Pruning Neural Networks with Matrix Methods

Benjamin Ebanks [* 1] **Yajvan Ravan** [* 1]

Abstract

In the rapidly advancing field of deep learning, the architecture of neural networks has grown increasingly complex, featuring numerous layers and an extensive parameter set. This complexity enhances model performance by enabling the capture of more detailed data patterns, thereby increasing accuracy. However, it also introduces challenges such as parameter redundancy and reduced model interpretability. To address these issues, neural network pruning has emerged as an effective solution. Pruning simplifies these networks by eliminating superfluous weights, thereby streamlining the model without substantially diminishing its predictive accuracy. In this research, we explore three prominent approximation techniques for neural network pruning: low-rank approximation, principal component analysis (PCA), and randomized numerical linear algebra. Each method offers a distinct approach to reducing model complexity while maintaining performance. Our study aims to evaluate and compare the effectiveness of these techniques in the context of neural network optimization, providing insights into their practical applications and limitations.

1. Introduction

In this research, we delve into the application of neural network pruning across a spectrum of network architectures, namely a standard Multilayer Perceptron (MLP), and more complex models such as DenseNet (121 layers, 8 million parameters), ResNet (50 layers, 25 million parameters), and AlexNet (8 layers, 60 million parameters). These models, trained on MNIST and ImageNet datasets, provide a diverse basis for evaluating the impacts of pruning on different architectures that vary in depth, width, and types of layers. By employing three distinct approximation techniques—low-rank approximation, principal component analysis (PCA), and randomized numerical linear algebra—we aim to scrutinize how each method influences key performance metrics like accuracy and inference speed. Our hypothesis posits that larger networks, due to their parameter redundancy, are likely to benefit more substantially from pruning. This study is structured to not only experimentally assess the effectiveness of these pruning techniques but also to theoretically analyze the resultant performance enhancements, comparing these findings with empirical data to draw conclusions about the viability and efficacy of neural network pruning in modern AI applications.

2. Background

2.1. Linear Layer

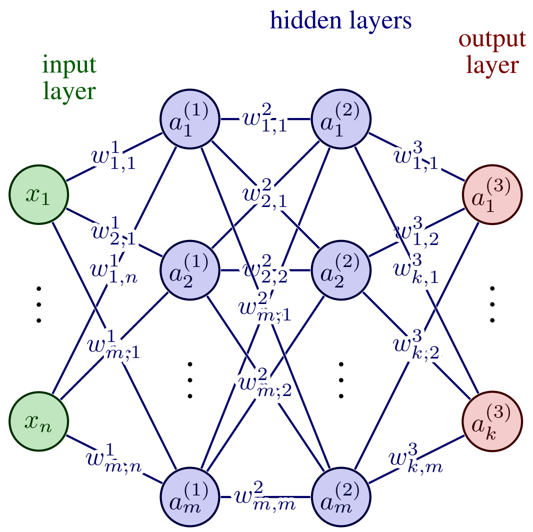


Figure 1. Illustration of multiple linear network layers

The architecture of a very basic fully connected linear network is depicted in Figure 1. The input is $1 \times n$ vector $\mathbf{x} = [x_1, \dots, x_n]^T$. The first layer, with m hidden units takes x and turns it into a^1, where $a^1 = [a_1^1, \dots, a_m^1]$ and $a_i^1 = \sum_{j=1}^{n} w_{i,j} x_j$, where $w_{a,b}$ are learnable weights. We can succinctly write this in matrix notation as

$$\mathbf{a}^1 = W^1 \mathbf{x}$$

where

$$W^1 = \begin{pmatrix} w_{1,1}^1 & \dots & w_{1,n}^1 \\ \vdots & \ddots & \\ w_{m,1}^1 & & w_{m,n}^1 \end{pmatrix}$$

Likewise, we can write $\mathbf{a}^2 = W^2 \mathbf{a}^1 = W^2 W^1 \mathbf{x}$ and so on.

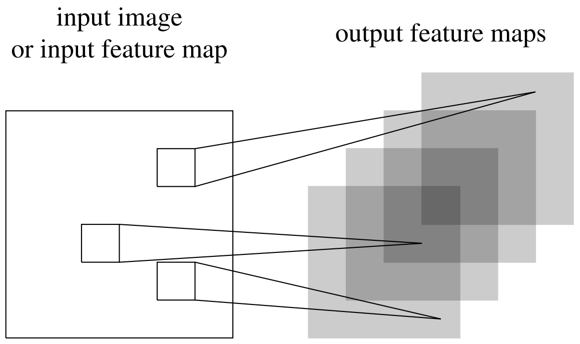


Figure 2. Illustration of a single convolutional layer. The input image (if $l = 1$) or a feature map of the previous layer is convolved by different filters to yield the output feature maps of layer l.

A full linear layer also adds biases and non-linear activation function f such that

$$\mathbf{a}^1 = f(W^1\mathbf{x} + b^1)$$

where $\mathbf{a}^1, b^1 \in \mathbb{R}^m$, $\mathbf{x} \in \mathbb{R}^n$ and $W_1 \in \mathbb{R}^{m\times n}$. Finally, stacking many of these linear layers gives us a complete fully connected neural network.

2.2. Convolutional Layer

For processing images, convolutional layers have proved to be quite effective. A simple convolutional layer across a single channel image is depicted in Figure 2. Typically an image $\mathbf{I}$ of size $(1 \times H_{in} \times W_{in})$, is convolved with a single kernel of size (k, k), with stride S (denoting the number of pixels to skip each operation) and padding P to produce a feature map $\mathbf{A}$ of size $(1 \times H_{out} \times W_{out})$ where

$$W_{out} = \frac{W_{in} - K + 2P}{S} + 1$$

$$H_{out} = \frac{H_{in} - K + 2P}{S} + 1$$

We denote the convolution as

$$\mathbf{A} = K * \mathbf{I}$$

For multichannel images $\mathbf{I}$ of $(C_{in} \times H_{in} \times W_{in})$, we use C_{in} kernels of size (k, k) for each channel to produce feature map $\mathbf{A}$ of size $(1 \times H_{out} \times W_{out})$ such that

$$\mathbf{A} = \sum_{i=1}^{C_{in}} K_i * \mathbf{I_i}$$

Finally, to produce multiple feature maps, we use $C_{out} \times C_{in}$ kernels of size (k, k) and simply stack the feature maps to get output $\mathbf{A}$ of size $(C_{out} \times H_{out} \times W_{out})$

$$\mathbf{A_j} = \sum_{i=1}^{C_{in}} K_{j,i} * \mathbf{I_i}$$

To complete the convolutional layer, we simply add a bias b of size $(C_{out} \times H_{out} \times Wout)$ and activation function f such that

$$\mathbf{A_j} = f\left(\sum_{i=1}^{C_{in}} K_{j,i} * \mathbf{I_i} + b_j\right)$$

2.3. Related Work

Work on neural network pruning has been done for many years. As early as 1993, Levin et. al. proposed principal components pruning. In that work, the authors used PCA to prune the weights of fully connected networks. They use a training set to compute the correlation matrix for the input to each layer, and compute the principal components of the output. They then delete nodes that do not increase the validation error (Levin et al., 1993).

Recent work has also focused on pruning CNNs. In (Garg et al., 2018), they used PCA on the feature maps generated by a deep convolutional network to identify the optimal number of channels for each layer of the network. In addition to identifying an optimal width, they also use this to identiyf an optimal depth, as the depth at which the width does not continue to increase. The authors then retrained the network with the new architecture and demonstrated minimal performance loss. However, this method involves retraining, which may not necessarily be feasible and will be computationally intensive.

In (Jaderberg et al., 2014) the authors perform low rank approximation of filter/weight banks both spatially (in the last two dimensions) and depthwise (in the first two dimensions). Likewise, in (Denton et al., 2014), the authors use 3-D and 4-D singular value decomposition to compute low-rank approximations of filters and show minimal increase in error. While they find a significant improvement in model size, they also note that practical improvements in speed are variable due to hardware and software specifics.

In this work, we propose and compare a similar array of methods to prune networks, inspired by the above and the algorithms from class.

3. Methodology

We investigated 3 different methods of Neural Network pruning, each serving a different purpose: Randomized Matrix Multiplication, PCA, and Low-Rank Approximation.

3.1. Randomized Linear Algebra

We use randomized matrix multiplication to speed up the computation of the matrix multiplication used in the networks. Namely, the randomized matrix multiplication algorithm is as follows below. It takes matrix $A \in \mathbb{R}^{m\times p}$ where $A = [A_1, \ldots, A_p]$ and $B \in \mathbb{R}^{p\times n}$ where $B =$

$[B_1, \dots, B_p]^T$ for $A_i, B_i \in \mathbb{R}^p$, and a number of samples c.

Algorithm 1 Randomized Matrix Multiplication

```
input $A \in \mathbb{R}^{m \times p}$, $B \in \mathbb{R}^{p \times n}$, $c \in \mathbb{N}$
 1: $p \leftarrow [A_1 B_1, \dots, A_p B_p]^T$
 2: Draw samples $(i_1, \dots, i_c)$ from $\{1, 2, \dots, p\}$ such that
    $P[i_k = j] \propto p_j$
 3: $ApproxAB \leftarrow 0$
 4: $k \leftarrow 1$
 5: while $k \leq c$ do
 6:     $ApproxAB \leftarrow ApproxAB + \frac{1}{c} * A_{i_k} B_{i_k}^T$
 7:     $k \leftarrow k + 1$
 8: end while
output $ApproxAB$
```

The theoretical complexity of this multiplication is $O(mnc)$, while the complexity of regular matrix multiplication is $O(mnp)$. Thus, if $c << p$, then theoretically, this algorithm will be much faster than regular matrix multiplication. Furthermore, as shown in class, $\mathbb{E}[ApproxAB] = AB$, thus we hypothesize that we are unlikely to lose much performance.

To apply this algorithm to a linear layer, such as the one from Section 2.1, we simply pass in W^1 and $\mathbf{x}$ to Algorithm 1 with a fixed c. We predicted that the inference time should increase linearly as a function of c.

3.2. PCA

We also investigated the use of PCA in pruning the size of the hidden layers of a neural network. Consider again the network from Section 2.1. The first layer performs the operation $\mathbf{a}^1 = f(W^1\mathbf{x}+b^1)$, and the second layer performs $\mathbf{a}^2 = f(W^2\mathbf{a}^1 + b^2)$ Note that the number of columns of W^1 is fixed by the number of input features, however, the number of rows is a choice of the model architecture. More hidden units allow the neural network to fit to more complex data, however, large networks may be redundant in the features they express. We can use PCA to reduce the dimension of the intermediate features, i.e. decrease the number of hidden units in the hidden layers. This reduces the dimensionality of W^1, thereby decreasing both the size and inference speed of the model.

The algorithm is displayed in Algorithm 2

Algorithm 2 PCA Fully Connected Network Pruning

```
input Trained Consecutive Layer Weights
    $W^1, W^2, \dots, W^k$, where $W^i \in \mathbb{R}^{m_{i-1} \times m_i}$, variance
    threshold $c \in [0, 1]$
Require: $m_{i-1} \geq m_i$ for all $i$
 1: $P = I$
 2: $i \leftarrow 1$
 3: while $i \leq k$ do
 4:     $w \leftarrow Mean(W_1^i, W_2^i, \dots, W_{m_i}^i)$
        { $W_j^i$ is the j-th column of $W^i$}
 5:     $W \leftarrow [W_1^i - w, W_2^i - w, \dots, W_{m_i}^i - w]$
 6:     $U_i \Sigma_i V_i^T =$ FullSVD($W$)
 7:     $TotalVar \leftarrow \sum_{j=1}^{m_i} \sigma_j^2$
 8:     while $j \leq m_i$ do
 9:         if $\sum_{t=1}^{j} \sigma_t^2 > c * TotalVar$ then
              Break
10:         end if
11:         $j \leftarrow j + 1$
12:     end while
13:     $P' \leftarrow [U_{i,1}, U_{i,2}, \dots U_{i,j}]$
        { $U_{i,j}$ is the j-th column of $U_i$}
14:     $W^i \leftarrow P'^T W^i P$
15:     $P \leftarrow P'$
16:     $i \leftarrow i + 1$
17: end while
output $W^1, W^2, \dots, W^k$
```

Let us dissect this using the linear network from $Section$ 2.1 Suppose that W^1 begins at $m_0 \times m_1$. In order to reduce the hidden units in the first layer, we take the trained network, and perform SVD to decompose W^1 and then reproject it onto a smaller latent space. Each column of W^1 encoded the relationship between an input unit, and all of the output units, and thus, we can treat each one as a data point with output units as features. We therefore wish to maintain as much variance between the input units with as minimal features as possible, and thus perform PCA on the columns of W.

We do this by computing the Singular Value Decomposition of W^1 and taking the principal components that give the desired fraction of the total variance. This gives us a projection matrix P' which essentially projects $\mathbf{a}^1$ onto a lower dimensional subspace. Therefore, we must use the inverse of P', which is P'^T, to also project the input space of W^2, as seen in line 12. Then, we simply repeat for each layer. Note that we require $m_{i-1} \geq m_i$, otherwise when performing PCA of W^i, we would have less data points than features.

We note that the size of W^i decreases drastically, by more than a factor of 4 in some of our experiments, and this reduces computation speed and model size drastically. Thus, we can use this method on a pruned network to achieve significant performance improvements.

3.3. Low-Rank Approximation

Thirdly, we used low-rank approximation to speed up the computation of the matrix multiplication. At a high level, consider some matrix $A \in \mathbb{R}^{n \times n}$. We can write A as CR

where $C \in \mathbb{R}^{n\times r}$ and $R \in \mathbb{R}^{r\times n}$ where $r = rank(A)$. Note that for $x \in \mathbb{R}^n$, computing Ax takes $O(n^2)$ time, while CRx takes $O(2rn)$ time, and likewise holds true for the memory needed for A. Thus, for $r << 2n$, we find significant speed improvement in multiplying by A.

The algorithm that we use to prune a trained network is shown below.

Algorithm 3 Low-Rank Fully Connected Network Pruning

```
input Trained Consecutive Layer Weights
      $W^1, W^2, \dots, W^k$, rank $r$
1: $i \leftarrow 1$
2: while $i \leq k$ do
3:     $U_i \Sigma_i V_i^T =$ FullSVD($W^i$)
4:     $W_1^i \leftarrow [U_{i,1}, U_{i,2}, \dots U_{i,r}]$
5:     $W_2^i \leftarrow [\sigma_1 V_{i,1}, \sigma_2 V_{i,2}, \dots \sigma_k V_{i,r}]^T$
       { $U_{i,j}$ is the j-th column of $U_i$}
6:     $i \leftarrow i + 1$
7: end while
output $W_1^1, W_2^1, W_1^2, W_2^2, \dots, W_1^k, W_2^k$
```

In essence, we split each layer into two linear layers, $W^i \to W_1^i, W_2^i$, such that both have rank r. Then, we can approximate $\mathbf{a}^i = f(W^i \mathbf{a}^{i-1} + b^i)$ as

$$\hat{\mathbf{a}^i} = f\left(W_2^i W_1^i \left(\hat{\mathbf{a}^{i-1}}\right) + b^i\right)$$

Note that the space and multiplication time of W decreases drastically if r is small, with the tradeoff that a very small r may not be able to accurately capture W.

3.4. CNNs

Each of the algorithms above applies readily to any linear or fully connected layer as depicted in Section 2.1. However, some challenges are presented when applying it to a convolutional layer, as the convolution operation is not a simple matrix multiplication.

However, as a linear operation, convolution can in fact be rewritten as matrix multiplication. Consider the generalized convolutional layer from Section 2.2

$$\mathbf{A_j} = f\left(\sum_{i=1}^{C_{in}} K_{j,i} * \mathbf{I_i} + b_j\right)$$

We can write the convolution operation as $K'I'$, where K' is the "flattened" version of K, i.e the shape of K' is $(C_{out} \times (C_{in} * k * k))$ and I' is a Toeplitz matrix constructed from I. This allows us to directly apply Algorithm 1 and Algorithm 3 to the K' matrix, by treating this convolutional layer as a linear layer with weight K'.

The application of Algorithm 2 is not as straightforward. One cannot naively apply this algorithm, since successive convolutional layers do not have the same input/output dimension of K' as linear layers do. Thus, instead of following this algorithm, we implement the algorithm from (Garg et al., 2018) and compare it to the methods above. Unlike the methods mentioned so far, this one requires retraining, which presents significant computational overhead.

3.5. Dataset

We used the MNIST dataset (Deng, 2012) to measure the performance of our fully connected linear models. Each image is $1 \times 28 \times 28$ and contains a single handwritten digit, and the task is to identify the digit. This essentially amounts to multiclass classification with 10 classes.

We used the ImageNet dataset (Deng et al., 2009) to measure the performance of our CNN networks. Each image is 3 channels and the task is to classify it into one of 1000 classes. We used the validation split, which contained 50,000 images.

3.6. Models

For fully connected networks (FCNs), we built two simple networks with architectures described in Table 1. One of them is very simple, while the other one is both deep and wide, for the purpose of measuring the impact of the algorithms described above. We trained both for 10 epochs on a random 90% split of MNIST before pruning.

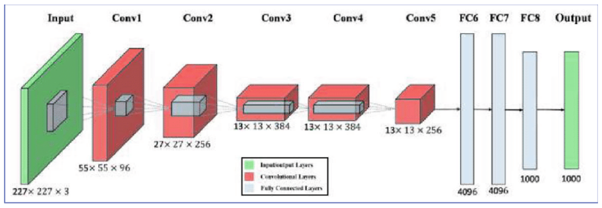
Network Type	In Units	Hidden Units	Out Units
Simple FCN	784	120, 84	10
Large FCN	784	128, 512, 2048, 8192, 2048, 512, 128	10

Table 1. Network Specification for our fully connected linear networks

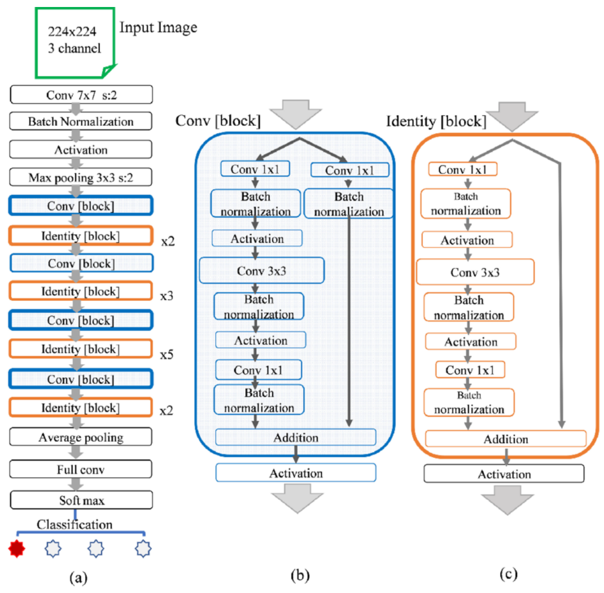
For CNNs, we use out-of-the-box pretrained networks. Namely, we use Alexnet (Krizhevsky et al., 2012), ResNet50 (He et al., 2015), and DenseNet121 (Huang et al., 2016). The network architectures are detailed in Figure 3.

3.7. Implementation and Evaluation

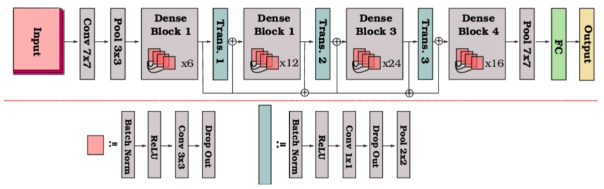
For each of the algorithms, we implemented baselines that such that the only difference between the baseline and the pruning algorithm was the specific operation itself. For Algorithm 1, we implemented a module that simply toggles between performing random matrix multiplication and regular matrix multiplicaiton. For Algorithm 2, we used the same architecture, and simply modified the weights in place. For Algorithm 3, we implement a linear module that takes two weights and one bias, to limit any differences between function overheads.



(a) AlexNet



(b) ResNet50



(c) DenseNet121

Figure 3. Network Architectures of AlexNet, ResNet50, DenseNet121. AlexNet contains over 60 million parameters, and 5 convolutional layers. ResNet50 contains 850,000 parameters and 4 convolutional layers with kernel size greater than 1. DenseNet contains 25 million parameters and 59 convolutional layers.

We have two primary evaluation metrics to help us quantify the improvement with our algorithms. Firstly, we measure accuracy (accuracy and standard deviation) on a random set of test images from ImageNet or MNIST. Secondly, we measure inference time (accuracy and standard deviation) across 20 runs.

For Algorithm 1 we measure both metrics as a function of the number of random samples we take. We expect that the average accuracy should increase with the number of random samples, the standard deviation should decrease, and the inference time should increase linearly.

For Algorithm 2 we measure both metrics as a function of the % of variance retained. We expect that the average accuracy should increase with the variance retained, the standard deviation should decrease, and the inference time should also increase although perhaps not linearly.

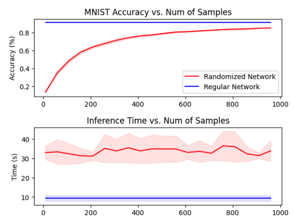
For Algorithm 3 we measure both metrics as a function of the rank. We expect that the average accuracy should increase with the rank, the standard deviation should decrease, and the inference time should also increase linearly.

4. Results

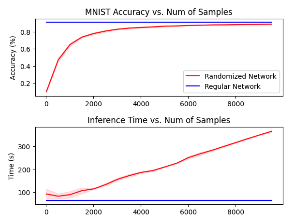
All of the graphs below depict the the accuracy of the modified network vs the regular network, and the bottom graphs show the inference time. Note that inference time does not include the time needed to compute the pruning, but simply the speed of the pruned model compared to the speed of the original model. Both curves contain mean accuracy and standard deviation (shaded) across 20 trials.

4.1. Fully Connected Networks

4.1.1. Randomized Multiplication



(a) Simple FCN



(b) Large FCN

Figure 4. The effect on accuracy and inference time of modifying the FCNs with **random matrix multiplication (Algorithm 1)** as a function of the **number of samples** used. The top graphs show the accuracy of the randomized network (red) vs the regular network(blue), and the bottom graphs show the inference time.

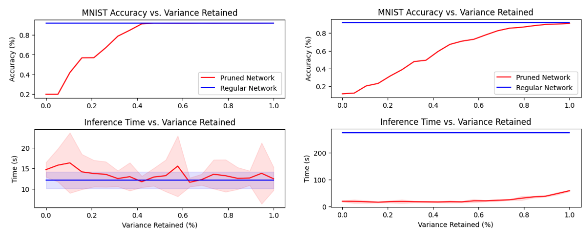
The results of modifying our Fully Connected Network architectures with randomized matrix multiplication are shown in Figure 4. We note two main trends. Firstly, the accuracy increases sublinearly with the number of random samples, and converges to the accuracy of the unmodified network. This follows what is expected since the expected value of random matrix multiplication is the same as the actual multiplication.

Secondly, an interesting trend is depicted in the inference time. For the simple FCN, we note that the inference time of the randomized network is the same as the unmodified network, while the inference time of the large network is significantly smaller, slowly increasing at the end. We can rationalize this trend, by noting that randomized multiplication involves an computational overhead from drawing the random samples that likely increases linearly with the number of samples. Furthermore, the number of multiplications performed increases linearly with the number of samples, so we expect the computation time to increase similarly. Thus, it seems that for the simple network, the computational bottleneck is not in the matrix multiplication, since the time does not increase with the number of samples, and the randomization worsens the time by adding overhead of drawing samples.

However, for the large network, using randomization drastically worsens the inference time. We note that the time of the randomized network seems to be dominated by overhead until we reach around 1000 samples, after which the time needed increases.The accuracy seems to not be affected. This is most likely because the large network contains many redundant units for this task, and thus the variance of each random multiplication is quite low.

Thus, randomized multiplication does not seem to improve the performance of our fully connected networks. The overhead of drawing samples and the number of samples needed to achieve good performance causes the inference time of the randomized network to be significantly larger than the unmodified network. Thus, while theoretically effective, this method does not work well in practice.

4.1.2. PCA



(a) Simple FCN (b) Large FCN

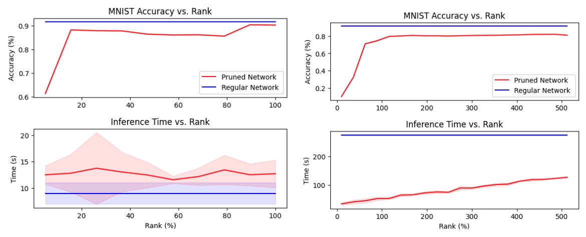
Figure 5. The effect on accuracy and inference time of pruning the FCNs with **PCA (Algorithm 2)** as a function of the **fraction of variance retained**. The top graphs show the accuracy of the randomized network (red) vs the regular network(blue), and the bottom graphs show the inference time

The results of pruning our fully connected networks with PCA is shown in Figure 5. We note that the accuracy increases as a function of variance retained, as expected. For the simple FCN, we do not require much variance to be retained to achieve equal performance, while for the large FCN, we need most of the variance retained. This is likely because in the small FCN, the classification task is more concentrated in each unit, while for the large FCN, the task is more evenly split among units. We also noted that in this case, 99% variance in the large FCN retained less than half the units, which corresponds with many of the units being redundant.

Furthermore, The inference time is drastically improved. We see a similar trend in the time for the simple network as with random multiplication. It seems that the computational time is not bottlenecked by the size of the multiplication, and therefore, PCA does not affect the time as much. However, for the large network, PCA leads to substantial improvements, decreasing the time by a factor of over 20. This directly follows from the reduction in dimensionality and multiplication size due to PCA pruning.

Thus, it seems that PCA pruning is quite effective for substantially improving model performance without a substantial loss in accuracy. The theoretical effectiveness is best realized for larger networks where multiplication is a bottleneck.

4.1.3. Low-Rank Approximation



(a) Simple FCN (b) Large FCN

Figure 6. The effect on accuracy and inference time of pruning the FCNs with the **low-rank approximation (Algorithm 3)** as a function of **rank** used. The top graphs show the accuracy of the randomized network (red) vs the regular network(blue), and the bottom graphs show the inference time

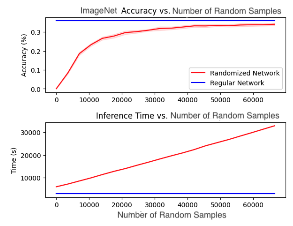
The results of pruning our network with a low-rank approximation is shown in Figure 6. We note that the trend in accuracy follows what is expected. As the rank increases, the approximation of each weight gets better, and therefore, the accuracy increases, converging to the unpruned network. However, we note an interesting trend, that the accuracy seems to increase in steps, rather than smoothly as in random multiplication or pca. This is due to the discrete sizes of each layer. For example, for rank between 84 and 120, our simple FCN can only be pruned in one layer, while for ranks less than 84, we can prune multiple layers. This leads to discontinuities in the accuracy curve, and is the reason that the accuracy large FCN does not converge under a rank of 1000, as the largest layer has 8192 units.

The inference time follows a similar trend as PCA. For the simple FCN, time is not limited by the multiplication, and as a result, we do not get a performance increase. However, for the large network, we do get substantial performance increase similar to PCA, and as expected, it increases linearly with rank.

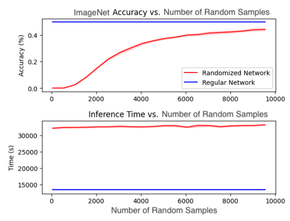
Thus, it seems that low-rank approximation pruning is quite effective for substantially improving model performance without a substantial loss in accuracy. The theoretical effectiveness is best realized for larger networks where multiplication is a bottleneck.

4.2. Convolutional Networks

4.2.1. Randomized Multiplication



(a) AlexNet



(b) ResNet50

Figure 7. The effect on ImageNet accuracy and inference time of modifying AlexNet and ResNet50 with **random matrix multiplication (Algorithm 1)** as a function of **the number of samples** used. The top graphs show the accuracy of the randomized network (red) vs the regular network(blue), and the bottom graphs show the inference time

The results of applying randomization to the CNN models is shown in Figure 7. We note that the trend in accuracy follows the same as from Section 4.1.1 as expected. However, we note that the number of samples needed is larger by two orders of magnitude. This is because the images are significantly larger, and constructing the Toeplitz matrix makes them even larger.

The trend in inference time is similar to Section 4.1.1. We note that for ResNet50, the inference time is unaffected by the number of samples drawn, but that it is much larger than the unmodified network. This is likely because ResNet50 only contains 4 convolutional layers, and they are smaller convolutions, and therefore are not likely the computational bottleneck. As mentioned before, drawing samples adds an overhead. AlexNet's inference time follows the trend of the large FCN from Section 4.1.1 for the same reason.

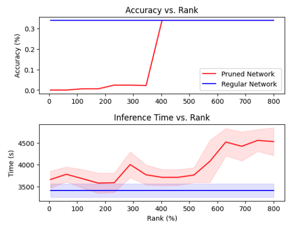
As DenseNet121 is more complex than AlexNet, we chose not to perform this experiment on DenseNet121 as our computational resources were constrained.

4.2.2. PCA

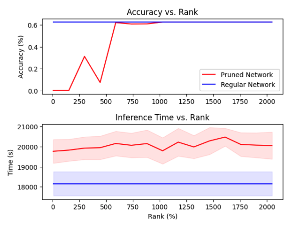
This method did not perform well in our initial experiments, and thus we chose not to perform extensive experiments with it. We trained AlexNet from scratch, performed the channel pruning with PCA mentioned in Section 3.4, and then retrained the pruned architecture with the same training hyperparameters. However, the original network attained an accuracy of 95% while the pruned architecture achieved an accuracy of 3%.

Thus, we decided that this method was not very effective without tweaking training hyperparameters and dealing with the difficulties of retraining.

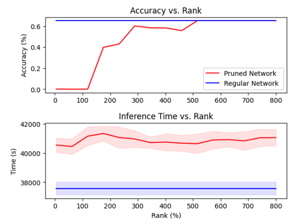
4.2.3. Low-Rank Approximation



(a) AlexNet



(b) ResNet50



(c) DenseNet121

Figure 8. The effect on ImageNet accuracy and inference time of pruning AlexNet and ResNet50 with the **low-rank approximation (Algorithm 3)** as a function of the **rank** used. The top graphs show the accuracy of the randomized network (red) vs the regular network(blue), and the bottom graphs show the inference time

The results of applying the low-rank approximation to all 3 CNNs is shown in Figure 8. We note that the accuracy follows the expected trend as from Section 4.1.3. It is interesting to note that the increase in accuracy as the rank increases has high variance for AlexNet and ResNet50, most likely due to limited number of convolutional layers, while the accuracy curve for DenseNet 121 is much smoother.

It is interesting to note that the inference time for all of the networks was substantially higher than the pruned networks. One reason may be that the time necessary for loading and unloading the GPU may be larger for pairs of matrices in the low-rank version compared to a single matrix. Another reason may be that computing one multiplication, due to hardware acceleration, may simply be faster than computing two successive lower-rank multiplications. However, it is unclear what the reason is and will require further investigation.

Thus, it seems that this method is not particularly useful for CNNs, due to computational overhead and practical differences in computing power.

5. Discussion

This study evaluated the efficacy of three neural network pruning methods across different architectures and datasets. Our findings revealed significant variations in the impact of these methods based on the complexity and size of the networks.

5.1. Effectiveness of Pruning Methods

Randomized Matrix Multiplication and **Low-Rank Approximation** were particularly effective in reducing model size and computational complexity. These methods often maintained high accuracy with significantly fewer computational resources, validating their theoretical benefits discussed earlier. However, the reduction in inference time was not as pronounced as the reduction in size, echoing the findings of Denton et al. (Denton et al., 2014), who noted that practical gains in speed are not always commensurate with theoretical improvements due to hardware and software constraints.

PCA Pruning showed mixed results. While effective in reducing the dimensionality and enhancing the inference speed of large fully connected networks, its application to convolutional neural networks (CNNs) was less successful without additional hyperparameter tuning. This aligns with Garg et al. (Garg et al., 2018), who achieved better results with comprehensive model retraining—a step we did not undertake due to our focus on methods that do not require extensive retraining.

5.2. Network Suitability

Our results suggest that large and complex models, such as the Large FCN and DenseNet121, benefit more from pruning methods compared to simpler models. These networks, originally designed with high redundancy to capture complex patterns, can tolerate significant reductions in complexity without substantial performance loss. In contrast, simpler models or those with fewer layers, such as the Simple FCN and ResNet50, exhibited minimal benefits from pruning, likely due to their already optimized structure for the tasks.

5.3. Future Directions

Given the observed discrepancies between theoretical and practical outcomes, future research should explore several avenues:

- **Improvement of PCA Application:** Investigating methods to compute spatial or depthwise separability in PCA without retraining could make PCA more applicable to CNNs.
- **Optimization of Pruning Techniques:** Refining the application of randomized and low-rank methods to operate directly on the convolution operations without unfolding could preserve spatial hierarchies and reduce computational overhead.
- **Hardware Optimizations:** Developing hardware that specifically enhances the efficiency of operations like randomized matrix multiplication could bridge the gap between theoretical and practical performance improvements.

6. Conclusion

In conclusion, while pruning techniques hold promise for reducing neural network complexity and enhancing computational efficiency, their application must be carefully tailored to the network architecture and the specific requirements of the task. Future work should continue to refine these methods and explore new ways to optimize network pruning for emerging AI applications.

References

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Denton, E., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. *CoRR*, abs/1404.0736, 2014. URL `http://arxiv.org/abs/1404.0736`.

Garg, I., Panda, P., and Roy, K. A low effort approach to structured CNN design using PCA. *CoRR*, abs/1812.06224, 2018. URL `http://arxiv.org/abs/1812.06224`.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

Huang, G., Liu, Z., and Weinberger, K. Q. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. URL `http://arxiv.org/abs/1608.06993`.

Jaderberg, M., Vedaldi, A., and Zisserman, A. Speeding up convolutional neural networks with low rank expansions. *CoRR*, abs/1405.3866, 2014. URL `http://arxiv.org/abs/1405.3866`.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K. (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Levin, A., Leen, T., and Moody, J. Fast pruning using principal components. In Cowan, J., Tesauro, G., and Alspector, J. (eds.), *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann, 1993. URL https://proceedings.neurips.cc/paper_files/paper/1993/file/872488f88d1b2db54d55bc8bba2fad1b-Paper.pdf.