

Bài 13 - Model SSD trong Object Detection

05 Oct 2019 - phamdinhhkhanh

Menu

- 1. Giới thiệu SSD model
 - Một số định nghĩa
- 2. Single Shot Detector là gì?
 - 2.1. Kiến trúc của mô hình
 - 2.2. Quá trình huấn luyện
- 3. Code
 - 3.1. Keras Layers
 - 3.1.2. Anchor Box
 - 3.1.2. Model
 - 3.2. Khởi tạo model
- 4. Tổng kết
- 5. Tài liệu.

1. Giới thiệu SSD model

Ở bài 12 (<https://phamdinhhkhanh.github.io/2019/09/29/OverviewObjectDetection.html>) tôi đã giới thiệu đến các bạn tổng thể các lớp mô hình khác nhau trong object detection. Các kiến trúc cũ hơn có thể kể đến như R-CNN, fast R-CNN. Đặc điểm của chúng là tốc độ xử lý thấp, không đáp ứng được trong việc object detection realtime. Các mạng state-of-art hơn như SSD và YOLOv2, YOLOv3 là những kiến trúc có tốc độ xử lý nhanh mà vẫn đảm bảo về độ chính xác nhờ những thay đổi trong kiến trúc mạng nhằm gói gọn quá trình phát hiện và phân loại vật thể trong 1 lần và cắt bớt được các xử lý không cần thiết.

Trong bài này chúng ta sẽ tìm hiểu về kiến trúc, cách thức hoạt động đi kèm ví dụ thực tiễn để xây dựng một lớp mô hình SSD (Single Shot MultiBox Detector) trong object detection.

Cũng giống như hầu hết các kiến trúc object detection khác, đầu vào của SSD là tọa độ bounding box của vật thể (hay còn gọi là offsets của bounding box) và nhãn của vật thể chứa trong bounding box. Điểm đặc biệt làm nên tốc độ của SSD model là mô hình sử dụng một mạng neural duy nhất. Cách tiếp cận của nó dựa trên việc nhận diện vật thể trong các feature maps (là một output shape 3D của một mạng deep CNN sau khi bỏ các fully connected layers cuối) có độ phân giải khác nhau. Mô hình sẽ tạo ra một lưới các ô vuông gọi là grid cells trên các feature maps, mỗi ô được gọi là một cell và từ tâm của mỗi cell xác định một tập hợp các boxes mặc định (default boxes) để dự đoán khung hình có khả năng bao quanh vật thể. Tại thời điểm dự báo, mạng neural sẽ trả về 2 giá trị đó là: phân phối xác suất nhãn của vật thể chứa trong bounding box và một tọa độ gọi là offsets của bounding box. Quá trình huấn luyện cũng là quá trình tinh chỉnh xác suất nhãn và bounding box về đúng với các giá trị ground truth input của mô hình (gồm nhãn và offsets bounding box).

Thêm nữa, network được kết hợp bởi rất nhiều các feature maps với những độ phân giải khác nhau giúp phát hiện được những vật thể đa dạng các kích thước và hình dạng. Trái với mô hình fast R-CNN, SSD bỏ qua bước tạo mặt nạ region proposal network để đề xuất vùng vật thể. Thay vào đó tất cả quá trình phát hiện vật thể và phân loại vật thể được thực hiện trong cùng 1 mạng. Bản thân tên của mô hình - Single Shot MultiBox Detector cũng nói lên được rằng mô hình sử dụng nhiều khung hình box với tỷ lệ scales khác nhau nhằm nhận diện vùng vật thể và phân loại

vật thể, giảm thiểu được bước tạo region proposal network so với fast R-CNN nên tăng tốc độ xử lý lên nhiều lần mà tốc độ xử lý vẫn đảm bảo. Bên dưới là bảng so sánh tốc độ running của các mô hình object detection.

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 × 600
Fast YOLO	52.7	155	1	98	448 × 448
YOLO (VGG16)	66.4	21	1	98	448 × 448
SSD300	74.3	46	1	8732	300 × 300
SSD512	76.8	19	1	24564	512 × 512
SSD300	74.3	59	8	8732	300 × 300
SSD512	76.8	22	8	24564	512 × 512


Hình 1: Bảng so sánh tốc độ xử lý và độ chính xác của các lớp model object detection (source: table 7 - SSD: Single Shot MultiBox Detector (<https://arxiv.org/pdf/1512.02325.pdf>)). Ta thấy SSD512 (mô hình SSD với kích thước đầu vào của ảnh là 512 x 512 x 3) có độ chính xác mAP là cao nhất trong khi tốc độ xử lý gần đạt mức realtime là 22 fps.

Tóm gọn lại mô hình SSD sẽ là kết hợp của 2 bước:

- Trích xuất các feature map từ mạng CNN.
- Áp dụng convolutional filters (hoặc kernel filters) để phát hiện vật thể trên các feature map có độ phân giải (resolution) khác nhau.

Một số định nghĩa

- **scale:** Độ phóng đại so với khung hình gốc. VD: Nếu khung hình gốc có giá trị là (w, h) thì sau scale khung hình mới có kích thước là (sw, sh) . Giá trị của s thường nằm trong khoảng $s \in (0, 1]$. Scale sẽ kết hợp với aspect ratio để nhận được các khung hình có tỷ lệ cạnh w/h khác nhau.
- **aspect ratio:** Tỷ lệ cạnh, được đo bằng tỷ lệ giữa w/h nhằm xác định hình dạng tương đối của khung hình bao chứa vật thể. Chẳng hạn nếu vật thể là người thường có aspect ratio = 1 : 3 hoặc xe cộ nhìn từ phía trước là 1 : 1.
- **bounding box:** Khung hình bao chứa vật thể được xác định trong quá trình huấn luyện.
- **ground truth box:** Khung hình được xác định trước từ bộ dữ liệu thông qua tọa độ (c_x, c_y, w, h) giúp xác định vật thể.
- **offsets:** Các tọa độ (c_x, c_y, w, h) để xác định vật thể.
- **IoU:** Tỷ lệ Intersection of Union là tỷ lệ đo lường mức độ giao nhau giữa 2 khung hình (thường là khung hình dự báo và khung hình ground truth) để nhằm xác định 2 khung hình overlap không. Tỷ lệ này được tính dựa trên phần diện tích giao nhau giữa 2 khung hình với phần tổng diện tích giao nhau và không giao nhau giữa chúng.

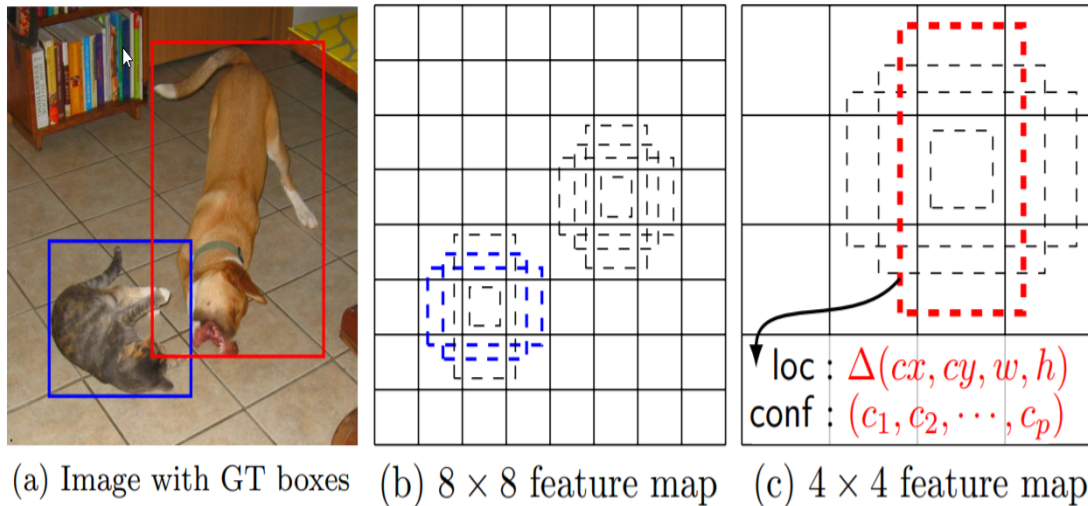


$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

- **positive matching prediction:** Khung được dự báo (predicted box) là vùng có vật thể là đúng, được xác định dựa trên tỷ lệ IoU > 0.5 giữa predicted box với ground truth box.

- **negative matching prediction:** Khung được dự báo (predicted box) là vùng không chứa vật thể là đúng, cũng được xác định dựa trên $\text{IoU} < 0.5$ giữa predicted box với ground truth box.

2. Single Shot Detector là gì?



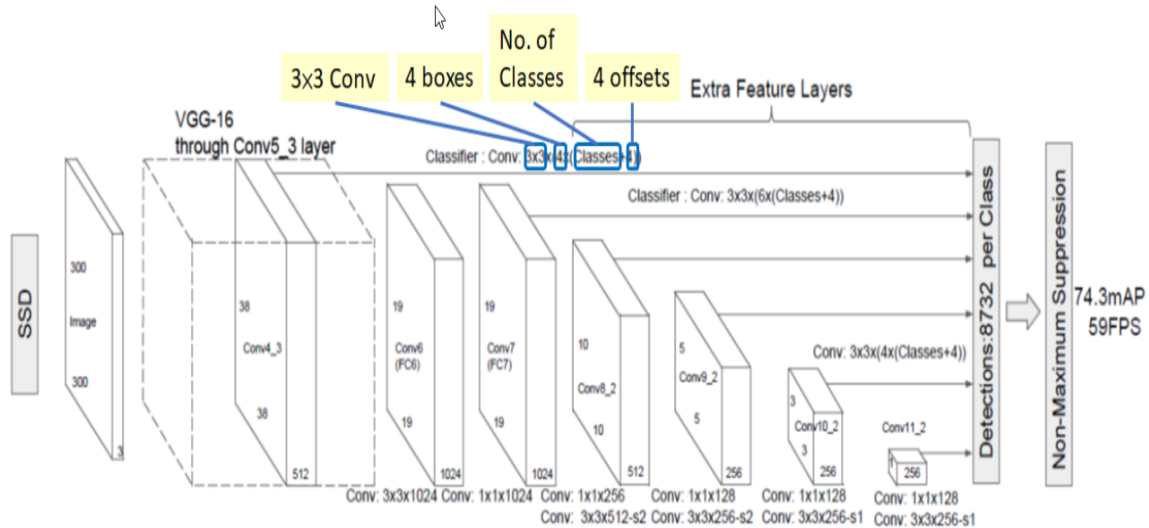
Hình 2: Cách thức phân chia feature map để nhận diện các hình ảnh với những kích thước khác nhau.

SSD chỉ cần duy nhất đầu vào là 1 bức ảnh và các ground truth boxes ám chỉ vị trí bounding box các vật thể trong suốt quá trình huấn luyện. Trong quá trình phát hiện vật thể, trên mỗi một feature map, chúng ta đánh giá các một tập hợp nhỏ gồm những default boxes tương ứng với các tỷ lệ cạnh khác nhau (aspect ratio) lên các features map có kích thước (scales) khác nhau (chẳng hạn kích thước 8×8 và 4×4 trong hình (b) và (c)). Đối với mỗi default box (các boxes nét đứt trong hình) ta cần dự báo một phân phối xác suất $\mathbf{c} = (c_1, c_2, \dots, c_n)$ tương ứng với các class $C = C_1, C_2, \dots, C_n$. Tại thời điểm huấn luyện, đầu tiên chúng ta cần match default boxes với ground truth boxes sao cho mức độ sai số được đo lường qua localization loss là nhỏ nhất (thường là hàm Smooth L1 - sẽ trình bày ở mục 2.2). Sau đó ta sẽ tìm cách tối thiểu hóa sai số của nhãn dự báo tương ứng với mỗi vật thể được phát hiện trong default boxes thông qua confidence loss (thường là hàm softmax - sẽ trình bày ở mục 2.2).

Như vậy loss function của object detection sẽ khác với loss function của các tác vụ image classification ở chỗ có thêm localization loss về sai số vị trí của predicted boxes so với ground truth boxes.

Đó là nguyên lý hoạt động chung của SSD. Tuy nhiên kiến trúc các layers và hàm loss function của SSD cụ thể là gì ta sẽ tìm hiểu bên dưới.

2.1. Kiến trúc của mô hình



Hình 3: Sơ đồ kiến trúc của mạng SSD.

SSD dựa trên một tiến trình lan truyền thuận của một kiến trúc chuẩn (chẳng hạn VGG16) để tạo ra một khối feature map output gồm 3 chiều ở giai đoạn sớm. Chúng ta gọi kiến trúc mạng này là base network (tính từ input Image đến Conv7 trong hình 3). Sau đó chúng ta sẽ thêm những kiến trúc phía sau base network để tiến hành nhận diện vật thể như phần Extra Feature Layers trong sơ đồ. Các layers này được diễn giải đơn giản như sau:

- **Các layer của mô hình SSD:**

- **Input Layer:** Nhận input đầu vào là các bức ảnh có kích thước (width x height x channels) = $300 \times 300 \times 3$ đối với kiến trúc SSD300 hoặc $500 \times 500 \times 3$ đối với kiến trúc SSD500.
- **Conv5_3 Layer:** Chính là base network sử dụng kiến trúc của VGG16 nhưng loại bỏ một số layers fully connected ở cuối cùng. Output của layer này chính là Conv4_3 Layer và là một feature map có kích thước $38 \times 38 \times 512$.
- **Conv4_3 Layer:** Ta có thể coi Conv4_3 là một feature map có kích thước $38 \times 38 \times 512$. Trên feature map này ta sẽ áp dụng 2 biến đổi chính đó là:

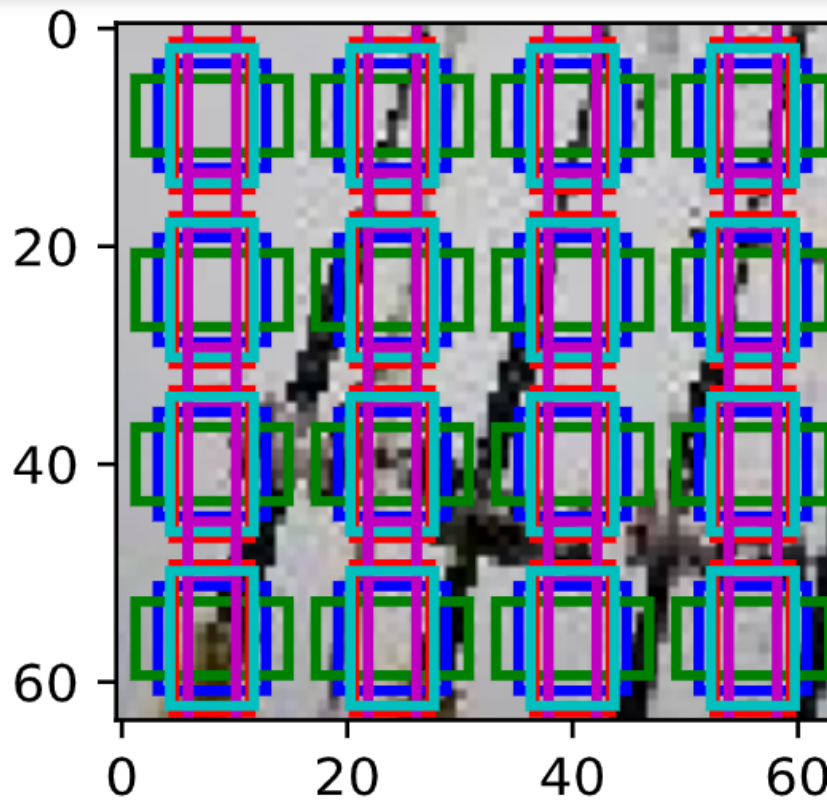
Áp dụng một convolutional layer như một mạng CNN thông thường để thu được output layer tiếp theo. Cụ thể convolutional layer có convolutional kernel kích thước $3 \times 3 \times 1024$, đầu ra thu được Conv6 có kích thước là $19 \times 19 \times 1024$.

Đồng thời ở bước này ta cũng áp dụng một classifier (như trong sơ đồ) và cũng dựa trên convolutional filter kích thước 3×3 để nhằm nhận diện vật thể trên feature map. Đây là một quá trình khá phức tạp vì nó phải đảm bảo phát hiện vật thể (thông qua phát hiện bounding box) và phân loại vật thể. Quá trình này thực hiện tương tự như mô tả ở hình 2. Đầu tiên ta sẽ phân chia feature map kích thước $38 \times 38 \times 512$ thành một grid cell kích thước 38×38 (bỏ qua độ sâu vì ta sẽ thực hiện tích chập trên toàn bộ độ sâu). Sau đó mỗi một cell trên grid cell sẽ tạo ra 4 default bounding boxes với các tỷ lệ khung hình khác nhau (aspect ratio), mỗi một default bounding box ta cần tìm các tham số sau: phân phối xác suất của nhãn là một véc tơ có $n_classes + 1$ chiều (Lưu ý số lượng classes luôn cộng thêm 1 cho background). Đồng thời chúng ta cần thêm 4 tham số là offsets để xác định bounding box của vật thể trong khung hình. Do đó trên một default bounding box sẽ có $n_classes + 4$ tham số và trên 1 cell sẽ có $4 \times (n_classes + 4)$ output cần dự báo. Nhân với số cells của Conv4_3 để thu được số lượng output là một tensor kích thước $38 \times 38 \times 4 \times (n_classes + 5)$, trong trường hợp coi background cũng là 1 nhãn thì tensor có kích thước $38 \times 38 \times 4 \times (n_classes + 4)$. Và số lượng các bounding box được sản sinh ra là $38 \times 38 \times 4$.

- Quá trình áp dụng classifier lên feature map cũng tương tự với các layer Conv7, Conv8_2, Conv9, Conv10_2, Conv11_2. Shape của các layer sau sẽ phụ thuộc vào cách thức áp dụng tích chập (convolutional process) ở layer trước, kích thước kernel filter (như trong sơ đồ trên thì kernel_size luôn là 3×3) và stride (độ lớn bước nhảy) của tích chập. Trên mỗi cell thuộc feature map ta xác định một lượng 4 hoặc 6 các default bounding boxes. Do đó, số lượng các default boxes sản sinh ra ở các layers tiếp theo lần lượt như sau:
 - **Conv7**: $19 \times 19 \times 6 = 2166$ boxes (6 boxes/cell)
 - **Conv8_2**: $10 \times 10 \times 6 = 600$ boxes (6 boxes/cell)
 - **Conv9_2**: $5 \times 5 \times 6 = 150$ boxes (6 boxes/cell)
 - **Conv10_2**: $3 \times 3 \times 4 = 36$ boxes (4 boxes/cell)
 - **Conv11_2**: $1 \times 1 \times 4 = 4$ boxes (4 boxes/cell) Tổng số lượng các boxes ở output sẽ là: $5776 + 2166 + 600 + 150 + 36 + 4 = 8732$. Tức là chúng ta cần phải dự đoán class cho khoảng 8732 khung hình ở output. Số lượng này lớn hơn rất nhiều so với YOLO khi chỉ phải dự đoán chỉ 98 khung hình ở output. Đó là lý do tại sao thuật toán có tốc độ chậm hơn so với YOLO. Các bạn có hình dung ra một khung hình ở output chúng ta cần dự đoán các chiều nào không? Tôi xin nhắc lại đó là các vec tơ output tương ứng cho mỗi default bounding box dạng:

$$y^T = [\underbrace{x, y, w, h}_{\text{bounding box}}, \underbrace{c_1, c_2, \dots, c_C}_{\text{scores of C classes}}]$$

- **Áp dụng các feature map với các kích thước khác nhau:** Sau khi thu được feature map ở base network. Chúng ta sẽ tiếp tục thêm các convolutional layers. Những layers này sẽ nhằm giảm kích thước của feature map từ đó giảm số lượng khung hình cần dự báo và cho phép dự báo và nhận diện vật thể ở nhiều hình dạng kích thước khác nhau. Những feature map có kích thước lớn sẽ phát hiện tốt các vật thể nhỏ và các feature map kích thước nhỏ giúp phát hiện tốt hơn các vật thể lớn. Cụ thể hơn về kích thước kernel filters sẽ xem ở sơ đồ kiến trúc trong phần Extra Features Layers.
- **Dự báo thông qua mạng tích chập đối với object:** Mỗi một feature layer thêm vào ở Extra Features Layers sẽ tạo ra một tập hợp cố định các output y giúp nhận diện vật thể trong ảnh thông qua áp dụng các convolutional filters. Kích thước ở đầu ra (with x height \times channel) ở mỗi loại kích thước feature layer sẽ phụ thuộc vào kernel filters và được tính toán hoàn toàn tương tự như đối với mạng neural tích chập thông thường. Xem thêm mục 1. lý thuyết về mạng tích chập trong Giới thiệu mạng neural tích chập (<https://www.kaggle.com/phamdinhhkhanh/convolutional-neural-network>) để hiểu tích chập được tính như thế nào và output shape là bao nhiêu.
- **Default box và tỷ lệ cạnh (aspect ratio):** Chúng ta cần liên kết một tập hợp default bounding boxes với mỗi một cell trên feature map. Các default boxes sẽ phân bố lát gạch trên feature map theo thứ tự từ trên xuống dưới và từ trái qua phải để tính tích chập, do đó vị trí của mỗi default box tương ứng với cell mà nó liên kết là cố định tương ứng với một vùng ảnh trên bức ảnh gốc. Cụ thể như hình ảnh minh họa bên dưới:



Hình 4: Vị trí của các default bounding box trên bức ảnh gốc khi áp dụng trên feature map có kích thước 4×4 . Như vậy grid cells sẽ có kích thước là 4×4 và trên mỗi cell ta sẽ xác định 4 defaults bounding boxes khác nhau như hình vẽ. Tâm của các bounding boxes này là trùng nhau và chính là tọa độ tâm của các cell mà nó liên kết.

Tại mỗi một default bounding box của feature map chúng ta dự báo 4 offsets tương ứng với một tọa độ và kích thước của nó. 4 offsets ở đây được hiểu là một tọa độ gồm 4 tham số (c_x, c_y, w, h) . Trong đó (c_x, c_y) giúp xác định tâm và (w, h) là kích thước dài rộng của bounding box. Thành phần thứ 2 được dự báo là điểm số của bounding box tương ứng với mỗi class. Lưu ý ta sẽ có thêm một class thứ $C + 1$ để đánh dấu trường hợp default bounding box không có vật thể (hoặc rơi vào background).

- Ví dụ đối với một feature map có kích thước $m \times n$ tương ứng với p channels (chẳng hạn như kích thước 8×8 hoặc 4×4), một kernel filter kích thước $3 \times 3 \times p$ sẽ được áp dụng trên toàn bộ feature layer.
- Các giá trị trong kernel này chính là các tham số của mô hình và được tinh chỉnh trong quá trình training.
- Các kernel filter sẽ dự đoán đồng thời **Xác suất nhãn** và **kích thước offset** tương ứng với tọa độ của default box.
- Với mỗi location (hoặc cell) nằm trên feature map ta sẽ liên kết nó với k bounding boxes. Các boxes này có kích thước khác nhau và tỷ lệ cạnh khác nhau.
- Với mỗi một bounding box, chúng ta tính được phân phối điểm của C classes là $c = (c_1, c_2, \dots, c_C)$ và 4 offsets tương ứng với kích thước ban đầu của default bounding box.
- Kết quả cuối cùng ta thu được $(C + 4) \times mnk$ outputs.

Các default box của chúng ta tương tự như anchor boxes trong mạng faster R-CNN nhưng được áp dụng trên một vài feature maps với những độ phân giải khác nhau. Điều này cho phép các default bounding box phân biệt hiệu quả kích thước vật thể khác nhau.

Kết thúc phần này chúng ta đã hiểu được kiến trúc các layer của mạng SSD. Tuy nhiên quá trình huấn luyện và hàm loss function của SSD vẫn còn là một điều bí ẩn. Liệu hàm loss function của SSD có gì khác so với các thuật toán Image classification? Quá trình tối ưu cần xét đến những mất mát nào? Hãy tìm hiểu ở phần tiếp theo.

Top

2.2. Quá trình huấn luyện

Chiến lược mapping default box Trong suốt quá trình huấn luyện ta cần mapping các default boxes có tỷ lệ aspect ratio khác nhau với ground truth box. Để mapping được chúng với nhau ta cần đo lường chỉ số IoU (Intersection of Union) hoặc chỉ số Jaccard overlap index (https://en.wikipedia.org/wiki/Jaccard_index) được dùng để đo lường tỷ lệ diện tích giao nhau giữa 2 vùng hình ảnh so với tổng diện tích (không tính phần giao nhau) của chúng. Chúng ta sẽ match các default boxes với bất kì ground truth nào có threshold > 0.5 .

Như chúng ta đã biết trên mỗi cell chỉ quy định một số lượng nhất định (4 hoặc 6, tùy từng feature map) các default bounding box. Vậy các default bounding box này được xác định trước thông qua aspect ratio và scale hay ngẫu nhiên? Trên thực tế là chúng hầu hết được xác định từ trước để giảm thiểu sự đa dạng về số lượng khung hình/cell mà vẫn bounding được hầu hết các vật thể. Tập hợp các khung hình được xác định phải đảm bảo sao cho mỗi một ground truth bất kì đều có thể tìm được một default bounding box gần nó nhất. Do đó một thuật toán K-mean clustering được thực hiện trên aspect ratio của mỗi ground truth image nhằm phân cụm các khung hình ground truth thành các nhóm tương đương về hình dạng. Tâm của các clusters (còn gọi là centroids) sẽ được dùng làm các giá trị aspect ratio đại diện để tính default bounding box. Tôi hi vọng các bạn hiểu những gì tôi vừa trình bày? Không quá phức tạp phải không?

Huấn luyện để tìm ra object: Việc dự báo các object sẽ được thực hiện trên tập hợp các khung hình output của mạng SSD. Đặt $x_{ij}^k = 0, 1$ là chỉ số đánh giá cho việc matching giữa default bounding box thứ i với ground truth box thứ j đối với nhãn thứ k . Trong quá trình mapping chúng ta có thể có nhiều bounding box được map vào cùng 1 ground truth box với cùng 1 nhãn dự báo nên tổng $\sum_i x_{ij}^k \geq 1$. Hàm loss function là tổng có trọng số của localization loss (loc) và confidence loss (conf):

$$L(x, c, p, g) = \frac{1}{N} (L_{conf}(x, c) + \alpha L_{loc}(x, p, g)) \quad (1)$$

Trong đó N là số lượng các default boxes matching với ground truth boxes. Ta nhận thấy giá trị của hàm loss function của SSD hoàn toàn giống với faster R-CNN và bao gồm 2 thành phần:

1. localization loss: là một hàm Smooth L1 đo lường sai số giữa tham số của box dự báo (predicted box) (p) và ground truth box (g) như bên dưới. Chúng ta sẽ cần hồi qui các offsets cho tâm (x, y) và của default bounding box (d) và các chiều dài h và chiều rộng w .

$$L_{loc}(x, p, g) = \sum_{i \in Pos} \sum_{m \in \{x, y, w, h\}} x_{ij}^k L_1^{\text{smooth}}(p_i^m - \hat{g}_j^m)$$

Như vậy Localization loss chỉ xét trên các positive matching ($i \in Pos$) giữa predicted bounding box với ground truth bounding box. Nếu $IoU > 0.5$ thì được coi là positive matching (tức predicted bounding box chứa vật thể). Trái lại, nếu $IoU \leq 0.5$ ta không cần quan tâm và coi như xóa các predicted bounding box này khỏi hình ảnh. Thành phần

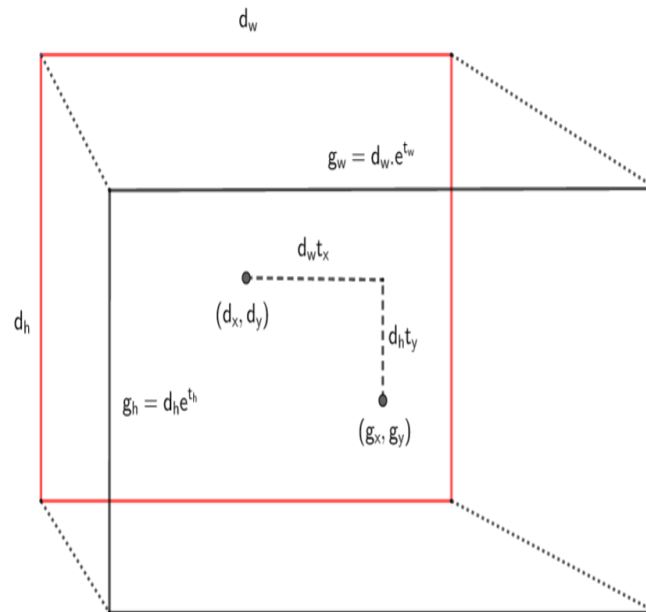
$\sum_{m \in \{x, y, w, h\}} x_{ij}^k L_1^{\text{smooth}}(p_i^m - \hat{g}_j^m)$ chính là tổng khoảng cách giữa predicted box (p) và ground truth box (g) trên 4 offsets x, y, w, h .

Nếu để nguyên các giá trị tọa độ tâm và kích thước của khung hình sẽ rất khó để xác định sai số một cách chuẩn xác. Ta hãy so sánh sai số trong trường hợp khung hình lớn và khung hình bé. Trong trường hợp khung hình lớn có predicted box và ground truth box rất khớp nhau. Tuy nhiên do khung hình quá to nên khoảng cách tâm của chúng sẽ lớn một chút, giả định là aspect ratio của chúng bằng nhau. Còn trường hợp khung hình bé, sai số của tâm giữa predicted box và ground truth box có thể bé hơn trường hợp khung hình lớn về số tuyệt đối. Nhưng điều đó không

có nghĩa rằng predicted box và ground truth box của khung hình bé là rất khớp nhau. Chúng có thể cách nhau rất xa.

Do đó chúng ta cần phải chuẩn hóa kích thước width, height và tâm sao cho không có sự khác biệt trong trường hợp khung hình bé và lớn. Một phép chuẩn hóa các offset được thực hiện như sau:

Các tham số \hat{g}^m được tính thông qua đo lường chênh lệch khoảng cách tâm và mức độ thay đổi kích thước (scale) dài và rộng giữa lần lượt ground truth box so với default box. Để hiểu rõ hơn cách tính \hat{g}^m như thế nào ta xem hình minh họa đối với tính \hat{g}^m như bên dưới:



Hình 5: Hình chữ nhật viền đen đại diện cho ground truth box và hình chữ nhật viền đỏ đại diện cho default bounding box. (d_w, d_h) lần lượt là kích thước dài rộng và (d_x, d_y) là tọa độ tâm của default bounding box. Khi đó để chuyển từ tâm của default bounding box sang tâm của ground truth box ta cần 1 phép dịch chuyển tuyến tính các khoảng $(d_w t_x, d_h t_y)$. Kích thước các chiều dài và rộng được scale so với default bounding box số lần (e^{t_h}, e^{t_w}) .

- Khoảng cách tâm (p_x, p_y) của predicted box so với tâm (d_x, d_y) của default box:

$$\hat{g}_x = \frac{g_x - d_x}{d_w} \triangleq t_x$$

$$\hat{g}_y = \frac{g_y - d_y}{d_h} \triangleq t_y$$

Top

- Độ scale theo chiều dài và rộng (p_w, p_h) của predicted box so với chiều dài và rộng (d_w, d_h) của ground truth box:

$$\hat{g}_w = \log\left(\frac{d_w}{g_w}\right) \triangleq t_w$$

$$\hat{g}_h = \log\left(\frac{d_h}{g_h}\right) \triangleq t_h$$

Kí hiệu \triangleq nghĩa là đặt về trái bằng về phải. Ta nhận thấy các giá trị t_x, t_y, t_w, t_h là những tham số tính chỉnh kích thước của bounding box nhận giá trị trong khoảng từ $(-\infty, +\infty)$. Nếu các giá trị của t_x, t_y càng lớn thì khoảng cách giữa 2 tâm ground truth box và default box càng lớn. Giá trị của t_w, t_h càng lớn, tỷ lệ chênh lệch kích thước dài rộng giữa ground truth box và default box càng lớn. Ta gọi giá trị bộ tham số (t_x, t_y, t_w, t_h) là bộ tham số kích thước chuẩn hóa của ground truth box theo default box. Tương tự ta cũng tính được bộ tham số kích thước chuẩn hóa của predicted box theo default box bằng cách thế g_m bằng p_m với $m \in x, y, w, h$ trong những phương trình trên. Để hiểu hơn về các tham số chuẩn hóa có thể xem Faster R-CNN - Mục A Loss Function for Learning Region Proposals (<https://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf>). Khi đó khoảng cách giữa predicted box và ground truth box sẽ càng gần nếu khoảng cách giữa các bộ tham số chuẩn hóa giữa chúng càng gần. Tức khoảng cách giữa 2 véc tơ p và \hat{g} càng nhỏ càng tốt.

Khoảng cách này được đo lường thông qua hàm L_1^{smooth} là một kết hợp giữa norm chuẩn bậc 1 (đối với các giá trị tuyệt đối của x lớn) và norm chuẩn bậc 2 (đối với các giá trị tuyệt đối của x nhỏ) theo công thức sau:

$$L_1^{\text{smooth}}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

Trong trường hợp x là một véc tơ thì thay x ở về phải bằng giá trị norm chuẩn bậc 1 của x kí hiệu là $|x|$. Việc lựa chọn hàm loss function là smooth L1 là để giá trị của đạo hàm gradient descent cố định khi $|x|$ lớn và smoothing khi x nhỏ. Về norm chuẩn các bạn có thể xem trong bài tổng hợp ML appendix (<https://www.kaggle.com/phamdinhhkhanh/ml-appendix>). Trong phương trình của hàm localization loss thì các hằng số mà ta đã biết chính là \hat{g} . Biến cần tìm giá trị tối ưu chính là p . Sau khi tìm ra được nghiệm tối ưu của p ta sẽ tính ra predicted box nhờ phép chuyển đổi từ default box tương ứng.

2. confidence loss: là một hàm mất mát được tính toán dựa trên sai số dự báo nhãn. Đối với mỗi một positive match prediction, chúng ta phạt loss function theo confidence score của các nhãn tương ứng. Đối với mỗi một negative match prediction, chúng ta phạt loss function theo confidence score của nhãn '0' là nhãn đại diện cho background không chứa vật thể. Cụ thể hàm confidence loss như bên dưới:

$$L_{\text{conf}}(x, c) = - \sum_{i \in \text{Pos}} x_{ij}^k \log(\hat{c}_i^k) - \sum_{i \in \text{Neg}} \log(\hat{c}_i^0)$$

Trong trường hợp positive match prediction thì vùng được dự báo có vật thể chính xác là chứa vật thể. Do đó việc dự báo nhãn cho nó sẽ tương tự như một bài toán classification với hàm softmax thông thường có dạng $-\sum_{i \in \text{Pos}} x_{ij}^k \log(\hat{c}_i^p)$. Trong trường hợp negative match prediction tức vùng được dự báo là không chứa vật thể chúng ta sẽ chỉ có duy nhất một nhãn là 0. Và tất nhiên ta đã biết trước bounding box là không chứa vật thể nên xác suất để xảy ra nhóm 0 là $x_{ij}^0 = 1$. Do đó hàm softmax có dạng $-\sum_{i \in \text{Neg}} \log(\hat{c}_i^0)$. Top

Hàm loss function cuối cùng được tính là tổng của 2 confidence loss và localization loss như (1).

Lựa chọn kích cỡ (scales) và tỷ lệ cạnh (aspect ratio):

Các default boundary box được lựa chọn thông qua aspect ratio và scales. SSD sẽ xác định một tỷ lệ scale tương ứng với mỗi một features map trong Extra Feature Layers. Bắt đầu từ bên trái, Conv4_3 phát hiện các object tại các scale nhỏ nhất là $s_{min} = 0.2$ (đôi khi là 0.1) và sau đó gia tăng tuyến tính để layer cuối cùng ở phía bên phải có scale là $s_{max} = 0.9$ theo công thức:

$$s_k = s_{min} + \frac{s_{max} - s_{min}}{m - 1} (k - 1), k \in [1, m]$$

Với k là số thứ tự của layers. Kết hợp giữa giá trị scale với aspect ratio chúng ta sẽ tính được width và height của default boxes. Với các layers có 6 dự báo, SSD sẽ tạo ra 5 default boxes với các aspect ratios lần lượt là: 1, 2, 3, 1/2, 1/3. Sau đó width và height của default boxes được tính theo công thức:

$$w = scale * \sqrt{\text{aspect ratio}}$$

$$h = \frac{scale}{\sqrt{\text{aspect ratio}}}$$

Trong trường hợp aspect ratio = 1 thì ta sẽ thêm một default bounding box thứ 6 với scale được tính theo công thức:

$$s'_k = \sqrt{s_k s_{k+1}}$$

3. Code

Thuật toán SSD là một thuật toán rất phức tạp, có nhiều layers và các phases xử lý khác nhau. Vì vậy code này tôi không tự mình viết hết mà tham khảo từ SSD keras (https://github.com/pierluigiferrari/ssd_keras). Trong code tôi có chỉnh sửa lại một số đoạn và kèm theo diễn giải về từng step xử lý như thế nào.

3.1. Keras Layers

3.1.2. Anchor Box

Phần tinh túy nhất của SSD có lẽ là việc xác định các layers output của anchor box (hoặc default bounding box) ở các feature map. anchor box layer sẽ nhận đầu vào ra một feature map có kích thước (feature_width, feature_height, n_channels) và các scales, aspect ratios, trả ra đầu ra là một tensor kích thước (feature_width, feature_height, n_boxes, 4), trong đó chiều cuối cùng đại diện cho 4 offsets của bounding box như mô tả trong **Default box và tỷ lệ cạnh (aspect ratio)** của mục 2.1.

Code biến đổi khá phức tạp. Tôi trong đó các phần biến đổi chính được thực hiện trong hàm `call()`.

- **Bước 1:** Từ scale, size (giá trị lớn nhất của width và height), và aspect ratio ta xác định kích thước các cạnh của các bounding box theo công thức:

$$\begin{cases} box_h = scale * size / \sqrt{aspect\ ratio} \\ box_w = scale * size * \sqrt{aspect\ ratio} \end{cases}$$

Top

- **Bước 2:** Từ các cell trên feature map chiếu lại trên ảnh input image để thu được step khoảng cách giữa các center point của mỗi cell theo công thức:

$$\begin{cases} step_h = img_h / feature_map_h \\ step_w = img_w / feature_map_w \end{cases}$$

- **Bước 3:** Tính tọa độ các điểm (c_x, c_y, w, h) trên hình ảnh gốc dựa trên phép linear interpolation qua hàm `np.linspace()` (<https://docs.scipy.org/doc/numpy/reference/generated/numpy.linspace.html>).

$$\begin{cases} c_x = np.linspace(start_w, end_w, feature_map_w) \\ c_y = np.linspace(start_h, end_h, feature_map_h) \end{cases}$$

Kết quả trả về là một tensor có shape là $(feature_width, feature_height, n_boxes, 8)$, trong đó chiều cuối cùng = 8 tương ứng với 4 offsets của default bounding box và 4 variances đại diện cho các scales của default bounding box.

Top

```

1      from __future__ import division
2      import numpy as np
3      import keras.backend as K
4      from keras.engine.topology import InputSpec
5      from keras.engine.topology import Layer
6
7      from bounding_box_utils.bounding_box_utils import convert_coordinate
8
9      class AnchorBoxes(Layer):
10         '''
11         Tác dụng: Tạo ra một output tensor chứa tọa độ của các anchor box
12         Một tập hợp các 2D anchor boxes được tạo ra dựa trên aspect ratio
13
14         Input shape:
15             4D tensor shape `(batch, channels, height, width)` nếu `dim_ordering`
16             or `(batch, height, width, channels)` nếu `dim_ordering = 'th'`
17
18         Output shape:
19             5D tensor of shape `(batch, height, width, n_boxes, 8)`.
20             Chiều cuối cùng gồm 4 tọa độ của anchor box và 4 giá trị biến thiên
21         '''
22
23         def __init__(self,
24             img_height,
25             img_width,
26             this_scale,
27             next_scale,
28             aspect_ratios=[0.5, 1.0, 2.0],
29             two_boxes_for_ar1=True,
30             this_steps=None,
31             this_offsets=None,
32             clip_boxes=False,
33             variances=[0.1, 0.1, 0.2, 0.2],
34             coords='centroids',
35             normalize_coords=False,
36             **kwargs):
37             '''
38
39             Arguments:
40                 img_height (int): chiều cao input images.
41                 img_width (int): chiều rộng input images.
42                 this_scale (float): một giá trị float thuộc [0, 1], nhân
43                 next_scale (float): giá trị tiếp theo của scale. Được thay thế bằng
44                     `self.two_boxes_for_ar1 == True`.
45                 aspect_ratios (list, optional): tập hợp các aspect ratio
46                 two_boxes_for_ar1 (bool, optional): Được sử dụng chỉ khi
47                     Nếu `True`, hai default boxes được tạo ra khi aspect ratio
48                     default box thứ 2 sử dụng trung bình hình học giữa scale
49                 clip_boxes (bool, optional): Nếu đúng `True`, giới hạn tọa độ
50                 variances (list, optional): Tập hợp gồm 4 giá trị floats
51                 coords (str, optional): Tọa độ của box được sử dụng trong
52                     hoặc 'corners' định dạng `(xmin, ymin, xmax, ymax)`
53                 normalize_coords (bool, optional): Nếu `True` mô hình sử dụng
54             '''
55
56             if K.backend() != 'tensorflow':
57                 raise TypeError("This layer only supports TensorFlow at the moment")

```

Top

```

58     if (this_scale < 0) or (next_scale < 0) or (this_scale > 1):
59         raise ValueError("`this_scale` must be in [0, 1] and `next_scale` must be in [0, 1]")
60
61     if len(variances) != 4:
62         raise ValueError("4 variance values must be passed, but got %d" % len(variances))
63     variances = np.array(variances)
64     if np.any(variances <= 0):
65         raise ValueError("All variances must be >0, but the variances are %s" % str(variances))
66
67     self.img_height = img_height
68     self.img_width = img_width
69     self.this_scale = this_scale
70     self.next_scale = next_scale
71     self.aspect_ratios = aspect_ratios
72     self.two_boxes_for_ar1 = two_boxes_for_ar1
73     self.this_steps = this_steps
74     self.this_offsets = this_offsets
75     self.clip_boxes = clip_boxes
76     self.variances = variances
77     self.coords = coords
78     self.normalize_coords = normalize_coords
79     # Tính toán số lượng boxes trên 1 cell. TH aspect ratios = 1
80     if (1 in aspect_ratios) and two_boxes_for_ar1:
81         self.n_boxes = len(aspect_ratios) + 1
82     else:
83         self.n_boxes = len(aspect_ratios)
84     super(AnchorBoxes, self).__init__(**kwargs)
85
86     def build(self, input_shape):
87         self.input_spec = [InputSpec(shape=input_shape)]
88         super(AnchorBoxes, self).build(input_shape)
89
90     def call(self, x, mask=None):
91         """
92         Return: Trả về 1 anchor box tensor dựa trên shape của input
93
94         Tensor này được thiết kế như là hằng số và không tham gia vào quá trình cập nhật.
95
96         Arguments:
97             x (tensor): 4D tensor có shape `(batch, channels, height, width)` hoặc `(batch, height, width, channels)` nếu `dim_ordering='th'`
98             hoặc `(batch, height, width, channels)` nếu `dim_ordering='tf'`
99         """
100         #####
101         # Bước 1: Tính toán width và height của box với mỗi aspect ratio
102         #####
103         # Cận gần hơn của hình ảnh có thể được sử dụng để tính `width` và `height`
104         size = min(self.img_height, self.img_width)
105         # Tính toán box widths và heights cho toàn bộ aspect ratios
106         wh_list = []
107         for ar in self.aspect_ratios:
108             if (ar == 1):
109                 # Tính anchor box thông thường khi aspect ratio = 1.
110                 box_height = box_width = self.this_scale * size
111                 wh_list.append((box_width, box_height))
112             if self.two_boxes_for_ar1:
113                 # Tính version lớn hơn của anchor box sử dụng `sqrt`
114                 box_height = box_width = np.sqrt(self.this_scale * size * ar)
115                 wh_list.append((box_width, box_height))

```

```

116         else:
117             # Trường hợp còn lại box_height = scale/sqrt(aspect
118             box_height = self.this_scale * size // np.sqrt(ar)
119             box_width = int(self.this_scale * size * np.sqrt(ar)
120             wh_list.append((box_width, box_height))
121         # append vào width height list
122         wh_list = np.array(wh_list)
123
124         # Định hình input shape
125         if K.common.image_dim_ordering() == 'tf':
126             batch_size, feature_map_height, feature_map_width, featu
127         else:
128             batch_size, feature_map_channels, feature_map_height, fea
129
130         # Tính các center points của grid of box. Chúng là duy nhất
131         #####
132         # Bước 2: Tính các step size. Khoảng cách là bao xa giữa các
133         #####
134         if (self.this_steps is None):
135             step_height = self.img_height // feature_map_height
136             step_width = self.img_width // feature_map_width
137         else:
138             if isinstance(self.this_steps, (list, tuple)) and (len(s
139                 step_height = self.this_steps[0]
140                 step_width = self.this_steps[1]
141             elif isinstance(self.this_steps, (int, float)):
142                 step_height = self.this_steps
143                 step_width = self.this_steps
144         # Tính toán các offsets cho anchor box center point đầu tiên
145         if (self.this_offsets is None):
146             offset_height = 0.5
147             offset_width = 0.5
148         else:
149             if isinstance(self.this_offsets, (list, tuple)) and (len
150                 offset_height = self.this_offsets[0]
151                 offset_width = self.this_offsets[1]
152             elif isinstance(self.this_offsets, (int, float)):
153                 offset_height = self.this_offsets
154                 offset_width = self.this_offsets
155         #####
156         # Bước 3: Tính toán các tọa độ của (cx, cy, w, h) theo tọa đ
157         #####
158         # Bây h chúng ta có các offsets và step sizes, tính grid của
159         cy = np.linspace(offset_height * step_height, (offset_height
160         cx = np.linspace(offset_width * step_width, (offset_width +
161         cx_grid, cy_grid = np.meshgrid(cx, cy)
162         cx_grid = np.expand_dims(cx_grid, -1)
163         cy_grid = np.expand_dims(cy_grid, -1)
164
165
166         # Tạo một 4D tensor có shape `(feature_map_height, feature_m
167         # Chiều cuối cùng sẽ chứa `(cx, cy, w, h)`
168         boxes_tensor = np.zeros((feature_map_height, feature_map_wid
169
170         boxes_tensor[:, :, :, 0] = np.tile(cx_grid, (1, 1, self.n_bo
171         boxes_tensor[:, :, :, 1] = np.tile(cy_grid, (1, 1, self.n_bo
172         boxes_tensor[:, :, :, 2] = wh_list[:, 0] # đặt w
173         boxes_tensor[:, :, :, 3] = wh_list[:, 1] # đặt h

```

```

174
175     # Chuyển `(cx, cy, w, h)` sang `(xmin, xmax, ymin, ymax)`
176     boxes_tensor = convert_coordinates(boxes_tensor, start_index:
177
178     # Nếu `clip_boxes` = True, giới hạn các tọa độ nằm trên bound
179     if self.clip_boxes:
180         x_coords = boxes_tensor[:, :, :, [0, 2]]
181         x_coords[x_coords >= self.img_width] = self.img_width - 1
182         x_coords[x_coords < 0] = 0
183         boxes_tensor[:, :, :, [0, 2]] = x_coords
184         y_coords = boxes_tensor[:, :, :, [1, 3]]
185         y_coords[y_coords >= self.img_height] = self.img_height - 1
186         y_coords[y_coords < 0] = 0
187         boxes_tensor[:, :, :, [1, 3]] = y_coords
188
189     # Nếu `normalize_coords` = True, chuẩn hóa các tọa độ nằm tr
190     if self.normalize_coords:
191         boxes_tensor[:, :, :, [0, 2]] /= self.img_width
192         boxes_tensor[:, :, :, [1, 3]] /= self.img_height
193
194     if self.coords == 'centroids':
195         # Convert `(xmin, ymin, xmax, ymax)` to `(cx, cy, w, h)`
196         boxes_tensor = convert_coordinates(boxes_tensor, start_i
197     elif self.coords == 'minmax':
198         # Convert `(xmin, ymin, xmax, ymax)` to `(xmin, xmax, ym
199         boxes_tensor = convert_coordinates(boxes_tensor, start_i
200
201     # Tạo một tensor chứa các variances và append vào `boxes_ten
202     variances_tensor = np.zeros_like(boxes_tensor) # shape `(fea
203     variances_tensor += self.variances # Mở rộng thêm variances
204     # Bây h `boxes_tensor` trở thành tensor kích thước `(feature
205     boxes_tensor = np.concatenate((boxes_tensor, variances_tenso
206
207     # Bây h chuẩn bị trước một chiều cho `boxes_tensor` đại diện
208     # ta được một 5D tensor kích thước `(batch_size, feature_ma
209     boxes_tensor = np.expand_dims(boxes_tensor, axis=0)
210     boxes_tensor = K.tile(K.constant(boxes_tensor, dtype='float3
211
212     return boxes_tensor
213
214     def compute_output_shape(self, input_shape):
215         if K.common.image_dim_ordering() == 'tf':
216             batch_size, feature_map_height, feature_map_width, featu
217         else:
218             batch_size, feature_map_channels, feature_map_height, fe
219         return (batch_size, feature_map_height, feature_map_width, s
220
221     def get_config(self):
222         config = {
223             'img_height': self.img_height,
224             'img_width': self.img_width,
225             'this_scale': self.this_scale,
226             'next_scale': self.next_scale,
227             'aspect_ratios': list(self.aspect_ratios),
228             'two_boxes_for_ar1': self.two_boxes_for_ar1,
229             'clip_boxes': self.clip_boxes,
230             'variances': list(self.variances),
231             'coords': self.coords,

```

Top


```
232         'normalize_coords': self.normalize_coords
233     }
234     base_config = super(AnchorBoxes, self).get_config()
235     return dict(list(base_config.items()) + list(config.items()))
```

Bên dưới ta sẽ kiểm nghiệm kết quả test AnchorBoxes layer khi truyền thử nghiệm đầu vào là tensor x .

Top

```

1  # Test output of Anchor box
2  import tensorflow as tf
3  x = tf.random.normal(shape = (4, 38, 38, 512))
4
5  aspect_ratios_per_layer=[[1.0, 2.0, 0.5],
6                           [1.0, 2.0, 0.5, 3.0, 1.0/3.0],
7                           [1.0, 2.0, 0.5, 3.0, 1.0/3.0],
8                           [1.0, 2.0, 0.5, 3.0, 1.0/3.0],
9                           [1.0, 2.0, 0.5],
10                          [1.0, 2.0, 0.5]]
11
12 two_boxes_for_ar1=True
13 steps=[8, 16, 32, 64, 100, 300]
14 offsets=None
15 clip_boxes=False
16 variances=[0.1, 0.1, 0.2, 0.2]
17 coords='centroids'
18 normalize_coords=True
19 subtract_mean=[123, 117, 104]
20 divide_by_stddev=None
21 swap_channels=[2, 1, 0]
22 confidence_thresh=0.01
23 iou_threshold=0.45
24 top_k=200
25 nms_max_output_size=400
26
27 # Thiết lập tham số
28 img_height = 300
29 img_width = 300
30 img_channels = 3
31 mean_color = [123, 117, 104]
32 swap_channels = [2, 1, 0]
33 n_classes = 20
34 scales = [0.1, 0.2, 0.37, 0.54, 0.71, 0.88, 1.05]
35 aspect_ratios = [[1.0, 2.0, 0.5],
36                  [1.0, 2.0, 0.5, 3.0, 1.0/3.0],
37                  [1.0, 2.0, 0.5, 3.0, 1.0/3.0],
38                  [1.0, 2.0, 0.5, 3.0, 1.0/3.0],
39                  [1.0, 2.0, 0.5],
40                  [1.0, 2.0, 0.5]]
41 two_boxes_for_ar1 = True
42 steps = [8, 16, 32, 64, 100, 300]
43 offsets = [0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
44 clip_boxes = False
45 variances = [0.1, 0.1, 0.2, 0.2]
46 normalize_coords = True
47
48
49 anchors = AnchorBoxes(img_height, img_width, this_scale=scales[1], ne:
50 print('anchors shape: ', anchors.get_shape())

```

```

1  anchors shape: (4, 38, 38, 4, 8)

```

Top

Như vậy kết quả output của anchors box trả ra là (feature_width, feature_height, n_boxes, 8) là hợp lý.

3.1.2. Model

Các bước thực hiện để khởi tạo cấu trúc của mạng ssd_300 bao gồm:

- **Bước 1:** Xây dựng kiến trúc mạng bao gồm:
 - Bước 1.1: Xây dựng kiến trúc mạng base network theo VGG16 đã loại bỏ các fully connected layers ở cuối.
 - Bước 1.2: Áp dụng các convolutional filter có kích thước (3 x 3) để tính toán ra features map.
 - Bước 1.3: Xác định output phân phối xác suất theo các classes ứng với mỗi một default bounding box.
 - Bước 1.4: Xác định output các tham số offset của default bounding boxes tương ứng với mỗi cell trên các features map.
 - Bước 1.5: Tính toán các AnchorBoxes làm cơ sở để dự báo offsets cho các predicted bounding boxes bao quan vật thể. Giá trị của các AnchorBoxes chỉ hỗ trợ trong quá trình tính toán offsets và không xuất hiện ở output như giá trị cần dự báo.
- **Bước 2:** Reshape lại các output để đưa chúng về kích thước của (feature_map_w, feature_map_h, n_boxes, -1) . Trong đó -1 đại diện cho chiều cuối cùng được tính dựa vào các chiều còn lại theo hàm reshape.
- **Bước 3:** Liên kết các khối tensorflow output của bước 2 được tính từ confidence, các offsets của bounding box và các offsets của anchor box.
- **Bước 4:** Kết nối với output. Thêm layers softmax trước confidence của bounding box.

Top

```

1  from __future__ import division
2  import numpy as np
3  from keras.models import Model
4  from keras.layers import Input, Lambda, Activation, Conv2D, MaxPooling2D
5  from keras.regularizers import l2
6  import keras.backend as K
7
8  from keras_layers.keras_layer_AnchorBoxes import AnchorBoxes
9  from keras_layers.keras_layer_L2Normalization import L2Normalization
10 from keras_layers.keras_layer_DecodeDetections import DecodeDetections
11 from keras_layers.keras_layer_DecodeDetectionsFast import DecodeDetectionsFast
12
13 def ssd_300(image_size,
14             n_classes,
15             mode='training',
16             l2_regularization=0.0005,
17             min_scale=None,
18             max_scale=None,
19             scales=None,
20             aspect_ratios_global=None,
21             aspect_ratios_per_layer=[[1.0, 2.0, 0.5],
22                                     [1.0, 2.0, 0.5, 3.0, 1.0/3.0],
23                                     [1.0, 2.0, 0.5, 3.0, 1.0/3.0],
24                                     [1.0, 2.0, 0.5, 3.0, 1.0/3.0],
25                                     [1.0, 2.0, 0.5],
26                                     [1.0, 2.0, 0.5]],
27             two_boxes_for_ar1=True,
28             steps=[8, 16, 32, 64, 100, 300],
29             offsets=None,
30             clip_boxes=False,
31             variances=[0.1, 0.1, 0.2, 0.2],
32             coords='centroids',
33             normalize_coords=True,
34             subtract_mean=[123, 117, 104],
35             divide_by_stddev=None,
36             swap_channels=[2, 1, 0],
37             confidence_thresh=0.01,
38             iou_threshold=0.45,
39             top_k=200,
40             nms_max_output_size=400,
41             return_predictor_sizes=False):
42     '''
43     Xây dựng model SSD300 với keras.
44     Base network được sử dụng là VGG16.
45
46     Chú ý: Yêu cầu Keras>=v2.0; TensorFlow backend>=v1.0.
47
48     Arguments:
49         image_size (tuple): Kích thước image input `(height, width, channels)`.
50         n_classes (int): Số classes, chẳng hạn 20 cho Pascal VOC data.
51         mode (str, optional): Một trong những dạng 'training', 'inference',
52         'training' mode: Đầu ra của model là raw prediction tensors.
53         'inference' và 'inference_fast' modes: raw predictions đã được xử lý.
54         l2_regularization (float, optional): L2-regularization rate.
55         min_scale (float, optional): Nhân tố scaling nhỏ nhất cho các
56         của hình ảnh input.
57         max_scale (float, optional): Nhân tố scale lớn nhất cho các

```

Top

```

58     scales (list, optional): List các số floats chứa các nhân tố
59         List này phải lớn hơn số lượng các predictor layers là 1
60         Trong TH sử dụng scales thì interpolate theo min_scale và
61     aspect_ratios_global (list, optional): List của các aspect r
62     aspect_ratios_per_layer (list, optional): List của các list
63         Nếu được truyền vào sẽ override `aspect_ratios_global`.
64     two_boxes_for_ar1 (bool, optional): Chỉ áp dụng khi aspect r
65         Nếu `True`, 2 anchor boxes sẽ được tạo ra ứng với aspect
66         được tạo thành bằng trung bình hình học của scale và nex
67     steps (list, optional): `None` hoặc là list với rất nhiều cá
68         Mỗi phần tử đại diện cho mỗi một predictor layer có bao
69         steps có thể gồm 2 số đại diện cho (step_width, step_hei
70         nếu không có steps nào được đưa ra thì chúng ta sẽ tính
71     offsets (list, optional): None hoặc là các con số đại diện c
72     clip_boxes (bool, optional): Nếu `True`, giới hạn tọa độ các
73     variances (list, optional): Một list gồm 4 số floats >0. Một
74     coords (str, optional): Tọa độ của box được sử dụng bên trong
75         Có thể là dạng 'centroids' format `(cx, cy, w, h)` (box c
76         and height), 'minmax' format `(xmin, xmax, ymin, ymax)`,
77     normalize_coords (bool, optional): Được đặt là `True` nếu mo
78         chẳng hạn nếu model dự báo tọa độ box nằm trong [0, 1] t
79     subtract_mean (array-like, optional): `None` hoặc một array
80         Chẳng hạn truyền vào một list gồm 3 số nguyên để tính to
81     divide_by_stddev (array-like, optional): `None` hoặc một arr
82     swap_channels (list, optional): Là `False` hoặc một list các
83     confidence_thresh (float, optional): Một số float nằm trong
84     iou_threshold (float, optional): Một float nằm trong khoảng
85         sẽ được xem xét là chứa vật thể bên trong nó.
86     top_k (int, optional): Điểm dự báo cáo nhất được giữ trong m
87     nms_max_output_size (int, optional): Số lượng lớn nhất các d
88     return_predictor_sizes (bool, optional): Nếu `True`, hàm số
89         một list chứa các chiều của predictor layers.
90
91 Returns:
92     model: The Keras SSD300 model.
93     predictor_sizes (optional): Một numpy array chứa các phần `(l
94
95 References:
96     https://arxiv.org/abs/1512.02325v5
97     '''
98
99     n_predictor_layers = 6 # Số lượng các preductor convolutional la
100     n_classes += 1 # Số lượng classes, + 1 để tính thêm background c
101     l2_reg = l2_regularization # tham số chuẩn hóa của norm chuẩn l2
102     img_height, img_width, img_channels = image_size[0], image_size[
103
104     #####
105     # Một số lỗi ngoại lệ.
106     #####
107
108     if aspect_ratios_global is None and aspect_ratios_per_layer is N
109         raise ValueError("`aspect_ratios_global` and `aspect_ratios_
110     if aspect_ratios_per_layer:
111         if len(aspect_ratios_per_layer) != n_predictor_layers:
112             raise ValueError("It must be either aspect_ratios_per_la
113
114                                     Top
115     # Tạo list scales
116     if (min_scale is None or max_scale is None) and scales is None:

```

```

116         raise ValueError("Either `min_scale` and `max_scale` or `scales` must be provided")
117     if scales:
118         if len(scales) != n_predictor_layers+1:
119             raise ValueError("It must be either scales is None or len(scales) == n_predictor_layers+1")
120     else:
121         scales = np.linspace(min_scale, max_scale, n_predictor_layers+1)
122
123     if len(variances) != 4:
124         raise ValueError("4 variance values must be passed, but {} variance values were passed".format(len(variances)))
125     variances = np.array(variances)
126     if np.any(variances <= 0):
127         raise ValueError("All variances must be >0, but the variance {} was less than 0".format(variances))
128
129     if (not (steps is None)) and (len(steps) != n_predictor_layers):
130         raise ValueError("You must provide at least one step value per predictor layer")
131
132     if (not (offsets is None)) and (len(offsets) != n_predictor_layers):
133         raise ValueError("You must provide at least one offset value per predictor layer")
134
135     #####
136     # Tính các tham số của anchor box.
137     #####
138
139     # Thiết lập aspect ratios cho mỗi predictor layer (chỉ cần thiết lập một lần)
140     if aspect_ratios_per_layer:
141         aspect_ratios = aspect_ratios_per_layer
142     else:
143         aspect_ratios = [aspect_ratios_global] * n_predictor_layers
144
145     # Tính số lượng boxes được dự báo / 1 cell cho mỗi predictor layer
146     # Chúng ta cần biết bao nhiêu channels các predictor layers cần
147     if aspect_ratios_per_layer:
148         n_boxes = []
149         for ar in aspect_ratios_per_layer:
150             if (1 in ar) & two_boxes_for_ar1:
151                 n_boxes.append(len(ar) + 1) # +1 cho trường hợp aspect_ratio = 1
152             else:
153                 n_boxes.append(len(ar))
154     else: # Nếu chỉ 1 global aspect ratio list được truyền vào thì số lượng boxes được dự báo / 1 cell cho mỗi predictor layer
155         if (1 in aspect_ratios_global) & two_boxes_for_ar1:
156             n_boxes = len(aspect_ratios_global) + 1
157         else:
158             n_boxes = len(aspect_ratios_global)
159         n_boxes = [n_boxes] * n_predictor_layers
160
161     if steps is None:
162         steps = [None] * n_predictor_layers
163     if offsets is None:
164         offsets = [None] * n_predictor_layers
165
166     #####
167     # Xác định các hàm số cho Lambda layers bên dưới.
168     #####
169
170     def identity_layer(tensor):
171         return tensor
172
173     def input_mean_normalization(tensor):

```

Top

```

174         return tensor - np.array(subtract_mean)
175
176     def input_stddev_normalization(tensor):
177         return tensor / np.array(divide_by_stddev)
178
179     def input_channel_swap(tensor):
180         if len(swap_channels) == 3:
181             return K.stack([tensor[..., swap_channels[0]], tensor[...
182         elif len(swap_channels) == 4:
183             return K.stack([tensor[..., swap_channels[0]], tensor[...
184
185     #####
186     # Bước 1: Xây dựng network.
187     #####
188
189     x = Input(shape=(img_height, img_width, img_channels))
190
191     x1 = Lambda(identity_layer, output_shape=(img_height, img_width,
192     if not (subtract_mean is None):
193         x1 = Lambda(input_mean_normalization, output_shape=(img_heigh
194     if not (divide_by_stddev is None):
195         x1 = Lambda(input_stddev_normalization, output_shape=(img_he
196     if swap_channels:
197         x1 = Lambda(input_channel_swap, output_shape=(img_height, im
198
199     #####
200     # Bước 1.1: Tính toán base network là mạng VGG16
201     #####
202
203     conv1_1 = Conv2D(64, (3, 3), activation='relu', padding='same',
204     conv1_2 = Conv2D(64, (3, 3), activation='relu', padding='same',
205     pool1 = MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='
206
207     conv2_1 = Conv2D(128, (3, 3), activation='relu', padding='same',
208     conv2_2 = Conv2D(128, (3, 3), activation='relu', padding='same',
209     pool2 = MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='
210
211     conv3_1 = Conv2D(256, (3, 3), activation='relu', padding='same',
212     conv3_2 = Conv2D(256, (3, 3), activation='relu', padding='same',
213     conv3_3 = Conv2D(256, (3, 3), activation='relu', padding='same',
214     pool3 = MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='
215
216     conv4_1 = Conv2D(512, (3, 3), activation='relu', padding='same',
217     conv4_2 = Conv2D(512, (3, 3), activation='relu', padding='same',
218     conv4_3 = Conv2D(512, (3, 3), activation='relu', padding='same',
219     pool4 = MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='
220
221     conv5_1 = Conv2D(512, (3, 3), activation='relu', padding='same',
222     conv5_2 = Conv2D(512, (3, 3), activation='relu', padding='same',
223     conv5_3 = Conv2D(512, (3, 3), activation='relu', padding='same',
224     pool5 = MaxPooling2D(pool_size=(3, 3), strides=(1, 1), padding='
225
226     #####
227     # Bước 1.2: Áp dụng các convolutional filter có kích thước (3 x
228     #####
229
230     fc6 = Conv2D(1024, (3, 3), dilation_rate=(6, 6), activation='relu
231     print('fully connected 6: ', fc6.get_shape())

```

Top


```

232     fc7 = Conv2D(1024, (1, 1), activation='relu', padding='same', kernel_size=(1, 1))
233     print('fully connected 7: ', fc7.get_shape())
234     conv6_1 = Conv2D(256, (1, 1), activation='relu', padding='same', kernel_size=(1, 1))
235     conv6_1 = ZeroPadding2D(padding=((1, 1), (1, 1)), name='conv6_padding')
236     conv6_2 = Conv2D(512, (3, 3), strides=(2, 2), activation='relu', kernel_size=(3, 3))
237     print('conv6_2: ', conv6_2.get_shape())
238     conv7_1 = Conv2D(128, (1, 1), activation='relu', padding='same', kernel_size=(1, 1))
239     conv7_1 = ZeroPadding2D(padding=((1, 1), (1, 1)), name='conv7_padding')
240     conv7_2 = Conv2D(256, (3, 3), strides=(2, 2), activation='relu', kernel_size=(3, 3))
241     print('conv7_2: ', conv7_2.get_shape())
242     conv8_1 = Conv2D(128, (1, 1), activation='relu', padding='same', kernel_size=(1, 1))
243     conv8_2 = Conv2D(256, (3, 3), strides=(1, 1), activation='relu', kernel_size=(3, 3))
244     print('conv8_2: ', conv8_2.get_shape())
245     conv9_1 = Conv2D(128, (1, 1), activation='relu', padding='same', kernel_size=(1, 1))
246     conv9_2 = Conv2D(256, (3, 3), strides=(1, 1), activation='relu', kernel_size=(3, 3))
247     print('conv9_2: ', conv9_2.get_shape())
248     # Feed conv4_3 vào the L2 normalization layer
249     conv4_3_norm = L2Normalization(gamma_init=20, name='conv4_3_norm')
250     print('conv4_3_norm.shape: ', conv4_3_norm.get_shape())
251
252     #####
253     # Bước 1.3: Xác định output phân phối xác suất theo các classes
254     #####
255
256     ### Xây dựng các convolutional predictor layers tại top của base
257     # Chúng ta dự báo các giá trị confidence cho mỗi box, do đó confidence layers có shape: (batch, height, width, channels)
258     # Đầu ra của confidence layers có shape: (batch, height, width, channels)
259     conv4_3_norm_mbox_conf = Conv2D(n_boxes[0] * n_classes, (3, 3), padding='same', kernel_size=(3, 3))
260     print('conv4_3_norm_mbox_conf.shape: ', conv4_3_norm_mbox_conf.get_shape())
261     fc7_mbox_conf = Conv2D(n_boxes[1] * n_classes, (3, 3), padding='same', kernel_size=(3, 3))
262     print('fc7_mbox_conf.shape: ', fc7_mbox_conf.get_shape())
263     conv6_2_mbox_conf = Conv2D(n_boxes[2] * n_classes, (3, 3), padding='same', kernel_size=(3, 3))
264     conv7_2_mbox_conf = Conv2D(n_boxes[3] * n_classes, (3, 3), padding='same', kernel_size=(3, 3))
265     conv8_2_mbox_conf = Conv2D(n_boxes[4] * n_classes, (3, 3), padding='same', kernel_size=(3, 3))
266     conv9_2_mbox_conf = Conv2D(n_boxes[5] * n_classes, (3, 3), padding='same', kernel_size=(3, 3))
267     print('conv9_2_mbox_conf: ', conv9_2_mbox_conf.get_shape())
268
269     #####
270     # Bước 1.4: Xác định output các tham số offset của default bounding boxes
271     #####
272
273     # Chúng ta dự báo 4 tọa độ cho mỗi box, do đó localization predictor layers có shape: (batch, height, width, channels)
274     # Output shape của localization layers: (batch, height, width, channels)
275     conv4_3_norm_mbox_loc = Conv2D(n_boxes[0] * 4, (3, 3), padding='same', kernel_size=(3, 3))
276     print('conv4_3_norm_mbox_loc: ', conv4_3_norm_mbox_loc.get_shape())
277     fc7_mbox_loc = Conv2D(n_boxes[1] * 4, (3, 3), padding='same', kernel_size=(3, 3))
278     conv6_2_mbox_loc = Conv2D(n_boxes[2] * 4, (3, 3), padding='same', kernel_size=(3, 3))
279     conv7_2_mbox_loc = Conv2D(n_boxes[3] * 4, (3, 3), padding='same', kernel_size=(3, 3))
280     conv8_2_mbox_loc = Conv2D(n_boxes[4] * 4, (3, 3), padding='same', kernel_size=(3, 3))
281     conv9_2_mbox_loc = Conv2D(n_boxes[5] * 4, (3, 3), padding='same', kernel_size=(3, 3))
282     print('conv9_2_mbox_loc: ', conv9_2_mbox_loc.get_shape())
283
284     #####
285     # Bước 1.5: Tính toán các AnchorBoxes làm cơ sở để dự báo offset
286     #####
287
288     Top
289     ### Khởi tạo các anchor boxes (được gọi là "priors" trong code gốc)
290     # Shape output của anchors: (batch, height, width, n_boxes, 8)

```

```

290 conv4_3_norm_mbox_priorbox = AnchorBoxes(img_height, img_width,
291                                           two_boxes_for_ar1=two_boxes_for_ar1,
292                                           variances=variances, coords=coords)
293 print('conv4_3_norm_mbox_priorbox: ', conv4_3_norm_mbox_priorbox.get_shape())
294 fc7_mbox_priorbox = AnchorBoxes(img_height, img_width, this_scale,
295                                 two_boxes_for_ar1=two_boxes_for_ar1,
296                                 variances=variances, coords=coords)
297 print('fc7_mbox_priorbox: ', fc7_mbox_priorbox.get_shape())
298 conv6_2_mbox_priorbox = AnchorBoxes(img_height, img_width, this_scale,
299                                     two_boxes_for_ar1=two_boxes_for_ar1,
300                                     variances=variances, coords=coords)
301 print('conv6_2_mbox_priorbox: ', conv6_2_mbox_priorbox.get_shape())
302 conv7_2_mbox_priorbox = AnchorBoxes(img_height, img_width, this_scale,
303                                     two_boxes_for_ar1=two_boxes_for_ar1,
304                                     variances=variances, coords=coords)
305 print('conv7_2_mbox_priorbox: ', conv7_2_mbox_priorbox.get_shape())
306 conv8_2_mbox_priorbox = AnchorBoxes(img_height, img_width, this_scale,
307                                     two_boxes_for_ar1=two_boxes_for_ar1,
308                                     variances=variances, coords=coords)
309 print('conv8_2_mbox_priorbox: ', conv8_2_mbox_priorbox.get_shape())
310 conv9_2_mbox_priorbox = AnchorBoxes(img_height, img_width, this_scale,
311                                     two_boxes_for_ar1=two_boxes_for_ar1,
312                                     variances=variances, coords=coords)
313 print('conv9_2_mbox_priorbox: ', conv9_2_mbox_priorbox.get_shape())
314
315 #####
316 # Bước 2: Reshape lại các output tensor shape
317 #####
318
319 #####
320 # Bước 2.1: Reshape output của class predictions
321 #####
322
323 # Reshape các class predictions, trả về 3D tensors có shape `(batch_size, num_classes, spatial_dims)`
324 # Chúng ta muốn các classes là tách biệt nhau trên last axis để
325 conv4_3_norm_mbox_conf_reshape = Reshape((-1, n_classes), name='conv4_3_norm_mbox_conf_reshape')
326 fc7_mbox_conf_reshape = Reshape((-1, n_classes), name='fc7_mbox_conf_reshape')
327 conv6_2_mbox_conf_reshape = Reshape((-1, n_classes), name='conv6_2_mbox_conf_reshape')
328 conv7_2_mbox_conf_reshape = Reshape((-1, n_classes), name='conv7_2_mbox_conf_reshape')
329 conv8_2_mbox_conf_reshape = Reshape((-1, n_classes), name='conv8_2_mbox_conf_reshape')
330 conv9_2_mbox_conf_reshape = Reshape((-1, n_classes), name='conv9_2_mbox_conf_reshape')
331 print('conv4_3_norm_mbox_conf_reshape: ', conv4_3_norm_mbox_conf_reshape.get_shape())
332 print('fc7_mbox_conf_reshape: ', fc7_mbox_conf_reshape.get_shape())
333 print('conv9_2_mbox_conf_reshape: ', conv9_2_mbox_conf_reshape.get_shape())
334 print('conv9_2_mbox_conf_reshape: ', conv9_2_mbox_conf_reshape.get_shape())
335 print('conv9_2_mbox_conf_reshape: ', conv9_2_mbox_conf_reshape.get_shape())
336
337 #####
338 # Bước 2.2: Reshape output của bounding box predictions
339 #####
340
341 # Reshape các box predictions, trả về 3D tensors có shape `(batch_size, num_classes, spatial_dims)`
342 # Chúng ta muốn 4 tọa độ box là tách biệt nhau trên last axis để
343 conv4_3_norm_mbox_loc_reshape = Reshape((-1, 4), name='conv4_3_norm_mbox_loc_reshape')
344 fc7_mbox_loc_reshape = Reshape((-1, 4), name='fc7_mbox_loc_reshape')
345 conv6_2_mbox_loc_reshape = Reshape((-1, 4), name='conv6_2_mbox_loc_reshape')
346 conv7_2_mbox_loc_reshape = Reshape((-1, 4), name='conv7_2_mbox_loc_reshape')
347 conv8_2_mbox_loc_reshape = Reshape((-1, 4), name='conv8_2_mbox_loc_reshape')

```

```

348 conv9_2_mbox_loc_reshape = Reshape((-1, 4), name='conv9_2_mbox_lo
349 print('conv4_3_norm_mbox_loc_reshape: ', conv4_3_norm_mbox_loc_r
350 print('fc7_mbox_loc_reshape: ', fc7_mbox_loc_reshape.get_shape()
351 print('conv6_2_mbox_loc_reshape: ', conv6_2_mbox_loc_reshape.get
352 print('conv7_2_mbox_loc_reshape: ', conv7_2_mbox_loc_reshape.get
353 print('conv8_2_mbox_loc_reshape: ', conv8_2_mbox_loc_reshape.get
354 print('conv9_2_mbox_loc_reshape: ', conv9_2_mbox_loc_reshape.get
355
356 #####
357 # Bước 2.3: Reshape output của anchor box
358 #####
359
360 # Reshape anchor box tensors, trả về 3D tensors có shape `(batch,
361 conv4_3_norm_mbox_priorbox_reshape = Reshape((-1, 8), name='conv
362 fc7_mbox_priorbox_reshape = Reshape((-1, 8), name='fc7_mbox_prio
363 conv6_2_mbox_priorbox_reshape = Reshape((-1, 8), name='conv6_2_ml
364 conv7_2_mbox_priorbox_reshape = Reshape((-1, 8), name='conv7_2_ml
365 conv8_2_mbox_priorbox_reshape = Reshape((-1, 8), name='conv8_2_ml
366 conv9_2_mbox_priorbox_reshape = Reshape((-1, 8), name='conv9_2_ml
367 print('conv4_3_norm_mbox_priorbox_reshape: ', conv4_3_norm_mbox_
368 print('fc7_mbox_priorbox_reshape: ', fc7_mbox_priorbox_reshape.g
369 print('conv6_2_mbox_priorbox_reshape: ', conv6_2_mbox_priorbox_r
370 print('conv7_2_mbox_priorbox_reshape: ', conv7_2_mbox_priorbox_r
371 print('conv8_2_mbox_priorbox_reshape: ', conv8_2_mbox_priorbox_r
372 print('conv9_2_mbox_priorbox_reshape: ', conv9_2_mbox_priorbox_r
373 ### Concatenate các predictions từ các layers khác nhau
374
375 #####
376 # Bước 3: Concatenate các boxes trên layers
377 #####
378
379 #####
380 # Bước 3.1: Concatenate confidence output box
381 #####
382
383 # Axis 0 (batch) và axis 2 (n_classes hoặc 4) là xác định duy nh
384 # nên chúng ta muốn concatenate theo axis 1, số lượng các boxes
385 # Output shape của `mbox_conf`: (batch, n_boxes_total, n_classes
386 mbox_conf = Concatenate(axis=1, name='mbox_conf')([conv4_3_norm_m
387                                                     fc7_mbox_conf
388                                                     conv6_2_mbox_
389                                                     conv7_2_mbox_
390                                                     conv8_2_mbox_
391                                                     conv9_2_mbox_
392 print('mbox_conf.shape: ', mbox_conf.get_shape())
393
394 #####
395 # Bước 3.2: Concatenate location output box
396 #####
397
398 # Output shape của `mbox_loc`: (batch, n_boxes_total, 4)
399 mbox_loc = Concatenate(axis=1, name='mbox_loc')([conv4_3_norm_mb
400                                                     fc7_mbox_loc_re
401                                                     conv6_2_mbox_lo
402                                                     conv7_2_mbox_lo
403                                                     conv8_2_mbox_lo
404                                                     conv9_2_mbox_lo
405

```

```

406     print('mbox_loc.shape: ', mbox_loc.get_shape())
407
408     #####
409     # Bước 3.3: Concatenate anchor output box
410     #####
411
412     # Output shape của `mbox_priorbox`: (batch, n_boxes_total, 8)
413     mbox_priorbox = Concatenate(axis=1, name='mbox_priorbox')([conv4_
414                                                                fc7_m
415                                                                conv6_
416                                                                conv7_
417                                                                conv8_
418                                                                conv9_
419
420     print('mbox_priorbox.shape: ', mbox_priorbox.get_shape())
421
422     #####
423     # Bước 4: Tính toán output
424     #####
425
426     #####
427     # Bước 4.1 : Xây dựng các hàm loss function cho confidence
428     #####
429
430     # tọa độ của box predictions sẽ được truyền vào hàm loss function
431     # nhưng cho các dự báo lớp, chúng ta sẽ áp dụng một hàm softmax
432     mbox_conf_softmax = Activation('softmax', name='mbox_conf_softmax')
433
434     # Concatenate các class và box predictions và the anchors thành
435     # Đầu ra của `predictions`: (batch, n_boxes_total, n_classes + 4)
436     predictions = Concatenate(axis=2, name='predictions')([mbox_conf_
437     print('predictions.shape: ', predictions.get_shape())
438     if mode == 'training':
439         model = Model(inputs=x, outputs=predictions)
440     elif mode == 'inference':
441         decoded_predictions = DecodeDetections(confidence_thresh=conf
442                                                iou_threshold=iou_thre
443                                                top_k=top_k,
444                                                nms_max_output_size=n
445                                                coords=coords,
446                                                normalize_coords=norm
447                                                img_height=img_height
448                                                img_width=img_width,
449                                                name='decoded_predict
450         model = Model(inputs=x, outputs=decoded_predictions)
451     elif mode == 'inference_fast':
452         decoded_predictions = DecodeDetectionsFast(confidence_thresh=
453                                                    iou_threshold=iou
454                                                    top_k=top_k,
455                                                    nms_max_output_si
456                                                    coords=coords,
457                                                    normalize_coords=
458                                                    img_height=img_he
459                                                    img_width=img_wid
460                                                    name='decoded_pre
461         model = Model(inputs=x, outputs=decoded_predictions) Top
462     else:
463         raise ValueError("`mode` must be one of 'training', 'inference'")

```

```

464
465         if return_predictor_sizes:
466             predictor_sizes = np.array([conv4_3_norm_mbox_conf._keras_shape[1:3],
467                                         fc7_mbox_conf._keras_shape[1:3],
468                                         conv6_2_mbox_conf._keras_shape[1:3],
469                                         conv7_2_mbox_conf._keras_shape[1:3],
470                                         conv8_2_mbox_conf._keras_shape[1:3],
471                                         conv9_2_mbox_conf._keras_shape[1:3]])
472             return model, predictor_sizes
473         else:
474             return model

```

3.2. Khởi tạo model

Để khởi tạo mô hình chúng ta cần khai báo các tham số chính bao gồm:

- `img_height`: Chiều cao hình ảnh input
- `img_width`: Chiều rộng hình ảnh input
- `img_channels`: Số kênh của hình ảnh input.
- `n_classes`: Số lượng nhãn của bộ dữ liệu
- `scales`: List các giá trị scales của mô hình ở mỗi một layer detector.
- `aspect_ratios`: List các aspect ratios tương ứng ở mỗi layer detector.
- `variances`: Các tham số biến đổi dùng để tính các anchor box.

```

1  from keras.optimizers import Adam, SGD
2  from keras.callbacks import ModelCheckpoint, LearningRateScheduler, TensorBoard
3  from keras import backend as K
4  from keras.models import load_model
5  from math import ceil
6  import numpy as np
7  from matplotlib import pyplot as plt
8
9  # from models.keras_ssd300 import ssd_300
10 from keras_loss_function.keras_ssd_loss import SSDLoss
11 from keras_layers.keras_layer_AnchorBoxes import AnchorBoxes
12 from keras_layers.keras_layer_DecodeDetections import DecodeDetections
13 from keras_layers.keras_layer_DecodeDetectionsFast import DecodeDetectionsFast
14 from keras_layers.keras_layer_L2Normalization import L2Normalization
15
16 from ssd_encoder_decoder.ssd_input_encoder import SSDInputEncoder
17 from ssd_encoder_decoder.ssd_output_decoder import decode_detections,
18
19 from data_generator.object_detection_2d_data_generator import DataGenerator
20 from data_generator.object_detection_2d_geometric_ops import Resize
21 from data_generator.object_detection_2d_photometric_ops import ConvertColor
22 from data_generator.data_augmentation_chain_original_ssd import SSDDataAugmentation
23 from data_generator.object_detection_2d_misc_utils import apply_inverse_transform
24
25 %matplotlib inline

```

Thiết lập tham số cho mô hình

Top

```

1  img_height = 300
2  img_width = 300
3  img_channels = 3
4  mean_color = [123, 117, 104]
5  swap_channels = [2, 1, 0]
6  n_classes = 20
7  scales = [0.1, 0.2, 0.37, 0.54, 0.71, 0.88, 1.05]
8  aspect_ratios = [[1.0, 2.0, 0.5],
9                   [1.0, 2.0, 0.5, 3.0, 1.0/3.0],
10                  [1.0, 2.0, 0.5, 3.0, 1.0/3.0],
11                  [1.0, 2.0, 0.5, 3.0, 1.0/3.0],
12                  [1.0, 2.0, 0.5],
13                  [1.0, 2.0, 0.5]]
14  two_boxes_for_ar1 = True
15  steps = [8, 16, 32, 64, 100, 300]
16  offsets = [0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
17  clip_boxes = False
18  variances = [0.1, 0.1, 0.2, 0.2]
19  normalize_coors = True

```

Khởi tạo một mô hình `ssd_300` dựa trên các tham số đã thiết lập.

```

1  # 1: Build Keras model.
2
3  K.clear_session() # xóa các object tại session cũ.
4
5  model = ssd_300(image_size=(img_height, img_width, img_channels),
6                  n_classes=n_classes,
7                  mode='training',
8                  l2_regularization=0.0005,
9                  scales=scales,
10                 aspect_ratios_per_layer=aspect_ratios,
11                 two_boxes_for_ar1=two_boxes_for_ar1,
12                 steps=steps,
13                 offsets=offsets,
14                 clip_boxes=clip_boxes,
15                 variances=variances,
16                 normalize_coors=normalize_coors,
17                 subtract_mean=mean_color,
18                 swap_channels=swap_channels)
19
20 # 2: Chúng ta có thể load trọng số của mô hình pretrain.
21
22 weights_path = 'pretrain_model/VGG_ILSVRC_16_layers_fc_reduced.h5'
23
24 model.load_weights(weights_path, by_name=True)
25
26 # 3: Khởi tạo optimizer và compile vào model.
27
28 sgd = SGD(lr=0.001, momentum=0.9, decay=0.0, nesterov=False)
29
30 ssd_loss = SSDLoss(neg_pos_ratio=3, alpha=1.0)
31
32 model.compile(optimizer=sgd, loss=ssd_loss.compute_loss)

```

Top

Kiểm tra kiến trúc các layers của mô hình:

`model.summary()`

Top

1 Model: "model_1"

2	Layer (type)	Output Shape	Param #	Connections
3	=====	=====	=====	=====
4	input_1 (InputLayer)	(None, 300, 300, 3)	0	
5	identity_layer (Lambda)	(None, 300, 300, 3)	0	input_1
6	input_mean_normalization (Lambda)	(None, 300, 300, 3)	0	identity_layer
7	input_channel_swap (Lambda)	(None, 300, 300, 3)	0	input_mean_normalization
8	conv1_1 (Conv2D)	(None, 300, 300, 64)	1792	input_channel_swap
9	conv1_2 (Conv2D)	(None, 300, 300, 64)	36928	conv1_1
10	pool1 (MaxPooling2D)	(None, 150, 150, 64)	0	conv1_2
11	conv2_1 (Conv2D)	(None, 150, 150, 128)	73856	pool1
12	conv2_2 (Conv2D)	(None, 150, 150, 128)	147584	conv2_1
13	pool2 (MaxPooling2D)	(None, 75, 75, 128)	0	conv2_2
14	conv3_1 (Conv2D)	(None, 75, 75, 256)	295168	pool2
15	conv3_2 (Conv2D)	(None, 75, 75, 256)	590080	conv3_1
16	conv3_3 (Conv2D)	(None, 75, 75, 256)	590080	conv3_2
17	pool3 (MaxPooling2D)	(None, 38, 38, 256)	0	conv3_3
18	conv4_1 (Conv2D)	(None, 38, 38, 512)	1180160	pool3
19	conv4_2 (Conv2D)	(None, 38, 38, 512)	2359808	conv4_1
20	conv4_3 (Conv2D)	(None, 38, 38, 512)	2359808	conv4_2
21	pool4 (MaxPooling2D)	(None, 19, 19, 512)	0	conv4_3
22	conv5_1 (Conv2D)	(None, 19, 19, 512)	2359808	pool4
23	conv5_2 (Conv2D)	(None, 19, 19, 512)	2359808	conv5_1
24	conv5_3 (Conv2D)	(None, 19, 19, 512)	2359808	conv5_2
25	pool5 (MaxPooling2D)	(None, 19, 19, 512)	0	conv5_3
26	fc6 (Conv2D)	(None, 19, 19, 1024)	4719616	pool5
27	fc7 (Conv2D)	(None, 19, 19, 1024)	1049600	fc6
28	conv6_1 (Conv2D)	(None, 19, 19, 256)	262400	fc7
29	conv6_padding (ZeroPadding2D)	(None, 21, 21, 256)	0	conv6_1
30	conv6_2 (Conv2D)	(None, 10, 10, 512)	1180160	conv6_padding

Top

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

conv7_1 (Conv2D)	(None, 10, 10, 128)	65664	conv
conv7_padding (ZeroPadding2D)	(None, 12, 12, 128)	0	conv
conv7_2 (Conv2D)	(None, 5, 5, 256)	295168	conv
conv8_1 (Conv2D)	(None, 5, 5, 128)	32896	conv
conv8_2 (Conv2D)	(None, 3, 3, 256)	295168	conv
conv9_1 (Conv2D)	(None, 3, 3, 128)	32896	conv
conv4_3_norm (L2Normalization)	(None, 38, 38, 512)	512	conv
conv9_2 (Conv2D)	(None, 1, 1, 256)	295168	conv
conv4_3_norm_mbox_conf (Conv2D)	(None, 38, 38, 84)	387156	conv
fc7_mbox_conf (Conv2D)	(None, 19, 19, 126)	1161342	fc7
conv6_2_mbox_conf (Conv2D)	(None, 10, 10, 126)	580734	conv
conv7_2_mbox_conf (Conv2D)	(None, 5, 5, 126)	290430	conv
conv8_2_mbox_conf (Conv2D)	(None, 3, 3, 84)	193620	conv
conv9_2_mbox_conf (Conv2D)	(None, 1, 1, 84)	193620	conv
conv4_3_norm_mbox_loc (Conv2D)	(None, 38, 38, 16)	73744	conv
fc7_mbox_loc (Conv2D)	(None, 19, 19, 24)	221208	fc7
conv6_2_mbox_loc (Conv2D)	(None, 10, 10, 24)	110616	conv
conv7_2_mbox_loc (Conv2D)	(None, 5, 5, 24)	55320	conv
conv8_2_mbox_loc (Conv2D)	(None, 3, 3, 16)	36880	conv
conv9_2_mbox_loc (Conv2D)	(None, 1, 1, 16)	36880	conv
conv4_3_norm_mbox_conf_reshape	(None, 5776, 21)	0	conv
fc7_mbox_conf_reshape (Reshape)	(None, 2166, 21)	0	fc7
conv6_2_mbox_conf_reshape (Resh	(None, 600, 21)	0	conv
conv7_2_mbox_conf_reshape (Resh	(None, 150, 21)	0	conv
conv8_2_mbox_conf_reshape (Resh	(None, 36, 21)	0	conv
conv9_2_mbox_conf_reshape (Resh	(None, 4, 21)	0	conv
conv4_3_norm_mbox_priorbox (Anc	(None, 38, 38, 4, 8)	0	conv
fc7_mbox_priorbox (AnchorBoxes)	(None, 19, 19, 6, 8)	0	Top fc7
conv6_2_mbox_priorbox (AnchorBo	(None, 10, 10, 6, 8)	0	conv

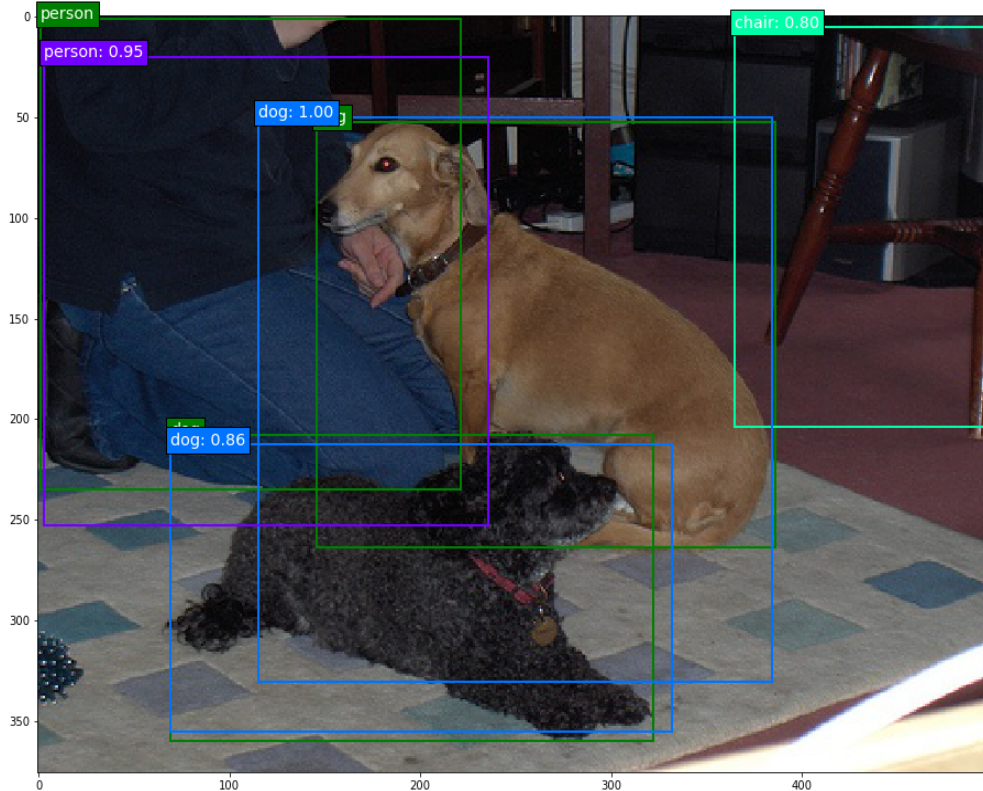
116				
117	conv7_2_mbox_priorbox	(AnchorBo (None, 5, 5, 6, 8))	0	con
118				
119	conv8_2_mbox_priorbox	(AnchorBo (None, 3, 3, 4, 8))	0	con
120				
121	conv9_2_mbox_priorbox	(AnchorBo (None, 1, 1, 4, 8))	0	con
122				
123	mbox_conf	(Concatenate) (None, 8732, 21)	0	con
124				fc7.
125				con
126				con
127				con
128				con
129				
130	conv4_3_norm_mbox_loc_reshape	((None, 5776, 4))	0	con
131				
132	fc7_mbox_loc_reshape	(Reshape) (None, 2166, 4)	0	fc7.
133				
134	conv6_2_mbox_loc_reshape	(Resha (None, 600, 4))	0	con
135				
136	conv7_2_mbox_loc_reshape	(Resha (None, 150, 4))	0	con
137				
138	conv8_2_mbox_loc_reshape	(Resha (None, 36, 4))	0	con
139				
140	conv9_2_mbox_loc_reshape	(Resha (None, 4, 4))	0	con
141				
142	conv4_3_norm_mbox_priorbox_resh	(None, 5776, 8)	0	con
143				
144	fc7_mbox_priorbox_reshape	(Resh (None, 2166, 8))	0	fc7.
145				
146	conv6_2_mbox_priorbox_reshape	((None, 600, 8))	0	con
147				
148	conv7_2_mbox_priorbox_reshape	((None, 150, 8))	0	con
149				
150	conv8_2_mbox_priorbox_reshape	((None, 36, 8))	0	con
151				
152	conv9_2_mbox_priorbox_reshape	((None, 4, 8))	0	con
153				
154	mbox_conf_softmax	(Activation) (None, 8732, 21)	0	mbo
155				
156	mbox_loc	(Concatenate) (None, 8732, 4)	0	con
157				fc7.
158				con
159				con
160				con
161				con
162				
163	mbox_priorbox	(Concatenate) (None, 8732, 8)	0	con
164				fc7.
165				con
166				con
167				con
168				con
169				
170	predictions	(Concatenate) (None, 8732, 33)	0	mbo
171				Top mbo
172				mbo
173	=====			

```

174 Total params: 26,285,486
175 Trainable params: 26,285,486
176 Non-trainable params: 0
177

```

2 phần xử lý trên chính là những xử lý mấu chốt của thuật toán mà chúng ta cần nắm được. Phần khởi tạo các data_generator và huấn luyện mô hình khá đơn giản các bạn có thể tham khảo code gốc tại SSD_keras - git repository (https://github.com/pierluigiferrari/ssd_keras), rất chi tiết. Khi đưa vào 1 hình ảnh, thuật toán sẽ trả về kết quả bao gồm các khung hình bao quan vật thể kèm theo nhãn và xác suất của lớp mà vật thể bao trong khung hình có thể thuộc về nhất. Thuật toán có thể dự báo nhiều vật thể có kích thước to nhỏ khác nhau.



4. Tổng kết

Như vậy qua bài viết này tôi đã trình bày cho bạn đọc tổng quát kiến trúc và cách thức hoạt động của thuật toán SSD. Đây là một trong những thuật toán có độ chính xác cao và tốc độ xử lý tương đối nhanh. Tôi xin tổng kết lại một số ý chính:

- Kiến trúc của mô hình SSD bao gồm một base network là một mạng deep CNN được lược bỏ các layers fully connected ở giai đoạn đầu nhằm trích lọc các features.
- Các bộ lọc tích chập kích thước (3 x 3) được áp dụng trên các features map ở những layers tiếp theo nhằm làm giảm kích thước của ảnh. Từ đó giúp nhận diện được các hình ảnh ở nhiều kích thước khác nhau.
- Trên mỗi một cell của feature map ta tạo ra một tập hợp các default bounding box có scale và aspect ratio khác nhau. Tọa độ của các default bounding box dựa sử dụng để dự báo offsets của khung hình bao quan vật thể.

Hi vọng rằng chúng ta có thể nắm vững được thuật toán và tự xây dựng cho mình một mạng SSD để nhận diện vật thể.

Cuối cùng không thể thiếu là các tài liệu mà tôi đã tham khảo để xây dựng bài viết này. Top

5. Tài liệu.

1. SSD: Single Shot MultiBox Detector - Wei liu và cộng sự (<https://arxiv.org/abs/1512.02325>)
2. SSD: object detection single shot multibox detector for real time processing - jonathan hui (https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06)
3. SSD_keras github repository (https://github.com/pierluigiferrari/ssd_keras)
4. SSD caffe github repository - Weiliu (<https://github.com/weiliu89/caffe/tree/ssd>)
5. Bài 12 - Các thuật toán Object Detection (<https://phamdinhhkhanh.github.io/2019/09/29/OverviewObjectDetection.html>)

Top