Improvement on the Quaternion-based models: extension to larger datasets and Batch Normalization

Adhilsha A & Aritra Mukhopadhyay
Department of Computer Sciences
National Institute of Science Education and Research Bhubaneswar
P.O. Jatni, Khurda 752050, Odisha, India
adhilsha@niser.ac.in & aritra.mukhopadhyay@niser.ac.in

Abstract

Quaternion neural networks are particularly well-suited for image processing tasks because they can naturally represent complex color information in a compact and efficient way, making them a promising candidate for a variety of computer vision applications. The library used for such, contain experimental implementations which we are trying improve along with the implementation of Batch Normalization for larger datasets. As of now, we were able to significantly increase the speed of Quaternion model training by improving forward propagation without loss of accuracy.

1 Introduction

The baseline models in this project are all *Neural Networks* (NNs). Neural networks are machine learning algorithms inspired by the human brain that recognize patterns and relationships in data for making predictions or decisions. In this project, we will be more concerned with *forward propagation*, *backward propagation* and *batch normalization* (See Appendix) within neural networks to improve its speed.

We also have *Convolutional Neural Networks* (CNNs) are neural networks specifically designed for image recognition and computer vision tasks. They consist of multiple *convolutional layers*, along with *pooling layers*, activation functions, and fully connected layers (See Appendix). CNNs are capable of recognizing and classifying objects with a high degree of accuracy, widely used in applications such as self-driving cars, facial recognition, and medical imaging.

Another important factor in our goal is *Quaternions*, a four-dimensional extension of complex numbers, represented by a vector of the form $q = a + bi + cj + dk$. Quaternion models are models that use Quaternions as a mathematical representation. The most important characteristic of such a model is the reduction in the weights by 4 times although the amount of calculations remains the same (See Appendix). Quaternion neural networks are particularly well-suited for image processing tasks because they can naturally represent complex colour information in a compact and efficient way. This makes them a promising approach for a variety of computer vision applications.

2 Previous works

Our work is based on the work done by Sahel Mohammad Iqbal and Subhankar Mishra [1] on Pruning of Quaternion models along the lines of Lottery Ticket hypothesis. Their work showed that, in situations where resources are severely limited, a sparse Quaternion network may be a more suitable option than a sparse real model with similar structure. Pruning is a technique used in deep neural network training to minimize resource requirements by removing redundant weights. The

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

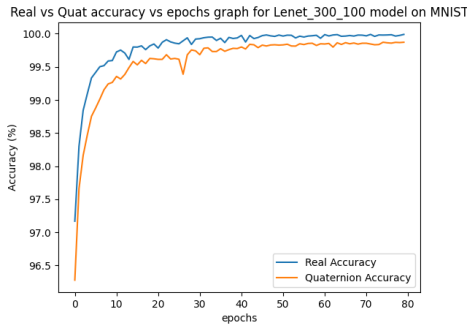

Figure 1: Real and Quaternion model accuracy vs epochs for LeNet-300-100 on MNIST

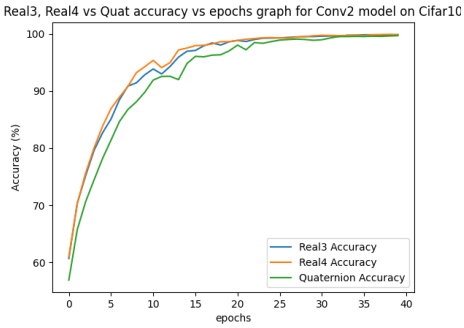

Figure 2: Real3, Real4 and Quaternion model accuracy vs epochs for Conv2 on CIFAR-10

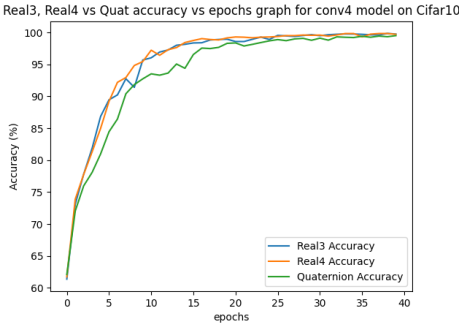

Figure 3: Real3, Real4 and Quaternion model accuracy vs epochs for Conv4 on CIFAR-10

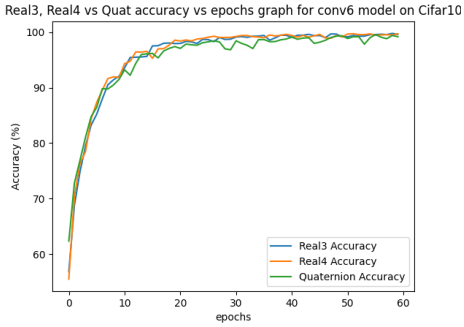

Figure 4: Real3, Real4 and Quaternion model accuracy vs epochs for Conv6 on CIFAR-10

Lottery Ticket Hypothesis suggests that there exists a smaller subnetwork that can perform as well as the full network. This subnetwork can be identified by pruning unimportant weights during the training process without affecting the performance of the overall model.

Furthermore, for machine learning tasks that have multi-dimensional input data, using more complex data embeddings such as Quaternions or complex numbers can reduce the number of parameters while maintaining precision. Their work showed that, in situations where resources are severely limited, a sparse Quaternion network may be a more suitable option than a sparse real model with similar structure.

3 Datasets and Baseline models

Till now, we have used leNet-300-100 model on the MNIST dataset and 3 sorts of CNN models (Conv2, Conv4 and Conv6) on the CIFAR-10 dataset. (see appendix A.4). As for the models, we have *LeNet-300-100* neural network for image classification tasks. It has three fully connected layers (input_nodes-300-100-output_nodes) with ReLU activation function. The model takes in a flattened 28x28 image and outputs a tensor of size 10 representing the probability distribution over 10 digit classes.

The Conv2, Conv4, and Conv6 models are convolutional neural networks designed for image classification tasks. They consist of multiple convolutional layers with ReLU activation functions, followed by max pooling to downsample the feature maps and reduce the spatial dimensions. The output of the convolutional layers is then passed through fully connected layers with ReLU activations before a final output layer with a specified number of classes or output channels. The main differences between these models lie in the number of convolutional layers, output channels, and size of kernels used (See Appendix A.5).

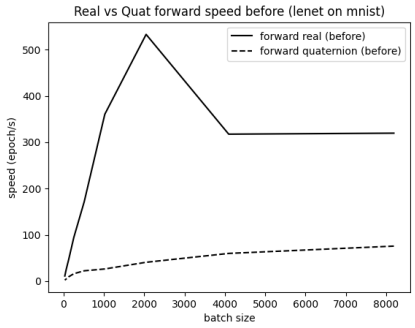

Figure 5: forward propagation speed vs batch size for LeNet model before code change

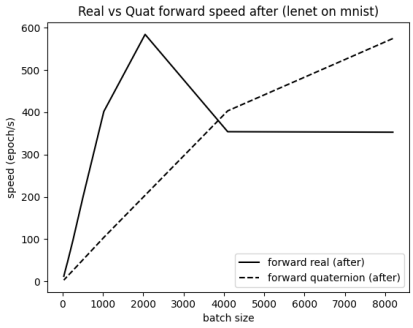

Figure 6: forward propagation speed vs batch size for LeNet model after code change

We trained the LeNet-300-100 model with MNIST and the CNNs with CIFAR-10. We used the hyperparameters from the original paper itself, which is batch size 60 for all models, 40 training epochs for all models except Conv6 whereas Conv6 had 60 training epochs. The Adam optimizer was used for all model but with different learning rates. While Conv4 and Conv6 has 0.0003 where as Conv2 used 0.0002 and LeNet-300-100 used 0.0012 as learning rates. The accuracy versus epochs graphs for these models are given in Figures 1-4.

The code and stored results can be found at the Github repository QuartLT23. This also includes the experiments and results discussed further in the report.

4 Experiments

4.1 Improvement on Quaternion forward propagation

The initial observation we saw was the slow running of Quaternion models. To trace the problem, we observed the forward and backward propagation time consumption separately for LeNet model on MNIST and found that the forward propagation in Quaternion models was taking most of the time, irrespective of batch size.

On further analysis, We found the three steps of forward propagation. They are:

1. building 4×4 Quaternion to real matrix (w)
2. Finding $wx + b$ (applying linear function)
3. typecasting the output of step 2 to a Quaternion tensor.

We checked the time taken by these three steps separately for batch size 8192. We found that step 1 took around $0.5\ ms$, step 2 took $1\ ms$ and step 3 took the rest of the $28.5\ ms$ out of the total $30\ ms$ taken by forward propagation. The typecasting step is taking around 95% of the time. This was because of a line `q.cpu()` which was needlessly copying the x to the CPU memory (the RAM) after every layer. Being a highly experimental part of library, we changed it to `q.cuda()` and got rid of the redundant operations. This improved the speed by almost double ($22.5\ it/s$ to $57.5\ it/s$). Repeating the batch size experiment again, we got the following before and after the change results as given in Figures 5 and 6.

4.2 Batch size experiment on Conv6 with CIFAR-10

We ran the Conv6 model with CIFAR-10 dataset for batch sizes ranging from 2^5 to 2^{13} with 80 training epochs and other hyper parameters same. Here, the forward propagation time comparison, backward propagation time comparison and accuracy comparison across varying batch sizes are given in Figures 7, 8 and 9. The forward propagation increases like a step function about which we still yet to uncover more info on. The backward propagation of Quaternion models outperformed the real models and this shows that the the reason of Quaternion model's slowness is solely from the forward propagation and improving it will bring about better speeds.

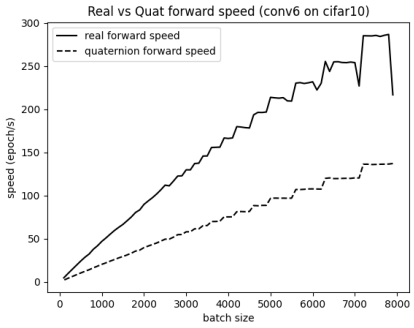

Figure 7: Real and Quaternion model forward propagation speed vs batch size for Conv6 model

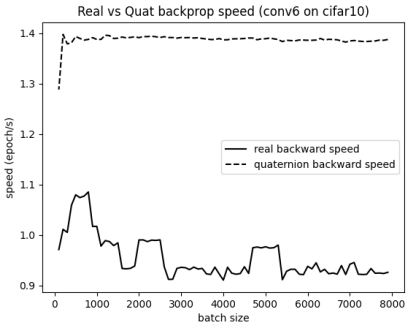

Figure 8: Real and Quaternion model backward propagation speed vs batch size for Conv6 model

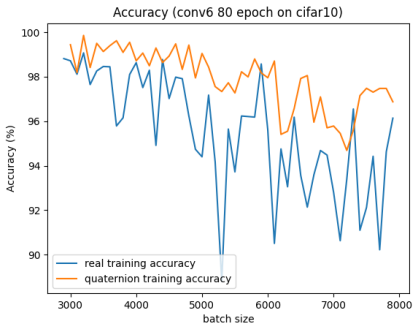

Figure 9: Real and Quaternion model accuracy vs batch size for Conv6 model

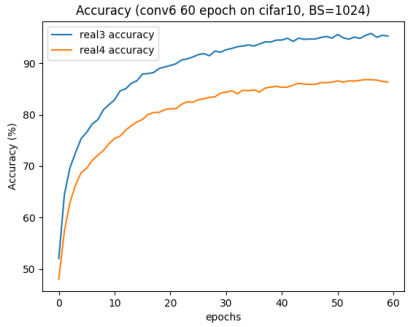

Figure 10: Real and Quaternion model accuracy vs epochs for Conv6 model, batch size 1024

Another interesting observation is the stability of accuracy across different batch sizes in Quaternion and real models. The Quaternion has a relative stable accuracy (around 3% change across batch size) while the real model is less stable (with a 9% change across batch size), see Figure 9.

4.3 Observation on batch size and accuracy on Conv6 Real3 and Real4

In Original work, Sahel [1] fed the RGB data as three channels and the grayscale data of the same image as the fourth one for Quaternion model input. He was comparing this with the real model trained on only RGBdata. This comparison showed the Quaternion model to be less accurate than the real model. So, to have a fair comparison of models, we ran the real Conv6 model with RGBdata+grayscale data (termed as *Real4* and the former as *Real3*) of images and compared, then both the models were of comparable accuracy. See Figure 4.

With an experiment of an increase in batch size to 1024, we saw the same models having a reduction in accuracy, worse in the case of Real4. See Figure 10. Though the reduction in accuracy is as expected, as we already saw from Krizhevsky [2014], Li et al. [2014], Keskar et al. [2016] and Hoffer et al. [2017] as cited in [2]. the Real4 model taking worse of it was unexpected; we plan to run more analysis on this too.

5 Further plans

Our future plans include writing the code for Batch Normalization code in Quaternion models. Along with the implementaion of batch normalization in more complex models and larger datasets, we plan to improve the speed of the code by possibly editing or writing alternate codes for the implemenations inside Quaternion models.

References

[1] Sahel Mohammad Iqbal and Subhankar Mishra. 2023. Neural Networks at a Fraction with Pruned Quaternions. In *6th Joint International Conference on Data Science Management of Data (10th ACM IKDD CODS and 28th COMAD) (CODS-COMAD 2023), January 4–7, 2023, Mumbai, India*. ACM, New York, NY, USA 9 Pages. DOI:https://doi.org/10.1145/3570991.3570997

[2] Yang You, Igor Gitman, and Boris Ginsburg. 2017. Large Batch Training of Convolutional Networks. arXiv. DOI:https://doi.org/10.48550/ARXIV.1708.03888

A Appendix

A.1 Forward propagation, Backward propagation and Batch normalization

Forward propagation in neural networks involves feeding input data through the network to obtain a prediction, where weights and biases are adjusted during training. *Backward propagation* involves calculating error and adjusting weights and biases to minimize error between predicted and actual output.

Batch normalization is a machine learning technique that improves neural network performance and stability by normalizing the input of each layer. It can be done by subtracting the batch mean and dividing by the batch standard deviation to reduce the internal covariate shift. The technique can speed up training and improve generalization performance in deep learning architectures like CNNs and RNNs.

A.2 Convolutional layers and Pooling layers

Convolution layers are used in neural networks to detect and extract features from data such as images. They use a filter to perform a convolution operation at each location, resulting in a feature map that captures specific features within the input data. Convolution layers are ideal for tasks such as image recognition and natural language processing where they can learn to detect patterns and relationships. *Pooling layers* downsample feature maps in neural networks by dividing input data into smaller regions and taking the maximum, minimum, or average value within each region. This reduces feature map size, preserves important information, and commonly used in convolutional neural networks to improve efficiency and reduce overfitting.

A.3 Quaternions and Quaternion based models

Quaternions are a four-dimensional extension of complex numbers, represented by a vector of the form:

$$q = a + bi + cj + dk$$

where a, b, c, and d are real numbers, and i, j, and k are imaginary units that satisfy the following rules:

$$i^2 = j^2 = k^2 = ijk = -1$$

Quaternion models are models that use Quaternions as a mathematical representation. The most important characteristic of such a model is the reduction in the weights by 4 times although the amount of calculations remains the same (See Appendix). Quaternion neural networks are particularly well-suited for image processing tasks because they can naturally represent complex color information in a compact and efficient way. This makes them a promising approach for a variety of computer vision applications.

The Hamilton product of two Quaternions is defined as:

$$q_1 q_2 = (a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2) + (a_1 b_2 + b_1 a_2 + c_1 d_2 - d_1 c_2)i$$

$$+(a_1c_2 - b_1d_2 + c_1a_2 + d_1b_2)j + (a_1d_2 + b_1c_2 - c_1b_2 + d_1a_2)k$$

where $q_1 = a_1 + b_1i + c_1j + d_1k$ and $q_2 = a_2 + b_2i + c_2j + d_2k$.

The 4x4 real representation matrix of a Quaternion is:

$$q = \begin{pmatrix} a & -b & -c & -d \\ b & a & -d & c \\ c & d & a & -b \\ d & -c & b & a \end{pmatrix}$$

Traditional neural networks are built with real numbers, where each neuron is a real number and each weight is a real number. For such networks, $p \times n$ weights are required in each layer (where p is the number of weights in the previous layer and n is the number of weights in the next layer). However, in Quaternion networks, every four neurons in a layer can be replaced with one Quaternion to create an equally complex network. Therefore, only $\frac{p}{4} \times \frac{n}{4}$ weights are needed. It should be noted that all of these weights must be Quaternions rather than real numbers. To store a Quaternion, four numbers are required, resulting in a total number of weights of $4 \times \frac{p}{4} \times \frac{n}{4} = \frac{p \times n}{4}$. Thus, using Quaternions, we achieve a four-fold reduction in weight. An important point to be noted is that, although the number of weights get reduced by 4, the number of calculation during training remains constant.

A.4 Datasets and models

The *MNIST dataset* comprises 70,000 grayscale images of handwritten digits (0-9), each of size 28x28 pixels, and is a widely used benchmark dataset in machine learning and computer vision research. The dataset is divided into a training set of 60,000 images and a test set of 10,000 images, and its goal is to train a machine learning algorithm to correctly classify digits based on their pixel values.

The *CIFAR-10 dataset* is a widely used benchmark dataset in computer vision research, consisting of 60,000 color images of size 32x32 pixels, each belonging to one of ten classes. It is used to evaluate the performance of various image classification algorithms, including CNNs, and is divided into a training set of 50,000 images and a test set of 10,000 images.

As for the models, we have *LeNet-300-100* neural network for image classification tasks. It has three fully connected layers (input_nodes-300-100-output_nodes) with ReLU activation function. The model takes in a flattened 28x28 image and outputs a tensor of size 10 representing the probability distribution over 10 digit classes.

The Conv2, Conv4, and Conv6 models are convolutional neural networks designed for image classification tasks. They consist of multiple convolutional layers with ReLU activation functions, followed by max pooling to downsample the feature maps and reduce the spatial dimensions. The output of the convolutional layers is then passed through fully connected layers with ReLU activations before a final output layer with a specified number of classes or output channels. The main differences between these models lie in the number of convolutional layers, output channels, and size of kernels used. Overall, these models are effective in extracting meaningful features from images and achieving high accuracy in classification tasks.