How to find a face

You can do it, and your camera can do it too – but how does face detection work?

The first time I looked at the rear display of a camera that had face-detection software, it was an interesting experience. Point it at a person, and the software would superimpose a coloured square over that person's face. This would enable you to more easily frame the photo, ensure correct exposure for the face compared to the rest of the scene and make sure that the face was properly focused.

So how did it manage it? What's so special about a face that enables the camera to identify that *this* set of pixels is a face, but *that* set isn't? And in real-time too? The camera doesn't have a chip with great processing power, either, so the algorithm must be extremely efficient. We should also remember that over the years, camera face-detection software has become pretty advanced. You can now expect the software in your point-and-shoot camera to work out not just the location of a face but also whether the person is smiling, and to take the photo automatically if so.

Back in 2001, Paul Viola and Michael Jones invented a new framework for detecting arbitrary objects and refined it for face detection. The algorithm is now known as the Viola-Jones framework.

The first thing to realise is that face detection, whether by Viola-Jones or not, is not an exact science. Just like we humans can be fooled by images that seem to contain a face when in reality they do not, so face-detection software can be hoodwinked. This phenomenon is known as pareidolia: the apparent recognition of something significant (usually a face or a human form) in something that doesn't have it naturally. There are many examples of this, the most prominent being perhaps the Face on Mars – a photo taken in the Cydonia region of Mars that appeared to contain a human face in the rock – or the image of the Virgin Mary that an American lady found in a grilled cheese sandwich. Face detection software can be fooled too, so we talk about the algorithm's rate of false positives (detecting a face when there is none) or false negatives (not detecting a face that's present).

“You can now expect your camera to work out whether a person is smiling”

A good guess

The Viola-Jones method has a very high accuracy rate – the researchers report a false negative rate of less than one per cent and a false positive rate of under 40 per cent, even when used with the simplest filter. (The full framework uses up to 32 filters, or 'classifiers'.)

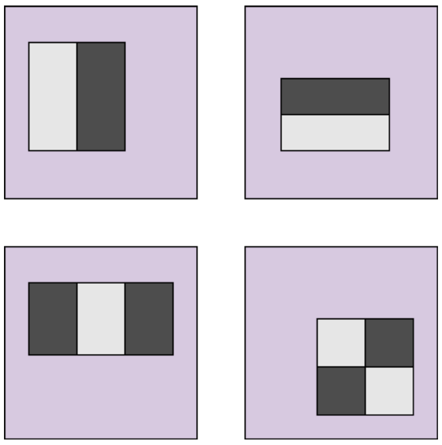
But we're getting ahead of ourselves. The breakthrough for Viola and Jones came when they didn't try to analyse the image directly: instead, they started to analyse rectangular 'features' in the image. These features are known as 'Haar-like features', due to the similarity of the analysis ►

Spotlight on… Fooling the system

Now that we see how the Viola-Jones system works, it's instructive to think of ways to fool it. After all, since face detection is the first step in face recognition, how can a master criminal who skirts the law avoid automatic systems and pass unnoticed?

Since the first classifier rejects the majority of non-face images, and it is the simplest, it makes sense to target it. If you fool that classifier, none of the others would be activated. To do that means changing the pattern of light and dark from eyes to cheekbones, or across the bridge of the nose. Perhaps the best idea, and one that probably wouldn't elicit too much finger-pointing (after all, it would be silly to become more noticeable to people in the street in an attempt to hide from automated systems), would be to use the black makeup used by American football players and accent under and over the eye. That would make the eyes lighter than the surroundings and thereby possibly fool the system.

Other research indicates that using dark makeup applied asymmetrically would work too, but this is more noticeable. One day, perhaps, it will be normal to apply makeup in this fashion in order to maintain privacy. ■


▲ **Figure 1: Types of features used in Viola-Jones.**

► of complex waveforms with Haar wavelets. These are simple square waveforms, and are named after Alfréd Haar, a Hungarian mathematician whow as working at the turn of the 20th century.

The first thing to do is strip colour from the image and work with a simple greyscale colour space. A simple way of doing this is to transform the image to the 'YCbCr' colour space, where the Y component is a measure of intensity or luminance in the original image (the other components can be ignored for now, although they could be used to distinguish flesh tones at a later detection stage). To calculate the Y component (the luma) of a pixel, use the formula Y = 0.299R + 0.587G + 0.114B, where the values R, G, B are the red, green and blue components of a pixel's value.

Smiling detection

Once you have a functioning Viola-Jones system, it's pretty easy – if tedious – to modify it to detect smiling faces. The trick is to merely train the system on another set of faces, all of which are smiling. Because the smile generally involves baring the teeth to a certain extent, the smile is lighter than its surroundings. This greater difference in intensity would be easily noticeable by a rectangular feature. ■

Now that we have an image defined as a set of intensity values (from 0 to 255), we can use them to look for large-scale rectangular features in the image. We do this by summing the intensity values in various rectangular blocks. Using these sums we can detect 'darker' blocks adjacent to 'lighter' blocks since the sum of the intensity pixels in a dark block will be less than the sum in a light block. These adjacent blocks are known as 'features'. (Note that, despite the fact we're using Viola-Jones to detect faces, the word 'feature' does not pertain to facial features in any way.)

> “If it passes through all the classifiers, the subwindow is classified as a face”

Viola-Jones defines several different types of features (see Figure 1). The first feature is a light block next to a dark one, vertically. The second is the same but horizontally. The third is a light block sandwiched between two dark blocks (or vice versa). The fourth is a four-rectangle feature as shown. Note that in a feature the rectangular blocks are the same size. The features can be at any scale or at any position in the image, so the features shown in Figure 1 are just examples at an arbitrary size.

A particular feature can be positioned and scaled onto the original image. The feature value is calculated as the sum of the pixel intensities in the light rectangle(s) minus the sum of the pixels in the dark rectangle(s). The value of the feature is then used in a filter to determine if that feature is 'present' in the original image. It still sounds computationally expensive (that is, slow). To improve the speed of summing the intensities of the pixels in a given rectangle, we make use of a trick. For a given image we can calculate what's known as the integral image at every point. This is merely the sum of the pixels of the rectangle from the upper-left corner to the given point, and it can be calculated for every point in the original image in a single pass across the image. The word 'integral' comes from the same meaning as 'integrating' – finding the area under a curve by adding together small rectangular areas.

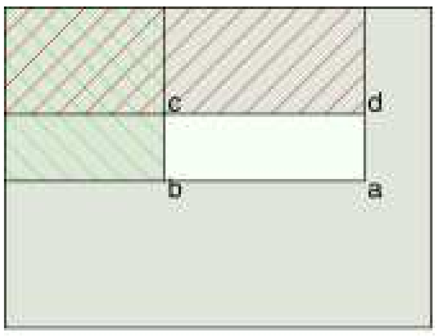
Intense calculations

Once we have the integral image for every point, we can calculate the sum of intensities for any arbitrary rectangle by using the identities shown in Figure 2. We want to calculate the sum of the pixels for the rectangle *abcd*. We start off with the integral image for point *a*, and then subtract the integral images for points *b* and *d*. This unfortunately takes off too much (as shown by the double hatching) and so we add back in the integral image for point *c*. As you can see, it's a very quick calculation once we have generated all the integral images, presumably stored in an array.

We now have in place all the requisites to calculate the summed intensity values for a set of features. But what are we going to do with them? Compared with analysing the pixels directly, features provide a coarse, low-resolution view of the image. They're good at detecting edges between light and dark, bars, and other simple structures.

What we do is to implement a learning system. We give the face detection routine a set of 24 x 24 images of faces (and another set of 24 x 24 images of things that are not faces) and train the routine to recognise faces and discard non-faces. Viola and Jones used a database culled from the internet of about 4,000 faces and 10,000 non-faces to do the training. What's involved in the training?

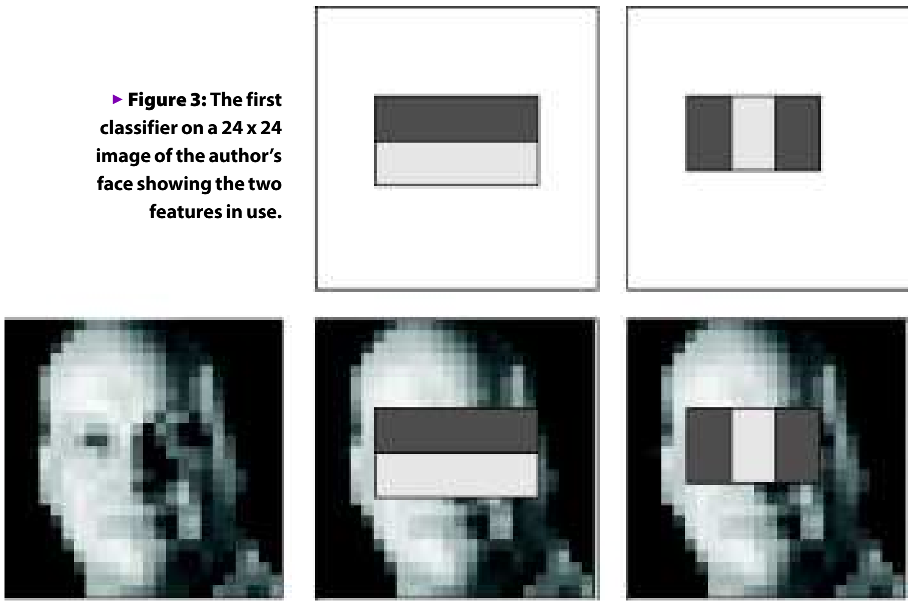
Using 24 x 24 images, there are some 45,000 different ways to place one of the four types of feature onto the image. For example, for the first type of feature, you could have rectangles one pixel wide by two pixels deep, all the way up to 1 x 24, then 2 x 2 to 2 x 24, and so on. These different-sized features would be placed in various positions across the image to test every possible feature for every possible size at every possible position. Note that the number of possible features, 45,000, is far greater than the number of pixels in a 24 x 24 image (a mere 576 pixels) and so you must reduce the number you use.



▲ **Figure 2: Calculating sums of intensities using integral images.**

Other objects

Although Viola-Jones was originally developed to enhance the speed of face detection, it can be used to detect other objects. Examples include security systems that can detect pedestrians rather than birds, cats and dogs. Microsoft's Project Natal is a very sophisticated system that can detect individual limbs (arms, legs, and even fingers) in 3D while ignoring any furniture, such as chairs, around the room. ■


▸ **Figure 3: The first classifier on a 24 x 24 image of the author's face showing the two features in use.**

Remember, you're calculating the difference between the pixel sum for the light and dark parts of the feature. You could decide on a threshold for difference (which could be tweaked during training) whereby a feature is assumed to be detected or not. Using this, you would then apply every one of the 45,000 possible features to your training set.

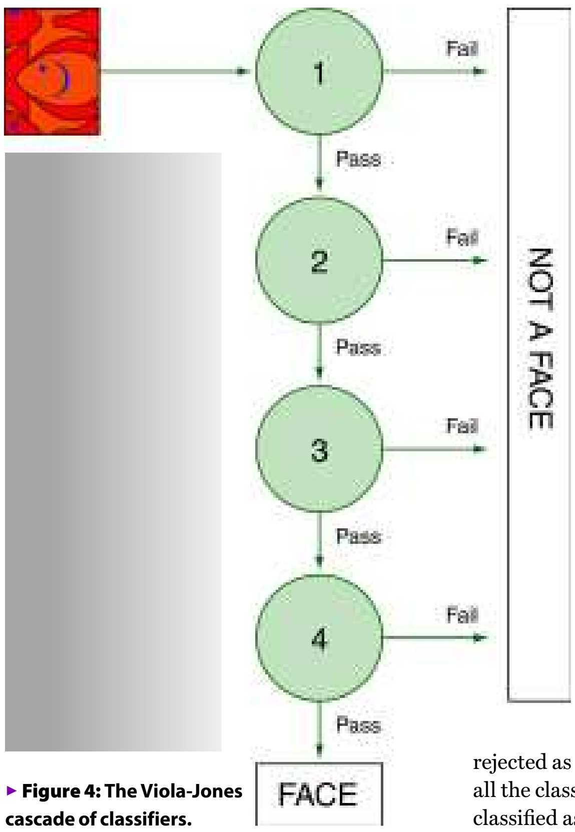
What you'd find is that certain features are worthless at determining whether an image is a face or not – that is, there would be no correlation between the feature identifying a face and it not being one, and vice versa. These would be rejected. However, some would have a high success rate at rejecting non-faces, and this is where the training comes in.

Face or not?

Viola and Jones then experimented with the remaining features to determine the best way of using them to classify an image as 'face' or 'non-face'. After experimentation they decided to use a training system called AdaBoost to build a classifier. AdaBoost is an artificial intelligence (AI) technique similar to a neural network, devised to combine 'weak' features into a 'stronger' classifier. Each feature within a classifier is assigned a weighting (tweaked during training) that defines how 'accurate' that classifier is. Low weighting means a weak feature, high weighting a stronger one. Add up the weightings of the features that test positive for a particular image and if that sum is above a threshold (again, tweakable during training) then that image is determined to be a face.

As it happens, during this training they found that there were two features that, when combined and properly tuned by AdaBoost into a single classifier, would pass though 100 per cent of the faces with a 40 per cent false positive rate (60 per cent of the non-faces would be rejected by this classifier). Figure 3 shows this simple classifier in action. It uses two features to test the image: a horizontal feature that measures the difference between the darker eyes and the lighter cheekbones, and the three-rectangle feature that tests for the darker eyes against the lighter bridge of the nose.

Although they had been trying to implement a strong classifier from a combination of 200 or so weak classifiers, this early success prompted them to build a cascade of classifiers instead of a single large one (see Figure 4). Each subwindow of the original image is tested against the first classifier. If it passes that classifier, it's tested against the second. If it passes that one, it's then tested against the third, and so on. If it fails at any stage of the testing, the subwindow is rejected as a possible face. If it passes through all the classifiers then the subwindow is classified as a face. The interesting thing is that the second and subsequent classifiers are not trained with the full training set. Instead they are trained on the images that pass the prior classifiers in the chain.



▸ **Figure 4: The Viola-Jones cascade of classifiers.**

The second and subsequent classifiers are more complex and have more features than the first one, so they require more computational time. It seems remarkable then that there is such a simple classifier that will reject so many sub-windows without having to go through the calculations required for the more complex classifiers. Remember that it's far more likely that a random image contains no face at all, and so being able to reject the majority of sub-windows with as small a set of calculations as possible is a good thing.

Final reckoning

The eventual cascade developed by Viola and Jones had 32 stages and used a total of 4,297 features (out of the original total of 45,000). The first classifier uses the two features described above, the next uses five further features and rejects 80 per cent of the non-faces. The next three use 20 features and subsequent ones even more. The whole training took several weeks.

Finally, when presented with an actual image, the face detector scans the complete image at multiple scales and with multiple locations per scale. The researchers found that scaling the image by a factor of 1.25 each time produced the most accurate results. They also discovered that they didn't need to test each individual location: they could skip a couple or so pixels each time and still get good results.

The overall result is that the Viola-Jones method produces accurate results very quickly, and certainly fast enough for the limited processing power of the average point-and-shoot camera to cope with. **PCP**

Julian M Bucknall has worked for companies ranging from TurboPower to Microsoft and is now CTO for Developer Express.
feedback@pcplus.co.uk