DRAFT: Quantization: How to Maintain Model Accuracy While Reducing the Size of the Model

## Paul Jojy

Johns Hopkins University

[pjojy1@jh.edu](mailto:pjojy1@jh.edu)

# Abstract

As machine learning becomes more prevalent and deep learning-based models become more popular, demand has risen for deep learning implementations on everything from mobile phones to space capable hardware. However, deep learning- based models are large and resource intensive; and in resource limited environments such as space, the models have to run within SWaP-C (Size, Weight, Power and Cost) constraints. This calls for efficient strategies to optimize the limited amount of available memory when running inference schemes. In this paper, I will be exploring one such scheme known as quantization. Quantization essentially truncates float weights; converting the weights to integer weights. Integer- only arithmetic is less intensive than floating point arithmetic. I will be exploring different quantization strategies and applying the strategies to a few commonly used models to observe how the model size is affected. I will also be analyzing throughput for comparison. One common concern with integer-only weights and arithmetic is that the model will lose accuracy. Therefore, I will provide an end-to- end procedure on how to train, quantize, and fine-tune a model such that accuracy is preserved.

# Introduction

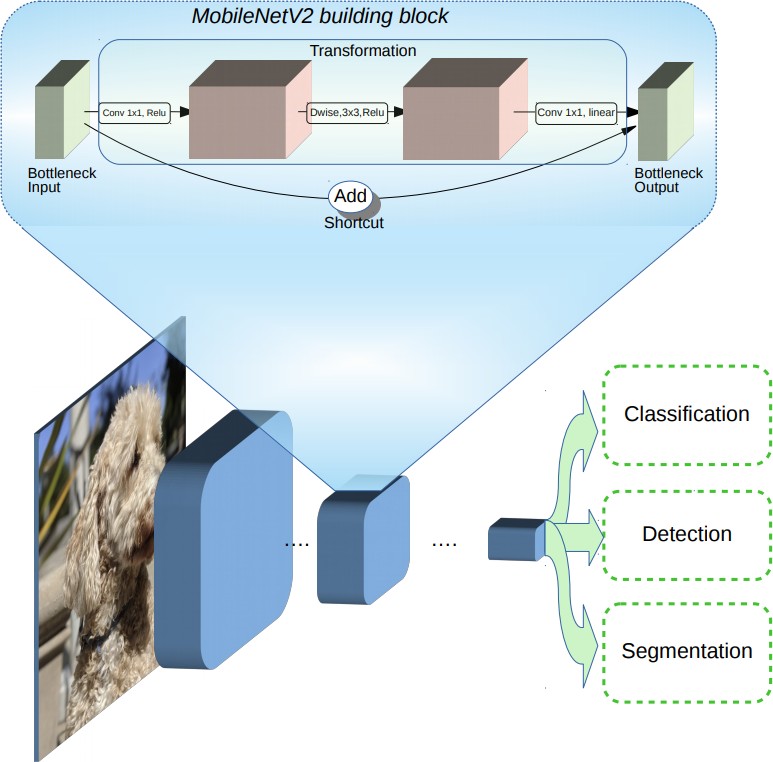
## Quantization Concept Overview

The concept of deep learning has been around for a long time. Innovation in the field has primarily been driven by classification accuracy. Therefore, neural network architectures have mostly grown without regard for model complexity and efficiency [1]. However, with the recent emergence and popularity of edge devices (such as phones, AR/VR devices, etc.), demand has increased for enabling these complex neural networks to run in resource constrained environments. In order for these models to run on edge devices while respecting these constraints, model complexity has to be reduced without sacrificing model accuracy. This is where quantization can be useful. Running a neural network on hardware typically involve millions or multiplication and addition operations in order to adjust the weights in the activation nodes. Quantization involves converting the floating point weights to integer format, substantially decreasing the memory footprint of the model (as less bits are needed to store integer weights). Lower-bit mathematical operations result could result in large computational gains and higher performance. One major concern with quantization is that essentially truncating the weights leads to loss of information captured by the floating point bits, leading to a dramatic drop in accuracy. However, innovation in the field has enabled quantization of the model with trivial drop in performance.

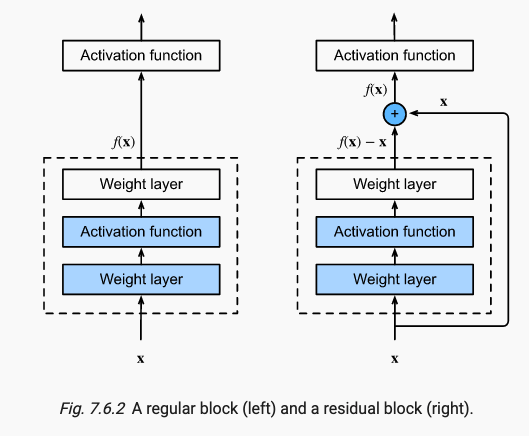
## Model Selection

In this paper, we will be exploring 2 different architectures: MobileNetV2 and ResNetV2 (with 50,101, and 152 skip connections).

MobileNetV2 is a popular architecture that’s designed with mobile devices in mind. This network is mainly used in classification, object detection, semantic segmentation. MobileNetV2 was unveiled by Google Research in 2018, as a significant upgrade to MobileNetV1. Like MobileNetV1, V2 also uses depthwise separable convolution as the basis. However, V2 includes linear bottlenecks between the layers, and skip connections between the bottlenecks [4]. The popularity of this model makes it a good candidate for experimentation. Another reason why MobileNetV2 is because it is known to have a large accuracy drop after quantization [1,6]. The below image is a representation of the underlying architecture, provided by Google AI blog [4].



ResNetV2 is another popular architecture that is mainly used for image classification, regression, and feature extraction. The main feature of ResNet is that it consists of stacks of residual blocks. The residual blocks can be skipped by a skip connection, which is used to counteract the Explod- ing/Vanishing gradient problem. This skip connection guarantees the the higher layer will perform at least as well as the lower layer. This architecture is a good candidate for this study because we can tweak the size of the model by adding 50,101, or 152 residual blocks. Therefore, we can perform an analysis at different model sizes of the same baseline architecture. The below image is a representation of the underlying architecture, which consists of residual blocks and skip connections. The left diagram represents a normal residual block, which is outlined by a dashed boundary. The right diagram introduces the skip connection to the architecture. This image was provided by the Dive into Deep Learning textbook [10].



## Quantization Strategies

We will be employing a few different quantization techniques in order to analyze and compare performance between the different strategies.

The first strategy we will try is Dynamic Range Quantization. This method statically quantizes only the weights from floating point to int8 format. Then, at inference, the weights are converted back to

floating point and computed using floating point kernels. This conversion is only done once and then is cached to reduce latency [5].

The next strategy we will employ is Full integer quantization. There are many devices and accelerators, such as the Xilinx (now AMD) Versal AI Core Series VCK190, that rely on integer only operations to speed up the system. In these instances, it’s best to employ full integer quantization. Like the name suggests, all model math is integer quantized. For this process, you also need to provide a small subset of the data in order to fine-tune the model. This prevents major accuracy loss.

Next, we will try Float16 quantization. Quantizing weights to float16 can reduce the model size by up to half. Since the weights are not being fully truncated to integer format, accuracy loss is minimal. The computations will also be faster than float32 computations [5]. However, it does not reduce latency as much as reducing the model to fixed point math.

Using multiple quantization strategies on different models allows us to analyze and compare as to which combinations yield the best results.

# Setup

## Data

All models were trained using the ImageNet dataset. This dataset is most popular among researchers and is featured in many quantization papers as a useful tool for benchmarking performance. ImageNet is an image database containing myriad images, and has been instrumental in advancing computer vision and deep learning research [11]. We will also be using a 1000 image subset of ImageNet, which contains 1 image per class [9]. This ensures a balanced subset of images which we can use to test our unquantized and quantized models.

## Procedure

All research was done using the Tensorflow Framework. Using preprocessed image test data, the trained models were used to generate a baseline accuracy, then put through various quantization schemes to produce quantized models. The quantized models are then used to measure accuracy using the same test dataset. The code supporting this research paper has been uploaded to github as a jupyter notebook [https://github.com/paulmjojy/705.603/blob/master/Research/](https://github.com/paulmjojy/705.603/blob/master/Research/Research.ipynb) [Research.ipynb](https://github.com/paulmjojy/705.603/blob/master/Research/Research.ipynb), which contains additional annotations regarding the process.

## Experiments

I conducted a few different experiments in order to understand the different quantization strategies. One test invloved taking each architecture through the different strategies, and saving the model to disk in order to compare the difference in memory footprint. Another experiment involved taking each architecture through the different strategies, and comparing accuracies on the test set. Finally, I compared the time it took for each model to predict classes for 1000 images. This time does not include converting the predicted labels to class names or calculating accuracy. it only consists of the time to run a prediction on 1000 images using the model. Please refer to the aforementioned jupyter notebook for specifics.

# Results and Analysis

In this section we will go over the results of aforementioned experiments. These results have been divided by the architecture they were performed on, in order to better visualize and compare the quantization strategies.

## MobileNet

* + 1. **Size Comparisons**

|  |  |
| --- | --- |
| **Model Type** | **Size (bytes)** |
| Original Model | 14,452,224 |
| Dynamic Range Quantized Model | 3,930,912 |
| Full Integer Quantized Model | 3,996,936 |
| Float16 Quantized Model | 7,036,176 |

The above table shows the difference in memory between the quantization strategies. Dynamic Range quantization drastically shrinks the model size by roughly 72.41%. Going from a Float32 to Float16 reduces the model by roughly half the size. All quantized models are much smaller than the original size.

## Performance Comparisons

|  |  |
| --- | --- |
| **Model Type** | **Accuracy%** |
| Original Model | 83.90 |
| Dynamic Range Quantized Model | 83.30 |
| Full Integer Quantized Model | 82.00 |
| Float16 Quantized Model | 83.70 |

While the original model has the best performance on the test set, Reducing the model size by 72.41% through dynamic quantization only results in a performance drop of 0.6%. Reducing the model size by half only results in a .2% performance drop in the Float16 model. The full integer quantized model has the biggest performance drop, but it is only worse by 1.90%. It is expected that full integer quantization will yield the largest performance drop, as converting to integer would cause the most performance loss.

## ResNet

* + 1. **Size Comparisons**

|  |  |
| --- | --- |
| **Model Type** | **Size (bytes)** |
| Original Model | 102,760,160 |
| Dynamic Range Quantized Model | 26,322,384 |
| Full Integer Quantized Model | 26,341,144 |
| Float16 Quantized Model | 51,198,176 |

Table 1: ResNet50V2.

|  |  |
| --- | --- |
| **Model Type** | **Size (bytes)** |
| Original Model | 179,298,792 |
| Dynamic Range Quantized Model | 46,081,616 |
| Full Integer Quantized Model | 46,103,696 |
| Float16 Quantized Model | 89,272,112 |

Table 2: ResNet101V2.

|  |  |
| --- | --- |
| **Model Type** | **Size (bytes)** |
| Original Model | 242,413,216 |
| Dynamic Range Quantized Model | 62,407,840 |
| Full Integer Quantized Model | 62,433,008 |
| Float16 Quantized Model | 120,646,400 |

Table 3: ResNet152V2.

Similar to the MobileNet model, the model size for all 3 ResNetV2 gets cut by roughly half going from Float32 to Float16. Using Dynamic Range Quantization cuts the model size drastically, by 74.40%. Again, we see that quantized models are much smaller than the original.

## Performance Comparisons

|  |  |
| --- | --- |
| **Model Type** | **Accuracy%** |
| Original Model | 73.10 |
| Dynamic Range Quantized Model | 72.90 |
| Full Integer Quantized Model | 72.10 |
| Float16 Quantized Model | 73.00 |

Table 4: ResNet50.

|  |  |
| --- | --- |
| **Model Type** | **Accuracy%** |
| Original Model | 76.70 |
| Dynamic Range Quantized Model | 76.20 |
| Full Integer Quantized Model | 76.10 |
| Float16 Quantized Model | 76.70 |

Table 5: ResNet101.

|  |  |
| --- | --- |
| **Model Type** | **Accuracy%** |
| Original Model | 77.50 |
| Dynamic Range Quantized Model | 77.10 |
| Full Integer Quantized Model | 76.60 |
| Float16 Quantized Model | 77.50 |

Table 6: ResNet152.

For all 3 ResNet models, we see a trivial drop in performance for any of the quantized accuracies, compared to the original accuracy. The biggest drop we see is in the Full integer quantized accuracy. As mentioned before, this is because converting all weights and activations to integer results in the most information loss.

# Conclusion

Quantization can be a very powerful tool to bring deep learning solutions to edge devices. As shown in the tables above, drastically reducing the memory footprint of any given model results in minimal accuracy loss. In these experiments, it seems like Dynamic Range quantization had the best size vs. accuracy tradeoff. However, it may not always be the best quantization strategy for every problem. Like mentioned before, some edge devices and hardware accelerators only allow integer computations. In this case, full integer quantization would be the best option. Future work might include measuring latency for non-quantized vs. quantized models in TensorFlow. The difficulty in doing this, however, is that TensorFlow contains optimized prediction/data generation methods for non-quantized models. Quantized models do not have optimized methods. Therefore, it would not be fair to compare latency between these 2 models on this framework.

# References

1. B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, D. Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. [https://openaccess.thecvf. com/content\_cvpr\_2018/papers/Jacob\_Quantization\_and\_Training\_CVPR\_2018\_paper.pdf](https://openaccess.thecvf.com/content_cvpr_2018/papers/Jacob_Quantization_and_Training_CVPR_2018_paper.pdf) 2018.
2. M. Nagel, M. Baalen, T. Blankevoort, M. Welling. Data-Free Quantization Through Weight Equalization and Bias Correction. <https://arxiv.org/pdf/1906.04721.pdf>2019.
3. I. Hubara, Y. Nahshan, Y. Hanani, R. Banner, D. Soudry. Improving Post Training Neural Quantization: Layer-wise Calibration and Integer Programming. <https://arxiv.org/pdf/2006.10518.pdf>2020.
4. Sandler, Mark, and Andrew Howard. “MobileNetV2: The next Generation of on-Device Com- puter Vision Networks.” Google AI Blog, 3 Apr. 2018, [https://ai.googleblog.com/2018/04/ mobilenetv2-next-generation-of-on.html](https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html)
5. “Post-Training Quantization : Tensorflow Lite.” TensorFlow, 17 Nov. 2021, [https://www.tensorflow. org/lite/performance/post\_training\_quantization](https://www.tensorflow.org/lite/performance/post_training_quantization)
6. “Vitis AI User Documentation.” Quantizing the Model, Xilinx, 17 Dec. 2020, [https://www.xilinx.com/ html\_docs/xilinx2019\_2/vitis\_doc/zvf1570695925069.html](https://www.xilinx.com/html_docs/xilinx2019_2/vitis_doc/zvf1570695925069.html)
7. “Here’s Why Quantization Matters for Ai.” Qualcomm News, Qualcomm, 12 Jan. 2022, [https://www. qualcomm.com/news/onq/2019/03/12/heres-why-quantization-matters-ai](https://www.qualcomm.com/news/onq/2019/03/12/heres-why-quantization-matters-ai)
8. Schwartz, Eli. “Elischwartz/Imagenet-Sample-Images: 1000 Images, One per Image-Net Class. for Easy Visualization/Exploration of Classes.” GitHub, 1 Sept. 2021, [https://github.com/EliSchwartz/ imagenet-sample-images](https://github.com/EliSchwartz/imagenet-sample-images)
9. Zhang, Aston, et al. “Dive into Deep Learning.” 7.6. Residual Networks (ResNet) - Dive into Deep Learning

0.17.4 Documentation, <https://d2l.ai/chapter_convolutional-modern/resnet.html>

1. ImageNet, Stanford University, 11 Mar. 2021, <https://www.image-net.org/index.php>
2. "Post-training integer quantization : Tensorflow Lite." TensorFlow, 06 Jan. 2022, [https://www. tensorflow.org/lite/performance/post\_training\_integer\_quant](https://www.tensorflow.org/lite/performance/post_training_integer_quant)