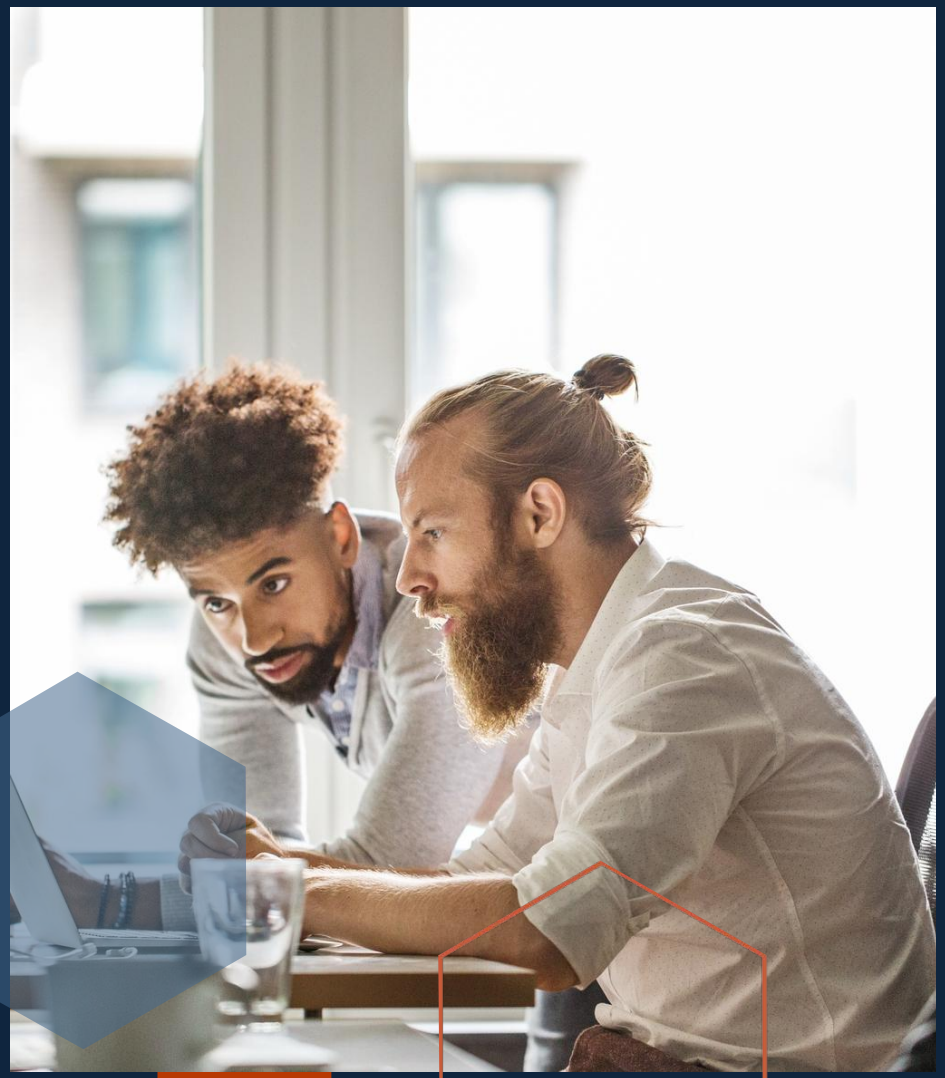
BTP Project

Topic : Memory optimization in Neural Networks for constrained Hardwares



Mentor : Dr. Binod Kumar

Group Members : Suyash Bansal (B20CS076)
Sudip Kumar(B20CS072)

Introduction

Problem Statement

- The primary challenge with deploying neural networks on constrained hardware in AIOT applications is the limited memory and computational power, necessitating the need for effective solution.
- Memory constraint does not allow the small IOT hardware devices to be able to perform computation intensive task involved in the deep learning based applications.
- Performance Retention, is another important problem which is quite important to be addressed that the performance of the model should not dip more than certain threshold.

Motivation

- Since memory is a limited resource, businesses frequently have to take the cost of acquiring and preserving memory into account. Improving memory efficiency can save money by lowering the requirement for more hardware. Optimizing memory usage enables more effective use of the resources that are available.

- The memory capabilities of many contemporary gadgets, including smartphones and Internet of Things (IoT) devices, are constrained. In these situations, optimizing memory consumption is crucial to making sure that programs function properly without using up all of the available resources. As systems and applications get larger, effective memory management becomes essential.

Methodologies

Weight augmentation

- Scalar addition
- Scalar multiplication
- Bit flip method

Weight centralization

- Unitary weight centralization
- Layer wise weight centralization
- Batch wise weight centralization

Weight Augmentation

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.
   a. *new_weight = original_weight + scalar Noise .*

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:
   a. *new_weight = original_weight * scaler.*

3. **Bit flip method:** 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.
   a. *As each weight is of 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'.*

Weight Centralization

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

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 be having some central weights values instead of all the dense layer having just one central weight. Now with each layer possessing some central weight through which the rest of the weights are generated.

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.
   a. *Number of central weights = total number of weights / batch Size*

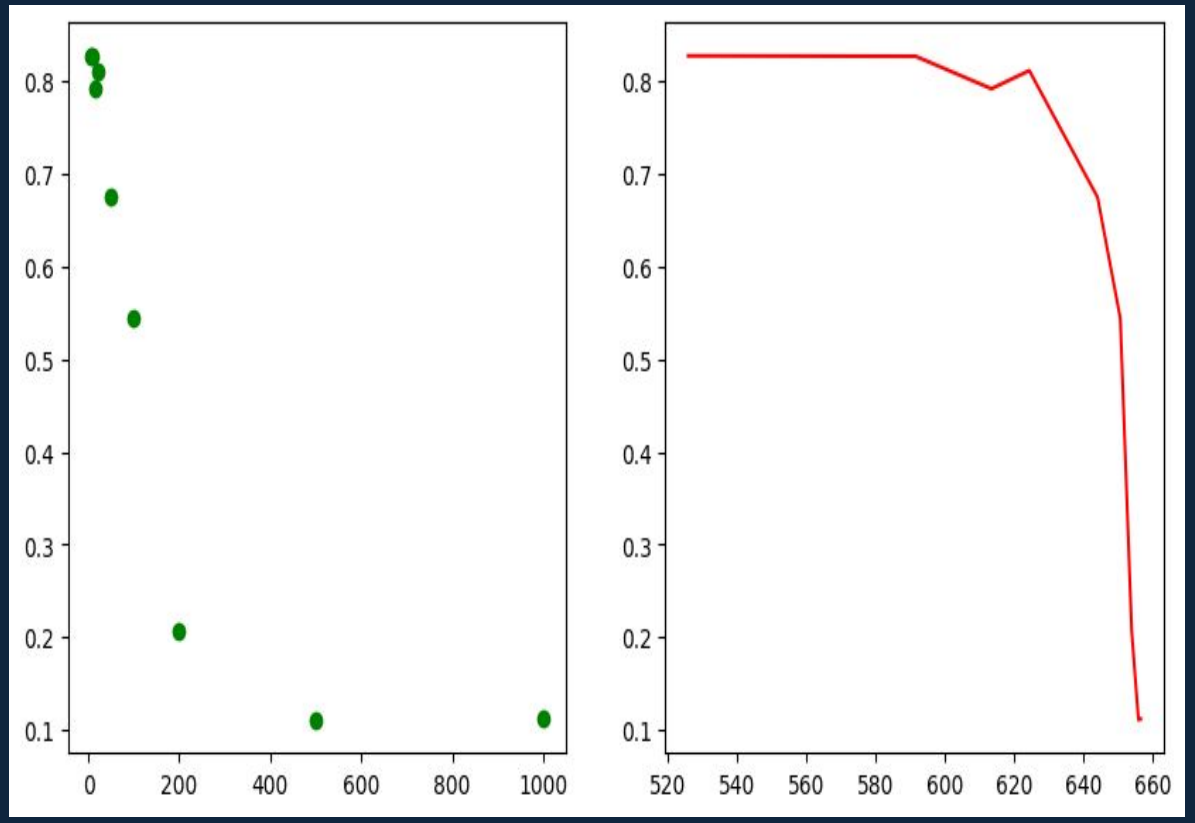
Testing on Standard neural networks

- MobileNet V2 model
- ShuffleNet V2 model
- SqueezeNet model
- 5 layer MLP
- Custom 4 layer MLP

MobileNet V2 model

- 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 are depicted in the graph.

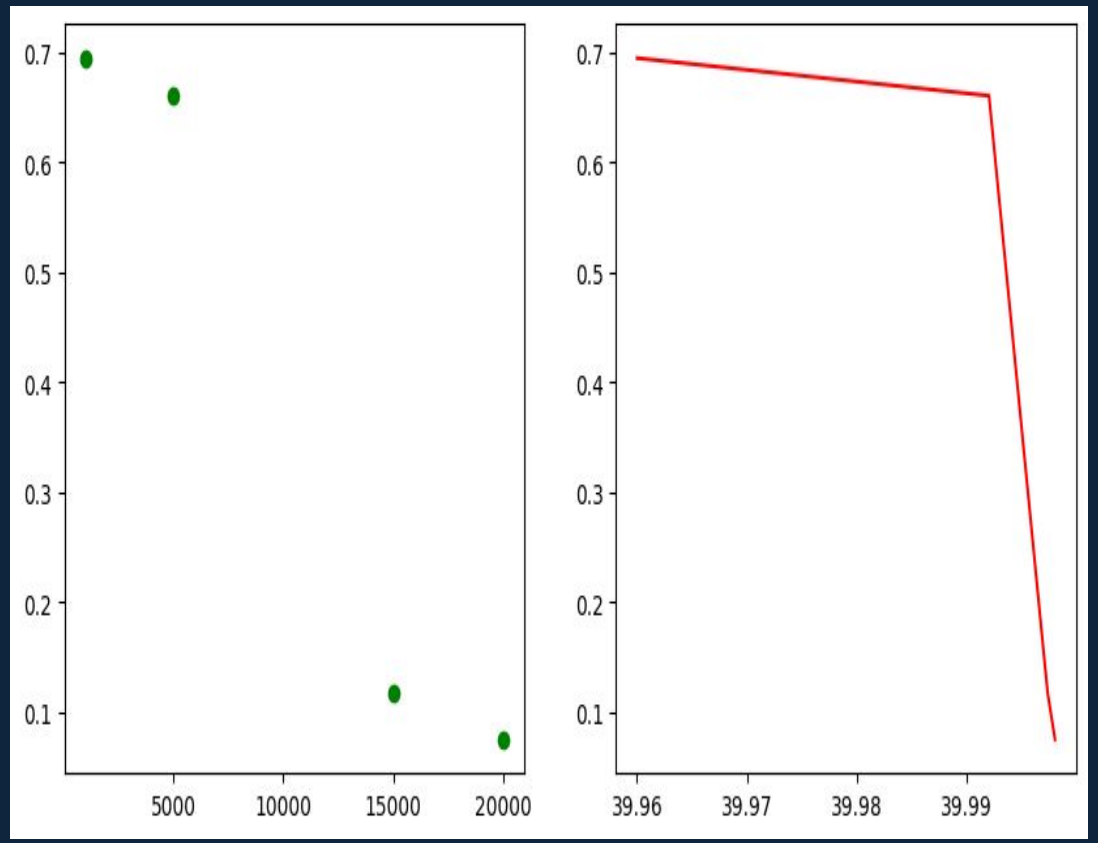
- Hardwares Based on the above graphs we got for the accuracy vs memory savings we can infer that the accuracy is dropped after batch size goes past 100 to 200 which means total weights / batch size will be the amount of memory it would have used for storing the central weights which includes the measures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



ShuffleNet V2 model

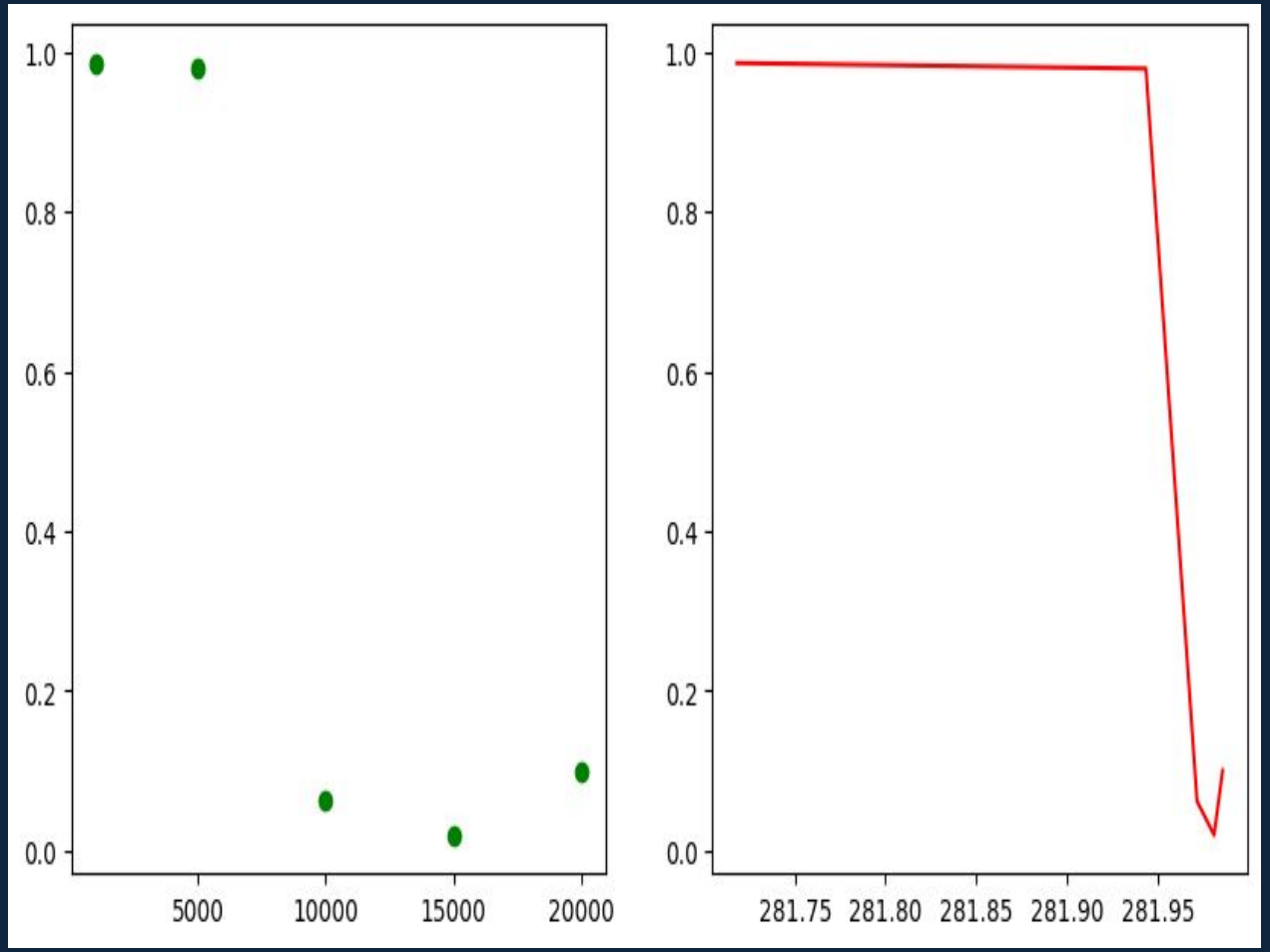
- Shuffle Net v2 is another lightweight model which is highly used on low computation intensive hardware is a lightweight neural network architecture. It was created for effective and precise picture classification on systems with limited resources, such as mobile phones. ShuffleNetV2 modulates input channels by applying grouped convolutions, which split the channels into subsets and perform distinct convolutional operations on each subset. This reduces the computational cost relative to standard convolutions.

- Abrupt decrease in the model's accuracy when the batch size is increased to 15K, indicating that there is a limit to how much the accuracy may be decreased and that doing so would render the model unsuitable for use in practical applications.



SqueezeNet model

- It is incredibly accurate model with a significantly less number of parameters.
- Uses "Fire Modules," which combine a squeeze layer (1x1 convolutional layer) and expand layers (1x1 and 3x3 convolutional layers), is one of its standout features.
- Here, it is evident that accuracy decreases noticeably beyond a certain batch size, indicating that batch size 5000 serves as the threshold in this instance. Additionally, the amount of weighted memory that we are able to preserve in these circumstances is quite large—nearly 280–290 Mb, which is noteworthy.



Observations

- Batch weight centralization can be universally applied to any model across diverse datasets by determining an optimal threshold batch size, ensuring its efficiency, effectiveness, and accuracy for the intended application.

- The versatility of Batch weight centralization is demonstrated by its observed generalizability, which enables it to perform as intended when trained, tested, and developed with an appropriate batch size threshold in mind.

- For mobileNet v2 we observed that out of these batch sizes [5, 10,15,20,50 , 100, 200 , 500 , 1000], 100 is most appropriate for maintaining the model's accuracy at a respectable level.

- However, if we desired significant or whole memory saving from the neural network, we might use this memory optimization for mostly dense layer networks, which includes multilayer perceptron type neural networks.

Impact of weight centralization on accuracy

The primary determining element for the algorithm's success is the correctness of the model following the use of the Batch-Wise Weight Centralization procedure. As a result, the effect of weight centralization on accuracy can be examined as illustrated 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

Memory size optimization

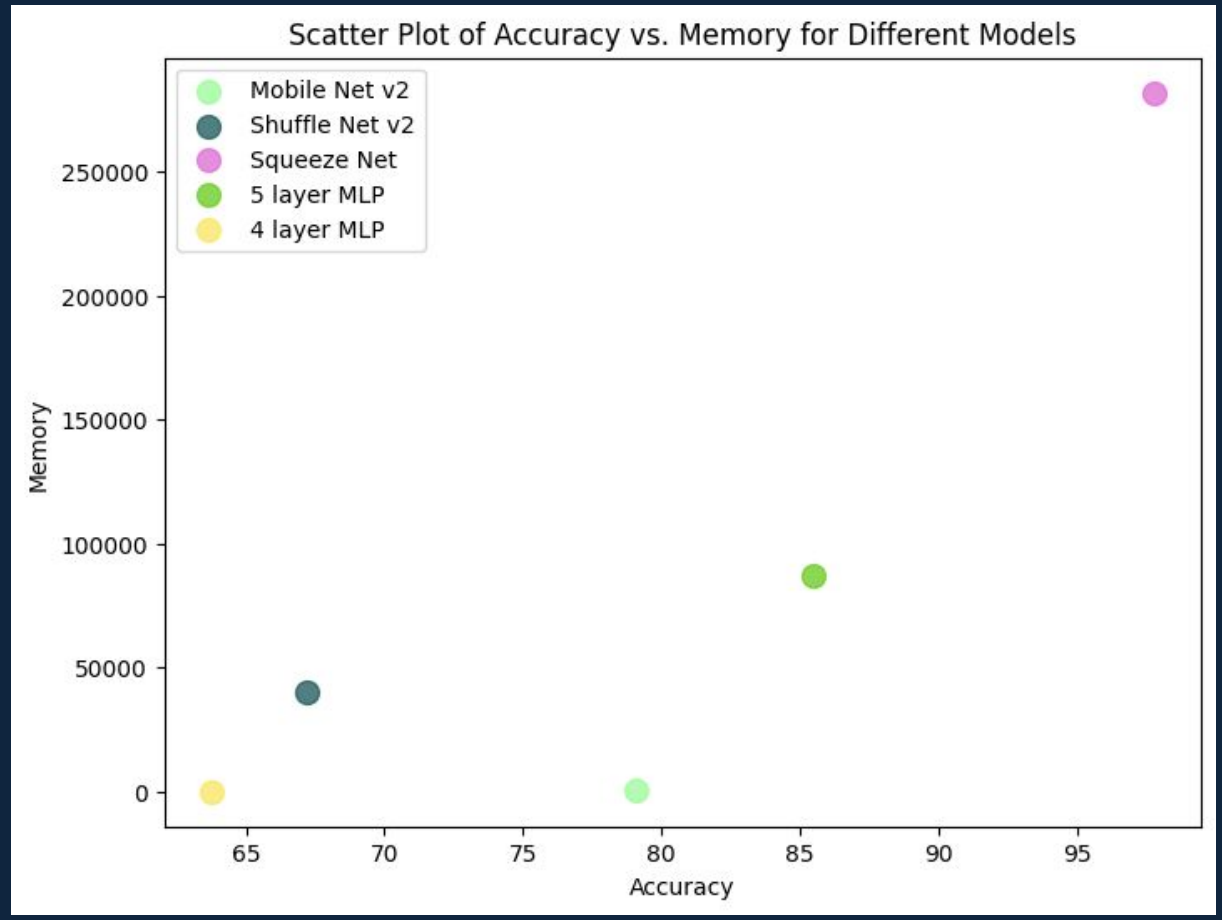
Another important component that determines how optimistic the approach is is how much memory is saved on the model once the Batch wise Weight centralization algorithm is done. Thus, it is possible to observe, as indicated below, how weight centralization affects the amount of memory saved.

	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

Accuracy VS Memory tradeoff

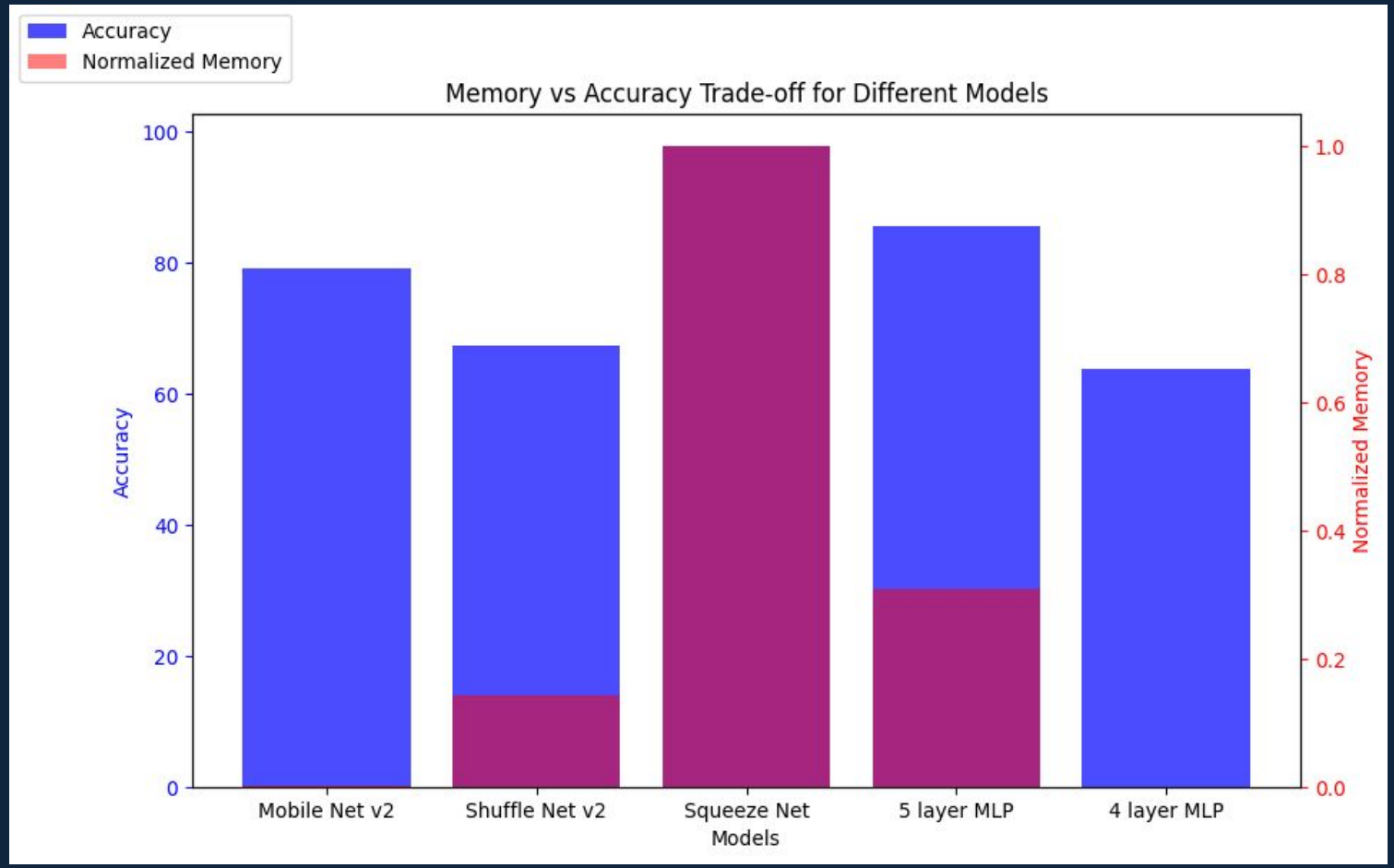
- When making trade-offs between accuracy and memory savings, both should be taken into account. Obtaining an estimate for each of the parameters is crucial. The trade-off may now be understood for different models using the following graph.

- Here in the graph it can be observed that for some data we have high accuracy as well as high memory savings while some might have low memory saving



Accuracy VS Memory Barplot

The bar plot shows which model performs best based on what we can see; the pinkish color indicates that accuracy and memory savings are both high, but in MNIST and 4 layer MLP, it is evident that accuracy is controlled and memory is not as highly optimized as it is in the other models.



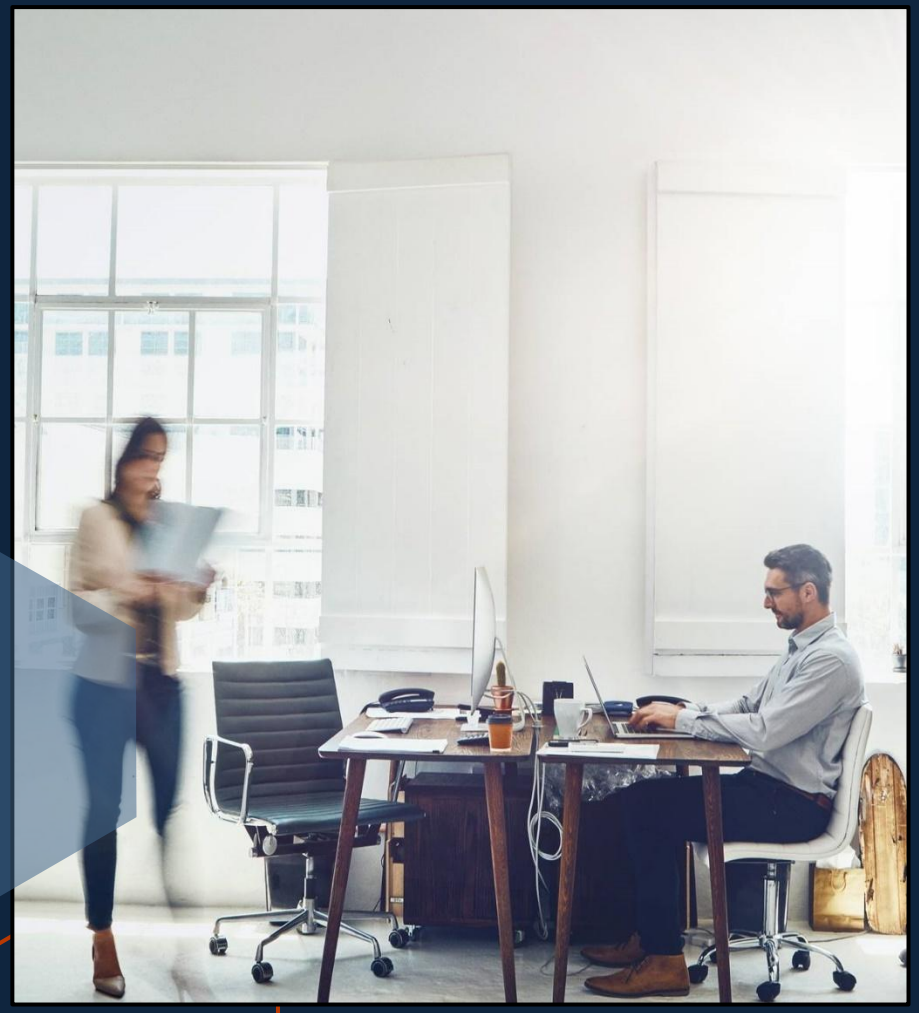
Conclusion

- Augmenting weights in initial layers has a more pronounced impact on accuracy compared to augmenting weights in the last layers in a layer-wise fashion.
- Batch-wise weight centralization proved to be a consistently effective method across various datasets and models, outperforming unitary weight centralization and layer-wise weight centralization, which demonstrated less consistent and generalizable accuracy improvements.
- The Batch-wise weight centralization optimization algorithm was successfully applied and tested on various MLP models, including standard and non-standard ones like MobileNet V2, ShuffleNet V2, and SqueezeNet V2, revealing varying degrees of memory conservation across different models.

Future plan

- Observing neural network weights, it's noted that many have zero magnitudes; leveraging this, omitting these weights in hardware operations by directly using bias values can enhance neural network efficiency and reduce computational load on the ALU section. This optimization improves hardware efficiency by avoiding unnecessary weight computations for zero-magnitude weights.

- This method is termed as **‘Zero Skipping’** where we are skipping zeros multiplication and saving the time for its calculation.






Thank you