Convolutional Neural Networks

Object Detection, Semantic Segmentation

Rafał Nowak April 29, 2025

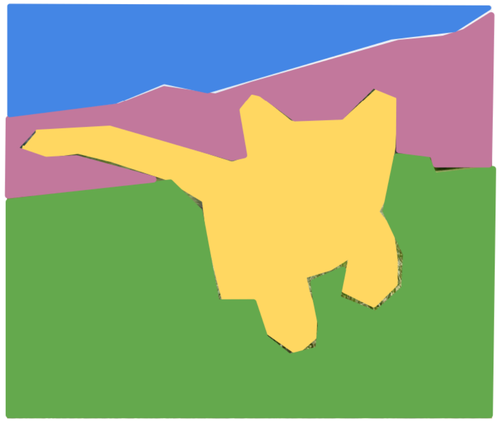
Computer Vision Tasks

Classification

CAT

No spatial extent

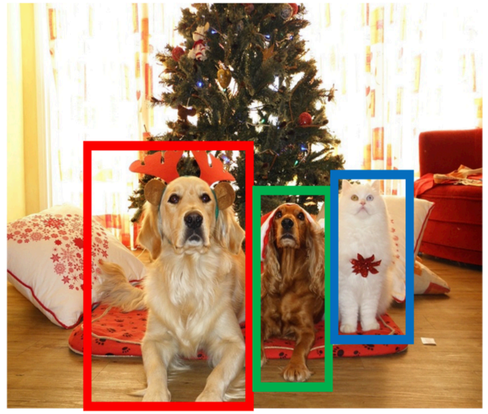
Semantic Segmentation



GRASS, CAT, TREE, SKY

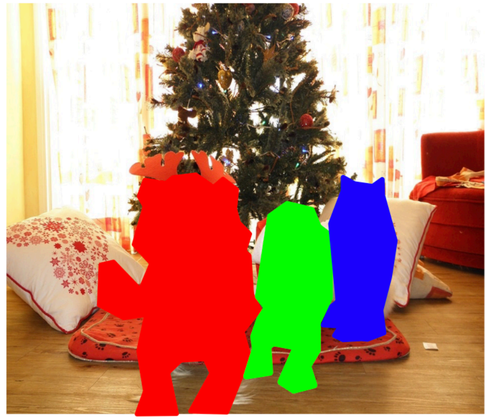
No objects, just pixels

Object Detection



DOG, DOG, CAT

Instance Segmentation



DOG, DOG, CAT

Multiple Objects

Object detection

Evaluation Metrics

- **IoU** (Intersection over Union):
  - How well boxes overlap ground truth
- **mAP** (mean Average Precision):
  - How accurate predictions are across classes

IoU

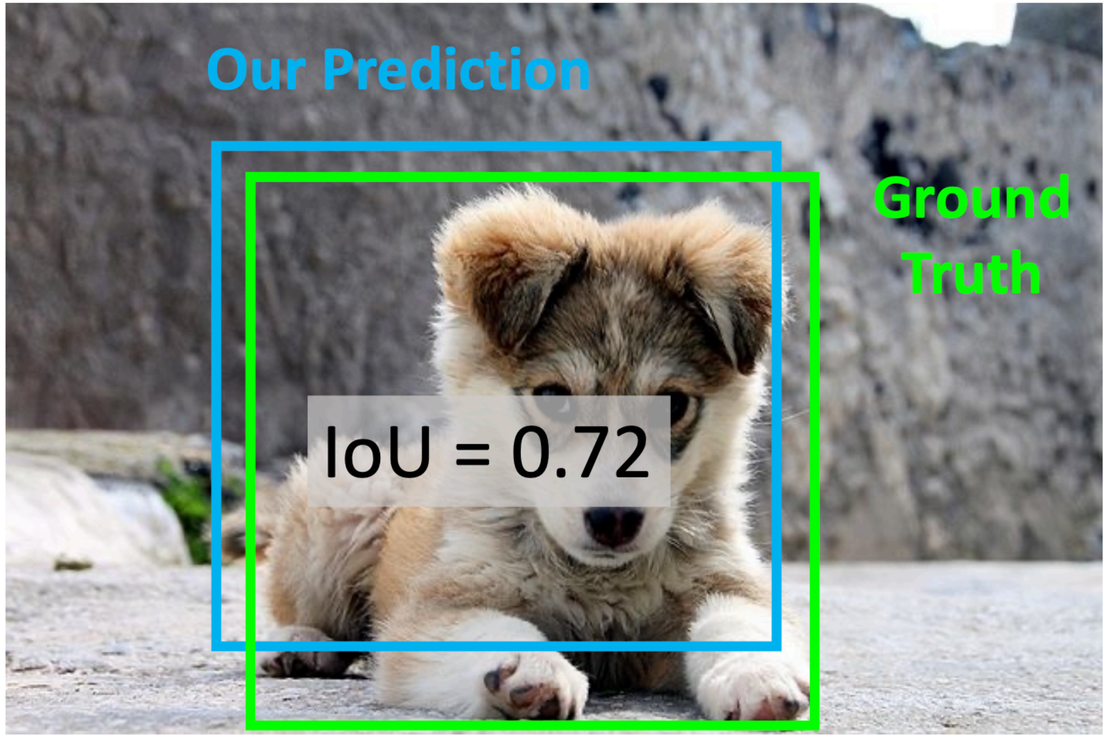
Comparing Boxes: Intersection over Union (IoU)

How can we compare our prediction to the ground-truth box?

Intersection over Union (IoU)
(Also called “Jaccard similarity” or “Jaccard index”):

$$\frac{\textit{Area of Intersection}}{\textit{Area of Union}}$$

IoU > 0.5 is “decent”,
IoU > 0.7 is “pretty good”,



NMS

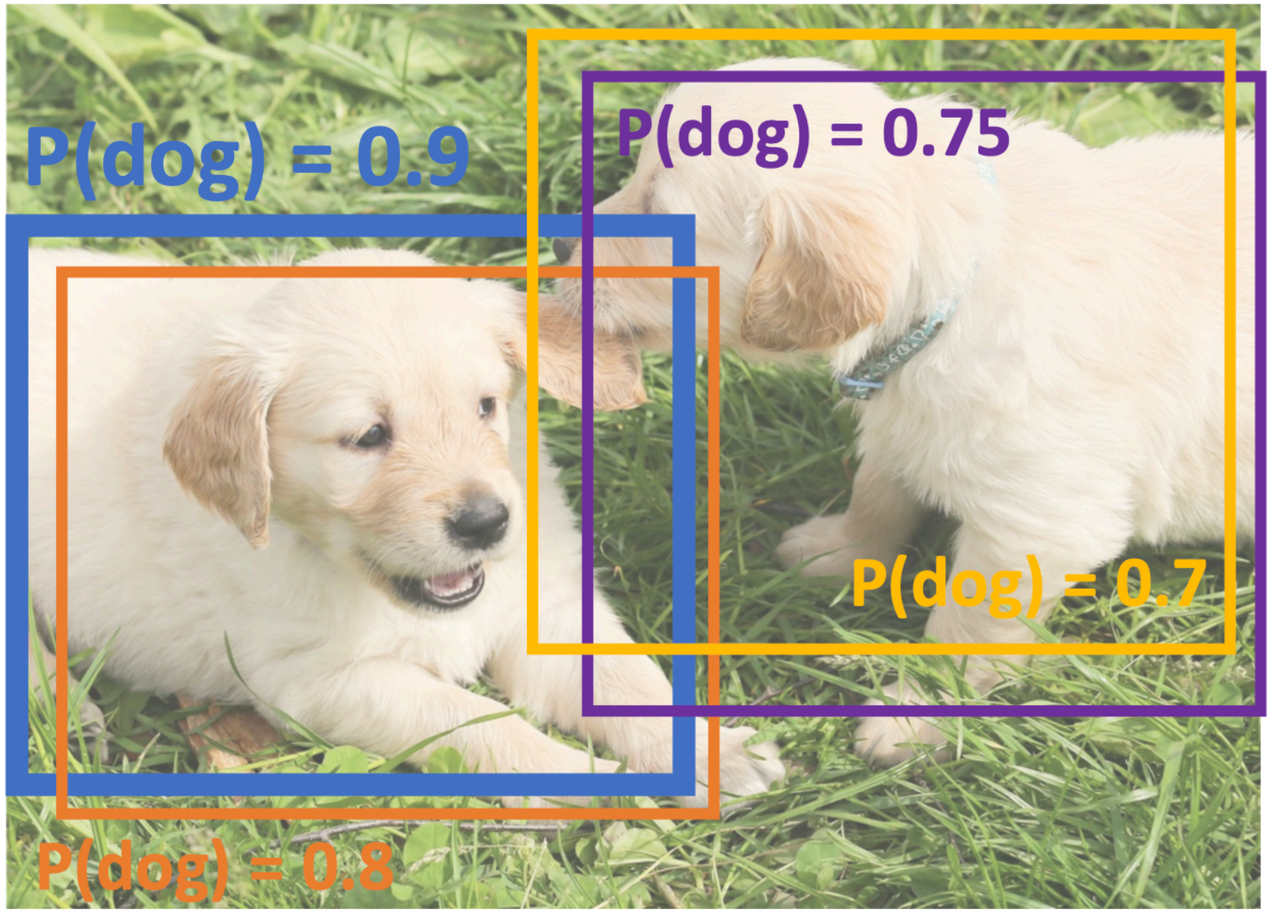
Overlapping Boxes: Non-Max Suppression (NMS)

Problem: Object detectors often output many overlapping detections:

Solution: Post-process raw detections using **Non-Max Suppression (NMS)**

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with IoU > threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1

IoU(■, ■) = **0.78**
IoU(■, ■) = 0.05
IoU(■, ■) = 0.07



mean Average Precision

- Two main ideas:
  - **Precision**: Of the boxes predicted, how many are correct?
  - **Recall**: Of all real objects, how many did the model find?
- Average **precision** across **recall** levels, and then across all classes.

mean Average Precision

- **Step 1:** Gather predictions and ground truth
  - For a given class (e.g., "car"):
    - Collect all predicted boxes for that class across all images
    - Sort them by **confidence score** (highest first)
- **Step 2:** Match predictions to ground truth
  - For each predicted box:
    - Compute **IoU** (Intersection over Union) with ground truth boxes
    - If IoU > threshold (e.g. 0.5), and it's the **best match**, it’s a **True Positive** (TP)
    - If not, it’s a **False Positive** (FP)
  - Any unmatched ground truth is a **False Negative** (FN)

mean Average Precision

- **Step 3: Compute Precision and Recall** (for a given threshold, like 0.5)

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

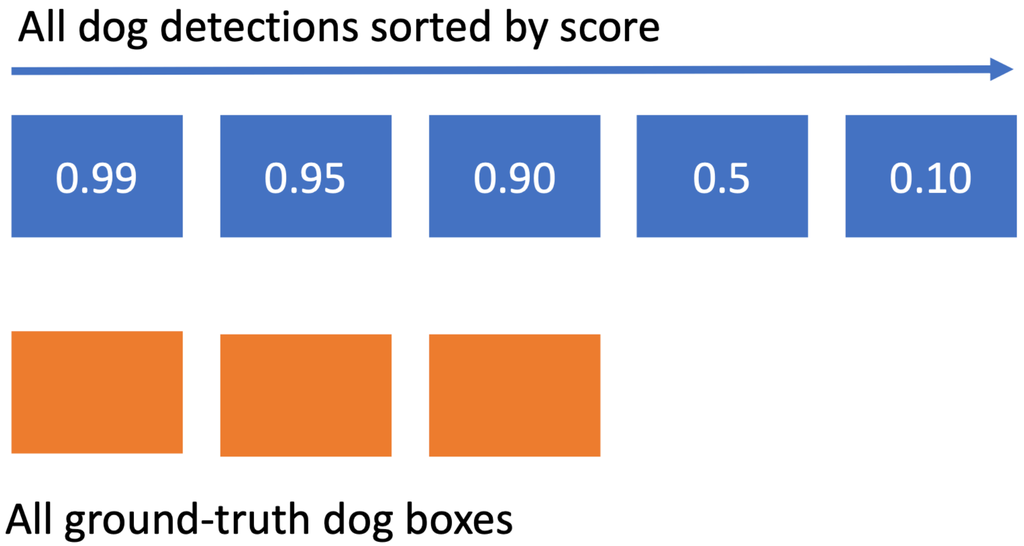
- **Step 4: Calculate Average Precision (AP)**
  - **AP** = area under the Precision-Recall curve.
  - There are two common ways to do this:
    - VOC-style: 11-point interpolation
    - COCO-style: integrate over 101 points of recall from 0 to 1

mean Average Precision @ 0.5

Example

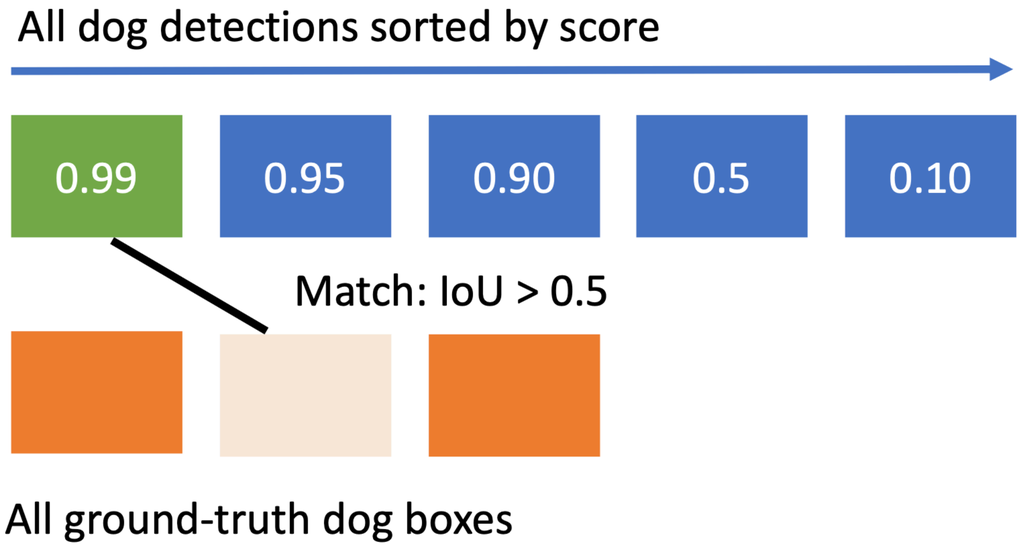
Evaluating Object Detectors: Mean Average Precision (mAP)

1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)



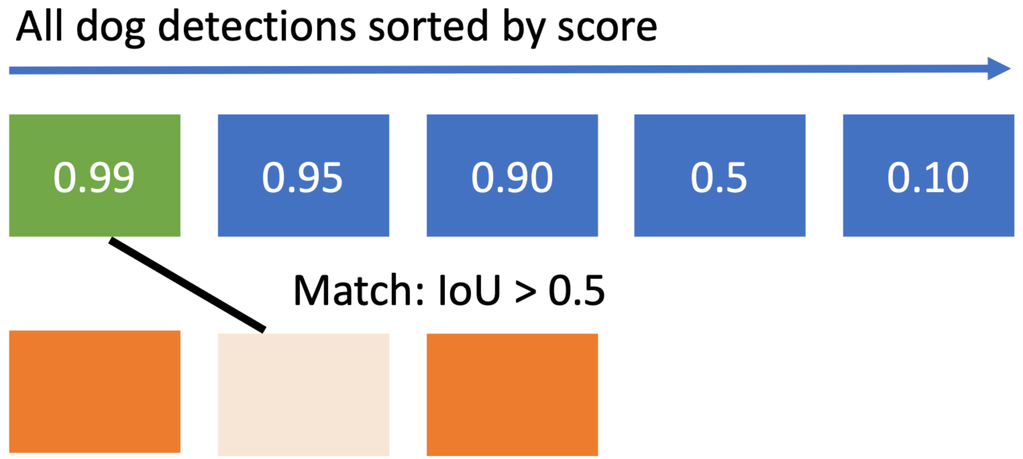
Evaluating Object Detectors: Mean Average Precision (mAP)

1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)
        1. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
        2. Otherwise mark it as negative

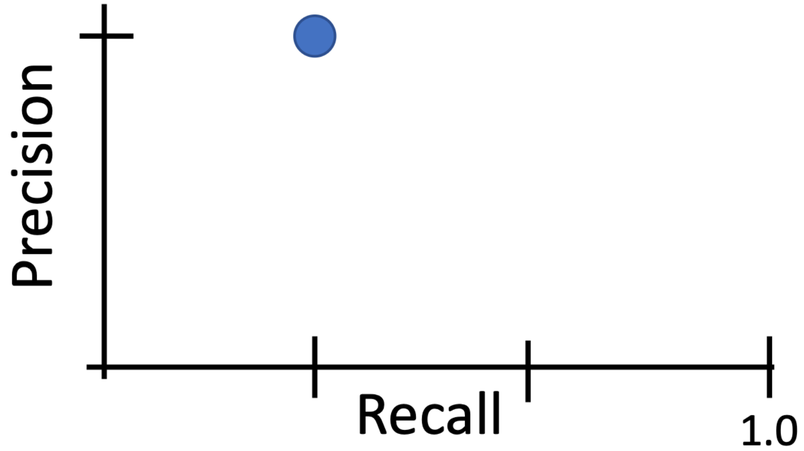


Evaluating Object Detectors: Mean Average Precision (mAP)

1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)
        1. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
        2. Otherwise mark it as negative
        3. Plot a point on PR Curve

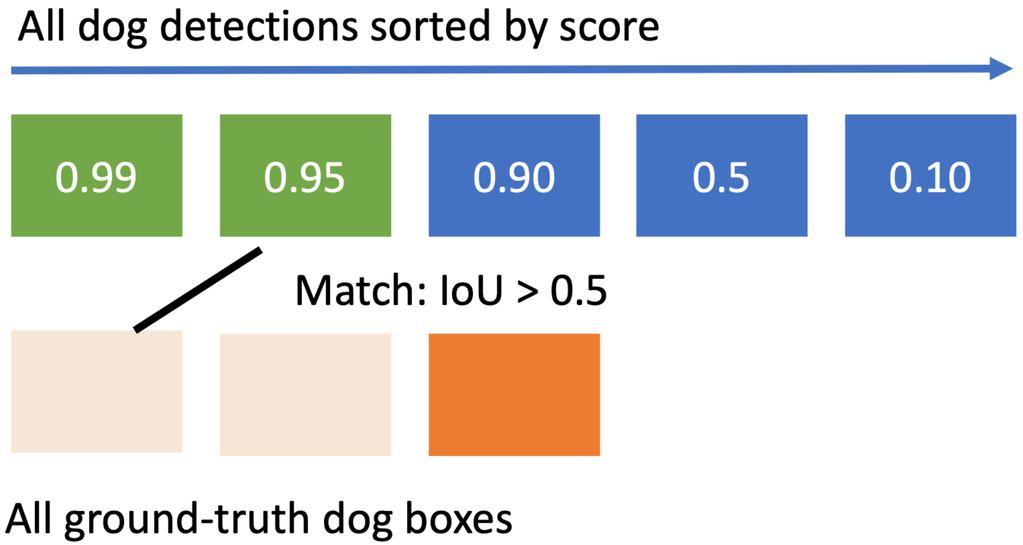


Precision = 1/1 = 1.0
Recall = 1/3 = 0.33

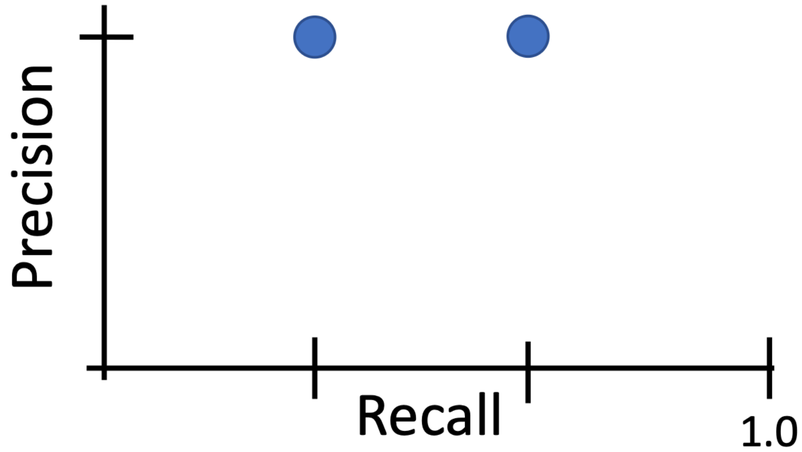


Evaluating Object Detectors: Mean Average Precision (mAP)

1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)
        1. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
        2. Otherwise mark it as negative
        3. Plot a point on PR Curve

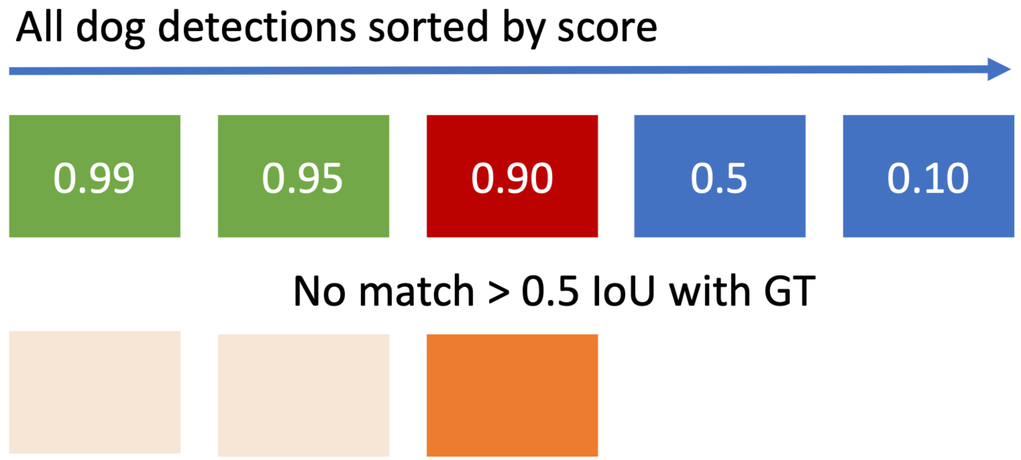


Precision = 2/2 = 1.0
Recall = 2/3 = 0.67



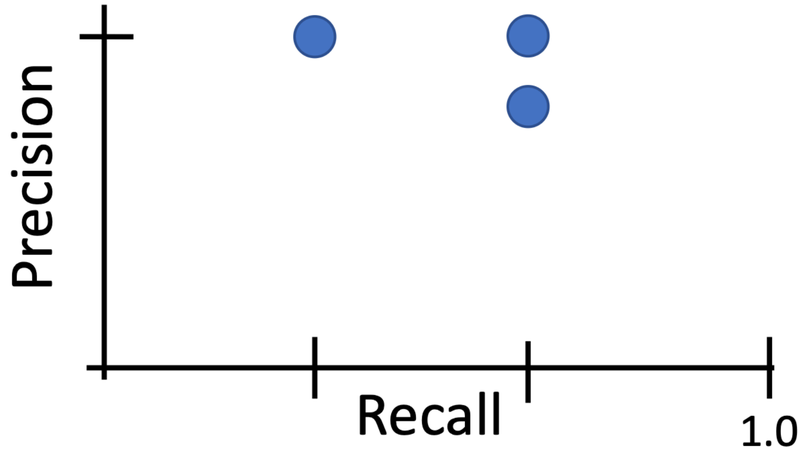
Evaluating Object Detectors: Mean Average Precision (mAP)

1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)
        1. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
        2. Otherwise mark it as negative
        3. Plot a point on PR Curve



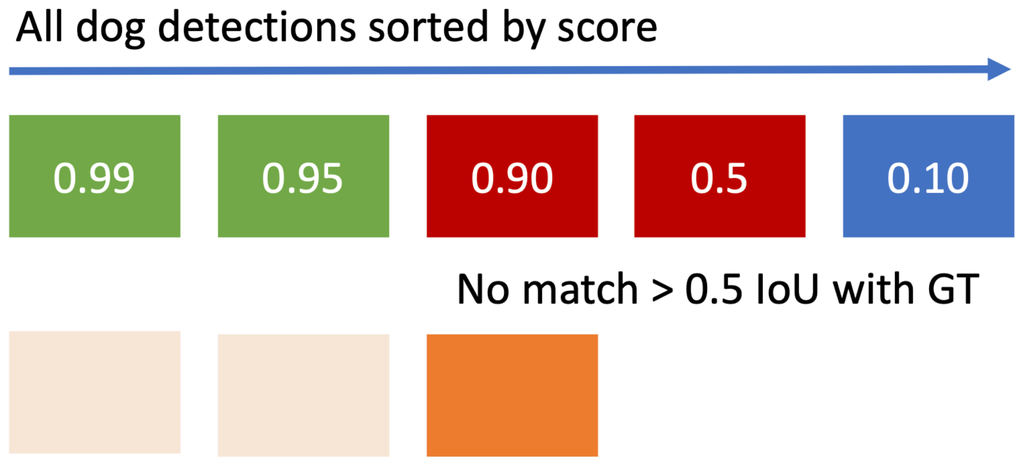
All ground-truth dog boxes

Precision = 2/3 = 0.67
Recall = 2/3 = 0.67

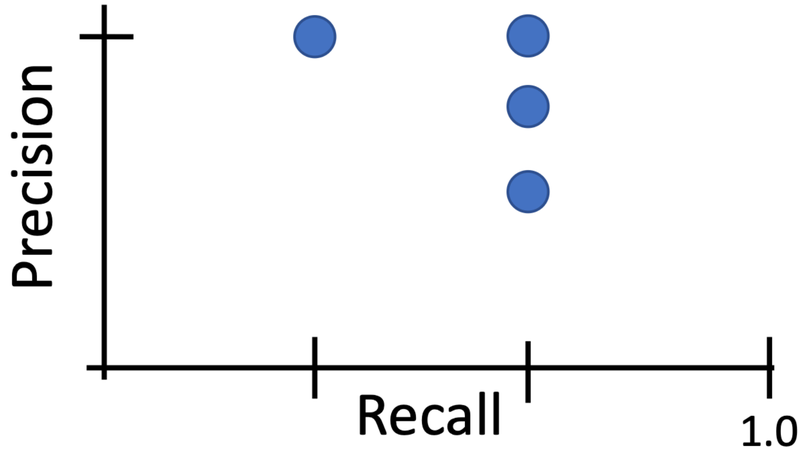


Evaluating Object Detectors: Mean Average Precision (mAP)

1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)
        1. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
        2. Otherwise mark it as negative
        3. Plot a point on PR Curve

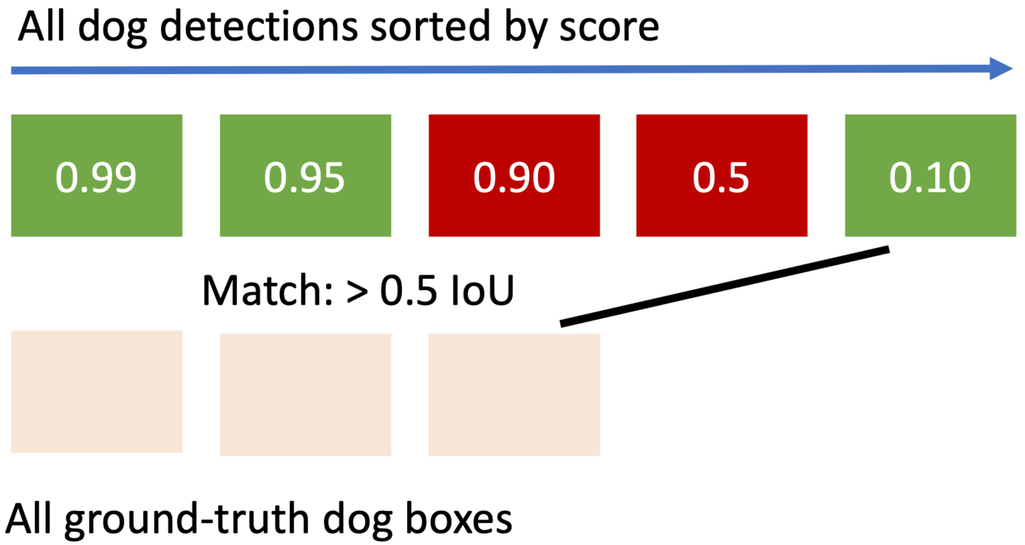


Precision = 2/4 = 0.5
Recall = 2/3 = 0.67

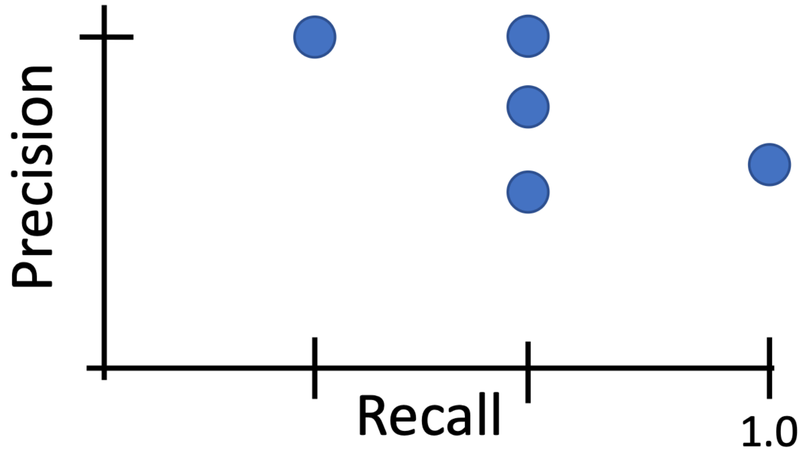


Evaluating Object Detectors: Mean Average Precision (mAP)

1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)
        1. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
        2. Otherwise mark it as negative
        3. Plot a point on PR Curve

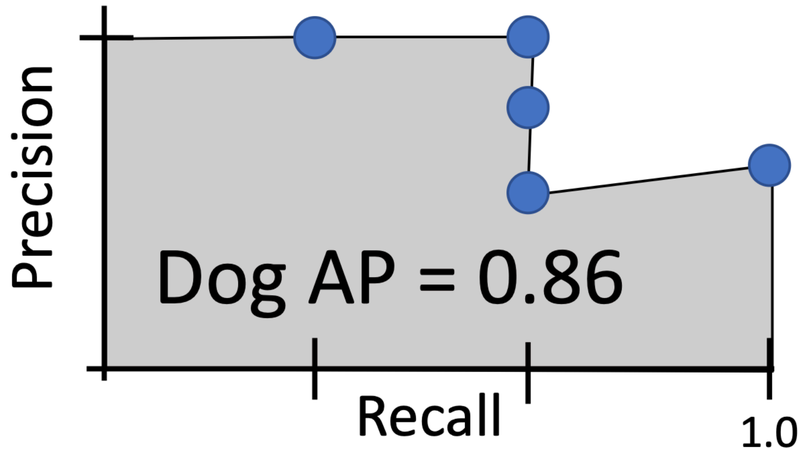


Precision = 3/5 = 0.6
Recall = 3/3 = 1.0



Evaluating Object Detectors: Mean Average Precision (mAP)

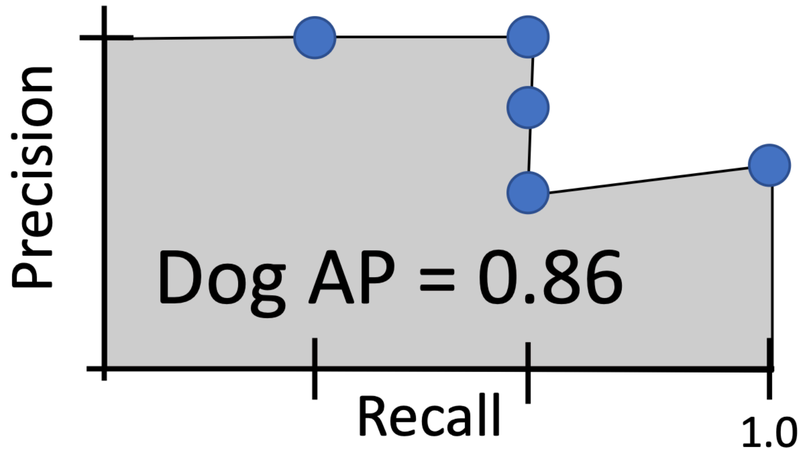
1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)
        1. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
        2. Otherwise mark it as negative
        3. Plot a point on PR Curve
    2. Average Precision (AP) = area under PR curve



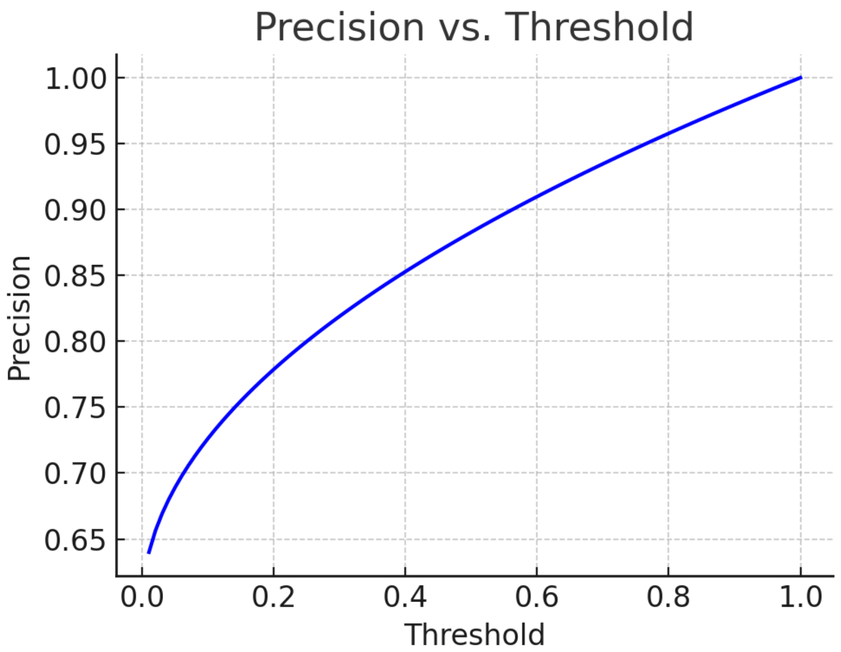
Evaluating Object Detectors: Mean Average Precision (mAP)

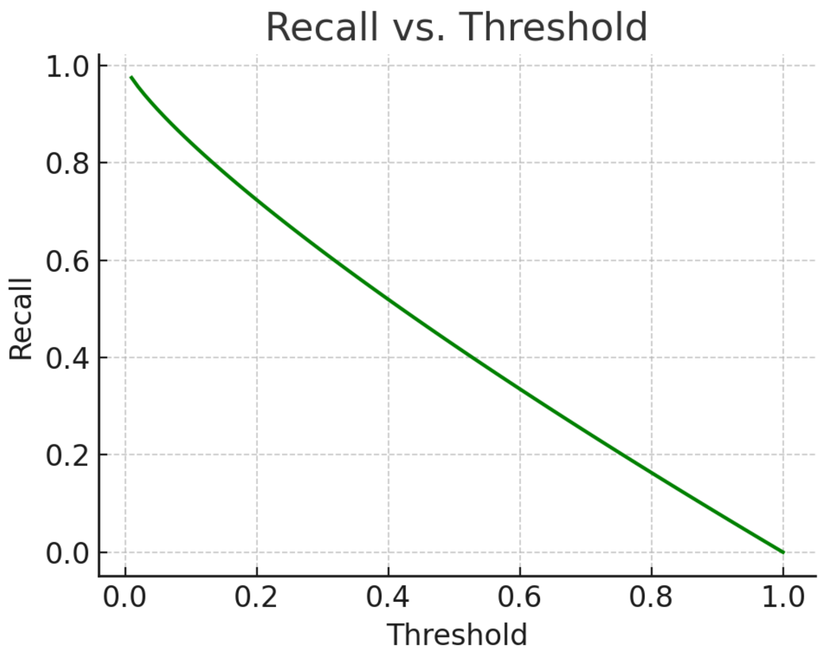
1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)
        1. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
        2. Otherwise mark it as negative
        3. Plot a point on PR Curve
    2. Average Precision (AP) = area under PR curve

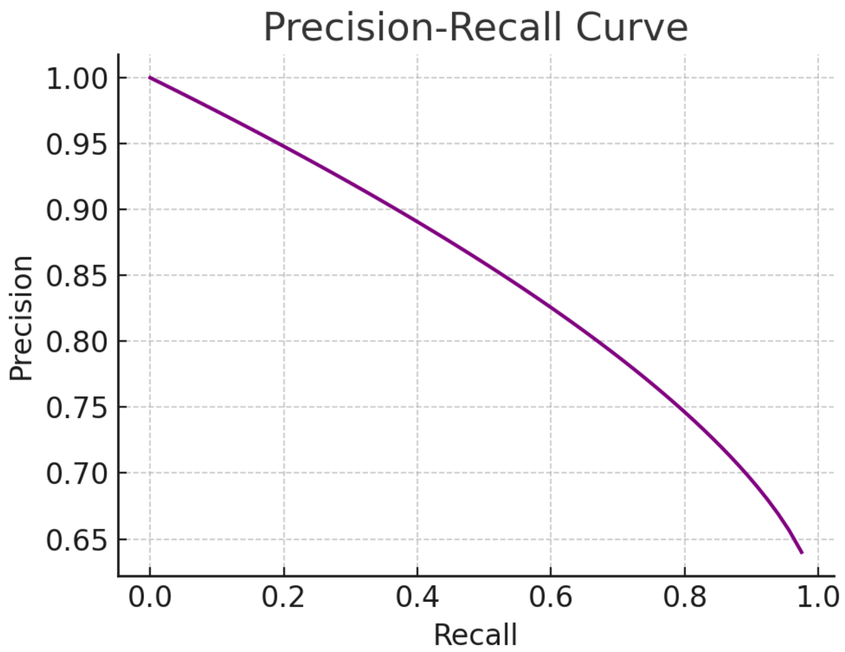
How to get AP = 1.0: Hit all GT boxes with IoU > 0.5, and have no "false positive" detections ranked above any "true positives"



mean Average Precision







- VOC-style: 11-point interpolation
- COCO-style: integrate over 101 points of recall from 0 to 1

mean Average Precision

- **Step 5: Calculate mean Average Precision (mAP)**
  - Once you have AP for each class, you average them:

$$\text{mAP} = \frac{1}{C} \sum_{c=1}^{C} \text{AP}_c$$

"Is my model detecting **all objects**, and **only the right ones**, with **good boxes**?"

Evaluating Object Detectors: Mean Average Precision (mAP)

1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)
        1. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
        2. Otherwise mark it as negative
        3. Plot a point on PR Curve
    2. Average Precision (AP) = area under PR curve
3. Mean Average Precision (mAP) = average of AP for each category
4. For "COCO mAP": Compute mAP@thresh for each IoU threshold (0.5, 0.55, 0.6, …, 0.95) and take average

mAP@0.5 = 0.77

mAP@0.55 = 0.71

mAP@0.60 = 0.65

…

mAP@0.95 = 0.2

COCO mAP = 0.4

What is mAP?

Results on Pascal VOC (test set)

- https://arxiv.org/pdf/1612.08242

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	**78.6**	40

YOLO9000:
Better, Faster, Stronger

Joseph Redmon*†, Ali Farhadi*†
University of Washington*, Allen Institute for AI†
http://pjreddie.com/yolo9000/

Abstract

We introduce YOLO9000, a state-of-the-art, real-time object detection system that can detect over 9000 object categories. First we propose various improvements to the YOLO detection method, both novel and drawn from prior work. The improved model, YOLOv2, is state-of-the-art on standard detection tasks like PASCAL *VOC and COCO. Using a novel, multi-scale training method the same YOLOv2 model can run at varying sizes, offering an easy tradeoff between speed and accuracy. At 67 FPS, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 FPS, YOLOv2 gets 78.6 mAP, outperforming state-of-the-art methods like Faster R-CNN with ResNet and SSD while still running significantly faster. Finally we propose a method to jointly train on object detection and classification. Using this method we train YOLO9000 simultaneously on the COCO detection dataset and the ImageNet classification dataset. Our joint training allows YOLO9000 to predict detections for object classes that don't have labelled detection data. We validate our approach on the ImageNet detection task. YOLO9000 gets 19.7 mAP on the ImageNet detection validation set despite only having detection data for 44 of the 200 classes. On the 156 classes not in COCO, YOLO9000 gets 16.0 mAP. But YOLO can detect more than just 200 classes; it predicts detections for more than 9000 different object categories. And it still runs in real-time.*

1. Introduction

General purpose object detection should be fast, accurate, and able to recognize a wide variety of objects. Since the introduction of neural networks, detection frameworks have become increasingly fast and accurate. However, most detection methods are still constrained to a small set of objects.

Current object detection datasets are limited compared to datasets for other tasks like classification and tagging. The most common detection datasets contain thousands to hundreds of thousands of images with dozens to hundreds of tags [3] [10] [2]. Classification datasets have millions of images with tens or hundreds of thousands of categories [20] [2].

We would like detection to scale to level of object classification. However, labelling images for detection is far more expensive than labelling for classification or tagging (tags are often user-supplied for free). Thus we are unlikely

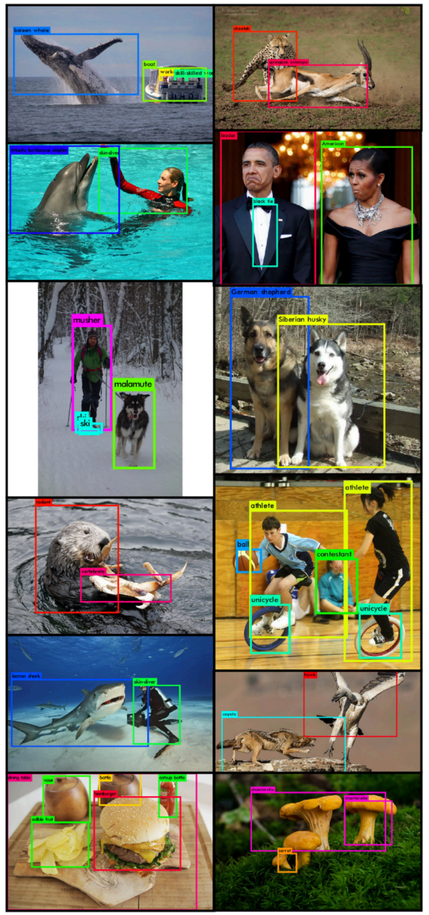


Figure 1: YOLO9000. YOLO9000 can detect a wide variety of object classes in real-time.

YOLO

You Only Look Once: Unified, Real-Time Object Detection

- https://arxiv.org/abs/1506.02640

You Only Look Once:
Unified, Real-Time Object Detection

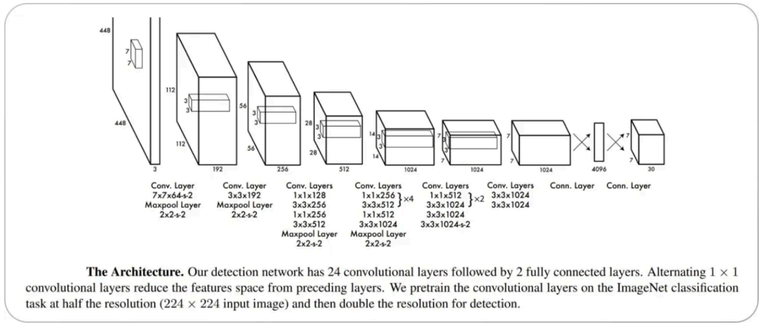
Joseph Redmon*, Santosh Divvala*†, Ross Girshick¶, Ali Farhadi*†
University of Washington*, Allen Institute for AI†, Facebook AI Research¶
http://pjreddie.com/yolo/

YOLO

You Only Look Once: Unified, Real-Time Object Detection

- ### How does YOLO work? YOLO Architecture

  The YOLO algorithm takes an image as input and then uses a simple deep convolutional neural network to detect objects in the image. The architecture of the CNN model that forms the backbone of YOLO is shown below.

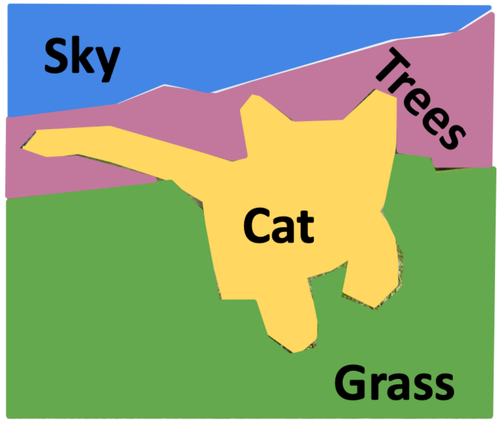
  The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

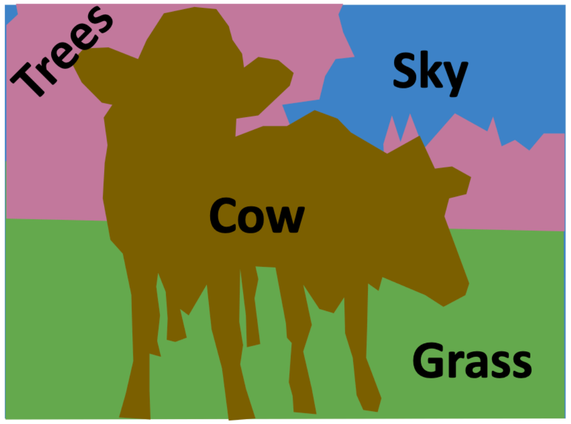
- https://medium.com/@v7.editors/yolo-algorithm-for-object-detection-explained-examples-90ccfd8a5642

Semantic segmantation

Label each pixel in the image with a category label

Don’t differentiate instances, only care about pixels

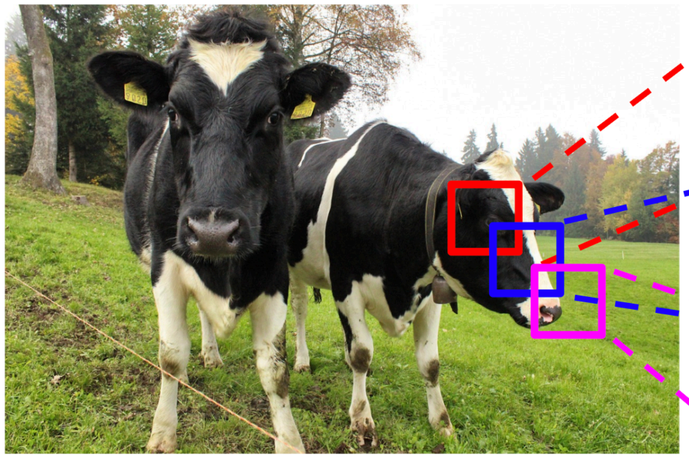




slides: **Lecture 15,** 37 — 70

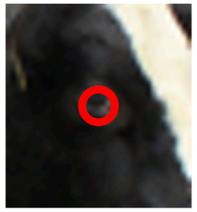
Semantic Segmentation Idea: Sliding Window

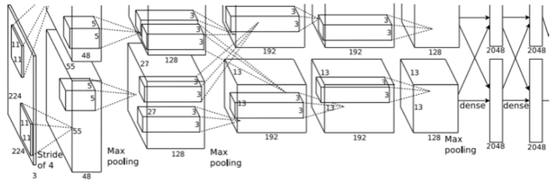
Full image



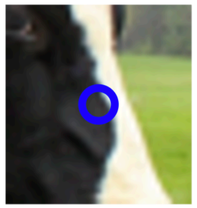
Extract patch

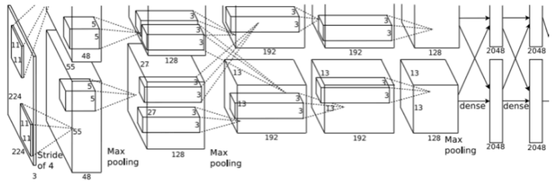
Classify center pixel with CNN



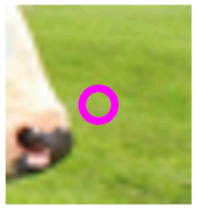


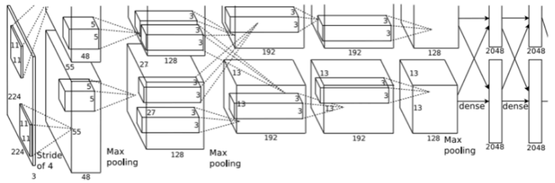
Cow





Cow





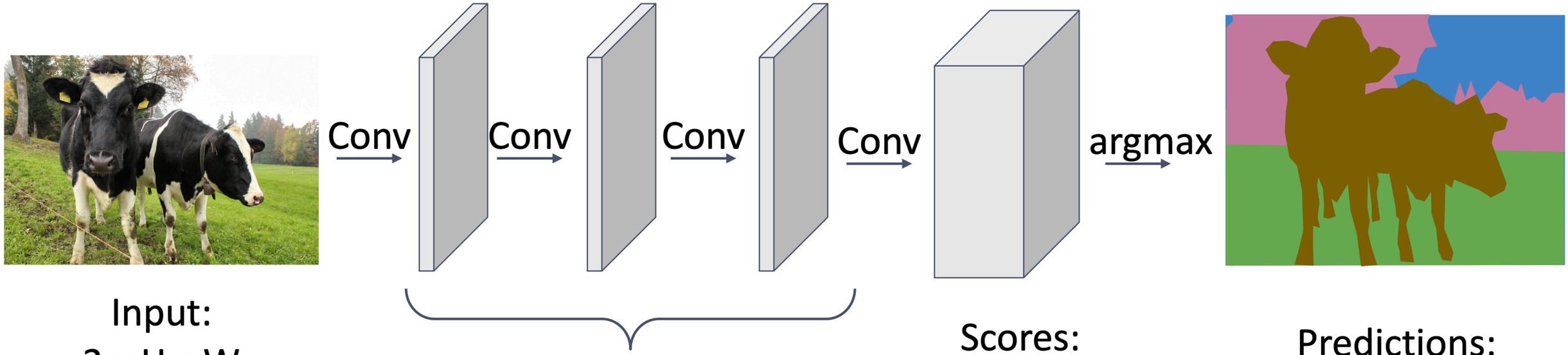
Grass

Farabet et al, “Learning Hierarchical Features for Scene Labeling,” TPAMI 2013

Pinheiro and Collobert, “Recurrent Convolutional Neural Networks for Scene Labeling”, ICML 2014

Semantic Segmentation: Fully Convolutional Network

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!



Input:
3 x H x W

Convolutions:
D x H x W

Scores:
C x H x W

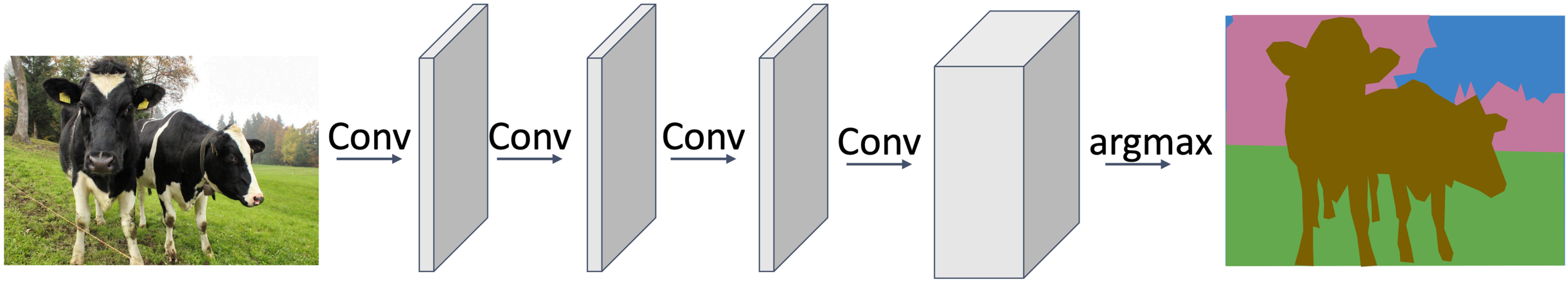
Predictions:
H x W

Loss function: Per-Pixel cross-entropy

Long et al, "Fully convolutional networks for semantic segmentation", CVPR 2015

Semantic Segmentation: Fully Convolutional Network

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!



Input:
3 x H x W

Problem #1: Effective receptive field size is linear in number of conv layers: With L 3x3 conv layers, receptive field is 1+2L

Problem #2: Convolution on high res images is expensive! Recall ResNet stem aggressively downsamples

Long et al, "Fully convolutional networks for semantic segmentation", CVPR 2015

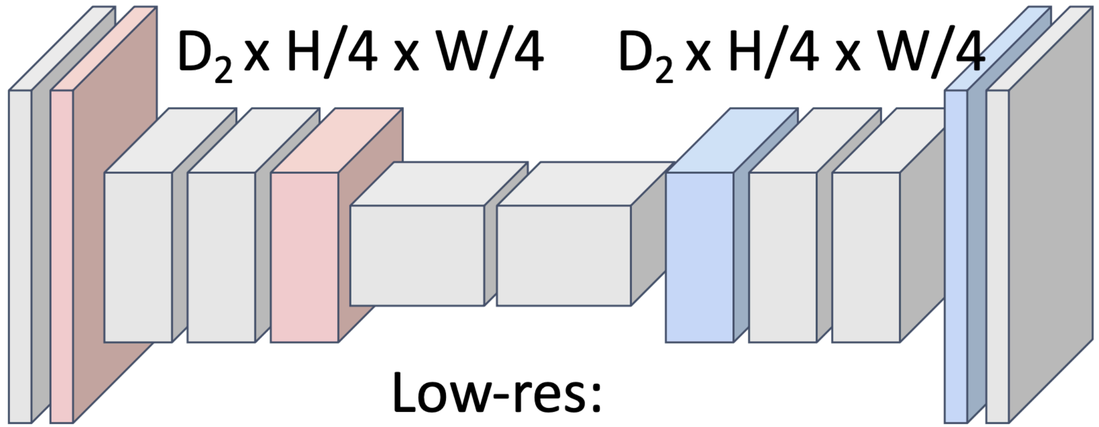
Semantic Segmentation: Fully Convolutional Network

Downsampling: Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling: ???

Input:
3 x H x W

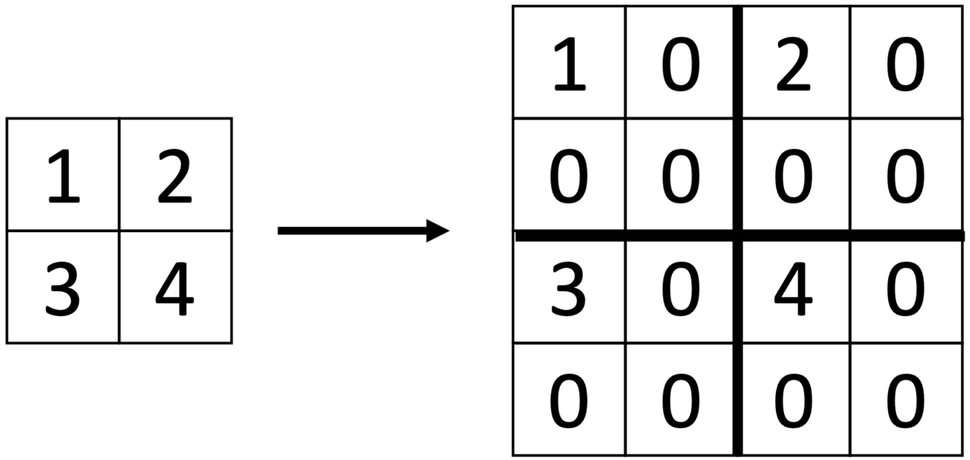


Predictions:
H x W

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

In-Network Upsampling: “Unpooling”

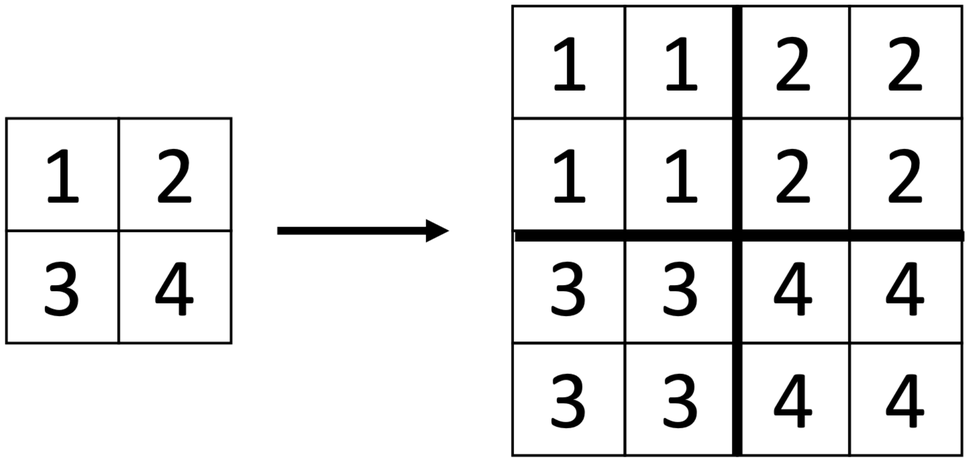
Bed of Nails



Input
C x 2 x 2

Output
C x 4 x 4

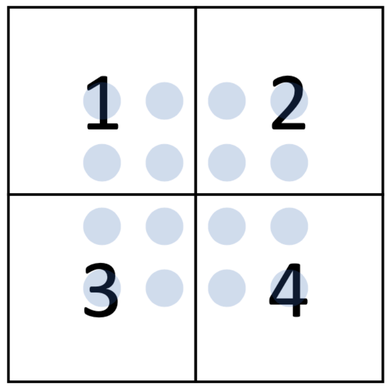
Nearest Neighbor



Input
C x 2 x 2

Output
C x 4 x 4

In-Network Upsampling: Bilinear Interpolation



1.00	1.25	1.75	2.00
1.50	1.75	2.25	2.50
2.50	2.75	3.25	3.50
3.00	3.25	3.75	4.00

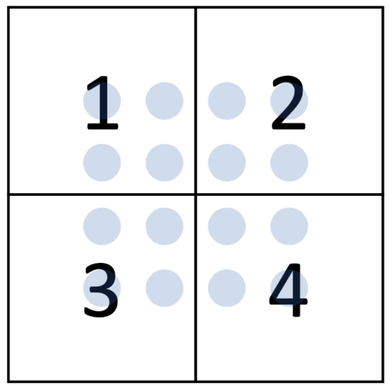
Input: C x 2 x 2

Output: C x 4 x 4

$$f_{x,y} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - i|) \max(0, 1 - |y - j|) \quad i \in \{\lfloor x \rfloor - 1, \ldots, \lceil x \rceil + 1\} \quad j \in \{\lfloor y \rfloor - 1, \ldots, \lceil y \rceil + 1\}$$

Use two closest neighbors in x and y to construct linear approximations

In-Network Upsampling: Bicubic Interpolation



0.68	1.02	1.56	1.89
1.35	1.68	2.23	2.56
2.44	2.77	3.32	3.65
3.11	3.44	3.98	4.32

Input: C x 2 x 2

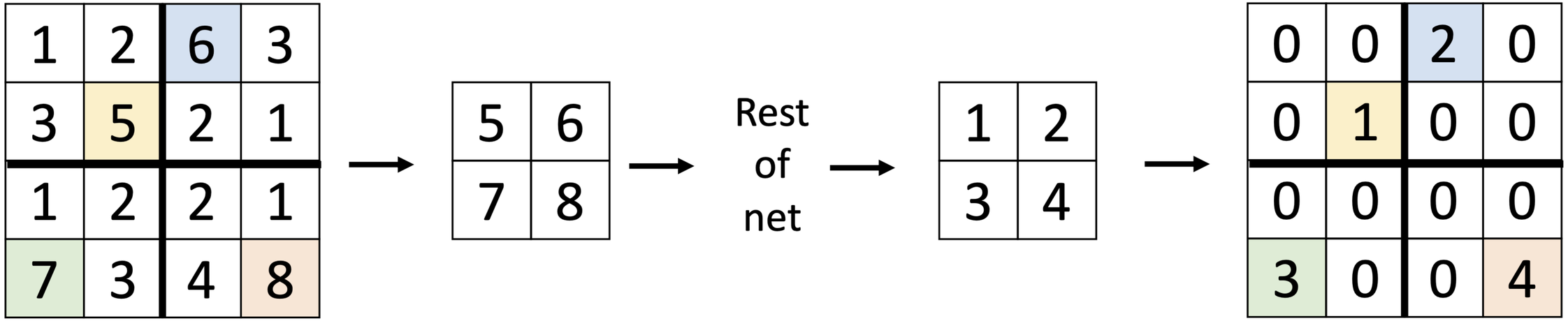
Output: C x 4 x 4

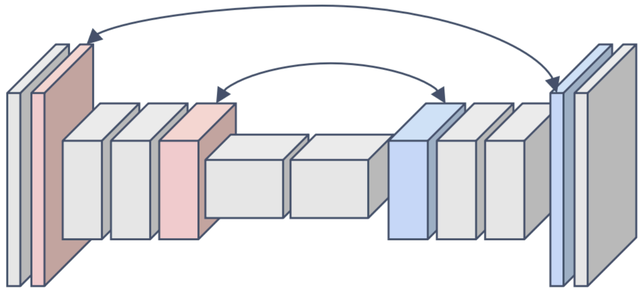
Use **three** closest neighbors in x and y to construct **cubic** approximations
(This is how we normally resize images!)

In-Network Upsampling: "Max Unpooling"

Max Pooling: Remember which position had the max

Max Unpooling: Place into remembered positions



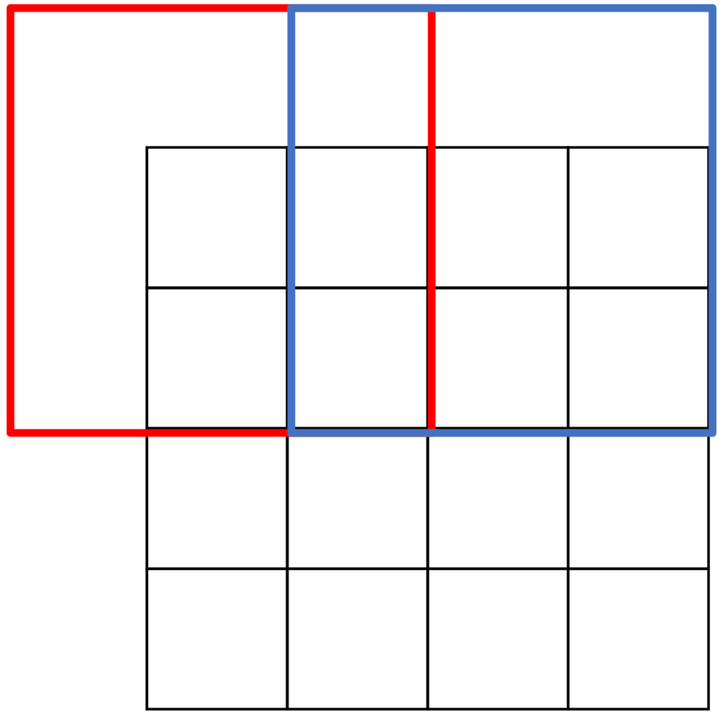


Pair each downsampling layer with an upsampling layer

Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

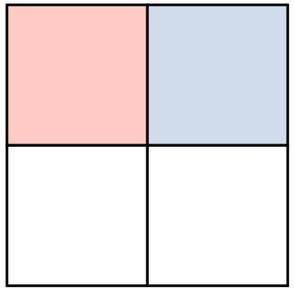
Learnable Upsampling: Transposed Convolution

Recall: Normal 3 x 3 convolution, stride 2, pad 1



Convolution with stride > 1 is “Learnable Downsampling”
Can we use stride < 1 for “Learnable Upsampling”?

Dot product between input and filter

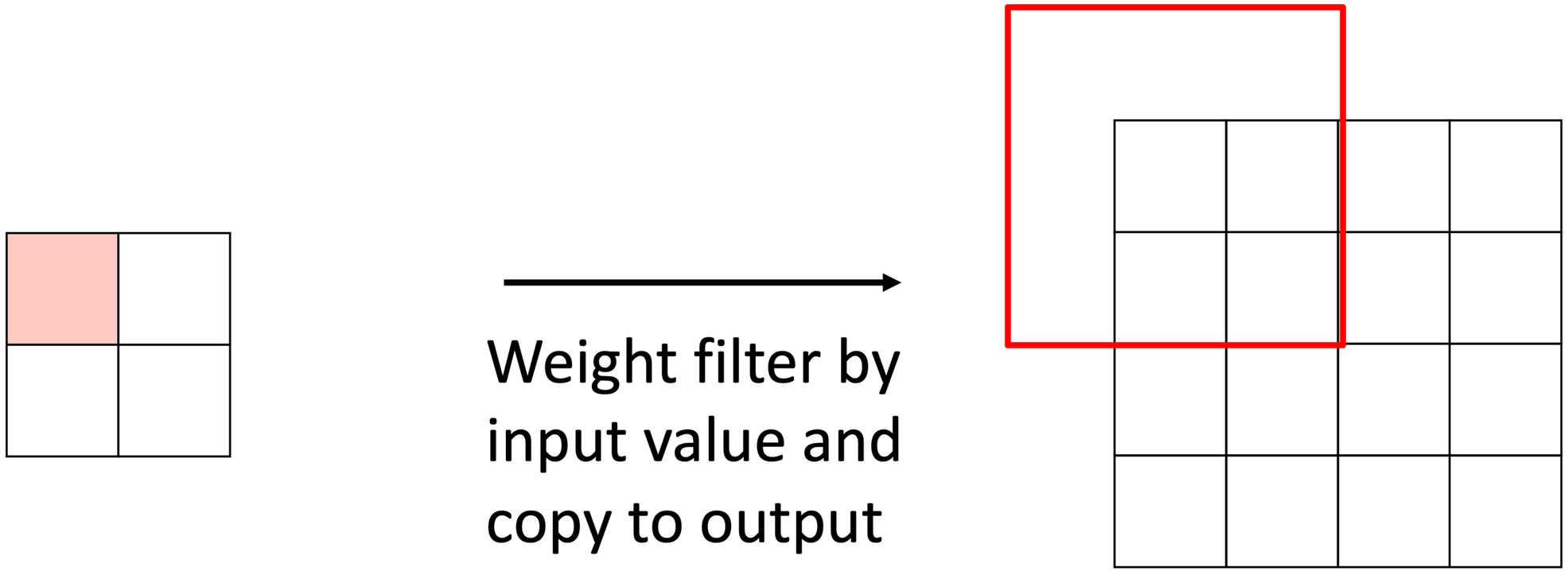


Input: 4 x 4

Output: 2 x 2

Learnable Upsampling: Transposed Convolution

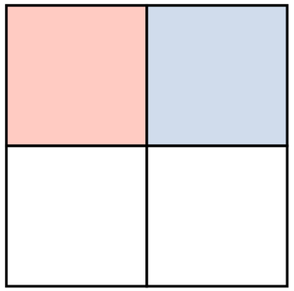
3 x 3 **convolution transpose**, stride 2



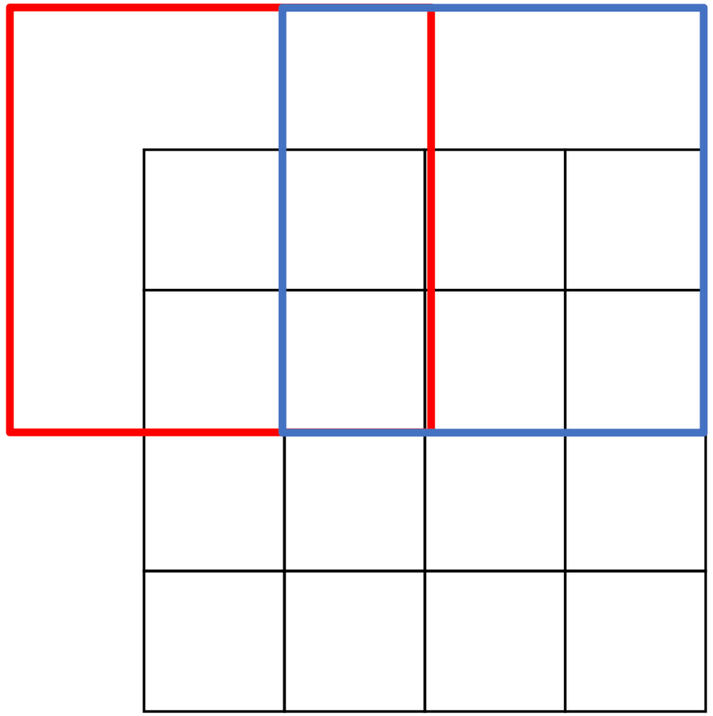
Learnable Upsampling: Transposed Convolution

3 x 3 **convolution transpose**, stride 2

Filter moves 2 pixels in output
for every 1 pixel in input



Weight filter by
input value and
copy to output



Input: 2 x 2

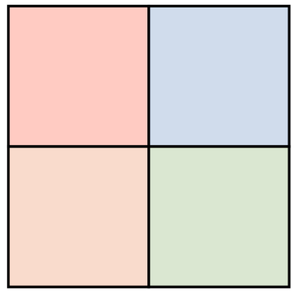
Output: 4 x 4

Learnable Upsampling: Transposed Convolution

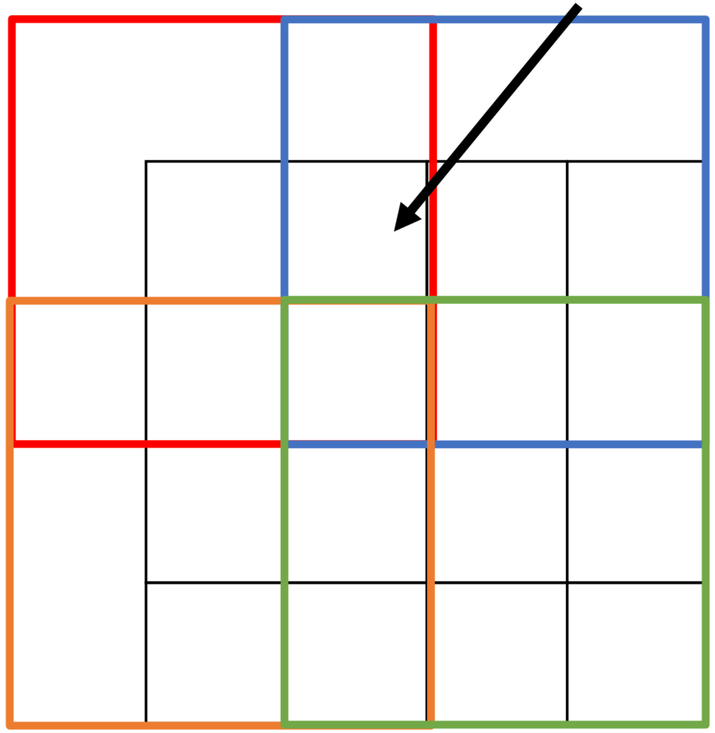
3 x 3 **convolution transpose**, stride 2

Sum where output overlaps

This gives 5x5 output – need to trim one pixel from top and left to give 4x4 output



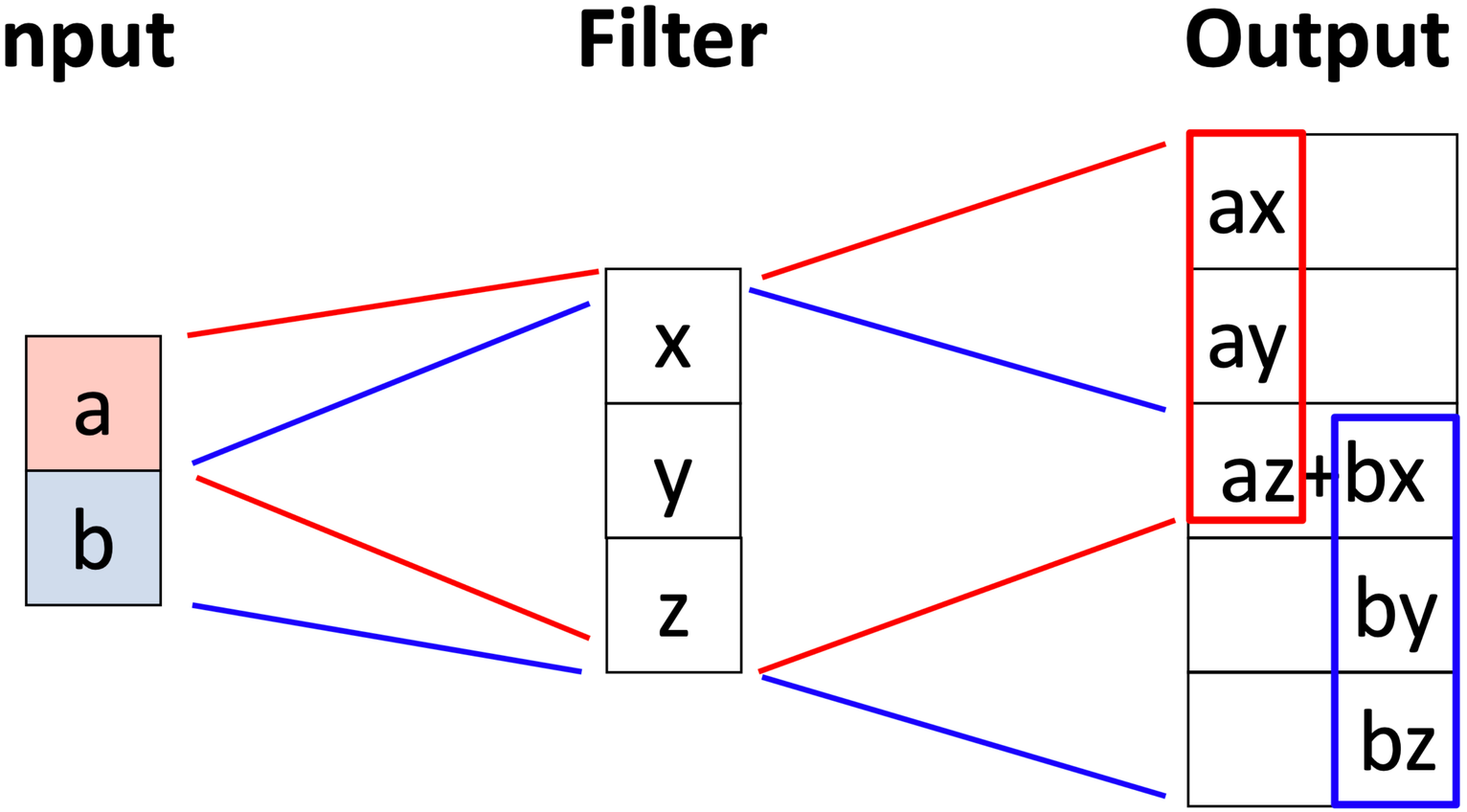
Weight filter by input value and copy to output



Input: 2 x 2

Output: 4 x 4

Transposed Convolution: 1D example



Output has copies of filter weighted by input

Stride 2: Move 2 pixels output for each pixel in input

Sum at overlaps

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

Transposed convolution multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

When stride=1, transposed conv is just a regular conv (with different padding rules)

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=2, padding=1

Transposed convolution multiplies by the transpose of the same matrix:

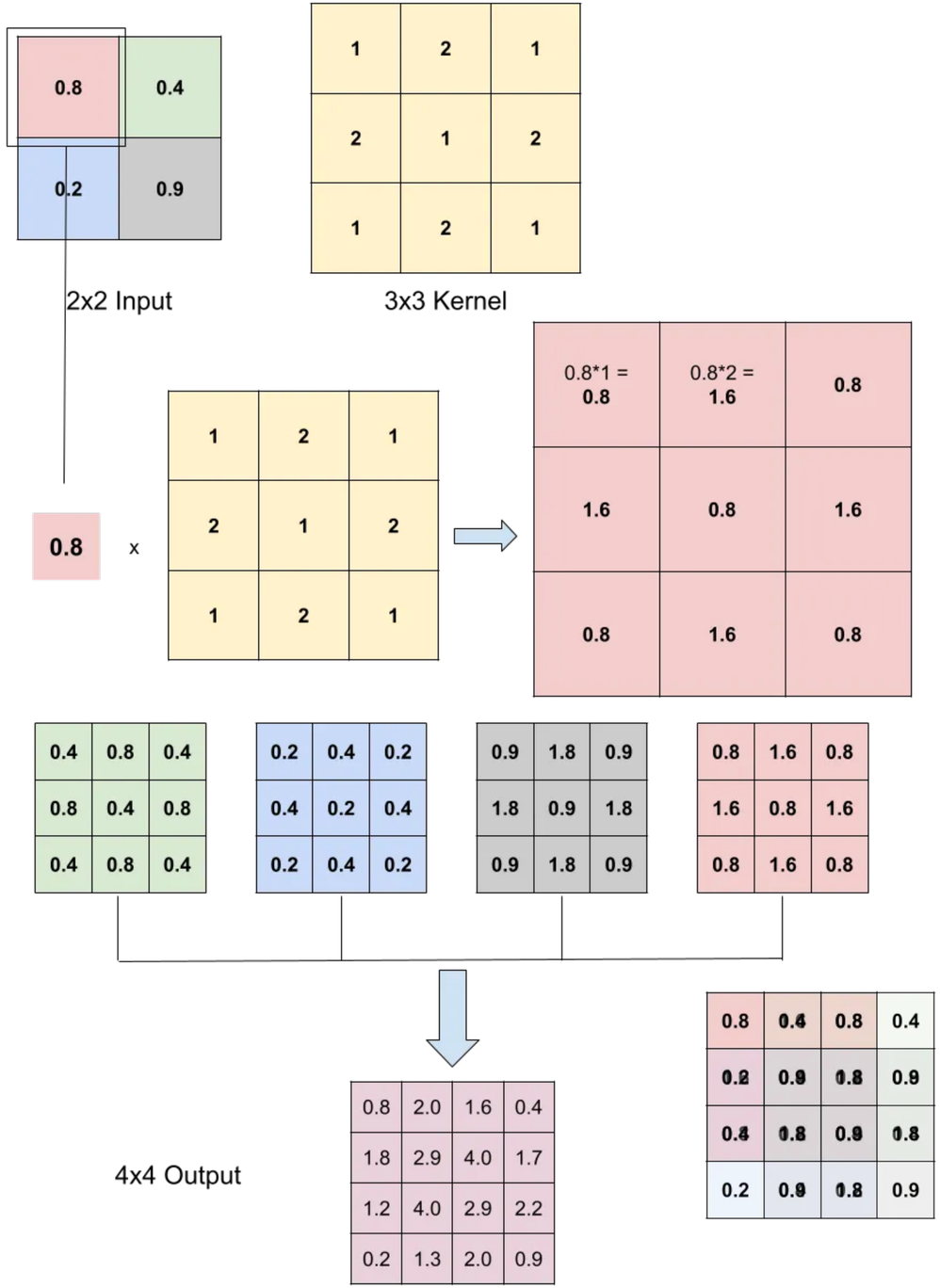
$$\vec{x} *^T \vec{a} = X^T \vec{a}$$

$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

When stride>1, transposed convolution cannot be expressed as normal conv

Conv2D Transpose

padding = 0



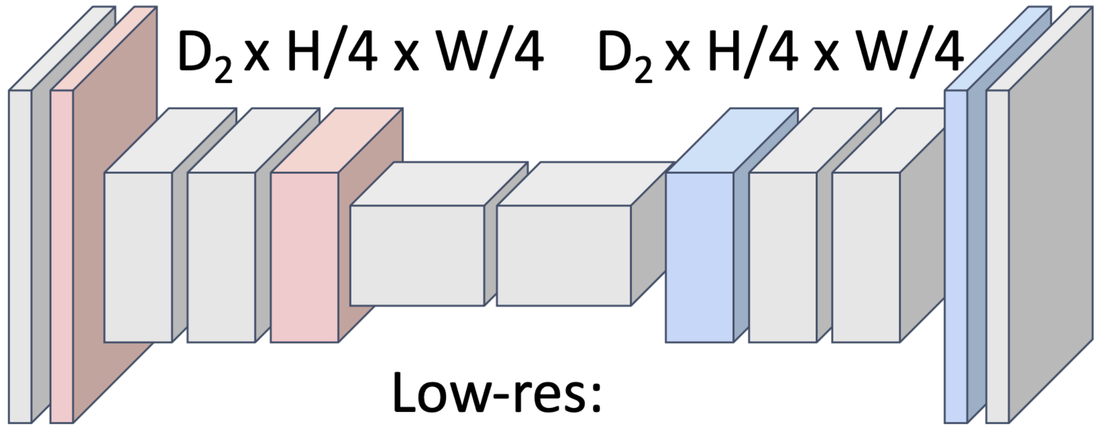
Semantic Segmentation: Fully Convolutional Network

Downsampling: Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling: linterpolation, transposed conv

Input:
3 x H x W



Predictions:
H x W

Loss function: Per-Pixel cross-entropy

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Dice Coefficient Loss

Dice Cofficient

- **Statistical Measure**: Evaluates similarity between predicted and ground-truth pixel sets.
- **Formula**:

$$\text{Dice Coefficient} = \frac{2 \times |A \cap B|}{|A| + |B|}$$

- A = set of predicted pixels
- B = set of ground-truth pixels

- **Interpretation**: 1 = Perfect overlap, 0 = No overlap.

$$\text{Dice Loss} = 1 - \text{Dice Coefficient}$$

Dice Coefficient Loss

Dice Cofficient

- Dice measures the overlap between entire predicted and ground-truth regions.
- It does not care how many pixels are background or object — it only cares about matching the shape.
- Equalizes focus between big and small regions.
- Even if a class occupies only 5% of the pixels, Dice will punish mistakes in that small region strongly.

Dice Coefficient Loss

backprop

$$\text{Dice Coefficient} = \frac{2\sum p_i g_i}{\sum p_i + \sum g_i}$$

```
loss = dice_loss(prediction, ground_truth)
loss.backward()  # backprop happens normally!
```

- p_i = predicted probability for pixel i (between 0 and 1).
- g_i = ground-truth label for pixel i (0 or 1).

Semantic Segmantation

Metrics

Imagine you are segmentice image having:

- background (90%)
- cat (8%)
- dog (2%)

Semantic Segmentation

Metrics: mIoU (mean IoU)

For a single class *c*:

$$\mathrm{IoU}_c = \frac{\mathrm{True\ Positive}_c}{\mathrm{True\ Positive}_c + \mathrm{False\ Positive}_c + \mathrm{False\ Negative}_c}$$

mIoU (mean IoU) is simply the average of IoUs over all classes:

$$\mathrm{mIoU} = \frac{1}{N_{\mathrm{classes}}} \sum_c \mathrm{IoU}_c$$

Semantic Segmentation

Metrics: Pixel Accuracy (PA)

- ratio of correctly predicted pixels to total pixels

$$\text{Pixel Accuracy} = \frac{\text{Number of Correct Pixels}}{\text{Total Number of Pixels}}$$

Semantic Segmentation

Metrics: Frequency Weighted IoU (FWIoU)

- weights each class's IoU by its **pixel frequency** (i.e., how common that class is)

$$\text{FWIoU} = \sum_{c} \left(\frac{n_c}{\text{Total pixels}} \times \text{IoU}_c \right)$$

n_c = number of ground truth pixels of class c

FCN

Pytorch implementation

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class SimpleFCN(nn.Module):
    def __init__(self, num_classes):
        super(SimpleFCN, self).__init__()

        # Encoder (Feature extraction)
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, padding=1)  # input RGB image
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, padding=1)

        # Decoder (Upsampling to original size)
        self.deconv1 = nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2)  # upsample
        self.deconv2 = nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2)
        self.deconv3 = nn.ConvTranspose2d(64, num_classes, kernel_size=2, stride=2)
```

```python
def forward(self, x):
    # Encoder
    x = F.relu(self.conv1(x))
    x = F.max_pool2d(x, 2)  # Downsample by 2
    x = F.relu(self.conv2(x))
    x = F.max_pool2d(x, 2)
    x = F.relu(self.conv3(x))
    x = F.max_pool2d(x, 2)

    # Decoder
    x = F.relu(self.deconv1(x))
    x = F.relu(self.deconv2(x))
    x = self.deconv3(x)  # Output logits for each class

    return x
```

FCN

with skip connections -> sharp boundaries

```python
class SimpleFCN8(nn.Module):
    def __init__(self, num_classes):
        super(SimpleFCN8, self).__init__()

        # Encoder
        self.conv1 = nn.Conv2d(3, 64, 3, padding=1)  # output: [B,64,H,W]
        self.conv2 = nn.Conv2d(64, 128, 3, padding=1)  # [B,128,H/2,W/2]
        self.conv3 = nn.Conv2d(128, 256, 3, padding=1)  # [B,256,H/4,W/4]

        # Decoder
        self.deconv1 = nn.ConvTranspose2d(256, 128, 2, stride=2)  # upsample H/4 -> H/2
        self.deconv2 = nn.ConvTranspose2d(128, 64, 2, stride=2)   # upsample H/2 -> H
        self.final = nn.Conv2d(64, num_classes, 1)  # 1x1 conv to output classes
```

```python
def forward(self, x):
    # Encoder
    x1 = F.relu(self.conv1(x))   # x1: [B,64,H,W]
    x1_pool = F.max_pool2d(x1, 2)  # [B,64,H/2,W/2]

    x2 = F.relu(self.conv2(x1_pool))  # x2: [B,128,H/2,W/2]
    x2_pool = F.max_pool2d(x2, 2)     # [B,128,H/4,W/4]

    x3 = F.relu(self.conv3(x2_pool))  # x3: [B,256,H/4,W/4]
    x3_pool = F.max_pool2d(x3, 2)     # [B,256,H/8,W/8]

    # Decoder
    x = self.deconv1(x3_pool)  # upsample to H/4
    x = x + x3                 # add skip connection (deepest feature map)

    x = self.deconv2(x)         # upsample to H/2
    x = x + x2                 # add skip connection (mid feature map)

    x = F.interpolate(x, scale_factor=2, mode='bilinear', align_corners=False)
    x = x + x1                 # add skip connection (early feature map)

    x = self.final(x)           # final classification layer

    return x
```

U-NET

U-Net: Convolutional Networks for Biomedical Image Segmentation

Olaf Ronneberger, Philipp Fischer, and Thomas Brox

Computer Science Department and BIOSS Centre for Biological Signalling Studies,
University of Freiburg, Germany
ronneber@informatik.uni-freiburg.de,
WWW home page: http://lmb.informatik.uni-freiburg.de/

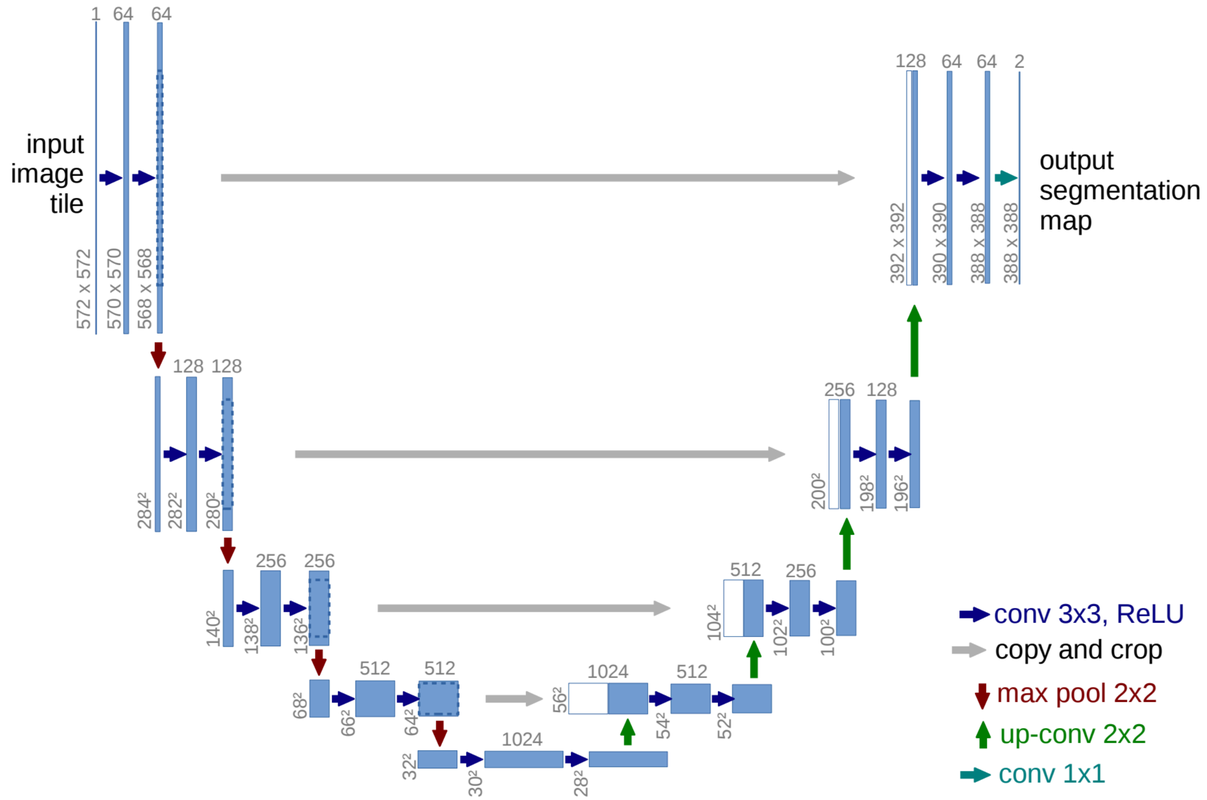


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.