Detection of Objects
in
Varied and Complex Images

Department of Computer Science & Engineering
School of Engineering and Applied Science
Bennett University, Greater Noida
Uttar Pradesh (India), 201310

Mentors
Ms. Surbhi Gupta
Ms. Shambhavi Mishra

Submitted by
Shruti Gupta
Sharon Andrea Gomez
Pratyush Roy
Akhil Chandail
Yash Chaudary

Detection of Objects in Varied and Complex Images

Shruti Gupta, Sharon Andrea Gomez, Pratyush, Akhil Chandail,Yash Chaudary,
Surbhi Gupta, Shambhavi Mishra

***Abstract*—The objective of the Object Detection track of the Open Images Challenge 2019 was to detect multiple objects within an image, and identify the class they belonged to. To achieve this, we have used two different models, YOLOv3 and RetinaNet, implemented on Keras, using Python3. The ultimate objective of this challenge is to speed up the research in scene understanding, which is identification of objects and the relationships between them, to gain semantic knowledge of the image context and contents. Both models have their own strengths, speed and accuracy, respectively. The models have been trained on a subset of the original dataset due to time and computational constraints.**

***Index Terms*— Computer Vision, Deep Learning, Image Processing, Object Detection, RetinaNet, YOLOv3**

I. Introduction

Object Detection is one of the most important domains of Deep Learning, with applications in several areas. Research in this domain is ongoing, with efforts being made to develop better and more efficient approaches and algorithms. Aptly termed Computer Vision, the objective is to enable computers to identify objects in an image or video as efficiently as humans do. This has been called *scene understanding*, that is to build a representation of the content of a picture, what the objects present in the image are, their locations, the relationships between them and their significant roles in the scene.

Google's open-source image dataset, *Open Images V5* (OID) is currently the largest existing image dataset, with approximately 9 million images. It contains image-level labels, their respective bounding boxes, object segmentation masks, and visual relationships. These images are chosen carefully to be very diverse and complex containing 7 objects per image on an average.

The Open Images Challenge 2019, hosted on Kaggle, focuses on three tracks- object detection, visual relationships, and image segmentation. The object detection track of this competition is mainly focused on identifying various objects present in the image, and localizing them, using tight bounding boxes. In addition to this, the class hierarchy had to be taken into account, and objects had to be identified as members of their superclasses as well as their own class, as shown in Figure 1.

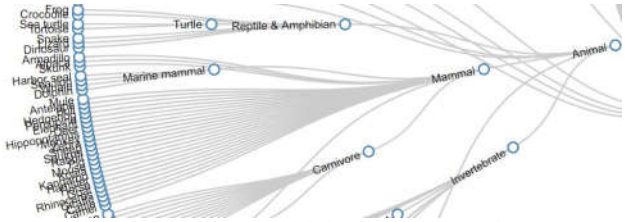

Fig. 1. A section of the tree hierarchy of classes, as given on the Open Images website [2]

This paper illustrates our proposed solutions using the *YOLOv3* and *RetinaNet* models for object identification, and then further processing the model output to identify the superclasses and additional labels. Other models that have been used for Object Detection purposes include Single Shot Detection (SSD), Region-based Convolutional Neural Networks (RCNN), Faster RCNN (FRCNN), Spatial Pyramid Pooling (SPP-Net) among others. While FRCNN and RetinaNet have given better accuracy, the time taken is high. SSD and YOLOv3 perform considerably better in terms of speed but give slightly less accurate results. Thus, it is a matter of achieving a favourable tradeoff of speed versus accuracy, which depends on the application domain.

II. Related Work

The progress in Computer Vision is mainly the result of datasets such as ImageNet, PASCAL VOC, CIFAR-10, COCO etc. that are readily available at public disposal. Various models were developed under the umbrella of object detection, each with its own distinct features and advantages.

OID aims to improve the state-of-the-art performance by providing the right combination of varied, complex images along with the rightly tailored annotations to fuel the generalizability of the machine learning models, unlike the previously used datasets. OID is unique when compared with its predecessors such as MS COCO, not only in being the largest dataset available, but in its annotation styling. The annotation style of the other datasets is that all classes are exhaustively covered in each image, whether present or not. In OID, instances of classes verified to not exist in the image are

Shruti Gupta is with Bhilai Institute of Technology, Durg, Chhattisgarh, India. (e-mail: shrutiguptarkc@gmail.com).

Sharon Andrea Gomez is with Mar Baselios College of Engineering and Technology, Kerala, India (e-mail: sharon.ag97@yahoo.com).

Pratyush Roy is with KIET Group of Institutions, Ghaziabad, Uttar Pradesh, India (e-mail: pratyushroy240898@gmail.com,).

Vanshaj Goel is with Group of Institutions, Ghaziabad, Uttar Pradesh, India (e-mail: vanshaj.goel.1998@gmail.com).

Yash Choudary is with Maharaja Agrasen Institute of Technology, New Delhi, India (e-mail: yash612@gmail.com).

not annotated. This sparse annotation approach helps in expanding the classes without having the annotations themselves be an exploding data, thus leaving the created model open to improvement.

Models like RetinaNet and FRCNN give better results in terms of accuracy (measured in mAP), as evaluated on the MS COCO dataset. SSD, R-FCN and YOLO give better speed, with some loss in accuracy. In real-time object detection applications, YOLO and SSD are preferred, as it is viable to compromise on accuracy, where the speed of detection is of primary importance. FRCNN, being a two-stage detector, takes more time than single-stage detectors such as YOLO and SSD, but offers much higher accuracy (mAP) than either of them. In applications where time is not of utmost importance, FRCNN performs better. RetinaNet takes longer to detect an object, but its accuracy is the highest of all mentioned models, as on the COCO dataset.

III. Methodology

A. Preprocessing of data

The Open Images V5 dataset for contains about 9 million images in all, which have been annotated for 600 classes of objects. These classes are organized in a semantic hierarchy, which was also provided. The images for the training, testing, and validation sets (approximately 561 GB) were downloaded, along with the respective annotations for the training and validation sets, which are in a CSV format, as specified on the Open Images website. The various attributes that we used are explained below-

1. *ImageID*: A unique identification for each image in the dataset
2. *LabelName*: The ID of the class that the object identified by the bounding box belongs to.
3. *Confidence:* The probability that the bounding box contains an object of the specified class, a value between 0 and 1.
4. *XMin, XMax, YMin, YMax*: The position and size of the bounding box is specified by its top-left and bottom-right vertices (XMin, YMin) and (XMax, YMax). Each of them has a value between 0 and 1.
5. *Class Name*: The name of the class to which the object identified by the bounding box belongs.

The input to the model is in the form of a .txt file, which contains ImageID and the corresponding PredictionString, with information about each bounding box identified for that image. Each row represents a single image.

Row format: *image_file_path box1 box2 ... boxN*

Box format: *x_min,y_min,x_max,y_max,class_id*

Additionally, the images, once they have been loaded, are resized and converted to an array of shape (416, 416, 3) i.e. 416 pixels by 416 pixels and 3 channels for the RGB values of each pixel. The original aspect ratio of the image remains unchanged by using padding where required. The absolute coordinates of the bounding boxes need to be adjusted accordingly, by converting the normalized coordinates such that they lie in the same relative position on the resized image as on the original.

RetinaNet takes information about the bounding box coordinates as input, along with the image (which we resized to 640x640) and the class of the object, denoted by class index. The coordinates need to be absolute coordinates for that image, rather than normalized.

Due to constraints on time and computational resources, we used a randomized subset of the original dataset for our project.

B. Model Architecture

We have used the YOLOv3 model, which was developed by Joseph Redmon and Ali Farhadi. Rather than using modified classifiers on sections of an image to detect objects in it, YOLOv3 applies a neural network on the entire image at once, hence the name, 'You Only Look Once'. Detection is performed using a 1x1 kernel, and a bounding box is determined for each cell of the feature map. The IOU (Intersection over Union) is calculated to measure the degree of overlap between two bounding boxes identifying the same object, and Non-Maximal Suppression is performed to eliminate all bounding boxes with a confidence score lower than the specified threshold value.

YOLOv3 is based on Darknet, and consists of 106 layers of a fully convolutional architecture, shown in Figure 2, with upsampling layers that permit it to perform detection at three different scaling levels. This increases its ability to detect smaller objects. It uses 9 anchor boxes per cell of the feature map, 3 at each scale. It also supports multilabel classification i.e. when an object belongs to multiple classes, as it does not softmax the final result, thereby eliminating single-class prediction. The loss function contains cross-entropy error terms, which means that logistic regression is used for class prediction and confidence calculation.

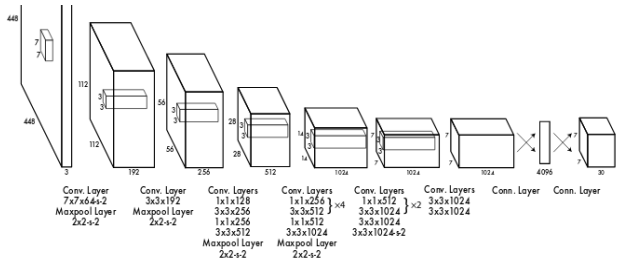

Fig.2. The architecture of the YOLOv3 Model, as developed by Joseph Redmon and Ali Farhadi. [5]

RetinaNet is a model that uses Resnet50/101 and Feature Pyramid Network (FPN) as its backbone structures. It has two task subnetworks, each for handling a specific task – classification, and bounding box regression.

The classification subnet predicts the probability of object presence at each spatial position, for each of the 9 anchors and N object classes. It takes an input feature map with 16x16 channels from a pyramid level, and the subnet applies 4 3x3 convolution layers, each with 256 filters and each followed by ReLU activations. Finally, sigmoid activations are attached to the outputs. Focal loss is applied as the loss function.

The box regression subnet is similar to the classification net, but the parameters are not shared. It outputs the object location, with respect to the anchor box, if an object

exists. Smooth_l1_loss with sigma equal to 3 is applied as the loss function to this part of the sub-network.

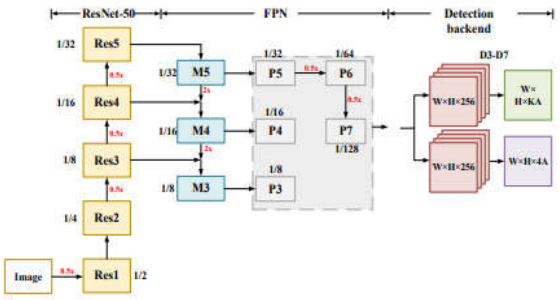

Fig. 3. Architecture of the Retina Net Model

IV. Experimental Results

A. Dataset

As explained in the previous section, the dataset used by us is a randomized subset of the original Open Images v5, approximately 95 GB in size, and consisting of 500,000 images. The image is loaded and preprocessed, which is then passed as input to the model. The desired output is the prediction string, which consists of the class identification, the confidence score, and the bounding box vertices.

In addition to this, the semantic hierarchy of the annotated class was given in a tree format, as a JSON file. This had to be restructured and converted into a .txt file containing the class label, class code, and the list of superclasses for each class label.

B. Implementation Details

The framework for our code comes from *qqwweee*'s GitHub repository *keras-yolo3*. We have used the same base model, and modified it as per our requirements. The original model was trained on the COCO dataset, and we have retrained it to use our custom dataset. The pretrained weights of YOLOv3, as given on the website, need to be converted from the Darknet model to a Keras model for implementation. The basic framework contained functionality for object detection, but the annotations did not contain multi-label classification. Hence, the semantic hierarchy had to be incorporated, to identify an object as a member of its superclasses as well as its own class.

The model was then trained on our custom dataset, for 50 epochs. The ratio of validation data to training data was 1:9 i.e. 10% of the total data was used for validation. The loss was calculated using a custom function, and regular logs were created after every 2 epochs. Validation loss was used as a metric during training, and the final evaluation metric was mAP, as explained in the next section.

RetinaNet has been implemented using Fizyr model, which is a reliable and bug-free model design. It involves various data augmentation techniques and its hyperparameters are decided after much experimenting using keras-tensorflow libraries. The model used pretrained weights as on the COCO dataset, and uses ResNet 50 as a backbone network.

C. Evaluation Measures

For validation purposes, the YOLOv3 model uses a custom loss function which contains cross-entropy terms, and thus uses logistic regression.

The RetinaNet model uses the alpha variant of the focal loss cross-entropy function for calculation of validation loss.

The final evaluation metric for Object Detection purposes is usually Average Precision (AP). Average precision (AP) is the area under the precision-recall curve, precision being the accuracy and recall being the measure of if all the positives are identified. Since both precision and recall are values between 0 and 1, the value of AP is also the same. mAP (mean average precision) is the average of AP computed over each class.

$$AP = \int_0^1 p(r)\, dr$$

D. Testing Examples

The Open Images Challenge 2019 provided their own test dataset for the challenge, which contains about 100,000 images. Shown below are some of the output images that were generated by the model, along with the text annotations generated for the same. As the model was trained on a fraction of the extensive actual dataset, and for lesser epochs, due to resource constraints, the accuracy remains to be further improved. However, it is still fairly good at detecting objects and their classes.

Initially both models were trained on a set of 895 images, selected at random from the dataset. This resulted in a random distribution of the classes appearing in the image annotations, but the models were able to pick up on some of the common classes such as everyday objects, humans, dogs, cats etc.

YOLOv3 showed a loss of 80.02, while RetinaNet showed a loss of 2.1.

Fig. 4. YOLOv3 Test results for two images, with the bounding boxes drawn.
The first image identifies the classes 'Dog', 'Carnivore', and 'Animal'.
The second image identifies the classes 'Man' and 'Person'.

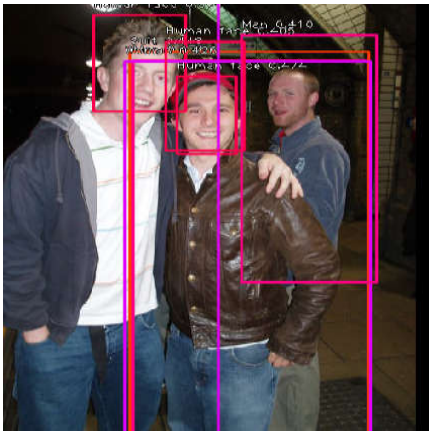

Fig. 5. RetinaNet Test Results for a multiple-object image, identifying 'Man' and 'Human Face'

V. Conclusion

Results indicate that our model can successfully detect the objects and categorize them into the specified classes and super classes, implementing the class hierarchy. Time and resource constraints did not permit us to train on the entire dataset, but we hope to achieve it in the future. This would greatly enhance our accuracy, due to the increased training data. Further modifications could be made to the models by adjusting hyper parameters, and further processing of the model output could result in a variety of results, according to the desired applications. The project gave us deeper insights into the domain of Deep Learning Applications, as well as the opportunity to work on supercomputer GPUs.

References

[1] " Open Images Challenge 2019: Object Detection Track " https://www.kaggle.com/c/open-images-2019-object-detection

[2] " Open Images Dataset Stats V5 " https://storage.googleapis.com/openimages/web/factsfigures.html

[3] " A Keras Implementation of YOLOv3 (Tensorflow backend) " https://github.com/qqwweee/keras-yolo3

[4] " YOLO: Real Time Object Detection " https://github.com/pjreddie/darknet

[5] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement" https://arxiv.org/abs/1804.02767

[6] Sik ho Tsang, "Review FPN"

[7] Tsung -Yi lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar, "Focal loss for dense object detection"

[8] " Keras implementation of RetinaNet object detection" https://github.com/fizyr/keras-retinanet