(4)

AD

AD-E402 076

Technical Report ARFSD-TR-90008

A NEURAL NETWORK OBJECT RECOGNITION SYSTEM

F. P. Kuhl
Project Engineer
ARDEC

A. P. Reeves
R. J. Prokop
Cornell University

July 1990

US ARMY
ARMAMENT MUNITIONS
& CHEMICAL COMMAND
ARMAMENT RDE CENTER

U.S. ARMY ARMAMENT RESEARCH, DEVELOPMENT AND ENGINEERING CENTER

Fire Support Armaments Center

Picatinny Arsenal, New Jersey

Approved for public release; distribution unlimited.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER Technical Report ARFSD-TR-90008	5. MONITORING ORGANIZATION REPORT NUMBER
6a. NAME OF PERFORMING ORGANIZATION ARDEC, FSAC (cont) — 6b. OFFICE SYMBOL SMCAR-FSF-RC	7a. NAME OF MONITORING ORGANIZATION U.S. Army Research Office
6c. ADDRESS (CITY, STATE, AND ZIP CODE) Fire Control Division Picatinny Arsenal, NJ 07806-5000 (cont)	7b. ADDRESS (CITY, STATE, AND ZIP CODE) P.O. Box 12211 Research Triangle Park, NC 27709-2211
8a. NAME OF FUNDING/SPONSORING ORGANIZATION ARDEC, IMD STINFO Br — 8b. OFFICE SYMBOL SMCAR-IMI-I	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (CITY, STATE, AND ZIP CODE) Picatinny Arsenal, NJ 07806-5000	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (INCLUDE SECURITY CLASSIFICATION)
A NEURAL NETWORK OBJECT RECOGNITION SYSTEM

12. PERSONAL AUTHOR(S)
F. P. Kuhl, Project Engineer, ARDEC and A. P. Reeves and R. J. Prokop, Cornell University

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (YEAR, MONTH, DAY)	15. PAGE COUNT
Final	FROM Oct 88 TO Feb 90	July 1990	59

16. SUPPLEMENTARY NOTATION
Task was performed under a Scientific Services Agreement issued by Battelle, Research Triangle Park Office 200 Park Drive, P.O. Box 12297, Research Triangle Park, NC 27709

17. COSATI CODES			18. SUBJECT TERMS (CONTINUE ON REVERSE IF NECESSARY AND IDENTIFY BY BLOCK NUMBER)
FIELD	GROUP	SUB-GROUP	Neural networks Object recognition Target identification

19. ABSTRACT (CONTINUE ON REVERSE IF NECESSARY AND IDENTIFY BY BLOCK NUMBER)

A system involving neural networks has been developed to identify objects as an instance of a prespecified set of object classes. This system is useful for exploring different neural network configurations. There are three main computation phases of a model based object recognition system: segmentation, feature extraction, and object classification. This report focuses on the object classification stage. For segmentation, a neural network based segmenter developed by M. Tenorio can be used. Other, more conventional segmentation schemes are also available. Neural networks for feature extraction are at an early stage of development; other more conventional techniques have proved to be very fast and effective. Several conventional techniques are available with the current system. Neural network based feature extraction may be added at a later date. The classification stage consists of a very flexible neural network simulator which may be configured for a wide range of different network concepts. The system is based on the VISIX computer vision software environment. It provides an interactive experimentation facility that may be used on a large number of UNIX based workstations. The system has been tested with a number of simple object recognition tasks. (KR)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL I. HAZENDARI	22b. TELEPHONE (INCLUDE AREA CODE) (201) 724-3316 — 22c. OFFICE SYMBOL SMCAR-IMI-I

DD FORM 1473, 84 MAR

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

6a and 6b. (cont)

Cornell University
School of Electrical Engineering
410 Phillips Hall, Ithaca, NY 14853

CONTENTS

FIGURES

INTRODUCTION

The use of networks of simple neuron cells or the connectionist approach to computing has received much attention recently due to advances in the network teaching techniques. The basic concept of the neural network approach is that a computation function may be learned from example; that is by showing the network examples of the behavior of the desired function it will learn to respond to the desired behavior for new inputs.

The focus of this work is on the identification of an object's type from a set of image features. In the simplest form, this requires matching an input parameter vector (feature representation) to the closest vector that the network has been taught. New back projection learning algorithms have recently been been developed that can teach a network to implement an arbitrary complex decision surface in hyperspace; providing that a network has a sufficient number of neurons. These are two main research issues: how to present the vectors to the network and to encode the desired responses, and how to improve the learning algorithm. The problems with existing learning algorithms are: (1) convergence is not guaranteed even with sufficient neurons; (2) learning is very slow, typically thousands of iterations of the input vectors may be required; and (3) only an analytic surface can be taught whereas a probabilistic behavior may be more appropriate.

Neural Network Models

A taxonomy of neural models has been presented by Lippmann (ref 1). The taxonomy initially divides nets between those with binary inputs versus those with continuous inputs. The next level of distinction is based on whether the network is trained with our without supervision. Below this, networks are distinguished by their topology. A neural network model may therefore be characterized by three properties; the computational element, the network topology, and the training algorithm.

The computational elements of a neural network mode! are nonlinear analog modules sometimes referred to as neurons. A neuron is a multi-input single output device. The output is a nonlinear function of a weighted sum of the inputs. The nonlinear function is set for a given class of neural elements.

Various network topology strategies have been proposed that use combinations of single and multilayer as well as feed forward and feedback connections to achieve the desired results. For example, it has been established that a network of three neuron layers is capable of realizing an arbitrarily complex decision surface in hyperspace. However there is currently no general method for predicting how many neurons are required for a given task.

The training algorithm determines the initial values for the weights in all the neurons and then specifies how the weights should be updated during training to achieve the desired output. The main problem with current neural nets is that there isn't any good, robust training algorithms to reliably adjust to weights.

Application of Neural Networks to Object Recognition

The model based object recognition paradigm identifies an unknown object as a member of a set of known objects. Three fundamental stages that may be identified in most model based recognition systems are image preprocessing (segmentation), object feature extraction (representation), and matching (classification).

Model-based techniques utilize abstract representations to characterize objects. These representations are defined by measurable object features extracted from segmented object imagery and any a priori knowledge available. These features may be considered to define an n-dimensional feature space. The feature representations of an object then map to points in this feature space. A classification scheme is used to partition the feature space to decision regions based on training information and to identify unknown input vectors based on these regions.

The conventional classification schemes used for vision feature spaces are statistically based decision surfaces. For standard classification tasks, where class distributions may be modeled by Gaussian distributions, statistical decision techniques have been used. For complex decision surfaces the k-nearest neighbors technique has been found to be useful.

Neural network models have great potential for use in object recognition due to the number of parallel computations and decisions required for classification. The potential advantages of the neural network classification approach are:

Automatic adaptive behavior--It is possible that a neural network can be "taught" to modify its behavior if the task changes in time or new teaching information is acquired.

No formal model requirement--Most conventional techniques require that a strict model computation be established before an algorithm is developed. It is possible for a neural network to operate with very little formal structure.

Fast operation--It may be possible to use parallel hardware and a smaller number of operations to solve a problem with a neural network rather than a conventional algorithm.

It is important to note that none of the above advantages is assured even when a neural network is carefully engineered to an application. Furthermore, it is quite possible that neural networks will require more computation and will give inferior results to well established algorithms. If there is a good direct algorithm for solving a task, then the direct implementation of the algorithm is probably far superior to a neural network approach.

It has been shown that neural nets have applications in each stage of the object recognition process. For example, neural network models have been developed that can perform low-level image processing functions such as convolution integrals, Gaussian-filtering, and edge-enhancement (ref 2) that may be used in the preprocessing stage. More importantly, however, is the ability of neural networks to perform image segmentation. Such a neural network model has been developed by M. F. Tenorio (ref 3). Additionally, neural network models are being developed to perform invariant transformations such as the Hough transform (ref 2) that may be used for feature extraction. However, fast and robust feature extraction using neural networks is still at an early stage of development that requires further investigation.

Overview

This first section of the report serves as an introduction to neural networks and illustrates their application to model based object recognition. The second section is a presentation of the theory behind a neural net classifier developed for object recognition. Input encoding techniques are presented along with network topologies and training techniques. The third section describes the framework for a neural network object recognition testbed developed to explore different neural network configurations. Section four is a user's guide to installation and operation of the testbed. An example experiment, along with variations is presented. Appendix A is a collection of Unix style manual pages describing the neural network testbed modules. Appendix B is a selection of manual pages describing modules of the VISIX (ref 4) image processing system that are relevant to the use of the testbed.

NEURAL NETWORK CLASSIFIER

A neural network classifier has been developed for use in a model based object recognition system. The network is intended to receive inputs in the form of object feature vectors. The network will be trained with a representative set of feature vectors. The class of each of the training vectors is known a priori so that the training may be supervised. When a vector of unknown class is input, after the net has been trained, the output of the network is a binary vector that is an one-in-n indication of an object's class.

Input Encoding

Network input encoding is based on a feature vector extracted from an object image. Since an effective neural network model for object feature extraction is yet to be developed, more traditional schemes for object feature extraction were chosen. A desirable property of many existing object feature techniques is their invariance to changes in object to sensor distance, position, and orientation. In other words, it would be desirable for the abstract representation of a given object to be viewed from a given aspect to be the same regardless of the distance from the object to the sensor, the position of the object within the sensor's field of view, and the orientation of the sensor with respect to the object (referred to as size, position, in-plane rotation invariance, respectively). Several techniques have been presented that meet these invariance criteria. These include the techniques of standard moments (ref 5) and normalized Fourier descriptors (ref 6). These abstract object representations are more efficient than object images since they typically require approximately only 10 to 20 real numbers to represent an object. Additionally, they have well defined transform properties that allow simple object manipulation in the feature domain. A short description of each technique is presented.

Normalized or "standard" moments (ref 7) are based calculation of the two-dimensional Cartesian moments, m_{pq}, of the image, $f(x,y)$, given by

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x,y)\, dx\, dy \qquad p,q = 0,1,2,\ldots$$

Moments represent object features such as area, center of mass (COM), and principal axes. Moments are desirable features because they have well defined linear transform properties that allow simple object manipulation in the moment domain. For example, an object represented by the moment set, M_{pq}, may be scaled by a factor, α, using the following transform

$$m'_{pq} = \alpha^{2+p+q}\, m_{pq}\, \beta$$

Similarly, an object may be translated by α and β in the x and y directions respectively, using the following transform

$$m'_{pq} = \sum_{r=0}^{p} \sum_{s=0}^{q} \binom{p}{r} \binom{q}{s} \alpha^{p\,r} \beta^{q\,s} m_{rs}$$

And finally, an object may be rotated by an angle θ using the following transform

$$m'_{pq} = \sum_{r=0}^{p} \sum_{s=0}^{q} \binom{p}{r} \binom{q}{s} (-1)^{q-s} (\cos\theta)^{q+r-s} m_{p+q-r-s,\, r+s}$$

Invariance of the moment values is achieved through normalization of the object features in the moment domain using the transformation described above. Specifically, the normalized moment values are given in table 1.

Table 1. Moment normalization values

Standard moment	Normalization
$M_{00} = 1$	Object area set to 1
$M_{10} = 0$ $M_{01} = 0$	Object translated such that COM is at origin
$M_{11} = 0$	Major principal axis is aligned with x-axis
$M_{20} = M_{02}$	Object area is differentially scaled to make aspect ratio 1

Calculation of standard moments has actually been implemented in two different ways in the system. The moments may be calculated in a conventional manner from object silhouette and grey-level imagery. Optionally, a "fast" moment calculation is available that determines silhouette moment values based on the object boundary only.

The technique of normalized Fourier descriptors (ref 6) characterizes an object based on the contour of its silhouette. Consider a closed contour, representing an object boundary, in the complex plane. If it is traced once in the counterclockwise direction with constant velocity, v, a complex function, z(t) with parameter t is obtained. For consistency, v may be chosen so that the time required to traverse the contour is always 2π. Traversing the contour continuously yields a periodic function which may be expanded in a convergent Fourier series. A Fourier descriptor, F, of the contour is the complex Fourier series expansion of z(t).

$$F(n) = \frac{\pi}{2} \int_{0}^{\infty} z(t)\, e^{-jnt}\, dt$$

In practice, the contour is taken from a digitized image; therefore, z(t) is not available as a continuous function. If z(k) is a uniformly sampled version of z(t) of dimension N, the discrete Fourier transform gives us the N lowest frequency coefficients, F(0) • • • F(N-1), directly.

Invariance of Fourier descriptors is also accomplished through normalization. The frequency domain transformations that affect size, position, and orientation of the contour follow directly from the properties of the DFT. Size normalization is accomplished by dividing all F(n) by $|F(1)|$. Translation normalization is accomplished by setting F(0) = 0. Finally, in-plane rotation and starting point position are normalized by changing the phase of the coefficients such that F(1) and F(k), the next largest coefficient, have a phase of zero.

Either of these techniques may be used to create invariant object feature vectors as input encodings for the neural network. Previous research, however, has shown the moment technique to be more effective than Fourier descriptors in the presence of sensor noise.

Neuron Model

The simplest neuron has N inputs, x_n, each with a variable weight, w_n. Each neuron has a single output, y, that is a nonlinear function of the sum of the weighted inputs and a threshold, θ. The equation for this neuron is then

$$y = f\left(\sum_{n=0}^{N-1} w_n x_n - \theta\right)$$

where f is a nonlinear activation function that causes y to have a value typically between [-1,1] or [0,1]. Some example activation functions are hard limiters, thresholds, and sigmoid functions.

Network Topology

There are two aspects to the network topology: the number of layers and the strategy used to interconnect layers. The neural network classifier uses a feedforward design that is configurable in a single or multilayer connection as specified by the user. A typical interconnection strategy is to connect each neuron of a given layer to all outputs of the previous layer. This strategy is the one that has been used; however, the network simulator has the capability of implementing any intralayer strategy.

The number of layers directly effects the type of decision regions that may be formed by the network. Therefore, a configuration may be selected that is appropriate for the distribution of classes in the feature space. For example, a single layer, feed-forward network made up of the neurons described above with hard limiting activation functions, may be trained to create a decision region that is a half-space represented by a hyperplane in feature space. A two-layer network is implemented as a combination of decisions made by a set of single layer networks. Consequently, a two-layer network can create decision regions that are defined by the intersection of a set of half-spaces in feature space. The resultant decision spaces for a two-layer network are convex open or closed regions. Finally, a three layer, feedforward network is implemented as a combination of decision made by two-layer networks. A three-layer network, therefore, is all that is required to create arbitrary convex or concave closed decision regions in feature space.

Training Algorithms

The algorithm used for training in the neural network classifier is based on the delta rule. The delta rule is a supervised learning procedure that relies on pairs of input and desired output encodings to update weight values. This technique additionally requires a continuous differentiable activation function. The procedure for training the network using the delta rule follows.

Initially, all the weights are set to a small random value. Then, for each training input, the network first produces a computed output vector. This computed output is then compared to the desired output vector to determine the error. If the computed output and desired output are equal (zero error), then no changes are made to the weights. If the error is not zero, then the weights are updated to reduce the difference between the computed and desired output.

Given a single layer, fully connected network with N inputs and M neurons (outputs), the input vector, x_n, the computed output vector, y_m, and the desired (training) output vector, t_m, the rule for updating the input weights, w_{nm}, is

$$\delta_m = t_m - y_m$$

$$\Delta_{nm} = \eta \, \delta_m \, x_n$$

$$w'_{nm} = w_{nm} + \Delta_{nm}$$

where w'_{nm} is the updated weight and η is a positive gain factor chosen to be less than 1.0. This rule is applied for all pairs of input and desired output vectors repeatedly until Δ_{nm} is zero for all weights and all input vectors.

The delta rule is derived from finding a set of weights that minimize the error between calculated and desired output using a gradient descent method operating on the error surface in weight space. This derivation is given in reference 8.

The delta rule may be extended to multilayered feedforward networks using a technique known as back propagation. This technique first computes an output vector, y_m, by propagating the input, x_n, completely through the multilayer network. The next phase is a backward pass through the network during which the weights at each layer are updated. This requires the determination of error vectors, δ, at each layer of the net. The weights at each layer are then updated according to the delta rule

$$\Delta_{nm} = \eta\, \delta_m\, x_n$$

$$w'_{nm} = w_{nm} + \Delta_{nm}$$

where x_n is an input at the current layer (an output of the previous layer). Initially, δ is computed for the final (output) layer of the network. The equation for δ at the output layer in a multilayered network is given by

$$\delta_m = (t_m - y_m) f'(\mathrm{net}_m)$$

where f' is the derivative of the activation function and net_m is given by

$$\mathrm{net}_m = \sum_n w_{nm}\, x_n$$

where n is over all the inputs to the neuron m. This δ is then used to update the weights in the final layer of the net. Next, the error for the preceding layer is computed. The error for this layer, δ', may be computed recursively from the previously computed error, δ, using

$$\delta'_m = f'(\mathrm{net}_m) \sum_n \delta_n\, w_{nm}$$

where n is over all nodes in the previous layer. This error, δ', is used to update the weights at this layer of the net. This process is repeated for each preceding layer until the first layer is reached. A complete derivation of this procedure is given in reference 8.

The value of η, the learning rate, must be chosen to provide a fast convergence without leading to oscillation. A momentum term may be added to the delta rule to help prevent oscillations. Weights are updated using a momentum factor, α, by the following rule

$$w'_{nm} = w_{nm} + \eta\, \delta_m\, x_n + \alpha(w_{nm} - w'_{nm})$$

The effect of alpha is to filter out high-frequency variations in the error surface during gradient descent. These high-frequency variations cause oscillations when larger weight steps (η) are taken.

Output Encoding

The neural network classifier employs a one-in-N output encoding technique that uses one neuron per class in the output layer of the net. The network is trained so that only one output neuron fires for a given class. The output vector is therefore a binary vector with each bit position representing a different class. The output vector is therefore a binary vector with each bit position representing a different class.

Comparison with Traditional Classifiers

The classifier previously described may be referred to as a Back-Propagation Classifier (ref 9). This may be compared to conventional classifiers that are typically probabilistic (Bayesian) or exemplar (k-nearest neighbor).

Probabilistic classifiers assume a priori class distributions in feature space. These distributions are estimated using supervised training data *assuming all the training data* are available simultaneously. These classifiers provide optimal performance when sufficient training data are available to arrive at an accurate estimation of the class distribution or at least an estimate that is consistent with the distribution of the test data.

Exemplar classifiers perform classification based on the identity of the training vector that is nearest to the test vector. The k-nearest neighbors may be found by determining the k minimum Euclidean distances between the test vector and the set of training vectors in feature space. These classifiers require a minimum training time but require significant memory and computation time to perform actual classification.

In general, back-propagation classifiers are implemented using single or multilayer neural networks with sigmoidal activation functions. They use supervised, gradient-descent training techniques that minimize the error between the calculated and desired output (as described above). These classifiers are characterized by long training times that increase with the number of layers and, in turn, the complexity of the decision regions formed. Additionally, no method exists for determining the correct number of nodes required in the intermediate layers of multilayer networks to form required complex decision regions. However, once a network is trained, actual classification may be performed very quickly by simply propagating the input vector forward through the net.

OBJECT RECOGNITION TESTBED

An object recognition system using neural network techniques has been developed. A modular approach has been taken to allow the system to be used as a testbed for performance comparison of neural network and conventional techniques throughout the recognition process. The task of this system is the recognition of unknown objects from silhouette and grey-level imagery based on similarity with imagery of known objects. Such a system may be partitioned into three stages: image segmentation, input encoding, and neural network classification (fig. 1).

Image Segmentation

The preprocessing of the input imagery should result in a set of distinct object images that will be used to generate feature vectors (object representations) for training (known objects) or classification (unknown objects). As mentioned previously, this stage may be accomplished using a neural network. Such a network has been developed which uses a single layer feedback network scheme (ref 3). Alternatively, the user may utilize any segmentation algorithm desired for performance comparison.

Input Encoding

Input encoding is accomplished using object feature extraction techniques to create input vectors for the training and testing the network. There are currently two major feature extraction techniques available in the system, normalized (standard) moments and normalized Fourier descriptors. These features may be normalized with respect to scale, translation, and in-plane rotation of the object. This normalization is configurable. In the test performed here normalization has been made with respect to all the above parameters. The modularity of the system also allows the user to specify and install custom input encoding techniques.

Neural Network Classification

The classification stage is implemented with a user configurable neural network that allows the user to specify such network parameters as:

Number of layers in the network

Number of neurons per layer

Number of inputs per neuron

Network interconnection topology

Neuron type

Output encoding

Training strategy

These parameters are easily changed to perform performance comparison of different neural network techniques on the same recognition task.

Process and File Structure

A flow graph of the processes used and the files created during training and testing of the neural network object recognition system is shown in figure 2.

To train the system, an input image is first segmented using a user specified segmentation algorithm. A set of image segments is then extracted with one object per image. The operator may then interactively assign the class identifiers to each of the segmented objects to supervise the training process. A feature vector is generated for each labeled (operator classified) segmented object. This procedure is repeated for all input images. The set of feature vectors of all the training object images is then used to train the neural network. The desired output of the neural net for each training input is automatically determined by the class assigned to each object by the operator.

To test the system, the operator follows the same basic procedure as for training. In this case, however, the operator does not have to interactively label the test objects. The system will automatically number the test object sequentially if desired. The set of feature vectors of all the test object images is then used to test the neural network. Each test input vector is applied to the network to produce a classified output encoding. To evaluate the classification results, the user may request the system to display the original test imagery with the determined classes overlayed on the objects.

TESTBED USER'S GUIDE

Installation

The user should first change directory to the location for the neural network. For purposes of illustration we will refer to this directory as NNET. Load the tape into this directory using the command

```
tar -xvbf 126/dev/rst0
```

Directory Structure

After the tape has been loaded the user should see the following subdirectories in the NNET directory (there may be additional files/dirs):

images/ man/ nbin/ nnet/ nn_setup prep/ vbin/

The contents and use of each of these directories is as follows:

images--location for the input imagery used to train and test the network

man--location of the neural network module manual pages

nbin--location of the neural network module executables

nnet--location of the neural network configuration files

pep--directory used by the neural network modules for the creation of intermediate data files

vbin--location of the visix executables

nn_setup--shell script that is used to set the environment variables (see below)

Paths to Executables

In order to run the neural net modules, the user must have the following directories in his path:

NNET/nbin NNET/vbin

Environment Variables

The neutral network modules use several environment variables to determine the system configuration. These variables are automatically set using a special file called nn_setup.

The specific variables set within nn_setup and their meaning are:

NN_DISPLAY--specifies the frame buffer type (COLOR|MONO|-TEK4010)

NN_DISPROC--specifies the image display procedure (disdl)

NN_IMPATH--specifies the path to image directory (NNET/images)

NN_NETPATH--specifies the path to net configuration directory (NNET/net)

NN_PREPATH--specifies the path to preprocessing directory (NNET/prep)

NN_CONFIG--specifies the root name of configuration file (net1)

NN_CLASSES--specifies the number of classes for one-in-n classification(3)

NN_SEGMENT--shell script to run image segmentation program (nn_seg)

The initial values of these variables are shown in parenthesis. They may be changed to suit the users needs by editing nn_setup.

!!! IMPROTANT !!!

In order for the values set in nn_setup to *take effect* upon execution of any of the nnet modules, the user must have a copy of nn_setup in the current working directory.

Startup and Execution

Many of the network modules need to display imagery during their execution. Therefore, the network modules should only be run from within a graphical window environment such as Sunview or X.

On Sun workstations, special Sunview startup and root menu files are provided in the files

NNET/.sunview   NNET/.rootmenu

The use of these files is optional. They only provide an organized window configuration for the neural network programs. If Sunview is invoked using the startup files provided, the screen will be configured as shown in figure 3.

Neural Network Modules

There are three modules typically invoked directly by the user to perform neural network experiments:

netrun--interactive script that invokes several sub-modules ot test and train the neural net

netpic--module used to display labeled versions of the unknown imagery after classification

show--module that allows the user to view the original input imagery as well as the intermediate data and image files created by the network

It is usually not necessary for the user to invoke any of the neural network system submodules, (NNET/nbin/nn_. . .), directly.

Example Object Recognition Experiment

To illustrate the use of the neural network object recognition system, a simple recognition experiment is performed.

In this experiment it is assumed that there are images containing known objects in the file:

NNET/images/default.im

and several unknown images whose contents we wish to classify:

NNET/images/fulrak.im...scene.im...scene2.im

In addition, input coding will be performed using standard moments. The use of a particular input encoding requires a specific network configuration. An example neural network configuration file for standard moments is given in the file:

NNET/nnet/net1.mts.con

Note that "net1" is the identifier for this configuration as specified by the NN_CONFIG environment variable set in nn_setup. The input encoding is designated by the intermediate extension ".mts". The ".con" extension indicates that this is a nnet configuration file. This file will be used to configure the neural network for the purposes of this experiment.

The experiment is performed in two basic stages:

1. Train the neural network using default.im

2. Test the neural network with fulrak.im, scene.im scene2.im

Training the Net

The command to train the neural network for this experiment is

```
netrun verb fvec=mts default.im
```

A listing of the output is shown in figure 4.

Initially, netrun displays the neural network system parameters as well as the contents of the current configuration file as a sanity check (fig. 4).

Next, netrun describes each stage of preprocessing (fig. 4) by indicating which intermediate file is being created. At one point, netrun will display the input image with a default set of labels (fig.5). The user is asked to relabel the objects by specifying a list of classes based on the order of the labels in the default image. The results of this labeling is shown in figure 6.

Netrun finally creates an input vector and trains the net (fig. 4). Upon convergence, netrun displays the number of iterations required.

Testing the Net

The command for testing the neural network in this experiment is

```
netrun test verb fvec=mts fulrak.im scene.im scene2.im
```

The listing of the output is shown in figure 7.

Again, netrun displays the neural network system parameters as well as the contents of the current configuration file as a sanity check (fig. 7)

Next, preprocessing is performed as for training, with the exception that, in this case, the user is not prompted for object labels (fig. 7).

Finally, netrun creates an input vector and applies it to the trained net. A tabular listing is displayed showing the unknown object number and the determined (found) class (fig.7). The user may now request a graphical depiction of the classification results with the command

```
netpic mts
```

This will first display an image "key" to associate class numbers with known objects (fig.8). Next, each test (unknown) input image is displayed with the determined class numbers for each of its segments (figs. 9 through 11).

Once the network has been trained, the user may run the test stage on any unknown images. Additionally, the show command may be used to view the contents of any of the intermediate files.

Neural Network Environment Variations

The neural network system has been designed to allow the user to easily vary the experiment to facilitate performance comparison between different techniques at various stages of the recognition procedure.

Changing Input Encoding

The same experiment could be performed as described above using normalized Fourier discriptors instead of moments by simply changing the occurrances of mts in the example to fd. Specifically,

```
netrun verb fvec=fd default.im

netrun test verb fvec=fd fulrak.im scene.im scene2.im

netpic fd
```

Note that the system will now configure itself based on the file

```
NNET/nnet/net1.fd.con
```

The input encodings currently supported are:

mom--fast silhouette moments generated from image boundary

mts--standard silhouette and grey moments

fd--normalized Fourier descriptors

vec--a "quick" vector

Note that the show command may be used to display the vectors of any of these encodings.

Changing the Network Configuration File

The user may configure the neural network by creating a file

```
"config_name"."fvec".con
```

Note that the configuration file is usually appropriate only for a specific type of input encoding. As an example, this is the contents of the configuration file "net1.mts.con":

```
UU.VISIX: <This is a comment area>

1   The program version (network type)
1   The number of networks (layers)
11  The number of inputs to each neuron
3   The number of neurons
1   The interconnectin pattern, 1=Full
1   The neuron type: 1-threshold, 2-rap, 3=sigmoid
```

This file specifies a single layer, fully connected, 3 neuron network with 11 inputs (the input vector length). After parameters in this file are set to the desired values, this new configuration may be used by editing nn_setup to

```
setenv NN_CONFIG "config_name"
```

Changing the Segmentation Algorithm

Another variation might be to use an alternative segmentation algorithm. This is accomplished by changing the value of NN_SEGMENT in nn-setup. Currently, a second shell script using an alternative segmentation algorithm developed by M. F. Tenorio is available. This module is called nn_ten3. To install this module, edit nn_setup and set

```
setenv NN_SEGMENT nn_ten3
```

The user may now rerun the experiment as above to see the effect of the new segmentation program.

In general, to create a new segmentation module, the user specifies a shell script that takes a root file name as input. The input image to the user's segmentation script will be in

```
NNET/images/"root".im
```

The new segmentation script should call the user's algorithm and create the segmented image in

```
NNET/prep/"root".seg
```

To install this new script, edit nn_setup and set

```
setenv NN_SEGMENT "user's script name"
```

Object Classification Using the Vicom

A late addition to the neural network environment is the ability to test and train the network based on images captured by the Vicom system. To use the current Vicom image, the user need only enter the name VICOM as one of the inputs to netrun. For example, once the net has been trained with the appropriate objects, the objects currently in the vicom field of view may be classified by using the command

```
netrun test verb fvec=mts VICOM
```

This will cause the system to automatically grab the current vicom image and store it in a file called

```
NNET/images/vicpic.im
```

This file will then be applies to the neural network as any other image. Note that the user must rename this file if it is to be saved for future use since this "vicpic.im" is overwritten each time a new image is grabbed.

Short-Cutting Image Preprocessing

The neural network object recognition system has been designed to allow the user to skip any of the image preprocessing steps if the appropriate intermediate data already exists. For example, once an experiment has been executed, input encoding files (moment ferature vectors) now exist for the input imagery. Therefore, it is not necessary for the user to go through the entire image preprocessing stage again to run the experiment. Specifically, when the following experiment is run

```
netrun test verb fvec-mts fulrak.im
```

the system creates a the input encoding file

```
fulrak.mts
```

Now, if the user wishes to re-execute the test, use

```
netrun test verb fvec=mts fulrak.mts
```

to prevent regeneration of the moment feature vectors.

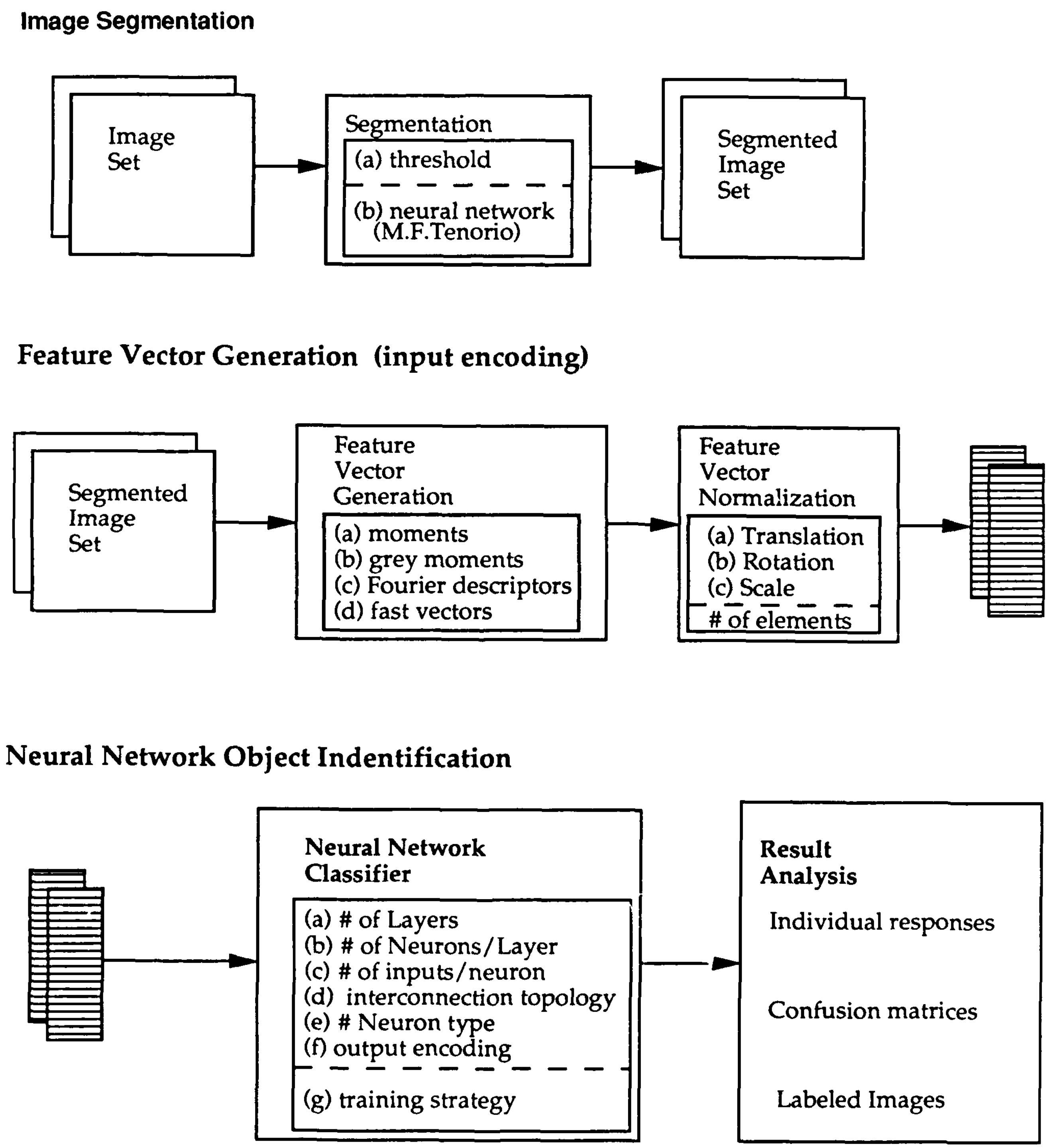


Figure 1. Neural network object identification system

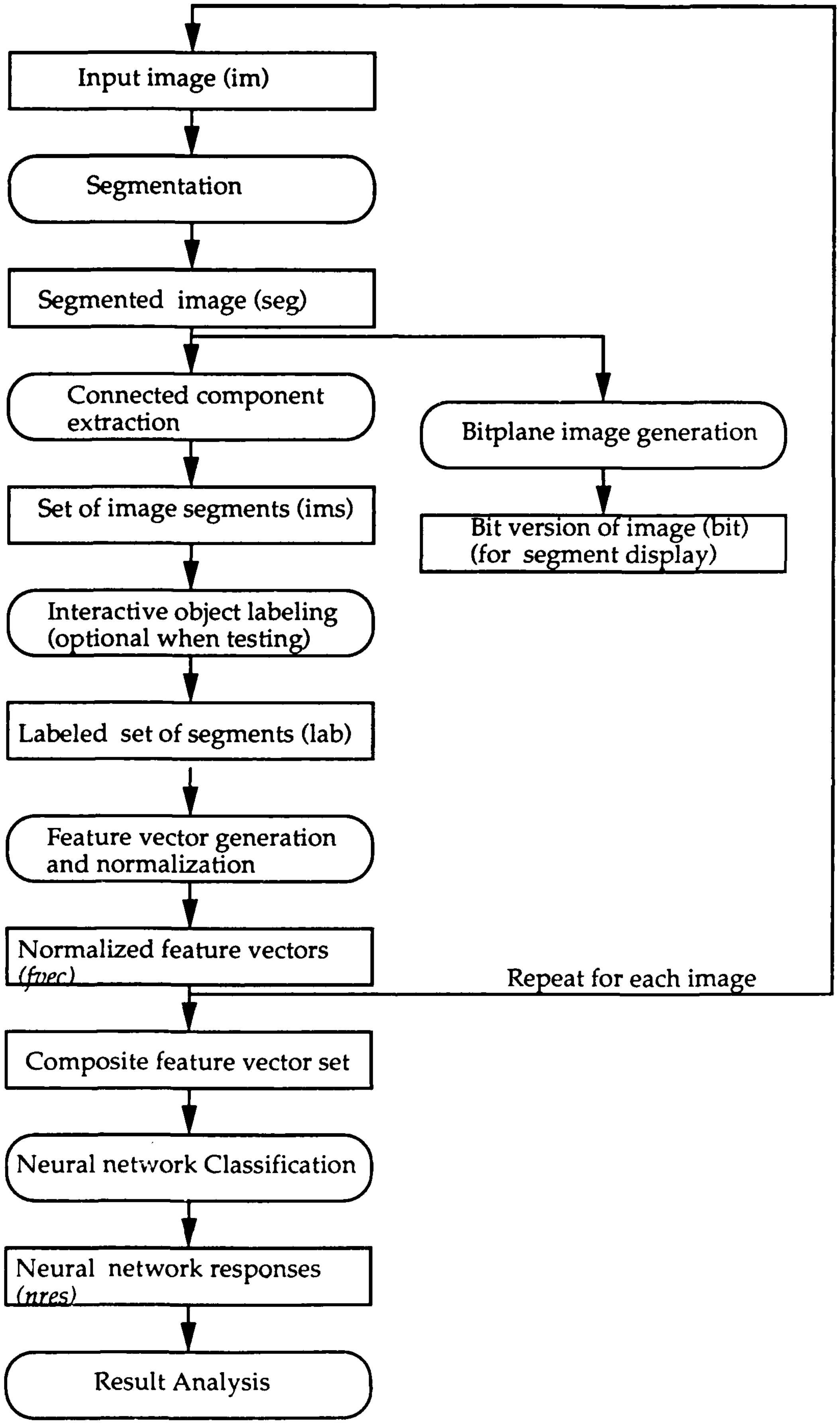
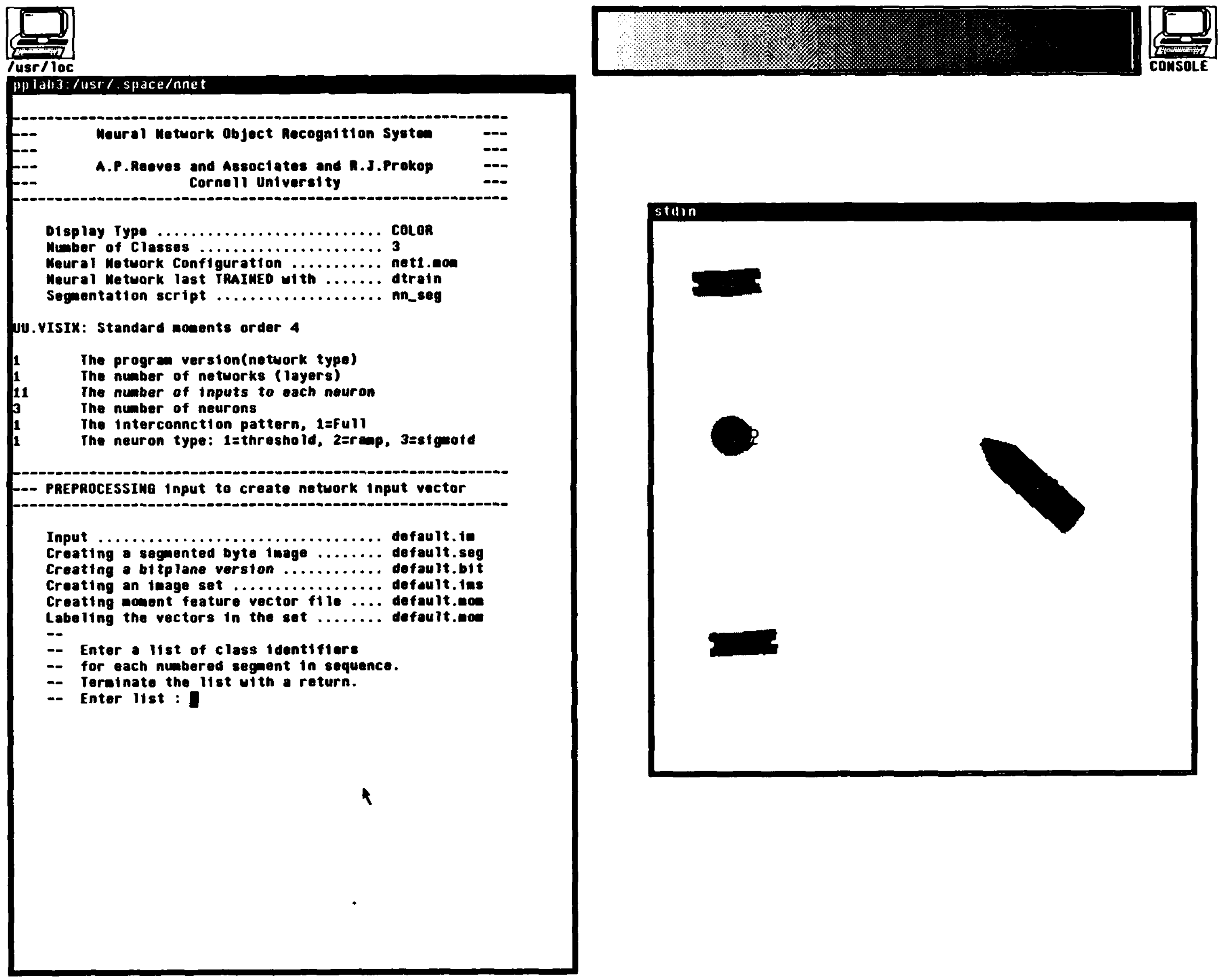

Input image (im)
Segmentation
Segmented image (seg)
Connected component extraction
Bitplane image generation
Set of image segments (ims)
Bit version of image (bit) (for segment display)
Interactive object labeling (optional when testing)
Labeled set of segments (lab)
Feature vector generation and normalization
Normalized feature vectors *(fvec)*
Repeat for each image
Composite feature vector set
Neural network Classification
Neural network responses *(nres)*
Result Analysis

Figure 2. Process and file structure for neural network object identification



```
/usr/loc
pplab3:/usr/.space/nnet

---------------------------------------------------------------
---        Neural Network Object Recognition System          ---
---                                                          ---
---        A.P.Reeves and Associates and R.J.Prokop          ---
---                  Cornell University                      ---
---------------------------------------------------------------

    Display Type ........................... COLOR
    Number of Classes ...................... 3
    Neural Network Configuration ........... net1.mom
    Neural Network last TRAINED with ....... dtrain
    Segmentation script .................... nn_seg

UU.VISIX: Standard moments order 4

1       The program version(network type)
1       The number of networks (layers)
11      The number of inputs to each neuron
3       The number of neurons
1       The interconnction pattern, 1=Full
1       The neuron type: 1=threshold, 2=ramp, 3=sigmoid

---------------------------------------------------------------
--- PREPROCESSING input to create network input vector
---------------------------------------------------------------

    Input .................................. default.im
    Creating a segmented byte image ........ default.seg
    Creating a bitplane version ............ default.bit
    Creating an image set .................. default.ims
    Creating moment feature vector file .... default.mom
    Labeling the vectors in the set ........ default.mom
    --
    --  Enter a list of class identifiers
    --  for each numbered segment in sequence.
    --  Terminate the list with a return.
    --  Enter list : █
```

Figure 3. Sunview screen configuration

```
    Display Type .......................... SUNGREY
    Number of Classes ..................... 3
    Neural Network Configuration .......... net1.mts
    Neural Network last TRAINED with ...... default
    Segmentation script ................... nn_seg

UU.VISIX: Standard Moments order 4

1       The program version(network type)
1       The number of networks (layers)
11      The number of inputs to each neuron
3       The number of neurons
1       The interconnction pattern, 1=Full
1       The neuron type: 1=threshold, 2=ramp, 3=sigmoid




-----------------------------------------------------------
--- PREPROCESSING input to create network input vector
-----------------------------------------------------------

    Input ................................ default.im
    Creating a segmented byte image ........ default.seg
    Creating a bitplane version ............ default.bit
    Creating an image set .................. default.ims
    Labeling the images in the set ......... default.ims

              [--- see figure 5     (ed.) ---]
    --
    --  Enter a list of class identifiers
    --  for each numbered segment in sequence.
    --  Terminate the list with a return.
    --  Enter list : 3 2 1 3
    --  Is this labeling correct? (y/n) : y
              [--- see figure 6     (ed.) ---]
    --
    Creating moment feature vector file .... default.mts

    Creating the network input data ........ input.mts

-----------------------------------------------------------
--- TRAINING the Neural Network
-----------------------------------------------------------

    Training data : default

    Neural Network Log :

UU.VISIX:Neural Network Log File

687     - the number of iterations for convergence.
```

Figure 4. Listing of training output

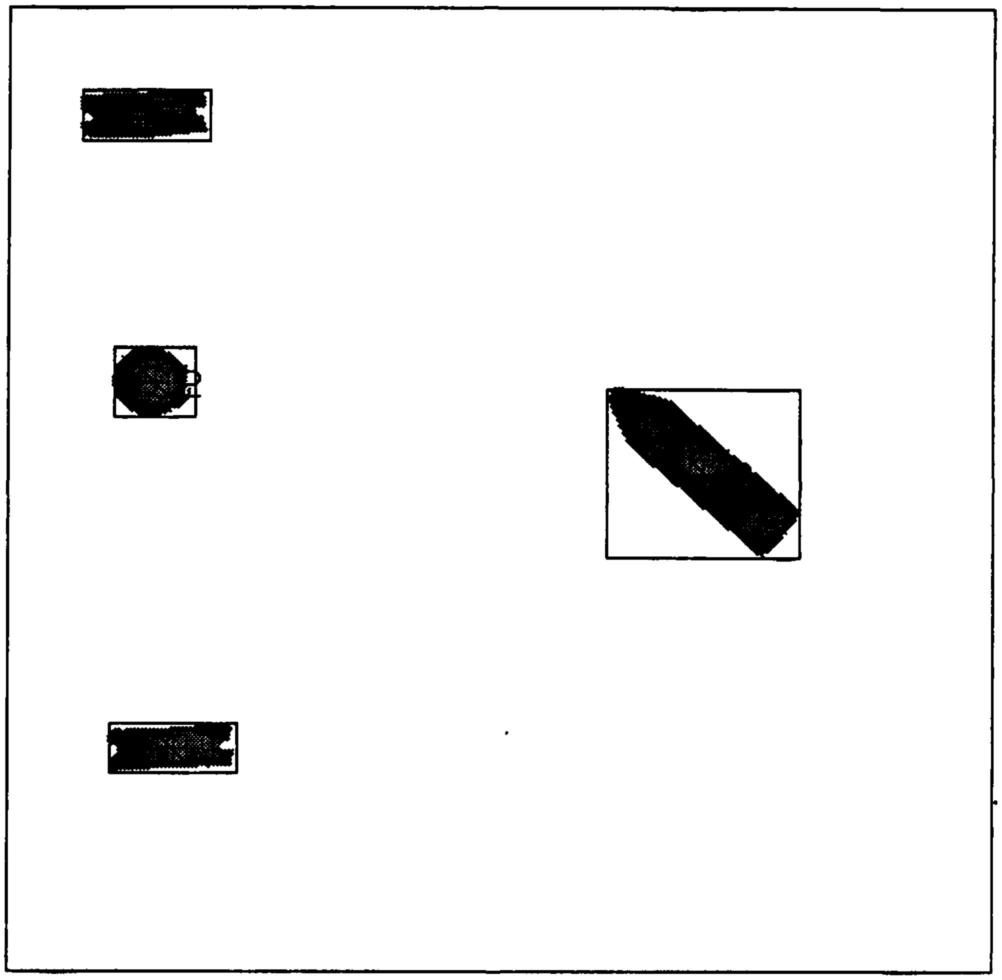


Figure 5. Segmented input image used for training (default)

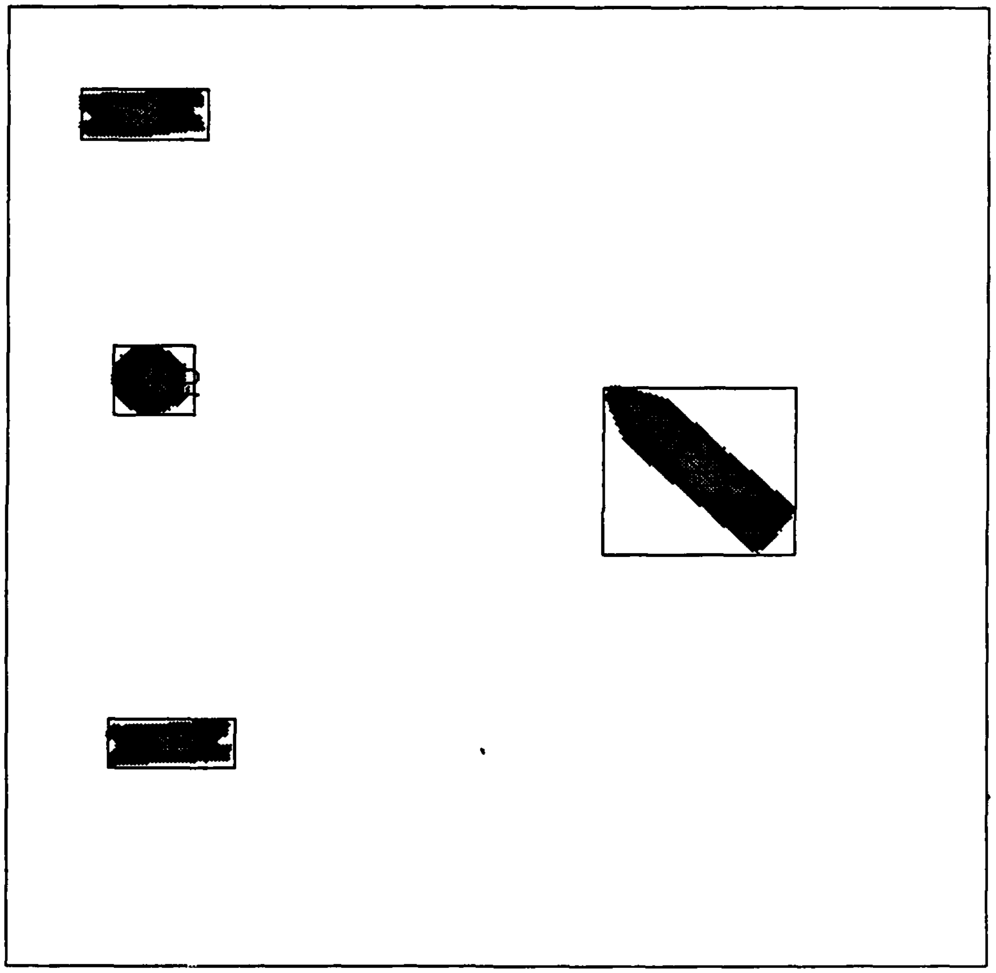


Figure 6. Labeled input image used for training (default)

```
    Display Type .......................... SUNGREY
    Number of Classes ..................... 3
    Neural Network Configuration .......... net1.mts
    Neural Network last TRAINED with ...... default
    Segmentation script ................... nn_seg

UU.VISIX: Standard Moments order 4

1       The program version(network type)
1       The number of networks (layers)
11      The number of inputs to each neuron
3       The number of neurons
1       The interconnction pattern, 1=Full
1       The neuron type: 1=threshold, 2=ramp, 3=sigmoid
```

```
-------------------------------------------------------------
--- PREPROCESSING input to create network input vector
-------------------------------------------------------------

    Input ................................. scene.im
    Creating a segmented byte image ........ scene.seg
    Creating a bitplane version ............ scene.bit
    Creating an image set .................. scene.ims
    Creating moment feature vector file .... scene.mts

    Input ................................. scene2.im
    Creating a segmented byte image ........ scene2.seg
    Creating a bitplane version ............ scene2.bit
    Creating an image set .................. scene2.ims
    Creating moment feature vector file .... scene2.mts

    Input ................................. fulrak.im
    Creating a segmented byte image ........ fulrak.seg
    Creating a bitplane version ............ fulrak.bit
    Creating an image set .................. fulrak.ims
    Creating moment feature vector file .... fulrak.mts

    Creating the network input data ........ input.mts
```

Figure 7. Listing of test output

```
------------------------------------------------------------
--- TESTING the Neural Network
------------------------------------------------------------

    Object  Class Found Misfire

       1      1     1     0
       2      2     1     0
       3      3     1     0
       4      4     1     0
       5      5     3     0
       6      1     1     0
       7      2     2     0
       8      3     2     0
       9      4     1     0
      10      5     1     0
      11      6     3     0
      12      1     1     0
      13      2     2     0
      14      3     2     0
      15      4     2     0
      16      5     3     0

------------------------------------------------------------
--- Classification Complete.
------------------------------------------------------------

    To display labeled images enter > netpic mts
```

[-----   see figures 8 through 11 (ed.) ----]

Figure 7. (cont)

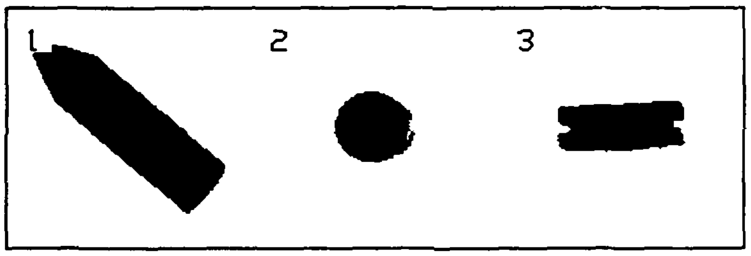


Figure 8. Classification key image

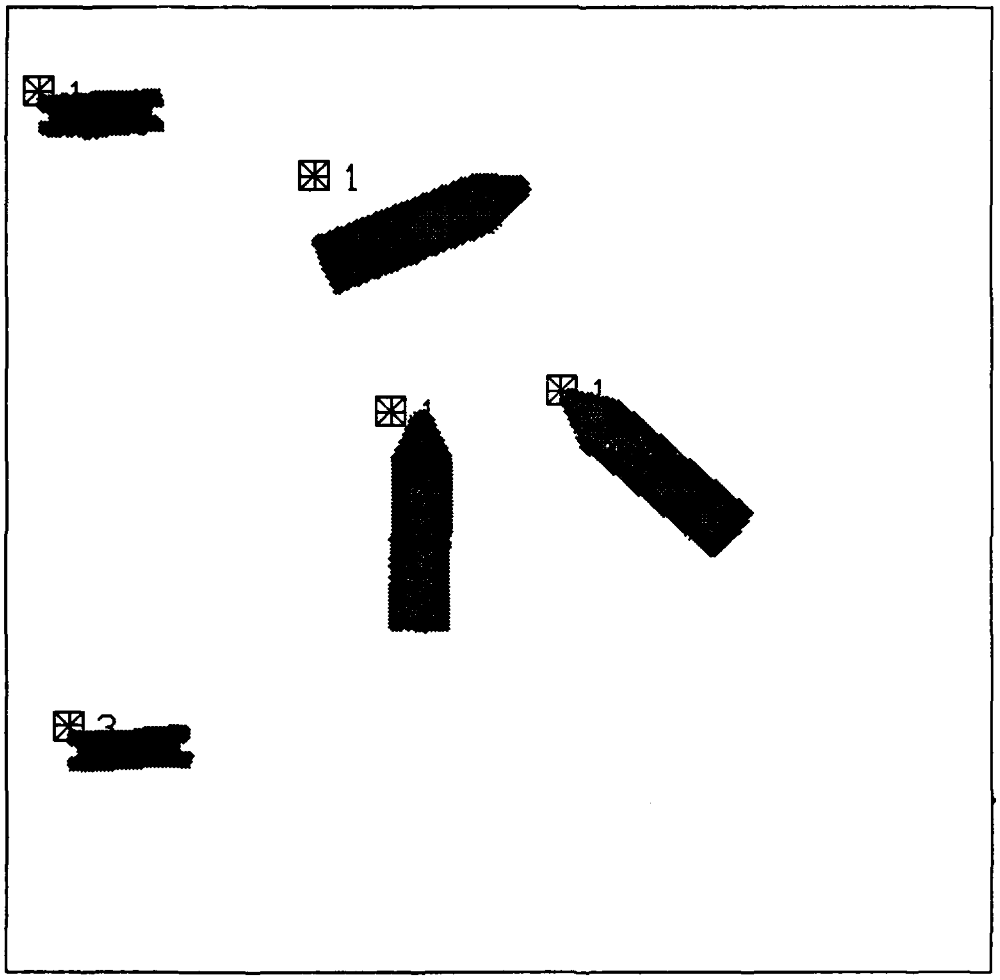


Figure 9. Labeled test image (scene)

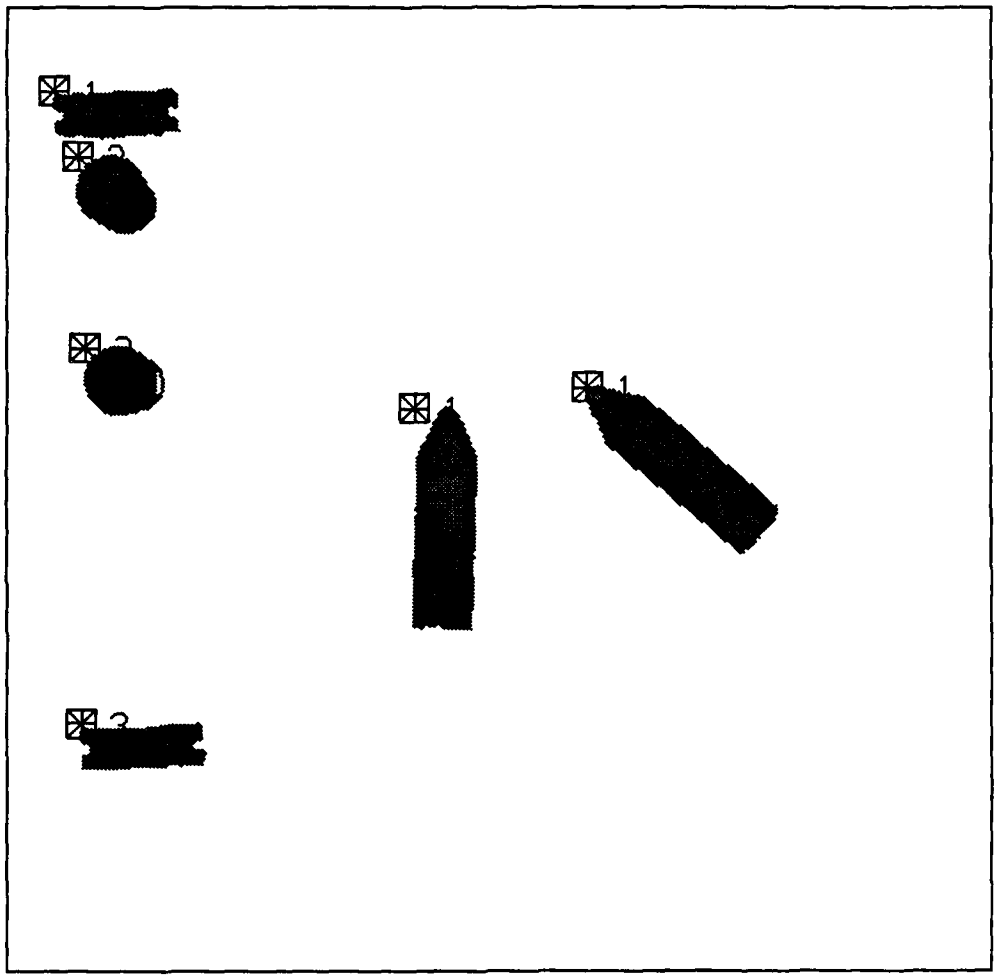


Figure 10. Labeled test image (scene2)

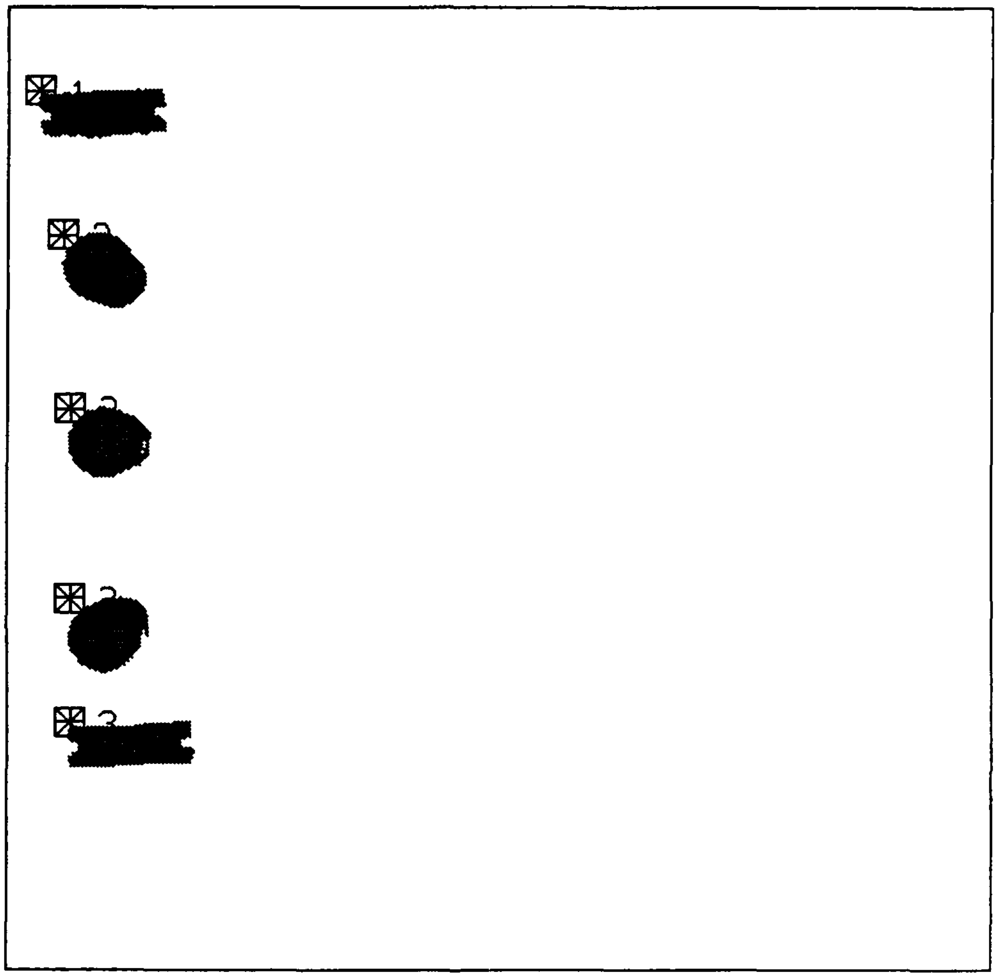


Figure 11. Labeled test image (fulrak)

REFERENCES

1. Lippmann, R. P., "An Introduction to Computing with Neural Nets," IEEE ASSP Magazine, pp 4-21, April 1987.

2. Roth, M. W., "Survey of Neural Network Technology for Automatic Target Recognition," IEEE Transactions on Neural Networks, vol 1, no. 1, pp 28-43, March 1990.

3. Tenorio, M. F., "Serial and Parallel Image Segmentation Based on MAP Estimation Techniques," Final Report D.O. 0779 C.N. DAAL03-86-D-0001, Armament RD&E Center, Picatinny Arsenal, NJ, August 1988.

4. Reves, A. P. and Associates, "VISIX 2.0 Users Manual," VISIX: A Computer Vision System for UNIX based Computer Systems, July 1989.

5. Reeves, A. P., Prokop, R. J., and Andrews, S. E., "Three Dimensional Shape Analysis Using Moments and Fourier Descriptors," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol 10, no. 6, pp 937-943, November 1988.

6. Wallace, T. P. and Wintz, P.A., "An Efficient Three-Dimensional Aircraft Recognition Algorithm Using Normalized Fourier Descriptors," Computer Graphics and Image Processing, vol 13, pp 99-126, 1980.

7. Reeves, A. P. and Rostampour, A., "Shame Analysis of Segmented Objects Using Moments," IEEE Computer Society Conference on Pattern Recognition and Image Processing, pp 171-174, Augusg 1981.

8. Rkumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning Internal Representations by Error Propagation," Parallel Distributed Computing, pp 318-362, MIT Press, 1986.

9. Lippmann, R. P., "Pattern Classification Using Neural Networks," IEEE Communications Magazine, pp 47-64, November 1989.

APPENDIX A

NEURAL NETWORK OBJECT RECOGNITION MODULES

NAME

netrun – run and test a neural network

SYNOPSIS

netrun [test] [time] [check] [verb] [config] [fvec=<type>] input-list

DESCRIPTION

Netrun trains or tests a neural network classifier with segments extracted from a set of images. If a list of images is not specified on the command line, netrun prompts the user for a list of image file names. A composite feature vector of all the segments from all the files is generated and applied to a neural network.

The full procedure for processing an image, *name.im*, into a neural network input vector file, *name.fvec* along with intermediate files, is as follows :

1. partition the image into segments *(name.seg)*. using the utility *nn_seg*.
2. generate an image set of these segments *(name.ims)* using the utility *nn_ims*.
   and generete a bitplane version of the image *(name.bit)* using the utility *nn_bit*.
3. generate a feature vector file *(name.mom, name.fd, name.mts, or name.vec)*
   containing one vector per segment
   using the utility *nn_mom, nn_fd, nn_mts, or nn_vec* respectively.
4. in training mode, prompt the user to supply labels for each segment
   and update *(name.ims)* and *(name.<fvec>)* with this information using the utility *nn_lab*.

If more than one image is specified, each is processed (steps 1-4) in turn.

5. All feature vector files are then concatenated to create a single neural network
   input file, *input.<fvec>*.
6. The input file is applied to *nnet(1)* for for training or testing.
7. In testing mode, the output of nnet is directed through for a tabular listing of *anet(1)*
   classification results.
8. At this point, in testing mode, a graphical representation of classification
   results may be generated using the utility *netpic*

<fvec>

Image processing steps can be skipped by specifiying an intermediate filename extension with the filename. For example, to skip the segmentation stage, specify the file by the name *name.seg* rather than *name.im*. To skip the interactive labeling stage use the extension *.lab*. This is useful when it is desired to compare the results of different feature vector formulations.

OPTIONS

test Test the network instead of train it. The result for each segment will be output.

verb Set the verbose mode. All proccesing stages will be preceeded with a messge to the terminal.

time Print the time required for each processing stage.

check The commands that would be executed are printed to the terminal. No data is actually processed.

config Display the current system parameters and network configuration. No data is processed.

fvec=<type>
The feature vector type to be used may be specified. Current types are :

mom	- fast moment generation (silhouette moments from chain code)
mts	- slower moment generation (silhouette and grey moments)
fd	- Fourier descriptors

vec - quick vector

ENVIRONMENT

Netrun is a large shell script that calls the utilities described in nn_utils(1). The location of these utilities must be in the user's path.

Netrun requires three special subdirectories for its operation. The actual names for these directories are specified using environment variables. (see below)

The first is the "image" subdirectory. This directory is the location of the input imagery.

The second is the "preprocessing" subdirectory. Netnet uses this directory to create the intermediate files described above.

The third is the "net" subdirectory. This directory is the location of the neural network configuration and data files. A network configuration is specified in a file *<name>.<fvec>.con*. The configuration name and feature vector type to use in a given experiment may be specified using the environment variables described below.

The environment variables used by the neural network to determine the system configuration are :

NN_IMPATH
specifies the path to "image" subdirectory

NN_NETPATH
specifies the path to "net" subdirectory

NN_PREPATH
specifies the path to "preprocessing" subdirectory

NN_CLASSES
the number of classes for one-in-n classification

NN_DISPLAY
the system frame buffer type (SUNGREY | SUNMONO | TEK4010).

NN_CONFIG
the root name of the neural network configuration file

NN_SEGMENT
the shell script to run image segmentation program

NN_DISPROC
the image display procedure

These environment variables are set in the file *nn_setup*. This file is "source"d each time netrun is executed.

EXAMPLE

As an example, we may train a neutral network using moment feature vectors (mom), generated from the images tr1.im tr2.im tr3.im. We first need an appropriate configuration file in the "net" directory. For example, if we name the configuration *netx* then the file *netx.mom.con* might contain the following information

```
UU.VISIX: Config file for moments (11 elements)

1       The program version(network type)
1       The number of networks (layers)
11      The number of inputs to each neuron
3       The number of neurons
1       The interconnction pattern, 1=Full
1       The neuron type: 1=threshold, 2=ramp, 3=sigmoid
```

Note that NN_CONFIG should be set to "netx"

To actually train the net from the images tr1.im tr2.im tr3.im enter

netrun verb fvec=mom tr1.im tr2.im tr3.im

The system will prompt the user for labels for each segment found within the images.

Now, to test the network, i.e. to classify an unknown set of objects in the images unk1.im unk2.im unk3.im unk4.im, enter

nnet test verb fvec=mom unk1.im unk2.im

When the classification is finished, the user may enter

netpic mom

to display labeled versions of the input imagery.

AUTHOR

Anthony P. Reeves R.J. Prokop

SEE ALSO

nn_utils(1) nnet(1) show(1)

NAME

nn_seg, nn_ims, nn_lab, nn_bit, nn_fd, nn_mom, nn_vec, nn_mts, nn_pic – Neural network modules

SYNOPSIS

nn_... <filename>

DESCRIPTION

These modules are shell files that perform a processing function on images stored in the image and preprocessing directories specified by the environment variables *NN_IMPATH* and *NN_PREPATH* They may be easily modified to accomodate different experiment designs. See netrun(1) for an example of their usage in an experiment. (netrun is a shell file which calls the above commands to do the actual work.) Most of thses modules are simple one-line shell files.

The *<filename>* argument specifies the root of a file to be processed. Pertinent file extensions are automatically generated by the selected module.

MODULES

nn_seg Generate a segmented image. The input file is *name.im* the output file is *name.seg*. This shell is provided as a simple image segmenter. A custom image segmentation program may be used by incorporating it in a shell script that has the same input and output files as nn_seg. This new shell script may be installed in the system with the NN_SEGMENT environment variable.

nn_ims Generate an image set with each subimage having one of the segments. The input file is *name.seg* the output file is *name.ims.*

nn_bit Generate a bitplane version of the segmented image. The input file is *name.seg* the output file is *name.bit.*

nn_fd Generate Fourier descriptor feature vectors, one for each segment. The input file is *name.ims* the output file is *name.fd.*

nn_mom Generate moment feature vectors for each segment of a segmented image. The input file is *name.seg* the output file is *name.mom.*

nn_mts Generate moment feature vectors from an image set. The input file is *name.seg* the output file is *name.mts.*

nn_vec Generate quick vector feature vectors; one for each segment. The input file is *name.seg* the output file is *name.vec.*

nn_lab Assign classes (or labels) interactively to an image or vector set. The input file is *name.ims* or *name.fvec* the output file is *name.ims* or *name.fvec* respectively, with class id's updated as specified. (fvec is one of mom, mts, fd, or vec)

nn_pic Generates a graphical output of classification results. A "key" image is first generated to show the object classes. Each segmented test input image is then displayed with the segments labeled by class.

AUTHOR

Anthony P. Reeves

SEE ALSO

netrun(1)

NAME

show – Display Neural network files

SYNOPSIS

show <filename> [a] [g] [<num>]

DESCRIPTION

Show is a general purpose display progr^m for showing the contents of data files for the neural network simulation package. The <filename> argument should be complete with an extension. The original input image files are all located in the neural network "image" directory. The remaining data files are all located in the neural network "preprocessing" directory.

Show examines the environment variable NN_DISPLAY to determine the actual display type and NN_IMPATH and NN_PREPATH to locate the data files.

OPTIONS

a	Display an anotated image with all segments marked.
g	For feature vector formats; plot the feature vector as a graph.
<num>	For some feature vector files num specifies a specific image or feature vector from a set.

FILE EXTENSIONS

.im	Grey level input image.
.seg	A segmented image.
.bit	A bitplane version of the segemented image.
.mom	Moment feature vector,
.vec	The fast vector feature vectors.
.fd	Fourier Descriptor feature vectors.

If no extension is specified the default is to display the grey level image (.im).

AUTHOR

Anthony P. Reeves

SEE ALSO

netrun(1)

APPENDIX B

GENERAL VISIX MODULES

NAME

nnet – neural network simulator

SYNOPSIS

nnet if=infile cf=cffile net=netfile [tf=tfile] [ol=logfile] [of=resfile] [-l] [-c] [-d] [-i] [-r] [in=num iterations] [g=gain] [wl=lower bound of initial weight] [wu=upper bound of initial weight] [m=momentum gain] [rs=random seed] [maxi=max iteration]

DESCRIPTION

Nnet simulates a neural network. It may be used both to train a network and to test a network with new inputs. The configuration of the network is specified in cffile. The contents of the neural network are stored in netfile.

OPTIONS

-l Set the network to operate in the learning mode. Input feature vectors read from *infile* are presented to the network which is trained to repond with the value of a corresponding vector in the training file *tfile*. The sequence of training vectors is repeatedly presented to the network until the correct output is obtained for all inputs.

ol=<logfile>
Statistics gathered during each iteration are stored in *logfile.*

-c The *-c* flag specifies the classification mode. A response is generated for each input vector; responses are recorded in *resfile.*

-d The *-d* flag is originally designed for debugging, but you can use it to examine the contents of the net work. This function is recommended to be used in conjuction with the *in* option. The net work contents will be displayed in the following format:

```
Layer #0 (input layer):
         input #1 input #2 input #3   .    .
neuron #1 |     |     |     |     |     |
neuron #2 |     |     |     |     |     |
neuron #3 |     |     |     |     |     |
    .  |     |     |     |     |     |
    .  |     |     |     |     |     |
    .  |     |     |     |     |     |
Layer #1 (next layer):
    .
    .
Layer #n (output layer):
```

-i This flag was originally used for debugging also. You can use it as a progress report. It shows the difference between the actual output and the desired output of the net work at every *in=* number of iterations.

-r This flag tells the program to READ in an existing net work *(net=netfile).* You must also provide the CORRECT corresponding configuration file *(cf=cffile).*

in=<int> This number is an integer specifying the number of iteration between each report from the debugging flags *-d and -i.* Default value is 100 iterations.

g=<float>
This number is the GAIN factor used in the learning mode. Default value is 0.75

wl=<float>
This number specifies the lower bound of the random initial weights. Default value is 0.5.

wu=<float>

This number specifies the upper bound of the random initial weights. Default value is 1.5.

m=<float>
: This number is the gain factor for the momentum Default value is 0.75.

rs=<int> This integer specifies a seed for the random number generator used to generate the initial weights.

maxi=<int>
: This integer specifies a maximum number of iterations during training. Default value is 100.

FORMATS

The neural network configuration is specified in a VISIX text file with the following format:

```
UU.VISIX:Neural Network Specification File

1       The program version (network type)
1       The number of networks (layers)
10      The number of inputs to each neuron
20      The number of neurons.
1       The interconnection pattern, 1=full
2       The neuron type: 1=threshold, 2=ramp 3=sigmoid
[The above four items are repeated for each network layer]
```

The neural network is stored as a real "image" matrix with each row representing the contents of a single neuron. For multiple level networks an image set format is used with one image representing each network layer.

The input file type depends upon the type of the network. Currently a real vector format (or real image where each row represents a vector) is supported. The training file, *tfile* consists of a set of byte vectors (or a byte image) where each non-zero byte represents a 1 response and zero represents a zero response. One vector (row corresponds to a single input vector (row). The results file is in the dame format as the training file. The logfile is a VISIX text file.

ADDITIONAL FEATURES

This simulator provides additional options when interrupted during training. An interrupt is signaled by <CNTRL-C>.

Here are the options:
1) Stop right now and save the current net.
2) Look at the current net.
3) Continue learning.
4) Stop and don't save anything.

This simulator will also display the content of a netfile when neither the *-c* nor the *-l* flag is specified. But be sure the cffile (configuration file) is compatible with the netfile.

AUTHOR

??, 2/89

NAME

seggen - generates a set of segment vectors from an image

SYNOPSIS

seggen [-m] [or=n] [-c] [-g] [-v] [-s] [-l] [-c4] [-r] [min=val] [if=][<filename>] [of=<ofile>] [ig=<igfile>]

DESCRIPTION

Seggen extracts all segments from an image (or an image set) and generates feature vectors for each of them. Segments are identified by being a connected set of nonzero pixels. An alternate format is that each segment consists of pixels having a single grey level value. The input image may be format 1 or 2. By default, the output is a set of images, each of which contains one segment with 255 indicating a segment pixel and 0 indicating the background. All image-segment images are the same size which is large enough to contain the largest segment.

OPTIONS

-s Image-segment images are made just large enough to contain their own segment.

-l Each image segment will have a unique grey level value (label).

-c4 The boundary will be traced using a 4-connected rule. The default is that the boundary is 8-connected.

-c The output is a chain code feature vector; the location of the start of the chain code relative to the original image is recorded in the subheader.

-v The output is a xy format feature vector; the location of the start of the vector relative to the original image is recorded in the subheader.

-m The output is a moment feature vector; the location of the origin of the moments relative to the original image is recorded in the each subheader. Moments are generated by tracing the boundary of a segment assuming that the segment has no holes.

or=<n> compute moments up to order n. The default and maximum order is 6.

-g Image segments are output but the actual grey level values of the segments are maintained.

-r Update the header to indicate the number of image segments detected. This is done after all segments have been output and may cause a delay if the output is piped.

min=value
Set a minimum size for the area of the segments. If a segment covers an area less than *value* then it is not output.

ig=<igfilename>
Image segments are output but the grey levels are extracted from a second (byte) input image specified by igfilename.

AUTHOR

BUGS

The ig=, or=, -v, -g, and -c4 options have not yet been implemented.
The -s flag makes all segments that same size as the input image.

NAME

mraw – generate raw silhouette and grey level moments from an image

SYNOPSIS

mraw if=imagefile of=momentfile or=order th=threshold [x= y= z=] [-im]

DESCRIPTION

Mraw generates raw silhouette and grey level moment vectors from a pds image. The output is a two channel vector in pds feature vector format. Vector elements are output as 4 byte real data. Channel 1 is the raw silhouette moment vector and channel 2 is the grey level moment vector.

The parameters *x*, *y*, and *z* specifiy the original objects rotation. They are written to the subheader of each channel and are not used for computation. The defaults for x, y, and z are zero. If the *-im* parameter is specified then class id. and object rotation parameters are read from the image header description section.

The parameter *thresh* specifies the grey level threshold. If thresh is positive, all pixels greater than or equal to this value are considered part of the segment. If thresh is negative, all pixels having a value less than or equal to the absolute value of thresh are considered to be part of the segment. The default is th=1 .

The parameter *order* specifies the order of the moments. A moment set of order n will have (n+1)*(n+2)/2 elements in its feature vector. If no order is specified, the default is or=6 .

It should be noted that the grey moment vector variance is written to the subheader in sub(1). This value is used by the "norm" when performing grey moment normalization.

This program can process sequencies of images that are in image set format.

AUTHOR

R. J. Prokop

SEE ALSO

mnorm(1f), pds(5), vecpds(3f)

NAME

mnorm – normalize raw moments

SYNOPSIS

mnorm [inputfiles & parameters ...]

DESCRIPTION

Mnorm is a general moment normalization program. Input data may be silhouette and/or boundary and/or grey-level (range) moment vectors in PDS feature vector format. Mnorm also accepts combined silhouette-boundary or combined silhouette-grey-level data files.

Mnorm performs several different normalizations. If input data is "raw", mnorm may be used to perform size, translation, and rotation normalization on a moment set. For grey-level data, one of three methods of size normalization may be selected. In addition, mnorm can compute the rotation augmentation and/or aspect normalization and/or Legendre normalization of the moment set.

Rotation augmentation causes new vectors to be generated when an ambiguous rotation angle is encountered. The new vectors correspond to rotations of +90 and/or -90 and/or 180 degrees. The borderline thresholds for ambiguous rotations may be specified.

Aspect normalization converts the ellipsoid of inertia of the object to a circle while keeping the area set to 1. The aspect ratio is assigned to the m20 element in the normalized feature vector.

The following is a list of parameters for specifying the input file(s) and selecting the normalization type.

sf=sfile Inputfile "sfile" is a single channel PDS feature vector file containing silhouette data.

bf=bfile Inputfile "bfile" is a single channel PDS feature vector file containing boundary data.

gf=gfile Inputfile "gfile" is a single channel PDS feature vector file containing grey-level (range) data.

sg=sgfile
Inputfile "sgfile" is a two channel PDS feature vector file with silhouette data in channel 1 and grey-level (range) data in channel 2.

sb=sbfile
Inputfile "sbfile" is a two channel PDS feature vector file with silhouette data in channel 1 and boundary data in channel 2.

-raw Performs size, translation, and rotation normalization on raw input data. The result is a set of "standard" moments.

-rot Selects rotation augmentation. This normalization is only performed on silhouette data. .

-asp Selects aspect normalization.

-leg Selects Legendre normalization.

-stat Writes rotation augmentation statistics to standard output.

-par Normally, mnorm will compute the normalization parameters used for size, translation and rotation normalization of raw grey-level data. The parameter "-par" causes mnorm to use the computed translation and rotation normalization parameters from raw silhouette data to normalize raw grey-level data.

gr=n Selects type of grey-level scale normalization. For gr=1, the standard deviation of the visible surface is used to scale the z dimension so that the normalized standard deviation is 1. The mean of the normalized moments is set to 1. For gr=2, the perceived volume is scaled using the normalization factor that sets the perceived area of the corresponding raw silhouette moments to 1. The mean of the normalized moments is set to 0. For gr=3, normalization is the same as gr=2 except that the mean of the normalized moments is set to 1.

th20=r th30=r

These parameters set the thresholds for rotation augmentation. Th20 sets the threshold on the difference between m20 and m02 before a +90 or -90 degree rotated augment vector is generated. If m20 is sufficiently close to m02, the augmentation is performed. Th30 sets the threshold on m30 before a 180 degree rotated augment vector is generated. The default values are th20=0.01 and th30=0.005 .

BUGS

There is currently no boundary normalization. Legendre normalization works only on silhouette data and cannot be called along with aspect normalization. Aspect normalization works only for silhouette data. Silhouette data must be provided if grey-level data normalization is desired.

NAME

threshold – threshold an image to a binary value

SYNOPSIS

threshold if=infile of=outfile [th=**tval**]

DESCRIPTION

Threshold compares each pixel with the threshold value *tval* and sets the ouput true (255) if the pixel is greater or equal to *tval* and false (0) otherwise. If *tval* is less than zero then the output is true if an input pixel is less than or equal to (-tval) and false otherwise. Th default value for *tval* is 128.

AUTHOR

A. P. Reeves

NAME

fdgen – generate normalized fourier descriptors

SYNOPSIS

fdgen if=infile of=outfile [ft=filter-type fv=filter-value] [-s -r -t] [-x xf=xfile -y yf=yfile -o -p -j] [-i xy=(no. XY points) el=(no. of input FDs)]

DESCRIPTION

fdgen reads a file containing X and Y coordinate vectors and computes a normalized Fourier descriptor set for coefficients from -16 to +15 (complex) for each vector. The contour may be filtered to reduce quantization errors. The filtering window is specified as a fraction of the total contour size by the parameter filter-value; e.g. 0.04 specifies a 4% window. The parameter type determines the type of filtering to be used:

1 -- rectangular
2 -- triangular
3 -- gaussian

The most frequently used values for filter and type are 0.04 and 1.

Other options are as follows:

-s no scale normalization
-r no rotation normalization
-t no translation normalization

-i inverse transform: takes Fourier descriptor input and produces X and Y vector output

xy= number of X, Y pairs in inverse
el= number of descriptors to be read from 'infile' for inverse
-x generate raw file named 'x' of X data points or real coefficients for plotting xf='xfile' same as -x but call output file 'xfile'

-y, yf='yfile' same as -x and xf for Y data points or imaginary coefficients

the following options apply for use with the x or y raw files:

-o reorder real and imag raw outputs to be -16,-15,...,-1,dc, 1,...,15
-p put input into raw outputs instead of output
-j join contour: duplicates first x,y value as last to close a contour

SEE ALSO

ccdxy(1f)

AUTHOR

Eric Rossin

BUGS

Written in FORTRAN. Input limited to 1024 data points.

NAME

xtrim – select elements from a PDS feature vector

SYNOPSIS

xtrim if=infile of=outfile lf=list [-r] [-m]

DESCRIPTION

Xtrim copies specified elements of the input vector to the output. In the case of multi-channel data, the default is to remove the same elements from each channel. If the *-m* flag is set, then xtrim merges all channels of a vector into a single channel and then copies specified elements of the merged vector to the output vector. If the *-r* flag is set, then xtrim copies all elements except those specified to the output vector. The parameter *lf* specifies the file containing a list of element numbers. This file should be an ASCII file containing integers in any order.

BUGS

Data cannot be variable length i.e. feature vector length must be specified in the main header. The user may only specify 128 elements to be saved or removed at a time.

AUTHOR

R. J. Prokop

SEE ALSO

pds(5), mraw(1f), mnorm(1f)

DISTRIBUTION LIST

Commander
Armament Research, Development and Engineering Center
U.S. Army Armament, Munitions and Chemical Command
ATTN: SMCAR-IMI-I (5)
SMCAR-FSF-RC (15)
Picatinny Arsenal, NJ 07806-5000

Commander
U.S. Army Armament, Munitions and Chemical Command
ATTN: AMSMC-GCL(D)
Picatinny Arsenal, NJ 07806-5000

Administrator
Defense Technical Information Center
ATTN: Accessions Division (12)
Cameron Station
Alexandria, VA 22304-6145

Director
U.S. Army Materiel Systems Analysis Activity
ATTN: AMXSY-MP
Aberdeen Proving Ground, MD 21005-5066

Commander
Chemical Research, Development and Engineering Center
U.S. Army Armament, Munitions and Chemical Command
ATTN: SMCCR-MSI
Aberdeen Proving Ground, MD 21010-5423

Commander
Chemical Research, Development and Engineering Center
U.S. Army Armament, Munitions and Chemical Command
ATTN: SMCCR-RSP-A
Aberdeen Proving Ground, MD 21010-5423

Director
Ballistic Research Laboratory
ATTN: AMXBR-OD-ST
Aberdeen Proving Ground, MD 21005-5066

Chief
Benet Weapons Laboratory, CCAC
Armament Research, Development and Engineering Center
U.S. Army Armament, Munitions and Chemical Command
ATTN: SMCAR-CCB-TL
Watervliet, NY 12189-5000

Commander
U.S. Army Armament, Munitions and Chemical Command
ATTN: SMCAR-ESP-L
Rock Island, IL 61299-6000

Director
U.S. Army TRADOC Systems Analysis Activity
ATTN: ATAA-SL
White Sands Missile Range, NM 88002