The Pennsylvania State University
The Graduate School
College of Engineering

OBJECT RECOGNITION USING STRUCTURED FEATURE EXTRACTION WITH A RECOFIGURABLE, NEUROSYNAPTIC PROCESSOR

A Thesis in
Computer Science and Engineering
by
Priyanka Gomatam

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2017

The thesis of Priyanka Gomatam was reviewed and approved* by the following:

Vijaykrishnan Narayanan
Distinguished Professor of Computer Science and Engineering and Electrical Engineering
Thesis Advisor

John Sampson
Professor of Computer Science and Engineering
Co-Advisor

Chitaranjan Das
Distinguished Professor of Computer Science and Engineering
Department Head of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

Neuromorphic hardware has culminated increased interest, with a focus on designing efficient platforms that can support neural network tasks. Many algorithms used in applications today extensively perform pre-processing of data, in which structured features of the data are extracted and used for classification. The idea is to explore the ability to implement a structured computation in a neuromorphic platform. This involves leveraging the operations that are best suited for neuromorphic hardware, and using them to achieve the same results as a traditional algorithm. In this paper, a case study of mapping the feature extraction stage of pedestrian detection using Histogram of Oriented Gradients (HoG) onto a neuromorphic platform is performed. Further, this neuromorphic feature extractor is then connected to a neural network based classifier. The performance of the feature extraction done by a 1:1 mapping of the algorithm is evaluated against other neuromorphic implementations, as well as an FPGA implementation. The neuromorphic platform chosen for this experiment is IBM's TrueNorth, a Neurosynaptic System.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Motivation

As the popularity of machine learning and artificial intelligence algorithms is increasing, so is the need for highly intelligent systems with high processing capabilities. Current supercomputing and high performance architectures have shown promising results in terms of speed of computing. At the same time, there has been an increased interest in hardware platforms designed to support neural network inference tasks. Attempts to achieve this has led to the branching out of computing research, and one such extension has been the advancements made in neuromorphic computing. The idea is to explore the capabilities and drawbacks of the neuromorphic computing paradigm by implementing an existing algorithm and using the results to evaluate its characteristics. Research into systems supporting classification and other tasks via neural network inference has been growing rapidly. For example, [2–5] propose a variety of neuromorphic hardware platforms with neural network accelerators. The idea is to have highly parallel processing units, with specialized functionality for each unit. The von Neumann design involves data movement between memory and the processing components. Hence, the platforms are optimized according to the task, in order to reduce these movements, and thereby make them very efficient for the task at hand. The field of neuromorphic computing has evolved with the goal of innovating computers with the ability to perform tasks that the brain does efficiently. Improving existing high performance

computing systems while adhering to power constraints is becoming increasingly challenging. While there has been a lot of research to explore continual exploiting of these architectures, lateral research focusses on building new and better models to overcome the drawbacks of the existing ones. In this exploration, the model of the brain comes as a natural inspiration to building a computing device. Furthermore, the fact that a computation that requires about 12 GW of power on a hypothetical today's-computer is performed in about 20 W of power by the brain makes it an even more interesting design to mimic. [6]

1.2 Cognitive Computing Models

Over the recent few years, there has been significant focus on building intelligent machines in the field of artificial intelligence. The difference, however, that the cognitive computing field is looking to achieve, is to build an aggregate model that prioritizes software as well as hardware aspects. The approaches to neuromorphic computing can be broadly classified into three types. Companies like Google, Apple and Facebook have performed various software implementations of artificial neural networks (ANNs), by using deep learning methodologies. This approach primarily focusses on algorithmically modelling specific functionalities of the brain such as knowledge and reasoning, planning and communicating. On the other hand, the second approach focusses on architecturally modelling the brain, identifying the physical properties of the brain and understanding how their interactions help in performing tasks. The third approach integrates the first two approaches of machine learning and brain modelling architectures, and one such implementation is IBM's TrueNorth neuromorphic architecture. The TrueNorth architecture provides a holistic environment to perform neuromorphic computing, from the algorithmic level to the architectural level. This is the platform used for our implementation of pedestrian detection, although the overall approach is generic. Chapter 2 discusses the analysis and background of the architecture, Chapter 3 elaborates on the algorithm and its implementation, Chapter 4 provides an overview of the methodology and validation, Chapter 5 discusses the results and the conclusion.

Chapter 2
Analysis and Background

2.1 Introduction

The IBM Neurosynaptic System has been targeted for this paper. The high-level design is generic and is broadly applicable to other neuromorphic systems as well. This section provides some background on the TrueNorth architecture and its programming.

With the effort to create a holistic environment, IBM's TrueNorth facilitates the neuromorphic design by providing a system at every layer of abstraction [7]. There are five provisions made in order to support different stages of the programming cycle.

- Neuromorphic Simulator: It helps simulate a set of neurosynaptic cores which make up the architecture.
- Spiking Neuron Model: This is the main "arithmetic" unit of the core. It performs computations in a spiking neural network fashion, where the output fires if the input values are higher than a preset membrane potential. This is explained in further detail.
- Programming Model: The programming is done using the MATLAB environment, where the overall design is made of basic building blocks called "corelets", and each corelet has a network of neurosynaptic cores. The axon types (inputs), neuron types (which determine the output), IO for connectivity and cores can be set up and programmed to fit the needs of the application.

- Library: The library consists of pre-designed corelets in the program library which help implement large scale algorithms.
- Laboratory: A tool with an end-to-end software environment to test how to use the simulator with example programs and visualizations of results.

2.2 Spiking Neural Networks

Traditional computers use digital representations and boolean logic to perform computations. On the contrary, cognitive architectures manifest an event-driven mechanism with higher levels of distribution and parallelism. The underlying concept behind a spiking neural network is that, the neurons do not fire at regular intervals (such as at each clock cycle), but instead, fire only when the weighted sum of the inputs, known as the membrane potential, exceed a predefined threshold value. Essentially, TrueNorth implements a network of **integrate-and-fire** spiking neurons [8]. A dense network of neurons are created, and the *synapses* receive excitements from the input pins known as axons. If the input spike received is over a certain threshold, the neuron cell generates a spike. The basic principle of spiking neural networks is different from traditional neural networks, thereby calling for a new set of learning rules. The spiking neuron model can be extended to develop a convolutional neural network (CNN), which can be used for more widespread and complex applications.

Since this type of an architectural design is fundamentally different from conventional architectures, programming it also requires different approaches. The conventional program model is usually implemented in digital circuits which are typically general-purpose in nature, and von Neumann architectures call for a separate memory unit with which the processing unit has to communicate, thereby making the computational unit and the memory unit as two distinct and separate entities. TrueNorth, on the other hand, provides a highly parallel structure to program and the memory and computational units are integrated. The idea behind the design is also inherently overcomes the bottleneck that is usually created in a separated memory and computational unit design [8].

2.2.1 Crossbar Structure

The spiking neural network (SNN) model employs an interconnected network of neurons, as shown in the Figure 2.1. The output neuron spikes depend on the weights of the synapses. This kind of a model is particularly useful for applications with inherent pattern matching solutions. SNNs can be extended to work for more complex and compute-intensive applications such as video detection. The design of the neuromorphic architecture is event-driven. Furthermore, the model can be extended to perform certain functions such as its ability to respond to biological sensors, and to simultaneously process data from multiple input sources.

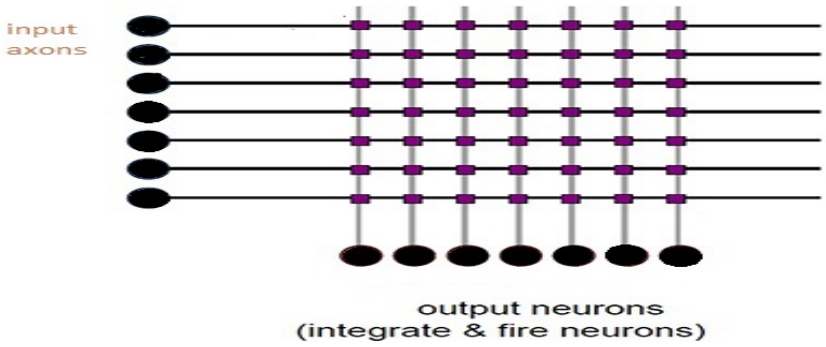

Figure 2.1: Crossbar structure of Spiking Neural Networks

2.3 The TrueNorth Architecture

The design of the specific neurosynaptic core that is used has 256 axons, connected directly to 256 neurons and a 256 x 256 crossbar, as illustrated in the previous section [2,8]. The neuron fires based on the axon input and the crossbar connection. These neurons can be connected to any other axon of another core. The firing of this neuron then acts as an input to the other core's synaptic network. In this manner, neuronal spikes event-drive the operations of the entire system.

If the synapse value is not zero for a given input-output axon-neuron duplet, then the membrane potential of that neuron is incremented. The incremented weight depends on the axon type, and can be established either deterministically

or stochastically. If the accumulated membrane potential exceeds its threshold, then the neuron spikes. Consequently, this spike can be sent to a corresponding target axon through the communication network. Now let us compare this model to the integrated memory-communication-computation model discussed earlier. The synapses represent memory, the neuron-axon connections represent communication and the neurons represent computation. The entire system is event-driven and the design overcomes the von Neumann bottlenecks.

The inputs are transformed from a spatio-temporal stream of input spikes into a spatio-temporal stream of output spikes. The architecture consists of one million individually programmable neurons and 256 million individually programmable synapses. The memory is represented by the synaptic network which can store over 400 million bits. The communication model in TrueNorth is hierarchical, and the spikes are routed through the network among cores. This design reduces network traffic and makes communication quicker and easier. The cores can be tiled together, and there are 4096 cores present. The interfacing of the cores does not require any extra circuitry.

2.4 Programming Model

The programming model [7] provides a mechanism to specify the network of cores, along with the inputs and outputs. The desired algorithm can be implemented by taking a divide-and-conquer approach. The core is constructed by creating a fundamental, indivisible building block in the form of a single neurosynaptic core. The programming environment allows the construction of *corelets*, which encompass the set of inputs, outputs and the synaptic values.

The corelet, essentially, encapsulates all the synaptic interactions in the network, such as the number of cores, neruon and axon types and their connectivity, and only reveals the input to the network and output from the network. The input is given to the corelet through connectors, and the output is sent out through connectors.

As mentioned before, a divide-and-conquer approach is taken to organize the corelets in a hierarchical fashion. A given corelet can be made of one or more sub-corelets. Each sub-corelet has the same abstraction as the corelet itself, it has a set of input and output connectors and it encapsulates the actual synaptic behavior. In this manner, the computation can be broken down into smaller solvable units,

each unit being processed by a corelet or a subcorelet. The output of one subcorelet can be connected as the input of another corelet and thereby the communication helps establish a modular computational unit.

Once the input and output connectors are set up for the corelet, the actual computational design of each one can be set up. Recall that the architecture follows a spiking neural network model. The axon acts as an input, and whenever the synaptic matrix is a non-zero value for an axon-neuron pair, the neuron increments its membrane potential. The programming model allows the axon types to be defined, which means that the value by which the membrane potential is incremented can be determined by specifying an axon type to an incoming input spike. The programming model then allows for the definition of the neuron types. This is done by specifying weights for each axon type and whether they are positive or negative. The way in which the weight is assigned depends on the application network and what we want our end result to be.

The axons are labeled with one of four possible axon types. The weight of the synapse is determined by the entry associated with the corresponding axon type at each crossbar point that is set to 1. If the chip is run on a stochastic mode, then a random number is added to the threshold to determine if there is a spike or not.

After the axon types and neuron types are set up, we can define the synaptic matrix. This is done in order to link the input spikes from the axons to the corresponding output neuron. Note that it is necessary for all the input spikes to be associated with some output neuron, because the input spike must have a destination. On the other hand, it is not necessary for all the neurons to have a destination axon. If certain input axons do not have a destination neuron, then those connections are marked disconnected in the synapse matrix.

Connectivity on the TrueNorth chip is two-fold: the local connectivity within a core that connects inputs to outputs, and the global connectivity that allows connection between cores, and also connections to be made off-chip.

A TrueNorth core consumes appox. 16 uW, this implies that 4096 cores consume 66 mW at 0.8V [2, 8].

Clearly, the programming model allows for a methodical programming structure. It is designed in such a way that the underlying neural network architecture can be efficiently used by mapping our algorithm onto the network in a neural network fashion.

Additionally, another design approach is presented with TrueNorth, wherein the configurations of the hardware are not programmed. Known as the Energy Efficient Deep Neuromorphic Network (Eedn) [9], it is like a convolutional neural network that is specific for the TrueNorth platform, and uses back propogation to update the weights of the hidden layers during training. The key features of Eedn include having a spiking neuron model which have a threshold activation function, a hidden full precision layer that maps to ternary weights (-1, 0, 1), and filters sized to match the crossbar size of Truenorth.

Chapter 3
Feature Extraction for Object Detection on TrueNorth

3.1 Introduction

At this point in the digital era, it has become clichéd to state that tremendous amounts of data have been generated over the last few years. This has given rise to opportunities for innovating interesting technologies that leverage the abundant availability of data. Particularly, many useful applications have emerged in the fields of image processing and computer vision. Object detection is one of the main functionalities of several exciting computer vision applications such as face recognition and video surveillance. A popular algorithm used to perform object detection is the Histogram of Oriented Gradients (HOG), which is a descendant of the Scale-Invariant Feature Transform (SIFT) algorithm. The original HOG algorithm was proposed by Dalal and Triggs in 2005 [10], and gives an interesting method to preprocess images to create a feature descriptor, that can be used to train a classifier. Object detection is used in machine learning applications, which are the primary applications that TrueNorth is targeted at. Therefore, the implementation of HoG can be structured in order to actually use the underlying neuromorphic architecture to perform its functions.

Given that computer vision applications are exactly the kind of applications that we want to perform on a high performance, low power machine, it acts as a good network to experiment with on the TrueNorth architecture. The description of the algorithm, its traditional implementation and the TrueNorth version are

described in the following sections.

3.2 Histogram of Oriented Gradients

With an increase in the use of the Internet, the amount of multimedia data generated have been considerably high. In order to process such type of data, several systems have been designed to effectively represent the multimedia information. Visual descriptors, also known as image descriptors, are a way of describing images and can be used for applications such as object detection. The Histogram of Oriented Gradients is a feature descriptor algorithm that is effective to perform object detection.

3.2.0.1 Traditional Implementation

The method computes a histogram of image gradients, which provides a way of characterizing the appearance and shape of local objects. The image is converted into a dense grid and the histogram is computed over the grid. Basically, the image is divided into smaller regions of equal size and these regions are known as cells. The gradient direction or the edge orientations are computed for the pixels of each cell. The histogram entries of all the cells are then combined together to form the overall descriptor of the image.

The combined histogram representation of the image is known as *Histogram of Oriented Gradient descriptors*. In order to perform object detection, the obtained HoG descriptor is used to train a classifier such as a support vector machine. The overall flow of the HoG object detection chain is shown below.

Each part of the block diagram illustrated above are described in the following sections.

1. Input Image

The input image given to the feature descriptor can typically be of any dimensions. However, the images used to test out the algorithm originally were 64 x 128 in size. The original image is assumed to be in an RGB colour space, but is converted to greyscale before applying the algorithm to it.

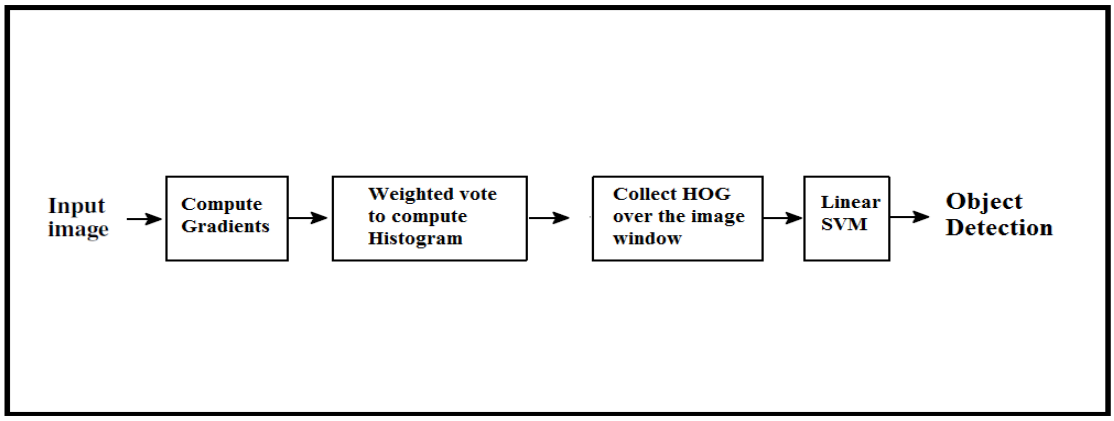

Figure 3.1: HoG Object Detection

2. Cell division

The original image is divided into grid of 8 x 8 pixel cells. Dalal and Triggs experimented with several cell sizes and determined that an 8 x 8 cell is of optimal size. The histogram is computed over each cell by taking a weighted vote of the gradient directions. Therefore, if the input image is 64 x 128 and each cell is 8 x 8, then there are 128 cells in the image. This gives rise to 128 histograms and the final descriptor is determined by concatenating the histogram values.

3. Computing the Gradients

There are several schemes to calculate the gradient. A 1-D discrete derivative mask is used over the image. This can be 1-D point derivative, cubic corrected or Sobel masks. It was determined that a 1-D centred point derivative worked best in calculating the gradient.

The gradient is computed in both the x- and y- directions. In order to calculate the x-direction gradient, the mask is [-1, 0, 1], and the transpose of it is used to calculate the gradient in the y-direction. It was concluded that using large masks usually degrades performance. Typically, for colour images, the gradients are calculated separately for each colour channel and the one with the largest norm value is taken as the pixel's gradient vector. However, in our study, the colour image is transformed into a greyscale image, which makes the calculation of the gradient more straightforward.

$$D_x = [-1\ 0\ 1] \tag{3.1}$$

$$D_y = [-1\ 0\ 1]' \tag{3.2}$$

The first order image gradients are computed as shown above. The gradient values capture information such as contour, silhouette and texture but is resistant to variations in illumination. Variant methods could possibly include second order image derivatives, but that is decided based on the type of objects being detected. As our target is pedestrain detection, we follow the derivatives used in the original paper.

So, given an image I, the x and y derivatives using the following convolution operation:

$$I_x = I * D_x \tag{3.3}$$

$$I_y = I * D_y \tag{3.4}$$

4. Gradient Direction

The gradient direction is determined in order to find out the direction in which the gradient is prominently present. After computing the derivatives of the image, the angle and magnitude are computed as follows:

$$\theta = arctan(Ix/Iy) \tag{3.5}$$

The inverse tangent of the fraction between Ix and Iy gives the dominant angle of the gradient descent in the cell. Similarly, the magnitude of the gradient is obtained by the following:

$$Magnitude = \sqrt{I_x^2 + I_y^2} \tag{3.6}$$

The derivative value, angle and magnitude are computed for each pixel of the cell.

5. Orientation Binning

In the next step, the cell histograms are created. The histogram is formed by

obtaining a weighted vote from each cell for the orientation, based on the values found in the gradient computation. The histogram cells can be spread out either over 0 to 180 degrees or 0 to 360 degrees.

In this project, we consider 18, evenly spaced histogram channels over 0 to 360 degrees, therefore, each bin corresponds to a range of 20 degrees. The weighted vote can be done in several ways. Either the magnitude of the gradient can be used or the count of the angles can be used. In this case, we use a count of the angles in order to get the weight vote cast by each pixel. In the results section, however, there are comparisons with a model that uses the magnitude instead of the count.

Additionally, the original algorithm performs normalization by dividing the image into blocks. Each block contains a certain number of cells and the block striding value depends upon the level of normalization we want to obtain. Within each block, the histogram values of each cell is normalized either using L1-norm or L2-norm. Normalizing the histograms was known to show significant improvement in the results obtained as opposed to non-normalized data, which is also depicted in the results section of our experiment.

6. Forming the HOG Descriptor

After calculating the histogram of oriented gradients for every cell using the aforementioned process, the final feature descriptor is obtained by combining the histogram values of all the cells. This descriptor provides an efficient representation of the image and can be used for training and classification.

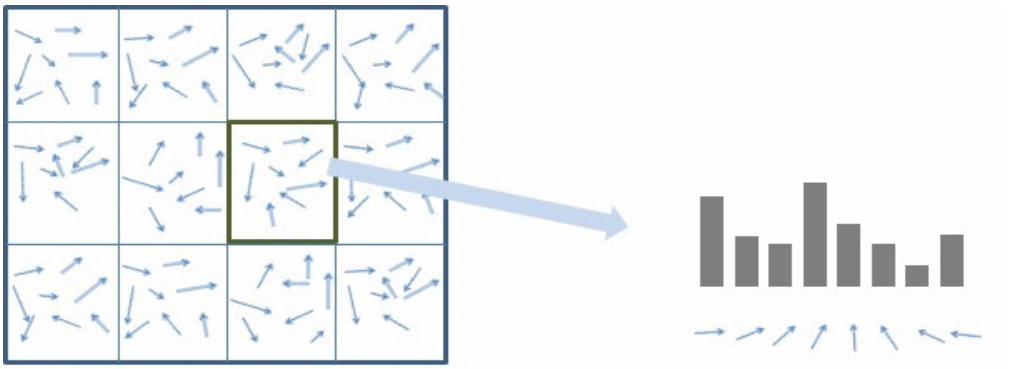

Figure 3.2: Feature descriptor formation *(Source: Internet)*

7. Classification

As a final step in object detection, the obtained feature descriptor is used to train a classifier, typically a Linear Support Vector Machine (SVM). The feature descriptor is obtained for positive and negative images during the training phase. In the classifying phase, the new image's feature descriptor is obtained and classified according to the evaluated SVM result.

3.2.0.2 TrueNorth Implementation

The challenge of implementing the HOG algorithm to the TrueNorth architecture is mapping it to a neural network style. The algorithm has to be transformed in such a way that we can make the input go through a neurosynaptic computer and obtain the resultant feature descriptor as the output. We will refer to the main computation unit as the HoG corelet.

1. High-level Description

As described in 4.1, the implementation of HoG at the cell level can be broken down into the following blocks:

- Computation of the derivatives and gradient
- Computation the gradient angle
- Orientation binning

Even though the orientation binning is done simply by counting the occurences of an angle within each range of the bin, the magnitude is evaluated to perform a comparison of performance. Therefore the HoG corelet is divided into three subcorelets, to perform each of the operations described. In order to drive the input values at regular intervals according to the number of spikes corresponding value, there is an additional corelet that acts like a clock.

The corelet takes in one cell at a time, that is, it takes a 10x10 px input (8x8 with padding in order to apply the derivative). Taking advantage of the parallel nature of Truenorth, the overall design has several such corelets, so that all the cells of a 64x128 image can be sent to a feature-extraction corelet simultaneously.

This is a trade-off between resources and power consumed vs. parallel processing speed-up.

2. Gradient Corelet

The derivative of the image is computed by using the centered 1-D derivative mask. The corelet takes in an input of 10 x 10 pixels, in order to compute the gradient value for the 8 x 8 pixel cell. This corelet is designed to take in the pixel values as the axon values. The output obtained from the neurons are the values of Ix and Iy. Since it is a 10 x 10 input, the number of axons used is 100 and it uses one entire core to generate the values of Ix, -Ix, Iy and -Iy.

3. Gradient Angle Corelet

The angle is calculated using an arc tangent function in the traditional HoG. However, mapping the arc tangent function on to a neural network architecture is not feasible. Therefore, in order to compute the dominant angle, we take different values of theta to calculate the polar form of the Ix Iy values along each theta. Since each value of Ix or Iy has 18 output values for each pixel, and we have 64 pixels in total, the total number of output values we need to get is 18 * 64.

4. Maxpooling Corelet

The orientation found in all directions is sent to the maxpooling corelet. Here, we calculate the maximum angle by performing a max-pool on each of the 18 computed orientations.

Note that TrueNorth is especially efficient for pattern-matching applications. The max-pooling corelet takes care of actually finding out the maximum angle value. It is this value that is used to perform orientation binning. It takes in the values of all the angles computed by the previous corelet. It performs pattern matching of all these angles with the maximum angle found and the output generated tells us which angle was the most prominent for that pixel.

The workflow is as described in Table 3.1 [1]. This way, the HoG computation is mapped on to the neurosynaptic cores of TrueNorth. The output of the core would be the generated histogram bin values of a given cell. We can iteratively compute the histogram values for all the cells and obtain the feature descriptor for the entire image.

Operation	Original Computation	TrueNorth Computation
Gradient vector	Using filters (-1 0 1) & (-1 0 1)' Result: Ix and Iy	Using filters (-1 0 1),(1 0 -1), (-1 0 1)', (1 0 -1)' Result: Ix, -Ix, Iy, -Iy
Gradient Angle	$\theta = tan^{-1}\frac{I_x}{I_y}$	θ for which $(I_x cos\theta + I_y sin\theta)$ is maximum.
Gradient Magnitude	$\sqrt{I_x^2 + I_y^2}$	$(I_x cos\theta + I_y sin\theta)$
Histogram	Binned by magnitude, either choosing 9 bins from 0 to 180 or 18 bins from 0 to 360.	Binned by count, with 18 bins from 0 to 360.

Table 3.1: HoG implementation: Original v/s TrueNorth *Source: [1]*

After the feature descriptor is obtained, they can be used to train a classifier. Results have been obtained for training both an SVM as well as Eedn. After the training phase, a new test image is given to TrueNorth to obtain its feature descriptor. The resultant value can then be given classifier to perform object detection, that is to say whether an object is present or not. Typically, in HOG, a sliding window estimation is performed on the image in order to take care of geometric consistencies. We can perform the same in TrueNorth by sending each image window to the cores to obtain the feature descriptor vector.

5. Histogram

As illustrated in Table 1, the gradient vector is found by performing low precision pattern matching. Furthermore, the expression $(I_x cos\theta + I_y sin\theta)$ is equivalent to the magnitude of the gradient. This form can be efficiently mapped on to the hardware. The gradient angle is the direction in which the gradient is dominant. The angle can, therefore, be found by evaluating the magnitude in different directions, and

find the one in which the magnitude is maximum. The resultant angles are binned in the histogram.

Certain operations such as inner product and pattern matching can be done very efficiently on TrueNorth. Mapping the exact arithmetic primitives of the HoG on TrueNorth will not be making use of the features of TrueNorth optimally. Therefore, the approximate method described above proves to be an efficient way of achieving similar results. The approximate method is validated against a full-precision version as well as an FPGA HoG pipeline. The results of these are shown in the next chapter.

In order to make the comparison more widespread, a software version of the feature extraction method used on TrueNorth was created. This allowed for testing the scope of the method beyond the constraints imposed by TrueNorth's architecture, in order to make the results useful for neuromorphic architectures in general. This software model achieved over 99.5% correlation with the TrueNorth model when it was quantized with the same width as TrueNorth.

Chapter 4
Methodology and Validation

4.1 Introduction

The main contribution of my work is the implementation of the approximate HoG on TrueNorth. The operations of inner product, pattern matching and comparison were mapped on to the neuromorphic hardware. This provided an implementation that was a 1:1 mapping of the original algorithm *(NApprox)*, which could then be compared with other novel implementations such as a network trained to learn the HoG features instead of programming the primitives *(Parrot)*, and a method that skips the pre-processing step altogether to create a raw-image to classifier system *(Absorbed Features/Monolithic Classifier)*.

These methods are briefly described in this section, followed by the overall results obtained.

4.2 Other Implementations of HoG/Object Detection

4.2.1 Learned Features by the Parrot-HoG

In the approximate HoG (referred to as NApprox henceforth), the detailed operations are programmed in order to perform feature extraction. These detailed operations are essentially the arithmetic applied on the raw pixels of the image, in order to transform it to a meaningful representation of its features that can be used to distinguish it. Instead of actually performing the arithmetic, a network can be designed to to behave like the feature extractor. This kind of an implementation is usually termed as a "parrot transformation", as the attempt is to mimic a feature

extractor [11]. A neural network was constructed to output the confidence with which a particular cell belongs to a class (the degree to which the region belongs to a particular orientation), where each class is the orientation to which the cell could belong. This will produce an equivalent feature vector as that of HoG. More details of the Parrot-HoG can be found in [1].

4.2.2 Absorbed Features a.k.a Monolithic Classifier

This implementation is an attempt to evaluate if the pre-processing, feature extraction stage is valuable to the results. A classifier is trained on the raw pixels of an image to perform the detection. The training set used for this classifier is identical to the previous parrot transformation approach.

4.3 Validation of the Approximated Method

As described in Table 3.1, the HoG algorithms was broken down into its primitives, and each primitive was either directly mapped to the TrueNorth version, or underwent some mathematical transformations to better suit the neuromorphic paradigm. This approximated version of HoG is validated in order to prove that it, in fact, works identical to the original algorithm and does, in fact, extract features that are meaningful to perform object detection.

4.3.0.1 Precision-Recall Graphs

A precision-recall graph is typically used in binary classification - in our case, whether there is a pedestrian or not, positive or negative. The graph has two components as its name indicates, precision and recall.

Precision is defined as the fraction of retrieved instances that are relevant. This means that it corresponds to the number of true positives, out of the total number of "positive" predictions made (true and false). Recall, on the other hand, is the fraction of relevant instances that are retrieved. In other words, it is the ration of true positives predicted out of all the actually positive results. Typically, precision and recall are inversely related such that if recall increases, precision decreases.

We use precision-recall graphs in order to depict the performance of the different feature extraction or object detection techniques. In this case, the plot is miss

rate vs. false positives per image. These curves make a good depiction of how the different methods perform against one another.

4.3.0.2 Validation Results

This section covers the results of the validation ofthe approximated method. A support vector machine was built, comparable as that used in [12] for classification of an FPGA-based feature extractor using HoG. Using LibSVM, [13], a linear SVM classifier was trained using the 2,416 positive person images and 12,180 negative images from the INRIA Person Dataset. The classifier was also retrained with false negatives, in order to improve accuracy.

The false negative retraining involves performing one round of training of the classifier, and using the cross-validation results to obtain false negatives. The classifier is then retrained with the negative training images to filter false positives. This information is then augmented to the SVM model as negatives. This improves the prediction when the classifier is used in the test phase.

It is vital to note that the features obtained from TrueNorth are not full precision numbers. Rather, they are the quantized representation transduced from the output spikes. While validating the approximate method, it was necessary to not allow the representation to pull back the results, as the representation is purely platform-specific, while the overall algorithm is a generic approach. For this purpose, the results of pre-processing using NApprox were also obtained in full precision by using the software implementation as described in Chapter 3. The validation was done on both the high precision, as well as low precision versions of NApprox, against the baseline FPGA HoG pipeline outlines in [12].

The descriptions of the three methods used for comparison are provided below. FPGA HoG is a histogram with 9 orientation bins, weighted voting in magnitude with fixed-point computation. NApprox(fp) is the full precision version of the primitives obtained from the neuromorphic HoG where the histogram has 18 orientation bins, the voting is count-based and the computation is floating point. NApprox is the basic, TrueNorth-compatible feature extractor output which is not in full precision, and follow the same implementation details otherwise as NApprox(fp). All three implementations include contrast normalization over blocks of size 2 x 2 cells, and L2Norm is the corresponding L2 normalization performed on the blocks.

Figure 4.1 depicts the precision-recall graphs of the three implementations. This figure shows the quality of the approximated algorithm, in full precision mode as well as the reduced precision mode.

The TrueNorth implementation note the following important details. Firstly, the images used are in greyscale instead of RGB, therefore there is only a single color channel. Each feature-extracting corelet takes in a 10x10 px input, in order to apply a centered 1-d point derivative and obtain the result of an 8x8 cell. The additional pixels are padded around the 8x8 cell in order to be able to apply the derivative completely on one cell. Normally, the histogram weighted voting is done by magnitude of the gradient. Furthermore, as discrete orientation bins do not accurately describe an angle orientation between two neighbouring bins, normally a bilinear interpolation is used to avoid aliasing. [10]. However, the TrueNorth implementation only takes into account the count at each bin.

Each HoG window is provided with a 10x10 cell, resulting in a feature vector of size 7x15x18x4 for a 64x128 image.

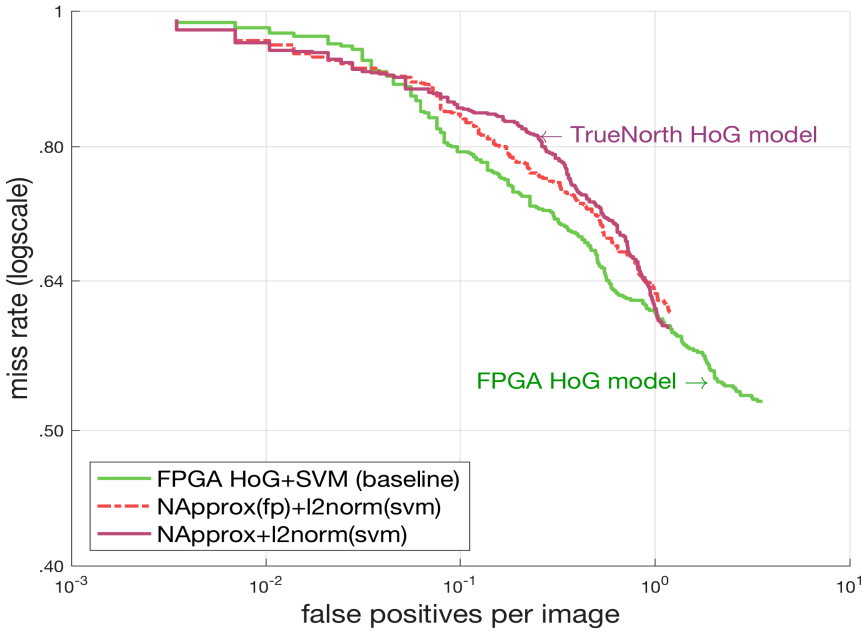

Figure 4.1: Precision-Recall curve: Validation of HoG approximation *Source:* [1]

The miss rate versus false positives acts as a proxy for precision-recall curve as proposed by Dollar et al [14]. The test cases were also obtained from the INRIA

Dataset. The methodology used to perform detection was to input windows of an image of increasing size to the SVM. Each time, the window size is increased by 1.1 times. In a single full HD image, there are many detection windows of varying sizes. Based on the classifier output at each of these windows, an accurate bounding box can be drawn around the detected object (pedestrian). A method called non-maximum suppression [10] is used in order to reduce the number of windows obtained.

Naturally, when these windows are given as input to the classifier, we may get some smaller windows classified as positive. In order to provide a more meaningful reslult, the detection's overlapped region has to be at least half of the ground truth image in order to be classified as a true positive. All other detections are dismissed as false positives.

Chapter 5
Results

5.1 Introduction

This section goes over all the implementations and their results. We begin with an end-to-end cobject recognition system built on TrueNorth. The specifications of the end-to-end are as follows.

The first section is the feature extractor. This is the implementation of the NApprox described in earlier chapters, with several feature extraction corelets created in parallel. Each corelet is capable of processing a 10x10 input, and is made up of 23 cores to perform the primitive operations. For a 64x128 image, there are 128 cells in total. Therefore, 128 such corelets are created in order to process all the cells simultaneously. Following this, the histogram is obtained with all the resultant feature vectors concatenated.

The output of the feature extractor is converted to its floating point quantized format. This data is given as input to the Eedn Classifier. The Eedn network is built with the dataset obtained from the feature extractor module. The first layer is a transduction layer that converts the floating point representations to spikes. The classifier is then trained on these input spikes in many iterations. Figure 4.2 shows how the accuracy changes with each iteration of the training. The accuracy results of the training on Eedn was 92%. This was obtained from a 3-layer Eedn network.

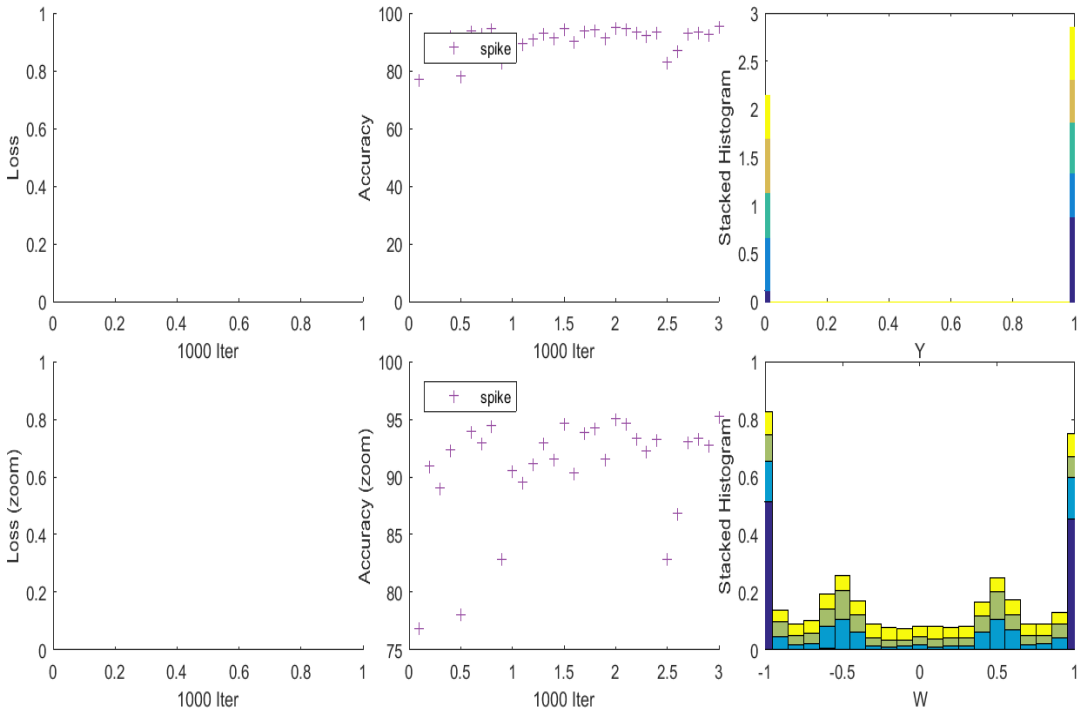

Figure 5.1: Training Eedn

5.2 Other results

The following section describes the overall performance of the three approaches described in Chapter 4. It is important to note that block normalization is not performed on TrueNorth directly. The same trained Eedn network is used for the three approaches. The network has 18 layers, using 2864 cores. For the Parrot-HoG, a 2-layer Eedn was used, which used 8 cores for each cell. This results in a total of 1024 cores for a 64x128 px image. In total, therefore, 3888 cores are used.

The features of the NApprox were used to train the 2864-core Eedn classifier. This is the focus of the approximate implementation. Other implementations such as Parrot and Monolithic were used to train the 2864 and combined (3888) core networks respectively.

The miss rate versus false positives is plotted and depicted in Figure 5.2.

Power Efficiency is evaluated for all the three approaches as well. For the NApprox design, clocks were used in otder to accumulate the weighted sum for multiple ticks, usually 1 ms per tick, to provide orecise inner product results. The

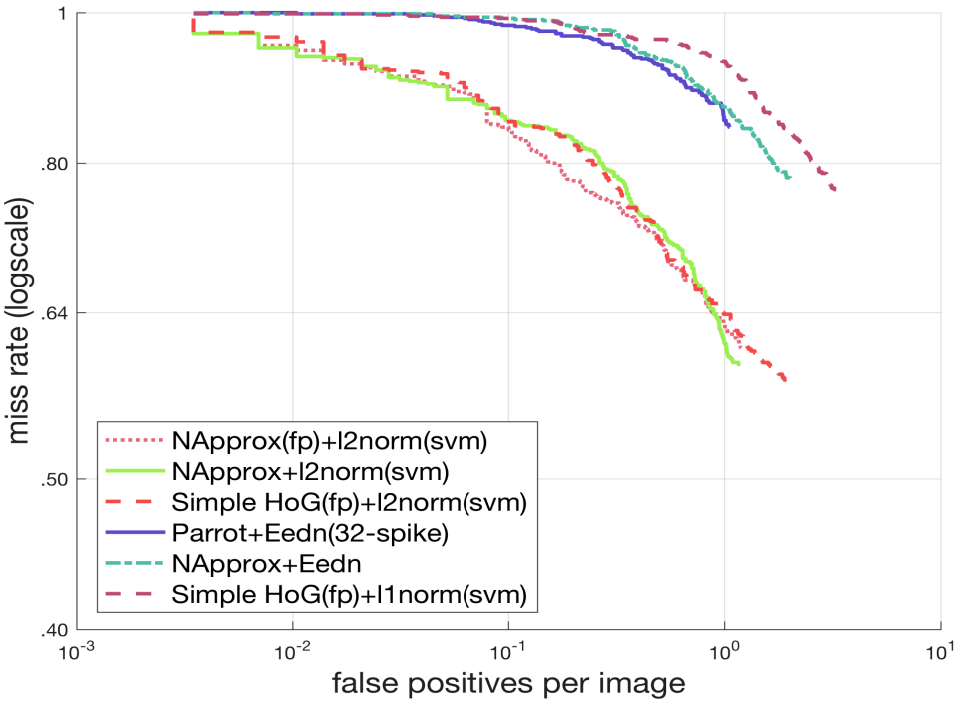

Figure 5.2: Evaluation of Pedestrain Detection Approaches *Source:* [1]

encoding is 64-spike representation of the input image. The throughput a single HoG module produces is 15 cells/sec. Therefore, in the parallel implementation, we process all the cells at 1/15th of a second, or 66.68 ms. The total power consumed by the NApprox is **40 W at 26 fps**.

Chapter 6
Conclusion

6.1 Conclusions

The NApprox implementation provides a good baseline for the implementation of HoG in TrueNorth. It provides reasonable insight into what it takes to perform a direct 1:1 mapping of an algorithm's arithmetic primitives on a neuromorphic platform. This in turn helps determine whether more sophisticated implementations are required to better leverage the characteristics of a neuromorphic hardware to perform pedestrian detection. In that endeavour, two other implementations, namely the Parrot-HOG and the Monolithic Classifier were made. Not only was the implementation vital in demonstrating how arithmetic primitives can be mapped on TrueNorth, it also helps expose the inherent characteristics of a neuromorphic architecture.

The evaluations resulted in a system that consumes 40W in 26 fps. While this may sound like a high number, it is due to the high amount of resources required to achieve that magnitude of frames per second. Furthermore, the 1:1 implementation has several contributions. Firstly, it provides an understanding of the basic implementation of a structured algorithm on a neuromorphic hardware platform. Secondly, it exposes the fact that neural network platforms can also be used to map structured operations. It also gives rise to interesting perspectives on neuromorphic hardwares, to evaluate if certain functionalities can be added to such platforms in order to make them more versatile while still achieving the advantages of its paradigm.

Bibliography

[1] W. Y. Tsai, e. a. (2017) "Co-training of Feature Extraction and Classification using Partitioned Convolutional Neural Networks," *Design Automation Conference*.

[2] Merolla, P. A. e. a. "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, **345**(6197), pp. 668–673.

[3] Chen, Y. e. a. "A machine-learning supercomputer," *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 609–622.

[4] Furber, S. B. e. a. "Overview of the spinnaker system architecture," *IEEE Transactions on Computers*, (12), pp. 2454–2467.

[5] Schemmel, J. e. a. "Live demonstration: A scaled-down version of the brainscales wafer-scale neuromorphic system." *2012 IEEE International Symposium on Circuits and Systems*, p. 702.

[6] Modha, D. "Introducing a Brain-inspired Computer," *http://www.radiolocman.com/review/article.html?di=162687*.

[7] Amir, A. e. a. *The 2013 International Joint Conference on Neural Networks (IJCNN)*.

[8] Akopyan, F. e. a. "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1537–1557.

[9] Esser, S. K. e. a. (2016) "Convolutional networks for fast, energy-efficient neuromorphic computing," *CoRR*, **abs/1603.08270**.

[10] Dallal and Triggs "Histograms of Oriented Gradients for Human Detection," *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, pp. 886–893.

[11] Esmaeilzadeh, H. e. a. (IEEE Computer Society, 2012) "Neural acceleration for general-purpose approximate programs," *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, **47**, pp. 449–460.

[12] Advani, S. K. "Large-Scale Object Recognition for Embedded Wearable Platforms," *PhD Thesis, The Pennsylvania State University, University Park, PA, USA, 2016*.

[13] Chang, C. C. and C. J. Lin (1995) "Libsvm: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, **2**(3), pp. 1–27.

[14] Dollar, P. and P. e. a. Perona "Pedestrian detection: An evaluation of the state of the art," *IEEE Trans. Pattern Anal. Mach. Intell.*, (4), pp. 743–761.