Bounding Box Prediction and Object Detection: Internship Project Report

Georgi Vitanov

January 10, 2025

Abstract

This report details the implementation and analysis of a bounding box prediction and object detection project. The study leverages two pre-trained models—VGG16 and Fast R-CNN with ResNet50—alongside a custom lightweight convolutional neural network (CNN). With a focus on code modularity and extensibility, the project emphasizes usability and scalability over state-of-the-art performance. The dataset comprises 250 images, divided into training, validation, and testing sets. A hyperparameter tuning experiment was conducted for the learning rate using VGG16. The report also outlines potential future improvements, including advanced augmentation strategies, using state-of-the-art models, and including negative labels.

Introduction

This project aims to predict bounding boxes and identify objects within images using deep learning models. The modular code design enables easy replacement of components such as models, loss functions, and loggers, allowing seamless experimentation with different configurations.

The dataset comprises 250 labeled images, divided into training, validation, and testing sets. Pre-trained models, including VGG16 and Fast R-CNN with ResNet50, were chosen for their strong performance in object detection tasks. A custom CNN inspired by VGG16 was also developed to explore lightweight alternatives. Due to limited computational resources, most experiments were conducted on a local machine.

Methodology

Model Selection and Modification

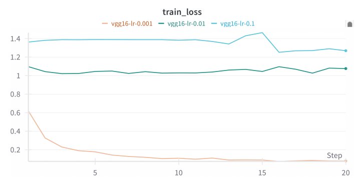
The project employs two pre-trained models and a custom CNN:

- **VGG16:** Modified to fit the dataset by replacing its classification head.
- **Fast R-CNN with ResNet50:** Known for robust object detection, similarly modified for compatibility with the dataset.
- **Custom CNN:** A lightweight model inspired by VGG16, designed to explore simpler architectures.

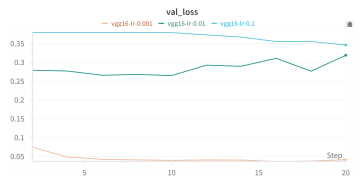
Hyperparameter Tuning

A learning rate hyperparameter tuning experiment was conducted using VGG16, testing three values: 0.1, 0.01, and 0.001. Results showed that lower learning rates significantly improved performance, with 0.001 yielding the best loss reduction.

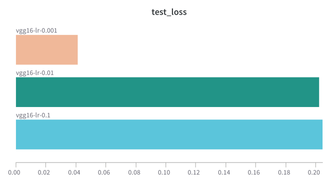
It was also observed that the models converge under 10 epochs with an appropriate learning rate. Therefore, the entire training was conducted for 20 epochs, with validation performed every 2 epochs. MSE was used as a loss.



(a) Train loss



(b) Validation loss



(c) Caption for Image 3

Figure 1: A row of three images.

Implementation Details

The code is written in a highly modular fashion:

- Configurations (e.g., models, datasets, loggers, and loss functions) are managed through a YAML file.
- Weights & Biases (WandB) is integrated for logging.
- The code supports distributed training, though this was not tested due to resource limitations.

The repository includes a demo notebook for running inference on new datasets and visualizing predictions. Alternatively, inference and evaluation can be performed via command-line scripts.

Future Improvements

While the project establishes a strong foundation, several enhancements could be implemented to improve performance and scalability:

1. Leveraging State-of-the-Art (SOTA) Models

Although VGG16 and Fast R-CNN were chosen for their reliability, newer models could achieve better results:

- **YOLO (You Only Look Once):** Renowned for real-time object detection capabilities, YOLO models could enable faster and more accurate predictions.
- **DETR (DEtection TRansformer):** A transformer-based approach with state-of-the-art performance on complex datasets.

2. Incorporating Negative Labels

The current dataset only contains positive labels (images with objects). Incorporating negative samples (images without any objects) could improve the model's ability to differentiate between true and false positives, increasing robustness in real-world scenarios.

3. Adding Noise to Bounding Boxes

Introducing controlled noise to bounding box labels during training could improve the model's tolerance to label inaccuracies, common in manually annotated datasets.

4. Advanced Augmentation Strategies

Exploring sophisticated data augmentation techniques could help improve model generalization. Examples include:

- Random distortions include hue, saturation, and brightness shifts.
- Geometric transformations, including rotations, scaling, and shearing.
- CutMix or MixUp, which combine parts of different images for augmented training samples.

5. TensorFlow Integration

The project is implemented in PyTorch due to its flexibility and extensive support for object detection frameworks. However, incorporating TensorFlow could broaden accessibility and leverage features such as TensorFlow Extended (TFX) for model deployment pipelines. Notably, PyTorch now supports exporting models to TensorFlow-compatible formats, making this integration feasible.

6. Expanding Hyperparameter Tuning

While this project focuses on learning rate tuning, extending the search to other hyperparameters such as batch size, optimizer choice, and weight initialization methods could yield better performance. Tools like Optuna or Ray Tune could automate this process for large-scale experiments.

Conclusion

This project successfully demonstrates bounding box prediction and object detection using pre-trained models and a custom CNN. While the focus was on creating modular and maintainable code, the results highlight the potential for further improvements. Future work should explore state-of-the-art models, advanced augmentation techniques, and the integration of TensorFlow for broader applicability. With additional computational resources, hyperparameter tuning and distributed training could be leveraged to achieve state-of-the-art performance.