Research on Edge Boxes: Locating Object Proposals from Edges

Zhengjie Fan
ECE397
Advisor: Thomas Moon

May 2025

1 Introduction

The generation of object proposals plays a critical role in modern computer vision tasks such as object detection and recognition, and it is particularly important in fields like autonomous driving and intelligent surveillance. Among various approaches, *Edge Boxes*, proposed by Zitnick and Dollár in 2014 [1], stands out due to its simplicity. The algorithm utilizes edge responses in an image and scores candidate bounding boxes based on how well the enclosed edges appear to form a coherent object.

In this report, I present the complete process of reproducing the Edge Boxes algorithm in Python, including edge detection, edge grouping, candidate box sampling, and scoring. I made specific improvements to the sampling and scoring stages so that the algorithm remains functional even in Python—despite being over 100 times slower than the original C++ version. Through this work, I aim to verify the effectiveness of the method, analyze its performance on real images, and explore potential areas for improvement. This implementation has deepened my understanding of object proposal techniques and provided insight into the future development of such algorithms.

2 Algorithm Description

The Edge Boxes algorithm proposes object candidates by leveraging densely computed edge information. This implementation follows the original method,

while introducing practical simplifications and improvements to make it suitable for Python.

2.1 Edge Detection

Here, we use *Structured Edge Detection*, as introduced in the paper *Structured Forests for Fast Edge Detection* [2], to extract edge information from images. Since the focus of this study is not on the process of obtaining edge groups, we do not discuss this part in detail.

However, it is worth noting that this model is highly effective at avoiding the over-detection of textures within objects—an issue commonly encountered when using traditional high-frequency edge extraction methods such as those based on the Fourier transform. This distinction is crucial for ensuring the correctness of the subsequent stages of the algorithm.

2.2 Edge Grouping

To organize the edge pixels into meaningful groups, we compute the gradient direction of the edge map using Sobel filters in both x and y directions. The gradient angle $\theta(x, y)$ at each pixel is calculated as:

$$\theta(x, y) = \arctan 2(dy, dx)$$

We then quantize these angles into 8 discrete orientations (every 45°) to simplify subsequent grouping:

$$orientation\_bin(x, y) = \left\lfloor \frac{\theta(x, y) + 22.5^\circ}{45^\circ} \right\rfloor \bmod 8$$

For each of the 8 orientation bins, we apply **connected component labeling** to identify locally coherent edge segments. This gives us an initial segmentation of the edge map into distinct edge groups s_i, each with similar direction and spatial continuity.

To further improve the coherence of groups, we merge adjacent groups if their average orientations differ by less than a threshold (e.g., $\frac{\pi}{2}$). This step ensures that edge fragments belonging to the same underlying structure are combined.

This process yields a labeled map of edge groups, which forms the basis for affinity graph construction and later object proposal scoring.

2.3 Filtering Weak Edge Groups

To remove noisy or small edge fragments, we filter out edge groups whose total edge strength is below a threshold τ (e.g., $\tau = 5$). Specifically, for each group s_i, we compute:

$$mag\_sum(s_i) = \sum_{(x,y)\in s_i} E(x, y)$$

Only groups with $mag\_sum \geq \tau$ are retained. The removed pixels are reassigned to the nearest valid group with the most similar orientation. For each valid group, we store its total edge strength and center coordinates for later use in box scoring.

2.4 Affinity Graph

We construct a group-level affinity graph to model the spatial and directional relationships between edge segments.

First, we build a neighbor list for each edge group using 8-connected pixels. Then, we compute the dominant orientation θ_i of each group using the average of per-pixel gradient directions.

Next, for each pair of neighboring groups (s_i, s_j), we define their affinity as:

$$a(s_i, s_j) = |\cos(\theta_i - \theta_{ij}) \cdot \cos(\theta_j - \theta_{ij})|^\gamma$$

where θ_{ij} is the angle between the centers of s_i and s_j, and $\gamma = 2$.

Only neighboring groups with sufficient directional alignment and spatial proximity are connected in the graph. The purpose of the affinity graph is to establish meaningful connections between separate but closely related edge groups. In the scoring process, we aim to assign high scores to boxes that enclose important contours without significantly intersecting them. Therefore, identifying which groups constitute the main object contours is crucial. The affinity graph enables us to better penalize boxes that intersect with these groups, while rewarding those that fully contain them.

2.5 Scoring Candidate Boxes

Each candidate box $b = (x, y, w, h)$ is assigned a score based on the edge groups it contains and their enclosure confidence.

To determine whether an edge group s_i is enclosed within a candidate box b, we compute an enclosure weight $w_b(s_i) \in [0, 1]$.

If the group is fully inside the box, we assign:

$$w_b(s_i) = 1$$

If the group touches or connects to the boundary, we estimate the likelihood of being connected to outside content using the affinity graph. Specifically, we define:

$$w_b(s_i) = 1 - \max_{p \in \mathcal{P}_{s_i \to S_b}} \prod_{(u,v) \in p} a(u, v)$$

where:

S_b is the set of edge groups that touch the boundary of b, $\mathcal{P}_{s_i \to S_b}$ is the set of affinity paths from s_i to any group in S_b, $a(u, v)$ is the affinity between neighboring groups u and v.

This definition ensures that groups highly connected to the boundary (via high-affinity paths) receive lower $w_b(s_i)$, reflecting a lower confidence of being fully enclosed.

The raw score is computed as:

$$score_{raw}(b) = \sum_i w_b(s_i) \cdot m_i$$

where m_i is the total edge strength of group s_i.

A penalty is applied to groups partially inside the box:

$$penalty_{corr}(b) = \sum_i (1 - w_b(s_i)) \cdot m_i$$

We normalize the difference by box perimeter using exponent κ:

$$score_{norm}(b) = \frac{score_{raw}(b) - penalty_{corr}(b)}{2(w+h)^{\kappa}}$$

where κ is set to 1.5.

In addition, a center penalty is applied to suppress boxes where edges concentrate only in the center:

$$penalty_{center}(b) = \frac{\sum\limits_{(x,y)\in b_{center}} E(x,y)}{2(w+h)^{\kappa}}$$

We define the center region of a candidate box $b = (x, y, w, h)$ as:

$$b_{center} = \left(x + \frac{w}{4},\ y + \frac{h}{4},\ \frac{w}{2},\ \frac{h}{2}\right)$$

The final score is:

$$final\_score(b) = score_{norm}(b) - penalty_{center}(b)$$

This scoring function favors boxes that fully enclose high-strength edge groups and penalizes those that intersect object contours or contain only central texture.

2.6 Box Sampling Strategy

My strategy balances exploration (via random perturbation) and exploitation (via greedy acceptance), and is particularly useful in low-cost frameworks like EdgeBoxes.

2.6.1 Random Sampling Strategy

To explore the large space of possible bounding boxes, we adopt a simple random sampling strategy. For each image, we generate n candidate boxes with randomized sizes and positions, and evaluate their scores using the full scoring function.

This approach offers several practical benefits:

- **Efficiency**: Sampling avoids the exhaustive sliding-window search used in traditional detectors, making it lightweight and fast to implement.
- **Coverage**: By generating boxes at various scales and aspect ratios, we increase the chance of capturing object contours, especially for objects of irregular size or position.
- **Simplicity**: Unlike heuristic region-growing or clustering methods, random sampling does not rely on low-level segmentation priors.

Despite its simplicity, random sampling proves effective when combined with a strong scoring function and sufficient sample count (e.g., $n = 300$). In our experiments, even a few hundred samples yield high-quality proposals with low computational cost.

2.6.2 Local Refinement via Random Walk

To further improve the quality of proposals obtained from random sampling, we apply a greedy local search strategy based on random perturbations.

Starting from the best box we got from Random Sampling Strategy b, we iteratively generate perturbed versions b' by adding small random shifts to the position and size:

$$b' = (x + \Delta x,\ y + \Delta y,\ w + \Delta w,\ h + \Delta h)$$

where $\Delta x, \Delta y \in [-\delta_{pos}, \delta_{pos}]$ and $\Delta w, \Delta h \in [-\delta_{size}, \delta_{size}]$ are sampled randomly.

At each iteration, we perform multiple trials (e.g., 8), and accept the first perturbed box with a strictly higher score than the current best. This forms a greedy random walk over the search space.

The process continues for a fixed number of iterations or until no further improvement is found. This refinement helps fine-tune promising boxes, especially when the initial sampling resolution is coarse.

3 Results and Analysis

3.1 Original Image and Edge Map

Each image shows the structured edge detection result for a different input example. Brighter regions indicate stronger edge responses.

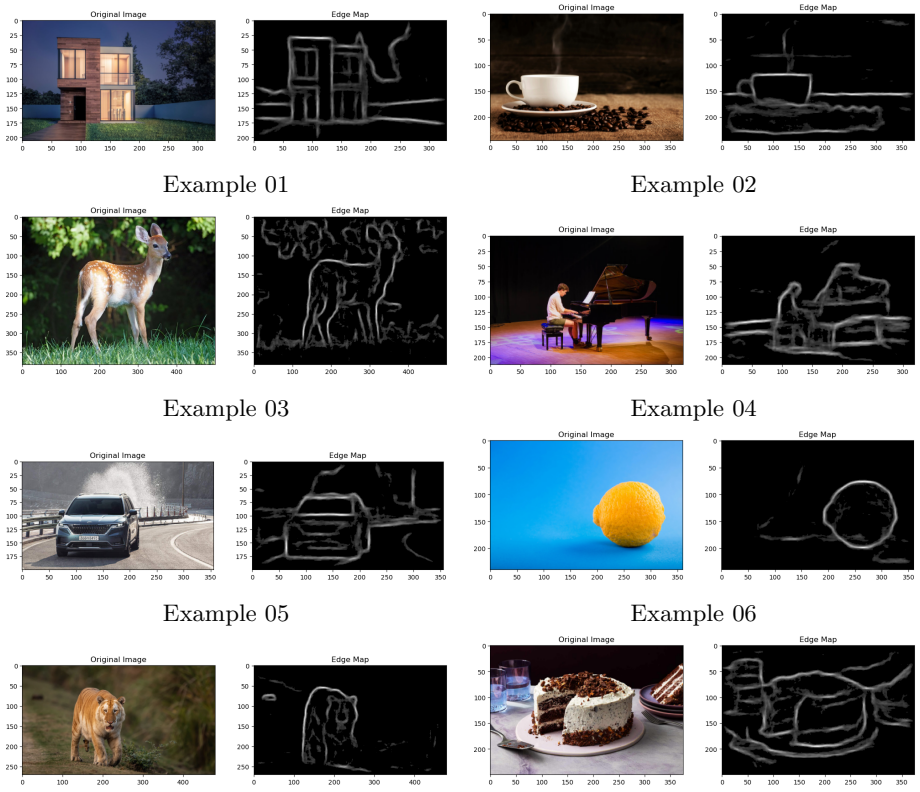
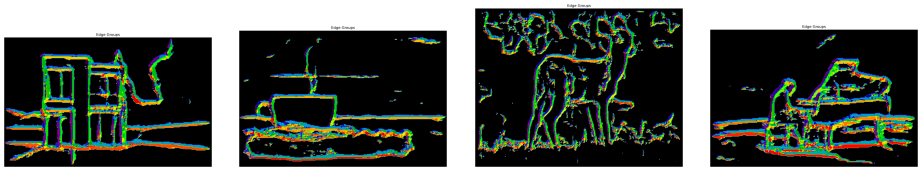


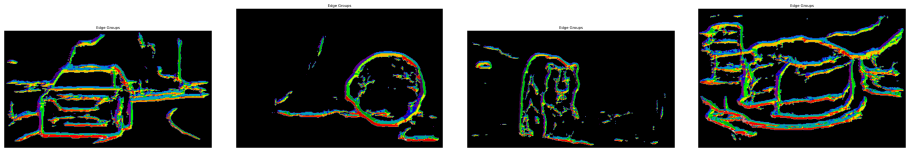
Figure 1: Edge Maps detected using Structured Edge Detection

3.2 Edge Groups

The detected edge pixels are grouped into segments based on local orientation and connectivity.



Example 01 Example 02 Example 03 Example 04

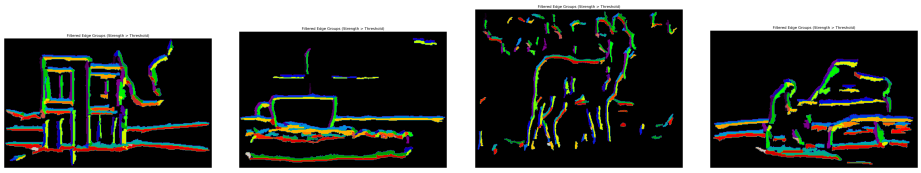


Example 05 Example 06 Example 07 Example 08

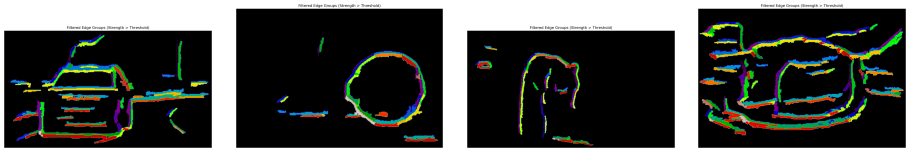
Figure 2: Edge groups after quantization and connected-component labeling

3.3 Filtered Edge Groups

Weak edge segments are removed, and noisy fragments are merged to form cleaner edge structures.



Example 01 Example 02 Example 03 Example 04

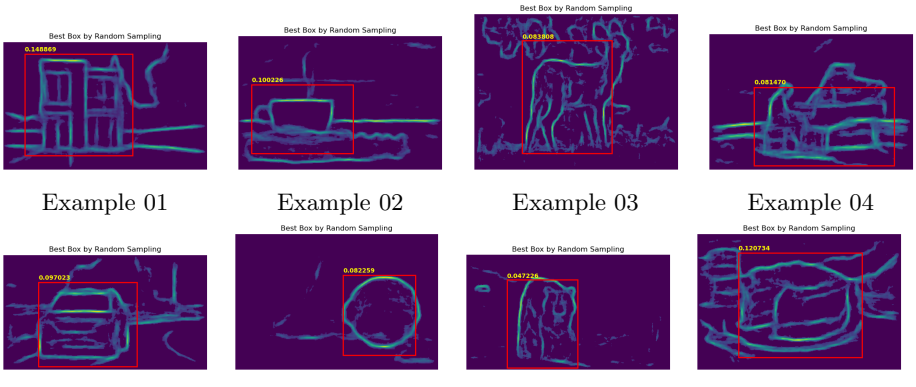


Example 05 Example 06 Example 07 Example 08

Figure 3: Filtered edge groups with strength thresholding and merging

3.4 Best Box by Random Sampling

The best proposal found by scoring 300 randomly sampled boxes per image.



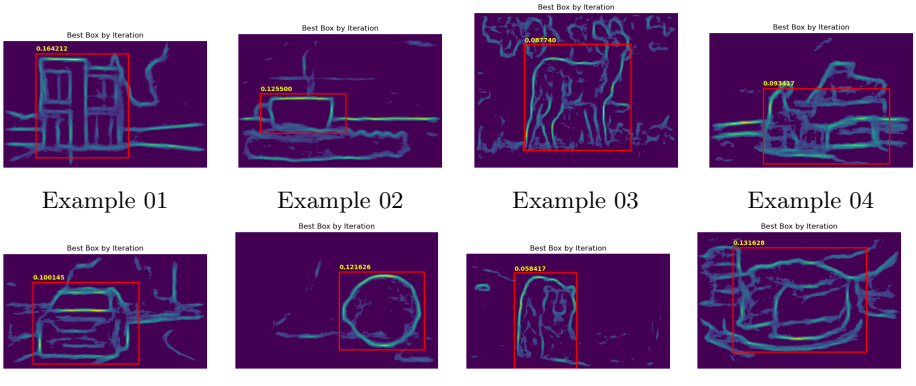
Example 01 Example 02 Example 03 Example 04

Example 05 Example 06 Example 07 Example 08

Figure 4: Best box by random sampling

3.5 Best Box by Iterative Refinement

Each proposal is refined using a greedy local random walk algorithm to improve the score.



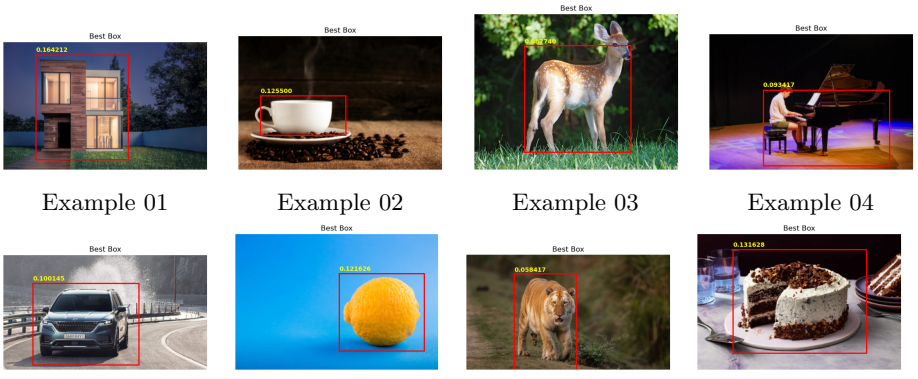
Example 01 Example 02 Example 03 Example 04

Example 05 Example 06 Example 07 Example 08

Figure 5: Best box after refinement via random walk

3.6 Final Result (RGB)

The final proposal is overlaid on the original RGB image to visualize the detected object.



Example 01 Example 02 Example 03 Example 04

Example 05 Example 06 Example 07 Example 08

Figure 6: Final proposed box on original RGB image

4 Conclusion and Future Improvements

From the results shown above, we can see that the algorithm overall functions correctly. In most examples, the selected boxes successfully localize the target objects. A few cases, such as Example 03 and 04, are also very close to perfect results.

The main difference between my implementation and the original paper lies in the box selection strategy. The original method exhaustively searches over a large set of boxes with varying sizes and aspect ratios. However, in practice, the region in which objects are likely to appear can often be effectively covered by a much smaller number of randomly sampled boxes. Therefore, random sampling is demonstrated to be a simpler yet effective alternative for generating high-quality proposals.

Several potential directions for improvement are worth mentioning:

- **Edge-aware background suppression.** Applying selective blurring or suppression to edge regions in the picture may help reduce interference from textures in the background, leading to cleaner edge maps and better grouping.
- **Multi-path iterative refinement.** Instead of refining only one box, multiple refinement paths can be run in parallel from different initial proposals. Non-maximum suppression (NMS) can then be applied to avoid duplicate detections and enable multi-object localization.
- **Box generation from connected edge structures.** For large, closed

edge contours, it may be beneficial to directly generate bounding boxes around these components. This can be especially effective for well-contained objects.

- **Improved edge detection.** Since the entire pipeline depends on the quality of the edge map, further improving the edge detection algorithm could significantly enhance overall performance.

References

[1] C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges," in *European Conference on Computer Vision (ECCV)*, pp. 391–405, Springer, 2014.

[2] P. Dollár and C. L. Zitnick, "Structured forests for fast edge detection," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1841–1848, 2013.