

Neural network compression is necessary due to storage related issues that often arise on resource constrained 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 various underlying 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[1], [2], not all techniques preserve this density and can result in weight matrices with various degrees of sparsity which in turn have varying degrees of regularity. These techniques, the resulting representations of parameters, and their consequences will be discussed in this section.

0.0.1 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 [3]–[6], but more recent research shows that some pruning methodologies can produce pruned networks with no loss of accuracy [7].

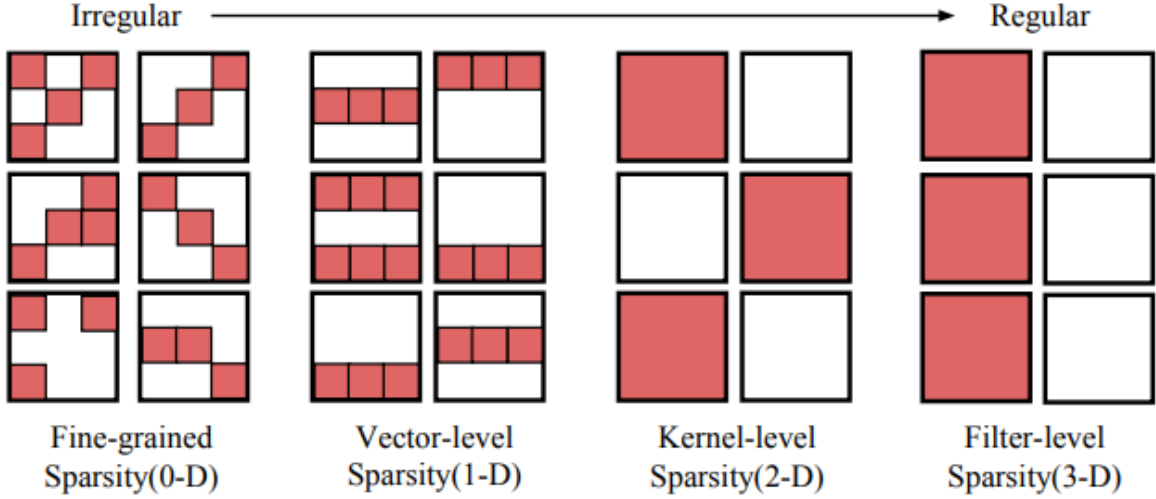


Figure 1: Sparse structures in a 4-dimensional weight tensor. Regular sparsity makes hardware acceleration easier.

(Adopted figure from [8])

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 [8], this is known as the **granularity of sparsity**. Figure 1 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 aforementioned granularities.

The influential paper Optimal Brain Damage by LeCun et al [5] 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 [9], [10], a software solution has been theorized by Han et al [11] that would involve developing a customized GPU kernel that supports indirect matrix entry lookup and a relative matrix indexing format, see Section 0.0.2 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 convolutional filter.

This style of pruning however can have a significant impact on the accuracy of the network, but as demonstrated by Wen et al [12] accelerating networks with very coarse-grained pruning is straightforward because the model is smaller but still dense, so libraries such as BLAS are able to take full advantage of the structure.

0.0.2 Quantisation and Weight Sharing

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 unnecessary, extensive research [13], [14] has shown reducing the precision of parameters can have an extremely small impact on the accuracy.

Fixed-point quantisation is the process of limiting the floating point precision of each parameter 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), Umuroglu et al. [15] used binary quantisation with an FPGA and achieved startling classification latencies ($0.31\mu s$ 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 ??.

Utilising fixed-point and binary quantisation can result 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 [11] 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. 2.

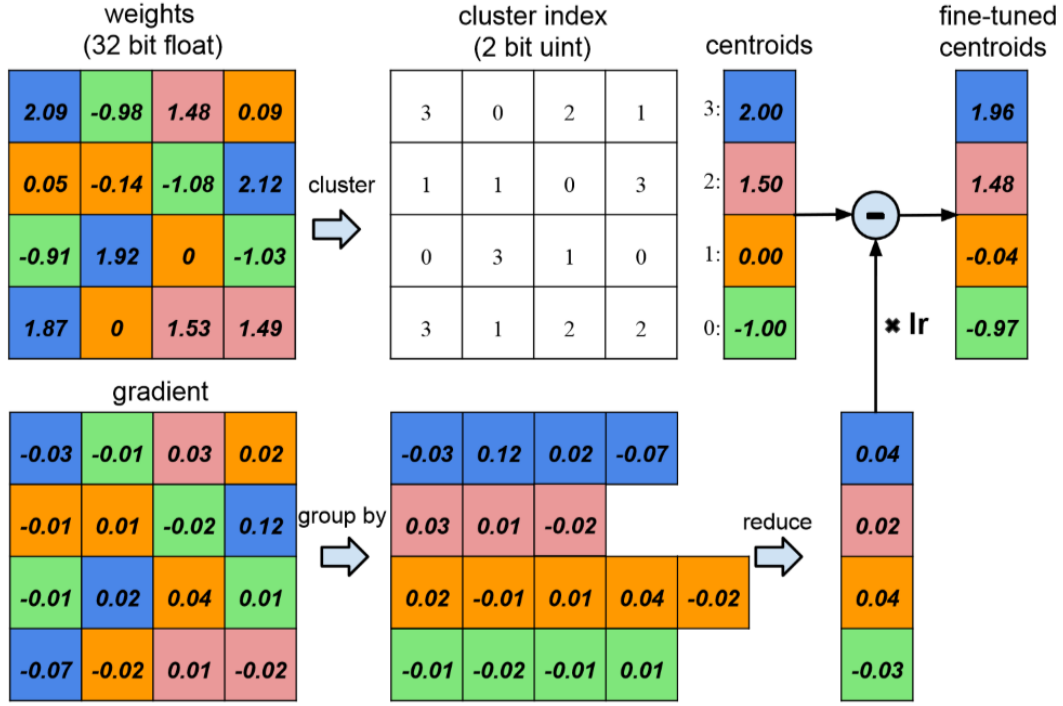


Figure 2: Weight sharing by quantisation with centroid fine-tuning using gradients (Adopted figure from [11])

0.0.3 Distillation

0.0.4 Low-rank Factorization

0.0.5 Network Design Strategies