Zaid Alayedy

Patricia Mcmanus

ITAI-1378

2 December/2024

Object Detection using TensorFlow and Pascal VOC

Object Detection Evaluation Core Concepts

Object detection models need to be evaluated on two fronts:

Classification Accuracy: Did the model correctly identify the object's class (e.g., person, car, bird)?

Yes, the model's classification accuracy indicates whether it correctly identified the object's class (e.g., person, car, bird). This metric evaluates the model's ability to assign the correct label to detected objects in the dataset.

Localization Accuracy: Did the model accurately draw a bounding box around the object?

Yes, localization accuracy measures whether the model accurately drew a bounding box around the object. This is typically evaluated using metrics like Intersection Over Union (IoU), which compares the overlap between the predicted bounding box and the ground truth. A higher IoU indicates better localization

Questions for Reflection and Analysis:

Conceptual Understanding:

What is the main difference between image classification and object detection? How is this difference evident in the output of this exercise?

The main difference between image classification and object detection are the goals and the kind of output they generate. Image classification aims to discover whether a particular object or class is represented in the whole image. It labels all the images with one label or probability representing the class it is most likely a part of. For example, when we had an image with a cat, the model would return "cat" as the caption for the entire image, regardless of where the cat is in the picture and anything else in the scene.

Object detection, by contrast, detects and places multiple objects in an image. It produces a more complex output, generating bounding boxes around each object it finds and labelling them with its class. For instance, if you have an image of a dog and a cat, the model will create bounding boxes for both "dog" and "cat". In this example, the distinction is clear as object detection returns both classification (labels for objects found) and localization (bounding boxes), whereas image classification would have returned only one label per image, unaltered with object positions.

Explain why we chose the SSD MobileNet V2 model for this task. What are its advantages and limitations, especially in the context of limited computational resources?

SSD MobileNet V2 was selected for this purpose because it offers a great combination of speed, precision, and performance that would be optimal for real-time object detection in low-power devices. SSD (Single Shot MultiBox Detector) - SSD detects multiple objects in one forward network run and is therefore faster than region-based detectors. This is further optimized by depthwise separable convolutions, which cut the number of parameters and computation costs significantly without sacrificing accuracy using MobileNet V2, a lightweight deep neural network. The combination makes SSD MobileNet V2 especially useful for applications that need to be deployed at the edge such as phones, drones, or embedded systems.

But the model is not infinite. It is computationally fast, but not as good as heavier models such as Faster R-CNN when it comes to identifying small or overlapped objects. The small architecture will not fare well in large scenes where precise feature extraction is essential. SSD MobileNet V2 offers a reasonable trade-off in performance with a reduction in accuracy and with a reduction in inference time and hardware complexity in a small compute environment, where the real-time speed and portability are critical factors.

Code Interpretation:

Describe the role of the find\_images\_with\_classes function. Why is it useful when working with a large dataset like COCO?

The find\_images\_with\_classes function is essential for managing and organizing large datasets like COCO, as it identifies and retrieves images containing specific object classes. This function streamlines the process of data selection, ensuring that the subset of images used for training or evaluation focuses on relevant categories, such as "car," "person," or "cat." In a dataset as vast and diverse as COCO, this targeted selection is critical for reducing computational overhead and avoiding the inefficiency of processing irrelevant data. Additionally, the function helps create smaller, customized datasets tailored to specific tasks or experiments, such as detecting only vehicles or animals. By filtering and balancing the data representation of the desired classes, it also improves model training and evaluation by preventing the inclusion of unnecessary or redundant data that might skew results. Overall, the find\_images\_with\_classes function is a powerful tool for enhancing efficiency, relevance, and control when working with large-scale datasets.

In the plot\_detections function, how does the threshold value (threshold=0.5) impact the number of objects displayed?

Explain how the heatmap visualization helps you understand the model's confidence in its detections.

The threshold value in the plot\_detections function determines the minimum confidence level for displaying detected objects; a higher threshold (e.g., 0.7) shows only highly confident detections, reducing false positives but potentially missing some objects, while a lower threshold (e.g., 0.3) includes more detections, risking false positives. The heatmap visualization provides insight into the model's confidence by highlighting areas of high probability for object presence, helping to understand confidence distribution, identify errors, evaluate localization accuracy, and guide threshold tuning by visually representing how the model prioritizes regions in the image.

Observing Results and Limitations:

Run the exercise multiple times. Which types of objects does the model tend to detect more accurately? Which ones are more challenging? Can you explain why?

After running the exercise repeatedly, the model detects larger and distinct objects (cars, people, or any other objects with a large background image) more accurately. These are easier objects for the model to recognise as they are feature rich, have sufficient spatial information and are not prone to mixing with other objects. These classes are likely better learned by the model because there’s a lot of labeled information in the training dataset, and they tend to dominate images more generally.

Conversely, when small, overlapped, or less well-defined objects are considered (such as small animals, objects that are partially blocked, or items with textures blurring out background), the model fails. These are harder to recognize since they contain relatively little information within the bounding box and the model is less capable of gleaning distinctive features. The variation in lighting, background and obscurity is another factor that makes these things difficult to distinguish.

This error in detection may often be explained by a combination of the wrong dataset, class undercount in the training data, and the fundamental model architecture’s inability to accommodate small or interfering objects. In order to optimise performance, methods such as data augmentation, rebalancing the dataset or more elaborate architectures might be required.

Observe the bounding boxes. Are there any instances where the boxes are inaccurate or miss the object entirely? What factors in the images might be contributing to these errors?

Bounding box inaccuracies or missed objects in object detection models often occur due to several factors present in the images. Inaccuracies may include bounding boxes that are too large, too small, or off-center, while missed objects occur when the model fails to detect an object entirely. These errors are frequently caused by challenges such as small object size, which limits identifiable features, or occlusion, where objects are partially hidden behind others. Poor lighting conditions or shadows can obscure object features, making recognition and localization more difficult. Similarly, complex or noisy backgrounds can confuse the model, leading it to misinterpret background elements as part of the object. Additionally, objects that resemble other classes in the dataset may be misclassified, while unusual viewing angles or distortions can hinder the model’s ability to recognize or properly localize objects. These issues underscore the need for diverse training datasets, improved annotation practices, and advanced techniques to enhance the model’s accuracy and robustness.

How would you expect the accuracy of the model to change if we had used the entire Pascal VOC 2007 dataset instead of a small subset? Why?

Had the model been trained on the entire Pascal VOC 2007 dataset, instead of a small subset, then it should have come up very close. It’s because a larger dataset yields a greater range of examples for each class, including variation in object size, orientation, lighting, and background. That variation allows the model to generalise and identify objects across a broader range of actual circumstances. Training on a small sample can also result in overfitting, where the model excels on the small sample of samples it has observed but does poorly on new/unseen data. The full dataset exposes the model to more conditions and reduces overfitting while allowing it to better recognize and classify objects under various conditions. Additionally, the more data, the more features the model will be able to learn for difficult classes which makes it more robust and reliable.

Critical Thinking:

How could you modify the code to detect a specific set of objects, like only animals or only vehicles?

You can filter the classes of interest either during data prep or post processing to adjust the code to detect just a few objects, like only animals or only vehicles. When creating the dataset, you can customize the data loader to only load the target classes by excluding images and annotations that do not contain animals (e.g., "cat," "dog," "bird") or vehicles ("car," "bus," "bicycle"). Instead, at the post-processing step, you can filter out objects that are detected for non-target classes so that only objects from your target classes are captured. We can achieve this by defining a function that sorts detected objects based on class labels before rendering or testing them. You can also edit the model’s class mappings to be oriented toward the correct classes and have the model output reflect the correct classes. Finally, the evaluation metrics like precision and recall must be adjusted to estimate performance for the targeted group. Such adjustments reduce detection steps and enable the model to consider only certain object types, but not others.

If you wanted to train your own object detection model, what steps would you need to take?

For training an object detection model, begin by specifying the problem and selecting a model architecture such as SSD, YOLO, or Faster R-CNN for your speed and accuracy. Gather and fill bounding boxes of diverse data with labels such as LabelImg. Split the data into training, validation, and test sets, and augment it with augmentations such as flipping and scaling to make it more robust. : Use a deep learning framework such as TensorFlow or PyTorch and train the model using pretrained weights for transfer learning. Create a pipeline with proper loss functions, an optimizer (Adam, SGD), and learning rate schedule. Train the model, test and validate its accuracy based on metrics such as mAP and precision, run it on hidden data and show predictions. Parameterize or load data, and deploy with tools such as TensorFlow Lite or ONNX. At last, run the model into your production environment and use it in production to make sure it is appropriate.

What are some challenges you might encounter?

Training an object detection model is difficult in a number of ways, beginning with collecting and annotation a huge, varied dataset that can take time and become error-prone. Class imbalance — in which certain classes over and others under-represent themselves — can result in discriminatory predictions and inaccurate generalisations for minority classes. Ample or unbalanced data also makes the model prone to overfitting when used for new data. Training these models requires a huge amount of computational power (GPUs or TPUs) that might not always be available. Small or occluded objects are difficult to find because there is less visual detail available, and blurry or noisy backgrounds may induce false positives or misses. Additionally, hyperparameter tuning, such as modifying the learning rate or augmentation algorithms, can be time-consuming and iterative.

Given the limitations of this model, in what real-world scenarios might it still be useful for object detection?

1. Surveillance and Security: The model can be used to detect individuals, vehicles, or anomalous objects in an environment being tracked. It’s moderately accurate enough to warn operators about potential hazards.
2. Shop Analytics: In a shop, it can analyze movement of customer, foot traffic or stock, providing valuable business intelligence.
3. Autonomous Vehicles: While not suited for many situations, the model is able to help distinguish simple road-based objects such as vehicles, pedestrians or stop signs in a controlled environment.

Reference:

Machine Learning and Object Detection Basics:

TensorFlow Object Detection API Documentation.

YOLO and Faster R-CNN Model Information: https://pjreddie.com/darknet/yolo/.