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 17, 2023

Abstract

The process of neural network pruning is a highly effective technique aimed at reducing the computational and memory demands of large neural networks. Its utility has been extensively demonstrated in mitigating the negative effects of overfitting and, in certain instances, surpassing the performance of unpruned 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. These factors serve as the foundation for implementing a well-defined pruning schedule. Through comprehensive evaluations conducted on multiple widely-accepted benchmarks, we demonstrate that our method achieves remarkable levels of sparsity while simultaneously maintaining competitive accuracy.

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

*The authors gratefully acknowledge *please remember to list all relevant funding sources in the unblinded version*

1 Introduction

In artificial neural networks (ANN) and machine learning (ML), parameters represent what the network has learned from the data. 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 state-of-the-art deep learning language model, GPT-3 (Brown et al., 2020), comprises 175 billion machine learning parameters. While deeper networks with higher parameter counts often yield better results, the large number of connections can introduce challenges such as memory limitations, overfitting, and lack of generalizability. To address these issues, various methods have been developed.

Regularization is a popular method for addressing overfitting by adding a penalty term to the loss function to prevent the model from overfitting. It effectively reduces the complexity of the model and improves its generalizability. However, regularization alone may not be effective in reducing the number of parameters in the model.

Neural network pruning is a widely used method for reducing the number of parameters in deep learning models, thereby decreasing computational complexity and memory requirements (Zhou et al., 2021). Pruning is crucial for deploying large models on resource-constrained devices such as personal computers, mobile phones and tablets. This technique involves removing weights or neurons from the network that have a negligible impact on its performance while retaining the most important ones. Traditional pruning methods often rely on heuristics or sensitivity analysis to determine which weights to remove, which can be suboptimal and lack a principled approach (Blalock et al., 2020).

Several pruning methods have been proposed, including weight pruning, neuron pruning, and filter pruning (Han et al., 2015; Li et al., 2017; He et al., 2018). Weight pruning involves removing individual weights from the network based on their magnitude or other criteria, while neuron pruning and filter pruning involve removing entire neurons or filters deemed

unimportant. One prominent neuron pruning technique is "dropout" (Srivastava et al., 2014), which randomly drops connections during training based on a heuristic. While these 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 pruning, the weights of the network are treated as random variables with a prior distribution, which can be updated to a posterior distribution using Bayes' rule. This approach allows us to quantify the uncertainty associated with each weight and select the most important ones based on their relevance to the task at hand. Bayesian pruning offers several advantages over traditional pruning methods, including a principled framework for weight selection, the ability to incorporate prior knowledge about the network structure, and the potential for improved performance when combined with other techniques like importance weighting. The posterior distribution reflects our updated belief about the weights based on observed data and can be used to calculate the probability that each weight is important for the task at hand. Variational inference, which involves minimizing the Kullback-Leibler (KL) divergence between the true posterior and an approximate distribution, 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).

In this work, we propose a Bayesian pruning algorithm that employs approximate Bayesian inference to calculate the posterior distribution of the weights and determine which ones to prune. Bayesian pruning provides a principled approach for selecting the most important weights in a neural network and can significantly reduce computational and memory requirements without sacrificing accuracy. We follow a similar approach to (Zhu and Gupta, 2017), iteratively pruning with varying levels of sparsity and monitoring accuracy for different levels of sparsity. Our approach uses Bayesian hypothesis testing to develop iterative pruning schedule to eliminate connections during training. We evaluate our method on standard benchmarks and demonstrate that it achieves high levels of sparsity

while maintaining competitive accuracy. Our approach also allows us to incorporate prior knowledge about the network structure and can be extended to handle more complex pruning scenarios. Overall, our results highlight the potential of Bayesian pruning as a promising approach for reducing the complexity of deep neural networks.

2 Methods

2.0.1 Pruning Neural Networks using Bayesian Inference

The pruning system, depicted 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 compared with the ground truth labels to compute the loss. The gradients of the weights are then computed through backpropagation, and an optimizer such as SGD or Adam (Kingma and Ba, 2015) is used to update the weights. After each epoch, the weights are pruned using the pruning algorithm, and the pruned weights are used in the subsequent epochs. The pruning algorithm leverages Bayesian hypothesis testing.

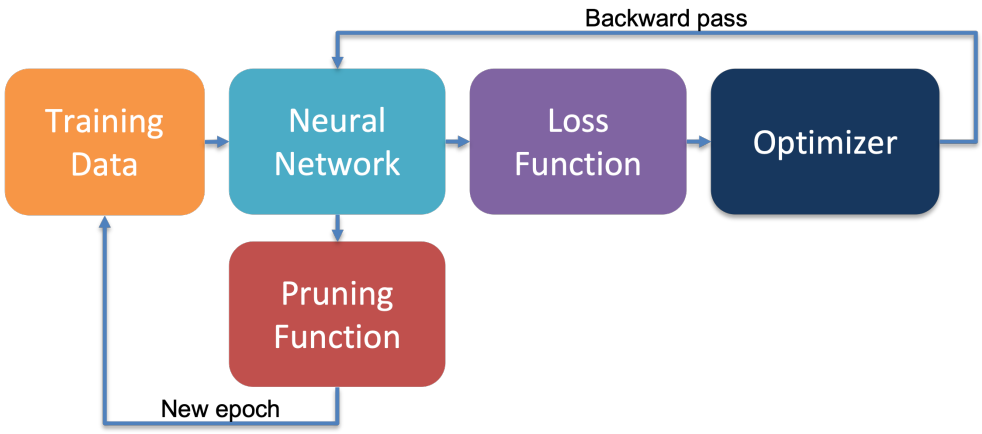

Figure 1: Pruning system block diagram.

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.

To enable layer-wise pruning, a common sparsity-inducing prior is introduced. This prior, denoted as $p(W^{(l)}|\theta^{(l)})$, is applied to the weight matrix $W^{(l)}$ of layer l. The prior distribution $p(w_i^{(l)}|\theta^{(l)})$ depends on hyperparameters $\theta^{(l)}$ that define the sparsity-inducing prior. A Gaussian prior with mean μ and variance σ^2 is used for each weight parameter $w_i^{(l)}$:

$$p(w_i^{(l)}|\theta^{(l)}) = \mathcal{N}(\mu, \sigma^2)$$

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. 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,

$$\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} \odot \mathbf{m} \tag{1}$$

where $\odot$ represents element-wise multiplication, **w**old is the old weight matrix, and $\mathbf{m}$ 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 $\mathbf{w}_{\text{new}}$ has the same dimensions as $\mathbf{w}_{\text{old}}$, but with some of its weights pruned. Algorithm 1 outlines the Bayesian pruning process.

Algorithm 1 Bayesian Pruning Algorithm

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

Compute the posterior probability of the weights before pruning

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

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

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.

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

$f(\cdot, \theta)$: Neural network model with parameters θ
r: Desired sparsity level, β Bayes factor threshold
Calculate log posterior probability $p(\theta|\mathcal{D})$
if $BF_{01} > \beta$ **then**
    for all weights $w_i \in \theta$ **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 $p(\theta|\mathcal{D})$ after pruning
Calculate Bayes factor BF_{01}

Magnitude-based pruning

Magnitude-based 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

$f(\cdot, \theta)$: Neural network model with parameters θ
r: Desired sparsity level, β Bayes factor threshold
Calculate log posterior probability $p(\theta|\mathcal{D})$
if $BF_{01} > \beta$ **then**
    for all weights $w_i \in \theta$ **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 $p(\theta|\mathcal{D})$ 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 five different levels of sparsity are 25%, 50%, 75%, 90% and 99%. We use the Adam optimizer with a learning rate of 0.001 and a batch size of 64 for all experiments. We use the PyTorch DataLoader class to load and preprocess the 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.

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

Datasets

MNIST dataset The MNIST dataset (Lecun et al., 1998) consists of 60,000 training images and 10,000 test images of handwritten digits. Each image is 28×28 pixels and is grayscale. The images are normalized to have zero mean and unit variance. The images are flattened into a 784-dimensional vector and fed into the neural network. The network is trained to classify the images into one of the 10 classes.

MNIST-Fashion dataset The MNIST-Fashion dataset (Xiao et al., 2017) consists of 60,000 training images and 10,000 test images of fashion items. Each image is 28×28 pixels and is grayscale. The images are normalized to have zero mean and unit variance. The images are flattened into a 784-dimensional vector to be fed into the fully connected neural network. The network is trained to classify the images into one of the 10 classes.

CIFAR-10 dataset The CIFAR-10 dataset (Krizhevsky, 2009) consists of 50,000 training images and 10,000 test images of 10 classes of objects. Each image is 32×32 pixels and is RGB. The images are normalized to have zero mean and unit variance. The images are flattened into a 3072-dimensional vector and fed into the neural network. The network is trained to classify the images into one of the 10 classes.

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.

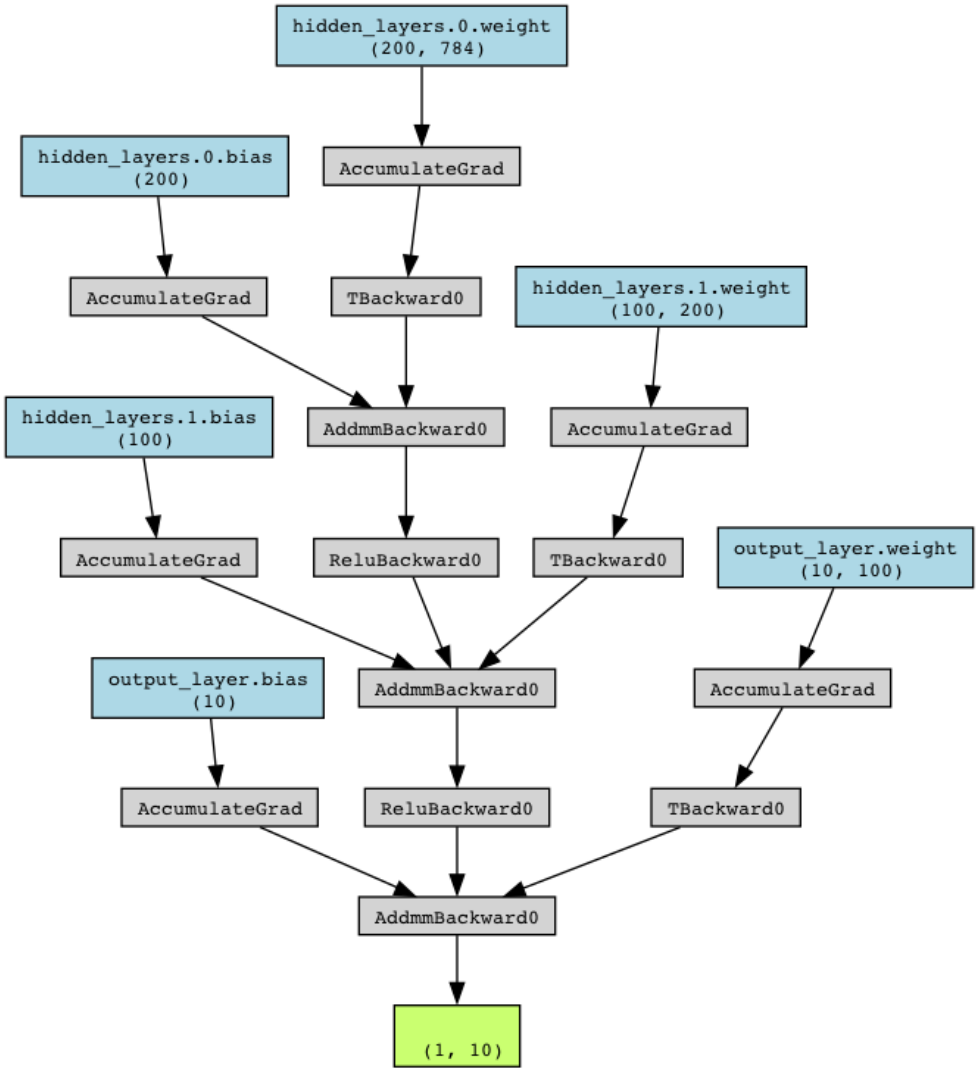

Figure 2: Fully connected neural network architecture

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

$$\begin{aligned} \mathbf{h}_1 &= \text{ReLU}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \\ \mathbf{h}_2 &= \text{ReLU}(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2) \\ \mathbf{y} &= \mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3 \end{aligned} \tag{2}$$

where $\mathbf{x}$ is the input, $\mathbf{h}_1$ and $\mathbf{h}_2$ are the two hidden layers, $\mathbf{y}$ is the output, $\mathbf{W}_1$, $\mathbf{W}_2$

and $\mathbf{W}_3$ are the weight matrices, $\mathbf{b}_1$, $\mathbf{b}_2$ and $\mathbf{b}_3$ are the bias vectors, and ReLU$(\cdot)$ is the rectified linear unit activation function.

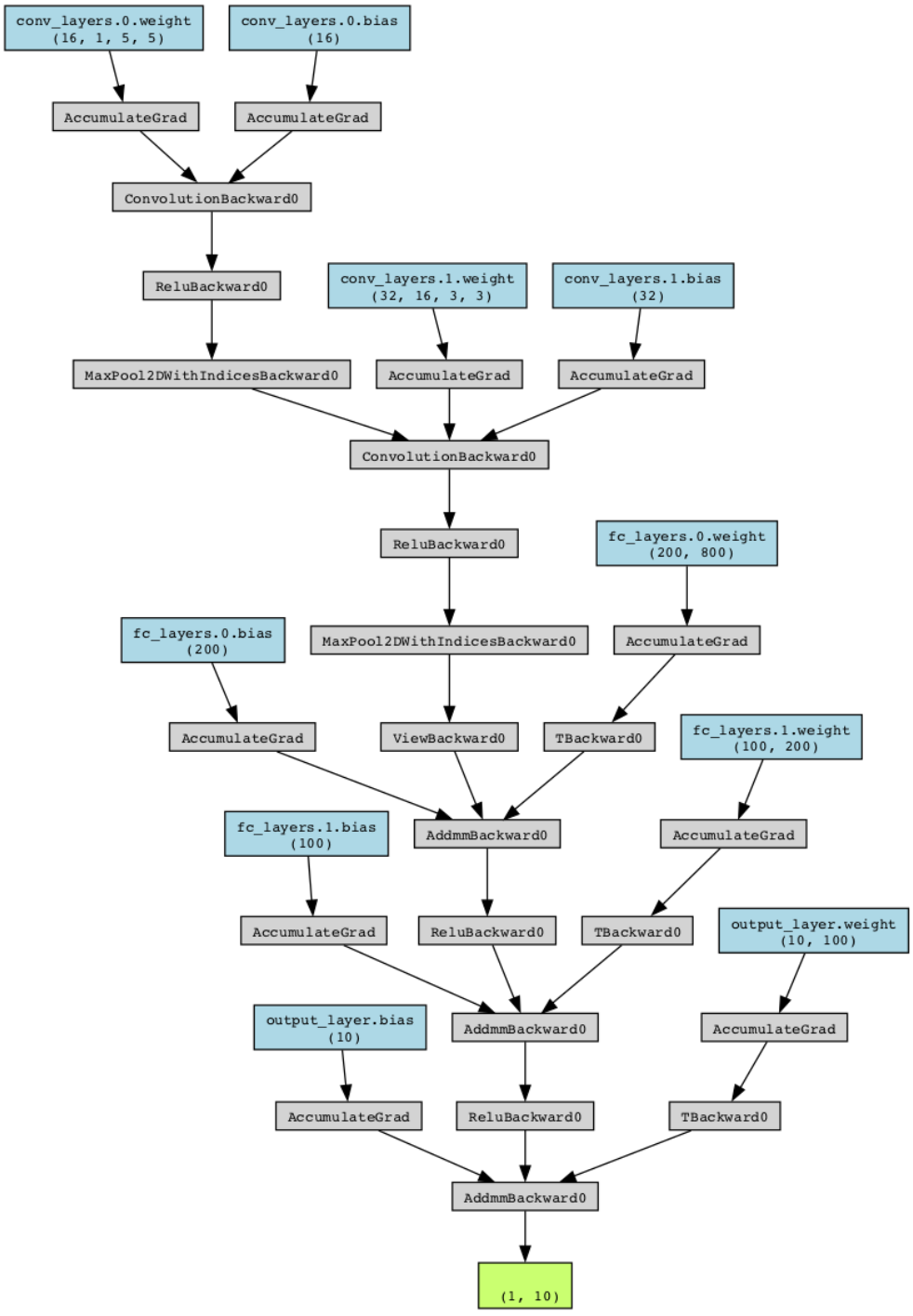

Figure 3: Convolutional neural network architecture

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

3. Equation 3 describes the forward pass of the network.

$$
\begin{aligned}
\mathbf{h}_1 &= \text{ReLU}(\text{Conv2d}(\mathbf{x}, \mathbf{W}_1) + \mathbf{b}_1) \\
\mathbf{h}_2 &= \text{MaxPool2d}(\mathbf{h}_1) \\
\mathbf{h}_3 &= \text{ReLU}(\text{Conv2d}(\mathbf{h}_2, \mathbf{W}_2) + \mathbf{b}_2) \\
\mathbf{h}_4 &= \text{MaxPool2d}(\mathbf{h}_3) \\
\mathbf{h}_5 &= \text{ReLU}(\mathbf{W}_3\mathbf{h}_4 + \mathbf{b}_3) \\
\mathbf{h}_6 &= \text{ReLU}(\mathbf{W}_4\mathbf{h}_5 + \mathbf{b}_4) \\
\mathbf{y} &= \mathbf{W}_5\mathbf{h}_6 + \mathbf{b}_5
\end{aligned}
\tag{3}
$$

where $\mathbf{x}$ is the input, $\mathbf{h}_1$, $\mathbf{h}_2$, $\mathbf{h}_3$, $\mathbf{h}_4 \cdots \mathbf{h}_6$ are the hidden layers, $\mathbf{y}$ is the output, $\mathbf{W}_1$, $\mathbf{W}_2$, $\cdots$ $\mathbf{W}_5$ are the weight matrices, $\mathbf{b}_1$, $\mathbf{b}_2$, $\cdots$ $\mathbf{b}_5$ are the bias vectors, Conv2d$(\cdot)$ is the convolutional layer, MaxPool2d$(\cdot)$ is the max pooling layer, and ReLU$(\cdot)$ is the rectified linear unit activation function.

2.1 Results

The following sections present the results of the experiments. The results are presented in the following order: (1) MNIST dataset, (2) CIFAR-10 dataset, and (3) CIFAR-100 dataset. The results are presented in the form of learning curves, accuracy, and sparsity. The learning curves show the training and validation loss as a function of the number of epochs. 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 sparsity levels are 25%, 50%, 75%, 90%, and 99%. The results are presented for both random pruning and magnitude pruning under a Bayesian framework. The results are compared to baseline, which is the model trained without pruning. The results are presented for both FCN and CNN architectures.

MNIST dataset

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. More figures for different sparsity levels are shown in Appendix 2.2.

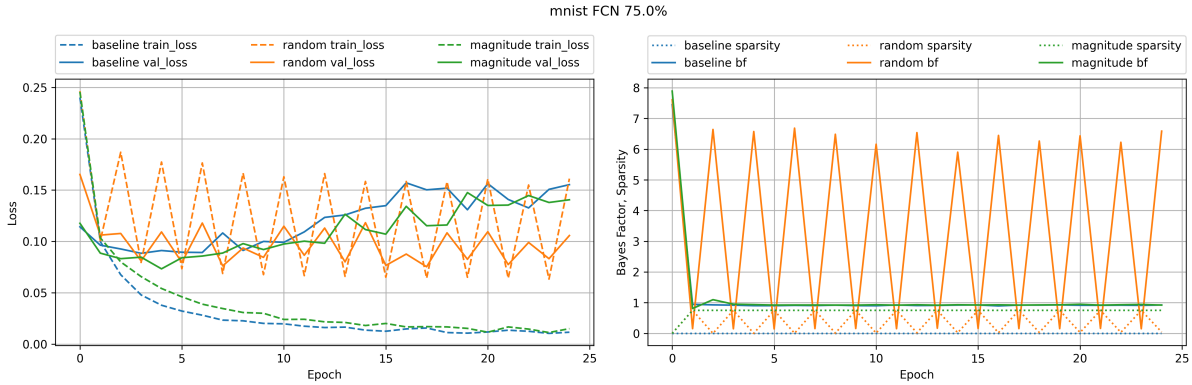

Figure 4: Learning curves for a FCN with MNIST dataset.

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 during the training epochs.

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

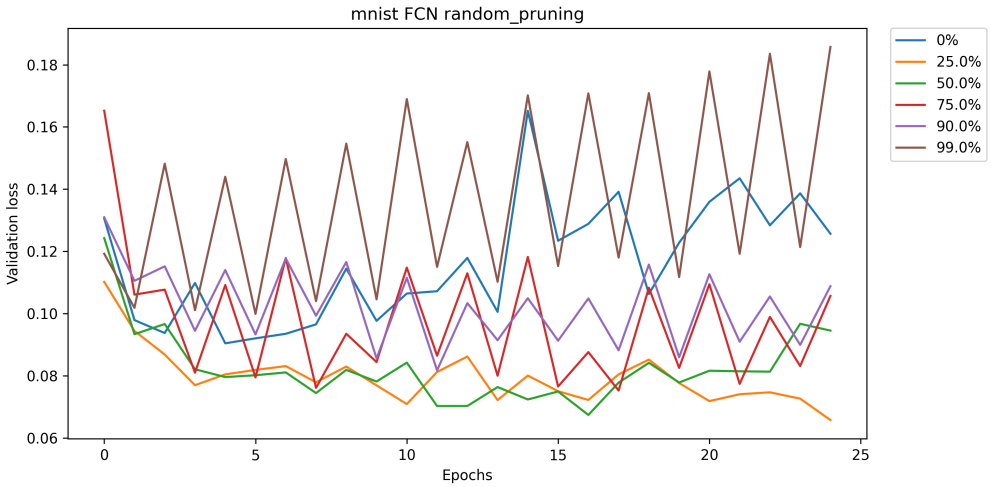

Figure 5: Validation loss of random pruning for different sparsity levels.

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.

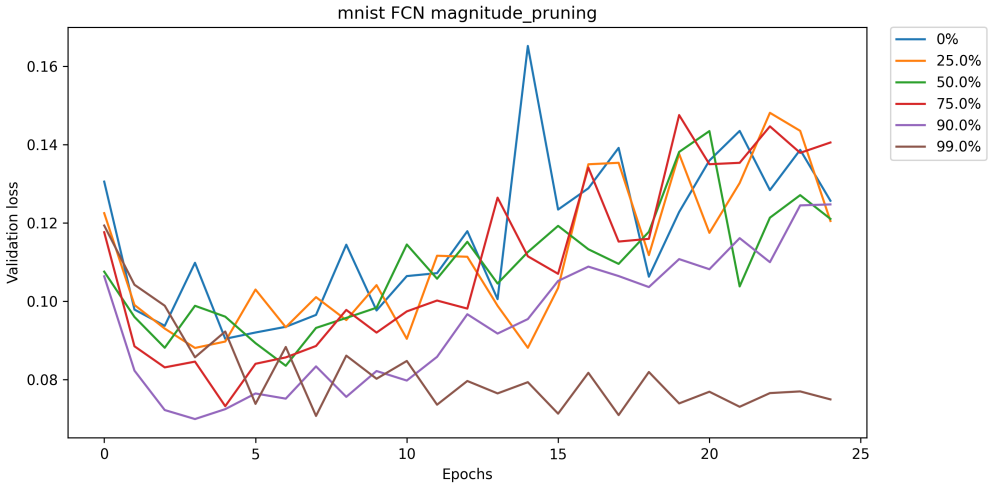

Figure 6: Validation loss of magnitude pruning for different sparsity levels.

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.

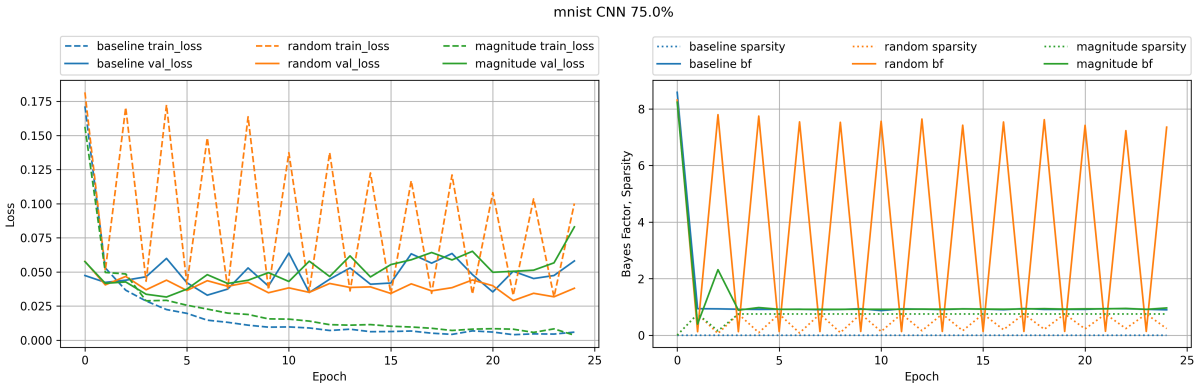

Figure 7: Learning curves for CNN with MNIST dataset.

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 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. More figures for different sparsity levels are shown in Appendix 2.2.

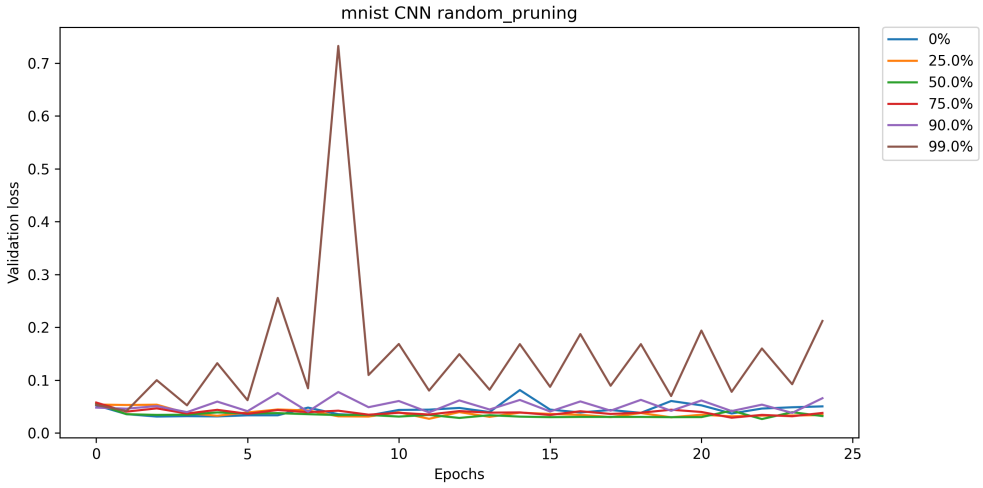

Figure 8: Validation loss of random pruning for different sparsity levels.

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.

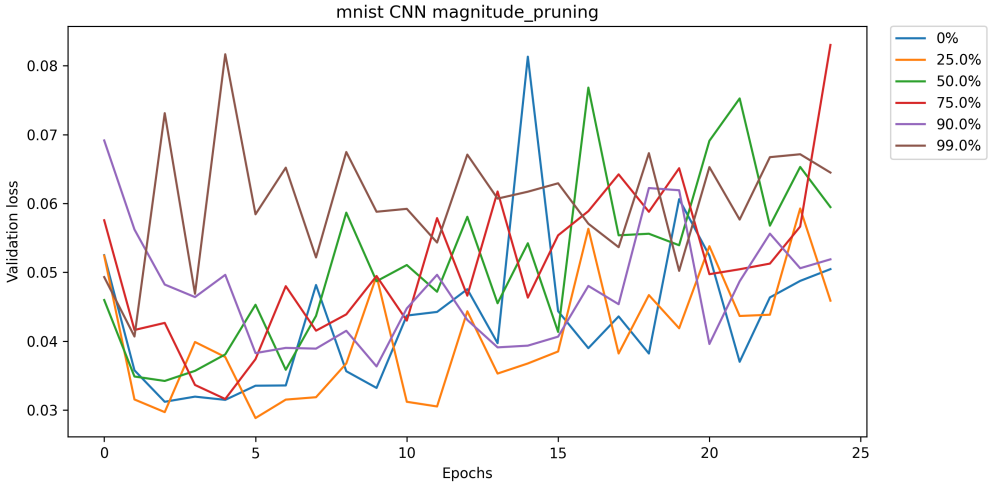

Figure 9: Validation loss of magnitude pruning for different sparsity levels.

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. More figures for different sparsity levels are shown in Appendix 2.2.

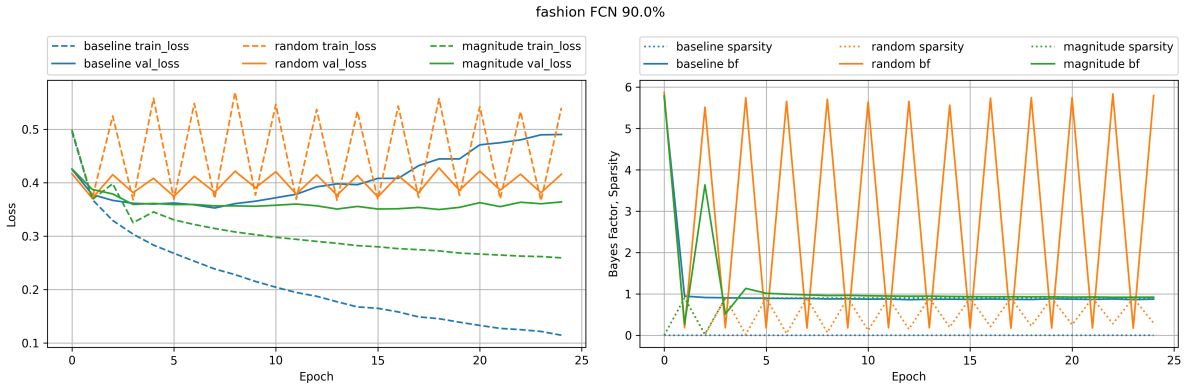

Figure 10: Learning curves for FCN with MNIST Fashion dataset.

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.

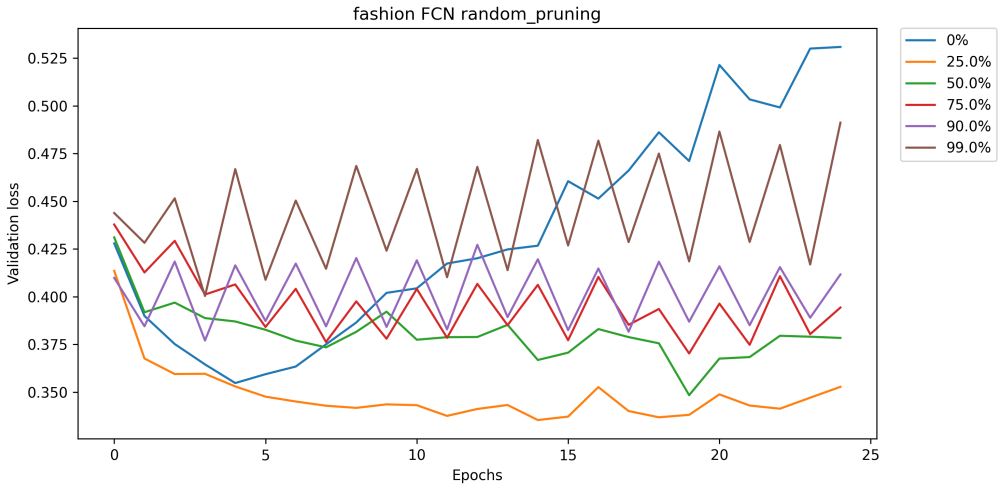

Figure 11: Validation loss of random pruning for different sparsity levels.

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.

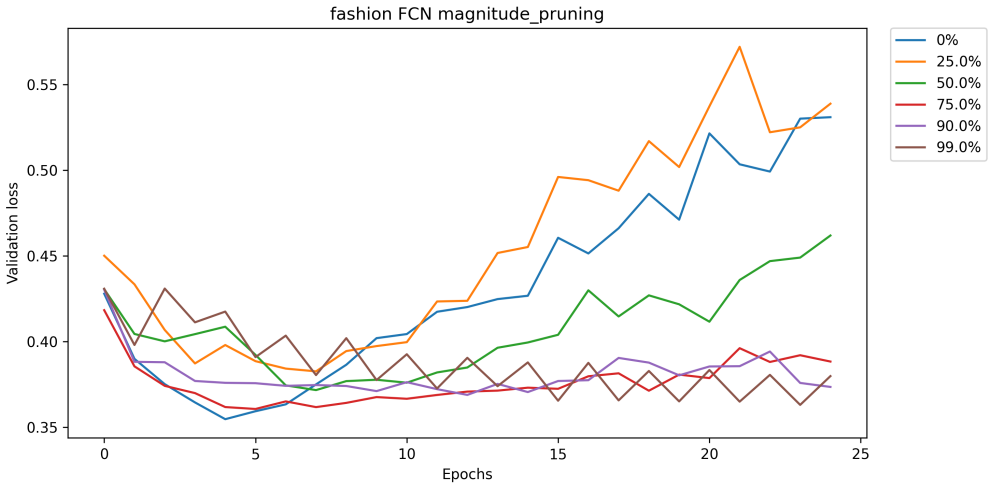

Figure 12: Validation loss of magnitude pruning for different sparsity levels.

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.

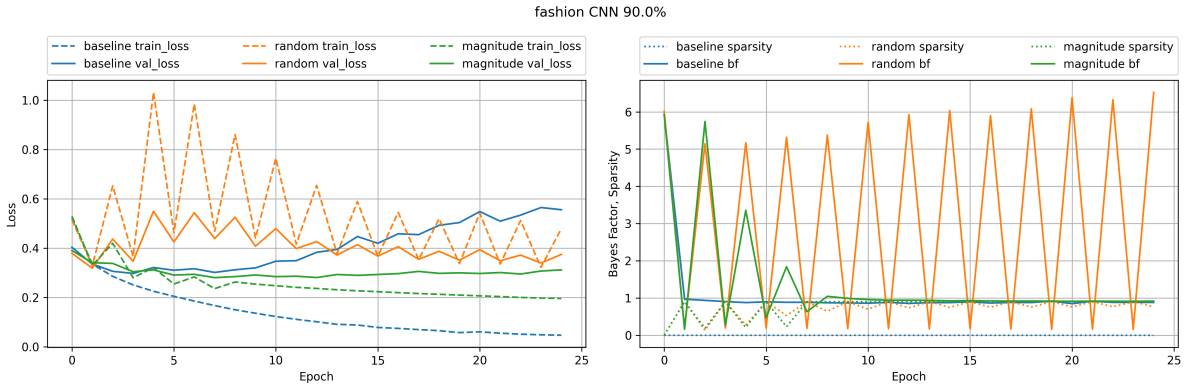

Figure 13: Learning curves for CNN with MNIST Fashion dataset.

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 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 shows the validation accuracy of magnitude pruning for different sparsity levels. Similar to the MNIST dataset, magnitude pruning helps in reducing the effects of

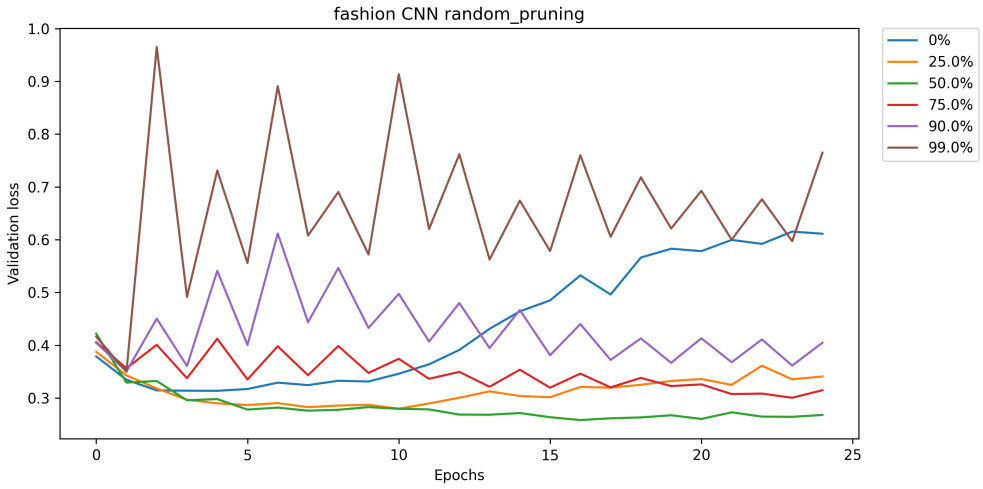

Figure 14: Validation loss of random pruning for different sparsity levels.

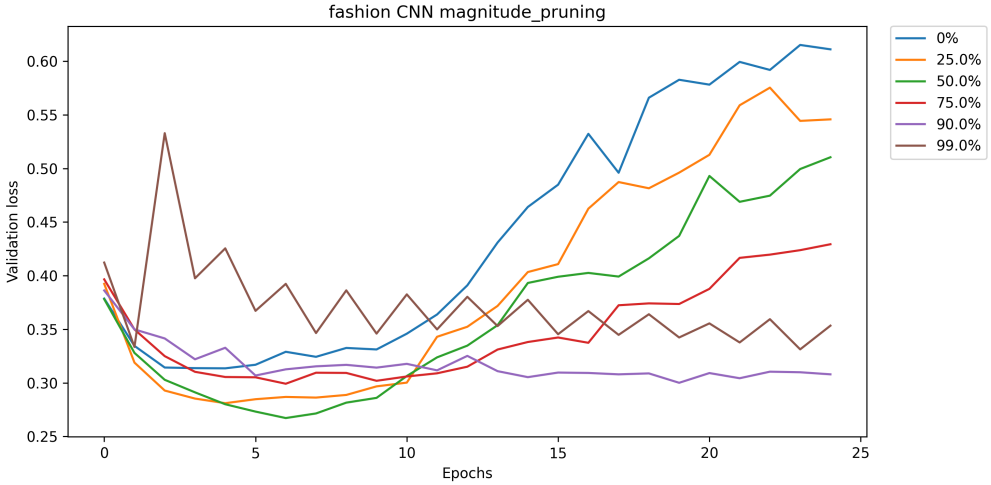

Figure 15: Validation loss of magnitude pruning for different sparsity levels.

overfitting. The validation loss continues to improve as 99% sparsity is achieved.

CIFAR-10 dataset

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. More figures for different sparsity levels are shown in Appendix 2.2.

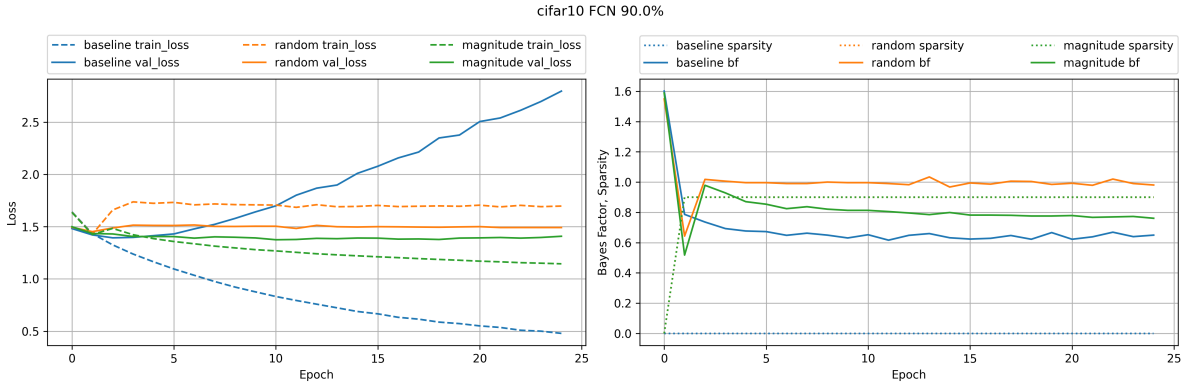

Figure 16: Learning curves for FCN with CIFAR-10 dataset.

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.

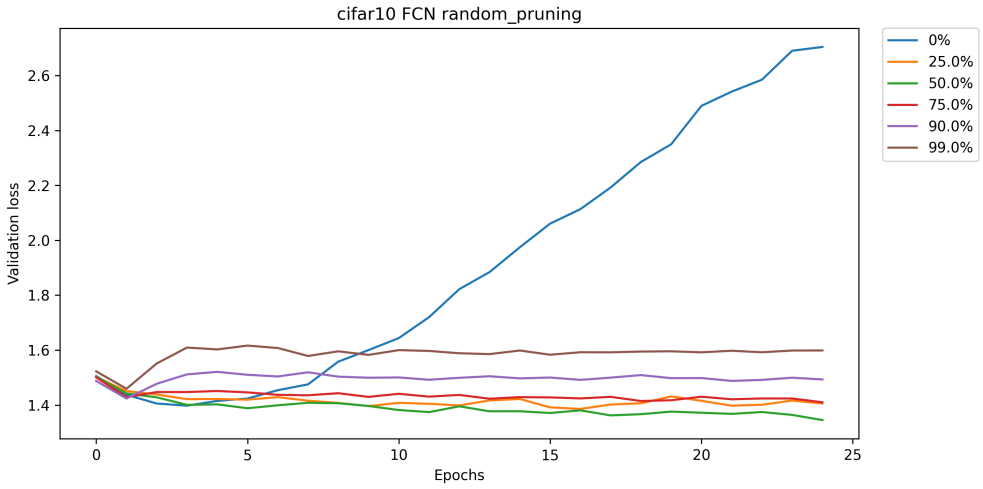

Figure 17: Validation loss of random pruning for different sparsity levels.

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.

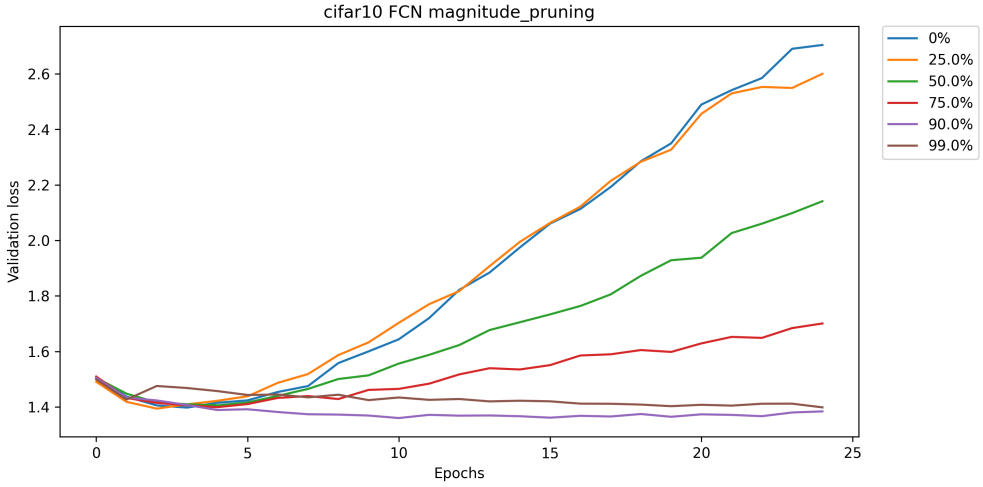

Figure 18: Validation loss of magnitude pruning for different sparsity levels.

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.

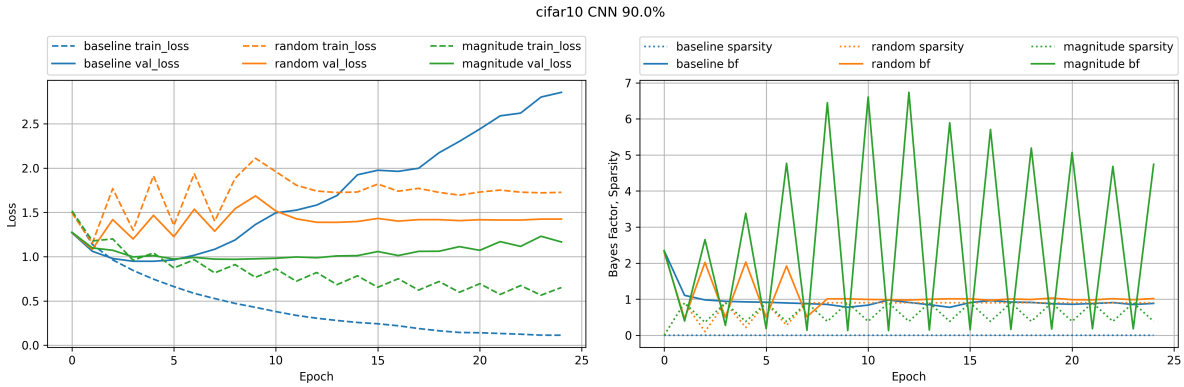

Figure 19: Learning curves for CNN with CIFAR-10 dataset.

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. More figures for different sparsity levels are shown in Appendix 2.2.

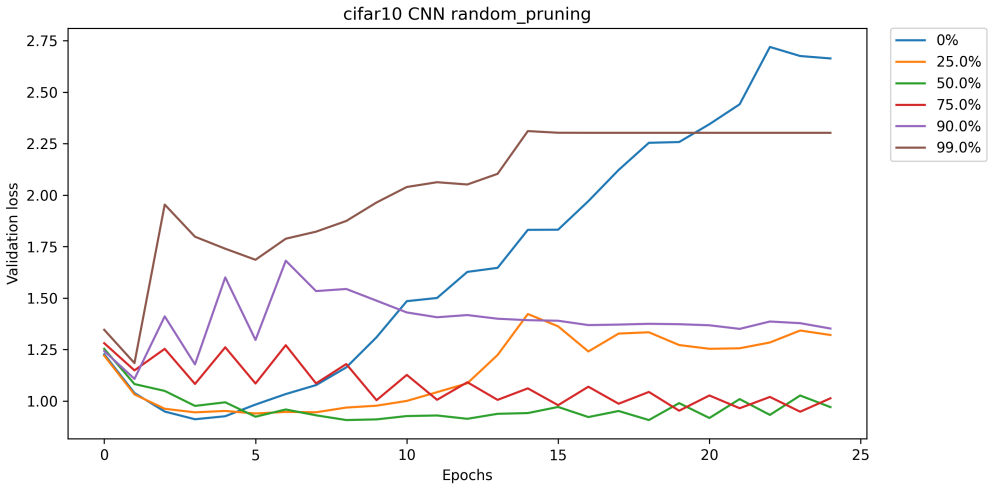

Figure 20: Validation loss of random pruning for different sparsity levels.

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%.

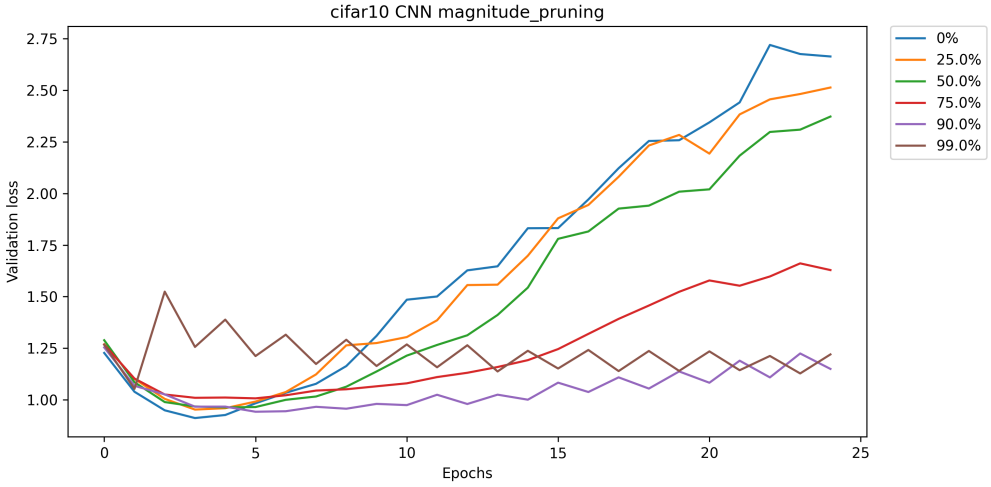

Figure 21: Validation loss of magnitude pruning for different sparsity levels.

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.

Summary of results,

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.

2.2 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 conducted experiments on various datasets and network architectures confirm that Bayesian pruning offers a principled approach to eliminate network connections. This method effectively trains neural networks with fewer parameters.

In summary, the Bayesian pruning method successfully prunes networks to higher sparsity levels while maintaining accuracy. It outperforms unpruned networks and achieves comparable or better accuracy than traditional pruning methods. The findings suggest that Bayesian pruning is a valuable technique for training efficient neural networks.

SUPPLEMENTARY MATERIAL

Bayesian Pruning Learning Curves: Figures showing learning curves for all three datasets at different sparsity levels for a FCN, CNN.

MNIST Learning Curves

The following figures show the learning curves for the Bayesian pruning method on the MNIST dataset for a FCN, CNN at different sparsity levels.

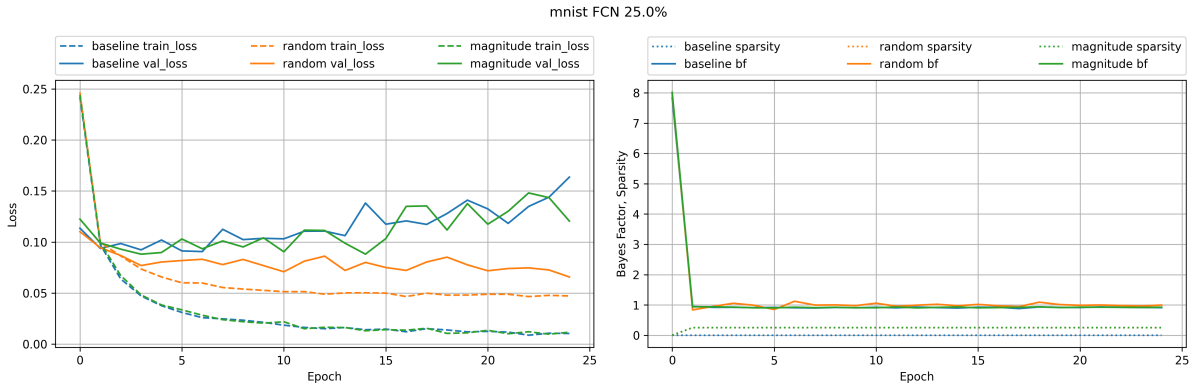

Figure 22: MNIST (FCN 25%) learning curve for the Bayesian pruning method.

Figure 22 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a FCN at 25% sparsity. Both the baseline and Bayesian magnitude methods validation loss deteriorate as training progresses. Only Bayesian random pruning is able to maintain a low validation loss. Pruning only 25 % of the weights with the lowest magnitude does not help combat overfitting as there is still sufficient weights to memorize the training data. The Bayesian random pruning method is able to maintain a low validation loss because it prunes weights randomly, and thus is able to prune weights that are not necessarily the least important weights. Bayes factor remains below one at the same level as the baseline method for the Bayesian magnitude method which indicates that the model is not a good fit for the data throughout the training epochs.

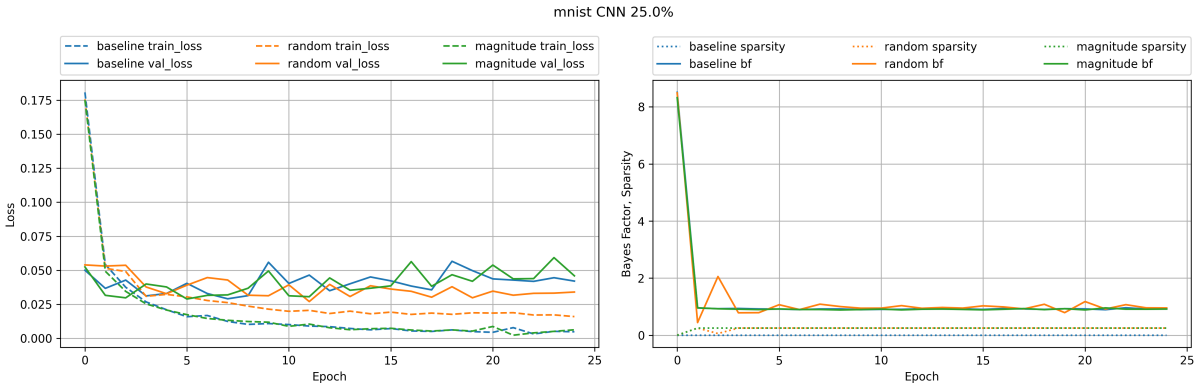

Figure 23: MNIST (CNN 25%) learning curves for the Bayesian pruning method.

Figure 23 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a CNN at 25% sparsity. Baseline, Bayesian random and Bayesian magnitude shows lower amount of overfitting compared to its FCN counterpart. There does not seem to be a significant difference between the baseline and Bayesian pruning methods. Bayes factor remains below one at the same level as the baseline method for Bayesian pruning methods, suggesting a similar level of fit to the training data.

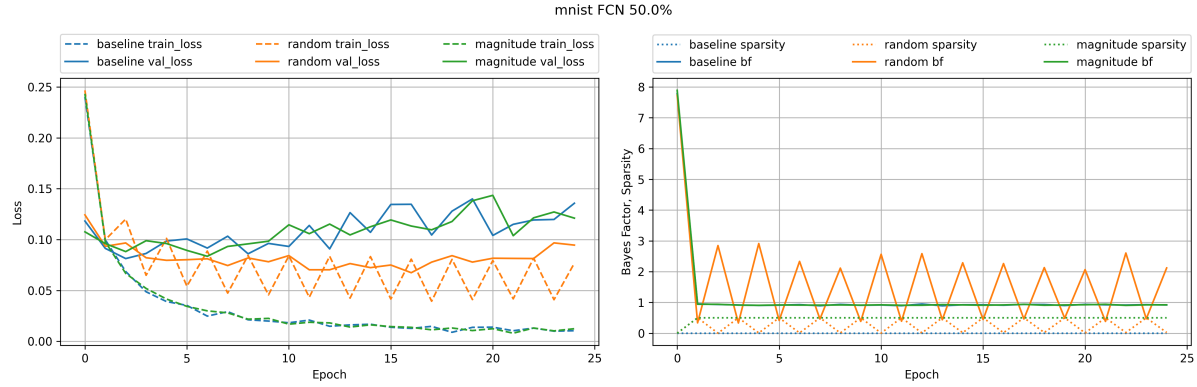

Figure 24: MNIST (FCN 50%) learning curves for the Bayesian pruning method.

Figure 24 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a FCN at 50% sparsity. Observing the validation losses, Bayesian random method does slightly better than baseline and Bayesian magnitude methods. Bayes factor remains below one at the same level as the baseline method for Bayesian

pruning methods, suggesting a similar level of fit to the training data.

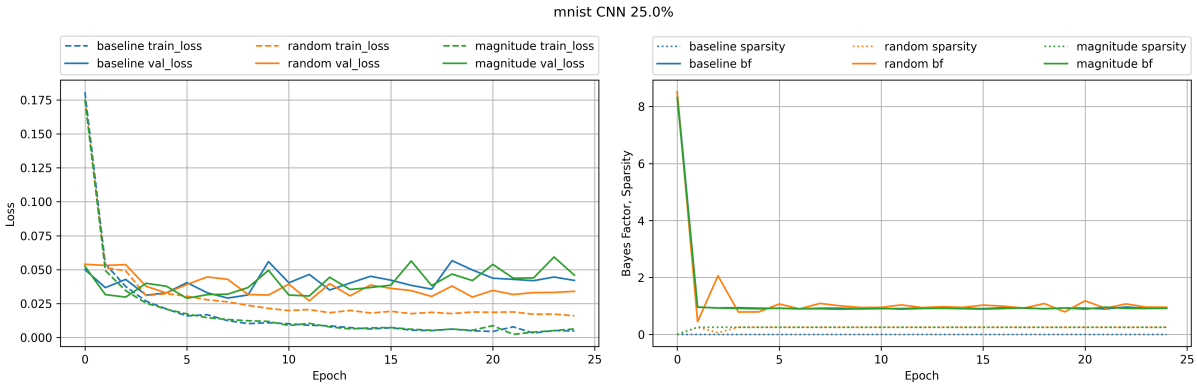

Figure 25: MNIST (CNN 50%) learning curves for the Bayesian pruning method.

Figure 25 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a CNN at 50% sparsity. Bayesian random method does slightly better than baseline and Bayesian magnitude methods. Bayes random method has higher Bayes factor than the baseline and Bayesian magnitude methods, suggesting a better fit to the training data.

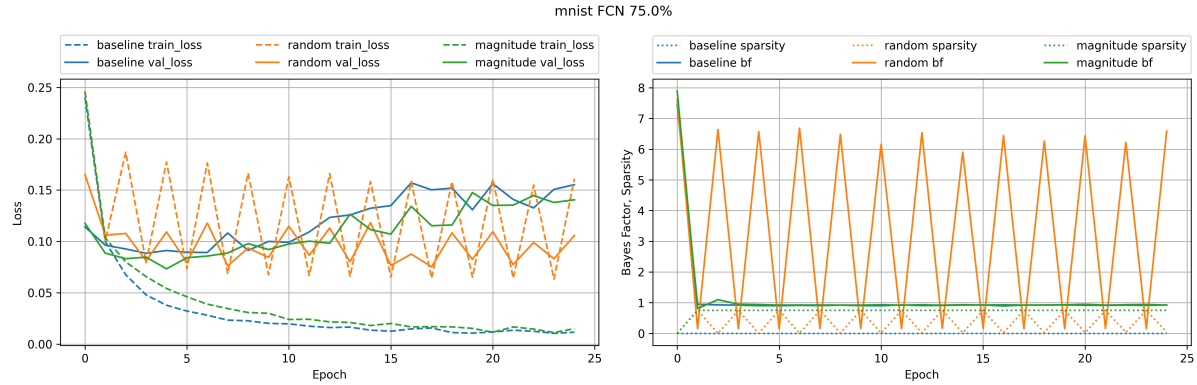

Figure 26: MNIST (FCN 75%) learning curves for the Bayesian pruning method.

Figure 26 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a FCN at 75% sparsity. Bayesian random method does slightly better than baseline and Bayesian magnitude methods. Bayes random method has higher Bayes factor than the baseline and Bayesian magnitude methods, suggesting a better

fit to the training data.

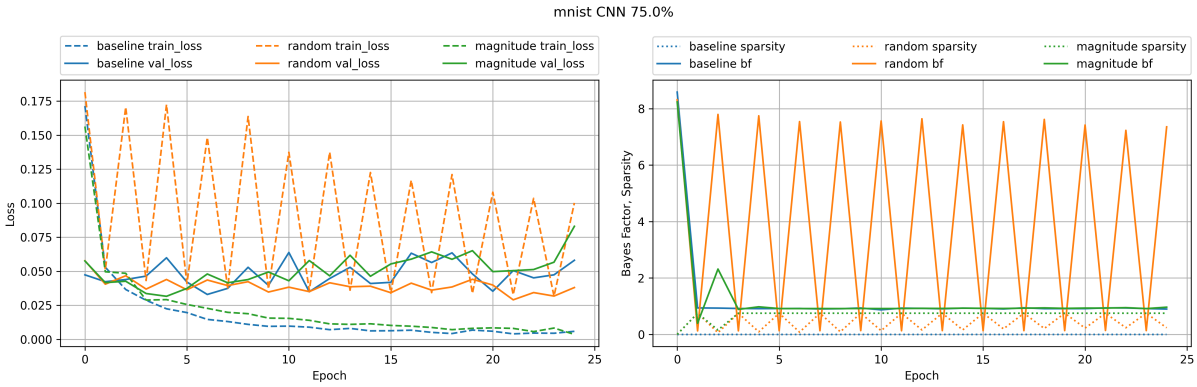

Figure 27: MNIST (CNN 75%) learning curves for the Bayesian pruning method.

Figure 27 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a CNN at 75% sparsity. Bayesian random method does slightly better than baseline and Bayesian magnitude methods. Bayes random method has higher Bayes factor than the baseline and Bayesian magnitude methods, suggesting a better fit to the training data.

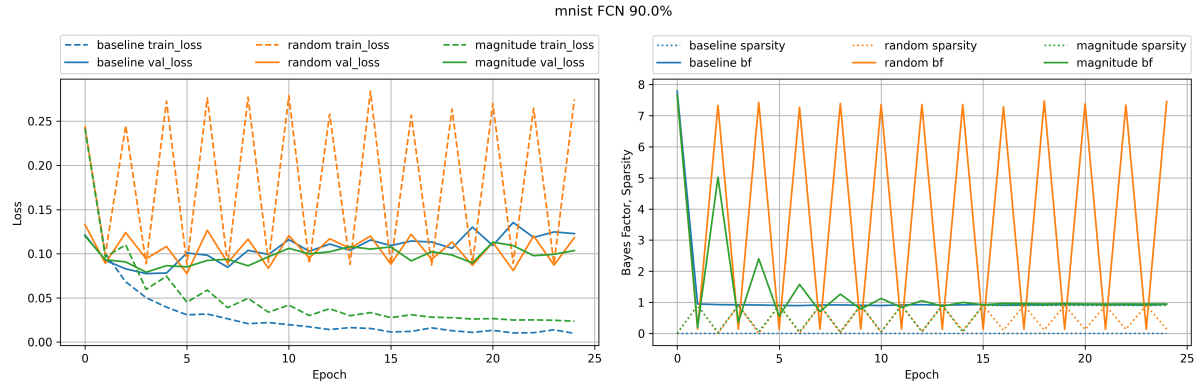

Figure 28: MNIST (FCN 90%) learning curves for the Bayesian pruning method.

Figure 28 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a FCN at 90% sparsity. Baseline, Bayesian random and Bayesian magnitude models show similar amount of overfitting. The validation loss starts to deteriorate as 90% of weights are dropped. Bayes factor remains below one at the same level as

the baseline method for Bayesian magnitude, Bayesian random method shows higher Bayes factor, suggesting a better fit to the training data.

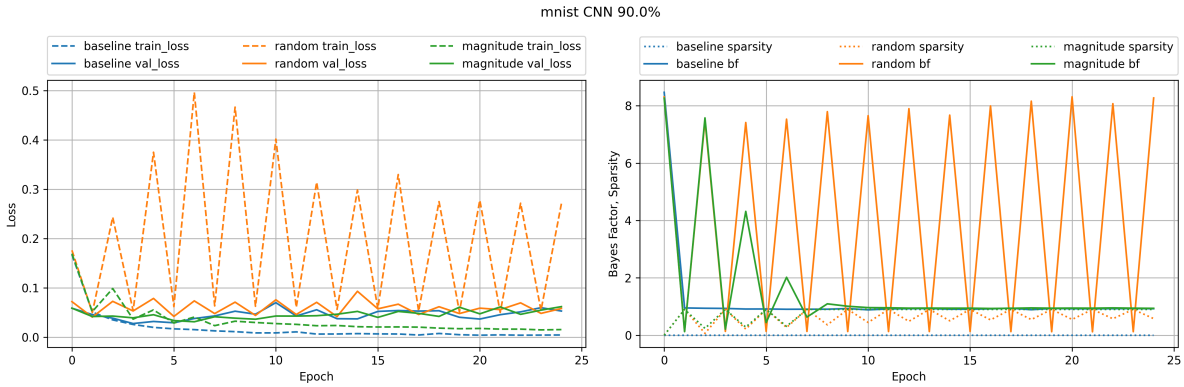

Figure 29: MNIST (CNN 90%) learning curves for the Bayesian pruning method.

Figure 29 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a CNN at 90% sparsity. Baseline, Bayesian random and Bayesian magnitude models show similar amount of overfitting but lower than the FCN models at 90% sparsity. The validation loss starts to deteriorate as 90% of weights are dropped. Bayes factor remains below one at the same level as the baseline method for Bayesian magnitude, Bayesian random method shows higher Bayes factor, suggesting a better fit to the training data.

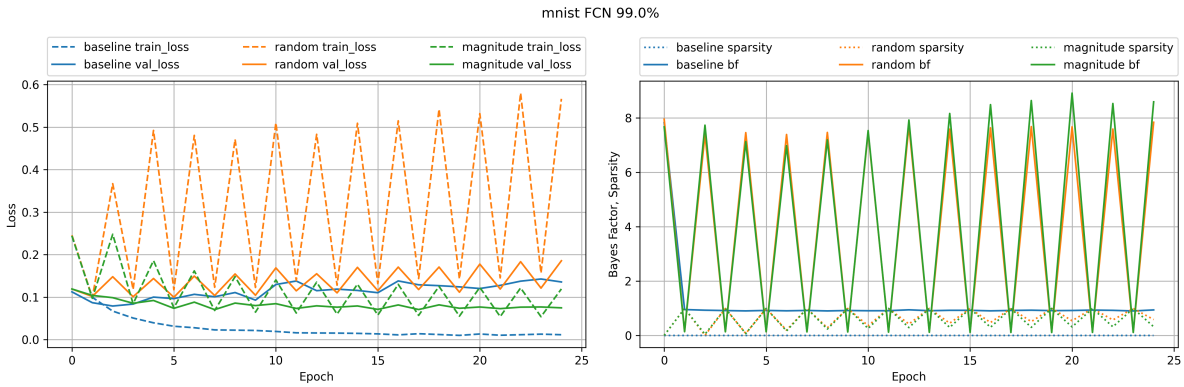

Figure 30: MNIST (FCN 99%) learning curves for the Bayesian pruning method.

Figure 30 shows the learning curves for the Bayesian pruning method on the MNIST

dataset for a FCN at 99% sparsity. Baseline and Bayesian random show similar amount of overfitting. Bayesian magnitude performs best, with the lowest validation loss. Bayes factor for Bayesian random and Bayesian magnitude have a similar trend, suggesting a similar level of fit to the training data.

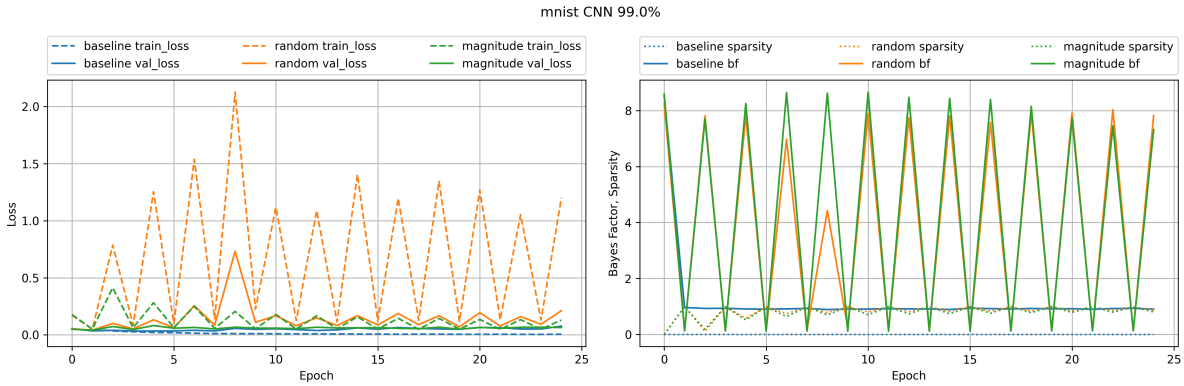

Figure 31: MNIST (CNN 99%) learning curves for the Bayesian pruning method.

Figure 31 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a CNN at 99% sparsity. Baseline, Bayesian random and Bayesian magnitude show similar amount of overfitting. Bayes factor for Bayesian random and Bayesian magnitude have a similar trend, suggesting a similar level of fit to the training data.

MNIST Fashion Learning Curves

The following figures show the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a FCN, CNN at different sparsity levels.

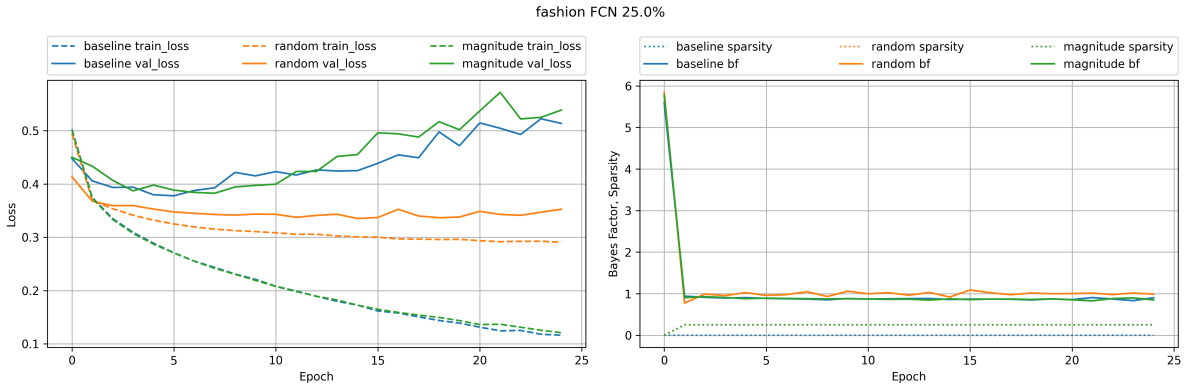

Figure 32: MNIST Fashion (FCN 25%) learning curves for the Bayesian pruning method.

Figure 32 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a FCN at 25% sparsity. Both the baseline and Bayesian magnitude methods show similar levels of overfitting. Only Bayesian random pruning is able to combat overfitting to some extent. Pruning only 25 % of the weights with the lowest magnitude does not help combat overfitting as there is still sufficient weights to memorize the training data. The Bayesian random pruning method is able to maintain a low validation loss because it prunes weights randomly, and thus is able to prune weights that are not necessarily the least important weights. Bayes factor remains below one at the same level as the baseline method for the Bayesian magnitude method. Bayesian random method shows higher Bayes factor, suggesting a better fit to the training data.

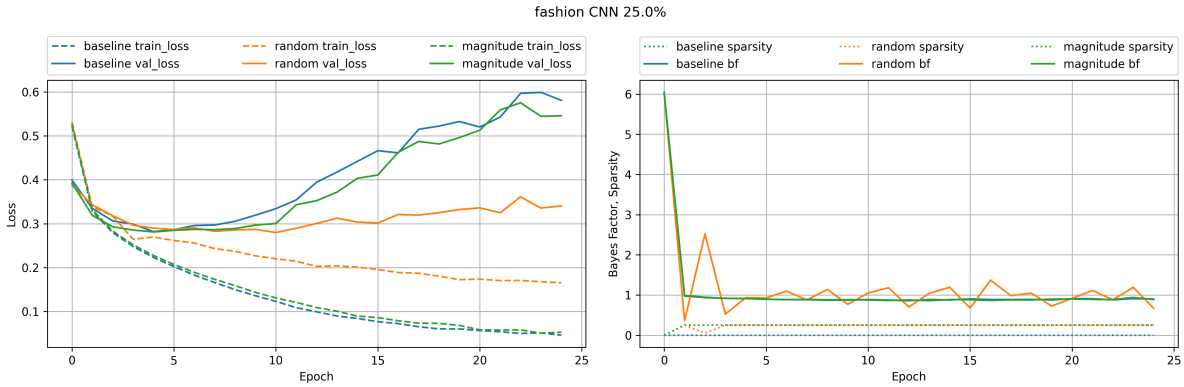

Figure 33: MNIST Fashion (CNN 25%) learning curves for the Bayesian pruning method.

Figure 33 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a CNN at 25% sparsity. Both the baseline and Bayesian magnitude methods show similar levels of overfitting. Only Bayesian random pruning is able to combat overfitting to some extent. Pruning only 25 % of the weights with the lowest magnitude does not help combat overfitting as there is still sufficient weights to memorize the training data. The Bayesian random pruning method is able to maintain a low validation loss because it prunes weights randomly, and thus is able to prune weights that are not necessarily the least important weights. Bayes factor remains below one at the same level as the baseline method for the Bayesian magnitude method. Bayesian random method shows higher Bayes factor, suggesting a better fit to the training data.

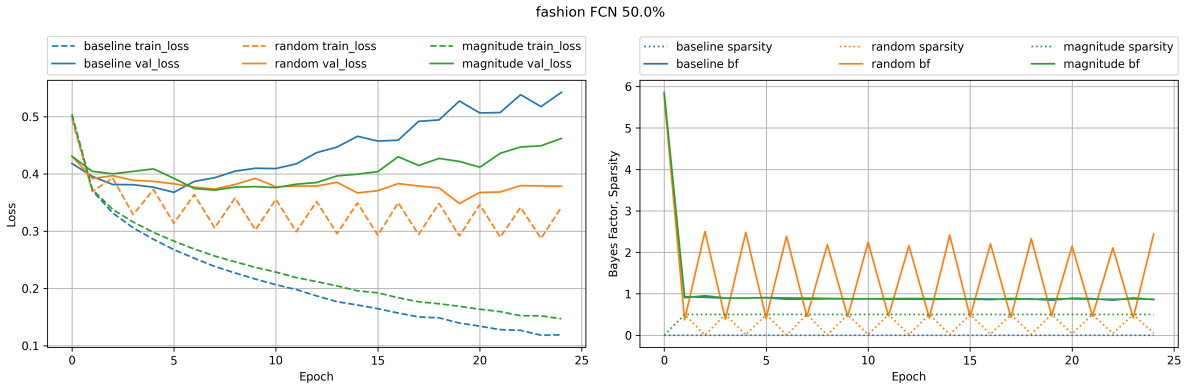

Figure 34: MNIST Fashion (FCN 50%) learning curves for the Bayesian pruning method.

Figure 34 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a FCN at 50% sparsity. Bayesian magnitude starts do better than baseline model, Bayesian random does better than both. Bayes factor for Bayesian random is better than baseline and Bayesian magnitude, suggesting a better fit to the training data.

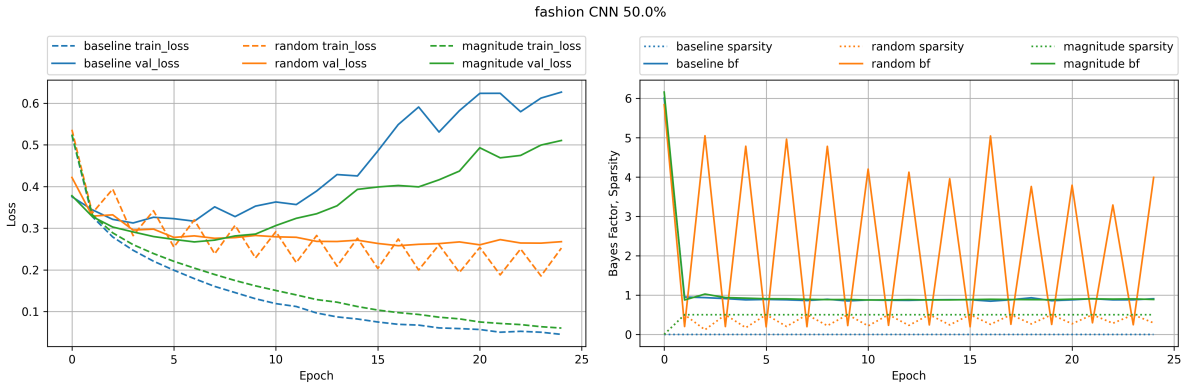

Figure 35: MNIST Fashion (CNN 50%) learning curves for the Bayesian pruning method.

Figure 35 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a CNN at 50% sparsity. Both the baseline and Bayesian magnitude methods show similar levels of overfitting. Only Bayesian random pruning is able to combat overfitting to some extent. Bayes factor remains close to one for all methods, suggesting a similar fit to training data.

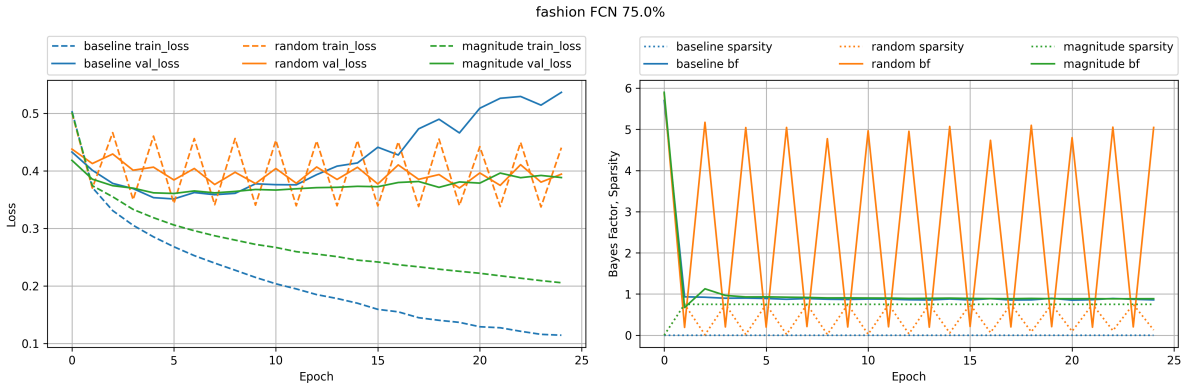

Figure 36: MNIST Fashion (FCN 75%) learning curves for the Bayesian pruning method.

Figure 36 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a FCN at 75% sparsity. Both Bayesian random and Bayesian magnitude combats overfitting compared to the baseline model. Bayes factor for Bayesian random is better than baseline and Bayesian magnitude, suggesting a better fit to the training data.

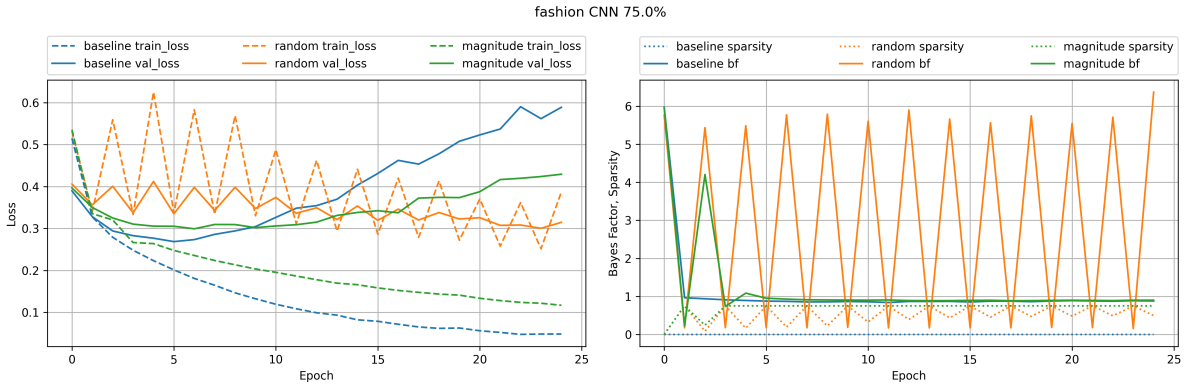

Figure 37: MNIST Fashion (CNN 75%) learning curves for the Bayesian pruning method.

Figure 37 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a CNN at 75% sparsity. Bayesian magnitude does slightly better than baseline, Bayesian random does better than both to combat overfitting. Bayes factor for Bayesian random is better than baseline and Bayesian magnitude, suggesting a better fit to the training data.

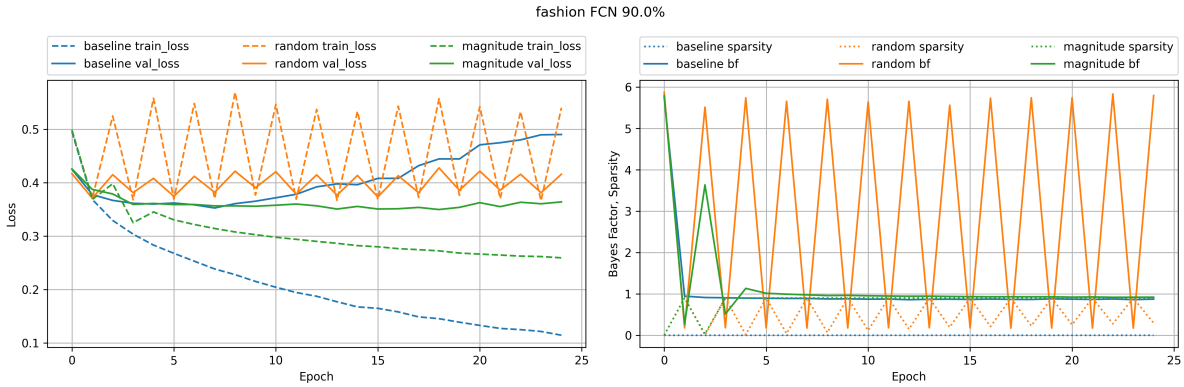

Figure 38: MNIST Fashion (FCN 90%) learning curves for the Bayesian pruning method.

Figure 38 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a FCN at 90% sparsity. Bayesian magnitude does best in terms of combatting overfitting. Bayes factor for Bayesian random is higher than baseline and Bayesian magnitude, suggesting a better fit to the training data.

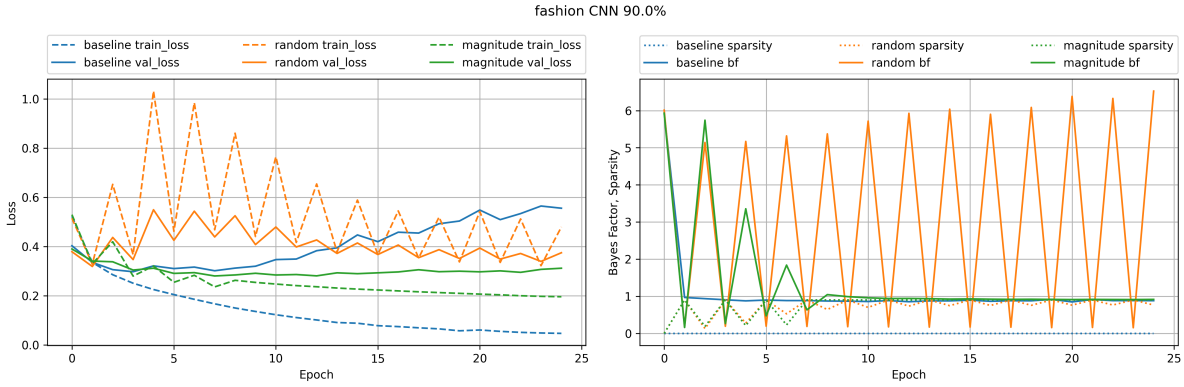

Figure 39: MNIST Fashion (CNN 90%) learning curves for the Bayesian pruning method.

Figure 39 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a CNN at 90% sparsity. Bayesian magnitude does best in terms of combatting overfitting. Bayes factor for Bayesian random is higher than baseline and Bayesian magnitude, suggesting a better fit to the training data.

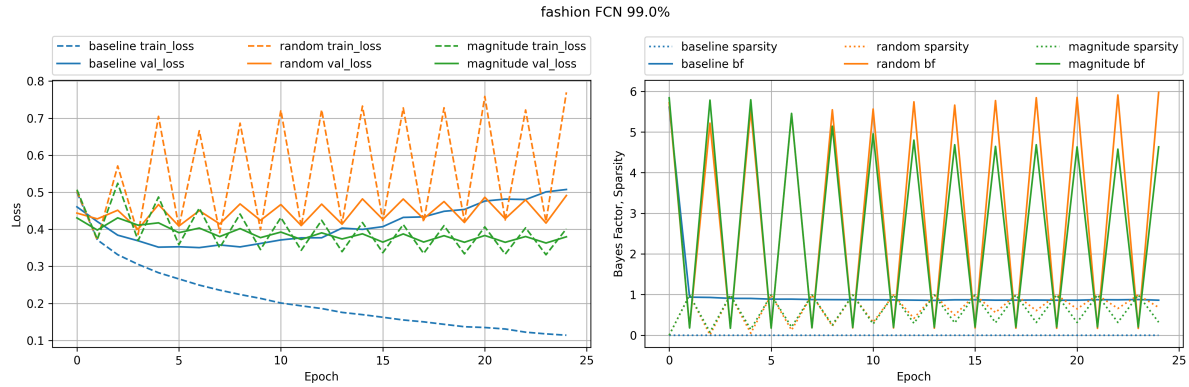

Figure 40: MNIST Fashion (FCN 99%) learning curves for the Bayesian pruning method.

Figure 40 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a FCN at 99% sparsity. Bayesian magnitude does best in terms of combatting overfitting. Bayes factor for Bayesian random and Bayesian magnitude remain higher than baseline, suggesting a better fit to the training data.

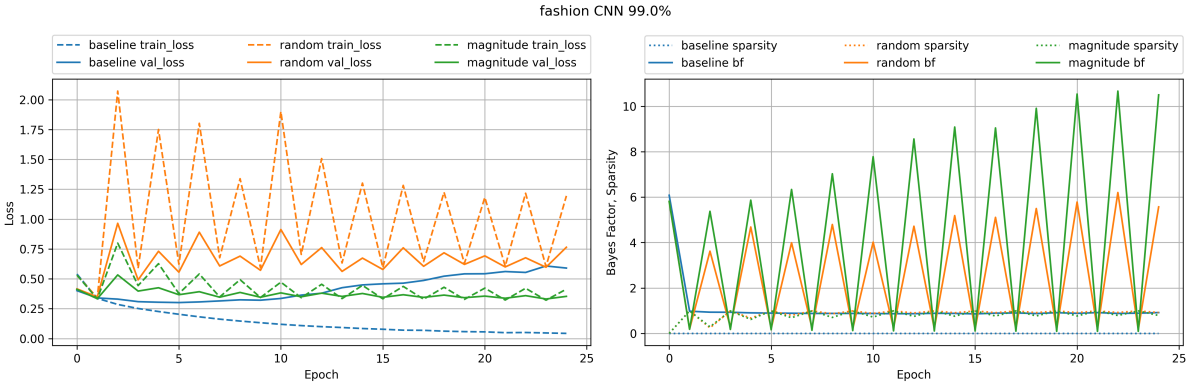

Figure 41: MNIST Fashion (CNN 99%) learning curves for the Bayesian pruning method.

Figure 41 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a CNN at 99% sparsity. Bayesian magnitude does best in terms of combatting overfitting. Bayes factor for Bayesian magnitude remain higher, suggesting a better fit to the training data.

CIFAR-10 Learning Curves

The following figures show the learning curves for the Bayesian pruning method on the MNIST dataset for a FCN, CNN at different levels of sparsity.

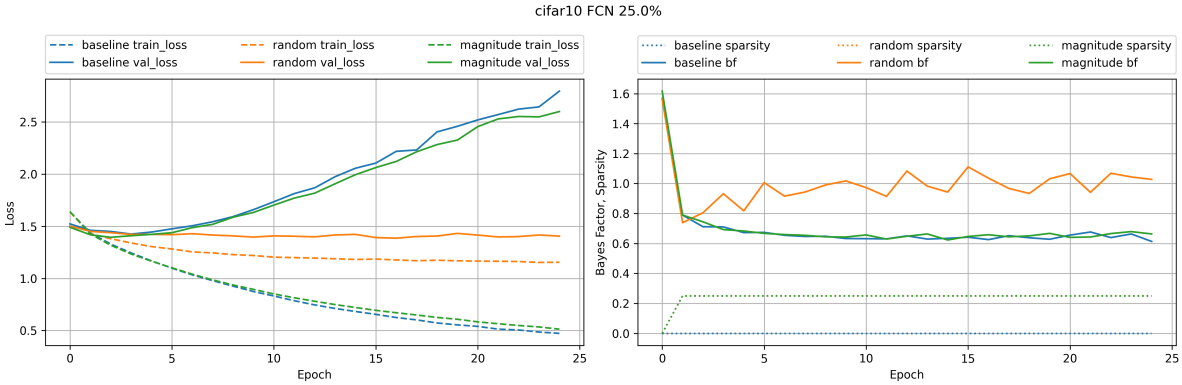

Figure 42: CIFAR-10 (FCN 25%) learning curves for the Bayesian pruning method.

Figure 42 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a FCN at 25% sparsity. Both the baseline and Bayesian magnitude methods validation loss deteriorate as training progresses. Only Bayesian random pruning is able to maintain a low validation loss. Pruning only 25 % of the weights with the lowest magnitude does not help combat overfitting as there is still sufficient weights to memorize the training data. The Bayesian random pruning method is able to maintain a low validation loss because it prunes weights randomly, and thus is able to prune weights that are not necessarily the least important weights. Bayes factor remains below one at the same level as the baseline method for the Bayesian magnitude method which indicates that the model is not a good fit for the data throughout the training epochs.

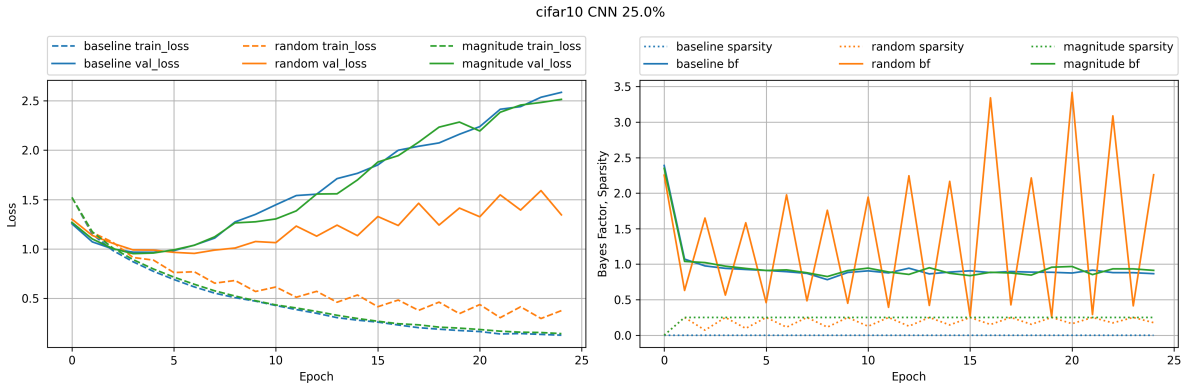

Figure 43: CIFAR-10 (CNN 25%) learning curves for the Bayesian pruning method.

Figure 43 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a CNN at 25% sparsity. Baseline, Bayesian random and Bayesian magnitude methods validation loss deteriorate as training progresses. Only random pruning is able to combat overfitting to some extent. Bayes factor lies close to one for the Bayesian magnitude method which indicates that the model is a better fit compared to the FCN model at 25% sparsity.

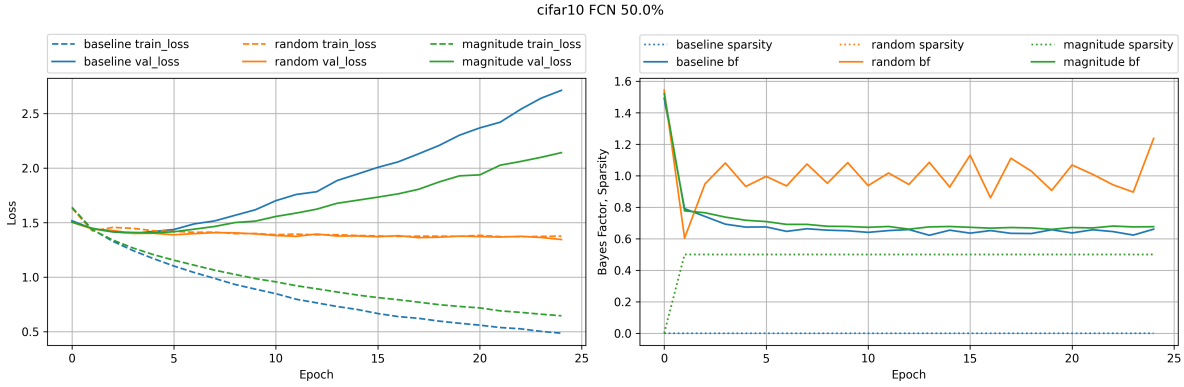

Figure 44: CIFAR-10 (FCN 50%) learning curves for the Bayesian pruning method.

Figure 44 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a FCN at 50% sparsity. Both the baseline and Bayesian magnitude methods validation loss deteriorate as training progresses. Only random pruning is able to combat overfitting. Bayes factor remains below one at the same level as the

baseline method for the Bayesian magnitude method which indicates that the model is not a good fit for the data throughout the training epochs.

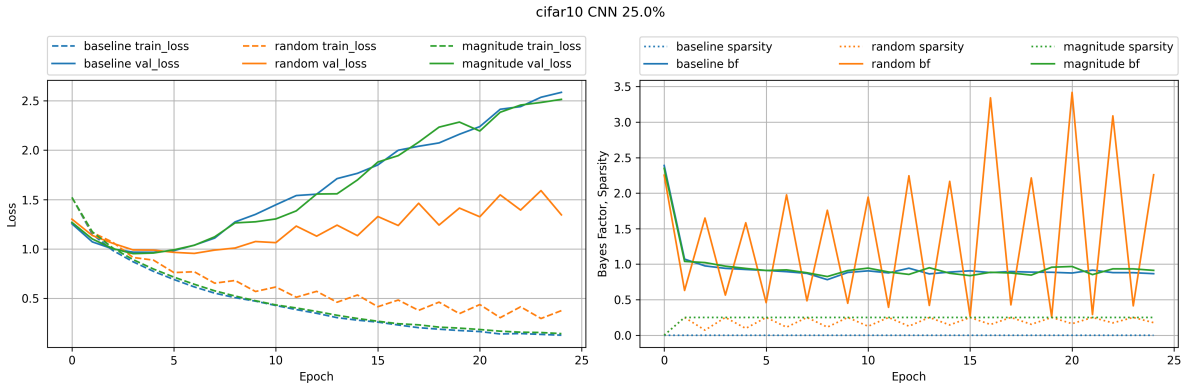

Figure 45: CIFAR-10 (CNN 50%) learning curves for the Bayesian pruning method.

Figure 45 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a CNN at 50% sparsity. Both the baseline, Bayesian Random and Bayesian magnitude methods validation loss deteriorate as training progresses. Only random pruning is able to combat overfitting to some extent. Bayes factor lies close to one for the Bayesian magnitude method which indicates that the model is a better fit compared to the CNN model at 25% sparsity. Bayes factor lies close to one for the Bayesian magnitude method which indicates that the model is a better fit compared to the FCN model at 50% sparsity.

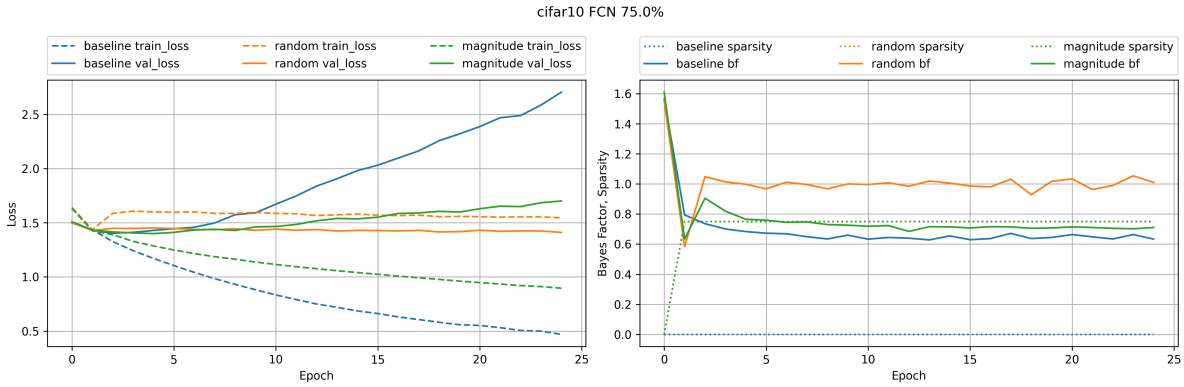

Figure 46: CIFAR-10 (FCN 75%) learning curves for the Bayesian pruning method.

Figure 46 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a FCN at 75% sparsity. Both the baseline, Bayesian Random and Bayesian magnitude methods validation loss deteriorate as training progresses. Only random pruning is able to combat overfitting. Bayes factor remains below one and slightly better than the baseline method for the Bayesian magnitude method which indicates that the model is not a good fit for the data but better than baseline model.

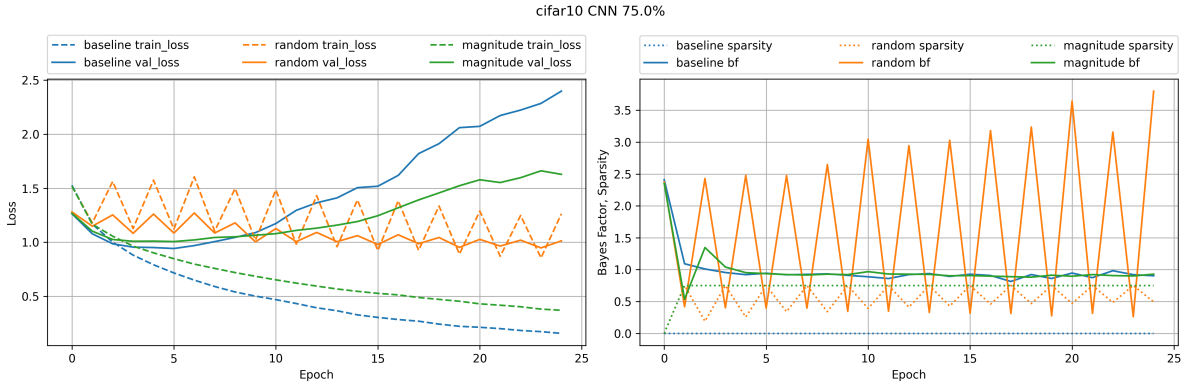

Figure 47: CIFAR-10 (CNN 75%) learning curves for the Bayesian pruning method.

Figure 47 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a CNN at 75% sparsity. Both the baseline and Bayesian magnitude methods validation loss deteriorate as training progresses. Only random pruning is able to combat overfitting to some extent. Bayes factor lies below one for the Bayesian magnitude method but better compared to the FCN model at 75%.

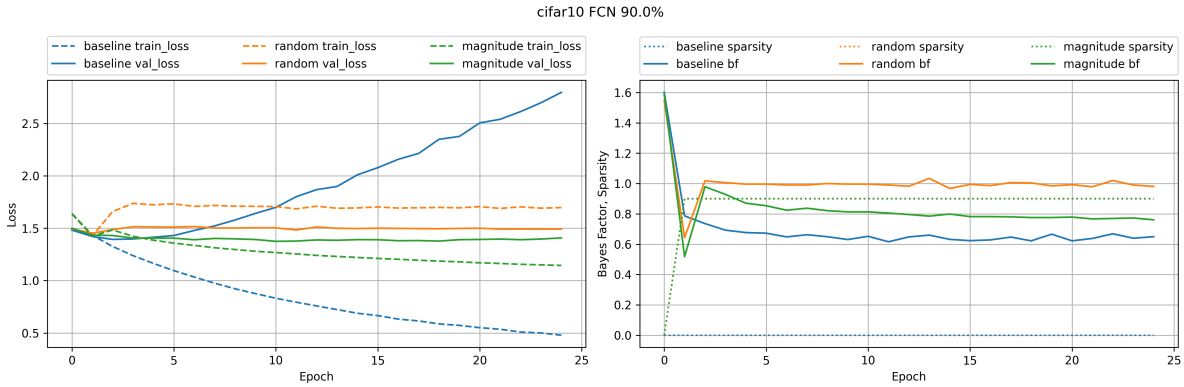

Figure 48: CIFAR-10 (FCN 90%) learning curves for the Bayesian pruning method.

Figure 48 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a FCN at 90% sparsity. Both the Bayesian Random and Bayesian magnitude methods are able to combat overfitting. Bayes factor remains below one and slightly better than the baseline method for the Bayesian magnitude method which indicates that the model is not a good fit for the data but better than baseline model. Bayesian Random method fits the data better.

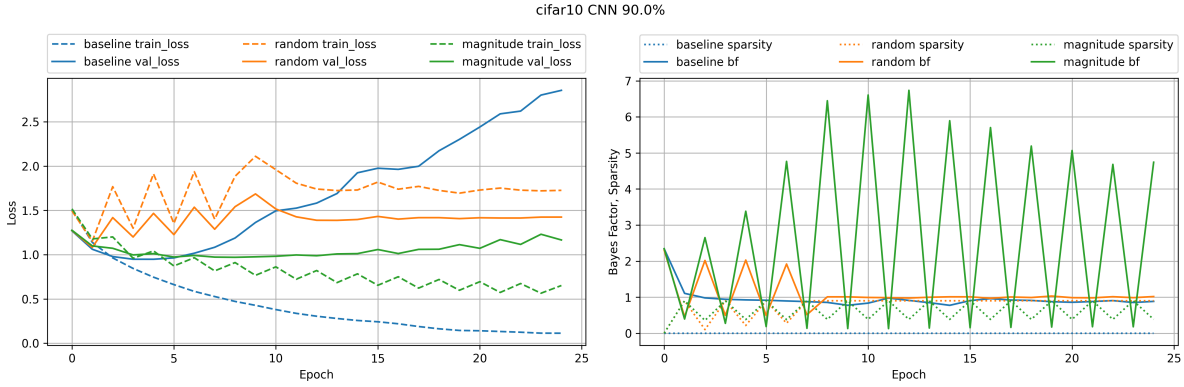

Figure 49: CIFAR-10 (CNN 90%) learning curves for the Bayesian pruning method.

Figure 49 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a CNN at 90% sparsity. Both the baseline and Bayesian magnitude methods validation loss deteriorate as training progresses. Only random pruning is able to combat overfitting to some extent. Bayes factor remains high and fluctuating

throughout training suggesting a better fit with fewer parameters.

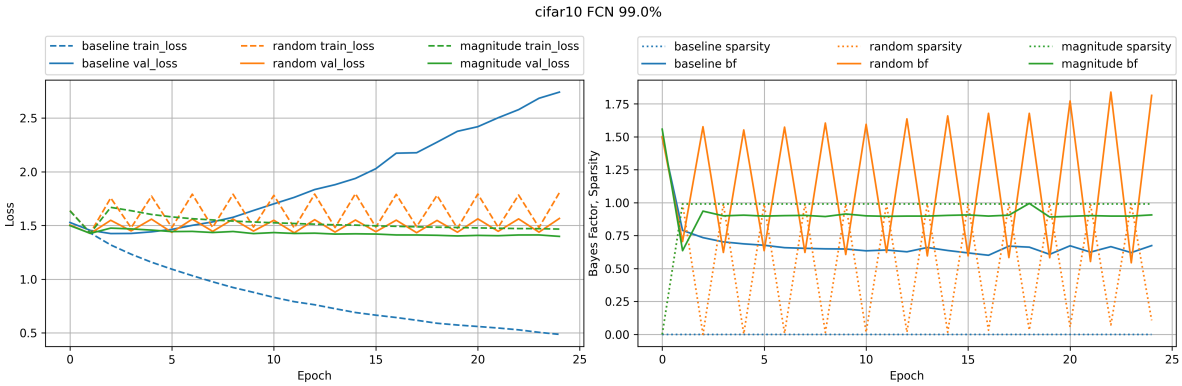

Figure 50: CIFAR-10 (FCN 99%) learning curves for the Bayesian pruning method.

Figure 50 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a FCN at 99% sparsity. Both Bayesian random and Bayesian magnitude pruning methods are able to combat overfitting. Bayes factor remains below one and better than the baseline method for the Bayesian magnitude method which indicates that the model is not a good fit for the data but better than baseline model. Bayesian Random method fits the data better.

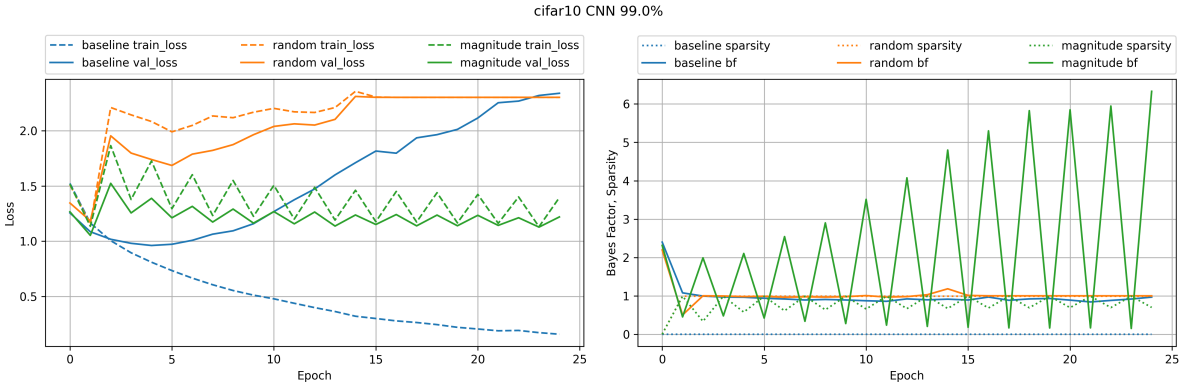

Figure 51: CIFAR-10 (CNN 99%) learning curves for the Bayesian pruning method.

Figure 51 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a CNN at 99% sparsity. The validation loss deteriorates for Bayesian random pruning from the beginning of the training period as 99% of weights are

pruned. Bayesian magnitude method is able to combat overfitting. Bayes factor remains high throughout training for the Bayesian magnitude method. The model is able to fit the data better with fewer parameters.

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 networks. *arXiv preprint arXiv:1505.05424*.

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*.

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.

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

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.

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

Zhou, S., Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou (2021). Rethinking the value of network pruning. *arXiv preprint arXiv:2104.06937*.

Zhu, M. and S. Gupta (2017). To prune, or not to prune: exploring the efficacy of pruning for model compression.