Object Recognition by Parts

Mattias Boström

MSc by Research in Neural Networks and Pattern Recognition

The University of Aston in Birmingham

September.1997

THE UNIVERSITY OF ASTON IN BIRMINGHAM

Object Recognition by Parts

MATTIAS BOSTRÖM

MSc by Research in Neural Networks and Pattern Recognition, 1997

Thesis Summary

Object recognition is one of the major challenges in computer vision and a vast number of approaches have been proposed. One approach to this problem is to try to recognize an object by building feature detectors for its various parts, and then checking that the parts lie the correct spatial relationships. A key advantage of the recognition-by-parts strategy is that it is robust to problems of occlusion that bedevil strategies based on observing the whole object.

In this thesis we examine feature detectors for images of 3-D objects which use a $m \times m$ window of the image as the input. A number of feature detectors including multiple logistic regression, linear subspace models, k-nearest-neighbours and different types of artificial neural networks are investigated. The performance of these classifiers has been assessed using both representative test sets and receiver-operating-characteristic (ROC) curves.

Keywords: PCA, K-Nearest-Neighbours, Neural Networks, Object Recognition

Acknowledgements

First of all I would like to thank my supervisor Dr. Chris Williams for his guidance and good advice throughout the project. Apart from sharing his knowledge and insight into object recognition with me he also provided me with good references.

Thanks also to my fellow MSc students at the Pattern Analysis and Neural Network program who in addition to giving good advice and useful tips also were supportive late nights when software bugs and writers block made life miserable.

The rest of the people in the Neural Computing Research Group who always have been willing to help and have made this year at Aston an interesting and stimulating year to remember.

Linköping University in Sweden that sent me here to do my final year as an exchange student deserves to be mentioned.

And finally but not least Claudia for giving support and correcting my English.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer vision deals with the problem of automating processes and representations used for vision perception. It includes areas such as image processing, statistical pattern classification, techniques for geometric modelling and cognitive processing. As we can see it spans a large number of topics and many of them are treated as a research area in themselves because of their complexity and usefulness. In our everyday life we don't think too much about all the difficult tasks our biological vision system has to cope with, but when it fails we usually become aware of it. One important capability we have is to recognise objects. The basic case is to separate objects from the background, but we also have more complex and specialised capabilities like face recognition. The ability to perform object recognition is essential in order to interact with our surroundings. Often it is not enough to just recognise an object, we also need to know where it is in our environment, i.e. localise it, which can be achieved in the same process.

Some tasks concern two dimensional objects like recognising letters when we read and picking out Mona-Lisa in a painting. Others deal with three-dimensional objects like finding a coffee mug when looking down in the sink which is different from recognising it in someone's hand. The appearance of the mug is different, although it is the same object. Obviously a three dimensional object recognition problem is more complex because of the higher number of degrees of freedom (this is not the same as saying that recognising letters is easy!).

For the reasons mentioned above, i.e. importance and complexity, object recognition is one of the major challenges in computer vision and a vast number of approaches have been proposed. An obvious component of it is to have some sort of sensor and map this sensor input to a suitable space where the objects have certain properties. For a vision system to be able to recognise objects, it must have knowledge about the objects stored in its memory. This knowledge about the objects is then compared to the mapped data from the sensor input to find a match in some sense. A successful matching means that the object is recognised. The most common sensor and the one that has been used in all the work below is a camera whose output is digitised into a gray-scale or colour image. To incorporate this knowledge about objects several different methods can be applied.

1.1 Recognising objects using global properties

One category tries to recognise objects using global properties of the object. One example of this approach is Turk and Pentland (1991), which treats face recognition as a pattern recognition problem and uses principal components of the face images, referred to as eigenfaces, to model the faces. A new image is recognised by projecting the image into "face space", consisting of the first few eigenfaces. By calculating the Euclidian distance to different face-classes a classification is made.

A similar approach is used in Murase and Nayar (1995) where a technique to automatically learn three-dimensional objects from their appearance in two-dimensional images is presented. The appearance of an object is the combined effect of its shape, reflectance properties, pose in the scene, and the illumination conditions. Each object is modelled with a manifold, parameterised by pose and illumination, in the universal eigenspace, which is used to distinguish between objects.

One limitation with this approach is that before we can recognise the object we have to find it in the scene. Segmentation of an input image is done in the first example by using motion detection. For face-detection one might argue that most of the time

a face is in motion but for other objects this is not always the case and other methods have to be used. Another problem is that if an object is occluded these algorithms have difficulties as they rely on using the whole object for the recognition.

1.2 Object recognition by parts

Another technique, that is more robust to the above mentioned problems, can be described as recognition by parts. The idea is to extract local features from the image and consider all possible correspondences between them and similar local features in the model. Occlusion will remove some features from consideration but by allowing missing features in the matching phase the object can still be recognised.

In Grimson (1990) this approach is thoroughly examined using geometric properties, like edges and surfaces, as local features. With primitive feature detectors like edge detectors etc. we get many positive responses from an image, i.e. we can find a lot of edges in it. This results in a correspondence space that is too big to search. But as the objects in question are rigid, shapes invariant under the allowed transformations are used to constrain the correspondence search. For instance relative angle between two edges can be used to eliminate a large number of constellations from consideration. A number of different constraints are examined. Both two and three dimensional objects are discussed.

Another way of reducing the correspondence search is to use more complex features to reduce the matching that has to be performed. An example of this is Burl and Perona (1996) which considers non-rigid objects, namely faces. To detect the features a technique based on matching descriptors produced by multi-orientation, multi-scale Gaussian derivative filters was used. These local feature detectors are used to identify candidate feature locations for eyes, lips and so on, which are grouped into hypotheses. The hypotheses are scored based on the spatial arrangement of the features. A joint probability density over feature positions is used for scoring which allows for deformability of the model.

Similar work has been carried out in Cootes and Taylor (1996) where statistical feature detectors are constructed and easily detectable features are automatically chosen to form a model. All plausible sets of features are located in an image and tested against a shape model to locate the object of interest. The main difference compared to Burl and Perona (1996) is the type of feature detector used. Here principal components are used to create "eigenfeatures" which are then used in the same manner as the "eigenfaces" in Turk and Pentland (1991).

In this thesis we are seeking good feature detectors. If we want an recognition-by-parts algorithm for rigid, 3-D objects what kind of local feature detector should we use? The appearance of the features in two dimensional gray-scale images are used as the space where feature detection is made. The geometric properties of the objects are not explicitly used in the detection. The methods tested include linear and non-linear statistical pattern recognition methods. We compare the feature detectors by using both representative test sets and receiver-operating-characteristic (ROC) curves.

1.3 Thesis outline

Chapter 2 describes the database that has been used for this project. It also explains how the images in this database has been processed to construct input and target data for the different methods.

Chapter 3 describes the theory behind all the different methods that have been used in this project. These include linear, generalised linear, non-linear and kernel methods.

Chapter 4 explains the two different ways in which we have evaluated the performance of the methods. The results from the experiments with the different models are given and a comparison between them is made.

In **Chapter 5** I discuss the results from the experiments and give some suggestions on how to improve the performance and possible directions for future work.

Chapter 2

The Data

The ultimate goal in object recognition is to take a cluttered image and identify an object in real-time. However as a first step when exploring new approaches it might be sensible to try the methods without confusing background and time constraints. The data used in this project comes from Columbia Object Image Library (COIL-20) (Nene, Nayar, and Murase 1996). This is a set of 20 objects in gray-scale seen in figure 2.1. For each object there are 72 images showing the object from different angles.

2.1 COIL-20

The image acquisition was done in the following way. An object was placed on a turntable covered with black cloth, also the background was covered with black cloth. The turntable was rotated 360 degrees and at every 5 degrees of rotation an image was taken. The digitised images where resized to 128×128 pixels with preserved aspect ratio. In addition to the resizing every image was histogram stretched, giving it pixel intensities between 0 and 255. The images were saved as 8-bit PGM (portable graymap) images.

Figure 2.1: The 20 different objects in the COIL-20 database

2.2 Generation of Data

The 72 images of each object were divided into training-set (32 images), validation-set (16 images) and test-set (24 images). This was done randomly but with a constraint making sure that the training images where spread out over all the 72 images. A software tool was developed to interactively label features in the images, this and other software tools are described in appendix A. All the programming was done in MATLAB (The MathWorks, Inc.).

A number of object specific features were chosen and their position in the images determined. The objects used are the piggy-bank and the toy car. In figure 2.2 and 2.3 example images are shown where some of the feature-labels (table 2.1 and 2.2 respectively) are visible. There are also 10 randomly placed background-labels in every image, used to represent the background. As constraint for the placement of the 10 background-labels a minimum distance of 15 pixels to nearest other label was used.

Using the position of all the labels $m \times m$ pixel blocks have been cut out from the corresponding images. Finally placing the rows from one block after each other creates a vector for each label that has been used as input data for the models, denoted $\boldsymbol{x}$ with length d. The size of the block that we used for most of the experiments in this project is 15×15 pixels which results in 225-dimensional input vectors, $d = 225$. The

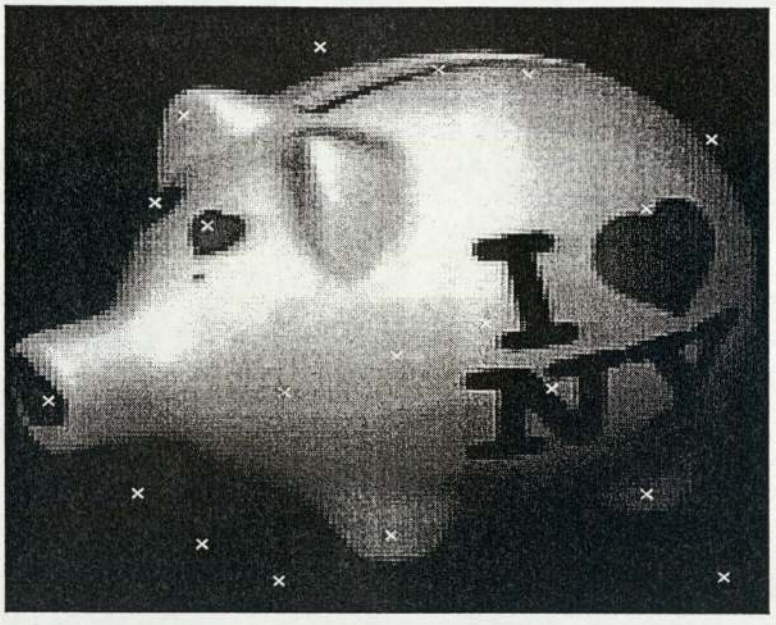

Figure 2.2: One view of the piggy-bank

Nr.	Feature Name	Nr.	Feature Name
1	Front Left Leg	9	Right Eye
2	Back Left Leg	10	Tail
3	Front Right Leg	11	Money Slot
4	Back Right Leg	12	N
5	Snout	13	Top of Right Heart
6	Left Ear	14	Top of Left Heart
7	Right Ear	15	Background
8	Left Eye		

Table 2.1: The piggy-bank features

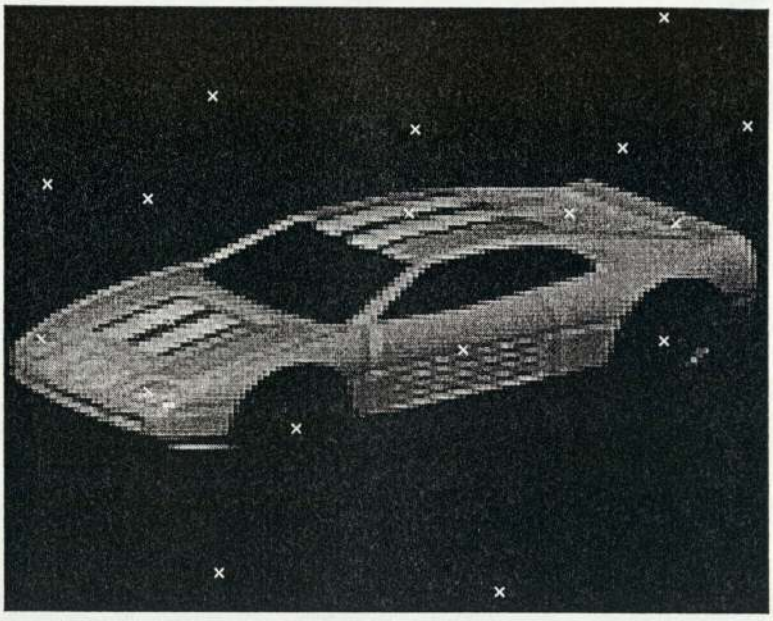

Figure 2.3: One view of the toy car

Nr.	Feature Name	Nr.	Feature Name
1	Front Left Wheel	8	Back Plate
2	Back Left Wheel	9	Left Front Light
3	Front Right Wheel	10	Right Front Light
4	Back Right Wheel	11	Left Door
5	Hood	12	Right Door
6	Top	13	Background
7	Trunk		

Table 2.2: The toy car features

original range of the values for each pixel have been transformed from [0,255] to [-1,1]. To each vector $\boldsymbol{x}$ an output vector $\mathbf{t}$ is associated. These have the length c which is the number of different features (classes). The vectors $\mathbf{t}$ are 1-of-c binary coded, meaning that $t_i = \delta_{ij}$ where $j =$ feature-number and δ is the Kronecker delta function[1].

Due to the fact that the images are obtained by a rotation of the object, each feature is not present in all the images. This means that in the training data each feature is represented by approximately 10 to 32 different example vectors, depending on the location of the feature on the actual object. This means that we can have as few as 10 examples of a class in a 225-dimensional space which is a key problem in this project.

Since we are labelling by hand, the exact centre of a feature is difficult to determine. Even if the label would have been placed in a adjacent pixel, it should still be a valid example of the same feature. This is known as invariance to shift and can be exploited to try to solve the problem above. If we transform our original $\boldsymbol{x}$ to a shifted version and add these to the training set, with the same target vectors $\boldsymbol{t}$, we increase the number of training examples without having to do more labelling or get more images of the object. We can shift the $\boldsymbol{x}$ in 4-directions (horizontal and vertical) or 8-directions (the above plus diagonal) resulting in 5 or 9 times as much training data. This can be done by multiplying $\boldsymbol{x}$ with a perturbation matrix, copying a row or column, from the original block cut out from the image, into the same row or column while shifting all the others one step. Eg. a shift to the left is done by shifting all columns one step to the left and placing a copy of the old first column as a new first column.

[1] $\delta_{ij} = 1$ iff $i = j$

Chapter 3

Methods

This chapter introduces the different ways in which we can model the data in order to classify novel data. What is meant by this is that for each input vector $\mathbf{x}$ there is a corresponding output vector $\mathbf{t}$ indicating which class $\mathbf{x}$ corresponds to. The training data is used to build a model of the data and when a novel input vector is presented to the model, the model predicts which class this vector corresponds to. There are a number of different ways in which the models can be built. The parameters in the model can be explicitly calculated from the training data in one shot or a training procedure can be iterated in which an error function is minimised to find the parameters for the model. The model can be either linear or non-linear.

Section 3.1 explains principal components analysis (PCA), which is a linear projection method. Sections 3.2 and 3.3 explain two other methods, multiple independent attributes (MIA) and multiple logistic regression (MLR). The linear subspace classifier (LSC) is handled in section 3.4 , and a kernel method (K-nearest-neighbours (K-NN)) in section 3.5. Two different kinds of neural networks (NN) are explained in sections 3.6 (softmax NN) and 3.7 (MIA NN).

3.1 Principal Component Analysis

Principal components analysis (PCA) (Bishop 1995), also known as Karhunen-Lóeve expansion, is a projection method, performing a linear transformation on the data. It is used for dimensionality reduction and works by the following principle. We have a dataset $(x^i, i = 1, .., N)$ with a mean $\boldsymbol{\mu}$ and a covariance matrix Σ defined as

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}^n \ , \ \Sigma = \sum_{n=1}^{N} (\boldsymbol{x}^n - \boldsymbol{\mu})(\boldsymbol{x}^n - \boldsymbol{\mu})^T. \tag{3.1}$$

The vector $\boldsymbol{x}$ in our original space can be written as

$$\boldsymbol{x} = \sum_{i=1}^{d} z_i \boldsymbol{u}_i \tag{3.2}$$

where d is the dimension of the original space and u_i is a set of orthonormal vectors. If we want to use a lower dimension to express $\boldsymbol{x}$ we can do it by writing it as

$$\tilde{\boldsymbol{x}} = \sum_{i=1}^{M} z_i \boldsymbol{u}_i + \sum_{i=M+1}^{d} b_i \boldsymbol{u}_i \tag{3.3}$$

where the constants (b_i) in the second term replaces the coefficients (z_i). This is like cutting out a hyper-plane in the original space. The error that occurs due to this is

$$\boldsymbol{x} - \tilde{\boldsymbol{x}} = \sum_{i=M+1}^{d} (z_i - b_i)\boldsymbol{u}_i. \tag{3.4}$$

We can formulate the sum-of-squares error over the entire dataset, containing N different vectors, introduced by this procedure as

$$E = \frac{1}{2}\sum_{n}\sum_{i=M+1}^{d} (z_i^n - b_i)^2. \tag{3.5}$$

If we want to minimise this with respect to the b_i's we first have to set

$$\frac{\partial E}{\partial b_i} = \sum_{n=1}^{N} (z_i^n - b_i) = 0. \tag{3.6}$$

Which using equations 3.1 and 3.2 gives us

$$b_i = \frac{1}{N}\sum_{n=1}^{N} z_i^n = \boldsymbol{u}_i^T \boldsymbol{\mu}. \tag{3.7}$$

We can now rewrite 3.5 using 3.1 and 3.2 as

$$\begin{aligned} E &= \frac{1}{2}\sum_{n=1}^{N}\sum_{i=M+1}^{d}(\boldsymbol{u}_i^T\boldsymbol{x}^n - \boldsymbol{u}_i^T\boldsymbol{\mu})^2 \\ &= \frac{1}{2}\sum_{i=M+1}^{d}\sum_{n=1}^{N}(\boldsymbol{u}_i^T(\boldsymbol{x}^n - \boldsymbol{\mu}))^2 \\ &= \frac{1}{2}\sum_{i=M+1}^{d}\boldsymbol{u}_i^T\Sigma\boldsymbol{u}_i \end{aligned} \tag{3.8}$$

In (Bishop 1995) it is shown that in order to minimise E you should choose the $d - M$ smallest eigenvectors of the covariance matrix of $\mathbf{X}$ as the $\boldsymbol{u}_i$'s to replace. This leaves us with the residual error

$$E_{Min} = \frac{1}{2}\sum_{i=M+1}^{d}\lambda_i, \tag{3.9}$$

where λ_i are the eigenvalues of Σ. This means that to reduce a d dimensional vector to M dimensions in order to preserve most of the information we should project them onto the M largest eigenvectors of the covariance matrix, the principal components. This has the effect that most of the variation in the data is accounted for by the first few PC's. This method was used on the training data to reduce the dimension of the input vectors by projecting them down into what we call the universal PC space. It was also used to calculate the class specific subspaces used in the Linear Subspace Classifier method described in section 3.4.

3.2 Multiple Independent Attributes (MIA) Classifier

We can construct a linear discriminant function $y(\boldsymbol{x})$ to predict if an input vector $\boldsymbol{x}$ belongs to a certain class or not as

$$y(\boldsymbol{x}) = \sum_{i=1}^{d} w_i x_i + w_0 \tag{3.10}$$

where d is the number of input dimensions, w_i are the parameters in the discriminant function and w_0 is the bias. We say that $\boldsymbol{x} \in \mathbb{C}$ if $y(\boldsymbol{x}) > 0$ otherwise not. The equation 3.10 can be written in a simpler form if we include the bias in the sum by introducing $x_0 = 1$, which gives us

$$y(\boldsymbol{x}) = \sum_{i=0}^{d} w_i x_i \tag{3.11}$$

If we want to do some post-processing of the prediction it can be useful to be able to interpret $y(\boldsymbol{x})$ as a probability, i.e. in the range [0,1]. This can be achieved by using the logistic sigmoid

$$g(a) \equiv \frac{1}{1 + \exp(-a)} \tag{3.12}$$

with the sum in equation 3.11 as the input a. This also makes it easy to decide the target for the training of the model, namely if $\boldsymbol{x} \in \mathbb{C}$ then $t = 1$ otherwise $t = 0$.

If we have more than one class we can extend this model by having one output y_k for each class which gives us something that can be viewed as a number of one-layer neural networks (see figure 3.1).

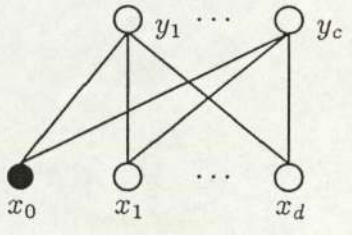

Figure 3.1: Structure of MIA and MLR Classifier

The y_k's can be described using equations 3.11 and 3.12

$$y_k(\boldsymbol{x}) = g(\sum_{i=0}^{d} w_{ki} x_i) \tag{3.13}$$

To decide which class (attribute) the input vector $\boldsymbol{x}$ belongs to we have to compare the different outputs and pick the largest one. This model is called multiple independent attributes (MIA) (Bishop 1995). Compared to the softmax model (discussed

in section 3.3) which gives us normalised probabilities this model can give several attributes a high probability.

The number of free parameters in the model is $(d+1)$ for each class which is $(d+1)c$ in total, but since the outputs are independent we can look at each net separately. Consequently if the number of training examples (N) is smaller than $d+1$, then there are an infinity of exact solutions and the model is not well specified by the data. If $N = d+1$ we have a generalised linear system with a unique solution for the w's. And finally if $N > d+1$ it is not possible to solve the system exactly and we have to find the weight vector $\mathbf{w}^*$ that minimises the error function (equation 3.14). For the problems we have looked at there is sufficient data so that $N > d+1$ regardless of if we use the original 225 dimensional input or the smaller universal space discussed in section 3.1. In order to find $\mathbf{w}^*$ the scaled conjugate gradient (scg) search method (see for instance (Bishop 1995)) is used to minimise the negative log likelihood cross-entropy error function,

$$E = -\sum_n \sum_{k=1}^{c} t_k^n \ln y_k^n + (1 - t_k^n) \ln(1 - y_k^n). \tag{3.14}$$

When we use the y_k as given in equation 3.13 this error function has one unique minimum with respect to $\boldsymbol{w}$.

As mentioned in chapter 2 background labels have been added to help train the models to distinguish between features and background. However a correct classification of the background is not as important as correct classification of features. Since we have more examples of background than of features (approx. 50% of the labels are background and the rest are divided among the different features) a model always guessing that the background feature is the class of the input would perform quite well. One way to try to avoid this is to make the background labels less important in the training. This can be done by associating a weight (α_k^n) to all data points, setting this to one for all features except for the background features for which it is set to a lower value. The error function given in equation 3.14 can then be modified so that (α_k^n) is multiplied to the factors inside the summation. We can call this method importance

weighting of the training data.

3.3 Multiple Logistic Regression with softmax output

Multiple logistic regression (MLR) is very similar to MIA in section 3.2. If we instead of having independent outputs make sure that we have a normalised probability, i.e. the sum of the outputs equals one, we get the MLR. We can do this by using a different activation function. Instead of the logistic sigmoid in equation 3.12 we use the softmax function:

$$g(a_k) \equiv \frac{\exp(a_k)}{\sum_{k'=1}^{n} \exp(a_{k'})} \tag{3.15}$$

with the sum in equation 3.11 as the input a_k's. We have a layout as in figure 3.1 with the difference that the y_k's depend on each other. This also means that instead of c nets with $d+1$ free parameters we have one big net with $c(d+1)$ free parameters. We now have the following conditions on the number of training examples N. If the number of training examples is smaller than $c(d+1)$, then there are an infinity of exact solutions and the model is not well specified by the data. If $N = c(d+1)$ we have a generalised linear system with a global minima with respect to $\boldsymbol{w}$. And finally if $N > c(d+1)$ it is not possible to solve the system exactly and have to find the weight vector $\mathbf{w}^*$ that minimises the error function (equation 3.16). In our cases we don't have enough training data and therefore we have to do dimensionality reduction before we can feed our input to the model and we use the universal space discussed in section 3.1. The error function for this net is different from the one for MIA. Since we have softmax output we get the negative log likelihood error function

$$E = -\sum_{n}\sum_{k=1}^{c} t_k^n \ln y_k^n. \tag{3.16}$$

We use the same search method (scg) as for the error function given in equation 3.14 to find the global minimum.

The same method, with (α_k^n) as weighting of labels, as used for the cross-entropy error function in equation 3.14 can be used for this error function.

3.4 Linear Subspace Classifier (LSC)

In section 3.1 we explained how we calculated the universal PC space. If we don't use all our data to do PCA, but instead it is divided into c different sets according to which class the data belongs to, and we perform PCA for each set of vectors, we get feature-specific PC spaces. These are optimised to represent the input vectors for the specific features. If we assume that all of the examples for each feature are localised in a small subspace of the original space, and the subspaces are different from all of the PC spaces for the other features, we can say that the error due to dimensionality reduction should be smallest for a novel data point when the correct PC space is used. The sum-of-squares error could be calculated by squaring the expression in equation 3.4 but another way to calculate it is

$$E = (\boldsymbol{x} - \tilde{\boldsymbol{x}})^T(\boldsymbol{x} - \tilde{\boldsymbol{x}}) \tag{3.17}$$

where $\tilde{\boldsymbol{x}}$ is the reconstructed approximation given by

$$\tilde{\boldsymbol{x}} = Q^T Q(\boldsymbol{x} - \boldsymbol{\mu}) + \boldsymbol{\mu}. \tag{3.18}$$

In this formula Q is a matrix with the columns set to the basis vectors in the PC-space and $\boldsymbol{\mu}$ is the mean calculated for each class in the original space. If we compare these reconstructions errors we can classify $\boldsymbol{x}$ as belonging to the class with the smallest error, that is to the class that have the PC-space that lies closest to $\boldsymbol{x}$. We can call this method Linear Subspace Classifier (LSC) A common guideline for how many PC's are needed in each subspace is that 95% of the variance should be explained by the PC's. If we want the same number of PC's in all subspace we can take the average.

In figure 3.2 we can see an example of what it could look like in a two-dimensional space with two one-dimensional linear subspaces. PC A is the first principal component

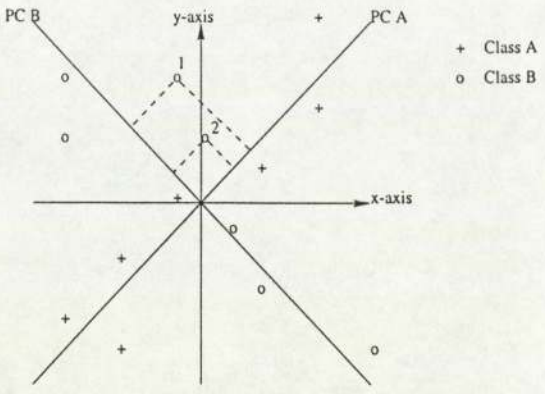

Figure 3.2: A Linear Subspace Classifier in a two dimensional space

for class A and PC B the first for class B. They both have $\boldsymbol{\mu} = 0$, otherwise the PC's would be transposed to have the centre at $\boldsymbol{\mu}$. The reconstructions error for the data-point 1 is equal to the squared distance along the dotted lines. This means that data-point 1 would be correctly classified as belonging to class B. The data-point 2 is closer to PC A which means it would be classified as class A. All the other class B data-points would be correctly classified in this example. In (Turk and Pentland 1991) LSC (called eigenfaces) is used to recognise face images and in (Hinton, Revow, and Dayan 1994) multiple LSC for each class are used to recognise digits.

3.5 K-Nearest-Neighbours

The K-Nearest-Neighbours algorithm (Bishop 1995) (Ripley 1996) can be seen as a template matching model or a kernel based model. All the training data is saved and when we want to classify a novel data point $\boldsymbol{x}^{new}$ we start by computing the distances from $\boldsymbol{x}^{new}$ to all the input vectors in the training data. We then find the K vectors that are closest to $\boldsymbol{x}^{new}$ and look at the classes for these; let these labels be denoted $t^1..t^K$. Divide these into subsets $T^1..T^c$, one for each class. y^{new} the output value for $\boldsymbol{x}^{new}$ is assigned to be the same class as the largest set T^i. If there is a tie we have to break it in some way. One way is to look at the sets of equal size and choose the one which contains the t^i with the lowest index i, that is the one that is closest to $\boldsymbol{x}^{new}$, to

win.

An other important issue is the choice of metric. One common choice is the Euclidean distance, where all the dimensions are equally important. Another possible one is the Mahalanobis distance

$$d(\boldsymbol{x}, \mathbf{z}) = \sqrt{(\boldsymbol{x} - \mathbf{z})^{\mathrm{T}} \mathbf{A} (\boldsymbol{x} - \mathbf{z})} \tag{3.19}$$

with A as the inverse covariance matrix. This is very expensive computationally, an issue which is discussed further in section 4.6.

One problem with this algorithm is that we have to save all training data and for each data point to be classified we have to calculate as many distances as there are training data. There are some algorithms that try to get around this problem. One is called multiedit algorithm (Ripley 1996) and means that you discard from the training data the points that don't improve the classification performance, i.e. you save the "important" points. Another is a tree-search algorithm (Fukunaga and Narendra 1975) that reduces the number of distances that have to be calculated in order to make a classification. The idea is to cluster training vectors that are close to each other together. For these clusters you then calculate a mean position and the distance to the point in the cluster that are farthest from the mean. By using a branch and bound search in a tree structure with this information the number of vectors to consider when we search for the closest one is reduced. These are also discussed in section 4.6.

3.6 Softmax Neural Networks

The neural network architecture used in this project is the Multi Layer Perceptron (MLP) which is a more complex non-linear model than MLR with softmax output described in section 3.3. The name can be understood if we look at the schematic figure of a MLP in figure 3.3.

It consists of one input layer, one output layer and a number of hidden layers in between. It has been shown that it is enough to have a MLP with one hidden

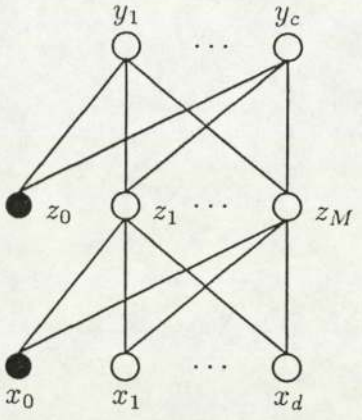

Figure 3.3: The topology of a Neural Network.

layer, if the number of nodes in the hidden layer is large enough, to be an universal approximator, i.e. it can model any arbitrary function. We can describe the network with the following equations

$$z_i(\boldsymbol{x}) = g(\sum_{j=0}^{n} w_{ij}^1 x_j) \tag{3.20}$$

where x is the input, z is the output from the hidden layer, n is the input dimension and M is the number of hidden units. The bias is included in the sum as in equation 3.11 by setting $z_0 = 1$. g is the activation function for the hidden nodes and is a logistic sigmoidal function as described in equation 3.12. The output from the hidden layer serves as input to the output layer.

$$y_k(z) = \tilde{g}(\sum_{j=0}^{c} w_{kj}^2 z_j) \tag{3.21}$$

In this equation c is the number of classes, i.e. the number of output nodes. The activation function $\tilde{g}$ in this layer is for the Softmax Neural Network a normalised exponential function, called softmax which gives the model its name, which is the same as in equation 3.15. The number of parameters in this model is $(d+1)M + (M+1)c$. To find these parameters an error function is minimised. The same error function

(equation 3.16) as in section 3.3 is used. Again scg is used to try to find the global minimum, but since we have a more complex function y_k here the error function can have a number of local minima where the search algorithm can get stuck. Another complication is that we have to calculate the gradient of the error function not only with respect to $\boldsymbol{w}^2$ but also to $\boldsymbol{w}^1$ in the first layer. This is done with a method called back-propagation (eg. (Bishop 1995)).

One way to try to improve the performance of a Neural Network is to train committees of networks and let them vote((Bishop 1995)). In the simple approach used here the only difference between the nets is the random weight-vector ($\boldsymbol{w}$) used to initiate the training. This results in that the nets get stuck in different local minima but (hopefully) some of them find a good minimum. The reduction in the error can also be viewed as arising from reduced variance in the prediction. No weighting of the votes or Bayesian approach is used, the class with most votes wins ((Bishop 1995)).

3.7 Multiple Independent Attributes Neural Networks (MIA NN)

The multiple independent attributes Neural Network works the same way as the softmax Neural Network in section 3.6 apart from the activation function in the output layer $\tilde{g}$ which instead of the softmax function is the logistic sigmoid which is the same as in equation 3.12. The difference between these two Neural Networks is therefore the same as the difference between the models in sections 3.3 and 3.2, that the outputs in MIA NN represents the posterior probability that a data-point is of a certain class independently from the probabilities that it belongs to any other class, while the output from a Softmax NN is normalised to sum to one. The error function is the same (equation 3.14) as for the MIA method described in section 3.2.

Chapter 4

Results

Using the data described in chapter 2 we want to test the different methods in chapter 3 to see if we can find which one is best suited for use in a object-recognition-by-parts scheme. The two objects used in the comparison are the piggy-bank (third row, third column in figure 2.1) and a toy car (first row third column in figure 2.1). We have used two different methods for comparison between models which are described in sections 4.1 and 4.2. In sections 4.3 to 4.8 the results achieved by applying these two comparison methods on the different models are discussed. In section 4.9 the difference in the performance between the methods is discussed. A short discussion of the effect of using different block sizes appears in section 4.10.

4.1 Error Rates

If we feed the validation set into our models we can calculate the percentage of the labels that are correctly classified. But as mentioned earlier it is not so important to get the background correct as the features. Therefore it can be valuable to distinguish between correctly classified feature labels and correctly classified background labels. With these statistics we can compare different approaches with the same method or different methods with each other.

Sometimes an even more detailed study of the performance is needed. If we have a

$c \times c$ matrix where c is the number of classes and let each row stand for the true class and each column stand for classified class, a correctly classified class 1 data-point would end up in position (1,1) and a class 2 data-point classified as class 3 in position (2,3). If we normalise the entries in each row this gives us a percentage of the true members of a class being classified as a certain class. To make this easier to interpret we can display this, not as numbers, but as squares with size proportional to the value and the result is a confusion matrix. The diagonal represents all the correct classified labels and we can also see which class (feature) that is most easily *confused* with another class and which ones are easy to classify. In figure 4.6 confusion matrices for the performance of the best models for each method are shown.

4.2 ROC Curves

Instead of using a data set with a limited number of labelled input vectors to test the performance of our models, we can feed an entire image into the classifier. For the models using the original space as input we have to cut out a 15-by-15 pixel block around the pixel, transform it into an input vector (as described in section 2.2) and repeat this for every pixel. This gives us an input matrix where each row is an input vector from a pixel. We feed this into the classifier and transform the output back to a matrix, giving us a classification for each pixel. For models using the universal space the procedure is more efficient. We filter the image with the principal components (two-dimensional convolution) and transforms the resulting matrices to column vectors that we places together in a matrix. Each row now consists of the responses from the different filters for a pixel. This matrix is fed into the classifier and the resulting prediction vector is as above transformed to a matrix with the image dimension. This is the same procedure that would be used to do the actual object recognition once the best method to be used have been determined. For every pixel we get a classification (the resulting matrix is a label-map) and a value describing our confidence in the classification (probability-map). We can pick the K pixels from each class in the label-

map that have the highest confidence. For each of these pixels we can calculate the distance to the correct position using Pythagoras theorem. If one of these K pixels is closer to the correct position than a threshold τ, we know that if we allow $K-1$ false alarms we will find the feature position (within τ pixels). The threshold should give us a circle with radius $= \tau$ within which we classify a guess as correct but since we have a discrete number of possible distances due to the use of pixels we end up with a very edgy circle. We can repeat this procedure with all the test images for different K's. For each class/feature we can then calculate the probability (P_f) of finding the feature within the K best guesses . P_f is the fraction of correct predictions over all images containing the feature. So for $K = 1$, P_f is the probability that the top classification in the probability-map is correct (within τ pixels), and for $K = 2$ that one of the two top classifications is correct and so on.

A receiver operating characteristic (ROC) curve is a graph used to present the performance of a system responding to a signal. In (Green and Swets 1966) they are used to plot the probability of correct response to a stimulus against the probability of an incorrect response. With a slight modification we can use them here to plot P_f against K which is good to know when we want to construct our object recognition system. A high P_f for low K obviously means a good feature detector, or at least an easily detectable feature, in this case the ROC curve is close to the top left corner in the plot.

In figure 4.1 some examples of ROC curves are shown. The ROC curve for the 'front right leg' is an example of a good feature detector, with high P_f for low K, and the one for the 'snout' is an example of a not very good feature detector.

4.2.1 Local Maxima

If we look at the above mentioned label-map and probability-map we can have the case that we have all the highest confidence values for a certain class grouped together, due to e.g. the shift invariance. In this case we are "wasting" our guesses when

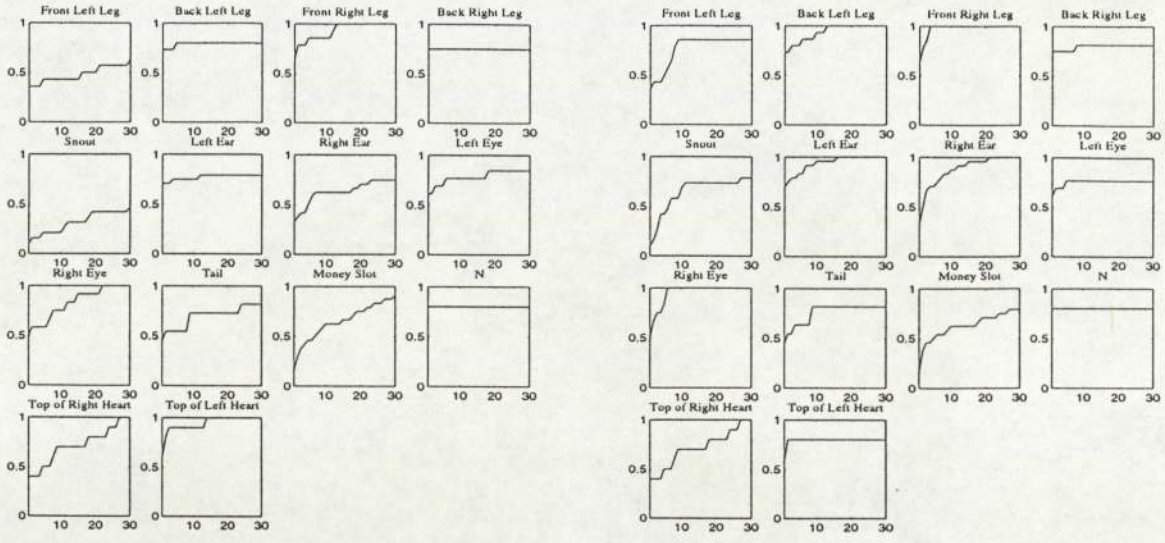

Figure 4.1: The ROC curves for the LSC model with $\tau = 3$. P_k is plotted against K. Piggy-bank test data used. To the left the original probability-map is used and to the right the local maximum probability-map.

we calculate the ROC curves. A method is needed to group these pixels with high confidence together. One solution is to find the local maxima in the probability-map and only use these when looking for the K best guesses. This local maxima probability-map is created in the following way. For each pixel in the probability-map all eight neighbours are checked for a pixel that has a higher confidence and has been classified as the same class in the label-map. If a pixel is found that fulfils these two criteria, the value for the current pixel is set to zero in the probability-map, i.e. we have no confidence in that this pixel is the feature any more. The product of this procedure is a local maxima probability-map with only the local maxima set to non-zero values.

Figure 4.2 shows where a model using LSC in universal space thinks the pigs snout is. The gray areas are all the pixel classified as the snout. In the left half we have the normal probability-map and the crosses are the 20 most likely positions, highest value in the probability-map. The 20:th guess is only one pixel away from the correct position (the ring). In the right half the local maxima probability-map is used. The number of possible locations is much lower and the 4:th guess is the one only one pixel away from the correct position. This is an example when the use of local maxima probability-map

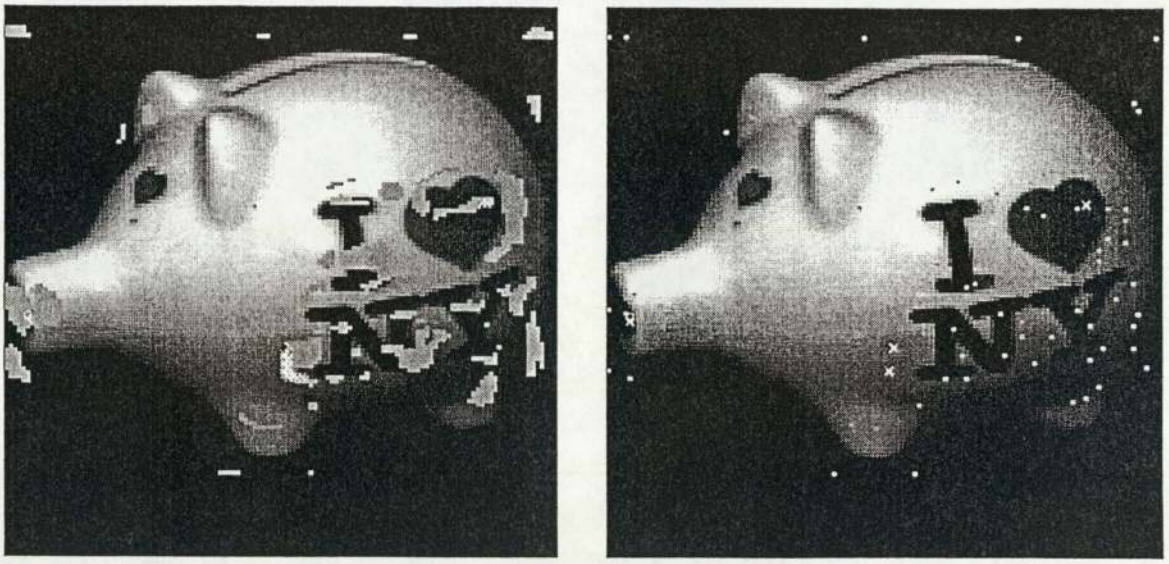

Figure 4.2: An example of the difference between the probability-map (to the left) and the local maxima probability-map (to the right) for the snout. The LSC in universal space is used.

worked very well. If the most likely positions in original probability-map had been more spread out they could all have been local maxima and no improvement would have been achieved. An even worse scenario would be if the guess near the correct position was not a local maximum and was eliminated in the local maxima probability-map, which could lead to worse performance.

4.2.2 Comparison of ROC curves

Plotting ROC curves allows us to compare one model's performance on different features and also different models on the same feature. Using visual inspection it is easy to see which model performs best on a specific feature and which feature that is easiest to classify, but if we want a more objective measure or want to compare model performance over all the features we have some problems. The area under a ROC curve is normally used as a measure when different curves are compared, but normal ROC curves are also plotted as a probability with respect to another probability. This gives us an area that is limited to the range $[0, 1]$. In our case we plot P_f with respect to K where we choose a maximum K (here 30 is used). Having the same K for all our

plots is absolutely necessary to be able to compare the results and we can normalise the areas, divide by K, which gives us the range $[0, 1]$ (the average P_f). But we can still get misleading results from this. Imagine one curve that rises slowly to a high P_f and another curve that rises fast but to a lower P_f. These two could have the same area but very different characteristic, the first one is good at classifying if we allow for a large number of false alarms and the second does fairly good for a small K but does not perform better even if we allow for a higher K. In figure 4.1 we can for instance look at the ROC curve for the 'right eye' which has larger area than the ROC curve for 'N' but for a low K the later is preferred anyway. Since we don't know what value of K is going to be used we will compare the methods based on the areas under the ROC curves but when a choice of method is made for use in a object recognition algorithm and the K is known, inspection of the ROC curves is a better way. We have also the choice of τ, the threshold for when a classification is correct. This depends on the matching strategy used to compare hypothesis of features with a model of the object (see section 5.2). We have used $\tau = 3, 5$ and 7 and the results seems consistent in the sense that the ranking of the methods are independent of the choice of τ.

4.3 Multiple Independent Attributes Classifier

For this method we can use both the original space and the universal space. If we use the universal space we loose information but we get less parameters to optimise and the training procedure takes shorter time, also the classification is faster.

Using down-weighting of the background, as discussed in section 3.2, introduces one new parameter that we could *optimise* over. For the MIA and MLR (section 4.4) methods this is possible but for the neural networks methods in sections 4.7 and 4.8 this would be infeasible. Optimise might also be a too strong word for it. Since we trade better performance on the feature labels against poorer performance on the background labels, there is no *optimal* choice. There is only a choice of how much tradeoff we want. Because of this a value of 0.1 was chosen as weighting for the

background labels compared to 1 for the feature labels. This was done because we add ten background labels to each image and we can have only one feature label of each sort in an image. Using this weighting means that the importance of the background is roughly equal to the importance of the other features.

The results used are the average over ten models.

4.3.1 Error Rates

Since we don't have an output node for the background, instead all the feature output nodes should show zero ideally, we have to find a threshold to decide when an input vector should be classified as a background label. Here we have used the threshold that gives correct classification of 95% of the background labels in the training data.

When the different models were compared using the test data (table 4.1) the one with the original space as input space performed better on the feature labels and the one with the universal space better on the background labels. The model using the original space did not seem to have enough complexity or free parameters to benefit from the importance weighting of training data, the error rate was almost identical. For the model using the universal space a very small improvement on the classification of the feature labels and a worse performance on the background was achieved. The use of committees did not make any difference for this method as there is only one global minimum and our search method can find it. In figure 4.6 the confusion matrix for the best model for this method is shown, which after consulting the error rate for the validation set as well as the test set was the model using the original space without background weighting. The piggy-bank test data is used and the most striking thing about the confusion matrix is that the four legs (features 1-4) are notoriously difficult to distinguish from each other. We have a lot of different misclassifications between the other features too.

% correct Piggy-bank (left) Toy car (right)							
Model		Features		Background		Total	
MIA		71%	61%	60%	72%	64%	67%
Orig. space	Imp. Weight	71%	60%	62%	72%	66%	67%
MIA		50%	51%	87%	75%	70%	65%
Univ. space	Imp. Weight	54%	53%	54%	73%	54%	65%

Table 4.1: Performance for different MIA models

4.3.2 ROC Curves

The model that gave the best performance in section 4.3.1 was used to calculate the ROC curves but the performance was poor. We don't have to worry about the threshold for background classification here since we only look at the pixels with the highest values in the probability-map anyway. 10 out of 14 features on the piggy-bank and all 12 features on the toy car got an increased ROC curve area when local maxima probability-map was used, but still it was not very good. In figure 4.7 we can see the performance on the different features for the MIA in original space model.

4.4 Multiple Logistic Regression Classifier

As we mentioned in section 3.3 we have to use the Universal space for this model, since we don't have enough training data to train the model otherwise. 10 models were trained and the average results were used for comparison.

4.4.1 Error Rates

The performance (shown in table 4.2) turns out to be quite poor for the feature labels and good for the background. We can improve this result by using importance weighting of the training data which results in a better classification of the feature labels, but we have to pay for this with a poorer performance on the background. As for the MIA

model above, use of committees does not improve the performance since there is only one global minimum. In figure 4.6 the confusion matrix for the importance weighted model is shown. Also here the pigs legs are confused, a lot of random misclassifications and quite a few features are classified as background.

% correct Piggy-bank (left) Toy car (right)							
Model		Features		Background		Total	
MLR		48%	39%	92%	88%	71%	68%
	Imp. Weight	61%	62%	69%	72%	65%	68%

Table 4.2: Performance for different MLR models.

4.4.2 ROC Curves

This is the model that gives the worst performance of all. 11 out of 14 features on the piggy-bank and 11 out of 12 features on the toy car got an increased ROC curve area when local maxima probability-map was used, but the performance is still not good. In figure 4.7 we can see the performance on the different features for the MLR with importance weighting model.

4.5 Linear Subspace Classifier

If we use the 95% rule (see section 3.4) to choose the number of dimensions for each subspace, we get between 4 and 10. The average is 5 so we can also use a model with 5-dimensional subspaces for all the features which is easier to handle. We can also project the input vectors into the universal space and then calculate subspaces. This results in loss of information but also in great speed gain.

In top left corner of figure 4.3 we can see the average image of the 'tail' feature in the training data for the piggy-bank. This is subtracted from the input vectors in the training data (the bottom row in figure 4.3 shows three of the training vectors) and

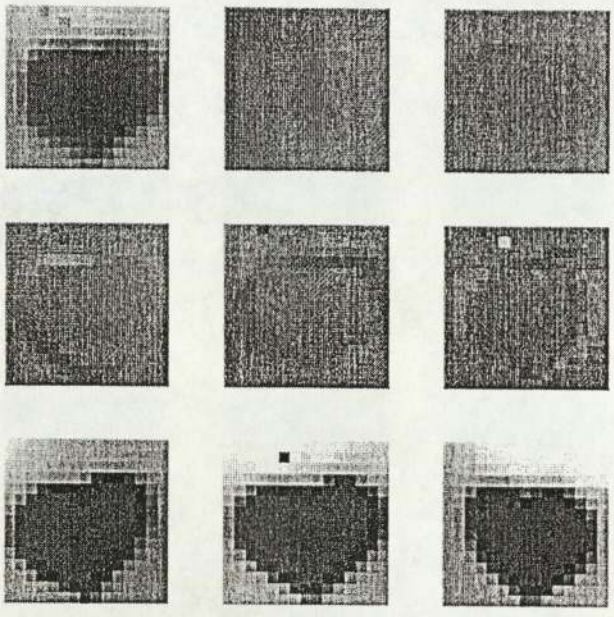

Figure 4.3: The average image for the tail feature, the first five PCs and three example images.

then the principal components are calculated (the first five PCs are shown in figure 4.3). The deviation from the average image can be described by a five dimensional vector (how much of the five PCs the image *contains*). If we multiply this vector with the PCs and add the average image back we end up with an image containing 95% of the information of the original image. As the PCs are adapted to describe the 'tail' images best we lose more information when we use this basis to describe other features. An example can be seen in figure 4.4. The top two images shows a 'tail' vector from the validation set, to the left the original one and to the right the reconstructed image. The error in the reconstructed image is 8 gray-scales/pixel in average. The middle row shows how a 'right eye' image is reconstructed. As we can see the image itself is not that different from a 'tail' image, they are both a heart but at different angles. The reconstructed image to the right has an error of 27 gray-scales/pixel in average. The last row shows a 'front right leg' image which is not very similar to the 'tail' and the reconstruction error for the image to the left is 40 gray-scales/pixel in average and the result is shown to the right. This is the principle behind the LSC method. The LSC in universal space can not be visualised in this way but it works the same way except that

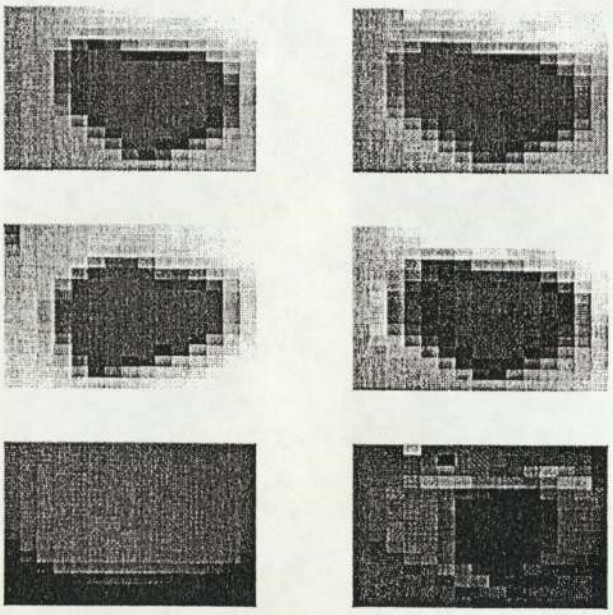

Figure 4.4: Reconstruction using 'tail' PCs for features in the validation set. To the left original feature and to the right the reconstructed.

it starts with compressed versions of the features, i.e. the projections in the universal space.

4.5.1 Error Rates

The model using original space gives a good performance but is computationally intensive. Using the LSC in the universal space gives worse performance but increases the speed significantly. The results using the test set is shown in table 4.3. In figure 4.6 we can see the performance of the model with 5 PC's in the original space. This model has (like all others) difficult to distinguish between the legs, left and right ear and eye (features 6,7 and 8,9) and the heart on the right and left side (features 13,14). A few features are classified as background but we do not have much of the random misclassifications that we had in the methods above.

% correct Piggy-bank (left) Toy car (right)							
Model		Features		Background		Total	
LSC	Orig. Space	83%	73%	94%	92%	89%	85%
	Univ. Space	72%	69%	84%	84%	79%	78%

Table 4.3: Performance for different LSC models

4.5.2 ROC Curves

The performance is very good both when using the original space and LSC in the universal subspace as we can see in figure 4.7. 10 out of 14 features on the piggy-bank and 9 out of 12 features on the toy car got an increased ROC curve area when local maxima probability-map was used for the model using original space. For the model using universal space the numbers were 12 out of 14 features for the piggy-bank and 8 out of 12 features for the toy car. The model using the original space needed approximately 8 times as long time to classify one image. The LSC model in original space is the models that gives the best performance of all.

4.6 K-Nearest-Neighbours Classifier

The two methods suggested in section 3.5 to reduce the amount of calculations needed to make a classification don't work for our problem. The multiedit algorithm discards almost all non-background vectors from the training data. This is probably explained by the fact that approximately 50% of the training data is background labels. For the tree search algorithm it is difficult to cluster the training data due to the high dimensionality. The result is that the entire tree has to be searched anyway and the number of distances needed to be calculated is not reduced.

We have 4 models to compare, combinations of original and universal space with Euclidian or Mahalanobis distance.

4.6.1 Error Rates

If two input vectors are projected into the Universal space the Euclidian distance between the two projections is a measure of similarity between the two vectors. This means that using a Mahalanobis distance in the Universal space would decrease performance and the experiments showed that this also was the case. The best performance, independent of choice of metric, was $K = 1$. This is probably due to the random distribution of background labels. The optimal K was determined by performing leave-one-out cross-validation. This is when a classification is made for each input vector of the training data, when that particular vector has been left out from the training. The percentage of correct classified training vectors where calculated for different values of K and compared. The same K were also optimal when the performance on the validation data was compared.

In table 4.4 the results on the test data is shown. In the original space the models using Euclidian space again performed better. Both models using Euclidian distance were almost equally good when compared over both the test set and validation set. But a drawback with the model using the original space is that this is much more computationally intensive (around 25 times more calculations are needed), since in the higher dimensional space there are many more coordinates to calculate distances over. In figure 4.6 we can see the performance of the 1-NN model in the universal space. Here more features are classified as background, which is quite natural since they are *randomly* spread out and easily can be the nearest neighbour to an input vector. Apart from that the structure of the misclassifications are roughly the same as for the LSC model, mostly between similar features.

4.6.2 ROC Curves

The original space is too computationally expensive to use. By using the universal space we can reduce the number of calculations to 3% but it is still computationally intensive, almost as much as the LSC model in original space. 8 out of 14 features on

% correct Piggy-bank (left) Toy car (right)							
Model		Features		Background		Total	
K-NN	K=1 Euclidian	84%	75%	95%	91%	90%	85%
Orig. space	K=1 Mahalanobis	22%	11%	96%	92%	61%	60%
K-NN	K=1 Euclidian	83%	83%	92%	90%	88%	87%
Univ. space	K=1 Mahalanobis	71%	65%	93%	90%	83%	80%

Table 4.4: Performance for different K-NN models

the piggy-bank and 8 out of 12 features on the toy car got an increased ROC curve area when local maxima probability-map was used. In figure 4.7 we can see the performance on the different features for 1-NN model in universal space.

4.7 Softmax Neural Networks Classifier

The universal space has to be used for this method to reduce the number of free parameters that we have to optimise. The networks were trained using scg for 250 epochs. This can seem high but was chosen to make sure that all nets were trained enough. Visual inspection of the validation error to find a suitable time for early stopping for each net was infeasible, because of the high number of networks trained. This extensive training can lead to over-fitting in the trained networks, which is the same as high variance component in the error, but using committees should reduce this high variance and give us a reasonably good performance.

In order to decide how many hidden nodes that is needed to model the data several different nets were trained. The number of hidden nodes (M) varied from 10 to 15 and 10 nets were trained for each M. The only difference between the training was the weights in the net from the beginning that were generated randomly. For each M the average performance on the validation set was calculated. The M that had the best average performance on the features were chosen as the optimal M unless the

performance on the background was very poor. If the average is plotted against M a peak in the plot is wanted, but this is not always present. Instead the curve can be jagged and if the performance is almost the same for different M, the one with the lowest M was chosen. That is a simpler model with the same performance is preferred. The optimal M for different models are printed in table 4.5. In figure 4.5 is an example which shows the performance of a softmax NN averaged over 10 nets for each value of M. This was the first net to be trained so here M varies from 1 to 20. 14 hidden units were found to be optimal for this model but one might argue that 12 is almost as good.

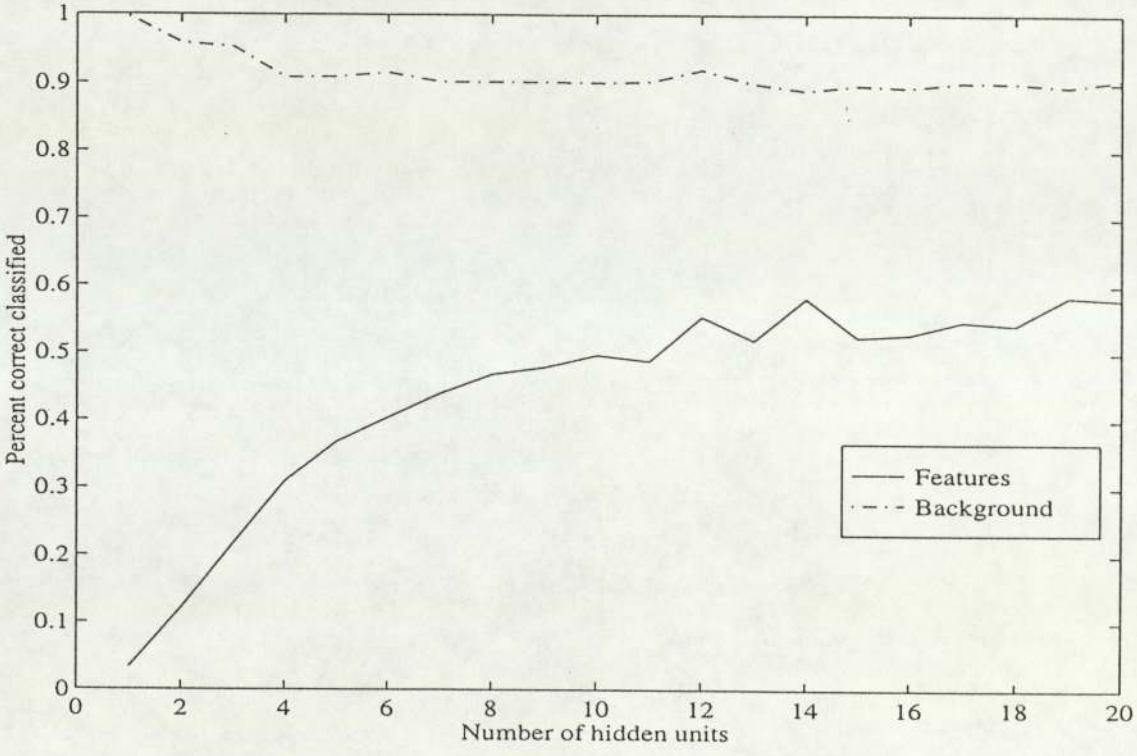

Figure 4.5: The average performance of Softmax NN as a function of M.

4.7.1 Error Rates

Since the error function has a number of local minima where the search algorithm can get stuck and we can have slightly over-fitted networks, committees can be used to improve performance. As we can see in table 4.5 committees increased the performance

with around 10 percentage units. The method of down-weighting of the background labels also improves the performance for the feature labels but as usual we have to pay for this with worse performance on the background labels. Using shift invariance adds a little bit to the performance as well. In figure 4.6 we can see the confusion matrix for the best version, a softmax model with shift invariance, 15 hidden units and a committee of ten nets. Many misclassifications are features classified as background, but we also have the usual ones between legs, ears and eyes.

% correct Piggy-bank (left) Toy car (right)							
Model		Features		Background		Total	
Softmax NN	14/13 hn	61%	53%	90%	87%	76%	73%
	Com. 12/11 hn	70%	66%	95%	93%	83%	82%
Softmax NN	13/11 hn	68%	56%	90%	89%	80%	76%
Shifted	Com. 15/15 hn	80%	80%	95%	92%	88%	87%
Softmax NN Imp.	14/12 hn	66%	63%	76%	79%	72%	72%
	Com. 12/12 hn	75%	75%	78%	82%	77%	80%
Softmax NN Imp.	12/14 hn	73%	72%	75%	81%	74%	77%
Shifted	Com. 11/14 hn	80%	84%	76%	84%	78%	84%

Table 4.5: Performance for different Softmax NN models. hn = hidden nodes. Ex. 14/13 hn = 14 hidden nodes for the piggy-bank model and 13 hidden nodes for the toy car model.

4.7.2 ROC Curves

10 out of 14 features on the piggy-bank and 8 out of 12 features on the toy car got an increased ROC curve area when using local maxima probability-map. This method takes only half the time compared to LSC and is faster that the K-NN method. The performance can be compared in figure 4.7 where the softmax model with 11 hidden units, importance weighting, shift invariance and 10 nets committee is used.

4.8 Multiple Independent Attributes Neural Networks Classifier

Also for this method we have to use universal space to be able to optimise the parameters. The same method as for softmax in section 4.7 was used to train the networks and to find the optimal M.

4.8.1 Error Rates

Committees also here improved the result for the same reason as for the softmax method above. The MIA neural network does not have a special background output, but a low probability for the other classes is interpreted as implying that a background label is the input vector. As for the MIA method (in section 4.3) we have to find a threshold for the background labels and we used the same rule, i.e. that 95% of the background labels in the training data were correctly classified. This also means that down-weighting of the background labels only makes it harder to find the correct threshold, since the range of the outputs are not *stretched* as much. Because of this we could not see a clear improvement of the performance when weighting was used. In table 4.6 the performance for the different models are given. In figure 4.6 we can see the performance of a MIA NN model with shift invariance, 14 hidden units and a committee of ten nets. Apart from the easily confusable features which are the same as for all the other models a lot of the misclassification comes from background being classified as features, which is not that serious.

4.8.2 ROC Curves

11 out of 14 features on the piggy-bank data and 9 out of 12 features on the toy car data got an increased ROC curve area when using local maxima probability-map. The speed is the same as for the softmax method. In figure 4.7 the performance is shown.

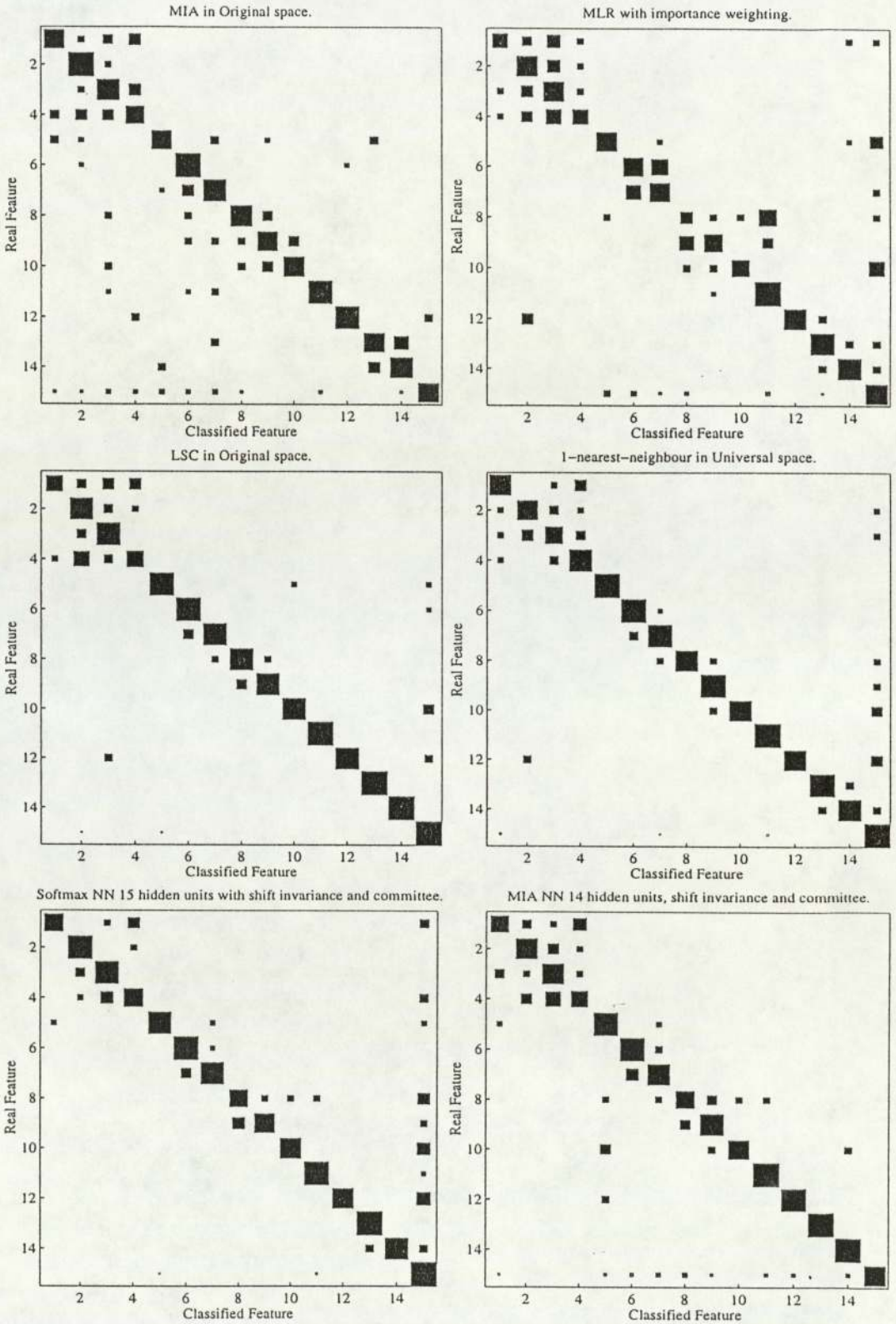

Figure 4.6: The confusion matrices of the best models for each method. Test performed on the piggy-bank test data.

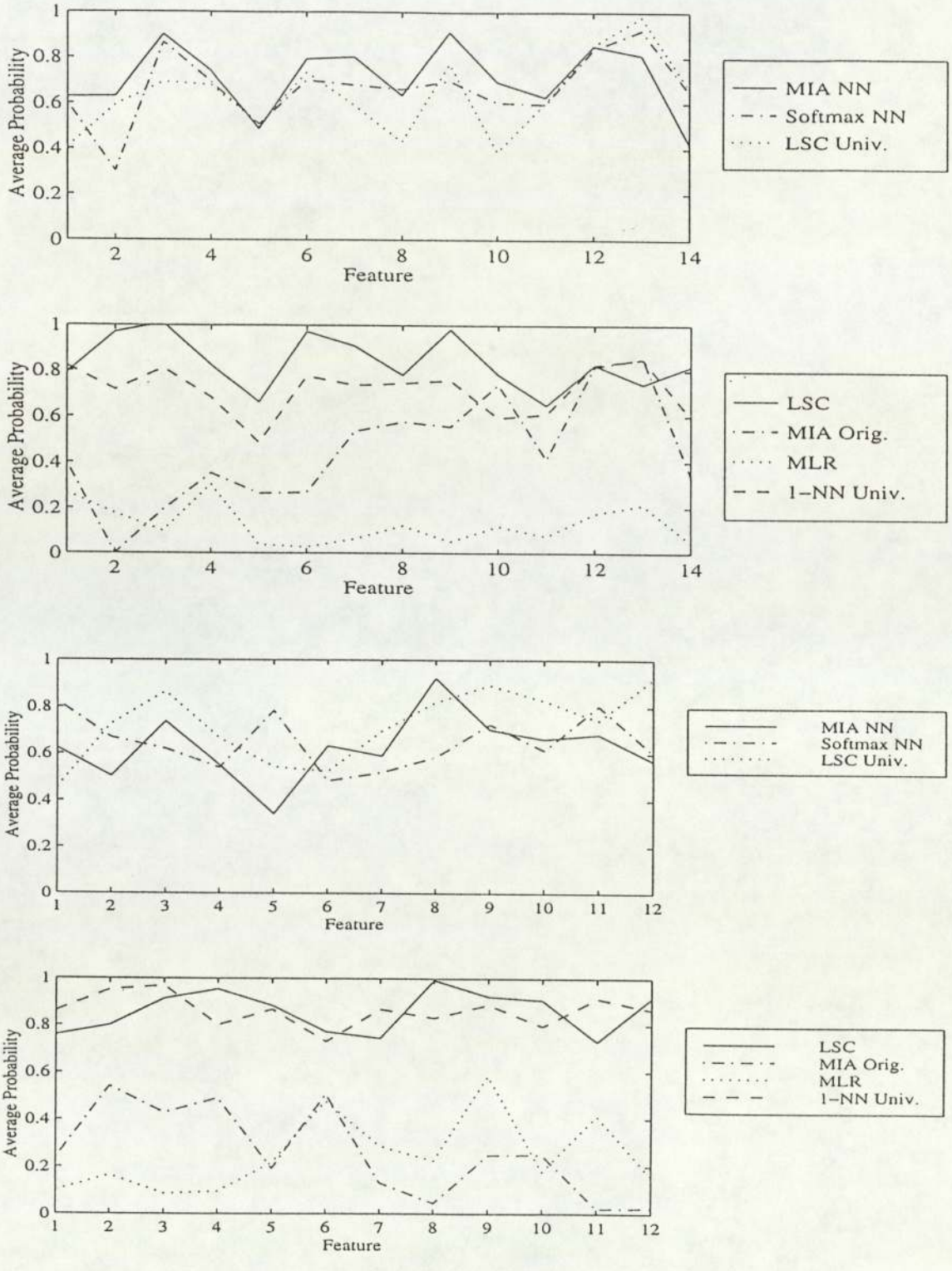

Figure 4.7: The normalised areas under ROC curves for different models. This is equal to the average probability over K that the correct feature position is found within $K-1$ false alarms. Local maxima probability-map and $\tau = 3$. In the top two plots Piggy-bank test data is used and in the bottom two the toy car test data. Feature numbers can be found in table 2.1 and 2.2.

% correct Piggy-bank (left) Toy car (right)							
Model		Features		Background		Total	
MIA NN	10/13 hn	54%	53%	62%	58%	58%	56%
	Com. 11/13 hn	71%	67%	74%	81%	72%	75%
MIA NN	14/13 hn	66%	72%	58%	69%	62%	70%
Shifted	Com. 14/14 hn	75%	75%	72%	80%	74%	78%
MIA NN Imp.	13/14 hn	51%	59%	84%	48%	68%	53%
	Com. 10/14 hn	60%	70%	90%	73%	76%	72%
MIA NN Imp.	13/13 hn	64%	62%	57%	56%	60%	58%
Shifted	Com. 14/11 hn	77%	74%	71%	76%	74%	75%

Table 4.6: Performance for different MIA NN models. hn = hidden nodes. Ex. 14/13 hn = 14 hidden nodes for the piggy-bank model and 13 hidden nodes for the toy car model.

4.9 Comparison between the methods

There are many aspects to be considered when the methods are compared. Not surprisingly we don't have one method that is better than all the other in every aspect and depending on the intended use, different methods can be the best suited. In figure 4.8 the percentage of correct classification for each method are shown as a staple diagram. According to this diagram 1-nearest-neighbour should be the best classifier, if we use error rates as criteria. In table 4.7 the time needed to classify one 128 by 128 pixel image is given for the different methods which also is an important aspect. We can try to summarise the pros and cons for the different methods.

- The MIA method using the original space is not very computationally intensive but also not very good. The performance is not good enough to use in an object recognition algorithm.

- The MLR method is very fast but the performance is poor and not good enough to use in an object recognition algorithm.

- The LSC method is computationally intensive when the original space is used but using the universal space makes it three times as fast as the neural network methods. The performance when ROC curves are used for evaluation is very good. The LSC in original space is the best and the one in universal space is also good.

- The K-nearest-neighbours method performed well when assessed with the error-rate and is among the best when ROC curves are used for evaluation. It is computationally intensive even using the universal space.

- The softmax method is in the middle when it comes to computational expense and also performance.

- The MIA network is in the middle when it comes to computational expense but the performance is very good.

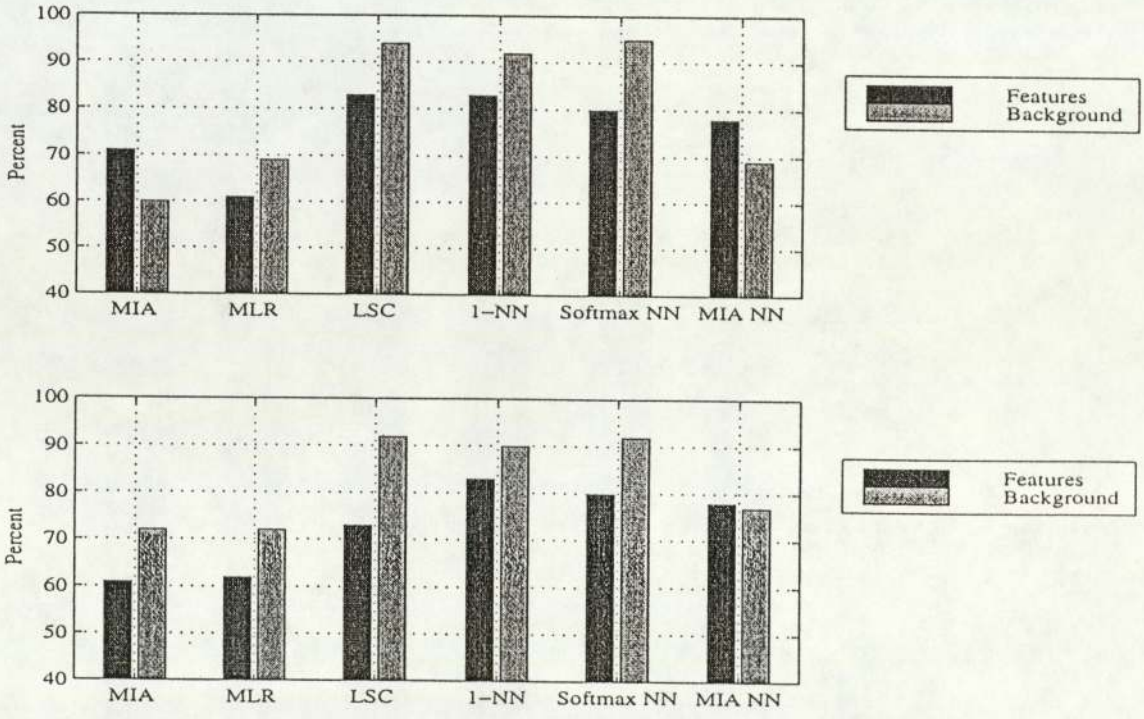

Figure 4.8: Percent correct classified for the best model for each method. Top diagram uses the piggy-bank data and the bottom one uses the toy car data.

Model	CPU time / image
MIA orig. space	17.4 s
MLR	3.6 s
LSC	100.2 s
LSC univ. space	13.7 s
1-NN univ. space	72.5 s
Softmax NN Com.	54.1 s
MIA NN Com.	50.0 s

Table 4.7: CPU time needed to label one image on a 200MHz R10000 cpu.

Another way to plot the average probabilities for ROC curves is to order them in descending order which makes it easier to compare them (figure 4.9). According to this plot the LSC model is the best one, this can be compared to figure 4.8 above. This can also be used to study the effect of not using all the features for recognition of an object. If we can choose to exclude some of the features from a model of an object without losing the ability to recognise it we could achieve better performance. If we exclude features that are hard to recognise we both reduce the number of false alarms and makes the search among hypothesis easier. This is discussed further in section 5.2. Features on the piggy-bank that are hard to recognise are not the same for all methods but some that most of the models have difficulties to find are the 'snout', the 'money slot' and the 'tail'. In Appendix B the ROC curves for the best models for each method is shown for different values of τ.

So which method is the best? If the time to do a classification is not an issue, then obviously the LSC in original space is the best choice. This method has the best performance when the ROC curves are used for evaluation and it is quite easy to implement. The only drawback with this method is that it is slow, so if a fast classification is needed other choices are better. The MIA NN method is twice as fast as the LSC classifier when committees are used. The performance is still good and

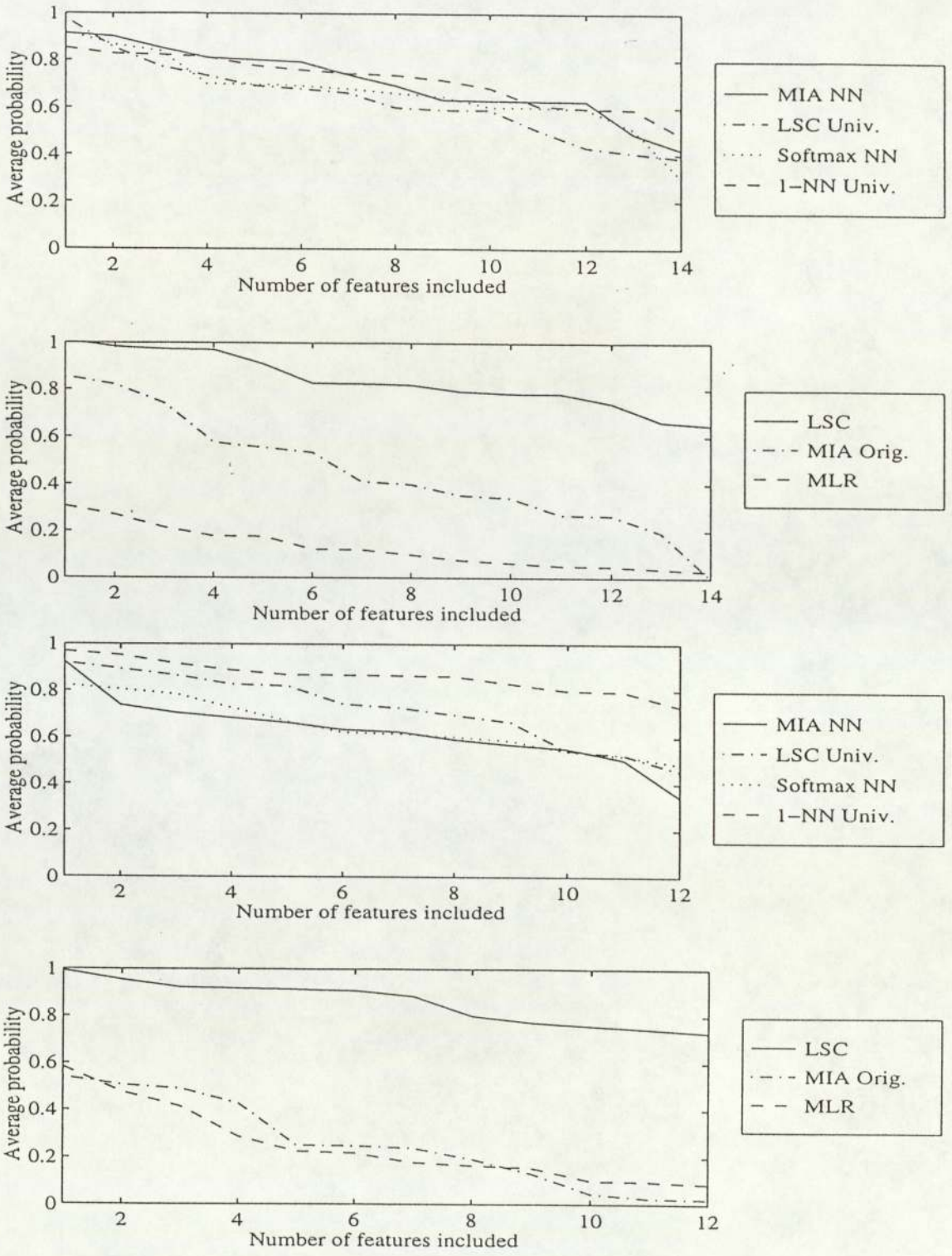

Figure 4.9: The normalised areas under ROC curves for different models sorted in decreasing order. Piggy-bank test data used in upper two plots and toy car data in the two lower plots.

the only drawback is that the training of the networks is time consuming and several different sizes have to be trained and compared. The LSC in universal space is almost four times as fast as MIA NN method and the performance is almost as good. This makes it a good candidate. For the LSC in original space feature detector, 11 out of 14 feature detectors had over 80% probability of finding the correct feature position when the parameters $K = 10$ and $\tau = 5$ were used. The average probability was 86.5%.

4.10 Block-sizes

The effect of using different block sizes when the input vectors are constructed from the original image has not been tested for all methods. The different block-sizes used were 9×9, 11×11, 13×13, 15×15, 17×17, 19×19 and 21×21. When the LSC method in original space was used the performance got better and better the larger block size that was used when the comparison was made using the error rates. In figure 4.10 the confusion matrices for the models using the smallest and largest block size are shown. The percentage of correctly classified features spanned from 68% for the 9×9 model to 83% for the 21×21 model. But when the ROC curves were calculated the smallest block size had 6-7% worse performance than the largest block size and the number of calculations needed was only 20% of the numbers needed for the largest size. Figure 4.11 shows three of the ROC curve areas. This means that if we want faster recognition we might be able to use LSC feature detectors but with smaller blocksize.

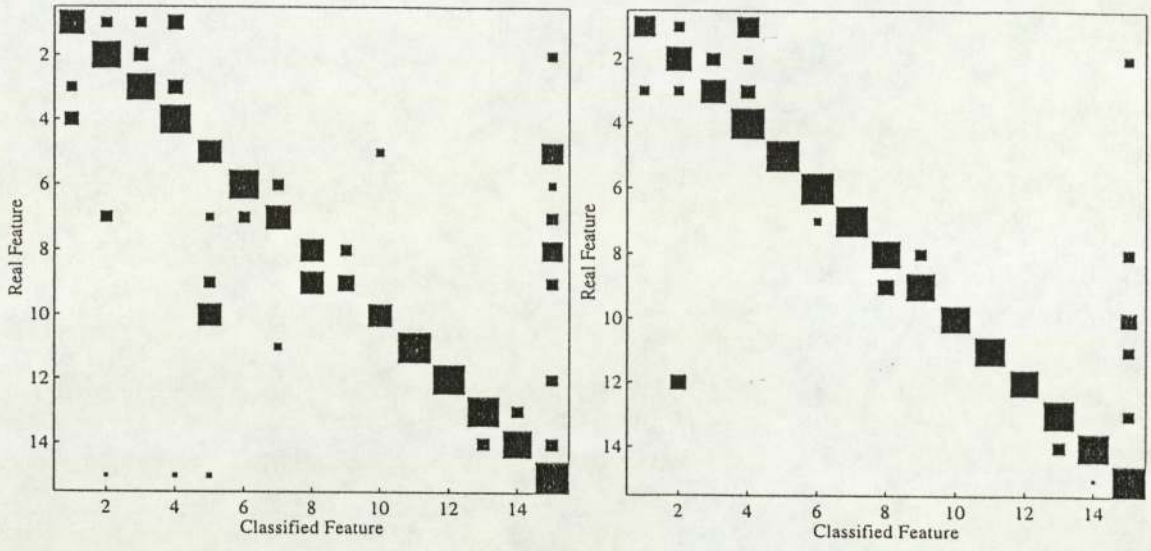

Figure 4.10: The confusion matrices for LSC models using block sizes 9×9 to the left and 21×21 to the right.

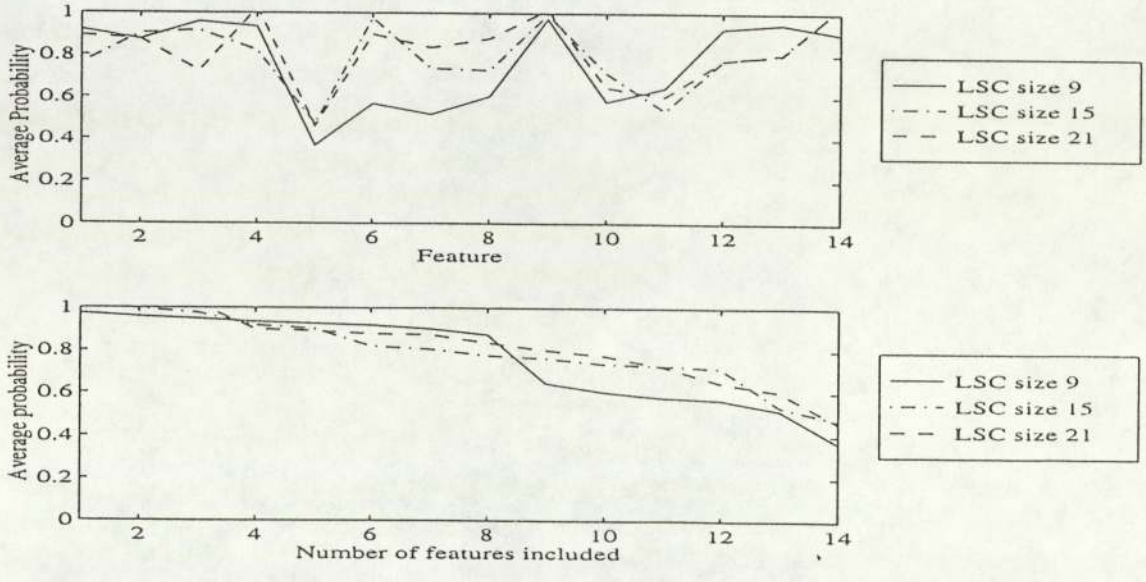

Figure 4.11: The ROC curve areas for LSC models using block sizes 9×9, 15×15 and 21×21, in the lower plot the areas for the features are in descending order.

Chapter 5

Discussion

Section 5.1 summarises the content of the thesis. In section 5.2 some possible directions for future work is given.

5.1 Summary

In this thesis we have examined several different feature detectors for use in a recognition by parts algorithm for three dimensional rigid objects. For two different objects, features have been chosen and labelled in images showing the object from different angles. Out of these labelled images $m \times m$ pixel wide blocks have been cut out and turned into training, validation and test data. One kernel method, one linear method, two generalised linear methods and two non-linear methods was used to construct feature detectors based on the training data. These feature detectors have been compared by error rates on the test data and also on how well they detect features in the test images. The later was done by constructing ROC curves that plots the probability of predicting the correct position for the feature as a function of the number of false alarms. The speed of classification for the different methods have also been taken into account.

5.1.1 Main contributions

- A label tool for labelling images has been built.
- Functions to generate training data for classifiers from the labelled images have been coded.
- K-nearest neighbour, LSC, MIA and MLR classifiers have been coded.
- After the tests performed in this thesis the LSC in original space can be recommended as the classifier best suited for use in a recognition by parts algorithm, if the time for classification is not critical. If the time for classification is a constraint the LSC in universal space or MIA neural networks are the classifiers best suited for use in a recognition by parts algorithm.

The first three items are addressed in appendix A and the last one in chapter 4

5.2 Possible direction for future work

The first thing to do is to build a recognition by parts system that is using LSC in original space. The LSC in original space should be used because speed is not important at first, only performance. To be able to recognise an object we have to have some sort of model of the object, a geometric model is probably the first try, and use this to do correspondence matching. Given the performance of the LSC feature detectors presented in chapter 4 we have a few problems that need to be solved. Allowing a high number of false alarms for every feature detector results in a higher probability to detect the true feature, but it also gives us a larger correspondence search space. To reduce the number we have to use constrains. For instance we can use something similar to the conditional search approach used in (Burl and Perona 1996). This approach uses the fact that given the position of two features, the possible position of all the other features are highly constrained and a lot of constellations can be excluded from the search. Here we can allow for an uncertainty for the position and use different

values for τ. Since we are viewing three dimensional objects from different angles we also know that hypothesis containing features from opposite sides of the object are not possible, i.e. we can not have an image of the piggy-bank that contains both the heart on the right side and the heart on the left side of the pig, so these can also be excluded. Another way to reduce the number of constellations is to not use all of the features. If the features that are most difficult to recognise can be excluded from our model of the object the number of false constellations as well as the number of possible constellations would decrease. We must also make it possible to recognise an object with missing features. There are two reasons for this. First of all because we want robustness against occlusion and secondly because even for high K we don't have a 100% probability of finding all features. Experiments with a system like this would reveal a suitable K to use to get good enough performance without having too large correspondence search space.

Instead of using the simple four-way shift invariance described in section 2.2, we can use more complex shift invariance like eight-way, several pixels shift or rotational invariance to give us more training examples and a more robust performance.

We have used the first few principal components to filter the images, which gives us an optimal linear dimension reduction. Instead of using a linear method a non-linear method such as auto-associative net (Bishop 1995), could be used. The reason we have not tried this is because of the number of free parameters that have to be optimised. We don't have enough training examples to do this. If, however, invariance is used to artificially create more training examples or a smaller block size is used , we might have enough training data to train an auto-associative net.

More training examples would also make it possible to try to use factor analysis (FA) (Everitt 1984). Factor analysis is a latent variable model and can, like PCA, be used to do dimensionality reduction. As mentioned above, PCA gives the optimal linear dimension reduction so using FA for this is pointless. A more interesting use would be to construct the linear subspaces used in LSC. FA tries to determine if a

set of observed variables can be explained by a small number of uncorrelated latent variables. Each observed variable is a function of the latent variables plus a residual term. To find the latent variables the covariance matrix that the model give rise to is fitted as good as possible to the covariance matrix of the observed variables. If we have less examples than the dimension of the examples , as in our case, this is not possible without adding additional constraints. If we could determine the latent variables for a feature we know that all the training examples of this feature lies spread around the subspace spanned by the latent variables at a distance of the residual terms. We could use this information to make a classifier.

Bibliography

Bishop, C. M. (1995). *Neural Networks and Pattern Recognition.* New York: Oxford University Press.

Burl, M. and P. Perona (1996, June). Recognition of planar object classes. In *Proceedings of the 1996 IEEE Computer Society Conference on Computer Vision and Pattern Recognition.*

Cootes, T. and C. Taylor (1996). Locating objects of varying shape using statistical feature detectors. In *Proceedings from the 4:th European Conference on Computer Vision 1996*, pp. 465–474. Spring-Verlag.

Everitt, B. (1984). *An Introduction to Latent Variable Models.* Chapman and Hall.

Fukunaga, K. and P. Narendra (1975). A branch and bound algorithm for computing k-nearest neighbors. *IEEE Transaction on Computers* (24), 750–753.

Green, D. and J. Swets (1966). *Signal detection theory and psychophysics.* John Wiley and Sons, INC.

Grimson, W. (1990). *Object Recognition by Computer.* Cambridge, Massachusetts: The MIT Press.

Hinton, G. E., M. Revow, and P. Dayan (1994). Recognizing handwritten digits using mixtures of linear models. In *Advances in Neural Information Processing System*, Volume 7, pp. 1015–1022.

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

Nene, S. A., S. K. Nayar, and H. Murase (1996). Columbia object image library (coil-20). Technical report, Department of Computer Science.

Ripley, B. D. (1996). *Pattern Recognition and Neural Networks.* Cambridge University Press.

Turk, M. and A. Pentland (1991). Eigenfaces for recognition. *Journal of Cognitive Neuroscience 3*(1), 71–86.

Appendix A

Software

All the experiments in this thesis were implemented in matlab (The MathWorks, Inc.). The two neural network methods were implemented using Netlab, a tool-box for matlab developed by Prof. Chris Bishop and Dr. Ian Nabney at Aston University. This tool-box also contains the scg search algorithm. The rest of the tools are described in this appendix and has been implemented as part of this thesis.

A.1 Label-tool

The label-tool is the interactive matlab program that is used to label images. In Figure A.1 the menu for the label-tool is shown, in the following description a number in parenthesis is a reference to the corresponding number in this figure. The label-tool is intended for use with the COIL-20 database described in Chapter 2, which contains of twenty objects with 72 images for every object. In the text field (2) the name of the image to be labelled should be typed. The format of the name is 'objx_n' where x is the the object number (between 1 and 20) and n is the image number (between 0 nd 71). When this has been confirmed by pressing Return, the image can be loaded by pressing the *Load* button (1). The image is loaded and displayed in the image-window. If there exists a label-file for this image this is loaded. The label-file is a file containing the positions of the labels for a specific image and the feature number for the label.

The labels are displayed as crosses on the image (see Figure 2.2 or 2.3). All objects need a label-list file that contains the feature names for the object. If one is already created it is loaded together with the image, otherwise one has to be created before labelling is possible. To create or add a feature name to the label-list, *New* should be chosen in the label-list pop-up menu (6). A dialog box comes up where the name of the feature should be entered. This name will now show up in the label-list pop-up menu. By selecting *New* in the label-list pop-up menu repeatedly one can add all the feature names that should be associated with this object. Pressing the *Labellist save* button (4) save this list. To place a label on the image the wanted label should be chosen from the label-list pop-up menu and then the *Label* button (5) should be pressed. The left mouse button places the label, the middle mouse button removes a label and the right mouse button displays information about a label. When all the features in the image have been labelled one can save the resulting label-file by pressing the *Save* button (3). The *Update* button (8) can be used to refresh the image. The *Help* button (9) displays a short help text. Pressing the *Quit* button (7) exits the program.

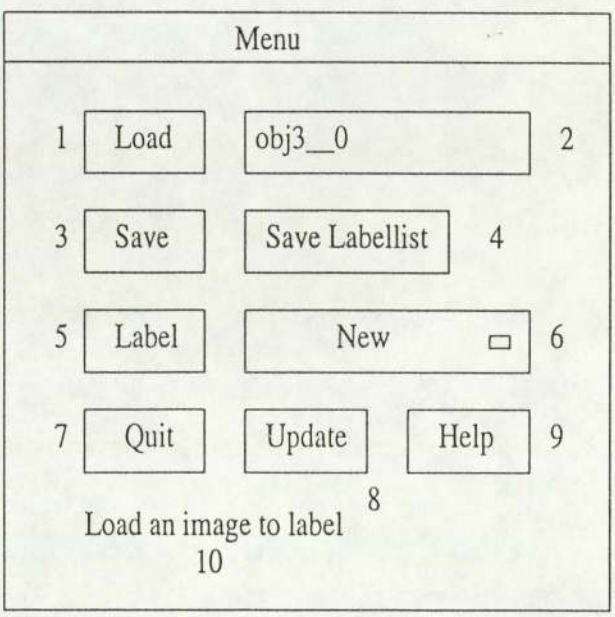

Figure A.1: The user interface for the label tool.

A.2 From Label-list to Training Data

We now have a label-file for every image. In Section 2.2 the addition of background labels is mentioned, this is done by using the function *randlab* that takes a file name and a distance d as arguments and adds 10 random labels to the label-file that are at least d pixels away from the other labels. The function *label2data* cuts out $m \times m$ pixels around the label positions and creates input vectors and output vectors and saves this to a .nnd file.

A.3 Preprocessing and Classifiers.

Sending in all the training input vectors to the *pca* function creates the projection matrix for the universal space. Sending in the input vectors for one feature at the time produces the projection matrices for the LSC classifier. The functions *ml*, *mlerr*, *mlgrad* and *mlfwd* are used to construct and make classifications with the MIA and MLR models. The *knn* function are used to make classifications using the K-nearest-neighbour models.

A.4 Viewing the results.

To calculate error rates and make confusion matrices the function *confmat* is used. The function *rocmatrix* constructs ROC curves. To view the result of classification on different images *displaytest* is used. This tool lets you choose the image to view, which feature to look at and which classifier output to use. It views all the areas that have been predicted as the specified feature and the top p guesses, where p can be specified. To create the local maxima probability-map the function *locmax* is used.

Appendix B

ROC Curves

Here the ROC curves for the best models for each method are displayed. To the left the piggy-bank test data is used and to the right the toy car test data. The local maxima probability-map is used. For an explanation of the ROC curves see section 4.2.

B.1 Plots of ROC curves using threshold $\tau = 3$

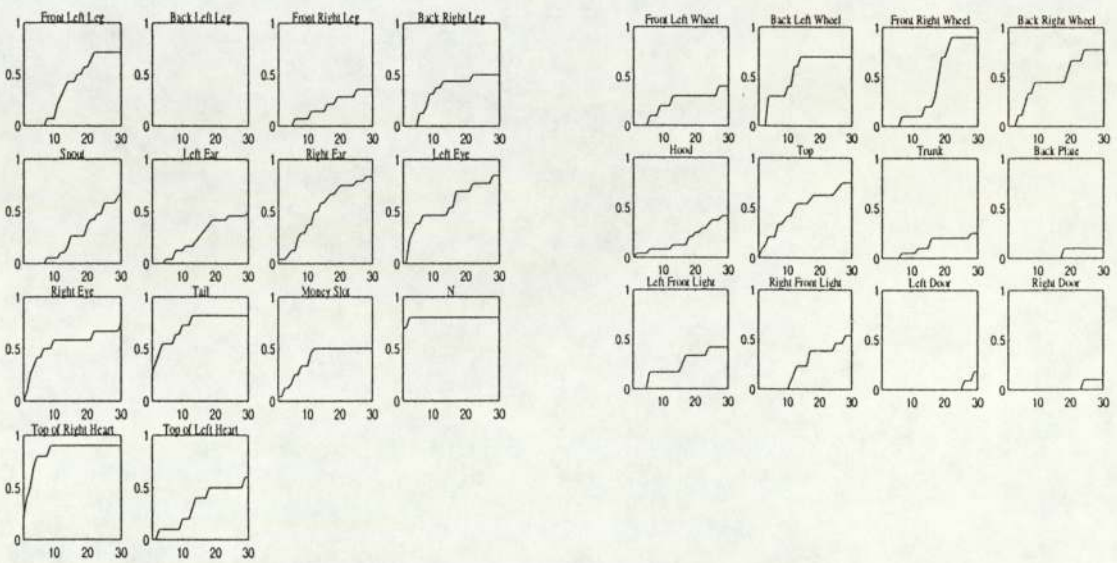

Figure B.1: ROC curves for the MIA model in original space.

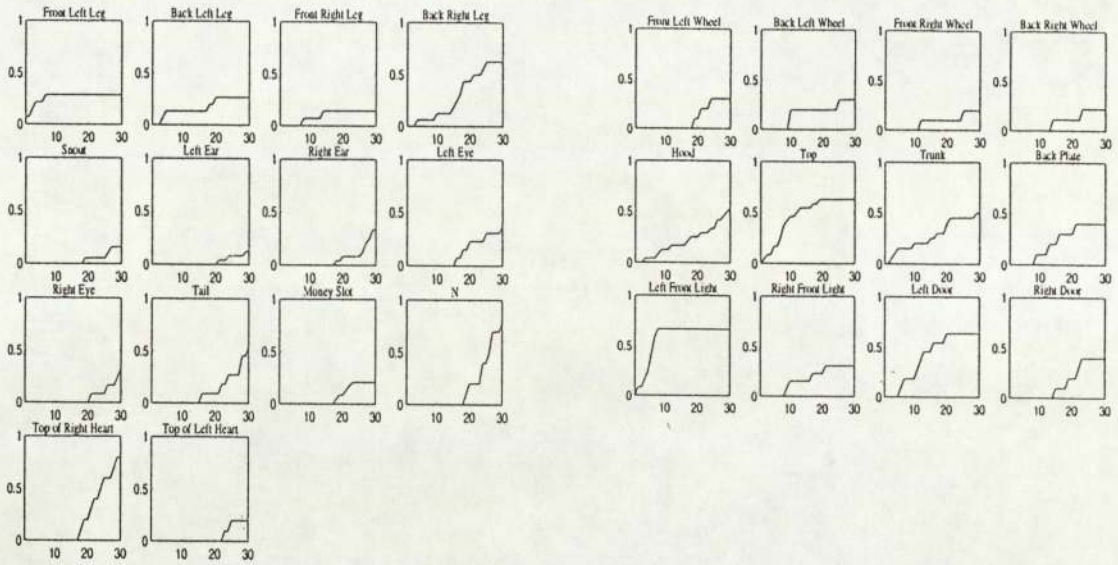

Figure B.2: ROC curves for the MLR model.

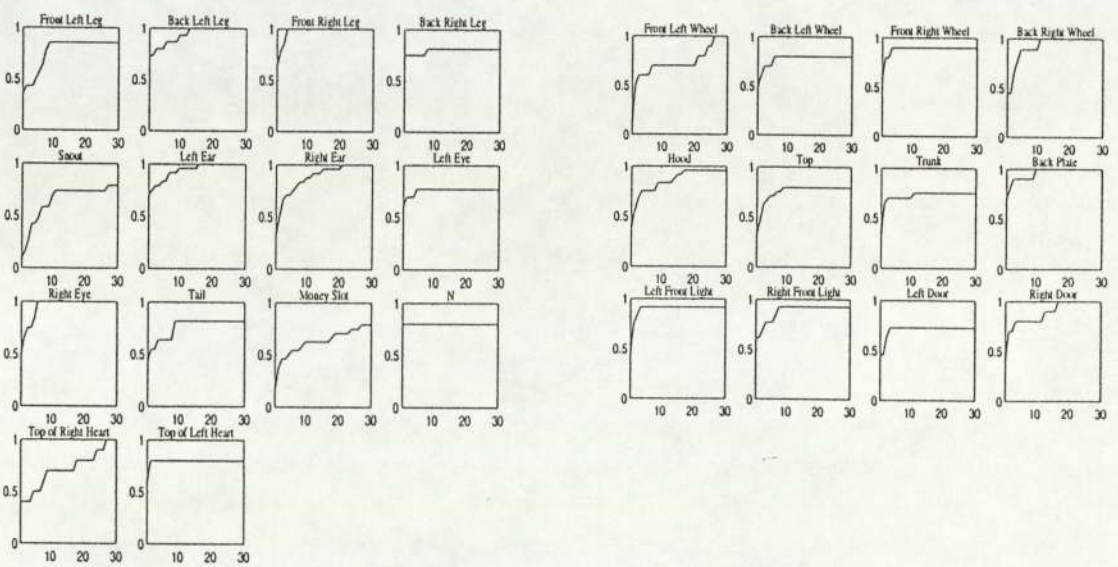

Figure B.3: ROC curves for the LSC in original space.

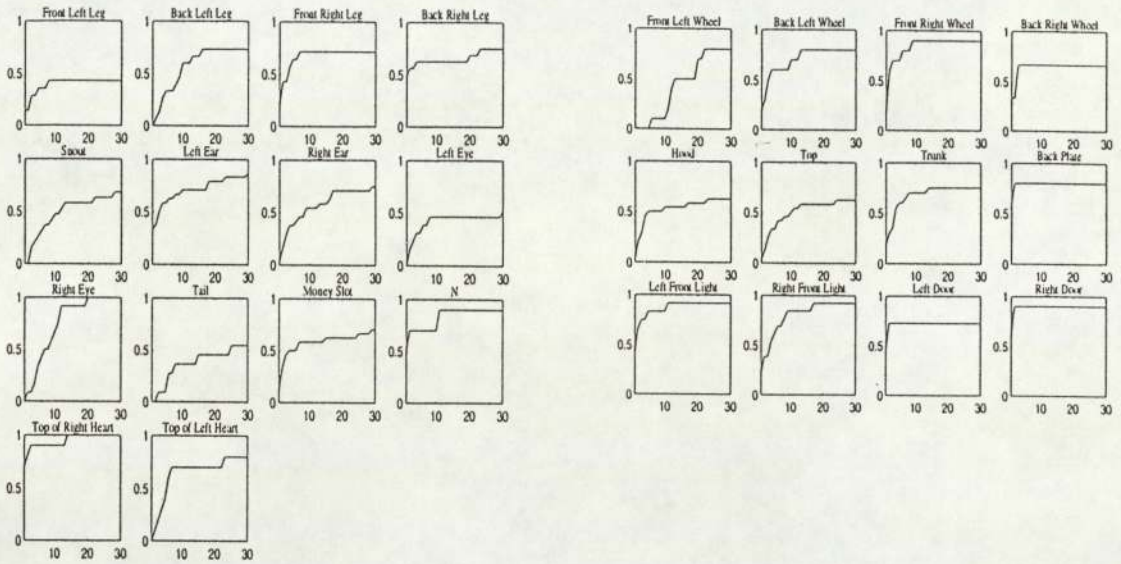

Figure B.4: ROC curves for the LSC model in universal space.

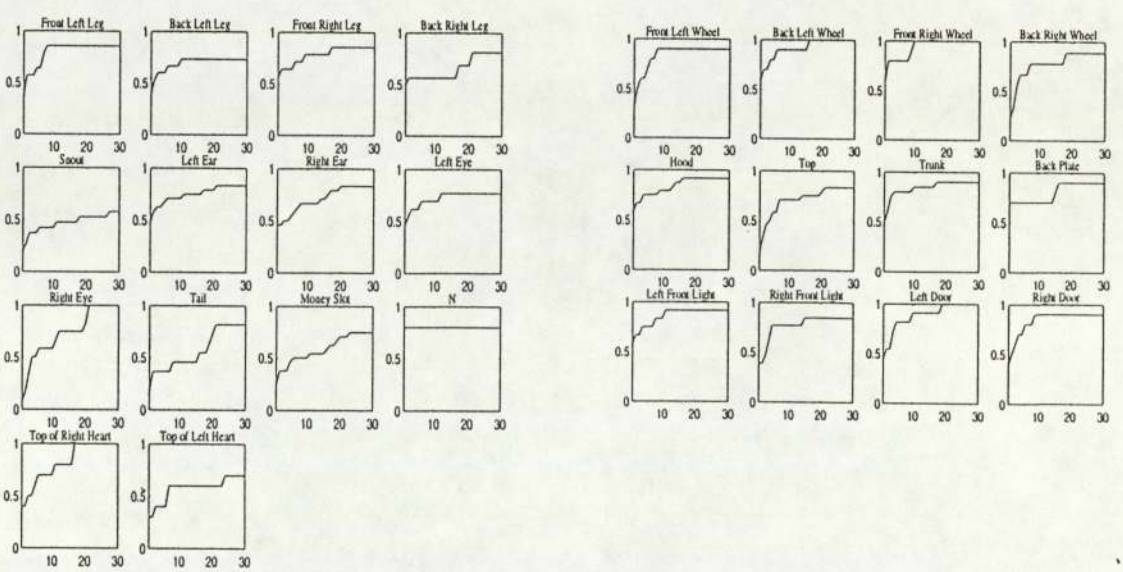

Figure B.5: ROC curves for the 1-nearest-neighbours in universal space.

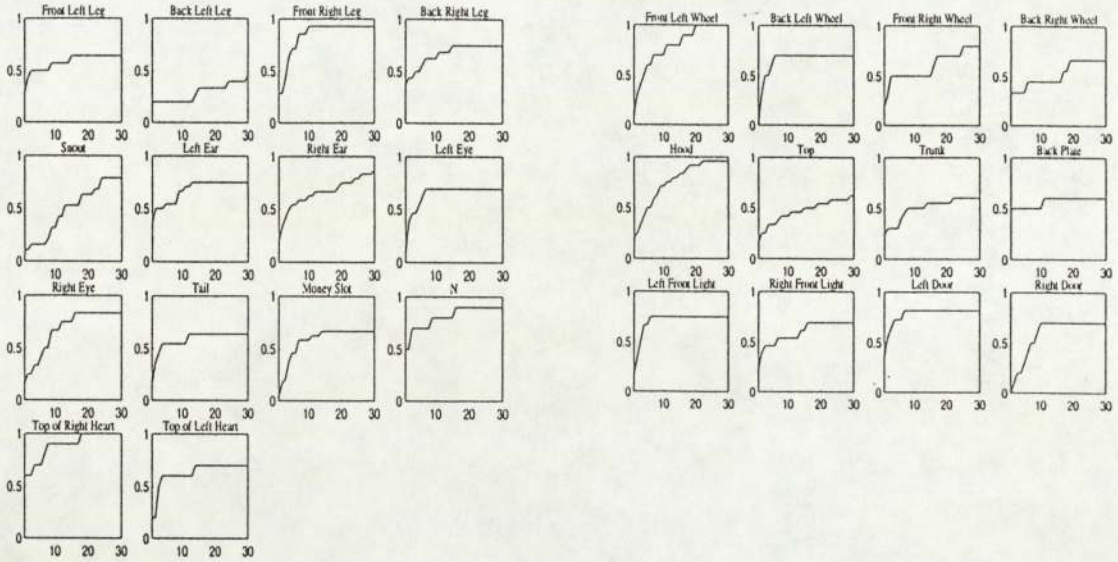

Figure B.6: ROC curves for the softmax NN with importance weighting, shift invariance and committees.

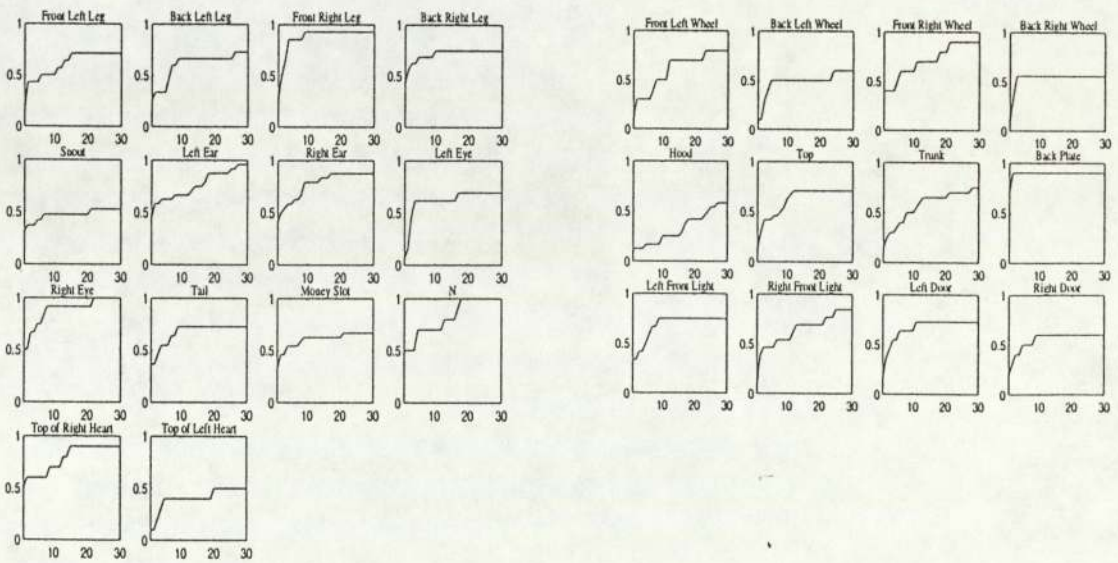

Figure B.7: ROC curves for MIA NN using shift invariance and committees.

B.2 Plots of ROC curves using threshold $\tau = 5$

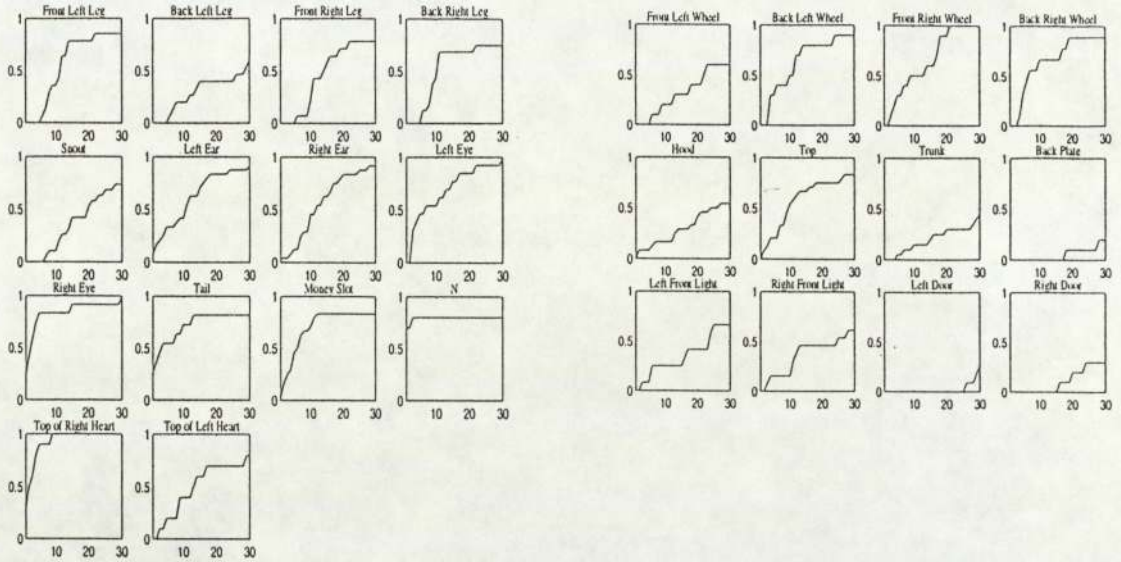

Figure B.8: ROC curves for the MIA model in original space.

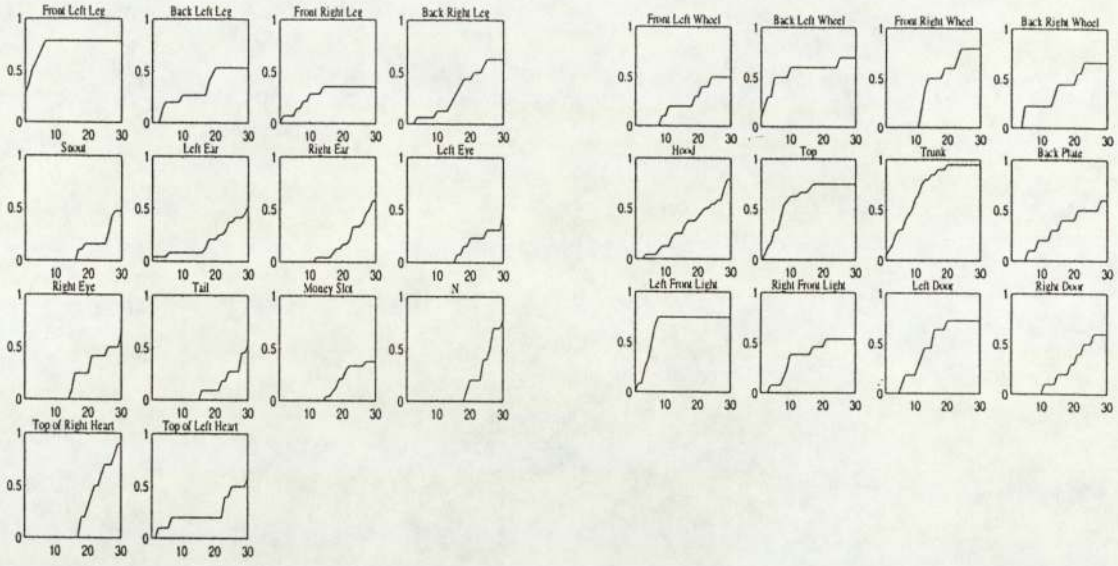

Figure B.9: ROC curves for the MLR model.

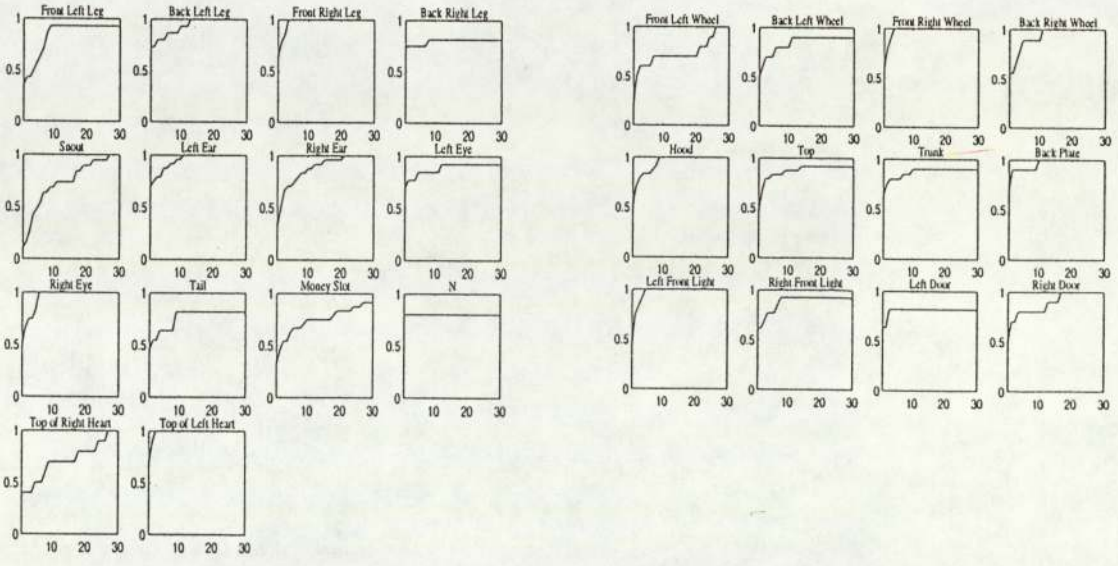

Figure B.10: ROC curves for the LSC in original space.

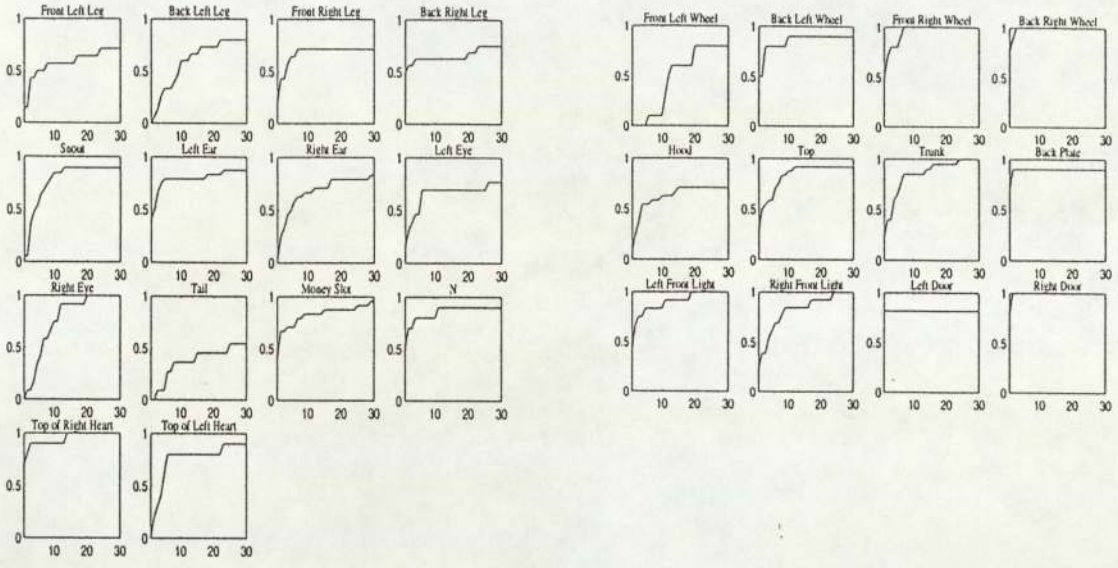

Figure B.11: ROC curves for the LSC model in universal space.

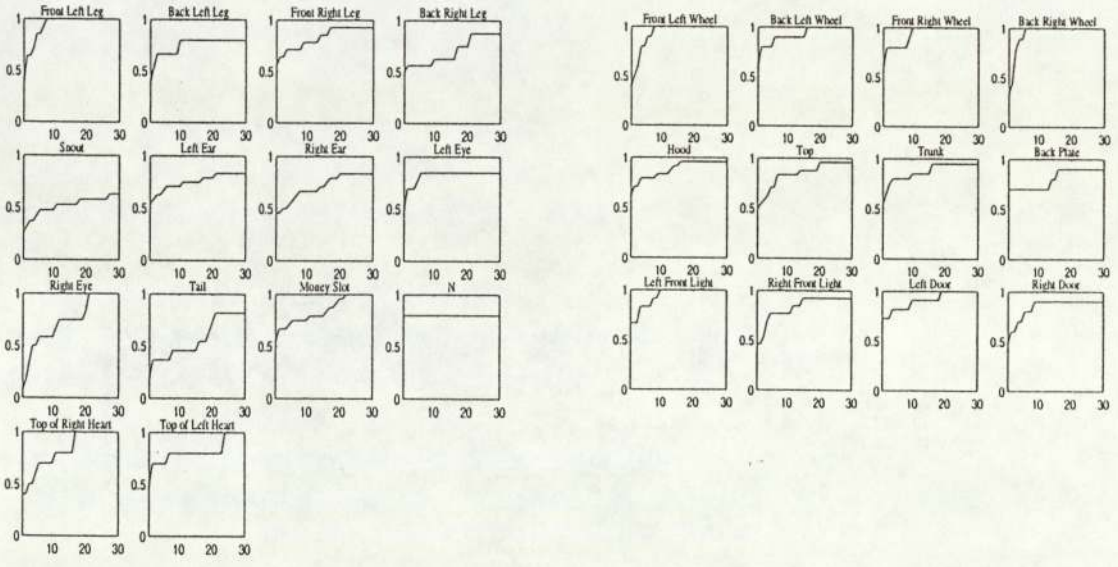

Figure B.12: ROC curves for the 1-nearest-neighbours in universal space.

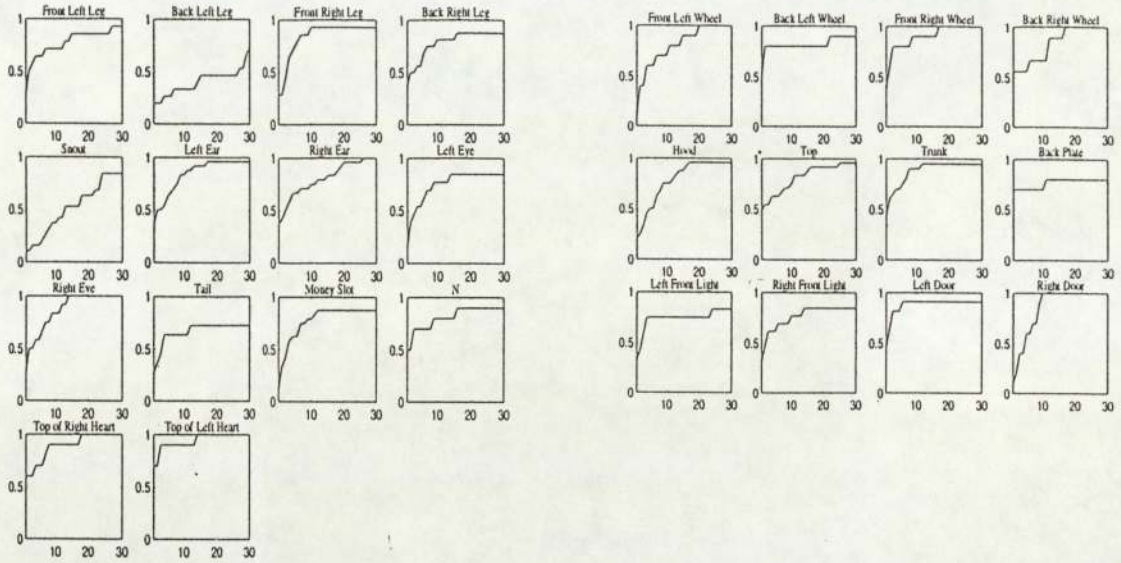

Figure B.13: ROC curves for the softmax NN with importance weighting, shift invariance and committes.

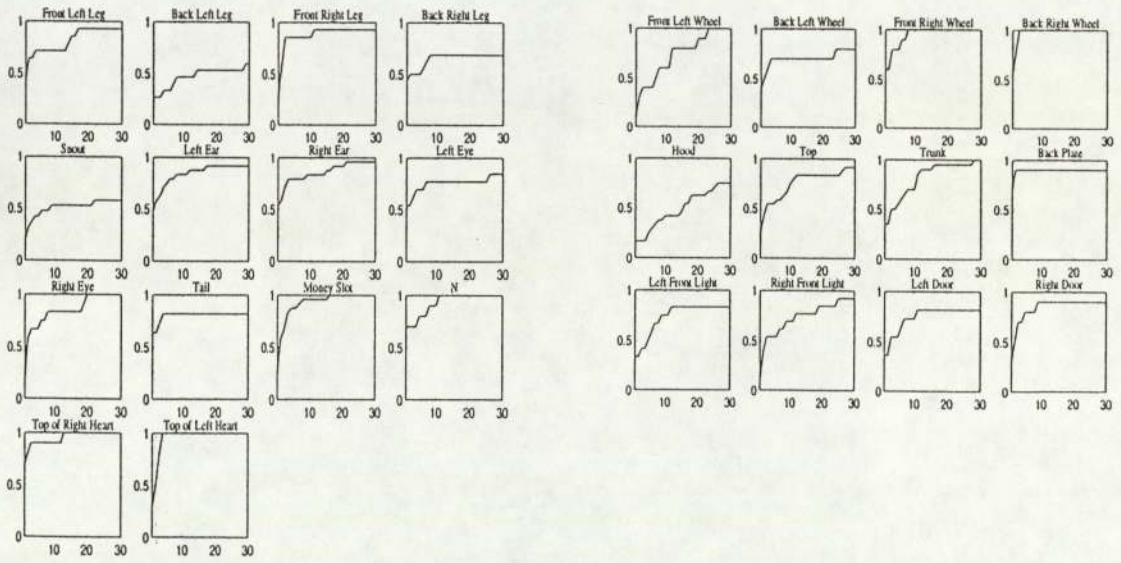

Figure B.14: ROC curves for MIA NN using shift invariance and committees.