IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 1 May 2023, accepted 30 May 2023, date of publication 8 June 2023, date of current version 14 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3284388

RESEARCH ARTICLE

Impact of Mixed Precision Techniques on Training and Inference Efficiency of Deep Neural Networks

MARION DÖRRICH, MINGCHENG FAN, AND ANDREAS M. KIST, (Member, IEEE)
Department of Artificial Intelligence in Biomedical Engineering, Friedrich-Alexander-Universiät Erlangen-Nürnberg, 91052 Erlangen, Germany

Corresponding author: Andreas M. Kist (andreas.kist@fau.de)

This work was supported by Kompetenznetzwerk für Technisch-Wissenschaftliches Hoch- und Höchstleistungsrechnen in Bayern (KONWIHR).

ABSTRACT In the deep learning community, increasingly large models are being developed, leading to rapidly growing computational costs and energy costs. Recently, a new trend has been arising, advocating that researchers should also report the energy efficiency besides their model's performance in their papers. Previous research has shown that reduced precision can be helpful to improve energy efficiency. Based on this finding, we propose a simple practice to effectively improve the energy efficiency of training and inference, i.e., training the model with mixed precision and deploying it on Edge TPUs. We evaluated its effectiveness by comparing the speed-up of a state-of-the-art semantic segmentation architecture with respect to different typical usage scenarios, including using different devices, deep learning frameworks, model sizes, and batch sizes. Our results show that enabled mixed precision can gain up to a 1.9× speedup compared to the most common and default float32 data type on GPUs. Deploying the models on Edge TPU further boosted the inference by a factor of 6. Our approach allows researchers to accelerate their training and inference procedures without jeopardizing the model's accuracy, meanwhile reducing energy consumption and electricity cost easily without changing their model architecture or retraining. Furthermore, our approach is helpful in reducing the carbon footprint used to train and deploy the neural network and thus has a positive effect on environmental resources.

INDEX TERMS Deep learning, green AI, energy efficiency, mixed precision training, quantization, edge TPU.

I. INTRODUCTION

In AI research, there is a trend that significant computational resources are required to achieve the next state-of-the-art performance. As a result, a large amount of energy is consumed, which has two major effects: (i) economically - high energy costs for researchers and institutions and (ii) ecologically - resulting in a large carbon footprint.

Recently, the Green AI movement has arisen, trying to arouse the attention of the AI community to the energy efficiency of deep neural networks [1]. Different works have proposed suggestions from different perspectives on improving the energy efficiency of deep neural networks. Patterson et al. suggested reducing energy consumption by carefully choosing the hardware and data center [2]. Georgiou et al. investigated the difference between using two of the most popular DL frameworks, PyTorch and Tensorflow, regarding the energy efficiency during training and inference of different models [3]. Xu et al. focused on the model architecture's impact on energy consumption during training [4]. One of the perspectives advocates using data types of lower precision during the model inference phase, i.e., quantization, to reduce energy consumption while maintaining the model accuracy because, firstly, all operations of a quantized model are carried out in a lower precision, which can reduce computation energy and data access energy during inference [5], [6], [7], [8]. Secondly, quantized models usually run on edge-class devices like Edge TPU, which naturally consume much less energy than high-performance devices. We adopt the advocation of running

The associate editor coordinating the review of this manuscript and approving it for publication was Oussama Habachi.

quantized models on edge-class devices and also recommend using lower precision during the training phase to save more energy.

So far, we can conclude a simple and straightforward approach to improve the energy efficiency of model development, i.e., introducing lower precision into training and inference and deploying models on edge-class devices. However, despite the fact that the concept is simple, the execution is not. First, introducing lower precision into training is usually complex and requires many modifications to the current training code. For researchers with available training codes, it hinders them from using lower yet sufficient precision. This increases the cost of switching to Green AI to some degree. Fortunately, regarding the idea of introducing lower precision into training, several mixed precision training approaches have been proposed [9], [10]. These approaches were well implemented and documented for the popular deep learning frameworks PyTorch and TensorFlow, and the currently popular devices (e.g., NVIDIA RTX 3090 GPU) also support them. Mixed precision training [9] recommended using lower precision (for example, IEEE 16 bits float point number) for the computation while using single precision (IEEE 32 bits float point number) for weights-updating. Mixed precision training can gain speed up and reduce energy consumption while maintaining a reasonable accuracy level. However, we notice that the energy efficiency of mixed precision training has rarely been analyzed with respect to different usage scenarios. Second, as one of the famous edge-class devices, Edge TPU was proven to have a satisfactory inference speed while maintaining reasonable accuracy [11], [12], [13], yet to our best knowledge, the comparison of the inference speed between Edge TPU and HPC GPUs has not been extensively investigated.

Based on these knowledge gaps, we devoted ourselves to verifying the effectiveness of mixed precision training API with respect to different usage scenarios (including tensor processing hardware, deep learning frameworks, batch size, and model size) and comparing the speed-up effect of Edge TPU with high-performance computing devices when the model size is varying. We calculated the quotient of the single-sample-processing time of mixed precision training and float32 training, i.e., speed-up, to refer to the effectiveness of mixed precision training. We notice that sophisticated methods exist to estimate energy consumption during model development in previous works [2], [4], [5], [6], [14], [15]. For examples, Panda et al. [5] designed a 2-D systolic array accelerator to calculate the energy efficiency. Vasquez et al. [6] designed a precision-scalable process in-memory hardware platform to calculate the energy consumption and efficiency improvement. However, for simplicity, we used speed-up to represent the acceleration effect and energy efficiency of mixed precision training because it is of more concern to the researchers that mainly focus on neural network development while lack experience in hardware engineering. Similarly, to benefit as many deep learning researchers as possible, we used a model based on U-Net [16]. Because it is widely used for image segmentation tasks in the Deep Learning realm. Specifically, our work has the following contributions:

- We verified that mixed precision training has a speed-up effect in almost all the usage scenarios we investigated. Using TensorFlow on Nvidia RTX 3090 GPU gained the largest speed-up ($1.9\times$).
- We verified that mixed precision training does not impact model accuracy, despite the devices, frameworks, model size, and batch size.
- We found that a larger model gains a larger speed-up effect.
- We verified that Edge TPU merely slightly impacts model accuracy (except for exceptionally large models).
- We found that compared with HPC GPUs, Edge TPU has a comparable or better acceleration for smaller models.

In summary, our work helps researchers who consider switching to Green AI to take the first yet effective step. Besides, our work is also helpful for AI developers who seek faster and more cost-effective training and inference.

II. METHODS

A. DATASET

We used the publicly available biomedical dataset BAGLS (Benchmark for Automatic Glottis Segmentation) [17] as a generic, binary, and medical-oriented semantic segmentation task. The BAGLS dataset contains images acquired from laryngeal high-speed videoendoscopy [18]. For each image, a segmentation mask of the glottis serves as the ground truth for semantic segmentation. We randomly took 4096 samples of this dataset and cropped them to a resolution of 224×224 pixels. The images were converted to grayscale, and the pixel values were normalized to a range between -1 and 1. In experiments on the Edge TPU, the pixel values were normalized to a range between 0 and 1.

B. NETWORK ARCHITECTURE AND HYPER-PARAMETERS

We trained encoder-decoder deep neural networks for semantic segmentation of the glottal area, as shown in Figure 1. In our case, we relied on a modified U-Net architecture [16], a fully convolutional neural network consisting of an encoder for feature extraction and a decoder for localization. Its hallmark is the existence of skip connections between the encoder and decoder to enhance the segmentation quality.

The initial number of filters f in Figure 1 was varied to account for different model sizes and capacities. As shown in Figure 1a, the number of filters increases in each step of the encoder and decreases in the decoder correspondingly. Each model was trained using Dice loss [19] and Adam optimizer [20] with a learning rate of 0.0005. Among all model hyper-parameters, we were especially interested in the model size (Model size refers to the initial number of filters of the U-Net.) and the batch size, and thus we adjusted them in our experiments to test if they have influence on the effectiveness of the mixed precision training API. We set the batch size

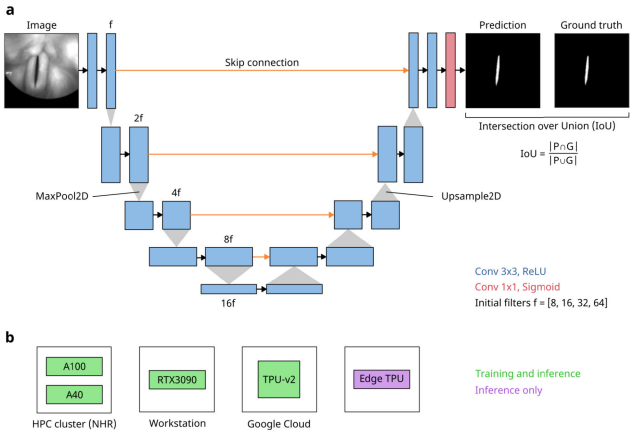

FIGURE 1. Segmentation network and investigated devices. (a) U-Net architecture with four encoding and four decoding blocks. Each box corresponds to a convolutional layer. The number of layers was fixed but the number of initial filters *f* was varied. The Intersection over Union (IoU) was computed between prediction and ground truth to assess the segmentation quality. (b) Infrastructure overview containing different Deep Learning hardware accelerators.

to 128, which is the maximum possible batch size in most settings. However, on RTX3090, we had to lower the batch size to 64 for the model with 32 initial filters and lower the batch size to 32 for the model with 64 filters due to the limited CUDA memory. On Cloud TPU, we set the global batch size to 128, which means that the local batch size per TPU core was 16.

C. DEVICES AND FRAMEWORKS

The same U-Net architecture was implemented using the frameworks PyTorch (version 1.12.1) and TensorFlow (version 2.9.2) [21] with Keras API. We trained all models on several graphics processing units (GPUs): Nvidia RTX3090, A100, and A40, all with Tensor Cores supporting mixed precision training (float16 and float32). The GPUs A100 and A40 were accessed via a High-Performance Computing (HPC) cluster of the Erlangen National High-Performance Computing Center (NHR). We used different versions of CUDA to ensure compatibility with each GPU and framework. We used CUDA 11.2 for TensorFlow, but CUDA 11.6 (on HPC) and CUDA 11.3 (on RTX3090) for PyTorch.

Tensor processing units (TPUs) are application-specific integrated circuits (ASICs) that Google designed to accelerate Deep Learning tasks [22]. We used a Cloud TPU v2 with 8 Cores, which was accessed through Google Cloud Platform. We selected the TPU software versions "tpu-vm-tf-2.10.0" and "tpu-vm-pt-1.10". On the TPU, we additionally installed the framework PyTorch Lightning (version 1.5.10), a similar yet high-level API compared to PyTorch.

The Edge TPU is an edge device built by Coral, designed for accelerating IoT-based solutions. In contrast to GPUs and TPUs, the Edge TPU can only be used for inference, and it only accepts batch size of 1 during inference. For our experiments, we used the Coral USB accelerator and the PyCoral API. Since it is an integer-only device, it only supports 8-bit quantized and compiled models in TFlite format.

D. MIXED PRECISION TRAINING API

Both TensorFlow and PyTorch use single precision, i.e., float32, by default and support automatic mixed precision, i.e., a mix of float32 and float16, for training and inference. The implementation of both frameworks are based on the key ideas of mixed-precision training proposed by Micikevicius et al. [9], i.e., carrying out computation in float16, updating weights in float32 and applying loss scaling to prevent gradient underflow.

Using Keras, automatic mixed precision can be activated by setting the global policy to "mixed_float16" for all layers. This data type policy enables each layer to use float32 for creating and storing the weights and float16 for computations. Note that not all computations are carried out in float16; for example, weights updating is carried out in float32 to ensure that the model's quality is not affected. Dynamic loss scaling is automatically performed to avoid the gradient of activations underflow to zero. As recommended by the TensorFlow mixed-precision API documentation, we set the network's final activation data type to float32 to prevent numeric issues.

Using PyTorch, automatic mixed precision can be utilized by installing the Automatic Mixed Precision (AMP) API of Nvidia Apex. It provides users with different implementations of mixed precision training. We chose the"O1" level because it is similar to the implementation in Keras and also recommended by Nvidia for better numeric stability. In the "O1" level, "apex.amp" patches PyTorch functions that each PyTorch module calls internally according to a white/blacklist, such that these functions inputs (including activations and weights) are cast inside, then these functions perform the computation and generate output with the desired data types. About the white/blacklist of PyTorch functions, for example, if an operation does not require much precision and can acquire significant speedup when performed in lower precision, such as matrix multiplications, then it will be carried out in float16. Furthermore, dynamic loss scaling is automatically applied to avoid gradient underflow.

Google's Cloud TPUs use the compiler Accelerated Linear Algebra (XLA). This compiler automatically uses bfloat16 for some operations under the hood. The bfloat16 format requires only half of the memory space but preserves the same dynamic range as the float32 format. Moreover, it was shown that bfloat16 training of neural networks can achieve the same state-of-the-art results as the standard 32-bit format [10]. Using TensorFlow, a mix of float32 and bfloat16 can be used but is not likely to lead to much higher performance. PyTorch Lightning also supports this mixed precision policy.

Table 1 summarizes the investigated combinations of devices, data types, and frameworks. In addition to single and mixed precision training, we also explored using pure half-precision on GPUs to see if we can gain more speedup.

TABLE 1. Overview of investigated devices, data types, and frameworks.

Devices	Data types		Frameworks	
GPU	fp32	fp32+fp16	TF	PyTorch
TPU	fp32	fp32+fp16	TF	PyTorch
EdgeTPU	uint8		TFLite	

E. MEASURING PERFORMANCE

The segmentation quality was assessed by computing the Intersection over Union (IoU) score using k-fold cross-validation where k = 4. The reason we used k-fold cross-validation is to validate the effect of each experimental setting in a more statistically reliable way. For each experimental setting, We ran it on 4 folds, recorded the results of 4 folds, calculated the mean of them, and then adopted the mean as the final result. Since models with few filters usually need more training epochs, we used the early stopping technique to ensure that every model converged similarly. Specifically, this criterion suggests stopping the training when the validation IoU score is no longer increasing for ten epochs.

To determine the training speed, we first measured the mean epoch duration, which was also averaged over the folds. (However, we omitted the first training epoch because it was usually much slower than the following epochs due to program initialization.) We calculated the single-sample processing time from the average epoch duration. To determine the inference speed, we used the same metric, i.e., single-sample processing time. The measuring procedure was similar. We first fed batches of images to each trained model repeatedly and measured the mean duration of processing one batch. Then we calculated the single-sample processing time by dividing the mean duration of one batch by the batch size. We also calculated the speedup gained from mixed precision compared to the baseline, i.e., using single precision, with respect to each usage scenario (A usage scenario is characterized by a combination of choices of device, framework, model size and batch size). The speedup was defined as the ratio between the throughputs, i.e., the number of samples processed per second.

All samples were preloaded in RAM before training. Additionally, with the help of the TensorBoard profiler, we ensured that our input pipeline was not a performance bottleneck.

F. QUANTIZATION AND DEPLOYMENT TO EDGE TPU

1) DEPLOYING MODELS ON EDGE TPU

To deploy our models on Edge TPU, the trained TensorFlow models were quantized and converted to the TensorFlow Lite format. Next, the models were compiled and all operations were mapped to Edge TPU.

The PyTorch models need to be converted to TensorFlow models before quantization and conversion. Thus, we restricted ourselves to TensorFlow models only.

We successfully converted and compiled float32 models with 8, 16, and 32 initial filters. For models trained with the ''mixed_float16'' precision policy, we first converted them to float32 models by replacing the precision policy of each layer with ''float32'' because the Tensorflow Lite converter does not support quantizing and converting the models trained with ''mixed_float16'' precision policy. During compilation, some ''Upsample2D'' operations of the models with 32 initial filters were not be mapped to Edge TPU successfuly and thus remained on CPU due to the Edge TPU memory constraint. To avoid the inference latency caused by this, we replaced part of the ''Upsample2D'' operations with a custom upsampling operation, proposed by Kist et al. [13], to ensure all operations can be mapped to Edge TPU. The entire model with 64 initial filters was mapped to Edge TPU successfully, but its predictions were compromised. Further studies about the reason of this problem are needed.

2) PRINCIPLES OF QUANTIZATION

In the context of Edge TPU, quantization follows the ''Post-training integer quantization'' scheme provided by TensorFlow Lite to represent the model's weights, activations, input, and output using 8-bit fixed point numbers. This quantization scheme is essentially an affine mapping, following the equation 1 [23]:

$$q = \text{round}(f/s) + z, \tag{1}$$

where q represents the parameters in float32 precision. s is a scaling factor in float32 precision, it maps the real values in the range $[a, b]$ to an integer in the range of $[-128, 127]$ thus it is determined by $s = (b - a)/(2^8 - 1)$. And z is the integer representation of the floating-point zero [24]. Note that quantization is not rounding a floating point number to the nearest integer. For example, for floating point numbers of the range [0.0, 1.0], the floating point zero should be mapped to the integer -128 instead of the integer 0.

With the quantization scheme above, a 32-bit floating-point number can be represented by an 8-bit fixed-point number without much accuracy loss. Furthermore, floating-point operations like convolution can be carried out in integer arithmetic, which can save a lot of memory and computation costs. Quantization may affect the accuracy. Generally, a small drop in accuracy can be observed, and in rare cases, models can gain some accuracy after the quantization.

III. RESULTS

We implemented the same U-Net architecture using PyTorch and TensorFlow and carried out training and inference on different devices including Nvidia RTX3090, A100, A40 GPUs and Google Cloud TPUs. On each device, we varied model size, batch size and precision policy to investigate the influence that these factors have on the effectiveness of the mixed precision API. Besides, we quantized the float32 models using ''post-training integer quantization'' and then deployed them on Edge TPU. At last, we compared the inference speed of Edge TPU and HPC GPUs.

A. MIXED PRECISION TRAINING APIs HAVE BETTER USABILITY THAN MANUALLY CONFIGURING THE PRECISION

To demonstrate the usability of the mixed precision APIs that we investigated from another perspective, we explored pure half-precision training, which is not recommended by the API yet of particular interest to many researchers; for example, in Keras, we manually set each layer's data type policy to float16 to make model weights being created, stored, and updated in float16. We also tried another approach, i.e., setting the default float type of TensorFlow as float16 to enforce all computations to be performed in 16-bit. In both cases, dynamic loss scaling must be handled manually, introducing an additional hyper-parameter to tune. For PyTorch, we used "O3" level to enable pure float16 training. Under "O3" level, no loss scaling is performed, model weights are created and stored in float16, and all computations, including updating the weights, are performed in float16. However, all these trials resulted in a non-trivial drop in the IoU score compared to full precision training. Micikevicius et al. [9], [25] analyzed the reason. They claimed the reason has three folds. First, during the weights updating procedure, the magnitude of the updates is too tiny compared to the weights. With float16 arithmetic of performing add-operation, this can lead to the updates underflow to zero, which hinders the network from being fully updated. Second, before updating, the error tensor itself can fall out of the representative range of float16 and thus underflow to zero. Third, during the loss-calculating procedure, the accumulation operation is performed in float16, likely leading to loss overflow and thus sabotaging the whole training. It is worth pointing out that the solutions to the unstable training problem were implemented into the mixed precision API.

B. MIXED PRECISION SPEEDS UP TRAINING AND INFERENCE

We trained the traditional U-Net architecture with 64 initial filters using single and mixed precision to investigate potential differences in the performance. Figure 2a shows the single-sample processing time for the training phase using float32 on different devices with different deep learning frameworks. It can be seen as a baseline. It also describes the computing power of the different devices and the compatibility between frameworks and devices. As shown in Figure 2a, we observed in our experiments that the training on TPU was more than twice as fast as on the GPU A100 when using TensorFlow, which is within our expectations because Cloud TPU is more specialized for tensor processing than GPUs. Figure 2b shows how the effectiveness of mixed precision API varies on different devices with different frameworks. We observed that on all GPUs, the single-sample processing time of PyTorch was shorter compared to TensorFlow. However, this is not necessarily implying a superior performance of the mixed

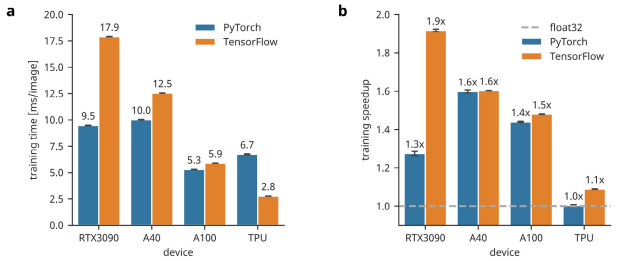

FIGURE 2. Training speed and speedup of U-Net with 64 filters on different devices. The value of each bar is the mean value of 4 folds. We also measured the standard deviation of each value over the 4 folds, and drew it as error bar on top of each bar. (a) Average time per image during training using float32 precision. (b) Speedup gained from mixed precision training compared to the baseline, i.e. float32.

precision API of PyTorch than TensorFlow. This could also be caused by different CUDA versions installed with each framework. As shown in Figure 2b, we observed in our experiments that mixed precision drastically reduced the training time on all investigated GPUs to varying extents. For example, a 1.9x speedup was reached on RTX3090 using TensorFlow. This is most likely due to the fact that the three investigated GPUs have Tensor Cores that enable fast float16 matrix multiplications. As shown in Figure 2b, we did not observe any significant speed-up effect when we applied mixed precision training manually on Cloud TPUs. We infer the reason might be that the XLA compiler performs many computations using bfloat16 under the hood.

C. MIXED PRECISION TRAINING DID NOT REDUCE SEGMENTATION QUALITY

We studied whether mixed precision training would affect the quality of predictions by computing the IoU score using k-fold cross-validation. Despite the varying model size, figure 3 shows that mixed precision did not reduce our models' segmentation quality. This finding confirms that mixed precision training does not lead to a loss of model accuracy [9]. Additionally, given the same early stopping technique and the same stopping criterion applied, we observed in our experiments that all models reached approximately the same IoU score, as shown in Figure 3. We found that, for example, a model with 8 filters reached about the same IoU score as larger models. However, a small model usually takes more epochs to converge.

D. SPEED DEPENDS ON MODEL SIZE AND BATCH SIZE

Despite the frameworks and precision, we found that the training and inference speed (i.e., single-sample processing time) decreased when the model became larger, as demonstrated in Figure 4.

However, larger models gain more speed-up effect from mixed precision training, despite devices and frameworks, as Figure 5 shows. In contrast, mixed precision does not necessarily result in a speedup for very small models,

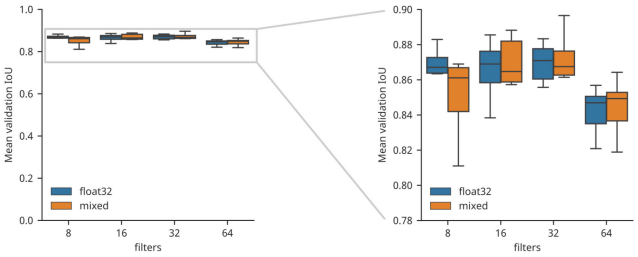

FIGURE 3. Distribution of mean validation IoU score from 4-fold cross-validation, using TensorFlow and PyTorch on different devices. The IoU score was approximately the same using single and mixed precision training.

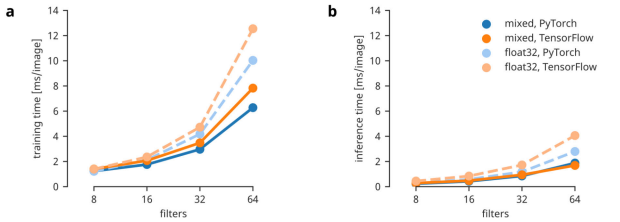

FIGURE 4. Comparison of training and inference time using single and mixed precision, as well as different frameworks on the GPU A40. (a) Mean training time over 4 folds with a fixed batch size of 128. (b) Mean inference time over 4 folds with a fixed batch size of 128.

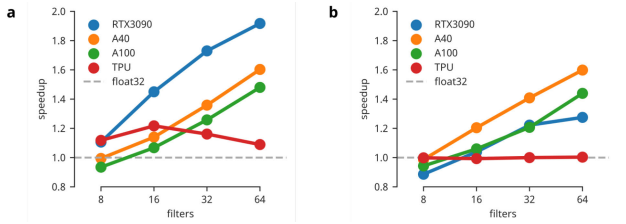

FIGURE 5. Training speedup (averaged over 4 folds) gained from mixed precision compared to the baseline, i.e. single precision training. (a) Using TensorFlow. (b) Using PyTorch.

such as U-Net with 8 initial filters. Note that when we increased the model size, we did not observe an obvious trend for the speed-up effect on Cloud TPUs, as demonstrated in Figure 2.

Furthermore, to investigate the influence that batch size has on the effectiveness of the mixed precision training API, we varied the batch size during inference while fixing the model size to 64. We found that the positive speed-up effect could be guaranteed only if the batch size exceeds eight, as Figure 6 demonstrates. As shown in Figure 6, we also observed in our experiments that the Tensorflow inference speed is slower than the PyTorch inference speed when the batch size is small.

E. EDGE TPU INFERENCE TIME AND IoU COMPARISON

To compare the inference speed between Edge TPU and HPC GPUs, we prepared the models trained in float32 precision and deployed them on different devices. Due to the limited memory on Edge TPU, the only acceptable batch size is one.

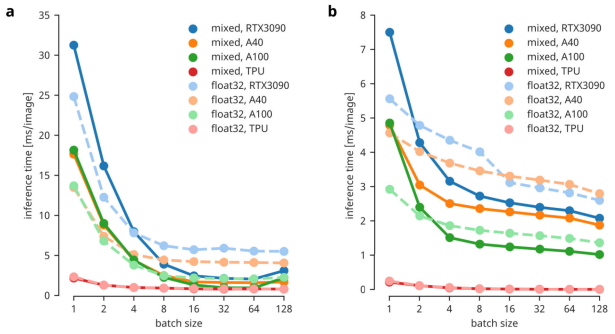

FIGURE 6. Mean inference time per image on different devices using single and mixed precision on U-Net of 64 initial filters. For the TPU, the local batch size per TPU core is displayed. (a) Mean inference time per image using TensorFlow. (b) Mean inference time per image using PyTorch.

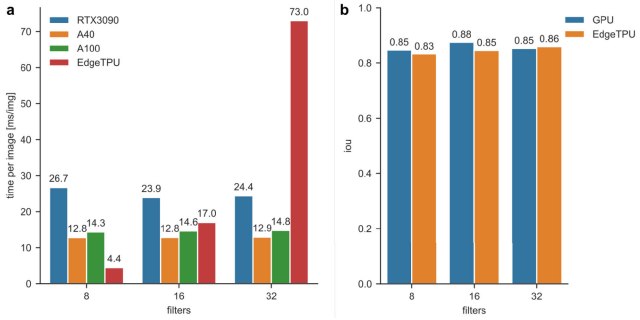

FIGURE 7. (a) Comparison of the mean inference time per image of different models that were trained in single precision with different initial filters and a batch size of one using TensorFlow framework. (b) Comparison of the mean segmentation IoU between Edge TPU and GPUs.

Thus we applied this batch size to all devices during the inference. We observed that for a small model, such as U-Net with 8 initial filters, Edge TPU outperformed all HPC GPUs regarding the inference speed. However, Edge TPU lost its advantage when the model size increased, as Figure 7a shows. We infer it is due to the limited memory and power design of Edge TPU. The IoU in Figure 7b was calculated as follows: for IoU of Edge TPU, it was averaged over 1024 validation images; for the IoU score of GPUs, it was first averaged over 1024 images, then averaged over different devices, i.e., A40, A100, RTX3090. Compared with inference on GPUs, only a slight drop was observed when we ran quantized models on Edge TPU. The results prove that Edge TPU can boost models' inference speed and maintain their performance simultaneously. The comparison does not involve models trained with "mixed_float16" precision policy because after we quantized these models, we did not observe any difference between these models and the quantized models trained with float32 regarding inference speed and IoU. We infer the reason is that the models trained with mixed precision policy store their weights in float32 and thus have the same ability (compared to the models trained with float32) to represent what the networks learn from the data.

IV. DISCUSSION

A. TAKEAWAYS

1) SPEEDING UP TRAINING ON GPUs

Our experiments show that the mixed precision training API provided by TensorFlow and Nvidia is a helpful tool to speed up training procedures on GPUs with little effort and without jeopardizing model performance. We show that researchers should expect a variable speed-up effect when using the mixed precision training API with different devices, frameworks, model sizes, or batch sizes. Especially we observed that the difference of the runtime does exist between using PyTorch and Tensorflow, just as Georgiou et al. reported in [3]. Additionally, with a larger model and larger batch size, researchers can expect a more significant speed-up effect from mixed precision training.

2) USING TPUs FOR COMPLEX NEURAL NETWORKS

Our findings suggest that for training a model with many parameters on a large dataset, a TPU should be considered as it is much faster than modern GPUs. A drawback is that TPUs are not as accessible. Google Colab offers free use of TPUs. However, their availability is often limited. Another option is accessing TPUs via the Google Cloud Platform.

3) FAST AND LOW-COST INFERENCE ON EDGE TPU

The inference speed of our smallest encoder-decoder network is much faster on Edge TPU compared to GPUs, although it requires much less energy and costs. Therefore, Edge TPUs should be considered as an alternative to GPUs in terms of inference. In addition, Edge TPU can be used for efficient AI-based applications on portable devices. It is possible to deploy a mixed precision-trained model to Edge TPU by converting it to a single precision model at first and then quantizing it. However, limitations of supported operations and model sizes must be considered.

B. LIMITATIONS

We used "speed-up" as an indirect metric to estimate the energy efficiency of the mixed precision training API. We notice that in previous works, researchers either proposed their own method [2], [5], [6] or used widely accepted methods [14], [15] of estimating energy consumption. This may limit the contribution of our study, preventing it from being used to accurately quantify the energy efficiency of the mixed precision training API. However, we believe our work is still helpful for researchers seeking a simple but effective way to make their network development more energy-efficient and economical.

To align the experimental environment, we installed the same version of PyTorch for different devices (same for TensorFlow). However, we did not enforce the same CUDA version between PyTorch and TensorFlow. This may affect the validity of the framework comparison and the effectiveness of the mixed precision training API may be also affected. Georgiou et al. [3] proposed using a Docker container to create a consistent environment in which all dependencies of TensorFlow and PyTorch are the same. This is especially convenient for testing models on different devices.

V. CONCLUSION

In our work, we have considered as many factors as possible in relation to the effectiveness of the mixed precision API to showcase to researchers the efficient ways of using it. Besides, we showcase the great advantage of using Edge TPU for model inference by comparing its inference speed with HPC GPUs. To our best knowledge, this area has rarely been investigated. We highlight a simple but helpful practice of using mixed-precision training API during training and deploying models on Edge TPU for inference, as this can bring about up to 1.9× speed-up for training and up to 6.0× speed-up for inference with little effort. With the ultra-low power consumption and cost of an Edge TPU (compared to the HPC GPUs), the out-of-the-box approach can help researchers who lack experience with sophisticated energy-efficient methods but want to save energy, reduce research funding for power costs and equipment costs, or accelerate their model development to achieve their goals.

ACKNOWLEDGMENT

The authors would like to thank the HPC Erlangen for their support during the project.

(Marion Dörrich and Mingcheng Fan contributed equally to this work.)

REFERENCES

[1] R. Schwartz, J. Dodge, N. Smith, and O. Etzioni, "Green AI," *Commun. ACM*, vol. 63, no. 12, pp. 54–63, Dec. 2020.

[2] D. Patterson, J. Gonzalez, Q. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. So, M. Texier, and J. Dean, "Carbon emissions and large neural network training," 2021, *arXiv:2104.10350*.

[3] S. Georgiou, M. Kechagia, T. Sharma, F. Sarro, and Y. Zou, "Green AI: Do deep learning frameworks have different costs?" in *Proc. IEEE/ACM 44th Int. Conf. Softw. Eng. (ICSE)*, May 2022, pp. 1082–1094.

[4] Y. Xu, S. Martínez-Fernández, M. Martinez, and X. Franch, "Energy efficiency of training neural network architectures: An empirical study," in *Proc. 56th Hawaii Int. Conf. Syst. Sci.*, 2023, pp. 781–790.

[5] P. Panda, "QUANOS: Adversarial noise sensitivity driven hybrid quantization of neural networks," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Design*, Aug. 2020, pp. 187–192.

[6] K. Vasquez, Y. Venkatesha, A. Bhattacharjee, A. Moitra, and P. Panda, "Activation density based mixed-precision quantization for energy efficient neural networks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 1360–1365.

[7] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 4820–4828.

[8] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.

[9] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," 2017, *arXiv:1710.03740*.

[10] D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, J. Yang, J. Park, A. Heinecke, E. Georganas, S. Srinivasan, A. Kundu, M. Smelyanskiy, B. Kaul, and P. Dubey, "A study of BFLOAT16 for deep learning training," 2019, *arXiv:1905.12322*.

[11] P. Amanatidis, G. Iosifidis, and D. Karampatzakis, "Comparative evaluation of machine learning inference machines on edge-class devices," in *Proc. 25th Pan-Hellenic Conf. Informat.*, Nov. 2021, pp. 102–106.
[12] J. Civit-Masot, F. Luna-Perejón, J. M. R. Corral, M. Domínguez-Morales, A. Morgado-Estévez, and A. Civit, "A study on the use of edge TPUs for eye fundus image segmentation," *Eng. Appl. Artif. Intell.*, vol. 104, Sep. 2021, Art. no. 104384.
[13] A. M. Kist and M. Döllinger, "Efficient biomedical image segmentation on EdgeTPUs at point of care," *IEEE Access*, vol. 8, pp. 139356–139366, 2020.
[14] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1135–1143.
[15] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 3645–3650.
[16] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervent.*, Munich, Germany: Springer, Oct. 2015, pp. 234–241.
[17] P. Gómez, A. M. Kist, P. Schlegel, D. A. Berry, D. K. Chhetri, S. Dürr, M. Echternach, A. M. Johnson, S. Kniesburges, M. Kunduk, Y. Maryn, A. Schützenberger, M. Verguts, and M. Döllinger, "BAGLS, a multihospital benchmark for automatic glottis segmentation," *Sci. Data*, vol. 7, no. 1, p. 186, Jun. 2020.
[18] A. M. Kist, S. Dürr, A. Schützenberger, and M. Döllinger, "OpenHSV: An open platform for laryngeal high-speed videoendoscopy," *Sci. Rep.*, vol. 11, no. 1, Jul. 2021, Art. no. 13760.
[19] F. Milletari, N. Navab, and S. Ahmadi, "V-Net: Fully convolutional neural networks for volumetric medical image segmentation," in *Proc. 4th Int. Conf. 3D Vis. (3DV)*, Oct. 2016, pp. 565–571.
[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[21] M. Abadi, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2016, *arXiv:1603.04467*.
[22] N. P. Jouppi, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 1–12.
[23] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018, *arXiv:1806.08342*.
[24] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," 2021, *arXiv:2103.13630*.
[25] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis.* Amsterdam, The Netherlands: Springer, Oct. 2016, pp. 21–37.

MARION DÖRRICH received the B.Eng. degree in medical engineering from the Nuremberg Institute of Technology Georg Simon Ohm, Germany, in 2019, and the M.Sc. degree in medical engineering from Friedrich-Alexander University Erlangen-Nuremberg (FAU), Germany, in 2022. She is currently a Research Associate with FAU. Her research interests include artificial intelligence, deep learning, image processing, and medical engineering.

MINGCHENG FAN received the B.Eng. degree in mechatronics from the Niederrhein University of the Applied Science, Germany, in 2019. He is currently pursuing the degree in medical engineering with Friedrich-Alexander University Erlangen-Nuremberg (FAU), Germany. He is also a Student Assistant with FAU. His research interests include artificial intelligence, deep learning, image processing, and medical engineering.

ANDREAS M. KIST (Member, IEEE) received the B.Sc. and M.Sc. degrees in molecular medicine from Friedrich-Alexander-University Erlangen-Nnberg (FAU), Germany, and the Ph.D. degree from the Max-Planck-Institute for Neurobiology, Martinsried, Germany. After his postdoctoral research with University Hospital Erlangen, he was appointed as an Assistant Professor with FAU, in 2021, leading his own research group focusing on biomedical signal processing. He was awarded the XION Innovation Prize and the Add-on Fellowship of the Joachim-Herz-Foundation.