Development of State of the Art AI Vision Algorithm on Xilinx Alveo U-200 FPGA Cloud and CPU+GPU Platform

Vigneshwaren Sunder
EEE Dept,
University of Nottingham

Abstract:- Since the start of modern computer algorithm practices, a lot of new techniques on computer vision has been developed. This results in utilizing machine learning algorithms and has provided attributes to the evolution of convolutional neural networks for building state-of-the-art object detection, segmentation and classification algorithms. These CNN can achieve human-like results in computer vision application, however with the expense of more computation. To meet machine learning application requirements on hardware deployments, various AI-Accelerated FPGA development kits have been developed along with specialized toolkits aimed at efficient optimization and deployment of the models. In theory, the FPGA solutions can have similar accuracy, better inference time and power consumption compared to the GPUs, however, it comes at the cost of limited CNN model support and additional FPGA hardware design complexity. In this thesis, an existing object detection algorithm has been studied and real-time simulation of the object detection algorithm which works under a darknet framework utilizing both CPU+GPU efficiently using CUDA by Nvida. Implementation of GoogleNet and ResNet50 object detection algorithm on a cloud-based FPGA platform using Xilinx Vitis-AI Toolkit has been carried out. The tools utilize different strategies like model quantization and hardware architecture set up to achieve an accuracy similar to a GPU with at least 10% difference. A broad case study on hardware and software configurations made on Xilinx ALVEO U-200 FPGA for efficient deployment via the cloud has been carried out. Results of both the simulation platforms have been compared and discussed for further optimization and developments.

Keywords- Machine Learning, AI Vision, State-of-the-art-CNN, CNN on FPGA, Hardware Accelerators for CNN, Computer Vision using Artificial Intelligence.

I. INTRODUCTION

Early applications of computer vision involve replicating the vision seen by the computers to be visible to humans. This includes the applications of cameras, x-ray images, color detection etc., early experiments in computer vision started in the 1950s and it soon found its application to use for commercial purpose mainly to distinguish between typed and handwritten text by the 1970s. The term artificial neural network was first coined in the late 1950s [1]. It is a computational model of the animal neuron in brains. They were modelled to perform a higher computational task that conventional algorithms had little to no success in performing them. However, due to hardware limitation at that time, these ANN did not find its true potential. It was in the last decade, the driving force on ANN architecture modelling has paved its way for the improvements of CNN algorithms for computer vision application. Instead of using predefined features of ML algorithms, CNN (a class of DNN) can take advantage of the increasing hardware memory and computational capability to carry out large training rapidly improved computation capability to learn from huge training datasets resulting in notable recognition capability. After the year 2012 when the winning method of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) which used CNN to detect up to 60 classes [2]. Since then, CNN has been mainly used for computer vision applications. Thanks to the advancement of deep learning techniques for training, the AI-powered Computer vision can now read and perceive what’s seen by the computer and gives meaningful output with efficient prediction capabilities. Various CNN architectures have been developed since then to meet specific results in the field in terms of applications, hardware compatibilities and computational complexity. Popular ones include: LeNet, GoogLeNet, ResNet, Darknet and RCNN [3].

The FPGA can be considered as an array of millions of gates that can be reprogrammed to run a specific algorithm. They provide massive parallelism targeted for that specific algorithm by making use of its combination of logic gates (lookup tables), digital signal processors for multiplying matrices, static RAM for the temporary storage of result in the memory of those computations and switching blocks that let us control the connection between programmable blocks. Most of the FPGAs used in development usually comes with plugged in systems on chip (SoC) components; which includes CPUs, PCI Express, DMA connections and Ethernet controllers. These make the FPGA to be feasible and reliable in development stages unlike ASICs [7].

The project involves comparing the architectures of existing CNN algorithms and effective implementation on various platforms. In our project, we will be using Xilinx Alveo U200 accelerator cards via a cloud based FPGA HPC machine offered by Nimbix, Inc. This is the newest and up to date fastest deploying FPGA for AI vision applications. It is capable of delivering massive integration and capabilities with ASIC architectures, system level performance massive I/O, memory bandwidth, data flow, advanced DSP and packet-processing performance. We made use of Xilinx U200 FPGA as a hardware platform [9].
A. Artificial Neural Network:

An ANN is a computational model of a biological neuron. It is inspired by the early research made by D. O. Hebb in 1940s [17]. Though these ANN does not work as accurately and sensitive as biological neurons, it can replicate the same learning behaviour. It does some mathematical calculations to derive an output as the result of activation functions similar to that of a biological neuron’s action potential.

From the above figure, the input is fed to the dendrite to the nucleus. The nucleus triggers an electrochemical signal and is sent to synapses (axon terminal). This happens only when the aggregated electrochemical signal is higher than the synaptic threshold, it results in an electrochemical spike and moves down the axon to dendrites of other neurons.

B. Components of ANN:

The components of a typical neural network may be summarized as follows:

- **Connections:** They carry information signals that are processed by the neurons
- **Propagation function and network input:** They are responsible for receiving the output from a neuron and multiplying with a weight value. They convert the vector value of input signals into a scalar value. The network input is the output of the propagation function. These signals then move to the activation function

\[
\text{net}_j = \sum_{i} \theta_{ij}x_iw_{i,j}
\]

- **Activation Functions:** Based on the nature of our model, each neuron in the architecture has to be activated desirably. This can be made to happen when the neuronal input signal crosses a given threshold value and thereby making the neuron sensitive and starts to fire. Let \( j \) be a neuron with its unique threshold value \( \theta_{ij} \). The activation function can be written as

\[
a_j(t) = f_{\text{act}}(\text{net}_j(t), a_j(t-1), \theta_{ij})
\]

The activation function of most of the neurons in architecture is typically the same but with different threshold values. These functions play a major role in deriving the appropriate output from input data. Some of the common activation functions are:

- **Output function:** The output function of a neuron \( j \) calculates the values which are transferred to the other neurons connected to \( j \).
- **Learning Strategy:** Every model of the neural network has specialized learning strategies. It’s an algorithm which determines how the neural networks are trained and placed to get the desired output. We will discuss in detail about popular learning techniques used in the upcoming sections.

C. Inference vs Training:

It is essential to know the two phases when it comes to deploying ML models. These are Inference and Training.

- **Training:** This stage works on annotated datasets that deal with training the network to tune for desired output predictions. This phase gets generally implemented on typical deep learning algorithms like back-propagation which iterates and updates the parameters involved in the CNN. These parameters can be convolution weights to increase and tune the threshold value and thereby improving the prediction power of the network. One could make use of the available annotation tools to annotate manually with classes on the desired input sample images. This phase occurs at high computation power and is generally made to train for days or even weeks in high-end HPC machines. There are many open-source pre-trained weights available on the internet. These weights are trained to be deployed on various state-of-the-art models without having to perform a training phase again.

- **Inference:** This phase is generally the result of the training phase. The model in this phase can now predict the class of completely new input that were not used in the training phase. The inference phases are generally fast and are applicable for real-time object detection. Hence, to meet these needs, industries tend to use cloud machines to carry out the training phase and update the pre-trained weights on the prototype edge machine for inference.

D. CONVOLUTION NEURAL NETWORK:

The idea of CNN was introduced long back for hand-written digit recognition by [LeCun et al., 1998]. CNN, a class of DNN, is a feed-forward ANN that are applied to analyzing computer vision. The mathematical definition of convolution is the operation on two functions that together produce an output function expressing how the signal fed is modified during the path. Images are high dimensional vectors. These are computed in the form of matrices and would result in large number of parameters being. To address this, new CNN architectures were introduced which are composed of batch of computational layers through which the input pixel values are made to pass through based
on their functionalities. It involves processing the pixels of the images by forming a special matrix.

![Diagram of CNN architectures](image)

**FIG 2: STATE-OF-THE-ART CNN ARCHITECTURES**

**E. Building Blocks of CNN:**

In this section, we shall look at the basic building blocks of CNNs in general. There are many states of the art CNN architectures available which we will discuss in the next chapter. A typical CNN network contains the following:

**Convolution Layer**

This Process is a 2D convolution to the input pixels. The pixels of each input, filter and its output are mapped as given in the figure below. The filter has the same number of layers as input volume channels, and output volume has the same depth as the number of filters. Filters play a major role to derive a particular feature from an image. For instance, one could use filters to detect horizontal or vertical lines lying on a specific range of matrices. Using filters we could make pixel colours that are not necessary during the learning process.

**Activation Layer**

It is used to improve the performance of the model in non-linear functions without affecting or degrading the computational efficiency of convolutional layers. From the many available activation functions, for CNN the most favourable one is ReLu as it results in faster training. Leaky ReLU is also used as it can address the vanishing gradient problem.

**Pooling Layer**

These layers are common to be added after conv layers in order to minimize the spatial size of the representation. This can enable us to reduce the number of parameters used in the computation and thereby can increase the efficiency. We do this by selecting maximum average or sum of a batch of pixels. This can reduce the overfitting problem. Pooling layers apply a non-linear desampling on activation maps in order to use smaller filter kernel size and reduce the pixel matrix dimensions.

**Fully Connected Layer and Loss Layer:**

It is a typical neural network architecture which maps the extracted features from the sample images to the desired output. The objective of this layer is to take the result of conv and pooling layer and map it into a label. The output from the previous layers is flattened in a single vector representing the probability of each feature’s property. It is then passed to a softmax to give confidence.

The Loss layer is an optimization layer used in training. It adjusts the current weights by using a cost function which we will discuss in detail how neurons are trained section.

**II. STATE-OF-THE-ART CNN ALGORITHMS:**

Using the building blocks of a CNN network, several efficiency states of the art easily deployable CNN algorithms were made for computer vision applications. We will look into few well known CNN models and perform simulations on CPU+GPU and FPGA platforms. LeNet was the first state-of-the-art CNN architecture is introduced by Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner in 1998 [12]. It was mainly trained to recognize handwritten digits and document recognition. The LeNet-5 architecture is quite simple and is illustrated in the below block diagram.

The LeNet-5 architecture consists of two sets of convolutional and average pooling layers, followed by a flattening convolutional layer, then two fully-connected layers and finally a softmax classifier. Since the last decade, a lot of new efficient CNN models were developed. The figure below shows the remarkable models’ architectures.

**A. AlexNet:**

In 2015, AlexNet was outperformed by Microsoft's very deep CNN with over 100 layers, which won the ImageNet 2015 contest [13]. The AlexNet architecture consists of 8 layers: 5 convolution layers with each followed by pooling or 3 fully connected layers. Some notable features of AlexNet are as follows. The use of non-linear ReLU activation function showed significant improvement and was able to less 25% error in training performance over tanh and sigmoid activation function.

The pooling layer in AlexNet makes use of overlapping to shoot outputs to the neighbouring neurons. This showed less error while training and avoided the chances of overfitting. This is because when overlapping are not used, the output from various occurrences tend to be similar to the previous iteration. This causes overfitting. A typical model has over 60million weight parameters after training. AlexNet allows us to train the neurons by using multiple GPUs. It can significantly reduce the training time and can accommodate huge models.

**Advantages:**

- The use of 3 fully connected layers enables us to achieve good accuracy at the end as the parameters are finely tuned in the network.
- It can achieve 94.5% accuracy on imagenet dataset.
- It is able to off-centre objects and its detected classes on an inference of image are reasonable.
GoogLeNet: 

GoogLeNet is an upgrade of LeNet architecture that is focussed on easy deploying on GPUs [8]. It is very inefficient to have feedforward architecture to manipulate input matrices with 512 channels. A 3x3 kernel filter will have an unimaginably dense connection in its architecture when convoluting images with more channels. To address this, GoogLeNet proposed an architecture which has a sparse connection between the activations which results in vanishing some not important connection in 512 input channels. It came up with a model which is popularly known as Inception architecture, built with more than 1000 sample images in a dataset, the width and number of conv layers and also the kernel size of these layers are made small. They are of 5x5, 3x3 and 1x1 in dimensions. The use of 1x1 filters showed great influence in improving the overall model training accuracy. Let's take an example input having 192 channels. The inception model generally has a 3x3 kernel size of 128 filters and 5x5 kernel size of 32 filters. Convoluting with these dimensions over 25x32x192 input image dimensions will result in too deep a network. To avoid this, 1x1 conv is used to reduce the dimensions of the input channels. So 1x1 kernel sizes of 16 filters are introduced before feeding into the 5x5 filters.

Advantages:

- The use of fully connected layers along with simple global average pooling layer which can average out the channel values in the 2D feature map after the last convolution layer can reduce the number of parameters and hence improving memory and computation efficiency.
- It achieves 93.3% accuracy on ImageNet and is found to be faster than VGG.
- The entire model is less sized ~100MB whereas VGG is ~500MB

Disadvantage:

- There aren't any proposed disadvantages to this model. Moreover, the newer version like Xception Network showed
improved performance in terms of accuracy and speed

E. ResNet50:

ResNet or Residual network contains 50 deep neural networks [4]. The main objective of the CNN architectures are to minimize the gradient of the function over a period of learning time. The gradients are updated during backpropagation. As we discussed earlier, the backpropagation takes 2 paths while transmitting the gradient back to the input layer. These identity shortcuts can be used when the input and output have the same direction.

\[ y = F(x, \{W_l\}) + x. \]

Where \( F(x, W_l) \) is the residual mapping to be learned

\( y \) is the output and \( x \) is the input layers considered in the network.

In the case of non-identical dimensions, extra zero paddings are introduced in excess matrices.

\[ y = F(x, \{W_l\}) + W_s x. \]

Here \( W_s \) linear projection matrix is introduced as extra paddings.

Advantages:

- The use of bottleneck layers in this framework has shown significant efficiency in the performance
- The model size is small ~102MB. This is because of replacing fully connected layers with global average pooling.

Disadvantage:

- The number of layers is too deep. Every percentage of improvement results in an increase in computation, memory cost and time taken to train.

F. State of the art Object Detection Algorithms:

With the above models, we can easily perform the classification of different classes. The computer vision application involves Classification, Object Detection and Multiple object detection with localization. The latter can be achieved using a single shot math action that doesn’t involve deep learning layers.

However, with deep learning layer’s architecture, we could achieve better accuracy with drastically increased computation, but in our thesis, we will be demonstrating the YOLO object detection which works under single shot bounding box and object detection. Other popular models like SSD, RCNN and Faster RCNN also exists but we will not be discussing it.

But when it comes to creating a bounding box of multiple objects within a given image, we need to perform an efficient algorithm that can be defined as follows.

YOLO’s Object Detection Approach

First, the image is cut into various equally spaced grids (SxS size), and for each grid, it detects bounding box and the corresponding classes by an output vector of dimension \( S^2 S(C+5) \), \( C \) is the number of classes. Then a single convolutional layer is used to perform the classification for that grid. This layer has a typical loss function as an error between the label vector and output activation. All the classes are generally fed forward into the same conv layer to define the parameters.

![Fig 24: Object Detection Mapping on Grid Sample](image)

To understand the workflow, let’s consider the grids to be 3x3 capable of detecting 3 classes namely, Cars, pedestrians and cycles. The label \( y \) is a vector consisting of an eight-dimensional vector.

And when an object is placed in multiple grids, the algorithm has to be modified in a way that it can detect objects. We introduce anchor boxes that define the ground truth of the object. Below, a midpoint of the anchor box is introduced. Intersection over Union (IoU) computes the intersection over the union between the actual bounding box and the predicted one. IoU = Area of the yellow box (Intersection) / Area of the green box (Union) Then the prediction is good enough if the IoU is greater than a threshold.
Fig 4: Mapping the boundary box on sample image[17]

The with the output box parameter values, we can define the score for the probability of the class.

\[
\text{scores} = \begin{bmatrix}
    0 & 0 & 0.12 \\
    0 & 0.13 & 0.44 \\
    0.07 & 0.01 & 0.09 \\
\end{bmatrix}
\]

Find the max score: 0.44

Box: \( (b_x, b_y, b_w, b_h) \)

Class: \( c = 3 \) (car)

This object detection technique is almost the same for other models too, the only change happens in the deep neural network layers used in inference. Yolo has 24 conv layers and then 2 FC layers. Yolo uses some 1x1 conv layers to reduce the depth of the reduction layers. The last 7x7x1024 deep conv layer is then flattened and fed into those 2 FC layers and the last conv layer of output 7x7x30. It uses 3 loss function. One is for classification, another is for localization to define the error between the predicted boundary box and ground truth, and the confident loss for object detection.

Advantages:

- Yolo provides faster real-time inference with about 45 FPS which is currently the fastest one so far.
- The use of mean average precision (mAP) one single layer to predict the bounding boxes has led to lightweight deployment for inference.
- It provides easy to use pre-trained weight files. Moreover, it is built on the Darknet framework which is rewritten using CUDA and TensorFlow thus making it efficient to train the model using the home desktop with full GPU efficiency.

III. HARDWARE AS AI ACCELERATORS

CNN network needs large datasets and requires huge runtime, maybe weeks or even months if the network is trained for more classes. To address this, the use of dedicated hardware came to existence. AI Accelerators are specialized hardware designed to run algorithms that require parallel operations to reduce the time complexity problems. Computers traditionally use a special-purpose processor to rely on along with a general CPU. This includes GPU, video cards, TPU and FPGAs. During training, the process involves parallel computation with the massive data flow of FP operation that involves matrix and vector operations.

A. DL Frameworks:

A deep learning framework is an interface, library or a tool which allows us to build deep learning models more easily and quickly, without getting into the details of underlying algorithms. They provide a clear and concise way for defining models using a collection of pre-built and optimized components. It enables users to deploy a large number of layers parameters and provide parallelism to the algorithm by just using these DL frameworks. These are optimized for better performance and reliability, easy to understand, provide parallelism, and above all, there are many frameworks available and are open-sourced. So users can code ML algorithms in the programming languages they like. In the following section, we will look at some popular open-GPU as Accelerators

Throughput optimization: The FPGA can process the dataflow in a parallel flow. In throughput optimization, this flow is exploited to handle mapping of data that are inefficient to a general sophisticated array. In short, if the computation process of a particular stage of the network is large, it can be mapped in dedicated array memory to handle it and thereby reducing the complexity of memory handling and time taken to process it.

Latency optimization: Some applications require to infer the lowest single-image latency. In these situations, the xDNN processing pipeline can be easily configured. We have demonstrated the CNN models performance in both throughput and latency modes.

The Xilinx DNN engine in the Vitis AI makes it easy and reliable for production. It has specific execution paths for performing convolutions, pooling, padding, adding other filters for various frameworks. As parallelism is one of our primary motives, a new filter or input can be fed parallel to the network graph enabling adaptable parallel processing. As the conventional model made using CPU+GPU frameworks and languages has to be converted to optimize FPGA architecture, the Xilinx Vitis AI SDK stack has its own compiler stacks namely:

Network Compiler. This compiler produces the network instruction sequence and is being executed. It enables implementing data-flow management and tensor-level control to the network.

Model Quantizer. These enable the computation complexity of the model to reduce effectively and it provides a significant improvement in performance in terms of data memory management. It converts 32-bit floating-point FP32
weight and activations to fixed point INT8. A fixed point integer data type in the network can improve the speed and power efficiency by requiring less memory bandwidth. It takes the frozen graph for the TensorFlow framework and performs pre-processing by performing batch normalization and removing unwanted nodes. The activations, weights and biases and then quantized to a given bit size. It must be noted that not all models can perform model quantizer and its application is limited. Standard versions of Caffe and TensorFlow framework cannot be quantized using this and has to be configured separately.

**AI Compiler** - It works like any other typical compiler by mapping the model in the form of the instruction set and data flow. It performs sophisticated optimizations like instruction scheduling, layer fusion and on-chip memory management. It can analyse the data and allocate optimum location for data onto the memory and enables reusing on-chip memory.

**AI Library** - The library contains high-level libraries and APIs for AI inference with DPU. This plays a major role by enabling users who don’t have prior knowledge of FPGA architecture to implement and debug the code. The xDNN Runtime provides reusable programs, and the scheduler simplifies the communication and programming of the DPU engine. Its runtime also makes cloud-to-edge deployment more feasible.

IV. IMPLEMENTATION AND RESULTS

A. Simulations on CPU+GPU

All the simulations here are performed using ipython notebooks on google collab platform. We made use of Fashion MNIST dataset with 10 classes. The models are capable of detecting more classes, but since it takes more time, we limited to 10. All the models were trained using 64 batches, 10 epochs and with learning rate at 0.01.

1) Alexnet Results

![FIG 5: ALEXNET TRAINING RESULT GRAPH](image)

We have used 5 conv layers and 3 fully connected layers (represented as dense n layers) to build the architecture of the Alexnet model. The input image to Alexnet model is a 256x256x3 RGB dimension. Our original dimensions of fashion MNIST dataset is 26x26. So we upscale its dimensions by 224x224. The first conv layer contains 96 kernels of size 11x11x3, which we defined in our code. The filter process in 54x54 channels. These channels are calculated using the term

\[
(input\text{dimension} - conv\ layer\ dimension)/\ stride + 1.
\]

The same calculations go for all the other layers. The use of multiple kernels of the same size in conv layers can extract very deep features from the image. There are 2 max-pooling layers after the first and second conv layers. And we added an overlapping max-pooling layer after the fifth conv layer. We applied ReLu nonlinearity after every conv layers and a local normalization is added after the first two convolution layers. The fully connected layers perform softmax classification. A detailed overview of the algorithm design and flow is given in the source code.

2) NiN Results

![FIG 6: NIN TRAINING RESULT GRAPH](image)

It can be seen that our NiN model is slightly similar to AlexNet because it is based on the Alexnet architecture but with slight improvements

The NiN uses 11x1, 5x5 and 3x3 conv layers and its channel numbers are identical to Alexnet. We added a max-pooling layer with stride:2 and kernel 3x3x3 size. The number of output channel is equal to the classes making the NiN denser and a global average pooling is performed to derive the vector logics. Since the model is dense we can see that it took more time to finish the training for the given hyperparameters. The fully connected layers use ReLU activation.
3) **VGG-11 Results**

In VGG, we added 5 sequential blocks, each having consecutive conv layers of 3x3 kernels and max-pooling of 2x2 of stride: 2 to minimize the resolution of the sample. The first conv layer has 64 output channels of kernel 112x112. The dimensions of the conv layer’s channel are increased by 2 and its kernel size is decreased by half. It gets to 7x7 before getting flattened and put to the fully connected layer

The performance of the VGG is computation-intensive, however, we got the maximum training accuracy for the same hyperparameters used in all other models.

4) **GoogleLeNet Results**

The first layer uses 64 kernels of 1x1 and 3x3 conv layers with 24 channels. The other layers use the same as given in the above figure. We placed 3 inception blocks in our network as given in the figure (in googlenet intro section). It consists of 4 parallel paths. The first 3 paths have a conv layer of dimensions 1x11x1,3x33, and 5x5x5 to extract features in different special size. The middle paths perform the 1x11x1 conv to reduce the model complexity by vanishing the input channels. The fourth path uses 3x3 max-pooling along with 1x11x1 conv layer to modify the channels. A detailed explanation about code design and flow in the source code.

5) **Resnet-18 Results**

The resnet uses a special type of function as given below to define the learning rate and other hyperparameters. For all \( f \in \mathbb{F} \) exists weight \( W \). The function given \( f^*F \) which is our best fit for the function \( F \) and it is the base for determining the residual blocks used in the network.

\[
 f^*F := \text{argmin}_f L(X, Y, f) \text{ subject to } f \in \mathbb{F}.
\]

The first two layers are similar to google net except for the fact that resnet has 4 blocks of residual function modules. Each module contains 1x11x1 conv layer. The max-pooling layer here is used to reduce the dimensions of the sample. It can be seen that the width and height of the output from each residual modules have been halved but the number of kernels has been doubled. Due to this, we can achieve low training loss in less time when compared to other models. We can easily create other Resnet models like Resnet-152 or Resnet-50 by changing the residual blocks and make the network more deeper to detect more classes effectively. We will look in detail about the code design and flow in our source code.

6) **Comparison of the above Models**

| MODEL | Loss | Train Acc | Test Acc | Neurf Layers |
|-------|------|-----------|----------|--------------|
| AlexNet | 0.280 | 0.899 | 0.897 | 13 |
| NIN | 0.584 | 0.781 | 0.797 | 15 |
| VGG | 0.172 | 0.597 | 0.928 | 21 |
| GoogleNet | 0.33 | 0.876 | 0.874 | R+parallelLayers |
| ResNet | 0.026 | 0.992 | 0.927 | 18 (50 people) |

7) **Yolo Object Detection Result**

We have performed an object detection algorithm on the personal laptop with intel i5 CPU+2G Nvidia MX150 integrated GPU+8G RAM. We were able to get real-time
inference at less than 20 FPS. The model claims to provide a better result with higher system configurations.

The above figure shows the output that we fed to the network as input images. The model was able to map and detect almost all the objects present in the image.

B. Simulations on Cloud-based ALVEO-U200/U50 FPGA

1) GoogLeNet Result

The inference of the given input image is shown above after passing 106 DNN layers. 75 convolution layers and the remaining 31 includes pooling, FCN and other layers. A detailed explanation of code design and flow is given in the source code. Three objects were detected in this image: dog, person and horse. A non-max suppression is used to detect multiple objects with mAP. It takes the list and calculates for the images with the scores more than a predefined threshold. Then the intersection of union and confident threshold was responsible for making the bounding boxes and detect the object. We have used the pre-trained weights file available from the Yolo darknet’s official website which is capable of detecting up to 60 classes. We have performed a real-time demonstration using YOLO algorithm and was able to provide real-time detection at around 20 FPS which can be further improved by using better CPU+GPU platform.

The above figure shows the output that we fed to the network as input images. The model was able to map and detect almost all the objects present in the image.

B. Simulations on Cloud-based ALVEO-U200/U50 FPGA

1) GoogLeNet Result

The inference of the given input image is shown above after passing 106 DNN layers. 75 convolution layers and the remaining 31 includes pooling, FCN and other layers. A detailed explanation of code design and flow is given in the source code. Three objects were detected in this image: dog, person and horse. A non-max suppression is used to detect multiple objects with mAP. It takes the list and calculates for the images with the scores more than a predefined threshold. Then the intersection of union and confident threshold was responsible for making the bounding boxes and detect the object. We have used the pre-trained weights file available from the Yolo darknet’s official website which is capable of detecting up to 60 classes. We have performed a real-time demonstration using YOLO algorithm and was able to provide real-time detection at around 20 FPS which can be further improved by using better CPU+GPU platform.

The above figure shows the output that we fed to the network as input images. The model was able to map and detect almost all the objects present in the image.

B. Simulations on Cloud-based ALVEO-U200/U50 FPGA

1) GoogLeNet Result

The inference of the given input image is shown above after passing 106 DNN layers. 75 convolution layers and the remaining 31 includes pooling, FCN and other layers. A detailed explanation of code design and flow is given in the source code. Three objects were detected in this image: dog, person and horse. A non-max suppression is used to detect multiple objects with mAP. It takes the list and calculates for the images with the scores more than a predefined threshold. Then the intersection of union and confident threshold was responsible for making the bounding boxes and detect the object. We have used the pre-trained weights file available from the Yolo darknet’s official website which is capable of detecting up to 60 classes. We have performed a real-time demonstration using YOLO algorithm and was able to provide real-time detection at around 20 FPS which can be further improved by using better CPU+GPU platform.

The above figure shows the output that we fed to the network as input images. The model was able to map and detect almost all the objects present in the image.

B. Simulations on Cloud-based ALVEO-U200/U50 FPGA

1) GoogLeNet Result

The inference of the given input image is shown above after passing 106 DNN layers. 75 convolution layers and the remaining 31 includes pooling, FCN and other layers. A detailed explanation of code design and flow is given in the source code. Three objects were detected in this image: dog, person and horse. A non-max suppression is used to detect multiple objects with mAP. It takes the list and calculates for the images with the scores more than a predefined threshold. Then the intersection of union and confident threshold was responsible for making the bounding boxes and detect the object. We have used the pre-trained weights file available from the Yolo darknet’s official website which is capable of detecting up to 60 classes. We have performed a real-time demonstration using YOLO algorithm and was able to provide real-time detection at around 20 FPS which can be further improved by using better CPU+GPU platform.
2) ResNet Results:

We have performed an inference simulation for a set of sample inputs and the output is predicted with prediction probabilities.

| Stage                  | INT8        | INT16       |
|------------------------|-------------|-------------|
| After Passing FP-GCA   | 86.399 ms   | 50.316 ms   |
| After Passing FC Layer  | 0.994 ms    | 0.264 ms    |
| After Passing Softmax Function to give classification | 0.116 ms | 0.0729 ms |

3) Simulation of Yolov3 on ALVEO_U50

We have performed an inference simulation for a set of sample inputs and the output is predicted with prediction probabilities. A set of 10 sample images has been fed and each image may have multiple objects in it. The YOLO algorithm can predict all the objects in a given image with prediction probability. The time taken to complete the inference was not derived. Some configurations have to be made in source code to print the time taken. This Xilinx U50 FPGA was recently launched in May 2020 and I didn’t have enough time to study the flow of the source code.

V. CONCLUSION, DISCUSSIONS AND FUTURE WORKS:

Thus this project has accomplished the primary objective that is to run the object detection algorithm on FPGA. Many reported successes of deep learning algorithms for computer vision tasks have motivated the development of hardware implementations of CNNs. In particular, there has been increased interest in FPGAs as a platform to accelerate the post-training inference computations of CNNs. To achieve high performance and low energy cost, a CNN accelerator must 1) fully utilize the limited computing resources to maximize the parallelism, 2) exploit data locality by saving only the required data in on-chip buffers to minimize the cost of hardware memory management, and 3) manage the data storage patterns in the hardware to increase the data reuse.

The state-of-the-art CNN algorithms are studied and demonstrated for different hardware platforms. We have also discussed in detail about hardware architecture of ALVEO U200/U50 FPGA board that gives an efficient design method tools for optimum performance. We also trained 5 CNN architecture, analyzed its performance and plotted the efficiency graph using ipython notebooks by google cloud platform. The dataset used for training is MNIST Fashion object detection dataset which has about 10 classes. Furthermore, the design flow to prepare a dataset from scratch and considerations to develop and train our own CNN model has been given along with this work. Newly updated CNN model, what is not added in this report, applications and future works to improve our study. Although ASICs provide better inference, we can see that the FPGA is still preferred because of its portability and reliability.

Future Works: We were not able to achieve deploying the algorithm on Edge Platform due to version error, lack of support on FPGA hardware and time constraints. The future works may include deploying it on board and controlling actuators based on the inferred inputs as FPGAs can be programmed to do anything, it can be easily deployed in mobile devices. A more efficient and robust object detection algorithm called YOLOv4 was recently released in early march which claims that it can be...
deployed on mobile phones using PC hardware for inference via wireless connections. This has lots of scopes in terms of education, GPS guidance and even on cooking. With the growing 5G Technology, the application for real-time inference can find its true potential enabling massive wireless data transfer with the grid and can provide a robust incremental learning environment. We can use this work to study the power consumed on each platform along with its time taken and efficiency values.

VI. REFERENCES

[1] Richard Szeliski. “Computer Vision: Algorithms and Applications.” Springer Science & Business Media, pp. 10-16, 2010.

[2] Smirnov, E., Timoshenko, D. and Andrianov, S., 2014. Comparison of Regularization Methods for ImageNet Classification with Deep Convolutional Neural Networks. AASRI Procedia, 6, pp.89-94.

[3] Anurag Arnab, Ondrej Miksik, and Philip H. S. Torr. On the robustness of semantic segmentation models to adversarial attacks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018.

[4] K. He, X. Zhang, S. Ren and J. Sun. "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[5] A. Kong and B. Zhao. "A High Efficient Architecture for Convolution Neural Network Accelerator," 2019 2nd International Conference on Intelligent Autonomous Systems (ICoIAS), Singapore, Singapore, 2019, pp. 131-134, doi: 10.1109/ICoIAS.2019.00029.

[6] A. Kyriakos, V. Kitsakis, A. Louropoulos, E. Papatheofanous, I. Patraons and D. Reiss, "High Performance Accelerator for CNN Applications," 2019 29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), Rhodes, Greece, 2019, pp. 135-140, doi: 10.1109/PATMOS.2019.8862166.

[7] C. Hao et al., "A Hybrid GPU + FPGA System Design for Autonomous Driving Cars," 2019 IEEE International Workshop on Signal Processing Systems (SiPS), Nanjing, China, 2019, pp. 121-126, doi: 10.1109/SiPS47522.2019.9020540.

[8] P. Aswathy, Siddhartha and D. Mishra, "Deep GoogLeNet Features for Visual Object Tracking," 2018 IEEE 13th International Conference on Industrial and Information Systems (ICIS), Rupnagar, India, 2018, pp. 60-66, doi: 10.1109/ICINFS.2018.8721317.

[9] Bishop, C. M. (2006), Pattern Recognition and Machine Learning, Springer, ISBN 978-0-387-31073-2

[10] Min Lin, Qiang Chen, Shuicheng Yan. “Network In Network” arXiv:1312.4400.

[11] Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. (1998). "Gradient-based learning applied to document recognition" (PDF). Proceedings of the IEEE. 86 (11): 2278–2324. doi:10.1109/5.726791.

[12] Krizhevsky, Alex, Ilya Sutskever and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks.” NIPS (2012).

[13] Min Lin, Qiang Chen, Shuicheng Yan. “Network In Network” arXiv:1312.4400.

[14] S. Gu and L. Ding. "A Complex-Valued VGG Network Based Deep Learning Algorithm for Image Recognition." 2018 Ninth International Conference on Intelligent Control and Information Processing (ICICIIP), Wanzhou, 2018, pp. 340-343, doi: 10.1109/ICICIIP.2018.8606702.

[15] M. F. Haque, H. Lim and D. Kang. "Object Detection Based on VGG with ResNet Network." 2019 International Conference on Electronics, Information, and Communication (ICEIC), Auckland, New Zealand, 2019, pp. 1-3, doi: 10.23919/ELINFOCOM.2019.8706476.

[16] [17]. Hebb, Donald (1949), “The Organization of Behavior.” New York: Wiley. ISBN 978-1-135-63190-1.