Selective Search for Object Recognition

Uijlings et al.
Overview

● Introduction
● Object Recognition
● Selective Search
  ○ Similarity Metrics
● Results
Object Recognition

**Goal:**

**Problem:** Where do we look in the image for the object?
One Solution

**Idea:** Exhaustively search for objects.

**Problem:** Extremely slow, must process tens of thousands of candidate objects.

One Solution

Idea: Running a scanning detector is cheaper than running a recognizer, so do that first.

1. Exhaustively search for candidate objects with a generic detector.
2. Run recognition algorithm only on candidate objects.

Problem: What about oddly-shaped objects? Will we need to scan with windows of many different shapes?

Segmentation

**Idea:** If we correctly segment the image before running object recognition, we can use our segmentations as candidate objects.

**Advantages:** Can be efficient, makes no assumptions about object sizes or shapes.
General Approach

Original Image → Search → Candidate Boxes → Object Recognition → Final Detections

Key contribution of this paper
Overview

● Introduction

● Object Recognition

● Selective Search
  ○ Similarity Metrics

● Results
Recognition Algorithm

Basic approach:

- Bag of words model, with SIFT-based feature descriptors
- Spatial pyramid with four levels to encode some spatial information
- SVM for classification
Object Recognition

Training:

- **Ground truth**
- **Object hypotheses**
- **Training Examples**
- **Model** (SVM, Histogram Intersection Kernel)
- **False Positives**
- **Training Examples**

Positive examples: Difficult negatives if overlap with positive 20-50%
Object Recognition

Step 1: Train Initial Model

Positive Examples: From ground truth.
Negative Examples: Sample hypotheses that overlap 20-50% with ground truth.
Object Recognition

Step 2: Search for False Positives

Run model on image and collect mistakes.
Object Recognition

Step 3: Retrain Model

Add **false positives** as new **negative examples**, retrain.
Overview

- Introduction
- Object Recognition
- Selective Search
  - Similarity Metrics
- Results
Hierarchical Image Representation

Images are actually 2D representations of a 3D world.
Objects can be on top of, behind, or parts of other objects.

We can encode this with an object/segment hierarchy.
Segmentation is Hard

As we saw in Project 1, it’s not always clear what separates an object.

Kittens are distinguishable by color (sort of), but not texture.

Chameleon is distinguishable by texture, but not color.
Segmentation is Hard

As we saw in Project 1, it’s not always clear what separates an object.

Wheels are part of the car, but not similar in color or texture.

How do we recognize that the head and body/sweater are the same “person”?
Selective Search

Goals:

1. Detect objects at any scale.
   a. Hierarchical algorithms are good at this.

2. Consider multiple grouping criteria.
   a. Detect differences in color, texture, brightness, etc.

3. Be fast.

Idea: Use bottom-up grouping of image regions to generate a hierarchy of small to large regions.
Selective Search

Step 1: Generate initial sub-segmentation

Goal: Generate many regions, each of which belongs to at most one object.

Using the method described by Felzenszwalb et al. from week 1 works well.

Selective Search

Step 2: Recursively combine similar regions into larger ones.

Greedy algorithm:
1. From set of regions, choose two that are most similar.
2. Combine them into a single, larger region.
3. Repeat until only one region remains.

This yields a hierarchy of successively larger regions, just like we want.
Selective Search

Step 2: Recursively combine similar regions into larger ones.
Selective Search

Step 3: Use the generated regions to produce candidate object locations.
Overview

- Introduction
- Object Recognition
- Selective Search
  - Similarity Metrics
- Results
Similarity

What do we mean by “similarity”?

Goals:

1. Use multiple grouping criteria.
2. Lead to a balanced hierarchy of small to large objects.
3. Be efficient to compute: should be able to quickly combine measurements in two regions.
Similarity

What do we mean by “similarity”?

Two-pronged approach:

1. Choose a **color space** that captures interesting things.
   a. Different color spaces have different invariants, and different responses to changes in color.

2. Choose a **similarity metric** for that space that captures everything we’re interested: color, texture, size, and shape.
Similarity

RGB (red, green, blue) is a good baseline, but changes in illumination (shadows, light intensity) affect all three channels.
Similarity

HSV (hue, saturation, value) encodes color information in the hue channel, which is invariant to changes in lighting. Additionally, saturation is insensitive to shadows, and value is insensitive to brightness changes.
**Similarity**

Lab uses a lightness channel and two color channels (a and b). It’s calibrated to be *perceptually uniform*. Like HSV, it’s also somewhat invariant to changes in brightness and shadow.
Similarity

Similarity Measures: Color Similarity

Create a color histogram $C$ for each channel in region $r$. In the paper, 25 bins were used, for 75 total dimensions.

We can measure similarity with histogram intersection:

$$s_{colour}(r_i, r_j) = \sum_{k=1}^{n} \min(c_i^k, c_j^k)$$
Similarity

Similarity Measures: Texture Similarity

Can measure textures with a HOG-like feature:
1. Extract gaussian derivatives of the image in 8 directions and for each channel.
2. Construct a 10-bin histogram for each, resulting in a 240-dimensional descriptor.
Similarity

Similarity Measures: Size Similarity

We want small regions to merge into larger ones, to create a balanced hierarchy.

Solution: Add a size component to our similarity metric, that ensures small regions are more similar to each other.

\[ s_{size}(r_i, r_j) = 1 - \frac{\text{size}(r_i) + \text{size}(r_j)}{\text{size}(im)} \]
Similarity

Similarity Measures: Shape Compatibility

We also want our merged regions to be cohesive, so we can add a measure of how well two regions “fit together”.

\[ \text{fill}(r_i, r_j) = 1 - \frac{\text{size}(BB_{ij}) - \text{size}(r_i) - \text{size}(r_i)}{\text{size}(im)} \]
Similarity

Final similarity metric:
We measure the similarity between two patches as a **linear combination** of the four given metrics:

\[
s(r_i, r_j) = a_1 s_{\text{colour}}(r_i, r_j) + a_2 s_{\text{texture}}(r_i, r_j) + \\
a_3 s_{\text{size}}(r_i, r_j) + a_4 s_{\text{fill}}(r_i, r_j),
\]

Then, we can create a diverse collection of region-merging strategies by considering different weighted combinations in different color spaces.
Overview

● Introduction
● Object Recognition
● Selective Search
  ○ Similarity Metrics
● Results
Evaluation

Measuring box quality:

We introduce a metric called **Average Best Overlap**:

$$\text{ABO} = \frac{1}{|G^c|} \sum_{g_i^c \in G^c} \max_{l_j \in L} \text{Overlap}(g_i^c, l_j)$$

Overlap between ground truth and best selected box.

Average of “best overlaps” across all images.
Segmentation Results

Note that HSV, Lab, and rgbI do noticeably better than RGB.

Texture on its own performs worse than the color, size, and fill similarity metrics.

The best similarity measure overall uses all four metrics.
### Segmentation Results

Combining strategies improves performance even more:

<table>
<thead>
<tr>
<th>Version</th>
<th>Diversification Strategies</th>
<th>MABO</th>
<th># win</th>
<th># strategies</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Strategy</td>
<td>HSV</td>
<td>0.693</td>
<td>362</td>
<td>1</td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td>C+T+S+F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$k = 100$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Selective Search Fast</td>
<td>HSV, Lab</td>
<td>0.799</td>
<td>2147</td>
<td>8</td>
<td>3.79</td>
</tr>
<tr>
<td></td>
<td>C+T+S+F, T+S+F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$k = 50, 100$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Selective Search Quality</td>
<td>HSV, Lab, rgI, H, I</td>
<td>0.878</td>
<td>10,108</td>
<td>80</td>
<td>17.15</td>
</tr>
<tr>
<td></td>
<td>C+T+S+F, T+S+F, F, S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$k = 50, 100, 150, 300$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Using an ensemble greatly improves performance, at the cost of runtime (more candidate windows to check).
Segmentation Results

“Quality” can outperform “Fast” even when returning the same number of boxes (when the number of boxes is truncated).

<table>
<thead>
<tr>
<th>method</th>
<th>recall</th>
<th>MABO</th>
<th># windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arbelaez et al. [3]</td>
<td>0.752</td>
<td>0.649 ± 0.193</td>
<td>418</td>
</tr>
<tr>
<td>Alexe et al. [2]</td>
<td>0.944</td>
<td>0.694 ± 0.111</td>
<td>1,853</td>
</tr>
<tr>
<td>Harzallah et al. [16]</td>
<td>0.830</td>
<td>-</td>
<td>200 per class</td>
</tr>
<tr>
<td>Carreira and Sminchisescu [4]</td>
<td>0.879</td>
<td>0.770 ± 0.084</td>
<td>517</td>
</tr>
<tr>
<td>Endres and Hoiem [9]</td>
<td>0.912</td>
<td>0.791 ± 0.082</td>
<td>790</td>
</tr>
<tr>
<td>Felzenszwalb et al. [12]</td>
<td>0.933</td>
<td>0.829 ± 0.052</td>
<td>100,352 per class</td>
</tr>
<tr>
<td>Vedaldi et al. [34]</td>
<td>0.940</td>
<td>-</td>
<td>10,000 per class</td>
</tr>
<tr>
<td>Single Strategy</td>
<td>0.840</td>
<td>0.690 ± 0.171</td>
<td>289</td>
</tr>
</tbody>
</table>

**Selective search “Fast”**
- Recall: 0.980
- MABO: 0.804 ± 0.046
- # windows: 2,134

**Selective search “Quality”**
- Recall: 0.991
- MABO: 0.879 ± 0.039
- # windows: 10,097

Excellent performance with fewer boxes than previous algorithms, which speeds up recognition.
Segmentation Results

(a) Bike: 0.863
(b) Cow: 0.874
(c) Chair: 0.884
(d) Person: 0.882
Segmentation Results

[4] [9]
Recognition Results

Object recognition performance (average precision per class on Pascal VOC 2010):

<table>
<thead>
<tr>
<th>System</th>
<th>plane</th>
<th>bike</th>
<th>bird</th>
<th>boat</th>
<th>bottle</th>
<th>bus</th>
<th>car</th>
<th>cat</th>
<th>chair</th>
<th>cow</th>
<th>table</th>
<th>dog</th>
<th>horse</th>
<th>motor</th>
<th>person</th>
<th>plant</th>
<th>sheep</th>
<th>sofa</th>
<th>train</th>
<th>tv</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUS</td>
<td>.491</td>
<td>.524</td>
<td>.178</td>
<td>.120</td>
<td>.306</td>
<td>.535</td>
<td>.328</td>
<td>.373</td>
<td>.177</td>
<td>.306</td>
<td>.277</td>
<td>.295</td>
<td>.519</td>
<td>.563</td>
<td>.442</td>
<td>.096</td>
<td>.148</td>
<td>.279</td>
<td>.495</td>
<td>.384</td>
</tr>
</tbody>
</table>

A couple of notable misses compared to other techniques, but best on about half, and best on average.
Effect of Location Quality

Performance is pretty close to “optimal” with only a few thousand iterations.
Summary

- We can speed up object recognition by applying a segmentation algorithm first, to help select object locations.

- Selective Search is a flexible hierarchical segmentation algorithm for this purpose.

- Performance is improved by using a diverse set of segmentation criteria.

- The performance of Selective Search and the complete object recognition pipeline are both very competitive with other approaches.
Questions?