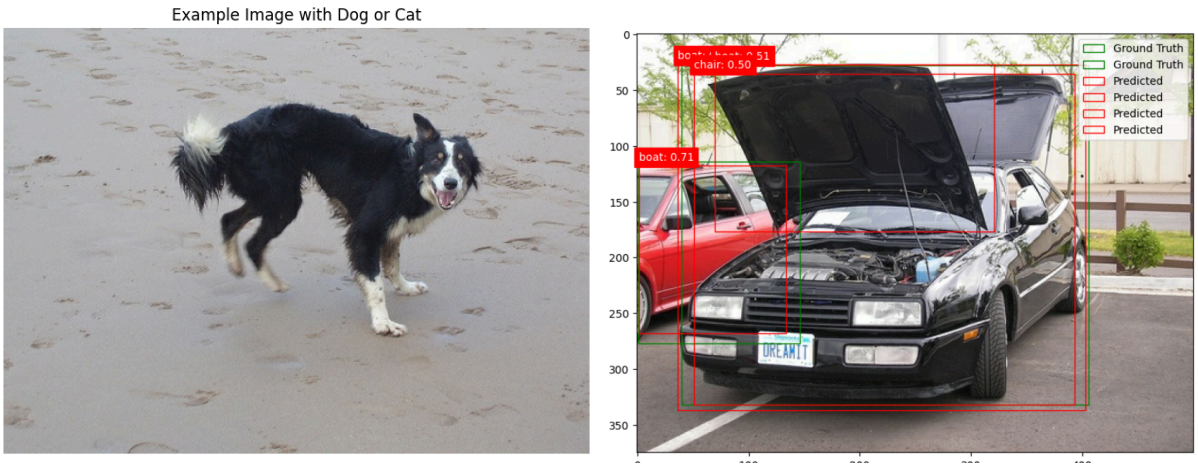
Ulises Zamarripa

Lab06 Journal

The core concept of this lab was to adapt our image classification code to object detection. I learned that image classification assigns a single label to the image, while object detection focuses on multiple objects within an image. After loading every library, I noticed that the load_data() function has a TensorFlow module tfds.load(). It automatically downloads the PASCAL VOC 2007 dataset, which has been trained on the COCO dataset (which is a large-scale object detection dataset), making it efficient for object detection tasks. The advantages include high efficiency, a small model size, and an excellent balance between speed and accuracy. The limitations include: lower accuracy compared to larger models, and poor performance on small objects.

image classification || object detection



We chose SSD Mobilenet V2 because it was lightweight and fast, the only tradeoff being its accuracy, as seen from the picture above, which is perfect for learning, since the outcome was to understand the core concepts, not accuracy. We use hub.load() to load the model into a detector. Essentially, turning our image classification code into an object detection code, because now we have access to functions like:

- `num_detections`: a `tf.int` tensor with only one value, the number of detections `[N]`.
- `detection_boxes`: a `tf.float32` tensor of shape `[N, 4]` containing bounding box coordinates in the following order: `[ymin, xmin, ymax, xmax]`.
- `detection_classes`: a `tf.int` tensor of shape `[N]` containing detection class index from the label file.
- `detection_scores`: a `tf.float32` tensor of shape `[N]` containing detection scores.
- `raw_detection_boxes`: a `tf.float32` tensor of shape `[1, M, 4]` containing decoded detection boxes without Non-Max suppression. `M` is the number of raw detections.
- `raw_detection_scores`: a `tf.float32` tensor of shape `[1, M, 90]` and contains class score logits for raw detection boxes. `M` is the number of raw detections.
- `detection_anchor_indices`: a `tf.float32` tensor of shape `[N]` and contains the anchor indices of the detections after NMS.
- `detection_multiclass_scores`: a `tf.float32` tensor of shape `[1, N, 91]` and contains class score distribution (including background) for detection boxes in the image including background class.

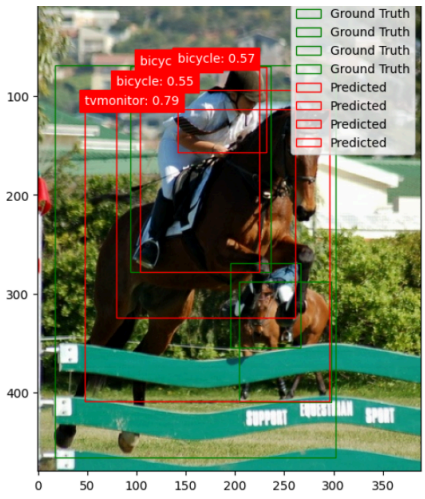
The role of the find_images_with_classes function is to act as a search filter, pulling out images with the targeted class. The function takes three arguments (train_dataset, class_names, target_classes=['dog']). This is useful when working with a large dataset, and it's crucial when you need to process a specific class of images. It's also used for testing a model's limits.

I observed that the red squares above the image contained a decimal number. I discovered that this number indicates the detection score, reflecting the model's confidence in the presence of an object. This is the output of the plot_detections function. The code checks if the score is below 0.5, but lowering the threshold will show more objects with less accuracy. Increasing the value will reduce the results but increase their accuracy. A heatmap visualization shows the model's confidence and state before its output. Rather than only displaying the final box, a heatmap colors areas of the image to indicate the model's likelihood of an object being present. This visualization offers more insight than just a number, as it shows how the model thinks.

The evaluate_model_performance function has five evaluation metrics. Two of the most important are precision and recall. Precision measures how accurately the model identifies an object. Recall means the number of objects that were positively identified.

$$precision = \frac{true\ possitives}{true\ possitives + Fasle\ possitives} \quad recall = \frac{true\ possitives}{true\ possitives + false\ negatives}$$

Since we are working with a small dataset, it struggles to find objects that are blocked by other objects. In the first picture, I can see that it's a car, but the model doesn't know that because its accuracy is low. It excels at identifying objects without obstruction, such as people, cars, and dogs. You can see from the picture below that the bounding boxes miss the object entirely and are incorrect because the objects are not fully visible. This could be changed if we used the entire Pascal VOC 2007 dataset instead of a small subset. Training on more data will enable our model to encounter a variety of backgrounds, object sizes, and lighting conditions. Making it more accurate.



This is how I would modify the code to detect a specific set of objects, like a dog.

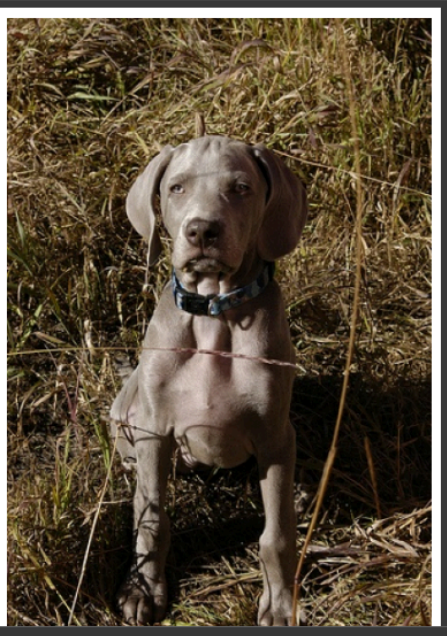
```
images_with_dog = find_images_with_classes(train_dataset, class_names, target_classes=['dog'])

if images_with_dog:

    example_to_display = images_with_dog[4]

    image_to_display = example_to_display["image"].numpy()

    # Display the image
    plt.figure(figsize=(8, 8))
    plt.imshow(image_to_display)
    plt.axis('off')
    plt.show()
else:
    print("No images")
```



bles Terminal

If I wanted to train my own model, I would start with a data collection phase. I would gather as much data as possible. After this step, I would preprocess my raw data by labeling objects to create a map. If I were working with pictures, I would ensure they are the same file type. After this, I would choose a model like SSD Mobilenet V2 and adjust the parameters such as learning rate and batch size. This step takes the most time due to the extensive experimenting needed for fine-tuning. While the model has its limitations, its speed and low resource consumption make it effective for phone applications.