ỨNG DỤNG FASTER R-CNN ĐỂ XÁC ĐỊNH ĐỐI TƯỢNG NHANH CHÓNG TRONG HÌNH ẢNH

Nguyễn Minh Trí, CH1602013 CNTT-Đợt I, Công Nghệ Thông Tin, ĐH Công Nghệ Thông Tin, TP. Hồ Chí Minh

Tóm Tắt

Ứng dụng Faster R-CNN là một trong những phương thức được sữ dụng để nhận diện đối tượng trong hình ảnh một cách nhanh chóng và hiệu quả. Trong kỹ nguyên hiện nay, việc làm sao cho máy tính có thể nhận dạng được đối tượng là một nhu cầu dường như không thể thiếu. Phân loại chính xác đối tượng sẽ giúp cho máy tính đáp ứng đúng các yêu cầu một cách nhanh chóng và chính xác nhất có thể. Trong nội dung nghiên cứu này, em sẽ trình bày về kết quả nghiên cứu và ứng dụng Faster R-CNN với mã nguồn ứng dụng thực tế tại github ở địa chỉ: https://github.com/trimsc/vra-k2/tree/master/Final/

Keyword: Faster R-CNN, Real-Time Object Detection, Neural Network.

1. Giới thiệu

Ngày nay càng có nhiều ứng dụng thực tế trong việc vận dụng máy học vào công nghệ thông tin, trong đó thuật toán Faster R-CNN là mô hình thuật toán vận dụng máy học trong việc nhận diện hình ảnh và phát hiện các đối tượng có trong ảnh theo tập dữ liệu huấn luyện. Trên thực tế Faster RCNN là một thuật toán để tìm kiếm vị trí của các vật thể trong một bức ảnh. Thuật toán này sẽ có đầu ra là những hình hộp bao quanh, cùng với việc xác định vật thể bên trong hộp đó là gì. Phiên bản đầu tiên của Faster RCNN là RCNN.

R-CNN

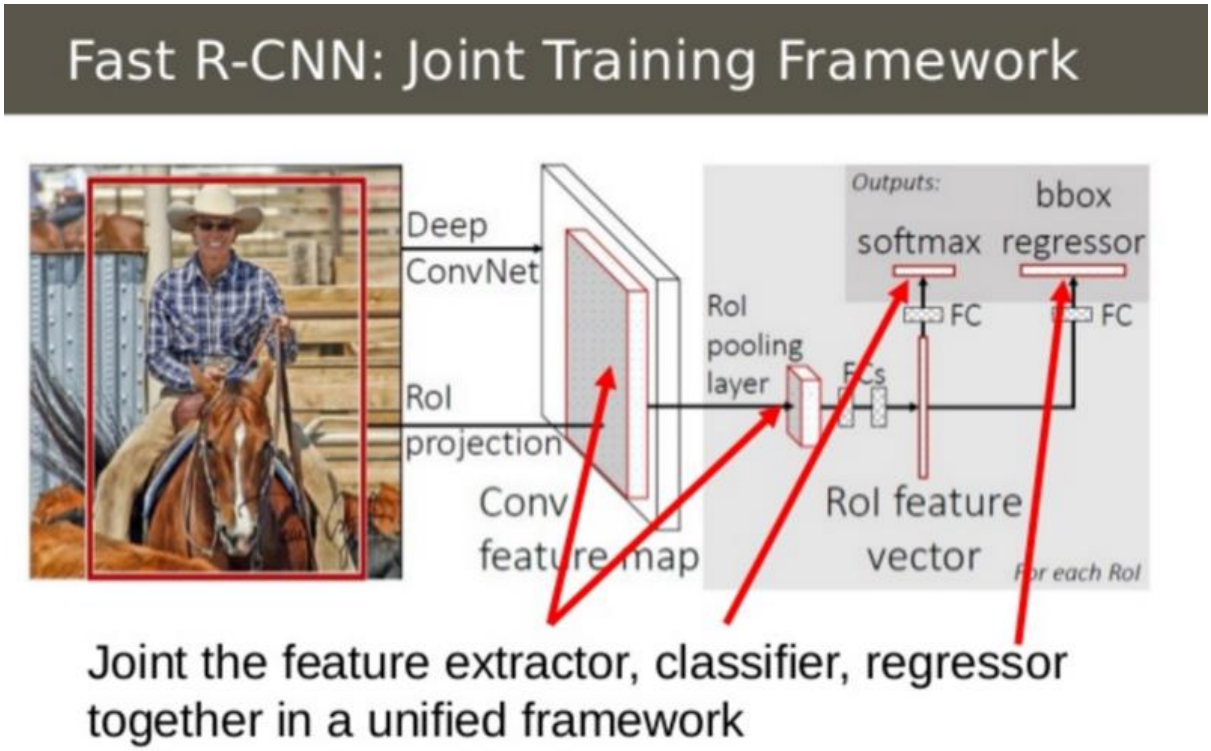
Với R-CNN ban đầu sử dụng một thuật toán gọi là selective search để đưa ra các bounding boxes, hay còn gọi là region proposals, các hình hộp này sẽ là các vùng có thể có vật thể ở bên trong. Nó sử dụng các mạng đã được huấn luyện sẵn như Alex-net, VGG-16 để tính toán feed-forward và các regions thu được convolutional features của mỗi region, sau đó huấn luyện SVM để xác định được vật thể nào được chứa trong region proposal đó. Trong thuật toán này chủ yếu sử dụng phương pháp Linear Regression để hiệu chỉnh các giá trị như vị trí các đỉnh của một region proposer.

Faster R-CNN

Nâng cấp hơn so với R-CNN, Faster R-CNN sử dụng các mạng huấn luyện sẵn để feed-forward để xác định các region proposals, quá trình huấn luyện này sẽ tốn nhiều thời gian bởi với mỗi ảnh thuật toán selective search sẽ cho ra hàng nghìn region proposals.

Khi chạy thuật toán chỉ feed-forward một lần đối với ảnh gốc, thu được các convolutional features của ảnh đó. Ví dụ với một hình ảnh có kích thước 600∗600∗3, ta sẽ thu được số lượng các convolutional features với kích thước 37∗37∗512. Kích thước của features bị giảm nhỏ khoảng 16 lần còn lại là 60037.

Dựa vào kích thước cùng vị trí của các region proposals đối với ảnh gốc, ta sẽ tính toán được vị trí của region proposal trong convolutional features. Và sau đó sử dụng các giá trị convolutional faetures của region proposal, ta dự đoán được vị trí các đỉnh của bounding box cũng như vật thể nằm trong bounding box là gì.



Đối với Faster R-CNN, do chia sẻ tính toán giữa các region trong ảnh, tốc độ thực thực thi của thuật toán sẽ được giảm từ 120s mỗi ảnh xuống chỉ còn 2s.

Phần tính toán gây ra nghẽn chính là phần đưa ra các region proposal đầu vào, chỉ có thể thực thi tuần tự trên CPU. Faster RCNN giải quyết vấn đề này bằng cách sử dụng DNN để tính toán các region proposals này.

RPN

RPN giải quyết các vấn đề trên bằng cách huấn luyện mạng neural network để đảm nhận thay vai trò của các thuật toán như selective search vốn rất chậm chạp.

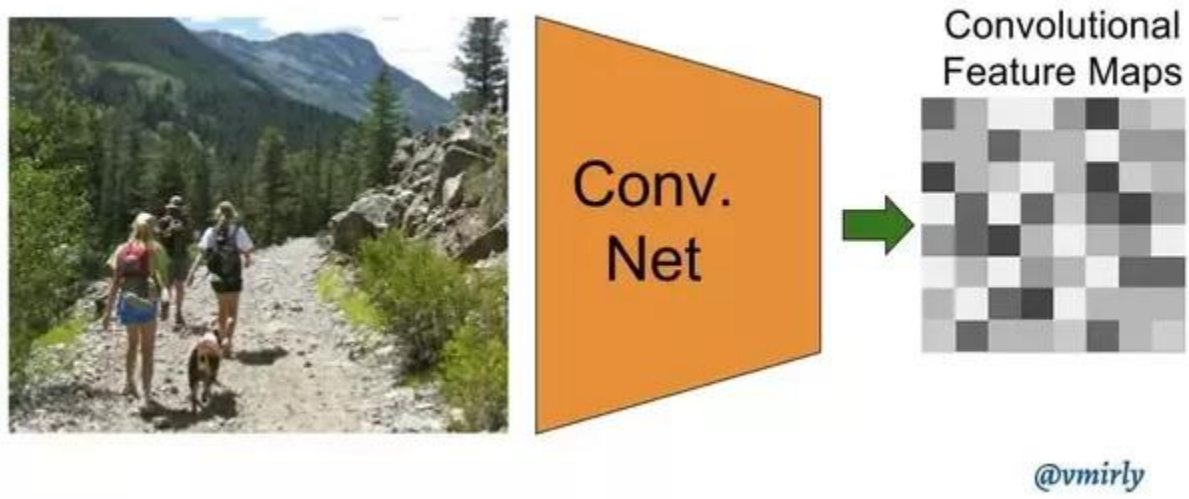
Một Region Proposal Network nhận đầu vào là ảnh với kích thước bất kì và cho đầu ra là region proposal (tập vị trí của các hình chữ nhật có thể chứa vật thể), cùng với xác suất chứa vật thể của hình chữ nhật tương ứng.

2. Cấu trúc

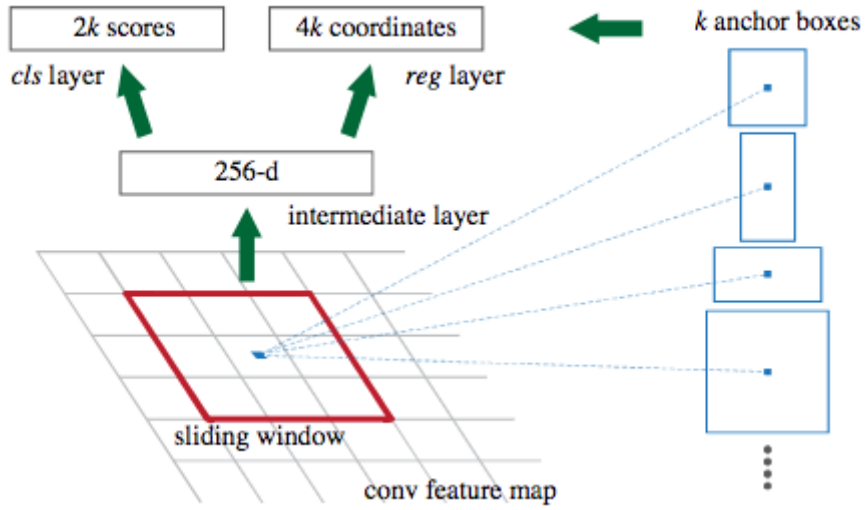
Cách hoạt động RPN có 2 bước chính:

2.1. Feed-forward ảnh qua DNN thu được convolutional features.

Với các mạng Convolution Network có sẵn như VGG-16, ZFNet, để dễ dàng cho việc giải thích, chúng ta sẽ lấy ví dụ ở đây là mạng VGG-16. Mạng VGG-16 chứa 13 convolutions layer kích thước 3×3 cùng với 5 max pooling layer kích thước 2×2. Khi đầu vào là một ảnh có kích thước 3×W×H, đầu ra sẽ nhận được 3×W′×H′ với W′=W/16 và H′=H/16



2.2. Sử dụng một cửa sổ trượt lên convolutional features.



Để tạo ra region proposals, chúng ta sử dụng một network hay còn gọi là cửa sổ trượt (sliding-window) kích thước n×n trượt trên convolutional features. Đầu ra của network này là đầu vào của 2 fully-connected layer dự

đoán vị trí của regions (box-regression layer), cũng như xác suất chứa object(box-classification) của hộp ấy. Tại mỗi vị trí của cửa sổ trượt chúng ta dự đoán đồng thời nhiều nhiều region proposal cùng một lúc, gọi k là số proposal tương ứng với mỗi vị trí. Vậy regreg layer có 4k đầu ra dự đoán vị trí của kk proposal, cls layer chứa 2k đầu ra dự đoán xác suất chứa vật thể của proposal.

Tại sao phải tạo ra những anchors này. Theo cách hiểu của bản thân tôi thì, trong bài toán xác định vị trí vật thể, số lượng đầu ra của mỗi ảnh là khác nhau. Ví dụ một bức ảnh có thể có 2 vật thể, một bức ảnh khác có 4 vật thể. Vì số lượng output là không cố định ta phải dựa vào các anchor để cố định hóa số lượng output này.

Đối với mỗi bức ảnh, ta đều sinh ra các anchors tương ứng phụ thuộc vào kích cỡ của ảnh đó, bằng cách tính giá trị overlap của anchors với ground truth boxes, ta có thể xác định được anchors đó là positive hay negative.

RPN (

(features): Sequential (

(0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): ReLU (inplace)

(2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(3): ReLU (inplace)

(4): MaxPool2d (size=(2, 2), stride=(2, 2), dilation=(1, 1))

(5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(6): ReLU (inplace)

(7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(8): ReLU (inplace)

(9): MaxPool2d (size=(2, 2), stride=(2, 2), dilation=(1, 1))

(10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(11): ReLU (inplace)

(12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(13): ReLU (inplace)

(14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

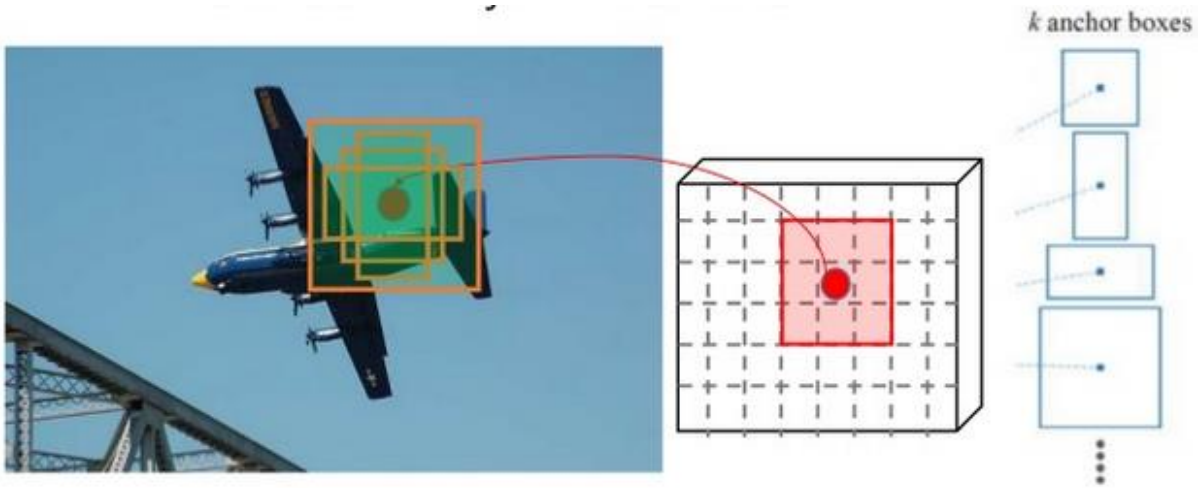
(15): ReLU (inplace)

(16): MaxPool2d (size=(2, 2), stride=(2, 2), dilation=(1, 1))

(17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

```
    (18): ReLU (inplace)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU (inplace)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU (inplace)
    (23): MaxPool2d (size=(2, 2), stride=(2, 2), dilation=(1, 1))
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU (inplace)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU (inplace)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU (inplace)
    (30): MaxPool2d (size=(2, 2), stride=(2, 2), dilation=(1, 1))
  )
  (conv1): Conv2d (
    (conv): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (relu): ReLU (inplace)
  )
  (score_conv): Conv2d (
    (conv): Conv2d(512, 18, kernel_size=(1, 1), stride=(1, 1))
  )
  (bbox_conv): Conv2d (
    (conv): Conv2d(512, 36, kernel_size=(1, 1), stride=(1, 1)))
  )
```

2.3. Anchors



Sau khi đã có đầu ra của các region proposal, chúng ta sẽ tìm hiểu về khái niệm anchors. Tại mỗi vị trí của sliding window trên convolutional features, chúng ta tạo ra k anchors tương ứng ở hình ảnh gốc. Ta sử dụng 1 hình vuông, 2 hình chữ nhật với tỉ lệ chiều rộng, chiều dài là 1-2, 2-1, cùng với 3 kích cỡ khác nhau, như vậy k=3×3=9.

Các anchors này sẽ được gán mác là positive hoặc negative dựa vào diện tích overlap với ground truth box theo luật như sau.

- Các anchor được phân loại là positive nếu :
    - Là anchor có tỉ lệ diện tích chồng chéo trên diện tích chồng chập (Intersection-over- Union - viết tắt IoU) overlap lớn nhất với một ground truth box.
    - Là anchor có tỉ lệ IoU với một ground truth lớn hơn 0.7
- Các anchor được phân noại là negative nếu có giá trị IoU bé hơn 0.3
- Các anchor không thỏa mãn 2 điều kiện nêu trên thì bỏ qua. Không được đánh giá trong quá trình training object.

Ta tạo ra những anchors này với hai mục dích chính như sau:

- Dựa phân loại của anchor, để dự đoán xác suất chứa vật thể của các region proposal
- Dựa vào khoảng cách từ anchor đến ground truth box, để dự đoán vị trí của bounding box.

Từ đây ta xác định được tiêu đầu ra của box-regression layer và box-classification được nhắc tới ở phần cấu trúc mạng RPN.

Box-classification dự đoán xác suất chứa vật thể của kk region proposal, tương ứng với kkanchor tại từng vị trí của sliding-window.

Box-regression dự đoán khoảng cách tư anchor đến ground truth box tương ứng.

2.4. Loss function

Loss function sẽ được định nghĩa theo công thức sau

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*)$$

Với i là index của anchor trong mini-batch và p_i là xác suất dự đoán của anchor i là một đối tượng. Giá trị nhãn ground-truth p_i^* là một nếu anchor là positive, và là không khi anchor là negative.

- t_i là một vector 4 chiều biểu diễn giá trị tọa độ của bounding box đã được dự đoán.
- t_i^* là vector 4 chiều biểu diễn giá trị tọa độ của ground-truth box tương ứng với positive anchor.
- L_{cls} là log loss của 2 class (object và non-object)
- L_{reg} dùng SmoothL1Loss

2.5. Công thức tính Smooth L1

$$loss(x,y) = \sum \begin{cases} 0.5 * (x_i - y_i)^2, if|x_i - y_i| < 1 \\ |x_i - y_i| - 0.5, otherwise \end{cases}$$

anchor_target_layer.py [dòng 208-227]

```
bbox_targets = _compute_targets(anchors, gt_boxes[argmax_overlaps, :])


bbox_inside_weights = np.zeros((len(inds_inside), 4), dtype=np.float32)
bbox_inside_weights[labels == 1, :] = np.array(cfg.TRAIN.RPN_BBOX_INSIDE_WEIGHTS)


bbox_outside_weights = np.zeros((len(inds_inside), 4), dtype=np.float32)
if cfg.TRAIN.RPN_POSITIVE_WEIGHT < 0:
    # uniform weighting of examples (given non-uniform sampling)
    num_examples = np.sum(labels >= 0) + 1
    positive_weights = np.ones((1, 4)) * 1.0 / num_examples
    negative_weights = np.ones((1, 4)) * 1.0 / num_examples
else:
```

```
        assert ((cfg.TRAIN.RPN_POSITIVE_WEIGHT > 0) &
              (cfg.TRAIN.RPN_POSITIVE_WEIGHT < 1))
        positive_weights = (cfg.TRAIN.RPN_POSITIVE_WEIGHT /
                    (np.sum(labels == 1)) + 1)
        negative_weights = ((1.0 - cfg.TRAIN.RPN_POSITIVE_WEIGHT) /
                    (np.sum(labels == 0)) + 1)
      bbox_outside_weights[labels == 1, :] = positive_weights
      bbox_outside_weights[labels == 0, :] = negative_weights
```

bbox_inside_weights tương ứng với giá trị nhãn p_i^* có giá trị bằng một khi anchor tương ứng là positive anchors bbox_outside_weights là hệ số để cân bằng giữa positive anchor và negative anchors và đã nhân với giá trị $1/N_{reg}$. Trong cấu hình đưa ra bởi tác giả thì TRAIN.RPN_POSITIVE_WEIGHT = -1. Lúc này giá trị hệ số là bằng nhau. Định ngĩa của loss function

```
layer {
  name: "rpn_loss_bbox"
  type: "SmoothL1Loss"
  bottom: "rpn_bbox_pred"
  bottom: "rpn_bbox_targets"
  bottom: 'rpn_bbox_inside_weights'
  bottom: 'rpn_bbox_outside_weights'
  top: "rpn_loss_bbox"
  loss_weight: 1
  smooth_l1_loss_param { sigma: 3.0 }
}
```

3. Thực nghiệm

Kết quả thực nghiệm kiểm chứng thuật toán Faster R-CNN được thực hiện trên Jupyter Notebook và được xuất sang tập tin FastRCNN_TEST_RESULT.html kèm theo. Trong thực nghiệm trên dữ liệu dataset Coco, với tập dữ liệu này ta có 330 ngàn hình ảnh, 1.5 triệu đối tượng, 80 phân loại và 250 ngàn đối tượng người với các thông tin mô tả kèm theo.

4. Lời kết

Qua bài tổng kết này em đã nắm được một phần lý thuyết cũng như cách huấn luyện mạng Faster R-CNN. Với những kiến thức được thầy hướng dẫn trên lớp cũng phần nào giúp em nắm rỏ hơn về các khái niệm. Em xin cảm ơn thầy Duy và thầy Khang đã hướng dẫn em và các bạn trong quá trình học tập Chuyên đề Nhận Dạng Thị Giác Và Ứng Dụng này. Em xin chân thành cảm ơn.