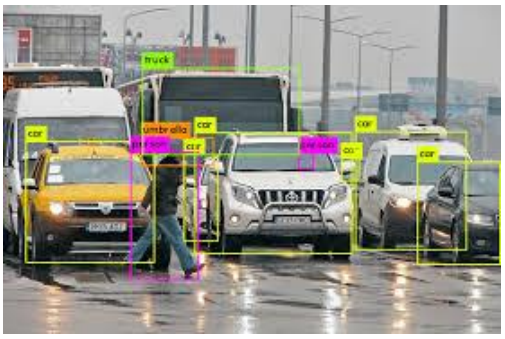
Object Detection using YOLOv3 Algorithm

Abstract

During this highly technological era, under artificial intelligence and machine learning, some of the most magnificent researches are conducted. Amongst those, robotics is one of the most significant technical development. Object detection mainly deals with identifying or detecting semantic objects and certain classes from a single image or video and in some condition also from a live stream. It has vast amount of applications not only in robotics but also in day to day life as well. Some major applications in the areas of object detection are security surveillance, self-driving automated vehicles, tracking objects, activity recognition, pedestrian detection, etc. and detecting more than one object from a given image is one of the most difficult tasks in unsupervised learning. On the other hand, Image classification algorithms deals with the identification of main or primary (single) object from an image only. Multiple times classification algorithm is applied to the image to dig out all the objects from an image. In this paper, we would like to propose an object detection model that is capable of identifying objects from the given image along with its accuracy.



Boundary boxes in object detection

Introduction

In computer vision, there are three types of tasks:

1. Object localization:

Object localization is the primary concept behind all image detection algorithms. Object localization refers to detection of objects and presenting them in different ways.

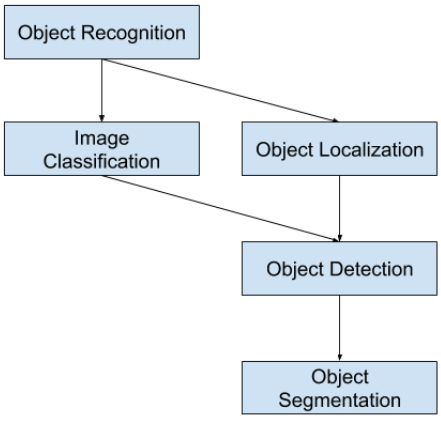
Instance segmentation is one of those ways of localization. In instance segmentation, a bounding box is created around the object, and a label to which that object belongs is created at the top of the object. Whereas in image segmentation, the boundary is created around the object and it's also pixel-wise. Instance segmentation creates square or rectangle boundaries.

2. Image Classification:

Image Classification concerns with type or class of the object in the given image. Output for this algorithm is a class label for input photographs.

3. Object Detection:

Object detection is a computer vision technique to identify various objects from an image or a video.
Object detection outputs class labels with proper boundary boxes for objects in the image.



Difference between Object Detection, Image Classification and Object Localization

Various algorithms can be used for Object Detection using Deep Learning some of them are:

1. **YOLO:**

YOLO uses convolutional neural networks for predict ion of class labels and object's location also.

YOLOv3 is more accurate and faster than previous versions and SSD (Single Shot MultiBox Detector).

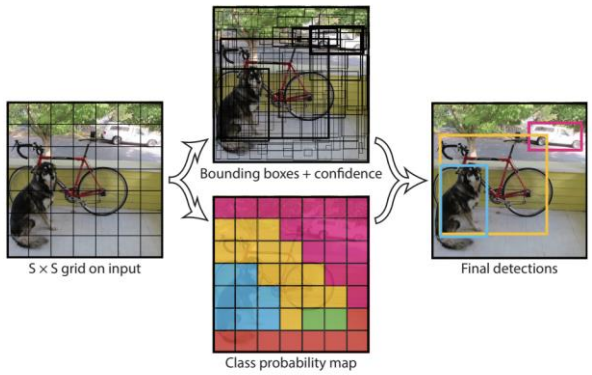
Here is the brief introduction on the previous versions of YOLO algorithm:

YOLOv1:

YOLOv1 uses the darknet framework and works the same as YOLOv3 works. But the main problem with yolov1 was cluster classification. It cannot recognize small objects which are closer to each other. The algorithm reads them as a cluster and identifies only some of them.

YOLOv2:

Yolov2 introduced many new features over version 1. Such as batch normalization, anchor boxes, multiscale training, and also new darknet architecture. It works better than YOLOv1 with good accuracy and can also detect small objects from an image.



Yolo object detection along with boundary boxes and confidence score

How does YOLO work?

YOLO looks at an image only once. And then applies only one neural network on the image. The image is divided into the SxS grid. Each cell can predict N bounding boxes. A rectangle is formed from these bounding boxes to enclose an object within it. So total, SxSxN boxes can be predicted from the given image. Each bounding boxes has its own predicted probabilities.

YOLO outputs a confidence score that tells us how certain it is that the predicted bounding box encloses the object. The task here is to categorize each object based on the class shown in the image or the video. The output for a given input image will be a confidence score that tells us how certain it is that the predicted bounding box encloses the object.

2. **SSD:**

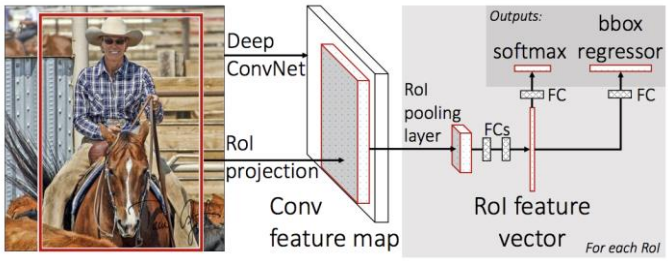
SSD (Single Shot MultiBox Detector) algorithms have a good balance between its accuracy and speed. SSD runs a small two-dimensional convolutional kernel on the feature map generated by CNN and image. By doing so, it automatically predicts bounding boxes and classification probability.

SSD works better on large objects than the smaller one. We can say that its performance is as good as Faster-RCNN for large objects.

3. **Faster R-CNN:**

Faster R-CNN consists of Region Proposal Network (RPN) and Fast-RCNN. Anchor boxes are introduced in faster-RCNN. There are three types of anchor boxes, 128x128, 256x256, and 512x512. Additionally, it has three aspect ratios. (1:1, 2:1, 1:2). This gives a total of 9 boxes to predict the probability. This is the output of RPN, which is given to Fast-RCNN. Here, the spatial pooling technique is applied along with regression.

Faster R-CNN is 10 times faster than Fast R-CNN for the same accuracy output.



Object detection with R-CNN

Dataset Description

For this project, we used the dataset provided by our professor, **MS COCO (Microsoft common objects in context) dataset.**

- The data is accomplished by gathering pictures of complex ordinary scenes containing basic items in their common setting.
- It is large-scale segmentation, captioning and object detection dataset. The task is to categorize each object based on the class shown in the image.
- The image feature consists of object segmentation. Also, there are more than 330 thousand images out of which more than 200 thousand images are labeled. There are 91 stuff categories, 80 object categories along with 1 and a half million object instances. Each image consists of at least 5 captions.
- The training set consists of 118 thousand images along with annotations - 2017 Train Images (18 GB), and 2017 Train/Val annotations (241 MB)

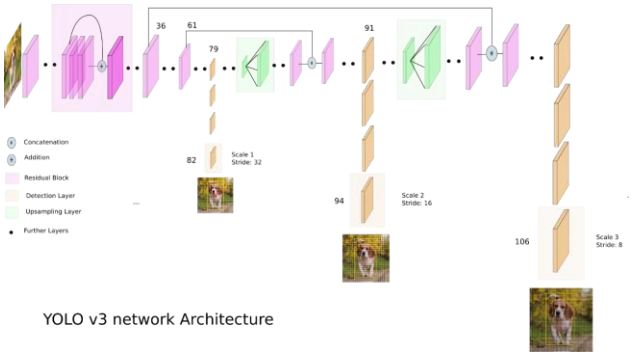
Project description

Description:

We used MSCOCO dataset as mentioned above for model training. We used the Keras-Tensorflow module to implement the YOLOv3 model. We used Keras because it is easy to implement and works better with computer vision models and it is capable of running on the top of Tensorflow.

YOLOv3:

Following are the primary concepts of the YOLOv3 algorithm:



Yolov3 Network Architecture

Darknet architecture:

YOLOv3 uses darknet – 53 architecture for feature extraction from an image. As the name suggests it has 53 convolutional layers. We know that the number of layers suggests how deep the architecture is. YOLOv3 is a way deeper architecture than yolov2, as it goes only to 19 layers. There are 3x3 and 1x1 filters in YOLOv3 architecture which helps to extract features from the image, in every different layer.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

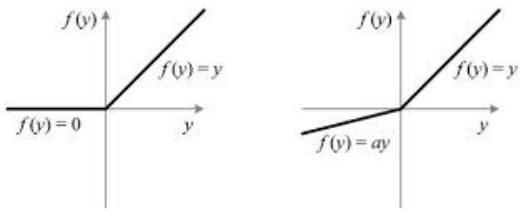
Darknet-53 architecture

Leaky ReLU:

We all know that ReLU (Rectified Linear Unit) is a good activation function for neural networks. But one step ahead, leaky ReLU is way better than ReLU itself. One of the most benefits given by leaky ReLU is, it almost removes the "dying ReLU" problem. Dying ReLU problem refers to the output given by ReLU function for negative inputs (It's always 0). In leaky ReLU, there is a small negative slope for negative input values. Such that, output for that input values will tend to zero but not actually zero, which helps in getting better neurons at each layer of the convolutional network and so in output results.

Here is a basic code to implement leaky ReLU in the program and a chart to represent its functionality. On the left-hand side is ReLU function, and the other chart postulates leaky ReLU. The function, f(y) = ay is the slope line for negative values, and in practice, the value of 'a' is nearly around 0.01 – 0.001.

```
def leaky_relu(alpha, x):
 if x&amp;amp;amp;amp;amp;amp;amp;lt;=0:
  return x
 else:
  return alpha * x
```



ReLU and leaky ReLU Function Graph

Bounding boxes:
Bounding boxes are used to differentiate identified objects from one another. Logistic regression is used to predict the confidence score of the box.

Anchor boxes:
In a single network layer, anchor boxes are responsible for classification and prediction. It uses k means clustering method for prediction.

Batch normalization:
Generally, we normalize our input data with activation functions or some other techniques to improve the performance of our model. If that is the case, then can we normalize our hidden layers to improve model efficiency? Batch normalization is used for normalizing input and hidden layers of the architecture for improvement. Conventionally, it improves accuracy by 2%, and also overfitting can be reduced to negligible.

Accuracy threshold:
In our model, we used the accuracy threshold value. Mostly what model does, is that it generates bounding boxes for the objects which are not clearly visible (which are not required to be identified) with lesser accuracy than other objects. After giving threshold accuracy value to the model, it will not consider objects which give less accuracy than the threshold, which outputs in better detection.

Configuration parameter:
Additionally, the model can predict more than one bounding box for one object which sometimes overlaps on each other. The configuration parameter is set as an overlap amount for boxes. We used different values for the parameter to check how those boxes are made for different values. If we consider its value as 1.0, then it will show all the bounding boxes possible around an object. And for 0.1, it will remove some required boxes from the account. So generally, 0.5 or 0.6 are used to get better and systematic output.

Main references used for your project:

1. From a blog "How to implement a YOLOv3 object detector from scratch in PyTorch" on medium.com, we get to know about overall architecture and other features of YOLOv3. Like, bounding boxes, improvements in YOLOv3 over YOLOv2, anchor boxes, loss function. [6]
2. From this blog, we get to know about how YOLOv3 works with darknet architecture. [7]
3. Working on Keras in object detection. [8]

Difference in APPROACH/METHOD between our project and the main projects of our references:

The references that we used are using mostly pre-trained models and weights of the dataset. Also, they were some complex models with lesser accuracy than what we got in our model. Some models were working with a lot of data, which were out of our resources. Some of them have used limited datasets, and some have used a lot of data. But in our project, MS COCO dataset is used for getting the accuracy. We implemented the configuration parameter and threshold accuracy and with their combination, we achieved the best accuracy.

For object detection, generally, leaky ReLU is used as an activation function and we are using that in the model. But for testing purposes, we once tried simple ReLU function but did not get desired results. Other than that, all the parameters like anchor boxes, bounding boxes, CNN layers are retained in our model.

Difference in ACCURACY/PERFORMANCE between our project and the main projects of our references:

We saw a lot of implementation codes and snippets we got some idea about implementation.

1. Configuration parameter
In our project, there is one function that sets the configuration parameter. What this parameter does is, it generalizes the bounding boxes which generate around one object only. For a single object, we get multiple boxes. This parameter chooses the best box from them which can get a label accordingly. In one of those references, one person did not set the value for the configuration parameter and gets the accuracy of around 70%. But after implementing this parameter our accuracy has risen by 8 to 10%.

2. Accuracy Threshold value
While using different threshold values for accuracy, we got different numbers of objects in each output. This is because, every time we change threshold value, objects with accuracy below threshold value will be dropped from final accuracy measurement and output. Due to this, the model will not get the same number of objects each time.

These are some results we got for the same image with different threshold parameters, but with the same configuration parameter value (0.5).

Accuracy Threshold value	Accuracy	Number of objects:

60% (0.6)	88.165%	7
40% (0.4)	83.94%	8
80% (0.8)	98.54%	4

List of our contributions in the project:

1. Data augmentation. (Augmented annotation files with dataset).
2. With the threshold and configuration parameters ideal combination we took out the best accuracy.
3. Trained the Keras model using YOLOv3 to generate new weights to detect the objects in the image.
4. Tried different activation functions to see the variation in the accuracy of the image detection.
5. Calculated an individual accuracy of each object in the image then took the average of accuracies of all the objects in that image to get a cumulative accuracy of all the objects in that image.
6. Increase in accuracy with respect to the original reference we used for our project.

Analysis

What did we do well?

At the starting phase of the project, we did not have any idea about Keras, TensorFlow, Neural Networks, Activation Functions, YOLO models. But now we have a better understanding of these terms and due to which we were able to create a basic model to classify some objects from a given image.

The references we used were not calculating accuracy for each individual object, separately. We thought to calculate that and to combine all the calculated accuracies and achieve a more general accuracy for a single image which helped us to get more general ideas about the overall performance of the algorithm.

What could we have done better?

At the starting phase of the project, we were thinking about implementing the project not only for image or single photo but also for real-time videos. Such as, detecting cars and people in the given video of city traffic. But because of less time, increasing difficulties to implement the project, and also due to limited amount of resources we were not able to implement this idea.

If we were able to implement this thing than it could be helpful in big areas of computer vision like a surveillance system, unmanned vehicle systems, robotics, etc.

What is left for future work?

For future work, we can work on getting better and better accuracy for just a single image. After that, we can think of our basic and main idea to implement it for real-time video, for tracking application. This application requires a lot of high computation computers to generate an effective algorithm and proper environment to deploy that system.

One more idea is that, making a shopping application that detects the object and gives the best price, link to buy it from, and reviews of that item.

Conclusion

Object detection is a key concept in robotics and computer vision areas researches. It needs a huge amount of attention to give superior results in better advancements of humankind. Although, there have been many researches conducted and many more are still going on in computer vision, they are not enough as compared to technological advancements of the 21st century. Object detection can be highly used in real-time applications like tracking applications, surveillance systems, pedestrian detection, and unmanned vehicle systems.

But what we have learned so far in this project, is that for object identification, YOLOv3 is a well-defined approach. It gives good accuracy even for small datasets, because of it having deep architecture and complex model network.

References

1. Lin, Maire, Michael, Belongie, Serge, Bourdev, … Piotr. (2015, February 21). Microsoft COCO: Common Objects in Context. Retrieved from https://arxiv.org/abs/1405.0312
2. Bansal, N. (2020, March 9). Object Detection using YoloV3 and OpenCV. Retrieved from https://towardsdatascience.com/object-detection-using-yolov3-and-opencv-19ee0792a420
3. Redmon, J. (n.d.). Retrieved from https://pjreddie.com/darknet/yolo/
4. Tianxiang, & Liu, I. (2019, September 30). Real-time Mobile Video Object Detection using Tensorflow. Retrieved from https://towardsdatascience.com/real-time-mobile-video-object-detection-using-tensorflow-a75fa0c5859d
5. Top stories about Yolo written in 2019. (n.d.). Retrieved from https://medium.com/tag/yolo/archive/2019
6. Kathuria, A. (2018, April 29). What's new in YOLO v3? Retrieved from https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b
7. Redmon, J. (n.d.). Retrieved from https://pjreddie.com/darknet/yolo/
8. Rieke, J. (2018, October 29). Object detection with neural networks. Retrieved from https://towardsdatascience.com/object-detection-with-neural-networks-a4e2c46b4491