Computer Vision

Linear Algebra Applications in Object Detection for Autonomous Vehicles

Sathvik Koneru, Aakash Balaji, Jonathan Yun, Raghav Jain

Introduction

What is Computer Vision?

Computer vision is a field of computer science that works on enabling computers to see, identify, and process images in the same way that human vision does, and then provide appropriate output.

Why is Computer Vision Important?

Computer vision gives machines the capability to receive and analyze visual data on its own, and use this information to make effective predictions.

It allows any computer-controlled machine/vehicle, from robots to drones to autonomous vehicles, to improve efficiency, intelligence, and safety.

Computer vision's dominance in our society has become ever prevalent due to the increasing presence of digital media. Teenagers' favorite Snapchat filters and Animoji's are all reliant on computer vision, further cementing its place in our day to day lives.

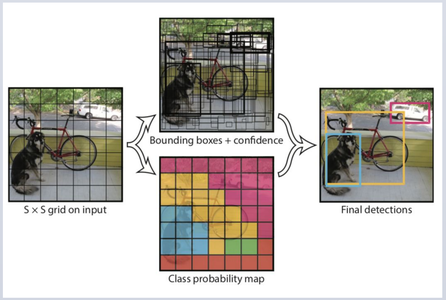
How Does Computer Vision Relate to Linear Algebra?

For the purposes of our project, computer vision involves the usage of matrices to store pre-processed image data as RGB values. The image is then taken in and then run through an algorithm, in our case the YOLO algorithm, which involves the usage of convolution. During this process the images, in the form of matrices, are scanned for objects and output vectors are generated that tell us where the object is located in the image and with what probability.

Computer vision in autonomous vehicles also involves the usage of the linear algebra concept of convolution to improve speed and efficiency. When analyzing different subsections of the image for objects, convolution allows the algorithm to reuse many already-computed values decreasing the number of executions in a computer vision algorithm by millions in some cases.

Objective

The objective of our project was to learn more about how computer vision is utilized in autonomous vehicles. More specifically, our goal was to take a deeper look at the YOLO algorithm and analyze the effects of changing parameters on the output from a given data set. Due to time constraints, we focused only on the threshold parameter which is the minimum probability needed in order for the algorithm to identify an object.



References

YOLO Paper: https://arxiv.org/pdf/1506.02640.pdf

https://www.techopedia.com/definition/32309/computer-vision

https://github.com/enggen/Deep-Learning-Coursera/blob/master/Convolutional%20Neural%20Networks/Week3/Car%20detection%20for%20Autonomous%20Driving/Autonomous%20driving%20application%20-%20Car%20detection%20-%20v1.ipynb

https://en.wikipedia.org/wiki/Computer_vision

https://www.coursera.org/learn/convolutional-neural-networks?specialization=deep-learning

https://skymind.ai/wiki/autonomous-vehicle

YOLO Algorithm

The YOLO (You Only Look Once) algorithm is popular for automated object detection in real-time due to its high accuracy as well as its efficient processing speed. The algorithm is called YOLO because it only requires one forward propagation pass through the integrated neural network to make predictions for any arbitrary image. The implementation steps for this algorithm are detailed below.

Model Details

Input Data Formatting

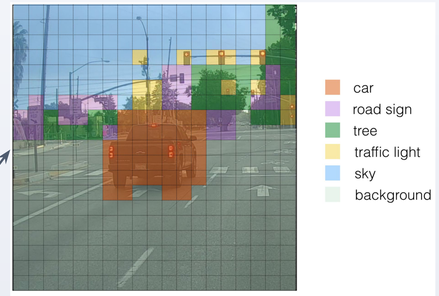
The input for the algorithm is a batch dataset containing images of shapes. Each preprocessed image contains its dimensions (width and height) in pixels as well as its RGB values at each pixel. These images are then passed into the integrated pre-trained neural network.

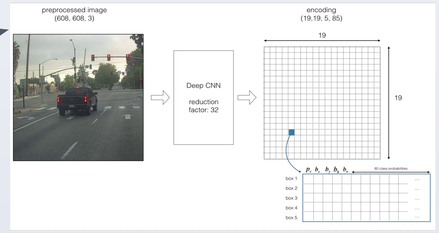
Pre-Trained Neural Network

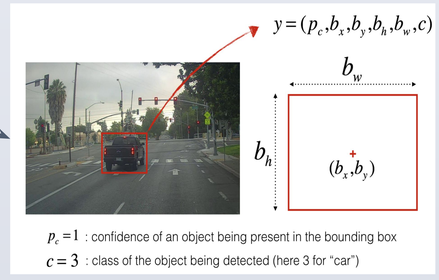
The neural network takes in images in the format specified above and transforms them into encoded matrices given a predetermined reduction factor and anchor box count. The reduction factor scales down the dimensions of the original image, and the anchor box count determines how many bounding boxes each cell of the encoded matrix represents (bounding boxes explained below). Anchor boxes are utilized to help correct miscalculations caused by objects overlapping across multiple cells of the encoded matrix.

Bounding Boxes

Bounding boxes are generated around notable objects, such as cars, traffic lights, trees, etc. Each bounding box is represented as 6 numbers: p_c, b_x, b_y, b_h, b_w, and c. The value p_c represents the confidence probability of an object being present in the bounding box. The values b_x and b_y represent the x-y coordinates of the midpoint of the box, and that midpoint is used to predict what kind of object is captured by the bounding box. The values b_h and b_w represent the height and width of the bounding box, respectively. Lastly, the value c represents the class of the object being detected (varies from object to object).





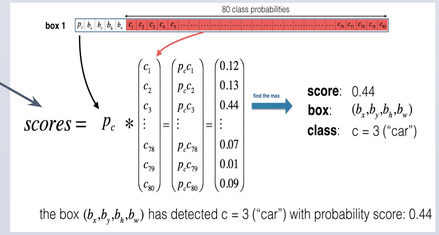


Filtering with a Threshold on Class Scores

The first filter that the YOLO algorithm uses is a score threshold, where only boxes that meet a minimum probability will remain.

When we run the YOLO algorithm, it passes over the each grid cell, and and returns vectors corresponding to its bounding box predictions. To calculate the scores for each bounding box vector, we multiply p_c, the probability there is some object, by each one of the class probabilities outputted. From there, we take the max score and predict the box corresponds to the max score object.

By using a user-specified threshold, we can be more confident about our bounding boxes but may tradeoff more objects being detected.



Non-Max Suppression

The YOLO algorithm uses a second filter for selecting the right boxes as you still end up with a lot of overlapping boxes after applying the first filter. In this step, first the box with the highest score is selected, then its overlap with other boxes is calculated. Finally, the overlapping areas are removed through an important function called "Intersection Over Union." This process is repeated until there's no more boxes with a lower score than the current selected box.

Testing and Prediction Results

To play around with the YOLO algorithm, we decided to look at how changing threshold values would affect the predictions. As we suspected, by increasing the default score threshold from 0.6, our results did not include objects (vehicles) that were located in sparse areas, objects that were at odd angles, small in size or objects that overlapped with each other. Based on our tests, the higher the threshold was, the more confident the prediction results were, but we lost valid predictions that seemed less confident to the algorithm.

Testing With Different Thresholds

```
def yolo_filter_boxes(box_confidence, boxes, box_class_probs, threshold):
    """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)
    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 boxe
    boxes -- tensor of shape (None, 4), containing (b_x, b_y, b_h, b_w) coordinates of selecte
    classes -- tensor of shape (None,), containing the index of the class detected by the selec
    """

    # Step 1: Compute box scores
    box_scores = np.multiply(box_confidence, box_class_probs)

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

    # 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)
    filtering_mask = K.greater_equal(box_class_scores, threshold)

    # Step 4: Apply the mask to scores, boxes and classes
    scores = tf.boolean_mask(box_class_scores, filtering_mask)
    boxes = tf.boolean_mask(boxes, filtering_mask)
    classes = tf.boolean_mask(box_classes, filtering_mask)

    return scores, boxes, classes
```

Default Threshold = 0.6

```
out_scores, out_boxes, out_classes = predict(sess, "test.jpg")

Found 7 boxes for test.jpg
car 0.60 (925, 285) (1045, 374)
car 0.66 (706, 279) (786, 350)
bus 0.67 (5, 266) (220, 407)
car 0.70 (947, 324) (1280, 705)
car 0.74 (159, 303) (346, 440)
car 0.80 (761, 282) (942, 412)
car 0.89 (367, 300) (745, 648)
```


Threshold = 0.7

```
In [48]: out_scores, out_boxes, out_classes = predict(sess, "test.jpg")

Found 3 boxes for test.jpg
car 0.74 (159, 303) (346, 440)
car 0.80 (761, 282) (942, 412)
car 0.89 (367, 300) (745, 648)
```


Threshold = 0.8

```
out_scores, out_boxes, out_classes = predict(sess, "test.jpg")

Found 1 boxes for test.jpg
car 0.89 (367, 300) (745, 648)
```


YOLO in the Real World