6-1-2023

Performance Analysis of CNN Model for Image Classification with Intel OpenVINO on CPU and GPU

Md Maksud-Ul-Kabir Rico
University of Windsor

Performance Analysis of CNN Model for Image Classification with Intel OpenVINO on CPU and GPU

By

Md Maksud-Ul-Kabir Rico

A Thesis
Submitted to the Faculty of Graduate Studies
through the Department of Electrical and Computer Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science
at the University of Windsor

Windsor, Ontario, Canada

2023

Performance Analysis of CNN Model for Image Classification with Intel OpenVINO on CPU and GPU

by

Md Maksud-Ul-Kabir Rico

APPROVED BY:

_____

D. Wu
School of Computer Science

_____

K. Tepe
Department of Electrical and Computer Engineering

_____

M. A.S. Khalid, Advisor
Department of Electrical and Computer Engineering

April 14, 2023

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the Work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Deep learning (DL) has proven to be a significant solution for analyzing complex datasets such as images, videos, text, and speech. Convolutional neural networks (CNN) have proven to be one of the most popular and powerful deep neural networks to perform image classification. However, due to its high computational complexity, high speed and accuracy required in many real-world applications, CNN implementation presents a computational challenge for computing devices.

The recent advances in hardware have led to the emergence of the graphical processing unit (GPU) as a solution for speeding up the process of executing complex deep learning algorithms. Although a central processing unit (CPU) is designed to handle a wide range of tasks quickly, it is limited in the concurrency of tasks that it can execute in parallel. This research presents a comparative analysis of CPU and GPU for image classification using the pre-trained *Caffe vgg16* CNN model optimized by Intel OpenVINO's model optimizer feature. OpenVINO is an open-source toolkit for optimizing and deploying DL inference. It also boosts deep learning performance in computer vision, speech recognition, and other common tasks. Performance characteristics of the optimized model for image classification were studied by running it on the Intel Core i5-1035G1 CPU and Intel UHD Graphics G1 GPU. Moreover, the accuracy was tested by running the optimized models on the first 50,000 images of the *ImageNet* 2012 validation dataset.

The research indicates that GPU implementation is on average 1.5x times faster than the CPU implementation for the single precision optimized model and on average 2x times faster than the CPU implementation for the half-precision optimized model. On CPU, the single precision optimized model achieves 70.96% top-1 accuracy and 89.88% top-5 accuracy, and the half-precision optimized model achieves 64.64% top-1 accuracy and 86.21% top-5 accuracy. On GPU, the difference between the single precision optimized model and the half-precision optimized model on top-1 and top-5 accuracies are almost the same as CPU. The research also shows that there exists a significant latency-throughput trade-off.

ACKNOWLEDGEMENTS

Firstly, I would like to express my deepest appreciation to Dr. Khalid for his guidance and encouragement during my study at the University of Windsor. This endeavour would not have been possible without his time and support. I would also like to thank my thesis committee members Dr. Tepe and Dr. Wu for offering their valuable time and insightful comments. Finally, I am extremely grateful to my parents for their love, belief, and support. Without them, this thesis would not be completed.

TABLE OF CONTENTS

CHAPTER 1
Introduction

1.1 Motivation

Conventional neural network (CNN) is considered one of the most significant models which help in developing computer vision applications [1]. These models play an imperative role in performance optimization. Moreover, image categorization has applications in robotics, surveillance, search, transportation, and other areas. One of the most intricate tasks of computer vision research is classifying images [4]. It searches for regions in an image that potentially includes a specific object before extracting and classifying each region using an image classification model. Consequently, a good algorithm is necessary for a decent picture classification approach. According to the *ImageNet* competition, CNNs are the models that are most frequently utilized for these applications. CNNs are neural networks that perform well at image and video classification. In addition, CNNs have also been applied to a wide range of other tasks, including face and object identification, autonomous driving, and drone navigation.

The initial section of a standard architecture CNN is made up of several convolutional and pooling layers for automatic feature extraction, while the second section is made up of fully connected (FC) layers for classification [5]. It has also been observed that CNNs operate by displaying a portion of an image, and then looking for objects, such as vertical lines, arcs, or circles, that the network can identify from these components. After that, the image is categorized utilizing several attributes. Each of the two types of layers is made up of feature maps. The input image's characteristics are recognized by the first convolutional layer. Typically, it is made up of several feature maps, each of which recognizes a particular feature. A second type of layer known as the pooling layer is typically added after the convolutional layer. Furthermore, each convolutional layer map feeds information into the corresponding feature map of the following pooling layer [5]. Until the final pooling layer is reached, the convolution and pooling layers are alternated in accordance with the network depth. The output layer is the last one before the FC layer.

The topology of the network is defined by the depth of the network and the number of neurons. Modern CNNs generally have millions of parameters and thousands of neurons grouped in five or more hidden layers. Designing a network with an optimum topology is one of the key difficulties in successfully deploying a CNN. In addition to this, the number of parameters and memory needed for a CNN model all depends on its topology [3]. The selection of topology is still primarily based on trial and error, though. The large amount of processing required by CNNs calls for dedicated and tailored software and hardware support methods. Usually, CNNs are executed on general-purpose processors, such as CPUs and GPUs. In addition, CPUs as well as GPUs have limitations like high power consumption and limited memory that limit their use and suitability for real-time and real-world applications in drones, robots, and self-driving cars. As the CNN applications increase in number and complexity, so do the software and hardware architectures for their execution and training.

The amount and diversity of research on CNN inference acceleration in recent years demonstrate tremendous industrial and academic interest. Many software toolkits and hardware designs have been devoted to accelerating CNN inference. Among those software toolkits, Open Visual Inference Neural Network Optimization (OpenVINO) is designed by Intel to provide facilitation as well as acceleration for the development of models based on deep learning in various frameworks [6]. In addition, the toolset enables deep learning on Intel-designed hardware accelerators as well as heterogeneous platforms (CPUs, GPUs as well as FPGAs). The toolkit is expected to optimize CNN models in computer vision as well as achieve higher performance than standalone deep learning frameworks. Accelerators like the CPUs GPUs and FPGAs can be used to match the desired performance metrics (latency, throughput, and execution time) by utilizing Intel OpenVINO's optimization features [7]. The main motivation for conducting this research study is that it helps in getting better insights into the performance of a CNN model for classifying images on CPU and GPU using Intel OpenVINO toolkit.

1.2 Objectives

The main aim of this research study is to analyze the performance of a CNN model for image classification with Intel OpenVINO on CPU and GPU. This study also helps in

identifying the performance improvement of the CNN model inference on the CPU and GPU through the toolkit. In conclusion, this study contributes to the understanding of the inference performance of the CNN model by providing data and information.

1.3 Organization

The rest of the thesis is organized as follows:

Chapter 2 provides background information about HPC tools, supervised machine learning, and related work on convolutional neural network implementation. First, a basic overview of Intel OpenVINO will be presented. Then a detailed description of the widely used neural network architecture: CNN is provided. Finally, the related research on CNN implementation on CPUs and GPUs will be briefly covered.

In chapter 3, the design flow and performance optimization schemes for inference of the design toolkit will be discussed. It'll begin by introducing the design flow of Intel OpenVINO, followed by optimization strategies for latency and throughput.

In chapter 4, test approaches and evaluation results will be illustrated for the proposed CNN model. It begins with the introduction of experimental setup including software and hardware information. Then the performance of the proposed model will be analyzed in terms of accuracy, throughput, execution time, and latency.

Finally, chapter 5 presents conclusions and suggestions for further related research in future.

CHAPTER 2

Background and Related Work

Many recent developments in deep learning technology enable several sophisticated applications that have taken place at the edge, making prevailing edge devices like a speaker, mobile phones, television, and camera more functional. Such applications range from tasks related to computer vision like object detection, speech recognition, segmentation, Voice detection, and image classification typically leverage pre-trained deep learning models for inference on input data. However, executing model inference directly over edge devices has become more difficult for lesser latency and lower network bandwidth [8]. Typical edge devices are equipped with the system on a chip (SoC) that implements multiple compute units like the graphical processing unit (GPU), optional digital system processor, network processing unit, and central processing unit (CPU) [9]. In practice, most of the model inference at the edge is further executed on the CPU because of smooth programmability and more flexible portability among various systems on chips.

There is significant research on optimizing deep learning model inference, notably CNN models, directly on edge devices. It is preferable to use additional computation units to address the DL model inference tasks. There are studies concentrated on using mobile GPUs to handle computationally demanding operations like convolution and matrix multiplication on mobile devices [10]. Their model coverage was typically limited since they did not pay much attention to optimizing the vision-specific operators on the integrated GPUs. Murthy *et al* [11] have suggested multiple methods for improving the performance of object detection and image classification models on standard integrated GPUs. Since DL and CNNs frequently need several layers and parameters to function correctly, such research and applications are best suited for mid-range to high-end GPUs. These GPUs require a lot of power, are expensive to create and produce, and are pricey for the ordinary customer.

Deep learning workloads are being accelerated across a wide range of hardware, including CPUs, GPUs, FPGAs, and specialized accelerators, as deep learning shows increasing power in practical applications [8]. Contemporary deep learning frameworks typically use these optimized implementations to execute deep learning training as well as inference on

the respective hardware targets. There are additional works designed specifically for inference to satisfy its needs for low latency and minimal binary sizes on various hardware targets.

2.1 Convolutional Neural Network

Supervised machine learning is a type of machine learning that can be implemented using deep neural network algorithms. Supervised machine learning algorithms can learn with enough training using various data types. The algorithm's function depends on the neural network (NN). The NN works like a human brain which computes information using millions of neurons [12].

A range of research has been conducted, and applications have been developed by applying supervised machine learning algorithms. Supervised learning is known to predict or categorize a specific result of interest. Supervised categorization is one of the functions that intelligent systems carry out most frequently. A number of Artificial Intelligence (Logical/Symbolic approaches), Perceptron-based methods, and Statistics-based methods have been developed, such as Bayesian Networks and Instance-based techniques [12].

CNN is a supervised type of deep learning that is preferably used in image recognition and computer vision. With the help of its multiple layers, it processes and extracts important features from images [12].

2.1.1 Architecture

In computer vision workloads, convolutional neural networks are frequently utilized. Typically, a CNN model is represented as a computation graph in which a node stands in for an operation and a directed edge pointing from node X to node Y indicates that the output of operation X serves as (part of) the inputs of operation Y, meaning that Y cannot be executed before X. To retrieve the result from a model inference, the input data must be sent through the graph [1].

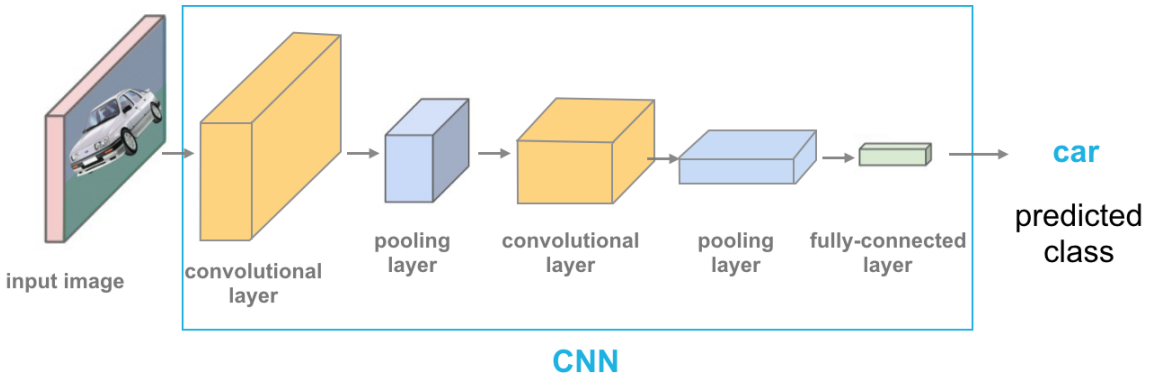

Figure 1: Convolutional Neural Network [1]

2.1.2 VGG16 CNN Model

In the quest to make computers "see" the world, *vgg16* proved to be a pivotal turning point. The discipline of computer vision (CV) has advanced significantly in this area several decades ago. *vgg16* is among the important discoveries that paved the way for more developments in this field. *vgg16* (also called OxfordNet) is a convolutional neural network architecture named after the Visual Geometry Group from Oxford, which developed it [14]. Andrew Zisserman and Karen Simonyan developed this convolutional neural network (CNN) model from the University of Oxford [14]. The concept for the model was released in 2013, but the actual model was shown as part of the ILSVRC *ImageNet* Challenge in 2014 [15]. Each year, the *ImageNet* Large Scale Visual Recognition Challenge (ILSVRC) assessed methods for large-scale picture categorization (and object recognition) [16]. It is still considered to be an excellent vision model.

vgg16 was demonstrated to be the model with the best performance out of all the settings on the *ImageNet* dataset [15]. It is a convolutional neural network that is 16 layers deep. The model loads a set of weights pre-trained on *ImageNet*. The model achieves 92.7% top-5 test accuracy in *ImageNet*, which is a dataset of over 14 million images belonging to 1000 classes. The model's setup is considered to have a fixed 224 × 224 image with the three channels as its input. The picture is transmitted through the first stack of two convolution layers after ReLU activations, with a minuscule 3 × 3 receptive area. Each of these two layers has 64 filters. The convolution stride is fixed at 1 pixel, whereas the padding is 1 pixel. In this configuration, the spatial resolution is preserved, and the

dimensions of the output activation map coincide with those of the input image. Then, spatial max pooling is applied to the activation maps using a 2-pixel stride over a 2 x 2 pixel frame. As a result, the activations' size is reduced by half. At the bottom of the first stack, some activations are 112 x 112 x 64 in size [14].

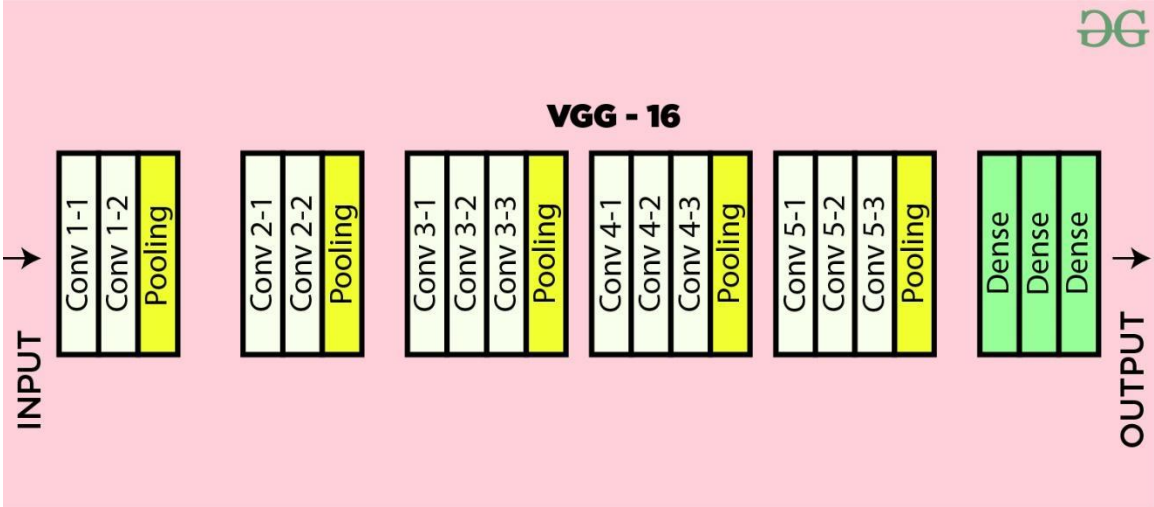

Figure 2: VGG16 CNN Model [14]

The second stack, which is identical to the first but contains 128 filters as opposed to 64 in the first stack, is then applied to the activations. As a consequence, the size is 56 x 56 x 128 after the second layer. The third stack is subsequently added, which comprises three convolutional layers and a max pool layer. Due to the 256 filters that were used in this instance, the output size of the stack is 28 x 28 x 256. Then, two stacks of three convolutional layers are built, each with 512 filters. An output of 7 x 7 x 512 will be given by both of these stacks. The three fully connected layers that come after the convolutional layer stacks are separated by a flattening layer. The 1,000 neurons of the last fully connected layer, which serves as the output layer and corresponds to the 1,000 possible classes in the *ImageNet* dataset, are equal to each of the prior two layers' 4,096 neurons. After the output layer comes the Softmax activation layer, which is used for category classification [14].

2.2 Intel OpenVINO Overview

Open Visual Inference Neural Network Optimization (OpenVINO) is a set of tools designed by Intel to provide facilitation as well as acceleration for the development of

models based on deep learning in various frameworks. In addition, the toolset enables deep learning on Intel-designed hardware accelerators as well as heterogeneous platforms (CPUs, GPUs as well as FPGAs). The toolkit is expected to optimize CNN models in computer vision as well as achieve higher performance than standalone deep learning frameworks [7].

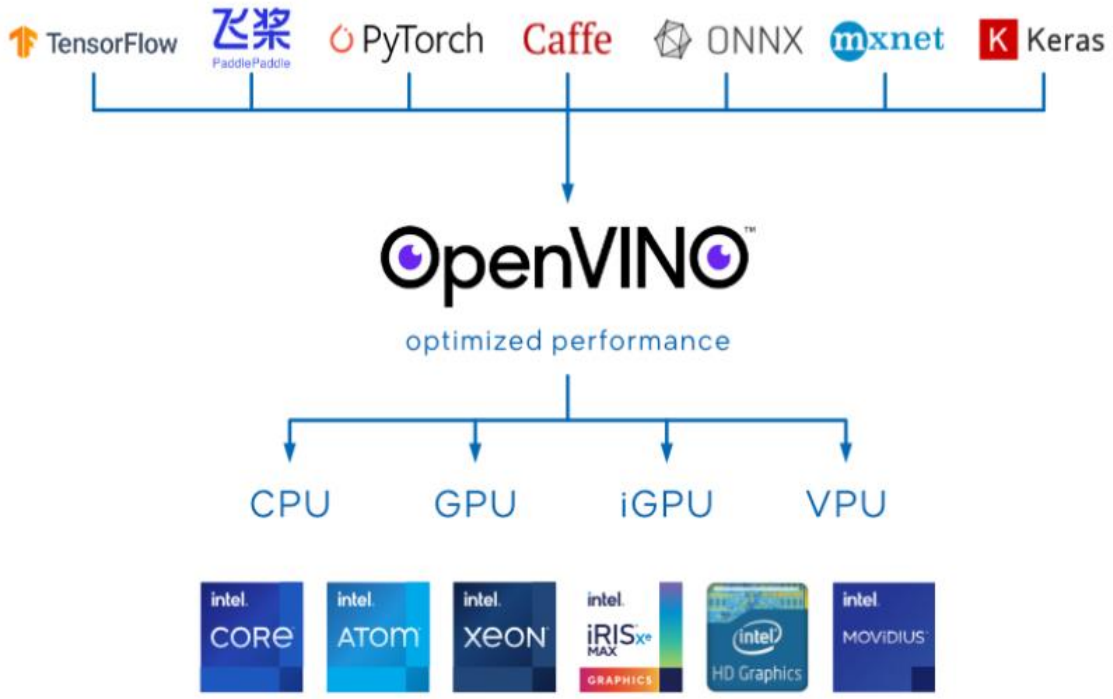

Figure 3: Intel OpenVINO Supported Frameworks and Devices [6]

As shown in Figure 4, OpenVINO consists of two main components: the model optimizer and the inference engine. The model optimizer converts the pre-trained models into two files (.xml and .bin). These are known as intermediate representation (IR) files. The inference engine is then utilized to execute the models on different hardware platforms. For different hardware, it uses different plugins [7].

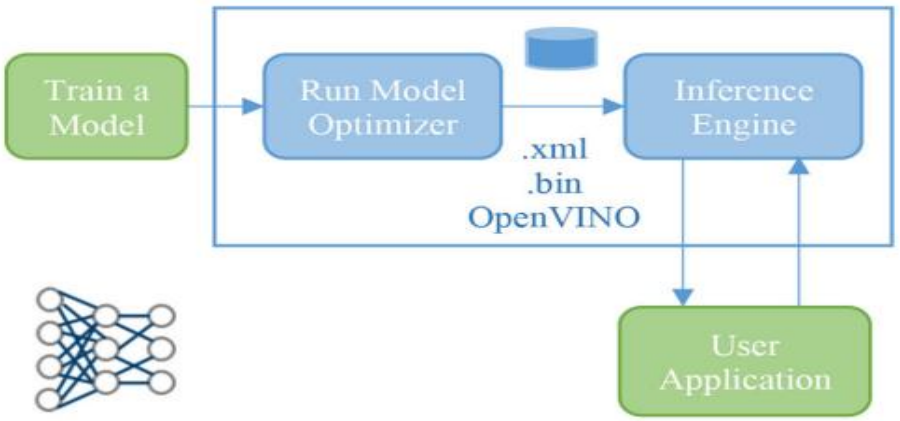

Figure 4: General Workflow of OpenVINO [7]

2.2.1 Workflow

Some of the deep learning models had inference times that were slower than anticipated. Intel's OpenVINO framework is leveraged to speed up the underlying neural networks in order to address this. Model Optimizer and Inference Engine are the foundational elements of OpenVINO. The Model Optimizer is in charge of converting pre-trained models from our common neural network models, such as CNN, into a format that OpenVINO can use [7]. OpenVINO's Inference Engine feature is utilized to significantly reduce inference times once the model has been tweaked. This results in a significant performance improvement over the conventional models that were previously converted.

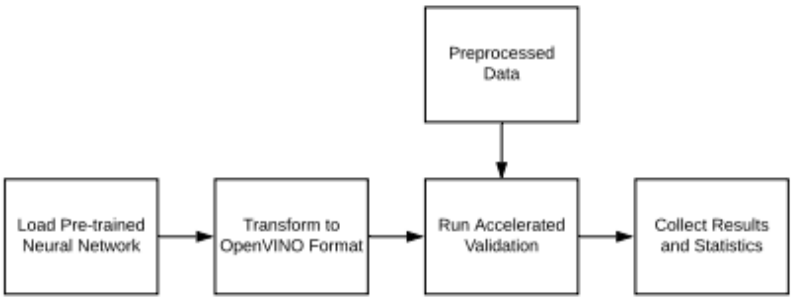

Figure 5: Advanced Workflow of OpenVINO [6]

Pre-trained models can be downloaded and modified for use with OpenVINO by following the toolkit's Model Downloader and Model Optimizer features. Model Optimizer is a cross-

platform command-line utility that makes it easier to switch between the training and deployment settings, analyses static models, and modifies deep learning models for the best performance on end-point target devices. Models can be picked from the wide range offered in Intel's Open Model Zoo [7].

A variety of post-training and training-stage optimization techniques help to improve a model to have higher inference performance. The Post-Training Optimization tool uses specialized techniques without model retraining or fine-tuning to speed up the inference of deep learning models, such as post-training quantization [7].

The toolkit is a collection of C++ libraries with bindings for C and Python that provide a standard API for delivering inference solutions on various Intel hardware platforms. It introduces the simplest type of deployment and the fastest method for doing it [7].

2.2.2 Supported Intel Devices

OpenVINO can infer various DL models on different hardware devices. It contains several plugins that help to load models on several hardware devices. The plugins are maintained in open-source by the OpenVINO team. The list of compatible devices with additional details is given below [6].

Device	Plugin	Library	Short Description
CPU	Intel CPU	openvino_intel_cpu_plugin	Intel Xeon with Intel Advanced Vector Extensions 2 (Intel AVX2). Intel Advanced Vector Extensions 512 (Intel AVX-512), and AVXS12_BF16, Intel Core Processors with Intel AVX2, Intel Atom Processors with Intel Streaming SIMD Extensions (Intel SSE)
GPU	Intel GPU	openvino_intel_gpu_plugin	Intel Processor Graphics, including Intel HD Graphics and Intel Iris Graphics
VPU	Myriad Plugin	openvino_intel_myriad_plugin	Intel Neural Compute Stick 2 powered by the Intel Movidius Myriad X

Table 1: Intel OpenVINO Supported Intel Devices [6]

2.2.3 Performance Benchmarking App

The benchmark app is a unique feature of Intel's OpenVINO. Models in the OpenVINO IR (model.xml and model.bin) and ONNX (model.onnx) formats can be used with the benchmarking application. The opening virtual environment should be activated before executing the benchmark app. A detailed command is used to conduct benchmarking on a model with the default settings.

By default, the application will load the chosen model onto any hardware device and run inference. It publishes details about benchmark settings as it loads. When benchmarking is finished, it reports the average throughput and the lowest, average, and maximum inferencing delay.

By adjusting a few of the user model's execution settings, the benchmark results can be enhanced beyond the default setting. To optimize the runtime for a higher FPS or a shorter inferencing time, the "throughput" or "latency" performance suggestions can be utilized. [17]

2.2.4 Accuracy Checker App

The Accuracy Checker is a deep learning accuracy validation system that is expandable, adaptable, and adjustable. The tool's modular design enables the replication of the validation process and the collection of aggregated quality indicators for well-known datasets supported by OpenVINO. Installing the necessary frameworks is necessary to evaluate some models. After installing all the prerequisites, the Accuracy Checker function can be used [18].

2.3 Related Works

Modern CNN has proven their effectiveness in various applications. Much research is going on improving the optimization technique of CNN models. Andriyanov [19] ran the pre-trained *SSD_MobileNet_V2_COCO* CNN model on Intel ® Core ™ i5-4460 CPU for image classification using *TensorFlow* and OpenVINO Inference Engine. In this research, COCO dataset was used. This research showed that the use of OpenVINO had increased the performance of the *SSD_MobileNet_V2_COCO* CNN model by 130 times on average.

Zunin [20] provided a performance analysis of various neural networks using the plug-ins of multiple devices and heterogeneous plug-ins on multiple connected devices of the OpenVINO toolkit. *DUC*, *SSD*, *RetinaNet*, *Tiny YOLOV2*, and *YOLOV2-COCO* were selected for this research. Testing was carried out using the *COCO Test2017* dataset. The targeting platforms were Intel Core i9-9900KF CPU, Intel Core i9-9900KF CPU, Intel Core i9-9900KF CPU, Intel HD Graphics 630, and Intel UHD Graphics 620. The research results showed that the multi-device and heterogeneous plugins do not increase performance compared to a single device.

Guerrouj *et al.* [21] evaluated the inference performance of 10 classification models and 9 object detection models using the OpenVINO toolkit. This research also analyzed the implementation of these models on the DE5a-Net DDR4 equipped with an Arria 10 GX FPGA. The research concluded that the heterogeneous FPGA/CPU architectures provided on average 3.5 times faster acceleration than a homogenous CPU architecture for classification models.

Demidovskij *et al.* [22] provided a simple and effective neural network optimization strategy. The paper introduced three consequent steps: input reduction, quantization, and optimal inference configuration search. The paper represented a comprehensive solution with assisting users in deploying an optimized model to their application. The above research works show that the Intel OpenVINO toolkit necessitates application engineers with all the required functionality to optimize and deploy the CNN models.

CHAPTER 3

Methodology and Optimization

This chapter discusses the research methodology used to achieve performance optimization of the pre-trained *vgg16* CNN model for image classification. Intel OpenVINO automatically optimizes deep learning pipelines by utilizing its optimization features and inferencing parallelism across CPU, GPU, FPGAs, and other Intel processors. The sections of this chapter describe the optimization process used in Intel OpenVINO.

3.1 Design Methodology

The methodology is designed with the help of Intel OpenVINO, which enables the user to optimize a deep learning model from various frameworks and deploy it on a range of Intel hardware platforms. To decrease end-to-end latency and increase throughput, users can integrate and offload extra activities for pre-and post-processing to accelerators. In a CNN model, a graph typically refres to the visual representation of the network architecture. It illustrates the different layers in the network and how they are connected to each other. The performance of model inference may be improved by optimizing the graph, for example, by pruning unneeded nodes and edges and pre-calculating values independent of input data [26].

Convolutions account for the majority of the computation in CNN model inference. Contemporary hardware platform's parallelization and vectorization properties can be utilized to speed up performance. By arranging the data layout and, by extension, the computation in an architecture-friendly fashion, it is possible to obtain excellent performance of convolution operations on hardware platforms [24].

Optimizing the convolution operation is difficult for overall performance for the workload of CNN as it requires most of the computation. Due to this, many studies have gone through works which mainly focus on code-level for higher performance [25].

3.2 Optimization Processes

The prior works typically dig deep into the assembly code level for high performance. It is a well-studied subject. One of the types of research has demonstrated how to optimize a

single convolutional layer using the most recent CPU technologies (SIMD, FMA, parallelization) without using laborious assembly code or C++ intrinsic [26]. It is thus simple to expand our optimization by managing the implementation at a high level. The following section discusses the optimization process for the project. This section consists of input shape, quantization, and batching.

3.2.1 Input Shape

The most significant rule in higher-performance computing is to make the computationally intensive part faster. The model optimizer can make the model's performance more effective by providing a different shape. It can be acquired with two parameters: ***input_shape*** and ***static_shape***. These are utilized for specific conditions. The new shape of the data will not change any further from one inference to the other. It is suggested to set up ***static_shape*** alongside ***input_shape*** for the inputs. However, it might not be beneficial from the performance and memory consumption perspective. The model optimizer will utilize the input shape parameter for this research. Similar functionality is present through the reshape method as well. Furthermore, the model optimizer cannot convert the shape of other layers of some models from various frameworks. These issues could relate to the models having inputs of the undefined ranks and the case of cutting off sections of the model [27]. Models with dynamic input shapes have unknown dimensions which are supported by the model optimizer during conversion. However, it is advised to set up shapes using the ***input_shape*** parameter. A similar function can be done through the ***reshaping*** method. OpenVINO API could have restrictions to infer models with undefined dimensions over some hardware. In such cases, the parameter of the ***input_shape*** and ***reshape*** method could help resolve the undefined dimensions [27]. In this way, the user must specify the input shapes explicitly by utilizing the ***input_shape*** parameter. Using the model optimizer feature, the input data shape of the pre-trained *vgg16* CNN *Caffe* model is converted to 2, 3, 200, and 200. The format is B, C H, and W, where B is batch, C is channel, H is height, and W is width.

3.2.2 Data Quantization

The feature of the primary optimization of the toolkit is uniform quantization. Generally, this technique supports the arbitrary number of bits greater than two, representing the

activations and weights. By low precision, it is implied that the inference of DL models in the precision lower than 32 bits, such as FLOAT16. The lower precision models, known as quantized models, substantially speed up the inference [32]. The pre-trained models were trained in floating-point precision and then transformed to integer representation with floating quantization operation between the layers.

Throughout the quantization procedures, the operations of *FakeQuantized* are inserted automatically into the model graph based on a pre-defined hardware target in case of producing the most optimized model, which is hardware friendly. Later on, different quantized algorithms could tune the parameters of *FakeQuantize* or remove some of the operations in case of meeting accuracy criteria. Finally, the final *FakeQuantized* models could be interpreted and transformed into the real lower precision model at the runtime of getting real improvement based on performance [29].

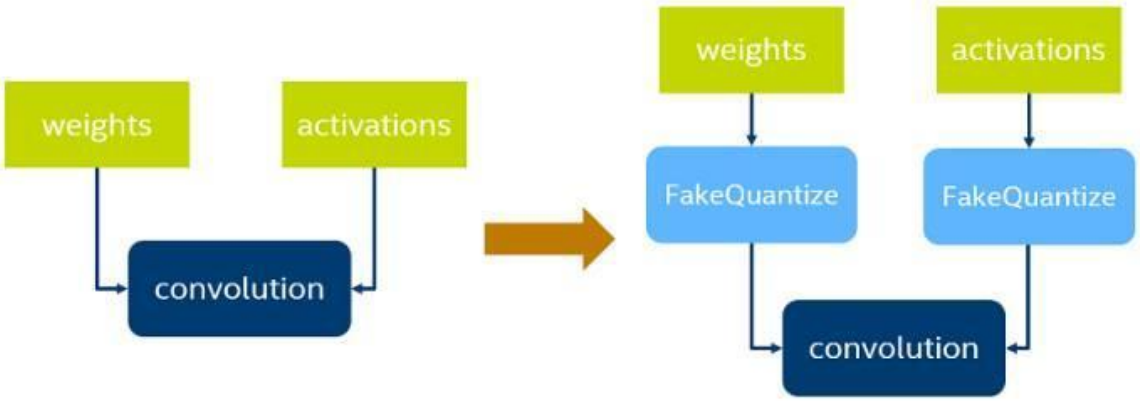

Figure 6: *FakeQuantize* Diagram [29]

The toolkit provides numerous auxiliary and quantization techniques, which aid in regaining accuracy after quantizing weights and activations. In addition, one can use independent optimization pipelines created by algorithms to quantize a model. For DNN model quantization, the only two quantization techniques for low precision that have been confirmed and approved are as follows:

***DefaultQuantization*:** It has been utilized as the default method to get faster but, in most cases, very accurate outcomes for low-precision quantization. The algorithm has three methods which are applied sequentially to a model. The methods are *ActivationChannelAlignment*, *MinMaxQuantization*, and *FastBiasCorrection*. The *ActivationChannelAlignment* method allows the model to align ranges of the convolutional layer's output activations with limiting the quantization error. The *MinMaxQuantization* is the vanilla quantization method that spontaneously inlays *FakeQuantize* operations into the model graph based on the specified target hardware platform. Finally, the *FastBiasCorrection* adjusts biases of convolutional and fully-connected layers based on the quantization error of the layer to make the overall error unbiased [30].

Figure 7: *DefaultQuantization* Algorithm [30]

***AccuracyAwareQuantization*:** This algorithm is meant to perform a low-precision quantization and renders a model to stay in the pre-defined range of accuracy drop. Compared to the *DefaultQuantization* algorithm, it causes a degradation in performance. The reason is that some layers can be reverted to the original precision. As this algorithm requires much time for quantization, it wasn't utilized [31].

The common workflow discussed below is assigned to the deep learning framework. It consists of three elements: Model Enabling, Post Training Quantization and Quantization Aware Training [32]. The Post Training Quantization is the easiest way to get the optimized models. The latter could be considered an addition or alternative when the first does not provide appropriate outcomes. All three steps are briefly discussed below.

Model Enabling: For this step, the user must ensure that the trained model on the targeted dataset can be inferred successfully with the OpenVINO Inference Engine in the floating-point precision. This procedure includes utilizing the model optimizer tool to

convert the model from the source framework to OpenVINO intermediate representation (IR) format and then running it onto Intel hardware platforms with the inference engine [32].

Post-Training Quantization (POT): As for the initial step. It has been suggested to utilize the low-precision quantization from the POT. In most cases, getting the accurate quantization model is possible as this stage does not require any model re-training. The sole thing required is to represent a dataset that is usually hundreds of images. It can also be utilized to gather statistics throughout the quantization procedure. Post-training quantization is fast, and it usually takes a few minutes. It depends on model size and HW [32]. The pre-trained *vgg16* CNN *Caffe* model is optimized to a 16-bit floating-point representation for this research.

Quantization-Aware Training (QAT): If the quantized model's accuracy does not satisfy the accuracy criteria, a step implies the QAT by utilizing the Open VINO-based training frameworks. At this stage, it can be assumed that the user has the original pipeline training of the model, which is written on *TensorFlow* or *PyTorch*. After the step, the user could get the accurate, optimized model, which could be converted to intermediate representation files using the model optimizer feature and inferred using the inference engine feature of OpenVINO [32]. Unfortunately, the *Caffe* framework doesn't support this component, so it wasn't utilized.

3.2.3 Batching and Streaming

The hardware accelerators are optimized for the huge compute parallelism, and so batching assists in saturating the leads and device to huge throughput. However, the streams would already assist in helping to hide communication overheads as well as certain bubbles in running and scheduling multiple kernels simultaneously that are not hardware efficient as compared to the kernel on several inputs at once. Moreover, batching should be leveraged for the throughput on the CPU and GPU. There are multiple primary techniques for utilizing the application performance, such as collecting direct inputs on the application side and sending batch requests to OpenVINO. It provides flexibility, as the approach doesn't need re-designing of logical application. In addition, it assists in sending individual inference requests with the batch size. In both of these cases, the size of the optimal batch

is particular to the device. As per the explanation, the optimal batch size also relies on the model, and other factors, such as inference precision [28].

Devices perform differently with different batch sizes. Along with the batch size, the stream number is another critical factor in improving performance. Multiple devices need different execution stream numbers to saturate. For some cases, stream combination and batching might be the requirement to enhance throughput. A single possible strategy of throughput optimization is the set upper bound for the latency and increase the batch size and the stream number until the tail latency has been met [28]. The optimal execution parameters and numbers can be found after running performance tests and ensuring overall performance validation

Usually, using a moderate (2-8) batch size in addition to the maximum number of streams on high-end CPUs may improve the performance. Typically, the GPU runs two requests per stream, so two streams can serve four requests. Also, in the case of GPU, the batching delivers better throughput for four and more requests (each with a small batch size like 2) [28]. For the above reason, the batch size is chosen to be two in this research.

It is imperative to keep these streams busy by running as many inference requests as possible. The number of streams should be less or equal to the number of requests to avoid wasting resources. It also helps to run the application simultaneously [28]. Therefore, four streams and inference requests are chosen to be four in this research.

CHAPTER 4

Experimental Results and Analysis

This chapter presents the results obtained from implementing the *vgg16 Caffe* CNN model on CPU and GPU using Intel OpenVINO. First, we describe the experimental setup used for hardware implementation. Then, the accuracy, latency, throughput, and execution time of the model are reported and analyzed in detail for both mentioned devices. Lastly, the performance comparison between CPU and GPU will be discussed.

4.1 Experimental Setup

The pre-trained *vgg16 Caffe* model was implemented on Intel Core i5-1035G1 CPU and Intel UHD graphics G1 GPU. Intel Core i5-1035G1 has four cores. It is part of the Core i5 line-up that utilizes Sunny Cove-U's architecture with BGA 1526 socket. Moreover, Intel Core i5-1035G1 has 6 MB of the L3 caches. It operates at 1000MHz by default but boosts up to 3.6 GHZ, which depends on the workload. Its highest memory speed is 3200 MHZ, but with overclocking, it goes even higher [33]. Moreover, its processor features integrated UHD graphics. The Intel UHD Graphics G1 is the integrated graphics card in Ice Lake SoCs. It offers 32 of 64 Execution Units (EU). It has no dedicated graphics memory, and its clock rate depends on the processor model. The GPU can operate at a frequency of 300 MHz, but it can boost up to 900 MHz [34].

4.2 Optimization Results

In this section, firstly, the accuracy of the model with single precision floating point and half-precision floating point representation will be discussed. Then, the performance results (including execution time, latency, and throughput) of the model with both floating point representations will be evaluated. The model has been optimized and implemented on CPU and GPU using Intel OpenVINO. A single image and the *ImageNet* 2012 dataset were used for validation separately.

4.2.1 Accuracy

The accuracy of each floating-point representation was tested by running the model on initial 50 thousand images of the *ImageNet* validation dataset. The model was implemented on both CPU and GPU using the Intel OpenVINO toolkit. The Top-1 and Top-2 accuracies

of the *vgg16 Cafee* model on the *ImageNet* dataset are approximately 71.59% and 89.88%, respectively [40]. The *vgg16 Caffe* CNN model's input height is 224 and input width is 224. In this research, the model's input height is changed to 200 and input width is changed to 200. The Top-1 and Top-5 accuracy measurements with the absolute and relative errors obtained after CPU and GPU implementation of both floating-point representations are given in Table 2 and Table 3, respectively.

	Single-Precision Floating Point		**Half-Precision Floating Point**	
Top-1 Accuracy	70.96%	Absolute Error: 0.006	64.64%	Absolute Error: 6.326
		Relative Error: 8.455e-05		Relative Error: 0.08914
Top-5 Accuracy	89.88%	Absolute Error: 0.002	86.21%	Absolute Error: 3.664
		Relative Error: 2.225e-05		Relative Error: 0.04077

Table 2: The Accuracy Comparison of *vgg16* CNN with *ImageNet* Dataset Model Running on Intel Core i5-1035G1 CPU

	Single-Precision Floating Point		**Half-Precision Floating Point**	
Top-1 Accuracy	70.95 %	Absolute Error: 0.022	64.63 %	Absolute Error: 6.34
		Relative Error: 0.00031		Relative Error: 0.08934
Top-5 Accuracy	89.86 %	Absolute Error: 0.018	86.19 %	Absolute Error: 3.69
		Relative Error: 0.0002003		Relative Error: 0.04106

Table 3: The Accuracy Comparison of *vgg16* CNN Model with *ImageNet* Dataset Running on Intel UHD Graphics G1 GPU

On CPU, the single precision optimized model achieves 70.96% top-1 accuracy and 89.88% top-5 accuracy, and the half-precision optimized model achieves 64.64% top-1 accuracy and 86.21% top-5 accuracy, resulting in 6.32% and 3.67% accuracy loss. On GPU, the difference between the single precision optimized model and the half-precision

optimized model on top-1 and top-5 accuracies are almost the same as CPU. The optimized models accuarcies are close to the *Caffe* model's (with original input size) accuarcies. Therefore, accuracies are satisfactory, and the models can be used for image classification applications.

4.2.2 Latency

The latency in image classification refers to the time delay between the input of an image and the output of the predicted class label by the model [41]. It is a measure of the response time of the model. The latency report of the model's single-point and half-point design implementation on CPU and GPU for a single image classification is shown in Figure 8, and the *ImageNet* dataset classification is shown in Figure 9. On CPU, for a single image, the latency of model's single-precision floating point design is 767.67 ms and the model's half-precision design is 592.58 ms. On GPU, the latency of model's single-precision floating point design is 472.56 ms and model's half-precision design is 303.72 ms.

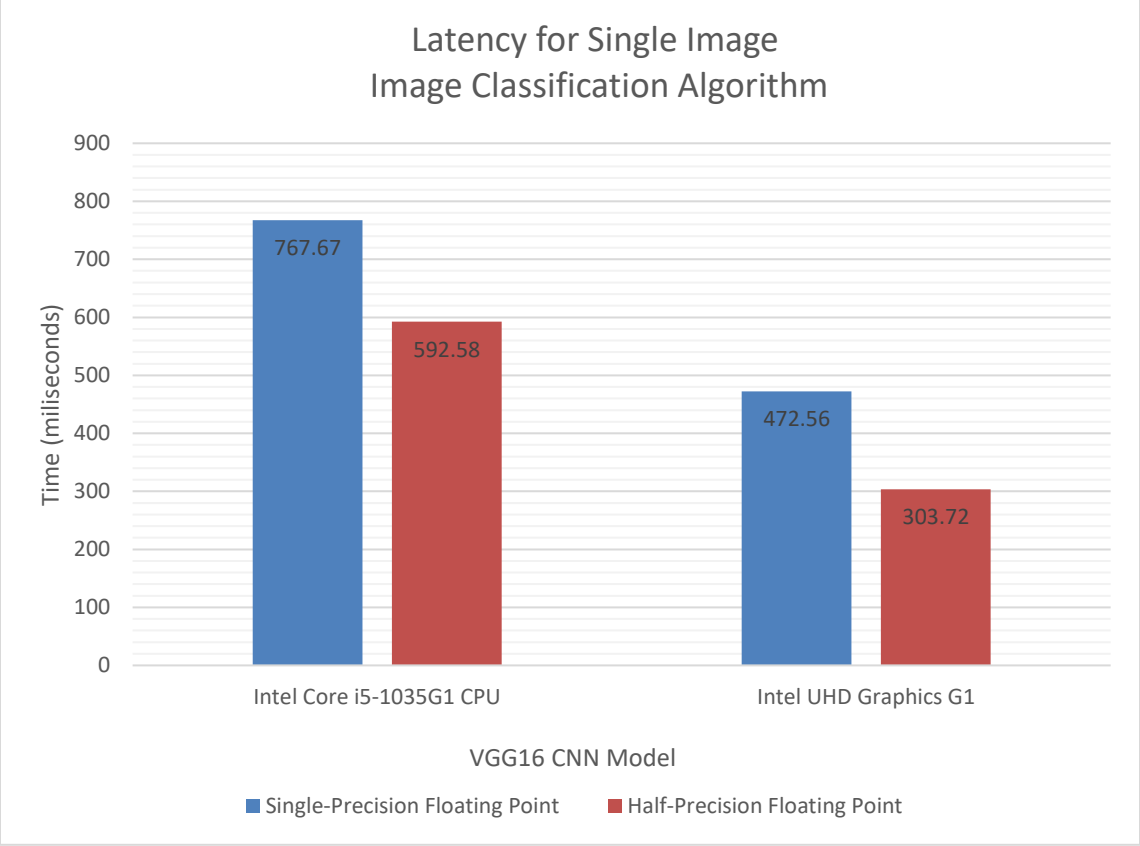

Figure 8: The Latency of the *vgg16* CNN Model with a Single Image Running on CPU and GPU

For the *ImageNet* dataset, on CPU, the average latency of the model's single-precision design is 721.2 ms, and the model's half-precision design is 689.9 ms. On GPU, the average latency of the model's single-precision design and the model's half-precision design is 444.2 ms and 261.19 ms, respectively.

From the illustrated figures, it can be said that the lower precision design shows less delay compared to the higher one. The GPU device also shows less latency measurement than the CPU device.

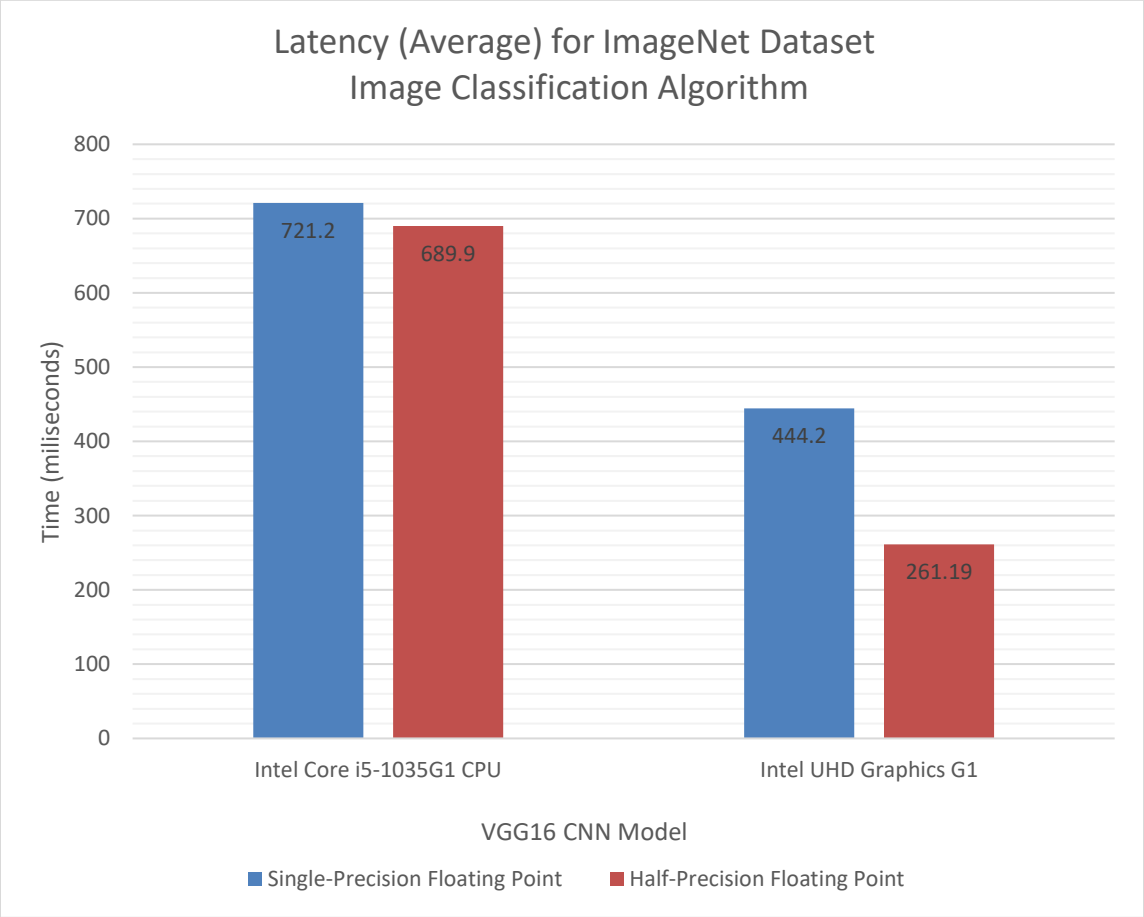

Figure 9: The Average Latency of *vgg16* CNN Model with the *ImageNet* Dataset Running on CPU and GPU

4.2.3 Throughput

The throughput in image classification refers to the number of images that can be processesed by a model in a given amount of time [42]. It is a measure of the model's speed. The throughput measurement of the model's single-point and half-point design

implementation on CPU and GPU for a single image classification is illustrated in Figure 10, and the *ImageNet* dataset classification is illustrated in Figure 11. On CPU, for a single image, the throughput of the model's single-precision floating point design is 5.21 frames per seconds (FPS), and the model's half-precision design is 6.73 FPS. On GPU, for a single image, the throughput of the model's single-precision floating point design is 8.46 FPS, and the model's half-precision design is 13.17 FPS.

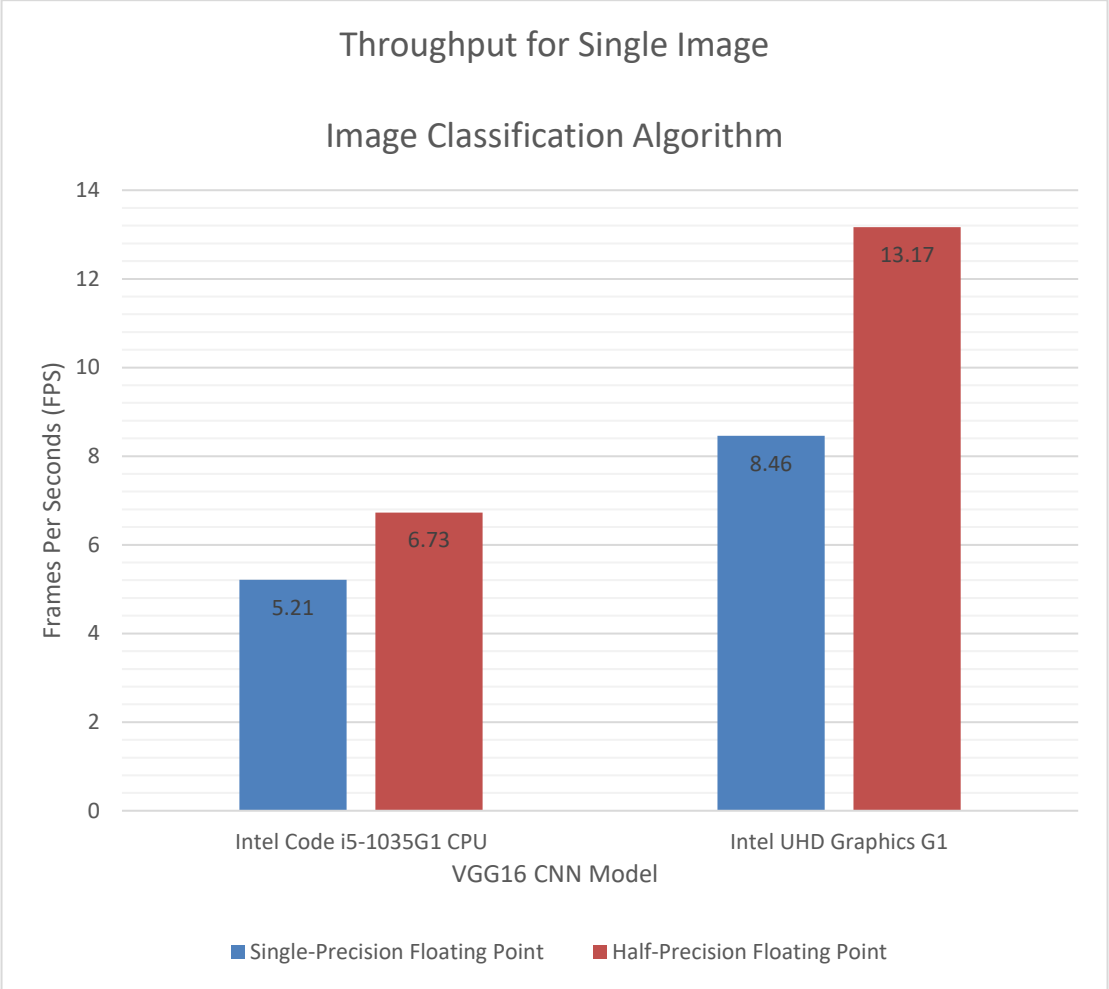

Figure 10: The Throughput Measurement of the *vgg16* CNN Model with a Single Image Running on CPU and GPU

For the *ImageNet* dataset, on CPU, the throughput of the model's single-precision design is 5.5 FPS, and the model's half-precision design is 5.7 FPS. On GPU, the throughput of the model's single-precision design and the model's half-precision design is 8.04 FPS and 11.19 FPS, respectively.

From the illustrated figures, it can be said that the lower precision design is faster compared to the higher one. Also, the GPU device is 1.5x – 2x times faster compared to the CPU device.

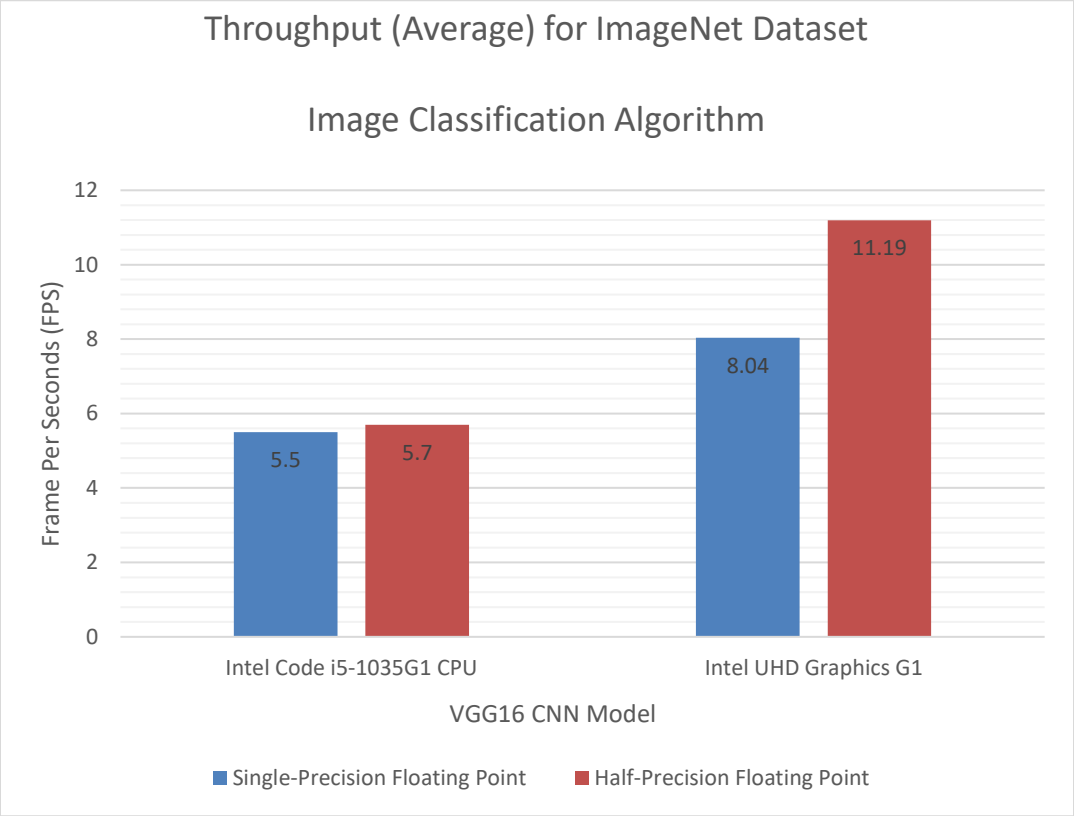

Figure 11: The Throughput Measurement of the *vgg16* CNN Model with the *ImageNet* Dataset Running on CPU and GPU

4.2.4 Execution Time

The execution time of the model for a single image depiction and the *ImageNet* 2012 dataset is depicted in Figure 12 and Figure 13. On CPU, for a single image, the single precision floating point model takes only 61.47 seconds to identify and 220.52 minutes to classify the whole dataset. The half-precision model takes 61.6 seconds to classify and 194.52 minutes to classify the whole dataset. The lower precision model is 1.4 – 11.8 percentage faster than the higher one. On GPU, the single precision floating point model takes only 61.44 seconds to classify an image and 98.78 minutes to classify the whole dataset. The half-precision model takes 60.45 seconds to identify an image and 45.76

minutes to identify the whole dataset. Again, the lower precision model is 1.6 – 53.7 percentage faster than the higher one.

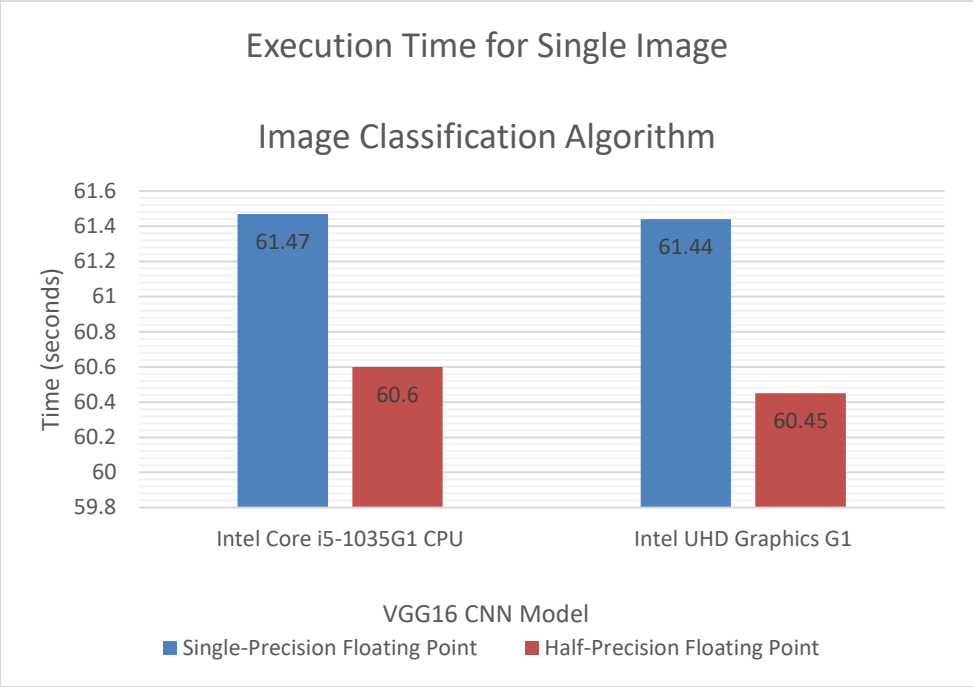

Figure 12: The Execution Time of the *vgg16* CNN Model with a Single Image Running on CPU and GPU

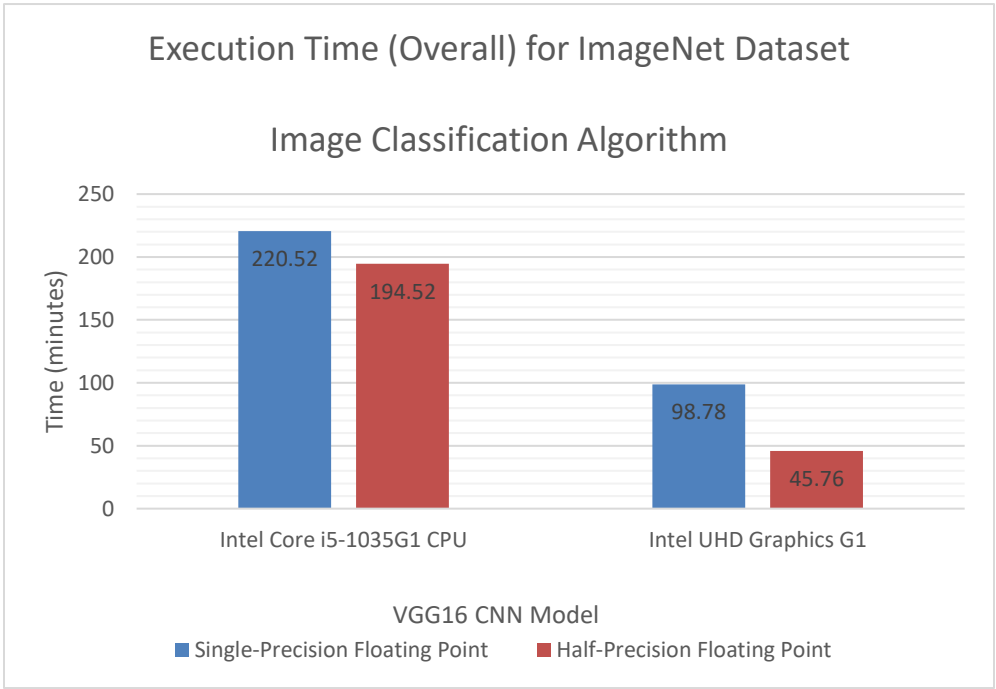

Figure 13: The Execution Time of the *vgg16* CNN Model with the *ImageNet* Dataset Running on CPU and GPU

4.3 Result Comparison

The model's single-point and half-point representation design is tested for a single image and the *ImageNet* dataset on CPU and GPU separately, and various performance measurements were extracted.

On CPU, the single precision optimized model achieves 70.96% top-1 accuracy and 89.88% top-5 accuracy, and the half-precision optimized model achieves 64.64% top-1 accuracy and 86.21% top-5 accuracy, resulting in 6.32% and 3.67% accuracy loss. The almost same thing can be said in the case of GPU implementation. Therefore, the accuracy performance for both CPU and GPU devices is practically identical. The latency report of CPU compared to GPU implementation of the model's both single precision and half-precision design demonstrated in Figure 8 and Figure 9 demonstrates the better performance of GPU. The minimum latency measurement is 261.19 FPS for the model's half-precision design to classify a single image implemented on GPU. Again, the average throughput measurement of CPU compared to GPU implementation of the model's single-precision and half-precision design demonstrated in Figure 10 and Figure 11 verify the better performance of GPU with at least 1.5x times faster execution of image classification algorithm. The execution time depicted in Figure 12 and Figure 13 also shows higher performance for GPU implementation of the model's lower-precious design.

In every measurement, the lower precision model performs better than the higher one. Also, the GPU device is 1.5x – 2x times faster than the CPU device for implementing image classification algorithm.

CHAPTER 5
Conclusion

5.1 Summary

Conventional neural networks are among the most popular artificial neural networks in image recognition and processing. However, a CNN model acceleration has attained massive importance in the deep learning community for real-time performance. Therefore, an appropriate accelerator is required to meet the various requirements of application engineers. In this work, the pre-trained *vgg16* CNN *Caffe* model inference performance is evaluated on Intel Core i5-1035G1 CPU and Intel UHD Graphics G1 GPU using Intel OpenVINO toolkit.

Intel OpenVINO toolkit facilitates the optimization of a deep learning model from a framework and deployment onto various Intel hardware platforms, including CPU, GPU, and FPGAs. The toolkit's Model Optimizer tool helps to accelerate inference by introducing data arrangement, quantization, batching, and other optimization features to the model. Furthermore, the toolkit's Benchmarking App aids in generating reports of the model's throughput and latency measurement running on Intel hardware, while the Accuracy Checker App validates the model's accuracy.

The pre-trained *Caffe* model was run using Intel OpenVINO on Intel hardware platforms, and a single image and *ImageNet* 2012 dataset were used for validation purposes. In this research, the highest latency was 767.67 ms for the model with FP32 while running on CPU to classify a single image. On the other hand, the lowest average was 261.19 ms for the model with FP16 while running on GPU to classify the *ImageNet* dataset. The research result showed the model with FP32 and FP16 runs on the GPU 1.6x-1.9x times faster than on the CPU for a single image. In the case of the *ImageNet* dataset, the model with FP32 and FP16 runs on the GPU average 1.5x-2x times faster than on the CPU. Moreover, from the figures, it seems that the execution time taken by the CPU is almost 2.2x-4.2x times higher than that of a GPU to classify the whole *ImageNet* dataset. Lastly, the top-1 and top-5 accuracies of the model's FP32 and FP16 design are almost identical for CPU and GPU.

To conclude, between CPU and GPU, the performance of GPU always remains higher. That's because GPUs have more cores than CPUs and high memory bandwidth, enabling them to handle massive workloads and conduct multiple functions through high parallelization [23]. Moreover, the low-precision model's significant increase in performance makes them appropriate for various applications. By providing outstanding acceleration without performance loss, the Intel OpenVINO toolkit has proven to be an excellent choice for deep learning researchers to deploy neural network models to perform predictions-based applications.

5.2 Future Work

Many types of research have been carried out, and software has been developed to deploy deep learning inference models on various accelerators. The Intel OpenVINO toolkit offers implementation of neural networks without needing additional modification or development of additional software. Moreover, It allows the execution of the machine learning models on broad range of devices, such as CPUs, GPUs, VPUs, and FPGAs.

Intel FPGA AI Suite is a deep learning Inference tool. The Intel FPGA AI Suite was a part of Intel OpenVINO as Intel FPGA DLA Suite. However, the DLA features were transitioned to Intel FPGA AI Suite. The tool supports Intel Agilex FPGA, Intel Cyclone 10 GX FPGA, and Intel Arria 10 FPGA [35]. It would be interesting to see the performance of the optimized neural network models on the supported FPGA devices.

The OpenVINO toolkit enables the heterogenous-platform deployment of compute vision problems based on image processing, machine learning, and deep learning. The model's performance can be accelerated through this process. Lastly, the model can be optimized using high-level synthesis techniques.

REFERENCES

[1] "Convolutional Neural Network." [Online]. Available: https://cezannec.github.io/Convolutional_Neural_Networks/ [Accessed: March 12, 2022].

[2] Toshi Sinha, Brijesh Verma, and Ali Haidar, "Optimization of Convolutional Neural Network Parameters for Image Classification," *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8285338.

[3] Y. Chen, K. Zhu, L. Zhu, X. He, P. Ghamisi, and Jon Atli Benediktsson, "Automatic Design of Convolutional Neural Network for Hyperspectral Image Classification," *IEEE Transactions on Geoscience and Remote Sensing*, IEEE, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8703410.

[4] X. Lei, H. Pan, and X. Huang, "A Dilated CNN Model for Image Classification," *IEEE Access*, IEEE, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8756165.

[5] M. Dhouibi, A. K. B. Salem, and S. B. Saoud, "Optimization of CNN Model for Image Classification," *2021 IEEE International Conference on Design & Test of Integrated Micro & Nano-Systems (DTS)*, IEEE, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9497988.

[6] "Intel Distribution of OpenVINO Toolkit." [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html [Accessed: June 01, 2022].

[7] "What is OpenVINO? - The Ultimate Overview in 2022." [Online]. Available: https://viso.ai/computer-vision/intel-openvino-toolkit-overview/ [Accessed: June 01, 2022].

[8] Albert Reuther, et al., "Survey of Machine Learning Accelerators," *MIT Lincoln Laboratory Supercomputing Center*, Lexington, MA, USA

[9] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A Survey of FPGA-based Neural Network Accelerators," *ACM Transactions on Reconfigurable Technology and Systems*, ACM, 2021. [Online]. Available: https://dl.acm.org/doi/10.1145/3289185.

[10] L. N. Huynh, R. K. Balan, and Y. Lee, "DeepSense: A GPU-based Deep Convolutional Neural Network Framework on Commodity Mobile Devices," *Proceedings of the 2016 Workshop on Wearable Systems and Applications*, ACM, 2016. [Online]. Available: https://dl.acm.org/doi/10.1145/2935643.2935650.

[11] C. B. Murthy, M. F. Hashmi, N. D. Bokde, and Z. W. Geem, "Investigations of Object Detection in Images/Videos using Various Deep Learning Techniques and Embedded Platforms—A Comprehensive Review," *Applied Sciences*, MDPI AG, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/9/3280.

[12] "Introduction to Supervised Deep Learning Algorithms!" [Online]. Available: https://www.analyticsvidhya.com/blog/2021/05/introduction-to-supervised-deep-learning-algorithms/ [Accessed: May 29, 2022].

[13] "Different types of CNN models." [Online]. Available: https://iq.opengenus.org/different-types-of-cnn-models/ [Accessed: May 29, 2022].

[14] "VGG Very Deep Convolutional Networks (VGGNet)-What you need to know." [Online]. Available: https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/ [Accessed: May 29, 2022].

[15] "ImageNet Large Scale Visual Recognition Challenge 2014 (ILSVRC2014)." [Online]. Available: https://www.image-net.org/challenges/LSVRC/2014/ [Accessed: May 29, 2022].

[16] A. Elhassouny and F. Smarandache, "Trends in Deep Convolutional Neural Networks Architectures: A Review," *2019 International Conference of Computer Science and Renewable Energies (ICCSRE)*, IEEE, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8807741.

[17] "Benchmark C++ Tool." [Online]. Available: https://docs.openvino.ai/2021.1/openvino_inference_engine_samples_benchmark_app_README.html [Accessed: June 01, 2022].

[18] "Deep learning accuracy validation framework." [Online]. Available: https://docs.openvino.ai/2020.1/_tools_accuracy_checker_README.html [Accessed: June 01, 2022].

[19] N. A. Andriyanov, "Analysis of the Acceleration of Neural Networks Inference on Intel Processors Based on OpenVINO Toolkit," *2020 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO)*, IEEE, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9166067.

[20] V. V. Zunin, "Intel OpenVINO Toolkit for Computer Vision: Object Detection and Semantic Segmentation," *2021 International Russian Automation Conference (RusAutoCon)*, IEEE, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9537452.

[21] F. Z. Guerrouj, M. Abouzahir, M. Ramzi, and E. M. Abdali, "Analysis of the Acceleration of Deep Learning Inference Models on a Heterogeneous Architecture based on OpenVINO," *2021 4th International Symposium on Advanced Electrical and Communication Technologies (ISAECT)*, IEEE, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9668607.

[22] A. Demidovskij *et al*., "Accelerating Object Detection Models Inference within Deep Learning Workbench," *2021 International Conference on Engineering and Emerging Technologies (ICEET)*, IEEE, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9659634.

[22] N. Andriyanov and G. Papakostas, "Optimization and Benchmarking of Convolutional Networks with Quantization and OpenVINO in Baggage Image Recognition," *2022 VIII International Conference on Information Technology and Nanotechnology (ITNT)*, IEEE, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9848757.

[23] A. Jayasimhan and P. Pabitha, "A comparison between CPU and GPU for image classification using Convolutional Neural Networks," *2022 International Conference on Communication, Computing, and Internet of Things (IC3IoT)*, IEEE, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9767990.

[24] Y. Liu, Y. Wang, R. Yu, M. Li, V. Sharma, and Y. Wang, "Optimizing CNN Model Inference on CPUs," *2019 USENIX Annual Technical Conference,* arXiv, 2019. [Online]. Available: https://arxiv.org/abs/1809.02697.

[25] M. Li *et al*., "The Deep Learning Compiler: A Comprehensive Survey," *IEEE Transactions on Parallel and Distributed Systems*, IEEE, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9222299.

[26] S. Mittal, P. Rajput, and S. Subramoney, "A Survey of Deep Learning on CPUs: Opportunities and Co-Optimizations," *IEEE Transactions on Neural Networks and Learning Systems*, IEEE, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9410437.

[27] "Setting Input Shapes." [Online]. Available: https://docs.openvino.ai/2022.1/openvino_docs_MO_DG_prepare_model_convert_model_Converting_Model.html#doxid-openvino-docs-m-o-d-g-prepare-model-convert-model-converting-model-1when-to-specify-input-shapes [Accessed: December 01, 2022].

[28] "Using Advanced Throughput Options: Streams and Batching." [Online]. Available: https://docs.openvino.ai/2022.2/openvino_docs_deployment_optimization_guide_tput_advanced.html#doxid-openvino-docs-deployment-optimization-guide-tput-advanced [Accessed: December 01, 2022].

[29] "Quantization." [Online]. Available: https://docs.openvino.ai/2021.4/pot_compression_algorithms_quantization_README.html [Accessed: December 01, 2022].

[30] "DefaultQuantization Algorithm." [Online]. Available: https://docs.openvino.ai/2021.4/pot_compression_algorithms_quantization_default_REA

DME.html#doxid-pot-compression-algorithms-quantization-default-r-e-a-d-m-e [Accessed: December 01, 2022].

[31] "AccuracyAwareQunatization Algorithm." [Online]. Available: https://docs.openvino.ai/2021.4/pot_compression_algorithms_quantization_accuracy_aware_README.html#doxid-pot-compression-algorithms-quantization-accuracy-aware-r-e-a-d-m-e [Accessed: December 01, 2022].

[32] "Low Precision Optimization Guide." [Online]. Available: https://docs.openvino.ai/2021.1/pot_docs_LowPrecisionOptimizationGuide.html#model_optimization_workflow [Accessed: December 01, 2022].

[33] "Intel Core i5-1035G1 Processors." [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/196603/intel-core-i51035g1-processor-6m-cache-up-to-3-60-ghz.html [Accessed: December 01, 2022].

[34] "Intel UHD G1 Graphics of 10th Gen Intel Core G1 Processors." [Online]. Available: https://laptoping.com/gpus/product/intel-uhd-g1/ [Accessed: December 01, 2022].

[35] "Intel FPGA AI Suite." [Online]. Available: https://www.intel.ca/content/www/ca/en/software/programmable/fpga-ai-suite/overview.html [Accessed: December 01, 2022].

[36] Huyuan Li, "Acceleration of Deep Learning on FPGA", MASc Thesis, University of Windsor, ON, Canada.

[37] F. Sultana, A. Sufian, and P. Dutta, "Advancements in Image Classification using Convolutional Neural Network," *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, arXiv, 2018. [Online]. Available: https://arxiv.org/abs/1905.03288.

[38] N. C. Thompson, K. Greenwald, K. Lee, and G. F. Manso, "The Computational Limits of Deep Learning," arXiv, 2020. [Online]. Available: https://arxiv.org/abs/2007.05558.

[39] M. Sharma and M. Khalid, "FPGA-based Hardware Acceleration of Machine Learning Algorithms" (Unpublished).

[40] "vgg16." [Online]. Available: https://docs.openvino.ai/2022.1/omz_models_model_vgg16.html [Accessed: July 01, 2022].

[41] "What is Latency in Machine Learning (ML)?" [Online]. Available: https://iq.opengenus.org/latency-ml/ [Accessed: August 01, 2022].

[42] "What is Throughput in Machine Learning (ML)?" [Online]. Available: https://iq.opengenus.org/throughput-ml/ [Accessed: August 01, 2022].

VITA AUCTORIS

NAME:	Md Maksud-Ul-Kabir Rico
PLACE OF BIRTH:	Dhaka, Bangladesh
EDUCATION:	American International University-Bangladesh, BSc, Dhaka, Bangladesh, 2015
	University of Windsor, MASc, Windsor, ON, Canada, 2023