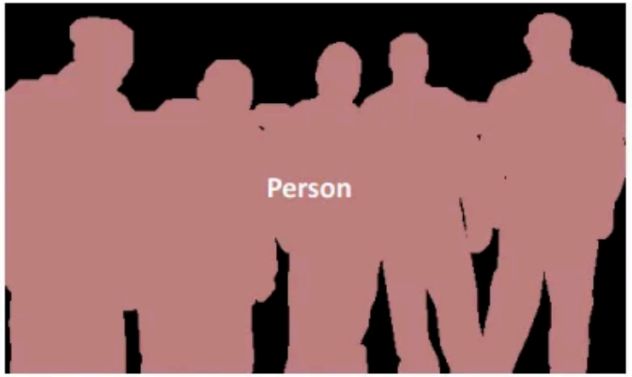
Lecture 16: Object Detection + Adversarial Examples

Announcements:

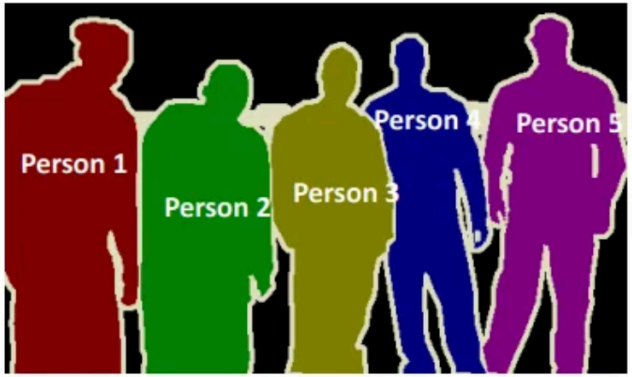
- Today we'll cover object detection and adversarial examples.
- This Wednesday's lecture is canceled. Instead, I will upload a video to Bruin Learn that completes the material in this course.
- The project and its accompanying data have been uploaded to Bruin Learn. It is due **March 15, 2024** (Friday of Week 10).
    - You will be allowed to use PyTorch, Keras, or other deep learning libraries for the project.
- Midterm regrades are due tonight, submitted by 11:59pm.

Segmentation

Object Detection



Semantic Segmentation



Instance Segmentation

From COCO:

"The panoptic segmentation task involves **assigning a semantic label and instance id for each pixel of an image, which requires generating dense, coherent scene segmentations.** The stuff annotations for this task come from the COCO-Stuff project described in this paper. For more details about the panoptic task, including evaluation metrics, please see the panoptic segmentation paper."

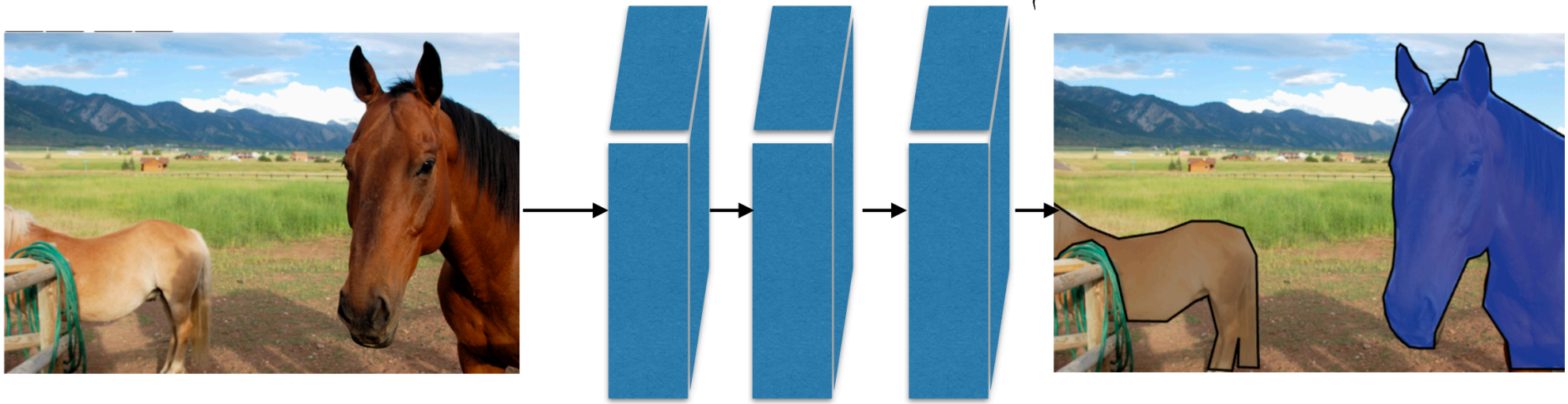
"Panotptic annotations defines defines 200 classes but only uses 133"

Segmentation

Why not just train one CNN to do this?

$256 \cdot 256 \cdot 1000 = 65{,}536{,}000$ #'s

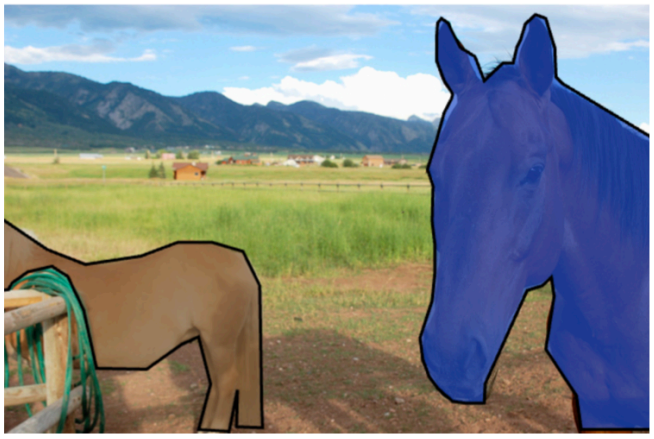
4 bytes ⇒ 250MB

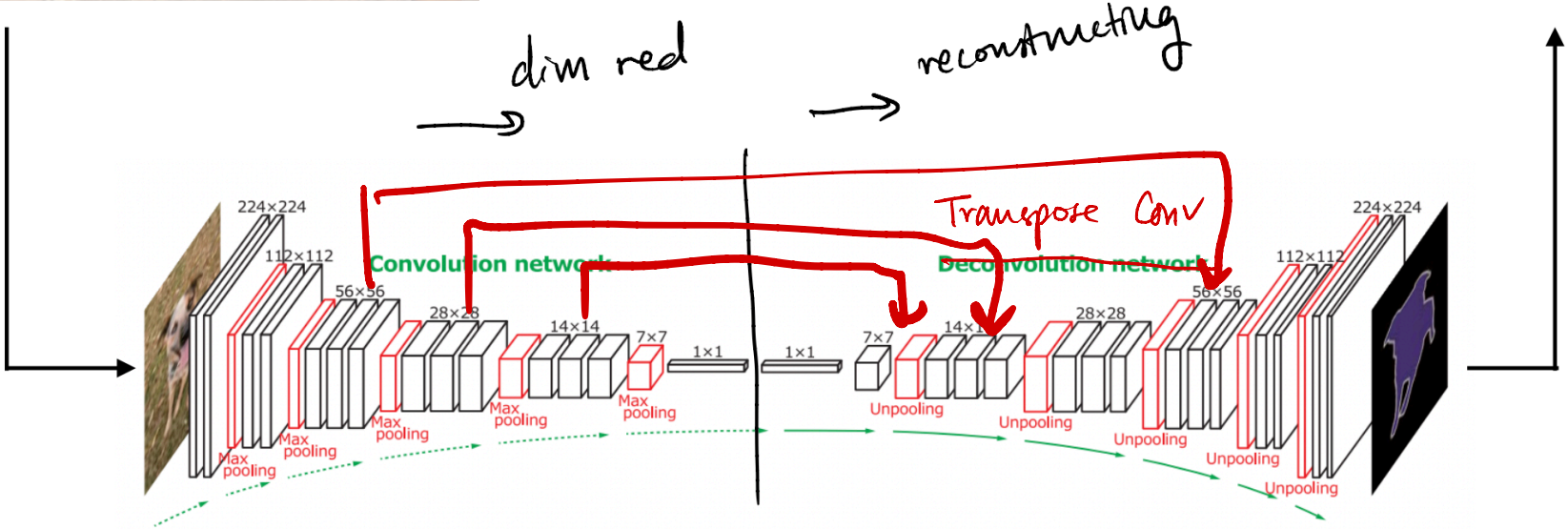


This should be a C x H x W tensor of ~~scores~~ probs per pixel.

Question: What is the major con of this approach?

Have a bottleneck to reduce the number of parameters



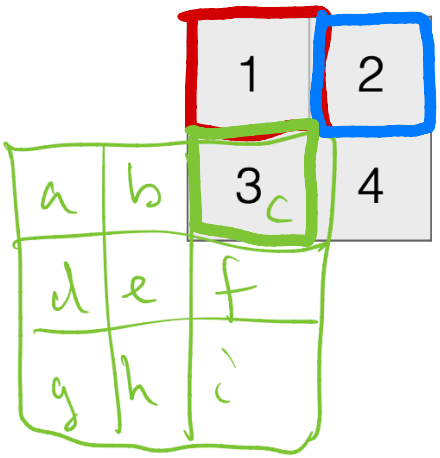


https://arxiv.org/pdf/1505.04366.pdf

Transposed convolution

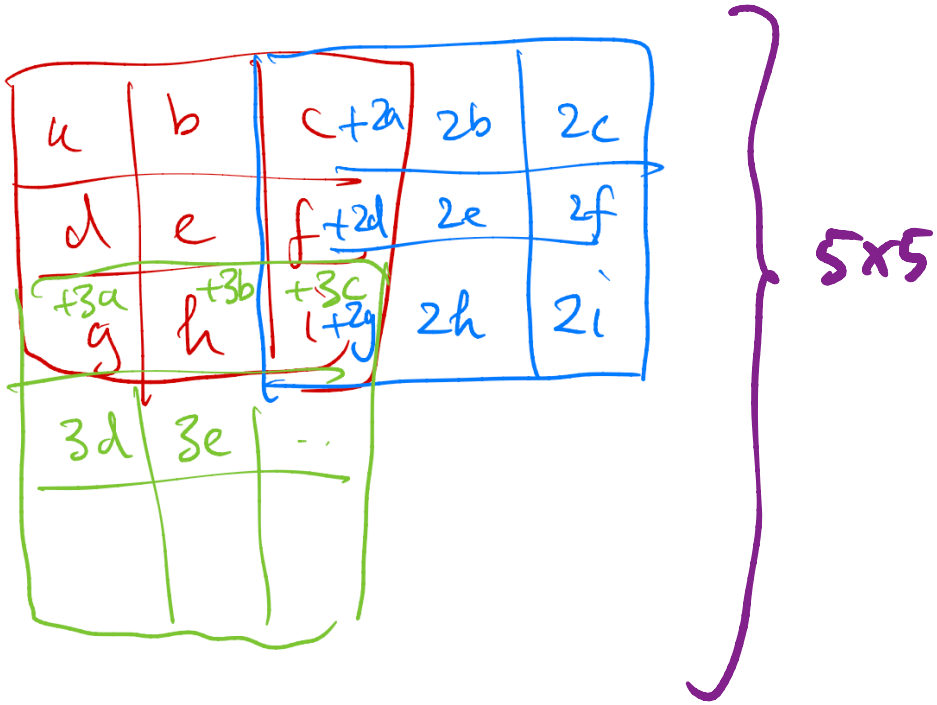
stride 2 transposed convolution

2x2



3x3

a	b	c
d	e	f
g	h	i



Transposed convolution

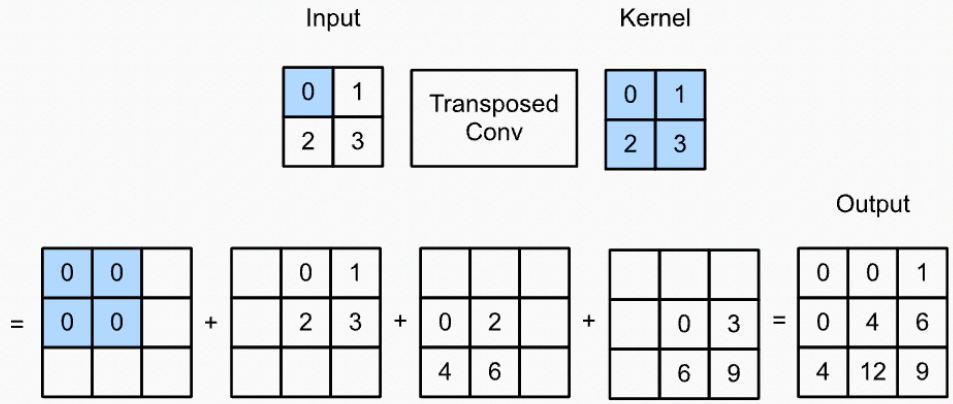


Fig. 14.10.1 Transposed convolution with a 2×2 kernel. The shaded portions are a portion of an intermediate tensor as well as the input and kernel tensor elements used for the computation.

https://d2l.ai/chapter_computer-vision/transposed-conv.html

Transposed convolution

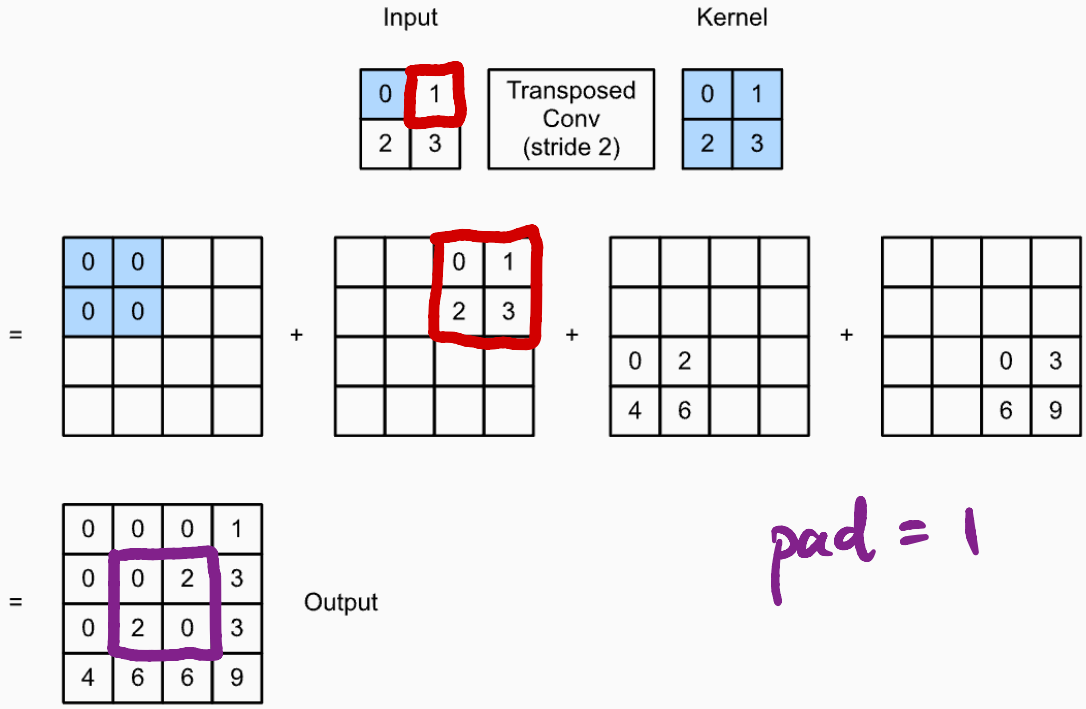


Fig. 14.10.2 Transposed convolution with a 2×2 kernel with stride of 2. The shaded portions are a portion of an intermediate tensor as well as the input and kernel tensor elements used for the computation.

https://d2l.ai/chapter_computer-vision/transposed-conv.html

Transposed convolution — padding removes from the output

Different from in the regular convolution where padding is applied to input, it is applied to output in the transposed convolution. For example, when specifying the padding number on either side of the height and width as 1, the first and last rows and columns will be removed from the transposed convolution output.

The output of a transposed convolution is therefore:

(input_size - 1) * stride + kernel_size - 2*pad

https://d2l.ai/chapter_computer-vision/transposed-conv.html

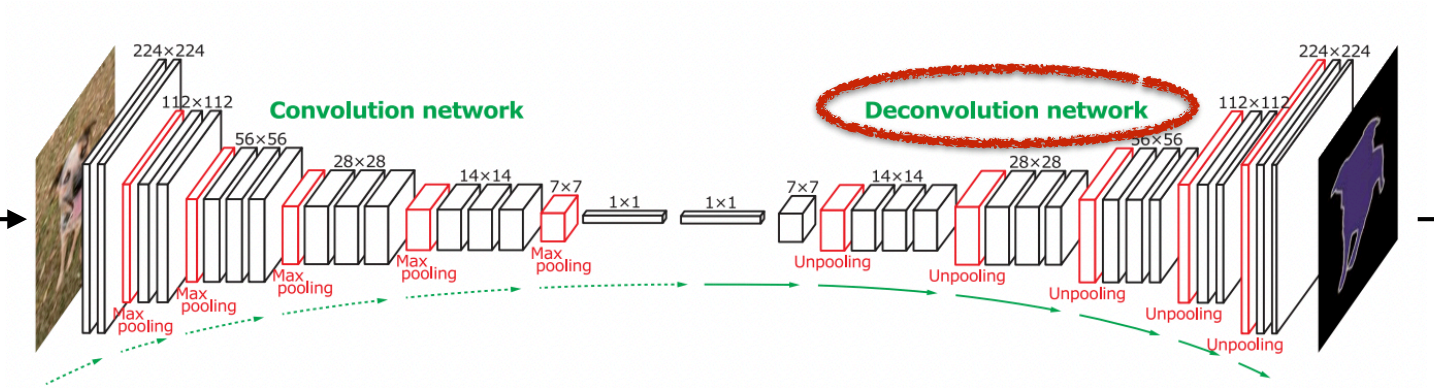
One more padding example

What is the size of an output of a transposed convolution with:

Input: 2x2
Kernel: 3x3
Stride = 2
Pad = 1

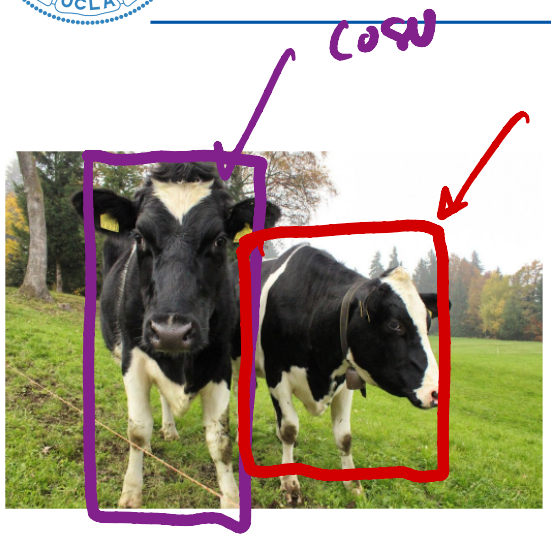
From equation, (input_size - 1) * stride + kernel_size - 2*pad = 2 + 3 - 2 = 3.

Semantic labeling of each pixel

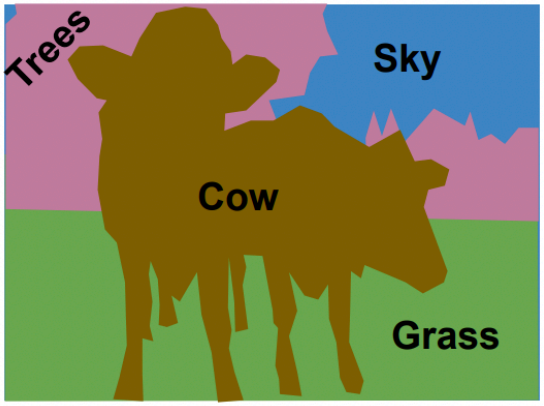


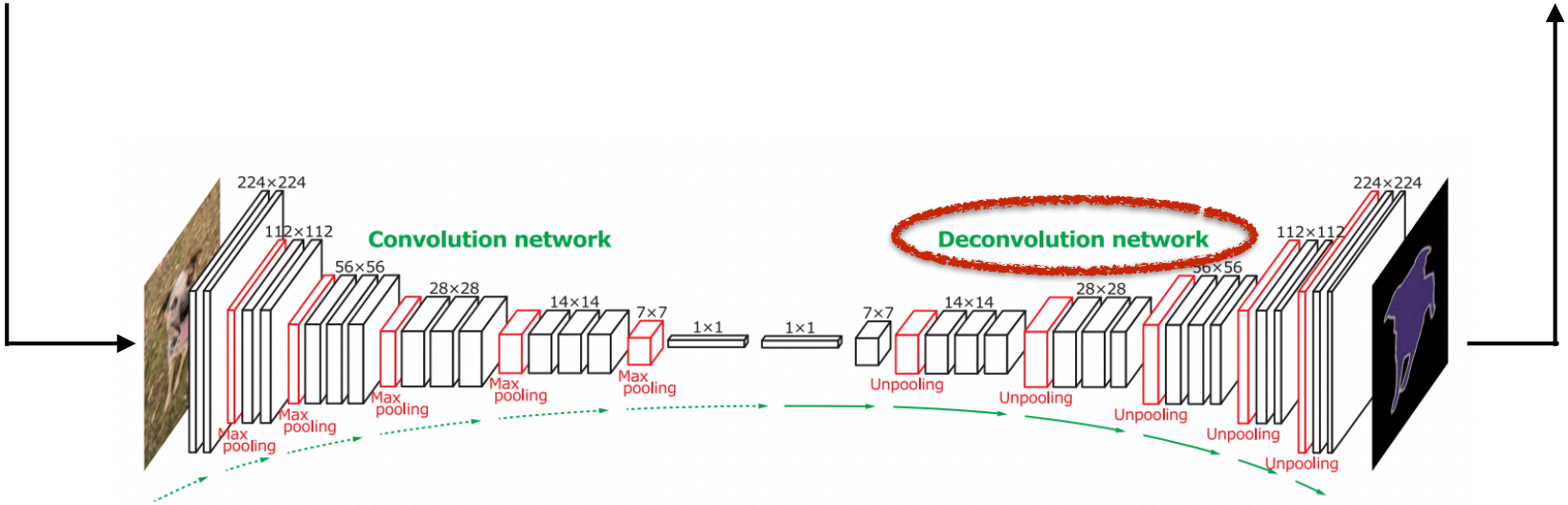
https://arxiv.org/pdf/1505.04366.pdf

Semantic labeling of each pixel



"obj. detection"

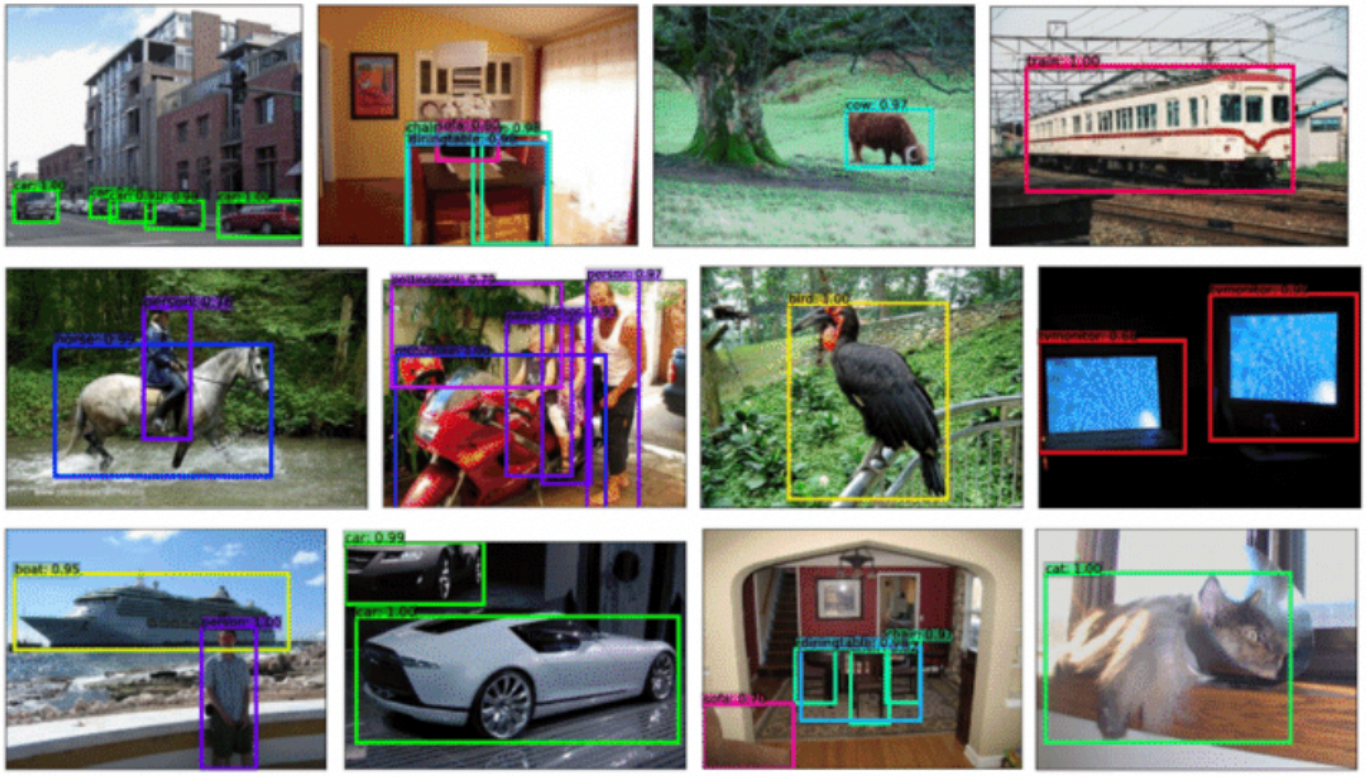




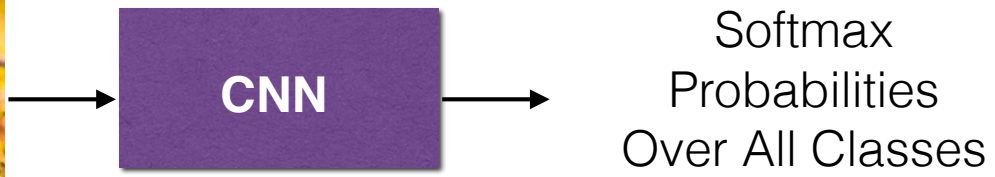
While this labels every pixel, it doesn't separate instances… like the two cows here. To do this, we also need ***localization*** — *where is an object in the image?* This extends to object detection — multiple objects in an image, all localized.

Object detection

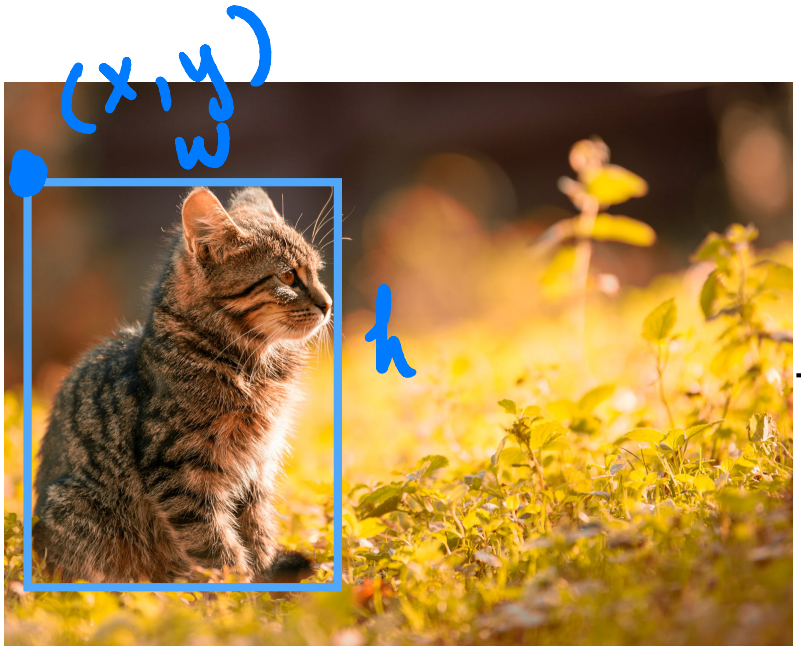
We'll look next at object detection. This tells us not only that this object is in the image, but where it is. This will eventually lead to instance segmentation.



Single object detection (i.e. classification + localization)



Single object detection

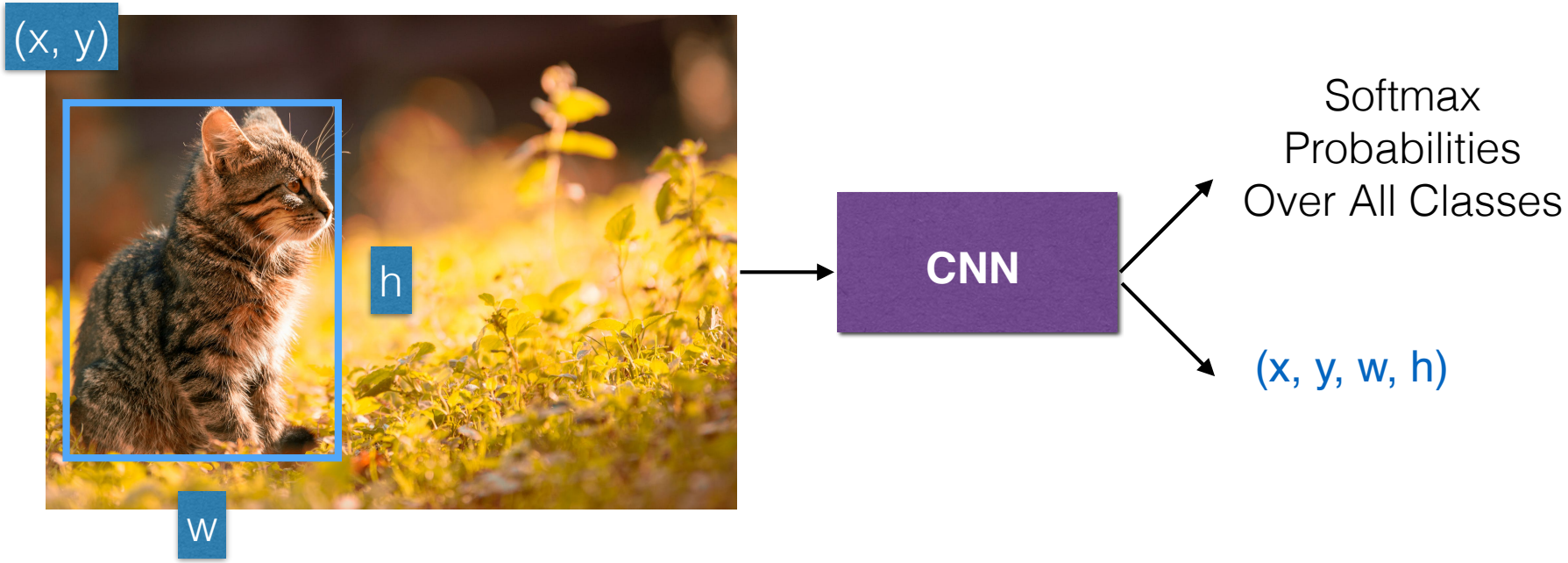


CNN

Softmax
Probabilities
Over All Classes

(x, y, w, h)

Single object detection



The training data needs to come with this information!

Single object detection

CNN

softmax

(x, y, w, h)

$(\hat{x} - x)^2$

$(\hat{y} - y)^2$

$(\hat{w} - w)^2$

$(\hat{h} - h)^2$

Localization of the cat is scored via a regression loss (e.g., L2 loss between predicted x, y, w, h, and true x, y, w, h)

Single object detection (classification + localization)

CNN

softmax → CE

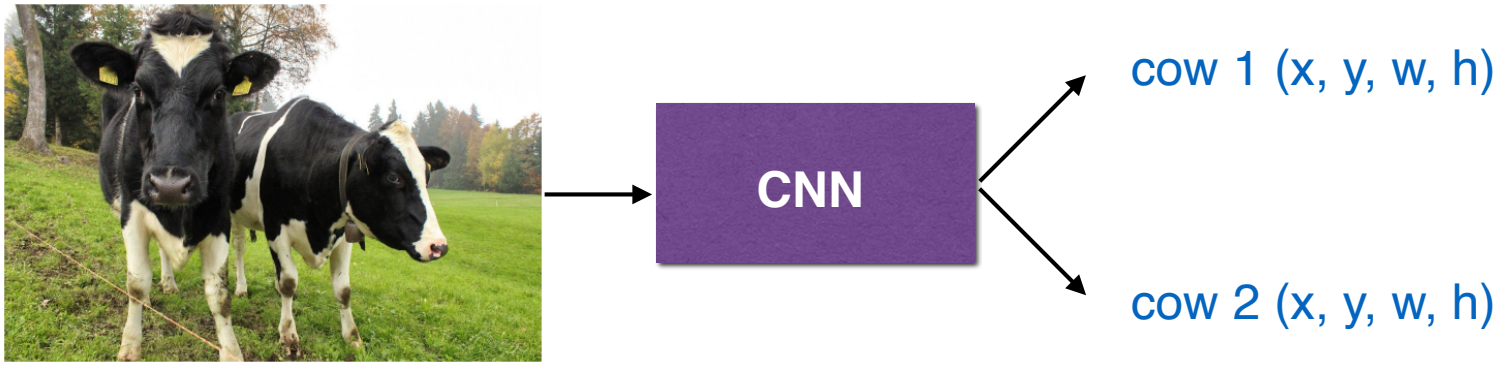
+

(x, y, w, h) → regression

L = ?

How does this extend to multiple objects?

Multiple object detection

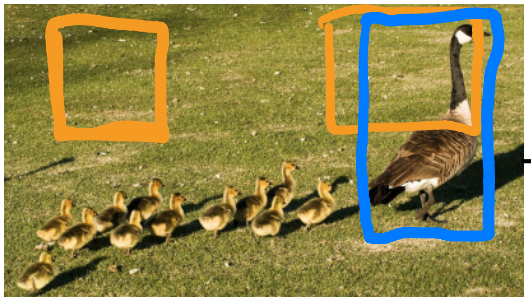


Multiple object detection

CNN

cow 1 (x, y, w, h)

cow 2 (x, y, w, h)



CNN

duck 1 (x, y, w, h)

duck 2 (x, y, w, h)

:

duck 12 (x, y, w, h)

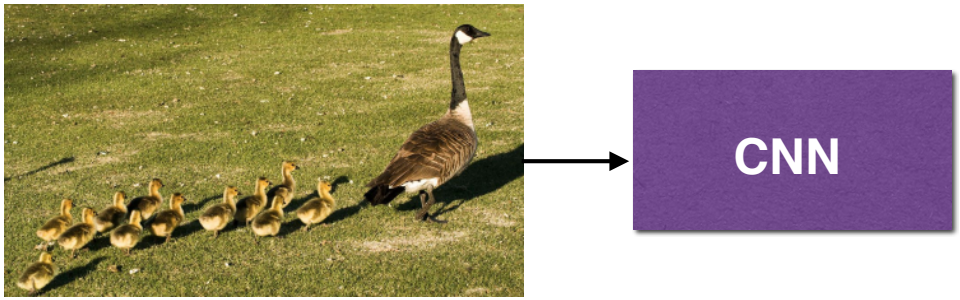
How do we handle multiple outputs?

Multiple object detection

We cannot output a variable number of objects easily, so we need a different solution.

So what if we instead iterate over sliding windows, and if it detects an object above some threshold, then it will output not only the class but also a bounding box.

We will need to add a class for “background” meaning there is no object.

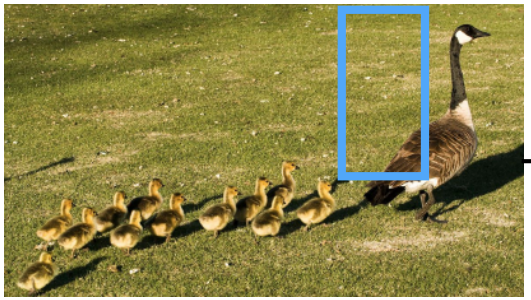


Multiple object detection

We cannot output a variable number of objects easily, so we need a different solution.

So what if we instead iterate over sliding windows, and if it detects an object above some threshold, then it will output not only the class but also a bounding box.

We will need to add a class for “background” meaning there is no object.



softmax where
background has highest probability

Multiple object detection

We cannot output a variable number of objects easily, so we need a different solution.

So what if we instead iterate over sliding windows, and if it detects an object above some threshold, then it will output not only the class but also a bounding box.

We will need to add a class for “background” meaning there is no object.

Multiple object detection

We cannot output a variable number of objects easily, so we need a different solution.

So what if we instead iterate over sliding windows, and if it detects an object above some threshold, then it will output not only the class but also a bounding box.

We will need to add a class for “background” meaning there is no object.

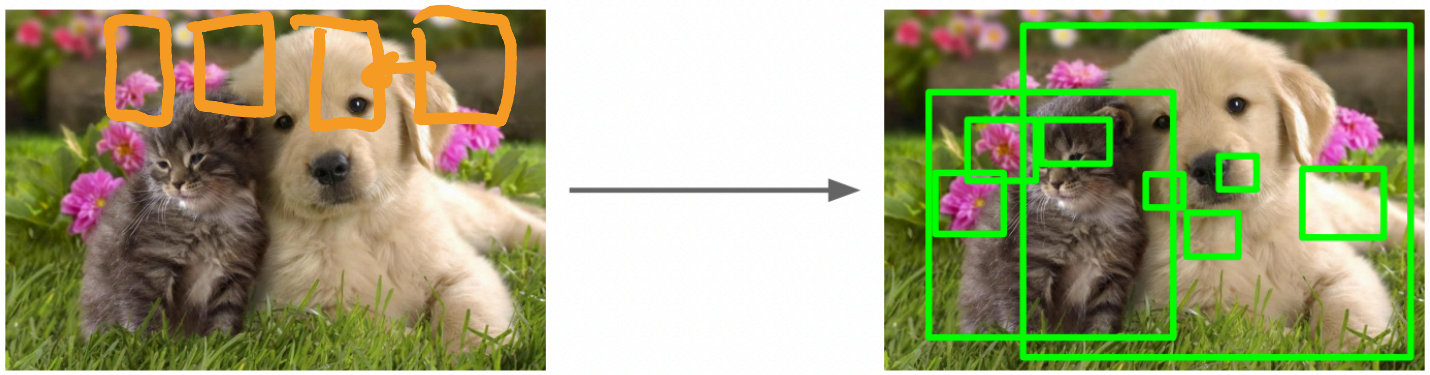
But we have the same problem of this taking a long time (many sliding windows)!

Multiple object detection

Big problem: how do we get around having to test many different crops?

Idea 1: choose crops "wisely"? Maybe with traditional image processing techniques? (like looking at edges…)

Solution 1: Region proposals



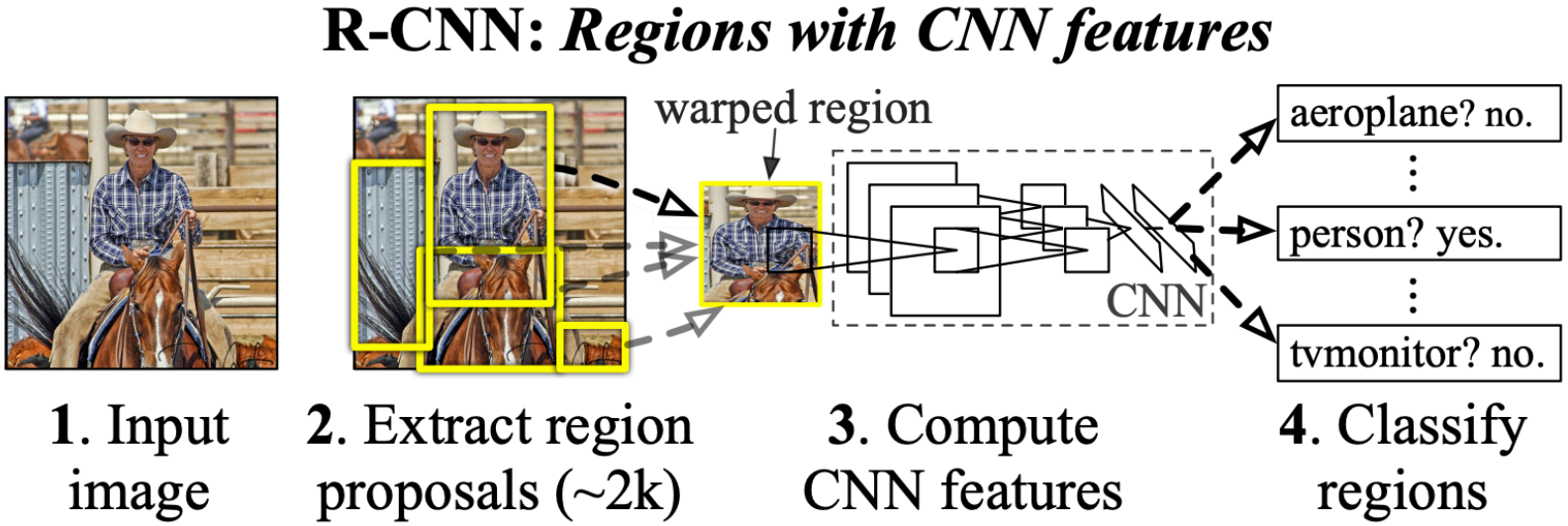
http://cs231n.stanford.edu/

Get "region proposals" from traditional techniques.

*note, we won't discuss in detail what these traditional techniques are because we'll soon see that we can do better without them.

Solution 1: Region proposals

Use a (non-deep learning) to propose regions (R-CNN).

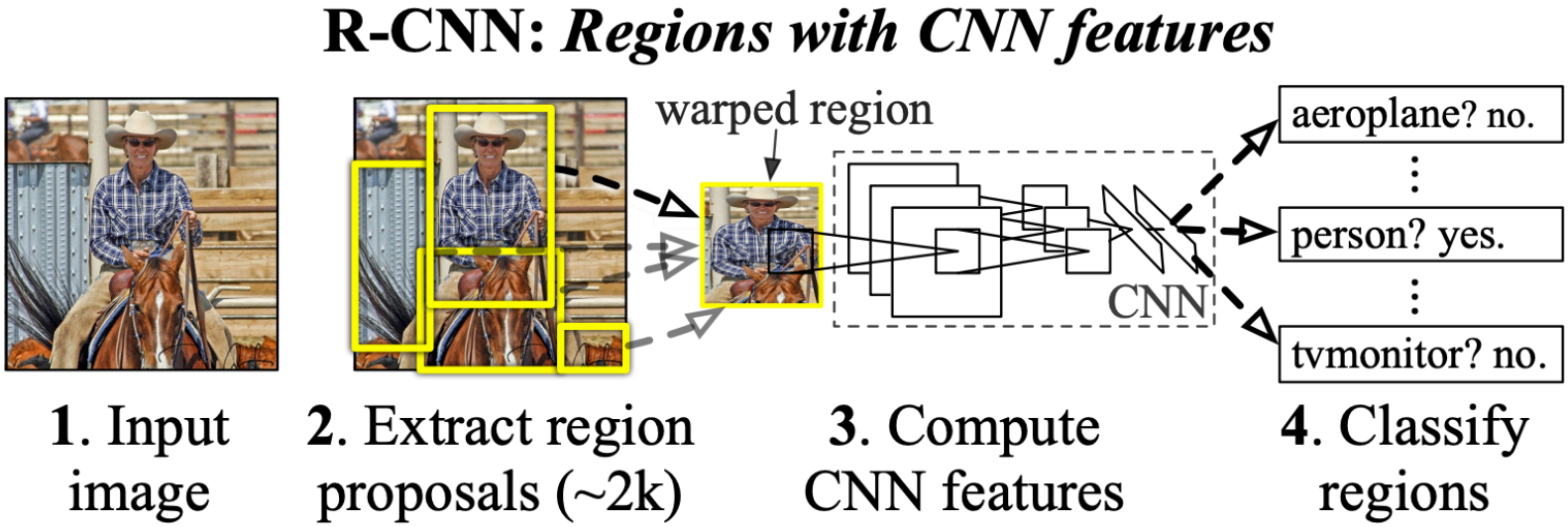


"Selective Search"

https://arxiv.org/pdf/1311.2524.pdf

Solution 1: Region proposals

Use a (non-deep learning) to propose regions (R-CNN).



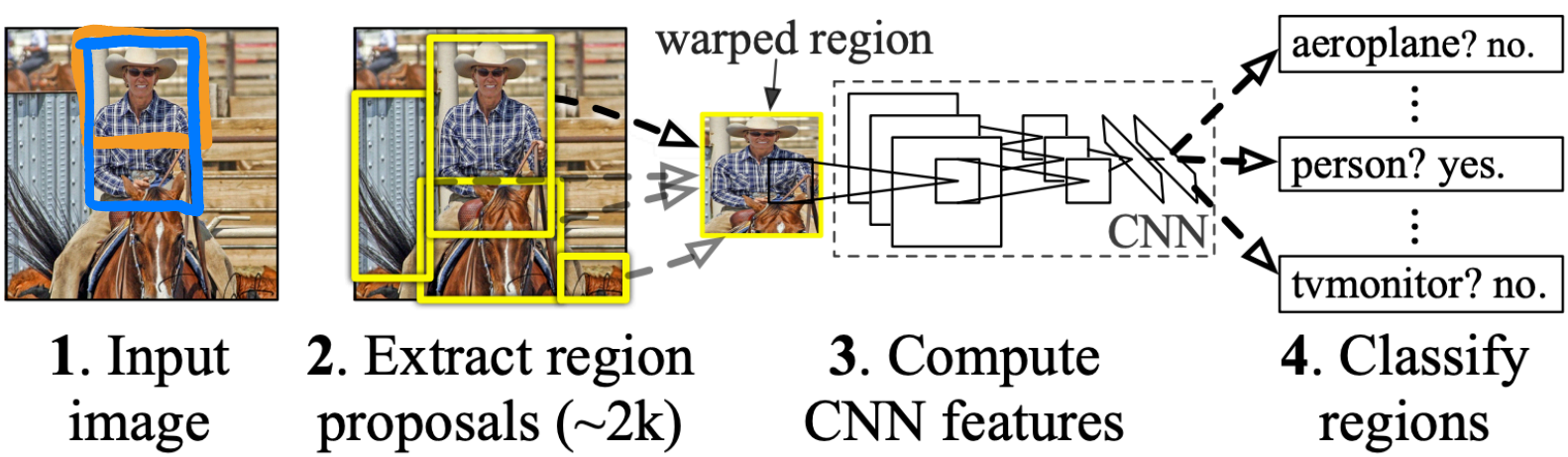
Examples include: objectness [1], selective search [39], category-independent object proposals [14], constrained parametric min-cuts (CPMC) [5], multi-scale combinatorial grouping [3], and Cireşan et al. [6], who detect mitotic cells by applying a CNN to regularly-spaced square crops, which are a special case of region proposals. While R-CNN is agnostic to the particular region proposal method, we use selective search to enable a controlled comparison with prior detection work (e.g., [39, 41]).

https://arxiv.org/pdf/1311.2524.pdf

Solution 1: Region proposals

Use a (non-deep learning) to propose regions (R-CNN).

1. Region Proposals
2. Classify every region
3. Class-specific BB



What if region proposal doesn't contain an object? Assign bg class

What if region proposal only contains part of an object? Re-assign BB

https://arxiv.org/pdf/1311.2524.pdf

Solution 1: Region proposals

Limitations:

1. **Training is a multi-stage pipeline.** R-CNN first fine-tunes a ConvNet on object proposals using log loss. Then, it fits SVMs to ConvNet features. These SVMs act as object detectors, replacing the softmax classifier learnt by fine-tuning. In the third training stage, bounding-box regressors are learned.
2. **Training is expensive in space and time.** For SVM and bounding-box regressor training, features are extracted from each object proposal in each image and written to disk. With very deep networks, such as VGG16, this process takes 2.5 GPU-days for the 5k images of the VOC07 trainval set. These features require hundreds of gigabytes of storage.
3. **Object detection is slow.** At test-time, features are extracted from each object proposal in each test image. Detection with VGG16 takes 47s / image (on a GPU).

https://arxiv.org/pdf/1504.08083.pdf

Solution 1: Region proposals

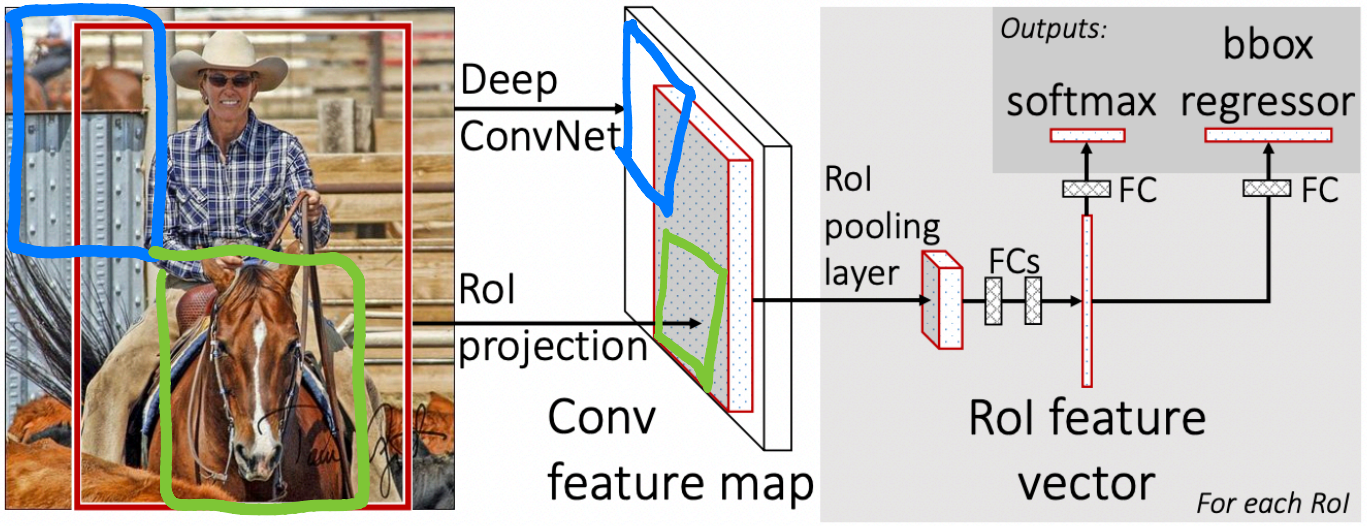
Limitations:

1. **Training is a multi-stage pipeline.** R-CNN first fine-tunes a ConvNet on object proposals using log loss. Then, it fits SVMs to ConvNet features. These SVMs act as object detectors, replacing the softmax classifier learnt by fine-tuning. In the third training stage, bounding-box regressors are learned.
2. **Training is expensive in space and time.** For SVM and bounding-box regressor training, features are extracted from each object proposal in each image and written to disk. With very deep networks, such as VGG16, this process takes 2.5 GPU-days for the 5k images of the VOC07 trainval set. These features require hundreds of gigabytes of storage.
3. **Object detection is slow.** At test-time, features are extracted from each object proposal in each test image. Detection with VGG16 takes 47s / image (on a GPU).

https://arxiv.org/pdf/1504.08083.pdf

How can we make it faster?

Solution 2: Fast R-CNN



https://arxiv.org/pdf/1504.08083.pdf

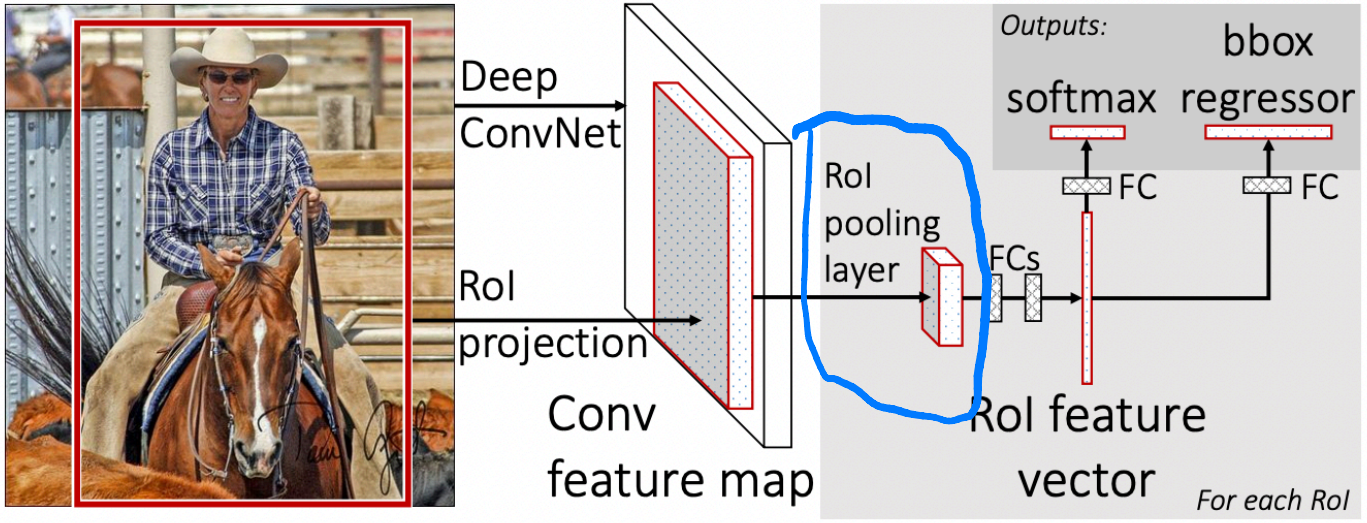
Idea: do the convolution over the image **only once (instead of 2000 times)**.

Then find the the RoI projection (spatial extent) in the convolutional feature space.

Solution 2: Fast R-CNN

Solution 2: Fast R-CNN

(w, h, c)

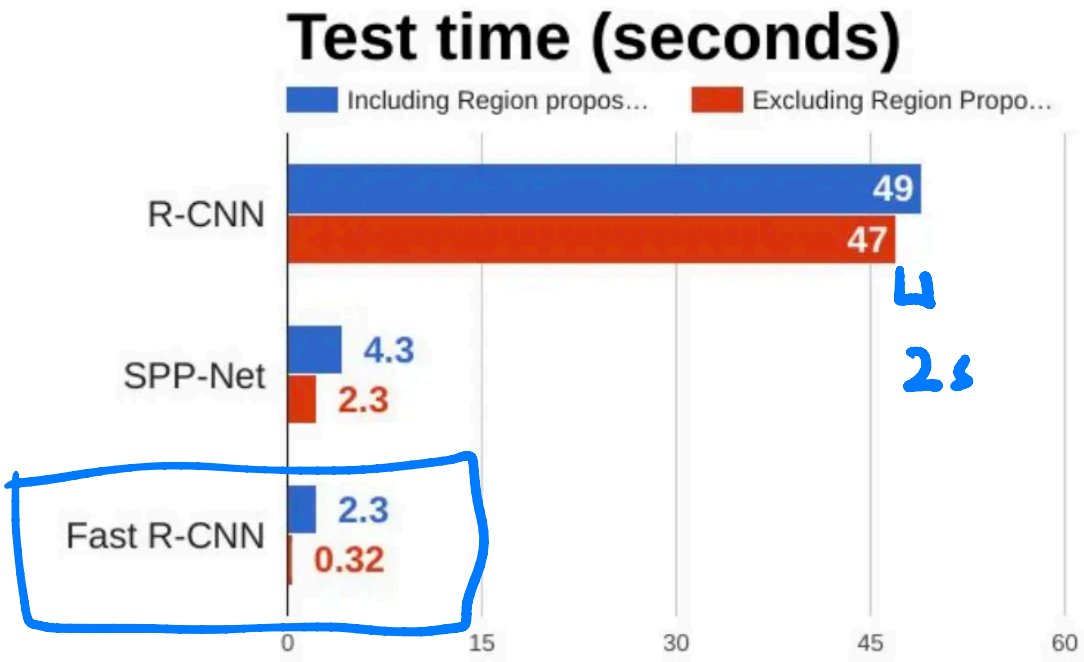


https://arxiv.org/pdf/1504.08083.pdf

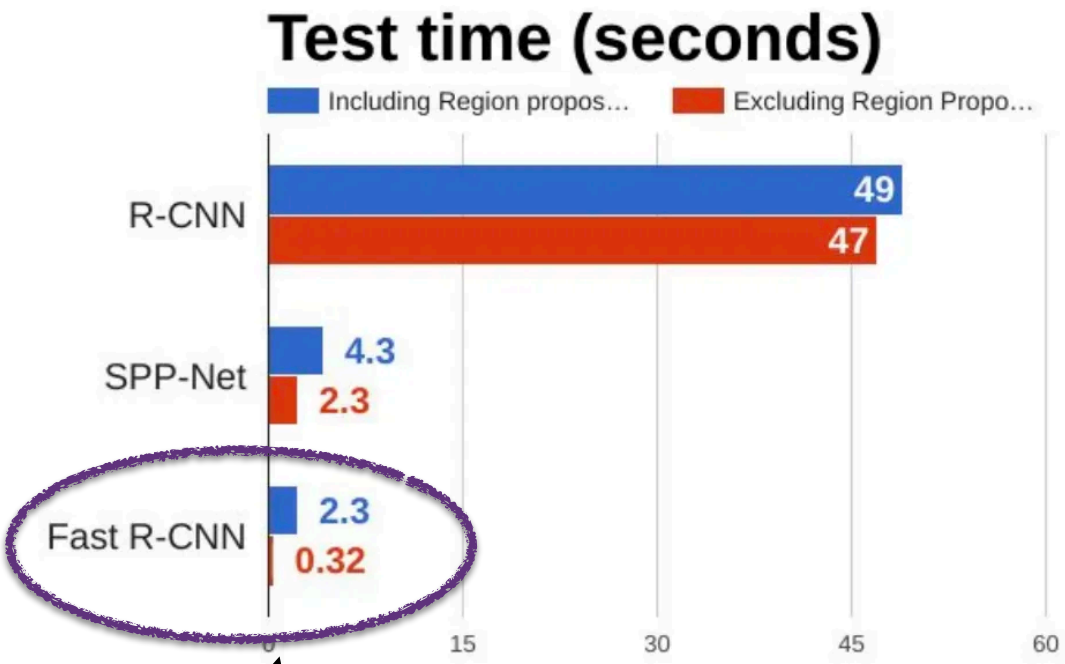
Problem: These features have different sizes, but are passed into a FC network.

Solution: This network uses an RoI pooling layer that makes all the widths and heights 7 x 7 (through max pooling). These then go into the FC net.

Solution 2: Fast R-CNN

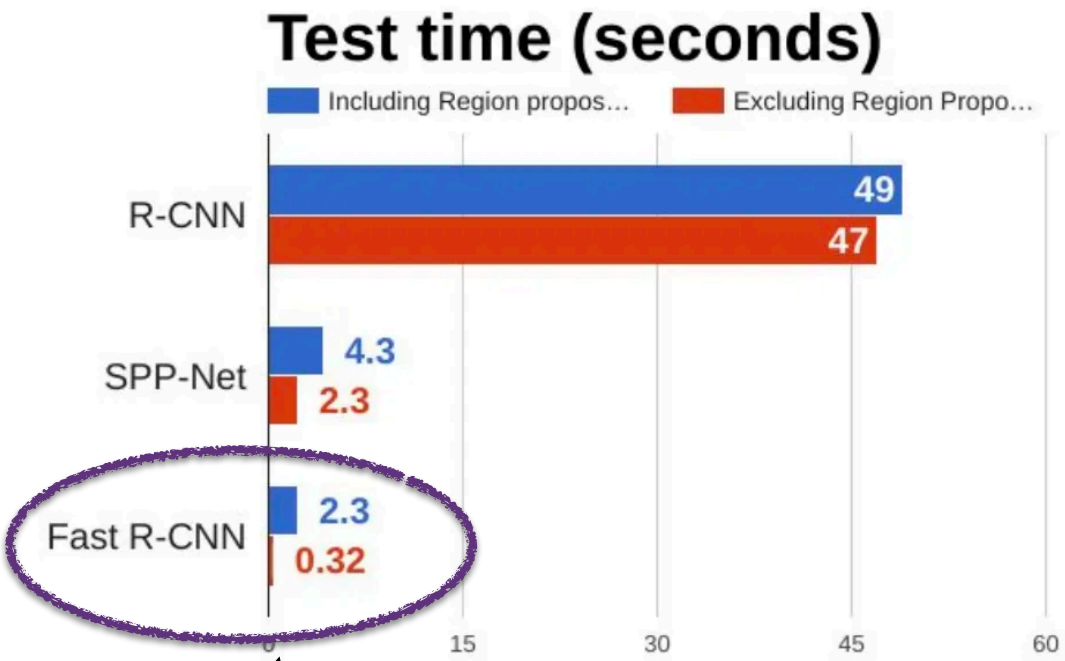


Solution 2: Fast R-CNN



Most of the computation time is from region proposals!

Solution 2: Fast R-CNN



Most of the computation time is from region proposals!

Why not use a neural network to do region proposals?

Solution 3: Faster R-CNN

Why not use a neural network to do region proposals?

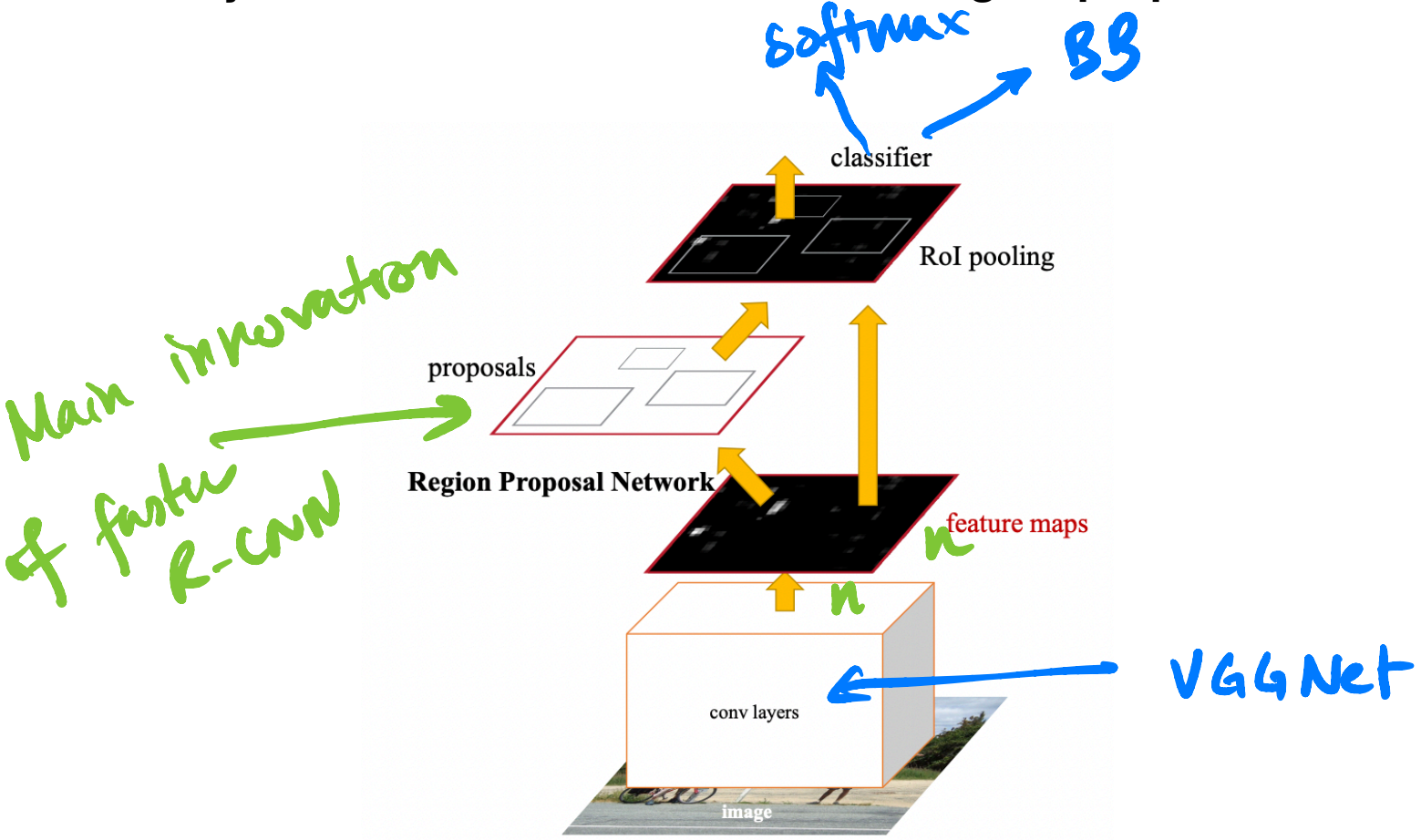


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

https://arxiv.org/pdf/1506.01497.pdf

Solution 3: Faster R-CNN

How does the region proposal network work?

After running a CNN on the input image:

- There is an (n x n) feature map from the initial convolutional layers.
- They extract, using a sliding window, 3x3 windows of this feature map.
  - 3x3 is small, right?
  - Actually, because the effective RF is quite large, this is 171x171 pixels for ZFNet and 228x228 pixels for VGGNet.
- They predict k region proposals for each window. $=9$
  - $2k$ softmax scores (for each of k regions, classify object vs background). $=9$ Binary
  - $4k$ regression outputs (x, y, w, h for each region).
  - They used k=9 for 3 scales and 3 aspect ratios.

https://arxiv.org/pdf/1506.01497.pdf

Solution 3: Faster R-CNN

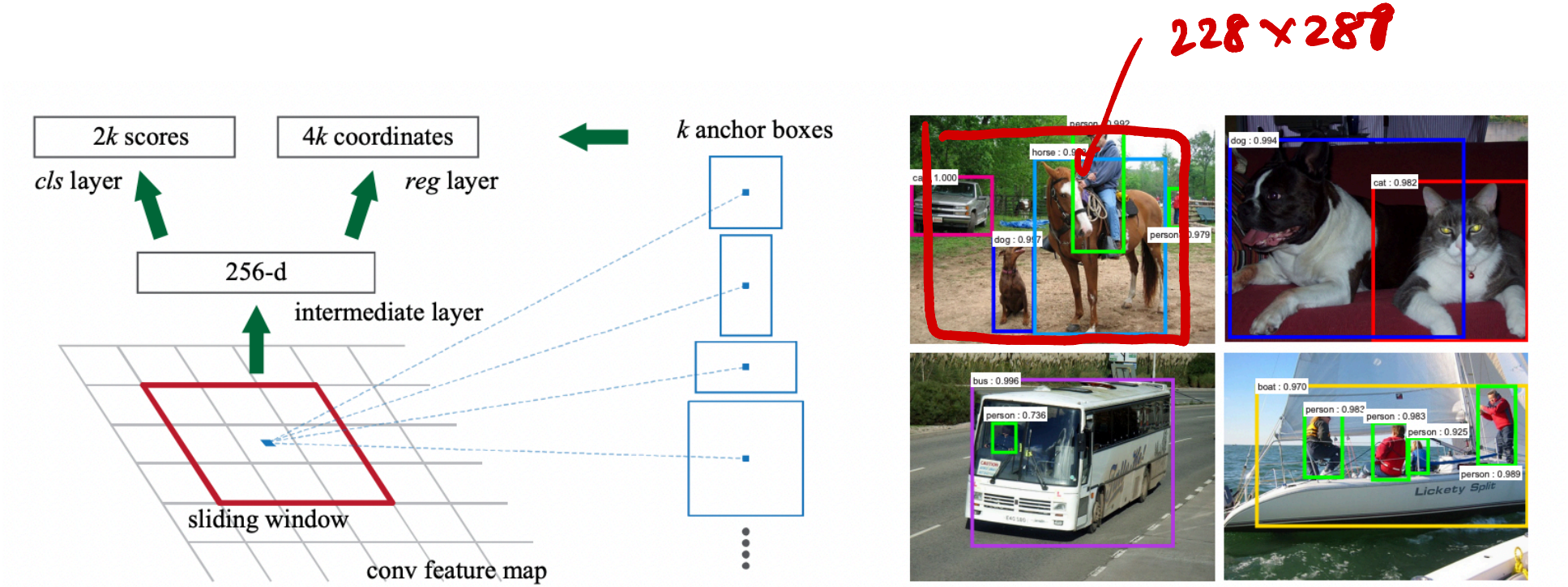


Figure 3: **Left**: Region Proposal Network (RPN). **Right**: Example detections using RPN proposals on PASCAL VOC 2007 test. Our method detects objects in a wide range of scales and aspect ratios.

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*). \quad (1)$$

https://arxiv.org/pdf/1506.01497.pdf

Solution 3: Faster R-CNN

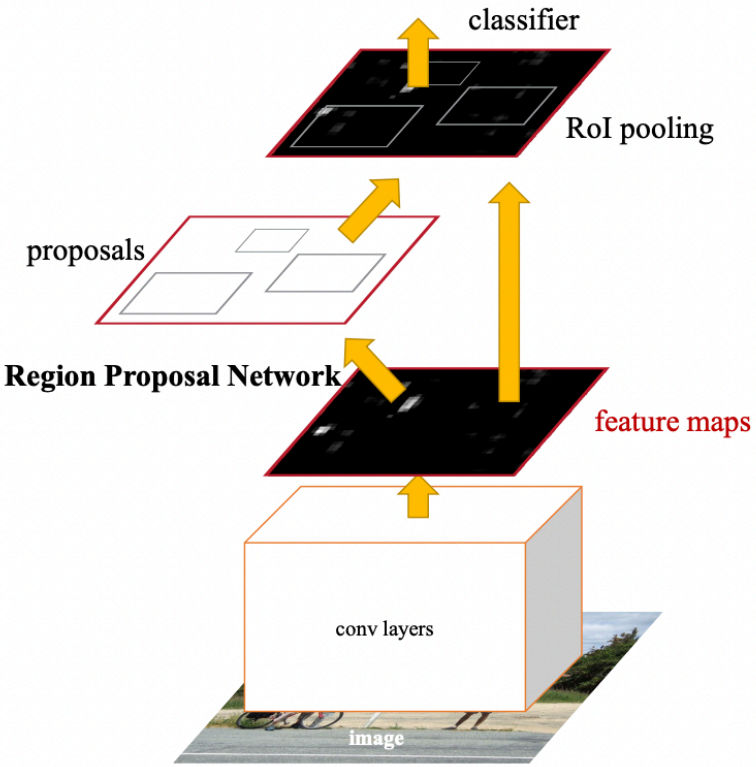


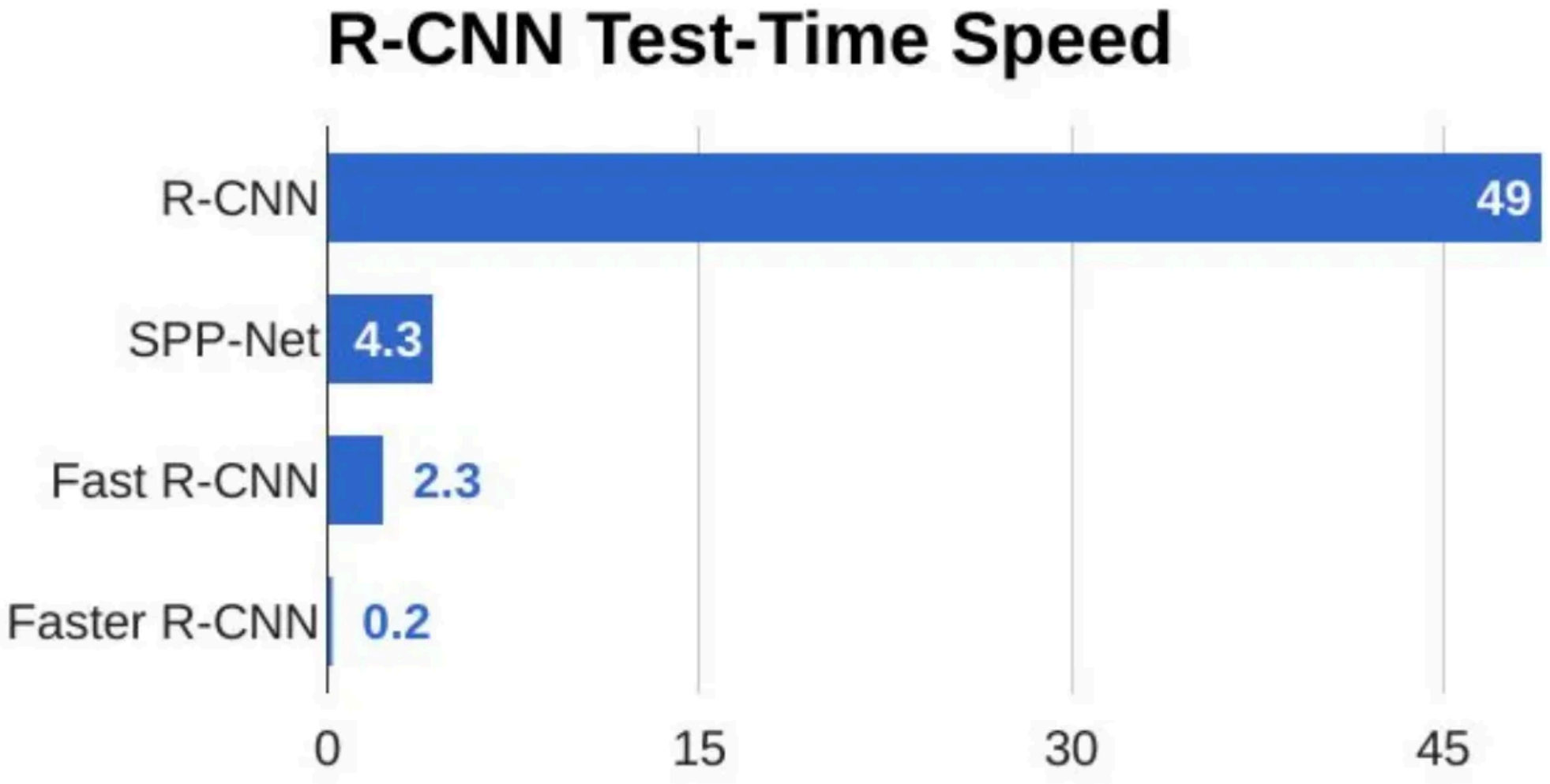
Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

After proposals, they do RoI pooling and classification + bounding boxes, as they also did in Fast R-CNN.

The key difference is the CNN does the region proposals now instead of a separate algorithm.

https://arxiv.org/pdf/1506.01497.pdf

Solution 3: Faster R-CNN



Solution 3: Faster R-CNN

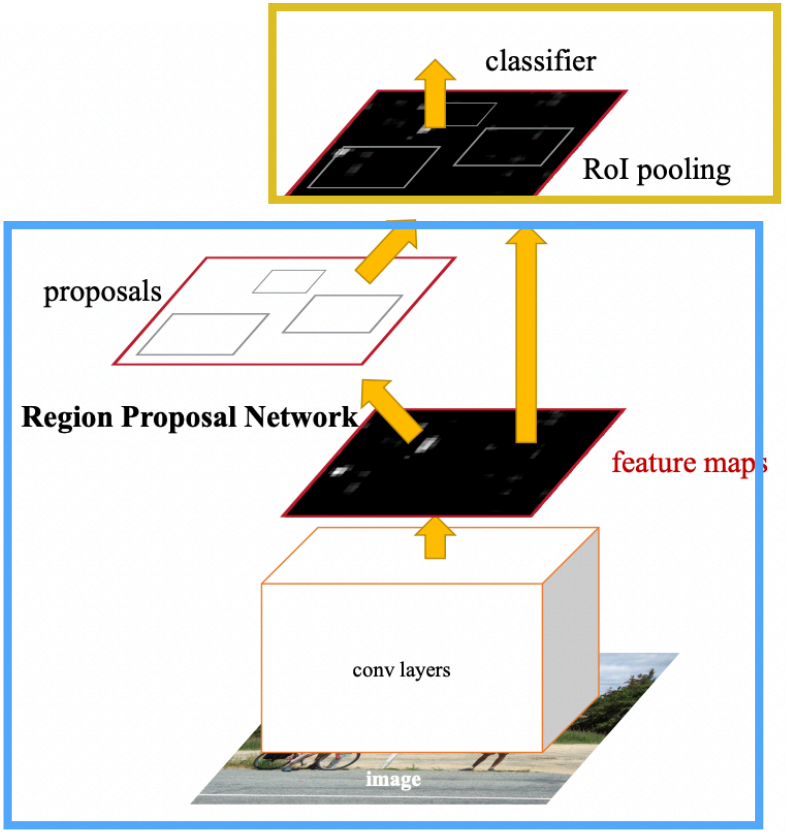


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

Question: What are some suboptimalities (if any) with this architecture? Can we improve on it?

https://arxiv.org/pdf/1506.01497.pdf

Solution 3: Faster R-CNN

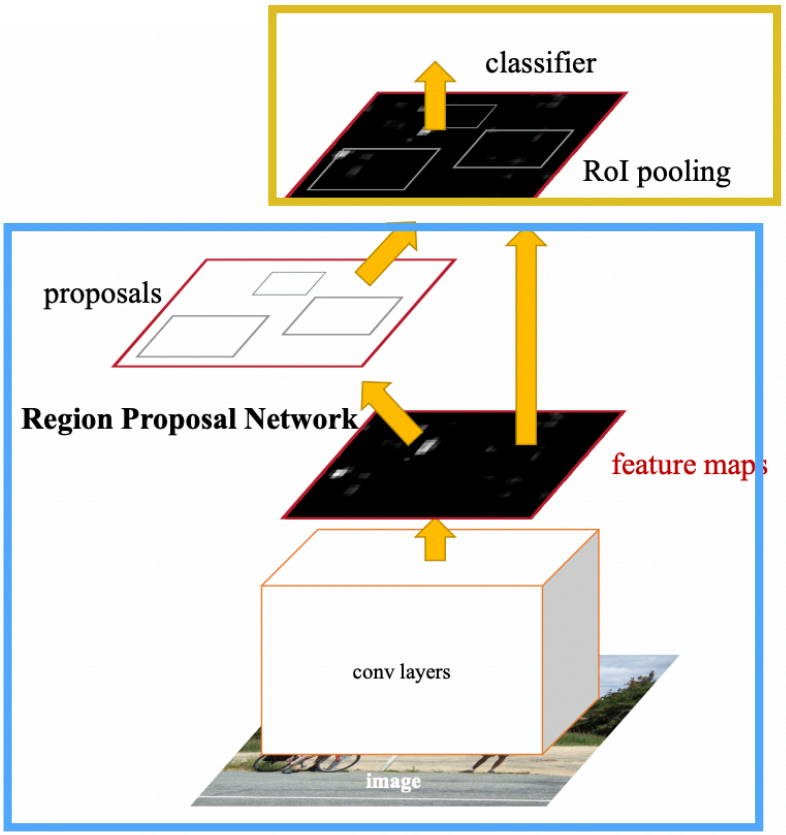


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

Question: What are some suboptimalities (if any) with this architecture? Can we improve on it?

Area for improvement:

Can we turn two stage training into one stage training?

https://arxiv.org/pdf/1506.01497.pdf

YOLO

YOLO — YOU ONLY LIVE ONCE.

YOLO

YOLO — YOU ONLY LIVE ONCE.

LOOK

YOLO

YOLO — YOU ONLY LIVE ONCE.

LOOK

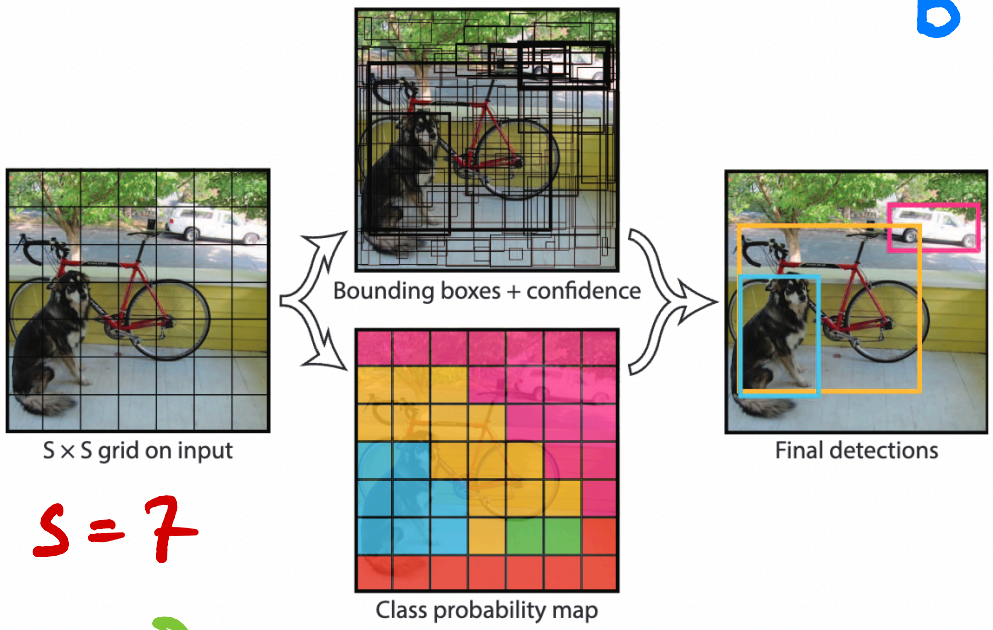
… but you kinda look 49 times.

YOLO

Basic idea: have a single CNN output bounding boxes and their classes. To get around region proposals, we just divide an image into a 7x7 grid, and for each of the 49 grid cells, use a CNN to predict B bounding boxes + their confidences (dx, dy, dw, dh, confidence) and C class probabilities for that box.

B = 2

C = 20



S = 7

7 × 7 × (10 + 20)

Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

https://arxiv.org/pdf/1506.02640.pdf

YOLO

In more detail…

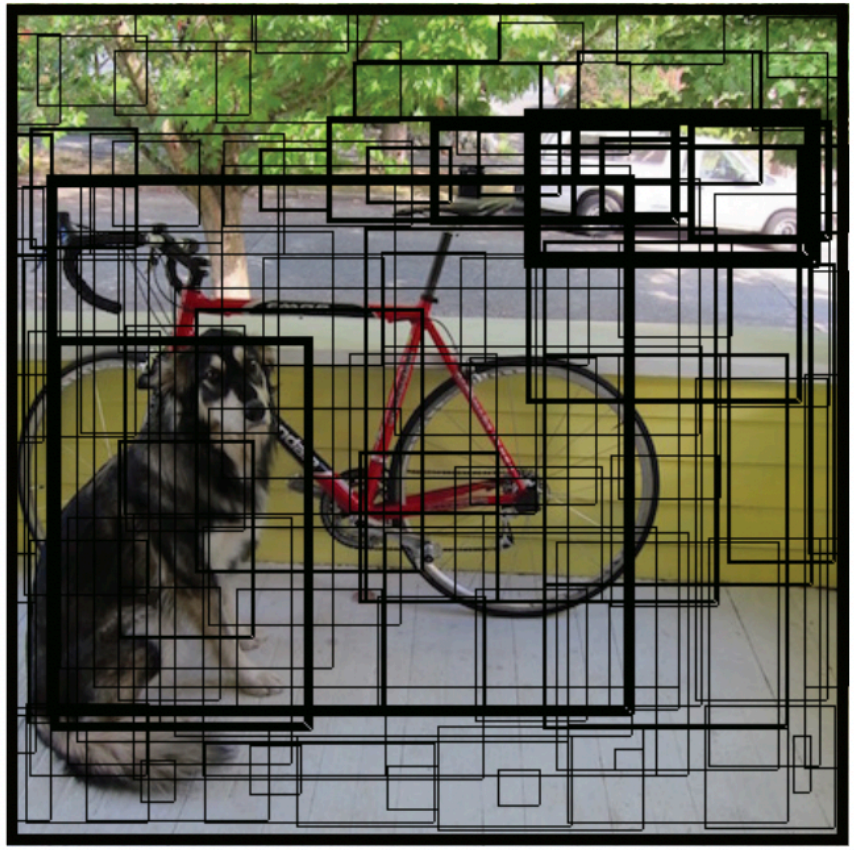
S × S grid on input

Our system divides the input image into an $S \times S$ grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.

S=7

https://arxiv.org/pdf/1506.02640.pdf

YOLO



Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. Formally we define confidence as $\Pr(\text{Object}) * \text{IOU}^{\text{truth}}_{\text{pred}}$. If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth.

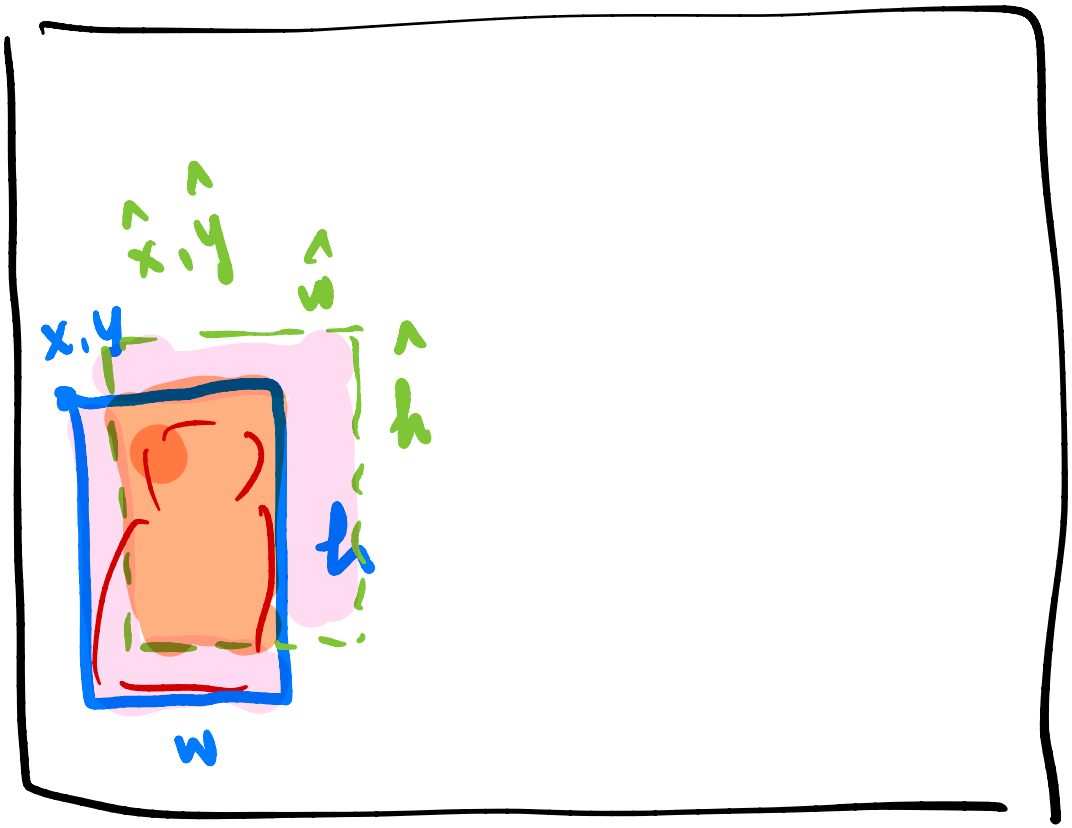
B=2

Binary classification:

Pr(Object) vs Pr(Background)

https://arxiv.org/pdf/1506.02640.pdf

Brief aside: Intersection over Union

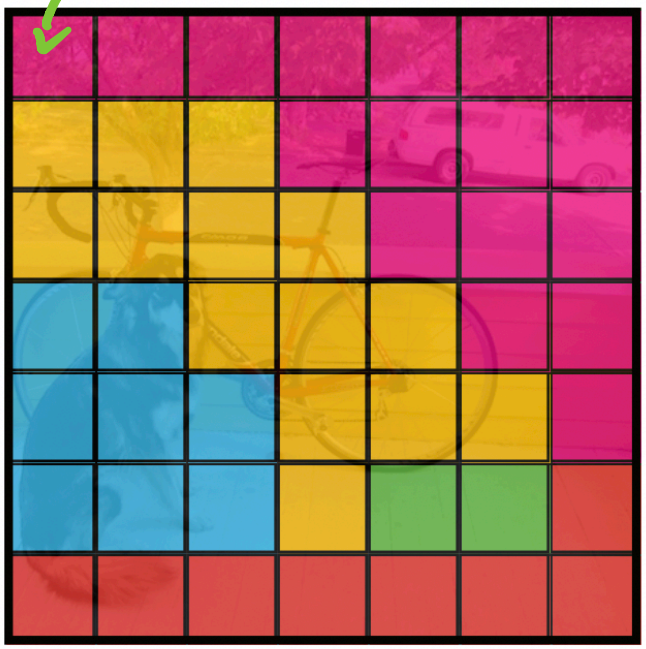


$$IoU = \frac{\text{intersection}}{\text{union}}$$

YOLO

P(class | object)

20 D softmax distribution



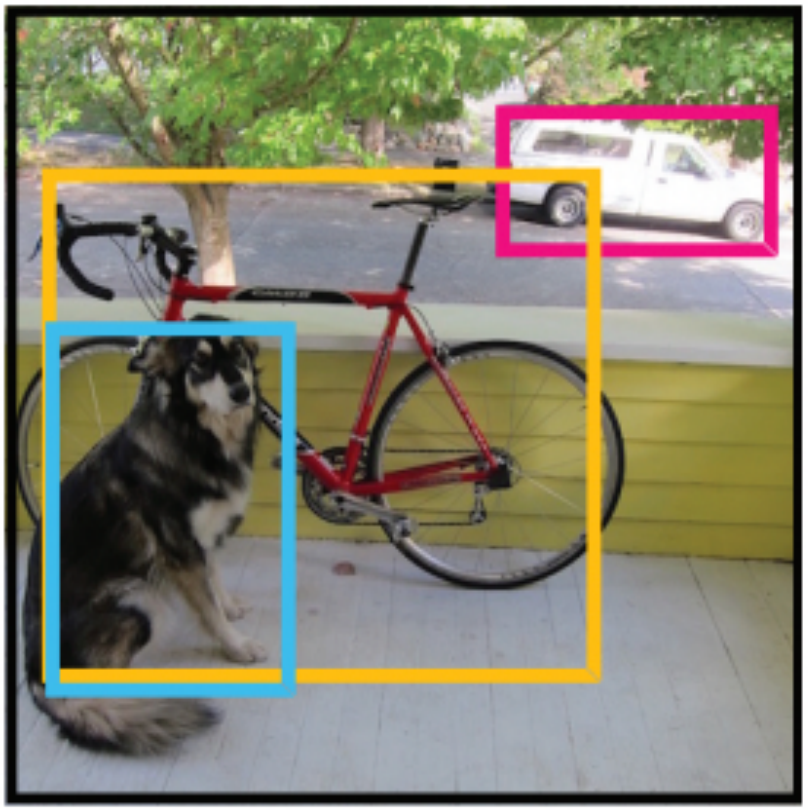
Class probability map

C = 20

Each grid cell also predicts C conditional class probabilities, $\Pr(\text{Class}_i|\text{Object})$. These probabilities are conditioned on the grid cell containing an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes B.

https://arxiv.org/pdf/1506.02640.pdf

YOLO



Final detections

At test time we multiply the conditional class probabilities and the individual box confidence predictions,

$$\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}} \quad (1)$$

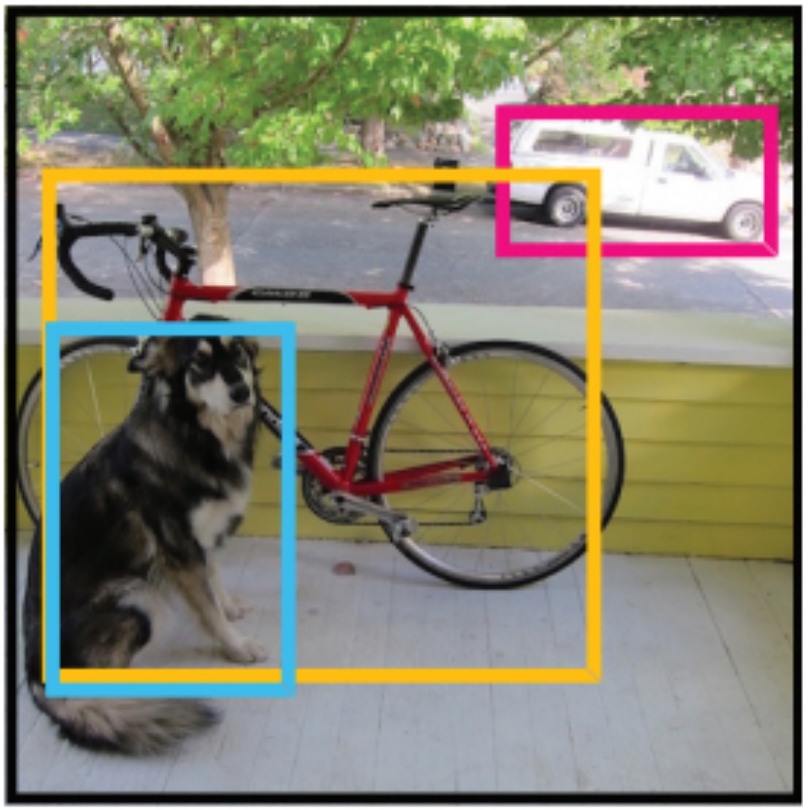
which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

2D

Conf.

https://arxiv.org/pdf/1506.02640.pdf

YOLO



Final detections

At test time we multiply the conditional class probabilities and the individual box confidence predictions,

$$\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}} \quad (1)$$

which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

For evaluating YOLO on PASCAL VOC, we use $S = 7$, $B = 2$. PASCAL VOC has 20 labelled classes so $C = 20$. Our final prediction is a $7 \times 7 \times 30$ tensor.

https://arxiv.org/pdf/1506.02640.pdf

YOLO

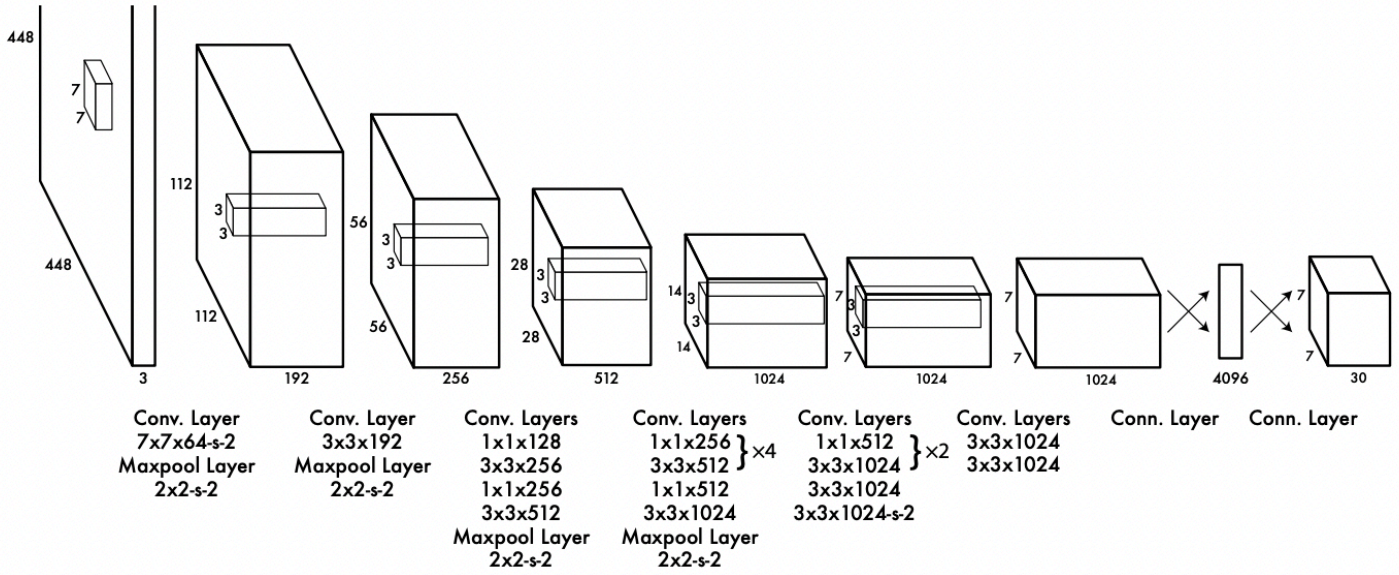


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

1x1 convolutions inspired by GoogLeNet.

https://arxiv.org/pdf/1506.02640.pdf

YOLO

Loss function:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

regression losses x, y, w, h

obj. vs BG

class probs.

https://arxiv.org/pdf/1506.02640.pdf

Non-maximal suppression

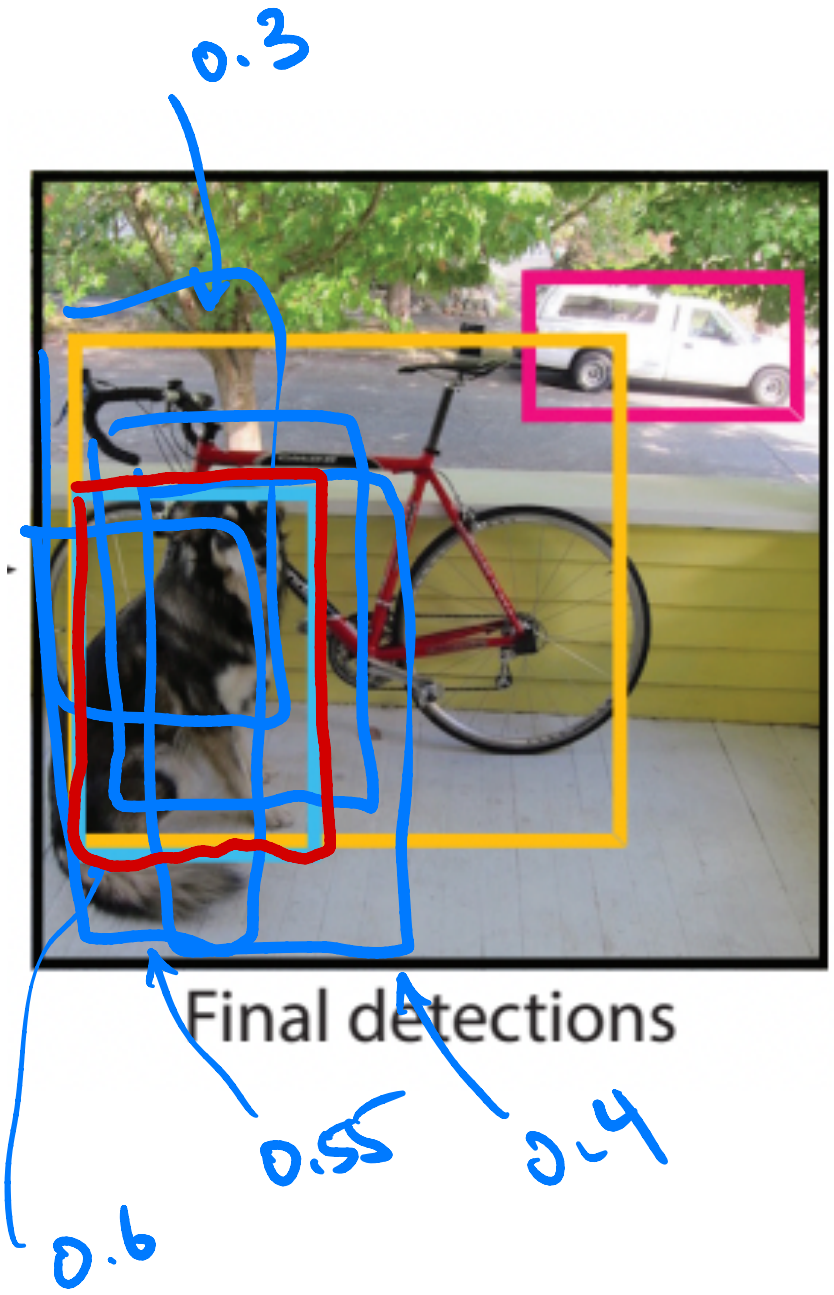
After training, at inference time, the class probabilities and bounding boxes for every grid cell are computed. Which ones are selected? This is via **non-maximal suppression**.

BB: $p_c = p(\text{obj.}) \cdot p(\text{class} \mid \text{obj.})$

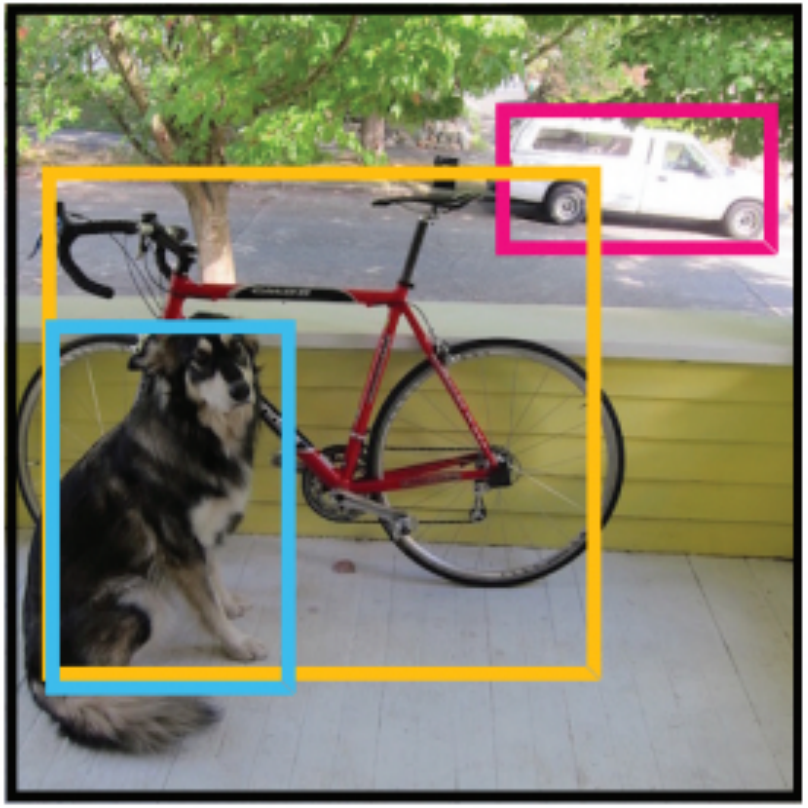
Threshold: discard all BBs with $p_c < 0.5$

Remaining BBs: I pick the BB w/ the highest p_c, then I throw away BB's w/ IOU ≥ 0.5 with the BB having highest p_c.

Sample YOLO outputs



Instance segmentation?



Final detections

Instance segmentation?

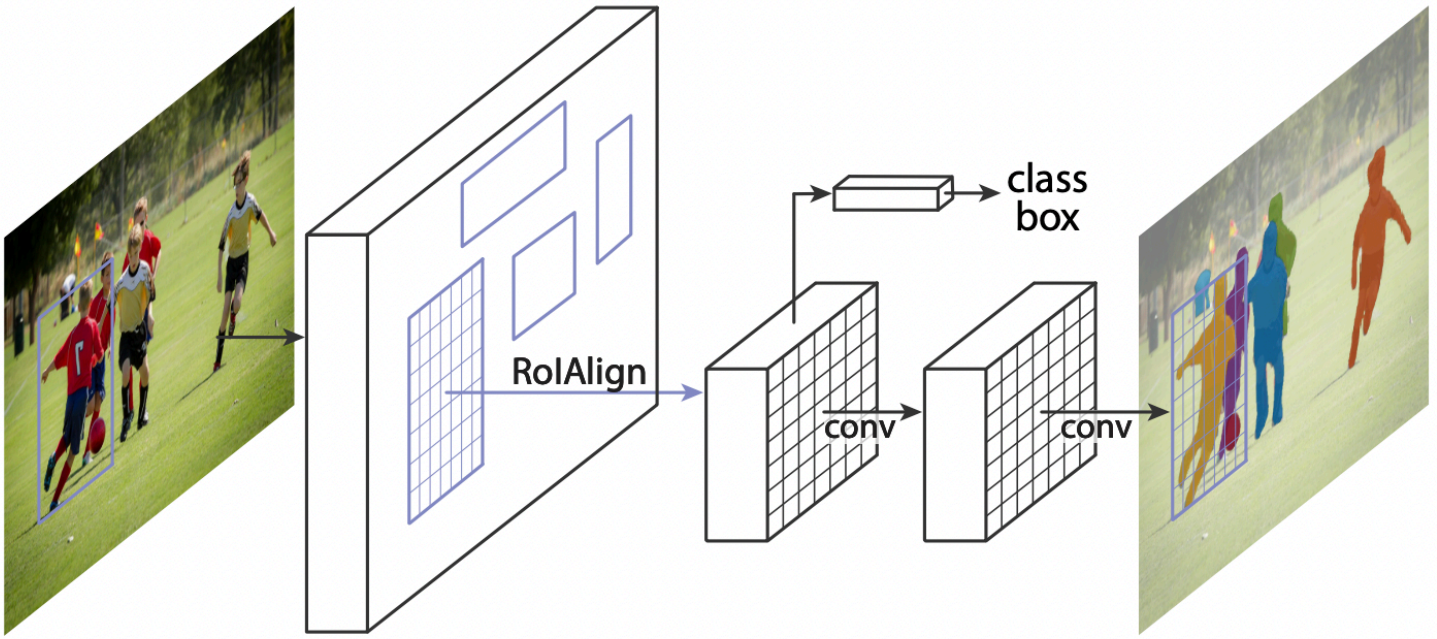
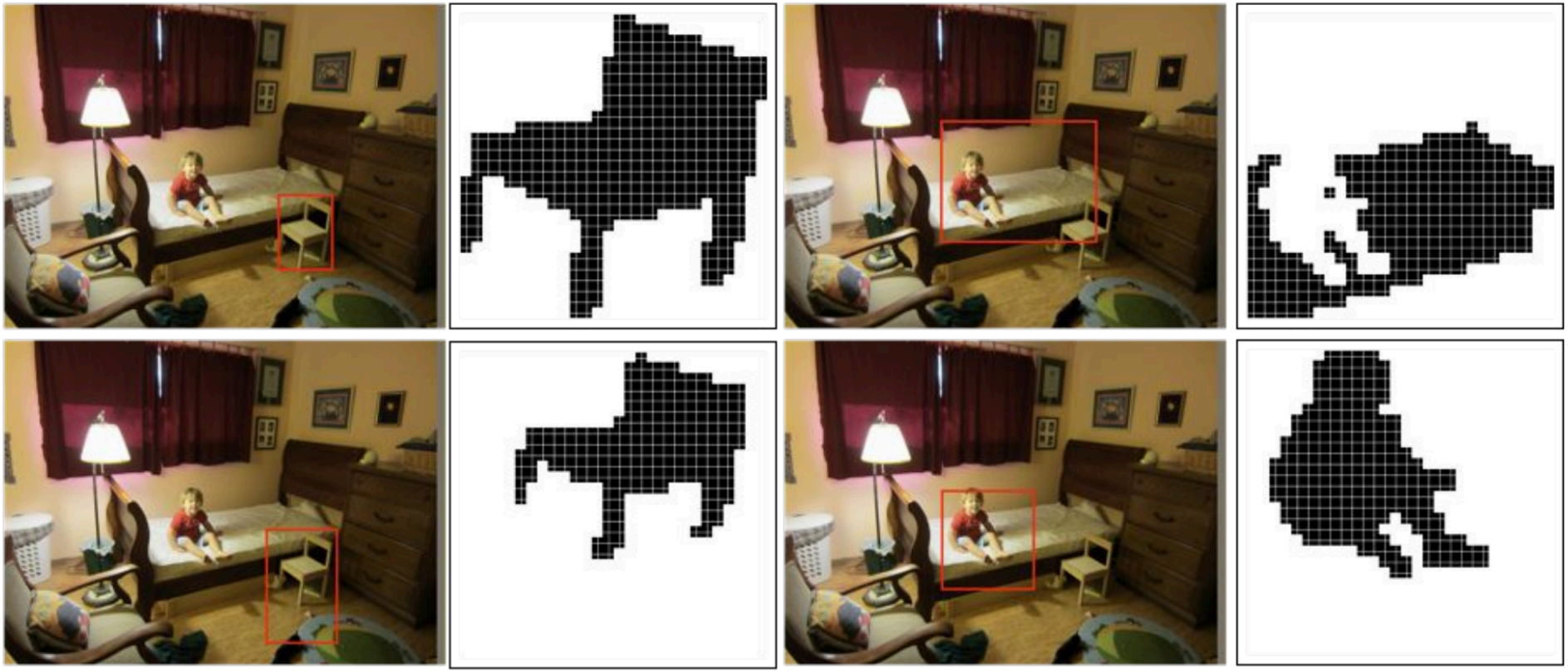


Figure 1. The **Mask R-CNN** framework for instance segmentation.

https://arxiv.org/pdf/1703.06870.pdf

Instance segmentation?



https://arxiv.org/pdf/1703.06870.pdf

Instance segmentation?

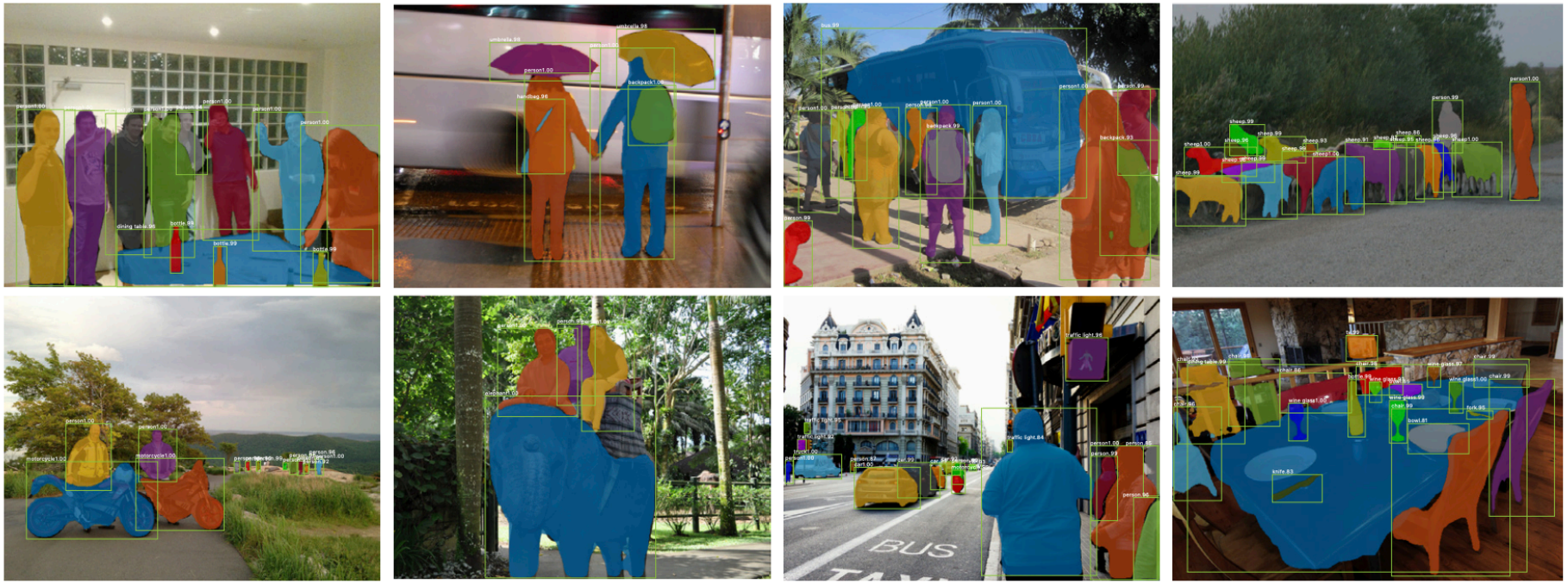
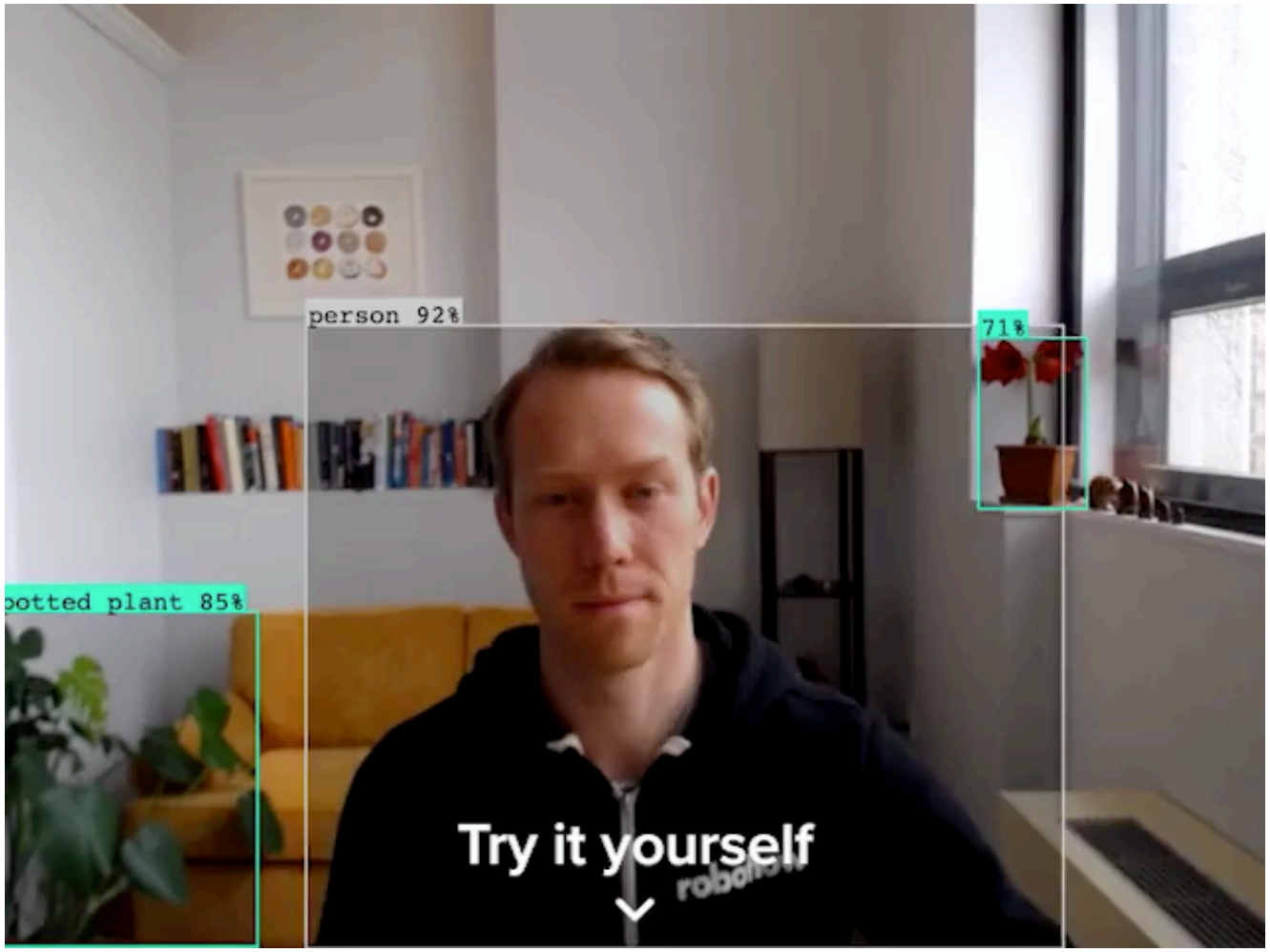


Figure 2. **Mask R-CNN** results on the COCO test set. These results are based on ResNet-101 [19], achieving a *mask* AP of 35.7 and running at 5 fps. Masks are shown in color, and bounding box, category, and confidences are also shown.

https://arxiv.org/pdf/1703.06870.pdf

AI is pretty good at this today



https://yolov8.com/

A video from my lab

Adversarial examples

In this last topic of lecture, we'll discuss adversarial examples and neural network susceptibility to them.

- We've heard a lot of success stories of neural networks.
- But one notable area where results are more discouraging (hence, an active area of research) is that of handling adversarial examples.
- In short, it's not difficult to attack or fool a neural network into misclassifying results.
- And there are some characteristics of these attacks that are especially worrying.

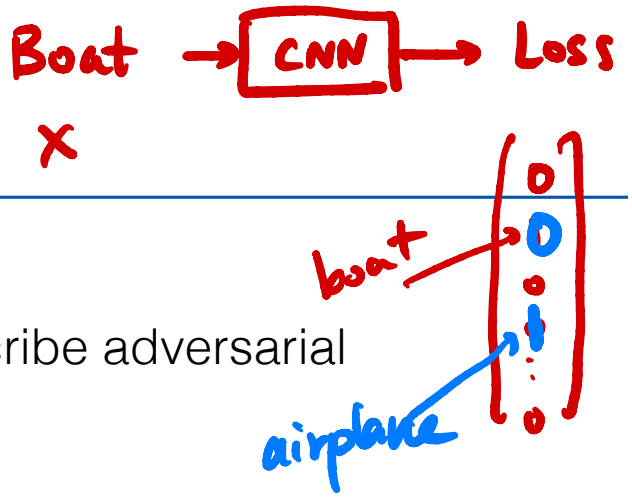
Adversarial examples

Many machine learning models (not just neural networks) are vulnerable to adversarial examples. In this lecture, we'll focus on adversarial examples in neural networks.

- Adversarial examples are examples that are specifically tailored so that the neural network misclassifies them.
- These specifically tailored images are only slightly different from correctly classified examples.
- In many cases, adversarial and non-adversarial examples are indistinguishable to the human eye.

Adversarial examples expose a blind spot in neural networks.

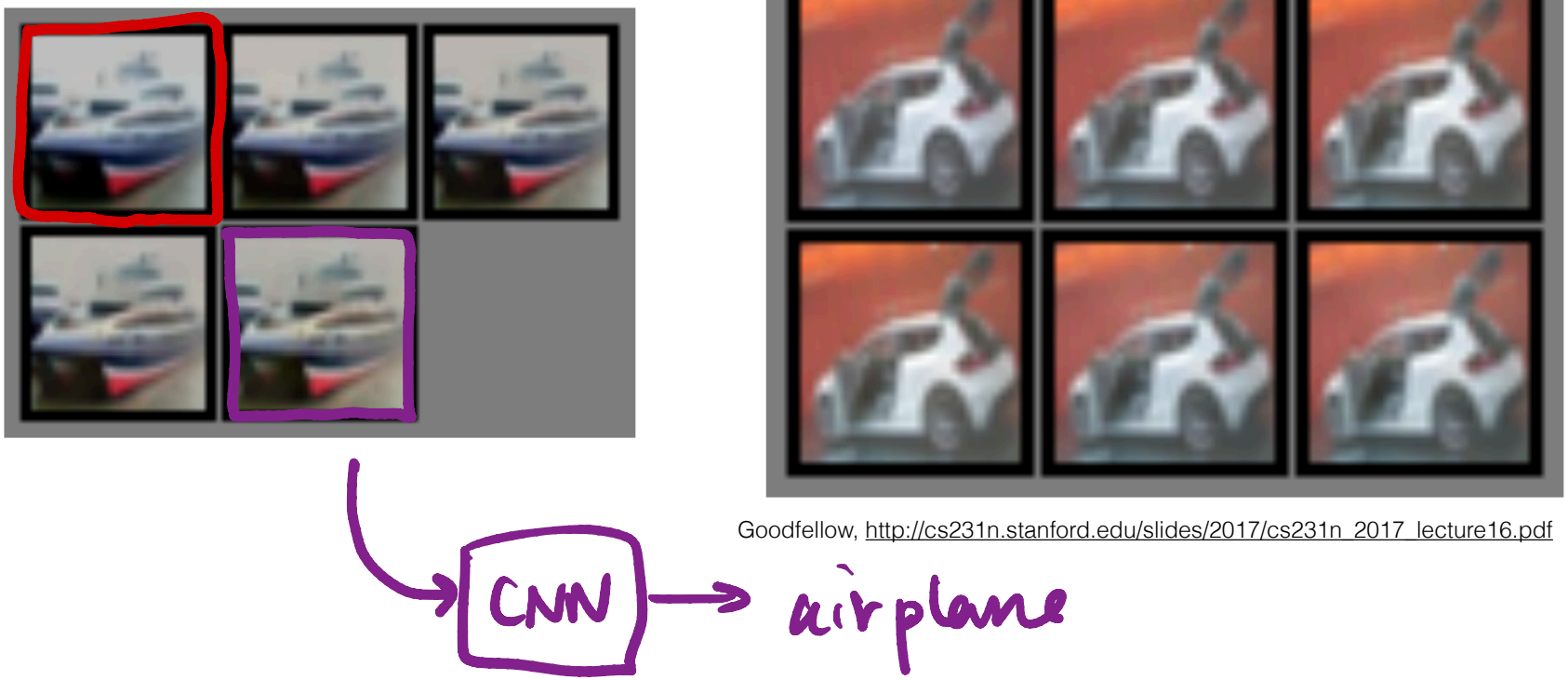
Adversarial examples



Szegedy et al., 2013, was one of the early papers to describe adversarial examples in neural networks.

In these figures below, the experiment was to slowly turn an object into an airplane and see how the network would change the image.

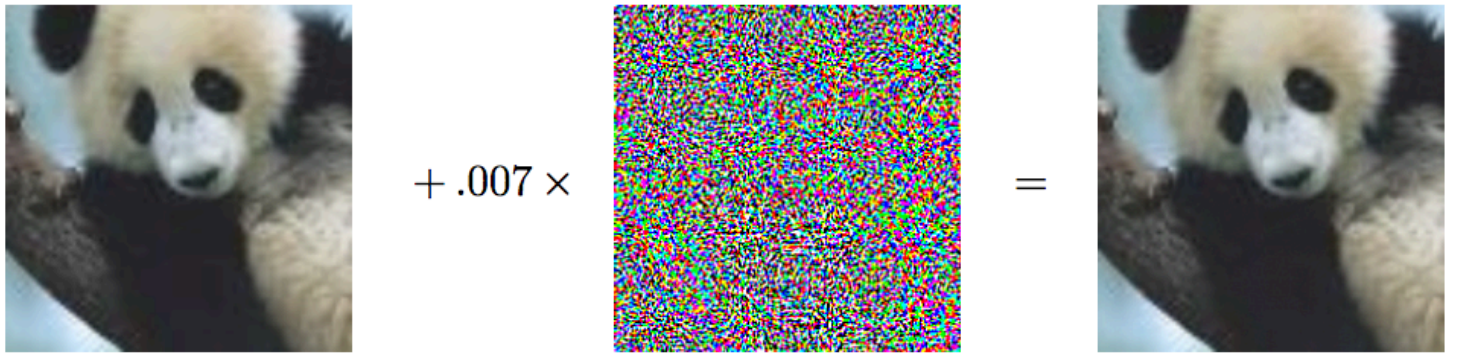
$$x \leftarrow x - \frac{\partial L}{\partial x}$$



Goodfellow, http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture16.pdf

Adversarial examples

Adversarial noise can be designed in such a way that it can add imperceptible noise at the input, but completely alter the network output. Note also that the network is extremely confident that the panda is a gibbon.



$\boldsymbol{x}$

"panda"
57.7% confidence

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
99.3 % confidence

Adversarial examples

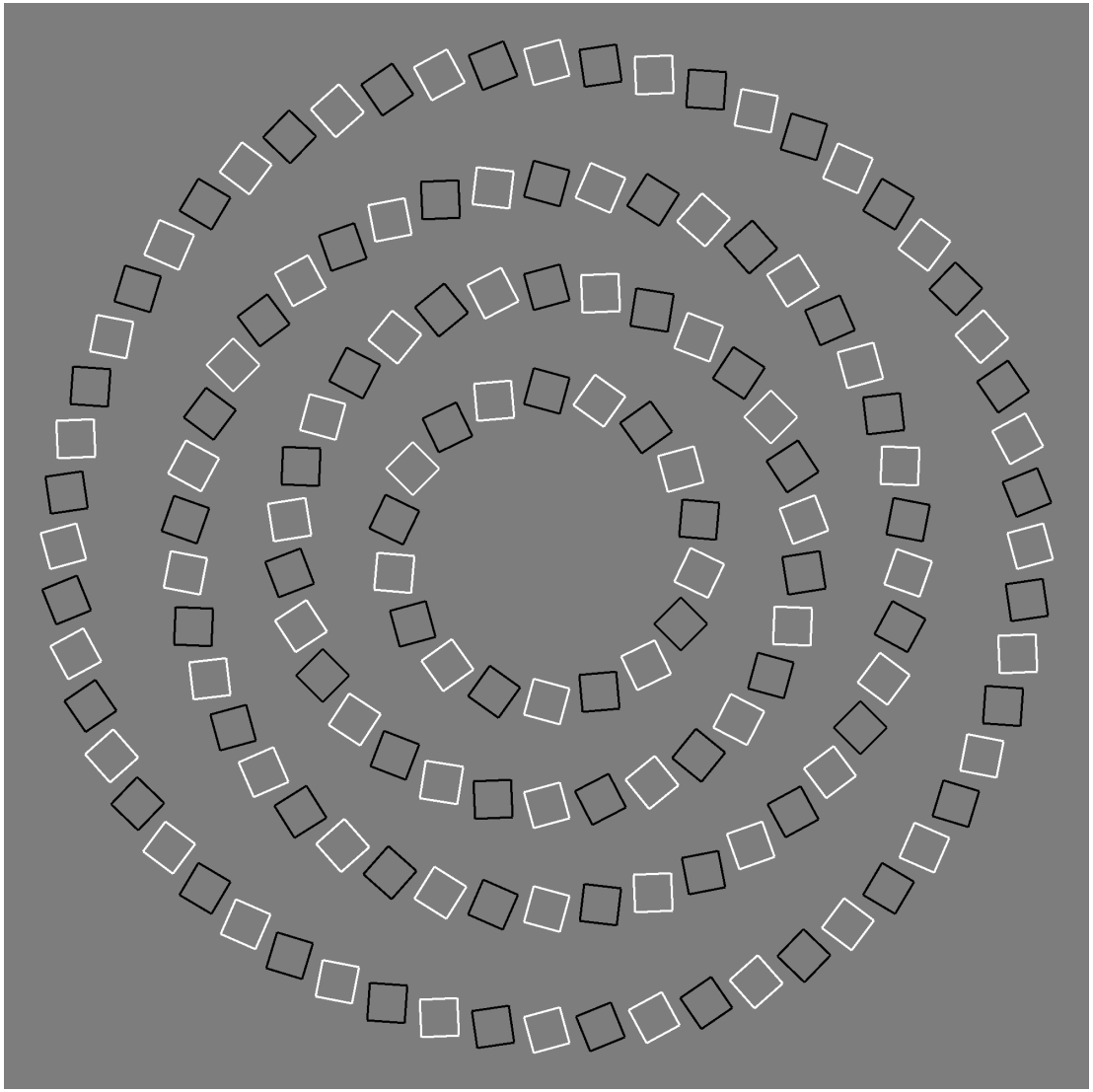
An aside: does the brain have adversarial examples?

Adversarial examples

An aside: does the brain have adversarial examples?

Adversarial examples

An aside: does the brain have adversarial examples?



Pinna illusion

Adversarial examples

This seems like really bad news.

As if that wasn't enough, it gets worse. But before we get there, let's build some intuition for why these examples occur.

Fooling a linear binary classifier

Adversarial example intuition

Let's take the examle of a simple binary classifier using logistic regression. Here, the network computes $z = \sigma(\mathbf{w}^T\mathbf{x} + b)$ and outputs 1 if $z > 0.5$ and 0 otherwise. Consider $b = 0$ and $\mathbf{x}, \mathbf{w} \in \mathbb{R}^{10}$, with the following values:

$$\begin{aligned} \mathbf{w} &= \begin{bmatrix} -5 & 3 & -2 & 2 & -2 & -5 & -4 & 3 & 4 & -4 \end{bmatrix} \\ \mathbf{x} &= \begin{bmatrix} -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1-1 & -1 \end{bmatrix} \end{aligned}$$

With these two settings of $\mathbf{w}$ and $\mathbf{x}$, we have that $z = 0.12$, i.e., it's fairly confident that it is class 0.

Example: say we wanted to add a small perturbation to "fool" the classifier that this is not class 0, but class 1. To do this change, we will allow you to either add 0.1 or -0.1 to every entry of $\mathbf{x}$. How will you modify $\mathbf{x}$?

Fooling a linear binary classifier

Adversarial example intuition (cont.)

$$\mathbf{w} = \begin{bmatrix} -5 & 3 & -2 & 2 & -2 & -5 & -4 & 3 & 4 & -4 \end{bmatrix}$$
$$\mathbf{x} = \begin{bmatrix} -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1-1 & -1 \end{bmatrix}$$

In this example, we want to make $\mathbf{w}^T\mathbf{x}$ less negative, so if $w_i < 0$, then we want to subtract 0.1 from x_i, and vice versa. With this intuition, we transform $\mathbf{x}$ to be:

$$\mathbf{x} = \begin{bmatrix} -1.1 & -0.9 & 0.9 & -0.9 & 0.9 & 0.9 & -1.1 & 1.1-0.9 & -1.1 \end{bmatrix}$$

Under this transformation, $\sigma(\mathbf{w}^T\mathbf{x}) = 0.8$, i.e., the classifier is confident that this is in class 1 instead of class 0.