Optimizing Digital Hardware Perceptrons for Multi-Spectral Image Classification.

Reid Porter, Neal Harvey, Simon Perkins, James Theiler, Steven Brumby,
Jeff Bloch, Maya Gokhale, John Szymanski
Nonproliferation and International Security Division, Los Alamos National Laboratory, New Mexico.
rporter@lanl.gov

Abstract. We propose a system for solving pixel-based multi-spectral image classification problems with high throughput pipelined hardware. We introduce a new shared weight network architecture that contains both neural network and morphological network functionality. We then describe its implementation on Reconfigurable Computers. The implementation provides speed-up for our system in two ways. (1) In the optimization of our network, using Evolutionary Algorithms, for new features and data sets of interest. (2) In the application of an optimized network to large image databases, or directly at the sensor as required. We apply our system to 4 feature identification problems of practical interest, and compare its performance to two advanced software systems designed specifically for multi-spectral image classification. We achieve comparable performance in both training and testing. We estimate speed-up of two orders of magnitude compared to a Pentium III 500 MHz software implementation.

Keywords: Neural network, morphological network, shared weight network, reconfigurable computers, field programmable gate arrays, evolutionary algorithms, evolvable hardware.

1. Introduction

Multi-spectral sensors are producing increasing volumes of remotely sensed imagery. With such large quantities of data, image analysis is becoming both an expensive and difficult problem. The computational bottleneck appears in two places. The first is algorithm development. There are a large number of features that are of potential interest in remotely sensed imagery and a large variety of data sets that can be exploited. A system that can rapidly develop algorithms for different features, and different sensors is therefore very attractive. Pattern recognition systems are an ideal candidate for such problems, and potentially can provide specialized solutions using a general-purpose tool. The second computational bottleneck appears in the application of algorithms to large image databases or directly to the sensor as the data is acquired. Traditionally pattern recognition systems are developed in software, and then if possible, ported to dedicated hardware systems, to obtain real-time performance. This approach suffers from long development times. In this paper we present a Reconfigurable Computer based pattern recognition system that finds high throughput digital hardware solutions to multi-spectral image classification problems directly.

While pattern recognition aims to produce general-purpose tools, there is often problem specific knowledge that can be included to obtain better performance. For example, modern multi-spectral sensors are now being produced with high spatial resolution. Pattern recognition algorithms can therefore utilize both spectral and spatial information. One approach is to decompose the problem into feature extraction, followed by classification. With this approach, a number of predefined spatial algorithms are applied to the raw image data. Classification is then performed in the transformed feature space. In practice, feature extraction and classification are tightly coupled. A good set of features will make classification easier, but at the same time, the relevant features will depend on the type of classifier used. For this reason the feature set must usually be carefully chosen with respect to the particular problem. In an effort to produce a more *general-purpose* pattern recognition tool, other approaches such as wrapper and filter techniques [1], have been suggested. We will compare our system to two software systems that use the wrapper approach. In this case, a number of features are generated, a classifier optimized and performance measured. The feature set is then modified, often according to heuristic or stochastic techniques, and a classifier re-optimized. The process continues iteratively, until the desired performance is obtained.

Another approach to the feature extraction / classification problem is to include problem domain knowledge in the classifier architecture itself. Convolutional, or shared weight neural networks, are an example of this approach, and use architectural constraints to implement known invariants. Shared weight networks have been successively applied to several problems in speech and image processing. [2], [3], [4]. Another example of this approach was presented in [5] where a morphological shared

weight neural network was used for an automatic target recognition problem. In this case, a shared weight morphological input layer was used to implement the hit or miss transform for feature extraction. This was followed by feed-forward neural network for classification. In both these cases the choice of network components is dictated, to some degree, by the constraints of gradient based learning algorithms.

In this paper, we present a new shared weight architecture implemented using Reconfigurable Computers (RC) [6]. Reconfigurable computers are based on digital logic devices known as Field Programmable Gate Arrays (FPGAs). FPGAs contain an array of uncommitted digital logic resources that can be configured to implement application specific processing by downloading a configuration bit-stream to the device. FPGAs offer performance within a factor of 10 to Application Specific Integrated Circuits, but since they can be programmed many times, they have many of the benefits and flexibility of software. A reconfigurable computer usually incorporates a number of FPGA devices, with local memory, on a plug-in board that communicates with a host computer through a global bus.

We use a stochastic optimization technique, from the field of Evolutionary Algorithms (EA), to optimize the network parameters [7]. This means we can essentially ignore the learning algorithm and define our architecture specifically for the problem. It also allows us to constrain our solution to the hardware resources available in Reconfigurable Computers. Section 2, will describe our architecture and motivations in more detail. Section 3 then describes its implementation on a Reconfigurable Computer. This section also describes how the hardware implementation is used to accelerate the computationally intensive EA, leading to training time equal to or less than more computationally efficient gradient-based techniques. In Section 4, we will describe the Evolutionary Algorithm that we used in more detail. In Section 5 we will compare our system to two software systems that use the wrapper technique on several practical multi-spectral pattern recognition problems. Section 6 we present our conclusions and suggest directions for future work.

2. Network Design

Parallel, distributed processing, seen in network and cellular architectures, are an attractive model for computation. This model is particularly suitable for hardware design for the following reasons:

- Inherent Parallel Processing: The final output of a network is a result of partial calculations performed by each node.
- Simple Processing Elements: Each node of the network need only be capable of solving part of a particular problem and therefore are relatively simple
- Modular: Nodes are usually homogeneous across the network leading to simple large-scale designs.

For these reasons, networks appear to be a good starting point from which to develop high-throughput solutions to pattern recognition problems. This is not a new thing, and is partly why neural networks have received considerable attention for practical problems.

2.1 A Generalized Perceptron

Traditional neural networks, using linear perceptrons, involve multiplication of inputs by weights, and then summing the results. This linear operation is then followed by a non-linear activation function to produce the perceptron output. The linear perceptron, suggested by Rosenblatt [8] is defined in equation 1, where $m_i, x_i \in \Re$ and *Sign(a)* is a hard-limiting function.

$$y = Sign\left(\left[\sum_{i=1}^{N} m_i x_i\right] - \theta\right) \tag{1}$$

where

$$Sign(b) = \begin{cases} 1 & if \quad b > 0 \\ 0 & otherwise \end{cases} \tag{2}$$

Neural networks have been applied to remotely sensed satellite imagery by several authors [9], [10]. Neural networks have also been implemented on FPGAs by several researchers to accelerate both training and application of particular networks. A fundamental operation in neural networks is multiplication. This can be expensive to implement on FPGAs as the number of nodes and connectivity within the network grows. Several techniques have been used to reduce this problem: implementation of partially connected neural networks [11], and time multiplexing of network nodes using partial reconfiguration [12]. FPGA implementations can provide significant speed-up compared to software implementations, and have the advantage of flexibility, which is of benefit to many applications. However, for other applications FPGAs cannot provide sufficient densities of neurons and ASIC implementations, often analogue, are the preferred solution.

For the morphological perceptron, the operations of multiplication and addition are replaced by addition and maximum / minimum respectively. The definition of the morphological perceptron comes from work presented in [13] and is defined by equation 3:

$$y = Sign\left(\left[\bigwedge_{i=1}^{N} a_i(x_i + w_i)\right] - \theta\right) \tag{3}$$

where: $w_i, x_i \in \Re$

$a_i \in \{1, -1\}$

$\bigwedge_{i=1}^{N} b_i$ is the minimum of the set $\{b_1 \ldots b_N\}$

Morphological perceptrons have been shown to have equivalent classification power to the linear perceptron [14] and can be implemented on FPGAs much more efficiently than traditional neural networks. In [15] we found that a morphological network required approximately one quarter of the resources of a neural network implementation of comparable size. Several other researchers have suggested morphological networks but in other forms. Morphological networks presented in [16] use morphological maximum and minimums to replace the linear perceptron addition, but use multiplicative weights on the inputs. Min-max classifiers in [17] are similar in principle to [13], but they consider a restricted set of additive weight parameters.

In this paper we implement a shared weight network architecture that includes both morphological and linear perceptron functionality. This is described by equation 4.

$$y = Sign\left(\left[F_{i=1}^{N} m_i(x_i + w_i)\right] - \theta\right) \tag{4}$$

where: $m_i, w_i, x_i \in \Re$

$F_{i=1}^{N} b_i$ is a function chosen from a discrete set that operates on the set $\{b_1 \ldots b_N\}$

The generalized perceptron has both multiplicative and additive weights. In addition, there are parameters that discretely select F from a finite set. We impose several constraints on this potentially large set of functions. First, there is a maximum fan-in of 2 applied to the building blocks used to construct the Function F . This means that a function F of N variables can always be decomposed into a number of nested 2 variable functions. Second, adjustable weights are only applied at particular locations in our architecture. For example, a generalized perceptron with 4 inputs would be constructed with 2-input building blocks in a multi-layered network shown in Figure 1.

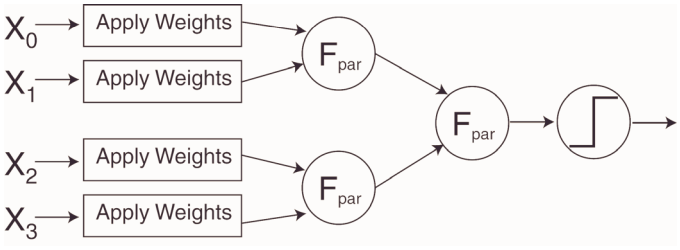

Figure 1: Constructing Larger Fan-in Perceptrons.

The function set from which F is drawn, is defined by the network of 2-input parameterized functions F_{par}. Three binary parameters *{Mux, Func and Morph}* define a set of 8 functions that are applied to the two inputs, which are summarized in Table 1.

Parameters ∈ {0,1}			F_{par} applied to inputs b_1 and b_2
Mux	*Func*	*Morph*	
0	0	0	Average $(b_1 + b_2)/2$
0	0	1	Difference $(b_1 - b_2)/2$
0	1	0	Absolute Average $\lvert (b_1 + b_2)/2 \rvert$
0	1	1	Absolute Difference $\lvert (b_1 - b_2)/2 \rvert$
1	0	0	Maximum $\vee \{b_1, b_2\}$
1	0	1	Minimum $\wedge \{b_1, b_2\}$
1	1	0	Select b_1
1	1	1	Select b_2

Table 1: Functions defined for $F_{i=1}^{N} b_i$ for N = 2

This function set is motivated primarily by the use of the generalized perceptron within a shared weight network for spatial filtering. Convolutional, or shared weight neural networks, arrange linear perceptrons in layers. The input to each perceptron is usually from a small local neighborhood. Weights are shared amongst all perceptrons in the layer. Figure 2 illustrates this approach for a 3x3 neighborhood leading to a perceptron fan in of N=9. The image array of perceptrons (0,0) through (M,M) share the same weight matrix.

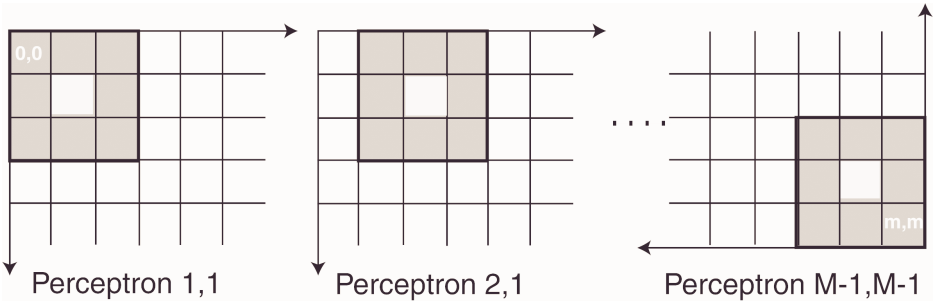

Figure 2: Neighborhoods of Shared Weight Perceptrons.

This architecture implements a linear spatial filter by convolution of the image with the shared weight matrix. One of the first implementations of convolutional neural networks was called the Neocognitron and was reported in [4]. They have been applied to a wide range of problems in image and signal processing [2]. Linear spatial filters can implement low, high and band-pass frequency-domain filters and are ideal for suppressing purely Gaussian noise. When noise is non-Gaussian, nonlinear filters seen in morphological image processing can be more effective. For example median filters are ideal for suppressing Laplacian noise [18]. As with the linear perceptron, the morphological perceptron can also be used in a shared weight network. Such a network can implement a family of non-linear spatial filters including soft morphological erosion and dilation [19].

A natural approach to deal with the combination of Gaussian and Non-Gaussian noise found in most real-world images, is to define a hybrid system that incorporates both linear are non-linear filtering components. Other examples of this approach are L-filters [20] where multiple order-statistic filters are combined with a linear combination. In another approach [21], the outputs from a bank of linear phase FIR filters are combined with a median or order statistic filter. A comprehensive account of nonlinear filters, particularly with respect to hybrid approaches can be found in [22]. By including both linear and morphological perceptron functionality, a shared weight generalized perceptron network can implement this type of hybrid architecture.

In Table 1 it can be seen that the function set also includes the absolute value. This is motivated by edge detection. In gradient weight kernels suggested by Roberts [23] and Sobel [24], two linear filter weight matrices are used to estimate the gradient in two orthogonal directions. In many practical problems, it is

the magnitude of the gradient that is of interest, not necessarily the direction. This is usually calculated with the sum of squares, or a sum of absolute values of the two orthogonal directions. A similar quantity is often seen in spatial filters suggested for texture discrimination. Laws in [25] suggested several 3x3 and 5x5 weight matrices specifically for texture. Similar to Gabor filters [26], they implement asymmetric band-pass filters. During feature extraction a bank of these filters are often applied to an image. The sum of squares, or the sum of absolute values (texture energy measures), of the filter responses has been suggested as the most useful quantities. This will be further discussed in the next section.

2.2 The Spatial Layer.

This section gives more detailed discussion of how the generalized perceptron is used within a shared weight network. In this discussion it is convenient to refer to the parts of the perceptron separately. That is, F_{par}: the three parameter functions described by Table 1 will be referred to as a *function building blocks*. Application of both additive and multiplicative coefficients according to Equation 4 will be referred to as *weighting.*

$$b_i = m_i * (x_i + w_i) \tag{5}$$

We first describe the spatial perceptron. It receives input from a 5x5 pixel neighborhood depicted in Figure 3. The node consists of a hierarchical network built from F_{par} and input-weighting units.

Figure 3: 5 x5 Neighborhood of the Spatial Perceptron.

At the top level, the 25 inputs are combined into 3 *rings*. These are grayed in Figure 3. The 5x5 ring has 16 inputs and the 3x3 ring has 8 inputs. The 3rd ring is simply the center pixel. *Weighting* is applied to the output of each ring. The center and 3x3 ring are then combined with a function building block. This output is then combined with the 5x5 ring using a second function building block. The hierarchical *summation*[1] of inputs leads to a center bias in the average. This corresponds to a weight of 8 applied to the center pixel, and a weight of 1 for pixels in both 3x3 and 5x5 rings. To achieve a semi-flat, or Gaussian average the multiplicative coefficients, which combine the 3 rings, can be used. In addition, each ring can also return a maximum, minimum or a subset of order statistics. By associating weights with these rings, a hybrid linear/non-linear spatial filter is implemented.

[1] It is often convenient to describe our architecture in terms of particular parameter specializations.

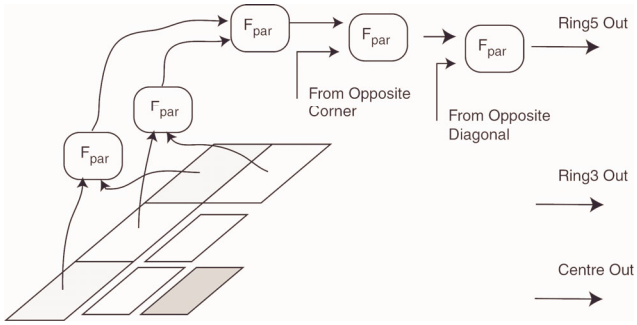

Figure 4: Order of Combination for 5x5 Ring.

Edge detection dictates the order of pixel combination within the 3x3 and 5x5 rings. The order for the 5x5 case is illustrated in Figure 4. First, pixels in each corner of the ring are combined. In the 5x5 case, a 4-input, 2 layer network of function building blocks is used. In the 3x3 ring, there are only 2 pixels associated with a corner and therefore only 1 function building block is required. In both cases, this sub-network can be configured to return the average, maximum or minimum of any subset of pixels in the corner. To estimate a gradient opposite corners are then combined with another function building block. In this case, the absolute value of the difference can be used to calculate the magnitude of the edge response. The two diagonals that result are then combined with a final function building block. This is most clearly seen in Figure 4. Texture measures based on linear spatial filters have been described as combinations of center weighted spot detectors and edge masks [27]. By combining edge responses from one ring with weighted averages of other rings, the spatial processor can effectively implement these types of band-pass texture measures.

To force rotationally invariant operators, and to reduce the number of parameters, only one quarter of the network is parameterized. The parameters for the top left quadrant of the network are used, or shared by the other three quadrants. This is a common way of enforcing rotationally invariant structuring elements when optimizing morphological filters [28]. Figure 5 illustrates the technique. Only the top-left portion of the neighborhood with gray background has parameters. The parameters are then rotated through the four quadrants. In this example, a particular set of parameters produces a filter that depends only on pixels that are crossed. In terms of morphology, the structuring element that results can be seen on the right of Figure 5.

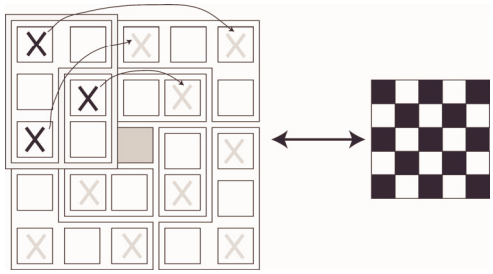

Figure 5: Enforcing Rotational Invariance.

Although rotational invariance is a desirable property, the use of symmetric spatial filters is not the only solution. In fact, the more powerful asymmetric spatial filter can be implemented at the cost of increased resources and larger parameter space. This is illustrated in Figure 6. The entire spatial network is parameterized and then multiple *parameter sharing* rotations of this network are applied to achieve rotational invariance. Similar to filter bank approaches used for texture classification, the outputs are then combined with additional function building blocks. We have implemented and experimented with this configuration, however for the remainder of this paper we will restrict our attention to the symmetrical spatial filters of Figure 5.

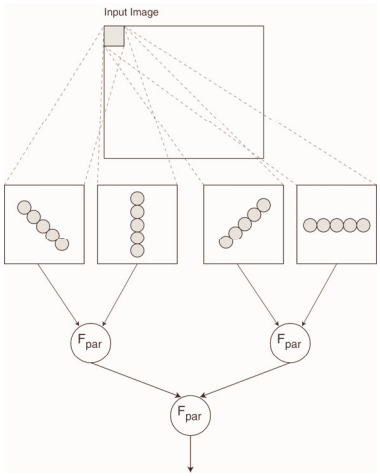

Figure 6: Implementing Rotationally Invariant Asymmetric Spatial Filters.

2.3 A Multi-Layered Network.

In this section we describe how a multi-layered network is constructed from the spatial layer described. We first introduce a second type of layer, known as the *spectral layer*. This layer is built from generalized perceptrons whose multiple inputs are taken from the same spatial location. For example, in 3-band color imagery, a perceptron in the spectral layer will receive input from the red, blue and green channels of a single co-registered location. The term Spectral Layer suggests the more general application to multi-spectral imagery.

To simplify the hardware implementation the number of bits used to represent perceptron input and output values is kept constant. This allows layers to be easily cascaded to form larger networks. Application of weights within the spectral and spatial layers means the number of bits must be increased within the perceptron to maintain full precision. To reduce the number of bits at the output we implemented a parameterized activation function for both spectral and spatial perceptrons defined by Equation 6. This replaces the hard-limiting activation function described earlier.

$$ACT_{par}(b) = \begin{cases} Max\_Range & if \quad b/par > Max\_Range \\ Min\_Range & if \quad b/par < Min\_Range \\ b/par & \text{otherwise} \end{cases} \tag{6}$$

where *par* is an positive integer parameter,
Max_Range and Min_Range are predefined integer constants.

This activation function is illustrated in Figure 7 and applies linear scaling according to an integer parameter, and saturation at pre-defined constants.

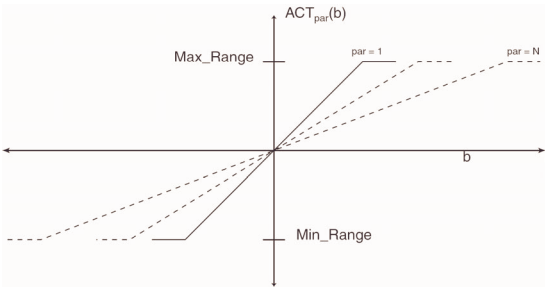

Figure 7: Parameterized Activation Function.

Multi-layered networks are constructed by using pairs of spectral and spatial layers in series or parallel. This is similar to the Neocognitron architecture in which pairs of S-cells and C-cells are used alternatively [29]. Figure 8 illustrates the structure of the particular implementation we experimented with. We implement 4 layers in parallel, and 6 layers (3 Spectral / Spatial pairs) in series. The entire network has 16 inputs. In this example the input imagery has 4 spectral channels.

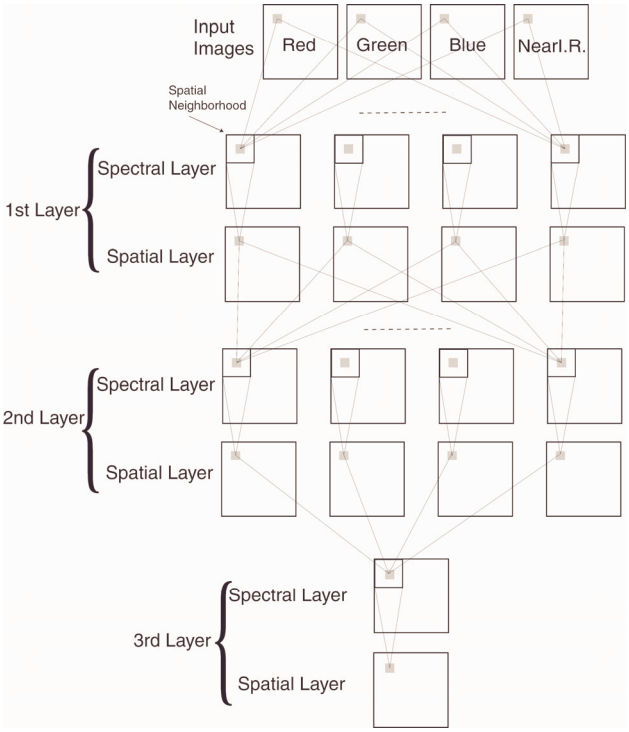

Figure 8: A 9-layer Shared Weight Network.

3. Hardware Implementation

For our implementation we used the Firebird reconfigurable computer from Annapolis Microsystems [30]. This is a 64-bit PCI card that contains a Virtex 2000E FPGA made by Xilinx Inc. [31], and a total of 40 Mbytes of on-board memory distributed in 5 independent banks.

There are two goals for our implementation:

1. To implement high-throughput solutions to the multi-spectral image classification problem.
2. To accelerate the evolutionary algorithm, or training of our network.

Shared weight networks provide the architecture necessary for the first goal since the perceptrons within a particular layer can be easily time-multiplexed. The entire network is implemented with 18 highly pipelined perceptrons (9 spectral and 9 spatial perceptrons). Input data is provided to the network, one pixel from each input channel, in raster-scan order. On-chip memory resources are used to buffer rows of pixels so that, after an associated latency, a spatial layer perceptron has access to the entire neighborhood each clock cycle. Input to the network is assumed to be 8-bit, 2's complement integers in the range –127 to 127. The network maintains this data path width and produces an 8-bit 2's complement output. Max_Range and Min_Range from equation 6 are therefore 127 and –127 respectively. The sign of the pixel output dictates what class a pixel is assigned to for the 2-class classification problem. Note, this effectively fixes the threshold parameter θ from Equation 4 at zero.

The second implementation goal leads to additional on-chip infrastructure, and a tight coupling between the RC and the host workstation. Evolutionary algorithms can be considered a 'sample and test' design paradigm [32]. This involves choosing a set of parameters, applying the function to the training data, and finally assessing the performance. When this approach is applied to image processing problems, application can involve several data intensive operations. By implementing these operations in RC, the evolutionary algorithm can be greatly accelerated. Figure 9 illustrates the additional on-chip infrastructure that is required.

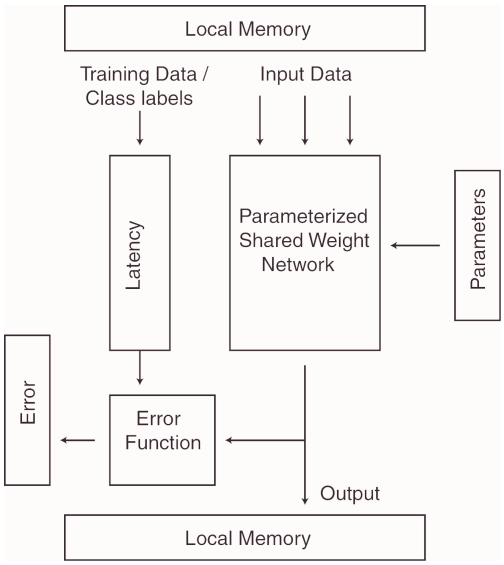

Figure9: Pipelined 'Sample and Test' Architecture.

The host processor writes parameters to on-chip registers, which dictates a candidate solution in the parameterized shared weight network. Input images are then passed through the network, producing an output image. At the same time, the target classification (class labels) are passed to a delay unit. This unit implements latency equivalent to the shared weight network. The latency adjusted training data is then compared to the network output in the Error Function unit. An error is calculated and stored in on-chip registers where it can be read by the host.

As mentioned earlier, the most computationally intensive component for our problem is the application of the parameterized network to the input data, and therefore the error function could be calculated elsewhere. We prefer to calculate this value on-chip due to the communication limitations between host processor and reconfigurable computer imposed by the PCI global bus. To efficiently evaluate a large number of candidate solutions this communication must be minimized. Figure 10 illustrates the communication between host workstation and RC during training.

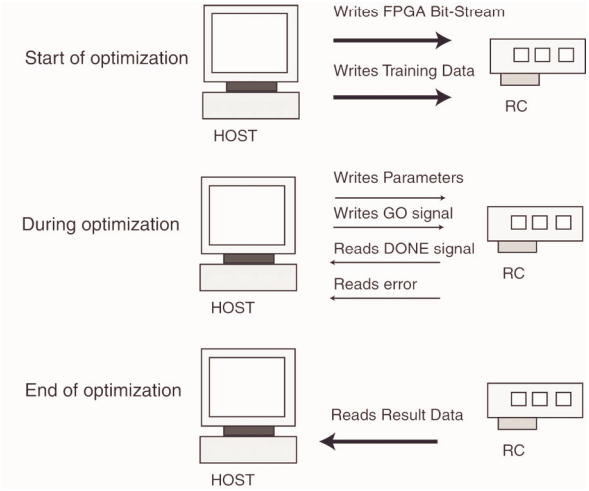

Figure 10: Communication Between Host and Reconfigurable Computer (RC).

In Figure 10, large volume input data and class labels are loaded once at the start of optimization to the RC local memory. Communication between host and the RC during optimization involves downloading a particular set of parameters, initiating the network evaluation, and then retrieving the output error.

Only at the end of optimization, is the result image from the lowest error network retrieved for inspection.

In our implementation we used a binary error metric based on a weighted hamming distance between the network output and the training data class labels. It is defined in Equation 7.

$$Error = \left(T_F \middle/ T_T \right) * 0.5 + \left(F_F \middle/ F_T \right) * 0.5 \quad (7)$$

where:

T_F is the number of Class 1 pixels misclassified by the network and T_T is the total number of Class 1 pixels in the training set. Similarly, F_F is the total number of class 2 pixels misclassified and F_T is the total number of class 2 pixels in the training set.

Positive output pixels are assigned to one class, and negative output pixels the second class. Since in binary classification problems, finding all Class 2 pixels is equivalent to finding the Class 1 pixels, two errors need to be calculated: one error for class 1 pixels being positive and a second error for class 1 pixels being negative. The host program retrieves both these errors from the RC and chooses the best one.

Note, this error metric is suitable only for binary or two-class classification problems. Secondly, only classification error is considered. No measure is made of the certainty in decision such as a distance from the decision boundary. The benefit of the weighted hamming metric is the simplicity of on-chip implementation.

3.1 Resource Usage

The 18-layer network was implemented at 50MHz. The resource estimates from both Synthesis and Place and Route software are summarized in Table 2. It can be seen that post synthesis the usage was estimated at 45%, while after place and route it grows to 64%. This indicates there is significant room to optimize the design. All components of the network and fitness evaluator architectures were designed with structural VHDL to which placement constraints can be applied. This effectively allows the design to be manually placed, which would bring the 64% usage closer to 45%. Manually placing the design would also allow higher clock rates to be achieved.

Resource	**Number**	**Percent of chip**
Post Synthesis		
Number of SLICES	8706 out of 19200	45%
Post Place and Route		
Number of SLICES	12427 out of 19200	64%
Number of BLOCKRAMs	56 out of 160	35%
Number of Tri-state buffers	2256 out of 19520	11%

Table 2: Network Resource Usage.

Note, SLICES are an abstract unit of digital logic resources (Look up tables and registers) for Virtex FPGA devices. BLOCKRAMs refer to dedicated memory elements that are also available on Virtex FPGAs. These were used to implement the spatial layer row-buffering and other latencies required to pipeline the network.

3.2 Evaluation of Speed-up

Evaluating the speed-up of the RC implementation compared to software implementations is a difficult problem since the quality of the pattern recognition algorithm is also of interest. In this comparison we ignore algorithm quality, and compare the hardware execution speed to a high-level software approximation of the network components. For the software, execution time was estimated by implementing a number of optimized image processing operators. For each Spectral Layer in the network, a linear combination was used. For each Spatial Layer, a 5x5 neighborhood average was calculated. The software experiment therefore performed a total of 9 linear combinations of 4 images and 9 neighborhood averages. This software implementation is therefore simpler (and less powerful)

than the RC implementation. The hardware execution time was the average from 1000 network evaluations. The execution times and relative speed-up are summarized in Table 3. It can be seen that the RC implementation, based on a Virtex FPGA device (introduced late 1998) obtains a speed-up of two orders of magnitude compared to the software implementation running on a 500 MHz Pentium III (introduced February 1999).

Image Size (pixels)	Software Evaluation Time (Seconds)	RC Evaluation Time (Seconds)	Speedup
65536	0.18	0.001	112
131072	0.36	0.003	124
262144	0.71	0.006	129
524288	1.39	0.01	122
1048576	2.75	0.02	136

Table 3: Evaluation Times for Software and RC Implementations.

4. Optimization with Evolutionary Algorithms

In this section we describe the evolutionary algorithm that is used to optimize the parameterized shared weight network. EA have been applied extensively to neural network design and optimization in a number of different ways. We use EA to optimize parameters of a fixed topology network. This is similar to optimization of neural network weights in [33] and [34]. The flexibility of EA means the topology can also be optimized for a particular problem [35]. Developmental encoding has also been suggested, which optimizes a program whose instructions dictate placement and connectivity of network nodes [36]. A good review of evolutionary neural networks can be found in [7].

When optimizing network architectures, competition (the main evolutionary pressure in traditional EA) is not the only factor. Since sub-components in a network are dependent, and only contribute partially to a complete solution, collaboration is also required. The idea is to decompose, or modularize the optimization problem and find a collection of sub-components that work well together. We first define the subcomponent of our network, known as a node, as a spectral and spatial perceptron pair. The solution space for the entire network is therefore defined by the parameters of 9 nodes. We will discuss the node representation in Section 4.1. We then describe the genetic operators, which are used to produce new candidate solutions, in Section 4.2. In section 4.3 we will describe the evolutionary neural network technique that we used to optimize the complete 9-node network.

4.1 Representation

The parameter space for a node is defined by a combination of integers and binary bits. The hardware implementation dictated particular representations for particular parameters. However, this can be significantly different from the representation used to optimize the network. The representation of parameters in both hardware and software are summarized in Table 4. The additive coefficients are stored in the hardware registers using two's complement representation. For the multiplicative coefficients, a sign bit is used (MSB). The activation function is represented by an unsigned integer. There is also an additional unsigned integer parameter associated with each input to the network. This is used to select a particular channel of the multi-spectral input image. This parameter has a range from 1 to the number of bands. The Firebird local memory dictated an upper limit of 12 input channels. The mutation strategy that is applied to parameters to produce new candidate solutions are also summarized in Table 4.

Parameter representation in Software	Parameter representation in Hardware	Mutation Strategy
Node Parameters (Spectral and Spatial Perceptrons)		
BIT Func BIT Morph BIT Mux	Binary bit Binary bit Binary bit	Bit flip Bit flip Bit flip
INT Sum_Coef	Two's Complements form of: Sign(Sum_Coef)*$2^{\|Sum\_Coef\|}$	±1 in range {–6 to 6}
INT Mult_Coef	Sign Bit Representation of Mult_Coef	±1 in range {–7 to 7}
UNSIGNED Scaling (Activation Function)	Tri-state control lines.	±1 in range {0 to 4}
Channel Chooser: 4 Per Node (Input layer nodes only)		
UNSIGNED Band	Tri-state control lines	±1 in range {1 to Number of Bands in Training Data}

Table 4: Software Chromosome for Spectral / Spatial Network Node.

4.2 Genetic Operators

Mutation can be applied to a node in a variety of ways, most easily visualized as a mutation tree. For each mutation there is a probability of a particular branch being taken. This is illustrated in Figure 11. This figure illustrates a 4 input spectral perceptron, followed by the parameterized activation function. This is followed by the spatial perceptron and finally the second activation function. Once the end of the mutation tree is reached mutation points are chosen with equal probability. The choice of probabilities is fairly arbitrary and is based on familiarity with the representation and experimentation.

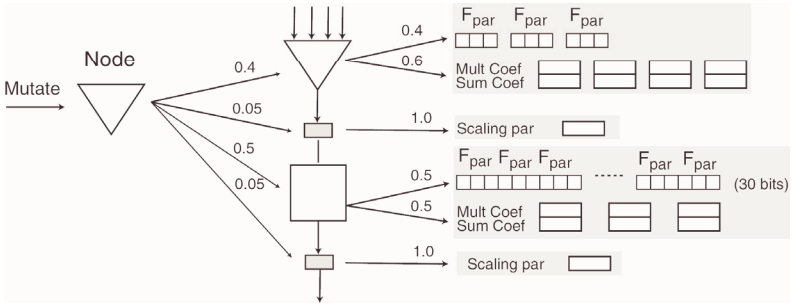

Figure 11: Hierarchical Implementation of Mutation.

For nodes at the input-layer, parameters are also included to select the appropriate channel from the input data. There is a 20% chance that an input layer node will randomly mutate one of these 4 input parameters. The remaining 80% of the time, mutation is applied according to Figure 11.

Crossover within the node is applied in a similar way to mutation and is illustrated in Figure 12. In this case there is a 50% chance that the crossover is applied to the spectral component and 50% chance the spatial component. Within these components, crossover points (illustrated by dashed lines) are chosen with equal probability.

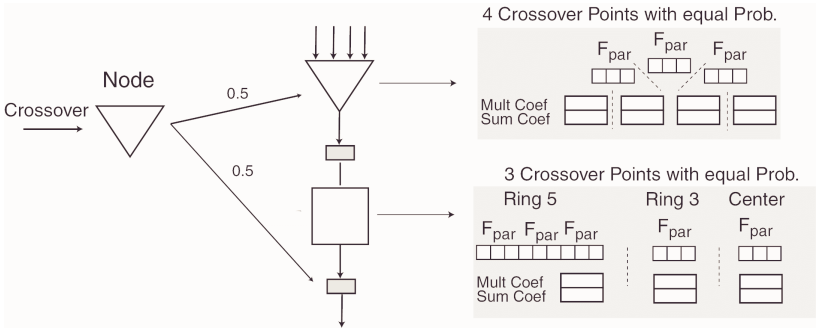

Figure 12: Hierarchical Implementation of Crossover.

4.3 Evolutionary Algorithm

In our implementation we implement 9 error metric units, one for each spectral/spatial layer. This allows us to efficiently implement an *Incremental Learning* approach. In the incremental optimization of neural networks described in [37], and [38], the optimization begins with one network node. Once this node has reached a specified level of error, or there is no improvement in error after a specified number of generations, another node is introduced. In some cases, the first node is fixed and the EA is only applied to the second node. In other cases, the parameter space is extended to include both nodes are then evolution continues as normal. This process continues until the network has reached a desired error or maximum number of nodes.

In our implementation, the four 1st layer nodes are optimized in parallel in four different populations. Since an error is calculated on the output from each node, these populations can be optimized independently. Within each population, we used a simple generational Genetic Algorithm with elitism [39]. In the second stage, the best 1st layer nodes in each population are configured and remain fixed. The 4 nodes in the 2nd layer are then optimized independently in 4 populations. In the third stage, the best 2nd layer nodes are also configured. Both 1st and 2nd layer nodes remain fixed and only the output node is optimized.

To maximally utilize the fitness evaluator resources, all 9 nodes should be involved in evolution at all times. This is not possible with the *Incremental Learning* approach, and some nodes remain fixed while others are evolved. It is possible to evolve higher layer nodes while lower-level nodes are evolved. This means the 1st, 2nd and 3rd layers are evolved in Stage 1. Only the 2nd and 3rd are evolved in Stage 2 and just the 3rd layer in Stage 3. This is illustrated in Figure 13 for clarity. The arrows in this figure indicate that optimization of the node configuration is based on the nodes output. This can produce unpredictable fluctuations in the higher-layer scores, since the data they are supplied with can vary from one evaluation to the next. At the start of the network evolution there is another affect. That is, reward is given to nodes that simply pass the data on. They are rewarded for the high scores from the lower layers and therefore not the processing they perform.

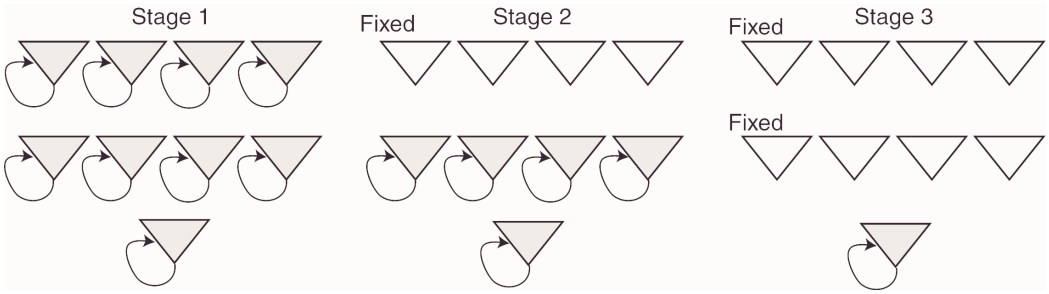

Figure 13: 3 Stage Incremental EA

After *Incremental Evolution*, a variable number of *Optimization Cycles* are applied. This is motivated by the fact that nodes that may not score well individually can be very useful within the network and in fact may lead to better scores in the final output. This is the main reason why competition is not the only factor in network optimization, and co-operative behavior is desired. Various mechanisms have been

suggested for implementing this with varying levels of complexity [40]. The Optimization Cycle approach that we used is most similar to greedy strategy suggested in [37].

In the Optimization Cycle, nodes receive reward based on the final network output. There are 9 stages to the optimization cycle, 1 for each node in the network. The network is configured with the best nodes from each population that were found in the incremental development phase. Each node in the network is then evolved in turn using the fitness calculated on the final network output. This is illustrated for clarity in Figure 14.

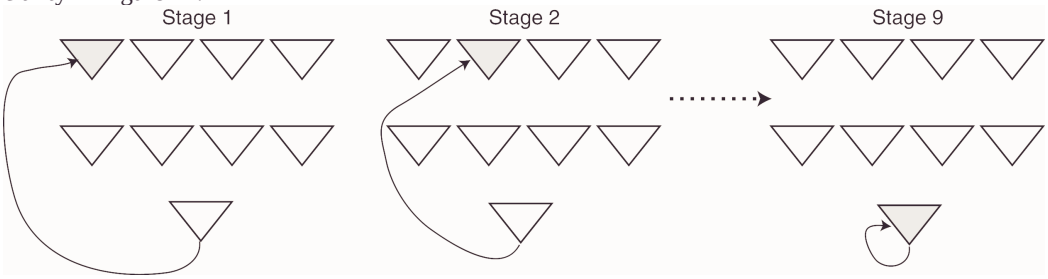

Figure 14: The 9 Stage Optimization Cycle

5. Performance Comparison

In this section we compare the accuracy of our system to two software pattern recognition systems designed for multi-spectral image classification. The first software system is GENIE [41], which implements the wrapper feature selection method. In this system, features are constructed by combining a number of image processing primitives in a graph. A Fisher linear discriminant is then applied to the graph outputs to produce a classification. A genetic algorithm is used to modify graphs from one iteration to the next. The image processing primitives include a rich set of linear and non-linear spatial filters, as well as several more complex spectral processing algorithms. A more detailed description of the architecture can be found in [42]. The second system, known as AFREET, is also a wrapper technique. It produces features by building a number of independent trees from image processing primitives. It then applies a Support Vector Machine (SVM), typically without kernels, in place of the fisher discriminant. The feature set is updated according to a greedy heuristic strategy between iterations [43]. SVM have gained considerable interest for pattern recognition problems since they implement explicit measures to reduce well-defined bounds on generalization error [44]. The image processing primitives contain a rich variety of both spatial and spectral algorithms similar to the GENIE system.

In previous experiments we compared our system on 4 different features of interest, over three different scenes [45]. The scenes are generated from the MODIS airborne simulator data set [46]. Preprocessing was applied to produce a 10-channel image that simulates the output from the first 10 channels of the Multi-spectral Thermal Imaging sensor reported in [47]. The features were chosen to span a range of difficulties. The first feature of interest is water. This is the easiest problem of the four since water has a unique spectral signature. The second problem is to identify the golf courses. It is believed that this problem is of moderate difficulty but should have distinguishable spectral properties. The type of grass used in golf courses is often unique and therefore may be detected with purely spectral information. The third feature of interest is not as well defined, and is simply urban or 'built-up' areas. Urban areas can include a wide variety of materials and therefore spectral signatures. Spatial information is therefore believed to play an important role in identifying this feature. The fourth feature of interest is roads. This problem is significantly different from the previous three broad area features. We include it to investigate the versatility of our system, however we do not expect that our pixel based classification approach would be competitive with dedicated road finding algorithms.

In previous experiments, we used a leave two-out scheme for training/testing. That is, one scene was used for training and algorithms were then tested on the remaining two scenes. This was repeated for all 3 permutations. While the ultimate goal of our systems is to produce algorithms that can be applied across multiple scenes and periods of time, we found that it was difficult to solve this problem using only one scene for training. One solution to this problem lies in remote sensing and application of advanced preprocessing to reduce the variation encountered in environment and sensor over long periods of time. In this experiment, we attempt to overcome this problem by providing the pattern recognition systems with a more accurate representation of the problem. We introduce a fourth scene for

each feature. The 4 scenes are then divided into two images by tiling non-overlapping portions from each scene. This means that half of each of the 4 scenes is represented in both training and testing data. Examples of the tiled data sets, and associated class labels are illustrated in Figure 15 for the golf course, and road finding problems. For the class labels, white indicates the feature of interest, gray indicates non-feature and black corresponds to *don't know* and does not contribute to the error. The class labels was generated with a graphical *point and paint* program and can appear arbitrary. Regions that are ambiguous are left as *don't know* to avoid providing inconsistent training data.

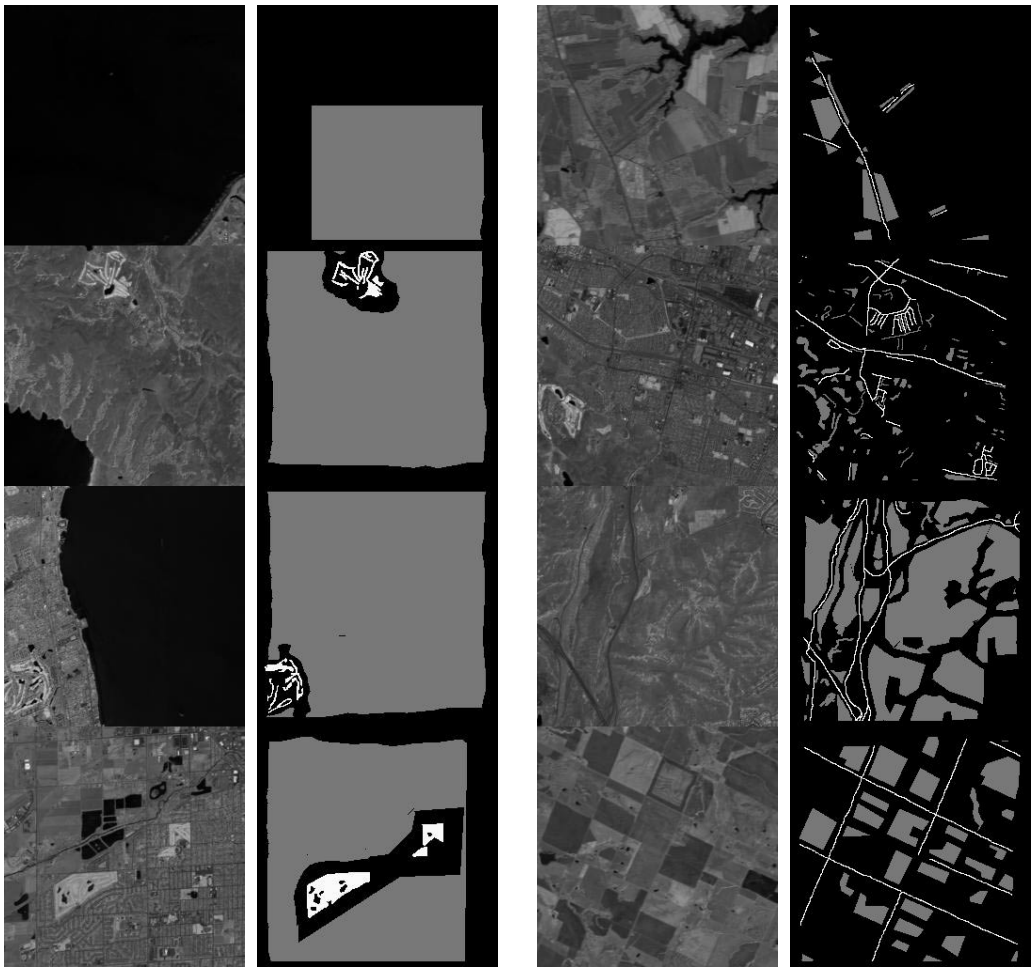

Figure 15: Tiled Images used for Training for the Golf Course and Road Finding Features.

5.1 Results

Convergence of evolutionary algorithms is difficult to define. For this reason we apply our system to all problems at 4 different levels of effort, which are detailed in Table 5. For the GENIE software system, a population of 100 candidate graphs is evolved over 100 iterations for each problem. The average training time for the GENIE system was 19 hours. The AFREET system was applied for 200 iterations (feature selection/SVM optimization). To accelerate the SVM optimization, AFREET sub-samples the training data. For this reason, execution time does not depend on the training image size, as in the other systems, but rather the problem difficulty. Execution times varied from 9 minutes through to 50 minutes and averaged 19 minutes for all the problems.

Effort Level	Low	Medium	High	Extreme
Execution Time	34.7 seconds	3.2 minutes	15 minutes	56 minutes
Population	50	100	200	200
Generations	120	240	240	480
Optimization Cycles	1	2	6	12

Table 5: Levels of Effort used in Training.

All systems were trained on each of the two tiled images, for each problem, in turn. The result is then applied to the second image, for each problem, to obtain an out-of-sample test scores. We define a score out of 1000 based on the error metric previously defined in Equation 7. This is described in Equation 8.

$$Score = (1 - Error) * 1000 \tag{8}$$

The scores achieved in training and testing are summarized in Tables 6 and 7 respectively.

Training Image	Shared Weight Network				GENIE (Avg. 19 hrs)	AFREET Time (minutes)	
	Low	Medium	High	Extreme			
Cloud 1	997.6	999.4	1000	1000	999.9	995.7	10.3
Cloud 2	994	994.9	999.8	999.5	998.6	998.7	9
Golf 1	992.3	995	998.8	997.2	997.3	999.7	9.3
Golf 2	996.1	998.7	999	999.7	998.7	997.9	13.8
Urban 1	881.6	974.3	983.9	987	992.1	984.7	25.3
Urban 2	947	982.6	991.3	993.2	996.5	992	12.3
Road 1	818.3	878.8	892.6	898.2	911	903.6	50
Road 2	904.1	930	917.6	925.4	944.6	935.9	22.7

Table 6: Fitness Scores Obtained on Training Data.

Training Image	Shared Weight Network				GENIE	AFREET
	Low	Medium	High	Extreme		
Cloud 1	989	968	981.4	991	978.5	819.6
Cloud 2	995	995	997.2	997.3	999.9	971.3
Golf 1	836.5	962	987	971.4	823.5	966.7
Golf 2	977.6	984	986	987	968.6	998.7
Urban 1	852	957.5	948	947.4	973.2	980
Urban 2	817.2	858.3	922.5	946	936.2	943.5
Road 1	837.4	897.3	883.8	909.3	935.5	913.7
Road 2	800.8	855.9	847.9	816.6	869.7	838.2

Table 7: Fitness Scores Obtained on Testing Data.

Table 8 shows the training and testing scores that were achieved when our system is applied with 8 different random seeds to the Road 1 problem. This experiment was conducted to investigate the variability of our system at the different levels of effort.

Run	1	2	3	4	5	6	7	8	Average	S.D.
Low										
Training	785.6	835.2	837	813.5	793.4	802.6	845.8	774.7	811	26.2
Testing	798.3	866.4	827.2	827.2	816.5	796	893	790.5	826.9	36.1
Medium										
Training	878.5	904.8	901.7	859.4	912.7	880	831.8	872.3	880.2	26.6
Testing	896.7	858	894.2	890	910.9	903.7	876.3	890.3	890	16.5
High										
Training	902	908.9	873.6	909.6	883.3	902.3	901.6	884.2	895.7	13.4
Testing	901.6	911.1	899.4	928.5	906.9	895.2	916.6	865	903	18.7
Extreme										
Training	894.1	922	899	897.9	871.9	925.4	902	903.7	902	16.7
Testing	871.5	917.1	914.4	909.6	896	913.2	911.1	916.3	906.2	15.5

Table 8: Fitness Scores Over 8 runs for the Road Problem (tested on the second road tile).

5.2 Discussion

With low effort, our system performed poorly on both training and test images. This illustrates the difficulty of the problems, and therefore does not measure the true capacity of the system. With increased effort, the approach shows potential as a practical pattern recognition system. For the more difficult problems, our approach is usually outperformed on the training data by the GENIE and AFREET systems. This indicates that our system may lack classification power for difficult training data, compared to the software systems. The network has a fixed number of resources with which it can work, and therefore this is not surprising. In contrast, both GENIE and AFREET systems are able to form extremely complex algorithms to fit training data. Test data results indicate that our system does not appear to suffer from over fitting. It is hypothesized that the limited resources of our system may be responsible for its good performance on test data.

The results of Table 8 indicate there is variation in performance from one run to the next, particularly at low levels of effort. This is expected from an evolutionary algorithm system. It can be seen for high-levels of effort the variation is reduced, which indicates that a more efficient EA search strategy would help with this problem. In addition, it is noted that for two of the runs at extreme level of effort, our system actually obtained higher training data scores than both the GENIE and AFREET systems. This indicates that the system may in fact have sufficient resources for practical problems, however the difficulty in obtaining these scores is a problem. A different EA strategy may help with this problem. Another solution would be to increase the classification power of the system, hence providing a richer solution space that could potentially be searched more easily.

To improve our system, it is concluded that the classification power should be increased. The fact that the system has a fixed set of resources for problems of varying levels of difficulty may also be a limitation, and a more flexible use of the resources may be desirable. At the same time, it is known that increasing classifier complexity can lead to problems of over-fitting. Therefore, it is likely that including explicit measures to avoid over fitting, such as a non-binary error function, on-chip cross validation or boosting [48], would be beneficial to a more complex system.

6. Conclusions and Future Directions

Evolutionary Algorithms and Reconfigurable Computers have been used independently for a number of years. More recently, these fields have been combined in the field of Evolvable Hardware. This means mutual benefit: the long computation times of Evolutionary Algorithms is avoided, and digital hardware building blocks can be more easily optimized. This new design environment seems ideal for producing high throughput hardware solutions to optimization problems. In this paper we demonstrated this approach for solving large data volume pattern recognition problems.

We exploited the flexibility of 'sample and test' optimization to define our parameter space in terms of feature extraction algorithms used in image processing. This resulted in a *hybrid* feature extraction/classification architectures that is scalable, inherently parallel and easily implemented. Analysis of both processing speed and pattern recognition quality suggest our approach can produce viable high-performance solutions to the multi-spectral classification problem.

Our current implementation uses a static feed-forward architecture. In the field of image and signal processing, this is considered a Finite Impulse Response (FIR) filter architecture. We intend to extend our work to include feed-back or state to network layers. This will lead to an Infinite Impulse Response (IIR) filter architecture. We are particularly interested in these architectures since they are naturally suited to real-time video image processing, where temporal information can be exploited. In practice, IIR filters are usually much more difficult to design than FIR filters. The combination of Reconfigurable Computing and 'sample and test' optimization provides us with a unique framework for designing these filters.

7. References

1. Blum, A. and P. Langley, *Selection of Relevant Features and Examples in Machine Learning*. Artificial Intelligence, **97**(1-2): p. 245-271, 1997.
2. LeCun, Y. and B. Boser, *Convolutional networks for images, speech and time series*, in *The Handbook of Brain Science and Neural Networks*, M. Arbib, Editor. 1995, MIT Press: Cambridge, MA. p. 255-258.
3. Neubauer, C., *Evaluation of Convolutional Neural Networks for Visual Recognition.* IEEE Transactions on Neural Networks, **9**(4): p. 685-696, 1998.
4. Fukushima, K., *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*. Biol. Cybern, **36**: p. 193-202, 1980.
5. Won, Y. and P.D. Gader. *Morphological Shared-Weight Neural Network for Pattern Classification and Automatic Target Detection*. in *IEEE International Conferenec on Neural Networks*. 1995.
6. Villasenor, J. and W. Mangione-Smith, *Configurable Computing*. Scientific American,:, 1997.
7. Yao, X., *A review of evolutionary artificial neural networks.* International Journal of Intelligent Systems, **8**(4): p. 539-577, 1993.
8. Rosenblatt, F., *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. 1962, Washington D.C.: Spartan.
9. Figueiredo, M.A. and C. Gloster. *Implementation of a Probabilistic Neural Network for Multi-spectral Image Classification on an FPGA Based Custom Computing Machine*. in *Vth Brazilian Symposium on Neural Networks*. Belo Horizonte, Brazil: IEEE Computer Society. 1998.
10. Paola, J.D. and R.A. Schowengerdt, *A review and analysis of backpropagation neural networks for classification of remotely-sensed multi-spectral imagery.* International Journal of Remote Sensing, **16**(16): p. 3033-3058, 1995.
11. Chung, Y.Y., M.T. Wong, N.W. Bergmann, and M. Bennamoun. *Implementing Neural Network in Custom Computers*. in *IEEE International Conference on Systems, Man and Cybernetics : Conference Theme : Intelligent Systems for Humans in a Cyberworld*. San Diego, California, USA: IEEE. 1998.
12. Eldredge, J.G. and B.L.Hutchings, *Run-Time Reconfiguration: a method for enhancing the functional density of SRAM-based FPGAs.* Journal of VLSI Signal Processing, **12**(1): p. 67-86, 1996.
13. Ritter, G.X. and P. Sussner. *An introduction to morphological neural networks*. in *13th International Conference on Pattern Recognition*. Vienna, Austria. 1996.
14. Sussner, P. *Morphological Perceptron Learning*. in *Joint Conference on the Science and Technology of Intelligent Systems*. Maryland: IEEE. 1998.
15. R. Porter, M. Gokhale, N. Harvey, S. Perkins, and C. Young. *Evolving network architectures with custom computers for multi-spectral feature identification*. in *Third NASA/DoD Workshop on Evolvable Hardware, EH-2001*. Long Beach, CA. 2001.
16. Wilson, S.S. *Morphological Networks*. in *Visual Communications and Image Processing IV*: SPIE. 1989.
17. Yang, P. and P. Maragos, *Min-Max Classifiers: Learnability, Design and Application*. Pattern Recognition, **28**(6): p. 879-899, 1995.

18. Gabbouj, M., E.J. Coyle, and N.C. Gallagher, *An overview of median and stack filtering*. Circuits, Systems, and Signal Processing, **11**(1): p. 7-45, 1992.
19. Pu, C.C. and F.Y. Shih. *Soft Mathematical Morphology: Binary and Gray Scale*. in *International Workshop on Mathematical Morphology and its Applications to Signal Processing*. Barcelona, Spain. 1993.
20. Bovik, A.C., T. Huang, and D. Munson, *A generalization of median filtering using linear combinations or order statistics.* IEEE Trans. Acoust., Speech, Signal Processing, **31**: p. 1342-1350, 1983.
21. Heinonen, P. and Y. Neuvo, *FIR-median hybrid filters.* IEEE Trans. Acoust., Speech, Signal Processing, **35**: p. 832-838, 1987.
22. Astola, J. and P. Kuosmanen, *Fundamentals of Nonlinear Digital Filtering*. 1997, New York: CRC Press.
23. Roberts, L.G., *Machine Perception of Three-Dimensional Solids*, in *Optical and Electro-Optical Information Processing*, J.T. Tippet, Editor. 1965, MIT Press: Cambridge, Mass.
24. Sobel, I., *Camera Models and Machine Perception*. 1970, Stanford Artificial Intelligence Lab: Palo Alto.
25. Laws, K.I. *Texture energy measures*. in *Proceedings of Image Understanding Workshop*. 1979.
26. Jain, A.K. and F. Farrokhnia, *Unsupervised Texture Segmentation using Gabor Filters.* Pattern Recognition, **24**(12): p. 1167-1186, 1991.
27. Pietikainen, M., A. Rosenfeld, and L.S. Davis, *Experiments with Texture Classification Using Averages of Local Pattern Matches.* IEEE Transactions on Systems, Man and Cybernetics, **SMC-13**(3), 1983.
28. Kraft, P., N.R. Harvey, and S. Marshall, *Parallel genetic algorithms in the optimization of morphological filters: a general design tool.* Journal of Electronic Imaging, **6**(4): p. 504-516, 1997.
29. Fukushima, K., *Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition.* Neural Networks, **1**: p. 119-130, 1988.
30. Annapolis, M., *http://www.annapmicro.com/*. 2001.
31. Xilinx, *Virtex Configuration Architecure: Advanced User's Guide*. 1999, Xilinx Inc.
32. Miller, J.F., D. Job, and V.K. Vassilev, *Principles in the Evolutionary Design of Digital Circuits - Part I.* Genetic Programming and Evolvable Machines, **1**(1): p. 7-35, 2000.
33. Whitley, D., S. Dominic, R. Das, and C. Anderson, *Genetic Reinforcement Learning for Neurocontrol problems.* Machine Learning, **13**: p. 259-284, 1993.
34. Floreano, D. and F. Mondada, *Automatic creation of an autonomous agent: Genetic evolution of a neural network driven robot.*, in *From Animals to Animats 3: Proc. 3rd Int. Conf. Simulation of Adaptive Behaviour*, D.C. J.A. Meyer, P. Husbands, S. Wilson, Editor. 1994, MIT Press: Cambridge.
35. Harvey, I., P. Husbands, and D. Cliff, *Seeing the light: Artificial evolution, real vision*, in *From Animals to Animats 3: Proc. 3rd Int. Conf. Simulation of Adaptive Behaviour*, J.A.M. D. Cliff, S. Wilson, Editor. 1994, MIT Press.
36. Gurau, F., *Automatic definition of sub-neural networks*. 1994, Labortatoire de l'Informatique du Parallelism, Ecole Normale Superieure de Lyon: Lyon, France.
37. Potter, M.A. and K.A.D. Jong. *Evolving Neural Networks with Collaborative Species*. in *Proceedings of the 1995 Summer Computer Simulation Conference*. Ontario, Canada. 1995.
38. Meeden, L., *An Incremental approach to developing intelligent neural network controllers for robots.* IEEE Transactions on Systems, Man and Cybernetics: Part B, **26**(3): p. 474-485, 1996.
39. Mitchell, M., J.P. Crutchfield, and R. Das. *Evolving Cellular Automata with Genetic Algorithms: A Review of Recent Work*. in *First International Conference on Evolutionary Computation and Its Applications (EvCA'96)*. Moscow, Russia. 1996.
40. Moriarty, D.E. and R. Miikkulaiinen, *Forming Neural Networks through Efficient and Adaptive Coevolution.* Evolutionary Computation, **5**(4), 1998.
41. Theiler, J., N.R. Harvey, S.P. Brumby, J.J. Szymanski, S. Alferink, S. Perkins, R. Porter, and J.J. Block. *Evolving Retrieval Algorithms with a Genetic Programming Scheme.* in *Proc. SPIE*. 1999.
42. Perkins, S., J. Theiler, S.P. Brumby, N.R. Harvey, and R.B. Porter. *GENIE: A Hybrid Genetic Algorithm for Feature Classification in Multispectral Images.* in *Proc. SPIE*. 2000.
43. S. Perkins, N.R. Harvey, S.P. Brumby, and K. Lacker. *Support Vector Machines for Broad Area Feature Extraction in Remotely Sensed Images.* in *Proc. SPIE 4381*. 2001.

44. Vapnik, V.N., *Statistical Learning Theory*. Wiley Series on Adaptive and Learning Systems for Signal Processing, Communications, and Control, ed. S. Haykin. 1998, New York: John Wiley & Sons, Inc.
45. R. B. Porter, M. Gokhale, N. R. Harvey, S. J. Perkins, and C. Young. *Evolving a spatio-spectral network on reconfigurable computing for multispectral feature identification*. in *Proc. SPIE*. 2001.
46. King, M.D., W.P. Menzel, P.S. Grant, J.S. Myers, G.T. Arnold, S.E. Platnick, L.E. Gumley, S.C. Tsay, C.C. Moeller, M. Fitzgerald, K.S. Brown, and F.G. Osterwisch, *Airborne scanning spectrometer for remote sensing of cloud, aerosol, water vapor and surface properties.* Journal of Atmos. Oceanic Technol., **13**: p. 777-794, 1996.
47. Weber, P.G., B.C. Brock, A.J. Garret, B.W. Smith, C.C. Borel, W.B. Clodius, S.C. Bender, R.R. Kay, and M.L. Decker. *Multispectral Thermal Imager mission overview*. in *Proc. SPIE*. 1999.
48. Schapire, R.E., Y. Freund, P. Bartlett, and W.S. Lee. *Boosting the margin: a new explanation for the effectiveness of voting methods*. in *Proc. 14th International Conference on Machine Learning*: Morgan Kaufmann. 1997.