HARDWARE ACCELERATION OF MACHINE LEARNING USING FPGA

Project report submitted in partial fulfillment of the requirement for the degree of

BACHELOR OF TECHNOLOGY

IN

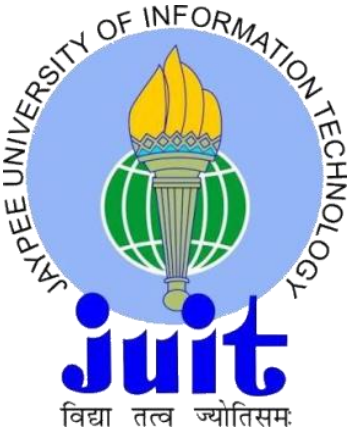
ELECTRONICS AND COMMUNICATION ENGINEERING

By

Rakshit Sharma (181015)
Raghav Sharma (181028)

UNDER THE GUIDANCE OF

Mr. Anuj Kumar Maurya



JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

May 2022

TABLE OF CONTENTS

DECLERATION

We hereby declare that the work reported in the B.Tech Project Report entitled **"Hardware Acceleration of Machine Learning Using FPGA"** submitted at **Jaypee University of Information Technology, Waknaghat, India** is an authentic record of our work carried out under the supervision of **Mr. Anuj Kumar Maurya.** We have not submitted this work elsewhere for any other degree or diploma.

Rakshit Sharma	Raghav Sharma
181015	181028

This is to certify that the above statement made by the candidates is correct to the best of my knowledge .

Mr. Anuj Kumar Maurya

Date:

Head of the Department/Project Coordinator

ACKNOWLEDGEMENT

We would like to express our deepest appreciation to Mr. Anuj Kumar Maurya for helping us throughout the project and without whom this project would have been a very difficult task. We are highly indebted to him for his guidance and constant supervision as well as for providing motivation & his support for this project. He consistently guided us towards the completion of this project. I would also like to express my gratitude towards my parents, members of JUIT and respected HOD sir Dr. Rajiv Kumar for their kind co-operation and encouragement which helped me us completing this project. My thanks and appreciations also go to our colleagues who have helped me out with their abilities and academic support in developing the project. We would like to thank Xilinx for providing us with the Kria KV 260 FPGA board which has been the main testing equipment in this project under Xilinx Adaptive Computing Challenge 2021 as well as Xilinx Engineering Team which helped us better understand the capabilities of the FPGA hardware as well as implement our algorithms in the Kria KV 260 board for this Project. Without such help this project would not have been possible and we are truly grateful for Xilinx for providing us with this opportunity.

LIST OF ACRONYMS AND ABBREVATIONS

CCTV: Closed Circuit Television

YOLO: You Only Look Once

RGB: Red Green Blue

CNN: Convolutional neural network

FC: Fully Connected

GAN: generative adversarial network

PPM: Portable Pixel Map

JPEG: Joint Photographic Expert Group

GIF: Graphics Interchange Format

TIFF: Tag Image File Format

PNG: Portable Network Graphics

FPGA: Field-programmable gate array

SSD: Single Shot Detector

CPU: Central Processing Unit

GPU: Graphics Processing Unit

SOM: System on Module

SOC: System on Chip

CLB: Configurable Logic Block

PLD: Programmable Logic Devices

MUX: Multiplexer

IAS: Imager Access System

USB: Universal Serial Bus

I/O: Input Output

DPU: Deep Learning Processing Unit

BGR: Blue Green Red

HSV: Hue Saturation Value

SLI: Scalable Link Interface

TDP: Thermal Design Power

FPS: Framerates per Second

CUDA: Compute Unified Device Architecture

HW: Hardware

JTAG: Joint Tag Action Group

EDA: Electronic Design Automation

IP Core: Intellectual Property Core

BSP: Board Support Package

COM: Communication

AVC: Advanced Video Coding

API: Application Programming Interface

IP: Internet Protocol

VCC: Common Collector Voltage

ADAS: Advance Driver Assistance System

FSD: Full Self Driving

ML: Machine Learning

EDP: Energy Delay Product

LIST OF SYMBOLS

Gx : Kernal

Gy : Kernal

$\sum$: summation

* : Multiplication

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This project deals with the comparison of different hardware inference method for machine learning including CPU, GPU as well as FPGA implementation. We are particularly interested in performance difference between GPU and FPGA as both are expected to perform better in parallelized tasks.

For testing different hardware acceleration methods we used 12 samples which are categorised into 3 different categories depending upon the complexity of the real world scenario. The Machine Learning algorithm used for the benchmarking of different hardware is SSD or Single Shot Detector. It was observed in this project that as the complexity of the task increases this report concludes that the performance difference between FPGA accelerated hardware inference and GPU accelerated hardware inference increases with FPGA leading the GPU and CPU inference. This suggests that for more complex vision Kernels FPGA implementation outperforms GPU inference performance. If we look at the efficiency of the system lowest power consumption was observed in Kria KV260 FPGA compared to the GPU tested in this project report. Suggesting that for embedded applications FPGA accelerated inference is much more practical compared to GPU accelerated inference. This Project report also looks at the Energy Delay product (EDP) which takes into account the relative performance as well as efficiency matrix to arrive at a more useful differentiating parameter for hardware efficacy. The FPGA ends up performing best in this parameter.

In this Project the FPGA ended up outperforming GPU hardware inference by about 10 times on average while having better Energy efficiency thus making it ideal for real world Machine Learning Implementation. In the end the Project Report concludes that for Hardware acceleration of Machine Learning Models in practical real world scenarios FPGA accelerated hardware inference is the most ideal solution compared to the solutions tested in this project report.

CHAPTER 1
INTRODUCTION

Deep learning networks can now classify images better than humans, demonstrating how effective this technology is, when we observe and interact with the world, however, we do considerably more than merely identify pictures. Within our area of vision, we also locate and categories each piece. These are far more difficult activities that machines are currently unable to complete as effectively as humans. Deep learning has surpassed more traditional computer vision algorithms in the literature as the preferred method for image identification problems. Convolution neural networks excel at picture classification in the subject of computer vision, which entails categorizing images given a list of classes and having the network discover the strongest class present in the image. For these object detection techniques there are various algorithms available out of which the most popoular ones are YOLO(You only look once) and SSD-MobileNet which uses Single shot multibox detector technique in order to classify the images into thier respective category. Deep learning algorithms that are for image classifications that are the most famous one are AlexNet, GoogleNet, MobileNet , VGGNet. Out of these algorithms the one that we have used for our classification is MobileNet because Mobile Net is the lightest and the fastest one out of these and after the introduction of the SSD with MobileNet it has become even more faster and efficient .

Now Coming to the implementation these algorithms can be implemented on a CPU, GPU or FPGA .So now what we will be doing is to compare some of the factors that will decide which of the following would perform better under certain circumstances and will be able to provide us with the faster, efficient and accurate results for our object detection.

Now when talking about the implementation on FPGA the board that we have used for the object detections results is Xilinx Kria Kv-260 which is kv-26 SoM equipped which can provide us with competitive and even better performance than the others and also the main factor for using this is that it provides us great performance at lower power supply and higher Frames per Second when put under the test and further we will see the real world images where object are getting detected correctly and also show results of the sample video taken for object detection.

CHAPTER 2
IMAGE PROCESSING

Image - processing entails altering an image using a variety of ways till we achieve our aim.

The final output might be a picture or a related feature of that image. This information can be utilised for further investigation and decision-making.

2.1 What is an image ?

A 2D function F(x,y) can be used to represent a picture, where x and y are spatial coordinates. The intensity of a picture at a specific value of x,y equals the amplitude of F at that position. A digital picture is one in which the x,y, and amplitude values are all finite. It's a collection of pixels organised into columns and rows. Pixels are picture components that store information about colour and intensity. A picture can also be represented in 3D, using the spatial coordinates x, y, and z. Pixels are grouped in a matrix format. An RGB picture is what this is called.

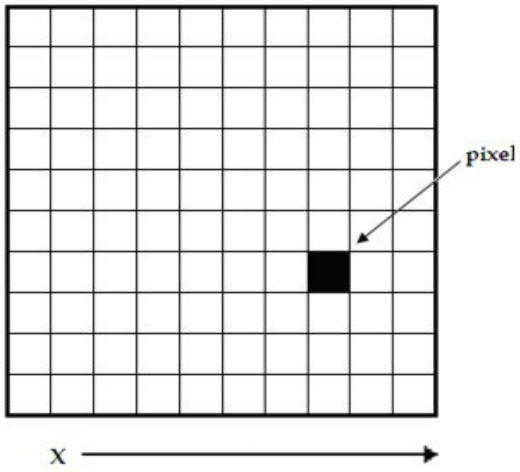


Fig 2.1: Image representation

2.1.1 Types of images:

1) RGB picture: The Red, Green, and Blue channels are three levels of a two-dimensional image.

2) Grayscale graphics have only one channel and are made up of shades of black and white.

2.2 Image Identification

Now image identification can be further be divided into 3 categories which are image classification , object detection and image segmentation.

2.2.1 Image Classification

it is based upon the salient features in which the image is classified into which category it belongs to , in this the entire image is classified as one object and further on comparison gives the result as the classified image .

Fig 2.2.1a: Image Classification. Dog in a picture

In this we can see that an entire image is being classified as dog and there is no box which is basically highlighting our object because here our entire image is the object .

Now the deep learning algorithm that is being used here for image classification is the MobileNet and the dataset that we have used for image classification is ImageNet which are 1000 classes. So what is being done is that an entire image is taken and based upon the most salient features the image is compared and based upon the most features matched in our ImageNet dataset the result is declared just like we saw here in the image that the entire image is being classified as the Dog.

So here we come across a problem that is what if there is a image something like (Shown below)

Fig 2.2.1b: Image Classification problem. Dog and cat in a picture

Here in this image we can see that there are two animals one is a cat and the other one is a dog. So if we use image classification it won't give us a appropriate output because dog and cat both are the salient feature of this image that's where object detection comes into play.

2.2.2 Object detection

So object detection is also based upon the salient features, in object detection it specifies the location of multiple objects in the image.

Object detection is the combination of:

- Classification
- Localization

Here in object detection we will be using single shot multi box detector. So what single shot multibox detector does is that it divides the image that is to be used for object detection into small patches and then based upon the combination of these patches based upon the most salient features it joins those patches and then ask the classifier to classify the image and based upon the comparison with the dataset the objects are detected .Taking an example of a mage below.

Fig 2.2.2: Object Detection. Dog in a picture

Now here in this image we can see that in our image the dog is being identified as an object and has been marked with a rectangular box around it . As we know that object detection is combination of two classification and localization, So the algorithm that we have used here is the combination of both which we have taken the MobileNet algorithm from the classification and for localization we will be using SSD basically for the object detection we will be using SSD-MobileNetv3 which is the latest version . There are other algorithms as well like YOLO (You Only Look Once) which is also a very famous algorithm for object detection but the Most light weight and faster algorithm is SSD-MobileNet .

So in this image what's happening is that the image is being divided into small patches and then combining these patches and based upon the most salient features our object s getting detected .the dataset that we have used here is the Coco which has 80 classes.

2.2.3 Image Segmentation

Image Segmentation is also based upon the salient features, Here foreground and background is segmented . In image segmentation rather than to classify the object here each pixel is classified to be part of some object .

Fig 2.2.3a: Image Segmentation Sample

So this is a normal image in which we can see a street where there are people, cars , trees so after image segmentation we will get a result (shown below)

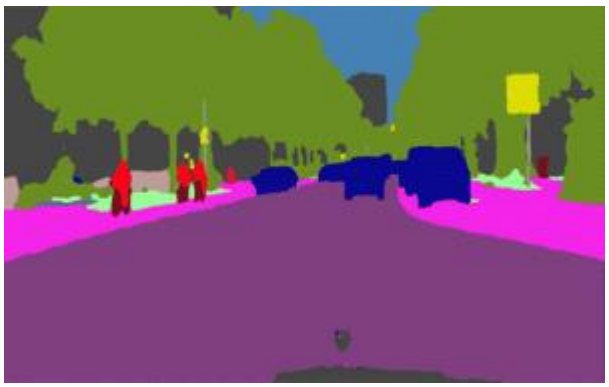


Fig 2.2.3b: Image Segmentation result

In this image we can see that each pixel is being considered as an object as we can see that people are being shown with red, cars with blue, trees with green so each pixel is getting classified.

The dataset for image segmentation is CitySpace.

2.3 Image Processing Algorithms

2.3.1 MobileNet

MobileNet is a Convolutional neural network built for mobile and embedded vision. They are based on a simplified architecture that builds lightweight deep neural networks with low latency for mobile and embedded devices using depth wise separable convolutions.

The use of automated search algorithms and network design can be used to improve the overall state of the art of classification. We may release two new MobileNet models as a result of this process: MobileNetV3-Large and MobileNetV3-Small are two versions of MobileNetV3 that are designed for large and low resource use cases, respectively. After that, these models are tweaked and applied to tasks like object detection and semantic segmentation.

MobileNetV3 Large and Small models are being developed to power on-device computer vision with the next generation of high-accuracy, effective neural network models. The new networks increase the state of the art by demonstrating how to develop effective models by combining automated search with innovative design advances.

2.3.2 SSD (Single Shot MultiBox Detector)

Deep learning networks can now classify images better than humans, demonstrating how powerful this technology is. When we observe and interact with the world, however, we do far more than just classify images. Within our field of view, we also localise and classify each piece. These are far more difficult activities that machines are currently unable to complete as effectively as humans.

Multibox - The bounding box regression technique of SSD is inspired by Szegedy's work on MultiBox, a method for fast class-agnostic bounding box coordinate proposals. Interestingly, in the work done on MultiBox an Inception-style Convolutional network is used. The 1x1 convolutions that you see below help in dimensionality reduction since the number of dimensions will go down (but "width" and "height" will remain the same).

The loss function in MultiBox also included two important components that made their way onto SSD:

1) Confidence Loss: this metric indicates how confident the network is in the calculated bounding box's objectless. This loss is calculated using categorical cross-entropy.

2) Location Loss: this metric indicates how far the network's predicted bounding boxes differ from the training set's ground truth bounding boxes. Here, L2-Norm is employed.

SSD is a deep neural network-based approach for recognising objects in pictures. Per feature map location. The main working of SSD is that it firstly converts the output into discrete chunks which are represented in terms of building bounding boxes. It includes various aspect ratio and different scales.

Firstly for each and every default box a probable score is calculated. This score is then adjusted to fit the object shape with a box. In addition to that to be better compatible with different sizes the network looks at different resolutions of feature maps and then outputs probable values. SSD ids the a very simple algorithm that worked on proposed objects. The way it achieves this is that it first eliminates proposed object development and the next step which is subsequent pixel resampling or even feature resampling is eliminated while encapsulating all in a single straightforward network. As a result SSD is much simpler and required less computational power integration very well with other systems.

For Training the SSD only an input image is required along with an additional truth box for each subject. After then a variety of different aspect ratio is passed as test cases to examine rare cases of original boxes using Convolutional theorems. After this step these original boxes are then compared against round truth to give us the probability of shape of the object as well as its class along with the confidence probability for each class. For example, the two original boxes which includes a cat and a one with a dog are being output as positive implying truth while the other s are output as false implying negatives or false.

Multi-scale feature maps for detection - Detection using multi-scale feature maps At the end of the truncated base network, we add Convolutional feature layers. These layers shrink in size over

time, allowing detections to be predicted at several scales. Each feature layer has a different Convolutional model for predicting detections.

Convolutional predictors for detection - Using a collection of Convolutional filters, each new feature layer may generate a defined set of detection predictions. On top of the SSD network design, they are indicated. For predicting the Basic detection we use a scenario of a tiny 3 by 3 Kernel which outputs a relative score for the predicted probability of the shape offset as well as category compared to original box case. The layer used for this task is of size m by n having p number of channels. The kernel than produces a value at every m by n region in the area it is being used. The output value of the bounding box offset are compared to a standard

Default boxes and aspect ratios – For considering the feature of the map cell by default a bounding box is put at the front of the Neural Network. Using convolution each feature of the map is attached to the tile while keeping the corresponding relationship the same as before. The probability of each offset in proportion to the original box shape is calculated with the per class score that puts this probability of class presence in the bounding box. Thus for class c offset values are calculated in proportional to the original box shape out of k for every box shape as well as for each and every position.

2.4 Some other image processing algorithms

2.4.1 Morphological image processing

Because noise may damage binary regions formed by simple thresholding, morphological image processing aims to eradicate faults from binary images. It also helps with image smoothing by employing opening and closing procedures.

To broaden morphological processes, grayscale images can be employed. It's made up of non-linear techniques that deal with an image's feature structure. It is governed by the numerical values of pixels rather than the order in which they are placed. This approach analyses a picture by comparing the suitable neighborhood pixels to a small template called as a structuring element, which can be placed in a variety of locations in the image. A small 0 and 1 value matrix serves as a structural element.

The two basic morphological image processing processes are dilation and erosion:

1) The dilation process in a photograph adds pixels to the limits of the item.

2) During the erosion process, pixels from the object's edges are eliminated.

The amount of pixels to be removed is directly proportional to the size original picture of the structuring element. A structuring element is a 0/1 matrix that can be any form or size. It is placed in all possible locations throughout the image and compared to the pixels in the immediate vicinity. The square structural element 'A' fits within the item we want to choose, 'B' intersects it, and 'C' is on the outside.

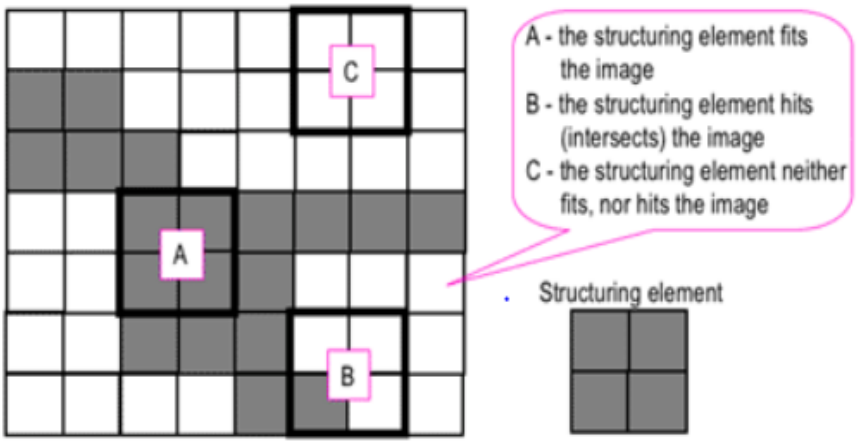


Fig 2.4.1a: Morphological Image Processing

The square structuring element ‘A’ fits in the object we want to select, the ‘B’ intersects the object and ‘C’ is out of the object.

2.4.2 Gaussian Image Processing

Gaussian blur, also known as Gaussian smoothing, is the result of blurring a photograph with a Gaussian function. It's a method for obfuscating details and minimizing visual noise.

It's a method for obfuscating details and minimizing visual noise. Looking at a picture via a transparent screen provides the same visual effect as blurring it. It's sometimes used in computer vision as a deep learning data augmentation strategy or for image improvement at various scales.

To take use of the Gaussian blur's separable property, divide the operation into two stages. The First pass of this algorithm works to basically blur the 1 dimensional image kernel in both the vertical as well as the horizontal axes. For the 2nd pass the same one Dimensional Kernel Blur outs the remaining Axes.

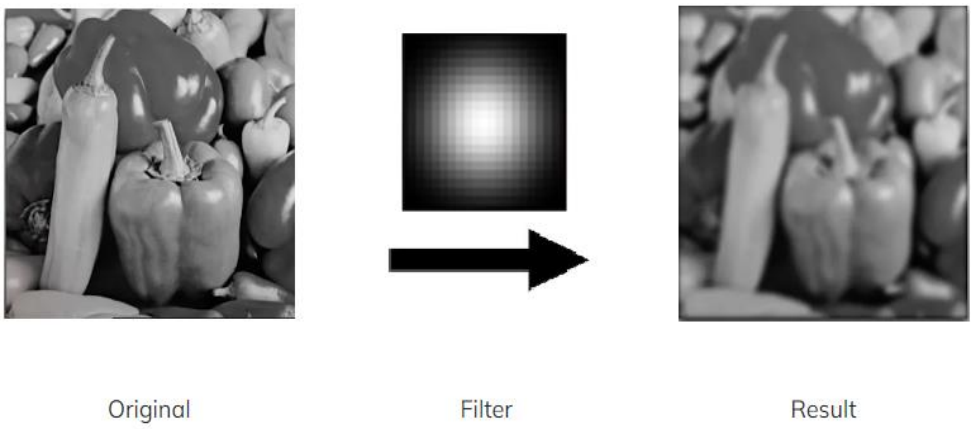


Fig 2.4.2: Gaussian Image Processing

Some of the edges are a little less detailed than others. The pixels closest to the centre are given more weight by the filter than those further away. Low-pass filters, such as Gaussian filters, attenuate high frequencies. It's a frequent technique for edge detecting.

2.4.3 Fourier Transform in image processing

The Fourier Transform is primarily utilized to divide the image into its sine component and cosine component. . It may be used for picture reconstruction, compression, and filtering, among other things. We'll consider the discrete Fourier transform because we're talking about pictures. Consider a sinusoid, which is made up of three components:

1) Magnitude – related to contrast

2) Spatial frequency – related to brightness

3) Phase – related to color information

2.5 Image processing using Neural Networks

Neural Networks are multi level structure that is made up of data points which are called nodes or the neurons of the neural structure. These neurons are what do the basic processing in the NN. They perform somewhat similarly to human brain that's why the analogy. The main methodology tis that the input data which is fed to the NN is processed upon and the Pattern in it is detected though weights and biases and the result is given in terms of activations in the output layer

A basic neural network has three layers:

i. Input layer(at front of NN)
ii. Hidden layer(at middle of NN)

iii. Output layer(at the end of NN)

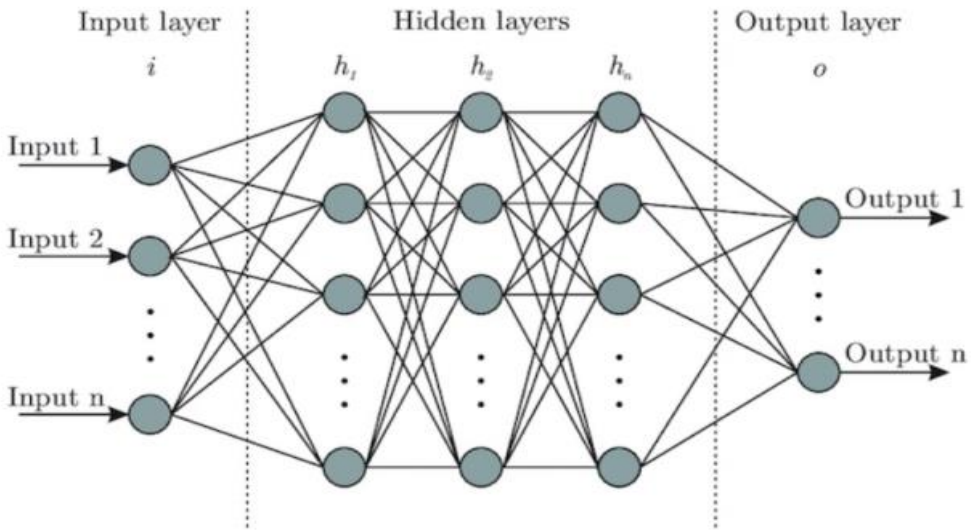


Fig 2.5: Neural Networks

We input the data in the input layers of the CNN. The output layer gives us with the probability of the result and the middle layer does the calculations of the weights and biases of the Neural Network. A neural network should have at least one hidden layer.

The neural network's basic operation is as follows:

1) Consider the following example: each pixel is provided as input to each neuron in the first layer, and neurons in one layer are connected to neurons in the next layer through channels.

2) Weight is a numerical number applied to each of these channels.

3) The inputs are multiplied by the weights, and the resulting weighted sum is supplied to the hidden layers.

4) The output from the hidden layers is routed through an activation function, which determines whether or not a certain neuron is active.

5) Data is sent to the next buried levels by the stimulated neurons. Data is transmitted through the network in this manner, which is known as Forward Propagation.

6) The neuron with the greatest value predicts the output in the output layer. The probability values are the outputs.

7) To determine the error, the anticipated output is compared to the actual output. Backpropagation refers to the process of transferring information back over a network.

8) Weights are modified based on this information. This forward and backward propagation cycle is repeated on several inputs until the network accurately predicts the output in the majority of situations.

9) The neural network's training procedure is now complete. In some circumstances, the time required to train the neural network may be excessive.

2.5.1 Generative Adversarial Networks

GAN is basically made out of two the models out of which one is the Generator and the other one is the Discriminator. As the term Discriminator Suggests that it will discriminate the false or incorrectly identified image whereas here the generator prepares to obtain real look alike images now coming back to the discriminator it will discriminate whether the image is correct or incorrect

this generally happens due to the lack of availability to view the original image due to this it might not be able to provide us with the correct result in the starting that why generator turns out to be slow in the start but unlike generator discriminator here can view the original pictures even though this sounds pretty clear but the correct and incorrect images are mixed together so it is difficult for the discriminator to discriminate or evaluate the pictures.

As we are working on the generator we need various types of different outputs that why feed it with some external noise or disturbance so that it will be able to create different type of instances and not end up with the same thing here the discriminators comes into play as the discriminator helps it to improve the outcomes. After specific amount of time Due to external noise there will be more complicated outcomes which will end up making the job of discriminator hard even the end-user will be satisfied with the outcome as discriminator gets to know about the pattern that a generator will be following as the amount of data increases the outcome improves.

That’s why these are fantastic for creating and manipulating images. Face Aging, Photo Blending, Super Resolution, Photo Inpainting, and Clothing Translation are some of the uses of GANs.

2.5.2 Convolutional Neural Network

1. ConvNets, or Convolutional neural networks, are made up of three layers:
2. Convolutional Layer (CONV): This is the primary component of CNN, and it is in charge of executing convolution operations.

3. The Kernel/Filter is the component in this layer that performs the convolution process (matrix). The Kernel will give out horizontal and vertical modifications which are directly linked to stride rate.

4. Pooling Layer (POOL): is in charge of dimensionality reduction. The aim here is to decrease the computing requirement for this data processing. If we look at the types there are two types of pooling present ; the First type is Maximum pooling and the second type is minimum pooling. The highest valued area filled by the image Kernel is given by max pooling. Whereas the average of area of image filled by the kernel gives average pooling.

5. Completely Connected Layer (FC): The fully linked layer (FC) works alongside the serialized input suggesting that each input node is connected to each and every one of neuron. These layers are present in the end of this CNN structure.

6. CNN is mostly used to extract features from images using its layers. For classifying images using this CNN approach we determine each layer between 0 &1. 0 giving minimum probability and 1 giving maximum probability for activation

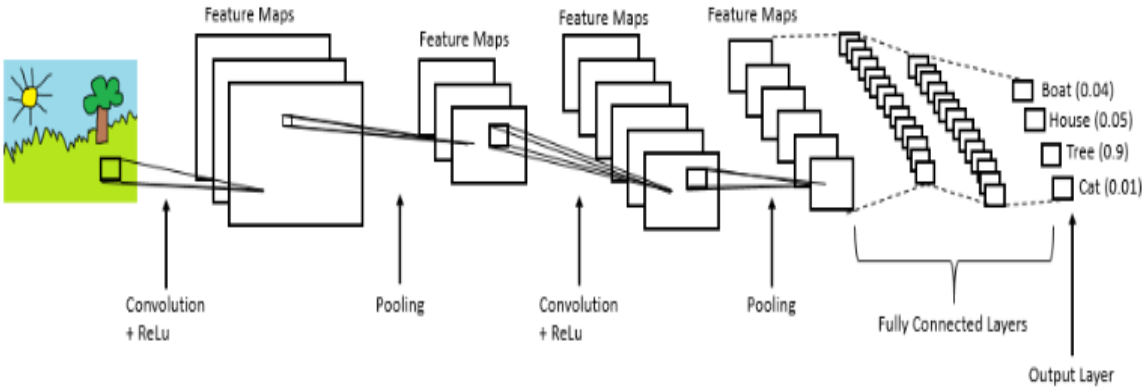


Fig 2.6: Convolutional Neural Network

CHAPTER 3
HARDWARE

3.1 What is FPGA ?

FPGA or Field Programmable gate array is an advance version of PLD devices, In FPGA it comes with logic blocks that has gate arrays, so these gate arrays are divided into small units of gate arrays which are called CLB or Configurable Logic Blocks. There can be 100 gates in a unit or CLB but there will be several of these units with 100 gates each, so in total there will be 10000 or 100000 or can be even more amount of gates these 100000 gates will be grouped into smaller groups which will contain 50 or 100 gates these groups or units are CLB's. So the each CLB will have specific function to perform. Now all the CLB's are gate equivalent that does not mean that all the components inside that CLB will be a gate a typically CLB will consist of a MUX, Flip Flops, register and few gates so that a small a design can be made out of all these .Each CLB will have some standard feature but different configurability of MUX, gates, Flip Flops. So, our main job is to break the circuit or design that is to be implemented into mall groups and configure those groups into the CLB's and then does the interconnection efficiently.

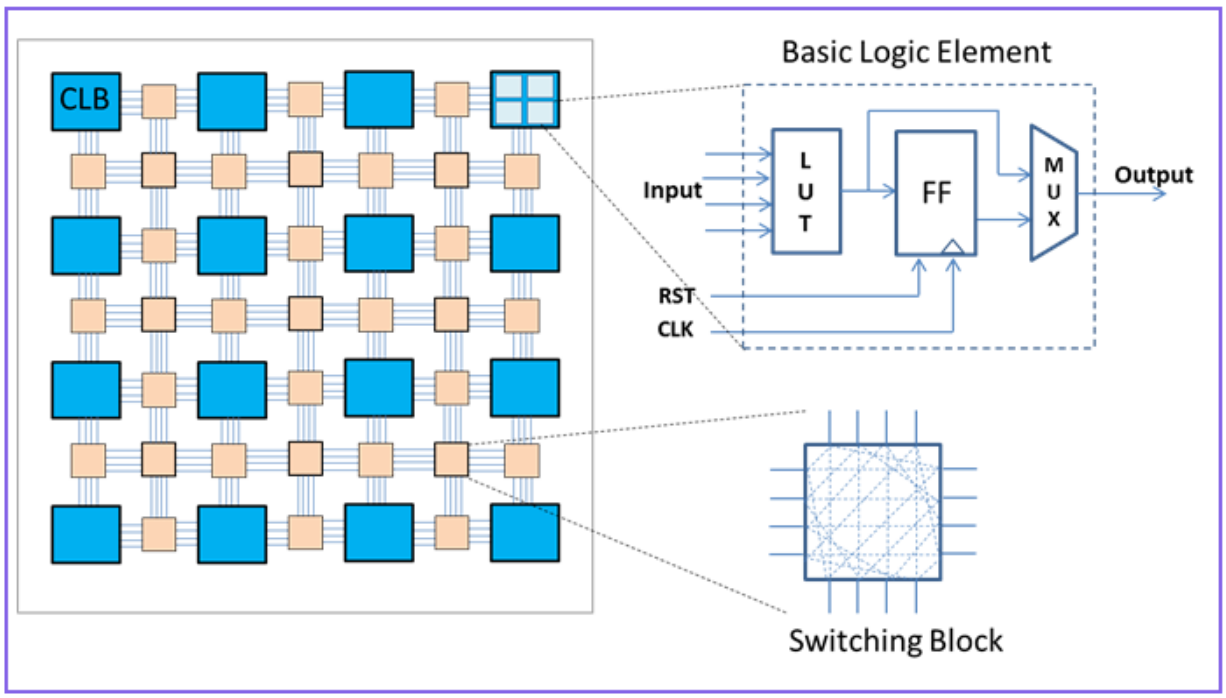


Fig 3.1: FPGA Basic Logic Element

3.2 Xilinx kria kv-260

3.2.1 Overview

The Xilinx KriaTM KV260 Vision AI Starter Kit is a ready-to-use platform for developing vision applications without the need for extensive hardware design skills. A non-production version of the production K26 SOM is installed in the KV260. This SOM, along with a fan heat sink, is attached to a vision-optimized evaluation carrier card, which has multi-camera capability via ON Semiconductor Imager Access System (IAS) and Raspberry Pi interfaces.

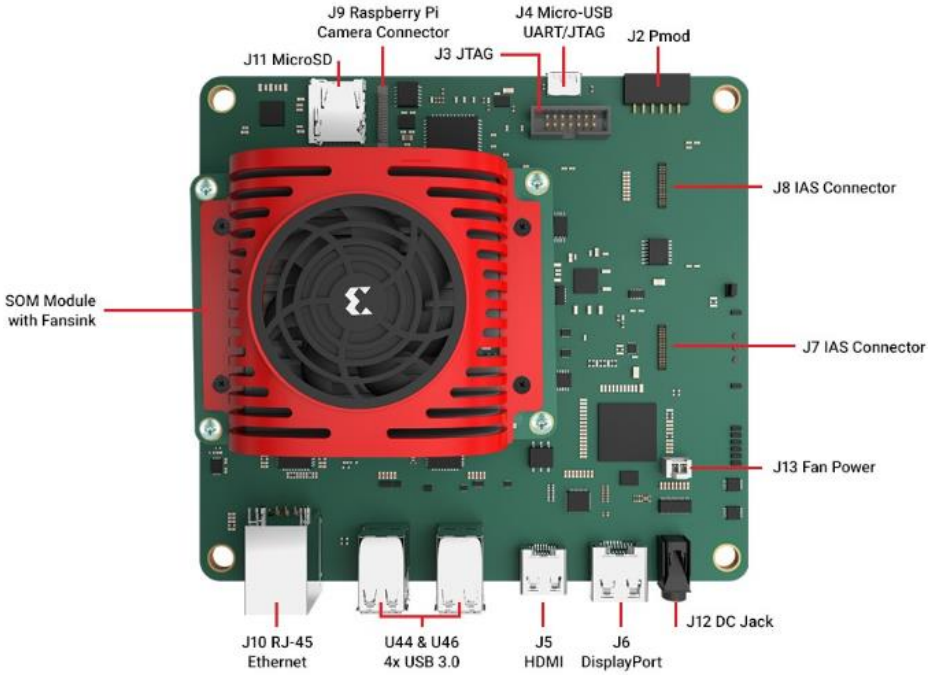


Fig 3.2.1a: Kria KV260 Board Diagram

Fig 3.2.1b: Actual Kria KV260 board Connected to PC

Features

- Multi-Camera Support: Up to 8 interfaces
- Raspberry Pi MIPI sensor interface
- USB camera support
- HDMI, DisplayPort Outputs

Flexible Connectivity

- 1Gb Ethernet
- USB 3.0 / 2.0

3.2.2 Specification

PARAMETER	KV260
Device	Zynq® UltraScale+™ MPSoC
Form factor	SOM + Carrier Card + Thermal Solution
Starter kit dimensions	119mm x 140mm x 36mm
Thermal cooling solution	Active (Fan + Heatsink)
System logic cells	256K
Block RAM blocks	144
UltraRAM blocks	64
DSP slices	1.2K
Ethernet interface	One 10/100/1000 Mb/s
DDR memory	4GB (4 x 512Mb x 16-bit) [non-ECC]
Primary boot memory	512Mb QSPI
Secondary boot memory	SDHC card
Device security	Zynq UltraScale+ MPSoC hardware root of trust (RoT) in support of secure boot. Infineon TPM2.0 in support of measured boot.
Image sensor processor	OnSemi AP1302 13MP ISP
IAS MIPI sensor interfaces	x2
Raspberry Pi camera interface	x1
Pmod 12-pin interface	x1
USB3.0/2.0 interface	x4
DisplayPort 1.2a	x1
HDMI 1.4	x1

Fig 3.2.2: Kria KV260 Hardware Specification

3.3 kria kv-26 SoM

The Kria K26 SOM is a vision AI and video analytics SOM designed to meet current and future industry demands. The Kria SOM combines an adaptable SOC based on the Zynq UltraScale+TM MPSoC architecture with all of the core components required to support the SoOC in a package small enough to fit in the palm of your hand (such as memory and power).

It's easy to customise production deployments. The Kria SOM is paired with a simple end-user-designed carrier card that includes connectivity and other components unique to that user's end system**.**

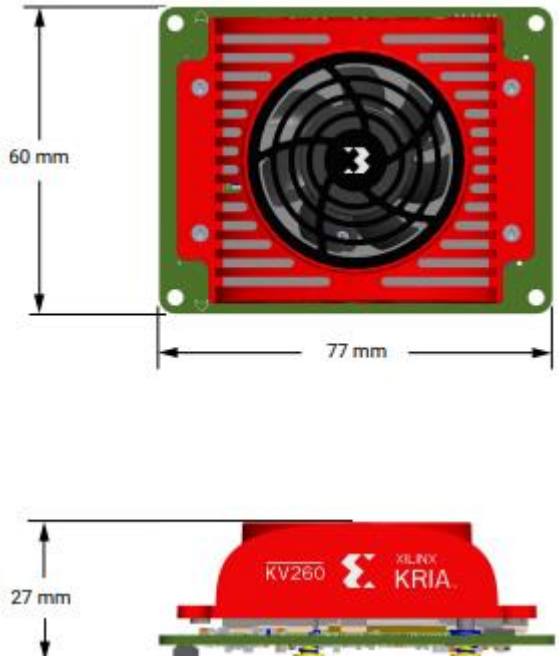


Fig 3.3: Kria KV260 SOM (system on module)

Unlike GPUs, where the data flow is fixed, Xilinx hardware offers flexibility to uniquely reconfigure the data path to achieve maximum throughput and lower latencies. Also, the programmable data path reduces the need for batching, which is a major drawback in GPUs and becomes a trade-off between lower latencies or higher throughput. The Kria SOM's flexible architecture has shown great potential in sparse networks, which is one of the hot trends in current ML applications. Another important feature—and one that makes Kria SOMs even more flexible is the any-to-any I/O connection. This enables the K26 SOM to connect to any device, network, or storage device without the need for a host CPU.

Intelligent applications require privacy, low power, security, and cheap cost in addition to sub-millisecond latency. The Kria K26 SOM, which is based on the Zynq MPSoC architecture, offers

best-in-class performance/watt and lower total cost of ownership, making it an attractive contender for edge devices. Because Kria SOMs are hardware adjustable, the K26's solutions are scalable and future-proof.

3.4 kria kv -260 overall block diagram

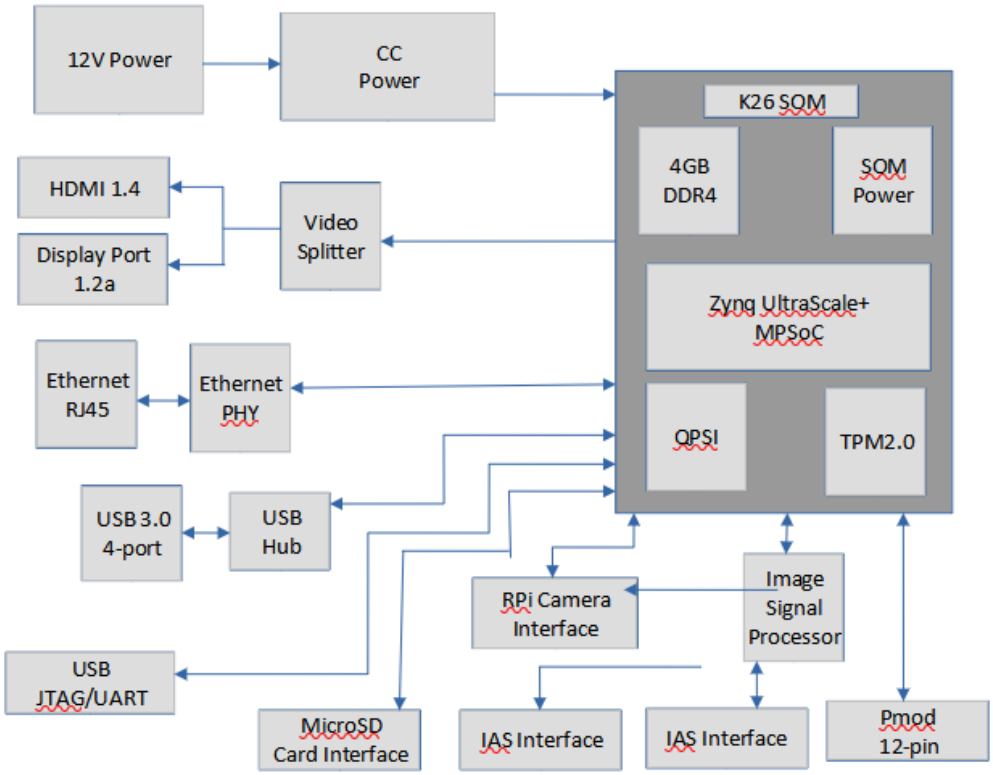


Fig 3.4: Kria KV260 overall blockdiagram

The performance was then compared on the K26 SOM with two DPU configurations, namely B3136 and 4096 DPU, on a couple of the industry's leading models such as Tiny Yolo V3, SSD Mobilenet V1, ResNet50, and so on. The results were compared to benchmarking data available on Nvidia's website. The K26 SOM outperformed both the Jetson Nano and Jetson TX2 devices, according to the findings. Not only did the Kria SOM outperform the Jetson Nano in terms of throughput by more than 4X, but it also outperformed the Jetson TX2 in terms of performance/watt by more than 2X.

3.5 Vehicle Detection

We will create a module that will be responsible for detecting the number of cars in the image that has been provided as input. It will offer the number of vehicles in each vehicle class, such as car, bike, bus, truck, and rickshaw, as an output.

3.5.1 YOLO and CNN

YOLO is popular because it has a high level of accuracy and can run in real-time. To create predictions, the method "only looks once" at the picture in the sense that it only takes one forward propagation pass through the neural network. It then returns detected objects together with bounding boxes after non-max suppression (which ensures that the object detection algorithm only identifies each object once). A single CNN is going to guess where several bounding boxes are and determine it with a number with YOLO. We'll use Google Colab for vehicle detection.

3.5.2 Python Packages

A package is the collection of modules, where a module refers to design and implementation of specific functionality to be implemented in a program.

3.5.3 OpenCV

It stands for Open Source Computer Vision Library. This collection contains over 2000 well optimized algorithms for computer vision and machine learning. Opencv may be used in image processing in a variety of ways, some of which are given below:

1) Converting photos from one colour space to another, such as BGR to HSV, BGR to grey, and so on.
2) Performing image thresholding, such as basic thresholding, adaptive thresholding, and so on.
3) Applying custom filters to photos and blurring images are examples of image smoothing.
4) Using photographs to perform morphological processes
5) Then Image pyramids are constructed.
6) Using the GrabCut method to extract foreground from photos.
7) Watershed algorithm is used to segment images.

CHAPTER 4
IMPLEMENTATION

4.1 Output Methodology

A total of 12 samples of different scenarios were tested involving varying complexity with CPU inference against GPU accelerated inference as well as FPGA accelerated inference. The results are plotted in a table and Graph are drawn to better analyze the results.

4.1.1 Hardware testing methodology

- Tested SSD against CPU vs GPU vs FPGA hardware inference for 12 video samples.
- Calculate relative Performance from fps average of CPU vs GPU vs FPGA
- Relative performance plotted in Graph

4.1.2 Global chip shortage

The Semiconductor shortage and high demand caused GPU prices to increase several fold to the point that even several years old GPU end up being more expensive now than the time they were launched (several years ago). Also Global Tread of cyptocurrency mining is launching of several thousand crypto startups is future deteriorating the solution as cryptocurrency mining is primarily done in GPU and the algorithms being run on them are very heavy on power draw causing not on high power consumption put future stressing the electricity grid which is also a big challenge for climate change. There is also a question on cryptocurrency need because in the end we us mostly use fossil fuels for electricity generation and basically cryptocurrency mining is just the guess (requires computational power) of random computer generated blocks in a competitive block chain. The main goal of cryptocurrency was decentralization of money from banks and government thought the same can be achieved by other methods such as proof of stake method rather than proof of work method. The Algorithm of such crypto currencies is such that of each block to be mined it has to be guessed by the system. The average guess time it takes to currently predict a Crypto key of the current block in the block chain is inversely proportional to computational power. And since all the nodes or Computers are also predicting the Cryptographic key for the same block, the more computational power you have the more is the probability to correctly mine the block in the block chain. So people/organizations involved in cryptographic mining end us having Several High end GPUs in SLI all mining the same block in the block chain. Since there is a Crypto boom many of these organizations end us being very large financially and end up creating large Crypto farms with several thousand high ends GPU to increase their chances of mining the block much faster than other nodes. These farms take us energy equivalent to Entire towns and smaller cities raising the question on environmental effects of crptocurrency mining. Also most famous crypto currencies such as bitcoin and ethereum currently run on proof of work system it is slates to change for ethereum soon. As ethereum plans to implement proof of stake instead of proof of work by the end of 2022 hopefully will relive the already stressed GPU market. As a result we used google colab for GPU Hardware acceleration for Machine Learning algorithms for free.

4.1.3 Google Colab

For Output Implementation of Hardware acceleration of Machine Learning of FPGA we tested the Kria KV 260 against traditional CPU rendering and GPU accelerated Inference. For This Purpose 12 samples were taken and Tested across each hardware for frames per second (fps) performance. The Samples have been carefully selected to represent the real world scenario which is described

in more detail in section 4.6.

The three hardware solutions were performed on following hardware:

1. For CPU Inference the system was tested on Intel core i3 3240 a dual core quad thread processor with max TDP of 55 Watts clocking at 3.3 GHz which was the fastest CPU available to us.
2. For GPU Inference due to lack of GPU with us and a global chip shortage causing inflated GPU prices as a result of Covid pandemic we looked at cloud solutions for GPU inference.
3. As a result for GPU testing and the lack of a CUDA capable GPU (For running GPU acceleration on Nvidia GPUs) we used cloud solution Goggle Colab.
4. For FPGA we are using the Xilinx Kria KV 260 SOM board which we won under Xilinx Adaptive computing challenge 2021.

4.2 Model Training and Description

For Model Training we are using a Pre-trained Model since the Project only deals with inference performance. The model Used is SSD with Pre calculated Weights and Biases in inference graphs that are imported into the Jupyter notebook.

4.3 Type 1 – CPU Inference

For CPU analysis the SSD code is being implemented in the Jupyter notebook. The code is being run on Intel i3 – 3240 Processor for Inference

4.3.1 Type 1 – CPU Inference Rendering

Usage of available CPU machine

Implementation in Jupiter notebook

Requirements

- Open CV version 4.9.2 ; doesn't run in 4.5.5 (need to upgrade)
- Python 3.6 ;doesn't run in latest python version due to compatibility issue in open cv
- For additional system GPU acceleration need Nvidia CUDA though need a CUDA capable Nvidia GPU (not available to us)

4.3.2 Jupyter Notebook Implementation

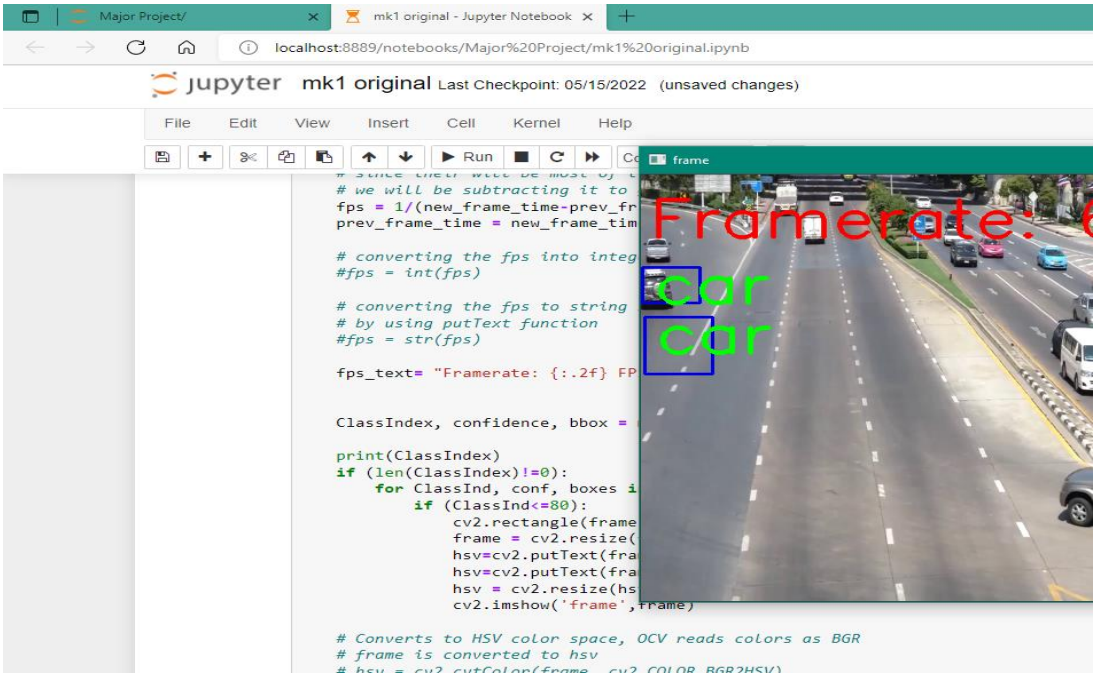


Fig 4.3.2: SSD code being implemented in CPU using Jupyter notebook

4.3.3 Power Draw – HW monitor + CPU-Z

Power Draw can be analyzed using HW Monitor software which is an open source utility that provides very accurate Temperature monitoring of CPU for performance analysis and clock speed evaluation we are using Windows Task Manager and CPU z software.

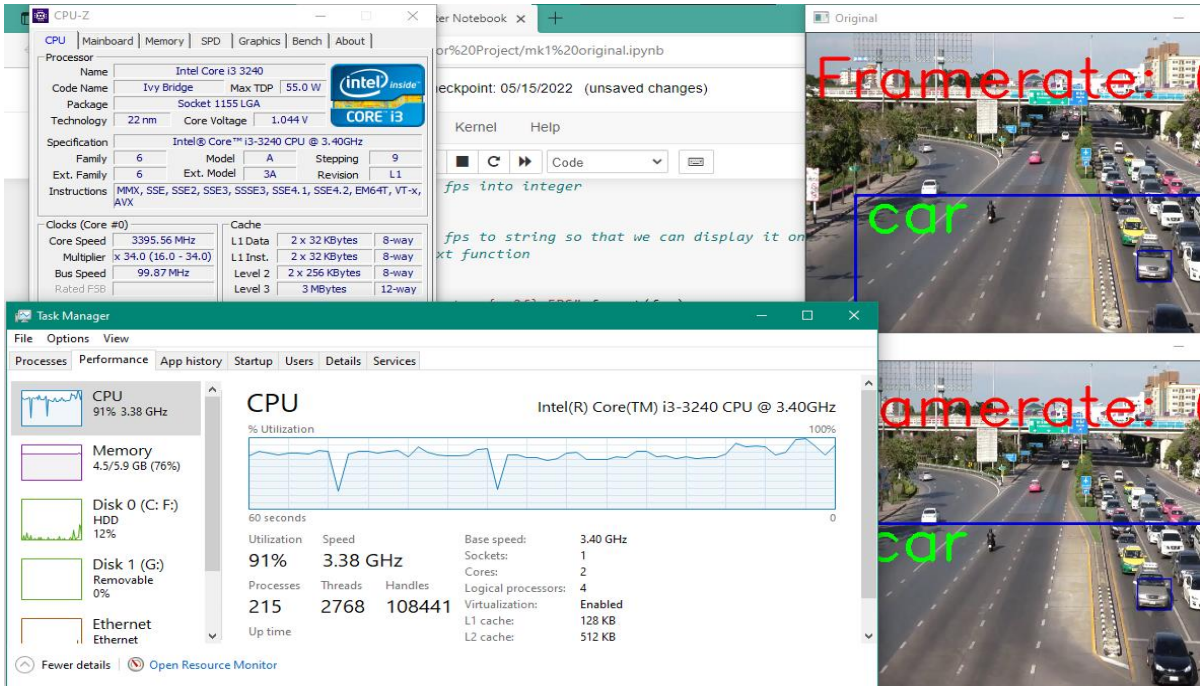


Fig 4.3.3a: Performance cap of CPU during Jupyter python 3 execution using CPU-z and Windows Task Manager

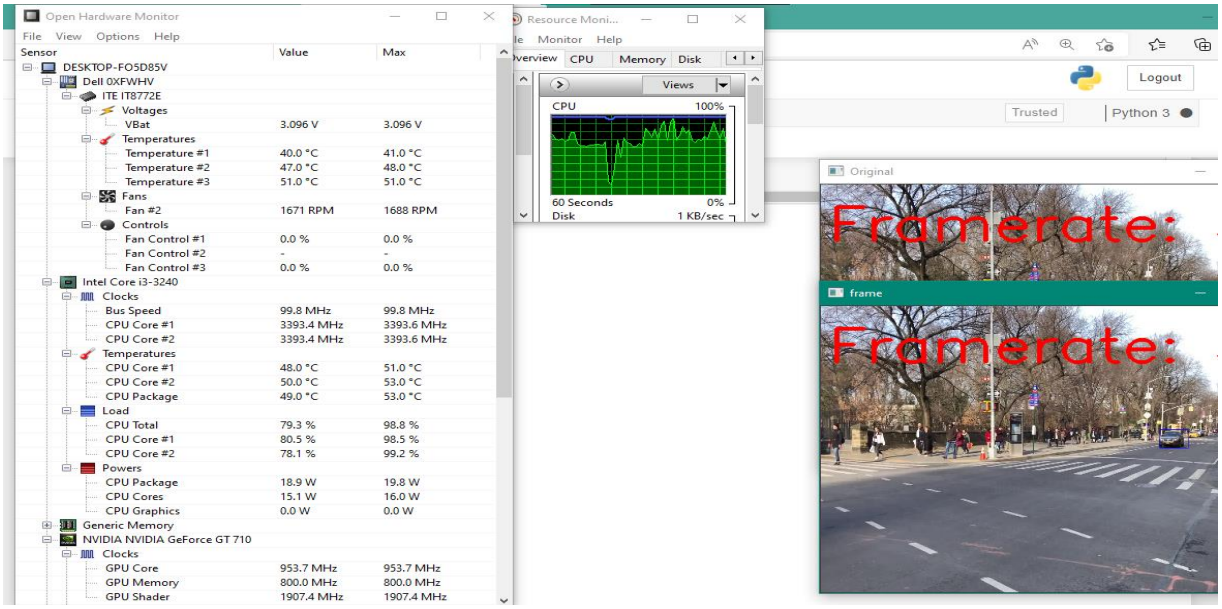


Fig 4.3.3b: Power Draw of CPU during Jupyter python 3 execution using HW bot Monitor

4.4 Type 2 – GPU Inference

For Testing the Inference of GPU we are using Google Colab GPU runtime

4.4.1 Type 2 GPU Inference Rendering

Implementation in Google Colab itself.

Usage of Google Colab GPU runtime acceleration machine from:

Runtime> change runtime type > hardware acceleration > GPU

Requirements

- Open CV version 4.9.2 ; doesn't run in 4.5.5 (need to upgrade)
- Avi input only
- Runtime gets recycled after some in free version so need to reupload files again
- Gets disconnected from server after some time if not in active use
- Limit in disk space available and ram assigned

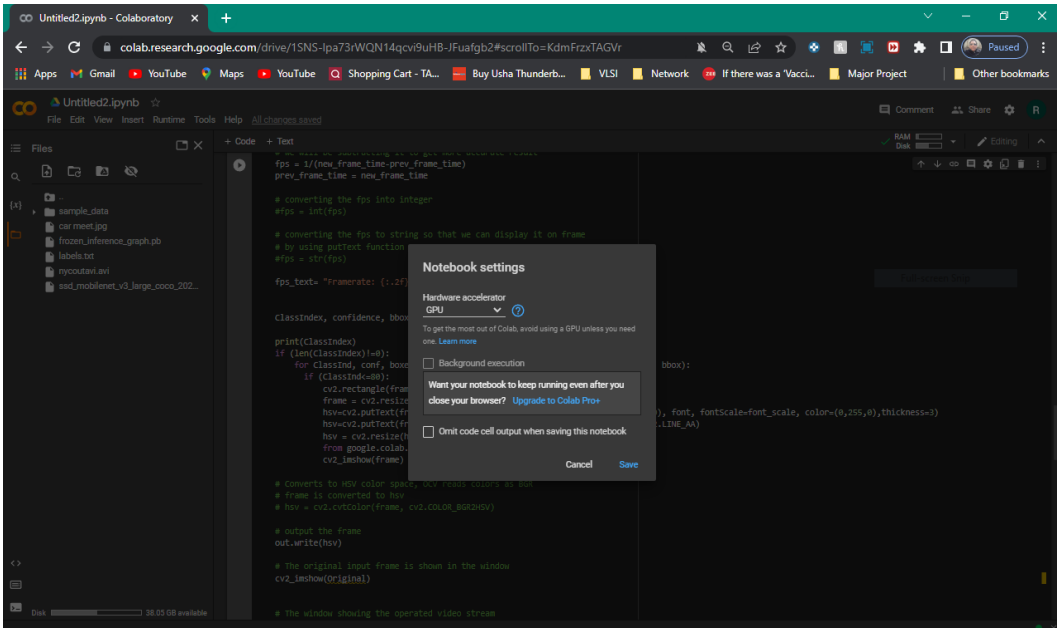


Fig 4.4.1a: Running GPU hardware acceleration in Google Colab

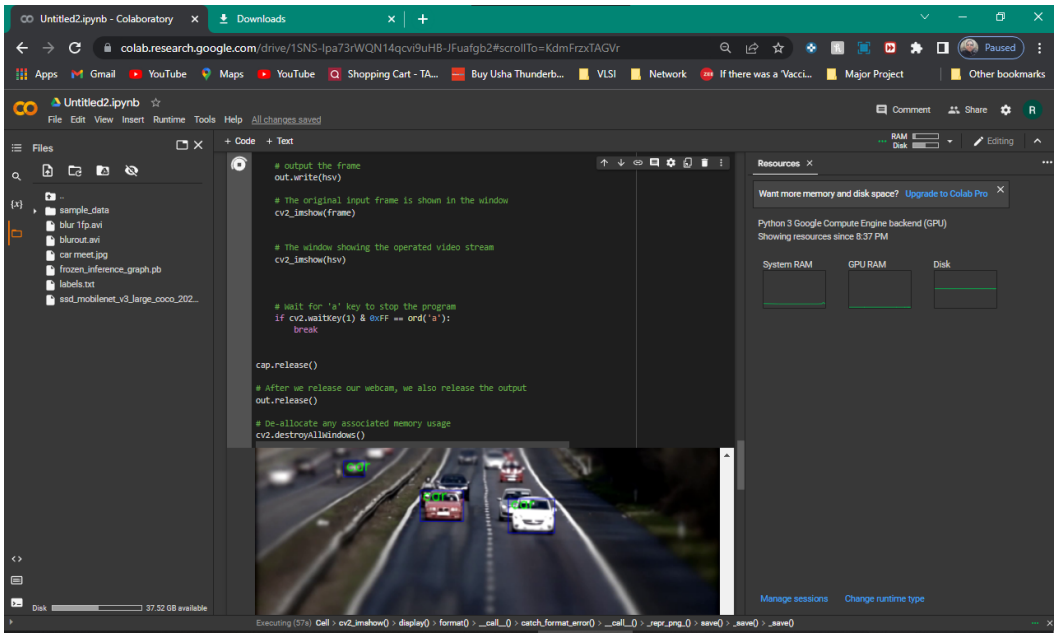


Fig 4.4.1b: GPU Inference being run on Google Colab

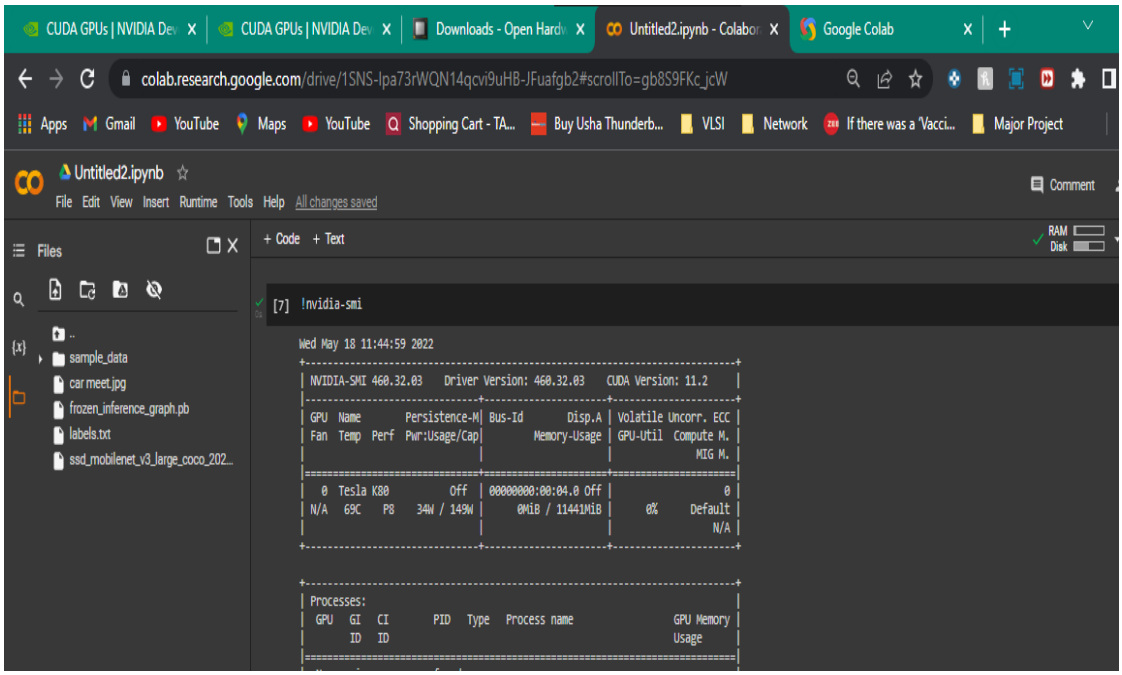


Fig 4.4.1c: Nvidia smi function to analyzed power draw on Nvidia GPU

4.5 Type 3 – FPGA (Kria KV 260) Inference

For Testing the Inference of FPGA we are using Kria KV60 board we use Xilinx Kria KV260 FPGA Board.

4.5.1 Type 3 FPGA Inference Rendering

Usage of Xilinx Kria KV 260 FPGA board

Implementation in Ubuntu/Petalinux

Requirements:

- FPGA board
- Included getting started kit

4.5.2 Petalinux

The PetaLinux Tools package includes everything you'll need to modify, create, and deploy Embedded Linux solutions on Xilinx processors. The solution, which interacts with the Xilinx hardware design tools to simplify the creation of Linux systems for Xilinx FPGA Hardware, is tailored to increase design productivity.

Developers can use these tools to make changes to the boot loader, Linux kernel, or Linux applications. They may use the bundled complete system simulator (QEMU) or actual hardware via network or JTAG to add new kernels, device drivers, programmes, libraries, and boot and test software stacks.

1. Custom BSP Generation Tools - PetaLinux tools allow developers to align the software platform with the hardware design when new features and devices are added. The PetaLinux tools will create a bespoke Linux Board Support Package for you, complete with device drivers for Xilinx embedded processor IP cores, kernel, and boot loader settings. Software engineers may concentrate on their value-added applications rather than low-level development duties with this capability.

2. Linux Configuration Tools - Tools for customizing the boot loader, Linux kernel, file system, libraries, and system settings are included in PetaLinux. These setup tools are completely aware of Xilinx hardware development tools and custom-hardware-specific data files, allowing device drivers for Xilinx embedded IP cores, for example, to be automatically produced and deployed based on the device's engineer-specified address.

3. Software Development Tools -PetaLinux utilities include development templates that let software developers to construct custom device drivers, applications, libraries, and BSP settings. The PetaLinux tools enable developers to package and share all software components for easy installation and usage among PetaLinux developers after the product's software baseline (BSP, device drivers, core apps, etc.) has been built.

Xilinx official guide was followed for Petalinux creation using Xilinx BSP following the user guide mentioned in reference.

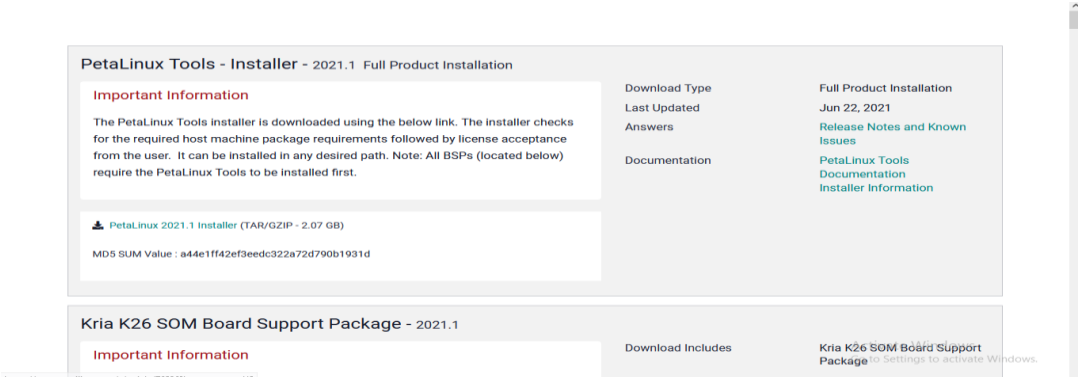


Fig 4.5.2a: Petalinux Board Support Package (BSP) for creating the Petalinux project

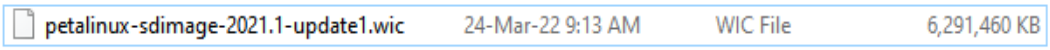


Fig 4.5.2b: final .wic Petalinux project file created

4.5.3 Balena Etcher

BalenaEtcher is a free and open-source application for creating live SD cards and USB flash drives by writing image files such as.iso and.img files, as well as zipped folders, to storage media.

Balena created it, and it's licensed under the Apache License 2.0.

The petalinux project file created in section 4.5.2 is burned in the SD card using Balena Etcher Tool. The SD card can then be inserted into the Kria SOM board for running the Petalinux project

4.5.4 Tera Term

Tera Term is a free open source terminal emulator program for windows which is capable of emulating different types of terminals. It supports SS1 and SSH2 serial port connections and is often used to automate tasks related to remote connections from PC to other hardware. In our case since the FPGA board relies on serial communication with the SOM the Tera Term provides that serial port connectivity to communicate directly with this FPGA Fabric.

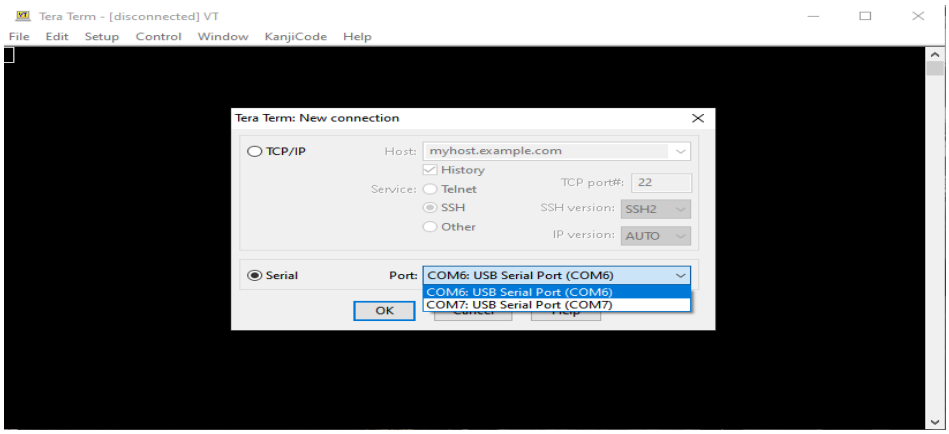


Fig 4.5.4a: COM port setting for Kria KV 260. Note that board has to be found among all the available serial COM ports.

Fig 4.5.4b: After booting into the Kria SOM setting SOM password for the default Petalinux user

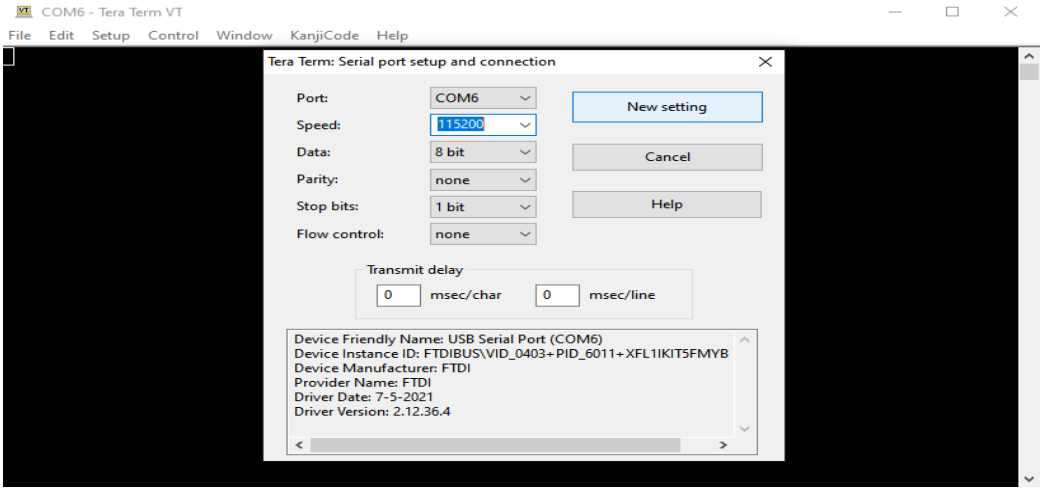


Fig 4.5.4c: COM port setting for Kria KV 260. Note that board will fail to correctly boot without these settings as the default baud rate is too low for serial communication with Kria SOM

The Default setting of COM port provide gibberish output so they need to be changed to following settings for effective serial communication between the board and the Kria SOM

- Correct COM port for Krai (need to find that out)
- Speed/baud rate - 115200
- Data - 8 bit
- Parity - none
- Stop bit - 1 bit
- Flow control - none

4.5.5 Xmutil Packagegroup

Xmutil package group - xmutil is in charge of loading all of the accelerated apps.

The tool may be used to check the platform's status, manage accelerated apps, and perform a variety of other tasks.

```
xilinx-k26-starterkit-2021_1:~$ sudo xmutil getpkgs
Password:

Searching package feed for packagegroups compatible with: kv260

accelize-packagegroup-kv260-drmdemo.noarch                              2.0-1.pl
2021_1.0                                        accelize-2021_1-noarch

packagegroup-kv260-aibox-reid.noarch                                    1.0-r0.0
                                                oe-remote-repo-sswreleases
-rel-v2021.1-generic-updates-rpm-noarch
packagegroup-kv260-defect-detect.noarch                                 1.0-r0.0
                                                oe-remote-repo-sswreleases
-rel-v2021.1-generic-updates-rpm-noarch
xilinx-k26-starterkit-2021_1:~$
```

Fig 4.5.5a: Different packagegroups compatible with kria SOM

```
COM6 - Tera Term VT
File  Edit  Setup  Control  Window  KanjiCode  Help
xilinx-k26-starterkit-2021_1:~$ sudo xmutil      listapps
                    Accelerator                          Base           Type
   #slots         Active_slot

                         kv260-dp                        kv260-dp       XRT_FLAT
        0                     0,
                    kv260-smartcam                  kv260-smartcam      XRT_FLAT
        0                    -1
              kv260-nlp-smartvision          kv260-nlp-smartvision      XRT_FLAT
        0                    -1
Socket 9 closed by client
xilinx-k26-starterkit-2021_1:~$ sudo xmutil      unloadapp
DFX-MGRD> daemon removing accel at slot 0

DFX-MGRD> Removing accel kv260-dp from slot 0

Accelerator successfully removed.
Socket 9 closed by client
xilinx-k26-starterkit-2021_1:~$ sudo xmutil      loadapp kv260-smartcam
DFX-MGRD> daemon loading accel kv260-smartcam

DFX-MGRD> Successfully loaded base design.

Accelerator loaded to slot 0
```

Fig 4.5.5b: Accelerator being loaded in the Kria SOM using Tera Term

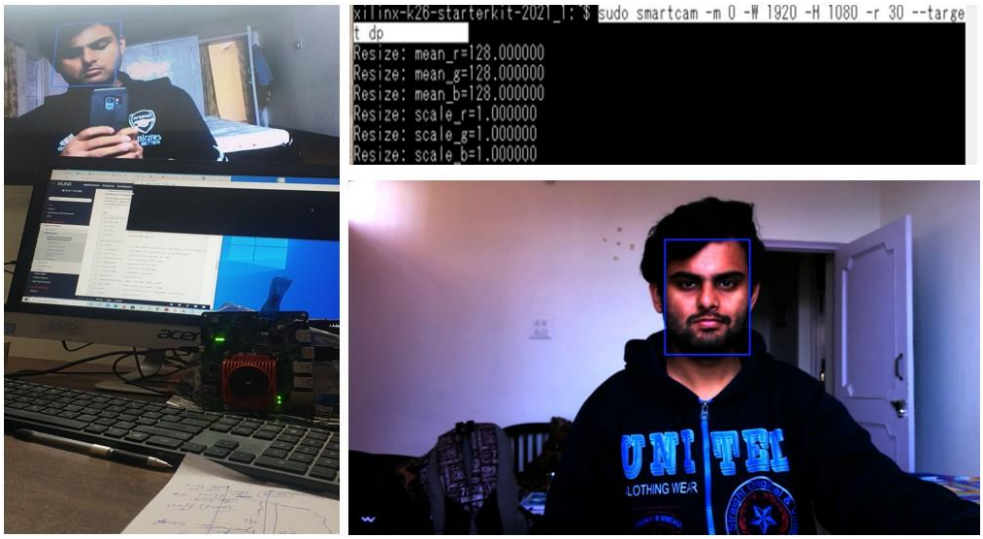


Fig 4.5.5c: Accelerator output after being loaded with face Recognition algorithm

4.5.6 Pixabay

All the video files were taken from Pixabay which is a commercial license free video hosting platform and thus the video samples can be used free of charge for any purpose including commercial purposes. 12 Samples were taken from of 1920p by 1080p were taken from Pixabay for testing of various hardware in this project.

4.5.7 Ubuntu LTS

A Ubuntu is a Debian-based Linux distribution that uses largely free and open-source software.

For Internet of Things devices and robots, Ubuntu is available in three editions: Desktop, Server, and Core. All versions can be installed on a single computer or in a virtual machine. Because Debian was (and still is) open source, Shuttleworth used it as the foundation for his operating system, which he dubbed Ubuntu.

Ubuntu translates to "humanity toward others" and "I am who I am because of who we all are."

We have already installed Ubuntu 20.02 LTS in our VM machine which is used in this project.

4.5.8 ffmpeg

FFmpeg is a free and open-source software project that consists of a collection of libraries and tools for dealing with video, audio, and other types of multimedia files and streams.

The command-line ffmpeg utility, which is used to process video and audio files, lies at the heart of this project as we use it to convert mp4 file into .h264 files which are supported by KriaKv 260 FPGA

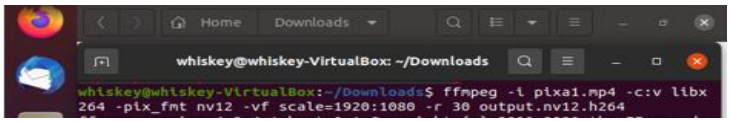


Fig 4.5.8a: ffmpeg command for converting .mp4 file to .h264 file

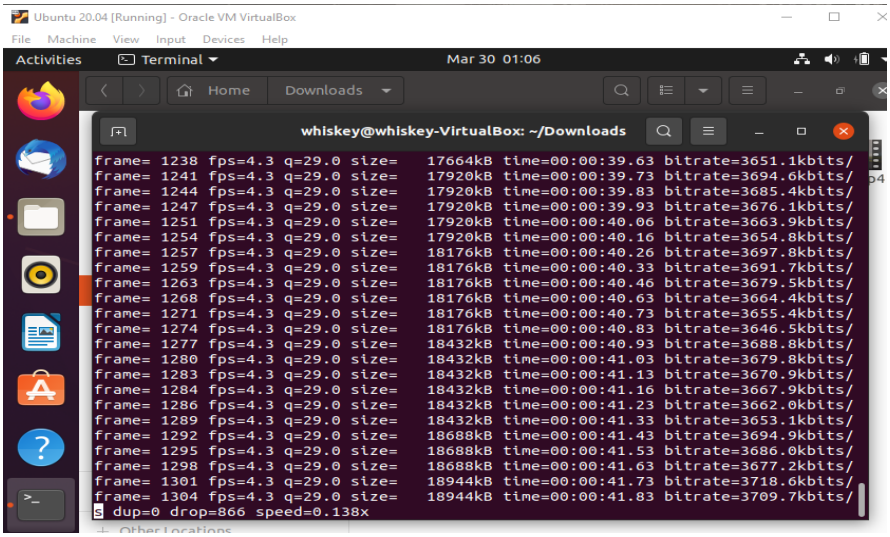


Fig 4.5.8b: mp4 frames being converted to .h264 frames using ffmpeg

4.5.9 .H264/.H265 video encoding

Advanced Video Coding (AVC), also referred to as H.264 or MPEG-4 Part 10, is a video compression standard based on block-oriented, motion-compensated coding. .h264 is much more

powerful algorithm for encoding compared to MPEG-4 aka mp4 Encoding in h.264 is 1.5 to 2 times more efficient than mp4. Since the h264 has much higher data bitrate compared to the same file size mp4 encoded video file, h264 is more computationally expensive. Such encoding methods are much more used in cameras and surveillance cameras much more suited for embedded applications. Since

Kria only takes .h264/.h265 input thus in this project .h264 files were used for test Kria Inference Performance.

4.5.10 WinSCP

AWinSCP is a free and open-source client for Microsoft Windows that supports SSH File Transfer Protocol, File Transfer Protocol, WebDAV, Amazon S3, and secure copy protocol. Its primary purpose is to transmit files securely between a local computer and a distant server.

We use WinSCP to transfer files between Kria board and Windows particularly input and output test files. Follow the following steps in WinSCP for Kria files transfer:

1) Find IP of your Kria board using: ifconfig command in Tera Term

2) This IP is used to connect to the Kria KV 260 for easy file transfer & the password is the password of the Kria board that was set during initialization.

3) Root file is the output file from the board

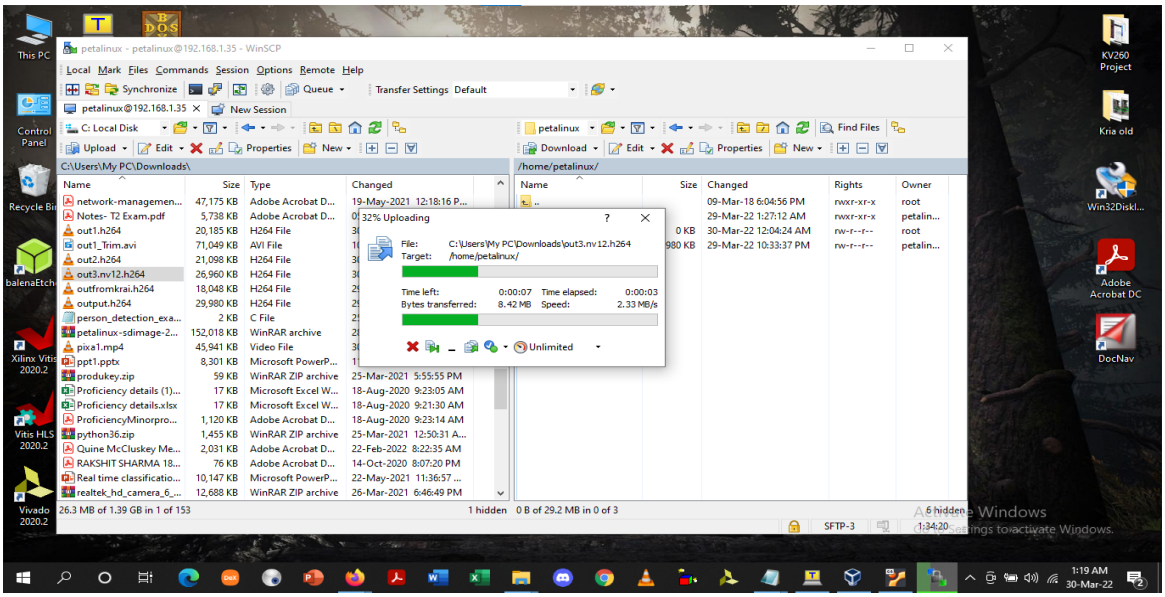


Fig 4.5.10: WinSCP transferring files between Petalinux (Kria SOM) and windows system

4.5.11 Docker

Docker is a free and open platform for building, delivering, and operating apps. Docker allows to decouple apps from infrastructure, allowing to swiftly release software. Since it is a prerequisite for installation of Vitis AI we first need to install docker into our Ubuntu system by using the official guide provide by docker given in reference. Use following command for docker installation:

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin

Fig 4.5.11a: Installing docker in Ubuntu system

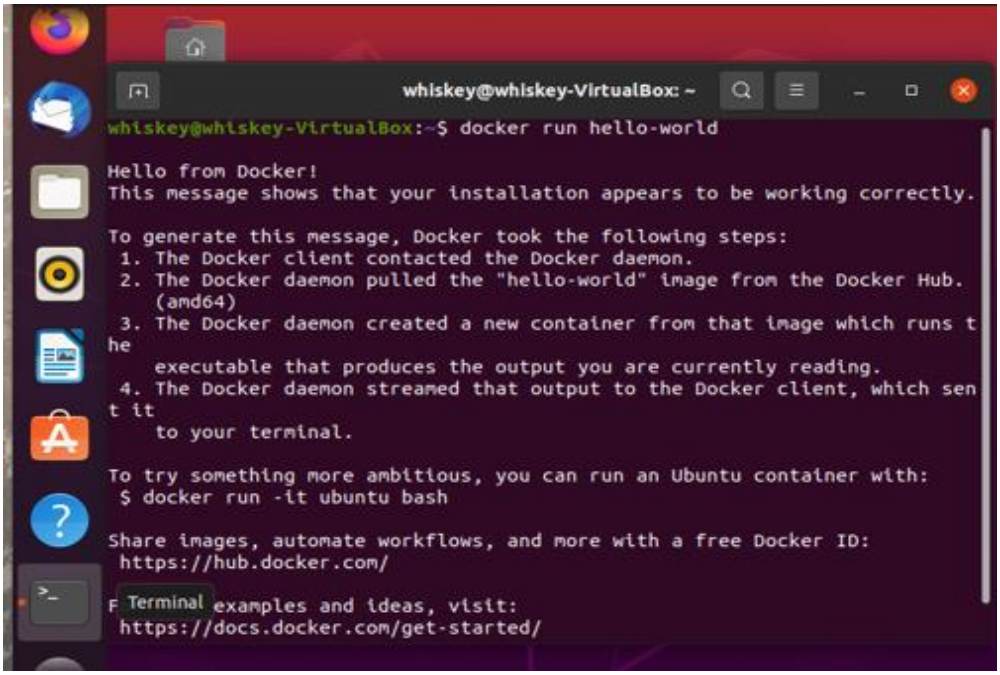


Fig 4.5.11b: Verifying the docker installation

4.5.12 Viti AI

The Vitis AI development environment is a customized platform for AI inference acceleration.

The Vitis AI development environment includes APIs to prune, quantize, optimize, and compile pre-trained networks for AI inference, as well as deep learning frameworks like Tensor flow and Caffe. Official Xilinx github repository can be checked in more detail mentioned in Reference number . This official Xilinx guide was followed by us to install Vitis AI in Ubuntu LTS 20.04 in VM Machine. Following the official guide the downloaded dpu image image would be named xilinx-kv260-dpu-v2021.2-v2.0.0.img and would be around 2.5 GB.

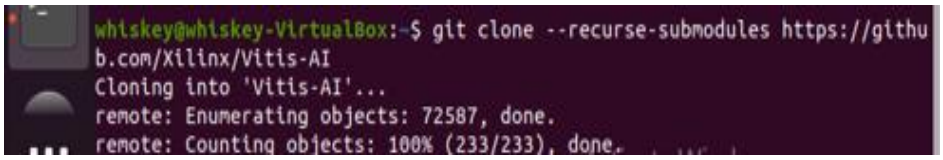


Fig 4.5.12a: Cloning Vitis AI from Xilinx official github Vitis AI link

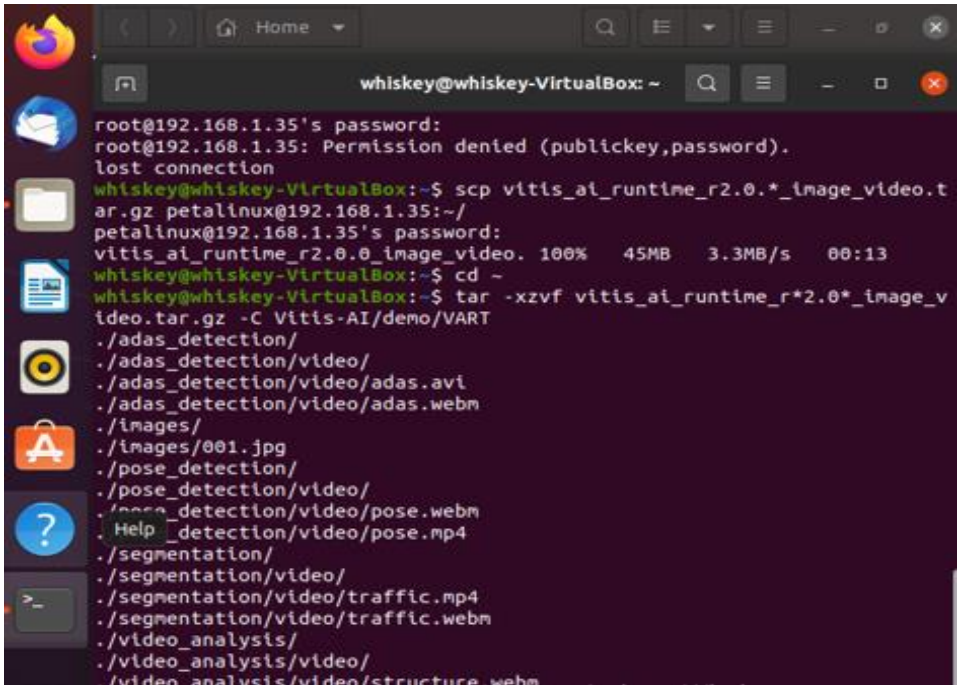


Fig 4.5.12b: Installing of Vitis AI image and running demo samples by Xilinx

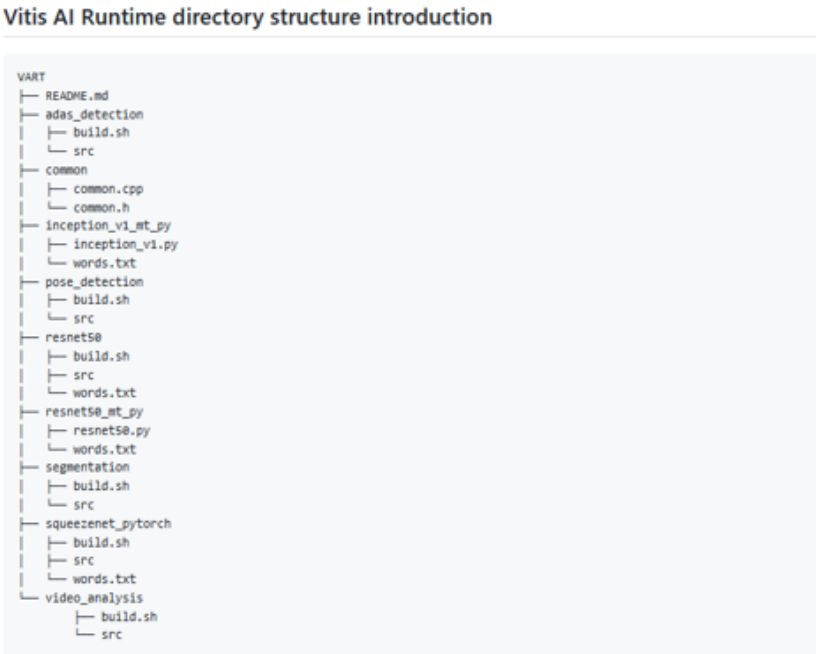


Fig 4.5.12c: Vitis AI Runtime directory structure from Xilinx official github page

4.5.13 Jupyter Installation

For running Jupyter file installation we need to run it on chrome based web browser. Following command gives us the list of already running jupyter servers :

Jupyter-server list

Now for authentication of Jupyter server in Tera Term we first need to find the IP address to the board for this purpose type the following command in Tera Term :

ifconfig

The ip address will be in the format 192.xxx.x.xx. Here our IP address is 192.168.1.40. This IP address is written in Tera Term using the following command:

python3 /usr/bin/jupyter-lab –nobrowser –notebook- dir=/home/petalinux/notebook –ip=192.xxx.x.xx &

This will authenticate the Kria SOM with the Jupyter notebook and similarly Jupyter code can be implemented.

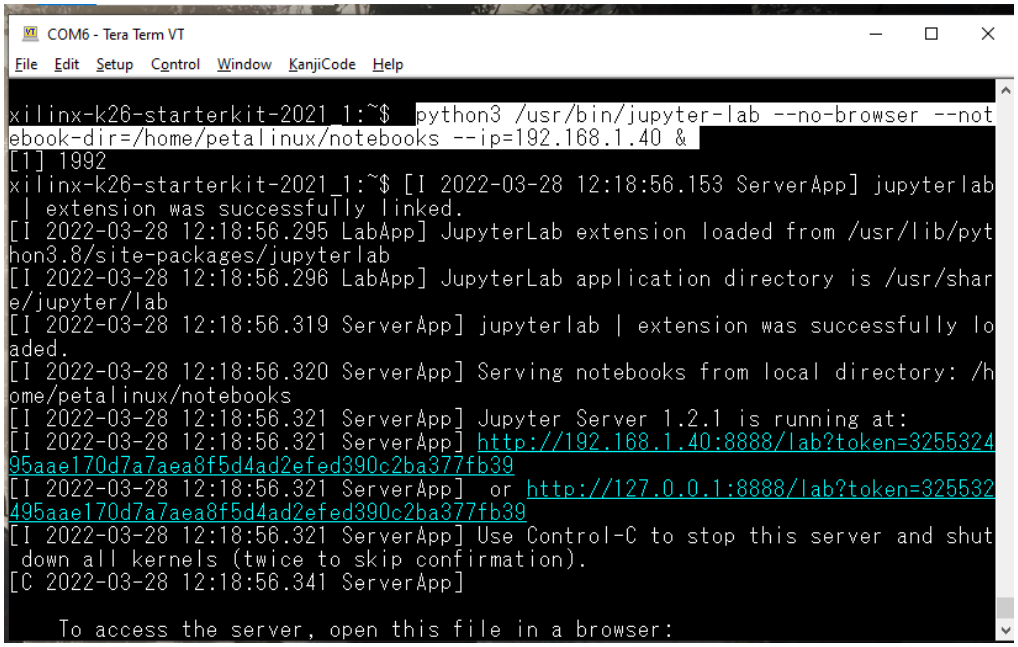


Fig 4.5.13a: Authenticating the Kria SOM to Jupyter notebook in Tera Term

Now, the link can be copied to any chrome based web browser to implement Jupyter notebook for python code.

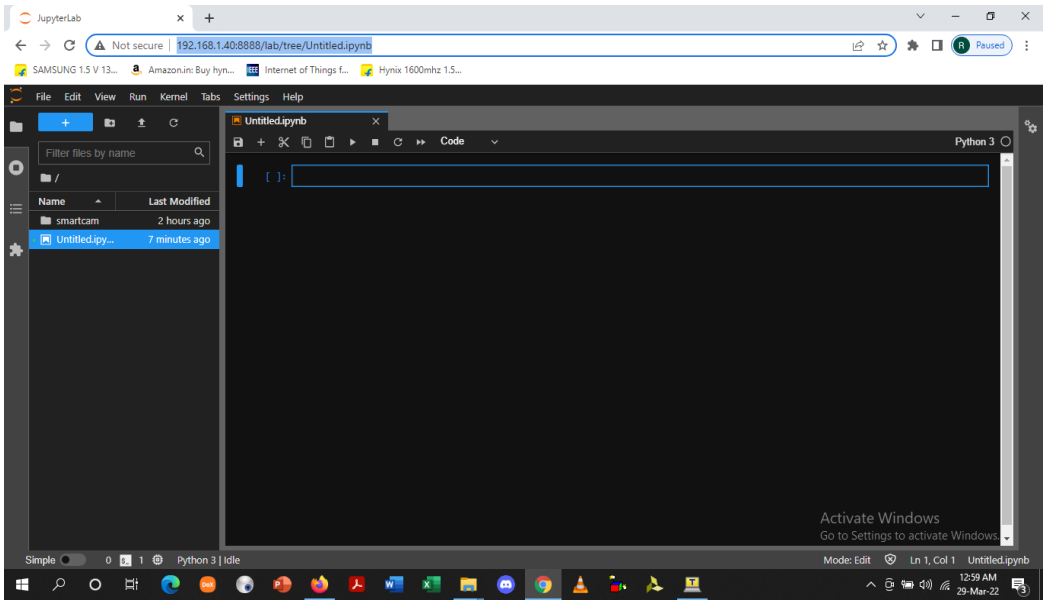


Fig 4.5.13b: Jupyter notebook being implemented in Google chrome

4.5.14 Platform Stats

For Platform statistics Xilinx Xmutil tools can be utilized by using the following command in Tera Term under a root user :

`# xmutil platformstats –p

-p gives platform statistics on power draw

-p can be replaced by –a to get all the statistics of the Kria SOM

This platform can be used to check the system power draw, VCC core voltages, package temperature as well as other useful parameters. These system statistics for power draw from these tools was used in this project later on.

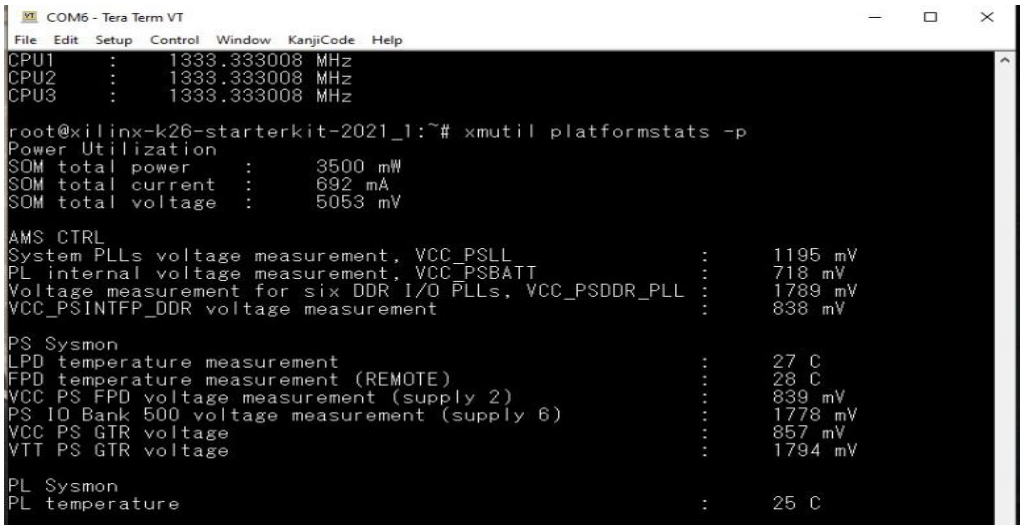


Fig 4.5.14: Xilinx xmutil tool being implemented for system power draw and other statistics in Tera Term

4.6 Sample Selection and categories

A big problem holding Machine Learning algorithms from being implemented in real world scenarios is the lack of accuracy in real world scenarios in some particularly challenging cases. Even in the most cutting edge research happening in the field of Machine Learning particularly in ADAS (Advance Driver Assistance System) cars or FSD (Full Self Driving) cars required a human to be constantly in driving seat due to the lack of accuracy on such machine learning algorithms in untrained test cases or in other world unseen scenarios as was evident in several crashes of Tesla's self driving cars and Uber autonomous car, which actually caused human lives. Although a Machine Learning model on average will always be better than Human driver in terms of accidents and fatalities caused because in general Humans are terrible driver, the ethical issue involved in case of an accident caused by self driving cars make it necessary for us to make sure that such machine learning solutions perform much better in unseen scenarios.

Such an approach requires the system to be tested thoroughly against most of unseen scenarios that can occur in real life. Though it is impossible to train a ML model for all unforeseen scenarios, a variety of general test cases can be taken to test for most of these unseen scenarios.

Considering such an approach we tested the system across 12 different samples. Each sample is a bit different than other representing a different real world scene. The samples were further categorized into 3 categories depending upon the complexity of the scene with each category having exactly 4 outputs. This will help us analyze the performance of different models in different magnitude of complexity. These categories are further explained in section 4.6.1. 4.6.2 and 4.6.3. Each category involves certain variation in scene to better utilize the limited time and resources available to us and to get the most out of these limited test cases. Since the goal of this project was to analyze hardware acceleration of machine learning on FPGA, a variety of test scenarios will result in much more accurate representation of its performance in real world scenarios.

4.6.1 Category 1 output

- Describes a Computationally expensive object detection scenario.
- More than or equal to 12 object detections per Frame.
- Represents someway complex scene such as traffic road. Cases include both high volume high density scenes.
- Sample 9 to Sample 12 represent such case.

Category 1 (n>=12)

Sample 1 - Typical traffic junction of a city. It includes large number of pedestrian making this scene particularly challenging for Machine Learning Models due to large number of classes involved.

Sample 2 – Scene of an Indian city Junction. This scene represents a typical Indian scenario with high density high volume traffic with large number of object classes, particularly challenging due to high complexity. Better performance here will represent a much more suitable platform for machine learning for places like India.

Sample 3 –Represents a high volume traffic scene such as observed in traffic junctions or toll plaza. This scenario is particularly challenging due to the sheer volume of object being detected in each frame.

Sample 4 –Represents high volume high speed traffic. This scenario is particularly complex due to

high variability as object classes change rapidly with little frames.

4.6.2 Category 2 output

- Describes a Computationally moderate object detection scenario.
- More than 6 but less than 12 object detections per Frame.
- Represents someway moderate scene such as a normal moderate traffic road. Cases include either high volume or high density scenes.
- Sample 5 to Sample 8 represent such case.

Category 2 (6<n<12)

Sample 5 - Fast moving, down highway. View from bottom. Represent fast moving object classes
Sample 6 –A 6 lane highway. Represent high volume moderate speed Scenario.
Sample 7 – Indian scenario with large object classes low volume scene.
Sample 8 – Represents a roundabout.

4.6.3 Category 3 output

- Describes a Computationally inexpensive object detection scenario.
- Less than or equal to 6 object detections per Frame.
- Represents someway simple scene such as somewhat empty traffic road.
- Sample 9 to Sample 12 represent such case.

Category 3 (n=<6)

Sample 9 – Low Light (evening/morning) scene. This scene was particularly selected to test low light performance of different hardware.
Sample 10 – Defocus scenario. This scene represents a foggy weather or a camera lens fog or rain.
Sample 11 – Overwatch. Represents a bridge mounted camera system. View from top.
Sample 12 -Moving frame (car). Represent a moving camera mounted in a car. Such scenario may represent ADAS or FSD system.

4.7 Sample Output

4.7.1 Category 1 sample outputs

Sample 1

Fig 4.7.1a : 4 screen Grid of output of different hardware inference of sample 1. Top left screen is the original video. Top right screen is output of CPU inference. Bottom left is output of GPU inference. Bottom right is the output of FPGA inference.

Sample 2

Fig 4.7.1b : 4 screen Grid of output of different hardware inference of sample 2. Top left screen is the original video. Top right screen is output of CPU inference. Bottom left is output of GPU inference. Bottom right is the output of FPGA inference.

Sample 3

Fig 4.7.1c : 4 screen Grid of output of different hardware inference of sample 3. Top left screen is the original video. Top right screen is output of CPU inference. Bottom left is output of GPU inference. Bottom right is the output of FPGA inference.

Sample 4

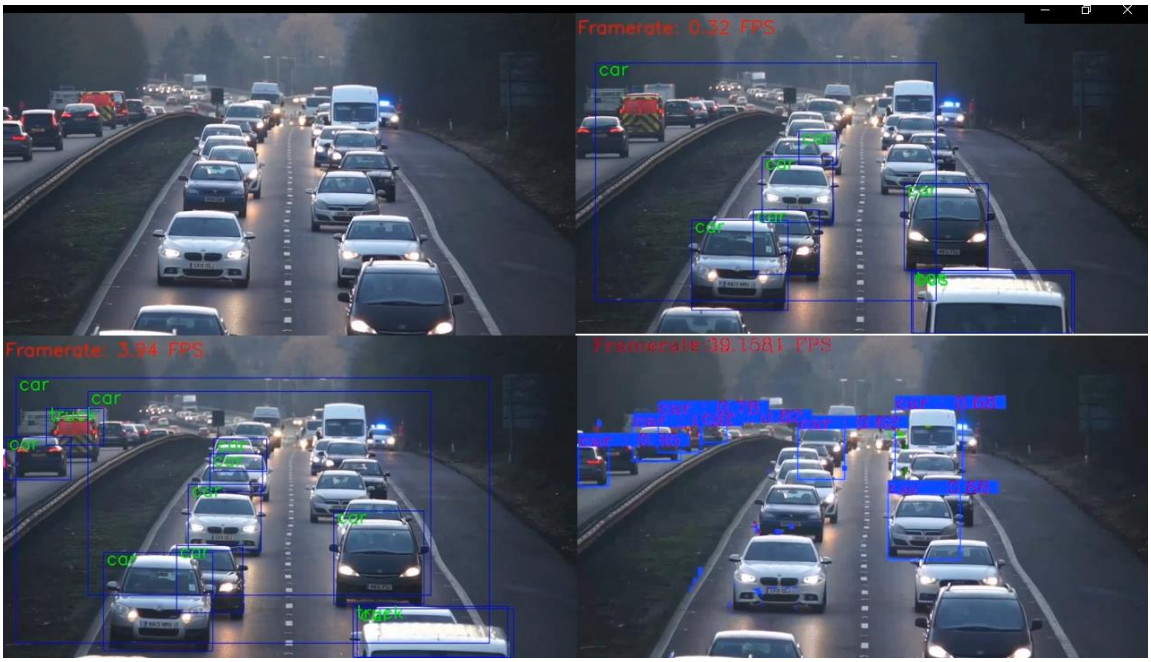


Fig 4.7.1d : 4 screen Grid of output of different hardware inference of sample 4. Top left screen is the original video. Top right screen is output of CPU inference. Bottom left is output of GPU inference. Bottom right is the output of FPGA inference.

4.7.2 Category 2 samples outputs

Sample 5

Fig 4.7.2a : 4 screen Grid of output of different hardware inference of sample 5. Top left screen is the original video. Top right screen is output of CPU inference. Bottom left is output of GPU inference. Bottom right is the output of FPGA inference.

Sample 6

Fig 4.7.2b : 4 screen Grid of output of different hardware inference of sample 6. Top left screen is the original video. Top right screen is output of CPU inference. Bottom left is output of GPU inference. Bottom right is the output of FPGA inference.

Sample 7

Fig 4.7.2c : 4 screen Grid of output of different hardware inference of sample 7. Top left screen is the original video. Top right screen is output of CPU inference. Bottom left is output of GPU inference. Bottom right is the output of FPGA inference.

Sample 8

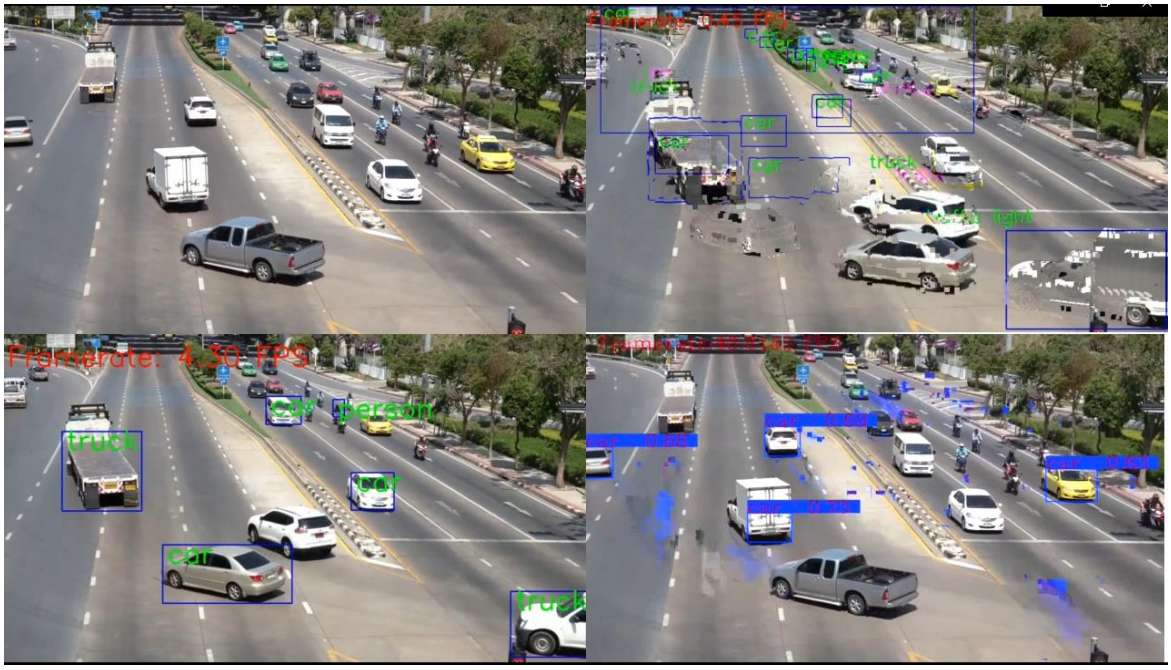


Fig 4.7.2d : 4 screen Grid of output of different hardware inference of sample 8. Top left screen is the original video. Top right screen is output of CPU inference. Bottom left is output of GPU inference. Bottom right is the output of FPGA inference.

4.7.3 Category 3 sample outputs

Sample 9

Fig 4.7.3a : 4 screen Grid of output of different hardware inference of sample 9. Top left screen is the original video. Top right screen is output of CPU inference. Bottom left is output of GPU inference. Bottom right is the output of FPGA inference.

Sample 10

Fig 4.7.3b : 4 screen Grid of output of different hardware inference of sample 10. Top left screen is the original video. Top right screen is output of CPU inference. Bottom left is output of GPU inference. Bottom right is the output of FPGA inference.

Sample 11

Fig 4.7.3c : 4 screen Grid of output of different hardware inference of sample 11. Top left screen is the original video. Top right screen is output of CPU inference. Bottom left is output of GPU inference. Bottom right is the output of FPGA inference.

Sample 12

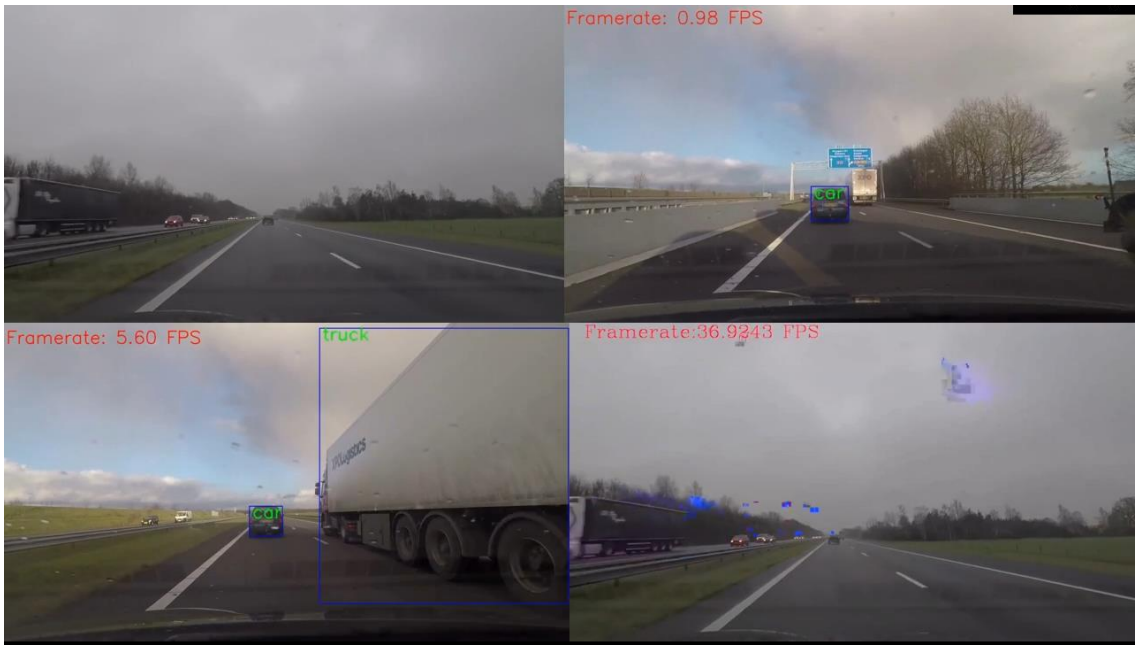


Fig 4.7.3d : 4 screen Grid of output of different hardware inference of sample 12. Top left screen is the original video. Top right screen is output of CPU inference. Bottom left is output of GPU inference. Bottom right is the output of FPGA inference.

4.8 Output chart and Calculations

4.8.1 Output table

The observed performance of 3 Hardware implementations were marked against all 12 video sample marking total 36 data points. These samples were further marked against the 3 categories mentioned in section 4.6.1, 4.6.2 and section 4.6.3 for category 1, category 2 and category 3 respectively. The category represents scene complexity and fps represents frames per second which implies the number of frames the hardware was able to render each second. All input video samples are 1920p*1080p. So, each frame has around 2 million pixels which is lot of data to be processed. For GPU rendering the input files were avi input whereas the FPGA uses .h264 format (since Kria-KV260 is not compatible with avi/mp4 input) mentioned in 4.5.9 which is raw data format thus representing larger bitrates to be processed.

The CPU utilized for CPU plot is Intel i3 3240 – a dual core quad thread processor with max TDP of 55Watts clocked at 3.3 GHz

The GPU utilized in this project is Nvidia Tesla K80 (Cloud Platform- Google colab)

The FPGA utilized is Xilinx Kria KV260 SOM

All 12 video samples were tested against these 3 hardware and the results were plotted in the table 4.8.1a and 4.8.1b.

Table 4.8.1a: Raw hardware performance of category wise distributed all 12 video samples. Fps represents frames per second of the rendered video.

		Hardware	Hardware Acceleration Method	
Category	Sample Number	CPU	GPU	FPGA (Kria KV260)
Category 1 (n<=6)	Sample 1	0.12 fps	2.53 fps	36.71 fps
	Sample 2	0.23 fps	2.59 fps	36.31 fps
	Sample 3	0.29 fps	4.20 fps	40.77 fps
	Sample 4	0.32 fps	3.07 fps	36.51 fps
Category 2 (6<n<12)	Sample 5	0.31 fps	3.07 fps	36.15 fps
	Sample 6	0.31 fps	4.63 fps	36.39 fps
	Sample 7	0.20 fps	3.44 fps	38.34 fps
	Sample 8	0.48 fps	4.02 fps	36.63 fps
Category 3 (n>=12)	Sample 9	1.03 fps	4.99 fps	37.90 fps
	Sample 10	1.18 fps	4.91 fps	36.04 fps
	Sample 11	0.75 fps	3.93 fps	37.27 fps
	Sample 12	1.39 fps	5.42 fps	36.75 fps

Table 4.8.1b: Hardware Performance Relative to GPU Inference of category wise distributed all 12 samples.

		Hardware performance relative to GPU Inference		
	Sample	CPU	GPU	FPGA (Kria KV260)
Category 1 (n<=6)	Sample 1	0.05x	1x	14.51x
	Sample 2	0.09x	1x	14.02x
	Sample 3	0.07x	1x	9.71x
	Sample 4	0.10x	1x	11.78x
Category 2 (6<n<12)	Sample 5	0.07x	1x	7.86x
	Sample 6	0.10x	1x	11.62x
	Sample 7	0.06x	1x	11.15x
	Sample 8	0.12x	1x	9.1x
Category 3 (n>=12)	Sample 9	0.21x	1x	7.6x
	Sample 10	0.24x	1x	7.3x
	Sample 11	0.19x	1x	9.48x
	Sample 12	0.26x	1x	6.78x

4.8.2 Output Graphs

The results from table 4.8.1a and 4.8.1b were plotted in the bar graph in Fig 4.8.2a to get a sense of relative performance of these 3 hardware inferences on all 12 samples. Further a line plot was plotted in Fig 4.8.2b to better analyze the performance trend if the three categories are taken into account as line chart better represents the relative difference in performance of different samples.

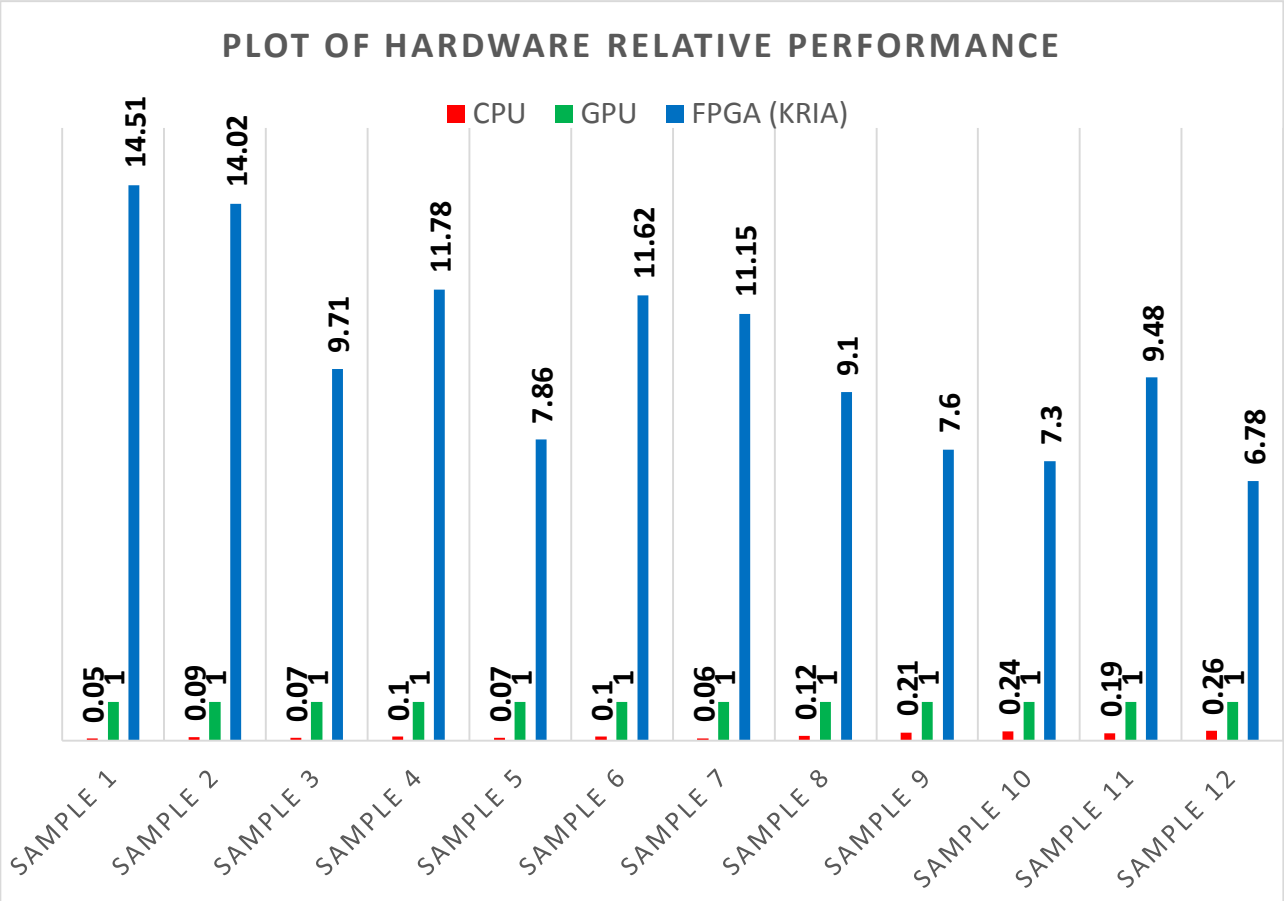


Fig 4.8.2a : Plot of relative hardware performance compared with each other CPU vs GPU vs FPGA for all 12 video samples. (Bar Graph)

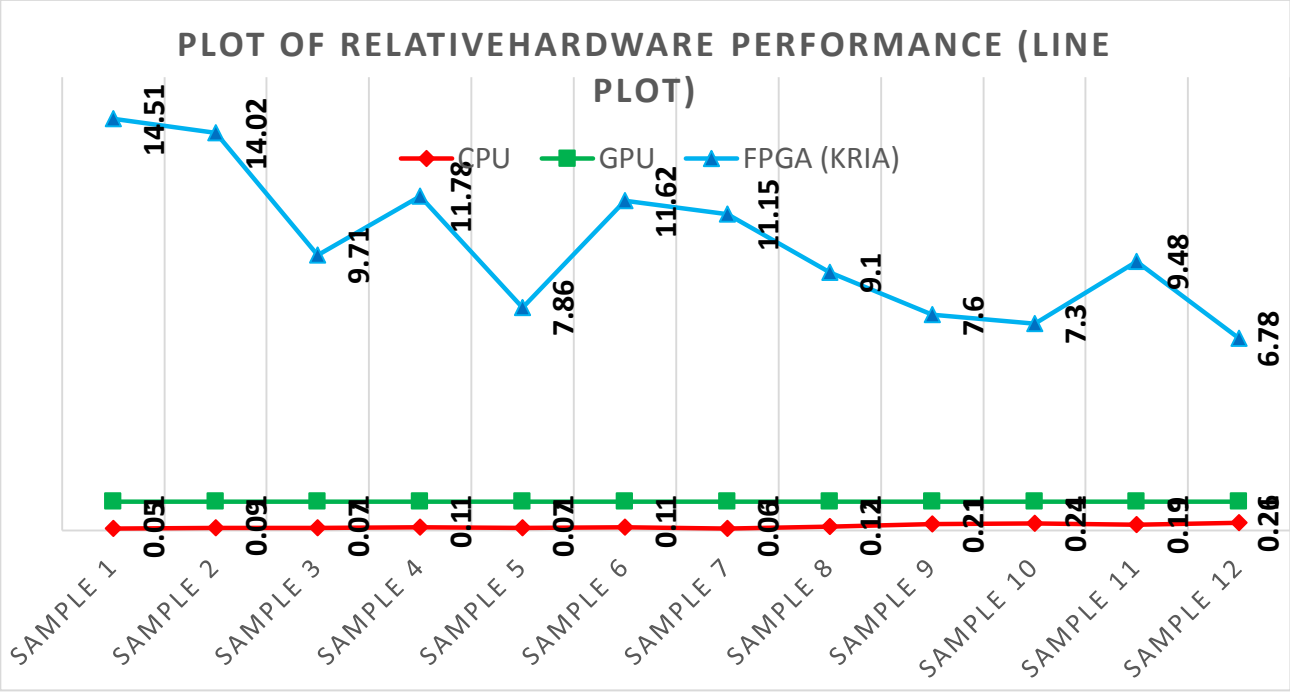


Fig 4.8.2b : Plot of relative hardware performance compared with each other CPU vs GPU vs FPGA for all 12 video samples. (Line Plot)

4.10 Conclusion

1) If we look at relative performance of different hardware inference methods from table 4.8.1b we can clearly see that on average the FPGA inference performance is 10.1 times faster than GPU inference. This performance varies from the 6.7 times to 14.5 times in the samples we tested.

2) If we look at category wise difference in performance between GPU inference and CPU inference we see:

- For category 1 we observe we observe a 12.6 times performance improvement of FPGA compared to GPU inference
- For category 2 we observe we observe a 9.9 times performance improvement of FPGA compared to GPU inference
- For category 3 we observe we observe a 7.8 times performance improvement of FPGA compared to GPU inference

1) Both the Inference ended up being significantly faster than CPU inference as Vision Kernels Perform very well in parallel computation. GPU perform computations parallel as they have several thousand cores for easy parallel computation as well as FPGA which by design can be configured to Perform parallel computation highly efficiently as per the algorithm being implemented. Since, CPU perform computations serially, they end up performing poorly. So, clearly there is a need for a hardware accelerated solution for running these computationally expensive Machine Learning Algorithms for real world scenarios which involve high density high volume object detections (without lowering input resolution of the camera and losing valuable frame data in the process). Also, the accuracy of the model is seen to increase in FPGA hardware acceleration as more computational resources are available for object localization and object classification.
2) From 2 point as well as from line plot in Fig 4.8.2b, if we look at category wise difference in performance between different hardware acceleration we can clearly see a tread as we move towards the right side of graph there is a decrease in the separation of lines between the hardware implementations. As we know from category segmentation moving right side of the chart i.e. moving from category 1 towards category 3 there is a decrease in scene complexity (see section 4.6). This suggests that the FPGA hardware is much more effective in complex object detection scenarios involving high volume high density objects in each frame. Thus, for more complex practical scenarios involving large number of objects with high density, this report concludes that FPGA hardware acceleration performance much better compared to GPU hardware acceleration for model Machine Learning inference.
3) From Point 4 and sample 2 & sample 7 both of which include Indian roads involving large

variety of classes with large volume object detection with high density object detections in each frame particularly in sample 2 additionally also having high speed traffic as the original input video was sped up (particularly challenging test case). Since, sample 7 ended up observing the largest difference in relative performance in its category it is evident for such challenging practical scenarios such as India the FPGA accelerated of Machine Learning algorithms is suited much more than GPU accelerated inference.

4) The Kria KV260 FPGA Works with either .h264/.h265 while GPU works with mp4/avi input. Since Kria was tested on .h264 files and .h64 files have a much higher bitrates as explained in section .FPGA is much more suitable for scenarios which might require higher resolution camera input such as for ADAS (Advanced Driver Assistance System) or FSD (Full Self Driving) systems. These self driving solutions which are being invested upon in millions by large AI companies such as Google's Waymo, Uber's Aurora and Tesla require high resolution images fed from HD camera onboard to resolve objects which are far away. Thus, FPGA is much more suited for such task as it was tested on higher bit rate video input.
5) Although the Inference performance of Machine Learning is much better in FPGA compared to GPU the actual training of these Machine learning Models has not been tested by us in this report since we used an already pre-trained machine learning model explained in Section. Although the reference paper we took for this project (Reference number [2]) suggests that Model training may end us performing better (in at least some scenarios) in GPU as compared to FPGA due to availability of high bandwidth large amount of memory, this cannot be independently verified yet by us as this report only looks at Inference performance. Further research is needed in this aspect but if GPU indeed ended up performing better a possible solution may be to train the Machine learning model first on GPU and then for inference FPGAs can be implemented.
6) Power Draw - Looking at the efficiency of the hardware acceleration solution From section 4.3.3 and section 4.4.2 section 4.5.14 and section we can see a power drown of 19.8 Watts, 34 Watts and 15 Watts for CPU inference, GPU Inference and FPGA inference respectively. Since the raw performance FPGA is already better than GPU and CPU, The lower power draw indicates that FPGA is the most efficient among them. Thus, for embedded solutions or computing in the edge or for implementation in real world such as traffic light management (embedded FPGA in traffic light etc.) etc. lower power draw of FPGA means that it can be powered by solar panels for a completed embedded solution father decreasing the hardware cost in such a project.
7) EDP (Energy Delay Product) - Run-time or energy per frame alone do not provide a complete

picture. A hardware platform can be incredibly energy efficient while yet being much too sluggish to be useful. The Energy Delay Product (EDP) statistic considers the algorithm's throughput in milliseconds per frame (ms/frame) as well as the energy spent per frame (mJ/frame). The product of energy/frame and delay time is the EDP. This allows for a fair comparison when selecting which hardware architecture is best for a given calculation. Lower EDP indicates that the hardware design is capable of doing specified compute tasks with less power and in less time.

EDP = energy/Frame * Delay time

[Energy/Frame = Energy consumed (mJ) / Frame rate output

Delay time = (1000/fps achieved) mS

Fps= frames per second]

Since both the parameters are already lower in FPGA compared to GPU. We can without calculation conclude that EDP is much lower in FPGA compared to GPU. Thus, FPGA outperforms GPU in this parameter.

8) In final Wrapping of conclusion this report concludes that for hardware acceleration of Machine Learning Inference FPGA outperform GPU for SSD Models. Additionally, this performance difference increases with increase in complexity of scenario. Thus, for complex vision kernels FPGA hardware accelerated inference must be considered compared to other solutions

APPENDIX

Software Code

```
import cv2

cv2.__version__

import matplotlib.pyplot as plt

cd "C:\Users\My PC\Desktop\Major Project"

config_file =  'ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt'
frozen_model = 'frozen_inference_graph.pb'

model = cv2.dnn_DetectionModel(frozen_model,config_file)

classLabels=[]
file_name='labels.txt'
with open(file_name,'rt')as fpt:
    classLabels = fpt.read().rstrip('\n').split('\n')

print(classLabels)

print(len(classLabels))n

model.setInputSize(320,320)
model.setInputScale(1.0/127.5)
model.setInputMean((127.5,127.5,127.5))
model.setInputSwapRB(True)

img = cv2.imread('car meet.jpg')  ##read an image

plt.imshow(img)

plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))



classIndex, confidece, bbox = model.detect(img,confThreshold=0.5)
print(classIndex)
```

```
  font_scale = 3
  font = cv2.FONT_HERSHEY_PLAIN
  for ClassInd,conf,boxes in zip(classIndex.flatten(),confidence.flatten(), bbox):
     cv2.rectangle(img,boxes,(225,0,0),2)
     cv2.putText(img,classLabels[ClassInd-1],(boxes[0]+10,boxes[1]+40), font,
  fontScale=font_scale, color=(0,255,0),thickness=3)
  plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))



# Video Demo
import numpy as np
import cv2
import time
cap = cv2.VideoCapture("nycoutavi.avi")
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('outtest17.avi', fourcc, 30.0, (1920, 1080))
font_scale = 2
font =  cv2.FONT_HERSHEY_PLAIN
prev_frame_time = 0  # FPS
new_frame_time = 0
while(True):
    ret, frame = cap.read()
    if not ret:
        break
    font = cv2.FONT_HERSHEY_SIMPLEX
    new_frame_time = time.time()
    fps = 1/(new_frame_time-prev_frame_time)      # Calculating the fps
    prev_frame_time = new_frame_time
    fps_text= "Framerate: {:.2f} FPS".format(fps)
    ClassIndex, confidence, bbox = model.detect(frame,confThreshold=0.55)
    print(ClassIndex)
```

```
    if (len(ClassIndex)!=0):
        for ClassInd, conf, boxes in zip(ClassIndex.flatten(), confidence.flatten(), bbox):
            if (ClassInd<=80):
                cv2.rectangle(frame,boxes,(225,0,0),2)
                frame = cv2.resize(frame, (1920, 1080))
                hsv=cv2.putText(frame,classLabels[ClassInd-1],(boxes[0]+10,boxes[1]+40),          font,
fontScale=font_scale, color=(0,255,0),thickness=3)
                hsv=cv2.putText(frame, fps_text, (7, 70), font, 2, (0, 0, 255), 2, cv2.LINE_AA)
                hsv = cv2.resize(hsv, (1920, 1080))
                cv2.imshow('frame',frame)
    out.write(hsv)
    cv2.imshow('Original', frame)
    cv2.imshow('frame', hsv)
    if cv2.waitKey(1) & 0xFF == ord('a'):
        break
cap.release()
out.release()
cv2.destroyAllWindows()
```

REFERENCES

[1] TomTom.com, 'Tom Tom World Traffic Index', 2019. [Online]. Available: https://www.tomtom.com/en_gb/traffic-index/ranking/

[2] Murad Qasaimeh*, Kristof Denolfy, Jack Loy, Kees Vissersy, Joseph Zambreno*, and Phillip H. Jones*, "Comparing Energy Efficiency of CPU, GPU and FPGA Implementations for Vision Kernels", *Iowa State University, IA, USA, yXilinx Research Labs, CA, USA. (Qasaimeh, et al., 2019)

[3] J. Redmon, 'Darknet: Open Source Neural Networks in C', 2016. [Online]. Available: https://pjreddie.com/darknet/

[4] https://www.xilinx.com/products/som/kria/kv260-vision-starter-kit/kv260-getting-started/getting-started.html

[5] https://xilinx.github.io/kria-apps-docs/main/build/html/docs/build_petalinux.html https://docs.xilinx.com/r/2021.1-English/ug1144-petalinux-tools-reference-guide/Revision-History

[6] https://github.com/Xilinx/Vitis-AI/blob/master/setup/mpsoc/VART/README.md

[7] https://docs.docker.com/engine/install/ubuntu/

[8] https://ffmpeg.org/documentation.html

[9] https://ttssh2.osdn.jp/index.html.en