Contents

1 Fundamentals 1
2 Convolution 1
3 Low-level vision 2
4 Intermediate-level vision 2
5 Humans 3
6 Stereo 4
7 Machine learning 4
8 Evaluating vision systems 6

1 Fundamentals

An image is a representation the projection of the world around us onto a 2D plane, typically captured by a camera. Some 'cameras' can capture in wavebands other than the visible (400–700 nm), such as the infra-red or X-ray, or even measure information such as depth (as in the Kinect). A single-channel image is stored as follows with pixel values normally as unsigned bytes.

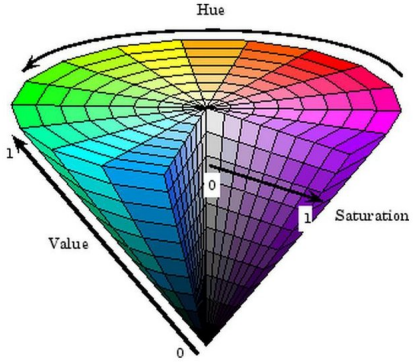
↓ y increasing	x increasing → 0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	255	0	0	0	0
2	0	0	0	255	255	255	0	0	0
3	0	0	255	255	255	255	255	0	0
4	0	0	0	0	255	0	0	0	0
5	0	0	0	0	255	0	0	0	0
6	0	0	0	0	255	0	0	0	0
7	0	0	0	0	255	0	0	0	0
8	0	0	0	0	255	0	0	0	0
9	0	0	0	0	0	0	0	0	0

► Colour images contain red, green and blue values for every pixel, so are indexed by row, column and colour channel: `im[y,x,c]`. OpenCV stores colour images as blue, green and red using a representation compatible with `numpy`:

```
>>> print (len (mono_im.shape))
2
>>> print (len (colour_im.shape))
3
```

It is common to process colour images in HSV space:

```
hsvim = cv2.cvtColor (im, cv2.COLOR_BGR2HSV)
```



► A single sum for calculating the mean

$$\bar{P} = \frac{1}{N}\sum_{i=1}^{N} P_i$$

usually corresponds to loops over rows, columns and channels:

```
def mean (im):
    "Return the mean of the image im."
    ny, nx, nc = im.shape
    total = 0
    for y in range (0, ny):
        for x in range (0, nx):
            for c in range (0, nc):
                total += im[y,x,c]
    return total / ny / nx / nc
```

The standard deviation of an image is

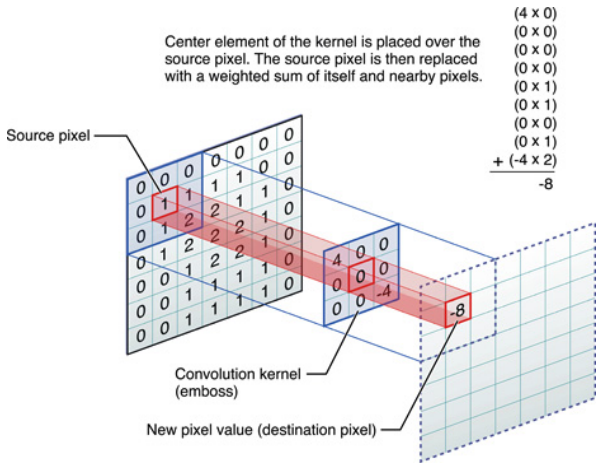
$$\sigma^2 = \frac{1}{N}\sum_i \left(P_i - \bar{P}\right)^2$$
$$= \frac{1}{N}\left(\sum P_i^2 - \frac{1}{N}\left(\sum P_i\right)^2\right)$$

► Images using only a small range of grey levels can be contrast-stretched by scaling the lowest value P_l to V_l and the highest P_h to V_h, so each pixel P_i becomes

$$(V_h - V_l)\frac{P_i - P_l}{P_h - P_l} + V_l$$

Normally $V_l = 0$ and $V_h = 255$.

2 Convolution



Note that you cannot compute a convolution in place. Common masks include:

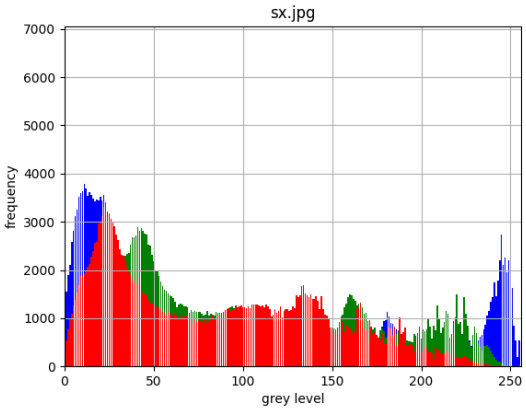
```
                1 1 1 1 1
  1 1 1         1 1 1 1 1          1  1  1
  1 1 1         1 1 1 1 1          1 -8  1
  1 1 1         1 1 1 1 1          1  1  1
                1 1 1 1 1
 3x3 blur       5x5 blur        Laplacian
```

► Rather than adding or taking the mean at each pixel, one can compute the *median*, the *minimum* or *maximum*. When used on binarized images, the latter two are the basis of *morphology* (*shrink* and *expand*), with other operations based on them such as including *open* (shrink then expand to remove small objects) and *close* (expand then shrink to fill in small gaps)

► *Matched filtering* involves designing the vales of the mask to emphasize particular features in an image.

3 Low-level vision

This typically starts with a histogram:



Histograms are most useful for helping decide where to set a threshold τ to segment images into foreground and background — though Otsu's method can set τ automatically when the numbers of foreground and background pixels are roughly the same. The OpenCV calls for ordinary and Otsu thresholding are

```
t, bim = cv2.threshold (im, thresh, 255,
        cv2.THRESH_BINARY)
t, bim = cv2.threshold (im, thresh, 255,
        cv2.THRESH_BINARY+cv2.THRESH_OTSU)
```

► With binarized images from thresholding, it is common to label regions, so that:

0	0	0	0	0	0	0	0	0	0
0	1	1	0	1	1	0	0	1	1
1	1	1	0	0	1	0	0	1	1
0	0	0	0	0	1	1	0	0	0
1	1	0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0	0	0
0	1	1	1	0	0	1	1	1	1
0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0	0	0

goes to:

0	0	0	0	0	0	0	0	0	0
0	1	1	0	2	2	0	0	3	3
1	1	1	0	0	2	0	0	3	3
0	0	0	0	0	2	2	0	0	0
4	4	0	0	0	0	2	2	0	0
0	0	0	0	0	0	0	0	0	0
0	5	0	5	0	6	0	0	0	0
0	5	5	5	0	0	7	7	7	7
0	0	0	0	0	0	0	7	7	0
0	0	0	0	0	0	0	0	0	0

In OpenCV, this is done when finding contours:

```
cv2.findContours (bim, cv2.RETR_EXTERNAL,
                  cv2.CHAIN_APPROX_SIMPLE)
```

► Segmenting by thresholding or colour often yield spurious regions or small gaps within regions. These can often be deleted on the basis of size or filled in with morphological operations.

► Having identified and labelled regions (or found contours in OpenCV), one way to identify or distinguish them is by region properties: perimeter, area, moments, convexity, bounding box, aspect ratio, rectangularity, circularity, number of holes, and so on.

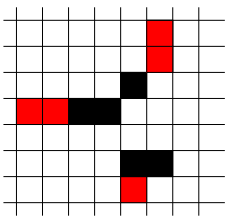
4 Intermediate-level vision

Canny's edge detector is based around:

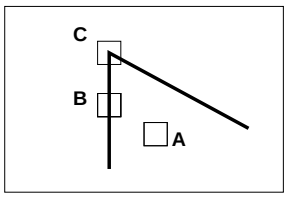
- it should respond only to edges
- all edges should be found
- edges should be in the correct places
- multiple edges should not be found where only one exists

Its stages are:

1. Convolve the image with a Gaussian mask
2. Find differences in horizontal and vertical directions
3. Find the magnitudes and directions of the edge gradients
4. Perform non-maximum suppression to thin edges
5. Perform hysteresis thresholding to join edge segments.



- All edge detectors suffer from the *aperture problem*: you cannot tell where you are on a line that fills the field of view. This is not the case for corners.
- Moravec's Corner detection is based around the idea that a region around a corner differs significantly from regions displaced from it in any direction.

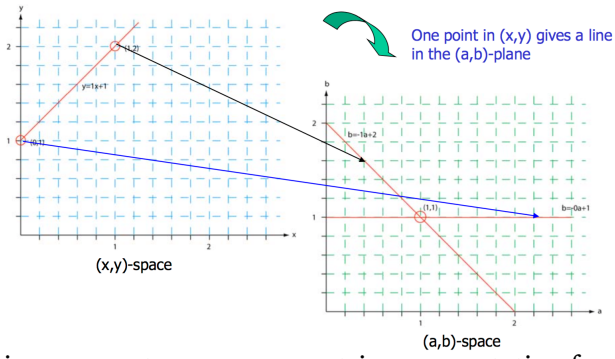


We compute for pixel values $P(x,y)$ and mask values $W(u,v)$:

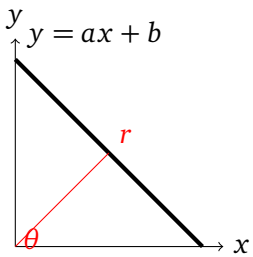
$$E(x,y) = \sum_{u,v} W(u,v)\,(P(x+u,y+v) - P(u,v))^2\,.$$

and consider shifts of $(1,0)$, $(1,1)$, $(0,1)$ and $(-1,1)$, and look for local *maxima* in $\min(E)$ above some threshold value to identify corners.

- Harris & Stephens improve on Moravec by considering the direction of the corner. Other well-known corner detectors include FAST, based around the idea that at a corner more than half the pixels will be dark or light.
- The line Hough transform fits a straight line to pixels that may lie on an edge using an 'accumulator' indexed by gradient and intercept. Each point produces a line in the accumulator



It is common to use a parametric representation for lines instead of $y = ax + b$.



Now $x\cos\theta + y\sin\theta = r$ so we have to draw sine waves in the accumulator array instead of straight lines. We look for peaks in it to identify lines. There are versions of the Hough transform for circles and other shapes.

- To describe the region around a corner, one can use BRIEF. For a patch of size $S \times S$ encompassing a corner in the image P, some N locations (x_i, y_i) in the patch are chosen and a series of bitwise 'tests' τ are made for different combinations of i and j:

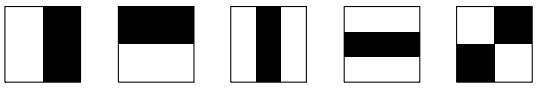
$$\tau = \begin{cases} 1 & \text{if } P(x_i,y_i) < P(x_j,y_j) \\ 0 & \text{otherwise} \end{cases}$$

The resulting binary numbers are combined into a string of bits, the *BRIEF descriptor*. Comparing these descriptors is done using the Hamming distance.

- BRIEF is affected badly by scale or rotation changes, so people use closely-related ORB, which positions the corner in the centre of the patch and computes the centroid. The direction of the line from this corner to the centroid gives the orientation.
- SIFT is arguably the best feature matcher, working when there are scale and orientation changes. To find matches between (say) objects in successive frames of a video, first calculate the SIFT features for each frame. Then, for each feature of the first frame in turn, compare it with *all* features in the second frame and choose the one with the lowest score as being the best match. SIFT is quite slow, though GPU versions of it are now available. The features tend to lie within regions, not at (say) corners.

5 Humans

The best-known way to find faces is via the Viola-Jones algorithm, based around Haar features because they can be computed quickly (see below).



To train, images were reduced to 24×24 pixels and all 162,336 possible features computed. A learning algorithm, Adaboost, finds which combination of features minimizes the number of mis-classifications of face and non-face regions. Each feature individually may not perform well but the weak classifiers combine to produce a strong classifier. Their face detection system has 38 stages, the first five of which involve only 1, 10, 25, 25 and 50 features; the total number of features required is about 6,000.

- Haar features can be calculated in constant time irrespective of their size by using *integral images*, where each pixel contains the sum of all image pixels above and to the left of it.
- In OpenCV, we load a "cascade" of Haar features and apply it to the image to determine all the faces, which we can then iterate over.

```
casc = c2.CascadeClassifier ("...")
faces = casc.detectMultiScale (im,
  scaleFactor=1.1, minNeighbors=5)
for (x, y, w, h) in faces:
  cv2.rectangle (im, (x,y), (x+w,y+h),
    (255,0,0), 2)
```

There are cascades for eyes, mouths and other types of image feature.

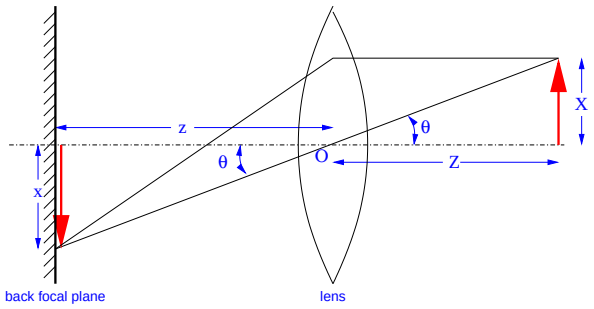
- ► Having found a face, it can be *normalized* so that (say) the eyes are always located in the same place on an image.
- ► Face recognition can be done using any of ML algorithm described in section 7.
- ► HOG, the histogram of oriented gradients, describes local regions by partitioning the image into cells and, for each one, computing a histogram of gradient directions using the masks

$$\begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

The outputs are used to calculate edge direction as for Canny, and the resulting angle is used to increment one of about 10 bins, usually by the corresponding gradient magnitude. HOG has proven to be useful for detecting pedestrians and people running.

6 Stereo

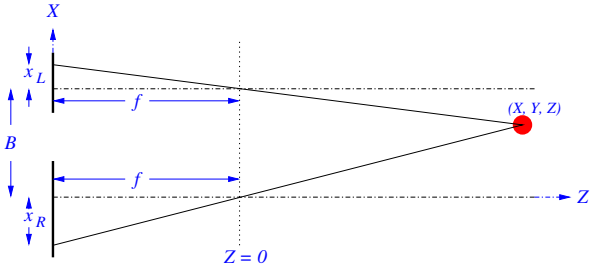
A camera works roughly as follows:



so that

$$\tan\frac{x}{z} = \tan\frac{X}{Z}.$$

- ► To determine the distance Z to an object of unknown size, we use a pair of identical cameras arranged with their optical axes parallel.

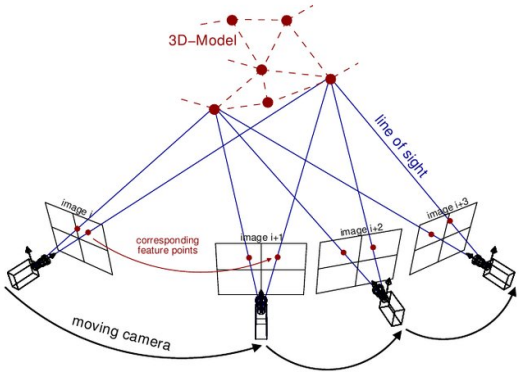


from which we can obtain the expression

$$Z = \frac{fB}{x_L - x_R}$$

where f is the focal length, B the separation of the optical axes and x_L and x_R the equivalent positions in the left and right cameras. All measurements have to be in the same units, of course.

- ► To propagate uncertainty when calculating distance (or anything else), the rules are:
  - when adding or subtracting values, add the uncertainties
  - when multiplying or dividing values, add the fractional uncertainties
- ► When many images are available, *structure from motion* can be used to build a complete 3D reconstruction. Doing this requires a good image feature matcher such as SIFT.



- ► Visual SLAM is based around the same principles as structure from motion but works in real time by replacing the slow, accurate SIFT with a video rate but less accurate matcher such as Harris&Stephens with ORB.

7 Machine learning

ML techniques fall into two broad categories:

Unsupervised: try to partition data into classes with no knowledge of the underlying problem: *e.g.*, k-means

Supervised: use training examples to 'learn' how a set of data is partitioned into classes; *e.g.*, support vector machine (SVM), multi-layer perceptron (MLP), random forest (RF), genetic programming (GP), convolutional neural network (CNN)

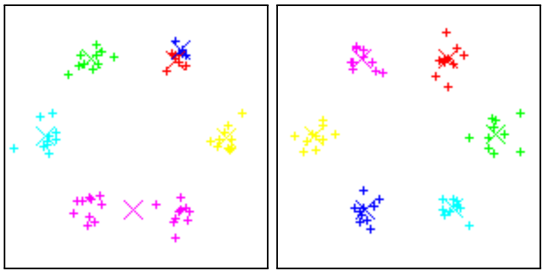
As supervised learners use more data, they usually perform better — though the cost of producing the 'ground truth' classifications can be high. Note that

one can use parameters extracted from imagery (a *feature vector*) rather then the images themselves as training data.

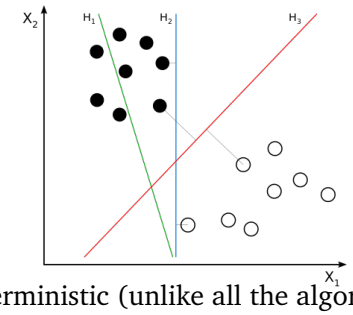
► k-means alternates between:

- assigning data points to clusters based on the current centroids; and
- choosing centroids (cluster centres) based on the current assignment of data points to clusters.

Note that the algorithm has to be told the number of clusters to look for. It doesn't always converge to the best result, so multiple runs can 'vote' for a good solution.



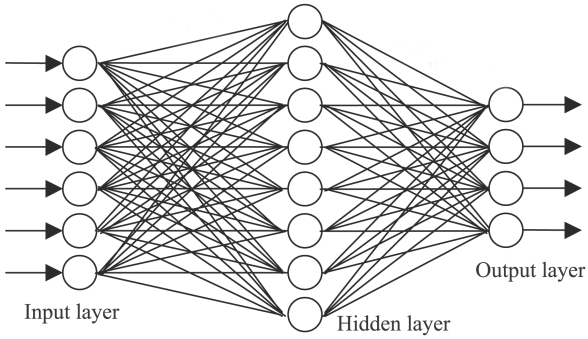
► SVM works out the best (maximum margin) hyperplane between classes. If there is not a linear (planar) separation between classes, it doesn't work, though scaling parts of the data before SVM and unscaling them afterwards sometimes helps.



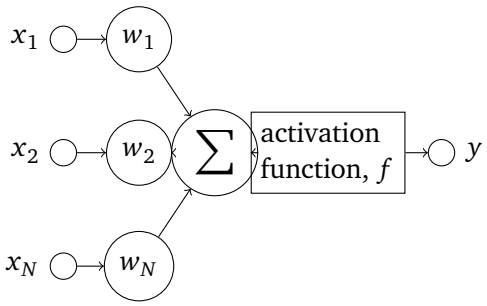
SVM is deterministic (unlike all the algorithms that follow here), so is pretty fast. Care needs to be taken with the parameter settings for it to work well.

► Genetic programming involves generating a random population of programs, evaluating how well they work on a problem using a *cost function*, then "mating" the better-performing ones in a way that mimics sexual reproduction to produce "children." This is repeated a number of times. For computer vision, one can define operators that perform useful tasks: convolution, thresholding, *etc*. Perhaps surprisingly, this can work very well on real-world problems.

► The MLP comprises a number of fully-connected layers of "neurons," each of which is simple. The data are presented at the *input layer*, which pass them to *hidden layers*, and they in turn pass them to a *output layer*. Recent years have seen computational performance increase to the point that a dozen or more hidden layers are not uncommon.



The connections between neurons each has a weight w_i which is determined during training.



$$y = f\left(\sum_{i=1}^{N} x_i w_i\right)$$

where x_i are the inputs to a neuron and w_i the corresponding connection strengths. Popular choices for $f(\cdot)$ are tanh, Gaussian, *etc*.

Training involves setting initial values for the weights and adapting them through a series of *epochs*, often using the *back-propagation* algorithm. If there are too few training examples, the network often experiences *over-training* in which it recognizes only the training patterns and not similar ones. Hence, it is common to perform *data augmentation*: shifting, rotation and adding noise to existing training examples. Training is quite slow but execution is fast.

► CNNs, specific to images, are based around learning custom convolution layers. They comprise a number of types of layer:

INPUT: receives the raw pixels of the data
CONV: computes the output of neurons that are connected to local regions of the input — in other words, it performs a convolution with coefficients that are learned from the data
RELU: applies an element-wise activation function, which may be as simple as `max(0,x)` to threshold at zero
POOL: down-samples or averages regions of its input, so that the overall width and height of the network is reduced
FC: the final layer is usually fully-connected, just as in an MLP; it computes the class scores and hence returns the calculated class of the pattern presented at the network's input

CONV and FC layers transform the data as a function of their inputs according to the weights and biases of the neurons, which means they must be trained. Conversely, RELU and POOL layers implement a fixed function and do not require training. CNNs can have tens or even hundreds of layers. Popular ones include LeNet, AlexNet, GoogLeNet, VGGNet and ResNet.

► CNNs can take literally weeks to train on commodity hardware, so it is increasingly common to use one pre-trained on (say) the huge ImageNet database and train only an additional FC layer. This is known as *transfer learning*.

► Visually identical inputs to a deep network can yield totally different classes as output, and with high confidence values. This has been exploited in *Generative Adversarial Networks* (GANs), in which two neural networks are pitted against each other. A *generator* network produces input to a *discriminator* one; both are trained concurrently, with the generator trying to produce examples that the discriminator cannot distinguish from real images, while the discriminator tries to spot the generator's fakes.

► Most ML techniques are "black boxes," meaning that a human cannot understand how they work in detail; this makes them less desirable for areas such as medicine and aircraft. Exceptions are SVM and GP, which *are* interpretable.

8 Evaluating vision systems

Evaluation requires "ground truth" imagery for which the answer is known, often assessed by experts in the problem domain. Data used for testing must be different from those used for training. Each individual test has four possible outcomes:

true positive (TP): the feature was found and given the correct label

false positive (FP): the feature was found but given an incorrect label

false negative (FN): a feature that should have been found was not

true negative (TN): a feature was not present and no feature was found

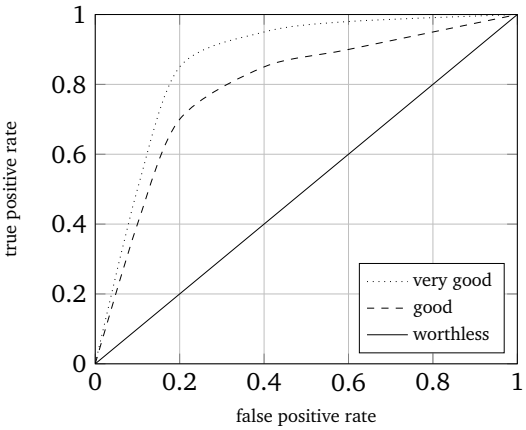
There is always a trade-off between true positive and false positive detections. We can derive some further measures from these:

$$\text{accuracy} = \frac{\text{TP}+\text{TN}}{N} \quad \text{precision} = \frac{\text{TP}}{\text{TP}+\text{FP}}$$

$$\text{sensitivity} = \text{recall} = \frac{\text{TP}}{\text{TP}+\text{FN}} \qquad \text{specificity} = \frac{\text{TN}}{\text{FP}+\text{TN}}$$

where N is the number of tests.

► If some parameter that governs how the system operates can be adjusted, it is possible to draw a ROC curve.



Other curves are often drawn, *e.g.* precision–recall.

► Some people use the area under the ROC curve to assess which of several algorithms is best. This is inappropriate for computer vision because one operates a vision system at one specific set of parameter settings; and it fails to take into account the number of tests. There is a statistical test, Mcnemar's test, which is better. One must run algorithms on the same data and record the outcome of each individual test to build up the table

	A failed	A succeeded
B failed	N_{ff}	N_{sf}
B succeeded	N_{fs}	N_{ss}

where N_{ss} is the number of cases where both algorithms succeeded, *etc*. Then one calculates the statistic

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

which is interpreted using the following table

Z value	two-tailed confidence	one-tailed confidence
1.646	90%	95.0%
1.960	95%	97.5%
2.326	98%	99.0%
2.576	99%	99.5%

For example, if $Z \geq 1.96$, the probability that the algorithms' performance could have differed (two-tailed test) due to an unfortunate set of data is 5% or one in twenty.