International Journal of Information Technology (IJIT)
Volume 4, Issue 01, January-June 2023, pp. 58-64 Article ID: IJIT_4_01_007
Available online at https://iaeme.com/Home/issue/IJIT?Volume=4&Issue=1
Journal ID: 4573-3410, DOI: https://doi.org/10.17605/OSF.IO/NRHBZ

MODEL OPTIMIZATION OF GENERATIVE AI MODELS: MAKING AI FASTER

Surabhi Sinha
Department of Computer Science, University of Southern California, Los Angeles, USA

Mayukh Maitra
Department of Computer Science, Stony Brook University, New York, USA

ABSTRACT

The field of deep learning has been advancing at a breakneck speed recently, and one can now find examples of its use in every facet of present-day human existence. It is now absolutely necessary to optimize these models in such a way that they can be put to use in real life since every month hundreds of new models are introduced and their capabilities continue to advance with each new model. Most of the time, modelers find it challenging to productionalize their models due to constraints imposed by the hardware or a low throughput rate brought on by models that have not been optimized. In this work, we discuss some of the architecturally enhanced algorithms and innovative approaches such as dynamic networks along with ML frameworks enabling deep learning model optimization across various platforms and devices.

Keywords: Dynamic networks, DNN, OpenVino, TFLite Macro, FLOPs

Cite this Article: Surabhi Sinha and Mayukh Maitra, Model Optimization of Generative AI Models: Making AI Faster, *International Journal of Information Technology (IJIT)*, 4(1), 2023, pp. 58-64.
https://iaeme.com/Home/issue/IJIT?Volume=4&Issue=1

INTRODUCTION

Deep learning has seen a number of notable breakthroughs over the past few years, which have had a considerable impact on the inference speed of models. The rate at which models may be trained and put into use has significantly accelerated in recent years as a result of the proliferation of cutting-edge hardware and software technologies. This has made it possible to produce more efficient and accurate predictions, which in turn has enabled organizations to improve their decision-making processes and their operations as a whole. It is an exciting time in the field of artificial intelligence and machine learning since deep learning is continuing to evolve, and we may anticipate seeing even larger advances in the speed at which models draw conclusions. With the advent of generative AI, limitless potentials have been unearthed for one to explore. However, with impressive capability comes increasing complexity, and issues such as latency get even more critical to be resolved in order to utilize the best out of such models.

There are various techniques that have evolved over the years for reducing model latency. Architectural and design changes in deep learning models to perform algorithmic optimizations have been one of the most effective ways to reduce model size such as MobileNet [1] which is built on a lightweight and streamlined architecture that utilizes convolutions that can be separated depth-wise. Additionally one can develop smaller architectures such as SqueezeNet [2] with fewer parameters by identifying high-performing DNNs [3] which can achieve comparable accuracies to more complex networks with far fewer parameters thereby being more compressed and fast. One can also combine multiple techniques such as model pruning, quantization, etc. to optimize models for latency similar to the deep compression pipeline [4]. In addition to this, dynamic adaptation of deep learning models is an emerging field in the domain of model optimization. Dynamic networks are able to modify their structures or parameters in response to varying inputs, which results in a number of significant benefits in areas such as accuracy, computing efficiency, adaptability, and so on [5]. Deep learning models are not just limited to research and are currently omnipresent. From ML-enabled websites and services to wearable devices as small as a wristwatch, all utilize the power of AI in some form or other and it is highly critical for the models to be highly responsive for practicality and a better user experience. One such work describes the utilization of wearable hardware resources in an effective manner as the primary focus in order to lessen the impact of latency and make real-time vision applications possible [6].

In addition to hardware and architectural changes, there has been significant development in the field of deep learning frameworks specifically catering towards reducing latency. In this work, we discuss a few of the model optimization frameworks and also see how they can reduce latency as compared to baseline Pytorch models. Efforts to reduce latency will be essential to the establishment of scalable, accessible, and real-time solutions as deep learning continues to progress and find applications in a wide variety of fields.

OPTIMIZATION FRAMEWORKS

Model optimization frameworks make it possible to fine-tune and optimize deep learning models in accordance with certain tasks or hardware platforms. These frameworks provide a variety of optimization techniques, such as pruning, quantization, and compression, that can be used to improve the performance and efficiency of models in production situations. These frameworks can also be used to reduce the amount of space required to store model data. Developers can reduce the computational and memory needs of their models by making use of these frameworks, which, in turn, results in faster inference times and cheaper hardware costs. In general, model optimization frameworks are an important tool for ensuring that deep learning models are optimized for their intended use cases and can be deployed effectively in production situations. This is accomplished by ensuring that the models can be deployed effectively in production contexts.

OPENVINO FRAMEWORK

Intel has developed an open-source toolset called OpenVino [7] that may be used for the purpose of optimizing and deploying deep learning models. It has the ability to improve both the performance and the efficiency of models on a variety of different hardware platforms. You may also deploy OpenVino converted models to cloud instances that have specialized Intel CPUs and would be very efficient for performing inferences on these optimized models. Figure 1 describes a typical pipeline for converting a Pytorch model to OpenVino. OpenVino conversions are done in float FP16 data type which also helps in model compression by halving the default float32 numeric precision.

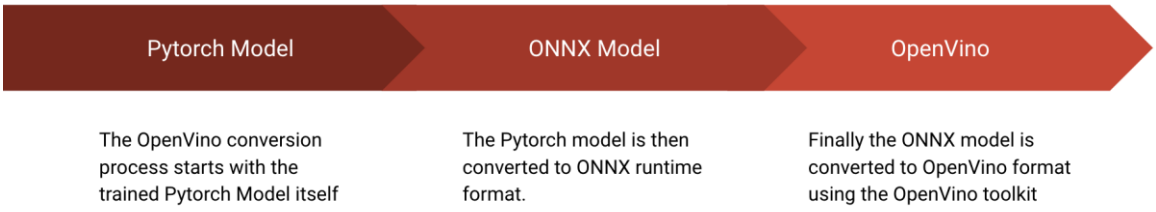

Figure 1: Pytorch to OpenVino conversion pipeline

In this work, we have performed experiments on various Pytorch pre-trained models and compared the inference times of default Pytorch models to that of OpenVino. We utilized 6 default pre-trained models for our experiments. In Figure 2 we can see that OpenVino significantly outperforms Pytorch when it comes to latency. From the experiments, we can see that on average OpenVino reduces inference time by at least 4 times when compared to Pytorch.

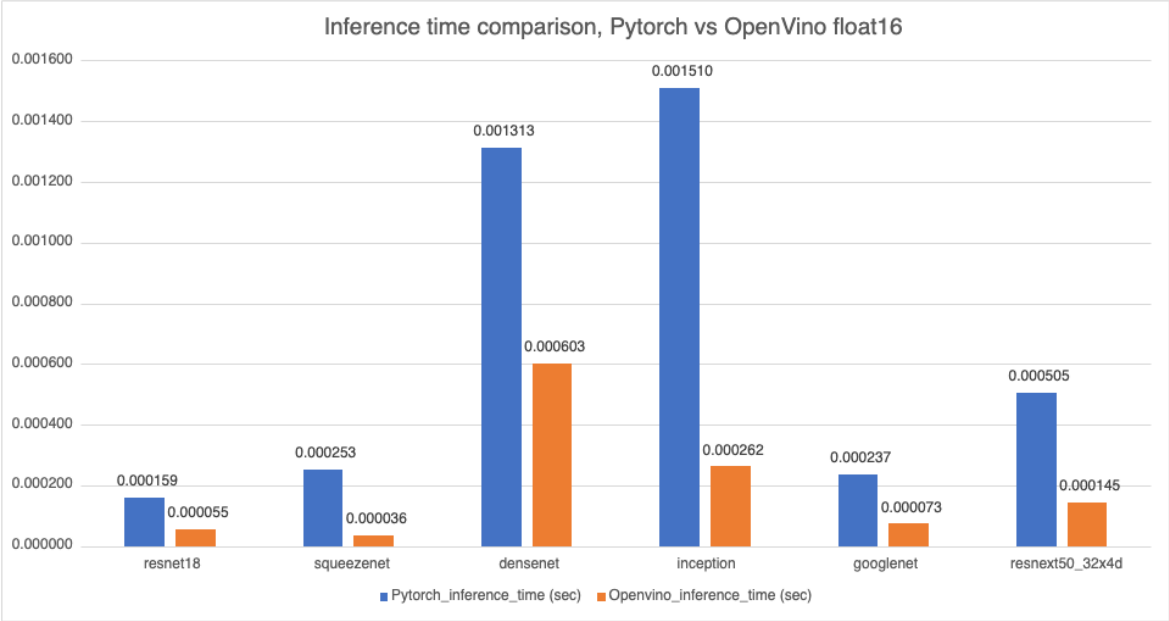

Figure 2: Inference time comparison between Pytorch and OpenVino

In addition to comparing inference times across models we also compared how inference time varies over multiple iterations for Resnet18 [8] and DenseNet [9] models as shown in Figure 3 and Figure 4 respectively. We can observe that OpenVino consistently outperforms Pytorch inference and is much more consistent as compared to the latter.

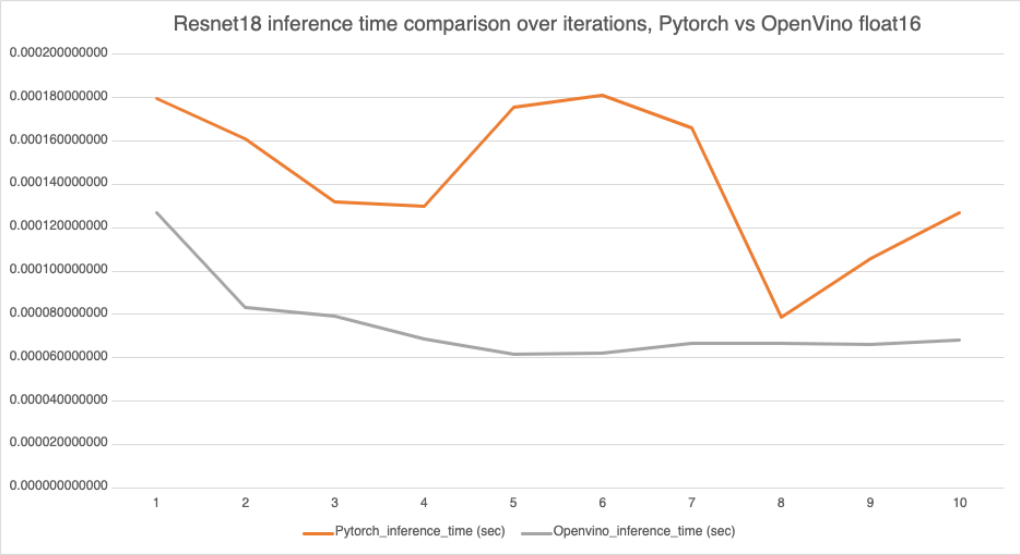

Figure 2: Inference time comparisons across iterations between Pytorch and OpenVino for ResNet18

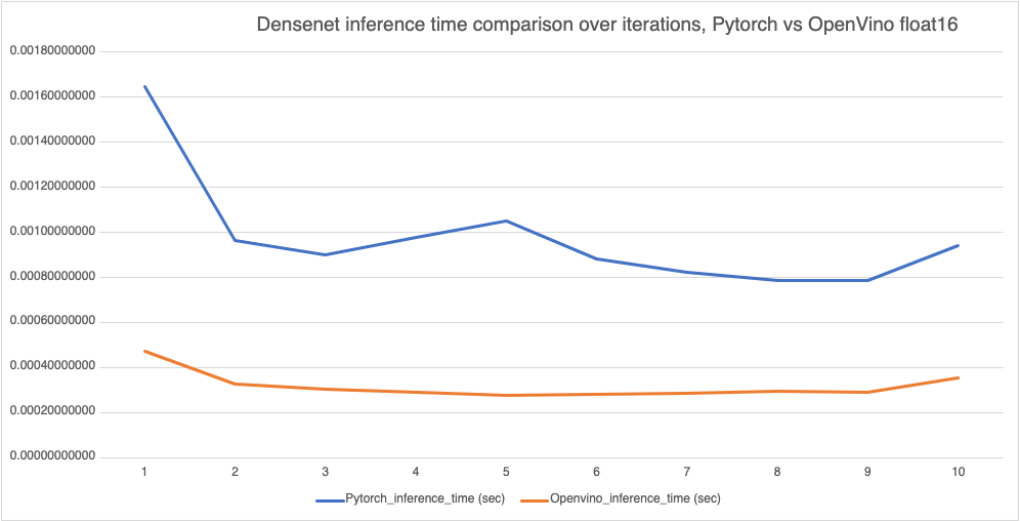

Figure 3: Inference time comparisons across iterations between Pytorch and OpenVino for DenseNet

In addition to runtime comparisons, we also wanted to explore how model sizes have an impact on runtimes. In Figure 4 we observe how Pytorch's model size relates to its inference times.

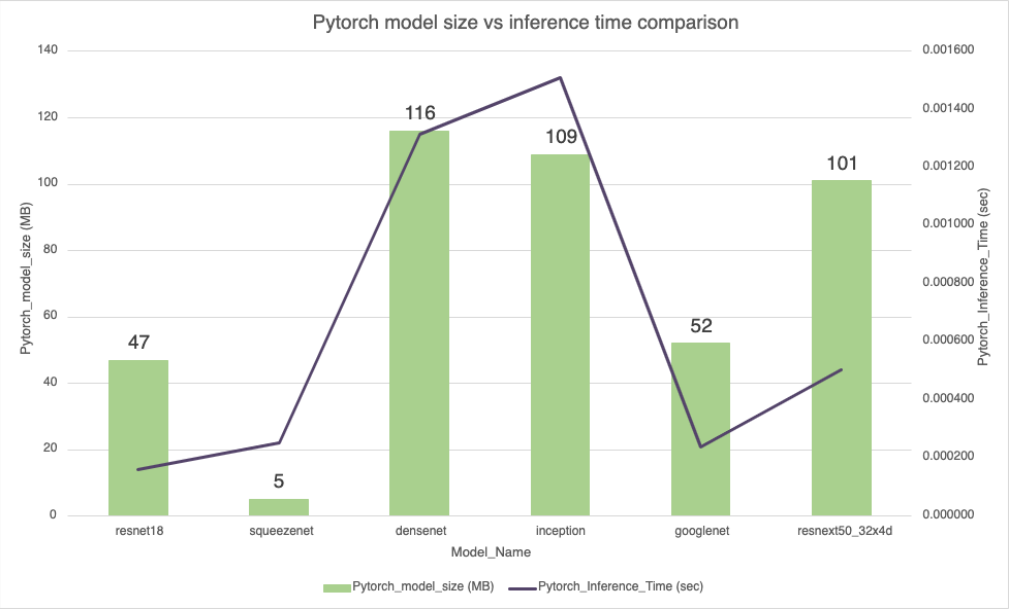

Figure 4: Inference time comparisons with respect to model size for Pytorch

Figure 5 represents how OpenVino's model size relates to its inference times. If we observe, in Pytorch a higher model size does not necessarily denote a higher inference time as can be seen between DenseNet and InceptionNet because inference also depends on the number of floating point operations (FLOPs). However, in OpenVino we observe a slightly stronger relation between model sizes and inference times.

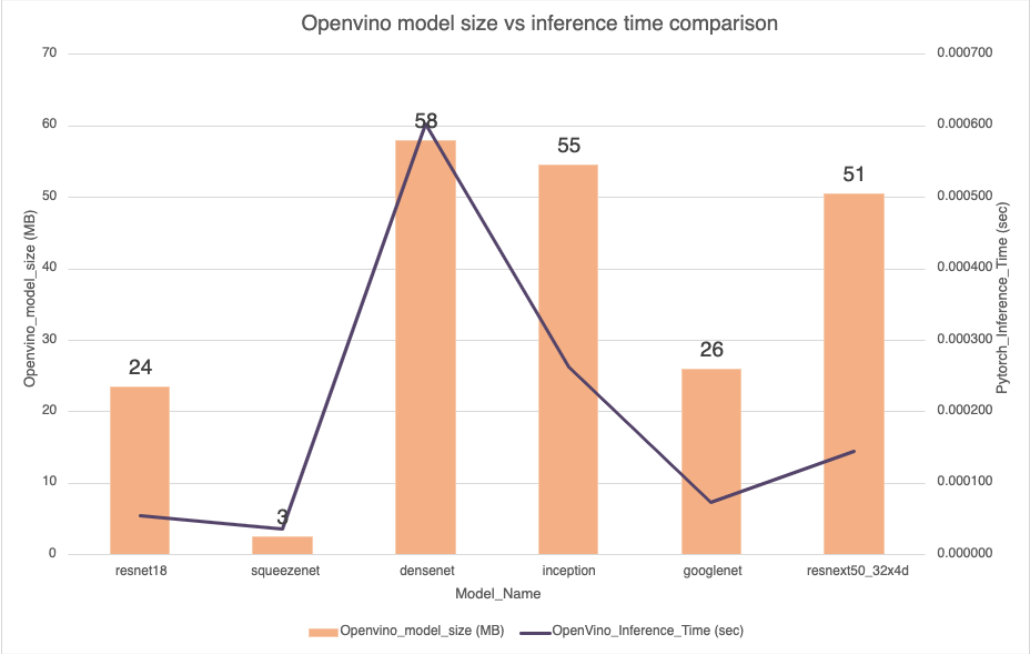

Figure 4: Inference time comparisons with respect to model size for OpenVino

OpenVino has unquestionably made the road to the commercialization of a deep learning model a great deal more productive. Not only can we deploy these models on a wide variety of hardware devices, but we can also do it on cloud instances to provide a seamless experience for application users. However, it is possible that it does not support all specialized or customized operators or structures out of the box. When this happens, more customization or manual optimization could be necessary in order to reach the best possible performance.

TENSORFLOW LITE MICRO

The ecosystem for embedded devices is rather chaotic and disorganized. Many of the features that are widely found in mainstream systems, such as dynamic memory allocation and virtual memory, are often omitted by system makers in order to enhance efficiency. These characteristics include those that allow for cross-platform compatibility. TF Micro [10] is a machine learning inference framework designed for use on embedded systems to execute deep learning models.

The application of machine learning is made substantially more widespread thanks to TF Lite Macro's ability to bring deep learning to embedded devices. TFLM is a machine learning framework that has been developed and designed to function successfully and efficiently on embedded devices with only a few kilobytes of memory at a time. TF Micro addresses the performance constraints enforced by embedded-system resource limits as well as the fragmentation difficulties that almost eliminate the possibility of cross-platform compatibility. The framework implements a one-of-a-kind interpreter-based method that, in addition to addressing these issues, enables flexibility. The TFLM framework's primary contributions consist of addressing two of the most singular issues that are exclusive to embedded systems. These challenges are hardware variability in a dispersed environment, a lack of software functionality, and significant limitations of resources.

CONCLUSION

The development of deep neural networks has led to an increase in the level of complexity of the models used in deep learning. Nowadays, models might have millions or even billions of parameters, which means that in order to train and infer from them, enormous computational resources are required. Model optimization strategies try to cut down on the amount of computing needed for these complicated models while simultaneously improving their overall efficiency. Many applications, particularly those that are deployed on edge devices, have restricted access to computational resources like memory, processing power, and energy. This is especially true for edge devices. It is absolutely necessary to optimize models for these contexts with limited resources in order to enable both efficient deployment and real-time inference. In this work, we observed how OpenVino framework can significantly reduce model inference times as compared to Pytorch.

Deep learning is becoming more accessible and cost-effective for researchers, organizations, and individual users thanks to model optimization approaches, which assist minimize the computing costs involved with model training and inference. All of these considerations have jointly spurred the need for model optimization as a topic of study, which has in turn fostered research and innovation in the development of architectures, algorithms, and frameworks that help in optimizing deep learning models in a variety of platforms.

REFERENCES

[1] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. ArXiv. /abs/1704.04861

[2] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. ArXiv. /abs/1602.07360

[3] Huang Yi, Sun Shiyu, Duan Xiusheng and Chen Zhigang, "A study on Deep Neural Networks framework," 2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), Xi'an, China, 2016, pp. 1519-1522, doi: 10.1109/IMCEC.2016.7867471.

[4] Han, S., Mao, H., & Dally, W. J. (2015). Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. ArXiv. /abs/1510.00149

[5] Han, Y., Huang, G., Song, S., Yang, L., Wang, H., & Wang, Y. (2021). Dynamic Neural Networks: A Survey. ArXiv. /abs/2102.04906

[6] Akhil Mathur, Nicholas D. Lane, Sourav Bhattacharya, Aidan Boran, Claudio Forlivesi, and Fahim Kawsar. 2017. DeepEye: Resource Efficient Local Execution of Multiple Deep Vision Models using Wearable Commodity Hardware. In Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '17). Association for Computing Machinery, New York, NY, USA, 68–81. https://doi.org/10.1145/3081333.3081359

[7] Intel OpenVino (https://docs.openvino.ai/2022.3/home.html)

[8] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. ArXiv. /abs/1512.03385

[9] Huang, G., Liu, Z., & Weinberger, K. Q. (2016). Densely Connected Convolutional Networks. ArXiv. /abs/1608.06993

[10] David, R., Duke, J., Jain, A., Reddi, V. J., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Regev, S., Rhodes, R., Wang, T., & Warden, P. (2020). TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems. ArXiv. /abs/2010.08678

Citation: Surabhi Sinha and Mayukh Maitra, Model Optimization of Generative AI Models: Making AI Faster, International Journal of Information Technology (IJIT), 4(1), 2023, pp. 58-64

https://doi.org/10.17605/OSF.IO/NRHBZ

Article Link:
https://iaeme.com/MasterAdmin/Journal_uploads/IJIT/VOLUME_4_ISSUE_1/IJIT_4_01_007.pdf

editor@iaeme.com