CV Topic 15 Object Detection

Dr. V Masilamani
masila@iiitdm.ac.in

Department of Computer Science and Engineering
IIITDM Kancheepuram
Chennai-127

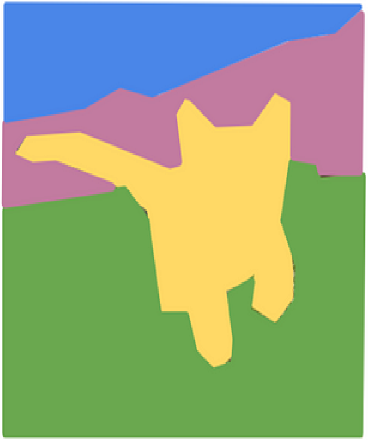
Outline

1. Segmentation Vs Object Localization Vs Object Classification Vs Object Detection
2. Object Classification
3. AdaBoost Classifier
4. Viola-Jones Object Detection Algorithm

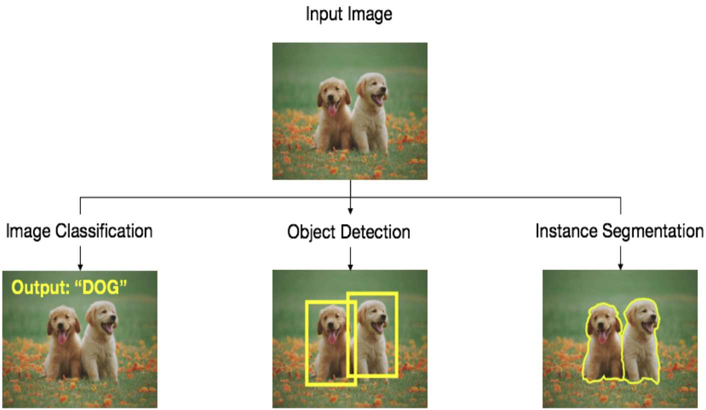
Segmentation Vs Object Localization Vs Object Classification Vs Object Detection

- Each **segment** of the segmented image may corresponds to either an **object** or **background**
- **Object Localization:** Finding rectangular sub image where the object is present
- **Object Classification:** Given the object, find the category of the object
  - To find the category, a predefined set of possible categories will be given
- **Object Detection: Object localization + Object Classification**
  - **Face detection:**
    - Finding rectangle around a face is localization.
    - Classifying the rectangular area into face category or non-face category is a classification
- Given a face image, **identifying name** of the face is also classification problem

Segmentation Vs Localization



Segmentation Vs Object Localization Vs Object Classification Vs Object Detection (cont.)



- **Simple Object Detection method**
  1. Do the segmentation on image
  2. classify each segment into the object category or non-object category
- **Segmentation to Classification**
  - Represent the segment using vector, called feature/pattern vector
  - Build a classifier that learns the relationship between the feature vector and the label of the corresponding segment(object) (**known as training phase**)
  - The classifier built in the training phase need to be tested with another set of example objects(**Known as testing phase**)
  - the tested model can be deployed for the application purpose(**Known as deployment phase**)

Object Classification

- Build **single classifier** that learns relationship between objects and their labels form the training data set
  - Bayes classifier
  - Support Vector Machine
  - Decision Tree
  - Classifier can also be built using regressor(By Thresholding)
    - Linear Regression
    - Logistic Regression
- Build **multiple classifiers**, called weak classifiers, and then build a strong classifier using those week classifiers. This process is called as **ensemble modeling**
- **Ensemble Model:** Machine Learning model that involves k number of models, and classify the object based on the outputs of the those k models for that object

Object Classification (cont.)

- Typed of ensemble model
  - Bagging: Train all the models in parallel. Aggregate of outputs of all these models will be considered for the output of ensemble model
    - Random Forest
  - Boosting:
    - AdaBoost
    - GradientBoost
    - XGBoost
- **Boosting:** Ensembling model building technique that involves k weak models, and builds a strong model
- **Adaptive Boosting (Adaboost):**
  - One way for a new predictor to correct its predecessor is to pay a bit more attention to the training instances that the predecessor under-fitted.

Object Classification (cont.)

- This results in new predictors focusing more and more on the hard cases.
- This is known as Adaptive Boosting

- One way of building Adaboost is as follows
  - Let $\{x_1, x_2, ..., x_N\}$ be the training data set
  - Assign weight $w_i = \frac{1}{N}$ for each data x_i for $1 \leq i \leq N$
  - All the k week models are arranged in series
  - Train the first model in the series on train data set
  - Train the next model such a way that that data that are misclassified by the previous model are correctly classified with high probability
    - This is done by updating the weights of the misclassified data with high values
  - Repeat the previous step until required amount of training accuracy is achieved

Object Classification (cont.)

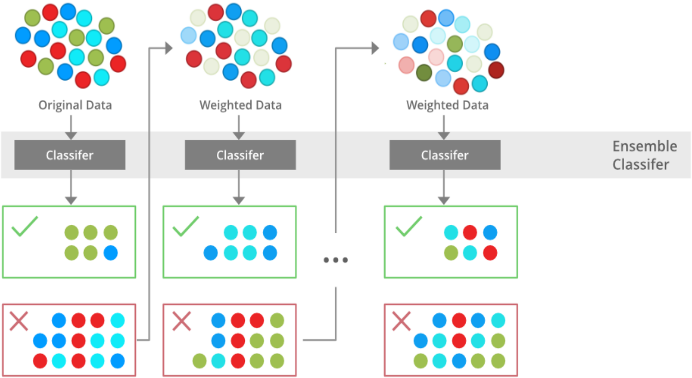


Figure 1: Training of boosting model

AdaBoost Classifier

- Given examples $(x_1, y_1), (x_2, y_2), ...(x_n, y_n)$, where $y_i = 1$ for positive examples, $y_i = 0$ for negative examples
- Initialize Weights
  - $w_{1,i} = \frac{1}{2m}$ when x_i is positive example
  - $w_{1,i} = \frac{1}{2l}$ when x_i is negative example
- For $t = 1, ... T$
  1. Normalize the weights: $w_{t,i} = \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$ (w_t is a pdf)
  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 $w_{t,i}$ as
     - $\epsilon_j = \sum_i w_{t,i}|h_j(x_i) - y_i|$
  3. Choose the classifier, h_t, which has the lowest error ϵ_t
  4. Update the weights

- $w_{t+1,i} = w_{t,i}\beta_t^{(1-e_i)}$, where
  $e_i = 1$ if x_i is misclassified
  $e_i = 0$ if x_i is classified correctly
  $\beta_t = \frac{\epsilon_t}{1-\epsilon^t}$

- The final strong classifier is
  - $h(x) = 1$ if $\sum_{t=1}^{T} \alpha_t h_t(x) \geq (1/2) \sum_{t=1}^{T} \alpha_t$
  - $h(x) = 0$ otherwise, where
    - $\alpha_t = log(\frac{1}{\beta_t})$

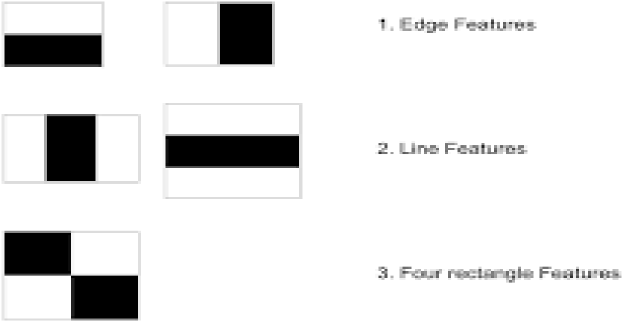
viola-Jones Object Detection Algorithm

- Paul Viola and Michael Jones, Rapid Object Detection using a Boosted Cascade of Simple Features, CVPR, 2001
- Very fast, but not very accurate
- Useful for real time object detection where inaccuracy can be tolerated (**15 fps**)
- It can be run in edge devices
- Some terminologies
  - Four Haar-Like filters considered
  - Feature: The sum of products of filter coefficients with the corresponding pixels values in the image, for one alignment of the filter in the image
- Viola-Jones object detection is also called as Haar-Cascade Classifier

Steps Viola-Jones Algorithm

1. Use Adaboost algorithm to train strong classifiers by training weak classifiers
   - To find the Haar-like features, use integral image for faster computation
2. Build cascade of strong classifiers with different set of features for each strong classifier
3. Integration of Multiple Detection
4. Build Multiple Cascade Detectors, and use Voting Scheme to improve the detection results

Haar-like features considered



Haar-like features -Why

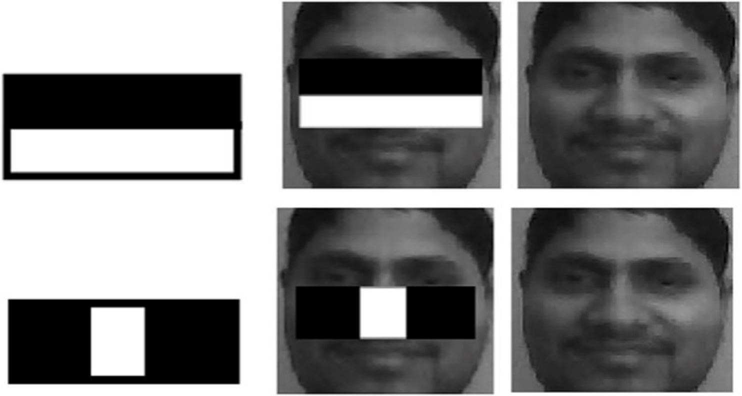
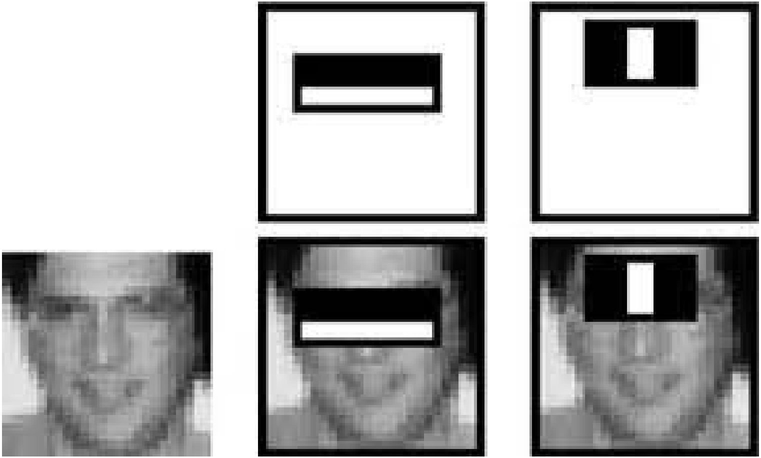
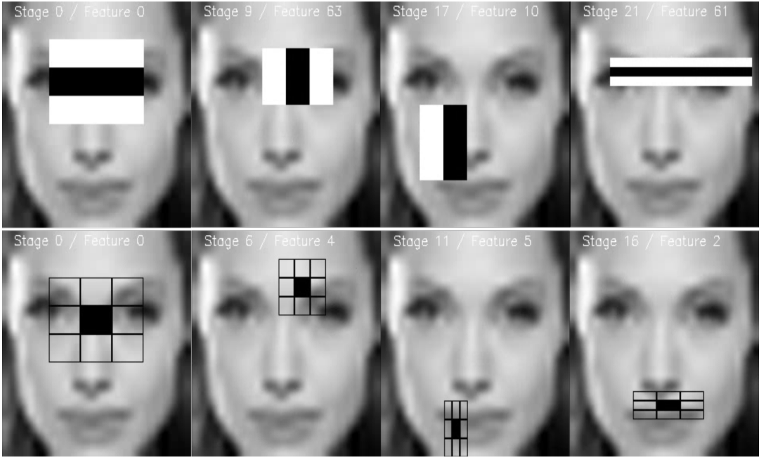


Fig. 3. - Feature detection.[6]

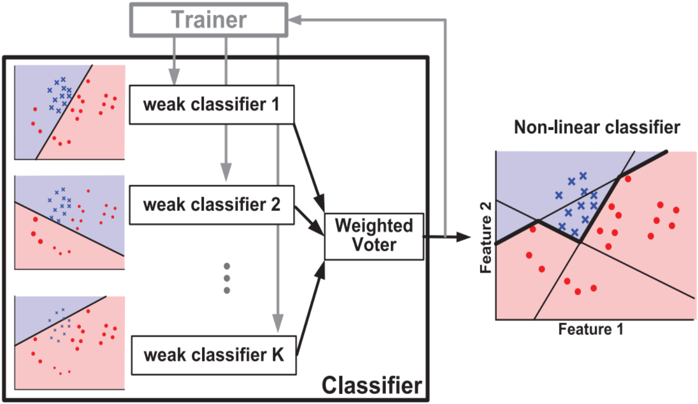
Haar-like features -Why (cont.)



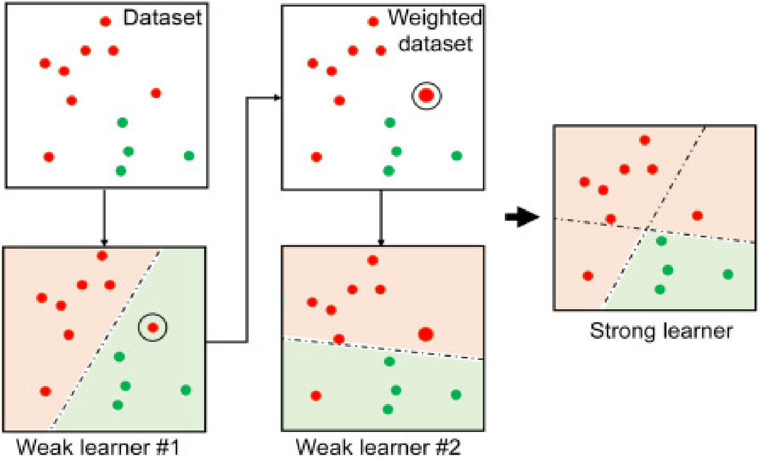
Haar-like features -Why (cont.)



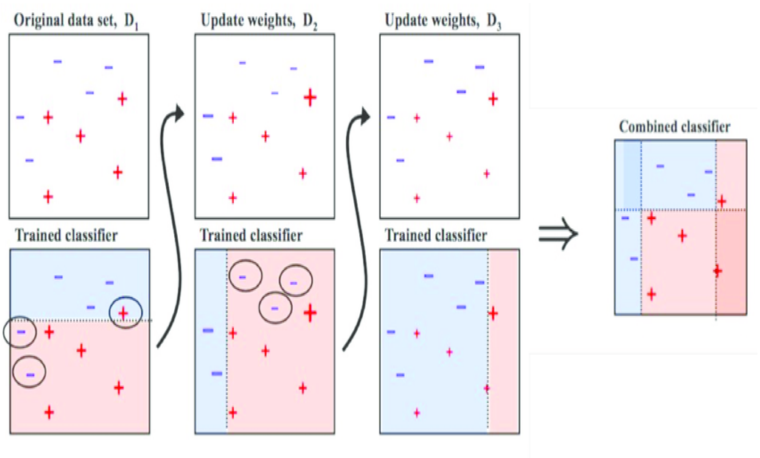
Training of strong classifier through weak classifiers in conventional boosting



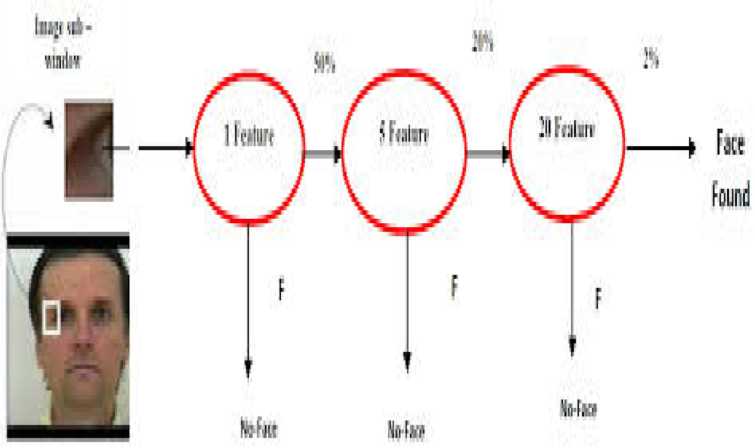
Training of strong classifier through weak classifiers in conventional boosting (cont.)



Training of strong classifier through weak classifiers in conventional boosting (cont.)



Building Cascade of Strong classifiers



Building Cascade of Strong classifiers (cont.)

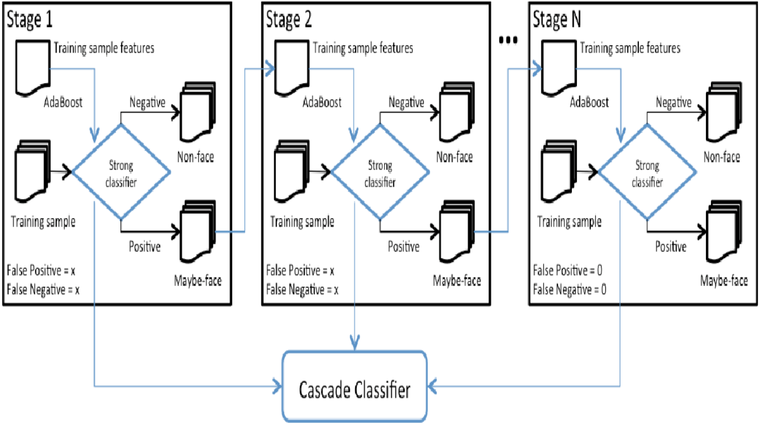


Figure 4. Cascade work flow

How to Detect Bounding box of faces

- Use the cascade of strong classifiers to detect face
  - For each 24×24 subimage of given image, check
    - if the cascade of strong classifier gives positive in all stages, then report window boundary as the bounding box, and slide the window
    - If it gives negative response in any stage, slide the window in the image

Integration of Multiple bounding boxes of the same Face

- Partition the set of all bounding boxes obtained by the cascaded classifier such that in each set in the partition, overlapping windows are present
- To find two bounding boxes B_1 and B_2 have overlapping significantly
  - Find $\text{IoU}(B_1, B_2) = \frac{(B_1 \bigcap B_2)}{(B_1 \bigcup B_2)}$
  - If $\text{IoU}(B_1, B_2) >= 0.5$ then keep B_1 and drop B_2
- Find the bounding box by taking average of corresponding corners of all the windows with positive response

Data set

- The face training set consisted of 4916 hand labeled faces, and 9544 non face images, scaled and aligned to a base resolution of 24 by 24 pixels.
- Tested on: MIT+CMU frontal face test set, which consists with 507 labeled frontal faces

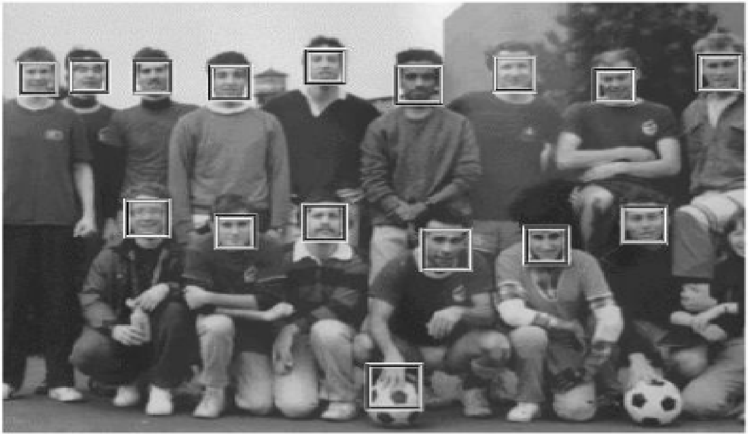
Data set (cont.)

Results

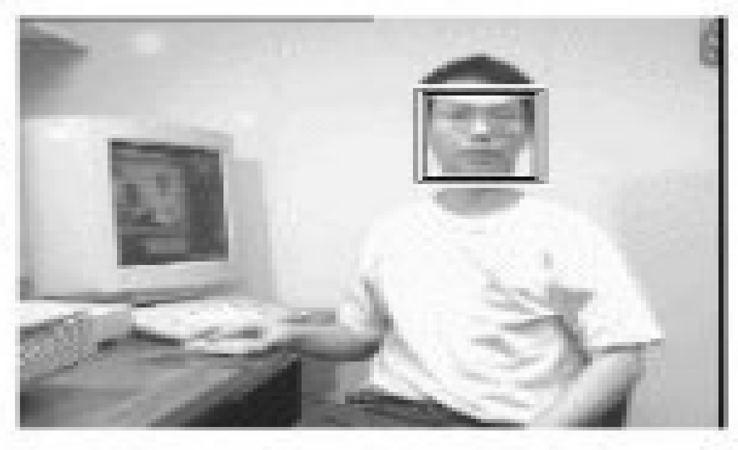
Detector \ False detections	10	31	50	65	78	95	167
Viola-Jones	76.1%	88.4%	91.4%	92.0%	92.1%	92.9%	93.9%
Viola-Jones (voting)	81.1%	89.7%	92.1%	93.1%	93.1%	93.2 %	93.7%
Rowley-Baluja-Kanade	83.2%	86.0%	-	-	-	89.2%	90.1%
Schneiderman-Kanade	-	-	-	94.4%	-	-	-
Roth-Yang-Ahuja	-	-	-	-	(94.8%)	-	-

Results (cont.)

Results (cont.)



Results (cont.)



Results (cont.)