LAB MANUAL - 2D1427 Image Based Recognition and Classification

Babak Rasolzadeh

January 16, 2007
Face Detection

*Real-time face detection in multi-scale images with an attentional cascade of boosted classifiers.*

In this project you will explore the machine learning method called Adaboost by implementing it for the computer vision task of real-time face detection in images. Real-time performance is achieved by exploiting a so-called *attentional cascade*. The final classifier/detector should be capable of detecting upright frontal faces observed in reasonable lighting conditions.

Face detection is an important problem in image processing. It could potentially be the first step in many applications – marking areas of interest for better quality encoding for television broadcasting, content-based representation (in MPEG-4), face recognition, face tracking and gender recognition. In fact for this latter task computer-based algorithms out-perform humans.

During the past decade, many methods and techniques have been gradually developed and applied to solve the problem. These include vector quantization with multiple codebooks, face templates and Principal Component Analysis (PCA). The latter technique is directly related to Eigenfaces and Fisherfaces. Here we will develop a face detection system based on the well-known work of Paul Viola and Michael Jones[2]. This basically involves the interpretation of Haar-like features in a boosted cascade, see paper on the course homepage.

**The competition**

In the end of the course/lab there will be a small competition between those groups that have managed to perform the lab successfully. The objective of the competition is to have a face detector that has accuracy as well as speed. The group that maximizes these two criteria will win the competition and rewarded with a small surprise-price.

**Theory and Background**

There is a loosely defined difference between the process of face detection and that of face recognition. The former assumes that each *instance* of a face belongs to a more general class of objects called *faces*. Thus, the process of face recognition is a more specific process where you try to identify which specific instance of a face you are looking at. Of course this is a sliding scale since one could go on and say that a specific face, e.g. the face of Bill Gates, can have different instances as well (i.e. happy, sad, tired etc.). That would be an even more specific recognition process. In other words, the difference between detection and recognition depends strongly on what class levels one has defined.
However, for the sake of the argument that there is indeed a difference between the two, one could say that at the detection level one needs to identify a set of generalized features that apply for the class we are trying to identify. The localization of such features can be accomplished by a number of common methods. There are basically four different approaches to the problem of face detection:

1. **Knowledge-based methods**: Rules are encoded based on the human knowledge of the defining features of a human face. A majority of these rules capture the relationship between features. [10, 8]

2. **Feature invariant methods**: Algorithms designed to find structural features of a face that are invariant to the common problems of pose, occlusion, expression, image conditions and rotation. [5, 6, 11]

3. **Template matching methods**: Given a sample set a corresponding standard facial pattern set is produced. The relation between the sample image and the defined pattern set is computed and used to provide inference. [9, 7]

4. **Appearance-based methods**: Similar to template matching methods. The goal here is to achieve higher accuracy through larger variation in training data, since one uses statistics without any prior model assumptions. [3, 12]

In this project, the focus will be on a specific appearance-based method, namely the Viola-Jones face detector. This technique relies on the use of simple Haar-like features that are evaluated quickly through the use of a new image representation.

Over-complete Haar-like features with boosted cascade has proven to be an effective approach to visual object detection, capable of processing images extremely rapidly and achieving high detection rates with very low false alarms. The effectiveness of this method comes from four key contributions. The first one is a set of simple masks which are similar to Haar filters. The second part is an image representation called Integral image which allows these features to be computed very quickly. The third contribution is a learning algorithm, based on Adaboost which selects a small number of features from a large set and yields extremely effective classifier. The last one is a method for combining increasingly more complex classifier in a Cascade structure which allow background region of an image to be quickly discarded while spending more computation on promising object-like regions. This is sometimes referred to as an attentional cascade since it spends more computational effort on the more plausible target regions. We will introduce each of these contributions and discuss them in detail below, before going on and implementing them in Matlab.
Feature extraction: Haar-like features

There are two motivations for using features instead of the pixel intensities directly. Firstly, features encode domain knowledge better than pixels. The other reason is that a feature-based system can be much faster than a pixel-based system.

![Feature Examples](image)

Figure 1: Four examples of the type of features normally used in the Viola-Jones system.

In its simplest form, the Viola-Jones features can be thought of as pixel intensity set evaluations. This is where the sum of the luminance of the pixels in the white region(s) of the feature is subtracted from the sum of the luminance in the dark region(s). This difference value is used as the feature value. The position and size of the feature can vary over the detection-box that is used. So for example feature of type 3 in figure 1 will have 4 parameters: the position \((x, y)\) and the size of the white \((w)\) and black \((b)\) regions respectively plus the height \((h)\) of the feature. See figure 3.

**Exercise:** Calculate the number of distinct features as a function of \((n, m)\), for each of the four types in figure 1.

Fast feature extraction: Integral Image

In order to be successful a detection algorithm must possess two key features; accuracy and speed. There is generally a trade-off between the two. Through the use of a new image representation, the integral image, Viola & Jones describe a means for the fast feature evaluation that proves to be an effective way to speed up the classification task.

The integral image \(Int\) of an image \(I\) is defined as:

\[
Int(x, y) = \sum_{x'=0}^{x} \sum_{y'=0}^{y} I(x', y')
\]  

(1)

In other words the integral image at location \((x, y)\) is the sum of all pixel-values above and left of \((x, y)\), inclusive.

The brilliance in using an integral image to speed up a feature extraction
lies in the fact that any rectangle in a image can be calculated from the corresponding integral image, by indexing the integral image only four times. Given a rectangle specified as four coordinates \((x_1, y_1)\) upper left and \((x_4, y_4)\) lower right (see figure 2), evaluating the area of a rectangle is done in four integral image references:

\[
A(x_1, y_1, x_4, y_4) = Int(x_1, y_1) + Int(x_4, y_4) - Int(x_1, y_4) - Int(x_4, y_1) \quad (2)
\]

**Exercise:** Write the features value of each of the four types in figure 1, as a function of the parameters in figure 3, given the integral image \(Int\) (with size \(n \times m\)).

For a given set of training images, we can extract a large collection of features very fast using the idea above. The hypothesis of Viola & Jones is that a very small number of these features can be combined to form an effective classifier.

**The weak classifiers**

How can these simple features be used to build classifiers? First we will consider weak classifiers. These are of the form - given a single feature vector
$f_i$ evaluated at $x$ the output of the weak classifier $h_i(x)$ is either 0 or 1. The output depends on whether the feature value is less than a given threshold $\theta_i$:

$$h_i(x) = \begin{cases} 1 & \text{if } p_if_i(x) < p_i\theta_i \\ 0 & \text{otherwise} \end{cases}$$

(3)

where $p_i$ is the parity and $x$ is the image-box to be classified. Thus our set of features define a set of weak classifiers. From the evaluation of each feature type on training data it is possible to estimate the value of each classifier’s threshold and its parity variable.

There are two basic methods for determining the threshold value associated with a feature vector. Both methods rely on estimating two probability distributions - the distribution of the values of the feature when applied to the positive samples (face data) and to the negative samples (non-face data). With these distributions, the threshold can be determined either by taking the average of their means or by finding the crossover point [1]. This cross-over point corresponds to:

$$f_i \text{ s.t. } p(f_i|\text{non-face}) = p(f_i|\text{face})$$

(4)

(See figure 4). In this project, that choice is left to the student. See the matlab section.

![Figure 4: An example of how the distribution of feature values for a specific feature may look like over the set of all training samples.](image)

**Feature reduction: AdaBoost**

Boosting (like Bagging) is the process of forming strong hypothesis through linear combination of weak ones. In the context of Viola-Jones face detection
(a binary classification task) weak hypotheses can be represented as the weak classifiers that are derived from the extracted set of features.

The idea of combining weak hypotheses to form strong ones is a logical step, akin to the same logic that we as humans use during the decision making process. For example, to determine that someone is who they say they are we may ask them a series of questions, each one possibly no stronger than the prior, but when the person has answered all the questions we make a stronger decision about the validity of the persons identity.

An implementation of AdaBoost or Adaptive Boosting is shown in the algorithm table 1.

The core idea behind the use of AdaBoost is the application of a weight distribution to the sample set and the modification of the distribution during each iteration of the algorithm. At the beginning the weight distribution is flat, but after each iteration of the algorithm each of the weak hypotheses returns a classification on each of the sample-images. If the classification is correct the weight on that image is reduced (seen as an easier sample), otherwise there is no change to its weight. Therefore, weak classifiers that manage to classify difficult sample-images (i.e. with high weights) are given higher weighting in the final strong classifier. Some of the initial rectangular features selected by AdaBoost in an example run are shown in Figure 5.

![Figure 5: Example of some features selected by AdaBoost.](image)

**Fast decision structure: The attentional cascade**

Increasing the speed of a classification task generally implies that the classification error will increase. This is because the decreasing the time for classification usually involves decreasing the number of evaluations, or weak classifiers. However, this will significantly decrease accuracy. Viola & Jones proposed a method for both reducing the classification time and maintaining classifier robustness and accuracy though the use of a classifier cascade. The logic behind the structure of the cascade is quite elegant, the key being that in the early stages of the tree the classifier structure is largely naive, yet able
Algorithm 1 AdaBoost

**Input:** Example images \((x_1, ..., x_n)\), and associated labels \((y_1, ..., y_n)\), where \(y_i \in \{0, 1\}\). \(y_i = 0\) denotes a negative example and \(y_i = 1\) a positive one. \(m\) is the number of negative examples and \(l = n - m\) the number of positive examples.

**Initialise:** Set the \(n\) weights to:

\[
    w_{1,i} = \begin{cases} 
        (2m)^{-1} & \text{if } y_i = 0 \\
        (2l)^{-1} & \text{if } y_i = 1 
\end{cases} \tag{5}
\]

for \(t = 1, \cdots, T\) do

1. Normalize the weights,

\[
    w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}} \tag{6}
\]

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 the \(w_{t,i}\)’s as

\[
    e_j = \sum_i w_{t,i} |h_j(x_i) - y_i|.
\]

3. Then choose the classifier \(h_t\) as the \(h_j\) that gives the lowest error \(e_j\). Set \(e_t\) to that \(e_j\).

4. Update the weights:

\[
    w_{t+1,i} = w_{t,i} \beta_t^{1-e_i} \tag{7}
\]

where \(e_i = (1)0\) if example \(x_i\) is classified (in)correctly, and

\[
    \beta_i = \frac{e_i}{1-e_i}.
\]

end for

**Output:** A strong classifier defined by:

\[
    h(x) = \begin{cases} 
        1 & \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\
        0 & \text{otherwise}
\end{cases} \tag{8}
\]

where \(\alpha_t = \log \frac{1}{\beta_t}\).
to accurately classify negative samples with a small number of features. As a positive sample progresses through the cascade, assuming that the sample is indeed positively classified, then the process of classification will become finer, and the number of features that are evaluated will increase. (See figure 6).

The use of cascade capitalizes on the fact that in a large image during a detection task a large majority of the sub-windows observed by the scanning classifier (detection-box) will be rejected, since only a small regional area corresponds to the targets (i.e. faces). For this reason, the generality of the first number of stages must be sufficiently high to reduce the number of these false positive sub-windows from progressing into the later stages of the cascade.

The goal in a competitive algorithm is to provide inference with the lowest possible false positives, and highest possible detection rate. Viola & Jones show that given a trained classifier cascade, the false positive rate is the product of all the false positive rates in the chain. Based on this and deeper reasoning for the motivation of the cascade, they provide a generic algorithm for the training process that the cascade must undertake in order to build its stages (See algorithm table 2). In this algorithm both the minimum acceptable detection rate $d$ and the maximum false positive rate $f$ for each layer are required.

Figure 6: The attentional cascade using increasingly specialized classifiers.
Algorithm 2 Cascade Training Algorithm

**Input:** Allowed false positive rates $f$, detection rates $d$ and the target false positive rate $F_{\text{target}}$. A set $\mathcal{P}$ of positive (faces) and of $\mathcal{N}$ negative examples (non-faces).

**Initialise:** Set $F_0 = 1.0$, $D_0 = 1.0$, $i = 0$.

while $F_i > F_{\text{target}}$ and $n_i < N$ do

- $i = i + 1$
- $n_i = 0$, $F_i = F_{i-1}$

while $F_i > f \times F_{i-1}$ do

- $n_i = n_i + 1$
- Use $\mathcal{P}$ and $\mathcal{N}$ to train with $n_i$ features using AdaBoost.
- Evaluate the current cascaded classifier on the validation set to determine $F_i$ and $D_i$.
- Decrease the threshold of the $i$th classifier until the current cascaded classifier has a detection rate of at least $d \times D_{i-1}$ (this also affects $F_i$).

end while

- $\mathcal{N} = \emptyset$.
- If $F_i > F_{\text{target}}$ then evaluate the current cascaded detector on the set of the non-face images and put any false detections into the set $\mathcal{N}$.

end while
Matlab code and Database

You can find all the relevant files and documentation to this project on the course webpage (www.csc.kth.se/utbildning/kth/kurser/2D1427/bik07/). In this section there will be a brief overview of the Matlab code and dataset used in the laboratory tasks. Almost all matlab-functions and scripts have help comments. Just type

>> help filename

in the matlab command prompt to see the help.

The main functions of the lab are listed in their order of execution in the file AdaBoost_main.m. Observe that we will work with windows of size 19×19 pixels. This assumes the detection-box is 19×19.

The function makelist.m runs a script that enables the user to select a directory of images to be used as a database. Both training and validation sets should be in the same directory. The output of this function is a file list_img.txt that lists all the positive samples (faces) and all the negative samples (non-faces) from examining the filenames of the images.

This file is then read by the second function in AdaBoost_main.m, namely the ReadImageFile.m function. It initially reads all the face and non-face image-files listed in list_img.txt and stores each image as a row vector of length 361(= 19²). These row vectors are stacked on top of each other to form two matrices - an \(N_p \times 361\) matrix FaceData containing the face image data and an \(N_n \times 361\) matrix NonFaceData containing the non-face image data. Next the image data is normalized and the integral image of each image is created. For this purpose the image-data is padded with extra boundaries on the right and bottom of each image. Before ReadImageFile.m exits, the resulting matrices cumFace and cumNonFace are saved to disk.

Your first task will be to write a function cumImageJN.m that performs this cumulative sum (see Task 1).

The third function call in AdaBoost_main.m is to makeImagesF.m. This function generates the set of Viola-Jones features that will be used later on. It is strongly recommended the student reads the code in the function violaboxeszero.m. It demonstrates how the feature generation is done in an optimal way with respect to reducing computational complexity and evaluating a specific feature on the set of all images (the cumFace and cumNonFace matrices). There will be questions on violaboxeszero.m later on (see Task 1).

The final function call in AdaBoost_main.m is to TrainCascade.m. In this function the goal is to create a cascade of strong classifiers (see Task 3). In order for the competition between students to be fair, we’ve set an upper-limit for the total number of weak classifiers (in all stages) to 100. How these
100 classifiers are distributed among the strong classifiers of the cascade is up to the students. Just remember you will be judged both on accuracy and speed! To do this, the function TrainCascade.m uses another function called TrainAdaB_stage.m which takes as input the desired number of weak classifiers and creates a single stage (strong classifier) with as many features. It uses the AdaBoost algorithm. When calling TrainAdaB_stage.m you must also specify the ratio of test data that you want to set aside (extracted from all sample data) for validation. Completing this function requires several steps and deeper understanding of the AdaBoost algorithm. There are a couple of functions (findThreshold.m, TestStage.m) inside TrainAdaB_stage.m the student needs to complete (see Task 2).

When AdaBoost_main.m is done it will create a cascaded face-detector that will be used by the function FaceDetector.m. At this final stage of the project the student will need to write a Matlab function that reads in an image and tries to locate the faces in that image, see Task 3. These functions are the ones run on a test set in the final competition at the end of the course.

Database

The database we use for this project can be found on the course webpage. It consists of 2000 positive (ADAFACES) and 4000 negative (ADANFACES) samples. There is, however, a simple trick to double the size of this database. Since our method is feature invariant, every image will have a completely different and unique counterpart - the mirror of that image. So by mirroring every image in the dataset you will have “new” samples. We recommend you do this before you start with the rest of this lab. (See Task 0).

Task 0 - Preliminaries

First of all you need to download the matlab library for the lab at /afs/nada.kth.se/home/1/u16rglu1/Public/2D1427. Under the subdirectory database you will find the database of positive (ADAFACES.zip) and negative (ADANFACES.zip) samples, you need to download these too. When this is done you need to write a matlab script mirror.m that doubles the existing database by creating the mirrored image of every image in the database. Here you may want to use the matlab function fliplr (see matlab Help). Your script should run this for every image of the database and save the mirrored versions.

The function makelist.m creates the list list_img.txt of paths to the database images. This list will be used by ReadImageFile.m for reading
the files. But it could also be used for the mirror.m script above. Just remember to re-run makelist.m after you’ve mirrored the database.

Task I - Integral images and Haar-like features

In the file ReadImageFile.m there is a function call to a cumImageJN.m. It takes as input an image, shaped as a row vector of size $1 \times N$, and returns the corresponding integral image (a row vector of size $1 \times N$).

**Exercise 1 (Programming)**

The first task is to write the function cumImageJN.m, run and test it. *(The matlab functions 'cumsum' and 'reshape' may be useful.)*

As a test you can try this function call:

```matlab
>> cumImageJN(ones(1, 9))
```

the resulting output should be `[1 2 3 2 4 6 3 6 9]`.

Next study the function violaboxzero.m. Try and understand it thoroughly. Think about what the appropriate (most optimal) feature description would be if the images are vectorized (reshaped as vectors instead of matrices).

**Exercise 2 (Programming)**

Finish the code in the function violaboxzero.m. Observe that as an example the code for creating features of the type as in figure 1.1. The students task is to complete the sections for creating features of the other types. *(Noted by 'Fill this part').*

Before going on, check with the lab-assistant if you’ve done this correctly.

**Exercise 3 (Written)**

How are the Viola-Jones features represented in this implementation? What is the benefit of this implementation?
Task II - AdaBoost and Classifier construction

In the function file TrainAdaB_stage.m there are several lines of code missing. Your task here is to fill in these lines and make sure the code runs properly before going on to the next stages. The code is divided into SECTIONS so that the student can find the different sub-tasks more easily, see TrainAdaB_stage.m.

You should note that this function takes a long time to run on the whole dataset. So your code should be first developed and tested on a small control sample mentioned at the end of this Task. We will now make our way through this function and complete some programming and written exercises while doing so.

Classifier Construction via AdaBoost

Look at SECTION 2 of the code. The AdaBoost algorithm requires an initialization of the weights it uses.

Exercise 4 (Programming)

Write this weight initialization. You can choose to have different weights for positive and negative samples.

In SECTION 3 just before each AdaBoost iteration there has to be an weight normalization.

Exercise 5 (Written)

Why does this normalization have to occur? If you did not normalize what would happen?

Exercise 6 (Programming)

Complete the code in SECTION 3 with the weight normalization.

On the line just before SECTION 4 starts, there is the central computation of the algorithm - calculation of the feature-value responses. It is calculated as \( f_{out} = x_{train} * f(:,fNcount) \).
Exercise 7 (Written)
Why is that and what does the comment "feature value for all samples - vectorised so that NO for-loops needed!" really mean?

When you have found out the feature value of a specific feature for ALL samples, you basically have all the information needed to construct the two histograms in Figure 4.

Exercise 8 (Programming)
In SECTION 4 complete the function findThreshold.m to find an appropriate threshold on a feature response to separate the positive and negative sets.

This function should take the feature value responses (featval=fout), the positive and negative sample weights (posWeights=w(1:num_trainp) and negWeights=w(num_trainp+1:end) in the code) and return the optimal threshold with the appropriate parity. Your function call will probably look something like:

[threshold, parity]= findThreshold(featval, posWeights, negWeights);

Note that featval is a vector of length N, where N is the number of training images. The output-variable threshold must be a scalar and parity should be either 1 or -1. You can test your findThreshold function by reassuring yourself that min(featval)< parity*threshold < max(featval).

After checking for number of mis-classifications we need to compute the weighted classification error.

Exercise 9 (Programming)
In SECTION 5 write code to calculate Error (the weighted misclassification error) according to Viola-Jones’ AdaBoost definition. Note that err_classify should only depend on y_train and posComp.

After finding the weighted classification error (Error) you are now ready to update the values of the parameters in the AdaBoost algorithm.

Exercise 10 (Programming)
In SECTION 6 write the code to calculate the updated values for the parameters $\beta_t (beta_t), \alpha_t (alpha_t)$ and the weights $w$. 
In practice if the number of samples is low the variable minError can be zero. To prevent dividing by zero add a small constant (∼ 1e−6) to it.

**Classifier Evaluation**

You now have final strong classifier. You can test it on the portion of test data (x_test, y_test) set aside at the beginning of TrainAdaB Stage.m. See Section 7 in TrainAdaB Stage.m.

**Exercise 11 (Programming)**

Write a function TestStage.m that runs your final strong data on the test data (x_test, y_test). It should output the fraction of true-positives (tp) and true-negatives (tn).

The call to this function should probably look like:

```matlab
[tp,tn] = TestStage(fNbestArray,thetaBestArray,pBestArray,alpha_t_Array,x_test,y_test);
```

where fNbestArray is the array of id’s (row indexes in the feature matrix saved in the file feature.mat) of the features included in the strong classifier, thetaBestArray is their corresponding thresholds, pBestArray their corresponding parities, and alpha_t_Array their corresponding feature weights (α-values). Note that all of the four vectors above are 1 × T where T is the number of features in the strong classifier.

**NOTE:** The definition of true-positive and true-negatives can be found by studying table 1.

<table>
<thead>
<tr>
<th>Predicted class</th>
<th>True class</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
<td>True-Positive (TP)</td>
<td>False-Negative (FN)</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>False-Positive (FP)</td>
<td>True-Negative (TN)</td>
</tr>
</tbody>
</table>

Table 1: The relationships between predicted class vs. true class.

Note in table 1 that $\frac{TP + FN}{|P|} = \frac{FP + TN}{|N|} = 1$, where |P| and |N| are the number of positive and negative samples, respectively. In other words the rates for each parameter is $tp = \frac{TP}{|P|}$, $fn = \frac{FN}{|P|}$, $fp = \frac{FP}{|N|}$ and $tn = \frac{TN}{|N|}$.

The number of true-positives and false-positives will vary depending on the threshold applied to the final strong classifier. The ROC-curve (Receiver Operating Characteristic) is a way to summarize this variation. It is a curve
which plots the fraction of true-positives Vs the fraction of false-positives as the threshold varies from $-\infty$ to $+\infty$. From this curve you can ascertain what loss in classifier specificity (false-positive rate) you will have to endure for a required accuracy (true-positive rate).

**Exercise 12 (Programming)**

Write a function to calculate and plot the ROC-curve for your classifier.

**Debug**

Before you go on, it is highly recommended that you test your final `TrainAdaB_stage` on some simple (not too time-consuming) data in order to reassure yourself that things are working properly. For this task there is some sample data on the course webpage. Download the files `t_images.mat` and `t_features.mat` and replace the names on lines 19 and 20 in the code with these names instead.

**Exercise 13 (Programming)**

Run the AdaBoost classifier training function you have just completed with the small dataset you have just downloaded. When the code runs, make sure that no warnings or errors are detected. Use the call:

```matlab
>> [fN,theta,pBest,alpha_t,tp,tn]=TrainAdaB_stage(10,0.5,0)
```

i.e. where the number of stages (strong classifiers) in the cascade is 10 and the ratio of training to test data is 0.5

Once you are happy that your code runs smoothly and does what you expect it to do, you can analyze the performance and structure of the classifier you have trained. The performance can be evaluated by calculating and visualizing the ROC-curve. While the structure of the classifier can be examined by seeing which features were selected to build your classifier. For the latter task a couple of functions have been written. For instance the function `DisplayFeature.m` allows the visualization of a feature. For example, to illustrate feature number $i$:

```matlab
>> boxen = DisplayFeature(19,19,f(:,i));
>> imagesc(-pBestArray(i)*box); colormap(gray); axis equal;
```
Exercise 14 (Programming/Written)

1. Calculate and plot the ROC-curve for your classifier.

2. Which 10 features were selected for your strong classifier? Display them and save the images.

3. Run the script command and describe what you see.

    >> show_classifier(fN, alpha_t, 1);

PS - Don’t forget to change the lines 19 and 20 back to their original form (images.mat and feature.mat) when you are done. - DS

Task III - Real-time face detection

Now you have a method TrainAdaB-stage that generates a strong classifier. It can be used to create the cascade of classifiers described in algorithm 2. Given this cascade of classifiers it is then possible to run an efficient and fast face detection. Task III is divided into two sections - building the cascade of classifiers and implementing face-detection on test image.

Cascade of Classifiers

In the matlab file TrainCascade.m you have to implement the algorithm described in algorithm table 2 using the function TrainAdaB-stage. The required validation dataset can be marked by the test_ratio parameter in the call to the TrainAdaB-stage function. Remember that the TOTAL number of features in the cascade (weak classifiers) cannot exceed 100. So the choice of how many features to use at each level of the cascade is up to you.

Following the algorithm in table 2, there are 3 important steps here:

1. Updating the negative training set at each level

2. The number of features used at each level is decided by the specified classification rate.

3. Test whether the approach taken in step 2 results in a better classifier than having a fixed number of features at each level of the cascade. (Better means classification performance and speed).
Also remember to save the final cascaded classifier - preferably done by saving the array of feature indexes and the corresponding array of $\alpha$-values and parities.

**Exercise 15 (Programming)**

Complete the function `TrainCascade.m`. Run and debug it.

Once your `TrainCascade` is ready you can run `AdaBoost_main.m`. The program will create a cascade that is saved and ready to be used.

**Face detection**

Now you are ready to create a function `FaceDetector.m` that uses this cascade to detect faces. A skeleton for this function is already given in the file. Functions such as selecting input-image, the sliding window that crops patches over different locations and scales are already implemented and can be useful tools. (See `FaceDetector.m`).

**Exercise 16 (Programming)**

Complete and play with the function `FaceDetector.m`. Upon completion, the function should return the windows of the image which were labeled as faces by your classifier, i.e. the output should be the original image with coloured squares marking the detected faces. Remember to take care of the issue of multiple responses in neighbouring positions. This can be done for example by restricting how close two distinct faces can be or by taking the average position of multiple detections that are relatively close to each other.

If you are using a windows computer there is an entertaining application to the program you’ve just created.

**Exercise 17 (Just for fun)**

Just connect a webcam to your computer and comment lines 11-14 and uncomment lines 15-16 in the `FaceDetector.m` file. Also uncomment the last line in that file. Now run your code again!
Task IV - Evaluation and testing

There is actually not much left to do now. The only thing you need to do now is to reassure yourself that the code complies with the restrictions of the competition;
You need to save the structure cascade.mat as a mat-file in matlab and email it to babak2@kth.se. If everything is done right the competition script will read your cascade into the detector we’ve designed and run it on a test set separate from the training set you’ve been given.
As stated before, the winning group of this competition will be announced at the end of the course. The two criteria that will be measured by the competition script are speed and accuracy (where accuracy means high true-positive and low false-positive).

GOOD LUCK!
Bibliography


