# AxNN: Energy-Efficient Neuromorphic Systems using Approximate Computing *

Swagath Venkataramani, Ashish Ranjan, Kaushik Roy and Anand Raghunathan
School of Electrical and Computer Engineering, Purdue University
{venkata0,aranjan,kaushik,raghunathan}@purdue.edu

## ABSTRACT

Neuromorphic algorithms, which are comprised of highly complex, large-scale networks of artificial neurons, are increasingly used for a variety of recognition, classification, search and vision tasks. However, their computational and energy requirements can be quite high, and hence their energy-efficient implementation is of great interest.

We propose a new approach to design energy-efficient hardware implementations of large-scale neural networks (NNs) using *approximate computing*. Our work is motivated by the observations that (i) NNs are used in applications where less-than-perfect results are acceptable, and often inevitable, and (ii) they are highly resilient to inexactness in many (but not all) of their constituent computations. We make two key contributions. First, we propose a method to transform any given NN into an Approximate Neural Network (AxNN). This is performed by (i) adapting the backpropagation technique, which is commonly used to train these networks, to quantify the impact of approximating each neuron to the overall network quality (*e.g.*, classification accuracy), and (ii) selectively approximating those neurons that impact network quality the least. Further, we make the key observation that training is a naturally error-healing process that can be used to mitigate the impact of approximations to neurons. Therefore, we incrementally retrain the network with the approximations in-place, reclaiming a significant portion of the quality ceded by approximations. As a second contribution, we propose a programmable and quality-configurable neuromorphic processing engine (QCNPE), which utilizes arrays of specialized processing elements that execute neuron computations with dynamically configurable accuracies and can be used to execute AxNNs from diverse applications. We evaluated the proposed approach by constructing AXNNs for 6 recognition applications (ranging in complexity from 12-47,818 neurons and 160-3,155,968 connections) and executing them on two different platforms – QCNPE implementation containing 272 processing elements in 45nm technology and a commodity Intel Xeon server. Our results demonstrate 1.14X-1.92X energy benefits for virtually no loss ($< 0.5\%$) in output quality, and even higher improvements (upto 2.3X) when some loss (upto 7.5%) in output quality is acceptable.

## Categories and Subject Descriptors

B.7.1 [**INTEGRATED CIRCUITS**]: VLSI (Very large scale integration)

## Keywords

Neuromorphic Systems; Large-scale Neural Networks; Approximate Computing; Energy Efficiency

*This work was supported in part by the National Science Foundation under grant nos. 1018621 and 1320808.


*ISLPED'14,* August 11–13, 2014, La Jolla, CA, USA.

http://dx.doi.org/10.1145/2627369.2627613.

## 1. INTRODUCTION

The field of neuromorphic computing has garnered significant interest in the past decade due to a confluence of trends from neuroscience, machine learning, semiconductor technology, and high performance computing. An important development within this field has been the advent of large-scale neural networks (NNs) such as Deep Learning Networks [1–3] (DLNs), Hierarchical Temporal Memory [4] (HTM), *etc.* These biologically inspired algorithms have shown state-of-the-art results on a variety of recognition, classification and inference tasks. Hence, they are deployed in many real world applications such as Google image search [5], Google Now speech recognition [6], and Apple Siri voice recognition [7], among others. However, they are also highly computationally intensive due to their large scale and dense connectivity. For example, SuperVision [2], a DLN that recently won the Imagenet visual recognition challenge, contains 650,000 neurons and 60 million connections, and demands compute performance in the order of 2-4 GOPS per classification. With energy efficiency becoming a primary concern across the computing spectrum from data centers to mobile devices, the energy-efficient realization of large-scale neural networks is of great importance.

In this work, we propose the use of *approximate computing* for energy-efficient implementation of neuromorphic systems. Approximate computing [8–14] is an emerging design paradigm that leverages the intrinsic resilience of applications, *i.e.*, their ability to produce results of acceptable quality even when many of their computations are performed in an approximate manner. This ability to relax the accuracy of computations is translated into significant improvements in energy or performance by utilizing a variety of hardware [9–11, 13, 14] and software [8, 12] techniques.

A key question in approximate computing is which computations to approximate, and by how much. The judicious selection of approximations is critical to maximizing the benefits from approximate computing while ensuring minimal degradation in output quality. For this purpose, it is necessary to determine the impact of approximating various internal computations on the eventual application output quality. In the context of neuromorphic systems, we address this challenge by leveraging backpropagation, an operation that is widely utilized for NN training. We observe that backpropagation provides a measure of the sensitivity of the NN outputs to each neuron in the network; thereby, it can be utilized to identify neurons that are likely to be more resilient to approximations.

The process of training provides a further opportunity to maximize the benefits of approximate computing in NNs. Training is an inherently error-healing process, since it modulates the weights associated with each neuron in the NN such that the error at the network outputs is minimized. Therefore, we suggest that training can also be used to compensate for approximations. Further, this synergy can be exploited in an iterative approximate-and-retrain loop to enhance the benefits of approximate computing.

Based on the above insights, we propose a method to construct ApproXimate Neural Networks (AxNNs) that consists of three key steps. First, it utilizes backpropagation to *characterize* the importance of each neuron in the NN and identify those that impact output quality the least. Next, the AxNN is created by selectively replacing less significant neurons in the network with *approximate* versions that are more energy-efficient. Towards this end, we utilize *precision scal-*

*ing*, a popular approximate design technique, and modulate the precisions of the inputs and the weights of the neurons to realize versions with different accuracy *vs.* energy trade-offs. Once the approximate NN is formed, we adapt the weights of the neurons in the approximated network by *incrementally retraining* them. Since training is a naturally error-healing process, this allows us to reclaim a significant portion of the quality ceded by approximations. For a given output quality, retraining may create further opportunities to approximate the NN, resulting in increased energy benefits. We develop an automatic design methodology to generate AxNNs by iterating the aforementioned characterize, approximate and retrain steps in a quality-constrained loop.

Another contribution of our work is the design of a quality-configurable Neuromorphic Processing Engine (QCNPE), which provides a programmable hardware platform for efficiently executing AxNNs with arbitrary topologies, weights, and degrees of approximation. QCNPE features a 2D array of Neural Computation Units (NCUs) and a 1D array of Activation Function Units (AFUs) that together enable the efficient execution of neural networks. We equip the NCUs and AFUs with hardware mechanisms based on precision scaling to effectively translate the reduced precision of neurons into energy benefits at run-time.

In summary, the key contributions of this work are:

- We propose a new avenue for energy efficiency in neuromorphic systems by using approximate computing. We propose the concept of ApproXimate Neural Networks (AxNNs) that leverage backpropagation to maximize the energy benefits from approximate computing, while utilizing the inherent healing nature of the training process to minimize their impact on output quality.
- Embodying the above design principle, we develop a systematic methodology, which can automatically generate AxNNs for any given neural network. The methodology is independent of the NN topology, network parameters and the training dataset.
- We design a programmable and quality-configurable Neuromorphic Processing Engine (QCNPE) that can be used to efficiently execute AxNNs.
- We construct approximate versions of 6 popular large-scale NN applications using the proposed AxNN design methodology and execute them on two different platforms – QCNPE and commodity Intel Xeon server – to demonstrate significant improvements in energy for negligible loss in output quality.

The rest of the paper is organized as follows. Section 2 presents an overview of related efforts. Section 3 provides relevant background on NNs. Section 4 outlines the proposed AxNN design methodology. Section 5 details the architecture of QCNPE. Section 6 describes the experimental methodology and the NN applications used. Section 7 presents the results and Section 8 concludes the paper.

## 2. RELATED WORK

Our work is at the confluence of two important (and hitherto disconnected) areas of research—efficient realization of neuromorphic systems, and approximate computing. This section presents an overview of research efforts in each of these areas and places our contributions in their context.

**Efficient neuromorphic systems:** Previous efforts have explored two major directions for the efficient implementation of NNs. The first is accelerator based computing, in which custom architectures that are optimized to the computation and communication patterns of NNs are designed. A spectrum of architectures ranging from application-specific NN designs [15] to programmable neural processors [16, 17] have been proposed. Further, NN implementations on other programmable accelerators such as GPUs [18] have also been explored. The second prominent is the use of emerging device technologies such as resistive RAM [19], memristor based crossbar arrays [20], and spintronics [21], to realize neurons and synapses more efficiently.

In this work, we identify a new dimension to optimize neuromorphic systems. We leverage their intrinsic resilience and employ approximate computing to construct NN implementations that are significantly more efficient. Note that the approximate computing techniques presented in this work are *complementary* to the above efforts, *i.e.,* they can be employed together to further enhance efficiency.

**Approximate computing:** There has been growing interest in the field of approximate computing, leading to efforts at various levels of design abstraction, spanning software, architecture and circuits. At the software level, these techniques improve performance by either skipping computations [8], relaxing dependencies [8], or replacing expensive functions with approximate versions [12]. In hardware, the use of voltage over-scaling and circuit simplification have been explored to gain energy efficiency with some (acceptable) loss in output quality [9, 10, 22]. While these initial efforts targeted application-specific hardware, approximate computing has recently been extended to the domain of programmable processors [13, 14]. Finally, with the growing interest in approximate hardware platforms, supporting design automation techniques have also been explored [11].

We extend the state-of-the-art in approximate computing along two key fronts. First, we apply approximate computing to NNs, an important class of applications that demonstrate significant intrinsic resilience. Second, we show how the unique properties of NNs can be exploited to maximize the benefits of approximate computing. The use of backpropagation to systematically identify less significant neurons in NNs leads to a superior energy-quality trade-off. Further, interleaved approximation and training extends the degree to which approximate computing can be utilized in NNs.

## 3. NEURAL NETS: PRELIMINARIES

Neural networks can be broadly described as systems that functionally abstract the computational behavior of the human brain. The fundamental computation unit of NNs is called a *neuron*, which is densely interconnected with several others to constitute a neural network. Each neuron in the network, as shown in Figure 1(a), computes a weighted sum of all its inputs, followed by a non-linear activation function on the weighted sum to produce the output.

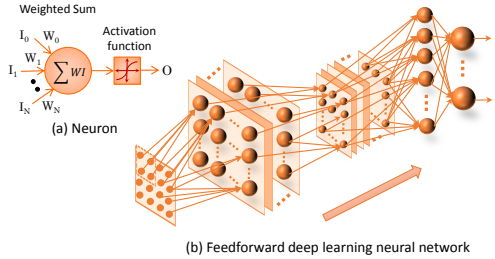


**Figure 1: Neural network preliminaries**

While the proposed approach can be applied to various classes of NNs, in our discussions we consider the most prevalent form, *viz. feedforward NNs*, wherein the neurons are connected to form an acyclic network, as illustrated in Figure 1(b). The operation of NNs typically involves 2 phases *viz.* training and testing. In the training phase, the parameters of the NN (weights of each neuron) are identified based on the training dataset. Once the NN is trained, it enters the testing or evaluation phase, in which it is used to perform the desired application. A brief description of the steps involved in testing and training are provided below.

**Evaluating NNs—Forward Propagation:** Forward propagation, used widely in both testing and training, is the process of evaluating the outputs of the NN. In forward propagation, the inputs are fed to the neurons in the first layer, where they are processed and propagated to the neurons in the next layer. This process is repeated at all the network layers and the NN outputs are eventually computed.

**Training NNs—Backpropagation:** The training process iterates over a dataset of training instances, pre-labeled with golden outputs for the NN, to identify the values of network

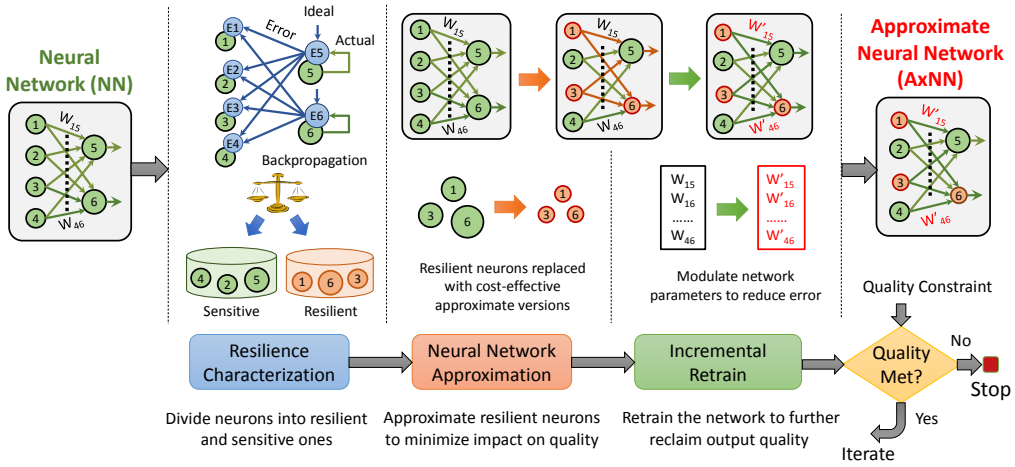


**Figure 2: Overview of the Approximate Neural Networks (AxNN) design approach**

parameters that maximize the application output quality. The network parameters are typically initialized randomly and are successively refined in each iteration as described below. First, the NN is evaluated for a random training instance using forward propagation, and the error at the network output (with reference to the golden output) is computed. Next, a key step called *backpropagation* is invoked, which redistributes the error at the NN output backward in the network, all the way to its inputs. Thus, backpropagation quantifies the error contributed by each neuron in the network towards the global network error. Knowing the respective error contributions, the network parameters associated with each neuron are modulated such that the error at its output is reduced. Mathematically, the parameter update process is formulated as a gradient descent optimization problem as shown in Equation 1. In this equation, $w_{ji}$ represents the weight of the connection between neuron $i$ and $j$, $E$ denotes the global error, $\alpha$ denotes the learning rate, and $\psi'$ is the first derivative of the activation function. The $\Delta w_{ji}$ is computed by propagating the error back in the network through all the connections in the downstream of $j$ to the output.

$$\Delta w_{ji} = -\alpha \frac{\partial E}{\partial w_{ji}} = \sum_{k \in DownStream(j)} \left( \alpha \psi' w_{kj} \frac{\partial E}{\partial w_{kj}} \right) \quad (1)$$

In the proposed methodology to construct AxNNs, we utilize two unique properties: (i) the ability to apportion global errors to local computations by using backpropagation, and (ii) the ability to self-heal local errors in the network during training. A detailed description of the principles behind AxNNs and their design are provided in Section 4.

# 4. AxNN: APPROACH AND DESIGN METHODOLOGY

Approximate Neural Networks (AxNNs) are neural networks whose constituent computations have been subject to approximations, resulting in improved energy efficiency with acceptable output quality. This section outlines the key ideas behind AxNNs and the proposed design methodology.

## 4.1 AxNN: Design Approach

An AxNN can be viewed as a transformed version of a trained NN, where the transformation introduces approximations such that the resulting energy is minimized while the output quality meets a specified constraint. As shown in Figure 2, this transformation involves three key steps: (i) Resilience characterization, wherein the neural network is analyzed to identify neurons that impact output quality the least, (ii) Neural network approximation, in which the neurons that were determined to be resilient in the characterization step are approximated, and finally (iii) Incremental retraining wherein the network is retrained with the approximations in-place such that the loss in quality is further minimized. The following subsections provide an in-depth description of each step in the process.

### 4.1.1 Neural network resilience characterization

A significant challenge to employing approximate computing in any application is to distinguish computations that the application output is highly sensitive to (and hence cannot be approximated) from resilient ones that may be subject to approximations. In the context of neuromorphic systems, we propose to utilize backpropagation to characterize the resilience of each neuron. Backpropagation apportions the error at the output of the NN to the outputs of individual neurons. Thereby, it provides a measure of the error contributed by each neuron to the outputs of the network. We make the following key observation: neurons that contribute the least to the global error are more resilient *i.e.*, more amenable to approximations. Conversely, neurons contributing the highest error during backpropagation are deemed sensitive.

Based on the above insight, we propose a resilience characterization procedure that involves the following operations. For each instance in the training dataset, the error at the output of the neural network is computed using forward propagation. Next, the errors are propagated back to the outputs of individual neurons and their average error contribution over all inputs in the training set is obtained. The neurons are then sorted based on the magnitude of their average error contribution, and a pre-determined threshold is used to classify them as resilient or sensitive. We note that, unlike the actual training process, the network parameters are not altered during the resilience characterization step.

### 4.1.2 Approximation of resilient neurons

In the approximation step, the AxNN is formed by replacing *approximate neurons* in place of the resilient neurons identified during resilience characterization. Approximate neurons are inaccurate but cost-effective hardware or software implementations of the original neuron functionality and are the primary source of the energy efficiency in AxNNs. Approximate neurons can be designed using a wide range of approximate computing techniques. In this work, we utilize precision scaling, a popular technique in which the precisions (bit-widths) of the input operands and the neuron weights are modulated based on their degree of resilience. In addition, we also explore the use of piecewise-linear approximations of the activation function. These approximations may lead to improved efficiency on various hardware platforms. However, the proposed QCNPE architecture, described in Section 5, is specifically designed to translate the reduced precision requirements of the approximate neurons into energy improvements.

### 4.1.3 Incremental retrain of AxNN

Although approximations are themselves introduced in a quality-aware manner, we show how to further minimize their

impact by leveraging the training process. As discussed in Section 3, the training process modulates the parameters associated with each neuron such that the global error is minimized. In fully-accurate NNs, the output error originates from untrained or partially trained network parameters. However, in the case of AxNNs, we intentionally supplement this error with a secondary source, *viz.* approximate neurons. Since training by nature has the ability to minimize errors at neuron outputs, we assert that errors introduced by approximations can also potentially benefit from it. Leveraging this insight, we propose to retrain the AxNN parameters with approximations in-place. The retraining process, as shown in Figure 2, suitably adjusts the AxNN parameters, thereby alleviating the impact of approximation-induced errors. Since retraining improves the output quality of the AxNN, it enables new opportunities to perform additional approximations. This synergy between approximation and training can be captured in an iterative approximate-and-retrain loop, as described in the next sub-section.

Retraining the AxNN after approximations increases the overall runtime of the training process. However, we note that the retraining is incremental *i.e.*, it is carried out for very few iterations (2 iterations in our experiments). Typically, the training process in NNs takes several tens to hundreds of iterations and therefore the increase in run-time complexity due to retraining is small. Also, in typical use cases of NN applications, the training process is performed once or very infrequently. On the other hand, the testing or evaluation phase, in which the actual classification is performed using the NN, extends for much longer periods of time. Since AxNNs yield significant energy benefits in the more critical evaluation phase, a small increase in the cost of training is a favorable trade-off. The impact of retraining on the overall energy and quality of AxNNs for different applications is discussed in Section 7.4.

## 4.2 AxNN Design Methodology

Algorithm 1 describes the pseudocode of the systematic methodology that we propose to automatically construct AxNNs. The inputs are a pre-trained neural network ($NN$), its corresponding training dataset ($TrData$) and a quality constraint ($Q$) that dictates the degradation in quality tolerable in the approximate implementation. The quality specifications are application-specific and are typically used during the process of constructing and training the NN itself. The algorithm iteratively builds the AxNN by successively approximating the NN in each iteration (lines 3-15), while ensuring that the quality bounds are satisfied.

**Algorithm 1** AxNN: Design methodology

**Input:** Pre-trained neural network: $NN$,
Training dataset: $TrData$, Quality constraint: $Q$
**Output:** Approximate neural network: $AxNN$
1: **Begin**
2: Initialize: $AxNN_{temp} = NN$
3: **while** $Q_{AxNN} > Q$ **do**
4: $AxNN = AxNN_{temp}$
5: $Layer.E_{List} \leftarrow$ Energy estimates of $AxNN$ layers
6: $Layer_{Emax} \leftarrow$ max $(Layer.E_{List})$
7: $Layer_{Emax}.\Delta =$ **backpropagation** $(AxNN, TrData)$
8: $\Delta_{mean} =$ **mean** $(Layer_{Emax}.\Delta)$
9: **for** each $N$: Neuron $\in Layer_{Emax}$ **do**
10: **if** $Layer_{Emax}.\Delta(N) < \alpha * \Delta_{mean}$ **then**
11: $AxNN_{temp} =$ Approximate $N$ in $AxNN$
12: **end if**
13: **end for**
14: $AxNN_{temp} =$ train $(AxNN_{temp}, TrData, K$ epochs$)$
15: **end while**
16: **return** $AxNN$
17: **End**

The following steps are performed in each iteration of Algorithm 1. First, an estimate of energy consumed by each layer of the NN ($Layer.E_{List}$) is computed (line 5). For this purpose, we employ a high-level energy model of the quality-configurable neuromorphic processing engine discussed in Section 5. However, other energy models based on the complexity of neurons and the density of interconnections can also be utilized. We thus identify the most energy-intensive layer ($Layer_{Emax}$) in the network (line 6) and target its constituent neurons for approximations. Next, the resilience of each neuron in $Layer_{Emax}$ is characterized by finding the average error at its output over the entire training set using backpropagation (line 7). We then compute the mean of these errors ($\Delta_{mean}$) and neurons whose error is below a threshold $\alpha * \Delta_{mean}$ are deemed resilient. Each of the resilient neurons previously identified are approximated in steps by gradually reducing their precision of computations (lines 9-13). The approximate neural network ($AxNN$) is thus obtained. Next, the $AxNN$ is incrementally retrained for a small number of iterations ($K$ epochs) to further improve its quality (line 14). After retraining, if the $AxNN$ meets the specified quality constraint, then lines 4-14 are repeated and the network is further approximated. If not, the last valid $AxNN$ is produced as the output.

The above design methodology can be utilized to construct energy-efficient approximate versions of any neural network, subject to the desired quality requirements. Furthermore, any approximation technique may be used, although approximations that result in better energy *vs.* quality tradeoffs are clearly desirable.

# 5. QUALITY CONFIGURABLE NEUROMORPHIC PROCESSING ENGINE

In this section, we describe the proposed quality configurable Neuromorphic Processing Engine (QCNPE) that provides a hardware platform to execute AxNNs. The QCNPE is a many-core architecture that exploits the fine-grained data parallelism and data re-use patterns of NNs. A key feature of QCNPE is that it contains specialized processing elements whose accuracies (and energy) are dynamically configurable and hence can be used to efficiently execute neurons with various degrees of approximation.

Figure 3 shows the block diagram of QCNPE. It contains 2 types of processing elements: (i) a 2D array of neural compute units (NCUs), and (ii) a 1D array of activation function units (AFUs). The NCUs contain a 2-level datapath with an accumulator register and compute the weighted sum of a stream of inputs over multiple cycles. The NCUs are connected to their nearest neighbors in the 2D array and receive inputs from the left and top NCUs, which are then propagated to their right and bottom neighbors in the next cycle. The NCUs along the top and left borders of the 2D array receive inputs from two 1D arrays of First-In-First-Out (FIFO) memory elements placed along the borders. Functionally, the NCUs are designed to perform the weighted sum operation associated with each neuron. For this purpose, the inputs are streamed in along the rows and weights along the columns and operated upon within each NCU. Note that the inputs and weights are re-used by all NCUs in a given row and column respectively, which is a typical data flow pattern in NNs, wherein the inputs fan-out to several neurons and the weights are shared amongst neurons/across inputs.

In order to facilitate execution with different accuracies, the NCUs are designed with a *precision control register*, which is initialized at the beginning of the 2D array operation. This is used to modulate the precision of the NCU inputs before they are operated within the NCU. Scaling the precision of input operands naturally results in power savings due to the reduction in switching activity in the NCU. In QCNPE, this is further enhanced by clock gating the LSB bit slices of the NCU accumulator register. In our implementation, the area and power overheads to enable quality configurability amounted to less than 5% of the overall NCU.

The activation function units (AFUs), located on the right border of the 2D array, are designed to perform the non-linear operation on the weighted sum computed in the NCUs. As

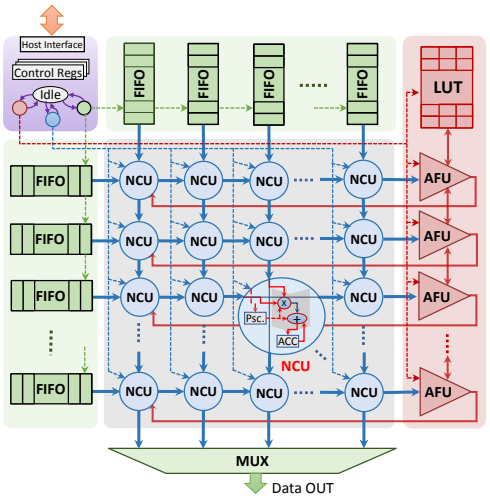


**Figure 3: Block diagram of QCNPE**

shown in Figure 3, this is carried out in a cyclical fashion, wherein the weighted sums from the NCUs in each row are streamed out and the outputs of the AFUs are stored back to the respective elements.

In summary, the QCNPE architecture provides an energy-efficient hardware platform to execute AxNNs of any given topology, interconnectivity pattern and degrees of approximation in their neurons.

# 6. EXPERIMENTAL METHODOLOGY

This section describes the experimental methodology and the benchmarks used in our evaluation of AxNNs. The QCNPE was implemented at the Register-Transfer Level (RTL) in Verilog HDL and mapped to the IBM 45nm technology using Synopsys Design Compiler. Synopsys Power Compiler was used to estimate the energy consumption of the implementation. The key micro-architectural parameters and implementation metrics are shown in Figure 4.

| Micro-architectural Parameters | Value |
|---|---|
| Array Dimension | 16 X 16 |
| No. of NCU/AFU | 256/16 |
| FIFO count | 32 |
| FIFO depth | 32 |

| Metric | Value |
|---|---|
| Feature Size | 45nm |
| Area | 1.7 $mm^2$ |
| Power | 517.2 mW |
| Gate Count | 390392 |
| Frequency | 1GHz |

**Figure 4: QCNPE parameters and metrics**

Neural networks used in 6 popular classification and recognition applications, listed in Figure 5, were used as benchmarks in our experiments. The number of layers, neurons and connections in the networks are also provided in Figure 5. The benchmarks were ported manually to QCNPE and the baseline was well optimized for energy. We utilized classification accuracy, *i.e.,* the fraction of instances correctly classified as the measure of quality for all the benchmarks.

| Applications | Dataset | Layers | Neurons | Connections |
|---|---|---|---|---|
| House Number Recognition | SVHN | 8 | 47818 | 799616 |
| Object Classification | CIFAR | 6 | 38282 | 808608 |
| Digit Recognition | MNIST | 6 | 8010 | 43036 |
| Face Detection | YUV faces | 4 | 13362 | 25552 |
| Object Recognition MLP | CIFAR | 2 | 1034 | 3155968 |
| Census Data Analysis | Adult | 2 | 12 | 160 |

**Figure 5: NN benchmarks**

# 7. RESULTS

In this section, we present the results of experiments that demonstrate the energy efficiency offered by AxNNs.

## 7.1 Energy benefits of AxNN

Figure 6 shows the energy improvement obtained using AxNNs for various output quality (classification accuracy) constraints. The energy of each AxNN is normalized to a fully-accurate QCNPE implementation in which none of the neurons are approximated. Note that this is already a highly optimized baseline since the QCNPE architecture is highly customized to the characteristics of NNs. Across all benchmarks, AxNN consistently provides significant energy benefits between 1.14X-1.92X for virtually no loss ($< 0.5\%$) in application output quality. When the quality constraints are relaxed to $< 2.5\%$ and $< 7.5\%$, the benefits increase to 1.35X-1.95X and 1.41X-2.3X, respectively. On an average, AxNN achieves 1.43X, 1.58X and 1.75X improvement in application energy for the different quality constraints.

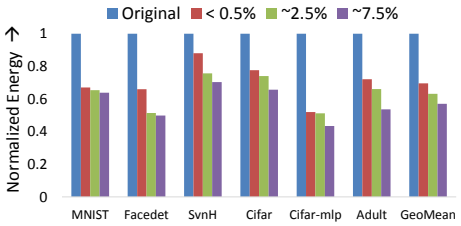


**Figure 6: Improvement in energy using AxNN**

## 7.2 Uniform approximation: Comparison

We now illustrate the effectiveness of the proposed resilience characterization methodology for NNs, by comparing it with a naïve approach wherein all the neurons in the NN are approximated uniformly. Figure 7 shows the energy *vs.* accuracy trade-off curves thus obtained for 3 different applications. We observe in all three cases that the energy improvement obtained using AxNNs is substantially better at all quality levels, compared to uniform approximation. Thus, it is critical to identify neurons that are amenable to approximations and directly applying approximate computing techniques without the proposed resilience characterization step would lead to limited benefits.

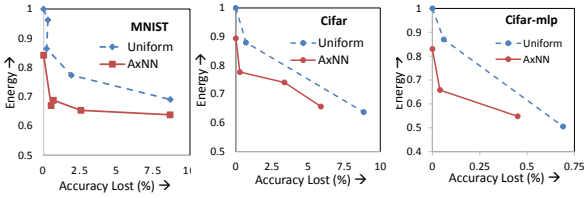


**Figure 7: Quality *vs.* energy trade-offs with uniform and AxNN approximations**

## 7.3 Resilience Characterization: Insights

We present insights into the process of identifying resilient neurons in NNs and illustrate them using the digit recognition application (MNIST) [3] as an example. The NN takes a pixel map of a handwritten digit as its input and classifies it amongst digits 0,1 ...9. The network contains 6 layers and progressively extracts feature maps from the input image in the first four layers and combines them in layers 5 and 6 to infer the class of the input. Each pixel in each feature map of each layer corresponds to the output of a neuron.

Figure 8 shows the average errors (obtained using back-propagation) at the outputs of all neurons in four selected layers of the digit recognition network. The neurons are color-coded (blue to red) based on the magnitude of their errors and are located on the feature maps corresponding to the pixel they generate. We observe that the resilience of the neurons varies widely (6 orders of magnitude) across all layers and to a substantial extent (4 orders of magnitude) within a given layer. We also find that the fraction of neurons that are resilient decreases sharply as we move closer to the NN outputs. This is attributed to the fact that neurons in the initial layers typically process features local to a certain region of the image, while neurons in the final layers infer global features from the previously extracted local features. Since errors in global inferences are less tolerable, the neurons in the final layers are correspondingly more sensitive. Further, errors in neurons closer to the inputs have a greater chance of being compensated or filtered-out as they propagate through the NN.

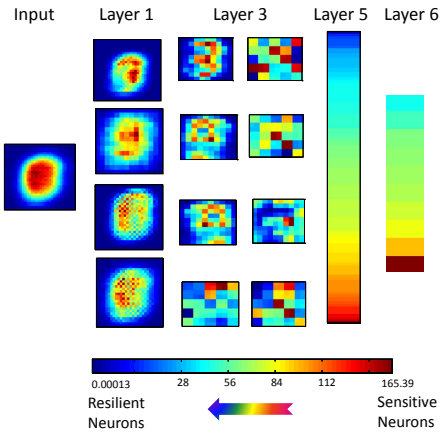


**Figure 8: Neuron average error maps in MNIST [3]**

We also observe a significant correlation between the resilience of neurons and the region of the image on which they operate. For example, in layer 1 of Figure 8, neurons that process the center of the input image, where information is typically concentrated, are less resilient. The neurons become progressively more resilient when proceeding towards the borders of the image. Thus, the resilience characterization methodology utilized in AxNNs captures the physical intuitions behind the resilience of neurons in NNs.

## 7.4 Impact of Retraining

To understand the benefits of incrementally retraining the network with the approximations in place, Figure 9 plots the normalized energy-quality trade-off obtained with and without the retraining step in the AxNN methodology for four applications. We observe in all four cases that AxNN with retraining provides a superior trade-off, *i.e.*, lower energy for a given target quality. This is because, retraining recovers a good amount of quality lost due to approximations, thereby allowing additional approximations for the same quality.

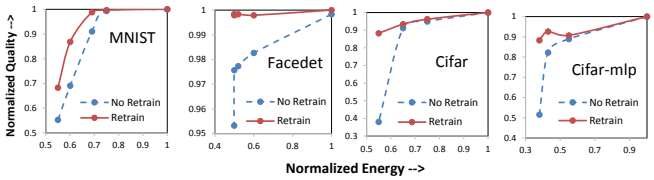


**Figure 9: Impact of retraining on energy and quality**

Across all our benchmarks, retraining the AxNN increased the run time of the training process on an average by 21.5%. As discussed in Section 4.1.3, we believe that this moderate increase in the training time is quite insignificant relative to the energy benefits provided during the energy-critical testing phase of the application.

## 7.5 AxNNs on Commodity Platforms

In the previous subsections, the neurons were approximated by scaling the precision of their input operands and the energy benefits were evaluated using the proposed QC-NPE architecture. We now evaluate the benefits of AxNNs on commodity platforms by designing approximate software implementations of NNs. Towards this end, we replace the activation functions of selected neurons in the network, identified by the AxNN methodology, with an approximate but significantly faster piecewise linear function. The original and approximate implementations were executed on a server with an Intel Xeon processor at 2.7 GHz and 132 GB memory. We note that the software baseline implementation was aggressively optimized for performance.

Figure 10 shows the normalized runtime and quality of the software AxNN implementations, with varying fraction of neurons approximated, for three applications. The graphs reveal that, as the fraction of neurons approximated by the AxNN methodology increases, the runtime decreases proportionally. However, the corresponding decrease in the application output quality is disproportionately small due to the careful selection of neurons and re-training. On an average, the runtime speedup is 1.35X with $< 0.5\%$ loss in the output quality. These results underscore the generality of the AxNN methodology with respect to both the approximate computing technique employed to create approximate neurons, as well as the hardware platform used for their execution.

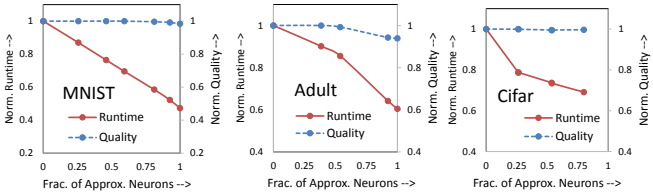


**Figure 10: AxNN runtime on commodity platform**

## 8. CONCLUSION

Neuromorphic systems are growing increasingly prevalent and are popularly employed in a wide variety of classification, recognition, search and computer vision applications. In this work, we utilize approximate computing, an emerging design paradigm, to design energy-efficient neuromorphic systems. We propose the concept of Approximate Neural Networks (AxNNs), in which neurons that impact output quality the least are systematically identified and approximated. The AxNN is then retrained with the approximations in place, leading to additional opportunities to further approximate the network. Also, we design a quality configurable neuromorphic processing engine that can be utilized to efficiently execute AxNNs. Our experiments on six NN applications demonstrated significant improvements in energy for negligible loss in the output quality.

## 9. REFERENCES


[1] K. Kavukcuoglu et. al. Learning convolutional feature hierarchies for visual recognition. In *NIPS*, 2010.

[2] A. Krizhevsky et. al. Imagenet classification with deep convolutional neural networks. In *NIPS, 2012*.

[3] Y. Lecun et. al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), Nov 1998.

[4] J. Hawkins et. al. Hierarchical temporal memory: Concepts, theory, and terminology. Numenta Inc. Whitepaper, 2006.

[5] George Rosenberg. Improving photo search: A step across the semantic gap. June 2009.

[6] Jeffrey Dean et. al. Large scale distributed deep networks. In *NIPS*, 2012.

[7] Scientists See Promise in Deep-Learning Programs, www.nytimes.com /2012/11/24/science/scientists-see-advances-in-deep-learning-a-part-of-artificial-intelligence.html.

[8] S. Chakradhar and A. Raghunathan. Best-effort computing: Re-thinking parallel software and hardware. In *Proc. DAC '10*.

[9] V. K. Chippa et. al. Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency. In *Proc. DAC '10*.

[10] R. Hegde et. al. Energy-efficient signal processing via algorithmic noise-tolerance. In *Proc. ISLPED '99*, pages 30–35.

[11] S. Venkataramani et. al. SALSA: systematic logic synthesis of approximate circuits. In *Proc. DAC '12*.

[12] H. Esmaeilzadeh et. al. Neural acceleration for general-purpose approximate programs. In *MICRO, 2012*.

[13] H. Esmaeilzadeh et. al. Architecture support for disciplined approximate programming. In *Proc. ASPLOS 2012*.

[14] S. Venkataramani et al. Quality programmable vector processors for approximate computing. In *Proc. MICRO*, 2013.

[15] C. Farabet et al. Neuflow: A runtime reconfigurable dataflow processor for vision. In *Proc. CVPRW, 2011*.

[16] S. Chakradhar et. al. A dynamically configurable coprocessor for convolutional neural networks. In *Proc. ISCA '10*.

[17] E. Painkras et. al. Spinnaker: A multi-core system-on-chip for massively-parallel neural net simulation. In *Proc. CICC '12*.

[18] J. Ngiam et. al. On optimization methods for deep learning. In *Proc. ICML*, pages 265–272, 2011.

[19] B. Rajendran et. al. Specifications of nanoscale devices and circuits for neuromorphic computational systems. *IEEE Trans. on Electron Devices*, 60(1):246–253, 2013.

[20] Sung Hyun Jo et. al. Nanoscale memristor device as synapse in neuromorphic systems. *Nano Letters '10*.

[21] K. Roy et al. Beyond charge-based computation: Boolean and non-Boolean computing with spin torque devices. In *Proc. ISLPED*, pages 139–142, Sep. 2013.

[22] K. Palem et. al. Sustaining moore's law in embedded computing through probabilistic and approximate design: Retrospects and prospects. In *Proc. CASES*, pages 1–10, 2009.