Automatic Image Orientation

By Aleksey Trofimov

Machine Learning, CMPS 242, Fall 2009

University of California, Santa Cruz
Overview

• Want to be able to orient images
• Turns out it only works in selected cases
• Three Types of Features
  – Color Presence Features
  – Viola & Jones Features
  – Edge Direction Features
• Instead of a 2-Step Classifier, do Ranking
• Classification with Adaptive Boosting
• ~90% correctly oriented
Why Orient Images

- Cheap Cameras take all photos as Landscape
- Actually three possible Orientations (0°, 90°, 270°)
- Want a classifier that will determine orientation
  - Could be a Two-Step classifier
    - First Determine whether it should be rotated (0° or not)
    - Then determine which way it should be oriented (90° or 270°)
  - Could be just one (already-right-oriented classifier)
    - Rotate the image two different ways
    - Pick the one with lowest should-be-rotated score.
- Turns out the second approach works much better
  - Should-be-rotated score = AdaBoost Prediction (h(x))
Only images taken outside.
Only images taken during the day.
Mostly images taken looking straight.
Only human-orientable in less than a second.
Otherwise only get 60-80% accuracy.
Feature Set 1 – Color Presence

• Index colors by ratio of Red to Green to Blue

• Equivalent to
  – Brightening Each Pixel
    • Divide by Max(Red, Green, Blue) then multiply by 256
  – Separating Colors into 64 bins (2-bit color per channel)
    • Divide by 64 then ceil, append RGB bits together

• Divide the image into a # by # cell grid (# = 8)

• Feature Vector is the presence of each color in cells
Feature Set 1 – Color Presence

Allows to notice Blue Sky on top

Can see through shade

Finds Shades of Red toward the bottom

Notices Green Grass on the bottom
Feature Set 2 – Viola and Jones

• Actually just an approximation to V&J
• Grayscale the image \((G = .3 R + .59 G + .11 B)\)
• Break up into a \# by \# cell grid
• Compute the sum of all pixels in each cell (there’s a trick to it)
• Compute presence of simple features like
  – \([1 \ 0]\) (left cell brighter than the right cell)
  – \([1 \ 0 \ 1]\) (middle cell darker than two h-adjacent cells)
  – \([1 \ 0; \ 0 \ 1]\) (Dark region heading NE)
  – \([0 \ 1 \ 0]\) (Middle cell brighter than two h-adjacent cells)
  – There’s also \([1 \ 0 \ 1]^T, \ [0 \ 1; \ 1 \ 0], \ [1 \ 0]^T\) and \([0 \ 1 \ 0]^T\).
• Feature vector is just a concatenation of the simple features
Sky is usually brighter than ground...

Actually I have no idea what else it finds, but it finds something.
Feature Set 3 – Edge Direction

• This one is the hardest to compute
• Take 4 channels (Red, Green, Blue, Grayscale)
  – Note: RGB have each pixel brightened to max (like for feature set 1)
• Compute Edges in each one
  – Convolve with \([1 \ 0 \ -1; \ 0 \ 0 \ 0; \ 1 \ 0 \ -1]\) to find horizontal edges
  – Convolve with \([1 \ 0 \ -1; \ 0 \ 0 \ 0; \ 1 \ 0 \ -1]^T\) to find vertical edges
  – Double Step Edge Strength Approximation = |Vertical + Horizontal|
• Compute Edge Directions
  – Direction Index = Floor \(\{ \text{Tan}^{-1}(\text{Horizontal} / \text{Vertical}) \times 5.09 \}\) + 1
• Make an # by # cell grid
• For each cell, for each direction, compute sum of edge strengths
• Feature Vector is a strong presence of each direction in each cell
  – Feature Vector = [ horizontal edges > % threshold in cell 1, ... ]
### Feature Set 3 – Edge Direction

<table>
<thead>
<tr>
<th></th>
<th>Combined</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>Grayscale</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Original</strong></td>
<td><img src="combined-original.png" alt="Image" /></td>
<td><img src="red-original.png" alt="Image" /></td>
<td><img src="green-original.png" alt="Image" /></td>
<td><img src="blue-original.png" alt="Image" /></td>
<td><img src="grayscale-original.png" alt="Image" /></td>
</tr>
<tr>
<td><strong>Brightened</strong></td>
<td><img src="combined-brightened.png" alt="Image" /></td>
<td><img src="red-brightened.png" alt="Image" /></td>
<td><img src="green-brightened.png" alt="Image" /></td>
<td><img src="blue-brightened.png" alt="Image" /></td>
<td><img src="grayscale-brightened.png" alt="Image" /></td>
</tr>
<tr>
<td><strong>Edges</strong></td>
<td><img src="combined-edges.png" alt="Image" /></td>
<td><img src="red-edges.png" alt="Image" /></td>
<td><img src="green-edges.png" alt="Image" /></td>
<td><img src="blue-edges.png" alt="Image" /></td>
<td><img src="grayscale-edges.png" alt="Image" /></td>
</tr>
</tbody>
</table>

Compare the Edges in Grayscale and Combined. Edge directions are taken from the grayscale image.
Feature Set 3 – Edge Direction

Can detect strong horizontal edges in the middle

Usually lots of edges toward the bottom
• Of course, size of cells matter for all features
• So, vary it and append into one giant feature vector
• For this project, I used an 8 by 8 grid
• From there, accumulated into 4 by 4 and 2 by 2
  – Except for V&J, which are cheap, so 7 by 7, 6 by 6 and etc.
• I’m sure more cells give better performance
  – But they are exponentially harder to compute
Classification

• It makes sense to negate features
  – So in total I worked with ~7000 features.

• Adaptive Boosting was used to select best features
  – 50% of features usually come from Edge Directions,
  – 28% from Viola & Jones and 22% from Colors

• Perceptron and Logistic Regression performed worse than AdaBoost quite significantly.
  – Too many junk features

• Possible to have features be more descriptive, if they have % of pixels of a certain color, difference between cells for V&J and actual strength for each edge direction.
  – Can’t use AdaBoost on it
  – Perceptron and Logistic Regression still do worse
With Adaptive Boosting, Loss Function was:

\[ \text{# of Misclassifications} + \sum |\text{Prediction}| \times (\text{Prediction} \neq \text{label}) \]
Results

• Get 89.8% Accuracy with S.D. of 2.77%
• Here are some images that tend get misclassified:
The Ultimate Test

Took my cheap camera, and took 34 photos outside.

Tried to classify with the alphas from previous set (180 features).
Got 31 out of 34 (91%) correct!

Here’s what it missed:
Possible Future Work

- Do a smoother Spatial Pyramid
- Normalize each pixel for color features
- Viola & Jones on Multiple Channels
- Gives Edges on different channels different weights.
- More Depth to Spatial Pyramids
- Try the same algorithm on faces
- Use stronger feature (not just 0s and 1s) with Perceptron or Logistic Regression and Boost it.
- Cut out Square-Sized parts of image, classify them, then rank and let the majority decide.