RATE-COMPLEXITY OPTIMIZATION IN LOSSLESS NEURAL-BASED IMAGE COMPRESSION

Lucas S. Lopes⋆ *Ricardo L. de Queiroz*⋆ *Philip A. Chou*†

⋆ Universidade de Brasilia, Brasilia, Brazil
† packet.media, Seattle, USA

ABSTRACT

Neural networks are now widely used in image compression. Network architecture and hyperparameter choices impact both compression performance and complexity, but (as we show) there are many examples where higher complexity does not entail better compression. Thus, it is desirable to perform rate-complexity optimization over the space of hyperparameters. In the context of neural-based lossless image compression, we propose an algorithm that traces hyperparameter choices of points on or near the lower convex hull of the cloud of rate-complexity points produced by all combinations of hyperparameters, without having to know in advance the rate-complexity performance of each combination. This reduces the training/evaluation load of the rate-complexity optimization by over 50% in our experiments, for each of three measures of complexity: multiply/add operations per pixel, Joules per pixel, and encoded network size.

***Index Terms*—** Lower convex hull, neural networks, data compression

1. INTRODUCTION

The use of neural networks (NNs) for both lossless and lossy image compression has become common. In order to design a network for such purpose, one must choose an architecture and decide on a potentially large number of hyperparameters, such as the number layers, the numbers of channels or filters in each layer, the dimension and stride of each filter, and so forth. The choice of hyperparameters affects not only the performance of the NN for its intended purpose, but also its complexity. In this paper, we address the problem of optimizing over a combinatorially large set of possible NN hyperparameters to find the desired bitrate-complexity tradeoff in lossless image compression. A subsequent paper will address a similar problem in lossy image compression.

For lossless compression, a NN is typically used as a probability model, often called a context model or entropy model, to drive an arithmetic coder for entropy coding. For example, if y is a bit that needs to be encoded in context $\mathbf{x}$, the NN can approximate the conditional probability $p = P(y = 1|\mathbf{x})$ by network output $q = f_\theta(\mathbf{x})$, where the network parameters $\boldsymbol{\theta}$ are trained on a large sequence of contexts $\{\mathbf{x}_k\}$ and the corresponding sequence of binary symbols $\{y_k\}$ to minimize the cross entropy loss

This work was supported by CNPq under grant 301647/2018-6.

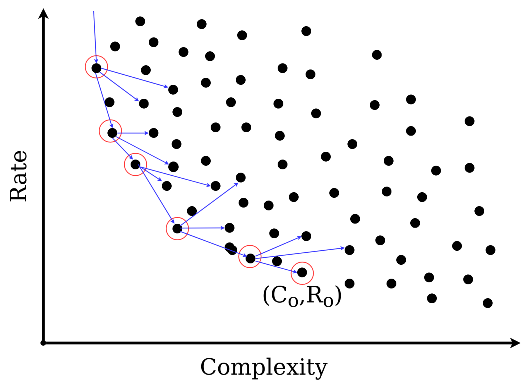

Fig. 1. Illustration of our algorithm, which aims to trace the lower convex hull of a cloud of rate-complexity operating points without knowing all of their positions.

$$R = -\sum_k \left(y_k \log_2(f_{\boldsymbol{\theta}}(\mathbf{x}_k)) + (1 - y_k)\log_2(1 - f_{\boldsymbol{\theta}}(\mathbf{x}_k))\right). \tag{1}$$

This is equivalent to minimizing the number of bits used to encode the training sequence $\{y_k\}$ given $\{\mathbf{x}_k\}$ by an arithmetic coder driven by the NN and is also equivalent to minimizing the KL divergence between conditional distributions p and q [1]. In principle, the bitrate can be as low as the conditional entropy $H(Y|\mathbf{X})$, if the NN is sufficiently accurate.

Figure 1 illustrates the tradeoff between bitrate and complexity. Each point in the Rate-Complexity plane represents the average bitrate and complexity (in some measure, discussed later) of the NN with a particular choice of hyperparameters, trained on a sufficiently large training set. While it seems reasonable to assume that more complex NNs should be able to achieve better performance, as illustrated here (and verified in our experimental results), this is not always the case. In fact, for *most* choices of hyperparameters, there exist other choices of hyperparameters whose networks have *both* lower rate and lower complexity. Only hyperparameters whose networks have rate-complexity performance lying on the *lower convex hull* (LCH) of all such points may be considered optimal. Indeed any point *not* on the LCH is dominated by at least one other achievable point[1] in both rate and complexity. Thus our goal is to find networks whose rate-complexity performance are on or close to the LCH, without having to train and evaluate the rate-complexity performance of networks with every possible hyperparameter choice. Once we find a set of networks on or near the LCH, we can choose among them to find the network with the desired tradeoff between bitrate and complexity, for example the network with the lowest bitrate subject to a complexity constraint.

The primary contribution of this paper is a greedy algorithm for tracing the LCH while evaluating only a small fraction of the total number of possible hyperparameter choices. We are motivated by previous works, such as [2, 3, 4], which trace the exact LCH in tree-structured domains where the performance criteria have certain properties. Since our domain of neural network hyperparameter choices does not readily admit these properties for rate and complexity measures, we do not trace the exact LCH, but rather use a greedy algorithm, which we call the Greedy LCH (GLCH) algorithm, to trace it approximately. However, we provide bounds on the complexity of our algorithm and demonstrate through experimental results that it is able to find networks with performance on or near the LCH while evaluating only a small number of networks.

There is a couple of recent works that tackle the problem of jointly optimizing rate, complexity, and distortion in neural compression [5, 6]. Both aim at controlling complexity through specific network hyperparameters. Works on neural network compression are also closely related, for example [7]–[8]. Finally, energy constrained data compression has previously been studied [9]–[10].

2. THE GLCH ALGORITHM

In this section we present the GLCH algorithm, but phrase it more generally in terms of the NN's *loss* rather than its *rate*. The reason for this is that in a subsequent paper, the same algorithm will be applied to rate-distortion-complexity optimization, for which the loss will be a rate-distortion Lagrangian rather than the rate.

Let $\mathbf{h} = (h_1, \ldots, h_K)$ denote a vector of K hyperparameters, with hyperparameter h_k taking values in $v_k(1), \ldots, v_k(T_k)$, or without loss of generality, in $1, \ldots, T_k$. The number of possible hyperparameter vectors is $N = \prod_{k=1}^{K} T_k$, which grows exponentially in the number of hyperparameters K. Our goal is to find hyperparameter vectors $\mathbf{h}$ that are "good" (e.g., on or near the lower convex hull in the loss-complexity plane) while evaluating (i.e., training) networks for only a small fraction of the total number of possible networks N.

[1]A rate-complexity point $P = (C, R)$ is *achievable* if there exist two choices of hyperparameters having rate-complexity points P_1 and P_2, and $\alpha \in [0, 1]$, such that P is the convex combination $P = \alpha P_1 + (1 - \alpha)P_2$.

Algorithm 1 GLCH Algorithm

```
Input: the graph $\mathcal{G}$ of all possible hyperparameter vectors
1: Set the open and closed sets to the empty set: $\mathcal{O} \leftarrow \emptyset, \mathcal{C} \leftarrow \emptyset$
2: Train/evaluate the minimal node $\mathbf{h}_1$ and add to the open set $\mathcal{O}$
3: repeat
4:     Select one node from the open set: $\mathbf{h} \leftarrow select(\mathcal{O}, \mathcal{C})$
5:     Move the node to the closed set: $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{h}\}, \mathcal{O} \leftarrow \mathcal{O} \backslash \{\mathbf{h}\}$
6:     Train/evaluate all children (i.e., out-neighbors) of the parent
       node $\mathbf{h}$ and add to $\mathcal{O}$, if they are not already in $\mathcal{O}$
7: until $\mathbf{h}$ is the maximal node $\mathbf{h}_N$, or satisfies an early termination
       condition
Output: the set $\mathcal{O} \cup \mathcal{C}$ of visited nodes
```

Towards that end, we let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with vertex set $\mathcal{V}$ of N vertices, or *nodes*, identified with the collection of all possible hyperparameter vectors $\mathbf{h}_1, \ldots, \mathbf{h}_N$, and edge set $\mathcal{E}$ of directed edges, or *links*, from a node $\mathbf{h}$ to a node $\mathbf{h}'$ whenever $\mathbf{h}'$ agrees with $\mathbf{h}$ in every coordinate except one, say the kth coordinate, and in that coordinate, $h'_k = h_k + 1$. That is, we obtain $\mathbf{h}'$ from $\mathbf{h}$ by increasing the kth coordinate by 1. Thus the graph $\mathcal{G}$ corresponds to a K-dimensional rectangular grid of $T_1 \times \cdots \times T_K$ nodes with links to their immediate next neighbors along each axis.

It is easy to see that the graph $\mathcal{G}$ has the following properties: (1) the graph is directed and acyclic; (2) the out-degree of each node is at most K; (3) there is a unique "minimal" node with in-degree 0, namely $\mathbf{h}_1 = (1, 1, \ldots, 1)$, which can be considered the *root* of the graph; (4) there is a unique "maximal" node with out-degree 0, namely $\mathbf{h}_N = (T_1, \ldots, T_K)$; (5) to any node, there exists a path from the root; (6) all paths from the root to a node have the same length, which can be considered the *depth* of the node; (7) the longest path in the graph through the graph starts at the minimal node and ends at the maximal node; (8) the length of the longest path (i.e., the *diameter*) of the graph is $d = \sum_{k=1}^{K} T_k$, which is the depth of the maximal node.

We will visit (i.e., train/evaluate) only a subset of nodes in the graph with the GLCH Algorithm shown in Alg. 1. At every step during execution of the algorithm, the set of nodes $\mathcal{V}$ is partitioned into three sets: a set of open nodes $\mathcal{O}$, a set of closed nodes $\mathcal{C}$, and a set of unvisited nodes. Initially, all nodes are unvisited. Whenever a node is added to the open set, it is trained/evaluated. Some nodes are eventually moved from the open set into the closed set and are not considered further. The union of nodes in the open and closed sets are the set of nodes that have been visited (trained/evaluated).

The *select* function determines variants of the GLCH algorithm. In particular, if the *select* function is *constrained* to choose a node $\mathbf{h}$ from $\mathcal{O}$ only if it is among the deepest open nodes $\mathcal{O}' \subset \mathcal{O}$, whose path from the root is longest, then the GLCH Algorithm will keep extending the longest path and thus will terminate in at most d steps (the diameter

Algorithm 2 Unconstrained *select* Function

```
Input: $\mathcal{O}, \mathcal{C}$
1: Find node $\mathbf{h}^* \in \mathcal{O}$ with least complexity $C_{\mathbf{h}^*}$ s.t. $(C_{\mathbf{h}^*}, L_{\mathbf{h}^*})$ is on LCH of $\{(C_{\mathbf{h}}, L_{\mathbf{h}}) : \mathbf{h} \in \mathcal{O} \cup \mathcal{C}\}$ if exists, else with most complexity on LCH of $\{(C_{\mathbf{h}}, L_{\mathbf{h}}) : \mathbf{h} \in \mathcal{O}\}$
Output: $\mathbf{h}^*$
```

of the graph), having visited at most dK nodes. Since dK grow quadratically in K, whereas N grows exponentially in K, generally $dK \ll N$, so there will be a large computational savings in this constrained case. If the *select* function is not thus constrained, then there may be no upper bound on the computation of the GLCH Algorithm short of N. In either case the algorithm is guaranteed to terminate correctly in at most N steps, and typically in far fewer. We will examine both constrained and unconstrained *select* functions.

The properties of the graph and the bound (if any) on the number of steps and nodes visited by the algorithm, as discussed above, do not depend in any way on the loss-complexity performances of the networks associated with the hyperparameters. However, the *select* function, and possibly the early termination condition, typically depend on these loss-complexity performances.

Let $L_{\mathbf{h}}$ and $C_{\mathbf{h}}$ be the loss and complexity of the neural network associated with the node with hyperparameter vector $\mathbf{h}$. After the GLCH Algorithm returns the set $\mathcal{O} \cup \mathcal{C}$ of trained/evaluated nodes to the user, typically the user computes the lower convex hull of the set $\{(C_{\mathbf{h}}, L_{\mathbf{h}}) : \mathbf{h} \in \mathcal{O} \cup \mathcal{C}\}$ in the loss-complexity plane. From this lower convex hull, the user may select appropriate high-performance networks, such as ones meeting a desired loss or complexity constraint. The hope is that these networks are on or near the lower convex hull of the set $\{(C_{\mathbf{h}}, L_{\mathbf{h}}) : \mathbf{h} \in \mathcal{V}\}$.

Because we are interested in finding networks on or near the LCH in the loss-complexity plane, our unconstrained (resp. constrained) *select* function chooses the node $\mathbf{h}$ among the nodes in the open set $\mathcal{O}$ (resp. subset $\mathcal{O}'$) that 1) is on the LCH of nodes in $\mathcal{O} \cup \mathcal{C}$, and 2) has the least complexity $C_{\mathbf{h}}$ among such nodes on the LCH, as shown in Algs. 2-3. In the event there is no $\mathbf{h}$ in $\mathcal{O}$ (resp. $\mathcal{O}'$) with loss-complexity performance on the specified LCH, then the function falls back to choosing among nodes in $\mathcal{O}$ (resp. $\mathcal{O}'$) with performance on the LCH of $\mathcal{O}$ (resp. $\mathcal{O}'$), which are guaranteed to exist.

The cost of computing the lower convex hull of $\mathcal{O}$ or $\mathcal{O}'$ is trivial in relation to the cost of visiting a node, since visiting a node entails evaluating the loss-complexity performance $(C_{\mathbf{h}}, L_{\mathbf{h}})$ of the node $\mathbf{h}$, which in turn entails training the network with hyperparameter vector $\mathbf{h}$. Thus we ignore the cost of computing the lower convex hull. Note that the algorithm for computing the *lower* convex hull is a simple modification to the planar convex hull algorithms in [11].

Note that since the Constrained *select* Function always chooses to extend the longest path, when used by the GLCH

Algorithm 3 Constrained *select* Function

```
Input: $\mathcal{O}, \mathcal{C}$
1: Let $\mathcal{O}' \subset \mathcal{O}$ be the subset of $\mathcal{O}$ whose distance from the root (i.e., depth) is largest
2: Find node $\mathbf{h}^* \in \mathcal{O}'$ with least complexity $C_{\mathbf{h}^*}$ s.t. $(C_{\mathbf{h}^*}, L_{\mathbf{h}^*})$ is on LCH of $\{(C_{\mathbf{h}}, L_{\mathbf{h}}) : \mathbf{h} \in \mathcal{O} \cup \mathcal{C}\}$ if exists, else with most complexity on LCH of $\{(C_{\mathbf{h}'}, L_{\mathbf{h}'}) : \mathbf{h}' \in \mathcal{O}'\}$
Output: $\mathbf{h}^*$
```

Algorithm, the algorithm will visit a set of nodes corresponding to a maximally unbalanced tree, as illustrated in Fig. 1. In the figure, the leaves of the tree represent nodes in the open set $\mathcal{O}$, while the interior nodes in the tree represent nodes in the closed set $\mathcal{C}$. All other nodes are unvisited. Importantly, the subset of open nodes $\mathcal{O}' \subset \mathcal{O}$ with the longest path to the root all have the same parent.

Our experiments will show that the GLCH Algorithm with the Constrained *select* Function usually visits nodes close to the lower convex hull. On the other hand, the CLGH Algorithm with the Unconstrained *select* Function can sometimes find points even closer to the lower convex hull, with not much additional computation, even though the computation is no longer polynomially bounded.

3. EXPERIMENTAL RESULTS

We demonstrate the GLCH algorithm on the problem of designing neural networks for losslessly compressing binary images of documents pages. Our training set comprises ten 1024×768-pixel binary images scanned from pages of a scientific journal, and our evaluation (test) set likewise comprises ten images from other pages of the same journal.

As the context model for the arithmetic coding, we consider a simple multi-layer perceptron (MLP) with two hidden layers, each with either 10, 20, 40, 80, 160, 320, or 640 hidden units. Thus there are $K = 2$ hyperparameters h_k each taking one of $T_k = 7$ values. The total number of hyperparameter vectors is $N = 49$. For every pixel, the MLP is used to estimate the probability of the next pixel using a context of 32 closest causal pixels (i.e., pixels that have already been encoded). Each network is trained to minimize the binary cross entropy loss (1) using stochastic gradient descent over 100 epochs with a learning rate of 0.0001 and a batch size of 1024 pixels. The models are trained on an NVidia GTX 1080Ti GPU.

As performance measures, bitrate is measured in bits per pixel, while complexity is measured in one of three ways: (1) multiply/add operations per pixel, (2) joules per pixel, and (3) model bits. *Multiply/add operations per pixel* are computed as the number of model parameters, since in an MLP each parameter is used once per pixel. *Joules* are estimated by measuring watts every second using the Nvidia-smi Toolkit® then adding up all values. Since the energy measurements are

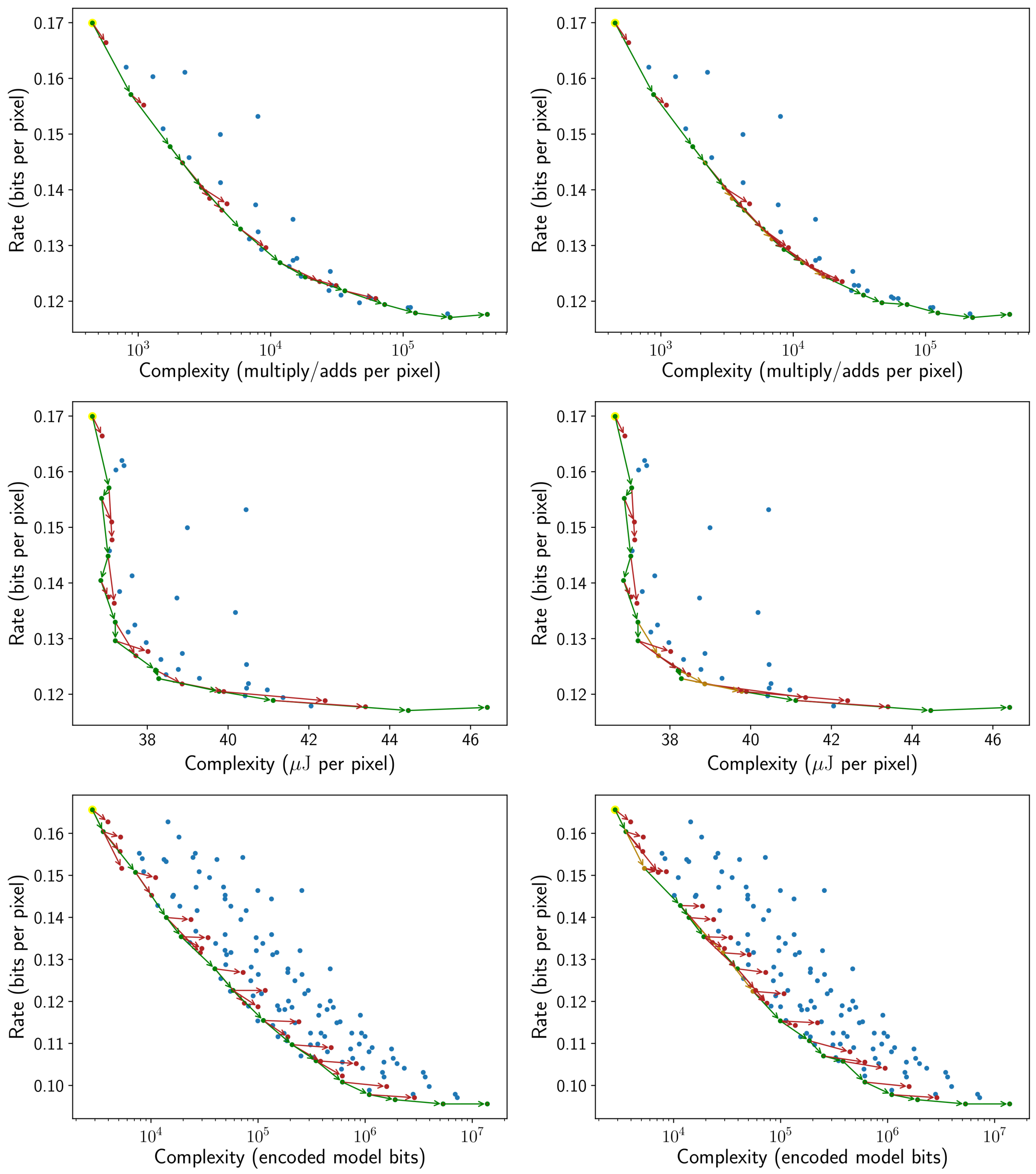

Fig. 2. Rate-Complexity plots. Bitrate is in bits per pixel and complexity is in multiply/add operations per pixel (top row), μJoules per pixel (middle row), and encoded model bits (bottom row). Execution and final state of the GLCH algorithm are shown for the constrained (left column) and unconstrained (right column) selection function. Dots are rate-complexity performance of hyperparameters/nodes. At termination, blue nodes are never visited, red nodes have been visited (moved to the open set $\mathcal{O}$) but are never selected, green and yellow nodes have been visited and selected (moved to the closed set $\mathcal{C}$). Arrows show parent-child relationships from green or yellow parents to red, green, or yellow children, and take the child color. A green arrow and child indicate that the child is selected in the step immediately following its parent's selection, while yellow indicates a gap between the parent's and child's selection.

very noisy, we repeat the measurements over the test set 200 times. Our results are averages over these many repetitions. *Joules per pixel* are obtained by dividing by the total number of pixels in the test set. *Model bits* refers to the number of bits needed to represent the model, e.g., for transmission along with the compressed data in a universal coding scenario [12, 13]. The number of models bits equals the number of network parameters times 8, 16, or 32, depending on the quantization level. Thus, when we used model bits as a measure of complexity, we have a third hyperparameter h_3 taking one of $T_3 = 3$ values, to indicate the quantization level, for a total of $N = T_1T_2T_3 = 147$ possible hyperparameter vectors. We do not retrain the networks for the different quantization levels.

Figure 2 shows a 3×2 grid of rate-complexity plots for all experiments, with the three rows of the grid corresponding to the three complexity measures, and the two columns corresponding to the GLCH algorithm using Constrained and Unconstrained *select* Functions, respectively. In each plot, each dot represents the rate-complexity point for one of the N possible hyperparameter vector choices. Colors of the dots and arrows are explained in the caption of the figure.

It can be observed that for all three complexity measures, the GLCH algorithm with the Constrained *select* Function finds nearly all points on the lower convex hull, while visiting (i.e., training/evaluating) only a fraction of the N possible hyperparameter choices. Indeed, the GLCH algorithm with the Constrained *select* Function requires training/evaluating only 22 networks for the MA/pixel measure, only 24 networks for the energy measure, and only 23 networks for the model bits measure. The GLCH algorithm with the Unconstrained *select* Function finds the few points on the lower convex hull that are missed by the Constrained *select* Function, while having to train/evaluate only a few additional networks.

Though we do not show the results here, similar results hold for other natural measures of complexity, such as runtime, Watts, number of parameters, floating operations per pixel, etc.

4. CONCLUSIONS

We have pointed out that different hyperparameter choices lead to neural-based coders with very different complexities and compression performances. Careless picking of one choice may lead to very sub-optimal coders. Exploring all possible hyperparameter choices may, on the other hand, be very time consuming, as it requires iterative training and evaluation on large datasets for every possible choice. We proposed an efficient algorithm to find the approximate lower convex hull of the cloud of all rate-complexity points of a given neural-based image coder, without exploring all possiblities. We successfully tested our algorithms using three complexity measures, for which our algorithm proved to be quite useful.

While the present paper focused on *lossless* image coding, our algorithm is general enough to apply to other loss functions beside bitrate. Another paper in which we deal with rate-distortion-complexity optimization of *lossy* image coders will be forthcoming. Moreover, even though our work concerns image compression, the GLCH algorithm is applicable to any neural network where there is a complexity-benefit tradeoff.

5. REFERENCES

[1] Lucas S. Lopes, Philip A. Chou, and Ricardo L. de Queiroz, "Adaptive context modeling for arithmetic coding using perceptrons," *IEEE Signal Processing Letters*, vol. 29, pp. 2382–2386, 2022.

[2] Y. Shoham and A. Gersho, "Efficient bit allocation for an arbitrary set of quantizers (speech coding)," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 9, pp. 1445–1453, 1988.

[3] P.A. Chou, T. Lookabaugh, and R.M. Gray, "Optimal pruning with applications to tree-structured source coding and modeling," *IEEE Transactions on Information Theory*, vol. 35, no. 2, pp. 299–315, 1989.

[4] S. W. Wu and A. Gersho, "Rate-constrained picture-adaptive quantization for jpeg baseline coders," in *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1993, vol. 5, pp. 389–392 vol.5.

[5] Y. Gao, R. Feng, Z. Guo, and Z. Chen, "Exploring the rate-distortion-complexity optimization in neural image compression," 2023.

[6] Jinyang Guo, Dong Xu, and Guo Lu, "Cbanet: Toward complexity and bitrate adaptive deep image compression using a single network," *IEEE Transactions on Image Processing*, vol. 32, pp. 2049–2062, 2023.

[7] H. Louati, S. Bechikh, A. Louati, A. Aldaej, and L. B. Said, "Joint design and compression of convolutional neural networks as a bi-level optimization problem," *Neural Computing and Applications*, vol. 34, pp. 15007–15029, 2022.

[8] Jun-Hyuk Kim, Jun-Ho Choi, Jaehyuk Chang, and Jong-Seok Lee, "Efficient deep learning-based lossy image compression via asymmetric autoencoder and pruning," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 2063–2067.

[9] Tiago A. da Fonseca and de Ricardo L. Queiroz, "Energy-constrained real-time h.264/avc video coding," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 1739–1743.

[10] Zhihai He, Yongfang Liang, Lulin Chen, I. Ahmad, and Dapeng Wu, "Power-rate-distortion analysis for wireless video communication under energy constraints," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 5, pp. 645–658, 2005.

[11] Wikipedia contributors, "Convex hull algorithms," 2023, [Online; accessed 29-Aug-2023].

[12] P. A. Chou, M. Effros, and R. M. Gray, "A vector quantization approach to universal noiseless coding and quantization," *IEEE Transactions on Information Theory*, vol. 42, no. 4, pp. 1109–1138, 1996.

[13] K. Sayood, *Introduction to Data Compression, Fifth Edition*, Morgan Kaufmann Publishers Inc., Cambridge, MA, USA, 5th edition, 2017.