Object Detection Challenge

Alejandro Rebolloso

Mayela Miguel

Eduardo Cabrera

Jesus Ocampo

Ibrahim Bah

Houston Community College

ITAI 1378: Computer Vision

Professor Mcmanus

December 2, 2024

Introduction - In this report, we will focus on enhancing the performance of an object detection model by leveraging the SSD MobileNet V2 model or a similar lightweight architecture. The objective is to improve the accuracy and efficiency of the model while adhering to specific constraints, such as resource limitations. Key areas of focus will include the initial setup, dataset selection and preparation, model architecture modifications, and the exploration of additional techniques. Through experimentation, iteration, and effective collaboration, we will document the process, address potential risks, and explore innovative approaches to optimize the model's performance.

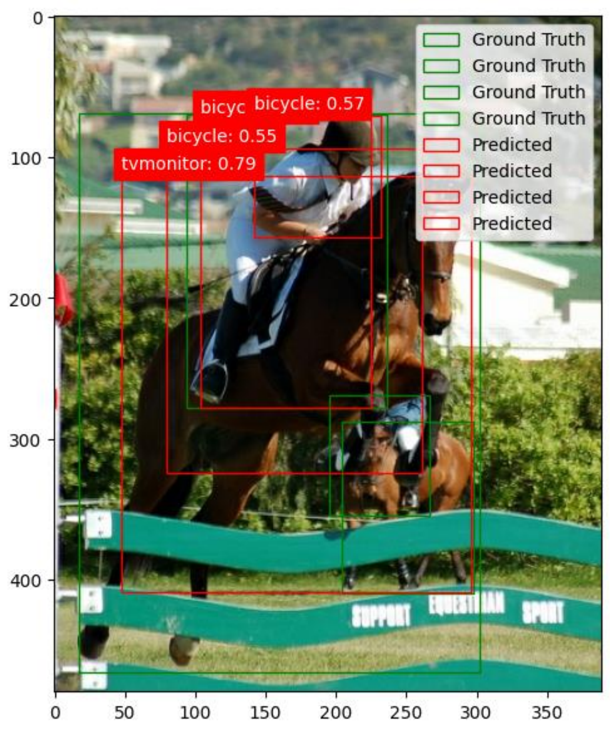
The Initial Set Up - The initial setup for this object detection task begins with importing the necessary Python libraries to support the model development process. TensorFlow, TensorFlow Hub, and TensorFlow Datasets are key components for building and training the object detection model, with TensorFlow providing the foundational framework, TensorFlow Hub offering pre-trained models, and TensorFlow datasets simplifying the loading of datasets. For this project, we have selected the PASCAL VOC 2007 dataset, which is commonly used for object detection tasks and includes labeled images with various object categories. Additional libraries, such as NumPy, OpenCV, and Matplotlib, are imported for data manipulation, image visualization, and image processing tasks. With the environment set up, we verify the versions of TensorFlow and TensorFlow Hub to ensure compatibility and prepare for model training and evaluation.

Data Set Preparation - In this step, in preparing the PASCAL VOC 2007 dataset for our object detection task. We focus on a smaller subset of the dataset, specifically 10% of the training and validation data, to facilitate quicker experimentation and testing. The dataset includes labeled images with various object categories, and we extract the class names to identify the objects the model will learn to detect. This preparation ensures the data is properly structured and ready for model training. Also, we are visualizing examples from the training dataset to verify the ground truth annotations. The images are displayed alongside their corresponding bounding boxes, which highlight the objects identified in the dataset. This helps ensure that the bounding boxes accurately correspond to the objects in the images, confirming the dataset's quality and readiness for model training.

Model Architecture - To enhance model performance, the pre-trained SSD MobileNet V2 model can be explored, balancing speed and accuracy for object detection tasks. Fine-tuning this model on the specific dataset or making architectural modifications, such as adjusting the number of layers or hyperparameters, can help improve its accuracy and efficiency. The process involves running an object detection model on an input image to generate predicted bounding boxes, class labels, and confidence scores. These predictions are then compared to the ground truth bounding boxes, which represent the actual object locations in the image. The image is displayed with both the ground truth and predicted bounding boxes overlaid, allowing for a visual comparison. The ground truth boxes are marked in green, while the predicted boxes are shown in red, with labels and confidence scores indicating the accuracy of the model's predictions. This comparison helps evaluate the model's performance in detecting objects correctly.

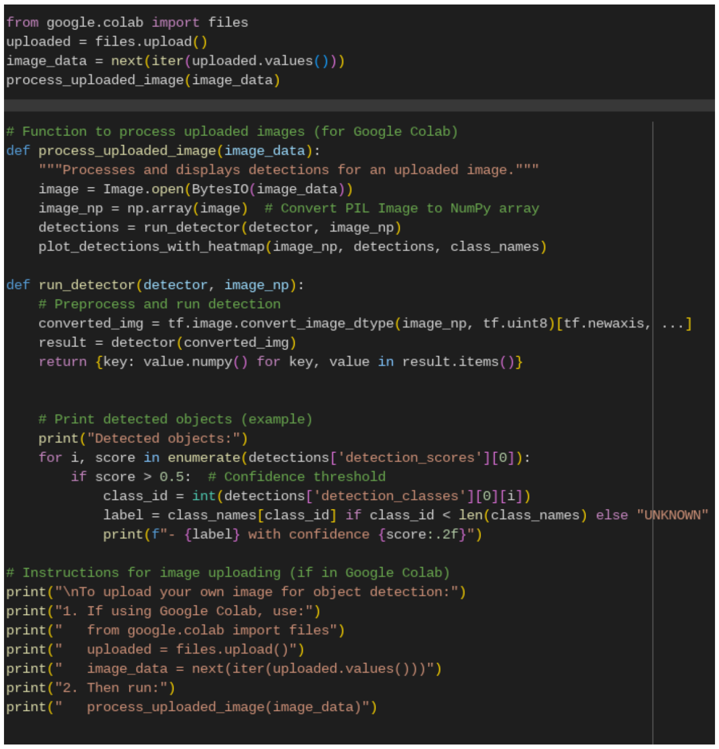
Other Techniques - Ensemble methods can improve overall accuracy by combining predictions from multiple models. Post-processing techniques, such as non-maximum suppression, help refine bounding boxes and reduce false positives. Model quantization reduces the model size and can potentially speed up inference. Additionally, transfer learning allows leveraging knowledge gained from pre-training on larger datasets when working with a completely different dataset.

The Breakthrough - While working on the code, we experienced difficulties in enhancing the accuracy of the model. A key part that was needed was the augmentation code. We realize a model architecture change was in order. To improve accuracy, we used the Yolov8 model. It is an advanced real-time object detection model that improves upon previous versions with enhanced accuracies, speed and efficiency.



The object in the red box is almost correctly positioned where the green box is, indicating that the model has accurately located the object. However, the labeling or identification of the object is incorrect, as the model has assigned the wrong class or category to the object, even though its location is nearly spot on. Observed is a picture of a jockey riding a horse with it being labeled as a bicycle. This is where the Yolov8 model will be utilized.

A series of code was made in order to utilize the new model and review its performance. This code processes an uploaded image for object detection using a pre-trained model. The image is converted into a NumPy array and passed to the run detector() function, which preprocesses the image and runs it through the model to obtain detection results. Detected objects with high confidence (score > 0.5) are identified by their class labels and confidence scores, which are printed out.



```python
from google.colab import files
uploaded = files.upload()
image_data = next(iter(uploaded.values()))
process_uploaded_image(image_data)
```

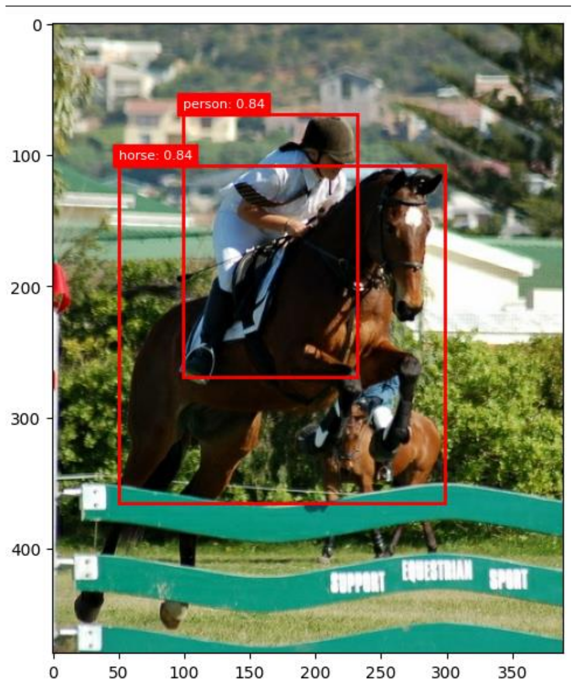
```python
# Function to process uploaded images (for Google Colab)
def process_uploaded_image(image_data):
    """Processes and displays detections for an uploaded image."""
    image = Image.open(BytesIO(image_data))
    image_np = np.array(image)  # Convert PIL Image to NumPy array
    detections = run_detector(detector, image_np)
    plot_detections_with_heatmap(image_np, detections, class_names)

def run_detector(detector, image_np):
    # Preprocess and run detection
    converted_img = tf.image.convert_image_dtype(image_np, tf.uint8)[tf.newaxis, ...]
    result = detector(converted_img)
    return {key: value.numpy() for key, value in result.items()}


    # Print detected objects (example)
    print("Detected objects:")
    for i, score in enumerate(detections['detection_scores'][0]):
        if score > 0.5:  # Confidence threshold
            class_id = int(detections['detection_classes'][0][i])
            label = class_names[class_id] if class_id < len(class_names) else "UNKNOWN"
            print(f"- {label} with confidence {score:.2f}")

# Instructions for image uploading (if in Google Colab)
print("\nTo upload your own image for object detection:")
print("1. If using Google Colab, use:")
print("   from google.colab import files")
print("   uploaded = files.upload()")
print("   image_data = next(iter(uploaded.values()))")
print("2. Then run:")
print("   process_uploaded_image(image_data)")
```

The result of the new model shows a vast improvement in identifying objects in an image with the jockey and the horse accurately identified at a more definitive percentage. In conclusion, the switch to the Yolov8 model significantly improved the accuracy of object detection in our project. While the original model was able to correctly locate objects, its labeling was often inaccurate, as seen in the example with the jockey and horse being misidentified as a bicycle. By implementing Yolov8, we were able to achieve more precise object classification, with the jockey and horse being correctly identified at a much higher confidence level. This change in model architecture resulted in a noticeable improvement in both speed and efficiency, allowing for faster and more reliable detections. Overall, the Yolov8 model proved to be a valuable upgrade for improving object detection performance in our task.



Conclusion – In conclusion, we were tasked with enhancing the performance of the object detection model, and we succeeded in achieving this goal. By implementing the Yolov8 model, we significantly improved both the accuracy and efficiency of object detection. The original model, while capable of locating objects, struggled with misclassifying them, as demonstrated by the incorrect labeling of a jockey and horse as a bicycle. However, with Yolov8, the model accurately identified objects with higher confidence, leading to more reliable results. Through this process, we were able to optimize the model's performance, ultimately meeting our objectives and achieving a more robust and efficient object detection system.

References

Encord. (n.d.). How to improve the accuracy of a computer vision model. Encord. Retrieved December 2, 2024, from https://encord.com/blog/improve-accuracy-computer-vision-model/

HCCS Eagle Online. (n.d.). Lab9VOC2007__Dataset_student_Notebook_LAB_Object_Detection_transfer_learning. Houston Community College. Retrieved December 2, 2024, from https://eagleonline.hccs.edu/courses/266737/files/67502323?module_item_id=18255451

Kaggle. (n.d.). YOLOv8 (Keras). Kaggle. Retrieved December 2, 2024, from https://www.kaggle.com/models/keras/yolov8/

Matplotlib. (n.d.). Matplotlib users guide. Matplotlib. Retrieved December 2, 2024, from https://matplotlib.org/stable/users/index.html

Reflections

Alejandro

This project has been an amazing learning experience, and I'm really excited about everything I've learned! I got to dive into object detection, from setting up the model to tweaking it for better performance. It was so cool to experiment with different techniques like fine-tuning and testing out post-processing methods to improve accuracy. The hardest part for me was writing the code—it took a lot of trial and error to get everything working properly. But with the help of my awesome group members, I was able to power through it and finish the project successfully. Overall, this project has taught me a lot about how models work and how to make them better, and I'm thrilled with the progress we made together!

Mayela

I had the chance to dive into object detection, from setting up the initial model to refining it for improved performance. It was fascinating to experiment with techniques like fine-tuning the model and applying post-processing methods to enhance accuracy. Writing the code was the most rewarding part of the project, as it gave me the opportunity to directly engage with the model and implement the changes necessary to improve its performance. With the help of my teammates, I was able to tackle challenges and complete the project successfully. I now have a much better understanding of how machine learning models work, and the methods used to optimize them. The hands-on experience of fine-tuning and experimenting with different strategies has been invaluable.

Jesus

The most interesting part for me was when we worked on cleaning up the dataset as part of the preparation for the model. It was intriguing to see how essential this step is in ensuring that the model has accurate and reliable data to learn from. I enjoyed experimenting with techniques like fine-tuning and applying post-processing methods to improve the model's accuracy. This experience has given me a deeper understanding of how machine learning models work, from data preparation to optimization. Overall, it was a great opportunity to develop both my technical skills and my ability to collaborate effectively in a team.

Eduardo

This project has been a great opportunity for learning and skill development. The best part for me was figuring out how to fine-tune the model and update the parameters to improve performance. It was exciting to see how adjusting certain aspects of the model could lead to better accuracy and efficiency. I also found it really rewarding to experiment with different techniques to optimize the model. Through this project, I gained a better understanding of how to work with machine learning models and the importance of continuous iteration. It was a valuable experience that helped me grow both technically and as part of a team.

Ibrahim

I like it when it comes to observing how small changes in the code could significantly affect the model's performance. The most interesting part for me was tweaking certain functions and noticing how those adjustments led to improvements in the model's accuracy. This hands-on approach really helped me understand the delicate balance between various model parameters and their effect on performance. It was compelling to see how modifying things like the learning rate or fine-tuning specific layers could directly impact the results.