CSEP 576: Object Detection with Convolutional Networks

Jonathan Huang (jonathanhuang@google.com)

University of Washington 17 May 2018

Google AI
Today’s Yesterday’s Image Taggers just returned a bag of words...
Imagenet Progress Over the Years

Source: ImageNet: Where have we been? Where are we going? Fei Fei Li, Jia Deng

Human performance

Number of Entries

Classification Errors (top-5)
Now: boxes, segments, human pose...

Based on a figure from Jia Deng
From Classification to Detection

Detection = Classification + Localization

- Variable # outputs
- Need to classify based on much fewer pixels than in Imagenet setting; Requires context!

Photo credit: Michael Mina
Object Detection Applications
Object Detection Applications
Object Detection Applications

Bus Lane Blocked, He Trained His Computer to Catch Scofflaws

Alex Bell developed a computer program that used a traffic camera to identify how often bus and bicycle lanes were blocked by unauthorized vehicles along one block in Harlem.

Image credit: NYTimes (author: Sarah Maslin Nir)
Object Detection Applications
Today

- Sliding Window Detectors
- Detection with Convolutional Networks
- How to Evaluate a Detector
- Practical tips/tricks
- Variations on a theme (instance segmentation, keypoint detection, video detection, etc... )
“Sliding Window” Detection
“Sliding Window” Detection
“Sliding Window” Detection
“Sliding Window” Detection
“Sliding Window” Detection
“Sliding Window” Detection
“Sliding Window” Detection
“Sliding Window” Detection
“Sliding Window” Detection
“Sliding Window” Detection
“Sliding Window” Detection
“Sliding Window” Detection

background
“Sliding Window” Detection
“Sliding Window” Detection

Compute within-region features, then classify
“Sliding Window” Detection

Typical to enlarge region to include some “context”
Sliding window placement

Slide over **fine grid** in x, y, scale, aspect ratio

Slide over **coarse grid** in x, y, scale, aspect ratio

Slow and Accurate

Fast and Not-so-accurate

(... or can it be?)
Bounding Box Regression

Idea:
Also predict continuous offset from anchor to “snap” onto object
Idea:
Also predict continuous offset from anchor to “snap” onto object
Today

- Sliding Window Detectors
- Detection with Convolutional Networks
- How to Evaluate a Detector
- Practical tips/tricks
- Variations on a theme (instance segmentation, keypoint detection, video detection, etc...)
Using convolutional networks for detection

Agenda for next few slides:
- Cover a simplified convnet approach for generating detections in detail;
- Touch on more modern architectures (all of which are based on the same concept)

- Extract features at sliding window positions via convolution
- Deep networks -> large receptive fields that can account for context
Think of each feature vector $v_{ij}$ as corresponding to a sliding window (anchor).

Category score = $\text{SoftMax}(W_{cls} \cdot v_{ij})$

Offset from anchor = $W_{loc} \cdot v_{ij}$

Use the same $W_{loc}$ and $W_{cls}$ for all $i, j$ in anchor grid.

Anchors assumed to be:
- of the same shape, and
- contained and centered in receptive field
Target Assignment

Step 1: Match anchor boxes to groundtruth boxes (based on Euclidean distance or overlap area)

Step 2: Give each anchor a classification and regression target
  - If anchor has no matching groundtruth, it classifies as 0 and no regression target is given

Class target matrix (one entry per anchor)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Location targets (only for matched anchors)

\[
\begin{align*}
g_{\text{xmin}} & - \text{anchor}_{\text{xmin}} \\
g_{\text{ymin}} & - \text{anchor}_{\text{ymin}} \\
g_{\text{xmax}} & - \text{anchor}_{\text{xmax}} \\
g_{\text{ymax}} & - \text{anchor}_{\text{ymax}}
\end{align*}
\]
Typical Training Objective

**Per-anchor Loss:**

\[ L(\text{anchor } a) = \alpha \cdot \delta(a \text{ has matching groundtruth}) \cdot L_2(t_{loc}^i, W_{loc} \cdot v_i^j) + \beta \cdot \text{SoftMaxCrossEntropy}(t_{cls}^i, W_{cls} \cdot v_i^j) \]

**Total Loss:** Average per-anchor loss over anchors

**Challenge:** Dealing with class imbalance (usually way more negative anchors (class 0) than positive anchors)

**Solutions:** Subsampling negative anchors, downweighting the loss contribution of negatives, hard mining, etc...
Dealing with multiple detections of the same object

**Duplicate detection problem:** Typically many anchors will detect the same underlying object and give slightly different boxes, with slightly different scores.

**Solution:** remove detections if they overlap too much with another higher scoring detection.
Non Max Suppression (NMS)

Algorithm:
1. Sort detections in decreasing order with respect to score
2. Iterate through sorted detections:
   a. Reject a detection if it overlaps with a previous (unrejected) detection with IOU greater than some threshold
3. Return all unrejected detections

Some shortcomings of NMS to remember:
- Imposes a hard limitation on how close objects can be in order to be detected
- Similar classes do not suppress each other
A simplified convnet for detection

Think of each feature vector $v_{ij}$ as corresponding to a sliding window (anchor).

Category score $= \text{SoftMax}(W^{\text{cls}} \cdot v_{ij})$

Offset from anchor $= W^{\text{loc}} \cdot v_{ij}$
A simplified convnet for detection

Think of each feature vector $v_{ij}$ as corresponding to a sliding window (anchor).

Category score = $\text{SoftMax}(W_{\text{cls}} \cdot v_{ij})$

Offset from anchor = $W_{\text{loc}} \cdot v_{ij}$

Use the same $W_{\text{loc}}$ and $W_{\text{cls}}$ for all $i, j$ in anchor grid if anchors are:

- of the same shape, and
- contained and centered in receptive field
A simplified convnet for detection

Think of each feature vector $v_{ij}$ as corresponding to a sliding window (anchor).

Category score = $\text{SoftMax}(W^{cls} \cdot v_{ij})$

Offset from anchor = $W^{loc} \cdot v_{ij}$

Use **convolution** to do simultaneous prediction for all anchors:

Category score = $\text{SoftMax}(\text{Conv}(v; W^{cls}))$

Offset from anchor = $\text{Conv}(v; W^{loc})$
Think of each feature vector $v_{ij}$ as corresponding to a sliding window (anchor).

Category score = $\text{SoftMax}(W_{\text{cls}} \cdot v_{ij})$

Offset from anchor = $W_{\text{loc}} \cdot v_{ij}$

Use convolution to do simultaneous prediction for all anchors:

Category score = $\text{SoftMax}(\text{Conv}(v; W_{\text{cls}}))$

Offset from anchor = $\text{Conv}(v; W_{\text{loc}})$

But… if anchors need to be the same shape, how do we handle different scales/aspect ratios?
Solution: use multiple $W^{loc}$ and $W^{cls}$ (one for each aspect ratio/scale)

```
SoftMax($W^{cls,ar1} \cdot v_{ij}$)
$W^{loc,ar1} \cdot v_{ij}$

SoftMax($W^{cls,ar2} \cdot v_{ij}$)
$W^{loc,ar2} \cdot v_{ij}$

SoftMax($W^{cls,ar3} \cdot v_{ij}$)
$W^{loc,ar3} \cdot v_{ij}$
...
```
Fancier Solution: use multiple anchor grid resolutions
Detection “meta-architectures” are a recipe for converting classification architectures into detection architectures.

SSD
(Single Shot Detector --- encapsulates Multibox, YOLO, YOLO v2)
[Liu et al 2016]
Another popular meta-architecture

Faster R-CNN
(Faster Region-based Convolutional Networks)
[Ren et al 2015]
And yet another... but that’s about it!

R-FCN
(Region based Fully Convolutional Networks)
Dai et al, 2016
Today

- Sliding Window Detectors
- Detection with Convolutional Networks
- **How to Evaluate a Detector**
- Practical tips/tricks
- Variations on a theme (instance segmentation, keypoint detection, video detection, etc...)
How do we know how good our model is?

Accuracy: 75%

For image classification, life is easy :)

For image classification, life is easy :)
Evaluating Detectors is harder :(

Problem 1: Metrics must handle location errors

Should we consider this detection to be correct?
Evaluating Detectors is harder :(

Problem 2: Metrics must account for overprediction and underprediction
Intersection over Union (IOU)

\[
\text{IOU} = \frac{\text{Intersection}}{\text{Union}}
\]

Detection is “correct” if \( \text{IOU} > \alpha \)
Intersection over Union (IOU)

- IoU = 0.5
- IoU = 0.7
- IoU = 0.95

True/False Positives and Missed Objects

- Match detections and groundtruth instances based on IOU
- Count missed groundtruth objects
- Mark detections as TP or FP based on whether IOU>α
Summarizing Performance with Precision/Recall

**Precision**: Of the detections our model produced, how many were correct (i.e. True Positives)?

\[
\text{Precision} = \frac{\#TP}{\#TP + \#FP}
\]

**Recall**: Of the groundtruth instances in our data, what fraction of instances were correctly detected (i.e., not missed)?

\[
\text{Recall} = \frac{\#TP}{\#\text{Groundtruth Objects}}
\]

*Remember: Precision and Recall are in [0, 1] and higher is better.*
Trading off between Precision and Recall

Detectors usually produce thousands of boxes (sliding windows), each with some score/confidence;

Last step of detection pipeline: *use score threshold to select final detections*
Trading off between Precision and Recall

Detectors usually produce thousands of boxes (sliding windows), each with some score/confidence; Last step of detection pipeline: use score threshold to select final detections
Trading off between Precision and Recall

Detectors usually produce thousands of boxes (sliding windows), each with some score/confidence;

Last step of detection pipeline: *use score threshold to select final detections*
Trading off between Precision and Recall

Detectors usually produce thousands of boxes (sliding windows), each with some score/confidence;

Last step of detection pipeline: use score threshold to select final detections
Trading off between Precision and Recall

Detectors usually produce thousands of boxes (sliding windows), each with some score/confidence;

Last step of detection pipeline: use score threshold to select final detections

When would it be better to be on one side of this spectrum than the other?
Precision/Recall Curves and AP (Average Precision)

Precision / Recall curve

Precision

Recall

0.0  1.0
Precision/Recall Curves and AP (Average Precision)

AP = Average Precision = Area under PR curve

Remember:
- AP is always in [0, 1]
- Higher AP is better
- Always relative to an IOU criterion, e.g., AP@.5 IOU, AP@.75 IOU, etc...
You should know:

- How to mark detections as True or False positives based on IOU
- What Precision and Recall mean
- And have some vague idea about how P-R Curves and Average Precision are computed :)}
Today

- Sliding Window Detectors
- Detection with Convolutional Networks
- How to Evaluate a Detector
- **Practical tips/tricks**
- Variations on a theme (instance segmentation, keypoint detection, video detection, etc...)
Pick a point on the speed/accuracy tradeoff curve

Caution: this graph is a few years old now!
Rule of thumb: SSD (diamonds) faster than R-FCN (squares), which is faster than Faster R-CNN (circles)
SSD with MobileNet (and low resolution images) is fastest
There is a “pareto-optimal” curve. Those are our favorite detectors!
SSD w/MobileNet (Low Resolution)

SSD w/Inception V2 (Low Resolution)

Faster R-CNN w/Resnet101, 100 proposals

RFCN w/Resnet101, 300 proposals

Faster R-CNN w/Inception Resnet V2, 300 proposals
Initialize from a model pre-trained to classify some other dataset (the larger the better)

JFT 300M ➔ Transfer weights ➔ 18K labels ➔ Detections

<table>
<thead>
<tr>
<th>Method</th>
<th>mAP@0.5</th>
<th>mAP@[0.5,0.95]</th>
</tr>
</thead>
<tbody>
<tr>
<td>He et al. [16]</td>
<td>53.3</td>
<td>32.2</td>
</tr>
<tr>
<td>ImageNet</td>
<td>53.6</td>
<td>34.3</td>
</tr>
<tr>
<td>300M</td>
<td>56.9</td>
<td>36.7</td>
</tr>
<tr>
<td>ImageNet+300M</td>
<td>58.0</td>
<td>37.4</td>
</tr>
<tr>
<td>Inception ResNet [37]</td>
<td>56.3</td>
<td>35.5</td>
</tr>
</tbody>
</table>

See “Revisiting Unreasonable Effectiveness of Data in Deep Learning Era” [Sun et al 2017]
Use lower resolution images for speed

Lower resolution much faster; tends to miss smaller objects
Use a small number of proposals for speed (for proposal based architectures)

Lower # of proposals much faster; sacrifices a bit of recall
Replace stride 2 convolutions with stride 1

Slower, can boost performance on small objects

**Problem**: Doing this directly can reduce receptive field size...
Replace stride 2 convolutions with stride 1

Slower, can boost performance on small objects

Problem: Doing this directly can reduce receptive field size...
Replace stride 2 convolutions with stride 1

Slower, can boost performance on small objects

Problem: Doing this directly can reduce receptive field size...

Solution: Use atrous convolution (convolution with holes) to compensate at the second layer.
Today

- Sliding Window Detectors
- Detection with Convolutional Networks
- How to Evaluate a Detector
- Practical tips/tricks
- Variations on a theme (instance segmentation, keypoint detection, video detection, etc...)
Detection in Videos

**Video vs static image detection:**
- Frames often deteriorated
- Adjacent frames are often near-identical; wasteful to run full detection every frame
- Useful to exploit motion cues
Instance Segmentation: the next step up from bounding boxes

classify
(classify and regress bounding box per object)
(bounding box) detection
(classify per pixel)
semantic segmentation
(classify per pixel per object)
instance segmentation
Mask R-CNN
Example results from ADE20K

Slide courtesy of Alireza Fathi
Keypoint Detection

Slide courtesy of George Papandreou
Learning with less supervision

Labeling is hard work!

COCO dataset:
- 200K labeled images
- 1.5 million object instances
- 80 object categories
- ~40 person-years of labeling time!

Masks take ~x15 time to label compared to bounding boxes.

Can we learn to predict masks without explicit groundtruth mask annotations?

One idea: using “cut+paste” to get indirect feedback for mask predictions

**Supervised question:** “is this predicted mask correct?”

**Weakly supervised question:** “if I generate a new image by cut+pasting pixels inside the mask to a new part of the image, does it look plausible?”

Formalizing the Cut+Paste signal as a GAN (Generative Adversarial Network)

★ Both generator and discriminator are trained jointly.
Mask R-CNN trained using Cut+Paste GAN
Summary

- Detectors are important and mature tech
- Sliding Window still the way to go
- Convnets can put the sliding in sliding window
- Detectors are evaluated with PR curves
- Bounding boxes are only the first step to complex scene understanding
Tensorflow Object Detection API

Creating accurate machine learning models capable of localizing and identifying multiple objects in a single image remains a core challenge in computer vision. The TensorFlow Object Detection API is an open source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models. At Google we've certainly found this codebase to be useful for our computer vision needs, and we hope that you will as well.

Contributions to the codebase are welcome and we would love to hear back from you if you find this API useful. Finally if you use the Tensorflow Object Detection API for a research publication, please consider citing:

"Speed/accuracy trade-offs for modern convolutional object detectors."

Maintainers

- Jonathan Huang, github: jch1
- Vivek Rathod, github: tombstone
- Derek Chow, github: derekchow
Configuring a model using the API

```plaintext
model {
  faster_rcnn {
    num_classes: 3
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1024
      }
    }
  }
  feature_extractor {
    type: 'faster_rcnn_resnet101'
    first_stage_features_stride: 16
  }
  ...}
```

- {cars, people, stop signs}
- High resolution input images
- Faster R-CNN, Resnet 101
Configuring training using the API

train_config: {
    batch_size: 32
    fine_tune_checkpoint: "home/jonathanhuang/..."
    optimizer {
        rms_prop_optimizer: {
            learning_rate: {
                exponential_decay_learning_rate {
                    initial_learning_rate: 0.005
                    decay_steps: 200000
                    decay_factor: 0.95
                }
            }
        }
    }
...
# TF Object Detection API Model Zoo

## COCO-trained models

<table>
<thead>
<tr>
<th>Model name</th>
<th>Speed (ms)</th>
<th>COCO mAP[1]</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssd_mobilenet_v1_coco</td>
<td>30</td>
<td>21</td>
<td>Boxes</td>
</tr>
<tr>
<td>ssd_mobilenet_v2_coco</td>
<td>31</td>
<td>22</td>
<td>Boxes</td>
</tr>
<tr>
<td>ssdlite_mobilenet_v2_coco</td>
<td>27</td>
<td>22</td>
<td>Boxes</td>
</tr>
<tr>
<td>ssd_inception_v2_coco</td>
<td>42</td>
<td>24</td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_inception_v2_coco</td>
<td>58</td>
<td>28</td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_resnet50_coco</td>
<td>89</td>
<td>30</td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_resnet50_lowproposals_coco</td>
<td>64</td>
<td></td>
<td>Boxes</td>
</tr>
<tr>
<td>rfcn_resnet101_coco</td>
<td>92</td>
<td>30</td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_resnet101_coco</td>
<td>106</td>
<td>32</td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_resnet101_lowproposals_coco</td>
<td>82</td>
<td></td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_inception_resnet_v2_atrous_coco</td>
<td>620</td>
<td>37</td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco</td>
<td>241</td>
<td></td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_resnet50_atrous_lowproposals_coco</td>
<td>1833</td>
<td>43</td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_nas_lowproposals_coco</td>
<td>540</td>
<td></td>
<td>Boxes</td>
</tr>
<tr>
<td>mask_rcnn_inception_resnet_v2_atrous_coco</td>
<td>771</td>
<td>36</td>
<td>Masks</td>
</tr>
<tr>
<td>mask_rcnn_inception_v2_coco</td>
<td>79</td>
<td>25</td>
<td>Masks</td>
</tr>
<tr>
<td>mask_rcnn_resnet101_atrous_coco</td>
<td>470</td>
<td>33</td>
<td>Masks</td>
</tr>
<tr>
<td>mask_rcnn_resnet50_atrous_coco</td>
<td>343</td>
<td>29</td>
<td>Masks</td>
</tr>
</tbody>
</table>

## Kitti-trained models

<table>
<thead>
<tr>
<th>Model name</th>
<th>Speed (ms)</th>
<th>Pascal mAP@0.5</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>faster_rcnn_resnet101_kitti</td>
<td>79</td>
<td>87</td>
<td>Boxes</td>
</tr>
</tbody>
</table>

## Open Images-trained models

<table>
<thead>
<tr>
<th>Model name</th>
<th>Speed (ms)</th>
<th>Open Images mAP@0.5[2]</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>faster_rcnn_inception_resnet_v2_atrous_oid</td>
<td>727</td>
<td>37</td>
<td>Boxes</td>
</tr>
<tr>
<td>faster_rcnn_inception_resnet_v2_atrous_lowproposals_oid</td>
<td>347</td>
<td></td>
<td>Boxes</td>
</tr>
</tbody>
</table>

## AVA v2.1 trained models

<table>
<thead>
<tr>
<th>Model name</th>
<th>Speed (ms)</th>
<th>Pascal mAP@0.5</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>faster_rcnn_resnet101_ava_v2.1</td>
<td>93</td>
<td>11</td>
<td>Boxes</td>
</tr>
</tbody>
</table>
Community Creations!

How to train your own Object Detector with TensorFlow's Object Detector API

This is a follow-up post on “Building a Real-Time Object Recognition App with TensorFlow and OpenCV” where I focus on training my own classes. Specifically, I trained my own Raccoon detector on a dataset that I collected and labeled by myself. The full dataset is available on my GitHub repo.

By the way, here is the Raccoon detector in action:

![Raccoon detector in action](image)

If you want to know the details, you should continue reading!

---

Is Google TensorFlow Object Detection API the easiest way to implement image recognition?

Yes, and here's why...

There are many different ways to do image recognition. Google recently released a new TensorFlow Object Detection API, which computer vision everywhere a locus. Any offering from Google is one to be taken lightly, and not decided to try my hands on this new API and see it in action from your laptop (for the result below):

![Image result from Google](image)

You can find the full code on my GitHub repo.
Thanks!