

Chapter 1

Introduction

Machine learning is in existence since the mid of 1900s with the initial definitions given by Arthur Lee Samuel in 1959. As per his definition machine learning is the “field of concentration that enables computers to learn without being parallel programmed” [17]. The definition remains constant today as well as implies the point of any advancement done in this area. Artificial Intelligence turned into a fascinating territory of research with regards to late 20th century. However, it was not taken up by numerous sorts of research as the computational prerequisites for created calculations were excessively high. Be that as it may, as computers turned out to be increasingly powerful, the enthusiasm of specialists developed again around there. Alongside the development of computational power, many artificial intelligence calculations have likewise been redesigned to make the memory and power production.

The basic and most important part of artificial intelligence is image processing and detection. Object detection and tracking is an important challenging task within the area in Computer Vision that try to detect, recognize and track objects over a stream of images called video. It helps to understand and describe object behavior instead of monitoring the computer by human operators. It aims to locate moving objects in a video file or surveillance camera. Object tracking is the process of locating an object or multiple objects using a single camera, multiple cameras or a given video file. With the development of high quality imaging sensors, the quality and resolution of the image are improved, as well as the exponential increment in computation required to object detection and tracking applications and processing algorithms. In Object Detection and Tracking I have to detect a target object and track that object in consecutive frames of a video file. The analysis of images and videos can provide important information to applications. Object detection allows to identify regions of interest (stationary or in movement), Object tracking follows the object movement and generates trajectories, Object classification allows to distinguish the different identified objects (human, car, truck, motorcycle, license plate, etc.), and activity recognition will analyze the activities and behaviors performed by individuals or group of objects.

Object detection is a fundamental task in computer vision. Its goal is to identify the position of different objects in an image. This position can be indicated by a bounding rectangle or by a single dots mask that represents the pixels corresponding to the object. The

difficulty lies in dealing with multiple possible appearances, ignoring irrelevant information, such as noise in the image, which can affect the results or execution time. Object detection often relies on or includes object classification and object recognition. Object classification aims to determine the correct class of an input object. The classification problem can be binary and in particular one can look for the presence or the absence of a specific object class (e.g. faces, people, etc.). If this problem is solved, a sliding window approach that checks every position of the identified object can transform the classifier into a detector. This observation lies at the heart of most modern detection approaches which are essentially fast classifiers. In object recognition, specific objects are often recognized often from images containing objects of the same class. Besides the obvious advantage of needing less time to check each position, it also involves a coarse level filtering of negative examples and has a benefit on classification. For practical systems such a module is essential to ensure fast execution and robust operation. Object detection can be placed in the context of computer vision and it is related to other domains such as: image processing, pattern recognition, machine learning, robotics and artificial intelligence. In image processing important elements are: interest point detection, corner detection, feature descriptors and segmentation. Pattern recognition enables the recognition of higher level features such as: lines, shapes or textures. Machine learning provides knowledge about choosing, training and evaluating powerful classifiers for the task at hand. Artificial intelligence develops classifiers that are based on human physiology such as neural networks.

This thesis will mainly focus on implementation of convolutional neural networks on FPGA with the evolution of various methods and approaches to detect objects using dynamic objects. We will discuss more about this in following chapters which discusses the theory of used approaches.

1.1 Machine Learning (Artificial Intelligence):

Artificial Intelligence has achieved a lot of development in various areas. The greater part of research in these areas has similar points, as depicted in the early paragraph, but utilizes various algorithms to accomplish the goal. Probably the most critical learning algorithms are given below:

1.1.1 Unsupervised Learning:

In this type of leaning ^[11], the training data used as input to this algorithm does not have a specified output. The algorithm's job is to recognize the structure of the data provided as input and then manage this data as per the requirement. These algorithms are not as much

popular as compared to the algorithms using supervised learning methods. In any case, regardless are critical in applications for repetition reduction and applying numerical calculations on information structures. Apriori and K-means are some prevalent calculations utilizing these algorithms.

1.1.2 Supervised Learning:

In this type of algorithms, the machine attempts to gather a function from the training data. The training data is regularly a combination of inputs and related outputs. The algorithm designs itself to produce explicit outputs for relating inputs. After preparing thousands or contingent upon the application, a huge number of preparing trained data tests; it tends to be tried testing on new data for approval. In many situations, the calculations improves with additional training. This algorithm is used to take care of issues identified with order and relapse. A portion of the well-known calculations that utilize this learning method are logistic regression, and artificial neural networks (ANNs). After an impressive research in the field of neural networks ^[11], specialists thought of convolutional neural networks (CNNs). These CNNs have now turned into the base of deep learning. Aside from this two noteworthy learning strategies, there are additionally semi-supervised techniques. That utilizes a combination of two methods.

1.2 Artificial Neural Networks:

ANN ^[15] is a standout amongst the most utilized Artificial Intelligence algorithms today. It is based freely on the structure of neurons and axons that assists various living beings to accomplish complex errands on an everyday bases. By taking in different exercises from childhood, we train the neural systems in our mind to gain proficiency with specific reactions to explicit circumstances. This organic system comprises of around 100 billion neurons for a normal individual. The interconnection of billions of these neurons is in charge of human intelligence. The figure demonstrates a straightforward structure of this organic neuron system.

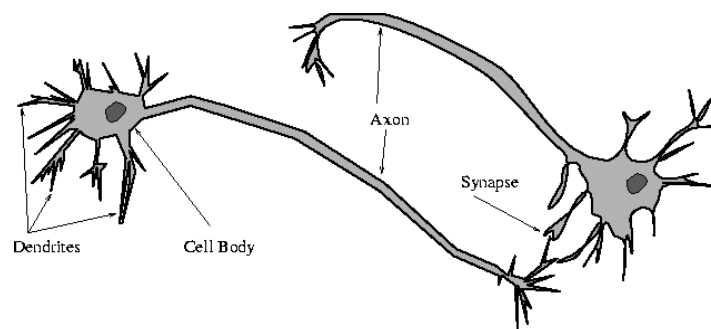


Figure 1.2 (a): Organic Neuron System ^[22]

In figure 1.1, Dendrites and Synapse are the interconnections between different cells and neural pathways. The idea of artificial neural network is same as organic neuron system i.e. to have the interconnection between millions of neurons and to decide in complicated scenarios. The training of these artificial neurons can be done by feeding them the training data and by adjusting values to achieve the required output. The figure below will represent a simple neural network:

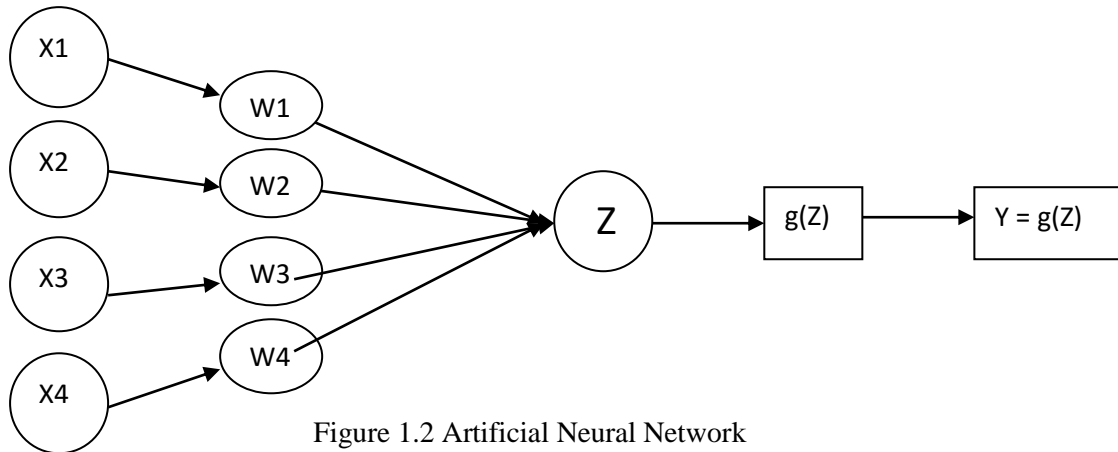


Figure 1.2 Artificial Neural Network

The equation for a simple Neural Network is as follows:

$$Z = \sum_{i=1}^3 W_i X_i - b$$

Where b is a bias correction. Convolutional Neural Network are a special case of ANNs. The main difference between them is the way of connectivity established between various layers of neurons.

1.3 Problem Statement and Aim:

The main purpose of this thesis is to develop and design an FPGA based Neural Network implemented framework for detection of real time dynamic objects in real time environment. As there is availability of large variation in neural networks, the evolution of neural networks will be done based on accuracy and speed of the neural network. Detection of the objects can be defined as the problem as the object moves around the scene. These problems arise due to loss in information, noise in images, complex object shapes, scene illumination changes and real time processing requirements.

The aim of this thesis is to implement a flexible and reconfigurable Neural Network structure on FPGA by creating a custom hardware platform for use in FPGA which will include a neural network implementation.

1.4 Motivation:

The rapid improvement in technology makes video acquisition sensor or devices to benefit from better comparative costs. This is the cause of increasing number of applications in different areas that can be used more efficiently. A digital video is a collection of sequential images with a constant time interval. So there is more information present in the video, objects and background evolves over time. After studying the literature, it is seen that detecting objects in a particular video sequence or any surveillance camera is a really challenging task for a computer vision application. Video processing is really time consuming due to a huge number of data presents in a video sequence. Moreover, video tracking is currently of immense interest due to its implication in video surveillance, security, medical equipment's or robotic systems. Video tracking offers a context for the extraction of significant information such as scene motion, background subtraction, object classification, interaction of object with background and other objects, human identification, behavior of human with objects and background, etc. Therefore it is seen that there is a wide range of research possibilities open in relation to video tracking. However, on the other end there is constant reduction in prototyping speed of neural networks for testing and GPU based frameworks had already achieved this goal at a great extent but there is still requirement in FPGA based frameworks. As in the influence of this motivation, there are two main sources needed to be drawn.

1.4.1 Environment recognition:

Machine learning is growing gradually in its own way in various areas. These areas range from natural language processing to image processing. The aim of this thesis is to develop a neural network framework implemented on FPGA platform which will be used in detection of objects in real time environment. The specified problem which is being focused in this thesis is to recognize different objects in the vehicle utilizing machine learning instead of traditional picture by preparing strategies. The fascinating objects including people on foot, vehicle, and bikes etc. the utilization of environment recognition will not only guides the driving help framework in settling on the correct choices yet can likewise fundamentally reduce accidents brought about by human blunder. Recent research plans to utilize information combination strategies on the information from sources, for example, radar, video and various different sensors to prepare artificial intelligence and machine learning systems with most elevated precision.

1.4.2 FPGA based Machine Learning Framework:

Numerous organizations are utilizing artificial intelligence in the automotive space. The majority of the research is being accomplished by executing the suitable AI algorithms on GPUs. This is nice as an initial step however high power utilization of most of the GPUs make them bothersome for automotive manufacturers. The second step towards its path ought to be to search for a quick and power cordial arrangement. Thus, the proposition introduces the underlying thought of a FPGA based AI system. We plan to make our usage reconfigurable so that an assortment of systems can be actualized by evolving some key parameters of the plan. The theory additionally intends to decisively demonstrate that an FPGA can use its huge parallelization and customization ability to make the usage in a fast manner and less power eager when contrasted with a GPU.

1.5 Structure of thesis:

The thesis is structure in 5 chapters. In chapter 2, we will discuss about the Literature Survey done in order to find recent approaches. In chapter 3, we will discuss some explorations on the basis of related researches that had been carried out in past few years and compares them to our proposed solution. We will cover a theoretical background of different algorithms, libraries and approaches that are being used in this work. Here we will also see why I have selected YOLO as a better approach for implementation in FPGA platform as compared to other available networks. In the same chapter we will also have a short look at the datasets used in this work. Chapter 4 will represent a short description of the hardware used in this work. Alongside the same chapter will also represent the descriptions of the tools used in creation of the FPGA platform and also about the language and web architecture used in this project. Chapter 5 will present the experimental results I have achieved during this work. Chapter 6 will conclude the thesis and discuss some tasks that can be considered for future development. Suitable Bibliography and annexes can be found at the last of the thesis.

Chapter 2

Literature Survey

The research conducted so far for object detection and tracking objects in video surveillance system. Tracking is the process to locating the interested object within a sequence of frames, from its first appearance to its last. The type of object and its description within the system depends on the application. During the time that it is present in the scene it may be occluded by other objects of interest or fixed obstacles within the scene. A tracking system should be able to predict the position of any occluded objects. On the other side, implementing a neural network system specially designed for object detection on FPGA is also a difficult task. In order to imply the optimized C/C++ codes from python code is the primary requirement. The below section represents the state of art and literature survey used to learn various important techniques and implementation.

2.1 State of Art:

The system developed here will have the following features that will make it state of art at a true basis:

- The created structure will almost certainly actualize any CNNs. This is additionally valid for vast systems as the structure has an inherent capacity to execute network by separating them into little parts. This implies an extensive number of info highlights can be given into the system a couple of thousand parameters at an opportunity to actualize them.
- A hardware platform has been created with HDMI functionality as well as neural network implementation by using several Xilinx tools in such a way that it can be used in PYNQ board to detect objects using Jupyter and Python Language as well as it can be modified in case of any requirement.

2.2 Literature Survey:

2.2.1 “Recognition of Indian License Plate Number from live stream Videos”.

Author: Sachin Prabhu B., Subramaniam Kalambur and Dinkar Sitaram

Year of Publication: 2017 ICACCI

Abstract: Many developed countries have Automatic License plate Recognition (ALPR) System implemented for traffic management. ALPR is a surveillance system which extracts the information of vehicles by capturing images. Human intervention might result in delay

and missing out some violations. Hence automating the process using surveillance camera is a challenging task. In this paper they have compared the accuracy of three approaches i.e. OpenALPR, k-NN and CNN.

Conclusion: this paper depicts the analysis that Convolutional Neural Network (CNN) is the most suitable for recognizing license plates.

2.2.2 “Situation-aware Framework from Moving Object Detection in Moving Camera”

Abstract: Kimin Yun, Jongin Lim, and Jim Young Choi.

Year of Publication: 2017 ICPRS.

Abstract: In this article, the authors proposed a situation based detection system to detect objects in dynamic background environments with the use of dynamic camera. There were some problems raised from dynamic background: false positives from inaccurate camera motion estimation, sudden scene changing, and slow motion object relative to camera movement and motion model limitation in a dashcam video.

Conclusion: an incorporated structure handling issues has been brought up in the moving object detection using dynamic cameras. The structure depends on double mode displaying with intentional inspecting and receives scene conditional adjustment of the model.

2.2.3 “Moving object detection by a mounted moving camera”.

Authors: Ozge Mercanoglu Sincan, Vahid Babaei Ajabshir, Hacer Yalim Keles and Sileyman Tosun.

Abstract: Accurate moving object is an important parameter for Surveillance systems Background subtraction works well for static camera but not for dynamic camera as the object also moves. In this article they proposed an optical flow based moving object detection algorithm for the video captured by mounted moving camera.

Conclusion: they discover the interest points in successive video frames and track them with pyramidal Lucas-Kanade strategy. In disposing a camera movement, a frame difference technique is utilized to recognize moving objects. They used versatile thresholding considering distinctive lighting conditions in a similar video and tried to calculate using Golbasi dataset and Hopkins dataset and noticed that the proposed technique identifies moving object with high precision and low false cautions.

2.2.4 “Real-time face detection using Moving camera”.

Authors: Deng-Yuan Huang, Chao-Ho Chen, Tsong-Yi Chen, Jian-He Wu and Chien-Chuan Ko.

Abstract: this article proposed a system consist of three modules: Detection of faces based on skin color, edges, and face area; verification of faces based on HOG (Histogram of Object Gradient) features and two-class C-SVM (Support Vector Machine) classifier; and face tracking. The proposed method can avoid a huge amount of computation time and accuracy can be improved.

Conclusion: test results demonstrated that the proposed strategy can effectively identify a large portion of human countenances, showing the attainability of the proposed technique.

2.2.5 “Object Detection using Deep Neural Networks”.

Authors: Malay Shah and Rupal Kapdi.

Abstract: the problem discussed in this article is depends on object detection using Deep Neural Network especially Convolutional Neural Network (CNN). Object detection was previously performed by using only conventional deep convolutional neural network whereas using regional based convolution network increases the accuracy and also decreases the time required to complete the program. The dataset used is PASCAL VOC 2012.

Conclusion: this articles concludes that the regional convolution network is more optimized at basic level and it is in dispute that RCNN is the best solution to the problem or not the results depicted is valid only in certain parameters. Another researches can engender new parameters and may achieve less error rates.

2.2.6 “On-Road Object Detection using Deep Neural Network”.

Authors: Huieun Kim, Youngwan Lee, Byeounghak Yim, Eunsoo Park and Hakil Kim.

Abstract: the authors proposed a strategy for autonomous vehicle and advanced driver assistance system to solve the issue related to the accidents caused by industrialization of vehicles. They compared the detection accuracy among objects classes and analyzed the recognition results with fine-tuned single shot multibox detector on KITTI dataset.

Conclusion: the author concluded that fine-tuned SSD on road dataset using data augmentation can improve the detection result and to adjust end-to-end detector more practically in vehicles, research on deep compression for reducing memory of the model has to be accompanied.

2.2.7 “Pedestrian detection based on YOLO Network Model”.

Authors: Wendo Lan, Jianwu Dang, Yangping Wang, and Song Wang.

Abstract: in this article the authors had improved the network structure and proposed a new structure YOLO-R which has an addition of three pass through layers (Route layer and Reorg layer) in old network structure. They had also changed the layer number of pass through layer

connection in the original YOLO algorithm from layer 16 to layer 12 to increase the ability of the network to extract the information of the pedestrian.

Conclusion: the improvement of the structure was tested on the INRIA dataset. The results depict that this method is effectively improve the detection accuracy of pedestrians, while reducing the false detection rate and the missed detection rate and the speed of detection reach up to 25fps.

2.2.8 “Real-time face detection based on YOLO”.

Authors: Wang Yang and Zheng Jiachun.

Abstract: The article is based on face detection using YOLO network model. The authors are using this algorithm because the model possess fast detection speed and is suitable for target detection in real-time environment. It has better detection accuracy and faster detection time in comparison to other similar target detection systems.

Conclusion: the article concludes that the face detection method based on YOLO has stronger robustness and faster detection speed. Being in a complex environment, YOLO guarantees the high detection accuracy and can meet the real-time requirements at the same time.

2.2.9 “Deep Neural Network accelerator based on FPGA”.

Authors: Thang Viet Huynh

Year of Publication: 2017, NAFOSTED Conference on Information and Computer Science.

Abstract: In this article, the author proposed an efficient architecture for hardware realization of Deep Neural Network on FPGA. The architecture employs only one single physical computing layer to perform the whole computational fabric of fully-connected feed forward deep neural network with customizable number of layers, number of neurons per layer and number of inputs.

Conclusion: The author concludes that for performance evaluation, the handwritten digit recognition application with MNIST database is performed which reported an excellent amount of recognition rate i.e. 97.20% and speed is 15.81 fps.

2.2.10 “Are coarse grained overlays ready for general purpose Application Acceleration on FPGAs?”

Authors: Abhishek Kumar Jain, Douglas L. Maskell and Suhaib A. Fahmy.

Year of publication: 2016 IEEE.

Abstract: This article propose the discussion about the role of overlay architectures in enabling general purpose FPGA application acceleration.

Conclusion: the authors examined the use of overlays for general purpose on-demand application acceleration and noticed that with an efficient DMA controller, data transfer and the overlay clearly reflects the benefits of an overlay for supporting hardware acceleration of tasks.

Chapter 3

Explorations of Neural Approaches

Machine Learning:

Learning algorithms are widely used in computer vision applications. Before considering image related tasks. Machine learning has emerged as a useful tool for modeling problems that are otherwise difficult to formulate exactly. Classical computer programs are explicitly programmed by hand to perform a task. With machine learning, some portion of the human contribution is replaced by a learning algorithm. As availability of computational capacity and data has increased, machine learning has become more and more practical over the years, to the point of being almost ubiquitous.

3.1 Neural networks:

Neural networks are a popular type of machine learning model. A special case of a neural network called Binary neural network (BNN) is the primary focus of this thesis.

3.1.1 Binary Neural Network:

Binary neural networks are networks with binary weights and activations at run time. At training time these weights and activations are used for computing gradients. However, the gradients and true weights are stored in full precision. This procedure allows us to effectively train a network on systems with fewer resources.

Binarization Procedure:

- **Forward binarization:**

For forward propagation, it will require two *binary* matrices; thus binarize the weight matrix and the incoming activation from the previous layer. The binarization procedure is illustrated below.

$$\begin{matrix} \begin{bmatrix} -0.4 & -0.4 & 0.9 \\ 0.9 & 0.4 & 0.8 \\ 0.4 & -0.4 & -0.4 \end{bmatrix} & \approx & 0.2 & \begin{bmatrix} -1 & -1 & 1 \\ 1 & 1 & 1 \\ 1 & -1 & -1 \end{bmatrix} \\ W & & & \alpha W^B \end{matrix}$$
$$W^B = \text{sign}(W)$$
$$\alpha = \frac{1}{n} \|W\|_{l_1}$$

Figure 3.1.1(a): Forward Binarization [21]

- **Gradient Propagation through Discretization:**

The derivative of the sign function is zero almost everywhere, making it incompatible with back propagation. Thus, a straight-through estimator is used. This preserves the gradient's information and cancels large gradients. The gradient propagation procedure is illustrated below.

$$\begin{array}{ccc}
 \begin{array}{|c|c|c|} \hline -0.4 & -0.4 & 1.4 \\ \hline 1.2 & 0.4 & 0.8 \\ \hline 0.4 & -0.4 & -0.4 \\ \hline \end{array} & \approx & \begin{array}{|c|c|c|} \hline -0.4 & -0.4 & 0 \\ \hline 0 & 0.4 & 0.8 \\ \hline 0.4 & -0.4 & -0.4 \\ \hline \end{array} \\
 g_q & & g_r
 \end{array}
 \quad g_r = g_q \cdot 1_{|r| \leq 1}$$

Figure 3.1.1(b): Gradient Propagation ^[21]

- **Forward Pass for Binary Matrices:**

The figure below represents a matrix composed of only +/- 1s, which can be converted to a binary matrix, and replace fused multiply-add (FMA) operations with bit manipulations to give the same results. This observation will be used to generate binary matrices, which allows gaining significant speedup and decreasing the memory overhead notably. It is important to note that the pop count function increments for every ON bit and decrements for every OFF bit. This is not how built in pop count function of GCC works. However, the built in pop count function can be used in the following pattern (if using unsigned int to bit-pack matrices).

$$\text{Value} = 2 * (\text{built in pop count}(\sim (A \wedge B))) - 32$$

$$\begin{array}{ccc}
 \begin{array}{|c|c|c|} \hline -1 & 1 & 1 \\ \hline -1 & -1 & -1 \\ \hline +1 & 1 & -1 \\ \hline \end{array} & \begin{array}{|c|} \hline -1 \\ \hline +1 \\ \hline +1 \\ \hline \end{array} & = & \begin{array}{|c|c|c|} \hline (-1 \times 1) + (1 \times 1) + (1 \times 1) \\ \hline (-1 \times -1) + (1 \times -1) + (1 \times -1) \\ \hline (-1 \times 1) + (1 \times 1) + (1 \times -1) \\ \hline \end{array} & = & \begin{array}{|c|} \hline 3 \\ \hline -1 \\ \hline -1 \\ \hline \end{array} \\
 \\
 \begin{array}{|c|c|c|} \hline 0 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline +1 & 1 & 0 \\ \hline \end{array} & \begin{array}{|c|} \hline 0 \\ \hline +1 \\ \hline +1 \\ \hline \end{array} & = & \begin{array}{|c|c|c|} \hline \text{popcnt}(\text{xnor}(011, 011)) \\ \hline \text{popcnt}(\text{xnor}(011, 000)) \\ \hline \text{popcnt}(\text{xnor}(011, 110)) \\ \hline \end{array} & = & \begin{array}{|c|} \hline 3 \\ \hline -1 \\ \hline -1 \\ \hline \end{array}
 \end{array}$$

popcnt(xnor(011, 110)) = popcnt(xnor(0,1), xnor(1,1), xnor(1,0)) = popcnt(010) = -1 + 1 - 1 = -1

Figure 3.1.1 (c): Forward Passed Binary Matrices ^[21]

- **Implementing binary neural networks:**

A bare-bones version of a BNN of the structure below was implemented in the Wolfram Language. The network was trained on the Intel CPUs on the MNIST dataset.

This network has the following layers:

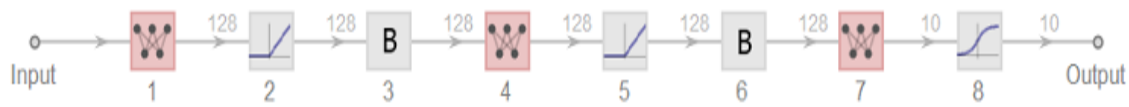


Figure 3.1.1(d): Implementation of Binary Neural Network ^[21]

- 1 Fully connected (128)
- 2 Ramp - rectified linear unit (ReLU) activation function
- 3 Binarize activation
- 4 Fully connected (128)
- 5 Ramp - ReLU activation function
- 6 Binarize activations
- 7 Fully connected (10)
- 8 Sigmoid activation function

The input layer is left unbinarized. This is due to the high-dimensional geometry of neural networks. After numerical experiments on CIFAR10 conducted, it was seen that the dot products of activations with the pre-binarization and post-binarization weights are highly correlated. However, this correlation is weaker in the first layer. This is a strong reason to never binarize the first layer of a BNN.

- ***Dot product preservation***

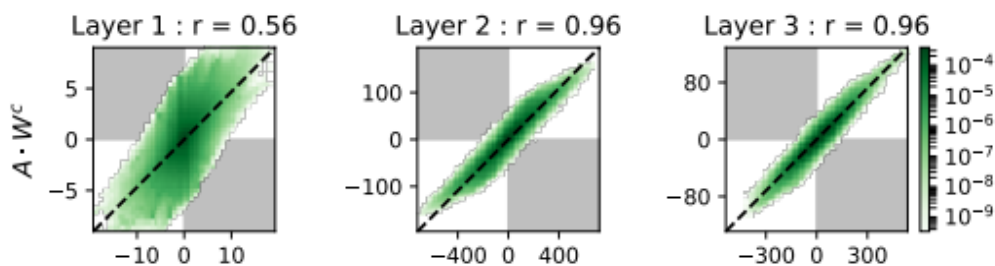


Figure 3.1.1(e): Dot Product Preservation ^[21]

Figure shows a 2D histogram of the dot products between the binarized weights and the activations (x-axis), and the dot products between the continuous weights and the activations (y-axis).

- **Angle preservation property**

It has been demonstrated that binarization approximately preserves the direction of high-dimensional vectors. The angle between a random vector and its binarized version converges to approximately 37 degrees as the vector dimension approaches infinity. This network uses Adam optimizer.

- **Accuracy analysis**

The BNN of the architecture previously provided was trained on the MNIST dataset. It is clearly represented that the BNN has slower convergence than the full precision counterpart. However, it is also observed that a fairly low parameter network (the BNN) is able to closely mimic the performance of the full precision network.

These metrics further validate the legitimacy of the dot product preservation and the angle preservation property discussed above.

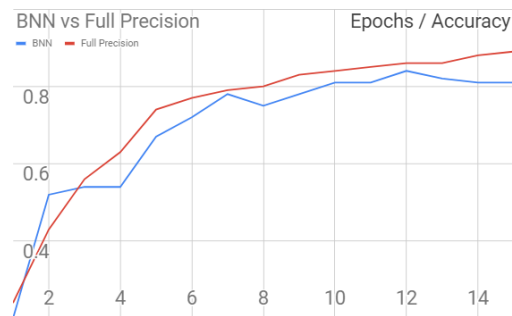


Figure 3.1.1(f): Graph of accuracy analysis of Binary Neural network ^[21]

3.1.2 YOLO:

YOLO stands for You Look Only Once and is a network for object detection task consisting the determination of the location on the images where some sort of objects is present, as well classifying objects. YOLO was initially introduced in June 2015 followed up by the latest version in November 2016. YOLO offers a truly integrated solution with a simplified implementation of GPU. Darknet is also developed by the same team so it is quite easy to run the network using the configuration files used in Darknet. This neural network is very useful for the open source community as it is very difficult and hectic to implement a neural network on existing framework due to introduction of new algorithms which are not supported by the frameworks.

Architecture:

The basic and major advantage of using YOLO is that YOLO does not support any Regional Proposal Network (RPN). RPN is a popular algorithm which gained attention of many Data scientists and AI Engineers. It has enormous application to detect objects in a self-driven car, assisting disabled person and helps them to get independent etc. Actually this algorithm specially dwells into the logic and math behind how an algorithm gets the box around the recognized object. It is also considered as the backbone of R-CNN. R-CNN is the older model used to perform this type of task which uses pipeline to perform in multiple steps and can be slow and also hard to optimize at the same time because every individual component

must be trained separately in R-CNN. YOLO do the same job with just a single neural network. In YOLO, the detection and the classification are done by the same network trained end-to-end. This makes the architecture simpler. YOLO has three basic models:

- Regular YOLO model
- Small YOLO model
- Tiny YOLO model

Yolo is likewise a massive system like the recently examined systems. However, there is a new tiny model developed by the development team which is smaller than the original one. This tiny model based implementation is known as Fast YOLO. This Fast YOLO has 11 layers and out of these 9 layers is convolutional and the rest 2 layers are completely associated layers. This model is very small model as in comparison with R-FCN which contains ResNET layers and VGG- 16 with RPN which posse's almost 20 layers joined in Faster R-CNN. The figure below shows the architecture of tiny YOLO

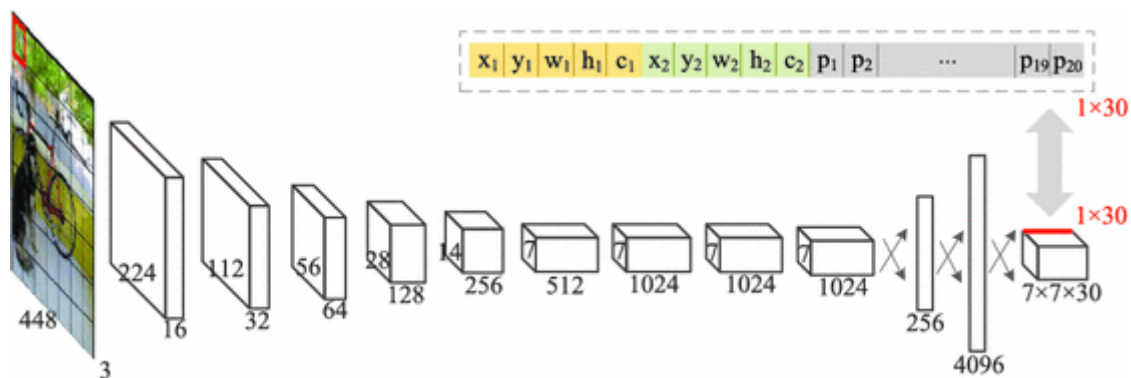


Figure 3.1.1(g): YOLO Architecture ^[19]

Specifications:

Tiny YOLO require almost 33.3 MB of space in order to process any image by assuming 32-bit is required by each parameter. The output results can be reproduced just by adding the parameters and calculating memory space for them. Tiny YOLO has 45 millions of weights parameters and size of these weights is almost 170MB. The pooling layers and activation layers does not have any parameters. So they will be calculated during the calculation of memory space. The tiny/Fast YOLO network model is made uo of some basic layers and they are following:

- Convolutional Layer.
- Max Pooling Layer.

- ReLU Activation Layer.
- Fully Connected Layer.

3.1.3 Convolutional Neural Network:

I had given a short description about the Convolutional Neural Network in Chapter 1 in the ANN section. The major difference between CNN and other neural network is lies in the manner their neurons connect to each other and interconnected between the layers more efficiently by selecting the neurons that will have the maximum effect on the output computations systematically. A convolutional neural network gives better result with the reduction in memory and power. Different layers are used in this network to compress memory and eliminate neurons with as much as less impact.

Terminology:

Some of the most popularly used terminologies for a convolutional neural network are feature maps, kernel, bias, and stride. A clear understanding of these terms is very important in order to understand how a neural network model works.

- **Feature Maps:** the first layer of a convolutional neural network is an input image layer in which the convolutional kernel is applied and another next layer is generated. Hence, the data at the first layer is called *Input Features* and the data at the second layer is called *Output Feature*. Both of this jointly known as feature maps. Now for the next step, the data at the second layer is known as *input feature* and data at the next layer is known as *output feature*.
- **Kernel:** kernel is a one type of small filter which generates the output features from input features and is also known as feature extractor. As we can see in the figure, each and every layer has some depths. The number inside the kernel is known as weights and these weights are generated by the training of a neural model.
- **Bias:** bias is a fixed constant number which is added to all the pixels on which the kernel is applied.
- **Stride:** stride is the amount of a pixel which is moved by a kernel. For example, a shift of one pixel will be known as stride of 1. Generally stride are only considered at vertically or horizontally movements, they are not considered at diagonally movements.

Convolutional layer:

A convolutional layer is the most important layer in a convolutional neural network. It is the basic layer whose multiple instances can be joined together to create a form of a

convolutional neural network without any assistance from any other layers. A convolutional layer has three major components:

- Input feature maps
- Feature extraction
- Output feature maps

The feature extraction generates output feature from the IP features by performing simple convolution. After generation of every output feature, the feature extraction slides with a stride of 1 or more as it is configured.

Pooling Layer:

Pooling layer helps in decreasing the size of feature maps. Because of huge memory requirements on convolutional neural network, it is important to decrease the feature size so that the essential information is yet saved. We regularly do max pooling for this reason. In this procedure, we select the most extreme incentive from a predefined kernel region. The most ordinarily utilized kernel size for max pooling in CNNs is 2x2. The figure below represents the way a feature map can be decreased to a large portion of its size by utilizing a maximum pooling filter. It is vital to note here that much of the time the extent of stride for pooling is same as the size of kernel. Any way in some unique applications like semantic division, overlapping maximum pooling can be utilized to extend feature maps.

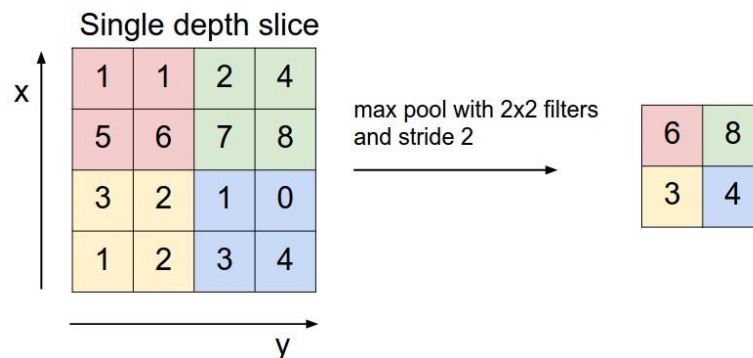


Figure 3.1.1(h): Maximum Pooling Layer^[22]

Activation Layer:

Activation layer is another way to simplify feature maps. There are various types of activation layers available which we can utilize. Some of the functions are mentioned below:

- Rectified Linear Unit (ReLU)
- Logistic functions (Sigmoid)
- Tangent functions

Here we will not focus on rectified linear unit as it is the most popularly used by many researchers and we had also used the same. For standard ReLU, we easily convert all the negative values to 0 and keep the positive values in their original one. Another function that can be utilized is Leaky ReLU. In this type of function all the negative values are multiplied by a certain factor. We are using this configuration as YOLO works on the same configuration.

Fully Connected Layers:

This layer is based on fully connected neural networks i.e. FCNs. In CNNs pooling layer and activation layers are utilized in some of the layers in order to reduce the size of the network. After that the fully connected layers are been utilized in order to connect the neurons with each other. The figure below represents a fully connected layer.

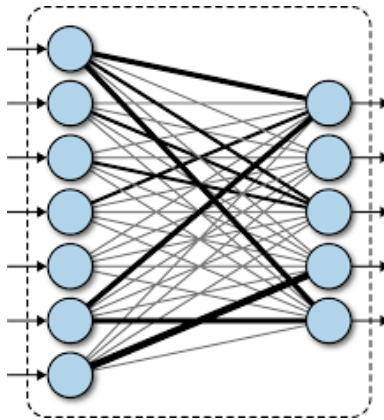


Figure 3.1.1(i): Fully connected layer ^[24]

3.2 Libraries Considered:

There are many libraries used from a couple of years. They have provided a convenient way to implement various convolutional neural networks without the requirement of converting complex codes. Most of the frameworks and libraries make the description of the networks in programming languages like C or C++ etc. the user can easily define the network easily along with training parameters. It is not necessarily required to write code in C or C++ for the users who want to change the network. There are several important and popularly known libraries and frameworks used at the recent time in the industries and they are:

- Caffe
- Torch
- Tensorflow

- Theano
- Darknet
- OpenCV

In this thesis we have utilized Tensorflow and OpenCV. Both of this are introduced by industrial community and open source libraries and framework used at a large extent in research.

3.2.1 Tensorflow:

TensorFlow is a free software library focused on machine learning created by Google. Initially released as part of the Apache 2.0 open-source license, TensorFlow was originally developed by engineers and researchers of the Google Brain Team, mainly for internal use. TensorFlow is considered the successor of the closed-source application Belief and is presently used by Google for research and production purposes. TensorFlow is considered the first serious implementation of a framework focused on deep learning. TensorFlow is also known as Google TensorFlow.

TensorFlow derives its name from the multidimensional arrays known as tensors, which are used by the neural networks for different operations. According to Google, compared to DistBelief, TensorFlow is faster, smarter and more flexible and can be easily adaptable to new areas and products. It was mainly created for deep neural network research and for facilitating machine learning, though TensorFlow has been used in a wide range of other areas as well.

TensorFlow function works by sorting through layers of data (nodes) as part of learning. In the first layer, the system determines the basic features of the object. As deeper movements occur, it looks for more refined information regarding the object. The sorting of images is done at a faster rate, thus giving users more valuable information. TensorFlow is available on different operating systems such as Linux, Windows, and MacOS and also on mobile operating platforms like iOS and Android. One of the salient features of TensorFlow is that it is capable of running on multiple CPUs and GPUs. The computations in TensorFlow are reported as state-full dataflow graphs. Currently TensorFlow is used in over six thousand free online repositories.

3.2.2 Open Computer Vision (OpenCV):

Computer vision deals with the extraction of important information from the contents of digital images or video. This is distinct from image processing, which involves manipulating visual information on the pixel level. Applications of computer vision

include image classification, visual detection, and 3D scene reconstruction from 2D images, image retrieval, augmented reality, and machine vision and traffic automation. Today, machine learning is a necessary component of many computer vision algorithms. Such algorithms can be described as a combination of image processing and machine learning. Effective solutions require algorithms that can cope with the vast amount of information contained in visual images, and critically for many applications, can carry out the computation in real time.

3.3 Datasets:

Machine learning is the most important and famous process for developing Artificial Intelligence which is based on software and applications. To train such machines a vast amount of related and effective dataset is required to apply the right algorithm and API to enable the machine work smoothly without human interference. There are three types of data sets: **Training, Dev and Test**, which are used at various stage of development. Training dataset is the largest of three of them, while test data functions as seal of approval and you don't need to use till the end of the development.

Training Dataset: The training data set in Machine Learning is the actual dataset used to train the model for performing various actions. This is the actual data the ongoing development process models learn with various API and algorithm to train the machine to work automatically.

Testing Dataset: The training data set in Machine Learning is the actual dataset used to train the model for performing various actions. This is the actual data the ongoing development process models learn with various API and algorithm to train the machine to work automatically.

Since the aim is to explore various neural network in order to implement on FPGA. We have explored a couple of datasets to be utilized along with these neural networks and Libraries. These datasets have been used as reference.

3.3.1. Cifar 10 and Cifar 100:

The CIFAR-10 dataset consists of 60000 (32x32) color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. The images in the dataset are classified in following

classes: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship and Truck respectively. The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

CIFAR-100 dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 super classes. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the super class to which it belongs).

3.3.2. COCO:

Common Object in Context (COCO) dataset is a large scale object detection, segmentation and captioning dataset with several features. For example, object segmentation, recognition in context, 80 object categories, 5 captions per image, labeling etc. the name contains a word context because images in this dataset are taken from daily life scenes and provides contextual information. This dataset is a bunch of 80000 training images and 40000 validation images. This dataset is explored in order to use it with YOLO network as this network is initially trained with COCO dataset.

Chapter 4

Hardware, Tools and Languages

As discussed earlier in the previous chapter, the main aim of the thesis is to represent a FPGA platform re-created which includes neural network implementation, and HDMI and Audio implementation. This platform is to be used in SDSoC tool which is introduced by Xilinx. This platform can be modified according to the need of the user. In this chapter, we will discuss the development flow, short description of tools and the way they were used in this project. The targeted FPGA board used in this project is PYNQ which possess the base functionality of Zynq board along with Python productivity. We have selected this board because it possess a small architecture i.e. 32-bit and provides the web based architecture which can be accessed by using Jupyter web-tool.

4.1 PYNQ (Python Productivity for Zynq) board:

PYNQ is an open-source hardware platform for the Pynq open-source framework introduced and manufactured by Xilinx in 2017. This board is very easy to use in designing embedded systems and easiest board having designed embedded system with Xilinx Zynq Systems on Chips (SoCs). The platform supports the Python language and libraries as programming environment, helping designers to explore new benefits of combining programmable logic (hardware accelerated functionality) to microprocessors in a single board. Pynq helps it users to create high performance embedded applications along with various important features like low latency control, video processing at high frame rate, hardware accelerated algorithms and parallel hardware execution

There are two types of PYNQ boards available: PYNQ-Z1 and PYNQ-Z2. In this project, I have used Pynq-Z1. The board is supplied by Digilent Inc. PYNQ is an open-source framework that enables embedded programmers to explore the capabilities of Xilinx Zynq All Programmable SoCs (APSoCs) simplifying the use of programmable logic circuits as hardware accelerators. Instead, the APSoC is programmed using Python, with the code developed and tested directly on the board. The programmable logic circuits are imported as hardware libraries and programmed through their APIs in essentially the same way that the software libraries are imported and programmed.

The programmable part of PYNQ-Z1 is based on ARM A9 CPUs and Linux-based programming environment that includes:

- A web server hosting the Jupyter Notebook design environment

- The IPython kernel and packages
- A hardware library and API for the hardware accelerators running on the FPGA (Field Programmable Gate Array), the reconfigurable digital circuit.

Xilinx provides Vivado webPACK tools, free of cost for the designers who want to extend the base system by contributing new hardware libraries and IPs (Intellectual Property components). PYNQ is intended to be used by a wide range of designers and developers including:

- Software developers who want to take advantage of the capabilities of Zynq and programmable hardware without using ASIC-style design tools to design hardware.
- System architects who want an easy software interface and framework for rapid prototyping and development of their Zynq design.
- Hardware designers who want their designs to be used by the widest possible audience.

4.2 Jupyter:

The notebook allows the console-based approach to interactive computing in a way by providing a web-based application suitable for developing, documenting, and executing code, and communicating the results. The Jupyter notebook combines two components:

- **A web application:** A browser-based tool for modifying the documents, which includes text, mathematic calculations, computations and media output.
- **Notebook documents:** A representation of all content available in the web application, including inputs and outputs of the computations, explanatory text, mathematics, images, and media representations of objects.

Main features of the web application:

- In-browser editing for code, with automatic syntax highlighting, indentation, and tab completion/introspection.
- Allows to execute code from the browser, with the results of computations attached to the code which generated them.
- Displaying the result of computation using media representations
- In-browser editing for text using the Markdown markup language, which can provide commentary for the code, and it is not limited to plain text.
- The ability to easily include mathematical notation within markdown cells using latex, and rendered natively by MathJax.

Notebook documents:

A notebook document contains the inputs and outputs of the session as well as the code but is not meant for execution. In this way, notebook files serve a complete computational record of a session, interleaving executable code with explanatory text, mathematics, and representations of results. These documents are JSON files and saved with the **“.ipynb”** extension. Since JSON is a plain text format, they are shareable. Notebooks may be exported in the static formats, including HTML (for example, for blog posts), LaTeX, PDF, and slide shows, via the nbconvert command. Furthermore, any **“.ipynb”** notebook document available from a public URL can be shared via the Jupyter Notebook Viewer. This service loads the notebook document from the URL and renders it as a static web page. The results may thus be shareable without other users needing to install the Jupyter notebook themselves.

Starting the notebook server:

To start running notebook servers from the command line use the following command:

- **Jupyter Notebook:**

To create new notebooks from the dashboard with the New Notebook open existing ones by clicking on their name. You can also drag and drop **“.ipynb”** notebooks and standard **“.py”** Python source code files into the notebook list area. When starting a notebook server from the command line, you can also open a particular notebook directly, bypassing the dashboard, with Jupyter Notebook **“my_notebook.ipynb”**. The **“.ipynb”** extension is assumed if no extension is given. When you are inside an open notebook, **“Open”** menu option will open the dashboard in a new browser tab, allows to open another notebook from the notebook directory or to create a new notebook.

- **Creating a new notebook document:**

A new notebook may be created at any time, either from the dashboard, or using the **“File → New”** menu option in an active notebook. The new notebook is created in the same directory and will open in a new browser tab. It will also be reflected as a new entry in the notebook list on the dashboard. An open notebook has exactly one active session connected to a kernel, which will execute code sent by the user and communicate back results. This kernel remains active if the web browser window is closed, and reopening the same notebook from the dashboard will reconnect the web application to the same kernel. In the dashboard,

notebooks with an active kernel have a Shutdown button next to them, whereas notebooks without an active kernel have a Delete button in its place.

- **Structure of a notebook document:**

The notebook consists of a sequence of cells. A cell is a multiline text input field, and its contents can be executed by using **Shift-Enter**, or by clicking either the “**Play** or Cell, Run” button in the menu bar. The execution behavior of a cell is determined by the cell’s type. There are three types of cells: **code cells, markdown cells, and raw cells**. Every cell starts off being a code cell, but its type can be changed by using a drop-down on the toolbar or via keyboard shortcuts.

- **Code cells:**

A code cell allows you to edit and write new code, with full syntax highlighting and tab completion. The programming language depends on the kernel, and the default kernel (IPython) runs Python code. When a code cell is executed, the code is sent to the kernel associated with the notebook. The results that are returned from this computation are then displayed in the notebook as the cell’s output. The output can be in the form of text, audio, image or video, including matplotlib figures and HTML table. This is known as IPython’s display capability.

- **Markdown cells:**

In IPython header text can be written by marking up text with the Markdown language. The corresponding cells are called Markdown cells. The Markdown language provides a simple way to perform the text markup with different fonts, bold, form lists, size etc. If you want to provide structure for your document, you can use markdown headings. Markdown headings consist of 1 to 6 hash (#) signs followed by a space and the title of the section. It is also used as a hint or explanation when exporting to other document formats, like PDF. When a Markdown cell is executed, the Markdown code is converted into the corresponding formatted text. Markdown allows arbitrary HTML code for formatting.

4.3 Python:

Python is probably the easiest-to-learn and nicest-to-use programming language with widespread use. Python code is clear to read and write, and it is concise without being cryptic. Python is a very expressive language, which means that I can usually write far fewer lines of Python code than requiring for an equivalent application written in, say, C++ or Java. Python is a cross-platform language. In general, the same Python program can be run on Windows and Unix-like systems such as Linux, BSD, and Mac OS X, simply by copying the

file or files that makes the program to the target machine, with no “building” or compiling necessary.

It is possible to create Python programs that use platform-specific functionality, but this is rarely necessary since most of the Python’s standard library and most third-party libraries are fully cross-platform. One of Python’s great strengths is that it comes with a very complete standard library this allows user to download a file from the Internet, unpack a compressed archive file, or create a web server, all with just one or a few lines of code. And in addition to the standard library, thousands of third party libraries are available, some providing more powerful and sophisticated facilities than the standard library for example, the Twisted networking library and the NumPy numeric library. While others provide functionality that is too specialized to be included in the standard library for example, the SimPy simulation package. Most of the third-party libraries are available from the Python Package Index.

Python can be used to program object-oriented programs as it is an object-oriented language. Python 3 is the upgraded version of python language which is advanced version on Python 2, some older practices are no longer appropriate or necessary in Python 3, and new practices have been introduced to take advantage of Python 3 features. Python 3 is a better language than Python 2. It builds on the many years of experience with Python 2 and adds lots of new features to make it even more convenient, easier, and consistent to use than Python 2.

4.4. Overlay:

Overlays are analogous to software libraries. A programmer can download overlays into the Zynq PL at runtime to provide functionality required by the software application. An Overlay is a class of Programmable Logic design. Programmable Logic designs are usually highly optimized for a specific task. Overlays however, are designed to be configurable, and reusable for broad set of applications. A PYNQ overlay will have a Python interface, allowing a software programmer to use it like any other Python package. A programmer can use an overlay, but will not usually create the overlays, as this is a specialized task for a hardware designer. This section will give an overview of the process of creating an overlay and integrating it into PYNQ, but will not cover the hardware design process in detail.

An overlay consists of two main parts; the Programmable Logic (PL) design, and the Python API. Xilinx Vivado EDA (Electronic Design Automation) software is used to create the PL. The tool will generate a *bitstream* or *binary* file (.bit file) that is used to program the

Zynq PL. The free webpack version of Vivado can be used with the PYNQ-Z1 board to create overlays.

There are some differences between the standard Zynq design process and designing overlays for PYNQ. A Vivado project for a Zynq design consists of two parts; the PL design, and its associated PS configuration settings. The PS configuration includes settings for system clocks, including the clocks used in the PL. The PYNQ image which is used to boot the board configures the Zynq PS at boot time. Overlays are downloaded as required by the programmer on the PL and this process will not reconfigure the PS. This means that overlay designers should ensure the PS settings in their Vivado project match the PYNQ image settings.

4.5. Tools:

Design Flow:

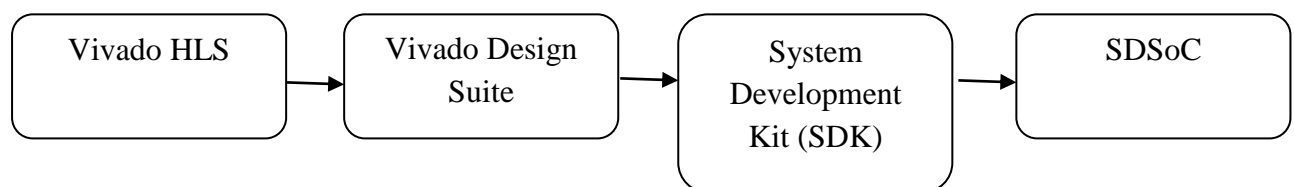


Figure 4: Design Flow of Platform

4.5.1 HLS (High Level Synthesis):

The Vivado High-Level Synthesis (HLS) tool enables C, C++ and SystemC programs to be directly targeted into Xilinx devices without the need to manually create RTL. Vivado HLS is widely reviewed to increase developer productivity, and is confirmed to support C++ classes, templates, functions and operator overloading. The tool includes a built-in logic simulator, high-level synthesis, and a tool chain that converts C code into programmable logic. Vivado 2014.1 introduced support for automatically converting OpenCL kernels to IP for Xilinx devices. OpenCL kernels are programs that execute across various CPU, GPU and FPGA platforms. The Vivado Simulator is a component of the Vivado Design Suite. It is a compiled-language simulator that supports mixed-language, TCL scripts, encrypted IP and enhanced verification. The Vivado IP Integrator allows engineers to quickly integrate and configure IP from the large Xilinx IP library. The Integrator is also tuned for MathWorks Simulink designs built with Xilinx's System Generator and Vivado High-Level Synthesis. The Vivado TCL Store is a scripting system for developing add-ons to Vivado, and can be used to add to and modify Vivado's capabilities. TCL stands for Tool

Command Language, and is the scripting language on which Vivado itself is based. All the Vivado's underlying functions can be invoked and controlled via TCL scripts.

Device Support:

As of 2018, Xilinx recommends Vivado Design Suite for new designs with Ultrascale, Ultrascale+, Virtex-7, Kintex-7, Artix-7, and Zynq-7000. Vivado supports newer high capacity devices, and speeds the design of programmable logic and I/O. Vivado provides faster integration and implementation for programmable systems into devices with 3D stacked silicon interconnect technology, ARM processing systems, analog mixed signal (AMS), and many semiconductor intellectual property (IP) cores.

Creating the IP to be used by an Overlay:

An overlay is created for a particular FPGA device and platform. The overlay will include the IP block that will be invoked during overlay execution. The IP block must then be created for the specific FPGA family it will run on. For instance, during Vivado project setting, we specify the Zynq 7020 family by selecting the *Target Device* (xc7z020clg400-1). We can also configure the clocks and associated frequencies to use.

4.5.2 Vivado Design Suite:

Vivado is the Xilinx EDA tool that supports the generation of hardware for the 7 series FPGA, Ultrascale and other recent families. All tools in Vivado except SDK and Vivado HLS (High Level Synthesis) are integrated on a common GUI (Graphical User Interface). The Vivado IP Integrator allows engineers to quickly integrate and configure IP from the large Xilinx IP library. The Vivado TCL Store is a scripting system for developing add-ons to Vivado, and can be used to add to and modify Vivado's capabilities.

Vivado, introduced in April 2012, is a design environment for FPGA products from Xilinx, and is tightly-coupled to the architecture of such chips. Vivado Design Suite is a software suite produced by Xilinx for synthesis and analysis of HDL (Hardware Description Language such as VHDL or Verilog) designs. Vivado includes electronic system level (ESL) design tools for synthesizing and verifying C-based algorithmic IP; standards based packaging of both algorithmic and RTL IP for reuse; standards based IP stitching and systems integration of all types of system building blocks; and the verification of blocks and systems. Vivado allows developers to synthesize their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with a programmer.

Exporting the IP and associated overlay to the SD card:

Within SDSoc we can export the created IP block to be used by the Python code as an overlay. The tool will create a TCL file and a bitstream. The bitstream is the code that will be used to configure the FPGA to implement the overlay IP. The TCL file can also be used to build the project within Vivado and see the system block diagram. Within Vivado we can see and export the created block diagram using the GUI (File>Export>Block Design) or by running a TCL command (`write_bd_tcl`). It will generate the bitstream and the tcl file, both must have the same name as the top IP block to be exported and invoked by the Overlay Python command. The exported files must be copied into the SD card and placed in the directory to be seen by the Python tool.

Loading and using the Overlay

Within a Jupyter sheet, we can invoke in Python the overlay to be used as a hardware accelerated function by importing the Overlay package (***from Pynq import Overlay***). For instance, the base overlay (it contains the basic functionality for the PL of the Pynq board) can be loaded onto the PL by using the following Python command: **ol = Overlay("base.bit")**. This will execute the TCL file `base.tcl` that will load the bitstream `base.bit` on the PL region and return the object that will be used to invoke within Python the hardware functionality available.

You can use the **ip_dict** member function of the Overlay class to see the content of the overlay. The Overlay package generates a dictionary called **ip_dict** containing the names of the IPs in a specific overlay (e.g. `base.bit`). The dictionary can be used to reference an IP by name in the Python code, rather than by a hard coded address. It can also check the IP available in an overlay. For instance, the python code **ol.ip_dict** will show the IPs in the form `'SEG_mb_bram_ctrl_1_Mem0': ['0x40000000', '0x10000', None]`. Each entry in the IP dictionary is returned as a key-value pair after parsing the TCL file exported with the bitstream. All the IP instance names are parsed from the *.tcl file (e.g. `base.tcl`) in the address segment section. The key is the name of the IP instance (e.g. `SEG_mb_bram_ctrl_1_Mem0`) followed by 3 items: the base address of the addressable IP (in HEX), the address range in bytes (in HEX) and the state associated with the IP (it is `NONE` by default, but it can be user defined).

The PL package can be used to find the addressable IPs currently in the programmable logic. For that you must first import the package (***from Pynq import PL***, then invoke the dictionary **PL.ip_dict**).

4.5.4 SDSoC (Software-Defined System on Chip):

SDSoC is one of the newest development tools available from Xilinx for designing embedded systems. SDSoC stands for Software Defined System-On-a-Chip. SDSoC aims to provide a software-like development environment for implementing applications on multi-processor platforms and seamlessly enable migration of functions executing in software to functions executing in hardware (HW). Within a given application, user can specify which functions to be implemented in HW. The tool automatically generates the equivalent HDL code for the HW portion of the reconfigurable circuit.


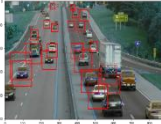

The design steps (synthesis, bit stream generation for HW and compile/link for the SW portion) are guided by user-defined pragmas, to perform a specific optimization or impose user constraints. First step is to select a target platform (the platform where the desired hardware will be implemented, this can be a board or an FPGA device), then cross-compile the application (for the software part composed by drivers and APIs), and ensure it runs properly on the platform (during debugging or profiling). The part of the original program identified as compute-intensive hot spot can then migrate into the PL to improve the system performance. The migration will consist on the creation of the IP block for the selected function, which will then be encapsulated, interfaced and drivers will be created for software interaction. The SDSoC tool can generate a complete system-on-chip, which includes the SD card boot image for the application.

The tool allows instrumentation of the code to analyze performance, and if necessary, optimize the system and hardware functions using a set of directives (pragmas) and tools within the SDSoC development environment

Chapter 5

Experiments and Results

The results we achieved by evaluating the above mentioned algorithms are in terms of speed and accuracy we can state that among all this algorithms, Yolo is the best algorithms to develop a system for the detection of moving objects using moving camera because as we know Yolo is already a pre-trained model with coco dataset available with 80 classes. It can be implemented on 32-bit FPGA as well as CPU also. We have evaluated all the algorithms on CPU with the help of anaconda navigator which provides virtual environment and various important and usable libraries such as scipy, pandas etc and also with Pynq FPGA board. The result achieved during the evaluation is presented in Table 1. The result presented is of images focusing on different situations and different objectives and is in terms of speed per frame and accuracy of the model per frame.

Frame	OpenCV	BNN	Tensorflow	Yolo
 (a)	3.7954 Haarcascade classifier 54%	0.00035 cifar-10 classifier 71%	0.076 Keras (Cifar-100) classifier 69%	0.0012 Coco dataset 73%
 (b)	2.7875 Haarcascade classifier 62%	0.00051 cifar-10 classifier 73%	0.063 keras(Cifar-100) 68%	0.0039 Coco dataset 70%
 (c)	2.1238 Haarcascade classifier 59%	0.00041 cifar-10 classifier 68%	0.067 Keras classifier (Cifar-100) 70%	0.0025 Coco dataset 75%


	2.3514 Haarcascade Classifier 58%	0.00045 Cifar-10 classifier 71%	0.066 Keras classifier (cifar-100) 70%	0.0029 Coco dataset 71%
---	---	---------------------------------------	--	-------------------------------

Table 5.1: Evaluation results

By the above mentioned experiment, I came to a result that Yolo is the most flexible and accurate neural network I can train and implement in Pynq FPGA board. After training and testing the network on CPU I needed to create an IP in order to test it in FPGA hence I used Xilinx High Level Synthesis Tool because of its synthesis and RTL export functionality.

After exporting RTL design, a custom Yolo IP can be created and Inserted in Vivado tool in order to create an Overlay which can be imported in Pynq FPGA board and we can test this Overlay by using Jupyter web Architecture and Python Productivity Language. In Figure 5.1 represents the Design of Overlay created using Xilinx Vivado tool and The result I had achieved using the HDMI functionality of this overlay is represented in figure 5.2.

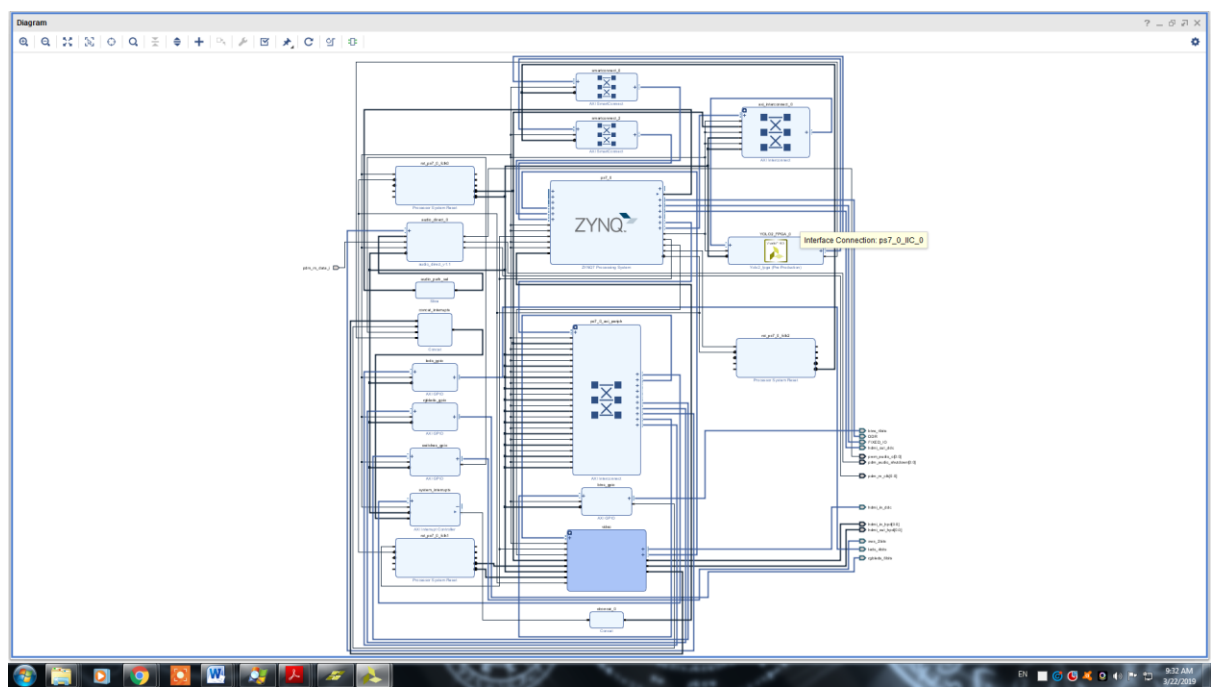


Figure 5.1: Vivado Design of Overlay

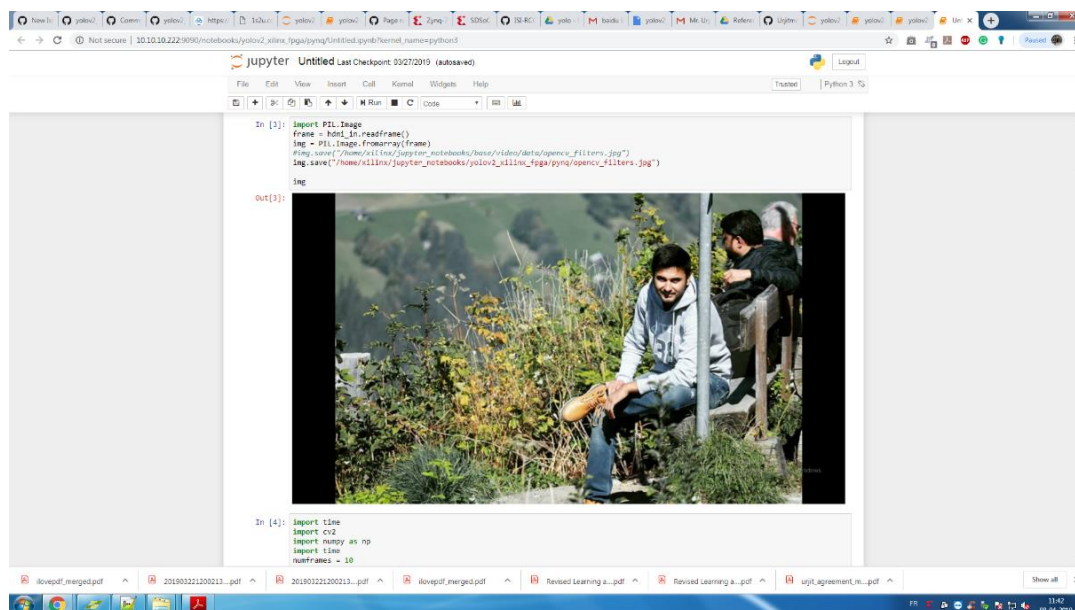


Figure 5.2: Image captured through HDMI of the FPGA

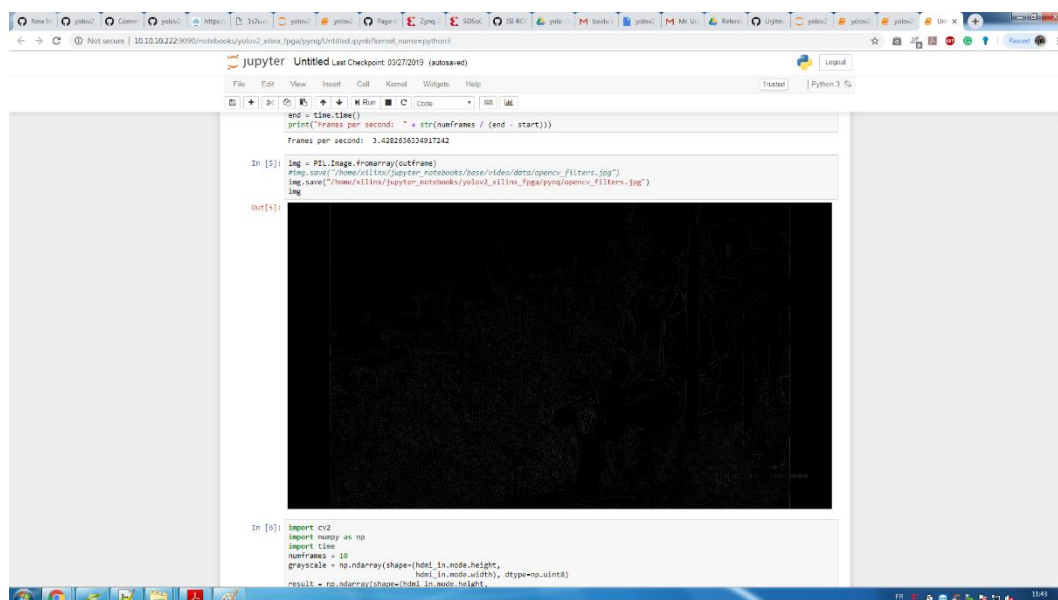


Figure 5.3: Image achieved after processing Grayscale on the captured image

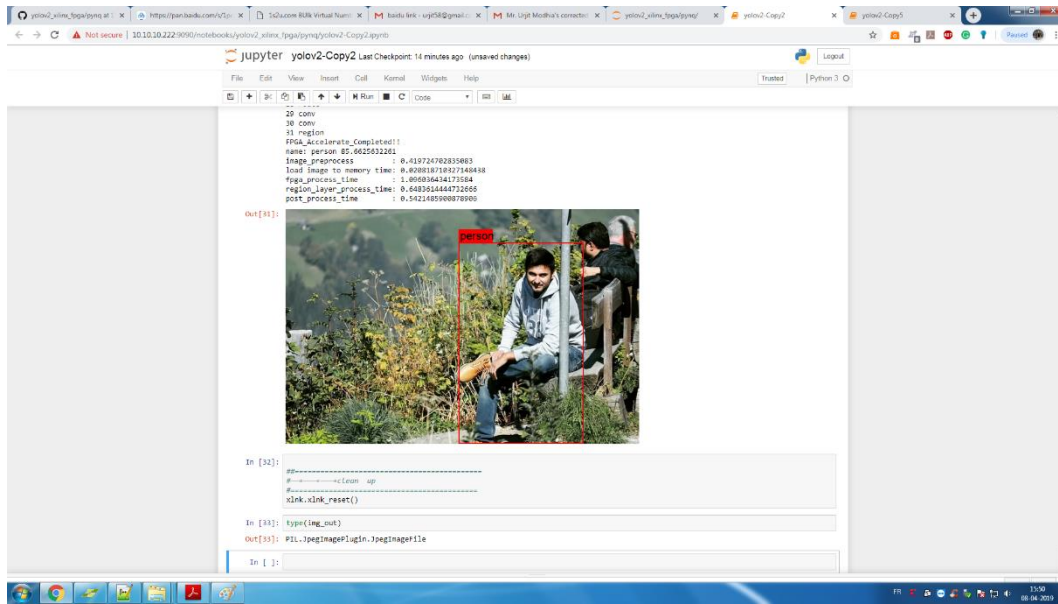


Figure 5.4: neural network result (YOLO)

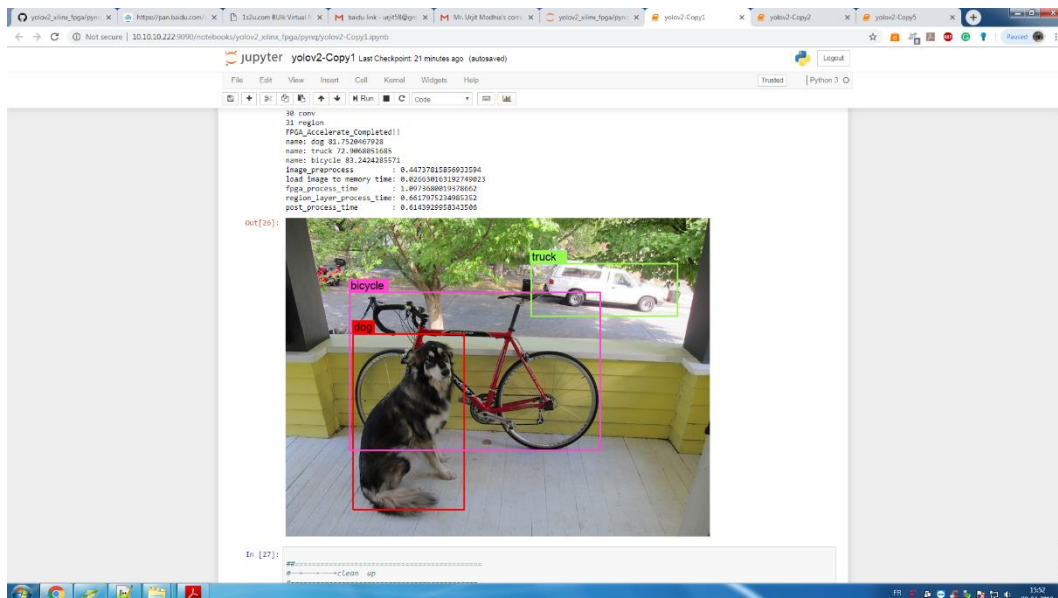


Figure 5.5: neural network result (YOLO)

All the results presented above belongs to the Overlay which was created using Xilinx tools. Now the result below is belongs to the test done on the platform using SDSoC tool. This result presents the LUTs, Flipflop's, BRAMs and DSPs used by the platform while testing the platform

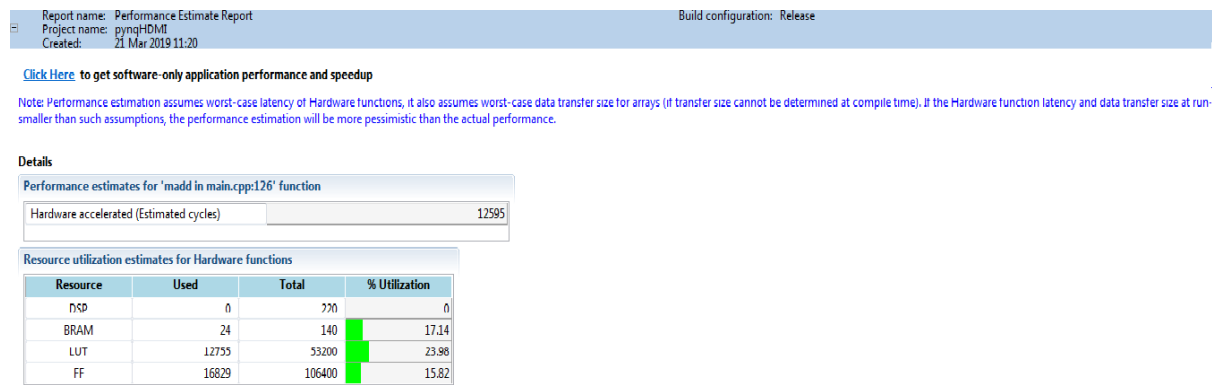


Figure 5.6: SDSoC result

Chapter 6

Conclusion and Future Work

In this project, I conclude about the progress of the project and states that in this project I have studied and practiced several neural networks as well as several platforms. In this I have found that there was no prebuilt platform available to be work with SDSoC tool and there was very few neural networks which was implemented on FPGA but those neural network algorithms are very hard to train and to re-implement on FPGA. Hence I had trained a pre-built neural network which is Yolo and tried to implement this network on FPGA by creating its IP. I had used Xilinx tools and Xilinx FPGA board which was introduced by Digilent. I have used this custom created IP in Vivado and Created Overlay by generating several files (.bit and .tcl). I had also used Vivado to generate .dsa file which was used in Xilinx SDK tool. This tool generated boot files and other useful files to be used in SDSoC in order to create a platform to be used in SDSoC for other projects and which can help other to work with. This platform contains Yolo IP Block as well as HDMI IP block along with Audio block. Hence this platform can be used in neural network based detection using cameras. These cameras can be connected directly with the FPGA board using HDMI In and OUT ports of the board. In addition to that this platform can also be used in processing on audios.

This project can be used as future work for those who wants to use the created platform in SDSoC or they can modify its Vivado Design according to their need. Yolo is a small Convolutional Neural network available in this platform only with one IP block but as a future work this IP block can be modified by updating its weights and training or the design can be stretched by creating multiple IP blocks according to the requirement. Another future work can be done by modifying this platform by training and implementing BNN in platform.

Bibliography

Papers

1. Byeongho Heo; Kimin Yun; Jin Young Choi; “Appearance and Motion Based Deep Learning Architecture for Moving Object Detection in Moving Camera”, IEEE International Conference on Image Processing (ICIP) 2017.
2. Ozge Mercanoglu Sincan, Vahid Babaei Ajabshir, Hacer Yalim Keles, Suleyman Tosun; “Moving object detection by a mounted moving camera”; IEEE EUROCON 2015.
3. Wang Zhiqiang, Liu Jun; “A review of object detection based on convolutional neural network”; 36th Chinese Control Conference (CCC) 2017.
4. Ola Younis, Waleed Al-Nuaimy, Fiona Rowe, Mohammad H. Alomari; “Real-time Detection of wearable Camera Motion Using Optical Flow”; IEEE Congress on Evolutionary Computation, At Rio de Janeiro, BRAZIL 2018.
5. Kimin Tun, Jongin Lim, Jin young Choi; “Situation-aware Framework From Moving object Detection in moving camera”; 8th International Conference of Pattern Recognition Systems (ICPRS) 2017.
6. Deng-Yuan Huang, Chao-Ho Chen, Tsong-Yi Chen, Jian-He Wu, Chien-Chuan Ko; “Real-Time Face Detection Using a Moving Camera”; 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA) 2018.
7. Sudhir K. Ingole, Shital B. Gundre; “Character Feature based Indian Vehicle License Plate Detection and Recognition”; International Conference on Intelligent Computing and Control (I2C2) 2017.
8. Won Jin Kim, In-So Kweon; “Moving Object Detection and Tracking From Moving Camera”; 8th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI) 2011.
9. Tao Cheng, Shijan Lu; “Object-Level Motion Detection From Moving Cameras”; IEEE Transactions on Circuits and Systems for Video Technology, Nov. 2017.
10. Sachin Prabhu B., Subramaniam Kalambur, Dinkar Sitaram; “Recognition Of Indian License Plate Number from Live Stream Videos”; International Conference on Advances in Computing, Communications and Informatics (ICACCI) 2017.
11. Trevor Hastie, Robert Tibshirani, and Jerome Friedman. “Unsupervised learning”. The elements of statistical learning. Springer, 2009, pp. 485–585.
12. Ling Hu, Qiang Ni; “IOT-Driven Automated Object Detection Algorithm for Urban Surveillance Systems in Smart Cities”; IEEE Internet of Things Journal, April 2018.
13. Rajesh Kumar Tripathi; Anand Singh Jalal ; Charul Bhatnagar; “A framework for abandoned object detection from video surveillance”-; Fourth National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG) 2013.
14. Waritchana Rakumthong ; Natpaphat Phetcharaladakun; Wichuda wealveerakup; N. Kamnoonwatana; “Unattended and stolen object detection based on relocating of existing object”; 2014 Third ICT International Student Project Conference (ICT-ISPC)
15. B Yegnanarayana. Artificial neural networks. PHI Learning Pvt. Ltd., 2009.
16. Jichan Lee ; Sungsoo Lim ; Jun-Geon Kim; Bomin Kim ; Daeho Lee; “Moving Object detection using Background Subtraction and Motion Depth Detection in depth image sequences”; The 18th IEEE International Symposium on Consumer Electronics (ISCE) 2014.

17. Arthur L Samuel. "Some studies in machine learning using the game of checkers"; IBM Journal of research and development (1959).
18. Yuanyuan Wu; Xiaohai He; "Moving Object Detection With a Freely Moving Camera via Background Motion Subtraction"; Truong Q. Nguyen; IEEE Transactions on Circuits and Systems for Video Technology 2017.
19. Jing Ma, Li Chen, Zhiyong Gao; "Hardware Implementation and Optimization of Tiny-YOLO Network"; International Forum on Digital TV and Wireless Multimedia Communications (IFTC) 2017.

Websites

20. <http://www.pynq.io/>
21. <https://software.intel.com/en-us/articles/binary-neural-networks/>
22. www.quora.com
23. <https://stackoverflow.com/>

Books

24. Reza Bosagh Zadeh, Bharath Ramsundar (2018), Tensorflow for Deep Learning; O'Reilly Media Inc.