Bandwidth-Aware Adaptive Codec for DNN Inference Offloading in IoT

Xiufeng Xie[1], Ning Zhou[2], Wentao Zhu[2], and Ji Liu[1]

[1] Kwai Inc, USA
{xiufengxie,jiliu}@kuaishou.com
[2] Amazon, USA
{ningzhou,wentaozhu}@amazon.com

Abstract. The lightweight nature of IoT devices makes it challenging to run deep neural networks (DNNs) locally for applications like augmented reality. Recent advances in IoT communication like LTE-M have significantly boosted the link bandwidth, enabling IoT devices to stream visual data to edge servers running DNNs for inference. However, uncompressed visual data can still easily overload the IoT link, and the wireless spectrum is shared by numerous IoT devices, causing unstable link bandwidth. Mainstream codecs can reduce the traffic but at the cost of severe inference accuracy drops. Recent works on differentiable JPEG train the codec to tackle the damage to inference accuracy. But they rely on heuristic configurations in the loss function to balance the rate-accuracy tradeoff, providing no guarantee to meet the IoT bandwidth constraint. This paper presents AutoJPEG, a bandwidth-aware adaptive compression solution that learns the JPEG encoding parameters to optimize the DNN inference accuracy under bandwidth constraints. We model the compressed image size as a closed-form function of encoding parameters by analyzing the JPEG codec workflow. Furthermore, we formulate a constrained optimization framework to minimize the original DNN loss while ensuring the image size strictly meets the bandwidth constraint. Our evaluation validates AutoJPEG on various DNN models and datasets. In our experiments, AutoJPEG outperforms the mainstream codecs (like JPEG and WebP) and the state-of-the-art solutions that optimize the image codec for DNN inference.

Keywords: DNN-Friendly Image Compression; IoT; Edge Computing; Inference Offloading; ADMM Optimizer; Bandwidth Constraint

1 Introduction

AI applications are becoming ubiquitous in the Internet-of-Things (IoT) era. For example, wearable items like augmented reality (AR) glasses can employ deep neural networks (DNNs) to understand the environment and show the users an augmented world. However, IoT devices like AR glasses are generally resource-constrained with limited computing power, memory, and battery capacity, thus cannot afford to run a DNN of any size. Recent advances [14], [13], [15] in IoT

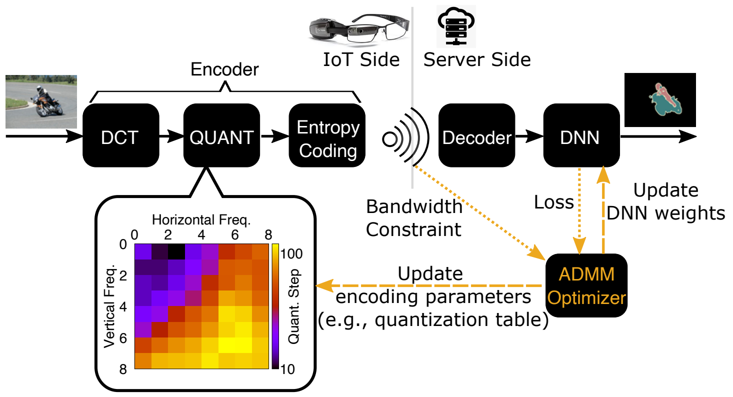


Fig. 1: Given the IoT uplink bandwidth constraint, AutoJPEG uses an alternating direction method of multipliers (ADMM) optimizer to learn the DNN model weights and the encoding parameters jointly.

communication like 5G and Long Term Evolution for Machines (LTE-M) have made it possible to transport the captured data from the IoT device to an edge computing server running DNN inference. Due to the limited wireless bandwidth and huge IoT device population in the future, the IoT device needs to compress large data footprints such as images or videos. Unfortunately, the compression artifacts of mainstream codecs (e.g., JPEG) usually cause severe inference accuracy drop because they target human perception rather than DNNs.

This paper presents AutoJPEG, a bandwidth-aware data compression solution for DNN inference offloading. As outlined in Fig. 1, AutoJPEG jointly learns the DNN weights and encoding parameters like the quantization table of discrete cosine transform (DCT) coefficients in an end-to-end fashion. The core of AutoJPEG is a constrained optimization framework to minimize the DNN loss with the image size as the constraint, enabling it to balance the tradeoff between the inference accuracy and bandwidth cost. The offline optimization and deployment (e.g., wireless broadcast) of encoding parameters to the IoT devices happen only once before the online inference, adding no extra end-to-end latency. The IoT device encodes its captured images using the optimized encoding parameters. It then sends compressed images to the server that runs the optimized DNN for inference. Considering the dynamic IoT bandwidth, we optimize multiple sets of encoding parameters in advance, each for a particular bandwidth constraint. Then the encoder adapts between these sets following the current bandwidth.

We built AutoJPEG on top of the prevalent JPEG codec as a testbed of our optimization framework, which can be easily extended to improve codecs with a similar architecture like MPEG. Our implementation is compatible with JPEG since it only trains the configurable encoding parameters in the JPEG standard. Our evaluation in §4 includes semantic segmentation (large image, complicated DNN) and classification (small image, simple DNN). In our experiments, AutoJPEG outperforms recent DNN-friendly compression works [36], DNN inference offloading works [20], and mainstream codecs like JPEG and WebP, in terms of both inference accuracy and compression ratio.

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

– To the best of our knowledge, AutoJPEG is the first bandwidth-aware data compression solution optimized for inference offloading in IoT networks.
– We dig deep into a representative codec workflow to obtain the closed-form expressions of the compressed image size & recovered image data and use smoothed estimators to make them compatible with gradient descent.
– The crux of this work is a constrained optimization framework designed to learn the encoding parameters under strict bandwidth constraint — a contribution from the methodological aspect as prior works are typically a collection of heuristic designs with no guarantee to meet the constraint.

2 Related Work

Running DNNs on resource-constrained mobile devices like smartphones is challenging. DNN weight pruning [27], [11], [10], [23], [9], [42], [23], [9], [42], [28], [24], [35] and weight quantization [5], [4], [29], [17], [22], [41] can reduce the computation load with little or no impact on the inference accuracy. Another line of work [20], [16] splits the DNN model at a "bottleneck" to form a lightweight head model deployed at the mobile devices and a heavier tail model at the server. However, the above solutions cannot migrate to IoT scenarios because most IoT devices cannot even afford to run a lightweight DNN.

Recent works investigated the DNN-friendly JPEG codec where the compressed images are inputs of DNN inference rather than human vision. DeepN-JPEG [25] assumes the standard deviation of a DCT coefficient determines its contribution to DNN learning, thus using a heuristic function to map the standard deviations of DCT coefficients to a JPEG quantization table. GRACE [36] uses the gradient *w.r.t.* the loss function to measure a DNN's perceptual sensitivity to different DCT frequency bands and then optimizes the JPEG quantization table following the frequency-domain sensitivity. Making JPEG trainable [2], [26], [40], [32] is another related research direction, which incorporates the encoder and DNN in an end-to-end differentiable training framework. A comprehensive work [26] investigates the tradeoff between rate, distortion, and inference accuracy in such a differentiable setup by formulating a loss function as a weighted sum of loss components corresponding to the three targets.

AutoJPEG differs from the above works in the following aspects. 1) most existing works roughly estimate the compressed image size, whereas our work digs into the details of the encoding algorithms (like run-length encoding) and the data formats to estimate a more accurate image size. 2) As shown in Fig. 2, existing works typically use a weighted sum with configurable weight coefficients to combine the original loss and the penalty of image size into one loss function, which fails to precisely control the image size to fit the bandwidth constraint. In contrast, we formulate a constrained optimization problem, guaranteeing the compressed image size meets the given IoT bandwidth constraint.

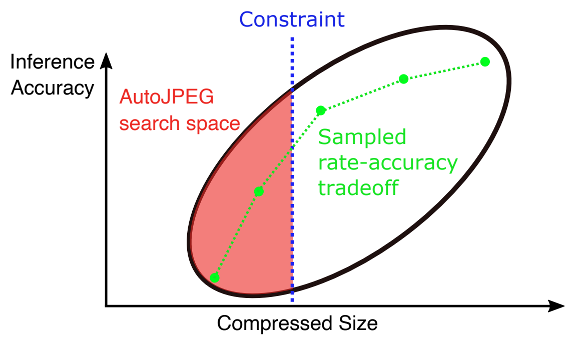


Fig. 2: AutoJPEG searches for the optimal encoding parameters under the bandwidth constraint, while existing works coarsely sample the rate-accuracy relation by tuning the weight coefficients in the weighted-sum loss. (The area in the black oval is the space of encoding parameters allowed by JPEG.)

3 AutoJPEG Design

In this section, we first mathematically model the JPEG codec from the viewpoint of differentiable and non-differentiable operations (§3.1), then discuss how AutoJPEG makes JPEG codec trainable (§3.3), and elaborate on AutoJPEG's ADMM-based optimization (§3.4) framework.

3.1 Mathematical Modeling of JPEG Codec Workflow

(1) **RGB-to-YUV conversion (differentiable).** First, the encoder converts the 3-channel RGB image $\boldsymbol{x} = (\boldsymbol{r}, \boldsymbol{g}, \boldsymbol{b})^\top$ to the YUV color space as $\boldsymbol{x}_\tau = (\boldsymbol{y}, \boldsymbol{u}, \boldsymbol{v})^\top$. The YUV color space provides more room for compression by concentrating salient information to the $\boldsymbol{y}$ channel so that the other two less informative channels $\boldsymbol{u}$ and $\boldsymbol{v}$ allow more compression. The parameter $\boldsymbol{\omega} = (w_r, w_g, w_b)^\top$ uniquely defines the RGB-to-YUV conversion as shown in Eq. (1).

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix} = \begin{bmatrix} w_r & w_g & w_b \\ -\frac{1}{2}\frac{w_r}{1-w_b} & -\frac{1}{2}\frac{w_g}{1-w_b} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2}\frac{w_g}{1-w_r} & -\frac{1}{2}\frac{w_b}{1-w_r} \end{bmatrix} \begin{bmatrix} \boldsymbol{r} \\ \boldsymbol{g} \\ \boldsymbol{b} \end{bmatrix} \tag{1}$$

Eq. (1) is a linear transformation and thus differentiable. It can be written as $\boldsymbol{x}_\tau = \Phi_\omega \boldsymbol{x}$, where Φ_ω is the 3×3 matrix that defines the transform. Conventional encoders choose $\boldsymbol{\omega}$ based on the color sensitivity of the human vision system [6] , e.g., JPEG uses $(w_r, w_g, w_b) = (0.299, 0.587.0.114)$ following [18]. Overall, $\boldsymbol{\omega}$ follows the constraint $\boldsymbol{\omega}^\top \mathbf{1} = 1, \boldsymbol{\omega} \geq 0$. The decoder can recover the RGB image from the YUV image by the inverse transformation $\boldsymbol{x} = \Phi_\omega^{-1} \boldsymbol{x}_\tau$.

(2) **DCT (differentiable).** The encoder then uses the 2D Discrete Cosine Transform (DCT) to convert the YUV image $\boldsymbol{x}_\tau$ to the DCT coefficients $\boldsymbol{X} = \mathrm{DCT}(\boldsymbol{x}_\tau) = (\mathrm{DCT}(\boldsymbol{y}), \mathrm{DCT}(\boldsymbol{u}), \mathrm{DCT}(\boldsymbol{v}))$. DCT is a linear transformation and differentiable. The decoder can recover the image from the DCT coefficients by the Inverse Discrete Cosine Transform (IDCT), which is also differentiable.

***(3)* Quantization (non-differentiable).** Next, the encoder quantizes each channel of the DCT coefficients $\boldsymbol{X}$ by a quantization table. For simplicity, we denote $\boldsymbol{X} = (X_1, \ldots, X_N)$ as all DCT coefficients across 3 channels, and $\boldsymbol{Q} = (q_1, \ldots, q_N)$ contains all elements of the 3 quantization tables, where q_i is the *quantization step* on the i-th DCT coefficient. The quantized result is denoted as $\boldsymbol{X_Q} := \mathcal{R}(\boldsymbol{X}/\boldsymbol{Q})$, where $\mathcal{R}(\cdot)$ denotes the round function $\lfloor\cdot\rceil$ and $\mathcal{R}(\boldsymbol{X}/\boldsymbol{Q}) = (\mathcal{R}(X_1/q_1), \ldots, \mathcal{R}(X_N/q_N))$. Quantization is non-differentiable. Later, the decoder can only recover a lossy version of the original DCT coefficients by dequantization as $\hat{\boldsymbol{X}}_{\boldsymbol{Q}} = \boldsymbol{Q} \cdot \boldsymbol{X_Q}$, and dequantization is differentiable.

***(4)* Lossless encoding (transparent).** The encoder then vectorizes the quantized DCT coefficients $\boldsymbol{X_Q}$ by running a zigzag scan from low to high frequency, which forms groups of consecutive zeros. Run-length encoding (RLE) leverages such consecutive zeros for compression: the encoder only stores the non-zero data points and the count of consecutive 0s after each non-zero data point (*i.e.,* the *skip length*). We elaborate on the image size analysis in §3.2. The JPEG encoder then employs entropy coding to reduce the image size further. Both RLE and entropy coding are lossless, which means the gradient can pass through them in backward propagation.

***(5)* Decoding (differentiable).** The decoder recovers the image from the compressed data by reversing the above encoding steps: entropy decoding (transparent), RLE decoding (transparent), dequantization (differentiable), IDCT (differentiable), and YUV-to-RGB conversion (differentiable). The recovered image $\hat{\boldsymbol{x}}$ is not exactly the same as the original image $\boldsymbol{x}$. Using the recovered image $\hat{\boldsymbol{x}}$ as the DNN's input, the inference accuracy is generally lower than using the original image due to the compression artifacts.

Summary. Let $\mathcal{N}(\boldsymbol{Q}, \boldsymbol{\omega}, \cdot)$ denote the end-to-end JPEG encoding & decoding process, it can be summarized as:

$$\mathcal{N}(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x}) := \mathrm{IDCT}\left(\Phi_{\boldsymbol{\omega}}^{-1}\left(\boldsymbol{Q} \cdot \mathcal{R}\left(\frac{\mathrm{DCT}(\Phi_{\boldsymbol{\omega}} \boldsymbol{x})}{\boldsymbol{Q}}\right)\right)\right) \tag{2}$$

Since RLE and entropy coding are lossless, their encoding and decoding cancel out with each other in Eq. (2). $\mathcal{N}(\boldsymbol{Q}, \boldsymbol{\omega}, \cdot)$ is non-differentiable.

3.2 Modeling Compressed Image Size of JPEG Encoder

Let $\mathcal{H}$ denote the size of image $\boldsymbol{x}$ after JPEG compression, we want to model it as a function $\mathcal{H}(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x})$ of the quantization table $\boldsymbol{Q}$, RGB-to-YUV parameters $\boldsymbol{\omega}$, and the original image $\boldsymbol{x}$. Since the effect of the entropy coding on the image size is always marginal and difficult to model, we use the total number of bits after RLE as an estimation, which is an upper bound of the actual size. If $\mathcal{H}$ is below the image size constraint, the actual size strictly satisfies the constraint. $\mathcal{H}$ consists of two parts: *(i)* the number of bits ($\mathcal{V}$) carrying the value of quantized data points $\boldsymbol{X_Q}$ and *(ii)* the number of bits ($\mathcal{M}$) carrying the supporting data associated with each non-zero data point, including the bit lengths for variable-length binary encoding, the sign bit (DCT can yield negative values), and the skip lengths for RLE.

$$\mathcal{H}\left(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x}\right) = \mathcal{V}\left(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x}\right) + \mathcal{M}\left(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x}\right) \tag{3}$$

***(i)* Bits carrying the datapoint values (non-differentiable).** We look into $\mathcal{V}$ first. Note that a data point with value 0 consumes 0 bit in RLE. Given a quantized DCT data point with value $\mathcal{R}(X_j/q_j)$, Eq. (4) shows the number of bits b_j to represent this data point. We can further remove the round function $\mathcal{R}(\cdot)$ without affecting the value of b_j:

$$b_j = 1 + \left\lfloor \log_2 \left(\mathcal{R}\left(\frac{|X_j|}{q_j} \right) + 0.5 \right) \right\rfloor = 1 + \left\lfloor \log_2 \left(\frac{|X_j|}{q_j} + 0.5 \right) \right\rfloor \tag{4}$$

By summing the sizes of all quantized data points, we have:

$$\mathcal{V} = \sum_{j=1}^{N} b_j = N + \sum_{j=1}^{N} \left\lfloor \log_2 \left(\frac{|X_j|}{q_j} + 0.5 \right) \right\rfloor \tag{5}$$

***(ii)* Bits carrying the supporting data (non-differentiable).** Then we look into $\mathcal{M}$. For each non-zero data point, the JPEG encoder spends 4 bits to represent its binary data length, and it also uses 4 bit to represent the skip length. Since only the non-zero data point needs the sign bit in RLE, we also count it as 1 bit here. Therefore, for each non-zero data point X_j, besides the data size b_j, the encoder needs an extra 9 bits. As a result, the total supporting data size $\mathcal{M}$ is the scaled L0 norm of the quantized data $\mathcal{R}\left(\boldsymbol{X}/\boldsymbol{Q}\right)$:

$$\mathcal{M} = 9 \left\| \mathcal{R}\left(\boldsymbol{X}/\boldsymbol{Q}\right) \right\|_0 \tag{6}$$

3.3 AutoJPEG Makes JPEG Codec End-to-End Trainable

The compression artifacts harming the DNN inference accuracy come from the quantization and are indirectly affected by the RGB-to-YUV conversion. Therefore, we optimize the quantization table $\boldsymbol{Q}$ and the RGB-to-YUV parameters $\boldsymbol{\omega}$ to make the compression artifacts mostly "invisible" to the DNN.

As shown in Eq. (7), AutoJPEG jointly optimizes the encoding parameters ($\boldsymbol{Q}$ and $\boldsymbol{\omega}$) and DNN weights $\mathcal{W}$. The object is to minimize the DNN loss function $\ell(\cdot)$ under a given image size constraint $\mathcal{C} := \mathcal{H}_{\text{raw}} S$, where $S \in (0, 1]$ is the compression ratio, $\mathcal{H}_{\text{raw}}$ is the uncompressed image size, $\boldsymbol{x}^{(i)}$ is the i-th training sample, $\mathcal{X} = \{x^{(1)}, \ldots, x^{(M)}\}$ is the dataset with M samples, $\mathcal{N}(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x}^{(i)})$ from Eq. (2) is the recovered image, $\mathcal{H}(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x}^{(i)})$ from Eq. (3) is the compressed image size in bits.

$$\min_{\mathcal{W}, \boldsymbol{Q}, \boldsymbol{\omega}} \quad \sum_{i=1}^{M} \ell\left(\mathcal{W}, \mathcal{N}(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x}^{(i)})\right) \tag{7a}$$

$$\text{s.t.} \quad \mathcal{H}(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x}^{(i)}) \leq \mathcal{C}, \quad \forall \boldsymbol{x}^{(i)} \in \mathcal{X} \tag{7b}$$

$$\boldsymbol{\omega}^\top \mathbf{1} = 1, \boldsymbol{\omega} \geq 0 \tag{7c}$$

Solving the problem in Eq.(7) is difficult because both $\mathcal{N}(\boldsymbol{Q},\boldsymbol{\omega},\boldsymbol{x}^{(i)})$ and $\mathcal{H}(\boldsymbol{Q},\boldsymbol{\omega},\boldsymbol{x}^{(i)})$ are non-differentiable. In what follows, we discuss how to design the smoothed estimators $\hat{\mathcal{N}}(\boldsymbol{Q},\boldsymbol{\omega},\boldsymbol{x}^{(i)})$ and $\hat{\mathcal{H}}(\boldsymbol{Q},\boldsymbol{\omega},\boldsymbol{x}^{(i)})$ to enable gradient descent.

Smoothed estimator of the image size. The image size $\mathcal{H}(\boldsymbol{Q},\boldsymbol{\omega},\boldsymbol{x}^{(i)})$ is the sum of $\mathcal{V}$ and $\mathcal{M}$. However, both $\mathcal{V}$ and $\mathcal{M}$ are non-differentiable, so we define their smoothed estimators $\hat{\mathcal{V}}$ and $\hat{\mathcal{M}}$ to enable gradient descent.

$$\hat{\mathcal{H}}(\boldsymbol{Q},\boldsymbol{\omega},\boldsymbol{x}^{(i)}) = \hat{\mathcal{V}}(\boldsymbol{Q},\boldsymbol{\omega},\boldsymbol{x}^{(i)}) + \hat{\mathcal{M}}(\boldsymbol{Q},\boldsymbol{\omega},\boldsymbol{x}^{(i)}) \tag{8}$$

The representation of $\mathcal{V}$ in Eq. (5) can be rewritten as:

$$\mathcal{V} = 2N + \sum_{j=1}^{N} \left\lfloor \log_2 \left(\max\Big(\frac{X_j^{(i)}}{q_j} + \frac{1}{2}, \frac{1}{2} \Big) \right) \right\rfloor + \sum_{j=1}^{N} \left\lfloor \log_2 \left(\max\Big(-\frac{X_j^{(i)}}{q_j} + \frac{1}{2}, \frac{1}{2} \Big) \right) \right\rfloor \tag{9}$$

The floor function $\lfloor\cdot\rfloor$ in the above equation is piece-wise constant, which is non-differentiable at integer inputs while having 0 gradient elsewhere, and thus is incompatible[3] with gradient descent. We use the straight through estimator (STE) to solve this problem and define the smoothed estimator of $\mathcal{V}$ as:

$$\hat{\mathcal{V}}(\boldsymbol{Q},\boldsymbol{\omega},\boldsymbol{x}) := \begin{cases} \mathcal{V}(\boldsymbol{Q},\boldsymbol{\omega},\boldsymbol{x}), & \text{if in forward pass} \\ \mathcal{V}^d(\boldsymbol{Q},\boldsymbol{\omega},\boldsymbol{x}), & \text{otherwise} \end{cases}$$

$\hat{\mathcal{V}}$ equals $\mathcal{V}$ in the forward pass, while $\mathcal{V}^d$ is simply Eq. (9) without the floor functions $\lfloor\cdot\rfloor$ so that gradients can pass through in the backward propagation.

$\mathcal{M}$ in Eq. (6) is essentially an L0 regularization for sparsity, and the common practice is to relax it to the L1 norm. We also remove the non-differentiable round function $\mathcal{R}(\cdot)$ following the idea of STE. The smoothed estimator $\hat{\mathcal{M}}$ is:

$$\hat{\mathcal{M}}(\boldsymbol{Q},\boldsymbol{\omega},\boldsymbol{x}) := \begin{cases} \mathcal{M}(\boldsymbol{Q},\boldsymbol{\omega},\boldsymbol{x}), & \text{if in forward pass} \\ 9\left\|(\boldsymbol{X}^{(i)}/\boldsymbol{Q}\right\|_1, & \text{otherwise} \end{cases}$$

Smoothed estimator of the recovered image. According to Eq. (2), the non-differentiable part of $\mathcal{N}(\boldsymbol{Q},\boldsymbol{\omega},\boldsymbol{x}^{(i)})$, the DNN input recovered from the compressed image, is the round function $\mathcal{R}(\cdot)$, but we cannot apply STE to remove $\mathcal{R}(x)$. For the quantization and recovery process, the recovered value $Y_j = \mathcal{R}(X_j/q_j)\cdot q_j$ with quantization step q_j, if we simply ignore the round function, then we have $Y_j = (X_j/q_j)q_j = X_j$, where q_j is cancelled out. Since this work is about learning the optimal q_j, we cannot let q_j disappear.

[3] Although the max function is also non-differentiable, gradients can still pass through the max function during backward propagation (just like ReLU).

$\mathcal{R}(\cdot)$ can be written as a piece-wise constant function $\mathcal{R}(x) = \mathcal{A}(x - 0.5 - K) + K$, if $K \leq x < K + 1$. Different piece of the function has different constant integer K whose value equals $\lfloor x \rfloor$. $\mathcal{A}(\cdot)$ is an unit step function which is non-differentiable at 0 and has 0 gradient elsewhere.

$$\mathcal{A}(x) := \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \quad \hat{\mathcal{A}}(x) := \begin{cases} \mathcal{A}(x), & \text{if in forward pass} \\ \frac{1}{2}\left(\tanh(Tx) + 1\right), & \text{otherwise} \end{cases}$$

We use a smoothed estimator $\hat{\mathcal{A}}(x)$ of the step function $\mathcal{A}(x)$ to enable gradient descent, where T controls the steepness of the smoothed step function. This work uses $T = 5$ to balance the approximation accuracy and training stability as discussed in [38]. By replacing $\mathcal{A}(\cdot)$ in function $\mathcal{R}(\cdot)$ with $\hat{\mathcal{A}}(\cdot)$, we have $\hat{\mathcal{R}}(\cdot)$, a smoothed estimator of $\mathcal{R}(\cdot)$. Then the DNN input recovered from the compressed image can be estimated as $\hat{\mathcal{N}}(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x})$ in Eq. (10), which allows gradients to pass through during backward propagation.

$$\hat{\mathcal{N}}\left(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x}\right) = \mathrm{IDCT}\left(\Phi_{\boldsymbol{\omega}}^{-1}\left(\boldsymbol{Q} \cdot \hat{\mathcal{R}}\left(\frac{\mathrm{DCT}\left(\Phi_{\boldsymbol{\omega}}\boldsymbol{x}\right)}{\boldsymbol{Q}}\right)\right)\right) \tag{10}$$

By replacing the non-differentiable operators in Eq. (7) with the smoothed estimators in Eq. (8) and (10), we have a new optimization problem:

$$\min_{\mathcal{W}, \boldsymbol{Q}, \boldsymbol{\omega}} \quad \sum_{i=1}^{M} \ell\left(\mathcal{W}, \hat{\mathcal{N}}(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x}^{(i)})\right) \tag{11a}$$

$$\text{s.t.} \quad \hat{\mathcal{H}}(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x}^{(i)}) \leq \mathcal{C}, \ \forall \boldsymbol{x}^{(i)} \in \mathcal{X} \tag{11b}$$

$$\boldsymbol{\omega}^{\top}\mathbf{1} = 1, \boldsymbol{\omega} \geq 0 \tag{11c}$$

3.4 Solving the Optimization Problem Using ADMM

In what follows, we discuss how AutoJPEG solves the constrained optimization problem in Eq. (11). First, we use the augmented Lagrangian method to convert the problem to its equivalent minimax problem:

$$\min_{\mathcal{W}, \boldsymbol{Q}, \boldsymbol{\omega}} \max_{z \geq 0} \quad \sum_{i=1}^{M} \left(\ell(\mathcal{W}, \hat{\mathcal{N}}(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x}^{(i)})) + \zeta(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x}^{(i)}, z)\right) \tag{12a}$$

$$\text{s.t.} \quad \boldsymbol{\omega}^{\top}\mathbf{1} = 1, \boldsymbol{\omega} \geq 0 \tag{12b}$$

where $\zeta(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x}^{(i)}, z)$ is the augmented Lagrangian:

$$\zeta(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x}^{(i)}, z) := \frac{\rho_z}{2}\left[\hat{\mathcal{H}}(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x}^{(i)}) - \mathcal{C}\right]_{+}^{2} + z\left(\hat{\mathcal{H}}(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x}^{(i)}) - \mathcal{C}\right)$$

We use $[\cdot]_{+}$ to denote the non-negative clamp $\max(\cdot, 0)$. ρ_z is the learning rate of the dual variable z when using ADMM to solve the problem.

Algorithm 1: Codec & DNN Joint Optimization Under Size Constraint

```
Input  : Image size constraint $\mathcal{C}$,
         Pretrained DNN weights $\mathcal{W}^I$,
         BT.601 RGB-to-YUV parameter $\boldsymbol{\omega}^I$
Output : Quantization table $\boldsymbol{Q}^*$,
         RGB-to-YUV parameters $\boldsymbol{\omega}^*$,
         DNN weights $\mathcal{W}^*$
Initialize $\mathcal{W} = \mathcal{W}^I, \boldsymbol{Q} = J_{3,8,8}, \boldsymbol{\omega} = \boldsymbol{\omega}^I$
while $\boldsymbol{x}^{(t)} \in \mathcal{X}$ and $\mathcal{H}\left(\boldsymbol{Q}, \boldsymbol{\omega}, \boldsymbol{x}^{(t)}\right) > \mathcal{H}_c$ do
    Update the primary variable $\mathcal{W}$ by SGD following Eq.(13);
    Update the primary variable $\boldsymbol{Q}$ by SGD following Eq.(14);
    Update the primary variable $\boldsymbol{\omega}$ by PGD following Eq.(15);
    Update the dual variable $z$ following Eq.(16)
end
$\boldsymbol{Q}^* = \boldsymbol{Q}$, $\boldsymbol{\omega}^* = \boldsymbol{\omega}$, $\mathcal{W}^* = \mathcal{W}$
```

Following Alg. 1, in the t-th iteration, we first optimize the DNN weights $\mathcal{W}$ by stochastic gradient descent (SGD) with learning rate $\rho_{\mathcal{W}}$:

$$\mathcal{W}^{(t+1)} = \mathcal{W}^{(t)} - \rho_{\mathcal{W}} \nabla_{\mathcal{W}^{(t)}} \mathcal{L}^{(t)} \tag{13}$$

$\mathcal{L}^{(t)}$ is the loss $\sum_{i=1}^{M} (\ell(\mathcal{W}^{(t)}, \hat{\mathcal{N}}(\boldsymbol{Q}^{(t)}, \boldsymbol{\omega}^{(t)}, \boldsymbol{x}^{(t)})) + \zeta(\boldsymbol{Q}^{(t)}, \boldsymbol{\omega}^{(t)}, \boldsymbol{x}^{(t)}, z^{(t)}))$ from Eq. (12a) in the t-th iteration. We then use SGD to optimize the quantization table $\boldsymbol{Q}$ with learning rate ρ_q:

$$\boldsymbol{Q}^{(t+1)} = \boldsymbol{Q}^{(t)} - \rho_q \nabla_{\boldsymbol{Q}^{(t)}} \mathcal{L}^{(t)} \tag{14}$$

The constraint $\boldsymbol{\omega}^\top \mathbf{1} = 1, \boldsymbol{\omega} \geq 0$ is an unit 3-simplex, hence the variable $\boldsymbol{\omega}$ can be optimized by projected gradient descent (PGD) with learning rate $\rho_{\boldsymbol{\omega}}$:

$$\boldsymbol{\omega}^{(t+1)} = \mathcal{P}(\boldsymbol{\omega}^{(t)} - \rho_\omega \nabla_{\boldsymbol{\omega}^{(t)}} \mathcal{L}^{(t)}) \tag{15}$$

where $\mathcal{P}(\cdot)$ is the projection to an unit 3-simplex.

Finally, the optimizer performs the dual update by updating the dual variable z using project gradient ascent with learning rate ρ_z, where the non-negative clamp $[\cdot]_+$ projects the gradient ascent result to the space where $z \geq 0$:

$$z^{(t+1)} = \left[z^{(t)} + \rho_z \left(\hat{\mathcal{H}}(\boldsymbol{Q}^{(t)}, \boldsymbol{\omega}^{(t)}, \boldsymbol{x}^{(t)}) - \mathcal{C}\right)\right]_+ \tag{16}$$

The initial quantization table $\boldsymbol{Q}$ is a $3 \times 8 \times 8$ all-one matrix $J_{3,8,8}$ that barely compresses the image. Meanwhile, the initial RGB-to-YUV parameters $\boldsymbol{\omega}^I = (0.299, 0.587, 0.114)$ comes from BT.601 standard, and the initial DNN weights $\mathcal{W}^I$ comes from a model pre-trained by uncompressed training dataset.

4 Evaluation

4.1 Experiment Setup

Benchmarks. We compare AutoJPEG with a recent work GRACE [36] that optimizes quantization tables following the DNN's perceptual sensitivity. Neurosurgeon [20] is another benchmark, which partitions the DNN into two parts, one running on the client and the other on the server. Our benchmarks also include the mainstream codecs like JPEG [34], PNG [31], WebP [7], and H.264 [30].

Datasets. Since semantic segmentation typically requires high-quality images that may overload the IoT link, it is an ideal application of AutoJPEG. Our evaluation in §4.2 tests semantic segmentation on the Cityscapes dataset [3] with 2048×1024 resolution and lossless PNG format. We also evaluate AutoJPEG in classification tasks on CIFAR10 [21] dataset with small 32×32 images in §4.3. Such scenarios can be found in low-cost IoT devices.

DNN models. AutoJPEG aims to achieve less inference accuracy loss than other encoders. Therefore, we test commonly used models instead of the latest models with state-of-the-art accuracy. For image classification, we use the ResNet models including ResNet20, ResNet32, ResNet56, ResNet110 [8] and VGG models including VGG11 and VGG13 [33]. For semantic segmentation, we use the dialated ResNet models [39] DRN-D-38 and DRN-D-22, following the evaluation setup of GRACE, the vital benchmark algorithm.

Hardware & training details. We run the experiments on a machine with a 2.1GHz Intel Xeon Gold 5218R CPU, 120GB memory, and 4 Nvidia Geforce RTX 2080Ti GPUs. The ADMM-based optimization framework and DNNs are based on PyTorch. The joint optimization finishes when the compressed image size satisfies the given constraint so there is no preset total epoch number. The initial model weights are pretrained on uncompressed datasets so we set the learning rate of the model weights to a small 10^{-4} for finetuning.

4.2 AutoJPEG in Semantic Segmentation

Fig. 3 shows AutoJPEG's superior performance to existing solutions when compressing the input image for semantic segmentation model. The target DNN model is DRN-D-38, and the dataset is Cityscapes. We observe that AutoJPEG achieves the best balance between the inference accuracy and compression ratio.

Comparison with Neurosurgeon. Neurosurgeon has poor performance because its effectiveness relies on the DNN structure. It only works for DNNs with "bottleneck" layers whose output tensor is small. However, for DNNs to perform complicated tasks, the output tensor of every hidden layer can be bulky. Here the output data size of any layer in DRN-D-38 is larger than the input image, so the Neurosurgeon fails to reduce the data size. We further improve it as Neurosurgeon-Prune by using the DNN pruning solution from [37] to carve a "bottleneck" layer if the DNN does not have any, and thus the output tensor size of the partitioning point is reduced. Fig. 3 shows that Neurosurgeon-Prune's

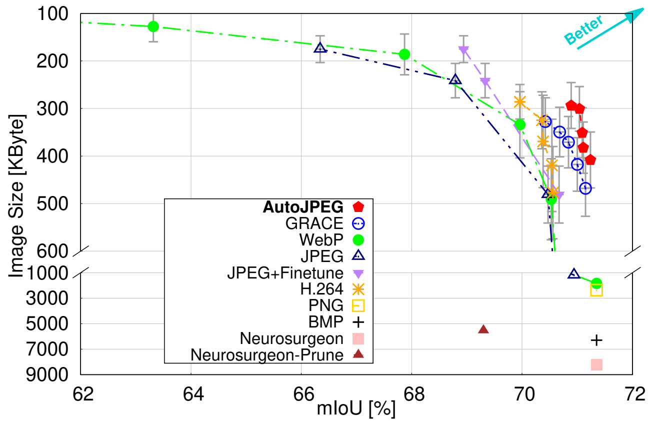
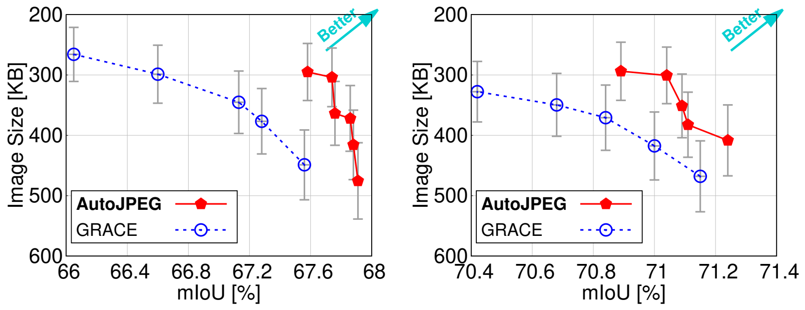


Fig. 3: AutoJPEG achieves a better balance between the DNN inference accuracy (mIoU) and compressed image size than existing solutions.

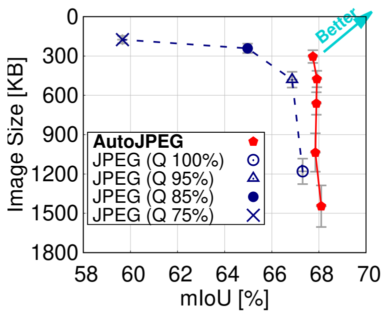


(a) Model: DRN-D-22 (b) Model: DRN-D-38

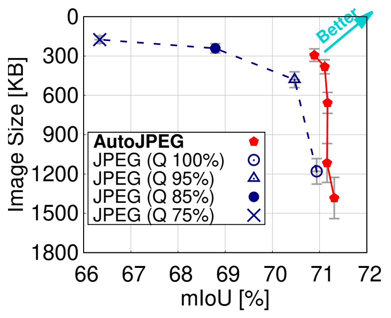
Fig. 4: Our solution AutoJPEG vs. GRACE.

performance is still far behind AutoJPEG. It only slightly compresses the data size but lowers the mIoU by more than 2%, because the channel-wise pruning is too coarse-grained compared to the spectral quantization in AutoJPEG.

Comparison with GRACE. We then perform a detailed comparison between AutoJPEG and GRACE, the state-of-the-art solution on DNN-friendly image compression. We run the experiments under multiple compression levels for both algorithms to profile their tradeoff between the image size and inference accuracy. As shown in Fig. 4a and 4b, AutoJPEG outperforms GRACE on both models by achieving higher mIoU at smaller image size. For example, from Fig. 4b, we observe that AutoJPEG achieves a 71.24% mIoU at image size of 408KB, while GRACE has a 0.24% lower mIoU at a larger image size of 418KB.

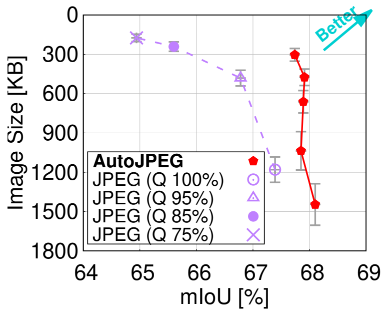


(a) Model: DRN-D-22

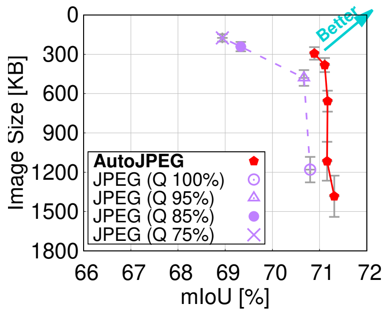


(b) Model: DRN-D-38

Fig. 5: Our solution AutoJPEG vs. JPEG.



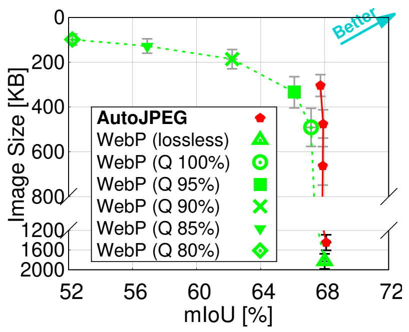
(a) Model: DRN-D-22



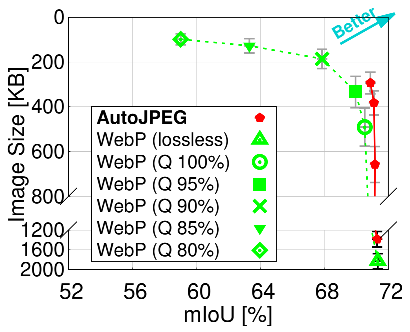
(b) Model: DRN-D-38

Fig. 6: Our solution AutoJPEG vs. JPEG+FT.

Comparison with JPEG and WebP. In this experiment, we first compare AutoJPEG with the JPEG encoder. We use multiple JPEG quality levels (75%, 85%, 95%, and 100%) since image size depends on the quality level ranging from 1% to 100% [34]. As shown in Fig. 5, AutoJPEG achieves higher mIoU than JPEG with similar image size. When the image size is small, JPEG suffers from a significant accuracy drop, while AutoJPEG's accuracy only reduces slightly. For instance, in Fig. 5a, the 85% JPEG quality causes a 7.43% mIoU loss compared to the uncompressed image. AutoJPEG with a similar size achieves a high mIoU with only 0.26% accuracy loss. When the image size is large, AutoJPEG still outperforms JPEG. For instance, the 100% JPEG quality yields a large image size of 1180KB but causes a 0.69% mIoU loss, while AutoJPEG has a slight 0.15% mIoU loss at a smaller image size of 1036KB. Finetuning the DNN by the JPEG-encoded image (JPEG+FT) can improve the inference accuracy, but the accuracy remains far below AutoJPEG as shown in Fig. 6. We further compare AutoJPEG with the WebP encoder of quality 80%, 85%, 90%, 100%, and lossless. As shown in Fig. 7, AutoJPEG also outperforms WebP in our experiments.



(a) Model: DRN-D-22



(b) Model: DRN-D-38

Fig. 7: Our solution AutoJPEG vs. WebP.

DNN Tuning	**YUV Tuning**	**mIoU (%)**	**Avg. Size (KB)**
✓	✓	71.24	408
✓	×	71.17	419
×	✓	70.94	404
×	×	70.75	402

Table 1: Ablation study for components of AutoJPEG's joint optimization.

Ablation study. We validate AutoJPEG's components separately in this experiment. The results in Table 1 show that enabling both DNN tuning and YUV tuning yields the highest mIoU. Without YUV tuning (using JPEG's default YUV color space), the mIoU drops by 0.07% even when the image size is slightly larger. We also evaluate AutoJPEG without DNN tuning (use the original pre-trained DNN for inference), as some users may not want to update the DNNs already deployed on servers. In this case, the mIoU is slightly lower than when the DNN tuning is on but still outperforms using JPEG with 95% quality.

4.3 AutoJPEG in Image Classification

We further evaluate AutoJPEG in image classification tasks on CIFAR10. The experiment setup follows §4.1, and the benchmark is the widely used 75% quality JPEG. Fig. 8 shows that AutoJPEG outperforms JPEG (no matter with or without finetuning the DNN by the JPEG training set) on all tested DNN models, with significantly higher Top-1 classification accuracy and smaller image size. For instance, AutoJPEG reduces the image size by 13% while improving the Top-1 classification accuracy by 4.73% on ResNet56 even when the JPEG benchmark uses DNN finetuned by JPEG-encoded dataset for inference.

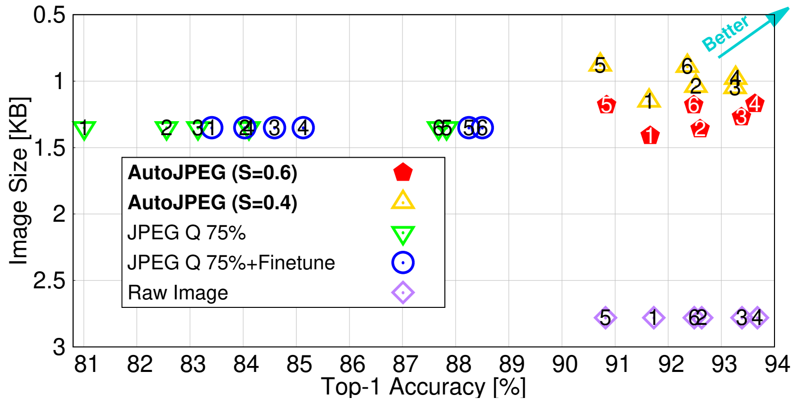


Fig. 8: Comparison of classification accuracy (Top-1) and compressed image size targeting different models, label 1, 2, 3, 4, 5, 6 correspond to ResNet20, ResNet32, ResNet56, ResNet110, VGG11, VGG13, respectively.

5 Limitations and Future Works

Our constrained optimization algorithm uses STE [1] to approximate the non-differentiable functions. A more rigorous approach to handle such non-differentiable functions is proximal gradient descent [12], which guarantees the same convergence rate as gradient descent. However, finding the proximity operator is challenging, and we leave it to future work. Meanwhile, our end-to-end framework enables joint optimization of the DNN model weights quantization (like QAT [19]) and DNN input image quantization, which is a future direction we plan to explore. Finally, inference offloading faces a general challenge that is out of the scope of this work: communication latency. A low-latency link is required to exploit inference offloading in delay-sensitive applications like AR.

6 Conclusion

This paper presents AutoJPEG, a DNN-friendly image compression solution tailored for DNN inference offloading in IoT networks. By harnessing ADMM to jointly optimize the encoding parameters and DNN model weights under a given image size constraint, AutoJPEG achieves significant IoT link bandwidth saving while preserving high inference accuracy. It has low complexity and is compatible with the JPEG codec, making deployment easy. In our evaluation of semantic segmentation and image classification tasks, AutoJPEG demonstrates superior performance to recent DNN-friendly data compression algorithms, DNN splitting algorithms, and mainstream codecs.

References

1. Bengio, Y., Léonard, N., Courville, A.: Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. arXiv preprint arXiv:1308.3432 (2013)
2. Choi, J., Han, B.: Task-Aware Quantization Network for JPEG Image Compression. In: European Conference on Computer Vision. pp. 309–324. Springer (2020)
3. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The Cityscapes Dataset for Semantic Urban Scene Understanding. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3213–3223 (2016), `https://www.cityscapes-dataset.com/`
4. Courbariaux, M., Bengio, Y., David, J.P.: BinaryConnect: Training Deep Neural Networks with Binary Weights during Propagations. In: Advances in Neural Information Processing Systems (2015)
5. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to + 1 or -1. arXiv preprint arXiv:1602.02830 (2016)
6. Fairman, H., Brill, M., Hemmendinger, H.: How the CIE 1931 Color-matching Functions were Derived from Wright-Guild Data. Color Research and Application **22**(1), 11–23 (1997)
7. Google: An Image Format for the Web (2021), `https://developers.google.com/speed/webp/`, [Online]
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 770–778 (2016)
9. He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y.: Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18. pp. 2234–2240. International Joint Conferences on Artificial Intelligence Organization (7 2018). https://doi.org/10.24963/ijcai.2018/309, `https://doi.org/10.24963/ijcai.2018/309`
10. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S.: AMC: AutoML for Model Compression and Acceleration on Mobile Devices. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 784–800 (2018)
11. He, Y., Zhang, X., Sun, J.: Channel Pruning for Accelerating Very Deep Neural Networks. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1389–1397 (2017)
12. Hiriart-Urruty, J.B., Lemaréchal, C.: Convex Analysis and Minimization Algorithms I: Fundamentals, vol. 305. Springer Science & Business Media (2013)
13. Hoglund, A., Bergman, J., Lin, X., Liberg, O., Ratilainen, A., Razaghi, H.S., Tirronen, T., Yavuz, E.A.: Overview of 3GPP Release 14 Further Enhanced MTC. IEEE Communications Standards Magazine **2**(2), 84–89 (2018)
14. Hoglund, A., Lin, X., Liberg, O., Behravan, A., Yavuz, E.A., Van Der Zee, M., Sui, Y., Tirronen, T., Ratilainen, A., Eriksson, D.: Overview of 3GPP Release 14 Enhanced NB-IoT. IEEE Network **31**(6), 16–22 (2017)
15. Hoymann, C., Astely, D., Stattin, M., Wikstrom, G., Cheng, J.F., Hoglund, A., Frenne, M., Blasco, R., Huschke, J., Gunnarsson, F.: LTE Release 14 Outlook. IEEE Communications Magazine **54**(6), 44–49 (2016)

16. Hu, C., Bao, W., Wang, D., Liu, F.: Dynamic Adaptive DNN Surgery for Inference Acceleration On the Edge. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications. pp. 1423–1431. IEEE (2019)
17. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized Neural Networks. In: Advances in Neural Information Processing Systems (2016)
18. ITUR: BT 601: Studio Encoding Parameters of Digital Television for Standard 4: 3 and Wide-screen 16: 9 Aspect Ratios. ITU-R Rec. BT **656** (1995)
19. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D.: Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2704–2713 (2018)
20. Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., Tang, L.: Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. ACM SIGARCH Computer Architecture News **45**(1), 615–629 (2017)
21. Krizhevsky, A., et al.: Learning Multiple Layers of Features from Tiny Images. Tech. rep., University of Toronto (2009), `https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz`
22. Li, F., Zhang, B., Liu, B.: Ternary Weight Networks. arXiv preprint arXiv:1605.04711 (2016)
23. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, P.H.: Pruning Filters for Efficient ConvNets. International Conference on Learning Representations (2017)
24. Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y., Shao, L.: HRank: Filter Pruning using High-Rank Feature Map. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1529–1538 (2020)
25. Liu, Z., Liu, T., Wen, W., Jiang, L., Xu, J., Wang, Y., Quan, G.: DeepN-JPEG: A Deep Neural Network Favorable JPEG-based Image Compression Framework. In: Proceedings of the 55th Annual Design Automation Conference. pp. 1–6 (2018)
26. Luo, X., Talebi, H., Yang, F., Elad, M., Milanfar, P.: The Rate-Distortion-Accuracy Tradeoff: JPEG Case Study. arXiv preprint arXiv:2008.00605 (2020)
27. Ma, X., Guo, F.M., Niu, W., Lin, X., Tang, J., Ma, K., Ren, B., Wang, Y.: PCONV: The Missing but Desirable Sparsity in DNN Weight Pruning for Real-Time Execution on Mobile Devices. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 5117–5124 (2020)
28. Peng, H., Wu, J., Chen, S., Huang, J.: Collaborative Channel Pruning for Deep Networks. In: International Conference on Machine Learning. pp. 5113–5122 (2019)
29. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In: European Conference on Computer Vision (2016)
30. Richardson, I.E.: H. 264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia. John Wiley & Sons (2004)
31. Roelofs, G., Koman, R.: PNG: the Definitive Guide. O'Reilly & Associates, Inc. (1999)
32. Shin, R., Song, D.: JPEG-resistant Adversarial Images. In: NIPS 2017 Workshop on Machine Learning and Computer Security. vol. 1 (2017)
33. Simonyan, K., Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556 (2014)
34. Wallace, G.K.: The JPEG Still Picture Compression Standard. IEEE Transactions on Consumer Electronics (1992)
35. Wang, Y., Zhang, X., Xie, L., Zhou, J., Su, H., Zhang, B., Hu, X.: Pruning from Scratch. In: AAAI Conference on Artificial Intelligence (2020)

36. Xie, X., Kim, K.H.: Source Compression with Bounded DNN Perception Loss for IoT Edge Computer Vision. In: The 25th Annual International Conference on Mobile Computing and Networking. pp. 1–16 (2019)
37. Yang, H., Zhu, Y., Liu, J.: ECC: Platform-Independent Energy-Constrained Deep Neural Network Compression via a Bilinear Regression Model. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 11206–11215 (2019)
38. Yang, J., Shen, X., Xing, J., Tian, X., Li, H., Deng, B., Huang, J., Hua, X.s.: Quantization Networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7308–7316 (2019)
39. Yu, F., Koltun, V., Funkhouser, T.: Dilated Residual Networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 472–480 (2017)
40. Zhang, C., Karjauv, A., Benz, P., Kweon, I.S.: Towards Robust Data Hiding Against (JPEG) Compression: A Pseudo-Differentiable Deep Learning Approach. arXiv preprint arXiv:2101.00973 (2020)
41. Zhu, C., Han, S., Mao, H., Dally, W.J.: Trained Ternary Quantization. arXiv preprint arXiv:1612.01064 (2016)
42. Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., Huang, J., Zhu, J.: Discrimination-Aware Channel Pruning for Deep Neural Networks. In: Advances in Neural Information Processing Systems. pp. 875–886 (2018)