# Lab 3: Faster R-CNN final-report

CHI YEONG HEO[1],
[1]School of Computer Science and Engineering, Pusan National University, Busan 46241 Republic of Korea

## 1. Introduction

This report presents an analysis of the implementation of Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks[2], training on the Recycle Trash Dataset[1] created by NAVER Connect Foundation.

## 2. Methodology

### 2.1. Dataset

The Recycle Trash Dataset, developed and made publicly available by NAVER Connect Foundation, comprises a diverse collection of indoor and outdoor waste images. This dataset includes 21,818 1024 × 1024 images annotated for 107,935 objects, with each object labeled with bounding boxes and segmentation data. The full dataset is approximately 120 GB in compressed format, and all annotations are provided in the COCO format for ease of use in object detection frameworks.

For this implementation, the dataset has been divided into training and testing sets, consisting of 4,883 and 4,871 images, respectively. The dataset includes 10 distinct classes, defined as follows:

- 0: General waste
- 1: Paper
- 2: Paper pack
- 3: Metal
- 4: Glass
- 5: Plastic
- 6: Styrofoam
- 7: Plastic bag
- 8: Battery
- 9: Clothing

### 2.2. Data Augmentation

Data augmentation is a technique employed to increase the effective size and variability of a training dataset by applying random transformations to the input data. This approach enhances the model's generalization capabilities by exposing it to a broader range of representations of the original images, which helps mitigate overfitting and improves performance on unseen data.

For this implementation, the original image dimensions (1024 × 1024 pixels) are maintained without resizing. Pixel values are normalized to the range [0, 1] prior to applying data augmentation.

#### 2.2.1. Random Flipping

Random Flipping is used to introduce orientation variability by mirroring images with a specified probability, allowing the model to learn orientation-invariant features. This augmentation step contributes to improved performance on unseen test data by enhancing robustness to different orientations.

In this implementation, Random Flipping is applied with a probability of 0.5. The images may be flipped horizontally, vertically, or both horizontally and vertically.

### 2.3. Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is an iterative optimization algorithm extensively utilized for training machine learning models, particularly neural networks. The fundamental objective of SGD is to minimize a given loss function by iteratively updating the model's parameters in the direction of the negative gradient, thereby approaching the local or global minimum. Unlike traditional Gradient Descent, which evaluates the gradient using the entire dataset, SGD performs updates based on a randomly selected subset, or mini-batch, of the data. This stochastic process introduces noise into the optimization trajectory, which can facilitate escaping local minima and accelerate convergence.

A variant of SGD employed incorporates momentum, a technique that enhances the convergence properties of the standard SGD algorithm. Momentum is designed to accelerate SGD by adding a fraction of the previous update vector to the current update, thereby damping oscillations and stabilizing the optimization path. The PyTorch library implements Nesterov momentum based on the formula from [3].

**Algorithm 1:** Stochastic Gradient Descent with Momentum and Nesterov

**Input:** $\gamma$ (lr), $\theta_0$ (params), $f(\theta)$ (objective), $\lambda$ (weight decay), $\mu$ (momentum), $\tau$ (dampening), *nesterov*, *maximize*

**for** $t = 1$ *to* ... **do**
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  **if** $\lambda \neq 0$ **then**
    $g_t \leftarrow g_t + \lambda\,\theta_{t-1}$
  **if** $\mu \neq 0$ **then**
    **if** $t > 1$ **then**
      $b_t \leftarrow \mu b_{t-1} + (1-\tau) g_t$
    **else**
      $b_t \leftarrow g_t$
    **if** *nesterov* **then**
      $g_t \leftarrow g_t + \mu b_t$
    **else**
      $g_t \leftarrow b_t$
  **if** *maximize* **then**
    $\theta_t \leftarrow \theta_{t-1} + \gamma g_t$
  **else**
    $\theta_t \leftarrow \theta_{t-1} - \gamma g_t$
**return** $\theta_t$

### 2.4. Rectified Linear Unit (ReLU)

The Rectified Linear Unit (ReLU) is an extensively adopted activation function within the domain of neural networks, particularly noted for its simplicity and efficacy in deep learning models. Formally, ReLU is defined by the piecewise linear function $f(x) = \max(0, x)$, where $x$ denotes the input to a given neuron. Consequently, the ReLU function yields an output of zero for any negative input, while preserving the input value for non-negative entries as a linear identity.

## 3. Implementation Details

### 3.1. Faster R-CNN Model Architecture

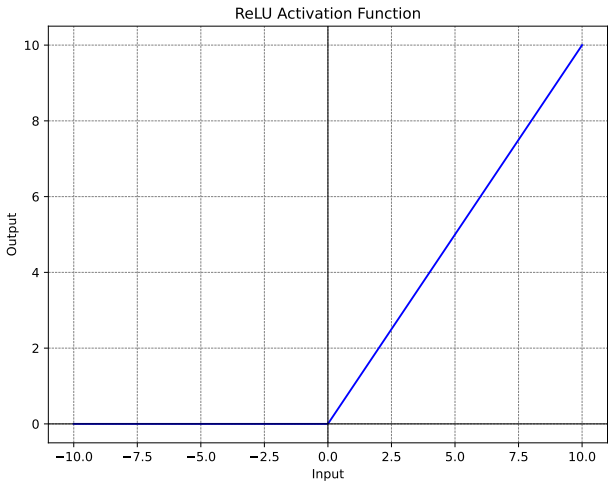


Figure 1: Rectified Linear Unit

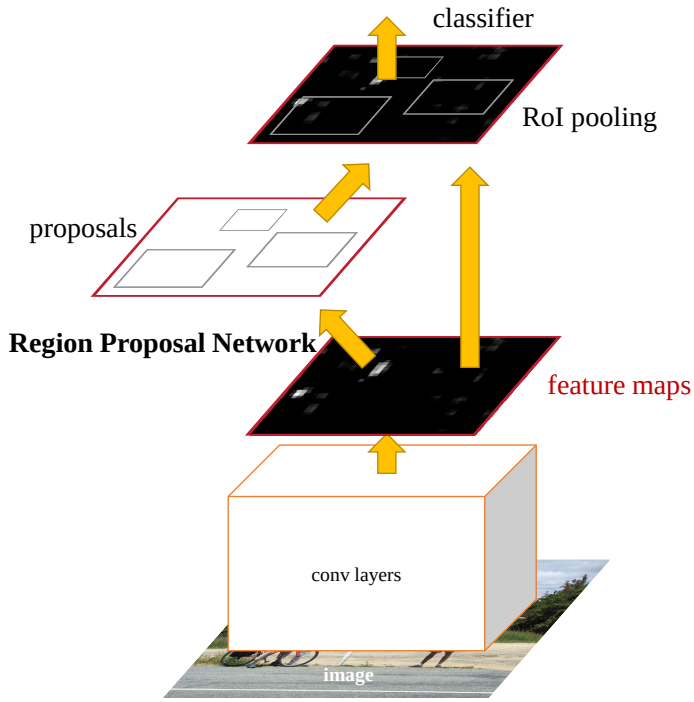


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

The implemented Faster R-CNN model architecture consists of three primary components (Fig 2). The initial component is a feature extractor that processes input images to generate feature maps. These feature maps are then passed to the Region Proposal Network (RPN), which produces a set of candidate regions, known as Regions of Interest (ROIs), likely containing objects. The final component, the ROI head, takes the ROIs and feature maps as input to predict object localization $(x, y, w, h)$ and an objectness score for each candidate region.

#### 3.1.1. VGG-16

The VGG-16 network architecture is utilized within Faster R-CNN, serving as both the feature extractor and classifier. The original convolutional configurations of VGG-16 are presented in Table 1. For this implementation, configuration "D" is employed, as it is the default setting of the 'vgg16' module in the torchvision library. The VGG-16 network is segmented into a feature extractor and classifier, as outlined in Table 2.

#### 3.1.2. Region Proposal Network

The Region Proposal Network (RPN) is a key component in Faster R-CNN, responsible for generating potential bounding boxes or ROIs that may contain objects. The RPN is designed to efficiently produce a dense set of anchors across an image, scoring each anchor's likelihood of containing an object and predicting bounding box offsets to refine the anchor positions.

The implemented RPN consists of a feature extraction layer, followed by two convolutional layers for objectness scores and bounding box regression. Anchors are generated using different aspect ratios and scales, covering a variety of object shapes and sizes across the image. The network architecture for the RPN is detailed in Table **??**.

The RPN takes as input the feature map generated by the backbone network. The feature map is first processed by a $3 \times 3$ convolutional layer, producing intermediate features that capture local context. Next, a $1 \times 1$ convolutional layer with two channels per anchor outputs objectness scores (foreground/background) for each anchor, and another $1 \times 1$ convolutional layer with four channels per anchor outputs bounding box offsets for refining anchor positions. Refer to Fig 3. Anchors are shifted across all feature map locations, providing a comprehensive set of potential proposals.

In this implementation, nine anchor boxes are generated per pixel location, with anchor ratios of 0.5, 1, and 2, and anchor scales of 8, 16, and 32. These anchors ensure coverage of different object shapes and sizes within the image. A proposal layer, defined by the `ProposalCreator` class, then filters and ranks the anchors, selecting the top candidate regions to pass to the next stage of the Faster R-CNN network for further classification and refinement.

#### 3.1.3. ROI Head

The Region of Interest (ROI) Head in Faster R-CNN performs final classification and bounding box regression on the proposals generated by the RPN. This component is critical for accurately categorizing each proposal as either a specific object class or background and refining the bounding box coordinates for precise localization.

The implemented ROI Head consists of an ROI Pooling layer, which resizes the ROIs to a fixed spatial size, followed by fully connected layers (classifier) that produce the final predictions. The architecture details of the ROI Head are presented in Table 4.

The ROI Head first processes each ROI through an ROI Pooling layer, which adjusts the dimensions of each proposal region to a fixed size (defined by `roi_size`). This operation ensures that proposals of varying sizes can be fed into the same fully connected layers. After pooling, the flattened features pass through a series of fully connected layers, representing a classifier pre-trained on VGG-16. The output is then divided into two branches: a bounding box regressor (`cls_loc`) and a classifier (`score`). The bounding box regressor outputs adjustments to the bounding box coordinates, while the classifier outputs the probability scores for each class, including the background class.

The ROI Head uses VGG-16's pre-trained fully connected layers (FC1 and FC2) as a base classifier. In this implementation, the bounding box regressor (`cls_loc`) predicts four values for each class representing the bounding box offsets, while the classifier (`score`) outputs the probability of each class for the ROI. Both output branches are initialized with a normal distribution with mean is 0 and standard deviation is 0.001 for `cls_loc` and 0.01 for `score`.

Overall, the ROI Head refines and classifies the proposals with high precision, enabling Faster R-CNN to deliver accurate object detections.

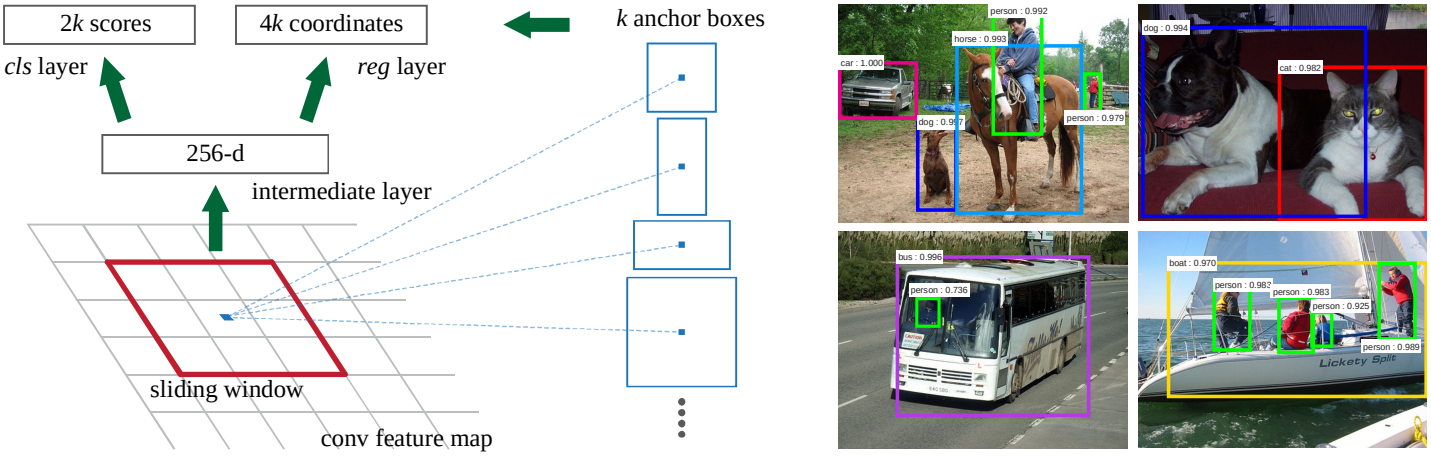


Figure 3: **Left**: Region Proposal Network (RPN). **Right**: Example detections using RPN proposals on PASCAL VOC 2007 test. Our method detects objects in a wide range of scales and aspect ratios.

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224 × 224 RGB image) | | | | | |
| conv3-64 | conv3-64 **LRN** | conv3-64 **conv3-64** | conv3-64 conv3-64 | conv3-64 conv3-64 | conv3-64 conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128 **conv3-128** | conv3-128 conv3-128 | conv3-128 conv3-128 | conv3-128 conv3-128 |
| maxpool | | | | | |
| conv3-256 conv3-256 | conv3-256 conv3-256 | conv3-256 conv3-256 | conv3-256 conv3-256 **conv1-256** | conv3-256 conv3-256 **conv3-256** | conv3-256 conv3-256 **conv3-256** |
| maxpool | | | | | |
| conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 **conv1-512** | conv3-512 conv3-512 **conv3-512** | conv3-512 conv3-512 **conv3-512** |
| maxpool | | | | | |
| conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 **conv1-512** | conv3-512 conv3-512 **conv3-512** | conv3-512 conv3-512 **conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

Table 2: decomposed VGG-16 Architecture

| | layer name | output size | configuration |
|---|---|---|---|
| Feature Extractor | conv1_1 | 32x32 | ( 3x3, 64 ) |
| | relu | 32x32 | - |
| | conv1_2 | 32x32 | ( 3x3, 64 ) |
| | relu | 32x32 | - |
| | maxpool | 16x16 | ( 2x2, stride=2) |
| | conv2_1 | 16x16 | ( 3x3, 128 ) |
| | relu | 16x16 | - |
| | conv2_2 | 16x16 | ( 3x3, 128 ) |
| | relu | 16x16 | - |
| | maxpool | 8x8 | ( 2x2, stride=2) |
| | conv3_1 | 8x8 | ( 3x3, 256 ) |
| | relu | 8x8 | - |
| | conv3_2 | 8x8 | ( 3x3, 256 ) |
| | relu | 8x8 | - |
| | conv3_3 | 8x8 | ( 3x3, 256 ) |
| | relu | 8x8 | - |
| | maxpool | 4x4 | ( 2x2, stride=2) |
| | conv4_1 | 4x4 | ( 3x3, 512 ) |
| | relu | 4x4 | - |
| | conv4_2 | 4x4 | ( 3x3, 512 ) |
| | relu | 4x4 | - |
| | conv4_3 | 4x4 | ( 3x3, 512 ) |
| | relu | 4x4 | - |
| | maxpool | 2x2 | ( 2x2, stride=2) |
| | conv5_1 | 2x2 | ( 3x3, 512 ) |
| | relu | 2x2 | - |
| | conv5_2 | 2x2 | ( 3x3, 512 ) |
| | relu | 2x2 | - |
| | conv5_3 | 2x2 | ( 3x3, 512 ) |
| | relu | 2x2 | - |
| | maxpool | 1x1 | ( 2x2, stride=2) |
| Classifier | fc1 | 4096 | |
| | relu | 4096 | |
| | fc2 | 4096 | |
| | relu | 4096 | |

### 3.2. Loss Function

The Faster R-CNN training process optimizes a multi-task loss function that combines both Region Proposal Network (RPN) and Fast R-CNN losses. The total loss function is defined as follows:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i,p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i,t_i^*). \quad (1)$$

As shown in the implementation, the total loss consists of four components:

Table 3: Region Proposal Network (RPN) architecture

| Layer | Filter Size | Output Channels |
|---|---|---|
| `Conv1` | $3 \times 3$ | 512 |
| `Score` | $1 \times 1$ | $2 \times$ num_anchors |
| `Loc` | $1 \times 1$ | $4 \times$ num_anchors |

Table 4: Region of Interest (ROI) Head architecture

| Layer | Input Size | Output Size |
|---|---|---|
| `RoIPool` | Variable | $7 \times 7 \times 512$ |
| `FC1` | $7 \times 7 \times 512$ | 4096 |
| `FC2` | 4096 | 4096 |
| `cls_loc` | 4096 | $n\_class \times 4$ |
| `score` | 4096 | $n\_class$ |

- RPN localization loss ($L_{\text{rpn\_loc}}$)
- RPN classification loss ($L_{\text{rpn\_cls}}$)
- ROI localization loss ($L_{\text{roi\_loc}}$)
- ROI classification loss ($L_{\text{roi\_cls}}$)

For both RPN and ROI localization losses, the model employs a smooth L1 loss function with configurable $\sigma$ parameters (`rpn_sigma` and `roi_sigma`). The smooth L1 loss is implemented in the `_smooth_l1_loss` function:

$$L_{\text{smooth\_L1}}(x,t) = \sum_i \begin{cases} \frac{\sigma^2}{2}(w_i(x_i - t_i))^2 & \text{if } |w_i(x_i - t_i)| < \frac{1}{\sigma^2} \\ |w_i(x_i - t_i)| - \frac{0.5}{\sigma^2} & \text{otherwise} \end{cases} \quad (2)$$

For classification losses, the model uses standard cross-entropy loss.

The localization losses are only computed for positive examples (where $p_i^* > 0$), as implemented in the `_fast_rcnn_loc_loss` function. This is achieved by applying an `in_weight` mask that zeros out the loss for negative examples. The loss is then normalized by the number of positive examples.

The final loss is computed as the sum of all four components:

$$L_{\text{total}} = L_{\text{rpn\_loc}} + L_{\text{rpn\_cls}} + L_{\text{roi\_loc}} + L_{\text{roi\_cls}} \quad (3)$$

This combined loss is then used to update both the RPN and detection network parameters through backpropagation in the `train_step` method.

## 4. Results

### 4.1. Inference Visualization

To evaluate the qualitative performance of our trained model, I conducted a visual analysis of detection results on a subset of test images. The model demonstrates its object detection capabilities by generating class-specific bounding box predictions accompanied by confidence scores. The visualization reveals that while the model successfully identifies and localizes objects across various scenes, there are minor spatial misalignments between some predicted bounding boxes and their target objects. These slight localization discrepancies provide insights into potential areas for model refinement, particularly in terms of bounding box regression accuracy.

## 5. Conclusion

This report presented a detailed implementation and analysis of the Faster R-CNN object detection framework. Our implementation follows the two-stage detection paradigm, comprising a Region Proposal Network and a detection network, unified within a single deep neural architecture.

Our qualitative analysis through visualization demonstrates the model's capability to detect and localize objects across diverse scenarios. While the model shows promising performance in object detection tasks, there remain opportunities for improvement, particularly in the precision of bounding box localization. The slight misalignments observed in some predictions suggest potential areas for future optimization, such as fine-tuning the bounding box regression parameters or exploring alternative anchor configurations.

Future work could focus on several aspects to enhance the current implementation:

- Investigating the impact of different backbone architectures on detection performance
- Optimizing the anchor generation strategy to better handle objects of varying scales and aspect ratios
- Implementing additional data augmentation methods to enhance model robustness

Overall, this implementation provides a solid foundation for understanding and extending the Faster R-CNN architecture for various object detection applications.

[1] NAVER Connect Foundation. Recycle trash dataset, 2024. URL https://github.com/connectfoundation/naverconnect-dataset-trash. Licensed under Creative Commons Attribution 4.0 International (CC BY 4.0).

[2] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016. URL https://arxiv.org/abs/1506.01497.

[3] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

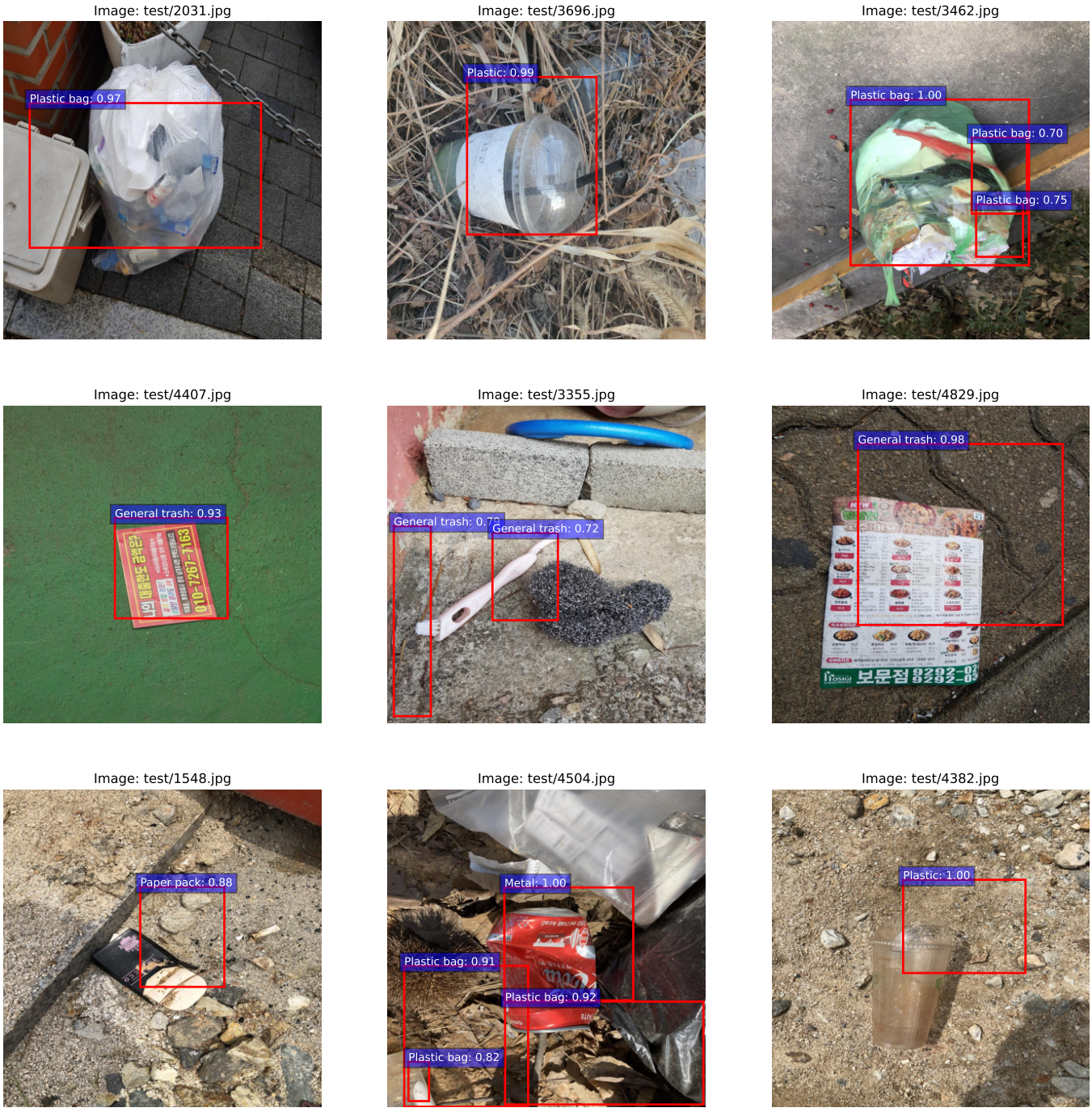


Figure 4: Visualization of model inference results on a representative subset of test images (n=9). The displayed bounding boxes represent detections with confidence scores exceeding 0.7. Each prediction includes a class label and its corresponding confidence score.