Learning and Extracting Primal-Sketch Features in a Log-polar Image Representation*

HERMAN MARTINS GOMES[a], ROBERT B FISHER[b]

[a]Departamento de Sistemas e Computação, Universidade Federal da Paraíba
Av. Aprígio Veloso s/n, 58109-970 Campina Grande, PB, Brasil, hmg@dsc.ufpb.br
[b]Institute of Perception, Action and Behaviour, Division of Informatics, Edinburgh University
5 Forrest Hill Edinburgh EH1 2QL, Edinburgh, UK, rbf@dai.ed.ac.uk

Abstract. This paper presents a novel and more successful learning based approach to extracting low level features in a retina-like (log-polar) image representation. The low level features (*edges*, *bars*, *blobs* and *ends*) are based on Marr's primal sketch hypothesis for the human visual system [10]. The feature extraction process used a neural network that learns examples of the features in a window of receptive fields of the image representation. An architecture designed to encode the feature's class, position, orientation and contrast has been proposed and tested. Success depended on the incorporation of a function to normalise the feature's orientation and a PCA pre-processing module to produce better separation in the feature space.

1 Motivations

Traditional image feature extraction operators have usually been designed by hand, work independently of each other and act on Cartesian images (an artifact of sensor architecture). However, the architecture of the primate vision system seems to be quite different, and we can use this to produce interesting results in artificial vision systems.

The outermost primate retinal region is formed by rings with approximately the same number of receptive fields, whose distance from the retina centre can be expressed in terms of an exponential function [14]. The mapping from this region to the visual cortex can be mathematically approximated by a log-polar representation [13], which transforms both rotation and scaling in the Cartesian domain into translation in the log-polar domain and cuts off most of the complexity involved when recognising objects at different scales and orientations [12]. Moreover, the representation is space-variant, i.e., there is a high resolution centre surrounded by a progressively low resolution periphery, allowing a more compact representation for the image data. There are several examples in the literature of vision systems that take advantage of log-polar images [5, 9, 8].

The log-polar representation presented in this paper is composed of low-level features extracted using a different approach. The low level features (*edges*, *bars*, *blobs* and *ends*) are based on Marr's primal sketch hypothesis for the human visual system [10].. The primal sketch represents a richer representation for the image data and provides cues for an attention mechanism under the experimental evidence that they seem to attract visual attention [15].

Instead of trying to manually build a model for completely describing the features, which could be error prone and present some difficulties because of the unusual sensor geometry and the receptive field integration, learning the features was a sensible option. In this paper, a neural network approach was used due to its adequacy when learning data in which there is no obvious symbolic representation. An architecture designed to encode the feature's class, position, orientation and contrast has been proposed and tested. Success depended on the incorporation of a function to normalise the feature's orientation and a PCA pre-processing module to produce better separation in the feature space.

2 Related Work

Neural network learning of edge features has already been discussed in the literature. Some attempts have obtained only limited success, as for example the work of Pham and Bayro-Corrochano [11], in which a concatenation of two perceptrons was used: one for noise filtering and another one for edge detection. The edge detection network was trained to recover a given edge component within a 3x3 window at 8 different orientations (the position of an output neuron represented the orientation, and the node's output value corresponded to the edge intensity value). The results showed that the neural network approach presented a performance slightly inferior to that of the Sobel edge detector.

Chen *et al* [2] presented an edge labelling process in which a neural network was trained with synthetic data from a model of an ideal step edge. The aim was to label the central pixel of a 5x5 image patch as edge or non-edge. Illumination and rotation normalisation were performed over the image patch before feeding it into the neural network, which reduced some of the problem complexity. They compared the visual output of their system with the output produced by the Canny edge detector when applied to the same noisy data, and have found that the neural network has better noise tolerance than the Canny edge detector.

It is important to establish the differences between our

*This work is supported by CNPq and DSC/COPIN/UFPB, Brazil.

work and the above approaches. A general difference is that we have chosen a model which tries to capture interesting properties of the primate visual system architecture. Most previous research used a Cartesian feature space whereas we detect features in the log-polar space. Moreover, our aim is to classify several different features, in addition to edges, at number of different orientations and contrasts.

In the system developed by Grove and Fisher [5], primal sketch features were extracted within a log-polar image, as in this paper, but using a set of logical operators instead of a learning based approach. The operators were manually defined as expressions involving the pixels of a 1-ring window of 7 receptive fields which was applied throughout the log-polar image. Figure 1 illustrates the operators designed to detect *blobs* and *edges*.

```
+Blob=MIN(x-a,x-b,x-c,x-d,x-e,x-f)
-Blob=MIN(a-x,b-x,c-x,d-x,e-x,f-x)

Edge@0=ABS(f+a+b-c-d-e)/3
Edge@60=ABS(a+b+c-d-e-f)/3
Edge@120=ABS(b+c+d-e-f-a)/3
Edge@180=ABS(c+d+e-f-a-b)/3
Edge@240=ABS(d+e+f-a-b-c)/3
Edge@300=ABS(e+f+a-b-c-d)/3

Edge=MIN(Edge@0, Edge@60, ..., Edge@300)
```

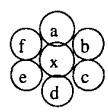

Figure 1: The mask used in Grove & Fisher's system [5] to detect features. Each pixel {x,a,b,c,d,e,f} corresponds to a particular receptive field output in polar coordinates. Detectors for *blobs* and *edges* are shown in the picture.

One of the problems with the above approach is that operators are heuristically defined and, therefore, there is no guarantee that they will work correctly with all possible cases and that they will allow graceful degradation. Also, if a different window size or window shape was needed, it would be necessary to manually design new logical expressions for the operators, which can lead to mistakes.

3 Image Representation

The input Cartesian image is resampled through the use of a mask consisting of concentric rings of overlapping circular receptive fields, whose centres are geometrically spaced from the centre of the mask (Fig. 2). If we define an image that is accessed by using the rings (logarithm of the distance of the rings to the retina centre) and sectors of the previous mask, then we have a log-polar representation. The innermost region, named the fovea, contains a high density hexagonal receptive field grid. We simulated a hexagonal packing outside the fovea by shifting each consecutive ring by half of the angle defining a sector of receptive fields. The radius of the nth outer retinal layer (or ring) is: $R(n) = \beta^n R(0)$, where $R(0)$ is the radius of the first layer exterior to the fovea and β defines the geometrical progression of distances of receptive field layers from the retinal centre (we have used $\beta \approx 1.1$). Similarly, the radius $r(n)$ of a particular receptive field in layer n is $r(n) = \beta^n r(0)$.

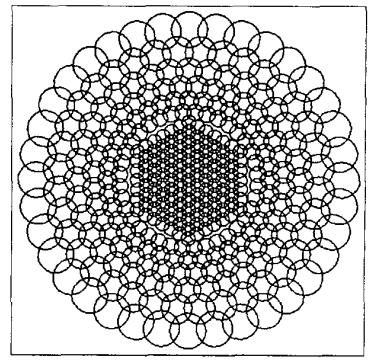

Figure 2: Retina structure. In order to enhance details in the figure, parameters different from those chosen in the experiments were used.

We have defined the fovea as having 11 layers of receptive fields. Each receptive field in the fovea has a radius of 0.5 of a pixel. Outside the fovea, there are 33 more layers of receptive fields distributed accordingly to the previous equation. A receptive field overlaps with each of its neighbours by approximately 53% of its diameter. These parameters produced a retina with a diameter of 256 pixels.

3.1 Estimating the Reflectance Information

The output of a given receptive field is calculated according to the following equation:

$$O = \sum_{x^2+y^2 \leq r^2} I(x,y)F(x,y) \qquad (1)$$

where O is the neuron output, $I(x,y)$ is the perceived intensity and $F(x,y)$ is the receptive field function, defined as a normalised Gaussian. Both I and F are applied to points (x,y) in the receptive field circular domain of radius r.

We designed a method for estimating the original reflectance information from the objects which is derived from the receptive field computation. By taking the logarithm of the intensities and assuming that $I(x,y) = E(x,y)\ R(x,y)$, where E is the irradiance falling on the object, and R is the local surface reflectance, we have:

$$O' = log(E) + \sum_{x^2+y^2 \leq r^2} log(R(x,y))F(x,y) \qquad (2)$$

The $log(E)$ term in Eq. (2) is nearly constant over local image regions and therefore makes the receptive field computation O' a good approximation for the weighted logarithm of the reflectance. Since the feature extraction operators described in the next section have 1) linear preprocessing in the initial projection stage and 2) the projection weights sum to approximately zero, then the projection

of the $log(E)$ terms in a feature neighbourhood will also be approximately zero. Thus, the feature extraction is primarily based on the reflectance structure of the neighbourhood.

4 Proposed Approach

Features are trained and detected in a window of receptive fields composed of a central receptive field plus its next 6 and 12 surrounding neighbours, totalling 19 receptive fields hexagonally distributed.

When centred within these windows, the oriented features (*edges*, *bars* and *ends*) can appear at several distinct orientations. As a result of the receptive field window structure, we have decided to detect *edges* and *ends* at 12 possible orientations and *bars* at 6 possible orientations. Since *bars* are indistinguishable by end direction, they have the same angular resolution as the *edges* and *ends*.

For training purposes, synthetic exemplars of the features are drawn in a fixed position on the input image corresponding roughly to a particular window of receptive fields. Then, the output of these 19 receptive fields is processed and used as input to the neural network classifiers.

4.1 Feature Detector

The overall system architecture is discussed here and details of the individual processes are given in subsequent sections.

The first step is to normalise the feature orientation (Sec. 4.2). Then, principal components are computed from a training set (counter-examples are not taken into account), see Sec. 4.3. Finally, only a subset of the principal components is chosen. This selection consists in choosing the *eigenvectors* associated to the highest *eigenvalues*. This is done for each of the 7 feature classes: *edge*, ± *bar*, ± *blob*, ± *end* (a + sign is assigned to features that have a darker background and a − sign to features that have a brighter background, Sec. 5.1 details this separation).

The next step is to project exemplars of features onto the previously selected subset of *eigenvectors* and use this information as training inputs to neural network modules (Sec. 4.4). The desired outputs for the neural networks are encoded from the feature's contrast (Sec. 4.5).

In order to extract features from a real image, a process similar to the training one is implemented, with the difference that the feature's projection is fed into a set of trained networks and a classification rule is used to interpret the network outputs (Sec. 4.6).

The last step (Sec. 5.4) is to improve the feature sets by selecting incorrectly classified features from real images, which gives the 'fine tuning' aspect of the approach. The above process is repeated until a satisfactory classification is achieved over a set of test images.

4.2 Normalising the Feature Orientation

If a receptive field window could be normalised into a standard orientation before applying the PCA technique (explained in section 4.3), the problem could be simplified because now we would end up with a smaller set of principal components related only to the normalised orientation.

To perform this task, we defined a symmetry operator as a gradient mask by associating negative weights to a subset of the receptive fields in the retinal window, and positive weights to the remaining receptive fields. By iteratively rotating the symmetry operator with respect to the central receptive field and applying it to a receptive field window, the detected orientation will be the one which maximises the absolute value of convolution. The last step consists of rotating the feature to a standard orientation. The operator is applied at the 12 orientations defined by the receptive fields in the outer ring of the window, giving a resolution of 30°.

There is a different symmetry operator for each of the oriented features. Note that it is not necessary to know which feature type we have before normalising, as we normalise with all feature types and apply the corresponding PCA and classification. More precisely, the operator's output for orientation $\theta \in \{0, 30, \ldots, 330\}$ is defined by:

$$Op_\theta^f(n,s) = \left| \sum_{i=0}^{2} \sum_{j=0}^{6i} w_\theta^f(i,j)\, V(G(i,j,n,s)) \right| \qquad (3)$$

where f is the feature type: *edge*, *bar* or *end*; (n, s) are the coordinates for the central pixel of the receptive field window; $w_\theta^f(i,j)$ is the operator's weight at the local window's coordinate (i,j); G is a function that maps from local to global coordinates within the retina; and V is the receptive field value at a given retinal point. Depending upon the feature type and orientation, the weights at each position can be either $+\frac{1}{N}$ or $-\frac{1}{M}$, where $N + M$=19, the number of receptive fields within the window (see Tab. 1). We select the θ that maximises Eq. (3) and then rotate the 19 receptive fields about the central field by $-\theta$.

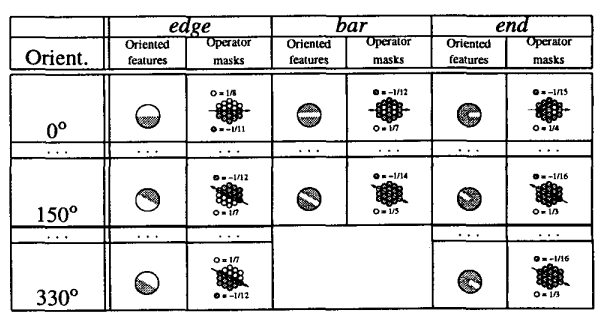

Table 1: Symmetry operators used to detect feature orientations. White circles represent a weight of $\frac{1}{N}$, and darker ones represent a weight of $-\frac{1}{M}$. N and M are the number of white and dark circles within the operator's mask, so that weights always sum to zero within any given mask. Arrows indicate the preferred feature orientation.

4.3 Principal Component Analysis (PCA)

PCA [6, 7] is a multivariate technique in which a number of related variables are transformed into a set of uncorrelated ones. These variables are called the *eigenvectors* and the coefficients used to reconstruct the original data are called the *eigenvalues*. These *eigenvectors* correspond to the directions of the principal components of the original data and their statistical significance is determined by the corresponding *eigenvalues*. Given a $m \times n$ matrix X containing m observations of n variables, PCA entails finding matrices V and D so that they satisfy the equation $C\ V = V\ D$, where $C = X^{\mathrm{T}}\ X$ is the covariance matrix, V is a $n \times n$ matrix containing the the *eigenvectors* of C and D is a diagonal $n \times n$ matrix containing the *eigenvalues* of C. Matrix X is normalised by subtracting, column by column, the mean value of the variables from each of its elements.

The determination of the *eigenvalues* and *eigenvectors* can be performed using Singular Value Decomposition (SVD). The SVD theorem states that given a $m \times n$ matrix X (as above), then there are orthogonal matrices U $(m \times m)$ and V $(n \times n)$ such that:

$$X = U \times \Sigma \times V^{\mathrm{T}} \tag{4}$$

where Σ is a $m \times n$ diagonal matrix containing the singular values of X, which are also the positive square roots of the (non-negative) *eigenvalues* of $X^{\mathrm{T}}\ X$, the columns of matrix U contain the *eigenvectors* of $X\ X^{\mathrm{T}}$, and the columns of matrix V contain the *eigenvectors* of $X^{\mathrm{T}}\ X$ [6, 7].

Our goal is to decompose the training sets into their *eigenvalues* and *eigenvectors* (or principal components). One decomposition is obtained for each of the four feature type's training sets (*edge*, *bar*, *blob* and *end*). We don't need to distinguish between positive (+) and negative (−) representations of features because the removal of the mean values leaves the two forms being the negative of each other. Each receptive field within a 19-receptive field window will be a variable, and each sample in a feature's training set will be an observation. More precisely:

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,19} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,19} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,19} \end{bmatrix} \tag{5}$$

where $x_{i,j}$ is the jth receptive field of the ith observation or training sample. The principal components are stored in a matrix V_i of 19×19 values for each class of features i.

An unknown data sample named Y (in our case, a 19-receptive-field-window obtained from a test image) can be projected onto the set of principal components V_i (obtained as described above) by a simple matrix multiplication: $P_i = Y' \times V_i$. The resulting projection P_i represents that data sample in terms of the principal components for a given primal sketch feature class i. If a component of the projection P_i is large, then it suggests that our data is close to the pattern the eigenvector represents, and if we have a low valued projection the conclusion is the other way around. We repeat this projection for all classes $i \in \{edge, bar, blob, end\}$.

Table 2 shows the first 7 principal components resulting from the application of the singular value decomposition on the normalised orientation training data.

	edge	*+bar*	*-bar*	*+blob*	*-blob*	*+end*	*-end*
1	15444.51	6614.50	8307.51	1406.85	2415.70	14811.19	25152.20
2	8574.82	3332.06	2665.63	156.96	94.70	4442.84	2667.40
3	238.69	155.40	155.47	3.44	3.30	222.40	221.31
4	63.58	69.19	69.69	0.17	0.17	68.29	68.73
5	10.12	31.89	31.95	0.16	0.15	51.79	52.03
6	2.31	10.18	10.07	0.15	0.13	7.35	7.34
7	1.99	2.48	2.48	0.13	0.13	4.48	4.59

Table 2: Pictorial representation of the first 7 *eigenvectors* and *eigenvalues* extracted from the training sets for each of the primal sketch feature classes. Under each picture is the corresponding *eigenvalue*. Note that the first 3-6 *eigenvectors* encode most of the variation.

PCA is used in our problem to transform the inputs that the neural network modules will receive. However, instead of using all 4x19 projections from the four feature classes, we combine (concatenate) subsets of the most representative principal components. There are in the literature several *ad-hoc* techniques to select only a subset of the most important principal components [7]. The simplest technique is to select the principal components associated with the largest *eigenvalues*. The general idea behind this selection is that we will be using the components that most contribute to describing the data. On the other hand, we are not interested in components representing the average intensities of the input patterns (the first components of Tab. 2) because those components would not help the process of spreading out the feature classes in the input space. Moreover the first component will not have approximate sum to zero, which is unsuitable for the reflectance estimation discussed in Sec. 3.1. We also do not need to keep the components for features of both positive and negative contrasts, as they are almost the same in Tab. 2. From the above, the following subsets of principal components were kept: {2,3,4,5,6} for *edge*, *bar* and *end*; and {2, 3} for *blob* features. Thus the input to the neural networks is reduced to a vector of 17 elements.

4.4 Neural Network Architecture

We used MLP-*backpropagation* networks minimising a least square error metric, due to its simplicity and reasonable computational power. An initial attempt [4] was to partition three of the feature classes (*bars*, *blobs* and *ends*) into six new feature classes according to their contrast intensity (*positive* or *negative*). Then, seven different neural modules, each one designed for a particular feature, was built. These modules had an input layer composed of 19 neurons, followed by a hidden layer and an output layer containing neurons associated with each of the 6 or 12 standard orientations plus one neuron representing a non-feature class. The strength of response of an output neuron was trained as a function of the feature's contrast. In the case of *blobs*, the network's output layer had only 2 neurons, one coding the *blob* itself and the other coding the *non-blob* class. The training of these networks converged quickly, however there were lots of misclassifications when testing on real images, probably caused by a training set with insufficient examples and by poor separation in the input space.

In order to achieve better understanding of the problem, we performed a principal component analysis on the training data. We realised how the complexity of the problem could be reduced by using PCA to increase the class separation at the network inputs level. The lack of exemplars in the training set was tackled by using a bootstrapping approach which is discussed in Sec. 5.

The final networks architecture was almost identical to the architecture described above, differing only at the input and output layers: (a) instead of receiving inputs directly from the 19 receptive field windows, the networks receive the results of the PCA pre-processing module; and (b) instead of several output neurons to represent all the possible feature's orientations (now this is done by a separate module), there were only 2 output neurons, neuron N representing the feature and neuron $\tilde{N}$ representing the non-feature class. A total of seven neural networks were built, one for each of the seven feature classes: *edge*, *+bar*, *−bar*, *+blob*, *−blob*, *+end*, *−end*. Each network had 17 inputs, 9 hidden neurons and 2 output neurons, all fully connected.

4.5 Coding the Contrast Information

The contrast within a retinal window is calculated according to the Eq. (6) [1]. The desired output for a neuron representing a particular feature was represented in terms of this contrast.

$$c = \frac{|L_{max} - L_{min}|}{L_{max} + L_{min}} \tag{6}$$

where L_{max} and L_{min} are the minimum and maximum intensities found in an image patch, respectively.

4.6 Classification Rule

We used a classification rule that takes into account the strategy used during training: whenever a feature that should be recognised by a neural module is presented then the network is trained to output the feature's contrast through neuron N, and zero through neuron $\tilde{N}$; if a counter-example is presented, then neuron N should now output 0 and neuron $\tilde{N}$, 1. However, we need to be more tolerant when classifying untrained features, as the network outputs will not necessarily produce a sharp separation between feature and non-feature classes. Let's consider a neural module representing a given feature class with its 2 output neurons; the classification rule used in our experiments is presented in the following pseudo-code, which basically states that a module recognises a features whenever its neuron N produces an output that is above a threshold THD and also above the output of the other neuron $\tilde{N}$:

if $((O(N) > O(\tilde{N}))$ **and** $(O(N) > THD))$
  then $C = O(N)$
  else $C = 0$
endif

where THD is the classification threshold, $O()$ is a function that returns the neuron's numerical output, and C is the output of the classification rule, representing the contrast of the detected feature (C is 0 when no feature is detected). Section 5.3 explains how we obtained the value of THD.

5 Training and Evaluation

We constructed an initial training set from synthetically generated features. This seems to be a better approach than manually extracting and labelling lots of features from real images because a large number of feature variations can be easily generated and, by using the hypothesis that the feature examples are chosen from a more descriptive set, this allows for a smaller training set to be constructed. After some initial experiments, we reached the conclusion that, although the synthetic training set was very useful, the manual selection of a small set of features from real images was still required for achieving better results.

5.1 Synthetic Training Data

Contrasts within the set $\pm\{0.3, 0.4, 0.6, 0.8, 1.0\}$ were used when drawing *bars*, *blobs* and *ends*. A negative contrast here means that the intensities in the feature background are higher than the ones in the feature itself. *Edges* were drawn using only positive contrasts, i.e. the intensities in the region above the feature orientation line are greater than the intensities in the lower region in order to avoid the generation of the same pattern twice as the feature orientations covers the whole circle. Fifteen different combinations of intensity were used in the generation of the contrasts for all of the features. One of the intensities was taken from the set $\{85, 170, 255\}$ and the other was derived according to Eq. (6) to produce the desired contrast.

Ends and *edges* were generated at 12 different orientations in the range ($0^o, \ldots, 330^o$), in steps of 30°, while

bars were generated only at 6 orientations in the range (0°, ..., 150°) due to symmetry reasons, and also in steps of 30°. Other sources of variability in the training sets were the "size" of the feature and the use of Gaussian additive noise. *Blobs*, *bars* and *ends* were allowed to vary in size according to 0.6, 0.7 and 0.8 of the central receptive field diameter. Gaussian additive noise was added to the drawn features in order to broaden the training set.

Random *counter examples* were generated in order to help the training process. These *counter examples* simulate unstructured input data and data from other low level features not considered in this work that are inevitably present in real images. Counter examples sets were also enriched with exemplars from the other feature classes. Table 3 contains some examples of the training features.

	edge	*+bar*	*-blob*	*+end*	*counter* example
Cartesian inputs					not applicable
Retinal outputs					

Table 3: Some examples of the training features. Differences in contrast between Cartesian inputs and retinal outputs are due to the logarithmic receptive field computation.

5.2 Initial Training

We used a learning rate of 0.005, momentum of 0.95 and a neural module was considered trained when all the training patterns passed with a 0.1 error bound. On average, it took about one thousand epochs to train each module.

5.3 Evaluating the System's Performance

The trained networks were tested with a set of unknown synthetic features created using the same generators used to build the initial training sets, with the difference that now the features were generated at arbitrary orientations and contrasts. We generated 80 test exemplars per class by linearly varying the contrast within the range of 0.21 to 1.0, with a step of 0.01 (-0.21 to -1.0, for negative features). The intensities used to produce the above contrasts were chosen randomly as well as the remaining parameters specifying orientation, noise level and size.

Figure 3 shows the target output contrasts (vertical axis) used for all the testing sets (the horizontal axis shows the pattern number). The first 80 patterns are examples of the feature class in order of increasing contrast, and the next 83 patterns are counter examples (the first 40 were randomly generated and the remaining 43 were randomly chosen from other class examples).

Figure 4 shows the actual network outputs of the *edge* classifier when fed with the synthetic testing data. All the

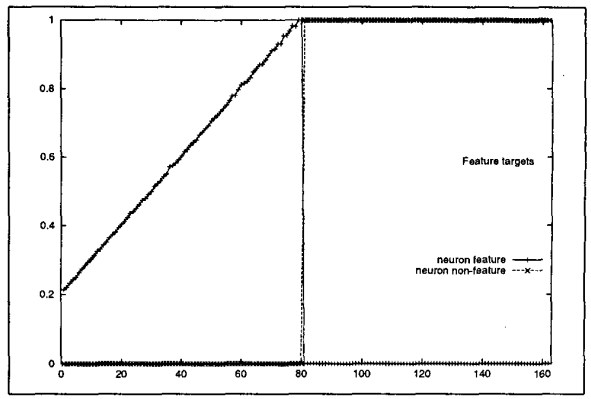

Figure 3: Target outputs for the testing sets.

other classifiers presented graphs very similar to that of the *edge* classifier. The oscillations along the network outputs for the testing sets when compared to the target outputs in Fig. 3 are partially explained by the convergence error of 0.1 used during the neural network training. The same applies to the small oscillations in the second half of the picture, corresponding to the networks response to counter-examples. Reducing the neural network convergence error could reduce the prediction errors during testing, but at the cost of a slower training and risk of *overfitting*, which could incur loss in generalisation. Another cause for the oscillations within the first half of the graphs are the small prediction errors caused by the symmetry operators.

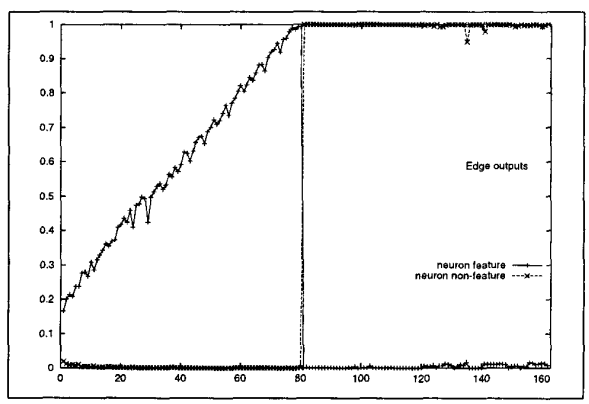

Figure 4: Outputs from the trained *edge* neural module when applied to its synthetic testing set.

We computed the absolute differences between real and estimated orientations when processing the feature's through the symmetry operators. As an indication of good performance, the top errors were all under the operator's resolution of 30 degrees.

The value of the threshold THD (see Sec. 4.6) was chosen as the highest classification threshold that produces best overall performance. The reason for this is that we are not interested in detecting very low contrast features. We

have varied the threshold from 0.0 to 1.0, in steps of 0.01, and measured the classification errors. From these experiments we observed that any threshold smaller than 0.15 is associated with the best overall performances (smallest classification errors) in all the networks.

5.4 Adding Features from Real Images

The final step of the approach is to enrich the training sets with features extracted from real images. Initially a set of real images was tested using the modules trained with synthetic features. Then a small subset of these images was used as the source of additional training exemplars, which were manually selected. The intensities used to compute the contrast of these new exemplars are estimated from their retinal outputs as if they were composed of uniform intensity patches in the Cartesian domain (similarly to the synthetic features). Finally, the neural modules were re-trained with the improved training sets and tested over the same initial set of images. This process is repeated until the visual output of the extracted features is satisfactory. Preliminary experiments have shown that the addition of only a few exemplars of real features (typically 5-10) was enough to cause a visible improvement in the networks generalisation ability. Images illustrating this improvement were not included in the paper because of limitations in space.

5.5 Experimenting on Synthetic and Real Images

In order to help the reader understanding the outputs of the feature extraction system described in this paper and before applying it to any real images, which may contain complicated textures and features not easily spotted by the eye, a number of synthetic images were initially used. These examples are shown in Fig. 5. From a subjective evaluation, apart from some small uncertainty with regards to the feature location and smaller accuracy at the centre of the image, where the receptive field size is too small to match the feature's size, it is possible to conclude that the classifiers are doing a reasonable job at detecting the features they have been designed for.

Figure 6 compares the results of the proposed and previous approaches when applied to the same real test image. There, we can see that the new approach brought a reasonable improvement to the feature extraction process not only with respect to accuracy (more features correctly extracted) but also to the enhanced quantization of the contrast.

6 Concluding Remarks

A previous attempt to extract primal sketch features achieved only partial success [5]. Features were detected using a number of manually defined logical operators within a fixed retinal window, which, when applied to real images, failed to detect some low contrasting features as well as misclassified a number of others (see Fig. 6 for reference).

The approach presented in this paper takes advantage of the inherent learning (from examples) and generalisation properties of neural networks. Instead of designing feature extraction operators from scratch, the main idea was to use a set of exemplars of features and non-features to train a neural network. An initial architecture received some refinements and ended up having seven independent neural modules (only connected through the training sets, i.e. examples from one module used as counter-examples in the others), one module for each of the feature classes considered, receiving inputs from a PCA pre-processing module, whose main purpose was to spread out the classes in the feature's manifold.

This novel approach presented better results with respect to the number of correctly classified features, provided a richer description for the image data with the addition of an estimate for the feature's contrast, and became a more flexible solution to the problem in the sense that whenever a new feature class is required, only its training set needs to be provided. Primal sketch features obtained by the modules discussed in this paper are being successfully used as image representations for the problem of learning structural relationships from sets of iconic (2D) object models obtained from a sequence of scenes [3].

References

[1] V. Bruce, P. Green, and M. Georgeson. *Visual Perception: Physiology, Psychology, and Ecology*. Psychology Press, 3rd edition, 1996.

[2] W. C. Chen, N. A. Thacker, and P. I. Rockett. A neural network for probabilistic edge labelling trained with a step edge model. In *Proc. of 5th Int. Conf. on Image Processing and its Applications*, pages 618–621, Edinburgh, July 1995.

[3] H. M. Gomes and R. B. Fisher. Structural learning from iconic representations. *Lecture Notes in Artificial Intelligence*, 1952:399–408, 2000. Springer-Verlag.

[4] H. M. Gomes, R. B. Fisher, and J. Hallam. A retina-like image representation of primal sketch features extracted using a neural network approach. In *Proc. of Noblesse Workshop on Non-Linear Model Based Image Analysis*, pages 251–256, Glasgow, July 1998. Springer-Verlag.

[5] T. D. Grove and R. B. Fisher. Attention in iconic object matching. In *Proc. of British Machine Vision Conf.*, volume 1, pages 293–302, Edinburgh, 1996.

[6] J. E. Jackson. *A User's Guide to Principal Components*. John Wiley & Sons, 1991.

[7] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, 1986.

[8] F. Jurie. A new log-polar mapping for space variant imaging: application to face detection and tracking. *Pattern Recognition*, 32:865–875, 1999.

[9] F. L. Lim, G. A. W. West, and S. Venkatesh. Use of log polar space for foveation and feature recognition. *IEE Proc. Vision, Image and Signal Proc.*, 144(6):323–331, Dec 1997.

[10] D. Marr. *Vision*. W. H. Freeman and Co., 1982.

[11] D. T. Pham and E. Bayro-Corrochano. Neural networks for low-level image processing. In *Proc. of the IEE Int. Conf. on Artificial Neural Networks*, pages 809–812, 1992.

[12] G. Sandini and M. Tristarelli. Vision and space-variant sensing. In H. Wechsler, editor, *Neural Networks for Perception*, chapter II.11, pages 398–425. Academic Press, 1992.

[13] E. L. Schwartz. Spatial mapping in primate sensory projection: analytic structure and relevance to perception. *Biological Cybernetics*, 25:181–194, 1977.

[14] S. W. Wilson. On the retino-cortical mapping. *Int. J. Man-Machine Studies*, 18:361–389, 1983.

[15] A. L. Yarbus. *Eye Movements and Vision*. Plenum, 1967.

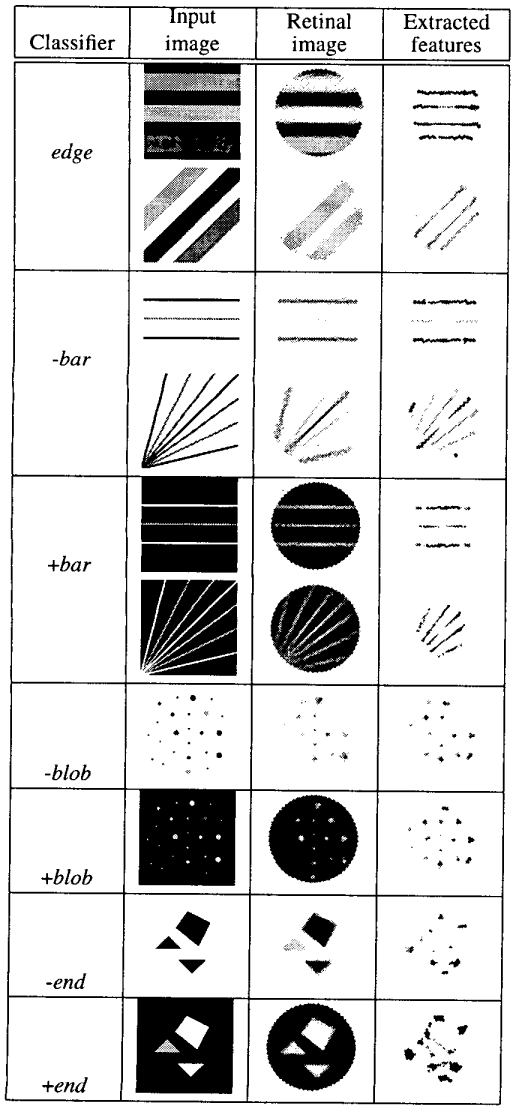

Figure 5: Testing the classifiers on synthetic images

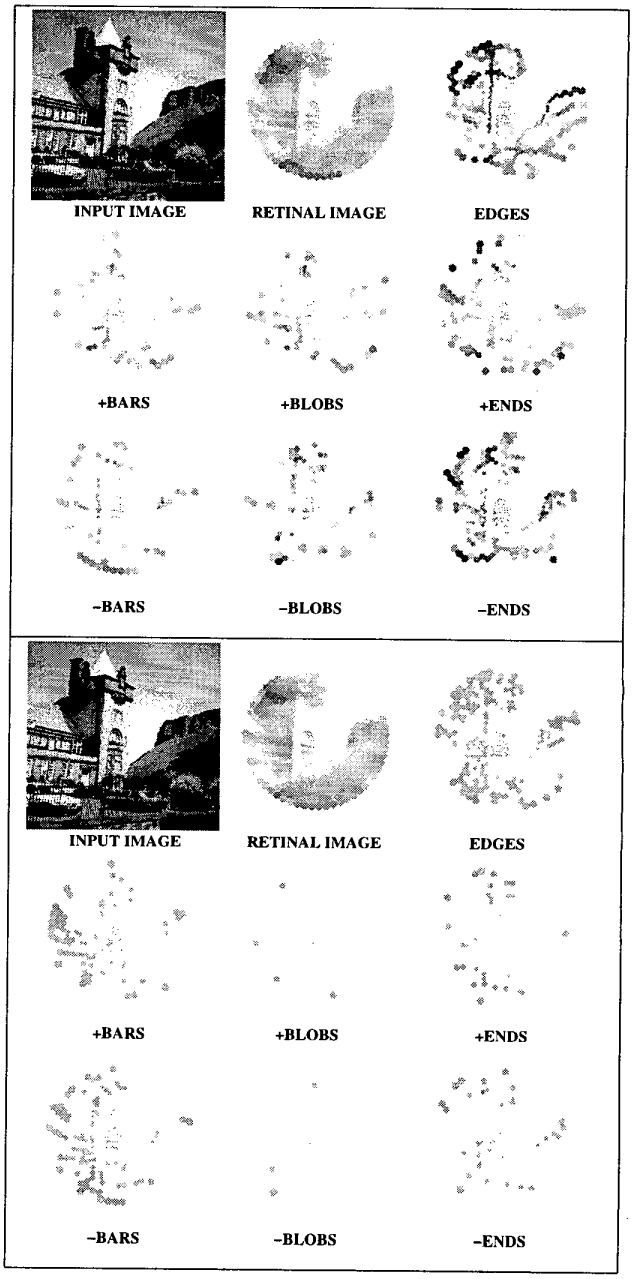

Figure 6: Testing the classifiers on real images. From top to bottom: final results from our approach and from a previous approach proposed by Grove and Fisher [5].