Performance of Deep Learning Models with Different Precision

Haoyu Wang(hw3256)*, Yuchen Kou(yk2493)*, Zonghui Hu(zh2393)*
New York University Tandon School of Engineering

Abstract

Training of large-scale deep neural networks is often constrained by the available computational resources. The efficiency of resource utilization hence has become a trending topic. To better understand the impact of data precision on the training of deep learning models, we implement FP16, FP32, and AMP on the training of SSD300 and ResNet-18 for object detection and image classification tasks respectively. We use Pascal Visual Object Classes (VOC) data from the years 2007 and 2012 and CIFAR-10 as the dataset. Different than previous research, our results indicate that FP32 has a higher accuracy but takes the longest training time.

Introduction

As the application scenario of machine learning becomes more complex, neural networks have grown in size over the past few years. During the training, large amounts of data read/write and computing have high requirements on hardware resources. Larger models typically require more computational and memory resources to train. The performance of a trained model is still limited by arithmetic bandwidth, memory bandwidth, and latency. Therefore, the efficiency of resource utilization in deep learning has become a trending topic. Data precision, in relation to representation and arithmetic performance, is a critical element that can affect performance.

Generally, higher precision should bring higher accuracy but at the same time require more memory and take more time to train. Low precision, relatively, should enable us to train larger networks and train faster with less accuracy. In this study, we will do several experiments using FP32 (Single-precision), FP16 (Half-precision), and AMP (Automatic mixed precision), to explore the impacts of data precision on different models and computer vision tasks.

We implement FP16, FP32, and AMP on two models, SSD300 with VGG16 backbone and ResNet-18. To test the impact of different precision, we train the models for object detection and image classification tasks respectively. The dataset we use for the object detection task is Pascal Visual Object Classes (VOC) data from the years 2007 and 2012. For the image classification task, we use CIFAR-10 as it is one of the most widely used datasets for machine learning research. We then compare the training time, test loss, test accuracy and mean average precision (MAP) for the object detection task in each experiment. Our test results can be found in this GitHub repository [1].

The organization of this paper is as follows: Section II describes related works. Section III explains the basic background for our research. Section IV illustrates the experiment result. The last section gives a conclusion.

Related Work

Determining the precision of the data representation and the compute units is a critical design choice in the hardware (analog or digital) implementation of artificial neural networks. FP32 nowadays is the default size of float for all calculations and is in use in most Deep Learning models. Even standard Programming Languages supported FP32 as the default float datatype. But before everyone agrees with this default precision, numerous related works investigate neural network training using different number representations and provide valuable insights into the behavior of the limited precision training of neural networks.

Iwata et al. implement the back-propagation algorithm using 24-bit floating-point processing units and propose that 24-bit floating-point operation provides sufficient accuracy(Iwata et al. 1989). Holt et al. analyze the finite precision error within back-propagation and iterative learning(Holt and Hwang 1991). Empirical evaluation results indicate that 8-bit to 16-bit is sufficient in back-propagation learning. Hoehfeld et al. present an empirical study of the effects of limited numerical precision in the neuron network employing cascade-correlation learning algorithm. Experiment results illustrate that limited precision in sigmoid computation has little impact on training(Hoehfeld and Fahlman 1992). Limited precision in weights however may cause the failure of learning. They then introduce probabilistic rounding, a procedure that can reduce precision requirements in gradient-based learning techniques to 8 bits.

Networks that early papers considered are often limited to few hidden layers and hidden units, so to some extent,

*These authors contributed equally.

[1] https://github.com/HW0327/Performance-of-Deep-Learning-Models-with-Different-Precisions

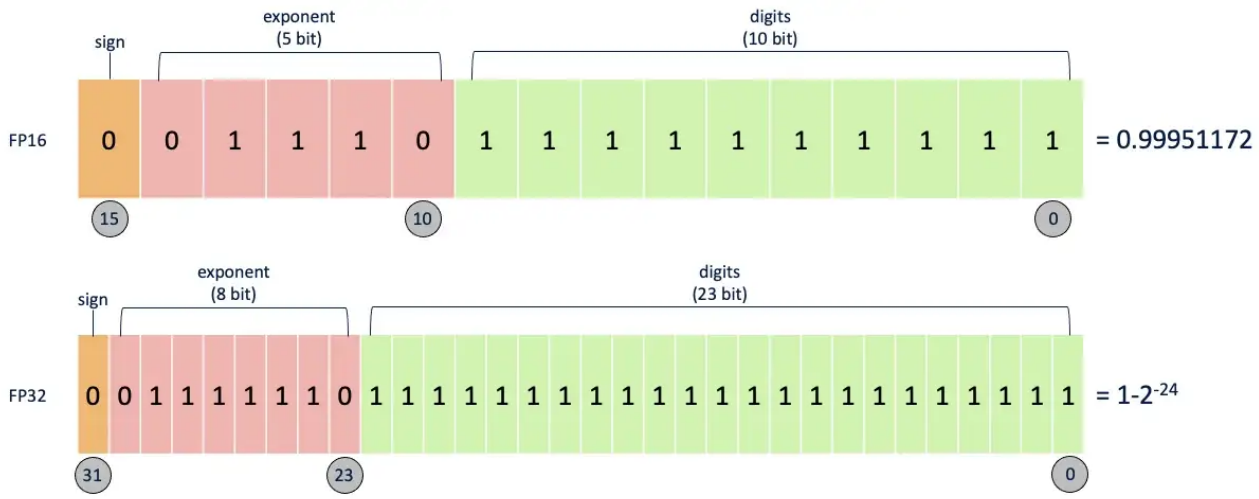

Figure 1: Comparison of the format for FP16 (top) and FP32 (bottom) floating-point numbers. The number shown, for illustrative purposes, is the largest number less than one that can be represented by each format.(Davis 2021)
.

few of them think that machine learning has a great need for high precision. In recent works, though, the need for high precision is highly debated. Chen et al. claim that 32-bit fixed-point representation is necessary to achieve convergence while training a convolutional neural network on the MNIST dataset(Chen et al. 2014). While Gupta et al. refute that Using stochastic rounding during fixed-point computations can reduce the precision requirement to 16-bit fixed-point numbers (Gupta et al. 2015).

Building on these insights, we present the performance of DNNs trained by different precision to clearly illustrate the impact of precision on state-of-the-art neural networks.

Background

FP32

The standard FP32 format is supported by almost any modern Processing Unit, and normally FP32 numbers are referred to as single-precision floating points. In neural networks, this is the default format to represent most network weights, biases, and activation, in short, most parameters. It can also be used for any scientific computations which don't require more than 6 significant decimal digits. FP32 precision format bits are divided as follows: 1 bit for the sign of the number, 8 bits for the exponent, or the magnitude, and 23 bits for the mantissa or the fraction, as shown in Figure 1.

Every format has a range of numbers that can be represented with, and with FP32, one can represent numbers of the magnitude of order 10^{38}, with 7-9 significant decimal digits.

FP16

In contrast to FP32, and as the number 16 suggests, a number represented by FP16 format is called a half-precision floating point number. It is mainly used in Deep Neural-Network training as of late because FP16 takes less memory, and enables training of larger models or training with larger mini-batches, and theoretically, it takes less time in calculations than FP32. Execution time can be sensitive to memory or arithmetic bandwidth. Half-precision halves the number of bytes accessed, thus reducing the time spent in memory-limited layers. This comes with a significant loss in the range that FP16 covers and the precision it can actually hold. FP16 precision format bits are divided as 1 bit for the sign, as always, 5 bits for the exponent or the magnitude, and 10 bits for the precision or the fraction as shown in Figure 1. The representable range for FP16 is in the order of 10^{-8} to 65504 with 4 significant decimal digits.

Automatic Mixed Precision

AMP (Automatic Mixed Precision) was released by NVIDIA in 2018, which is an extension for PyTorch, and provided a streamlined solution for using mixed-precision training in it(NVIDIA-Corporation 2022). Mixed precision training achieves all these benefits of FP16 while ensuring that no task-specific accuracy is lost compared to full precision training. It does so by identifying the steps that require full precision and using 32-bit floating point for only those steps while using 16-bit floating point everywhere else. Using mixed precision training requires two steps:

1. Porting the model to use the FP16 data type where appropriate.

2. Adding loss scaling to preserve small gradient values.

SSD300 model

The SSD300 model comprises 2 parts, extract feature maps and applying convolution filters to detect objects. (Liu et al. 2016) In our experiment, the model uses VGG16 to extract feature maps. Then it detects objects using the Conv4_3 layer. The structure is shown in Figure 2.

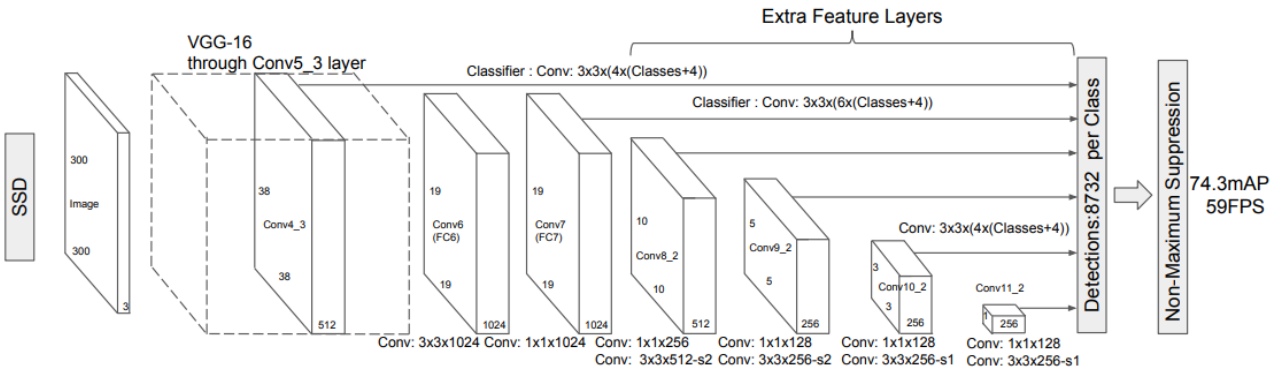

Figure 2: SSD architecture(Liu et al. 2016)

.

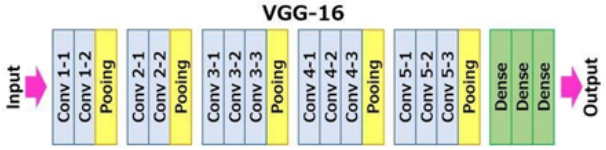

Figure 3: VGG16 architecture(Learning 2021)

.

VGG16 is a type of CNN (Convolutional Neural Network) that is considered to be one of the best computer vision models to date. The 16 in VGG16 refers to 16 layers that have weights. In VGG16 there are thirteen convolutional layers, five Max Pooling layers, and three Dense layers which sum up to 21 layers but it has only sixteen weight layers i.e., the learnable parameters layer. The structure is shown in figure 3. One thing to note is that VGG16, instead of having a large number of hyper-parameters, focused on having convolution layers of 3x3 filter with stride 1 and always used the same padding and maxpool layer of 2x2 filter of stride 2. Three Fully-Connected (FC) layers follow a stack of convolutional layers: the first two have 4096 channels each, and the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer.

After extracting the feature maps, SSD applies 6 more auxiliary 3×3 convolution filters for each cell to make predictions. These filters compute the results just like the regular CNN filters. Each filter outputs 25 channels: 21 scores for each class plus one boundary box. Since our goal is to measure the impact of precision on different neural networks, detailed model theories will not be discussed in this report.

The loss function used in the SSD300 is called MultiBox Loss, which is a weighted sum of the localization loss and the confidence loss. The localization loss is for the predicted locations of the boxes, it is a Smooth L1 loss between the predicted box and the ground truth box parameters. The confidence loss is for the predicted class scores, it is the softmax loss over multiple classes' confidences.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 4: ResNets architecture

.

ResNet-18

The first two layers of ResNet-18 are the 7*7 convolutional layers with 64 output channels and a stride of 2, which is followed by the 3*3 max-pooling layer with a stride of 2. ResNet-18 uses four modules made up of residual blocks, each of which uses several residual blocks with the same number of output channels. The number of channels in the first module is the same as the number of input channels. In the first residual block for each of the subsequent modules, the number of channels is doubled compared with that of the previous module, and the height and width are halved. There are 4 convolutional layers in each module (excluding the 1*1 convolutional layer). Together with the first 7*7 convolutional layer and the final fully connected layer, there are 18 layers in total. Therefore, this model is commonly known as ResNet-18. The structure can be found in Figure 4.

Experiment Results

To discover the impacts of different precisions on deep neural networks, we experiment with half mode (FP16), single mode (FP32), and amp mode (AMP) on ResNet-18 and SSD300 respectively. Due to the GPU running time limits on SageMaker and Google Colab, we reduce the number of training epochs to 58 for SSD300 and resumed training at a checkpoint each time after the previous session time ran out. We keep all the hyperparameters constant in the experiment to ensure the final performance is only impacted by the

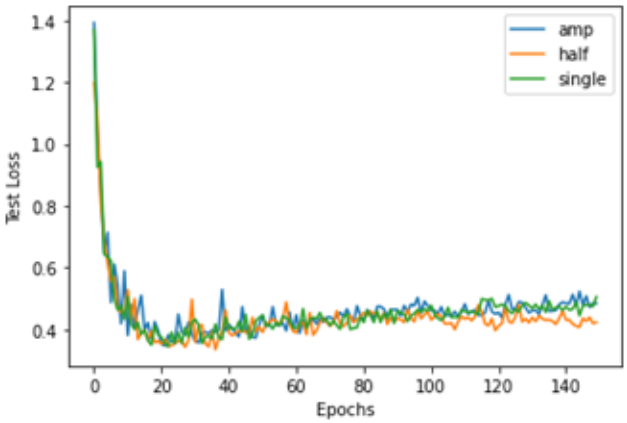

Figure 5: ResNet-18 test loss values

.

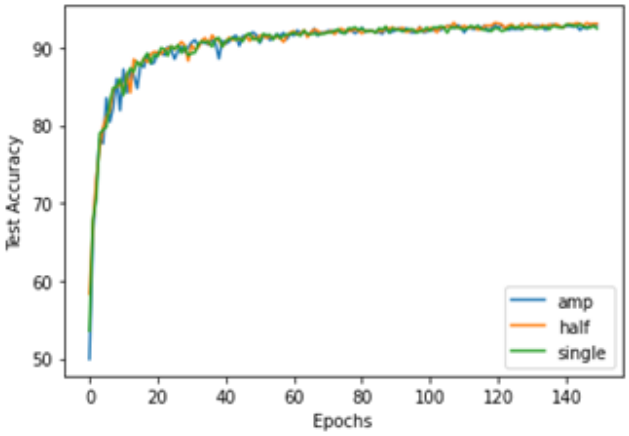

Figure 6: ResNet-18 test accuracies

.

precision.

Figure 5 shows the test loss values of ResNet-18 with different modes in the training process. In the first hundred epochs, the three modes are not that different. But after the hundredth round, the half mode became more stabilized than the other two modes and stood out at the end of our experiment.

In figure 6, there's not much difference between the three modes. They converged at a similar rate and all achieved 90% accuracy at the end. The reason might be that the precisions have small impacts on accuracy for a rather small model, like ResNet-18, which has size of 21 MB in half mode.

The average training time per epoch is quite different for the three modes as shown in Figure 7. ResNet-18 with half mode took 17.5 seconds per epoch, amp mode took 18.7 seconds, and single mode took the longest 35.1 seconds.

The test loss values of SSD300 with the three modes are shown below. With amp mode, SSD300 achieved the lowest loss value, 2.48, at the end of the experiment. One thing to note is that SSD300 with half mode led to lower loss value

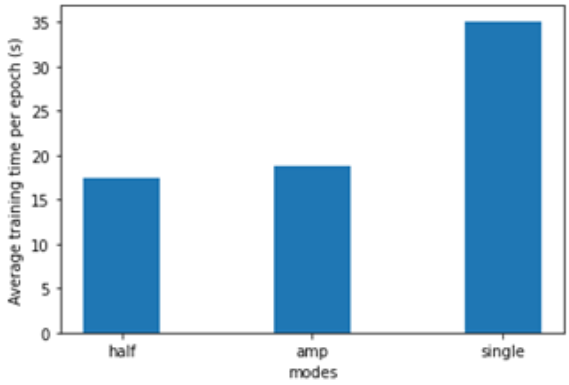

Figure 7: ResNet-18 average training time per epoch

.

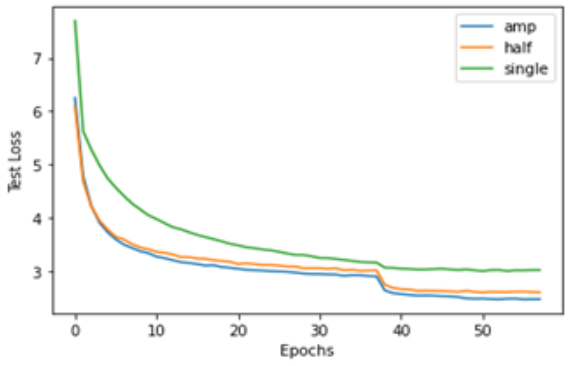

Figure 8: SSD300 test loss values

.

than single mode, the reason might be the same as we discuss in the results of ResNet-18 loss values.

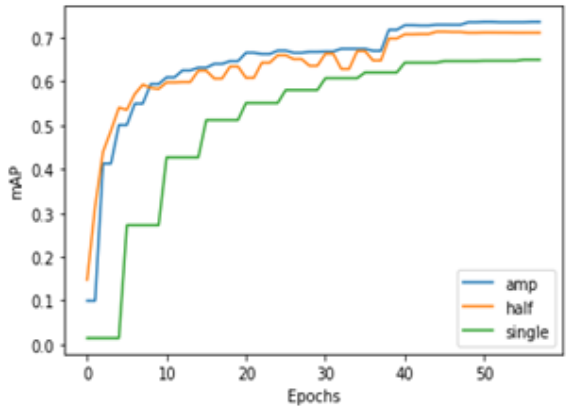

Figure 9: SSD300 Mean Average Precision

.

Since Mean Average Precision is a conventional way to measure the model accuracy for object detection tasks, we record and plot them out in Figure 9. SSD300 with amp mode led to 73.6% at the end of the experiment. Our re-

sult is close to 74.3%, which is the MAP in SSD: Single Shot MultiBox Detector (Wei, 2016). Considering the limited computing resources and training time, we evaluate the MAP every five epochs for single mode, and every two epochs for half and amp.

Finally, we plot the average training time per epoch for SSD300 with the three modes, shown in Figure 10. Half mode led to the shortest time, 0.23 hours, amp mode led to 0.30 hours, and single mode led to 0.33 hours. The half mode takes a huge advantage on the training time, especially when the number of training epochs becomes large.

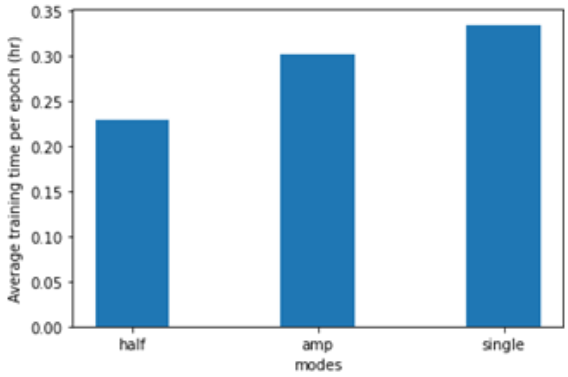

Figure 10: SSD300 average training time per epoch

.

Conclusion

In this research, We study the effect of different precision data representations and computation on deep neural network training. In the experiments, we choose SSD300 with VGG16 backbone and ResNet-18, which are two state-of-the-art models in objection detection and image classification fields respectively. In terms of the dataset, we select well-known VOC2007 and 2012 for the objection detection task, and CIFAR-10 for the image classification task. Both models are trained through single-precision, half-precision, and automatic mixed precision, holding other hyperparameters constant. According to the results, we can conclude that half-precision does shorten the training time without compromising training accuracy, where the accuracy is even higher than single-precision. On the other hand, though automatic mixed precision slightly increases training time, it improves training accuracy.

References

Chen, Y.; Luo, T.; Liu, S.; Zhang, S.; He, L.; Wang, J.; Li, L.; Chen, T.; Xu, Z.; Sun, N.; and Temam, O. 2014. DaDianNao: A Machine-Learning Supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 609–622.

Davis, J. 2021. Understanding mixed precision training.

Gupta, S.; Agrawal, A.; Gopalakrishnan, K.; and Narayanan, P. 2015. Deep Learning with Limited Numerical Precision. In Bach, F.; and Blei, D., eds., *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, 1737–1746. Lille, France: PMLR.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

Hoehfeld, M.; and Fahlman, S. 1992. Learning with limited numerical precision using the cascade-correlation algorithm. *IEEE Transactions on Neural Networks*, 3(4): 602–611.

Holt, J.; and Hwang, J.-N. 1991. Finite precision error analysis of neural network electronic hardware implementations. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume i, 519–525 vol.1.

Iwata; Yoshida; Matsuda; Sato; and Suzumura. 1989. An artificial neural network accelerator using general purpose 24 bit floating point digital signal processors. In *International 1989 Joint Conference on Neural Networks*, 171–175 vol.2.

Learning, G. 2021. Everything you need to know about VGG16.

Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.-Y.; and Berg, A. C. 2016. SSD: Single Shot MultiBox Detector. In *ECCV*.

NVIDIA-Corporation. 2022. Train With Mixed Precision.