Hyperspectral Image Visualization with a 3-D SOM

Johannes Jordan, Elli Angelopoulou
Pattern Recognition Lab, University of Erlangen-Nuremberg

Use-case Scenario:
- Interactive multispectral image analysis
- False-color visualization in short time
- Non-linear methods typically too slow

Contributions:
- Revisit false-coloring based on the Self-organizing Map (SOM) [1]
- Design a novel SOM for higher output quality
- Benchmark new method against PCA on diverse images

Source code is available at:
http://gerbil.sf.net/

Algorithm

Self-organizing map for false-coloring:
- 3D mesh of model vectors (neurons)
- Mapping: spectral vector \( \mathbf{v} \rightarrow \) model vector \( \mathbf{m}_j \)
- Result after training: Topological representation of original spectral distribution
- SOM topology coordinates mapped to \( R, G, B \)

SOM training phase:
- For \( d \) bands, we have \( n \) model vectors \( \mathbf{m}_j \in \mathbb{R}^d \), and a side length \( \sigma = \sqrt{\frac{n}{\pi}} \).
- The best matching unit (BMU) \( \mathbf{m}_j \) has the index:
  \[
  c(\mathbf{v}) = \arg\min_{j} \sum_{i} d(\mathbf{v}; \mathbf{m}_j).
  \]
- The location of \( \mathbf{m}_j \) is \( r^j = \sigma \cdot \lfloor \frac{1}{\sigma} \cdot t \rfloor \).
- At iteration \( t \), the neighborhood function
  \[
  h_{\sigma}(t) = \alpha(t) \cdot \exp \left( -\frac{1}{2}(t^2) \right)
  \]
  defines the influence of \( \mathbf{v}(t) \) on each \( \mathbf{m}_j \).

False-color generation:
- The false-color values of a pixel \( \mathbf{v} \) are obtained as
  \[
  r_{\mathbf{v}} = \frac{r^c(\mathbf{v})}{\sigma} \quad g_{\mathbf{v}} = \frac{g^c(\mathbf{v})}{\sigma} \quad b_{\mathbf{v}} = \frac{b^c(\mathbf{v})}{\sigma}.
  \]
- Problem: Quantization effects, low quality output
- Examples with \( \sigma = 4 \) (as suggested in [2])

New Method

Provide higher quality output:
- Use larger SOM size: \( 10^3 \) neurons (previous work: 64 to 256 neurons)
- Train with 100,000 samples
- Change false-color generation: several BMUs instead of single BMU

BMU lookup and combination:
- We first obtain a vector of BMU indices \( c(\mathbf{v}) = \arg\min_{j} d(\mathbf{v}; \mathbf{m}_j) \).
- Then, for each pixel \( \mathbf{v} \), we calculate location \( r' \) as
  \[
  r' = \sum_{j} w_j r^c(\mathbf{m}_j)
  \]
  given \( w_j = C \cdot w_j / \sum_{j} w_j, \quad v_m_{M_{Cj}} = C \cdot (d(\mathbf{v}; \mathbf{m}_j) < d(\mathbf{v}; \mathbf{m}_{Cj+1}) \).
- We finally obtain
  \[
  r_{x,y} = \frac{r'_x}{n'}, \quad g_{x,y} = \frac{r'_y}{n'}, \quad b_{x,y} = \frac{r'_z}{n'}
  \]

Novel rank-based weighting scheme:
- Always highlight influence of primary BMU
- Exponential decay of rank weights
- Example comparison with \( C = 10 \) against \( w_j = \frac{1}{C} \cdot v_j \).

Timing Results

- Measured on Intel Core i7-2600 CPU
- Training takes almost all time
- Computational complexity independent of image size, parameter \( C \)

<table>
<thead>
<tr>
<th>#neurons</th>
<th>6^3</th>
<th>8^3</th>
<th>10^3</th>
</tr>
</thead>
<tbody>
<tr>
<td>remote sensing images</td>
<td>4.8 s</td>
<td>10.1 s</td>
<td>17.8 s</td>
</tr>
<tr>
<td>CAVE images</td>
<td>2.9 s</td>
<td>6.0 s</td>
<td>10.1 s</td>
</tr>
</tbody>
</table>

References