Object Detection using Haar-like Features

CS 395T: Visual Recognition and Search
Harshdeep Singh
The Detector

• Using boosted cascades of Haar-like features

• Proposed by [Viola, Jones 2001]

• Implementation available in OpenCV
Haar-like features

- feature = \( w_1 \times \text{RecSum}(r_1) + w_2 \times \text{RecSum}(r_2) \)
- Weights can be positive or negative
- Weights are directly proportional to the area
- Calculated at every point and scale
Weak Classifier

• A weak classifier \( h(x, f, p, \theta) \) consists of
  – feature \( f \)
  – threshold \( \theta \)
  – polarity \( p \), such that

\[
h(x, f, p, \theta) = \begin{cases} 
1 & \text{if } p f(x) < p \theta \\
0 & \text{otherwise} 
\end{cases}
\]

• Requirement
  – Should perform better than random chance
Attentional Cascade

- Initial stages have less features (faster computation)
- More time spent on evaluating more promising sub-windows
Cascade Creation - Walkthrough

• Input:
  – \( f \) = Maximum acceptable false positive rate per layer (0.5)
  – \( d \) = Minimum acceptable detection rate per layer (0.995)
  – \( F_{\text{target}} \) = Target overall false positive rate
    • Or maximum number of stages in the cascade
    • For \( n_{\text{Stages}} = 14 \), \( F_{\text{target}} = f^{n_{\text{Stages}}} = 6.1 \times 10^{-5} \)
  – \( P \) = Set of positive examples
    • 200 distorted versions of a synthetic image
  – \( N \) = Set of negative examples
    • 100 images from BACKGROUND_Google category of Caltech 101 dataset
Cascade Creation - Walkthrough

\[ F_0 = 1 \]
\[ i = 0 \]
while \( F_i > F_{\text{target}} \) and \( i < n\text{Stages} \)
\[ i = i + 1 \]

Train Classifier for stage \( i \)
- Initialize Weights
- Normalize Weights
- Pick the (next) best weak classifier
- Update Weights
- Evaluate \( f_i \)
if \( f_i > f \)
    go back to Normalize Weights
Combine weak classifiers to form the strong stage classifier
Evaluate \( F_i \)
Cascade Creation - Walkthrough

\[ F_0 = 1 \]
\[ i = 0 \]
\[ \text{while } F_i > F_{\text{target}} \text{ and } i < n\text{Stages} \]
\[ i = i + 1 \]

Train Classifier for stage \( i \)

- Initialize Weights
- Normalize Weights
- Pick the (next) best weak classifier
- Update Weights
- Evaluate \( f_i \)
- if \( f_i > f \) go back to Normalize Weights

Combine weak classifiers to form the strong stage classifier
Evaluate \( F_i \)

\[ F_i = \text{False alarm rate of the cascade with } i \text{ stages} \]
Cascade Creation - Walkthrough

\[ F_0 = 1 \]
\[ i = 0 \]

while \( F_i > F_{\text{target}} \) and \( i < n\text{Stages} \)

\[ i = i + 1 \]

Train Classifier for stage \( i \)

- Initialize Weights
- Normalize Weights
- Pick the (next) best weak classifier
- Update Weights
- Evaluate \( f_i \)

if \( f_i > f \)

    go back to Normalize Weights

Combine weak classifiers to form the strong stage classifier

Evaluate \( F_i \)

\( F_i \) = False alarm rate of the cascade with \( i \) stages
Cascade Creation - Walkthrough

\[ F_0 = 1 \]
\[ i = 0 \]

while \( F_i > F_{\text{target}} \) and \( i < n\text{Stages} \)

\[ i = i + 1 \]

Train Classifier for stage \( i \)

Initialize Weights
Normalize Weights
Pick the (next) best weak classifier
Update Weights
Evaluate \( f_i \)
if \( f_i > f \)

go back to Normalize Weights
Combine weak classifiers to form the strong stage classifier
Evaluate \( F_i \)

Weight for each

<table>
<thead>
<tr>
<th></th>
<th>( \text{positive sample} )</th>
<th>( \frac{0.5}{m} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \text{negative sample} )</td>
<td>( \frac{0.5}{n} )</td>
</tr>
</tbody>
</table>

\( m \) – number of positive samples (200)
\( n \) – number of negative samples (100)
Cascade Creation - Walkthrough

\[ F_0 = 1 \]
\[ i = 0 \]
while \( F_i > F_{\text{target}} \) and \( i < n_{\text{Stages}} \)
\[ i = i + 1 \]
Train Classifier for stage \( i \)
  Initialize Weights
  Normalize Weights
  Pick the (next) best weak classifier
  Update Weights
  Evaluate \( f_i \)
  if \( f_i > f \)
    go back to Normalize Weights
  Combine weak classifiers to form the strong stage classifier
  Evaluate \( F_i \)

Weight for each
- positive sample: \( 0.5/m \)
- negative sample: \( 0.5/n \)

\( m \) – number of positive samples (200)
\( n \) – number of negative samples (100)
Cascade Creation - Walkthrough

\[ F_0 = 1 \]
\[ i = 0 \]

while \( F_i > F_{\text{target}} \) and \( i < n\text{Stages} \)
\[ i = i + 1 \]

Train Classifier for stage \( i \)
- Initialize Weights
- Normalize Weights

Pick the (next) best weak classifier
- Update Weights
- Evaluate \( f_i \)
- if \( f_i > f \)
  - go back to Normalize Weights

Combine weak classifiers to form the strong stage classifier
- Evaluate \( F_i \)

The one with minimum error
\[
\epsilon_t = \min_{f,p,q} \sum_i w_i |h(x_i, f, p, q) - y_i| 
\]
\[ \epsilon_t = 0.005 \]
Error minimization

T*: Total sum of weights of positive examples
T: Total sum of weights of negative examples
S*: Total sum of weights of positive examples below the current one
S: Total sum of weights of negative examples below the current one

e_1 = S^* + (T^- S^-)
e_2 = S^- + (T^* - S^*)
e = \min(e_1, e_2)
Cascade Creation - Walkthrough

\[ F_0 = 1 \]
\[ i = 0 \]

while \( F_i > F_{\text{target}} \) and \( i < n\text{Stages} \)
\[ i = i + 1 \]

Train Classifier for stage \( i \)
- Initialize Weights
- Normalize Weights
- Pick the (next) best weak classifier
- Update Weights
- Evaluate \( f_i \)
  - if \( f_i > f \)
    - go back to Normalize Weights
- Combine weak classifiers to form the strong stage classifier
- Evaluate \( F_i \)

\[ w_{t+1, i} = w_{t, i} \beta_{t}^{1 - \epsilon_{t}} \]

\( \epsilon_{i} = 0 \), if example \( x_{i} \) is classified correctly
\( \epsilon_{i} = 1 \), otherwise

\[ \beta_{t} = \frac{\epsilon_{t}}{1 - \epsilon_{t}} \]
Cascade Creation - Walkthrough

$F_0 = 1$

$f_i =$ number of negative samples that were detected by this stage/ total number of negative samples

$i = 0$

$F_i > F_{\text{target}}$ and $i < n\text{Stages}$

$i = i + 1$

Train Classifier for stage $i$

- Initialize Weights
- Normalize Weights
- Pick the (next) best weak classifier
- Update Weights

Evaluate $f_i$

if $f_i > f$

- go back to Normalize Weights

Combine weak classifiers to form the strong stage classifier

Evaluate $F_i$
Cascade Creation - Walkthrough

\[ F_0 = 1 \]
\[ i = 0 \]

while \( F_i > F_{\text{target}} \) and \( i < n\text{Stages} \)
\[ i = i + 1 \]

Train Classifier for stage \( i \)
  Initialize Weights
  Normalize Weights
  Pick the (next) best weak classifier
  Update Weights
  Evaluate \( f_i \)
  if \( f_i > f \)
    go back to Normalize Weights
  Combine weak classifiers to form the strong stage classifier
  Evaluate \( F_i \)

How far will you go to get down to \( f \)?
Cascade Creation - Walkthrough

$F_0 = 1$

$i = 0$

while $F_i > F_{\text{target}}$ and $i < n_{\text{Stages}}$

\[ i = i + 1 \]

Train Classifier for stage $i$

- Initialize Weights
- Normalize Weights
- Pick the (next) best weak classifier
- Update Weights
- Evaluate $f_i$
- if $f_i > f$
  go back to Normalize Weights

Combine weak classifiers to form the strong stage classifier

Evaluate $F_i$

\[
C(x) = \begin{cases} 
1 \sum_{t=1}^{T} \alpha_t h_t(x) & \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \ \text{otherwise} \\
0 & 
\end{cases}
\]

\[
\alpha_t = \log \frac{1}{\beta_t} \quad \beta_t = \frac{\epsilon_t}{1 - \epsilon_t}
\]

Weight is inversely proportional to the training error

**Paper**

Decrease threshold until the classifier has a detection rate of at least $d$

**OpenCV**

1. For each positive sample, find the weighted sum of all features
2. Sort these values
3. Set threshold = sorted_values[$(1-d) * |P|$]
Cascade Creation - Walkthrough

\[ F_0 = 1 \]

\[ i = 0 \]

while \( F_i > F_{\text{target}} \) and \( i < n_{\text{Stages}} \):

\[ i = i + 1 \]

Train Classifier for stage \( i \)

- Initialize Weights
- Normalize Weights
- Pick the (next) best weak classifier
- Update Weights
- Evaluate \( f_i \)
- if \( f_i > f \)
  - go back to Normalize Weights
- Combine weak classifiers to form the strong stage classifier
- Evaluate \( F_i \)
If \( f \) (maximum false alarm rate) is increased from 0.5 to 0.7, a cascade with only the first two stages is created.
Which features actually get selected?

Stage 0

Stage 1

Stage 21

... 10 more

... 206 more
Other Objects?

Caltech 101 dataset

“Most images have little or no clutter. The objects tend to be centered in each image. Most objects are presented in a stereotypical pose.”
Training

Hand label ROI in 40/64 images

Generate 1000 random distortions of a representative image

Negative samples taken from BACKGROUND_Google category of Caltech 101

Some features that get selected
Performance
Variation in Training Images

High accuracy categories

Low accuracy categories
Skin Color Approximation

- To filter results of face detector
- Derived from [Bradsky 1998]
- Template Image
  - Patches of faces of different subjects under varying lighting conditions
Skin Color Approximation

Create hue histogram

Normalize [0 – 255]

Back Projection

RGB -> HSV

Face image

\[ S = \text{Sum of pixel values in the back-projection} / \text{Area} \]

S > Threshold?

\[ S \geq \text{Threshold?} \]

N

Y
Result

Evaluated on 435 face images in the Caltech 101 dataset
When does it help?

Without skin filter  With skin filter
Rotated Features

1. Edge features
   (a) (b) (c) (d)

2. Line features
   (a) (b) (c) (d) (e) (f) (g) (h)

3. Center-surround features
   (a) (b)

4. Not used
   
---

An Extended Set of Haar-like Features for Rapid Object Detection, Lienhart and Maydt
Results

- **Precision with Basic Features**
- **Precision with Extended Features**
- **Recall with Basic Features**
- **Recall with Extended Features**

The graph shows the performance metrics across different categories such as Accordion, Airplanes, Butterfly, Faces_easy, and Soccer_ball.
Lessons

1. Viola Jones’ technique worked pretty well for faces and some other categories like airplanes and car_sides.
2. Did not work well with many other categories. A large number of false positives.
3. Accuracy depends largely on the amount of variation in training and test images.
4. In some cases, the training algorithm is not able to go below the maximum false alarm rate of a layer, even with a very large number of features.
5. Selected features for the first few stages are more “intuitive” than the later ones.
6. Skin color can be used to increase the precision of face detection at the cost of recall. Dependent on illumination.
7. Using rotated features can increase accuracy but not too much.
8. Training classifiers is slow! Let OpenCV use as much memory as you have.