CS 4644-DL / 7643-A: LECTURE 16
DANFEI XU

- 2D Computer Vision (Continued)
- 3D Vision: Representations and Neural Rendering

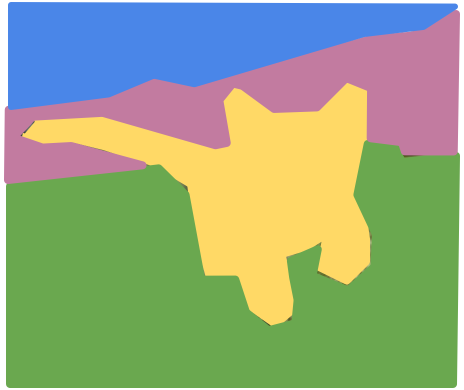
Computer Vision Tasks

Classification

CAT

No spatial extent

Semantic Segmentation



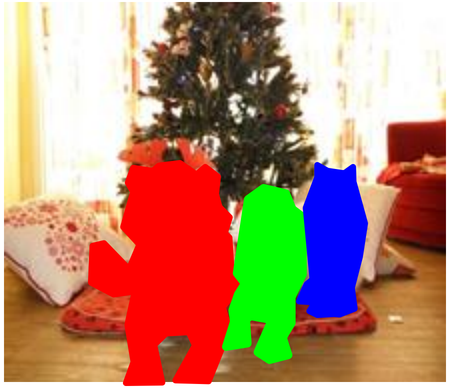
GRASS, CAT, TREE, SKY

No objects, just pixels

Object Detection

DOG, DOG, CAT

Instance Segmentation



DOG, DOG, CAT

Multiple Object

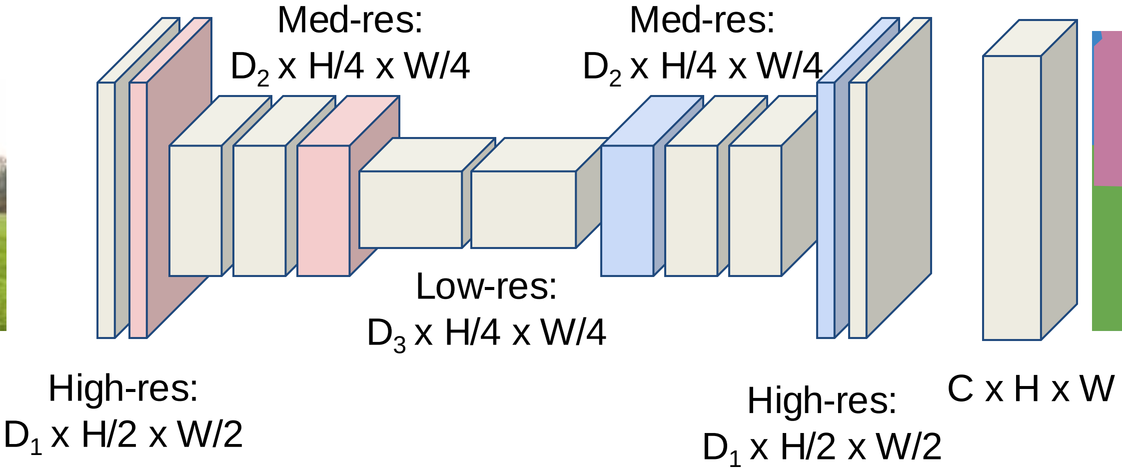
Semantic Segmentation Idea: Fully Convolutional

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



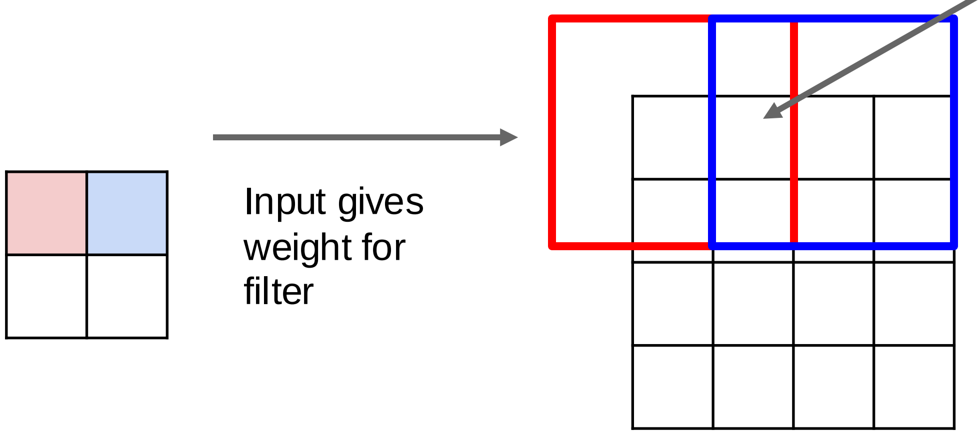
Predictions:
H x W

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

Learnable Upsampling: Transposed Convolution

3 x 3 **transpose** convolution, stride 2 pad 1

Sum where output overlaps



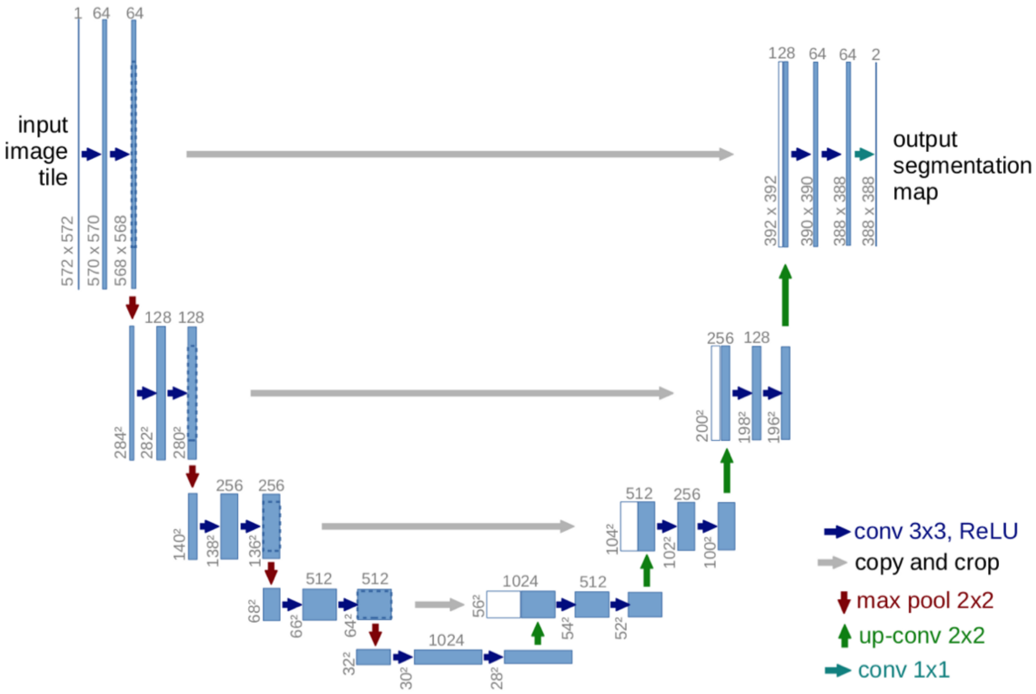
Input: 2 x 2

Output: 4 x 4

Filter moves 2 pixels in the <u>output</u> for every one pixel in the <u>input</u>

Stride gives ratio between movement in output and input

Semantic Segmentation: U-Net



Idea: Concatenate feature maps from the downsampling stage with the features in the upsampling stage.

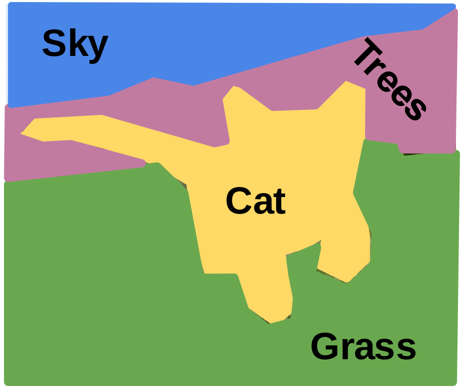
Very commonly used today!

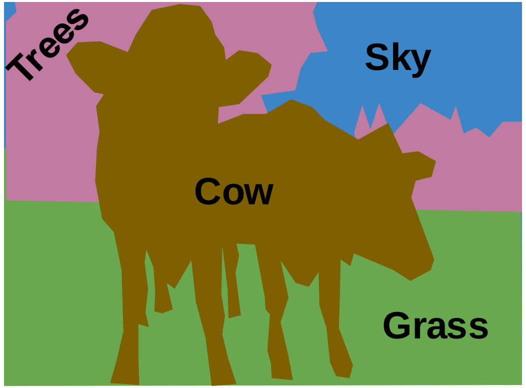
Ronneberger O, Fischer P, Brox T, 2015

Semantic Segmentation

Label each pixel in the image with a category label

Don't differentiate instances, only care about pixels





Object Detection

Classification

CAT

No spatial extent

Semantic Segmentation

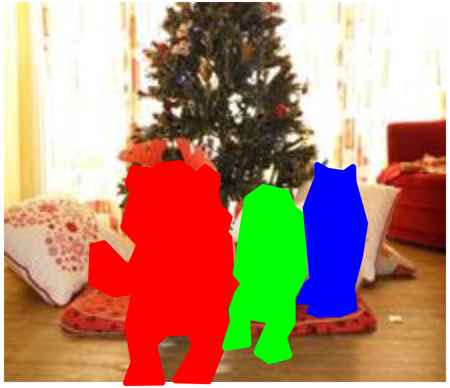
GRASS, CAT, TREE, SKY

No objects, just pixels

Object Detection

DOG, DOG, CAT

Instance Segmentation

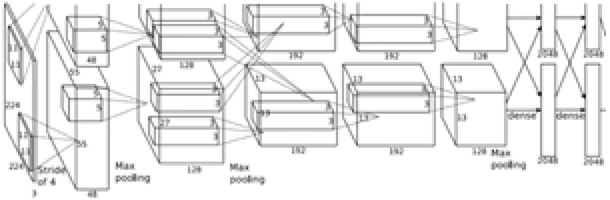


DOG, DOG, CAT

Multiple Object

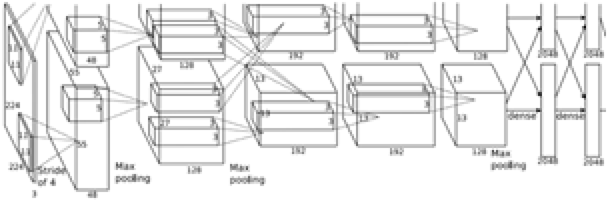
Object Detection: Multiple Objects

Each image needs a different number of outputs!



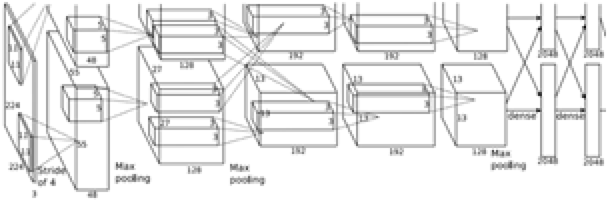
CAT: (x, y, w, h)

4 numbers



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

12 numbers

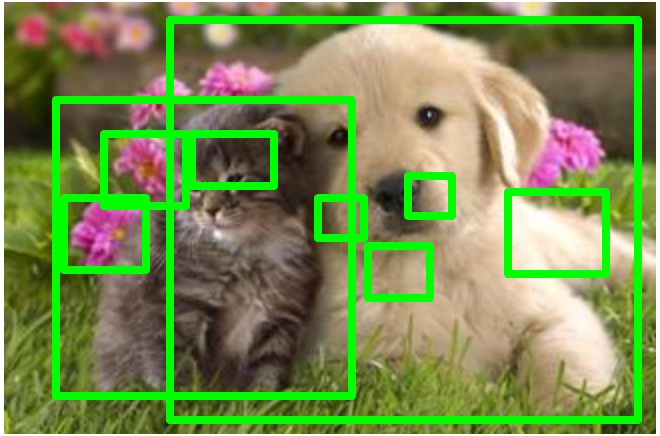


DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
….

Many numbers!

Region Proposals: Selective Search

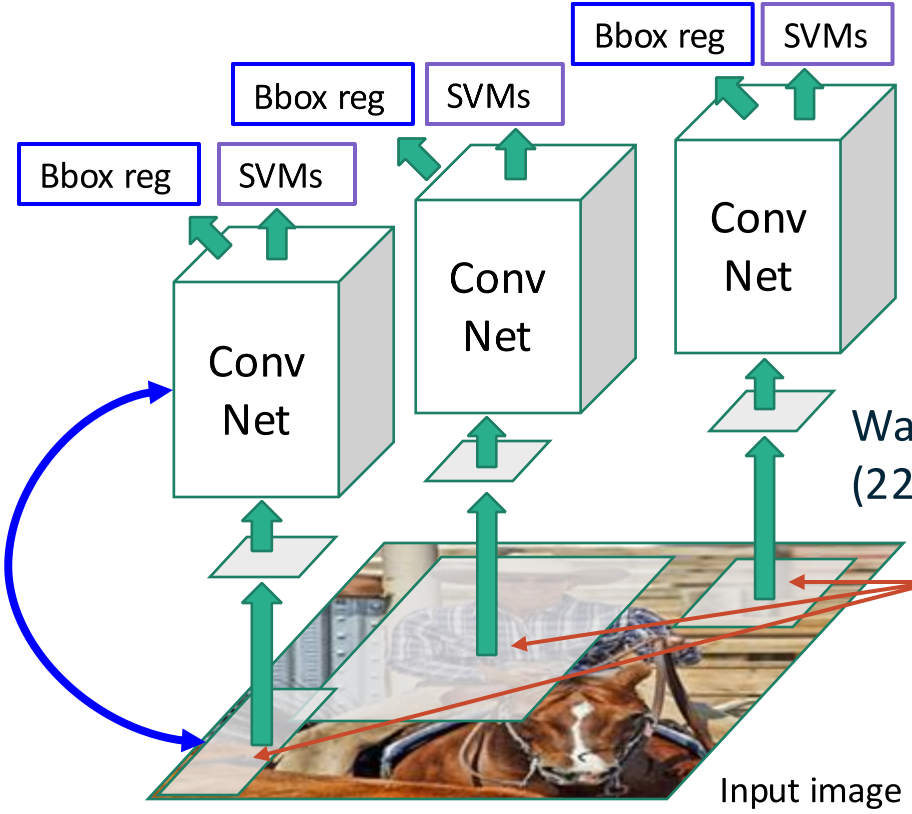
- Find “blobby” image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 2000 region proposals in a few seconds on CPU



Alexe et al, “Measuring the objectness of image windows”, TPAMI 2012
Uijlings et al, “Selective Search for Object Recognition”, IJCV 2013
Cheng et al, “BING: Binarized normed gradients for objectness estimation at 300fps”, CVPR 2014
Zitnick and Dollar, “Edge boxes: Locating object proposals from edges”, ECCV 2014

“Slow” R-CNN

Predict “corrections” to the RoI: 4 numbers: (dx, dy, dw, dh)



Classify regions with SVMs

Forward each region through ConvNet

Warped image regions (224x224 pixels)

Regions of Interest (RoI) from a proposal method (~2k)

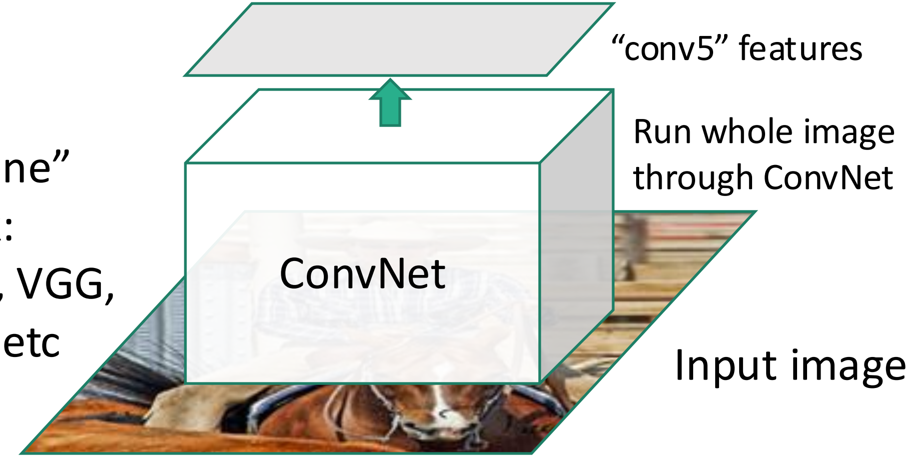
Problem: Very slow! Need to do ~2k independent forward passes for each image!

Idea: Pass the image through convnet before cropping! Crop the conv feature instead!

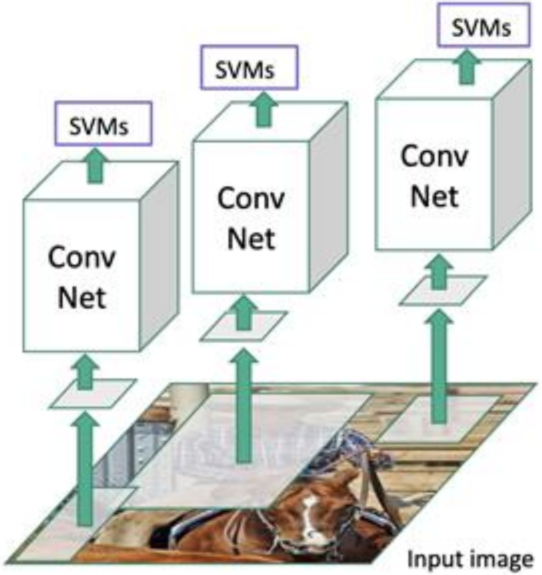
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

Fast R-CNN

“Backbone” network: AlexNet, VGG, ResNet, etc

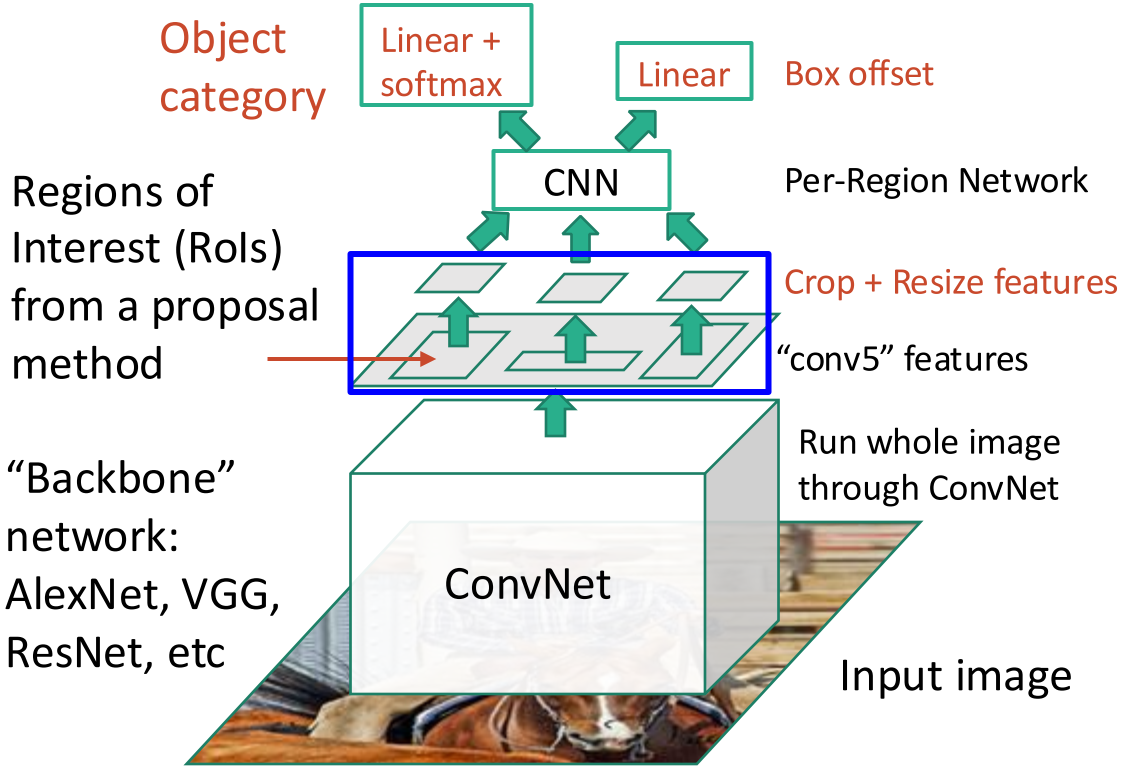


“Slow” R-CNN

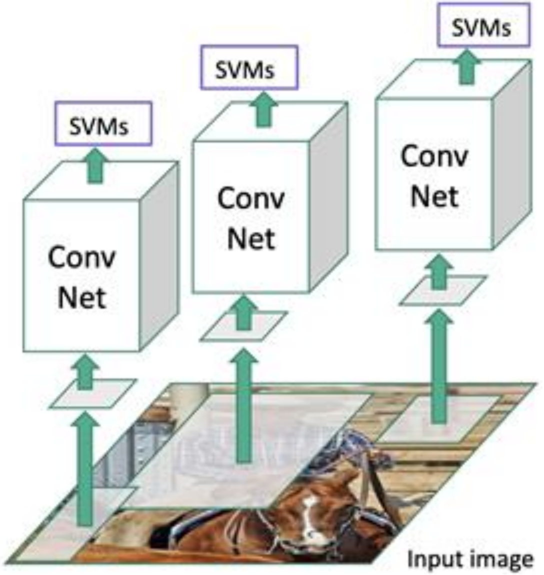


Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN

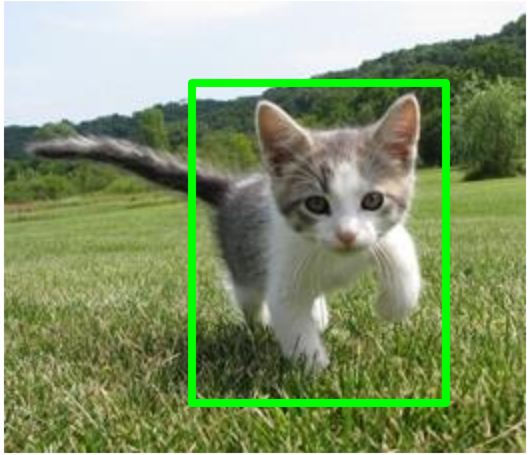


"Slow" R-CNN



Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Cropping Features: RoI Pool



Input Image
(e.g. 3 x 640 x 480)

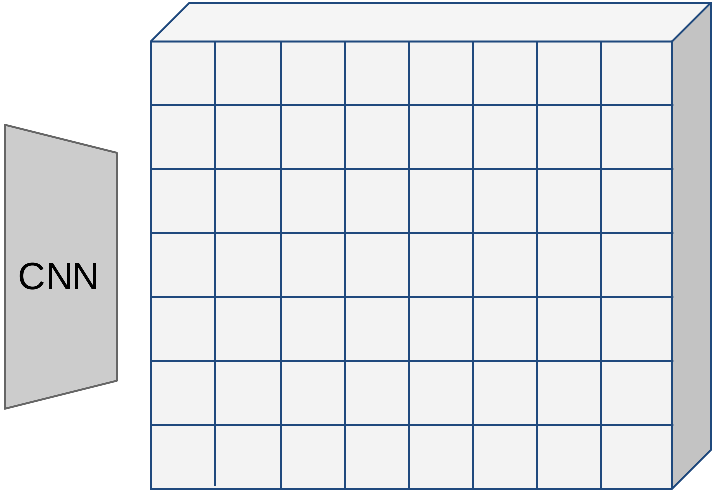
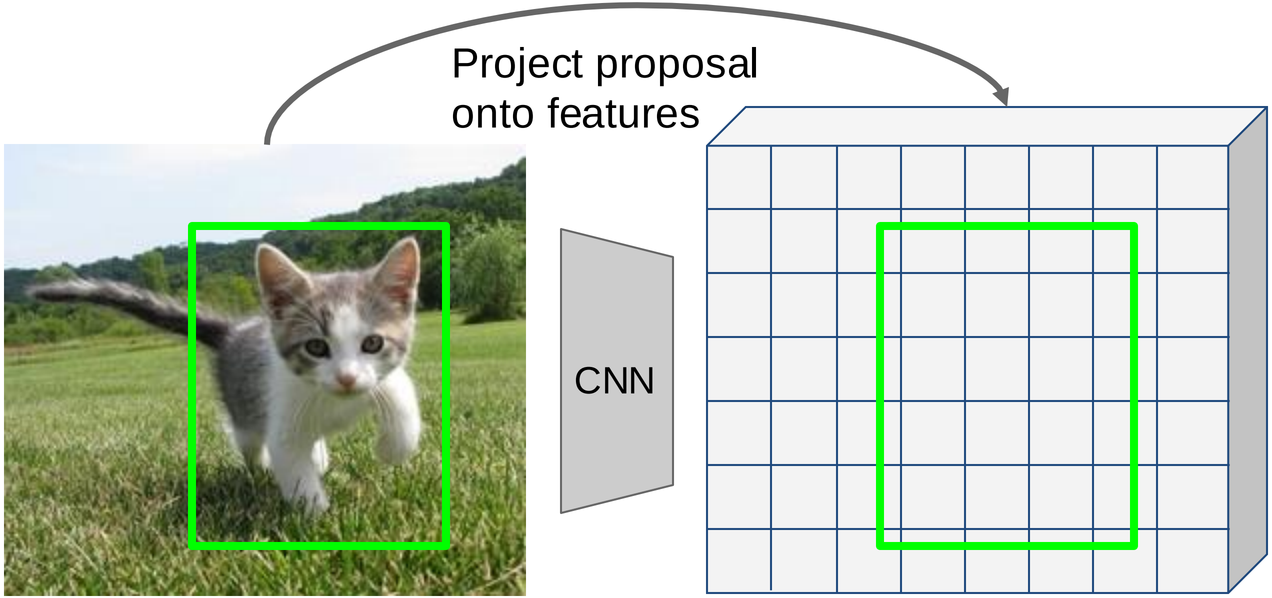


Image features: C x H x W
(e.g. 512 x 20 x 15)

Girshick, “Fast R-CNN”, ICCV 2015.

Girshick, “Fast R-CNN”, ICCV 2015.

Cropping Features: RoI Pool



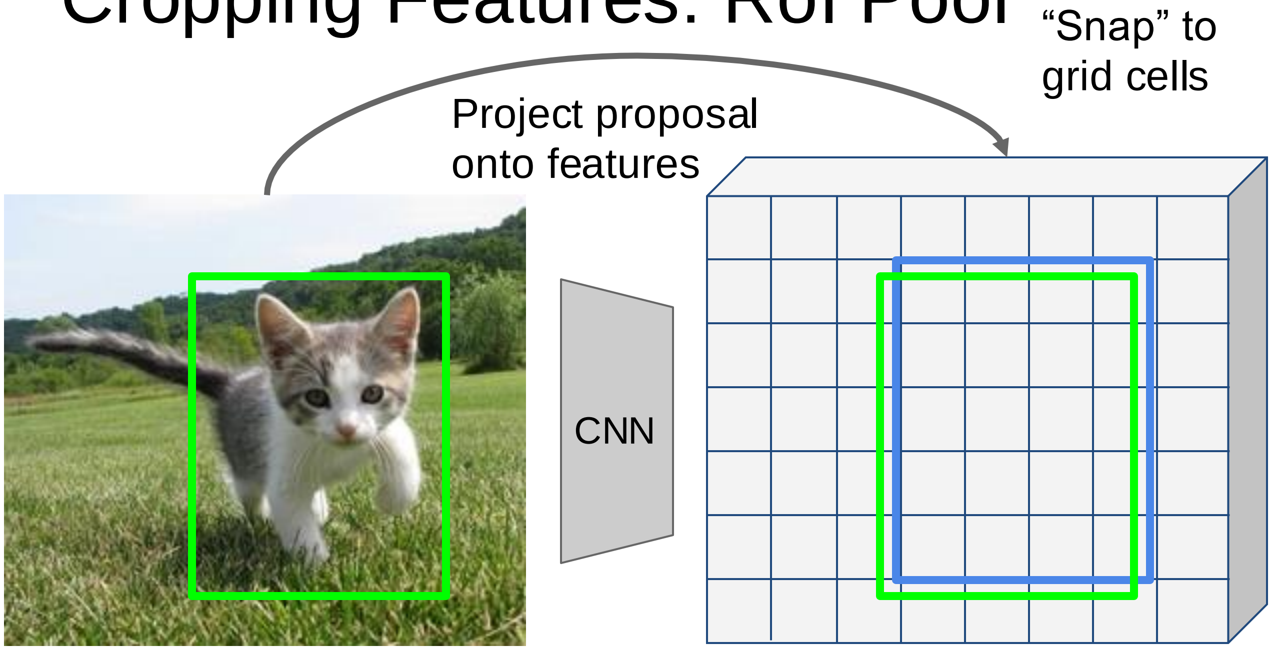
Input Image
(e.g. 3 x 640 x 480)

Image features: C x H x W
(e.g. 512 x 20 x 15)

Girshick, "Fast R-CNN", ICCV 2015.

Girshick, "Fast R-CNN", ICCV 2015.

Cropping Features: RoI Pool

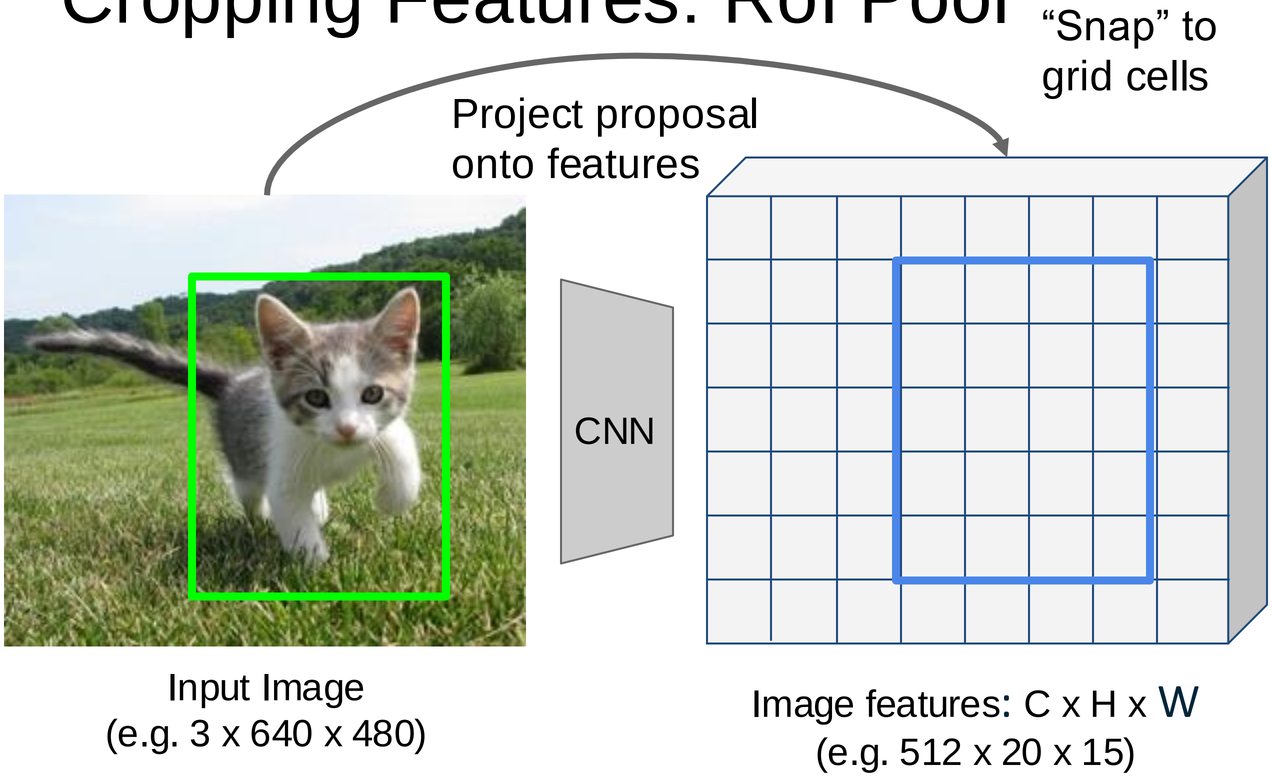


Input Image
(e.g. 3 x 640 x 480)

Image features: C x H x W
(e.g. 512 x 20 x 15)

Girshick, "Fast R-CNN", ICCV 2015.

Cropping Features: RoI Pool



Q: how do we resize the 512 x 20 x 15 region to, e.g., a 512 x 2 x 2 tensor?.

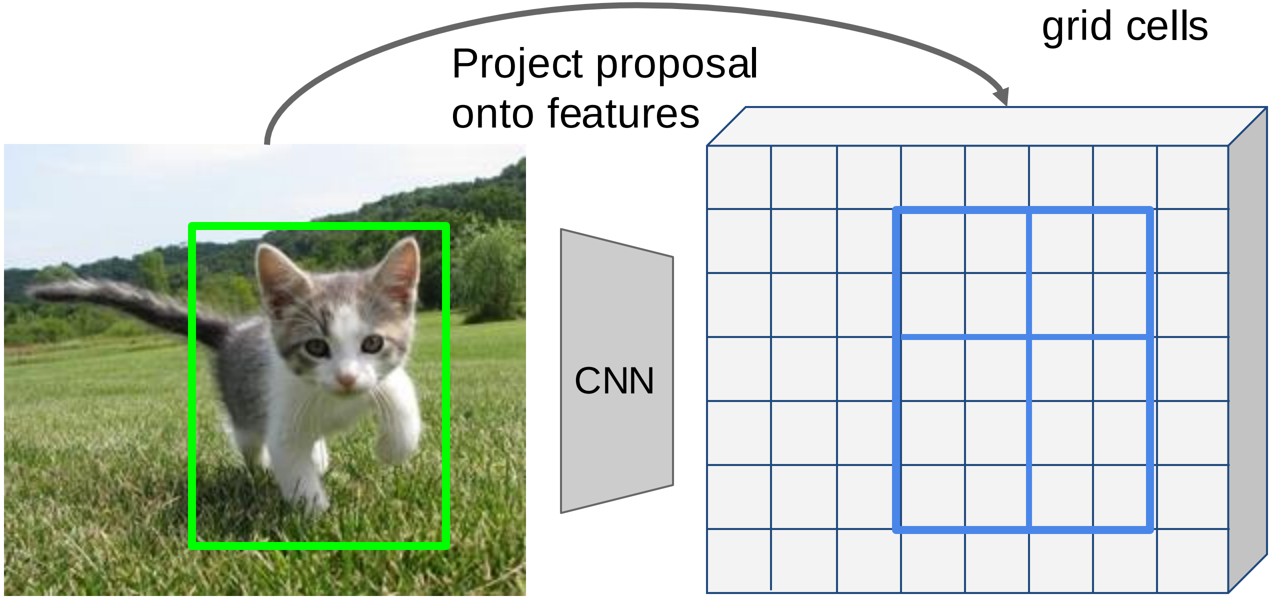
Girshick, "Fast R-CNN", ICCV 2015.

Cropping Features: RoI Pool

"Snap" to grid cells

Divide into 2x2 grid of (roughly) equal subregions

Project proposal onto features



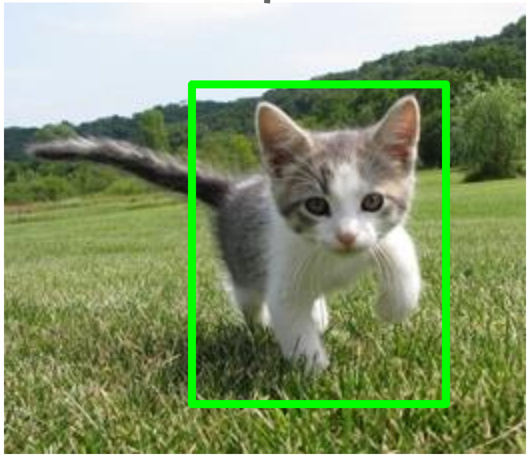
Input Image
(e.g. 3 x 640 x 480)

Image features: C x H x W
(e.g. 512 x 20 x 15)

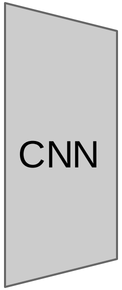
Q: how do we resize the 512 x 20 x 15 region to, e.g., a 512 x 2 x 2 tensor?.

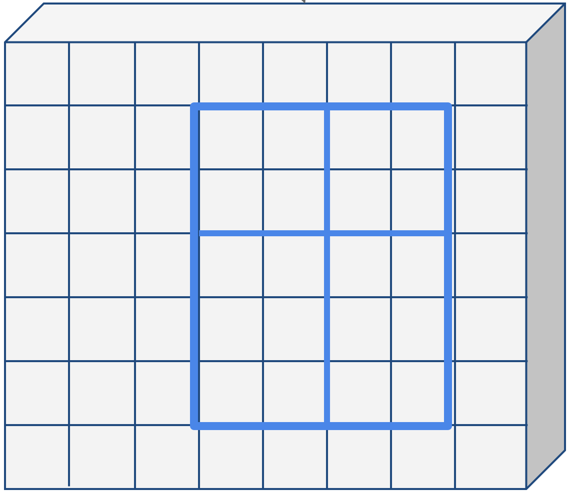
Girshick, "Fast R-CNN", ICCV 2015.

Cropping Features: RoI Pool



Project proposal
onto features

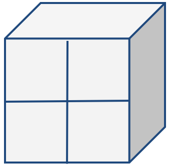




"Snap" to
grid cells

Divide into 2x2
grid of (roughly)
equal subregions

Max-pool within
each subregion



Input Image
(e.g. 3 x 640 x 480)

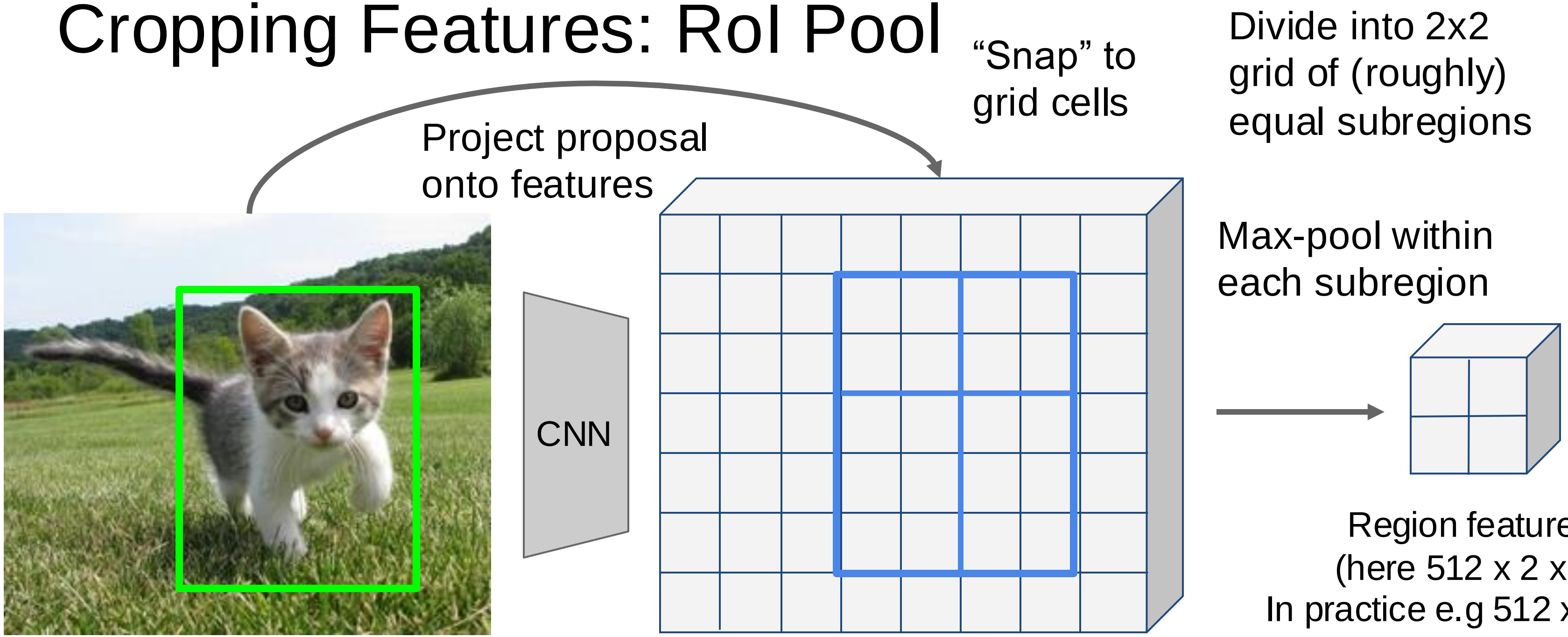
Image features: C x H x W
(e.g. 512 x 20 x 15)

Region features
(here 512 x 2 x 2;
In practice e.g 512 x 7 x 7)

Region features always the
same size even if input
regions have different sizes!

Girshick, "Fast R-CNN", ICCV 2015.

Cropping Features: RoI Pool



Input Image
(e.g. 3 x 640 x 480)

Image features: C x H x W
(e.g. 512 x 20 x 15)

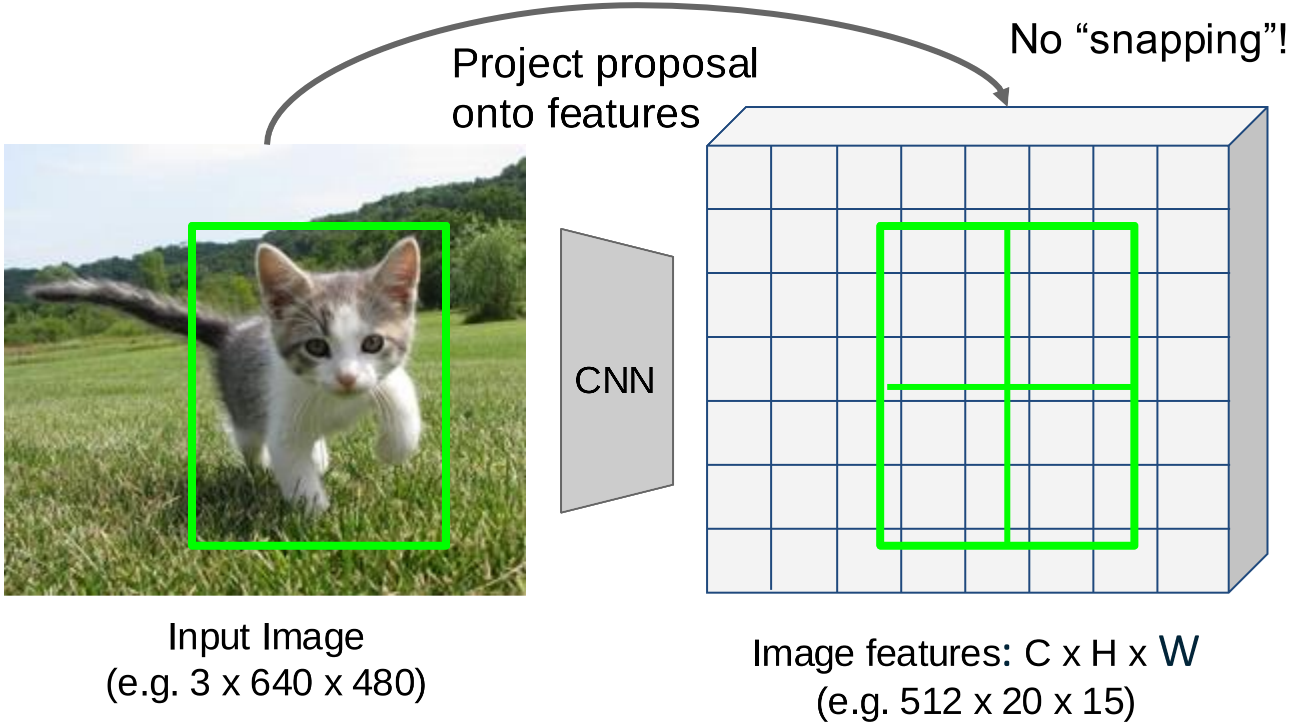
Region features
(here 512 x 2 x 2;
In practice e.g 512 x 7 x 7)

Region features always the same size even if input regions have different sizes!

Problem: Region features slightly misaligned

Girshick, “Fast R-CNN”, ICCV 2015.

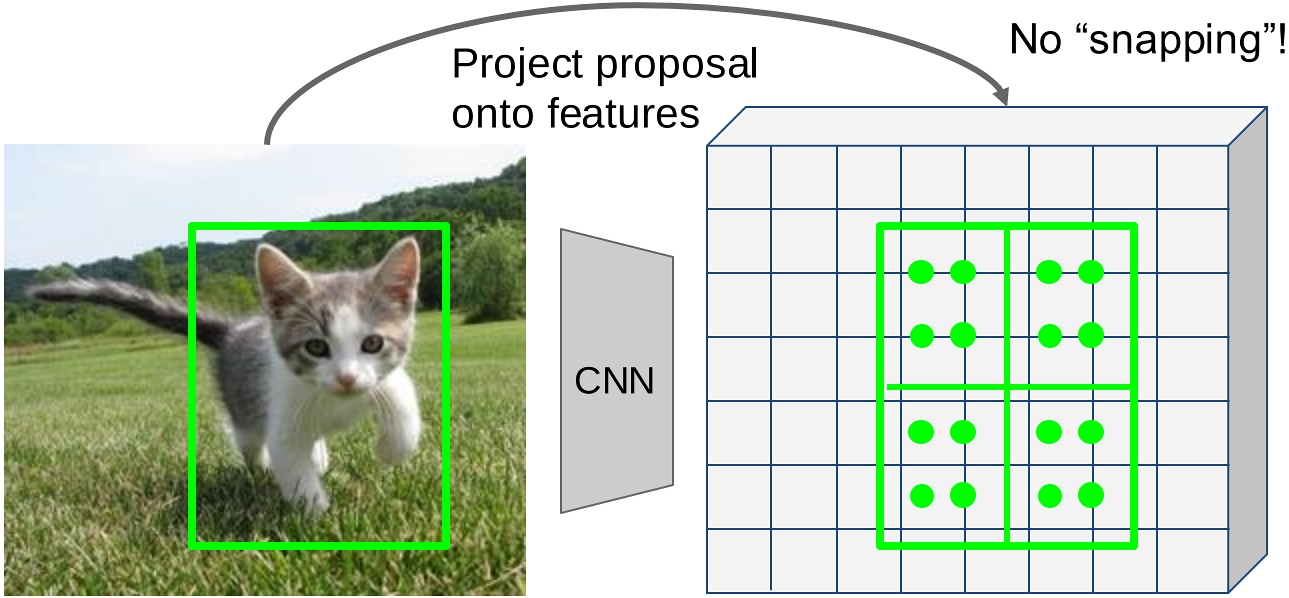
Cropping Features: RoI Align



He et al, “Mask R-CNN”, ICCV 2017

Cropping Features: RoI Align

Sample at regular points in each subregion using bilinear interpolation



Input Image
(e.g. 3 x 640 x 480)

Image features: C x H x W
(e.g. 512 x 20 x 15)

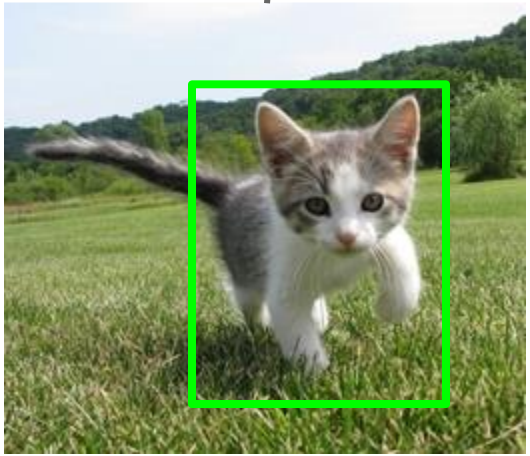
He et al, “Mask R-CNN”, ICCV 2017

Cropping Features: RoI Align

Sample at regular points in each subregion using bilinear interpolation

Project proposal onto features

No "snapping"!



Input Image
(e.g. 3 x 640 x 480)

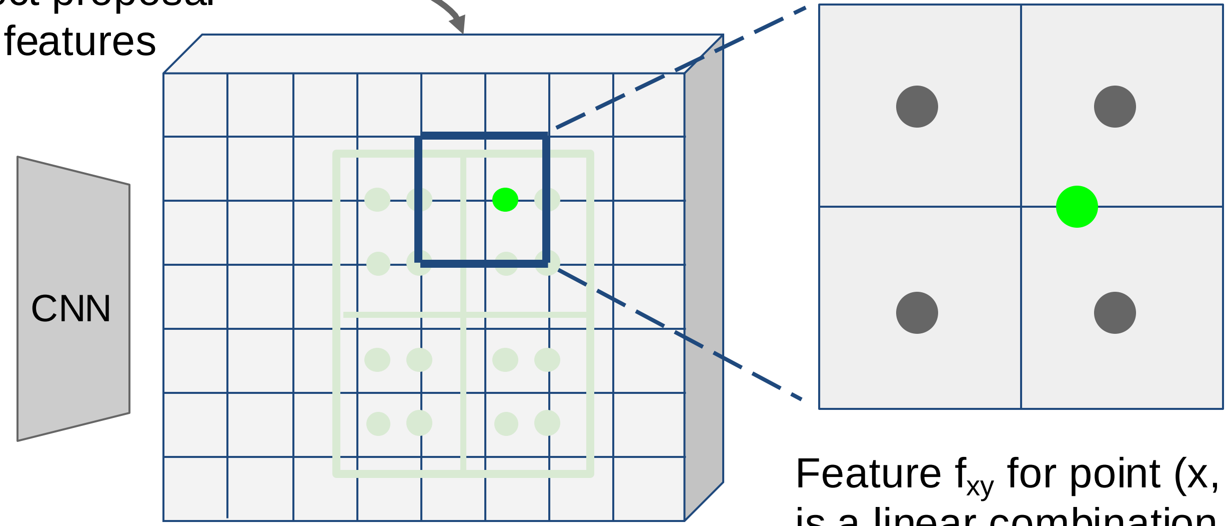


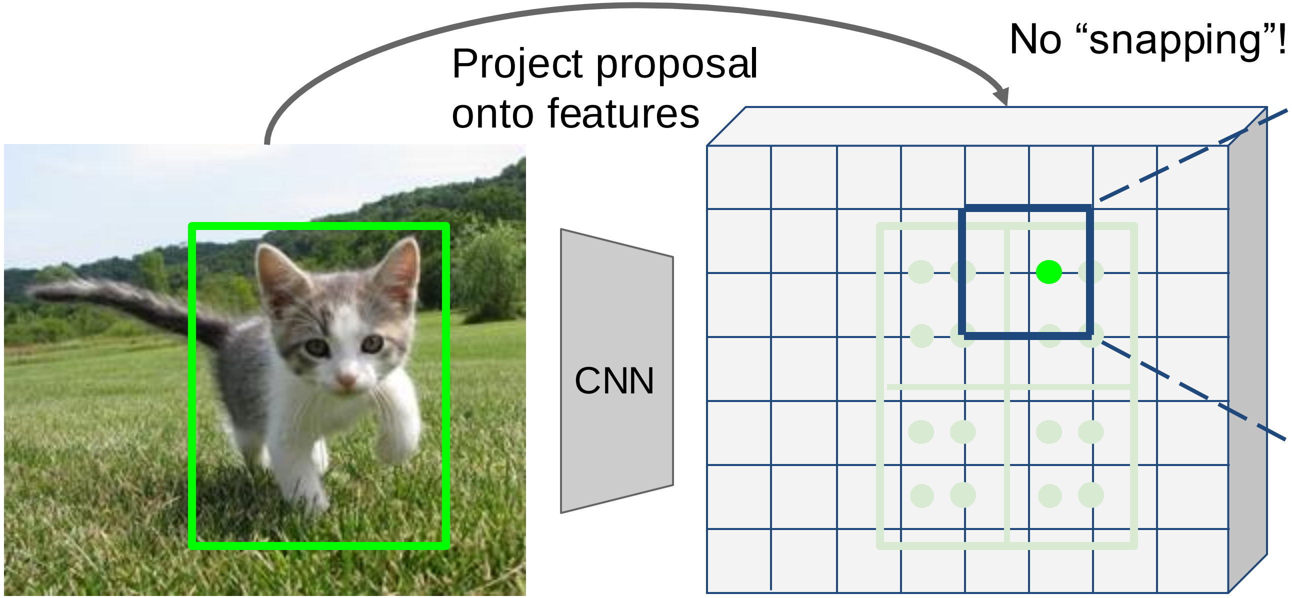
Image features: C x H x W
(e.g. 512 x 20 x 15)

Feature f_{xy} for point (x, y) is a linear combination of features at its four neighboring grid cells:

He et al, "Mask R-CNN", ICCV 2017

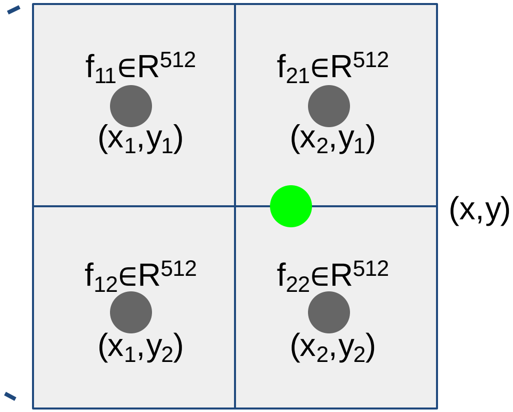
Cropping Features: RoI Align

Sample at regular points in each subregion using bilinear interpolation



Input Image
(e.g. 3 x 640 x 480)

Image features: C x H x W
(e.g. 512 x 20 x 15)



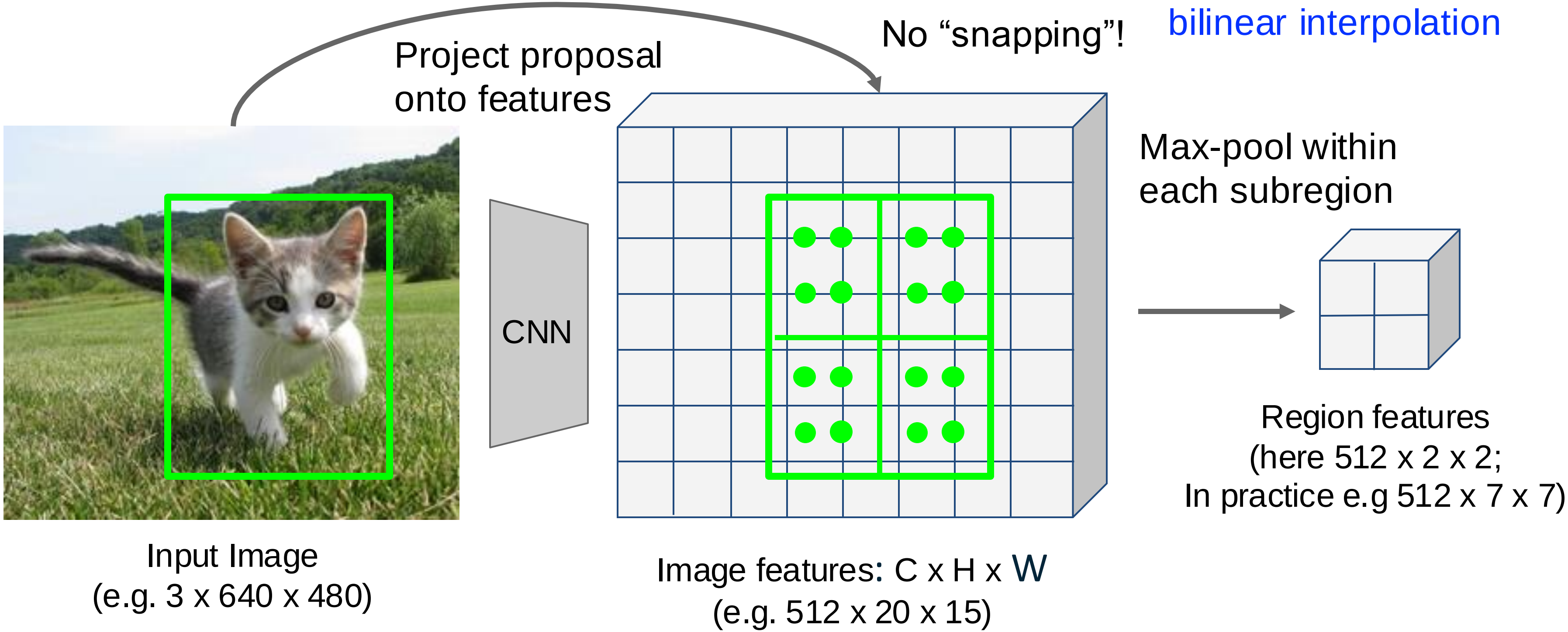
Feature f_{xy} for point (x, y) is a linear combination of features at its four neighboring grid cells:

$$f_{xy} = \sum_{i,j=1}^{2} f_{i,j} \max(0, 1 - |x - x_i|) \max(0, 1 - |y - y_j|)$$

He et al, "Mask R-CNN", ICCV 2017

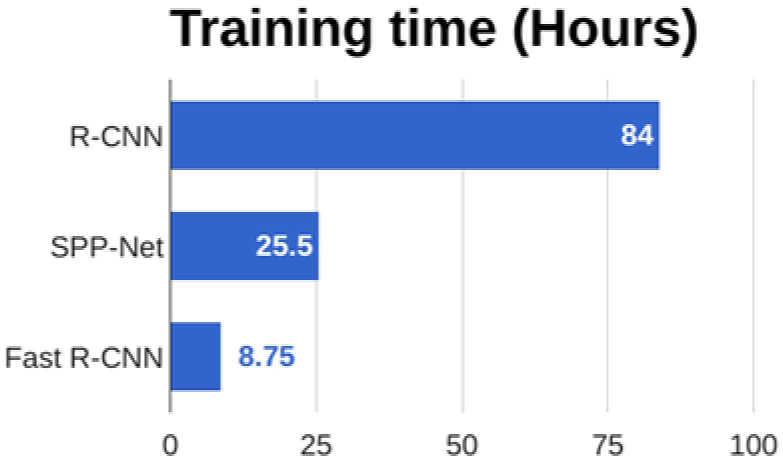
Cropping Features: RoI Align

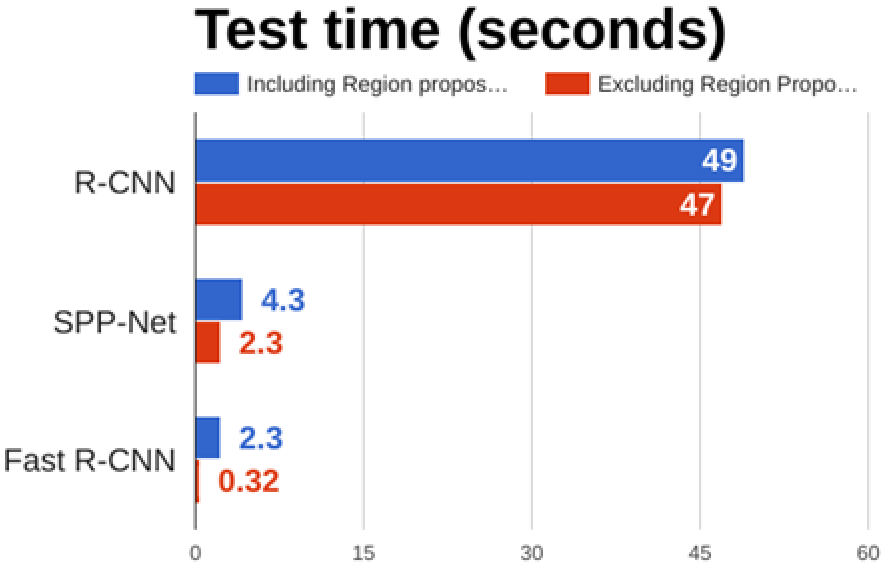
Sample at regular points in each subregion using bilinear interpolation



He et al, “Mask R-CNN”, ICCV 2017

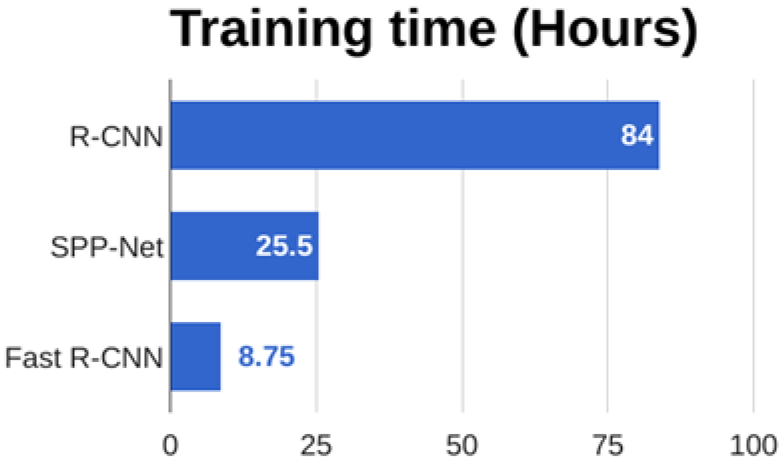
R-CNN vs Fast R-CNN

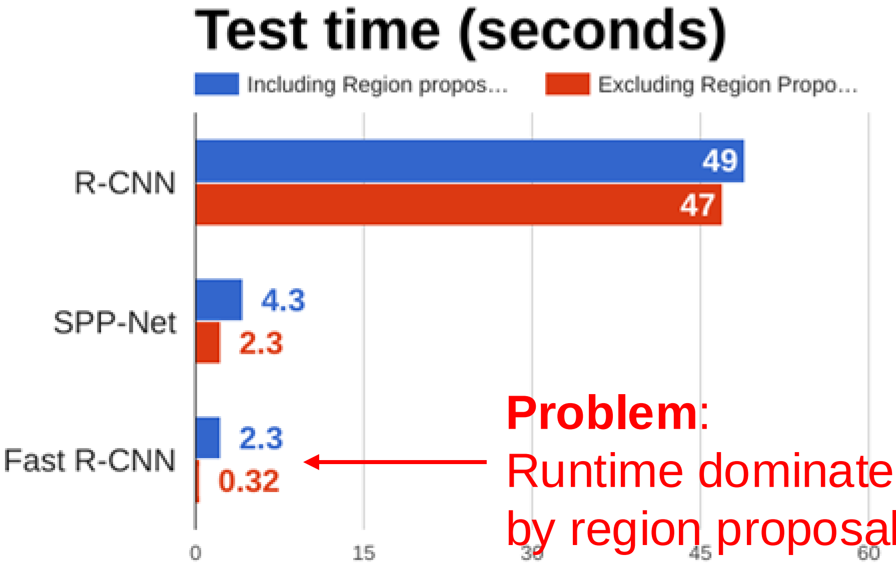




Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
He et al, “Spatial pyramid pooling in deep convolutional networks for visual recognition”, ECCV 2014
Girshick, “Fast R-CNN”, ICCV 2015

R-CNN vs Fast R-CNN





Problem:
Runtime dominated by region proposals!

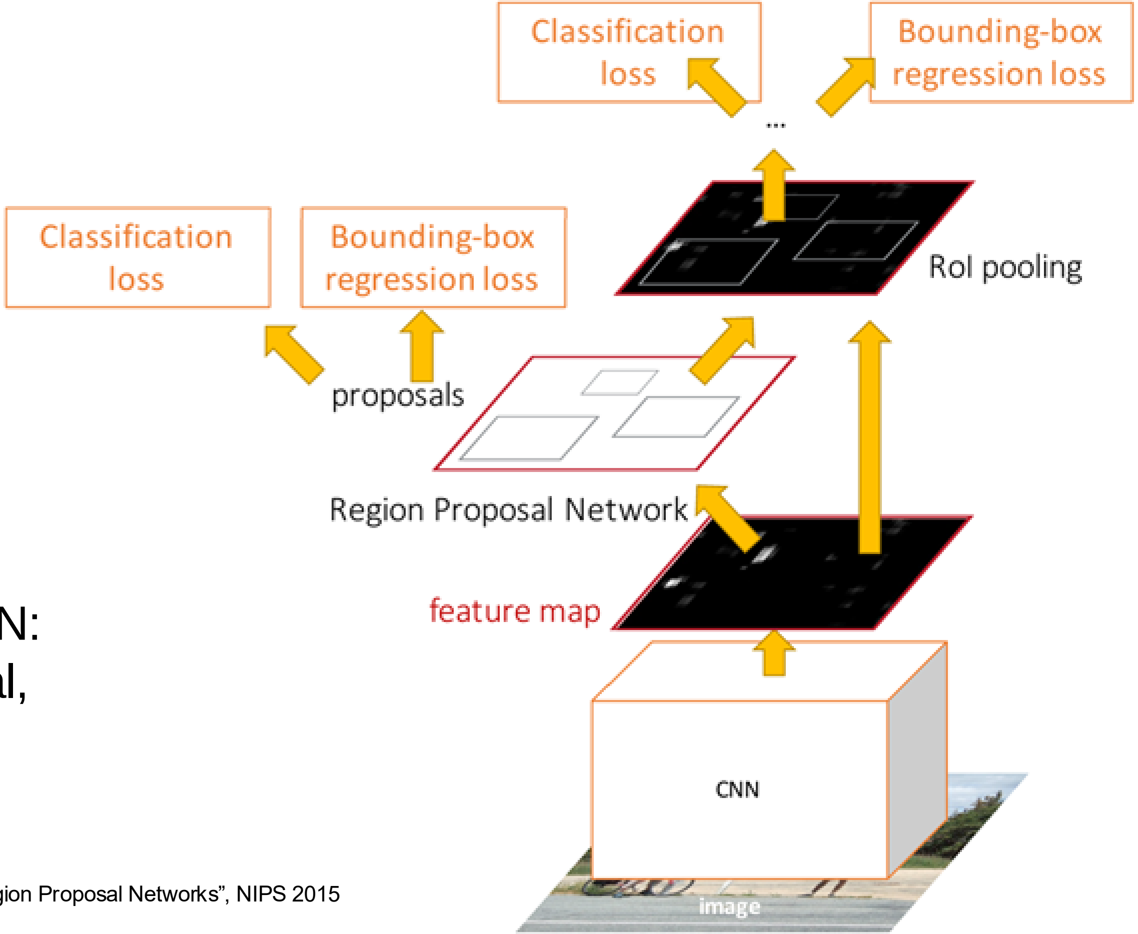
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
He et al, “Spatial pyramid pooling in deep convolutional networks for visual recognition”, ECCV 2014
Girshick, “Fast R-CNN”, ICCV 2015

Faster R-CNN:

Make CNN do proposals!

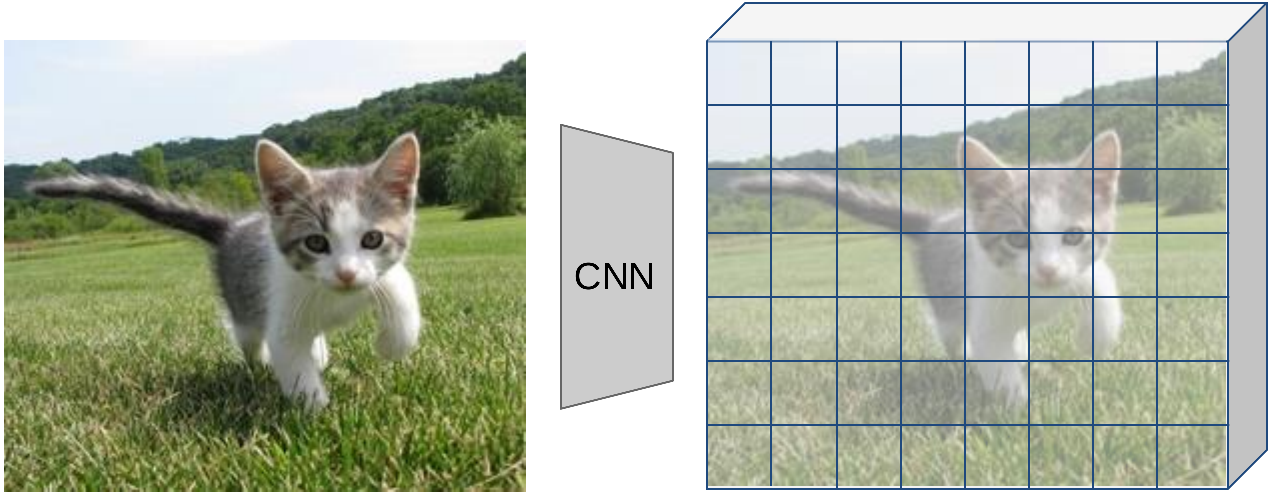
Insert **Region Proposal Network (RPN)** to predict proposals from features

Otherwise same as Fast R-CNN: Crop features for each proposal, classify each one



Ren et al, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, NIPS 2015

Region Proposal Network

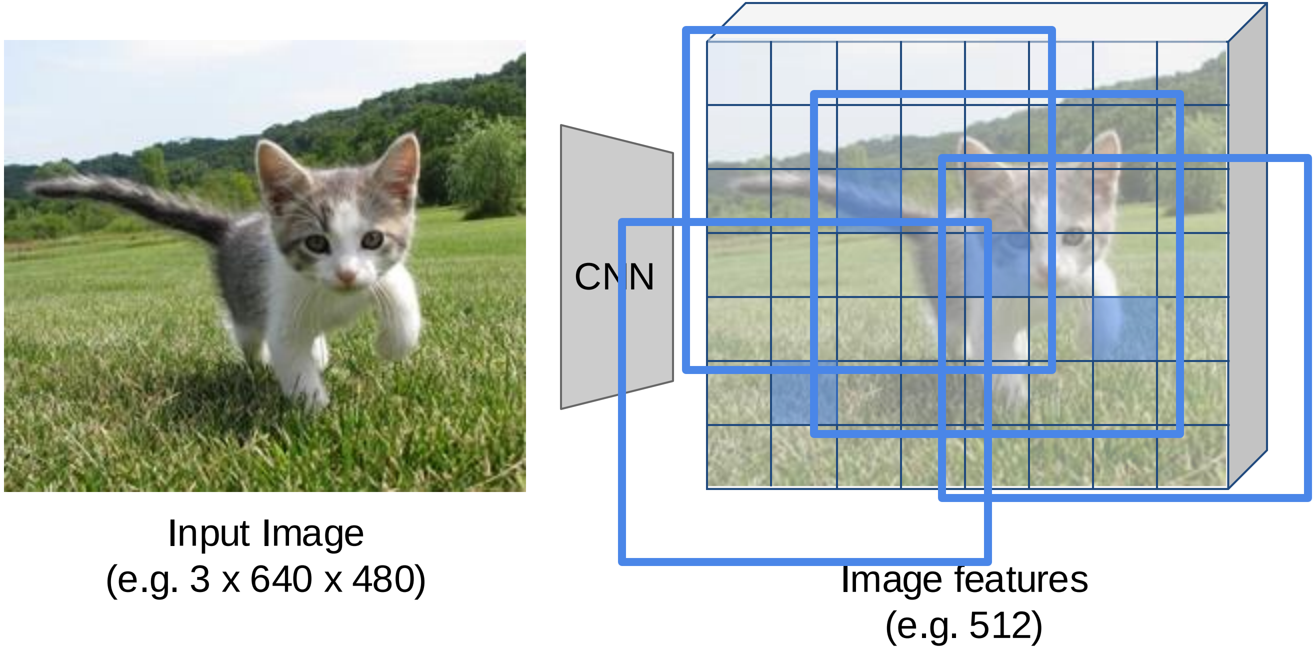


Input Image
(e.g. 3 x 640 x 480)

Image features
(e.g. 512 x 20 x 15)

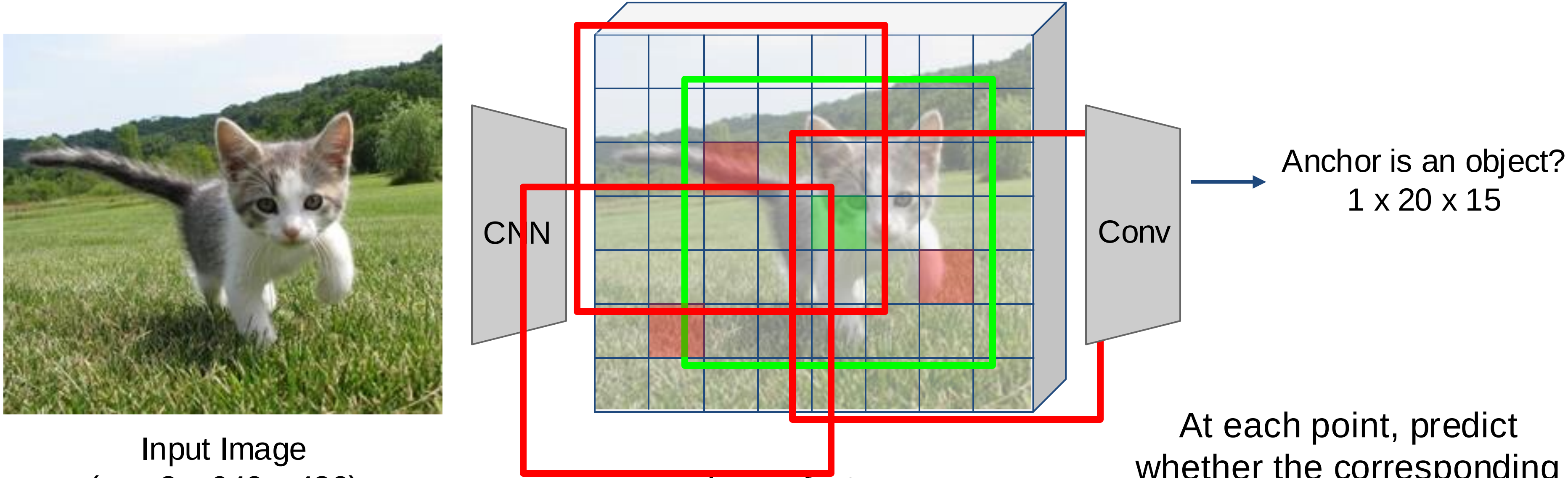
Region Proposal Network

Imagine an **anchor box** of fixed size at each point in the feature map



Region Proposal Network

Example: 20 x 15 **anchor box** uniformly sampled on the feature map



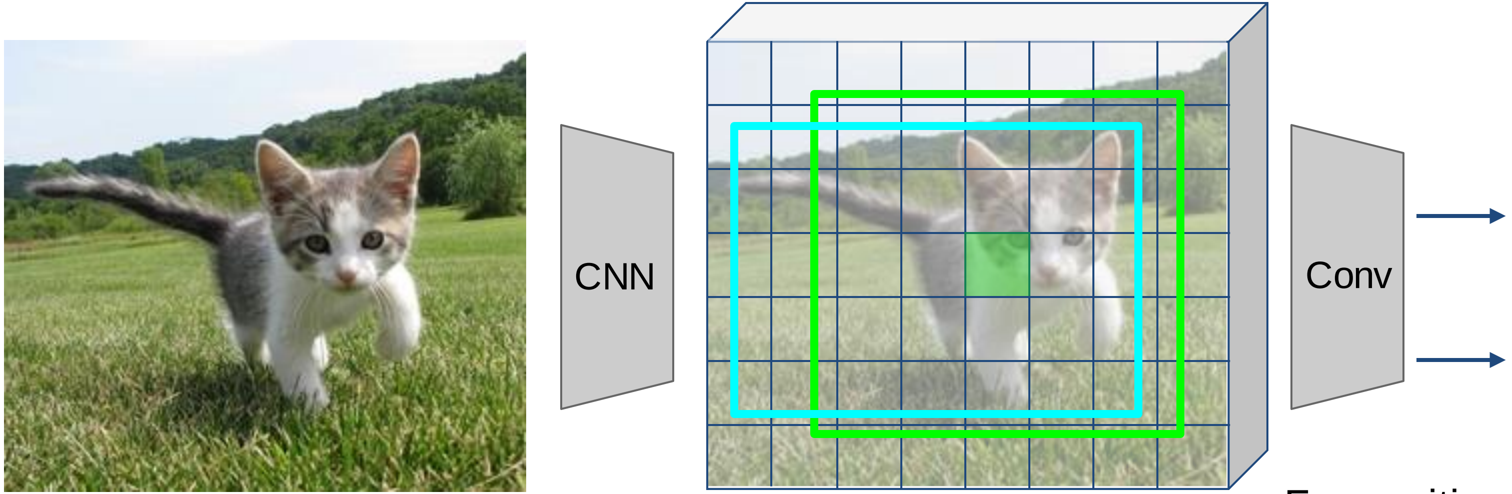
Input Image
(e.g. 3 x 640 x 480)

Image features
(e.g. 512)

At each point, predict whether the corresponding anchor contains an object (binary classification)

Region Proposal Network

Example: 20 x 15 **anchor box** uniformly sampled on the feature map



Anchor is an object?
1 x 20 x 15

Box corrections
4 x 20 x 15

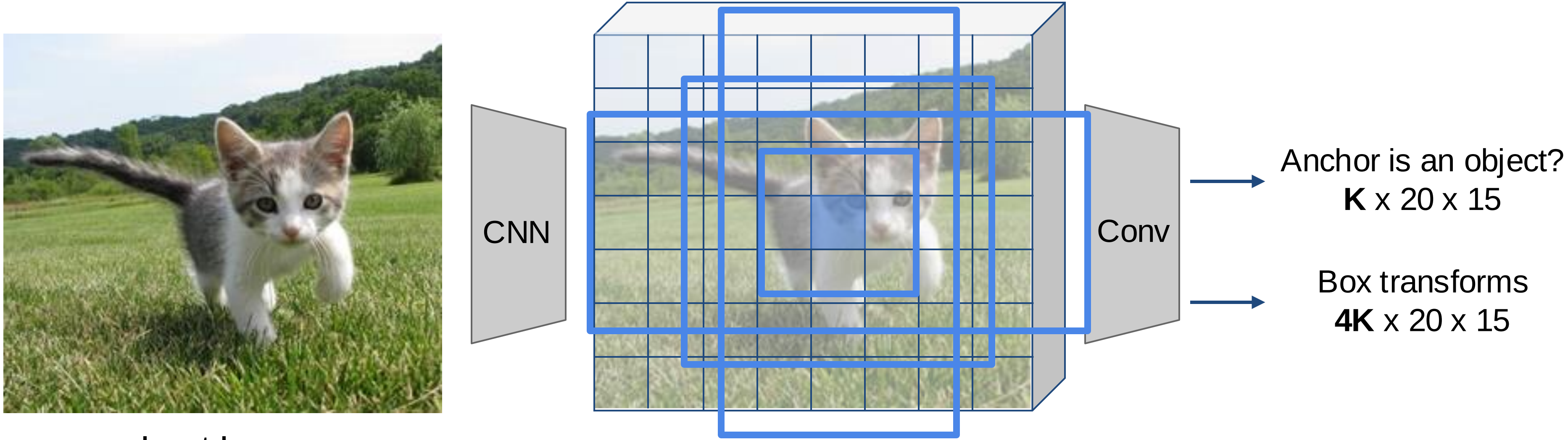
Input Image
(e.g. 3 x 640 x 480)

Image features
(e.g. 512)

For positive boxes, also predict a corrections from the anchor to the ground-truth box (regress 4 numbers per pixel)

Region Proposal Network

In practice use K different anchor boxes of different size / scale at each point

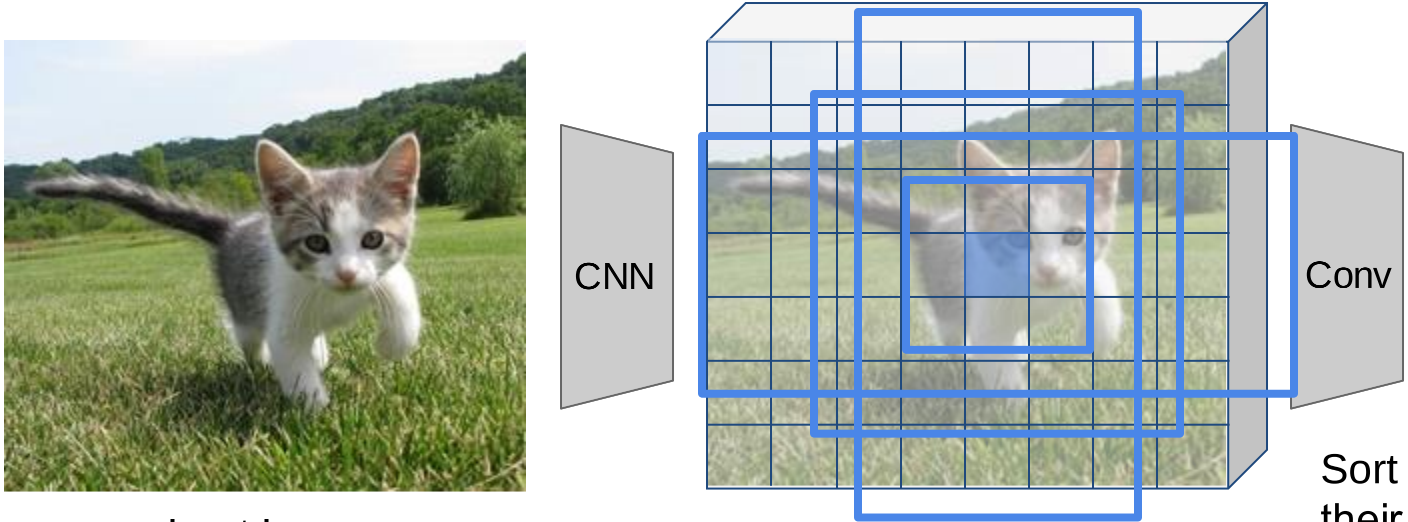


Input Image
(e.g. 3 x 640 x 480)

Image features
(e.g. 512)

Region Proposal Network

In practice use K different anchor boxes of different size / scale at each point



Input Image
(e.g. 3 x 640 x 480)

Image features
(e.g. 512)

Anchor is an object?
K x 20 x 15

Box transforms
4K x 20 x 15

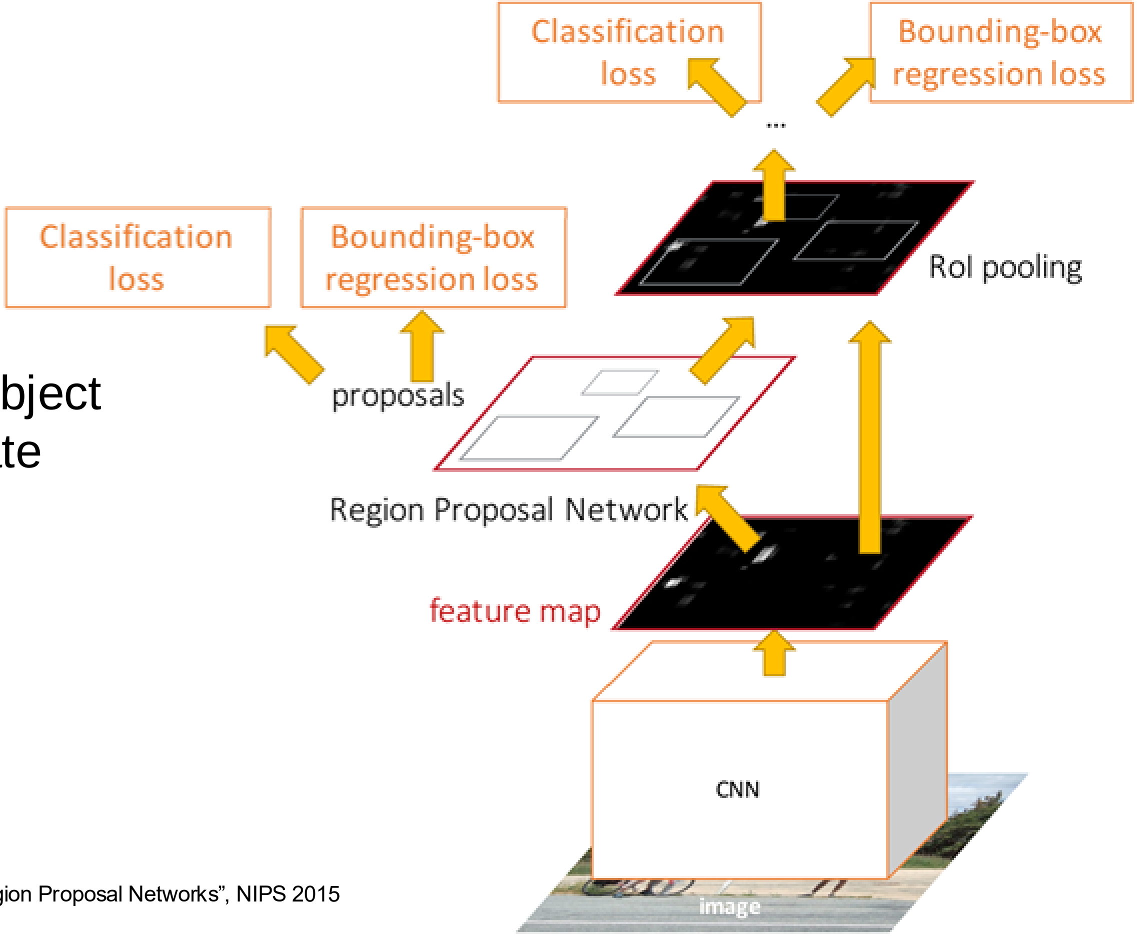
Sort the K*20*15 boxes by their “objectness” score, take top ~300 as our proposals

Fast<u>**er**</u> R-CNN:

Make CNN do proposals!

Jointly train with 4 losses:

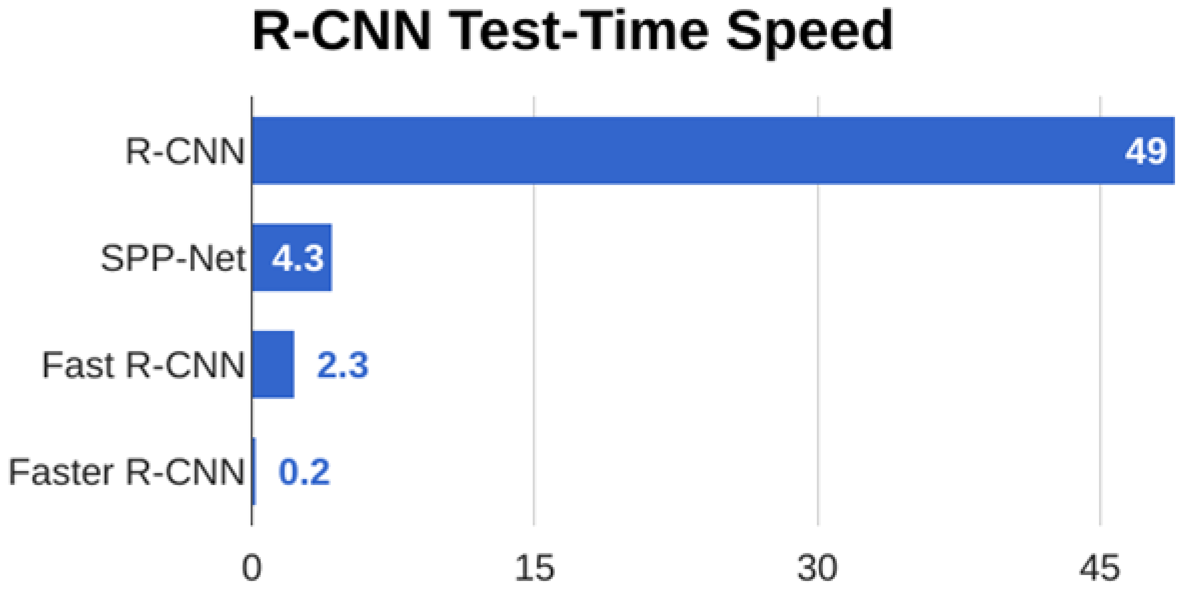
1. RPN classify object / not object
2. RPN regress box coordinate residuals
3. Final classification score (object classes)
4. Final box coordinates



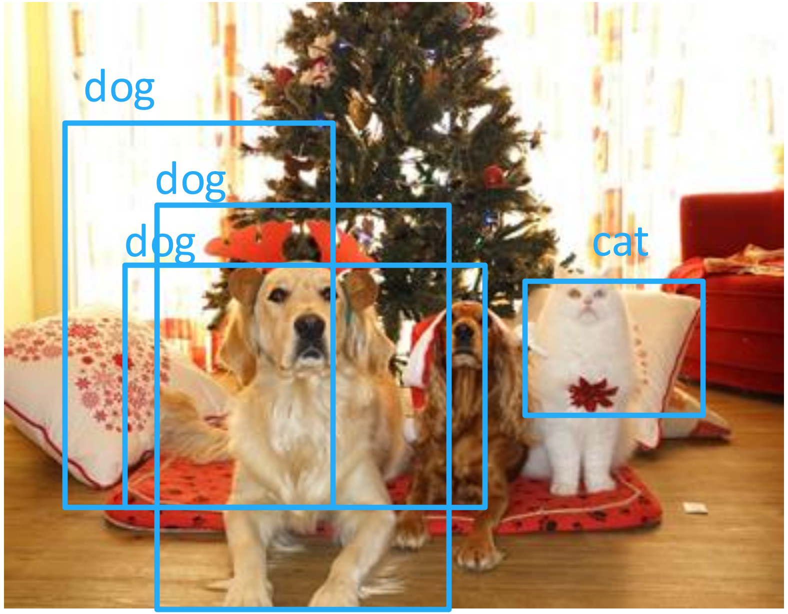
Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015

Fast**er** R-CNN:

Make CNN do proposals!



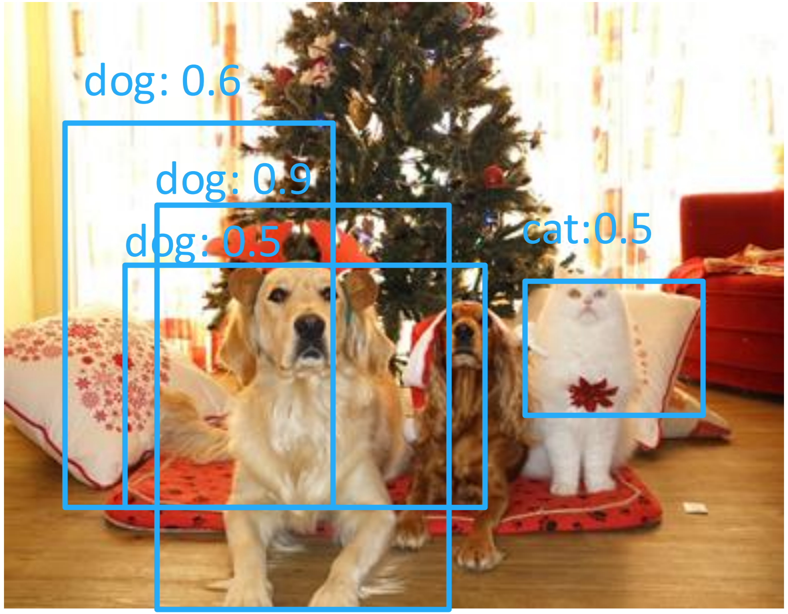
Which prediction to pick?



Problem: Detectors almost always generate more box predictions than the number of objects in the image!
E.g., 300 is an upper bound how many objects we wish to detect.

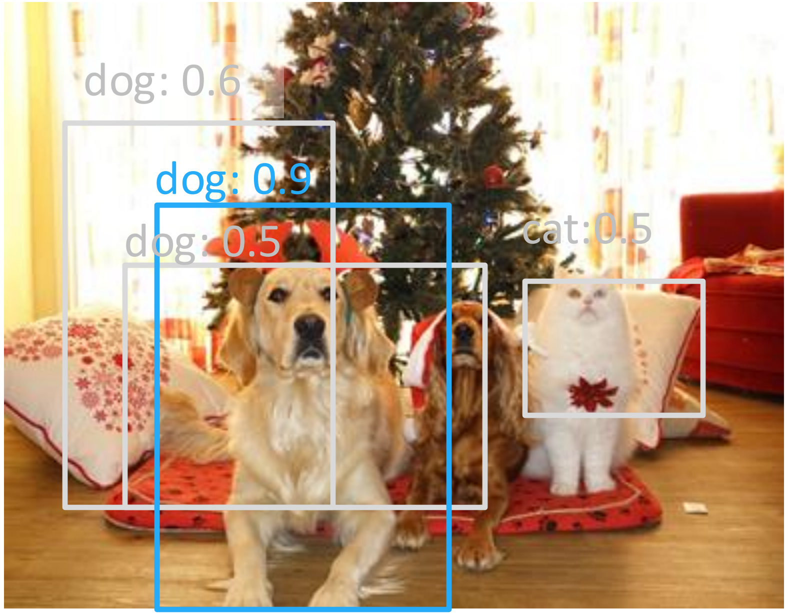
We need to remove the **redundant predictions!**

Non-Max Suppression (NMS)



Intuitively: locally pick the box that has the highest "objectless" or class score and suppress other boxes that have significant overlap with the chosen box

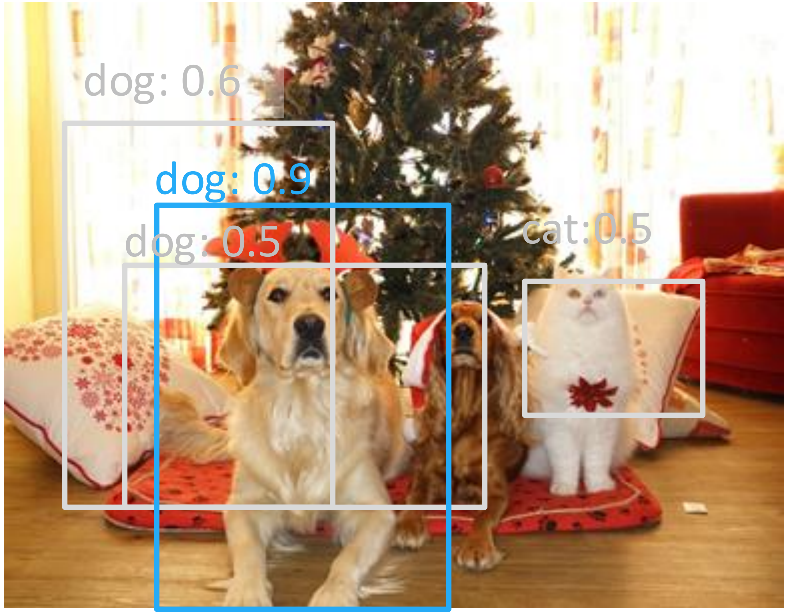
Non-Max Suppression (NMS)



Intuitively: locally pick the box that has the highest "objectless" or class score and suppress other boxes that have significant overlap with the chosen box

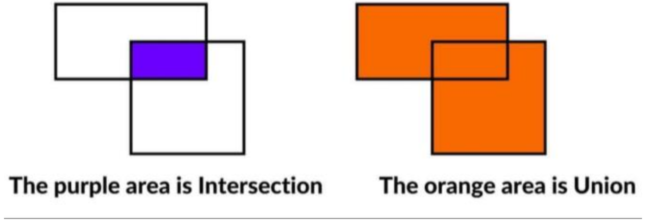
Step 1: Pick highest-score prediction box

Non-Max Suppression (NMS)

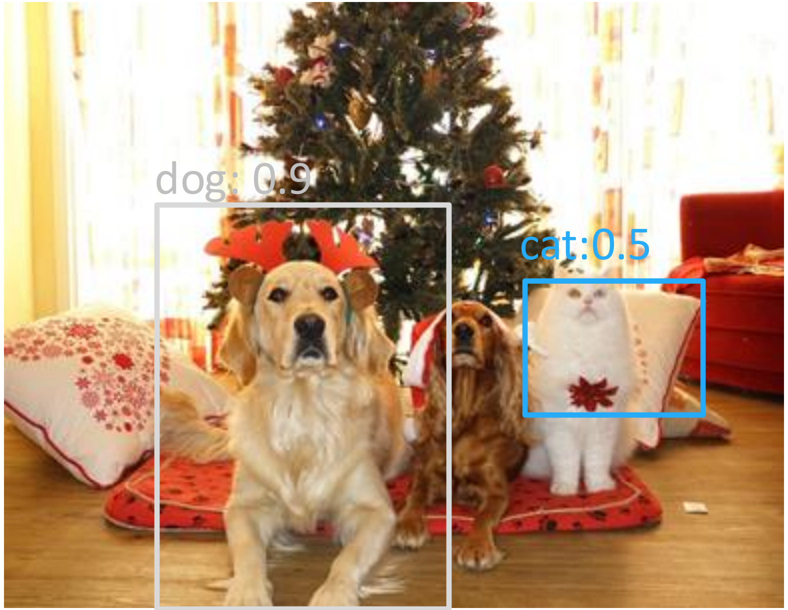


Intuitively: locally pick the box that has the highest "objectless" or class score and suppress other boxes that have significant overlap with the chosen box

Step 1: Pick highest-score prediction box
Step 2: Remove bounding boxes with **Intersection over Union (IoU) scores** higher than certain threshold (e.g., 0.5)



Non-Max Suppression (NMS)



Intuitively: locally pick the box that has the highest “objectless” or class score and suppress other boxes that have significant overlap with the chosen box

Step 1: Pick highest-score prediction box
Step 2: Remove bounding boxes with **Intersection over Union (IoU) scores** higher than certain threshold (e.g., 0.5)

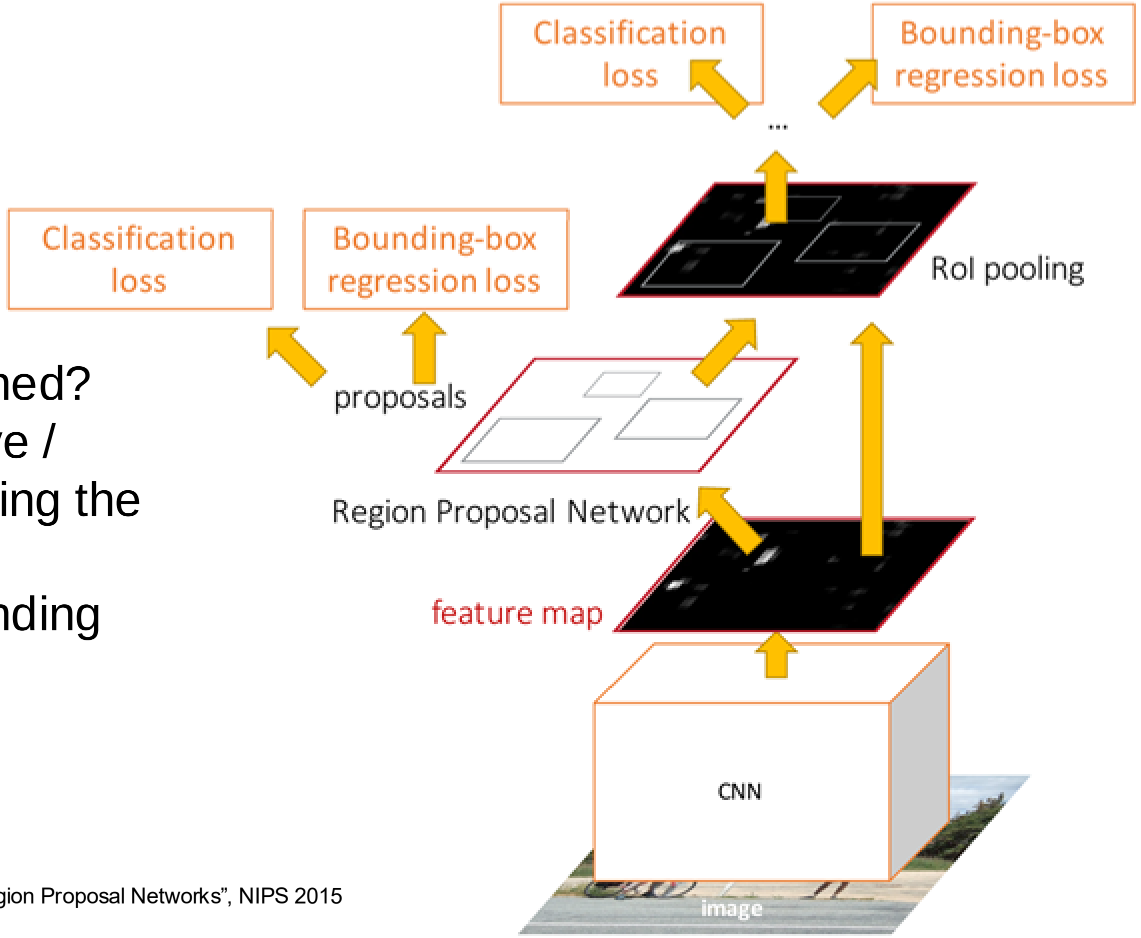
Go back to step 1

Fas<u>**ter**</u> R-CNN:

Make CNN do proposals!

Glossing over many details:

- How are anchors determined?
- How do we sample positive / negative samples for training the RPN?
- How to parameterize bounding box regression?



Ren et al, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, NIPS 2015

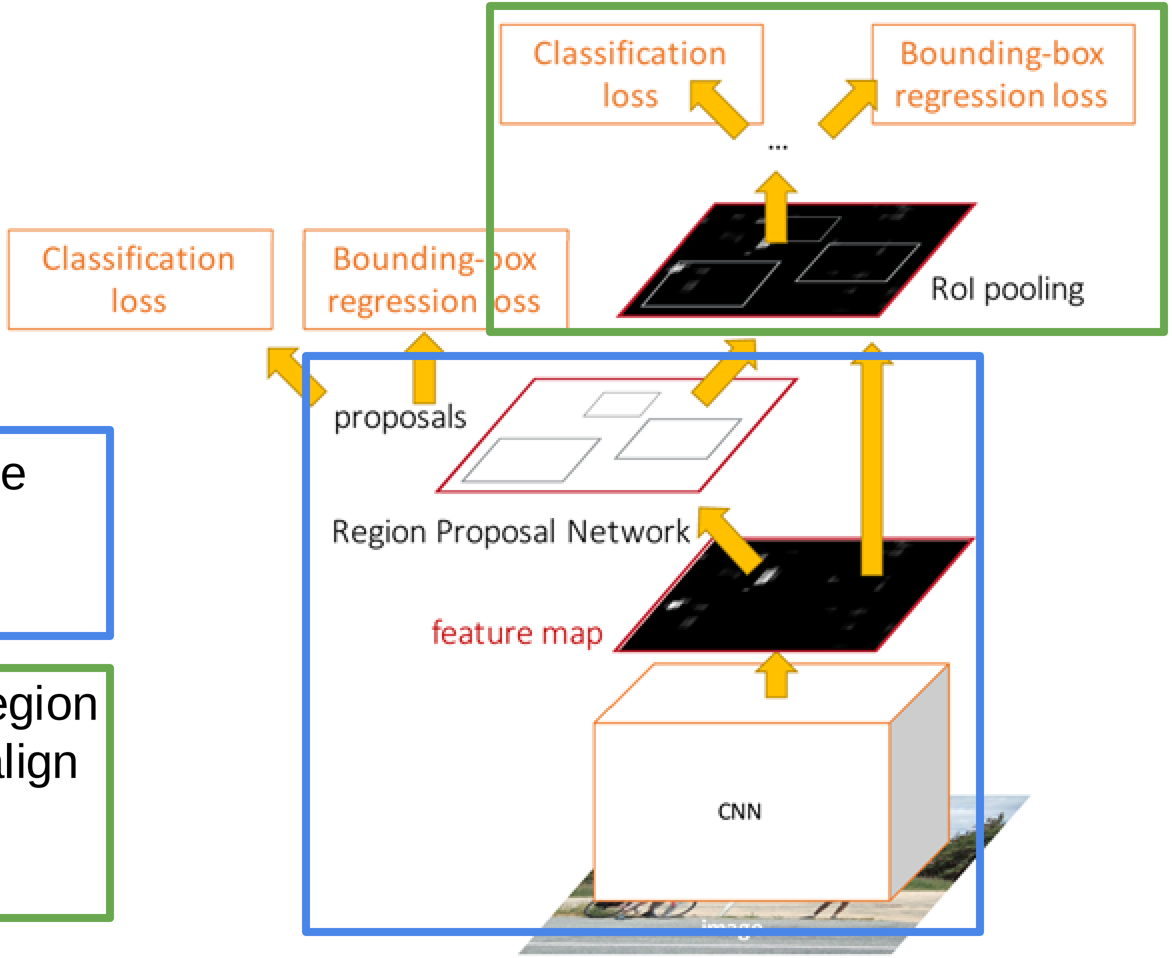
Fast<u>**er**</u> R-CNN:

Make CNN do proposals!

Faster R-CNN is a
Two-stage object detector

First stage: Run once per image
- Backbone network
- Region proposal network

Second stage: Run once per region
- Crop features: RoI pool / align
- Predict object class
- Prediction bbox offset



Faster R-CNN:

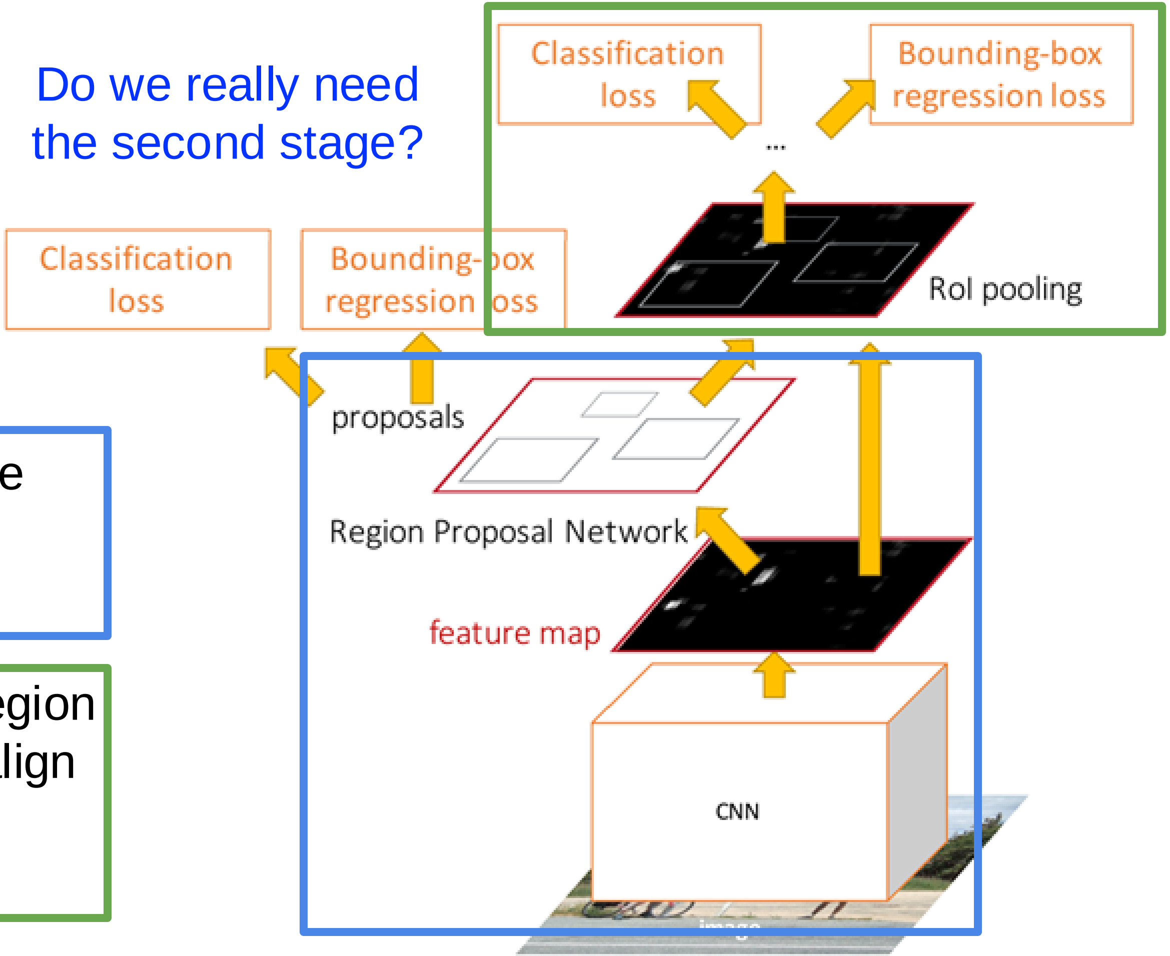
Make CNN do proposals!

Do we really need the second stage?

Faster R-CNN is a
Two-stage object detector

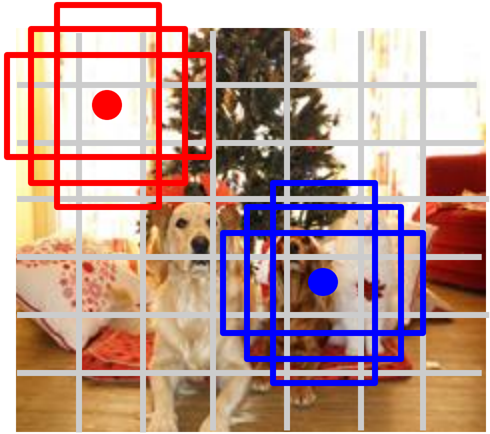
First stage: Run once per image
- Backbone network
- Region proposal network

Second stage: Run once per region
- Crop features: RoI pool / align
- Predict object class
- Prediction bbox offset



Single-Stage Object Detectors: YOLO / SSD / RetinaNet

Input image
3 x H x W



Divide image into grid
7 x 7

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

Within each grid cell:

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

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

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

Object Detection: Lots of variables ...

Backbone Network
VGG16
ResNet-101
Inception V2
Inception V3
Inception ResNet
MobileNet

"Meta-Architecture"
Two-stage: Faster R-CNN
Single-stage: YOLO / SSD
Hybrid: R-FCN

Image Size
 # Region Proposals
...

Takeaways
Faster R-CNN is slower but more accurate

SSD is much faster but not as accurate

Bigger / Deeper backbones work better

Huang et al, "Speed/accuracy trade-offs for modern convolutional object detectors", CVPR 2017
Zou et al, "Object Detection in 20 Years: A Survey", arXiv 2019

R-FCN: Dai et al, "R-FCN: Object Detection via Region-based Fully Convolutional Networks", NIPS 2016
Inception-V2: Ioffe and Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", ICML 2015
Inception V3: Szegedy et al, "Rethinking the Inception Architecture for Computer Vision", arXiv 2016
Inception ResNet: Szegedy et al, "Inception-V4, Inception-ResNet and the Impact of Residual Connections on Learning", arXiv 2016
MobileNet: Howard et al, "Efficient Convolutional Neural Networks for Mobile Vision Applications", arXiv 2017

Instance Segmentation

Classification

CAT

No spatial extent

Semantic Segmentation

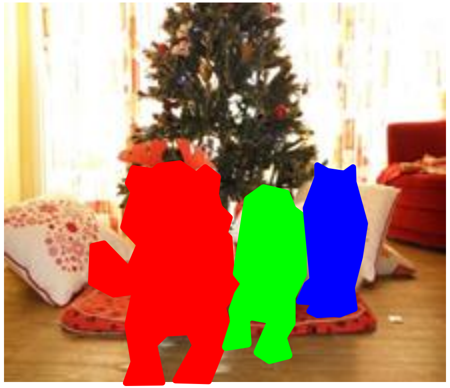
GRASS, CAT, TREE, SKY

No objects, just pixels

Object Detection

DOG, DOG, CAT

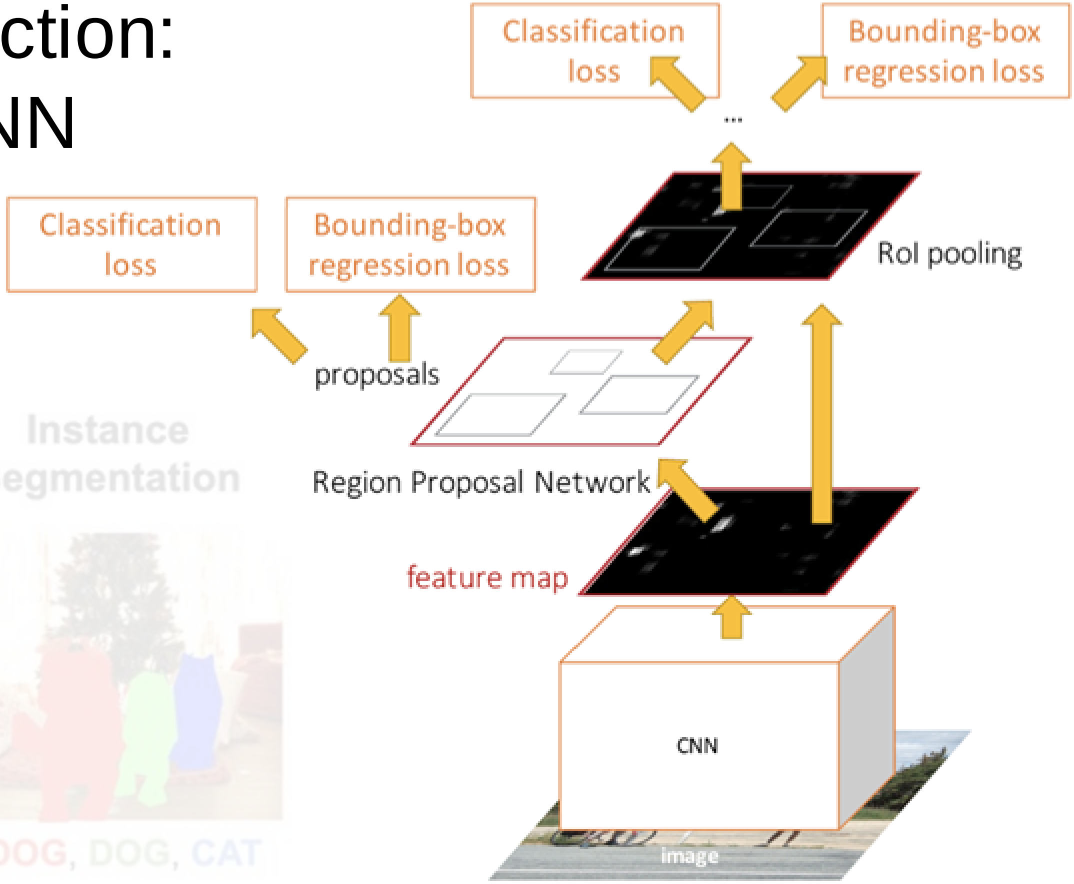
Instance Segmentation



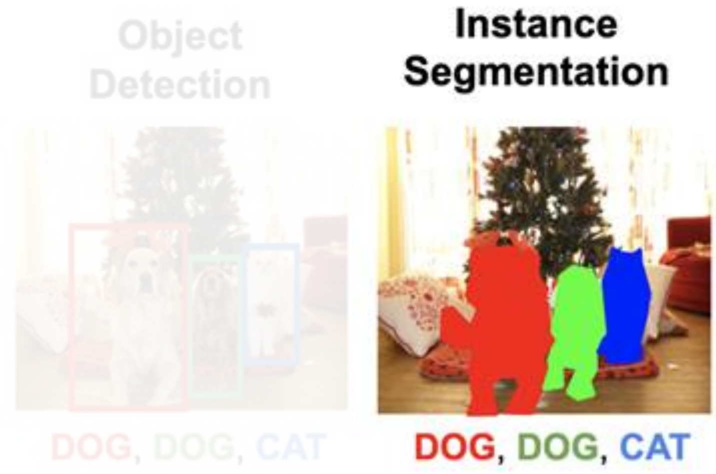
DOG, DOG, CAT

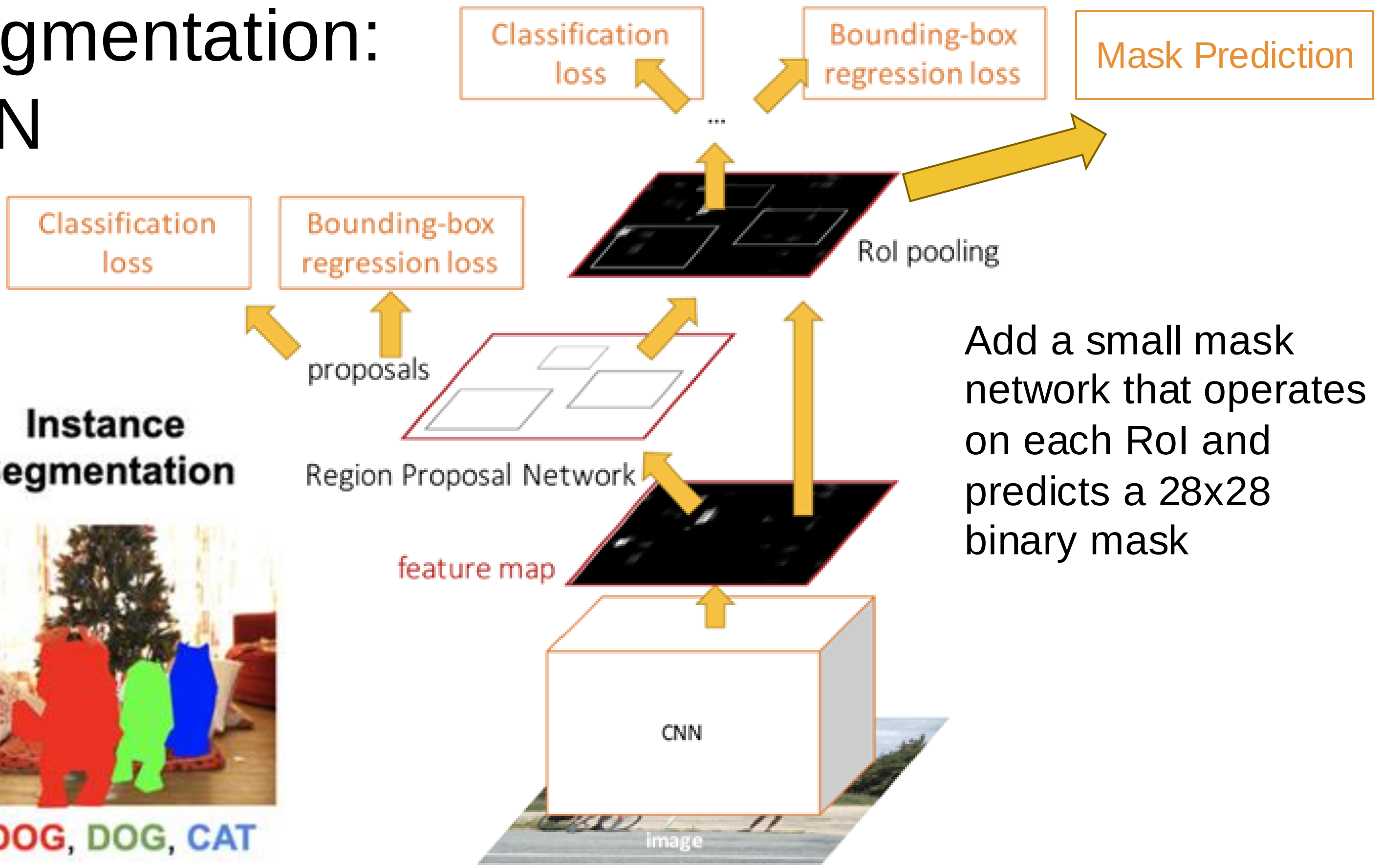
Multiple Object

Object Detection: Faster R-CNN



Instance Segmentation: Mask R-CNN

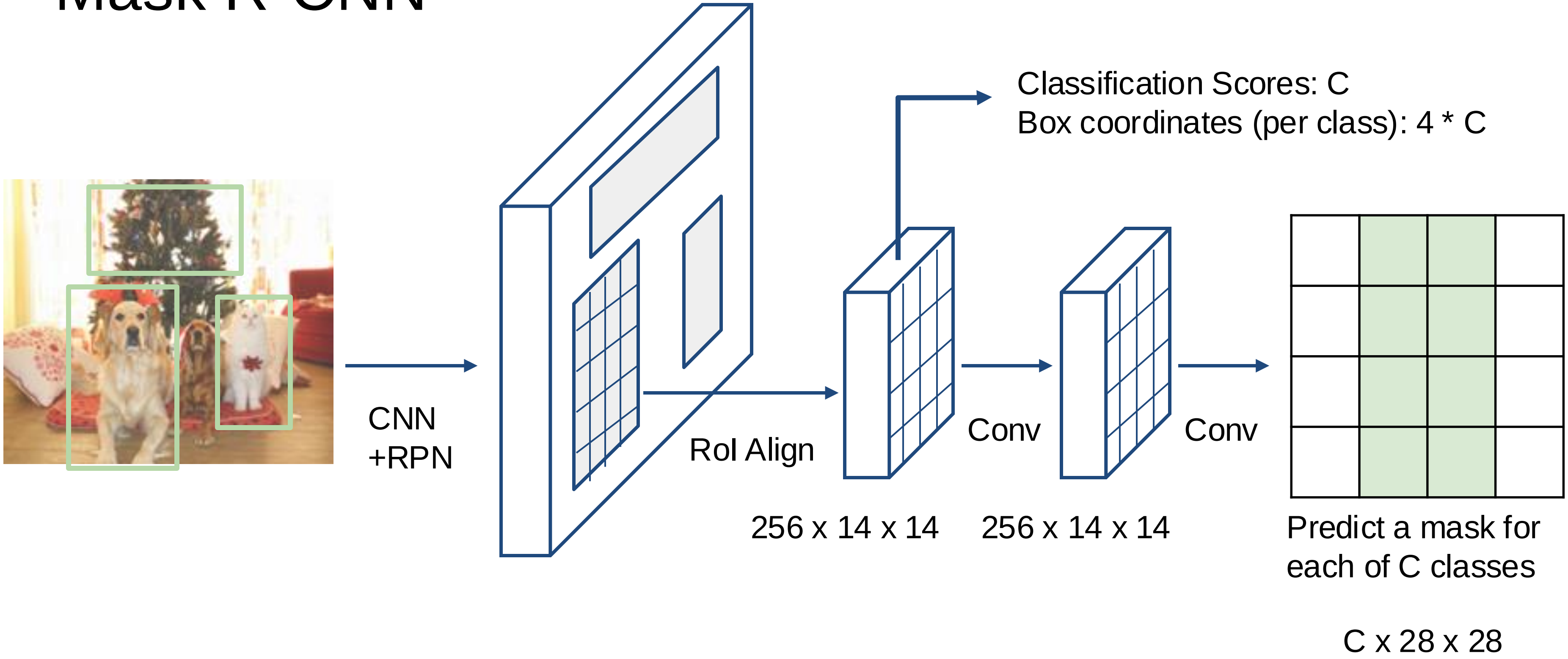




Add a small mask network that operates on each RoI and predicts a 28x28 binary mask

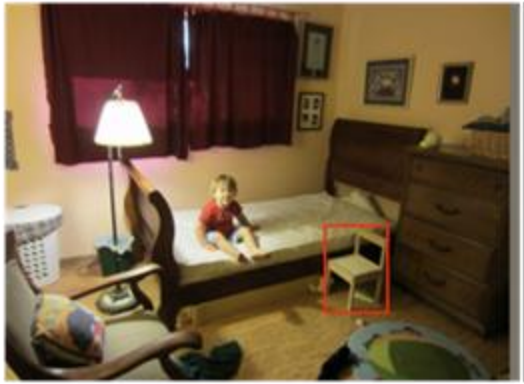
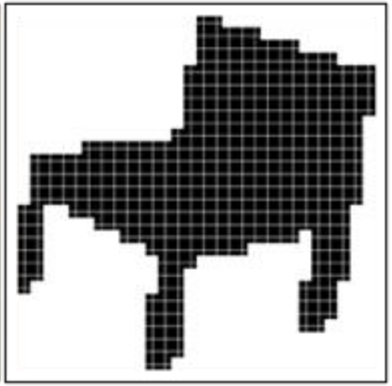
He et al, “Mask R-CNN”, ICCV 2017

Mask R-CNN

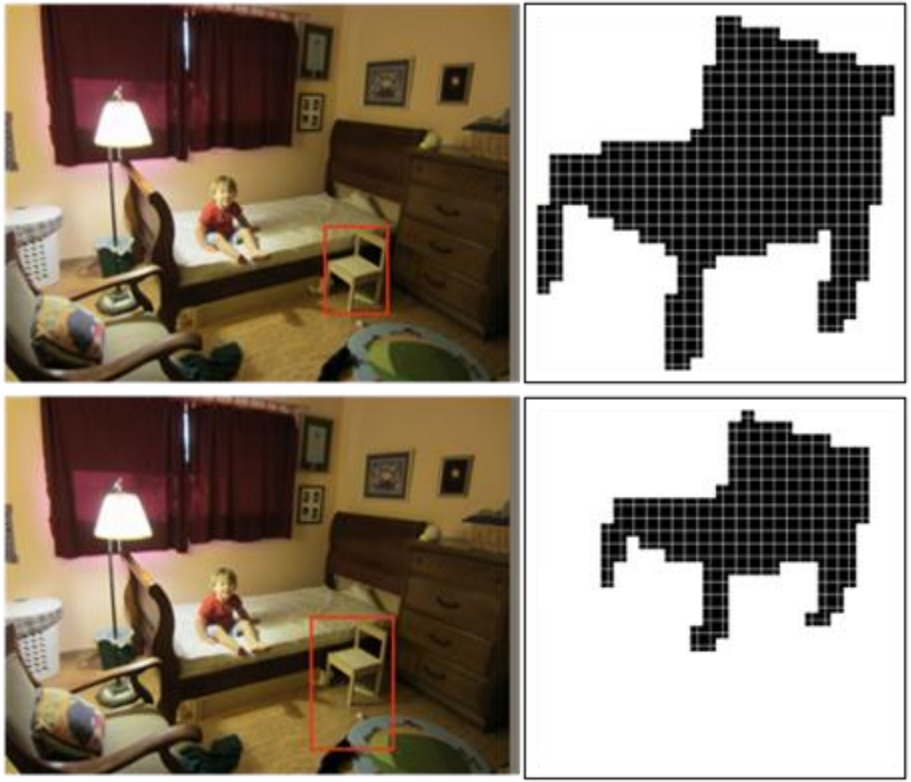


He et al, "Mask R-CNN", arXiv 2017

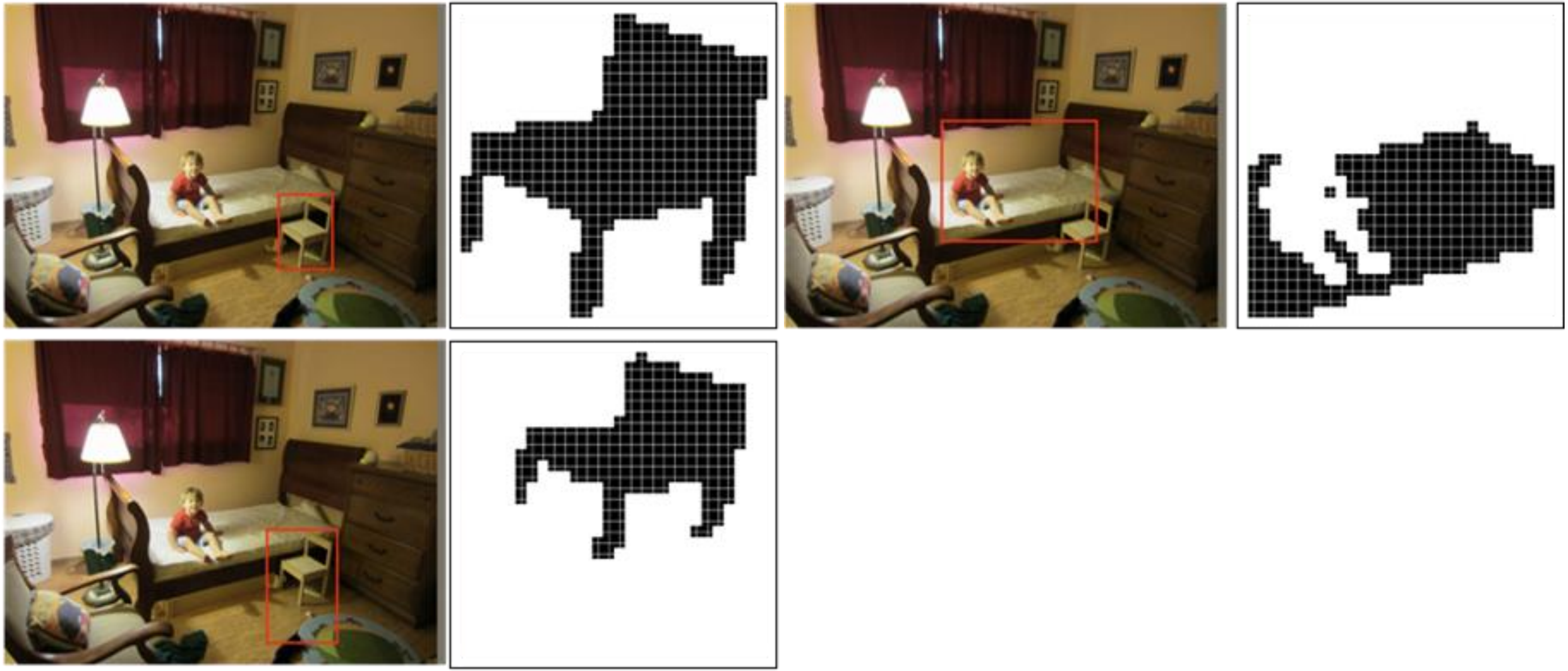
Mask R-CNN: Example Mask Training Targets

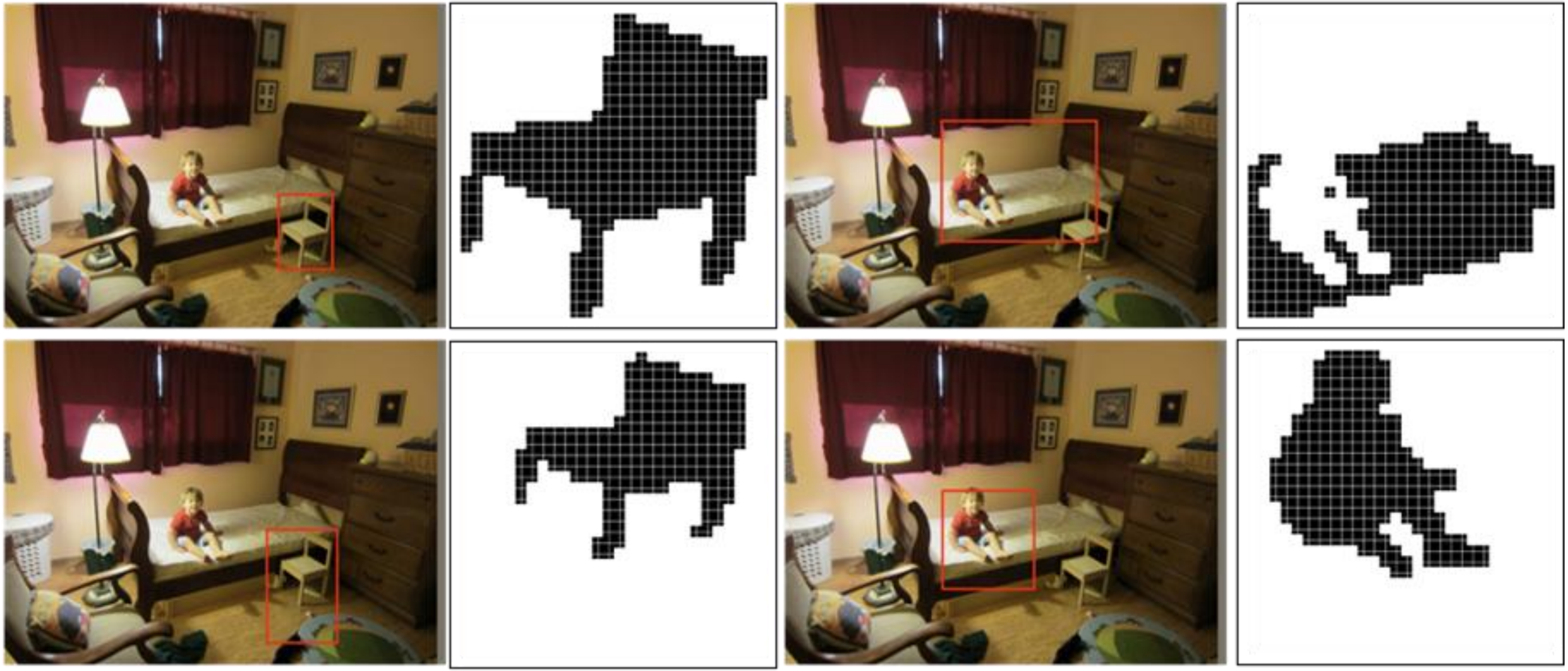
Mask R-CNN: Example Mask Training Targets



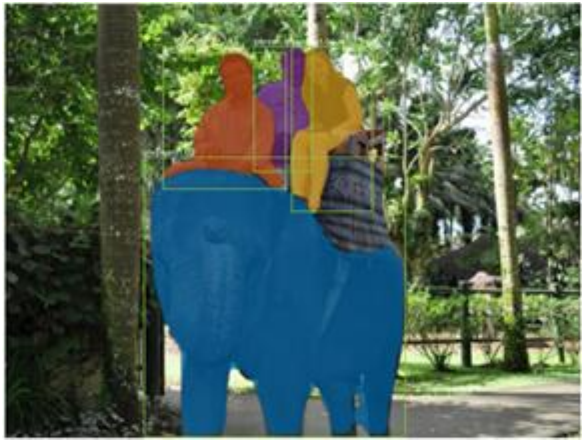
Mask R-CNN: Example Mask Training Targets



Mask R-CNN: Example Mask Training Targets



Mask R-CNN: Very Good Results!



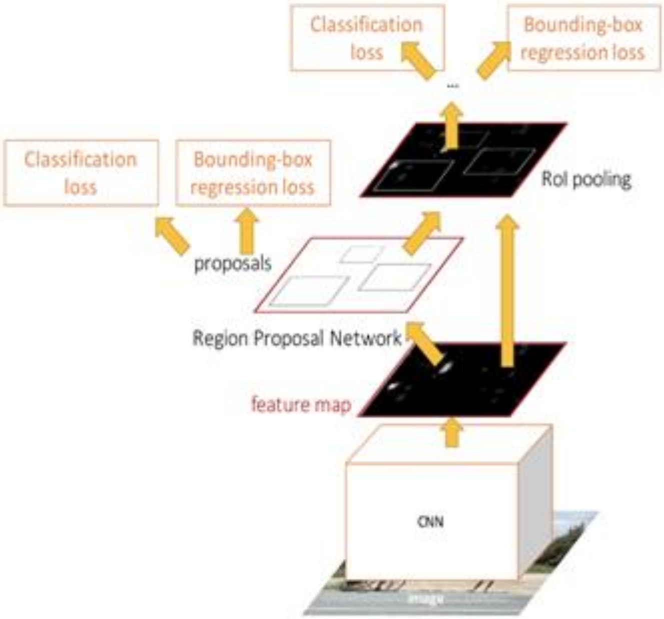
He et al, “Mask R-CNN”, ICCV 2017

Mask R-CNN
Also does pose

He et al, “Mask R-CNN”, ICCV 2017

RCNN Series

- **R-CNN**: Per-region detection, hand-crafted region proposal
- **Fast R-CNN**: Shared feature extraction, RoI Pooling, Anchors
- **Faster R-CNN**: Region Proposal Networks, RoI Align
- **Mask R-CNN**: Instance Segmentation



Detectors are becoming more complex!
Many hyperparameters to tune for each components …
Can we simplify it?

End-to-End Object Detection with Transformers

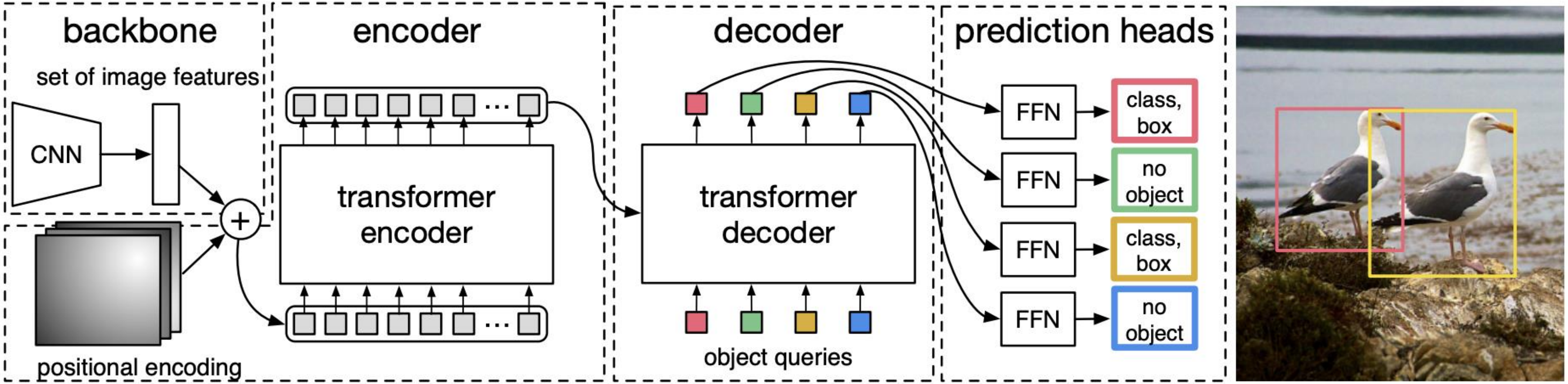
Nicolas Carion*, Francisco Massa*, Gabriel Synnaeve, Nicolas Usunier,
Alexander Kirillov, and Sergey Zagoruyko

Facebook AI

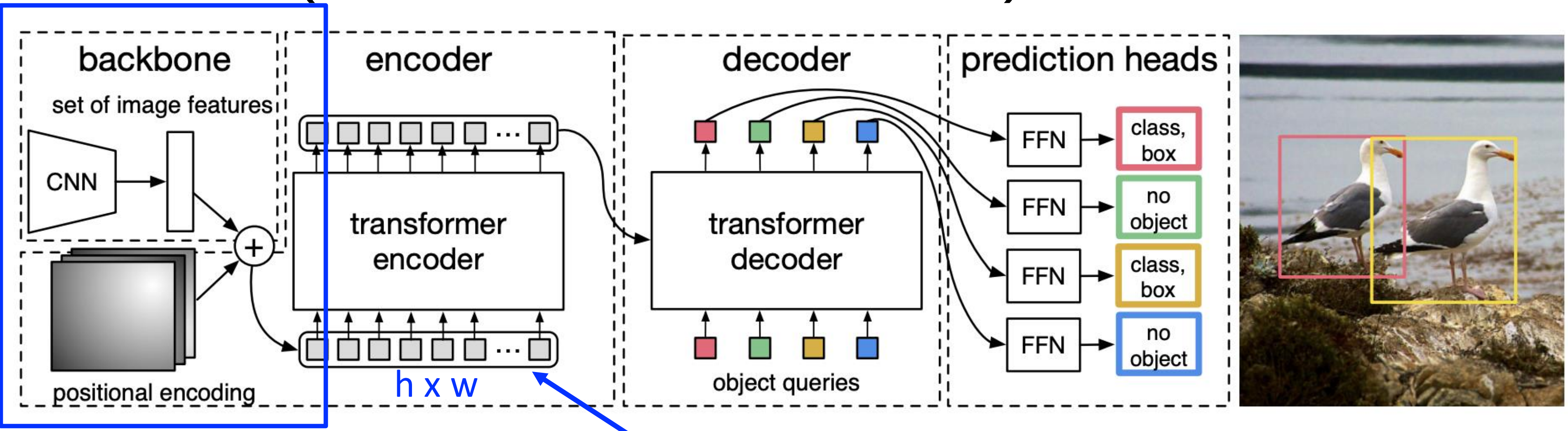
Key ideas:

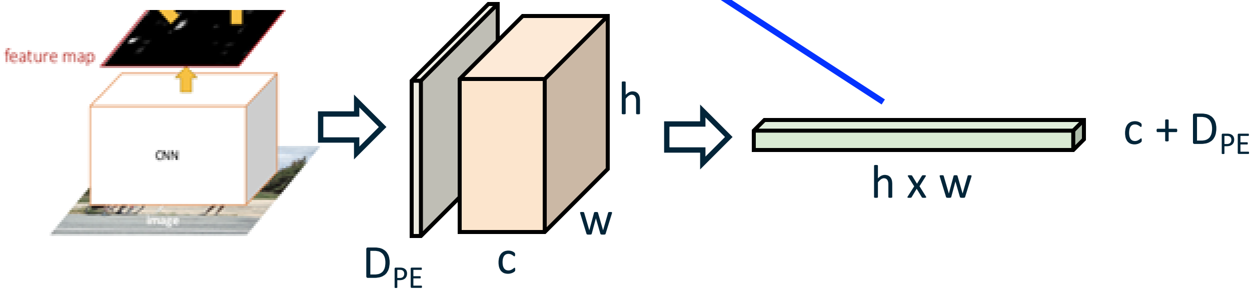
- Detection as a **set-to-set prediction** problem
- Use Transformer to model the detection problem

DETR (DEtection TRansformer)

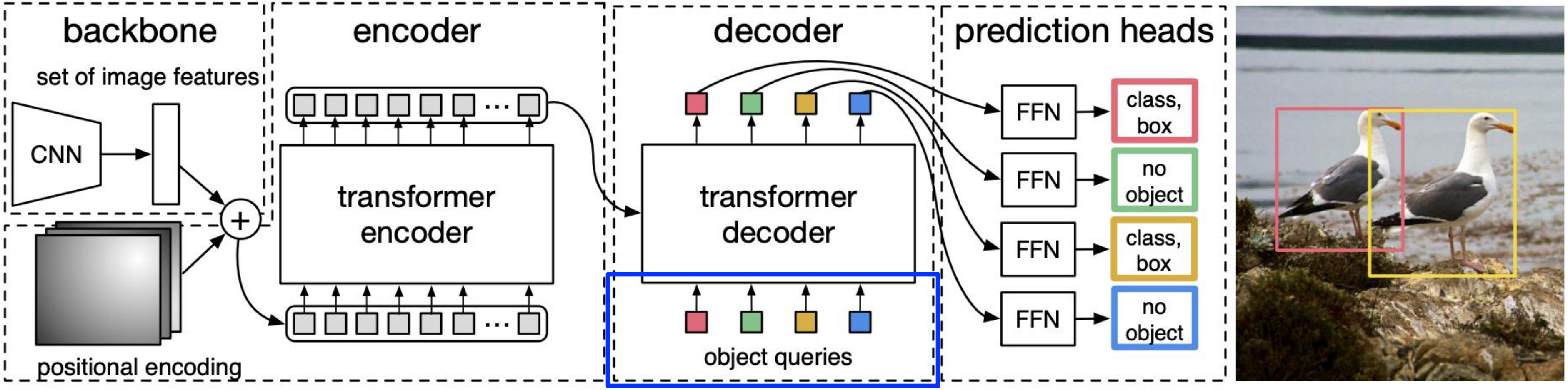


DETR (DEtection TRansformer)



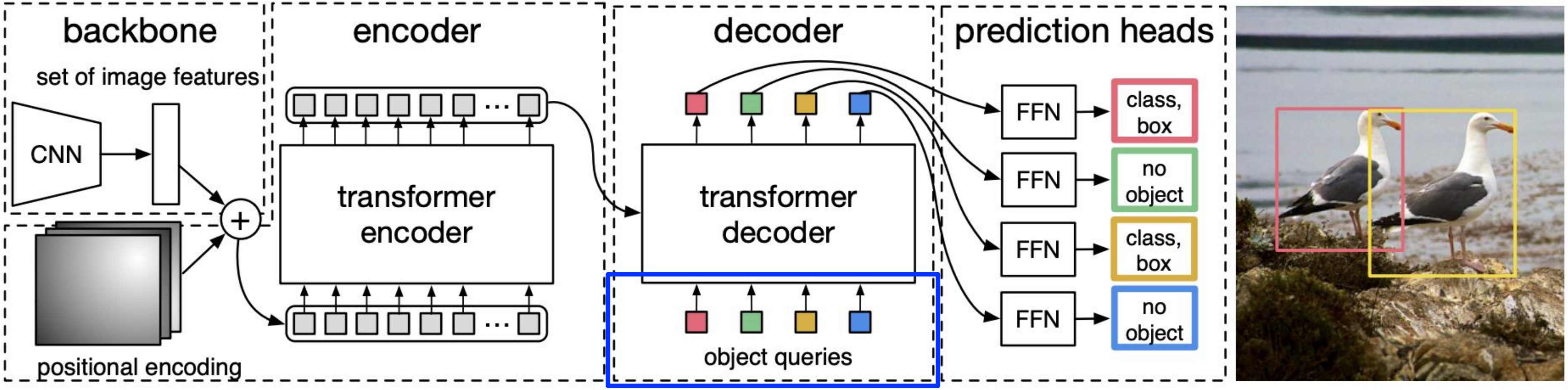


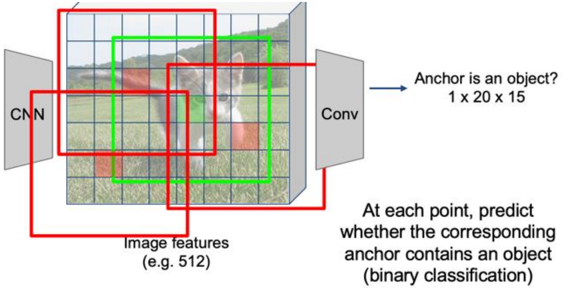
DETR (DEtection TRansformer)



A fixed set of learnable embeddings,
e.g., 300 size-N vectors
Q: Why?

DETR (DEtection TRansformer)





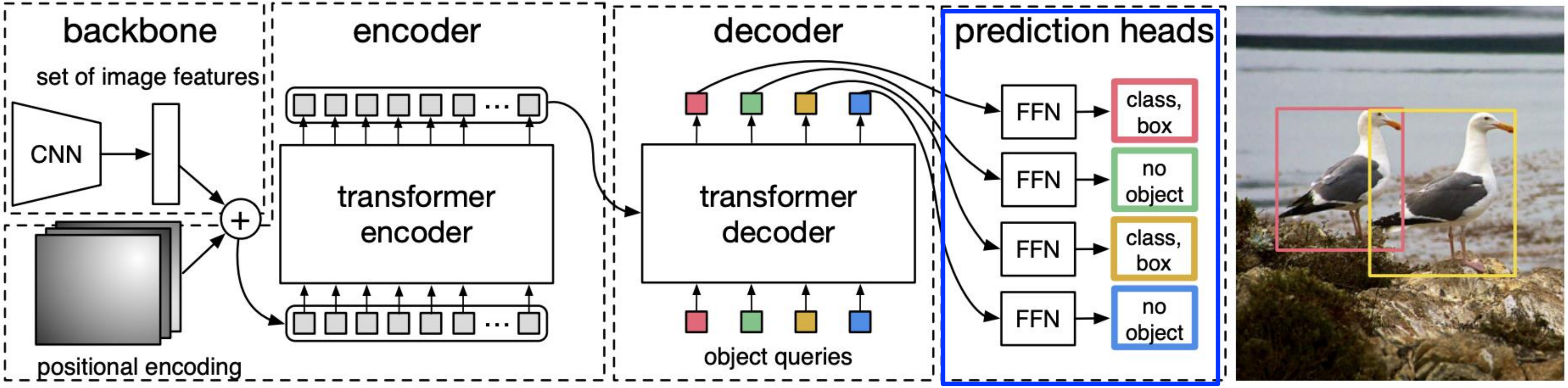
A fixed set of learnable embeddings, e.g., 300 size-N vectors

Q: Why?

A: Break the symmetry of predictions, so that each prediction is different.

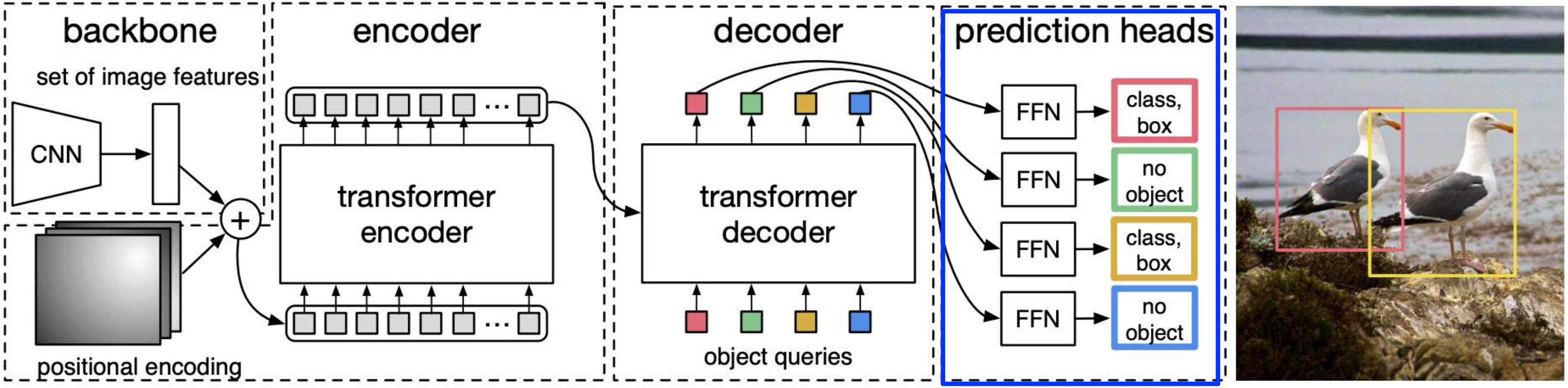
Analogous to anchors in *R-CNN, but **no spatial location**

DETR (DEtection TRansformer)



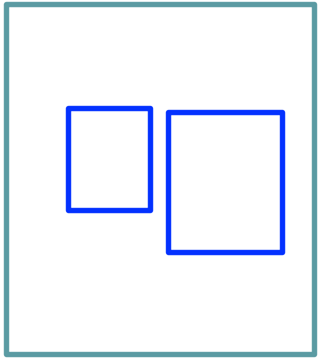
Problem: We don’t know which query corresponds to which ground truth during training! We can’t predetermine a fixed order like in sequence decoding.

DETR (DEtection TRansformer)

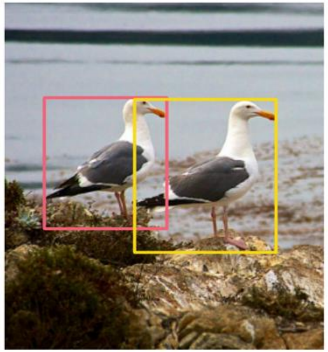


Problem: We don’t know which query corresponds to which ground truth during training! We can’t predetermine a fixed order like in sequence decoding.
Solution: Set matching loss --- train your model to generate a set of predictions that matches ground truth regardless of its order.

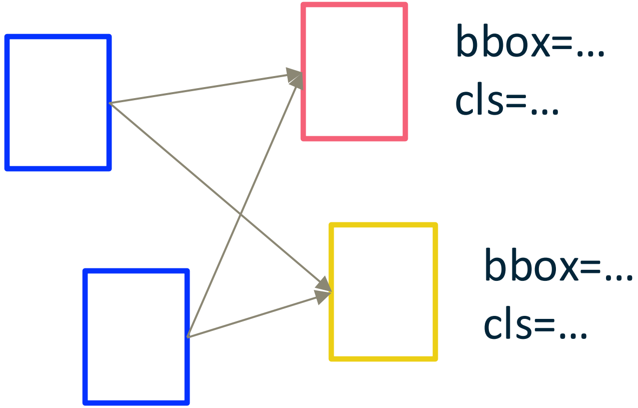
Hangarian Loss (Set Matching Loss)



Prediction



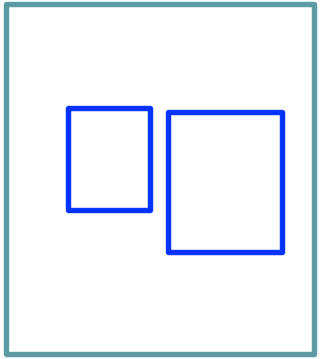
Ground Truth



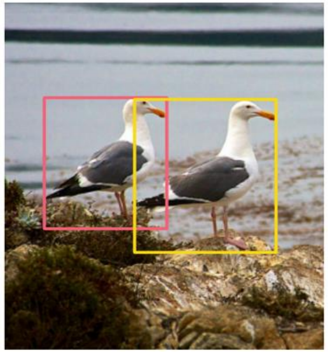
Goal: minimize bipartite distance

Problem: each query should be trained to match one ground truth. We don’t know the matching!

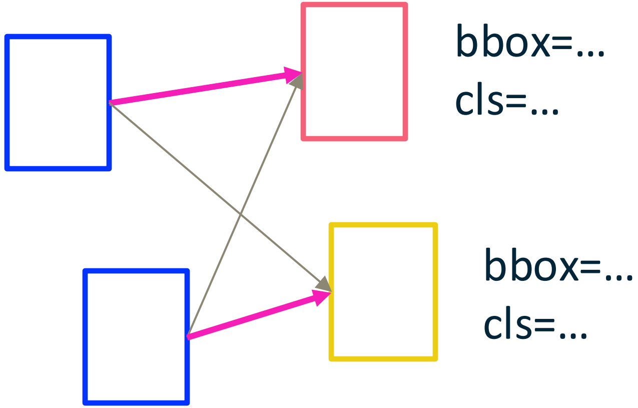
Hangarian Loss (Set Matching Loss)



Prediction



Ground Truth



Goal: minimize bipartite distance

1. **Hungarian matching:** find the minimum-loss bipartite matching between prediction and ground truth given the current prediction.
2. **Minimize matched loss:** Given the matched prediction and ground truth, minimize the detection loss (bounding box distance and classification CE loss)

DETR vs. FasterRCNN

Model	GFLOPS/FPS	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster RCNN-FPN+	180/26	42M	42.0	62.1	45.5	26.6	45.4	53.4
Faster RCNN-R101-FPN+	246/20	60M	44.0	63.9	**47.8**	**27.2**	48.1	56.0
DETR	86/28	41M	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5	187/12	41M	43.3	63.1	45.9	22.5	47.3	61.1
DETR-R101	152/20	60M	43.5	63.8	46.4	21.9	48.0	61.8
DETR-DC5-R101	253/10	60M	**44.9**	**64.7**	47.7	23.7	**49.5**	**62.3**

Similar size, simpler, and (mostly) better!

We are still detecting a fixed number of object with finite vocabulary …

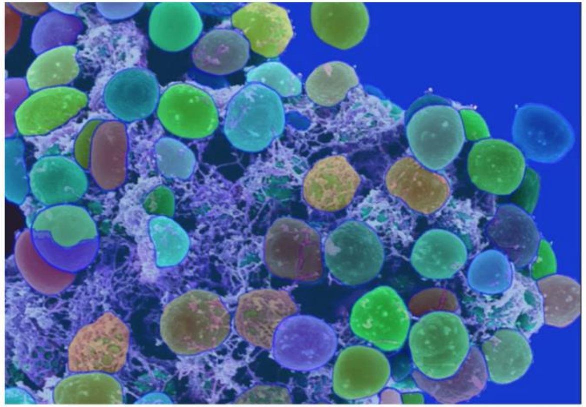
Segment Anything

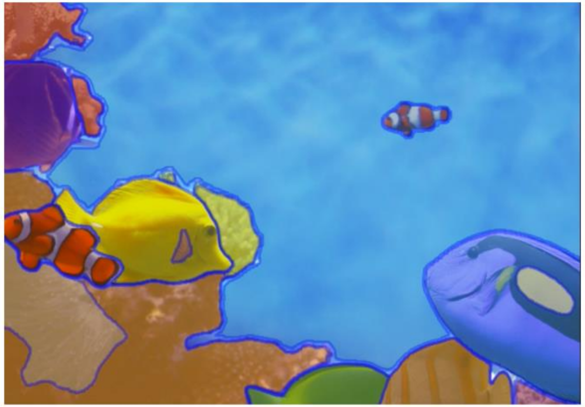
Alexander Kirillov[1,2,4] Eric Mintun[2] Nikhila Ravi[1,2] Hanzi Mao[2] Chloe Rolland[3] Laura Gustafson[3]
Tete Xiao[3] Spencer Whitehead Alexander C. Berg Wan-Yen Lo Piotr Dollár[4] Ross Girshick[4]
[1]project lead [2]joint first author [3]equal contribution [4]directional lead
Meta AI Research, FAIR

Key ideas:

- Query-based prediction instead of fixed set-to-set prediction
- Large-scale training data with auto-labeling

Foundation Image Segmentation Models





SegmentAnything (Meta AI, 2023)

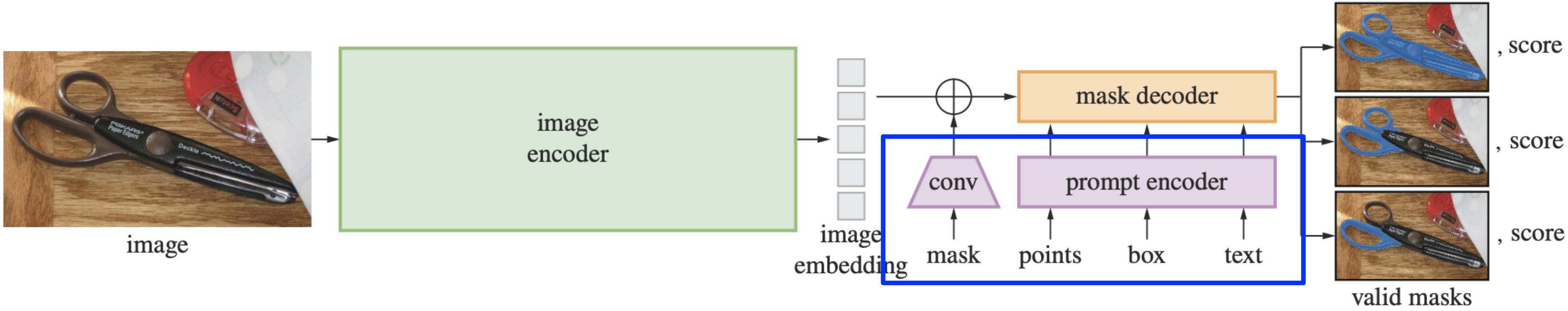
Try it yourself! https://segment-anything.com/demo#

Foundation Image Segmentation Models

SegmentAnything (Meta AI, 2023)

Try it yourself! https://segment-anything.com/demo#

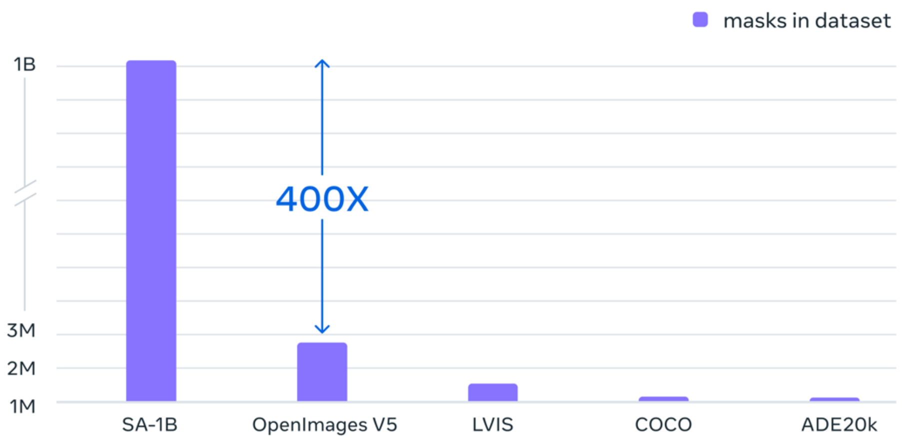
Foundation Image Segmentation Models

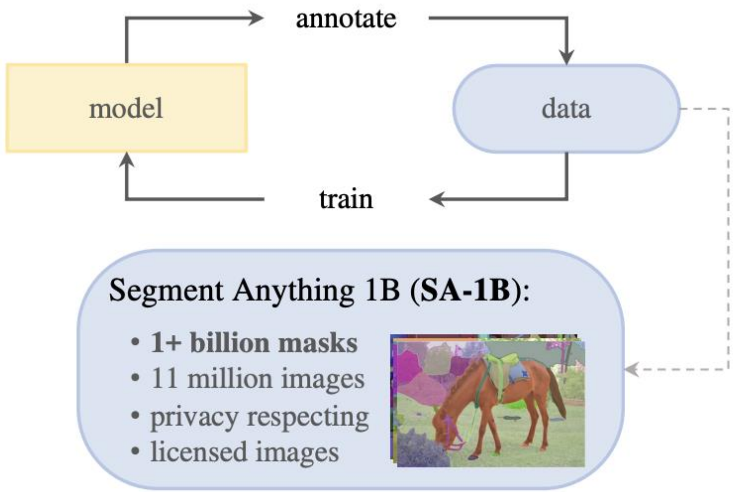


No more learned embeddings. Query anything you want!

SegmentAnything (Meta AI, 2023)

Foundation Image Segmentation Models





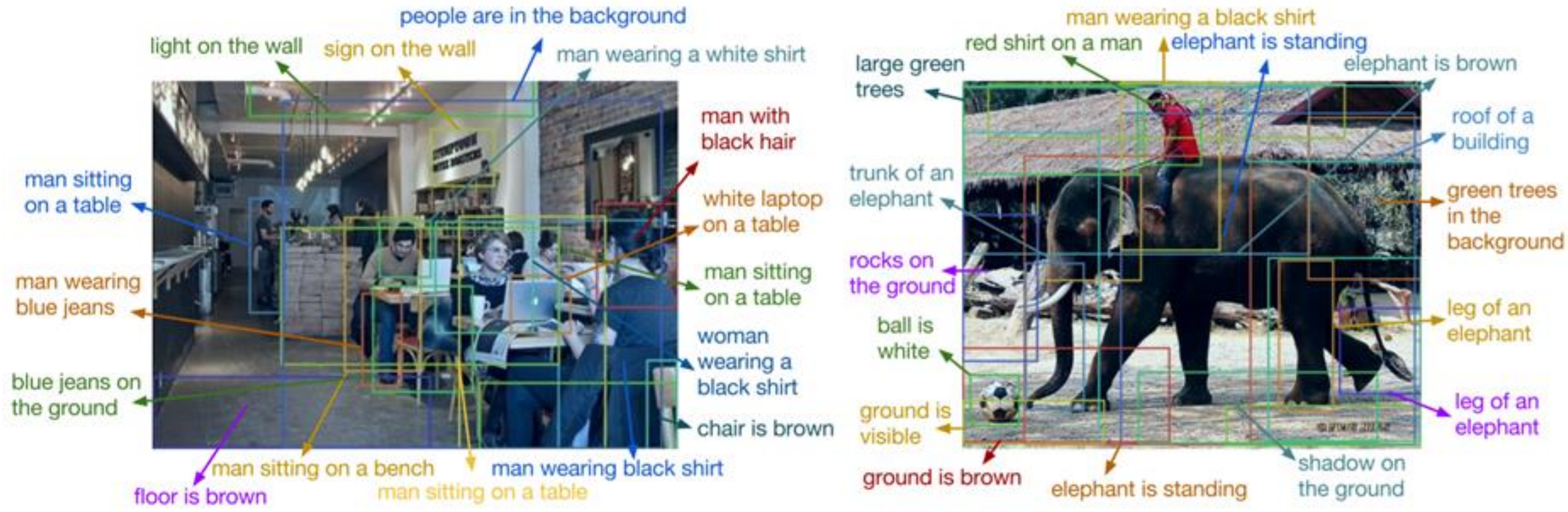
SegmentAnything (Meta AI, 2023)

Summary: Segmentation and Detection

- Object segmentation and detection are most common application of computer vision research.
- They have driven decades of advancement in Autonomous Vehicles, Robotics, traffic analytics, and basically any devices that have camera and adequate computing power (e.g., Smart Phone)
- Segmentation and Detection with DNNs evolved through similar paths (sliding window, feature sharing, input-output, alignment etc.)
- We have a new wave of foundation detection and segmentation models driven by Transformer + ConvNet + large dataset

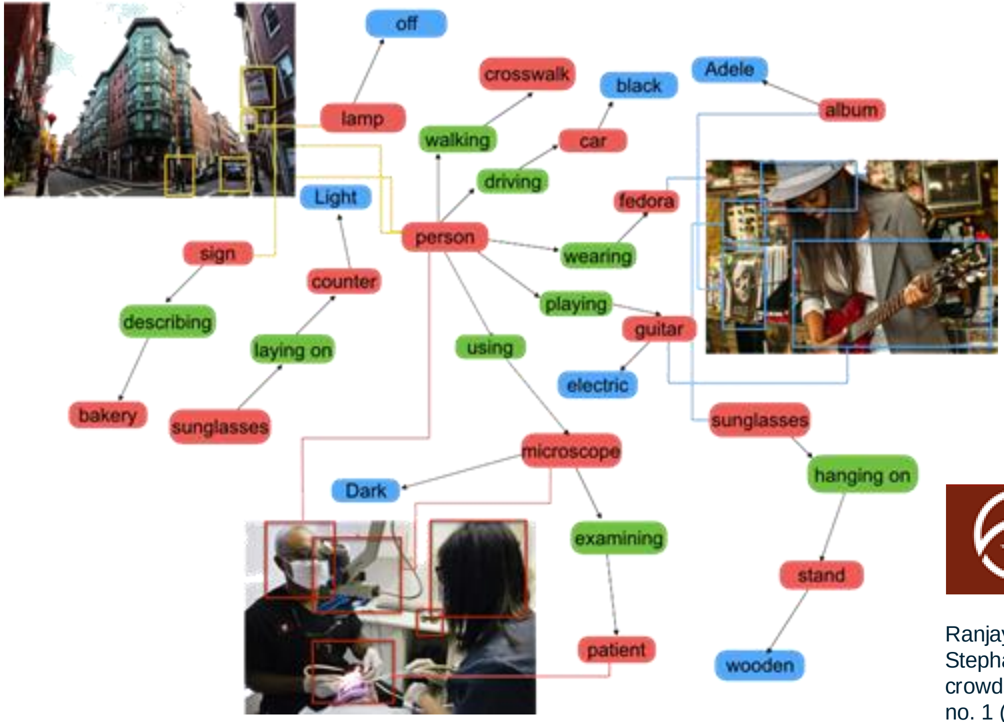
Beyond 2D Object Detection...

Object Detection + Captioning = Dense Captioning



Johnson, Karpathy, and Fei-Fei, "DenseCap: Fully Convolutional Localization Networks for Dense Captioning", CVPR 2016

Objects + Relationships = Scene Graphs

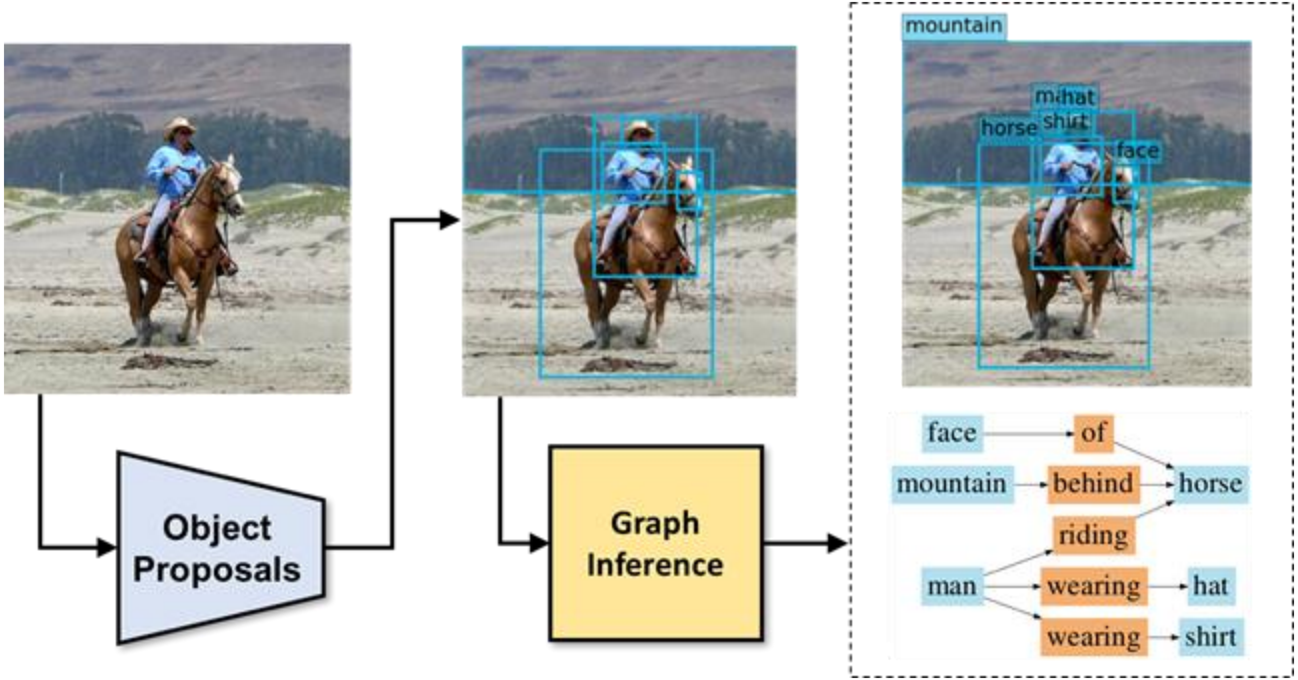


108,077 Images
5.4 Million Region Descriptions
1.7 Million Visual Question Answers
3.8 Million Object Instances
2.8 Million Attributes
2.3 Million Relationships
Everything Mapped to Wordnet Synsets

VISUALGENOME

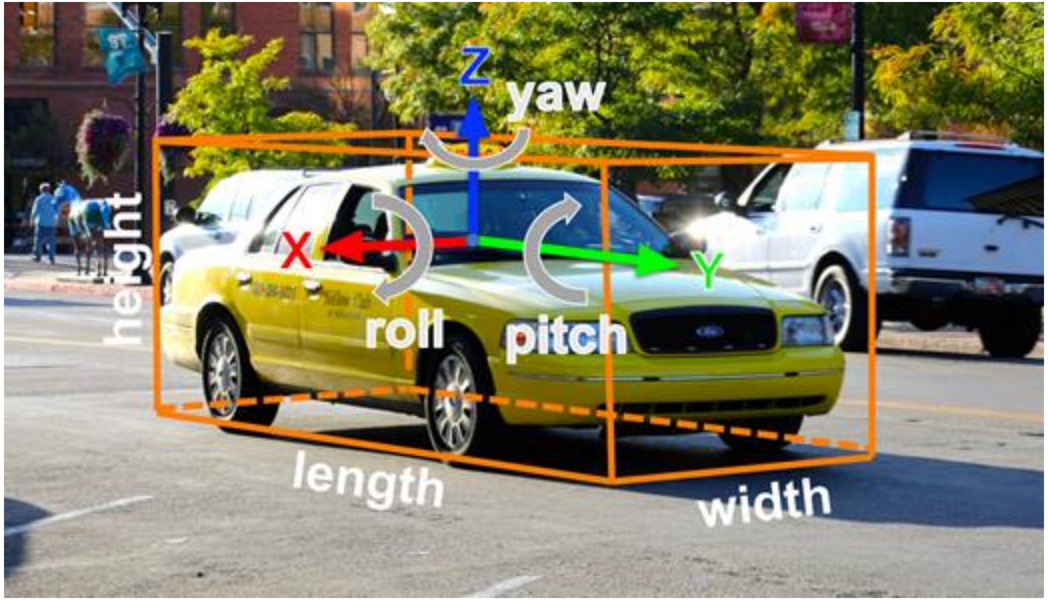
Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen et al. "Visual genome: Connecting language and vision using crowdsourced dense image annotations." International Journal of Computer Vision 123, no. 1 (2017): 32-73.

Scene Graph Prediction



Xu, Zhu, Choy, and Fei-Fei, “Scene Graph Generation by Iterative Message Passing”, CVPR 2017
Figure copyright IEEE, 2018. Reproduced for educational purposes.

3D Object Detection



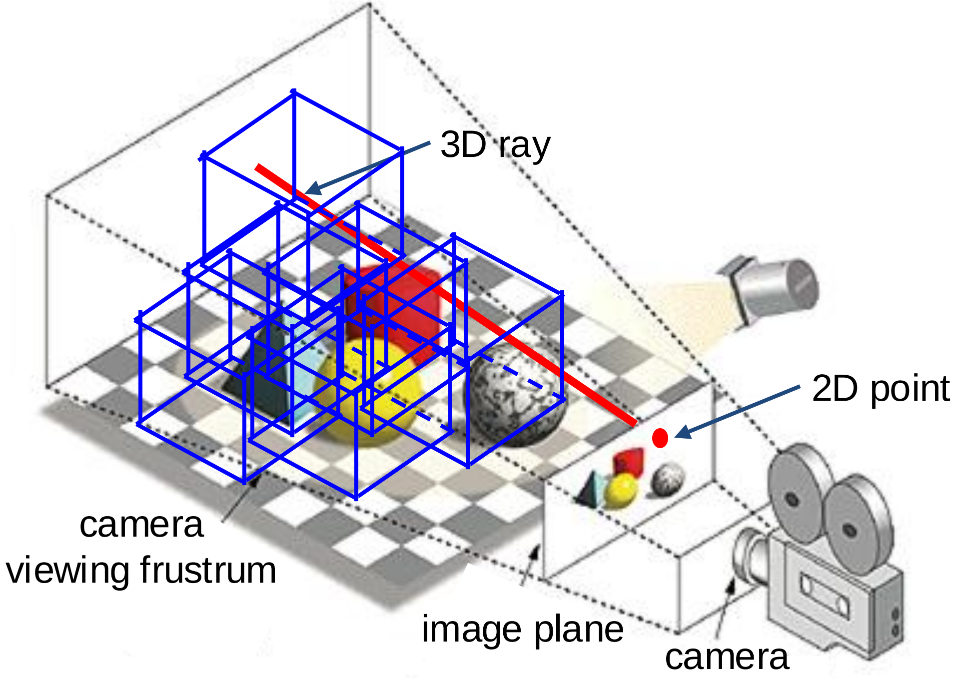
2D Object Detection:
2D bounding box
(x, y, w, h)

3D Object Detection:
3D oriented bounding box
(x, y, z, w, h, l, r, p, y)

Simplified bbox: no roll & pitch

Much harder problem than 2D object detection!

3D Object Detection: Simple Camera Model



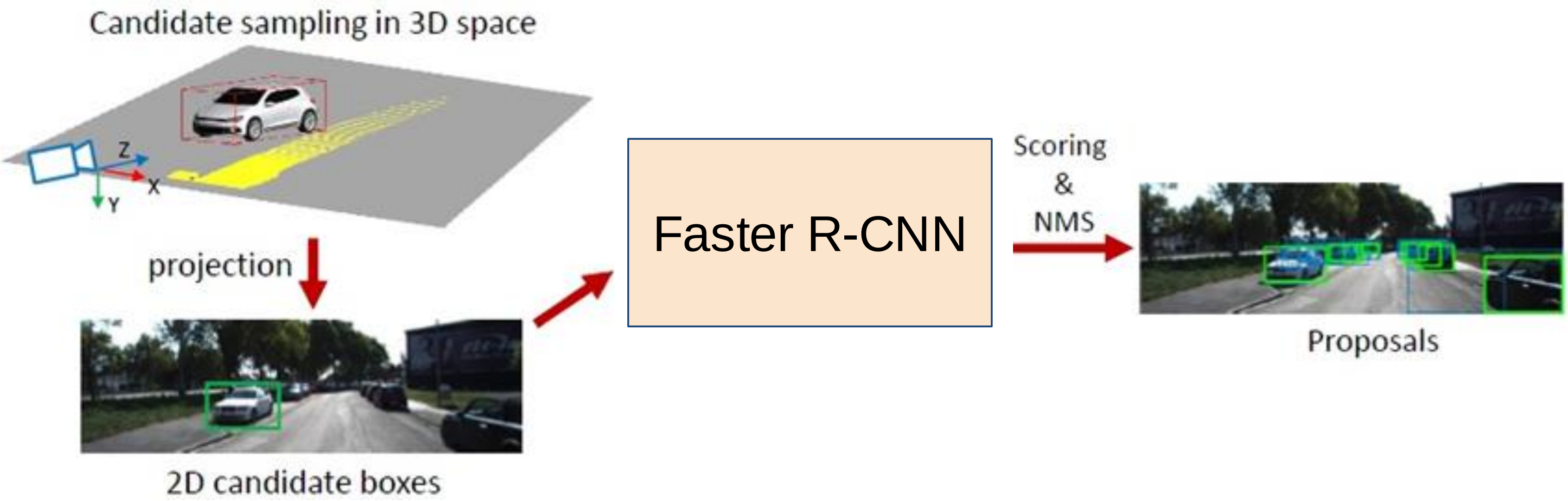
A point on the image plane corresponds to a **ray** in the 3D space

A 2D bounding box on an image is a **frustrum** in the 3D space

Localize an object in 3D:
The object can be anywhere in the **camera viewing frustrum**!

Image source: https://www.pcmag.com/encyclopedia_images/_FRUSTUM.GIF

3D Object Detection: Monocular Camera



- Same idea as Faster RCNN, but proposals are in 3D
- 3D bounding box proposal, regress 3D box parameters + class score

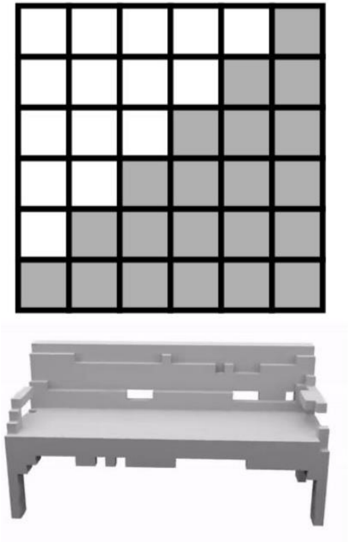
Chen, Xiaozhi, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. "Monocular 3d object detection for autonomous driving." CVPR 2016.

How to Represent 3D Data?

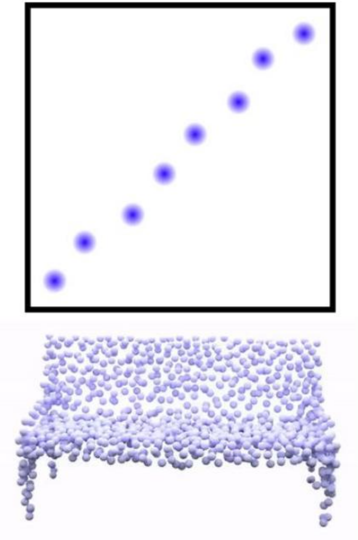
```
[[105 112 100 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  90 102 106 104  79  90 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 120 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  90 102  99  96  93 101  94]
 [106  91  61  64  69  95  80  85 101 107 109  90  75  84  96  95]
 [114 100  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [120 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  99  94]
 [130 120 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

?

3D Representations

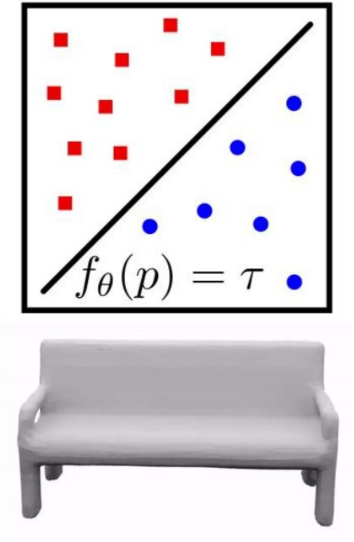


Occupancy Grid
[h, w, l]



Point Cloud
[num_pts, 3]

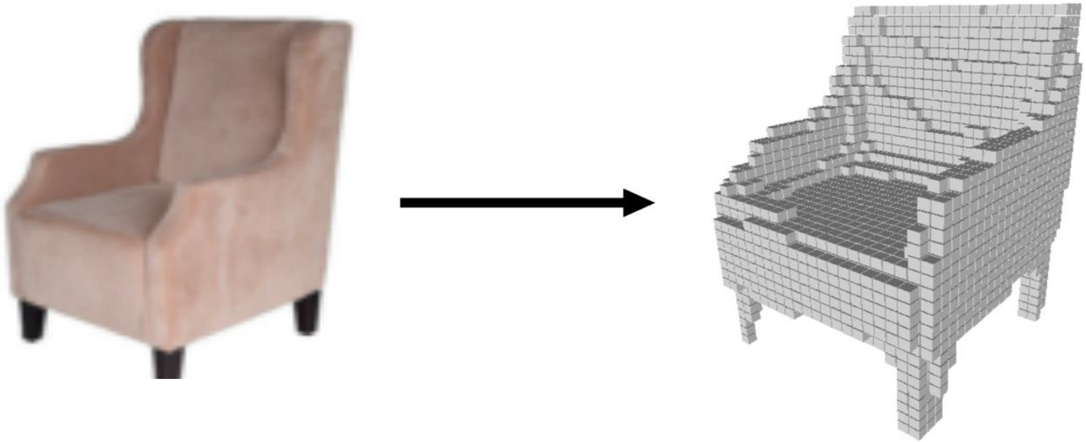
Surface Mesh
(edge list, face list, vertex list)



Implicit Functions
(x, y, z -> d)

Figure credit: Autonomous Vision Group

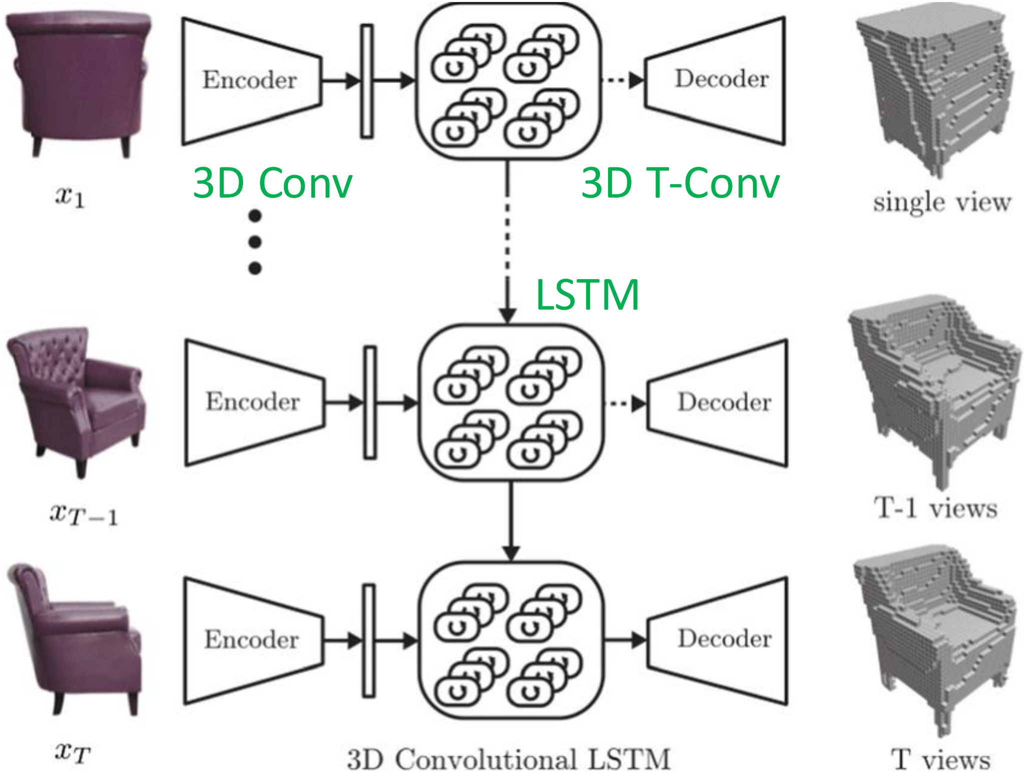
3D Occupancy Grid



Represent the “occupancy” of objects in 3D space with a 3D voxel grid

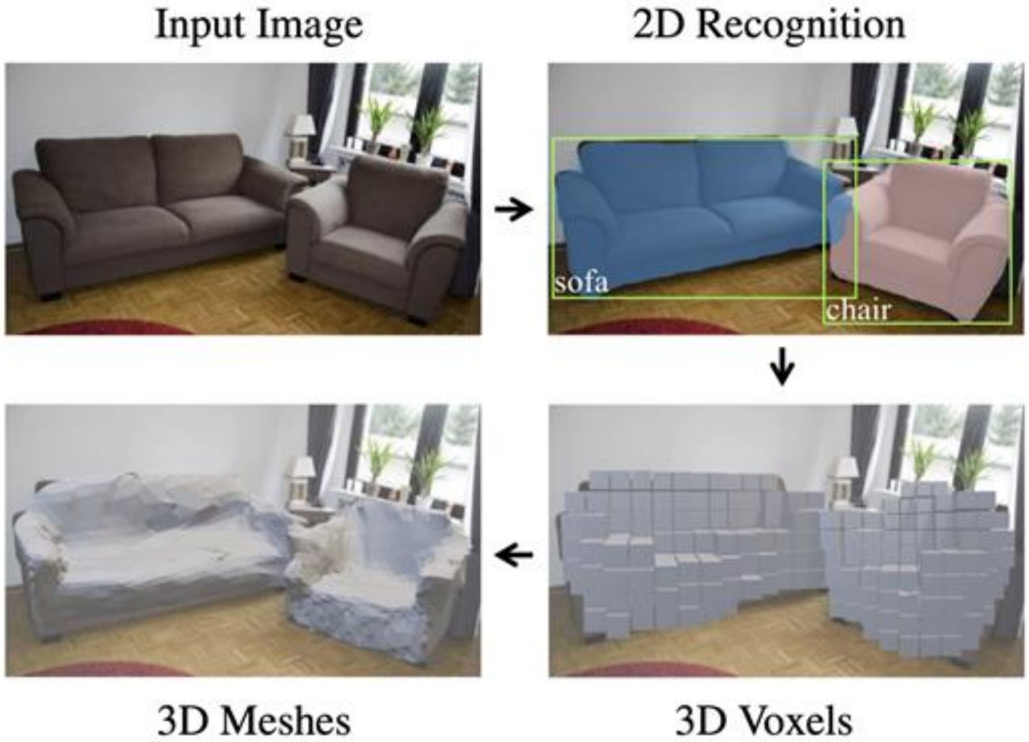
- $V \in \{0, 1\}^{[H,W,L]}$
- Just like segmentation in Masked-RCNN, but in 3D!
- Conceptually simple
- Not trivial to scale to high-resolution shapes

Predicting 3D Voxel Grid with 3D ConvNet



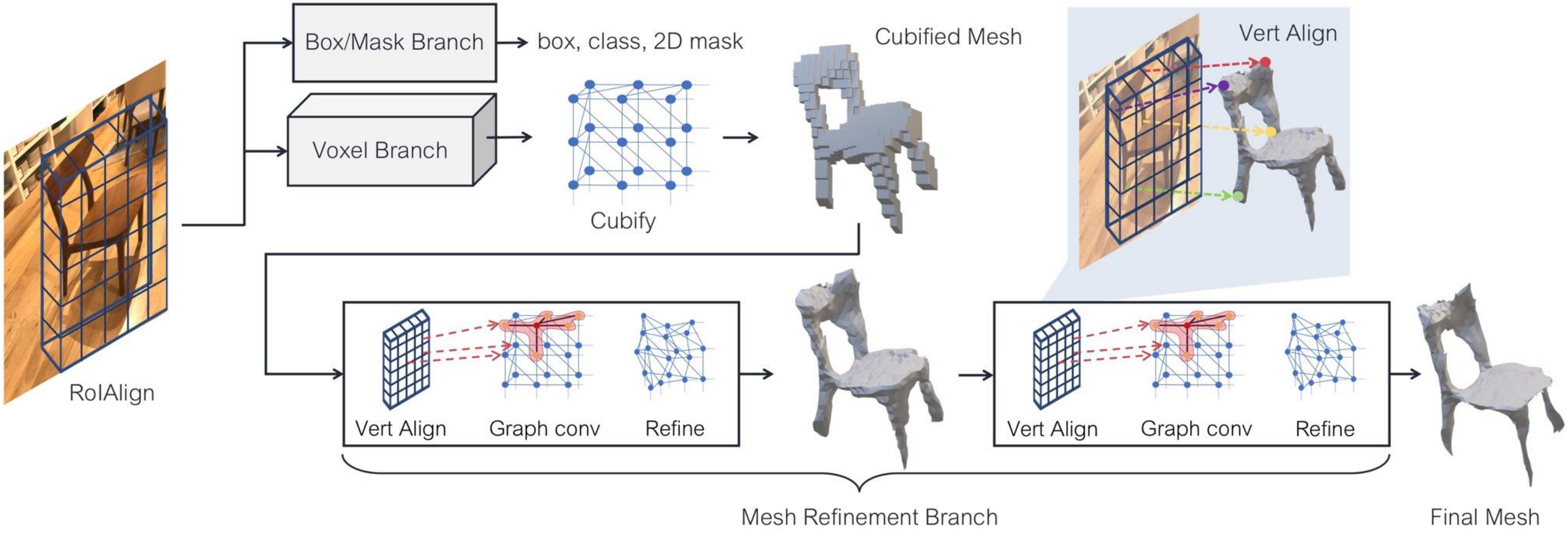
Cho et al. 2016, 3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction

Detection + Reconstruction: Mesh R-CNN



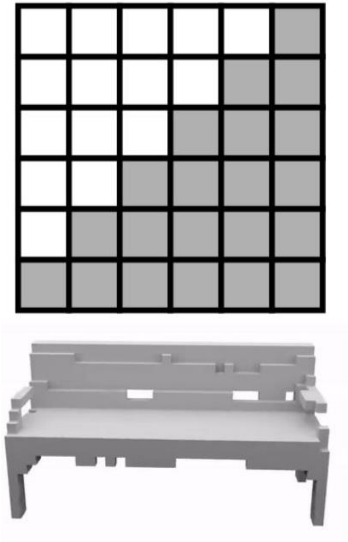
Gkioxari et al., Mesh RCNN, ICCV 2019

Detection + Reconstruction: Mesh R-CNN

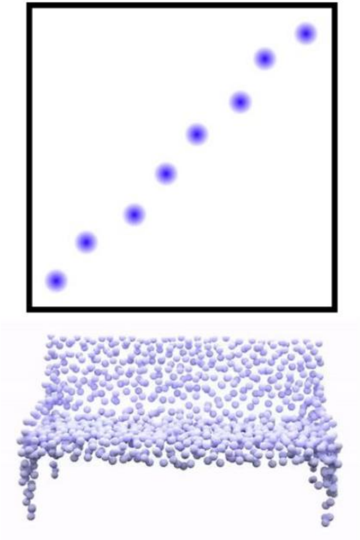


Gkioxari et al., Mesh RCNN, ICCV 2019

3D Representations

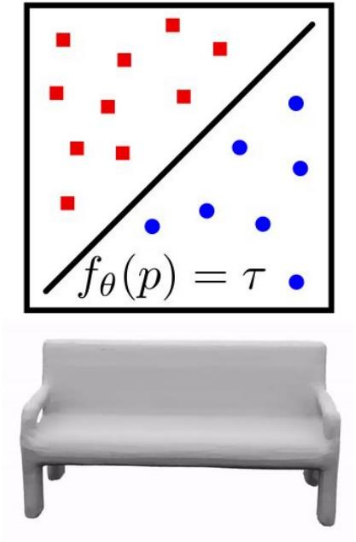


Occupancy Grid
[h, w, l]



Point Cloud
[num_pts, 3]

Surface Mesh
(edge list, face list, vertex list)

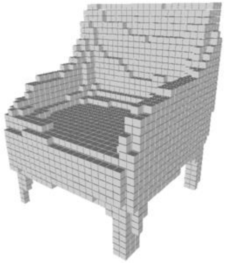


Implicit Functions
(x, y, z -> d)

Figure credit: Justin Johnson

What is an implicit representation for 3D data?

Example: representing a 3D occupancy grid



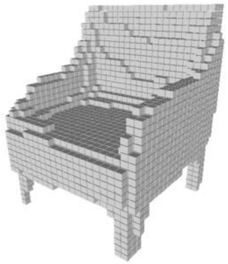
Explicit: A tensor of **3D voxel grid** $V \in \{0, 1\}^{[H,W,L]}$

Implicit: A **function** that maps locations to occupancies

$$F_\theta: x, y, z \rightarrow \{0, 1\}$$

What is an implicit representation for 3D data?

Example: representing a 3D occupancy grid



Explicit: A tensor of **3D voxel grid** $V \in \{0,1\}^{[H,W,L]}$

Implicit: A **function** that maps locations to occupancies

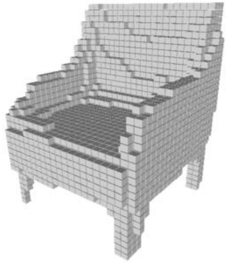
$$F_\theta: x, y, z \rightarrow \{0,1\}$$

Implicit representation describes 3D shapes using **mathematical functions** rather than explicit voxels, points, or mesh.
Example: Signed Distance Function

$$F_\theta:\ \mathbb{R}^3 \rightarrow \mathbb{R}$$

What is an implicit representation for 3D data?

Example: representing a 3D occupancy grid



Explicit: A tensor of **3D voxel grid** $V \in \{0, 1\}^{[H,W,L]}$

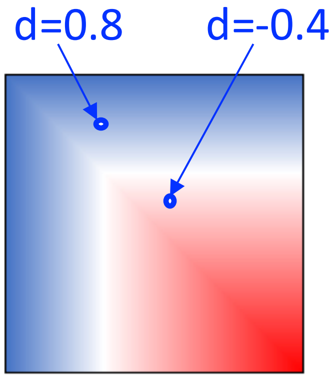
Implicit: A **function** that maps locations to occupancies

$$F_\theta: x, y, z \rightarrow \{0, 1\}$$

Implicit representation describes 3D shapes using **mathematical functions** rather than explicit voxels, points, or mesh.
Example: Signed Distance Function

$$F_\theta:\ \mathbb{R}^N \rightarrow \mathbb{R}$$

Can we represent more than just geometry?



SDF distance map

How far is a point from the nearest surface, and is the point *inside or outside* of the shape?

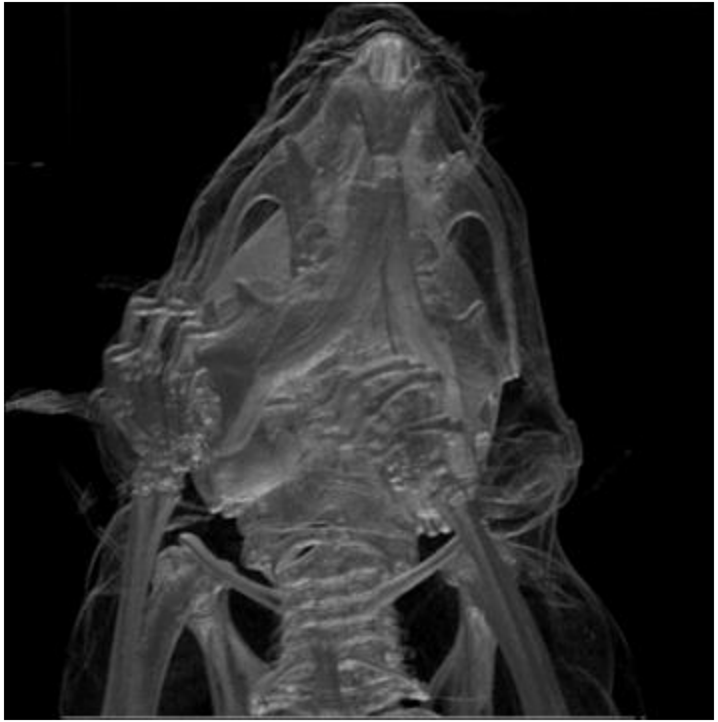
Implicit 3D Representation: Beyond Geometry

$$f_\theta(viewpoint) = Image$$

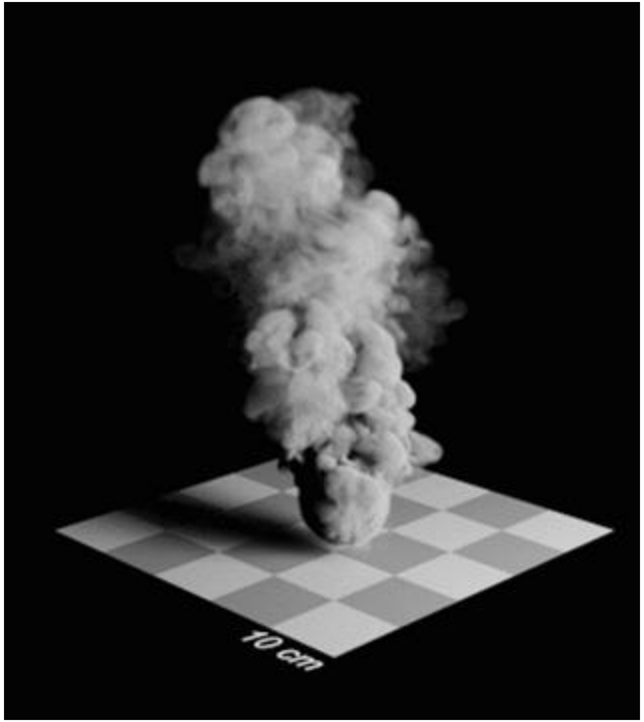
Goal: Learn an implicit 3D representation function that maps any camera viewpoint to full RGB images

Can we implicitly represent a full 3D scene, including its fine-grained geometry (e.g., surface occupancy) and appearance?

Basics: Volume Rendering

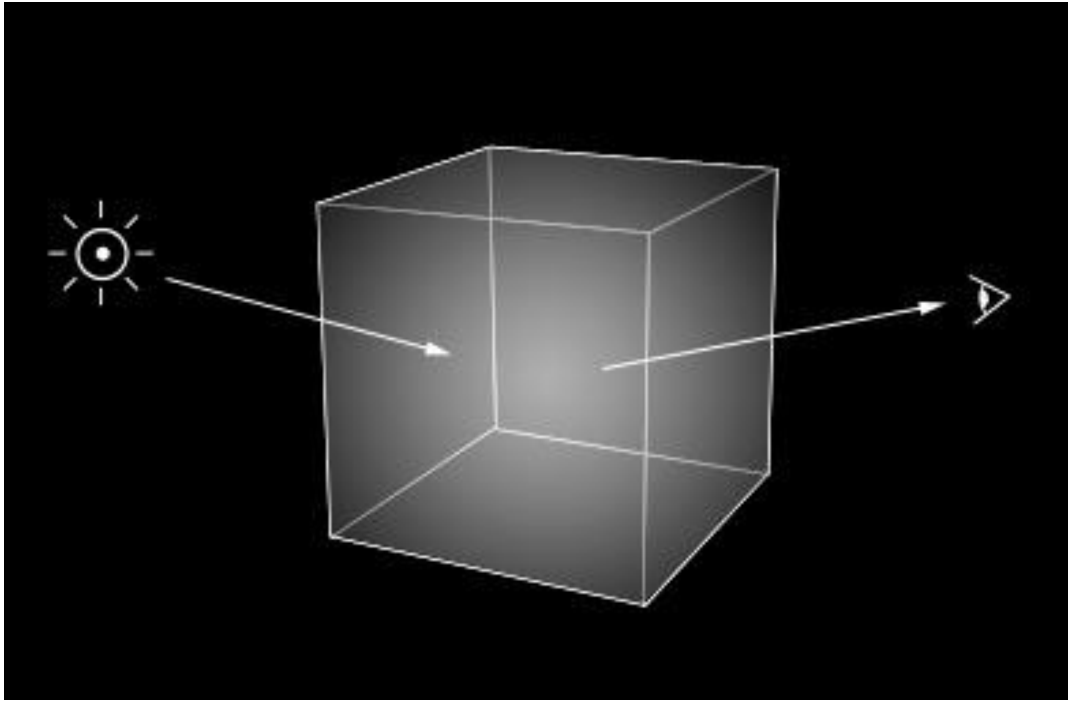


https://en.wikipedia.org/wiki/Volume_rendering

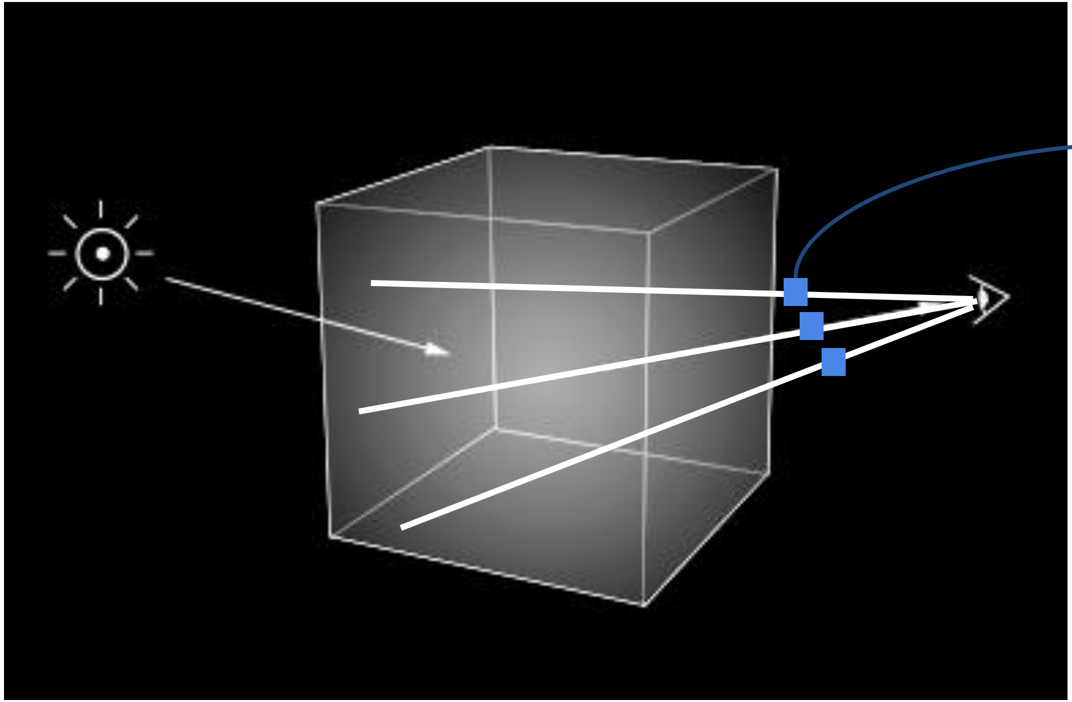


https://coronarenderer.freshdesk.com/support/solutions/articles/12000045276-how-to-use-the-corona-volume-grid-

Volume Rendering: Scene Representation



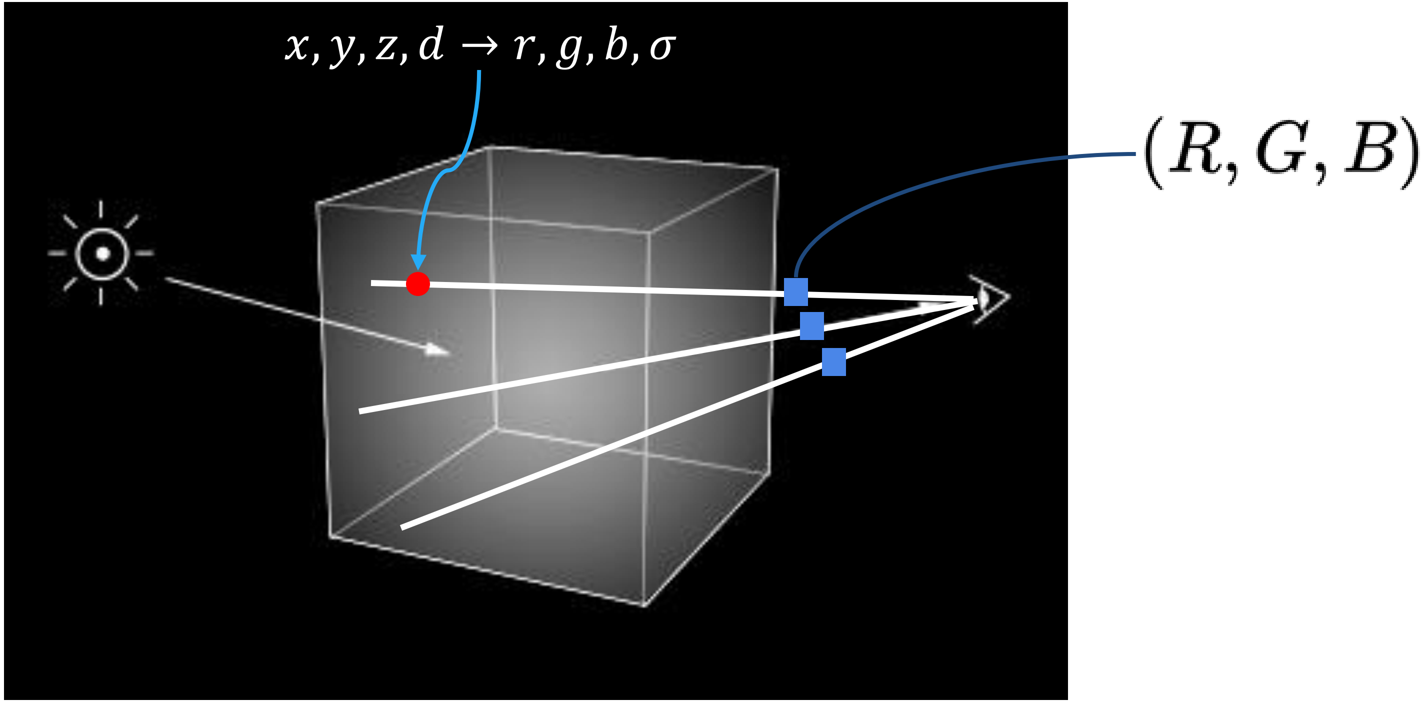
Volume Rendering: Scene Representation



(r, g, b)

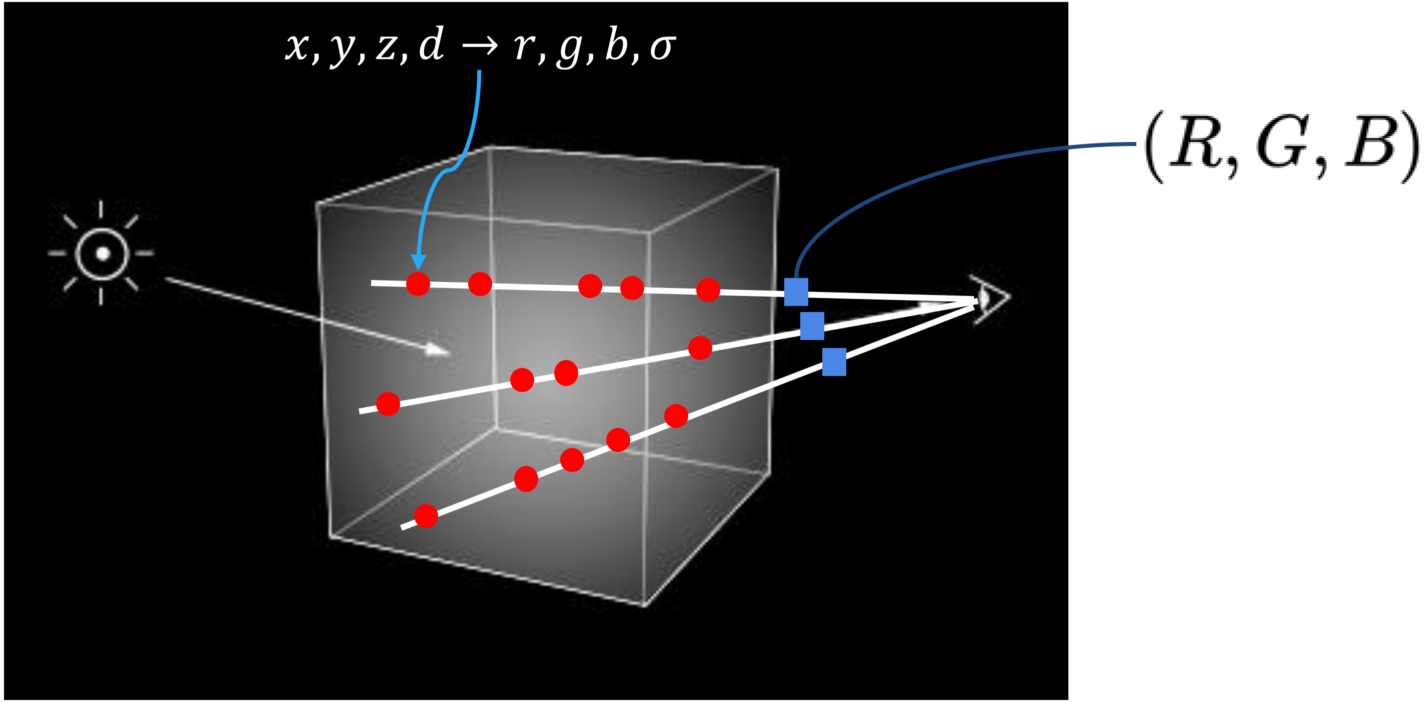
Volume Rendering: Scene Representation

Each location (x, y, z) emits certain color r, g, b when viewed with direction d.
We represent point occupancy continuously as density d.



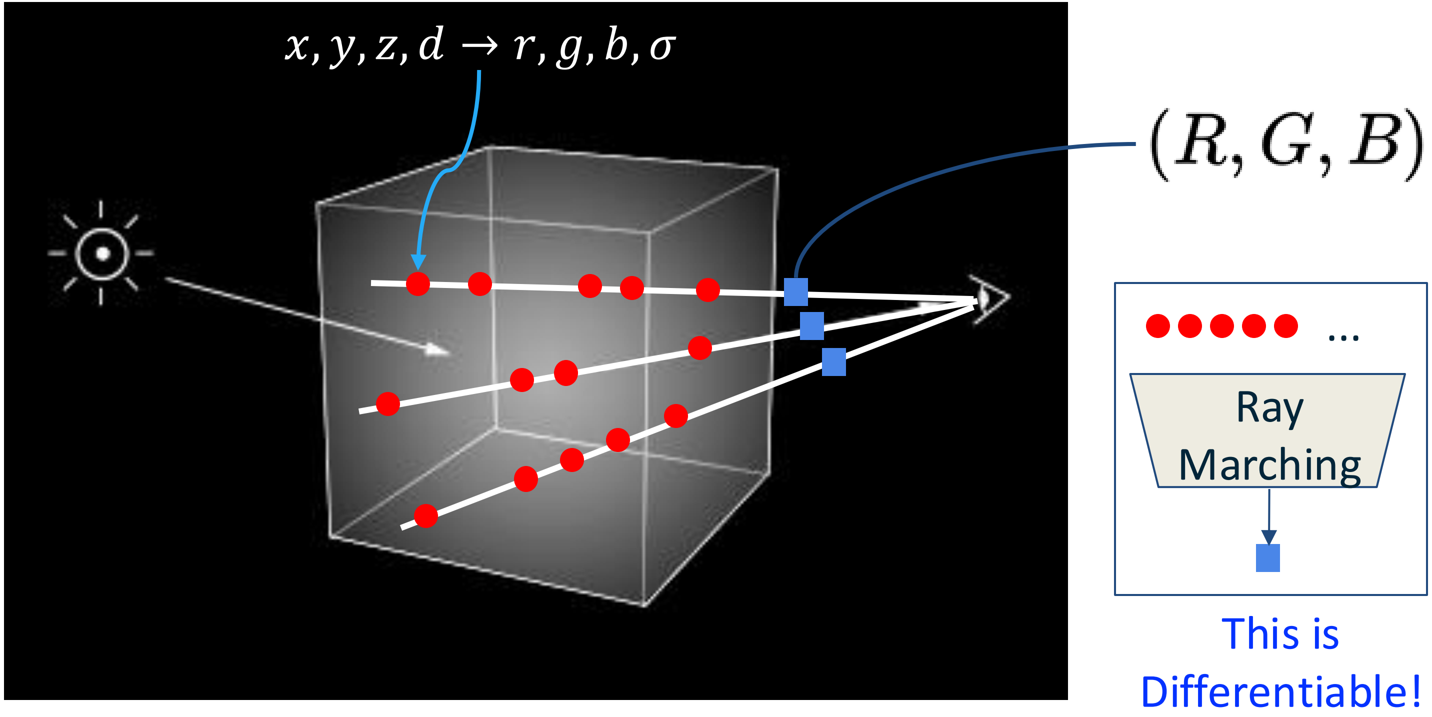
Volume Rendering: Scene Representation

Each location (x, y, z) emits certain color r, g, b when viewed with direction d.
We represent point occupancy continuously as density d.



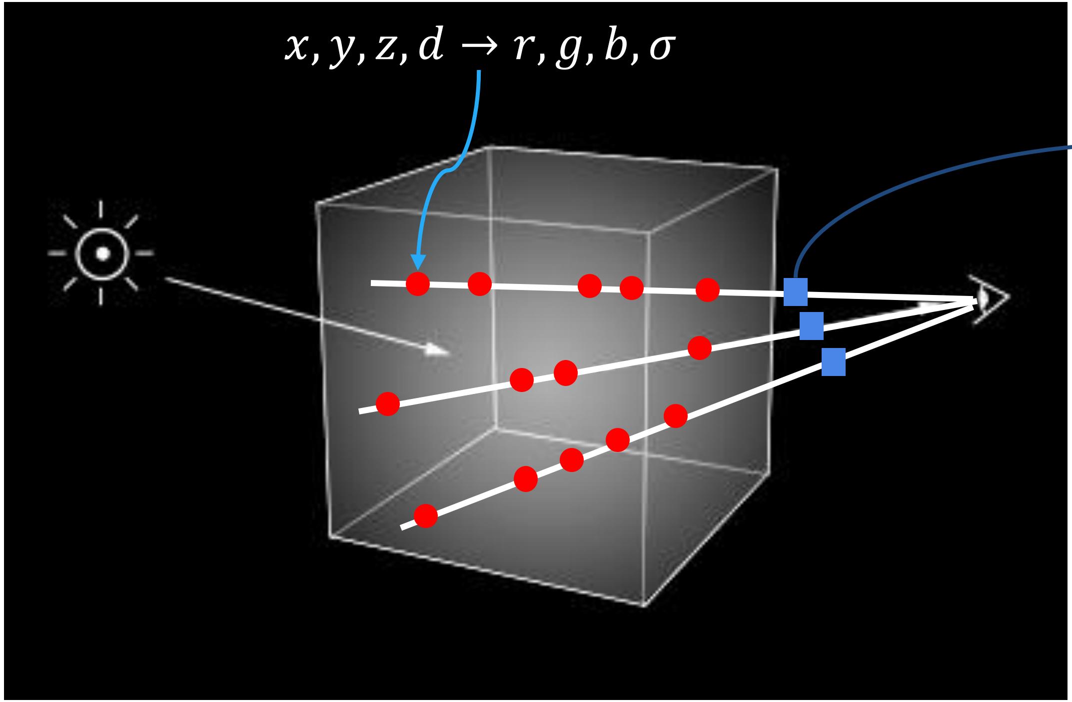
Volume Rendering: Ray Marching

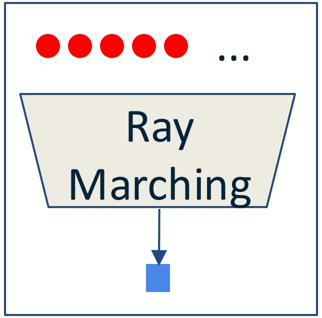
Ray Marching: Integrate color and density of points along a ray (via discretization) to render an RGB value. Render many points -> An image!



Volume Rendering: Ray Marching

Neural Radiance Field (NeRF): Train a neural network to represent the ray marching volume rendering function: $F_\theta(x, y, z, d) = (r, g, b, \sigma)$. **Each NN encodes a 3D scene**.





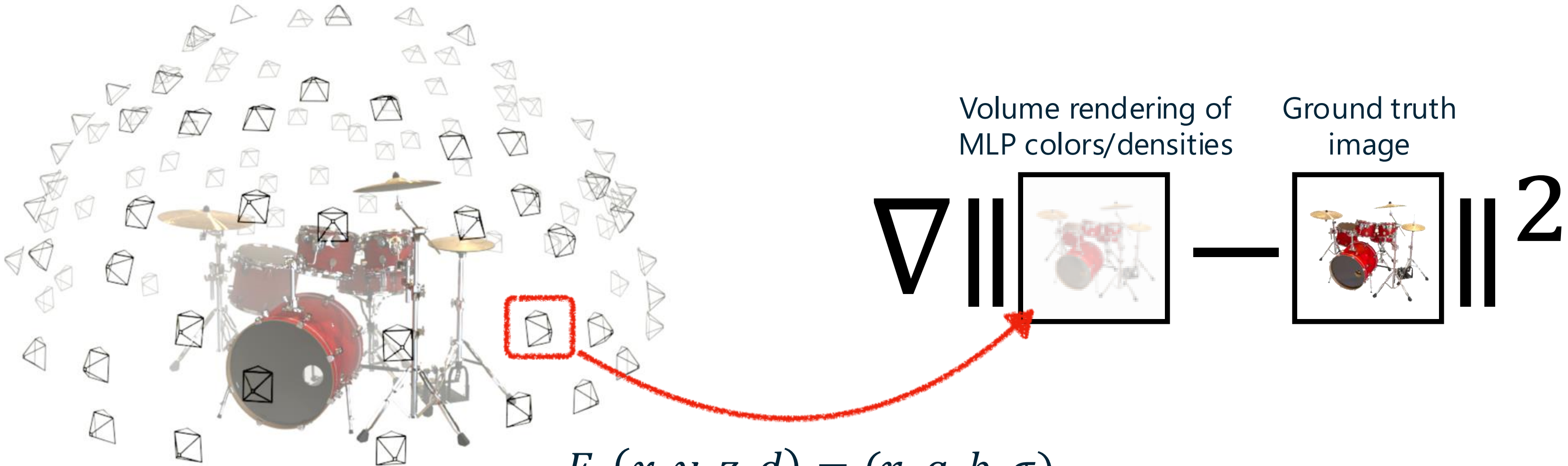
This is Differentiable!

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis

Ben Mildenhall[1*] Pratul P. Srinivasan[1*] Matthew Tancik[1*]
Jonathan T. Barron[2] Ravi Ramamoorthi[3] Ren Ng[1]

[1]UC Berkeley [2]Google Research [3]UC San Diego

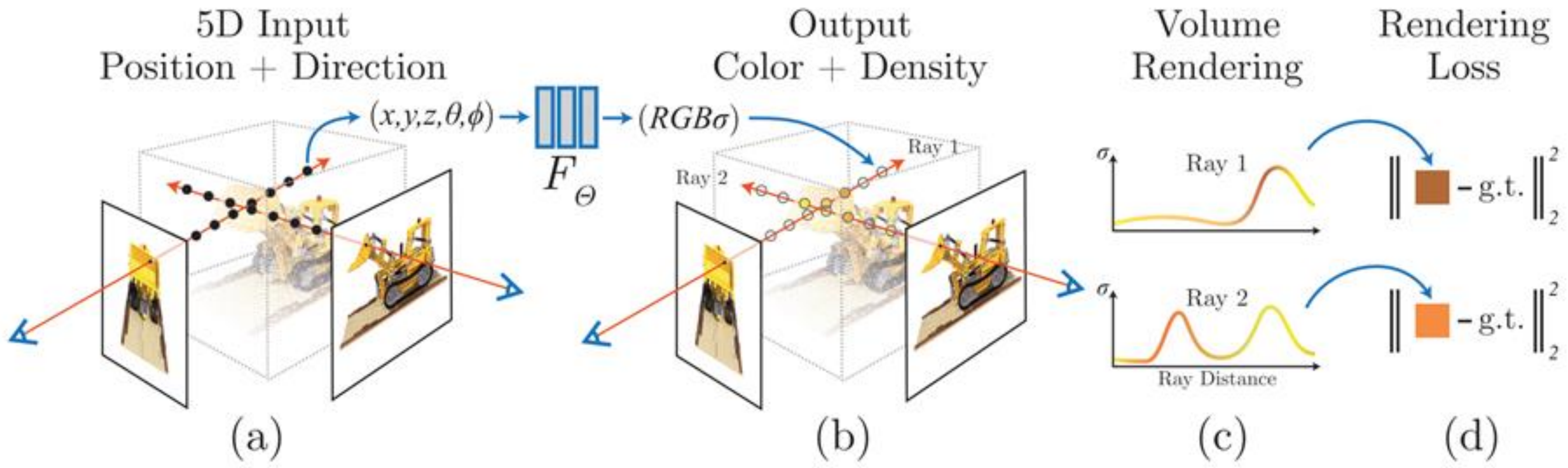
Train a Single Neural Network to Reproduce the Ground Truth Images of a Scene



$$F_\theta(x, y, z, d) = (r, g, b, \sigma)$$

Adapted from material from Pratul Srinivasan

NeRF Overview



NeRF: Optimization

The volume density $\sigma(\mathbf{x})$ can be interpreted as the differential probability of a ray terminating at an infinitesimal particle at location $\mathbf{x}$. The expected color $C(\mathbf{r})$ of camera ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ with near and far bounds t_n and t_f is:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt \,, \text{ where } T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right). \quad (1)$$

Solution: Numerically estimate the integral (quadrature).

1. Discretize the ray into bins.
2. Sample point in each bin.
3. Compute numerical integration.

NeRF: Optimization

The volume density $\sigma(\mathbf{x})$ can be interpreted as the differential probability of a ray terminating at an infinitesimal particle at location $\mathbf{x}$. The expected color $C(\mathbf{r})$ of camera ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ with near and far bounds t_n and t_f is:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt\,, \text{ where } T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right). \quad (1)$$

Solution: Numerically estimate the integral (quadrature).

1. Discretize the ray into bins.
2. Sample point in each bin.
3. Compute numerical integration.

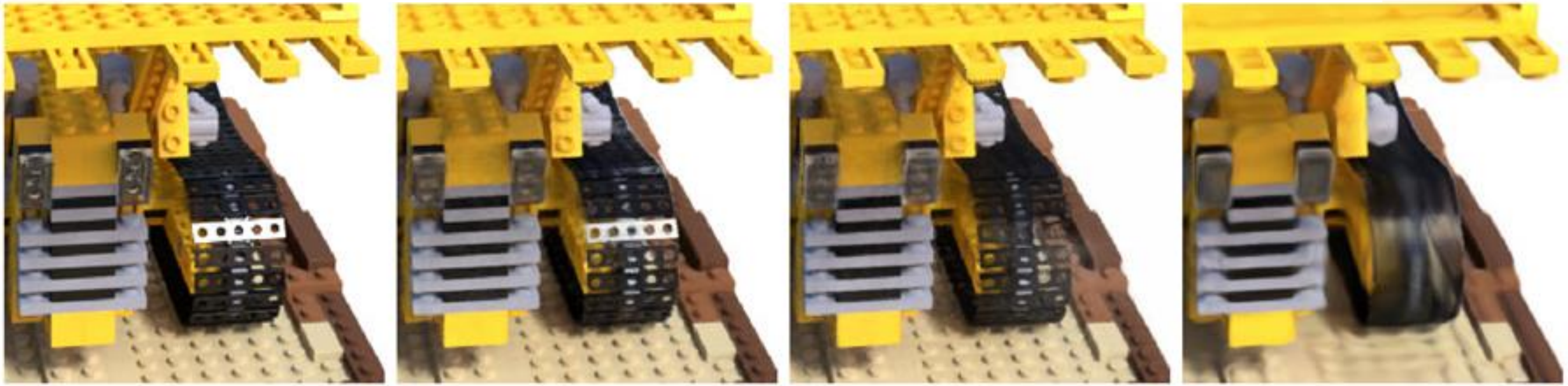
$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))c_i\,, \text{ where } T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j\delta_j\right)$$

Key Insight 1: Positional Encoding

Challenge: Having F_θ operate directly on (x, y, z, d) performs poorly.

Solution: Positional encoding

$$\gamma(p) = \left(\sin\left(2^0 \pi p\right), \cos\left(2^0 \pi p\right), \cdots, \sin\left(2^{L-1} \pi p\right), \cos\left(2^{L-1} \pi p\right) \right)$$



Ground Truth Complete Model No View Dependence No Positional Encoding

Key Insight 2: Hierarchical Volume Rendering

Challenge: Waste of compute on empty space.

Solution: coarse-to-fine prediction.

$$\hat{C}_c(\mathbf{r}) = \sum_{i=1}^{N_c} w_i c_i \,, \qquad w_i = T_i(1 - \exp(-\sigma_i \delta_i)) \,. \tag{5}$$

Normalizing these weights as $\hat{w}_i = w_i / \sum_{j=1}^{N_c} w_j$ produces a piecewise-constant PDF along the ray. We sample a second set of N_f locations from this distribution using inverse transform sampling, evaluate our "fine" network at the union of the first and second set of samples, and compute the final rendered color of the ray $\hat{C}_f(\mathbf{r})$ using Eqn. $\boxed{3}$ but using all $N_c + N_f$ samples. This procedure allocates more

NeRF encodes convincing view-dependent effects using directional dependence

Slide credit: Noah Snavely

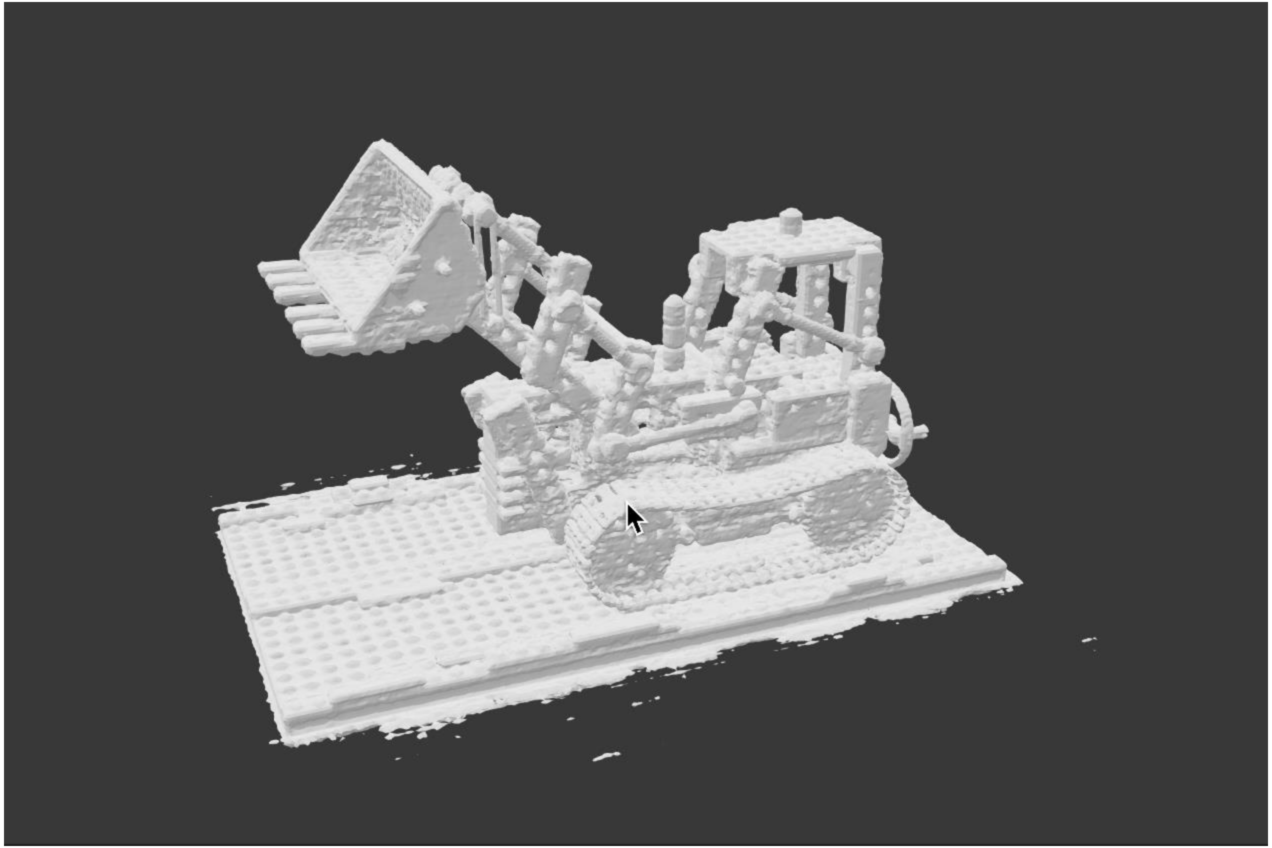
NeRF encodes convincing view-dependent effects using directional dependence

Slide credit: Noah Snavely

NeRF encodes detailed scene geometry with occlusion effects

Slide credit: Noah Snavely

NeRF encodes detailed scene geometry



Slide credit: Noah Snavely

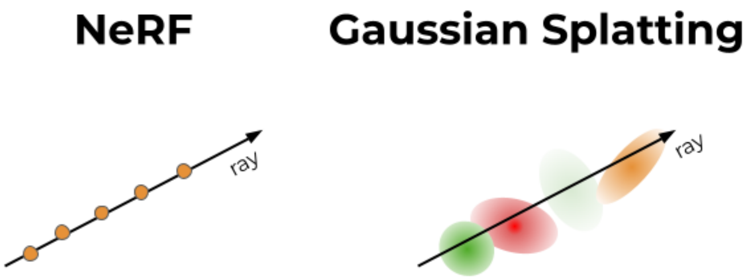
Space vs. Time Tradeoff

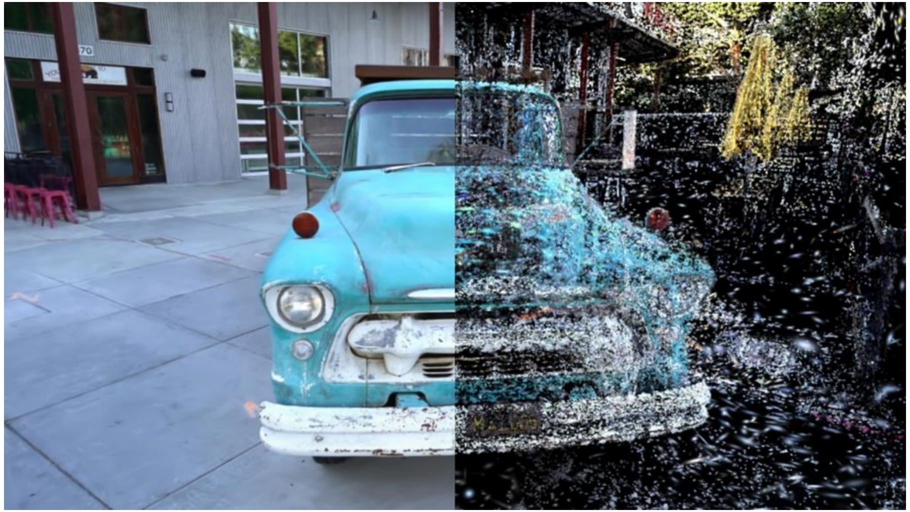
The biggest practical tradeoffs between these methods are time versus space. All compared single scene methods take at least 12 hours to train per scene. In contrast, LLFF can process a small input dataset in under 10 minutes. However, LLFF produces a large 3D voxel grid for every input image, resulting in enormous storage requirements (over 15GB for one "Realistic Synthetic" scene). Our method requires only 5 MB for the network weights (a relative compression of $3000\times$ compared to LLFF), which is even less memory than the *input images alone* for a single scene from any of our datasets.

3D Gaussian Splatting (Kerbl and Kopanas et al., 2023)

Key idea: 3D Gaussians as an **explicit representation** of a scene

- Train Gaussian blobs via inverse rendering (similar to NeRF)
- Store scene as Gaussian blobs instead of neural network weights (NeRF)
- Much faster during inference, but takes a lot of space to store





Summary: 3D Representation and Neural Rendering

- Representation matters a lot for 3D computer vision tasks (detection, reconstruction, etc.)
- 3D Voxels are intuitive representation of space but struggles with high-resolution shape and large scenes
- Implicit function emerge as a new paradigm in representing scenes with Neural Networks
- Neural volume rendering: represent scenes implicit as point-direction to color-density neural networks. Photorealistic rendering, slow to train and evaluate
- More recent works on trading off space and time