**Lego Detection**

**A Comparison Analysis between Faster RCNN and Yolo**

This is a brief project performance report for CS5330 Lab 3.

**Methods**

***Faster RCNN – ResNet50 Using PyTorch***

We decided to first try with ResNet50-FasterRCNN because it is a model that we are quite familiar with (lab 2, deep learning talk), but previously we have mostly benefited from pre-trained ResNet50 model, and this time we did try to have it natively trained and see how it works.

The general ***pipeline*** can be summed up as: A dataset of total 500 images (350 training, 75 validating, 75 testing), all .xml files are re-labelled to “lego” class, but still in the pascal VOC format, only 5 Epochs and 4 Batch size, a 0.001 learning rate, used Adam optimizer and was on purposed trained on local CPU not GPU because Macs has not the Nvidia graphic card so CUBA was not an available choice, confidence level was set to 0.5 initially.

It is a genuinely under-sized dataset, but the training process was still painfully slow, and the results were really upsetting. It certainly due to many and many reasons, we designed the training process to run for 5 epochs. Given that Faster R-CNN models generally require a longer training period to reach convergence (typically 20-50 epochs), our setup is likely limited by this lower epoch count. The batch size was set to 4 due to memory constraints, which limits batch-level learning and prevents the model from fully leveraging batch normalization techniques for stable gradient flows. The learning rate was initially set to 0.001, optimized through the Adam optimizer. Adam was selected for its faster convergence in environments with limited hardware resources, particularly because this model was trained locally on a CPU rather than on a GPU due to the lack of an NVIDIA-compatible GPU for CUDA support. This decision was influenced by hardware constraints, as my Mac lacks an NVIDIA GPU, making CPU-based training a practical choice despite its reduced speed and efficiency.

The Faster R-CNN framework leverages ResNet50 as a backbone for feature extraction, which is particularly suited for capturing fine-grained patterns from object textures and edges. However, one challenge in training from scratch is ensuring that the anchor boxes (regions where the model predicts object locations) are optimized to match the dimensions and aspect ratios of LEGO pieces. Since this project used the default Faster R-CNN anchor box settings, this may introduce challenges with detecting varying LEGO sizes. Additionally, the model is configured with a binary classification approach (background vs. LEGO), simplifying the task but potentially reducing the ability to discern between different LEGO types based on shape or size. The confidence threshold was set at 0.5 to control for detection sensitivity, a standard starting point which balances between capturing true positives and minimizing false positives.

**Results and Discussion**

***Faster RCNN – ResNet50 Using PyTorch***

With only 5 epochs, the model’s learning was limited, which was immediately apparent in its inability to generalize well on the test set. Object detection models generally require around 20-50 epochs for effective learning, but the combination of CPU-only training and a small dataset size (500 images) constrained the training potential. The limited batch size of 4 further hindered the model’s exposure to diverse feature patterns within each batch, which is important for stabilizing learning. Faster R-CNN’s batch normalization layers, which rely on larger batch sizes to compute stable statistics, thus underperformed under these conditions. This led to inconsistent detection results, particularly with smaller or overlapping LEGO pieces, as the model struggled to form a stable understanding of the “lego” class with such limited exposure.

Simplifying all LEGO pieces to a single class was a practical approach to streamline training but had notable drawbacks in model generalization. Since LEGO pieces vary significantly in color, shape, and size, merging them into one “lego” class likely introduced intra-class variations that were difficult for the model to reconcile. Consequently, the model frequently misclassified parts of the background as LEGO pieces, as it failed to capture distinguishing features of specific LEGO parts. The lack of class specificity also limits the model’s application, as it cannot differentiate between various LEGO components, impacting usability for tasks requiring finer classification.

Faster R-CNN’s anchor boxes are typically optimized for general object detection scenarios, but they lack specialization for the unique aspect ratios and dimensions of LEGO pieces. Consequently, the default anchor box settings struggled with accurately locating LEGO parts, especially those with non-standard shapes or dimensions. This was compounded by the fixed input image size of 600x600 pixels, which, while simplifying processing, may have altered the original aspect ratios of LEGO pieces, making it harder for the model to detect features accurately. Anchor box optimization, particularly for varying LEGO shapes, would likely enhance localization and detection confidence.

**Conclusion**