Implementation

Sampling

Like all face detection algorithms, the Viola-Jones approach works by finding common patterns and structures in from some kind of sample data. For a face detector that works reliably with unseen real-world input data, the positive and negative samples used for training have to represent the infinite diversity of this real-world input data as best as possible. Variance normalization is used to equalize all sample images in terms of brightness and contrast so the detector doesn't have to take different lightening conditions into account.

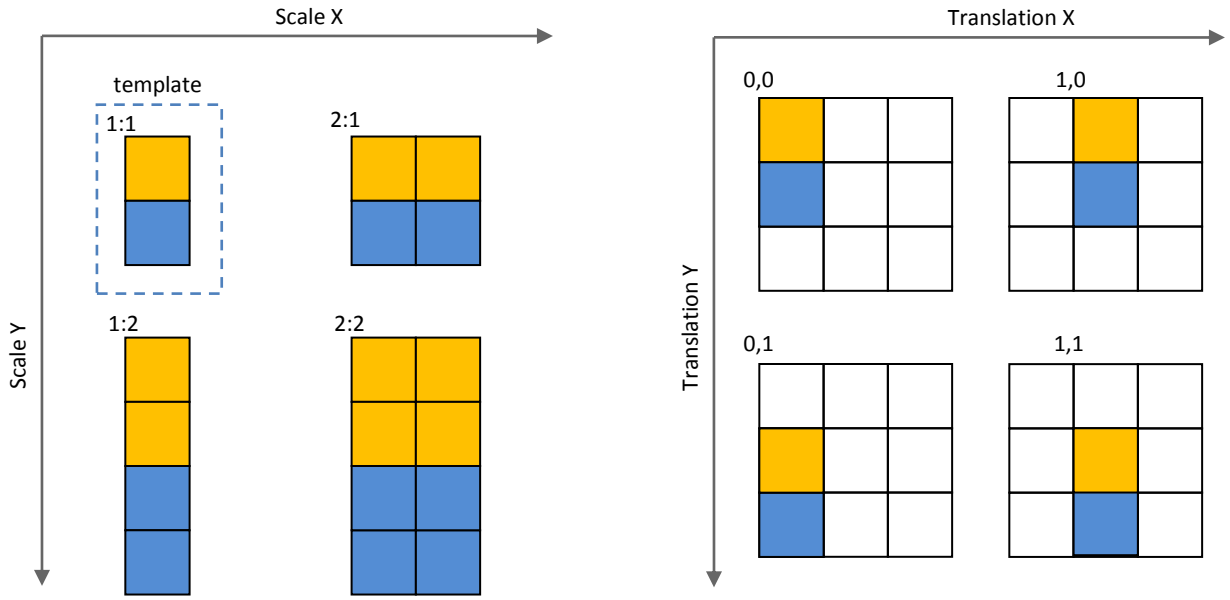
The final application supports extracting 24x24 sample faces used for training in various ways:

- Faces from the CMU image dataset using the provided face location table
- Faces from any image using the OpenCV face detector
- Faces from a single image containing a mosaic of faces (for training data that was exported from matlab)

Rectangle Features

Features are used to separate between faces and non-faces. Each feature is defined by two sets of rectangles. The value of a feature for a specific sample image is calculating by using the difference between the sums all the pixels that are selected by the two rectangle sets respectively.

Features are generated by using a given feature template, the feature of the given type without translation at the smallest possible scale. All the other features of this type are then computed by scaling the template across the x- and y-axis to get all scales of the feature that fit within the sample size. The final set of features is generated by translating each scaled feature to all possible locations within the sample size.



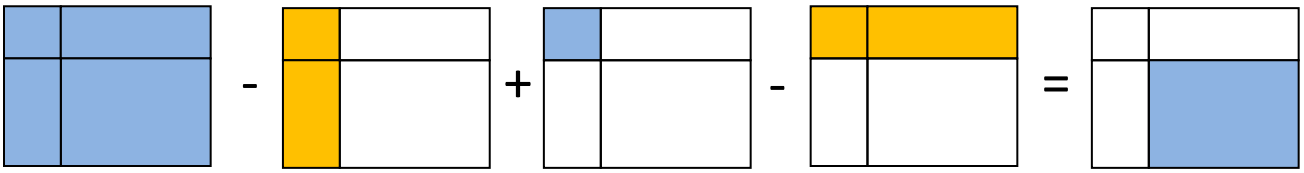
Integral Image

Calculating the pixel sum of a rectangle is a quite costly operation especially given the fact that during detection this process would have to be repeated multiple times for each feature for every single sub-window. Instead of using the original image to calculate the pixel sum for every rectangle, an integral image is first calculated from the original image and then used for all following computations. Each pixel in the integral image corresponds to the pixel sum between the origin and the pixel in the original image. Therefore the pixel sum of a rectangle can be calculated with just a few array references when using the integral image. The integral image itself can easily be computed in a single pass.

Generating the integral image:

```
for (int y = 0; y < h; y++) {
      for (int x = 0; x < w; x++) {
            double pixel = image[x][y];
            if (x > 0) pixel += integral[x - 1][y - 0];
            if (y > 0) pixel += integral[x - 0][y - 1];
            if (x > 0 && y > 0) pixel -= integral[x - 1][y - 1];
            integral[x][y] = pixel;
      }
}
```

Calculating the pixel sum of an area using the integral image:



When using an integral image, the amount of computation required to calculate the pixel sum of any area regardless of size is constant. Therefore when using more complex features, the feature value can be calculated more efficiently by subtracting the pixel sum of one of the rectangle sets from the total area sum of the rectangle.

AdaBoost Training

The iterative boosting consists of three major steps. First the weights are normalized. Then the feature that yields the least weighted minimum error is chosen. Afterwards the weights are updated with respect to the chosen classifier and its error. Every weight corresponds to a sample image from the training data and how well it can be classified by all the weak classifiers that have been chosen before. Therefore the weights that correspond to sample images that are classified correctly by the chosen classifier are reduced with respect to its error. Due to the updated weights, in the next iteration of boosting, the next classifier will focus on sample images that have been misclassified by previous classifiers.

Thresholding and Weak Classifiers

From the large set of features we have to select those that are useful for face detection. Therefore each feature will be tested against the entire training data. The feature value for each sample image will be calculated in order to find the threshold that best separates faces and non-faces. The threshold that yields the lowest weighted error can easily be determined using a simple brute-force approach. Each threshold has a numeric value as well as a polarity that indicates whether the faces are on the positive or on the negative side of the threshold value.

Even the best threshold for a specific feature will still yield a significant error, and is therefore called "weak" classifier. The error rates of usable weak classifiers are typically between 15% and 45%.

The thresholding required for each weak classifier has to be repeated in every iteration of the boosting process because the weights change with every iteration which will take a lot of time even on a modern processor. In order to speed up the training, the feature values for all samples however remain constant and can therefore be calculated and stored beforehand. The feature value for every single feature for every single sample image has to be stored which will easily take up many gigabytes of space so these values have to be stored on-disk.

Strong Classifier

A useful "strong" classifier with a minimal error rate can be constructed by combining multiple weak classifiers. But a strong classifier is not just a sequence of weak classifiers but also their corresponding weighted error rates that were determined during training. The error rate of a classifier is factored in during the evaluation of a sub-window as it represents how trust-worthy a specific classifier is.

The final application uses a single strong classifier. The solution suggested by Viola-Jones suggests a cascade of manually fine-tuned strong classifiers. This solution trades in a small amount of accuracy and quite a lot of development time for a large gain in performance. However adequate performance can be achieved by a single strong classifier. Suppose we have a 200 features strong classifier, if there is positive response from the first couple of weak classifiers then the final result is likely to be negative, therefore we can abort early, without computing all feature values.

The strong classifier can easily be tweaked to minimize false positives (at the cost of some false negatives) by adjusting the minimum ratio between positive and negative responses of the weak classifiers that is required for a positive detection.

Face Detector

The final strong classifier can only classify images of a certain scale and a certain size, so we need to slide a sub-window through the image and classify every single sub-window. We will then scale down the image and run sub-window classification again. Slightly scaling down the image further and further allows us to detect faces of any scale. The bounds of the detection rectangle on the scaled image have to be adjusted of course so they match the original image and not the scaled image.

Because the training data is not perfectly aligned and scaled, the detector will be tolerant to alignment and scale to a certain degree. Consequently the detector will come up with multiple detections at slightly different positions for a single face on the image. To solve this problem we just have to collect the detections that overlap with each other significantly and then use average location/size as the final detection rectangle.

Since normalized sample data is used for training, every sub-window of the real-world input has to be normalized as well. Before training the image is normalized by dividing all pixel values by the standard deviation. This approach is however not feasible during detection. Since each sub-window has a different standard deviation, preprocessing of each and every sub-window using data from the original image would be required thus reducing the benefits of using an integral image to nothing.

Instead the normalization can be achieved by in a highly efficient manner by using two integral images. The first one is used to calculate the area sum of the input image and the second one for the squared area sum. Using these two integral images, the standard deviation of a sub-window can quickly be computed using the equation below.

Equation for Standard Deviation (σ):

$$\sigma^2 = \frac{1}{n}\sum x^2 - m^2$$

The mean (m) can easily be calculated from the first integral image and the squared pixel sum ($\sum x^2$) can be calculated from the second integral image.

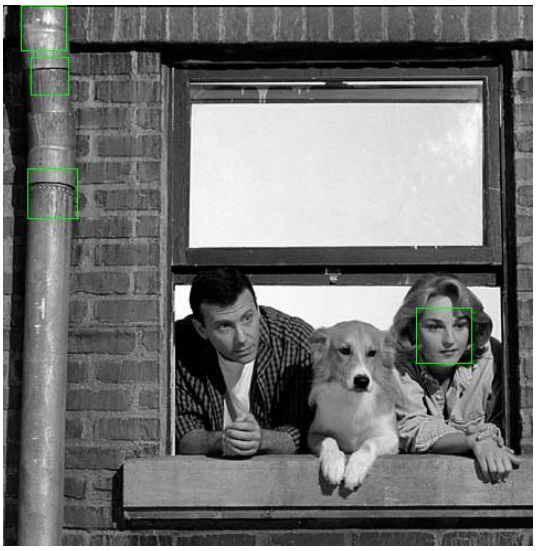
Post-normalization of the area sum is the same as pre-normalization of all pixel values:

$$\frac{1}{\sigma}\sum x = \sum \frac{x}{\sigma}$$

During detection normalization is achieved by simply dividing the feature value (sum of pixels) by the standard deviation ($\frac{1}{\sigma}\sum x$), instead doing it for every single pixel value ($\sum \frac{x}{\sigma}$).

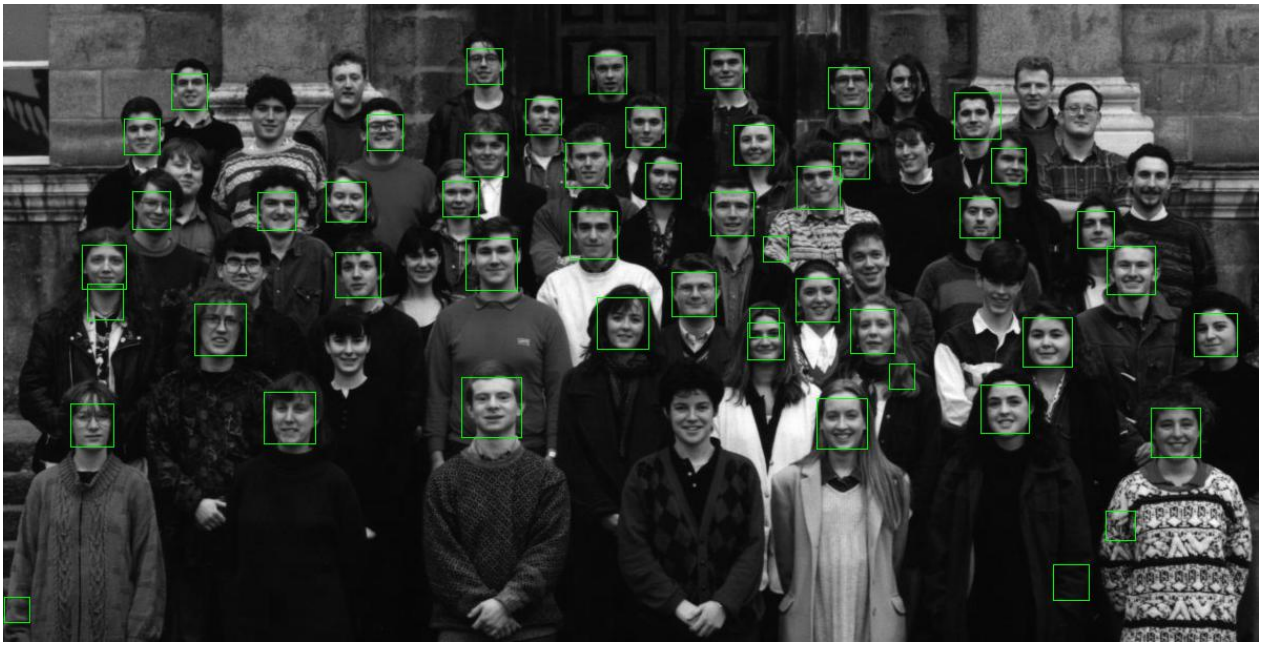
Evaluation

The use of rectangular features turns out to be very efficient and reasonable accurate. However training the detector requires huge amounts of processing. Due to the lack of hardware, the final detector uses only 23 features, which were selected in approximately 5 hours of training on a 2 GHz dual-core processor. Nevertheless this detector is able to identify up to 100% of the faces correctly depending on the image that is used for testing. On average however the detector will only detect half of the faces because the classification threshold is chosen very conservatively in order to eliminate false positives.



The left image (voyager2.gif) illustrates how the detector works on simple images. All faces can be classified correctly because they all look at the camera upfront so there is no rotation and there are no obstructions like hair or glasses. There are no false positives because the background is very plain and has very little texture that could be confused with face.

The right image (madaboutyou.gif) however shows how the detector has trouble with correctly classifying rotated faces. The pipe to the left of the image is misclassified as face multiple times. This can easily be understood by looking the features that are used in the final classifier. First of all there is usually a clear change in brightness between the face and the background to the left and to the right of the face. And secondly there are unusually dark areas slightly above the center of the sub-window representing the eyes. The parts of the pipe that have been misclassified as faces clearly exhibit those features.



This last image (class57.gif) shows that the current implementation is already quite promising, even though it's not perfect. Increasing accuracy and eliminating false positives is only a matter of using more features and fine-tuning.