Exploring 1.58-bit Quantization in Vision Transformers

Pooja Desur pgd2120, Raman Odgers rmo2141

Summary

- Exploring 1.58 bit optimization in Vision Transformers through different quantization methods
  - Post training quantization - weights and activations are quantized after the model has been trained, only impacts inference
  - Quantization aware training - training the model with quantization constraints from the beginning
- Study and compare quantization techniques in terms of model performance, throughput and model storage
- Models: Vision Transformers e.g ViT, DEIT (here)
- Dataset: Imagenet (here), CIFAR10 (here)
- Analysis:
  - Metrics: model accuracy, model storage size, throughput
  - Tools: huggingface, pytorch, wandb

Problem Motivation

How is Quantization Helpful?

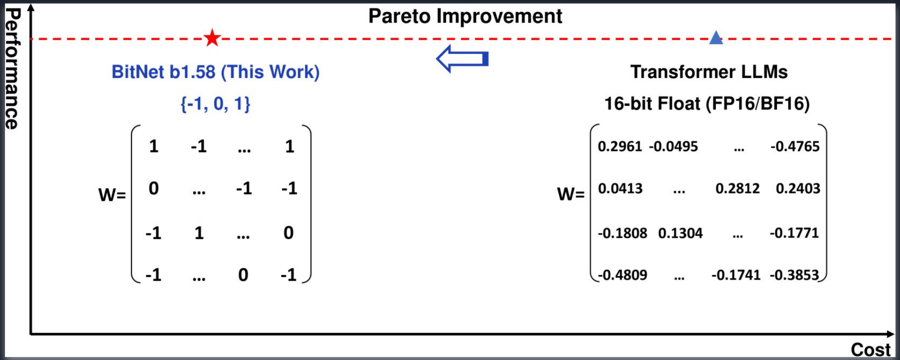
- Quantization helps to overcome the problems of limited computational resources and memory requirements without sacrificing the model's performance drastically
- The inference cost is reduced while maintaining model performance (increased throughput)
- Storage requirements for model weights are reduced by the quantisation factor.

Our Project

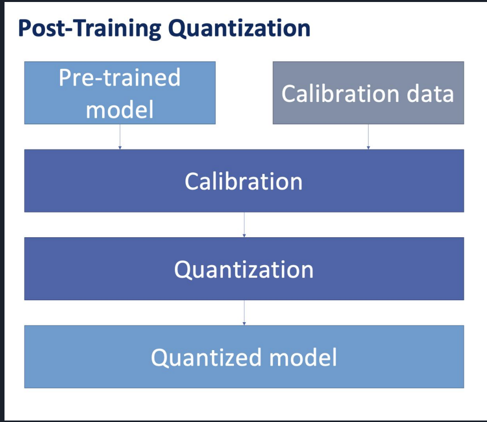
- Explore the various ways one bit quantization can be implemented in Vision models through post training and quantization aware training methods
- Create a Generalized framework to evaluate models through both methods of quantization
- Observe the performance of models and conclude if Vision models could benefit from 1 bit quantization

Background Work

- Project was inspired by the [1.58 bit quantization]() paper which quantizes each parameter to {-1,0,1} in LLMs
- Implementation of the paper is based on BitNet architecture which is a transformer in which each Linear layer of the model is replaced by a BitLinear method



- Additionally, this project builds off the existing literature in static post-training quantisation (S-PTQ) and Quantisation-aware training (QAT) when implementing our methods e.g ([here]()) and ([here]())



Technical Challenges

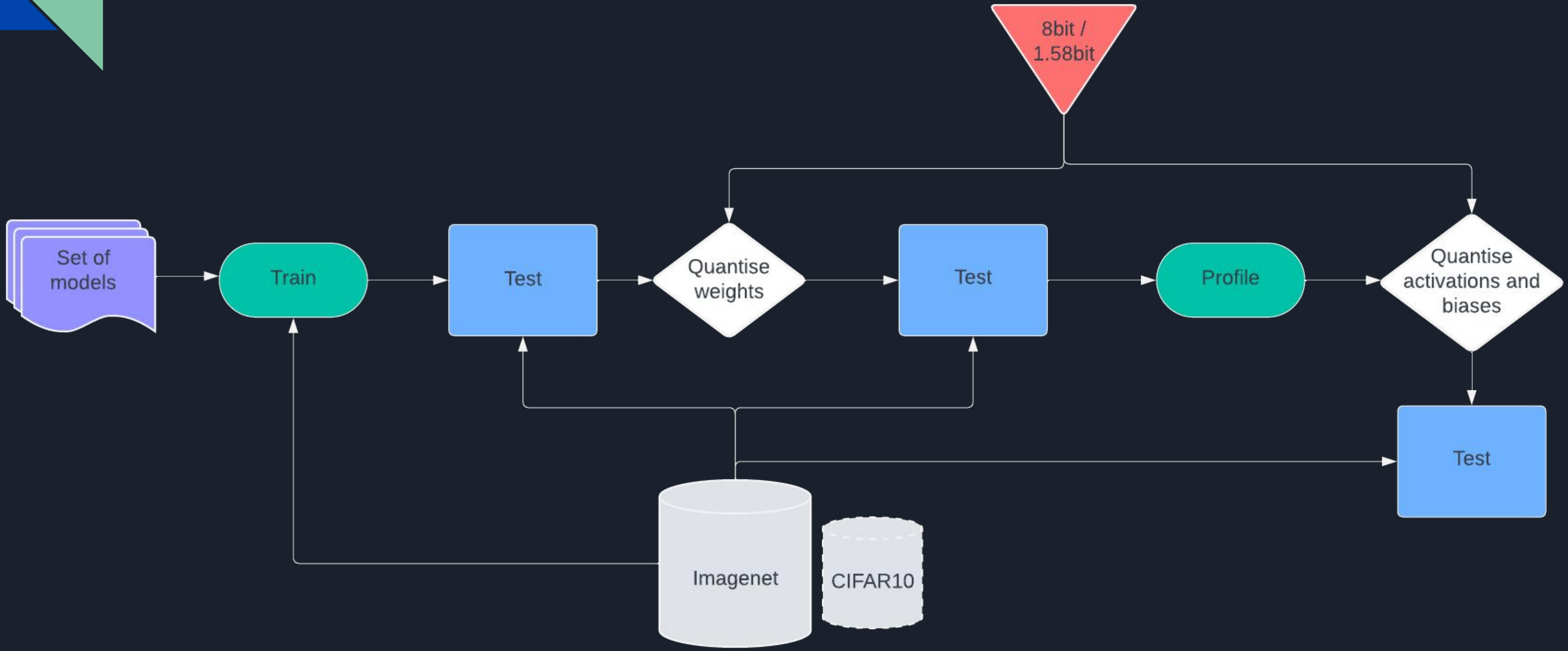
1. Triage on implementations of 1/1.58 bit quantization repos eg BitLinear
    a. Authors of the 1.58 paper do not provide official code, and mathematical representation is unclear.
    b. Bugs related to versions, transformer architecture etc
2. Failed exploration of Starcoder2
    a. As discussed in our midpoint progress report we spent a chunk of the semester failing to train the smallest starcoder2 model from scratch to generate performance baselines to compare our completed quantisation method with. This did not work for two reasons:
        i. GPU availability and model size: with limited GCP and terremoto availability training took far too long with the 3B parameter model.
        ii. Step function performance in code generation: For example, given how error-sensitive code correctness is
3. Scale of training barrage of models for QAT and classifier heads for PTQ

Approach

Post Training Quantization

- Using ‘covariate-shifted’ instantiation of Imagenet from GATE-engine
- Build generalisable 1bit PT-quantization package for transformer models
    - Functions can extract complex model architectures: nested modules, non-linear DAGs and return appropriate scales
- Ensure compatibility with huggingface imports and trainers
- Write robust, automated evaluation and comparison script to test performance
- Implement and test modifications to PTQ methodology
    - Asymmetric vs symmetric quantisation
    - Selective layer quantisation

PTQ Diagram



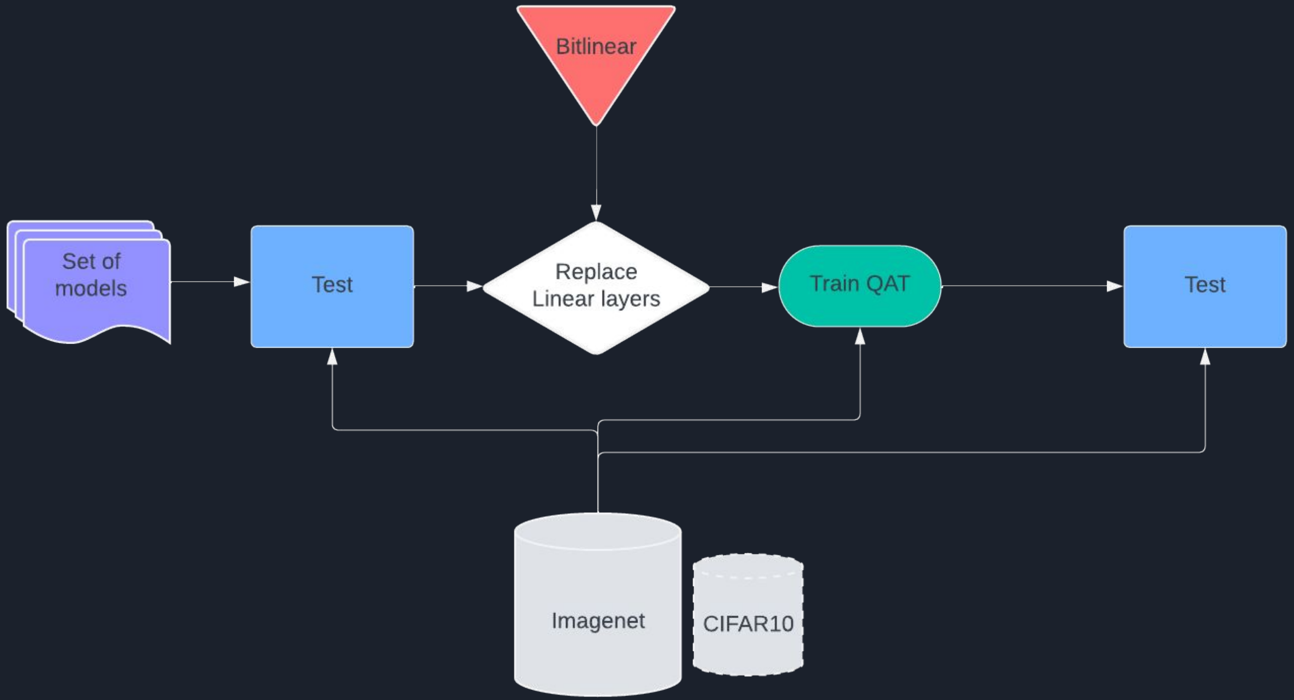
8bit /
1.58bit
Set of
models
Train
Test
Quantise
weights
Test
Profile
Quantise
activations and
biases
Test
Imagenet
CIFAR10

Quantization Aware Training

1. Train Vision Models **from scratch** after incorporating one bit quantization using BitNet library
   a. BitNet allows matrix multiplications to become addition operations
2. Replace each Linear Layer (in the Encoder and the Classifier), remaining architecture remains the exact same
3. Test hypothesis that 1-bit quantization in Vision models is useful
   a. Does it retain model performance?
   b. Does it increase throughput?

```
ViTForImageClassification(
  (vit): ViTModel(
    (embeddings): ViTEmbeddings(
      (patch_embeddings): ViTPatchEmbeddings(
        (projection): Conv2d(3, 192, kernel_size=(16, 16), stride=(16, 16))
      )
      (dropout): Dropout(p=0.0, inplace=False)
    )
    (encoder): ViTEncoder(
      (layer): ModuleList(
        (0-11): 12 x ViTLayer(
          (attention): ViTAttention(
            (attention): ViTSelfAttention(
              (query): BitLinear(in_features=192, out_features=192, bias=True)
              (key): BitLinear(in_features=192, out_features=192, bias=True)
              (value): BitLinear(in_features=192, out_features=192, bias=True)
              (dropout): Dropout(p=0.0, inplace=False)
            )
            (output): ViTSelfOutput(
              (dense): BitLinear(in_features=192, out_features=192, bias=True)
              (dropout): Dropout(p=0.0, inplace=False)
            )
          )
          (intermediate): ViTIntermediate(
            (dense): BitLinear(in_features=192, out_features=768, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): ViTOutput(
            (dense): BitLinear(in_features=768, out_features=192, bias=True)
            (dropout): Dropout(p=0.0, inplace=False)
          )
          (layernorm_before): LayerNorm((192,), eps=1e-12, elementwise_affine=True)
          (layernorm_after): LayerNorm((192,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
    (layernorm): LayerNorm((192,), eps=1e-12, elementwise_affine=True)
  )
  (classifier): BitLinear(in_features=192, out_features=1000, bias=True)
)
```

QAT diagram



Bitlinear
Set of models
Test
Replace Linear layers
Train QAT
Test
Imagenet
CIFAR10

Implementation

Dataset

- Mini ImageNet Dataset
  - Tested on CIFAR10
- Size: 60,000 images
- 1000 classes with 60 images of size 224x224 pixels
- Maintains class distribution of full ImageNet dataset
- Parquet Formatting leads to covariate shift and reduced pre-training performance.

Models

1. Vision Transformer (ViT) model
   a. Uses self-attention for feature extraction inspired by Transformer architecture meant for NLP
   b. 86 million parameters
2. Data-efficient Image Transformer (DeiT)
   a. Transformer based approach for image classification
   b. Uses knowledge distillation in training for data efficiency

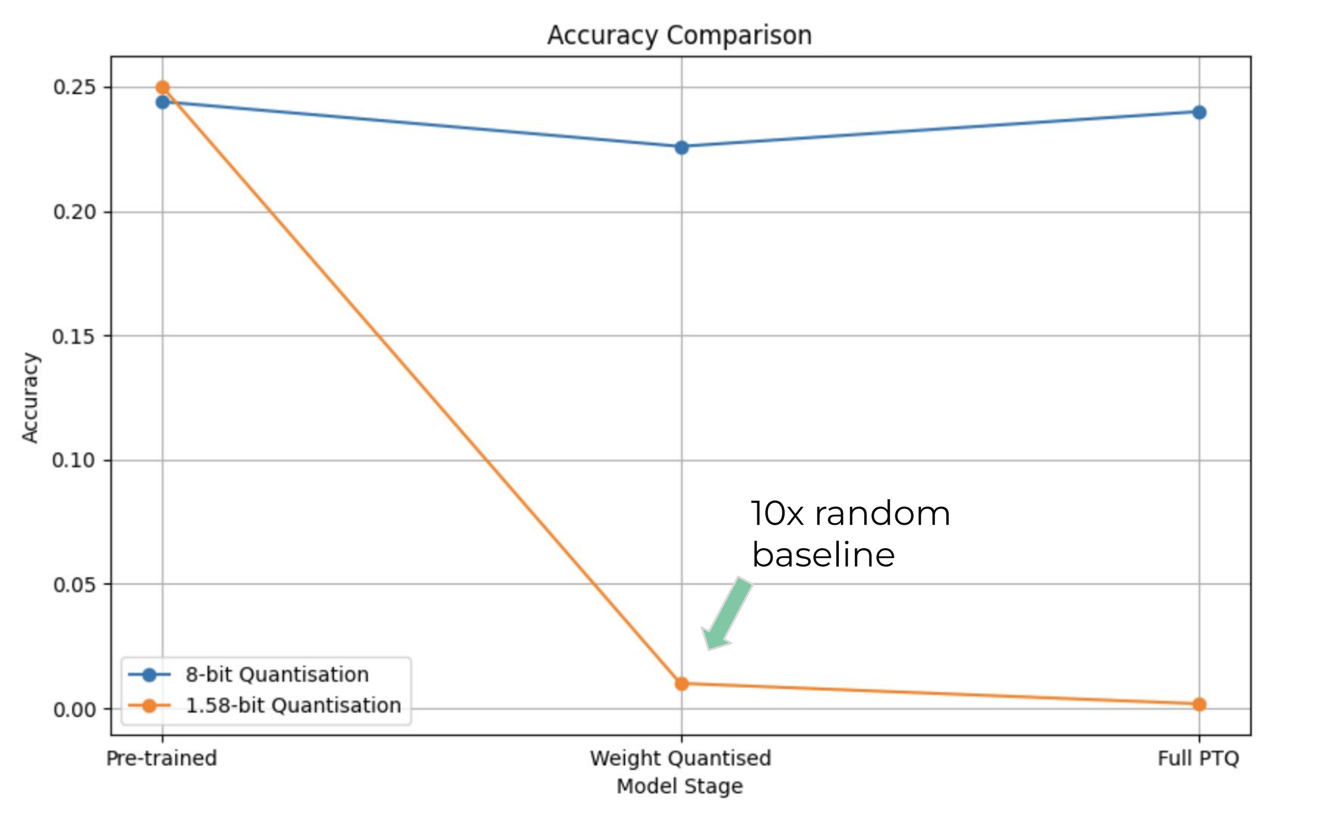
Evaluation

- Model Accuracy
- Quantization Error (difference between outputs of quantized model and original full precision model)
- Inference Time
- Model Storage

GITHUB

Experimental Evaluation

Post Training Quantization (Deit-Tiny)



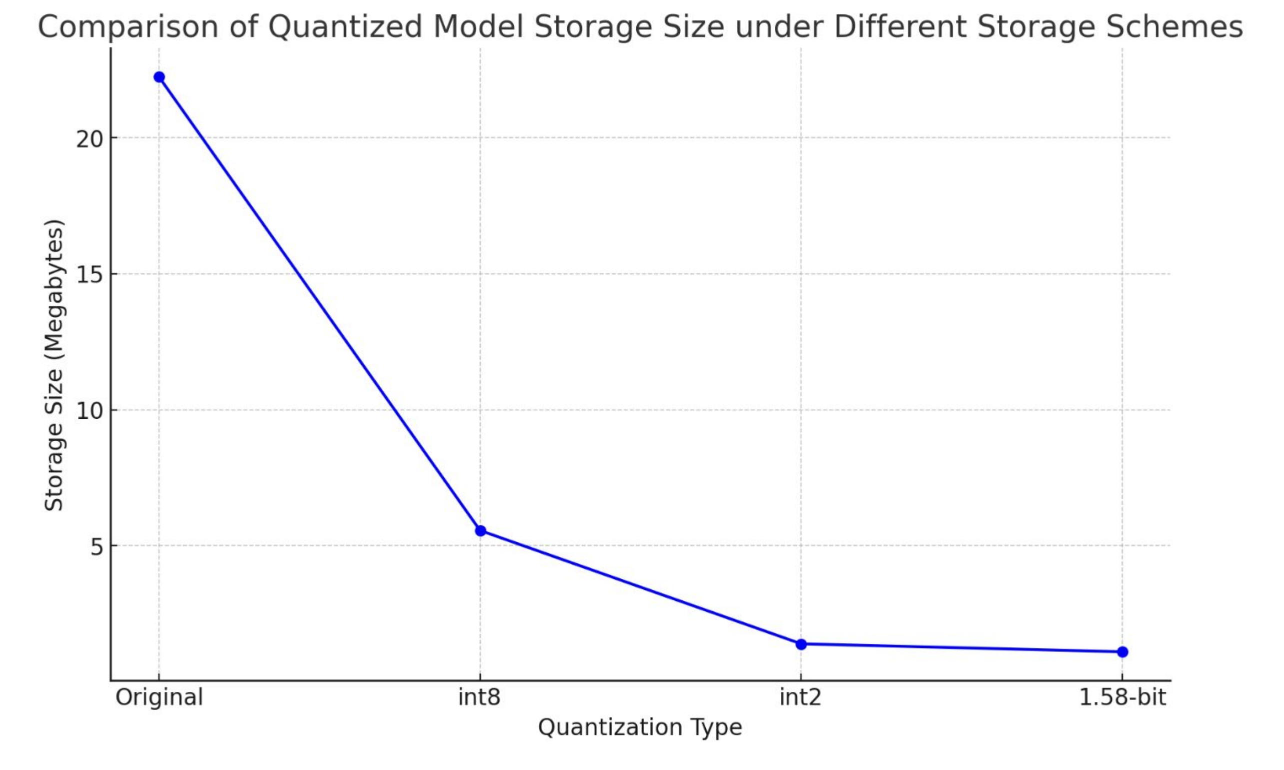
Quantization Aware Training

Model	Accuracy (without QAT)	Accuracy (with QAT)	Quantization Error	Throughput (samples per sec) (without QAT)	Throughput (samples per sec) (with QAT)
ViT-small	34.4	30.13	4.27	28.732	43.73
DeiT-tiny	33.28	28.35	4.93	73.306	154.342

- While the model performance does end up decreasing with QAT, we see that it retains the performance of the model much better than PQT and only drops by around 5%.
- The inference speed on the other hand almost is **increased by 2x** for both models when applying QAT

Model storage (DeIT)

- There is currently no datatype in native pytorch that can store below int8 precision. The datatype `torch.quint4x2` is physically stored as int8
- int2 or 1.58 storage would require updated hardware and software.



Conclusion

- 1.58 bit quantization is essentially inoperable for Post-Training Quantization.
- 1.58 bit Quantization-Aware Training retains some performance but has significant performance drops (much larger than reported for LLM/model scale in paper) probably due to the model sizes being much smaller.
- QAT improves throughput significantly (almost 2x)
- Hardware is not yet optimised to make use of reduced precision