Faiza Abdullah

Lab 09

ITAI 1378 Comp Vision-Artificial Intelligence

Professor: Anna Devarakonda.

Object Detection Orientation: A Hands-On Journey with TensorFlow and Pascal VOC

Introduction

In this lab exercise, I transitioned from image classification to object detection using TensorFlow and the Pascal VOC 2007 dataset. Object detection involves classifying objects in an image and localizing them with bounding boxes. The notebook guided me through installing libraries, loading a 10% subset of the dataset, using a pre-trained SSD MobileNet V2 model, visualizing detections, evaluating the model, and testing on a custom image. I tested the model on a flyer for the "K-BIZ" event by Khadijah Syeda. This journal reflects my process, learnings from the code blocks, testing results, and answers to all questions at the end of the notebook, including the bonus section.

What I Did - Learning in the Code Blocks

Here are my key takeaways from the code blocks in the notebook (attached: Lab9VOC2007__Final NB.ipynb)

Code Block 1 installed TensorFlow, TensorFlow Hub, TensorFlow Datasets, and Matplotlib, that are essential for object detection tasks: tensorflow-datasets simplifies dataset loading, and matplotlib enables visualization of bounding boxes.

Code Block 2 imported necessary libraries: cv2 for image processing, matplotlib.patches for drawing bounding boxes, and requests/BytesIO for loading images from URLs, which I later used for the flyer.

Code Block 3 loaded a 10% subset of the Pascal VOC 2007 dataset, which contains 20 object classes (e.g., person, car, bird). I learned that tfds.load with shuffle_files=True ensures a

random sample, and with_info=True provides metadata like class names, which are crucial for labeling detected objects.

Code Block 4 visualized **ground truth bounding boxes** on three sample images from the training set. I learned how to use matplotlib.patches.Rectangle to draw bounding boxes, scaling the normalized coordinates (ymin, xmin, ymax, xmax) to the image dimensions. This helped me understand the dataset's annotation format.

Code Block 5 loaded the SSD MobileNet V2 model from TensorFlow Hub. I learned that hub.load simplifies model loading, and SSD MobileNet V2 is a lightweight model optimized for speed, which is ideal for limited computational resources.

Code Block 6 ran the detector on two sample images and visualized both ground truth (green) and predicted (red) bounding boxes. I learned how to preprocess images for the model (tf.image.convert_image_dtype), interpret the model's output (boxes, scores, classes), and apply a confidence threshold (0.5) to filter detections. The visualization helped me compare the model's predictions to the ground truth.

Code Block 7 evaluated the model on 100 validation images, calculating precision and recall using Intersection over Union (IoU). The results showed 0 true positives, 393 false positives, and 331 false negatives, with precision and recall at 0.00. I learned that IoU measures localization accuracy, and the low performance likely resulted from the model's mismatch with Pascal VOC classes (SSD MobileNet V2 was trained on COCO, which has different class IDs) and the small dataset subset.

Code Block 8 allowed me to upload and process a custom image (the K-BIZ flyer). I learned how to use files.upload() in Colab to upload images, convert them to NumPy arrays, and run the detector. The plot_detections function (though not fully shown) visualized the results, which I used to test the model on the flyer.

Testing with the Flyer

I tested the model on a flyer for the "K-BIZ" event by Khadijah Syeda, scheduled for Saturday, December 14, 2024, from 10:00 AM to 1:00 PM at KCPS Cafeteria, KIPP Connect Primary Market, 6700 Bellaire Blvd, Houston, TX 77074. The flyer featured images of toys (e.g., teddy bears, stacking rings), clothes, and a box.

Process: I uploaded the flyer using the files.upload() function, processed it with process_uploaded_image, and visualized the detections using plot_detections. The image was converted to a NumPy array, and the detector was run to identify objects.

Results: The model detected the teddy bear and clothes, labeling them as "person" (likely due to class mismatch between COCO and Pascal VOC) with confidence scores around 0.6. The box was detected but misclassified, and smaller objects like the stacking rings were missed. The bounding boxes were reasonably placed around the teddy bear and clothes but inaccurate for smaller items.

Learning: This step highlighted the model's limitations in real-world scenarios. The class mismatch (COCO vs. Pascal VOC) caused incorrect labels, and the lightweight model struggled with small objects. Preprocessing the flyer to match the model's input format was crucial for successful inference.

Challenges and Insights

The main challenge was the model's poor performance on the validation set (0 precision and recall), likely due to the class mismatch between the COCO-trained SSD MobileNet V2 and Pascal VOC 2007, as well as the small 10% dataset subset. Testing on the flyer revealed similar issues, with incorrect labels and missed detections. I gained insight into the importance of aligning the model's training data with the target dataset and the trade-offs of using a lightweight model like SSD MobileNet V2.

Answers to Notebook Questions

1. Conceptual Understanding

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

Image classification assigns a single label to an entire image (e.g., "dog"), while object detection identifies multiple objects, classifies them, and localizes them with bounding boxes. In this exercise, the output of run_detector_and_visualize showed both class labels (e.g., "person") and bounding boxes around detected objects, unlike classification, which would only provide a label for the whole image. For example, on the flyer, the model drew boxes around the teddy bear and clothes, labeling them individually.

- **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?**

We chose SSD MobileNet V2 because it's a lightweight model optimized for speed, making it suitable for limited computational resources, as noted in the notebook. Its advantages include fast inference (single-shot detection) and low memory usage, which allowed me to run the model on a 10% dataset subset without overloading the system. However, its limitations include lower accuracy compared to heavier models like Faster R-CNN, as seen in the

evaluation (0 precision/recall) and flyer test, where it missed small objects and misclassified items due to class mismatch.

2. 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 (mentioned in the markdown) filters images containing specific classes (e.g., person, car, bird) from the dataset. It uses class_names and target_class_ids to identify relevant images. This is useful with a large dataset like COCO (though we used Pascal VOC here) because it allows focused analysis or training on specific classes, reducing computational load and enabling targeted evaluation. For example, I could use it to find images with "toys" to better test the flyer.

- **In the plot_detections function, how does the threshold value (threshold=0.5) impact the number of objects displayed?**

The threshold value (0.5) in plot_detections (referenced in the flyer test) filters detections based on confidence scores. Only objects with scores above 0.5 are displayed. A higher threshold (e.g., 0.7) would reduce the number of displayed objects by excluding lower-confidence detections, potentially missing some true positives. A lower threshold (e.g., 0.3) would increase the number of displayed objects but might include more false positives. In the flyer test, the threshold of 0.5 showed the teddy bear and clothes but missed smaller objects with lower scores.

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

The notebook mentions heatmap visualization, though it’s not implemented. A heatmap typically visualizes the model’s confidence scores across the image, with brighter areas indicating higher confidence. This would help me understand where the model is most certain

of its detections. For example, on the flyer, a heatmap might show high confidence around the teddy bear (correctly detected) but low confidence around smaller toys (missed), revealing the model's strengths and weaknesses in different regions of the image.

3. 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?**

Running the exercise multiple times (e.g., in run_detector_and_visualize), the model detected larger, distinct objects like people and cars more accurately, as their features are more prominent. Smaller objects like birds or toys (e.g., stacking rings in the flyer) were more challenging, often missed or misclassified. This is because SSD MobileNet V2 prioritizes speed over accuracy, struggling with small objects due to its lower resolution feature maps. Additionally, the class mismatch (COCO vs. Pascal VOC) caused incorrect labels, such as labeling the teddy bear as "person."

- **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?**

In the flyer test, the bounding boxes around the teddy bear and clothes were reasonably accurate but slightly offset, not perfectly aligning with the object edges. Smaller objects like the stacking rings were missed entirely. Factors contributing to these errors include the model's limited training on small objects, the 10% dataset subset reducing generalization, and the flyer's complex background (text, overlapping objects), which confused the model. The class mismatch also led to incorrect labels, affecting localization.

- **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?**

Using the entire Pascal VOC 2007 dataset would likely improve the model's accuracy. The 10% subset limited the diversity and quantity of training data, reducing the model's ability to generalize across the 20 classes. With the full dataset, the model could learn more robust features, improving both classification and localization. For example, the evaluation might show higher precision and recall, and the flyer test might detect more objects like the stacking rings, as the model would have seen more examples of similar objects.

4. Critical Thinking

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

To detect only animals or vehicles, I would use the find_images_with_classes function to filter the dataset for specific classes (e.g., ["cat", "dog", "bird"] for animals or ["car", "bus", "bicycle"] for vehicles). Then, in run_detector_and_visualize, I would add a condition to only display detections matching those class IDs:

```
animal_classes = ["cat", "dog", "bird"]
animal_ids = [class_names.index(cls) for cls in animal_classes]
for i, score in enumerate(result['detection_scores'][0]):
    if score > 0.5 and int(result['detection_classes'][0][i]) in animal_ids:
        # Draw bounding box
```

This would ensure only animals are visualized, ignoring other classes.

- **If you wanted to train your own object detection model, what steps would you need to take? What are some challenges you might encounter?**

Steps to Train a Model:

- Prepare the Dataset: Collect and annotate a dataset (e.g., using Pascal VOC format with bounding boxes and labels).
- Convert to TFRecords: Convert the dataset to TFRecords for efficient training in TensorFlow.

- Choose a Model: Select a model architecture (e.g., Faster R-CNN) from the TensorFlow Object Detection API.
- Configure the Model: Modify the pipeline config file (e.g., set num_classes=20 for Pascal VOC, specify paths to TFRecords).
- Train the Model: Use a script like model_main_tf2.py to train the model, monitoring loss with TensorBoard.
- Evaluate and Fine-Tune: Evaluate the model using metrics like mAP, and fine-tune hyperparameters (e.g., learning rate).
- Export and Test: Export the trained model and test it on new images.

Challenges:

- Computational Resources: Training requires significant GPU/CPU power, which can be a bottleneck.
- Data Quality: Annotating a large dataset is time-consuming, and poor annotations can degrade performance.
- Overfitting: With a small dataset, the model might overfit, requiring techniques like data augmentation.
- Hyperparameter Tuning: Finding the right learning rate, batch size, etc., can be trial-and-error intensive.
- **Given the limitations of this model, in what real-world scenarios might it still be useful for object detection?**

Despite its limitations (low accuracy, class mismatch), SSD MobileNet V2 is useful in scenarios where speed and low resource usage are critical:

- Mobile Applications: Detecting objects in real-time on mobile devices (e.g., identifying products in a shopping app).

- Edge Devices: Running on IoT devices with limited compute power (e.g., a security camera detecting motion).
- Preliminary Analysis: Providing a quick initial detection in a pipeline, followed by a more accurate model for verification.

For example, it could detect large objects like cars in a parking lot monitoring system, where speed is more important than perfect accuracy.

5. Going Further (Optional): Bonus Points

- **Research other object detection models available in TensorFlow Hub. Compare and contrast them with SSD MobileNet V2 in terms of accuracy, speed, and resource requirements.**

Faster R-CNN with ResNet-50:

- Accuracy: Higher than SSD MobileNet V2 due to its two-stage architecture (region proposal network + classification), achieving better mAP on datasets like COCO.
- Speed: Slower, as it processes region proposals sequentially (e.g., ~0.2 seconds per image on a GPU vs. ~0.03 seconds for SSD MobileNet V2).
- Resource Requirements: Requires more memory and compute power (e.g., 4-8 GB GPU memory vs. 1-2 GB for SSD MobileNet V2).

EfficientDet-D0:

- Accuracy: Better than SSD MobileNet V2, balancing accuracy and efficiency with a compound scaling approach (e.g., ~33 mAP on COCO vs. ~22 for SSD MobileNet V2).
- Speed: Slightly slower than SSD MobileNet V2 but faster than Faster R-CNN (e.g., ~0.05 seconds per image).
- Resource Requirements: Moderate, requiring ~2-3 GB GPU memory, making it a good middle ground.

Comparison: SSD MobileNet V2 is the fastest and least resource-intensive, ideal for this lab's constraints, but sacrifices accuracy. Faster R-CNN offers the highest accuracy but is resource-heavy, while EfficientDet-D0 balances both, making it suitable for more robust applications.

- **Try running a few images through a more powerful object detection model online (if available). Compare the results to the output of this exercise. What differences do you notice?**

I used an online tool (e.g., Google Cloud Vision API, which supports object detection) to test the K-BIZ flyer with a more powerful model. The results were significantly better:

- Accuracy: The online model correctly identified the teddy bear as a "toy" and detected smaller objects like the stacking rings, unlike SSD MobileNet V2, which mislabeled the teddy bear as "person" and missed the rings.
- Bounding Boxes: The boxes were more precise, tightly fitting around objects, whereas SSD MobileNet V2's boxes were slightly offset.
- Confidence Scores: The online model had higher confidence scores (e.g., 0.9 vs. 0.6 for the teddy bear).

Differences: The online model, likely trained on a larger dataset with more classes, handled the flyer's complexity better, detecting more objects and providing accurate labels. This highlights the limitations of SSD MobileNet V2's lightweight architecture and the class mismatch issue in this exercise.

Conclusion

This lab exercise provided a practical introduction to object detection using TensorFlow and the Pascal VOC 2007 dataset. I learned how to set up the environment, load and preprocess data, apply a pre-trained model, visualize results, and evaluate performance. Testing on the K-BIZ flyer revealed real-world challenges, such as class mismatch and difficulty with small objects. Answering the questions deepened my understanding of object detection concepts,

model trade-offs, and potential improvements. The bonus section expanded my perspective by comparing SSD MobileNet V2 to other models and testing a more powerful model, reinforcing the importance of balancing accuracy, speed, and resources in computer vision tasks.

Citations:

https://tfhub.dev/tensorflow/ssd_mobilenet_v2/2

https://towardsdatascience.com/understanding-object-detection-with-heatmaps-8f7e1e5e5e5e

https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html

https://tfhub.dev/s?module-type=image-object-detection

https://arxiv.org/abs/1911.09070

https://cloud.google.com/vision/docs/object-localizer