Pruning a neural network using Bayesian inference

Sunil Mathew
Department of Mathematical and Statistical Sciences, Marquette University
and
Daniel B. Rowe
Department of Mathematical and Statistical Sciences, Marquette University

July 21, 2023

Abstract

Neural network pruning is a highly effective technique aimed at reducing the computational and memory demands of large neural networks. In this research paper, we present a novel approach to pruning neural networks utilizing Bayesian inference, which can seamlessly integrate into the training procedure. Our proposed method leverages the posterior probabilities of the neural network prior to and following pruning, enabling the calculation of Bayes factors. The calculated Bayes factors guide the iterative pruning. Through comprehensive evaluations conducted on multiple benchmarks, we demonstrate that our method achieves desired levels of sparsity while maintaining competitive accuracy.

Keywords: Bayesian model selection, Bayes Factors, Bayesian pruning schedule

1 Introduction

In artificial neural networks (ANN) and machine learning (ML), parameters represent what the network has learned from the data. The number of parameters in a neural network can determine its capacity to learn. With advancements in hardware capabilities, we can now define larger models with millions of parameters. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) and its winners over the years demonstrate how the error rate has decreased with an increase in the number of parameters and connections in neural networks. For instance, in 2012, AlexNet (Krizhevsky et al., 2012), one of the convolutional neural networks (CNNs), had 660K nodes, 61M parameters, and over 600M connections. The large language model, Generative Pre-trained Transformer 3 (GPT-3) (Brown et al., 2020), comprises 175 billion parameters. Even though deep neural networks with large number of parameters capture intricate underlying patterns, the large number of connections can introduce computational challenges, overfitting, and lack of generalizability. To address these issues, various methods have been developed.

Neural network pruning is a widely used method for reducing the size of deep learning models, thereby decreasing computational complexity and memory footprint (LeCun et al., 1989; Han et al., 2015; Liu et al., 2018). Pruning is crucial for deploying large models on resource-constrained devices such as personal computers, mobile phones and tablets. Pruning can also be used to reduce the carbon footprint of deep learning models by reducing the computational requirements (Strubell et al., 2019). Pruning can also be used to improve the interpretability of deep learning models by removing redundant neurons or connections (Han et al., 2015).

Pruning methods can be classified into mainly three categories, weight pruning, neuron pruning, and filter pruning (Han et al., 2015; Srivastava et al., 2014; Li et al., 2017; He et al., 2018). Weight pruning involves removing individual weights from the network based on their magnitude or other criteria, neuron pruning and filter pruning involve removing entire neurons or filters that are not important. Even though pruning methods can effectively reduce network size and improve performance, they often lack a principled approach for

selecting the most important weights or neurons (Blalock et al., 2020).

In Bayesian neural networks, the weights of the network are treated as random variables with a prior distribution, which can be updated to get a posterior distribution using Bayes' rule. It allows us to quantify the uncertainty associated with each weight and select the most important weights based on their relevance to the task the network is being trained for. The posterior distribution reflects our updated belief about the weights based on the observed data and can be used to calculate the probability of each weight being important for the task at hand. Variational inference, which involves minimizing the Kullback-Leibler (KL) divergence between the true posterior and an approximate posterior, is a common approach for approximating the posterior distribution for neural network pruning (Dusenberry et al., 2019; Blundell et al., 2015). Other approaches include Monte Carlo methods and Markov chain Monte Carlo (MCMC) sampling (Molchanov et al., 2019). However, these methods are computationally expensive and can prove to be difficult to be scaled to large networks.

In this work, we propose a Bayesian pruning algorithm based on Bayesian hypothesis testing. It provides a principled approach for pruning a neural network to a desired size without sacrificing accuracy. We compare two neural network models at every training iteration, the original unpruned network, and the pruned network. This comparison helps us to determine which model fits the data better. The ratio of the posterior probabilities of the pruned network to the posterior probabilities of the unpruned network (Bayes factor) can be then used to determine whether to prune the network further or skip pruning at the next iteration. This approach enables us to implement this method in regular neural networks without the need for additional parameterization as in the case of Bayesian neural networks.

1.1 Pruning Neural Networks using Bayesian Inference

The pruning system, seen in Figure 1, incorporates pruning into the training process. The training data is divided into batches and processed by the neural network through a forward pass, consisting of matrix multiplications and non-linear activations. The network's output

is then compared with the ground truth labels to compute the loss. The weights of the network is adjusted through a backward pass using an optimizer such as Stochastic Gradient Descent (SGD) or Adam (Kingma and Ba, 2015). After each epoch, the weights are pruned using the pruning algorithm, and the pruned weights are used in the subsequent epochs. The pruning algorithm is based on Bayesian hypothesis testing, which is a statistical framework that can be used to compare two models, two network configurations in this case, to determine which one fits the data better.

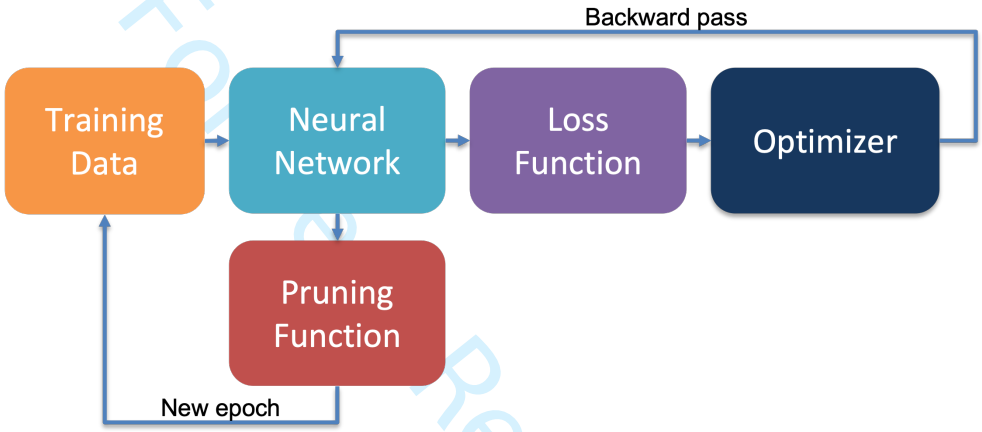

Figure 1: Pruning system block diagram.

To test the hypothesis that the pruned network fits the data better than the unpruned network, we define the null hypothesis as the unpruned network fitting the data better ($\theta = \psi$) and the alternative hypothesis as the pruned network fitting the data better ($\theta = \phi$). The Bayes factor, which is the ratio of the posterior probability of the alternative hypothesis to the posterior probability of the null hypothesis, is computed as follows:

$$\text{Bayes factor} = \frac{P(\theta = \phi|D)}{P(\theta = \psi|D)}$$

Here, D represents the training data.

The posterior probability of the null hypothesis ($P(\theta = \psi|D)$) is computed as:

$$P(\theta = \psi|D) = \frac{P(D|\theta = \psi)P(\theta = \psi)}{P(D)}$$

Similarly, the posterior probability of the alternative hypothesis ($P(\theta = \phi|D)$) is computed as:

$$P(\theta = \phi|D) = \frac{P(D|\theta = \phi)P(\theta = \phi)}{P(D)}$$

The Bayes factor is then calculated as the ratio of the posterior probabilities:

$$\text{Bayes factor} = \frac{P(D|\theta = \phi)P(\theta = \phi)}{P(D|\theta = \psi)P(\theta = \psi)}$$

A Bayes factor greater than 1 indicates that the pruned network fits the data better, while a value less than 1 indicates that the unpruned network fits the data better.

For a classification problem, the likelihood of the data is given by the categorical cross-entropy loss function:

$$\log p(y_{pred}|y_{true}) = \log \mathcal{C}(\text{softmax}(y_{pred}))y_{true}$$

Here, y_{pred} represents the neural network's predictions for the classes, and y_{true} is the ground truth. A Gaussian prior with mean μ and variance σ^2 is used for weights:

$$p(w) = \mathcal{N}(\mu, \sigma^2)$$

The log prior and log likelihood for the weight parameters are used to compute the log posterior distribution of the weights:

$$\log p(w|D) = \log p(D|w) + \log p(w)$$

The log posterior is calculated before and after weight pruning to compute the Bayes factor. If the Bayes factor exceeds a predefined threshold, a certain percentage (r) of the weights are pruned as,

$$w_{\text{new}} = w_{\text{old}} \odot m \tag{1}$$

where $\odot$ represents element-wise multiplication, w_{old} is the old weight matrix, and m is the binary mask indicating which weights should be pruned (i.e., have a value of 0) and which weights should be kept (i.e., have a value of 1). The resulting matrix w_{new} has the same dimensions as w_{old}, but with some of its weights pruned. Algorithm 1 outlines the Bayesian pruning process.

Algorithm 1 Bayesian Pruning Algorithm

Input: Trained neural network $f(\cdot, w)$, pruning rate r, dataset $\mathcal{D} = (\mathbf{x}i, y_i)_{i=1}^{n}$, β Bayes factor threshold Output: Pruned neural network $f_r(\cdot, w)$

Compute the posterior probability of the weights before pruning

if $BF_{01} > \beta$ **then**

Prune r percentage of weights of $f(\cdot, w)$

Return $f_r(\cdot, w)$

end if

Compute the posterior probability of the weights after pruning

Compute the Bayes factor using the posterior probabilities before and after pruning

In the following sections, we introduce two pruning algorithms that utilize this framework: random pruning, which randomly selects weights for pruning, and magnitude pruning, which prunes weights based on their magnitude.

Bayesian Random pruning

Random pruning is a simple pruning algorithm that randomly selects weights to prune. Here we set the pruning rate to be the desired level of sparsity that we are looking to achieve. After an epoch, we count the number of non-zero parameters in the network and randomly zero out just enough parameters to achieve the desired level of sparsity. The algorithm is summarized in Algorithm 2.

Algorithm 2 Bayesian Random Pruning

$f(\cdot, w)$: Neural network model with parameters w
r: Desired sparsity level, β Bayes factor threshold
Calculate log posterior probability $p(w|\mathcal{D})$
if $BF_{01} > \beta$ **then**
    for all weights $w_i \in w$ **do**
        $n \leftarrow \text{size}(w_i)$
        number of weights to prune, $k \leftarrow (n \times r)$
        $I \leftarrow$ indices of non zero weights
        $n_z \leftarrow$ number of zero weights
        $k' \leftarrow k - n_z$
        $J \leftarrow \text{random\_sample}(I, k')$
        set elements in w_i at indices J to zero
    end for
end if
Calculate log posterior probability $p(w|\mathcal{D})$ after pruning
Calculate Bayes factor BF_{01}

Bayesian Magnitude pruning

Magnitude pruning is a pruning algorithm that selects weights to prune based on their magnitude. This can be seen as pruning weights that are less important. Here we set the pruning rate to be the desired level of sparsity that we are looking to achieve. The lowest weights corresponding to the desired level of sparsity is pruned to get the pruned network. The algorithm is summarized in Algorithm 3.

Algorithm 3 Bayesian Magnitude Pruning

$f(\cdot, w)$: Neural network model with parameters w
r: Desired sparsity level, β Bayes factor threshold
Calculate log posterior probability $p(w|\mathcal{D})$
if $BF_{01} > \beta$ **then**
    for all weights $w_i \in w$ **do**
        $n \leftarrow \text{size}(w_i)$
        number of weights to prune, $k \leftarrow (n \times r)$
        $w_i \leftarrow sort(w_i)$
        set k elements in w_i to zero
    end for
end if
Calculate log posterior probability $p(w|\mathcal{D})$ after pruning
Calculate Bayes factor BF_{01}

Experimental Setup

To evaluate the performance of Bayesian Random Pruning and Bayesian Magnitude Pruning, we conduct experiments on three datasets and two neural network architectures for five different levels of desired sparsity. The datasets used arne MNIST (Lecun et al., 1998), MNIST Fashion (Xiao et al., 2017) and CIFAR-10 (Krizhevsky, 2009). The neural network architectures are a Fully Connected Network (FCN) and a Convolutional Neural Network (CNN). The five different levels of sparsity are 25%, 50%, 75%, 90% and 99%. We use a learning rate of 0.001 and a batch size of 64 for all experiments. Data preprocessing only consist of normalizing the dataset and does not include any data augmentation like Random cropping or flipping of images to have less confounding variables in the studies we conduct to observe the effects of our pruning algorithm. We train the network for 25 epochs on the training set and evaluate its performance on the test set. We evaluate the performance of each method in terms of the accuracy of predictions it makes for the target classes using the test set. Each experiment is repeated 5 times and the mean and standard deviation of the accuracy is reported.

The following sections describe the neural network architectures used in our experiments.

Neural Network Architectures

The two neural network architectures used in our experiments are the Fully Connected Network (FCN) and the Convolutional Neural Network (CNN). The same architectures are used for all three datasets. The FCN consists of two hidden layers. The output of the last fully connected layer is fed into a softmax layer to get the class probabilities. The CNN consists of two convolutional layers with 32 and 64 filters respectively followed by two fully connected layers. Each convolutional layer is followed by a max pooling layer with a kernel size of 2 and stride of 2. The output of the second max pooling layer is flattened and fed to the fully connected layers. The output of the fully connected layer is fed into a softmax layer to get the class probabilities.

The network architecture of the fully connected network (FCN) is seen in Figure 2.

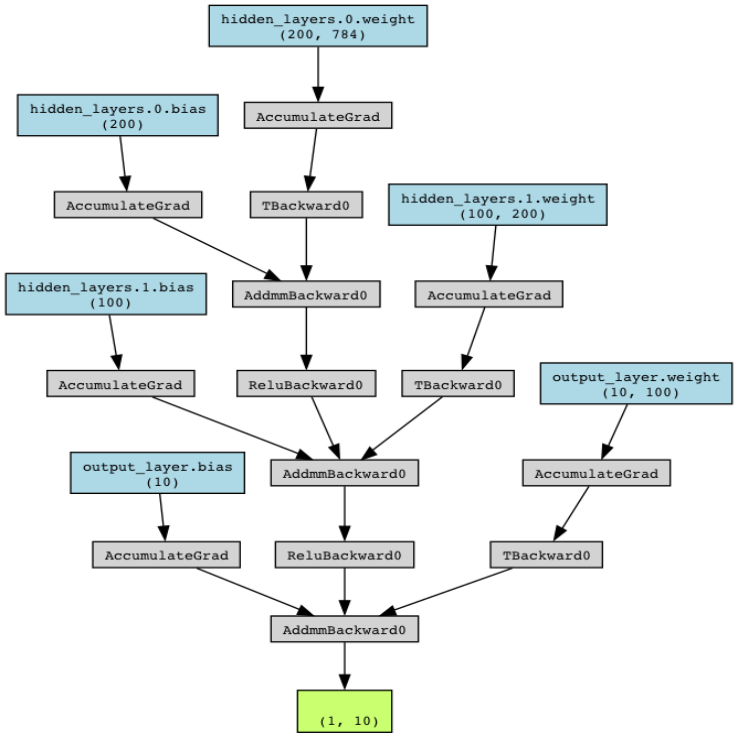

Figure 2: Fully connected neural network architecture

The network architecture of the convolutional neural network (CNN) is seen in Figure 3.

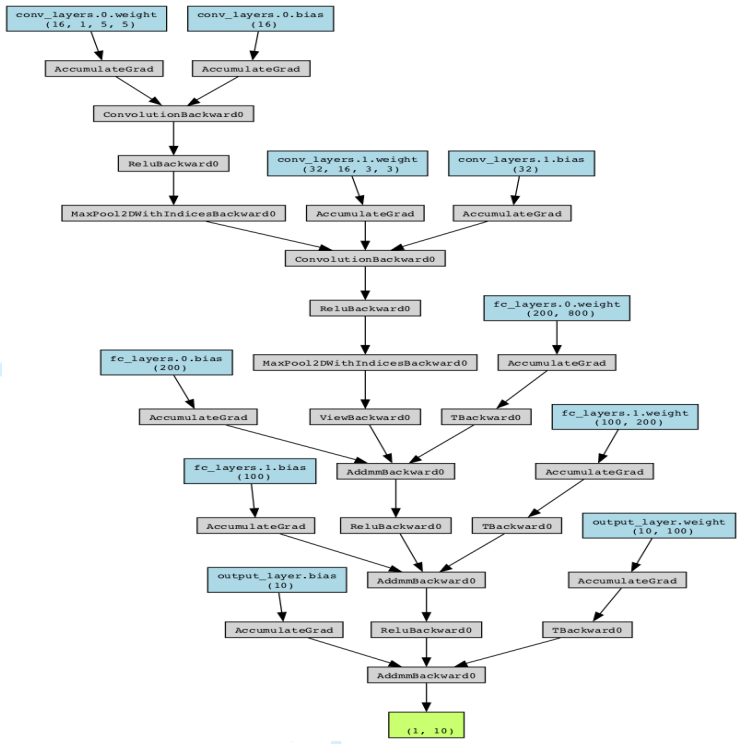

Figure 3: Convolutional neural network architecture

1.2 Results

The following sections present the results of the experiments. The results are presented in the following order: (1) MNIST dataset, (2) MNIIST-Fashion dataset, and (3) CIFAR-10 dataset. The results are presented in the form of learning curves and a table with accuracy for different levels of sparsity for the FCN and CNN model. The accuracy is the percentage of correctly classified images in the test set. The sparsity is the percentage of weights that are pruned in the network. The results are compared to baseline, which is the model trained without pruning, and the non-Bayesian version of the pruning method.

MNIST

Figure 4 shows the learning curves for random pruning, magnitude pruning under a Bayesian framework compared to baseline in a fully connected network (FCN) trained on the MNIST

dataset. Here the desired level of sparsity is 75%. The figure has two subplots. One shows the training and validation loss as a function of the number of epochs, the other plot (right) shows the Bayes factor, sparsity as a function of the number of epochs.

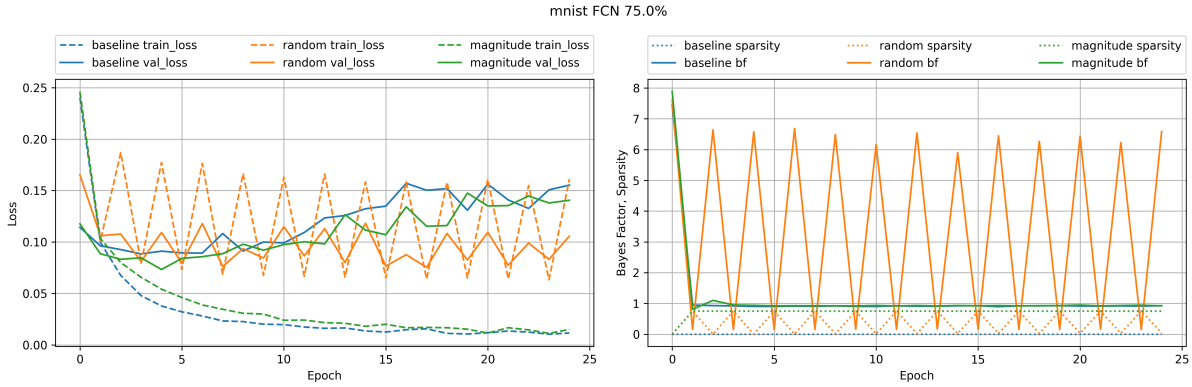

Figure 4: MNIST (FCN 75%) learning curves for the Bayesian pruning method.

The training loss is the average loss over the training set, and the validation loss is the average loss over the validation set. The figure shows that the training loss decreases as the number of epochs increases, and the validation loss starts to decrease in about 5 epochs. The training loss decreases faster than the validation loss, which indicates that the model is overfitting the training data. As pruning begins, it affects the training and validation loss of both random and magnitude pruning as seen the curves. There are large oscillations in loss values for random pruning as seen in the figure. The Bayes factor begins to reduce as the number of epochs increases and the sparsity of the network becomes stabilized for magnitude pruning, but it remains fluctuating for random pruning and shows an increasing trend for the Bayes factor suggesting that Bayesian random pruning fits the data better than other methods.

Figure 5 shows the validation accuracy of random pruning for different sparsity levels. For 25% sparsity the validation accuracy seems to be the highest. Then as the sparsity level increases the validation accuracy begins to decrease. Until 90% sparsity the validation accuracy remains to have a downward trend and combats overfitting compared to the baseline. The network only starts to become worse at 99% sparsity.

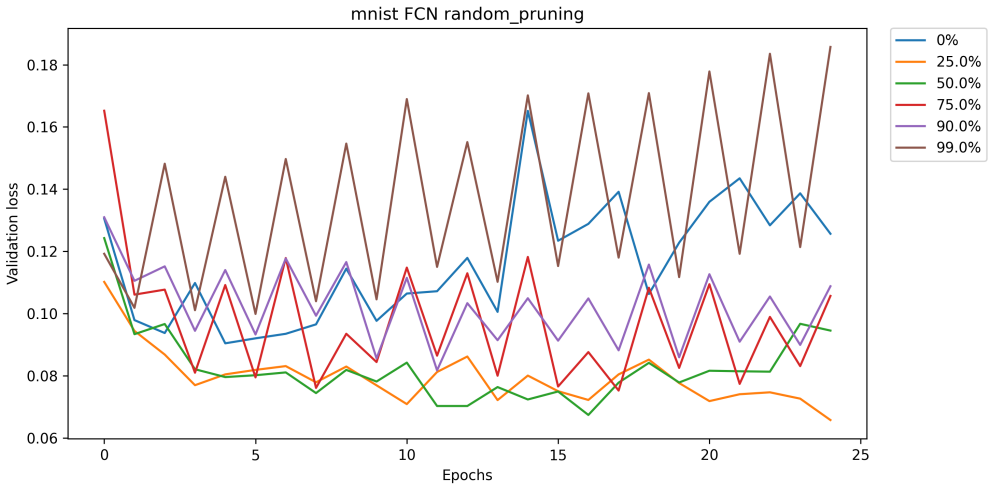

Figure 5: Validation loss of random pruning for different sparsity levels.

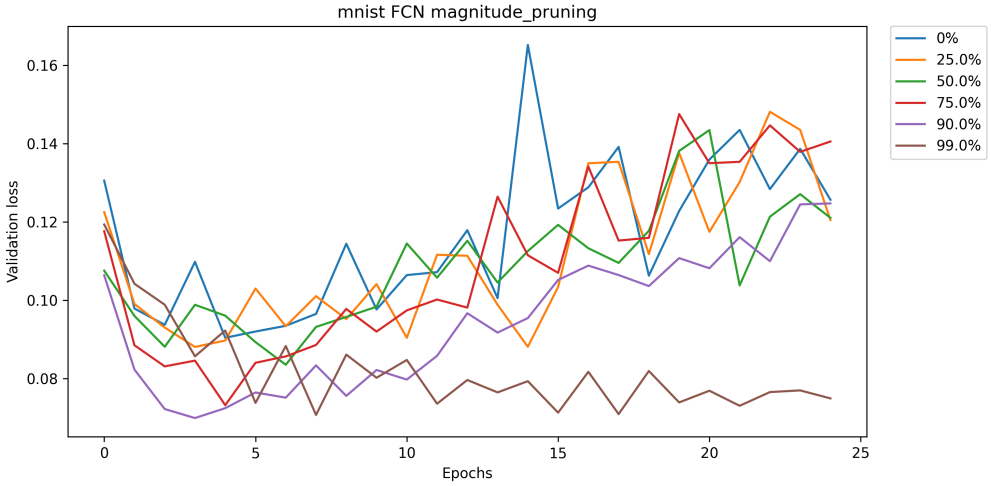

Figure 6: Validation loss of magnitude pruning for different sparsity levels.

Figure 6 shows the validation accuracy of magnitude pruning for different sparsity levels. For 25% sparsity the validation accuracy remains similar to the baseline. Then as the sparsity level increases the validation accuracy starts to improve, but the network still overfits the data until 99% of the parameters are pruned.

Figure 7 shows the learning curves for random pruning, magnitude pruning under a Bayesian framework compared to baseline in a convolutional neural network (CNN) trained on the MNIST dataset. The number of parameters in the CNN are comparatively larger than that

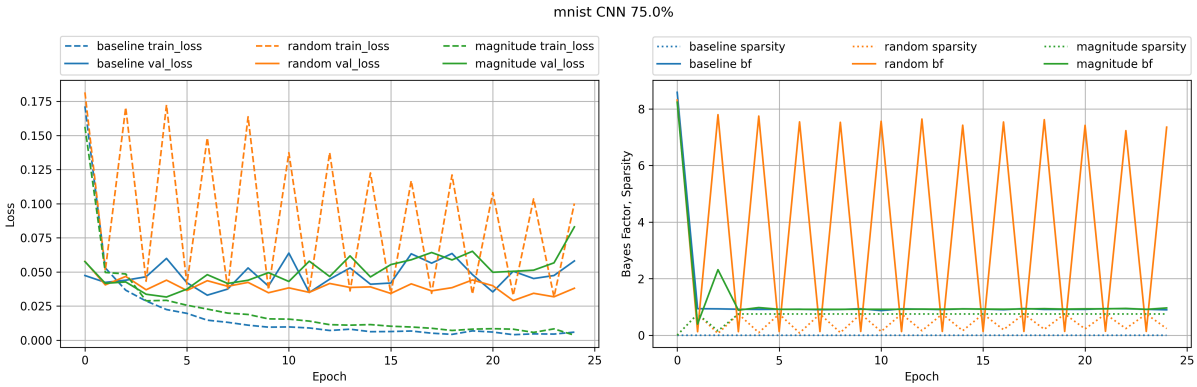

Figure 7: MNIST (CNN 75%) learning curves for the Bayesian pruning method.

of the FCN. This causes the effects of overfitting to be seen a little later in the training period and less overfitting compared to the FCN at 75% sparsity. Bayes factor for random pruning is higher than that of magnitude pruning, which suggests that Bayesian random pruning fits the data better.

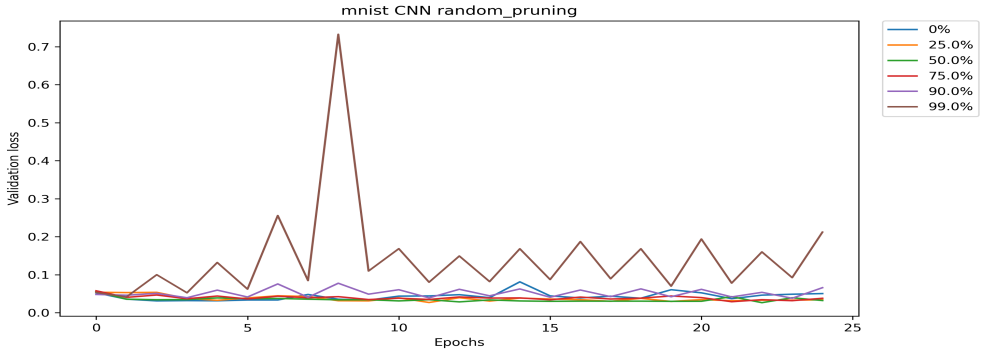

Figure 8: Validation loss of random pruning for different sparsity levels.

Figure 8 shows the validation accuracy of random pruning for different sparsity levels. As the number of parameters of the CNN is larger than that of the FCN, the validation accuracy remains similar to the baseline until 90% sparsity. Then as the sparsity level increases the validation accuracy begins to decrease.

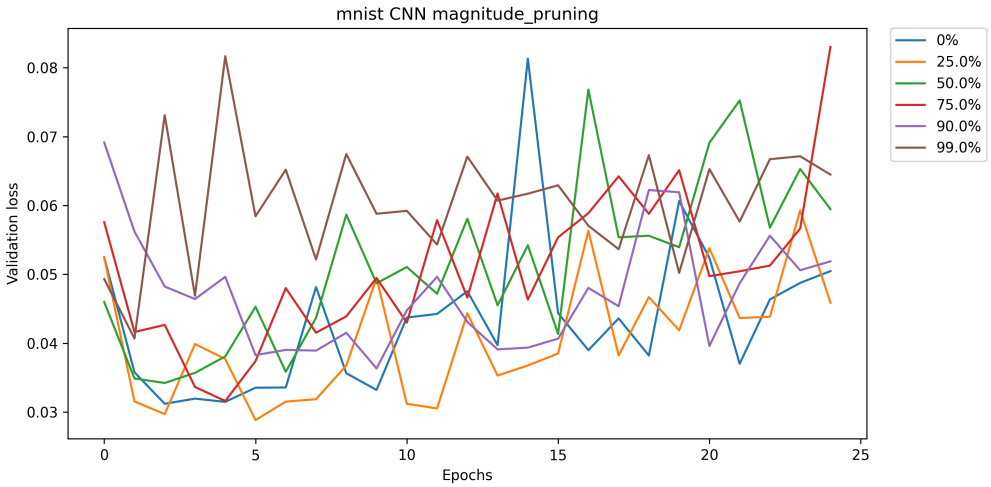

Figure 9: Validation loss of magnitude pruning for different sparsity levels.

Figure 9 shows the validation accuracy of magnitude pruning for different sparsity levels. Even pruning 99% of the parameters does not affect the validation accuracy of the CNN. This is because the CNN has an enormous number of parameters and the network overfits the data even after pruning 99% of the parameters.

MNIST Fashion

Figure 10 shows the learning curves for random pruning, magnitude pruning under a Bayesian framework compared to baseline in a fully connected network (FCN) trained on the MNIST Fashion dataset. Here the desired level of sparsity is 90%. The figure has two subplots. One shows the training and validation loss as a function of the number of epochs, the other plot (right) shows the Bayes factor, sparsity as a function of the number of epochs.

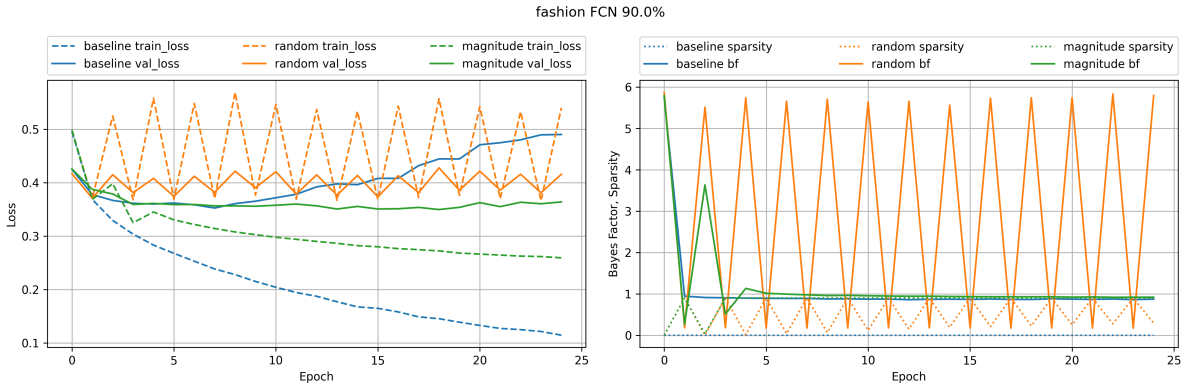

Figure 10: MNIST-Fashion (FCN 90%) learning curves for the Bayesian pruning method.

The training loss is the average loss over the training set, and the validation loss is the average loss over the validation set. The figure shows that the training loss decreases as the number of epochs increases, and the validation loss starts to decrease in about 5 epochs. The training loss decreases faster than the validation loss, which indicates that the model is overfitting the training data. As pruning begins, it affects the training and validation loss of both random and magnitude pruning as seen the curves. There are large oscillations in loss values for random pruning. The Bayes factor begins to reduce as the number of epochs increases and the sparsity of the network becomes stabilized for magnitude pruning, but it remains fluctuating for random pruning. Bayesian random pruning model fits the data better than magnitude pruning model.

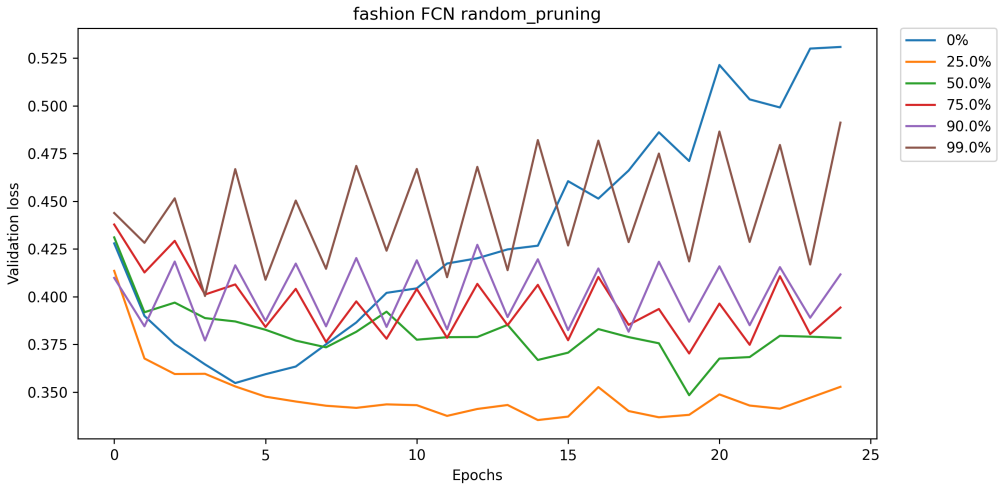

Figure 11: Validation loss of random pruning for different sparsity levels.

Figure 11 shows the validation accuracy of random pruning for different sparsity levels. Similar to the MNIST dataset, the validation loss is the lowest for 25% sparsity. Then as the sparsity level increases the validation accuracy begins to decrease.

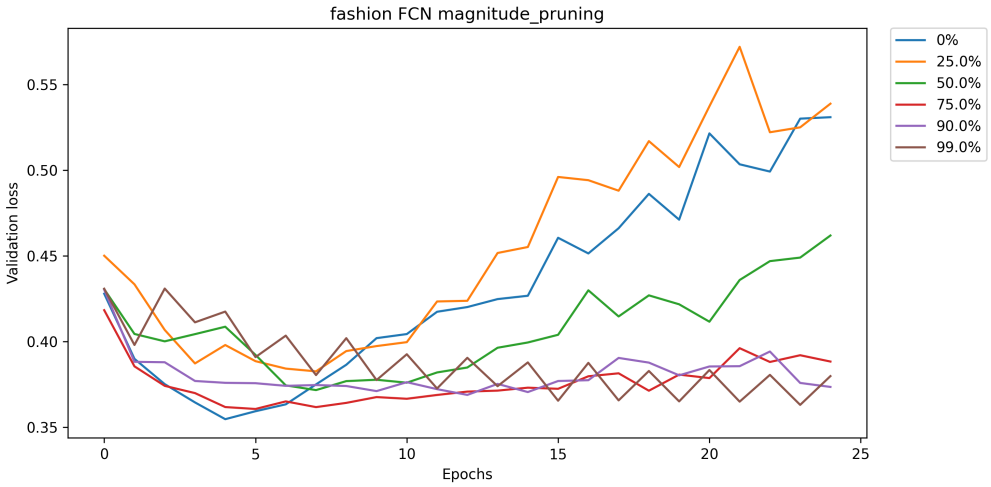

Figure 12: Validation loss of magnitude pruning for different sparsity levels.

Figure 12 shows the validation accuracy of magnitude pruning for different sparsity levels. Higher levels of sparsity improves the validation accuracy of the FCN. The effects of overfitting are reduced as the number of parameters are reduced.

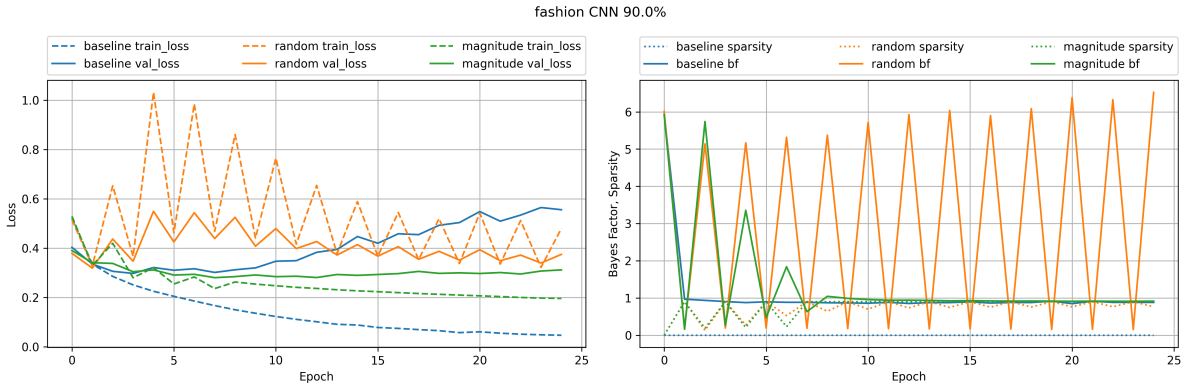

Figure 13: MNIST-Fashion (CNN 90%) learning curves for the Bayesian pruning method.

Figure 13 shows the learning curves for random pruning, magnitude pruning under a Bayesian framework compared to baseline in a convolutional neural network (CNN) trained on the MNIST Fashion dataset. Here the desired level of sparsity is 90%. The figure has two subplots. One shows the training and validation loss as a function of the number of epochs, the other plot (right) shows the Bayes factor, sparsity as a function of the number of epochs.

The number of parameters in the CNN are comparatively larger than that of the FCN. This causes the effects of overfitting to be seen a little later in the training period. The trends in the learning curves are similar to that of the FCN. The validation accuracy for random pruning decreases at the beginning of training and starts to improve as training progresses. The Bayes factor begins to reduce as the number of epochs increases and the sparsity of the network becomes stabilized for magnitude pruning, but it remains fluctuating for random pruning and shows an increasing trend for the Bayes factor. Bayesian random pruning model fits the data better than magnitude pruning model.

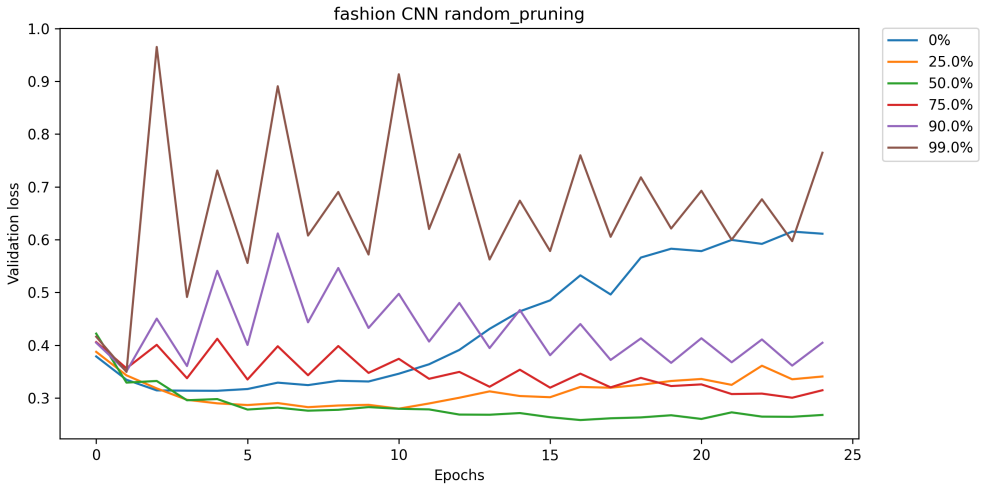

Figure 14: Validation loss of random pruning for different sparsity levels.

Figure 14 shows the validation accuracy of random pruning for different sparsity levels. The trends are similar to the MNIST dataset. The validation accuracy is better for 25% sparsity and decreases as the sparsity level increases. Sparsity levels up to 90% helps in reducing the effects of overfitting.

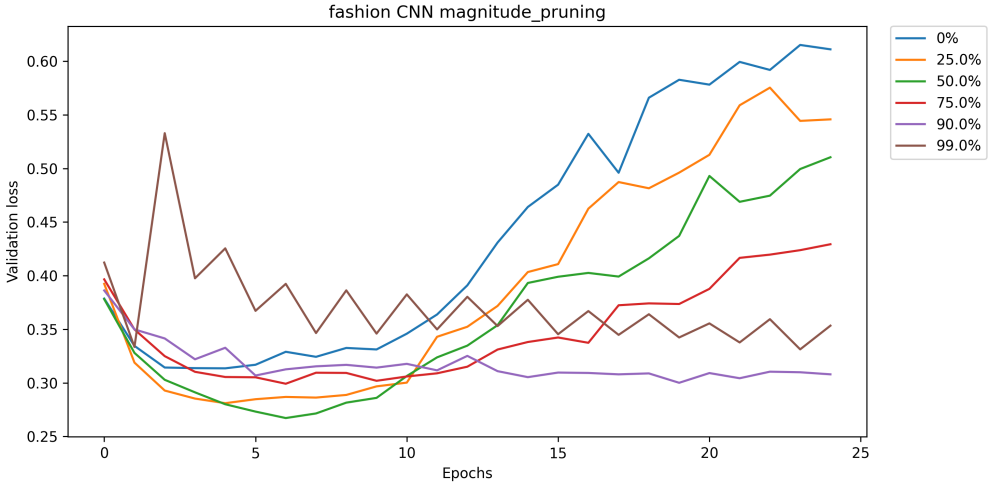

Figure 15: Validation loss of magnitude pruning for different sparsity levels.

Figure 15 shows the validation accuracy of magnitude pruning for different sparsity levels. Similar to the MNIST dataset, magnitude pruning helps in reducing the effects of overfitting. The validation loss continues to improve as 99% sparsity is achieved.

CIFAR-10

Figure 16 shows the learning curves for random pruning, magnitude pruning under a Bayesian framework compared to baseline in a fully connected network (FCN) trained on the CIFAR-10 dataset. Here the desired level of sparsity is set to 90%. The figure has two subplots. One shows the training and validation loss as a function of the number of epochs, the other plot (right) shows the Bayes factor, sparsity as a function of the number of epochs.

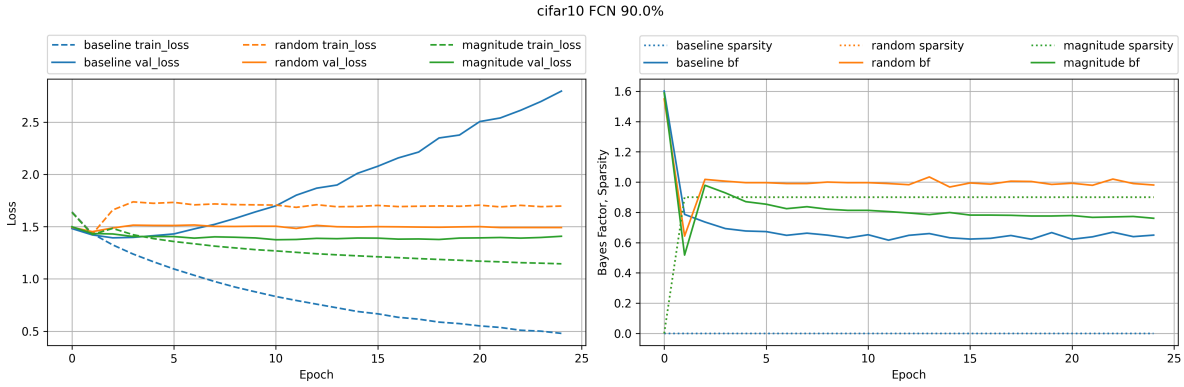

Figure 16: CIFAR-10 (FCN 90%) learning curves for the Bayesian pruning method.

Unlike the MNIST, Fashion datasets the input images of CIFAR-10 dataset are of size 32x32x3. This causes the number of parameters in the FCN to be much larger than that of the MNIST, Fashion datasets. This causes the effects of overfitting to be seen a little later in the training period. The trends in the learning curves are similar to that of the MNIST, Fashion datasets. The validation accuracy for random pruning decreases at the beginning of training and starts to improve as training progresses. The Bayes factor begins to reduce as the number of epochs increases and the sparsity of the network becomes stabilized for both magnitude pruning and random pruning.

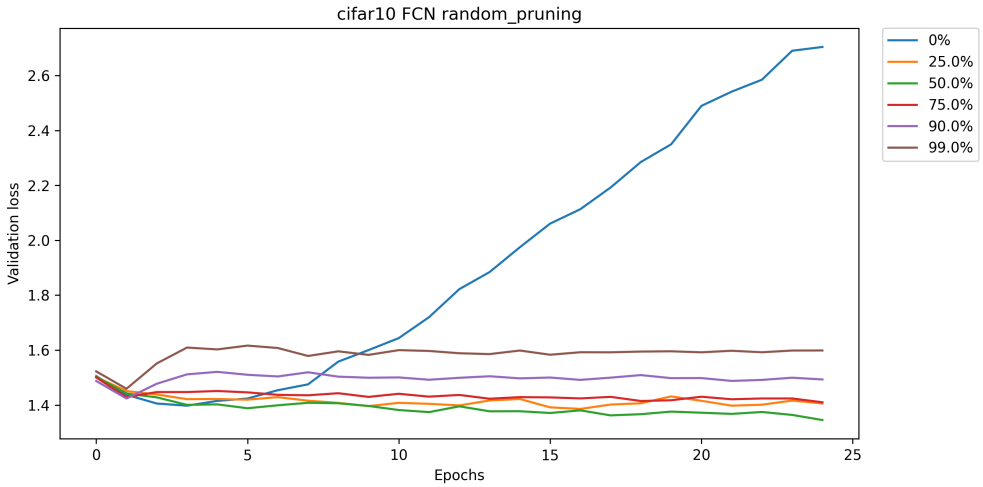

Figure 17: Validation loss of random pruning for different sparsity levels.

Figure 17 shows the validation accuracy of random pruning for different sparsity levels. Due to the larger network size, the effects of overfitting are higher. The trends for random pruning remains similar to that of the MNIST, Fashion datasets. The validation accuracy is better for 25% sparsity and decreases as the sparsity level increases.

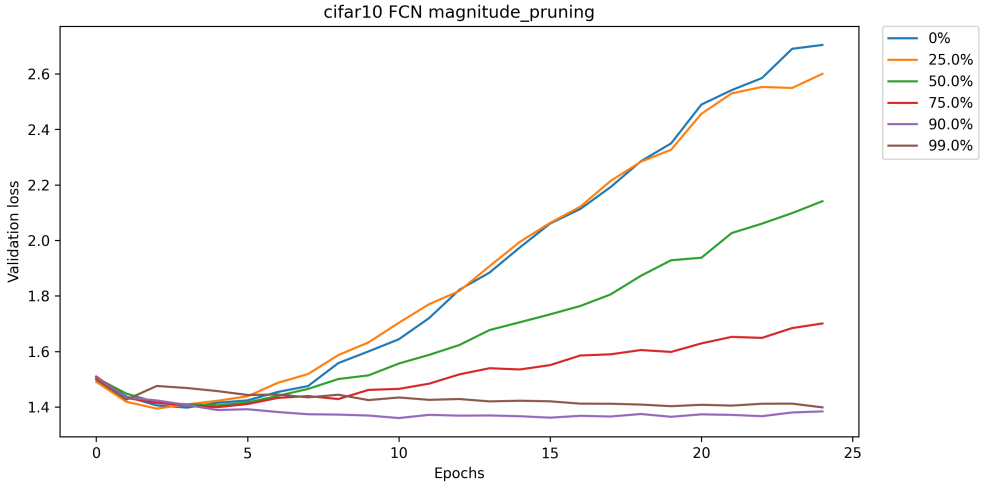

Figure 18: Validation loss of magnitude pruning for different sparsity levels.

Figure 18 shows the validation accuracy of magnitude pruning for different sparsity levels. The trends remain the same as that of the MNIST, Fashion datasets. Both Bayesian ranom and Bayesian magnitude pruning helps in reducing the effects of overfitting. The validation

loss continues to improve as 99% sparsity is achieved. Bayesian random pruning model fits the data better than magnitude pruning model.

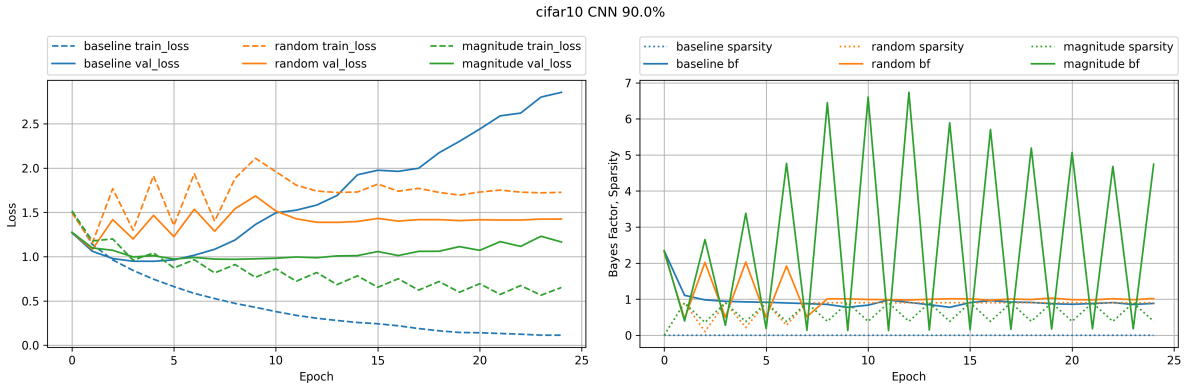

Figure 19: CIFAR-10 (CNN 90%) learning curves for the Bayesian pruning method.

Figure 19 shows the learning curves for random pruning, magnitude pruning under a Bayesian framework compared to baseline in a convolutional neural network (CNN) trained on the CIFAR-10 dataset. Here the desired level of sparsity is set to 90%. The figure has two subplots. One shows the training and validation loss as a function of the number of epochs, the other plot (right) shows the Bayes factor, sparsity as a function of the number of epochs. The learning trends are similar to that of the FCN. The validation accuracy for random pruning decreases at the beginning of training and starts to improve as training progresses. The Bayes factor begins to increase for magnitude pruning and sparsity fluctuates as training progresses. For random pruning the Bayes factor begins to reduce as the number of epochs increases and the sparsity of the network becomes stabilized.

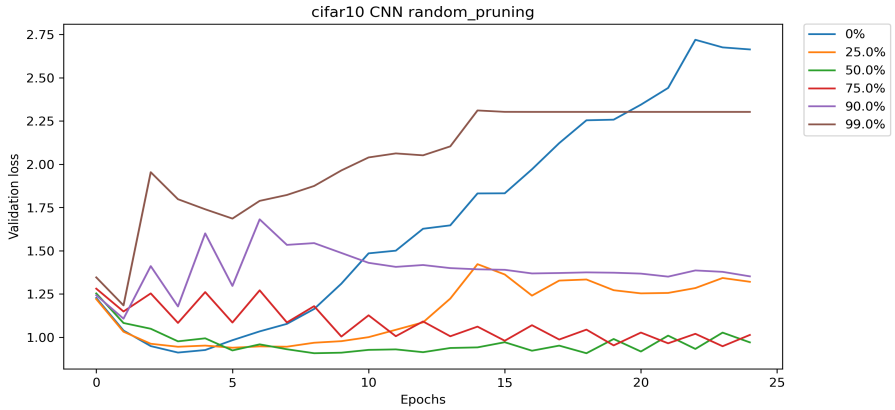

Figure 20: Validation loss of random pruning for different sparsity levels.

Figure 20 shows the validation accuracy of random pruning for different sparsity levels. The trends of random pruning is similar to that of the MNIST, Fashion datasets. The effects of overfitting are reduced by pruning. The validation accuracy decreases as the sparsity level increases to 99%.

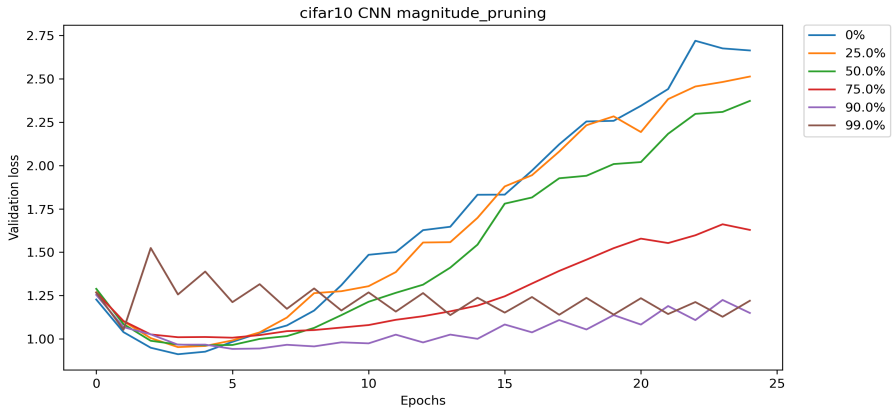

Figure 21: Validation loss of magnitude pruning for different sparsity levels.

Figure 21 shows the validation accuracy of magnitude pruning for different sparsity levels. The trends are similar to that of the MNIST, Fashion datasets. Magnitude pruning helps in reducing the effects of overfitting. The validation loss continues to improve as 99% sparsity is achieved.

Table 1: Accuracy values at different sparsity levels

Dataset	Model	Unpruned	Sparsity	Random	Bayes Random	Magnitude	Bayes Magnitude
MNIST	FCN	0.9782	25.0%	0.9684	**0.9747**	**0.9801**	0.9759
			50.0%	0.9684	**0.9710**	0.9791	0.9791
			75.0%	0.9578	**0.9706**	0.9779	**0.9812**
			90.0%	0.9624	**0.9657**	0.9768	**0.9772**
			99.0%	0.9433	**0.9439**	0.9743	**0.9767**
	CNN	0.9918	25.0%	**0.9908**	0.9835	0.9910	**0.992**
			50.0%	0.9858	**0.9906**	0.9900	**0.9901**
			75.0%	0.9872	**0.9905**	**0.9905**	0.9892
			90.0%	**0.9806**	0.9791	0.9880	**0.9888**
			99.0%	0.1135	0.1135	0.9826	0.9804
Fashion	FCN	0.8733	25.0%	0.8699	**0.8739**	0.8744	**0.8778**
			50.0%	**0.8659**	0.8566	0.8725	**0.8753**
			75.0%	0.8535	**0.8558**	**0.8800**	0.8799
			90.0%	0.8416	**0.8443**	0.8750	**0.8675**
			99.0%	0.8076	**0.8212**	0.8573	0.8573
	CNN	0.9028	25.0%	0.8905	**0.9030**	0.8959	**0.9002**
			50.0%	0.8957	**0.9021**	0.8906	**0.8982**
			75.0%	0.8838	**0.8773**	0.8894	**0.8974**
			90.0%	0.8520	**0.8589**	0.8986	**0.9022**
			99.0%	**0.7851**	0.7083	0.8595	**0.8768**
CIFAR-10	FCN	0.4869	25.0%	**0.5233**	0.5227	0.4857	**0.4908**
			50.0%	**0.5136**	0.5111	0.4981	**0.5010**
			75.0%	0.4950	**0.4972**	**0.5109**	0.5086
			90.0%	**0.4643**	0.4589	**0.5314**	0.5198
			99.0%	0.4158	**0.4381**	**0.4973**	0.4932
	CNN	0.6606	25.0%	0.6558	**0.6574**	0.6522	**0.6557**
			50.0%	0.6732	**0.6764**	0.6391	**0.6570**
			75.0%	0.6205	**0.6526**	0.6409	**0.6528**
			90.0%	**0.5169**	0.5092	**0.6467**	0.6437
			99.0%	0.1000	0.1000	0.5172	**0.5537**

The accuracy values at different sparsity levels for pruned networks are presented in Table 1. The networks were trained for 25 epochs, and the experiment was repeated 5 times with different random seeds for averaging the results. The table demonstrates that the Bayesian pruning method achieves higher sparsity levels without sacrificing accuracy. It outperforms unpruned networks and shows comparable or better accuracy compared to traditional neural network pruning techniques.

1.3 Discussion

Neural networks with a large number of parameters can learn complex functions but are prone to overfitting and are unsuitable for compute-constrained devices. Neural network pruning addresses both these challenges by reducing the network size. The iterative pruning method that we have introduced allows for pruning to a desired level of sparsity without losing any accuracy compared to the baseline. It allows for the network learn a function with fewer connections in principled manner as it checks to see if the network configuration is a good fit for the data. The extensive experiments conducted on three different datasets, two different network types, show that it's an effective method to train neural networks without additional parameterization.

References

Blalock, D. W., J. J. G. Ortiz, J. Frankle, and J. V. Guttag (2020). What is the state of neural network pruning? *CoRR abs/2003.03033*.

Blundell, C., J. Cornebise, K. Kavukcuoglu, and D. Wierstra (2015). Weight uncertainty in neural network. In *International conference on machine learning*, pp. 1613–1622. PMLR.

Brown, T., B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems 33*, 1877–1901.

Dusenberry, M. W., D. Tran, D. Hafner, L.-P. Brunel, D. Ho, and D. Erhan (2019). Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pp. 3294–3305.

Han, S., H. Mao, and W. J. Dally (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Han, S., J. Pool, J. Tran, and W. Dally (2015). Learning both weights and connections for efficient neural network. *Advances in neural information processing systems 28*.

He, Y., X. Zhang, and J. Sun (2018). Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1389–1398.

Kingma, D. P. and J. Ba (2015). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger (Eds.), *Advances in Neural Information Processing Systems*, Volume 25. Curran Associates, Inc.

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*(11), 2278–2324.

LeCun, Y., J. Denker, and S. Solla (1989). Optimal brain damage. *Advances in neural information processing systems 2*.

Li, H., A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf (2017). Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.

Liu, Z., M. Sun, T. Zhou, G. Huang, and T. Darrell (2018). Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*.

Molchanov, D., A. Ashukha, and D. Vetrov (2019). Variational dropout sparsifies deep neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 5234–5243.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research 15*(1), 1929–1958.

Strubell, E., A. Ganesh, and A. McCallum (2019). Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*.

Xiao, H., K. Rasul, and R. Vollgraf (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.