Evaluation of Neural Network Performance with CPU, GPU and TPU using CUDA Functionality

Mansi Patel
Computer Science
University of Maryland
Baltimore County
MD, USA
mansip1@umbc.edu

Prajakta Narsay
Computer Science
University of Maryland
Baltimore County
MD, USA
pnarsay1@umbc.edu

Yashwanth Saladi
Computer Science
University of Maryland
Baltimore County
MD, USA
ux96332@umbc.edu

***Abstract*—The field of AI is evolving rapidly and there are many applications which tend to make machines more intelligent. Computer Vision is one of the problem domains under Machine Learning which is the sub field of AI. CV is related to making a machine see and perceive through graphical data. Making machines learn is a time-consuming task. This is because they need to get trained on thousands of data samples. Processing these samples takes a lot of time and hence underlying hardware architecture plays a vital role in decreasing the time taken to train the model. This report is based on comparison of different hardware architectures for training Neural Network for the classification task. It compares Central Processing Unit, Graphic Processing unit and Tensor Processing Unit architectures. Google cloud based TPU, CUDA enabled GPU and CPU are used for comparison.**

***Index Terms*—Neural network, CNN, convolutional neural network, hardware performance, TPU, CPU, GPU, CUDA**

I. Motivation

Today Artificial Intelligence (AI) has many applications in a variety of fields like satellite imagery, medicine, self-driving cars, smart cities, wireless communication, etc. Subfields of AI like Machine Learning(ML) and Deep Learning(DL) make machines smart enough to learn without human intervention and perform tasks accordingly. The implementation aspect for AI, ML and DL consists of a major architecture called Neural Network (NN).

The concept of Neural Networks is inspired from the human brain which can learn and understand the environment around itself. There are neurons which are similar to the data nodes in the Neural Network. The input layer is similar to the sensory motor nerves in humans from where the input will be fed to the brain for further processing. The output layer is also present in the network which gives the decision or result regarding the input according to the training given to the model. In between the input and the output layer there is a series of connected neurons which actually are similar to the processing unit. These neurons are present in the layers and these layers constitute the hidden layers in the network. More the hidden layers, the deeper is the network. Each weight in the layer has a weight associated with itself which processes the coming input to extract features. Hence after the set of weights or filters are applied, the feature map is produced from which the model learns about the input data. After every epoch or the forward pass, all the parameters including the weights, the learning rate, number of neurons to be included etc. have to be updated. The model converges when the change in weight is stopped. This means that if there is a huge amount of data, large processing is required for all the Machine learning applications. These tasks can be considered as jobs. The better the hardware to process these jobs, the less is the latency issue vehicle training a Deep Learning Model Or the Neural Network.

In a subset of Deep Learning known as Computer Vision(CV), the algorithms like Convolutional Neural Networks(CNN) are popular to classify the images into different classes. The task discussed in this report is ‘Image Classification’ wherein the machine is trained to identify images on its own. This is made possible by training the machine using labeled images. This is a kind of supervised learning modality. Here, the labels are provided and the algorithm provides supervision to learn. But data preprocessing, handling such a large data set, training model and hyper parameter tuning takes a lot of time overall for the model to give a good accuracy. The DL applications have now been used in real time as well. This increases the need to speed up the training process with having a good accuracy as well. At this point, Computer Architecture comes into the picture. Using good hardware which can help reduce the latency and increase efficiency is important. In this project, we analyze what kind of hardware will be needed for training the model to classify the images in the FASHION MNIST dataset. This dataset contains about 70000 images belonging to ten different classes. Analyzing how much time it takes to train the model to learn to classify the above data is the main objective discussed in the report.

II. Related Work

Deep learning model training is computationally intensive, and there is a growing trend in the industry toward hardware specialization to boost results. ParaDnn, a parameterized deep learning benchmark suite that generates end-to-end models for completely connected (FC), convolutional (CNN), and recurrent (RNN) neural networks, to systematically benchmark deep learning frameworks [1]. More computation, time, and resources are expended as a result of increased use of advanced deep neural networks. Companies must reduce training time and inference response time to meet the demands of these models. Growing research and development of application specific integrated circuits of different types has resulted as a result of these costs and demands [2]. A detailed empirical study on the performance and energy efficiency of several common off-the-shelf processors (i.e., Intel CPU, NVIDIA GPU, AMD GPU, and Google TPU) in training DNNs was conducted by benchmarking a representative collection of deep learning workloads, including computation-intensive operations, classical convolutional neural networks (CNNs), recurrent neural networks (LSTM), Deep Speech 2, and Transformer [3].

Advances in multicore processors and accelerators have paved the way for more machine learning approaches to be explored and applied to a wide range of applications. These advancements, combined with the reversal of several patterns, including Moore's Law, have resulted in a flood of processors and accelerators that offer even more computational and machine learning power.From CPUs and GPUs to ASICs, FPGAs, and dataflow accelerators, these processors and accelerators come in a variety of shapes and sizes [4]. Deep learning has recently become common in a wide range of fields. CNN has been used by fashion companies on their e-commerce to address a variety of problems, including clothing identification, quest, and recommendation. Image classification is a crucial step in both of these implementations [5]. VGGNet is a more recent CNN architecture that is considerably more reliable. The question of which neural network architecture performs best under which scenario has always existed. The output of VGGNet and CNN deep learning architectures is evaluated and metrics are compared using the Fashion MNIST dataset [6].

The accuracy of four common CNN models for categorizing MNIST-fashion data, LeNet-5, AlexNet, VGG-16, and ResNet, found that ResNet was the best fit for the dataset [7]. The paper shows how a convolutional neural network can be used to solve an image categorization challenge. The MNIST and FashionMNIST datasets were used to evaluate the CNN model's performance. Experiments were carried out using different activation functions, optimizers, learning rates, dropout rates, and batch sizes. The results reveal that the activation function, optimizer, and dropout rate that are chosen have an impact on the correctness of the findings [8]. The online fashion market is constantly expanding, and an algorithm capable of recognizing items can assist clothes retailers in better understanding the profile of potential customers. Artificial Intelligence techniques that can recognize and categorize human clothing are required, and can be utilized to boost sales or better understand customers [9].

The CUDA architecture from NVIDIA is a powerful framework for developing highly parallel programs. The CUDA programming model helps programmers to write scalable programs without having to learn a plethora of new programming constructs by offering basic abstractions for hierarchical thread organization, memories, and synchronization. Many languages and programs are supported by the CUDA architecture [12]. They have looked at NVIDIA's Compute Unified Device Architecture, which was created for its GPUs. It's a graphics library with a series of APIs that allows us to use the GPU to render graphics on the computer screen. They have addressed the benefits of CUDA over OpenGL, as well as the CUDA process flow and model description, which led to reviews of its applications [13]. Modern graphics processing units (GPUs) have sufficiently versatile programming models that knowing their success will aid in the design of future manycore processors, whether GPUs or not. Studying these GPUs is useful in understanding tradeoffs between memory, data, and thread level parallelism due to the combination of many, multithreaded, SIMD cores [14].

Originally designed for computer video cards, graphics processing units (GPUs) have emerged as the most powerful chip in a high-performance workstation. GPU architectures are "manycore," with hundreds of cores capable of running thousands of threads in parallel, in contrast to multicore CPU architectures, which currently ship with two or four cores [15]. The CNN-SVM model was able to obtain a test accuracy of 99.04 percent utilizing the MNIST dataset, according to empirical data. Using the same dataset, the CNN-Softmax, on the other hand, was able to attain a test accuracy of 99.23 percent. Both models were also tested on the Fashion-MNIST dataset, which is a more demanding picture classification dataset than MNIST and was just released [16]. A thorough performance comparison of CUDA and OpenCL has been carried out. They chose 16 benchmarks spanning from synthetic to real-world applications. They examine the performance discrepancies in depth, considering programming paradigms, optimization methodologies, architectural features, and underlying compilers. The results suggest that CUDA works well in most applications. The results reveal that CUDA performs at most 30% better than OpenCL for most applications [18].

Studying these GPUs is valuable in understanding tradeoffs between memory, data, and thread level parallelism due to the integration of numerous, multithreaded, SIMD cores. Despite the fact that modern GPUs have orders of magnitude more raw computing power than modern CPUs, many critical applications, even ones with a lot of data level parallelism, do not reach their full potential [19]. Using the technology of Graphics Processing Units (GPU) and the Compute Unified Device Architecture (CUDA) platform, several fields of knowledge are benefiting from the reduction of computation time. In the case of evolutionary algorithms, which are naturally parallel, this technique could be useful for executing tests that take a long time to complete [20]. In machine learning systems, training Artificial Neural Networks (ANN) is a time-consuming process. The back-propagation algorithm is implemented on CUDA, a parallel computing architecture developed by NVIDIA, in this paper. The process is simplified using CUBLAS, a CUDA implementation of the Basic Linear

Algebra Subprograms library (BLAS) [21].

CudaRF beats both FastRF and LibRF in the examined classification job, according to an experimental comparison between the CUDA-based method (CudaRF) and state-of-the-art Random Forests techniques (FastRF and LibRF) [22]. A parallel tree-based combinatorial search and two computer vision applications, object detection/classification and object localization/segmentation, were chosen as test cases. The enhanced runtime hardware is compared to a standard workstation and a powerful Linux server with 20 physical cores. The results reveal that, given the same resources, the performance achieved using this cloud service is comparable to the performance achieved using dedicated testbeds. As a result, this service can be used to accelerate not only deep learning, but also other GPU-centric applications [23].

The tremendous speed and accuracy come at the expense of energy consumption, which has previously been overlooked in CNN design. The energy requirements for training such data sets grows exponentially as the amount of data sets grows. Deep learning frameworks and algorithms that are both accurate and energy efficient are highly desirable.They investigated the power consumption and energy efficiency of a variety of well-known CNNs and training frameworks on CPUs and GPUs [24]. Machine learning on dedicated hardware is usually a disaster due to expense, obsolescence, and inadequate software. In comparison to a 3 GHz P4 CPU, they propose a generic 2-layer fully connected neural network GPU implementation that achieves over 3/spl times/ speedup for both training and testing [26].

They used a GPU hardware (1152 cores) and the TensorFlow software library to build their own computing platforms to test the performance in practice. To test GPU performance, they ran deep learning computations with different numbers of hidden layers in a multilayer perceptron [27]. This paper describes a method for inferring predictive models using supervised learning algorithms based on dynamic profile data acquired via instrumented general-purpose processor runs. The results show that a modest number of easily accessible features can accurately anticipate the amount of GPU speedups on two separate high-end GPUs for a set of 18 parallel benchmarks, with accuracies ranging from 77% to 90% depending on the prediction mechanism and situation [28]. In this paper, they offer a framework for training and classifying arbitrary Convolutional Neural Networks (CNNs) on the GPU in this research. Training and classification on the GPU is 2 to 24 times faster than on the CPU, depending on the network topology [29].

TPU is a machine-learning application-specific integrated circuit (ASIC) that has lately been employed to handle large-scale scientific computing issues. As a proof-of-concept, we use TensorFlow to reconstruct images on TPUs using the alternating direction method of multipliers (ADMM). The reconstruction is based on radial-trajectory, multi-channel, sparsely sampled k-space [32]. The authors created an adaptive approach and algorithm for image pre-processing, as well as models of convolutional neural networks for cytological and histological image categorization, in this research study. The computer experiments were carried out utilizing CUDA technology on central and graphics processors [33]. Convolutional Neural Networks outperform previous approaches in the literature due to advantages such as hidden feature extraction, parallel processing enabled by the parallel computing, and real-time operation. In addition, the suggested method employs Convolutional Neural Networks. The picture categorization method is carried out in this work utilizing a LeNe model [34]. Commodity graphics processing units (GPUs) and NVIDIA's Compute Unified Device Architecture were used in the parallel implementation. It optimizes work allocation and input/output transfers between the CPU and the GPU, taking full advantage of the GPU's computing capacity as well as shared memory's high bandwidth and low latency [35].

The GPGPU (General-purpose computing on graphics processing units) paradigm was born as a result of recent hardware advancements. As a result, the GPU (Graphics Processing Unit) is now used for general-purpose algorithms as well as graphics programming. The usage of CUDA (Compute Unified Device Architecture) for 2D and 3D image processing is discussed in this study [36].The GPU-accelerated CatBoost (GPU-CatBoost) technique for hyperspectral image classification is introduced and compared using various characteristics in this article. An ensemble version of GPU-CatBoost, the GPU-accelerated CatBoost-Forest (GPU-CatBF) technique, is being developed to improve classification performance in both accurate and efficient ways [37]. Analyzing satellite photos is a time-consuming computing task that requires real-time execution or automation. Parallelism techniques can be used to reduce the amount of time required analyzing satellite photos. Graphics Processing Units (GPUs) are currently a popular solution for reducing the execution time of a variety of applications at a reasonable cost [38]. To begin, we review the current state of the art in terms of accuracy and computation time trade-off for relevant neural networks. We also conduct a survey of important strategies that contribute to increased accuracy. We'll go over image augmentation approaches, show how different optimizers work, and talk about ensemble strategies. The pipeline and procedures are based on our competition experience. The pipeline and approaches were inspired by our participation in a competition in which we were able to come up with a highly competitive solution and place fourth. Our technique is unique in that we merely employed the free Google Colab computation service and outperformed numerous more computationally intensive alternatives [39].

III. Design

A. Artificial Intelligence

AI can be defined as the study of intelligent agents, meaning systems which are able to pursue what is going on in its environment, analyze it and take actions towards maximizing the chances of successfully achieving or completing its tasks. It basically leads towards automation. For a system to achieve this level of intelligence and consciousness about its surroundings, it needs to learn first. This learning is achieved by training the system through various scenarios or examples.

Humans have been obsessed with automation since the very beginning of the technology. AI consists of different methods to help machines to learn, understand and act in a way which is similar to human intelligence. The basic AI methods include heuristic search methods, min max optimization problems, etc. Machine Learning is the subfield of AI which contains methodologies that use Statistical learning algorithms. The ML subsystem automatically learns and improves the performance on the basis of its experience. Hence the training process consists of the learning through experience, comparing the results, calculating the loss and then working towards minimizing the loss. The implementation aspects of the ML consists of the model. This model is actually referred to as a neural network. The neural network is built or constructed in a similar way to the human brain and hence it is called the neural network. The machine learning algorithms are mainly classified into three categories: Supervised Learning, Unsupervised Learning and Reinforcement learning.

When it comes to classifying the data, it is important to have a linear data structure so that the classification becomes easy. Many ML algorithms work very well on the linear data. But not every data set needed in the real world is fully linear in nature. In fact, the data generated today is highly non-linear in nature. For classifying such data and enabling the model to learn through this non-linearity, it is important to have a more complex structure of the Neural Network than what is used in many ML algorithms. This brings into picture the sub part of ML which is Deep Learning (DL). Here the architecture of the neural Network is more complex in terms of the layers used. Deep learning uses more hidden layers, neurons, connections, parameters and even more training data. Hence, the model requires more time to train and also the hardware needs to support the parallelism in order to process more jobs at once. Today there are many applications where DL can be used and it has its own sub-fields, like Computer Vision, Natural language processing, etc. Since the technology is booming, it has many real time applications. These real time applications need precision in results along with no latency in the results. All these applications are dependent on the hardware and the Deep Learning algorithms. Hence, it is important to have a perfect combination of Deep Learning and the hardware to get the desired output.

B. Images

As discussed before DL has many subfields. One of them is Computer vision, which deals with teaching machines and making them understand to visualize. It mostly deals with teaching or training machines to 'see'. This task is somehow similar to that in humans. The dataset here mostly contains the images or the pixelated values of images and their labels. These labels define which class the particular image belongs to. Various problems like self driving cars, face recognition, Age and gender detection, etc comes under Computer Vision. The algorithms require preprocessing the images. So Image processing techniques like applying various filters, edge detection methods, spatial filtering techniques, using histograms, etc are used to prepare the dataset that is required and suitable for training the model. Actually, the filters are used to extract the information from the images as features. These feature maps are used by the model to understand on what basis to classify the images and into which class.

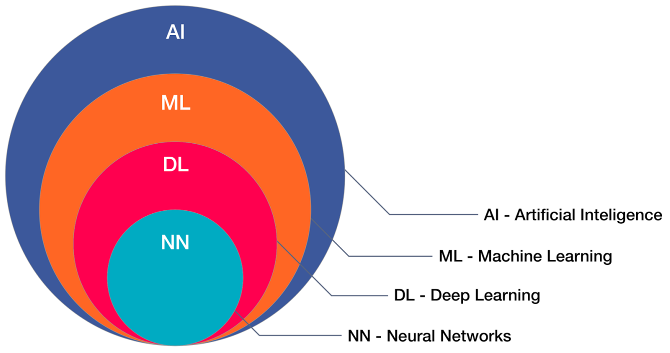


Fig. 1. Relation of AI, ML, DL, NN.

C. Computer Architecture

1) CPU: Central Processing Unit (CPU) executes instructions comprising computer programs. It basically performs arithmetic, logic, controlling, and input/output (I/O) operations given by the instructions in the specific program. The major components of the CPU are Arithmetic Logic Unit (ALU), Control Unit (CU) and Memory Unit (MU). ALU performs all the integer arithmetic and bitwise logical operations. The data words to be operated on (known as operands), status information from previous operations, and a code from the control unit indicating which operation to conduct are all inputs to the ALU. CU is the component of the CPU that directs the operation of the processor. It instructs the computer's memory, arithmetic and logic unit, and input and output devices on how to respond to the processor's commands. By delivering timing and control signals, it directs the operation of the other components. The CU is in charge of most computer resources. It controls data flow between the CPU and the rest of the system. A memory management unit (MU) is found in most high-end microprocessors, and it converts logical addresses into physical RAM addresses. Memory protection and paging capabilities are also included. They are useful for virtual memory. MMUs are typically absent from simpler processors, such as micro-controllers.

The majority of CPUs are synchronous circuits, meaning they use a clock signal to time their sequential operations. An external oscillator circuit provides a constant number of pulses per second in the form of a periodic square wave to generate the clock signal. The rate at which a CPU executes instructions is determined by the frequency of clock pulses; thus, the quicker the clock, the more instructions the CPU will execute per second. The clock period is larger than the maximum time required for all signals to propagate (travel) through the CPU in order to ensure proper function of the CPU. It's feasible to construct the entire CPU and the way it transports data around the "edges" of the rising and falling clock signal by setting the clock period to a value considerably above the worst-case propagation latency. This has the benefit of considerably simplifying the CPU, both in terms of design and in terms of component count. However, the entire CPU must wait on its slowest pieces even though some parts are much faster. Various methods to increase CPU parallelism should be used to overcome this issue.

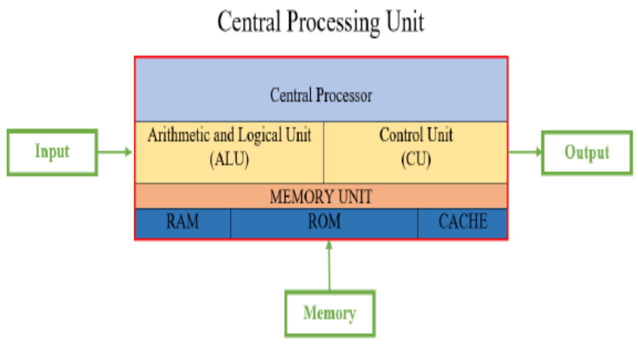


Fig. 2. Central Processing Unit (CPU)

CPU can execute one instruction at a time.So, the entire CPU must wait for that instruction to complete before it executes the next instruction. Sometimes, the CPU gets stuck when some instruction takes more than one clock cycle to complete the execution. There are two methods to achieve parallelism in CPU. 1) Instruction level parallelism (ILP) and 2) Task level parallelism (TLP). ILP increases the rate at which instructions are executed within a CPU and TLP increases the number of threads(processes) that a CPU can execute simultaneously.

2) GPU: GPU is the processor which is made up of many small numbers of specialized cores. These cores are designed to work in collaboration to provide a good throughput. This is a kind of critical computing chip. Because of its architecture, it is used in 3D rendering computational tasks. Today, visuals are required in many applications like computer vision tasks, image processing, animations, graphics, video games, etc. GPUs have evolved over time and are now parallel processors to handle these tasks. Deep Learning applications related to graphical computations need GPUs for faster processing and better results. For Deep Learning training, which needs training and in this case which is trained with 70000 images, the GPU accelerators play a vital role in processing the data. Additionally, GPU provides better Memory Bandwidth than CPUs. Training is an important task in the DL, which actually decides how accurately the model can learn and manage doing its tasks with higher precision. Hence, it requires a huge amount of data on which the model can train itself. These training on the data can be studied as an analogy with the jobs in traditional computing. Hence, to process these jobs, large computational operations are needed. GPUs provide these computations efficiently with parallel processing. Also, GPUs can be present on two positions on the die. This can be either situated on the motherboard or on the graphic card. This helps in throughput maximization since GPU's working focuses on utilizing all cores for performing a task.

The GPU used for the project in this report, was 12GB NVIDIA Tesla K80 GPU. This is a cloud-based GPU.This was used because this project aims at comparing the different approaches and by using all three architectures from the cloud, it is easy to compare without any bias. 12GB NVIDIA Tesla K80 GPU combines two graphic processors which supports dynamic parallelism. It combines 24 GB of memory with fast memory bandwidth and leading compute performance for single and double precision workloads. Equipped with the latest NVIDIA technology, the Tesla K80 monitors GPU usage to maximize throughput.

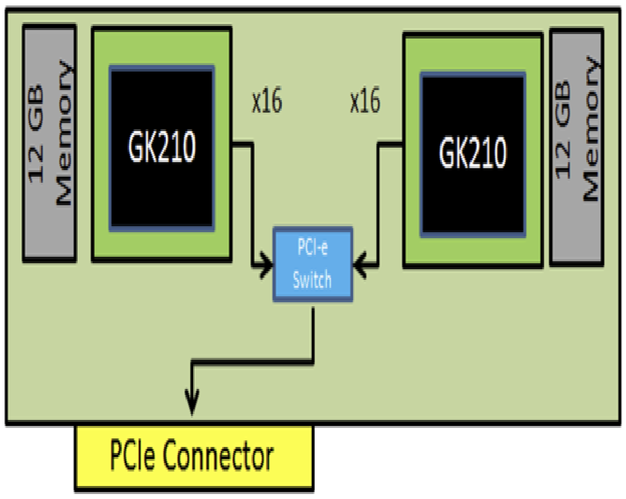


Fig. 3. Graphics Processing Unit (GPU)

3) TPU: Tensor Processing Unit was developed by Google and it belongs to the category of Application Specific Integrated Circuits (ASICs). This basically means, the development and use of TPU is specific to the task where tensors are used to process the data. They are designed to benefit the machine learning tasks. So, Google also came up with the library called Tensorflow. The TPU service from Google is available to the users through Google Cloud. It enables running ML workloads on TPUs using Tensorflow.

There are different TPU versions available on the google cloud. Each TPU version has specific hardware characteristics of the specific TPU device. TPU architecture consists of a TPU core, part of High- Bandwidth Memory (HBM) for each

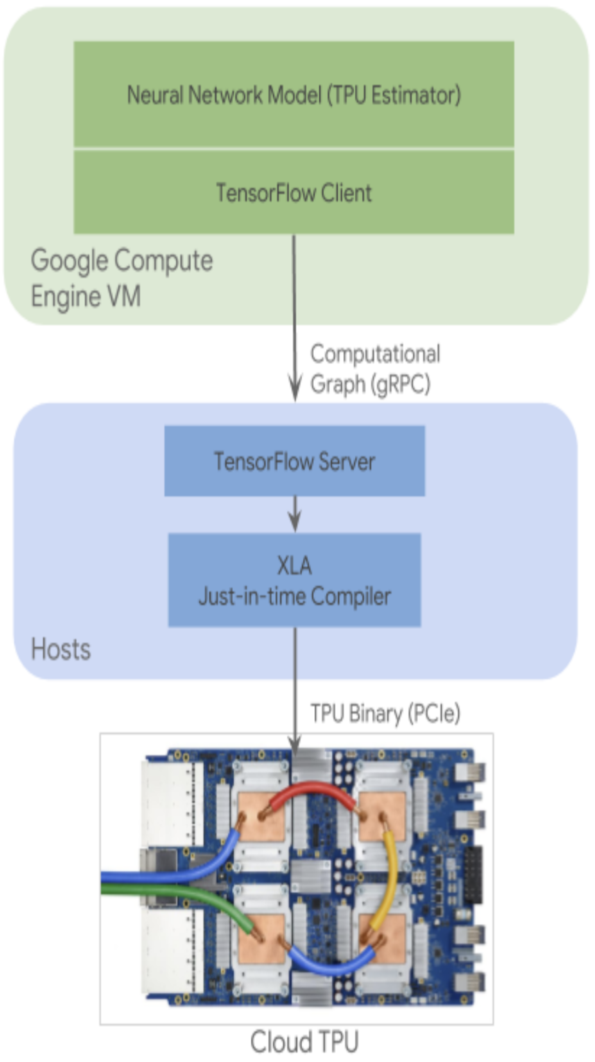


Fig. 4. Tensor Processing Unit (TPU)

core and the networking interface unit. The network interface unit is available for inter device communication. Each TPU core has various components like scalar, vector, and matrix units (MXU). These MXUs provide computation power in TPU chips. Addition to that, each MXU can perform 16K multiplication operations in each cycle with reduced bfloat16 precision value. Each core on a TPU device executes computations independently. The characteristic of High Bandwidth allows TPU chips to connect directly. There are different configurations in which TPUs are available at the Google Data center. They are: Single Device TPUs and TPU pods. Single device TPUs are individual TPUs which have high bandwidth and are dedicated in nature.

One of the important parts in TPU architecture is TPU Pods. A TPU pod has multiple TPUs connected to each other with a dedicated high speed network. The host in the user's TPU node distributes the workload to all TPU devices. This is a very important contribution which leads to the faster training of the ML model.

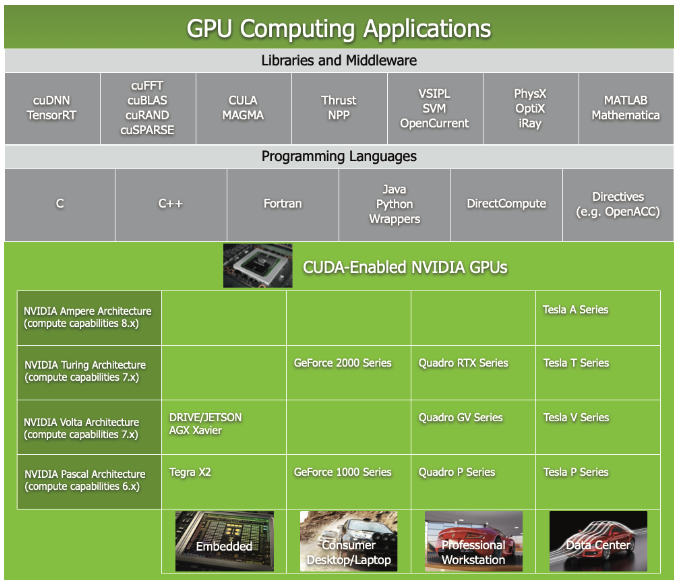
There are various TPU types that correspond to different versions. For example: The version 2 TPU pod has configuration of 64 devices with total 512 TPU v2 cores and 4 TiB of memory. The v3 TPU Pod provides a maximum configuration of 256 devices for a total 2048 TPU v3 cores and 32 TiB of TPU memory. Different versions are available and accessible to different areas of the world, since that's a cloud based service, it can be region specific. The zones where the Google cloud and Cloud based TPUs are present are the United States, Europe and Asia Pacific. The configurations of the cloud based TPU changes according to these zones.

Figure 4 shows the software architecture of Cloud based TPU consists of the following components:

1) TPU Estimator: It is basically a high level API which is developed on the Estimators and they extract maximum throughput and hence help in maximizing the performance.
2) TensorFlow Client : The Estimator converts the software programs into TensorFlow Operations. These operations are further converted to computational graphs by TensorFlow Client. It further sends the graphs to TensorFlow Server.
3) TensorFlow runs on Cloud TPU server. Server performs following actions after receiving computational graphs from the clients:
    a) Loading input from Cloud storage.
    b) Partition and separate the graph into different portions which then execute on Cloud TPU and then run on the CPU.
    c) Generating XLA operations of the sub graphs generated in the previous step. The graphs will get executed on the Cloud TPU.
    d) Invoke the XLA compiler to compile the operations in the subgraphs which contain tensors.
4) XLA Compiler : This compiler takes High Level Optimizer (HLO) operations as input from the server. Then, it generates the binary code which executes on cloud based TPU. The generated Binary is fed on Cloud TPU and then it is launched for execution. The system architecture supports the operations on the tensors exclusively. This helped the machine learning programs to train the model faster than other traditional computer hardware.

D. CUDA

NVIDIA announced CUDA in November 2006, a general-purpose parallel computing platform and programming model that takes advantage of the parallel compute engine in NVIDIA GPUs to tackle many complex computational tasks faster than a CPU. Developers can utilize C++ as a high-level programming language with CUDA because it comes with a software environment. Other languages, application programming interfaces, or directives-based techniques, such as FORTRAN, DirectCompute, and OpenACC, are supported, as shown in Figure .... . Because of the introduction of multicore CPUs and manycore GPUs, modern processing chips are now parallel systems. The issue is to create application software that transparently extends its parallelism to take advantage of the growing number of CPU cores, just how 3D graphics programs adapt their parallelism to manycore GPUs with widely changing core counts.The CUDA parallel programming model was created to address this issue while preserving a short learning curve for programmers who are already familiar with mainstream programming languages like C. Figure 5 shows GPU computing applications.


GPU Computing Applications

Libraries and Middleware

cuDNN TensorRT	cuFFT cuBLAS cuRAND cuSPARSE	CULA MAGMA	Thrust NPP	VSIPL SVM OpenCurrent	PhysX OptiX iRay	MATLAB Mathematica

Programming Languages

C	C++	Fortran	Java Python Wrappers	DirectCompute	Directives (e.g. OpenACC)

CUDA-Enabled NVIDIA GPUs

NVIDIA Ampere Architecture (compute capabilities 8.x)				Tesla A Series
NVIDIA Turing Architecture (compute capabilities 7.x)		GeForce 2000 Series	Quadro RTX Series	Tesla T Series
NVIDIA Volta Architecture (compute capabilities 7.x)	DRIVE/JETSON AGX Xavier		Quadro GV Series	Tesla V Series
NVIDIA Pascal Architecture (compute capabilities 6.x)	Tegra X2	GeForce 1000 Series	Quadro P Series	Tesla P Series
	Embedded	Consumer Desktop/Laptop	Professional Workstation	Data Center

Fig. 5. GPU Computing Applications (Source: NVIDIA)

Three important abstractions are at its core: a hierarchy of thread groups, shared memories, and barrier synchronization, all of which are simply exposed to the programmer as a small set of language extensions. Fine-grained data parallelism and thread parallelism are provided by these abstractions, which are nested within coarse-grained data parallelism and task parallelism. They instruct the programmer to divide the problem into large sub-problems that can be addressed in parallel by blocks of threads, and each sub-problem into finer parts that can be tackled cooperatively in parallel by all threads within the block. By allowing threads to participate when solving each sub-problem, this decomposition preserves language expressively while also enabling automatic scalability. Indeed, any block of threads can be scheduled on any of the GPU's available multiprocessors in any sequence, concurrently or sequentially, such that a built CUDA program can run on any number of multiprocessors, as shown in Figure ...., and only the runtime system has to know the physical multiprocessor count [31].

Google colab has been used for training the model on different hardwares. GPU on Google Colab has been enabled to train the model using CUDA functionality. Here we are using NVIDIA-SMI 465.19.01 CUDA with Tesla T series Version: 11.2 to train our model with GPU using CUDA functionality.

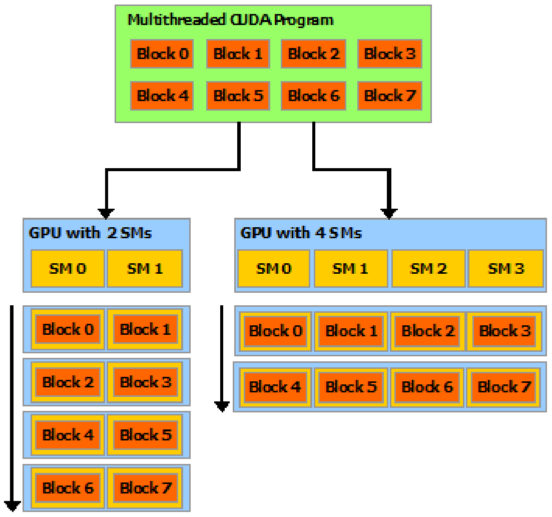


Fig. 6. Automatic Scalability (Source: NVIDIA)

E. Convolutional Neural Network (CNN)

Convolutional neural network is a class of Neural Network which specializes in processing the graphics and image data. Convolutional Neural Networks are made up of convolutional layers. These layers share learnable weights which are called kernels or feature maps. These weights are the weights to the neural networks.The algorithm is based on performing the convolution operation on the input data. These convolution operations are performed on the matrices.

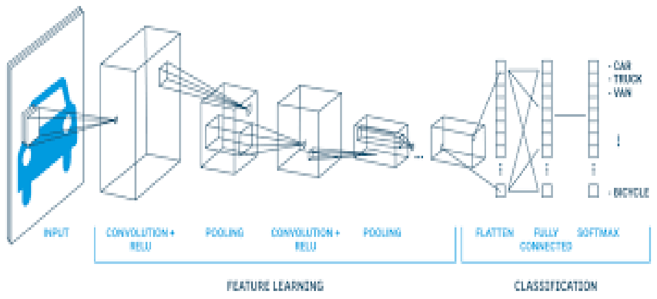


Fig. 7. CNN Architecture

The images used are in the form of pixelated values on which the convolution can be performed. These layers do convolution computation to the image data. The CNN also has pooling layers, activation functions , flatten and dense layers as referred in above figure. The pooling layer downsample along the length and height of the convoluted image. This pooling can be done either selecting the maximum value or averaging or minimum value of the pooling length and height.

$$n_{out} = [\frac{n_{in} + 2p - k}{s}] + 1$$

$$n_{in}: \text{ number of input features}$$

$$n_{out}: \text{ number of output features}$$

$$k: \text{ convolution kernel size}$$

$$p: \text{ convolution padding size}$$

$$s: \text{ convolution stride size}$$

The flatten layer converts the two dimensional arrays to one dimensional arrays; these layers are generally connected to dense layers. The end layers of the CNN will softmax layers it normalizes the input into probability distribution of possibilities of every class. The convolutional neural network is used for classification and segmentation. Convolutional neural networks helped in advancing computer vision applications. The convolutional neural networks are also called ConvNets. There are many predefined architectures readily available like AlexNet and GoogeNet. One needs to use these model architectures and readily train the model. This way it helps in reducing the in model building and these architectures also proved to work better for certain applications.

$$n_{out} = [\frac{n_{in} - k}{s}] + 1$$

$$n_{in}: \text{ number of input features}$$

$$n_{out}: \text{ number of output features}$$

$$k: \text{ convolution kernel size}$$

$$s: \text{ convolution stride size}$$

The convolutional neural networks are computationally expensive, due to increase in hardware capabilities and multi thread processing the use of CNN has been increased nowadays. It takes four to five times faster to train the model in GPU than CPU. Due to the available computational power recently we are seeing that there is an increase in applications of computer vision and we see self-driving cars hitting roads and increased surveillance systems with no man intervention. All these are because of Convolutional neural networks

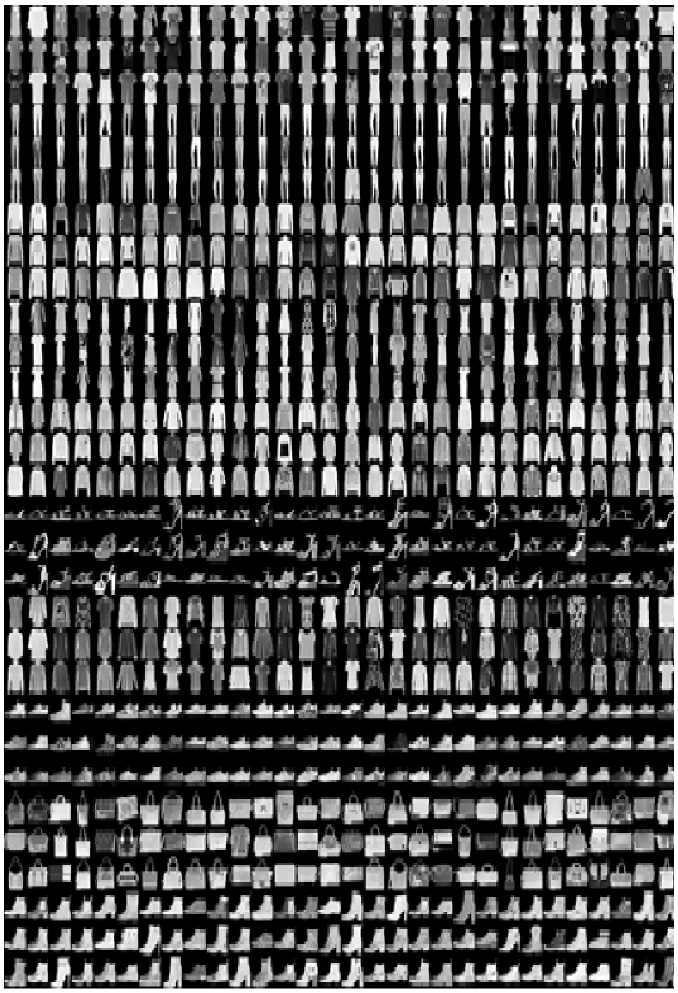


Fig. 8. Dataset

F. Dataset

For our work we are going to use the Fashion- MNIST dataset. This dataset have grayscale images of size (28 * 28) with 60,000 training set and 10,000 testset. This dataset is part of Zalando article images. These images are labeled into 10 classes. T-shirt/Top, Coat, Shirt, Trouser, ankle boots, Sneakers, Pullover, Bag, Sandel, Dress are the 10 class labels. This dataset is used as a benchmark dataset for many machine learning and deep learning models. Many scientists consider if their model does not work on this dataset it won't work on any dataset.

This is not the original MNIST dataset. Actual dataset contains handwritten images of 28 * 28 size. But this dataset is very easy and it's been used as a dataset for many models. This handwritten dataset is not being used as it was not able to test modern complex models. So the Fashion dataset replaced the original dataset which was more complex than handwritten digits and it has been used as a benchmark dataset for many modern models.

MNIST fashion dataset is readily available with many standard ML libraries like sklearn, TensorFlow, Pytorch and many other libraries. Many people implemented many loaders in many other programming languages. This data set is under MIT licence copyrights.In our work we load this dataset from tensorflow library.

G. Model

The model we compare all three hardwares is the same. This model is Convolution neural network with two convolution layers. Along with convolution layers we are using batch normalization layers, Dropout layers, max pooling, flatten and dense layers. First the input our model is feed into the Batch normalization layer. This layer normalises the input of this layer to 0 and 1. It means the input values are transformed between the 0 and 1 at the output. This will help us in getting rid of any outliers that try to influence the model performance.After batch normalization we and convolution layer. This layer creates convolution kernels which are feature maps for our model. After convolution the layer is connected to max pooling layers. The max pooling layer down-sample the input along the height and width by taking the maximum value. Figure 9 represents model architecture of convolutional neural networks.

After pooling we will be using a dropout layer. Dropout layer randomly selects a few input units and drops those units. It means the value of the inputs is zero. This is only done during training of models. Dropout layer reduces the dependency of all the layers and it will help increase the accuracy by decreasing the loss. The same set of layers will repeat for one more time. After that the flatten layer is applied. This layer converts the 2D input to one-Dimensional array. This array is further connected to dense layers and activation is applied. This is the architecture of the model we are comparing the hardwares. We evaluate the time for training this model.

H. Hyperparameter Tuning

Generally training of deep neural networks takes a lot of time. And there are many parameters like we have to configure. The tunable hyper parameters for CNN network are Kernel size in Convolutional network, dropout regularization, no. of units in hidden layers, pooling window size, Activation functions, learning rate, no.of epochs and batch size. Since for our work we will be concentrating on latency on different hardwares, so we don't be concentrating on hyperparameter tuning. Anyhow these parameters can be tuned let us discuss how each parameter is influencing the model and what parameters we will be using in our model.

Kernel size in the convolution layer will tell the convolution filter size of image the less the size the more features are extracted. For our model we will be using a kernel size of 5. If reduce the kernel size to thee there will be much more features will be extracted and learning parameters also increases. Dropout layer simply drops the units in the hidden layer. This prevents the model from overfitting. This dropout regularization parameter tells the fraction of units dropped during training of the model. For our model we used 0.3 dropout regularization for convolution layers and 0.5 regularization at the dense layer. Increasing the number of units in the hidden layer makes computation complex and takes a lot of time to train. If we reduce the no .of parameter the learnable parameters reduces and leads to model underfitting. So we should be wise while choosing the no. of hidden layers.

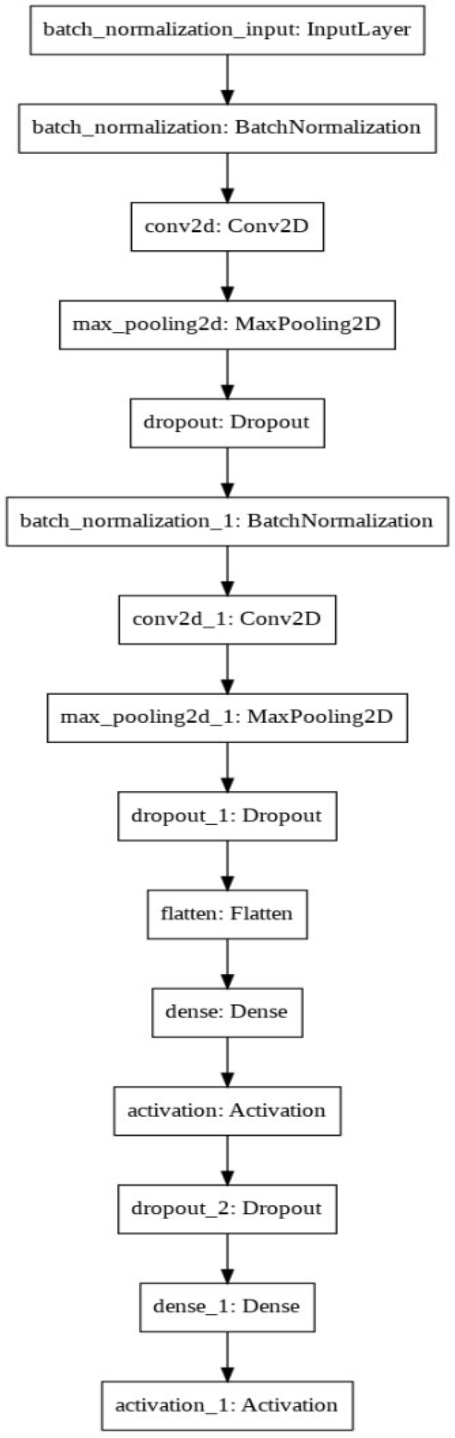

batch_normalization_input: InputLayer
batch_normalization: BatchNormalization
conv2d: Conv2D
max_pooling2d: MaxPooling2D
dropout: Dropout
batch_normalization_1: BatchNormalization
conv2d_1: Conv2D
max_pooling2d_1: MaxPooling2D
dropout_1: Dropout
flatten: Flatten
dense: Dense
activation: Activation
dropout_2: Dropout
dense_1: Dense
activation_1: Activation

Fig. 9. Model Architecture

In our model we started with 64 units in the first convolutional layer. From there we increased to 128 units in the next layer and finally we reduced it to 10 units as the output has 10 labels. There are different types of pooling layers for our model. We use a max pooling layer for our model with kernel size 2 and both horizontal and vertical stride of 2. If we increase the kernel size and stride units there is a chance that there will be loss of information.

Activation functions are used to introduce non-linearity to the model. There are many activation functions like ReLU, Sigmoid and Tanh Functions. ReLU which is a rectified linear unit which works better in cnn. So we are using the ReLU activation function in our model.

No. of epochs determines the no . of times the training data is passed into the model. We need to train the model till the test error and training error trade-off. For our model we will be using 15 epochs.

Batch size tells how many instances need to be trained for every mini batch. For our model 512 instances will be sent to training.

In our project, we will be comparing the hardwares. It means we will be comparing latency so we will compare how much time it takes to train the model. So we are not using any hypertunning, we just fix the parameters and train the model.

I. *Training*

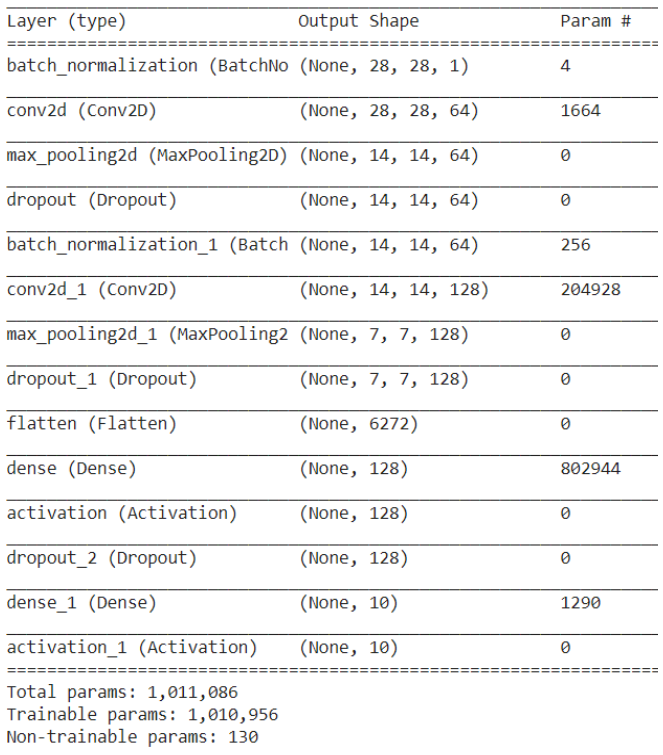
As discussed in above sections, we used 60,000 training images, the model has two convolution layers along with max pooling and dense layers.we trained our model for 15 epochs with batch size of 512. The model has learnable parameters over one lakh. We trained our model in google colab. With runtime environments such as TPU , GPU and CPU.

IV. Evaluation

We compared latency, train accuracy and test accuracy of Model in three different runtime environments. We tried to do all comparisons in the notebook but it's not possible as the kernel restarts when we run the program. This leads to loss of information variables. To eliminate this we had three different notebooks and each one is different run time. We also compared the time taken to train each epoch for the same batch size.

The Figure 11 represents Hardware vs train accuracy. As mentioned before, for the fair comparison, all three architectures were implemented on the Google Cloud. When it comes to training accuracy, the algorithm and the architecture of the model counts. Hence, 'ACCURACY' as a metric takes into account the hyperparameters used, the dataset on which the model is trained and the learning curve of the model. Hence there is no more difference between the accuracies of all three architectures.

Similarly, the Figure 12 depicts that the testing accuracy is almost the same and quite considerable. This means that the model was able to identify and classify the unseen examples correctly. This is an important aspect because it defines how



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization (BatchNo (None, 28, 28, 1)         4
_________________________________________________________________
conv2d (Conv2D)              (None, 28, 28, 64)        1664
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 14, 14, 64)        256
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 128)       204928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 128)               802944
_________________________________________________________________
activation (Activation)      (None, 128)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0
=================================================================
Total params: 1,011,086
Trainable params: 1,010,956
Non-trainable params: 130
```

Fig. 10. Model

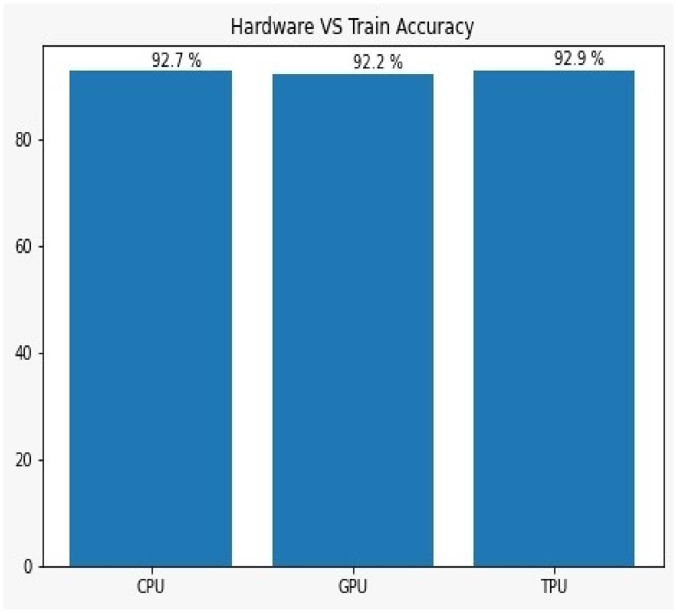


Fig. 11. Hardware vs Train Accuracy

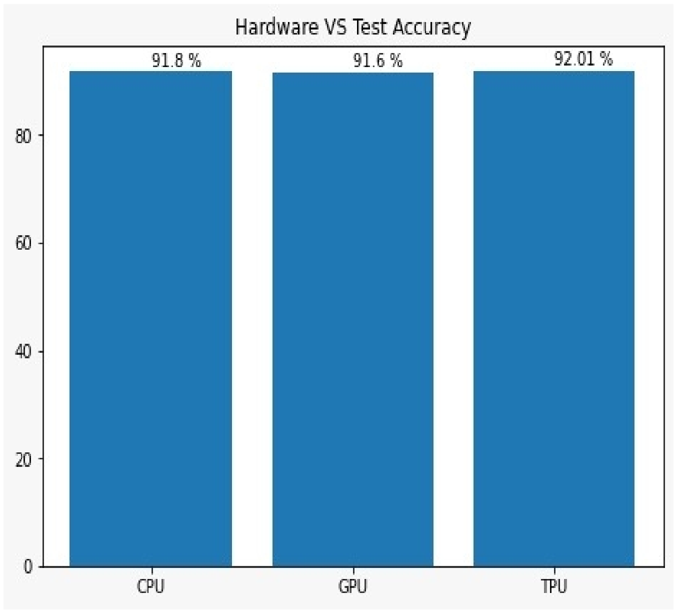


Fig. 12. Hardware vs Test Accuracy

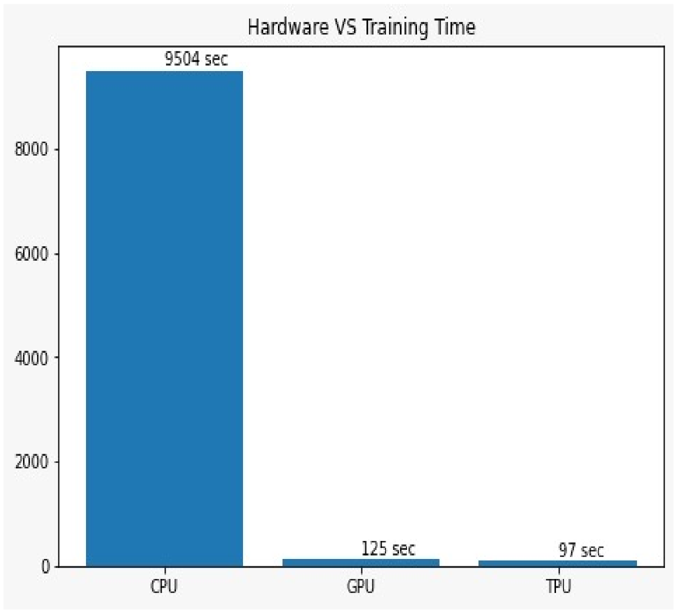


Fig. 13. Hardware vs Training Time

generalized the model is in terms of classifying the unknown samples.

The Figure 13 of hardware versus the training time is an important metric for evaluation with respect to this project. This is because, since the performance of the hardware is based on how fast it can process the data. In the Machine Learning problems, the most important task is to train the model and generalize it. This training is done on hundreds and thousands of data samples. Extracting features from all these samples , updating the weights, updating other parameters takes time. This task then becomes dependent on the underlying hardware used to process this data. The graph shows that the time required for TPU is the least. GPU also requires less time than CPU. This is because the code is implemented using Tensorflow and TPU is the specialized hardware to process the operations regarding the tensor datatype. With GPU, the CUDA functionality was enabled, the time taken was less than CPU.

When it comes to accuracy there not very distinguishable change can be observed. They almost have 91% accuracy as shown in the Figure 11. But TPU have better accuracy compared to others. TPU has an accuracy of 92%, refer Figure 11. They have almost the same test accuracy in Figure 12 containing test accuracy.

It took more than 9500 sec for CPU to train it took 127 sec for GPU and 97 sec for TPU. TPU and GPU are very much faster than CPU. GPU and TPU almost have the same training time if we increase the no. of epochs, as represented in Figure 13.

Figure 14 shows the trade off between the time needed for training a batch of samples and the number of epochs. This signifies the time taken for a batch to train rather than the entire

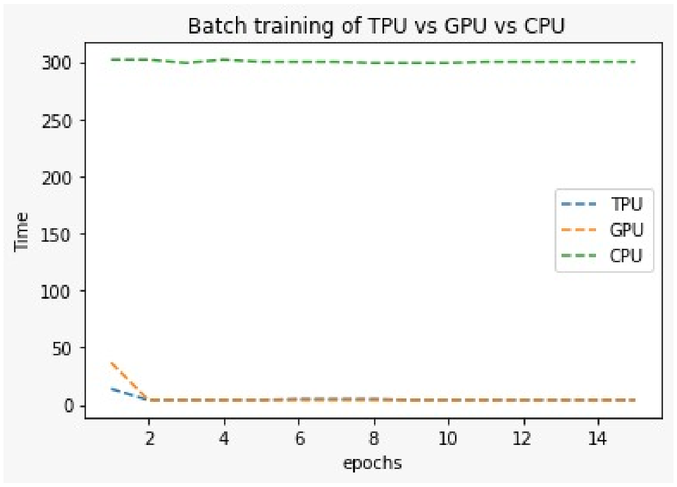


Fig. 14. TPU vs GPU

dataset at once. Figure 15 shows the same tradeoff between GPU and TPU. This graph is important to observe the slight difference between the latencies of GPU and TPU.

We felt that TPU is much better than GPU in all terms compared to GPU. If latency is not the problem, CPUs also have almost similar accuracy of TPU and GPU.

V. Issues encounter

We tried to train our model on our local computers but the major issue was, we don't have the TPU on our local system. So, to compare all the hardwares we need one platform on which we can train our model with various hardwares and compare the performance of them. So we used cloud

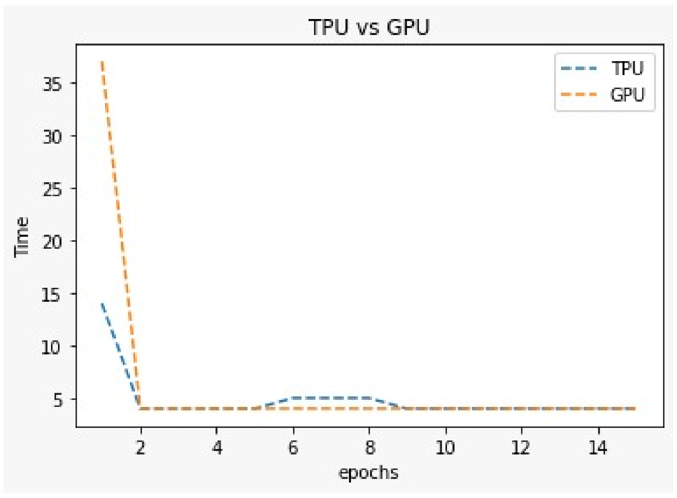


Fig. 15. Batch Trining of TPU, GPU and CPU

infrastructure Google Colab as it provides us CPU, GPU and TPU on the same platform. First we planned to run all model comparisons in one notebook. It's not possible to compare three hardwares in a single runtime. As we change run time the kernel dies and all the varabies will be cleared out. So we trained our model in three different notebooks with different runtime environments. We tried to train our model on a local system for CPU but it was taking too much time for the training. As we can see, it took more than 1hour even on the cloud platform.

VI. Conclusion

This report presented a comparison between the hardwares based on Machine Learning Implementation aspects. The metric for comparison is the time required for training of the model on each hardware. Also the CUDA functionality is used with GPU which enables GPU to have features like dynamic parallelism which in term helps the model train faster.TPU and GPU trains model 7-8 times faster than CPU. in terms of accuracy they don't have much difference. TPU is better in all aspects. If training time doesn't matter CPU also has almost similar accuracy of TPU and better than GPU. GPU and TPU are executing the instruction in parallel and CPU executes instructions serially. Because of the parallelism the latency is less for them compared to CPU. CUDA is a parallel computing platform and programming style that harnesses the power of the graphics processor unit (GPU) to enable remarkable gains in computing speed. So GPU with CUDA gives better performance.

References

[1] Wang, Yu Emma, Gu-Yeon Wei, and David Brooks. "Benchmarking tpu, gpu, and cpu platforms for deep learning." arXiv preprint arXiv:1907.10701 (2019).

[2]

[3] Y. Wang et al., "Benchmarking the Performance and Energy Efficiency of AI Accelerators for AI Training," 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), 2020, pp. 744-751, doi: 10.1109/CCGrid49817.2020.00-15.

[4] Reuther, Albert, et al. "Survey and benchmarking of machine learning accelerators." 2019 IEEE high performance extreme computing conference (HPEC). IEEE, 2019.

[5] M. Kayed, A. Anter and H. Mohamed, "Classification of Garments from Fashion MNIST Dataset Using CNN LeNet-5 Architecture," 2020 International Conference on Innovative Trends in Communication and Computer Engineering (ITCE), 2020, pp. 238-243, doi: 10.1109/ITCE48509.2020.9047776.

[6] S. B., A. Lesle A., B. Diwakar, K. R. and G. M., "Evaluating Performance of Deep Learning Architectures for Image Classification," 2020 5th International Conference on Communication and Electronics Systems (ICCES), 2020, pp. 917-922, doi: 10.1109/ICCES48766.2020.9137884.

[7] Zhang, Yue, "Evaluation of CNN Models with Fashion MNIST Data" (2019). Creative Components. 364.

[8] Kadam, Shivam Adamuthe, Amol Patil, Ashwini. (2020). CNN Model for Image Classification on MNIST and Fashion-MNIST Dataset. Journal of scientific research. 64. 374-384. 10.37398/JSR.2020.640251.

[9] A.S. Henrique, A.M.R. Fernandes, R. Lyra, V.R.Q. Leithardt, S.D. Correia, P. Crocker, R.L.S. Dazzi "Classifying Garments from Fashion-MNIST Dataset Through CNNs", Advances in Science, Technology and Engineering Systems Journal, vol. 6, no. 1, pp. 989-994 (2021).

[10] K. Kothapalli, R. Mukherjee, M. S. Rehman, S. Patidar, P. J. Narayanan and K. Srinathan, "A performance prediction model for the CUDA GPGPU platform," 2009 International Conference on High Performance Computing (HiPC), 2009, pp. 463-472, doi: 10.1109/HIPC.2009.5433179.

[11] M. Fatica, "CUDA toolkit and libraries," 2008 IEEE Hot Chips 20 Symposium (HCS), 2008, pp. 1-22, doi: 10.1109/HOTCHIPS.2008.7476520.

[12] M. Garland, "Parallel computing with CUDA," 2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS), 2010, pp. 1-1, doi: 10.1109/IPDPS.2010.5470378.

[13] R. S. Dehal, C. Munjal, A. A. Ansari and A. S. Kushwaha, "GPU Computing Revolution: CUDA," 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), 2018, pp. 197-201, doi: 10.1109/ICACCCN.2018.8748495.

[14] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," 2009 IEEE International Symposium on Performance Analysis of Systems and Software, 2009, pp. 163-174, doi: 10.1109/ISPASS.2009.4919648.

[15] D. Luebke, "CUDA: Scalable parallel programming for high-performance scientific computing," 2008 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro, 2008, pp. 836-838, doi: 10.1109/ISBI.2008.4541126.

[16] Agarap, Abien Fred. "An architecture combining convolutional neural network (CNN) and support vector machine (SVM) for image classification." arXiv preprint arXiv:1712.03541 (2017).

[17] Davidson, D.. "Introduction to GPU Computing and CUDA Programming: A Case Study on FOlD." (2010).

[18] J. Fang, A. L. Varbanescu and H. Sips, "A Comprehensive Performance Comparison of CUDA and OpenCL," 2011 International Conference on Parallel Processing, 2011, pp. 216-225, doi: 10.1109/ICPP.2011.45.

[19] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," 2009 IEEE International Symposium on Performance Analysis of Systems and Software, 2009, pp. 163-174, doi: 10.1109/ISPASS.2009.4919648.

[20] L. de P. Veronese and R. A. Krohling, "Differential evolution algorithm on the GPU with C-CUDA," IEEE Congress on Evolutionary Computation, 2010, pp. 1-7, doi: 10.1109/CEC.2010.5586219.

[21] X. Sierra-Canto, F. Madera-Ramirez and V. Uc-Cetina, "Parallel Training of a Back-Propagation Neural Network Using CUDA," 2010 Ninth International Conference on Machine Learning and Applications, 2010, pp. 307-312, doi: 10.1109/ICMLA.2010.52.

[22] H. Grahn, N. Lavesson, M. H. Lapajne and D. Slat, "CudaRF: A CUDA-based implementation of Random Forests," 2011 9th IEEE/ACS International Conference on Computer Systems and Applications (AICCSA), 2011, pp. 95-101, doi: 10.1109/AICCSA.2011.6126612.

[23] T. Carneiro, R. V. Medeiros Da NóBrega, T. Nepomuceno, G. Bian, V. H. C. De Albuquerque and P. P. R. Filho, "Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning

Applications," in IEEE Access, vol. 6, pp. 61677-61685, 2018, doi: 10.1109/ACCESS.2018.2874767.

[24] D. Li, X. Chen, M. Becchi and Z. Zong, "Evaluating the Energy Efficiency of Deep Convolutional Neural Networks on CPUs and GPUs," 2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom), 2016, pp. 477-484, doi: 10.1109/BDCloud-SocialCom-SustainCom.2016.76.

[25] E. Nurvitadhi, D. Sheffield, Jaewoong Sim, A. Mishra, G. Venkatesh and D. Marr, "Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC," 2016 International Conference on Field-Programmable Technology (FPT), 2016, pp. 77-84, doi: 10.1109/FPT.2016.7929192.

[26] D. Steinkraus, I. Buck and P. Y. Simard, "Using GPUs for machine learning algorithms," Eighth International Conference on Document Analysis and Recognition (ICDAR'05), 2005, pp. 1115-1120 Vol. 2, doi: 10.1109/ICDAR.2005.251.

[27] Y. J. Mo, J. Kim, J. Kim, A. Mohaisen and W. Lee, "Performance of deep learning computation with TensorFlow software library in GPU-capable multi-core computing platforms," 2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN), 2017, pp. 240-242, doi: 10.1109/ICUFN.2017.7993784.

[28] I. Baldini, S. J. Fink and E. Altman, "Predicting GPU Performance from CPU Runs Using Machine Learning," 2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing, 2014, pp. 254-261, doi: 10.1109/SBAC-PAD.2014.30.

[29] D. Strigl, K. Kofler and S. Podlipnig, "Performance and Scalability of GPU-Based Convolutional Neural Networks," 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, 2010, pp. 317-324, doi: 10.1109/PDP.2010.43.

[30] Meshkini K., Platos J., Ghassemain H. (2020) An Analysis of Convolutional Neural Network for Fashion Images Classification (Fashion-MNIST). In: Kovalev S., Tarassov V., Snasel V., Sukhanov A. (eds) Proceedings of the Fourth International Scientific Conference "Intelligent Information Technologies for Industry" (IITI'19). IITI 2019. Advances in Intelligent Systems and Computing, vol 1156. Springer, Cham.https://doi.org/10.1007/978-3-030-50097-9_10

[31] Programming Guide :: CUDA Toolkit Documentation. (2006). (C) Copyright 2005. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.htmlcuda-general-purpose-parallel-computing-architecture

[32] T. Lu, T. Marin, Y. Zhuo, Y. -F. Chen and C. Ma, "Accelerating MRI Reconstruction on TPUs," 2020 IEEE High Performance Extreme Computing Conference (HPEC), 2020, pp. 1-9, doi: 10.1109/HPEC43674.2020.9286192.

[33] O. Berezsky, O. Pitsun, L. Dubchak, P. Liashchynskyi and P. Liashchynskyi, "GPU-based biomedical image processing," 2018 XIV-th International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH), 2018, pp. 96-99, doi: 10.1109/MEMSTECH.2018.8365710.

[34] E. Cengil, A. Çinar and Z. Güler, "A GPU-based convolutional neural network approach for image classification," 2017 International Artificial Intelligence and Data Processing Symposium (IDAP), 2017, pp. 1-6, doi: 10.1109/IDAP.2017.8090194.

[35] Z. Wu et al., "GPU Parallel Implementation of Spatially Adaptive Hyperspectral Image Classification," in IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 11, no. 4, pp. 1131-1143, April 2018, doi: 10.1109/JSTARS.2017.2755639.

[36] A. Morar, F. Moldoveanu, A. Moldoveanu, O. Balan and V. Asavei, "GPU accelerated 2D and 3D image processing," 2017 Federated Conference on Computer Science and Information Systems (FedCSIS), 2017, pp. 653-656, doi: 10.15439/2017F265.

[37] A. Samat, E. Li, P. Du, S. Liu and J. Xia, "GPU-Accelerated CatBoost-Forest for Hyperspectral Image Classification Via Parallelized mRMR Ensemble Subspace Feature Selection," in IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 14, pp. 3200-3214, 2021, doi: 10.1109/JSTARS.2021.3063507.

[38] P. Valero-Lara, "MRF Satellite Image Classification on GPU," 2012 41st International Conference on Parallel Processing Workshops, 2012, pp. 149-156, doi: 10.1109/ICPPW.2012.24.

[39] M. Vajgl and P. Hurtik, "A pipeline for detecting and classifying objects in images," 2020 IEEE Third International Conference on Data Stream Mining Processing (DSMP), 2020, pp. 163-168, doi: 10.1109/DSMP47368.2020.9204121.