Improving Image Analysis Algorithms using Automatic Programming

Lars Vidar Magnusson


- Image analysis
- Edge detection and image segmentation
- Improving state-of-the-art algorithms with automatic programming.
Image analysis is a term used to cover a wide range of tasks that are related to computer vision i.e. we are attempting to make computers be able to see.

- Facial recognition
- Car safety systems
- Robotics
- Medical diagnosis
- Geological surveys
- ...

Image analysis is related to artificial intelligence.
Despite the wide range of applications, most image analysis uses a variation of the pipeline below.
Edge detection is the process of finding discontinuities, or abrupt changes in intensity, in digital images.

Algorithms are typically based on simple mathematic principles

- Difference
- Derivatives

Standard edge detectors.

- Sobel, Prewitt, Laplacian of Gaussian (LoG) (filter based)
- Canny [3] (filter based, but with significant additions)
- Portability boundary (Pb) [1]
Image segmentation is used to describe either the process of dividing an image into its constituent parts or separating an image into foreground and background.

Image segmentation is closely related to edge detection.

- Closed edges can be turned into regions

Standard image segmentation algorithms.

- Normalized Cut [9]
- Mean Shift [4]
- Felzenszwalb & Huttenlocher [5]
My PhD has been focused on using Automatic Programming to improve low-level image analysis algorithms.

- Image segmentation
  - Graph-Based Image Segmentation (one paper published [7] plus experimental foundation for a new one)
  - Pulse Coupled Neural Networks (PCNN) (work based on [2] in progress)
- Edge detection
  - Canny (two papers accepted [8], one in for review, several more planned)
  - gPb (experimental foundation under development)
  - Filter-based (one paper published on a logic filter [6] and one more planned)
All attempts at improving an algorithm with automatic programming follow the same high-level recipe.

- Port the algorithm to SML
- Select target program and translate it to ADATE ML
- Define a fitness function
- Start evolution
- Evaluate the improved program
As a first example, let us look at the work we did to improve graph-based image segmentation [7].

The original algorithm [5] is modification of a standard minimum spanning tree algorithm.

- Build a graph based on the differences between the pixels in the image.
- Sort the edges in non-decreasing order.
- Iterate through the sorted list and merge nodes if a custom requirement is met.

The algorithm was ported to SML and tested to make sure it produces the same results as the original.

The third step of the algorithm was chosen as the target function.
Segmentation quality is a matter of perception, so we need a metric that accounts for the ambiguous nature of the problem.

There are many different metrics for image segmentation.

- **F-measure** (fast and effective, but unclear what to do with multiple segments)
- **Variation of Information** (somewhat effective, but no clear way to incorporate multiple ground truths)
- **Segmentation Covering** (slow but effective)
- ...

Typically the type of benchmark is dependent on the type of images (ground truths) used.
Finding a Suitable Image Database

There are several image databases available on the Internet, but not all of them are suitable for our purpose.

In order to create a general improvement we need:

- Images of suitable size (too big $\Rightarrow$ slow run times, too small $\Rightarrow$ low level of detail)
- Ground truth images with identified regions from multiple subjects.
- The images to contain a wide range of different motives and objects.

Based on this we can come up with suitable candidates.

- BSDS500 (popular, large, natural images, multiple objects)
- Weizmann (smaller, natural images, single object)
The original algorithm operates as follows.

- Build a graph where each pixel is a node connected to its 8 immediate neighbors with an edge where the weight corresponds to the difference in intensity.
- Place each pixel in its own component (union find) with a threshold set to a constant $C$.
- Sort all the edges in non-decreasing order.
- Iterate the sorted edges and join the connected components if the weight is smaller than both thresholds.
- The threshold of the joined component is set to be the weight plus the constant $C$ divided by the size of the new component.
fun f ( Universe, SortedEdges, Constant ) =
  case SortedEdges of
    enil => Universe
  | econs ( CurrentEdge as edge( A, B, W, X ), RestEdges ) =>
    let
      val ( ComponentA, ThresholdA ) = find ( A, Universe )
      val ( ComponentB, ThresholdB ) = find ( B, Universe )
    in
      if differentComp ( ComponentA, ComponentB ) then
        if W < ThresholdA andalso W < ThresholdB then
          let
            val NewUniverse = union ( Universe, ComponentA, ComponentB )
            val ( Component, CurrentThreshold ) = find ( A, NewUniverse )
            updateThresholdValue ( Component, W + Constant / getComponentSize Comp )
          in
            f ( updateThresholdValue ( Component, W + Constant / getComponentSize Comp, NewUniverse ),
                RestEdges, Constant )
          end
        else
          f ( Universe, RestEdges, Constant )
        end
      else
        f ( Universe, RestEdges, Constant )
    end
The table below contains the Precision, Recall and F-measure for both the algorithms on both the training and test data.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Train P</th>
<th>Train R</th>
<th>Train F</th>
<th>Test P</th>
<th>Test R</th>
<th>Test F</th>
<th>Total P</th>
<th>Total R</th>
<th>Total F</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>0.79</td>
<td>0.86</td>
<td>0.79</td>
<td>0.81</td>
<td>0.82</td>
<td>0.78</td>
<td>0.80</td>
<td>0.84</td>
<td>0.79</td>
</tr>
<tr>
<td>Original</td>
<td>0.75</td>
<td>0.82</td>
<td>0.71</td>
<td>0.76</td>
<td>0.76</td>
<td>0.69</td>
<td>0.75</td>
<td>0.79</td>
<td>0.70</td>
</tr>
</tbody>
</table>

We also did a pairwise comparison of the two algorithms using student-t distribution on the differences, and we can say with 99 percent confidence that the new algorithm is between 1 and 17 percentage points better than the original on the test images.
fun f ( Universe, SortedEdges, Constant ) = 
case SortedEdges of 
enil => Universe |
econs ( CurrentEdge as edge ( A, B, W, X ), RestEdges ) => 
let 
  val ( ComponentA, ThresholdA ) = find ( A, Universe ) 
  val ( ComponentB, ThresholdB ) = find ( B, Universe ) 
in 
if differentComp ( ComponentA, ComponentB ) then 
  if W < ThresholdA andalso W < ThresholdB then 
    let 
      val NewUniverse = 
        updateThresholdValue ( 
          ComponentB, 
          W + Constant / 
          getComponentSize ( 
            if Constant < ThresholdA then 
              ComponentB 
            else 
              ComponentA ), 
          union ( Universe, ComponentA, ComponentB ) ) 
        in 
      f ( NewUniverse, RestEdges, Constant ) 
    end 
  else if W > ThresholdA andalso W > ThresholdB then 
    f ( Universe, RestEdges, getComponentSize ( ComponentB ) ) 
  else 
    f ( Universe, RestEdges, Constant ) 
else 
  f ( Universe, RestEdges, Constant ) 
end
The improved algorithm is quite similar to the original algorithm. There are three minor changes.

- Updates to the threshold are always made to *ComponentB*.
- It does not use the size of the new component to calculate the new threshold.
- It has introduced a mechanism that changes the constant if the weight of the current edge is larger than both the connected components thresholds.
Segmentation Comparison

Lars Vidar Magnusson
Scientific Method and Theory 2016
Segmentation Comparison
The Canny edge detector [3] is a popular algorithm that can be found in most image analysis platforms.

The algorithm works as follows.

- Smooth the image and find the gradient image.
- Perform *non-max suppression*.
- Find the final edge image by *hysteresis thresholding*.

We ported the entire algorithm into SML, and we decided to investigate the possibility of improving the three stages separately.

Let us first consider improving the second stage, non-max suppression.
Non-max suppression was designed to reduce multiple responses to a single edge.

The idea is to suppress gradient magnitudes that are less than either of the magnitudes along the gradient angle.

The Matlab implementation generates two gradient images, one for each axis.

Consider $d_x$ and $d_y$ as the x- and y-gradient for a given position.

If $d_y$ is positive and $d_x$ larger than $d_y$, or if $d_y$ is negative and $d_x$ is less than $d_y$, the gradient angle is in sector 1 or 5 respectively.

In this case the positions magnitude is suppressed if it is smaller than either of its neighbors along the gradient angle (found using linear interpolation).
We used the BSDS500 [1] in our experiments.

- Specifically designed for training contour detectors
- High quality annotations by multiple subjects
- Widely adopted by the industry

We used the average F-measure for evaluating the programs.

We reduced the time needed to evaluate by using only the best ground truth in each set.

- The ground truth with the highest F-measure when evaluated against the other ground truths in the set.
fun f ( d1, d2, m, m1, m2, m3, m4 ) =
  let
  fun lerp ( x, y, t ) =
    x*( 1.0−t )+y*t
  in
  case abs( d1/d2 ) of
t⇒ case lerp( m1, m2, t ) of
tm1⇒ case lerp( m3, m4, t ) of
tm2⇒ case m < tm1 of
false⇒ ( case m < tm2 of
false⇒ m
| true⇒ 0.0 )
| true⇒ 0.0
end
The Improved Program

fun f( d1, d2, m, m1, m2, m3, m4 ) =
  let
    fun lerp( x, y ) = x*( 1.0−m3 ) + y*m3
  in
  case m < lerp( m1, m2 ) of
    false => ( case m < lerp( m3, m4 ) of
      false => abs( m/tanh( m/d2 ) )
      | true => 0.0 )
    | true => 0.0
end

- smaller than the original.
- the interpolation parameter t has been removed.
- lerp has changed to use $m_3$ instead of $t$.
- now returns $m/(\tanh(m/d2))$ instead of $m$ on unsuppressed values.
The histograms show the distributions of $m_3$ in red and $t$ in blue.

On average $m_3$ is significantly smaller than $t$.

This has the effect that the linear interpolation will prioritize the axis-aligned neighbors over the diagonal when angles are close to the diagonal.
The scatter plots show the difference between $m$ and $m/(\tanh (m/d2))$. Considerable correlation between the two, but the latter is slightly larger.

The variation is caused by the denominator $\tanh (m/d2)$.

Values will be largest when the gradient angle is axis aligned and reduced when approaching the diagonal.
Benchmarks

We did our benchmarks with the dedicated test set in BSDS500, and we evaluated using the same function as in [1] (*accumulated* F-measure).

We test each of the algorithms using two constant configurations; one optimized for the entire dataset (OD), and one optimized for each image (OI).

<table>
<thead>
<tr>
<th></th>
<th>ODF</th>
<th>OIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCG</td>
<td>0.71</td>
<td>0.73</td>
</tr>
<tr>
<td>ADATE-Improved</td>
<td>0.618</td>
<td>0.657</td>
</tr>
<tr>
<td>Canny</td>
<td>0.606</td>
<td>0.652</td>
</tr>
</tbody>
</table>

The ADATE-improved algorithm has been improved by 1.1 percentage points or 1.9% with OD constants, and by 0.5 percentage points or 0.8% with OI constants.

A student-t test and a Wilcoxon signed-rank test gave a *p*-value of $6.45 \times 10^{-9}$ and $1.649 \times 10^{-9}$ respectively.
The ROC curves for the improved (red) and the original (blue).
Bibliography I

Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik.  
Contour detection and hierarchical image segmentation.  

Henrik Berg, Roland Olsson, Thomas Lindblad, and José Chilo.  
Automatic design of pulse coupled neurons for image segmentation.  
Neurocomputing for Vision Research; Advances in Blind Signal Processing.

John Canny.  
A computational approach to edge detection.  

Dorin Comaniciu and Peter Meer.  
Mean shift: A robust approach toward feature space analysis.  

Pedro Felzenszwalb and Daniel Huttenlocher.  
Efficient graph-based image segmentation.  
10.1023/B:VISI.0000022288.19776.77.

Kristin Larsen, Lars Vidar Magnusson, and Roland Olsson.  
Edge pixel classification using automatic programming.  
*Norsk Informatikkkonferanse (NIK)*, 2014.

Lars Vidar Magnusson and Roland Olsson.  
Improving graph-based image segmentation using automatic programming.  
Lars Vidar Magnusson and Roland Olsson.
Improving the canny edge detector using automatic programming: Improving non-max suppression.

Jianbo Shi and Jitendra Malik.
Normalized cuts and image segmentation.