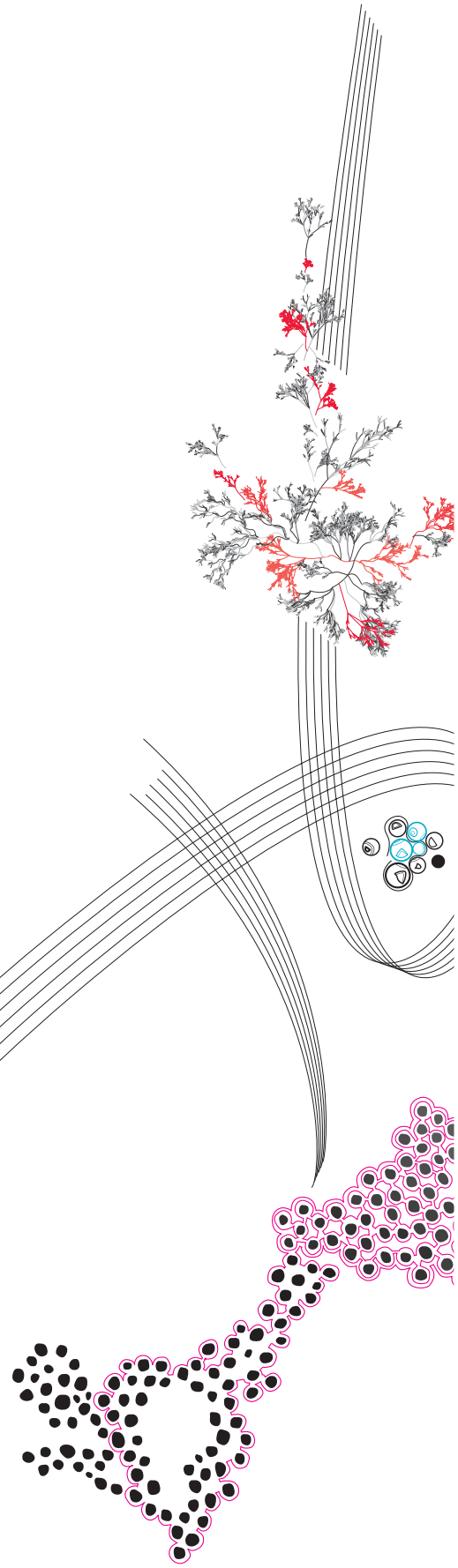


MSc Computer Science
Final Project

SPaS: Sparse Parameterized Shortcut Connections for Dynamic Sparse-to-Sparse Training

Mauk W. Muller

Supervisors:
Decebal C. Mocanu
Mannes Poel
Gwenn Englebienne
Bram Grooten
Aleksandra Nowak

September, 2023

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science,
University of Twente

UNIVERSITY OF TWENTE.

SPaS: Sparse Parameterized Shortcut Connections for Dynamic Sparse-to-Sparse Training

Mauk W. Muller
University of Twente
Enschede, Netherlands
m.w.muller@student.utwente.nl

Abstract

Sparse Neural Networks (SNNs) have proven themselves to be an effective method for both the reduction of computational costs and the improvement of performance in Neural Networks (NNs). Sparse-to-sparse (STS) training methods have managed to supersede pruning methods by allowing an optimal topology to be found during training. Dynamic Sparse Training (DST) methods like SET have managed to quadratically reduce the number of parameters, with no decrease in accuracy while shortcut connections have demonstrated their ability to enable deeper and better-performing networks such as in ResNets and DenseNets. Recent works have investigated sparsity for shortcut connections and have demonstrated both improvements in performance and reduction in parameter counts over purely sparse sequential models. In this thesis, we introduce **S**parse **Pa**rameterized **S**hortcut Connections (SPaS), which combines the principles of sparsity and shortcut connections, and a training schema, SPaSET, that enables SPaS networks to be trained dynamically. We apply SPaS to two typical deep learning architectures, i.e. Multi Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs), and evaluate these on computer vision and numerical classification tasks. SPaS improves information flow within networks and enables feature reuse. SPaS enables MLPs to be compressed over 25x without significantly compromising performance, up from compression rates of 5x on plain MLPs. In CNNs we find that SPaS improves performance over high-density regions for similar computational costs, providing up to 5% and 2% increases over the validation set compared to plain CNNs and DenseNet respectively.

1 Introduction

Artificial Neural Networks (ANNs) have managed to become one of the most successful machine learning methods by achieving state-of-the-art results over a broad range of application domains. Nevertheless, ANNs can grow to a level of parameters that exceed the capacity of commodity hardware, reaching as high as 175 billion parameters in GPT-3 [5] or even 1 trillion as speculated in GPT-4 [3]. Consequently, there is a need to develop methods to compress these state-of-the-art ANN models, while retaining their optimal performance. In this context, sparsification presents a promising approach to parameter reduction.

In 1989, it was demonstrated that densely trained ANNs can be pruned to a significant extent without compromising their performance [22]. In fact, it was observed that pruning ANNs can actually enhance their performance while reducing the amount parameters up to a factor of four. This spawned a class of methods called dense-to-sparse (DTS) training, which entails iteratively training and pruning a network. In 2018, the Lottery Ticket Hypothesis was introduced by Frankle et al. [10], which states that randomly-initialized networks contain subnetworks (*winning tickets*) that, when trained in isolation, can compete with the performance of the original network.

Despite the efficacy of DTS training, it presents certain limitations. Specifically, the largest trainable sparse model is confined by the size of the largest trainable dense model, limiting the potential size of sparse models. Additionally, while the final sparse model results in computational gains at the inferencing stage, little to no computational gains are made at the training stage. To address these limitations, sparse-to-sparse (STS) training was proposed by Mocanu et al. [31]. In STS, contrary to DTS, networks are already sparse at initialization. These initial methods, however, are based on a static topology, which greatly limits the network's expressibility, as determining the optimal topology before training is not trivial. As a result, static STS methods exhibit weaker performance in comparison to their dense counterparts. To alleviate this problem, Sparse Evolutionary Training (SET) was introduced by Mocanu et al. [30, 32], which enables the discovery of an optimal sparse connectivity pattern during the training (In-Time Over-Parameterization [28]). SET employs a random regrow strategy and prunes weights by magnitude, similar to synaptic shrinking which occurs in biological brains [6, 8]. The incorporation of dynamically changing topologies has later also been studied by [24–26]. In particular, Dettmers and Zettlemoyer [7] and Evci et al. [9] introduce gradient regrow strategies, which enhance convergence over random regrow strategies.

As networks become highly sparse they can easily suffer from layer-collapse, whereby an entire layer is prematurely pruned rendering a network untrainable [43], and impeded back propagation. A commonly used method for improving gradient flow in networks is to introduce shortcut connections. A shortcut connection links non-consecutive layers to one another. Shortcut connections are often employed

in an information-passing capacity, whereby residual information is fed forward without mutation. Such shortcut connections have enabled architectures like ResNets [13] and DenseNets [16] to reach layer-depths of up to 1000 layers. [18] and [41] investigate the effects of shortcut connections in finding sparse masks and demonstrate improved performance among high sparsity regimes and improved connectivity patterns, these methods additionally alleviate the risk of layer-collapse. While previous works have investigated sparsity and shortcut connections, there is a lack of attention in the literature on both dynamic sparse masks for shortcut connections and sparse shortcut connections with increased depth.

In this paper, we introduce **S**parse **Pa**rameterized **S**hortcut Connections: SPaS[1]. SPaS combines the advantages of sparsity and shortcut connections. Specifically, we apply SPaS to Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs), denoted by SPaS-MLP and SPaS-CNN respectively. We additionally introduce an evolutionary training algorithm, SPaSET, which allows us to find optimal sparse masks for both sequential and shortcut connections during training. We aim to answer the following research questions:

- **RQ1.** *What role do sparse shortcut connections play in the performance of ANNs?*
- **RQ2.** *Can we use the principles of Dynamic Sparse Training to find optimal sparse masks for shortcut connections?*
- **RQ3.** *To what extent can SPaS allow us to create even sparser networks?*

In this thesis, we answer the aforementioned research questions and aim to contribute the following:

- We introduce the SPaS architecture and its application to MLPs and CNNs. We show that SPaS can improve performance in MLPs and CNNs over their non-SPaS counterparts, providing up to 5% improvement over the validation data of various datasets.
- We introduce SPaSET, an evolutionary training algorithm enabling DST in SPaS networks, allowing sparse masks for both sequential and shortcut layers to be found during training. SPaSET enables SPaS-MLPs to maintain performance longer by improving gradient flow in high-sparsity regimes. Moreover, SPaSET improves performance in SPaS-CNNs among low-sparsity regimes. SPaSET provides up to 20% increased performance over SPaS networks with static sparse masks on CIFAR-10 and CIFAR-100 in high-sparsity regimes.
- We find that SPaS enables MLPs to be compressed up to 25x and 75x on numerical classification and computer vision tasks respectively. This is up from compression rates of 5x on plain MLPs while remaining competitive with plain dense and sparse MLPs.

[1]Our implementation of SPaS will be available in the near future on https://github.com/MaukWM/SPaS

1.1 Outline

In Section 2 we go through the various related works. We begin by elaborating on the concept of shortcut connections, after which we go in-depth on STS training methods. Finally, we take a look at other sparse architectures which incorporate shortcut connections. Section 3 discusses the necessary mathematical background. We first give the mathematical description of an MLP, after which we give a modified definition for sparse MLPs and introduce notation for shortcut connections. In Section 4 we expound on our proposed architecture by building on top of the mathematical basis from Section 3. We first introduce the concept of SPaS for MLP and CNN architectures, after which topological initialization and the evolutionary rules are explained. In Section 5 we explain the experimental setup that was used for the numerical results which are presented in Section 6. We explain our implementation, evaluation metrics, and training setup. In Section 6 the numerical results of the experiments are shown and explained. In Section 7 we present our conclusions drawn from the numerical results and also provide a discussion on the approach and its limitations. In Section 8 we discuss possible research directions which build on this thesis' work.

2 Related Work

In this section, we first introduce work performed in the area of shortcut connections, after which we go into sparse-to-sparse training methods. Finally, we look at related work that combines sparse-to-sparse training methods and shortcut connections.

2.1 Shortcut connections

Shortcut connections have been studied for a long time. One of the earliest practices in the training of MLPs is to add a linear connection from the network input to the output [37, 46]. Shortcut connections are often employed to tackle the exploding/vanishing gradient problem. For example, in a paper by Lee et al. (2015) [23], intermediate layers have been connected to auxiliary classifiers in convolutional networks. Another example is the GoogLeNet architecture with its Inception modules, which works with a concatenation of outputs on layers [42].

It is important to make a clear distinction between two types of shortcut connections, namely, *residual* and *non-residual* shortcut connections. The term residual, according to the Cambridge dictionary, is defined as *"remaining after most of something is gone"*. A residual shortcut connection

is a connection whereby the output of a previous layer (the remainder) is fed forward to subsequent layers after the original output has already been mutated by sequential layers ("after something has gone"), these connections are inherently parameter-free, which stands in contrast to SPaS, where our shortcut connections are always parameterized. More explicitly, these layers can be seen as having the identity function applied to them, this leads to residual connections also being called *identity* connections at times (or *identity* shortcuts as in ResNet [13]).

ResNets only allow residual connections up to one layer ahead. DenseNets, introduced by Huang et al. in 2017 [16], attempts to maximize information flow between layers by connecting all layers directly with one another: each layer connects to every subsequent layer in a residual, feed-forward fashion. In contrast to SPaS, shortcut connections in DenseNets are always residual. Similarly to SPaS-CNN, DenseNet always places such layers in blocks interspaced by transition layers. One important distinction, however, is that SPaS-CNN combines feature-maps through summation as opposed to concatenation. From this distinction, another difference emerges: concatenation enables DenseNets to naturally grow its feature-map size within blocks, while summation results in the SPaS-CNN architecture keeping feature-map size fixed within blocks. Due to this fact, DenseNet blocks warrant transition layers to apply a feature-map size reduction, while SPaS-CNN blocks warrant transition layers to perform feature-map size growth.

When a connection is *non-residual* it means that the connection does not simply pass residual information as is: the residual information is mutated with parameters, which is why we can also refer to *non-residual* connections as *parameterized* connections. Highway Networks by Srivastava et al. (2015) [39, 40] enable both *residual* and *non-residual* connections through the use of gates (as inspired by LSTMs [15]), these gates are data-dependent and have parameters. When the gates are "closed", layers in highway networks represent *non-residual* functions, when "open" they represent *residual connections*. While Highway Networks have the ability to learn which layers should be (non-)residual, in SPaS, all layers are consistently non-residual.

Shortcut connections are often referred to as skip connections, they both represent the same concept, and the terminology is used interchangeably in this thesis.

2.2 Sparse-to-sparse training

Sparse-to-sparse (STS) training was initially introduced by Mocanu et al. (2016) [31] by sparsifying a Restricted Boltzmann machine and keeping its sparsity static. Static STS training has also been investigated on other NN structures [2, 35]. A disadvantage of static STS training is the fact that network topologies must be designed before training. These topologies are often unable to model the data distribution well and can have connectivity patterns that impede backpropagation, leading to a worse performance when compared to dense counterparts.

Alternatively, one can find a suitable topology for static STS training by first training a dense model and pruning it, as in the Lottery Ticket Hypothesis [10]. This, however, defeats the purpose of STS training as it requires a dense model to be trained first.

In dynamic STS training, it is not a requirement that the initial network topology is able to model the data distribution, as dynamic STS training allows for the network topology to be mutated during training. There are two classes of methods when considering dynamic STS training.

The first are methods based on *random regrowth*. Random regrowth relies on selectively pruning and randomly regrowing connections in a network. Over time, as more evolutionary steps are executed, the network performs *in-time over-parameterization* [28] and will be able to find a topology suitable for modeling the data distribution.

One of the first random regrow methods introduced was *Sparse evolutionary training* (SET) in a paper by Mocanu et al. (2018) [32]. This method initializes a network with a sparse topology. After initializing the network, SET performs standard training procedures and weight updates. At the end of each epoch, however, an evolutionary step is done. First, the network is pruned by removing a fraction ξ of the smallest magnitude weights. After the pruning step weights are randomly added in the same amount as the ones previously removed. The sparsity of the network remains static throughout training.

SET has been shown to efficiently replace MLP's fully connected layers with sparse layers, resulting in a quadratic reduction of the number of parameters in MLP layers, at no expense of accuracy.

Other random regrowth methods include Dynamic Sparse Reparameterization [33] for CNNs with dynamic sparsity levels per layer. Selfish RNN (2021) [27] is a specialized training method for RNNs that outperforms its dense counterparts.

Furthermore, there are methods based on *gradient regrowth*. Dettmers and Zettlemoyer [7] and Evci et al. [9] introduce the idea of using momentum and gradient information for selecting weights for regrowth. By identifying weights that would reduce the error efficiently, pruned weights can be redistributed more effectively than in random regrowth schemes. One downside, however, is the fact that these methods are not as scalable as random regrowth because of the necessity to calculate gradients for non-existent weights, leaving overhead equalling the scaling of a dense network. Jayakumar et al. [19] introduces Top-KAST which improves RigL by only considering a subset of gradients from non-existent connections.

These STS methods are all designed for fully sequential networks, where SPaSET is an evolutionary algorithm designed to support networks with both sequential and shortcut connections. SPaSET additionally gives control over the

ratio R of sequential and shortcut connections during training through two different flavors of random regrowth that either fixes or unfixes the ratio R.

2.3 Sparse networks with shortcut connections

There has been some work on training sparse networks with shortcut connections. Jaiswal et al. [18] proposes a sparse mask training toolkit (ToST), whereby the training of a static sparse mask is explored by injecting the sparse mask with "Ghost" Skip Connections (GSk) that are gradually phased out during training, resulting in the final network being unchanged from its original architecture. SPaS differs from ToST by keeping its skip connections in the final architecture. Additionally, GSk are residual and their connectivity pattern does not evolve throughout training, whereas this is the case in SPaS. Skip connections in SPaS are also able to connect to deeper subsequent layers than ToST.

This paper also shows that ReLU [1] can result in a very sparse layer activation that can block gradient flow. To mitigate this issue they temporally replace ReLU with Swish [36] and Mish [29] through its "Ghost" Soft Neurons, which leads to more stable gradient flow [45].

Subramaniam and Sharma [41] propose N2NSkip which, similarly to SPaS, adds sparse parameterized shortcut connections to CNNs. These N2NSkip connections are pruned prior to training and their sparse mask is kept static throughout training, whereas SPaSET enables SPaS networks to evolve the sparse masks of shortcut connections. N2NSkip maintains an equal amount of sequential and shortcut connections while SPaS introduces a novel ratio parameter R that allows SPaS networks to be initialized with a specified ratio of sequential-shortcut connections and enables this ratio to evolve throughout training.

N2NSkip shows improvements over ResNet50 and VGG19 [38] architectures with 23M and 143M parameters respectively. They find, especially at high compression rates of 20x and 50x, that N2NSkip connections consistently produce a significantly lower test error on both CIFAR-10 and CIFAR-100 when compared to counterparts with residual skip connections, demonstrating the effectiveness of parameterizing conventionally residual skip connections.

They also perform a connectivity analysis by calculating the heat diffusion signature of the networks. They find that for both pruning methods, ResNet50 and VGG19 with N2NSkip connections have superior overall connectivity when compared to the baseline models.

3 Background

In this section we state the mathematical definition of a dense network, also known as a Multilayer Perceptron (MLP) [44], its sparse counterpart, and shortcut connections. These definitions are built upon further in Section 4 where the proposed architecture is defined.

3.1 Dense MLP definition

Given a dataset of N data points, denoted by $D = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}$, let a dense MLP (D-MLP) be defined as:

$$\hat{\mathbf{y}} = f(\mathbf{x}; \theta) \tag{1}$$

Where θ is the set of parameters (weights and biases) associated with the D-MLP. Each layer in the network parameters θ can be decomposed into $\mathbf{W}_l$ and $\mathbf{b}_l$ which are dense matrices:

$$\mathbf{W}_l \in \mathbb{R}^{n^{l-1} \times n^l}, \mathbf{b}_l \in \mathbb{R}^{n^l} \tag{2}$$

Where n^{l-1} and n^l represent the amount of neurons in layers $l-1$ and l respectively.

A sequential k-layered network is a composition of multivariate functions, defined as:

$$f = f_k \circ f_{k-1} \circ ... \circ f_2 \circ f_1 \tag{3}$$

Where each function f_l describes the forward pass of layer l, defined as:

$$f_l(x) = a(\mathbf{x}_{(l-1)}\mathbf{W}_l + \mathbf{b}_l) \tag{4}$$

Where a is the activation function and $\mathbf{x}_{(l-1)}$ is the output of the previous layer.

When training an MLP we seek to optimize θ through the minimization of a loss function L, for example, Mean Squared Error (MSE):

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)})^2 \tag{5}$$

$$\theta = \underset{\theta}{argmin}\ [L(\mathbf{y}, f(\mathbf{x}; \theta))] \tag{6}$$

3.2 Sparse MLP definition

For a sparse MLP (S-MLP), the goal is to reparameterize the dense θ into a sparse θ_S, where S represents the sparsity of the network. A S-MLP can then be described as:

$$\hat{\mathbf{y}} = f(\mathbf{x}; \theta_S) \tag{7}$$

This definition can then be used to describe both D-MLPs ($S = 0$) and S-MLPs ($0 < S < 1$).

The sparsity of the network is calculated as follows:

$$S = 1 - \frac{||\theta||_0}{||\theta||}, \tag{8}$$

where $||\theta||_0$ refers to the l_0 norm of θ (count of non-zero parameters) and $||\theta||$ refers to the count of possible parameters.

3.3 Shortcut connections

Many breakthroughs in deep learning have been attributed to the use of deeper networks [13, 17, 38]. One major obstacle in deep networks however is the vanishing/exploding gradients problem [4, 11]. This problem can be addressed by explicitly letting layers fit a residual mapping [13, 16]. In an MLP, we can describe a forward pass f_l with a residual connection as:

$$f_l(\mathbf{x}) = a(\mathbf{x}_{l-1}\mathbf{W}_l + \mathbf{b}_l) + \mathbf{x}_{l-1}, \tag{9}$$

where the addition of $\mathbf{x}_{l-1}$ outside the activation function a entails the passing of the output from the previous layer (the residue) to the next.

In a CNN, more specifically a ResNet, we can describe a layer with a residual mapping as:

$$\mathbf{x}_l = H_l(\mathbf{x}_{l-1}) + \mathbf{x}_{l-1}, \tag{10}$$

where $\mathbf{x}_l$ is the output of the l^{th} layer and $H_l(\cdot)$ is a non-linear transformation.

Departing from ResNets where the residual connections only skip over 1 layer, we define a layer in a DenseNet as:

$$\mathbf{x}_l = H_l([\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_{l-1}]), \tag{11}$$

where $[\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_{l-1}]$ refers to the concatenation of feature-maps produced in layers $0, ..., l-1$.

Moving to skip connections that permit parameterization, a layer in a Highway Network is defined as:

$$\mathbf{x}_l = H_l(\mathbf{x}_{l-1}) \cdot T_l(\mathbf{x}_{l-1}) + \mathbf{x}_{l-1} \cdot C_l(\mathbf{x}_{l-1}), \tag{12}$$

where $T_l(\cdot)$ and $C_l(\cdot)$ are the transform gate and carry gate respectively. The transform gate determines to what degree the input will be mutated and the carry gate determines to what degree the previous output must be carried over.

4 Methodology

In this section, we propose the architecture of **S**parse **Pa**rameterized **S**hortcut Connections (SPaS), which is built upon the mathematical foundation from Section 3. ToST [18] and N2NSkip [41] investigate sparse skip connections solely in the context of CNN architectures while SPaS is a general concept that can work in synergy with various ANN architectures. In this thesis, we demonstrate the effectiveness of SPaS in combination with two widely used neural network types, i.e., MLPs and CNNs, by introducing SPaS-MLPs and SPaS-CNNs. We introduce the SPaS-MLP and SPaS-CNN architecture, after which we explain the initialization of sparse masks. Also departing from N2NSkip and ToST, we introduce an evolutionary algorithm, SPaSET, that enables the sparse mask of SPaS to evolve throughout training, fully enabling DST in SPaS networks.

4.1 Sparse Parameterized Skip Connections

In this thesis, we propose **S**parse **Pa**rameterized **S**hortcut Connections (SPaS). In a SPaS network, we make the distinction between sequential and skip layers, wherein sequential layers are always connected to their neighbouring subsequent layer and, departing from N2NSkip and ToST, skip layers can be connected to *any* non-neighbouring subsequent layer as opposed to skipping up to 1 layer. Additionally, as the name implies, the connections are *parameterized*, meaning each skip layer has its own associated set of weights and biases that can mutate residual information from previous layers and regulate gradient flow by learning the relative importance of each gradient signal. Additionally, this parameterization gives SPaSET a simple metric for determining which shortcut connections to prune as residual connections don't have a magnitude with which to determine their relative importance.

Each SPaS network has an associated ratio of sequential and skip connections, which we define as R.

$$R = \frac{||\theta_{D=1}||_0}{||\theta||_0} \tag{13}$$

Where $||\theta_{D=1}||_0$ refers to the number of active sequential parameters and $||\theta||_0$ to the number of *all* active parameters. Here we introduce a new notation θ_D, where D refers to the amount of layers that a parameter traverses. We use this notation to concisely refer to the set of parameters in sequential and shortcut layers. We use $\theta_{D=1}$ to refer to the set of parameters over all sequential layers and $\theta_{D>1}$ to refer to the set of parameters over all shortcut layers.

A SPaS network where $R = 1$ represents a network built exclusively using sequential connections, making it identical to its non-SPaS counterpart: a SPaS-CNN and SPaS-MLP with $R = 1$ are functionally identical to their S-CNN and S-MLP counterparts respectively. A SPaS network with $R = 0$ represents a network built exclusively with skip connections.

While the SPaS architecture offers the possibility of networks with $R = 0$, such a network is accompanied by unnecessary overhead. The layer following the input layer of a network will never be connected to the input and, in a similar vein, the penultimate layer will remain disconnected from the output layer. This results in computational dead ends, both in forward and backpropagation.

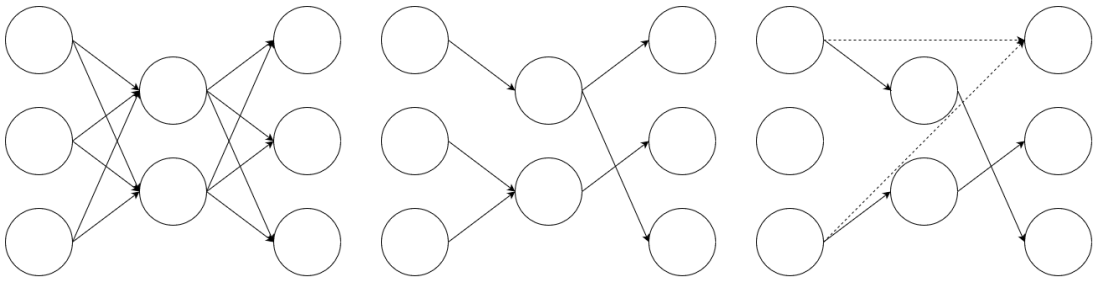
By allowing R to vary, we depart from N2NSkip, where all networks have fixed ratios of $R = 0.5$.

4.1.1 SPaS-MLP

To move from S-MLP to SPaS-MLP, we need to incorporate the newly defined ratio parameter R. We extend the S-MLP definition from Equation 7 and we define a SPaS-MLP as follows:

$$\hat{\mathbf{y}} = f(\mathbf{x}; \theta_{S,R}), \tag{14}$$

we can use this extended definition to represent both S-MLPs and SPaS-MLPs. We can describe an S-MLP as:



(a) $f(\mathbf{x};\theta_{S=1,R=1})$ **(b)** $f(\mathbf{x};\theta_{S=0.5,R=1})$ **(c)** $f(\mathbf{x};\theta_{S=0.5,R=0.66})$

Figure 1. *Example network architectures for various types of MLPs with their associated mathematical definitions. The solid lines are sequential connections and the dashed lines are shortcut connections.* ***(a)*** *shows a fully dense MLP (D-MLP).* ***(b)*** *shows a 50% sparse MLP (S-MLP).* ***(c)*** *shows a 50% sparse parameterized shortcut MLP (SPaS-MLP), where 66% of the connections are sequential.*

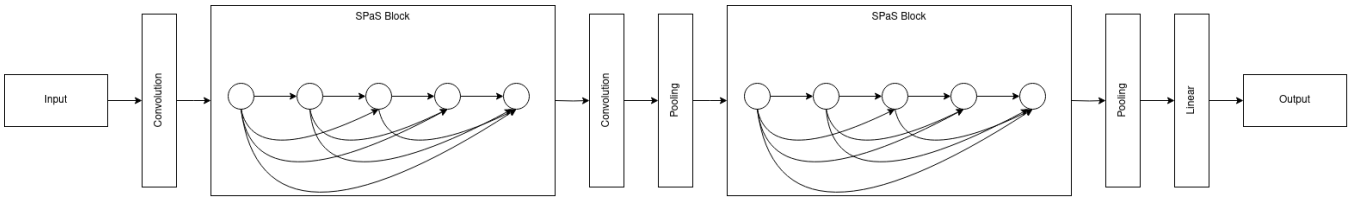


Figure 2. *A SPaS-CNN consisting of two 4-layer SPaS blocks. Within a SPaS block, each forward connection consists of its own set of weights and biases. Within a SPaS block, the results from multiple layers are combined by performing a summation over all the outputs while keeping the feature-map size consistent. SPaS blocks are interspaced by transition layers which perform downsampling and doubles the number of channels.*

$$f(\mathbf{x};\theta_S) = f(\mathbf{x};\theta_{S,R=1}). \tag{15}$$

Figure 1 shows a high-level overview of the architectures of D-MLP, S-MLP, and SPaS-MLP with their associated mathematical definition.

We use $\theta_{S,R}^{k,l}$ to describe the parameters connecting layer k to layer l. $\theta_{S,R}^{k,l}$ can be decomposed into sparse matrices:

$$\mathbf{W}_k \in \mathbb{R}^{n^k \times n^l}, \mathbf{b}_k \in \mathbb{R}^{n^l}, \tag{16}$$

where n^k and n^l represent the number of neurons in layers k and l respectively.

The forward pass of a SPaS-MLP is similarly defined as in Equation 3. In a SPaS-MLP, each function f_l is defined as follows:

$$f_l(x) = a((\sum_{k=1}^{l} \mathbf{x}_{(k-1)}\mathbf{W}_k + \mathbf{b}_k)), \tag{17}$$

where $\mathbf{W}_k$ and $\mathbf{b}_k$ represent the weights and biases utilized in a forward pass from layer k to layer l and a is the activation function. Important to note is that biases are not shared between layers: neurons in SPaS-MLP have multiple biases associated with them, and each set of biases b_k is used in the forward pass to a different subsequent layer.

4.1.2 SPaS-CNN Our proposed SPaS-CNN improves and enhances the typical DenseNet architecture by parameterizing its shortcut connections, enabling the relative gradient signal strength of reused features to be learned. Similar to DenseNet, the SPaS-CNN architecture consists of blocks interspaced by transition layers. A crucial difference, however, between blocks in DenseNet and SPaS-CNN, is the fact that SPaS-CNN combines results from previous layers through summation of their outputs, keeping the number of channels fixed, while DenseNet performs a concatenation, resulting in the number of channels to grow. DenseNet and SPaS-CNN both have complete access to all unaltered preceding outputs, but SPaS-CNN additionally has a unique layer for each preceding output that can mutate the features separately. This allows SPaS-CNN to more selectively pick out features from previous outputs, whereas DenseNet needs to share its weights over increasing amounts of features as dense block depth increases since a DenseNet typically gains more and more input channels than output channels as dense blocks become deeper.

SPaS Blocks. We describe the connectivity of a layer in a SPaS Block as follows:

$$\mathbf{x}_l = H_{0,l}(\mathbf{x}_0) + H_{1,l}(\mathbf{x}_1) + ... + H_{l-1,l}(\mathbf{x}_{l-1}), \tag{18}$$

where we define each $H_{k,l}$ as the function for layer k to l. Each preceding output is transformed by its own function with a unique set of parameters. Note that the outputs are summed, resulting in a fixed feature-map size within blocks.

Algorithm 1 Sparse Parameterized Shortcut Evolutionary Training (SPaSET) pseudocode

```
Set S, R;                                          ▷ Network Topology Hyperparameters
Set ξ;                                             ▷ Network Evolution Hyperparameters
Initialize network topology (Equations 20 and 21);
Initialize training parameters;
for each training epoch e do
    Perform standard training procedure and weight updates;
    Remove a fraction ξ of the smallest magnitude weights;
    if R is fixed then
        Randomly regrow new sequential connections in the same amount as the ones previously removed ;
        Randomly regrow new shortcut connections in the same amount as the ones previously removed ;
    else
        Randomly regrow an equal amount of sequential and shortcut connections until global-sparsity S is satisfied.
```

A SPaS block consists of k layers, which we will refer to as a block depth parameter. DenseNets also use k to describe the depth of a dense block, but they more specifically refer to k as a growth factor, since dense blocks grow their feature-maps.

Transition layers. SPaS blocks are interspaced by transition layers. These transition layers perform two functions. First, feature-map size typically must increase with the depth of a CNN to ensure good performance. Since feature-map size does not grow within SPaS blocks this is performed in transition layers. This is done through a $1x1$ convolutional layer that doubles the number of channels. Second, we are unable to downsample feature-maps within SPaS blocks, thus this action is performed by transition layers. This is done by a $2x2$ average pooling layer.

4.2 Sparsity initialization

Before we define how a sparse mask is initialized in a SPaS network we must also clarify how sparsity is calculated in a SPaS network. We do not calculate the sparsity of a SPaS network against the maximum amount of parameters in a SPaS network ($||\theta||$), but against the max amount of parameters in its exclusively sequential counterpart ($||\theta_{D=1}||$). This means that any sparsity parameter S for a network with SPaS would equal the number of parameters in the same network without SPaS. More specifically, we extend the sparsity calculation from Equation 8 to explicitly state this fact:

$$S = 1 - \frac{||\theta||_0}{||\theta_{D=1}||}, \tag{19}$$

With this clarified, we can now explain how we initialize a sparse mask for a SPaS network. Unless we have the rare case where $||\theta_{D=1}|| = ||\theta_{D>1}||$, we must make a clear distinction in the initialization of sequential and shortcut connections and define separate probability functions for both.

Since sparsity is calculated against an exclusively sequential network ($||\theta_{D=1}||$), we calculate the probability of a sequential connection in a SPaS network as:

$$p(\mathbf{W}_{ij}^{l-1,l}) = (1 - S) \cdot R, \tag{20}$$

where we mutate the base probability $1 - S$ with the novel ratio parameter R. We calculate the probability of a shortcut connection in a SPaS network as:

$$p(\mathbf{W}_{ij}^{k,l}) = \frac{||\theta_{D=1}||}{||\theta_{D>1}||} \cdot (1 - S) \cdot (1 - R), \tag{21}$$

where we normalize the base probability $(1 - S)$ against the ratio of $||\theta_{D=1}||$ with $||\theta_{D>1}||$ and mutate that with $(1 - R)$.

4.3 Sparse Parameterized Shortcut Evolutionary Training (SPaSET)

We propose SPaSET (**S**parse **Pa**rameterized **S**hortcut **E**volutionary **T**raining), based on SET, in which the evolutionary rules support SPaS. Pseudocode of SPaSET can be found in Algorithm 1. While SET and SPaSET both keep their global sparsity parameter S fixed throughout training, there is an important difference in the fact that SET additionally fixes layer-sparsity while SPaSET allows layer-sparsity to change throughout training. SPaS networks support two modes of R: fixed and unfixed. When R is fixed, the ratio of sequential-shortcut is restricted to the initialized ratio during evolutionary steps. When R is unfixed, the ratio of sequential-shortcut connections is allowed to change in evolutionary steps. SPaSET departs from both N2NSkip and ToST, where the ratio R is always fixed (though ToST phases out its shortcut layers through gated parameters during training) and the sparse masks in shortcut layers are static.

Pruning. For pruning we implemented a bottom-k pruning strategy: Removing a fraction of the smallest magnitude weights. There is a slight departure from SET here: SET prunes an equal amount of positive and negative weights while SPaSET is unrestricted by this rule and can prune any number of positive or negative weights.

Regrowing. We apply random regrow by regrowing the exact amount of connections as was removed during the pruning stage until the global-sparsity parameter S is satisfied. Depending on whether R is fixed there are two different regrowth strategies: If R is fixed we regrow the exact amount

of sequential and shortcut connections as was removed during the pruning stage. If R is unfixed we regrow an equal amount of sequential and shortcut connections.

Modified SET (SET*). The modifications from SET to SPaSET are also applied to our implementation of SET, which we will refer to as SET* from hereon out. SET* differs from SET by lifting the requirement of pruning equal amounts of positive and negative parameters and by loosening the fixed layer-sparsity requirement to a fixed global-sparsity.

5 Experimental Setup

In this section, we explain the experimental setup. We first present the datasets used, after which we explain the evaluation metrics. Finally, we go into detail on the models used for our experiments and the training scheme.

5.1 Datasets

Computer vision. We perform our experiments on the CIFAR-10 and CIFAR-100 datasets [21]. The CIFAR-10 dataset consists of 10 classes distributed over 60000 32x32 colour images, with 6000 images per class. The dataset consists of five training batches and one test batch, each consisting of 10000 images (1000 images per class). CIFAR-100 is similar to CIFAR-10, except that it has 100 classes distributed over 600 images. We use the prefixes C10- and C100- to indicate datasets models were trained on: C10-MLP and C100-MLP are MLPs trained on CIFAR-10 and CIFAR-100 respectively.

Numerical classification. We perform experiments on a subset of numerical classification tasks from Tabular Benchmark [12]. All datasets are binary classification problems. Datasets are prepared with an 80-20 train-test split. See Table 1 for a list of datasets used.

dataset_name	n_samples	n_features
electricity	38474	7
pol	10082	26
house_16H	13488	16
MagicTelescope (MT)	13376	10
Bioresponse (BR)	3434	419
MiniBooNE	72998	50
Higgs	940160	24
jannis	57580	54

Table 1. *Selection of numerical classification datasets from Tabular Benchmark [12] used in validation of SPaS-MLP.*

5.2 Models

All models were constructed using the PyTorch [34] library. Sparsity masks for sequential and shortcut layers were implemented with binary masks. We provide a framework that allows SPaS-MLP and SPaS-CNN networks with various depths/widths to be initialized with a specified ratio R and sparsity S (or parameter budget). We provide a training framework that allows networks to be trained statically or with SET*/SPaSET and control over evolutionary parameters such as the pruning rate ξ, (un)fixed R, and evolution interval. Our implementation will be accessible in the near future on https://github.com/MaukWM/SPaS.

ReLU can result in very sparse activations[1, 18], which can, in turn, decrease trainability by blocking gradient flow. This can be especially problematic in networks that are already sparse, such as SPaS. To mitigate this issue we decided to use Mish as the activation function for all non-final layers. All models were trained for 100 epochs, with a four-epoch early-stopping policy. The models are trained with the Adam optimizer [20] and a learning rate of 0.005. It needs to be noted that while Adam provides an adaptive learning rate that can improve convergence among dense models, the smaller learning rates later in the training process can hinder the search space exploration of good connectivity by reducing the ability of new connections to be competitive with old ones, resulting in new connections to be pruned more quickly. [45] shows that optimizers such as Adam and RMSProp [14] can, at times, have poor performance when used with L2 regularization or data augmentation in sparse networks. L1-Regularization is added with a coefficient of $5e^{-5}$. Cross entropy is used as a loss function. The models are trained with a batch size of 128. For a quick overview of the MLP and CNN models used in the computer vision experiments and their maximum parameter counts, we refer to Table 2.

Model	$\|\|\theta_{D=1}\|\|$	$\|\|\theta_{D>1}\|\|$	$\|\|\theta\|\|$
D/S-MLP	328k	-	328k
SPaS-MLP	328k	657k	985k
D/S-CNN	149k	-	149k
DenseNet	276k	-	276k
SPaS-CNN	149k	145k	295k

(a) CIFAR-10

Model	$\|\|\theta_{D=1}\|\|$	$\|\|\theta_{D>1}\|\|$	$\|\|\theta\|\|$
D/S-MLP	337k	-	337k
SPaS-MLP	337k	952k	1.3M
D/S-CNN	155k	-	155k
DenseNet	298k	-	298k
SPaS-CNN	155k	145k	301k

(b) CIFAR-100

Table 2. *Models used in the experiments on CIFAR-10/100 and their maximum parameter counts. D/S-MLP and D/S-CNN refer to dense/sparse MLPs/CNNs. $||\theta_{D=1}||$ refers to the count of possible parameters in all sequential layers. $||\theta_{D>1}||$ refers to the count of possible parameters in all shortcut layers. $||\theta||$ refers to the count of possible parameters in all layers.*

5.2.1 MLP All MLP models are constructed with three hidden layers of 100 neurons. We experimented on a dense MLP (D-MLP), a sparse MLP (S-MLP), and an MLP with

SPaS (SPaS-MLP). Each SPaS-MLP network is initialized with $R = 0.5$ which is kept fixed throughout training.

5.2.2 CNN We perform experiments on three different types of convolutional networks. A CNN with SPaS (SPaS-CNN) with $R = 0.5$, a plain CNN with exclusively sequential connections whose architecture can be described as a SPaS-CNN with $R = 1$, and finally a DenseNet.

The DenseNet consists of 3 dense blocks with growth factor $k = 4$. After the dense blocks, we have a classification layer that performs $4x4$ global average pooling with a stride of 4, followed by a forward layer with softmax. Important to note is that we depart slightly from the original DenseNet implementation by leaving out the bottleneck layers, which are used to improve computational efficiency by reducing the number of input feature-maps. We decided against adding these layers as we wanted the SPaS-CNN and DenseNet architectures to be as similar as possible. The SPaS-CNN architecture does not include bottleneck layers as SPaS-CNN would not benefit from any feature-map size reduction delivered by bottleneck layers as feature-map size does not grow. For a high-level architecture overview of the DenseNet, we refer to Table 6b in Appendix A.

In the (SPaS-)CNN, we define each $H_l(\cdot)$ as a composite function of batch normalization (BN), followed by a Mish activation function [29] and finally a 3x3 convolution (Conv). We chose a 3x3 kernel as it is also the typical choice in DenseNet architectures. In the transition layers, we add BN, followed by Mish before the $1x1$ convolutional layer that doubles the size of the feature-maps and the $2x2$ average pooling layer that downsamples the input. The SPaS-CNN consists of 3 SPaS blocks with depth $k = 3$. After the SPaS blocks, we have a classification layer that performs $4x4$ global average pooling with a stride of 4, followed by a forward layer with softmax. For a high-level architecture overview of the CNN and SPaS-CNN, we refer to Table 6a in Appendix A.

5.3 Compression

Compression rates of S-MLP and SPaS-MLP are in relation to the number of parameters in their D-MLP counterparts and are inversely proportional to network density: an S-MLP and SPaS-MLP with a compression rate of 5x have the same amount of parameters at 0.20 density (or 0.80 sparsity). Similarly, a compression rate of 0.5x refers to a network with a density of 2.0. Compression rates <1x are only applicable to SPaS-MLPs that contain more active parameters than are possible in their D-MLP counterparts. Compression rates in SPaS-CNNs are calculated differently than in MLPs: compression is not in comparison to its dense counterparts but to the architecture's own parameter count: a SPaS-CNN at 5x compression has more parameters than the same S-CNN at 5x compression as the max parameter count in SPaS-CNNs is larger than in S-CNNs. By doing this, we actually stray from how we propose sparsity is calculated in SPaS networks, as described in Section 4.2. We refer to the discussion in Section 7.1 as to why this is done.

5.4 Evaluation Metrics

For numerical classification tasks, we compare SPaS-MLPs with MLPs in both static and dynamic sparsity regimes. We evaluate the performance of a model by looking at the model's accuracy on the test set against compression rates of up to 75x.

For the computer vision tasks, we compare SPaS-MLPs with MLPs and we compare SPaS-CNNs with CNNs and DenseNets in both static and dynamic sparsity regimes up to compression rates of 75x. As in numerical classification, we evaluate the (SPaS-)MLPs by looking at the validation accuracy against compression rates. However, compression rates between CNNs, DenseNets, and SPaS-CNNs are not directly comparable, thus we compare by inferencing FLOPs (I-FLOPs).

5.5 Ablation Studies

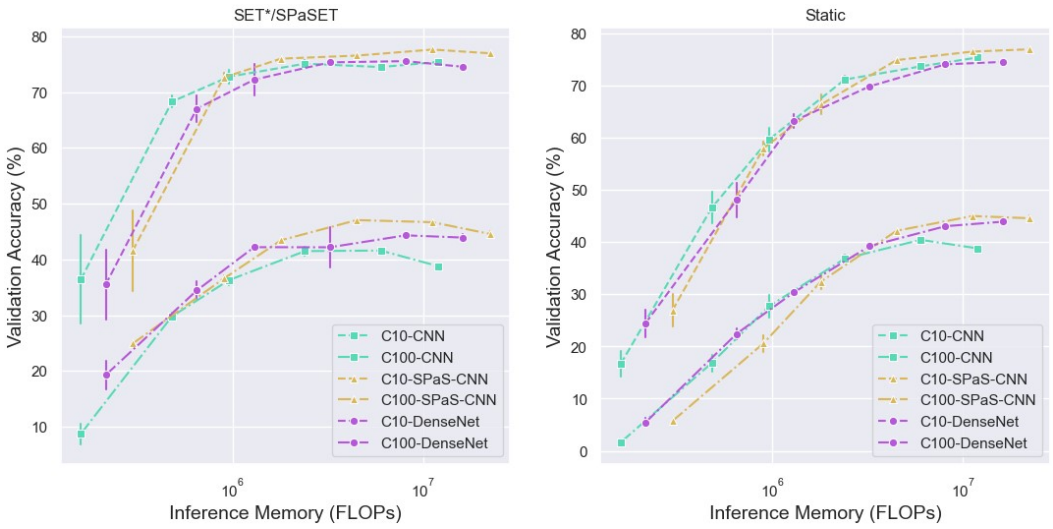
To investigate the efficacy of SPaS, it is imperative to investigate the impact that varying levels of R can have on the performance of the models, which is why we perform an additional ablation study for SPaS-MLP and SPaS-CNN over R in various sparsity regimes. These ablation studies are performed on the same model architectures and training regimes as described in the previous sections.

6 Results

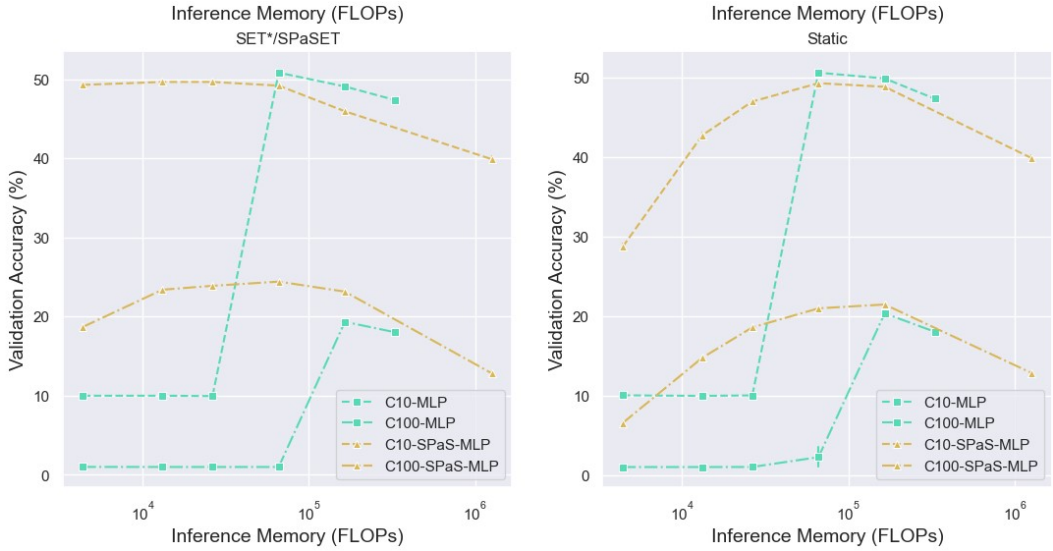
In this section, we gather the results from the experiments. We present our findings on SPaS for MLPs and CNNs over static and dynamic sparsity on numerical classification tasks selected from Tabular Benchmark [12] and the CIFAR10/100 datasets.

SPaS enables even sparser MLPs. In Table 3 and Figures 3b, 4 we observe that S-MLPs are unable to achieve non-random results beyond compression rates of 5x on both numerical classification and computer vision tasks, whilst SPaS-MLPs are still able to learn up to compression rates of 75x. The performance dropoff in S-MLPs is very sudden and is most likely caused by impeded backpropagation whereby the sparsity becomes too excessive for the model. SPaS-MLP can model the data with much fewer parameters and, while SPaS does not necessarily increase the expressive capabilities of the network, it enables gradients to be propagated between all layers, including direct propagation between the input and output. This enhanced connectivity relieves backpropagation concerns inherently present in S-MLPs.

SPaS on MLPs is comparable to other approaches and improves performance on CIFAR100 and MiniBooNE. In Table 5 and Figures 4, 7 we observe on numerical classification tasks that SPaS-MLP has comparable performance



(a) *CNNs*



(b) *MLPs*

Figure 3. *Validation accuracy (%) with standard error by inferencing FLOPs (I-FLOPs) for MLPs and CNNs on CIFAR10/100, where the models are prefixed as C10 and C100 respectively. The left column contains models dynamically trained with SET*/SPaSET, and the right column contains statically trained models. A tabular representation of these results can also be found in Tables 3 and 4. In both static and dynamic sparsity regimes, SPaS-CNN is able to outperform both plain CNNs and DenseNets on CIFAR10/100 for equal or fewer I-FLOPs.*

and in some cases outperforms its D-MLP/S-MLP counterparts: SPaS-MLP is only outperformed by its counterparts on the CIFAR-10, Higgs, and jannis dataset by ~1%, while SPaS-MLP outperforms its counterparts by ~25% on MiniBooNE. In Table 3 and Figure 3b, we find that S-MLP and SPaS-MLP provide significant improvements in performance over its D-MLP counterpart on CIFAR-10/100. On CIFAR-10, the S-MLP consistently outperforms the best SPaS-MLP by ~1% in both static and dynamic sparsity regimes. On CIFAR-100, SPaS-MLP consistently outperforms S-MLP, achieving a ~5% higher accuracy at a compression rate of 5x and maintaining its superiority even at compression rates of up to 25x.

Interesting to note is that making SPaS-MLP fully dense is detrimental to its performance in computer vision tasks, resulting in a performance decrease of ~7% over its D-MLP counterpart. The fully dense SPaS-MLPs for CIFAR-10/100 have over 3x the parameters of their D-MLP counterparts (more than the ~1.7x numerical classification tasks), the network does not overfit so the decrease in performance is most likely explained by an oversaturation of parameters

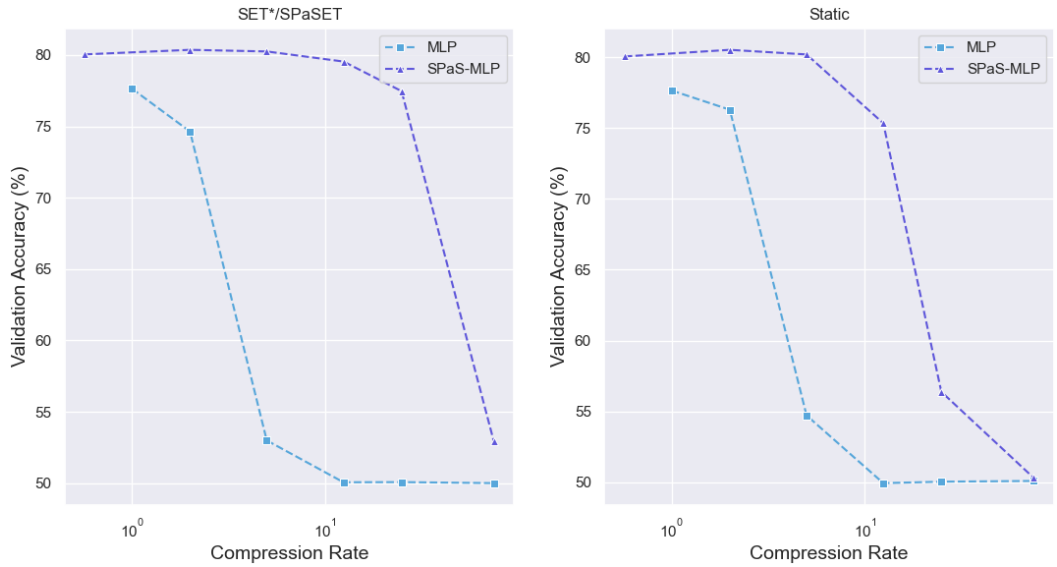


Figure 4. *Average IQM of the validation accuracy (%) for the numerical classification datasets as described in Table 1 by compression rate (from 1x to 75x). A breakdown of the average IQM into its subcomponents can be found in Figure 7 in Appendix C. In both static and dynamic sparsity regimes, SPaS-MLP maintains a higher average performance over all compression rates. SPaSET additionally enables SPaS-MLP to uphold performance up to higher compression rates in comparison to a static sparsity regime.*

	Dataset	CIFAR-10				CIFAR-100			
		MLP		SPaS-MLP		MLP		SPaS-MLP	
Sparsity Mode	Compression	IQM	I-FLOPS	IQM	I-FLOPS	IQM	I-FLOPS	IQM	I-FLOPS
N/A	Fully Dense	**47.44** (1x)	1x (3.3e5)	40.01 (0.33x)	3.82x (3.3e5)	**18.04** (1x)	1x (3.4e5)	12.86 (0.26x)	3.82x (3.4e5)
Static	2x	**49.84**	0.50x	48.94	0.50x	20.29	0.50x	**21.46**	0.50x
	5x	**50.60**	0.20x	49.35	0.20x	-	0.20x	**20.97**	0.20x
	12.5x	-	0.08x	**47.01**	0.08x	-	0.08x	**18.39**	0.08x
	25x	-	0.04x	**42.72**	0.04x	-	0.04x	**14.78**	0.04x
	75x	-	0.01x	**28.68**	0.01x	-	0.01x	**6.52**	0.01x
SET*/SPaSET	2x	**49.70**	0.50x	46.08	0.50x	19.20	0.50x	**22.98**	0.50x
	5x	**50.78**	0.20x	49.18	0.20x	-	0.20x	**24.56**	0.20x
	12.5x	-	0.08x	**49.51**	0.08x	-	0.08x	**24.00**	0.08x
	25x	-	0.04x	**49.78**	0.04x	-	0.04x	**23.39**	0.04x
	75x	-	0.01x	**49.33**	0.01x	-	0.01x	**18.77**	0.01x

Table 3. *Validation accuracy (%) on the CIFAR10/100 dataset for the MLPs. Performances highlighted in **bold** were the best performing for that compression rate and dataset. Results with a - did not achieve a non-random performance. Fully dense models have the compression rate added in (parenthesis). We observe that MLPs in the 2x-5x compression range are the best-performing models on CIFAR10 while SPaS-MLPs are the best-performing on CIFAR100 in that same compression range. As the density regime approaches >5x, MLPs are unable to achieve a non-random performance, while SPaS-MLP is still able to achieve results.*

and gradients, impeding its ability to properly model the data.

SPaS improves performance in CNNs. In Tables 4, 5 and Figure 3a we observe that SPaS-CNN can achieve higher accuracies than its D-CNN/S-CNN counterparts. On CIFAR-10 and CIFAR-100, the best performing SPaS-CNN models achieve 77.75% and 46.83% accuracy respectively, while D/S-CNN achieves up to 75.49% and 41.79%, giving SPaS-CNN +2.26% and +5.04% performance boost on CIFAR-10/100 over D/S-CNNs. SPaS-CNN even improves performance over DenseNets, where the best-performing DenseNets achieve 75.86% and 45.47% on CIFAR-10/100 respectively, giving SPaS-CNN a +1.89% and +1.36% increased performance over DenseNets. SPaS-CNN is able to achieve these improved results over CNNs and DenseNets at less or equal to the same amount of I-FLOPs.

SPaS-CNN does not outperform CNNs or DenseNets beyond compression rates of 5x: after 5x compression, the validation accuracy per I-FLOP count is less than either the CNN or the DenseNet, this is most noticeable in CIFAR-100.

The main advantage that SPaS-CNNs provide over plain CNNs is the fact that SPaS enables feature reuse. The

		CIFAR10						CIFAR100					
		CNN		DenseNet		SPaS-CNN		CNN		DenseNet		SPaS-CNN	
Sparsity Mode	Compression	IQM	I-FLOPS	IQM	I-FLOPS	IQM	I-FLOPS	IQM	I-FLOPS	IQM	I-FLOPS	IQM	I-FLOPS
N/A	1x	75.49	1x (1.2e7)	74.67	1.35x (1.2e7)	**77.00**	1.87x (1.2e7)	38.89	1x (1.2e7)	**44.49**	1.35x (1.2e7)	44.33	1.87x (1.2e7)
Static	2x	74.28	0.49x	74.21	0.68x	**76.39**	0.94x	40.33	0.49x	43.29	0.68x	**44.94**	0.94x
	5x	71.32	0.19x	69.96	0.27x	**75.15**	0.37x	37.05	0.19x	39.21	0.27x	**42.34**	0.37x
	12.5x	60.53	0.08x	63.90	0.11x	**67.47**	0.14x	29.29	0.08x	30.70	0.11x	**33.30**	0.14x
	25x	46.35	0.04x	50.69	0.05x	**57.55**	0.07x	18.22	0.04x	**23.22**	0.05x	21.61	0.07x
	75x	15.66	0.01x	24.44	0.02x	**25.40**	0.02x	1.31	0.01x	5.63	0.02x	**5.81**	0.02x
SET*/SPaSET	2x	74.35	0.49x	75.86	0.68x	**77.75**	0.94x	41.79	0.49x	44.48	0.68x	**46.81**	0.94x
	5x	75.35	0.19x	75.50	0.27x	**76.55**	0.37x	41.60	0.19x	45.47	0.27x	**46.83**	0.37x
	12.5x	73.85	0.08x	74.82	0.11x	**76.16**	0.14x	37.28	0.08x	42.20	0.11x	**43.59**	0.14x
	25x	68.24	0.04x	69.25	0.05x	**72.92**	0.07x	30.07	0.04x	35.57	0.05x	**36.52**	0.07x
	75x	38.47	0.01x	34.80	0.02x	**43.16**	0.02x	9.33	0.01x	19.08	0.02x	**24.97**	0.02x

Table 4. *Validation accuracy (%) IQM on CIFAR-10/100 for the CNN, DenseNet and SPaS-CNN. Performances highlighted in **bold** were the best performing for that compression rate and dataset. Results with a - did not achieve a non-random performance. For each model, the left column shows the IQM of the validation accuracy, and the right column shows the FLOP counts for a single inference (I-FLOPS). In CNNs, we observe that SPaS improves performance among all compression rates in both dynamic and static sparsity regimes. In particular, we observe that SPaS-CNN trained with SPaSET outperforms both S-CNN and DenseNet on CIFAR-10 and CIFAR-100 by at least ∼2%.*

DenseNet architecture also provides feature reuse, but SPaS even provides a performance boost over DenseNets. While DenseNet offers comprehensive feature reuse through the copying and concatenation of all previous features, SPaS-CNN, through parameterization of its skip connections, enables the network to mutate features from previous layers. This allows SPaS-CNN to perform more selective feature reuse and gives the network the ability to learn the relative importance of gradient signals from reused features.

It is of note that SPaS-CNN contains more trainable weights than its CNN and DenseNet counterparts, so we performed additional experiments with parameter budgets. In line with the experiments by compression, we find that SPaS improves performance among higher-density regions, but as the parameter budget shrinks, SPaS networks are unable to outperform either CNNs or DenseNets. For the results of the parameter budget experiment, we refer to Appendix B.

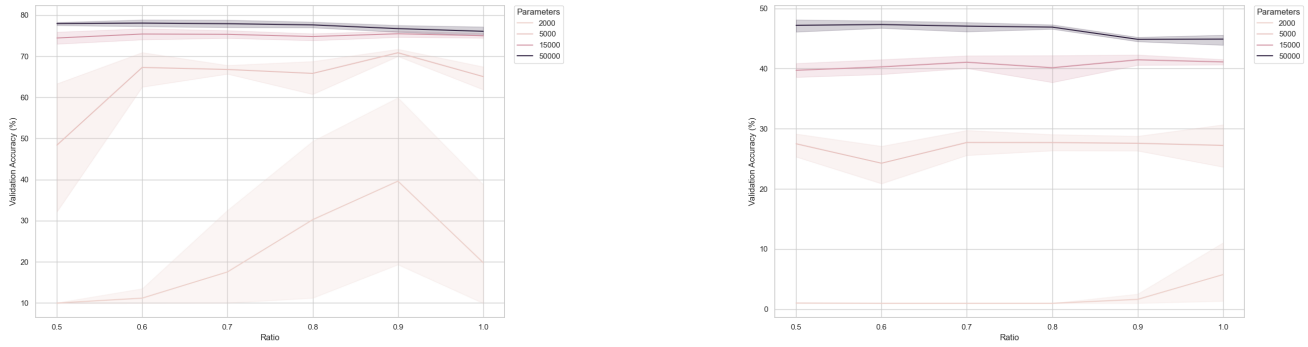
The ratio parameter R can have a significant impact on performance in low-density regimes. In Figure 5 we observe that in high-density regimes for SPaS-CNN, the value of R does not significantly impact performance, but R values in the range of [0.5-0.8] provided the best-performing models. In low-density regimes, the value of R becomes more impactful, whereby the choice of R can give as much as a 30% difference in performance. We find that lower ratios are preferred in low-density regimes, in the range of [0.8-1.0]. As the parameter budget becomes increasingly constrained and kernels become increasingly sparse, there is an increased risk of dead kernels, becoming unable to properly learn features and backpropagate. Additionally, lower ratios (more skip connections) at these constrained parameter budgets need to spread its parameters over more kernels, as there are typically more available kernels in skip layers than sequential layers.

	MLP		CNN		DenseNet	
Dataset	ΔIQM	ΔMax	ΔIQM	ΔMax	ΔIQM	ΔMax
CIFAR-10	**-1.00**	-1.25	**+2.26**	+1.34	**+1.89**	+1.45
CIFAR-100	+4.27	**+4.31**	**+5.04**	+4.84	**+1.36**	+0.59
electricity	+0.30	**+1.49**	-	-	-	-
pol	**+0.73**	+0.24	-	-	-	-
house_16H	**+0.62**	-0.48	-	-	-	-
MT	**+0.63**	+0.21	-	-	-	-
BR	**+0.35**	-1.10	-	-	-	-
MiniBooNE	**+24.54**	+4.88	-	-	-	-
Higgs	**-1.19**	-1.51	-	-	-	-
jannis	**-0.96**	-1.28	-	-	-	-

Table 5. *Performance benefit of SPaS-MLP over MLP, and SPaS-CNN over CNN and DenseNet. Selected models are the best performing for that dataset. Performance differences are obtained by subtracting the validation accuracy of the baseline model from the SPaS model. We calculate two metrics of performance difference: the interquartile mean (IQM) and the individual best-performing model (Max). A positive Δ indicates an advantage for SPaS over its counterpart. Results highlighted in **bold** indicate that SPaS has a larger benefit in performance for that metric. While SPaS does not always deliver higher performance in MLPs, SPaS consistently improves performance in CNNs. Additionally, in almost all cases, we observe that Δ IQM > Δ Max, indicating that SPaS brings more benefits to consistently finding high-performing models than finding a singular high-performing model.*

In Figure 6 we observe that SPaS-MLPs with both dynamic and static masks have higher performance as ratios decrease in low-density regimes, specifically, density regimes of [0.01-0.001]. In the most extreme density regime, the choice of R can result in differences of ∼10% in performance. Density regimes of [0.25-0.05] give a clear advantage to models with more sequential connections.

SPaS and SPaSET improve model stability over its counterparts. In Table 5 we observe that the performance



(a) CIFAR-10 **(b)** CIFAR-100

Figure 5. *Ablation on sequential-skip ratios for SPaS-CNNs on CIFAR-10/100. The ratio of sequential-skip connections (range [0.5-1.0]) is on the x-axis and the IQM of the Validation Accuracy (%) is on the y-axis. Experiments are performed with parameter budgets in the range of [2000-50000]. We observe that the highest density regime (50000 parameters) is somewhat agnostic to the choice of ratio R, but optimal performance is achieved with ratios in the range of [0.5-0.8]. In the lowest density regime (2000 parameters), we find the inverse: ratios of 0.9 and 1.0 provide the best performance on CIFAR-10/100 respectively.*

benefits of SPaS are more prominently visible in the validation accuracy's IQM than the individual best-performing models. This indicates that the performance benefits of SPaS and SPaSET are more apparent in model stability. SPaS in combination with SPaSET is more consistently able to find optimal sparse masks for both sequential and skip layers compared to models without SPaS. This improved stability most likely stems from the improved flow of gradients provided by SPaS and SPaSET's ability to find these optimal sparse masks.

SPaSET enables SPaS networks to maintain performance in high sparsity regimes. In Tables 3, 4 and Figures 3, 4 we find that SPaS networks trained with SPaSET provide a large improvement over SPaS networks trained with static sparse masks. SPaSET provides a slight boost in accuracy among high-density regimes, but as low-density regimes are approached the performance difference between static and SPaSET-trained networks becomes even more apparent. In CNNs, at compression rates of 75x, SPaSET provides up to ∼20% higher validation accuracy on both CIFAR10/100 over its static counterparts. In MLPs, SPaSET boosts performance by ∼20% and ∼12% on CIFAR10 and CIFAR100 respectively. In numerical classification, we observe a ∼20% boost at 25x compression.

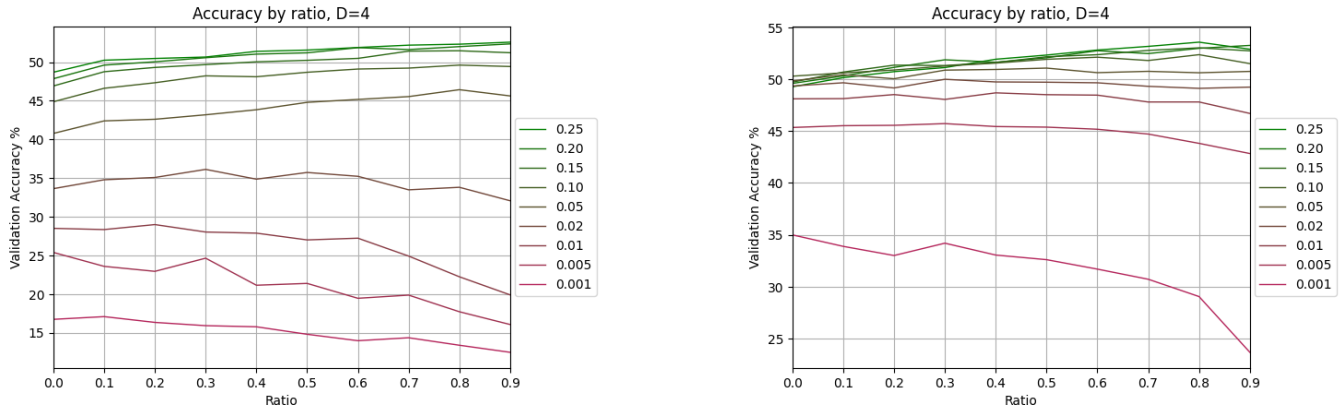
7 Discussion & Conclusion

In this thesis, we proposed a novel architecture that combines the principles of sparsity and shortcut connections called SPaS, and an evolutionary training algorithm that enables sparse masks for sequential and shortcut connections to be discovered during training, called SPaSET.

We introduced the SPaS architecture for MLPs and CNNs and, to answer **RQ1.** (*What role do sparse shortcut connections play in the performance of ANNs?*), we find that the skip connections in SPaS provide our models with improved gradient propagation across layers and enable feature reuse, delivering both improved model stability and higher performances. Both SPaS-MLP and SPaS-CNN deliver 4∼5% increased performance on CIFAR-100 over a plain MLP and CNN. SPaS-CNN even improves performance over DenseNets by 1∼2%. SPaS-MLP only provides a slight improvement in performance over its D/S-MLPs counterparts on most of the numerical classification tasks but is able to deliver a significant performance boost of over 24% over the MiniBooNE dataset.

To answer **RQ2.** (*Can we use the principles of Dynamic Sparse Training to find optimal sparse masks for shortcut connections?*), we introduced SPaSET. SPaSET applies the principles of Dynamic Sparse Training (DST) to shortcut connections, delivering us a training scheme that enables optimal sparse mask for both sequential and shortcut connections to be discovered during training in SPaS networks. SPaSET improves model performance by up to 20% over static sparse masks in high sparsity regimes on both computer vision and numerical classification tasks.

To answer **RQ3.** (*To what extent can SPaS allow us to create even sparser networks?*), we found that SPaS enables MLPs to be compressed up to 25x in numerical classification and 75x in computer vision over 5x on plain MLPs while remaining competitive with plain dense and sparse MLPs. As sparsity increases and the network becomes more constrained, connections and neurons can become isolated. This can potentially result in dead ends and restricted backpropagation. Additionally, as the network becomes more compressed, more and more of the parameter budget must be expended to guarantee



(a) Static

(b) SPaSET

Figure 6. *Ablation on sequential-skip ratios for SPaS-MLPs on CIFAR-10. The ratio of sequential-skip connections (range [0.0-0.9]) is on the x-axis and the average (5 runs) Validation Accuracy (%) is on the y-axis. Experiments are performed with densities in the range of [0.25-0.001]. We observe that, in both static and dynamic contexts, low-density regimes ([0.01-0.001]) enjoy improved performance from lower values of R, while high-density regimes ([0.25-0.10]) benefit from higher values of R.*

propagation from input to output. In an MLP, propagating over multiple layers necessitates multiple connections, while a SPaS-MLP can do the same with just a single connection.

SPaS additionally introduces the novel sequential-shortcut ratio parameter R. We find that the tuning of R can have a significant impact on the performance of both SPaS-MLP and SPaS-CNN in varying sparsity regimes. On CIFAR10/100, this difference is most significant in low-sparsity regimes, resulting in differences of 10% and 30% in MLPs and CNNs respectively.

7.1 Limitations

Binary masks. The sparse masks for both sequential and skip connections were implemented with binary masks. These binary masks result in the computational overhead growing exponentially with layer depth: As layer depth grows the amount of possible skip connections grows exponentially with each additional layer. This fact greatly limits the possible depth of SPaS networks we can perform experiments on.

Limited model sizes. Due to the computational overhead from the use of binary masks the sizes of our models were very limited as larger models would quickly run into memory restrictions. This has resulted in the CNNs being limited up to 300k parameters. DenseNets are usually employed with millions of parameters which can give performances of 93-95% on CIFAR-10 and 72-80% on CIFAR-100, whereas our limited DenseNets go up to 75% and 45% on CIFAR-10 and CIFAR-100 respectively. Future experiments with more computationally efficient masks, larger memory capacity, or truly sparse implementations would allow us to make a better comparison between the CNN, DenseNet, and SPaS-CNN.

Comparing CNN architectures. One issue that arose during experimentation was the question of how to make a fair comparison between CNN models. SPaS in MLPs allowed these models to still be compared one-to-one with the proposed sparsity calculation as a metric (namely, calculating sparsity by the SPaS networks' plain dense counterpart). With this metric, the amount of I-FLOPs for an 80% sparse S-MLP and SPaS-MLP are identical. The various CNN architectures featured in this thesis, however, have more complex features in their architectures, such as SPaS/dense blocks (and their associated growth rate), transition layers, and channel/kernel sizes. As a result, the proposed sparsity metric proved unsuitable for making one-to-one comparisons between different CNN architectures. Consequently, we opted to calculate sparsity relative to each model's own architecture. To facilitate comparisons between our CNN models, we used I-FLOPs as the measure of computational cost as opposed to compression rates as we did with MLPs.

8 Future work

Truly sparse. As the implementation was done with binary masks, the scale of experiments was severely limited. To investigate the efficacy of SPaS in very deep networks, experiments with a truly sparse implementation are required.

Investigating connectivity. It would be interesting to investigate the connectivity of SPaS vs. non-SPaS networks using metrics from graph theory (e.g. heat diffusion) to evaluate whether SPaS (and SPaSET) result in improved connectivity patterns. We have also not investigated what the resulting topologies of the best-performing models look like. It would be interesting to investigate what topological features result in more parameter-efficient models.

Unfixed R. In this thesis all experiments were performed

with a fixed R, disallowing the ratio of sequential-shortcut to change throughout training. Future work needs to determine what effect a fixed or unfixed R has on stability, performance, and the final connectivity pattern. Experiments with unfixed R could additionally lead to insights on how to tune R for various model architectures or indicate to what capacity SPaS models can self-tune R.

ERK sparse initialization We perform a uniform initialization of weights in SPaS networks. Typically, Erdős–Rényi random graphs (ERK) are used to initialize sparse topologies. Future work can seek how to extend ERK initialization to also apply to SPaS networks.

Kernel sizes in SPaS-CNN. We chose 3x3 kernels for our SPaS-CNN as it is also the typical choice of kernel size in DenseNets. Since SPaS-CNN has many fundamental differences in architecture, it would be wise to investigate how the choice of kernel size can impact performance in varying sparsity regimes and values of R.

Different SPaS Block Architectures. Our proposed implementation of SPaS in CNNs is not the only way to implement parameterized shortcut connections in CNNs. We could for example make the connections in SPaS Blocks partly residual, as inspired by ResNets:

$$\mathbf{x}_l = H_{0,l}(\mathbf{x}_0) + H_{1,l}(\mathbf{x}_1) + ... + H_{l-1,l}(\mathbf{x}_{l-1}) + \mathbf{x}_{l-1}, \quad (22)$$

or "densely" residual, not just feeding residual information from the previous layer but all previous layers:

$$\mathbf{x}_l = H_{0,l}(\mathbf{x}_0)+\mathbf{x}_0+H_{1,l}(\mathbf{x}_1)+\mathbf{x}_1+...+H_{l-1,l}(\mathbf{x}_{l-1})+\mathbf{x}_{l-1}. \quad (23)$$

We could also concatenate the results from SPaS Blocks into a growing feature map (as opposed to performing summation, keeping feature map size fixed within blocks). Such an implementation would more closely resemble a DenseNet:

$$\mathbf{x}_l = H_l([H_{0,l}(\mathbf{x}_0), H_{1,l}(\mathbf{x}_1), ..., H_{l-1,l}(\mathbf{x}_{l-1})]), \quad (24)$$

Acknowledgments

First, I would like to thank my supervisors, Decebal Mocanu, Aleksandra Nowak, and Bram Grooten. Their invaluable feedback and suggestions were instrumental throughout the creation of this thesis. I extend my appreciation to my fellow student, Adjorn van Engelenhoven, whose constant encouragement and sparring over ideas played a significant role in shaping this work. Finally, I would like to thank Emiel Steerneman for introducing me to the subject of sparse neural networks through exploratory talks, leading to the eventual creation of this work.

References

[1] Abien Fred Agarap. 2018. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375* (2018).

[2] Nir Ailon, Omer Leibovitch, and Vineet Nair. 2021. Sparse linear networks with a fixed butterfly structure: theory and practice. In *Uncertainty in Artificial Intelligence*. PMLR, 1174–1184.

[3] Matthias Bastian. 2023. GPT-4 has more than a trillion parameters - Report. https://the-decoder.com/gpt-4-has-a-trillion-parameters/. Accessed: 2023-09-06.

[4] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5, 2 (1994), 157–166.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[6] Luisa De Vivo, Michele Bellesi, William Marshall, Eric A Bushong, Mark H Ellisman, Giulio Tononi, and Chiara Cirelli. 2017. Ultrastructural evidence for synaptic scaling across the wake/sleep cycle. *Science* 355, 6324 (2017), 507–510.

[7] Tim Dettmers and Luke Zettlemoyer. 2019. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840* (2019).

[8] Graham H Diering, Raja S Nirujogi, Richard H Roth, Paul F Worley, Akhilesh Pandey, and Richard L Huganir. 2017. Homer1a drives homeostatic scaling-down of excitatory synapses during sleep. *Science* 355, 6324 (2017), 511–515.

[9] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. 2020. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*. PMLR, 2943–2952.

[10] Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635* (2018).

[11] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 249–256.

[12] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. 2022. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in Neural Information Processing Systems* 35 (2022), 507–520.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[14] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. 2012. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on* 14, 8 (2012), 2.

[15] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[16] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

[17] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. 2016. Deep networks with stochastic depth. In *European conference on computer vision*. Springer, 646–661.

[18] Ajay Jaiswal, Haoyu Ma, Tianlong Chen, Ying Ding, and Zhangyang Wang. 2022. Training Your Sparse Neural Network Better with Any Mask. https://doi.org/10.48550/ARXIV.2206.12755

[19] Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. 2020. Top-kast: Top-k always sparse training. *Advances in Neural Information Processing Systems* 33 (2020), 20744–20754.

[20] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[21] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. [n. d.]. CIFAR-10/100 (Canadian Institute for Advanced Research). ([n. d.]). http:

//www.cs.toronto.edu/~kriz/cifar.html
[22] Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. *Advances in neural information processing systems* 2 (1989).
[23] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. 2015. Deeply-supervised nets. In *Artificial intelligence and statistics*. PMLR, 562–570.
[24] Junjie Liu, Zhe Xu, Runbin Shi, Ray CC Cheung, and Hayden KH So. 2020. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. *arXiv preprint arXiv:2005.06870* (2020).
[25] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Zahra Atashgahi, Lu Yin, Huanyu Kou, Li Shen, Mykola Pechenizkiy, Zhangyang Wang, and Decebal Constantin Mocanu. 2021. Sparse training via boosting pruning plasticity with neuroregeneration. *Advances in Neural Information Processing Systems* 34 (2021), 9908–9922.
[26] Shiwei Liu, Decebal Constantin Mocanu, Amarsagar Reddy Ramapuram Matavalam, Yulong Pei, and Mykola Pechenizkiy. 2021. Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. *Neural Computing and Applications* 33 (2021), 2589–2604.
[27] Shiwei Liu, Decebal Constantin Mocanu, Yulong Pei, and Mykola Pechenizkiy. 2021. Selfish sparse rnn training. In *International Conference on Machine Learning*. PMLR, 6893–6904.
[28] Shiwei Liu, Lu Yin, Decebal Constantin Mocanu, and Mykola Pechenizkiy. 2021. Do we actually need dense over-parameterization? in-time over-parameterization in sparse training. In *International Conference on Machine Learning*. PMLR, 6989–7000.
[29] Diganta Misra. 2019. Mish: A self regularized non-monotonic activation function. *arXiv preprint arXiv:1908.08681* (2019).
[30] Decebal Constantin Mocanu et al. 2017. *Network computations in artificial intelligence*. Technische Universiteit Eindhoven.
[31] Decebal Constantin Mocanu, Elena Mocanu, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. 2016. A topological insight into restricted boltzmann machines. *Machine Learning* 104, 2 (2016), 243–270.
[32] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. 2018. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications* 9, 1 (2018), 1–12.
[33] Hesham Mostafa and Xin Wang. 2019. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*. PMLR, 4646–4655.
[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
[35] Ameya Prabhu, Girish Varma, and Anoop Namboodiri. 2018. Deep expander networks: Efficient deep networks from graph theory. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 20–35.
[36] Prajit Ramachandran, Barret Zoph, and Quoc V Le. 2017. Searching for activation functions. *arXiv preprint arXiv:1710.05941* (2017).
[37] Brian D Ripley. 2007. *Pattern recognition and neural networks*. Cambridge university press.
[38] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[39] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway networks. *arXiv preprint arXiv:1505.00387* (2015).
[40] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Training very deep networks. *Advances in neural information processing systems* 28 (2015).
[41] Arvind Subramaniam and Avinash Sharma. 2020. N2NSkip: Learning Highly Sparse Networks using Neuron-to-Neuron Skip Connections.. In *BMVC*.
[42] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.
[43] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. 2020. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in neural information processing systems* 33 (2020), 6377–6389.
[44] Hind Taud and JF Mas. 2018. Multilayer perceptron (MLP). In *Geomatic approaches for modeling land change scenarios*. Springer, 451–455.
[45] Kale-ab Tessera, Sara Hooker, and Benjamin Rosman. 2021. Keep the gradients flowing: Using gradient flow to study sparse network optimization. *arXiv preprint arXiv:2102.01670* (2021).
[46] William N Venables and Brian D Ripley. 2013. *Modern applied statistics with S-PLUS*. Springer Science & Business Media.

A CNN Architectures

Layers	Output Size	CNN/SPaS-CNN
Convolution	32x32x8	3 x 3 conv, stride 1
Pooling	16x16x8	3x3 max pool, stride 2
SPaS Block (1)	16x16x8	[3 x 3 conv] x 3
Transition Block (1)	16x16x16	1x1 conv
	8x8x16	2x2 avg pool, stride 2
SPaS Block (2)	8x8x16	[3 x 3 conv] x 3
Transition Block (2)	8x8x32	1x1 conv
	4x4x32	2x2 avg pool, stride 2
SPaS Block (3)	4x4x32	[3 x 3 conv] x 3
Classification Layer	1x1x32	4x4 global average pool, stride 4
	10/100	linear, softmax

(a) *CNN/SPaS-CNN*

Layers	Output Size	DenseNet
Convolution	32x32x8	3 x 3 conv, stride 1
Pooling	16x16x8	3x3 max pool, stride 2
Dense Block (1)	16x16x40	[3 x 3 conv] x 4
Transition Block (1)	16x16x20	1x1 conv
	8x8x20	2x2 avg pool, stride 2
Dense Block (2)	8x8x100	[3 x 3 conv] x 4
Transition Block (2)	8x8x50	1x1 conv
	4x4x50	2x2 avg pool, stride 2
Dense Block (3)	4x4x250	[3 x 3 conv] x 4
Classification Layer	1x1x250	4x4 global average pool, stride 4
	10/100	linear, softmax

(b) *DenseNet*

Table 6. *High-level overview of the CNN, SPaS-CNN, and DenseNet architectures for the experiments on CIFAR-10 and CIFAR-100*

B Parameter budget experiments

		MLP		SPaS-MLP		CNN		DenseNet		SPaS-CNN	
Sparsity Mode	$\|\theta\|$	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max
Static	1k	-	-	**14.02**±1.89	**16.18**	-	-	-	-	-	-
	2k	-	-	**20.95**±1.62	22.77	14.06±6.33	**24.35**	11.53±3.36	17.53	-	-
	5k	-	-	31.35±1.32	32.55	**41.53**±11.25	**51.03**	37.46±6.53	43.83	31.17±8.80	40.49
	15k	-	-	42.98±0.95	43.88	**67.07**±1.56	**69.22**	52.78±13.00	62.20	61.16±2.67	64.28
	50k	49.60±0.48	50.11	48.97±0.37	49.38	**74.57**±1.41	**76.13**	70.53±0.96	71.87	73.52±1.82	75.71
SET*/SPaSET	1k	-	-	**36.09**±0.63	**36.79**	-	-	-	-	-	-
	2k	-	-	45.99±0.90	46.92	**46.21**±18.7	**61.17**	40.17±18.05	58.36	14.97±11.1	34.73
	5k	-	-	48.60±0.40	49.20	**65.63**±5.39	**70.28**	39.58±22.71	64.38	63.96±11.6	70.23
	15k	-	-	49.52±0.68	50.20	73.51±1.85	75.39	73.85±1.46	74.79	**75.16**±1.03	**76.00**
	50k	42.38±18.05	51.08	49.74±0.56	50.33	74.90±2.27	77.35	76.24±1.36	77.66	**77.36**±0.56	**78.21**
N/A	Fully dense	47.89±0.88	49.26	39.82±0.94	41.17	73.73±1.66	75.22	73.26±7.02	77.49	**78.06**±0.72	**78.64**

Table 7. Experimental results of networks with *static/dynamic* sparsity and a fully dense baseline on CIFAR-10 by parameter budget $|\theta|$. Performances highlighted in **bold** were the best performing for that parameter budget. Results with a - did not achieve a non-random performance. For each model, the left column indicates the mean performance of the five experiments (with standard deviation), and the right column is the validation accuracy of the best-performing model from those five experiments. We observe that SPaS-CNN outperforms its counterparts down to 15k parameters. SPaS-MLP is able to uphold non-random performance down to 1k parameters as opposed to 50k in plain MLPs.

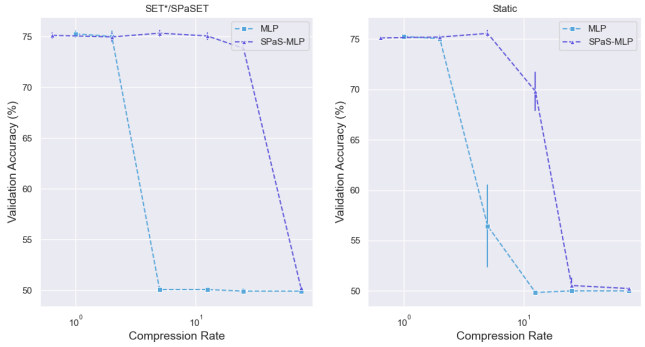
		MLP		SPaS-MLP		CNN		DenseNet		SPaS-CNN	
Sparsity Mode	$\|\theta\|$	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max
Static	1k	-	-	-	-	-	-	-	-	-	-
	2k	-	-	**3.17**±0.33	3.65	2.23±1.64	**4.15**	-	-	-	-
	5k	-	-	7.19±0.40	7.73	**15.87**±2.48	**18.62**	10.48±2.56	13.45	7.88±2.93	13.03
	15k	-	-	15.19±0.66	15.79	**30.46**±0.68	**30.96**	18.19±11.36	27.19	22.59±7.06	28.51
	50k	-	-	20.05±0.64	20.58	39.05±1.75	40.70	38.66±0.41	39.27	**40.31**±0.64	**41.13**
SET*/SPaSET	1k	-	-	-	-	-	-	-	-	-	-
	2k	-	-	**10.51**±0.65	**11.55**	-	-	-	-	-	-
	5k	-	-	19.35±0.17	19.63	26.26±2.14	29.17	**27.94**±1.78	30.22	27.90±2.28	**30.31**
	15k	-	-	23.12±0.40	23.67	38.30±1.45	40.16	36.68±3.38	39.71	**41.31**±1.10	**42.66**
	50k	-	-	24.25±0.37	24.58	44.21±0.61	45.09	45.35±1.12	46.64	**47.23**±0.68	**48.00**
N/A	Fully dense	18.31±0.63	19.19	13.12±0.57	13.77	38.70±2.5	41.19	43.11±2.00	45.76	**46.31**±0.97	**47.64**

Table 8. Experimental results of networks with *static/dynamic* sparsity and a fully dense baseline on CIFAR-100 by parameter budget $|\theta|$. Performances highlighted in **bold** were the best performing for that parameter budget. Results with a - did not achieve a non-random performance. For each model, the left column indicates the mean performance of the five experiments (with standard deviation), and the right column is the validation accuracy of the best-performing model from those five experiments. We observe that SPaS-CNN outperforms its counterparts down to 15k parameters. SPaS-MLP is able to uphold non-random performance down to 2k parameters while plain MLPs are unable to achieve a non-random performance with as much as 50k parameters.

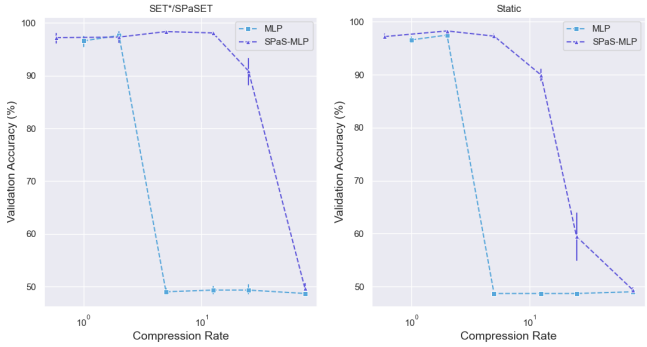
		MLP		SPaS-MLP	
Sparsity Mode	$\|\theta\|$	Mean	Max	Mean	Max
Static	1k	-	-	**53.27**±0.43	**53.87**
	2k	-	-	**65.30**±2.08	**67.70**
	5k	56.93±8.85	72.76	**71.20**±0.24	**71.38**
	15k	**72.77**±0.27	**73.11**	71.78±0.35	72.23
SET*/SPaSET	1k	-	-	**66.08**±2.74	**69.59**
	2k	-	-	**68.99**±1.40	**69.96**
	5k	-	-	**71.12**±0.35	**71.55**
	15k	**72.94**±0.35	**73.41**	71.69±0.29	72.07
N/A	Fully dense	**72.83**±0.43	**73.15**	71.08±1.04	72.11

Table 9. Experimental results of MLP and SPaS-MLP with *static/dynamic* sparsity and a fully dense baseline on the Higgs dataset by parameter budget $|\theta|$. Performances highlighted in **bold** were the best performing for that parameter budget. Results with a - did not achieve a non-random performance. For each model, the left column indicates the mean performance of the five experiments (with standard deviation), and the right column is the validation accuracy of the best-performing model from those five experiments. We observe that SPaS-MLP is able to uphold a non-random performance down to 1k parameters as opposed to 5k with plain MLPs.

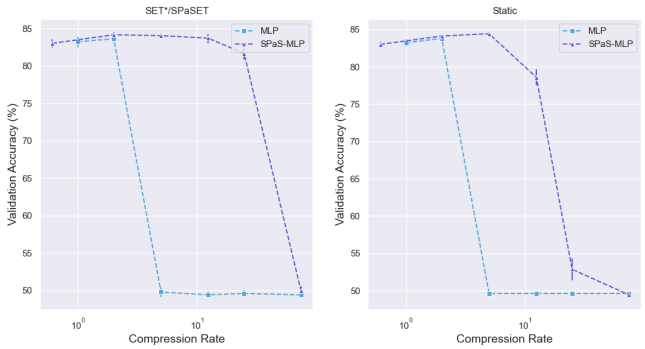
C Results from Tabular Benchmark



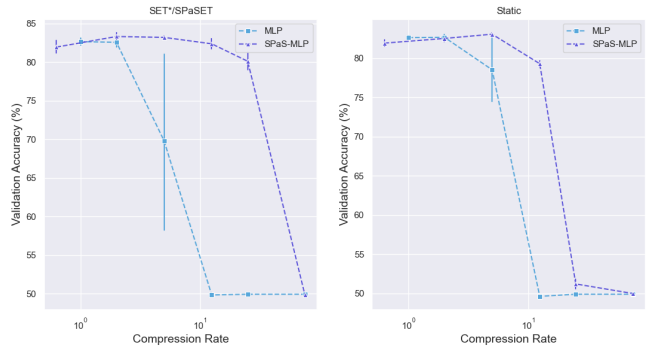
(a) electricity



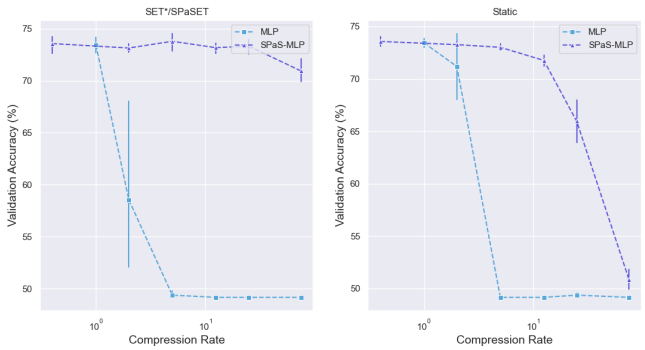
(b) pol



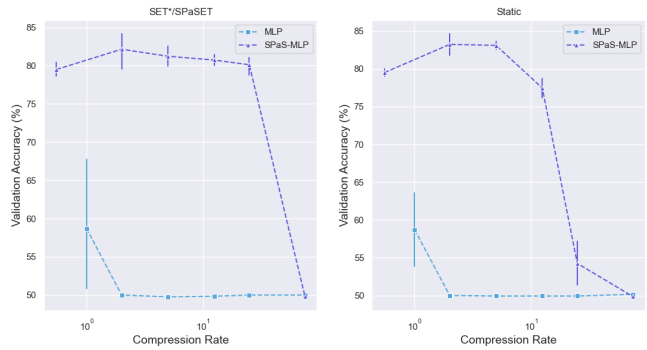
(c) house_16H



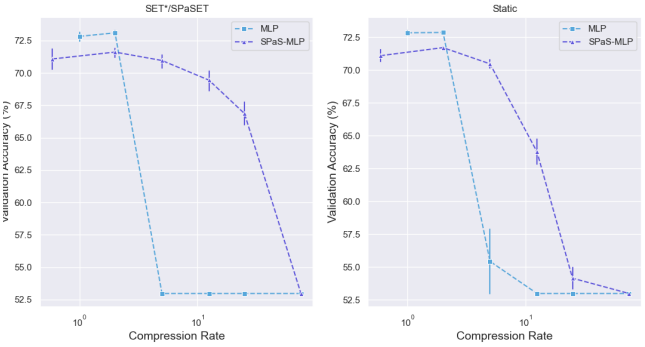
(d) MagicTelescope



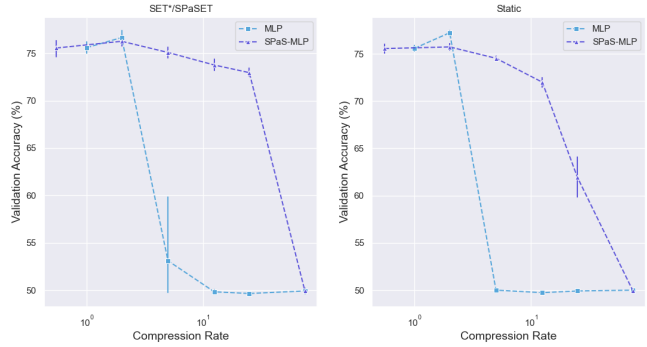
(e) Bioresponse



(f) MiniBooNE



(g) Higgs



(h) jannis

Figure 7. Results from Figure 4 isolated into the results of the individual datasets. Validation accuracy (%) IQM with standard error on the y-axis by compression rate (1x to 75x) on the x-axis.