Towards energy efficient deep learning

A Project Report

submitted by

GIRIDHUR SRIRAMAN

EE13B129

in partial fulfilment of the requirements

for the award of the degrees of

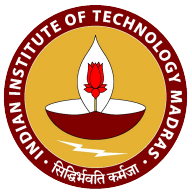
BACHELOR OF TECHNOLOGY

&

MASTER OF TECHNOLOGY

in

ELECTRICAL ENGINEERING



Reconfigurable Intelligent Systems Engineering

Interactive Intelligence Laboratory

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

CHENNAI 600036

JUNE 2018

CERTIFICATE

This is to certify that the project report titled **Towards energy efficient deep learning**, submitted by **Giridhur Sriraman (EE13B129)**, to the Indian Institute of Technology, Madras, in partial fulfillment of the requirements for the award of the degrees of **Bachelor of Technology in Electrical Engineering** and **Master of Technology in Electrical Engineering**, is a bona fide record of the work done by him in the Department of Electrical Engineering, IIT Madras. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof Ravindran B.
Project Guide
Professor
Department of Computer Science and Engineering
IIT Madras Chennai 600036

Prof. Aravind R.
Project Co-Guide
Professor
Department of Electrical Engineering
IIT Madras Chennai 600036

Place: Chennai

Date: June 19, 2018

ACKNOWLEDGEMENTS

This thesis wouldn't have been possible I haven't had the opportunity to meet with Prof Ravindran in my 2^{nd} year to work on an Analytics club project, as part of CFI. He has been a true friend, philosopher and guide, in almost all of my memorable endeavors during my time at the institute. I would also like to acknowledge Prof Anand Raghunathan (Purdue University) for being a great mentor and a gracious host, his support throughout the summer and the term after meant a lot; Prof Pratyush Panda, and Prof Aravind R, for guiding me through my final year, and agreeing to co-guide with Prof Ravindran. Special thanks to Athindran R., for helping me through my brick-walls, both academic and non academic. My family has also supported me through my final year in times of distress and need, my sincere thanks to them. And, finally, my heartfelt gratitude to all my friends and well wishers, for making my insti life memorable and enjoyable.

ABSTRACT

KEYWORDS: Deep Learning, energy efficiency, fine-grained control, neuromorphic, convolutional neural networks, image classification

Machine Learning has been all-pervading and its presence in devices around us cannot go unacknowledged. In this work, we explore the energy consumption of popular machine learning algorithms, with focus on deep convolutional neural networks applied on common tasks such as object detection and object classification; and we seek to identify and correct efficiency aspects of these systems.

This work will explore three key points in chapters :

1. Using fine-grained control on neural networks on embedded systems.
2. A neuromorphic approach to object detection and tracking in videos.
3. Modeling the power-accuracy trade off in hardware used for machine learning.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology, Madras
ML	Machine Learning
DL	Deep Learning
NN	Neural Network
DNN	Deep Neural Network
CNN	Convolutional Neural Network

This page intentionally left blank

CHAPTER 1

INTRODUCTION

1.1 Brief Introduction to Deep Learning

Machine learning is a field of computer science, which aims to give the ability to automated systems to progressively improve on a specific task through statistical techniques. ML can be broadly classified into three categories depending on the relation between the signal input and the target output :

1. **Supervised Learning** : the input and the target are both given with a predetermined metric to optimize on, examples include : Object Detection, Face Recognition.
2. **Unsupervised Learning** : the output target is unspecified, there is no specific accuracy metric, examples include : Clustering, Anomaly Detection, Density estimation.
3. **Reinforcement Learning** : the input is an "agent" which map situations to actions in order to maximize a reward output signal, with the actions not known before hand, examples include : getting around a maze, playing poker.

Deep learning is a branch of machine learning, which uses methods vaguely inspired by the structure of brain., and its constituent neurons. Deep learning has picked up tremendous pace over the past few years, thanks to advances in hardware and better mathematical models alike. DL has helped us achieve super-human accuracy in tasks that were previously deemed to be "difficult", such as object detection, tumor segmentation, and DL has surpassed humans in erstwhile unfathomable games such as Chess, and Go (Silver *et al.*, 2017).

The "perceptron" learning algorithm, invented in 1957, by Frank Rosenblatt has been widely acknowledged to be the first artificial "neuron", or the basic building block of a deep learning system (Rosenblatt, 1957). This algorithm proved to be quite popular, although its limited applicability to just linearly separable problems was an issue., which was later rectified by adding more layers.

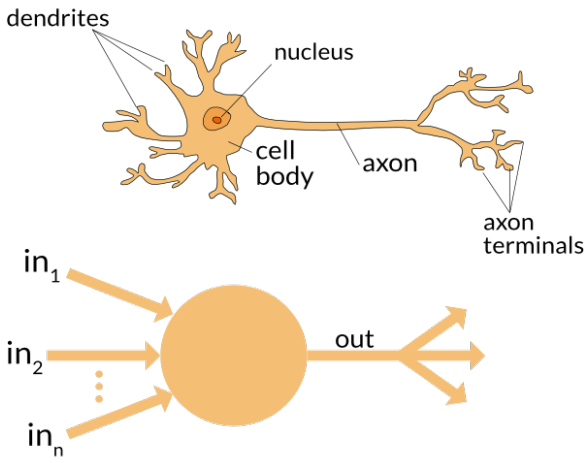


Figure 1.1: A neuron compared to a perceptron

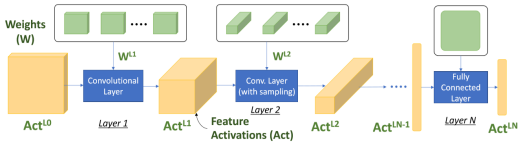


Figure 1.2: Schematic of a typical DNN

This multi-layer perceptron was somewhat difficult to train, and in 1986, (Rumelhart *et al.*, 1986) proved experimentally the back-propagation algorithm could generate useful internal representations in these neural networks.

Despite numerous advancements in neural networks, DL lost out to simpler algorithms such as SVM, Decision Trees, owing to the complexity in training them. Come late 2000s, compute power became cheap and far accessible in the form of Graphic processing units (GPUs), which were initially used for games. GPUs were capable for fast matrix and vector multiplication, resulting in orders of speed up over conventional CPUs, and thus a prominent factor behind the recent revival of Deep Learning. Almost all leading contests in pattern recognition,

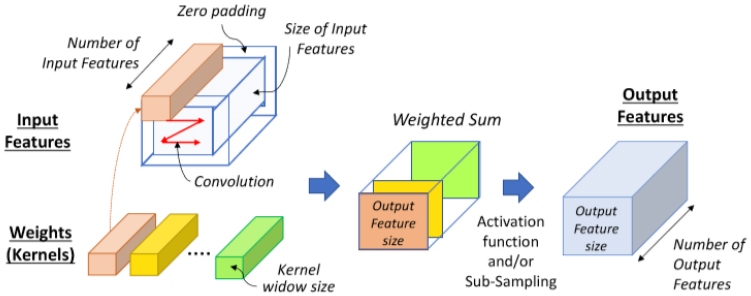
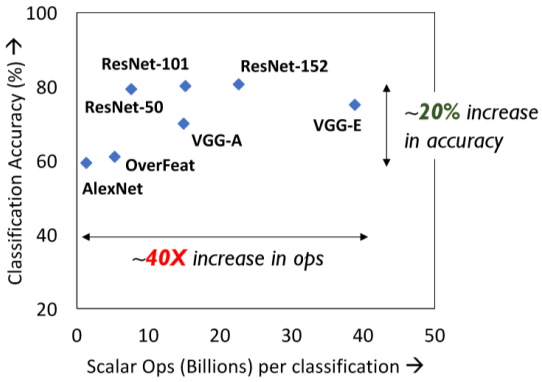


Figure 1.3: Schematic of a typical Convolution Layer



object detection etc, have now been won by GPUs and modified neural network architectures, beginning with Alexnet (Krizhevsky *et al.*, 2012) winning the Imagenet challenge in 2012. Winning entries of the latest Imagenet Challenge, Imagenet 2017 (Russakovsky *et al.*, 2015), have accuracies exceeding those of humans (reported at around 95%)!

1.2 depicts the data flow in a typical deep neural network, while 1.3 gives us a closer look at a convolutional layer in a deep neural network.

1.2 Motivation

With the growing ubiquity of Deep Learning, we seek to understand how DL influenced other aspects of computing, with specific interest in power consumption. The ultimate goal of DL is to achieve the standards of the human brain. The table here 1.1 Sandberg compares top of the end commercially available computers to the brain.

Table 1.1: Comparison of Brain vs PC

Parameter	Brain	PC
Weight	1.5kg	10kg
TFLOPS	>1000	15
Power	25 W	500 W
Value	Priceless	$2000

There is a considerable gap both in performance and efficiency for us to close!

1.2 compares recent recent neural network architectures in terms of their Ops count and accuracy.

Furthermore, with the development of newer architectures (topologies, layers, features sizes), and increasing data sizes in Deep Learning, this gap is only expected to widen. Over the recent few years, the growth in computational requirements has far outpaced the improvements in capability of computational platforms in the recent years.

This work tries to move DNNs an inch closer towards better energy efficiency, through contributions broadly grouped under the following heads:

1. Dynamic control of neural network parameters to reduce computation, on embedded systems
2. Dynamic object centered video object detection framework
3. Analyzing hardware trade-offs in improving accuracy

CHAPTER 2

Dynamic neural networks on embedded systems

In this chapter, we explore performance of Deep Learning systems on embedded platforms, such as Raspberry Pi^{TM}. The chapter begins with a brief description of related work, proceeds to a detailed discussion on DyVEDeep (Ganapathy *et al.*, 2017) describing its implementation and performance on Raspberry Pi 3. The following table describes the specifications of Raspberry Pi 3 2.1.

Table 2.1: Specifications of Raspberry Pi 3

Module	Specification
CPU	BCM 2837 ARM Cortex-A53 Quad Core @ 1.2GHz
RAM	1 GB LP-DDR2 900MHz
GPU	VideoCore IV

Raspberry Pi3 was chosen because it is by far the most popular ARM system in use, owing to its value for money, it retails for $35, both in industry as well as hobbyists for IoT, Machine Learning etc.

It is important to note that the GPU in the chipset cannot be explicitly used for any sort of acceleration, since it doesn't support CUDA or any similar API. The DRAM chipset is shared among both the CPU and GPU.

DyVEDeep : **D**ynamic **V**ariable **E**ffort for **Deep** neural networks uses the difference in composition of various inputs to improve its compute efficiency. This motivation stems from the following observations.

First, all inputs are not equally difficult for the network to process, since only those which lie close to the boundary need the full effort of the classifier, while those which lie far away can make do with a simple linear classifier. And, adding layers might seem like a good idea to increase accuracy while, this choice gives us diminishing yields. For example, in the context of the Imagenet challenge, computation requirements have gone over 15x from AlexNet to VGG, while there has only been a 16% improvement in accuracy. Visualization for the MNIST network is in 2

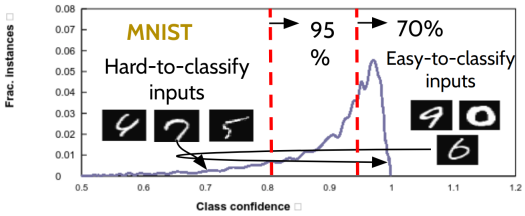


Figure 2.1: Classification threshold density plot for MNIST

Secondly, effort expended for a given input is spread across different parts of the network. In case of an object recognition problem, this means that computations stemming from regions where objects of interest are located, are more critical than regions without.

Finally, most deep neural network systems are static, ie, they expend the same (worst case) computational effort for all inputs, which leads to significant inefficiency. DyVEDeep addresses this by dynamically predicting and executing relevant computations, while skipping or approximating the rest. This means that, in effect, the network spends lesser effort on "easier" examples without sacrificing accuracy.

2.1 Related Work

Prior research in the area of improving efficiency of neural networks can be divided into these 4 broad sectors.

First, there has been a lot of work in focusing on parallelizing various aspects of the architecture, on commercial multi-core and GPGPU platforms. Different work strategies such as model, data and hybrid parallelism (Krizhevsky, 2014);(Das *et al.*, 2016), and techniques such as asynchronous SGD , 1-bit SGD (Seide *et al.*, 2014) are representative examples.

Secondly, there have been efforts focused on developing specialized hardware accelerators that realize fundamental computational kernels in deep neural networks, such as the Efficient Inference Engine (Han *et al.*, 2016), and NeuFlow (Farabet *et al.*, 2011), these however compromise on programmability, cost of specialized hardware in lieu of efficiency.

The third set of efforts focus on novel device technologies, whose characteristics intrinsically match compute primitives in deep neural networks, these include the neuromrphic chip TrueNorth (Modha, 2017), and spintronic neural design (Ramasubramanian *et al.*, 2014), among others.

Finally, there have been approaches looking at the overparametrization in DNNs, owing to their non-convex space, they approximate DNNs through various heuristics. We shall classify them into "Static" and "Dynamic" approaches respectively, with respect to the input.

Static Approaches: Most of the work to approximate DNNs has been static in nature, ie, they change the model independent of the input signals, and hence the same heuristic is applied across all inputs. These aim to reduce to overall model size of the network, by using techniques such as Network pruning, (by pruning the connections) as done in (LeCun *et al.*, 1990), (Han *et al.*, 2015*b*); reducing the computing precision as in (Venkataramani *et al.*, 2014), and (Anwar *et al.*, 2015); and storing weights in a compressed format (Han *et al.*, 2015*a*); Ex, for fully connected networks, Hashnets (Chen *et al.*, 2015) randomly group weights into bins using a low cost hash function, which share a common parameter value, thereby reducing the descriptor length of the neural network. Deep-Compression (Han *et al.*, 2015*a*), prunes connections by including a regularizer while training, and zero-ing out weights below a certain threshold, ie removing connections.

For convolutional layers, (Denton *et al.*, 2014) (Jaderberg *et al.*, 2014) exploit the linear structure of the network to find low-rank approximations., and then there are sparse CNNs as proposed by (Liu *et al.*, 2015*a*), by using a sparsity term in the objective function, leading to almost 90% of the parameters being zeroed out. PerforatedCNNs, are proposed by (Figurnov *et al.*, 2015), where only a few neurons are evaluated for each feature, statically determined during training.

Dynamic Approaches : These focus on changing parameters/computations involved depending on the input, and are more powerful than static techniques owning to their increased expressibility. Stochastic neurons have been the earliest approaches in this direction, as in(Bengio, 2013), which describes a method to estimate gradients for stochastic neurons, useful for conditional computation.

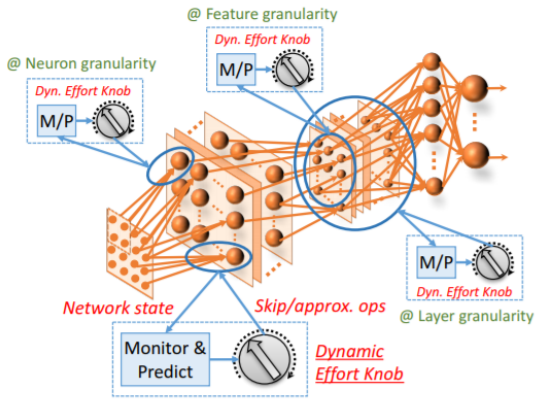


Figure 2.2: Schematic of the DyVEDeep

Standout, as described in (Ba and Frey, 2013) is another approach where the dropout probability for each neuron is computed in a binary belief network, in one shot, conditioned on the inputs. An extension of this, with dropout distribution of each layer being computed by the previous layer has been done in (Bengio *et al.*, 2015)

DyVEDeep is orthognal to all the above efforts, it uses deterministic heuristics rather than stochastic approaches. Such dynamic techniques have only been applied to small datasets and have not been extended to large-scale DNNs. DyVEDeep is applicable to both convolutional and fully-connected layers, measurable improvements can been seen on networks made for the ImageNet dataset

2.2 Approach and Implementation

First, we will discuss how various features of DyVEDeep are proposed and and then move on to implementation on an embedded platform,a Raspberry Pi3.

DyVEDeep looks to improve the processing efficiency of a DNN by equipping the system with "effort" knobs, that dynamically predict criticality of groups of computations with low overhead, and correspondingly skip/approximate them, as in 2.2. There are 3 different approaches detailed :

2.2.1 SPET - Saturation Prediction and Early Termination

SPET works at a neuron level, the finest level of granularity. SPET can be used for any neuron with an activation function that has atleast on side where it saturates, one such activation which is commonly used is the Rectified Linear Unit (ReLU), that saturates for negative values, to zero. Main motive behind SPET is that the sum-dot product between weights and neuron inputs is not indicative of the neuron's output, viz if the sum-product is established to be negative in sign, it is unnecessary to calculate the actual sum-product, since the output from ReLU will be zero.

SPET 2.2.1 uses a low and high threshold to monitor the partial sums being processed, and stops accordingly. The hyper-parameters for SPET, ie the low and high threshold are tuned through a grid search.For one-sided activation functions, such as ReLU, there needs to be only one threshold, either low or high, (for ReLU : low).

SPET's efficacy has been demonstrated in the CIFAR-10 2.2.1, where between 50% to 70% of the neurons saturate. Since, DNNs generally have a lot of saturated neurons, this performance is expected and observed for AlexNet, and Overfeat, as well.Additionally, the number of neurons saturating increases with their "depth" owing to their specialization to a particular higher order feature.

The prediction interval for SPET (number of inputs to be processed for partial sum before action can be taken) was found to be optimal at 50%, however this could also be made into a hyper-parameter. This is implemented by processing all the odd-indexed neuron inputs first, and then taking the decision to skip/compute the remaining inputs based on the partial sum and the threshold. Since, this is a hard decision, this imposes an upper cap on the maximum efficiency that it can attain. The prediction interval can be further decreased by ensuring that higher weights are processed first, owing to their increased expressibility towards the final sum.

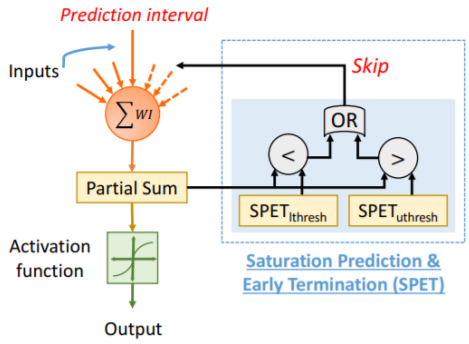


Figure 2.3: Schematic of the SPET heuristic

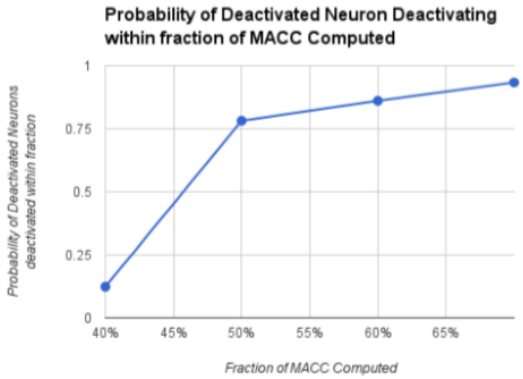


Figure 2.4: Saturation prediction accuracy at different prediction intervals

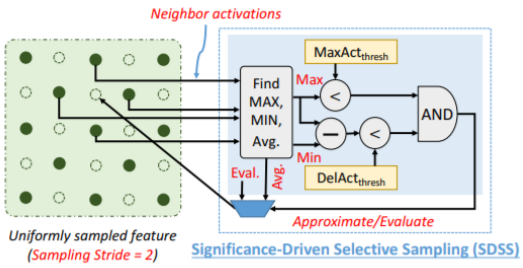


Figure 2.5: Schematic of SDSS

2.2.2 SDSS - Significance Driven Selective Sampling

SDSS operates at a feature-level granularity,in a convolutional layer in a DNN. It exploits the spatial locality of activations, within each feature, viz in images, neighboring pixels are expected to have similar values. For the convolution partial sum, the kernel slides over the image, and as this happens, the aforementioned redundancy inevitably creeps in, and this is observed in deeper layers of the network as well. This goes hand in hand with the saturated behaviour of activation functions, as variations between neighbours are masked if they fall in the same saturated range.

As described in their paper, SDSS involves two steps to exploit intra-feature locality :

1. **Uniform Sampling** : Activation features are computed for a subset of neurons by uniformly sampling the feature, a hyper-parameter SP, defines the periodicity of sampling in each dimension. This is based on covariance of neuron activations, and size of the feature. SP was chosen to be 2 across all experiments.
2. **Significance-driven Selective Evaluation** : For neurons that were not selected in the previous step, the activation values are approximated. Two other hyper-parameters : Max Activation value threshold and Delta Activation Value Threshold are defined. For a neuron whose activation is yet to be computed, ie, not computed in the previous step, we examine the activation values of its computed neighbours in all directions, and their maximum, range is computed.If the maximum, and range both are less than thresholds, the activation value of the neuron is approximated to average of its neighbours; if not, the neuron is evaluated.

SDSS, uses a simpler form of magnitude, and variance to decide a neuron's position with respect to its activation's criticality, and accordingly expends computational effort.

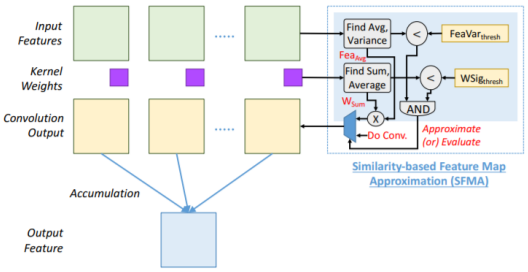


Figure 2.6: Schematic of SFMA

2.2.3 SFMA - Similarity-based Feature Map Approximation

SFMA, operates at a layer level, and exploits the correlation between activation values in a feature, in a way, different from SDSS. Here, the spatial locality is used to approximate computations that use the feature as their input. Consider a convolutional layer, in which one of the input features, has all of its neuronal activations similar, when the sliding window kernel operation is performed, the outputs are expected to be similar as well. SFMA approximates the entire convolution, by first finding the mean neuron activation, and summing up the kernel weights, then the approximate output is given by the product of this kernel-weight-sum, and the average input neuron activation.

This can be expressed mathematically as :

$$ConvOut_w = \Sigma_{i=0}^{k^2} w_i * W_i = \Sigma_{i=0}^{k^2} \mu * w_i + \Sigma_{i=0}^{k^2} (W_i - \mu) * w_i \approx \mu \Sigma_{i=0}^{k^2} w_i \tag{2.1}$$

The following hyper-parameters are defined to decide on which convolutions to apply the aforementioned approximations :

- Weight Significance Threshold : set as the sum of absolute values of kernel weights, is an approximate measure importance of current convolution to the output feature
- Feature Variance Threshold : set as variance of neuron activations in the feature

Given hyper-parameters, the convolution is approximated when (i) sum of absolute values of kernel weights are below the significance threshold, (indicating that the convolution isn't important and (ii) variance less than the threshold, indicating that the error incurred by replacing with the average is within tolerable

limits

2.2.4 Implementation specifics

The effort knobs can be integrated as follows. To combine, SPET and SDSS, each neuron activation across the uniformly sampled features of SDSS are computed with SPET. SPET is applied after SDSS is complete. SFMA collectively combines the inputs within a sliding window, to a neuron, into a single output and fits perfectly in the process of evaluating a neuron with SPET and SDSS.

SPET is applies to all kinds of layers, both convolutional and fully-connected, whereas the SDSS,and SFMA are tailored to Conv layers. The middle-depth convolutional layers have a lot of incoming inputs per neuron, and since they're expected to be saturated, SPET will be most beneficial in such cases. SDSS works best when feature sizes are large, since much fewer convolutions than necessary will be performed, and thus initial convolutional layers are likely to benefit most from SDSS. SFMA works best when there are a large number of features per layer, with smaller feature sizes, and is optimally made for middle and later convolutional layers.

The hyper-parameters are tuned layer by layer, first for each convolutional layer, and then for the fully connected layers. There are 6 Hyper-parameters in all, and they yield a direct trade-off between computational savings vs classification accuracy. The hyper-parameters were tuned on a pre-trained network, with the added constraint that the net drop in accuracy is less than 0.5%, this search was carried out over a range, and binary search was used, since the behavior of these hyper-parameters are monotonous with respect to accuracy over a continuous range.

2.2.5 Implementation platforms

While in the original paper, the authors describe implementation on a regular x86 platform, we seek to port this system to an ARMv7 platform. Since all the knobs were implemented with high level code, owing to their need for fine-grained

computations, viz, SPET looks at partial sum inputs to a neuron, which is usually computed as a matrix multiplication implemented in libraries such as OpenBLAS, ATLAS, etc, where assembly level optimization is done to increase performance. This rules out usage of low-level APIs provided by Caffe (Jia *et al.*, 2014) which use BLAS routines as their backbone, and hence the implementation was in high level code.

The baseline was made by implementing a plain vanilla convolution layer, with high level C++ code, needing to make a considerable comparison between the optimizations involved and

CMake system was used to build files for the ARMv7 platform, and methods needed to be refactored to make better use of RISC specific features, and improve memory management.

2.2.6 Experiment Methodology

We now describe the experiments carried out to evaluate DyVEDeep on the Raspberry Pi. Pre-trained DNN models from Caffe Model Zoo (BVLC), were chosen in view of DyVEDeep's adaptability. The following networks were used : CIFAR-10 network trained on the CIFAR-10 dataset; AlexNet (Krizhevsky *et al.*, 2012), compressedAlexNet (Han *et al.*, 2015*a*), VGG-16 (Simonyan and Zisserman, 2014) trained on ILSVRC '12 dataset, (Russakovsky *et al.*, 2015). The inputs for the network are generated by using a 224x224 center crop of images in the test set. 5% of the test inputs were randomly selected, and used to tune hyper-parameters. The rest were using in evaluating speed up and accuracy. This was done on an Intel Xeon server operating at 2.7GHz, with 128GB of RAM.

After obtaining the hyper-parameters, another 5% of test inputs were selected, and the models were run on Raspberry Pi3, with these optimal hyper-parameters. FLOP counters and timers were embedded within the code to measure compute reduction and execution time. Knobs only tested on convolutional layers, since they take up more time during forward inference, but reported times and benefits include time taken in all layers in the network

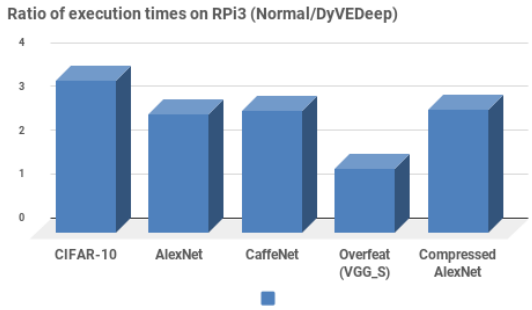


Figure 2.7: Improvement in execution time

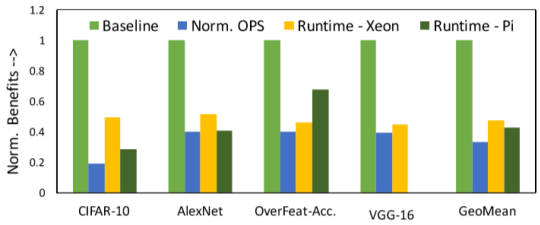


Figure 2.8: Improvement in computation count

2.3 Results

Results of all the experiments run are presented here.

2.3.1 Improvement in FLOPS and Execution time

First, we present the improvement in scalar operation and overall execution time. Figure 2.3.1 shows ratio of time taken to perform a normal forward inference on baseline to that in DyVEDeep. We see that DyVEDeep, attains a measurable speedup ranging between 1.5x-3.8x in time, corresponding to 1.5x-4x in ops count, as in **??** The overhead in running DyVEDeep was estimated to be less than 5% of the total number of operations. The difference in classification accuracy between DyVEDeep and the baseline was found to be less than 0.5% The time benefits don't seem to scale as much as the Ops count, this is expected since the dynamic effort knobs require a different memory access, need additional variables to be taken care of, and control operations in the software package set an upper limit to DyVEDeep's contribution to runtime.

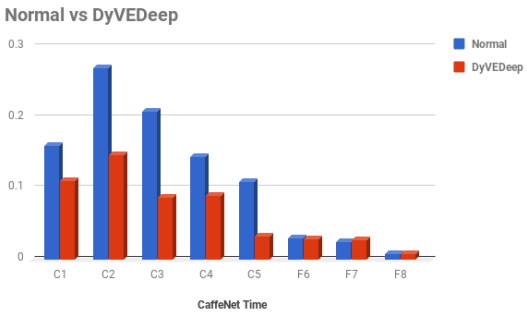


Figure 2.9: Layer-wise breakdown for CaffeNet architecture

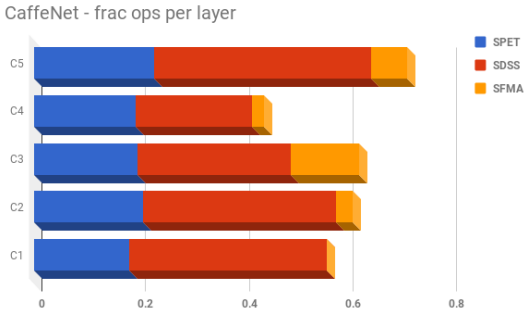


Figure 2.10: Knob-wise breakdown for CaffeNet architecture

2.3.2 Layer-wise and Knob-wise breakdown of savings

2.9 breaks the runtime savings across layers of the CaffeNet architecture, with layers horizontally, and runtime normalized to baseline runtime of the DNN, on y-axis. C1 layer has 11x11 kernels, with a stride of 4, and hence SDSS wouldn't be able to exploit a lot of local similarity. Since there're very few input features, SFMA is not very effective, and in early layers, very few neurons saturate, and so SPET's effectiveness is low. And thus, as seen in the figure, we achieve INSERTTEXTHEREx reduction in runtime in first two convolutional layers, which increases to 2.6x in C3,C4,C5.

2.10 breaks down savings into those from each knob, for each convolutional layer in CaffeNet. SDSS gives us maximum savings, close to 30% of scalar operations, while SPET and SFMA follow with 20% and 8% respectively. Our hypotheses about knobs being effective as we go deeper, was also observed experimentally.

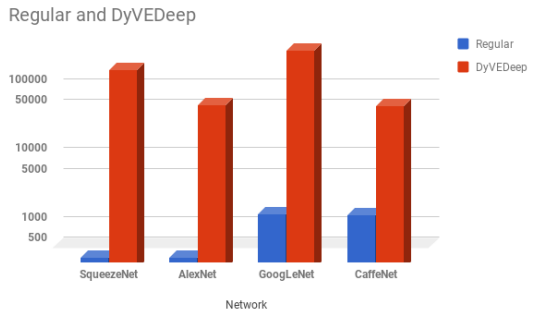


Figure 2.11: Comparison vs Regular inference

2.4 Limitations and Future Work

DyVEDeep is able to show similar effectiveness on embedded systems as well, and thus, is platform and network agnostic, it could be repackaged as a simple add-on module for convolutional neural network packages DyVEDeep performs considerably faster than the baseline, but it is of real concern that the baseline network is a little outdated, and is nowhere as fast as running times using BLAS libraries.This is mainly due to the fact that BLAS libraries are heavily tuned for performance at assembly level, while our code is at a very high level, thus suffers massive overheads. This can be seen in 2.4

Another shortcoming with DyVEDeep is that, it only runs on a CPU, and is not all the effort knobs are inherently parallelizeable to run on a GPU, which is the standard choice of platform for most, if not all deep learning packages. GPUs feature an aggregated cluster of reduced instruction CPUs with shared memory; GPUs are designed for parallel computing. SPET, and some features from SFMA can be ported to GPUs without incurring overheads, while SDSS which forms basis of most of the runtime improvement cannot be easily transformed to GPU code unless the core functionality is reworked to be composed as basic GPU stream operations, since SDSS involves a lot of branching, and serial processing.

CHAPTER 3

Multi-Fovea : Eye-inspired video object detector

Continuing with the dynamic effort model in DyVEDeep, we move on to one of the most common tasks in computer vision : object detection, where we specifically concentrate on performing object detection on video streams. Object detection has been a notable problem in computer vision, which traces its origins to a summer project in 1966, at MIT, put forward by Seymour Papert and Marvin Minsky (Papert, 1966). Needless to say, this helped them realize the complexity of the problem, and spawned a new field. Close to 5 decades on, we've computer vision systems overthrowing humans in object detection, with top-5 error rates in the Imagenet challenge achieving less than 4 % , human top-5 classification error rate being reported at around 5 % (Russakovsky *et al.*, 2015). This problem has also been the subject of an xkcd comic sketch 3.1

The ImageNet challenge has been one of the key driving forces behind object detection research, and deep learning models have seized the top positions since 2012, starting with AlexNet (Krizhevsky *et al.*, 2012). Object detection models are used in many other domains in the world today, including Robotics, Manufacturing, and lately, even in Transportation, with the advent of self driving cars.

The ImageNet Video challenge (Russakovsky *et al.*, 2015), started in 2015, is an annual challenge where submissions compete to win the top spots, where there models perform object detection in video streams, across 30 object categories (which are a subset of the 200 basic level categories in ImageNet). The winning entry in 2017 edition, attains a mean-average precision of 0.8172, is by a joint team from Imperial College, London and University of Sydney, with an optical flow based model (Zhu *et al.*, 2017).

Object detection in video has a wide variety of applications, starting from surveillance and anomaly detection, to navigation and path finding. It is thus pertinent to make a system that has an acceptable detection rate, while simultaneously being energy efficient.

Figure 3.1: xkcd # 1452, by Randall Munroe

The field of video processing involves many similar problems, tailored for different applications, broadly, (i) Object detection - refers to detecting and localizing objects within each frame, (ii) Object tracking - refers to following moving objects across the video stream, this has 2 subproblems - detection (same as (i)) and association (matching detections across time), and (iii) Segmentation - refers to splitting a given frame into different contiguous regions (such as separating the foreground and background,

The problem of detecting objects in videos cannot be directly applied, since the appearance of the objects might vary, (viz, A man facing the camera for a while and turning around) and the detections need to contain a temporal element, since the appearance of an object in a video is highly correlated between neighboring frames.This means that the collection of detections scores pertaining to the said object, over a certain window of time shouldn't change dramatically., and thus the temporal consistency of the detections need to be regularized. These extra constraints are very similar to the object tracking problem, and we borrow some concepts from there as well.

Deep CNNs have shown impressive performance on object tracking, as seen in (Nam and Han, 2015), other such papers. Temporal information been able to regularize the detections, as seen in many multi-pedestrian tracking settings such as (Bae and Yoon, 2014), (Leal-Taixé *et al.*, 2014). Directly using still-image trackers on object tracking results in 35.7% mAP compared to 45.3% mAP on

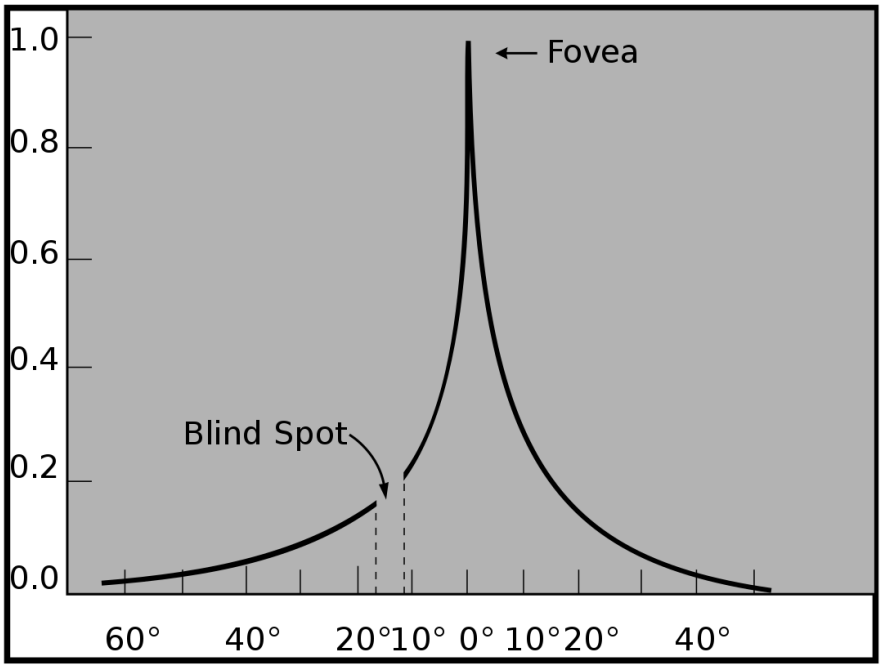


Figure 3.2: Normalized angular distribution of photo-receptors in the human eye

object proposals, this arises because the detector is sensitive to location changes, there are box mismatches between the tracked proposals and object proposals; as seen in (Krizhevsky *et al.*, 2012). Our vision for this problem however, is two fold, we are seeking to achieve measurably accuracy performance with real-time processing (30FPS or above)

This work introduces a video object recognition inspired by the biological eye, for efficient processing. The human eye works by focusing photons through a set of converging lenses - cornea and lens to the retina, composed of photo-receptive cells(rods and cones). The eye has an attention mechanism, guided by the brain, which helps us focus on specific objects of interest by keeping them in our line of sight, and detract interest from objects in peripheral vision. More detail is captured in the objects along the line of sight, owing to non-uniform concentration of photo-receptors in the retina as see in 3

The central portion with highest concentration of photo-receptors is termed the **Fovea**, and we adopt this term in our name owing to spending more computations for specific regions in the image.

Our work is able to achieve near real-time performance with an average mAP of 0.536 upon blind testing on the ImageNet VID data set, with the baseline

trained on ImageNet-DET dataset.

The following chapters will describe related work approaching the same problem, then a detailed discussion on our methodology and approach, finally concluded by results, and a brief discussion on limitations and future work.

3.1 Related Work

In this section, we will look at prior research in the area of object detection, which are extended to video processing. We will have a deeper look into methods which take efficiency into account, and beyond this we will explore work in efficient video processing - in other similar problems such as segmentation, caption and action localization.

3.1.1 Object Detection in Still-Images

Approaches in object detection can be broadly classified into : **Region Proposal Based**: Region proposal based approaches process only selected regions of the image to reduce compute effort on unnecessary regions. Selective Search (Uijlings *et al.*, 2013) is one of the earliest approaches where possible object locations for use in object detection found before processing them through a support vector machine. Such a search algorithm captures all scales, uses a diverse set of strategies to generate a hierarchy of possible object locations, and finally is efficiently calculated. R-CNN (Girshick *et al.*, 2014) establish the supremacy of CNN-based features over traditional HoG variants, uses modified selective search to improve object detection. To speed this process up further, Fast R-CNN (Girshick, 2015) pools are regions of interest, and uses a unified CNN based model, to obtain detections after a single pass. Faster R-CNN (Shaoqing Ren *et al.*, 2015) uses the CNN features maps to use for region proposals, and thus cuts down on the time further.

Single Stage Detectors: This class of object detectors use a single network to predict both bounding boxes and object classes. Famous architectures in this direction include the YOLO, and SSD families. YOLO (Redmon and Farhadi, 2018)

divides a region into smaller regions and applies a neural network which predicts bounding boxes and probabilities for each region. YOLO (You only Look once), performs only one pass through the CNN, and is extremely fast, however, each region is restricted to a single class, and hence YOLO struggles with small objects. SingleShotDetector (SSD) (Liu *et al.*, 2015*b*) is another popular family, uses a fully convolutional network, and applies small sized filters to predict classification scores and bounding boxes; SSD produces outputs after multiple convolutional layers to handle scale. RetinaNet (Lin *et al.*, 2017), uses a Feature-Pyramid-Network (Lin *et al.*, 2016) (feature outputs for an image at various scales, along with a modified loss function, which increases focus on misclassified examples.

Table 3.1: mAP and inference times (in ms) on COCO test-dev

Network	mAP	time
YOLOv3	21.6	25
Faster-R-CNN	24.2	198
SSD321	28.0	61
SSD513	31.2	125
FPN-FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	37.8	198

Current state of art object detectors use a single-stage approach to cut down on runtime. We've summarized mean-average precision (mAP) and inference time for a few of them in 3.1

3.1.2 Object detection/localisation in videos

Since, most of the aforesaid networks have fast enough runtime, they are usually run in a loop, for every frame to detect objects in a video, such as SSD, RetinaNet

Object localisation is a similar problem, on the YouTube objects dataset, where the underlying assumption is that there is only one object of interest in a frame, while each frame of ImageNet-VID may have many objects in a single frame. The evaluation metric for object localisation is "CorLoc", while mean Average precision is used on the ImageNet-VID task, this makes the ImageNet-VID challenge much more difficult, and closer to reality. Action recognition is another problem where the system is required to annotate a bounding box for human action of in-

terest. Since temporal coherence is of more importance here, most approaches use Optical flow, to propagate detections and features through time.

In, Object Detection with Tubelet networks (Kang *et al.*, 2016), the authors use a spatio-temporal tubelet proposals by combining still-image object detector with a tracking module, in a 3 step - object proposal, proposal scoring, and tracking. They perturb the proposals, and score these new ones to attain a greater accuracy. This performs at a meanAP of 47.5% over ImageNet VID.

Some approaches have also tried a mixture of CNNs and LSTMs to take advantage of the temporal dimensions, the downside is of course heavy computation, and a difficult training process, (Donahue *et al.*, 2015)

Other approaches, use optical flow to associate detections over time, and have performed exceedingly well at the challenge, the 2017 winners, (Zhu *et al.*, 2017), which warps ResNet feature maps based on optical flow, does a weighted aggregation to propose detections, this runs at around 204ms per frame, to achieve an mAP of 0.69, while the better configurations achieve higher mAPs with increased runtime.

3.1.3 Context based approaches

Here we will look at other papers discussing the use of fovea based methods : (Larochelle and Hinton, 2010) describes applying an attention field over the input, and processing the other parts with lesser importance; which is used to make a multi-fixation RBM, which performs as well as state of art on MNIST, and synthetic datasets

(Karpathy *et al.*, 2014) describes usage of a high-resolution feed (central fovea), and a low-resolution context feed for improving efficiency. The authors also consider "fusing" features within various time windows for increasing accuracy.

Our approach has been built on the idea of using many fovea and skipping unnecessary computation on the other areas. This puts our mAP over (Kang *et al.*, 2016), while giving near real-time performance

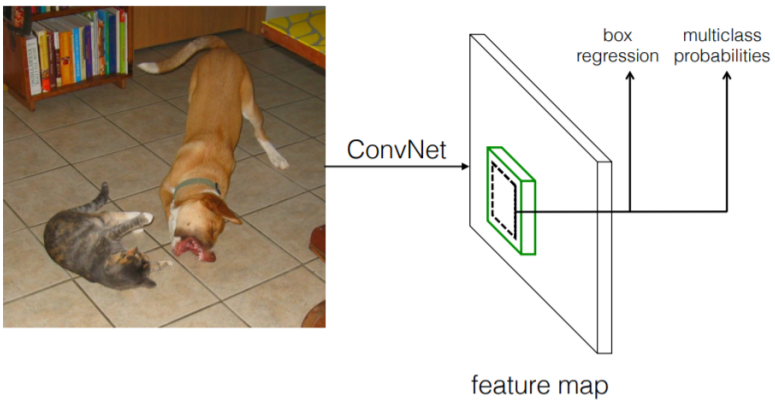
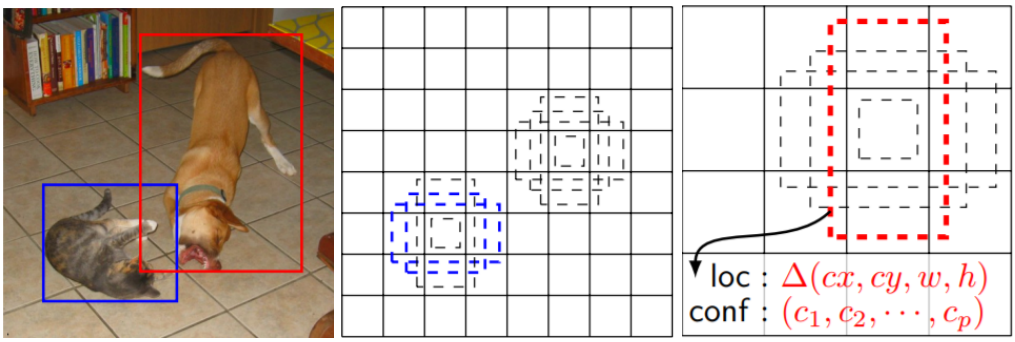


Figure 3.3: SSD Framework:a schematic



(a) (b) (c)

Figure 3.4: SSD Framework

3.2 Approach and Implementation

We divide this section into two parts. The first part will describe the choice of baseline, after a brief comparison of baselines; while the second part will delve into our approach, and implementation specifics. We choose the SSD300 (Liu *et al.*, 2015*b*) as our baseline network, and our approach is modular and works on top of it.

3.2.1 Baseline - SSD

SSD - Single Shot MultiBox detector - is a single stage detector, which is fully convolutional, which does bounding box regression, and gives us multiclass probabilities simultaneously. SSD is built on a pre-trained VGG16 network, schematic as in 3.2.1 .The approach predicts bounding box offsets, and multiclass probabilities for each box. Feature maps from many successive layers taken, this improves

detection accuracy across scales, and predictions are explicitly separated by aspect ratio; this leads to a very high accuracy. An example is depicted in 3.2.1 (Liu *et al.*, 2015*b*), (a) shows the Ground truth boxes, and (b) shows anchor boxes at a given location with 8x8 feature maps (c) shows similar for 4x4 feature maps, red box highlights regressed outputs from that box - location offsets, height,width and multiclass probabilities.

SSD's unique training methodology also gives it a headway in performance and precision, over other competitors such as YOLO. SSD uses a special "multibox" layer for prediction for both boxes and classes simultaneously. During training, a threshold is set, and any default box matching a ground truth box with jacquard overlap higher than the threshold is chosen, and the objective function is a weighted sum of the localization loss (L_{loc}) and the confidence loss (L_{conf}). Let $x_{ij}^{p} = 1,0$ be an indicator for matching the i-th default box to the j-th ground truth box, of category p. The authors choose a threshold of 0.5, and they define the overall loss as follows :

$$L(x,c,l,g) = \frac{1}{N}(L_{conf}(x,c) + \alpha L_{loc}(x,l,g)) \tag{3.1}$$

where, N is the number of matched default boxes. If N is 0, then loss is set to 0. The localization loss is a Smooth-L1 loss between the predicted box (l) and the ground truth box (g) parameters. The offsets for the center (cx, cy) of of the default bounding box (d), width (w) and height (h) are regressed, from ground truth values.

$$L_{loc}(x,l,g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx,cy,w,h\}} x_{ij}^{p} \text{smooth}_{L1}(l_i^m - \hat{g}_j^m) \tag{3.2}$$

$$\hat{g}_j^{cx} = (\hat{g}_j^{cx} - d_i^{cx})/d_i^{w} \tag{3.3}$$

$$\hat{g}_j^{cy} = (\hat{g}_j^{cy} - d_i^{cy})/d_i^{h} \tag{3.4}$$

$$\hat{g}_j^{w} = log(\frac{g_j^{w}}{d_j^{w}}) \tag{3.5}$$

$$\hat{g}_j^{h} = log(\frac{g_j^{h}}{d_j^{h}}) \tag{3.6}$$

The confidence loss is softmax loss over multiple class confidences(c) (since more than one class can exist in a box).

$$L_{conf}(x,c) = -\sum_{i\in Pos}^{N} x_{ij}^{p} log(\hat{c}_i^p) - \sum_{i\in Neg} log(\hat{c}_i^0) \tag{3.7}$$

$$\text{where}\, \hat{c}_i^p = \frac{exp(c_i^p)}{\Sigma_p exp(c_i^p)} \tag{3.8}$$

and α was set to 1 after cross-validation. This modified loss function, along with carefully chosen scales and aspect ratio for default boxes, hard negative mining and data augmentation using randomly sampled patches with minimum jacquard overlaps of varying thresholds are one of the major drivers behind this increase in accuracy.

SSD performs at 77.2% mAP for 300x300 input at 59FPS, and 79.8% mAP for 512x512 input on VOC2007. We further train SSD (for 300x300 input) on Imagenet DET + VID and we achieve an mAP of 71% at a threshold of 0.5.

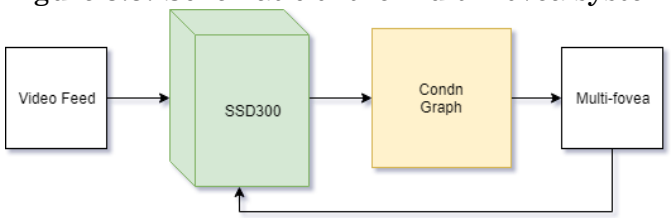
3.2.2 Multi-Fovea Layer

Algorithm 1 Multi-Fovea

procedure MFOVEA($frame, dbox, nObj$) ▷ Multi-fovea from a frame
  $maxObj \leftarrow 6$
  $z \leftarrow 0.1$
  if $nObj \leq 2$ **then** ▷ Defining fovea size
    $scale = 0.25$
  else if $nObj \leq 4$ **then**
    $scale = 0.5$
  else
    $scale = 1$
  if $nObj \leq maxObj$ **then** ▷ Breaking point for trade off
    $dboxNew \leftarrow stretch(dbox, z)$ ▷ Stretches bounding box by z
    $fovea \leftarrow montage(frame, dboxNew, scale)$ ▷ Bin-pack all detections
  else
    $fovea \leftarrow frame$
  return $fovea$

In this section, we describe the Multi-Fovea layer used on top of the SSD model. In order to give objects more importance, track only the detected boxes,

Figure 3.5: Schematic of the Multi-Fovea system



while occasionally checking the whole image to correct for drifts. Multi-Fovea consists of two processes - selecting frame to process, making the multi-fovea, and processing the fovea.

The first frame of the video is always processed, and if objects are detected in the first frame, then only every other $skip\_frame\_o = 5$ -th frame is processed. This is to make use of the temporal redundancy in the video,and if no objects are detected, then every other frame ($skip\_frame\_n = 2$) is processed.

To make the montage, we look at the number of detections in a video frame, for each detection, we add an allowance to the bounding box, to account for object movement in the next frame. This allowance (z) was chosen to be 0.1 after estimating average object velocities on the Imagenet-VID dataset. z needs to be higher for datasets with high velocity objects, and vice versa. After stretching the bounding boxes by this factor z, and cropping the relevant portions of the image, we bin-pack them into a rectangular montage, where each detection is resized into a square, with dimensions $scale * min(frame_{width}, frame_{height})$. The montage is only made if there are atmost $maxObj = 6$ objects in the frame, this number was chosen from the fact that the VID dataset on an average has 2/3 objects, and 95% of the time has less than 7 objects in the frame. The algorithm to make the multi-fovea is succinctly stated in 1

After the montage/multi-fovea is made, we record the transformations made to each of the initial detected boxes, through matrices for each detection. If the fovea happens to be the complete frame itself, then the matrices are set to the identity matrix. We now run SSD on this multi-fovea. Owing to variable size of the fovea, we appropriately run SSD upto the final layer or the penultimate layer accordingly. Next,for the detections we first run Non-Maximum Suppression and then run the inverse of the transforms to get their locations in the original frame.

(a) (b)

Figure 3.6: Sample working of the Multi-Fovea system

Table 3.2: *
Classwise mAP on ImageNet-VID

Class	Airplane	Bicycle	Bird	Bus	Car	Cow	Dog	Cat	Horse	Motorcycle	Sheep	Train	mAP
mAP(Baseline)	0.768	0.681	0.706	0.770	0.689	0.728	0.755	0.660	0.818	0.678	0.517	0.763	0.711
mAP(MultiFovea)	0.568	0.457	0.589	0.544	0.485	0.555	0.576	0.581	0.599	0.450	0.369	0.655	0.536

In order for the multi-fovea system to work better with smaller resolution images, we did an extra round of training with scaled down images from Imagenet-DET and ImageNet-VID-train

3.3 Results

SSD300 trained on Pascal VOC was taken and was further trained on ImageNet-DET and ImageNet-VID-Train. Tesla K80 was used for training and testing.

First, we report results on static object detection by both Multi-Fovea system and Baseline-SSD (without our round of training), and tested on imagenet-vid. Only common classes are reported.

As seen in 3.3, the baseline network outperforms the multi-fovea approach, but takes twice as much time to operate. Upon visual inspection of results, most negatives seemed to occur for small object sizes, which makes our extra round of training inevitable for this model. After training, the baseline SSD improves to an mAP of 0.71, at 0.5 threshold, and the multi-fovea model achieves an mAP 0.64 at 0.5 threshold.

$$s = \frac{R_{\text{feed}} \times \text{skip\_frame\_o}}{R_{\text{fovea}}} \tag{3.9}$$

This approach gives hard attention to the objects, and ensures lesser compute time, while actively ignoring the areas without objects. The speed up for any frame, s, is given by 3.9, where R_{feed} is resolution of input feed, which in our case is 300x300, R_{fovea} is resolution of the fovea, which for 6 objects is 300x300, and for 2 objects is 75x75. On an average, with 2-3 objects occurring in most frames, speedup is 2x.

3.4 Limitations and Future work

The proposed system works well owing to the adaptability of the underlying SSD network. Despite our proposal being modular, it cannot be expected to work at the same level for other neural network baselines.

Even with SSD, we see that there's a considerable gap to close in terms of accuracy between the baseline and our trained model. This motivates us to search for alternate ways of formulating the fovea, unlike our current model, where each detection in the fovea is resized to the same size, irrespective of its original size in the network.

Our method is not effective against high velocity objects, owing to the choice of factor z, choosing a large value for z doesn't make enough sense in terms of speedup. An interesting direction to look into to solve this problem is to choose z for each detection based on its velocity, this would be akin to calculating optical flow at specific regions in the image, and would be a good compromise between speed and accuracy, since winning entries of ImageNet-VID use Flow-based approaches.

Other video related problems include occlusion, defocus blur and motion blur, our method copes well with occlusion, but is resistant to some degree of blur owing to the frame-skip mechanism. This is acceptable since blur artifacts don't last long in a video.

This fovea-based model maybe extended to other video processing applications, as its well suited for low-power applications, such as surveillance, monitoring and anomaly detection, etc.

CHAPTER 4

Hardware Trade-offs

Any machine learning system has two major parts : the software/algorithms, and the hardware package it runs on. The preceding chapters dealt with improving algorithms independent of the other components of a machine learning system. In this chapter we explore trade-offs associated with various hardware platforms used to run DNNs. Experimental Simulations are carried out on the CPU as well as the memory chip.

Our approach delivers key insight into various problems posed by standard training and inference algorithms for common neural network architectures, and provides solutions to address them.

These experiments detail the fact that design of algorithms and hardware for machine learning should go hand-in-hand to ensure maximum efficiency.

4.1 Related Work

We look at prior work on injecting noise in Machine Learning models. Noise has mostly been a welcome addition in this area, since (Murray and Edwards, 1993) established that using additive noise in multi-layer perceptron improves generalizing performance of a neural network. And more recently, there's been a flurry of works in reduced precision weights in DNNs, popular examples being BinaryNet, with all weights being binary (Courbariaux and Bengio, 2016); extending it to TernaryNet (Alemdar *et al.*, 2016). These lead us to investigate what makes neural network architectures work with less precision, and yet fail when significant noise is added, found by (Merolla *et al.*, 2016).

Similarly, we've also seen experiments that injecting noise into inputs often cause networks to fail inconsiderately, these so called "adversarial" inputs have been a cause for concern, and (Zheng *et al.*, 2016) addresses this to some extent.

Noise added may be seen as originating from fault in hardware processes. Examples include quantization noise, where weights are reduced in precision owing the fixed width; binary noise, which occurs due to incorrect memory reads. Quantization noise can be simulated on two levels : one in the activations and inputs, and another in the neural network weights. We note that both are equivalent, since perturbation in activations can be seen as perturbation of weights in the previous layer, or as perturbation of inputs, with correspondingly scaled parameters. Typically, only weights and weight updates are quantized, owing to their long term presence, while activations are computed in higher precision.

4.2 Approach and Results

In this chapter, we define our experiments, and compare them against standard baselines to look at their resilience.

4.2.1 DNN Profiling experiments on Raspberry Pi

Raspberry Pi3 is a popular ARMv7 based board that runs Raspbian, a debian based linux distribution. We choose Raspberry Pi owing to its widespread presence, and reasonable cost. The CPU/GPU is one of the key components in an ML system, and consumes considerable amount of power. Modern GPUs such as TitanX, TitanV consume upto 250Watts, while modern server grade CPUs such as Xeon, i9 series report a Thermal Design Power of around 200Watts at most. RaspberryPi3's CPU, a Broadcom 2837, scales from 600MHz to 1.2GHz (without overclocking), and CPU voltage can stretch from 0.8V to 1.4V, default value : 1.2V. The chipset reports an idle power of 1.83W, and max power of 6.7W under stress, as reported in (Team, 2016) We were only able to vary the CPU clock speed in a stable fashion, voltage was locked in the range 1.2V-1.3V, owing to kernel restrictions.

Experiment Details

1. CPU clock speed varied from 600MHz to 1.2GHz

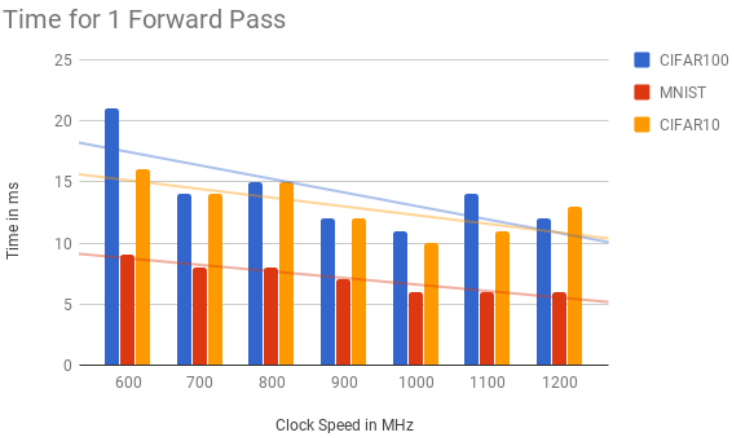


Figure 4.1: Inference time in ms

2. Keras system used for benchmarking
3. Inference times tested on MNIST (Le-Net), CIFAR-10, CIFAR-100 architectures

We report the inference times in 4.2.1.The inference times decrease with increase in clock speed as expected, and saturate beyond a point owing to time taken by other parts of the pipeline such as memory access, context switching etc. We now calculate the power consumed to see if there is a convex trend. In order to calculate power consumed, we use the approach followed in (Vogeleer *et al.*, 2014), as follows.

$$P_{cpu} = P_{dynamic} + P_{leak} \tag{4.1}$$

$$P_{dynamic} = P_{short} + P_{charge} \tag{4.2}$$

$$P_{charge} = \alpha C f V^2 \tag{4.3}$$

$$P_{short} = (\eta - 1) P_{charge} \tag{4.4}$$

$$P_{dynamic} = \eta \alpha C f V^2 \tag{4.5}$$

$$E_{dynamic}(t) = \int_0^t \eta \alpha C f V^2 \tag{4.6}$$

The Power consumed by CPU, may be divided into P_{leak}, arising out of leakage currents in transistors, and $P_{dynamic}$, owing to varying load across the transistors. This dynamic power may be further split into P_{short}, the power lost when the transistors conduct current (are in "on" mode), and P_{charge}, power needed to

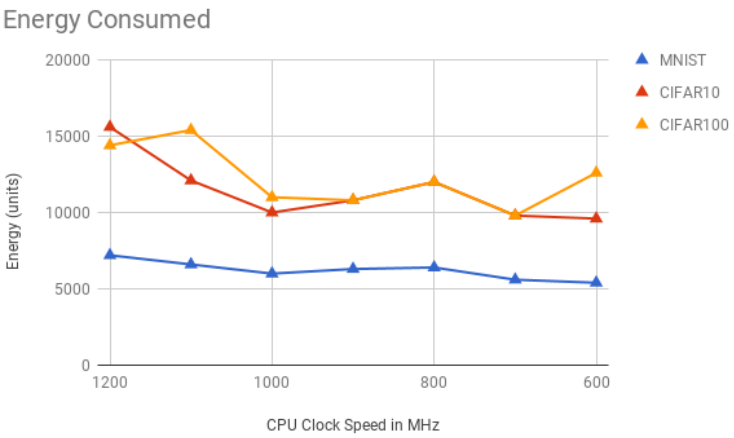


Figure 4.2: Energy consumed

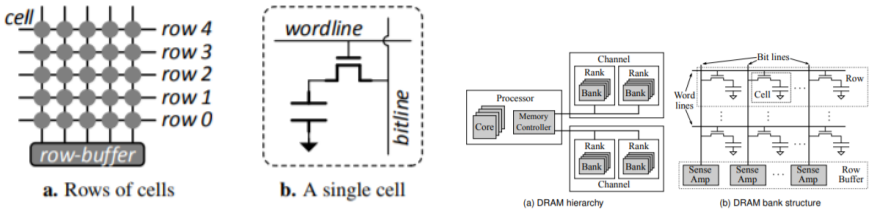
charge the gate's capacitors.In standard literature, P_{charge} is defined as $\alpha C f V^2$ (Eshraghian and Weste, 2000), where α is a proportionality constant indicating the percentage of time the system is active/switching, C is the system capacitance, and f is the clock frequency, with V being the voltage swing across C. P_{short} arises when a logic gate is toggled, and a short circuit current flows from Vcc to ground, this is non-negligible for billions of gates, and very high clock frequencies. We deem P_{short} to be proportional to P_{charge} for simplicity, and with η as scaling factor we obtain an expression for $P_{dynamic}$, as in 4.1

4.2.1 shows a plot of energy consumed with clock frequency. We see that for MNIST there is linear trend in energy consumed, owing to the small size of the network, memory accesses dominate the time taken, and hence energy increases with frequency.

While, for CIFAR networks, we see that there is a convex behavior, as hypothesized. Energy attains a minimum at both 1GHz and 600MHz, with inference time being lower at 1GHz, meaning the 1GHz parameter occupies a sweet spot giving us faster inference at reduced power.

4.2.2 Retention Time analysis for DNN weights

The weights of a DNN are typically too large to fit on the CPU/GPU cache SRAM, (AlexNet: 200MB, CIFAR-10 : 30 MB, and so on), and hence they are usually stored on the shared DRAM. SRAM (Static RAM) uses CMOS transistor based



(a) (b)

Figure 4.3: Structure & Hierarchy of DRAM

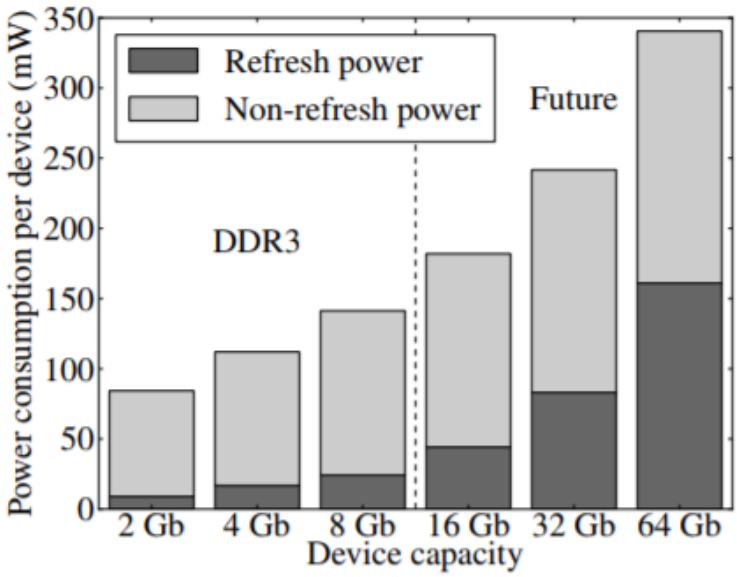


Figure 4.4: Power Consumption of DRAM Chips vs memory

flip-flop to store memory, as long as Vcc is kept within limits, and hence requires very less power. Owing to the complexity involved in making SRAM, it is used only where extremely fast access memory is required in tiny quantities(CPU/GPU caches), as depicted in 4.2.2, as it is very expensive to manufacture, since it requires 6 transistors per cell, compared to one transistor and one capacitor needed by DRAM.

DRAM (Dynamic RAM), 4.2.2 uses a capacitor to store charge, and a transistor to change the value. Since the capacitor loses charge over time, it needs periodic refreshing. This refresh command issued by the DRAM controller is not negligible in terms of power consumption, as shown in 4.2.2 (Liu *et al.*, 2012). A refresh command is issued each 7.8μs, and each cell is refreshed every 64ms according to the DDR3 DRAM standard. Profiling retention times in DDR3 RAM cells, as done in (Liu *et al.*, 2012) yields the following graph 4.2.2

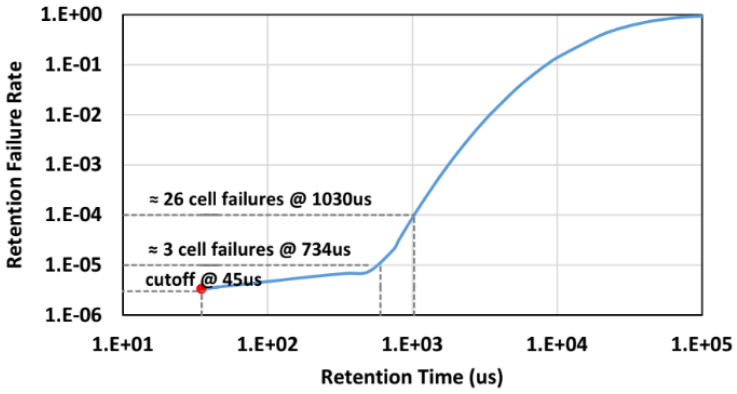


Figure 4.5: Retention time of DRAM Cells

Algorithm 2 Retention Time

Require: weights, weights of the DNN
Require: p_retain, retention probability

1: **procedure** MASKWEIGHT($weights$,$width$,p_retain) ▷ Bitwise-masking of weights
2: width ← 16 ▷ Quantizing weights to 16bits fixed point
3: weights ← quantize(weights,width)
4: masks ← generate_masks(width) ▷ Generates all possible masks for a given width
5: **for each** weight ∈ weights **do**
6: mask ← sample(weight,masks,p_retain)
7: weight ← weight & mask
return weights

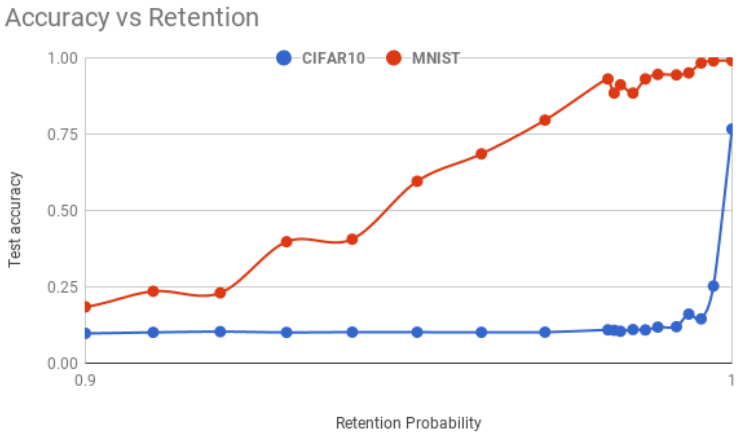
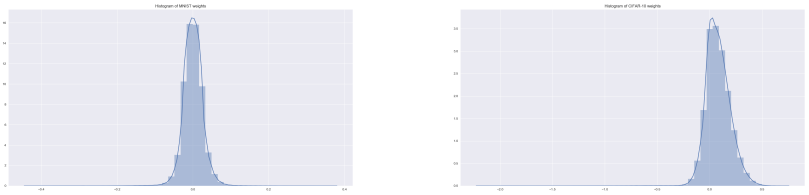


Figure 4.6: Accuracy vs Retention probability

We now wish to explore the effect of removing refresh on DRAM chips, in case of DNNs. We specifically explore their effect on neural network weights, since they are persist for a longer period of time, and activations are used almost immediately in the successive layers, so activation data is no longer needed after access, and hence, they can stay "unrefreshed". Since regular neural network weights are floating point variables, and cannot be subject to bitwise operations, we convert them into 16bit fixed point variables for this experiment. We perform bitwise-and on each weight with a randomly chosen mask. These masks correspond to normal operation (viz, 0x111111 ..) or any of the charged bits losing their charge (viz, 0x101111, 0x11011 ..) owing to exceeding the retention time. Since weights are usually uniformly distributed around zero, we apply masks independent of the weight value, however, we could also sample only applicable masks for each weight (viz, for a weight value of 1 say, the only applicable masks are 0x1111 ... and 0x11111111..0 and then do the aforesaid bitwise-and), both methods yielded similar results on CIFAR-10 and MNIST architectures. p_retain was varied over 0 to 1. 1-sec retention corresponds to a p_retain of 0.99 in DDR3 RAM.

We see that accuracy drops sharply as we cross 2% failure rate, this could occur because a lot of important bits begin to be erred and that affects the final output. We see that the curve is much sharper for CIFAR-10 as compared to MNIST, this is because of the fact that many weights in MNIST are very close to zero, and failures in such bits will have almost no effect on the accuracy, and hence effects are only felt at much higher failure rates.



(a) (b)

Figure 4.7: (a)MNIST and (b)CIFAR-10 weight distributions

This experiment considers the event where only one bit per weight fails, since anything more than that would be half as likely.

4.2.3 Resilience to normal noise in DNN weights

In this section, we try to expose normal noise to DNN weights, and see the effect on accuracy. We observe that adding normal noise while training to the inputs, gives rise to an L2 like regularization on the weights. For a linear regression case this can be derived as 4.7. Let $y \sim \Sigma w_i x_i$, for inputs x_i, and let $\epsilon_i \sim N(0, \sigma_i^2)$, with ϵ_i independent. Let y_{noise} be outcome of adding noise ϵ_i to input vector x_i. Since y is uncorrelated to ϵ_i, the last step follows from the penultimate step.

$$y_{noise} = \Sigma w_i x_i + \Sigma \epsilon_i w_i \tag{4.7}$$

Let y0 be actual value of y, SSE is given by (4.8)

$$E((y_{noisy} - y_0)^2) = E((\Sigma w_i x_i + \Sigma \epsilon_i w_i - y_0)^2) \tag{4.9}$$

$$= (y-t)^2 + 2E((y-t)\Sigma \epsilon_i w_i) + E((\Sigma \epsilon_i w_i)^2) \tag{4.10}$$

$$= (y-t)^2 + E((\Sigma \epsilon_i w_i)^2) \tag{4.11}$$

$$= (y-t)^2 + \Sigma w_i^2 \sigma_i^2 \tag{4.12}$$

$$\tag{4.13}$$

A quick look at the distribution of weights of both MNIST and CIFAR-10 architectures, shows that MNIST weights are almost normally distributed around zero, while CIFAR-10 weights are a little shifted to the right, and span a wider range.

(Merolla *et al.*, 2016) describes a similar approach where they add constant normal noise to each weight in the network. Here, we perform the following experiments on the MNIST, and the CIFAR-10 standard architectures.

1. Add normal noise to all weights, with zero mean, and std deviation proportional to the average weight (denoted by global) 2. For weights in each layer, add normal noise with zero mean, std deviation proportional to average layer weight (denoted by layer) 3. For weights in each layer, add normal noise with zero mean and std deviation proportional to the value of the weight itself. (denoted by indiv)

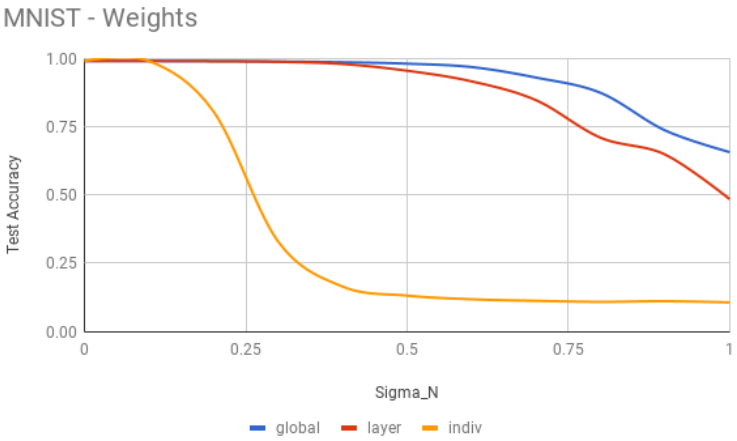


Figure 4.8: Normal Noise added to MNIST architecture

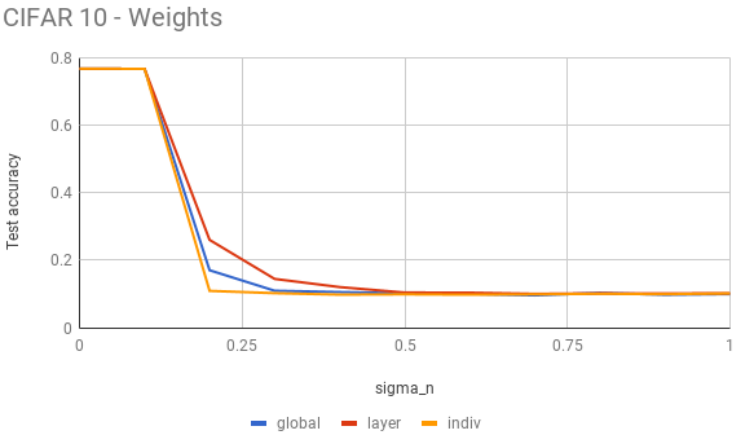


Figure 4.9: Normal Noise added to cifar-10 architecture

Results for MNIST 4.2.3 and CIFAR-10 4.2.3 are attached as above. We see that for σ_n less than around 10% of the original weight value, we do not see any significant change in test-error rate. This approximately gives us the depth of the minima of the loss function with respect to each weight. As σ_n increases, accuracy

drops sharply to that of random selection, this is more pronounced in "indiv" than in the other two, this is because the random errors added to each weight are of different variance, and the model does not have enough redundancy to deal with this. Specifically in CIFAR-10 we see that the "layer" variant drops slowly than the other two, this is most likely because of the fact that all layer weight averages are quite close to one another, and tend to behave like the "global" variant.

4.3 Conclusion

1. Profiling of DNNs on various platforms helps us find a sweet-spot configuration to balance execution speed and reduce power consumed, this will be helpful tuning hardware for always-on machine learning models
2. Retention analysis lead us to propose a two-tiered memory model for DNNs : one tier will be exclusively used for activations, gradients and other temporary variables, and this tier will have a lower refresh rate than the other tier, which will contain weights of various layers.
3. Analysis of weights perturbed with normal noise indicates a very complex error surface. We've identified the extents of safe zone around each weight, and this could be used to build more efficient training/inference algorithms.

Deep Neural Networks are have miles to go before they match the efficiency and accuracy of the final frontier : the brain. Statements in the media have often ridiculed Deep Learning to be the latest cause of global warming owing to excessive energy spent in training and inference. When machine learning algorithms have potential applications almost everywhere, our work is a reminder that not only do algorithms need to be accurate, but even the hardware that is used for DL, must be co-designed to maximize potential gains from efficiency and accuracy.

REFERENCES

1. **Alemdar, H.**, **N. Caldwell**, **V. Leroy**, **A. Prost-Boucle**, and **F. Pétrot** (2016). Ternary neural networks for resource-efficient AI applications. *CoRR*, **abs/1609.00222**. URL http://arxiv.org/abs/1609.00222.

2. **Anwar, S.**, **K. Hwang**, and **W. Sung**, Fixed point optimization of deep convolutional neural networks for object recognition. *In Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015.

3. **Ba, J.** and **B. Frey**, Adaptive dropout for training deep neural networks. *In Advances in Neural Information Processing Systems*. 2013.

4. **Bae, S.-H.** and **K.-J. Yoon**, Robust online multi-object tracking based on tracklet confidence and online discriminative appearance learning. *In Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014.

5. **Bengio, E.**, **P. Bacon**, **J. Pineau**, and **D. Precup** (2015). Conditional computation in neural networks for faster models. *CoRR*, **abs/1511.06297**. URL http://arxiv.org/abs/1511.06297.

6. **Bengio, Y.** (2013). Estimating or propagating gradients through stochastic neurons. *CoRR*, **abs/1305.2982**. URL http://arxiv.org/abs/1305.2982.

7. **BVLC** (). Caffe model zoo. URL http://caffe.berkeleyvision.org/model_zoo.html.

8. **Chen, W.**, **J. T. Wilson**, **S. Tyree**, **K. Q. Weinberger**, and **Y. Chen** (2015). Compressing neural networks with the hashing trick. *CoRR*, **abs/1504.04788**. URL http://arxiv.org/abs/1504.04788.

9. **Courbariaux, M.** and **Y. Bengio** (2016). Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, **abs/1602.02830**. URL http://arxiv.org/abs/1602.02830.

10. **Das, D.**, **S. Avancha**, **D. Mudigere**, **K. Vaidyanathan**, **S. Sridharan**, **D. D. Kalamkar**, **B. Kaul**, and **P. Dubey** (2016). Distributed deep learning using synchronous stochastic gradient descent. *CoRR*, **abs/1602.06709**. URL http://arxiv.org/abs/1602.06709.

11. **Denton, E.**, **W. Zaremba**, **J. Bruna**, **Y. LeCun**, and **R. Fergus** (2014). Exploiting linear structure within convolutional networks for efficient evaluation. *CoRR*, **abs/1404.0736**. URL http://arxiv.org/abs/1404.0736.

12. **Donahue, J.**, **L. Anne Hendricks**, **S. Guadarrama**, **M. Rohrbach**, **S. Venugopalan**, **K. Saenko**, and **T. Darrell**, Long-term recurrent convolutional networks for visual recognition and description. *In Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.

13. **Eshraghian, K.** and **N. Weste**, *Principles of CMOS VLSI design: a systems perspective*. Addison-Wesley Pub. Co., 2000.

14. **Farabet, C.**, **B. Martini**, **B. Corda**, **P. Akselrod**, **E. Culurciello**, and **Y. LeCun**, Neuflow: A runtime reconfigurable dataflow processor for vision. *In Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*. IEEE, 2011.

15. **Figurnov, M.**, **D. P. Vetrov**, and **P. Kohli** (2015). Perforatedcnns: Acceleration through elimination of redundant convolutions. *CoRR*, **abs/1504.08362**. URL `http://arxiv.org/abs/1504.08362`.

16. **Ganapathy, S.**, **S. Venkataramani**, **B. Ravindran**, and **A. Raghunathan** (2017). Dyvedeep: Dynamic variable effort deep neural networks. *CoRR*, **abs/1704.01137**. URL `http://arxiv.org/abs/1704.01137`.

17. **Girshick, R.**, Fast r-cnn. *In International Conference on Computer Vision (ICCV)*. 2015.

18. **Girshick, R.**, **J. Donahue**, **T. Darrell**, and **J. Malik**, Rich feature hierarchies for accurate object detection and semantic segmentation. *In Computer Vision and Pattern Recognition*. 2014.

19. **Han, S.**, **X. Liu**, **H. Mao**, **J. Pu**, **A. Pedram**, **M. A. Horowitz**, and **W. J. Dally**, Eie: efficient inference engine on compressed deep neural network. *In Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 2016.

20. **Han, S.**, **H. Mao**, and **W. J. Dally** (2015*a*). Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, **abs/1510.00149**. URL `http://arxiv.org/abs/1510.00149`.

21. **Han, S.**, **J. Pool**, **J. Tran**, and **W. J. Dally** (2015*b*). Learning both weights and connections for efficient neural networks. *CoRR*, **abs/1506.02626**. URL `http://arxiv.org/abs/1506.02626`.

22. **Jaderberg, M.**, **A. Vedaldi**, and **A. Zisserman**, Speeding up convolutional neural networks with low rank expansions. *In Proceedings of the British Machine Vision Conference*. BMVA Press, 2014.

23. **Jia, Y.**, **E. Shelhamer**, **J. Donahue**, **S. Karayev**, **J. Long**, **R. Girshick**, **S. Guadarrama**, and **T. Darrell** (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*.

24. **Kang, K.**, **W. Ouyang**, **H. Li**, and **X. Wang** (2016). Object detection from video tubelets with convolutional neural networks. *CoRR*, **abs/1604.04053**. URL `http://arxiv.org/abs/1604.04053`.

25. **Karpathy, A.**, **G. Toderici**, **S. Shetty**, **T. Leung**, **R. Sukthankar**, and **L. Fei-Fei**, Large-scale video classification with convolutional neural networks. *In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2014.

26. **Krizhevsky, A.** (2014). One weird trick for parallelizing convolutional neural networks. *CoRR*, **abs/1404.5997**. URL http://arxiv.org/abs/1404.5997.

27. **Krizhevsky, A.**, **I. Sutskever**, and **G. E. Hinton**, Imagenet classification with deep convolutional neural networks. *In Advances in neural information processing systems*. 2012.

28. **Larochelle, H.** and **G. E. Hinton**, Learning to combine foveal glimpses with a third-order boltzmann machine. *In Advances in neural information processing systems*. 2010.

29. **Leal-Taixé, L.**, **M. Fenzi**, **A. Kuznetsova**, **B. Rosenhahn**, and **S. Savarese**, Learning an image-based motion context for multiple people tracking. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014.

30. **LeCun, Y.**, **J. S. Denker**, and **S. A. Solla**, Optimal brain damage. *In Advances in neural information processing systems*. 1990.

31. **Lin, T.**, **P. Dollár**, **R. B. Girshick**, **K. He**, **B. Hariharan**, and **S. J. Belongie** (2016). Feature pyramid networks for object detection. *CoRR*, **abs/1612.03144**. URL http://arxiv.org/abs/1612.03144.

32. **Lin, T.**, **P. Goyal**, **R. B. Girshick**, **K. He**, and **P. Dollár** (2017). Focal loss for dense object detection. *CoRR*, **abs/1708.02002**. URL http://arxiv.org/abs/1708.02002.

33. **Liu, B.**, **M. Wang**, **H. Foroosh**, **M. Tappen**, and **M. Pensky**, Sparse convolutional neural networks. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015*a*.

34. **Liu, J.**, **B. Jaiyen**, **R. Veras**, and **O. Mutlu**, Raidr: Retention-aware intelligent dram refresh. *In ACM SIGARCH Computer Architecture News*, volume 40. IEEE Computer Society, 2012.

35. **Liu, W.**, **D. Anguelov**, **D. Erhan**, **C. Szegedy**, **S. E. Reed**, **C. Fu**, and **A. C. Berg** (2015*b*). SSD: single shot multibox detector. *CoRR*, **abs/1512.02325**. URL http://arxiv.org/abs/1512.02325.

36. **Merolla, P.**, **R. Appuswamy**, **J. Arthur**, **S. K. Esser**, and **D. Modha** (2016). Deep neural networks are robust to weight binarization and other non-linear distortions. *arXiv preprint arXiv:1606.01981*.

37. **Modha, D. S.** (2017). Introducing a brain-inspired computer. *Published online at IBM Research articles*. URL http://www.research.ibm.com/articles/brain-chip.shtml.

38. **Murray, A. F.** and **P. J. Edwards**, Synaptic weight noise during mlp learning enhances fault-tolerance, generalization and learning trajectory. *In Advances in neural information processing systems*. 1993.

39. **Nam, H.** and **B. Han** (2015). Learning multi-domain convolutional neural networks for visual tracking. *CoRR*, **abs/1510.07945**. URL http://arxiv.org/abs/1510.07945.

40. **Papert, S.** (1966). The summer vision project. URL http://dspace.mit.edu/bitstream/handle/1721.1/6125/AIM-100.pdf.

41. **Ramasubramanian, S. G.**, **R. Venkatesan**, **M. Sharad**, **K. Roy**, and **A. Raghunathan**, Spindle: Spintronic deep learning engine for large-scale neuromorphic computing. *In Proceedings of the 2014 International Symposium on Low Power Electronics and Design*, ISLPED '14. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-2975-0. URL http://doi.acm.org/10.1145/2627369.2627625.

42. **Redmon, J.** and **A. Farhadi** (2018). Yolov3: An incremental improvement. *arXiv*.

43. **Rosenblatt, F.** (1957). The perceptron–a perceiving and recognizing automaton.

44. **Rumelhart, D. E.**, **G. E. Hinton**, and **R. J. Williams** (1986). Learning representations by back-propagating errors. *nature*, **323**(6088), 533.

45. **Russakovsky, O.**, **J. Deng**, **H. Su**, **J. Krause**, **S. Satheesh**, **S. Ma**, **Z. Huang**, **A. Karpathy**, **A. Khosla**, **M. Bernstein**, **A. C. Berg**, and **L. Fei-Fei** (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, **115**(3), 211–252.

46. **Sandberg, A.** (2008). Whole brain emulation: A roadmap, technical report. (3). URL www.fhi.ox.ac.uk/reports/2008âĂŘ3.pdf.

47. **Seide, F.**, **H. Fu**, **J. Droppo**, **G. Li**, and **D. Yu**, 1-bit stochastic gradient descent and application to data-parallel distributed training of speech dnns. *In Interspeech 2014*. 2014.

48. **Shaoqing Ren, K. H.**, **R. Girshick**, and **J. Sun** (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*.

49. **Silver, D.**, **J. Schrittwieser**, **K. Simonyan**, **I. Antonoglou**, **A. Huang**, **A. Guez**, **T. Hubert**, **L. Baker**, **M. Lai**, **A. Bolton**, **Y. Chen**, **T. Lillicrap**, **F. Hui**, **L. Sifre**, **G. v. d. Driessche**, **T. Graepel**, and **D. Hassabis** (2017). Mastering the game of go without human knowledge. *Nature*, **550**(7676), 354. ISSN 1476-4687. URL http:https://doi.org/10.1038/nature24270.

50. **Simonyan, K.** and **A. Zisserman** (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, **abs/1409.1556**. URL http://arxiv.org/abs/1409.1556.

51. **Team, R. P.** (2016). URL https://www.raspberrypi.org/documentation/.

52. **Uijlings, J.**, **K. van de Sande**, **T. Gevers**, and **A.W.M.Smeulders** (2013). Selective search for object recognition. *International Journal of Computer Vision*. URL http://www.huppelen.nl/publications/selectiveSearchDraft.pdf.

53. **Venkataramani, S.**, **A. Ranjan**, **K. Roy**, and **A. Raghunathan**, Axnn: energy-efficient neuromorphic systems using approximate computing. *In Low Power Electronics and Design (ISLPED), 2014 IEEE/ACM International Symposium on*. IEEE, 2014.

54. **Vogeleer, K. D.**, **G. Memmi**, **P. Jouvelot**, and **F. Coelho** (2014). The energy/frequency convexity rule: Modeling and experimental validation on mobile devices. *CoRR*, **abs/1401.4655**. URL `http://arxiv.org/abs/1401.4655`.

55. **Zheng, S.**, **Y. Song**, **T. Leung**, and **I. J. Goodfellow** (2016). Improving the robustness of deep neural networks via stability training. *CoRR*, **abs/1604.04326**. URL `http://arxiv.org/abs/1604.04326`.

56. **Zhu, X.**, **Y. Wang**, **J. Dai**, **L. Yuan**, and **Y. Wei**, Flow-guided feature aggregation for video object detection. 2017.