BitAllocation: A Resource Allocation Algorithm For Fixed-Point Quantization

Adam Wei
University of Toronto
adam.wei@mail.utoronto.ca

Abstract

The growing size and computational complexity of state-of-the-art deep convolutional networks (DCNs) have greatly increased the memory, time, and power requirements of inference in many computer vision applications. Fixed-point quantization is an effective method that can alleviate some of these issues, but comes at the cost of reduced classification accuracy. In an attempt to minimize this accuracy degradation, we developed BitAllocation, a quantization pipeline that can aggressively compress DCNs to fixed-point data types without the need for retraining. Our key insight is to formulate quantization as a variation of the discrete resource allocation problem, where a *budget* of bits is to be allocated across the weights and activations in a way that minimizes the total quantization error. Although this problem is NP-hard, we develop a near linear time algorithm that solves it optimally for practical applications. Using this algorithm and no further retraining, we quantized 7 ImageNet DCNs to an average bitwidth of 5.5-6.25 bits with a 1-3% drop in top-1 accuracy. This corresponded to a 5.51x and 27.5x reduction in model size and cost of multiplications respectively. Although this paper presents an application in machine learning quantization, our algorithm can be used in other fields that involve resource allocation, such as economics, project management, and computer systems.

1 Introduction

In recent years, deep convolutional networks (DCNs) have become adept at solving difficult problems in computer vision, but unfortunately, many of these advances have come at the cost of larger model sizes and increased computational complexity [1, 2]. This is problematic for data centers that process machine learning workloads, mobile devices, and embedded systems where memory, computational resources, and power are limited.

The growing size of neural networks raises more concerns than the immediate issue of storage. In modern hardware, off-chip memory accesses are an order of magnitude slower and two orders of magnitude more power hungry than on-chip accesses [3]. Therefore, the speed and power-efficiency of state-of-the-art DCNs is often hindered by their large memory footprints [4]. A key factor that dictates a network's performance metrics is the data type of its weights and activations. Quantized models that use lower bitwidth data types are typically not as accurate as their full precision counterparts, but they incur less memory traffic, less computational complexity, and can be accelerated in hardware [5, 6]. For modern ML chips, the increase in computational speed is proportional to the reduction in bitwidth [7, 8, 9, 10], making quantized models faster, smaller, and more efficient.

To leverage these advantages, there has been considerable research in aggressive quantization for machine learning applications [5, 11, 12, 13]. Most state-of-the-art techniques incur accuracy loss, which can be mitigated by using more expressive quantization schemes, such as floating point [14], k-means [11], or dictionary-based mappings. While effective, these methods can be difficult to accelerate in hardware. Contrastingly, fixed-point data types can provide tremendous advantages for

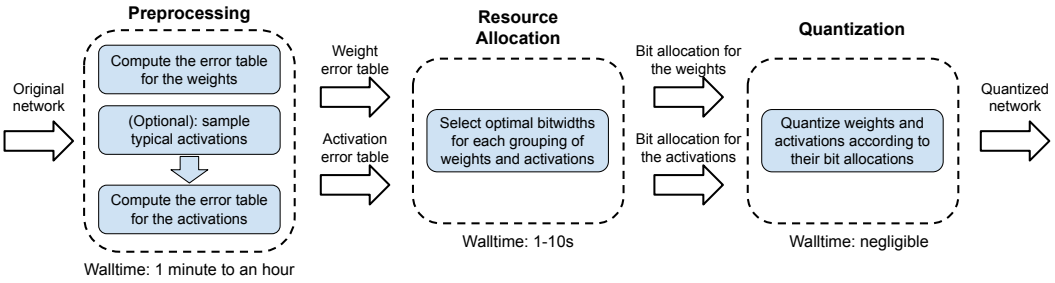


Figure 1: The BitAllocation pipeline consists of 3 stages: preprocessing, resource allocation, and quantization.

hardware since they can be implemented with bitshifts and integer operations. For instance, INT8 operations save up to 30x more energy and 116x more area when compared to FP32 [15]. The caveat is that fixed-point data types have limited precision and dynamic range, which often necessitates iterative retraining to recover accuracy [16, 17]. This hinders rapid model deployment since training often requires time, hardware, and fully labelled datasets. Therefore, there is a need for aggressive fixed-point quantization techniques that do not require retraining.

To address this, we present BitAllocation, a fast, effective, and configurable quantization pipeline. Our key insight is to formulate quantization as an optimization problem where a user-specified budget of bits is to be distributed across a model's weights and activations in a way that minimizes the total quantization error. We developed a novel resource allocation algorithm to solve this problem and use it in our BitAllocation pipeline, which consists of the following 3 stages (see Figure 1).

Preprocessing: The preprocessing stage computes the quantization error for each grouping[1] of weights and/or activations at all possible bitwidths. These error values are stored in *error tables* and passed onto the next stage. The idea is that in order to select optimal bitwidths across an entire model, we need to first understand the effect of quantization on each individual grouping.

Resource Allocation: This stage identifies the optimal bitwidth for each grouping that will minimize the total quantization error across the entire network while allocating less than a user-specified *budget* of bits. This is an NP-hard variation of the well-studied Resource Allocation problem which seeks to optimally allocate a discrete finite resource to a set of activities so as to minimize an objective [18]. In our formulation, the finite resource is the budget of bits, each activity is a grouping of weights or activations, and the objective is to minimize the total quantization error across the DCN. We present a novel algorithm that can solve this NP-hard problem for the vast majority of real world applications.

Quantization: The last stage simply quantizes each grouping of weights and activations to fixed-point according to the bitwidths selected by the Resource Allocation algorithm.

A major benefit of our pipeline is its configurability. The second stage uses the error tables to solve the optimization problem, but it does not care about the tables' dimensions or how their values were computed. Hence, the preprocessing stage is highly configurable and can compute the error tables using practically any quantization method, granularity, and metric for error. To limit the scope of our investigation, we choose to focus on quantizing ImageNet models with 3 different error metrics, channel-wise granularity, and fixed-point quantization. These experiments show that BitAllocation can reduce the size and cost of multiplications of DCNs by 5.51x and 27.5x respectively, with a 1-3% drop in top-1 accuracy. Even with the limited scope of our initial explorations, we have already produced some state-of-the-art results; thus, we see great potential in our pipeline and room for other researchers to build upon our work.

The main contributions of this paper can be summarized as follows:

1. We present a novel algorithm that solves the Resource Allocation problem for monotonic objective functions in real world applications. Although this paper presents an application in machine learning, the algorithm is far more general and can be used in other fields such as economics, project management, bandwidth allocation, and computer systems.

[1]The term 'grouping' refers to a set of weights or activations at the chosen granularity. For example, a grouping of weights at a channel-wise granularity would be a channel of weights.

2. We present BitAllocation, a highly configurable and promising pipeline for machine learning quantization that other researchers can improve upon.
3. We show that fixed-point quantization to at least 5.5-6 bits is possible without any retraining.

2 Related work

Fixed-point quantization Fixed-point DCNs are generally implemented in-training or post-training. In-training approaches train DCNs with low precision fixed-point arithmetic and are capable of achieving high classification accuracy [19, 12, 20, 21], but they require time and hardware for training. The same is true for post-training approaches that require finetuning [22].

In the spirit of fast model deployment, this paper will focus on post-training approaches that do not require training. Since fixed-point data types are extremely constrained, previous papers in this area provide an extra degree of freedom by allowing different layers to use different bitwidths [23, 24]. Judd et al. [24] and Hwang et al. [25] were able to quantize each layer/channel of a model to a semi-optimal bitwidth via an exhaustive search, while Lin et al. [13] used an optimization strategy based on the signal-to-quantization-noise ratio (SQNR). More recently, Banner et al. [26] and Choukroun et al. [27] were able to compress ImageNet models to an average of 4 bits without retraining. Both papers developed approaches that minimized a model's MSE, but their specific implementations differed. Unfortunately, the techniques used by Banner et al. [26] relied on the flexibility of linear quantization and are not directly transferable to a fixed-point implementation. To the best of our knowledge, the current state-of-the-art for fixed-point quantization without retraining is roughly 6-8 bits [13, 28, 29]. We draw inspiration from Lin et al. [13] and Banner et al. [26] to explore MSE and SQNR as potential error metrics for the BitAllocation pipeline. As demonstrated by *Deep Compression* [11], other common methods that reduce model complexity and size, such as pruning or hashing, are orthogonal to our approach and can be applied in tandem.

Resource Allocation The resource allocation problem has been thoroughly studied with its most active period of research between 1950-1980 [30, 31, 32, 33]. Its theory, common algorithms, and applications are documented in Ibaraki and Katoh's 1988 textbook [18]. According to their terminology, this paper is solving a variation of the discrete resource allocation problem for separable monotonic objective functions. This problem is known to be NP-hard, however polynomial time algorithms exist if the objective function is entirely concave [18], or concave on subintervals [34]. Unfortunately, these algorithms cannot be applied to our pipeline or other real world problems since most functions, such as the quantization error of a layer at different bitwidths, are not concave.

3 DCN fixed-point quantization

A value X can be mapped to an n-bit integer in the range $[-2^{n-1}, 2^{n-1}-1]$ according to a stepsize q as shown in equation (1), where q must be an integer power of 2. The quantized value of X can be recovered as $Q(X, n, q) = q \cdot Int(X, n, q)$.

$$Int(X, n, q) = \begin{cases} -2^{n-1} & X \leq -2^{n-1} \cdot q \\ \text{round}(\frac{X}{q}) & -2^{n-1} \cdot q < x < (2^{n-1}-1) \cdot q \\ 2^{n-1}-1 & (2^{n-1}-1) \cdot q \leq X \end{cases} \tag{1}$$

The most important step in developing good post-training quantization algorithms is choosing the stepsize and bitwidth for each grouping of weights and activations. Optimal stepsizes and clipping methods represent active areas of research [26, 35, 28], but they are not the focus of our paper. As such, we chose to employ a simple clipping method where the stepsize of each grouping is selected as the minimum value required to prevent overflow, rounded to the nearest power of 2.

Batch Normalization (BN) Layers BN layers are commonly used in DCNs to accelerate training, but during inference, they simply scale and shift the output of the previous layer [36]. In this paper, we fuse the BN layers into their previous layers before applying the BitAllocation pipeline to reduce cascading quantization error [5, 13, 28, 37].

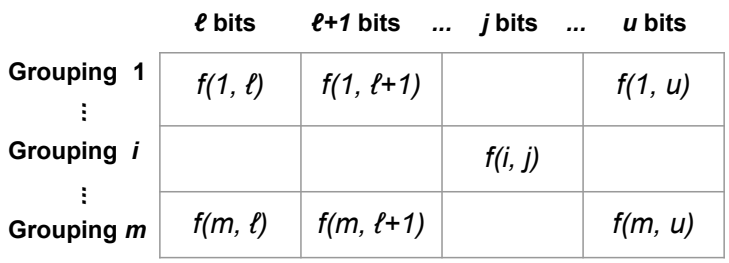


Figure 2: A graphical depiction of the error table. The value in the i^{th} row and j^{th} column is the quantization error from converting grouping *i* to *j*-bits, as measured by the objective function *f*.

4 BitAllocation Pipeline

The BitAllocation pipeline consists of 3 stages: preprocessing, resource allocation, and quantization. We only discuss the first 2 stages in depth since the last stage is mostly self-explanatory. Although this section defines the optimization problem in the context of DCN quantization, the algorithm, concepts, and lemmas we present are transferable to the general discrete resource allocation problem.

4.1 Stage 1: Preprocessing

The preprocessing stage of the pipeline generates an error table for the weights and/or activations separately. Suppose that a model has *m* groupings of weights, indexed from 1 to *m*, and that each grouping can be quantized to some bitwidth *b*. If *b* is bounded by a lower bound, *l*, and an upper bound, *u*, then there are $n = u - l + 1$ possible bitwidths for each grouping. These bounds exist to accommodate potential hardware constraints since many accelerators can only support arithmetic up to a certain maximum bitwidth. Lastly, let $f(i, j)$ be an objective function, such as MSE or SQNR, that computes the quantization error from converting grouping *i* to its *j*-bit fixed-point presentation.

With this notation, the error table for the weights is an *m* by *n* matrix where the rows represent groupings and the columns represent bitwidths. As depicted in Figure 2, the element in the i^{th} row and j^{th} column is $f(i, j)$. The error table for the activations is similarly defined, however its dimensions may differ since the weights and activations can have a different number of groupings.

The weight error table is straightforward to compute since the weights of a model are usually known. Simply quantize each grouping to each possible bitwidth and record the quantization errors in the error table. On the other hand, the activations need to be sampled with a small dataset of typical inputs before its error table can be computed. If a small dataset is unavailable, then the activations can be left unquantized or quantized to a set bitwidth. We explore the second option further in Section 5.3. Since the error tables for the weights and activations are computed independently of one another, they can use different granularities, datatypes, clipping methods, and objective functions. This gives our pipeline more flexibility to target different architectures or specific attributes of a model.

4.2 Stage 2: Resource Allocation

A lower bitwidth DCN should approximate its full precision counterpart by minimizing the amount of quantization error introduced during model conversion. This motivates the following optimization problem: what is the optimal bitwidth for each grouping of weights and activations that minimizes the total quantization error across the network, while allocating less bits than a user-specified *budget*? Note that this problem must be solved separately for both the weights and activations. To illustrate the problem, consider AlexNet [38], a model with 8 layers. Suppose we want to quantize the weights at a layer-wise granularity to 6 bits per layer. Then our optimization problem aims to distribute a *budget* of 48 bitwidths across the 8 layers of weights in a way that minimizes the quantization error. This process would be repeated separately for the activations.

Let's introduce some notation to formally define the problem. Suppose we are quantizing a DCN with *m* groupings. Let $B = [b_1, \ldots, b_i, \ldots b_m]$ be an *m* dimensional vector that represents a *bit allocation*. Each element b_i is an integer that indicates the quantized bitwidth of grouping *i*. As stated in Section

4.1, $l \leq b_i \leq u$ for all *i*, and f is an objective function that computes quantization error. Then, the optimization problem can be written concretely as:

$$\text{find } \arg\min_{B} \sum_{i=1}^{m} f(i, b_i), \text{ s.t. } \{b_i \in \mathbb{Z} \mid l \leq b_i \leq u\} \text{ and } \sum_{i=1}^{m} b_i \leq budget \tag{2}$$

The *budget* controls the degree of quantization and limits the memory footprint of the model. $\sum_{i=1}^{m} f(i, b_i)$ is the minimization objective and represents the total quantization error introduced during model conversion. As previously mentioned, this problem should be solved separately for the weights and activations since their error tables are different. In other words, the output of this stage consists of up to 2 bit allocations: one for the weights and one for the activations.

4.2.1 The BitAllocation algorithm

The optimization problem (2) is known to be NP-hard for non-concave objective functions and can be solved in exponential time with an exhaustive search [18]. Unfortunately, this is far too slow, especially at fine granularities where the number of groupings may be very large. Instead, we introduce the BitAllocation algorithm, which uses binary search to iteratively discover improving bit allocations until it reaches the optimal solution.

Previous algorithms [18] that solve this problem for concave objective functions define the following expression, where λ is a real number known as the Lagrange multiplier.

$$\mathcal{L}(B, \lambda) = \sum_{i=1}^{m} f(i, b_i) - \lambda \sum_{i=1}^{m} b_i \tag{3}$$

The fundamental idea is to find a correct value for the Lagrange multiplier, which we denote as λ_{budget}, such that the bit allocation which minimizes $\mathcal{L}(B, \lambda_{budget})$ uses the same number of bits as *budget*. The algorithms differ in how they identify λ_{budget}, but they all rely on the properties of concave functions. This is not possible in our case since f need not be concave. Instead, we introduce a new concept known as *the score*, which is conceptually similar to $\mathcal{L}(B, \lambda)$. For a given bit allocation B and a parameter *k*, $\{k \in \mathbb{R} \mid 0 \leq k \leq 1\}$, the score function, $S_k(B)$, computes a weighted sum of the total number of bits in *B*, and the total quantization error.

$$S_k(B) := k \sum_{i=1}^{m} b_i + (1 - k) \sum_{i=1}^{m} f(i, b_i) \tag{4}$$

For a given value of *k*, define B_k to be the bit allocation that minimizes S_k, and T_k to be the sum of the bitwidths in B_k. Concretely, $B_k = [b_{k,1}, b_{k,2}, ..., b_{k,m}] = \arg\min_B S_k(B)$ and $T_k = \sum_{i=1}^{m} b_{k,i}$. The underlying principle of our approach is the same as the Lagrange multiplier method: if we can discover a value for k_{budget} s.t. $T_{k_{budget}} = budget$, then $B_{k_{budget}}$ is the optimal solution to the resource allocation problem. The key insight of our score function is to bound the possible values of *k*, which enables us to binary search for k_{budget}. A summary of our algorithm is provided below.

For the weights and activations separately:

1. Choose an arbitrary value of *k* between 0 and 1.
2. Repeat until $T_k = budget$
   (a) For the given value of *k*, find B_k and compute the value of T_k.
   (b) Binary search for an improved value of *k* s.t. the value of T_k approaches *budget*.
      i. If $T_k < budget$, decrease the value of *k*
      ii. If $T_k > budget$, increase the value of *k*
      iii. If $T_k = budget$, break from the loop
3. Halt. The current value of *k* is k_{budget}. Return $B_{k_{budget}}$ as the optimal solution.

Unfortunately, k_{budget} may not always exist. This is not surprising since the problem is known to be NP-hard. In this case, our algorithm will produce the optimal solution to a slightly different formulation of the problem that uses a different value for *budget*. In practise, we do not need to worry about this edge case since k_{budget} almost always exists (see Table Section 6).

The remainder of this section will provide some more insight into the score function, a subroutine that can be used in Step 2a) to find B_k, a proof of correctness, brief runtime analysis, and pseudocode.

4.2.2 The score function

In fixed-point quantization, larger bitwidths will never increase quantization error since they provide more precision. Thus, there is a tradeoff between the number of bitwidths available in *budget* and the minimum overall error. The parameter *k* models this tradeoff by determining the degree to which each term of the score should be punished. If our goal is to minimize the score, then larger values of *k* will encourage less bits to be allocated by punishing larger bitwidths and vise-versa. To illustrate this more clearly, consider the two extreme values of *k*: $k = 1$ and $k = 0$.

$$S_1(B) = 1 \cdot \sum_{i=1}^{m} b_i + (1-1) \sum_{i=1}^{m} f(i, b_i) = \sum_{i=1}^{m} b_i \tag{5}$$

$$S_0(B) = 0 \cdot \sum_{i=1}^{m} b_i + (1-0) \sum_{i=1}^{m} f(i, b_i) = \sum_{i=1}^{m} f(i, b_i) \tag{6}$$

S_1 only penalizes the number of bits allocated; therefore, S_1 is minimized by allocating as few bits as possible. S_0 only penalizes quantization error; therefore, S_0 is minimized by allocating the largest possible bitwidths. Equations (5) and (6) show that as *k* decreases, the weight of the quantization error term will increase and more bits will need to be allocated to minimize the score. Between these 2 extreme values for *k*, there almost always exists a value for k_{budget}.

Minimizing the score In order to run our algorithm efficiently, we need a fast subroutine that can find B_k for all *k*. To accomplish this, we introduce the concept of a grouping's individual score. Suppose that grouping *i* is quantized to a bitwidth b. Then grouping *i*'s individual score is defined as

$$s_{k,i}(b) := kb + (1-k) f(i, b). \tag{7}$$

We can show that $S_k(B) = \sum_{i=1}^{m} s_{k,i}(b_i)$. Thus if $B_k = [b_{k,1}, \dots b_{k,m}]$ minimizes S_k, then $b_{k,i} = \arg\min_b s_{k,i}(b) \ \ \forall \ \ i \in [1, m]$. $b_{k,i}$ can be determined for each grouping *i* by computing $s_{k,i}(b)$ for all legal values of *b* and tracking the best option. As shown in Section 4.2.5, this subroutine can be implemented using the error table in *O(mn)* time, where *m* and *n* are dimensions of the table.

4.2.3 Proof of correctness

We will be proving two lemmas that lead naturally into a proof of correctness for our algorithm.

Lemma 4.1. *Let $B_k = \arg\min_B S_k(B)$ and $T_k = \sum_{i=1}^{m} b_{k,i}$. If $T_k = budget$, then B_k is a solution to the optimization problem.*

Proof. Let $B' = [b'_1, ..., b'_m]$ s.t. $\sum_{i=1}^{m} b'_i \leq T_k$. Since B_k minimizes S_k, the following must be true:

$$\begin{aligned} S_k(B_k) &\leq S_k(B') \\ k \sum_{i=1}^{m} b_{k,i} + (1-k) \sum_{i=1}^{m} f(i, b_{k,i}) &\leq k \sum_{i=1}^{m} b'_i + (1-k) \sum_{i=1}^{m} f(i, b'_i) \\ k \cdot T_k + (1-k) \sum_{i=1}^{m} f(i, b_{k,i}) &\leq k \cdot T_k + (1-k) \sum_{i=1}^{m} f(i, b'_i) \\ \implies \sum_{i=1}^{m} f(i, b_{k,i}) &\leq \sum_{i=1}^{m} f(i, b'_i) \end{aligned} \tag{8}$$

According to (8), B_k introduces at most the same amount of quantization error as any other bit allocation that uses T_k bits or less; i.e B_k is at least as good as the optimal solution. Thus, if $T_k = budget$, then B_k is a solution to the optimization problem. □

Lemma 4.2. *Assume that there exists a real value of k_{budget} s.t. $T_{k_{budget}} = budget$. Then binary search can be used to find the value of k_{budget}.*

Proof. Let *Total(k)* be a function, defined as $Total(k) := \sum_{i=1}^{m} b_{k,i}$, that computes T_k from *k*. Binary search can be used to find k_{budget} if *Total* decreases monotonically with respect to *k*. To prove this, we show that if *k* increases, each individual bitwidth $b_{k,i}$ will never increase. Note that if $b_{k,i}$ is already the maximum allowed bitwidth, then $b_{k,i}$ will not increase by definition.

Since B_k minimizes S_k, then $b_{k,i} = \arg\min_b s_{k,i}(b)$ as shown in Section 4.2.2. Let ϵ be a small positive value such that $k + \epsilon \leq 1$. For a grouping *i*, assume that the bitwidth $b_{k+\epsilon,i}$ minimizes $s_{k+\epsilon,i}$ and that $b_{k+\epsilon,i} > b_{k,i}$. We will show that this assumption is impossible by contradiction.

In fixed-point quantization, a larger bitwidth will always incur less or equal quantization error. Since we assumed that $b_{k+\epsilon,i} > b_{k,i}$, then $\Delta b := b_{k+\epsilon,i} - b_{k,i} > 0$ and $\Delta f := f(i, b_{k+\epsilon,i}) - f(i, b_{k,i}) \leq 0$. Since we defined $b_{k+\epsilon,i}$ as the bitwidth that minimizes $s_{k+\epsilon,i}$, then $s_{k+\epsilon,i}(b_{k+\epsilon,i}) \leq s_{k+\epsilon,i}(b_{k,i})$. Expanding and simplifying both sides of this inequality produces the following expression.

$$k\Delta b + (1-k)\Delta f \leq \epsilon(\Delta f - \Delta b) \tag{9}$$

Since $\epsilon > 0$, $\Delta b > 0$, and $\Delta f \leq 0$, then $\epsilon(\Delta f - \Delta b) < 0$. Therefore,

$$k\Delta b + (1-k)\Delta f < 0. \tag{10}$$

Similarly, $b_{k,i}$ is the bitwidth that minimizes $s_{k,i} \implies s_{k,i}(b_{k+\epsilon,i}) \geq s_{k,i}(b_{k,i})$. Expanding and simplifying terms in this inequality produces the following expression.

$$k\Delta b + (1-k)\Delta f \geq 0 \tag{11}$$

We have now arrived at a contradiction; inequalities (10) and (11) are inconsistent. Thus, our original assumption that $b_{k+\epsilon,i} > b_{k,i}$ must be false. This implies that the bitwidth of any individual grouping, and hence the value of *Total*, will never increase with *k*. Therefore, *Total* is a monotonically decreasing function and we can binary search with respect to T_k to find k_{budget}. □

Putting everything together In the proof of Lemma 4.2, we assumed that k_{budget} exists. Although we can generate contrived edge cases where this assumption is invalid, the experiments in Section 6 show that k_{budget} exists in practise. By Lemma 4.2, we can use binary search to find k_{budget} and by Lemma 4.1, $B_{k_{budget}}$ solves the optimization problem since $T_{k_{budget}} = budget$. Therefore, this algorithm finds the optimal solution in practical cases where k_{budget} exists.

4.2.4 Runtime analysis

The preprocessing stage is the most time consuming part of the pipeline and takes anywhere from a few seconds to an hour. The walltime of this stage depends on the size of the model and the dataset, the configuration of the pipeline, and the available hardware. Fortunately, the error tables only need to be computed once for each configuration and can be reused during future runs of the pipeline.

After preprocessing, the algorithm runs iterations of binary search to find the optimal solution. The time complexity of each iteration is dominated by the subroutine which minimizes S_k. Since this subroutine evaluates $s_{k,i}$ for all groupings at all possible bitwidths, it runs in *O(mn)* time, where *m* is the number of groupings and *n* is the number of possible bitwidths. The exact number of iterations required to find k_{budget} is difficult to analyze, but we can identify some rough upper bounds and provide empirical results. We implemented the value of *k* as a double in our code; thus, we need to run at most 53 iterations of binary search as the maximum precision of a double is 53 bits (52 mantissa bits and 1 implied bit). In practise, it only takes 10-30 iterations (see Section 6). Thus in real world situations, our algorithm runs in near linear time with respect to the size of the error table.

4.2.5 Pseudocode

5 Experiments

In order to use the BitAllocation pipeline, we need to choose a quantization error metric, denoted by *f*, for the preprocessing stage. This choice is entirely up to the user and can be varied to capture different attributes of a model. In this paper, we define and investigate 3 choices for *f*, which we call the MSE-objective, the SQNR-objective, and the loss-objective.

Algorithm 1 BitAllocation

procedure
  $m \leftarrow$ number of groupings, $n \leftarrow$ number of possible bitwidths for each grouping
  $l \leftarrow$ bitwidth lower bound, $u \leftarrow$ bitwidth upper bound ▷ Recall that $n = u - l + 1$
  $F \leftarrow$ float$[m][n]$
  Preprocessing: Pass through a dateset to compute the error table *F*.

  $B \leftarrow$ int$[m]$, $lo \leftarrow 0$, $hi \leftarrow 1$ ▷ track the best bit allocation in the *B* array
  repeat ▷ binary search for k_{budget}
    $k \leftarrow (lo + hi)/2$
    $B \leftarrow minimize\_score(k)$
    $T_k \leftarrow sum(B)$
    if $T_k > budget$ **then** $lo \leftarrow k$
    else if $T_k < budget$ **then** $hi \leftarrow k$
  until $T_k = budget$
End of algorithm: *B* contains the optimal bit allocation for the given *budget*

function minimize_score(k) ▷ subroutine from Section 4.2.2
  for all groupings *i* in $[1 : m]$ **do** ▷ find $B[i] = \arg\min_b s_{k,i}(b)$ for all *m* groupings
    $min\_score \leftarrow \infty$, $b_{k,i} \leftarrow -1$
    for b in $[l : u]$ **do**
      $s_{k,i} \leftarrow k \cdot b + (1 - k) \cdot F[i][b]$
      if $s_{k,i} < min\_score$ **then**
        $min\_score \leftarrow s_{k,i}$
        $b_{k,i} \leftarrow b$
    $B[i] = b_{k,i}$
  return B

5.1 Metrics for quantization error

MSE-objective Let $MSE(X, n)$ be a function that computes the MSE from quantizating an input *X* to *n* bits. We define the MSE-objective in equation (12), where X_i represents the i^{th} grouping of weights or activations and *j* represents the bitwidth.

$$f_{MSE}(i, j) := MSE(X_i, j)^2 \tag{12}$$

The additional square term is optional, but produces marginally better results at lower bitwidths. Without it, a disproportionate amount of groupings are given the minimum or maximum possible bitwidth. This performs poorly since the low bitwidth groupings may become bottlenecks for the model's accuracy. With the square term, groupings with large MSE become extremely costly for the minimization objective, which forces BitAllocation to distribute bitwidths more uniformly. As shown in Figure 3, this can reduce bottlenecks and improve accuracy.

SQNR-objective The signal to quantization noise ratio (SQNR) is a common metric used to measure the quality of an approximation [13]. Let $SQNR(X, n)$ be a function that computes the SQNR from quantizating an input *X* to *n* bits. We define the SQNR-objective as

$$f_{SQNR}(i, j) := SQNR(X_i, j)^{-2}, \tag{13}$$

where X_i represents the i^{th} grouping of weights or activations and *j* represents the bitwidth. The SQNR term is inverted to ensure that f_{SQNR} increases with quantization error, and squared for the same reasons discussed for the MSE-objective.

Loss-objective Previous works have used the gradient of the loss function to estimate the change in the loss from pruning individual weights [39, 40]. We draw inspiration from their work and use the gradients to estimate the change in the loss from quantization. The motivating idea is that lower change in loss from quantization should result in better accuracy. The change in loss from quantizing the i^{th} grouping of weights or activations (denoted X_i) to *j*-bit floating-point can be estimated by

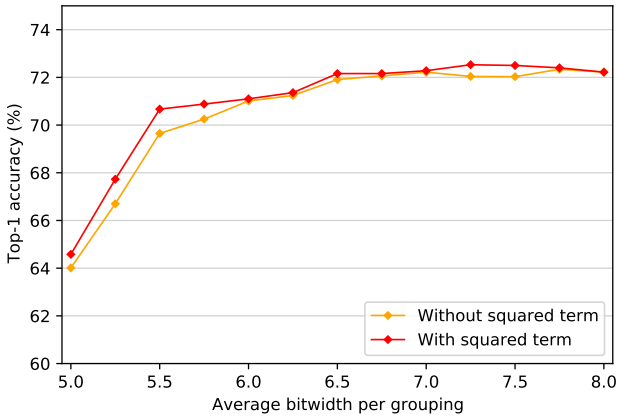


Figure 3: ResNet50's top-1 accuracy with and without the squared term

the function ΔL, where $\mathcal{L}$ represents the cross entropy loss function, $\odot$ is the element-wise matrix multiplication operator, and E returns the arithmetic mean of a matrix's elements.

$$\Delta L(i,j) := \mathrm{E}(|\frac{\partial \mathcal{L}}{\partial X_i} \odot (Q(X_i,j) - X_i)|), \tag{14}$$

Using ΔL directly for f in the BitAllocation algorithm yields poor results since the magnitude of the gradients vary dramatically across groupings. To address this issue, we normalize the mean of the ΔL values for each grouping. Concretely,

$$\Delta L_{norm}(i,j) := \frac{\Delta L(i,j)}{\mu_i}, \text{ where } \mu_i = \frac{1}{u-l+1}\sum_{j=l}^{u} \Delta L(i,j). \tag{15}$$

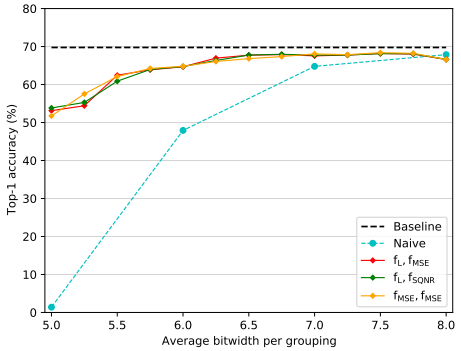
We use this definition of $\Delta L_{norm}(i,j)$ to define our loss-objective as $f_L(i,j) := \Delta L_{norm}(i,j)^2$. The term is squared for the same reasons discussed for the MSE-objective.

5.2 Results & evaluation

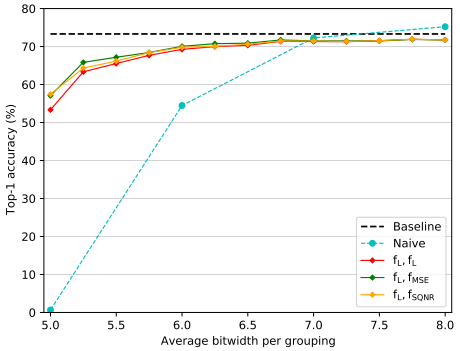
The BitAllocation pipeline was evaluated on 7 ImageNet DCNs using the ILSVRC2012 validation set and the PyTorch framework. To prevent overtuning, a different dataset was used to compute the activation error tables. We used a channel-wise granularity, a lower bitwidth bound of $l = 2$, an upper bitwidth bound of $u = 8$, and 3 different optimization objectives: f_{MSE}, f_{SQNR}, and f_L. The notation (f_W, f_A) represents a configuration that uses f_W and f_A for the weights and activations respectively. For example, the configuration (f_{MSE}, f_{SQNR}) uses the MSE-objective to quantize the weights and the SQNR-objective to quantize the activations.

The results of our main experiments in Figure 4 and Table 1 show that the BitAllocation pipeline significantly outperforms the naive quantization approach. Without any retraining, BitAllocation could quantize most models to 5.5-6 bits while staying within 1-3% of the baseline top-1 accuracy. This corresponded to a 5.51x reduction in model size and a 27.5x reduction in the cost of multiplications, making BitAllocation extremely competitive with the state-of-the-art [13, 28, 29].

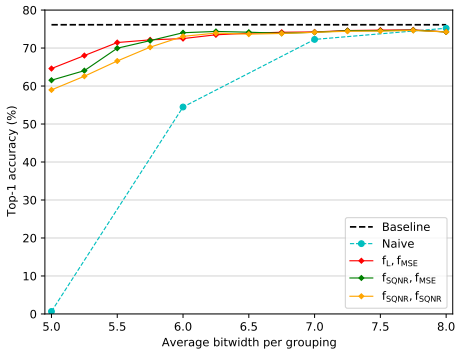
The most reliable configurations were (f_L, f_{MSE}), (f_L, f_{SQNR}), (f_{SQNR}, f_{MSE}), and (f_{SQNR}, f_{SQNR}). For a DCN with an unknown architecture, any of these should yield a high compression ratio and good accuracy. If more information is known about the architecture, then the objective functions can be more cleverly selected. Residual networks are best quantized with (f_L, f_{MSE}) and (f_{SQNR}, f_{MSE}), and models with multiple fully connected (FC) layers are best quantized with (f_{MSE}, f_{SQNR}) and (f_{SQNR}, f_{SQNR}). Note that the residual networks preferred f_{MSE} for the activations, while models with multiple FC layers preferred f_{SQNR}. This suggests that the choice of error metric for the activations may be more important than that of the weights, however more experiments would be required to confirm this.



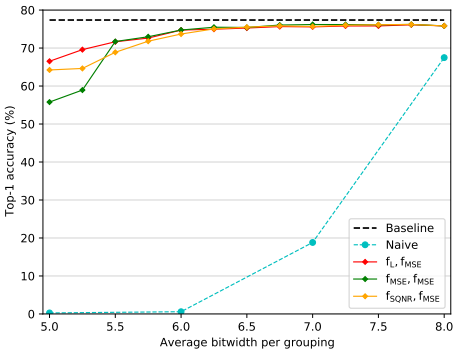
(a) resnet18



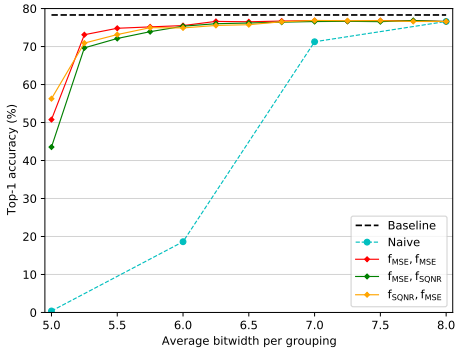
(b) resnet34



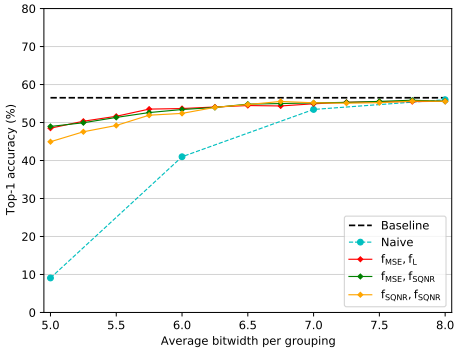
(c) resnet50



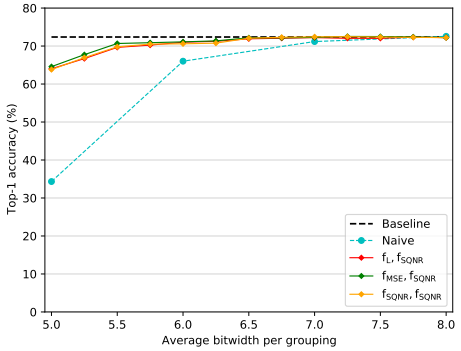
(d) resnet101



(e) resnet152



(f) alexnet



(g) vgg19

Figure 4: These graphs show the top-1 accuracies of the tested models at different average bitwidths using BitAllocation. The cyan circles represent the top-1 accuracy from uniform layer-wise quantization and serve as a benchmark for our experiments. Although all 9 configurations performed well for most models, we only present data for the top 3 configurations for each model to avoid cluttering.

Table 1: This table shows the minimum bitwidths required to stay within 3% of the baseline accuracy and other compression metrics. The bold numbers in brackets represent the accuracy loss from quantization. The reduction in model size and cost of multiplications were computed using the methods described by Goyal et al. [41].

Model	Optimization objectives	Min. avg bitwidth	BitAllocation Top-1 (%)	Baseline Top-1 (%)	Size reduction	Cost of multiplication reduction
ResNet18	(f_L, f_{MSE})	6.25	66.95 **(-2.81)**	69.76	5.112x	25.151x
ResNet34	(f_L, f_{MSE})	6.25	70.75 **(-2.56)**	73.31	5.160x	26.279x
ResNet50	(f_{SQNR}, f_{MSE})	6	74.03 **(-2.10)**	76.13	5.269x	28.346x
ResNet101	(f_{MSE}, f_{MSE})	6	74.74 **(-2.63)**	77.37	5.385x	27.501x
ResNet152	(f_{MSE}, f_{MSE})	6	75.52 **(-2.79)**	78.31	5.459x	27.414x
AlexNet	(f_{MSE}, f_L)	5.75	53.53 **(-2.99)**	56.52	5.797x	28.157x
VGG19	(f_{MSE}, f_{SQNR})	5.5	70.67 **(-1.71)**	72.38	6.414x	29.617x
Average	-	5.96	**(-2.51)**	-	5.514x	27.495x

5.3 Miscellaneous experiments

A major advantage of BitAllocation is its flexibility, We demonstrate this by exploring the following configurations for our pipeline, each applied individually.

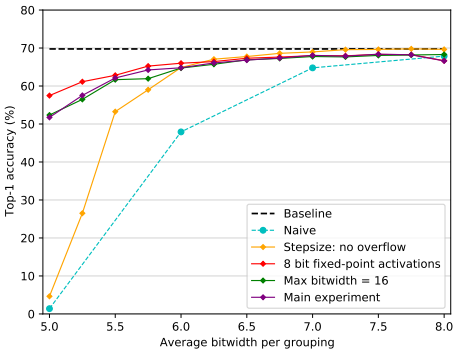
1. An upper bitwidth bound of 16 instead of 8 (i.e. $2 \leq b_i \leq 16$ for all groupings)
2. A "no-overflow" clipping method where the stepsize of each channel is selected to be the smallest power of 2 that prevents overflow
3. 8 bit fixed-point activations

The first configuration can be used for hardware accelerators that support fixed-point arithmetic up to 16 bits; the second configuration explores the effect of different clipping methods; and the third configuration can be used if a dataset for sampling the activations is unavailable, or if reducing a model's memory footprint is more important than reducing its computational complexity. Note that the last configuration is possible since the BitAllocaiton pipeline is applied separately to the weights and activations. The results of these miscellaneous experiments are shown in Figure 5. As expected, quantizing all activations to 8 bits produced the best results, which shows that our pipeline can achieve even higher compression ratios if computational complexity is not the utmost priority.

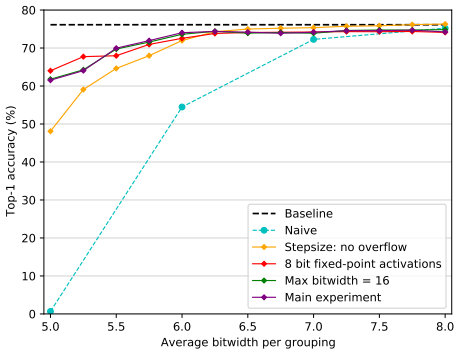
6 Limitations & future work

The greatest limitation of our pipeline is that our BitAllocation algorithm can only solve the optimization problem if k_{budget} exists. Luckily, this is usually the case for practical applications. In all our experiments, k_{budget} existed, and in most cases, it could be found by our Python script in 1-10 seconds using 10-30 iterations of binary search (see Tables 2 and 3). If somehow k_{budget} does not exist, our algorithm still produces an optimal solution, just for a slightly different value of *budget*. For some applications, this approximate solution may be sufficient. Alternatively, the algorithm could be rerun for $budget \pm 1$, as $k_{budget \pm 1}$ will almost certainly exist. For large models with thousands of channels, the difference between a single bitwidth in *budget* is negligible.

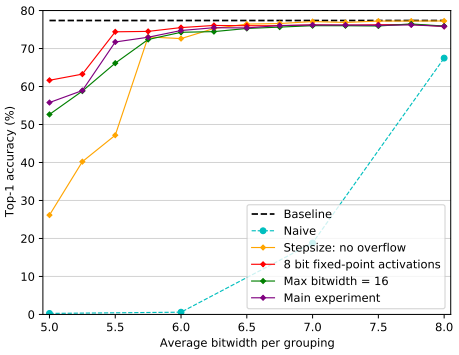
Although this paper presented an application of our algorithm in machine learning quantization, it is certainly not limited to such. At a high level, BitAllocation aims to allocate a finite resource to a set of activities in a way that minimizes or maximizes an objective. Future papers could apply our algorithm to solve similar problems in other fields such as economics, project management, or computer systems. The theory of the algorithm is also an interesting topic. If we can better understand



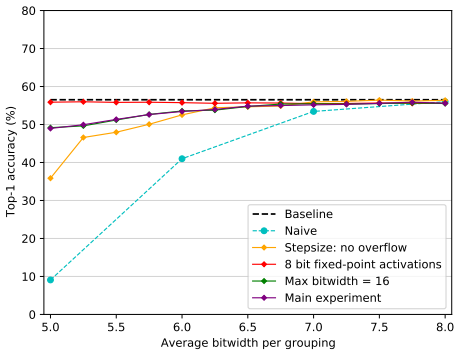
(a) resnet18 (f_{MSE}, f_{MSE})



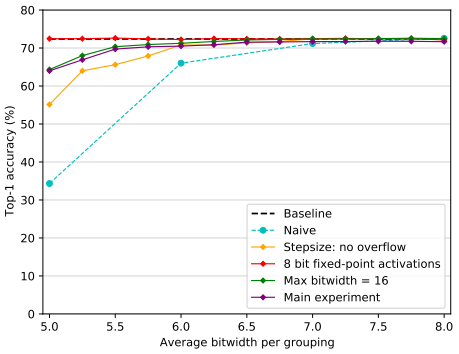
(b) resnet50 (f_{SQNR}, f_{MSE})



(c) resnet101 (f_{MSE}, f_{MSE})



(d) alexnet (f_{MSE}, f_{SQNR})



(e) vgg19 (f_{MSE}, f_{SQNR})

Figure 5: These graphs compare the accuracy of 3 different pipeline configurations on 5 ImageNet models. Data from the main experiments are also provided as reference.

Table 2: Iterations to find k_{budget} for the weights

Average Bitwidth	ResNet18	ResNet50	ResNet101	AlexNet	VGG19
5	21	25	25	27	18
5.5	19	28	21	22	18
6	23	25	32	29	28
6.5	26	32	29	28	27
7	31	29	32	36	30
7.5	29	33	27	33	33
Average	24.67	28.67	27.67	29.17	25.67

Table 3: Wall time in seconds to find k_{budget} for the weights

Average Bitwidth	ResNet18	ResNet50	ResNet101	AlexNet	VGG19
5	0.957	5.234	10.232	2.146	2.297
5.5	0.858	5.759	8.862	1.738	2.352
6	1.035	5.368	13.193	2.319	3.602
6.5	1.173	6.637	11.817	2.272	3.515
7	1.396	6.129	12.979	2.841	3.810
7.5	1.325	6.868	11.210	2.614	4.016
Average	1.124	5.999	11.382	2.322	3.265

when k_{budget} will or will not exist, then we can concretely define its limitations and its use-cases. Lastly, Sections 5.2 and 5.3 showed that different configurations greatly influence the accuracy of the quantized model. Other researchers can build upon our work and discover superior error metrics, clipping methods, and data types for the BitAllocation pipeline.

7 Conclusion

BitAllocation is a fast, flexible, and effective quantization pipeline for DCNs that does not require retraining. Our main contribution was to formulate quantization as a variation of the resource allocation problem, where a *budget* of bits is to be distributed across a DCN to minimize the total quantization error. We developed a novel near linear-time algorithm to solve this problem and use it to identify the optimal bitwidths in a DCN. BitAllocation's flexibility makes it adaptable to different model architectures and hardware implementations, its speed enables quantization at any granularity, and its effectiveness produces high compression ratios. We tested BitAllocation with fixed-point data types on 7 ImageNet models, and found that they could be quantized to 5.5-6.25 bits with a 1-3% decrease in accuracy. On average, this corresponded to a 5.51x reduction in model size and a 27.5x reduction in the cost of multiplications. Although these results are competitive with the state-of-the-art, the full potential of BitAllocation has not been fully realized. Superior pipeline configurations and other applications of the algorithm are exciting directions for future work. As such, BitAllocation is a promising quantization pipeline that is ideal for fast model deployment, hardware accelerators that can exploit variable bitwidth data, and devices with limited memory and power.

Acknowledgements

This work was completed in the summer of 2020 under the supervision of Prof. Andreas Moshovos from the University of Toronto. This work was supported by the University of Toronto's First Year Summer Research Fellowship Award.

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

[3] M. Nikolić, G. B. Hacene, C. Bannon, A. D. Lascorz, M. Courbariaux, Y. Bengio, V. Gripon, and A. Moshovos, "Bitpruning: Learning bitlengths for aggressive and accurate quantization," 2020.

[4] E. Park, D. Kim, and S. Yoo, *Energy-Efficient Neural Network Accelerator Based on Outlier-Aware Low-Precision Computation*. ISCA '18, IEEE Press, 2018.

[5] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018.

[6] M. Nikolic, M. Mahmoud, A. Moshovos, Y. Zhao, and R. D. Mullins, "Characterizing sources of ineffectual computations in deep learning networks," 2019.

[7] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, *Stripes: Bit-Serial Deep Neural Network Computing*. MICRO-49, IEEE Press, 2016.

[8] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, J. K. Kim, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks," 2018.

[9] A. Delmas, S. Sharify, P. Judd, K. Siu, M. Nikolic, and A. Moshovos, "Dpred: Making typical activation and weight values matter in deep learning computing," 2018.

[10] O. Bilaniuk, S. Wagner, Y. Savaria, and J.-P. David, *Bit-Slicing FPGA Accelerator for Quantized Neural Networks*. 2019.

[11] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2016.

[12] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," 2016.

[13] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, "Fixed point quantization of deep convolutional networks," 2016.

[14] C. Wu, M. Wang, X. Li, J. Lu, K. Wang, and L. He, "Phoenix: A low-precision floating-point quantization oriented architecture for convolutional neural networks," 2020.

[15] W. Dally, "High-performance hardware for machine learning," 2015.

[16] S. Anwar, K. Hwang, and W. Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1131–1135, 2015.

[17] S. Shin, Y. Boo, and W. Sung, "Fixed-point optimization of deep neural networks with adaptive step size retraining," 2017.

[18] N. K. Toshihide Ibaraki, *Resource Allocation Problems: Algorithmic Approaches*. Foundations of Computing, The MIT Press, 1988.

[19] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," 2015.

[20] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications," 2016.

[21] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," 2015.

[22] A. Goncharenko, A. Denisov, S. Alyamkin, and E. Terentev, "Trainable thresholds for neural network quantization," *Lecture Notes in Computer Science*, p. 302–312, 2019.

[23] J. Fromm, S. Patel, and M. Philipose, "Heterogeneous bitwidth binarization in convolutional neural networks," 2018.

[24] P. Judd, J. Albericio, T. Hetherington, T. Aamodt, N. E. Jerger, R. Urtasun, and A. Moshovos, "Reduced-precision strategies for bounded memory in deep neural nets," 2016.

[25] K. Hwang and W. Sung, "Fixed-point feedforward deep neural network design using weights +1, 0, and 1," 2014.

[26] R. Banner, Y. Nahshan, E. Hoffer, and D. Soudry, "Post-training 4-bit quantization of convolution networks for rapid-deployment," 2019.

[27] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev, "Low-bit quantization of neural networks for efficient inference," 2019.

[28] J. H. Lee, S. Ha, S. Choi, W.-J. Lee, and S. Lee, "Quantization for rapid deployment of deep neural networks," 2018.

[29] S. Migacz, "8-bit inference with tensorrt," May 2017.

[30] B. O. Koopman, "The optimum distribution of effort," 1953.

[31] W. Karush, "On a class of minimum-cost problems," 1958.

[32] A. Charnes and W. W. Cooper, "The theory of search: Optimum distribution of search effort," *Management Science*, vol. 5, no. 1, pp. 44–50, 1958.

[33] A. Federgruen and P. H. Zipkin, "Solution techniques for some allocation problems," *Mathematical Programming*, 1983.

[34] A. Andersson and F. Ygge, "Efficient resource allocation with non-concave objective functions," 2001.

[35] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "Pact: Parameterized clipping activation for quantized neural networks," 2018.

[36] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[37] R. Zhao, Y. Hu, J. Dotzel, C. D. Sa, and Z. Zhang, "Improving neural network quantization without retraining using outlier channel splitting," 2019.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[39] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," 2017.

[40] Y. Aflalo, A. Noy, M. Lin, I. Friedman, and L. Zelnik, "Knapsack pruning with inner distillation," 2020.

[41] R. Goyal, J. Vanschoren, V. van Acht, and S. Nijssen, "Fixed-point quantization of convolutional neural networks for quantized inference on embedded platforms," *CoRR*, vol. abs/2102.02147, 2021.