Appearance Based Object Recognition with a Large Dataset using Decision Trees

Philip Blackwell[1,2] philip.blackwell@anu.edu.au

David Austin[1,2] d.austin@computer.org

[1]Robotic Systems Lab, RSISE
Australian National University,
ACT 0200, Australia

[2]National ICT Australia,
Locked Bag 8001,
Canberra, ACT 2601, Australia

Abstract

A framework for a new object recognition system is presented that tailors the classic decision tree to the specific needs of object recognition. The key advantage of such a system is the ability to handle a large database of known objects with real-time performance, a failing of current object recognition systems. A new image database is presented that is challenging both in size (over 100,000 images), and content (highly similar objects). Early results, using the simplest features, are promising with a recognition rate of 89%.

1 Introduction

Visual object recognition is a huge field, for which a complete general solution is unattainable. Searching for a known object in a given scene, and identifying a given object are inherently different problems. The problem considered here is the latter. We assume that the given object is represented by one un-occluded, reasonably segmented view. This allows for easier selection of features. However, the general method of decision trees does not require these assumptions. The aim here is to build a system that is capable of scaling to a large number of known objects, with real-time performance.

There are many ways of comparing one object with another, much work has been done in this area. But no matter how simple and accurate the comparison is, there is no way of directly applying these methods to large object databases. Scaling with the number of recognisable objects is usually poor.

Support Vector Machines (SVMs) have been used successfully in various problems of pattern recognition. They provide a simple data driven approach to classification, finding optimal splitting hyperplanes between sets of feature vectors. They solve a two class problem (i.e. is a face, is not a face), and as such have been successfully used for object detection [Osuna *et al.*, 1997]. There are ways to consider multi-class problems as extensions of two class problems in [Pontil and Verri, 1998], splitting hyperplanes are defined between each pair of objects, and classification is carried out in the form of a tennis tournament. This achieves good results, but doesn't allow for a large number of objects, since the number of pairs grows as the square of the number of objects. An alternative method is presented in [Cortes and Vapnik, 1995], where splitting hyperplanes are generated one against k, resulting in linear order, but results are less clear, in general, the tennis tournament is preferred method.

Aspect graphs [Faugeras *et al.*, 1992] provide a good way of matching an object against a set of characteristic views. Although the majority of the work in the area is focusing on limited datasets since indexing is otherwise a problem, there have been attempts to use such methods on reasonably large databases. In [Cyr and Kimia, 2004] shock graphs are used as in improved method of indexing, a database of 64 objects is used, and good recognition results are achieved, but recognition still takes up to 45 minutes.

Good results were achieved for a large database (100 objects) in [Murase and Nayar, 1995]. A universal Eigenspace is calculated based on all learned images, in which each object is represented as a manifold (for all its different views). New images are projected into the Eigenspace, then classified as the object of the nearest manifold. The main difficulty is in the calculation of the Eigenspace, although it is an offline process that can reasonably be allowed significant time, it scales poorly both with the number of objects and number of views. This will be a limiting factor to the size of the database.

Ultimately, to scale efficiently, recognition must be at a higher level than simply matching against known objects. The general approach is to take some high level features that describe the object, and index them in some way. This is known as the indexing problem. Indexing will, in general, be highly sensitive to small changes in the features, this means that the features must be highly

robust to noise, rotation, scale, etc. which is very difficult to achieve for any meaningful dataset.

A decision tree is used to achieve fast, scalable recognition. A decision tree is a very simple and intuitive way of splitting data, a "divide and conquer" approach. Although widely used in the machine learning and data mining communities, decision trees haven't been used much for object recognition. The main reason for this is that the data tends to be split in one dimension at a time, usually thought to be too simple for a task as complicated as object recognition. However, there is no real reason that more sophisticated splits can't be used. Decision trees have two distinct advantages for object recognition: they naturally handle large databases, and they allow for a rich choice of features, since not all features need to be computed for classification.

Section 2 gives a brief overview of decision trees. Section 3 describes the object database used. Section 4 gives an overview of some initial experiments. Section 5 discuses some of the features that are planned to be incorporated into the decision tree.

2 Decision Trees

The concept of a decision tree is very simple and intuitive, and a simple example is a good starting point for the unfamiliar reader (Figure 1). Playing out very much like a game of "twenty questions". Where a decision tree in the interrogator's head is traversed according to the answers to questions posed at each node.

Decision trees are a classic machine learning and data mining technique, most work stemming from CART [Breiman *et al.*, 1984], and ID3, C4.5 and C5 developed by Quinlan [Quinlan, 1990; 1992]. There are two distinct processes that need to be implemented; building the tree, and then making classifications based upon it. Before describing these processes, some terminology is described.

- *Examples* are the things being classified, they are *labelled* to identify the *class* they belong to.
- The *dataset*, is the collection of examples.
- *Features* are measures that can be derived from examples, e.g. Circularity, could be a measure of how much an example is like a circle.
- *Questions* define boundaries in feature-space, for example if a feature took real numbered values a question might be that some feature of an example is less than 0.5.
- *Nodes* make up the *tree*, they contain examples, and if they are not terminal, contain a question the answers to which are represented as further nodes.

The tree is built by successively finding questions to split the dataset, in meaningful ways, until it can be split no more (either because the features of all examples are equal, or all examples belong to the same class). The building is top down, at each point information theory heuristics are used to decide at any point what is likely to be the "best" split.

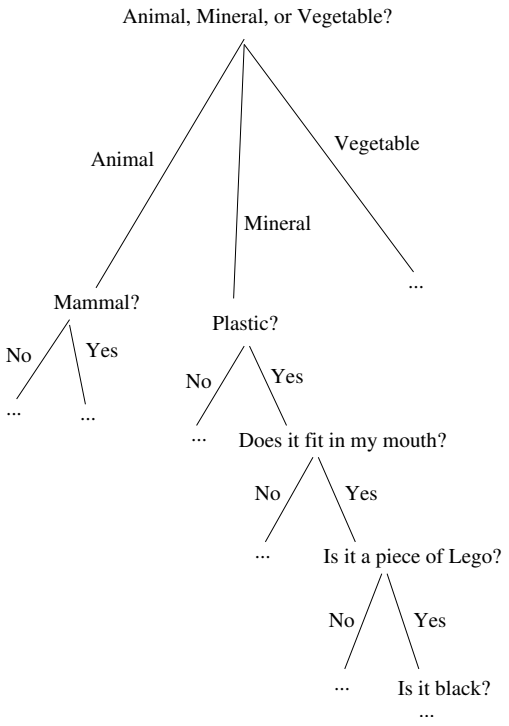

Figure 1: A small part of an example decision tree.

To classify an unseen example the tree is traversed depending on the answers to the question at each node, when a leaf node is reached the classification is made based on the learned examples in that node.

Good classification (both in terms of accuracy and speed) depends entirely on the tree. To achieve fast, accurate recognition, the tradeoff is a complicated, possibly slow training process. Ideally, there is no real limit to what questions to ask at any node, but there must be with time constraints. However, the aim here is to push the boundaries of what questions can be asked.

The key difficulty with a decision tree is that at each node there are many possible good ways to split the data, there is no way of knowing which split to aim for. Where as with, for example, an SVM finds the best way to differentiate between two given sets of examples.

3 Object Database

A common problem in computer vision is having overly simple test data. In the case of object recognition, the database of known objects can be trivially small, or distinct. Often, object recognition systems are only tested on a handful of objects, where scalability cannot be

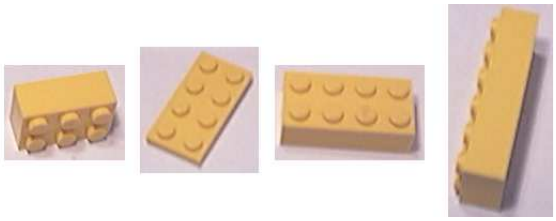

Figure 2: Four yellow bricks.

Figure 3: Four grey gears (scaled to similar sizes).

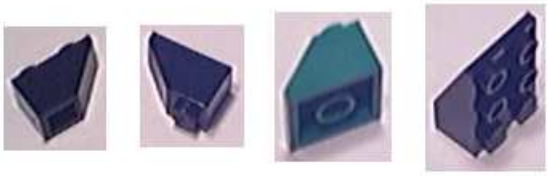

Figure 4: Four wedge pieces.

tested, and recognition rates can be biased.

There aren't many large test image databases, and of the available databases, none really allow for proper testing of the aims of this project. Probably the most widely used is COIL-100 [Nene *et al.*, 1996], having 100 objects with 72 views of each object, but it has some limitations. Firstly, the 72 views are quite limited, simply rotating the object around its central axis. Secondly, the objects are highly distinct. There are several simulated databases, that is databases of CAD and VRML models. These are good, because arbitrarily many views can be generated, but the absence of noise is hard to properly account for.

So a new database has been generated that has many objects from a similar class (all Lego), many views of each object (in random positions), and real noise. An important aspect is that the training and testing samples are chosen, randomly, from one database of typical views of the object.

The images are taken with a 640x480 webcam, with all settings on automatic. The resulting images, once cropped, are approximately 20-100 pixels in width and height.

Reasonable segmentation is made possible by having the objects on an otherwise empty, white platform. The segmentation is still automated, and of course, not perfect, shadows usually remain, to differing extents, specular reflections are often removed. Occlusion is avoided by only selecting images where the object is completely surrounded by the background. Lighting is controlled up to a point; there are halogen lamps directly above the platform in a fixed position, always on, but the platform is next to a window, in a well lit room.

Some example images (Figures 2, 3 and 4) of different types of bricks in the database.

3.1 Acquisition of images

To give the database the desired features, the method for acquiring images was somewhat unconventional. The camera was in a fixed position, and each Lego brick was repeatedly dropped from a height of about 15cm to fall on a platform in a random pose. Obviously, the machine had to be made out of Lego itself (Figure 5).

3.2 Size

The database is large both in number of classes (89), and number of examples per class (at least 1000), for a current total of 100,000 images.

This allows for thorough testing of performance (particularly scalability) both in terms of preprocessing/learning, and recognition.

Also, the difficulty of recognition increases with database size. If there aren't enough objects in the database, there will be trivial splits in feature space.

3.3 Range of Views

Given the subject matter, the method used here to acquire the images restricts the range of views - it is extremely unlikely that a block will land on anything other than one of its sides, and since the camera is in a fixed position it will be impossible to get a directly overhead view of a block. However the limitation is nothing like that of most databases, where views are limited to a fixed set of, usually regularly separated around a viewing sphere.

The database is representative of the views of the object that are likely to be seen. For example consider images of a block sitting on its base, and sitting on its end (Figure 6). The former will presumably be represented many more times in the database than the latter, and to good effect: the recognition system will be able to take into account that it is more important to be good at recognising the more common poses.

Of course, this doesn't mean that the database accurately represents the likely poses of Lego blocks "in the wild", but the results obtained using a database with some form of real world bias will be more representative of the results to expect when using real world data.

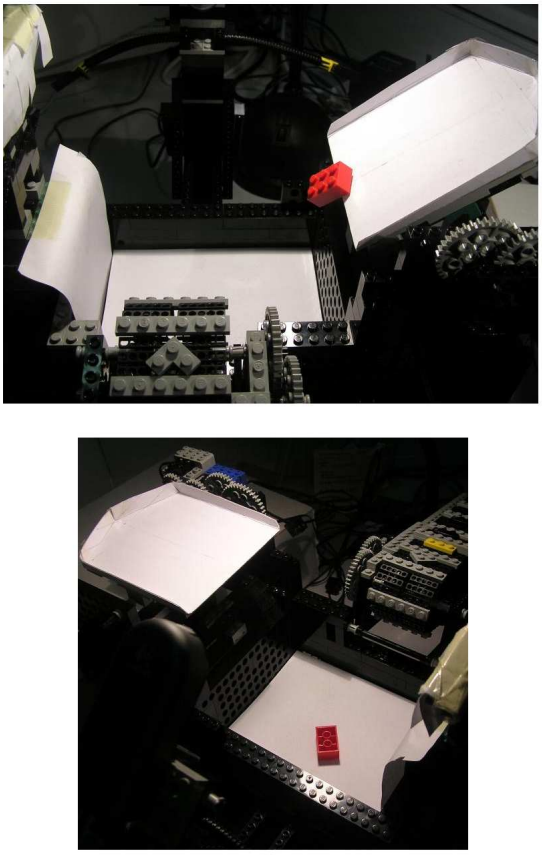

Figure 5: First dropping a piece, then taking a picture, then brick is pushed off the (raised) platform so the process can repeat.

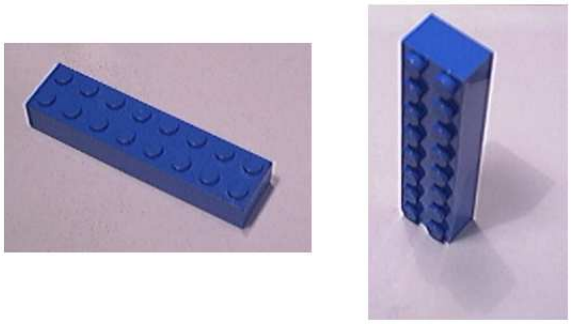

Figure 6: Two poses of an 8x2 block.

3.4 Lego Characteristics

The Lego bricks in general are quite simple objects, in some ways this makes the task of recognition quite simple, but in others quite difficult. The colour of the bricks is very distinctive; a yellow object in this database is quite obviously yellow. On the other hand, the simplicity of the objects can make them difficult to differentiate; the difference between a six units long axle, and a five units long axle is in many ways quite small.

Another point is that everybody can get Lego. If people want to conduct further experiments to evaluate their recognition systems when trained with this database, they will have easy access to identical objects.

4 Preliminary Results

4.1 Approach

The approach taken is, at a high level, quite conventional. Each object in the database is given a list of values for various features, and then, using a subset of the dataset a decision tree is built, based on those features. The tree is then tested using the remaining data.

The typical approach is to define an appropriate set of features for the dataset. Although applying this general technique to a particular dataset can be a challenging and worthwhile problem, that is not the aim here. Rather, the aim is to show that using simple features a decision tree system is capable of achieving good recognition rates on a challenging, large datasets with real-time performance.

4.2 Features

Since the aim isn't to build a system that can recognise Lego bricks for the sake of recognising Lego bricks, the emphasis is not on developing new features that are descriptive for Lego bricks. The features used are fairly simple, and mostly well known. Terminology used is mainly taken from [Csetverikov, 2003].

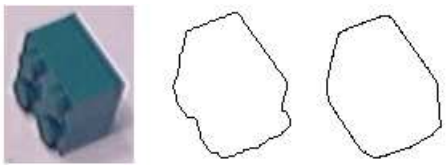

Figure 7: The contour, and convex hull of an image.

Images are assumed to be one "blob" of 8-connected pixels. Most of the features are shape based using features derived from the outer contour and convex hull (Figure 7).

Colour Due to the simplicity of the colour in the database only the simplest colour features were used. Three features (Red, Green, and Blue) were derived from the average values of the respective image channels, normalised by dividing by the average intensity.

Elongation The ratio of the the major axis length to the minor axis length is called Elongation. The axis lengths were taken as the width and height of the rectangle fit around the convex hull, such that the Elongation was maximised.

Circularity Another commonly used shape based feature is the ratio of the area to the perimeter squared, specifically $\frac{4\pi A}{p^2}$. Two measures of this direct feature were taken: ContourCircularity, and ConvexHullCircularity using the measurements for area, and perimeter from the contour and convex hull respectively. A third, similar measure, Circularity, was taken incorporating the diameter (from the convex hull), and the area (the number of pixels) $\frac{A}{\pi d^2}$. No argument is made as to which of these features is a more appropriate measure of circularity - it doesn't really matter. It is up to the decision tree to decide which feature will generate a more informative split at any given node.

Ratios of Areas The ratio of the area of the convex hull to the area inside the contour is a feature called Roughness. The ratio of the total area to the area of the contour is a feature called Filled.

Relaxed Convex Hull One measure used, that wasn't found in the literature attempts to measure how much the convex hull is like an N sided polygon for any given N. The convex hull is repeatedly relaxed[1] (Figure 8), from its original number of vertices to three (if possible) . The measure RelaxedRatio(N) was taken for $N \in [3, 8]$ as the ratio of the area of the convex hull to the area of the relaxed N-sided convex hull.

[1]The relaxation is a greedy process at each stage choosing the relaxation which least increases the area of the convex hull.

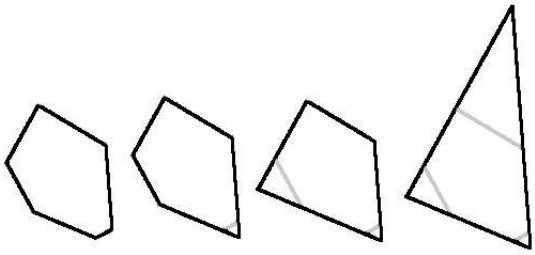

Figure 8: Successive relaxed convex hulls.

It should be noted that this algorithm does not find the smallest N-sided polygon that contains the convex hull, nor does is it intended to.

Gradient The above features take no account of the internal structure of the object. Various feature detectors were tried, but without much success. Due to the simple monotone nature of the images, and the limited resolution available from the webcam even edge detection proved very difficult, especially with the same parameters used across the database.

So only two very simple features were used; AverageGradient is defined as the average intensity of the Sobel gradient, PercentEdge is defined as the ratio of the number of edge pixels (found by the Canny edge detector) to the total number of pixels.

Symmetry Measures of symmetry are taken about the major, and minor axis as defined by the moments. For shape, the symmetry is defined as the ratio of the area of the convex hull to the area of the convex hull including all points mirrored about the respective axis, called CHMajorAxisSymmetry and CHMinorAxisSymmetry.

A measure of symmetry that takes into account the internal structure is defined as the Normalised Cross Correlation of the images and itself mirrored about the respective axis, called MajorAxisSymmetry and MinorAxisSymmetry.

4.3 Results

Despite the aim of 1000 examples of each class, the collection process was not perfect, some classes ended up with significantly more examples and some contained a few errors. These results (Figure 9) are based on a finalised database of 89 classes each with 900 examples. When training was performed with N examples per class, testing was performed with the remaining $900 - N$. The overall classification rates for each tree is defined as the

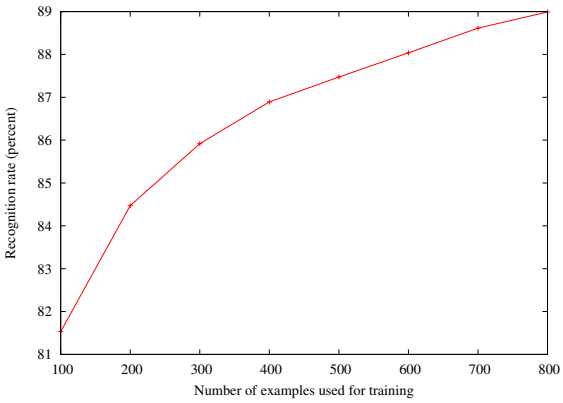

Figure 9: Recognition Rates for the whole database (89 classes).

percentage of correctly classified (unseen) tests. The classification of a particular example is taken as the modal class in the leaf node of the decision tree that the example descends to (in almost all cases, this modal class was the only class).

Classification rates are given per block (Figure 4.3) for the most exhaustive test (800 training examples per class).

Table 1: Classification rates per block. For each type of block an example image is given and classification rates are given for each colour, as well as for the colour blind tests.

Image	Class description	Recognition rates	
	angleblock-2x2	black	0.82
		no colour	0.76
	anglebracket-2x2	grey	0.83
		no colour	0.76
	axle-2	black	0.91
		no colour	0.90
	axle-3	black	0.96
		no colour	0.94
	axle-4	black	0.97
		no colour	0.93
	axle-5	black	0.98
		no colour	0.96
	axle-6	black	0.99
		no colour	0.97
	axlebrace-2x2	grey	0.84
		no colour	0.77
	beam-2x1	black	0.82
		blue	0.82
		red	0.87
		yellow	0.79

continued from previous page

Image	Class description	Recognition rates	
		no colour	0.47
	beam-4x1	black	0.83
		green	0.98
		red	0.92
		no colour	0.76
	beam-6x1	black	0.91
		blue	0.97
		red	0.96
		no colour	0.85
	beam-8x1	blue	0.98
		no colour	0.92
	beamxhole-2x1	green	0.76
		no colour	0.50
	block-2x1	black	0.74
		green	0.73
		yellow	0.79
		no colour	0.55
	block-2x2	black	0.86
		blue	0.93
		green	0.79
		red	0.90
		yellow	0.91
		no colour	0.70
	block-3x2	black	0.80
		blue	0.84
		green	0.67
		red	0.88
		yellow	0.85
		no colour	0.53
	block-4x2	black	0.93
		blue	0.95
		green	0.93
		red	0.94
		yellow	0.90
		no colour	0.78
	block-6x2	black	0.90
		blue	0.97
		red	0.93
		yellow	0.97
		no colour	0.80
	block-8x2	black	0.95
		yellow	0.99
		no colour	0.90
	blockcurve-3x2	green	0.65
		no colour	0.58
	cam-4	grey	0.96
		no colour	0.93
	compoundgear-half	grey	0.93

continued from previous page

Image	Class description	Recognition rates	
		no colour	0.85
	connector-4	grey	0.97
		no colour	0.96
	connector-l	red	0.99
		no colour	0.95
	endaxle-2x1	black	0.94
		no colour	0.87
	flat-1x1	grey	0.82
		no colour	0.74
	flat-2x1	grey	0.85
		no colour	0.78
	flat-2x2	green	0.95
		grey	0.92
		no colour	0.85
	flat-4x1	grey	0.85
		no colour	0.79
	flat-4x2	yellow	0.95
		no colour	0.90
	flat-8x1	grey	0.97
		no colour	0.94
	flathole-4x2	grey	0.93
		no colour	0.96
	flathole-6x2	grey	0.98
		no colour	0.98
	gear-12	grey	0.88
		no colour	0.92
	gear-16	grey	0.94
		no colour	0.93
	gear-24	grey	0.98
		no colour	0.98
	gear-40	grey	0.99
		no colour	0.99
	gear-9	grey	0.89
		no colour	0.88
	hingebase-3x1	black	0.84
		no colour	0.76
	holeconnector-3	black	0.94
		grey	0.92
		no colour	0.91
	holeconnector-5	grey	0.98
		no colour	0.95
	hub-2x1	yellow	0.95
		no colour	0.88
	hub-medium	white	0.99
		no colour	0.98
	hubtyre-medium	yellow	0.94
		no colour	0.94
	hubtyre-small	yellow	0.97

continued from previous page

Image	Class description	Recognition rates	
		no colour	0.96
	lbeam-5	black	0.92
		no colour	0.90
	ornament-8	grey	0.90
		no colour	0.86
	peg-1	grey	0.94
		no colour	0.93
	peg-2x2	yellow	0.98
		no colour	0.96
	pegend-1	blue	0.97
		no colour	0.78
	pulley-2	grey	0.94
		no colour	1.00
	pulley-6	grey	0.91
		no colour	0.98
	pulley-small	grey	0.83
		no colour	0.82
	rack-10	grey	0.83
		no colour	0.80
	reversewedge-2x1	black	0.63
		yellow	0.74
		no colour	0.60
	reversewedge-3x2	black	0.81
		no colour	0.77
	sensorpush-3x2	grey	0.87
		no colour	0.64
	wedge-2x1	black	0.66
		yellow	0.72
		no colour	0.63
	wedge-2x2	green	0.76
		no colour	0.67
	wing-1	orange	1.00
		no colour	0.99

As can be seen there are a few classes which make up the majority of errors. In many cases this is expected, for example, there are many views from which a BEAM-BLACK-2X1 looks just like a BLOCK-BLACK-2X1.

Another measure of performance is the size of the produced tree, we measure this by looking at the total number of terminal nodes (Table 2). It is quite clear that there are many classes which are easily handled, and some which are more troublesome. In general, these issues can be dealt with at a lower level through matching techniques.

Colour Blind

Part of the beauty of a decision tree is that high level splits can split well on simple attributes, and divide one large dataset into several small datasets. Unfortunately this makes our main objective, scalability, hard to test.

Table 2: Terminal nodes.

Number of examples (per class)	Number of terminal nodes
100	757
200	1,311
300	1,809
400	2,238
500	2,658
600	3,101
700	3,452
800	3,842

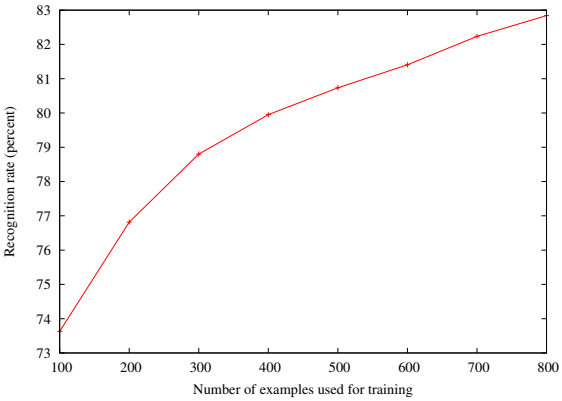

Figure 10: Recognition Rates for the limited database (60 classes) with the feature of colour removed.

In this case the early splits will inevitably be colour based, and due to the subject matter, will be quite accurate. In effect this divides one reasonably large dataset neatly into smaller more manageable datasets. Since several objects only appear in one colour, the recognition of them is made simple, for example there is never any need to differentiate between a hub and a gear, since they are different colours, despite the fact that they are both circular.

To alleviate this the system was tested in an effectively colour blind manner. That is, classes were relabelled to avoid any distinction based on colour[2], and the three features relating to colour were removed. The resulting database was reduced to 60 classes (Figure 10). Although some of the classes now had more examples, the number used to learn and test remained the same (randomly selected from all samples of the class).

The results take the same format as above.

4.4 Performance

All performance benchmarks were taken on an Intel® Pentium® 4 CPU 3.00GHz, with 1-Gigabyte of RAM, running Debian GNU/Linux. The features were all pre-processed, and took an average of 0.1 seconds to compute for each image. This could probably be considerably improved upon since all code was implemented from scratch (without taking advantage of optimised image processing libraries). The generation of the decision trees and average classification time were timed for several training examples per class (Table 3).

[2]For example BLOCK-BLACK-4X2 and BLOCK-GREEN-4X2 were both labelled BLOCK-4X2

Table 3: Performance for the first tests (89 classes)

Number of training examples (per class)	Avg time to build tree	Avg classification time
100	88s	0.2ms
200	176s	0.28ms
300	407s	0.55ms
400	535s	0.66ms
500	730s	0.9ms
600	555s	0.6ms
700	674s	0.67ms
800	778s	0.72ms

There are some small anomalies, but this is probably due to varying load on the testing system. Again, this could be significantly improved upon. For one thing the code is implemented in pure python, re-implementation in C would offer significant performance gains. It is obvious though, considering the feature calculation time, and the classification time, that at this point the bottleneck is in the feature computation.

5 Future Work

Rather than using one of the standard "off the shelf" decision tree systems, a decision tree system has been implemented in Python. This offers many opportunities to improve upon the conventional approaches used. At the current stage, the decision tree system is loosely an implementation of that described in [Breiman *et al.*, 1984], and at a high level this structure will remain the same. There are several possible improvements to this implementation, some taken from the machine learning and data mining literature, others somewhat specific to the task of object recognition.

There are two key advantages using decision trees in object recognition, over the traditional general use:

- Classifications can be validated. That is we have access to more information than just the features, once a classification has been made it can be tested against the learned examples in that terminal node.
- One of the main applications of decision trees is finding patterns in large datasets that can easily be interpreted by domain experts. That is, at least partially, why conventional trees often make overly simple splits. In this application simple intuitive splits are less of a priority.

Lazy Feature Evaluation Features only need to be computed for objects if they are needed by the decision tree. This in itself isn't going to be a huge performance improvement, but only by doing this are later improvements possible, ultimately, this frees us from being limited to a fixed number of features.

Computational Cost Based Feature Selection The classical use of decision trees is based on datasets where the attribute values are given. The difference in object recognition is that the features must be computed, but different features have different computational costs. By incorporating the computational cost into the selection of feature, deeper decision trees, which would normally be considered worse, may be considered better if though asking more questions, the questions are cheaper.

Multiple Questions Per Node Going back to the example of twenty questions, it was suggested that the perfect player has access to the perfect tree, but really, is there such a thing? Quite often with human players, even though they might be asking a reasonable question, ask a question that for which the answer is not entirely clear. In which case the interrogatee will ask the interrogator to "ask another question". Now obviously, this fuzzy boundary problem will not have a precise solution, but an attempt could be made.

Suppose for a moment that at each node rather than making a pair of child trees for *the best* question considered, a pair was made for *every* question considered. Statistical analysis could be done on each set of answers to determine a range around the boundary of the answer that was considered "uncertain". Now, when classifying at that node, if the answer to the first question was "uncertain", the second question could be tried, and so on. A similar system was considered in [Kohavi and Kunz, 1997]. There are several reasons why this isn't a widely used approach in the data mining community, but they might be less important in object recognition. In particular, it requires an increased number of nodes, but there are ways to limit this, especially if misclassifications are possible to detect. Secondly, the tree is generally harder to interpret by humans, but this is much less of an issue in this application.

Dynamic features One problem with decision trees, and indexing algorithms in general, is that the number of features is generally required to be static. Usually the input to the learning system is a set of N-tuples, for a fixed set of N features. In general, coming up with many features isn't too hard, but indexing them is.

Several options are available with decision trees to expand the list of features as the tree is descended and the dataset is divided. This allows for features that are only applicable once other conditions have been met. For example, you could imagine a question that measured relationships between polygonal faces. This feature could be only asked at nodes where it had already been established that there *were* polygonal faces.

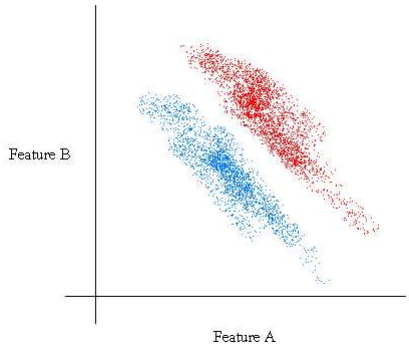

Figure 11: A trivial example where a multiple dimension split is clearly optimal.

Another option is parameterised features, again, as the tree is descended, the objects that are being split are somehow closer, more work may need to be done to meaningfully split the data.

There appears to be little work along these lines in the literature, mainly because it is only an option in a field such as object recognition when there is more to an entry than its N features.

Multi-dimensional Splits A common failing with decision trees is that they only split on one dimension, often this simply isn't good enough(Figure 11). Again, there are several reasons that multi dimensional splits aren't widely used in traditional decision trees. Firstly, it has generally been thought to be computationally prohibitive. The problem lies mainly in the choice of multiple dimensions - there are too many combinations. Secondly, again the trees are of less use to domain experts, since the multiple dimension splits are harder to interpret.

As previously, the intuitive splits do not concern us so much, but difficulty still lies in the combinatorial problems. But this is a current area of research, in [Cantu-Paz, 2003] evolutionary algorithms are used to generate so called oblique trees (i.e. not axis parallel). There are many such approaches that could be applied, and room for new approaches to this problem.

Traditional Machine Learning Improvements There are several techniques that can sometimes be used to improve the results of classifiers. Voting techniques,

where an ensemble of classifiers are built, such as Boosting and Bagging often improve results. Also different techniques of tree pruning can be useful. However, it is best not to apply these methods blindly, as shown in [Bauer and Kohavi, 1999] can sometimes be detrimental, especially if specialisations like the above have been applied.

6 Conclusion

A new object recognition system was built, capable of promising recognition rates on a large challenging image database. The performance is easily sub-second for recognition and training time very short, considering the size of the database. Decision trees are shown to handle the task of splitting up a large database very well. Several avenues of research have been considered, allowing for utilisation of the rich data available in image data within the structure of the decision tree. Clearly, there is much work that can be done in this area.

Acknowledgements

This work was supported by funding from National ICT Australia. National ICT Australia is funded by the Australian Government's Department of Communications, Information Technology and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Centre of Excellence program.

References

[Bauer and Kohavi, 1999] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139, 1999.

[Breiman *et al.*, 1984] L. Breiman, J. H. Friedman, R.A.Olshen, and C.J. Stone. *Classification and Regression Trees*. 1984.

[Cantu-Paz, 2003] Erick Cantu-Paz. Inducing oblique decision trees with evolutionary algorithms, 2003.

[Cortes and Vapnik, 1995] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[Csetverikov, 2003] Dmitrij Csetverikov. Basic algorithms for digital image analysis: a course, 2003.

[Cyr and Kimia, 2004] C. M. Cyr and B. B. Kimia. A similarity-based aspect-graph approach to 3d object recognition, April 2004.

[Faugeras *et al.*, 1992] Olivier Faugeras, Joe Mundy, Narendra Ahuja, Charles Dyer, Alex Pentland, Ramesh Jain, and Katsushi Ikeuchi. Why aspect graphs are not (yet) practical for computer vision. In *CVGIP Image Understanding Workshop*, volume 55, pages 212 – 218, March 1992.

[Kohavi and Kunz, 1997] Ron Kohavi and Clayton Kunz. Option decision trees with majority votes. In Doug Fisher, editor, *Machine Learning: Proceedings of the Fourteenth International Conference*. Morgan Kaufmann Publishers, Inc., 1997.

[Murase and Nayar, 1995] H. Murase and S. K. Nayar. Visual learning and recognition of 3d objects from appearance. *International Journal of Computer Vision*, 14(1):5–24, 1995.

[Nene *et al.*, 1996] S. Nene, S. Nayar, and H. Murase. Columbia object image library: Coil, 1996.

[Osuna *et al.*, 1997] E. Osuna, R. Freund, and F. Girosi. Training support vector machines:an application to face detection, 1997.

[Pontil and Verri, 1998] Massimiliano Pontil and Alessandro Verri. Support vector machines for 3d object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(6):637–646, 1998.

[Quinlan, 1990] J. R. Quinlan. Induction of decision trees. In Jude W. Shavlik and Thomas G. Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann, 1990. Originally published in *Machine Learning* 1:81–106, 1986.

[Quinlan, 1992] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1992.