Fixed Point quantization in ResNets for Image Recognition

Arnab Bhattacharya
ECE Department
State University of New York at Stony Brook
Stony Brook, USA
bhattacharya.arnab@stonybrook.edu

Anirban Bhattacharya
ECE Department
State University of New York at Stony Brook
Stony Brook, USA
anirban.bhattacharya@stonybrook.edu

I. Introduction

Traditionally, Convolutional Neural Networks have been used in image recognition problems with very high accuracy. However, the past few years have shown Residual Neural Networks(ResNets) to be a highly accurate structure for such applications. Research continues in the field in order to build more and more robust image recognition frameworks. Aided by the increase in processing power of the CPUs and ever increasing use of GPUs for Neural Network Processing, the complexities of the networks continue to increase dramatically. As these networks continue to grow and increase in complexity, the number of floating point operations in the training process also increases manifold.

Floating point operations consume a lot of resources when implemented in FPGAs. The operations on the floating point numbers are not very straight forward. They are complex in designs and consume a lot of computational resources. Therefore, such networks therefore perform poorly when implemented on FPGA boards. Fixed point representations provide a good alternative to traditional floating point representations by reducing the complexity of the underlying hardware computations while having minimal effect on the accuracy of the parameters. There have been previous studies showing fixed point quantization like Uniform Width, Gaussian, Laplacian and Gamma yield comparable accuracies to floating point representations in Deep Convolutional Neural Networks.

In this project, we aim to study the effects of fixed point quantization of floating point numbers on the accuracy of ResNets. The quantization methods decrease the computational complexity but also give lesser accuracy. Optimal quantization method for a network would be one which reduces the computational complexity significantly without adversely affecting the accuracy of the network. We try to study this balance between the accuracy and the complexity of the neural networks through various quantization methods on established ResNet structures used in image recognition.

II. BACKGROUND

In recent times one topic has gained a lot of momentum in the Image Recognition domain. This has been the Residual Neural Network structure. The new improved neural network structure provides the model a huge scope to learn detailed features in the image data. This increase in the learning capability comes with an additional cost of increased calculations. In case of these networks it is specifically Floating Point Operations which increases a lot with increasing complexity of the network. It is exactly in this area that there is a huge scope of improvement. Floating Point operations are very complex to implement on the hardware and it is also very time consuming for the hardware to do these operations. We can improve these performances by using a different kind of number representation. We have explored the use of Fixed Point representation of the decimal numbers to decrease this operation load.

A. Fixed Point Representation

To represent decimal numbers as binary numbers we need to have some kind of representation. We cannot convert the numbers on the decimal scale with the decimal point directly into the binary scale. For this there are number of different representations. The most commonly used representation for this purpose is the Floating Point Representation.

There are two formats of floating point representation, the Single Precision Floating Point and Double Precision Floating Point representations. The floating point representations does not have decimal point. There are no representation of decimal point on the binary representation. Even the representation is such that the decimal point does not exist at a fixed location. It is changing and can be basically thought of to be floating in the places. Hence the name Floating Point.

Moreover the huge range of floating point representation is mostly unused for regular applications. Most of our applications do not encounter values in the wide range. Most of the values used in the regular computations are very much in a range that can be represented well with the number of bits available in the hardware. One place where the floating point serves well is the precision of the numbers represented. The high precision does help us in getting the value right to a greater precision than any other representation for same number of bits.

Fixed Point deals very well with the problem of huge unused range and complex hardware for operations. The fixed point representations of the decimal numbers have small range that is more completely used during our operations. The numbers used in most of our models are can be very easily represented in the range of the fixed point representation. The fixed point

representation of the numbers are represented as an integer part and a fraction part. It can be represented as Q[m:n], where m is the number of bits taken to represent the number of bits for the integer part of the number and n is the number of bits used to represent the fraction part of the number.

One advantage of using this form of representation of the numbers is the extremely simple and fast hardware designs. The fixed point representations have incredibly simpler arithmetic operations design when compared to the floating point representations. The simpler hardware in turn also makes the computation overall a bit faster. This faster and simpler operation hardware makes any design done with fixed point very efficient from the resources point of view.

The fixed point numbers can be represented in the format Q[m:n] where m is the number of bits used to represent the integer part of the number and n is the number of bits used to represent the decimal part of the number. We are looking at two fixed point formats for our system. We are looking at Q[8:8] and Q[16:16] formats of fixed point numbers for application on our system.

Fixed Point representation also has some limitations when compared to the floating point representation. The floating point has a distinct advantage when it comes to the precision of values. The floating point representation can give a very high precision for the numbers. The fixed point representation can come close to the precision of floating point if it uses a very high number of bits to represent the number. The decreased precision of the number will inevitably result in some quantization noise being created when we represent numbers as fixed point. This quantization noise and the value we are ready to tolerate in this quantization noise is what decides the representation choice when designing the hardware.

We are basically trying to trade off our accuracy of our computations for the high complexity of designs involved in the floating point representations.

B. Residual Neural Network

Residual Neural Network, or ResNet as they are called, are a variation of Convolution Neural Network that can give high accuracy to the neural network model that is being built. The ResNet addresses one very vital limitation of the Convolution Neural Network. The CNN has a problem of vanishing gradient which comes into the picture with increase in the depth of the neural network. The ResNet overcomes this limitation by putting shortcuts between layers. These shortcuts are sometimes called highway connections because they let the gradient value reach the deeper layers faster as compared to the conventional CNNs.

Mathematically the concept of shortcut connection can be represented as follows,

The CNN represent the data through a layer as,

$$y = f(x)$$

The residual neural network layer can be represented as,

$$y = f(x) + x$$

The structure of a representative 34 layer ResNet can be seen in figure 1. This structure of neural network allows the gradients calculated for the neural network during the training process to be directly passed on to deeper layers without having to go through the intermediate layers. In doing this it eliminates the situation where the gradients due to optimization operations start to get so small that they basically disappear completely. The non-disappearing gradients make the model capable of learning if the number of layers are increased. This efficacy of learning procedure allows us to pack as many layers as required to make the model highly accurate.

This model has gained a huge popularity in the Image Recognition domain because the increased number of layers that this model provides gives us a way to pack more layers in the model and introduce an opportunity for the model to learn more and more features. The images in themselves have a lot of features that can be learnt and that can help them in gaining important distinction from each other.

The Residual Neural network, as we can see in the figure 1 has standard convolution blocks which operate on the data. Combination of these convolution blocks makes the residual neural networks. The two convolution blocks are bypassed by the shortcut connections each time. These convolution blocks are designed such that they do not reduce the dimensions of the incoming planes. This makes it possible for us to add the X at the end of the residual block. These two blocks of convolution network and the bypassing branch of X makes one Residual Neural Network block. One thing that is important to notice in the figure 1 is there are two kinds of shortcut connections between the residual blocks of network. There is one represented by solid line which means that the X is directly added to the output after the two convolution layers. But the dotted line represents the connection for which the bypassing connection has its own convolution operation.

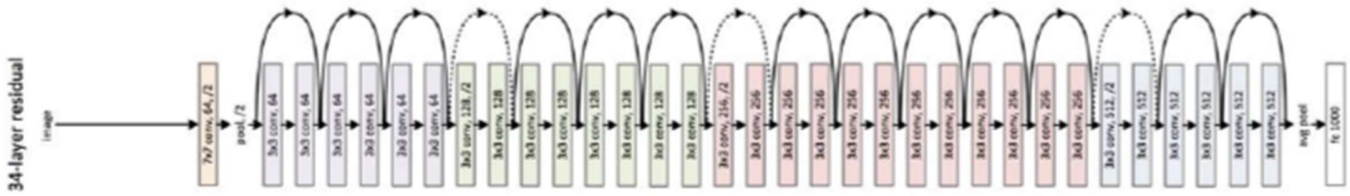

Figure 1: 34 Layer Residual Neural network [https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8]

The regular connections do not need a convolution in the bypass connection because the dimensions at the entry of the block is same as the dimension at the end of the block. This makes it possible for us to add X directly. But in case of the start of a layer where the dimensions of the input will be changed it is not possible to follow this suit. When the input channels need to be expanded and the input plane size needs to be reduced we need to carry out convolution on the bypass connection as well to make the addition at the end possible. As visible in the structure in figure 1 we can use as many number of blocks inside the ResNet structure. The dimensions of the input planes are never reduced between the blocks and hence we can add as many blocks as we want without ever fearing running out of input dimensions.

This structure of Residual Neural Network helps us in defining a new structure of Deep Neural Network which can is essence go very deep and calculate results with high accuracy.

III. IMPLEMENTATION

The implementation of our project is done in two parts. The first part being the ResNet model that we made and trained on Pytorch using the CIFAR-10 dataset. And the second part comes in the hardware accelerator built inside the FPGA of a Xilinx MiniZed board using the designs of CLP.

A. ResNet Model

We have used the Pytorch library available in python to design the deep learning network model. We have used the concept of residual Neural networks to design the model. We have used the 2-D Convolution method, 2-D Batch Normalization Method, Linear method and Shortcut method of the torch.nn module. We have also used 2-D Max Pooling Layer, 2-D Average Pooling Layer and Relu activation function from the torch.nn.Functional module. We have built a network with 14 Convolution layers in total.

Our model is made up of 3 residual layers. Each of them have two residual blocks. All the residual blocks in turn have 2 2-D Convolution layers. Each convolution layer is fitted with a Batch Normalization Layer to keep the values from accumulating to a very high number.

Figure 2 represents the structure of the residual neural network designed. The first layer of the model is a simple convolution layer with kernel size 3, zero padding of 1 and stride 1. This layer has input dimensions of 3X32X32 and an output size of 32X32X32. The output after this layer has 32 channels. The first residual layer does not change the dimensions of the model, hence the shortcut connection can be passed on directly on to be added to the output of the second convolution layer. The first residual layer will have two identical residual blocks. The residual blocks will each have two identical convolution layers. They both will have input and output dimensions of 32X32X32, kernel size as 3, stride as 1 and zero padding of 1. Each residual block will have the input from the start of the block being added to the output of the complete residual block before passed on to the input of the next residual block. The second and third residual layers will change the dimensions of the inputs. So for each of them the first residual block will be a bit different. The first residual block will be made of two convolution layers. The first layer will have kernel size 3, stride 2 and zero padding 1. The second convolution layer will have kernel size 3 stride 1 and zero padding 1. The first convolution layer in the block is the one that reduces the dimensions of the input data. The change in dimensions will mean that we will also have to run convolution on the shortcut connections. At last the network will have a layer of an average pooling with kernel size 4, followed by a max pooling layer with kernel size 4. This reduces the final output dimension from 128X8X8 to 128X1X1, which is just 128 numerical values. These values are then passed through a Linear layer with 128 parameters to give 10 final probability values for the 10 different classes.

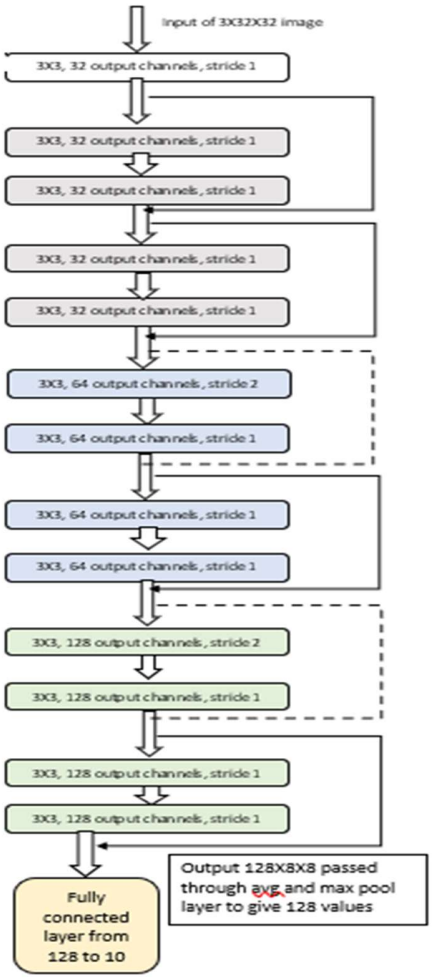

Figure 2: ResNet14 – A 14 convolution layer Residual Layer Neural Network Layer

B. Hardware Design

We have designed the hardware based on the Xilinx Minized FPGA Board. We implement CLP design on Xilinx Vivado to simulate convolutional neural networks and a C program on Xilinx SDK which communicates with it and also does a few of the computations for the model. In the following sections, we illustrate the design in a little more details.

Our design includes 2 CLP designs, one computing with a stride of 1 and another computing with a stride of 2. Each CLP

has a few memory blocks attached to it. In the case of Q[8:8] fixed point representation, we pack 2 16 bit integers into one 32-bit memory slot. We do the same for all the weight vectors also. Additionally, we have an internal buffer for storing the intermediate output values. It is further connected to an external buffer visible to the PS for storing the final output. The CLPs wait for the control signal from the PS. On receiving the control signal, the CLPs begin execution of the model. The PS also passes 2 signals to signify the start and end of calculation of a part of the output buffer. On receiving the end signal, the CLP transfers data from the invisible internal buffer to the visible external buffer from where it is read by the PS.

For the quantization of the data, we pick a format in the format of Q[n:n]. We then multiply the float number in question by 256 to shift the decimal place right by 8 places. Then, we cast the resulting floating point number to an integer value thereby producing an integer with the most significant 8 bits as the value before decimal place and the least significant 8 bits as the fractional part. We compute the results in CLPs and on retrieving the data back in our PS, we transform it back to a floating point and divide it by 256 to restore the exponent value back as it was before quantization. We also implement saturation in the quantization method. If the value exceeds the limits of signed integer on quantization, we just input the maximum or minimum representable integer number in place of the actual value.

C. Software Design

The PS implements the batch normalization, shortcut layer computations, max and average pooling as well as the last linear layer. The PS also controls the CLPs and does the data transfer to and from the CLPs.

The batch normalizations are implemented in the software by finding the mean of all the inputs in each layer of the network. This mean is then used to find the variance of the dataset and then compute the modified output values.

The shortcut layers are just convolutions with kernels of size 1X1. These layers are implemented fully in the software since they are just a accumulation of multiplications across all the output channels once over.

After going through all the convolutional layers, the data goes through an average pooling and max pooling layer to reduce the 8X8 output from the final convolution to a single element per channel. We implement both these layers simultaneously in the software level.

After that, we send the data through the final linear layer to convert the 128 channels into 10 elements corresponding to the 10 classes. Then, we just find the index of the max value among these 10 values to determine which class the data belongs to.

By going through all the layers, we compute the class of each data batch by batch. Then, we need to compare these values to the actual label passed with the inputs to determine whether our model predicted the right class for the inputs thereby identifying the image.

IV. EVALUATION

The hardware model is capable of running at the maximum frequency as given in table 1

Model	**Maximum Frequency Achieved**
Q[8:8] Fixed Point	85 MHz
Q[16:16] Fixed Point	75 MHz
Floating Point	77 MHz

Table 1: Maximum Frequency of the models as observed in Vivado

For a single layer of the convolution, the floating point accelerator runs in about 1.5 times the time that is taken by the fixed point accelerator. So the fixed point accelerator is definitely a faster model when compared to the floating point accelerator.

The accuracy achieved in the residual neural network model was close to 80%. The model is able to predict well for any given class but it does overfit the training data. But since the objective of the project was to see the performance of model on hardware we accepted the results and moved forward to the hardware section of the project.

V. CONCLUSION AND FUTURE WORK

There may be some bugs in the code remaining since we got stuck and did not get to the testing phase of the system in regards to the accuracy of the model on hardware.

VI. CONTRIBUTIONS

Work was divided among us in the following pattern,

Anirban Bhattacharya has worked on all python scripts and model training using the Pytorch library.

Arnab Bhattacharya has done all the work regarding the hardware design and Vivado project design.

REFERENCES

[1] Pablo Ruiz, Understanding and visualizing ResNets, https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8, October 2018

[2] Kuangliu-GitHub, https://github.com/kuangliu/pytorch-cifar/blob/9b0869d54827cee40e7e7130d6ea8f8a3976b7d4/models/resnet.py

[3] Peter Milder, Getting started with the Xilinx Zynq FPGA and Vivado, January 29, 2020