Object Detection From Scratch

Author's Note on Scope and Experience

This project represents my first attempt at building an object detection model entirely from scratch. The model architecture, training pipeline, and evaluation methodology were designed and implemented as a learning exercise to understand the fundamental components of modern object detection systems.

While the implementation is functional and achieves reasonable performance, some design choices are simplified and would benefit from further experimentation and refinement. The results presented here should be interpreted in that context.

1 Introduction

Object detection is a core problem in computer vision, involving both localization and classification of objects within an image. Modern detectors often rely on large pretrained backbones and extensive datasets. In this project, a simplified object detection system was implemented from scratch without using pretrained weights, with the primary goal of understanding detection fundamentals rather than maximizing benchmark performance.

The task considered is detection of simple geometric shapes in synthetic images.

2 Task Description

The detector is trained to identify the following five object classes:

- Circle
- Rectangle
- Triangle
- Star
- Pentagon

Each image contains between one and three objects, and the model predicts bounding boxes, objectness confidence, and class probabilities.

3 Dataset

Due to practical constraints in downloading large public datasets, a synthetic dataset was generated programmatically.

3.1 Dataset Generation

A custom script generates random images containing geometric shapes. For each image, the script records bounding box coordinates and class labels in JSON format.

3.2 Dataset Split

- Training set: 800 images
- Validation set: 200 images
- Test set: 200 images

3.3 Data Augmentation

The following augmentations were applied during training:

- Horizontal flips (50% probability)
- Random brightness adjustment in the range $[-30, +30]$

4 Model Architecture

The model design is inspired by YOLO-style detectors but simplified to reduce complexity.

4.1 Backbone Network

The backbone consists of five convolutional blocks:

- Convolution layers with batch normalization and Leaky ReLU activation
- Max pooling after each block for spatial downsampling

```
Input: 224 × 224 × 3
→ Conv Block 1 (32 channels) → MaxPool
→ Conv Block 2 (64 channels) → MaxPool
→ Conv Block 3 (128 channels) → MaxPool
→ Conv Block 4 (256 channels) → MaxPool
→ Conv Block 5 (512 → 256 channels)
Output: 14 × 14 × 256 feature map
```

4.2 Detection Head

The detection head operates on the final feature map:

- Grid size: 14×14
- Number of anchors per grid cell: 3
- Anchor sizes: 10×10, 25×25, 50×50

Each anchor predicts:

$$(x, y, w, h, c, p_1, \ldots, p_5)$$

where c is the objectness confidence and p_i are class probabilities.

4.3 Model Size

- Total parameters: 7,230,462
- Model size on disk: 27.6 MB

5 Training Procedure

5.1 Training Configuration

- Optimizer: Adam
- Learning rate: 0.001
- Batch size: 16
- Epochs: 30
- Learning rate schedule: Step decay (halved every 10 epochs)

5.2 Loss Function

The total loss is a weighted sum of three components:

- Coordinate loss (MSE on x, y, w, h)
- Confidence loss (binary cross-entropy)
- Classification loss (categorical cross-entropy)

The loss is defined as:

$$\mathcal{L} = \lambda_{\text{coord}}\mathcal{L}_{\text{coord}} + \mathcal{L}_{\text{obj}} + \lambda_{\text{noobj}}\mathcal{L}_{\text{noobj}} + \mathcal{L}_{\text{cls}}$$

with:

$$\lambda_{\text{coord}} = 5.0, \quad \lambda_{\text{noobj}} = 0.5$$

Training time was approximately 2.5 minutes on a GPU.

6 Results

6.1 Detection Performance

Class	**Average Precision (%)**
Circle	39.10
Rectangle	65.30
Triangle	83.23
Star	67.43
Pentagon	62.53
mAP@0.5	**63.52**

Triangles achieved the highest accuracy, likely due to their distinctive geometry. Circles were the most challenging, as bounding boxes poorly match circular shapes.

6.2 Inference Speed

- GPU inference speed: 802.8 FPS

This speed is aided by the small input resolution and lightweight model. CPU performance is expected to be significantly lower.

7 Accuracy–Speed Tradeoff

Aspect	This Model	Larger Detectors (e.g., YOLOv5)
Speed	Very High (>800 FPS)	Moderate (100–200 FPS)
Accuracy	Moderate (63% mAP)	High (80–90% mAP)
Model Size	Small (27 MB)	Large (>100 MB)

This highlights the inherent tradeoff between accuracy and inference speed in object detection systems.

8 Future Improvements

Potential improvements include:

1. Training on real-world images instead of synthetic data
2. Data-driven anchor box selection
3. Additional data augmentation (rotation, scaling)
4. Use of focal loss to address class imbalance
5. Deeper backbone with residual connections

9 Project Structure

```
object_detection/
 data/
    train/
    val/
    test/
 checkpoints/
 outputs/
    detections/
    detection_results.gif
    training_log.txt
    evaluation_results.txt
 dataset.py
 model.py
 train.py
 evaluate.py
 visualize.py
 Report.md
```

10 Usage

```
# Train the model
python train.py

# Evaluate performance
```

```
python evaluate.py

# Generate visualizations
python visualize.py

# Run full pipeline
python run_all.py
```

11 Conclusion

This project demonstrates a complete object detection pipeline implemented from scratch. While the detector is intentionally simple, it provides practical insight into anchor-based detection, loss design, mAP evaluation, and the accuracy–speed tradeoff inherent in real-world systems.