Video Propagation Networks

V. Jampani, R. Gadde and P. V. Gehler, CVPR 2017

Jonáš Šerých
2019-09-05
The Task

Given:

- Video sequence
- Per-pixel information (color, segmentation, ...) on few frames

Propagate the information to the whole video.
The Task

Given:

- Video sequence
- Per-pixel information (color, segmentation, ...) on few frames

Propagate the information to the whole video.
The Approach

Bilateral network
- image-adaptive spatio-temporal dense filtering
- straight-forward integration of temporal information

Spatial Network
- shallow CNN
  - spatial refinement
Bilateral Filtering – Introduction

Standard Gaussian filtering – weighted average of all pixel values:

\[ v'_i \approx \sum_{j=0}^{n} e^{-\|p_i - p_j\|^2} v_j \]

\[ p_i = (x_i, y_i) \]

- spatially close → bigger influence
Standard Gaussian filtering – weighted average of all pixel values:

\[
v'_i \approx \sum_{j=0}^{n} e^{-||p_i - p_j||^2} v_j
\]

\[p_i = (x_i, y_i)\]

- spatially close → bigger influence

Bilateral filtering:

\[p_i = (x_i, y_i, R_i, G_i, B_i)\]

- spatially close and visually similar → bigger influence
Edge-Preserving Bilateral Filtering Illustration

https://saplin.blogspot.com/2012/01/bilateral-image-filter-edge-preserving.html
Signal (coloring) on low-resolution image upsampled using high-resolution image guide.

Image from slides by Peter Gehler
The main idea: Use the current frame as a guide for information propagation from the past frames.

Use \((x, y, R, G, B, t)\) instead of \((x, y, R, G, B)\).
1. **Splat**: Embed input values $v_i$ at positions $p_i$ in a high-dimensional space.

2. **Blur**: Perform the filtering.

3. **Slice**: Sample the space at positions $p'_i$. 

![Diagram of Bilateral Filtering Process](image-url)
Naive Implementation

2D example:

- Just do a convolution with Gaussian filter.
- But what if the positions are not on the grid?

We could *splat* values onto the grid using bilinear interpolation:

OK, **but**: Regular square grid: $2^D$ neighboring vertices!
Efficient Implementation Using Permutohedral Lattice

*Permutohedral lattice*: only $D + 1$ neighboring vertices

1. Find the nearest lattice vertices and the corresponding weights in $O(D^2)$.
2. Accumulate weighted values in lattice vertices (*splat*).
3. Perform convolution on the lattice (*blur*).
4. Interpolate from the lattice (*slice*).
Linearity of Bilateral Filtering

Given (1-D for simplicity) values \( \mathbf{v} \in \mathbb{R}^N \) at positions \( \mathbf{p} \in \mathbb{R}^{N \times D} \):

- Construct \( S_{\text{splat}} \in \mathbb{R}^{M \times N} \) using \( \mathbf{p} \).
  - \( M \) … number of lattice points.
  - Each column of \( S_{\text{splat}} \) contains the weights of single input.
- Construct convolution in the matrix form \( B \in \mathbb{R}^{M \times M} \).
- Construct \( S_{\text{slice}} \in \mathbb{R}^{N \times M} \) similarly to \( S_{\text{splat}} \).

Then: \( \mathbf{v}' = S_{\text{slice}} \left( B \left( S_{\text{splat}} \mathbf{v} \right) \right) \)

Linear in \( \mathbf{v} \) and the convolution weights inside \( B \).

Backpropagation possible.
VPN Architecture

- splat in the first $BCL_{a,b}$ layers guided by previous frames
- the rest guided by the current frame
- ReLU after concatenations and spatial convolutions
- $\Lambda_{a,b}$ position scales found by validation
Some Setup Details

- splice: random sampling or superpixels (12000)
- bilateral convolutions with no neighborhood
- YCbCr instead of RGB
- weighting previous 9 frame values by $\alpha, \alpha^2, \alpha^3, \ldots$, where $\alpha = 0.5$ (!!!)
- optical flow for transformation of positions into current frame
- multi-stage training and inference
## Object Segmentation Results

<table>
<thead>
<tr>
<th>Method</th>
<th>(\text{IoU}^\uparrow)</th>
<th>(\mathcal{F}^\uparrow)</th>
<th>(\mathcal{T}^\downarrow)</th>
<th>Runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNN-Identity</td>
<td>67.0</td>
<td>67.1</td>
<td>36.3</td>
<td>0.21</td>
</tr>
<tr>
<td>VPN-Stage1</td>
<td>70.1</td>
<td>68.4</td>
<td>30.1</td>
<td>0.48</td>
</tr>
<tr>
<td>VPN-Stage2</td>
<td>71.3</td>
<td>68.9</td>
<td>30.2</td>
<td>0.75</td>
</tr>
<tr>
<td><strong>With pre-trained models</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeepLab</td>
<td>57.0</td>
<td>49.9</td>
<td>47.8</td>
<td>0.15</td>
</tr>
<tr>
<td>VPN-DeepLab</td>
<td><strong>75.0</strong></td>
<td><strong>72.4</strong></td>
<td><strong>29.5</strong></td>
<td>0.63</td>
</tr>
<tr>
<td>OFL [75]</td>
<td>71.1</td>
<td>67.9</td>
<td>22.1</td>
<td>(&gt;60)</td>
</tr>
<tr>
<td>BVS [53]</td>
<td>66.5</td>
<td>65.6</td>
<td>31.6</td>
<td>0.37</td>
</tr>
<tr>
<td>NLC [25]</td>
<td>64.1</td>
<td>59.3</td>
<td>35.6</td>
<td>20</td>
</tr>
<tr>
<td>FCP [60]</td>
<td>63.1</td>
<td>54.6</td>
<td>28.5</td>
<td>12</td>
</tr>
<tr>
<td>JMP [26]</td>
<td>60.7</td>
<td>58.6</td>
<td><strong>13.2</strong></td>
<td>12</td>
</tr>
<tr>
<td>HVS [29]</td>
<td>59.6</td>
<td>57.6</td>
<td>29.7</td>
<td>5</td>
</tr>
<tr>
<td>SEA [62]</td>
<td>55.6</td>
<td>53.3</td>
<td>13.7</td>
<td>6</td>
</tr>
</tbody>
</table>
## Semantic Segmentation Results

<table>
<thead>
<tr>
<th>Model</th>
<th>IoU</th>
<th>Runtime(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN-1 from [79]</td>
<td>65.3</td>
<td>0.38</td>
</tr>
<tr>
<td>+ FSO-CRF [43]</td>
<td>66.1</td>
<td>&gt;10</td>
</tr>
<tr>
<td>+ BNN-Identity</td>
<td>65.3</td>
<td>0.31</td>
</tr>
<tr>
<td>+ BNN-Identity-Flow</td>
<td>65.5</td>
<td>0.33</td>
</tr>
<tr>
<td>+ VPN (Ours)</td>
<td>66.5</td>
<td>0.35</td>
</tr>
<tr>
<td>+ VPN-Flow (Ours)</td>
<td>66.7</td>
<td>0.37</td>
</tr>
<tr>
<td>CNN-2 from [65]</td>
<td>68.9</td>
<td>0.30</td>
</tr>
<tr>
<td>+ VPN-Flow (Ours)</td>
<td>69.5</td>
<td>0.38</td>
</tr>
</tbody>
</table>
Color Propagation Example Outputs

Frame 2  Frame 7  Frame 13  Frame 19

VPN (Ours)  Levin et al.  GT-Color Input Video
Conclusions

- Efficient implementation of high-dimensional convolutions using permutohedral lattices
- Fast propagation of arbitrary data in video sequences

1. Take the hyperplane of $\mathbb{R}^{D+1}$ in which coordinates sum to zero. $H_D : \mathbf{x} \cdot \mathbf{1} = 0$

2. The hyperplane $H_D$ is spanned by “base” vectors:
   $(D, -1, \ldots, -1), (-1, D, -1, \ldots, -1), \ldots, (-1, \ldots, -1, D)$

3. Integer combinations of the “base” vectors are the lattice vertices.
View orthogonal to the hyperplane.
Integer combinations of the “base” vectors form the lattice.
Splat with Permutohedral Lattice

Each vertex has consistent coordinates modulo \((D + 1)\).
Permutohedron formed by the lattice points.
Splat with Permutohedral Lattice

The $H_D$ hyperplane is tiled by translations of the permutohedron.
The neighboring lattice vertices fully identified by closest 0-remainder point $\mathbf{l}_0$ and coordinate ordering of $\mathbf{x} - \mathbf{l}_0$. 
Finding closest remainder-0 vertex

1. $l_0 \leftarrow$ round coordinates of $x$ to nearest multiple of $(D + 1)$
2. Sort the coordinates by the amount of rounding
3. Iterate starting with the most rounded coordinate:
   3.1 If $l_0$ lies on $H_D$: finish
   3.2 Round in the opposite direction
   3.3 Go to the next coordinate
Splat with Permutohedral Lattice

1. Project input position \( \mathbf{p} \) into the \((D+1)\)-dimensional hyperplane \( H_D \).
2. Find closest remainder-0 point.
3. Find corresponding simplex.
4. Compute barycentric weights \( w_i, i \in \{1, 2, \ldots, D + 1\} \).
5. Accumulate the input value \( \mathbf{v} \) weighted by \( w_i \) into the neighboring lattice vertices (entries in a hash-table).