International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 06 | June 2020 www.irjet.net p-ISSN: 2395-0072

Acceleration of Deep Learning Image Classification Model

Ashwin P R

Department of Electronics and Communication Engineering, RV College of Engineering, Bangalore, Karnataka, India

---

Abstract - *Artificial intelligence and deep learning algorithms are widely used nowadays in almost every electronic devices, those algorithms need to be executed as quickly as possible in some time critical devices like self-driving cars etc. So for executing those Deep Learning algorithms, some accelerators are required to accelerate the time critical tasks. Most of the current technology uses AXI interface for offloading deep learning algorithms on to accelerator. In this paper the deep learning models are accelerated on GPUs. Intel integrated GPU is used for accelerating the various deep learning models using the Intel OpenVino software. Image classification models are the targeted models for acceleration. The benchmarking results of the selected deep learning models run on CPU* (Intel i5 4200 processor) *and Intel Integrated GPU present in the laptop are compared. All the simulations are carried on Windows Laptop using OpenVino software.*

Key Words: Artificial Intelligence, Deep Learning, GPU, CPU, Intel OpenVino.

1. INTRODUCTION

Nowadays Artificial Intelligence and Deep Learning are increasingly being used in many devices. These include edge devices which need to be energy efficient and don't have a lot of processing power with them unlike the huge servers [1]. For such ARM architecture based systems there is need of offloading common Artificial Intelligence and Deep Learning tasks to hardware such as FPGA and GPUs to obtain considerable amount of acceleration on them [2-4]. This acceleration of these tasks should happen in some critical systems like self-driving cars where it should rapidly detect the obstacles in its path and should locate the path where obstacles are less [6].

The growing complexity of software applications running on the low power devices like ARM processors call for the increase in the processing power. Typically, a RISC processors does not provide enough computational resources and the use of a specialized hardware or software accelerator is inevitable [5]. However in this paper, the benchmarking results like throughput, latency of the SqueezeNet model run on Intel CPU (Intel i5 4200 processor) and integrated Intel GPU in the laptop are obtained and compared for acceleration. The brief workflow of the software being used that is Intel OpenVino used for acceleration on GPU is discussed in software details section and then methodology obtained and results obtained are discussed in further sections. Also in next sub-sections brief introduction to convolutional neural networks and the various models used for computer vision tasks specially the models used for image classification task is given.

1.1 Convolutional Neural Networks and Image Classification models

The convolutional neural network (CNN) is a subdivision of deep learning neural networks. CNNs has made a major breakthrough in image classification and object recognition. They are most commonly used for analyzing the visual images and are majorly working towards the scenes in image classification. They are now found at peak of everything from Facebook's friend suggestion, photo tagging to self driving cars.

Image classification is the process of taking an input such as image of cat or dog and outputting a class or a probability that the input is a particular class. This process of image classification could be possible with a convolutional neural network. CNNs have an input layer, output layer, and hidden layers. The hidden layers consists of convolutional layers, ReLU layers, pooling layers, and fully connected layers. A CNN convolves the learned features with the input data and uses 2D convolutional layers. This means that this type of network is ideal for processing 2D images and also CNNs use very little preprocessing.

A CNN usually works by extracting the features from images automatically. By this the manual feature extraction can be completely eliminated. The features are not trained but they are learned while the network trains on a set of images. This makes deep learning models more accurate for computer vision tasks. CNNs learn feature detection through tens to hundreds of hidden layers. Each layer in CNN increments the complexity of the features that are learned while training. A CNN starts with an input image and then applies many different filters (convolution operation) to it to create a feature map and then applies a ReLU function to increase non-linearity of a feature map, this is because images are highly nonlinear, ReLU function removes negative values from an activation map by setting them to zero. And then applies pooling layer to each created feature maps in the convolution process and levels the pooled images into one long vector, and then inputs the vector into a fully connected artificial neural network and processes the features through the network. The final fully connected layer provides the probability of the class that we are looking for. Finally the

model trains through forward propagation and back propagation for many epochs. This repeats until the well-defined neural network with trained weights and feature detectors occur.

Some of the image classification models are LeNet, AlexNet, GoogleNet, ResNet, SqueezeNet and many more. All these models are trained on ImageNet dataset. ImageNet is a large visual database designed mostly for use in visual image classification and object recognition software research. More than 14 million images are hand annotated by the project to show that, what objects are pictured and in approximately one million of images, the bounding boxes are being provided. LeNet is the first image classification model developed, it consists of convolutional encoder consisting of two convolutional layers and a dense block consisting of three fully connected layers. AlexNet came after LeNet with some improvements, it is comprised of five convolutional layers, then followed by three fully connected layers and has a top 5 error rate of 10%. Compared to LeNet, it has more filters per layer and stacked convolutional layers and hence classification accuracy is increased. GoogleNet model is the successor of AlexNet model which has some improvements in its accuracy and it made by a team at Google and it also has a name Inception V1, which achieved a top 5 error rate lower than 7%. The architecture of GoogleNet is 22 layers deep and reduced the number of parameters by using batch normalization, RMSprop and image distortions compared to previous models. GoogleNet has only 4 million parameters, a drastic reduction compared to the 60 million parameters of AlexNet. Residual Neural Network or ResNet comes after GoogleNet which achieved a top 5 error rate of 3.57%. ResNet have up to 152 layers. It uses skip connections to jump towards certain layers in the process and has heavy batch normalization. This smart implementation of the architecture of ResNet makes it possible to have 6 times more layers than GoogleNet with less complexity and more accuracy.

In this paper benchmarking results are achieved on SqueezeNet model. In the next sub-section the description of SqueezeNet model is given.

1.2 SqueezeNet model

SqueezeNet is deep convolutional neural network computer vision applications. The main aim for designing the SqueezeNet model was to create a more smaller neural network compared to other neural networks with same accuracy, and with much less parameters that can efficiently fit into the computer memory and can easily be transported over a computer network. The smaller neural network is achieved by replacing 3x3 filters with 1x1 filter, which leads to 9X fewer parameters and then decreasing the number of input channels to 3x3 filters. With SqueezeNet, there is a 50× reduction in model size compared to AlexNet, while meeting or exceeding the top 1 and top 5 accuracy of AlexNet. With SqueezeNet model it also has a top 5 error rate of less than 7% and has a less parameters, with more accuracy.

In the next section the brief description of software being used i.e Intel OpenVino is given and how it is used to obtain benchmarking results of the pre-trained models. And then the methodology to obtain acceleration results on GPU is given.

2. SOFTWARE DETAILS

The Intel OpenVino software is used in this paper for achieving acceleration on Integrated Intel GPU. The OpenVINO software is a comprehensive toolkit which is used for developing and deploying the vision oriented solutions on Intel platforms. The block diagram, shown in the Fig. 1, depicts the deployment workflow of the software and the project.

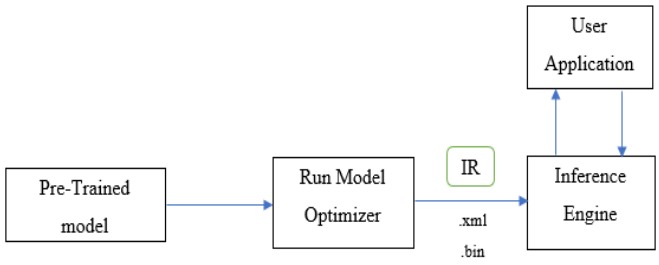

Fig -1: Workflow of OpenVino software

Description of workflow in steps are given below:

1. Configuring the Model Optimizer for the specific framework. The pre-trained model (.prototxt and .caffemodel for example) in some framework is given as input to the model optimizer.
2. By running the Model Optimizer which can produce an enhanced Intermediate Representation (IR) of the pre-trained model used based on the parameters like trained network topology, weights and biases values, and other important parameters.
3. Testing the selected model in the IR format (.xml and .bin) using the Inference Engine in the target environment with provided Inference Engine sample codes.
4. Integrating the Inference Engine in the application to deploy the model in the target device or platform.

3. METHODOLOGY OBTAINED

Existing methods of accelerating the deep learning models and convolutional neural network models were analyzed [7-9], where a random forest classifier and binarised neural network were accelerated on FPGA, CPU and GPU and compared the results obtained. Acceleration of the task is done in Intel Integrated Graphics GPU present in the laptop and the performance is compared with the results obtained when the model is run on the CPU. Algorithm for running the

model on GPU is designed by using the algorithm used for running the model on the CPU. First step is to download the required model i.e SqueezeNet model using the model downloader present in toolkit by running the downloader.py python file. Next step is to convert the model to the Inference Engine (IR) format. Go to the Model Optimizer directory and run the mo.py script with specifying the path to the model, model format and output directory to generate the IR files. The IR files are generated in mentioned folder after running the model optimizer file. This IR file and test dataset are used as input for the inference engine code. The benchmarking application code takes input as the required model and test dataset and provides benchmarking results of the model. Also the classification of image inference engine code is run which takes test dataset and the model in the form of IR as input.

The Intel Integrated Graphics is the GPU embedded on same die as the processor and shares available system memory, which can perform more tasks than CPU present. The results obtained by performing according to mentioned methodology is given in the next section.

4. RESULTS AND DISCUSSION

The selected model for the acceleration is image classification model such as SqueezeNet model as discussed in previous sections. Classification of images inference engine code is executed with the above model. For that code, the inputs are SqueezeNet model in Intermediate Representation format and one test image. In this experiment the image of car is given as test input.

The results obtained are of top ten probabilities of classes of the dataset with labels. The below screenshot in Fig.2, shows the classification results for the SqueezeNet model.

Fig -2: Image Classification result of SqueezeNet model

By looking at the results obtained, we can observe that the predicted probability of the car label is more compared to any other probabilities. And the SqueezeNet model has predicted the test image of car properly.

The results obtained in Fig. 2 is the predicted classification result of the test data. The performance of the model is obtained by running the benchmarking inference engine code by mentioning the CPU as the target device, with the model and test image as input. Below screenshot in Fig. 3 gives the performance benchmarking results of SqueezeNet model which is run on CPU of the laptop.

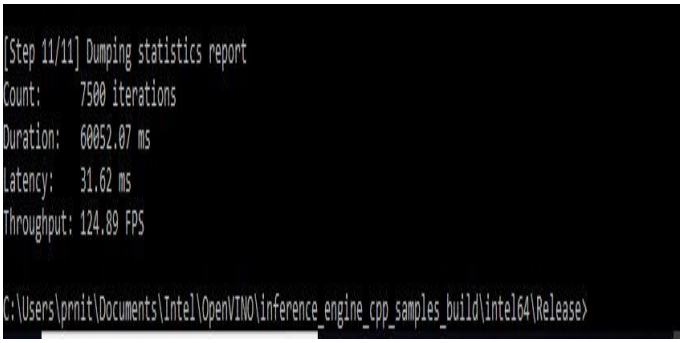

Fig -3: Performance Benchmark result of SqueezeNet model ran on CPU

The acceleration of the model is achieved by modifying the code used to run on CPU in order to run it on GPU. By running the model on GPU (Intel integrated graphics) present in laptop, throughput is improved and latency is decreased that means acceleration is achieved. Below figure (Fig. 4) shows performance benchmarking results of SqueezeNet models which is run on GPU.

Fig -4: Performance Benchmark result of SqueezeNet model ran on GPU

The above result shows that throughput is increased drastically by running the model on GPU compared to running it on CPU. And also a latency is decreased appreciably. That means delay of execution is reduced in GPU and acceleration is achieved.

5. CONCLUSIONS

First in the starting section brief description of convolutional neural networks, its main layers and how CNN is used in image classification models. Also some of the image classification models such as LeNet, AlexNet, GoogleNet,

ResNet and SqueezeNet are discussed and their performances are compared. And then the brief workflow of the software being used that is Intel OpenVino used for acceleration is discussed. Using this software, the acceleration of SqueezeNet model is achieved on GPU (Intel integrated graphics). The throughput of SqueezeNet model run on CPU (Intel i5 4200 processor) obtained is 124.89 FPS and latency is 31.62 ms. And the throughput of accelerated SqueezeNet model achieved is 391.06 FPS and the latency is 5.03 ms. Throughput and latency is improved, that is the acceleration of model is achieved.

REFERENCES

[1] A. Ignatov, R. Timofte, A. Kulik, S. Yang, K. Wang, F. Baum, M. Wu, L. Xu, and L. V. Gool, "AI benchmark: All about deep learning on smartphones in 2019," IEEE/CVF International Conference on Computer Vision Workshop (IC-CVW), IEEE, Oct. 2019. doi: 10.1109/iccvw.2019.00447.

[2] E. Nurvitadhi, S. Subhaschandra, G. Boudoukh, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. O. G. Hock, Y. T. Liew, K. Srivatsan, and D. Moss, "Can FPGAs beat GPUs in accelerating next-generation deep neural networks?" Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '17, ACM Press, 2017. doi: 10.1145/3020078.3021740.

[3] S. Zheng, A. Vishnu, and C. Ding, "Accelerating deep learning with shrinkage and recall," IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), IEEE, Dec. 2016. doi: 10.1109/icpads.2016.0129.

[4] V. T. Vu, G. Cats, and L. Wolters, "GPU acceleration of the dynamics routine in the HIRLAM weather forecast model," International Conference on High Performance Computing & Simulation, IEEE, Jun. 2010. doi: 10.1109/hpcs. 2010.5547152.

[5] T. Patyk, P. Salmela, T. Pitkaanen, P. Jaaskelainen, and J. Takala, "Design methodology for offloading software executions to FPGA," Journal of Signal Processing Systems, vol. 65, no. 2, pp. 245, Jul. 2011. doi: 10.1007/s11265-011-0606-x.

[6] S. Liu et al, "Creating Autonomous Vehicle Systems, Morgan Claypool Publishers", 2017.

[7] B. V. Essen, C. Macaraeg, M. Gokhale, and R. Prenger, "Accelerating a random forest classifer: Multi-core, GP-GPU, or FPGA?," IEEE 20th International Symposium on Field-Programmable Custom Computing Machines, IEEE, Apr. 2012. doi: 10.1109/fccm.2012.47.

[8] E. Nurvitadhi, D. Sheeld, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC," International Conference on Field-Programmable Technology (FPT), IEEE, Dec. 2016. doi: 10.1109/fpt.2016.7929192.

[9] K. Nagarajan, B. Holland, A. D. George, K. C. Slatton, and H. Lam, "Accelerating machine-learning algorithms on FPGAs using pattern-based decomposition," Journal of Signal Processing Systems, vol. 62, no. 1, pp. 43, Jan. 2009. doi: 10.1007/s11265-008-0337-9.