Performance Characterization in Computer Vision

Adrian F. Clark and Christine Clark
VASE Laboratory, Computer Science and Electronic Engineering
University of Essex, Colchester, CO4 3SQ, UK
{alien,cc}@essex.ac.uk

This document provides a tutorial on performance characterization in computer vision. It explains why learning to characterize the performances of vision techniques is crucial to the discipline's development. It describes the usual procedure for evaluating vision algorithms and how to assess whether performance differences are statistically valid. Some remarks are also made about the difficulties of vision 'in the wild.'

Contents

1 Introduction **2**

2 A Statistical Preamble **3**

3 The Performance Assessment and Characterization Processes **4**

4 Assessing an Individual Algorithm **7**
4.1 The Receiver Operating Characteristic Curve 7
4.2 Precision–Recall and Related Curves 8
4.3 Confusion Matrices 8

5 Comparing Algorithms **9**
5.1 Using ROC or Precision–Recall Curves 9
5.2 M^{c}Nemar's Test 10
5.3 Comparing Several Algorithms 11

6 Determining Robustness to Noise **12**

7 So Is This The Whole Story? **13**

This tutorial is derived from one developed under the ægis of a project called *Performance Characterization in Computer Vision* (project 1999-14159, funded under the European Union's IST programme). See http://peipa.essex.ac.uk/benchmark/ for further information.

1 Introduction

The discipline variously known as *Computer Vision*, *Machine Vision* and *Image Analysis* has its origins in the early artificial intelligence research of the late 1950s and early 1960s. Hence, roughly three generations of researchers have pitted their wits against the problem. The pioneers of the first generation worked with computers that were barely capable of handling image data — processing had to be done line-by-line from backing store — and programs almost always had to be run as batch jobs, ruling out any form of interaction. Even capturing digital images was an impressive feat. Under such difficult conditions, the techniques that were developed were inevitably based on the mathematics of image formation and exploited the values of pixels in neighbouring regions. Implementing them was a non-trivial task, so much so that pretty well any result was an impressive achievement.

The second generation of researchers coincided with the birth of the workstation. At last, an individual researcher could process images online, display them, and interact with them. These extra capabilities allowed researchers to develop algorithms that involved significant amounts of processing. A major characteristic of many algorithms developed during this second generation was the quest for optimality. By formulating and manipulating a set of equations that described the nature of the problem, a solution can usually be obtained by a least-squares method which, of course, is in some sense optimal. Consequently, any number of techniques appeared with this 'optimality' tag. Sadly, none of the papers that described them were able to provide credible *experimental* evidence that the results from the optimal technique was significantly better than existing (presumably sub-optimal) ones.

We are now well into the third generation. Computers, even PCs, are so fast and so well-endowed with storage that it is entirely feasible to process large datasets of images in a reasonable time; this has resulted use of vision techniques that learn[1] and, of more relevance to this document, to assessing the performance of algorithms. Perhaps the most visible (no pun intended) aspect of the latter is the competitions that are often organized in association with major vision conferences. These essentially ask the question "which algorithm is best?" Although a natural enough question to ask, it lacks subtlety and is potentially rather dangerous: if the community as a whole adopts an algorithm as "the standard" and concentrates on improving it further, that action can stifle research into other algorithms. A better approach is to make available a "strawman" algorithm which embodies an approach that is known to work but does not quite represent the state of the art. This might be, for example, the "eigenfaces" approach [1] without refinements for face recognition, the Canny edge detector, and so on. Authors can use the strawman for comparison, and anything that out-performs it is a good candidate for publication; conversely, anything that performs less well than the strawman needs improvement.

If asking which algorithm is best is unsubtle, then what is a more appropriate question? We believe researchers should be asking "why does one algorithm out-perform another?" To answer the latter question, one must explore what characteristics of the inputs affect the algorithms' performances and by how much. In fact, one can carry this process out on an algorithm in isolation as well as comparing algorithms. This is what is meant by *performance characterization*, the subject of this tutorial, and is a closer match to the way knowledge and understanding are advanced in other areas of science and engineering.

It may seem from the above that performance assessment and characterization are intel-

[1] Hoorah!

lectual exercises, divorced from the gritty realities of applying vision techniques to real-world problems; but nothing could be further from the truth. Vision techniques have a well-deserved reputation for being fragile, working well for one developer but failing dismally for another who applies them to imagery with slightly different properties. This should not come as a surprise, for very few researchers have made any effort to assess how well different algorithms work on imagery as its properties differ — as the amount of noise present changes, say — never mind making the algorithms more robust to them. So, far from being an abstract exercise, performance characterization is *absolutely essential* if computer vision is to escape from the research laboratory and be applied to the thousands of problems that would benefit from it. Indeed, the first paper reviewing this area has only fairly recently appeared [2].

The process conventionally adopted for performance assessment and characterization has not yet been expounded; that is done in Section 3, following a short statistical preamble in Section 2. Section 4 and Section 5 then describe the underlying statistical principles and describes those statistical tests, displays and graphs in common use for characterizing an individual algorithm and for comparing algorithms respectively. Finally, Section 7 gives some concluding remarks.

2 A Statistical Preamble

The amount of statistical knowledge needed in performance work is not that great; all that is needed to get going is an understanding what mean, variance (and standard deviation) measure — though some care is sometimes required to interpret results correctly. It is also important to realise that there are many statistical distributions (*e.g.,* tossing coins involves a binomial distribution), and that several of them approach the Gaussian distribution only for large numbers of samples.

Having mentioned coin tossing, let us use this to think about assessment. Imagine taking a fair coin and tossing it a number of times. Let us say that we obtain:

1. 3 heads from 10 tosses;
2. 30 heads from 100 tosses;
3. 300 heads from 1,000 tosses.

Let us now consider which of these is the most surprising.

A naïve answer is that they are all equally surprising as

$$\frac{3}{10} = \frac{30}{100} = \frac{300}{1000}$$

but this is the wrong way to interpret the results. We know that a coin toss is able to produce only two results, so it obeys a *binomial* distribution

$$P(t) = \binom{N}{t} p^t (1-p)^t \tag{1}$$

where $P(t)$ is the probability of obtaining t successes (heads) from N trials, p is the probability of success in a single trial and

$$\binom{N}{t} = \frac{N!}{k!(N-k)!}. \tag{2}$$

The mean of the distribution is Np and the variance $Np(1-p)$. As the coin is fair, $p = \frac{1}{2}$ and the three cases work out as:

1. The expected number of heads (*i.e.,* the mean) is $Np = 5$. The standard deviation is $\sqrt{Np(1-p)} = \sqrt{10 \times \frac{1}{2} \times \frac{1}{2}} = 1.58$. Hence, 3 heads is $(5-3)/1.58 \approx 1.3$ standard deviations from the mean.

2. The expected number of heads is 50 and the standard deviation is $\sqrt{100 \times \frac{1}{2} \times \frac{1}{2}} = 5$. 30 heads is $(50-30)/5 = 4$ standard deviations from the mean.

3. The expected number of heads is 500 and the standard deviation is $\sqrt{1000 \times \frac{1}{2} \times \frac{1}{2}} = 15.8$. 300 heads is $(500-300)/15.8 \approx 13$ standard deviations from the mean.

This means that — as you probably expected — the third case is by far the most surprising.

There are three things to take from this simple example. Firstly, as the number of experiments increases, results far away from the mean become increasingly unlikely to happen purely by chance. Secondly, intuition based around the mean and standard deviation, coupled with a notion of the distribution involved, is usually enough to get an idea of what is going on. And thirdly, it is possible to predict how likely events are to happen by chance.

3 The Performance Assessment and Characterization Processes

There are few occasions when it is possible to predict the performance of an algorithm analytically: there are normally too many underlying assumptions, or the task is just too complicated (but see [3] for a rare exception). So performance is almost universally assessed empirically, by running the program on a large set of input data whose correct outputs are known and counting the number of cases in which the program produces correct and incorrect results. Each individual test that is performed can yield one of four possible results:

True positive: (also known as *true acceptance* or *true match*) occurs when a test that should yield a correct result does so.

True negative: (also known as *true rejection* or *true non-match*) occurs when a test that should yield an incorrect result does so.

False positive: (also known as *false rejection*, *false non-match* or *type I error*) occurs when a test that should yield a correct result actually yields an incorrect one (in vision testing, usually the wrong class of output).

False negative: (also known as *false acceptance*, *false match*, *false alarm* or *type II error*) occurs when a test that should yield an incorrect result actually yields a correct one (*e.g.,* finding that a face detector works on a picture of a coffee cup).

There is occasionally some confusion in the literature over the terms “false negative” and “false positive,” which is why their meanings have been given here. The testing procedure involves keeping track of these four quantities. Performance assessment work normally uses them with little additional consideration: the algorithm with the highest true rate (or, equivalently, the lowest false rate) is normally taken in comparisons and competitions to be the best.

(a) Original image

(b) Corresponding template

(c) Ambiguity at edge of template

Figure 1: An object against a plain background

To be able to perform testing in this way, each individual test requires three pieces of information:

1. the input to the program under test;
2. the corresponding expected output from the program under test;
3. whether this output corresponds to a success or a failure.

Vision researchers rarely test explicitly for failures, *e.g.* by running a vision algorithm on an image whose pixels are all set to the same value.

To fix these ideas in our minds, let us consider the example of using a procedure to try detecting in an image an object surrounded by a plain background. Specifically, Figure 1(a) shows a vase against a plain background. A template image, Figure 1(b), can be constructed to determine which pixels are to be regarded as 'vase' pixels; the rest would be regarded as 'background'. On applying the procedure, if a pixel is classed as 'vase' and it is known from the template to be part of the vase, then this pixel is a *true positive*. If a pixel is classed as 'background' and it is known to be part of the background (*i.e.,* not vase), then this pixel is a *true negative*. If a pixel is classed as 'vase' but it is known to be part of the background, then this pixel is a *false positive*. If a pixel is classed as 'background' but it is known to be part of the vase, then this pixel is a *false negative*.

While it is obvious that the performance depends on how accurately the template has been determined, these values give a measure of algorithm performance. In particular, we should expect both false positives and false negatives to occur most frequently in the region where the object meets the background because there will be pixels where there are contributions from both the object and its background, as illustrated in Figure 1(c). It would be wise to weight errors in this region less than errors elsewhere in the image, or just to ignore these regions.

It must be appreciated that there is always a trade-off between true positive and false positive detection. If a procedure is set to detect all the true positive cases then it will also

tend to give a larger number of false positives. Conversely, if the procedure is set to minimize false positive detection then the number of true positives it detects will be greatly reduced. However, tables of true positives *etc.* are difficult to analyze and compare, so results are frequently shown graphically using ROC or similar curves (see Section 4).

It should in principle be possible to compare the success rates of algorithms obtained using different datasets; but in practice this does not work. This is, in effect, the same as saying that the datasets used in performing the evaluations are not large and comprehensive enough, for if they were it *would* be possible simply to compare success rates. The number of ways in which image data may vary is probably so large that it is not feasible to encompass all of them in a dataset, so it is currently necessary to use the same datasets when evaluating algorithms — and that means using the same training data as well as the same test data. Sadly, little effort has been expended on the production of standard datasets for testing vision algorithms until recently; the FERET dataset (*e.g.,* [4]) is probably the best example to date.

Most papers currently compare algorithms purely on the basis of their ROC or equivalent curves. However, this is dangerous, for this approach takes no account of the number of tests that has been performed: the size of the dataset may be sufficiently small that any difference in performance could have arisen purely by chance. Instead, a standard statistical test, M^{c}Nemar's test (see Section 5), should be used as it takes this into account. M^{c}Nemar's test requires that the results of applying both algorithms on the same dataset are available, so this fits in well with the comments in the previous paragraph.

An argument that is often put forward is that vision algorithms are designed to perform particular tasks, so it only makes sense to test an algorithm on data relating precisely to the problem, *i.e.* on real rather than simulated imagery. While this is true to a certain extent — the range of applications of vision tasks is indeed vast — it ignores the fact that there are generic algorithms that underlie practically all problem-specific techniques, *e.g.* edge detection. Indeed, this really illustrates the distinction between two different types of testing:

technology evaluation: the response of an algorithm to factors such as adjustment of its tuning parameters, noisy input data, *etc.*;

application evaluation: how well an algorithm performs a particular task;

where the terminology has been adapted from that in [5]. Technology evaluation is one example of performance characterization, and we shall return to this topic towards the end of this document.

To illustrate the distinction between technology and application evaluation, let us consider an example that will be familiar to most computer vision researchers, namely John Canny's edge detector [6]. Technology evaluation involves identifying any underlying assumptions (*e.g.,* additive noise) and assessing the effects of varying its tuning parameters (*e.g.,* its thresholds, the size of its Gaussian convolution mask). This is best done using *simulated* data, as it provides the only way that all characteristics of the data can be known. Conversely, application evaluation assesses the effectiveness of the technique for a particular task, such as locating line-segments in fMRI datasets. This second task must, of course, be performed using real data. If the former is performed well, the researcher will have some idea of how well the algorithm is likely to perform on the latter simply by estimating the characteristics of the fMRI data — how much and what type of noise, and so on.

4 Assessing an Individual Algorithm

Tables of true positives *etc.* are difficult to analyse and compare. Hence, researchers have introduced methods of presenting the data graphically. We shall consider two of these, the *receiver operating characteristic* (ROC) curve and the *precision–recall* curve. We shall also consider a display that is frequently used in describing the performance of classification studies, namely the *confusion matrix*. Other measures and displays do exist, of course; many of them are described in [7].

4.1 The Receiver Operating Characteristic Curve

A ROC curve is a plot of false positive rate against true positive rate as some parameter is varied. ROC curves were developed to assess the performance of radar operators during the second World War. These operators had to make the distinction between friend or foe targets, and also between targets and noise, from the blips they saw on their screens. Their ability to make these vital distinctions was called the receiver operating characteristic. These curves were taken up by the medical profession in the 1970s, who found them useful in bringing out the *sensitivity* (true positive rate) versus *specifity* ($1 -$ false positive rate) of, for example, diagnosis trials. ROC curves are as interpreted as follows (see Figure 2):

- the closer the curve approaches the top left-hand corner of the plot, the more accurate the test;
- the closer the curve is to a 45° diagonal, the worse the test;
- the area under the curve is a measure of the accuracy of the test;
- the plot highlights the trade-off between the true positive rate and the false positive rate: an increase in true positive rate is accompanied by an increase in false positive rate.

It should be noted that there does not appear to be a convention as to the orientation of the plot, so one encounters a variety of orientations in the literature; in such cases, the above interpretation must be adjusted accordingly.

Figure 2 shows ROC curves for a very good, a good and a very poor (worthless) test. As stated above, the area under each curve gives a measure of accuracy. An area of unity represents a perfect test, while a measure of 0.5 (*e.g.,* a 45° diagonal) represents a failed test (random performance). Various methods of estimating the area under the curves have been suggested, including using a maximum likelihood estimator to fit the data points to a smooth curve, using Simpson's rule, and fitting trapezoids under the curve. There are, however, more effective ways of assessing the overall accuracy of an algorithm, as we shall see.

Error considerations can be indicated on these plots. For example, if a single test is run on many different sets of images, then the mean false-positive rate can be plotted against the mean true-positive rate. The assessed confidence limits can then be plotted as error bars or error ellipses around the points.

Some researchers refer to the equal error rate (EER) of a particular test. The EER is the point at which the false positive rate is equal to the false negative rate. This may be of use in applications where the cost of each type of error is equal. The smaller the EER, the better.

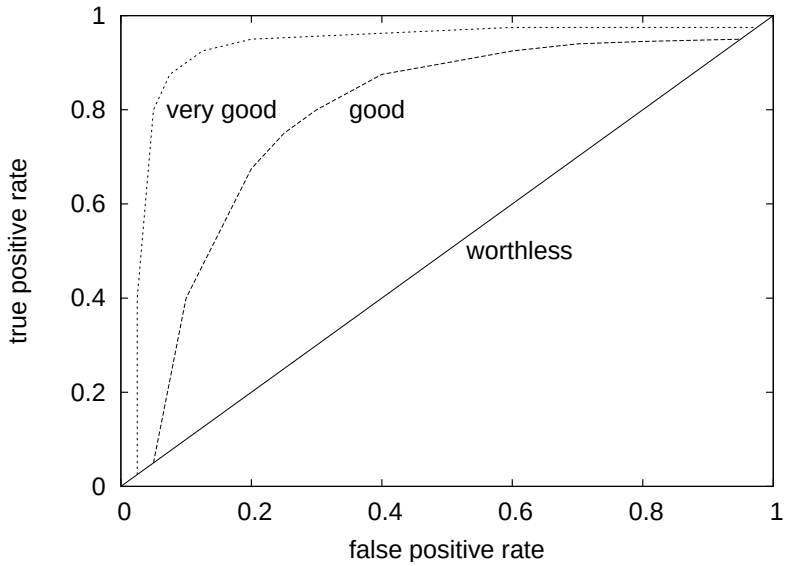

Figure 2: Examples of ROC Curves

4.2 Precision–Recall and Related Curves

If we write *TP* as the number of true positives *etc.*, then a number of quantities, mostly originating from the information retrieval field, can be derived from them. Those in most widespread use are:

$$\text{sensitivity} = \frac{TP}{TP + FN} \tag{3}$$

$$\text{specificity} = \frac{TN}{TN + FP} \tag{4}$$

$$\text{precision} = \frac{TP}{TP + FP} \tag{5}$$

$$\text{recall} = \frac{TP}{TP + FN} \tag{6}$$

$$F = 2\,\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{7}$$

Precision–recall curves are normally drawn with precision on the ordinate (y) axis, and they generally run from upper left to lower right. They are increasingly replacing ROC curves in vision papers.

We have seen that ROC and precision–recall curves are useful in assessing how different parameters applied to an algorithm affect performance. The following section describes how algorithms can be compared.

4.3 Confusion Matrices

A confusion matrix [8] contains information on the actual and predicted classifications performed by a system. For example, for a digit-recognition task, a confusion matrix like that in

Table 1 might arise.

actual	predicted									
	0	1	2	3	4	5	6	7	8	9
0	20	0	0	6	0	0	1	0	10	0
1	0	25	0	0	0	0	0	6	0	0
2	0	0	31	0	0	0	0	0	0	0
3	0	0	0	21	0	0	0	0	10	0
4	0	0	0	0	31	0	0	0	0	0
5	0	0	0	0	0	22	0	0	9	0
6	1	0	0	1	0	2	23	0	3	1
7	0	8	0	0	0	0	0	23	0	0
8	4	0	1	3	2	1	3	0	13	4
9	0	0	0	2	0	0	0	3	1	27

Table 1: Confusion Matrix for a Digit Recognition Task

Numbers along the leading diagonal of the table represent digits that have been classified correctly, while off-diagonal values show the number of mis-classifications. Hence, small numbers along the leading diagonal show cases in which classification performance has been poor, as with '8' in the table. Here, the actual digit '0' has been mis-classified as '8' ten times and as '6' once, while the digit '1' mis-classified as '7' six times. Conversely, the digit '2' has never been mis-classified. (Take care when you encounter this in the literature: it is often the transpose of this table.) There is no reason, of course, why the matrix should be symmetric.

In the particular case that there are two classes, success and failure, the confusion matrix just reports the number of true positives, *etc.* as shown below.

	predicted negative	predicted positive
actual negative	TN	FP
actual positive	FN	TP

5 Comparing Algorithms

5.1 Using ROC or Precision–Recall Curves

The most common way that algorithms are compared in the literature is by means of their ROC or precision–recall curves. This is acceptable to some extent; but the problem is that researchers hardly ever indicate the accuracy of the points in the curve using error bars or equivalent, and hence one cannot tell whether any differences in performance are significant.

ROC curves tend not to be as straightforward as those shown in Figure 2. Often the curves to be compared cross each other, and then it is up to the user to decide which curve represents the best method for their application. For example, Figure 3, shows that `alg1` may be superior to `alg2` when a high true-positive rate is required but `alg2` may be preferred when a low false-positive rate is required.

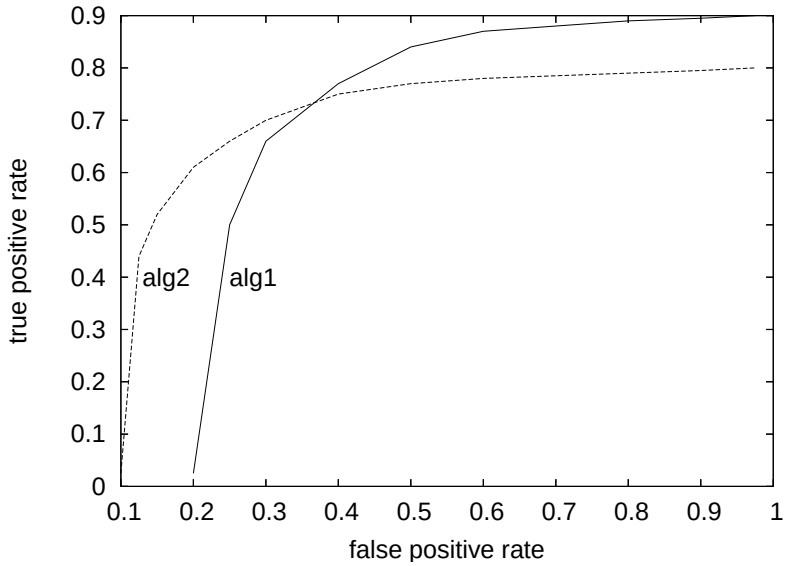

Figure 3: Crossing ROC Curves

As the accuracy of vision algorithms tends to be highly data-dependent, comparisons of curves obtained using different data sets should be treated with suspicion. Hence, the only viable way to compare algorithms is to run them on the same data. In principle, one could generate ROC or precision–recall curves for any number of algorithms, plot them with error bars, and perform visual comparisons. Even in this case, however, it is usually difficult to be sure whether one algorithm out-performs another significantly.

Hence, comparisons of algorithms tend to be performed with a specific set of tuning parameter values. (Running them with settings that correspond to the equal error rate is probably the most sensible.) When this is done, perhaps under the control of a test harness, an appropriate statistical test can be employed. This must take into account not only the number of false positives *etc.* but also the number of tests: if one algorithm obtains 50 more false positives than another in 100,000 tests, the difference is not likely to be significant; but the same difference in 100 tests almost certainly is (remember the statistical preamble in Section 2).

5.2 M^{c}Nemar's Test

The appropriate test to employ for this type of comparison is M^{c}Nemar's test. This is a form of chi-square test for matched paired data. Consider the following 2×2 table of results for two algorithms:

	Algorithm A Failed	Algorithm A Succeeded
Algorithm B Failed	N_{ff}	N_{sf}
Algorithm B Succeeded	N_{fs}	N_{ss}

Z value	Degree of confidence Two-tailed prediction	Degree of confidence One-tailed prediction
1.645	90%	95%
1.960	95%	97.5%
2.326	98%	99%
2.576	99%	99.5%

Table 2: Converting Z Scores onto Confidence Limits

M$^{\text{c}}$Nemar's test is:

$$\chi^2 = \frac{(|N_{sf} - N_{fs}| - 1)^2}{(N_{sf} + N_{fs})} \tag{8}$$

where the -1 is a continuity correction. We see that M$^{\text{c}}$Nemar's test employs both false positives and false negatives, rather than just one of them.

If $N_{sf}+N_{fs}$ (*i.e.,* the number of tests where the algorithms differ) is greater than about 20, then the value of χ^2 will be meaningful. We calculate the Z score (standard score), obtained from (8) as:

$$Z = \frac{(|N_{sf} - N_{fs}| - 1)}{\sqrt{N_{sf} + N_{fs}}} \tag{9}$$

If Algorithm A and Algorithm B give very similar results then Z will be near zero. As their results diverge, Z will increase. Confidence limits can be associated with the Z value as shown in Table 2; it is normal to look for $Z > 1.96$, which means that the results from the algorithms would be expected to differ by chance only one time in 20.[2] Values for two-tailed and one-tailed predictions are shown in the table as either may be needed, depending on the hypothesis used: if we assessing whether two algorithms differ, a two-tailed test should be used; but if we are determining whether one algorithm is better than another, a one-tailed test is needed.

Further information can also be gleaned from N_{sf} and N_{fs}: if these values are both large, then we have found places where Algorithm A succeeded while Algorithm B failed and *vice versa*. This is valuable to know, as we can devise a new algorithm that uses both in parallel and takes the value of Algorithm B where Algorithm A fails, and *vice versa* — this should yield an overall improvement in accuracy. This is actually a significant statement with regard to the design of vision systems: rather than combining the results from algorithms in the rather *ad hoc* manner that usually takes place, M$^{\text{c}}$Nemar's test provides a principled approach that tells us not only *how to do it* but also *when it is appropriate to do so* on the basis of technology evaluation — in other words, technology evaluation needs to be an inherent part of the algorithm design process.

5.3 Comparing Several Algorithms

Some caution is required when comparing more than two algorithms. M$^{\text{c}}$Nemar's test works on pairs of algorithms, so one ends up making a number of pairwise comparisons. We have already noted that it is possible for a comparison to yield a significant result purely by chance, so when we perform several such comparisons, the probability of this happening increases.

[2] This corresponds to two standard deviations from the mean of a Gaussian distribution.

More precisely, if the probability of a result arising by chance in any single comparison is α, then the probability of not making such an error is $1 - \alpha$. If two tests are independent, the probability of observing these two events together is the product of the individual events, so the probability of not making an error in N comparisons is $(1 - \alpha)^N$. If we take $\alpha = 0.05$ (the "one in twenty" rate mentioned earlier) with $N = 10$ algorithms, we obtain $(1 - 0.05)^{10} = 0.599$ — which means that the probability of making an error is about 0.4, definitely significant.

If we want the result of our entire comparison process to be have a confidence level of $\beta = 0.05$, then we must reduce the confidence level on each individual comparison by a factor that depends on N, the number of tests. Now

$$\beta = 1 - (1 - \alpha)^N \tag{10}$$

so we must set

$$\alpha = 1 = (1 - \beta)^{\frac{1}{N}} \tag{11}$$

This is known as a Šidàk correction. Because it involves a fractional power, it used to be difficult to calculate so a number of authors came up with the approximation

$$\alpha = \frac{\beta}{N} \tag{12}$$

which is most commonly called a Bonferroni correction, though both Boole and Dunn arrived at it independently. Technically, this is a first-order Taylor expansion of the Šidàk correction. Comparing the two, the Bonferroni correction is always more conservative than the (more correct) Šidàk one, so the latter is to be preferred.

The consequence of this is that, as you include more algorithms in a comparison, you must be more stringent when deciding that a result is significant. The authors have yet to see this done in a vision paper (indeed, they have yet to see the use of M^{c}Nemar's test), though they are aware of people in industry who have used it when evaluating vision systems.

6 Determining Robustness to Noise

It was mentioned in Section 1 that one of the problems with vision algorithms is their fragility to unexpected variations in the input. One aspect of variation — by no means the only one — is noise. As we know, light consists of photons; if one goes through the maths, the inter-arrival time of photons in conventional cameras from incoherent light sources (*e.g.,* sunlight or artificial light sources but not laser, ultrasound or radar) is Poisson-distributed and, for reasonable numbers of photons, this can be approximated as Gaussian. (The way in which the photons are captured and processed by camera systems affect the distribution, so Gaussian is not always a good assumption even though everyone appears to use it in practice.)

To determine how robust an algorithm is to noise, one simply takes a typical input image and generates (say) 100 versions of that image with added Gaussian-distributed noise of zero mean and known standard deviation. The images are then fed to the algorithm under test, one by one. If the algorithm produces a class label, that label should not change; and if the algorithm produces a measurement (*e.g.,* the location of a corner), plotting a histogram of the error in that measurement will show how the algorithm is affected by noise. If the input noise distribution is Gaussian, the distribution of the measurement error should also be Gaussian — if it is not, there is a problem with the algorithm — and the standard deviation of the errors in the output gives an idea of how well the algorithm withstands noise.

7 So Is This The Whole Story?

No. No matter how well something is tested in the research lab, even on real-world images captured in a variety of conditions, the moment you try out your system 'in the wild,' there will be a host of problems you hadn't thought of — shadows, changing illumination, uneven ground, people getting in the way, no handy mains power and a million other things. Most of these will affect the performance of a vision system. The most effective systems at the moment work in constrained environments, where for example the lighting is static and controllable, but increasingly researchers are trying to make their systems work with fewer constraints. This is laudable and will lead to vision systems that are much more reliable, though it is a long, slow process. I encourage you to become involved in this.

References

[1] M. Turk and A. Pentland. Eigen faces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[2] Neil A. Thacker, Adrian F. Clark, John L. Barron, J. Ross Beveridge, Patrick Courtney, William R. Crum, Visvanathan Ramesh, and Christine Clark. Performance characterization in computer vision: A guide to best practices. *Computer Vision and Image Understanding*, 109(3):305–334, March 2008.

[3] S. J. Maybank. Probabilistic analysis of the application of the cross ratio to model-based vision. *International. Journal of Computer Vision*, 16:5–33, 1995.

[4] P. J. Phillips, H. Wechsler, J. S. Huang, and P. J. Rauss. The FERET database and evaluation procedure for face-recognition algorithms. *Image and Vision Computing*, 16(5):295–306, 1998.

[5] N. A. Thacker, A. J. Lacey, and P. Courtney. An empirical design methology for the construction of computer vision systems. Technical report, Department of Imaging Science and Biomedical Engineering, Medical School, University of Manchester, UK, May 2003. `http://peipa.essex.ac.uk/benchmark/methodology/white-paper/methodology.pdf`.

[6] J. F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, November 1986.

[7] Paul R. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, 1995.

[8] Ron Kohavi and Foster J. Provost. Applications of data mining to electronic commerce. *Data Mining and Knowledge Discovery*, 5(1/2):5–10, 2001.