Rectangle Filters and AdaBoost

CSE 6367 – Computer Vision
Vassilis Athitsos
University of Texas at Arlington

A Generalized View of Classifiers

- We have studied two face detection methods:
  - Normalized correlation.
  - PCA.
- Each approach exhaustively evaluates all image suwbwindows.
- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.

Features and Classifiers

- Our goal, in the next slides, is to get a better understanding of:
  - What is a feature?
  - What is a classifier?

Normalized Correlation Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- What is a feature here?

Normalized Correlation Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- What is a feature here?
- Two possible answers:
  - Each pixel value is a feature.
  - The feature is the result of normalized correlation with the template.

Normalized Correlation Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- What is the score of each subwindow?

Normalized Correlation Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- What is the score of each subwindow?
  - The result of normalized correlation with the template.
  - Arguably, the score is the feature itself.

Normalized Correlation Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- How does the decision depend on the score?

Normalized Correlation Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- How does the decision depend on the score?
  - In **`find_template`**, faces are the top N scores.
    - N is an argument to the **`find_template`** function.
  - An alternative, is to check if score > threshold.
    - Then we must choose a threshold instead of N.

PCA-based Detection Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- What is a feature here?

PCA-based Detection Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- What is a feature here?
  - The result of the pca_projection function.
  - In other words, the K numbers that describe the projection of the subwindow on the space defined by the top K eigenfaces.

PCA-based Detection Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- What is the score of each subwindow?

PCA-based Detection Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- What is the score of each subwindow?
  - The result of the pca_score function.
  - The sum of differences between:
    - the original subwindow W, and
    - pca_backprojection(pca_projection(W)).

PCA-based Detection Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- How does the decision depend on the score?

PCA-based Detection Analysis

- Each subwindow is evaluated in three steps:
  - First step: extract *features.*
    - Feature: a piece of information extracted from a pattern.
  - Compute a score based on the features.
  - Make a decision based on the score.
- How does the decision depend on the score?
  - In **`find_faces`**, faces are the top N scores.
    - N is an argument to the **`find_faces`** function.
  - An alternative, is to check if score > threshold.
    - Then we must choose a threshold instead of N.

Defining a Classifier

- Choose features.
- Choose a scoring function.
- Choose a decision process.
  - The last part is usually straightforward.
  - We pick the top N scores, or we apply a threshold.
- Therefore, the two crucial components are:
  - Choosing features.
  - Choosing a scoring function.

What is a Feature?

- Any information extracted from an image.
- The number of possible features we can define is enormous.
- Any function F we can define that takes in an image as an argument and produces one or more numbers as output defines a feature.
  - Correlation with a template defines a feature.
  - Projecting to PCA space defines features.
  - More examples?

What is a Feature?

- Any information extracted from an image.
- The number of possible features we can define is enormous.
- Any function F we can define that takes in an image as an argument and produces one or more numbers as output defines a feature.
  - Correlation with a template defines a feature.
  - Projecting to PCA space defines features.
  - Average intensity.
  - Std of values in window.

Boosted Rectangle Filters

- Boosting is a machine learning method.
- Rectangle filters provide us with an easy way to define features.
- Boosted rectangle filters is an extremely popular computer vision method that answers the basic two questions in classifier design:
  - What features do we use? Rectangle filters.
  - What scoring function do we use? The result of boosting.

Boosted Rectangle Filters

- Boosted rectangle filters is an extremely popular computer vision method that answers the basic two questions in classifier design:
  - What features do we use? Rectangle filters.
  - What scoring function do we use? The result of boosting.
- The popularity of this method is due to two factors:
  - Can be applied in lots of problems.
  - Works well in lots of problems.

What is a Rectangle Filter?

- ```
  [1 1 1 1 -1 -1 -1 -1
   1 1 1 1 -1 -1 -1 -1
   1 1 1 1 -1 -1 -1 -1
   1 1 1 1 -1 -1 -1 -1]
  ```

- ```
  [1 1 1 -2 -2 -2 1 1 1
   1 1 1 -2 -2 -2 1 1 1
   1 1 1 -2 -2 -2 1 1 1
   1 1 1 -2 -2 -2 1 1 1
   1 1 1 -2 -2 -2 1 1 1
   1 1 1 -2 -2 -2 1 1 1]
  ```

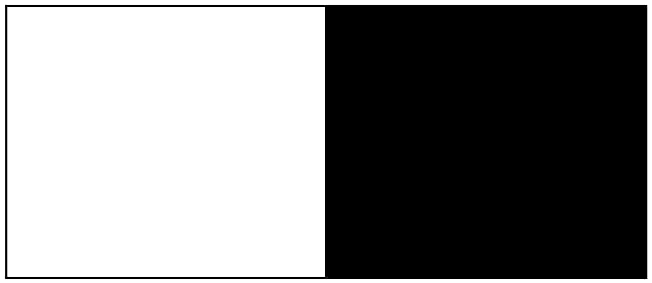
- ```
  [ 1  1  1  1  1  1
    1  1  1  1  1  1
    1  1  1  1  1  1
   -1 -1 -1 -1 -1 -1
   -1 -1 -1 -1 -1 -1
   -1 -1 -1 -1 -1 -1]
  ```

- ```
  [ 1  1  1 -1 -1 -1
    1  1  1 -1 -1 -1
    1  1  1 -1 -1 -1
    1  1  1 -1 -1 -1
   -1 -1 -1  1  1  1
   -1 -1 -1  1  1  1
   -1 -1 -1  1  1  1
   -1 -1 -1  1  1  1
  ```

Five Different Basic Shapes

- Type 1:
  - Two areas, horizontal.

```
[1 1 1 1 -1 -1 -1 -1
 1 1 1 1 -1 -1 -1 -1
 1 1 1 1 -1 -1 -1 -1
 1 1 1 1 -1 -1 -1 -1]
```



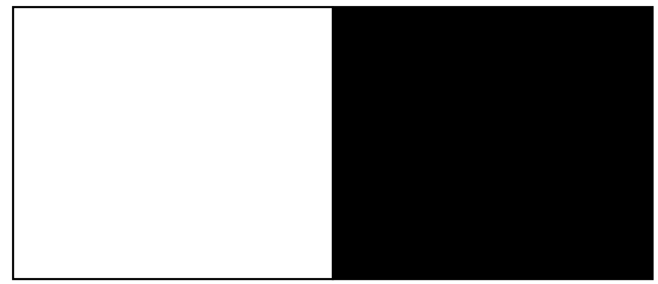
white: value = 1
black: value = -1

- To define:
  - Specify rectangle size:
    - number of rows
    - number of cols.

Five Different Basic Shapes

- Type 1:
  - Two areas, horizontal.

```
[1 1 1 1 -1 -1 -1 -1
 1 1 1 1 -1 -1 -1 -1
 1 1 1 1 -1 -1 -1 -1
 1 1 1 1 -1 -1 -1 -1]
```



white: value = 1
black: value = -1

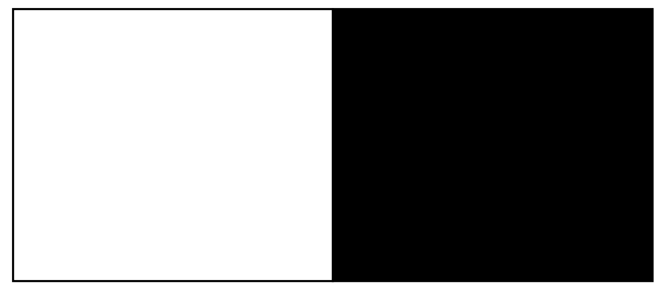
Have we ever used anything similar?

- To define:
  - Specify rectangle size:
    - number of rows
    - number of cols.

Five Different Basic Shapes

- Type 1:
  - Two areas, horizontal.

```
[1 1 1 1 -1 -1 -1 -1
 1 1 1 1 -1 -1 -1 -1
 1 1 1 1 -1 -1 -1 -1
 1 1 1 1 -1 -1 -1 -1]
```



white: value = 1
black: value = -1

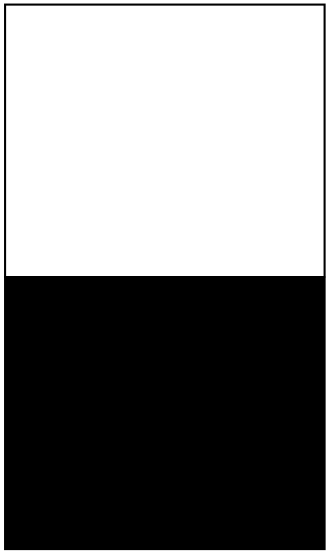
- To define:
  - Specify rectangle size:
    - number of rows
    - number of cols.

Have we ever used anything similar?
dx = [-1 0 -1] is a centered version of rectangle_filter1(1, 1).

Five Different Basic Shapes

- Type 2:
  - Two areas, vertical.

```
[ 1  1  1  1  1  1
  1  1  1  1  1  1
  1  1  1  1  1  1
 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1]
```



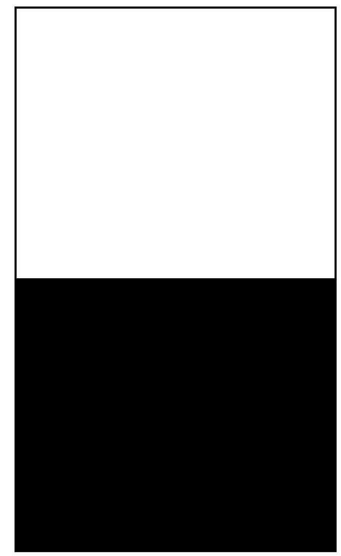
To define:

- Specify rectangle size:
  - number of rows
  - number of cols.

Five Different Basic Shapes

- Type 2:
  - Two areas, vertical.

```
[ 1  1  1  1  1  1
  1  1  1  1  1  1
  1  1  1  1  1  1
 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1]
```



To define:

- Specify rectangle size:
  - number of rows
  - number of cols.

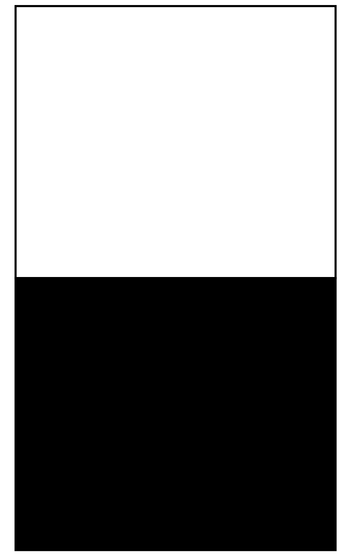
Have we ever used anything similar?

Five Different Basic Shapes

- Type 2:
  - Two areas, vertical.

```
[ 1  1  1  1  1  1
  1  1  1  1  1  1
  1  1  1  1  1  1
 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1]
```

To define:
  - Specify rectangle size:
    - number of rows
    - number of cols.



Have we ever used anything similar?
dy = [-1 0 -1]’ is a centered version of rectangle_filter2(1, 1).

Effect of Rectangle Size

- Larger rectangles → ?

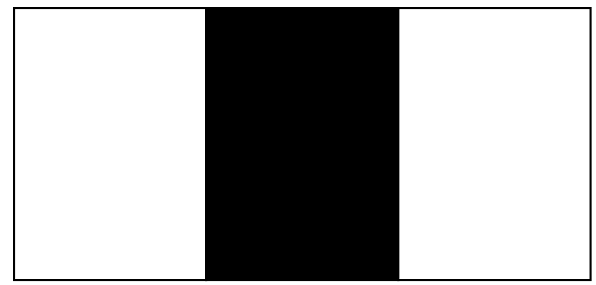
Effect of Rectangle Size

- Larger rectangles → emphasis on larger scale features.
- Smaller rectangles → emphasis on smaller scale features.

Five Different Basic Shapes

- Type 3:
  - Three areas, horizontal.

```
[1 1 -2 -2 1 1
 1 1 -2 -2 1 1
 1 1 -2 -2 1 1
 1 1 -2 -2 1 1]
```



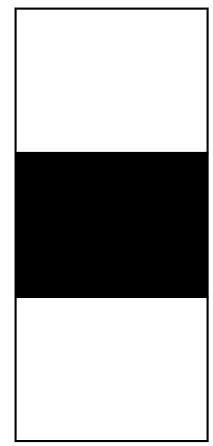
white: value = 1
black: value = -2

- To define:
  - Specify rectangle size:
    - number of rows
    - number of cols.

Five Different Basic Shapes

- Type 4:
  - Three areas, vertical.

```
[1  1  1  1
 1  1  1  1
-2 -2 -2 -2
-2 -2 -2 -2
 1  1  1  1
 1  1  1  1]
```



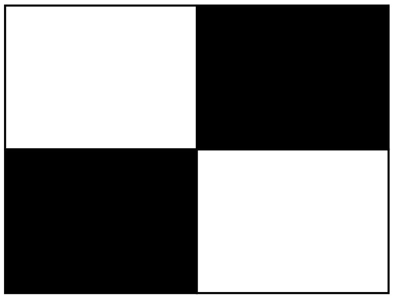
white: value = 1
black: value = -2

- To define:
  - Specify rectangle size:
    - number of rows
    - number of cols.

Five Different Basic Shapes

- Type 5:
  - Four areas, diagonal.

```
[1  1  1 -1 -1 -1
 1  1  1 -1 -1 -1
-1 -1 -1  1  1  1
-1 -1 -1  1  1  1]
```



- To define:
  - Specify rectangle size:
    - number of rows
    - number of cols.

white: value = 1
black: value = -1

Advantages of Rectangle Filters

- Easy to define and implement.
- Lots of them.
  - Intuitively, for many patterns that we want to detect/recognize, we can find some rectangle filters that are useful.
- Fast to compute.
  - How?

Advantages of Rectangle Filters

- Easy to define and implement.
- Lots of them.
  - Intuitively, for many patterns that we want to detect/recognize, we can find some rectangle filters that are useful.
- Fast to compute.
  - How?
  - Using integral images (defined in the next slides).

Integral Image

- Input: grayscale image A.
  - A(i, j) is the intensity value at pixel (i, j).
- Integral image B:
  - B(i, j) = sum(sum(A(1:i, 1:j))).
- F = rectangle_filter1(50, 40);
- How can I compute the response on subwindow A(101:150, 201:280)?

Integral Image

- Input: grayscale image A.
  - A(i, j) is the intensity value at pixel (i, j).
- Integral image B.
  - B(i, j) = sum(sum(A(1:i, 1:j))).
- F = rectangle_filter1(50, 40);
- How can I compute the response on subwindow A(101:150, 201:280)?
  - First approach: sum(sum(A(101:150, 201:280) .* F))
  - How many operations does that take? O(50*80)

Integral Image

- Input: grayscale image A.
- Integral image: B(i, j) = sum(sum(A(1:i, 1:j))).
- F = rectangle_filter1(50, 40);
- How can I compute the response on subwindow A(101:150, 201:280)?
  - sum(sum(A(101:150, 201:240))) –
    sum(sum(A(101:150, 241:280))).
  - where is F in the sum?

Integral Image

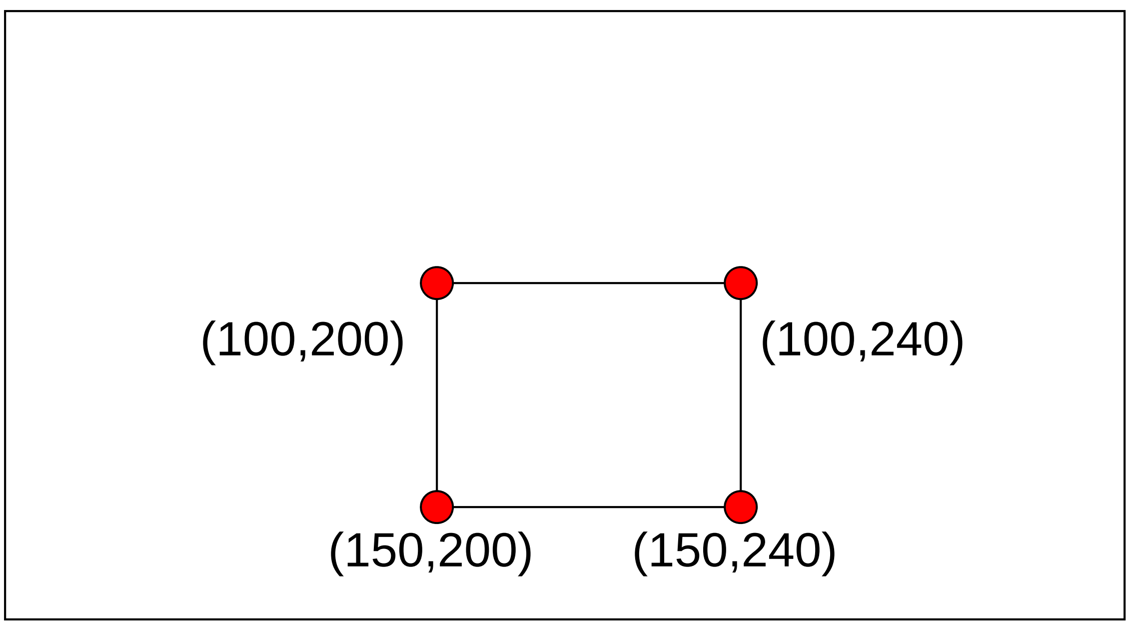
- Input: grayscale image A.
- Integral image: B(i, j) = sum(sum(A(1:i, 1:j))).
- F = rectangle_filter1(50, 40);
- How can I compute the response on subwindow A(101:150, 201:280)?
  - sum(sum(A(101:150, 201:240))) –
    sum(sum(A(101:150, 241:280))).
  - where is F in the sum?
    - F is encoded in specifying the submatrices that we sum.

Integral Image

- Integral image: B(i, j) = sum(sum(A(1:i, 1:j))).
- Compute sum(sum(A(101:150, 201:240))) fast using integral image:

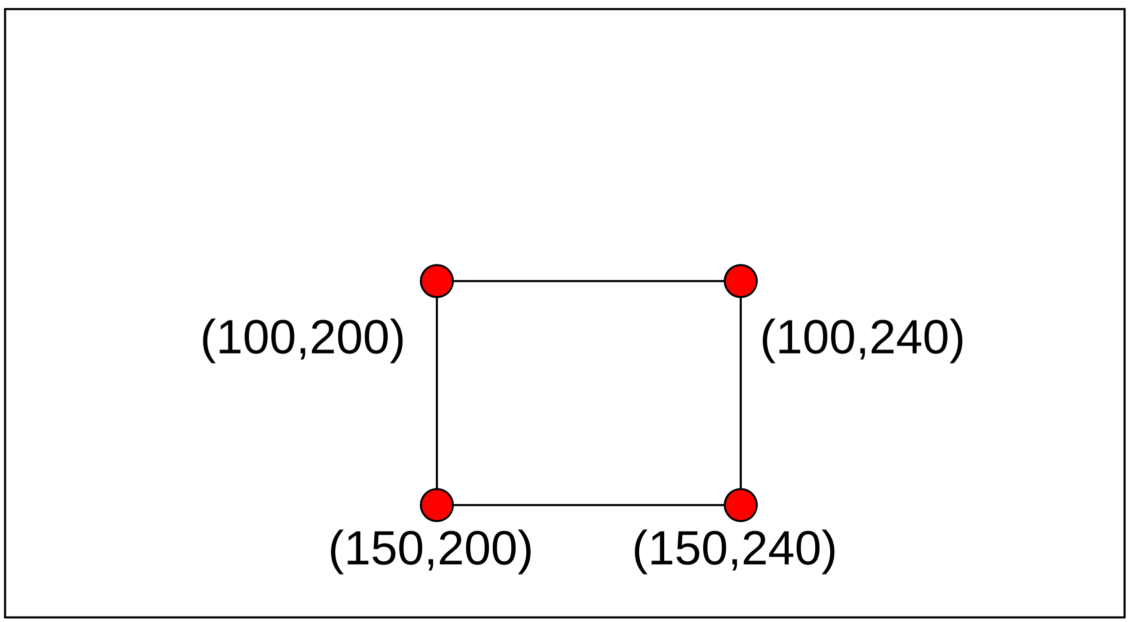
Integral Image

- Integral image: B(i, j) = sum(sum(A(1:i, 1:j))).
- Compute sum(sum(A(101:150, 201:240))) fast using integral image:
  - B(150,240)+B(100,200)–B(150,200)-B(100,200)



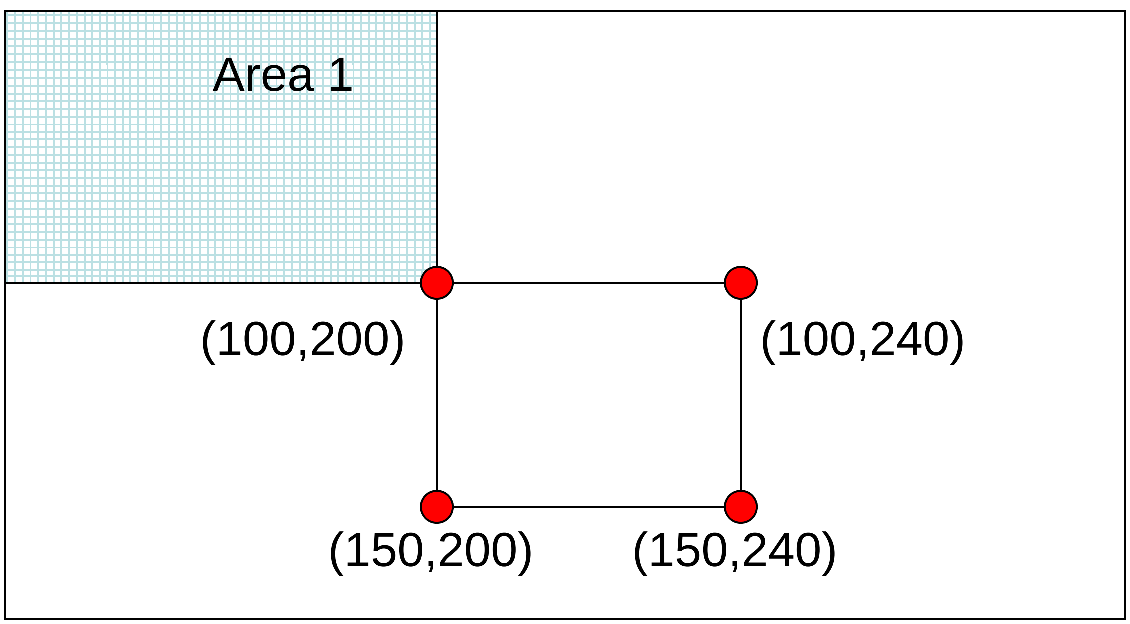
Integral Image

- Compute sum(sum(A(101:150, 201:240))) fast using integral image:
  - B(150,240)+B(100,200)–B(150,200)-B(100,200)



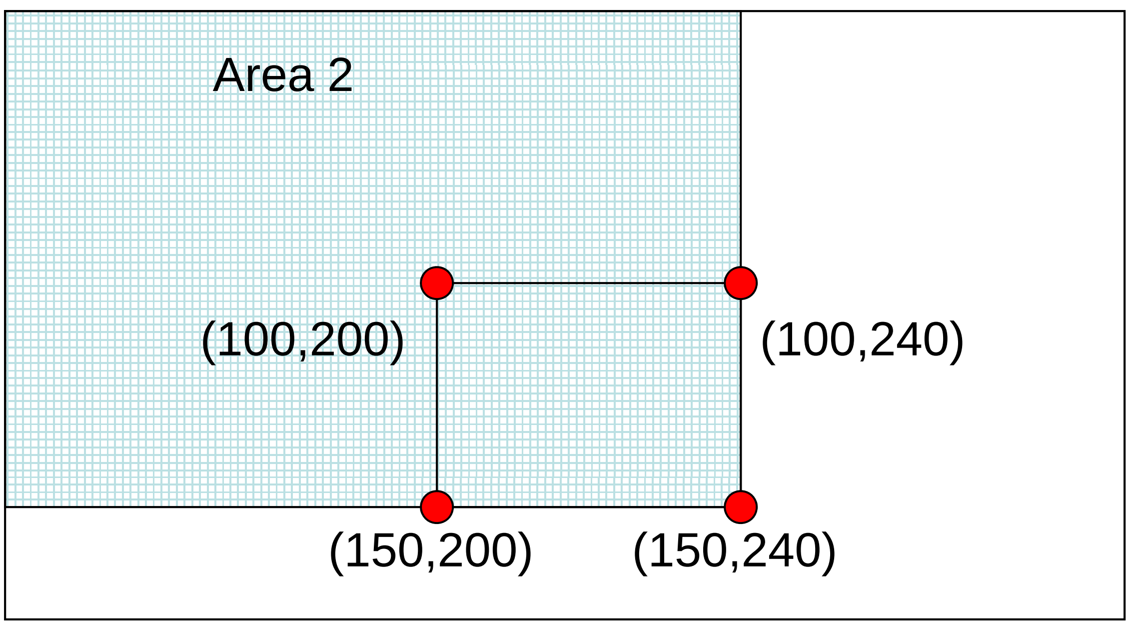
Integral Image

- Compute sum(sum(A(101:150, 201:240))) fast using integral image:
  - B(150,240)+B(100,200)–B(150,200)-B(100,200)
  - sum(Area1)+sum(Area2)–sum(Area3)–sum(Area4)



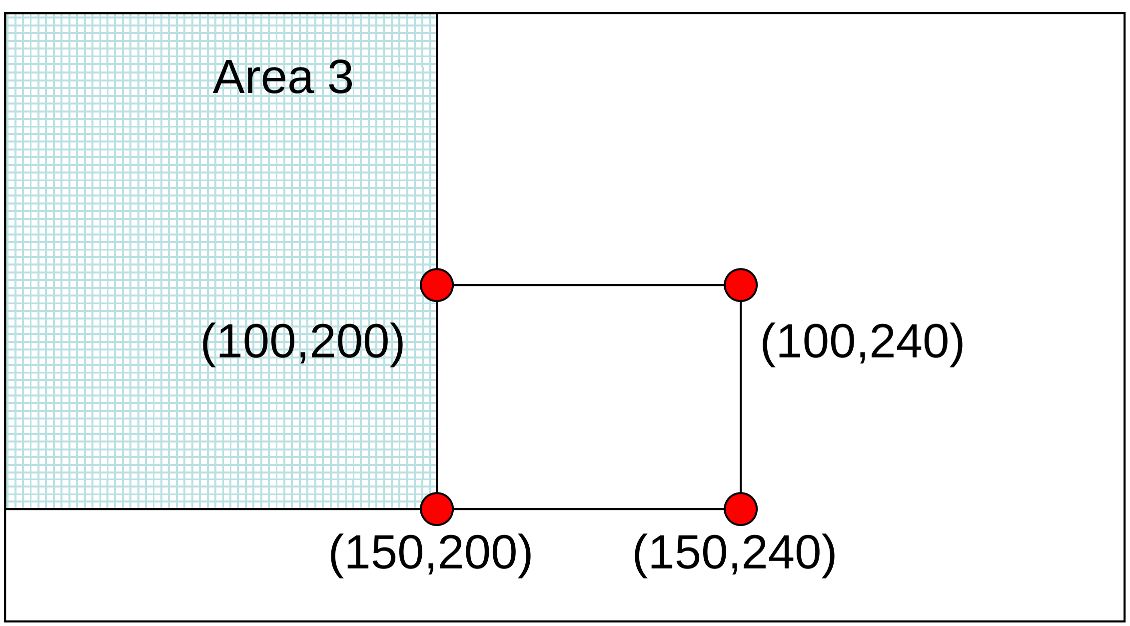
Integral Image

- Compute sum(sum(A(101:150, 201:240))) fast using integral image:
  - B(150,240)+B(100,200)–B(150,200)-B(100,200)
  - sum(Area1)+sum(Area2)–sum(Area3)–sum(Area4)



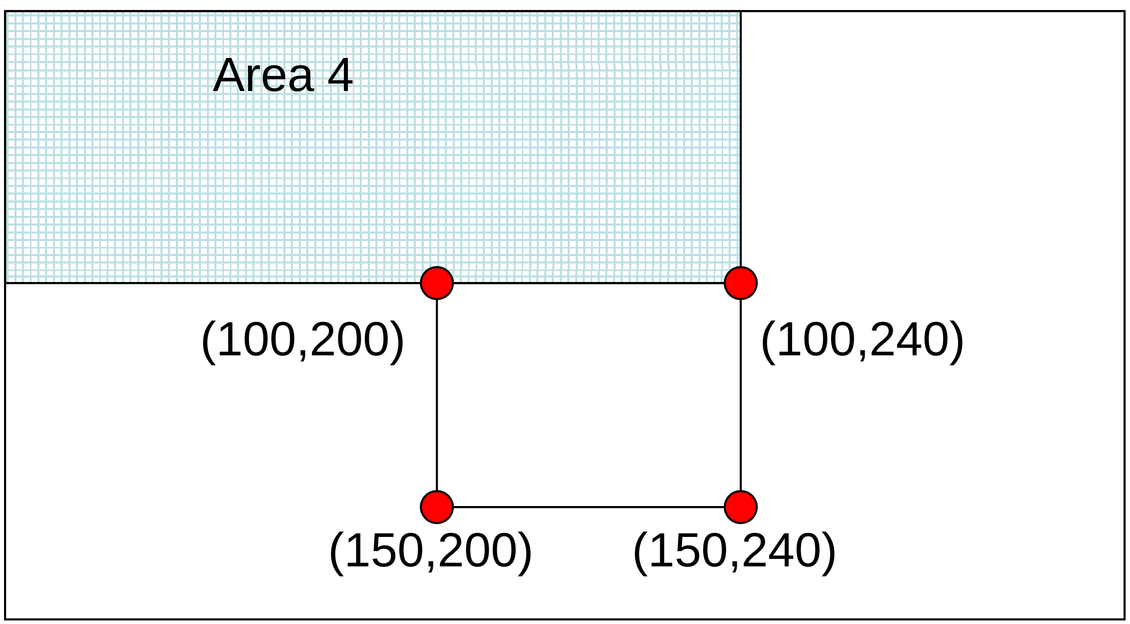
Integral Image

- Compute sum(sum(A(101:150, 201:240))) fast using integral image:
  - B(150,240)+B(100,200)–B(150,200)-B(100,200)
  - sum(Area1)+sum(Area2)–sum(Area3)–sum(Area4)



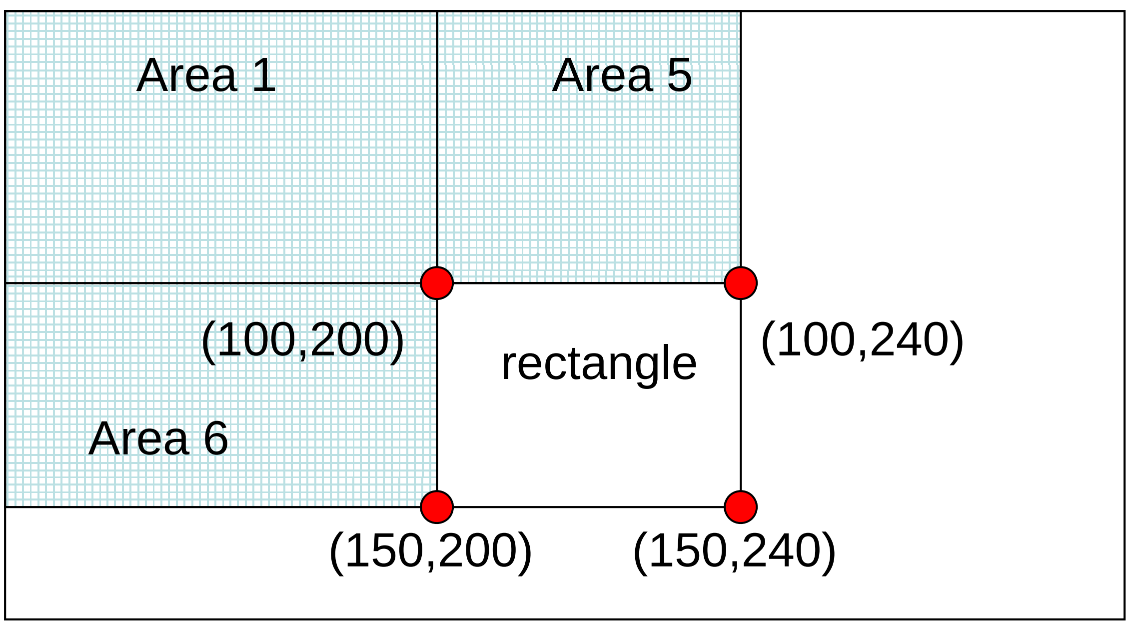
Integral Image

- Compute sum(sum(A(101:150, 201:240))) fast using integral image:
  - B(150,240)+B(100,200)–B(150,200)-B(100,200)
  - sum(Area1)+sum(Area2)–sum(Area3)–sum(Area4)



Integral Image

- Compute sum(sum(A(101:150, 201:240))) fast using integral image:
  - B(150,240)+B(100,200)–B(150,200)-B(100,200)
  - sum(Area1)+sum(Area2)–sum(Area3)–sum(Area4)



- Area1: added twice, subtracted twice.
- Area5: added once, subtracted once.
- Area6: added once, subtracted once.
- Rectangle: added once.

Rectangle Filters for Face Detection

- Key questions in using rectangle filters for face detection:
  - Or for any other classification task.
- How do we use a rectangle filter to extract information?
- What rectangle filter-based information is the most important?
- How do we combine information from different rectangle filters?

From Rectangle Filter to Classifier

- We want to define a classifier, that “guesses” if a specific image window is a face or not.
    - Remember, all face detector operate window-by-window.
- Convention:
    - – Classifier output +1 means “face”.
    - – Classifier output -1 means “not a face”.

What is a Classifier?

- What is the input of a classifier?
- What is the output of a classifier?

What is a Classifier?

- What is the input of a classifier?
  - An entire image, or
  - An image subwindow, or
  - Any type of pattern (audio, biological, …).
- What is the output of a classifier?

What is a Classifier?

- What is the input of a classifier?
  - An entire image, or
  - An image subwindow, or
  - Any type of pattern (audio, biological, …).
- What is the output of a classifier?
  - A *class label*.
  - We have a *finite* number of classes.

What is a Classifier?

- Focusing on face detection:
- A classifier takes as input an image *subwindow*.
- A classifier outputs +1 (for “face”) or -1 (for “non-face”).
- What is a classifier?

What is a Classifier?

- Focusing on face detection:
- A classifier takes as input an image *subwindow*.
- A classifier outputs +1 (for "face") or -1 (for "non-face").
- What is a classifier?
  - Any function that takes the specified input and produces the specified output.
- VERY IMPORTANT: "classifier" is not the same as "accurate classifier."

Classifier Based on a Rectangle Filter

- How can we define a classifier using a single rectangle filter?
- Goal:
  - We want something very fast.
  - It does not have to always be very accurate.
    - We will try LOTS of such classifiers, we are happy as long as SOME of them are very accurate.

Classifier Based on a Rectangle Filter

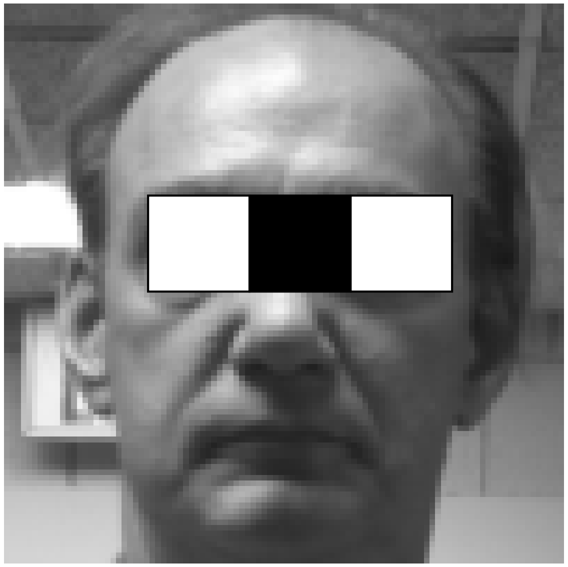
- How can we define a classifier using a single rectangle filter?
- Answer: choose response on a specific location.
- Classifier is specified by:
  - Type
  - Rectangle size
  - *Window offset.*
  - *Threshold.*

Classifier Based on a Rectangle Filter

- How can we define a classifier using a single rectangle filter?
- Answer: choose response on a specific location.
- Classifier is specified by:
  - Type
  - Rectangle size
  - *Window offset.*
  - *Threshold.*

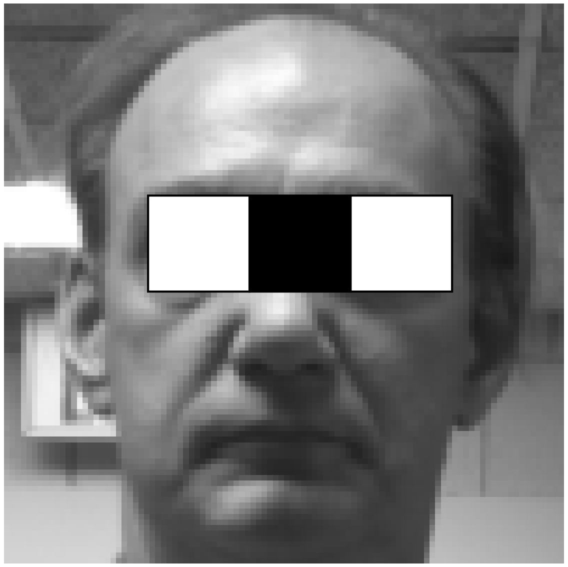
Classifier Based on a Rectangle Filter

- How can we define a classifier using a single rectangle filter?
- Answer: choose response on a specific location.
- Classifier is specified by:
  - Type
  - Rectangle size
  - *Window offset* (row, col).
  - *Threshold.*



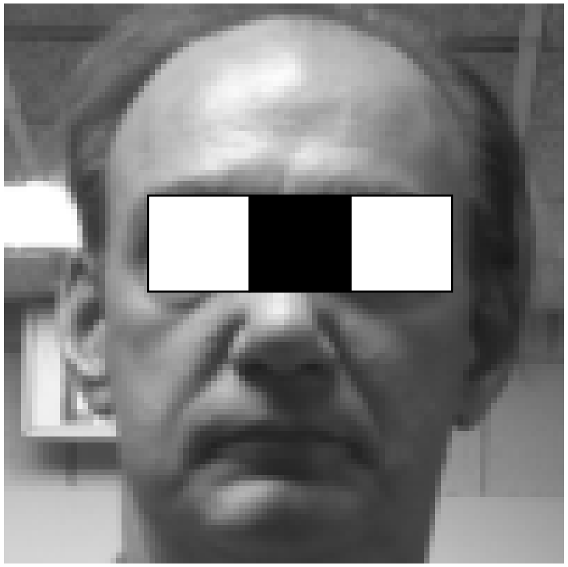
Classifier Based on a Rectangle Filter

- Classifier is specified by:
  - Type
  - Rectangle size
  - *Window offset* (row, col).
  - *Threshold.*
- How many classifiers do we get from a single rectangle filter?



Classifier Based on a Rectangle Filter

- Classifier is specified by:
  - Type
  - Rectangle size
  - *Window offset* (row, col).
  - *Threshold.*
- How many classifiers do we get from a single rectangle filter?
  - upper bound:
    # pixels * # thresholds



Useful Functions

- generate_classifier1
- generate_classifier2
- generate_classifier3
- generate_classifier4
- generate_classifier
- eval_weak_classifier

How Many Classifiers?

- Upper bound:
  - 31x25 offsets.
  - 5 types
  - 31x25 rectangle sizes.
  - Lots of different possible thresholds.
- Too many to evaluate.
- We can sample.
  - Many classifiers are similar to each other.
  - Approach: pick some (a few thousands).
  - Function: generate_classifier

Terminology

- For the purposes of this lecture, we will introduce the term *soft classifier.*
- A soft classifier is defined by specifying:
  - A rectangle filter.
    - (i.e., type of filter and size of rectangle).
  - An offset.
- A weak classifier is defined by specifying:
  - A soft classifier.
    - (i.e., rectangle filter and offset).
  - A threshold.

Precomputing Responses

- Suppose we have a training set of faces and non-faces.
- Suppose we have picked 1000 soft classifiers.
- For each training example, we can precompute the 1000 responses, and save them.
- This way, we know for each soft classifier and each example what the response is.

Precomputing Responses

- Suppose we have a training set of faces and non-faces.
- Suppose we have picked 1000 soft classifiers.
- For each training example, we can precompute the 1000 responses, and save them.
- This way, we know for each soft classifier and each example what the response is.
- How many classifiers does that correspond to?

Precomputing Responses

- Suppose we have a training set of faces and non-faces.
- Suppose we have picked 1000 soft classifiers.
- For each training example, we can precompute the 1000 responses, and save them.
- This way, we know for each soft classifier and each example what the response is.
- How many classifiers does that correspond to? upper bound: 1000 * # thresholds.

How Useful is a Classifier?

- First version: how useful is a soft classifier by itself?
- Measure the error.
  - Involves finding an optimal threshold.
  - Also involves deciding if faces tend to be above or below the threshold.

```
function [error, best_threshold, best_alpha] = ...
      weighted_error(responses, labels, weights, classifier)

classifier_responses = [responses(classifier,:)]';
minimum = min(classifier_responses);
maximum = max(classifier_responses);
step = (maximum - minimum) / 50;
best_error = 2;
for threshold = minimum:step:maximum
     thresholded = double(classifier_responses > threshold);
     thresholded(thresholded == 0) = -1;
     error1 = sum(weights .* (labels ~= thresholded));
     error = min(error1, 1 – error1);
     if (error < best_error)
         best_error = error;
         best_threshold = threshold;
         best_direction = 1;
         if (error1 > (1 – error1))
             best_direction = -1;
         end
     end
end
best_alpha = best_direction * 0.5 * log( (1 - error) / error);
if (best_error == 0)
    best_alpha = 1;
end
```

```
function [index, error, threshold, alpha] = ...
    find_best_classifier(responses, labels, weights)

classifier_number = size(responses, 1);
example_number = size(responses, 2);

% find best classifier
best_error = 2;

for classifier = 1:classifier_number
    [error threshold alpha] = weighted_error(responses, labels, ...
                                             weights, classifier);
    if (error < best_error)
        best_error = error;
        best_threshold = threshold;
        best_classifier = classifier;
        best_alpha = alpha;
    end
end

index = best_classifier;
error = best_error;
threshold = best_threshold;
alpha = best_alpha;
```

How to Combine Many Classifiers

- 3% error is not great for a face detector.
- Why?

How to Combine Many Classifiers

- 3% error is not great for a face detector.
- Why?
  - Because the number of windows is huge:
    - #pixels * #scales, = hundreds of thousands or more.
  - We do not want 1000 or more false positives.
- We need to combine information from many classifiers.
  - How?

Boosting (AdaBoost)

- Weak classifier:
  - Better than a random guess.
  - What does that mean?

Boosting (AdaBoost)

- We have a binary problem.
  - Face or no face.
  - George or not George.
  - Mary or not Mary.
- Weak classifier:
  - Better than a random guess.
  - What does that mean?
    - For a binary problem, less than 0.5 error rate.
- Strong classifier:
  - A “good” classifier (definition intentionally fuzzy).

Boosted Strong Classifier

- Linear combination of weak classifiers.
- Given lots of classifiers:
  - Choose *d* of them.
  - Choose a weight *alpha_d* for each of them.
- Final result:
  - *H(pattern) = alpha_1 * h1(pattern) + …*
    *+ alpha_d * hd(pattern).*
- Goal: strong classifier *H* should be significantly more accurate than each weak classifier *h1, h2, ...*

Detour: Optimization

- Final result:
  - *H(pattern) = alpha_1 * h1(pattern) + …*

    *+ alpha_d * h1(pattern).*
- How can we cast this as an optimization problem?

Detour: Optimization

- Final result:
  - *H(pattern) = alpha_1 * h1(pattern) + …*
    *+ alpha_d * h1(pattern).*
- How can we cast this as an optimization problem?
  - Identify free parameters:

Detour: Optimization

- Final result:
  - *H(pattern) = alpha_1 * h1(pattern) + …*

    *+ alpha_d * h1(pattern).*
- How can we cast this as an optimization problem?
  - Identify free parameters:
    - alphas (including zero weights for classifiers that are not selected).
    - Thresholds for the soft classifiers that were selected.

Detour: Optimization

- Final result:
  - *H(pattern) = alpha_1 * h1(pattern) + …*

    *+ alpha_d * h1(pattern).*
- How can we cast this as an optimization problem?
  - Identify free parameters.
  - Define optimization criterion:

Detour: Optimization

- Final result:
  - *H(pattern) = alpha_1 * h1(pattern) + …*

    *+ alpha_d * h1(pattern).*
- How can we cast this as an optimization problem?
  - Identify free parameters.
  - Define optimization criterion:
    - F(parameters) = error rate attained on training set using those parameters.
  - We want to find optimal parameters for minimizing F.

Detour: Optimization

- Optimization:
  - Can be global or local.
  - Lots of times when people say "optimal" they really mean "locally optimal".
- Given F(parameters) = error rate attained on training set using those parameters:
  - How can we do optimization?

Detour: Optimization

- Optimization:
  - Can be global or local.
  - Lots of times when people say “optimal” they really mean “locally optimal”.
- Given F(parameters) = error rate attained on training set using those parameters:
  - How can we do optimization?
    - Try many different combinations.
    - Gradient descent.
    - Greedy: choose weak classifiers one by one.
      - First, choose best weak classifier.
      - Then, at each iteration, choose next weak classifier and weight, to be the one giving the best results when combined with the previously picked classifiers and weights.

Boosted Strong Classifier

- Linear combination of weak classifiers.
- Given lots of classifiers:
  - Choose *d* of them.
  - Choose a weight *alpha_d* for each of them.
- Final result:
  - *H(pattern) = alpha_1 * h1(pattern) + …*
    *+ alpha_d * hd(pattern).*
- Goal: strong classifier *H* should be significantly more accurate than each weak classifier *h1, h2, ...*

Choosing the First Weak Classifier

- First task: choose *alpha_1, h1*.
  - Use function find_best_classifier.

Choosing the First Weak Classifier

- First task: choose *alpha_1, h1*.
  - Use function find_best_classifier.
  - Choose an *alpha_1* for that classifier.
    - Lower error → higher weight.

```
alpha = direction * 0.5 * log((1 - error) / error);
```

- Interpreting alpha formula:
  - When error is 0:

Choosing the First Weak Classifier

- First task: choose *alpha_1, h1*.
  - Use function find_best_classifier.
  - Choose an *alpha_1* for that classifier.
    - Lower error → higher weight.

```
alpha = direction * 0.5 * log((1 - error) / error);
```

- Interpreting alpha formula:
  - When error is 0:
    - *log(1/0) = log(inf) = inf*.
    - depending on direction, *alpha = +inf* or *-inf*

Choosing the First Weak Classifier

- First task: choose *alpha_1, h1*.
  - Use function find_best_classifier.
  - Choose an *alpha_1* for that classifier.
    - Lower error → higher weight.

```
alpha = direction * 0.5 * log((1 - error) / error);
```

- Interpreting alpha formula:
  - When error is 0.5:

Choosing the First Weak Classifier

- First task: choose *alpha_1, h1*.
  - Use function find_best_classifier.
  - Choose an *alpha_1* for that classifier.
    - Lower error → higher weight.

```
alpha = direction * 0.5 * log((1 - error) / error);
```

- Interpreting alpha formula:
  - When error is 0.5:
    - *log(0.5/0.5) = log(1) = ?*

Choosing the First Weak Classifier

- First task: choose *alpha_1, h1*.
  - Use function find_best_classifier.
  - Choose an *alpha_1* for that classifier.
    - Lower error → higher weight.

```
alpha = direction * 0.5 * log((1 - error) / error);
```

- Interpreting alpha formula:
  - When error is 0.5:
    - *log(0.5/0.5) = log(1) = 0.*
    - Capturing the fact that the weak classifier is useless.

Choosing the First Weak Classifier

- First task: choose *alpha_1, h1*.
  - Use function find_best_classifier.
  - Choose an *alpha_1* for that classifier.
    - Lower error → higher weight.

```
alpha = direction * 0.5 * log((1 - error) / error);
```

- Interpreting alpha formula:
  - When error is 0.9:

Choosing the First Weak Classifier

- First task: choose *alpha_1, h1*.
  - Use function find_best_classifier.
  - Choose an *alpha_1* for that classifier.
    - Lower error → higher weight.

```
alpha = direction * 0.5 * log((1 - error) / error);
```

- Interpreting alpha formula:
  - When error is 0.9:
    - *log(0.1/0.9) = log(.111) = ?*

Choosing the First Weak Classifier

- First task: choose *alpha_1, h1*.
  - Use function find_best_classifier.
  - Choose an *alpha_1* for that classifier.
    - Lower error → higher weight.

```
alpha = direction * 0.5 * log((1 - error) / error);
```

- Interpreting alpha formula:
  - When error is 0.9:
    - *log(0.1/0.9) = log(.111) = -2.197*
    - Capturing the fact that we should do the opposite of what the classifier says.

Updating Training Weights

- So far we have chosen *alpha_1 and h1*.
- Next step: update training weights.
- Initially, training weights are uniform:
  - each weight is 1/(number of training examples).

Updating Training Weights

- So far we have chosen *alpha_1 and h1*.
- Next step: update training weights.
- Initially, training weights are uniform:
  - each weight is 1/(number of training examples).
- VERY IMPORTANT:
  - In boosting there are two different types of weights.
    - An *alpha* for each weak classifier that is chosen.
    - A weight for each training weight at each round.

Updating Training Weights

- So far we have chosen *alpha_1 and h1*.
- Next step: update training weights.
- Initially, training weights are uniform:
  – each weight is 1/(number of training examples).
- After choosing *h1*, *alpha_1*, each training weight is updated:

```
weights(i) = weights(i) * exp(-alpha * h1(i) * labels(i));
```

- When sign(alpha) * h(i) == labels(i):
  – we multiply weight by a number < 1.

Updating Training Weights

- So far we have chosen *alpha_1 and h1*.
- Next step: update training weights.
- Initially, training weights are uniform:
  - each weight is 1/(number of training examples).
- After choosing *h1*, *alpha_1*, each training weight is updated:

```
weights(i) = weights(i) * exp(-alpha * h1(i) * labels(i));
```

- When sign(alpha) * h(i) != labels(i):
  - we multiply weight by a number > 1.

Updating Training Weights

- So far we have chosen *alpha_1 and h1*.
- Next step: update training weights.
- Initially, training weights are uniform:
  – each weight is 1/(number of training examples).
- After choosing *h1*, *alpha_1*, each training weight is updated:

```
weights(i) = weights(i) * exp(-alpha * h1(i) * labels(i));
```

- Effect: examples that got misclassified by *alpha_1 * h1* become more important.

Updating Training Weights

- So far we have chosen *alpha_1 and h1*.
- Next step: update training weights.
- Initially, training weights are uniform:
  – each weight is 1/(number of training examples).
- After choosing *h1*, *alpha_1*, each training weight is updated:

```
weights(i) = weights(i) * exp(-alpha * h1(i) * labels(i));
```

- Important: weights are always normalized to sum up to 1.
  – *weights = weights / sum(weights).*

Choosing the Second Weak Classifier

- Use function find_best_classifier.

```
find_best_classifier(responses, labels, weights)
```

  - Choose an *alpha_2* for that classifier.
    - Lower error → higher weight.

```
alpha = direction * 0.5 * log((1 - error) / error);
```

- This is *exactly* the same thing that we did for choosing the first weak classifier.
- Why would we get a different weak classifier?

Choosing the Second Weak Classifier

- Use function find_best_classifier.

```
find_best_classifier(responses, labels, weights)
```

  - Choose an *alpha_2* for that classifier.
    - Lower error → higher weight.

```
alpha = direction * 0.5 * log((1 - error) / error);
```

- This is *exactly* the same thing that we did for choosing the first weak classifier.
- Why would we get a different weak classifier?
  - ***Because the training weights have changed.***

A Training Round of AdaBoost

- Try all classifiers.
- See which one has lower error rate.
  - IMPORTANT: error rates are measured on *weighted* training set.
- Choose an alpha for that classifier.
  - Lower error → higher alpha.
- Update weights of training examples.

AdaBoost

- A simple for-loop, performing training rounds one at a time.

```matlab
function result = AdaBoost(responses, labels, rounds)

result = zeros(rounds, 3);
classifier_number = size(responses, 1);
example_number = size(responses, 2);
weights = ones(example_number, 1) / example_number;
boosted_responses = zeros(example_number, 1);

for round = 1:rounds
    % find index, threshold, and alpha of best classifier
    [best_classifier, best_error, threshold, alpha] = ...
        find_best_classifier(responses, labels, weights);
    result(round, 1:3) = [best_classifier, alpha, threshold];

    % get outputs of the weak classifier on training examples
    weak_responses = double([responses(best_classifier, :)]' > threshold);
    weak_responses(weak_responses == 0) = -1;

    % reweigh training objects;
    for i = 1:example_number
        weights(i) = weights(i) * exp(-alpha * weak_responses(i) * labels(i));
    end
    weights = weights / sum(weights);

    % update boosted responses
    boosted_responses = boosted_responses + alpha * weak_responses;
    thresholded = double(boosted_responses > 0);
    thresholded(thresholded == 0) = -1;
    error = mean(thresholded ~= labels);
    disp([round error best_error best_classifier alpha threshold]);
end
```

Key Intuiton

- Objects that are misclassified by the first chosen weak classifiers become more important.
- The next weak classifiers try to correct the mistakes of the first weak classifiers.

A Note on AdaBoost formulas.

- The actual formulas for AdaBoost may seem ad hoc:
  - Formula for choosing classifier alpha.

```
alpha = direction * 0.5 * log((1 - error) / error);
```

  - Formula for updating training weights.

```
weights(i) = weights(i) * exp(-alpha * h1(i) * labels(i));
```

- These formulas can be justified mathematically.
  - That is beyond the scope of our class.

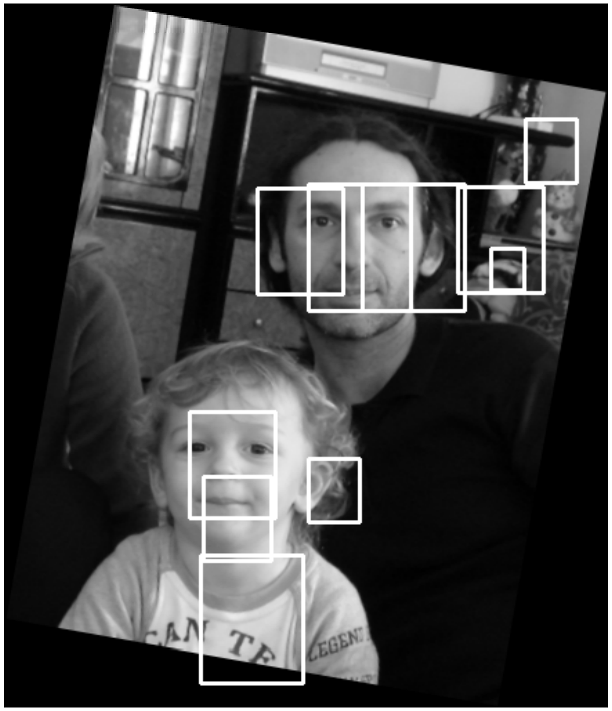
Applying Boosted Classifier

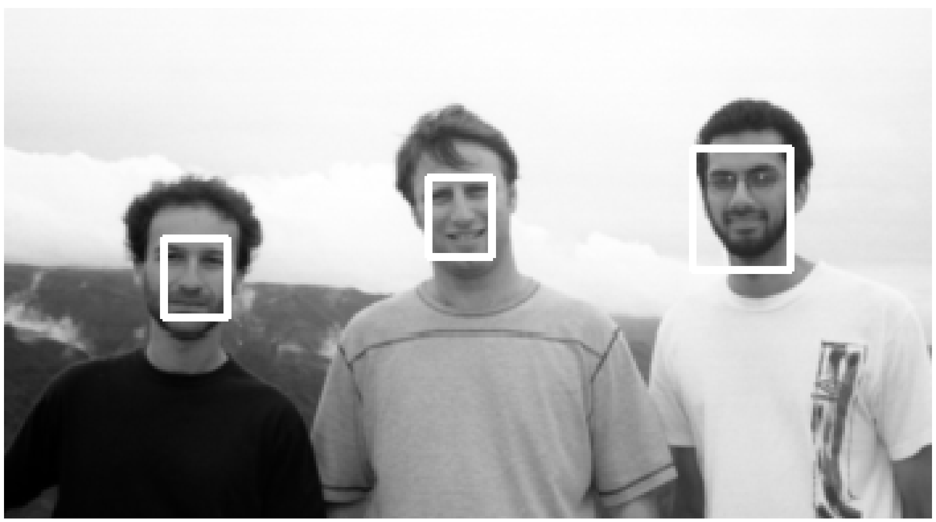
- Given an image:
- For every scale:
  - Compute integral image at that scale.
  - For every window:
    - For every weak classifier:
      - Compute response of window on that weak classifier.
      - Sum up the responses.
- For every pixel, get the best result among all windows (at all scales) centered at that pixel.
- See function boosted_detector_demo

Why Is it Useful

- We don’t need to break our heads over:
  - What features to use.
  - How to combine them.

Results





How to Improve Accuracy

- More classifiers.
- More training data.
- Multiple training sessions.
  - After each session, find non-face windows that are detected as faces, and include to the training set.

How to Improve Efficiency

- Use a *cascade of classifiers*.
- First, try a strong classifier containing only few weak classifiers.
  - A lot of non-face windows can be rejected with a suitable threshold.
- What remains, is evaluated by a more complicated strong classifier.

- CLASS PROJECT