AI&ML in Era of Climate Change - Assignment 1

LABBÉ Ugo, FADLI Firas, FRANCHI Charles

November 15, 2024

1 Introduction

This report focus on the applications and the pros and cons of Post-training quantization on Machine Learning models. We will talk about both Object detection models (ODM) and Large Language Models (LLM): their use, their cost and their efficacy - with and without quantization.

First of all, Object Detection Models are used to, as intended in their name, detect objects. To do so, ODM receive an image as input and output the bounding boxes and labels on detected objects.[1] They try to identify as many objects in an image as required, such as animals, utensils, cars and any other type of object that could be of use to detect. Some real life applications include ODM in security cameras (against theft or crowd counting), in cars (self-driving vehicles) or even in warehouse (for inventorying).

On the other hand, Large language models (LLM) are Machine Learning Models to manipulate and simulate language. They can use Transformers, Attention Mechanism [Vaswani, 2017] and language embeddings to have a deep understanding of languages. LLMs use text as input and output. Applications of LLMs could be Translation, texts analysis, text generation or more generally AI assistants.

In this report, we introduce the need of quantization and the challenge of ODMs and LLMs in section 2, then we detail the models, methods and datasets used in section 3, and present and discuss the results in section 4. We finally conclude in the section 5.

[1] What is Object Detection?.

2 Background

In this section, we will explain the need of quantization.

Nowadays, AI and precisely Deep Neural Networks are fashionable, and thus the use of this technology increase everyday. Because of this massive usage, the ecological impact of this technology have increased this past years.

Object Detection and Large Language Models are powerful mostly because they use an important number of parameters. In fact, the theory behind these models exist for several years or decades, but it is only thanks to the nowadays computational power that we can train and use these models in an interesting way. This computational power come with the cost of the energy needed : in the context of Global Warming, it is important to measure this ecological impact and to try to reduce it.

Also, even with the progress in hardware, the training of these models needs hours, days or even months to have a great accuracy, and it is important to find methods to overcome this problem for energy saving, but also for a practical usage.

Moreover, even inference after training can suffer from the size of the model : we sometimes need to have online usage of these models, like for ODMs and autonomous cars. Using smaller models could theoretically help speeding up these methods - as well as reducing the energy cost of using them.

Model quantization is a fresh method to address these drawbacks. This method reduces the size of the models and so the computa-

tional cost in time and energy of Deep Neural Networks. The main idea is to change the data type of the weights of the Network, going from floating numbers to integers[2], and/or reducing the numbers of bytes (going from 32 to 16 or even less bytes). There is two main types of quantization : Pre-training quantization, and Post-training quantization.

- Pre-training quantization, or quantization aware training[3], reduces the size and thus the computational complexity of the model using less precision (16 or 8 bytes) during training, or removing some redundant layers of a model if it is possible.

- Post-training quantization do the same but after training, so the model is reduced afterward, decreasing is storage size and theoretically speeding the inference computation.

Over the past years, several technics have been developed to perform quantization, and it is now a method implemented in several well-known machine learning software, such as TensorFlow.

In this report, we focus on the impact of the quantization at inference time, and thus on Post-training quantization. We look into two different models with different purpose, and analyze the impact of Post-Training quantization.

3 Experiments

To have a clear idea of the performance that can achieve quantization, we experiment it on two different models : one ODM and one LLM. For both, we test several size reductions and compare their performances on well-known datasets. We perform our tests using python TensorFlow environment in Google Colab. The hardware and software detailed can be find in appendix A.

It is important to notice the choice of the original model was highly affected by the hardware limitations. We tested several of them before choosing the ones we present further. For example, we try to use EffecientDet, a newer ODM that was just too heavy to use.

3.1 ODMs

For Object Detection, we compare 3 models : one with floating numbers in 32 bytes weights (float32) - which is the original model -, one with floating numbers in 16 bytes (float16), and one with mostly 8 bytes integers (int8).

To do so, we use the TensorFLow Lite module of TensorFlow, which allow a Post-training quantization of Keras model.

Float16 model was made using a static method, converting all the weights in float16 but, due to hardware limitations, still computing in float32 (and thus doing some conversion before computation). For the 8 bytes models, we use a dynamic method : the quantization convert all weights in int8, but if it faces some issues during computation regarding the integers type, it falls back into float32 computation.

3.1.1 Model

We test the quantization on the CenterNet Object detection model, which use the ResNet-v2-50 backbone. This model was trained using the COCO2017 Dataset (see next subsection). For that reason, it makes the evaluation of the quantization easier and clearer. The images were resized in a 512×512 pixels format for training, but we can use any image size for inference.

This model was designed in a way to detect "the most prominent objects in an image", and will thus failed for specific or critical tasks such obstacle detection. Also, the model can't use batch.

3.1.2 Dataset

To evaluate our Object detection models, we use the COCO2017 dataset.

The Common Objects in Context (COCO) dataset is a well-known dataset for object detections and image classification. The data

[2]Integers operations is faster to compute at the hardware level than floating numbers.

[3]Quantization aware training

focus on 80 objects in a lot of different everyday life application. There is two versions : 2014 and 2017. The difference between both is mostly the train/test split. This dataset is known and use for its clear and wide annotations on every pictures, like box coordinates and precise true boundaries.

However, the COCO dataset is often criticize because it is highly imbalanced between different objects categories, which lead to an inconstancy regarding model trainings.

For our project, we evaluate the model we quantized on the validation set of COCO2017 Dataset.

3.1.3 Evaluation

For ODMs, we evaluate our quantized models with three metrics : the model-size in Megabytes - to see how well we compressed the model-, the average inference speed i.e. the times to process one picture, and the Mean Average Precision (mAP)

More precisely, Average Precision metrics is the mAP@50, measuring precision at an Intersection over Union (IoU) threshold of 0.50. The precision measures how many detected elements were actually relevant (true positive over the amount of detections). The IoU measures how much detection boxes overlaps between their ground truth and what the models found. If the IoU is 1, the boxes are on top of each other.

$$\text{mAP@50} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i(0.5)$$

where:

- N is the total number of object categories.
- $\text{AP}_i(0.5)$ is the Average Precision for the i-th category at an IoU threshold of 0.5.

3.2 LLMs

For Large Language Models, we compare 4 models : the original one, and 3 quantized models with the weights in integers 8 bytes (int8), 4 bytes (int4) and 2 bytes (int2).

To do so, we use the Auto-gptq python library. To quantize the model, the library apply the GPTQ algorithm (Quantization for Generative Pre-trained Transformers) [Frantar et al., 2022], which quantized each row of weights independently so they minimize the final error. However, for the GPU computation, the model convert the weights in floating numbers 16 bytes in live.

3.2.1 Model

We use the facebook/opt-125m Model to test the quantization. This model is a Open Pre-trained Transformer Language Models (OPT) [Zhang et al., 2022], which are decoder only model. It has only 125 million parameters, which make it a light model regarding other LLM. This model are pre-tained on mostly english text, and are intend to be used on generative tasks.

A main disclaimer on the model is that it is strongly biased due to the internet database used for pre-training the model, which is not a neutral context . It is not supposed to have any impact on the comparison on quantization though.

3.2.2 Dataset

The LAnguage Modeling Broadened to Account for Discourse Aspects (LAMBADA) [Paperno et al., 2016] dataset is composed of 10,022 narrative english passages where the last word of the last sentence of the extract is the target. All extracts was human control to be sure it is possible to guess the last word target with and only with the context of the all extract (and not just the last sentence).

Some drawbacks of the LAMBADA dataset are first of all the specific fiction data, which thus limit its use regarding more real-life application, such as technical topics. Also, the different size length of the passages can be seen as an advantage or a drawback: it can be useful to see the length independence of the accuracy, but as the length is not equally distributed regarding the number of examples, it does not help a consistent learning. Finally, some criticism can be raised about the human subjectivity of the prediction of the target, making some examples less relevant regarding the prediction expected.

3.2.3 Evaluation

To evaluate the quantization on LLM, we use 3 metrics: the model-size in Megabytes - to see how well we compressed the model-, the average number of tokens per second, which can approximately be seen as the number of words generates in one second, and the top-k accuracy metrics.

The top-k accuracy metrics is a mean average of a 0-1 accuracy that checks if the expected target word is in the k-higher probability words of the models output. Thus, the top-1 accuracy check if the target word is exactly the predicted output of the model and is similar to an error counting function.

To compare our models, we use the top-1 accuracy as an arbitrary choice.

4 Results

4.1 Object detection models

4.1.1 Model size

After quantization, the different models where saved under the tflite format. We could then see how the model size (in MB) differed between the 3 types of quantization.

	Float32	Float16	8bit
Megabytes	96.33	48.23	24.57

Table 1: Model size (in MB) for the different quantization formats

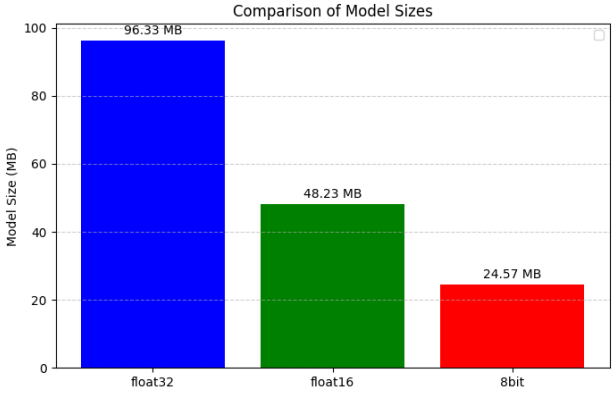

Figure 1: Model size (in MB) for the different quantization formats

The model size follows almost perfectly the weights division, their size being half the one of the preceding model.

4.1.2 Average Precision

Results can be found below, also including the mAP@50:95 and mAP@75.

	Float32	Float16	8bit
mAP@50	0.152	0.151	0.143
mAP@50:95	0.077	0.076	0.069
mAP@75	0.068	0.068	0.059

Table 2: Evaluation results for the different quantization formats.

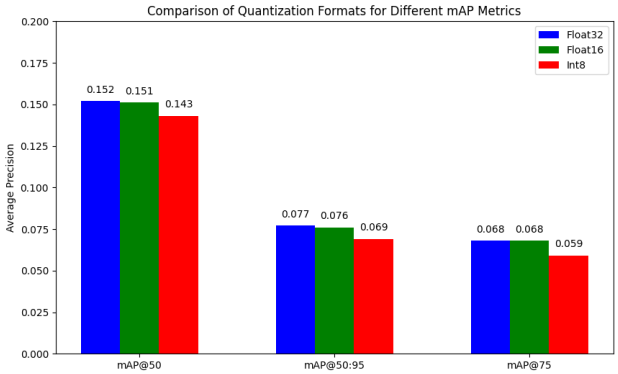

Figure 2: Comparison of mAP Metrics for Different Quantization formats

The result difference is very slight between the Float32 and Float16 models while it starts to dip a bit for the 8 bites one. The quantization is thus pretty good because we keep the global accuracy of the model.

4.1.3 Inference time

To measure how fast our different quantized models were, we took the time it took for the ResNet to give us its predictions on multiple images. In the mean time, the image size (width multiplied by height, in pixels) was stored in ordered to see if it mattered for the models.

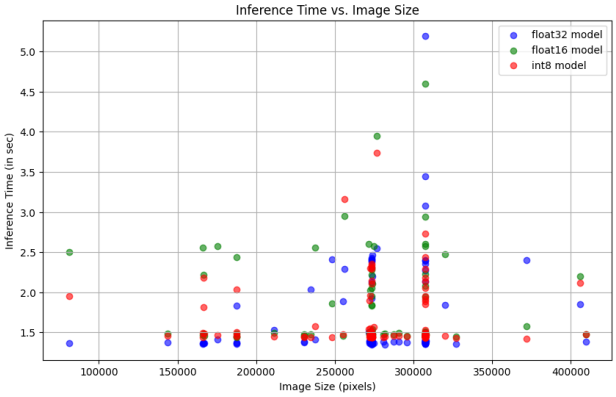

Figure 3: Comparison of Inference time per model vs image size

It's also important to note that the inference time highly depends on the environment, as after trying multiple times results can range from an average of 1.1 seconds to almost 2 seconds. Overall it is hard to say there is a significant difference between the models even though the float16 seems to take slightly more time the float32, which is a surprising results. We learned that intel CPU (such as the one used on our Google Colab environment) support float16 storage but convert to float32 before computation, which could explain why float16 ends up not being faster than the non-quantized version: https://huggingface.co/docs/optimum/concept_guides/quantization.

The 8 bites model however was consistently faster than the 2 others, but not by a significant margin.

4.2 Large Language models

4.2.1 Model Size

Now that we have quantized models, we evaluate them to observe how the model size (MB) changed between the 32-bits, 8-bits, 4-bits and 2-bits quantization levels.

	32-bit	8-bit	4-bit	2-bit
Model Size (MB)	477.75	38.36	19.18	9.59

Table 3: Model size (in MB) for different quantization levels

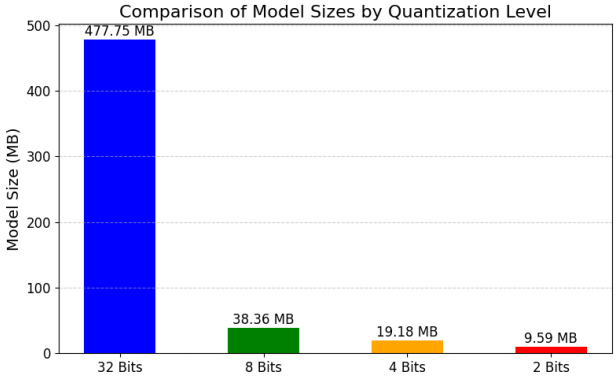

Figure 4: Model size (in MB) across quantization levels

The model size significantly decreased with each lower bit width. Quantization from 32 bits to 8 bits reduced the model size by roughly 90% and we'll see that the efficiency is almost the same in term of accuracy.

4.2.2 Top-1 Accuracy

Quantized models were evaluated on their top-1 accuracy using the LAMBADA dataset. Results are shown in Table 4, with accuracy decreasing as quantization educed bit width.

	32-bit (Baseline)	8-bit	4-bit	2-bit
Top-1 Accuracy (%)	17.40	17.30	15.00	0.40

Table 4: Top-1 Accuracy (%) for different quantization levels on the LAMBADA dataset.

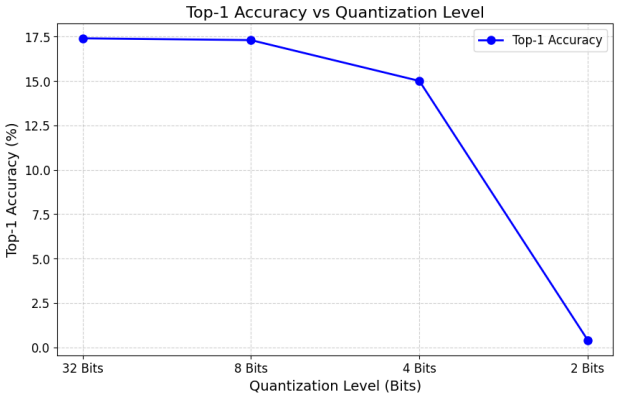

Figure 5: Top-1 Accuracy across different quantization levels

The result shows a minor accuracy for 8-bits quantization compared to the baseline. However, accuracy drops significantly at 4-bits and there is almost none at 2-bits. It

indicates that too much quantization impacts too much the model performance as we could have expected. But the baseline and the 8-bits quantization has nearly the same accuracy for a large size reduction.

4.2.3 Speed

The speed is measured in tokens per second, it was tested across all quantization levels. The speed for each models is in Table 5.

	32-bit (Baseline)	8-bit	4-bit	2-bit
Tokens/s	10,838.90	3,093.12	3,133.15	3,086.29

Table 5: Speed (tokens per second) for different quantization levels

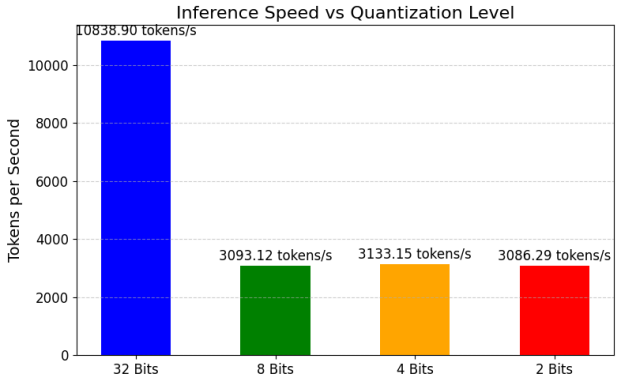

Figure 6: Inference speed (tokens/s) across different quantization levels

Speed was highest with the baseline 32-bit model and decrease then to roughly the same speed for each of the quantized models. This is not what we would expect from quantized models that would have a better speed since they use less memory. But it seems that it's because of poor efficient processing for low-bit precision because of the hardware used.

5 Conclusion

Quantization is a powerful tool that can help decrease the impact of both training and inference between model. In this report, we show that using small post-quantization do not have a significant impact on the model accuracy and can decrease the size of the model, which can be useful fo really big model to be embedded in small hardware, like for security camera related to autonomous cars. However, even if it could theoretically improve the speed, the hardware limitations of computation can often be a barrier to have a significant improvement. In the future, having a better compatibility between hardware and quantization could help achieving the goal to balance the global warming impact of AI.

Finally, quantization is a promising method that should gain in popularity in the Machine Learning Community.

A Hardware and Software specification

Google colab hardware on November 5th 2024 :

- 2 CPUs Intel(R) Xeon(R) CPU at 2.00GHz
- TPUs NVIDIA-SMI 535.104.05, Tesla T4 with 15360MiB VRAM - Driver Version: 535.104.05

Software use (November 5th 2024):

- Python version: 3.10.12
- Numpy version: 1.24.3
- TensorFlow version: 2.17.0
- CUDA Version: 12.2
- Optimum version: 1.23.3
- Auto-gptq version: 0.7.1

References

E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

D. Paperno, G. Kruszewski, A. Lazaridou, Q. N. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

A. Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.