GPU Facilitated Unsupervised Visual Feature Acquisition in Spiking Neural Networks

Blake Lemoine
University of Louisiana
Lafayette, LA
bal2277@gmail.com

Anthony S. Maida
maida@cacs.louisiana.edu

Abstract—This paper demonstrates that feature acquisition systems composed of spiking neurons trained by spike–timing–dependent plasticity (STDP) can effectively be scaled using General-Purpose computing on Graphics Processing Units (GPGPU). While previous studies have demonstrated this for classes with low intra–class variability, parallelization substantially increases the range of classes to which such a system can be productively applied. This system, like many existing feature learning systems, uses a hierarchical design based on the mammalian ventral visual pathway, alternating between selective layers that combine inputs and invariance layers that sample from inputs. Most systems do not, however, use spiking neurons as the brain does. Masquelier and Thorpe presented a system that produces highly informative features using a spiking convolutional network trained with STDP–based learning. However, this approach is costly in time and consequently is difficult to scale to a large library of internally complex features. Specifically, they reported that their system was incapable of learning the class “animal”, which was a major goal of that study. The present paper demonstrates that GPGPU parallelism can be leveraged to overcome the scaling limitations of the serial version. Highly informative features can then be generated in large quantities. The maximum complexity of classes learnable by this system is increased to encompass the natural class “animal” by GPGPU parallelization.

I. INTRODUCTION

Few machine learning algorithms operate directly on raw data. They instead operate on a feature-based representation of the data. In many visual recognition systems the features used to characterize data are handcrafted by the system designers. When handcrafted features are not used, supervised feature learning is the next most common approach. Silhouette, optical character, and trademark recognition [1] are example tasks where supervised feature learning has been successful. However, manually designing features or creating training sets for supervised learning both entail high costs in human intervention. Because of its obvious advantages, unsupervised feature learning is a fast growing field of research [2][3]. Recent work in unsupervised feature acquisition shows promising results. Several lines of research [4][5][6][7] have converged on similar solutions. The majority of emerging feature acquisition systems are designed, at least in part, based on the organization and logic of the sensory pathways in the neocortex. Specifically, the alternation of the neocortex between selective cells and invariance cells has majorly influenced the design of these systems since the Neocognitron [8]. However, these systems use the architecture of the ventral visual pathway but not its computational mechanisms. Sigmoid neurons and Restricted Boltzmann Machines (RBMs) dominate the field due to their well understood properties and the ability to train them using efficient algorithms with well understood convergence properties. While these types of neurons have well understood mathematical properties, they are only a rough approximation of the types of neurons used by the brain. The brain computes with spiking neurons. It does not train its neurons by explicit gradient descent but instead uses mechanisms like STDP. This paper is concerned with a system that uses both the architecture of the neocortex and its mechanisms [9].

The Masquelier and Thorpe [9] algorithm acquires features using a feedforward spiking convolution network that is trained using STDP [10][11]. The first layer creates an edge-detection feature map at four orientations as implemented by Gabor filters. From those basic features, the system builds more complex feature prototypes in the form of integrate–and–fire (IF) neuron weight matrices. The algorithm can practically learn only a few tens of features if implemented serially, thereby restricting the range of categories that can be learned. There are, however, meaningful categories for which ten or twenty highly informative features is sufficient. For face and motorbike images, they demonstrated good classification performance using a handful of highly informative features. However, one of their stated goals was to replicate in their system the human ability to quickly determine the presence or absence of an animal in an image [12]. Categories such as animal, which have high intraclass variability, were problematic for the algorithm due to the small number of features learned.

Scaling limitations are not uncommon amongst unsupervised feature learning systems. Understanding of how to successfully perform unsupervised feature acquisition is growing but the field still faces substantial difficulties. One of those difficulties is that many of the most effective mechanisms such as convolution and scale replication are very costly in computation time. Systems employ different trade-offs to lessen these costs. Some systems generate large quantities of features with low average relevance to category recognition [6][7]. These systems depend on the classification model training step to select the diagnostically relevant features. Other systems limit the number of features learned [9]. Some systems overcome these limitations by simply applying enormous amounts of computational resources to the task [3]. Finally, some systems combine the previously mentioned ap-
This paper demonstrates that a feature acquisition system composed of spiking neurons can be scaled to acquire hundreds of features if properly implemented using GPGPU. Increasing the size of the feature library increases the range of categories the system can learn to include the “animal” category which was an original goal of the Masquelier and Thorpe study [9]. The rest of this paper is organized as follows. First, an overview of the algorithm is presented. Each stage of the algorithm is described first from a conceptual level and then the parallelization of that stage is described. After the algorithm is presented, the experiments are described. The experiments include replication experiments as well as a novel experiment in which a class of images spanning multiple types of animals is learned. Finally, the authors discuss the results of the experiments and the conclusions which can be drawn.

II. ALGORITHM

A. Overview

The system parallelizes a hierarchical, feedforward, spiking convolutional network for image classification. The network acquires features during an acquisition phase and produces feature vectors during a classification phase. During the classification phase feature vector’s produced by the system serve as the input to a supervised classification model. Each layer in the feedforward stream performs one or more tasks such as feature extraction, winner-take-all (WTA) competition, or local lateral inhibition. The layers are organized to simulate an extrapolated model of the alternation of simple and complex cells found in the ventral visual pathway [14][15]. The input to the system is a grayscale image, normalized to a height of 300 pixels with aspect ratio preserved. Each input image is replicated at five different scales which are then processed by the layers labeled S1-C1-S2-C2 in Figure 1.

The S1 layer detects edges at four orientations: $\pi/8$, $3\pi/8$, $5\pi/8$, and $7\pi/8$. The C1 layer performs local WTA competition and local lateral inhibition. The S2 layer is composed of spiking integrate-and-fire (IF) neurons. Each S2 weight matrix represents a single feature prototype. An S2 weight matrix is shared by clone cells that are replicated at every possible scale and location. The C2 layer achieves global scale and position invariance by propagating only the spike time of the S2 cell with the earliest spike time. The C2 spike time is then used for either classification or weight update depending on the phase of operation (e.g. feature acquisition, image classification).

B. S1 Cells: Oriented Edge Detection

The S1 layer is made up of the elementary features that are combined to form intermediate-level feature prototypes. The feature acquisition algorithm is independent of the particular elementary features represented in the S1 map. The only assumption made by later layers is that the S1 layer produces a spike time map of the input image that preserves the topological properties of the input space. That is, two spike events that are located nearby in the map represent regions of the image that are near to each other. In this system the S1 features are four competing edge detectors with preferred orientations.

In regard to GPU parallelization, the S1 cells are implemented using two kernels. The first convolves the Gabor filters at four orientations with the scaled input image to produce four spike waves per scale at five spatial scales. This kernel produces a total of twenty spike waves. The spike times in these waves are inversely related to the responsiveness of a filter at that location. The second kernel is a WTA competition kernel. It propagates the earliest spike time of the four orientations at each location. All other spikes are set to the latest possible spike time.

C. C1 Cells: Local Position Invariance and Inhibition

C1 cells perform a local subsampling operation on the S1 layer output. This operation coarsely models the complex cells found in the visual system [16] to provide some degree of location invariance. Each C1 cell takes a 7x7 local area of an S1 map as its input and outputs the earliest one (min sampling). These cells each have one row and column of overlap in their input neighborhoods with their nearest neighbors in the C1 map. C1 cells are also locally inhibitory with other C1 cells to prevent uniformly oriented areas from dominating the feature acquisition process. The amount by which a C1 cell inhibits its neighbors decreases radially up to a maximum radial distance of five neighboring C1 cells.

In the GPU implementation, the C1 cells are implemented using two kernels. The first kernel performs the local position invariance operation. Each thread examines a C1 cell’s input region and sets its output to be the earliest spike time from that region. The output of the C1 subsampling kernel is the input to the C1 inhibition kernel. Each thread of the inhibition kernel examines the uninhibited spike time of its local neighbors, up to the maximal inhibitory distance. For each neighbor of a C1 cell with a spike time earlier than its own uninhibited spike time, that cell’s output time is inhibited by an appropriate amount. The output of the inhibition kernel is then input to the S2 cell layer.

D. S2 Cells: Combination of Elementary Features

The S2 layer is composed of IF neurons that discover composite features that are diagnostically informative for the target class. For each image scale there are four C1 maps. The S2 layer combines these maps into a single spike wave. After training, high feature salience at a location on the input image generates an early spike time at the corresponding location on the S2 map. Feature prototypes are realized as 16x16x4 weight matrices. A single S2 cell covers a 16x16 area of the C1 map over all four orientations. Each prototype is replicated as a clone cell at every scale and position. Each
set of 16x16x4 input spike times from the four C1 maps are run through the weight matrix via the IF accumulation algorithm to generate an S2 spike time.

A single kernel implements the IF algorithm. The implementation uses a variant of counting sort [17] to facilitate fast execution. Several simplifying assumptions about the input and output are made to allow for this optimization. The input spike times have a maximum and minimum firing time, \( T_{\text{max}} \) and \( T_{\text{min}} \). It is also assumed that the output need not be more precise than \( T_{\text{delta}} \). Therefore there are \((T_{\text{max}}-T_{\text{min}})/T_{\text{delta}}\) possible output values for each IF cell. Pseudocode for the algorithm appears in Figure 2. The computation cost of the algorithm is dominated by the sort function that finds the early spike times. This discretized algorithm runs in \( O(n) \) whereas a floating point precision IF cell runs in \( O(n \log n) \).

Each S2 cell has a 16x16x4 receptive field within the C1 map and, for each feature prototype, there is an S2 cell for each possible receptive field. For each S2 cell a correspondence is formed between the prototype weight matrix and the cells in its receptive field. The weights associated with the input neurons are accumulated into bins that correspond to their firing times. Once all weights are accumulated, the total in each bin is the amount of weight integrated from neurons that fired during that time span. The algorithm iteratively aggregates the weights from \( T_{\text{in}} \) through \( T_{\text{ax}} \) to compute the amount of weight integrated up to that time. Once the accumulated weight surpasses the firing threshold, the relevant bin’s time is output as the neuron’s firing time.

E. Weight Matrix Update

During training, weight matrices that represent prototypes are iteratively updated to learn regularities in the training data. Each matrix is updated once per iteration for each picture in the training set. STDP is used to compute the weight-update values. Whether a weight is increased or decreased depends on the relative order of the postsynaptic S2 spike time and the relevant presynaptic C1 spike time. The update rule given in Equation 1 determines the magnitude of the update to a weight. The learning constant \( a^+ \) varies from \( 2^{-6} \) to \( 2^{-2} \) as training progresses and the ratio \( a^+/a^- \) is held constant at \(-4/3\). For each feature, the weight update rule uses the time of that feature’s earliest spiking S2 clone as the postsynaptic time. The presynaptic times are given by that clone’s C1 inputs. If the presynaptic spike is earlier than the postsynaptic spike then the weight is increased, otherwise it is decreased. Two weight matrices might learn the same regularity. To prevent feature duplication, no two prototypes are allowed to use the same S2 location on an image to update their weight matrices.

\[
\delta w_{ij} = \begin{cases} 
  a^+ w_{ij} (1 - w_{ij}) & \text{for pre}_i < \text{post}_j \\
  a^- w_{ij} (1 - w_{ij}) & \text{otherwise}
\end{cases}
\]  

(1)
Parallel STDP weight update is implemented using three kernels. The first kernel performs update competition by finding the earliest spiking S2 clone for each prototype for each image. Two data points must be output by this kernel for each feature being trained: the time of the earliest S2 clone’s spike and its location on the C1 map. This is performed by a different variation on counting sort, referred to by the authors as counting selector. One thread is spawned per S2 clone. The output buffer which these threads share is an array constructed in the same manner as was time_steps in Figure 2 and is initialized to -1. For each thread, that thread’s id is saved into the array location that corresponds to the firing time of that thread’s S2 clone cell. The first nonnegative entry in the array is the relevant clone neuron. The location of the first nonnegative entry is the postsynaptic firing time.

The input to the second update kernel is the presynaptic inputs and the information about the earliest S2 firing times computed by the first STDP kernel. For a given weight matrix and image a thread corresponds to one weight matrix entry. Each thread computes whether its weight matrix value should be increased or decreased. The sign of the change is saved into an output array. The final update kernel computes for each prototype the resulting weight matrix by iterating over the outputs of the second kernel and iteratively applying the STDP update rule given in equation 1. This process is illustrated in Figure 3.

III. EXPERIMENTAL EVALUATION

Features acquired during the learning phase are evaluated based on their usefulness for classification. Two types of experiments were run differentiated by the diversity of the training data used in them. A single class problem trains features on a set of images that are conceived of as a single class. A multi-class problem trains features on a set of images conceived of as being from multiple classes. During the classification step, a single S2 competition kernel produces the C2 output spike times. For each image to be classified the C2 layer produces a single spike time per feature. The C2 spike times generated by all features on a given image form a feature vector for that image.

A single class problem is evaluated by training a two-class classifier. That classifier uses images like those in the training data as the positive class and background images as the negative class. The classifier used for this type of problem is a naive classifier with two thresholds. One threshold determines whether a feature is present in an image. Another threshold determines whether enough features are present in an image for it to belong to the positive class. Both thresholds are trained to maximize accuracy as in [9]. Using such a naive classifier illustrates the highly informative nature of the acquired features. A multi-class problem is evaluated by training a multi-class SVM [18] that uses the classes represented in the training data as class labels. The single class classifier answers the question: “Is this image like the training data?” The multi-class classifier answers the question: “Which class from the training data is this image like?”

The validation methods used for the two types of problems are different. The single class problems are evaluated using two-fold validation. Features were learned using half of the positive class images. A classifier was then trained using those same images as well as half of the negative class images. The accuracies reported in Figure 4 represent the accuracy of those classifiers when applied to the holdouts from the positive and negative classes. The multi-class problems are evaluated by multi-class SVMs trained using stratified five-fold cross validation. Features for this class of problems are trained on a set of images equally divided between the classes. The remaining images from each class are then used for SVM training. One fifth of the evaluation images are not used to train each of five models which are evaluated using the holdouts.

The experiments are further divided into replication experiments and novel experiments. Data sets for the experiments were drawn from the CalTech-101 data set [19]. Two single class problems and one multi-class problem from [9] were replicated. The single class problems trained twenty features on images of faces and twenty features on images of motorcycles. The features were then evaluated using naive threshold classifiers as described above. The results of this evaluation are presented in the right column of Figure 4a. The left column of Figure 4a contains the results reported by Masquelier and Thorpe for comparison. The multi-class problem trained forty features on a data set containing faces, motorcycles, and background images. These forty features were used to train a three-class SVM. The confusion matrix of that classifier is presented in Figure 4c. The confusion matrix for the classifier learned by Masquelier and Thorpe for this problem is given in Figure 4b for comparison.

The novel experiments used images drawn from ten different classes in the CalTech-101 data set, all of which were types of animals. Masquelier and Thorpe comment that “Classes with more intraclass variability (for example, animals) appear to pose a problem with our approach”. The Cal Tech machine vision data sets do not include a data set with more than one type of animal. They do, however, include many data sets consisting of a single animal type. A novel data set was constructed from ten such data sets. The
animals chosen for the new composite set are bears, camels, chimps, dogs, elephants, elk, goats, leopards, raccoons, and zebra. Forty images of each animal type were randomly selected for inclusion in the feature training set. The images from those classes not included were used as holdout images. Examples of the images used to form this training set are given in Figure 6. It should be noted that while the perspective of faces and motorcycles was normalized the perspective of animals was not.

Two hundred features were learned from the animals training set. Two experiments were run using those features. In one, the data set was conceived of as containing only a single class of images. The other conceived of the data set as containing ten distinct classes of images. The single class and multi-class experiments for the animals data set were run in the same manner as the comparable replication experiments. Both experiments used the same two hundred features. The results of these experiments are presented in Figures 5 and 7. The overall accuracy reported in Figure 5 reflects the performance of a naive threshold classifier on the holdout images combined with background images. Figure 5 further breaks down classifier performance based on type of animal. Figure 7 is the confusion matrix of the multi-class SVM trained on the holdout images. The overall performance of the confusion matrix in Figure 7 is low: 20.4%. These results are discussed below.

IV. DISCUSSION

This paper studied the effects of scaling of an algorithm for the unsupervised acquisition of visual features using GPGPU parallelism. The serial implementation was able to practically learn tens of features while the parallel implementation was able to learn hundreds of features using OpenCL on a GeForce GTX 460 graphics card. The parallel implementation significantly increased the speed with which features were learned as compared to evaluating the same OpenCL code on an Intel i5 650 cpu as seen in Figure 8. This speedup allowed a larger number of features to be learned which in turn allowed the system to learn to identify the category animal where the serial system could not. The system was
therefore able to learn a class with a high degree of intra-class variability, a qualitative improvement over the serial implementation. The system was not, however, successful at learning the animals data as a ten class problem.

There are several possible reasons why the multi-class problem failed. It is possible that two hundred features was still not enough. The SVM did perform better than chance which indicates that the features were useful differentiating the classes to some extent. It is possible that the inter-class variability of animals is based on elementary features other than edge detection. Features like color and texture may play a significant role in differentiating animal classes. It is also possible that another hierarchical layer is needed to differentiate animals of different types. All animals have eyes, ears, mouths an limbs but these are not composed the same way in all animals. It is possible that S3 cells could be trained to represent these variable compositions of intermediate-level features. Further research is needed to investigate each of these possibilities.

In conclusion, this study parallelized an algorithm for the unsupervised learning of visual features that is inspired by the mammalian visual system. This system uses spiking neurons which is a closer approximation to the computational mechanisms of the brain than are other unsupervised feature learning systems. Methods for parallelizing that algorithm using GPGPU technology were presented. The parallel implementation was evaluated by using trained features in single class and multi-class classification problems of varying complexity. Finally, it was demonstrated that the class of problems which can be solved using this algorithm is significantly expanded by scaling the number of features learned.

**REFERENCES**


<table>
<thead>
<tr>
<th>Bears</th>
<th>Camels</th>
<th>Chimps</th>
<th>Dogs</th>
<th>Elephants</th>
<th>Elk</th>
<th>Goats</th>
<th>Leopards</th>
<th>Raccoons</th>
<th>Zebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>87.25</td>
<td>9.09</td>
<td>5.45</td>
<td>4.90</td>
<td>3.82</td>
<td>3.96</td>
<td>2.68</td>
<td>1.58</td>
<td>5.0</td>
<td>3.13</td>
</tr>
<tr>
<td>0.98</td>
<td>5.45</td>
<td>17.27</td>
<td>14.7</td>
<td>12.98</td>
<td>6.93</td>
<td>13.39</td>
<td>13.68</td>
<td>15.0</td>
<td>18.7</td>
</tr>
<tr>
<td>0.98</td>
<td>10.0</td>
<td>10.91</td>
<td>4.55</td>
<td>9.16</td>
<td>10.89</td>
<td>10.71</td>
<td>8.95</td>
<td>12.14</td>
<td>8.33</td>
</tr>
<tr>
<td>0.98</td>
<td>9.09</td>
<td>12.73</td>
<td>10.0</td>
<td>14.5</td>
<td>11.88</td>
<td>8.04</td>
<td>4.74</td>
<td>5.71</td>
<td>6.25</td>
</tr>
<tr>
<td>1.96</td>
<td>10.9</td>
<td>10.0</td>
<td>18.18</td>
<td>14.5</td>
<td>12.87</td>
<td>14.29</td>
<td>12.11</td>
<td>9.29</td>
<td>11.46</td>
</tr>
<tr>
<td>0.0</td>
<td>8.18</td>
<td>3.63</td>
<td>5.45</td>
<td>8.4</td>
<td>11.88</td>
<td>5.36</td>
<td>4.21</td>
<td>7.14</td>
<td>7.29</td>
</tr>
<tr>
<td>0.98</td>
<td>12.72</td>
<td>11.81</td>
<td>10.91</td>
<td>13.74</td>
<td>9.9</td>
<td>11.6</td>
<td>10.53</td>
<td>12.14</td>
<td>11.46</td>
</tr>
<tr>
<td>2.94</td>
<td>11.81</td>
<td>10.0</td>
<td>3.64</td>
<td>4.58</td>
<td>3.96</td>
<td>9.82</td>
<td>30.53</td>
<td>14.29</td>
<td>10.42</td>
</tr>
<tr>
<td>2.94</td>
<td>18.18</td>
<td>15.45</td>
<td>18.16</td>
<td>16.03</td>
<td>22.77</td>
<td>20.54</td>
<td>13.68</td>
<td>17.14</td>
<td>18.75</td>
</tr>
<tr>
<td>0.98</td>
<td>4.55</td>
<td>2.73</td>
<td>3.64</td>
<td>2.29</td>
<td>4.95</td>
<td>3.57</td>
<td>0.0</td>
<td>2.14</td>
<td>4.17</td>
</tr>
</tbody>
</table>

Fig. 7: The confusion matrix for the ten class classifier on the animal data set using 200 features.