# High Speed Energy-Efficient Convolutional Neural Network with Approximate Computing

Keertana Settaluri
School of Electrical and Computer Engineering
University of California, Berkeley
Email: ksettaluri6@berkeley.edu

Emily Naviasky
School of Electrical and Computer Engineering
University of California, Berkeley
Email: enaviasky@berkeley.edu

***Abstract*—Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.**

***Index Terms*—Convolutional Neural Network, Approximate Computing, Energy Efficiency**

## I. Introduction

Neural networks have gained immense popularity in recent decades, primarily because of their utility in applications such as computer vision, speech recognition, and natural language processing. Creating a hardware implementation of a neural network, however, is extremely difficult due to the sheer amount of computation required. For example, AlexNet [1] uses 2.3 million weights, and requires 666 million MACs per 227x227 image. VGG16 [2] uses 14.7 million weights, and requires a staggering 15.3 billion MACs per 224x224 image. Understandably, a significant amount of time and research is being used to develop a more efficient, less power intensive, and faster hardware implementation of a neural network.

Although low power implementations have been developed, they are usually at the expense of generalization, wherein specialized dataflow and hardware are developed to minimize data movement and skip unnecessary read or write operations, as in the case of Eyeriss [3]. This hardware specific optimization limits the application to only the one in which it was designed.

### A. Approximate Computing

Hardware specific approximate computing is an optimization approach that efficiently reduces the precision of multiply and accumulate operations in order to speed up a system and save power.

Many approximate adders and multipliers have been designed to reduce energy at the expense of precision. The Approximate Mirror Adder (AMA), for example, removes transistors at the logic level, thus reducing node capacitance while inducing error in the output, and predicts up to 69% [4] savings in power when four out of 16 bits are implemented using an AMA. Voltage Overscaling (VOS) is another technique that reduces the supply voltage while trading off for lower SNR. Several papers have explored VOS designed circuits, and one decoder implementation claims a 22.5% savings in energy while minimally affecting SNR [5].

Considerably less effort has been put into designing approximate multipliers. Previous work explores removing the partial products of the least significant half of the bits in a multiplier and replacing them with 1's or 0's depending on the two multiplicands. Simulations in 0.18-$\mu$ technology estimate a minimum of 50% savings in power dissipation while still having considerable accuracy [6].

### B. Approximate Computing in Neural Networks

Neural networks provide the perfect platform for approximate computing, as they have an innate resilience to imperfections in precision. Implementations such as AxNN [7] and Liu's Deep Neural Network [8] show varying degrees of power savings for minimal loss in accuracy. AxNN claims up to 1.9X energy benefits for no loss in output quality, and an even greater factor of 2.3X when 7.5% loss is permissible at the output through the use of approximate neurons and incremental retraining [7]. Both of these utilizations however, utilize a single approximation algorithm.

Shit you need to mention:
1) Phones use neural networks
2) Phones use approx in conv neural nets to carry out their baby implementations
3) Neural net applications important
4) convNets are used in computer vision

The need for faster and more efficient methods of utilizing neural networks is immense. In particular, convolutional neural networks (convNets) are widely used in many computer vision applications such as pattern recognition and image classification. Though smart phone In applications where hardware capabilities are limited, such as in cellular devices, the efficient application of a neural network could open the door to more useful processing.
Present how its necessary for hardware. This is being done

in phones, but if theres a way to do processing faster more capabilities on the phone can be a thing. This paper presents this blah blah Make sure to mention possible generalization of the system.

## II. Problem Description

-is it worth doing approximate computing in CNN? CNN are a error tolerant application that is prevalent in many power conscious applications even though it requires a large amount of computation to train and use. It is an easy decision to want to trade some of that unnecessary accuracy for power savings and faster computation. However, accuracy for speed and power is not a decision made in a vacuum. Area on silicon is expensive and an entire block for approximate hardware would have to have significant enough power savings to be worth the area. ¡Was there more reasons besides area? I can't remember¿ It is the goal of this paper to explore whether CNN are an appropriate application for approximate computing in a quantitative way. We will begin ¡fuck not using I or we, fix it later¿ by examining CNNs and approximate computing in more detail so that we can ¡talk intelligently about them later.¿

### A. Convolutional Neural Nets

-What is a CNN? CNNs address the problem common to using neural nets in computer vision, which is that it requires an infeasible number of neural nodes¡?¿ in order to process a very small image if each pixel is given it's own node. ¡lots of nn stuff to ask keertana¿ -CNNs are for vision - how they differ from normal NNs (course thing/papers on CNNs) -algorithms and noise that are typically in the system -FOM, how their goodness is judged(error in NNs) -What is approximate computing? There are many implementations of approximate adders and multipliers. Approximate adders vary from using voltage scaling without scaling clock frequency to algorithmically ignoring lower and less important bits[][][]. Approximate multipliers might implement normal multiplier topologies with approximate adders[], but those are generally worse than multipliers that implement approximation at the multiplication level¡not a huge fan of that wording¿[]. - Describe approx comp we want to use -this one is superior cuz BLAH -what are some typical numbers for improvement -they saw X power/ speed improvement vs error -how we are going to model and judge FOM -paper on approximate CNN -They look like a good fit + decent ending sentence

## III. Solution

-Why should approximate computing in CNNs work? -quote some power numbers or something -judge workspace -other works for X loss in accuracy see gain in power/speed -graph power vs accuracy/ accuracy necessary -how much accuracy does a NN need anyway? -paper on approximate CNN - more depth, depending on how useful it was -What we hope to find -Power and speed improvement is significant versus loss of accuracy -Reiterate main question: is it worth doing approximate computing in CNN?

## IV. Description of Experimental Work

-How are we going to judge worth or not worth - Experimental setup -verilog for power numbers, size, error rate -python, lanet (reference more course thing), blah blah dont wanna train one -plug in error rate/introduce same amount of error -see if shit still works - judge goodness -FOM -what are we doing in particular to judge power/speed/usability of NN

## V. Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

## Appendix A
## Proof of the First Zonklar Equation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

## Acknowledgment

The authors would like to thank...

## References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.