Object Detection, Method of Viola and Jones
Integral Image, Boosting, and Cascade

Michael Chang
Detection Based On Rectangle Feature

- Rather than use pixels directly, form rectangles inside a 24 x 24 window

Figure 1: Example rectangle features shown relative to the enclosing detection window. The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles. Two-rectangle features are shown in (A) and (B). Figure (C) shows a three-rectangle feature, and (D) a four-rectangle feature.

- Motivation comes from Haar basis functions
Haar Basis Function

- **Haar Wavelet**

\[
\psi(t) = \begin{cases} 
1 & 0 \leq t < 1/2, \\
-1 & 1/2 \leq t < 1, \\
0 & \text{otherwise.}
\end{cases}
\]

- So Viola and Jones are sort of using a 2D Haar Basis
Haar-like Feature

- So the rectangles are the basis
  - The set of overall rectangles containing smaller rectangles is the “function” being represented
  - The coefficients in the basis are the sums of the individual rectangles
- The classifier is based on forming simple “features” out of the rectangles
  - **Two-rectangle feature** is difference between sum of pixels within two rectangles
  - **Three-rectangle feature** is sum within two outside rectangles subtracted from sum in center rectangle
  - **Four-rectangle feature** is difference between diagonal pairs of rectangles
- They claim features better capture particular domain knowledge and is faster than pixel-based
- Since window is 24 x 24, must compute 45,396 features
Rather than compute the sum of all possible rectangles, use integral image

\[
l(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y')
\]
Can be computed in one pass:
\[ l(x, y) = i(x, y) + l(x - 1, y) + l(x, y - 1) - l(x - 1, y - 1) \]

Any rectangle can be computed with four array references:

Suppose,
\[ A = (x_0, y_1), \quad B = (x_1, y_1), \quad C = (x_1, y_0), \quad D = (x_0, y_0), \]
Then,
\[ \sum_{x_0 \leq x \leq x_1, \ y_0 \leq y \leq y_1} i(x, y) = l(C) + l(A) - l(B) - l(D). \]
The Method of Viola and Jones

1. Train classification function based on features via AdaBoost
2. Construct cascade of classifiers to make detection more efficient
The Method of Viola and Jones

1. Train classification function based on features via AdaBoost
2. Construct cascade of classifiers to make detection more efficient
Training Classification Function

1. Start with weak classifiers based only on one feature:

\[ h_j(x) = \begin{cases} 
1 & \text{if } p_j f_j(x) < p_j \theta_j \\
0 & \text{otherwise.} 
\end{cases} \]

where \( h_j(x) \) is weak classifier, \( f_j \) is feature, \( \theta_j \) is threshold, \( p_j \) is parity (direction of inequality), and \( x \) is 24 x 24 sub-window.

2. Use AdaBoost to determine small set of good classification features.

3. Combine into large classification function function using a weighted majority vote.
The Training Algorithm

- Given example images \((x_1, y_1), \ldots, (x_n, y_n)\) where \(y_i = 0, 1\) for negative and positive examples respectively.
- Initialize weights \(w_{1,i} = \frac{1}{2m}, \frac{1}{2l}\) for \(y_i = 0, 1\) respectively, where \(m\) and \(l\) are the number of negatives and positives respectively.
- For \(t = 1, \ldots, T\):
  1. Normalize the weights,
     \[ w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}} \]
     
     so that \(w_t\) is a probability distribution.
  2. For each feature, \(j\), train a classifier \(h_j\) which is restricted to using a single feature. The error is evaluated with respect to \(w_t\), \(e_j = \sum_i w_i |h_j(x_i) - y_i|\).
  3. Choose the classifier, \(h_t\), with the lowest error \(e_t\).
  4. Update the weights:
     \[ w_{t+1,i} = w_{t,i} \beta_t^{1-e_i} \]
     
     where \(e_i = 0\) if example \(x_i\) is classified correctly, \(e_i = 1\) otherwise, and \(\beta_t = \frac{\epsilon_t}{1-\epsilon_t}\).

- The final strong classifier is:
  \[ h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases} \]

where \(\alpha_t = \log \frac{1}{\beta_t}\)

Table 1: The boosting algorithm for learning a query online. \(T\) hypotheses are constructed each using a single feature. The final hypothesis is a weighted linear combination of the \(T\) hypotheses where the weights are inversely proportional to the training errors.

Note that as \(\epsilon_t \to 0\), \(\beta_t \to 0\) which implies \(\alpha_t \to \infty\)
The Cascade

- AdaBoost selected two strong features in face detection but not good enough
- Correctly detects about 100% of faces but only rejects 60% non-faces
- Use as a first classifier in a cascade where each subsequent one has more features and has high detection still but less false-positives
The Cascade Problem

- Design cascade detector that has good detection rate, 85-95% for faces and low false positive rates ($10^{-5}$ or $10^{-6}$)
- It also must minimize computations
- False positive rate of cascade is

$$F = \prod_{i=1}^{K} f_i$$

where $F$ is false positive rate, $K$ is number of classifiers, and $f_i$ is false positive rate of $ith$ classifier

- Detection rate of cascade is

$$D = \prod_{i=1}^{K} d_i$$

where $d_i$ is detection rate of $ith$ classifier
The Cascade Problem

- Given expression for detection rate, it must be kept high since it is the product of numbers < 1 and goal is high
  - 10 stage classifier with each stage 0.99 detection rate has overall 0.9 detection rate
- But false positive rate only needs to be 0.3
- Most image sub-windows should fail early but only rare ones make it to classifier with most features
- Number of expected features which are evaluated is:

\[ N = n_0 + \sum_{i=1}^{K} \left( n_i \prod_{j<i} p_j \right) \]

where \( N \) is expected number of features evaluated, \( p_i \) is positive rate of \( ith \) classifier, and \( n_i \) are number of features in \( ith \) classifier
The Cascade Algorithm

- User selects values for $f$, the maximum acceptable false positive rate per layer and $d$, the minimum acceptable detection rate per layer.
- User selects target overall false positive rate, $F_{\text{target}}$.
- $P$ = set of positive examples
- $N$ = set of negative examples
- $F_0 = 1.0; D_0 = 1.0$
- $i = 0$
- while $F_i > F_{\text{target}}$
  - $i \leftarrow i + 1$
  - $n_i = 0; F_i = F_{i-1}$
  - while $F_i > f \times F_{i-1}$
    * $n_i \leftarrow n_i + 1$
    * Use $P$ and $N$ to train a classifier with $n_i$ features using AdaBoost
    * Evaluate current cascaded classifier on validation set to determine $F_i$ and $D_i$.
    * Decrease threshold for the ith classifier until the current cascaded classifier has a detection rate of at least $d \times D_{i-1}$ (this also affects $F_i$)
  - $N \leftarrow \emptyset$
  - If $F_i > F_{\text{target}}$ then evaluate the current cascaded detector on the set of non-face images and put any false detections into the set $N$

Table 2: The training algorithm for building a cascaded detector.

- Hard to optimize
- Another step in algorithm to check whether a stage can be removed or combine subsets?
They created a 32 layer cascade of classifiers which had 4297 features.
Classifiers trained with 4916 faces and 10,000 non-face sub-windows using Adaboost.
Training time was weeks (computer from years ago).
On MIT+CMU test set, 8 features out of 4297 are evaluated on average per sub-window.
On old computer, detector processes 384 x 288 image in .067 seconds.
Figure 9: ROC curves for our face detector on the MIT+CMU test set. The detector was run once using a step size of 1.0 and starting scale of 1.0 (75,081,800 sub-windows scanned) and then again using a step size of 1.5 and starting scale of 1.25 (18,901,947 sub-windows scanned). In both cases a scale factor of 1.25 was used.
They needed to do pre-processing to minimize effect of different lighting conditions

- Normalized variance
- They were able to compute variance from integral images
- Could normalize by operating on feature values rather than pixels

Needed to scale detector

Shifting detector across locations more than one pixel at a time decreases detection rate

Had multiple detections around each face

- Simply partitioned set of detections into disjoint subsets
- Two detections were in same subset if bounding regions overlap
- Each partition is one detection and corners are averaged between all detections
Summary

- Classify objects based on “Haar-like” features
- Train classifiers based on AdaBoost algorithm
- Use cascading to make detector more efficient