HW4:
On Implementation of Object Detectors

Notes

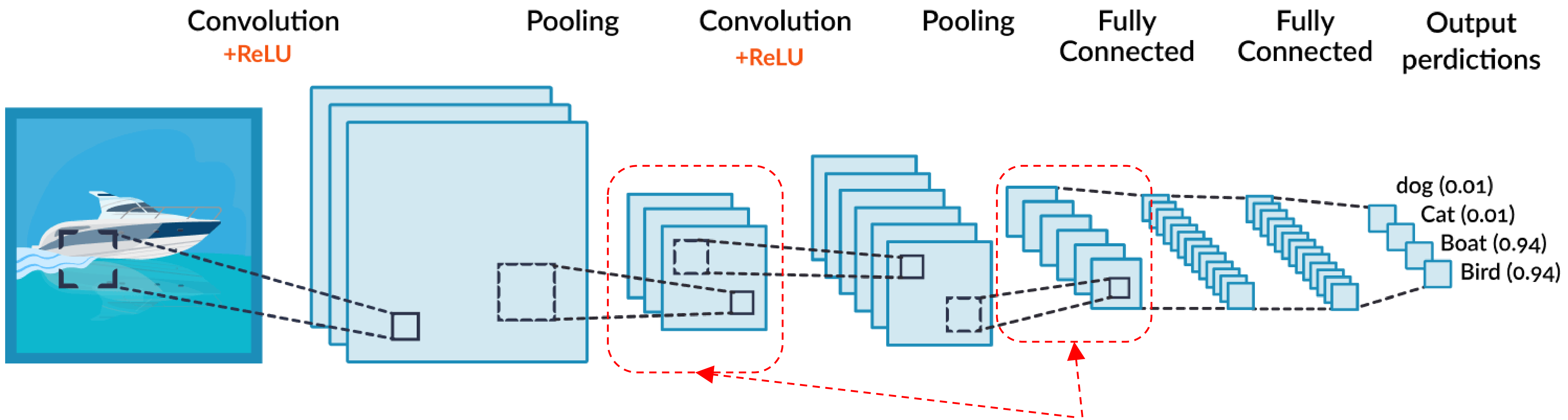
- Please make sure you read slide decks 16 – 17
- Please make sure you read chapter 50 of the textbook

Outline

- Recap of image classifiers
- Recap of object detectors
- Homework description (programming part)
- Homework description (written part)

Image classifier: Recap

- Given an "object-centric" image, a neural network classifier, like CNN classifiers, outputs the class label OR a probability vector of classes of the image



- Inside a CNN classifier, it generates multiple feature maps, which implicitly record spatial information within the image, e.g., where the object is.

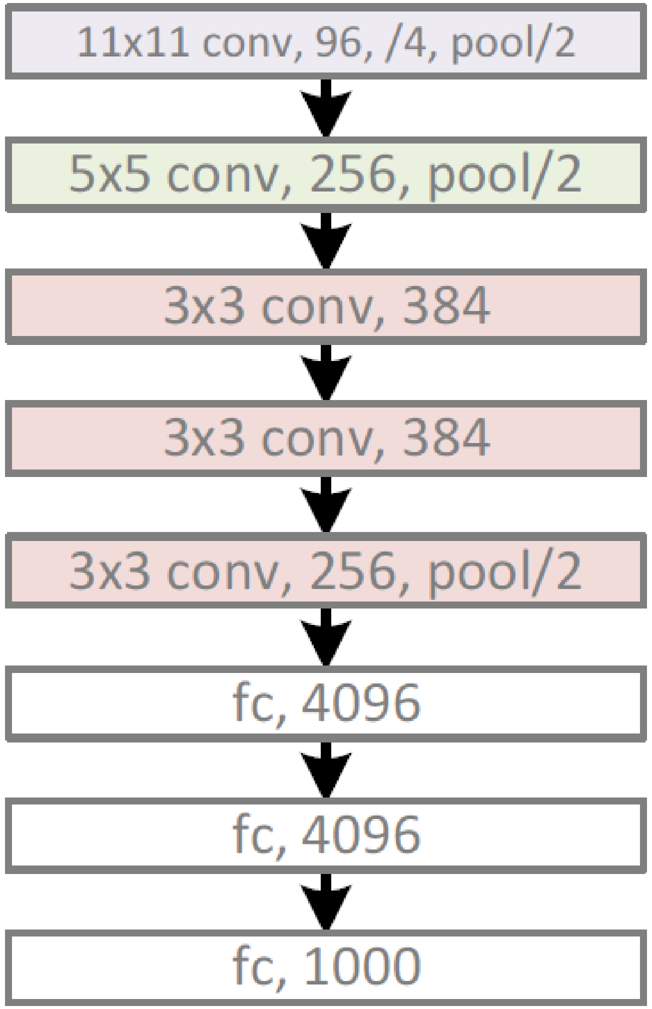
Image classifier: Recap

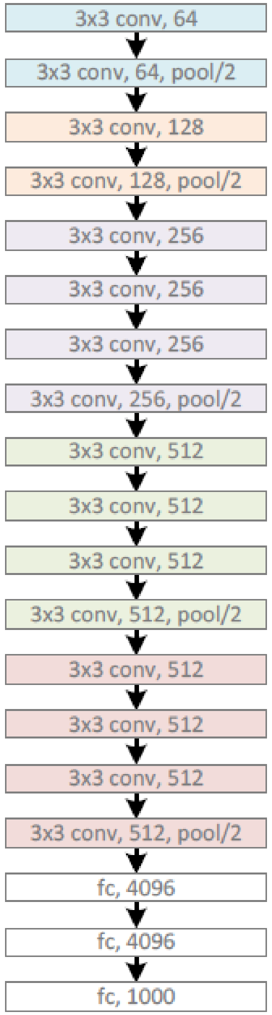
- AlexNet

[Krizhevsky et al., 2012]

- VGGnet

[Simonyan et al., 2015]



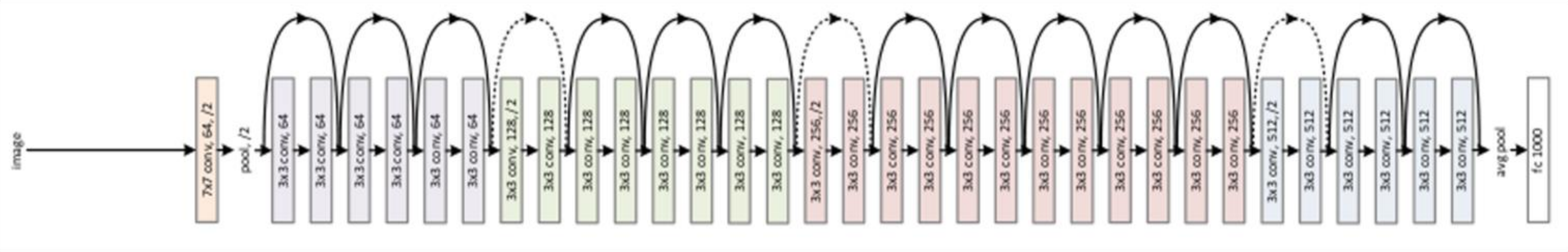


- A block: computation
- Edge: nodes/tensors

Image classifier: Recap

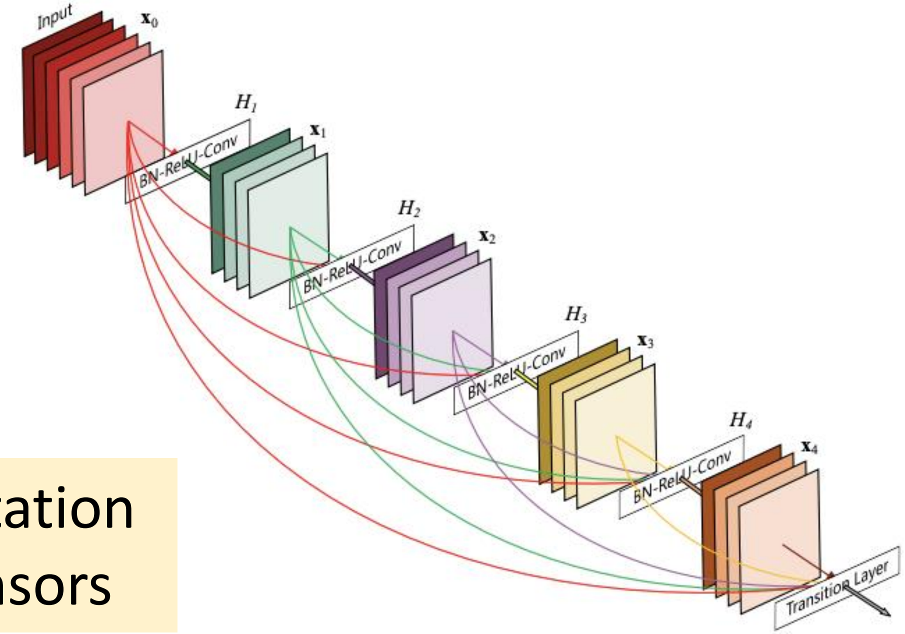
- ResNet

[He et al, 2016]



- DenseNet

[Huang et al, 2017]



- A block: computation
- Edge: nodes/tensors

Outline

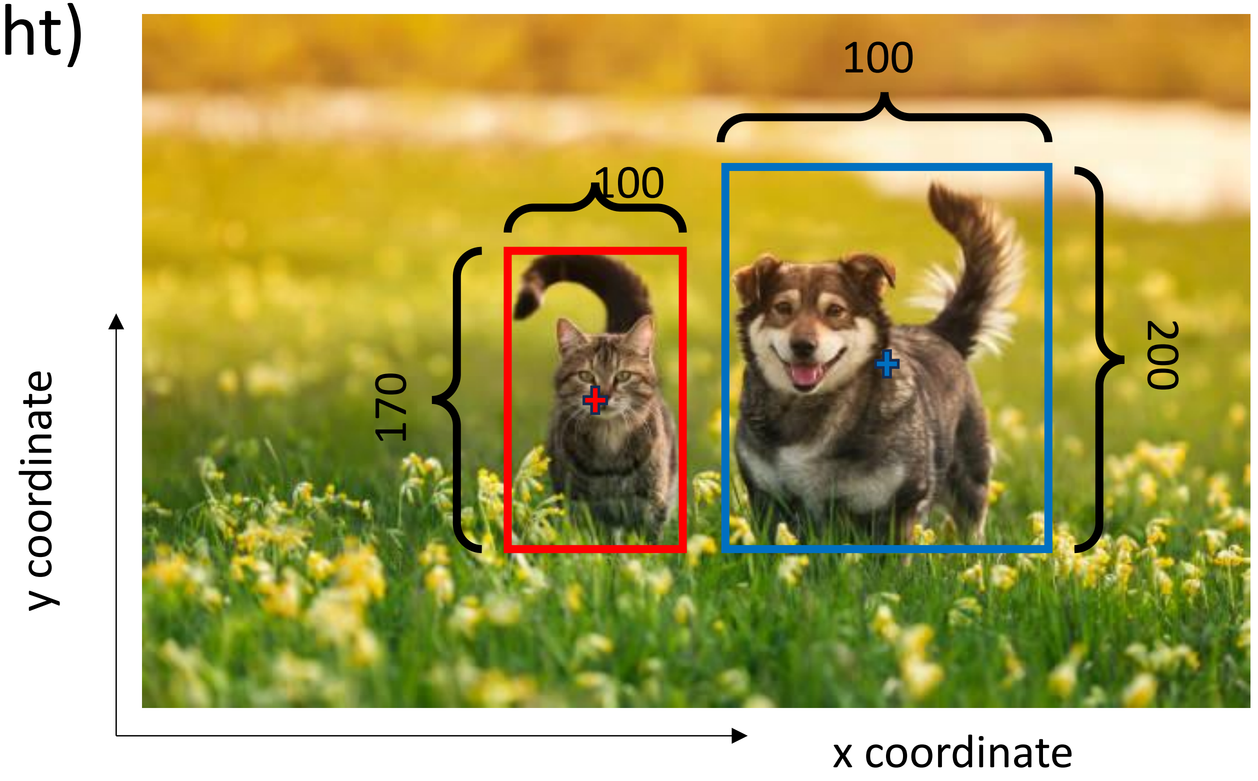
- Recap of image classifiers
- Recap of object detectors
- Homework description (coding part)
- Homework description (written part)

Object detection: Recap

- Given a "scene-centric" image, an object detector outputs a set of bounding boxes, each containing [class label, x-center, y-center, width, height]

- The image is 600 (width) x 400 (height)
- The output should be like:
  - [cat, 250, 180, 100, 170]
  - [dog, 400, 200, 170, 200]

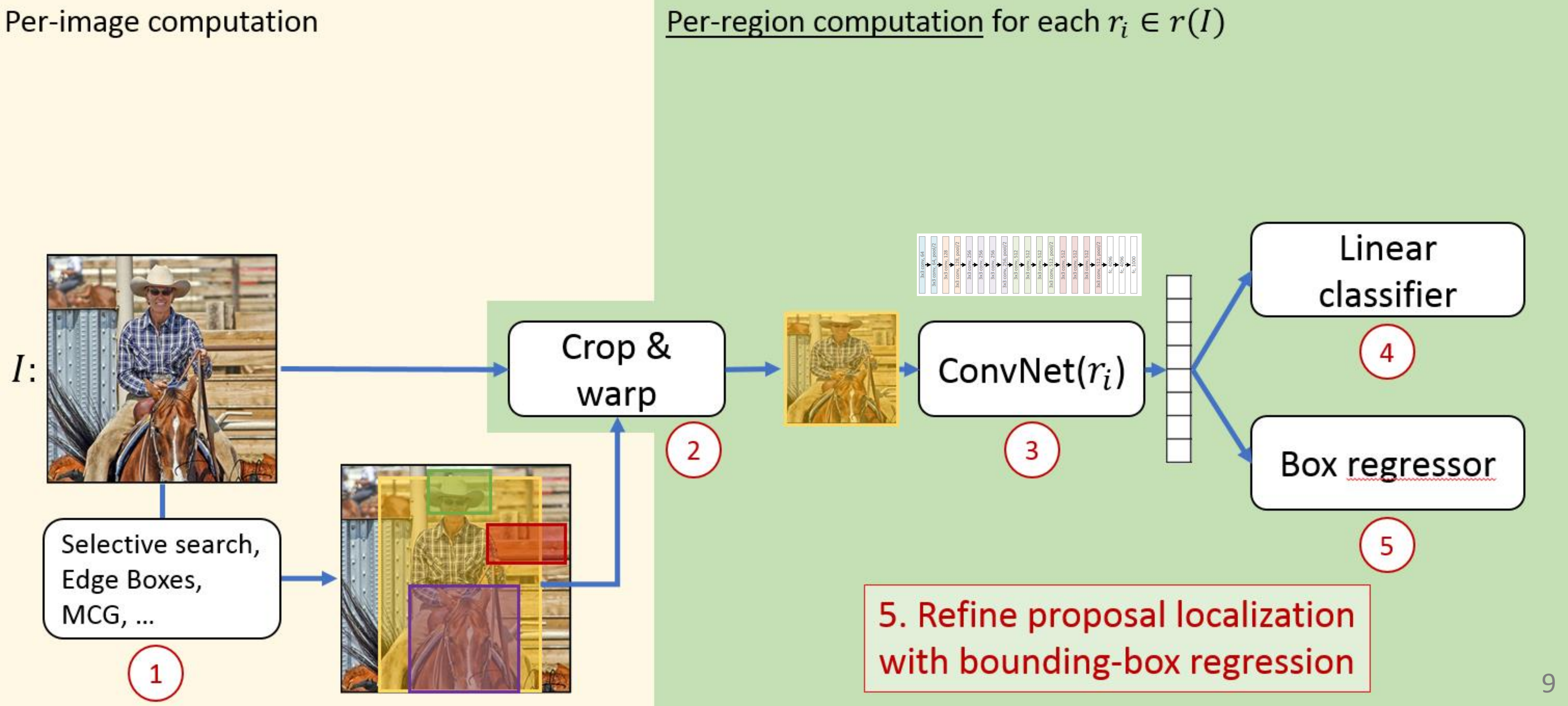
- **This can be done in two stages:**
  - **Create object proposals**
  - **Classify/regress each object proposal**



[Girshick et al., Rich feature hierarchies for accurate object detection and semantic segmentation, CVPR 2014]

R-CNN

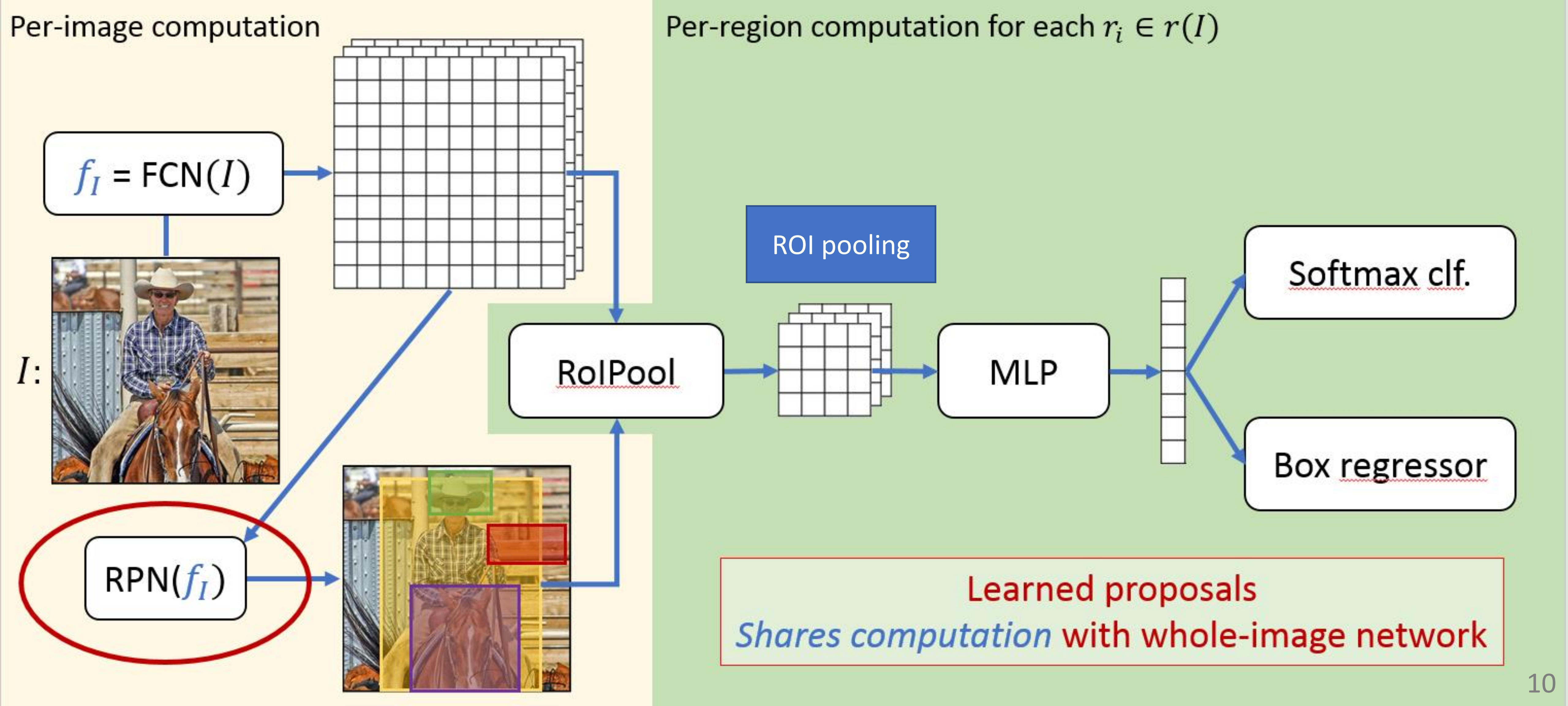
[Girshick, CVPR 2019 tutorial]



Faster R-CNN

[Ren et al., Faster r-cnn: Towards real-time object detection with region proposal networks, NIPS 2015]

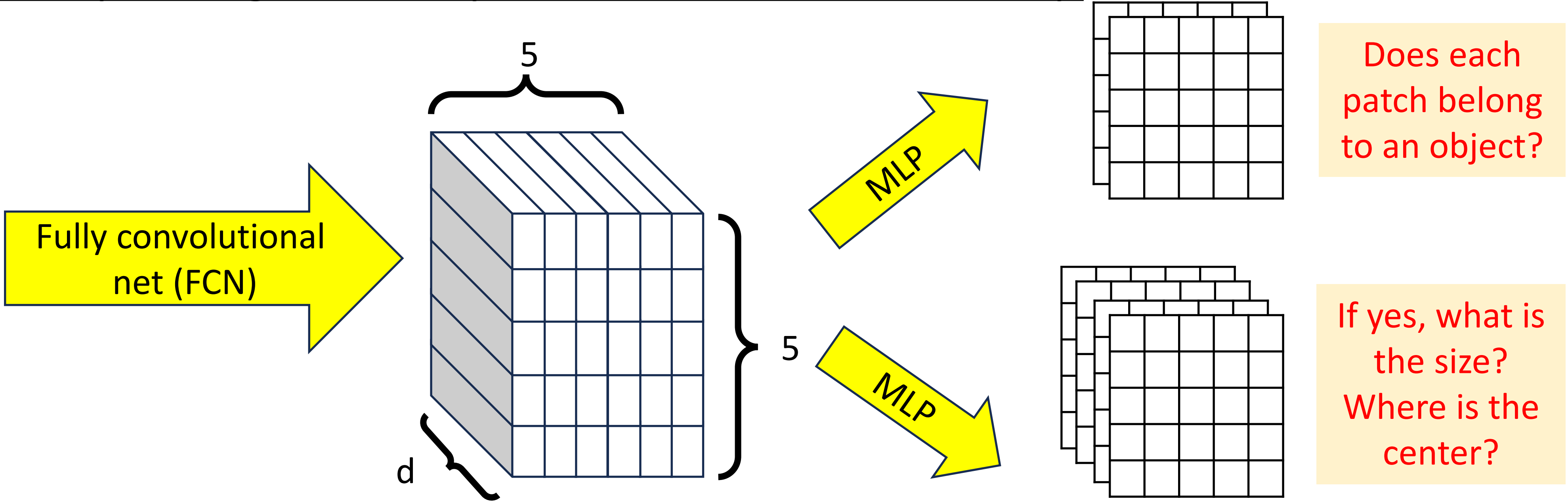
[Girshick, CVPR 2019 tutorial]



Object detection (object proposals): Recap

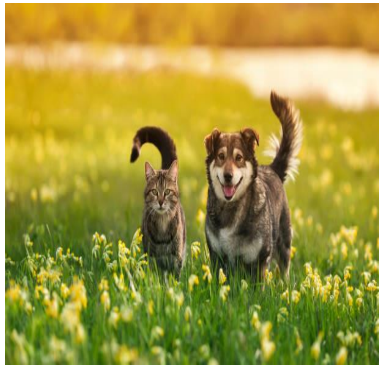
- The region proposal network (RPN) is proposed in this paper: https://arxiv.org/pdf/1506.01497

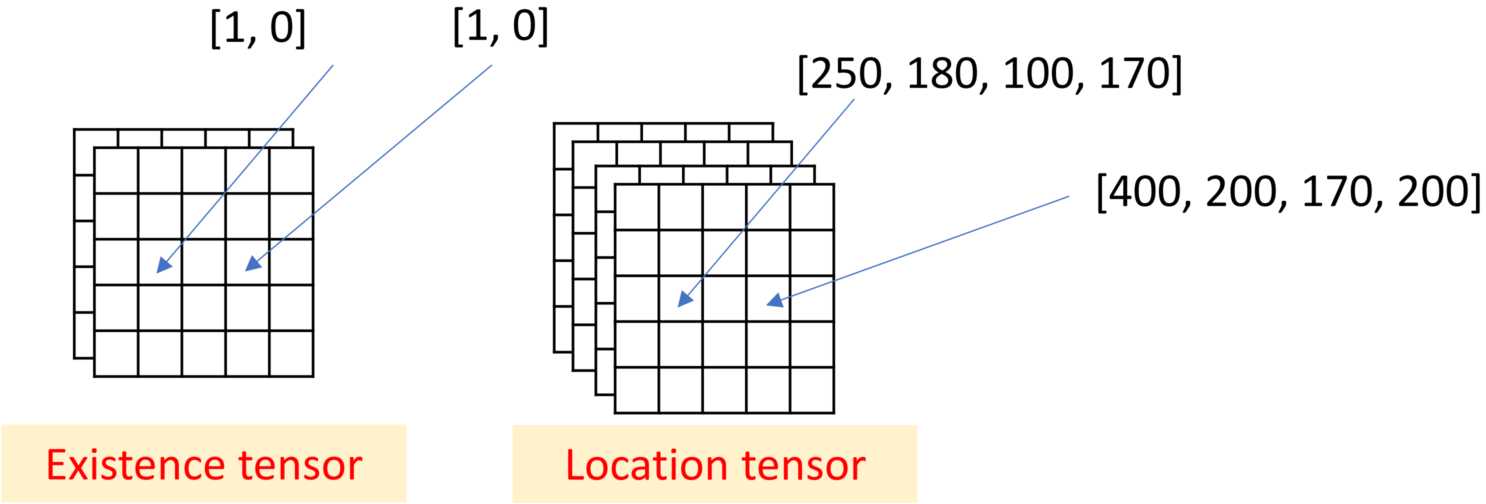
- The basic idea is to predict the existence of objects and their corresponding locations <u>at each spatial grid (i.e., patch) of the feature map</u>



Object detection (object proposals): Recap

- For example, if at each patch location, the model outputs the following vectors

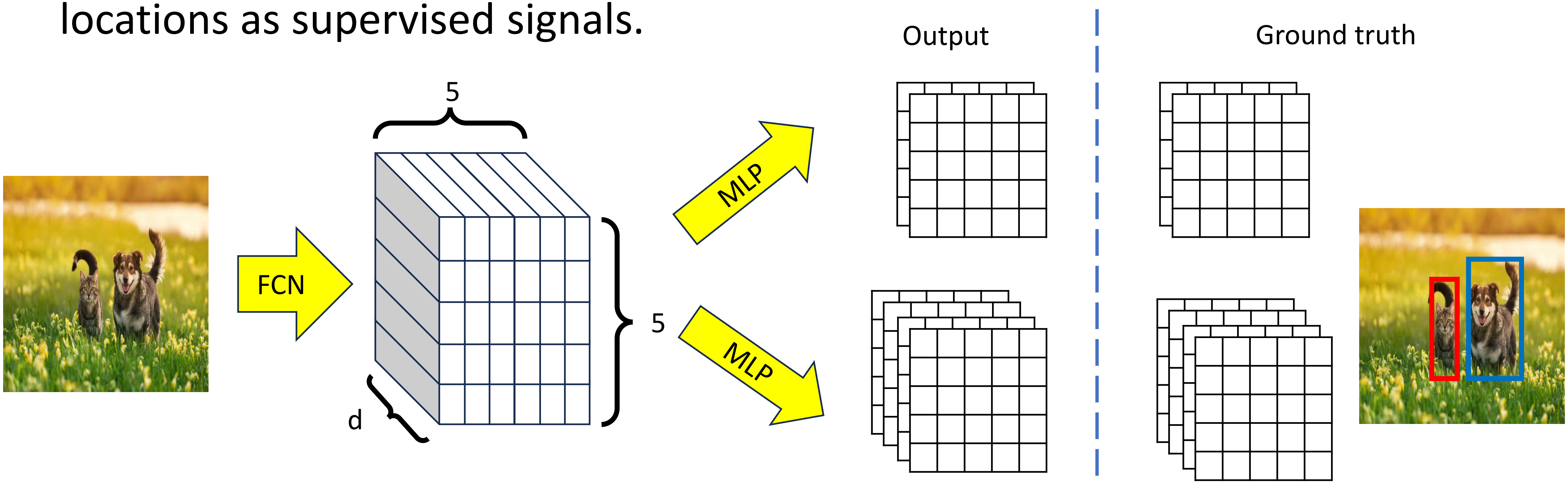




- Then we can write code to **read** this information and further output:
  - [250, 180, 100, 170]
  - [400, 200, 170, 200]

Object detection (object proposals): Recap

- For the neural network (FCN, MLPs) to accurately output the object locations, we must train it using stochastic gradient descent, using ground truth object locations as supervised signals.



- The ground truth tensors encode the ideal output tensors

Outline

- Recap of image classifiers
- Recap of object detectors
- Homework description (coding part)
- Homework description (written part)

Homework description

- You will focus on a ***simplified*** region proposal network (RPN).

- You are NOT asked to build it and train it.

- Instead, you are asked to:
  - Create the <u>ground truth (GT) tensors</u>, given the ground truth (GT) object locations
  - Given the <u>output tensors</u>, read them and output a list of object proposal locations

- Along with the implementation, you will also experience
  - Anchors
  - Non-Maximum Suppression (NMS)

Outline

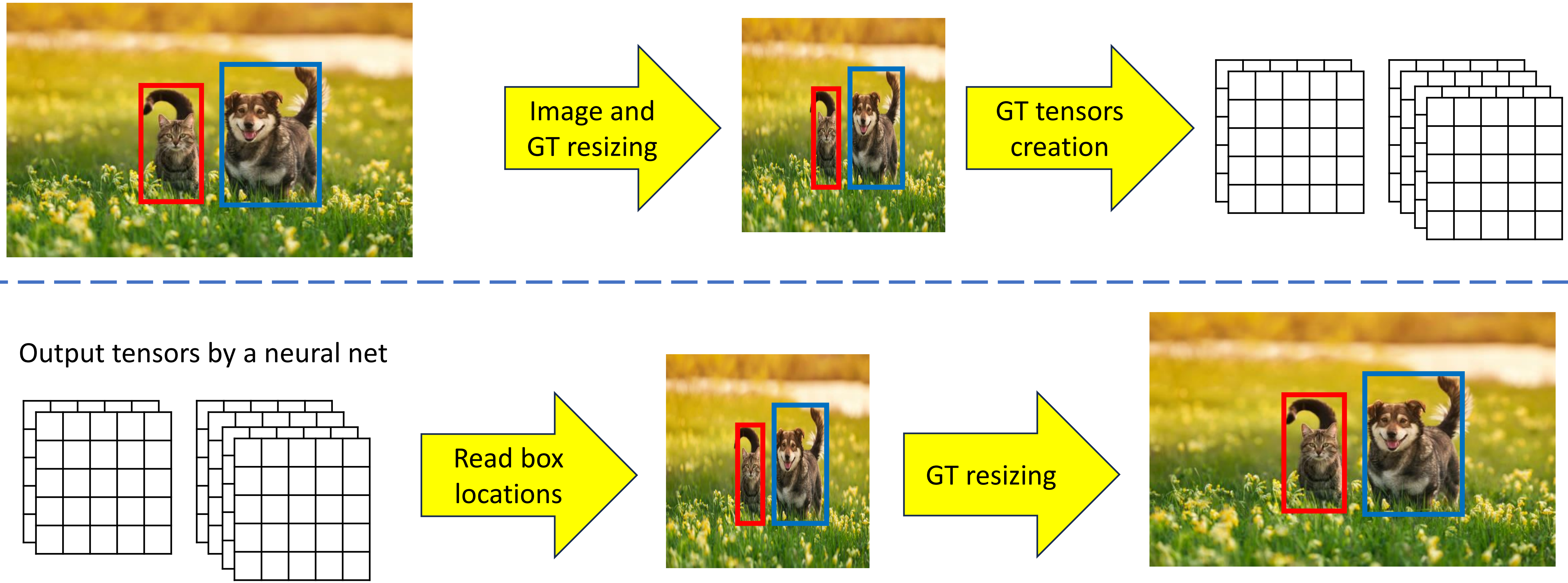
- Recap of image classifiers
- Recap of object detectors
- Homework description (coding part)
    - Naïve implementation
    - Implementation with anchors and offsets
    - Non-Maximum Suppression (NMS)
- Homework description (written part)

Caution!

- The image size and patch/grid size in this slide deck are examples. Please follow what is in the code to implement your answer.

Q1: Naïve implementation

- In Q1, you are to implement (or experience) the following steps



Q1: Naïve implementation: resizing

- Image and GT resizing

GT locations [u-center, v-center, width, height]
- [250, 180, 100, 170]
- [400, 200, 170, 200]

Image and GT resizing

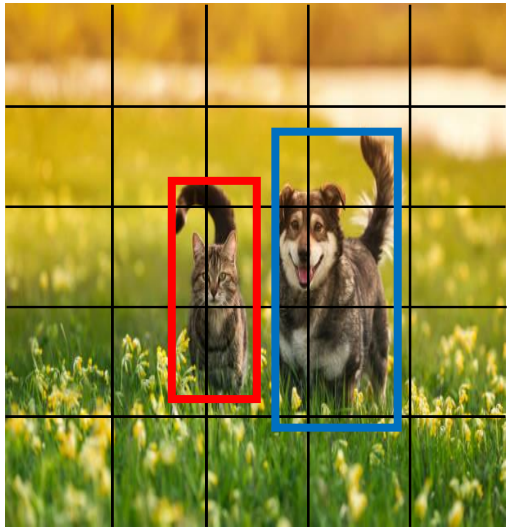
H

W

Resized GT locations
- Locations should be divided by the original image sizes and then multiplied by the new sizes

Q1: Naïve implementation: encoding

- Suppose the resized image is 200-by-200, and the feature map spatial resolution is 5-by-5, each patch is 40-by-40 of the resized image
- Now you need to create the following existence tensor with two channels:



Resized GT locations
- [83.3, 90, 33.3, 85]
- [133.3, 100, 56.7, 100]

GT tensors creation

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	1	1	0

Existence: Yes

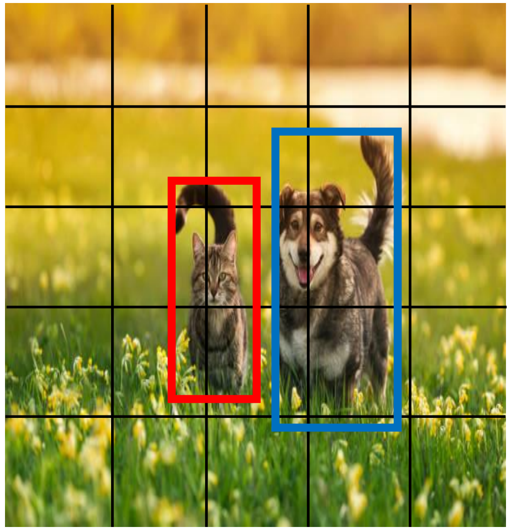
1	1	1	1	1
1	0	0	0	1
1	0	0	0	1
1	0	0	0	1
1	1	0	0	1

Existence: No

These two matrices sum to 1 at each patch location

Q1: Naïve implementation: encoding

- Basically, if a patch overlaps with ANY of the resized GT box, you set the "existence: Yes" to 1; otherwise, 0.



Resized GT locations
- [83.3, 90, 33.3, 85]
- [133.3, 100, 56.7, 100]

GT tensors creation

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	1	1	0

Existence: Yes

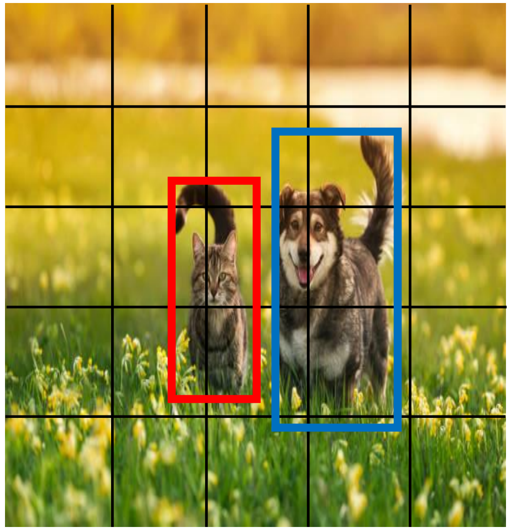
1	1	1	1	1
1	0	0	0	1
1	0	0	0	1
1	0	0	0	1
1	1	0	0	1

Existence: No

These two matrices sum to 1 at each patch location

Q1: Naïve implementation: encoding

- You also need to create the following <u>location tensor</u> with four channels:



Resized GT locations

- [83.3, 90, 33.3, 85]
- [133.3, 100, 56.7, 100]

GT tensors creation

x-center

0	0	0	0	0
0	83.3	?	133.3	0
0	83.3	?	133.3	0
0	83.3	?	133.3	0
0	0	133.3	133.3	0

y-center

0	0	0	0	0
0	90	?	100	0
0	90	?	100	0
0	90	?	100	0
0	0	100	100	0

0	0	0	0	0
0	33.3	?	56.7	0
0	33.3	?	56.7	0
0	33.3	?	56.7	0
0	0	56.7	56.7	0

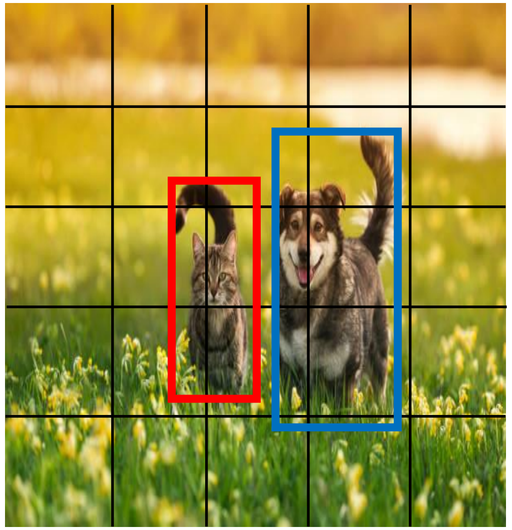
width

0	0	0	0	0
0	85	?	100	0
0	85	?	100	0
0	85	?	100	0
0	0	100	100	0

height

Q1: Naïve implementation: encoding

- If a patch overlaps with one resized GT box, you record the GT box information inside the location tensor
- What if a patch overlaps with multiple boxes?



Resized GT locations
- [83.3, 90, 33.3, 85]
- [133.3, 100, 56.7, 100]

GT tensors creation

x-center

0	0	0	0	0
0	83.3	?	133.3	0
0	83.3	?	133.3	0
0	83.3	?	133.3	0
0	0	133.3	133.3	0

y-center

0	0	0	0	0
0	90	?	100	0
0	90	?	100	0
0	90	?	100	0
0	0	100	100	0

0	0	0	0	0
0	33.3	?	56.7	0
0	33.3	?	56.7	0
0	33.3	?	56.7	0
0	0	56.7	56.7	0

width

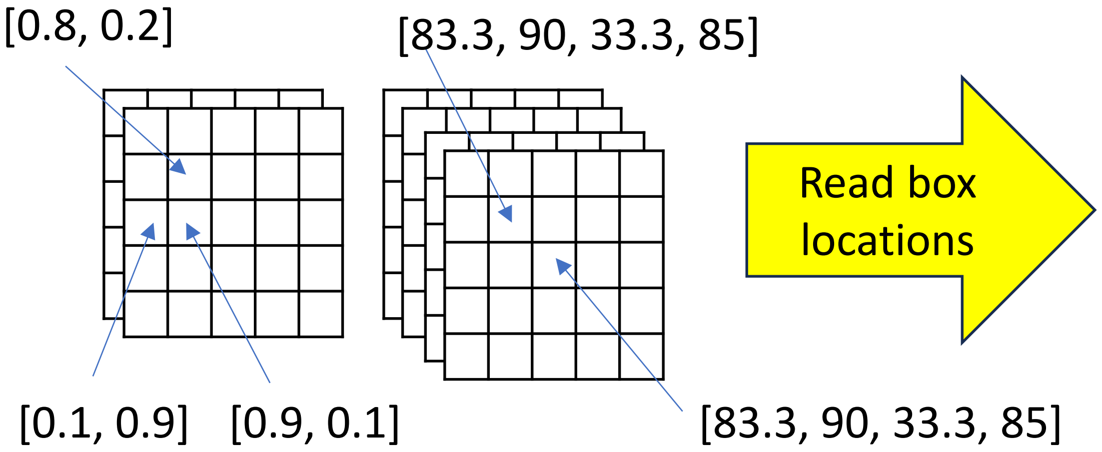
0	0	0	0	0
0	85	?	100	0
0	85	?	100	0
0	85	?	100	0
0	0	100	100	0

height

Q2: Naïve implementation: decoding

- Suppose an RPN, after training, can output both the existence and location tensors close to the ideal tensors, then we can read the object locations.
- Please note that in the “Existence: Yes” matrix, the outputted value may NOT be exactly 0 or 1, but a value within [0, 1]. We typically call it “confidence.”
- For patches whose “Existence: Yes” values > a threshold (e.g., 0.2), we will read out its corresponding location and output a box

Output tensors by a neural net



[x-center, y-center, width, height, “Existence: Yes” value]

[83.3, 90, 33.3, 85, 0.9]

[83.3, 90, 33.3, 85, 0.8]

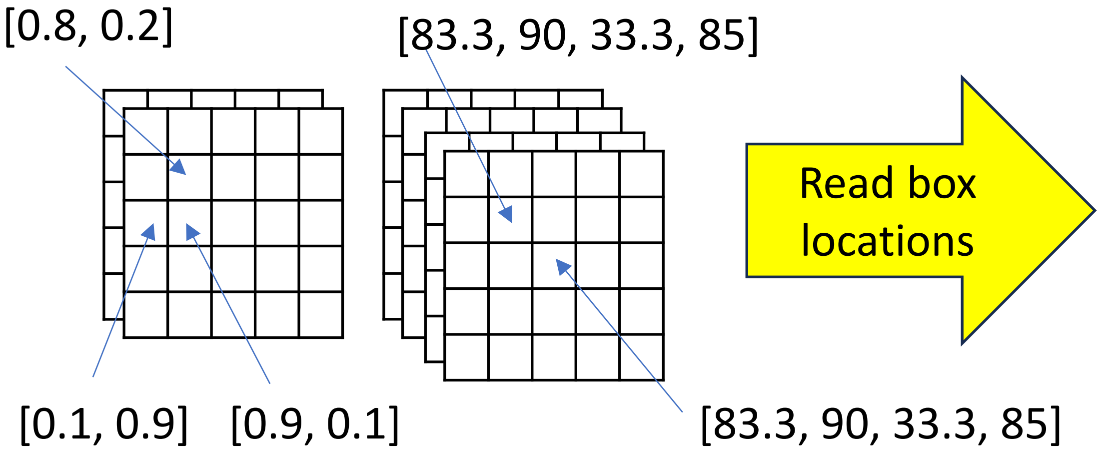
…

…

Q2: Naïve implementation: decoding

- That is, for the same GT box, we may output multiple boxes with potentially different **confidences** (i.e., “**Existence: Yes” values**)

Output tensors by a neural net



[x-center, y-center, width, height, “Existence: Yes” value]
[83.3, 90, 33.3, 85, 0.9]
[83.3, 90, 33.3, 85, 0.8]
…
…

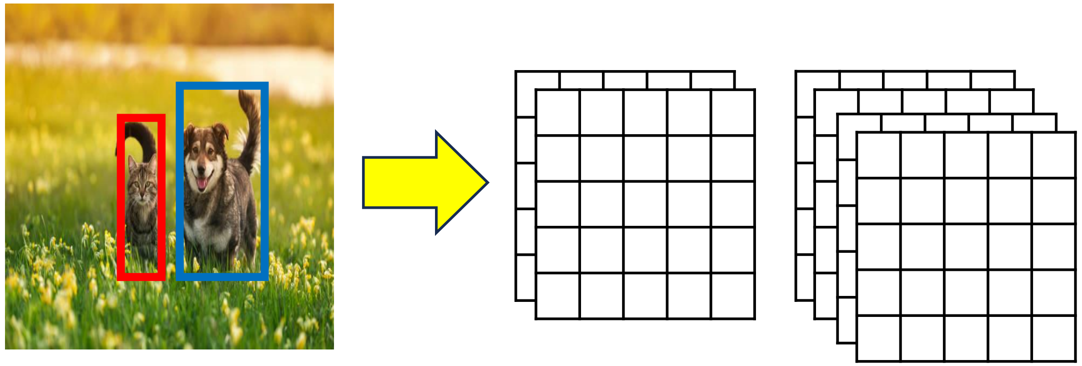
Outline

- Recap of image classifiers
- Recap of object detectors
- Homework description (coding part)
    - Naïve implementation
    - Implementation with anchors and offsets
    - Non-Maximum Suppression (NMS)
- Homework description (written part)

Q3: Advanced implementation

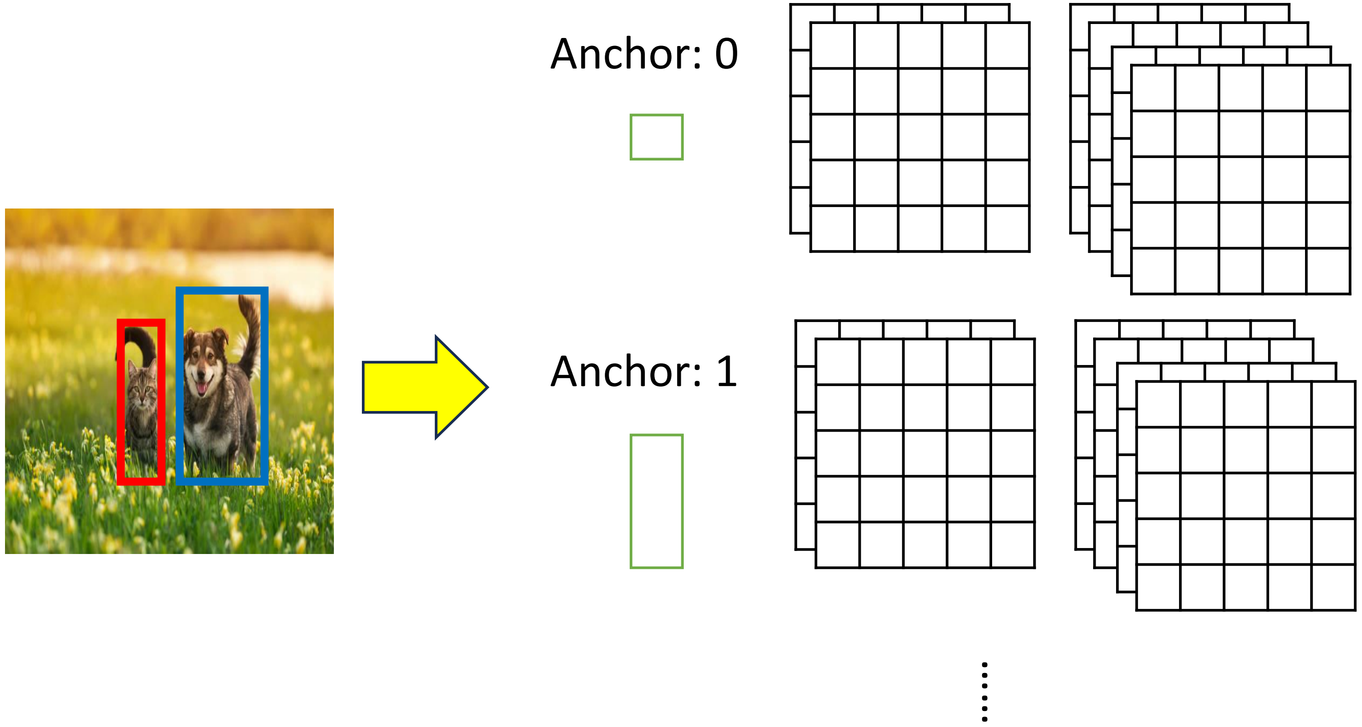
- Very much following the same steps in Q1, but you are now to implement (and experience) the use of anchors

Naïve implementation



“Existence: Yes” is 1 if a patch overlaps GT boxes

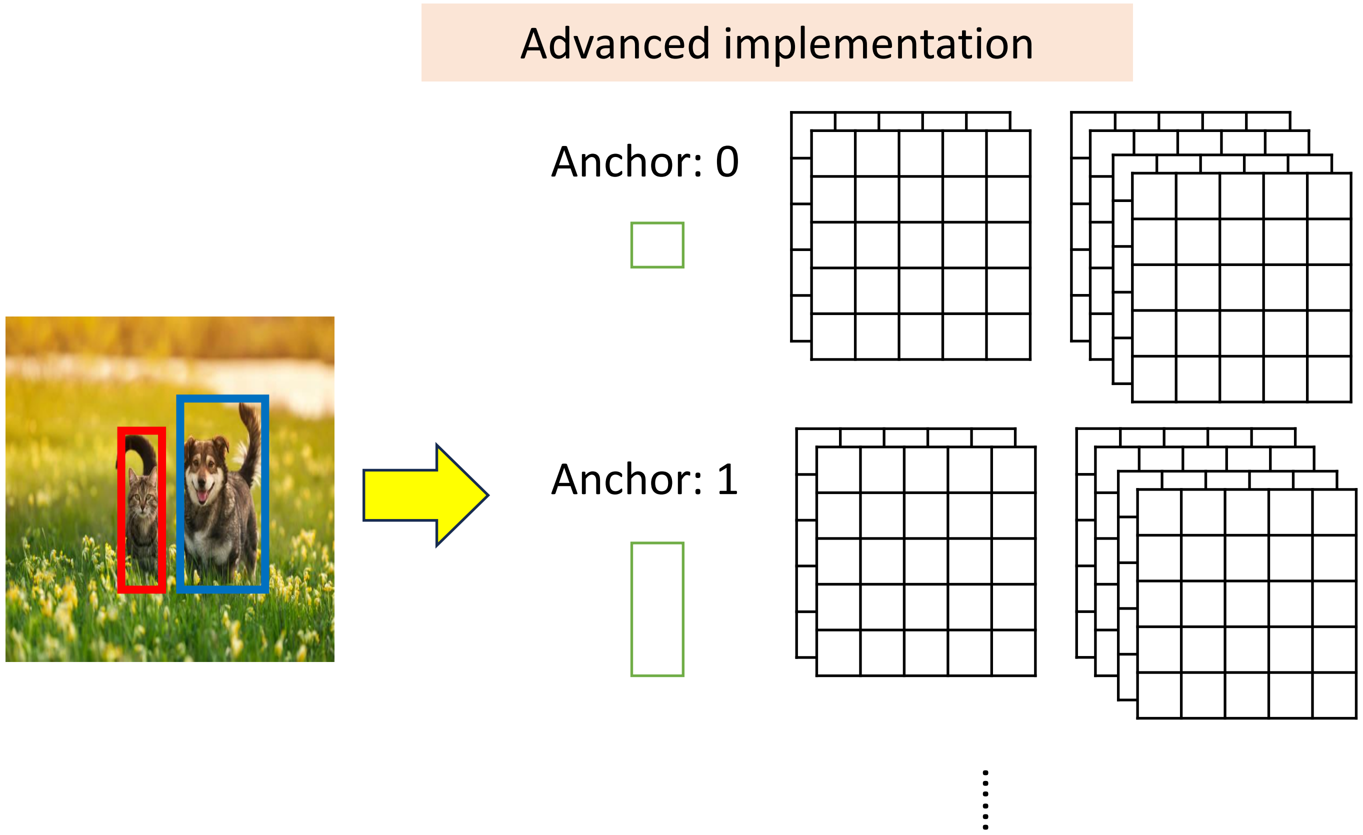
Advanced implementation



Q3: Advanced implementation

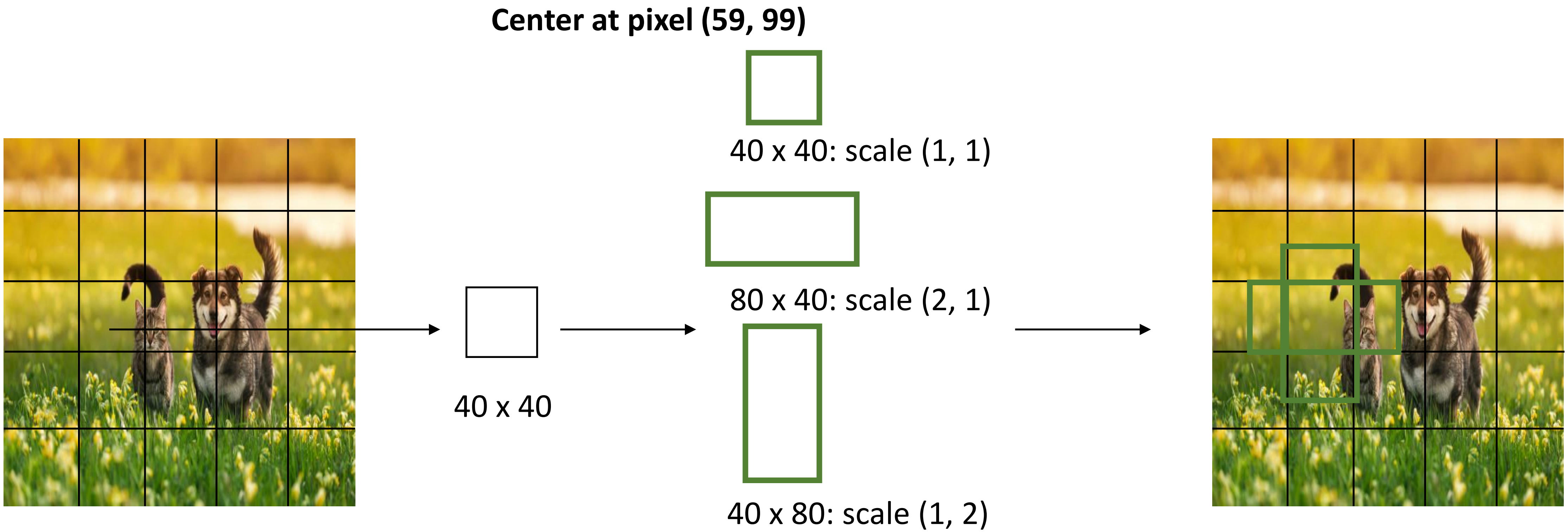
- That is, for each anchor, you need to create the corresponding existence and location tensors.

- If there are ***K*** anchors, you will need to create ***K*** corresponding existence and location tensors.



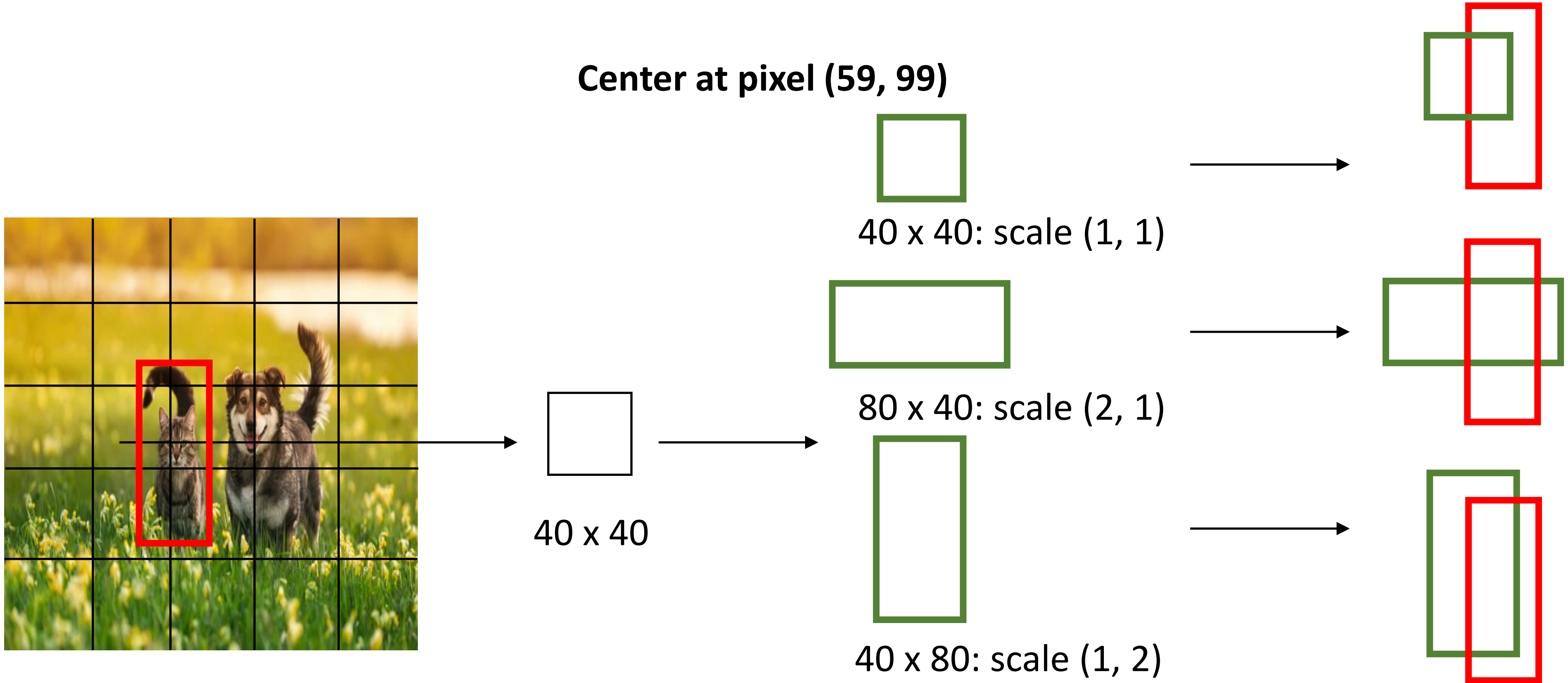
Q3: Advanced implementation

- An anchor is a <u>specific box shape</u> centered at each patch
- For example, if the image is 200-by-200, a patch is 40-by-40
- We can then consider different anchors (box shapes) by scaling the patch size



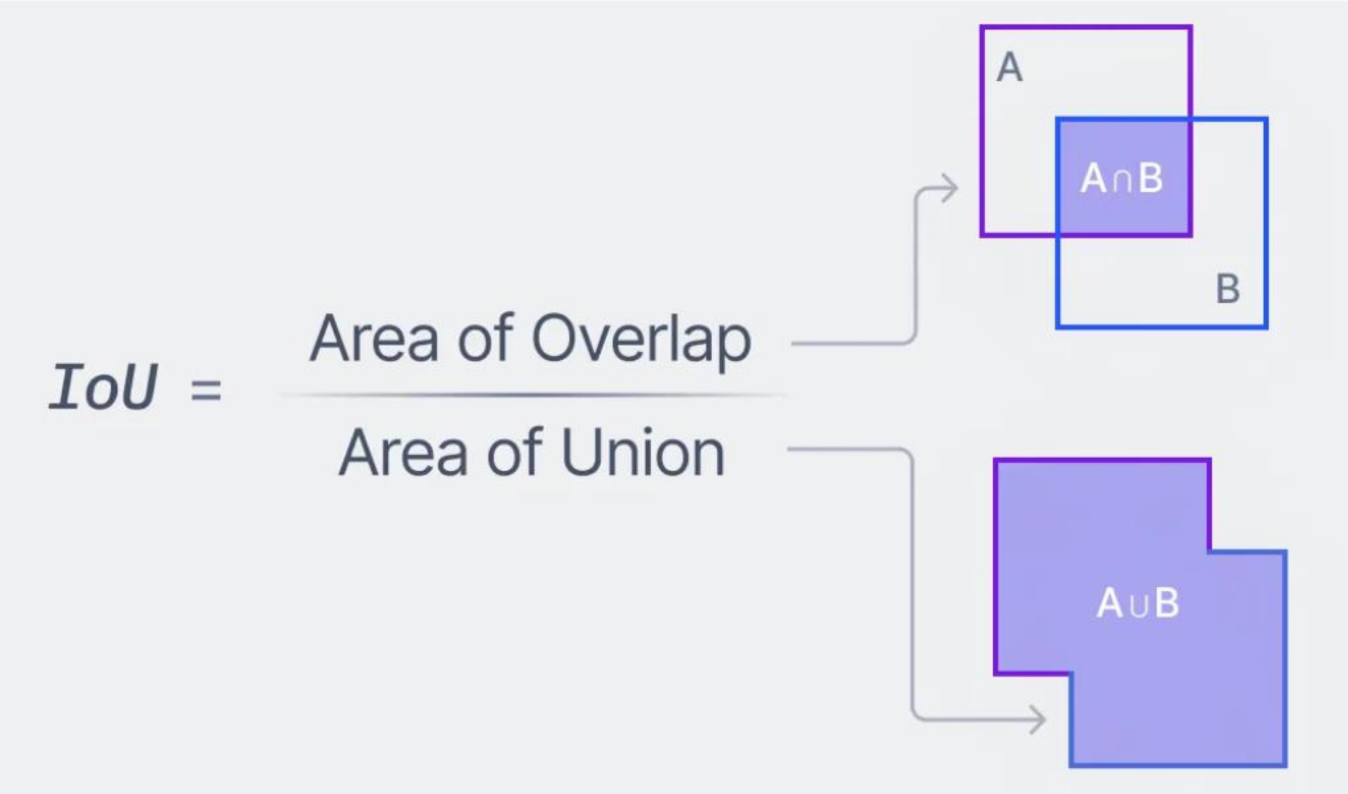
Q3: Advanced implementation: encoding

- If a GT object overlaps a patch (at this step, anchors are not used yet)
- We need to choose ONE anchor out of *K*, and set the corresponding "Existence: Yes" to be 1



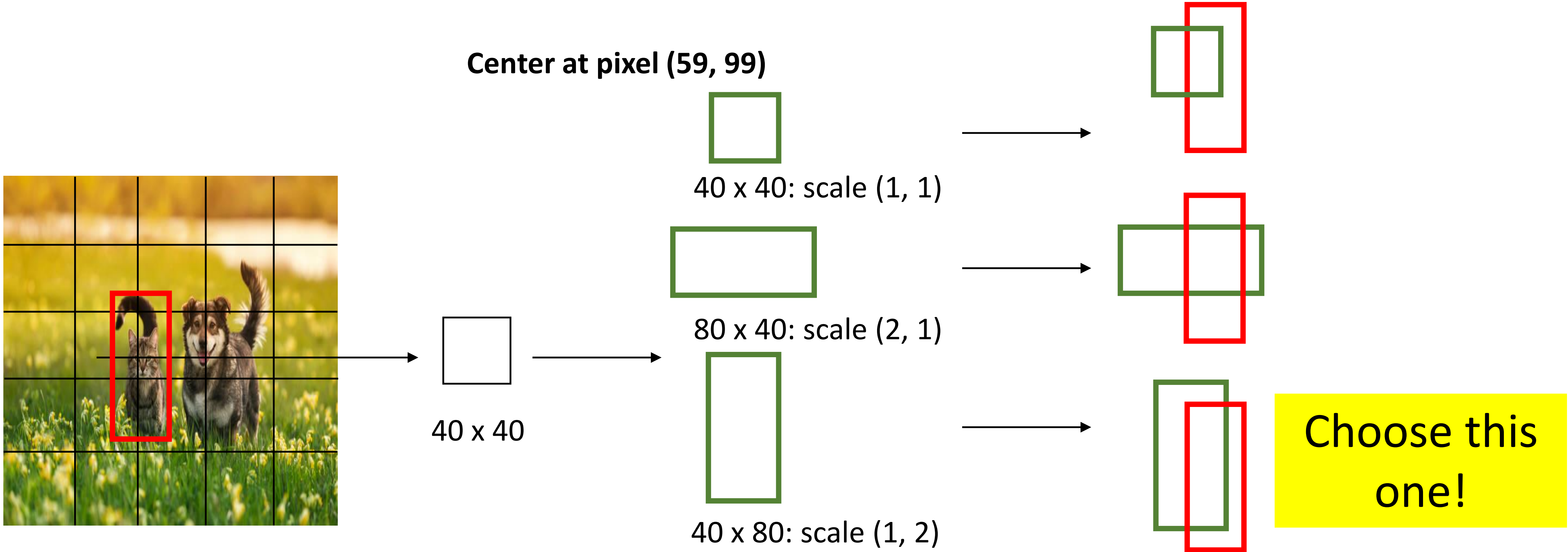
Q3: Advanced implementation: encoding

- Choose the anchor with the highest "Intersection over union (IoU)"



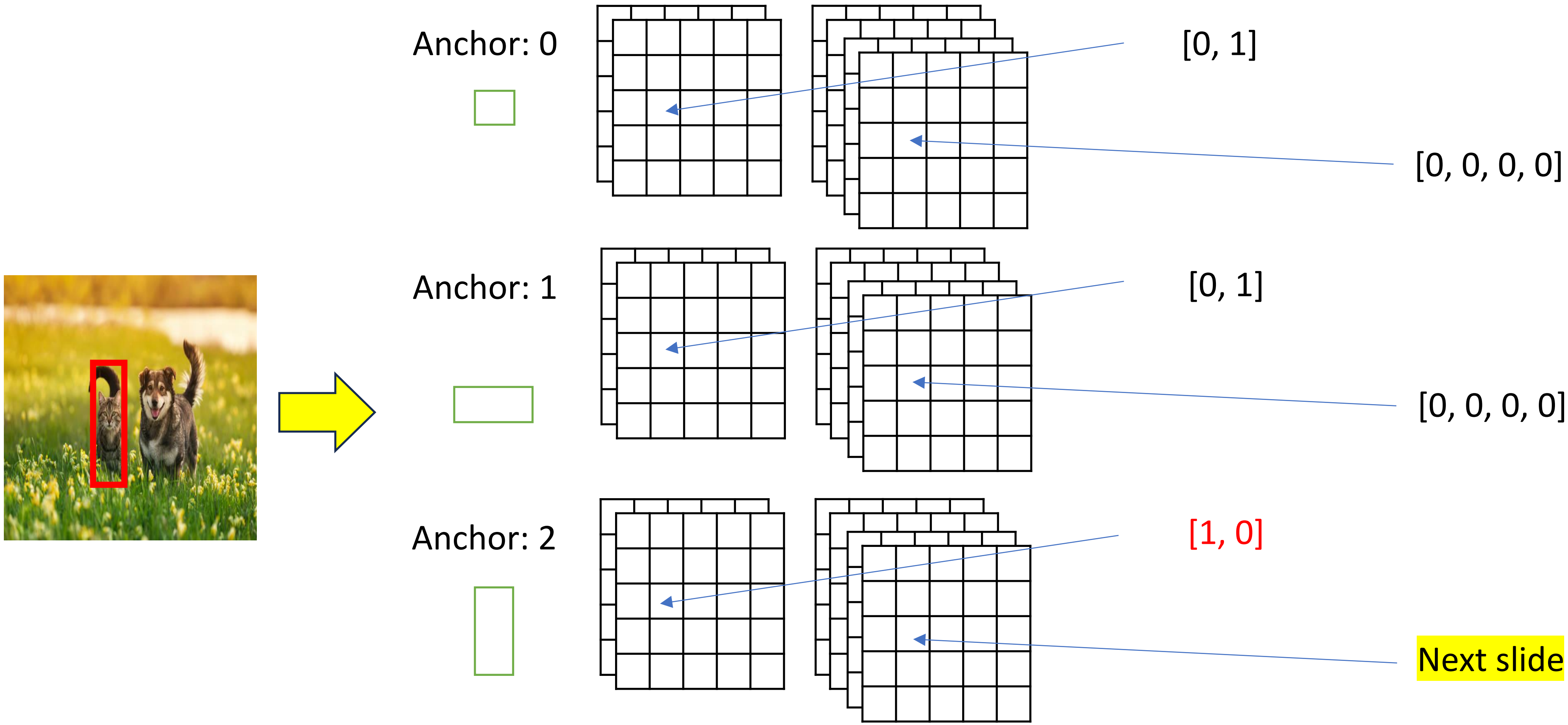
Q3: Advanced implementation: encoding

- Choose the anchor with the highest “Intersection over union (IoU)”



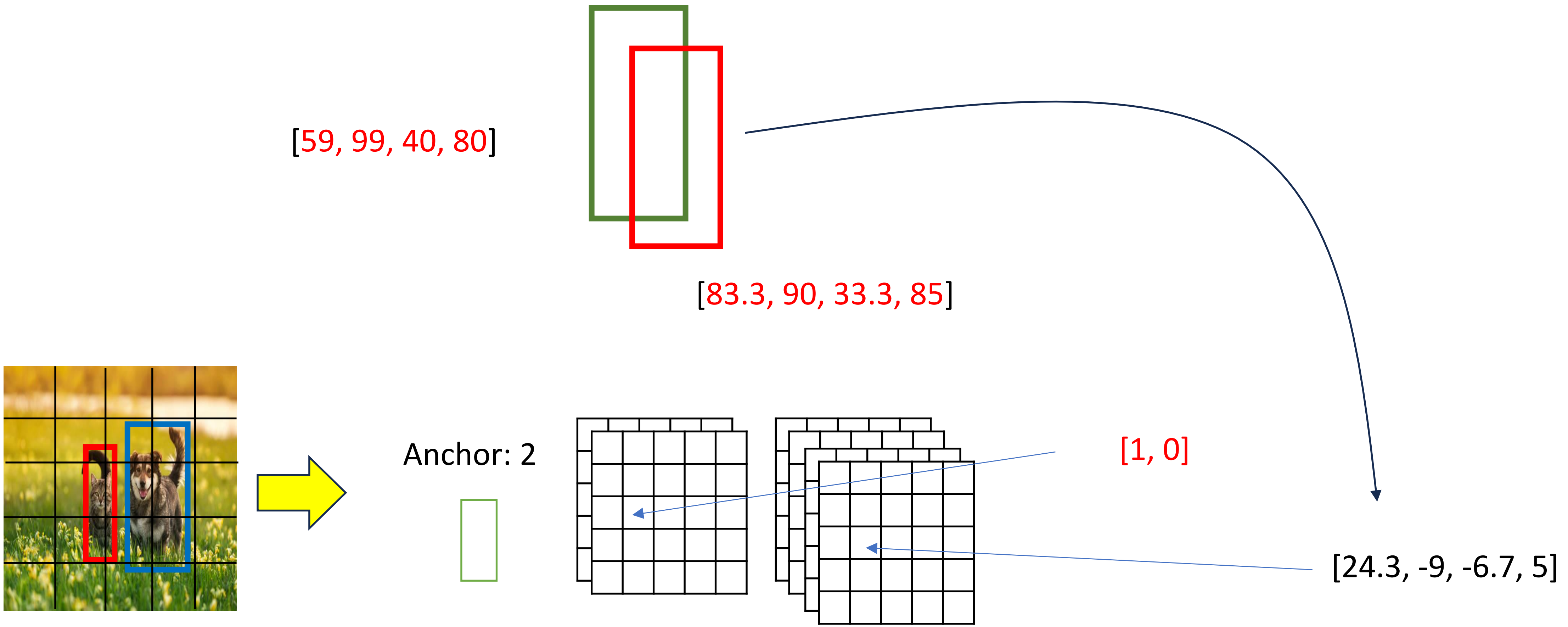
Q3: Advanced implementation: encoding

- Choose the anchor with the highest "Intersection over union (IoU)"



Q3: Advanced implementation: encoding

- Encode in the location tensor the "offsets" between the GT box and the chosen anchor



Q3: Advanced implementation: encoding

Pseudocode: when there are multiple GT objects

For each GT object:

For each patch:

If the current GT object and the current patch overlap

Choose the anchor that has the highest IoU with the current GT object

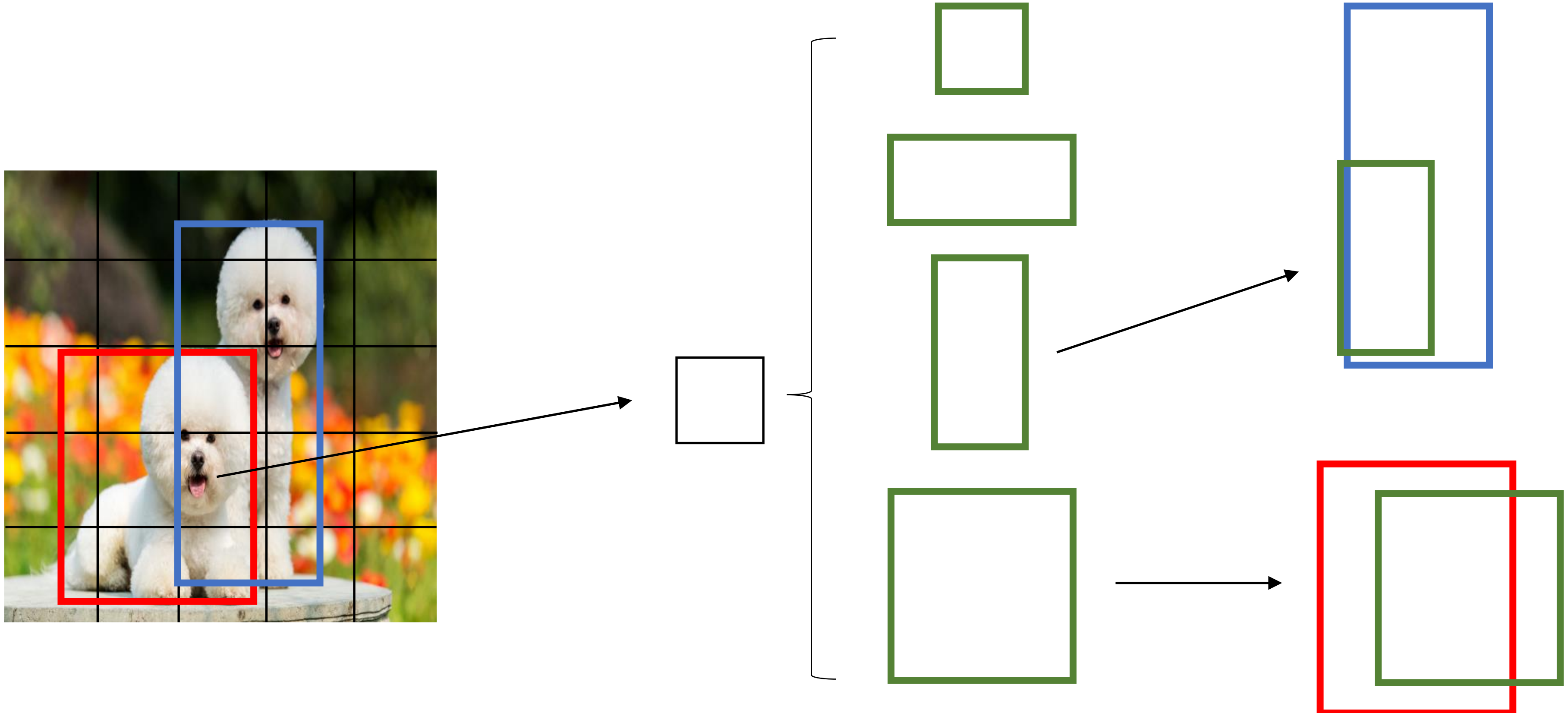
If the chosen anchor has not been used by other GT objects OR

If the current object has a larger IoU with the chosen anchor vs. other GT objects:

Record the existence and location offset based on the current GT object

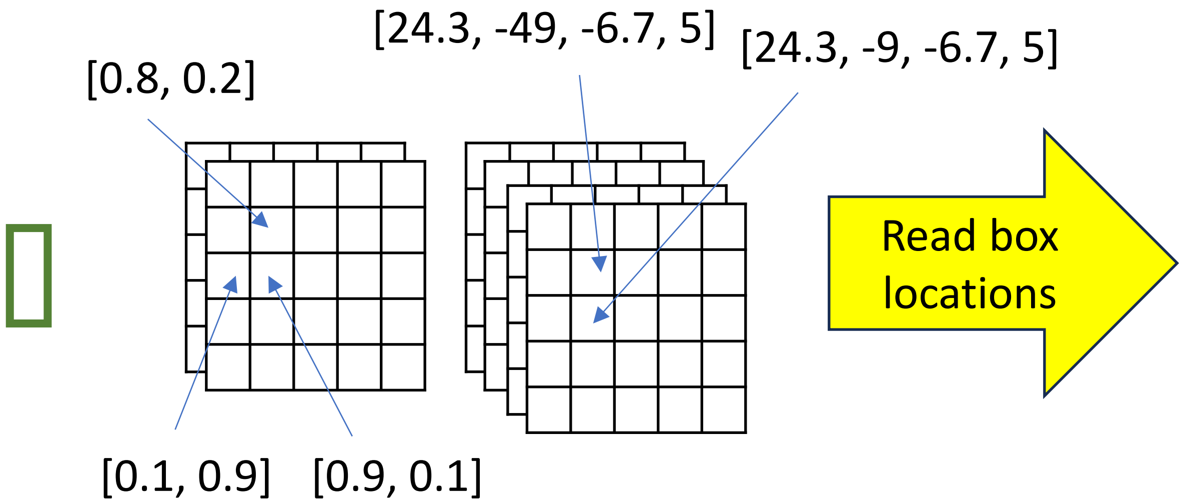
Q3: Advanced implementation: encoding

- That is, be careful about patches that overlap multiple GT objects
- At a patch, multiple anchors can be chosen (by multiple GT objects). However, one GT box can only choose one anchor at a patch.



Q4: Advanced implementation: decoding

- Suppose an RPN, after training, can output both the existence and location offset tensors close to the ideal tensors (with anchors), then we can read the object locations.
- For each anchor shape:
  - If a patch location has “Existence: Yes” values > a threshold (e.g., 0.2), we will read out its corresponding location (anchor location + offset) and output a box



[x-center, y-center, width, height, “Existence: Yes” value]

[24.3 + 59, -9 + 99, -6.7 + 40, 5 + 80, 0.9]

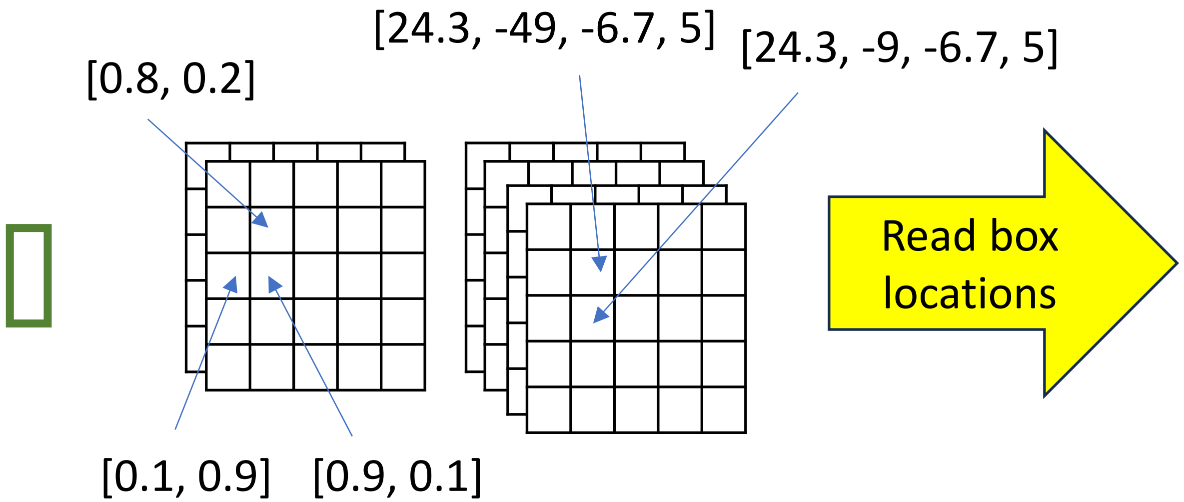
[24.3 + 59, -49 + 139, -6.7 + 40, 5 + 80, 0.8]

…

…

Q4: Advanced implementation: decoding

- That is, for the same GT box, we may output multiple boxes with potentially different **confidences** (i.e., “**Existence: Yes” values**)



[x-center, y-center, width, height, “Existence: Yes” value]

[24.3 + 59, -9 + 99, -6.7 + 40, 5 + 80, 0.9]

[24.3 + 59, -49 + 139, -6.7 + 40, 5 + 80, 0.8]

…

…

Q4: Advanced implementation: decoding

Pseudocode:

For each anchor:

    For each patch:

        If the “Existence: Yes” value > a threshold:

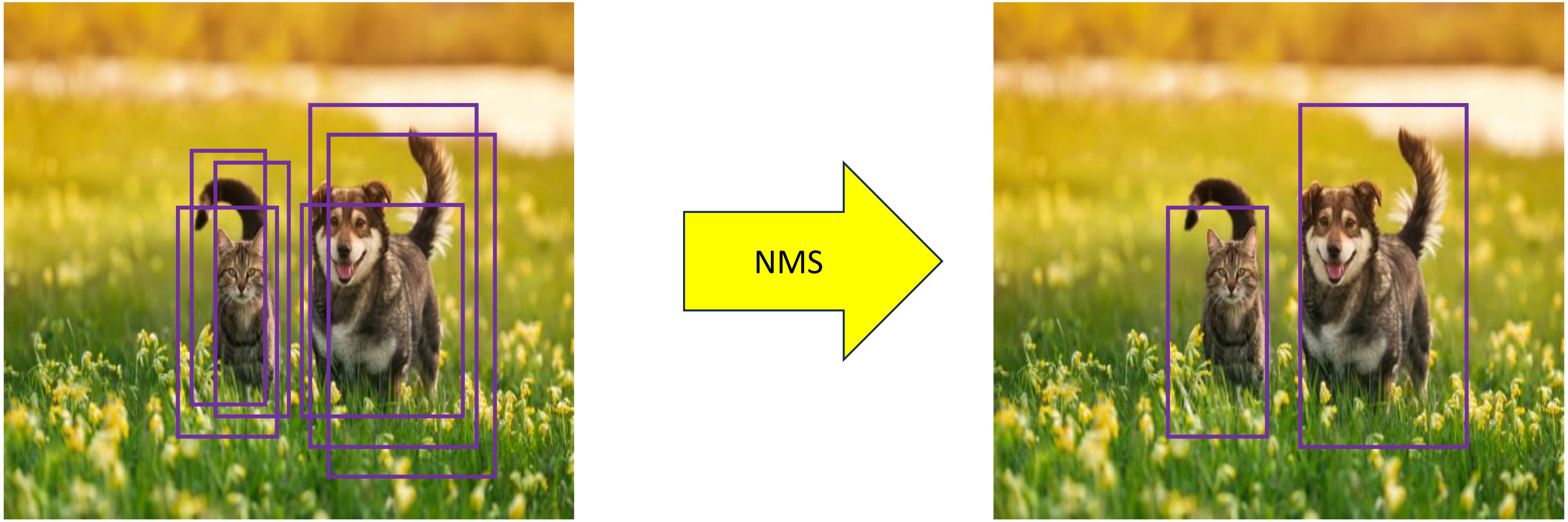
            Read out the box location (anchor location + offset) and confidence

Outline

- Recap of image classifiers
- Recap of object detectors
- Homework description (coding part)
  - Naïve implementation
  - Implementation with anchors and offsets
  - Non-Maximum Suppression (NMS)
- Homework description (written part)

Q5: Non-Maximum Suppression (NMS)

- The decoding rule may output multiple boxes for a single GT object
- We need to implement NMS to subsample them
- Please note that, at this stage, you do not know where GT objects are



Q5: Non-Maximum Suppression (NMS): Pseudocode

- Sort all the outputted boxes based on the “confidences,” from high to low
- Define an empty set “S” to record the subsampled boxes

- For each outputted box (from confidence high to low):
- If the current box has IoU with every box in “S” smaller than a threshold:
- Add the current box into the set “S”

Reference: https://builtin.com/machine-learning/non-maximum-suppression

Outline

- Recap of image classifiers
- Recap of object detectors
- Homework description (coding part)
- Homework description (written part)

Written Part: Q6

- Q6.1: Please briefly describe your strategy when a patch overlaps with multiple GT boxes (no more than 30 words) in Programming question Q1. That is, while in the programming question Q1, we ask you to choose a random GT box, do you have any other idea?

Written Part: Q6

- Q6.2: Given a 224-by-224 RGB image, what is the feature map size (e.g., 5 x 5 x 256) before the final fully connected layer (or final MLP) of the following classifiers? You may search for your answers online.

  - ResNet-50: https://arxiv.org/pdf/1512.03385
  - VGG-19: https://arxiv.org/pdf/1409.1556
  - ViT-B/32: https://arxiv.org/pdf/2010.11929

- Specifically, for ViT-style models, your answer should be # horizontal patches x # vertical patches x # channels or token dimensions