Few-shot Object Detection

a.k.a. «How not to invent a bicycle»

N. Popov

Faculty of Computer Science
Higher School of Economics

Mar. 23, 2018

Table of Contents

Object Detection

When there are objects you need to find

- Basic problem statement: "find all instances of an object in a scene"
- Next level: "find all instances of multiple types of objects in a scene"
- Also, could you please determine the classes while you're at it?
- Sounds not that complicated, right?

Russian Traffic Sign Dataset

Let us consider an example task of **traffic sign detection (and classification)** on RTSD:

Existing Solutions

Should we invent our very own bicycle?

- Naive solution:
  - split image into tiny pieces;
  - run each through pre-trained image classifier;
  - ???
  - PROFIT
- R-CNN, Fast R-CNN, Faster R-CNN:
  - region-based, but regions are computed sensibly.
  - Pros: works
  - Cons: slow, can't be trained end-to end.
- YOLO, YOLOv2, YOLOv3:
  - single phase, "You Only Look Once"
  - Pros: Fast as hell
  - Cons: Can't detect arbitrary amount of objects.
- ...and more.

Why make our own?

- All of these models have literally millons of weights; we need a **lot** of data to train them effectively.
- However, there are a lot of different kinds of traffic signs; some are very rare, and it is hard to get training samples for some of them.
- For example, in RTSD the most represented class has >40k examples; the least represented has 1.
- It is impossible to adequately train a regular detection model on this data in a straightforward way.
- What do we do???

One-shot learning

- One-shot learning to the rescue!
- The whole subfield of one-shot (and few-shot) learning has one main idea: "How do we train complex models on very little data?".
- Some of the common approaches include
  - Meta-learning: learning to learn on small datasets by using a *lot* of small datasets.
  - Metric learning: learning to compare unknown objects to known ones.
  - Tranfer learning: learning to use large datasets to improve quality on small ones.
  - And a whole zoo of other, less popular approaches, e.g. bayesian methods.

Has anybody done one-shot object detection before?

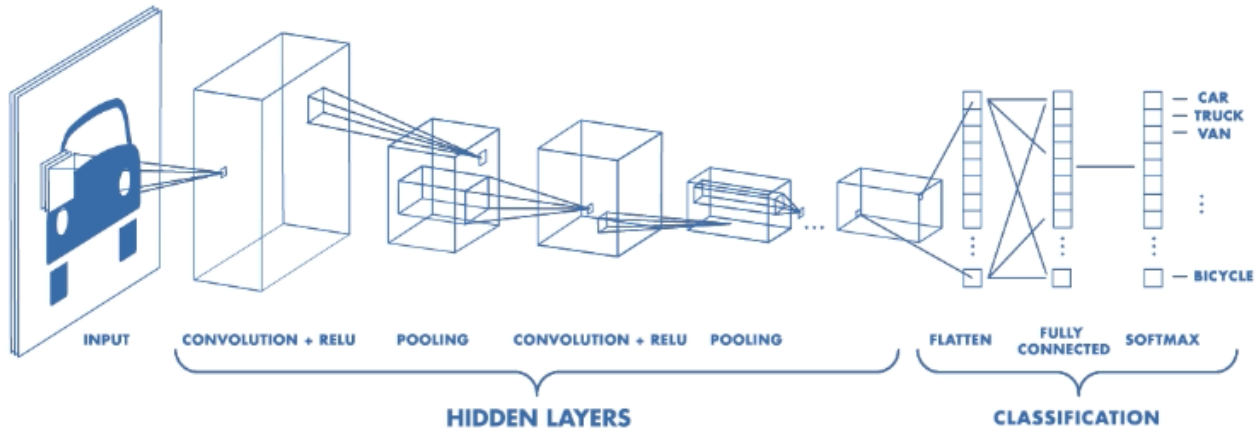
- No!
- Well, yes, but not really.
- X. Dong [et al.] in *Few-shot Object Detection* do use few-shot learning, but they also use a big unlabeled dataset.
- Wang Y.-X., Hebert M. in *Model recommendation: Generating object detectors from few samples* use some weird tricks which kind of are few-shot learning, but not really.
- So I've decided to make my own.

What model do we use?

- Scene images are very big and have a lot of unnecesary data: can we make the problem simpler?
- Let's start by learning to classify traffic signs: cut them out of the background and train a regular CNN to classify them.
- Then we will use this classification to score each pixel of the scene and find peaks of confidence.

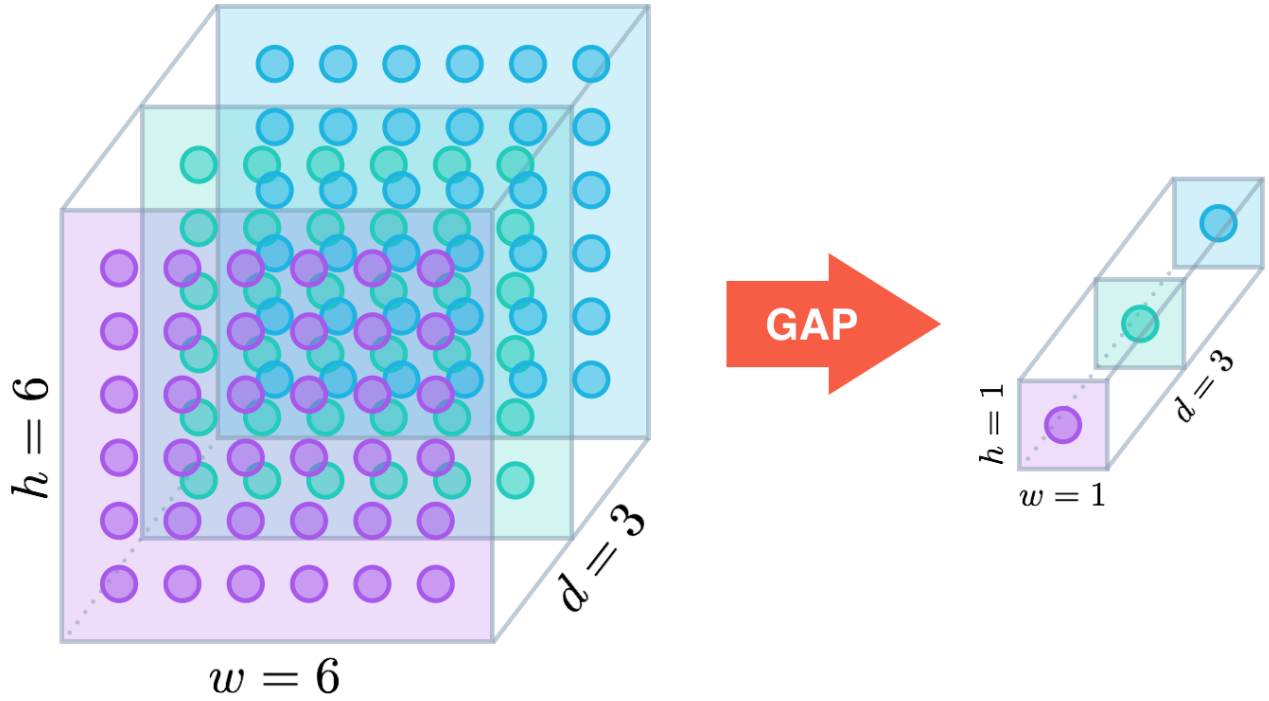
How do we use it?

There is a caveat: basic CNN can't receive arbitrary-sized images.



How do we use it?

Let's drop the last part and instead average by slice (also known as Global Average Pooling)



How do we train it?

- We will use **Reptile**: a general-purpose meta-learning algorithm;
- This is a replacement for regular SGD: instead of using SGD to learn one good model we:
  - Sample a small dataset: 5 different images, for example;
  - Train a small model to discern these 5 images;
  - Use the trained model as an initialization for the next model.
- In the end this should converge to an initialization that is easy to train to discern *any* 5 kinds of signs with only a single example of each.

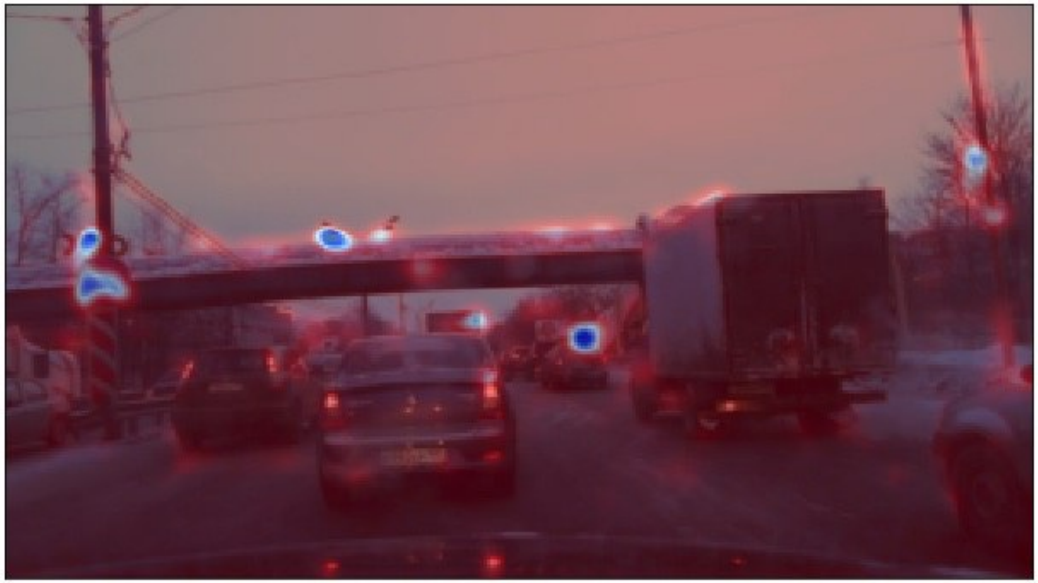
Reptile

Algorithm 1 Reptile

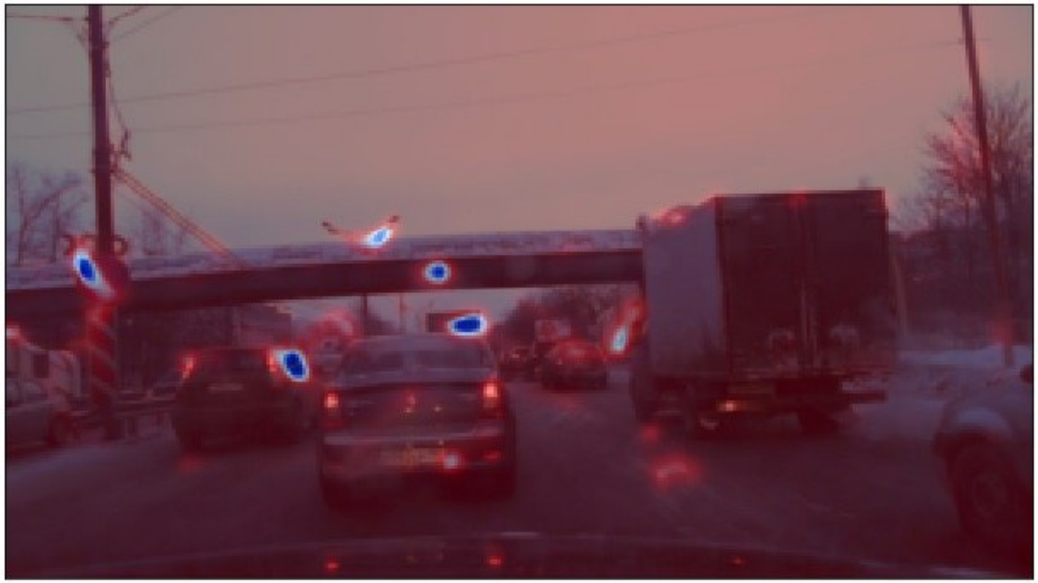
Initialize ϕ, initial parameters vector
for iteration $= 1, 2, \cdots$ **do**
    sample task τ
    $\tilde{\phi} \leftarrow \phi$
    for $i = 1, \cdots, k$ **do**
        Calculate $L_\tau(\tilde{\phi})$ — current loss
        $\tilde{\phi} \leftarrow \tilde{\phi} - \alpha \nabla L_\tau(\tilde{\phi})$
    $\phi \leftarrow \phi + \epsilon(\tilde{\phi} - \phi)$

What does in look like?

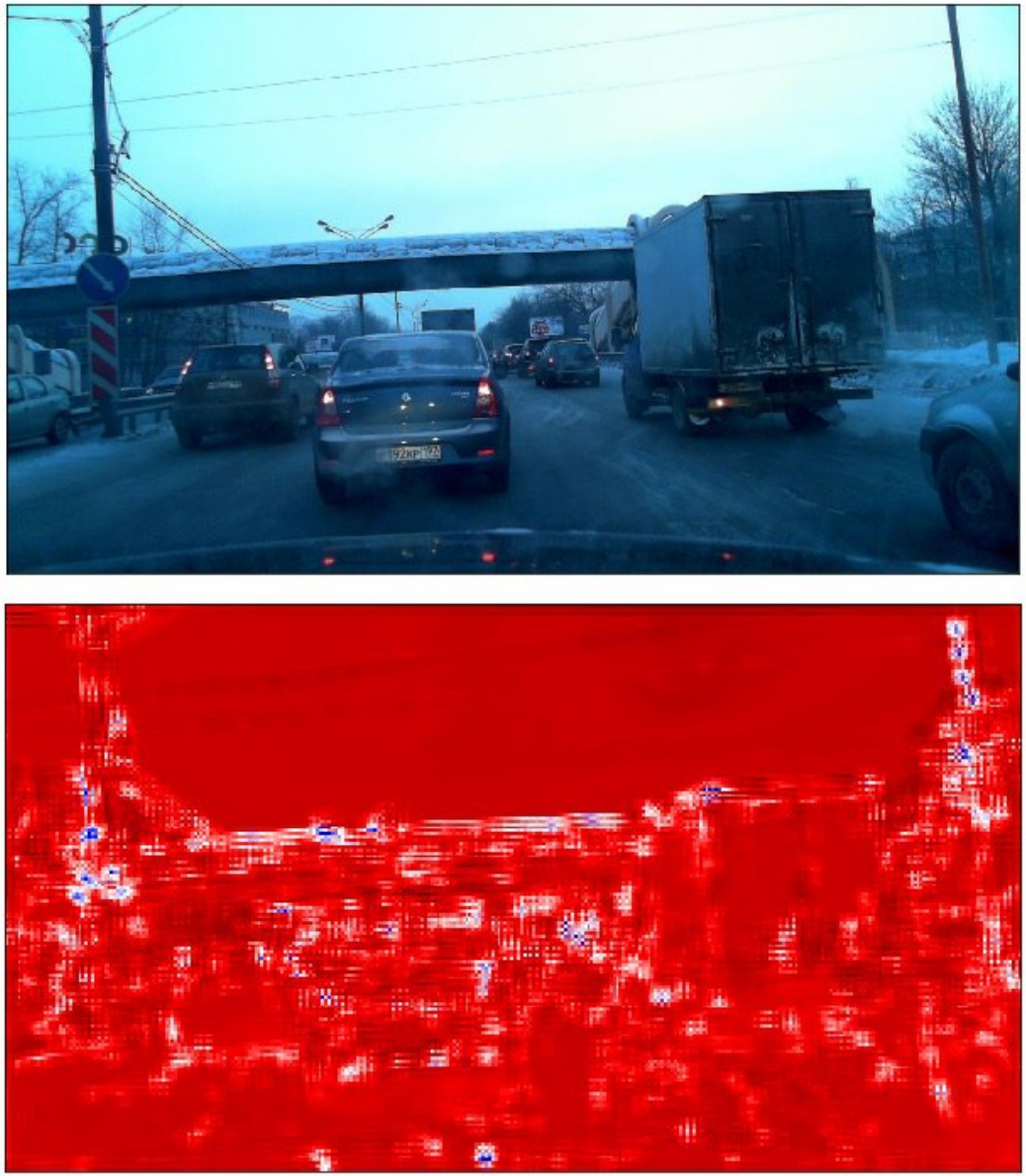
What does it really look like?



What does it really look like?



What does it really look like?



Why?

There is a fundamental flaw of this approach:

- The more layers we stack in the model, the better the classifiaction quality is;
- The more layers we stack in the model, the more its *perceptive field* grows;
- The bigger the perceptive field, the less localized the confidence is;
- The less localized the confidence is, the noisier the image becomes.

Possible improvements

- Train shallower but more powerful models; does not seem to be easily doable, but maybe?
- Use regularization to directly tie the confidence on a region to its central pixel;
- Reimplement again, there always is a possibility of a simple coding mistake.