EE-559 – Deep learning

8.3. Networks for object detection

François Fleuret

https://fleuret.org/ee559/

Apr 8, 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.

**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.
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.

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.

Final list of detections

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.

(Sermanet et al., 2013)

Combining the multiple boxes is done with an *ad hoc* greedy algorithm.
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 training and test time so it implicitly encodes context. This means our network reasons globally about the full image and all the objects in the image. YOLO makes less than half the number of background errors compared to Fast R-CNN.

Third, YOLO learns generalizable representations of objects. When trained on natural images and tested on艺术 images it struggles to precisely localize some objects, especially in unexpected inputs. YOLO is highly generalizable; it is less likely to break down when applied to new domains or unexpected inputs. YOLO still lags behind state-of-the-art detection systems in accuracy. While it can quickly identify objects in images, there are other systems that are more accurate. However, for many purposes, YOLO is sufficient.

The YOLO design enables end-to-end training and real-time inference. This means that our network reasons globally about the full image and all the objects in the image. YOLO makes less than half the number of background errors compared to Fast R-CNN. The system is also highly generalizable; it is less likely to break down when applied to new domains or unexpected inputs.

Our system divides the input image into an $S \times S$ grid and for each grid cell, regardless of the class of objects appearing in the box and how well the predicted box fits the bounds of the grid cell. The width $w$ and height $h$ of each grid cell is used to calculate the intersection over union (IOU) between the predicted box and any ground truth box.

The YOLO system works and evaluates it on the PASCAL VOC detection dataset. For evaluating YOLO on the PASCAL VOC dataset, we use a class probability map. Each grid cell predicts 5 bounding boxes, confidence for those boxes, and one set of class probabilities per grid cell, regardless of the class of objects appearing in the box and how well the predicted box fits the bounds of the grid cell. These predictions are encoded as an $S \times S \times 5 + 1$ tensor.

For evaluating YOLO on the PASCAL VOC dataset, we use a class probability map. Each grid cell predicts 5 bounding boxes, confidence for those boxes, and one set of class probabilities per grid cell, regardless of the class of objects appearing in the box and how well the predicted box fits the bounds of the grid cell. These predictions are encoded as an $S \times S \times 5 + 1$ tensor.

Our final prediction is a class probability map. 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 24) 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.

The system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell, regardless of the class of objects appearing in the box and how well the predicted box fits the bounds of the grid cell. The system predicts class probabilities and bounding boxes. The output probabilities and coordinates are multiplied by the conditional class probabilities $P_{\text{Class}}(\text{Object} | \text{Image})$.

At test time we multiply the conditional class probabilities $P_{\text{Class}}(\text{Object} | \text{Image})$ by a set of 24 class-specific confidence scores for each object. These scores encode both the probability of that class and the confidence the model is that the box contains an object and predict the output probabilities and coordinates. The confidence scores reflect how confident the model is that the box contains an object and predict the output probabilities and coordinates.
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)
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 $c = 1, \ldots, C$

- $1^\text{obj}_i$ is 1 if there is an object in cell $i$ and 0 otherwise,
- $1^\text{obj}_{i,j}$ 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 $1^\text{obj}_i = 1$, 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 $1_{i,j}^{obj}$'s and $c_{i,j}$, and then does one step to minimize

$$
\lambda_{\text{coord}} \sum_{i=1}^{S^2} \sum_{j=1}^{B} 1_{i,j}^{obj} \left( (x_i - \hat{x}_{i,j})^2 + (y_i - \hat{y}_{i,j})^2 + (\sqrt{w_i} - \sqrt{\hat{w}_{i,j}})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_{i,j}})^2 \right) \\
+ \lambda_{\text{obj}} \sum_{i=1}^{S^2} \sum_{j=1}^{B} 1_{i,j}^{obj} (c_{i,j} - \hat{c}_{i,j})^2 + \lambda_{\text{noobj}} \sum_{i=1}^{S^2} \sum_{j=1}^{B} \left( 1 - 1_{i,j}^{obj} \right) \hat{c}_{i,j}^2 \\
+ \lambda_{\text{classes}} \sum_{i=1}^{S^2} 1_i^{obj} \sum_{c=1}^{C} (p_{i,c} - \hat{p}_{i,c})^2.
$$

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 $448 \times 448$ input for detection, instead of $224 \times 224$,
- 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


