Optimizing 3D U-Net-based Brain Tumor Segmentation with Integer-arithmetic Deep Learning Accelerators

WEIJIA WANG and BILL LIN, University of California, San Diego

While gliomas have become the most common cancerous brain tumors, manual diagnoses from 3D MRIs are time-consuming and possibly inconsistent when conducted by different radiotherapists, which leads to the pressing demand for automatic segmentation of brain tumors. State-of-the-art approaches employ FCNs to automatically segment the MRI scans. In particular, 3D U-Net has achieved notable performance and motivated a series of subsequent works. However, their significant size and heavy computation have impeded their actual deployment. Although there exists a body of literature on the compression of CNNs using low-precision representations, they either focus on storage reduction without computational improvement or cause severe performance degradation. In this article, we propose a CNN training algorithm that approximates weights and activations using *non-negative integers* along with *trained affine mapping* functions. Moreover, our approach allows the dot-product operations to be performed in an *integer-arithmetic* manner and defers the floating-point decoding and encoding phases until the end of layers. Experimental results on BraTS 2018 show that our trained affine mapping approach achieves near full-precision dice accuracy with 8-bit weights and activations. In addition, we achieve a dice accuracy within 0.005 and 0.01 of the full-precision counterparts when using 4-bit and 2-bit precisions, respectively.

CCS Concepts: • **Computing methodologies → Neural networks;**

Additional Key Words and Phrases: Semantic segmentation, volumetric segmentation, fully convolutional network, quantization, training algorithm

ACM Reference format:
Weijia Wang and Bill Lin. 2022. Optimizing 3D U-Net-based Brain Tumor Segmentation with Integer-arithmetic Deep Learning Accelerators. *J. Emerg. Technol. Comput. Syst.* 18, 2, Article 25 (March 2022), 16 pages.
https://doi.org/10.1145/3495210

1 INTRODUCTION

In the past few years, brain tumors have become one of the most deadly cancers in the world, especially for relatively young patients. Brain tumors can generally be divided into two types: (1) primary brain tumors that originate in the brain, and (2) secondary brain tumors metastasized from other organs. In particular, gliomas are the most common malignant tumors that account for about 75% of all the brain cancers. Based on the growth potential and aggressiveness of the tumor, gliomas are categorized into four grades: Grades I and II are often referred to as **low-grade gliomas (LGG),** while grades III and IV are referred to as **high-grade gliomas (HGG).**

Authors' address: W. Wang and B. Lin, University of California, San Diego, Department of Electrical and Computer Engineering, La Jolla, CA, 92093-0407; emails: {wweijia, billlin}@eng.ucsd.edu.

1550-4832/2022/03-ART25 $15.00
https://doi.org/10.1145/3495210

Currently, **Magnetic Resonance Imaging (MRI)** modalities are the most commonly utilized technique for the brain tumor diagnosis. Various MRI modalities highlight different tissue properties, and brain tumors can be further categorized and segmented into multiple sub-regions. However, radiation therapists manually labeling the scans is burdensome, inefficient, and requires high technical expertise. In this context, there has been an emerging need for automatic brain tumor segmentation, and deep learning techniques are introduced due to their recent considerable success in image processing applications.

Multimodal Brain Tumor Segmentation Challenge 2018 (BraTS 2018) is a public benchmark that provides a set of 3D MRI scans with ground truths labeled by human experts, and the task is to develop machine learning algorithms to produce the segmentation labels of different glioma subregions. In particular, the training dataset of the challenge comprises 210 HGG and 75 LGG MRI cases. Each case consists of four MRI modalities of shape $240 \times 240 \times 155$, including (1) native (T1), (2) post-contrast T1-weighted (T1Gd), (3) T2-weighted (T2), and (4) T2 **Fluid Attenuated Inversion Recovery (FLAIR)** volumes. The ground truth segmentation is also $240 \times 240 \times 155$ volumetric images, which are manually delineated by one to four raters according to the same annotation protocol. Three labels are provided along with an additional background label, i.e., (1) the **necrotic and non-enhancing tumor core** (**NCR & NET**, label 1), (2) the peritumoral edema (ED, label 2), (3) and the GD-enhancing tumor (ET, label 4). Furthermore, the challenge participants are expected to segment the images into three sub-regions: (1) the ET region, (2) the **tumor core (TC)**, which is the combination of NCR, NET, and ET, and (3) the **whole tumor (WT)**, which includes all the three tumor labels. In addition, the benchmark provides 66 unlabeled validation cases and 161 unlabeled testing cases, based on which the participating algorithms are evaluated and the final ranking is computed. The number of LGG and HGG subjects are not specified in the validation and testing datasets.

While a number of **convolutional neural networks (CNNs)** and **fully convolutional networks (FCNs)** have been proposed with the growing demand and interest in automatic brain tumor segmentation, a major bottleneck of this application is the volumetric multi-channel modality images that take up significant memory and computational power, which can be expensive even for the latest and most powerful GPUs. For example, a whole multi-modal MRI image of the BraTS 2018 challenge [15] cannot fit into one single GPU and it needs to be cut into patches during training and inference. Furthermore, lighter platforms, e.g., medical devices, generally have more limited on-device memory and computational power. However, massively parallel deep learning accelerators have been developed to exploit low-bit-width arithmetic. For instance, NVIDIA has provided a **NVIDIA Deep Learning Accelerator (NVDLA)** framework [19] to address the computational demands of inference, which allows using multiple data types across its various functional units to save area and computational power, including as little as binary integers. Deep neural networks that are available to operate in a low-precision manner can be deployed using deep learning accelerators to improve the chip design by allowing more cores on the chip with limited area. Therefore, the model optimization and acceleration play a critical role for practical deployment.

Basically, storage cost can be reduced by using low-precision parameters, whereas cheap computational cost can be achieved by performing low-bit-width arithmetic, which takes advantages from both low-precision weights and activations. However, the performance of a full-precision network can be vulnerable when converting the model into fewer bits. In general, a model gets prohibitively ruined by inferring with directly "truncated" low-bit-width arithmetic. In this work, we propose a quantization technique along with a training strategy that supports the volumetric segmentation with the dot-product operations in an integer-arithmetic manner. The floating-point decoding and encoding phases are deferred until the end of layers.

The rest of this article is organized as follows: Section 2 introduces some neural network architectures for image segmentation and some others' works to compress and accelerate deep neural networks. Section 3 describes the formulation and training algorithm of our quantization approach, as well as a procedure to perform integer-arithmetic operations for post-training inference. Section 4 evaluates our approach in comparison with the full-precision model. Section 5 concludes the article.

2 RELATED WORK

2.1 Automatic Volumetric Segmentation

State-of-the-art works employ **fully convolutional networks (FCNs)** [14] for automatic brain tumor segmentation. Different from other common convolution neural networks that use a fully connected layer at the end, FCNs also employ a convolutional layer as the last layer to produce a pixel-wise prediction. In particular, a fundamental FCN architecture, namely, U-Net [23], consists of a contracting encoder (a.k.a. analysis path) and a successive expanding decoder (a.k.a. synthesis path). The encoding part analyzes the input image and interprets it as a feature map, which is then fed into the decoder. Moreover, high-resolution activations in the analysis path are concatenated with up-sampled outputs in the synthesis path through shortcut connections to achieve better localization performance. Due to the symmetric fully convolutional architecture, the decoding part constructs a label map with the same size of the input image, each of whose channels corresponds to a segmentation label. Within a channel, every pixel indicates the probability of the corresponding label being positive.

Though U-Nets have achieved an accuracy close to human performance in segmenting 2D images, when it is applied to volumetric medical images, 3D images have to be processed as multiple 2D slices and hence it fails to capture the relationship of adjacent slices. Therefore, some later works further propose volumetric extensions of the U-Net to produce smoother volumetric segmentation.

In particular, the authors of U-Net also propose their feasible solution to the volumetric segmentation problem, namely, 3D U-Net [29], by replacing the 2D convolutions in U-Net their 3D counterparts. An overview of the 3D U-Net is illustrated in Figure 1. As can be seen in Figure 1, like U-Net, 3D U-Net comprises the left analysis path and the right synthesis path. In particular, each stage of the encoder consists of two 3D convolutional layers with a kernel size of $3 \times 3 \times 3$ and a 3D max pooling layer to down-sample the feature map. On the other side, there are also two $3 \times 3 \times 3$ convolutions at each stage, and the up-sampling is performed with an up-convolutional layer (while some later works replace it with a nearest-neighbor up-sampling layer). The last layer of the network performs a $1 \times 1 \times 1$ convolution that resizes the number of output channels to match the number of labels. However, while a couple of 2D images easily fit into a single GPU, whole 3D images can be too big for the GPU memory, especially for training the network, since large memory footprint has to be stored for back-propagation. As one of the main bottlenecks of 3D U-Net, the whole volume sometimes has to be divided into several patches and fed sequentially into the network.

3D U-Net has been serving as a prototype for automatic volumetric segmentation and many later approaches are developed based on the 3D U-Net architecture and modules.

For example, Reference [26] proposes multi-level deep supervision based on the 3D U-Net architecture, in which the three stages in the synthesis path are referred to as three different levels: lower layers, middle layers, and upper layers. Besides connecting to the next level, the lower and middle levels (note the upper level is the final stage) are also followed by up-convolutional blocks that upscale their reconstructions to match the input resolution. Therefore, each of the three levels

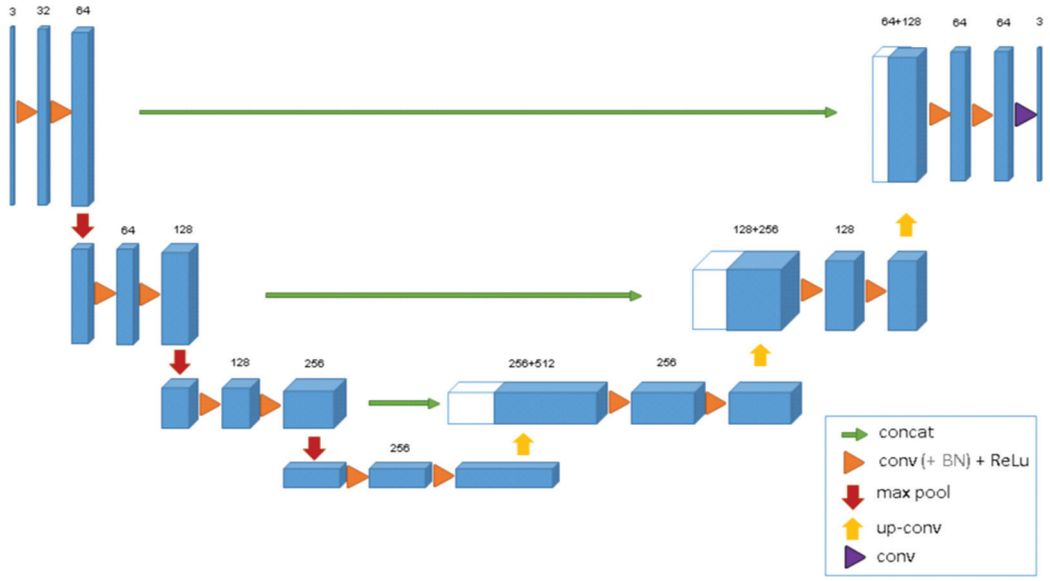

Fig. 1. The 3D U-Net architecture. Activations are shown as blue cuboids, whereas layers and other operations are displayed as arrows. The numbers above cuboids denote the channels of activations.

separately produces a segmentation output with the same resolution. It is discussed that the back-propagation performance is improved by calculating losses for the three different outputs, due to the fact that direct supervision on the hidden layers is more effective for the gradient computation.

V-Net [16], which is another volumetric derivation of U-Nets, replaces the pooling layers of the contracting path with 3D convolutions. It is discussed in their paper that convolutions can be applied to reduce the activation resolution by appropriately selecting kernel size and stride, i.e., a kernel size of $2 \times 2 \times 2$ and a stride of 2 halve the resolution of activations. The volumetric convolutional layers increase the receptive field and save the memory footprint during training, since they do not need to record the switches that associate the output and input of pooling layers for back-propagation. In addition, each stage (in both the encoder and the decoder) is a residual block in which the input is, after processed by the ReLU non-linearity, added directly to the output of the last convolutional layer. Compared with the non-residual U-Net architecture, the residual modules in V-Net help the network to better converge and achieve higher performance. The very last convolutional layer is similar to that of 3D U-Net, which has a kernel size of $1 \times 1 \times 1$ and it produces a probabilistic segmentation map by applying a voxel-wise softmax function to its output.

In addition, Attention U-Net proposes to highlight the more relevant activations with soft attention modules. To be specific, the authors argue that activations in the synthesis path are relatively imprecise, since they are constructed by the up-sampling. Standard U-Nets address the issue with the shortcut paths connecting the analysis path and synthesis path, which, nonetheless, brings heavy redundancy and distracts the network. Therefore, Attention U-Net introduces additive soft attention implemented at the shortcut connections on a per-voxel basis, which reduces the computational cost and improve the segmentation performance. Their experiments show that as the number of training epochs increases, Attention U-Net learns to focus more on the foreground areas, and they achieve a clear improvement in dice score compared with the standard 3D U-Net.

According to Reference [9], it is commonly believed that more specialized architectures are required for different segmentation tasks and there have been huge amounts of works designed for

few or even a single dataset in recent years, which results in troubles for researchers to identify and select the architecture that fits the best in their scenarios. Moreover, those kinds of models generally suffer from overfitting and a lack of adaptation. In this context, Reference [9] proposes nnU-Net with adaptive architectures. In particular, three basic U-Net architectures are included in nnU-Net: 2D-U-Net, 3D-U-Net, and 3D-UNet Cascade, which consists of two 3D-UNet cascaded in sequence to address the memory constraints for large images. All the three architectures are initialized with a specific patch size, batch size, and number of feature maps, which are automatically adjusted according to the median plane size of the training data. A five-fold cross-validation is utilized to choose an architecture (or ensemble) and its topology with the best performance as the final model. Experimental evaluations show that nnU-Net achieves state-of-the-art performance on several distinct datasets and even outperforms the specialized models for some tasks.

While there have been proposed many works on variant specialized architectures for different segmentation applications, they are in generally usually based on the standard 3D U-Net. Therefore, in this article, we adopt the basic 3D U-Net as our segmentation model with some small modifications to better fit with our problem and approach, which will be explained in the evaluation section.

2.2 Network Compression and Acceleration

As discussed in the previous section, the enormous size and computational cost are currently bottlenecks for these models to be practically deployed. Many methods have been proposed to overcome the efficiency challenge, including quantization [1, 4, 6, 7, 10, 11, 22, 25, 27, 28], pruning, [7], and other encoding approaches [6, 7]. In particular, these works roughly fall into two categories.

The first type of works focuses on the on-device storage optimization but gain no computational efficiency improvement to support real-time applications. Although network parameters are compressed into tiny models, they need to be converted back into full-precision values and the computation is carried out using floating-point representation. For example, Reference [7] proposes to "prune" network synapses by forcing some of the weights to zero. In addition, the non-zero weights are clustered into groups and encode the entries using Huffman coding to further reduce the storage per weight. The model can be decoded back into full precision with the code book, and they achieve significant compression rate with negligible accuracy loss. The same authors also propose in Reference [28] to quantize weights into ternary values (2-bit weights), which causes very little accuracy degradation by training the quantization centroids. Reference [22] considers the brain segmentation problem and derives their "3DQ" approach based on Reference [28], which also quantizes the full-precision weights into 2 bits. They further incorporate an additional factor to scale the quantization centroids and achieve near full-precision accuracy on two medical imaging 3D segmentation datasets. However, the downside of such technologies is that they do not bring any computational benefits and may even possibly worsen the speed due to the additional decoding phase.

Alternatively, some works directly train the parameters to be integers. In addition to the storage overhead reduction, such approaches also effectively reduce the number of floating-point operation for inferences and improve the computational efficiency. For instance, it is proposed in References [1, 25] to operate the neural networks, including training and inferences, with 8-bit-integer weights and activations, where the quantization centroids of Reference [25] is uniformly distributed between -1 and 1, while those of Reference [1] are derived from the maximum absolute values of the weights and activations. Further, DoReFa-Net [27] allows the weights and activations to be quantized into arbitrary bits. They decide the quantization centroids such that the value range of weights is limited to $[-1, 1]$ while activations are bounded within $[0, 1]$. These works directly approximate the full-precision model with low-bit-width values so they are able

to run with integer arithmetic. However, some approaches use the low-precision integer to index the quantization centroids. In Reference [10], weights and activations are encoded as non-negative integers on a per-layer basis and can be decoded into full-precision approximations with a pair of shifting and scaling operations. The shifting and scaling factors are directly derived from the full-precision model during the training phase such that all real-valued points fall within the range between the smallest and the greatest quantization centroids, i.e., the clustering is simply performed by taking the full-precision range with a uniformly partition on it. Moreover, they propose a "batch-normalization folding" technique that absorbs the parameters of batch normalization into the previous convolutional or fully connected layer to reduce the computational complexity.

However, clear accuracy drops are present in the above approaches. The reasons include:

- In Reference [10], an exponential moving average with the smoothing parameter being close to 1 is used to derive the factors for activations. Since the intermediate activations differ from sample to sample, this makes the factors highly depend on the latest batch and relatively volatile.
- Since the weights and activations of a well-trained model mostly follow the Gaussian and half-wave Gaussian distributions [3], respectively, a significant amount of points are concentrating around the mean value and 0. Therefore, for both weights and activations, it is unnecessary and sub-optimal for References [1, 10] to span a range covering all samples, especially when using a large mini-batch size or there exist extreme outliers. However, References [25, 27] force the weights between -1 and 1, which as well reduces the performance compared with networks with no such constraints.
- The centroids of weight approximations are not trained in these approaches, but directly computed from the full-precision distributions such that the same ranges are spanned by the quantization centers with the full-precision weights and activations, which makes the accuracy of the full-precision model form an upper bound of the quantized performance. However, due to the definite error introduced by representing continuous ranges using discrete centroids, the drop on performance is inevitable.

The motivation of this work is to improve the previous approaches and address the issues discussed above. In comparison with the first type of works, our approach grants an efficiency improvement on volumetric segmentation with the integer-arithmetic dot-product operations. Moreover, we allow using arbitrary bits for the quantization and aim to reduce the performance degradation by directly training the quantization factors together with other network parameters to minimize the segmentation loss rather than deriving them from the full-precision model, which grants the low-precision model a potential to even outperform the full-precision network.

3 TRAINED AFFINE MAPPING APPROACH

3.1 Learning the Mapping of Weights

In our approach, each full-precision synaptic weight $\tilde{w}_i$ is encoded as an m-bit integer $g_i \in \mathcal{G}$, where $\mathcal{G} = \{0, 1, \ldots, 2^m - 1\}$ is an affine space. Through a 1D affine mapping, an integer representation can be converted to the following full-precision approximation:

$$\hat{w}_i = S_w(g_i - Z), \tag{1}$$

where the linear transformation and the translation are conducted by the scaling factor S_w and the translation factor Z,[1] respectively, which are both floating-point numbers. Due to the properties of

[1]We define S_w and Z on a *per-layer* basis, namely, different pairs of factors are used for different layers. Note that this can be easily extended to a *per-kernel* basis to achieve better performance with acceptable additional storage overhead and negligible extra computational cost.

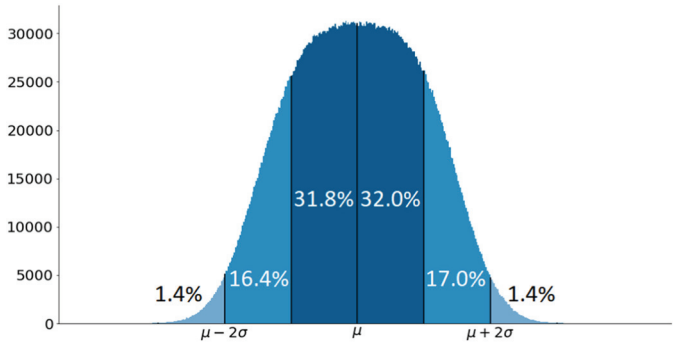

Fig. 2. The weight distribution of a hidden layer of the pre-trained full-precision model where each band has a width of one standard deviation. The weights generally follow a Gaussian distribution and the values less than one and two standard deviations away from the mean account for 63.8% and 97.2% of the set, respectively.

affine mapping, the uniform codes in $\mathcal{G}$ are mapped to a uniform distribution over the full-precision space, implying our affine mapping approach is essentially a uniform quantizer. However, different from g_i that is non-negative integers, the full-precision centroids span a range over both *positive* and *negative* numbers with the utilization of translation operation. Moreover, by appropriately adjusting the translation factor Z, our approach allows an *asymmetric* partition over the range of positive and negative values. For example, using $m = 2$ bits, $S_w = 0.5$ and $Z = 1$, $g_i = 0, 1, 2, 3$ correspond to the real-valued approximations $\hat{w}_i = -0.5, 0, 0.5, 1$, respectively. This provides us a substantial flexibility in determining and tuning the centroids of our quantizer.

Our training algorithm starts from a pre-trained model, where the full-precision weights practically follow a Gaussian distribution. Figure 2 illustrates the weight distribution of a hidden layer in the synthesis path of the pre-trained model, in which μ and σ stand for the mean and standard deviation of the distribution, respectively. Empirically, we find that the weight approximation space initialized across the interval of $[\mu - 2\sigma, \mu + 2\sigma]$ leads to a faster convergence and a higher accuracy, compared with the initialization over $[\mu - \sigma, \mu + \sigma]$. This is potentially due to the fact that only around 68% (64% in Figure 2) points of a Gaussian distribution fall into the range of $[\mu - \sigma, \mu + \sigma]$. While the rest 32% (36% in Figure 2) have relatively large magnitudes and are hence too critical to be clipped. However, more than 95% of the points are within the range of $[\mu - 2\sigma, \mu + 2\sigma]$ with the rest points tending to be outliers. Thus, clipping the weights beyond $[\mu - 2\sigma, \mu + 2\sigma]$ does not impact the network a lot. In particular, denote the range by $[r_{\min}, r_{\max}]$. We define $r_{\min}$ and $r_{\max}$ as follows:

$$r_{\min} = \mu_{\tilde{W}} - 2\sigma_{\tilde{W}}, \tag{2}$$

$$r_{\max} = \mu_{\tilde{W}} + 2\sigma_{\tilde{W}}, \tag{3}$$

where $\mu_{\tilde{W}}$ and $\sigma_{\tilde{W}}$ are the per-layer mean and standard deviation of the full-precision weights $\tilde{W}$. Then, S_w and Z are initialized as follows:

$$S_w = \frac{r_{\max} - r_{\min}}{2^m - 1}, \tag{4}$$

$$Z = -\frac{r_{\min}}{S_w}. \tag{5}$$

During training, unlike the weight quantization approach in Reference [10] that simply derives their quantization parameters such that the smallest and the greatest centroids equal to the

minimal and maximal real-valued weights, the key idea of our approach is re-training the latent full-precision weights to compensate the error introduced by the low-bit-width representation, while the scaling factor S_w and the translation factor Z are concurrently trained against the classification loss independently from other parameters.

During feed-forward pass, each latent full-precision weight $\tilde{w}_i$ is quantized into the low bit-depth code g_i according to:

$$g_i = \text{clip}\left(\text{round}\left(\frac{\tilde{w}_i}{S_w} + Z\right), 0, 2^m - 1\right), \tag{6}$$

where

$$\text{clip}\left(x, x_{\min}, x_{\max}\right) = \max\left(x_{\min}, \min(x, x_{\max})\right). \tag{7}$$

Then, we use the full-precision approximation expressed in Equation (1) to conduct the inference and calculate the loss L.

In back-propagation phase, the latent full-precision weights are injected back in preparation for the update. We use the gradient w.r.t. the weight approximation $\hat{w}_i$ to update the full-precision weight $\tilde{w}_i$:

$$\frac{\partial L}{\partial \tilde{w}_i} = \frac{\partial L}{\partial \hat{w}_i}. \tag{8}$$

Additionally, the scaling factor S_w and the translation factor Z are updated concurrently. Based on chain rule [2], the gradient w.r.t. S_w can be computed from Equation (1) as follows:

$$\frac{\partial L}{\partial S_w} = \sum_i \frac{\partial L}{\partial \hat{w}_i}\frac{\partial \hat{w}_i}{\partial S_w} = \sum_i (g_i - Z)\frac{\partial L}{\partial \hat{w}_i}. \tag{9}$$

Similarly, we calculate the gradient w.r.t. Z as follows:

$$\frac{\partial L}{\partial Z} = \sum_i \frac{\partial L}{\partial \hat{w}_i}\frac{\partial \hat{w}_i}{\partial Z} = -S_w \sum_i \frac{\partial L}{\partial \hat{w}_i}. \tag{10}$$

Then, the latent full-precision weights $\tilde{w}_i$, the scaling factor S_w and the translation factor Z are updated together directly towards the classification loss. As a result, in the next iteration, the full-precision weights $\tilde{w}_i$ and the affine factors have changed, hence the assignment g_i and the approximations $\hat{w}_i$ also have a probability to be different from the previous iteration, which in turn will apply an influence on the gradient w.r.t. S_w and Z in the next back-propagation stage. Generally speaking, our training procedure works essentially in a close manner of relaxation algorithms [13, 18], which repeatedly update both the centroids and the the assignments, while in our problem, the points $\tilde{w}_i$ are moving as well.

3.2 Linear Mapping of Activations

In the previous section, we discussed how full-precision weights can be approximated using m-bit non-negative integers along with an affine mapping operation. Nevertheless, practical implementations also limit the activation precision due to the efficiency challenges. In addition to the weight quantization, we also propose to reduce the activation bit-width based on the **half-wave Gaussian quantization (HWGQ)** approach proposed in Reference [3].

It is discussed in Reference [3] that batch normalization [8] and ReLU [5] are widely employed in state-of-the-art CNNs. In particular, the outputs of a convolutional layer are normalized by batch normalization into a Gaussian distribution with zero mean and unit variance. Moreover, ReLU is a non-linearity that simply drops the negative samples as follows:

$$\phi(x) = \max(0, x), \tag{11}$$

and it further trims activations into a half-wave Gaussian distribution.

Based on this observation, given a full-precision activation $\tilde{x}_i$, we would like to encode it with a p-bit unsigned integer $h_i \in \{0, 1, \ldots, 2^p - 1\}$, which corresponds to the floating-point approximation $\hat{x}_i$. Further, since the real-valued activations $\tilde{x}_i$ are half-wave Gaussian distributed with zero mean and unit variance, an optimal quantizer $Q(\tilde{x}_i) = \hat{x}_i$ can be computed by sampling from a standard distribution and applying an iterative relaxation algorithm until convergence. In particular, since the lower bound is explicitly defined at 0, which is also the crest of the distribution, we drop the translation term and approximate activations with the following linear mapping function:

$$\hat{x}_i = Q(\tilde{x}_i) = S_a h_i, \tag{12}$$

where S_a is a linear scaling factor of floating-point value, and the assignment variable h_i can be derived from $\tilde{x}_i$ as follows:

$$h_i = \text{clip}\left(\text{round}\left(\frac{\tilde{x}_i}{S_a}\right), 0, 2^p - 1\right). \tag{13}$$

For the scaling factor S_a, recall that batch normalization generally produces response activations that are Gaussianly distributed with zero mean and unit variance across all layers. Therefore, given the quantization precision p, there is no need to define or train different factors for different layers, and a same linear quantizer can be used across the network. In particular, an optimal quantizer defined on a distribution can be expressed in the sense of quadratic error minimization:

$$\arg\min_{Q} \int \varphi(x)(Q(x) - x)^2 dx. \tag{14}$$

Although Lloyd's algorithm [13] can be generally applied to solve the clustering problems, it breaks the linear constraints on centroids. Therefore, we propose a variation of Lloyd's algorithm to overcome this issue. During the step for center update, instead of computing the new centroids by simply taking an average for each cluster, all points are first normalized on a per-cluster basis such that they are on the magnitude of one scaling factor, and then the mean value is derived from all normalized samples as the updated factor S_a. In other words, we update S_a as follows:

$$S_a = \frac{\sum_{i \text{ for } h_i \neq 0} \frac{\tilde{x}_i}{h_i}}{\sum_{i \text{ for } h_i \neq 0} 1}. \tag{15}$$

A brief description of our clustering algorithm is summarized in Algorithm 1.

ALGORITHM 1: The computation of activation scaling factor S_a

Data: p, $\tilde{X} = \{\tilde{x_i}\}$ sampled from a standard half-wave Gaussian distribution
Initialize $S_a, H = \{\tilde{h_i}\}$
while H not converged **do**
  step 1: update centroids $\{0, S_a, \ldots, (2^p - 1)S_a\}$
  step 2: update H according to Equation (13)
  step 3: update S_a according to Equation (15)
end
Return: S_a

In feed-forward pass, the real-valued activations $\tilde{x}_i$ are quantized into the floating-point approximations $\hat{x}_i$. However, another problem is introduced by the quantization of activations, that during back-propagation phase, the stair-like rounding operation in Equation (13) makes the function completely non-differentiable and breaks the gradient chain. To address this issue, rather than propagating gradient back from the approximations $\hat{x}_i$ to the full-precision outputs $\tilde{x}_i$, we explicitly

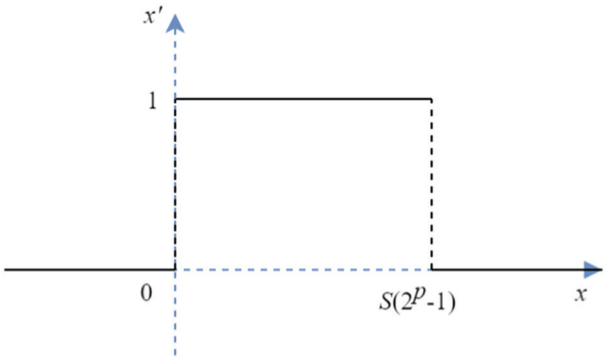

Fig. 3. The gradient of approximations w.r.t. the real-valued activations based on the gradient clipping approach.

define that the real-valued activations directly inherit the derivatives from their approximations and skip computing the derivatives of the non-differentiable rounding operations.

Moreover, it is discussed in Reference [3] that, since the activations with extremely large values are bounded to the greatest centroid by quantization, it causes a problem of gradient mismatch [12] by deriving derivatives from the quantized results. Therefore, we adopt the gradient clipping scheme proposed in Reference [21] and discard the derivatives of the real-valued activations that are beyond the range of our quantization centroids. To be specific, the gradient w.r.t. the activations before quantization is computed as follows:

$$\frac{\partial L}{\partial \tilde{x}_i} = \frac{\partial L}{\partial \hat{x}_i} \frac{\partial \hat{x}_i}{\partial \tilde{x}_i}, \tag{16}$$

where $\frac{\partial \hat{x}_i}{\partial \tilde{x}_i}$ is illustrated in Figure 3.

3.3 Optimizing 3D U-Net-based Brain Tumor Segmentation

As discussed above, our proposed approach encodes weights and activations with m-bit and p-bit unsigned integers, respectively. However, it can be seen in Equations (1) and (12) that the approximations after decoding are still carried out in a floating-point format, which do not effectively reduce the computational power. In this section, we study that after training, how inference can be operated in a more efficient manner. Specifically, the inference of a neuron followed by batch normalization and ReLU can be straightforwardly implemented as follows:

1. Activation decoding: $\hat{x}_i = S_a h_i$;
2. Weight decoding: $\hat{w}_i = S_w(g_i - Z)$;
3. Dot-product: $\tilde{y} = \sum_i \hat{w}_i \hat{x}_i + b$;
4. Batch normalization & ReLU: $\tilde{z} = \max(0, \frac{\tilde{y}-\mu}{\sqrt{\sigma^2+\epsilon}})$;
5. Activation encoding: $h_{\text{out}} = \text{clip}(\text{round}(\frac{\tilde{z}}{S_a}), 0, 2^p - 1)$;

where b is the bias of convolutional layers.

By combining steps 1, 2, and 3, the intermediate output $\tilde{y}$ can be simplified as follows:

$$\tilde{y} = S_w S_a \sum_i g_i h_i - S_w S_a Z \sum_i h_i + b. \tag{17}$$

As defined in the previous sections, g_i and h_i are non-negative integers of m and p bits, respectively. Therefore, the *dot-product* operations can be essentially performed using *integer-only arithmetic*

with the floating-point multiplications deferred after that. Furthermore, parameters in steps 4, 5, and Equation (17) become constants once the training is complete, hence can be absorbed into a single 2D affine function along with the rounding and clipping non-linearity:

$$h_{\text{out}} = \text{clip}\left(\text{round}\left(Av_1 + Bv_2 + C\right), 0, 2^p - 1\right), \tag{18}$$

where

$$v_1 = \sum_i g_i h_i, \tag{19}$$

$$v_2 = \sum_i h_i, \tag{20}$$

$$A = \frac{S_w}{\sqrt{\sigma^2 + \epsilon}}, \tag{21}$$

$$B = -\frac{S_w Z}{\sqrt{\sigma^2 + \epsilon}}, \tag{22}$$

$$C = \frac{b - \mu}{S_a \sqrt{\sigma^2 + \epsilon}}. \tag{23}$$

In conclusion, A, B, and C can be pre-computed and the inference procedure reduces into two steps:

1. Compute v_1 and v_2 according to Equations (19) and (20) in the integer domain;
2. Compute h_{out} according to Equation (18) in the floating-point domain.

Note that the floating-point operations in step 2 are conducted on a per-neuron basis, which is not in domination of the computational complexity. In this way, our trained affine mapping approach efficiently produces an improvement in terms of both storage and computational cost.

Moreover, some network architectures employ shortcut connections that concatenate the outputs of two layers. For example, in regards of 3D U-Net, the output at each stage of the encoder is directly concatenated with the input of the decoder at the same stage. However, since the same activation scaling factor S_a is adopted throughout all layers, a full-precision activation $\tilde{z}$ shall be encoded to the same unsigned integer h_{out} regardless of which layer it belongs to. Therefore, stacking the intermediate activation $\tilde{z}$ in step 4 is essentially equivalent to stacking the encoded value h_{out} in step 5, which leads to no additional operations besides independently computing h_{out} for the two layers according to Equation (18).

4 EVALUATION

4.1 Experimental Setup

We evaluate our approach on the BraTS 2018 challenge discussed in Section 1. In particular, while the ground truth segmentation of the official BraTS 2018 validation dataset is not publicly available, to perform hyper-parameter tuning and provide clear comparison between the prediction and ground truth, in spite of the official validation dataset, we use 10% of the training dataset as our validation dataset in our experiments and train the models with the rest 90% samples.[2] The N4ITK bias correction [24] approach is applied to all the MRI images to reduce the bias caused by using different scanners. We then clip the greatest as well as the smallest 2% voxels in each channel (modality) to remove outliers. Last, images are normalized to zero mean and unit variance on a per-channel basis while the non-brain regions are set to 0.

[2]Note that our experiments aim to illustrate the difference in performance between the full-precision and our quantized models. The full-precision model serves as a baseline and its absolute performance is not critical.

We employ the minorly modified 3D U-Net [29] in our experiments. In particular, we set the number of input channels to 4 to match the 4 modalities of the MRI scans. Moreover, we follow the approach of the first-place winner of BraTS 2018 [17], where the output of the network has three channels corresponding to the three tumor sub-region labels, i.e., the **whole tumor (WT)**, the **tumor core (TC)**, and the **enhancing tumor (ET)**, and they are then connected to a sigmoid activation function that produces the predicted probabilities of each label. As described in Reference [29], batch-normalization is introduced before each ReLU non-linearity, which is also adopted in our implementation so the architecture is compatible with our activation quantization scheme, as discussed in Section 3.2. The up-convolutional layers in the original architecture are replaced with up-sampling layers. (Note that while we choose to use up-sampling layers, it as well fits with our approach if applying batch-normalization and ReLU after each up-convolutional layer.) We use the multi-class dice loss function based on the dice loss proposed in Reference [16]. According to Reference [16], denote by p_i and q_i the voxels at the same location in the prediction and ground truth for a class, respectively, the single-class dice loss is defined as follows:

$$L_{\text{class}} = -\frac{2\sum_i p_i q_i}{\sum_i p_i^2 + \sum_i q_i^2}. \tag{24}$$

Given the dice loss of three tumor sub-regions L_{WT}, L_{TC}, and L_{ET}, our loss function is simply the summation of these three scores as follows:

$$L = L_{\text{WT}} + L_{\text{TC}} + L_{\text{ET}}. \tag{25}$$

In other words, the three tumor sub-regions are assigned with identical weights.

We randomly sample patches of size $96 \times 96 \times 96$ with a batch size of 2 to train the networks. All the low-precision models start from the pre-trained full-precision model and we re-train them using our quantization algorithm. We adopt an Adam optimizer to update the scaling and translation factors S and Z, with the learning rate initialized to be 1e-6. (While the gradient w.r.t. these factors is accumulated by every weight and the amount of weights in a 3D convolutional layer can be significant, it empirically works better with relatively small learning rates.) The other parameters (weights and biases, etc.) are tuned by another Adam optimizer with a learning rate of 1e-4 and a weight decay of 1e-5. We divide the learning rates of both optimizer by 10 every 1,000 batches and the models are trained for 4,000 batches.

The validation images are padded with 0 and partitioned into multiple $96 \times 96 \times 96$ patches. We feed the network to produce predictions of the same shape and reconstruct the label map. Different from the training procedure where we use probabilistic output to compute the dice loss, in evaluation, we binarize the output (before the sigmoid) with a threshold of 0 and use the binarized reconstruction to compare with ground truth and compute dice scores.

4.2 Segmentation Results

Besides our **trained affine mapping (TAM)** approach, we also evaluate the **full-precision (FP)** model along with three low-precision CNN training and inference techniques as our baselines: 3DQ [22], DoReFa-Net [27], and the naïve **Fixed-Point Number (FPN)** representation [20]. The first two techniques are previously discussed in Section 2, and FPN is the simplest quantization approach, which allows the valid bits to represent any continuous power-of-2 fractional values. For example, the 2-bit FPN 1.1, which consists of an integer bit and a fractional bit, encodes the decimal value $1 \times 2^0 + 1 \times 2^{-1} = 1.5$. In our FPN experiments, the weights are quantized using an additional bit as the sign bit, and we allow the represented bits (as long as they are continuous) to be away from the radix point while skipping the other more significant digits, e.g., using three unsigned bits 101 as in 0.0101 to encode $1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} = 0.3125$. In particular, all

Table 1. Validation Results (Mean Dice Scores) on the BraTS 2018 Dataset

	W bits	A bits	WT	TC	ET
FP			0.888 (0.0006)	0.801 (0.0073)	0.762 (0.0029)
FPN	8	8	0.744 (0.0456)	0.361 (0.1542)	0.144 (0.1786)
	4	4	0.666 (0.0921)	0.189 (0.1313)	0.074 (0.1039)
	2	2	0 (0)	0 (0)	0 (0)
3DQ	2	float	0.885 (0.0026)	0.792 (0.0014)	0.747 (0.0029)
DoReFa	8	8	0.874 (0.0036)	0.796 (0.0047)	0.754 (0.0047)
	4	4	0.873 (0.0017)	0.789 (0.0018)	0.750 (0.0022)
	2	2	0.874 (0.0024)	0.787 (0.0032)	0.750 (0.0038)
Ours	8	8	0.887 (0.0019)	0.801 (0.0020)	0.767 (0.0021)
	4	4	0.889 (0.0028)	0.797 (0.0016)	0.761 (0.0006)
	2	2	0.884 (0.0014)	0.793 (0.0014)	0.760 (0.0006)

W bits and A bits represent the number of bits used to encode weights and activations. WT, TC, and ET stand for the whole tumor, the tumor core, and the enhancing tumor, respectively. Experiments are repeated three times and we report the average results for validation, while standard deviations are shown in parentheses.

low-precision approaches except for FPN are re-trained from a pre-trained full-precision model, which is also used to derive the quantized weights for FPN so the represented bits minimize the mean squared error. In addition, since the activations are dependant on the input, it does not make sense for FPN to dynamically compute the best scales, thus, we simply truncate the activations with all bits being fractional bits. All models are trained on a NVIDIA RTX 2080 Ti GPU, and we compute the dice coefficients for the three tumor sub-regions. All experiments are repeated three times, and the average results are summarized in Table 1.

Among all low-precision techniques, FPN achives the worst performance. Even when using 8-bit weights and activations, the average dice for WT drastically drops from 0.888 to 0.744, while the TC and ET dice scores fall below 0.4 and 0.2, respectively. The performance further becomes worse with 4-bit weights and activations, and the model totally does not produce any useful information with 2-bit precision, which gets the dice coefficients of 0 for all the three sub-regions. The reason is that the error caused by such post-training compression approaches is not compensated, so the error accumulates throughout layers and critically hurts the model.

However, our trained affine mapping approach with 8-bit weights and activations achieves 0.887, 0.801, and 0.767 average dice for WT, TC, and ET, respectively, which are close to (or even better than) the full-precision model, while there is a dice loss of about 0.01 present in the 8-bit DoReFa-Net results. Moreover, TAM achieves negligibly small loss with 4-bit weights and activations (within 0.005 of the full-precision model), and about 0.01 degradation when using 2-bit precision, whereas DoReFa-Net shows a clearer performance drop in such cases. 3DQ achieves fairly good WT score, but it performs poorly for the TC and ET sub-regions. We also note that 3DQ uses floating-point activations and it is a compression approach that does not accelerate the inference, which is not as beneficial other baselines. The validation results, i.e., our approach outperforms other baselines and achieves a performance close to the full-precision model, verify that our factor-training scheme effectively tunes the centroids and reduces the loss introduced by the quantization.

In addition, we reconstruct the different parts of tumors (NCR & NET, ED, and ET) from our predicted labels to qualitatively illustrate our results. An axially sliced example from our validation set with the FLAIR modality as background is presented in Figure 4. As can be observed in the ground truth annotation, the red area (NCR & NET) has relatively more irregular shape, which

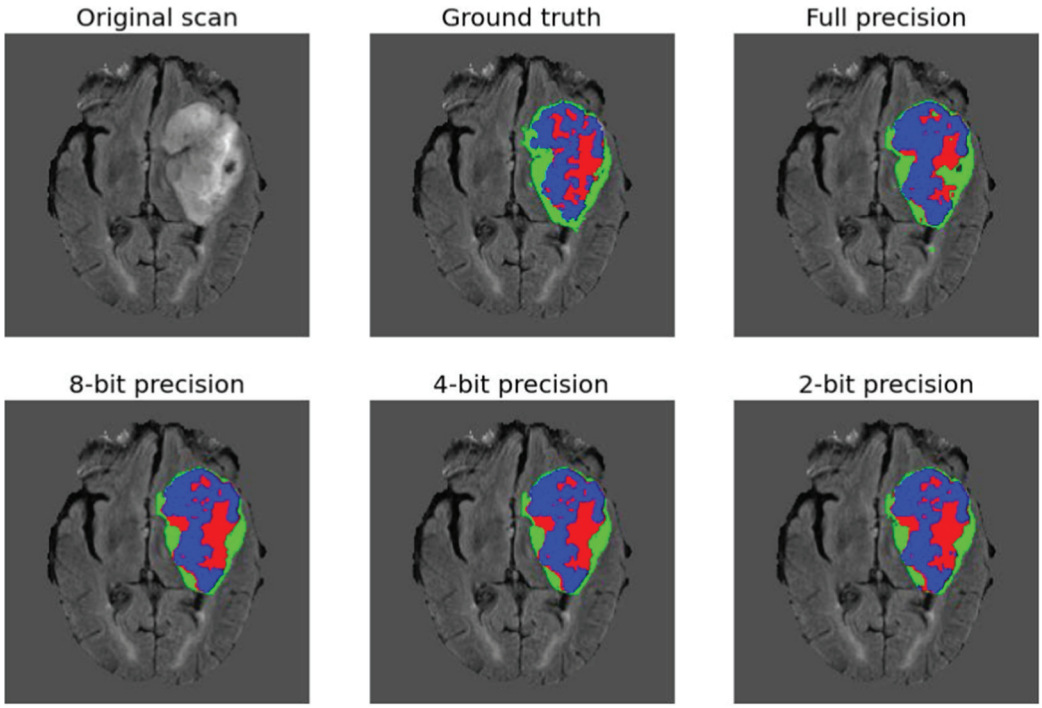

Fig. 4. An example of the predictions of the full-precision and our quantized models along with the ground truth annotations overlaid over the FLAIR MRI scan. The necrotic and non-enhancing tumor core (NCR & NET) are shown in red; the peritumoral edema (ED) is shown in green; and the GD-enhancing tumor (ET) is shown in blue. The prediction label WT is the combination of all the three colored areas, and TC is the union of red and blue.

makes it the most difficult part to accurately predict, while the green (ED) and blue (ET) regions also have some dotted details around their boundaries, i.e., some blue dots in the green area and green dots outside the main tumor. However, while all models perform badly in predicting the red region, the full-precision model additionally tries to capture the dotted feature of the ground truth, potentially due to the over-fitting problem, though regularization is already applied during training. However, it is almost impossible to perfectly predict these small dots and hence this actually increases the error of the full-precision model. Nevertheless, our quantized models produce relatively smoother annotations without these small dots. This might explain the reason why our quantized models sometimes outperform the full-precision model.

5 CONCLUSION

In this article, we consider the problem of optimizing the 3D U-Net with low-precision parameters and integer-arithmetic inference for efficient volumetric segmentation. In particular, we propose a trained affine mapping approach that encodes weights and activations as non-negative integers of dedicated bit-widths and recovers the floating-point approximations with affine mapping functions. The key idea of our work is that the scaling and translation factors for weights can be trained together with other parameters, whereas activations are generally normalized by batch normalization and **rectified linear units (ReLU)**, hence can be accurately approximated using the same function across all layers, which is pre-computed based on the standard half-wave

Gaussian distribution. In addition, with weights and activations encoded as low-precision integers, we propose to defer the floating-point computation of the affine mapping functions and combine it with the quantization procedure of the next layer. This technique simplifies the inference into two steps, in which the dot-product operations are carried out using unsigned-integer arithmetic and the floating-point multiplications are reduced onto a per-neuron basis. Evaluation results on the BraTS 2018 challenge show that the models quantized by our trained affine mapping algorithm using 2-bit weights and activations achieve a mean dice score within 0.01 relative to the full-precision model. Furthermore, our quantization achieves negligibly small degradation with 4-bit and 8-bit precisions.

REFERENCES

[1] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. 2018. Scalable methods for 8-bit training of neural networks. *CoRR* abs/1805.11046 (2018).

[2] Christopher M. Bishop. 1995. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY.

[3] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. 2017. Deep learning with low precision by half-wave Gaussian quantization. *CoRR* abs/1702.00953 (2017).

[4] Matthieu Courbariaux and Yoshua Bengio. 2016. BinaryNet: Training deep neural networks with weights and activations constrained to +1 or −1. *CoRR* abs/1602.02830 (2016).

[5] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Geoffrey Gordon, David Dunson, and Miroslav Dudík (Eds.), Vol. 15. PMLR, Fort Lauderdale, FL, 315–323. Retrieved from http://proceedings.mlr.press/v15/glorot11a.html.

[6] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir D. Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *CoRR* abs/1412.6115 (2014).

[7] Song Han, Huizi Mao, and William J. Dally. 2015. Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding. *CoRR* abs/1510.00149 (2015).

[8] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 448–456. Retrieved from http://jmlr.org/proceedings/papers/v37/ioffe15.pdf.

[9] Fabian Isensee, Jens Petersen, André Klein, David Zimmerer, Paul F. Jaeger, Simon Kohl, Jakob Wasserthal, Gregor Köhler, Tobias Norajitra, Sebastian J. Wirkert, and Klaus H. Maier-Hein. 2018. nnU-Net: Self-adapting framework for U-Net-based medical image segmentation. *CoRR* abs/1809.10486 (2018).

[10] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. 2017. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *CoRR* abs/1712.05877 (2017).

[11] Fengfu Li and Bin Liu. 2016. Ternary weight networks. *CoRR* abs/1605.04711 (2016).

[12] Darryl Dexu Lin and Sachin S. Talathi. 2016. Overcoming challenges in fixed point training of deep convolutional networks. *CoRR* abs/1607.02241 (2016).

[13] S. Lloyd. 1982. Least squares quantization in PCM. *IEEE Trans. Inf. Theor.* 28, 2 (Mar. 1982), 129–137. DOI: https://doi.org/10.1109/TIT.1982.1056489

[14] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2014. Fully convolutional networks for semantic segmentation. *CoRR* abs/1411.4038 (2014).

[15] B. H. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, J. Kirby, Y. Burren, N. Porz, J. Slotboom, R. Wiest, L. Lanczi, E. Gerstner, M. Weber, T. Arbel, B. B. Avants, N. Ayache, P. Buendia, D. L. Collins, N. Cordier, J. J. Corso, A. Criminisi, T. Das, H. Delingette, Ç. Demiralp, C. R. Durst, M. Dojat, S. Doyle, J. Festa, F. Forbes, E. Geremia, and B. Gloc. 2015. The multimodal brain tumor image segmentation benchmark (BRATS). *IEEE Trans. Med. Imag.* 34, 10 (2015), 1993–2024. DOI: https://doi.org/10.1109/TMI.2014.2377694

[16] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. 2016. V-Net: Fully convolutional neural networks for volumetric medical image segmentation. (2016). arXiv:cs.CV/1606.04797.

[17] Andriy Myronenko. 2018. 3D MRI brain tumor segmentation using autoencoder regularization. (2018). arXiv:cs.CV/1810.11654.

[18] S. Na, L. Xumin, and G. Yong. 2010. Research on k-means clustering algorithm: An improved k-means clustering algorithm. In *Proceedings of the 3rd International Symposium on Intelligent Information Technology and Security Informatics*. 63–67. DOI: https://doi.org/10.1109/IITSI.2010.74

[19] Nvidia. 2018. NVIDIA Deep Learning Accelerator. Retrieved from http://nvdla.org/.

[20] Erick L. Oberstar. 2007. Fixed-point representation & fractional math. *Oberstar Consult.* 9 (2007).

[21] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2012. Understanding the exploding gradient problem. *CoRR* abs/1211.5063 (2012).
[22] Magdalini Paschali, Stefano Gasperini, Abhijit Guha Roy, Michael Y.-S. Fang, and Nassir Navab. 2019. 3DQ: Compact quantized neural networks for volumetric whole brain segmentation. *CoRR* abs/1904.03110 (2019).
[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional networks for biomedical image segmentation. (2015). arXiv:cs.CV/1505.04597.
[24] N. J. Tustison, B. B. Avants, P. A. Cook, Y. Zheng, A. Egan, P. A. Yushkevich, and J. C. Gee. 2010. N4ITK: Improved N3 bias correction. *IEEE Trans. Med. Imag.* 29, 6 (2010), 1310–1320. DOI: https://doi.org/10.1109/TMI.2010.2046908
[25] Yukuan Yang, Shuang Wu, Lei Deng, Tianyi Yan, Yuan Xie, and Guoqi Li. 2019. Training high-performance and large-scale deep neural networks with full 8-bit integers. *CoRR* abs/1909.02384 (2019).
[26] Guodong Zeng, Xin Yang, Jing Li, Lequan Yu, Pheng-Ann Heng, and Guoyan Zheng. 2017. 3D U-Net with multi-level deep supervision: Fully automatic segmentation of proximal femur in 3D MR Images. 274–282. DOI: https://doi.org/10.1007/978-3-319-67389-9_32
[27] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. 2016. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR* abs/1606.06160 (2016).
[28] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. 2016. Trained ternary quantization. *CoRR* abs/1612.01064 (2016).
[29] Özgün Çiçek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 2016. 3D U-Net: Learning dense volumetric segmentation from sparse annotation. (2016). arXiv:cs.CV/1606.06650.

Received December 2020; revised October 2021; accepted November 2021