Real-Time Server Monitoring and CNN Inference on FPGA

August 2019

AUTHOR:

Debdeep Paul

Experimental Physics-Neutrino Group (EP-UNU).

SUPERVISOR:

Paola Sala

Experimental Physics-Neutrino Group (EP-UNU).

PROJECT SPECIFICATION

The Deep Underground Neutrino Experiment (DUNE) is a leading-edge, international experiment for neutrino science and proton decay studies built in the United States, hosted by Fermi National Accelerator Laboratory (Fermilab). The proto-DUNE team in CERN has build a prototype of these chambers to test and validate the technologies and design that will be applied to the construction of the DUNE Far Detector at the Sanford Underground Research Facility.

Recent years have seen a lot of popularity of Deep Learning among the High Energy Physics community. Presently, the EP-UNU group is investigating various new algorithms for data acquisition and event selection for the DUNE neutrino experiment for understanding and analyzing the different neutrino interactions. Since FPGA provides the best trade-off in terms of power and hardware parallelism, our team has been trying to use FPGA based solutions for CNN inference.

The major goal of this project is outlined as follows:

- Develop a lightweight web-based monitoring dashboard for the server hosting the FPGA for running the CNN Inference.
- Compare the performance of the CPU and FPGA for CNN inference in terms of accuracy, power and latency.
- Look into the problem of numerical overflow created as a result of using fixed point airthmetic during CNN Inference on the FPGA.

ACKNOWLEDGEMENT

I am extremely thankful and highly obliged to all the people who have guided and assisted me in completion of this work and I consider myself very fortunate to have got them all along.

Firstly, I am grateful to CERN and OpenLab for giving me this opportunity to be a part of this summer programme! It has certainly exceeded all my expectations and truly been an amazing experience.

I would take this opportunity to express my deepest gratitude to my supervisor, **Paola Sala** for her kind co-operation, encouragement and support. I also heartily thank **Manuel Jesus Rodriguez Alonso** for his valuable guidance and the constructive feedback. They were always there whenever I wanted any suggestions. Their ideas were always insightful and practical.

I also thank all my friends and colleagues of the CERN Openlab Programme as well as the Summer Students programme for making this summer memorable for me!

Last but not the least, I would like to thank my family for supporting me and providing constant encouragement and motivation.

Thank you all...

ABSTRACT

Neutrinos are subatomic particles, very similar to an electron, but without any electrical charge and a very negligible rest mass. They are the most abundant and perhaps the most mysterious matter particles in the universe!

Detecting and tracking neutrinos poses a different challenge both in terms of hardware and algorithms. Since they have very little interaction with matter, they are incredibly difficult to detect. So, every bit of information produced by the neutrino interactions is required to be stored and analyzed at an incredible fast rate.

Present day State-of-the-Art Convolutional Neural Networks (CNNs) have shown an un-precedented accuracy on many modern AI applications. They have become the de-facto standard for a wide range of tasks ranging from computer vision to machine translation due to their high accuracy and robustness.

Research has proved that CNNs also hold the potential to excel in High Energy Physics (HEP) applications as compared to traditional methods in identifying particle interactions in sampling calorimeters or Time Projection Chambers. However, these algorithms are both computationally and memory extensive which limits them from running on normal CPU for real-time and power constrained applications. Thus, customized hardware implementation of machine learning algorithms can be a promising solution for higher performance and improved throughput.

Another important aspect is to constantly monitor the server which hosts the FPGAs for the CNN inference and sent alert messages in crucial situations.

This project involved the development of a lightweight, easy to use solution for the server monitoring so that all the important metrices can be visualized at a glance in real-time. Secondly, to compare the CNN inference between the server and the FPGA in terms of accuracy, power consumption and latency. The project also focuses on dealing with the numerical overflow problem due to fixed point computations in FPGA during CNN inference.

Keywords: Proto-DUNE, Real-Time Server Monitoring, Convolutional Neural Network, FPGA, Fixed Point Computation.

TABLE OF CONTENTS

TABLE OF CONTENTS

PART B: CNN Inference on FPGA

INTRODUCTION

Real-Time monitoring of servers running crucial applications are an important part of any major data-center or research organization. This makes a simple, lightweight and easy-to -use web-based monitoring tool in huge demand. It ensures monitoring all the system resources associated with the server to understand their resource usage patterns and optimize them accordingly to provide a better end-user experience.

Recent advancement in the fields of Deep Learning has proven that Deep Convolutional Neural Networks (DCNN) can achieve human level accuracy, sometimes even exceeding in classification tasks. Research has shown that they have the potential to outperform traditional statistical based methods in High Energy Physics as well.

This project is divided into two parts: At first, we develop a solution for real-time server monitoring, and in the second phase, we have investigated the use of CNNs for understanding the Neutrino interactions in the Liquid Argon Time Projection Chamber of the proto-DUNE Experiment.

PART A: REAL-TIME SERVER MONITORING

The goal of this first part of the project was to develop a web interface dashboard to seamlessly monitor the performance of the server (np04-srv-102) hosting our two FPGAs, at a glance, and to report any abnormalities immediately for any further action. The dashboard will include an overview of all the important metrices like CPU, Memory, Disk, Network and also our custom parameters like CNN Inference time, data load and processing time, power consumption and detailed CPU core temperatures.

1. TOOLS USED

1.1 PROMETHEUS

Prometheus is a standalone open source monitoring and alerting toolkit. It is highly customizable and designed to deliver rich metrics without creating a drag on system performance. Though originally built at SoundCloud, since its inception in 2012, many companies and organizations have adopted Prometheus, and the project has a very active developer and user community.

Prometheus works well for recording any purely numeric time series. Most Prometheus components are written in Go while some are also written in Java, Python, and Ruby. It pushes metrics over http rather than pulling. Prometheus also facilitates sending alerts using Prometheus Alert-Manager.

Prometheus has a main central component called Prometheus Server which is responsible for scraping the metrics (for e.g. current CPU status, memory usage, etc) from the targets (which can be an entire Linux server, a stand-alone Apache server, a single process or a database service) at a

user-defined interval. These can then be stored in a time-series database locally or remotely and displayed back in the Prometheus server.

The Prometheus Node Exporter exposes a wide variety of hardware and kernel-related metrics. It facilitates easy means to measure various machine resources such as memory, disk and CPU utilization.

Figure 1 illustrates the architecture of Prometheus and some of its ecosystem components:

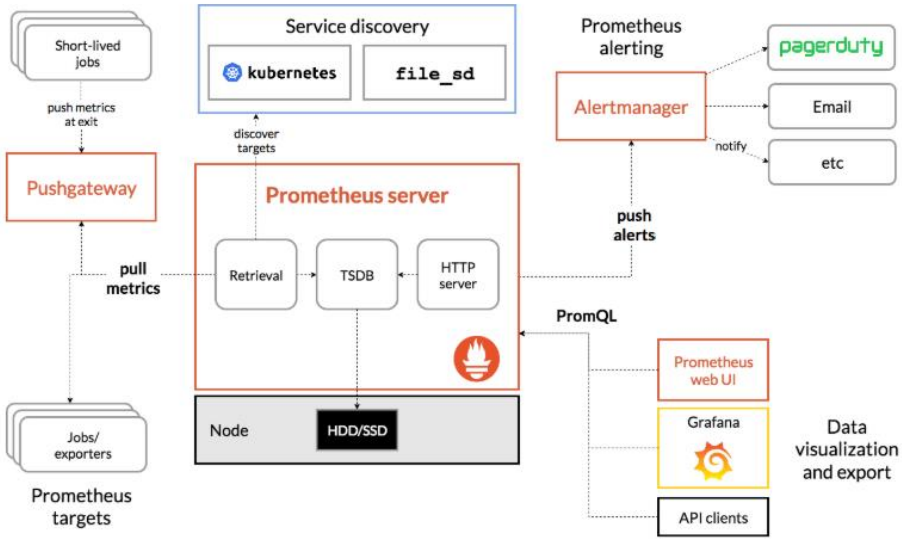

Figure 1. Prometheus Architecture.

1.2 INFLUXDB

InfluxDB is an open-source time series database developed by InfluxData. It is written in Go and optimized for fast, high-availability storage and retrieval of time series data. This includes APIs for storing and querying data, processing it in the background for monitoring and alerting purposes, user dashboards, visualizing and exploring the data and more. It is widely used in fields such as operations monitoring, application metrics, Internet of Things sensor data, and real-time analytics.

1.3 CHRONOGRAF

Chronograf is InfluxData's open source web application. Chronograf facilitates quick and easy visualization of the data that is stored in InfluxDB to build robust queries and alerts.

1.4 GRAFANA

Grafana is one of the most widely used visualization tools when it comes to time-series data analytics. Grafana allows to query, visualize and understand the metrics from any time-series database by numerous dynamic & reusable dashboards containing a wide range of graphs, charts and other plugins.

It also allows to seamlessly define alert rules and thresholds for the most important metrics. Grafana will continuously evaluate and send notifications to systems like Slack, email, Telegram, etc.

2. THE MONITORING FRAMEWORK

The server monitoring framework is depicted in Figure 2 .

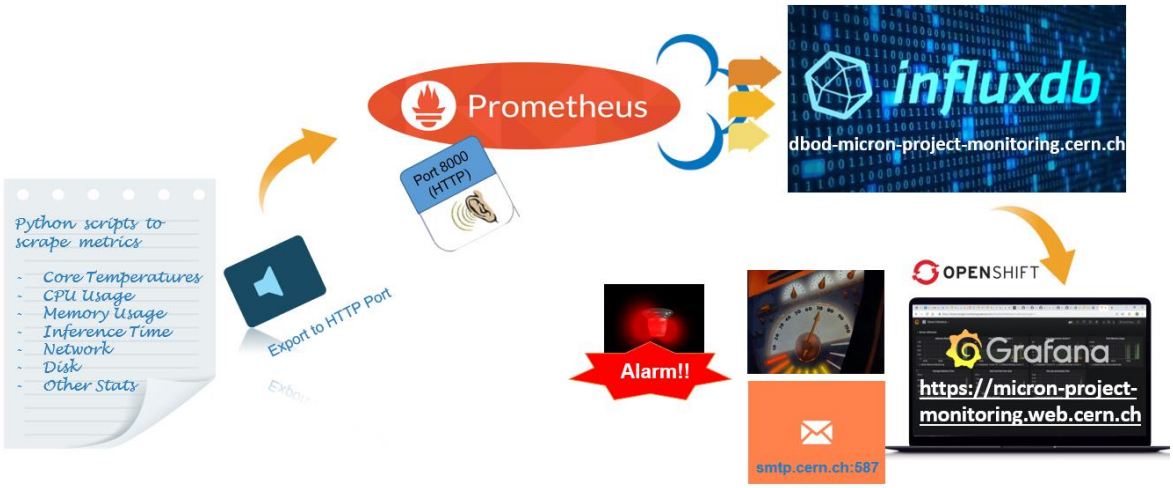

Figure 2. Server Monitoring Framework using Prometheus, Influxdb and Grafana.

First, we write python scripts to scrape the metrices which we want to monitor. The basic hardware and kernel-related metrics can be scraped easily through the Prometheus node-exporter client. We only write scripts to scrape the custom parameters which we want to monitor.

The metrices are exported through a dedicated http port. The Prometheus server collects metrics from targets (over HTTP) and stores them locally in the server itself. These metrices can be displayed in the local browser through the port (by default Prometheus server uses port 9090: https://localhost:9090).

We also export all the metrices to a remote database on CERN DBoD (Database on Demand) server. We have chosen InfluxDB as it provides one of the best standards and is most widely used for time-series databases.

The next-step was to create a web-page so that we can visualize our data through it. We host Grafana dashboards on the web-page and visualize the metrices by creating different graphs and charts.

The different graphs and dashboards on our website are explained in the later sections.

3. THE MONITORING DATABASE

We created our remote InfluDB database using CERN Database on Demand (DBoD) service. CERN DB On Demand offers a seamless way to create and manage databases. It also allows users to

perform tasks that are traditionally carried out by database administrators like operating system and database engine updates, access to backup and recovery services and some support for service continuity in case of hardware failure. We got a database of size 300GB in CERN server.

4. THE MONITORING WEB-SITE

We have created and hosted our website (**https://micron-project-monitoring.web.cern.ch**) on the CERN central web servers using CERN web-services. We created a PaaS (Platform-as-a-Service) web-application which enables deployment of small web applications as Docker containers in an Openshift cluster.

We run our own dedicated Grafana instance in central web servers through which we visualize the different metrics by quering the InfluxDB database.

5. GRAFANA DASHBOARDS

5.1 THE HOME DASHBOARD

The Grafana home dashboard contain all the basic parameter of CPU, Memory, Disk and Network usage. A screenshot of the Grafana home dashboard is shown in Figure 3.

For each of CPU, Memory, File System, Network and I/O we have separate panels in our main dashboard which shows the details of each of them. All these metrics are being displayed by querying the InfluxDB database through Grafana.

We also have our custom metrics for the CNN inference and detailed CPU core temperatures and power in separate dashboards which is explained in the next section.

Figure 3. Grafana Home Dashboard for Server Monitoring.

5.2 CPU CORE TEMPERATURE DASHBOARD

One of the main concerns about the server was the temperature of the CPU cores when training or inference was running for days. So we created a separated dashboard which had a detailed view of temperatures of all the 24 real cores of the server and also the average socket temperatures at a glance. Figure 4 shows a view the dashboard.

Figure 4. CPU Core Temperature Monitoring Dashboard.

5.3 MONITORING THE CNN INFERENCE PROCESS

The main objective behind the server monitoring was to study the important server metrics like temperature, memory, disk, etc. while the inference was running. We created a separate dashboard in Grafana to highlight these parameters (Figure 5).

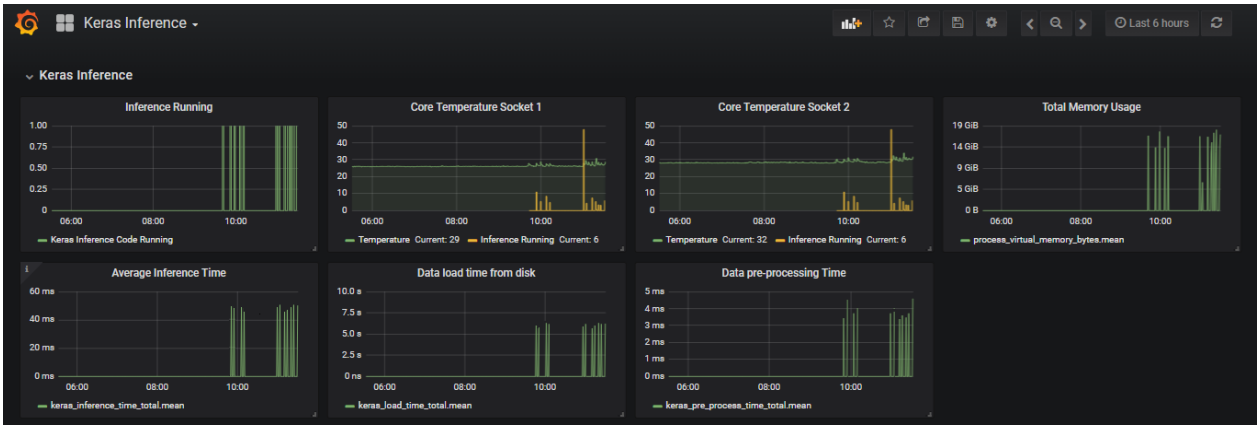

Figure 5. CNN inference Monitoring Dashboard (using Keras).

The spikes in the first graph highlights the time instances in which the inference code was called from the command line. The next two graphs highlight the overall temperatures of the 2 sockets in

the CPU, each containing 12 cores. The green continuous curve represents the temperature and the yellow spikes represents the instances at which inference was running. The rise is core temperatures during the inference can be observed from the graphs. The fourth graph in the figure shows the memory usage of the inference process.

We also monitor the exact time required for inference, data loading and data-processing for comparing the CPU performance with GPU and the FPGA. These are represented in the last three graphs respectively.

The average time required for the various processes in the CPU are shown in Table 1 in Section 11.

6. ALERTING

An important aspect of monitoring is sending alerts to the concerned persons when any of the parameters exceeds its threshold.

We have configured Grafana to send alerts via email. This is done by creating an alert channel in Grafana and then configuring the smtp settings in the Grafana custom.ini file.

```
#################################### SMTP / Emailing ##########################
[smtp]
enabled = true
host = cernmx.cern.ch:25
user = dpaul
# If the password contains # or ; you have to wrap it with triple quotes. Ex """#password;"""
password = *****
cert_file =
key_file =
skip_verify = true
from_address = debdeep.paul@cern.ch
from_name = Debdeep Paul

ehlo_identity =

#################################################################################
```

The next step is to define the alerting rules in Grafana. This is done by going back to the dashboard and setting up the thresholds and the custom messages from each of the graphs where an alert is required.

We can also send SMS notification to the concerned user by the CERN sms via email service. This service allows to send free of charge short messages to CERN GSMs by sending email to the following address: +4175411xxxx@mail2sms.cern.ch (xxxx is last 4 digits of the GSM).

PART B: CNN Inference on FPGA

This part of the project involved running the CNN inference on the CPU as well as the FPGA for comparison of their performance. Also, it focuses on investigating into the numerical overflow problem resulting due to fixed-point computations during the CNN inference on the FPGA.

We first give a brief description of CNN and the FPGA inference framework and then focus on the problems and the solutions which we dealt with.

7. CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNN) are a class of deep, feed-forward artificial neural networks (ANN), which are most popularly used in image classification, object recognition and similar tasks. They are particularly best suited for operation on 2D input data such as images.

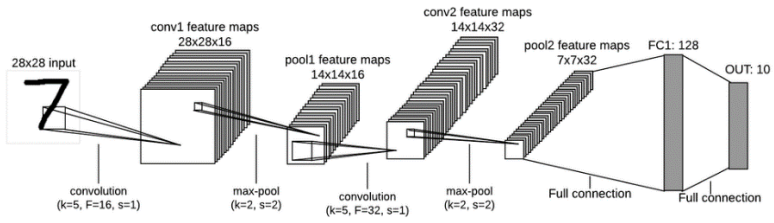

Figure 6. A typical CNN Architecture.

Research has shown that CNNs are well suited to a broad class of detectors used in High Energy Physics (HEP). In our project, we have also been trying to implement CNN to study and analyse different particle interactions in our Proto-DUNE detector. Each of these detectors (Single Phase and Dual Phase) is a 10x10x10-metre Liquid Argon Time Projection Chamber, containing about 800 tonnes of liquid argon.

We record the electrical signal released by ionising particles crossing the detector by a 3-dimensional electrode structure. When these signals are combined, they result in what is essentially an image of the physics interaction that takes place within the chamber. These captured signals are fed into neural networks for studying different interactions.

8. FPGA

Field Programmable Gate Arrays or FPGAs are programmable logic devices which are used for rapid prototyping of complex digital systems. FPGAs are pre-fabricated silicon chips containing a matrix of Configurable Logic Blocks (CLBs comprising of flip-flops, LUTs, multiplexers etc.) and I/Os connected via programmable routing. Most modern FPGAs also contain a heterogeneous mixture of different blocks like dedicated memory blocks and DSP Blocks.

Figure 7. Field Programmable Gate Array.

FPGAs can be configured by a designer after manufacturing; hence the term "field programmable". The FPGA configuration is generally specified using a hardware description language.
Once the HDL model is verified, the description is synthesized and mapped into the FPGA by special tools and softwares provided by the manufacturer.

8.1 CNN INFERENCE ON FPGA

FPGAs provides the perfect trade-off between the flexibility and programmability of software running on a general-purpose CPU and the speed and power efficiency of a custom designed Application Specific Integrated Circuit (ASIC).

CNN algorithms are inherently highly parallelizable in nature and FPGAs are designed to handle irregular parallelism and fine-grained computations which makes them the preferred solution for CNN deployment.

9. HIGH LEVEL SYNTHESIS (HLS)

High-level synthesis (HLS) is an automated design process that interprets an algorithmic description of a desired behavior and creates digital hardware that implements that behavior.

HLS starts with an algorithmic description in a high-level language such as SystemC and ANSI C/C++. The high-level synthesis tools handle the micro-architecture and transforms the code into RTL implementations. The RTL implementations are then used directly in a conventional logic synthesis flow to create a gate-level implementation. HLS greatly reduces the intellectual property (IP) development cycle from months to weeks.

10. CNN INFERENCE WORKFLOW

A typical Neural Network deployment consists of two phases: Training and Inference. Training involves both forward and backward passes to learn the weights and biases and naturally is more compute extensive than inference. Our Neural Network was trained using a GPU using our proto-DUNE dataset comprising of 13 Million images. The model architecture and the weights were saved in h5 format.

Our server hosts two FPGAs which are accessible through the PCI slots :

- **Micron SB-852**

SB-852 is a full-height, GPU-length, PCIe x16 Gen3 board designed to deliver unprecedented levels of high-bandwidth and low-latency performance. It is equipped with a Xilinx® Virtex Ultrascale+ FPGA, a 2GB Hybrid Memory Cube (HMC), up to 512GB of high-performance memory and two QSFP28 connectors.

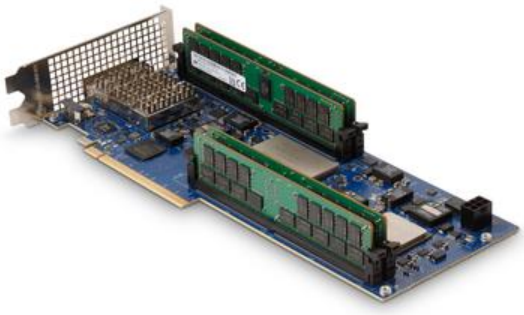

Figure 8. Micron SB-852.

- **Micron AC-510**

The Micron AC-510 features a Xilinx Kintex® UltraScale 060 FPGA with Micron high-bandwidth Hybrid Memory Cube (HMC). It features a bandwidth of up to 60 GB/s and also supports OpenCL™ framework along with Micron Pico Framework for efficient acceleration of demanding applications.

Figure 9. Micron AC-510.

To run the CNN inference on the FPGAs it is required to export the bit-file to the FPGA for mapping the hardware architecture onto the FPGA. This bit-file is generated by the Micron ForwardNext library. This library takes an ONNX model as input and generates the bit-stream. ONNX or One Neural Network Exchange format is an open format to represent deep learning models aimed to allow framework inter-portability between different machine learning libraries. The CPU can communicate to the FPGA via the PCI slots of the motherboard. The entire flow-chart is represented in Figure 10.

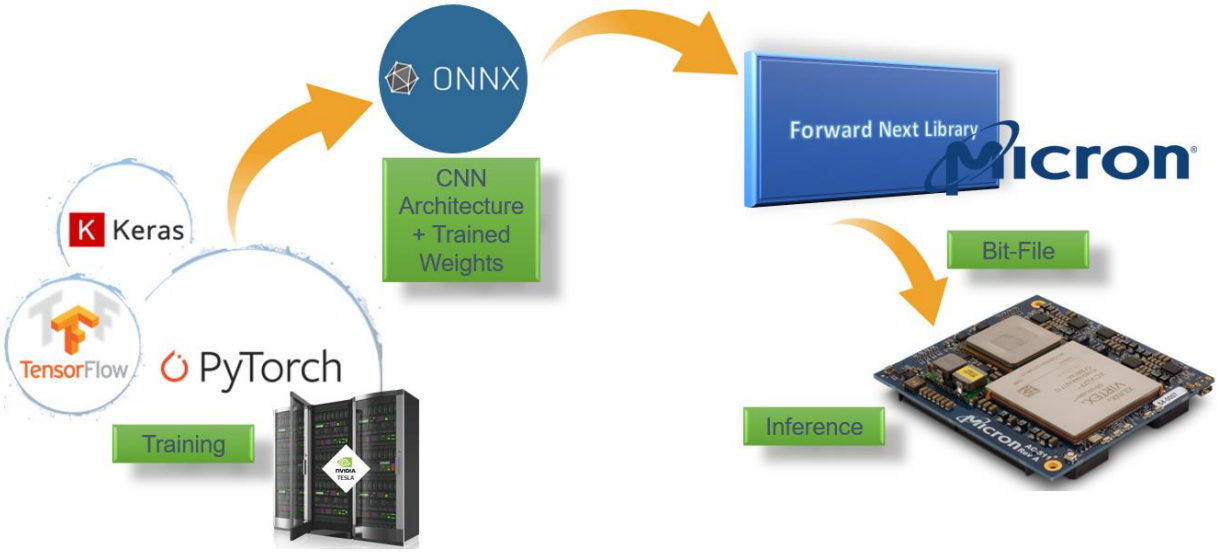

Figure 10. CNN Inference Workflow.

11. FPGA VS CPU FOR CNN INFERENCE

The Neural Network was trained on a subset of our DUNE dataset. We used a slightly modified version of the ResNet-18 Network for our classification with one softmax output over 13 values corresponding to the 13 types of neutrino interaction considered and saved the weights. We tested the inference both on the CPU and the FPGA. The server (CPU) contains a Intel Xeon Silver 4116, 2.1 GHz Processor with 195237MB DDR4 RAM.

Table 1 summarizes the inference time in the CPU and Table 2 on the Micron FPGA 510.

Process	Average Time
Average inference time for a single image	49.52706972757975 ms
Time taken to load model from disk	2.501638174057007 s
Time taken for image pre-processing	3.3588409423828125 ms

Table 1. Average Inference Time in CPU.

Process	Average Time
Average inference time for a single image	190.1567 ms
Time taken to load input to main memory	0.4680 ms
Time taken for converting float to int	12.1521 ms

Table 2. Average Inference Time in FPGA.

In both the cases, no other processes were running during inference. It can be observed that the CPU takes much shorter time for the actual inference to take place. This can be concluded due to the fact that in the CPU we were concatenating 150 images in a batch and then performing the inference and most deep learning libraries perform great with batch processing. The time that is mentioned here is the calculated time; dividing by 150. On the other hand, in the FPGA, we perform the inference of a single image at a time due to which the average time increases significantly. On changing the batch size to 1 in the CPU, we got similar timings, but still the CPU was faster.

As per as accuracy is concerned, both the CPU and FPGA gave almost similar probability results; with some minor differences due to the using fixed point in FPGA and float32 in CPU.

12. FIXED POINT COMPUTATIONS

In a fixed-point representation, there are a fixed number of digits before and after the decimal point. Designs implemented in fixed point are always more efficient than their equivalent in floating point because fixed-point implementations consume fewer resources. Also, fixed-point arithmetic is much more hardware-friendly in terms of complications and resource utilization. Reduction in FPGA resource usage inherently leads to lower power consumption.

The Micron library converts all the weights and biases to Q8.8 fixed point numbers, that is, there are 8 integer bits and 8 fractional bits in a number.

12.1 THE NUMERICAL OVERFLOW PROBLEM

We have used Q8.8 fixed point arithmetic all over the network. Since in fixed point the range and resolution become constrained, there is a possibility that the activations will overflow out of the range (-127, +127) as they are accumulated over the iterations.

From Figure 11 we can see that the overflow starts from node 18, which is a Convolution Layer and from then onwards, it keeps on accumulating. The maximum overflow (74.77 %) occurs at node 25 which is again a Convolution Layer. Figure 12 shows the type of layer each node is represented by. As the activations become out of the range, they are clipped off; which changes the value of the activations and ultimately leads to poor accuracy.

```
WARNING: overflows in node 18: 4.18 % (1340/32042)
WARNING: overflows in node 19: 0.09 % (49/57330)
WARNING: overflows in node 21: 1.98 % (503/25354)
WARNING: overflows in node 23: 14.40 % (3479/24160)
WARNING: overflows in node 24: 14.78 % (4477/30287)
WARNING: overflows in node 25: 74.77 % (15950/21332)
WARNING: overflows in node 26: 17.28 % (5843/33806)
WARNING: overflows in node 27: 13.11 % (5290/40341)
WARNING: overflows in node 28: 57.73 % (11732/20321)
WARNING: overflows in node 31: 33.33 % (5/15)
```

Figure 11. Percentage of Numerical Overflow in different nodes of the CNN.

```
type conv id=18 in=(H32,W32,P256) out=(H32,W32,P256) k=1x1 stride=1x1 dilation=1x1 pad=(0,0,0,0) 18->19
type conv id=19 in=(H32,W32,P256) out=(H32,W32,P256) k=3x3 stride=1x1 dilation=1x1 pad=(1,1,1,1) 19->20
type resconv id=20 in=(H32,W32,P256) out=(H32,W32,P256) k=3x3 stride=1x1 dilation=1x1 pad=(1,1,1,1) 20,18->21
type conv id=21 in=(H32,W32,P256) out=(H32,W32,P256) k=1x1 stride=1x1 dilation=1x1 pad=(0,0,0,0) 21->22
type conv id=22 in=(H32,W32,P256) out=(H16,W16,P512) k=1x1 stride=2x2 dilation=1x1 pad=(0,0,0,0) 21->23
type conv id=23 in=(H32,W32,P256) out=(H16,W16,P512) k=3x3 stride=2x2 dilation=1x1 pad=(0,1,0,1) 22->24
type resconv id=24 in=(H16,W16,P512) out=(H16,W16,P512) k=3x3 stride=1x1 dilation=1x1 pad=(1,1,1,1) 24,23->25
type conv id=25 in=(H16,W16,P512) out=(H16,W16,P512) k=1x1 stride=1x1 dilation=1x1 pad=(0,0,0,0) 25->26
type conv id=26 in=(H16,W16,P512) out=(H16,W16,P512) k=3x3 stride=1x1 dilation=1x1 pad=(1,1,1,1) 26->27
type resconv id=27 in=(H16,W16,P512) out=(H16,W16,P512) k=3x3 stride=1x1 dilation=1x1 pad=(1,1,1,1) 27,25->28
type conv id=28 in=(H16,W16,P512) out=(H16,W16,P512) k=1x1 stride=1x1 dilation=1x1 pad=(0,0,0,0) 28->29
type intlv id=29 in=(H16,W16,P512) out=(H16,W16,P512) 29->30
type avgpool id=30 in=(H16,W16,P512) out=(H1,W1,P512) k=16x16 stride=1x1 pad=(0,0,0,0) ceil=0 30->31
type conv id=31 in=(H4,W1,P128) out=(H1,W1,P13) k=4x1 stride=1x1 dilation=1x1 pad=(0,0,0,0) 31->32
type softmax id=32 in=(H1,W1,P13) out=(H1,W1,P13) 32->33
```

Figure 12. Node ID and type of layer of our CNN architecture.

12.2 POSSIBLE SOLUTIONS

Several different solutions came up in our team meetings to deal with the numerical overflow problem. They are summarized as follows:

- **Normalization of the Images** – In machine learning, normalization is usually used in data pre-processing to rescale all the features in a dataset to the same boundaries (0,1) without changing its actual behavior or nature.

Since all the initial values are small, there is a lesser probability of numerical overflow as compared to the original data-values. But normalization was already being applied during inference and it did not solve the overflow problem.

- **Use of Dynamic Quantization** – Dynamic Quantization can help reduce the overflow problem. The weights in a neural network are usually in the range of (-1,+1) ; while the output activations are much larger numerically. Thus, using a different fixed-point format for the weights and the activations can help to reduce the overflow.

Also the activations becomes larger as we go to the deeper layers while the initial activations are small. In this case, a dynamic fixed-point representation across the different layers of the CNN can be of help instead of using a static Q8.8 representation across the entire network.

But the Micron library which converts the ONNX model to the FPGA bit-stream does not currently support dynamic fixed point quantization and so this idea could not be implemented in hardware.

- **Change of Activation Function –** Use of tanh or sigmoid activation function instead of ReLU after the layers that have values overflowing may help to reduce the overflow. This comes from the idea that tanh and sigmoid activations restrict the final values of the activations within (-1,1) and (0, 1) respectively; whereas the ReLU function simply crops the negative activations to 0, leaving the positive activations as they are.

- **Training in Fixed Points –** Training the Network in fixed point itself may help to reduce the overflow as now both the weights as well as the errors gets adjusted over the iterations as they are quantized in each step. We tried to implement a couple of techniques for low-precision training which are described in Sections 13 and 14.

13. RISTRETTO

Ristretto is an automated CNN-approximation tool which condenses 32-bit floating point networks. However, it is based on Caffe framework. Currently, Ristretto supports Dynamic Fixed-Point representations, Minifloat (Bit-width reduced floating point numbers) as well as Power-of-two parameters (multiplier-less in hardware) quantization schemes. It supports test, train and fine-tuning CNNs with limited precision.

We tested this framework using basic LeNet architecture (comprising of 2 convolutional layers) as well as Cifar-10 full architecture (comprising of 3 convolutional layers) with MNIST and Cifar-10 dataset respectively. The accuracies are reported in Table 3.

CNN Architectures	Base Line Accuracy	Fixed Point Accuracy (Training)	Fixed Point Accuracy (Inference)
LeNet (MNIST)	98.76 %	98.76 %	97.61 %
Cifar-10 Full (Cifar-10)	81.62 %	80.84 %	79.26 %

Table 3. CNN accuracy using Ristretto.

However, we used batch normalization layers after each layer in our original network to bring the activation values within -1<0<1. But, currently, Ristretto framework doesn't support batch normalization. We tried removing the normalization layers; but changing the network degraded accuracy steeply; and also since Caffe was not widely used in CERN, we shifted to PyTorch.

14. QPYTORCH

QPyTorch is a low-precision arithmetic simulation package in PyTorch, designed to support low-precision machine learning, especially for researches in low-precision training. QPyTorch supports quantizing different numbers in the training process with customized low-precision format.

We used a basic VGG-16 Network with 10 output classes and trained on Cifar-10 dataset just to get familiar with the workflow. The training was run for 20 epochs. Both the weights and the backpropagated errors were quantized to Q8.8 format. There was a slight decrease in accuracy from the baseline Network trained in float32, by around 0.15%.

CONCLUSION AND FUTURE SCOPE

In this work, we have successfully implemented the server monitoring framework using Prometheus and InfluxDB database and displayed it in a seamless and organized way through various dashboards using Grafana in our project web-site. All the metrics can now be visualized at a glance and any abnormalities can be alerted by email as well as sms to the concerned person.

We also looked into the possibility of monitoring the FPGA as well which is running the inference. We could only record metrices such as inference time, data load time and processing time but could not monitor temperature nor the power consumption. This is due to the reason that the Micron FPGAs which we have, runs a proprietary software from Micron and the FPGA itself is not directly accessible to us. So, we cannot estimate power consumption by tools like Xilinx Power Estimator and scrape those metrices. All the inference process is taken care of by this software. And also, there was no direct support from Micron for monitoring power consumption of the FPGA.

We also investigated into the numerical overflow problem caused due to using fixed-point arithmetic during inference in the FPGA. We looked into training in fixed point and it gave good initial results. Now we have to test it using our DUNE dataset.

Future work in this field includes figuring out a way in which we can monitor the power consumption of the FPGA used for inference. Also, we have to investigate about the best possible way to deal with the numerical overflow problem during CNN inference due to fixed point computations in the FPGA.

REFERENCES

[1] Gupta S, Agrawal A, Gopalakrishnan K, Narayanan P. Deep learning with limited numerical precision. In International Conference on Machine Learning 2015 Jun 1 (pp. 1737-1746).

[2] Gysel P. Ristretto: Hardware-oriented approximation of convolutional neural networks. arXiv preprint arXiv:1605.06402. 2016 May 20.

[3] Courbariaux M, Bengio Y, David JP. Training deep neural networks with low precision multiplications. arXiv preprint arXiv:1412.7024. 2014 Dec 22.

[4] Wu S, Li G, Chen F, Shi L. Training and inference with integers in deep neural networks. arXiv preprint arXiv:1802.04680. 2018 Feb 13.

[5] Prometheus : https://prometheus.io/docs/introduction/overview/ (Figure 1).

[6] Grafana support for Prometheus : https://prometheus.io/docs/visualization/grafana/

[7] Qpytorch : https://github.com/Tiiiger/QPyTorch

[8] CERN PaaS web Application : https://information-technology.web.cern.ch/services/PaaS-Web-App

[9] Prometheus Node Exporter : https://github.com/prometheus/node_exporter

[10] DUNE : https://www.dunescience.org