DEEP LEARNING FOR VISUAL RECOGNITION

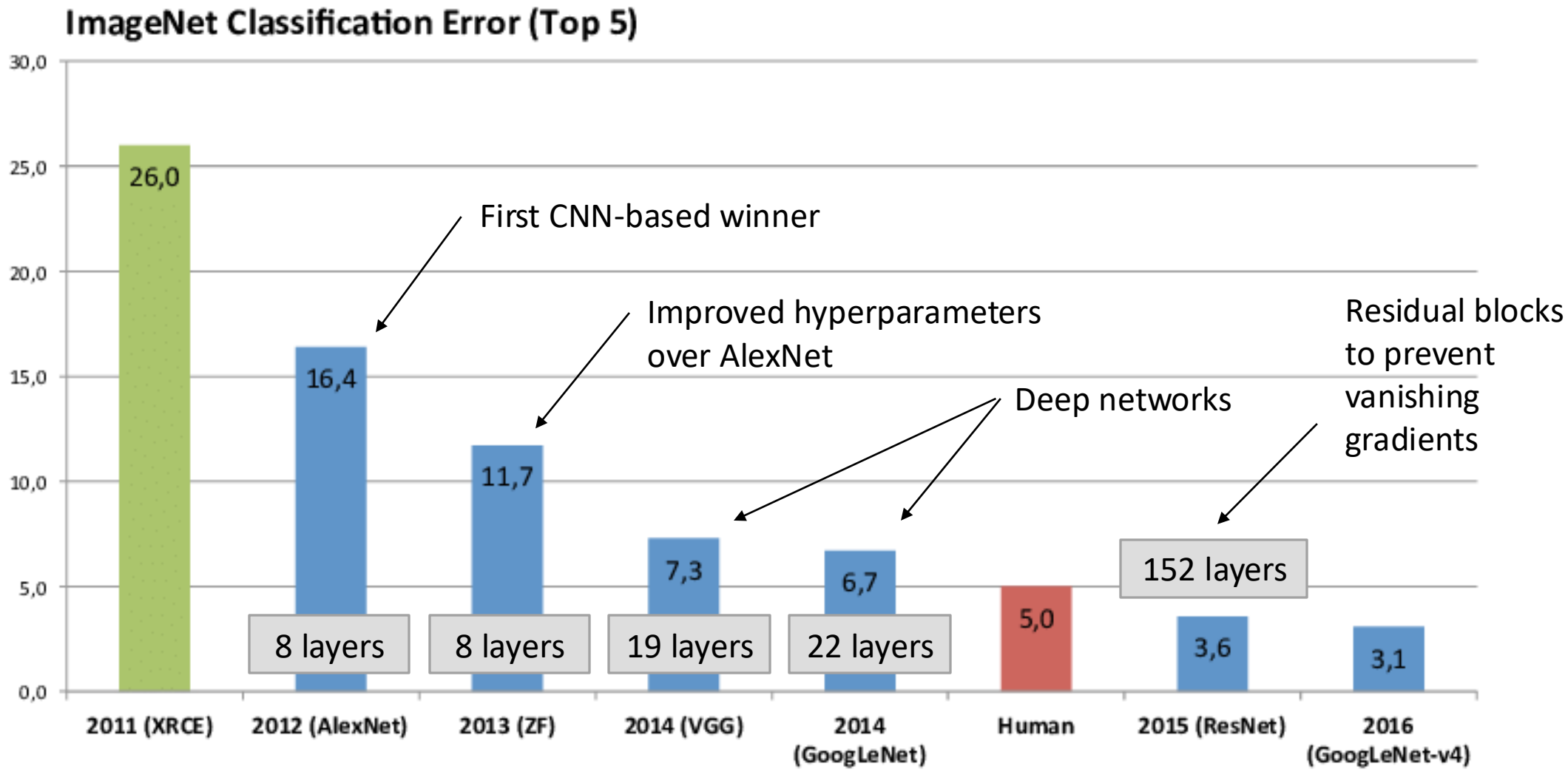
Lecture 8 – Object Detection and Segmentation

Henrik Pedersen, PhD
Part-time lecturer
Department of Computer Science
Aarhus University
hpe@cs.au.dk

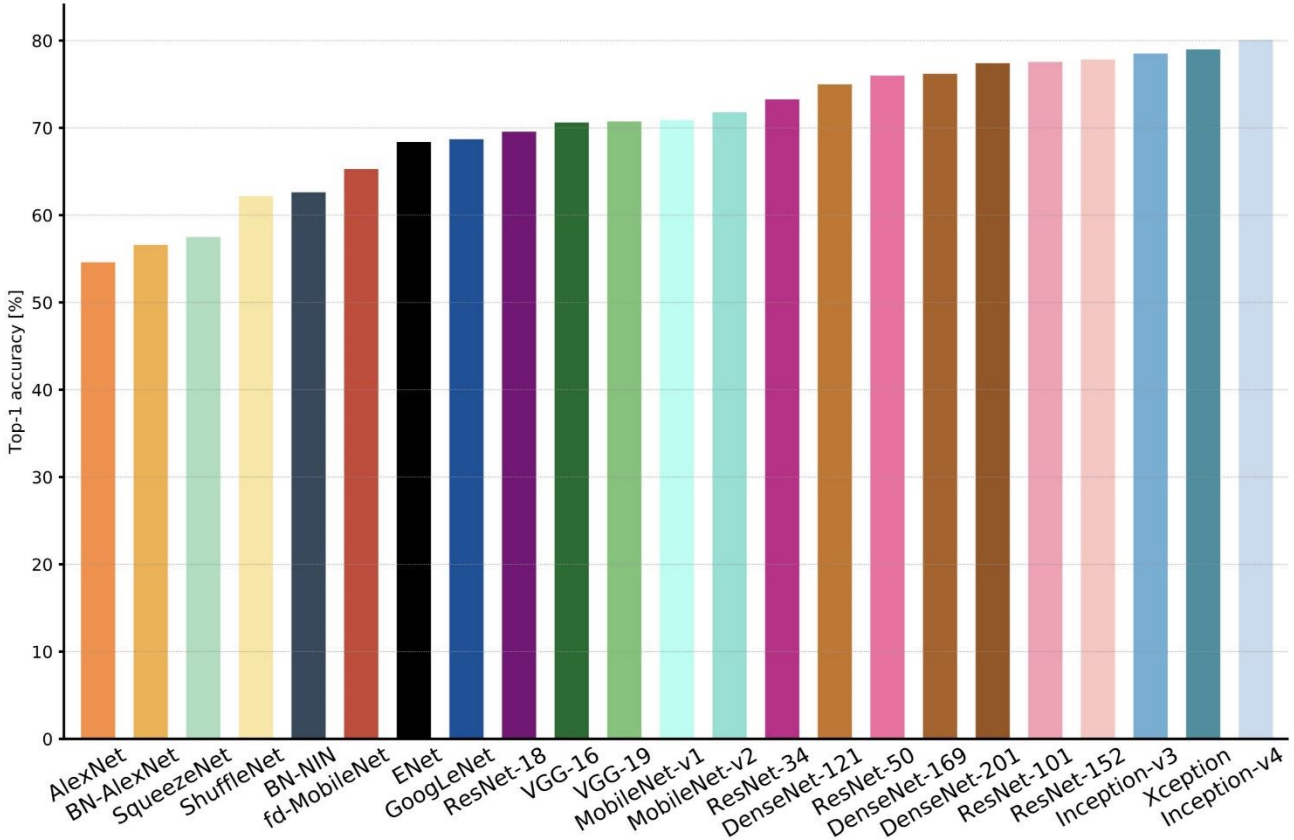
Today's agenda

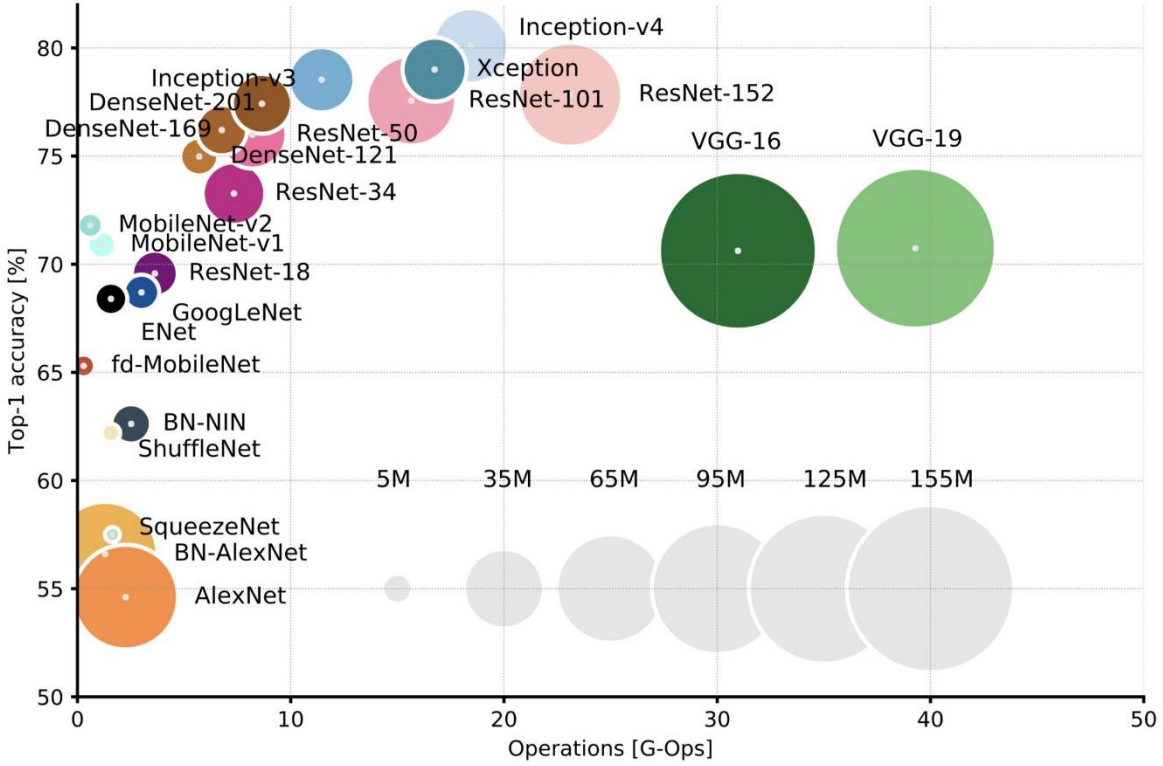
- You will learn about CNN architectures for object detection and object segmentation.
- Topics
  - Localization vs. detection vs. segmentation
  - Localization as regression
  - Sliding window
  - Efficient sliding window using fully convolutional networks
  - Region proposals
  - R-CNN (Regions with CNN features) and its variants
  - One-stage object detection models
  - Image-to-image translation networks
  - Fully Convolutional Networks (FCNs) for semantic segmentation
  - FCN improvements: Skip connections and upsampling techniques
  - Databases (PASCAL VOC, MS COCO)
  - Evaluation metrics

Last time – ConvNet architectures



Last time – Comparing complexity





Top-1 accuracy versus number of operations required for a single forward pass. The size of the blobs is proportional to the number of network parameters.

https://medium.com/@culurciello/analysis-of-deep-neural-networks-dcf398e71aae

Last time – ConvNet architectures

- Many popular architectures available in **model zoos**
  - Keras: https://keras.io/applications/
  - PyTorch: https://docs.pytorch.org/vision/main/models.html
  - TensorFlow: https://github.com/tensorflow/models
- ResNet or MobileNet are currently good defaults to use.
- For more reviews of CNN architecture: https://sh-tsang.medium.com/

Today – Basic computer vision tasks

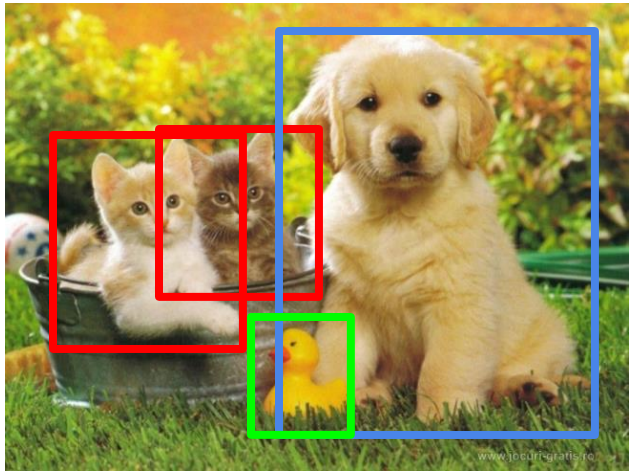
Classification

CAT

Classification
+ Localization

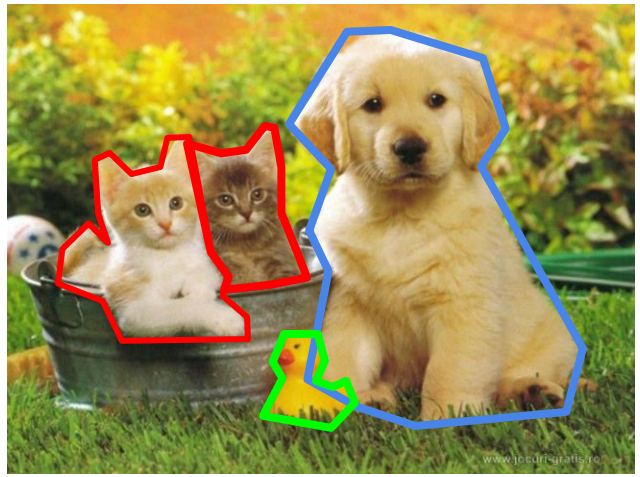
CAT

Object Detection



CAT, DOG, DUCK

Image
Segmentation



CAT, DOG, DUCK

Single object

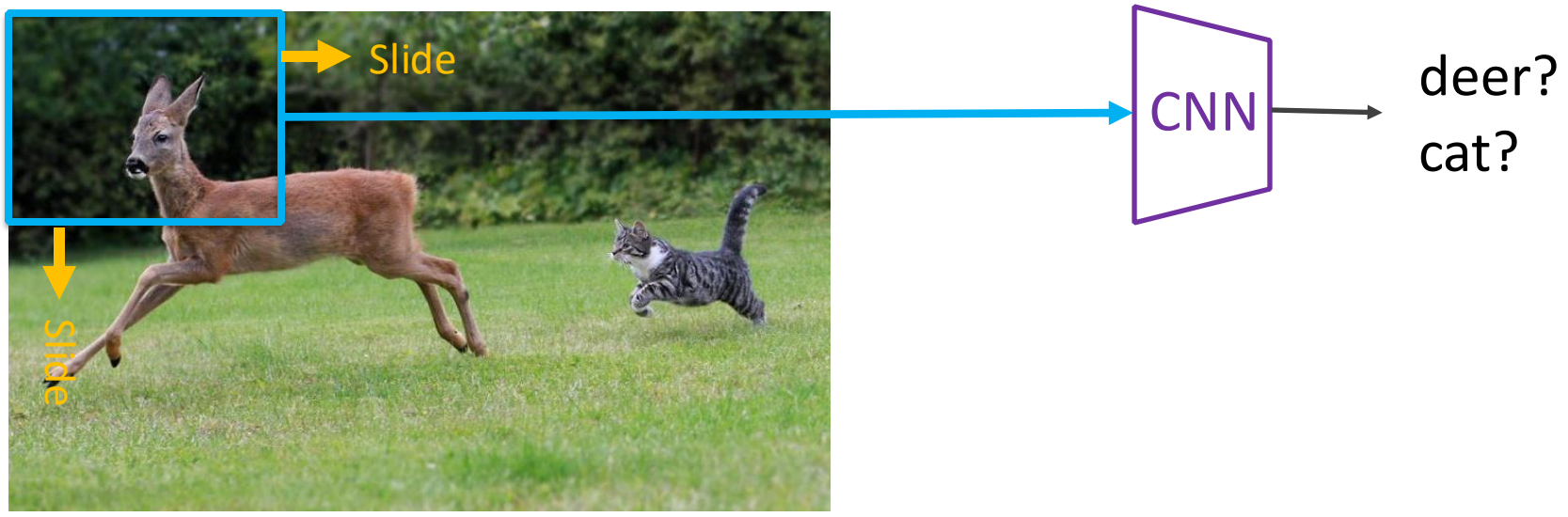
Multiple objects

Primer

EFFICIENT SLIDING WINDOW USING 1X1 CONVOLUTIONS

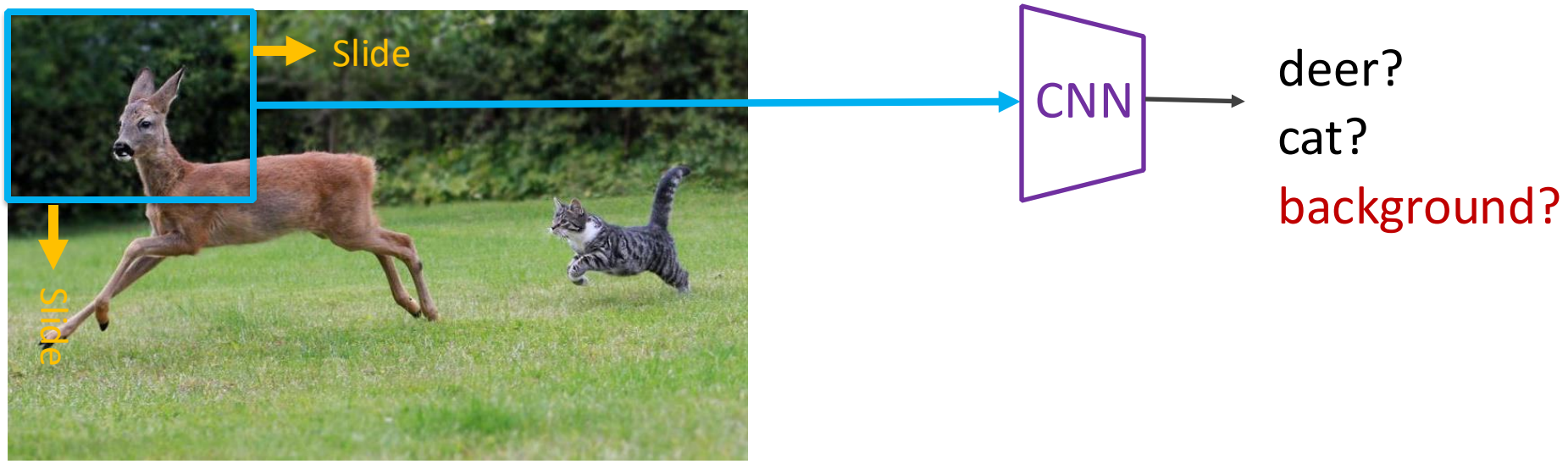
Object detection using sliding window

- **Problem:** How do we localize objects in images?
- **Simple idea:** Run CNN classifier at multiple locations using a sliding window.



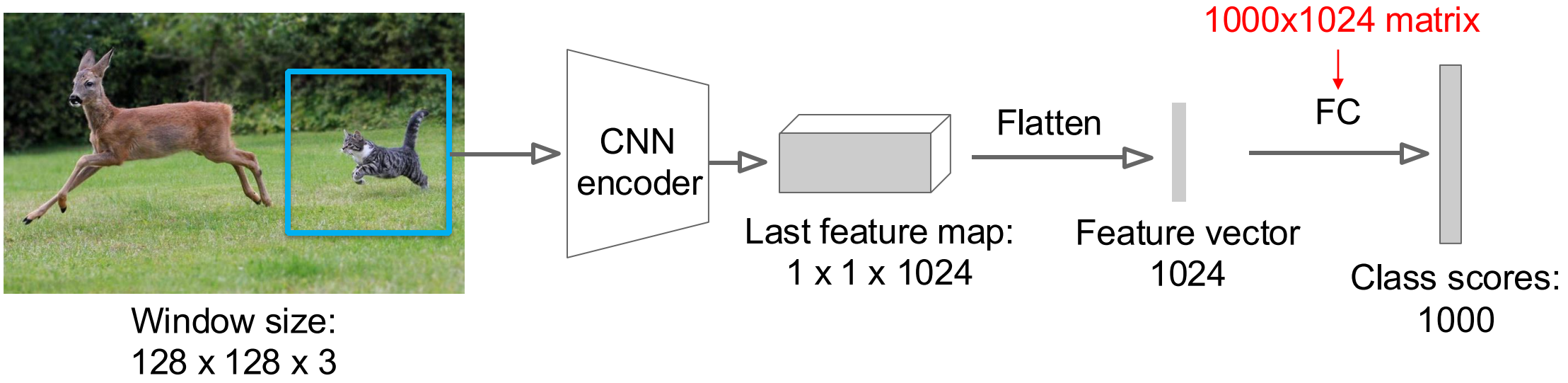
Issues with sliding window

1. It’s going to be very slow (one full forward pass per window location).
2. Objects can have different scales -> we would need to repeat the process for different window sizes.
3. Very unlikely that we will get tight/exact bounding boxes (only happens by chance).
4. Most windows will be false positives (i.e., no object) – usually a good idea to add a background class.



Standard CNN classifier with FC layer(s)

Example: Let's assume we designed our CNN encoder such that it takes input images (windows) of size 128x128x3 and produces a feature map of size 1x1x1024, and let's assume there are 1000 classes.



Side-note: Will not work on full image

If we input the full input image, the last feature will be larger than the expected 1x1x1024, and the math will break down at the FC layer.

Full image size:
256 x 384 x 3

CNN encoder

Last feature map:
2 x 3 x 1024

Flatten

Feature vector
2x3x1024

1000x1024 matrix

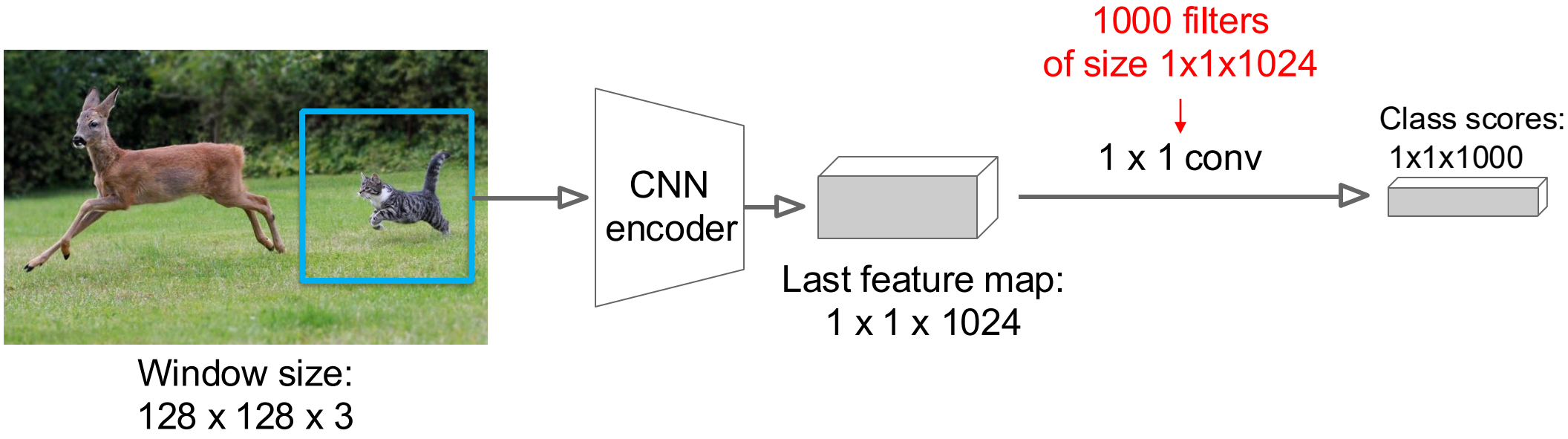
FC

Class scores:
1000

The math breaks because we cannot multiply a 1000x1024 matrix with a 2x3x1024-length vector.

Replacing FC layers with 1x1 convolutions

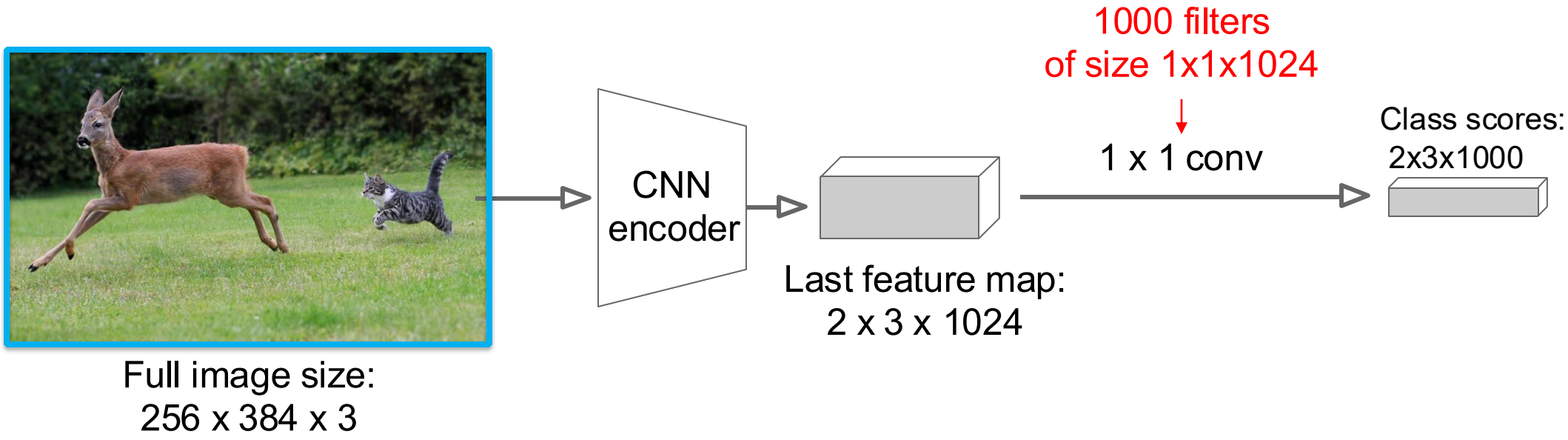
Observation: If we replace the fully connected (FC) layer(s) with 1x1 convolutions, the class scores are going to be the same, but there are no vectors anymore. Everything is just volumes!!!



Side-note: CNNs where the fully connected layers have been replaced by 1x1 convolutions are historically referred to as **Fully Convolutional Networks (FCNs)**.

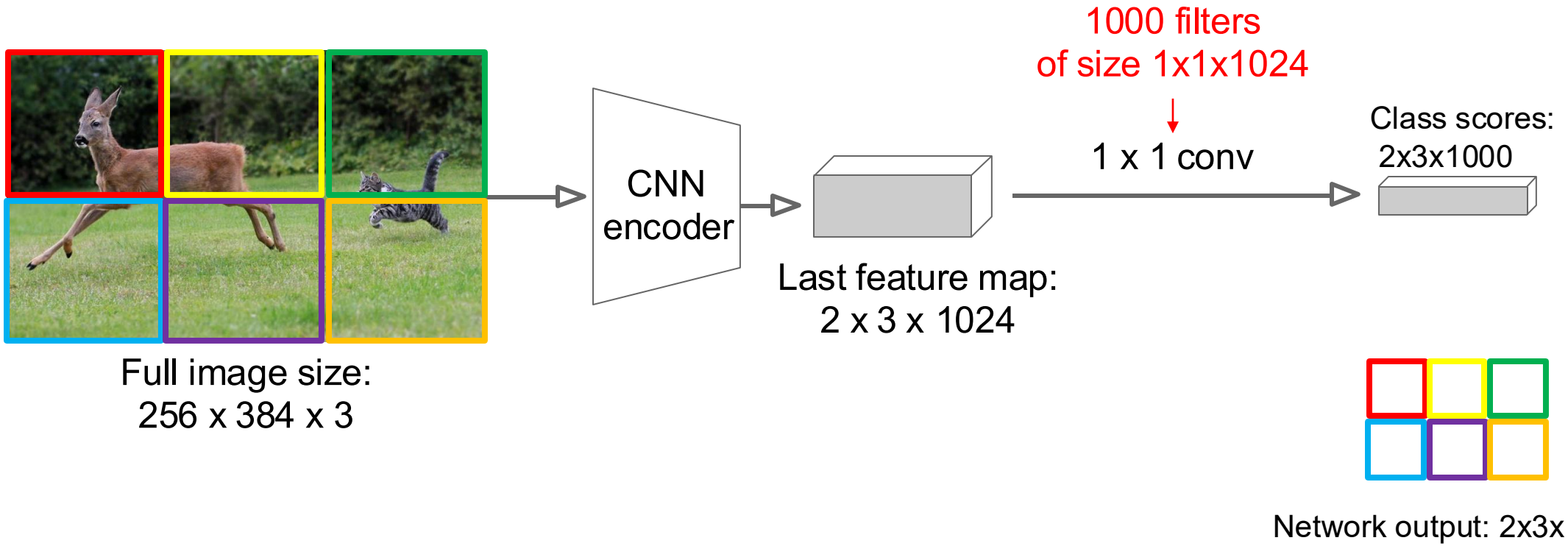
This will work on the full image!!!

We're just going to get a volume as output instead of a vector (of shape 2x3x1000 in the example below).



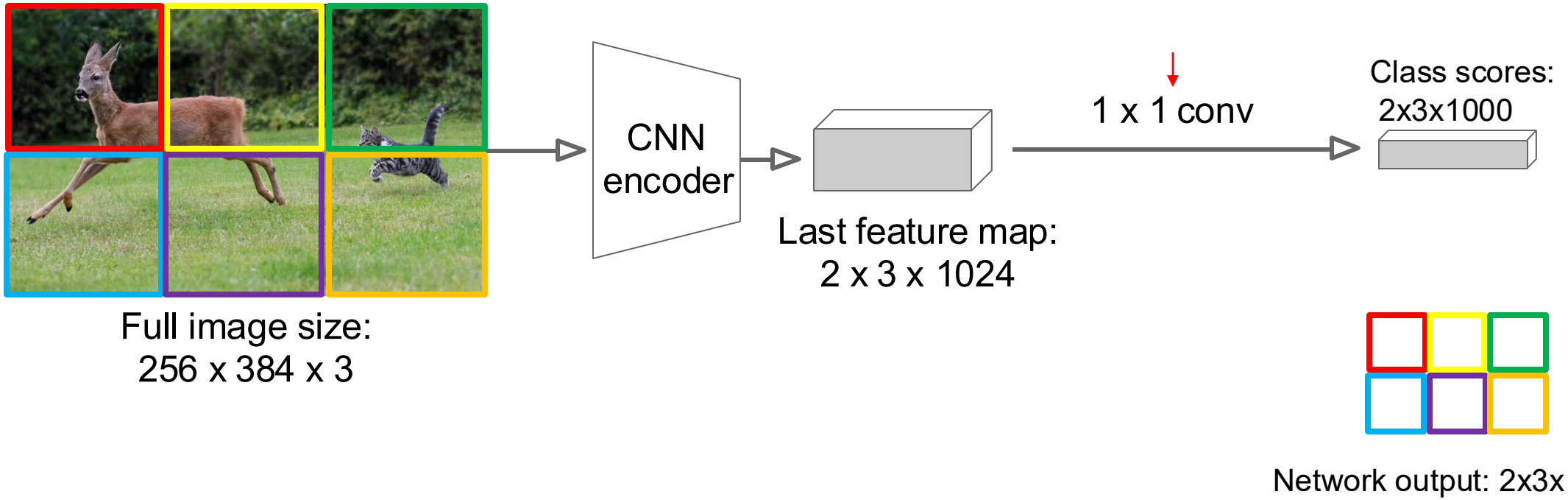
How to interpret the output?

We can map each pixel in the output volume back to its corresponding receptive field in the input image.
So, we are effectively just **sliding a classifier over the input image** – but this time using only one forward pass!!!



Efficient sliding window using 1x1 conv.

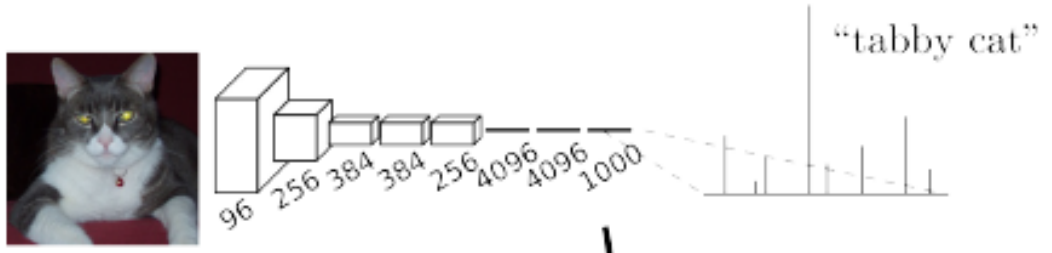
We can think of the 1x1 convolutions as doing a form of sliding window, but in a single forward pass (fast!).
However, we are not quite done yet – the deer and the cat are not fully contained in any of the windows.
We need an extra ingredient for this to work for object detection. That extra ingredient is **localization**.



Side-note: Link to image segmentation

- Replacing the FC layers with 1x1 convolutions is the first step towards implementing image segmentation (i.e., pixel-level classification).

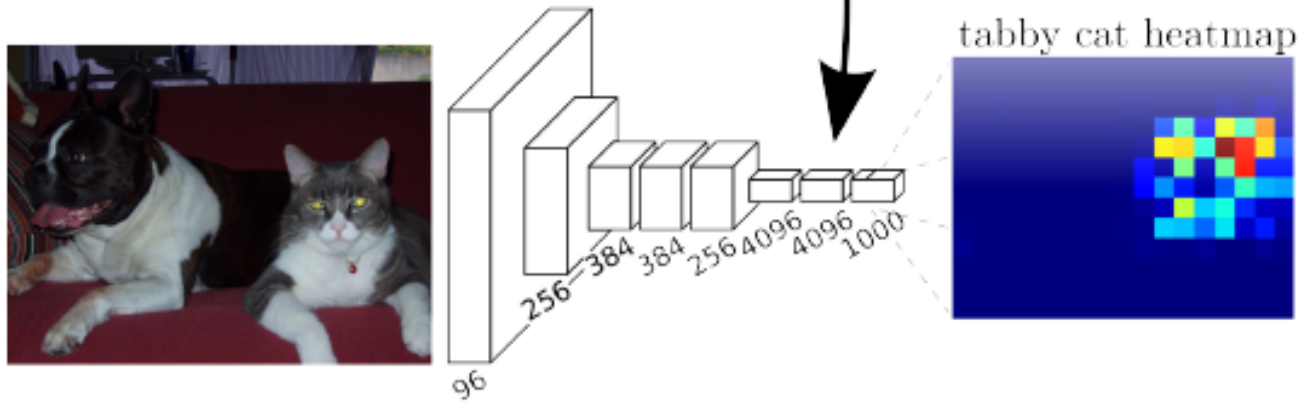
Standard CNN classifier:



Output is a vector of class probabilities
Shape: 1 x C

convolutionalization

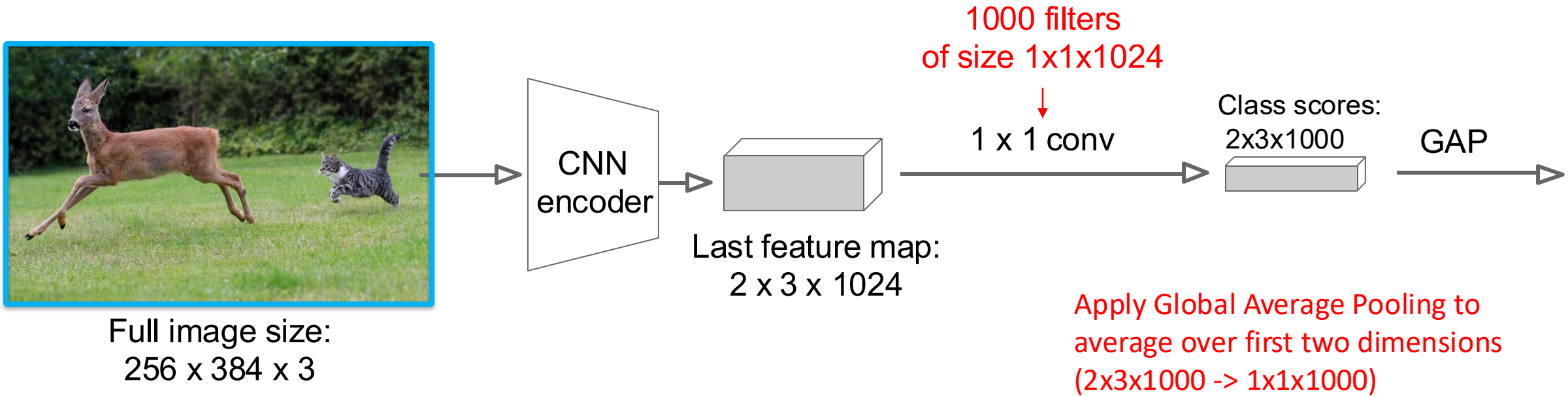
FCN:



Output is a volume of heatmaps of class probabilities
Shape: H x W x C

Side-note 2: Classifier with 1x1 convs

- If we want to have an image classifier, but with 1x1 convolutions instead of FC layers, we can just add a global average pooling layer to obtain an output vector of class probabilities.



Object localization

Computer vision tasks

Classification

Classification
+ Localization

Object Detection

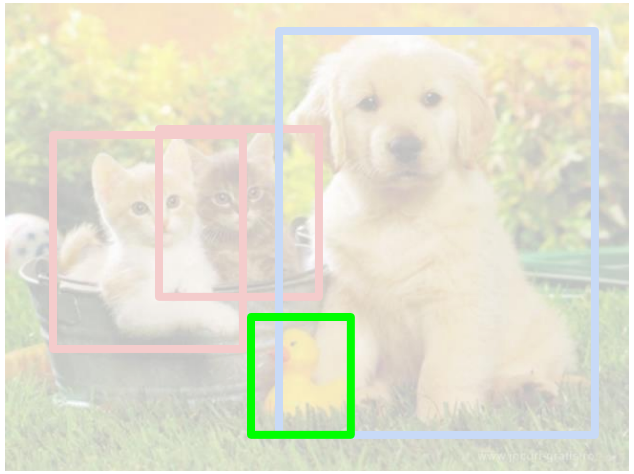
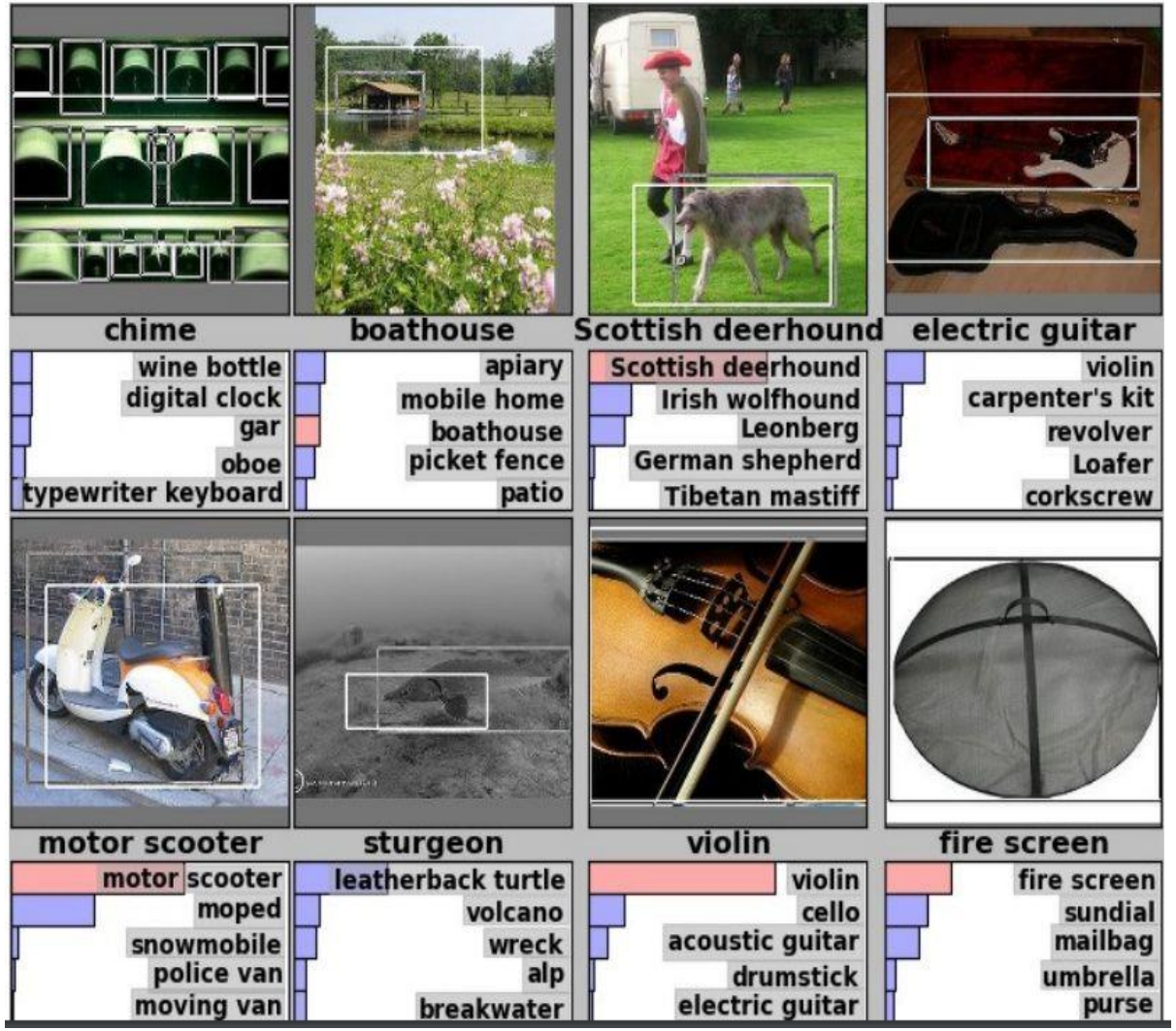


Image
Segmentation

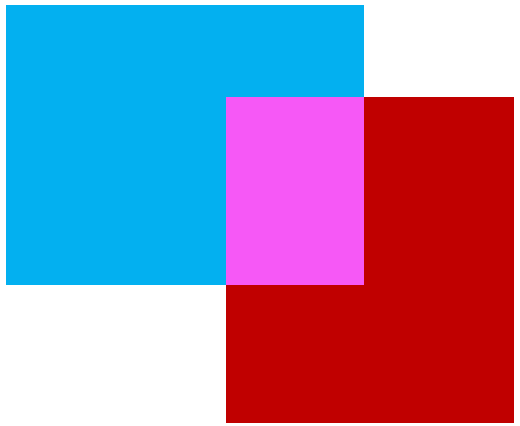
Classification + Localization: ImageNet

- 1000 classes (same as classification).
- Each image has **1 class and at least one bounding box**.
- ~800 training images per class.
- In the competition, **your algorithm should produce 5 (class, box) predictions**.
- Correct if at least one prediction has the correct class AND a bounding box with **at least 0.5 intersection over union (IoU)**.
- Also see slides from week 4: “HowToWriteAGoodReport”.



Object detection/localization evaluation

- We use a metric called “mean average precision” (mAP), covered in week 4.
- Compute average precision (AP) separately for each class, then average over classes.
- Combine all detections from all test images to draw a precision / recall curve for each class; AP is area under the curve.
- A detection is a true positive if it has IoU with the ground-truth box greater than some threshold (usually 0.5) (mAP@0.5).
- TL;DR mAP is a number from 0 to 100; high is good.



$$IoU(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Precision = #correct detections / total detections

Recall = #ground truth with matched detections / total ground truth

https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173

Classification + Localization

Classification: C classes
Input: Image
Output: Class label
Evaluation metric: Accuracy

CAT

Localization: Regression
Input: Image
Output: Box in the image (x, y, w, h)
Evaluation metric: Intersection over Union

(x, y, w, h)

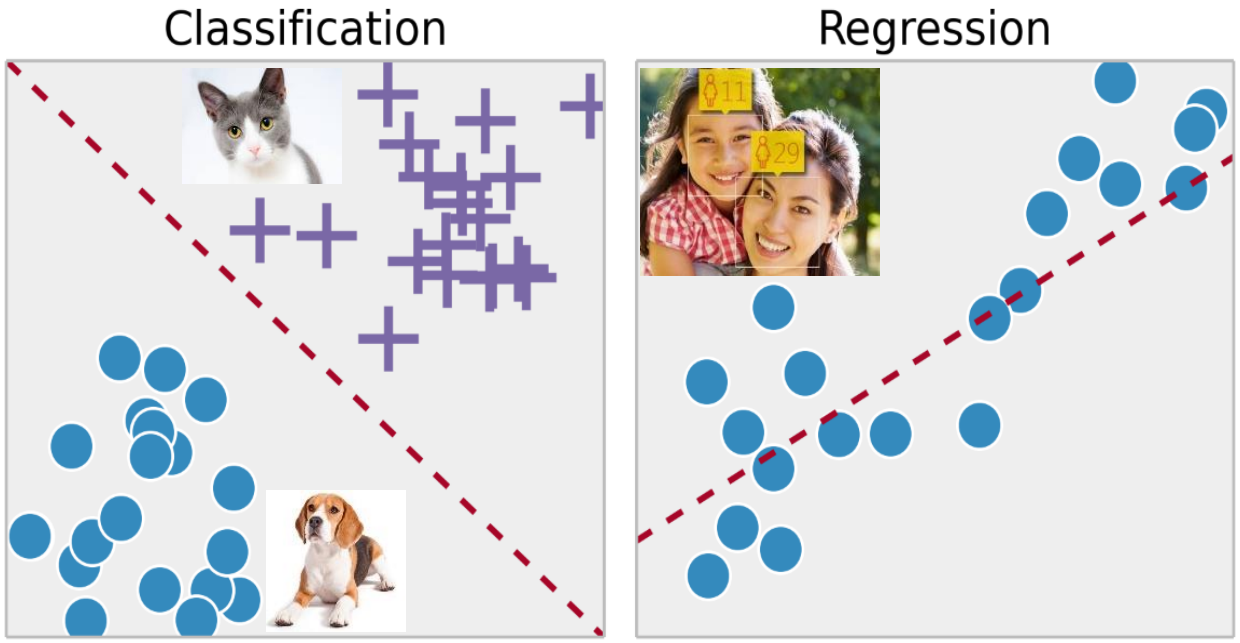
Classification + Localization: Do both

Recall regression vs. classification

Classification
Use features to distinguish between two or more classes.

Output:
Discrete labels
(”Dog” or ”Cat”)

Loss function:
Cross entropy



Regression
Use features to predict some functional relationship.

Output:
Real numbers
(”Age” of person in image)

Loss function:
L2, L1, MSE, …

Localization as regression

Reminder:

$$L2LossFunction = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2$$

Input: image

Only one object
(simpler than detection)

Neural Net

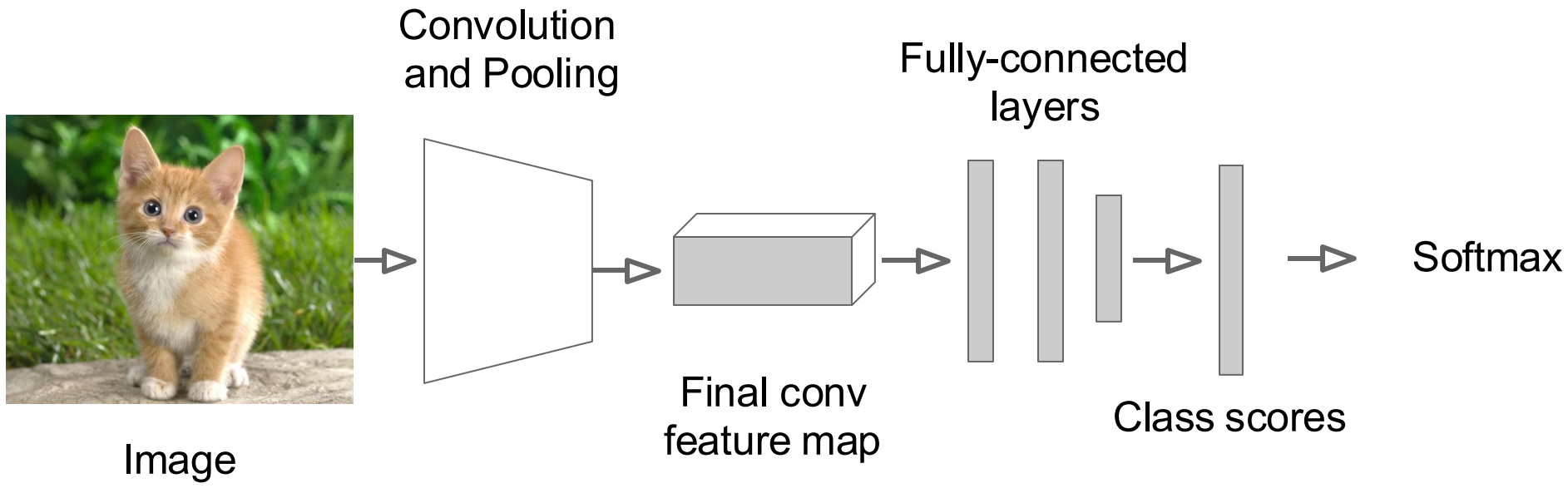
Predicted:
Box coordinates
(4 numbers)

True:
Box coordinates
(4 numbers)

Loss:
L2 distance

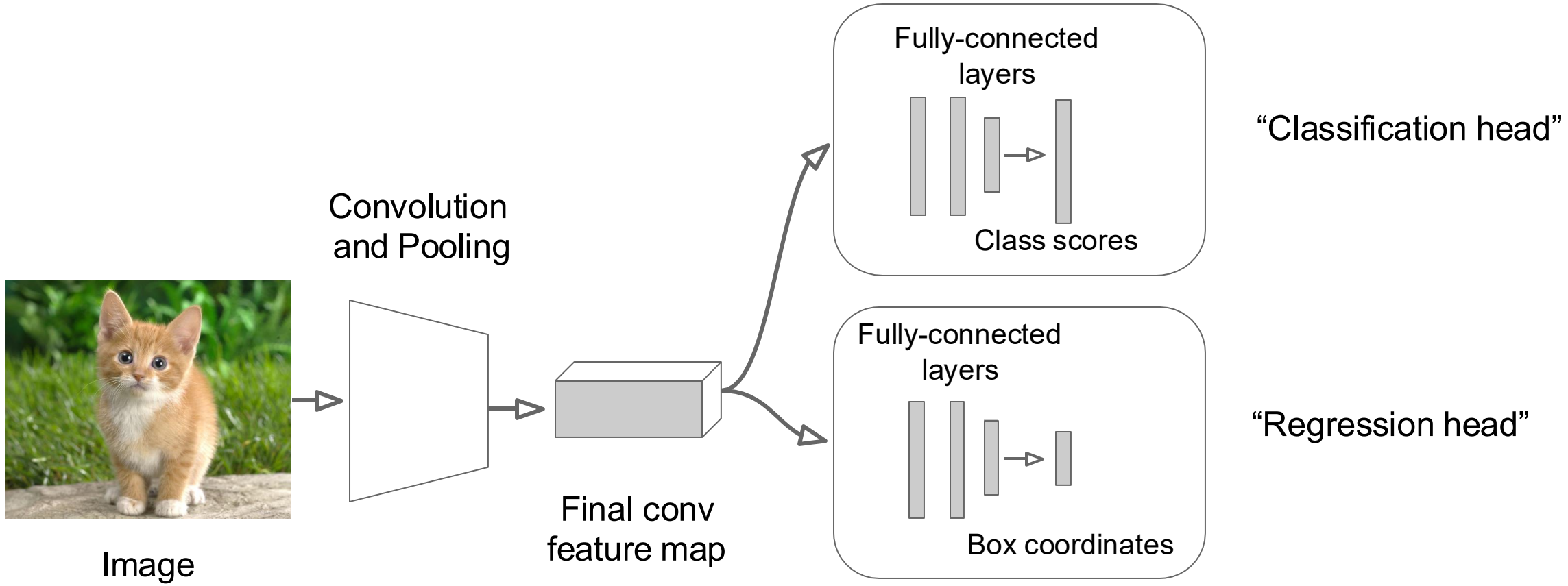
Classification + Localization

Step 1: Download a pre-trained CNN classifier (AlexNet, VGG, GoogLeNet)



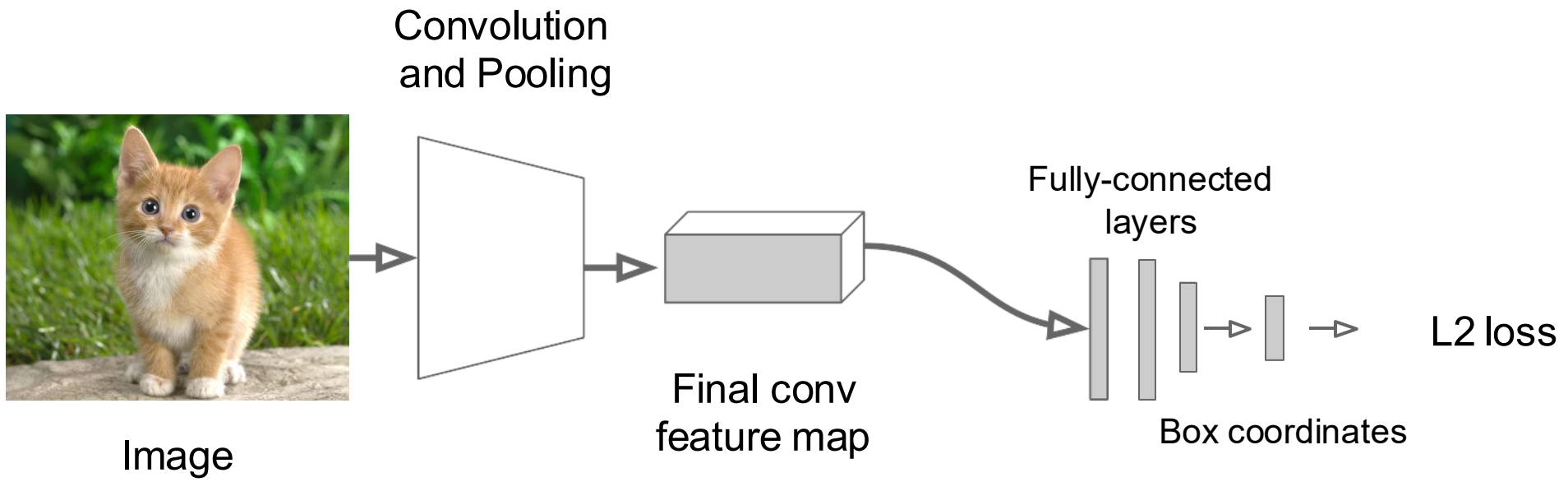
Classification + Localization

Step 2: Attach new fully-connected "regression head" to the network



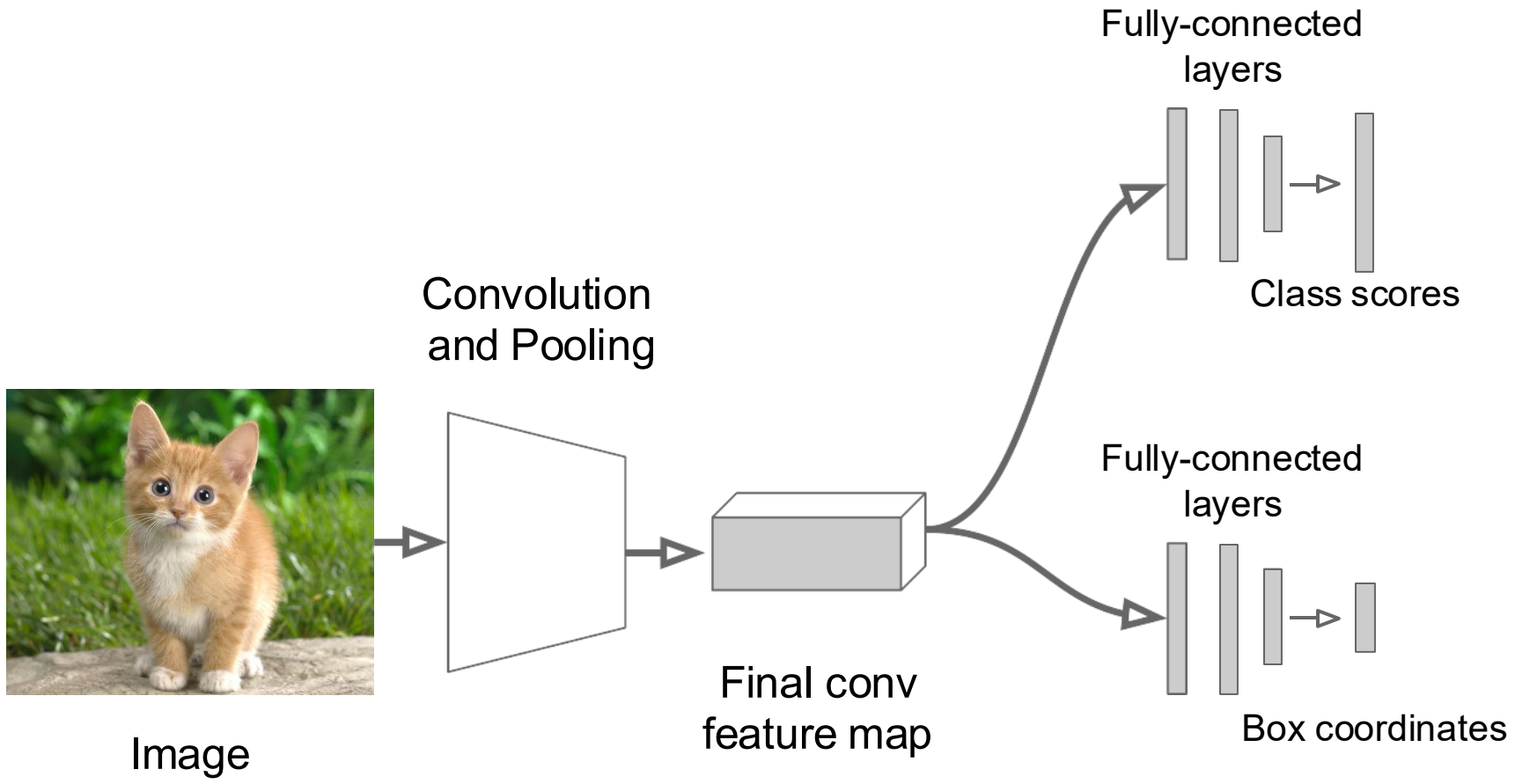
Classification + Localization

Step 3: Train the regression head (use L2 loss because it is regression)



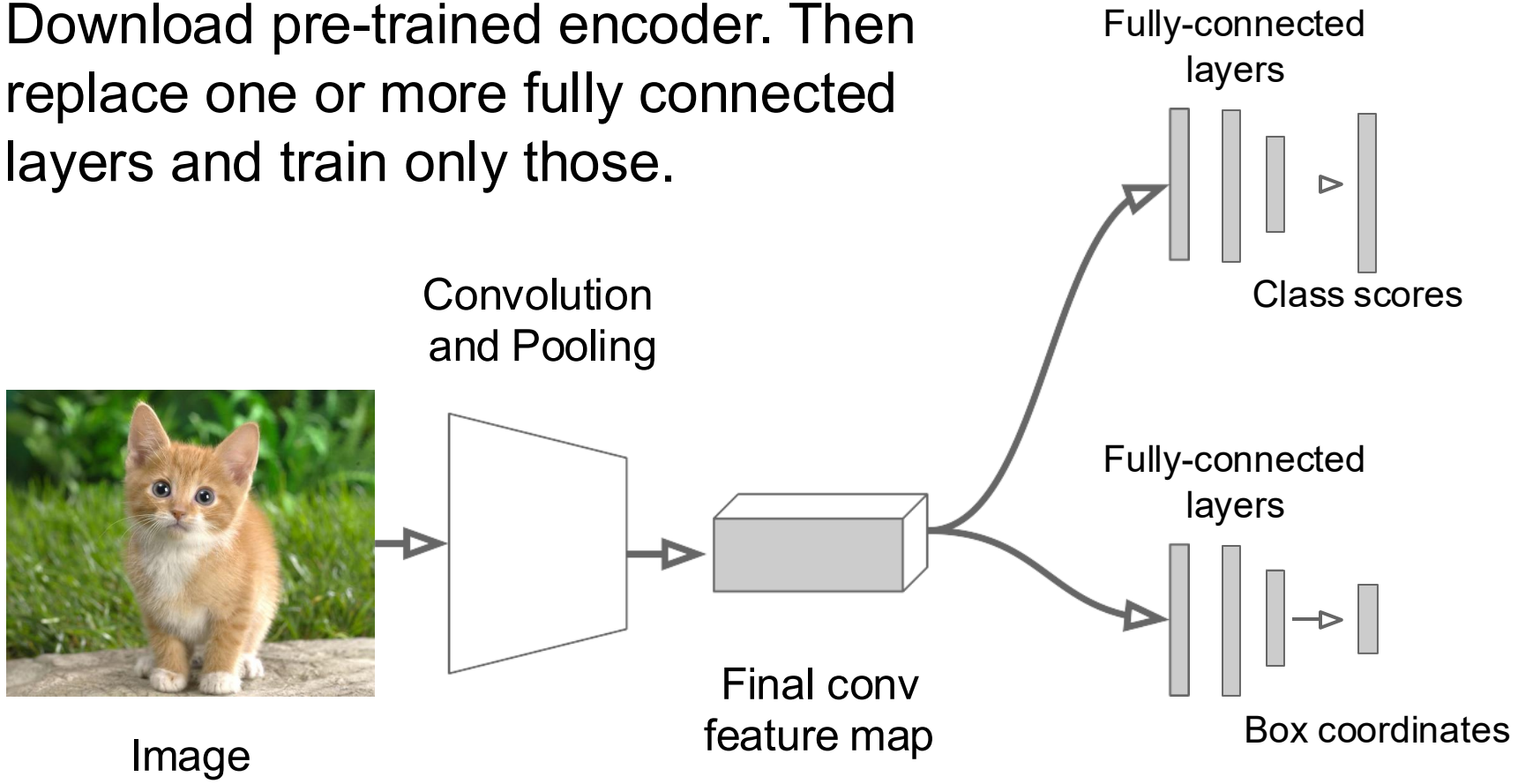
Classification + Localization

Step 4: At test time use both heads



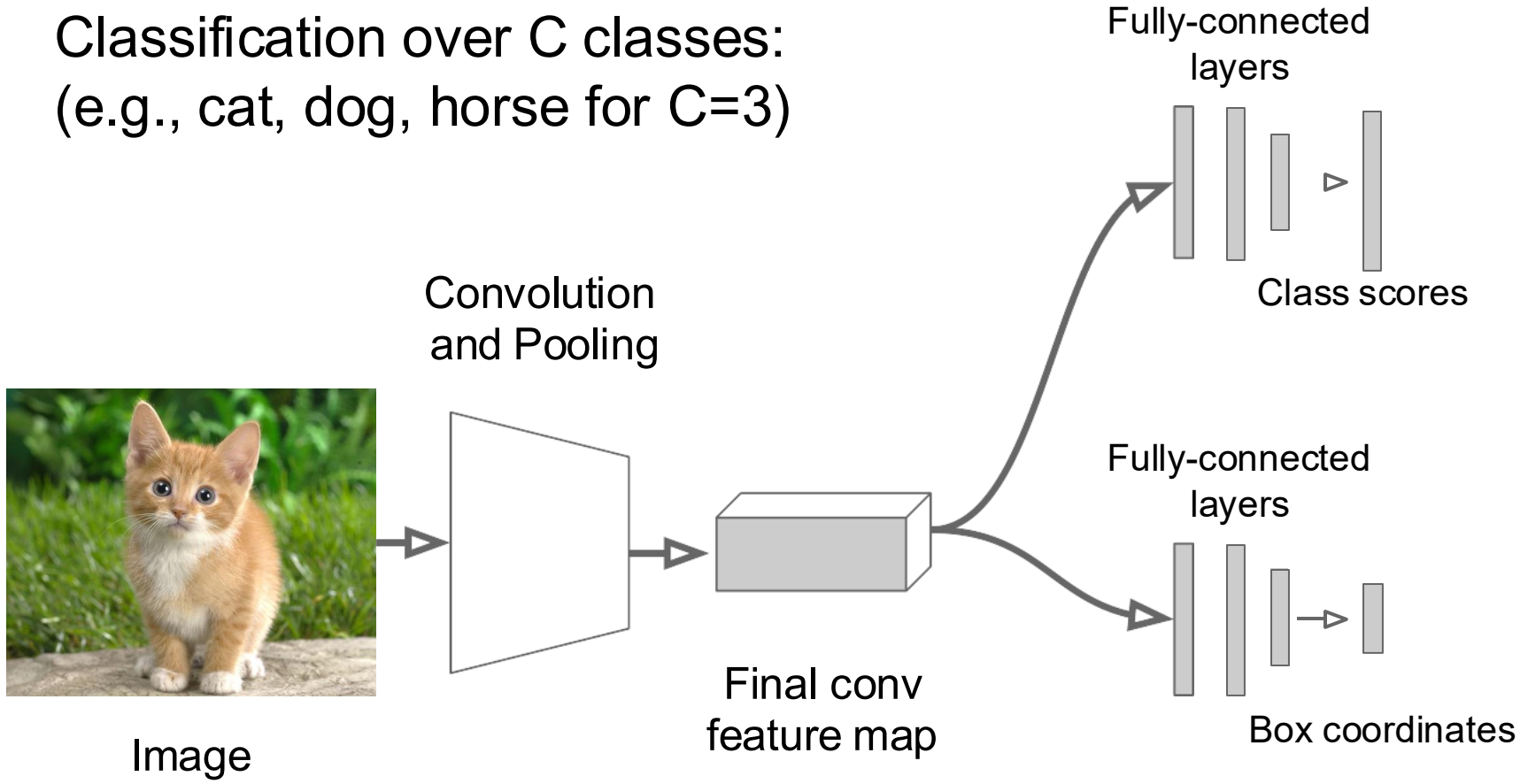
Aside: Notice this is transfer learning

Download pre-trained encoder. Then replace one or more fully connected layers and train only those.



Per-class vs class agnostic regression

Classification over C classes:
(e.g., cat, dog, horse for C=3)



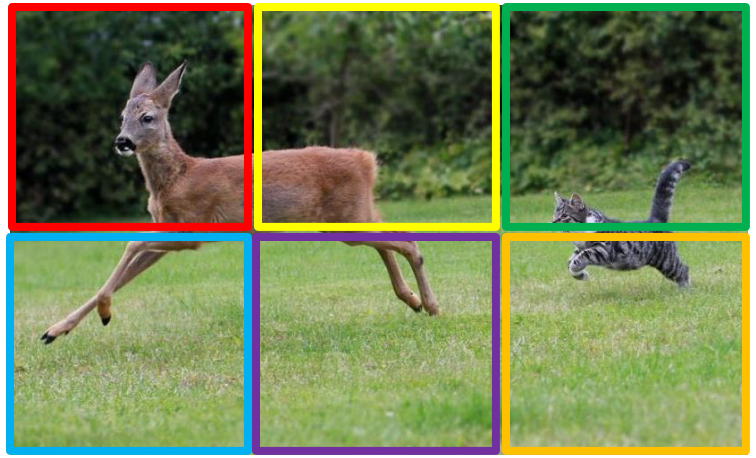
Classification head:
C numbers
(one per class)

Class agnostic:
4 numbers
(one box)

Class specific:
C x 4 numbers
(one box per class)

Link to sliding window

- In sliding window, the window is typically square, and its size is fixed, so **we can only detect square objects of a fixed size**.
- Also, it is very unlikely that the objects of interest will be fully contained in any of the windows.
- Localization (i.e., predicting the bounding box coordinates) fixes both of these two issues.

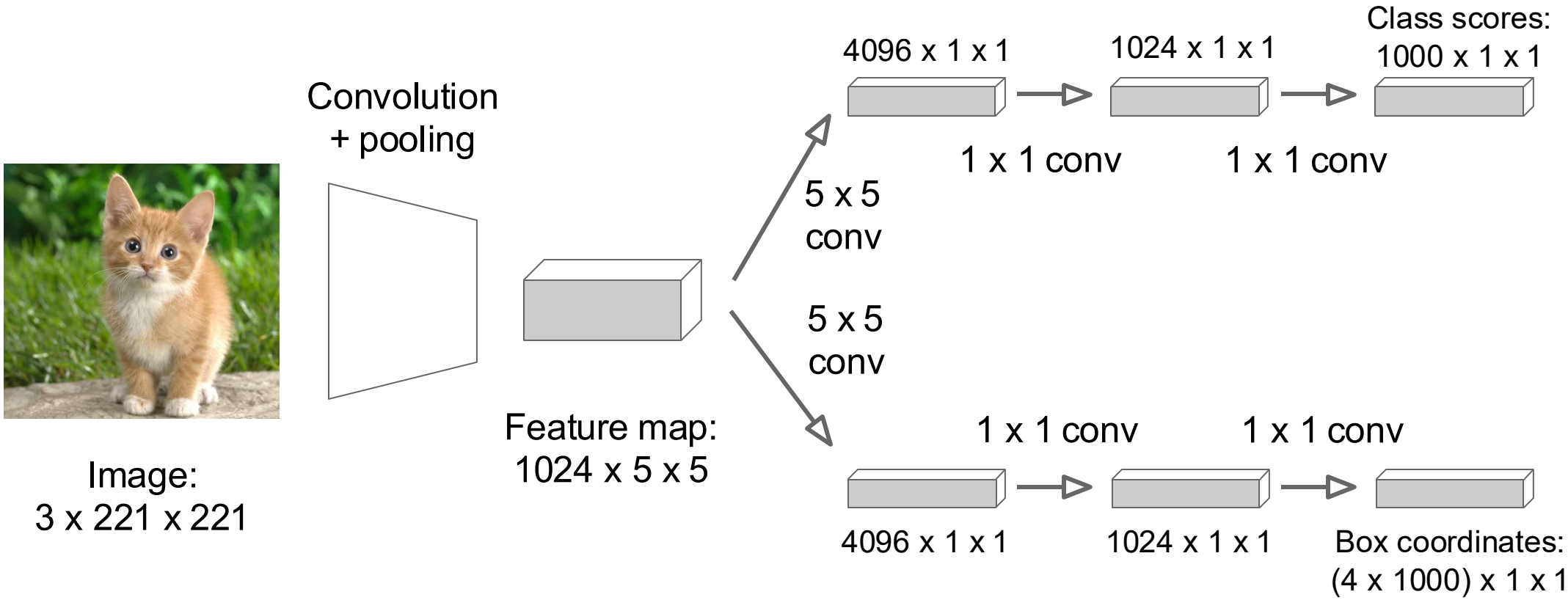


In sliding window, the window sizes are fixed and typically square in …

… but objects generally have different sizes and are not always square.

Combining localization with sliding window

By replacing the FC layers with 1x1 convolutions (i.e., by turning the network into a Fully Convolutional Network – **FCN**), we can perform classification and localization in a sliding window fashion.



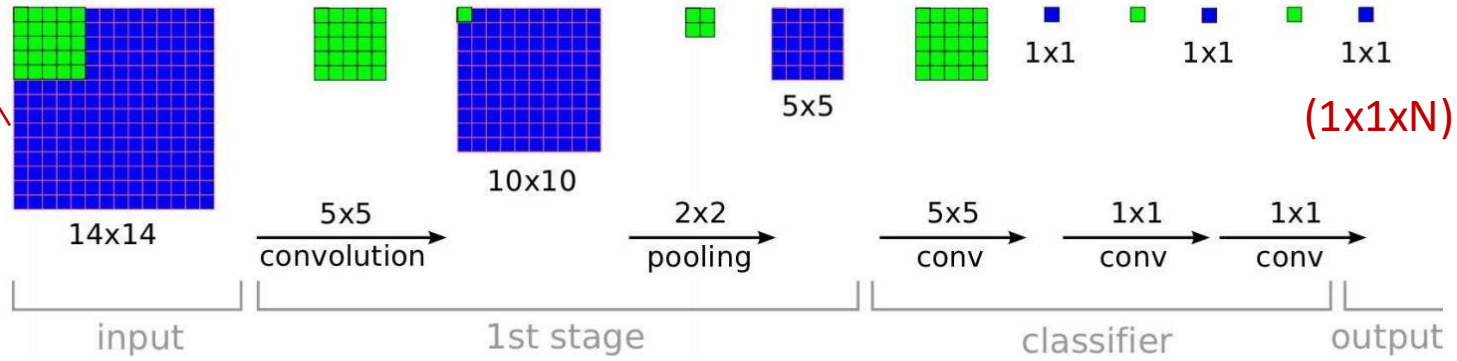
https://medium.com/coinmonks/review-of-overfeat-winner-of-ilsvrc-2013-localization-task-object-detection-a6f8b9044754

Training and testing an FCN

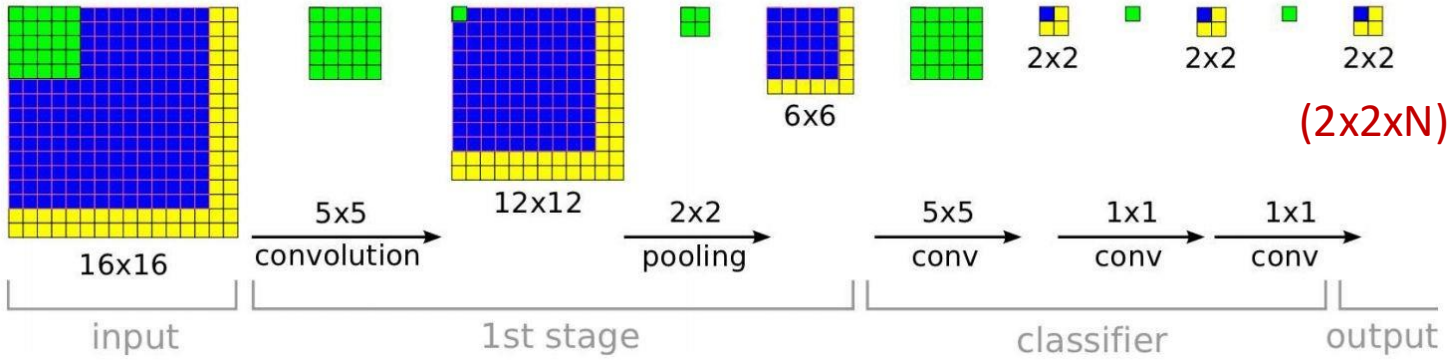
Blue: Feature map (depth not shown)
Green: Filter

Choose input size such that the output size is 1x1xN

Training time: Small image, 1 x 1 classifier output



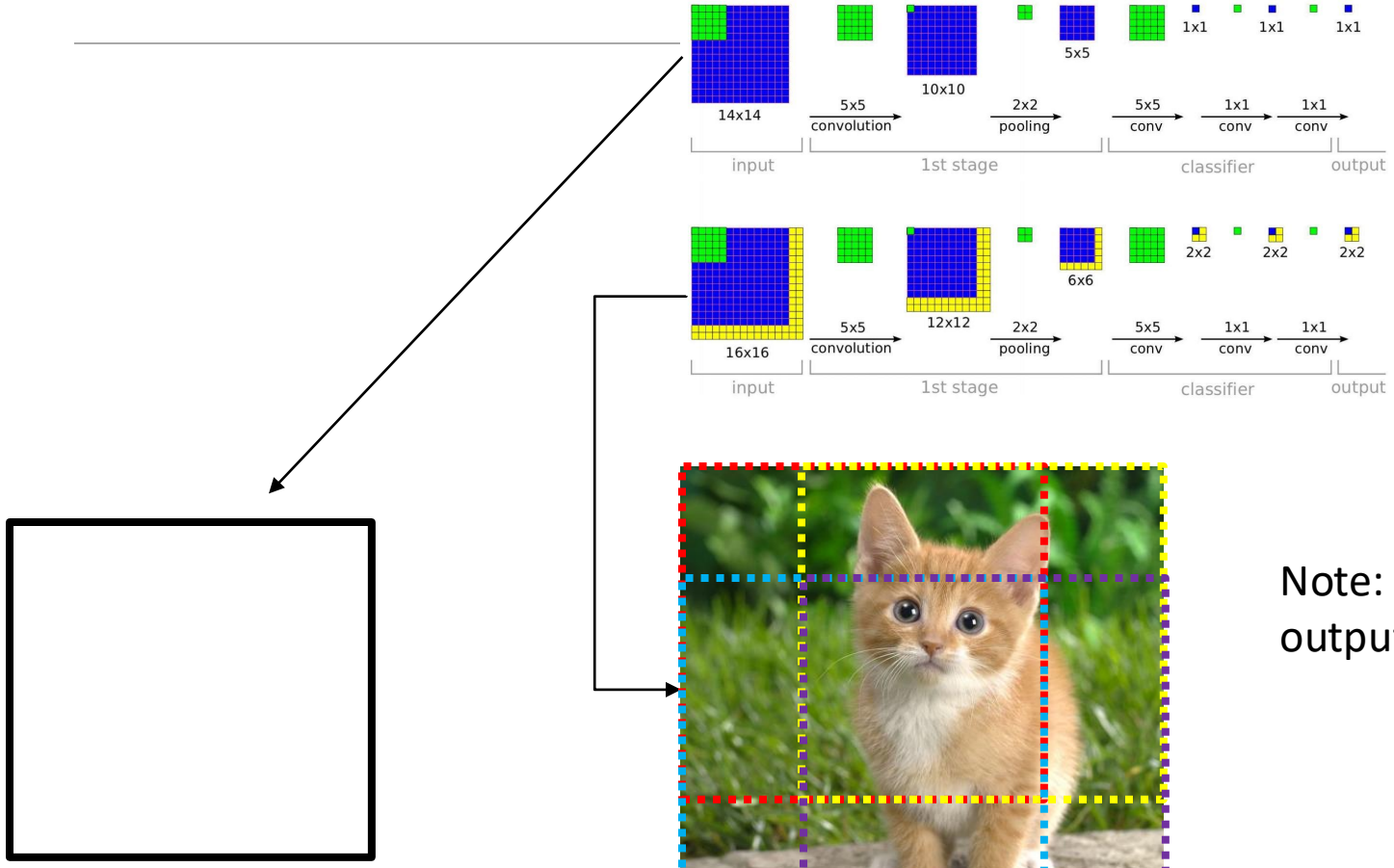
Test time: Larger image, 2 x 2 classifier output, only extra compute at yellow regions (= efficient sliding window)



"Overfeat": https://arxiv.org/pdf/1312.6229.pdf

https://medium.com/coinmonks/review-of-overfeat-winner-of-ilsvrc-2013-localization-task-object-detection-a6f8b9044754

Overfeat



Training time: Small image, 1 x 1 classifier output

Test time: Larger image, 2 x 2 classifier output, only extra compute at yellow regions

Train image size (smaller than test image size)

Test image size

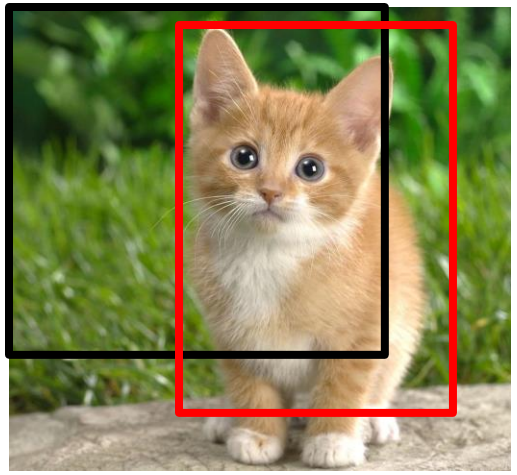
Note: This network is designed such that the output is 2x2. In general, it will be larger than that.

Network output: 2 x 2

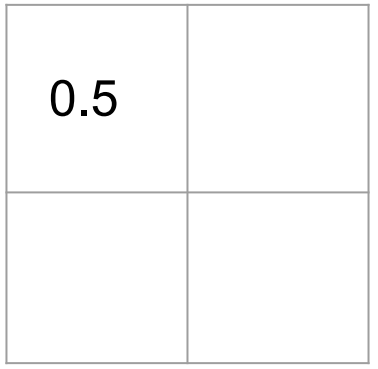
Overfeat

- For each output cell (black), we predict a bounding box (red) and a class score (for each class).
- Note that the predicted bounding box can extend beyond the output cell.

Train image size



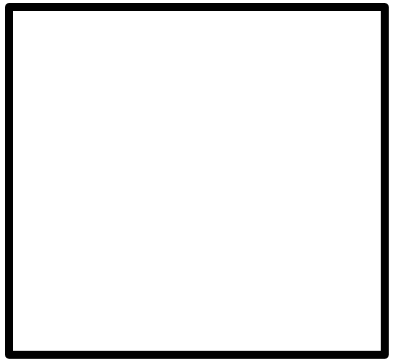
Predicted bounding box (localization) in larger image



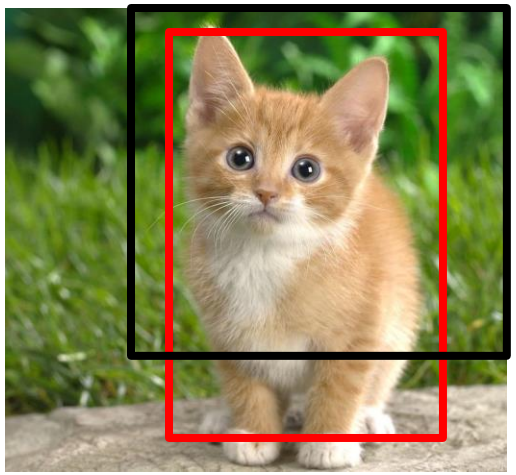
Classification scores: P(cat)

Overfeat

- For each output cell (black), we predict a bounding box (red) and a class score (for each class).
- Note that the predicted bounding box can extend beyond the output cell.



Train image size



Predicted bounding box (localization) in larger image

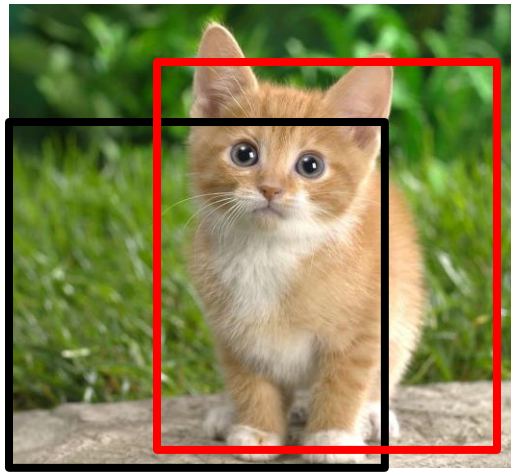
0.5	0.75

Classification scores: P(cat)

Overfeat

- For each output cell (black), we predict a bounding box (red) and a class score (for each class).
- Note that the predicted bounding box can extend beyond the output cell.

Train image size



Predicted bounding box (localization) in larger image

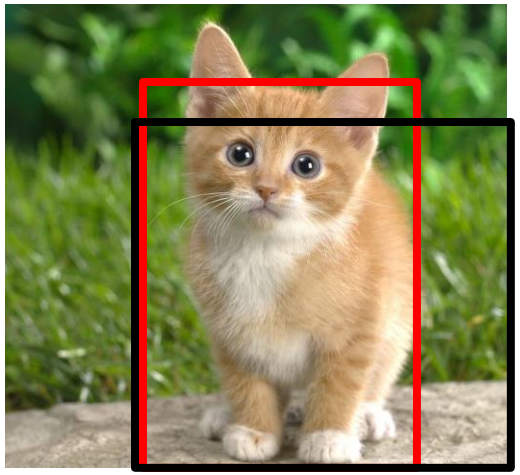
0.5	0.75
0.6	

Classification scores: P(cat)

Overfeat

- For each output cell (black), we predict a bounding box (red) and a class score (for each class).
- Note that the predicted bounding box can extend beyond the output cell.

Train image size



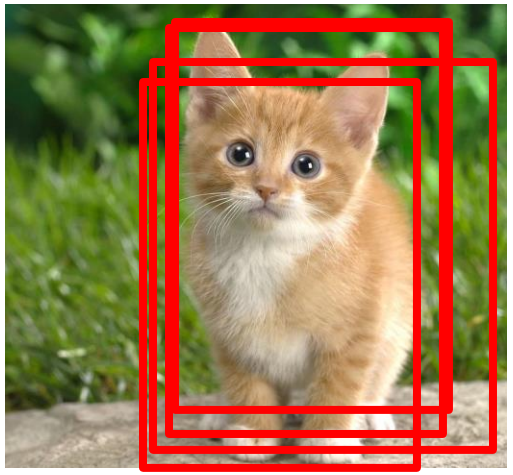
Predicted bounding box (localization) in larger image

0.5	0.75
0.6	0.8

Classification scores: P(cat)

Overfeat

Train image size



Predicted bounding box (localization) in larger image

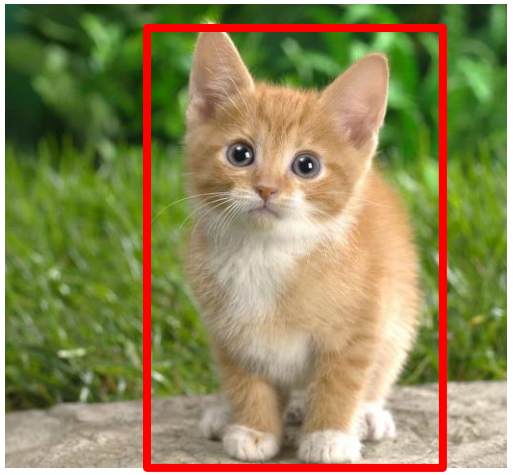
0.5	0.75
0.6	0.8

Classification scores: P(cat)

Overfeat

Greedily merge boxes and scores (details in paper)

Train image size



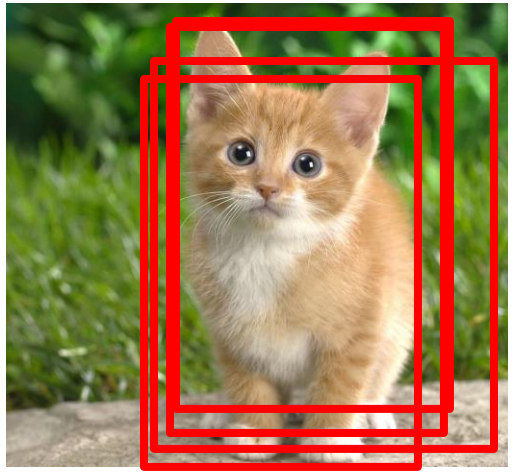
Predicted bounding box (localization) in larger image

0.8

Classification scores: P(cat)

Aside: Non-max suppression

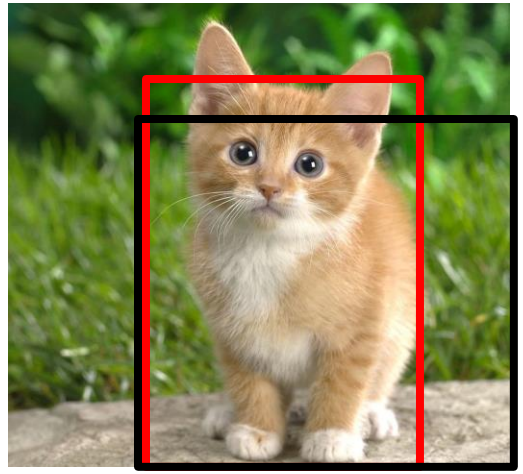
- Simple heuristic procedure that preserves only the best bounding boxes.
- Go down the list of detections starting from highest scoring.
- Eliminate any detection that overlaps highly with a higher scoring detection.



Predicted bounding box (localization) in larger image

0.5	0.75
0.6	0.8

Classification scores: P(cat)



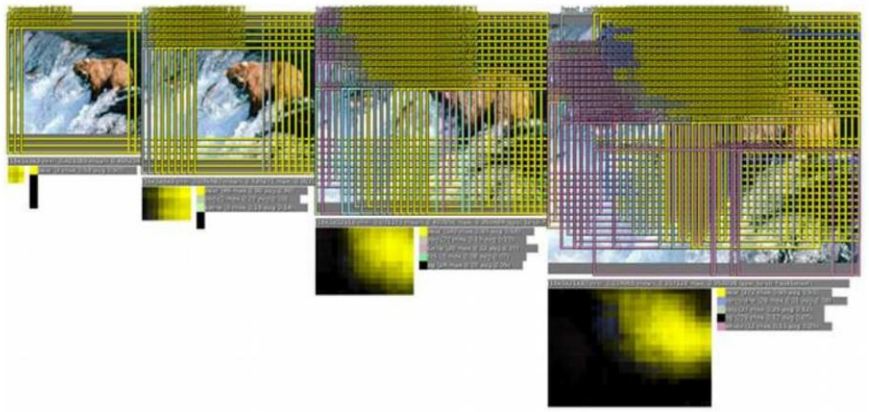
Best bounding box

https://www.pyimagesearch.com/2014/11/17/non-maximum-suppression-object-detection-python/

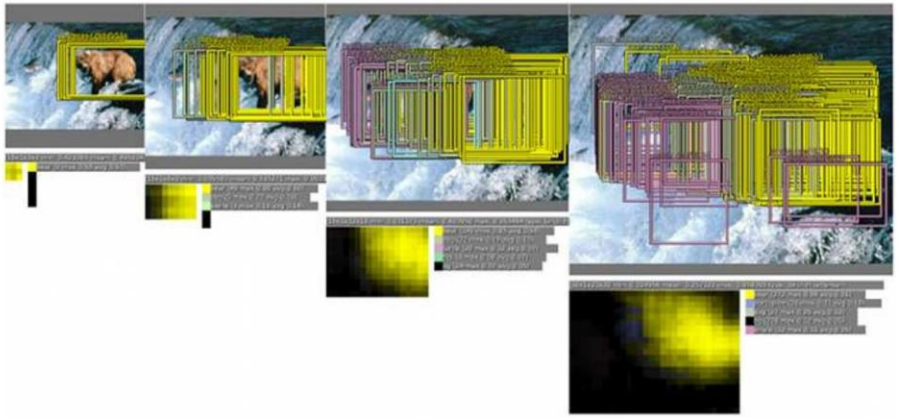
Overfeat

In practice use many sliding window locations and multiple scales (details in paper)
(to detect objects at different scales, change image size but keep window size fixed)

Window positions + score maps



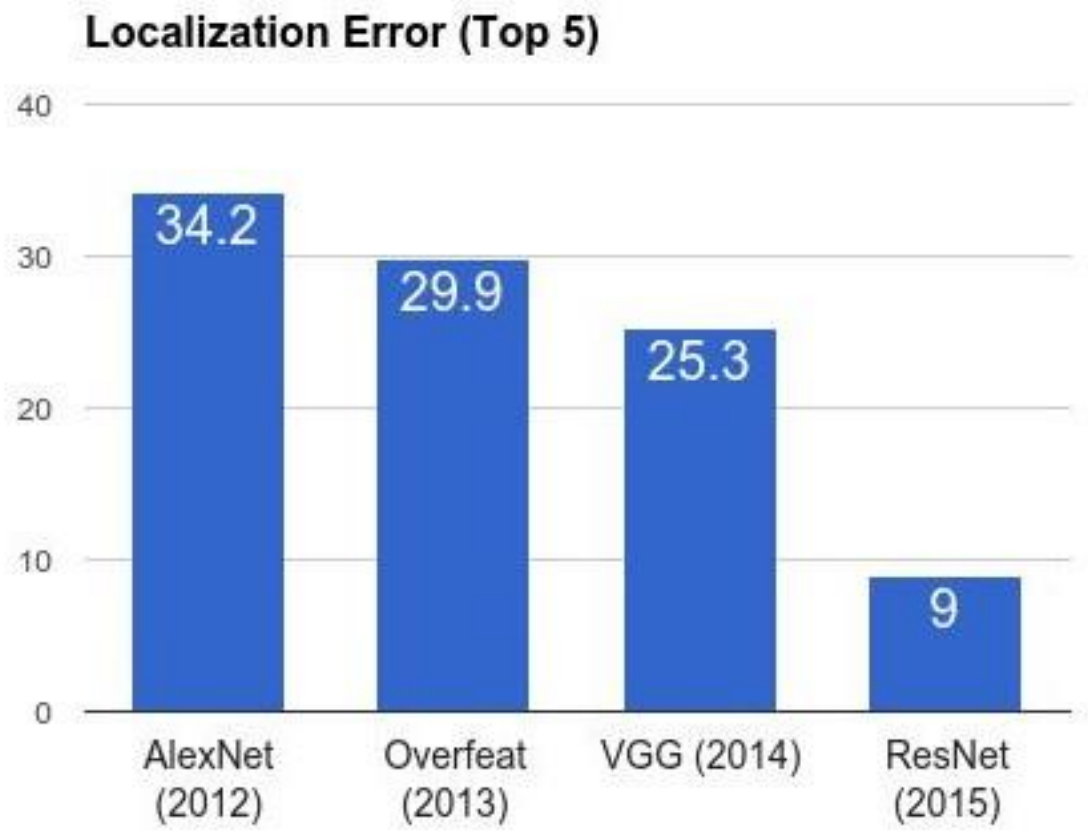
Box regression outputs



Final Predictions

"Overfeat": https://arxiv.org/pdf/1312.6229.pdf

ImageNet Classification + Localization



AlexNet: Localization method not published

Overfeat: Multiscale convolutional regression with box merging

VGG: Same as Overfeat, but fewer scales and locations; simpler method, gains all due to deeper features.

ResNet: Different localization method (RPN) and much deeper features.

Object detection

Computer vision tasks

Classification

Classification + Localization

Object Detection

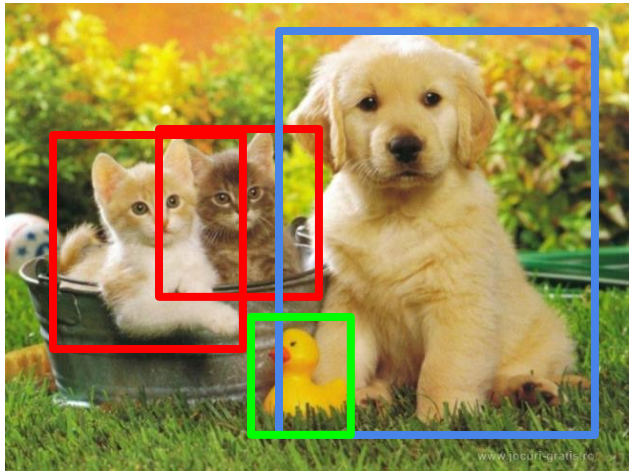
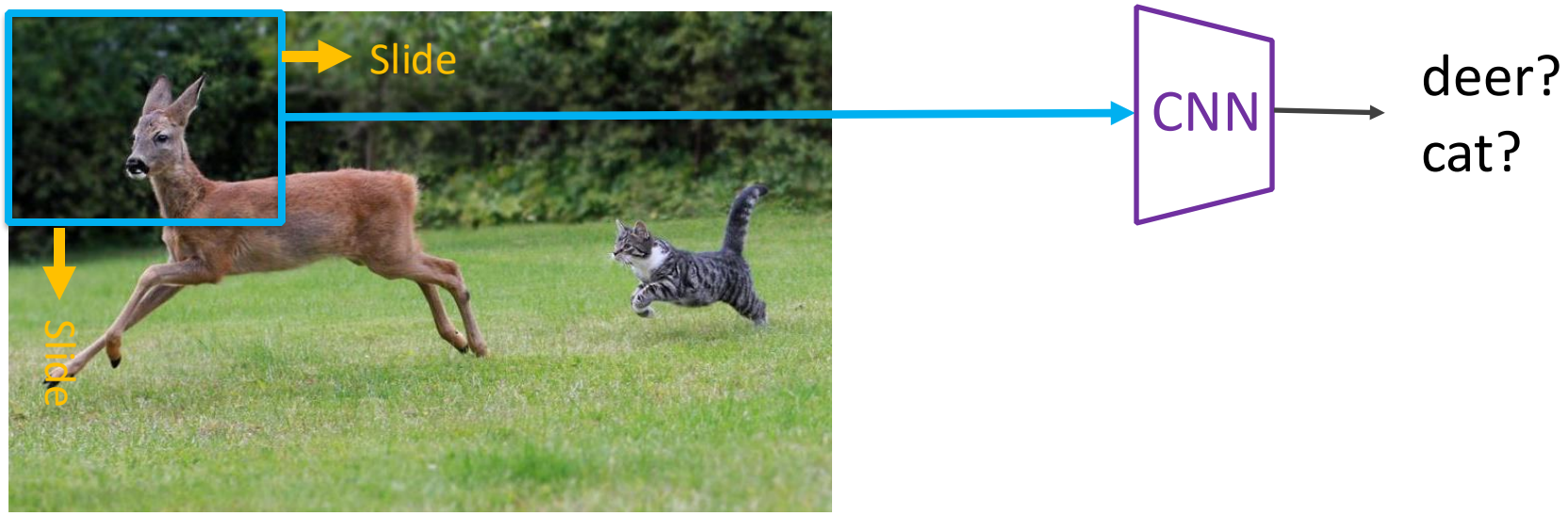


Image Segmentation

Multiple objects

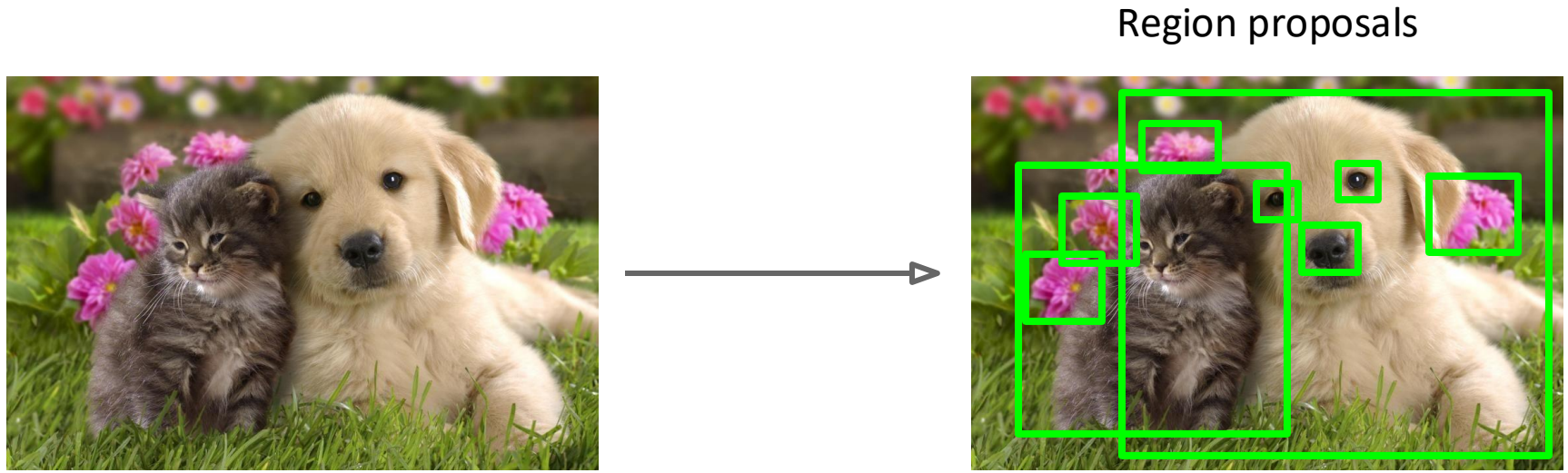
Object detection using sliding window

- **Problem:** Recall that for simple sliding window, we need to test many window positions and window scales (sizes) and use a computationally demanding classifier (CNN).
- **Idea:** Only look at a tiny subset of "good" candidate bounding boxes.



Region proposals with selective search

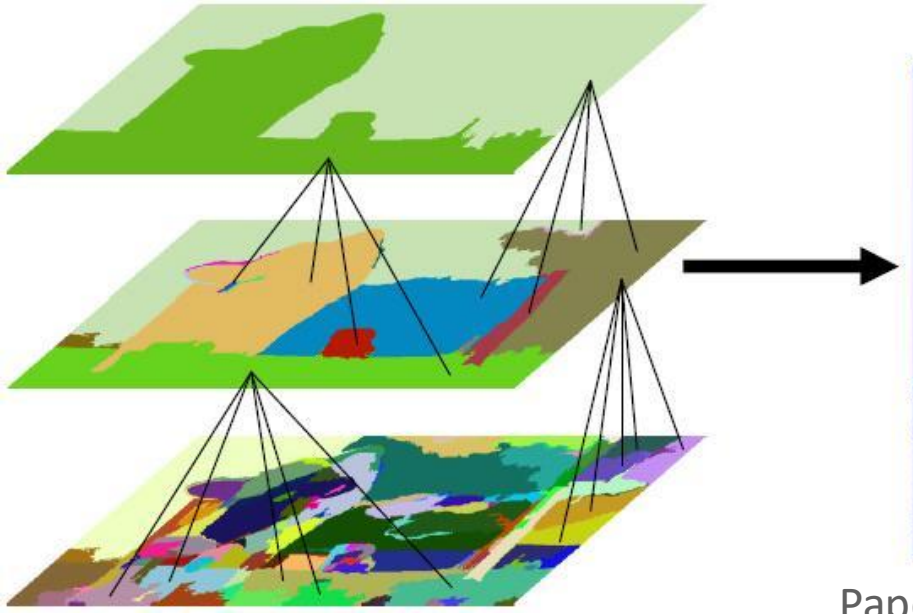
- **Basic idea:** Find candidate bounding boxes that are likely to contain an object (selective search) and run classifier only on those regions of the input image.
- Much faster than sliding window (exhaustive search).
- You can think of region proposals as a “class-agnostic” object detector.

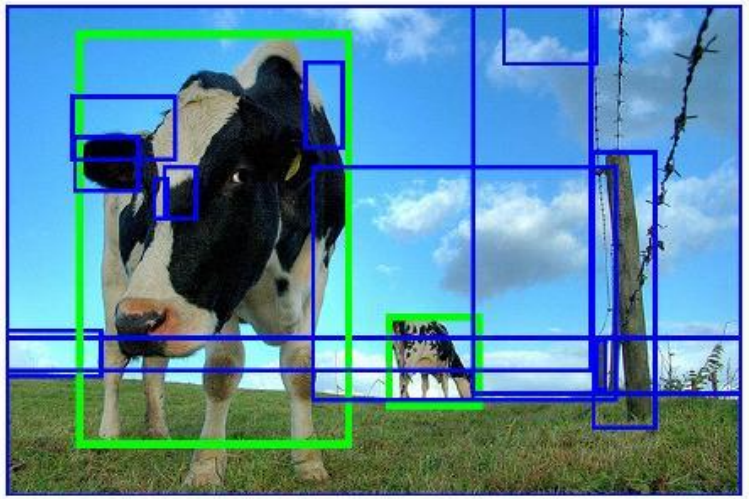


https://medium.com/brillio-data-science/object-detection-part-1-introduction-to-object-detection-and-region-proposals-68f6624c98f5

Region proposals with selective search

- Based on traditional, cluster-based image segmentation (groups similar pixels together).
- Many different segmentation algorithms (like k-means on color, k-means on color+position, etc.) with many hyperparameters (number of clusters, weights on edges).

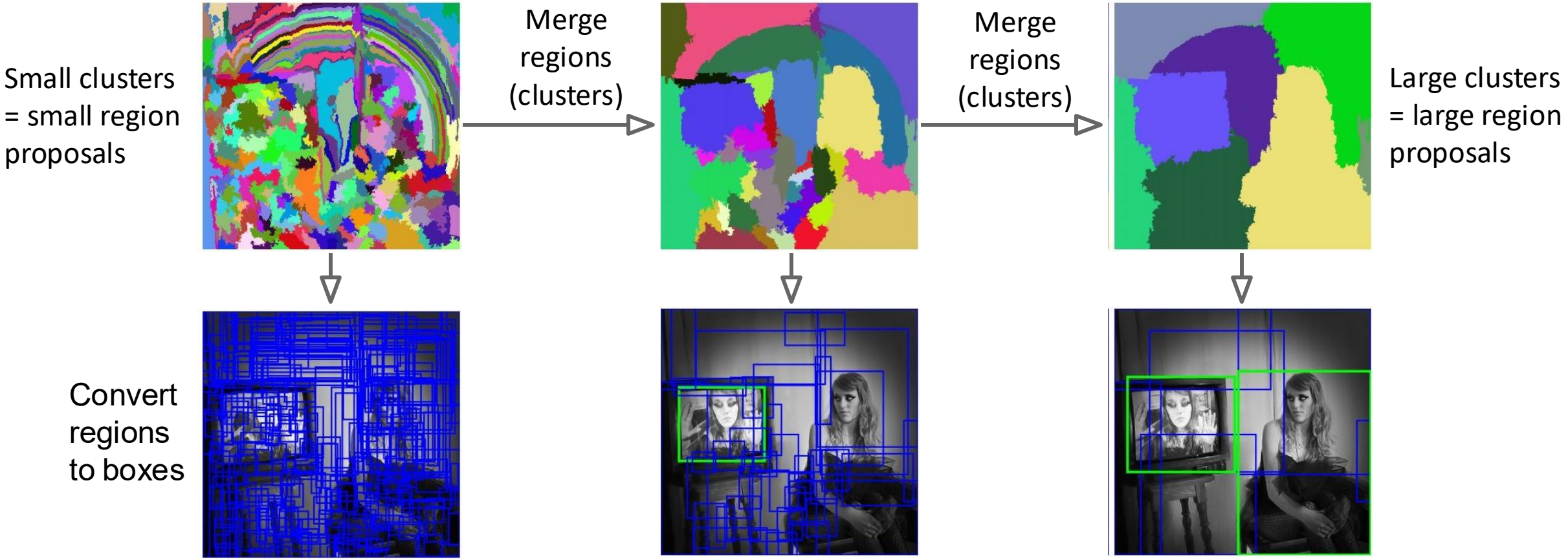




Paper: Selective Search for Object Recognition

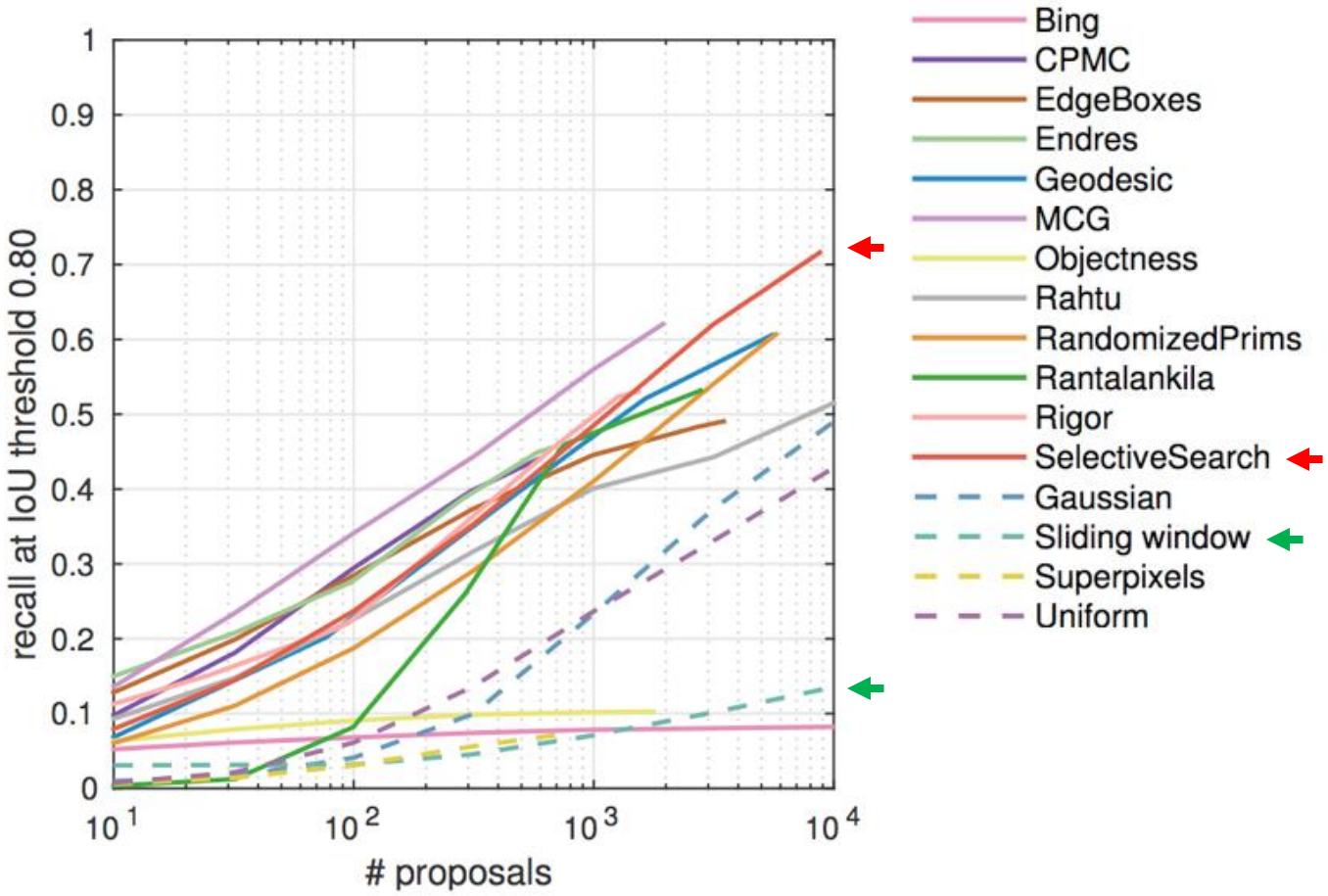
Region proposals with selective search

- Bottom-up segmentation, merging regions at multiple scales (also called super pixels)



Many region proposal strategies

- Tens of ways of generating region proposals.
- What fraction of ground truth objects have proposals near them?
- What the figure shows:
  - With **sliding window**, you will be wasting a lot of time – most of the object proposals (windows) do not contain an object.
  - With **selective search**, there is a much higher chance that an object proposal contains an object



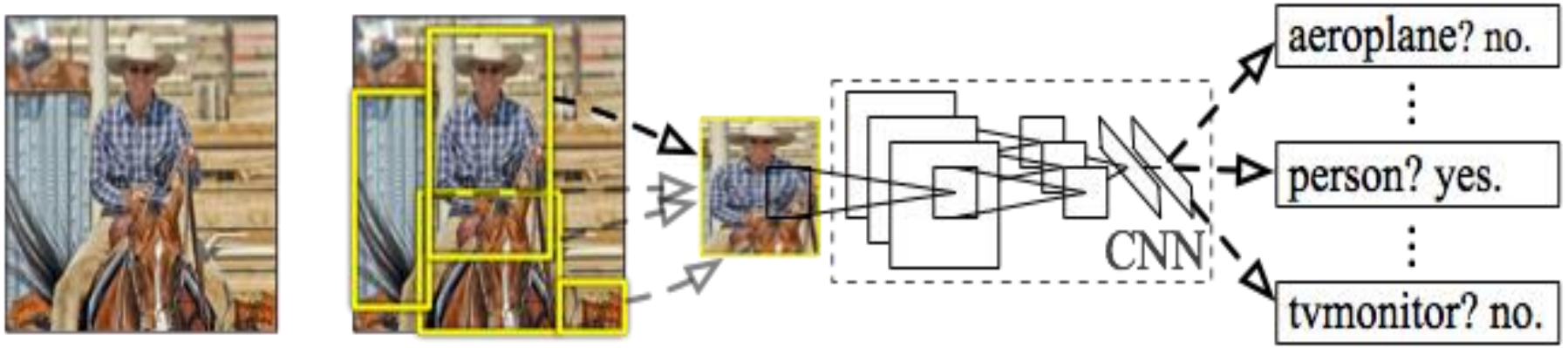
What makes for effective detection proposals? J. Hosang, R. Benenson, P. Dollar, B. Schiele. In TPAMI

What do we do with proposals?

- Each proposal is a group (cluster) of pixels.
- Take tight fitting box and classify it using a classifier
- Can leverage any image classification approach, like CNN or Support Vector Machines (SVMs)

R-CNN: Regions with CNN features

- For each region proposal, run through CNN (like AlexNet) and extract 4096-dimensional feature vector from last fully connected layer (fc_7).
- Use Support Vector Machine (SVM) classifier of pre-computed feature vectors.



Input image

Extract region proposals (~2k / image)

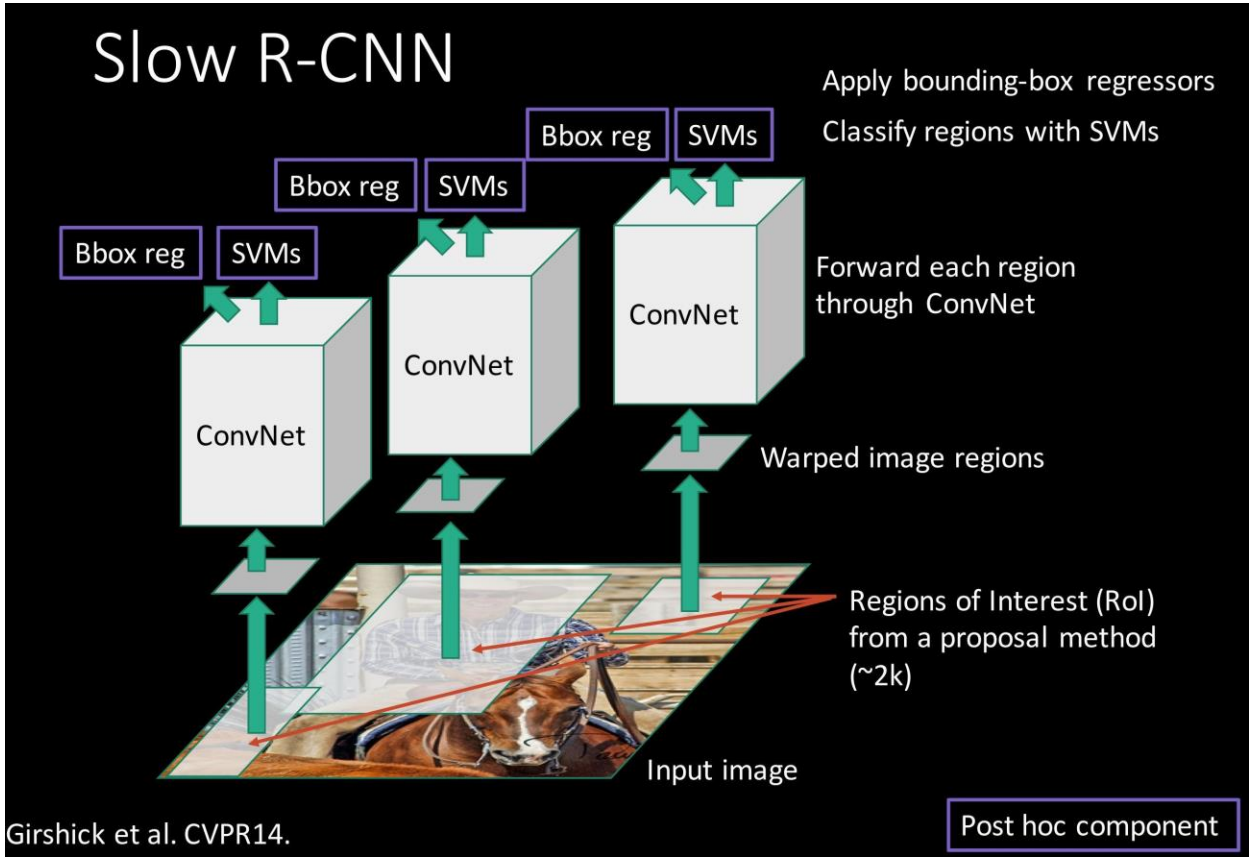
Compute CNN features

Classify regions (linear SVM)

R-CNN: https://people.eecs.berkeley.edu/~rbg/papers/r-cnn-cvpr.pdf

R-CNN steps

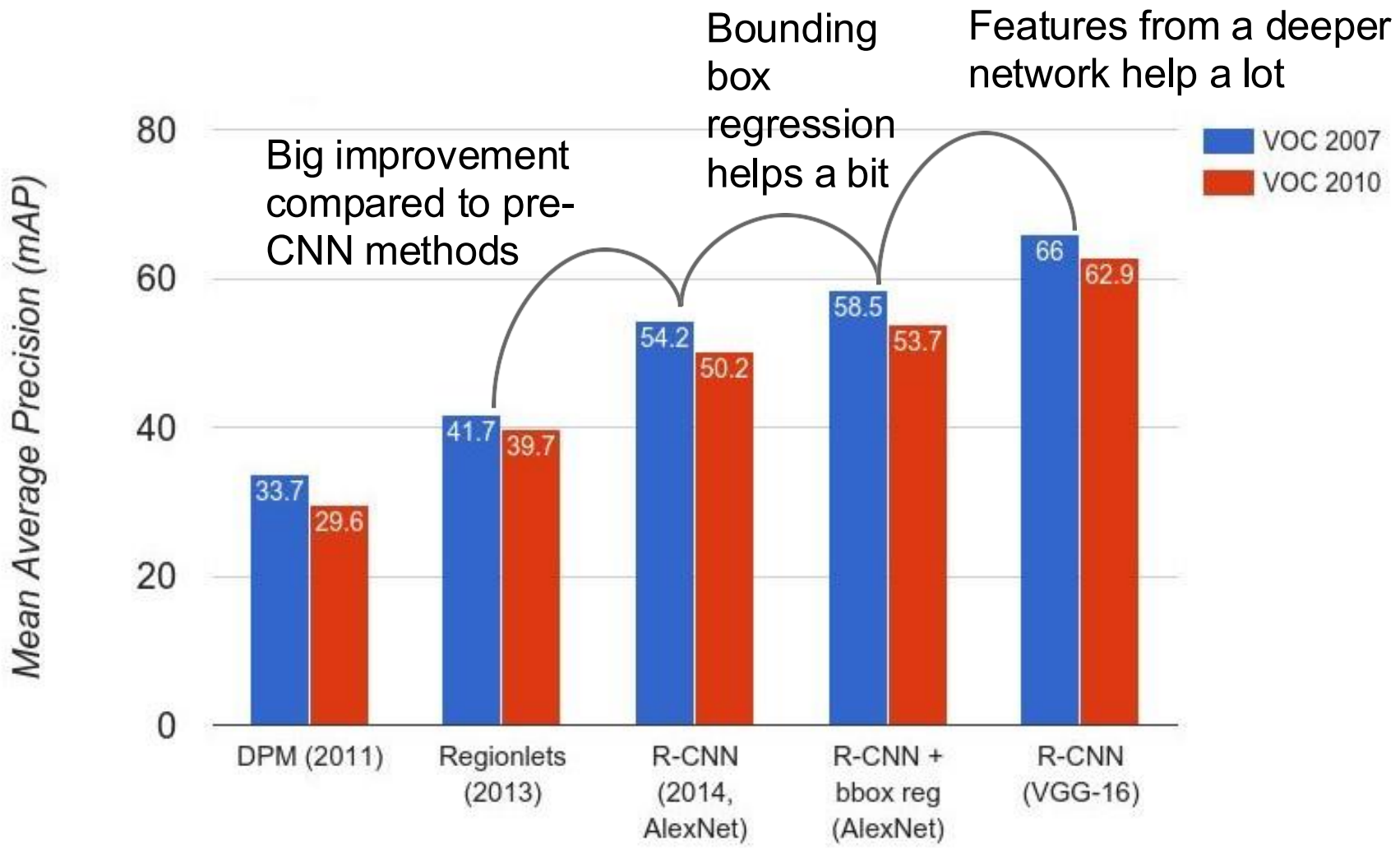
- **First step:** Detect 2000 regions of interest (RoI) using selective search.
- **Second step:** Candidate regions must be warped to input shape that AlexNet expects (227x227).
- **Third step:** Get 4096-dimensional feature vector from modified AlexNet that recognizes 20 classes + background (PASCAL VOC dataset – next slide).
- **Fourth step:** Classify using SVM and predict bounding box coordinates using regression for more precise localization.



Object detection datasets

	PASCAL VOC (2010)	ImageNet Detection (ILSVRC 2014)	MS-COCO (2014)
Number of classes	20	**200**	80
Number of images (train + val)	~20k	**~470k**	~120k
Mean objects per image	2.4	1.1	**7.2**

R-CNN results

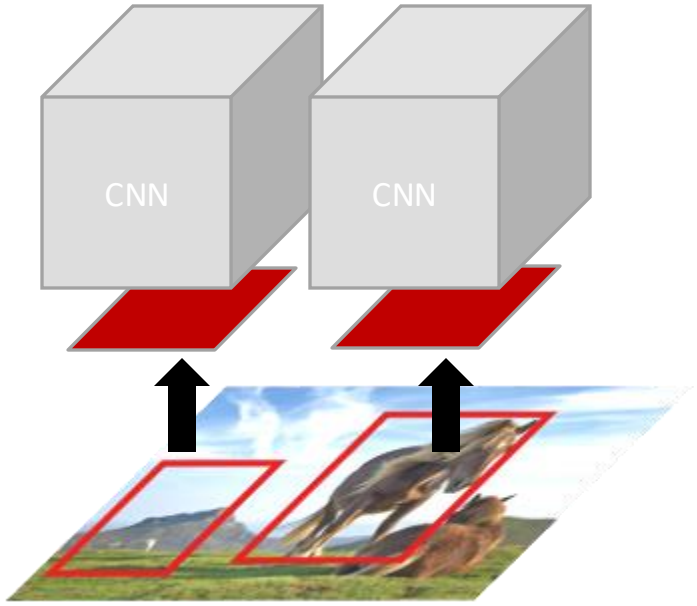


R-CNN problems

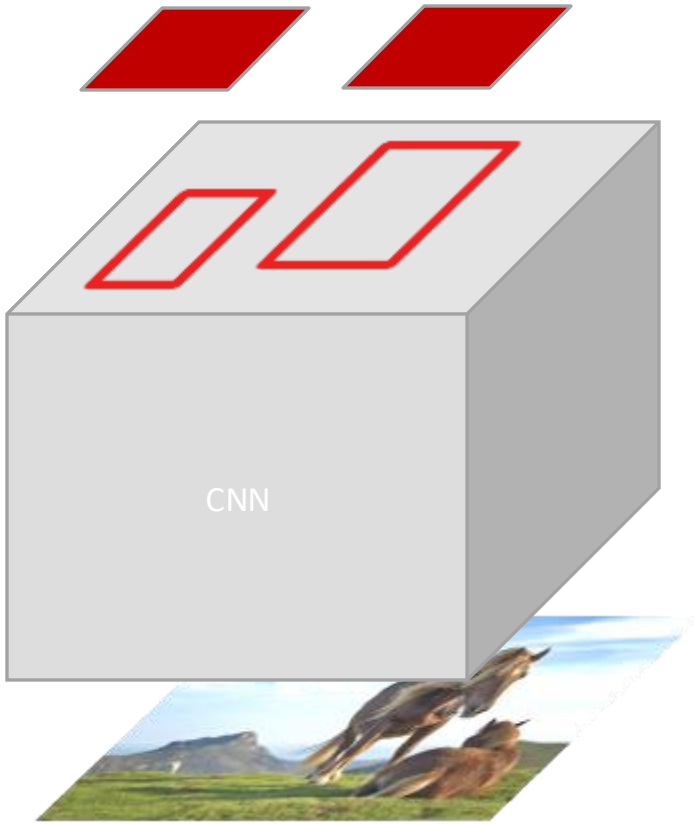
- **Slow at test-time:** need to run full forward pass of CNN for each of 2000 region proposals (takes around 47 seconds for each test image).
- **SVMs and regressors are post-hoc:** CNN features not updated/trained in response to SVMs and regressors.
- **Complex multistage training pipeline**, as you would have to classify 2000 region proposals per image.
- The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

Fast R-CNN

- **R-CNN problem #1:** Slow at test-time due to independent forward passes of the CNN.
- **Solution:** Share computation of convolutional layers between proposals for an image.



R-CNN

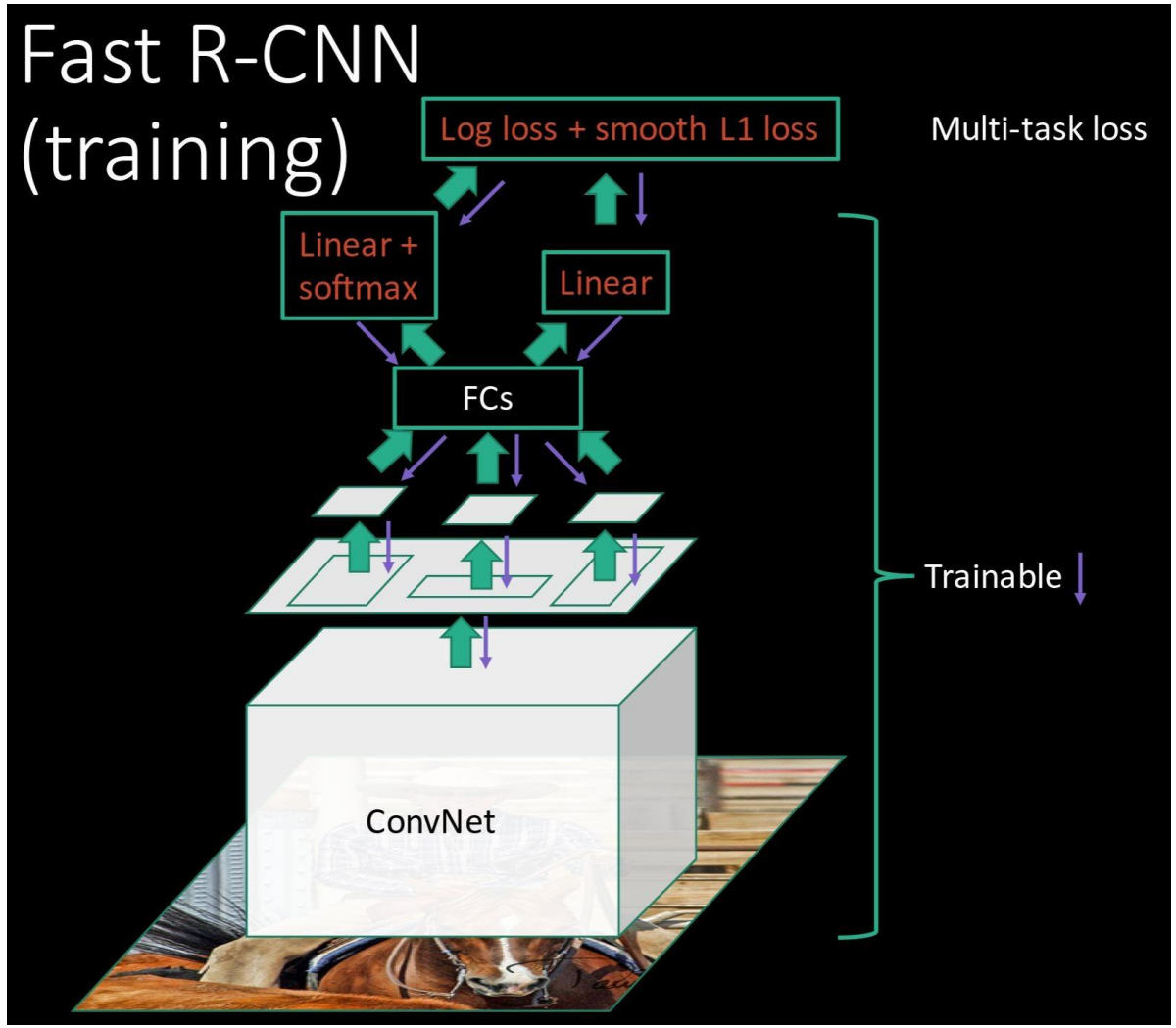


Fast R-CNN

Fast R-CNN: https://arxiv.org/abs/1504.08083

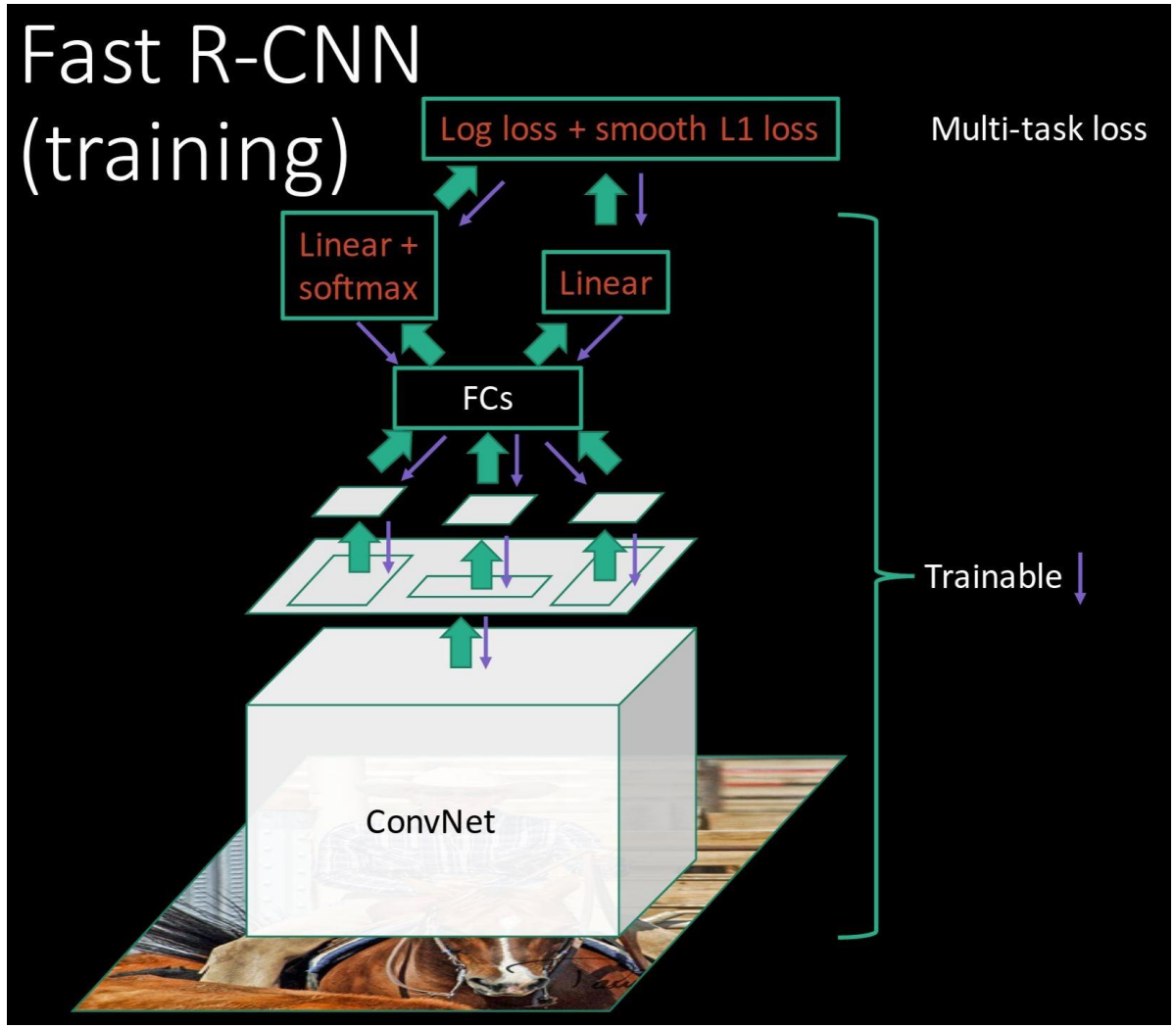
Fast R-CNN

- **R-CNN problem #2:** Post-hoc training: CNN features not updated/trained in response to SVMs and regressors.
- **R-CNN problem #3:** Complex training pipeline**.**
- **Solution:** Just train the whole system end-to-end all at once!



Fast R-CNN

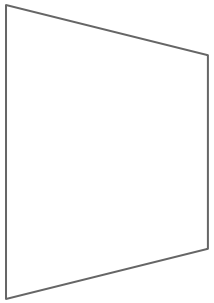
- **Q:** What are the two outputs ("Linear + softmax" and "Linear")?
- **A:**
  - "Linear + softmax" is the classifier
  - "Linear" is the bounding box (bbox) regressor
- Combined multi-task loss is sum of
  - classification loss (cross entropy = "Log loss")
  - bbox loss ("smooth L1 loss").
- **Q:** Fully connected layers expect fixed input shape – are input shapes fixed?
- **A:** No, different regions have different shapes.
- **Solution:** RoI pooling

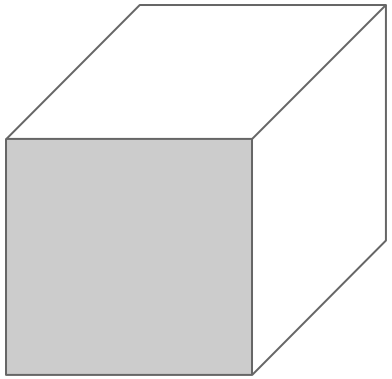


Fast R-CNN: Region of Interest (RoI) Pooling

Hi-res input image:
3 x 800 x 600
with region
proposal

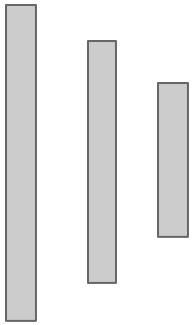
Convolution
and Pooling





Hi-res conv features:
C x H x W
with region proposal

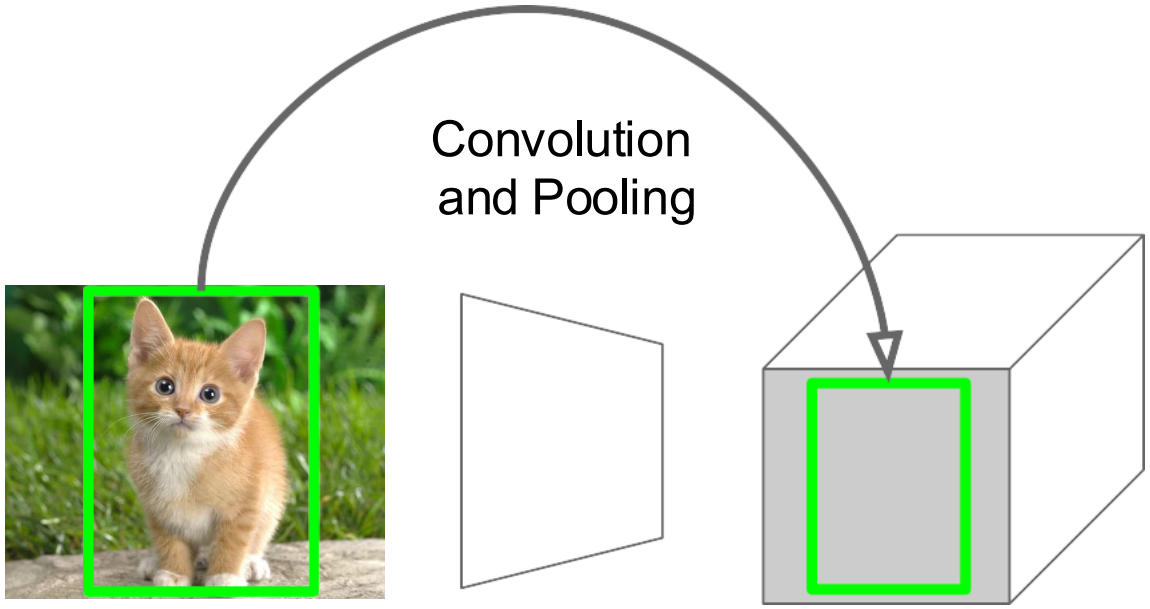
Fully-connected
layers



Problem: Fully-connected layers expect low-res conv features: C x h x w

https://deepsense.ai/blog/region-of-interest-pooling-explained/

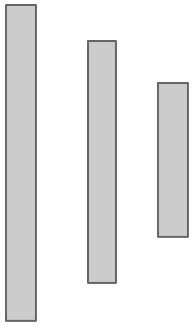
Fast R-CNN: Region of Interest (RoI) Pooling



Hi-res input image:
3 x 800 x 600
with region
proposal

Hi-res conv features:
C x H x W
with region proposal

Fully-connected
layers



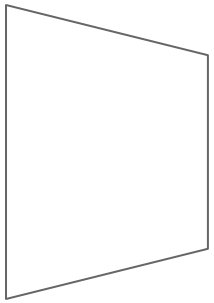
Problem: Fully-connected layers expect low-res conv features with a **fixed shape**: C x h x w

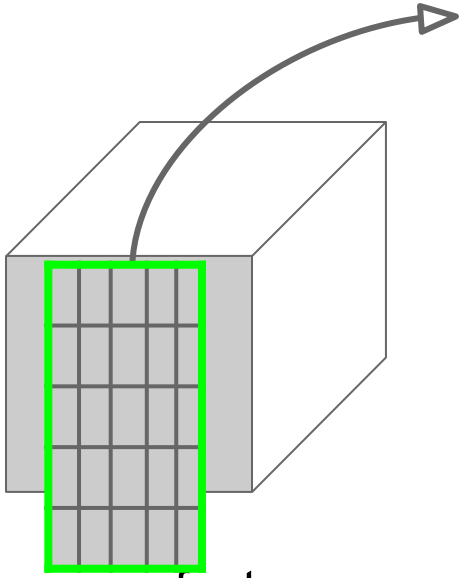
https://deepsense.ai/blog/region-of-interest-pooling-explained/

Fast R-CNN: Region of Interest (RoI) Pooling

Hi-res input image:
3 x 800 x 600
with region
proposal

Convolution
and Pooling

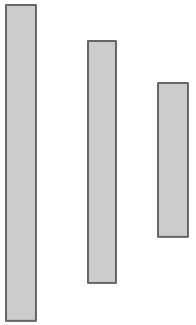




Hi-res conv features:
C x H x W
with region proposal

Divide projected
region into h x w grid,
where h and w are
predetermined
constants (e.g., h=w=5)

Fully-connected
layers



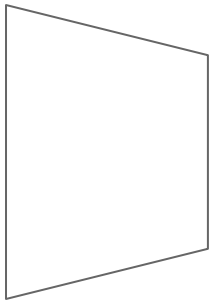
Problem: Fully-connected layers expect low-res conv features with a **fixed shape**: C x h x w

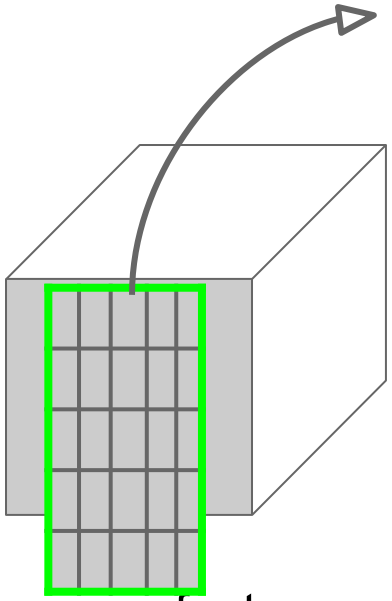
https://deepsense.ai/blog/region-of-interest-pooling-explained/

Fast R-CNN: Region of Interest (RoI) Pooling

Hi-res input image:
3 x 800 x 600
with region
proposal

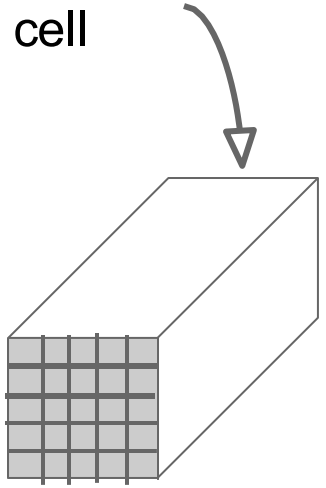
Convolution
and Pooling





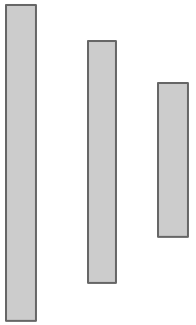
Hi-res conv features:
C x H x W
with region proposal

Max-pool within
each grid cell



RoI conv features:
C x h x w
for region proposal

Fully-connected
layers



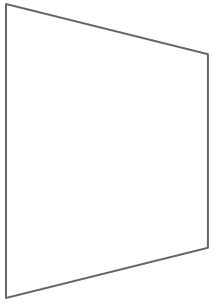
Problem: Fully-connected layers expect low-res conv features with a **fixed shape**: C x h x w

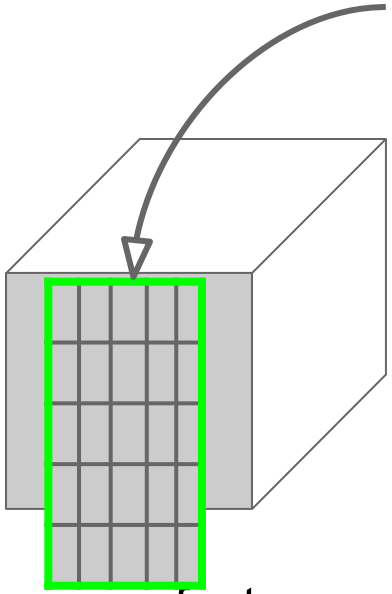
https://deepsense.ai/blog/region-of-interest-pooling-explained/

Fast R-CNN: Region of Interest (RoI) Pooling

Hi-res input image:
3 x 800 x 600
with region
proposal

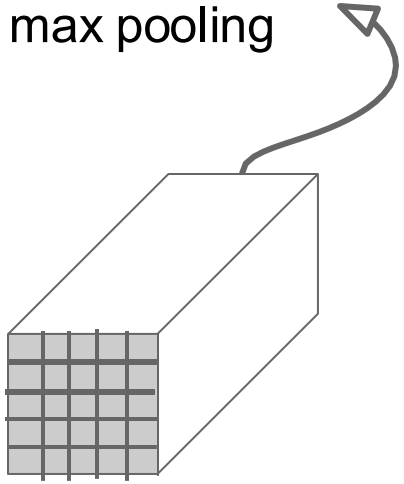
Convolution
and Pooling





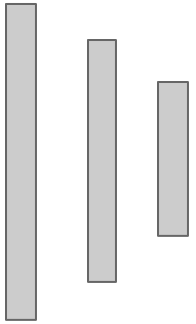
Hi-res conv features:
C x H x W
with region proposal

Can back propagate
similar to max pooling



RoI conv features:
C x h x w
for region proposal

Fully-connected
layers



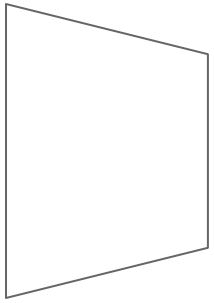
Problem: Fully-connected layers expect low-res conv features with a **fixed shape**: C x h x w

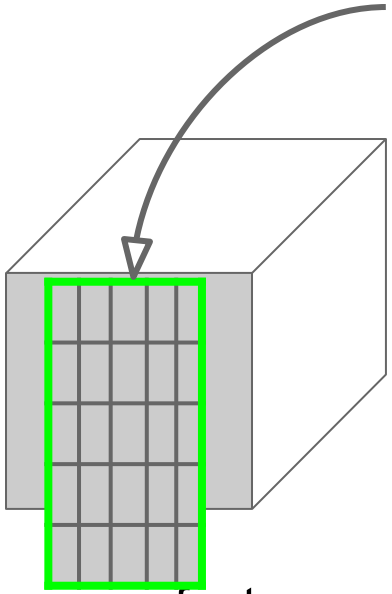
https://deepsense.ai/blog/region-of-interest-pooling-explained/

Fast R-CNN: Region of Interest (RoI) Pooling

Hi-res input image:
3 x 800 x 600
with region
proposal

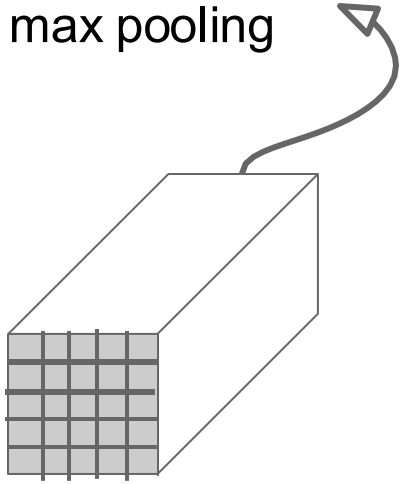
Convolution
and Pooling





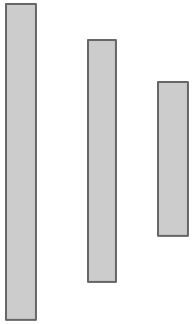
Hi-res conv features:
C x H x W
with region proposal

Can back propagate
similar to max pooling



RoI conv features:
C x h x w
for region proposal

Fully-connected
layers

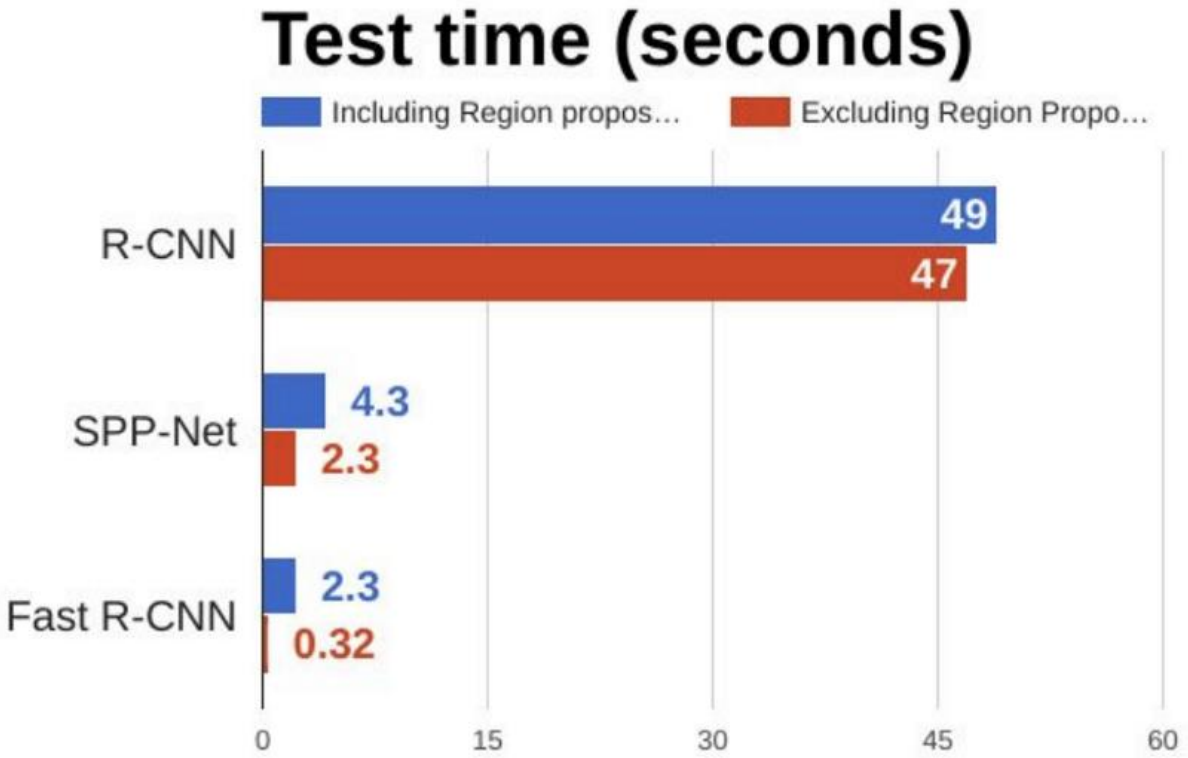


Problem: Fully-connected layers expect low-res conv features with a **fixed**
x h x w

Fast R-CNN is trained end-to-end. This figure shows only the classifier pipeline. There is a similar pipeline for the bounding box regressor

https://deepsense.ai/blog/region-of-interest-pooling-explained/

Fast R-CNN results



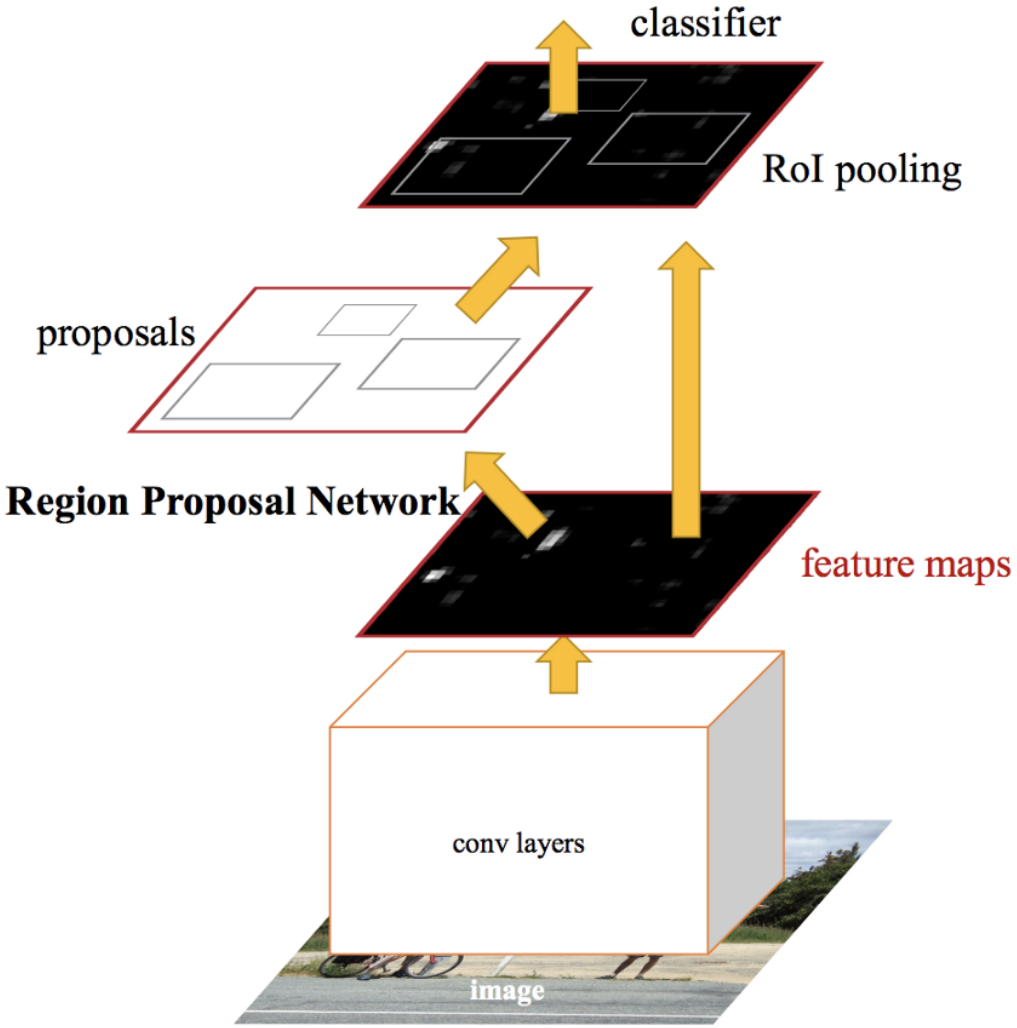
	R-CNN	**Fast R-CNN**
mAP (VOC 2007)	66.0	**66.9**

Fast R-CNN problems

- The performance of Fast R-CNN during testing time, including region proposals slows down the algorithm significantly when compared to not using region proposals.
- Therefore, **region proposals become bottlenecks** in Fast R-CNN algorithm affecting its performance.
- **Solution:** Just make the CNN do region proposals too!

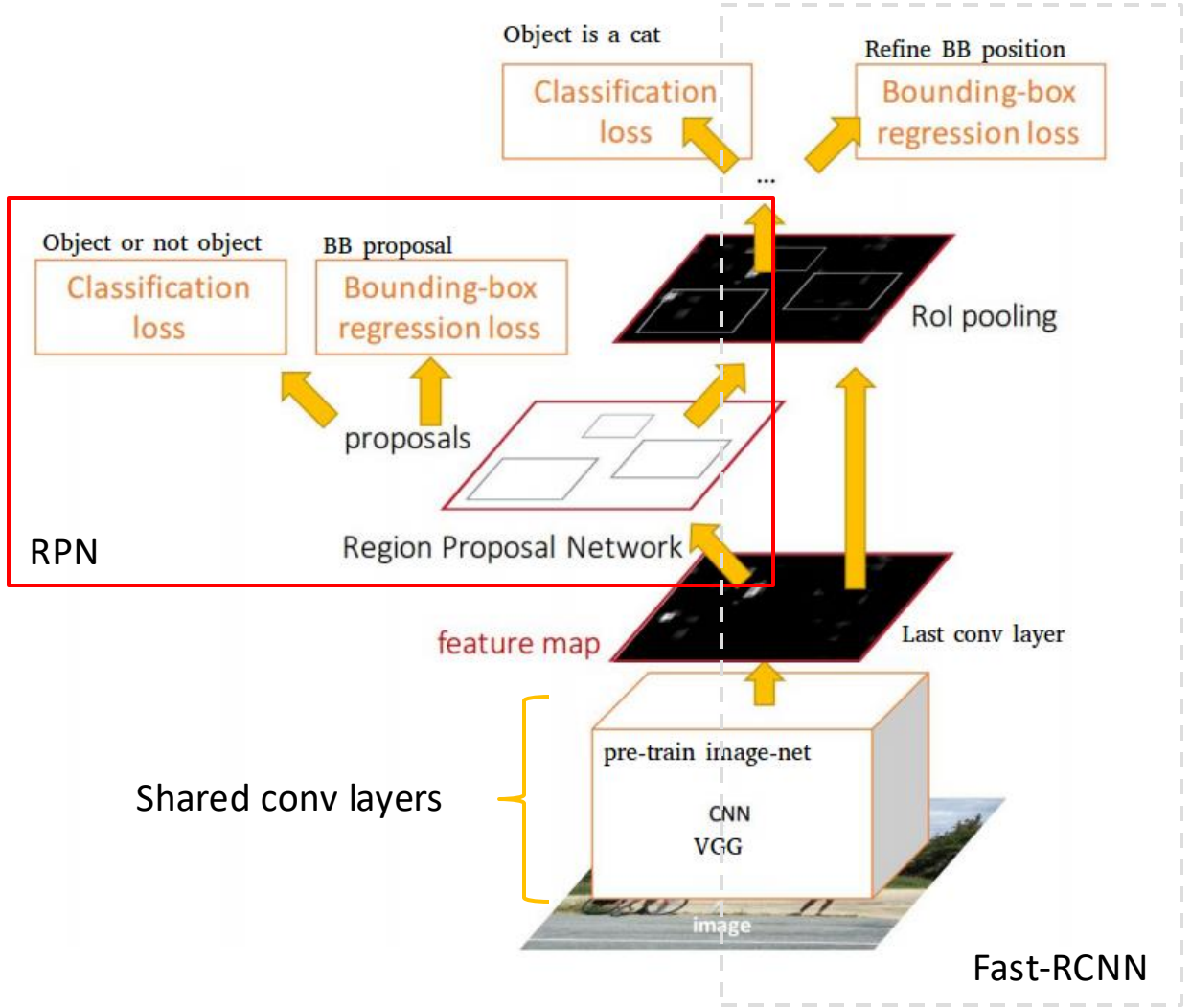
Faster R-CNN

- Insert a Region Proposal Network (RPN) after the last convolutional layer.
- RPN trained to produce region proposals directly; no need for external region proposals!
- After RPN, use RoI Pooling and an upstream classifier and bounding box regressor just like Fast R-CNN.
- Review: https://medium.com/data-science/review-faster-r-cnn-object-detection-f5685cb30202



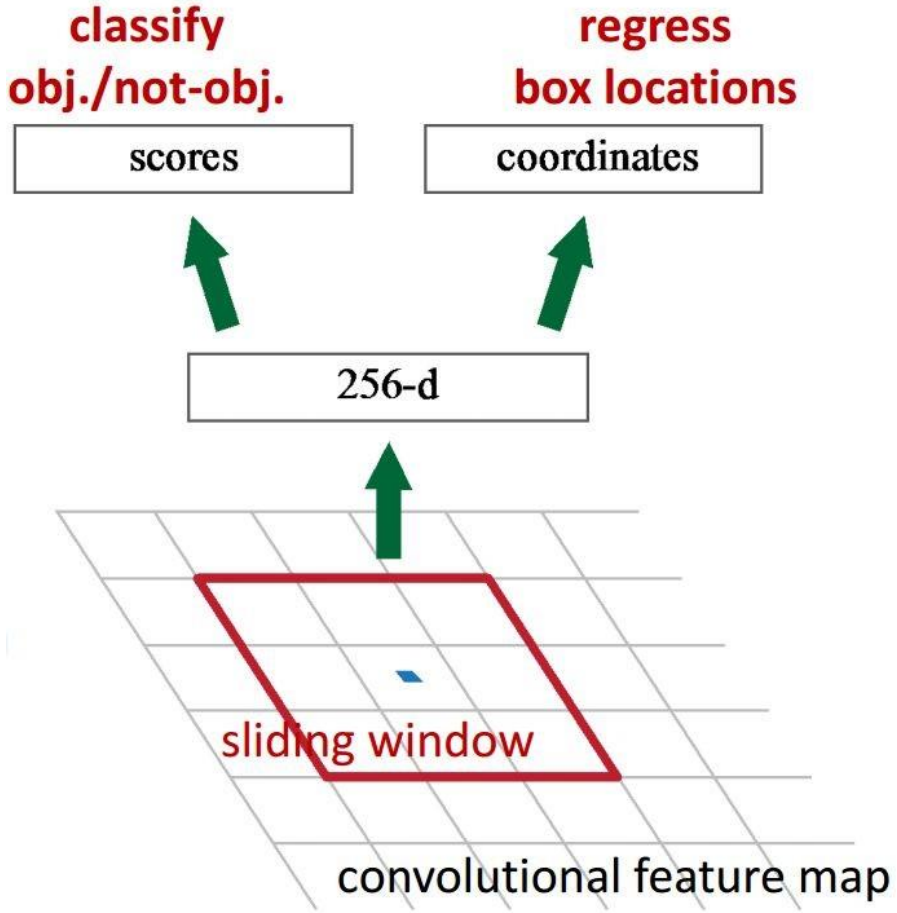
Faster R-CNN: https://arxiv.org/abs/1506.01497

Faster R-CNN full architecture



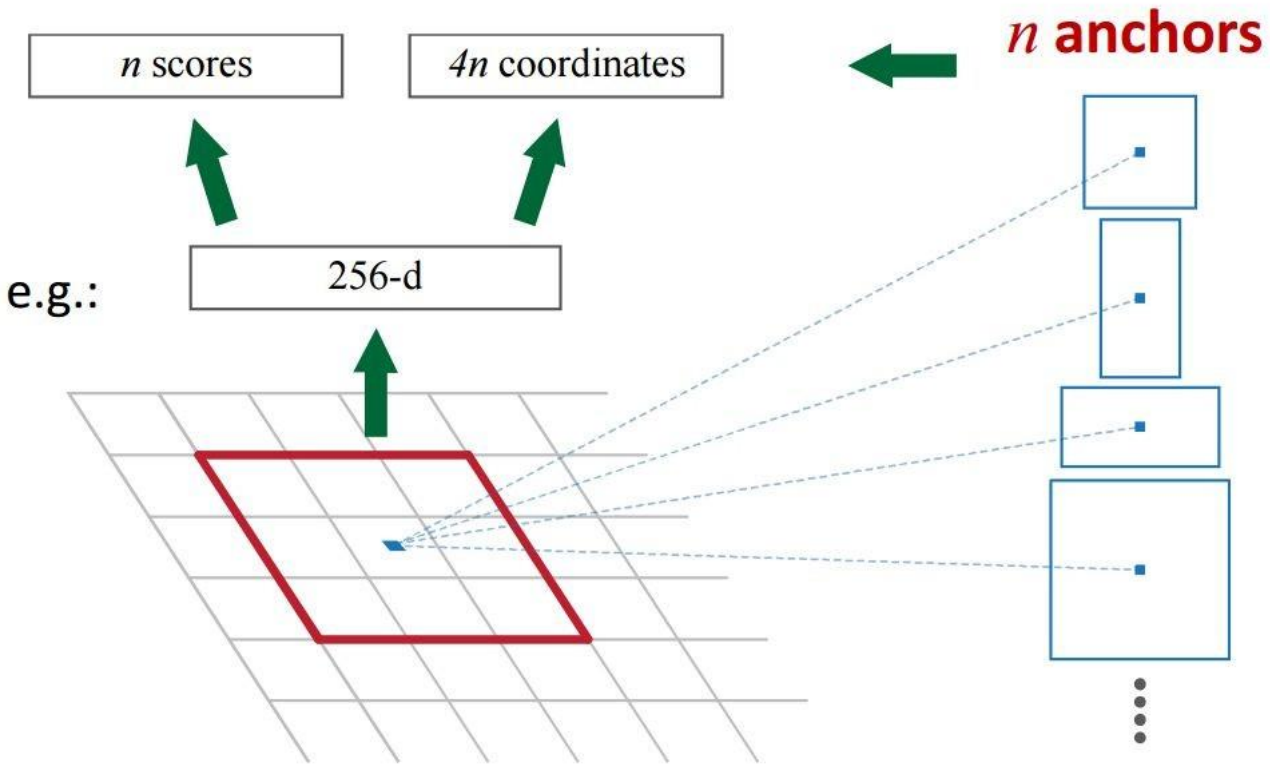
Faster R-CNN: Region Proposal Network

- Slide a small window over the last feature map of the CNN encoder.
- Build a small network for:
  - classifying object or not-object, and
  - regressing bbox locations
- Position of the sliding window provides localization information with reference to the image.
- Box regression provides finer localization information with reference to this sliding window.



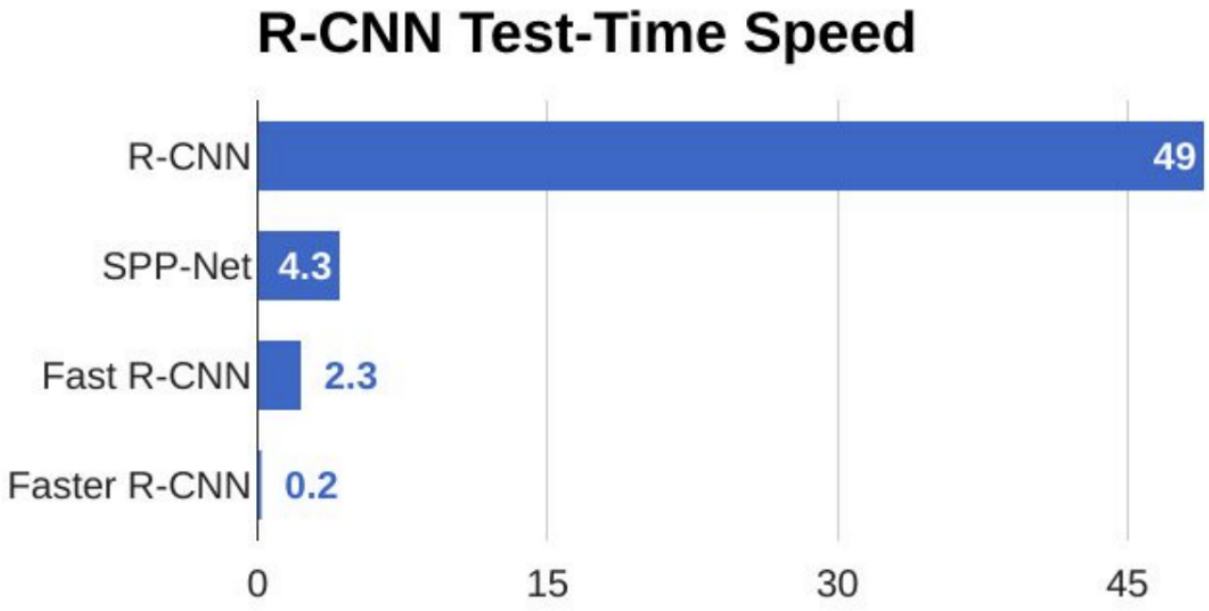
Faster R-CNN: Region Proposal Network

- At each location, consider boxes of many different sizes and aspect ratios
- Use **n anchor boxes** at each location.
- Anchors are **translation invariant:** use the same ones at every location
- Regression gives offsets from anchor boxes.
- Classification gives the probability that each (regressed) anchor shows an object.



Faster R-CNN results and comments

- In the paper: Ugly training pipeline
  - Use alternating optimization to train RPN, then Fast R-CNN with RPN proposals, etc.
  - More complex than it needs to be
- Since publication: Joint training! One network, four losses
  - RPN classification (anchor good / bad)
  - RPN regression (anchor -> proposal)
  - Fast R-CNN classification (over classes)
  - Fast R-CNN regression (proposal -> box)



- 250x times faster than R-CNN
- Same mAP as Fast R-CNN

Object detection state-of-the art

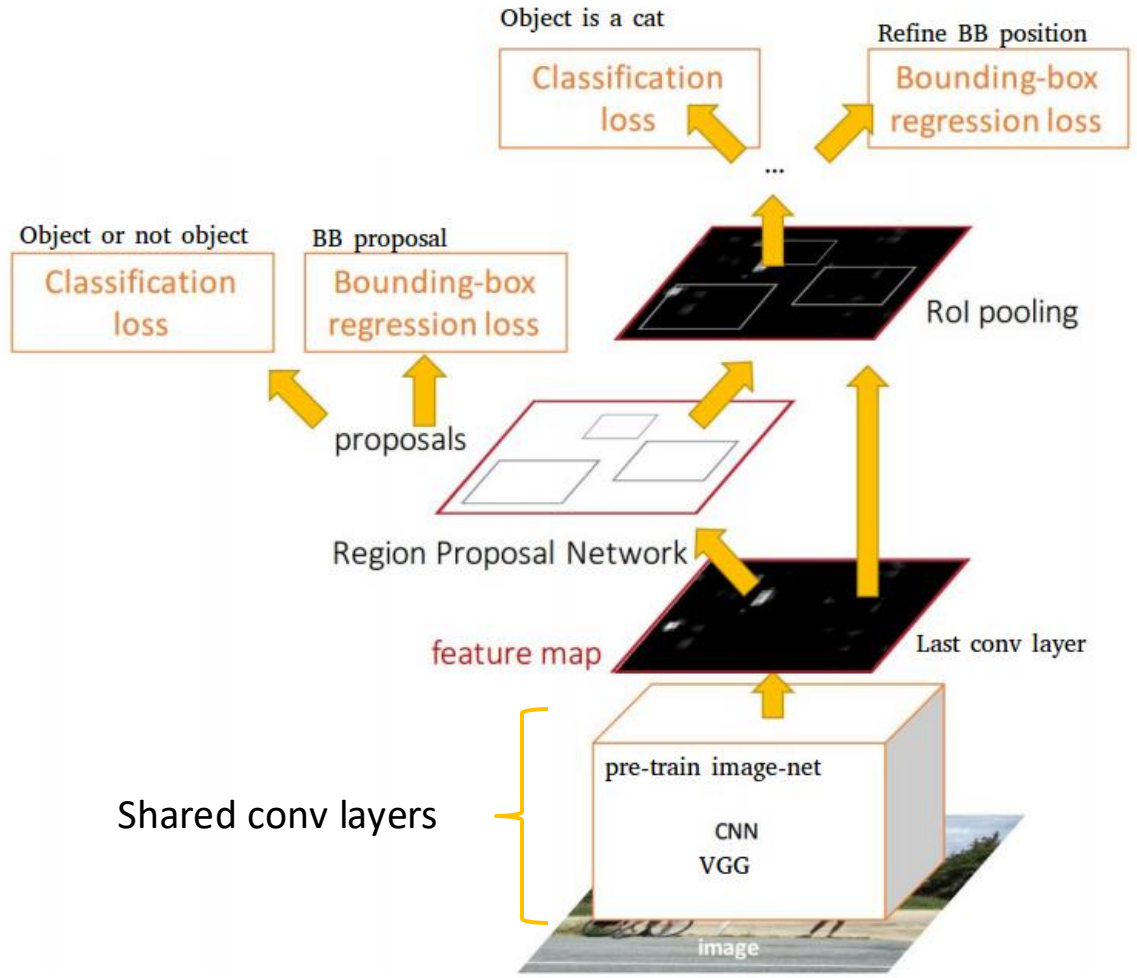
- ResNet 101 + Faster R-CNN + some extras

training data	COCO train		COCO trainval	
test data	COCO val		COCO test-dev	
mAP	@.5	@[.5, .95]	@.5	@[.5, .95]
baseline Faster R-CNN (VGG-16)	41.5	21.2		
baseline Faster R-CNN (ResNet-101)	48.4	27.2		
+box refinement	49.9	29.9		
+context	51.1	30.0	53.3	32.2
+multi-scale testing	53.8	32.5	**55.7**	**34.9**
ensemble			**59.0**	**37.4**

ResNet paper: https://arxiv.org/abs/1512.03385

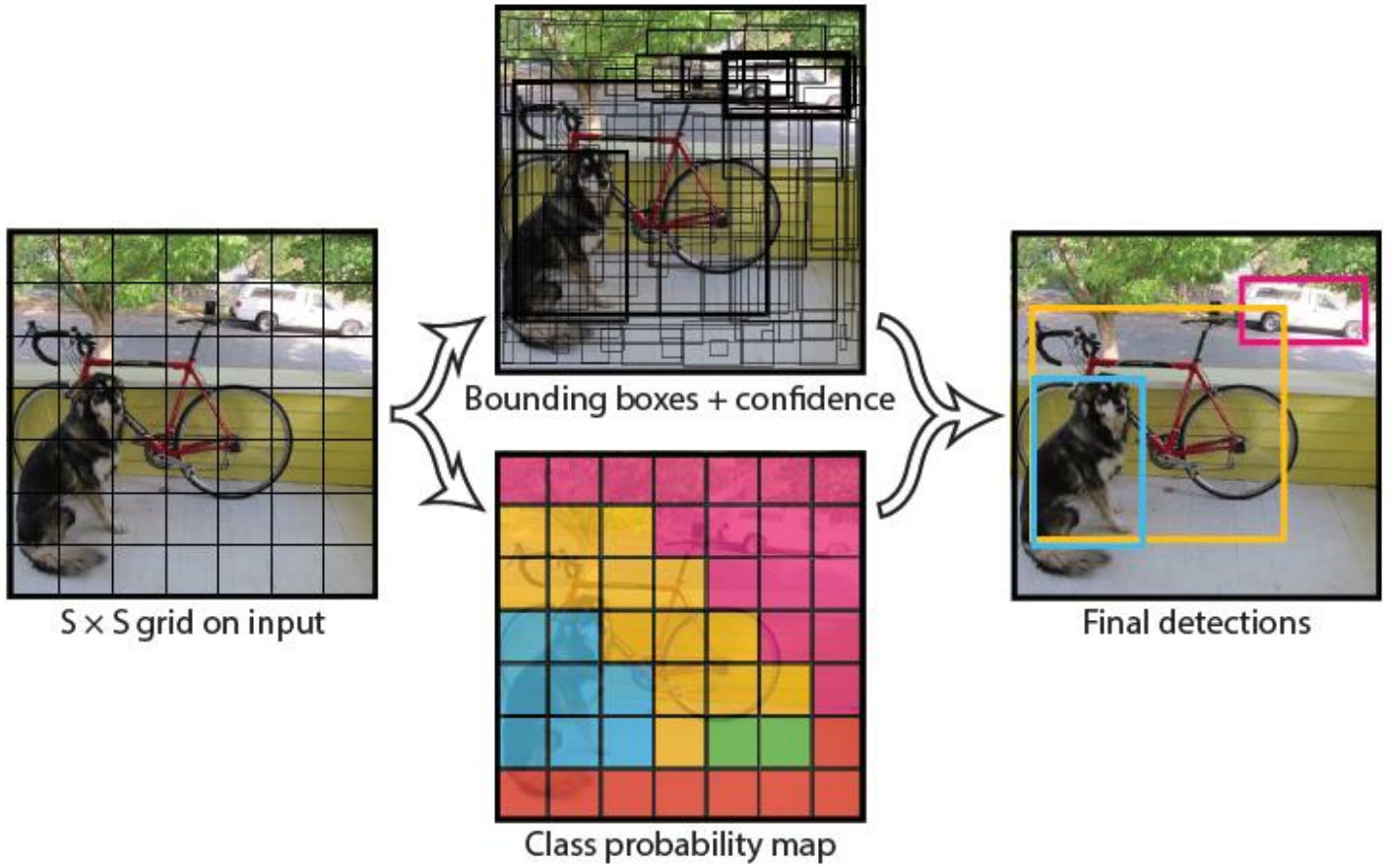
One-stage object detection models

- Faster R-CNN is a two-stage object detector
- First stage: Run once per image
  - Shared conv layer network
  - Region proposal network
- Second stage: Run once per region propsal
  - Crop features: RoI pool / align
  - Predict object class
  - Prediction bbox offset
- **Do we really need two stages?**



YOLO - You Only Look Once

- **Idea:** No region proposals
- A single regression problem, straight from image pixels to bounding box coordinates and class probabilities.
  - extremely fast
  - reason globally
  - learn generalizable representations
- Paper:
  - https://arxiv.org/abs/1506.02640
- Homepage:
  - https://pjreddie.com/darknet/yolo



YOLO details

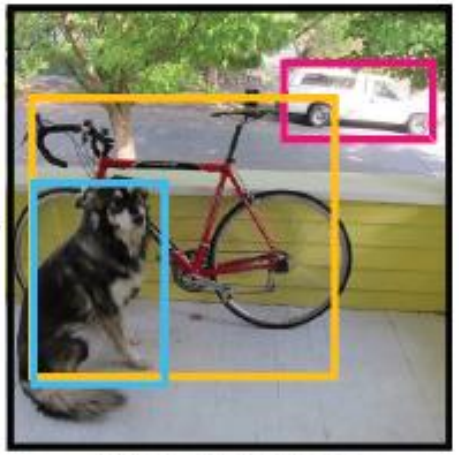
1. Divide the image into 7x7 cells.

S × S grid on input

Bounding boxes + confidence

Class probability map

2. For each cell, predict bbox coordinates and confidence scores (object or not object). Thickness of boxes = confidence.

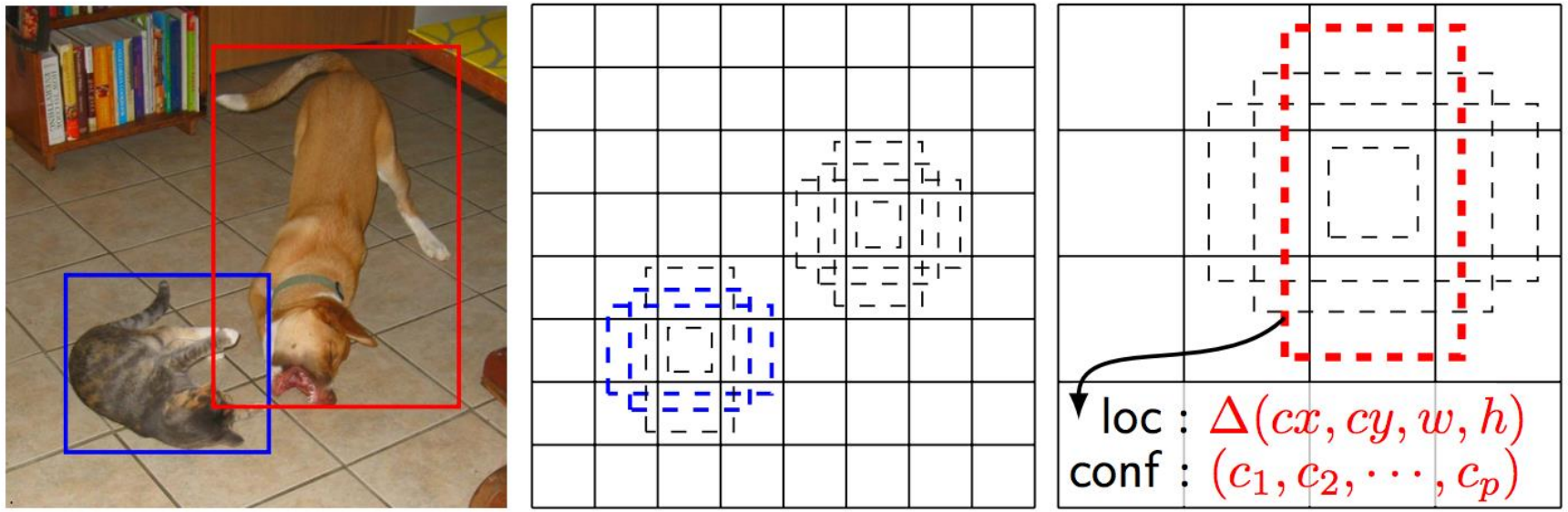


Final detections

3. For each cell, predict the class probabilities.

4. Final output – pick cells with confidence above threshold and non-max suppression to prevent overlapping bboxes

SSD: Single Shot Detector



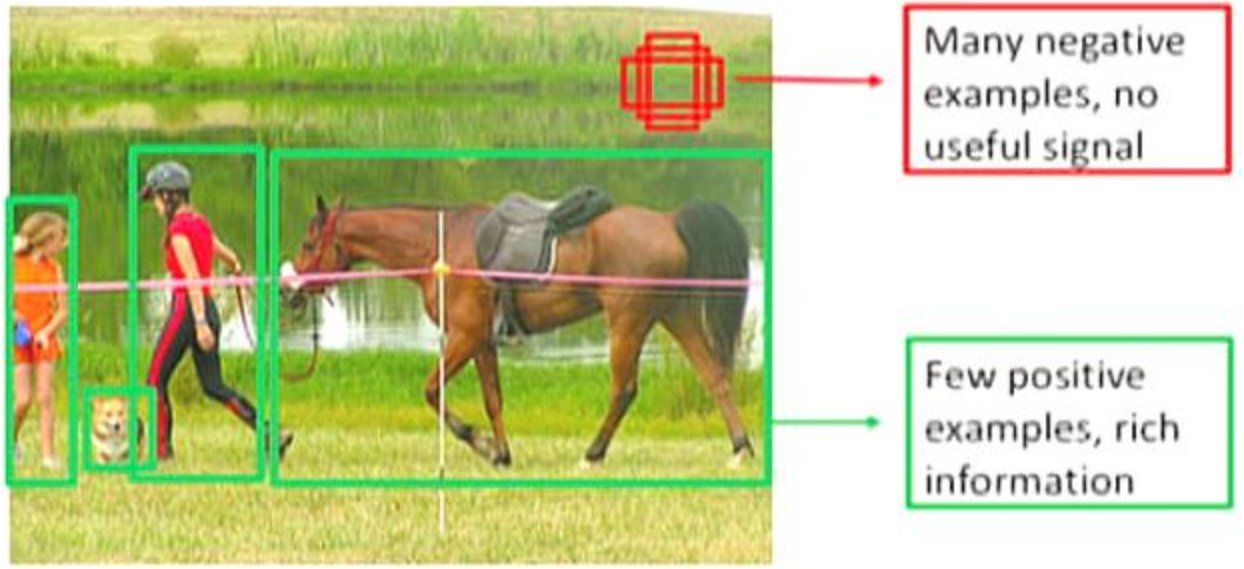
(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Idea: Similar to YOLO, but denser grid map, multiscale grid maps + data augmentation + hard negative mining + other design choices in the network.

SSD: https://arxiv.org/pdf/1512.02325.pdf

RetinaNet

- Extreme foreground-background class imbalance problem in one-stage detector training.
- RetinaNet uses a special **focal loss** to lower contribution from “easy” negative samples so that the loss is focusing on “hard” samples, which improves the prediction accuracy.
- Paper: https://arxiv.org/abs/1708.02002
- Review: https://towardsdatascience.com/review-retinanet-focal-loss-object-detection-38fba6afabe4



Object detection – many solutions

- Some recommended blogs posts (not mutually exclusive…):
  - https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
  - https://www.datacamp.com/community/tutorials/object-detection-guide
  - https://towardsdatascience.com/review-faster-r-cnn-object-detection-f5685cb30202
  - https://www.jeremyjordan.me/object-detection-one-stage/

Semantic segmentation

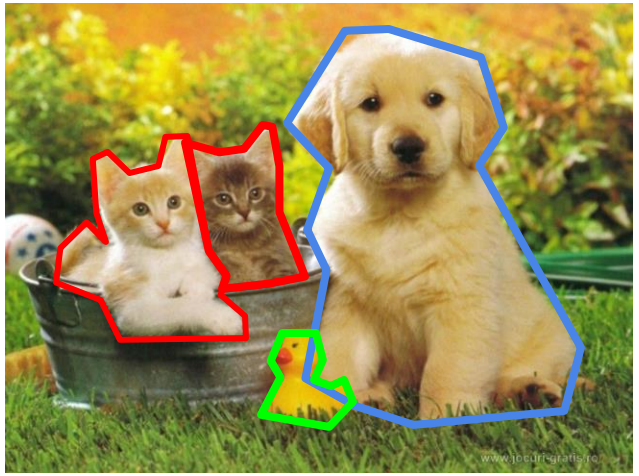
Computer vision tasks

Classification

Classification + Localization

Object Detection

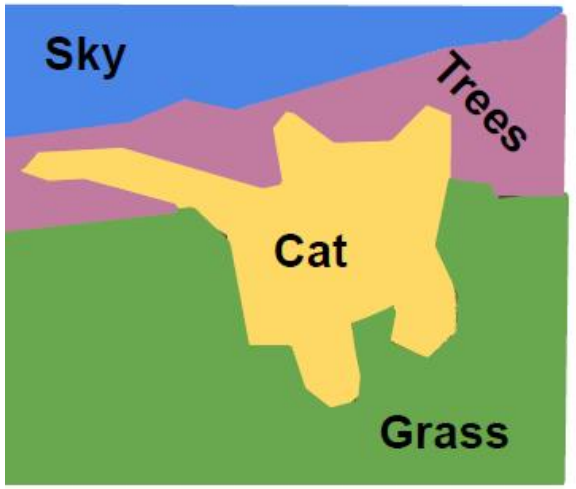
Image Segmentation

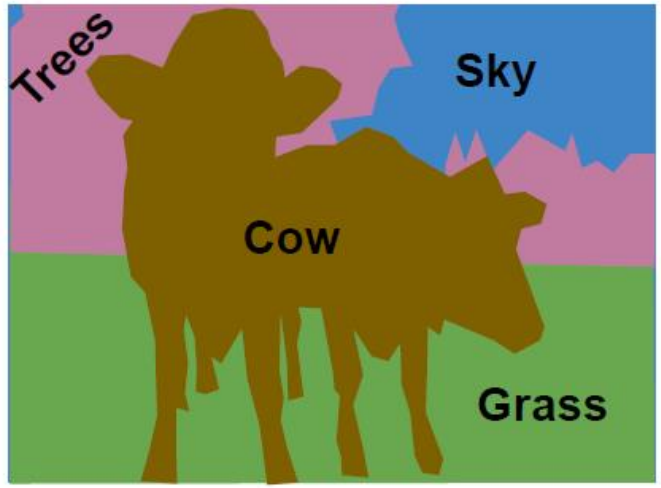


We will distinguish between **semantic segmentation** and **instance segmentation**. We will explain the difference later.

Semantic segmentation

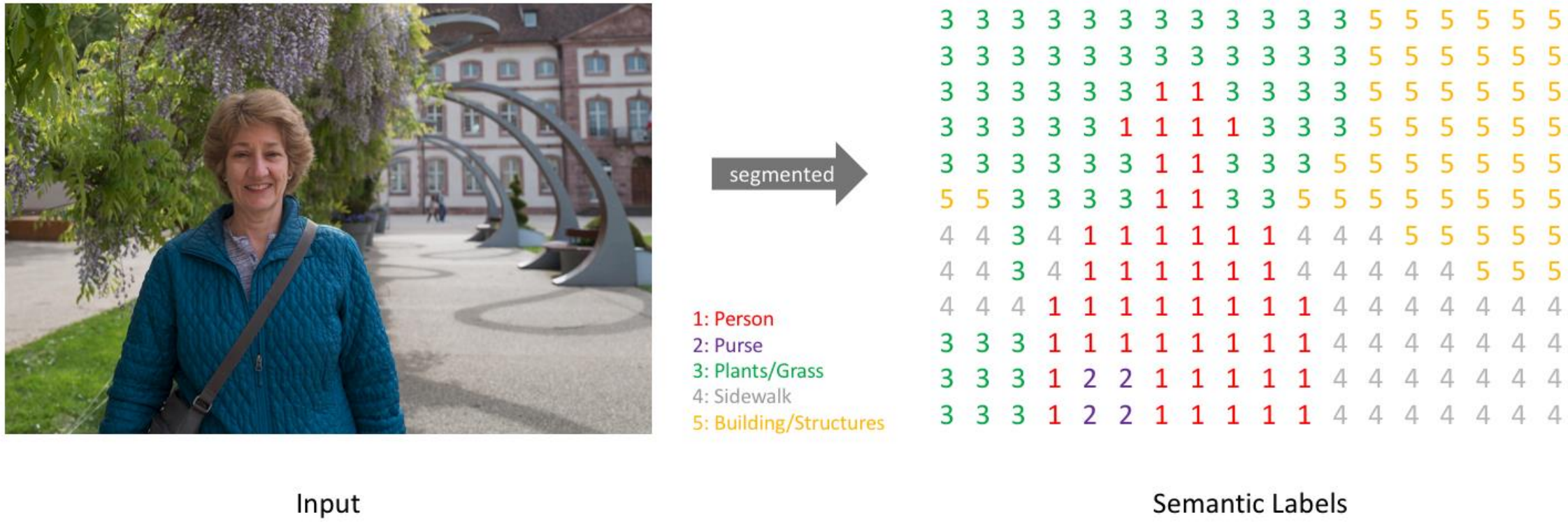
- **Label each pixel** in the image with a category label.
- Don’t differentiate object instances, only care about pixels
- For example, if there were 2 cows in an image, the semantic segmentation highlights the area they occupy, but it **will not be able to distinguish one cow from another**.





Representing the task

- Our goal is to take either a RGB color image (HxWx3) or a grayscale image (HxWx1 and output a segmentation map where each pixel contains a class label represented as an integer (HxW×1).



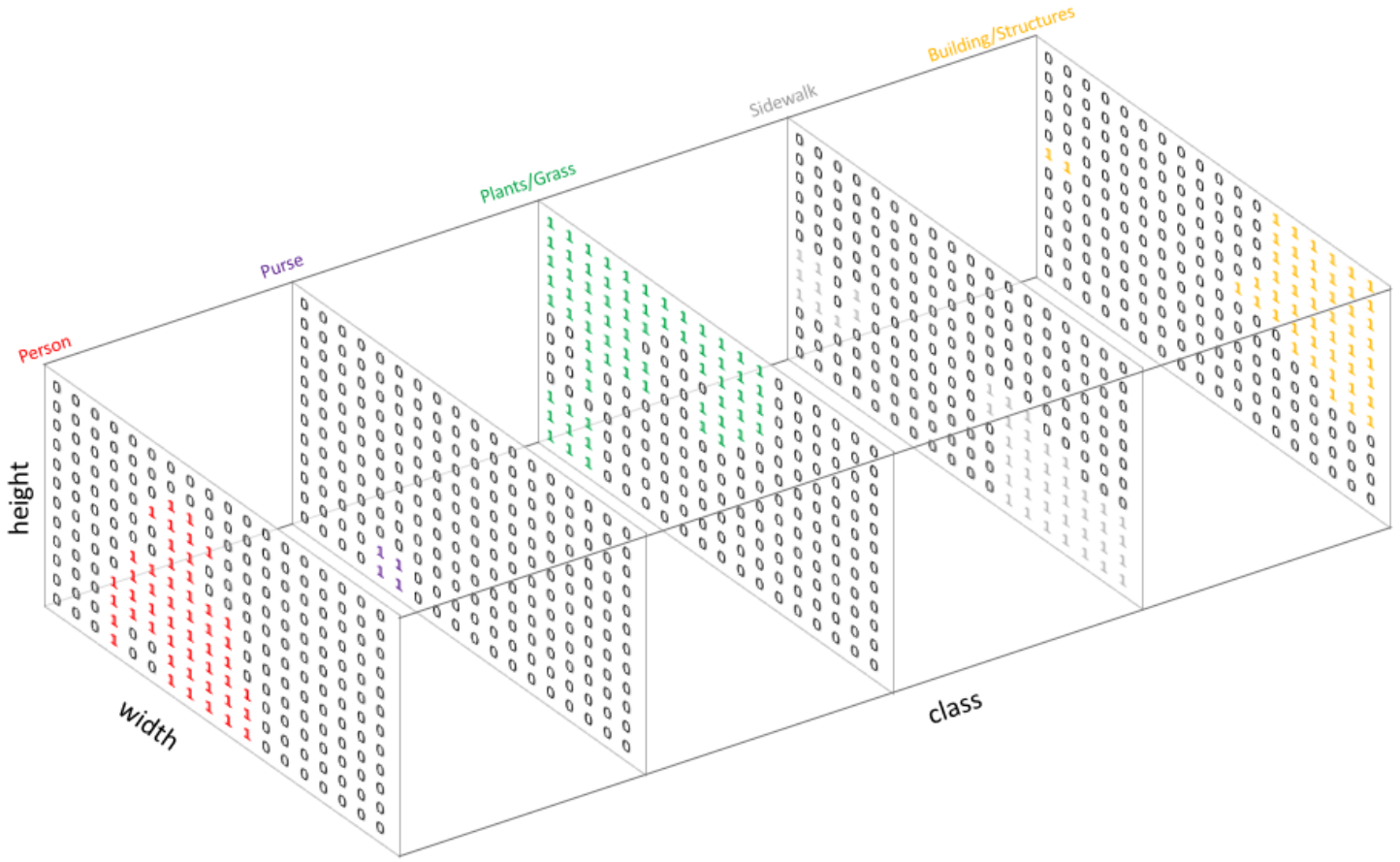
Input

Semantic Labels

https://www.jeremyjordan.me/semantic-segmentation/

Representing the task: One-hot encoding

- Similar to how we treat standard categorical values, we'll create our **target** by one-hot encoding the class labels - essentially creating an output channel for each of the possible classes.



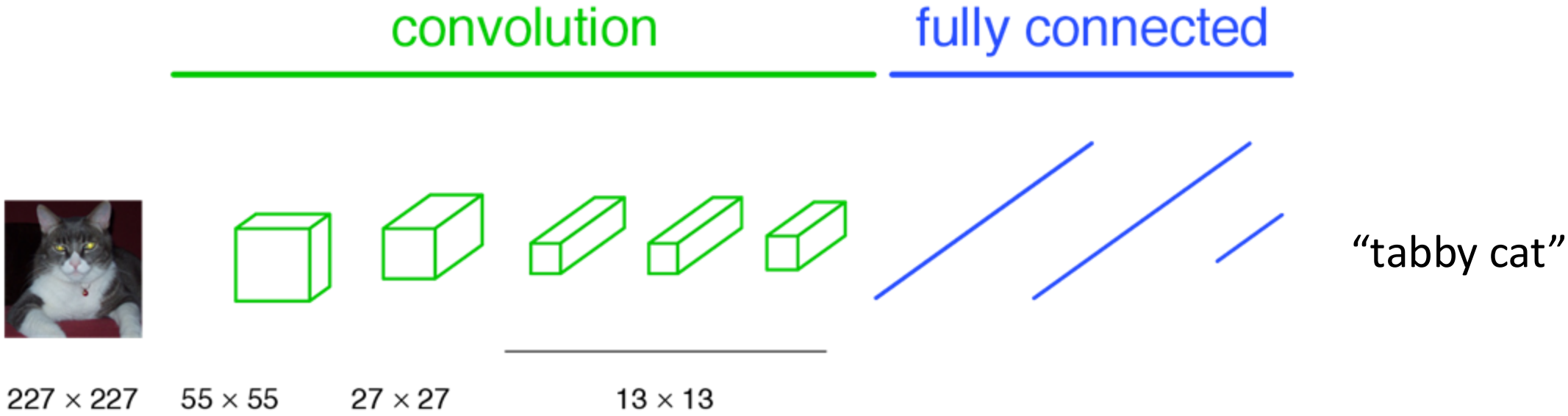
Representing the task: Depth-wise argmax

- Our model should predict a vector of class probabilities for each pixel (using softmax).
- The prediction can be collapsed into a segmentation map, like the one shown below, by taking the depth-wise **argmax** of each pixel vector.

0: Background/Unknown
1: Person
2: Purse
3: Plants/Grass
4: Sidewalk
5: Building/Structures

Idea: FCN

- Take a CNN classifier and make it fully convolutional

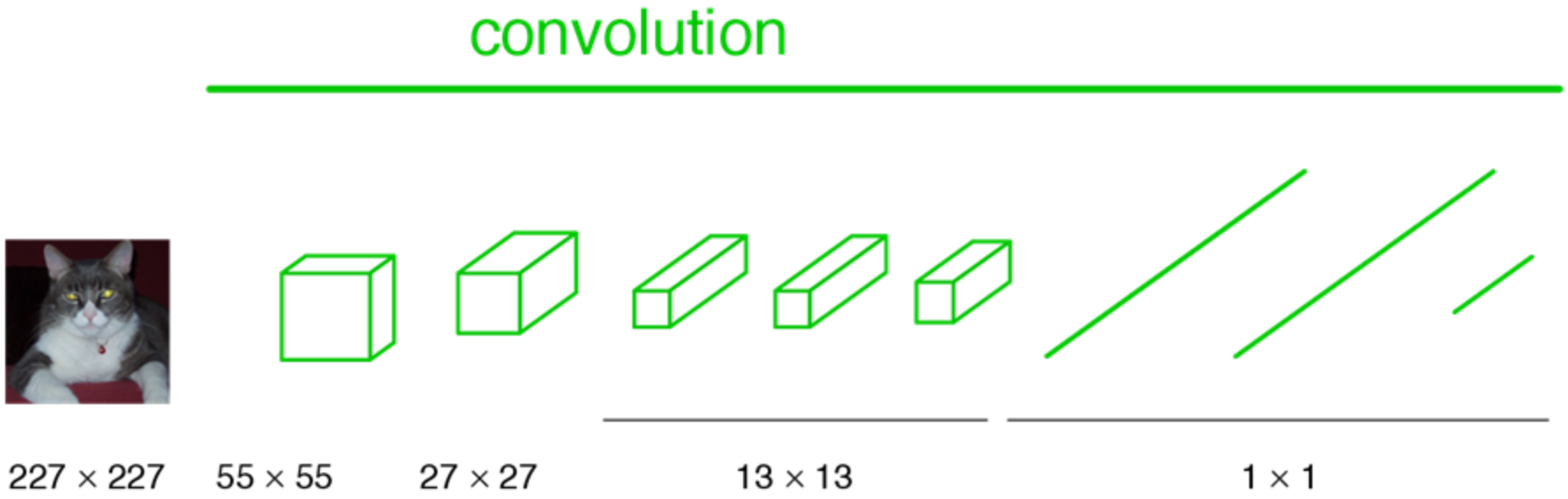


Conventional CNN classification network:

- Fixed size input (227x227x3): image
- Fixed size output (1x1000): class probabilities

Idea: FCN

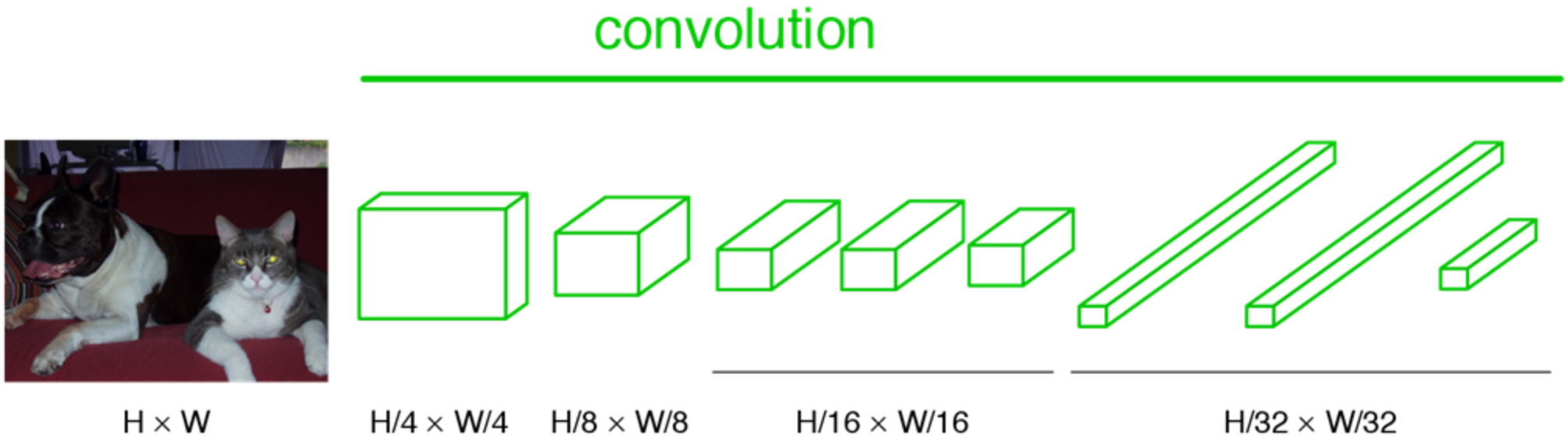
- Fully Convolutional Network (FCN): replace fully connected layers with 1x1xM convolutions.



Fully convolutional network (FCN):
- Variable size input shape
- Variable output shape

Idea: FCN

- Fully Convolutional Network (FCN): replace fully connected layers with 1x1xM convolutions.

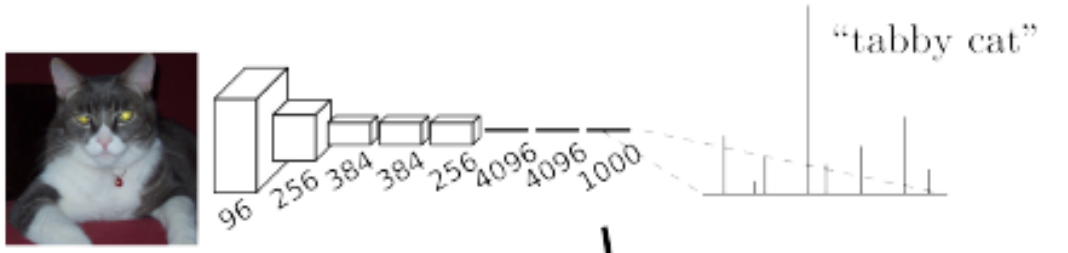


Example:

- Input shape: H x W x 3
- Output shape: H/32 x W/32 x 1000

Comparison

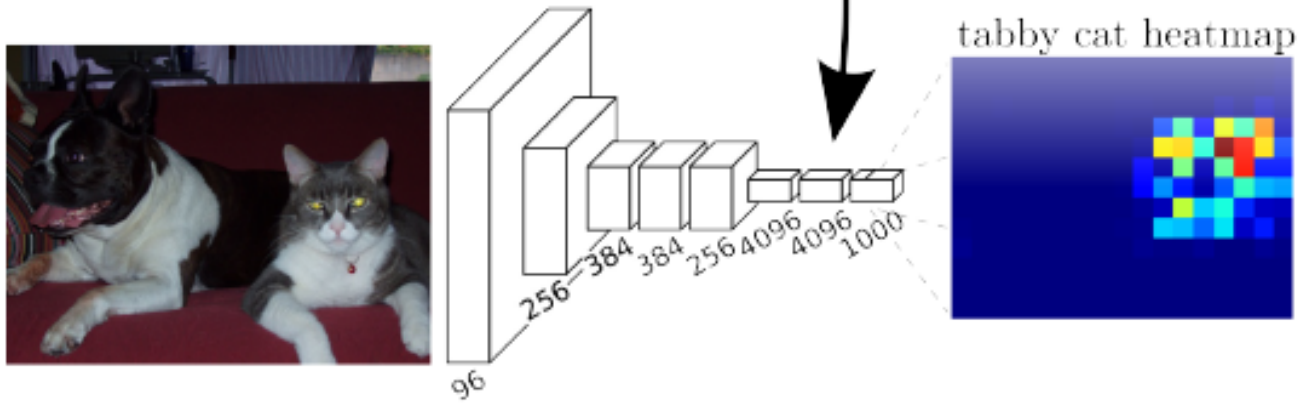
Standard CNN classifier:



Output is a vector of class probabilities
Shape: 1 x C

convolutionalization

FCN:

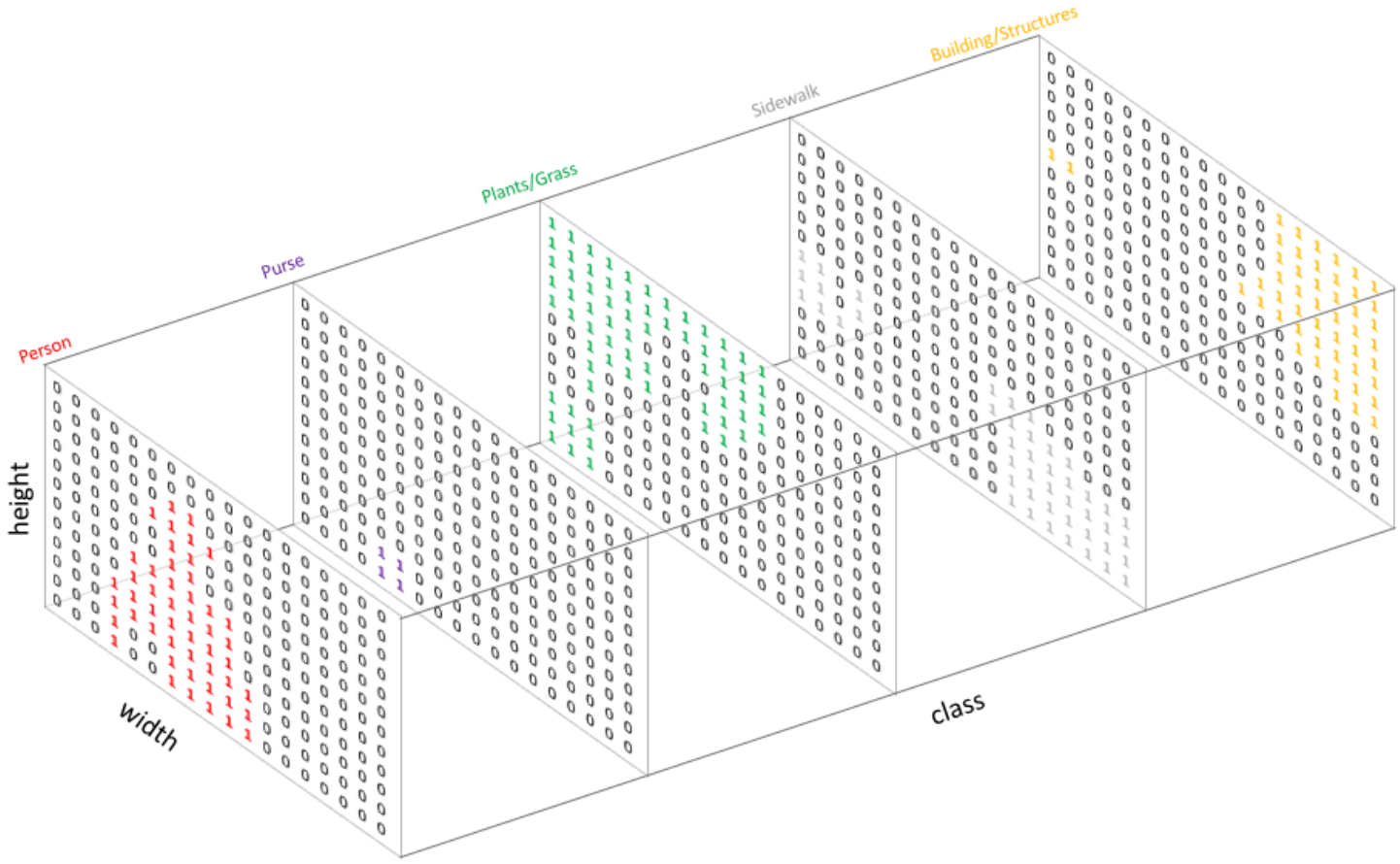


Output is a volume of heatmaps of class probabilities
Shape: H x W x C

Fig. 2. Transforming fully connected layers into convolution layers enables a classification net to output a spatial map. Adding differentiable interpolation layers and a spatial loss (as in Figure 1) produces an efficient machine for end-to-end pixelwise learning.

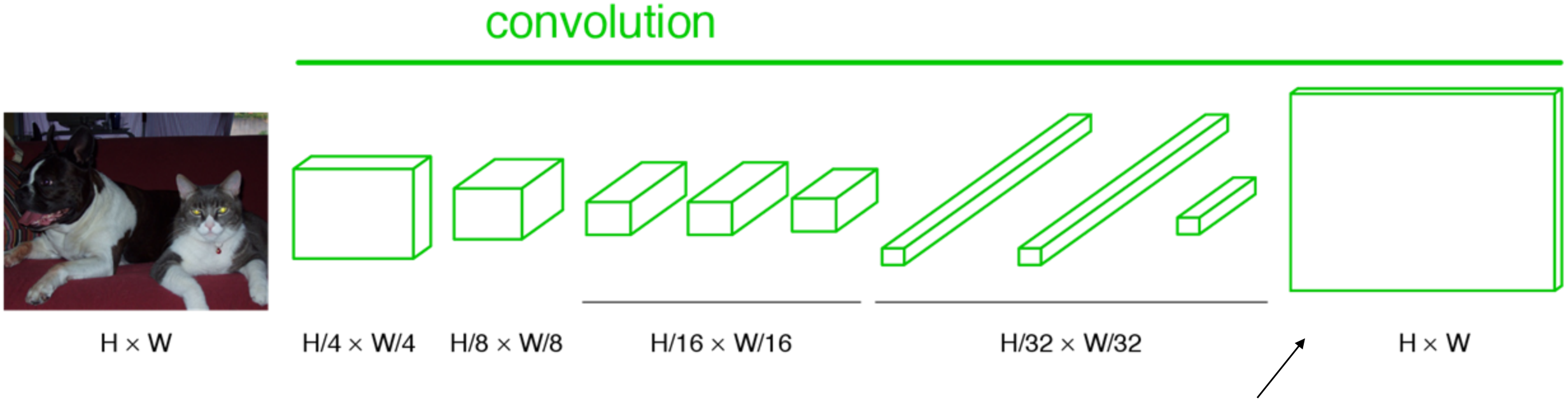
Recall: Class probability maps

- One output channel for each of the possible classes.



Problem with simple FCN

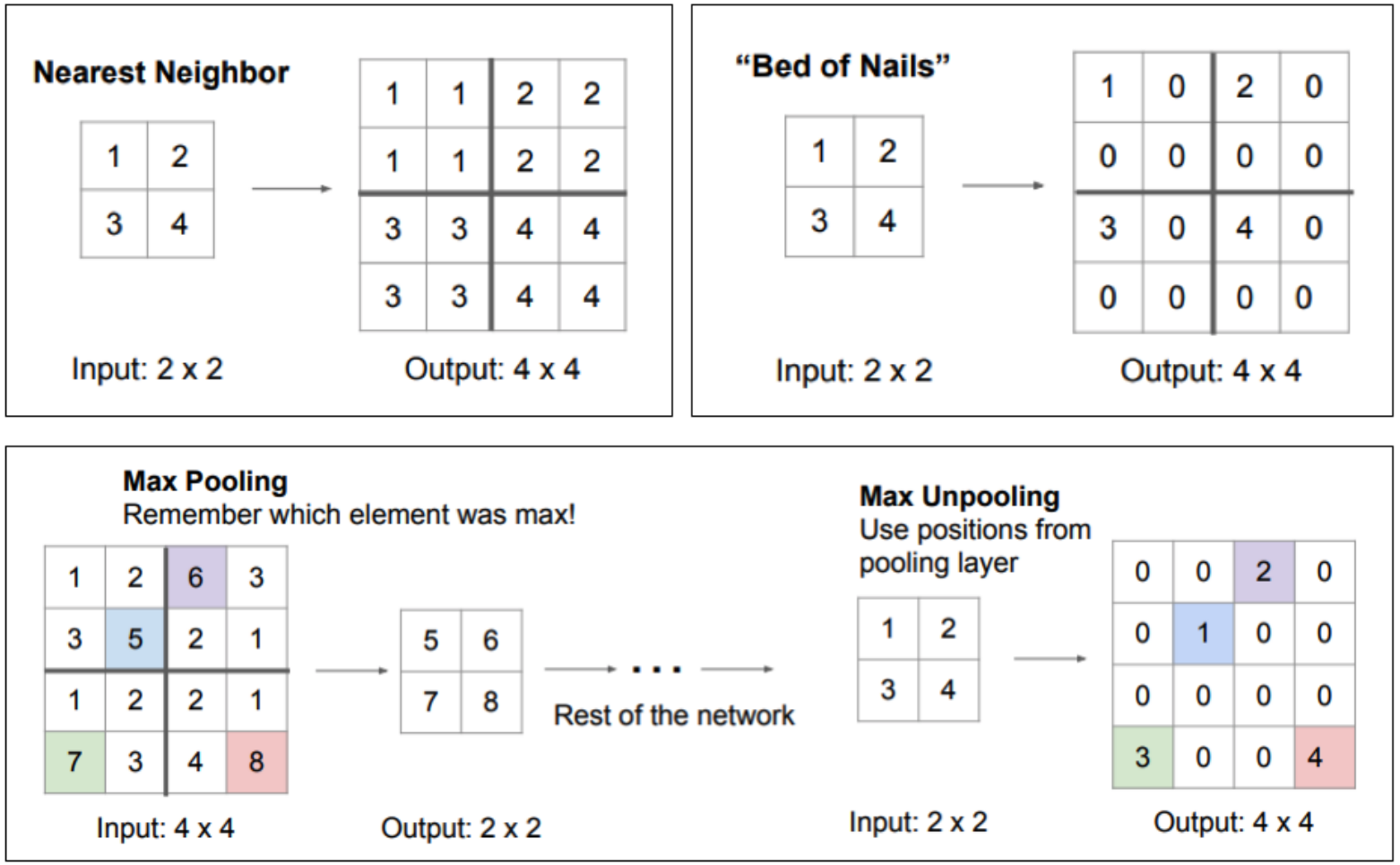
- The output is the same height/width as the final feature map (e.g., H/32 x W/32), but it should be same shape as input (H x W).



We need upsampling here:
Output should have same height and width as input

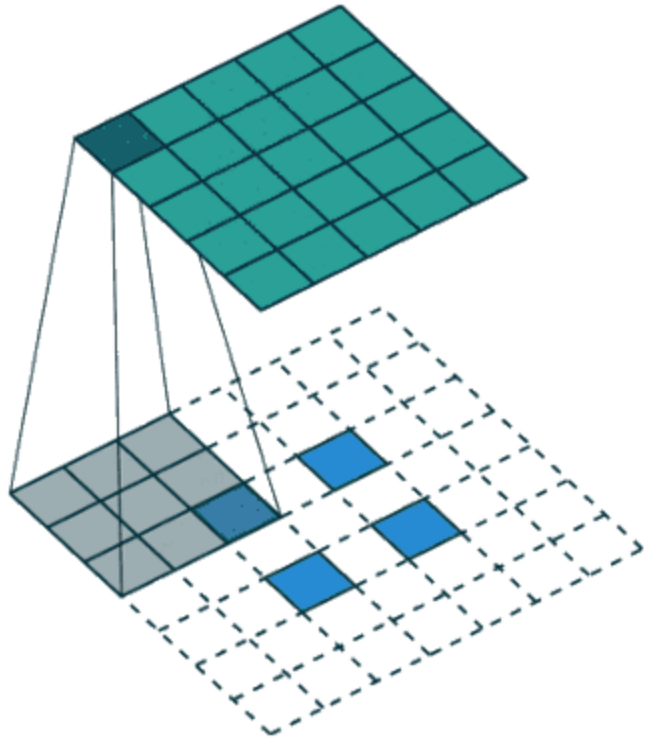
Methods for upsampling

- There are a few different approaches that we can use to *upsample* the resolution of a feature map.
- Whereas pooling operations downsample the resolution by summarizing a local area with a single value (ie. average or max pooling), "**unpooling**" operations upsample the resolution by distributing a single value into a higher resolution.



Learnable upsampling

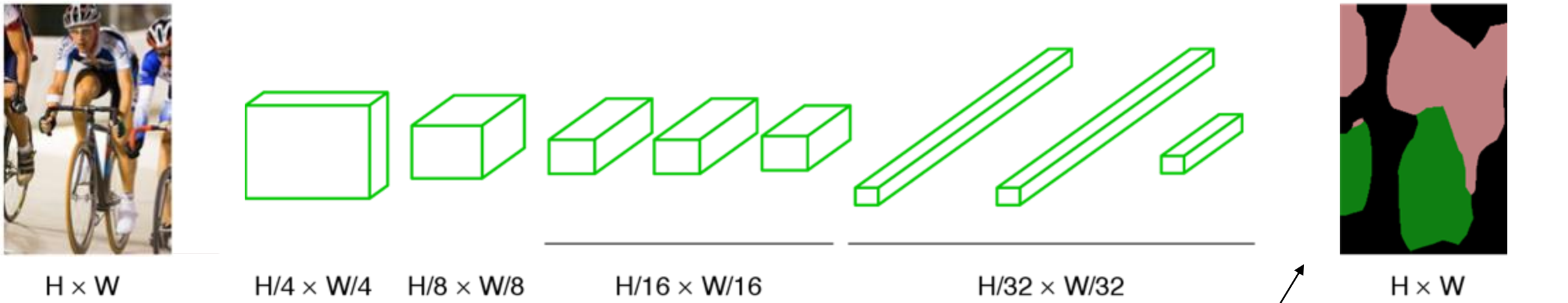
- **Transpose convolution** is by far the most popular approach as it allows for us to develop a **learned upsampling**.
- Standard convolution takes the dot product of the values currently in the filter's view and produce a single output.
- Transpose convolution instead takes a single value from the low-resolution feature map and multiply all of the weights in our filter by this value, projecting those weighted values into the output feature map.
- Aside: For filter sizes which produce an overlap in the output feature map (e.g. 3x3 filter with stride 2 - as shown in the example), the overlapping values are simply added together. Unfortunately, this tends to produce a **checkerboard artifact** in the output and is undesirable, so it's best to ensure that your filter size does not produce an overlap.



Up-sampling a 2 x 2 input to a 5 x 5 output.

32x upsampling with transpose conv.

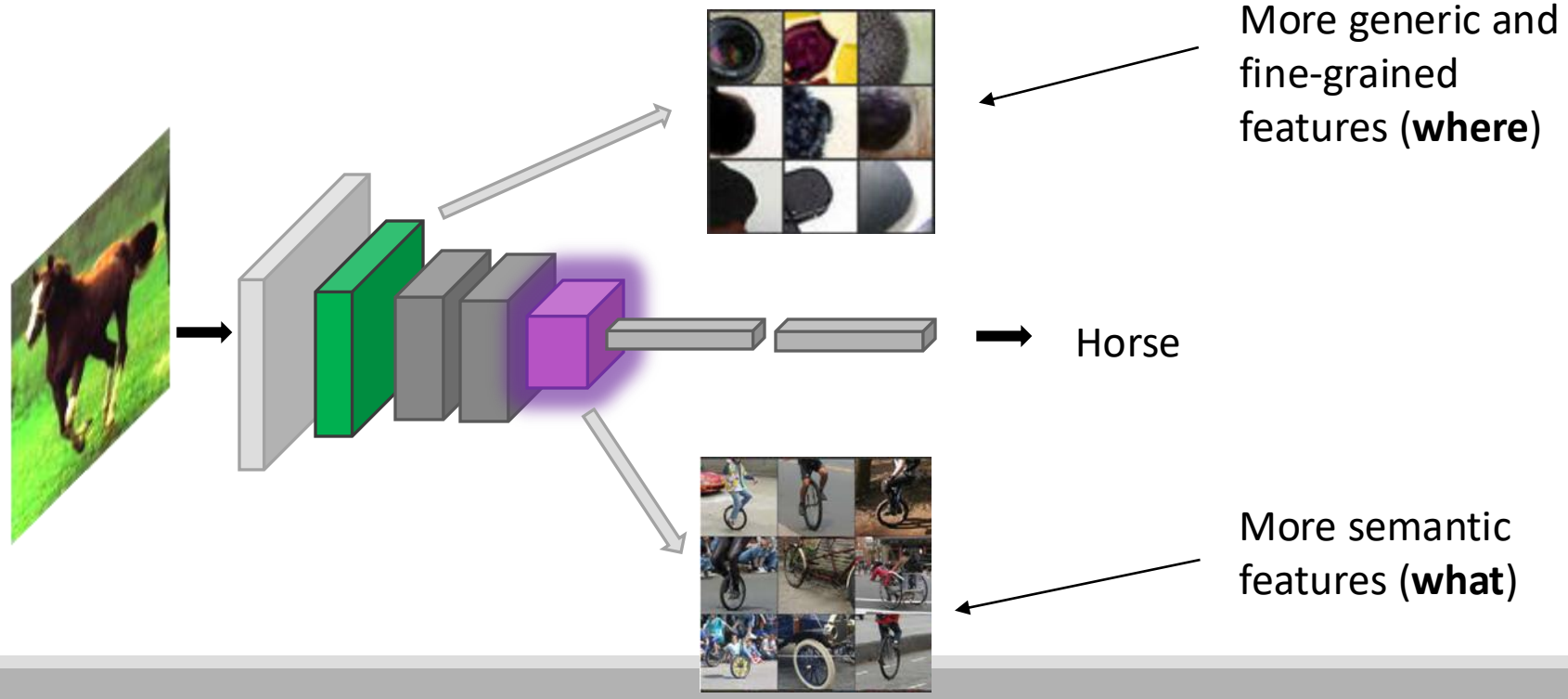
Ground truth for comparison



Because the encoder module reduces the resolution of the input by a factor of 32, the upsampling module (decoder) **struggles to produce fine-grained segmentations**. (Note that this image shows the depth-wise argmax)

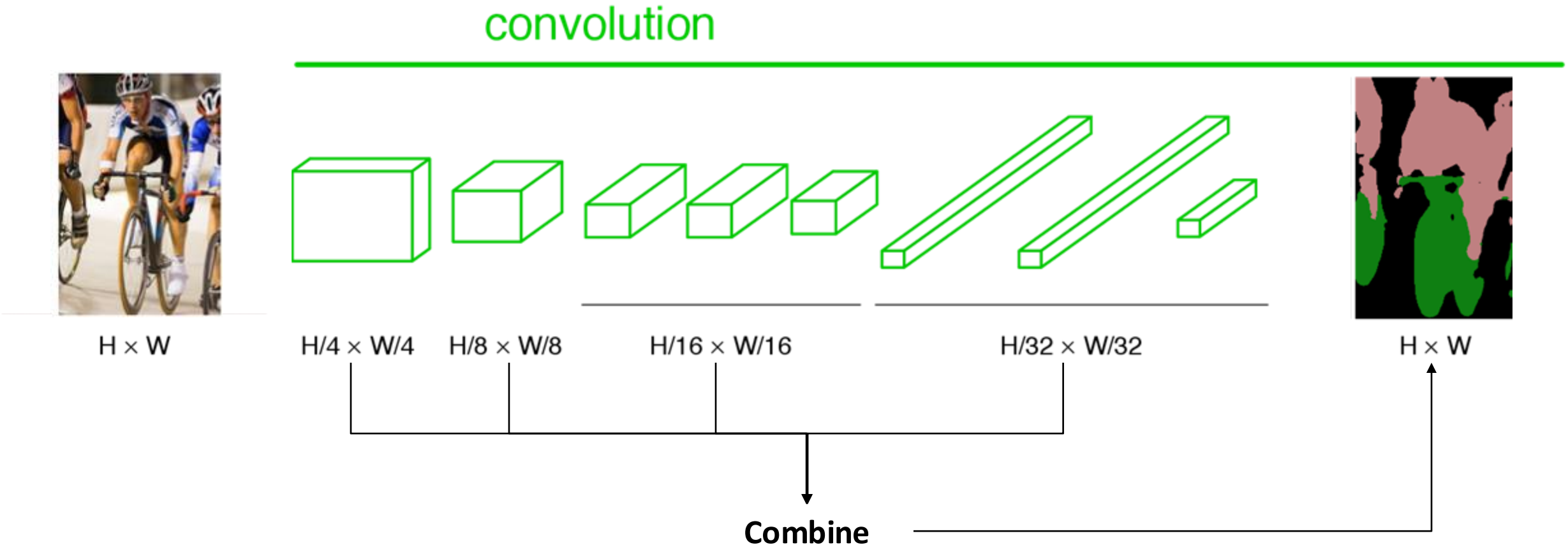
Problem: Need fine details!

- Semantic segmentation faces an inherent tension between semantics and location: global information from late layers resolves **what**, while local information from early layers resolves **where**…
- Combining fine layers and coarse layers lets the model make local predictions that respect global structure.

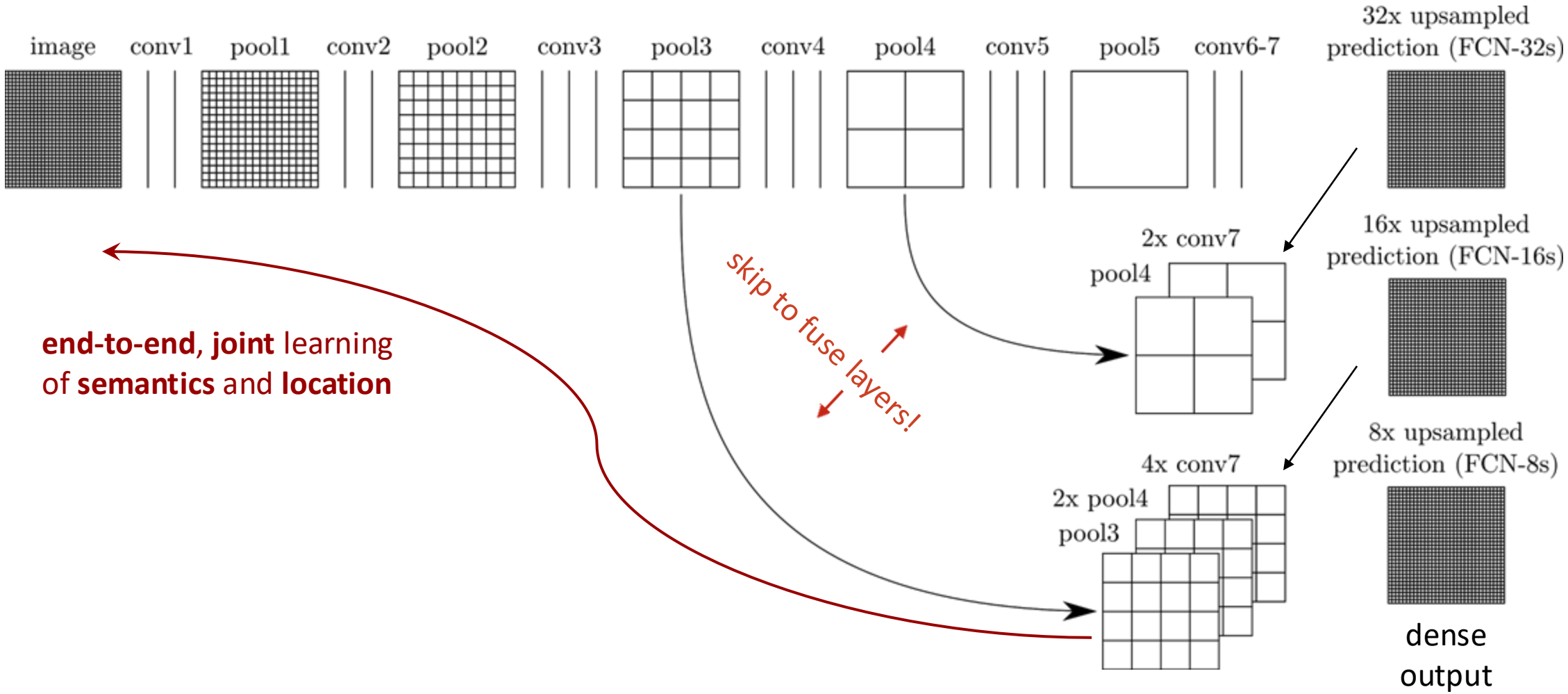


Skip layer refinement

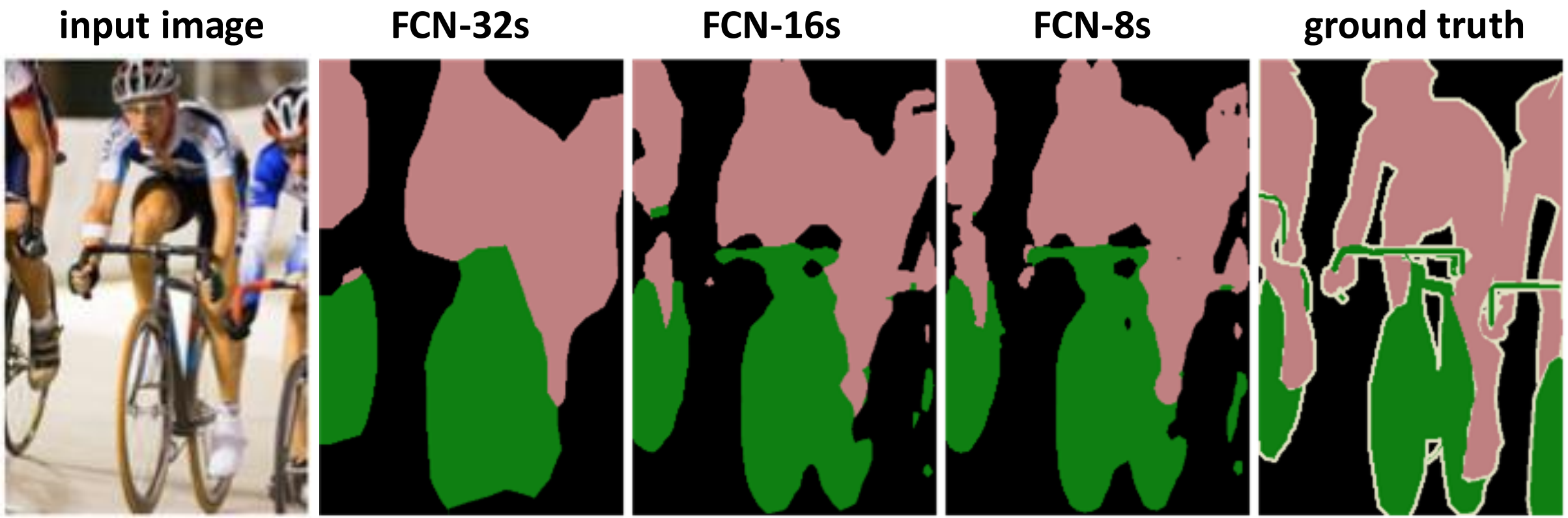
- Jointly learn to combine outputs from multiple layers.



Skip layer refinement

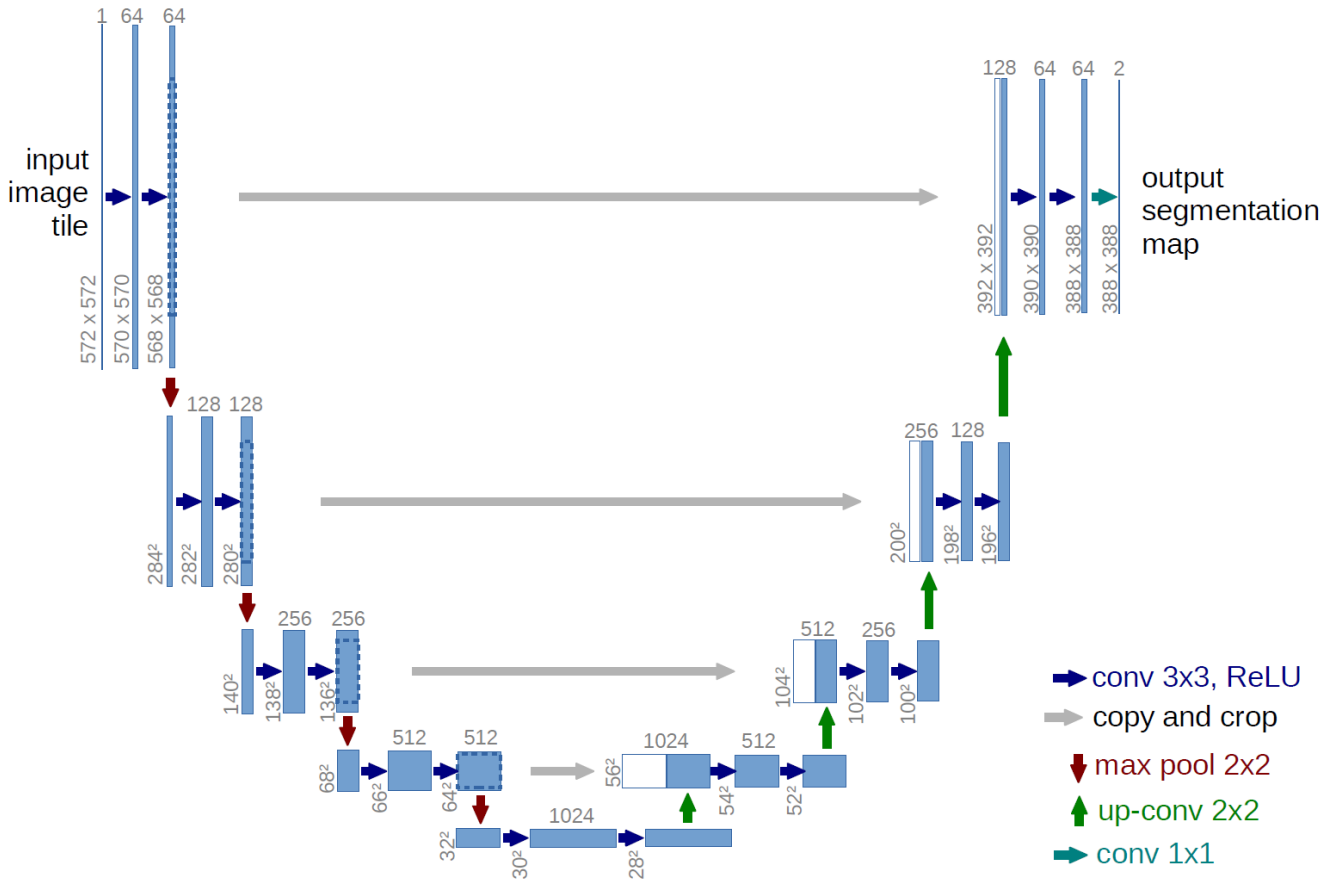


Skip layer refinement



U-Net: Further improvement

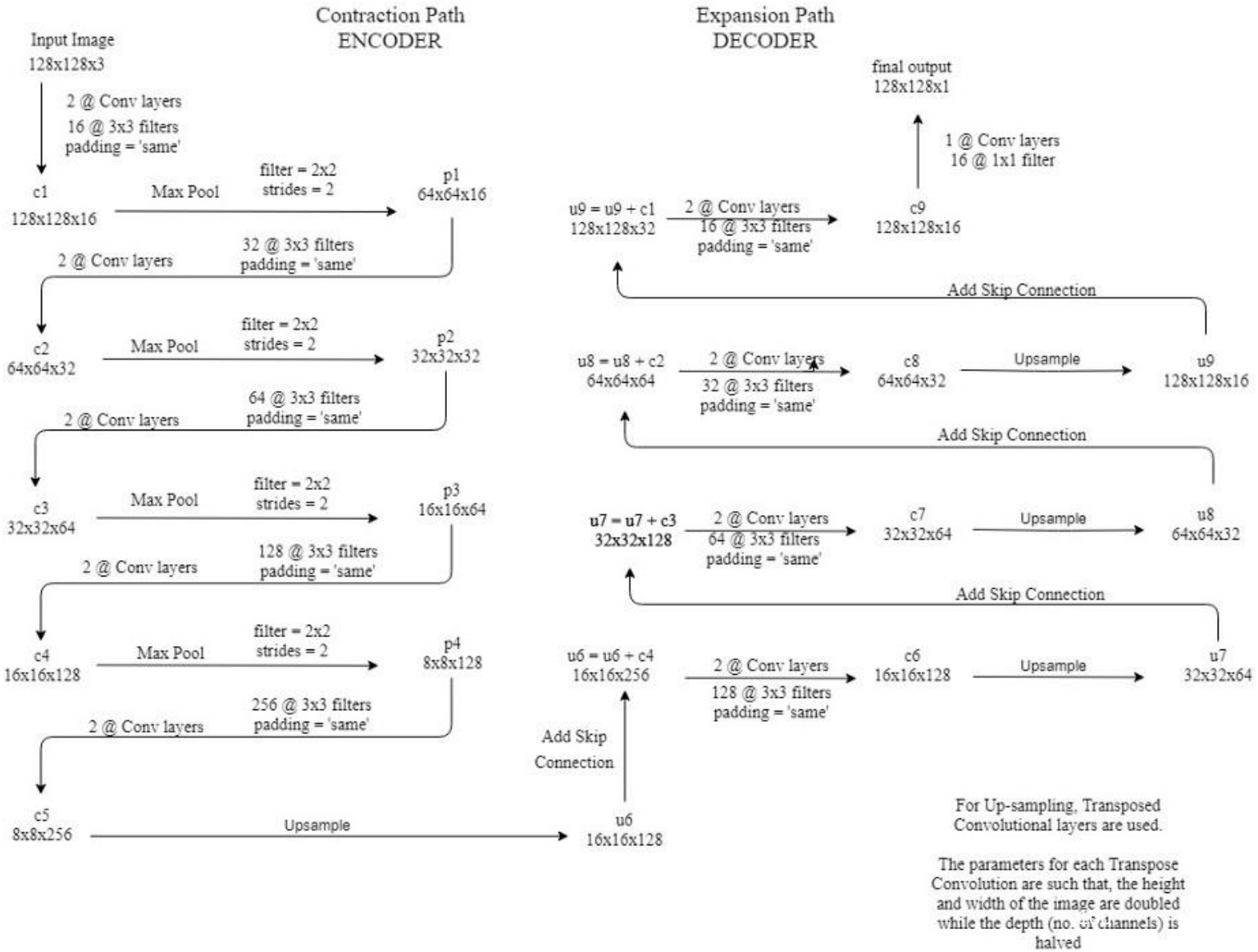
- FCN has complicated, multi-stage training pipeline (not end-to-end).
- U-Net improves upon the FCN architecture primarily through **expanding the capacity of the decoder** module of the network.
- The U-Net architecture "consists of a contracting path to capture context and a **symmetric** expanding path that enables precise localization."
- This simpler architecture has grown to be very popular and has been adapted for a variety of segmentation problems.



Paper: https://arxiv.org/abs/1505.04597

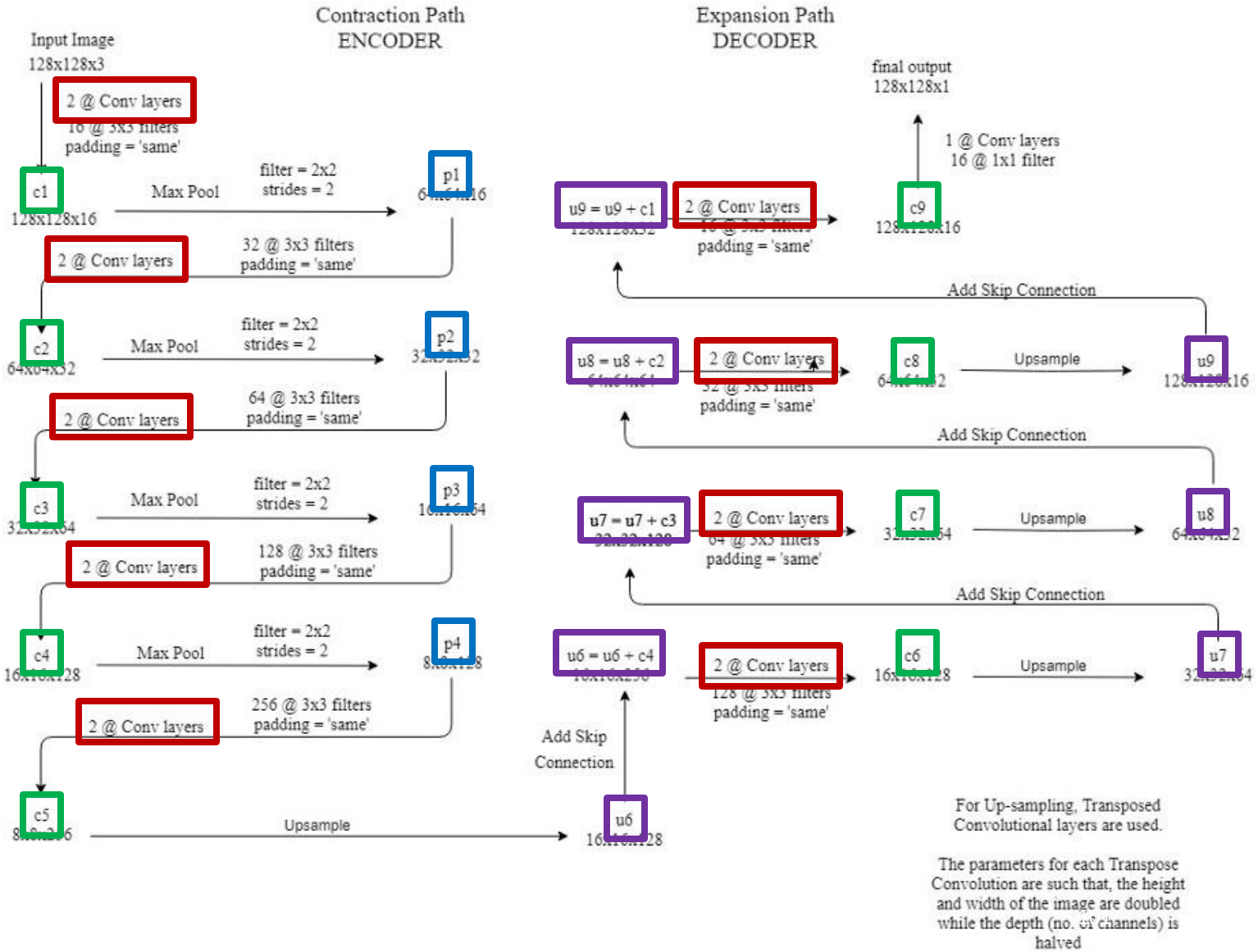
U-Net architecture

- Note that in the original paper, the size of the input image is 572x572x3.
- However, we will use input image of size 128x128x3.
- Hence the size at various locations will differ from that in the original paper but the core components remain the same.
- Also, the original architecture uses ‘valid’ padding. Here, we use ‘same’ padding to preserve height and width after each convolution.



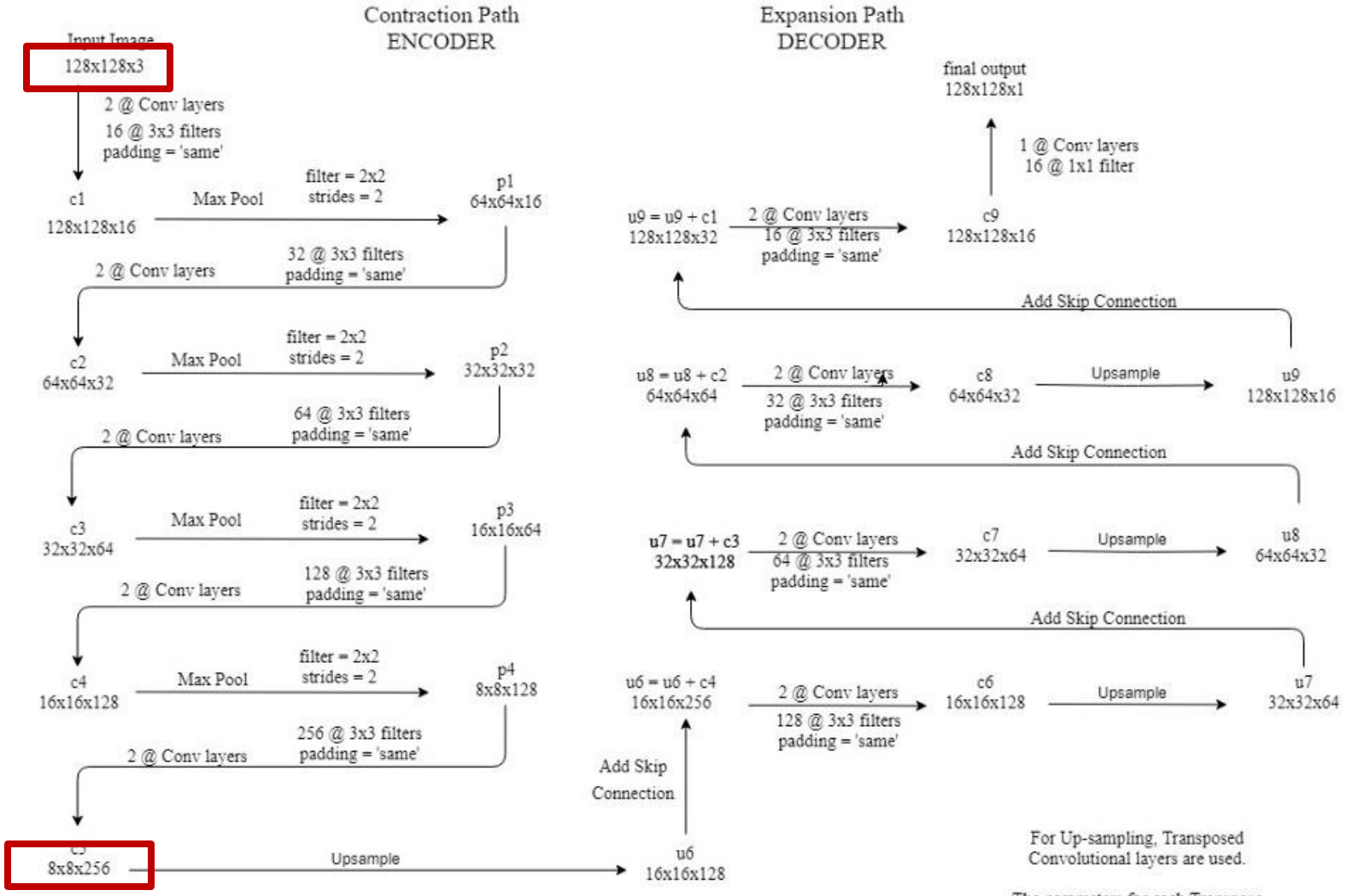
U-Net architecture

- **2@Conv** layers means that two consecutive Convolution Layers are applied
- **c1, c2, …. c9** are the output tensors of Convolutional Layers
- **p1, p2, p3 and p4** are the output tensors of Max Pooling Layers
- **u6, u7, u8 and u9** are the output tensors of up-sampling (transposed convolutional) layers.



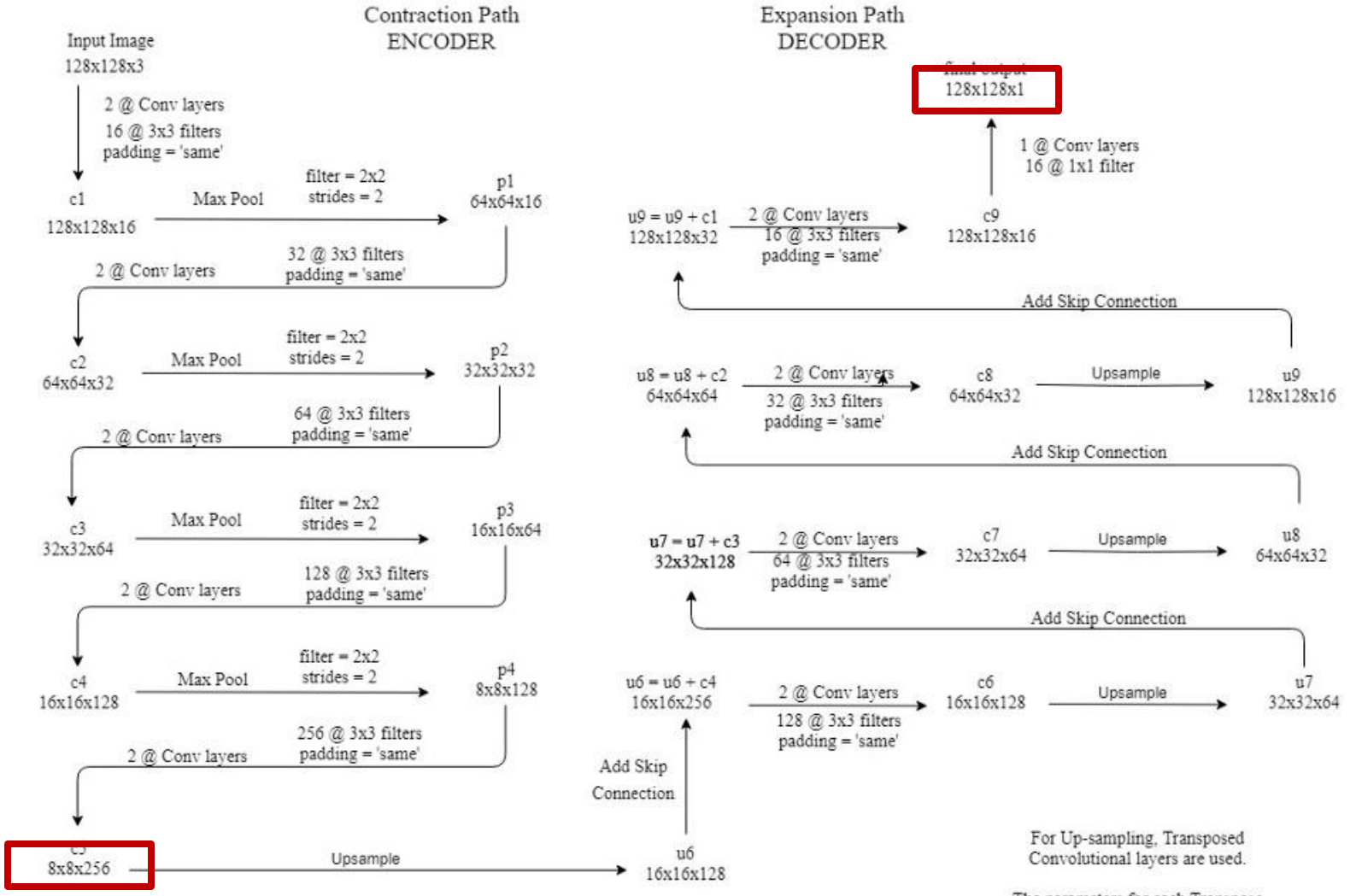
U-Net architecture

- The left-hand side is the contraction path (**Encoder**) where we apply regular convolutions and max pooling layers.
- In the Encoder, the size of the image gradually reduces while the depth gradually increases. **Starting from 128x128x3 to 8x8x256**.
- This basically means the network learns the "**WHAT**" information in the image, however it has lost the "**WHERE**" information.



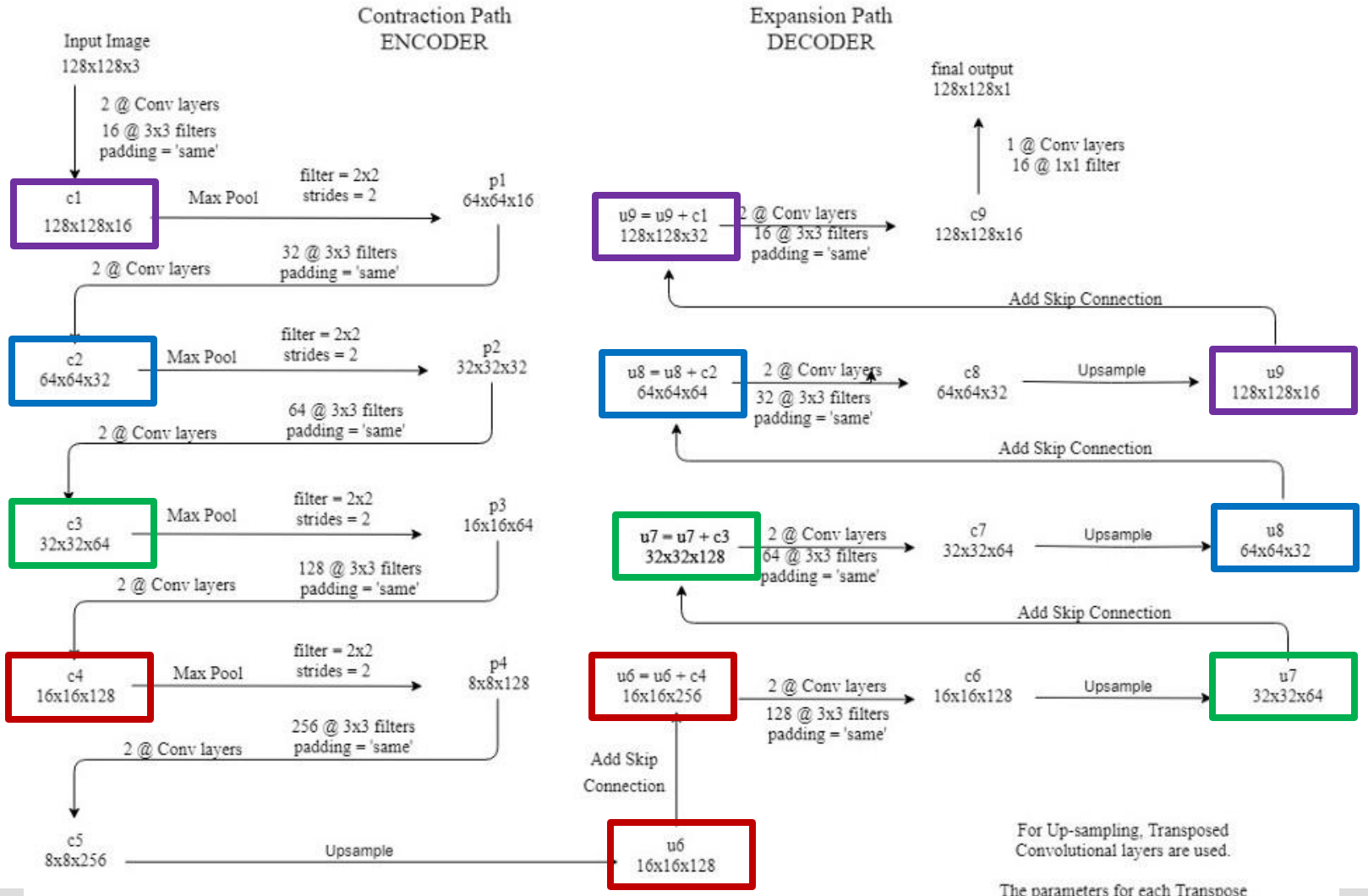
U-Net architecture

- The right-hand side is the expansion path (**Decoder**) where we apply transposed convolutions to upsample.
- In the decoder, the size of the image gradually increases, and the depth gradually decreases. **Starting from 8x8x256 to 128x128x1**.
- Intuitively, the Decoder **recovers the "WHERE"** information (precise localization) by gradually applying upsampling.



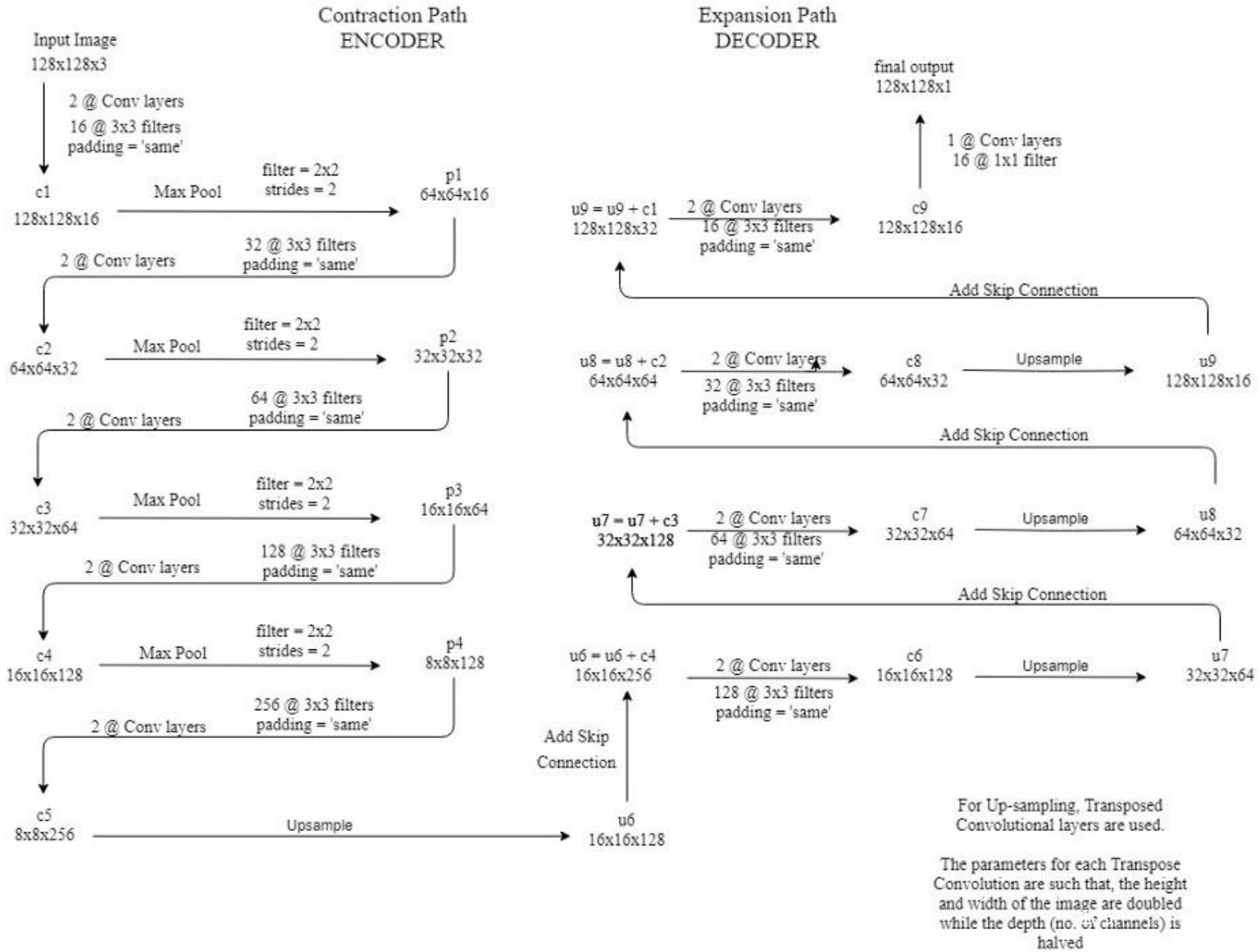
U-Net architecture

- To better recover the "WHERE" information, we use skip connections that concatenate the output of the transposed convolution layers with the feature maps from the Encoder at the same level:
  - **u6 = u6 + c4**
    u7 = u7 + c3
    u8 = u8 + c2
    u9 = u9 + c1
- After every concatenation we again apply two consecutive regular convolutions so that the model can learn to assemble a more precise output.



U-Net architecture

- This is what gives the architecture a symmetric U-shape, hence the name U-Net.
- On a high level, we have the following relationship:
  - Input (128x128x1) =>
  - Encoder => (8x8x256) =>
  - Decoder => Output (128x128x1)



U-Net

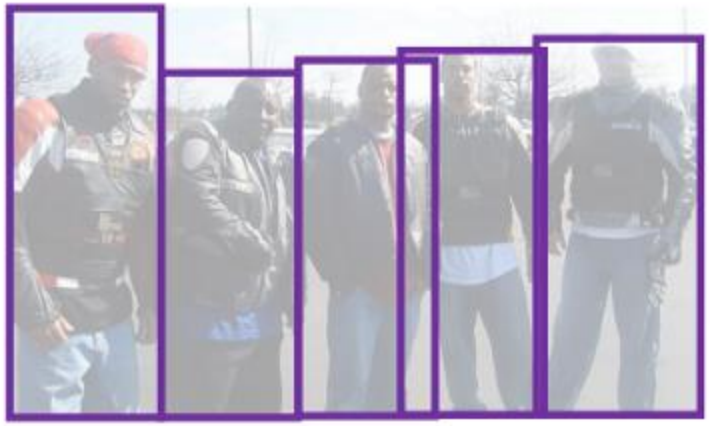
- Slightly simplified Keras code
- See original source here: https://gist.github.com/hlamba28/6073e8ef011a1d47080f67522018d7e4#file-tgf_unet_model-py

```
# Contracting Path (Encoder)
c1 = conv2d_block(input_img, n_filters * 1, kernel_size = 3, batchnorm = batchnorm)
p1 = MaxPooling2D((2, 2))(c1)
c2 = conv2d_block(p1, n_filters * 2, kernel_size = 3, batchnorm = batchnorm)
p2 = MaxPooling2D((2, 2))(c2)
c3 = conv2d_block(p2, n_filters * 4, kernel_size = 3, batchnorm = batchnorm)
p3 = MaxPooling2D((2, 2))(c3)
c4 = conv2d_block(p3, n_filters * 8, kernel_size = 3, batchnorm = batchnorm)
p4 = MaxPooling2D((2, 2))(c4)
c5 = conv2d_block(p4, n_filters = n_filters * 16, kernel_size = 3, batchnorm = batchnorm)
# Expansive Path (Decoder)
u6 = Conv2DTranspose(n_filters * 8, (3, 3), strides = (2, 2), padding = 'same')(c5)
→ u6 = concatenate([u6, c4])
c6 = conv2d_block(u6, n_filters * 8, kernel_size = 3, batchnorm = batchnorm)
u7 = Conv2DTranspose(n_filters * 4, (3, 3), strides = (2, 2), padding = 'same')(c6)
→ u7 = concatenate([u7, c3])
c7 = conv2d_block(u7, n_filters * 4, kernel_size = 3, batchnorm = batchnorm)
u8 = Conv2DTranspose(n_filters * 2, (3, 3), strides = (2, 2), padding = 'same')(c7)
→ u8 = concatenate([u8, c2])
c8 = conv2d_block(u8, n_filters * 2, kernel_size = 3, batchnorm = batchnorm)
u9 = Conv2DTranspose(n_filters * 1, (3, 3), strides = (2, 2), padding = 'same')(c8)
→ u9 = concatenate([u9, c1])
c9 = conv2d_block(u9, n_filters * 1, kernel_size = 3, batchnorm = batchnorm)
outputs = Conv2D(1, (1, 1), activation='sigmoid')(c9)
```

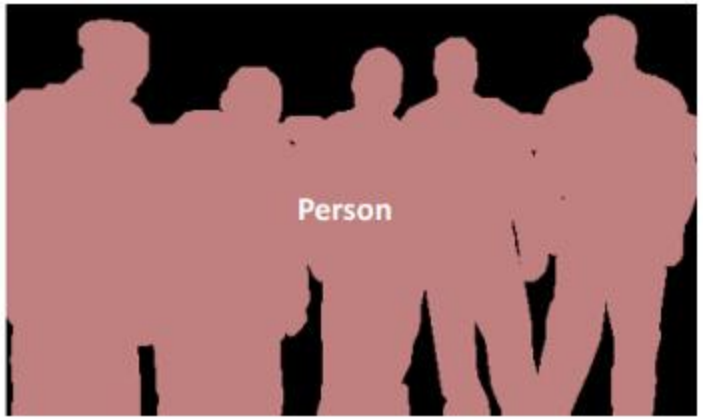
https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47

Instance segmentation

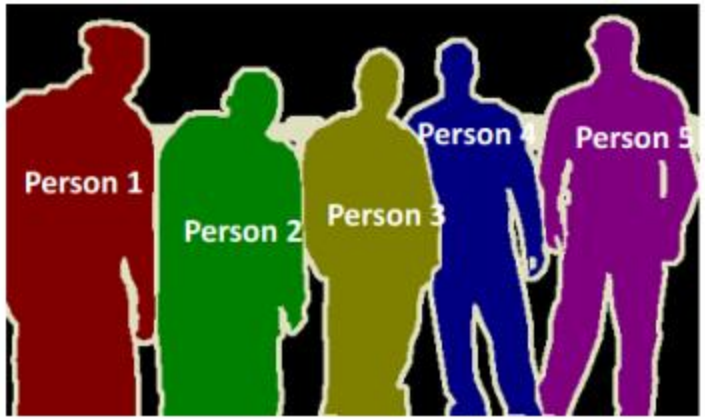
- **Problem:** Semantic segmentation does not highlight individual instances of a class differently.
- For example, if there were 3 cows in an image, the model would highlight the area they occupy, but it will not be able to distinguish one cow from another.
- Instance segmentation is one step ahead of semantic segmentation wherein along with pixel level classification, we expect the computer to classify each instance of a class separately.



Object Detection



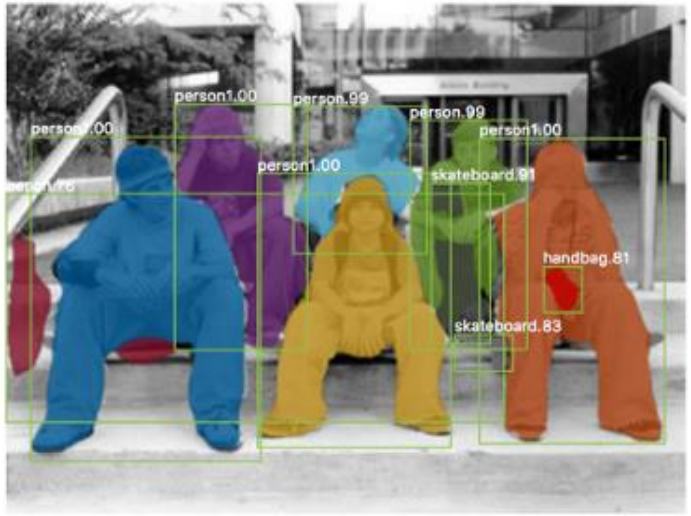
Semantic Segmentation



Instance Segmentation

Mask R-CNN

- If we want to add this functionality, we need to extend the task and introduce another term to complicate the already enormously large vocabulary of deep learning — **instance segmentation**.
- **Solution:** Mask R-CNN segments each “instance” of a class in an image in two steps:
  - Perform object detection to draw bounding boxes around each instance of a class.
  - Perform semantic segmentation on each of the bounding boxes.
- Review: https://medium.com/free-code-camp/mask-r-cnn-explained-7f82bec890e3



Mask R-CNN: https://arxiv.org/abs/1703.06870

Image-to-image translation problems

- For the most part, image segmentation is now quite achievable, and it's time to start thinking about innovative ways of using this idea of doing computer vision algorithms at a pixel-by-pixel level.
- Examples include
  - Optical flow estimation (motion estimation in video / between pairs of images).
  - Depth estimation
  - Surface normal estimation
  - Boundary detection
  - Human pose estimation
  - 3D reconstruction from 2D images
  - Neural style transfer

References

- **Overfeat:** https://arxiv.org/pdf/1312.6229.pdf
- **Selective search:** https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013/UijlingsIJCV2013.pdf
- **R-CNN:** https://people.eecs.berkeley.edu/~rbg/papers/r-cnn-cvpr.pdf
- **Fast R-CNN:** https://arxiv.org/abs/1504.08083
- **Faster R-CNN:** https://arxiv.org/abs/1506.01497
- **Mask R-CNN:** https://arxiv.org/pdf/1703.06870.pdf
- **YOLO:** https://arxiv.org/abs/1506.02640
- **SSD:** https://arxiv.org/pdf/1512.02325.pdf
- **FCN:** https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf
- **SegNet:** https://arxiv.org/pdf/1511.00561.pdf
- **U-Net:** https://arxiv.org/pdf/1505.04597.pdf

Recommended reading

- https://www.jeremyjordan.me/semantic-segmentation/
- https://www.jeremyjordan.me/evaluating-image-segmentation-models/
- https://towardsdatascience.com/review-segnet-semantic-segmentation-e66f2e30fb96
- https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
- https://www.datacamp.com/community/tutorials/object-detection-guide
- https://www.jeremyjordan.me/object-detection-one-stage/