Memory optimization in Deep Neural Network for constrained hardwares

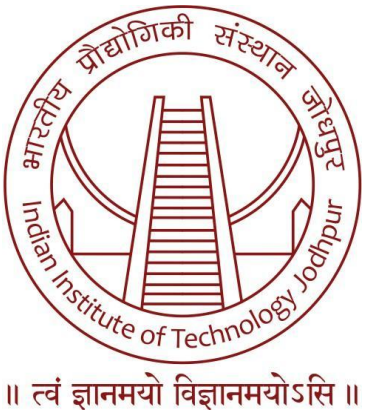
A Project Report Submitted by

Suyash Bansal

Sudip Kumar

in partial fulfillment of the requirements for the award of the degree of

B.Tech.



Indian Institute of Technology Jodhpur

Computer Science and Engineering

Dec, 2023

Declaration

We state the work which is presented here in this project report titled Memory Optimization in deep neural networks for constrained hardware devices is submitted to Indian Institute of Technology Jodhpur as a partial fulfillment towards the requirement for the award of the professional degree B.Tech. This report thus serves as an authentic work of the research and experimental work which is done under the supervision of Dr Binod Kumar. I also declare that in no circumstance the contents of this work is submitted by me or any entity to any Institution, domestic or abroad towards the requirement for award of any other degree.

Suyash Bansal

Signature

Suyash bansal

B20CS076

Sudip Kumar

Signature

Sudip Kumar

B20CS072

Certificate

This certificate is to attest that the Project Report titled Memory Optimization in deep neural networks for constrained hardware devices , sub- mitted by Suyash Bansal (B20CS076) and Sudip Kumar (B20CS072) to the Indian Institute of Technology Jodhpur for the award of the degree of B.Tech., is a bonafide record of the research work done by them under my supervision. To the best of my knowledge, the contents of this report, whether in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Signature

Dr. Binod Kumar

Acknowledgements

The work done towards the project titled Memory Optimization in deep neural networks for constrained hardware devices wouldn't have been possible had it not been for the great deal of support from my guide Dr Binod Kumar. The topic was quite challenging and my advisor helped me enough to traverse this research area as smoothly as possible. Furthermore I really am thankful to Phd Students Abhishek Yadav sir and also Vamsi sir for all the help in successful accomplishment of the project as without them the project wouldn't have been easy to implement.

Additionally I would also like to thank the faculty members of CSE department at Indian Institute of technology Jodhpur, without the guidance of whom the project might not have been completed as they were instrumental in not only telling the possible areas I can improve my project upon but also helping me to arrange the logistics needed for the project

Lastly and not the least I would also like to thank the evaluation panel for their constructive feedback on the project, they helped me in iterating over the project as they succinctly told the shortcomings and possible areas I can improve my project upon.

Abstract

The report contains the contents of the work done towards the project Memory Optimization in deep neural networks for constrained hardware devices. The project has two motivations. One of the motivations includes the opportunities it will create for the AIOT based application to be able to deploy on such small hardware constraint devices with good accuracy. Additionally it would also help power saving by reducing the energy requirement for data transfer in the hardware from one module to another.

Second Major motivation is to increase the use cases of Neural networks on the IOT based hardware devices to enhance the performance by using a minimum amount of memory.

Contents

Memory optimization in Neural Networks

1 Introduction and background

Neural networks have become strong tools in the field of deep learning and artificial intelligence, with a wide range of applications. Nevertheless, a major obstacle is its implementation on limited hardware, which is defined by low processing power. Memory optimization is one important issue that needs to be addressed in this situation.

Complex models in particular frequently need large amounts of memory, which may be more than limited hardware devices can provide. This restriction makes it more difficult for neural network-based applications to operate effectively in environments with limited resources, such edge devices, Internet of Things devices, and other embedded systems. Memory limitations are becoming more and more of a concern as these hardware platforms become essential to many different industries, such as industrial automation, smart cities, and healthcare.

Memory is a finite resource, and in many cases, organizations need to consider the cost associated with acquiring and maintaining memory. Optimizing memory usage can lead to cost savings by reducing the need for additional hardware resources. Efficient memory management allows for better utilization of available resources. It ensures that memory is used judiciously, preventing unnecessary consumption and wastage. Efficient memory usage leads to better overall system performance.

Optimized memory usage means that applications and systems can operate with lower resource requirements, resulting in faster response times and improved user experiences. Many modern devices, such as smartphones and Internet of Things (IoT) devices, have limited memory capacities. Optimizing memory usage is essential in these cases to ensure that applications run smoothly without draining the limited resources available. As applications and systems scale, efficient memory usage becomes crucial. Scalable solutions need to handle increasing workloads without a linear increase in resource requirements. Optimized memory usage facilitates better scalability.

Big data processing and analytics involve handling large datasets. Optimizing memory usage is crucial in these scenarios to prevent performance bottlenecks and ensure that data-intensive operations can be carried out efficiently. Therefore optimizing memory usage is a multifaceted concern that impacts performance, cost, scalability, energy efficiency, and overall system reliability in the contemporary computing landscape. It allows organizations to make the most of their resources and deliver better experiences to users while also addressing environmental and economic considerations.

2 Literature survey

A system's design heavily relies on optimization. It is required of an embedded system to satisfy user needs and optimize hardware and software specifications in order to minimize design complexity and expense. The number of components required, execution time, memory needs, power consumption, and resource allocation are the main variables that are optimized for hardware. The literature has suggested a number of optimization strategies to deal with the issue[1].

One of the most popular optimization techniques for a variety of engineering and non-engineering applications is the evolutionary algorithm. There are various way possible for memory optimization or how effectively work with a resource constrained device [2], Prior to feeding an input image into the CNN model, the input image will uniformly split into many sub-images, which will then be fed into the models one after the other until the output is combined after a specific layer.

The weight parameter is one of the most important aspects of a neural network's performance [3]. The method of initialization determines its significance [4]. A well-chosen initiation procedure will lead to a network that operates well. Generally speaking, the network's weights are initialized with tiny values. Resource constrained devices cannot implement larger networks in real time due to their limited computational power and insufficient memory. Therefore, smaller networks that maintain competitive accuracy but require less compute and memory must be developed. attempting to incorporate AutoAugment, a data augmentation method that helps to improve our model's accuracy. With this change in architecture, the model can be used for embedded devices, mobile devices deployed for real-time applications such as object recognition and autonomous vehicles, and devices with limited resources. Mish activation, heterogeneous kernel-based convolutions, and other architectural changes are involved [5].

Recently, in [6], they proposed a compact solution for signal classification in which three macro layers of a specially designed convolution neural network—hereafter referred to as NL-CNN, with NL standing for NonLinear—were fed signals through an image spectrogram converted by a specially designed feature extractor. A cascade of widely available Keras/Tensorflow layers, each level having a conventional (linear) convolution followed by a ReLu nonlinearity, is used to imitate non-linear convolution (NL) in each macro-layer [7][8].

The convolutional neural network (CNN), which was first proposed, has outperformed other techniques in image recognition, achieving impressive results. New neural network models that are appropriate for terminal devices, like MobileNet, SqueezeNet, and ShuffeNet, have also emerged in the CNN model in recent years due to the ongoing improvement of computational complexity and the growth of research on the deployment of neural networks on terminal devices. To cut down on computation, these network models feature unique units or structures.

As stated in [9], however, "it is not surprising that the design of neural net accelerators proceeds without much consideration of the characteristics of the latest deep neural net models, and the design of state-of-the-art deep

neural net models proceeds without much consideration of the latest hardware targets." Certain operations, like channel shuffle in ShuffeNet and channel split in ShuffeNet V2, among others, have an impact on hardware level that was not taken into account during the model's design. On the other hand, many FPGA designs were still optimized for models with high redundancy, like VGG16[10], and the most recent developments in deep neural network model development were not utilised.Moreover, general convolution operation optimisations like those in [11] do not benefit from the features of FPGA customisation.

The authors of the article[12] built the state-of-the-art ShuffeNet V2 model, which is based on FPGA. It performs better in terms of latency and resource utilization without causing additional performance loss, according to the analysis report, utilizing algorithm and hardware co-design techniques such data quantification, convolution unit enhancement, and shuffling unit redesign.

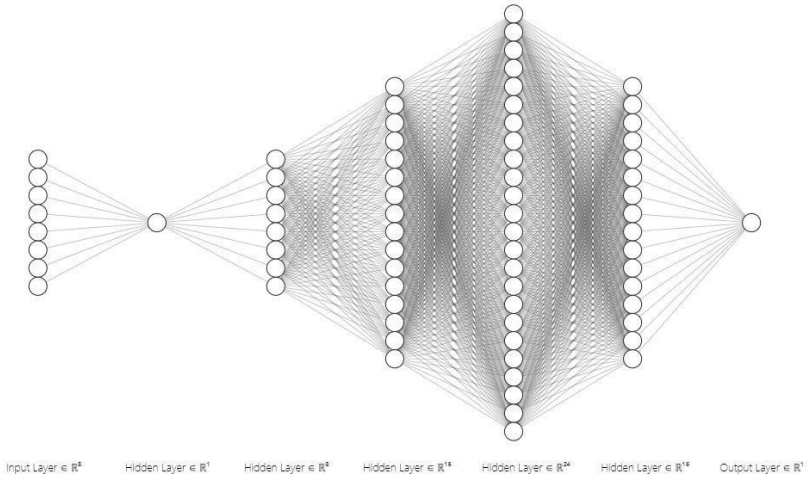
Two key lines of inquiry are being pursued in the study of FPGA-based model design. The first path tends to compress the model, trying to shrink it to a manageable size loss of accuracy. Related work includes data quantification[13], pruning or sparse matrix, and so on. The alternative approach involves optimizing the model's structure, which primarily entails: 1) Optimize the calculation unit 2) Optimize the loop structure 3) Optimize the memory access structure.

3 Problem definition and Objective

The major problem about the neural network running on the constrained hardware resources is that it has a limited amount of memory and computation power is quite small and limited which makes it a bit difficult to get the neural networks deployed on the AIOT application based devices. So this issue has to be addressed.

Hence our main objectives for the project are as follows

- Memory Efficiency Improvement : Firstly our objective is to reduce the amount of memory of trained neural network in order to reduce the memory of the trained model which would enhance the performance of the neural network on the hardware based systems as data buses in the hardware need not to transfer the data of weight again and again from storage to some computation unit like ALU.
- Performance Retention : Further we will observe the impacts of the memory optimization on the accuracy of the neural network as it is an important aspect which we have to address as the accuracy is the ultimate thing we need to achieve from the neural network in order to fulfill the application purpose it is used for.
- Through experimentation, analyzing the data obtained from testing we could come up with some good techniques which could be integrated with some Deep learning frameworks which are specially meant for the hardware and AIOT based applications.
- Analyzing the amount of memory reduction as it will be highly useful for the real time constrained devices. It genuinely helps in faster inference in such devices which makes it really useful.



4 Methodology

4.1 Augmenting weights of layers in neural networks

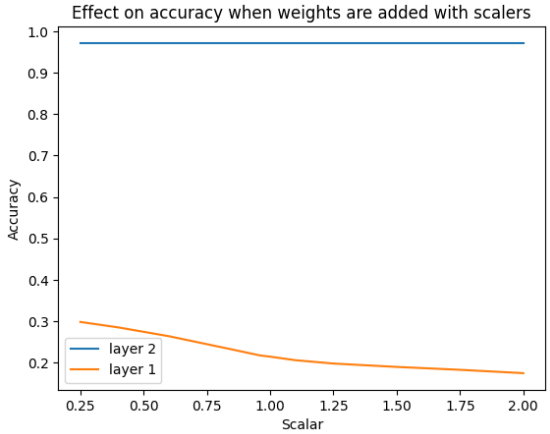
Augmentation in the weights of layers in a neural network allows one to observe the effect in accuracy after changing weights of each layer of the neural network through various techniques such as scalar addition, scalar multiplication and bitflip method. The weight augmentation helps us understand the impact it creates on the accuracy of the model on a certain dataset. These experiments have been performed in order to judge whether there is some significant effect on the accuracy after changing the weights and to know that which layers are more affected and which are not significantly affected can be determined through this augmentation.
Augmentation can be of multiple types and levels. Basically noise is added to the weights by variety of ways some of those are listed below -:

4.1.1 Scalar Addition

Scalar Addition is a method of augmenting the weights of the different layers of the neural networks by adding some noise in the form of some scalar quantities. Adding scalars to original weights of the layer then evaluating accuracy of the neural network model in order to check the impact of the scalar addition on the weights of neural networks.It can be achieved simply by adding the noise vector to the weights vector.

$$new\_weight = original\_weight + scalar\ Noise\ .$$

After changing the weights of the whole layer use new_weights for the evaluation of the model.
Here we have been altering the weights for the dense layers in the neural networks. We have here used a simple 4 - 5 layered dense neural network where we applied the noise augmentation on each of the layers and tested the accuracy changes of the model.
We have used Scalar Noise vector = [.25, 0.4, 0.6, 0.85, 0.96,1, 1.1,1.25,1.5,1.75,2.].



Accuracy impacts of noise addition

Here in the image it is quite evident that the accuracy drop is quite significantly on augmenting the weights of the initial layers which clearly specifies that model is untrained in the starting layers so those weights are crucial in the starting of the model but if we see the layer at the end of the model its weights augmentation does not have much impact on the accuracy of the model.

This clearly states that in the initial stages of the model training the weights of the feature values are quite important which cannot be altered but when the model is quite decently trained the weight alternation is not making much of the impact.

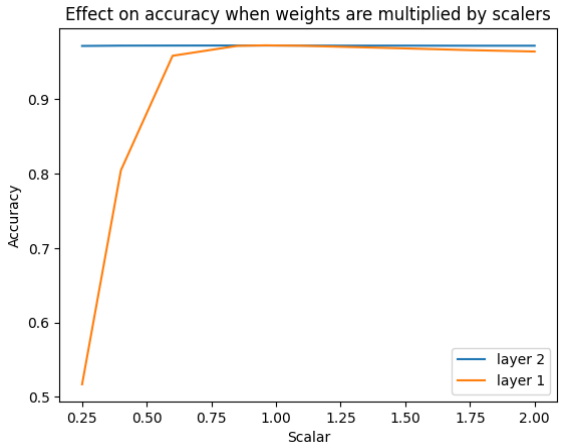
4.1.2 Scalar multiplication

It is another method for embedding noise in the values of the weights of the individual layers of the neural networks. Here the weights augmentation is observed by multiplying some scalar values to the weights of the layers in the neural network. This can be achieved through the formula:

*new_weight = original_weight * scaler.*

Further the weights alteration via multiplying some scalar quantities impacts accuracy in similar way as of the scalar addition. We have applied the same set of scalar noise vector to augment weights which is

Scalar Noise vector = [.25, 0.4, 0.6, 0.85, 0.96,1, 1.1,1.25,1.5,1.75,2.].



Accuracy impacts via scalar multiplication

Here in the above graph we can clearly see that the accuracy of the last layer, that is layer 2 is not changed significantly when we multiply the weights of the layer by the different scalar values from 0.25 to 2.00. But on the other hand we can observe that the weight change of layer 1 has changed the accuracy drastically. When the layer 1 weights are multiplied with the scalars less than 1 the accuracy increases as the weights are slowly attaining their original values and on scalar = 1 the accuracy becomes as of the original one's but it again starts to drop slowly when scalar value is increased.

From the above two experiments performed for many models on the weights of the neural network we can say that the accuracy drop is not significant for the last few layers of the neural network. While it is quite decent for the starting few layers of the neural network. Here both the operations performed so far are to check whether there is

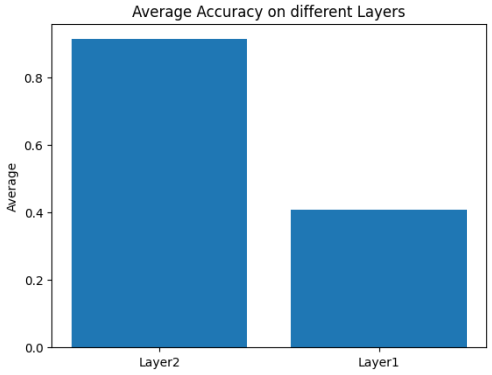
significant impact on accuracy when the weights are changed slightly, that is why a limited range of scalars have been selected.

4.1.3 Bit flip Method

This is another method of weight augmentation in the neural network layers where we alter the weights using the bits in the binary form. We have just altered the bits in it randomly which resulted in the change in the weight value of the layers.

Here in this method we randomly flip one of the bits of each original weight and after flipping bits for all the weights in the layer we evaluate the accuracy of the model.

As each weight is of 32 bits, randomly select one index and if the selected bit is '1' then convert it to '0' else if the selected bit is'0' then change it to '1'. This method provides more insight about the effect on accuracy after weight change for each layer.



This above bar Graph depicts that the average accuracy of layer 2 does not change much whereas if we observe the accuracy of layer1 its average accuracy dips significantly after the weight alteration. This clearly signifies the experiment that we can make changes to the last few layers of the model in order to keep the accuracy of the model intact.

4.2 Weight centralization

Based on the experiments performed above to check whether the centralization of weights affects accuracy we concluded that the weights augmentation can be done to some extent and still the accuracy does not suffer much. This allows us to introduce the way of memory optimization by not storing the entire weights which takes quite a bit of memory after the model is trained. Here we will just be storing the central tendency of the weights such as mean and the standard deviation which are sufficient to generate the random weights which can be used for the

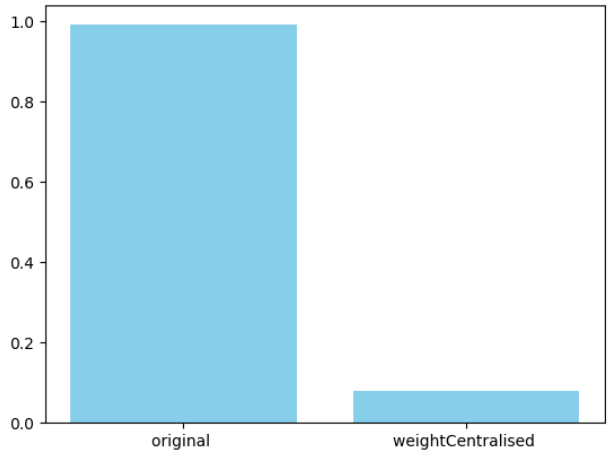
inference purpose as the accuracy has not been dropped much so it will allow reducing the required storage space from the entire weight unit to some or few weights. There are multiple ways to implement the weight centralization in the neural networks where some of them are explained below.

4.2.1 Unitary weight centralization

The main idea here is to generate a single central weight for all the layers of the whole neural network. This central weight then will be used to generate other weights on the go while inferencing on the model. To achieve this we calculate the mean of the weights of the whole network and standard deviation as well. Now we will generate a random new_weight using mean and standard deviation. Following formula is used to calculate new_weight:

New_weight = random.uniform(mean - standard deviation , mean + standard deviation) + bias .

Unitary weight generation is not at all good idea as it affects the accuracy too much which demolishes the model purpose.



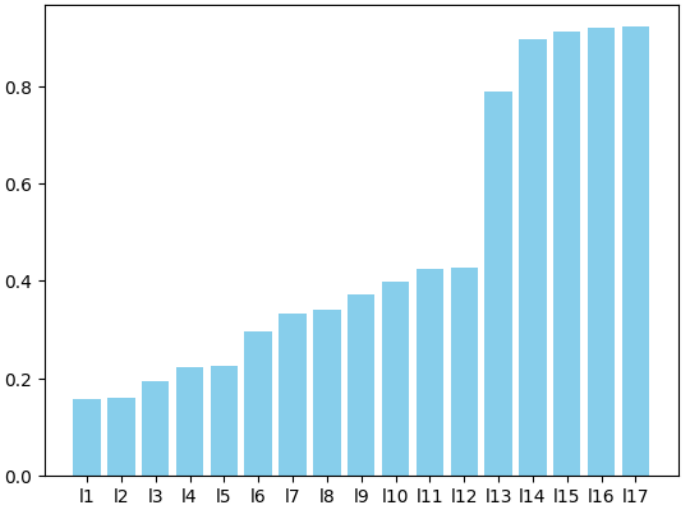
Accuracy comparison of unitary weight model to original

The above graph clearly shows that centralisation cannot be done unitarily as it is directly impacting the accuracy of the model to a very great extent which is not fulfilling the need for the application this model is used for. There has to be some better improvements in the weight centralised methods to maintain the accuracy to some good standard.

4.2.2 Layer wise weight centralization

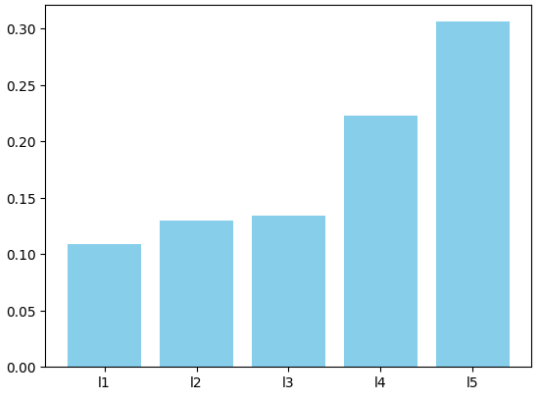
Layer wise centralisation of the weights is basically the centralisation of the weights by introducing some standard measure of central tendency for individual layers. Each dense layer will have some central weights values instead of all the dense layers having just one central weight. Now with each layer possessing some central weight through which the rest of the weights are generated. This shows a better performance as compared to the one where we have used earlier that was using only single weight as the parameter.

It will have a central mean and standard deviation for weights of each of the dense layer and then the new model trained will be optimized by just storing the central weights and the mean values and generating the rest of the values for weights randomly.



Accuracy vs layer number in Artificial Neural Network

Based on the above bar graph we can clearly state that the accuracy of the last layers does drop much as we can see in the bar graph above the accuracy has not dropped much as we go on from layer1 to layer 17 introducing the central weights for each of the layers in the neural network. The original accuracy of the model is nearly 98.18 on the testing data and it has dropped to 92.19 for last layer which is decent to still fulfill the purpose of the model .Here we can see that the model accuracy has not dropped much for the last layers but using only one central weight for each layer can be poor at times. It is not the case for every model that using only 1 central weight would lead to a decent value of accuracy. The accuracy will drop for many models if we use just one central weight.



Accuracy vs layer number in ANN

The above bar graph shows the case where the accuracy of the last layers is also dropped significantly by using only 1 central weight. So here we can see that the accuracy has come down to 30% nearly from 93.10 % which is the original accuracy of the model when the weights were original not the augmented ones. The amount of accuracy drop does not make it feasible for using it with real time applications.

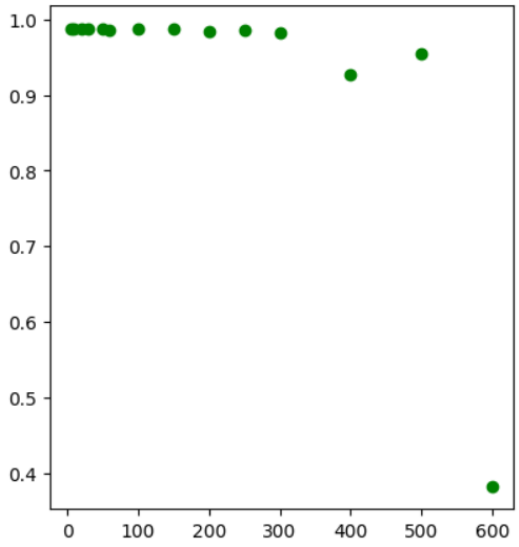
4.2.3 Batch wise weight centralization

Batch wise Weight centralization is the method of generating some central weights replacing the large amount of weights which is dependent on the size of the batches into which it is divided. Now if we generate some number of central weights instead of the just 1 weight that would sustain the accuracy as well as improve the memory requirements of the model.

Number of central weights = total number of weights / batch Size

Batch wise central weights is a very good method of working with the central weights which allows the overall noise reduction in the weights and hence maintaining the accuracy of the model.

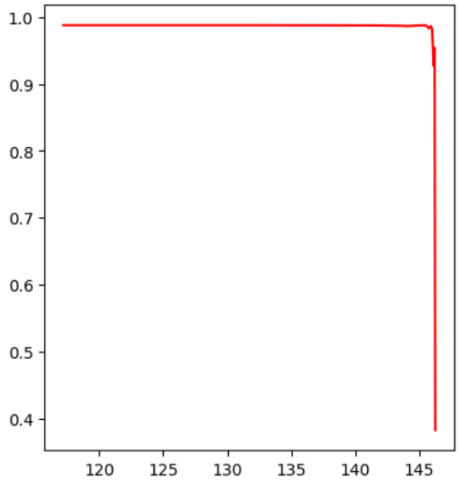
Now the Batch centralisation is a major method which is used to optimize the memory levels in the neural networks to make it work efficiently in the hardwares.



Batch size vs accuracy of the model

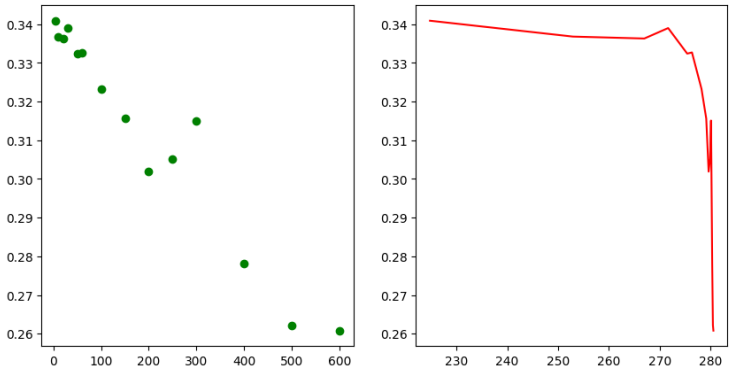
The scatter plot above shows the accuracy of the model on different batch sizes which are taken from 100 to nearly 600 and we can see that creating the batches of certain size does not drop the accuracy to great extent but after the batch size goes past 600 accuracy of the model is affected significantly and it drops sharply from near 90% to less than 40 % which indicates that the batch size cannot be increased more than that. The central weights generated are layer wise which means for each layer we have created the central weights that is not 1 but depends on the batch size so it will reduce the noise in the weights.

Further the memory reduction can be also observed from the graph , the amount of memory which is saved with this batch weight centralization.



Memory Saved (KB) vs accuracy drop

This parameter allows to see the amount of memory been optimized keeping the accuracy significant in the value is shown in the memory graph above where we can see the memory saving is nearly 140 KB upto which the accuracy is not dropped significantly.



There are more such examples where we could observe the accuracy drop differently but we can see that we can always choose some batch size which will allow us to optimize the memory as well maintain the accuracy of the model. The choice of the batch size depends on the application on which this optimization method is applied.

4.3 Testing on standard neural networks models

Experiments performed so far ensure that the batch centralisation is a method which can be generalized for all sorts of neural networks layers where the weights memory could be optimized by it. Now if we can prove that the batch weight centralisation works perfectly for the standard light weight models then it could be considered optimal for any sort of neural networks.

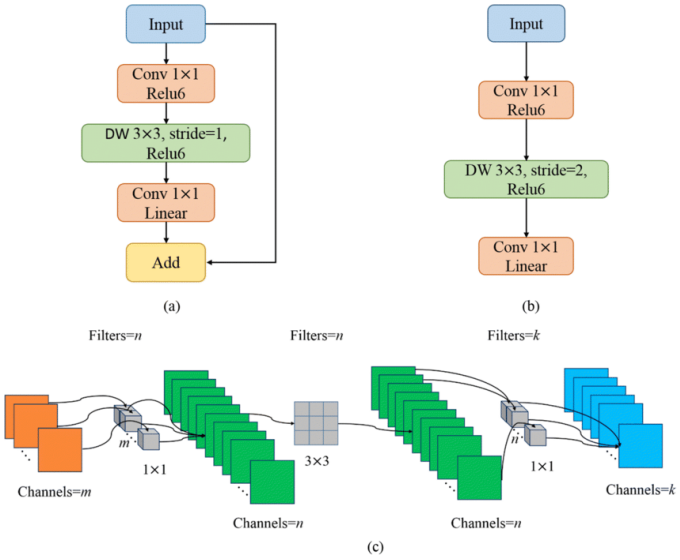
To test whether the proposed thesis of memory optimization will work for all the models it is necessary to generalize the method. For this purpose we have used some standard models like mobileNet V2, shuffleNet V2, SqueezeNet and some other non standard models as well so as to observe the accuracy pattern with the amount of memory conservation by the weights of the neural networks.

4.3.1 MobileNet V2 Model

Mobile Net v2 consists of multiple layers of different sorts of operations such as convolutional operations and global average pooling operations and many such other operations.

MobileNetV2 introduces the concept of inverted residuals, which uses lightweight depthwise separable convolutions inside inverted residual blocks.

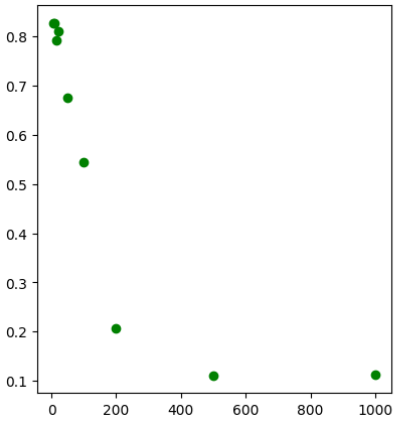
Linear bottlenecks are used to preserve information flow and avoid non-linearities in the bottleneck layers.



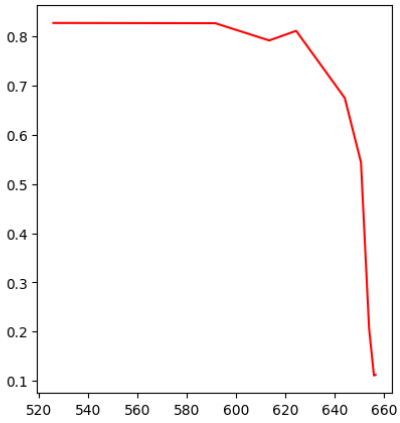
General architecture and channels of the MobileNetv2 . a Stride = 1 block; b Stride = 2 block; c Numbers of channels per layer in MobileNetv2

We have applied Mobile net V2 onto the standard MNIST dataset and on training we have found out that it has been trained on nearly 2,426,410 parameters.

Mobile Net V2 is the lightweight standard model which is generally applied to real time general applications on the hardwares. Testing the memory optimization algorithm on the Mobile net v2 results in the following way :



Accuracy vs batch size



Accuracy vs memory Saved (KB)

Based on the above graphs we got for the accuracy vs memory savings we can infer that the accuracy is dropped after batch size goes past 100 to 200 which means total weights / batch size will be the amount of memory it would have used for storing the central weights which includes the measures of central tendency.

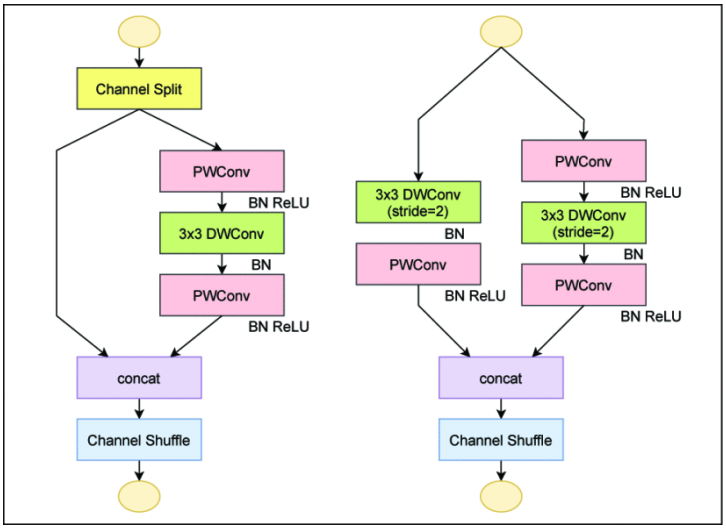
The batch sizes used for Mobile net v2 are :

[5, 10,15,20,50 , 100, 200 , 500 , 1000]

Out of these batch sizes the most suitable in keeping the accuracy of the model to be at certainly decent level is 100 where we can observe that the memory saving is also enough and the accuracy is also quite decent for the real time use cases.

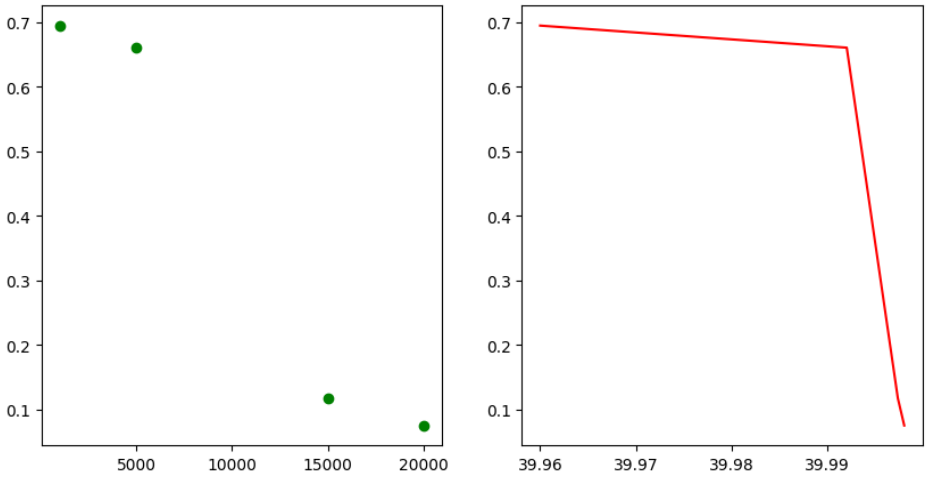
4.3.2 ShuffleNet V2 Model

Another lightweight model which is highly used on low computation intensive hardware is a lightweight neural network architecture called ShuffleNetV2 was created for effective and precise picture classification on systems with limited resources, such as mobile phones. It is a development of the first ShuffleNet, with enhancements made to retain low computational complexity while achieving higher performance.



The use of channel shuffling, which permits communication across channels across spatial dimensions, is the main innovation in ShuffleNetV2. This lowers computing costs and enhances the model's capacity to pick up intricate representations.Using grouped convolutions, which divide the input channels into subsets and apply different convolutional operations to each subset, ShuffleNetV2 operates on input channels. When compared to conventional convolutions, this lowers the computing cost.

Now applying the memory optimization on the Shuffle Net V2 we have observed following results as :



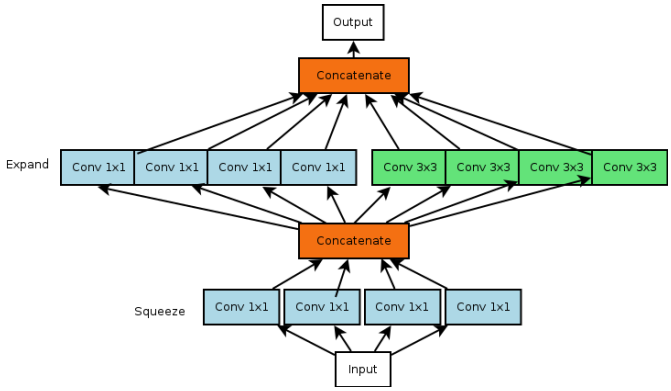
Shuffle Net v2

Here in the graph we can see that the accuracy is gradually dropping when batch size of the weight centralisation is made 10K from 5K but then there is sudden drop in the accuracy of the model when further the batch size is made to 15K which signifies that it is threshold upto which the accuracy could be dropped and dropping it further would not make it suitable for real application use.

Here I have used handwritten Alphabets as my dataset and model is trained on over 2000000 parameters. The results of the memory saving states that we can save nearly about 40 Mb of memory using this memory optimization technique in the shuffle net v2.

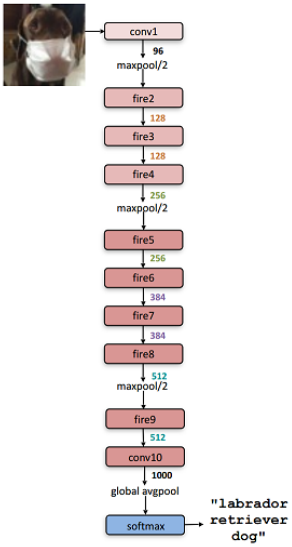
4.3.3 SqueezeNet Model

Compared to conventional deep learning models, squeezeNet, a lightweight neural network architecture, is incredibly accurate with a significantly less number of parameters because of its careful design. Using "Fire Modules," which combine a squeeze layer (1x1 convolutional layer) and expand layers (1x1 and 3x3 convolutional layers), is one of its standout features.



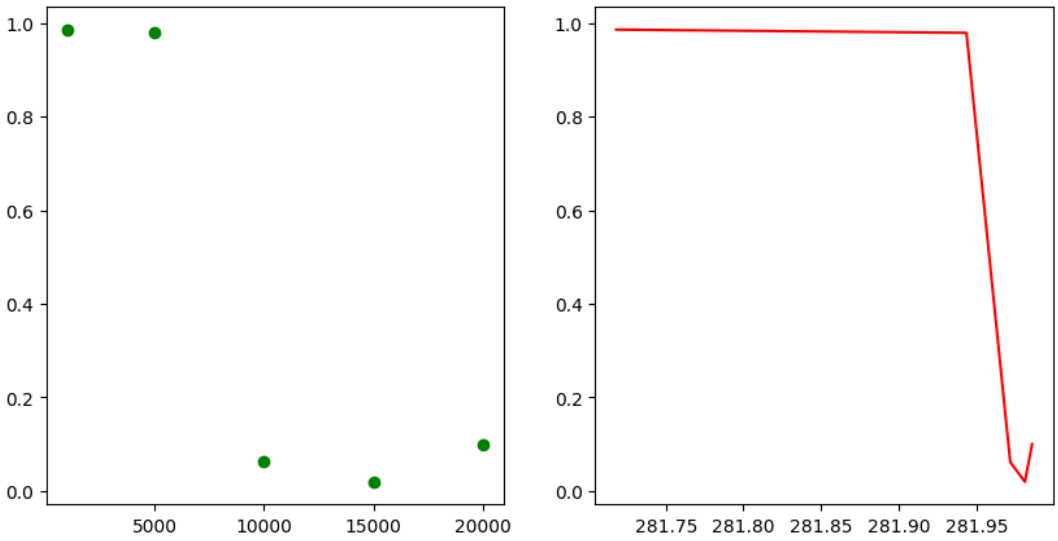
This creative approach maintains expressive power while lowering the number of parameters in the model. Furthermore, at the end, SqueezeNet uses Global Average Pooling (GAP) instead of fully connected layers, which

effectively lowers the total number of parameters and model size. Prior to applying more resource-intensive 3x3 convolutional layers, the strategic usage of 1x1 convolutional layers further optimizes computational efficiency by minimizing input channels.



General architecture of a squeezeNet model

Squeeze net also is quite oftenly used in the hardware devices such as IOT applications and multiple such other applications squeeze net's are extensively used . So optimizing memory of squeeze net could be highly useful to make it more efficient for the small hardware based applications.



Squeeze Net

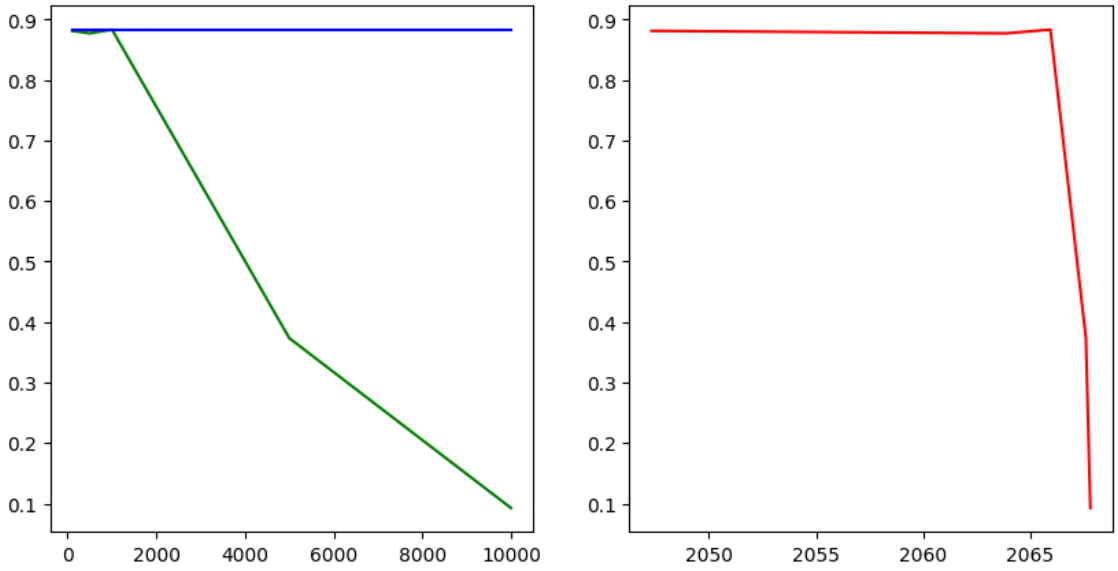
Similar graph for the squeeze net can be also observed as we have observed for the case of shuffle bet and Mobile net V2. Here we can clearly see that accuracy is quite significantly dropped after a particular batch size which means that batchsize 5000 is acting as threshold in this case and if we see the size of weights memory we are able to conserve in these cases is quite huge which is nearly around 280-290 Mb which is significant.

This model was applied on the CIFAR - 10 dataset where the parameters on which model is trained is nearly around 2.5 million.

4.3.4 Analysis

After collecting the datasets and testing the datasets on variety of models we have observed that the Batch weight centralisation can be generalized for any model over any dataset by deciding the threshold value for batch size for which it will remain efficient, effective and accurate enough to fulfill the purpose of the application for which it was trained tested and developed.

Upon testing the memory optimization algorithm on the standard models ensures that it could be used for any model. But with this memory optimization we could apply this for majorly dense layer networks which includes mainly multi layer perceptron type neural networks if we wanted major memory saving or complete memory saving from the neural network.



The graph below shows the memory saving as well as the accuracy vs batch size which implies that the memory saving for the cases of multilayer perceptron is too high which makes it feasible for the algorithm we have used for memory saving.

Now if we can see that the amount of weight parameters this batch centralisation is reducing during inference is quite significant which is revealed in the above image.

This MLP model is trained on a 5 layer dense neural network with numerous parameters in the training process of the neural network.

5 Theoretical/Numerical/Experimental findings

5.1 Impact of weight centralization on accuracy

The Impact of the accuracy on the model after the Batch wise Weight centralisation algorithm is applied is the major deciding factor for the algorithm to work. So to study the impact of weights centralisation on the accuracy can be observed as shown below.

	Model	Dataset	Original accuracy	Threshold batch size	Memory Optimized Accuracy
1	MobilenetV2	MNIST	82.99 %	50	79.12
2	ShuffleNetV2	CIFAR 10	69 %	5000	67.17
3	SqueezeNet	A_Z dataset	98.69%	5000	97.79
4	5 layer MLP	covid 19 prediction	92.76%	200	85.50
5	Custom 4 layer MLP	Poker hand prediction	70.56%	20	63.76

Accuracy comparison table

Here the table is completely revealing the model name , the dataset on which it is experimented and also the accuracy changes in the memory optimized model and the original model with the actual weights and other one with the centralized weights.

Further the important aspect of this table is that the memory optimized accuracy is something which varies highly according to the threshold batch size which would decide the number of weights that would remain after centralisation.

Accuracy impact on the weights are very much dependent on the model as well as dataset so it will not always be the case that the size of threshold batch would be very large it is quite possible that the size of the batch is significantly lower than expected. Here is one example where the batch size is nearly 20 weights combined to form a single centralized weight value which is quite less but it will ultimately save some amount of memory without affecting the accuracy much.

5.2 Memory size optimization

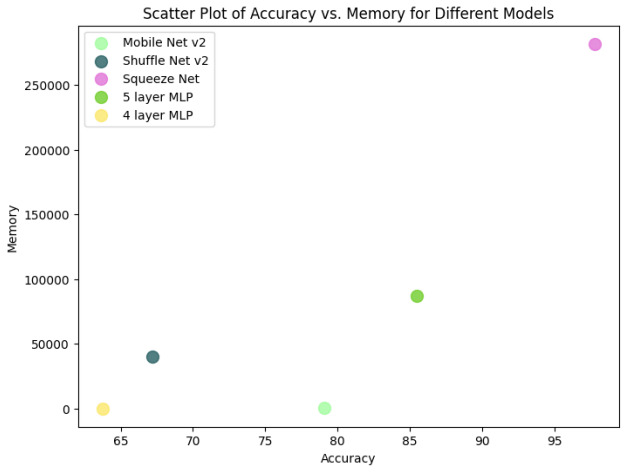
The amount of memory saving on the model after the Batch wise Weight centralisation algorithm is applied is another major deciding factor for the algorithm to be optimistic.. So to study the impact of weights centralisation on the amount of memory saved can be observed as shown below.

	Model	Dataset	Original accuracy	Threshold batch size	Memory Optimized
1	MobilenetV2	MNIST	82.99%	50	540 KB
2	ShuffleNetV2	CIFAR 10	69%	5000	40 MB
3	SqueezeNet	A_Z dataset	98.69%	5000	282 MB
4	5 layer MLP	covid 19 prediction	92.76%	200	87200 KB
5	Custom 4 layer MLP	Poker hand prediction	70.56%	20	34 KB

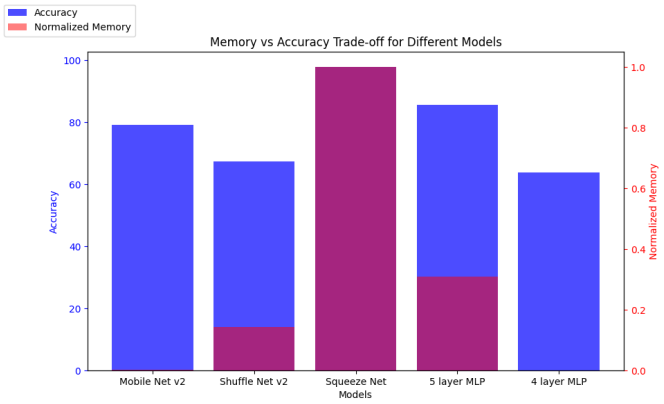
Table above is showing the amount of memory optimized for the major models with variety and standardization of the models. Now further there have been a lot of variations in the amount of memory being saved by the weight centralisation due to the difference in the number of parameters of the models and dataset and also the amount of model gets altered on augmenting the weights of the model. Now if we measure the memory saved for the case of the model with 4 dense layers we can see the memory saving is not significant which implies it is not always the case that memory saving would have a significant amount.

5.3 Accuracy VS memory tradeoff

It is quite significantly important to study the tradeoff between accuracy and memory savings in order to get an estimate of what is important for the application on which we are currently working on. Accuracy and memory savings both should be considered while creating a tradeoff among them. It is quite essential to get an estimate of both parameters. Now the tradeoff can be explained with the following graph for various models as :



This plot clearly specifies the accuracy vs memory tradeoff we can see that those who are above the line y = x are the ones with higher memory saving and the one below are the ones dominated by the accuracy but the memory saving is pretty low and the one on the line y = x means they are one with best situation the memory saving along with the memory savings is also high so here we can see that for the squeeze net the memory savings is quite high and also the accuracy is max so it tends to be the perfect model for the above applied algorithm for batch centralisation.



The bar plot also signifies which model is best in what we can see the pinkish color depicts that the accuracy and the memory savings is both high but in MNIST and 4 layer MLP we can clearly see that the accuracy is managed and the memory is not optimized very highly as compared to the ones of the rest of the models.

This clearly signifies the algorithm working and importance as well as the findings which suggest the threshold batch size can make the memory optimize along with the accuracy maintenance.

Summary and Future plan of work

The project we have attempted is to provide insights of the memory optimization using the Batch weight centralisation and observe the tradeoff between memory conservation and accuracy. We have basically experimented with weights of neural networks to obtain memory optimizing algorithms.

- We have initially tried to augment the weights in order to observe the changes pattern in accuracy how much it is getting affected via variety of augmentation techniques and we have applied those layer wise so we have observed that the last layers when augmented they are making the least impact on the accuracy while the initial layers when augmented they are making quite significant accuracy alterations
  .
- Based on the above observations we have actually tried to implement the weight centralisation method in different ways such as unitary weight centralisation , layer wise weight centralisation and then further batch wise weight centralisation . Among them batch wise centralisation was the one which worked for all datasets and models for some batch size which may be small or large depending upon the dataset as well as model. The rest of the two methods won't be always accurate and might work for some particular test case but they cannot be generalized.

- After adapting to the Batch weight centralisation we have applied and tested this optimizing algorithm on multiple standard and non standard MLP type models for assurance of its working. So it might be possible that you would find somewhere that memory conservation is too high while for other models memory conservation is not that significant. We have applied this on Mobile net v2, Shuffle net v2, squeeze net v2 .

Future plan of work includes introducing some more algorithms via experimental testing which would reduce the number of operations which occur during the model training from one layer to another layer. As we know that output neuron are generated via w*x + b where w and b are feature vectors. So an observation made out of the neural network weights there are multiple weights which are 0 in magnitude so in hardware we need not to send them to the ALU section for operations rather the output can be directly written for such cases as b so that would improve efficiency of the neural network on the hardware.

This method is termed as 'Zero Skipping ' where we are skipping zeros multiplication and saving the time for its calculation. It could be implemented by changing the architecture of the neural network where we would find 0 weights from the layers and skip their operations with the features nodes and directly write the output corresponding to the bias which would allow the neural network to improve time efficiency for the model inferencing.

Contribution

Suyash Bansal (B20CS076) :

In our project there are basically 3 segments. Experimental Testing, weight centralisation and standard model testing. Each part has multiple layers involved . Experimental testing involves the weight augmentation via various methods such as scalar multiplication, scalar addition and bit flipping and 8-bit quantization, where scalar multiplication and weight quantization are mainly done by me where I have experimented on the trained models by changing the weights of layers in the network by various methods such as scalar multiplication and weight quantization to observe the effect on the accuracy, also I have applied weight centralisation method. In weight centralization I have developed an algorithm to reduce the weights memory of the layers so that overall memory consumption can be reduced which will reduce the memory transmissions during the inference on the hardware so this will increase efficiency of inferencing for the model. Weight centralisation, I have applied various weight centralisation techniques like unitary weight centralisation , layer wise weight centralisation which were not performing well and accuracy drop on generating the random weight using the central weight for the entire model is augmenting the weight values to a great extent. Here Weight centralisation is done mainly for the dense layers of the neural networks as augmenting the filter data in convolutional or other layers can never optimize the memory much and will drop the accuracy of the model significantly which makes this algorithm useless. Hence it is applied for the dense layers of the neural network so that weight memory could be saved for them. Now further I tried Batch weight centralization which is quite useful in generating central weights selecting few weights according to the batch size which is successfully able to maintain the accuracy not dropping much when a certain threshold batch size is selected. Further I have applied this batch weight centralisation on multiple simple to complex neural networks to observe the accuracy changes. So this makes it generalizable that it is able to optimize some amount of memory based on the selection of batch sizes.

This has made the neural network optimize its memory as well efficiency will also improve as the time taken for data transmission from Static memory to Dynamic memory would be conserved rather the weights would be generated instantaneously by the ALU unit of the hardware by using the centralised weights and standard deviation.

After centralisation of the weights on the trained model I have further tried to create a tradeoff between the accuracy and the memory consumption so that the analysis of the accuracy requirement for the particular application can be done and accordingly we can decide the batch size to conserve and save memory for that trained neural network model.

Sudip Kumar (B20CS072) :

Experimental testing is a crucial part for developing any new technique. Here for weight augmentation I have used the bit flip method and scalar addition on different layers of the model to observe the effect on accuracy for each layer individually which played an important role in developing our memory optimization technique. Bit flip method and scalar addition were very crucial in proving that weight augmentation affects different layers differently, that is the initial layers show more drop in accuracy as compared with last layers.

Testing our developed memory optimization techniques on various standard and custom made models was mainly done by me. Weight centralization technique was tested in various ways such as unitary weight centralization which is to apply weight centralization on complete trained model it did not perform good which led to experimenting on our next technique, layer wise weight centralization which is to apply weight centralization on different dense layer of the trained model its accuracy was better than the later technique but not satisfactory, so finally we developed batch wise weight centralization, here we divide the weight of each layer into fixed batch sizes and then apply weight centralization on it. This technique performed as of our expectations. Testing all the three techniques on custom mode models and standard models proved our technique is effective in memory optimization for deep neural networks with memory constraints. Custom trained models such as 5 layer MLP and custom 4 layer MLP with shown accuracy are not dropping significantly after applying weight centralization on their dense layer only 5-10% drop was observed which is acceptable.

I have further tested our memory optimization technique in standard models such as MobileNet V2, ShuffleNet V2 and SqueezeNet neural network model to our surprise weight centralization technique performed even better for these standard models than custom models. Batch wise weight centralization technique was very effective in these standard models which led us to generalize that our developed memory optimization method is generally applicable for all neural network models. Finally checking how much memory is saved for different models. Based upon the batch size and number of dense layers in the models memory saving varies for different models. If batch size is small the memory saving is less but accuracy is good but if batch size is large memory saving is high but accuracy may be low for same cases. Different models have different batch size thresholds where they show optimal memory saving with decent accuracy retention.

References

1. Y.J. Pavitra; S. Jamuna; J. Manikandan, “Hardware Resource Optimization for Embedded System Design: A Brief Review”, “2018 3rd International Conference for Convergence in Technology (I2CT)”
2. Yasuo Ariki, Tetsuya Takiguchi, Tetsuhao Zhuang, Tristan Hascoet, Ryoichi Takashima, “Convolutional neural networks Memory optimization Inference with Splitting Image,”*2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)*
3. J. Vermaak and E. C. Botha, “Recurrent neural networks for short term load forecasting,” IEEE Trans. Power Syst., vol. 13, no. 1, pp. 126–132, 1998.
4. R. K. Srivastava, K. Greff, and J. Schmidhuber, “Training very deep networks,” in Advances in neural information processing systems, 2015, pp. 2377–2385.
5. Maneesh Ayi; Mohamed El-Sharkawy, “RMNv2: Reduced Mobilenet V2 for CIFAR10” , “2020 10th Annual Computing and Communication Workshop and Conference (CCWC)”
6. Radu Dogaru;Ioana Dogaru, “NL-CNN: A Resources-Constrained Deep Learning Model based on Nonlinear Convolution,” “*2021 12th International Symposium on Advanced Topics in Electrical Engineering (ATEE)”*.
7. A. Farina, A. Bellini and E. Armelloni, "Non-Linear convolution: a new approach for the auralization of distorting systems", *in Audio Engineering Society 110'th Convention, May 2001, Amsterdam.*
8. G. Zoumpourlis, A. Doumanoglou, N. Vretos, and P. Daras., “Nonlinear convolution filters for CNN-based learning”. *In ICCV, 2017.*
9. K. Kwon, A. Amid, A. Gholami, B. Wu, K. Asanovic, and K. Keutzer, “Co-design of deep neural nets and neural net accelerators for embedded vision applications,” *in Proceedings of the 55th Annual Design Automation Conference, ACM, June 2018.*
10. K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” Sept. 2014.
11. A. Ahmad and M. A. Pasha, “Ffconv: An fpga-based accelerator for fast convolution layers in convolutional neural networks,” *ACM Trans. Embedded. Compute. Syst., vol. 19, no. 2, pp. 15:1–15:24, 2020.*
12. Zimeng Fan; Wei Hu; Hong Guo; Fang Liu; Dian Xu, “Hardware and Algorithm Co-Optimization for pointwise convolution and channel shuffle in ShuffleNet V2,” *2021*

IEEE International Conference on Systems, Man, and Cybernetics (SMC).

13. J. Wang, Q. Lou, X. Zhang, C. Zhu, Y. Lin, and D. Chen, “Design flow of accelerating hybrid extremely low bit-width neural network in embedded fpga,” CoRR, vol. abs/1808.04311, 2018.