8.3. Networks for object detection

François Fleuret
https://fleuret.org/dlc/

Nov 29, 2020
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.

Parsing at fixed scale  
Final list of detections
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.

Parsing at fixed scale

Final list of detections
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.

Parsing at fixed scale

Final list of detections
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.

Parsing at fixed scale

Final list of detections
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.

Parsing at fixed scale

Final list of detections
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.

Parsing at fixed scale

Final list of detections
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.

**Parsing at fixed scale**

**Final list of detections**
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.

Parsing at fixed scale

Final list of detections
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.

Parsing at fixed scale

Final list of detections
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.

Parsing at fixed scale

Final list of detections
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.

Parsing at fixed scale

Final list of detections
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.

Parsing at fixed scale

Final list of detections
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.

Parsing at fixed scale  
Final list of detections
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.

Parsing at fixed scale

Final list of detections
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.

Parsing at fixed scale

Final list of detections
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.

Final list of detections
The simplest strategy for object detection is to classify local regions, at multiple scales and locations.

This “sliding window” approach evaluates a classifier multiple times, and its computational cost increases with the prediction accuracy.
This was mitigated in overfeat (Sermanet et al., 2013) by adding a regression part to predict the object’s bounding box.
This was mitigated in overfeat (Sermanet et al., 2013) by adding a regression part to predict the object’s bounding box.
In the single-object case, the convolutional layers are frozen, and the localization layers are trained with a $\ell_2$ loss.

Figure 7: Examples of bounding boxes produced by the regression network, before being combined into final predictions. The examples shown here are at a single scale. Predictions may be more optimal at other scales depending on the objects. Here, most of the bounding boxes which are initially organized as a grid, converge to a single location and scale. This indicates that the network is very confident in the location of the object, as opposed to being spread out randomly. The top left image shows that it can also correctly identify multiple locations if several objects are present. The various aspect ratios of the predicted bounding boxes shows that the network is able to cope with various object poses.

Combining the multiple boxes is done with an ad hoc greedy algorithm.

(Sermanet et al., 2013)
This architecture can be applied directly to detection by adding a class “Background” to the object classes.

Negative samples are taken in each scene either at random or by selecting the ones with the worst miss-classification.
This architecture can be applied directly to detection by adding a class “Background” to the object classes.

Negative samples are taken in each scene either at random or by selecting the ones with the worst miss-classification.

Surprisingly, using class-specific localization layers did not provide better results than having a single one shared across classes (Sermanet et al., 2013).
Other approaches evolved from AlexNet, relying on region proposals:

- Generate thousands of proposal bounding boxes with a non-CNN “objectness” approach such as Selective search (Uijlings et al., 2013),
- feed to an AlexNet-like network sub-images cropped and warped from the input image (“R-CNN”, Girshick et al., 2013), or from the convolutional feature maps to share computation (“Fast R-CNN”, Girshick, 2015).
Other approaches evolved from AlexNet, relying on **region proposals**:

- Generate thousands of proposal bounding boxes with a non-CNN “objectness” approach such as Selective search (Uijlings et al., 2013),
- feed to an AlexNet-like network sub-images cropped and warped from the input image (“R-CNN”, Girshick et al., 2013), or from the convolutional feature maps to share computation (“Fast R-CNN”, Girshick, 2015).

These methods suffer from the cost of the region proposal computation, which is non-convolutional and not implementable on GPU.

They were improved by Ren et al. (2015) in “Faster R-CNN” by replacing the region proposal algorithm with a convolutional processing similar to Overfeat.
The most famous algorithm from this lineage is “You Only Look Once” (YOLO, Redmon et al. 2015).
The most famous algorithm from this lineage is “You Only Look Once” (YOLO, Redmon et al. 2015).

It comes back to a classical architecture with a series of convolutional layers followed by a few fully connected layers. It is sometime described as “one shot” since a single information pathway suffices.
The most famous algorithm from this lineage is “You Only Look Once” (YOLO, Redmon et al. 2015).

It comes back to a classical architecture with a series of convolutional layers followed by a few fully connected layers. It is sometime described as “one shot” since a single information pathway suffices.

YOLO’s network is not a pre-existing one. It uses leaky ReLU, and its convolutional layers make use of the $1 \times 1$ bottleneck filters (Lin et al., 2013) to control the memory footprint and computational cost.
making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during the training process. YOLO's design enables end-to-end training and real-time speeds while maintaining high average precision. The YOLO design is especially about the full image and all the objects in the image.

Our system divides the input image into an S × S grid. Each grid cell predicts bounding boxes and confidence. The S grid on input
Bounding boxes + confidence
Class probability map
Final detections

Each bounding box consists of 5 predictions: bounding boxes and confidence. The \((x, y, w, h)\) coordinates represent the center and size of the predicted box, while the confidence score represents how confident the model is that the box contains an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes. For each grid cell, we predict the output probabilities and coordinates. These predictions are encoded as an \(S \times S\) tensor. For evaluating YOLO on PASCAL VOC and PASCAL VOC, we use \(S = 7\) for all experiments.

The \(S \times S\) grid helps to make predictions. Each grid cell is responsible for predicting the presence of objects in its corresponding area of the image. The grid is used to divide the input image into smaller segments, allowing the model to focus on different parts of the image.

We implement this model as a convolutional neural network (CNN) with 24 convolutional layers followed by 2 fully connected layers. The initial convolutional layers of the network extract features from the image, while the fully connected layers perform classification and regression.

We also train a fast version of YOLO designed to push the boundaries of fast object detection. Fast YOLO uses a neural network with fewer convolutional layers (9 instead of the 24 used by YOLO) and fewer filters in those layers. Other than the size of the network, all training and testing parameters are the same between YOLO and Fast YOLO.

At test time, we multiply the conditional class probability \(p_i^\text{class}\) and the individual box confidence prediction \(p_i^\text{conf}\) to get the final confidence score.

\[
\text{Confidence score} = p_i^\text{class} \times p_i^\text{conf}
\]

Where \(i\) is the index of the predicted object.

We also introduce a high-level constraint that only the object exists in that cell, the confidence scores should be zero. Otherwise, we want the confidence score to equal the object exists in that cell.

YOLO outperforms top detection methods like DPM, Fast R-CNN, and Faster R-CNN in accuracy. While it can quickly identify objects in images, it struggles to precisely localize some objects, especially when applied to new domains or unexpected inputs. YOLO learns generalizable representations of objects, making it less likely to break down when applied to new domains or inputs.

YOLO is less generalizable than Faster R-CNN, which can make it challenging to apply to new domains or unexpected inputs. However, YOLO is highly specialized in terms of object detection, and it is less sensitive to changes in the environment.

YOLO is a highly specialized object detection model that is less generalizable than Faster R-CNN. However, it is highly specialized in terms of object detection, and it is less sensitive to changes in the environment.

(Yedunathan et al., 2015)
The output corresponds to splitting the image into a regular $S \times S$ grid, with $S = 7$.

(Redmon et al., 2015)
The output corresponds to splitting the image into a regular $S \times S$ grid, with $S = 7$, and for each cell, to predict a 30d vector.
The output corresponds to splitting the image into a regular $S \times S$ grid, with $S = 7$, and for each cell, to predict a 30d vector:

- $B = 2$ bounding boxes coordinates and confidence,
- $C = 20$ class probabilities, corresponding to the classes of Pascal VOC.

(Redmon et al., 2015)

\[
\begin{align*}
\hat{x}_{i,1} & \quad \hat{y}_{i,1} & \quad \hat{w}_{i,1} & \quad \hat{h}_{i,1} & \quad \hat{c}_{i,1} & \quad \ldots & \quad \hat{x}_{i,B} & \quad \hat{y}_{i,B} & \quad \hat{w}_{i,B} & \quad \hat{h}_{i,B} & \quad \hat{c}_{i,B} & \quad \ldots & \quad \hat{p}_{i,1} & \quad \ldots & \quad \hat{p}_{i,C} \\
\end{align*}
\]

$5B$ values

$C$ values
So the network predicts class scores and bounding-box regressions, and although the output comes from fully connected layers, it has a 2D structure.
So the network predicts class scores and bounding-box regressions, and although the output comes from fully connected layers, it has a 2D structure.

It allows in particular YOLO to leverage the absolute location in the image to improve performance (e.g. vehicles tend to be at the bottom, umbrella at the top), which may or may not be desirable.
During training, YOLO makes the assumption that any of the $S^2$ cells contains at most [the center of] a single object. We define for every image, cell index $i = 1, \ldots, S^2$, predicted box index $j = 1, \ldots, B$ and class index $c = 1, \ldots, C$

- $1_{i}^{\text{obj}}$ is 1 if there is an object in cell $i$ and 0 otherwise,
- $1_{i,j}^{\text{obj}}$ is 1 if there is an object in cell $i$ and predicted box $j$ is the most fitting one, 0 otherwise.
- $p_{i,c}$ is 1 if there is an object of class $c$ in cell $i$, and 0 otherwise,
- $x_{i}, y_{i}, w_{i}, h_{i}$ the annotated object bounding box (defined only if $1_{i}^{\text{obj}} = 1$, and relative in location and scale to the cell),
- $c_{i,j}$ IOU between the predicted box and the ground truth target.
The training procedure first computes on each image the value of the \(1_{i,j}^{\text{obj}}\)’s and \(c_{i,j}\), and then does one step to minimize

\[
\lambda_{\text{coord}} \sum_{i=1}^{S^2} \sum_{j=1}^{B} 1_{i,j}^{\text{obj}} \left( (x_i - \hat{x}_{i,j})^2 + (y_i - \hat{y}_{i,j})^2 + (\sqrt{w_i} - \sqrt{\hat{w}_{i,j}})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_{i,j}})^2 \right)
\]

\[
+ \lambda_{\text{obj}} \sum_{i=1}^{S^2} \sum_{j=1}^{B} 1_{i,j}^{\text{obj}} (c_{i,j} - \hat{c}_{i,j})^2 + \lambda_{\text{noobj}} \sum_{i=1}^{S^2} \sum_{j=1}^{B} (1 - 1_{i,j}^{\text{obj}}) \hat{c}_{i,j}^2
\]

\[
+ \lambda_{\text{classes}} \sum_{i=1}^{S^2} 1_{i}^{\text{obj}} \sum_{c=1}^{C} (p_{i,c} - \hat{p}_{i,c})^2
\]

where \(\hat{p}_{i,c}, \hat{x}_{i,j}, \hat{y}_{i,j}, \hat{w}_{i,j}, \hat{h}_{i,j}, \hat{c}_{i,j}\) are the network’s outputs.

(slightly re-written from Redmon et al. 2015)
Training YOLO relies on many engineering choices that illustrate well how involved is deep-learning “in practice”:

- Pre-train the 20 first convolutional layers on ImageNet classification,
- use 448 × 448 input for detection, instead of 224 × 224,
- use Leaky ReLU for all layers,
- dropout after the first fully connected layer,
Training YOLO relies on many engineering choices that illustrate well how involved is deep-learning “in practice”:

- Pre-train the 20 first convolutional layers on ImageNet classification,
- use $448 \times 448$ input for detection, instead of $224 \times 224$,
- use Leaky ReLU for all layers,
- dropout after the first fully connected layer,
- normalize bounding boxes parameters in $[0, 1]$,
- use a quadratic loss not only for the bounding box coordinates, but also for the confidence and the class scores,
- reduce the weight of large bounding boxes by using the square roots of the size in the loss,
- reduce the importance of empty cells by weighting less the confidence-related loss on them,
Training YOLO relies on many engineering choices that illustrate well how involved is deep-learning “in practice”:

- Pre-train the 20 first convolutional layers on ImageNet classification,
- use $448 \times 448$ input for detection, instead of $224 \times 224$,
- use Leaky ReLU for all layers,
- dropout after the first fully connected layer,
- normalize bounding boxes parameters in $[0, 1]$,
- use a quadratic loss not only for the bounding box coordinates, but also for the confidence and the class scores,
- reduce the weight of large bounding boxes by using the square roots of the size in the loss,
- reduce the importance of empty cells by weighting less the confidence-related loss on them,
- use momentum 0.9, decay $5e^{-4}$,
- data augmentation with scaling, translation, and HSV transformation.

A critical technical point is the design of the loss function that articulates both a classification and a regression objectives.
The Single Shot Multi-box Detector (SSD, Liu et al., 2015) improves upon YOLO with a fully-convolutional architectures and multi-scale maps.
To summarize roughly how “one shot” deep detection can be achieved:

- networks trained on image classification capture localization information,
- regression layers can be attached to classification-trained networks,
- object localization does not have to be class-specific,
- multiple detection are estimated at each location to account for different aspect ratios and scales.
Object detection networks

- **AlexNet** (Krizhevsky et al., 2012)
  - Box regression
  - Region proposal + crop in image
- **Overfeat** (Sermanet et al., 2013)
- **R-CNN** (Girshick et al., 2013)
  - Crop in feature maps
- **Fast R-CNN** (Girshick, 2015)
  - Convolutional region proposal
- **Faster R-CNN** (Ren et al., 2015)
  - No crop
- **YOLO** (Redmon et al., 2015)
  - Fully convolutional + multi-scale maps
- **SSD** (Liu et al., 2015)
  - Multi-scale convolutions + multi boxes
The end
References


