Neural network programming: object detection

Erik Spence

SciNet HPC Consortium

7 May 2019

Today's slides

You can get the slides for today's class at the SciNet Education web page.

`https://support.scinet.utoronto.ca/education`

Click on the link for the class, and look under "File Storage", and find the "OD.pdf" file.

Today's class

Today's class will cover the following topics:

- Introduction to object detection.
- Region-proposal-based approaches.
- Single-stage approaches.

Please ask questions if something isn't clear.

All neural network architecture images in this class have been borrowed from the original papers.

Computer vision tasks

The application of neural networks to computer vision has undergone a lot of development in a short amount of time. We discussed image classification last class, but there are several types of computer vision tasks. They broadly fall into 4 categories.

- Image classification: what is this thing, when there is only one object in the image?
- Object localization: where is this thing, when there is only one object in this image?
- Object detection: where are the multiple things, and what are they?
- Object segmentation: what are the boundaries of the multiple things, and what are they?

These tasks are roughly in order of increasing difficulty. This class we will examine object detection.

Object detection: motivation

Object detection is the act of determining what things are in a picture, and where they are. As you might imagine, this has wide applications.

- Medical diagnosis: reading X-rays and other visual diagnostics.
- Self-driving cars: reading road signs, not hitting other cars.
- Security: intruder detection.
- Face detection: identifying people, counting people.
- Manufacturing: defect detection.
- Robotics: getting around, object manipulation.
- Science: astrophysics, geography, etc.

And of course, new applications are being developed all the time.

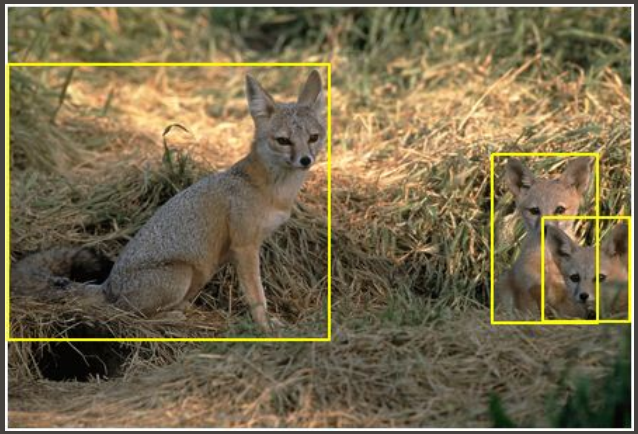
Object detection

Suppose there are multiple objects in an image. The goal of object detection is to tell what there is and where they are.

To achieve this, bounding boxes are usually put around the objects in the image, and the objects are classified.

So now there are two challenges:

- determine the bounding box size and location.
- classify the object within the box.



Object detection: data sets

As you might imagine, once this sub-field was identified as being awesome for competitions, data sets to address the need were developed.

Obviously, if we're going to train neural networks to this level of detail, the dataset sets themselves must have the same level of detail (bounding boxes with object classifications).

These data sets are fewer in number, since they must obviously be extremely large, yet detailed. Examples include:

- Open Images Dataset, 9M URLs of images, thousands of classes.
- PASCAL VOC 2012, 20 classes.
- Cars Dataset, 16K images of cars, 196 makes, models, years.
- COCO data set. Many images with many classes, segmentations and captions.

Mechanical Turk, an aside

You might wonder how these data sets are constructed. Obviously a person needs to go through these 9 million URLs and classify the objects and build the bounding boxes. Right?

The answer is, yes, someone needs to go through these photos one-by-one.

Amazon runs a service called Mechanical Turk. People are paid to do mundane things that machines cannot:

- classify images, build bounding boxes
- review data, remove duplicates
- categorization of data into groups
- translation services

And yes, people have used it to develop scientific data sets.

Object detection: approach types

So, how might you go about solving the object detection problem? There are two major approaches which have been explored:

- region-proposal-based methods: these are two-stage methods, which first propose "regions of interest" (areas which likely contain an object) and then refine the areas and categorize the object within them.
- single-stage methods: these methods treat object detection as a regression problem, proposing bounding boxes and classifications in a single step.

We will examine both families of approaches, and examine the strengths and weaknesses of each.

Region-proposal methods: sliding window

The region-proposal-based object detection methods all depend upon proposing regions which might contain objects. How do you figure out where objects might be in an image?

The first thing you might do is called the "sliding window" approach.

- Make a bunch of crops of the image.
- Run a image-classification CNN on each cropped sub-image, including a category for "background".
- If the CNN classifies something with any confidence, you've got a winner.

But this approach is extremely inefficient, since objects vary in location, size, aspect ratio. You need to calculate and check hundreds of thousands of crops for every image. This is essentially the brute-force method.

Region proposal methods: non-NN techniques

Instead of using brute force to find our objects, let's use a more-traditional image-processing technique to determine where to look for objects.

How do these algorithms work?

- These algorithms essentially find "blobs" in the image that are likely to contain something.
- The "Selective Search" algorithm, for example, uses graph-based segmentation to generate various regions, these are then agglomerated into larger regions of interest.
- Such algorithms can be run fairly quickly, generating about 2000 region proposals in a few seconds.

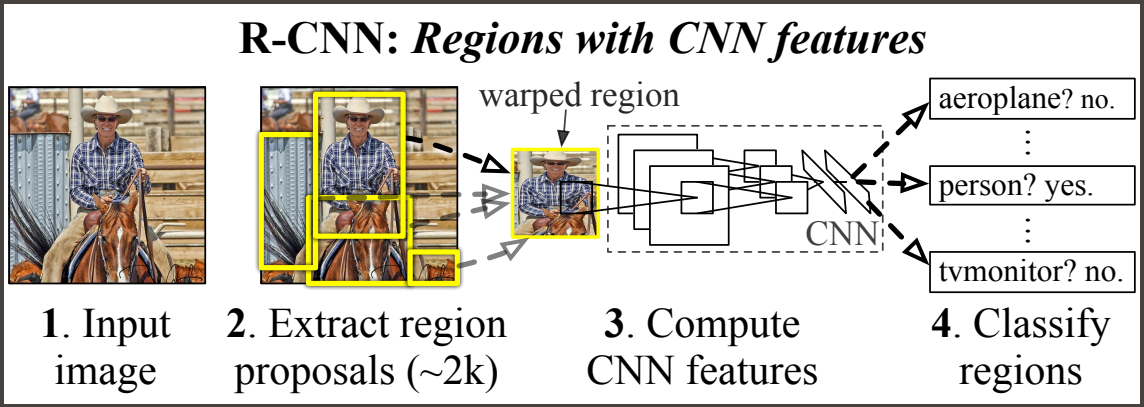
This is a much more intelligent approach than brute force.

R-CNN (2014)

One of the first neural-network-based object detectors which used a region-proposal system was R-CNN (Region CNN). How did it work?

- Run the input image through a region-proposal algorithm (Selective Search, for example).
- The regions of interest are then warped into a standard size, since the CNN only takes a standard-size input.
- Run the warped images through a classification CNN (a modified AlexNet), and use an SVM to do the classification.
- Run a correction to the bounding box.

This scored 53.3% mAP (mean Average Precision) on the PASCAL VOC 2012 data set (30% better than the previous best result).


R-CNN: Regions with CNN features
warped region
CNN
aeroplane? no.
person? yes.
tvmonitor? no.
1. Input image
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions

R-CNN, problems

Good first attempt, but there are certainly weaknesses to this model.

- Warping the region-proposals into a fixed input shape can reduce classification accuracy.
- Training takes a long time (over 3 days).
  - First the CNN is trained on the region proposals.
  - Then the softmax layer is thrown away and replaced by an SVM, and trained.
  - Then the bounding boxes are trained.
- Part of the reason the training was slow was because information was stored on hard drives during the training.
- Inference (forward pass of an image) is very slow (47 seconds per image).

This can be improved upon. Much of the slowness of the inference is due to the fact that 2000 regions of interest need to be run through the CNN.

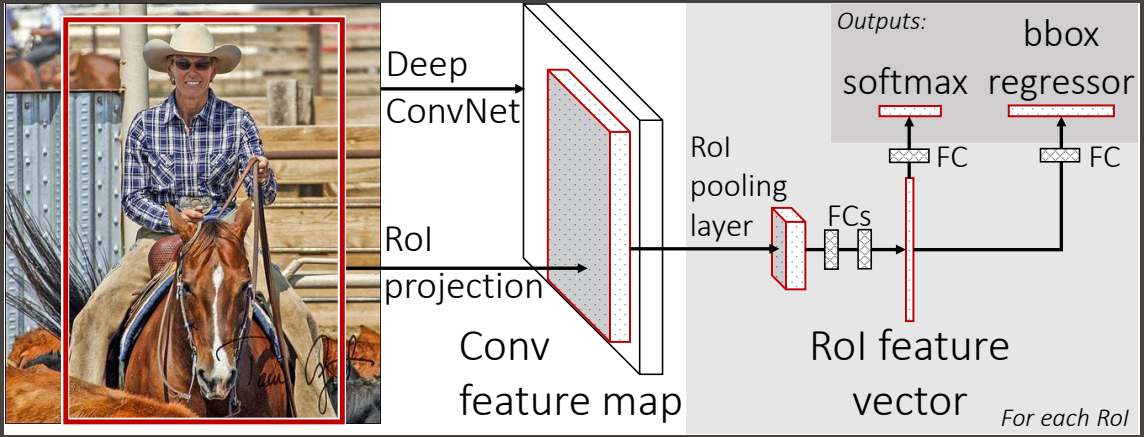
Fast R-CNN (2015)

Many of the problems with R-CNN were fixed in Fast R-CNN.

- We still run the image through a region proposal algorithm.
- But instead of running every (warped) proposed region through a CNN, instead run the whole image through a CNN, and then crop the proposed regions directly from the feature maps.
- Instead of warping images, use a "RoI Pooling" layer, which is a layer which can take arbitrary-size regions and collapse them into a standard input size.
- Fast R-CNN uses a multi-task loss function, which allows the whole network to be trained end-to-end. This leads to significant speedup in training.
- In fact, the speedup in the network inference is such that most of the computation time is spent calculating the regions of interest.

We only run through the CNN once. This makes things much faster.

Fast R-CNN, architecture



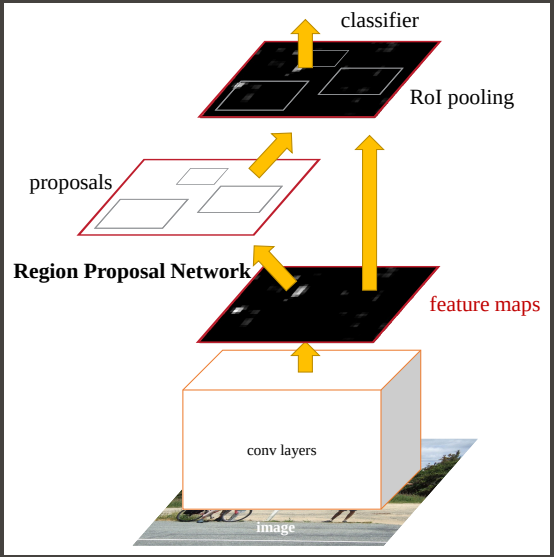
Faster R-CNN (2016)

But we can do even better. Introducing Faster R-CNN!

- As we said before, the main bottleneck in Fast R-CNN was the region proposal function.
- Faster R-CNN makes the region proposal part of the neural network, by adding another section of CNNs, which are separately trained.
- This results in 4 different losses which need to be balanced.
- Difficult to train, but fast!

This resulted in state-of-the-art accuracy at 5 FPS.

Faster R-CNN, architecture



R-CNN family, others

This list of region-proposal-based object detection methods is not complete. There are others that participated in the development of this line of research:

- SPP-net (2015): introduced the "spatial pyramid pooling" layer, thus removing the need to warp proposed regions into a specify shape.
- R-FCN (2016): remove all fully-connected layers and go full CNN.
- FPN (2017): feature pyramids are introduced, improving scale invariance. This is the starting architecture of RetinaNet, which we'll visit in a few slides.
- Mask R-CNN (2017): adapt the Faster R-CNN approach to image segmentation.
- TridentNet (2019): current state of the art. Based on Faster R-CNN, but uses multiple branches of dilated convolutional layers to improve the scale sensitivity.

There isn't time to go every innovation that has been developed. Most of these networks build on previous work, making modifications along the way.

Single-stage methods

Two-stage object detection methods first propose regions which may contain objects, and then go through a refinement stage. Single-stage methods take a different approach.

- These methods simultaneously predict bounding boxes, confidence levels and categories.
- There are many players:
  - YOLO (You Only Look Once) versions 1 (2016), 2 (2017), and 3 (2018).
  - SSD (Single-Shot Detector, 2016).
  - RetinaNet (2018).
  - And many many variations of the above, especially SSD.
- These networks are much faster at processing images than the two-stage detectors, since there is only one stage of processing, but lag in accuracy.

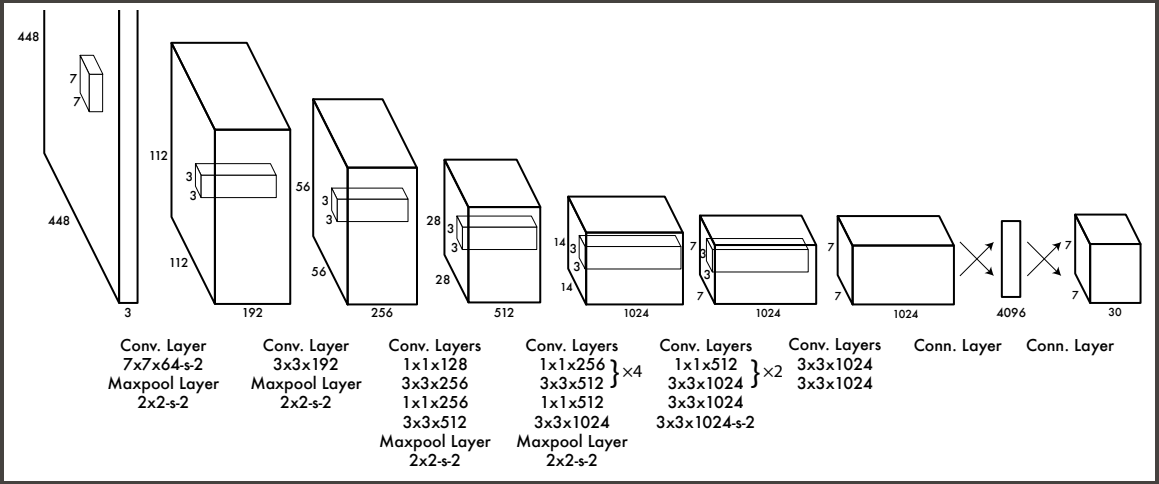
YOLO (2016)

YOLO's most notable feature is it was the first real-time object detector (45fps!).

- As a single-stage network, it consists of a single CNN.
- It simultaneously predicts bounding boxes and classification scores.
- YOLO takes the image, divides it into an S x S grid. In each grid cell, YOLO predicts
  - C class probilities,
  - N bounding boxes, and
  - N bounding box "objectness" scores.
- This network only resolves objects at a single scale, and struggles with objects in new aspect ratios.
- It also struggles with lots of small objects grouped together.

Versions 2 and 3 of the network have also been released, improving the accuracy while maintaining the model's rapidity.

YOLO, architecture



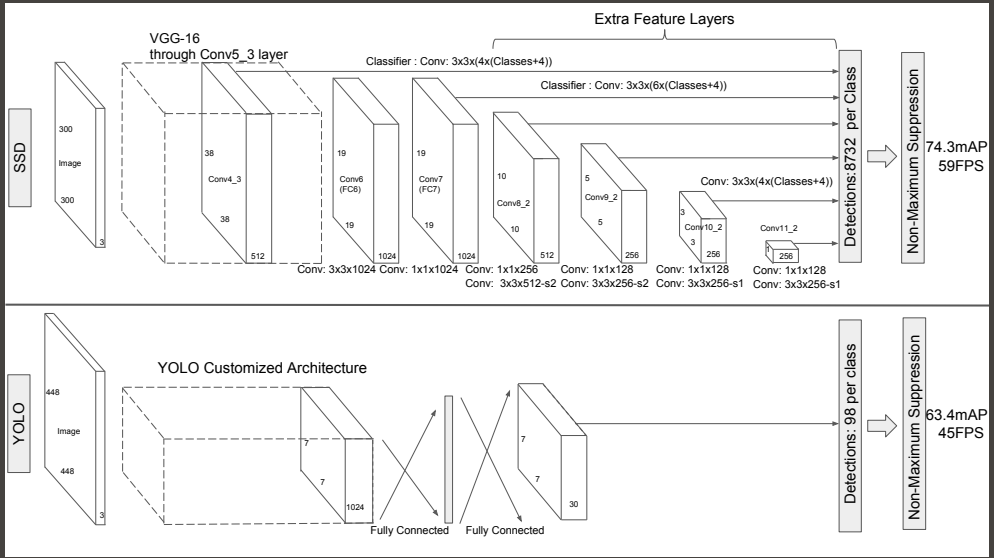
SSD (2016)

Single-Shot MultiBox Detector is particularly interesting, since it has similarities to the region proposal network used in Faster R-CNN.

- Built on top of a base network (VGG in the original version, Res-101 in later versions), but adds further convolutional layers.
- The output of the network's convolutional layers are fed to the subsequent layers, but also to the final layer.
- Because the resolution of the layers decreases, and the output of the layers are fed to the final layer, detections at different scales becomes easier.
- Each subsequent layer is chopped up in sections, and a set of k "anchor boxes" are used to find the bounding boxes.
- "Non-maximum suppression" is used to filter out bounding boxes which overlap and predict the same category.

Since 2016, many variations on SSD which improve performance have been developed.

SSD, architecture



Class imbalance

Single-stage detectors suffer from lower accuracies than two-stage detectors. This is caused by "class imbalance":

- Class imbalance means that the categories which are in the training data are not equally distributed.
- Two-stage detectors generate 1-2k candidate regions in the first stage. Most background images are rejected.
- Single-stage detectors have no way to know where objects are, so many many background objects end up getting examined.
- The vast majority of the "objects" which are studied by these networks are background.
- Hence the class imbalance; the network gets really good at learning what the background looks like.

This obviously leads to problems in training the network to identify real objects.

RetinaNet (2018)

The great contribution of the single-stage detector RetinaNet was to force the network to learn the objects instead of the background.

To accomplish this, a new probability-dependent loss function was introduced, "focal loss":

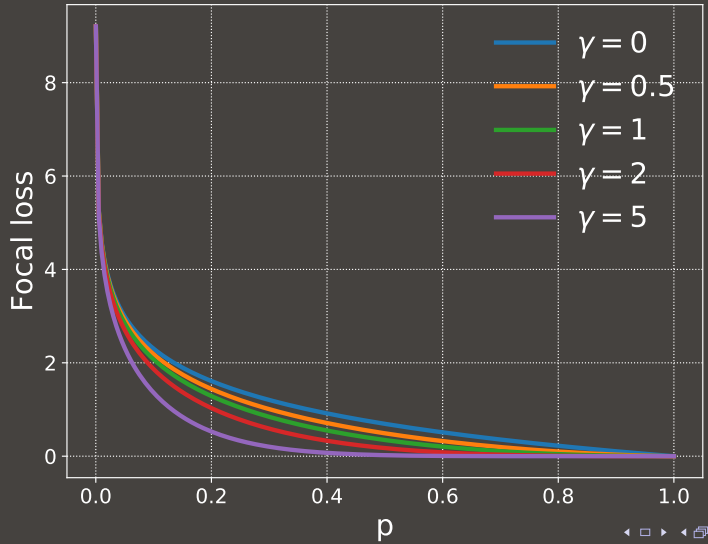
$$p_t = yp + (1-y)(1-p)$$

$$\mathbf{FL}(p_t) = -(1-p_t)^{\gamma} \log(p_t)$$

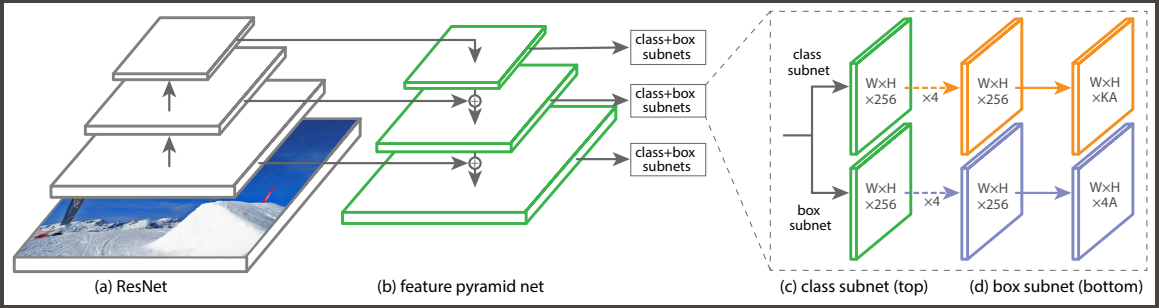
where p is the predicted probability for a given (binary) class, y is the class (0 or 1), FL is the focal loss, and γ is a hyperparameter.

The focal loss is just a modification of the cross-entropy loss, but it penalizes entries which are well-categorized, forcing the training to focus on the hard cases.

Focal loss

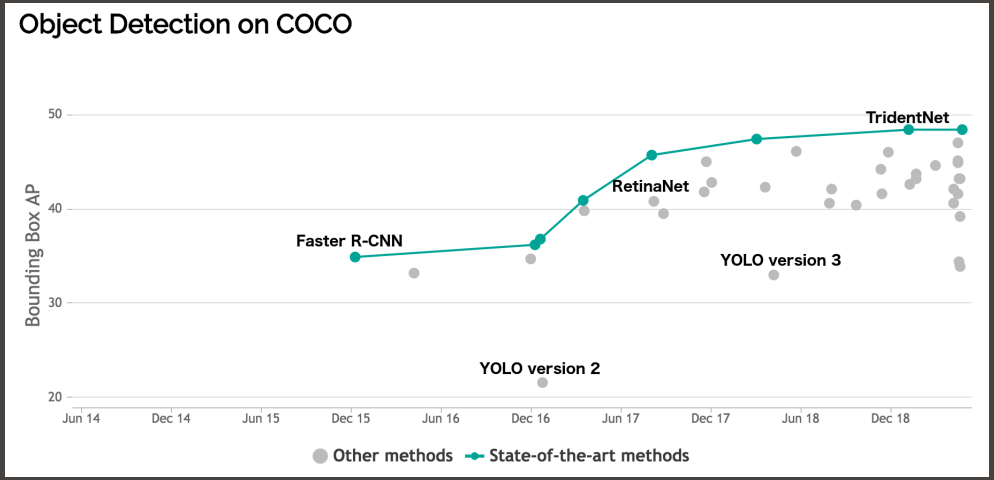

Focal loss
p
0
2
4
6
8
0.0
0.2
0.4
0.6
0.8
1.0
$\gamma = 0$
$\gamma = 0.5$
$\gamma = 1$
$\gamma = 2$
$\gamma = 5$

RetinaNet



RetinaNet uses a "Feature Pyramid Network", gathering information from all scales. It also uses anchor boxes, like SSD.

State of the art



https://paperswithcode.com/sota/object-detection-on-coco

Linky goodness

Single-stage methods:

- Original Yolo: https://arxiv.org/abs/1506.02640.
- SSD: https://arxiv.org/abs/1512.02325
- RetinaNet: https://arxiv.org/abs/1708.02002

Object detection data sets:

- http://ai.stanford.edu/~jkrause/cars/car_dataset.html
- http://host.robots.ox.ac.uk/pascal/VOC/voc2012
- https://github.com/openimages/dataset
- http://cocodataset.org

Linky goodness, continued

Region-proposal-based methods:

- R-CNN: https://arxiv.org/abs/1311.2524
- Fast R-CNN: https://arxiv.org/abs/1504.08083
- Faster R-CNN: https://arxiv.org/abs/1506.01497
- SPP-net: https://arxiv.org/abs/1406.4729
- R-FCN: https://arxiv.org/abs/1605.06409
- FPN: https://arxiv.org/abs/1612.03144
- Mask R-CNN: https://arxiv.org/abs/1703.06870
- TridentNet: https://arxiv.org/abs/1901.01892v1

Reviews:

- https://arxiv.org/abs/1807.05511
- https://arxiv.org/abs/1611.10012