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

December 10, 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: Pruning neural networks is a complex task in its own right, sucessful implementations require intricate knowledge of layer intedependencies and can be a daunting task even to the machine learning expert. This dissertation hypothesises that a classic hyperparameter optimisation algorithm can be applied to the domain of neural network pruning with the goal of reducing latency. We outline a plan to perform experiments that will highlight which pruning algorithms are most suitible to reduce inference latency. This paper constructs a methodology to develop a framework to automate the tuning of compression parameters, and test the results.

Contents

Acronyms

ASIC application specific integrated circuit.

BLAS basic linear algebra subprograms.

CNN convolutional neural network.

DNN deep neural network.

FC fully connected.

FPGA field programmable gate array.

NLP natural language processing.

RNN recurrent neural network.

SoC system on a chip.

TOPS trillion operations per second.

TPU tensor processing unit.

1 Introduction

1.1 Motivation

With the continued revolution of AI technologies a desire 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 resources alongside new and constantly evolving AI applications, inference at the edge can provide better privacy and latency than the remote datacenter alternatives.

Neural network compression is one avenue for bringing inference to the edge, intuitively we might think that a network with a smaller memory footprint would naturally have lower inference latency but this is often not always the case. Utilising neural network compression effectively requires expert level knowledge of not only the network structure but the consequences of compression because compression techniques such as pruning can have cascading effects througout a neural network. This alone can make compression a daunting task, even for experienced machine learning practicioners, it gets worse however, these compression algorithms often feature complex parameters with implications that may not be revealed until a substantial amount of time has already been invested in retraining a compressed model.

1.2 Hypothesis

Using a systematic compression method selection process combined with a bayesian optimisation algorithm we can partially automate compression parameter selection and imporve inference latency based on an accuracy threshold in a typical edge computing environment.

1.3 Research Aims

Aim 1 - This dissertation will research methodologies for reducing inference latency using a collection of off-the-shelf compression techniques, we will investigate which compression techniques have a positive effect on inference latency, and consider the context of this improvement with respect to the layer structure of the neural network.

Aim 2 - We will use this contextual information to select approriate compression methodologies and reduce the search space down to a single pruning algorithm per sub domain.

Aim 3 - Maintain a valid testing environment by using an edge based ai accelerator to perform inference, while training and compression will be performed on a GPU.

Aim 4 - Develop a interface to optimise compression parameters according to a metric representing the union of accuracy and latency.

Objectives

- **O0:** Develop a methodology to verify that the compression methods are actually being applied to the model being represented.
- **O1:** Select at least 1 neural network model to use for testing.
- **O2:** Select 2 suitable datasets for testing with a significant distinction between the cardinality of categories.
- **O3:** Evaluate a pool of compression algorithms with respect to end-to-end latency.
- **O4:** Measure latency for individual layers during inference.
- **O5:** Investigate the effect of composing select algorithms from different compression categories.
- **O6:** Select compression parameters to optimise.
- **O7:** Develop a interface to parameterise select compression methods.
- **O8:** Evaluate a model using a bayesian optimisation approach on compression parameters.

2 Background

This Section will be split into 4 subsections:

Section 2.1 - **Deep Learning**: An overview of the basic components of a deep neural network and the CNN model.

Section 2.2 - **Neural Network Compression**: Discusses neural network compression techniques and on how they change the underlying representations of DNNs.

Section 2.3 - **AI accelerators** Covers a few popular AI accelerators architectures, their strengths, weaknesses and specialisms.

Section 2.4 - **Memory factors for Deep Neural Networks**: Describes the how DNNs interact with memory, and discusses some of the implications of this.

2.1 Deep Neural Networks

2.1.1 Neural Networks & Deep Learning

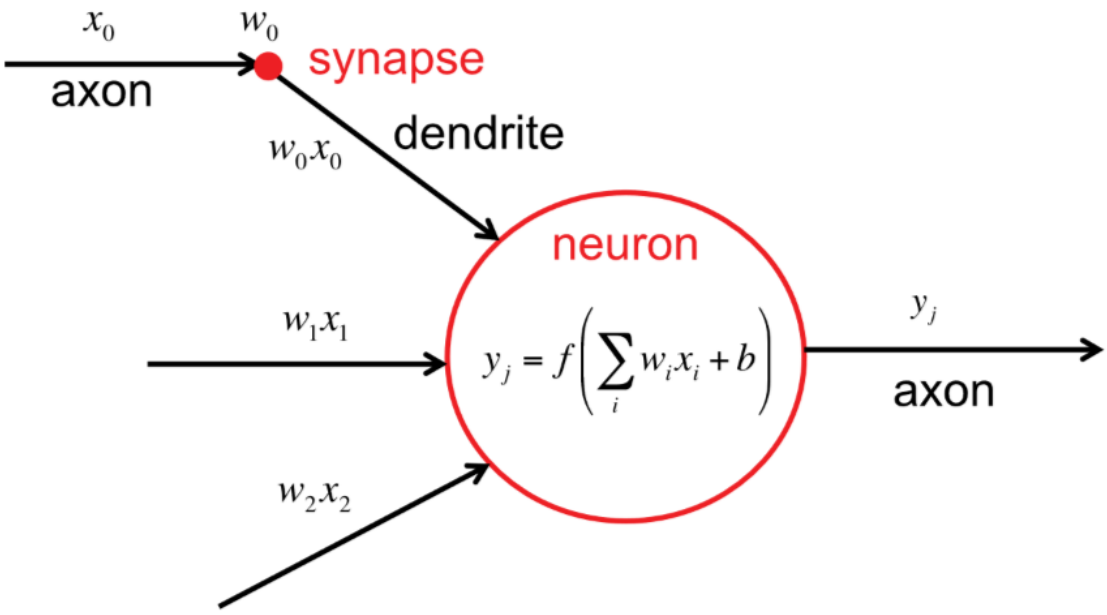

Figure 1: Neuron with corresponding biologically inspired labels.
(Adopted figure from [1])

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 [2].

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 [3]. 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. 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 [1].

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 randomly initialised weights, without which a DNN would just be a linear algebra operation [1], the cumulative effect of the activations in each layer results in elabourate causal chains of transormations that influence the aggregate activation of the network.

2.1.2 Inference and Training

Training or learning in the context of DNNs is the process of finding the optimal parameters (value for the weights and bias) in the network. Upon completion of training *inference* can be performed, this is where new input data is fed into the network, a series of operations is performed using the trained parameters, and some meaninful output is obtained such as a classification, regression, or function approximation. Many techniques can be used to search for optimal parameters, one example known as supervised learning is as follows: Begin by passing some training data through

the network, next the gap between the known ideal output (labels) and the computed outputs from the current weights is calculated using a loss function. Finally the weights are updated using an optimization process such as gradient descent coupled with some form of backward pass, backpropagation is a popular choice for this.

2.1.3 Convolutional Neural Networks

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 [4]–[6]. The influential paper by Hubel & Weisel [4] ultimately had a significant influence on the design of CNNs via the Neocognitron, as proposed by Fukushima in [7] and again evaluated in [8], these papers paved the way for the modern CNN.

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 [7]. 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 [5]. 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, 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 [6].

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

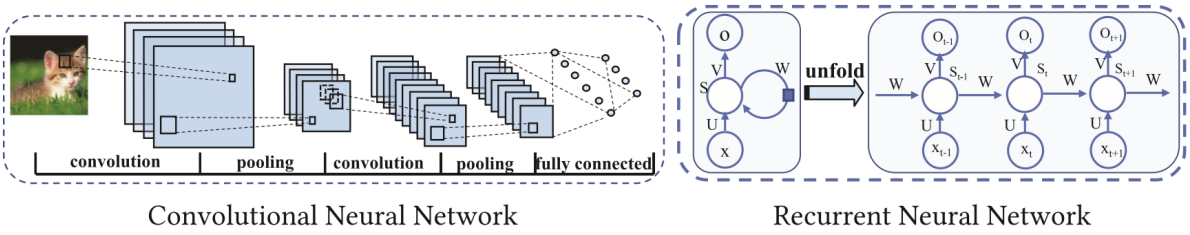

Figure 2: A typical example of a CNN (left) and RNN (right)
(Adopted figure from [9])

2.2 Neural Network Compression

Neural network compression is necessary due to storage related issues that often arise on resource contrained systems due to the high number of parameters that modern DNNs tend to use, state-of-the-art CNNs can have upwards of hundreds of millions of parameters. Different compression methods can result in variuous underlaying representations of the weight matrices, particularly with respect to its sparsity. Compression techniques that preserve the density of the weight matrix tend to result in inference acceleration on general-purpose processors[10], [11], not all techniques preserve this density and can result in weight matrices with various degrees of sparsity which in turn have varing degrees of regularity. These techniques, the resulting representations of parameters, and their consequences will be discussed in this section.

2.2.1 Pruning

Network pruning is the process of removing unimportant connections, leaving only the most informative connections. Typically pruning is performed by iterating over the following 3 steps: begin by evaluating the importance of parameters, next the least important parameters are pruned, and finally some fine tuning must be performed to recover accuracy. There has been a substantial amount of research into how pruning can be used to reduce overfitting and network complexity [12]–[15], but more recent research shows that some pruning methodologies can produce pruned networks with no loss of accuracy [16].

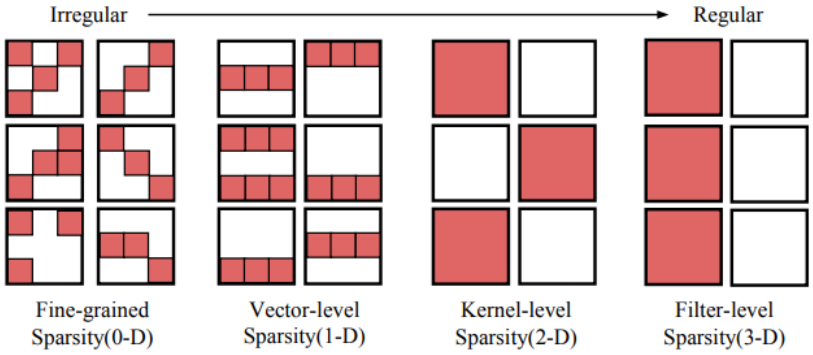

Figure 3: Sparse structures in a 4-dimensional weight tensor. Regular sparsity makes hardware acceleration easier.
(Adopted figure from [17])

This process of pruning the weight matrix within a DNN results in a sparse matrix representation of weights, where the degree of sparsity is determined by the pruning algorithm being used and hyperparameters that can be tuned for the situation, such as how much accuracy loss is considered acceptable, and to what degree the neural network needs to be compressed. The pattern of sparsity in a weight matrix is a fundamental factor when considering how to accelerate a pruned neural network [17], this is known as the **granularity of sparsity**. Figure 3 provides a visual representation of granularity of sparsity, the spectrum of granularity usually falls between either **fine-grained (unstructured)** or **course-grained (structured)**, pruning techniques are also categorised by the aformentioned granularities.

The influential paper Optimal Brain Damage by LeCun et al [14] was the first to propose a very fine-grained pruning technique by identifying and zeroing individual weights within a network. Fine-grained pruning results in a network that can be challenging to accelerate without custom hardware such as proposed in [18], [19], a software solution has been theorized by Han et al [20] that would involve developing a customized GPU kernel that supports indirect matrix entry lookup and a relative matrix indexing format, see Section 2.2.2 for further details on the necessary steps for this technique.

Coarse-grained pruning techniques such as channel and filter pruning preserve the density of the network by altering the dimensionality of the input/output vectors, channel pruning involves removing an entire channel in a feature map, filter level pruning likewise removes an entire convo-

lutional filter.

This style of pruning however can have a significant impact on the accuracy of the network, but as demonstrated by Wen et al [21] accelerating networks with very course-grained pruning is straightforward because the model smaller but still dense, so libraries such as BLAS are able to take full advantage of the structure.

2.2.2 Quantisation

Most off-the-shelf DNNs utilise floating-point-quantisation for their parameters, providing arbitrary precision, the cost of this precision can be quite high in terms of arithmetic operation latency, high resource use and higher power consumption. However this arbitrary precision is often unnecesssary, extensive research [22], [23] has shown reducing the precision of parameters can have an extremely small impact on the accuracy. Quantisation can be broadly categorised into two groups: non-linear quantisation and fixed-point (linear) quantisation.

Fixed-point quantisation is the process of limiting the floating point precision of each parameter (and potentially each activation) within a network to a fixed point.

In the extreme fixed-point quantisation can represent each parameter with only 1 bit (also known as binary quantisation) with up to a theoretical 32x compression rate (in practice this is often closer to 10.3x) [9], Umuroglu et al. [24] used binary quantisation with an FPGA and achieved startling classification latencies (0.31μs on the MINIST dataset) while maintaining 95.8% accuracy, this is largely because the entire model can be stored in on-chip memory this is discussed further in Section 2.4.1.

Method	Para.	Speed-up	Top-1 Err. ↑		Top-5 Err. ↑	
			No FT	FT	No FT	FT
CPD	-	3.19×	-	-	0.94%	0.44%
	-	4.52×	-	-	3.20%	1.22%
	-	6.51×	-	-	69.06%	18.63%
GBD	-	3.33×	12.43%	0.11%	-	-
	-	5.00×	21.93%	0.43%	-	-
	-	10.00×	48.33%	1.13%	-	-
Q-CNN	4/64	3.70×	10.55%	1.63%	8.97%	1.37%
	6/64	5.36×	15.93%	2.90%	14.71%	2.27%
	6/128	4.84×	10.62%	1.57%	9.10%	1.28%
	8/128	6.06×	18.84%	2.91%	18.05%	2.66%
Q-CNN (EC)	4/64	3.70×	0.35%	0.20%	0.27%	0.17%
	6/64	5.36×	0.64%	0.39%	0.50%	0.40%
	6/128	4.84×	0.27%	0.11%	0.34%	0.21%
	8/128	6.06×	0.55%	0.33%	0.50%	0.31%

Figure 4: Comparison of the speed-up when quantising a convolutional layer in Alexnet, 3 different methods.
(Adopted figure from [25])

Non-linear Quantisation is a technique where the weights are split into groups and then assigned a single weight, this grouping can be accomplished in a number of ways, Gong et al. [26] used vector quantisation with k-means clustering and achieved compression rates of up to 24x while keeping the difference of top-five accuracy within 1%. Wu et al. [25] quantised both FC and convolutional layers in Alexnet using their Q-CNN framework

The paper Deep Compression by Han et al [20] quantisation and weight sharing is taken a step further. First the weights are pruned and quantized, next clustering is employed to gather the quantized weights into bins (whose value is denoted by the centroid of that bin) finally 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. 5.

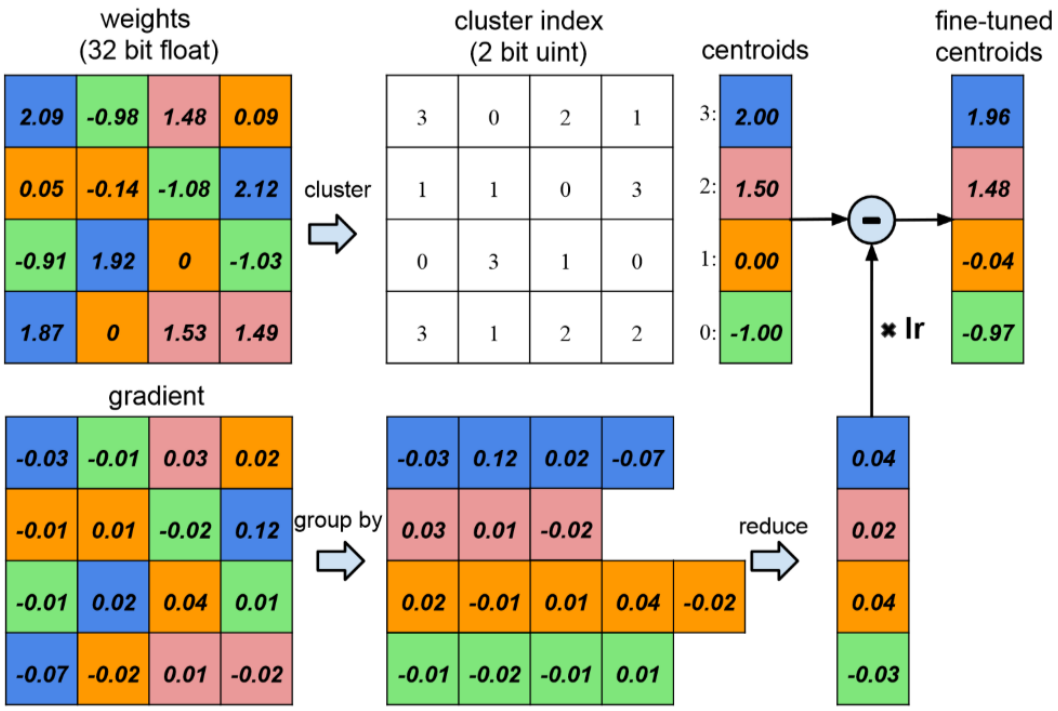

Figure 5: Weight sharing by quantisation with centroid fine-tuning using gradients **(Adopted figure from [20])**

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**

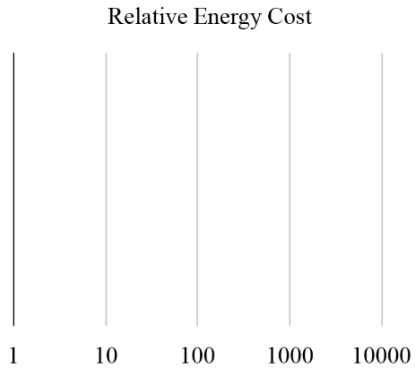

Figure 6: Energy table for 45nm CMOS process
(Adopted figure from [16])

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. 6 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. Hardware is commonly referred to as an AI accelerator, these can be built to accelerate both the *training* and *inference* stages of execution, this section will specifically focus on the *inference* phase, however many modern accelerators are capable of both.

2.3.1 VPU

One commercial hardware accelerator using a VPU architecture is the Intel Movidius Neural Compute Stick. It is a specialised SoC for computer vision applications, with a peak floating-point computational throughput of 1 TOPS, because of reasons described in Section 2.4.1 this peak throughput will be hard to achieve in any real world scenario.

- 16 VLIW (very long instruction word) SHAVE (streaming hybrid architecture vector engine)

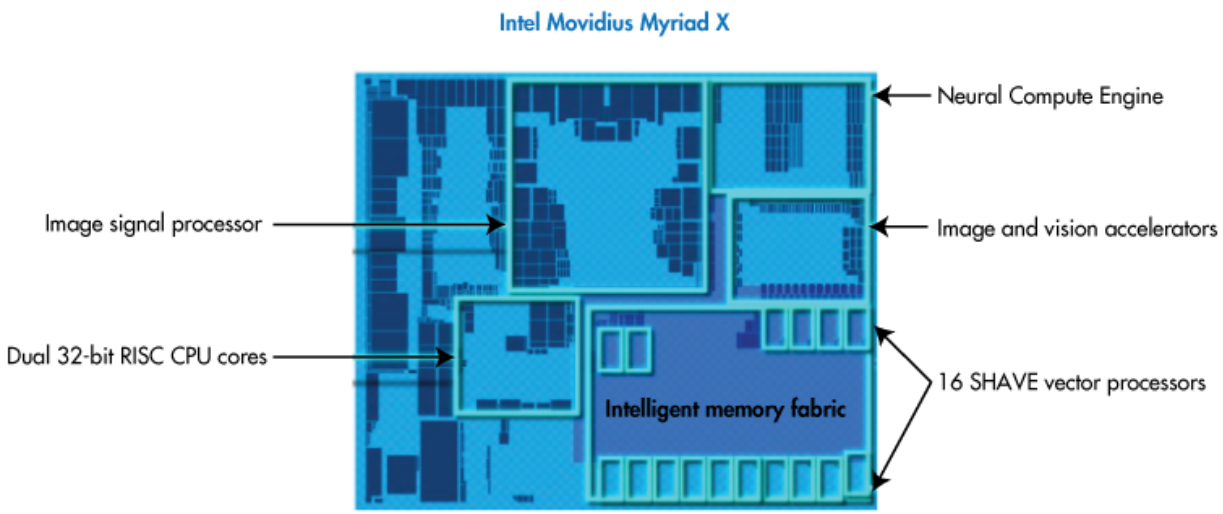

Figure 7: High level view of the Intel Movidius Myriad X VPU

processors, optimized for machine vision and able to run parts of a neural network in parallel.

- 2.5 MB On-Chip memory allowing for up to 400GB/s of internal bandwidth.
- 4Gb LPDDR4 DRAM

A key advantage of using hardware like the VPU is a customised computation pipeline that is optimised for high parallelism during inference. This however comes with the caveat that the OpenVINO framework is required to perform inference[27].

2.3.2 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 [28] the latest 4th generation TPUv4 is capable of more than double the matrix multiplication TFLOPs of TPUv3 (Wang et al. [29] 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 [29].

2.4 Memory factors for Deep Neural Networks

2.4.1 Memory Allocation

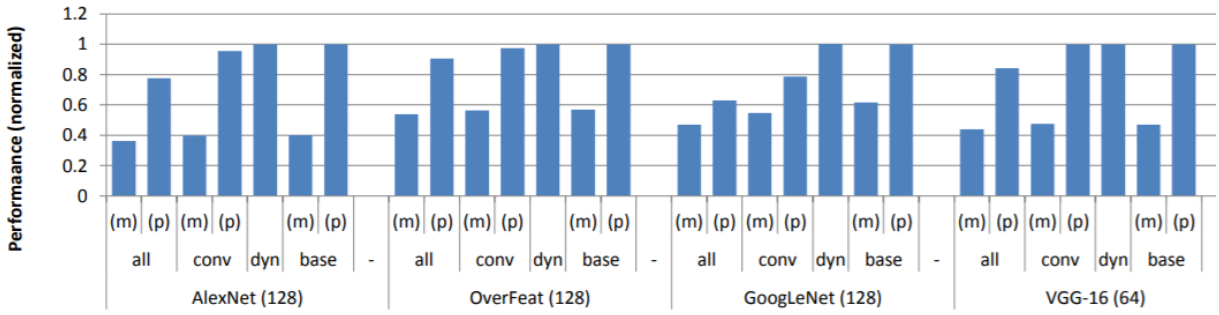

Figure 8: vDNN performance, showing the throughput using various memory allocation strategies. **(Adopted figure from [30])**

While designed specifically for training networks that would otherwise be to large for a GPU, the memory manager vDNN proposed by Rhu et al [30] does provide some insight into the importance of memory locality to neural network throughput. Fig. 8 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. 9. 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. 8 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.

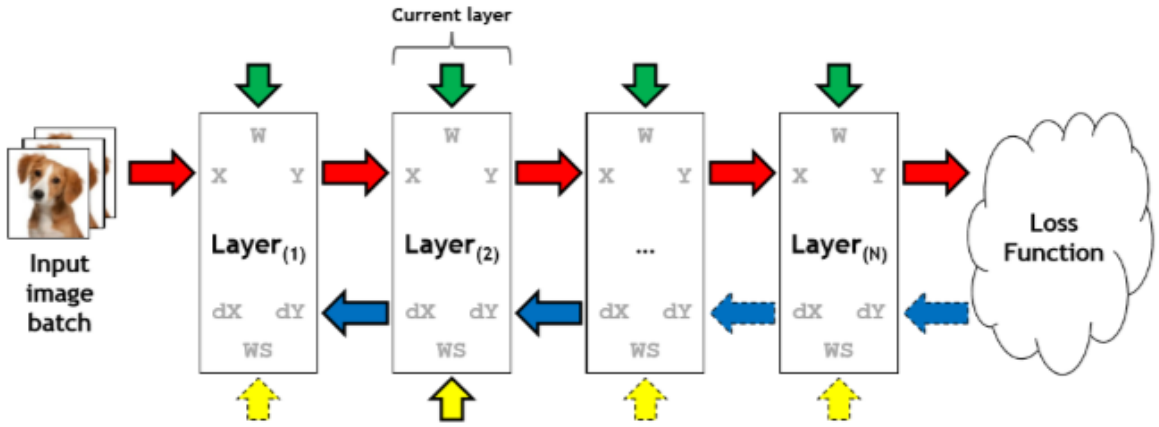

Figure 9: Memory allocations required for linear networks. All green (W) and red (X) arrows are allocated during inference, the blue and yellow arrows are allocated during training.
(Adopted figure from [30])

2.4.2 Memory Access

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. 10), memory bandwidth here can be an issue since loading these weights can be a significant bottleneck [31]. 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.

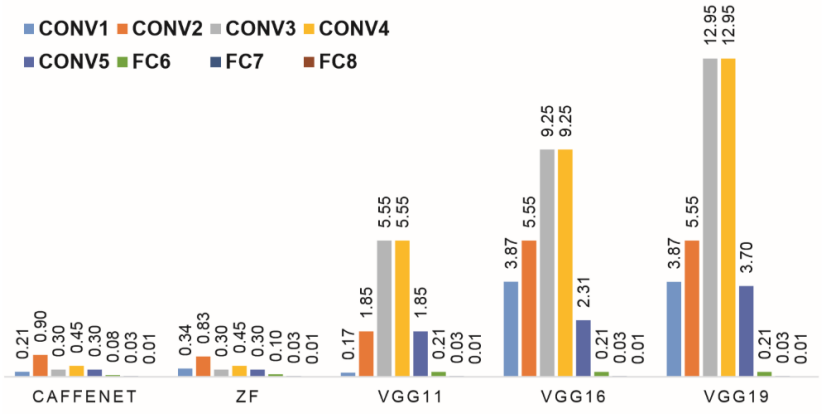

Figure 10: Operations demanded in different layers (GOP)
(Adopted figure from [31])

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 [20], in a follow up paper Han et al. [18] 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 FPGA inference engine for compressed networks) also shows that while compression does reduce the total number of operations, and a tangible speedup can be observed in the FC layers see Fig. 11, this technique when applied to convolutional layers has some issues.

Han et al [18] provide an elegant 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 [32], however implementing this is problematic (particularly in convolutional layers) due to the irregular memory access patterns, lack of library and kernel level support for this style of sparse matrix (as discussed in Section 2.2.1). It should also be noted that Fig. 11 is comparing general purpose compute hardware with a custom built FPGA, so the speedup while impressive would be more appropriate compared to other purpose built FPGAs, however the most pertinent part of this Figure is the single batch size FC layer comparison between dense and sparse matrices.

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 11: Wall clock time (μ) comparison for sparse and dense matrices in FC layers between CPU, GPU, mGPU and EIE (an FPGA custom accelerator)
(Adopted figure from [18])

3 Research

3.1 Research questions

The following are the questions this research will seek to investigate:

- Application of pruning algorithms on neural networks: how does changing the pruning paradigm impact latency? (Objectives O3, O4, and O5)
- Can we apply hyperparameter optimisation methods to neural network pruners to minimize latency? (Objective O7)
- To what degree can we improve latency (if at all) using an automated strategy while keeping the accuracy metrics above a set threshold? (Objective O8)

3.2 Research Methodology

3.2.1 Core Technology Summary

This sections provides a brief summary of the core technologies that will be utilised, the core functionality will be written in Python, a combination of different Python versions will be necessary to properly utilise Distiller and OpenVINO. This versioning issue is one of the reasons we use Redis, it can act as a message broker between Distiller and OpenVINO, we can facilitate a more flexible development by adopting this strategy (inference and training will be completely decoupled).

- Distiller - A neural network compression library.
- OpenVINO - A toolkit to perform inference.
- Weights & Biases - A tool for metric recording, visualisation, and parameter optimisation [33].
- Redis - A fast data structure store (used for FIFO queue).

3.2.2 Compression

The Distiller library was selected as the platform of choice to perform compression because it functions as a convienent toolbox to test a selection of compression methods under a common

environment, and critically a larg collection of compression methods [34]. The decision to use Distiller is reinforced by Aim 1, there is a rich selection of already-implemented (off-the-shelf) algorithms to choose from, the parameters of these compression algorithms are all defined according to a "scheduling recipe" that can be defined programmaatically [34]. This project focuses specifically on the pruning methods within distiller, thus we refer to the parts of the scheduling recipe concerned with pruners as the compression parameters, depending on the compression method being used these can be lists of layers to apply the method, initial & final sparsity, and groupings of the layer structure (channel vs filter).

We will need to consider the schedule for the learning-rate decay class to properly fine tune the model, we will define a fixed learning-rate decay for all experiments to imporove reliability of our results and reduce the search dimensions for our own optimiser.

Fig. 12 Shows a high level view of the Distiller library, to accomplish objective O7 we will primarily be interacting with the scheduler via the interface specified in objective O7.

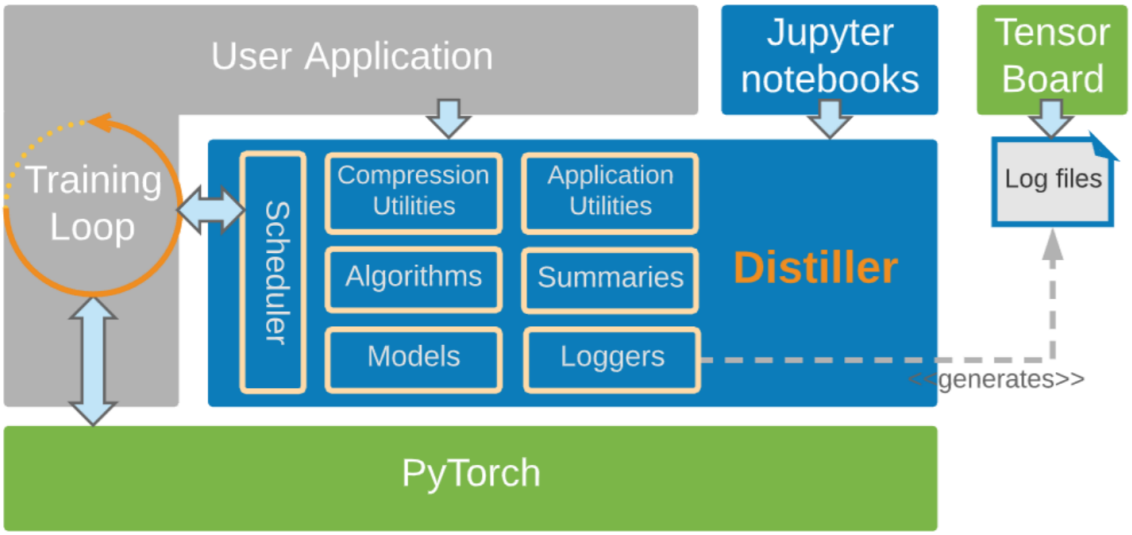

Figure 12: Distiller high-level architecture.
Figure adopted from [34]

Upon finishing training the Top1 and Top5 metrics will be recorded, ready to be recorded using Weights and Biases with the benchmark metrics when they are ready.

3.2.3 Benchmarking

OpenVINO will be the means by which we perform inference, and evaluate the latency of the compressed models (Objectives O3, O4, and O5). OpenVINO provides a deep learning deployment toolkit via an inference engine, it is compatible with the Neural Compute Stick and has many useful tools to produce metrics related to model and hardware performance.

Benchmarking will be performed using OpenVINO's built in benchmarking tool, we will record inference latency, individual layer latency, and a throughput metric (FPS). These metrics can be dumped into an easiliy processable .csv format, we will pass this data back to the training agent once the inference benchmark completes.

3.2.4 Manual Compression Parameter Selection

Using the layerwise benchmarking data we will attempt to filter out pruning algorithms that show a detrimental or negative imporvement in latency (Aim 1), this will reduce the scope of the problem and provide a narrower search space for optimisation (Aim 2).

3.2.5 Optimisation

To automate compression parameter tuning (Objective O7) we will use a bayesian optimisation technique that models the performance as a sample from a Gaussian process [35]. Convienently this process is neatly implemented in a machine learning data collection and visualisation tool know as Weights and Biases [33].

3.2.6 Training Benchmark Pipeline

Once the parameter tuner is ready we will need to produce a pipeline to move the models from the training host to the device hosting the neural compute stick, we will use a FIFO queue with a Redis store to accomplish this, a key advantage of this strategy is that it decouples the training system from the inference system, making it easy to add additional training agents.

4 Preliminary Evaluation

To demonstrate the necessity of objective O0, this section presents findings from a series of preliminary benchmarks. According to the literature covered in section 2.2.1 course-grained pruning algorithms should provide a demonstrable improvement in latency during inference. Likewise given the hardware requirements quantisation is an even more consistent in its ability to reduce inference latency (see section 2.2.2).

Table 1 presents findings when benchmarking inference of resnet20 with the CIFAR10 dataset on the NCS (table ref), these results show no real change between compression methods, this is an unexpected result. A run-to-run variance of 20ms was observed when running the same model repeatedly, so we can reasonably say that all these results are within margin of error.

Compression algorithm	Top 1 Accuracy	Top 5 Accuracy	Latency (ms)	Throughput (FPS)
N/A (baseline)	91.120	99.660	10.19	392.22
AGP filter, fine-grained, and row pruning	91.110	99.700	10.15	394.14
ssl channels removal training	90.700	99.680	10.03	398.22
ssl channels removal finetuning	91.610	99.780	10.17	389.17

Table 1: Preliminary NCS inference results, Resnet20 trained and tested with the CIFAR10 dataset.

Experiment stage 0 in Section 5.2.1 will investigate this further with the aim of verifying proper application of the compression scheduler to the model, and assessing factors in transferring the model to the NCS.

5 Evaluation Strategy

This section presents practical details of the evaluation process and how analysis will be conducted.

5.1 Preliminary considerations

Environment - To simulate an edge compute environment we will use the Intel Neural Compute stick for inference, training and compression will be performed on a consumer grade GPU, these environment choices satisfy Aim 3.

Determine models, and test datasets - In accordance with objectives O1 & O2, at least 1 popular model will be selected, the models layer structure should be considered during selection: depth, number of parameters, and number of convolutionalFC layers will be taken into account. Two datasets will be used with these models, one with a small number of classes such as CIFAR-10 and also a dataset with a much larger number of classes such as Imagenet. Ideally the selected model should have pretrained weights for both datasets, if necessary we will train and store the models ourselves.

Select compression algorithms - Select at least 4 algorithms that utilise different pruning methods. If feasable these algorithms should explore a spectrum of sub-domains of Pruning (such as fine-grained pruning vs filter or channel pruning). Any selected algorithms should have the capability to be applied to a specific layer (for this reason knowledge distillation techniques would not be suitable here). For compatibility reasons with the ONNX intermediate representation quantisation may not be exporeted from Distiller (as of Oct 2019) [34], *Experiment stage 0* (Section 5.2.1) will present a good opportunity to test if this is still the case. The selection of compression algorithms will also depend on how they are implemented within the Intel Distiller library, for example Automated Gradual Pruner works on a diverse set of neural network architectures so it would be a suitable choice [36].

5.2 Experiment tasks

5.2.1 Experiment stage 0: Verify preliminary results

This experimental stage is necessary due to the preliminary evaluation reported in Section 4.

The following steps will be taken to complete objective O0:

1. **Evaluate Compression Scheduler**: We will begin by closely assessing the compression scheduler behaviour, there are extensive tools for evaluating the sparsity metrics of pruned models. This will give us insight into how the compression algorithm is affecting the model

2. **Evaluate Intermediate Representation**: We used the ONNX format (an open standard format for represnting machinea learning models), to transfer our model from distiller to OpenVINO. We will take a closer look at this representation for issues with compatibility of sparse tensorsquantisation and the conversion process. One quick verification strategy is to convert the ONNX represtation back to distiller and re-evaluate the models sparsity properties

3. **Evaluate OpenVINO Representations**: Models converted from ONNX format by OpenVINO are then translated into a format consumable by OpenVINO's Inference Engine, this conversion process uses an OpenVINO tool called the Model Optimser. Transformations on the model during these stages will need to be investigated to confirm they do not interfere with the compressed model.

5.2.2 Experiment stage 1: Initial data gathering

Completion of the following stages will satisfy objectives O3, O4, and O5

1. **Aqcuire suite of baseline data**: Using a fixed test set from each dataset we will run inference on all the models with no compression techniques applied, to acquire a *baseline*. The end to end latency, individual layer latency and also the Top1/Top5 accuracy will be recored for each model/dataset pairing.

2. **Apply compression and gather full compression data**: For each compression algorithm and preselected parameters compress the models used in the *baseline* tests by selectively

applying the compression technique to a subset of relevant layers (i.e. layers which the algorithm can be applied). Next using the same testing data from *baseline*, perform inference with the compressed models. The same metrics will be logged as in the *baseline*. We will refer to this test as *full compression*.

3. **Evaluate full compression**: We will make observations about the resulting data, the key metric we are interested in is latency at this stage. First we will make general comparisons with the end-to-end latency and accuracy against the *baseline*. Next we will take a close look at the layer by layer latency against the *baseline*, to try and identify patterns with respect to the size and type of each layer, its location in the neural network, and varience in latency.

4. **Apply combinations of compression techniques**: Based on the results in the previous step we will cherry pick the best agorithm/parameter pairings, with respect to latency reduction for each domain represented in the selected algorithms. We will then apply a composition of these sucessful compression techinques to the models, using the same compression application strategies from *full compression*.

5. **Evaluate combined compression**: We will evaluate latency changes from *full compression* and *baseline*. Of particular interest will be any changes in the individual layer latencies.

5.2.3 Experiment stage 2: Develop optimisation interface

These stages concern objectives O6 and O7

1. **Parameterise compression algorithms**: Develop an interface to define the compression algorithm and its (distiller) scheduler settings. This will be a thin layer on top of distillers pre-existing scheduler api, the purpose of which will be to facilitate communication between an external parameter optimisation tool and distiller.

2. **Implement interface**: We will select the most performant aglorithms from Experiment stage 1 and include select parameters in the aforementioned interface. The parameters selection criteria will be based on observed layerwise latency improvement from Experiment stage 1.

3. **Define optimisation metric**: We will define an optimisation metric using an accuracy threshold as a user defined parameter. This will be the optimisation target.

4. **Integrate interface with benchmark suite**: link optimised distiller model generated via the interface with OpenVINO to run benchmarks

5.2.4 Experiment stage 3: Testing Compression optimisation

This stage of research will complete objective O8

1. **Run the optimiser**: Using the interface developed in stage 2 we will utilise a bayseian optimisation strategy (a sweep) that shows how a each parameter correlates with the target metric, this strategy will also reveal parameter importance metrics.

2. **Collect data**: We will gather the data aqcuired in the final experiment and draw conclusions based on the improvement (or lack of it) over the baseline and manual parameter selection. At this point we should be able to answer our hypothesis one way or another.

3. Extended evaulation of the data may be worthwhile, observations of the parameter importance and correlation metrics may shed some light on a direction of future research.

6 Design

In this section we present the design considerations for the software that will be developed in conjunction with this research project.

6.1 Functional Requirements

6.1.1 Inference Agent

The inference agent will function as a dedicated inference benchmarking and reporting tool. It will listen for queued models, run the inference benchmarks with the model and send the results to a queue dedicated to the sender of the model.

Code	Description	Importance
IA.FR1	Subscribe to inference queue	High
IA.FR2	Download ONNX model from queue	High
IA.FR3	Run OpenVINO benchmark with model	High
IA.FR4	Parse benchmark metrics	High
IA.FR5	Pulish metrics to queue corresponding to training agent	HIgh

6.1.2 Training Agent

The training agent will be responsible for applying the compression algorthm to the model and, once compression is complete, it will send the model to an inference queue.

Code	Description	Importance
TA.FR1	Apply compression a to given model based on a set of parameters	High
TA.FR2	Enqueue ONNX model to inference queue with an agent identifier	High
TA.FR3	Await the return of the benchmarking metrics via a dedicated queue	High
TA.FR4	Send metrics to Weights and Baises for recording	High

6.1.3 Optimiser Interface

The optimiser interface is a fairly simple program that is purely concerned with constructing tweaked OpenVINO scheduler definitions based on concise sets of parameters, its purpose is to

abstract away many parameters that we do not want to change run to run during experiments.

Code	Description	Importance
OI.FR1	Provide an interface for pruning parameters to be defined	High
OI.FR2	Construct an OpenVINO schedule based on some predefined defaults and the passed parameters	High
OI.FR3	Produce a .yaml file defining the schedule	Medium
OI.FR4	Log the .yaml schedule for future reuse	Low

6.2 Optimisation and Benchmarking suite

Figure 13 shows an entity relationship diagram to depict the benchmark and optimisation suite, either a hand crafted initial set of paramaters are provided to a *Training Agent* or they could be randomly initialised at the start of the optimisation cycle. The training agent will use the optimiser interface each time new compression parameters are passed to it, this will construct the appropriate scheduler that the training agent will utilise. This diagram also applies for the first set of experimental benchmarks (without an optimiser), in this case the *Training Agent* will only train once with the specified compression parameters, and the cycle would terminate when the metrics are sent to Weights and Biases.

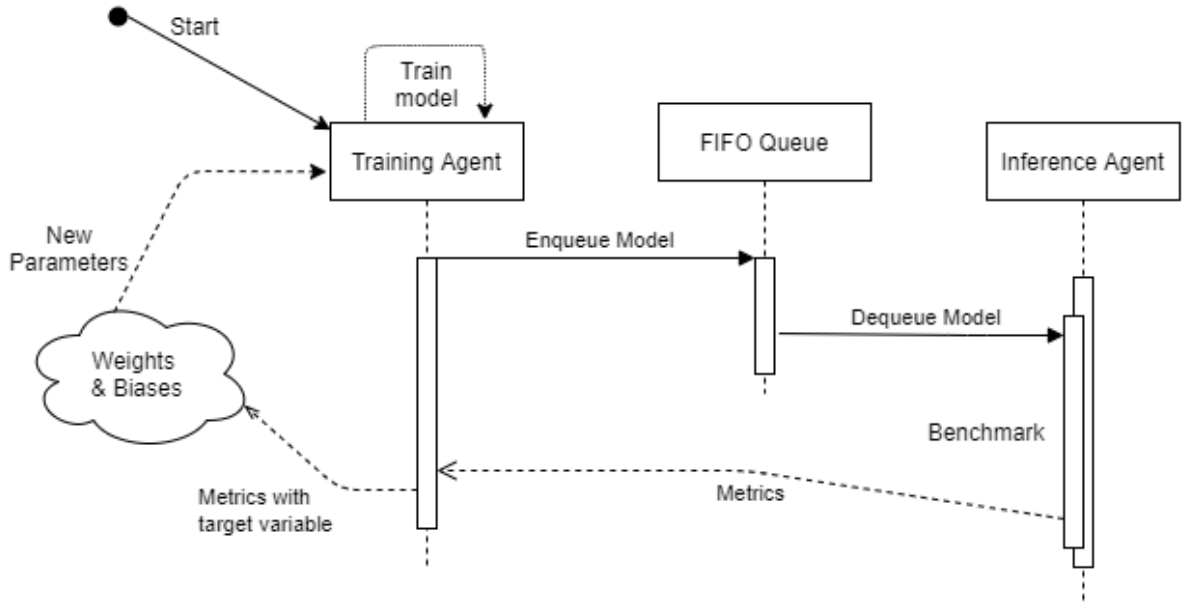

Figure 13: Parameter optimisation ER diagram

Using this system we could theoretically benefit from adding new *Training Agents* to the pool of agents until the consumers *Inference Agent* are unable to clear the queue. We intend to utilise at least two GPU, and one CPU *Training Agents*, with a single *Inference Agent* using the NCS

7 Project Management

7.1 Plan

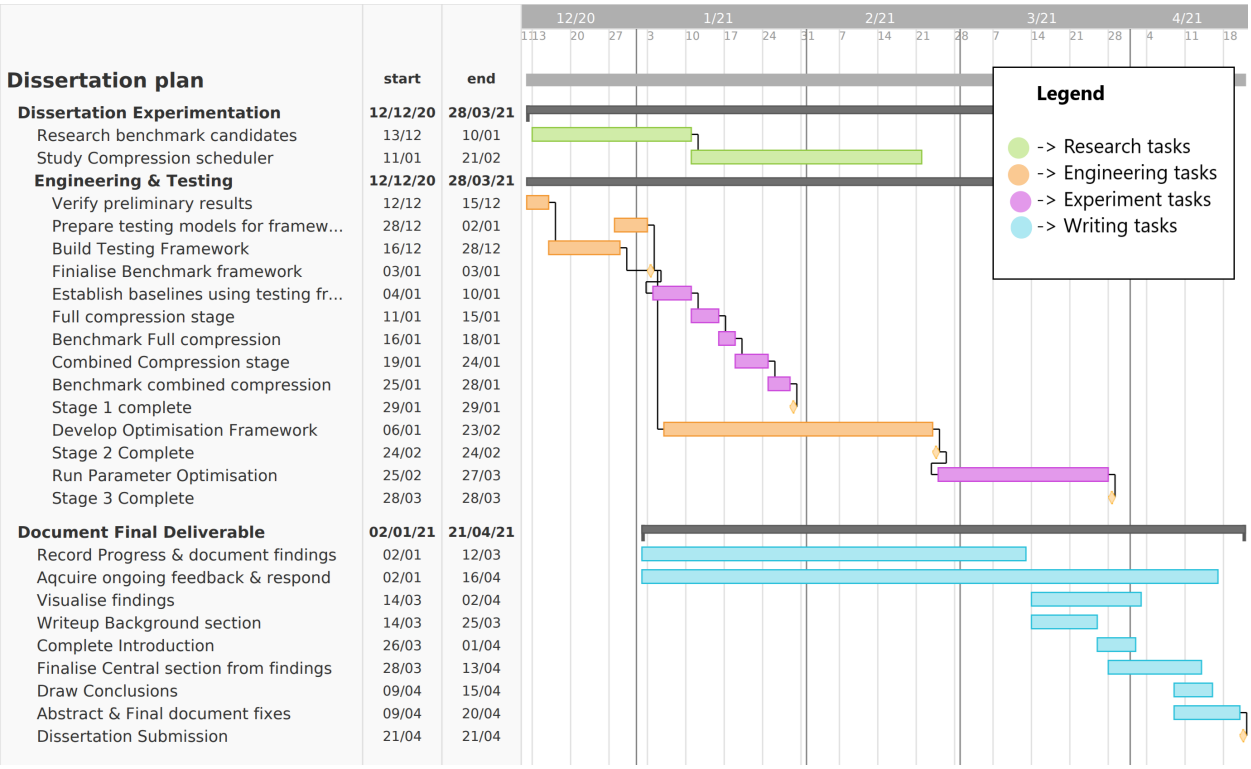

Figure 14: Project gantt chart

7.2 Risk Analysis

The following displays the anticipated risks with their accompanying mitigation strategies.

R1 Cannot compress model in Distiller (see Section 4).

- Investigate alternative compression libraries.

R2 Issues with ONNX compatability.

- Investigate alternative intermediate representations.
- Manually export Distiller model to PyTorch and then directly to OpenVINO.

R3 Bottleneck on time to train models.

- Add additional local Training agents (see Section 6.2).
- Scale the experimental model size down.
- Cloud training agents could be added.

R4 Bottleneck on time to benchmark inference.

- Investigate acquisition of a second NCS.

R5 Time management issues.

- Make use of the project plan to properly manage deadlines.
- Reduce development complexity (remove parallel training agents)

R6 Complexity of parameterising compression algorithm too high.

- Reduce scope of complexity by removing layer selection from the programatic definition and using a static layer definition.

R7 Hardware failure

- In the event of training agent hardware failure cloud resources can be accessed to perform training, the design plan is flexible to this.
- In the event of the NCS failing either a replacement will be sought out, or we could move inference off an edge environment and seek to continue latency optimisation there.

Likelihood	Near Certainty ∼90%			R1		
	Highly Likely ∼70%			R2		
	Likely ∼50%		R3	R5	R6	
	Low likelihood ∼30%					
	Not Likely ∼10%			R4	R7	
		Negligible	Minor	Moderate	Serious	Critical
		Impact of Non-Mitigated Risk				

7.3 Professional, Legal, Ethical, & Social issues

This dissertation will not use any participants in any experiments so there are no ethical or legal issues concerning the safety or well being of participants.

Care will be taken during the course of this dissertaion to maintain a professional standard of communication regarding this work with all contemporaries.

All software produced in the course of this work will be open source and licensed under the Apache License 2.0, this is in compliance with the existing Apache License 2.0 that is already in place on the Distiller and OpenVINO repositories if any modifications are made to the software of those respective libraries. Redis is licensed under the BSD license, Weights and Biases provides a free academic and open source license, this has been applied for and acquired.

The broader technological and social issues that this work may or may not play a part in are unknown, the field of machine learning is advancing at a tremendous pace (at the time of writing), and while it is vital to consider these implications they go far beyond the scope of this work. Any impact this dissertation will be just a miniscule part of a much larger technological revolution in which very few (if any) understand the full depth of the implications of their individual contributions.

A Back matter

A.1 References

References

[1] 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).

[2] 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).

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

[4] 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).

[5] 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,

[6] 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).

[7] 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).

[8] ——, "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).

[9] 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).

[10] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. (Apr. 24, 2015). "Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition." arXiv: 1412.6553 [cs], [Online]. Available: http://arxiv.org/abs/1412.6553 (visited on 11/23/2020).

[11] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating Very Deep Convolutional Networks for Classification and Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 10, pp. 1943–1955, Oct. 2016, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2015.2502579.

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

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

[14] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage," p. 8,

[15] N. Strom. (1997). "Phoneme probability estimation with dynamic sparsely connected artificial neural networks," undefined, [Online]. Available: /paper/Phoneme-probability-

estimation-with-dynamic-neural-Strom/a9392b9299972452ea6fbc3c605f76bb1e21ae42 (visited on 11/13/2020).

[16] 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).

[17] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally. (Jun. 4, 2017). "Exploring the Regularity of Sparse Structure in Convolutional Neural Networks." arXiv: 1705.08922 [cs, stat], [Online]. Available: http://arxiv.org/abs/1705.08922 (visited on 11/17/2020).

[18] 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).

[19] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks," p. 14, 2017.

[20] 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).

[21] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. (Oct. 18, 2016). "Learning Structured Sparsity in Deep Neural Networks." arXiv: 1608.03665 [cs, stat], [Online]. Available: http://arxiv.org/abs/1608.03665 (visited on 11/23/2020).

[22] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 2704–2713. [Online]. Available: https://openaccess.thecvf.com/

content_cvpr_2018/html/Jacob_Quantization_and_Training_CVPR_2018_paper.html (visited on 12/02/2020).

[23] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17, New York, NY, USA: Association for Computing Machinery, Feb. 22, 2017, pp. 45–54, ISBN: 978-1-4503-4354-1. DOI: 10.1145/3020078.3021736. [Online]. Available: https://doi.org/10.1145/3020078.3021736 (visited on 12/02/2020).

[24] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '17*, pp. 65–74, 2017. DOI: 10.1145/3020078.3021744. arXiv: 1612.07119. [Online]. Available: http://arxiv.org/abs/1612.07119 (visited on 10/01/2020).

[25] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized Convolutional Neural Networks for Mobile Devices," presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 4820–4828. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Wu_Quantized_Convolutional_Neural_CVPR_2016_paper.html (visited on 12/03/2020).

[26] Y. Gong, L. Liu, M. Yang, and L. Bourdev. (Dec. 18, 2014). "Compressing Deep Convolutional Networks using Vector Quantization." arXiv: 1412.6115 [cs], [Online]. Available: http://arxiv.org/abs/1412.6115 (visited on 12/03/2020).

[27] M. Antonini, T. H. Vu, C. Min, A. Montanari, A. Mathur, and F. Kawsar, "Resource Characterisation of Personal-Scale Sensing Models on Edge Accelerators," in *Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*, ser. AIChallengeIoT'19, New York, NY, USA: Association for Computing Machinery, Nov. 10, 2019, pp. 49–55, ISBN: 978-1-4503-7013-4. DOI: 10.1145/3363347.3363363. [Online]. Available: https://doi.org/10.1145/3363347.3363363 (visited on 12/10/2020).

[28] (). "Google wins MLPerf benchmark contest with fastest ML training supercomputer," Google Cloud Blog, [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).

[29] 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).

[30] 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).

[31] 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).

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

[33] L. Biewald, "Experiment Tracking with Weights & Biases," p. 5,

[34] N. Zmora, G. Jacob, L. Zlotnik, B. Elharar, and G. Novik. (Oct. 27, 2019). "Neural Network Distiller: A Python Package For DNN Compression Research." arXiv: 1910.12232 [cs, stat], [Online]. Available: http://arxiv.org/abs/1910.12232 (visited on 12/10/2020).

[35] J. Snoek, H. Larochelle, and R. P. Adams. (Aug. 29, 2012). "Practical Bayesian Optimization of Machine Learning Algorithms." arXiv: 1206.2944 [cs, stat], [Online]. Available: http://arxiv.org/abs/1206.2944 (visited on 12/09/2020).

[36] M. Zhu and S. Gupta. (Nov. 13, 2017). "To prune, or not to prune: Exploring the efficacy of pruning for model compression." arXiv: 1710.01878 [cs, stat], [Online]. Available: http://arxiv.org/abs/1710.01878 (visited on 12/04/2020).