Shift, Rotation and Scale Invariant Signatures for Two-Dimensional Contours, in a Neural Network Architecture

David McG. Squire* Terry M. Caelli

School of Computing, Curtin University of Technology
Perth, Western Australia, AUSTRALIA

June 30, 1995
1st International Conference, Mathematics of Neural Networks and Applications
3-7 July 1995, Lady Margaret Hall, Oxford

Abstract

A technique for obtaining shift, rotation and scale invariant signatures for two dimensional contours in images is proposed and demonstrated. The technique is based upon a measure of the degree to which the local tangent estimate at each point in the contour is consistent with having been generated by a shift, rotation or scale invariant function. An *invariance factor* is calculated at each point by comparing the orientation of the tangent vector with each of the vector fields corresponding to the Lie generators of the transformation groups for shift, rotation and scaling. The statistics of these invariance factors over all points in the contour are used to produce an *invariance signature* for the contour.

The operations described above are implemented in a Model-Based Neural Network: a network in which the structure of layers and nodes and the values of the weights are parameterised by constraints of the problem domain. This specific network calculates the invariance signature for a contour in an input image, and uses this as the input to hidden layers which form a traditional neural network classifier. It should be noted that the end result after constructing and training this system, is exactly the same as a traditional neural network: a collection of layers of nodes with weighted connections. The design and modeling process can thus be thought of as "compiling" a shift, rotation and scale invariant classifier into a neural network. The final product could be used at run-time on hardware or software designed solely for traditional neural networks.

We contend that these invariance signatures, whilst not uniquely characterising any given contour, are sufficient to characterise contours for many two-dimensional pattern recognition tasks. Results are presented demonstrating the utility of this technique for invariant pattern recognition.

1 Introduction

1.1 The Model-Based Approach to Building Neural Networks

Most neural network formulations, independent of the architecture or training technique employed, have several features in common. They are:

1. Problems are solved by the determination of weights in a weight space with a very high dimensionality.
2. Learning and run-time information processing are inherently parallel.
3. The weights are determined solely by applying a training algorithm which is driven by a training set of input and output exemplars.

The model-based approach to neural network design aims to retain some of the advantages of Traditional Neural Networks (TNNs), for instance parallel data-processing, but to modify the way in which the architecture of the network and the values of the weights are determined. In the Model-Based Neural Network (MBNN) formulation, the network designer can use expert knowledge of the problem domain and desired network performance to constrain this process.

This notion of the Model-Based Neural Network was introduced in [1]. In that paper we proposed networks

*David Squire was funded by an Australian Postgraduate Award

in which the connection weights were not independent parameters, as in a TNN, but were determined by a weighting function that could depend upon the relative positions of nodes in the network, and on several parameters. These parameters could be shared between all connections on the input to a given node, or between all connections to all nodes in a layer. It was demonstrated that this formulation allowed networks to be designed which incorporated layers that were guaranteed to perform certain operations on the input, such as specific sorts of spatial filtering. This greatly reduced the dimensionality of the space which had to be searched during training, since only a small number of independent parameters determined the weights on the connections to all nodes in a layer. The size of the training set could also be reduced, since the network was guaranteed to respond only to certain features of the input. Moreover, once training was completed the weighting function for each connection could be evaluated and that final value stored as the weight for that connection, so that the resultant network was just a collection of nodes connected by weighted connections, exactly as in a TNN.

The key notion to emerge from this work was that neural networks with highly desirable properties could be produced by using the designer's knowledge of the problem domain to constrain the process by which the connection weights were determined. The MBNN approach departs from the TNN view of neural networks as "black boxes", the final operation of which is determined entirely by the composition of the training set supplied to the training algorithm.

1.2 Model-Based Neural Networks and Invariant Pattern Recognition

Many see the fact that the parameters of neural networks are entirely "learnt" as one of their good features. It can, however, be a limitation. In order to achieve good performance, it is necessary that the training set be sufficiently large and varied to "span" the space of possible inputs for which the network is supposed to produce a sensible output. The collection of this data and the training of the network can be a very time-consuming process. The MBNN formulation allows the designer to create networks which are guaranteed to respond to the presence of particular features in the input image, and possibly to be invariant under certain transformations of the input image. This means that it is not necessary to provide the training process with a data set containing various shifted, distorted or otherwise transformed versions of the input patterns, such has long been a common approach to achieving invariant pattern recognition using neural networks [2].

In this work, the concept of Model-Based Neural Network has been extended to include networks that are modular. The modules which make up the MBNN have a well-defined functionality, and the weights on the connections in each module can be arrived at by any technique at all: some may be directly set by the designer, others might be found by training the module (rather than the whole network) for its specific task. This approach allows great flexibility to the network designer. Moreover, because modules can be trained separately, they can be trained to perform data processing tasks quite independent of the eventual classification of the input pattern.

2 Invariance Signatures

2.1 Functions Invariant With Respect To Lie Transformation Groups

Consider a function $F(x,y)$. We wish to determine its invariance with respect to a Lie transformation group. Let

$$\mathbf{G}(x,y) = \begin{bmatrix} g_x(x,y) & g_y(x,y) \end{bmatrix}^T \tag{1}$$

be the vector field corresponding to the Lie generator of the group. F is constant with respect to the action of the generator if

$$\nabla F \cdot \mathbf{G}(x,y) = 0, \tag{2}$$

i.e.

$$\frac{\partial F}{\partial x} g_x + \frac{\partial F}{\partial y} g_y = 0 \tag{3}$$

or

$$\frac{\frac{\partial F}{\partial x}}{\frac{\partial F}{\partial y}} = -\frac{g_y}{g_x} \tag{4}$$

Now consider a contour parameterised by t on which F is constant

$$F(x(t), y(t)) = C \ \ \forall \ \ t. \tag{5}$$

Since F is constant on the contour, we have:

$$\frac{dF}{dt} = \frac{\partial F}{\partial x}\frac{dx}{dt} + \frac{\partial F}{\partial y}\frac{dy}{dt} = 0, \tag{6}$$

consequently

$$\frac{\frac{\partial F}{\partial x}}{\frac{\partial F}{\partial y}} = -\frac{dy}{dx}. \tag{7}$$

Thus the condition for invariance with respect to the Lie transformation group generated by $\mathbf{G}$, given in equation 2, holds if

$$\frac{dy}{dx} = \frac{g_y}{g_x}. \tag{8}$$

on the contour.

2.2 Mapping Points from Image Space to Invariance Space

The technique is based upon comparing the orientation of the tangent to the contour at each point with those of the vector fields characterising given transformations. At each point both the tangent vector $\theta(x, y)$ and the vectors of the vector fields $\mathbf{v_c}(x, y)$ are normalised. The measure of consistency with a given invariance class c at a point (x, y), $\iota_c(x, y)$, is the absolute value of the dot product of the tangent vector with the vector field.

$$\iota_c(x, y) = \left| \hat{\theta}(x, y) \cdot \mathbf{v_c}(x, y) \right| \tag{9}$$

This operation maps each point from the two dimensional image to the interior of a unit hypercube in an n-dimensional *invariance space*, where each dimension corresponds to a particular invariance class (in this case rotation, dilation and translation):

$$\hat{\theta}(x, y) \longrightarrow \left[\begin{array}{ccc} \iota_{rot} & \iota_{dil} & \iota_{trans} \end{array} \right]^T \tag{10}$$

The magnitude of each component of the vector in invariance space gives the measure of the degree of invariance with respect to the corresponding transformation.

In order that the *invariance signature* of a contour be invariant under shifts of the contour, the origin of the image space is chosen to be the centroid of the points in the contour. The vector fields characterising functions invariant under given transformations are specified relative to this origin. The vector fields for rotation and dilation invariance are:

$$\mathbf{v_{rot}}(x, y) = \frac{1}{\sqrt{x^2 + y^2}} \left[\begin{array}{cc} -y & x \end{array} \right]^T \tag{11}$$

$$\mathbf{v_{dil}}(x, y) = \frac{1}{\sqrt{x^2 + y^2}} \left[\begin{array}{cc} x & y \end{array} \right]^T \tag{12}$$

The translation invariance case is somewhat different. The vector field used is constant for all (x, y), and the direction is determined by finding the eigenvector corresponding to the dominant eigenvalue, $\mathbf{e_1}$, of the coordinate covariance matrix of all the points in the contour. The direction of this eigenvector is the principal direction of the contour. It is inherently invariant to shifts of the image, and since $\mathbf{e_1}$ is calculated from the image each time it is required, this measure is not affected by rotations or dilations either.

$$\mathbf{v_{trans}}(x, y) = \left[\begin{array}{cc} e_{1_x} & e_{1_y} \end{array} \right]^T \tag{13}$$

The invariance signature for a given input image consists of a histogram of the invariance measures for all the "on" points in the image, for each invariance class. The number of bins in each histogram is determined by the network designer. This invariance signature is invariant under rotations, dilations, translations and reflections of the input image. It should be noted that this encoding of the image may not be unique, unlike some previous integral transform representations [3].

3 A Neural Network System For Computing Invariance Signatures

A Model-Based Neural Network was constructed to compute the invariance signature of an input pattern, and to classify the pattern on the basis of this. This MBNN consisted of a fairly complex system of neural network modules, some hand-coded and some trained on sub-tasks. A schematic diagram is shown in Figure 4 on page 7. Whilst the system appears complex, it retains the basic characteristics of a traditional feed-forward neural network. It consists entirely of simple nodes joined by weighted connections [1]. Each node i in the network compute the sum of its j weighted inputs, net_i,

$$net_j = \sum_j w_{ij} x_j. \tag{14}$$

This is then used as the input to the transfer function f of the node, which is either linear, $f(net_j) = net_j$, or the standard sigmoid,

$$f(net_j) = \frac{1}{1 + e^{-net_j}}. \tag{15}$$

[1] with the exception of the *Dominant Image Orientation Unit*, for which a neural network solution is still being developed.

The only departure from a traditional neural network at runtime is that some of the connection weights are dynamic, meaning that the weight is calculated by a node higher up in the network. This allows the MBNN to compute dot products, and also for some nodes to act as gates controlling whether or not the output of a node is transmitted. Since connection weights in any implementation of a neural network are no more than references to some stored value, this should not present any difficulty [2].

There is insufficient space in this work to give detailed descriptions of the all the sub-networks used to implement each functional unit of the system. A detailed description of the *Local Orientation Extractor* and the *Binning Unit* will be given as examples of the way in which some of the sub-networks were constructed.

3.1 Local Orientation Extraction

In order to compute the invariance signature for an image, it is necessary to estimate the tangent vector at each point of the contour, since the mapping to invariance space requires the dot product of this with the three vector fields described in section 2.2. A simple and robust estimate of the tangent vector at a point is the eigenvector corresponding to the largest eigenvalue of the covariance matrix of a square window centred on that point. This presents some difficulties for a neural network, as the mapping is not continuous. There are two discontinuities. The first arises when the dominant eigenvalue changes, and the orientation of the tangent estimate jumps by $\frac{\pi}{2}$ radians. The second is due to the flip in sign of the tangent vector when the orientation goes from π back to 0.

The first discontinuity is avoided by using a weighted tangent estimate. Rather than being unity, the magnitude of the estimated tangent vector corresponds to strength of the orientation. Let the eigenvalues of the covariance matrix be λ_1 and λ_2, where $\lambda_1 \geq \lambda_2$. The corresponding unit eigenvectors are $\hat{\mathbf{e}}_1$ and $\hat{\mathbf{e}}_2$. The weighted tangent vector estimate $\mathbf{s}$ is given by

$$\mathbf{s} = \begin{cases} \left(1 - \frac{\lambda_2}{\lambda_1}\right) \hat{\mathbf{e}}_1 & \lambda_1 \neq 0 \\ 0 & \lambda_1 = 0 \end{cases} \tag{16}$$

A standard three layer neural network with a 3×3 input layer, a 20 node hidden layer, and 2 output nodes was trained using backpropagation [4] to produce this output for all possible binary 3×3 input windows with a centre value of 1. The performance measure used, E, was the ratio of the sum squared error to a variance measure for the training set,

$$E = \frac{\sum_{c=1}^{N} \sum_{j=1}^{M} \left(t_{cj} - y_{cj}\right)^2}{\sum_{c=1}^{N} \sum_{j=1}^{M} \left(\overline{t_{cj}} - t_{cj}\right)^2} \tag{17}$$

where N is the number of training exemplars, M the number of nodes in the output layer, t_{cj} the desired output value of the node and y_{cj} the actual output value.

This network proved very sensitive during training, so it was necessary to use an extremely small learning rate. After 6000000 iterations with a learning rate of 0.0005, a value of $E = 3.11\%$ was reached. This was considered sufficiently accurate for this module.

This module is replicated across the input layer so that the x and y components of the tangent vector are estimated everywhere. The result is two images of the same size as the input layer. The values are also gated by the input image, so that values are only transmitted for nodes corresponding to ones in the input image.

It is noted that this is a similar problem to edge extraction, the difference being that edge extraction is usually performed on images containing greyscale gradients rather than on a binary, thin contour. Srinivasan *et al.* [5] have developed a neural network edge detector which produces a weighted vector output very much like the one described in equation 16. They used an encoder stage which was trained competitively, followed by a backpropagation-trained stage. The encoder produced weight distributions closely resembling Gabor filters of various orientations and phases. It is intended to try to produce a more compact and accurate tangent estimator using a MBNN incorporating a stage with Gabor weighting functions, as used in [1].

3.2 The Binning Unit

As an example of one of the modules for which the weights were determined entirely by hand, consider the binning unit shown in figure 1. There is one of these modules for each of the n bins comprising the invariance signature histogram for each invariance class. Each binning module is connected to all the nodes in the invariance image, and inputs to it are gated by the binary input image, so that only the N nodes corresponding to ones in the input image contribute.

The n bins have width $\frac{1}{n-1}$, since the first bin is cen-

[2] The aim of our research with MBNNs has never been biological plausibility. Perhaps this runtime weight calculation can be thought of as real-time, rapid, accurate training.

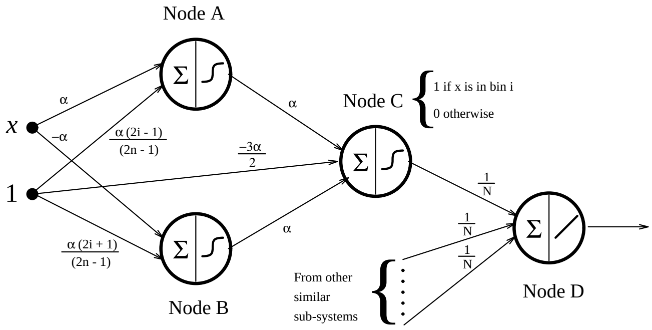

Figure 1: Neural Binning Subsystem

tred on 0, and the last on 1 [3]. The system is designed so that nodes A and B only have an output of 1 when the input x is within bin i. The condition for x to be in bin i is:

$$\frac{2i-1}{2(n-1)} < x < \frac{2i+1}{2(n-1)} \tag{18}$$

In order to detect this condition, the activations of nodes A and B are set to:

$$net_A = \alpha x - \frac{\alpha(2i-1)}{2(n-1)} \tag{19}$$

$$net_B = -\alpha x + \frac{\alpha(2i+1)}{2(n-1)} \tag{20}$$

where α is a large number, which causes the sigmoid response to approximate a step function. In this study $\alpha = 1000$ was used. The outputs of these nodes go to node C, which acts as an AND gate. Node D sums the contributions to bin i from all N nodes.

4 Simulation Results

In order to demonstrate the feasibility of such an invariance signature-based classifier, a small simulation was done. The task was to classify an input pattern as one of the ten first letters of the alphabet {a, b, c, d, e, f, g, h, i, j}. A lowercase example of each of these letters was obtained from an on-screen version of the helvetica font. A training set was created containing four examples of each letter[4], each differently rotated or reflected[5], within an 18×18 input image. A test set was created with three differently transformed examples of each letter.

The final stage of the invariance signature classifier was trained on this data, and the progress on the test set monitored during training. The final classifier used a 9 node hidden layer and was trained using backpropagation. A TNN with a 18×18 input layer, 9 node hidden layer and 10 node output layer was trained and monitored in the same way. The graphs showing progress during training appear in Figures 2 and 3.

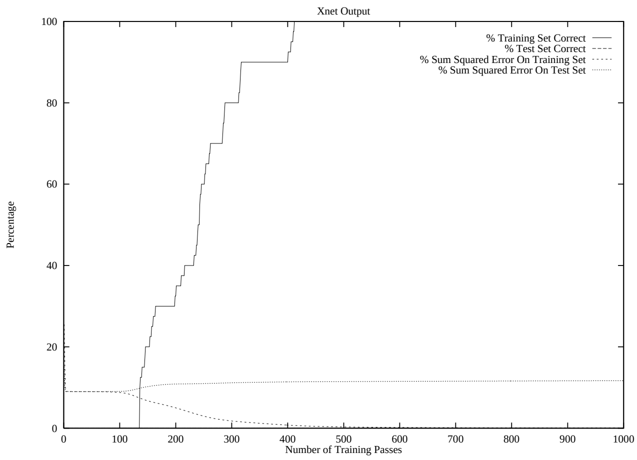

Figure 2: Tradition Neural Network Classifier Performance

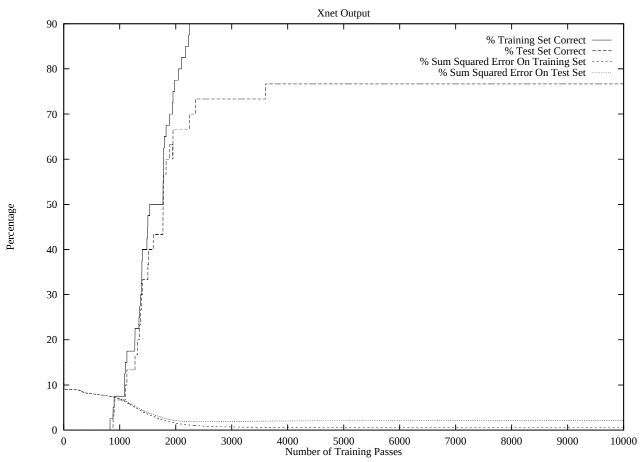

Figure 3: Invariance Signature-Based Classifier Performance

It is clear that the invariance based classifier, achieving 76.66% correct classification of the test data massively outperforms the TNN, which did not correctly

[3]This is because a bin ending at an extreme of the range would miss contributions from the extreme value since the edge of the neural bin is not vertical.

[4]The only reason that more than one example of each letter was required in the training set was that the signatures generated by different orientations of the letter were slightly different, due to the small residual error in the *Local Orientation Extractor*.

[5]all rotations were by multiples of $\frac{\pi}{2}$ radians.

classify *any* of the test data. It should be noted that had the *Local Orientation Extractor* been ideal, the best possible classification performance on the training data would have been 80%, since the characters “b” and “d” were identical under reflection in the font used.

5 Conclusion

It has been shown that it is possible using the model-based approach to neural network design to produce a neural network which is guaranteed to classify input contours on the basis of signatures derived from their local transformation properties which are rotation, dilation, translation and reflection invariant. The weights of this network were arrived at either by hand or by the training of sub-networks, but the resultant network retained the useful neural network properties of being composed entirely of similar simple nodes joined by weighted connections, with computation being inherently parallel. The MBNN approach can thus be seen as a useful technique for *compiling* a neural network, incorporating the designer's expert knowledge.

References

[1] Terry M. Caelli, David McG. Squire, and Tom P.J. Wild. Model-based neural networks. *Neural Networks*, 6:613–625, 1993.

[2] Kunihiko Fukushima, Sei Miyake, and Takayuki Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man and Cybernetics*, 13(5):826–834, 1983.

[3] Mario Ferraro and Terry M. Caelli. Lie transform groups, integral transforms, and invariant pattern recognition. *Spatial Vision*, 8(1):33–44, 1994.

[4] D.C. Plaut and G.E. Hinton. Learning sets of filters using backpropagation. *Computer Speech and Language*, 2:35–61, 1987.

[5] V. Srinivasan, P. Bhatia, and S.H. Ong. Edge detection using a neural network. *Pattern Recognition*, 27(12):1653–1662, 1994.

[6] Terry M. Caelli and Zhi-Qiang Liu. On the minimum number of templates required for shift, rotation and size invariant pattern recognition. *Pattern Recognition*, 21(3):205–216, 1988.

[7] Patrice Simard, Bernard Victorri, Yann Le Cun, and John Denker. Tangent Prop - a formalism for specifying selected invariances in an adaptive network. *Advances in Neural Information Processing Systems*, 4:895–903, 1992.

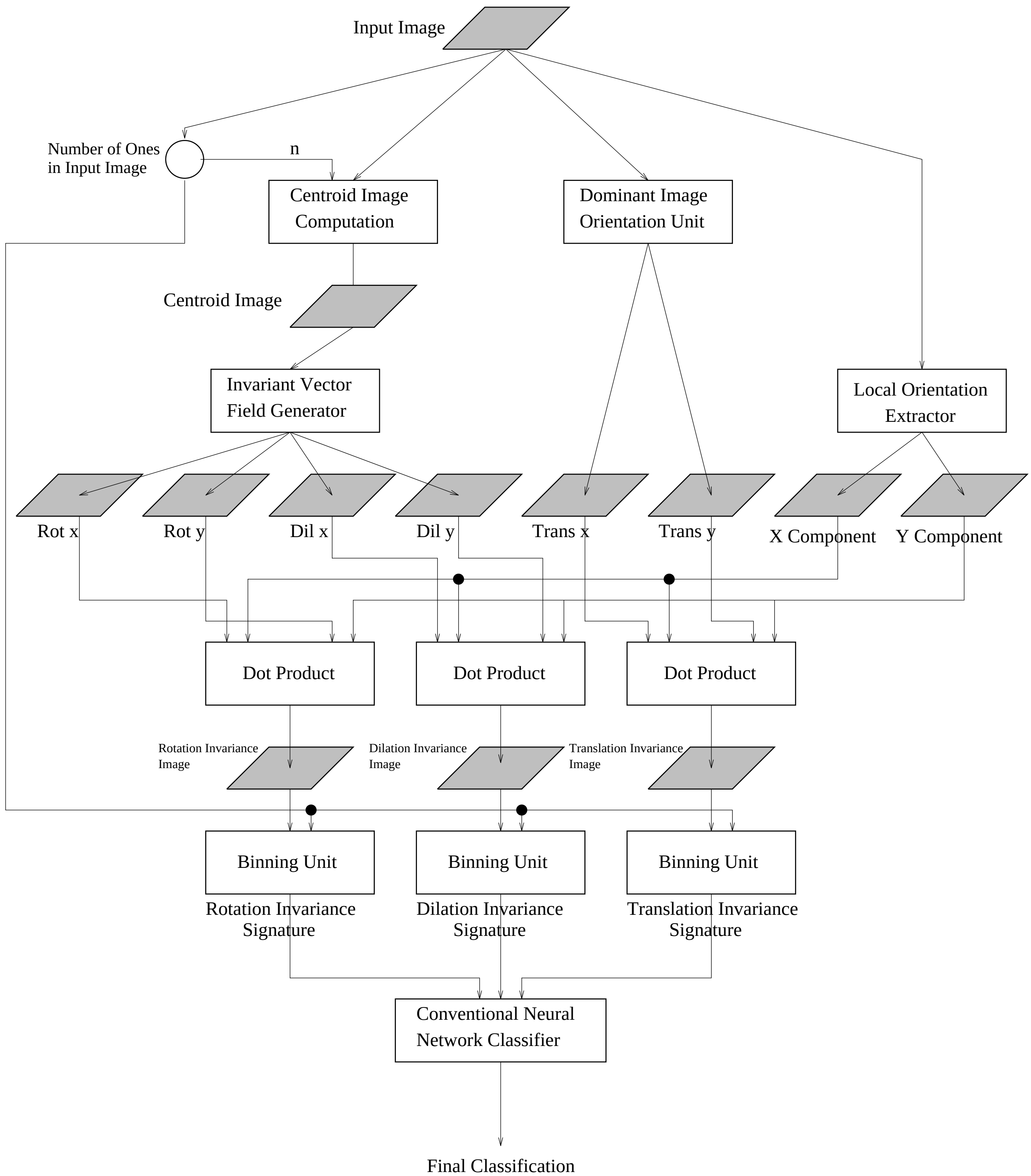

Figure 4: Invariance Signature-Based Contour Recognition System