Fast Edge Detection Using Structured Forests
by Piotr Dollar, C. Lawrence Zitnick (PAMI 2015)

Vojtěch Cvrček

Reading Group Presentation

June 6, 2018
Problem Definition

- given a set of training images and corresponding segmentation masks
Problem Definition

- given a set of training images and corresponding segmentation masks
- predict an edge map for a novel input image
Edge structure

Fast edge detection

June 6, 2018
Random Forest - Single Tree

\[ f_t : \mathcal{X} \rightarrow \mathcal{Y} \quad x \in \mathcal{X} \]

\[ h_{\theta_j} : \mathcal{X} \rightarrow \{L, R\} \]

\[ \delta \in \{0, 1\} \]

\[ y_1, y_2, y_3, y_4, y_5, y_6, y_7 \]

\[ x \in \mathcal{X} \]

Fast edge detection
Random Forest - Single Tree

\[ f_t : \mathcal{X} \rightarrow \mathcal{Y} \quad x \in \mathcal{X} \]

- \[ h_{\theta^1_j}^1(x) = x(k) < \tau \]
- \[ \theta^1_j = (k, \tau) \]
- \[ h_{\theta^2_j}^2(x) = x(k_1) - x(k_2) < \tau \]
- \[ \theta^2_j = (k_1, k_2, \tau) \]
- \[ h_{\theta_j}(x) = \delta h_{\theta^1_j}^1(x) + (1 - \delta) h_{\theta^2_j}^2(x) \]
- \[ \delta \in \{0, 1\} \]
Random Forest - Single Tree

\[ f_t : \mathcal{X} \rightarrow \mathcal{Y} \quad x \in \mathcal{X} \]
Random Forest - Single Tree

\( f_t : \mathcal{X} \rightarrow \mathcal{Y} \quad x \in \mathcal{X} \)

\[ h_{\theta_j}(x) = \begin{cases} x(k) & < \tau_{\theta_1} \\ x(k_1) - x(k_2) & < \tau_{\theta_2} \\ \delta h_{\theta_1}(x) + (1 - \delta) h_{\theta_2}(x) & \end{cases} \]

\( \delta \in \{0, 1\} \)
Random Forest - Single Tree

\[ \mathbf{f}_t : \mathbf{X} \rightarrow \mathbf{Y} \]

\[ x \in \mathbf{X} \]

\[ \mathbf{y}_1 \]

\[ \mathbf{y}_2 \]

\[ \mathbf{y}_3 \]

\[ \mathbf{y}_4 \]

\[ \mathbf{y}_5 \]

\[ \mathbf{y}_6 \]

\[ \mathbf{y}_7 \]
each tree is trained independently in a recursive manner
- each tree is trained independently in a recursive manner
- for a given node \( j \) and training set \( S_j \subset \mathcal{X} \times \mathcal{Y} \), randomly sample parameters \( \theta_j \) from parameters space
Training

- each tree is trained independently in a recursive manner
- for a given node $j$ and training set $S_j \subset \mathcal{X} \times \mathcal{Y}$, randomly sample parameters $\theta_j$ from parameters space
- Select $\theta_j$ resulting in a 'good' split of the data
information gain criterion:

\[ I_j = I(S_j, S_j^L, S_j^R), \]  

where \( S_j^L = \{(x, y) \in S_j \mid h(x, \theta) = 0\} \), \( S_j^R = S_j \setminus S_j^L \) are splits
Training - Information Gain Criterion

- **Information gain criterion:**

\[ I_j = I(S_j, S_j^L, S_j^R), \]  

where \( S_j^L = \{(x, y) \in S_j \mid h(x, \theta) = 0\} \), \( S_j^R = S_j \setminus S_j^L \) are splits

- \( \theta_j = \arg\max_{\theta} I_j(S_j, \theta) \)
Training - Information Gain Criterion

- **information gain criterion:**

\[
l_j = I(S_j, S_j^L, S_j^R),
\]  

where \( S_j^L = \{(x, y) \in S_j \mid h(x, \theta) = 0\} \), \( S_j^R = S_j \setminus S_j^L \) are splits

- \( \theta_j = \arg\max_\theta l_j(S_j, \theta) \)

- for multiclass classification (\( \mathcal{Y} \subset \mathbb{Z} \)) the standard definition of information gain is:

\[
l_j = H(S_j) - \sum_{k \in \{L, R\}} \frac{|S_j^k|}{|S_j|} H(S_j^k)
\]  

\[ (2) \]
**Training - Information Gain Criterion**

- **information gain criterion:**

\[
l_j = I(S_j, S_j^L, S_j^R),
\]

where \( S_j^L = \{(x, y) \in S_j \mid h(x, \theta) = 0\} \), \( S_j^R = S_j \setminus S_j^L \) are splits.

- \( \theta_j = \text{argmax}_\theta I_j(S_j, \theta) \)

- for multiclass classification \((\mathcal{Y} \subset \mathbb{Z})\) the standard definition of information gain is:

\[
l_j = H(S_j) - \sum_{k \in \{L, R\}} \frac{|S_j^k|}{|S_j|} H(S_j^k)
\]

where \( H(S) \) is either the Shannon entropy \((H(S) = - \sum_y p_y \log(p_y))\) or alternatively the Gini impurity \((H(S) = - \sum_y p_y (1 - p_y))\)
- training stops when a maximum depth is reached or if information gain or training set size fall below fixed threshold
Training

- training stops when a maximum depth is reached or if information gain or training set size fall below fixed threshold
- single output $y \in \mathcal{Y}$ is assigned to a leaf node based on a problem specific ensemble model
combining results from multiple trees depends on a problem specific ensemble
combining results from multiple trees depends on a problem specific ensemble

classification $\rightarrow$ majority voting
combining results from multiple trees depends on a problem specific ensemble

- classification → majority voting
- regression → averaging
Structured Forests

- structured output space \( \mathcal{Y} \), e.g.:
Structured Forests

- structured output space $\mathcal{Y}$, e.g.:
- use of structured outputs in random forests presents following challenges:
Structured Forests

- structured output space $\mathcal{Y}$, e.g.: 
- use of structured outputs in random forests presents following challenges:
  - computing splits for structured output spaces of high dimensions/complexity is time consuming
structured output space $\mathcal{Y}$, e.g.:

use of structured outputs in random forests presents following challenges:

- computing splits for structured output spaces of high dimensions/complexity is time consuming
- it is unclear how to define information gain

solution: mapping (somehow) structured output space $\mathcal{Y}$ into multiclass space $C = \{1, \ldots, k\}$
Structured Forests

- structured output space $\mathcal{Y}$, e.g.:
- use of structured outputs in random forests presents following challenges:
  - computing splits for structured output spaces of high dimensions/complexity is time consuming
  - it is unclear how to define information gain
- solution: mapping (somehow) structured output space $\mathcal{Y}$ into multiclass space $\mathcal{C} = \{1, \ldots, k\}$
How to efficiently compute splits?
How to efficiently compute splits?

Can the task be transformed into a multiclass problem?

\[ \mathcal{Y} \rightarrow C = \{1, \ldots, k\} \]
Structured Edges Clustering

- How to efficiently compute splits?
- Can the task be transformed into a multiclass problem?

\[ Y \xrightarrow{?} C = \{1, ..., k\} \]

- start with an intermediate mapping:

\[ \Pi : Y \rightarrow Z \quad (3) \]
Structured Edges Clustering

- How to efficiently compute splits?
- Can the task be transformed into a multiclass problem?

\[ \mathcal{Y} \rightarrow C = \{1, \ldots, k\} \]

- start with an intermediate mapping:

\[ \Pi : \mathcal{Y} \rightarrow \mathcal{Z} \quad (3) \]

- \( z = \Pi(y) \) is a **long** binary vector, which encodes whether every pair of pixels in the \( y \) belongs to the same or different segment
dimension of vectors $z \in \mathcal{Z}$ is reduced by PCA to $m = 5$, and clustering (k-means) splits $\mathcal{Z}$ into $k = 2$ clusters
dimension of vectors $z \in \mathcal{Z}$ is reduced by PCA to $m = 5$, and clustering (k-means) splits $\mathcal{Z}$ into $k = 2$ clusters

$Y \xrightarrow{\text{pairs}} \mathcal{Z} \xrightarrow{\text{PCA, k-means}} C$
Structured Edges Clustering

- The dimension of vectors $z \in \mathcal{Z}$ is reduced by PCA to $m = 5$, and clustering (k-means) splits $\mathcal{Z}$ into $k = 2$ clusters.

$$\mathcal{Y} \xrightarrow{\text{pairs}} \mathcal{Z} \xrightarrow{\text{PCA, k-means}} C$$

$I_j$ - multiclass case
since the elements of $\mathcal{Y}$ are of size $16 \times 16$, the dimension of $\mathcal{Z}$ is $\binom{256}{2}$
Structured Edges Clustering

- since the elements of $\mathcal{Y}$ are of size $16 \times 16$, the dimension of $\mathcal{Z}$ is $\binom{256}{2}$
- too expensive $\rightarrow$ randomly sample $m = 256$ dimensions of $\mathcal{Z}$
Structured Edges Clustering

- since the elements of $\mathcal{Y}$ are of size $16 \times 16$, the dimension of $\mathcal{Z}$ is $(256^2)$
- too expensive $\rightarrow$ randomly sample $m = 256$ dimensions of $\mathcal{Z}$
- $\mathcal{Y}$ sampled pairs $\rightarrow$ $\mathcal{Z}$ PCA, k-means $\rightarrow$ $C$
during training, we need to assign a single prediction to a leaf node.
Ensemble model

- during training, we need to assign a single prediction to a leaf node
- during testing, we need to combine multiple predictions into one
Ensemble model

- during training, we need to assign a single prediction to a leaf node
- during testing, we need to combine multiple predictions into one
- to select a single output from a set $y_1, ..., y_k \in Y$:

\[
\begin{align*}
z_i &= \prod_{y_i} \\
\text{select } y_k^* \text{ such that } k^* &= \arg\min_k \sum_{i,j} (z_{k^*j} - z_{ij})^2
\end{align*}
\]

a domain specific ensemble model (for edge map):

\[
y_i' = E[y_i']
\]
Ensemble model

- during training, we need to assign a single prediction to a leaf node
- during testing, we need to combine multiple predictions into one
- to select a single output from a set \( y_1, \ldots, y_k \in \mathcal{Y} \):
  - compute \( z_i = \prod y_i \)
during training, we need to assign a single prediction to a leaf node

during testing, we need to combine multiple predictions into one
to select a single output from a set $y_1, \ldots, y_k \in \mathcal{Y}$:

- compute $z_i = \Pi y_i$
- select $y_{k^*}$ such that $k^* = \arg\min_k \sum_{i,j} (z_{k,j} - z_{i,j})^2$ (medoid)
Ensemble model

- during training, we need to assign a single prediction to a leaf node
- during testing, we need to combine multiple predictions into one
- to select a single output from a set $y_1, ..., y_k \in \mathcal{Y}$:
  - compute $z_i = \Pi y_i$
  - select $y_{k*}$ such that $k^* = \arg\min_k \sum_{i,j} (z_{k,j} - z_{i,j})^2$ (medoid)
- a domain specific ensemble model (for edge map): $y'_{k*} = E[y'_i]$
input:
input:
- image patch $32 \times 32$, sampled into 7228 features
input:

- image patch $32 \times 32$, sampled into 7228 features
- corresponding segmentation mask $16 \times 16$
Structured Forest Training - Overview

- **input:**
  - image patch $32 \times 32$, sampled into 7228 features
  - corresponding segmentation mask $16 \times 16$
  - randomly selected features per split
Structured Forest Training - Overview

- **input:**
  - image patch $32 \times 32$, sampled into 7228 features
  - corresponding segmentation mask $16 \times 16$
  - randomly selected features per split
  - segmentation masks $\rightarrow$ clusters $\rightarrow$ split information gain
input:
- image patch $32 \times 32$, sampled into 7228 features
- corresponding segmentation mask $16 \times 16$
- randomly selected features per split
- segmentation masks $\rightarrow$ clusters $\rightarrow$ split information gain
- medoid $\rightarrow$ segmentation mask
Structured Forest Training - Overview

- **input:**
  - image patch $32 \times 32$, sampled into 7228 features
  - corresponding segmentation mask $16 \times 16$
  - randomly selected features per split
  - segmentation masks $\rightarrow$ clusters $\rightarrow$ split information gain
  - medoid $\rightarrow$ segmentation mask
  - averaging $\rightarrow$ soft edge map

Fast edge detection

June 6, 2018 14 / 22
Structured Forest Testing

![Structured Forest Diagram]

Fast edge detection
Structured Forest Testing

Fast edge detection
Structured Forest Testing

Fast edge detection

June 6, 2018
Structured Forest Testing

Decision:
Why is the method so fast?
Efficiency

- Why is the method so fast?
- a single decision tree $\rightarrow$ lots of pixel information
Efficiency

- Why is the method so fast?
  - a single decision tree $\rightarrow$ lots of pixel information
  - lots of pixel information $\rightarrow$ a small random forest $\rightarrow$ fast evaluation
Multiscale Detection

- multiscale version takes original, double, and half resolution of an input image
Multiscale Detection

- multiscale version takes original, double, and half resolution of an input image
- resulting three edge maps are averaged
Multiscale Detection

- multiscale version takes original, double, and half resolution of an input image
- resulting three edge maps are averaged
- slower ($\times 5$)
Multiscale Detection

- multiscale version takes original, double, and half resolution of an input image
- resulting three edge maps are averaged
- slower ($\times 5$)
- improved edge quality
individual predictions are noisy and do not perfectly align to each other or the underlying image data
individual predictions are noisy and do not perfectly align to each other or the underlying image data

- sharpening takes individual prediction $y$ and produces a new mask that better aligns it to the image patch $x$:
individual predictions are noisy and do not perfectly align to each other or the underlying image data.

- sharpening takes individual prediction $y$ and produces a new mask that better aligns it to the image patch $x$:
  - compute mean segment $s$ color, $\mu_s = E[x(j) \mid y(j) = s]$
individual predictions are noisy and do not perfectly align to each other or the underlying image data

sharpening takes individual prediction $y$ and produces a new mask that better aligns it to the image patch $x$:

- compute mean segment $s$ color, $\mu_s = E[x(j) \mid y(j) = s]$
- change pixel $j$ assignment if the pixel color $x(j)$ is closer to different segment ($s^* = \arg\min_s \|\mu_s - x(j)\|$) and such segment labeling is in 4-connected vicinity
individual predictions are noisy and do not perfectly align to each other or the underlying image data

- sharpening takes individual prediction $y$ and produces a new mask that better aligns it to the image patch $x$:
  - compute mean segment $s$ color, $\mu_s = E[x(j) \mid y(j) = s]$
  - change pixel $j$ assignment if the pixel color $x(j)$ is closer to different segment ($s^* = \arg\min_s ||\mu_s - x(j)||$) and such segment labeling is in 4-connected vicinity

- sharpening can be repeated multiple times, Dollar & Zitnick claims that in practice two steps suffice
Parameter Sweeps

(a) $m$ (size of $\mathcal{Z}$)

(b) $k$ (size of $\mathcal{C}$)
Parameter Sweeps

(c) # train patches $\times 10^4$

(d) # train images
Parameter Sweeps

(e) fraction ‘positives’

ODS x 100

72

70

68

66

0.2 0.3 0.4 0.5 0.6 0.7 0.8
Parameter Sweeps

(g) # decision trees

(h) max tree depth
<table>
<thead>
<tr>
<th></th>
<th>ODS</th>
<th>OIS</th>
<th>AP</th>
<th>R50</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human</td>
<td>.80</td>
<td>.80</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SE</td>
<td>.73</td>
<td>.75</td>
<td>.77</td>
<td>.90</td>
<td>30</td>
</tr>
<tr>
<td>SE+SH</td>
<td>.74</td>
<td>.76</td>
<td>.79</td>
<td>.93</td>
<td>12.5</td>
</tr>
<tr>
<td>SE+MS</td>
<td>.74</td>
<td>.76</td>
<td>.78</td>
<td>.90</td>
<td>6</td>
</tr>
<tr>
<td>SE+MS+SH</td>
<td>.75</td>
<td>.77</td>
<td>.80</td>
<td>.93</td>
<td>2.5</td>
</tr>
</tbody>
</table>
### Table 4. Results on BSDS500. *BSDS300 results, †GPU time

<table>
<thead>
<tr>
<th>Method</th>
<th>ODS</th>
<th>OIS</th>
<th>AP</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human</td>
<td>0.80</td>
<td>0.80</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Canny</td>
<td>0.600</td>
<td>0.640</td>
<td>0.580</td>
<td>15</td>
</tr>
<tr>
<td>Felz-Hutt [9]</td>
<td>0.610</td>
<td>0.640</td>
<td>0.560</td>
<td>10</td>
</tr>
<tr>
<td>BEL [5]</td>
<td>0.660*</td>
<td>-</td>
<td>-</td>
<td>1/10</td>
</tr>
<tr>
<td>gPb-owt-ucm [1]</td>
<td>0.726</td>
<td>0.757</td>
<td>0.696</td>
<td>1/240</td>
</tr>
<tr>
<td>Sketch Tokens [24]</td>
<td>0.727</td>
<td>0.746</td>
<td>0.780</td>
<td>1</td>
</tr>
<tr>
<td>SCG [31]</td>
<td>0.739</td>
<td>0.758</td>
<td>0.773</td>
<td>1/280</td>
</tr>
<tr>
<td><strong>SE-Var [6]</strong></td>
<td>0.746</td>
<td>0.767</td>
<td>0.803</td>
<td>2.5</td>
</tr>
<tr>
<td>OEF [13]</td>
<td>0.749</td>
<td>0.772</td>
<td>0.817</td>
<td>-</td>
</tr>
<tr>
<td>DeepNets [21]</td>
<td>0.738</td>
<td>0.759</td>
<td>0.758</td>
<td>1/5†</td>
</tr>
<tr>
<td>N4-Fields [10]</td>
<td>0.753</td>
<td>0.769</td>
<td>0.784</td>
<td>1/6†</td>
</tr>
<tr>
<td>DeepEdge [2]</td>
<td>0.753</td>
<td>0.772</td>
<td>0.807</td>
<td>1/10³†</td>
</tr>
<tr>
<td>CSCNN [19]</td>
<td>0.756</td>
<td>0.775</td>
<td>0.798</td>
<td>-</td>
</tr>
<tr>
<td>DeepContour [34]</td>
<td>0.756</td>
<td>0.773</td>
<td>0.797</td>
<td>1/30†</td>
</tr>
<tr>
<td><strong>HED (ours)</strong></td>
<td><strong>0.782</strong></td>
<td><strong>0.804</strong></td>
<td><strong>0.833</strong></td>
<td>2.5†, 1/12</td>
</tr>
</tbody>
</table>
Conclusion

- realtime
Conclusion

- realtime
- structured learning
Conclusion

- realtime
- structured learning
- suitable preprocessing step for methods requiring speed