Object Detection Using TensorFlow and COCO Dataset

Varit Kobutra

Houston Community College

Author notes

This journal is being submitted on July 18, 2024, for Professor Patricia McManus for Computer Vision ITAI 1378: 12461 at Houston Community College by Varit Kobutra.

Abstract

This reflective journal documents the process and outcomes of an object detection exercise using the TensorFlow framework and the COCO dataset. The exercise was conducted on Google Colab and Amazon SageMaker StudioLab, utilizing the SSD MobileNet V2 model for object detection. The journal explores the conceptual differences between image classification and object detection, the rationale behind choosing SSD MobileNet V2, and the functionality of key code components. Observations from the exercise highlight challenges such as dataset loading issues, resource constraints, and model accuracy. The journal concludes with critical reflections on modifying the code for specific object detection tasks, the steps involved in training a custom object detection model, and potential real-world applications of the SSD MobileNet V2 model.

Conceptual Understanding

Difference Between Image Classification and Object Detection

Image classification and object detection are both crucial tasks in computer vision but serve different purposes. Image classification assigns a single label to an entire image, identifying the primary subject (Chooch, 2024). In contrast, object detection identifies and locates multiple objects within an image, providing bounding boxes and class labels for each detected object (LabelYourData, 2023). This distinction is evident in the output of this exercise, where the object detection model outputs multiple bounding boxes and labels within a single image, as opposed to a single label for the entire image.

Choosing SSD MobileNet V2

The SSD MobileNet V2 model was chosen for its balance between accuracy and computational efficiency. It combines the efficiency of the Single Shot Multibox Detector (SSD) with the low complexity of MobileNet V2, making it suitable for devices with limited computational resources (Heerthi, 2024). The model uses depth-wise separable convolutions to reduce the number of parameters and computational cost, which is advantageous for real-time applications. However, its limitations include slightly lower accuracy compared to more complex models like Faster R-CNN, especially in scenarios requiring high precision (Kumar, 2023).

Code Interpretation

Role of `find_images_with_classes` Function

The `find_images_with_classes` function filters and retrieves images containing specific classes from a large dataset like COCO. This function is useful for focusing on

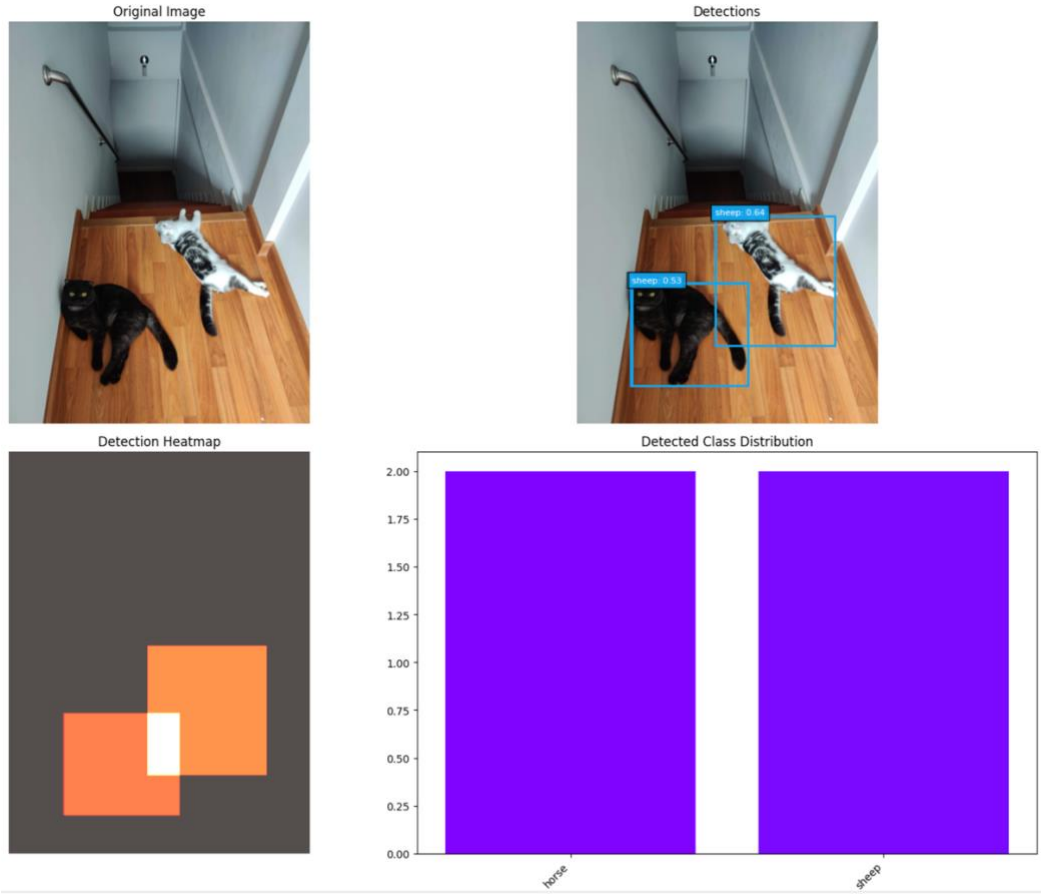
objects of interest and reducing the computational load by avoiding unnecessary processing of irrelevant images (Heerthi, 2024).

Impact of Threshold in `plot_detections` Function

The threshold value in the `plot_detections` function determines the minimum confidence score for displaying detected objects. A threshold of 0.5 means that only detections with a confidence score of 50% or higher will be shown. Lowering the threshold would display more objects, including less confident detections while raising it would show fewer, more confident detections (Heerthi, 2024).

Heatmap Visualization

Heatmap visualization helps in understanding the model's confidence by highlighting areas in the image where the model is most certain about its detections. This can reveal which parts of the image the model focuses on and how confident it is in identifying objects (Py-Feat, 2023).



(Image of uploading images of my own cats and process image detection)

Observing Results and Limitations

Accuracy of Object Detection

Upon running the exercise multiple times, the model tended to detect larger and more distinct objects more accurately, while smaller or partially obscured objects posed more challenges. For example, the model detected objects but often failed to classify them correctly. The initial detection results were amusingly inaccurate, with only 11 correct detections out of 234, resulting in an accuracy of 0.05. Factors contributing to these errors include occlusion, viewpoint variation, and complex backgrounds (Chooch, 2024).

Bounding Box Accuracy

Bounding boxes were sometimes inaccurate or missed objects entirely due to factors such as low resolution, occlusion, or similar colors between the object and background. These issues affected the model's ability to precisely localize objects. In one instance, an image of two cats was uploaded, and while the model detected them as objects, it did not classify them correctly (Chooch, 2024).

Using Entire COCO Dataset

Using the entire COCO dataset instead of a subset would likely improve the model's accuracy due to a larger and more diverse set of training examples. This would help the model generalize better to different objects and scenarios (Heerthi, 2024).

Technical Challenges and Solutions

During the exercise, several technical challenges were encountered. Initially, there was a failure to load the COCO dataset on Google Colab due to a connection issue. Switching to Amazon SageMaker StudioLab required using the headless version of OpenCV, as the standard version did not work out of the box. Despite this, resource exhaustion issues persisted, prompting an upgrade to a Google Colab Pro account, which provided over 200GB of disk space. The download and extraction of the dataset took approximately 12 minutes and 50 seconds, and creating the dataset took an additional 24 minutes.

Critical Thinking

Modifying Code for Specific Object Detection

To detect a specific set of objects, you can modify the code to filter detections by class IDs corresponding to those objects. For example, if you only want to detect animals, you can set the target classes to include only the class IDs for animals (GitHub, 2020).

Training Your Own Object Detection Model

To train your own object detection model, you would need to:

- Collect and annotate a dataset with bounding boxes and class labels.
- Choose an appropriate model architecture.
- Configure the training pipeline, including data augmentation and hyperparameters.
- Train the model on your dataset.
- Evaluate and fine-tune the model based on performance metrics like mAP and IoU.

Challenges include obtaining a sufficiently large and diverse dataset, ensuring accurate annotations, and managing computational resources for training (Heerthi, 2024).

Real-World Scenarios for SSD MobileNet V2

Despite its limitations, SSD MobileNet V2 can be useful in real-world scenarios requiring real-time object detection on devices with limited computational power, such as mobile phones, drones, and embedded systems for surveillance or industrial automation (Kumar, 2023).

Going Further

Comparing Other Models

Researching other object detection models available in TensorFlow Hub, such as Faster R-CNN, YOLO, and EfficientDet, reveals that:

- Faster R-CNN offers higher accuracy but is slower and more computationally intensive.
- YOLO is faster and suitable for real-time applications but may sacrifice some accuracy.
- EfficientDet balances accuracy and efficiency, making it suitable for various applications with limited resources (Heerthi, 2024).

References

Chooch. (2024). *What's the difference between object & image recognition?* Retrieved from https://www.chooch.com/blog/whats-the-difference-between-object-recognition-and-image-recognition/

GitHub. (2020). *Filtering ssd-mobilenet-v2*. Retrieved from https://github.com/dusty-nv/jetson-inference/issues/635

Heerthi, R. (2024). *Building an object detection system with MobileNet SSD and OpenCV.* Retrieved from https://www.linkedin.com/pulse/building-object-detection-system-mobilenet-ssd-opencv-heerthi-raja-h-ftb9c

Kumar, S. (2023). *Real-time detection of road-based objects using SSD MobileNet-v2 FPNlite with a new benchmark dataset.* Retrieved from https://typeset.io/questions/what-are-the-computational-advantages-of-using-mobilenet-ssd-gzalihqdw5

LabelYourData. (2023). *Image classification vs. object detection: Key differences*. Retrieved from https://labelyourdata.com/articles/object-detection-vs-image-classification

Py-Feat. (2023). *Detecting facial expressions from images*. Retrieved from https://py-feat.org/basic_tutorials/02_detector_imgs.html