# EagleView Assignment: Object Detection using Images from the COCO Dataset

Subodh Lonkar
Pune, India
learner.subodh@gmail.com
Code Repository: https://github.com/learner-subodh/EagleView_COCO-OD

***Abstract*— Object Detection has always been a vital problem and have always been looked upon with a great importance. Significance of this problem has increased even more with the use cases and applications like Tracking objects, detecting vehicles, self-driving cars, counting the crowd and many more that demand precise detection and localization of various objects. On the other hand, neural networks have taken the world by storm. And no surprises, that the area of Computer Vision and the problem of facial expressions recognitions hasn't remained untouched. Various algorithms and techniques like YOLO and R-CNNs have made huge contributions to this field. We adopt the fundamental concepts of deep learning and computer vision and use Google's EfficientDet architecture for solving this problem. This is just a first-hand implementation and can be improved if put in more time and resources.***

## I. INTRODUCTION

Object detection is a technique of the AI subset computer vision that is concerned with identifying objects and defining those by placing into distinct categories such as humans, cars, animals etc. It is a computer vision technique that allows us to identify and locate objects in an image or video. With this kind of identification and localization, object detection can be used to count objects in a scene and determine and track their precise locations, all while accurately labeling them. Object detection allows us to at once classify the types of things found while also locating instances of them within the image. It combines machine learning and deep learning to enable machines to identify different objects.

However, image recognition and object detection these terms are often used interchangeably but, both techniques are different. Object detection could detect multiple objects in an image or, in a video. Image recognition assigns a label to an image. A picture of a dog receives the label “dog”. A picture of two dogs, still receives the label “dog”. Object detection, on the other hand, draws a box around each dog and labels the box “dog”. The model predicts where each object is and what label should be applied. In that way, object detection provides more information about an image than recognition. The demand for trained experts in this field is pretty high and having a background in deep learning for computer vision with python can prove extremely vital in today's world.

*Why is Object Detection Important?*

Object detection is inextricably linked to other similar computer vision techniques like image recognition and image segmentation, in that it helps us understand and analyse scenes in images or video. But there are important differences. Image recognition only outputs a class label for an identified object, and image segmentation creates a pixel-level understanding of a scene’s elements. What separates object detection from these other tasks is its unique ability to locate objects within an image or video. This then allows us to count and then track those objects. Given these key distinctions and object detection’s unique capabilities, we can see how it can be applied in a number of ways:

- Crowd counting
- Self-driving cars
- Video surveillance
- Face detection
- Anomaly detection

Of course, this isn’t an exhaustive list, but it includes some of the primary ways in which object detection is shaping our future.

## II. COCO DATASET

The MS COCO dataset is a large-scale object detection, segmentation, and captioning dataset published by Microsoft. Machine Learning and Computer Vision engineers popularly use the COCO dataset for various computer vision projects.

Understanding visual scenes is a primary goal of computer vision; it involves recognizing what objects are present, localizing the objects in 2D and 3D, determining the object’s attributes, and characterizing the relationship between objects. Therefore, algorithms for object detection and object classification can be trained using the dataset.

COCO stands for Common Objects in Context, as the image dataset was created with the goal of advancing image recognition. The COCO dataset contains challenging, high-quality visual datasets for computer vision, mostly state-of-the-art neural networks. For example, COCO is often used to benchmark algorithms to compare the performance of real-time object detection. The format of the COCO dataset is automatically interpreted by advanced neural network libraries. It contains over 200’000 images of the total 330’000 images are labeled with 80 object categories which include “things” for which individual instances may be easily labeled (person, car, chair, etc.)

However, here we will be working with just two categories, Persons and Cars.

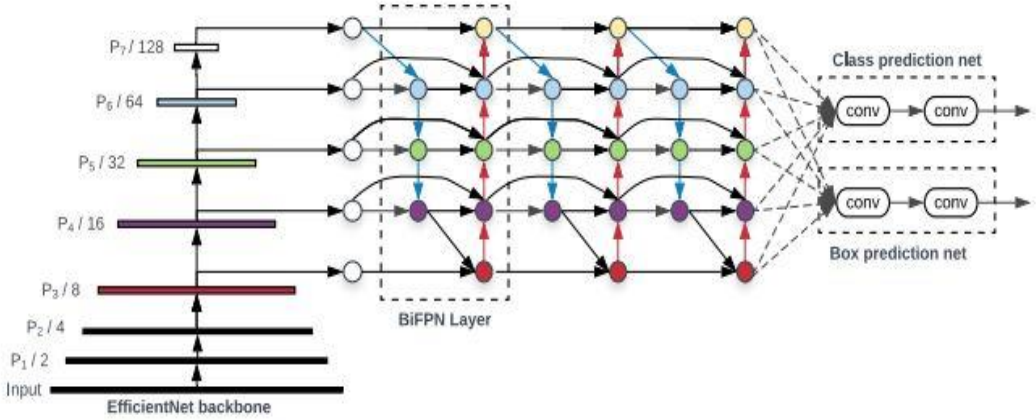


**Figure 1:** EfficientDet architecture – It employs EfficientNet as the backbone network, BiFPN as the feature network, and shared class/box prediction network. Both BiFPN layers and class/box net layers are repeated multiple times based on different resource constraints

## III. APPROACH

A number of popular object detection models belong to the R-CNN family. Short for region convolutional neural network, these architectures are based on the region proposal structure discussed above. Over the years, they've become both more accurate and more computationally efficient. There are also a number of models that belong to the single shot detector family. The main difference between these variants are their encoders and the specific configuration of predetermined anchors. MobileNet + SSD models feature a MobileNet-based encoder, SqueezeDet borrows the SqueezeNet encoder, and the YOLO model features its own convolutional architecture. SSDs make great choices for models destined for mobile or embedded devices.

But here I have used EfficientDet. In general, object detectors have three main components: a backbone that extracts features from the given image; a feature network that takes multiple levels of features from the backbone as input and outputs a list of fused features that represent salient characteristics of the image; and the final class/box network that uses the fused features to predict the class and location of each object. Refer Figure 1.

While most previous detectors simply employ a top-down feature pyramid network (FPN), we find top-down FPN is inherently limited by the one-way information flow. Alternative FPNs, such as PANet, add an additional bottom-up flow at the cost of more computation. Recent efforts to leverage neural architecture search (NAS) discovered the more complex NAS-FPN architecture. However, while this network structure is effective, it is also irregular and highly optimized for a specific task, which makes it difficult to adapt to other tasks. To address these issues, we propose a new bi-directional feature network, BiFPN, which incorporates the multi-level feature fusion idea from FPN/PANet/NAS-FPN that enables information to flow in both the top-down and bottom-up directions, while using regular and efficient connections. Refer Figure 2.

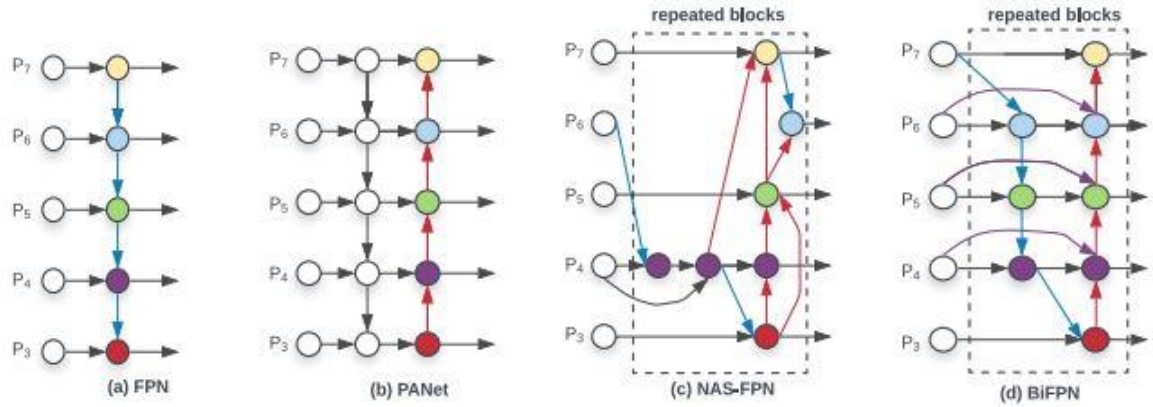


**Figure 2:** Feature network design – (a) FPN [23] introduces a top-down pathway to fuse multi-scale features from level 3 to 7 (P3 - P7); (b) PANet [26] adds an additional bottom-up pathway on top of FPN; (c) NAS-FPN [10] use neural architecture search to find an irregular feature network topology and then repeatedly apply the same block; (d) is our BiFPN with better accuracy and efficiency trade-offs.

*A. Assumptions:*

As the training data consists of only two classes, namely, Persons and Cars, I have assumed that the model in being built only for detection of these two object categories and thus, the model won't be able to detect other object categories. I have also assumed that we do not need real-time object detection and localization, thus, I haven't taken any such extra care for making it work into real-time. As the given data consists of training images, I have assumed that we cannot use any extra training data, thus, haven't used any other data or haven't increased the size &/or variety of dataset by performing image augmentation.

*B. Environment and Hardware:*

I have used Google Colaboratory for this task and have used Nvidia Tesla T4 Single GPU system.

*C. Overview of steps followed:*

I created a new GitHub repository and have used both the images and their annotations and prepared a combined dataframe. Various directories have been created for saving various data which can be seen in the ipynb. The task has been implemented using Tensorflow2.0. The given dataset contains 2239 images and 16772 annotations in those images with 2 classes, Person and Car.

*D. Metrics:*

Average Precision (AP) and Intersection Upon Union (IoU).

*E. Bounding Boxes:*

The COCO bounding box format is [top left x position, top left y position, width, height] which have been closely looked at and these transformations have been applied to the images.

*F. TensorFlow Records:*

I have the created TensorFlow Records for BytesList, FloatLists and Int64List and have performed a random train-test split by using 90% of the data for training & 10% of the data for testing & out of the train data we will be using 10% of the data for cross validation.

*G. Data Leakage:*

I have also checked for the data leakage problem just to make sure that we don't have overlapping images in validation or test datasets.

*H. Configurations:*

i. num_classes: 2 => As we have two classes, Person and Car.
ii. var_freeze_expr: (efficientnet|fpn_cells|resample_p6) => As we are using Google's efficientdet model architecture.
iii. lable_map: {1: 'person', 2: 'car'} => As we have two classes, Person and Car.
iv. train_file_pattern: train.record => As we will be using train.record TFRecords for the purpose of training.
v. val_file_pattern: valid.record => As we will be using valid.record TFRecords for the purpose of validation.
vi. model_name: efficientdet-d2
vii. Saving checkpoints to the checkpoints folder.
viii. Train Batch Size: 16
ix. Validation Batch Size: 16
x. Number of Epochs: 12

## IV. RESULTS & PERFORMANCE

Our task here is to develop and train a model on this dataset to detect person and cars in an image. Result images can be found under the *results* folder in the repository. Some sample result images are shown below. Refer Figures 4.1, 4.2 & 4.3 for sample result images.

Under similar accuracy constraints, EfficientDet models are 2x-4x faster on GPU, and 5x-11x faster on CPU than other detectors. Refer Figure 3 for performance insights of EfficientDet.

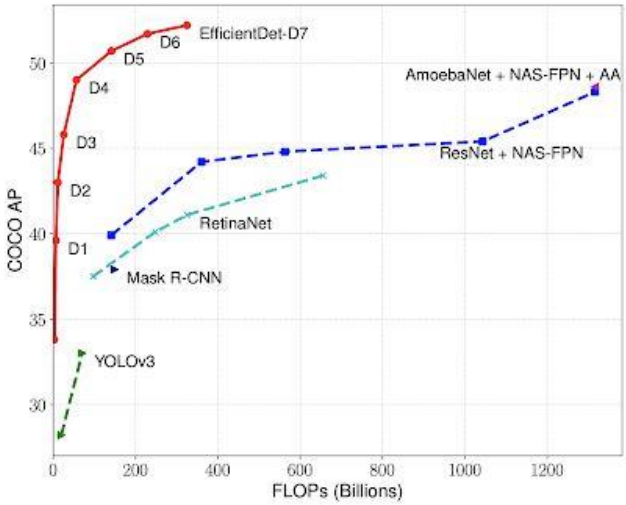


**Figure 3:** EfficientDet achieves state-of-the-art 52.2 mAP, up 1.5 points from the prior state of the art (not shown since it is at 3045B FLOPs) on COCO test-dev under the same setting. Under the same accuracy constraint, EfficientDet models are 4x-9x smaller and use 13x-42x less computation than previous detectors.

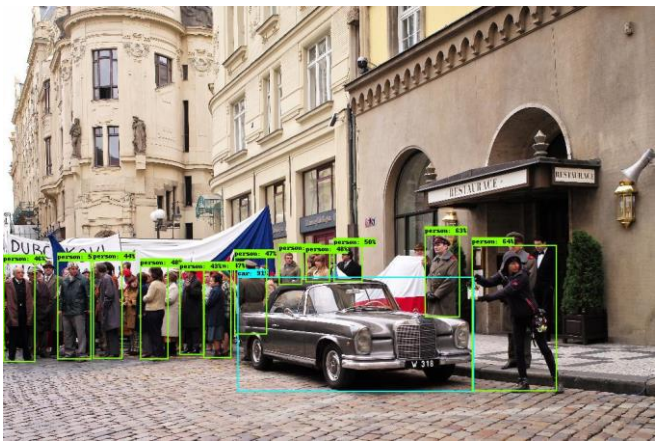

**Figure 4.1:** Sample Result Image

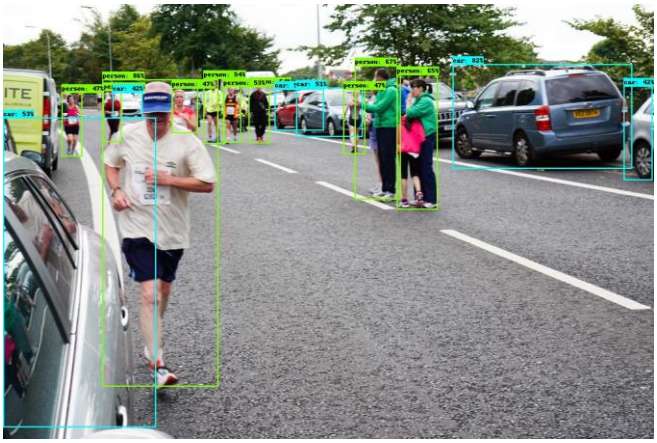

**Figure 4.2:** Sample Result Image

**Figure 4.3:** Sample Result Image

## V. STEPS TO REPLICATE RESULTS

Results discussed can be replicated by following the below steps:

i. Use *EagleView_COCO-OD.ipynb*
ii. Can be run either on Google Colab or on local environment
iii. To infer an image: *scripts* > *infer_single_image.py*

## VI. CONCLUSION

The problem of object detection is an extremely important one and can be solved in various ways by adopting various deep learning techniques. Here, I have used Google's EfficientDet for performing this task and have achieved decent first-hand results. These results can be easily interpreted by looking at the result images. EfficientDet performs a lot better than other approaches and can be seen from the figure. Considering limited computational resources, I have trained the model for 12 epochs. These results can be further improved by tuning the model hyperparameters or even by choosing some other technique over the EfficientDet.

## VII. FUTURE SCOPE

This is just a first-hand implementation of the task and following things can be tried to generate better results:

### A. *Deployment:*

The built system can be deployed to be used either as a web application or can be embedded in some target device. Necessary optimizations need to be performed in the code and the way be implement it as per the target device of deployment and the computational resources available. Easiest way to deploy for having a first-and demo will be making use of Streamlit and GitHub for building a web application.

### B. *Trying other techniques & more aggressive hyperparameter tuning:*

I have used the EfficientDet, but we can always give a shot by trying out some other state-of-the-art techniques like the YOLO or R-CNNs. Furthermore, more time and resources can be put in tuning the model hyperparameters in order to get the best results.

### C. *Image Augmentation:*

I haven't applied any augmentation techniques assuming that only the give data has to be used to training without any further modifications or additions. Performing augmentation will surely generate better results. Some operations that might prove significant in getting better results are zoom-in and zoom-out, increase and decrease in brightness or illumination, horizontal flipping and sheer operations, slight degree of rotation, etc.

## VIII. REFERENCES

Following resources have certainly helped me in getting a first-hand solution for this task:


[1] COCO Dataset: https://cocodataset.org/#home
[2] Google's automl repository for efficientdet: https://github.com/google/automl/tree/master/efficientdet
[3] EfficientDet: Scalable and Efficient Object Detection: https://arxiv.org/pdf/1911.09070.pdf
[4] Google AI Blog: EfficientDet: Towards Scalable and Efficient Object Detection: https://ai.googleblog.com/2020/04/efficientdet-towards-scalable-and.html
[5] Analytics India Magazine: EfficientDet: Guide to State-of-The-Art Object Detection Model: https://analyticsindiamag.com/efficientdet/
[6] Object detection: https://www.fritz.ai/object-detection/
[7] Computer Vision- COCO Dataset: https://viso.ai/computer-vision/coco-dataset/