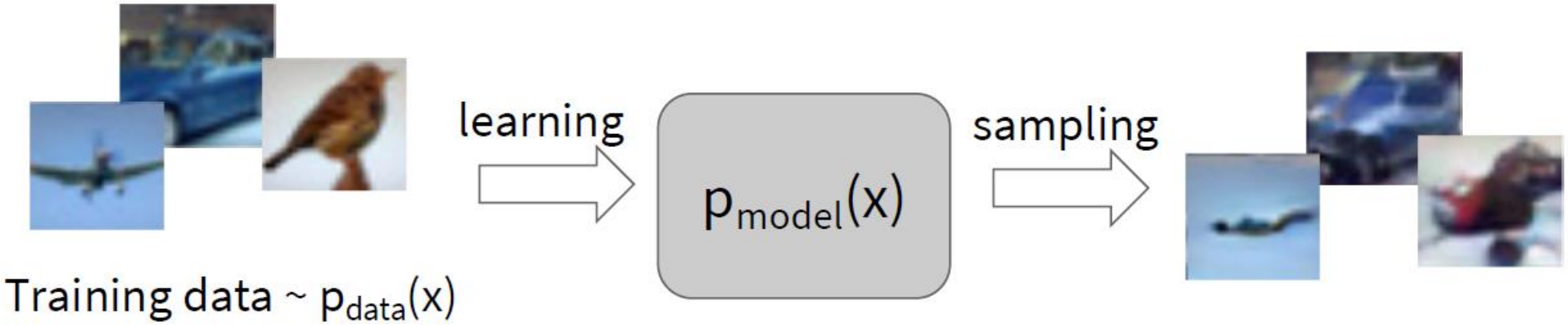
more complex distributions by applying change-of-variable technique (need for specialized architecture)

chain of denoising autoencoders

source

→ generalization: flow matching

Generative Modeling



Objectives:

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

Explicit density estimation: explicitly define and solve for $p_{model}(x)$
Implicit density estimation: learn model that can sample from $p_{model}(x)$ without explicitly defining it.