A Scalable Learning System for Video Recognition

R. Porter, C. Chakrabarti, N. Harvey, G. Kenyon
Los Alamos National Laboratory
Los Alamos, NM, 87545
505-665-7508
rporter@lanl.gov

Abstract— Learning has become an essential part of many image and video processing systems, but it is not often used as an end-to-end solution. Some of the most successful demonstrations of end-to-end learning have been with convolutional, or shared weight networks. We are interested in how this approach can scale and have developed a flexible framework for implementing and training large scale convolutional networks called Harpo. We present an overview of the Harpo framework and describe a multi-level learning strategy used to optimize convolutional networks for particular features of interest in video data streams. Harpo is designed to exploit reconfigurable hardware to accelerate massively parallel convolutional network components and achieve real-time processing speeds. In this paper we present initial software experiments which use the system to segment exhaust plumes coming from military vehicles in Unmanned Aerial Vehicle video data.

TABLE OF CONTENTS

1. INTRODUCTION

Computer vision has been a challenging research problem for over 50 years and is commonly a target problem for a large number of fields including information theory, graph theory, set theory, probability and statistics. One of the reasons why computer vision is so challenging is the problem scale. From asking “which pixels are red?”, it is a small leap for humans to ask “which picture is my mother?”, but for computer vision, the problem difficulty has increased exponentially.

Much research in computer vision has focused on delineating, and then finding solutions, to useful sub-tasks e.g. edge/texture detection, shape analysis, motion estimation etc. Often these sub-tasks are combined to produce systems that can solve more complex problems [1]. Sometimes, with the addition of knowledge-based approaches, systems are produced which, in fact, can answer “which picture is my mother?” and other high-level questions for specific applications [2]. However, while manual decomposition is generally a good approach to solving complex problems, it can run into problems when the decomposition is poorly matched to the problem structure. This is particularly evident in computer vision where there is a strong interdependence between sub-parts, and robust performance of one part (e.g. shape characterization) depends critically on robust performance of other parts (e.g. segmentation).

An alternative to manual decomposition, which has gained increased attention in recent years, is machine learning and the promise of automatic problem decomposition. In this case, real data is used to drive the design process and learning algorithms attempt to extract the important sub-structures. The appeal of this approach for computer vision is simple: Solutions are represented as homogenous black-boxes that are universal computing machines. The black-boxes are programmed to solve problems through high-level teaching, not low-level design, and the approach scales to solve more complex problems by simply “adding more stuff” to the black box.

One of the most successful demonstrations of this approach comes from research in shared weight or convolutional neural networks. Using gradient based learning Dr. LeCun [3] was able to develop one of the most accurate systems available for optical character recognition. Using a similar approach several other researchers have reported robust recognition systems in a large number of practical applications [4], [5]. Researchers have also considered convolutional networks in the time domain [6]. This can potentially lead to more compact solutions and is naturally suited to processing temporal data streams, such as video.

In our research we are developing techniques and technologies that can help scale the convolutional network approach to the ever-increasing complexity of computer vision problems faced by the defense and intelligence communities. In this paper we provide an overview of our proposed contribution to this problem, which is novel in two ways:

0-7803-8870-4/05/$20.00© 2005 IEEE.

(1) A multi-level learning architecture that can help address the large-scale learning problem. This is based on our previous success with multi-level learning architectures for automating analysis of remotely sensed satellite imagery [7].

(2) The ability to automatically map convolutional neural network components to reconfigurable hardware accelerators, so that large-scale solutions are made computationally feasible. This is based on previous success in accelerating satellite image processing using reconfigurable hardware [8].

2. Background

Convolutional Neural Networks

A convolutional neural network looks much like any other multi-layered neural network, in which a collection of tunable processing elements are connected in a feed-forward graph. The only difference in convolutional neural networks is that each processing element represents an entire array of elements (processing layer), each associated with a particular pixel location, and whose tunable parameters are all equal (shared). The processing elements also usually share a common connectivity. A processing element is connected to other elements within a small local neighborhood (or window). It can also be connected to elements in other processing layers that are typically in the same pixel location, or within a small local neighborhood of that location. For linear processing elements each connection has a multiplicative weight and all connections are summed. In this case the processing layer implements a convolution of the image with a kernel defined by the shared weights. In simple terms, a convolutional neural network is a collection of tunable convolution operators connected in a feed-forward network.

Multi-scale Convolutional Neural Networks

Fukushima [9] was amongst the first to experiment with convolutional neural networks and obtained good results for character recognition by applying convolutional neural networks within an image pyramid architecture: processing layers alternate between convolution and sub-sampling. This multi-scale architecture has been now widely adopted and appears to provide a robust representation in many object recognition problems.

Cellular Nonlinear Networks

In recurrent convolutional networks a processing element can also receive input from its output. This feedback means processing elements can implement state variables and hence a wide variety of dynamic behavior. The state variable formulation of recurrent convolutional networks was independently introduced as cellular neural / nonlinear networks and is currently promoted by some researchers as a new paradigm for spatio-temporal processing [10]. Research in this field has centered on the design and analysis of network dynamics. Several works also combine the multi-scale nature of convolutional networks with the temporal nature of cellular nonlinear networks [11]. For the rest of the paper we will use the term convolutional network (or network for short) in the most general sense and mean it to include all multi-scale, feed-forward and/or recurrent variations.

Hardware Implementation

Convolutional networks benefit greatly from specialized implementation compared to implementation with a general purpose processor. This is mainly due to the local neighborhood communication required by each processing element in spatial, spectral and temporal dimensions. In reconfigurable hardware systems the memory architecture is tailored to the application and can therefore implement neighborhood processing very efficiently. In previous work we obtained over two-orders of magnitude speed-up for a 9 layer convolutional network [8]. Reconfigurable hardware is based on digital devices and we therefore restrict our attention to those convolutional networks that are discrete in time.

3. Harpo System Overview

We are developing a flexible software / hardware system for designing and executing large-scale convolutional networks for image and video processing. We refer to the system as Harpo and an overview of the system is shown in Figure 1.

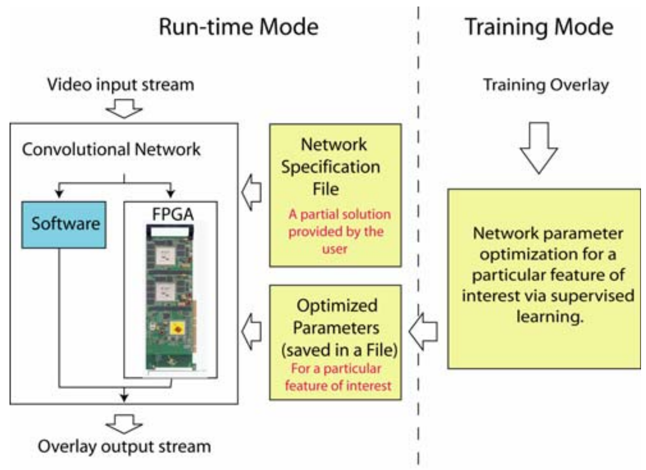

Run-time Mode
Training Mode
Video input stream
Training Overlay
Convolutional Network
Software
FPGA
Network Specification File
A partial solution provided by the user
Network parameter optimization for a particular feature of interest via supervised learning.
Optimized Parameters (saved in a File)
For a particular feature of interest
Overlay output stream

Figure 1 – Harpo system overview

There are two modes of operation: a *run-time* mode and a *training* mode. In the *run-time* mode (left of dashed line in Figure 1) Harpo is applied as an online system suitable for real-time implementation. At each time step, it receives a frame from a video sequence, executes some network and after some latency, produces an output frame.

In the *training* mode (right of dashed line in Figure 1) Harpo receives an additional training overlay. There is a frame to frame correspondence between the training overlay and the video input stream. Using supervised learning methods, the Harpo system attempts to optimize the network to produce output overlays that are in some sense *close* to the training overlay. The specific techniques will be discussed in Section 4.

The network that is executed by the Harpo run-time system is defined by two inputs: the network specification file and a network parameter file. The specification file contains information about the network that is constant from one training run to the next. The parameter file contains information about the network that has been learnt during training. Training mode takes the specification file as input and produces a parameter file. In run-time mode both specification and parameter files are inputs.

Network Specification File

The key to making the Harpo system flexible and scalable is a user defined input that we call the network specification file. This is a text file that defines a number of processing layers, and their connectivity, in an abstract and modular way. Figure 2 (top) provides an example specification file as well as the network architecture that Harpo will construct (below). To simplify Figure 2 the **Scale** processing layers are not shown but are implied by the changing size of the processing layers. Processing layers can be defined by the user and essentially receive an arbitrary dimension image cube as input and produce an arbitrary dimension image cube as output. The connectivity between layers define data paths that pass image cubes from one layer to the next. In Figure 2, two examples of user defined processing elements are **Linear** and **Scale**.

The specification file has a hierarchical lisp-like syntax. This serves two purposes. First, the specification file itself can be instantiated as a processing layer in a secondary specification file and hence large-scale solutions can be constructed with a typical building-block approach. Second, different learning strategies can be applied to different levels of the hierarchy (e.g. **Evolve** in Figure 2) enabling what we call multi-level learning. This is described more in Section 4.

The specification file defines the connectivity between processing layers, and the learning strategy, but it is also used to tailor fixed (or hard coded) functionality at run-time. An example of this in Figure 2 is the maximum **windowSize** of spatial filters, which due to hardware considerations, is fixed at run-time and cannot be learnt or adapted. Another example is the **initType** keyword seen in the first **Linear** processing layer. Setting the keyword to **Gabor** will force

```
( ThreeLayerPyramid
    ( inputMem .......................................... 0 )
    ( outputMem ........................................ 30 )
    ( iterationsPerFrame ................................ 1 )

    ( Linear
       ( inputMem ....................................... 0 )
       ( outputMem .................................... 1: 4 )
       ( numFeatures .................................... 4 )
       ( windowSize ..................................... 5 )
       ( initType ................................... Gabor )
    )

    ( Scale
        ( inputMem .................................... 1: 4 )
        ( outputMem ................................... 5 : 8 )
        ( scaleFactor .................................. 0.5 )
     )

   ( Evolve
       ( popSize ...................................... 100 )
       ( genSize ....................................... 20 )
       ( outputMem ..................................... 30 )
       ( trainFlag ...................................... 1 )

       ( SubNet
           ( Linear
                ( inputMem ............................ 5 : 8 )
                ( outputMem .......................... 9 : 16 )
                ( numFeatures ............................. 8 )
                ( windowSize .............................. 5 )
                ( initType ........................... Random )
           )
           ( Scale
                ( inputMem ........................... 9 : 16 )
                ( outputMem ......................... 19 : 26 )
                ( scaleFactor ............................. 2 )
           )
           ( Linear
                ( inputMem .......................... 19 : 26 )
                ( outputMem .............................. 30 )
                ( initType ........................... Fisher )
                (trainFlag ................................ 1 )
           )
       )
    )
)
```

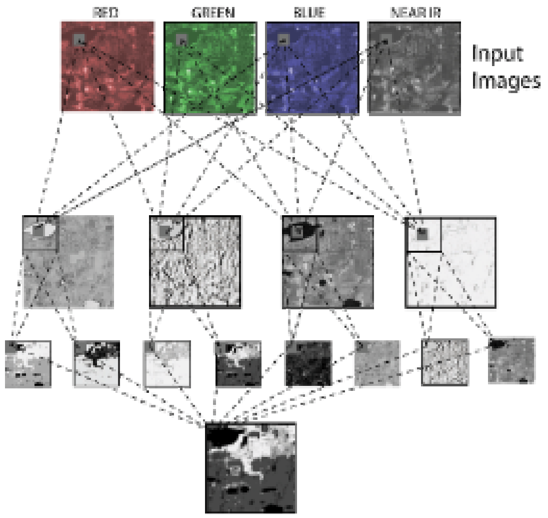


Figure 2. An example specification file and network.

the run-time system to use a fixed set of pre-defined kernels.

Network Parameter File

For other types of processing layers there is variable (or learnable) functionality. For example setting the **initType** keyword in the second **Linear** layer to **Random** will cause the run-time system to generate a random set of kernel weights. In the third **Linear** layer the same keyword is set to **Fisher** and implies the weights are found by using Fisher's linear discriminant. Processing layers with variable functionality are often accompanied by the **trainFlag** keyword. In this case adjustable parameters are found in the *training* mode and saved in the network parameter file. These parameters are then loaded and applied in the *run-time* mode.

Temporal Network Design

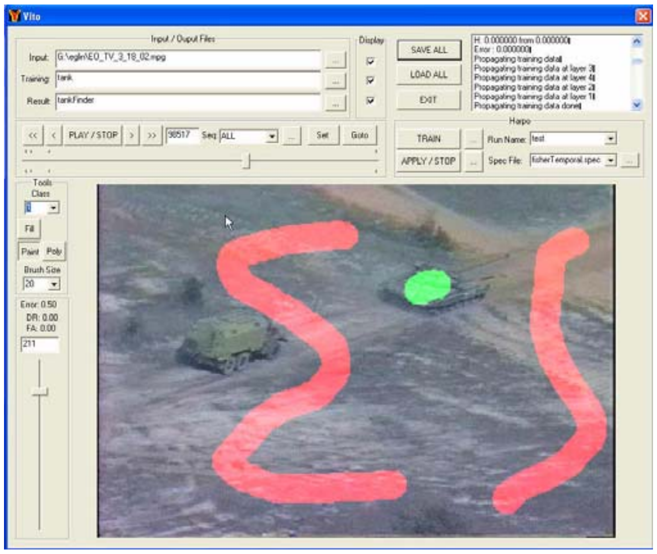
To exploit the temporal domain and/or introduce state variables the user must explicitly instantiate delay buffers (or FIFOs) in the network specification file. Sliding temporal windows are implemented by feeding the tapped FIFO outputs to subsequent processing layers. State variables are implemented by making recurrent processing layers which receive input from a delayed version of their output. The delay buffer is a processing layer like any other. It receives an image cube and produces an output cube at each time step. There is a fixed parameter **delayLength** that dictates how many time steps it will take for an input cube to appear at the output. The time step is an abstract quantity that is usually equal to or less than the frame rate of the input stream. It must be specified at run-time and is constant for all frames. The network level keyword, **interationsPerFrame**, specifies the time step relative to the frame rate. Temporal processing is entirely in the hands of the user and can be customized for various modalities and data types without adding complexity to the Harpo run-time system. Since we are generally interested in solutions that are local in time, the complexity of the specification file should not increase by too much.

Targeting Reconfigurable Hardware

A longer term goal of the Harpo system is to semi-automate the process of mapping networks to reconfigurable hardware. The specification file, and particularly our approach to the time domain, helps make this possible by forcing the user to model the data flow used in a real-time implementation. For example, memory bandwidth is explicitly allocated in the specification file via the delay buffer processing layer.

Providing Training Data with a Video Mark-up Tool

To help testing of the Harpo system we have implemented a graphical user interface that enables users to mark-up training data in video sequences. A screen-shot of the system is shown in Figure 3. Most native digital video formats like mpeg and avi are supported. The GUI allows the user to navigate the sequence and mark-up a training



Input: G:\eglin\EO_TV_3_18_02.mpg
Training: tank
Result: tankFinder
Seq: ALL
Run Name: test
Spec File: fisherTemporal.spec
Brush Size: 20
Error: 0.50
DR: 0.00
FA: 0.00

```
H: 0.000000 from 0.000000
Error : 0.000000
Propagating training data
Propagating training data at layer 3
Propagating training data at layer 4
Propagating training data at layer 2
Propagating training data at layer 1
Propagating training data done
```

Figure 3– Screenshot of the video markup tool

overlay by using typical drawing tools like paint brush, polygon and fill. For the two class classification problem features of interest are indicated with green markup and examples of the non-feature (or background) are indicated by red markup. These two types of mark-up are translated into +1 and -1 class labels to be presented to the Harpo system as a standard classification problem. Pixels that are not marked-up are assumed 'don't care' and do not contribute to error in the training procedure.

4. MULTI-LEVEL LEARNING

In training mode, the Harpo framework supports multiple types and levels of learning. Different learning methods can be combined sequentially and hierarchically.

By default, processing layers with the **trainFlag** will learn sequentially in the order defined by the specification file e.g. the first layer is trained, it is executed with its newly found parameters, propagating data to the second layer, which is then trained etc. This type of training operates at the layer level, is specific to a particular processing-layer and therefore usually user defined e.g. the Fisher Linear Discriminant is specific to linear processing layers.

Learning methods can also be combined hierarchically by using more general learning methods that wrap sub-networks. This enables the framework to obtain the benefit of both local and global search. Figure 4 provides an example of the multi-level learning system for the example in Figure 2. For each fitness evaluation of the evolutionary algorithm, the **SubNet** sub-network is trained via sequential

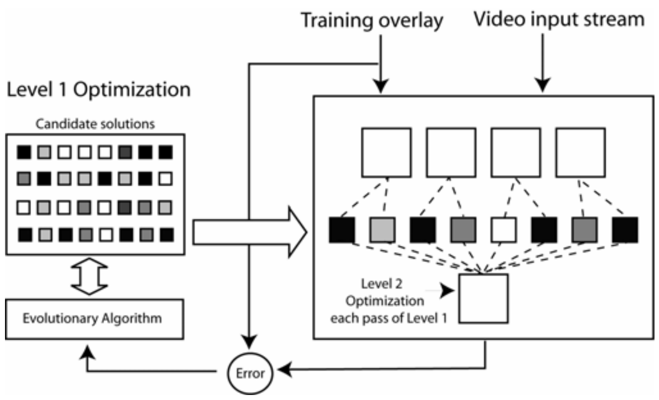


Figure 4– Example of multi-level feed-forward learning

(or feed-forward) local learning. The first **Linear** layer has no learnable parameters and for each evaluation will simply execute. The second **Linear** layer is wrapped by **Evolve** and therefore a population of candidate parameters is generated (randomly). For each candidate, this layer is executed, producing inputs for the third **Linear** layer which is then locally optimized (the **trainFlag** is true) using Fisher's linear discriminant.

Evolve

The **Evolve** layer interfaces to the GALib Genetic Algorithms written by Matthew Wall, at the Massachusetts Institute of Technology, and therefore has a wide selection of evolutionary algorithms available to it. Without further code specialization Harpo dynamically generates a linear chromosome at runtime based on the sub-network processing layers. Crossover points are constrained to fall at the boundary between layers and the mutation operator is layer specific and therefore user defined.

Boost

Harpo also implements several constructive learning techniques. These sub-networks are typically generalized additive models and have the form:

$$h(\vec{x}) = \sum_{i=1}^{D} w_i g_i(\vec{x}) \qquad (1)$$

where $sign(h(\vec{x}))$ predicts the class label for two-class classification and $g_i(\vec{x})$ is a sub-network. We use a technique called Boosting to incrementally build the model (1), in which case $g_i(\vec{x})$ is also called a weak learner [12]. The linear combination in (1) starts with zero terms. For each of D iterations we add one new weak learner $g_i(\vec{x})$ and weight w_i to the model. The procedure is greedy, but due to adaptive reweighting of training samples between iterations, boosting is able to build models that exhibit good generalization. A detailed discussion of the boosting procedure can be found in [13] and in most modern machine learning textbooks.

At each iteration of the boosting procedure we either use sample and test (e.g. stumps) or a learning algorithm to find $g(\vec{x})$ that will reduce error the most. A wide variety of weak learners $g(\vec{x})$ have been suggested, varying in complexity from simple thresholds to support vector machines. In the Harpo system a weak learner is implemented just like any other processing layer and the Boosting procedure can therefore be used at any point in the network hierarchy e.g. the Boost learning layer could wrap an Evolve learning layer and hence use an Evolutionary Algorithm as the weak learner.

Harpo implements two boosting algorithms. The first boosting algorithm is discrete AdaBoost [12]. In this case the algorithm combines *discrete* weak learners where $g(\vec{x}) \in \{-1,1\}$. The discrete adaBoost procedure provides a closed form solution for each w in (1). The second boosting algorithm is called Gentle AdaBoost and implements a type of real valued AdaBoost [13]. In this case, the algorithm also builds generalized additive models (1) but it combines weak learners with real valued outputs: $g_i(\vec{x}) \in \mathbb{R}$. For each weak learner we use weighted least squares to estimate scale parameters $a, b \in \mathbb{R}$ to adjust the weak learner: $a(g(\vec{x}) - b)$ before combination. In this case the value of w in equation (2) is fixed at 1 (since it is replaced by a above).

5. PROCESSING LAYERS

In this section we describe the processing layers that we implemented for the experiments in this paper. They are motivated primarily by our initial experiments into local learning algorithms and hence do not represent the complete set of processing layers that are currently available in the Harpo system.

Buffer

This processing layer is used to implement temporal processing and was outlined in Section 3. It has no learnable parameters.

Scale

This processing layer was used in Figure 2. The **scaleFactor** keyword is used to determine the output image size based on the input image. When reducing scale the

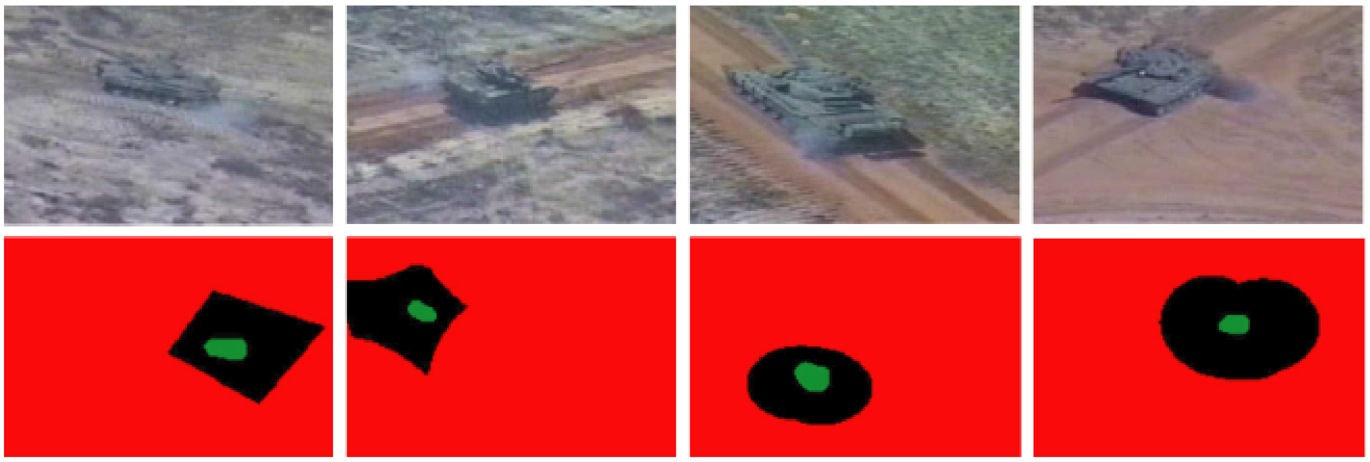


Figure 5 – Example training frames from 4 of the 10 sequences

default behavior is to apply Gaussian spatial smoothing and then subsample. When increasing scale we use linear interpolation. There are no learnable parameters.

Linear

This is also used in the Figure 2 example. It implements a spatial convolution of the input image with a set of weights known as a kernel. The keyword **windowSize** defines the width (and height) of the kernel. When the input is an image cube with depth *D*, the default kernel will include spatial neighborhoods from each image plane. In this case the kernel contains *N* = windowSize * windowSize * *D* weights. For the special case, windowSize = 1, the kernel has *N* = *D* weights and the convolution implements a simple linear combination of image planes. The default second-level training method is Fisher's linear discriminant.

Threshold

This layer can implement a hard, linear or tanh threshold. In training this layer finds optimal offset b and orientation $a \in \{-1,1\}$ so that $sign\big(a\left(g\left(x\right)-b\right)\big)$ has minimal error. In our experiments a threshold layer is placed after a linear layer to produce the discrete weak-learner required by the AdaBoost training layer.

6. EXPERIMENT DATA

To test the functionality of our system we developed an experiment where the target of interest is the exhaust plume produced by a military vehicle when it first moves off from a stationary position. This problem is a pixel classification or segmentation problem and therefore appropriate for our first experiments, which are based on simple, and relatively few building blocks. More complex tasks such as military vehicle recognition and identification will be investigated in future work.

The data consists of sequences of frames, each being 325 pixels wide by 256 pixels high with 3 colors and was recorded at 30 frames / second from a small (unstable) aircraft. We identified 10 separate exhaust plume sequences over a 20 minute recording period. Each sequence began with a stationary vehicle. As the vehicle moved off, a plume becomes visible (to our untrained eyes) and we used the video mark-up tool to manually segment 4 adjacent frames from each sequence. The sequences cover a wide variety of different scales, orientations and plume visibilities. In Figure 5 we show an example plume, and corresponding mark-up, from four of the ten sequences. Each image is a small tile (approximately 150 pixels by 100 pixels) cut from the original frame to make reproduction clearer. In our experiments, the Harpo system used the entire frame.

7. EXPERIMENTS

Our initial experiments investigate learning performance with respect to variation in network architecture and scale. There are 3 types of network investigated which we call Temporal, Multi-scale and Multi-stage. Temporal (Figure 6 – left) implements a single convolution in both temporal and spatial dimensions. Multi-scale (Figure 6 – middle) processes reduced scale versions of the input independently and the Multi-stage (Figure 6 – right) architecture implements the traditional hierarchical convolution network. We investigate the affect of adding processing layers using the Boosting procedure. This means each convolution layer in Figure 6 is actually by a linear combination of convolution layers. We used the discrete adaboost algorithm and Fisher's linear discriminant becomes the weak learner. For the Multi-scale and Multi-stage architectures we do not apply boosting to the final combination layer.

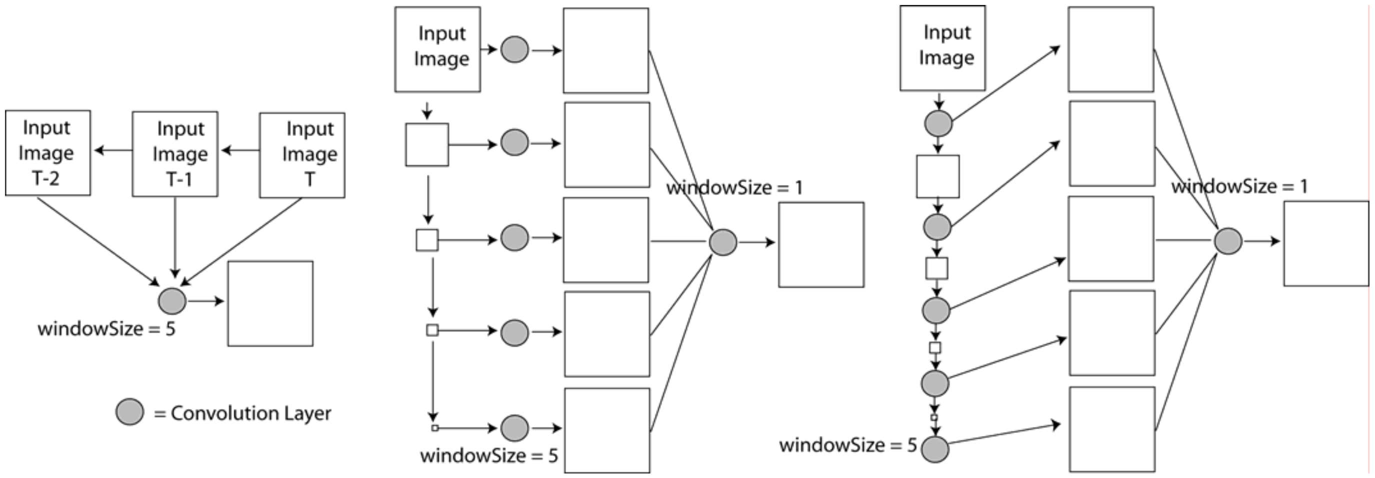


Figure 6 – The three architectures investigated: Temporal (left), Multi-scale (middle) and Multi-stage (right).

To compare each configuration (architecture and number of weak learners) we randomly chose 5 of the 10 plume sequences. Each configuration is trained on the 5 sequences and then applied to the remaining 5 sequences.

8. DISCUSSION

For the Temporal architecture we noticed very little improvement with additional convolution layers. In addition, the best performance for the 5 by 5 spatial window and 3 frame temporal window convolution was little improved over the Fisher discriminant applied to a single frame with no spatial information (a linear combination of red, green and blue channels). The multi-scale architecture appeared to benefit greatly from additional processing layers. But for the Multi-stage architecture, the feed-forward, greedy learning strategy seems particularly sub-optimal. We hypothesize this may be a common problem in deep pipeline architectures due to strong dependencies between layers.

Harpo was able to obtain reasonable results for the plume finding problem, particularly with the Multi-scale architecture. However, the poor performance of the Multi-stage architecture, which has traditionally performed extremely well for object recognition problems is a concern. Most reported successes with a Multi-stage architecture have used a back-propagation learning strategy and therefore our immediate goal is to incorporate back-propagation within the multi-level learning framework.

Another way to help the boosting feed forward approach find more cooperative behavior within a Multi-stage system is to run an Evolutionary Algorithm at the network level. This approach is already available within the Harpo framework and future experiments will determine if this is a viable alternative to back propagation.

Table 1. Milli% Error rates for 1 sampling of the plume problem (will average over 5 in the final paper)

Architecture	**Number of Layers used in Boosting**				
	1	3	5	10	20
Temporal	All errors are $\times 10^{-3}$ %				
Train	1.29	1.24	1.26	1.3	1.3
Test	1.88	1.89	1.99	2.1	2.1
Multi-scale					
Train	1.27	1.32	0.48	0.38	0.31
Test	1.55	1.41	1.18	1.26	1.24
Multi-stage					
Train	1.46	1.42	1.51	1.51	1.52
Test	1.91	1.90	1.81	1.74	2.03

9. CONCLUSION

The modular, regular architecture of convolutional neural networks is very attractive for learning, scalability and real-time implementation using massively parallel hardware. We have presented a flexible implementation and design environment for large scale convolutional networks called Harpo. This system enabled us to rapidly prototype and compare variations in network architecture for a plume segmentation problem. In future work we plan to extend the framework to include mechanisms for spatial, spectral and temporal invariances, as well as unsupervised learning techniques that are applied at run-time. With these extensions, it is hoped our system will produce robust solutions to complex object detection, recognition and tracking problems.

REFERENCES

1. Bennamoun, M. and B. Boashash, A structural-description-based vision system for automatic object recognition. IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics), 1997. 27(6): p. 893-906.

2. Haralick, R.M. and L.G. Shapiro, Computer and Robot Vision. Vol. II. 1993: Addison-Wesley Publishing Company Inc.

3. LeCun, Y., et al., Gradient-Based Learning Applied to Document Recognition. Intelligent Signal Processing, 2001: p. 306-351.

4. Nowlan, S. and J. Platt. A Convolutional Neural Network Hand TrackerProc. in Neural Information Processing Systems (NIPS94). 1995. Cambridge, MA.

5. Lawrence, S., et al., Face Recognition: Convolutional Neural Network Approach. IEEE Transactions on Neural Networks, 1997. 8(1): p. 98-113.

6. Chua, L.O., CNN: A Paradigm for Complexity. 1998: World Scientific Publishing Company.

7. Harvey, N.R., et al., Comparison of GENIE and conventional supervised classifiers for multispectral image feature extraction. IEEE Transactions on Geoscience and Remote Sensing, 2002. 40: p. 393-404.

8. R. Porter, N.H., S. Perkins, J.Theiler, S. Brumby, J. Bloch, M. Gokhale, J. Szymanski, Optimizing Digital Hardware Perceptrons for Multi-Spectral Image Classification. Journal of Mathematical Imaging and Vision, 2003. 19: p. 133-150.

9. Fukushima, K., Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition. Neural Networks, 1988. 1: p. 119-130.

10. Fortuna, L., et al., Cellular neural networks: a paradigm for nonlinear spatio-temporal processing, in IEEE Circuits and Systems Magazine. 2001. p. 6-21.

11. Behnke, S., Hierarchical Neural Networks for Image Interpretation, in Department of Mathematics and Computer Science. 2003, Freie Universitat Berlin: Berlin. p. 224.

12. Freund, Y. and R.E. Schapire. Experiments with a New Boosting Algorithm. in International Conference on Machine Learning. 1996.

13. Friedman, J., T. Hastie, and R. Tibshirani, Additive logistic regression: a statistical view of boosting. 1998, Dept. of Statistics, Stanford University.

BIOGRAPHY

Reid Porter received the B.S. degrees in electronic engineering and information technology and the Ph.D. degree in electronic engineering from the Queensland University of Technology, Australia, in 1996 and 2002 respectively. He is currently a staff member with the Space Data Systems group at Los Alamos National Laboratory. His research interests include massively parallel architectures, adaptive filters, machine learning and image and signal processing.

Neal Harvey received the B.S. degree in mechanical engineering from the University of Hertfordshire, U.K., in 1989, and the M.Sc. in information technology systems and the Ph.D. in nonlinear digital image processing from the University of Strathclyde, U.K. in 1992 and 1997, respectively. He is now a staff member at Los Alamos National Laboratory and his research interests include nonlinear digital filters, machine leaning, image processing and remote sensing.

Garrett T. Kenyon received his B.A degree in Physics from the University of California at Santa Cruz in 1984 and his M.S and Ph.D degrees in Physics from the University of Washington in Seattle in 1986 and 1990, respectively. He received further postdoctoral training at the Baylor College of Medicine, Division of Neuroscience, and at the University of Texas Medical School, Houston, Department of Neurobiology and Anatomy, the later under the supervision of Prof. Marshak. He has been a staff member in the Biological and Quantum Physics group at the Los Alamos National Laboratory since 2001. His research interests involve the application of computer simulations and theoretical techniques to the analysis of computation in biological neural networks.