Inference at the edge: the impact of compression on performance

Deliverable 1: Final year Dissertation

Bsc Computer Science: Artificial Intelligence

Sam Fay-Hunt — `sf52@hw.ac.uk`

Supervisor: Rob Stewart — `R.Stewart@hw.ac.uk`

November 15, 2020

DECLARATION

I, Sam Fay-Hunt confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: ....

Date: ....

Abstract: a short description of the project and the main work to be carried out – probably between one and two hundred words

Contents

1 Introduction

Summarising the context of the dissertation project, stating the aim and objectives of the project, identifying the problems to be solved to achieve the objectives, and sketching the organisation of the dissertation.

With the revolution of AI technologies a greater need to perform inference at the edge is becoming ever more prevalent. The argument for localising inference is only becoming stronger with the ever increasing avaliablilty of computation power alongside new and constatnly evolving AI applications, inference at the edge can provide better privacy and latency than the remote datacenter alternatives. This dissertation will focus on methodologys for improving inference performance with preexisting compression techniques.

These models can have a huge number of parameters so inference can sometimes be impractical. [1] - "see Table 1"

Issues with limited resource computation [2]

This dissertation will study the effect of pruning algorithms exposed by the Intel distiller framework on inference.

outline the document: We start with ..., then we cover x, y, and z ...

2 Background

Discussing related work found in the technical literature and its relevance to your project. This Section will be split into 4 subsections:

Section 2.1 - **Deep Learning**: An overview of the basic components of a neural network and the CNN & RNN models.

Section 2.2 - **Compression Types**: ...

Section 2.3 - **AI accelerators** stuff about AI accelerators

Section 2.4 - **Memory factors for Deep Neural Networks**: brief stuff about this section

2.1 Deep Neural Networks

2.1.1 Neural Networks & Deep Learning

- *Summary of NN*
- *Structure of NN*
- *Training & Inference stages*
- *weight update methodologies*
- *Feed Forwards*
- *Feedback Nerual Network*
- *Self-organizing Neural Network*
- *Weight parameters updated using back-propagation*

Deep learning is a subcategory of machine learning techniques where a hierarchy of layers perform some manner of information processsing with the goal of computing high level abstractions of the data by utilising low level abstractions identified in the early layers [3].

Neural networks fundamental purpose is to transform an input vector commonly referred to as X into an output vector $\hat{Y}$. The output vector $\hat{Y}$ is some form of classification such as a binary classification or a probability distribution over multiple classes [4]. Between the input layer (X) and the output layer ($\hat{Y}$) there exists some number of interior layers that are referred to as hidden layers, the hidden and output layers are composed of neurons that pass signals derived from weights through the network, this model of computing was inspired by connectionism and our understanding of the human brain, see Fig. 1 for labels of the analagous biological components.

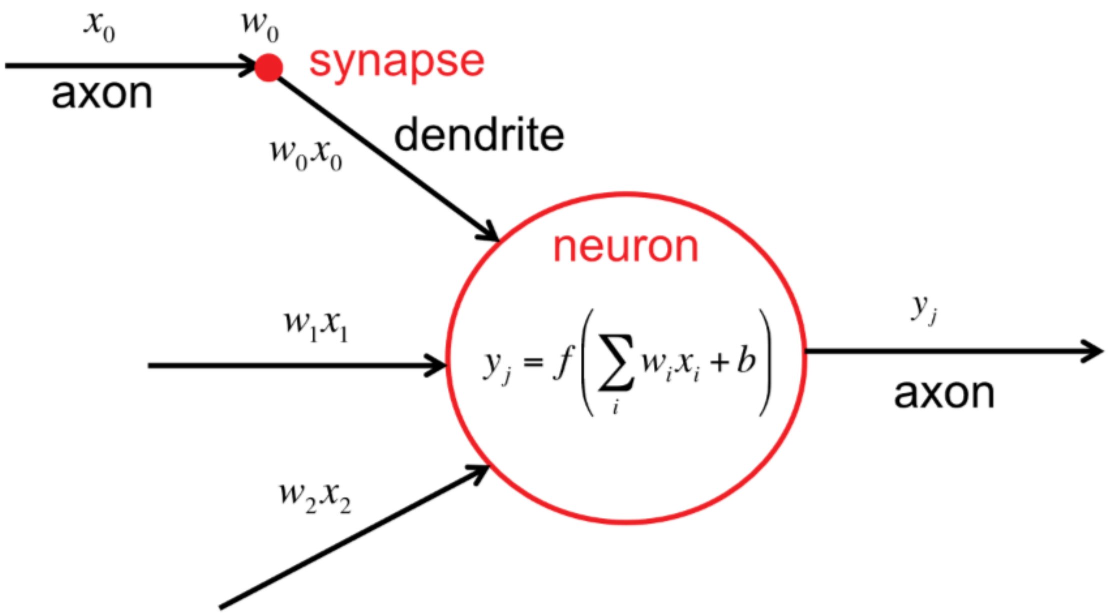

Figure 1: Neuron with corresponding biologically inspired labels.
(Adopted figure from [2])

Weights in a neural network effectively correspond to the synapses in the brain and the output of the neruon is modelled as the axon. All neruons in a Neural network have weights corresponding to their inputs, these weights are are intended to mirror the value scaling effect of a synapse by performing a weighted sum operation [2].

Neural networks and deep neural networks are often reffered to interchangably, they are primarily distinguished by the number of layers, there is no hard rule indicating when a neural network is considered deep but generally a network with more than 3 hidden layers is considered a deep neural network, the rest of this dissertaion will refer to DNNs for consistency. Each neuron in a DNN applies an non-linear activation function to the result of its weighted sum of inputs and weights, without which a DNN would just be a linear algebra operation [2], the cumulative effect of the activations in each layer results in elabourate causal chains of transormations than influence the aggregate activation of the network.

Backpropagation Although not the first to propose this approach [5] the 1986 paper Learning representations by back-propagating errors [6] popularised back-propagation for updating weights during training multi-layer networks.

DNNs can be categorised as feedforwards, feedback, and self-organising networks depending on their processing method [1].

There are many popular deep neural network architectures, this document will continue to outline the CNN & RNN architectures because these provide a high level overview of the type of models that will be used for the research posed in this dissertation.

2.1.2 Convolutional Neural Networks

Convolutional Neural Network (CNN)

- A class of DNN

- CNN consist of: Convolutional Layers, Pooling layers & fully connected layers.

- Convolutional Layers contain sets of filters/kernels

- Should emphesize the computation requirements in conv layers & the memory access requirements in FC layers (see page 28 [7])

Much like traditional nerual networks the CNN architecture was inspired by human and animal brains, the concept of processing the input with local receptive fields is conceptually similar some functionality of the cat's visual cortex [8]–[10]. The influential paper by Hubel & Weisel [8] ultimately had a significant influence on the design of CNNs via the Neocognitron, as proposed by Fukushima in [11] and again evaluated in [12]**(provide some comment on these papers)**.

A critical aspect of image recognition is robustness to input shift and distortion, this robustness was indicated as one of the primary achivements of the Neocognitron in Fukushima's paper [11]. LeCunn and Bengio provide comprehensive explainations of how traditional DNNs are so inefficient for these tasks

The local receptive fields enable neurons to extract low level features such as edges, corners, and end-points with respect to their orientation. CNNs are robust to input shift or distortion by using receptive fields to identify these low level features across the entire input space, performing local averaging and downsampling in the layers following convolution layers means the absolute location of the features is less important than the position of the features relative to the position of other identified features [9]. Each layer produces higher degrees of abstraction from the input layer, in doing so these abstractions retain important information about the input, these abstractions are referred to as feature maps. The layers performing downsampling are known as pooling layers,

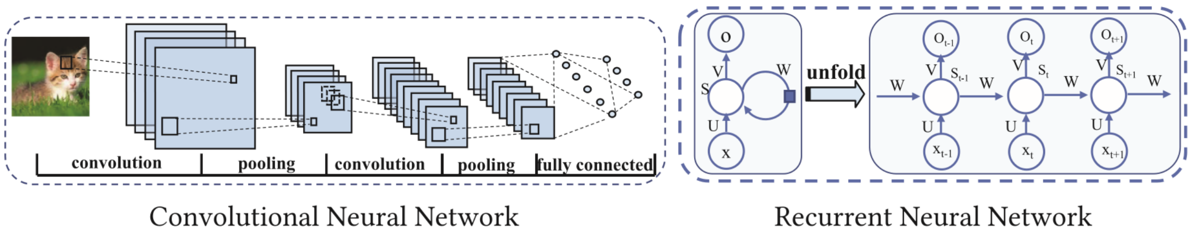

Figure 2: A typical example of a CNN (left) and RNN (right)
(Adopted figure from [1])

they reduce the resolution or dimensions of the feature map which reduces overfitting and speeds up training by reducing the number of parameters in the network [10].

Convolutional Networks for Images, Speech, and Time-Series by LeCunn & Bengio

CNNs have been found to be effective in many different AI domains, popular applications include: computer vision, NLP, and speech processing.

2.1.3 Recurrent Neural Netowrks

Recurrent Neural Network (RNN)

RNNs are deep learning models that use loops in their layer connections to make predictions with sequential inputs and maintain state over those inputs, this architecture is designed specifically for time series predictions [13].

2.2 Neural Network Compression

2.2.1 Pruning

Network pruning is the process of removing unimportant connections, leaving only the most informative connections. There has been a substantial amount of research into how pruning can be used to reduce overfitting and network complexity [14]–[17], but more recent research shows that some pruning methodologies can produce pruned networks with no loss of accuracy [18].

2.2.2 Quantization and Weight Sharing

Quantization is the process of limiting the number of bits used to represent each weight within a network, this process results in many weights using identical or very close weight values. These repeated weight values creates an ideal situation to use weight sharing techniques.

The paper Deep Compression by Han et al [19] weight sharing is taken a step further and clustering is employed to gather the quantized weights into bins (whose value is denoted by the centroid of that bin) then an index is assigned to each weight that points to the weights corresponding bin, the bins value is the centroid of that cluster, which is further fine-tuned by subtracting the sum of the gradients for each weight in the bin their respective centroid see Fig. 3.

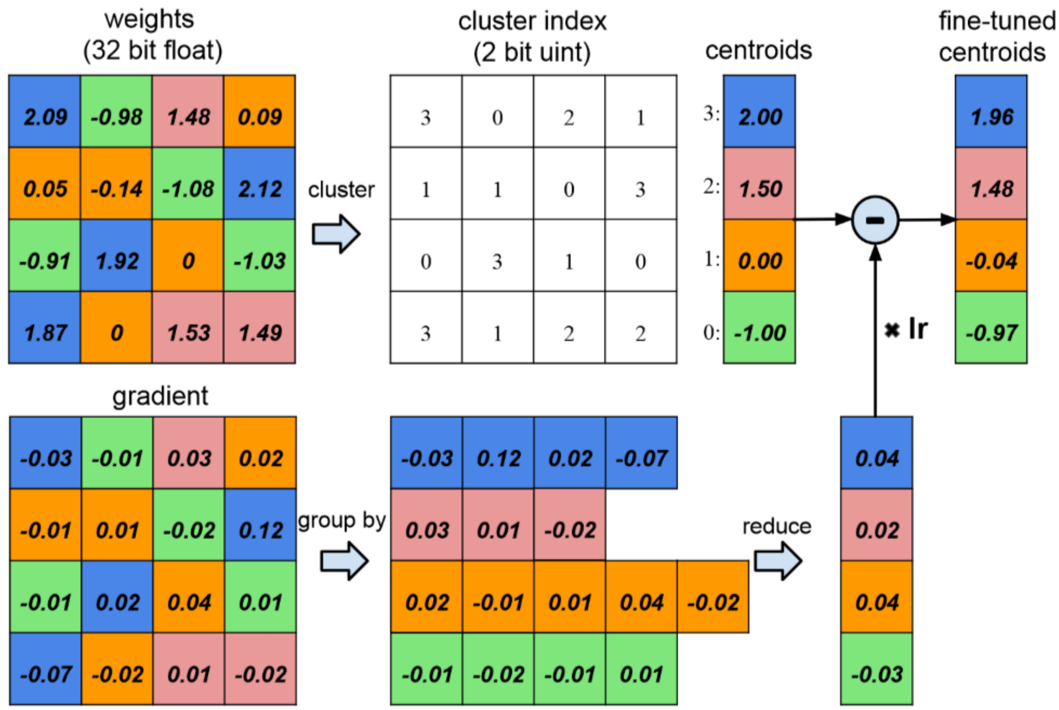

Figure 3: Weight sharing by quantization with centroid fine-tuning using gradients **(Adopted figure from [19])**

2.2.3 Distillation

2.2.4 Low-rank Factorization

2.2.5 Network Design Strategies

2.3 AI accelerators

Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit Register File	1	10
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit SRAM Cache	5	50
32 bit DRAM Memory	**640**	**6400**

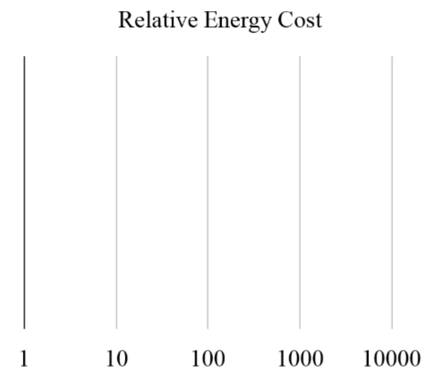

Figure 4: Energy table for 45nm CMOS process
(Adopted figure from [18])

The increasing popularity of DNNs for classification tasks such as computer vision, speech recognition and natural language processing has promted work to accelerate execution using specialised hardware. AI accelerators tend to prioritise improving the performance of networks from two perspectives; increasing compuational throughput, and decreasing energy consumption. Energy consumption is critical to the feasibility of performing inference on mobile devices, the dominant factor in this area is memory access, figure. 4 shows the energy consumption for a 32 bit floating point add operation and a 32 bit DRAM memory access on a 45nm CMOS chip, they note that DRAM memory access is 3 orders of magnitude of an add operation. Such hardware is commonly known as AI accelerators, these can be built to accelerate both the *training* and *inference* stages of execution, this section will specifically focus on the *inference* phase, however some of the following are capable of both.

2.3.1 GPU

2.3.2 VPU

One commercial hardware accelerator using a VPU architecture is the Intel Movidius Neural Compute Stick.

- 16 VLIW (very long instruction word) SHAVE (streaming hybrid architecture vector engine) processors, optimized for machine vision and able to run parts of a neural network in parallel

- 4Gb LPDDR3 DRAM

2.3.3 TPU

The TPU is a custom ASIC developed by google, designed specifically for TensorFlow, conventional access to these chips is via a cloud computing service. Google claims [20] the latest 4th generation TPUv4 is capable of more than double the matrix multiplication TFLOPs of TPUv3 (Wang et al. [21] describes a peak of 420 TFLOPs for the TPUv3). The TPU implements data parallelism in a manner prioritising batch size, one batch of training data is split evenly and sent to each core of the TPU, so total on-board memory determins the maximum data batch size. Each TPU core has a complete copy of the model in memory, so the maximum size of the model is determined by the amount of memory avaliable to each core [21].

2.3.4 APU

2.4 Memory factors for Deep Neural Networks

- Show that techniques to break the memory wall should result in higher throughput

- How does memory locality affect inference latency?

- How much cpu time is spent accessing memory?

- How do we use compression techniques to improve inference time (speedup)?

- Why does pruning not reduce inference time?

2.4.1 Memory Allocation

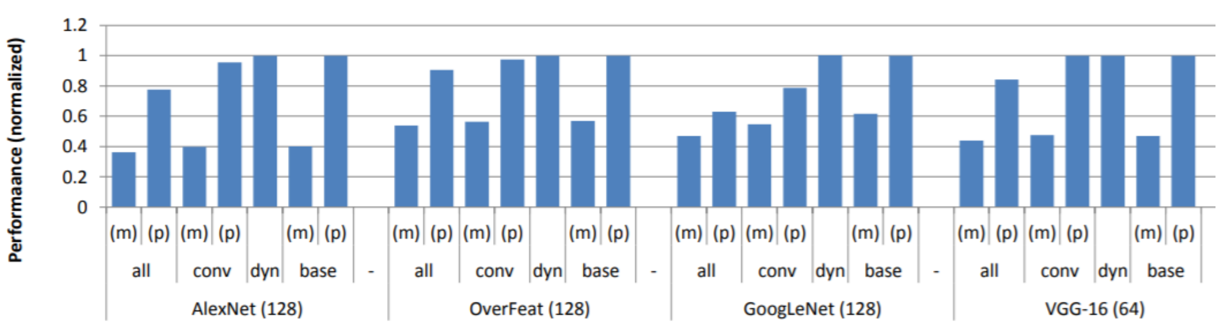

Figure 5: vDNN performance, showing the throughput using various memory allocation strategies. **(Adopted figure from [22])**

While designed specifically for training networks that would otherwise be to large for a GPU, the memory manager vDNN proposed by Rhu et al [22] does provide some insight into the importance of memory locality to neural network throughput. Fig. 5 summarizes the performance of neural networks using vDNN to manage memory compared to a baseline memory management policy (*base*). The vDNN policies include: static policies (denoted as *all* and *conv*) and a dynamic policy (*dyn*). *base* simply loads the full model into the GPU memory, consequently providing optimal memory locality. *all* refers to a policy of moving all Xs out of GPU memory, and *conv* only offloads Xs from convolutional layers, Xs are the input matrices to each layer, denoted by the red arrows in Fig. 6. Each of *base*, *conv* and *all* are evaluated using two distinct convolutional algorithms - memory-optimal (m) and performance-optimal (p). Finally the *dyn* allocation policy chooses (m) and (p) dynamically at runtime.

Observing the results in Fig. 5 where performance is characterized by latency during feature extraction layers; a significant performance loss is evident in the *all* policy compared to baseline, this loss is caused because no effort is made to optimise the location of network parameters in memory. In this example the memory allocations are being measured between memory in the GPU (VRAM) and host memory (DRAM) accessed via the PCI lanes. This does show how important the latency in memory access can be crucial for model throughput.

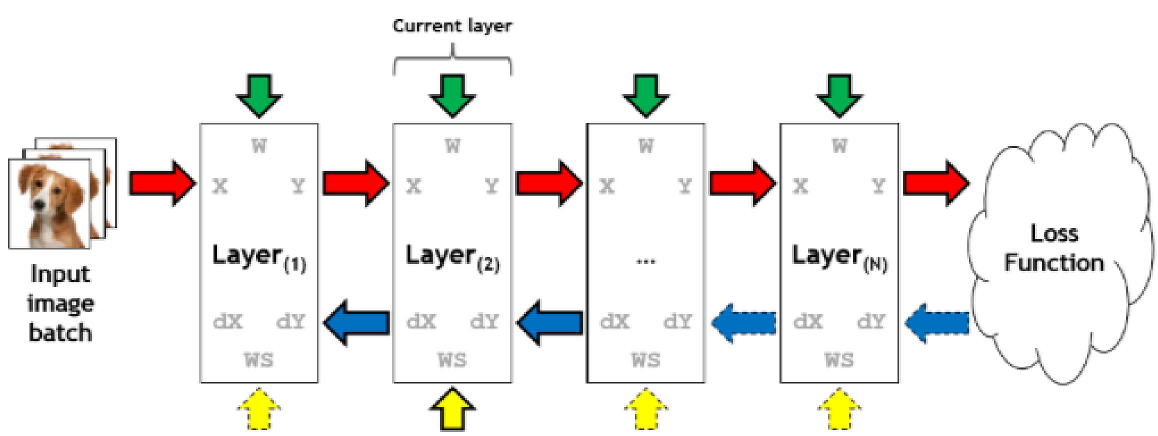

Figure 6: Memory allocations required for linear networks. All green (W) and red (X) arrows are allocated during inference, the blue and yellow arrows are allocated during training.
(Adopted figure from [22])

Justifies the need for compression ... pruning

2.4.2 Memory Access

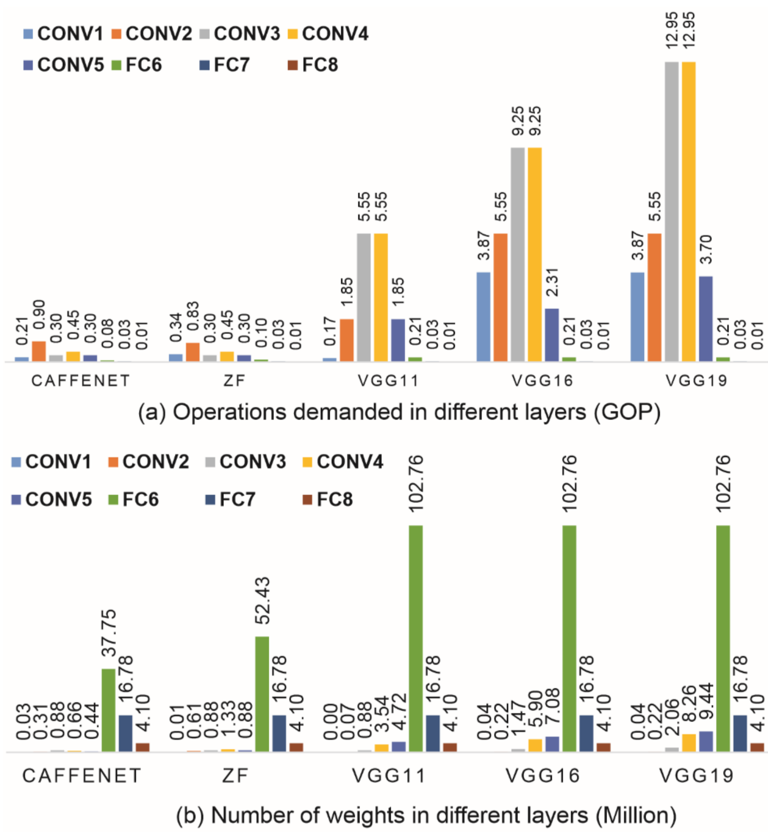

Figure 7: Complexity distribution of CNN models
(Adopted figure from [7])

A significant portion of DNN computation is matrix-vector multiplication, ideally weight reuse techniques can speed up these operations. However some DNNs feature FC layers with more than a hundred million weights (Fig. 7), memory bandwidth here can be an issue since loading these weights can be a significant bottleneck [7]. As observed in Section 2.4.1 this indicates that compression (Section 2.2) techniques could help alliviate this bottleneck by making parameters avaliable for cache reuse. However often modern networks are so large and complex there can still be an insufficient cache capacity for the full network parameters even when using modern compression techniques such as described in [19], in a follow up paper Han et al. [23] discuss this case where memory accesses occur for every operation because the codebook (from a pruned and then quantised network) cannot be reused properly. This paper proposes EIE (an inference engine for compressed networks) also shows that while compression does reduce the total number of operations, the irregular memory access patterns required when accessing a weight codebook via their indices mitigates the expected improvements in inference performance, see Fig. 8, this is due to the BLAS library for CPU and GPU not supporting indirect look-up and relative indexing [19].

Platform	Batch Size	Matrix Type	AlexNet			VGG16			NT-		
			FC6	FC7	FC8	FC6	FC7	FC8	We	Wd	LSTM
CPU (Core i7-5930k)	1	dense	7516.2	6187.1	1134.9	35022.8	5372.8	774.2	605.0	1361.4	470.5
		sparse	3066.5	1282.1	890.5	3774.3	545.1	777.3	261.2	437.4	260.0
	64	dense	318.4	188.9	45.8	1056.0	188.3	45.7	28.7	69.0	28.8
		sparse	1417.6	682.1	407.7	1780.3	274.9	363.1	117.7	176.4	107.4
GPU (Titan X)	1	dense	541.5	243.0	80.5	1467.8	243.0	80.5	65	90.1	51.9
		sparse	134.8	65.8	54.6	167.0	39.8	48.0	17.7	41.1	18.5
	64	dense	19.8	8.9	5.9	53.6	8.9	5.9	3.2	2.3	2.5
		sparse	94.6	51.5	23.2	121.5	24.4	22.0	10.9	11.0	9.0
mGPU (Tegra K1)	1	dense	12437.2	5765.0	2252.1	35427.0	5544.3	2243.1	1316	2565.5	956.9
		sparse	2879.3	1256.5	837.0	4377.2	626.3	745.1	240.6	570.6	315
	64	dense	1663.6	2056.8	298.0	2001.4	2050.7	483.9	87.8	956.3	95.2
		sparse	4003.9	1372.8	576.7	8024.8	660.2	544.1	236.3	187.7	186.5
EIE	**Theoretical Time**		**28.1**	**11.7**	**8.9**	**28.1**	**7.9**	**7.3**	**5.2**	**13.0**	**6.5**
	Actual Time		**30.3**	**12.2**	**9.9**	**34.4**	**8.7**	**8.4**	**8.0**	**13.9**	**7.5**

Figure 8: Wall clock time comparison for sparse and dense matrices between CPU, GPU, mGPU and EIE (an FPGA custom accelerator)
(Adopted figure from [23])

2.4.3 Optimisation Techniques

- *What can we do between out of box pruning and achieving a speedup?*

- *What changes can be made to the underlaying structures?*

Han et al [23] provide a clear description of a technique for exploiting the sparity of activations by storing an encoded sparse weight matrix in a variant of compressed sparse column format [24].

3 Research Methodology

This is required for research projects and should be linked back to the project aim and objectives. It should describe the research methods that will be employed in the project and the research questions that will be investigated.

1. build dataset of benchmarks from my systematic benchmark framework from models
2. perform pruning on models
3. run benchmark again with pruning
4. make adjustments to underlaying mechanism of parameter storage
5. verify adjustments do not break the model
5. run benchmarks again
6. draw conclusions

Find baselines/benchmarks

How to perform pruneing

Look at underlaying storage mechanism of parameters in Network

- provide some plots visualising the sparsity of the weights for the pruned matrix

perform some engineering of refactoring/altering these mechanisms

rerun systematic benchmarking framework

4 Design

Initial software design/sketch of research Methodology

5 Evaluation Strategy

Details of the evaluation and analysis to be conducted

6 Project Management

6.1 Plan

6.2 Risk Analysis

mention benchmarking NLP/NLG/Audio - text/text - audio models as a risk to the project

6.3 Professional, Legal & Ethical issues

A Back matter

A.1 References

References

[1] Y. Chen, B. Zheng, Z. Zhang, Q. Wang, C. Shen, and Q. Zhang, "Deep Learning on Mobile and Embedded Devices: State-of-the-art, Challenges, and Future Directions," *ACM Computing Surveys*, vol. 53, no. 4, pp. 1–37, Sep. 26, 2020, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3398209. [Online]. Available: https://dl.acm.org/doi/10.1145/3398209 (visited on 10/01/2020).

[2] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017, ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2017.2761740. [Online]. Available: http://ieeexplore.ieee.org/document/8114708/ (visited on 10/01/2020).

[3] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA Transactions on Signal and Information Processing*, vol. 3, e2, 2014, ISSN: 2048-7703. DOI: 10.1017/atsip.2013.9. [Online]. Available: https://www.cambridge.org/core/product/identifier/S2048770313000097/type/journal_article (visited on 10/16/2020).

[4] J. Thierry-Mieg, "How the fundamental concepts of mathematics and physics explain deep learning.," p. 16,

[5] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015, ISSN: 08936080. DOI: 10.1016/j.neunet.2014.09.003. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0893608014002135 (visited on 10/21/2020).

[6] G. Hinton, D. Rumelhart, and R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088,

[7] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16, New York, NY, USA: Association for Computing Machinery, Feb. 21, 2016, pp. 26–35, ISBN: 978-1-4503-3856-1. DOI: `10.1145/2847263.2847265`. [Online]. Available: `https://doi.org/10.1145/2847263.2847265` (visited on 11/02/2020).

[8] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of Physiology*, vol. 160, no. 1, pp. 106–154, Jan. 1, 1962, ISSN: 00223751. DOI: `10.1113/jphysiol.1962.sp006837`. [Online]. Available: `http://doi.wiley.com/10.1113/jphysiol.1962.sp006837` (visited on 10/15/2020).

[9] Y. LeCun, Y. Bengio, and T. B. Laboratories, "Convolutional Networks for Images, Speech, and Time-Series," *The handbook of brain theory and neural networks MIT Press*, p. 15,

[10] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. S. Iyengar, "A Survey on Deep Learning: Algorithms, Techniques, and Applications," *ACM Computing Surveys*, vol. 51, no. 5, pp. 1–36, Jan. 23, 2019, ISSN: 0360-0300, 1557-7341. DOI: `10.1145/3234150`. [Online]. Available: `https://dl.acm.org/doi/10.1145/3234150` (visited on 10/15/2020).

[11] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, Apr. 1980, ISSN: 0340-1200, 1432-0770. DOI: `10.1007/BF00344251`. [Online]. Available: `http://link.springer.com/10.1007/BF00344251` (visited on 10/18/2020).

[12] ——, "Neocognitron: A hierarchical neural network capable of visual pattern recognition," *Neural Networks*, vol. 1, no. 2, pp. 119–130, Jan. 1988, ISSN: 08936080. DOI: `10.1016/0893-6080(88)90014-7`. [Online]. Available: `https://linkinghub.elsevier.com/retrieve/pii/0893608088900147` (visited on 10/18/2020).

[13] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019, ISSN: 0018-9219, 1558-2256. DOI: `10.1109/JPROC.2019.2921977`. [Online]. Available: `https://ieeexplore.ieee.org/document/8763885/` (visited on 10/01/2020).

[14] S. J. Hanson and L. Y. Pratt, “Comparing Biases for Minimal Network Construction with Back-Propagation,” p. 9,

[15] B. Hassibi and D. G. Stork, “Second Order Derivatives for Network Pruning: Optimal Brain Surgeon,” p. 8,

[16] Y. LeCun, J. S. Denker, and S. A. Solla, “Optimal Brain Damage,” p. 8,

[17] N. Strom. (1997). “Phoneme probability estimation with dynamic sparsely connected artificial neural networks,” [Online]. Available: /paper/Phoneme-probability-estimation-with-dynamic-neural-Strom/a9392b9299972452ea6fbc3c605f76bb1e21ae42 (visited on 11/13/2020).

[18] S. Han, J. Pool, J. Tran, and W. J. Dally. (Oct. 30, 2015). “Learning both Weights and Connections for Efficient Neural Networks.” arXiv: 1506.02626 [cs], [Online]. Available: http://arxiv.org/abs/1506.02626 (visited on 10/30/2020).

[19] S. Han, H. Mao, and W. J. Dally. (Feb. 15, 2016). “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding.” arXiv: 1510.00149 [cs], [Online]. Available: http://arxiv.org/abs/1510.00149 (visited on 11/06/2020).

[20] (). “Google wins MLPerf benchmark contest with fastest ML training supercomputer,” [Online]. Available: https://cloud.google.com/blog/products/ai-machine-learning/google-breaks-ai-performance-records-in-mlperf-with-worlds-fastest-training-supercomputer/ (visited on 11/15/2020).

[21] Y. E. Wang, G.-Y. Wei, and D. Brooks. (Oct. 22, 2019). “Benchmarking TPU, GPU, and CPU Platforms for Deep Learning.” arXiv: 1907.10701 [cs, stat], [Online]. Available: http://arxiv.org/abs/1907.10701 (visited on 11/15/2020).

[22] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler. (Jul. 28, 2016). “vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design.” arXiv: 1602.08124 [cs], [Online]. Available: http://arxiv.org/abs/1602.08124 (visited on 10/30/2020).

[23] S. Han, 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 *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, Seoul, South Korea: IEEE, Jun. 2016, pp. 243–254, ISBN: 978-1-4673-8947-1. DOI: `10.1109/ISCA.2016.30`. [Online]. Available: `http://ieeexplore.ieee.org/document/7551397/` (visited on 11/02/2020).

[24] R. Vuduc, "Automatic Performance Tuning of Sparse Matrix Kernels," p. 455,

A.2 Appendices

to include additional material, consult with your supervisor.

Acronyms

ASIC application specific integrated circuit. 9

BLAS basic linear algebra subprograms. 12

CNN convolutional neural network. 2, 4, 5

DNN deep neural network. 3, 4

NLP natural language processing. 5

RNN recurrent neural network. 2, 4, 5

TPU tensor processing unit. 9