YOLO - एक ऑब्जेक्ट डिटेक्शन एल्गोरिथम

YOLO-an object detection algorithm

अंजलि पाठक, भावना वोहरा**,** कपिल गुप्ता

anjali_51710058@nitkkr.ac.in,vohrab1896@gmail.com,kapil@nitkkr.ac.in

Anjali Pathak, Bhawna Vohra, Kapil Gupta

Abstract

In this paper, an approach to object detection known as YOLO is presented. It is extremely fast. We use this algorithm to detect multiple objects in an image. The base YOLO model processes image in real-time at 45 frames per second. YOLO outperforms other detection methods including R-CNN as it is more generalized. It works on various types of datasets including artworks.

Keyword: YOLO, object detection, RCNN

सार- इस paper में, YOLO के रूप में obtained object detection के लिए एक approach present किया गया है। यह बेहद तेज है। हम एक image में कई objects का पता लगाने के लिए इस algorithm का use करते हैं। Base YOLO model 45 फ्रेम प्रति सेकंड की real-time में image को process करता है। यह अधिक generalized है इस प्रकार यह RCNN सहित अन्य detection models को बेहतर बनाता है। यह artworks सहित विभिन्न प्रकार के dataset पर काम करता है।

कीवर्ड: YOLO, object detection, RCNN

1. परिचय

मनुष्य के पास तेज visual system हैं। वे एक image देखते हैं और तुरंत उसमें objects के बारे में जानते हैं। तो, हमें इस level से मेल खाने के लिए अत्यधिक fast और accurate algorithm की आवश्यकता है। deep learning computer vision अब self-driving कारों को यह पता लगाने में मदद कर रहा है कि दूसरी कारें और पैदल यात्री उनसे बचने के लिए कहां हैं। यदि हम आपके cell phone को देखते हैं, तो कई apps हैं, जो भोजन की तस्वीरें, या किसी होटल की तस्वीरें, या scenery की मज़ेदार तस्वीरें दिखाते हैं।और उन apps को बनाने वाली कुछ कंपनियां deep learning के लिए सबसे attractive, सबसे beautiful, या सबसे relevant pictures का उपयोग कर रही हैं। Deep learning नए प्रकार की art को बनाने में सक्षम है। सबसे पहले, computer vision में तेजी से advances brand new applications को देखने के लिए enable कर रही है,

Issue December 2019, Vol. 31

ISSN: 2229-6913(Print), ISSN: 2320-0332(Online)

Article Received: 25 January 2019 Revised: 28 March 2019 Accepted: 24 June 2019 Publication: 24 Septemeber 2019

हालांकि वे कुछ साल पहले असंभव थे। computer vision समस्या का एक अन्य उदाहरण object detection है। इसलिए, यदि हम self-driving कार का निर्माण कर रहे हैं, तो शायद हमें यह पता लगाने की आवश्यकता नहीं है कि इस image में अन्य कारें हैं। लेकिन इसके बजाय, हमें इस image में अन्य cars की position का पता लगाने की आवश्यकता है, ताकि हमारी car उनसे बच सके। object detection में, आमतौर पर, हमें न केवल object के type का पता लगाना होता है, बल्कि उनके चारों ओर box भी बनाना होता है। हमारे पास यह पहचानने का कोई दूसरा तरीका है कि image इन objects के हैं। एक ही image में कई cars हो सकती हैं, या आपकी कार की निश्चित distance के near कम से कम हर एक। computer vision problems की चुनौतियों में से एक यह है कि input reality में बड़े हो सकते हैं। इसके अलावा, computational requirements और training के लिए memory requirements तीन अरब parameters वाला एक neural network बस थोड़ा सा infeasible है।

लेकिन computer vision applications के लिए, हम केवल छोटी छोटी images का उपयोग करके अटकना नहीं चाहते हैं। आप बड़ी images का उपयोग करना चाहते हैं। ऐसा करने के लिए, आपको convolution operation को बेहतर ढंग से लागू करने की आवश्यकता है, जो कि convolutional neural network के मूलभूत building blocks में से एक है। हमारे system का use करते हुए, आप केवल एक बार (YOLO) को एक image पर implement करते हैं, ताकि उसमें मौजूद objects की position का prediction लगाया जा सके। यह convolutional network एक साथ कई bounding box की prediction करता है। YOLO algorithm पर जाने से पहले, आइए पहले एक और algorithm पर चर्चा करें जो computational रूप से महंगा है।

2. Literature Review

2.1 Sliding Window Algorithm

1. Actual image size की तुलना में आकार की एक window को बहुत छोटा करें। इसे crop और इसे ConvNet (CNN) को पास करें और ConvNet को prediction करें।
2. Window को slide रहें और cropped images को ConvNet में पास करें।
3. इस window आकार के साथ image के सभी भागों को crop करने के बाद, सभी चरणों को फिर से थोड़ा बड़ा window आकार के लिए दोहराएं। फिर से फसली छवियों को ConvNet में पास करें और इसे भविष्यवाणियां करने दें।
4. अंत में, आपके पास cropped regions का एक set होगा, जिसमें object के class और bounding box के साथ कुछ object होगा।

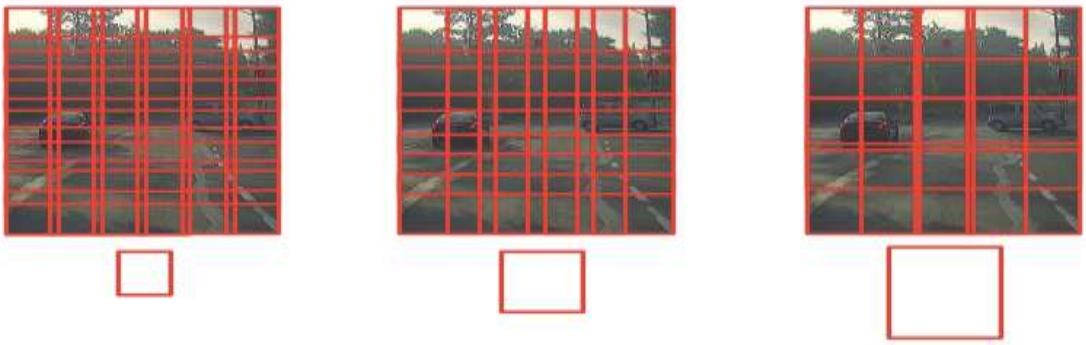


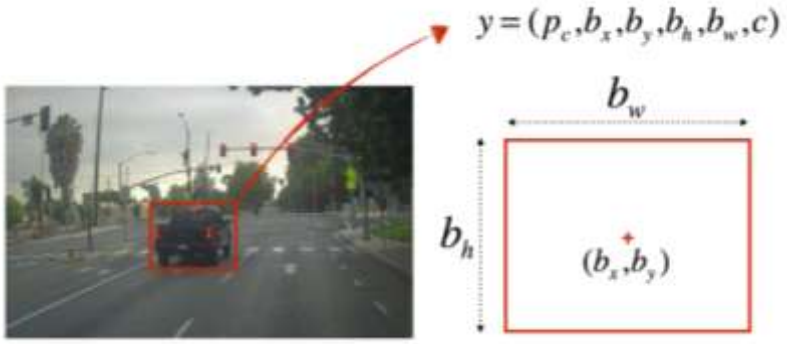
Fig 1

2.1.1. कमियां

1. Computational रूप से महंगा: कई images को crop करना और इसे ConvNet के माध्यम से pass करना computational रूप से बहुत महंगा होने जा रहा है।

2. गलत bounding box: हम पूरी image में square shape की window को खिसका रहे हैं, हो सकता है कि object rectangular हो या हो सकता है कि कोई भी square पूरी तरह से object के actual size से मेल न खाता हो। हालांकि इस algorithm में एक image में कई objects को find और localiz करने की ability है, लेकिन bounding box की accuracy अभी भी खराब है।

object detection में पहला कदम कुछ images को इकट्ठा करना और इसमें objects के चारों ओर bounding box खींचना और फिर उन images का उपयोग करके model को train करना है। एक उदाहरण के रूप में, हम self-driving कारों के मामले पर विचार कर सकते हैं। हम इन images को कार के सामने एक camera रखकर इकट्ठा कर सकते हैं। एक image में कई bounding boxes हो सकते हैं। एक bounding box y = (pc, bx, by, bh, c) जहां pc एक वस्तु का विश्वास है जो bounding box में मौजूद है, bx और by box के center के coordinates हैं, bh और bw हैं बॉक्स की height और width और c 1 से लेकर 80 तक है और यह पता की गई वस्तु के instance के range को दर्शाता है।YOLO algorithm convolutional neural network model पर आधारित है।



Issue December 2019, Vol. 31
ISSN: 2229-6913(Print), ISSN: 2320-0332(Online)
Article Received: 25 January 2019 Revised: 28 March 2019 Accepted: 24 June 2019 Publication: 24 Septemeber 2019

Fig 2

2.2. Model Details

1. Input आकार की images का एक batch है (m, 608,608,3) जहां m images की संख्या को दर्शाता है और 608 * 608 * 3 हर image के dimensions हैं।

2. Output label class नामों के साथ bounding box की एक list है। ऊपर बताए अनुसार प्रत्येक box को 6 factors द्वारा दर्शाया गया है।

Sliding window method की तरह, प्रत्येक image को grid cells में divide किया गया है। यदि object का center grid cell में आता है, तो grid cell उस object का पता लगाने के लिए जिम्मेदार होता है। प्रत्येक cell में 5 anchor boxes होते हैं। इस प्रकार, YOLO architecture इस प्रकार है:

IMAGE(m,608,608,3) → DEEP CNN → ENCODING(m,19,19,5,85).

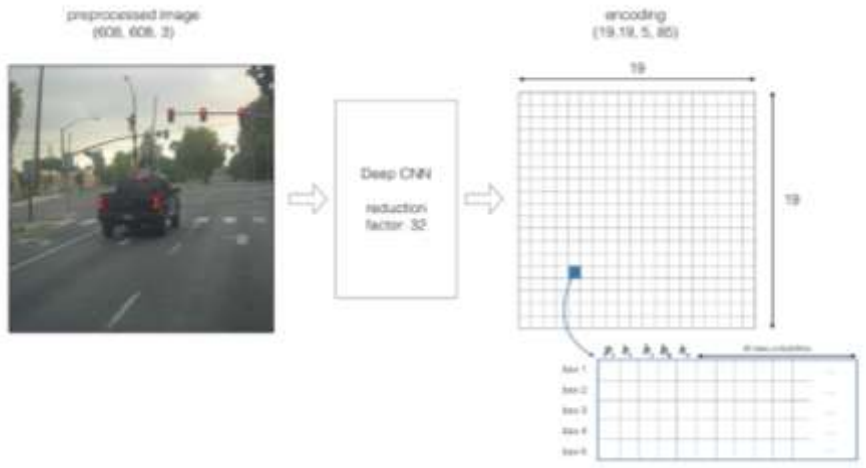


Fig 3

जहां प्रत्येक output image 19 * 19 * 5 * 85 के size की होती है जहां 5 नहीं है। लंगर बक्से और 85 सभी features को दर्शाता है। हम output image (19,19,5,85) को (19,19,425) तक flatten कर सकते हैं।अब, प्रत्येक cell के प्रत्येक box के लिए हम elementwise product ढूंढेंगे और इस probability को निकालेंगे कि box में एक निश्चित class है।हम विभिन्न रंगों में इन bounding boxes की plotting रचकर YOLO के output की कल्पना कर सकते हैं। ऐसा करने से हम कई bounding box के साथ समाप्त हो जाएंगे। इस प्रकार, संख्या को कम करने के लिए हमें algorithm के output को filter करना होगा। दो चरण किए जाते हैं-

1. किसी object के कम probability वाले box निकालें।

2. बहुत सारे boxes में से एक box का चयन करें जो एक दूसरे के साथ overlap होता है और एक ही object का पता लगाता है। Threshold value का उपयोग करके filter boxes।

इस प्रक्रिया में, हम thresholding द्वारा अपना पहला filter लागू करेंगे। Threshold value से कम class score मूल्य वाला कोई भी box हटा दिया जाता है।

हमारे पास कुल 19 * 19 * 5 * 85 number हैं, जहां प्रत्येक box में 85 नंबर का वर्णन किया गया है। तो, हम निम्नलिखित variables में (19,19,5,85) या (19,19,425) Tensor को फिर से rearrange करेंगे:

1. **box_confidence**: यह 19 * 19 grid cells के 5 boxes में से प्रत्येक के लिए pc (confidence probability है कि कुछ वस्तु है) आकार (19 * 19,5,1) का tensor है।

2.boxes: यह एक tensor आकार (19 * 19,5,4) युक्त है (bx, by, bh, bw) प्रत्येक 5 box प्रति grid cell के लिए।

3. box_class_probs: यह (19 * 19,5,80) आकार का एक tensor class probabilities (c1, c2, c3,… .c80) है जिसमें प्रत्येक grid cell के 5 boxes में से प्रत्येक के लिए 80 classes के लिए है।

Box_confidence और box_class_probs के elementwise product की गणना करके box score की गणना करें। Threshold का उपयोग करके एक mask बनाएं। यह mask उन boxes के लिए सही होना चाहिए जिन्हें आप रखना चाहते हैं।

जैसे: ([0.9,0.3.0.4,0.5,0.1] <0.4) returns [झूठा, सच्चा, झूठा, झूठा, सच्चा]।

TensorFlow का उपयोग करते हुए, इस mask को box_class_scores, boxes और box_classes पर लागू करके उन boxes को filter करें जिन्हें आप नहीं चाहते हैं।

Implementation निम्नानुसार किया जाता है:

```
# GRADED FUNCTION: yolo_filter_boxes

def yolo_filter_boxes(box_confidence, boxes, box_class_probs,
threshold = .6):
    """Filters YOLO boxes by thresholding on object and class
confidence.

    Arguments:
    box_confidence -- tensor of shape (19, 19, 5, 1)
    boxes -- tensor of shape (19, 19, 5, 4)
    box_class_probs -- tensor of shape (19, 19, 5, 80)
```


Issue December 2019, Vol. 31
ISSN: 2229-6913(Print), ISSN: 2320-0332(Online)
Article Received: 25 January 2019 Revised: 28 March 2019 Accepted: 24 June 2019 Publication: 24 Septemeber 2019

```
    threshold -- real value, if [ highest class probability score < threshold], then get rid of the corresponding box

    Returns:
    scores -- tensor of shape (None,), containing the class probability score for selected boxes
    boxes -- tensor of shape (None, 4), containing (b_x, b_y, b_h, b_w) coordinates of selected boxes
    classes -- tensor of shape (None,), containing the index of the class detected by the selected boxes

    Note: "None" is here because you don't know the exact number of selected boxes, as it depends on the threshold.
    For example, the actual output size of scores would be (10,) if there are 10 boxes.
    """
# Step 1: Compute box scores
    ### START CODE HERE ### (≈ 1 line)
    box_scores = np.multiply(box_confidence, box_class_probs)
    ### END CODE HERE ###

    # Step 2: Find the box_classes thanks to the max box_scores, keep track of the corresponding score
    ### START CODE HERE ### (≈ 2 lines)
    box_classes = K.argmax(box_scores, axis=-1)
    box_class_scores = K.max(box_scores, axis=-1)
    ### END CODE HERE ###

    # Step 3: Create a filtering mask based on "box_class_scores" by using "threshold". The mask should have the
    # same dimension as box_class_scores, and be True for the boxes you want to keep (with probability >= threshold)
    ### START CODE HERE ### (≈ 1 line)
    filtering_mask = K.greater_equal(box_class_scores, threshold)
    ### END CODE HERE ###
```

```
    # Step 4: Apply the mask to scores, boxes and classes
    ### START CODE HERE ### (≈ 3 lines)
scores = tf.boolean_mask(box_class_scores, filtering_mask)
    boxes = tf.boolean_mask(boxes, filtering_mask)
    classes = tf.boolean_mask(box_classes, filtering_mask)
    ### END CODE HERE ###

    return scores, boxes, classes
```

अब, पहले filter को लागू करने के बाद non-max suppression का उपयोग करके किया जाता है।

2.3 Non-max suppression

Threshold द्वारा filter करने के बाद भी, हम अभी भी बहुत सारे overlapping box के साथ समाप्त हो जाएंगे। सही boxes के selection के लिए एक और filter को non-max suppression (NMS) कहा जाता है।

Fig 4

इस concept में 3 steps शामिल हैं।

1. उच्चतम Score वाले box का selection करें।

2. अन्य सभी box के साथ इसका overlap ढूंढें और उन लोगों को हटा दें जिनके पास iou_threshold से अधिक overlap है।

3. इन steps को तब तक दोहराएं जब तक कोई और box न बचे।

इस process में एक function शामिल है जिसे "Intersection over Union" या IoU कहा जाता है।

यदि हमारे पास दो bounding boxes A1 और A2 हैं, तो IoU A1 और A2 के intersecting area के A1 और A2 के union area का अनुपात देता है।

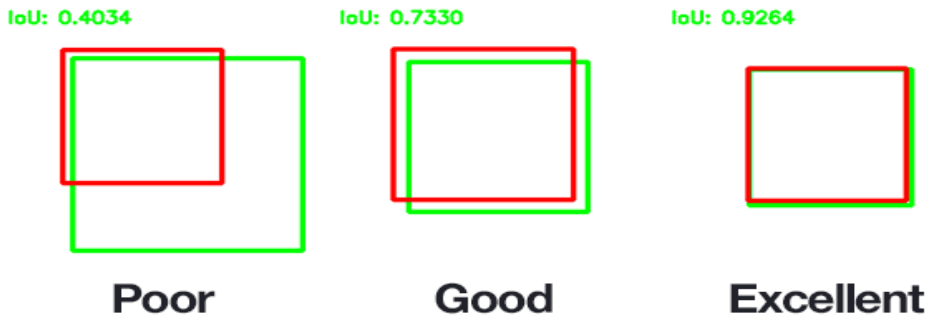


Fig 4

एक bounding box दो corner points द्वारा define किया गया है - ऊपरी बाएं और निचले दाएं (x1, y1, x2, y2)। Intersection area को खोजने के लिए हम इसकी चौड़ाई (x2-x1) द्वारा ऊंचाई (y2-y1) को multiply करते हैं।

जहां x1 दो boxes का अधिकतम निर्देशांक है

y1 दो boxes के y1 निर्देशांक की अधिकतम है

x2 दो boxes के x2 निर्देशांक में से न्यूनतम है

y2 दो boxes के y2 निर्देशांक में से न्यूनतम है

Union area की गणना area (A1) + area(A2) –Intersection area द्वारा की जा सकती है।

```
# GRADED FUNCTION: iou
def iou(box1, box2):
    """Implement the intersection over union (IoU) between box1 and box2
    Arguments:
    box1 -- first box, list object with coordinates (x1, y1, x2, y2)
    box2 -- second box, list object with coordinates (x1, y1, x2, y2)
    """
    # Calculate the (y1, x1, y2, x2) coordinates of the intersection of box1 and box2. Calculate its Area.
    ### START CODE HERE ### (≈ 5 lines)
    xi1 = max(box1[0], box2[0])
    yi1 = max(box1[1], box2[1])
    xi2 = min(box1[2], box2[2])
```


Issue December 2019, Vol. 31
ISSN: 2229-6913(Print), ISSN: 2320-0332(Online)
Article Received: 25 January 2019 Revised: 28 March 2019 Accepted: 24 June 2019 Publication: 24 Septemeber 2019

```
    yi2 = min(box1[3], box2[3])
    inter_area = (xi2 - xi1)*(yi2 - yi1)
    ### END CODE HERE ###

    # Calculate the Union area by using Formula: Union(A,B) = A + B
- Inter(A,B)
    ### START CODE HERE ### (≈ 3 lines)
    box1_area = (box1[3] - box1[1])*(box1[2]- box1[0])
    box2_area = (box2[3] - box2[1])*(box2[2]- box2[0])
    union_area = (box1_area + box2_area) - inter_area
    ### END CODE HERE ###

    # compute the IoU
    ### START CODE HERE ### (≈ 1 line)
    iou = inter_area / union_area
    ### END CODE HERE ###
    return iou


# GRADED FUNCTION: yolo_non_max_suppression

def yolo_non_max_suppression(scores, boxes, classes, max_boxes = 10,
iou_threshold = 0.5):
    """
    Applies Non-max suppression (NMS) to set of boxes

    Arguments:
    scores -- tensor of shape (None,), output of yolo_filter_boxes()
    boxes -- tensor of shape (None, 4), output of
yolo_filter_boxes() that have been scaled to the image size (see
later)
    classes -- tensor of shape (None,), output of
yolo_filter_boxes()
    max_boxes -- integer, maximum number of predicted boxes you'd
like
```

```
    iou_threshold -- real value, "intersection over union" threshold
used for NMS filtering

    Returns:
    scores -- tensor of shape (, None), predicted score for each box
    boxes -- tensor of shape (4, None), predicted box coordinates
    classes -- tensor of shape (, None), predicted class for each
box
    Note: The "None" dimension of the output tensors has obviously
to be less than max_boxes. Note also that this
    function will transpose the shapes of scores, boxes, classes.
This is made for convenience.
    """

max_boxes_tensor = K.variable(max_boxes, dtype='int32')     # tensor
to be used in tf.image.non_max_suppression()

K.get_session().run(tf.variables_initializer([max_boxes_tensor])) #
initialize variable max_boxes_tensor

    # Use tf.image.non_max_suppression() to get the list of indices
corresponding to boxes you keep
    ### START CODE HERE ### (≈ 1 line)
    nms_indices = tf.image.non_max_suppression(boxes, scores,
max_boxes_tensor, iou_threshold=iou_threshold)

    ### END CODE HERE ###

    # Use K.gather() to select only nms_indices from scores, boxes
and classes
    ### START CODE HERE ### (≈ 3 lines)
    scores = K.gather(scores, nms_indices)
    boxes = K.gather(boxes, nms_indices)
    classes = K.gather(classes, nms_indices)
    ### END CODE HERE ###
```

```
    return scores, boxes, classes
```

अब, ये filter गहरे CNN के हर output पर लागू होते हैं।

3. परिक्षण

80 squares और 5 anchor boxes की जानकारी 2 फ़ाइलों में stored है। हम इन राशियों को model में load करते हैं जैसा कि निम्नलिखित code snippet में दिखाया गया है।

```
class_names = read_classes("model_data/coco_classes.txt")
anchors = read_anchors("model_data/yolo_anchors.txt")
image_shape = (720., 1280.)
```

Loading a pretrained model

एक yolo model को train करने में बहुत समय लगता है और इसके लिए labelling bounding box के एक बड़े dataset की आवश्यकता होती है। इसके कारण, हम "Yolo.h5" में संग्रहीत एक पूर्व-मौजूदा Keras Yolo model को load करते हैं। यह निम्नलिखित code snippet द्वारा किया जाता है।

```
yolo_model=load_model("model_data/yolo.h5")
```

Convert output of the model to usable bounding box tensors

The output of the model is a tensor which we will use in all processings. The following code is used for that.

```
yolo_outputs=yolo_head(yolo_model.output, anchors, len(class_names))
```

3.1 Filtering boxes

Yolo_outputs ने yolo_model के सभी estimated box दिए। अब हम filtering कर सकते हैं और best box का चयन कर सकते हैं।

यह filtering पहले लागू किए गए filtering function को call करके किया जाता है। यह निम्नलिखित तरीके से किया जाता है।

score, boxes, classes = yolo_eval (yolo_outputs, image_shape)

3.2 चित्र पर ग्राफ़ चलाएँ

हमने एक देखा या ग्राफ बनाया है जो निम्नलिखित कार्य करता है: -

1. छवि yolo_model को इनपुट के रूप में दी गई है। इस मॉडल का उपयोग output खोजने के लिए किया जाता है।
2. यह output yolo_head का उपयोग करके संसाधित किया जाता है और data को tensor में बचाता है
3. एक filtering function, yolo_eval का उपयोग bounding box की संख्या को कम करने के लिए किया जाता है।

सबसे पहले image को निम्न code snippet का उपयोग करके संसाधित किया जाता है जो output:

1. **image**: drawing box के लिए आपकी image का एक पीआईएल प्रतिनिधित्व।
2. **image_data**: image का representation करने वाला एक numerical array।

image, image_data = preprocess_image("images/" + image_file, model_image_size = (608, 608))

निम्नलिखित prediction समारोह है जो YOLO model का उपयोग करके image का testing करने के लिए उपयोग किया जाता है।

def predict(sess, image_file):

"""

Runs the graph stored in "sess" to predict boxes for "image_file". Prints and plots the preditions.

Arguments:

sess -- your tensorflow/Keras session containing the YOLO graph

image_file -- name of an image stored in the "images" folder.

Returns:

out_scores -- tensor of shape (None,), scores of the predicted boxes

out_boxes -- tensor of shape (None, 4), coordinates of the predicted boxes

out_classes -- tensor of shape (None,), class index of the predicted boxes

Issue December 2019, Vol. 31
ISSN: 2229-6913(Print), ISSN: 2320-0332(Online)
Article Received: 25 January 2019 Revised: 28 March 2019 Accepted: 24 June 2019 Publication: 24 Septemeber 2019

```
    Note: "None" actually represents the number of predicted boxes, it varies between 0 and
max_boxes.
    """

    # Preprocess your image
    image, image_data = preprocess_image("images/" + image_file, model_image_size = (608,
608))

    # Run the session with the correct tensors and choose the correct placeholders in the
feed_dict.
    # You'll need to use feed_dict={yolo_model.input: ... , K.learning_phase(): 0})
    ### START CODE HERE ### (≈ 1 line)
    out_scores, out_boxes, out_classes = sess.run([scores, boxes,
classes], feed_dict={yolo_model.input: image_data,
K.learning_phase(): 0})
    ### END CODE HERE ###

    # Print predictions info
    print('Found {} boxes for {}'.format(len(out_boxes), image_file))
    # Generate colors for drawing bounding boxes.
    colors = generate_colors(class_names)
    # Draw bounding boxes on the image file
    draw_boxes(image, out_scores, out_boxes, out_classes,
class_names, colors)
    # Save the predicted bounding box on the image
    image.save(os.path.join("out", image_file), quality=90)
    # Display the results in the notebook
    output_image = scipy.misc.imread(os.path.join("out",
image_file))
    imshow(output_image)

    return out_scores, out_boxes, out_classes
```

4. References

1. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).

2. Redmon, J., & Farhadi, A. (2017). YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 7263-7271).

3. Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *CVPR (1)*, *1*(511-518), 3.

4. Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 580-587).