DECEMBER 2022

TEXAS
The University of Texas at Austin

PyTorch Low-Precision Fixed-Point Training Flow

ECE382V Cross Layer ML Final Project

Jason Ho, Preston Glenn, Jordon Kashanchi
The University of Texas at Austin

Motivation

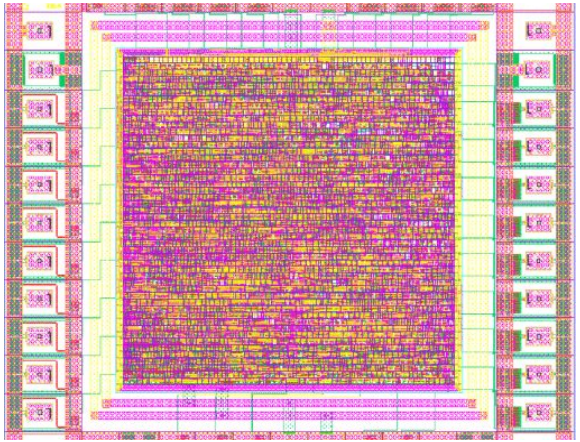
Deep learning training is incredibly computationally expensive due to the amount of precision used, Floating Point 32 bit typically, large size of models, and huge matrix operations.

There are multiple fixed point hardware advantages over floating point hardware.

- Less power consumption due to smaller physical size
- Increased speed and energy efficiency
- Lower latency times, memory usage, and computation

Alongside fixed point, we can take advantage of quantization to further reduce computation. However, using low bit-width fixed point data instead of high bit-width floating point data can also result in significant accuracy loss.

Can we train a model using quantized fixed point values without incurring significant accuracy loss?



Images: https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus-vu57p.html
https://www.researchgate.net/figure/Plot-of-ASIC-layout-submitted-for-manufacturing-The-die-size-is-337-mm-335-mm_fig5_271849890

What is Fixed Point?

- Allocates a binary integer part and a binary fractional part such that we can represent fractional numbers
- Determining the binary point, the separation between integer and fractional parts, trades off fractional precision for a greater range of values.
- Arithmetic operations are much cheaper than floating point as effectively binary arithmetic operations while lining up binary points.

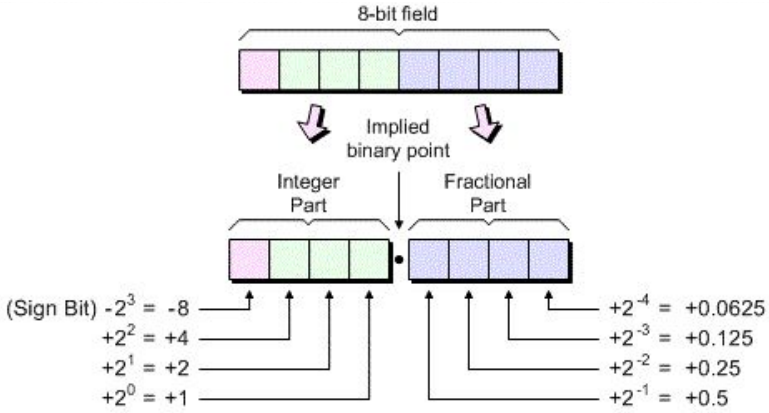
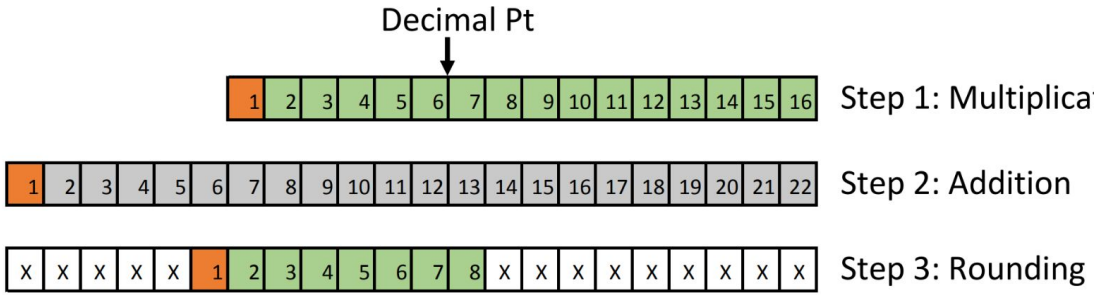
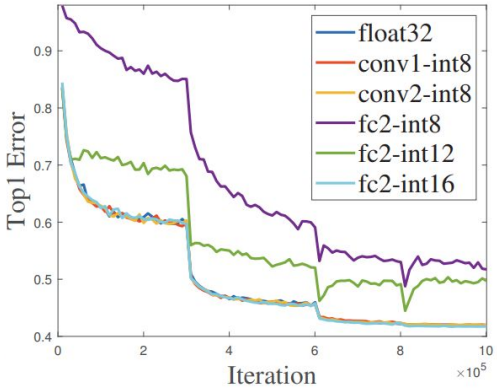


Image: https://www.allaboutcircuits.com/technical-articles/fixed-point-representation-the-q-format-and-addition-examples/

Background





(c) Training curve of AlexNet.

Prior literature gives us insight into what quantization strategies, fixed points formats, and hyper parameters have been successful in training.

The figure on the left from [1] shows an example of the instability in the network training phase when low precision activations are used. Meaning that quantization must be done diligently as to not lose significant precision after rounding.

The graph on the right from [2] shows the Top1 error when swapping out model layers with a fixed point format. We see that when the activation gradients of conv1 are quantized to int8 while keeping other layers float32, the training curve is the same as just using float32 and the final top1 accuracy is 58.01% on AlexNet. This proves that parts of the model can use fixed point without trading off accuracy.

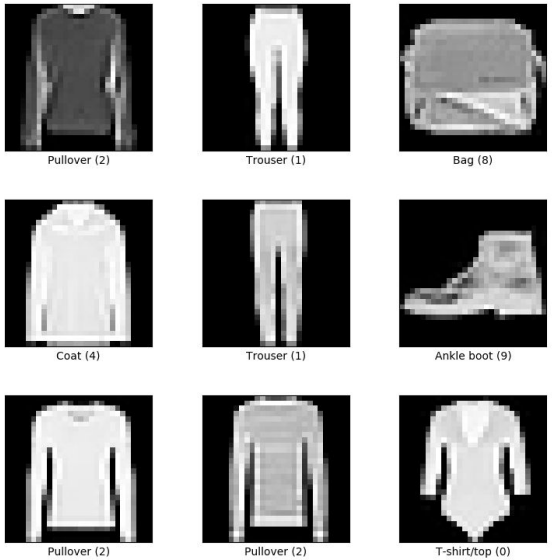
Approach on FashionMNIST

The model that was used was the same as Lab 3 which classified 28x28 FashionMNIST images, instead now we performed quantization on all parameters of training: loss, gradients, bias, weights, activation inputs, and activation outputs.

Previous literature only quantized parts of the model, but if everything is quantized, there is more hardware reuse.

The problem then becomes: **How do we choose fixed point format(s) to minimize accuracy loss?**

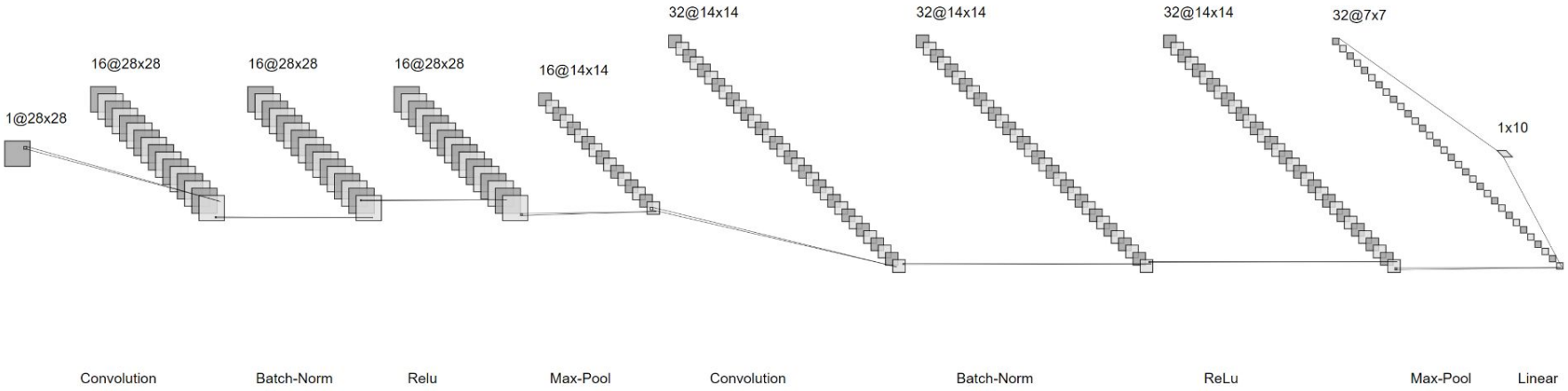
We use a FP32 model with the same model architecture and hyperparameters as our baseline model to compare against.



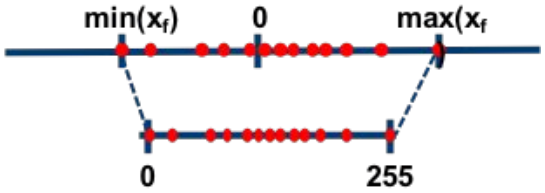
FashionMNIST Categories

Image: https://www.tensorflow.org/datasets/catalog/fashion_mnist

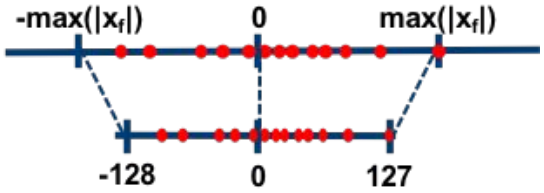
Model Architecture



Quantization Info



Asymmetric Mode



Symmetric Mode

How do we quantize? - Bias values may have a completely different range than loss, or gradients for example. We choose fixed point formats that minimize the quantization error, while being cognisant that it is expensive in hardware to support many different fixed point formats.

What do we quantize? - The more we are able to quantize, the greater the savings. That being said, quantizing certain layers in training may yield more savings. For example, quantizing weights and quantizing inputs into large parameter layers such as convolution or fully connected layers will yield greater savings than quantizing a batch normalization layer.

Images: https://intellabs.github.io/distiller/algo_quantization.html

Training Hyperparameters

- Rather than focus on fine-tuning hyperparameters, the fixed point formats of loss, bias, weights, gradients, activation_inputs, and activation_outputs affects accuracy more.

- There is also a lot of literature on optimal hyperparameters, therefore we do not focus on this part, even though we could get theoretical better results.

- Epochs = 5, Learning Rate = 0.001, Batch Size = 32, Adam Optimizer

Baseline Model

After referring to prior works and manually experimenting with fixed point formats, we established a baseline model that used fixed point formats as followed:

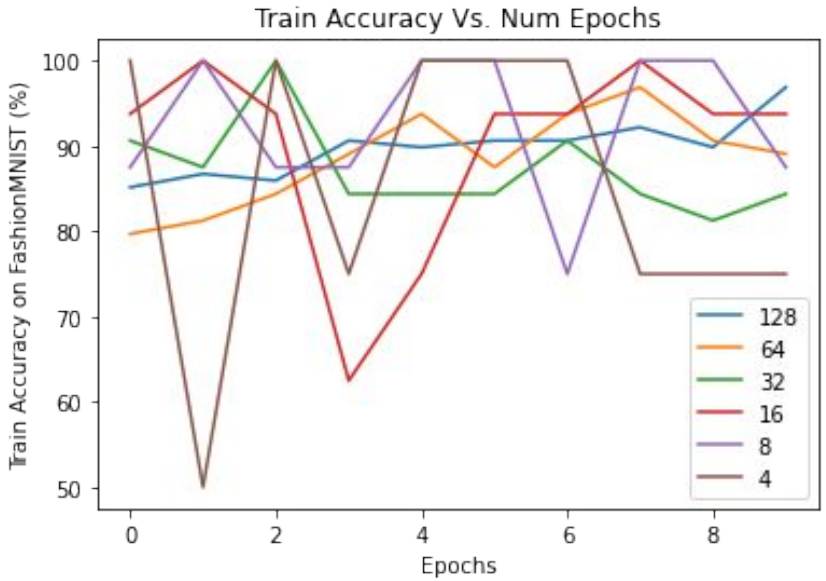
Activations	16 total bits, 8 fractional bits
Gradient	16 total bits, 10 fractional bits
Loss	16 total bits, 10 fractional bits
Weights	24 total bits, 12 fractional bits
Biases	16 total bits, 8 fractional bits

This achieved an accuracy of 87.5%, compared to 93.75% on the floating point model (FP32)

Performance Results - Basic Hyperparameters

To discover the best basic hyperparameters (number of epochs, batch size) to minimize accuracy loss, many experiments were iterated over these hyperparameter using the baseline model separately.

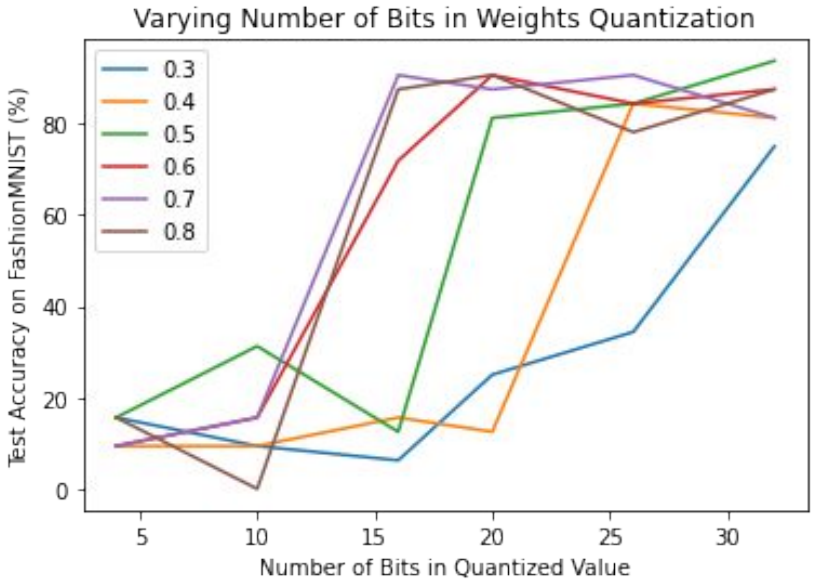
Shown on the right, each line represents a different batch size. For example, the green line represents a batch size of 32.



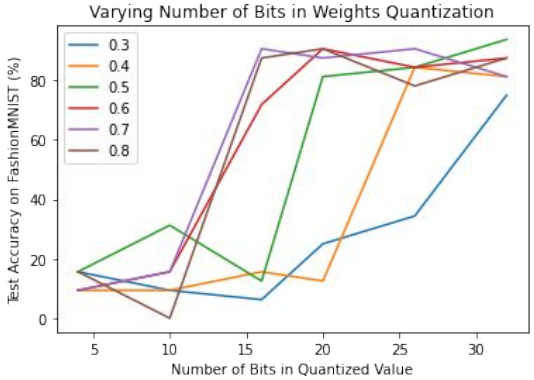
Performance Results - Quantization Hyperparameters

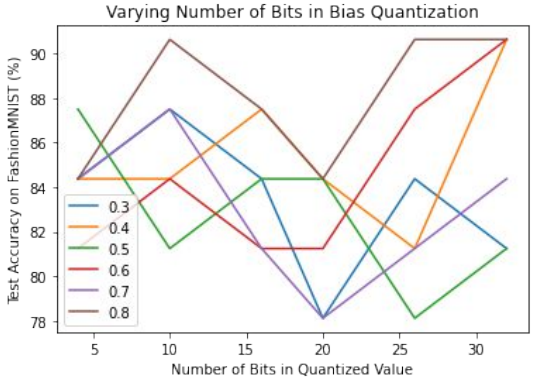
To discover the best fixed point formats to minimize accuracy loss, many experiments were iterated over each quantization hyperparameter using the baseline model separately.

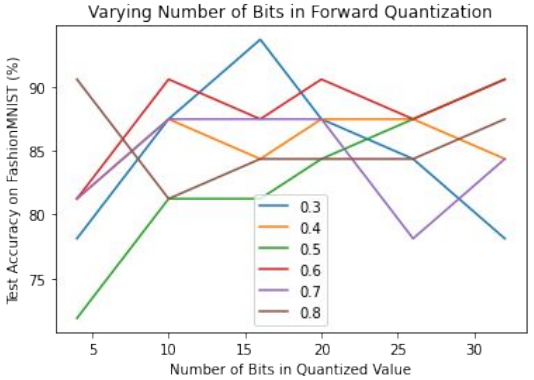
Shown on the right, each line represents a different composition of the fractional bits of the quantized value. For example, the green line represents half (0.5) of the quantized bits being used for the fixed point fraction.



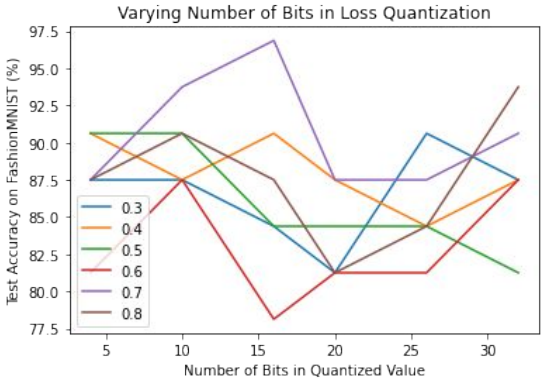
Performance Results

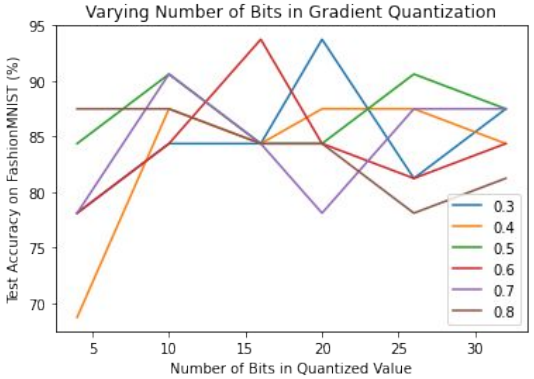






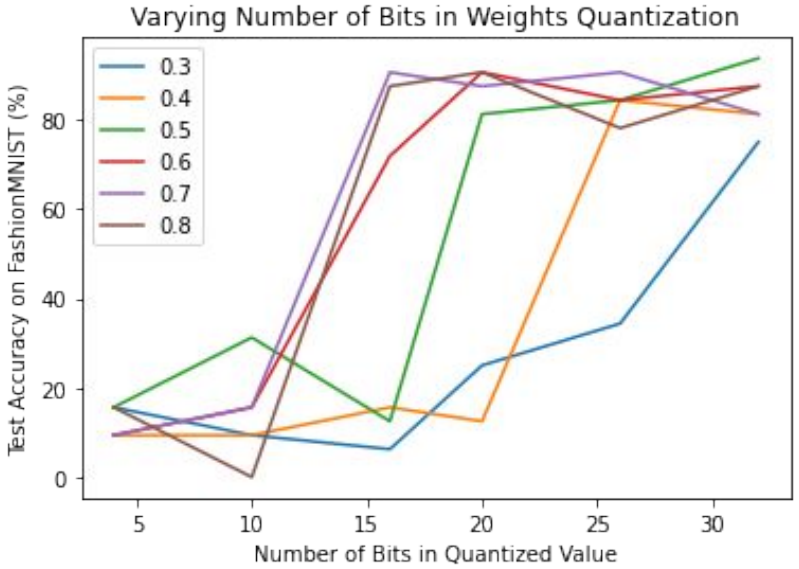
The same experiment was repeated for each of the quantization points.





Discussion of Performance Results

- For all graphs except changing weight fixed point formats, we attribute much of the test accuracy outputs to noise, but we can determine the relative amount of fractional bits that provide the best accuracies.
- The fixed point format of the weights severely affects test accuracy. We see that the weights value greater fractional precision.
- With this fractional precision, we can achieve results that rival 32 bits weights with just 16 bits!



Optimized Model

After basic and quantization hyperparameter tuning, we were able to achieve a fixed point training model that used lower precision formats than our baseline model, while maintaining high accuracy.

Activations	16 total bits, 4 fractional bits	*(Baseline, 16, 8)*
Gradient	16 total bits, 9 fractional bits	*(Baseline, 16, 10)*
Loss	16 total bits, 11 fractional bits	*(Baseline, 16, 10)*
Weights	16 total bits, 11 fractional bits	*(Baseline, 24, 12)*
Biases	10 total bits, 8 fractional bits	*(Baseline, 16, 8)*

This achieved an accuracy of 84.4% and had greatly reduced precision on weights and biases.

Conclusion

In this project we were able to train a FashionMNIST model in Pytorch with all parameters set to 16 bit precision fixed point notation, while only dropping 9.35% test accuracy.

We pinpointed the best model hyperparameters parameters, performed rigorous testing, and highlighted the best quantization technique from our exploration.

We discovered that all hyperparameters aside for weights and biases can be deeply quantized without trading off significant accuracy. In addition, a high composition of the fixed point format must be dedicated to fractional bits.

This fixed point model can be ported onto an FPGA or ASIC for better power, performance, and area than floating point hardware.

Future Work

In this project, all work and inference was done on one dataset with one model: FashionMNIST. In the future, this project could be expanded to evaluate additional models on different datasets. The FashionMNIST dataset was a relatively easy problem to learn.

Additionally, many of the graphs gathered failed to show any trends. Instead of evaluating the model at different parameters once, we could expand the testing to running each parameter combination a series of times and then averaging the results. This would allow us to better understand the significance of the results.

Bibliography

[1] D. D. Lin and S. S. Talathi, “Overcoming Challenges in Fixed Point Training of Deep Convolutional Networks,” arxiv.org, Jul. 2016, doi: 10.48550/arXiv.1607.02241.

[2] X. Zhang et al., "Fixed-Point Back-Propagation Training," 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 2327-2335, doi: 10.1109/CVPR42600.2020.00240.

[3] X. Chen, X. Hu, H. Zhou, and N. Xu, “FxpNet: Training a deep convolutional neural network in fixed-point representation,” 2017 International Joint Conference on Neural Networks (IJCNN), May 2017, doi: 10.1109/ijcnn.2017.7966159.

[4] S. Lee, J. Park, and D. Jeon, “TOWARD EFFICIENT LOW-PRECISION TRAINING: DATA FORMAT OPTIMIZATION AND HYSTERESIS QUANTIZATION,” ICLR 2022, Mar. 2022