LEGO Piece Detection using YOLOv5

Author: Zheng Gu

Methods

Research Question

This project addresses the challenge of detecting and counting LEGO pieces in images. The primary goal was to develop a deep learning model capable of:

1. Identifying LEGO pieces in various arrangements
2. Drawing accurate bounding boxes around each piece
3. Providing a total count of LEGO pieces in an image

The problem is formulated as an object detection task, where the model must learn to localize LEGO pieces regardless of their color, shape, or orientation.

Dataset

The original dataset consisted of 168,000 synthetic images with annotations in PASCAL VOC format, featuring 600 unique LEGO parts. For this project, I created smaller, more manageable subsets:

- A 500-image dataset for local development and quick prototyping
- A 5,000-image dataset for comprehensive training on Northeastern's computing cluster

For each subset, I performed the following preprocessing steps:

1. **Data validation**: Implemented a script and manual checks for random pictures to ensure annotation quality
2. **Data splitting**: Divided the data into training (~70%), validation (~15%), and testing (~15%) sets
3. **Annotation conversion**: Transformed PASCAL VOC format annotations to YOLO format
4. **Class simplification**: Consolidated all 600 LEGO piece classes into a single "lego" class

The `prepare_data.py` script handles this entire preprocessing pipeline, ensuring data quality and proper formatting for YOLO training.

Neural Network Architecture

I selected the YOLOv5 (You Only Look Once) architecture for this project due to its excellent balance of speed and accuracy in object detection tasks. YOLOv5 processes the entire image in a single forward pass, making it highly efficient for real-time applications.

For the experimental setup, I trained two different YOLOv5 variants:

1. **YOLOv5n** (nano): A lightweight model with 2.5M parameters, trained on the 500-image dataset for local testing
2. **YOLOv5s** (small): A slightly larger model with 9.1M parameters, trained on the 5,000-image dataset using Northeastern's computing cluster

The models were configured with the following parameters:

- **Input size**: 640×640 pixels
- **Batch size**: 16
- **Learning rate**: Auto-determined by the YOLOv5 framework
- **Confidence threshold**: 0.25 (for inference)
- **IoU threshold**: 0.5 (for evaluation)

The training process implemented by `train.py` leverages transfer learning by starting with pre-trained weights and fine-tuning on our LEGO dataset.

Evaluation Metrics

To assess model performance, I used mean Average Precision (mAP) at an IoU threshold of 0.5, which is a standard metric for object detection tasks. The mAP metric quantifies both localization accuracy and classification confidence by computing the area under the precision-recall curve.

The mAP@0.5 is calculated as follows:

1. For each image, the model produces bounding box predictions with confidence scores
2. Predictions are compared to ground truth using IoU (Intersection over Union):

   IoU = (Area of Overlap) / (Area of Union)
3. A prediction is considered a true positive if IoU ≥ 0.5 and the confidence exceeds the threshold
4. Precision (P) and Recall (R) are calculated at different confidence thresholds:

   P = TP / (TP + FP) R = TP / (TP + FN)

   where TP = true positives, FP = false positives, FN = false negatives
5. The precision-recall curve is plotted and the area under this curve gives the Average Precision (AP)
6. mAP@0.5 is the mean of the AP values across all classes (in our case, just the single "lego" class)

Additionally, I monitored:

- **Precision**: The ratio of true positive detections to all positive detections
- **Recall**: The ratio of true positive detections to all actual objects

The `evaluate.py` script handles the calculation of these metrics on the validation set.

Results and Discussion

Model Performance

Both models demonstrated strong performance in detecting LEGO pieces, with the larger dataset yielding better results as expected:

YOLOv5n (500-image dataset):

- **mAP@0.5**: 0.933
- **mAP@0.5-0.95**: 0.796
- **Precision**: 0.894
- **Recall**: 0.900

YOLOv5s (5000-image dataset):

- **mAP@0.5**: 0.980
- **mAP@0.5-0.95**: 0.938
- **Precision**: 0.978
- **Recall**: 0.957

The results demonstrate that:

1. Even the smaller YOLOv5n model achieved excellent performance with mAP@0.5 of 0.933
2. The larger YOLOv5s model with more training data provided near-perfect detection with mAP@0.5 of 0.980
3. The precision-recall balance is excellent for both models, suggesting robust detection capabilities

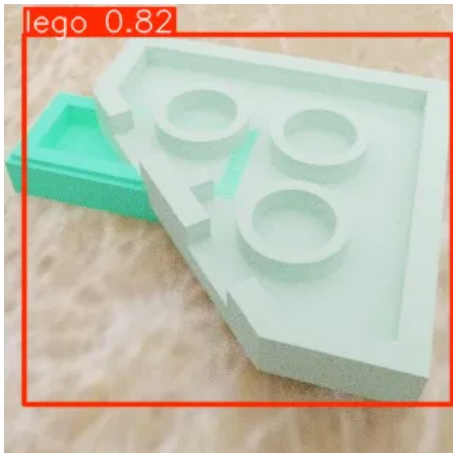
The interactive demo created with Gradio (implemented in `app.py`) provides a user-friendly interface to visualize these results, allowing for adjustment of the confidence threshold and testing on new images.

Limitations

Despite the impressive performance, several limitations should be acknowledged:

1. **Computation requirements**: The larger model demands significantly more computational power. On my laptop, the YOLOv5s model's inference time was over three times slower than the lightweight YOLOv5n model, even running on cluster. This presents a real trade-off between detection accuracy and processing speed that must be considered for any practical application.

2. **Overlapping LEGO detection**: While testing my model on various examples, I noticed it sometimes struggles with overlapping LEGO pieces. When bricks are stacked or partially covering each other, the model occasionally merges them into a single detection or misses the partially hidden pieces altogether. This isn't surprising since the YOLOv5 architecture uses Non-Maximum Suppression to eliminate duplicate detections, which can inadvertently merge overlapping objects. The synthetic training data also might not include enough examples of complex overlapping arrangements.



3. **Synthetic-to-real domain gap**: Since the model was trained exclusively on synthetic images, its performance might degrade when applied to real-world photographs of LEGO pieces with variable lighting conditions, backgrounds, and camera angles. This potential domain shift could affect the model's generalization ability.

Conclusion

My key findings include:

1. YOLOv5 architectures are highly effective for LEGO piece detection, with the best model achieving 98% mAP@0.5
2. Larger datasets and models provide incremental improvements in detection accuracy, at the cost of increased computational requirements
3. Even a small dataset of 500 images is sufficient to train a model with strong performance (93.3% mAP@0.5)

The benefits of this work include the ability to quickly and accurately count LEGO pieces in complex arrangements, which could be useful for inventory management, automated sorting systems, or assisting in LEGO construction verification.

The main shortcomings are the computational demands of the larger model and the challenges with detecting overlapping pieces, which limit certain real-world applications.

Future Research Directions

Several promising areas for future research include:

1. **Real-world domain adaptation**: Investigating methods to adapt the model trained on synthetic data to perform well on real-world photographs, possibly using techniques like domain randomization or adversarial training.

2. **Multi-class detection**: Extending the model to identify specific LEGO piece types, colors, and orientations would enable more advanced applications like automated building instruction validation.

3. **Lightweight model optimization**: Exploring model compression techniques such as pruning, quantization, and knowledge distillation to make the high-performing model more suitable for edge devices and real-time applications.

References

[1] J. Redmon, Yolo: Real-time object detection, https://pjreddie.com/darknet/yolo/ (accessed Mar. 7, 2025).
[2] Anthropic, "Claude," Large language model, 2023. [Online]. Available: https://www.anthropic.com/
[3] J. Redmon, Yolo: Real-time object detection, https://pjreddie.com/darknet/yolo/ (accessed Mar. 7, 2025).
[4] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, “The Pascal Visual Object Classes (VOC) challenge,” International Journal of Computer Vision, vol. 88, no. 2, pp. 303–338, Sep. 2009. doi:10.1007/s11263-009-0275-4
[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779–788, Jun. 2016. doi:10.1109/cvpr.2016.91
[6] Ultralytics, Ultralytics YOLO Docs, https://docs.ultralytics.com/ (accessed Mar. 7, 2025).