Data Science Institute
COLUMBIA UNIVERSITY

Fall 2020 Capstone Project

Progress Report I

Energy-Efficient AI on EDGE

Kumari Nishu (kn2492)
Neelam Patodia (np2723)
Mohit Chander Gulla (mcg2208)
Pritam Biswas (pb2796)
Prasham Dhaneshbhai Sheth (pds2136)

Table of Contents

1. Problem Definition and Progress Overview **2**

2. Methodology **2**

2.1 Quantization Background 2

2.1.1 Types of Quantization 3

2.1.2 Progress made in the field 3

2.1.3 Choice of Quantization 4

2.1.4 Benchmark Results 4

2.2 Our Approach 4

2.2.1 Data and Model Selection 5

2.2.2 Weight Binning Techniques 6

2.2.3 Quantizer Design Techniques 6

2.2.4 Granularity of Quantization: 8

3. Results **8**

3.1 ANN-based Classification 8

3.2 ANN-based Regression 9

3.3 Convolutional Neural Networks 11

3.4 Key Takeaways 12

3.5 Solutions to the listed Questions 14

3.5.1 Why does performance degrade with quantization? 14

3.5.2 Why do different datasets behave differently with quantization? 15

3.5.3 How to improve performance? 15

4. Next Steps **15**

4.1 Different levels of precision for different layers of a neural network 15

4.2 Quantization Aware Training 15

5. Individual Contribution **16**

6. References **17**

1. Problem Definition and Progress Overview

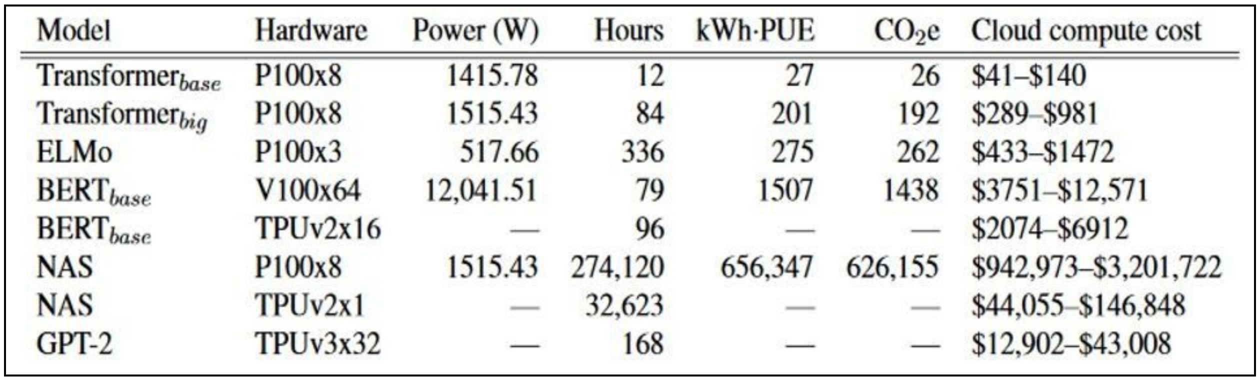
This capstone project is sponsored by General Electric. The project aims to develop techniques for training and inference of machine learning models with reduced carbon footprint. Recent estimates suggest training deep learning models such as BERT, ELMo and GPT-2 requires training on multiple GPU's for a few days. The carbon emissions from training a deep learning model is equivalent to 5 times the lifetime emissions of an average car. Hence, GE requires low-latency and lighter machine learning models without compromising accuracy, which can be deployed on GE's EDGE devices.

Our objective is to explore techniques to store a model using lower precision and assessing its effect during inference.

To tackle this problem we propose model inference on lower bits of precision than the default bit precision of 32 / 64 in Python. This method is referred to as Quantization in deep learning literature. This technique has multiple benefits - it reduces the size of the model, it leads to faster training and inference due to low precision arithmetic (requires specialized hardware).

Quantization in deep neural networks can be post-training quantization or quantization aware training. We have mainly concentrated on post-training quantization so far in this project. Post-training quantization does not affect the training of a model, instead it reduces the bit-precision of the learned weights during inference based on heuristics.

Our technology stack is - Python3, Pytorch, Distiller
We have setup a repository for our code which can be accessed at - **https://github.com/mohitgulla/Edge**



Model	Hardware	Power (W)	Hours	kWh·PUE	CO_2e	Cloud compute cost
$\text{Transformer}_{base}$	P100x8	1415.78	12	27	26	\$41–\$140
Transformer_{big}	P100x8	1515.43	84	201	192	\$289–\$981
ELMo	P100x3	517.66	336	275	262	\$433–\$1472
BERT_{base}	V100x64	12,041.51	79	1507	1438	\$3751–\$12,571
BERT_{base}	TPUv2x16	—	96	—	—	\$2074–\$6912
NAS	P100x8	1515.43	274,120	656,347	626,155	\$942,973–\$3,201,722
NAS	TPUv2x1	—	32,623	—	—	\$44,055–\$146,848
GPT-2	TPUv3x32	—	168	—	—	\$12,902–\$43,008

Table 1. Power usages and compute costs of deep learning models

2. Methodology

2.1 Quantization Background

Our objective is to compress the network as much as possible, to reduce the amount of bandwidth and computation required. One way to compress the network is through regularization or pruning to induce sparsity in weights and activation functions. Another method is quantization which is the process of representing a number by reducing the number of bits. For the scope of this project, we would focus on quantization based techniques for model compression.

2.1.1 Types of Quantization

Fundamentally quantization means introducing approximations and the resulting networks have slightly less accuracy. These techniques attempt to minimize the gap between the full floating point accuracy and the quantized accuracy. There are three broad quantization modes:

- **Dynamic Quantization**: The activations and weights are quantized on the fly and are thus referred to as dynamic. Only the computations (multiplication and convolutions) are efficient. Activations are read and written in floating point

- **Post Training Static Quantization**: Allows for both integer arithmetic operations and memory access. This is achieved by first obtaining activation/weight distributions and using that to quantize activations/weights. These allow us to pass quantized values without floating point-integer-floating point conversions. For this technique, selecting the range of the tensor is crucial. There are various ways to select a suitable range, starting from the actual min/max values of the tensor to derivation based on the tensor's range / distribution to come up with a narrower min/max range, in order to remove possible outliers. These again could be implemented in two modes:symmetric and asymmetric. In this report, we are focussing on this technique alone and further details have been listed ahead

- **Quantization Aware Training**: Quantization Aware Training is a technique that quantizes the model weights and activations during the forward and backward passes of training. Hence the model is aware that it will be quantized during inference, so it tries to learn quantized weights during the training itself. According to previous literature, this technique gives the highest accuracy metrics compared to post-training quantization as the model learns quantized weights only. The technique however comes with a disadvantage that it may significantly increase the training time, as the minibatch tensors and weight tensors would need to pass through quantization modules during the forward pass

2.1.2 Progress made in the field

Given the vast success of Convolutional Neural Networks (CNN) models, substantial efforts have been made towards the category of CNN network quantization. Some of the key contributions have been listed below:

- Gong et al. (2014) address the storage problem of AlexNet with **vector quantization** techniques. By replacing the weights in each of the three fully connected layers with respective floating-point centroid values obtained from the clustering
- **HashedNet** (Chen et al., 2015b) uses a hash function to randomly map pre-trained weights into hash buckets, and all the weights in the same hash bucket are constrained to share a single floating-point value
- Zhou et al. (2017) introduces **Incremental network quantization** (INQ) to account for the non-negligible accuracy loss for CNN quantization methods. It uses three interdependent operations, namely weight partition, group-wise quantization and re-training. There is no assumption on the CNN architecture. These three operations are repeated on the latest re-trained group in an iterative manner until all the weights are converted into low-precision ones, acting as an incremental network quantization and accuracy enhancement procedure

2.1.3 Choice of Quantization

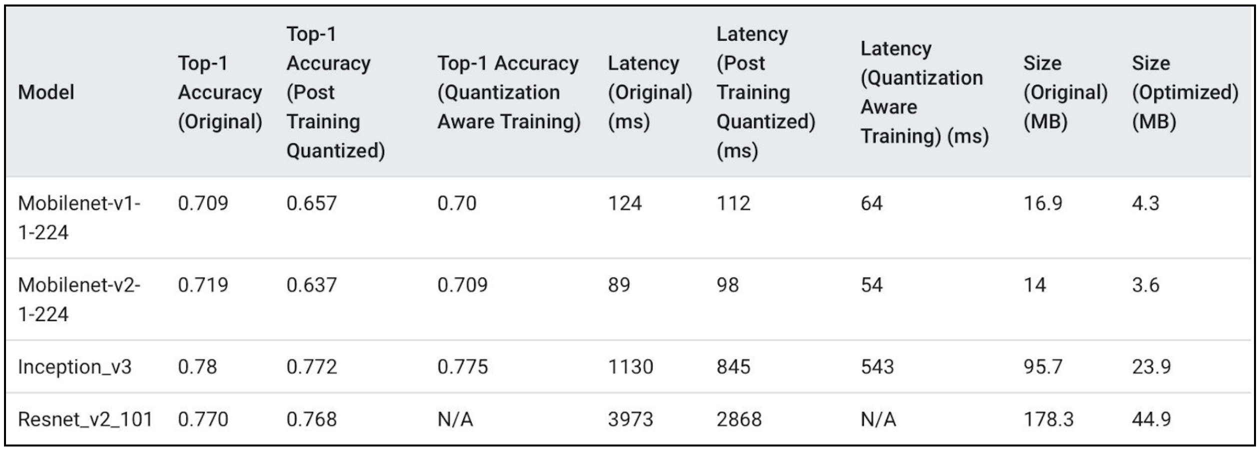
The choice of a quantization mode is affected by model requirements and operator support. As such for this project, we also aim to explore the preferred quantization techniques for a given architecture. The table below lists a few reasonings.

Model Type	**Preferred Scheme**	**Why**
LSTM/RNN	Dynamic Quantization	Throughput dominated by compute/memory bandwidth for weights
BERT/ Transformer	Dynamic Quantization	Throughput dominated by compute/memory bandwidth for weights
CNN	Static Quantization	Throughput limited by memory bandwidth for activations
CNN	Quantization Aware Training	In the case where accuracy can't be achieved with static quantization

Table 2: Choice of quantization as per model category [2]

2.1.4 Benchmark Results

Below are latency and accuracy results for post-training quantization and quantization-aware training on a few models.



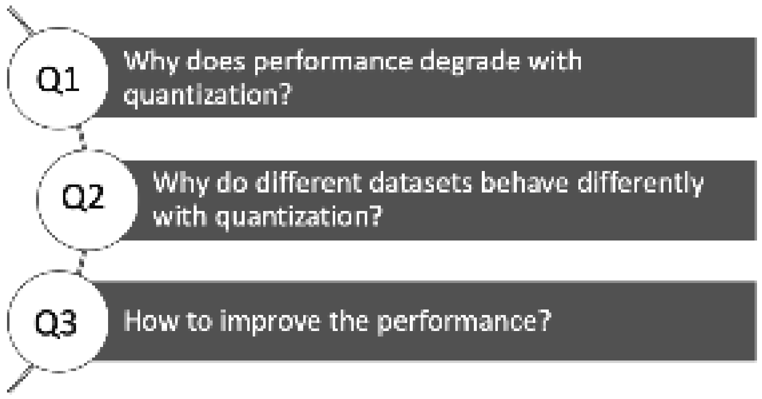
Model	Top-1 Accuracy (Original)	Top-1 Accuracy (Post Training Quantized)	Top-1 Accuracy (Quantization Aware Training)	Latency (Original) (ms)	Latency (Post Training Quantized) (ms)	Latency (Quantization Aware Training) (ms)	Size (Original) (MB)	Size (Optimized) (MB)
Mobilenet-v1-1-224	0.709	0.657	0.70	124	112	64	16.9	4.3
Mobilenet-v2-1-224	0.719	0.637	0.709	89	98	54	14	3.6
Inception_v3	0.78	0.772	0.775	1130	845	543	95.7	23.9
Resnet_v2_101	0.770	0.768	N/A	3973	2868	N/A	178.3	44.9

Table 3 : Benefits of model quantization for select CNN models [3]

2.2 Our Approach

As a starting point, we are looking at model compression as post-quantization of weights. The focus of the experiment is to analyze the trade-off between accuracy and precision for various quantization techniques, and not look into the trade-off between model size/latency and precision. We are using quantization only in the context of more efficient inference.

As of now, we have focussed on finding solutions to the following questions:


Q1
Why does performance degrade with quantization?
Q2
Why do different datasets behave differently with quantization?
Q3
How to improve the performance?

We attempt to answer the above questions by exploring varied quantization techniques on different model categories (Classification, Regression) across multiple datasets.

2.2.1 Data and Model Selection

For model selection, we decided to experiment with neural networks owing to a large number of trainable parameters. We experimented with both simple and complex architectures for classification and regression problems. Model compression exercise on models such as logistic regression, SVM which are dependent on the number of features in the dataset limits our scope in exploring the problem space. As we also needed to explore in particular why different datasets behave differently with quantization, we experimented with a given model architecture across varied datasets. Moderately sized datasets have been chosen to reduce chances of underfitting.

Model Category	Model Complexity	Dataset
ANN based Classification	Simple: 2 dense layers Complex: 5 dense layers	Churn Data Telescope Data
ANN based Regression	Simple: 2 dense layers Complex: 5 dense layers	California Housing Data MV Data
CNN	Vanilla CNN ResNet9	Fashion MNIST CIFAR-100

Table 4 : Different Network architectures and datasets used for this project

Dataset Details			
Dataset Name	**Number of Rows**	**Number of Features**	**Number of Classes**
Churn Data	10,000	14	2
Telescope Data	19,000	11	2
California Housing Data	20,600	8	-

MV Data	40,700	10	-
Fashion MNIST	70,000	-	10
CIFAR- 100	60,000	-	100

Table 5 : Details for the datasets used

2.2.2 Weight Binning Techniques

During post training quantization, given a representational precision value 'n' for a model, we need to specify 2^n unique values/bins. We then map the original weights of a layer to these unique values using rounding techniques. E.g. For a 2 bit precision, we have 4 unique values and we map the original weights of a layer to these 4 unique values (many-one). We have explored the following binning techniques:

- **Uniform Range**: The unique values are chosen from a uniform distribution over the range of values of the learnt weights
- **Histogram**: The learnt weights are plotted as a histogram with bins equal to the number of unique values in an n-bit representation and the center of the bin is chosen as the quantized weights/unique values
- **Normal Prior**: Assuming that the learnt weights have a normal distribution, we chose the unique values based on quantiles

2.2.3 Quantizer Design Techniques

- **Normal Rounding**:

Normal rounding involves rounding the floating point model weights to their closest value in uniform size bins. Currently we have done rounding on a per layer basis. We find the min and max values per layer and divide the range into equal sized bins. We then map each weight to its closest bin value and use these mapped values for inference. The algorithm pseudocode is described below:

PSEUDOCODE FOR NORMAL ROUNDING

```
func normal_rounding(model, precision):
    for layer in model.layers:
        min_val = get_min(layer) // get minimum weight value in the layer
        max_val = get_max(layer) // get maximum weight value in the layer
        num_bins = 2^precision
        bin_values = get_bins(min_val, max_val, num_bins) // get the values of all bins. This can be generated using different ways
        new_layer = None
        for weight in layer:
            mapped_weight = get_closest_match(weight, bin_values) // binary search for closest bin value
            new_layer.add(mapped_weight)
        layer = new_layer // cloning the layer with mapped values
    return model
```

- **Stochastic Rounding:**

This technique calculates probability values in order to make the decision of where to round.

$$\text{Round}(x) = \begin{cases} \lfloor x \rfloor & \text{with probability } 1 - (x - \lfloor x \rfloor) \\ \lfloor x \rfloor + 1 & \text{with probability } x - \lfloor x \rfloor \end{cases}$$

As an example, 2.4 has a 60% chance to round to 2, and a 40% chance to round to 3. Instead of directly mapping x to floor(x), here we map x to the unique value/bin which is closest to the original value and has the highest probability.

PSEUDOCODE FOR STOCHASTIC ROUNDING

```
1.func normal_rounding(model, precision):
2.      for layer in model.layers:
3.            min_val = get_min(layer) // get minimum weight value in the layer
4.            max_val = get_max(layer) // get maximum weight value in the layer
5.            num_bins = 2^precision
6.            bin_values = get_bins(min_val, max_val, num_bins) // get the values of all bins. This can be generated using different ways
7.            new_layer = None
8.            for weight in layer:
9.                  mapped_weight = get_highest_probability_match(weight, bin_values) // binary search for most likely bin value
10.                 new_layer.add(mapped_weight)
11.           layer = new_layer // cloning the layer with mapped values
12.     return model
```

- **Mid-Rise Rounding:**

This rounding technique is based on mid-rise quantization commonly used in signal processing for analog to digital signal conversion. It uses a parameter named delta which controls the granularity of the signal conversion. High values of delta would lead to significant loss of information whereas low values of delta would lead to near lossless signal conversion. We extend this idea to modifying our model weights post training. We divide the range of weights into equal sized bins which fixes the delta value. We then floor the value divided by delta and add 0.5 to it. The algorithm pseudocode is described below:

PSEUDOCODE FOR MID-RISE ROUNDING

```
1.func midrise_rounding(model, precision):
2.      for layer in model.layers:
3.            min_val = get_min(layer) // get minimum weight value in the layer
4.            max_val = get_max(layer) // get maximum weight value in the layer
5.            num_bins = 2^precision
6.            delta = (max_val - min_val) / num_bins
7.            new_layer = None
8.            for weight in layer:
9.                  mapped_weight = delta * (floor(weight / delta) + 0.5)
10.                 new_layer.add(mapped_weight)
11.           layer = new_layer // cloning the layer with mapped values
12.     return model
```

2.2.4 Granularity of Quantization:

- For the current set of models, we have focussed solely on weight quantization. We would be working with quantizing activation functions in the future
- We are performing per-layer quantization. As the distribution of weights varies across layers, this enables us to estimate the quantized weights effectively
- Currently, we are not considering per-channel quantization of weights
- For this analysis we are quantizing all layers of a given model using the same rounding technique and n-bit precision level

Normal rounding and Stochastic rounding techniques are dependent on the way unique values of weights are generated. Thus, we would be observing the performance of models for all possible combinations:

3 Weight Binning X 2 Quantizer Designs + 1 Mid-Rise Rounding = 7 Models/Dataset

3. Results

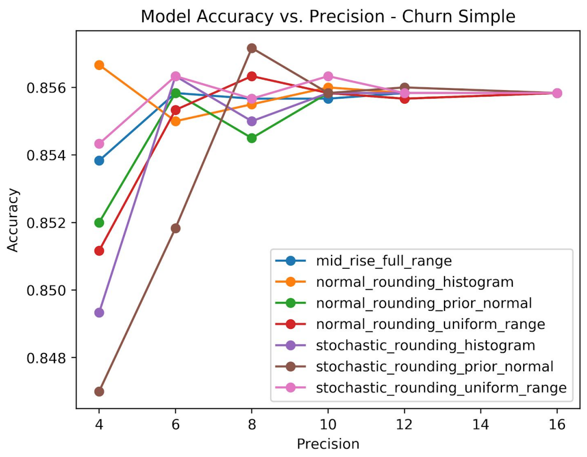
3.1 ANN-based Classification

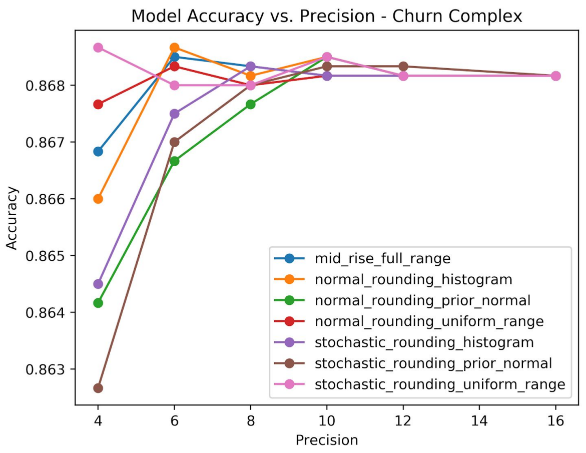
For classification, model performance is measured in terms of validation accuracy.

- **Dataset Used : Churn** (2 output Classes)

Model Complexity	Accuracy at Full Precision(32- bit)
Simple: 2 Dense Layers	0.8558
Complex: 5 Dense Layers	0.8682

Table 6: Full Precision Accuracy for Churn dataset based Classification Models





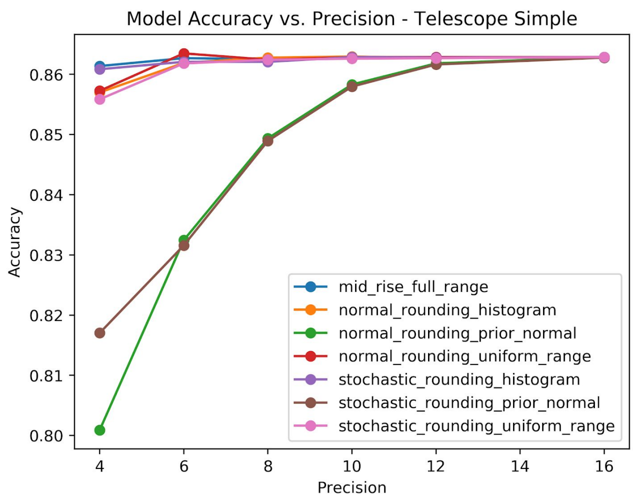
***The Precision on X-axis represents values in bits. N bit precision implies having 2^N unique possible values.**

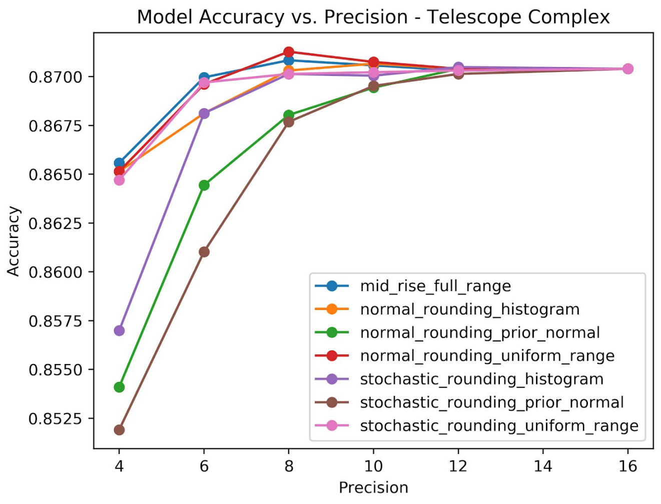
The above plots show the accuracy of simple and complex models on churn dataset classification, We notice that the accuracy drop is more pronounced for the complex model. This is because both the models fitted the data quite well with full-precision. Now when the precision decreases, the accuracy for the complex model decreases more as it has a larger number of parameters and higher weight variance.

- **Dataset used: Telescope** (2 Output Classes)

Model Complexity	**Accuracy at Full Precision(32- bit)**
Simple: 2 Dense Layers	0.8629
Complex: 5 Dense Layers	0.8704

Table 7: Full Precision Accuracy for Telescope dataset based Classification Models





The above plots show the accuracy of simple and complex models on telescope dataset classification. Similarly here, we find that the accuracy drop is more pronounced for the complex model. We find that stochastic rounding with uniform range is performing better on average than the other quantization methods.

3.2 ANN-based Regression

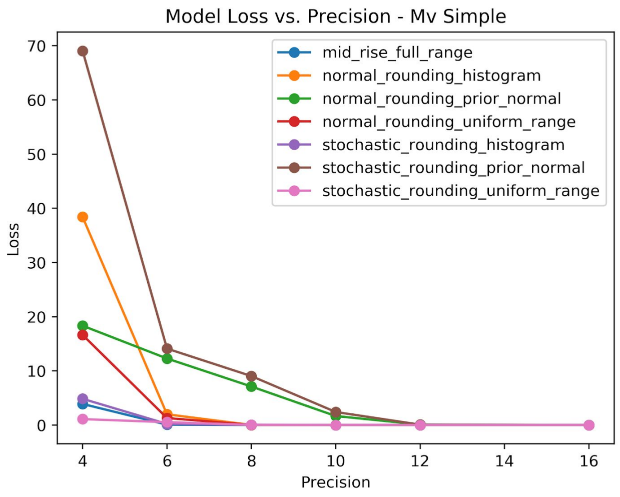
For regression, model performance is measured in terms of validation accuracy.

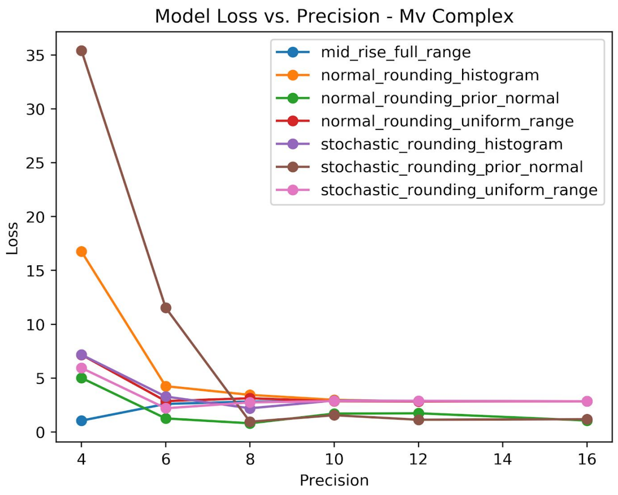
- **Dataset used: MV**

Model Complexity	**Loss at Full-Precision (32- bit)**
Simple: 2 Dense Layers	0.0146
Complex: 5 Dense Layers	2.8461

Table 8: Full Precision Loss for MV dataset based Regression Models

We notice that all methods other than histogram based rounding and prior normal stochastic rounding , other techniques seem to perform quite well on the regression model on MV Dataset. We also notice that the simpler model has higher loss value, which indicates it is slightly underfitted even without quantization. Another key observation that we got from here when evaluating the complex model is that quantization sometimes can be useful by helping us reduce the variance of the model and hence, behaving as a regularization technique. This is evident from the fact that a mid-rise quantized network at 4 bits of precision performs better than the ones at higher precision. This observation is particularly applicable when we have simpler datasets wherein a simpler model also performs well and hence, by increasing the complexity of the model we are increasing the chances for overfitting and hence, regularization in form of quantization would be particularly useful to reduce the overfitting.



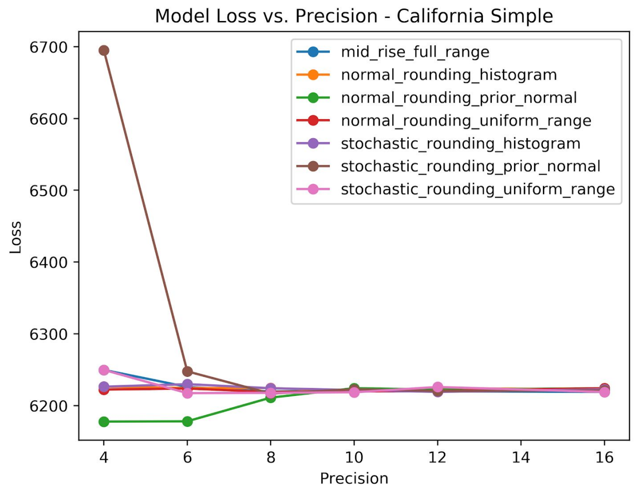


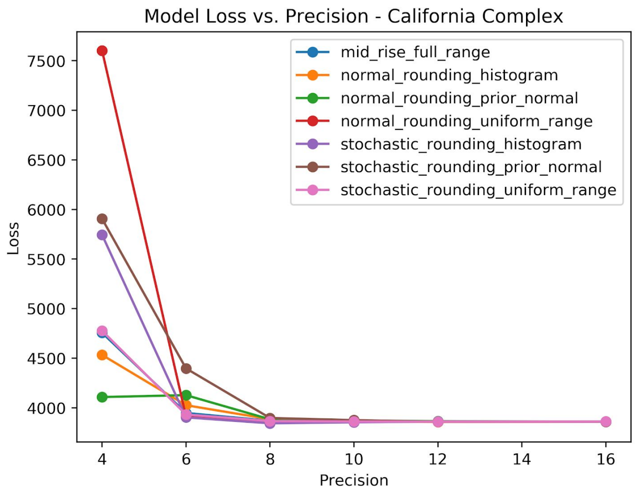
- **Dataset used: California Housing**

Model Complexity	Loss at Full-Precision (32- bit)
Simple: 2 Dense Layers	6218.87
Complex: 5 Dense Layers	3859.94

Table 9: Full Precision Loss for California dataset based Regression Models

We notice here that the validation is quite high even without the quantization, which shows that the model could be overfitted or under fitted. Further looking into the two plots we see that the complex model accuracy was more affected by the quantization which shows that the initial model was probably underfitted. Overall we notice that the regression models were more affected by the quantization process than the classification model. This could be because the classification problem had only 2-3 output classes.





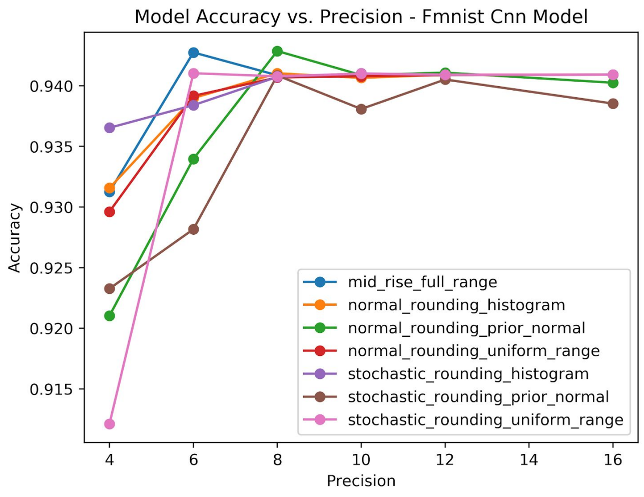
3.3 Convolutional Neural Networks

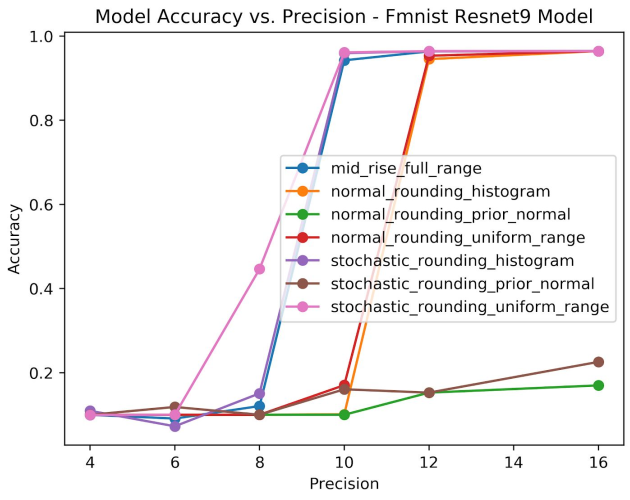
For CNN models, we would be observing the model performance in terms of accuracy at different precision levels.

- **Dataset used: F-MNIST**

Model Complexity	**Accuracy at Full Precision(32- bit)**
Vanilla CNN	0.9409
ResNet9	0.9641

Table 10: Full Precision Accuracy for F-MNIST dataset based CNNs





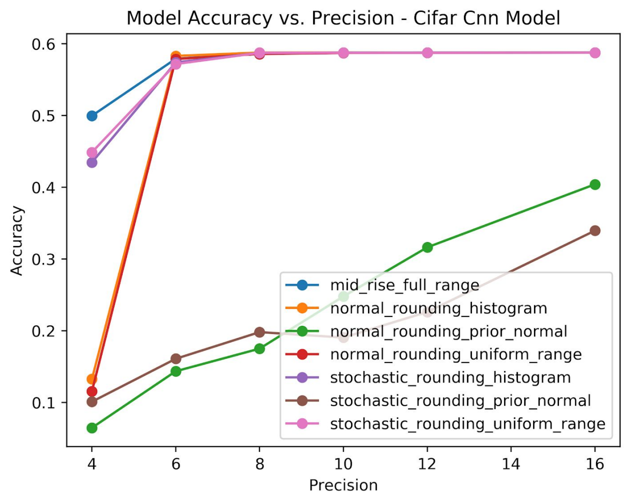
We observe that accuracy drops significantly for the ResNet9 at lower precision.This is evident as it has a large number of parameters and representing them at lower precision results in significant information loss. There is no significant drop in accuracy in case of Vanilla CNN as the number of trainable parameters is less and they could thus be represented at lower precision. In ResNet9, the stochastic rounding models maintain higher accuracy at 2 lesser bits of precision as compared to normal rounding models.

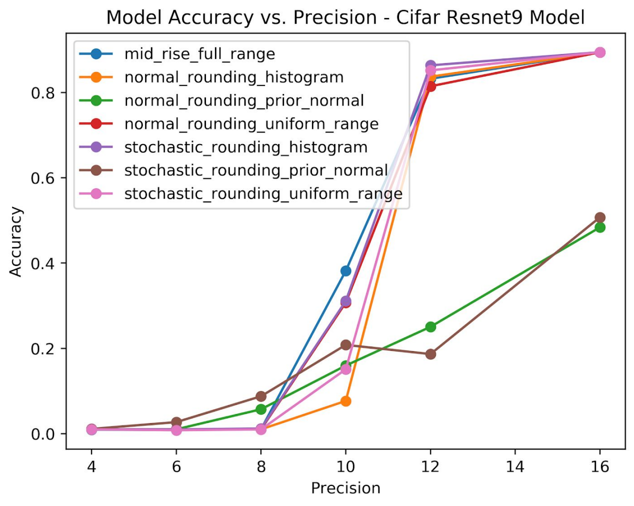
- **Dataset used: CIFAR-100**

Model Complexity	Accuracy at Full Precision(32- bit)
Vanilla CNN	0.5875
ResNet9	0.8947

Table 11: Full Precision Accuracy for CIFAR-100 dataset based CNNs

In the below two plots, we observe that for Resnet9, the accuracy drops much quicker as compared to simpler models as we reduce precision. This could be as ResNet has a larger number of parameters and quantization thus has more impact. Additionally, we observe that similar to the models discussed above, here also prior normal binning method performs poorly. In general, across both datasets we observe that stochastic rounding with uniform range is a better way to quantize the weights.





3.4 Key Takeaways

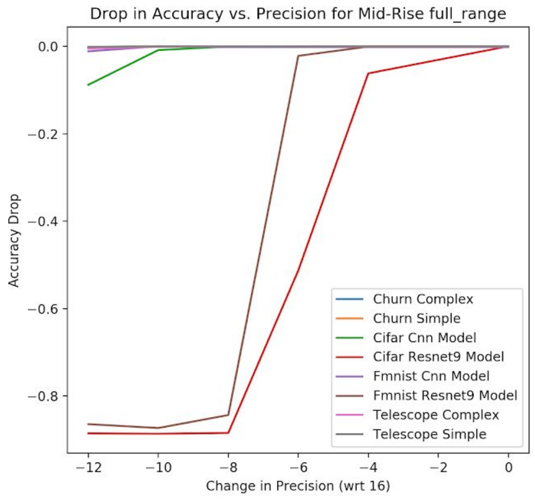
After having observed the tradeoff between accuracy and precision with respect to individual models and datasets, we now summarise the common findings/patterns across them:

- Performance of **Stochastic Rounding with Uniform Range** performs better in general
- However, the assumption of a **Normal Prior** is not an efficient way for determining the distribution of weights. This is evident from trends observed across all datasets/models.
- Full precision accuracy post quantization using best method:

  - In CNNs, complex models give full precision accuracy with **8-bit** precision whereas simple models give full precision accuracy with **12-bit** precision in general
  - In DNNs, simple and complex models give full precision accuracy with **6-bit** precision
- When we use the same quantization over models of different complexities and different datasets, model complexity affects the performance change majorly with decreasing precision.

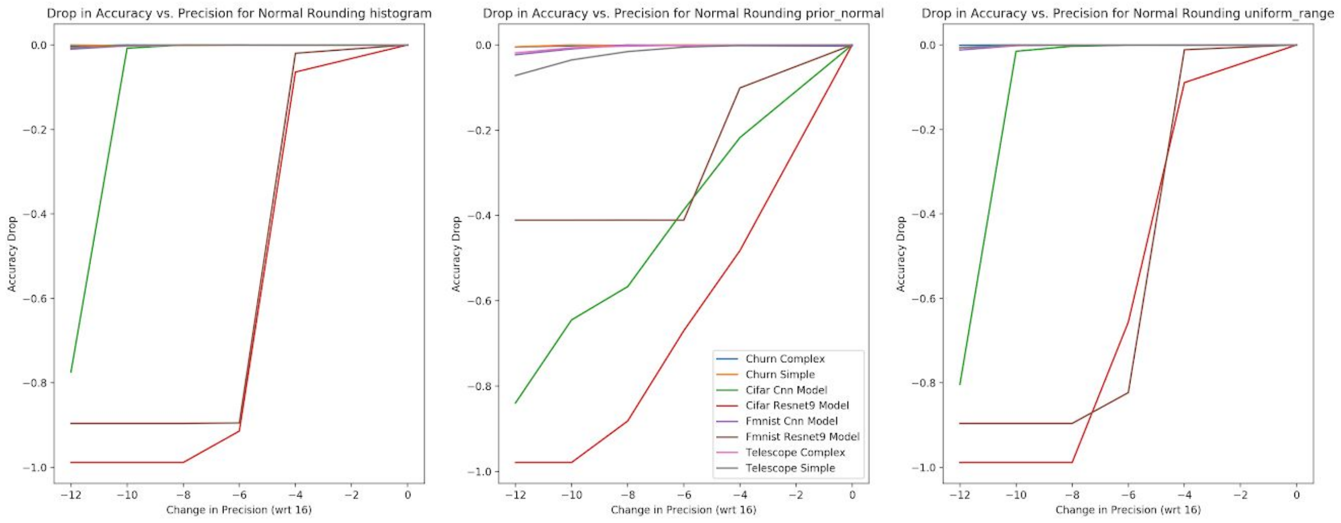
Mid-Rise Quantization:

  - ResNet 9 being the most complex one here is showing the most drop in accuracy



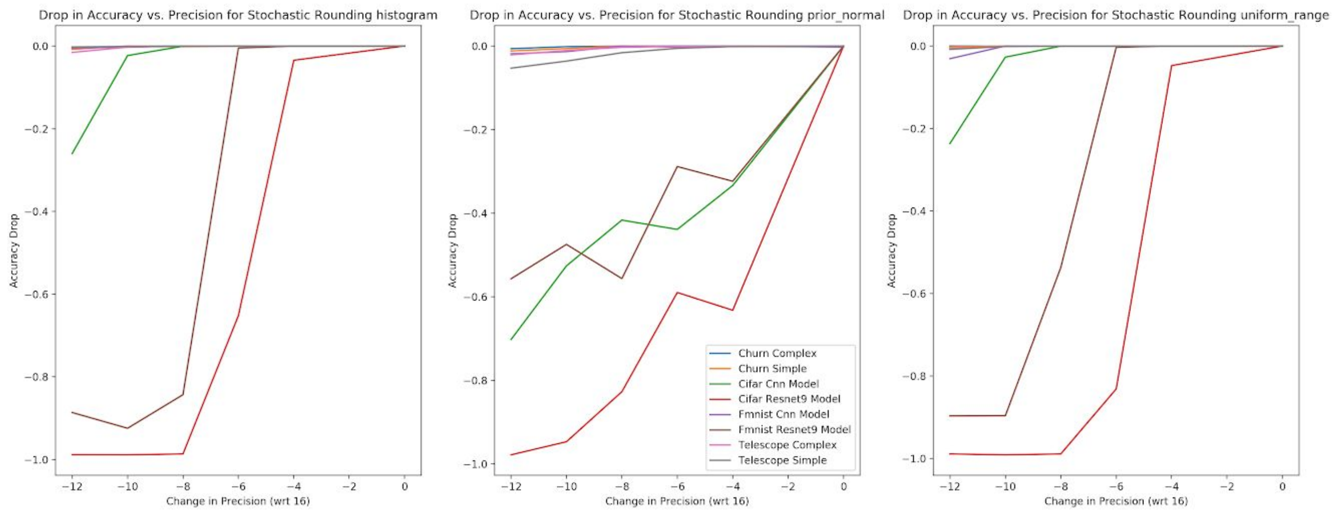
Quantization using Normal Rounding:

  - Rounding with distribution learnt from histogram and using uniform range behaves similarly
  - Prior normal assumption of weights worsens the performance with precision drop. Thus, we can evidently say a better prior should be used



Quantization using Stochastic Rounding:

- ○ Rounding with distribution learnt from histogram and using uniform range behaves similarly
- ○ Prior normal assumption of weights worsens the performance with precision drop. Thus, we can evidently say a better prior should be used
- ○ Also, stochastic rounding performs better than normal rounding (the dip in performance comes later)



- Change in **Dataset** keeping **Model Architecture** same : For CNNs, as the complexity of data increases in terms of number of classes and quality of images, CIFAR-100 vs F-MNIST, the model accuracy drops much faster as precision drops. For DNNs, the change in dataset does not influence the accuracy vs. precision curve much

- Change in **Model Architecture** keeping **Dataset** same: As the model complexity increases, the number of weights to be learned increases and we see a more significant impact of quantization on model accuracy.

3.5 Solutions to the listed Questions

3.5.1 Why does performance degrade with quantization?

- The change of weights on one layer affects the input for each of the subsequent layers. Thus, error propagates and get accumulated resulting in degradation of performance of deeper networks
- Weights vary in lesser magnitude and hence, maybe quantization is increasing the magnitude of difference between those
- If the learned weights of the model are noisy thus, giving lesser accuracy even with full precision, the quantization effect won’t be that significant (in terms of model performance)
- Initialization of weights can suit/deter certain quantization methods
- Information Loss due to low precision representation

3.5.2 Why do different datasets behave differently with quantization?

- With different datasets we have different underlying distribution of weights and hence, the magnitude of effect of quantization differs
- The natural decision boundary changes for each of the datasets and hence, model complexity changes which result in difference of effects caused by quantization functions (e.g.) the same architecture of the neural network can be used to learn a linear boundary as well as a radial boundary by just changing the weight
- With increase in the number of target classes, we require a more granular level of learning to assign the data points into different classes. More granular level needs more accurate weights giving us less flexibility for quantization

3.5.3 How to improve performance?

- Use of a better way to estimate the distribution of weights can help us determine the range more efficiently and hence the performance of rounding methods can be improved, for example: Using KDE to estimate distribution as compared to using a histogram
- Using more relevant priors based on the initialization functions can help us reduce the quantization weights
- Using dithering as a quantizer design. In terms of signal processing, in this process we add noise to a signal to mask the effects of quantization distortion

4. Next Steps

In addition to the methods discussed above for improving the performance of quantized models, we also discussed the following approaches and would now be focussing on their implementation:

4.1 Different levels of precision for different layers of a neural network

Based on this experiment, we realized that we could benefit from optimizing precision for each layer as the number of parameters are varied. For example, a layers could have 10^2 parameters and other layer within the same model could have 10^4 parameters. If we quantize them uniformly using 2^8 number of unique values, we are essentially keeping the same number of unique values in the 1st layer i.e. 10^2 and reducing only the 2nd layer to 2^8 parameters. Additionally, layers which have a higher number of parameters could benefit by keeping a relatively larger precision. Thus, as next steps we aim to achieve this quantization with budget constraints such that the total precision of the networks remains constant.

4.2 Quantization Aware Training

This technique is known to give the highest accuracy metrics compared to post-training quantization as the model learns quantized weights. QAT also quantizes the activation functions of the neural network nodes , thereby further reducing the number of bits used. We thus aim to evaluate and compare its performance across various networks.

5. Individual Contribution

- **Pritam Biswas** - Researched on post-training quantization techniques, quantization aware training, did POC for Neural Distiller library on pytorch, developed code for Pytorch Dataset loading for Regression and Classification Dataset, defined classes for Convolutional Neural Nets and Dense Neural Nets, wrote the code for training epochs, evaluation of accuracy and validation loss and logging of model results to log files.
- **Neelam Patodia -** Researched on post-training quantization, explored the functionalities of Neural Distiller library for quantization aware training, developed utility code for generating unique set of values for a given precision using different techniques and merging all quantization techniques
- **Mohit Chander Gulla -** Researched on techniques for post-training quantization. Wrote the initial code for various quantization methods - specifically mid-rise, normal rounding, and stochastic rounding. Generated model objects with weights at varying precisions for team members to conduct their experiments. Wrote the plotting code used for generating accuracy / loss vs. precision and change in accuracy vs. change in precision figures. Took on project management duties.
- **Kumari Nishu -** Literature review of post-training quantization methods. Developed code for Convolutional Neural Network of basic architecture, and models leveraging pre-trained models such as ResNet9 and ResNet50. Trained and evaluated these models for CIFAR100 and F-MNIST dataset. Implemented regular rounding and stochastic rounding functionality to be used for quantization. Evaluated the performance of CNN and ResNet9 model on CIFAR100 and F-MNIST dataset with different quantization methods and quantization precision
- **Prasham Dhaneshbhai Sheth -** Researched existing methods for post-quantization compression techniques as well as quantization aware training. Developed the architectures/codes for Artificial Neural Networks(simple and complex) to serve the 2 datasets for Classification and 2 datasets for Regression. Trained and evaluated the models along with the experimentation to evaluate the performance of these ANN models over different quantization methods with varying precision. Optimized the codes for regular rounding and stochastic rounding functionality to be used for quantization.

6. References

[1] Training a single AI model can emit as much carbon as five cars in their lifetimes - https://www.technologyreview.com/2019/06/06/239031/training-a-single-ai-model-can-emit-as-much-carbon-as-five-cars-in-their-lifetimes/

[2] Introduction to Quantization on PyTorch - https://pytorch.org/blog/introduction-to-quantization-on-pytorch/

[3] Post-training Quantization - https://www.tensorflow.org/lite/performance/post_training_quantization

[4] Distiller documentation - https://nervanasystems.github.io/distiller/index.html#what-is-distiller

[5] Raghuraman Krishnamoorthi (2018). Quantizing deep convolutional networks for efficient inference: A whitepaperCoRR, abs/1806.08342.

[6] Aojun Zhou and Anbang Yao and Yiwen Guo and Lin Xu and Yurong Chen (2017). Incremental Network Quantization: Towards Lossless CNNs with Low-Precision WeightsCoRR, abs/1702.03044.

[7] Suyog Gupta and Ankur Agrawal and Kailash Gopalakrishnan and Pritish Narayanan (2015). Deep Learning with Limited Numerical PrecisionCoRR, abs/1502.02551.

[8] Benoit Jacob and Skirmantas Kligys and Bo Chen and Menglong Zhu and Matthew Tang and Andrew G. Howard and Hartwig Adam and Dmitry Kalenichenko (2017). Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only InferenceCoRR, abs/1712.05877.

[9] Yunchao Gong and Liu Liu and Ming Yang and Lubomir D. Bourdev (2014). Compressing Deep Convolutional Networks using Vector QuantizationCoRR, abs/1412.6115.

[10] Wenlin Chen and James T. Wilson and Stephen Tyree and Kilian Q. Weinberger and Yixin Chen (2015). Compressing Neural Networks with the Hashing TrickCoRR, abs/1504.04788.

[11] Churn Dataset: https://www.kaggle.com/filippoo/deep-learning-az-ann

[12] Telescope Dataset: Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[13] MV Dataset: https://sci2s.ugr.es/keel/dataset.php?cod=84#sub1

[14] California Housing Dataset: https://sci2s.ugr.es/keel/dataset.php?cod=83#sub2