Pruning a neural network using Bayesian Inference

Journal:	*IEEE Transactions on Neural Networks and Learning Systems*
Manuscript ID	TNNLS-2024-P-32985
Manuscript Type:	Regular Paper
Keywords:	Bayesian pruning, Bayesian model selection, Bayes Factors, Neural Network Compression

Pruning a neural network using Bayesian Inference

***Abstract*— Neural network pruning is an effective technique for reducing the computational and memory requirements of large neural networks. It has been proven to be useful in reducing the effects of overfitting and in some cases provide better performance than its original unpruned network. In this paper, we propose a method for pruning neural networks using Bayesian inference that can be incorporated into the training process. We evaluate our method on several standard benchmarks and show that it achieves high levels of sparsity while maintaining competitive accuracy.**

***Index Terms*—Bayesian pruning, Bayesian model selection, Bayes Factors, Neural network compression**

I. Introduction

In artificial neural nets (ANN) and machine learning (ML), parameters represent what the network has learned from the data. Over time as the computational capabilities from a hardware perspective have advanced, we have now become able to define larger models with millions of parameters. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) and its winners over the year show us how the error rate has dropped with increase of the number of parameters and connections in neural network. In 2012, one of the CNNs AlexNet [1] had 660K nodes, 61M parameters and over 600M connections. The state-of-the-art deep learning language model GPT-3 [2] comprises of 175 billion machine learning parameters. While deeper high parameter nets provide better results, the large number of connections introduce the problems of memory, overfitting, and lack of generalizability. There have been numerous methods developed to address these problems.

Regularization is one of the most popular methods to address the problem of overfitting. It is a technique that adds a penalty term to the loss function to prevent the model from overfitting. It is a very effective method to reduce the complexity of the model and to improve its generalizability. However, it is not a very effective method for reducing the number of parameters in the model.

Neural network pruning is a popular method that focuses on reducing the number of parameters of the model, thereby reducing the computational complexity and memory requirements of deep learning models [3]. They are key to deploying large models on resource constrained devices like mobile phones and tablets. Pruning 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 typically rely on heuristics or sensitivity analysis to determine which weights to remove, but these approaches can be suboptimal and do not provide a principled way of selecting the most important weights [4].

Several pruning methods have been proposed in the literature, including weight pruning, neuron pruning, and filter pruning [5], [6], [7]. 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 that are deemed unimportant. One of the most prominent neuron pruning technique is a process called “drop-out.”[8]. Dropout uses a heuristic to determine which connections can be randomly dropped during training. These methods can be effective for reducing the size of the network and improving its performance, but they typically rely on heuristics or sensitivity analysis to determine which weights or neurons to remove, which can be suboptimal and do not provide a principled way of selecting the most important ones [9].

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 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 has several advantages over traditional pruning methods, including a principled framework for selecting the most important weights, the ability to incorporate prior knowledge about the network structure, and the potential for improved performance when combined with other techniques such as importance weighting. The posterior distribution reflects our updated belief about the weights based on the observed data and can be used to calculate the probability that each weight is important for the task at hand. One common approach for approximating the posterior distribution is to use variational inference, which involves minimizing the Kullback-Leibler divergence between the true posterior and an approximate distribution [10]. Other approaches include Monte Carlo methods and Markov chain Monte Carlo (MCMC) sampling [11].

In this chapter, we propose a Bayesian pruning algorithm that uses approximate Bayesian inference to calculate the posterior distribution of the weights and determine which ones to prune. Bayesian pruning offers a principled approach for selecting the most important weights in a neural network and can lead to significant reductions in computational and memory requirements without sacrificing accuracy. We follow an approach similar to [12] to iteratively prune with varying levels of sparsity and to monitor accuracy for different levels of sparsity. Our approach uses Bayesian hypothesis testing to compute the pruning threshold. We evaluate our method on standard benchmarks and show that it achieves high levels of sparsity while maintaining competitive accuracy. Our approach also allows us to incorporate prior knowledge about the

structure of the network and can be extended to handle more complex pruning scenarios. Overall, our results demonstrate the potential of Bayesian pruning as a promising approach for reducing the complexity of deep neural networks.

II. Methods

A. Pruning Neural Networks Using Bayesian Inference

Figure 1 shows the block diagram of the pruning system. After a forward pass through the layers of the neural network which consists of a series of matrix multiplications and non-linear activations, the output of the network is compared with the ground truth labels to compute the loss.

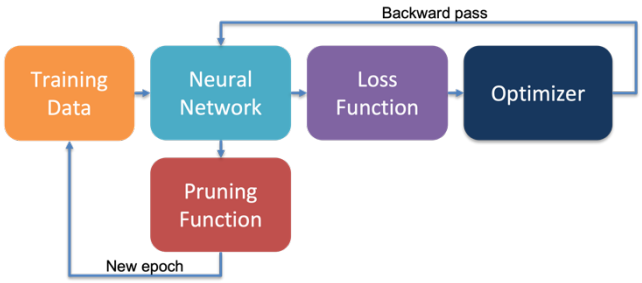

Fig. 1. Pruning system block diagram.

The loss is then backpropagated through the network to compute the gradients of the weights. The gradients are then used to update the weights using an optimizer such as SGD or Adam [13]. After the epoch is completed, the weights are pruned using the pruning algorithm. The pruned weights are then used in the next epoch.

Let ψ be the set of weights in the unpruned network and let ϕ be the set of weights in the pruned network. We want to test whether the pruned network fits the data better than the original unpruned network. The null hypothesis is that the unpruned network fits the data better than the unpruned network, i.e., $\theta = \psi$. The alternative hypothesis is that the pruned network fits the data better than the unpruned network, i.e., $\theta = \phi$.

To test these hypotheses, we compute the Bayes factor, which is the ratio of the posterior probability of the alternative hypothesis to the posterior probability of the null hypothesis:

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

where D is the training data.
We can compute the posterior probability of the null hypothesis as follows:

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

where $\mathrm{P}(D|\theta = \psi)$ is the likelihood of the data given the weights of the unpruned network, $\mathrm{P}(\theta = \psi)$ is the prior probability of the null hypothesis, and $\mathrm{P}(D)$ is the marginal likelihood of the data. $\mathrm{P}(\theta = \psi|D)$ represents the probability of the null hypothesis given the data.

Similarly, we can compute the posterior probability of the alternative hypothesis as follows:

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

where $\mathrm{P}(D|\theta = \phi)$ is the likelihood of the data given the weights of the pruned network, and $P(\theta = \phi)$ is the prior probability of the alternative hypothesis. $P(\theta = \phi|D)$ represents the probability of the alternative hypothesis given the data.
We can then compute the Bayes factor 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 value greater than 1 indicates that the pruned network fits the data better than the unpruned network. A Bayes factor value less than 1 indicates that the unpruned network fits the data better than the pruned network. We introduce shared prior distributions for the layer-wise pruning by specifying a common sparsity-inducing prior,

$$p\left(W^{(l)}|\theta^{(l)}\right) = \prod_{i=1}^{N^{(l)}} p\left(w_i^{(l)}|\theta^{(l)}\right)$$

where $W^{(l)}$ is the weight matrix of layer l, N^l is the number of weights in layer l, and $w_i^{(l)}$ is the ith weight in layer l. The prior distribution $p\left(w_i^{(l)}|\theta^{(l)}\right)$ is a function of the hyperparameters $\theta^{(l)}$ that specify the sparsity-inducing prior.

Here we train the neural network on the training set using stochastic gradient descent. We compute the posterior distribution of each weight using a Gaussian prior with mean μ and variance σ^2, where σ is a hyperparameter that controls the strength of the prior.

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

For a classification problem, the likelihood of the data is given by:

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

where $\mathcal{C}$ is the categorical cross-entropy loss function, y_{pred} is the neural network prediction for the classes and y_{true} is the ground truth. We calculate the log prior and log likelihood for the weight parameters to get the posterior distribution of the weights with the following equation:

$$\log p\,(w|D) = \log p\,(w) + \log p\,(D|w)\backslash$$
$$= \sum_{i=1}^{n} \log p\,(y_i|f(x_i), w) + \log p\,(w)$$

We use an iterative pruning algorithm that is incorporated to the training process. Pruning is performed after each epoch of training. The Bayes factor is used to determine whether to prune or not to achieve a desired level of sparsity. The percentage of weights to prune is a hyperparameter that can be

tuned to achieve the desired level of sparsity. The algorithm is summarized in Algorithm 1.

Algorithm 1 Bayesian Pruning Algorithm

Input: Trained neural network $f(\cdot, \theta)$, pruning rate r, dataset $\mathcal{D} = (xi, y_i)_{i=1}^{n}$, β Bayes factor threshold
Output: Pruned neural network $f_r(\cdot, \theta)$
1: Compute the posterior probability of the weights before pruning.
2: **If** $BF_{01} > \beta$ **then**
3: Prune r percentage of weights $f(\cdot, \theta)$
4: **end if**
5: Compute the posterior probability of the weights after pruning.
6: Compute the Bayes factor using the posterior probabilities before and after pruning.

In the following sections we introduce two pruning algorithms that use the above framework. The first algorithm is random pruning, which randomly selects weights to prune. The second algorithm is magnitude pruning, which selects weights to prune based on their magnitude.

B. 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 Pruning Algorithm

1: $f(\cdot, \theta)$: Neural network model with parameters θ
2: *r:* Desired sparsity level, β Bayes factor threshold
3: Calculate log posterior probability $p(\theta|D)$
4: **If** $BF_{01} > \beta$ **then**
5: **for all** weights $w_i \in \theta$ **do**
6: $n \leftarrow$ size(w_i)
7: number of weights to prune, $k \leftarrow (n \times r)$
8: $I \leftarrow$ indices of non-zero weights
9: $n_z \leftarrow$ number of zero weights
10: $k' \leftarrow k - n_z$
11: $J \leftarrow$ random sample(I, k')
12: set elements in w_i at indices J to zero
13: **end for**
14: **end if**
15: Calculate log posterior probability $p(\theta|\mathrm{D})$ after pruning
16: Calculate Bayes factor BF_{01}

C. 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 2 Bayesian Pruning Algorithm

1: $f(\cdot, \theta)$: Neural network model with parameters θ
2: *r:* Desired sparsity level, β Bayes factor threshold
3: Calculate log posterior probability $p(\theta|D)$
4: **If** $BF_{01} > \beta$ **then**
5: **for all** weights $w_i \in \theta$ **do**
6: $n \leftarrow$ size(w_i)
7: number of weights to prune, $k \leftarrow (n \times r)$
8: $w_i \leftarrow$ sort(w_i)
9: set k element in w_i to zero
10: **end for**
11: **end if**
12: Calculate log posterior probability $p(\theta|\mathrm{D})$ after pruning
13: Calculate Bayes factor BF_{01}

D. Datasets

MNIST dataset

The MNIST dataset [14] 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. The network is trained for 50 epochs.

MNIST-Fashion dataset

The MNIST-Fashion dataset, as described in the study by Xiao et al. (2017) [15] comprises 60,000 training images and 10,000 test images featuring various fashion items. Each image has a resolution of 28×28 pixels and is presented in grayscale. To ensure uniformity, the images are normalized to possess zero mean and unit variance. Prior to processing through the fully connected neural network, the images are flattened into a 784-dimensional vector. The objective of the network is to

categorize the images into one of the 10 classes. The training process involves 50 epochs.

CIFAR-10 dataset

The CIFAR-10 dataset [16] 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. The network is trained for 100 epochs.

D. Neural Network Architecture

The network architecture of the fully connected network (FCN) seen in Figure 2 is as follows:

$$\boldsymbol{h}_1 = ReLU(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)$$
$$\boldsymbol{h}_2 = ReLU(\boldsymbol{W}_2\boldsymbol{h}_1 + \boldsymbol{b}_2)$$
$$\boldsymbol{h}_3 = ReLU(\boldsymbol{W}_3\boldsymbol{h}_2 + \boldsymbol{b}_3)$$
$$\boldsymbol{y} = \boldsymbol{W}_4\boldsymbol{h}_3 + \boldsymbol{b}_4 \quad (1)$$

Where $\mathbf{x}$ is the input, $\mathbf{h}_1$, $\mathbf{h}_2$ and $\mathbf{h}_3$ are the three hidden layers, $\mathbf{y}$ is the output, $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{W}_3$ and $\mathbf{W}_4$ are the weight matrices, $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$ and $\mathbf{b}_4$ are the bias vectors and ReLU(.) is the rectified linear unit activation function.

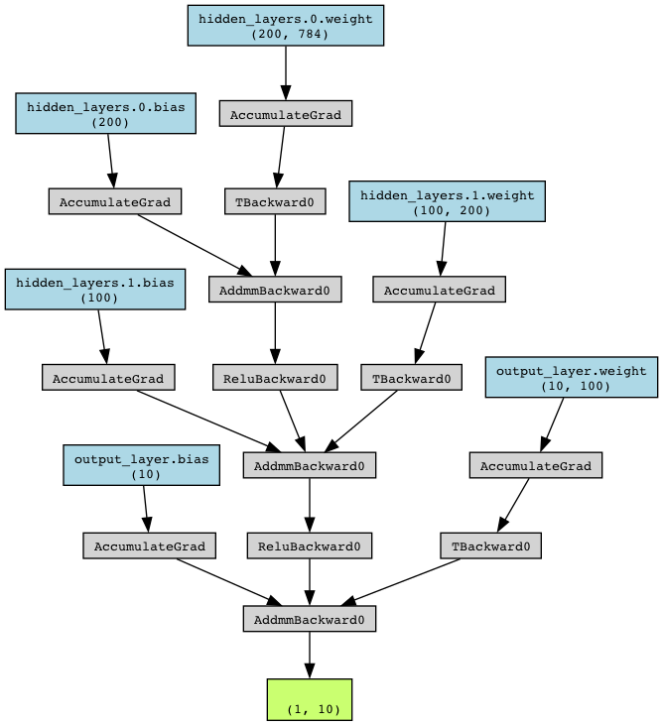

Fig. 2. Representation of Fully connected neural network architecture, given in Equation (1). See text for further details.

Convolutional neural network (CNN)

The neural network architecture of the convolutional neural network (CNN) seen in Figure 3 is as follows:

$$\mathbf{h}_1 = \text{ReLU}(\text{Conv2d}(\mathbf{x}, \mathbf{W}_1) + \mathbf{b}_1)$$
$$\mathbf{h}_2 = \text{MaxPool2d}(\mathbf{h}_1)$$
$$\mathbf{h}_3 = \text{ReLU}(\text{Conv2d}(\mathbf{h}_2, \mathbf{W}_2) + \mathbf{b}_2)$$
$$\mathbf{h}_4 = \text{MaxPool2d}(\mathbf{h}_3)$$
$$\mathbf{h}_5 = \text{ReLU}(\mathbf{W}_1\mathbf{h}_4 + \mathbf{b}_1)$$
$$\mathbf{h}_6 = \text{ReLU}(\mathbf{W}_2\mathbf{h}_5 + \mathbf{b}_2)$$
$$\mathbf{h}_7 = \text{ReLU}(\mathbf{W}_3\mathbf{h}_6 + \mathbf{b}_3)$$
$$\mathbf{y} = \mathbf{W}_8\mathbf{h}_7 + \mathbf{b}_8 \quad (2)$$

where $\mathbf{x}$ is the input, $\mathbf{h}_1$, $\mathbf{h}_2$, $\mathbf{h}_3$, $\mathbf{h}_4 \cdots \mathbf{h}_7$ are the hidden layers, $\mathbf{y}$ is the output, $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{W}_3$ and $\mathbf{W}_4$ are the weight matrices, $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$ and $\mathbf{b}_4$ are the bias vectors, Conv2d(·) is the convolutional layer, MaxPool2d(·) is the max pooling layer, and ReLU(·) is the rectified linear unit activation function.

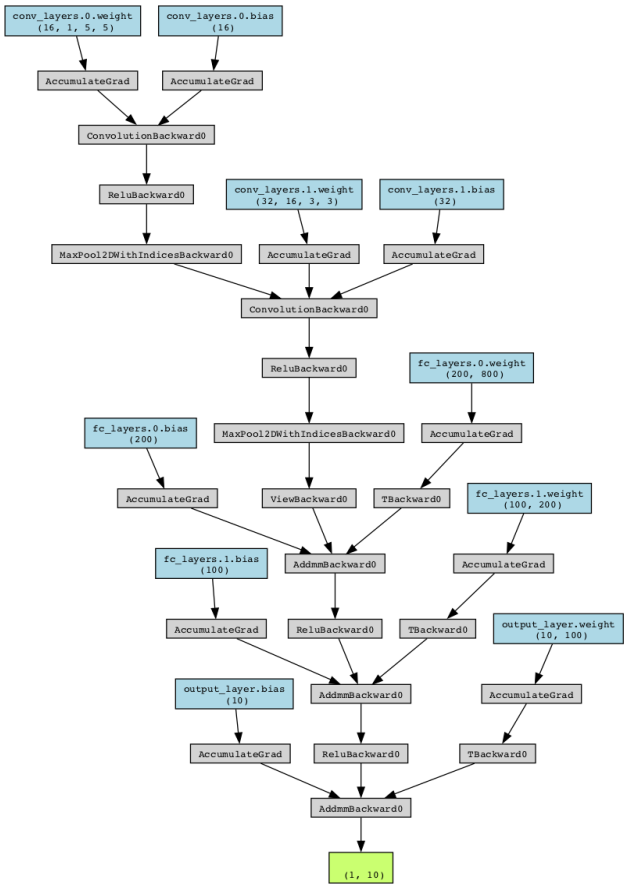

Fig.3. Representation of Convolutional neural network architecture, given in Equation (2). See text for further details.

D. Optimization

There are several optimization algorithms that can be used for training a neural network. The log posterior loss function is a non-convex function. It can have multiple local minima and saddle points. Gradient descent is a first-order optimization algorithm that can be used to find a local minimum of a function. However, it is not guaranteed to find the global minimum. Gradient descent is also sensitive to the learning rate. If the learning rate is too small, the algorithm will take a long time to converge. If the learning rate is too large, the algorithm may not converge at all. Stochastic gradient descent (SGD) is a variant of gradient descent that uses a random sample of the training data to estimate the gradient. This reduces the computational cost of each iteration and allows the algorithm to converge faster. However, SGD is also sensitive to the learning rate. Adam is an adaptive learning rate optimization algorithm that uses the first and second moments of the gradient to adaptively adjust the learning rate. It is more robust to the choice of learning rate and converges faster than SGD. So, we will be using an Adam optimizer Kingma and Ba (2015) for training our neural networks.

E. Implementation

Our implementation uses the PyTorch framework [17] for Bayesian inference. We use a Gaussian prior with mean 0 and variance σ^2 as the prior distribution for the weights, where σ is a hyperparameter that controls the strength of the prior. 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 fewer 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.

One experiment consists of evaluating a pruning method with one dataset, two different neural network architectures (FCN and CNN) and five different levels of desired sparsity. The levels of sparsity are 25%, 50%, 75%, 90% and 99%. We also plot the learning curves for each method, showing the evolution of the accuracy and sparsity during training. Both pruning techniques are not global, meaning the final sparsity level is achieved by pruning each layer individually.

III. RESULTS

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 B.

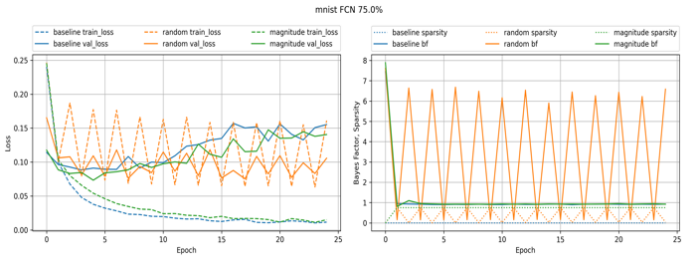

Figure 4: Learning curves for a FCN with MNIST dataset.

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.

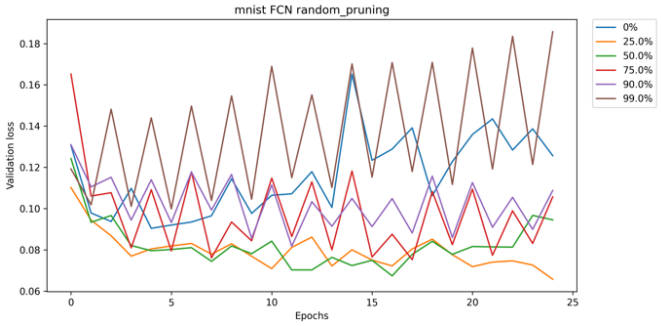

Figure 5: Validation loss of random pruning for different sparsity levels.

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 end and combats overfitting compared to the

baseline. The network only starts to become worse at 99% sparsity.

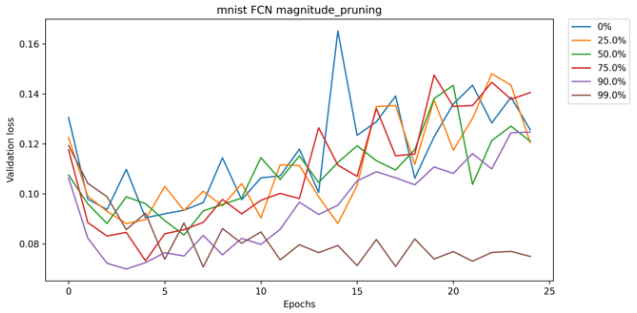

Figure 6: Validation loss of magnitude pruning for different sparsity levels.

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.

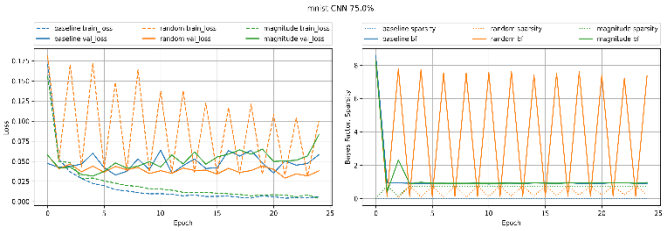

Figure 7: Learning curves for CNN with MNIST dataset.

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. The trends in the learning curves are similar to that of the FCN. 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.

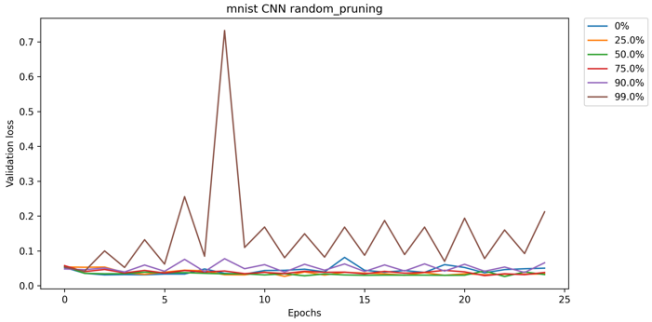

Figure 8: Validation loss of random pruning for different sparsity levels.

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.

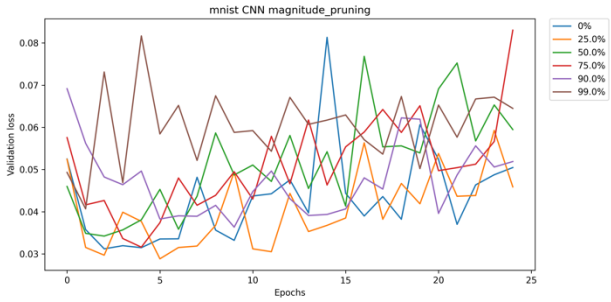

Figure 9: Validation loss of magnitude pruning for different sparsity levels.

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 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 C.

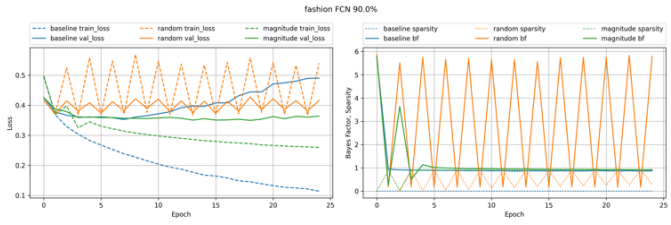

Figure 10: Learning curves for FCN with MNIST Fashion dataset.

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 is 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.

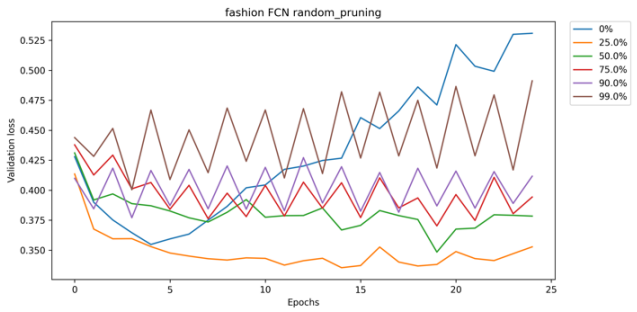

Figure 11: Validation loss of random pruning for different sparsity levels.

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.

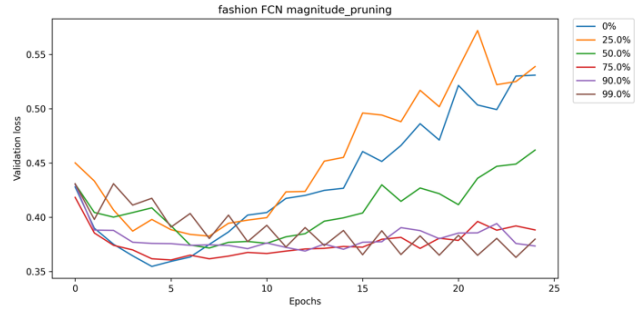

Figure 12: Validation loss of magnitude pruning for different sparsity levels.

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

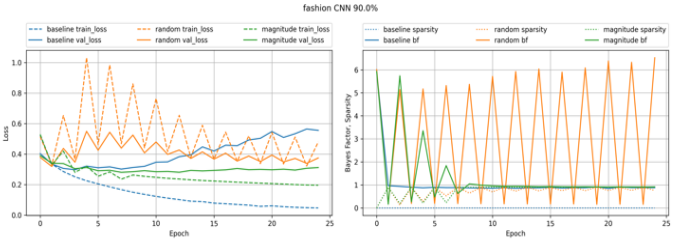

Figure 13: Learning curves for CNN with MNIST Fashion dataset.

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.

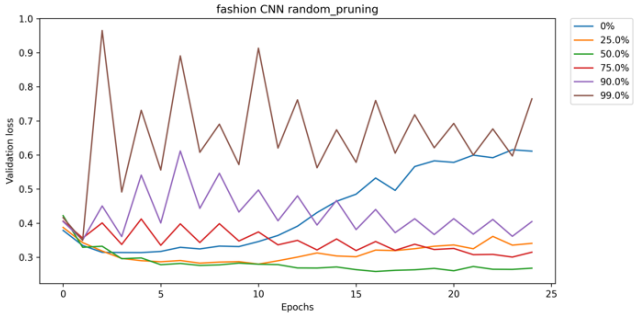

Figure 14: Validation loss of random pruning for different sparsity levels.

Figure 14 shows the validation accuracy of random pruning for different sparsity levels. The trends are similar 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.

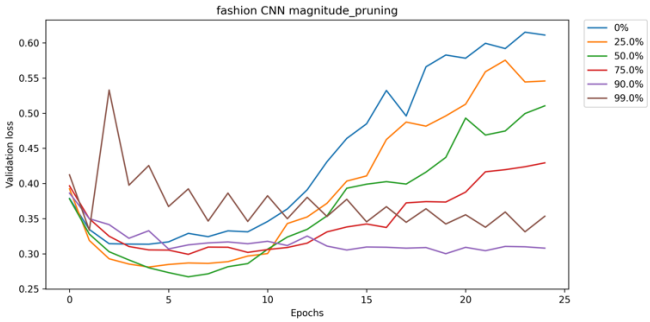

Figure 15: Validation loss of magnitude pruning for different sparsity levels.

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 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 D

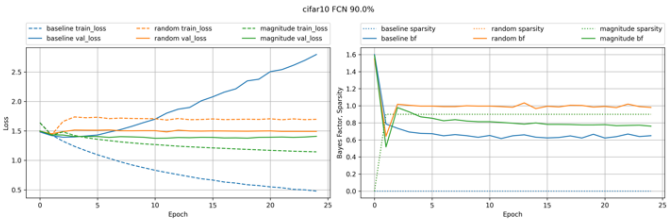

Figure 16: Learning curves for FCN with CIFAR-10 dataset.

Unlike the MNIST, Fashion datasets the input images of CIFAR-10 dataset are of size $32 \times 32 \times 3$. 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.

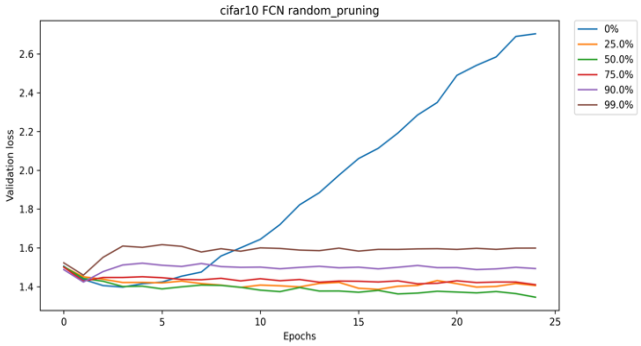

Figure 17: Validation loss of random pruning for different sparsity levels.

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.

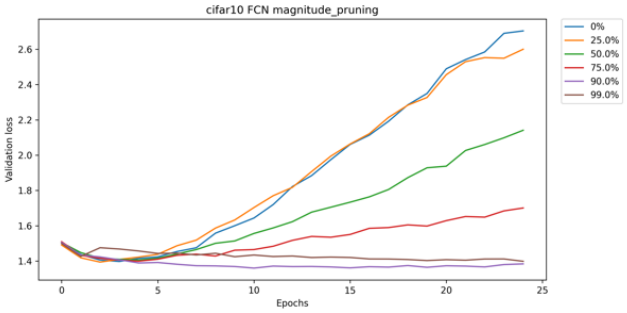

Figure 18: Validation loss of magnitude pruning for different sparsity levels.

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. Magnitude pruning helps in reducing the effects of overfitting. The validation loss continues to improve as 99% sparsity is achieved.

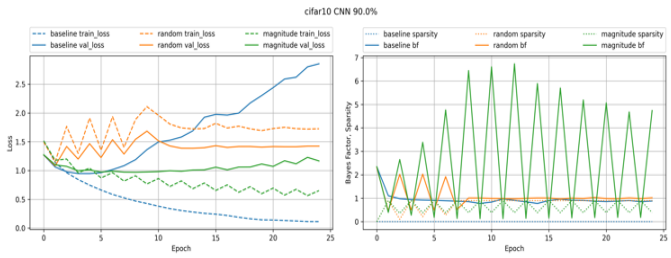

Figure 19: Learning curves for CNN with CIFAR-10 dataset.

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.

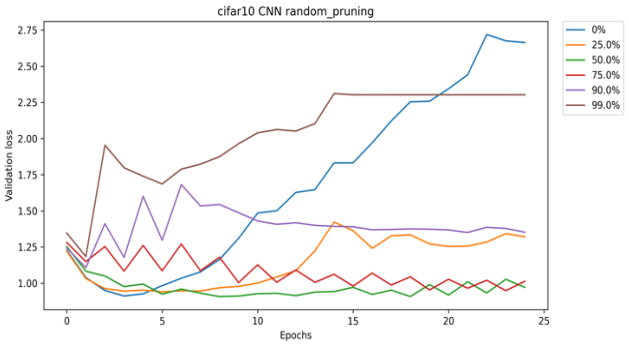

Figure 20: Validation loss of random pruning for different sparsity levels.

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

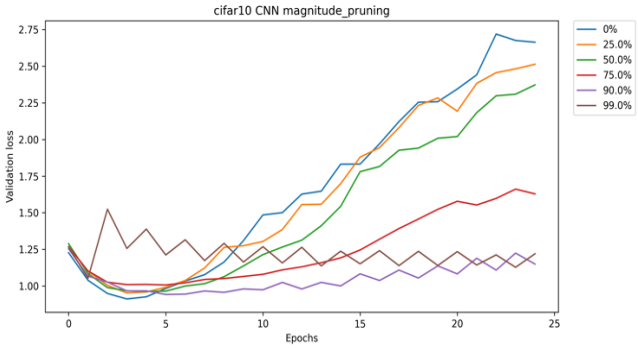

Figure 21: Validation loss of magnitude pruning for different sparsity levels.

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 shows the accuracy values of the pruned networks at different sparsity levels after training for 25 epochs. The experiment was repeated 5 times with different seeds for the random generator and averaged to get the results. The results show that the Bayesian pruning method can prune the networks to a higher sparsity level without losing any accuracy. They perform better than unpruned networks and have comparable or better accuracy than baseline neural network pruning methods.

IV. CONCLUSION

Neural networks with a very high number of parameters are capable of learning complex functions. Based on the size of the dataset they may overfit and produce undesirable results. They also cannot be deployed on compute constrained devices like mobile phones, tablets, and other devices. Neural network pruning provides a solution to both the problems of overfitting and size. The results of the experiments on three different datasets and two different architectures show that a Bayesian approach to neural network pruning can provide a principled way of eliminating connections in a network. This can provide an effective way to train neural networks with fewer parameters.

Table 1: Accuracy value 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**

V. ACKNOWLEDGMENTS

We would like to acknowledge support for this project from the Wehr Foundation's Computational Sciences Summer Research Fellows Program (CSSRFP) at Marquette University's Department of Mathematical and Statistical Sciences.

APPENDIX A. BAYESIAN PRUNING LEANING 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.

APPENDIX A1. MNIST LEARNING CURVES

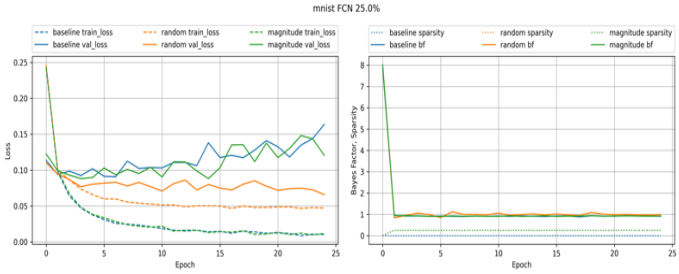

Figure 22: MNIST (FCN 25%) learning curve for the Bayesian pruning method.

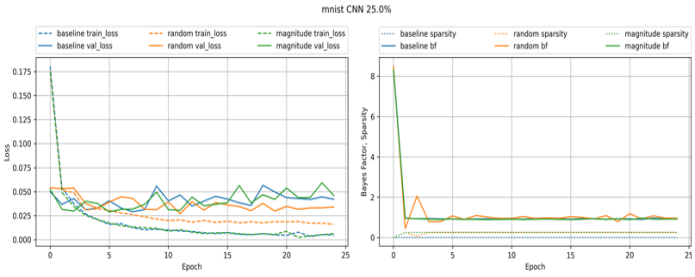

Figure 23: MNIST (CNN 25%) learning curve for the Bayesian pruning method.

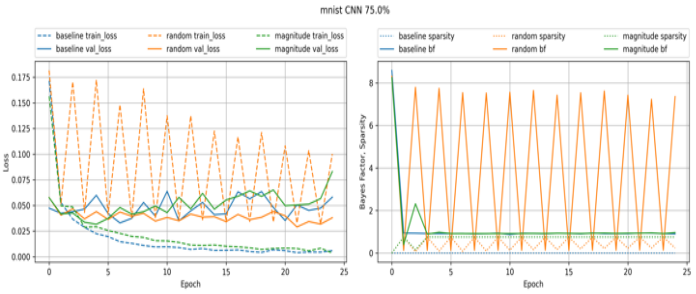

Figure 27: MNIST (CNN 75%) learning curve for the Bayesian pruning method.

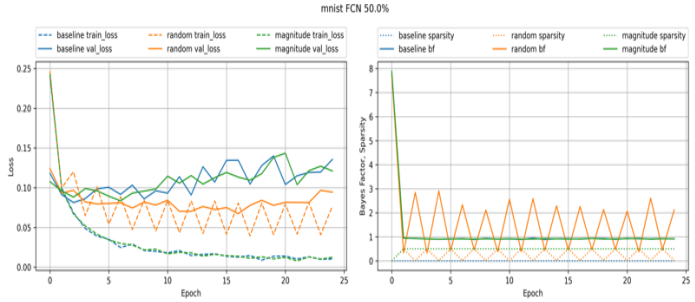

Figure 24: MNIST (FCN 50%) learning curve for the Bayesian pruning method.

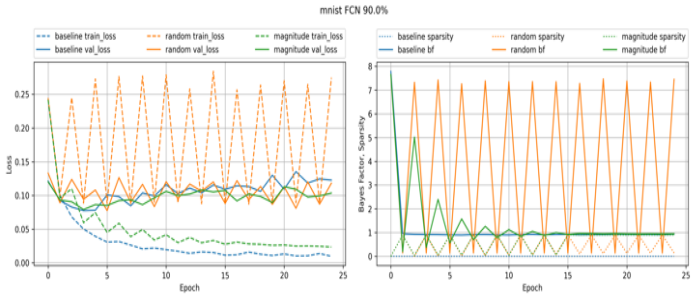

Figure 28: MNIST (FCN 90%) learning curve for the Bayesian pruning method.

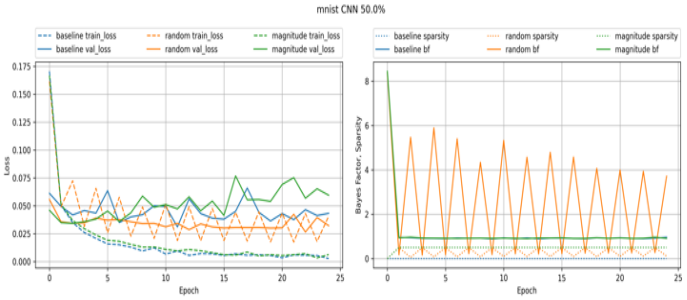

Figure 25: MNIST (CNN 50%) learning curve for the Bayesian pruning method.

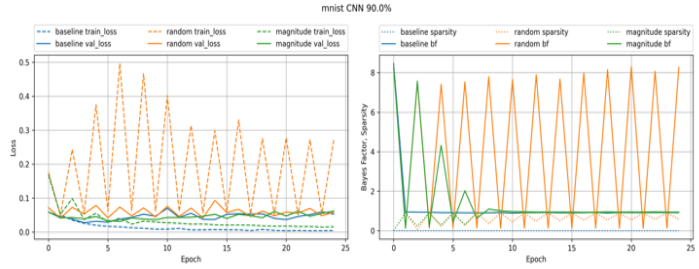

Figure 29: MNIST (CNN 90%) learning curve for the Bayesian pruning method.

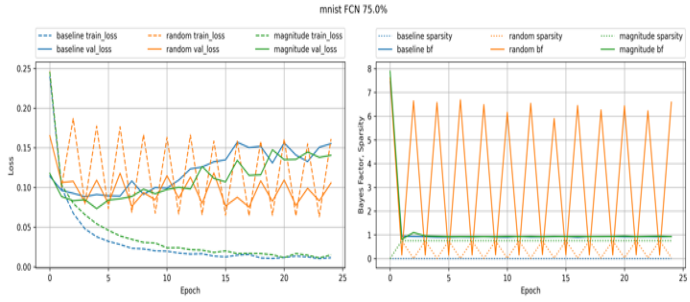

Figure 26: MNIST (FCN 75%) learning curve for the Bayesian pruning method.

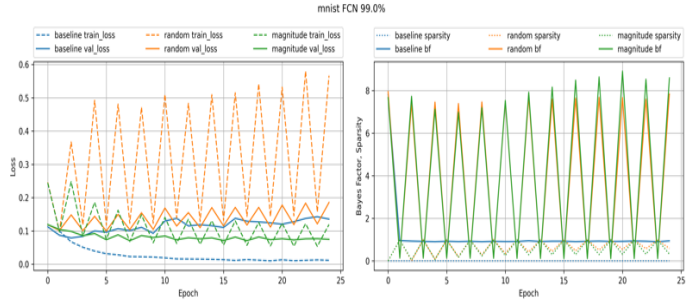

Figure 30: MNIST (FCN 99%) learning curve for the Bayesian pruning method.

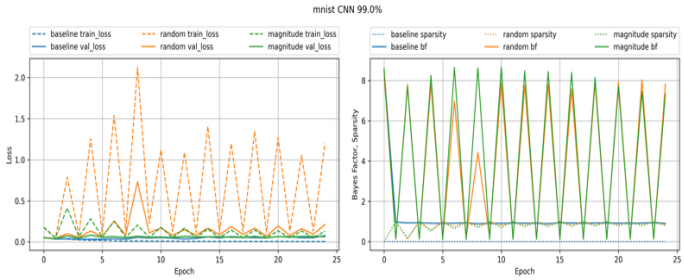

Figure 31: MNIST (CNN 99%) learning curve for the Bayesian pruning method.

APPENDIX A2. 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.

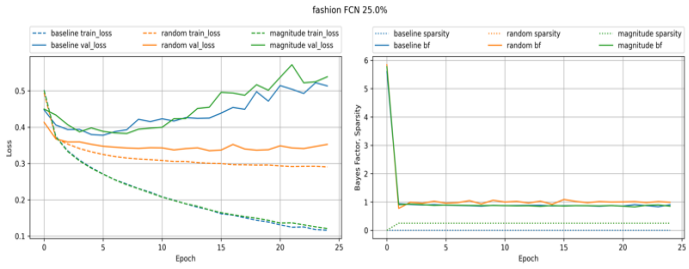

Figure 32: MNIST Fashion (FCN 25%) learning curve for the Bayesian pruning method.

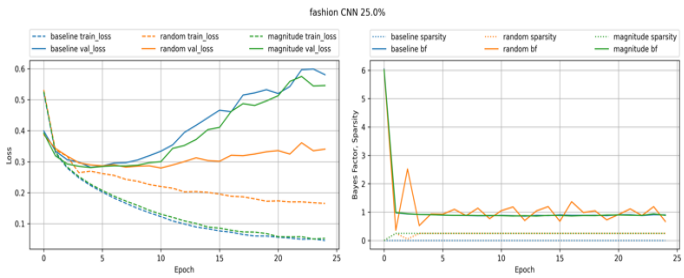

Figure 33: MNIST Fashion (CNN 25%) learning curve for the Bayesian pruning method.

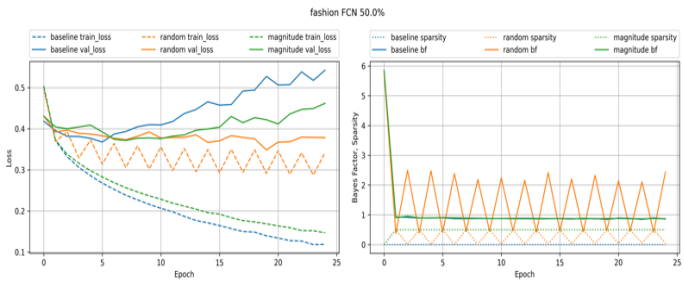

Figure 34: MNIST Fashion (FCN 50%) learning curve for the Bayesian pruning method.

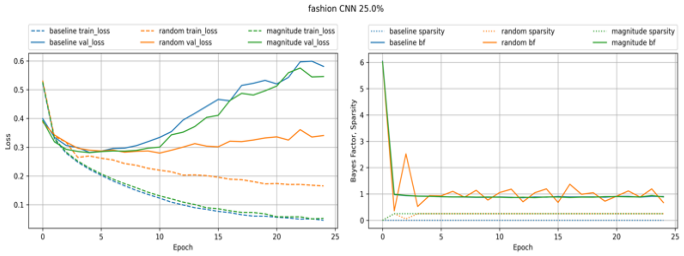

Figure 35: MNIST Fashion (CNN 50%) learning curve for the Bayesian pruning method.

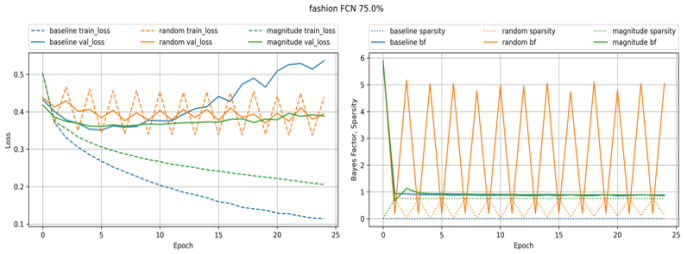

Figure 36: MNIST Fashion (FCN 75%) learning curve for the Bayesian pruning method.

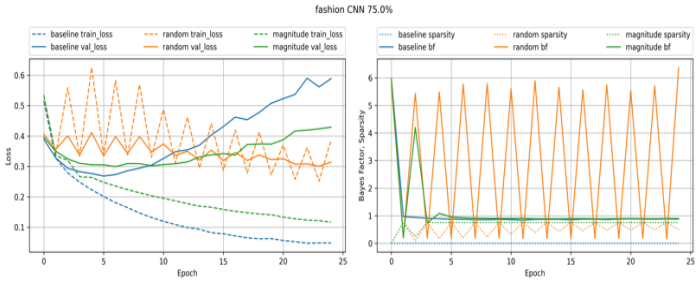

Figure 37: MNIST Fashion (CNN 75%) learning curve for the Bayesian pruning method.

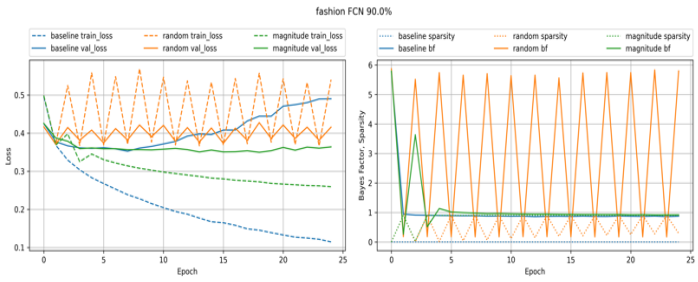

Figure 38: MNIST Fashion (FCN 90%) learning curve for the Bayesian pruning method.

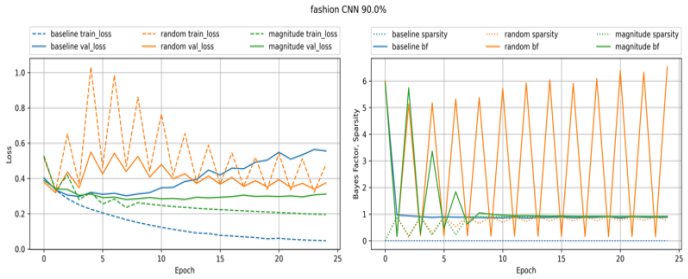

Figure 39: MNIST Fashion (CNN 90%) learning curve for the Bayesian pruning method.

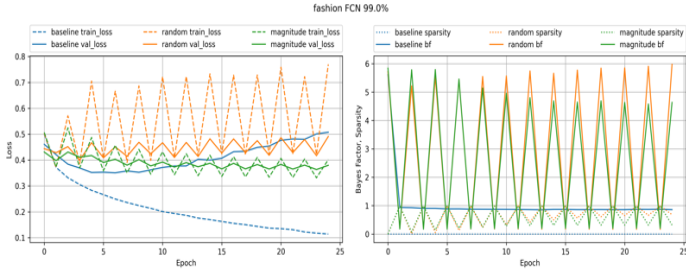

Figure 40: MNIST Fashion (FCN 99%) learning curve for the Bayesian pruning method.

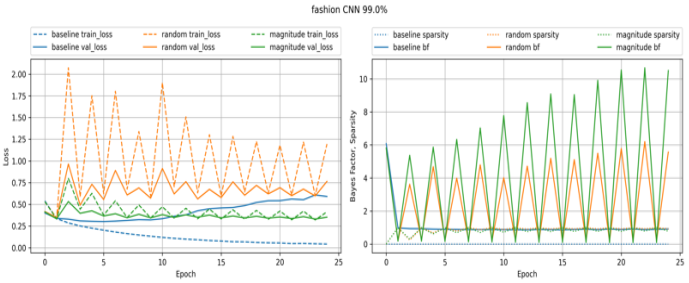

Figure 41: MNIST Fashion (CNN 99%) learning curve for the Bayesian pruning method.

APPENDIX A3. 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.

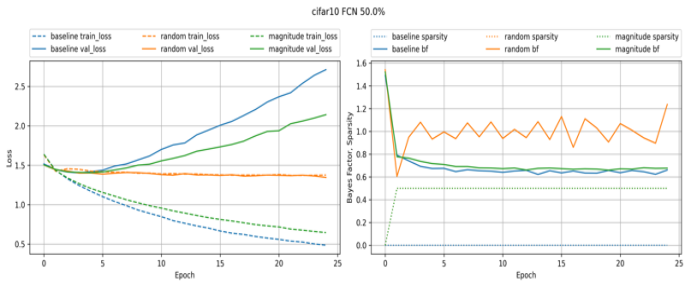

Figure 42: CIFAR-10 (FCN 25%) learning curve for the Bayesian pruning method.

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.

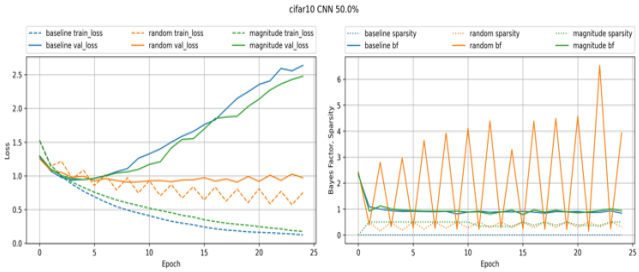

Figure 43: CIFAR-10 (CNN 25%) learning curve for the Bayesian pruning method.

Figure 43 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a CNN at 25\% sparsity. 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 FCN model at 25% sparsity.

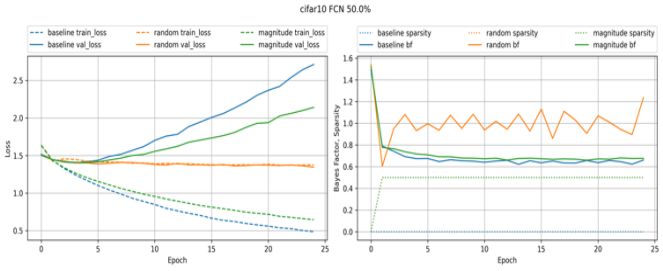

Figure 44: CIFAR-10 (FCN 50%) learning curve for the Bayesian pruning method.

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.

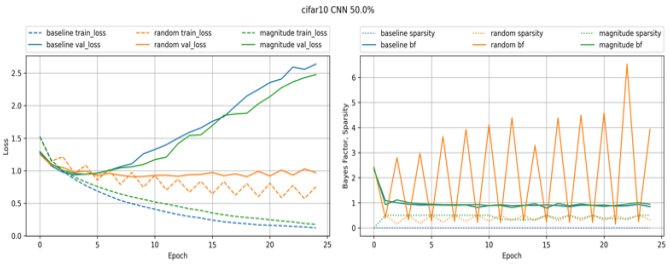

Figure 45: CIFAR-10 (CNN 50%) learning curve for the Bayesian pruning method.

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.

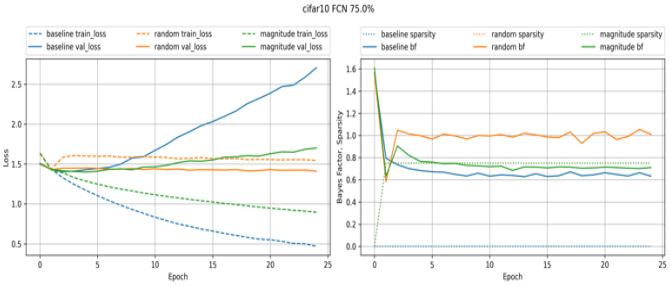

Figure 46: CIFAR-10 (FCN 75%) learning curve for the Bayesian pruning method.

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.

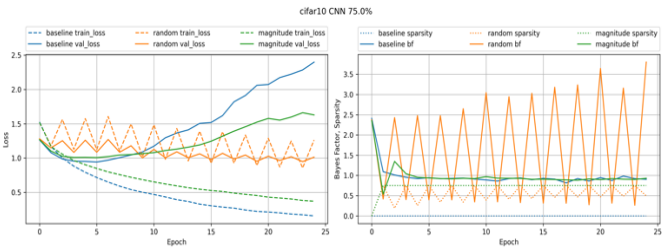

Figure 47: CIFAR-10 (CNN 75%) learning curve for the Bayesian pruning method.

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%.

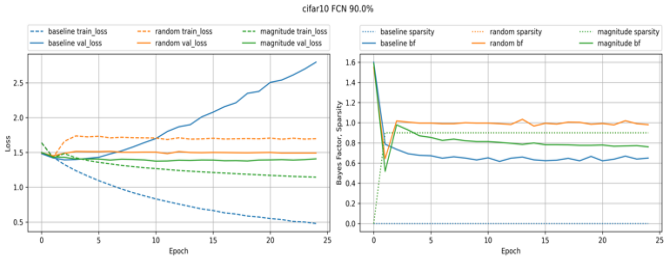

Figure 48: CIFAR-10 (FCN 90%) learning curve for the Bayesian pruning method.

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.

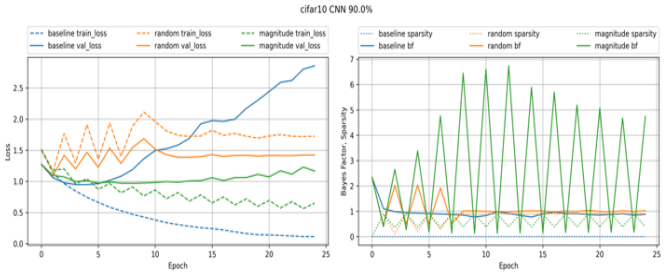

Figure 49: CIFAR-10 (CNN 90%) learning curve for the Bayesian pruning method.

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.

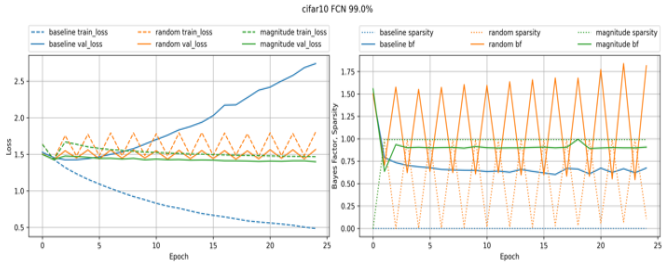

Figure 50: CIFAR-10 (FCN 99%) learning curve for the Bayesian pruning method.

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.

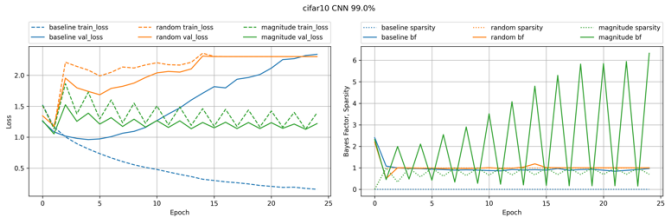

Figure 51: CIFAR-10 (CNN 99%) learning curve for the Bayesian pruning method.

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

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012.

[2] T. Brown *et al.*, "Language models are few-shot learners," *Adv Neural Inf Process Syst*, vol. 33, pp. 1877–1901, 2020.

[3] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Rethinking the Value of Network Pruning," *arXiv preprint arXiv:2104.06937*, 2021.

[4] D. Blalock, S. Kudugunta, and V. Shankar, "What is weight decay, and why does it work?," *arXiv preprint arXiv:2012.06106*, 2020.

[5] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *arXiv preprint arXiv:1510.00149*, 2015.

[6] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning Filters for Efficient ConvNets," *arXiv preprint arXiv:1608.08710*, 2017.

[7] Y. He, X. Zhang, and J. Sun, "Channel Pruning for Accelerating Very Deep Neural Networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1389–1398.

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[9] D. W. Blalock, J. J. G. Ortiz, J. Frankle, and J. V Guttag, "What is the State of Neural Network Pruning?," *CoRR*, vol. abs/2003.03033, 2020, [Online]. Available: https://arxiv.org/abs/2003.03033

[10] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," *arXiv preprint arXiv:1505.05424*, 2015.

[11] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, 2019, pp. 5234–5243.

[12] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression." 2017.

[13] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2015.

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998, doi: 10.1109/5.726791.

[15] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms." 2017.

[16] A. Krizhevsky, "Learning multiple layers of features from tiny images." 2009.

[17] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.