Learning Network Size while Training with ShrinkNets

Anonymous Authors[1]

Abstract

Let's write the abstract at the end

1. Introduction

As neural networks become increasingly widely deployed in a variety of applications – ranging from improving camera quality on mobile phones (**?**) to language translation (**?**) to text auto-completion (**?**) – and on diverse hardware architectures – from laptops to phones to embedded sensors – inference performance and model size are becoming equally as important as traditional measures of prediction quality. However, these three aspect of model performance – quality, performance and size – are largely optimized separately today, often with suboptimal results.

Of course, the problem of finding compressed or small networks is not new. Existing techniques typically aim make a pre-trained neural network smaller (**??**) by taking one of two approaches: either by applying quantitazation (**?**) or code compilation (**?**) techniques that can be applied blindly to any network, or they analyze the structure of the network and systematically prune connections (Han et al., 2015; Cun et al., 1990) or neurons (**?**) based on some loss function. Although these techniques can substantially reduce model size, they have several drawbacks. First, they often negatively impact model quality. Second, they can (surprisingly) negatively impact inference time as they transform dense matrix operations into sparse ones, which can be substantially slower to execute on modern hardware (**?**), especially GPUs which do not efficiently support sparse linear algebra. Third, these techniques generally start by optimizing particular architecture for prediction performance, and then, as a post- processing step, apply compression to generate a smaller model that meets the resource constraints of the deployment setting. Because the network architecture is essentially fixed during compression, model architectures that work better in small settings may be missed – this is especially true in large networks like many- layered CNNs, where it is infeasible to try explore even a small fraction of possible network configurations.

In contrast, in this paper we present a new method to simultaneously optimize both network size and model performance. The key idea is to learn the right network size at the same time that we optimize for prediction performance. Our approach, called *ShrinkNets*, starts with an *oversized* network, and dynamically shrinks it by eliminating neurons during training time. Our approach has two main benefits. First, we explore the architecture of models that are both small and perform well, rather than starting with a high- performance model and making it small. This allows us to efficiently generate a family of smaller and accurate models without an exhaustive and expensive hyperparameter search over the number of neurons in each layer. Second, in contrast to existing neural network compression techniques (Aghasi et al., 2016; Han et al., 2015), our approach results in models that are both not only small, but where the layers are dense, which means that inference time is also improved, e.g., on GPUs.

In summary, our contributions are as follows:

1. We propose a novel technique based on dynamically switching on and off neurons, which allows us to optimize the network size as the network is trained.
2. We show that our technique is a relaxation of group LASSO (Yuan & Lin, 2006) and prove that our problem admit many global minima.
3. **Sam:** Some claim about model size vs performance
4. We demonstrate the efficacy of our technique on both convolutional and fully-connected neural nets, showing that ShrinkNets finds networks within +/-X% of best hand-crafted accuracy in XX% of training time compared to existing hyperparameter optimization methods.
5. We also demonstrate that ShrinkNets can achieve this accuracy with only YY% of neurons.
6. **Sam:** Something about dense networks and then benefit over optimal brain damage
7. **Sam:** Some claim about inference time
8. **Sam:** Some claim about compatibility with existing compression techniques?

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

2. The ShrinkNets Approach

In this section we present the *Switch layer*, a core concept of our ShrinkNets approach that allows to learn the network size as part of the training process. Then, we explain how to adapt the training procedure to support the new layer. We begin with a high-level overview of our method.

2.1. Overview

At a high-level, the ShrinkNets approach consists of two interconnected stages. The first stage is concerned with identifying neurons that are not contributing to increasing the prediction accuracy of the network and deacivate them. The second stage is concerned with actually removing the neurons from the network and shrinking its size, therefore leading to faster inference times. We give an overview of both processes next:

Deactivating Neurons On-The-Fly: During the first stage, ShrinkNets applies an on/off switch to every neuron of an initially oversized network. We model the on/off switches by multiplying each input (or output) of each layer by a parameter $\boldsymbol{\theta}$, with values 0 or 1. A value of 0 will deactivate the neuron, while 1 will let the signal go through. These switches are part of a new layer, which we call **Switch layer** which can be applied to fully connected as well as convolutional layers.

We want to minimize the number of on switches to reduce the model size as much as we can, while preserving prediction accuracy. This can be achieved by jointly minimizing the objective of the network and a factor of the L0 norm of the vector containing all the on/off switches. Because finding an optimal binary assignement is an NP-Hard problem, we allow $\boldsymbol{\theta}$ to be a real number instead of a 0/1 value, thus constraining the L1 norm as opposed to the L0 norm.

Neuron Removal: During this stage, the neurons that are deactivated by the Switch layers are actually removed from the network, effectively shrinking the network size. This is what leads to faster inference times, as we demonstrate in our evaluation. We choose to remove neurons at training time because we have observed that allows the active neurons to adapt to the new network architecture. Existing techniques focus on neuron removal after training, and require an extra fine-tuning process to compensate for the removal.

In the remainder of this section we describe in detail the Switch layer as well as how to adapt the training process for ShrinkNets.

2.2. The Switch Layer

Let L be a layer in a neural network that takes an input tensor $\boldsymbol{x}$ and produces an output tensor y of shape $(c \times d_1 \times \cdots \times d_n)$ where c is the number of neurons in that layer. For instance, for fully connected layers, n=0 and the output is single dimensional of size c (ignoring batch size for now) while for a 2-D convolutional layer, n=2 and c is the number of output channels or feature maps.

Suppose we wish to tune the size of L by applying a switch layer. A switch layer S applied to the output of L can be parametrized by a vector $\theta \in \mathbb{R}^c$ such that the result of applying S to $L(\boldsymbol{x})$ is a tensor also of size $(c \times d_1 \times \cdots \times d_n)$ such that:

$$S_\theta(L(\boldsymbol{x}))_{i,\dots} = \boldsymbol{\theta}_i L(\boldsymbol{x})_{i,\dots} \tag{1}$$

Effectively, once passed through the switch layer, each output channel i produced by L is scaled by the corresponding θ_i. If for any k, if $\theta_i = 0$, the i^{th} channel is multiplied by zero and wont contribute to any computations after the switch layer. If this happens, we say the Switch layer has *deactivated* or killed the neuron of layer L corresponding to channel i.

2.3. Training ShrinkNets

To tune the size of a network, we place Switch Layers after each layer whose size we wish to tune; these are typically the fully connected and convolutional layers in a network. Since the switch layers are adept at deactivating neurons, we start with an oversized network (i.e. network with more capacity than required) and then use switch layers to deactivate or kill off neurons that are unnecessary.

Formally, we can express this procedure in terms of a sparsity constraint that pushes values in the θ vector to 0. Given a neural network parameterized by weights $\boldsymbol{W}$ and switch layer parameters $\boldsymbol{\theta}$, we optimize the following ShrinkNet loss that augments the regular training loss with a regularization term for the switch parameters and another on the network weights.

$$L_{SN}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{W}, \boldsymbol{\theta}) = L(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{W}) + \lambda \left\|\theta\right\|_1 + \lambda_2 \left\|\boldsymbol{W}\right\|_p \tag{2}$$

Relation to Group Sparsity (LASSO)

Our goal when designing ShrinkNets was to be able to remove inputs and outputs of layers. For classic fully connected layers, which are defined as :

$$f_{\boldsymbol{A},\boldsymbol{b}}(\boldsymbol{x}) = a(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}) \tag{3}$$

removing an input neuron j is equivalent to have $\left(\boldsymbol{A}^T\right)_j = \boldsymbol{0}$ and removing an output neuron i is the same as having $\boldsymbol{A}_i = \boldsymbol{0}$ and $\boldsymbol{b}_i = 0$. Solving optimization problems while trying to set entire groups of parameters to zero has been already studied and the most popular method is without doubt the group sparsity regularization [ref]. For any partition-

ning of the set of parameter defining a model in p groups: $\boldsymbol{\theta} = \bigcup_{i=1}^{P} \boldsymbol{\theta}_i$ we define it the following way:

$$\Omega_\lambda^{gp} = \lambda \sum_{i=1}^{p} \sqrt{\#\boldsymbol{\theta_i}} \left\|\boldsymbol{\theta_i}\right\|_2 \tag{4}$$

In the context of a fully-connected layer, the groups are either: columns of $\boldsymbol{A}$ if we want to remove inputs, or rows of $\boldsymbol{A}$ and the corresponding entry in $\boldsymbol{b}$ if we want to remove outputs. For simplicity, we will focus our analysis in the simple one-layer case. In this case filtering outputs does not make a lot of sense, this is why we will only consider the former case. The group sparsity regularization then becomes:

$$\Omega_\lambda^{gp} = \lambda \sum_{j=1}^{p} \left\| \left(\boldsymbol{A^T}\right)_{\boldsymbol{j}} \right\|_2 \tag{5}$$

Because $\forall i, \#\boldsymbol{\theta}_i = n$, To make the notation simpler, we embedded $\sqrt{n}$ inside λ.

Since group sparsity and ShrinkNets try to achieve the same goal we will try to understand their similarities and differences. First let's recall the two problems. The original ShrinkNet problem is:

$$\min_{\boldsymbol{A},\boldsymbol{\beta}} \left\|\boldsymbol{y} - \boldsymbol{A}\text{diag}\left(\boldsymbol{\beta}\right)\boldsymbol{x}\right\|_2^2 + \lambda \left\|\boldsymbol{\beta}\right\|_1 \tag{6}$$

And the Group Sparsity problem is:

$$\min_{\boldsymbol{A}} \left\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\right\|_2^2 + \Omega_\lambda^{gp} \tag{7}$$

We can prove the under the condition: $\forall j \in [\![1, p]\!], \left\| \left(\boldsymbol{A}^T\right)_j \right\|_2 = 1$ the two problems are equivalent (proposition A.1). However if we relax this constraint then shrinknet becomes non-convex and has no global minimum (propositions A.2 and A.3). Fortunately, by adding an extra term to the ShrinkNet regularization term we can proove that:

$$\min_{\boldsymbol{A},\boldsymbol{\beta}} \left\|\boldsymbol{y} - \boldsymbol{A}\text{diag}\left(\boldsymbol{\beta}\right)\boldsymbol{x}\right\|_2^2 + \Omega_\lambda^s + \lambda_2 \left\|A\right\|_p^p \tag{8}$$

has many global minimum (proposition A.4) for all $p > 0$. This is the reason we defined the *regularized ShrinkNet penalty* earlier this way:

$$\Omega_{\lambda,\lambda_2,p}^{rs} = \lambda \left\|\boldsymbol{\beta}\right\|_1 + \lambda_2 \left\|\boldsymbol{\theta}\right\|_p^p \tag{9}$$

In practice we observed that $p = 2$ or $p = 1$ are good choice, while the latter will also introduce additional sparsity in the parameters.

3. ShrinkNets in Practice

3.1. Neuron killing

Once Switch layers are placed in a network and initialized (sampled from the $\mathcal{N}(0, 1)$ distribution),

Threshold strategy: We kill neurons based on a threshold (in absolute value), this is the method used by deep compression. It is bad because it is scale dependant and is not robust to noise in the gradients (explain that)

Sign change strategy: When using this strategy we consider a neuron dead when it changes its sign. This strategy works well in practice but is sensible to noise. If we sample a gradient that is not representative of the dataset when performing gradient descent then we might wrongly kill a neuron.

Sign variance strategy: We measure the exponnential moving variance of the sign $(-1, 1)$ of each component in the *Switch Layer*, and consider it dead when it goes over a predefined threshold. Thus, this strategy introduce two extra parameters, one that controls the behavior of the innertia of the variance, and the other is the threshold.

According to our experiments on linear and logistic regressions the two last strategies reduce the number of parameter update performed during training by at least an order of magnitude with only minimal impact on the final error obtained by the model.

3.2. Implementation

We have implemented **Switch Layers** and the associated training procedure as a library in pytorch (Paszke et al., 2017). The layer can be freely mixed with other popular layers such as convolutional layers, batchnorm layers, fully connected layers etc. There are two key challenges that must be overcome while implementing **Switch Layers**. First, since ShrinkNets kill off a large fraction of neurons, we must remove the overhead of carrying around dead neurons. And second, we must update the various network layers and training machinery (e.g. optimizer state) to reflect the deactivation of neurons. We address these challenges by augmenting every layer with additional information about neuron status and implementing a *neural garbage collection* mechanism which prunes deactivated neurons on-the-fly and updates the state of the training machinery.

A second implementation challenge we address with ShrinkNets is the potentially higher inference time. This increase in inference time is both due to the extra computation introduced by switch layers and because our switch layers are not currently optimized for CUDA. Therefore, for now, our library provides a post-training routine to "fold"

switch layers into existing, optimized layers. Specifically, for every switch layer, we check if the parent or child of this layer is a linearly scalable layer (e.g. convolutional layer, fully connected layer, batchnormalization layer) and if so, we fold the switch layer into this neighboring layer by multiplying through by the switch weights. In the unlikely case that a switch layer is sandwiched between two non-linearly scalable layers, we leave the switch layer as is.

4. Evaluation

The goal of our evaluation is to show two main properties of *ShrinkNets*: they are exploring the space of possible sizes in a "smart" way, and that as a result we obtain the smaller or equal size than *Static Networks*. We will then evaluate the performance benefits implied by these gains in size.

4.1. Datasets and Setup

4.1.1. CIFAR10

Previous attempt at solving this problem focus on simple datasets and simple architectures (limited to Fully connected). We believe that self-resizing network really matter for problems where the architecture you need to solve them involve so many layers that it is impractical to explore the space of possible sizes. We considered ImageNet (Russakovsky et al., 2015) but the time required to train models is so long that it would have not been possible to train enough and get statistically significant results. This is why we picked CIFAR10 (Krizhevsky, 2009). It is an image classification dataset containing 60000 color images ($3 \times 32 \times 32$), belonging to 10 different classes.

To solve this task we use the VGG16 model (Srivastava et al., 2014). It is constituted of alternating convolutional layers and *MaxPool* layers interleaved by *BatchNorm* (Ioffe & Szegedy, 2015) and *ReLU* (Nair & Hinton, 2010) layers. The two last layers are Fully connected layers separated by just a *ReLU* activation function.

To turn it into a ShrinkNet we introduce *Switch Layers* after each *BatchNorm* and each Fully connected (except the last one).

ShrinkNets assume that the starting size of the network is an upper bound on the optimal size. We though that picking two times the recommended size for each layer (that was designed for ImageNet), is a generous upper bound. For the classification layers we use 5000 neurons as an upper bound where the ImageNet version uses 4096 **GI:** This is on the top of my head, need to be double checked.

We assume no prior knowledge on the optimal batch size, learning rate, λ or weight decay (λ_2). This is why, we randomly sample them from a range of reasonable values (**GI:** should we make them explicit ?). Training is done using our library, based on PyTorch (Paszke et al., 2017), that support dynamically resized layers. We train the network using gradient descent and *Adam* optimizer (Kingma & Ba, 2014). We start with the learning rate sampled randomly and every 5 epochs of non improvement in validation accuracy we divide the learning rate by 10. We stop training after 400 epochs or when the learning rate is under 10^{-7} whichever comes first. **GI:** Should I also give the details about the removal strategy we used and the γ and threshold I used ? because this is clearly getting boring For each of the models we trained, we pick the epoch with the best validation accuracy and report the corresponding testing accuracy. Because of the nature of our method, it can happen that for network that are aggressively compressed, the best validation accuracy is obtained early in the training, before the size has converged. To be sure that accuracy measured corresponds to the final shape and not the starting shape, we only consider the second half of the training when picking the best epoch. For each model, we also measure the total size, in number of floating point parameters, excluding the *Switch Layers* because as we saw in section 3.1, we can get rid of them when training is done.

We want to compare against classical (*Static*) networks. The number of parameters that control the size is large: 13 for the size of convolutional layers and 2 for the fully connected ones. Without Shrinknets to help and fuse all these parameters in a single λ it is infeasible to sample reasonably well a search space of that size. This is why we have to rely on the very well known heuristic that the original VGG architecture (and many CNNs) **GI:** try to find the paper that introduces this heuristic. For *Static Networks* we sample the size between 0.1 and 2 times the size optimized for ImageNet. We report the same numbers as we did for *ShrinkNets* and we compare the two distributions of models we obtain on the first plot of fig. 1.

4.1.2. COVERTYPE DATASET

Our second dataset we will evaluate on is COVERTYPE (Blackard, 1998). This dataset contains 581012 description of geographical area (elevation, inclination, etc...) and the goal is to predict the type of forest growing there. We picked this one for couple of reasons. First it is simple that we can reach good accuracy with only a few layers. This important because we want to show that *ShrinkNets* find sizes as good as *Static Networks*, even if we are sampling the entire space of possible sizes. The second reason is that [ref] also perform their evaluation on this dataset, which allow us to compare the results obtained by the two methods.

The only difference in terms of setup are minimal. We use the same architecture as (Scardapane et al., 2017), i.e. three layers Fully Connected neural network with no *Dropout* (Srivastava et al., 2014) nor *BatchNorm*. **GI:** Should we say

here that we don't expect Dropout to work here ? I could write an entire paragraph about it if needed. In this case for the *Static Networks*, we sample independently sizes of the three different layers to explore all possible architectures.

4.2. Results of the random search procedure

On the plot, the Pareto front for the two training methods clearly indicates that not only *ShrinkNets* find models that are as accurate as *Static Networks*, but they are more efficient in terms of size. Even when we sample the entire size space (in the `COVERTYPE` case), *ShrinkNets* are better or equally as good as *Static Networks*. it has two implications. First it reinforces, our observation in [ref] that showed that the models do not suffer from the non-convexity introduced by the multiplication by the *Switch Layer*. Secondly it could let us conjecture that the parametrization from a λ (real value) to the size of the size of the network (vector) is optimal or close to optimal. Indeed, if it was not the case then eventually a randomly sampled size would have beaten *ShrinkNets* models. **GI:** Too strong ?

4.3. Benefits of smaller sizes

We showed in the previous section that *ShrinkNets* were able to find more or at least as efficient as *Static Networks*. In this experiment we want to determine benefits of the smaller size we get.

For some applications, the absolute best accuracy is not something desirable if it implies having enormous models. This observation motivates our methodology: for a given target accuracy that you want your model to achieve, we pick the smallest that satisfy the constraint. For both *ShrinkNets* and *Static Networks* we measure multiple metrics and compute the ratio between the two. The metrics are interested in are: the final model size (in number of parameters) and inference speed on CPU and GPU for small batch sizes (1 input) and large batch sizes (depend on the dataset and hardware). Results are shown on fig. 1 and fig. 2 after the distribution of sizes and accuracy. We limit our plot to the $(80 - 100\%$ accuracy range because we consider that models with lower accuracy have less practical use).

Regarding size, the key observation we can make is that size improvenets are significant for the `CIFAR10`. In the range of accuracies we are interested in, improvements in size go from 4x to 40x. On the `COVERTYPE` dataset even if the compression ratio is always above 1, it hardly exceeds 3x, except for very high accuracy where *ShrinkNets* obtained some particularly good solutions.

We observe that unlike local sparsity compression based method, our method translates directly improvement in size to higher bandwidth at inference time. In situations where inference is already incredibly fast (`COVERTYPE` with a batch size of 1 and `CIFAR10` with a batch size of 1 on GPU), improvements are dominted by noise. However, on machine learning frameworks more focused on latency it should be possible to measure them.

4.4. Architectures obtained after convergence

We sucessfully demonstrated that the architecture *Shrinknets* converge to make sense both in term of size and performance. And for a given accuracy the size needed is significantly smaller than when we use the classic heursitc we commonly use to size convolutional neural networks. It might be an indication that this strategy is suboptimal and might need to be adjusted. During our experimentations on simpler datasets like `MNIST` (LeCun et al., 2001) and `FashionMNIST` (Xiao et al., 2017) we observed that even if they have the same number of classes, input features, and output distribution. For a fixed λ *ShrinkNets* converged to considerably bigger networks in the case of `FashionMNIST`. This evidence shows that final architecture not only depends on the output distribution or shape of the data but actually reflects the dataset, indeed we know that `MNIST` is a much easier problem than `FashionMNIST`.

We provide two examples of architectures learned by *ShrinkNet*. One for the best test available test accuracy and a slightly underperforming but significantly smaller that it's equivalent *Static Network*. For these two architectures we show the size at different time during training to give a sense how the convergence happen.

5. Related Work

- Hyperparameter optimization: random, bayesian opt, bandit methods
- distillation techniques
- post-training compression techniques
- group sparsity, non-parametric neural networks
- training dynamics paper: first overfitting and then randomization?, **GI:** Here is the ref, if you can introduce it in the flow (Shwartz-Ziv & Tishby, 2017)

Given the importance of network structure, many techniques have been proposed to find the best network structure for a given learning task. These techniques broadly fall into four categories: hyperparameter optimization strategies, post-training model compression for inference as well as model simplification, techniques to resize models during training, and automated architecture search methods.

The most popular techniques for hyperparameter optimization include simple methods such as random search (Bergstra & Yoshua, 2012) which have been shown to

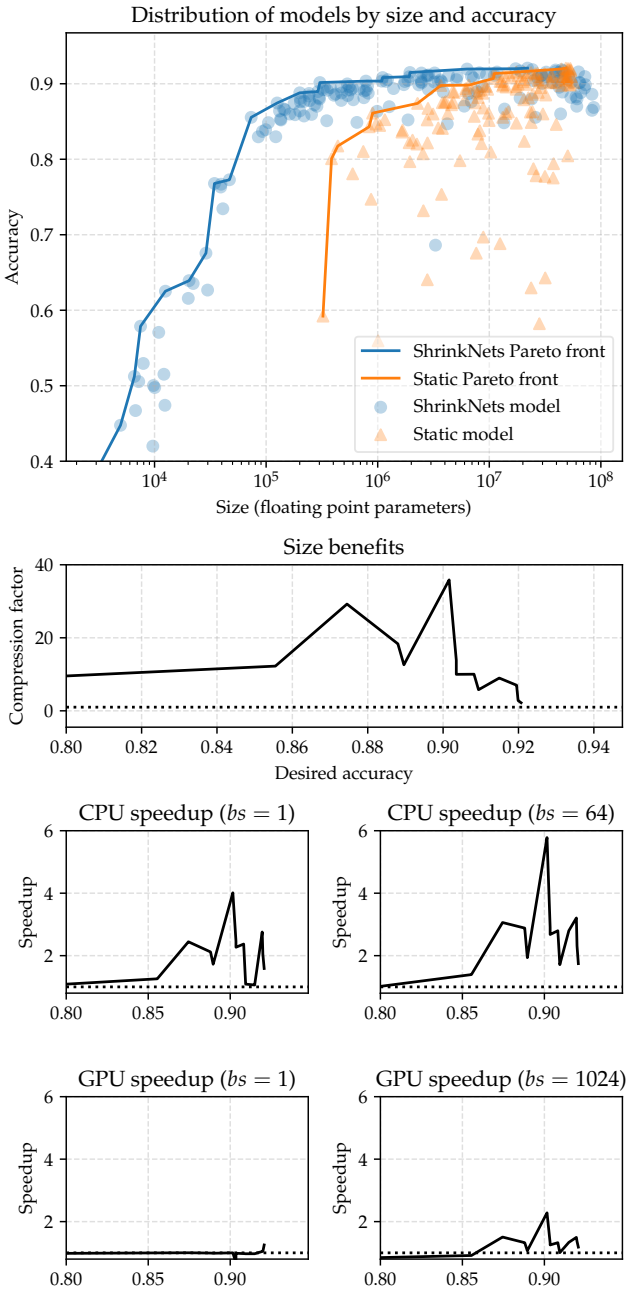

Figure 1. Summary of the result of random search over the hyper-parameters the `CIFAR10` dataset

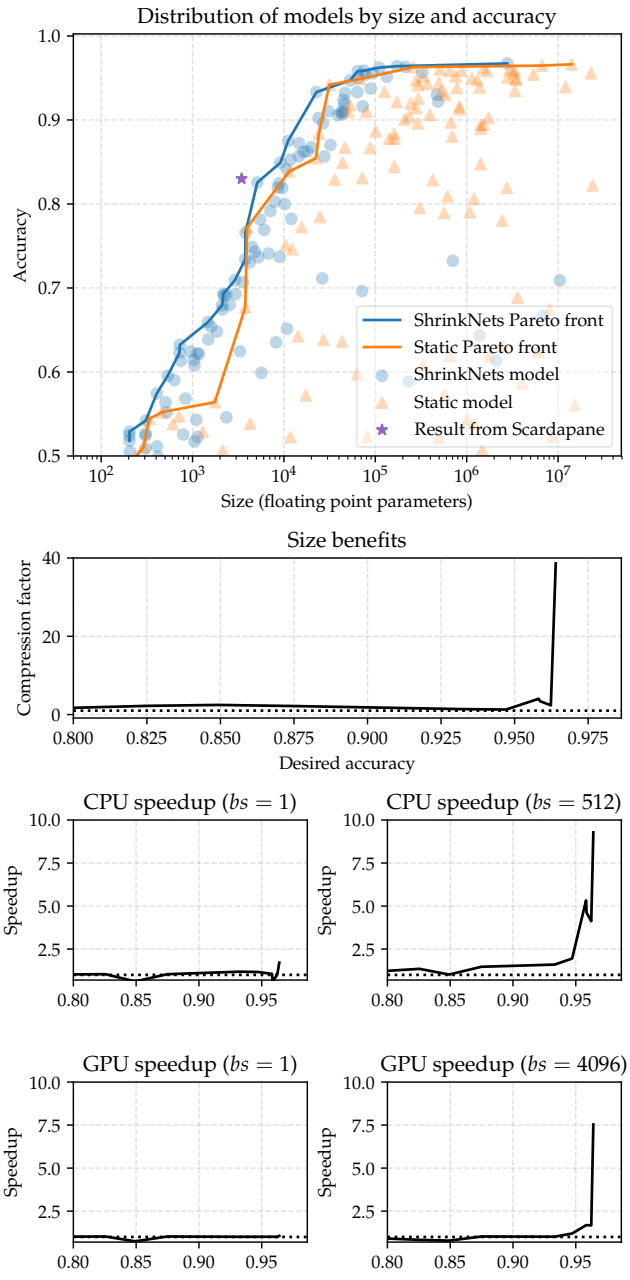

Figure 2. Summary of the result of random search over the hyper-parameters the `COVERTYPE` dataset

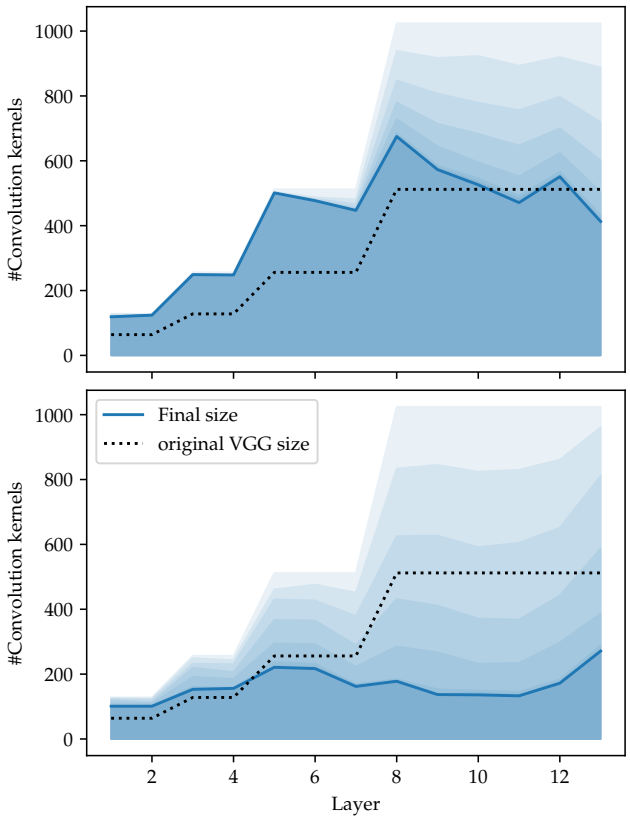

Figure 3. Evolution of the size of each layer over time (lighter: beginning, darker: end). On top a very large network performing 92.07%, at the bottom a simpler model with 90.5% accuracy

work suprisingly well compared to more complex methods such as those based on Bayesian optimization (Snoek et al., 2012). Techniques such as (Snoek et al., 2012) model generalization performance via Gaussian Processes (Rasmussen & Williams, 2006) and select hyperparameter combinations that come from uncertain areas of the hyperparameter space. Recently, methods based on bandit algorithms (e.g. (Li et al., 2016; Jamieson & Talwalkar, 2016)) have also become a popular way to tune hyperparameters. As noted before, all of the above techniques require many tens to hundreds of models to be trained, making this process computationally inefficient and slow.

In contrast with hyperparameter tuning methods, some methods such as DeepCompression (Han et al., 2015) seek to compress the network to make inference more efficient. This is accomplished by pruning connections and quantizing weights. On similar lines, multiple techniques such as (Romero et al., 2014; Hinton et al., 2015) have been proposed for distilling a network into a simpler network or a different model. Unlike our technique which works during training, these techiques are used after training and it would be interesting to apply them to ShrinkNets as well. (Abadi et al., 2016) share the common goal of removing entire blocks of parameter to maintain dense matrices, however their method only applies to convolutional layers.

The techniques closed to our work are those based on group sparsity such as (Scardapane et al., 2017), and those like (Philipp & Carbonell, 2017) who grow and shrink dynamically the size of the network during training.

Finally, there has also been recent work in automatically learning model architecture through the use of genetic algorithms and reinforcement learning techniques (Zoph & Le, 2016; Zoph et al., 2017). These techniques are focused more on learning higher-level architectures (e.g. building blocks for neural network architectures) as opposed to learning network size.

6. Discussion

Gl: Did not have time to write it but added some pointers

- Where does the method shine: We shine where there are too many layers To explore the network size and we have to rely on heuristics. It shines by it simplicity compared to other methods **Gl:** Do we want to talk about the number of lines of codes ?. Prunning while we train, from the compression results seem to be benificial to prune while training to achieve very good compression.
- Side-effects of method: smaller networks, time to train. There is not significant difference in training time, mostly because the filter layer is not optimized. How-

ever we can say the convergence speed is improved. For example for `CIFAR10` the mean number of epoch to reach the best is 62.5 for *ShrinkNets* vs 68.4 for normal networks. for `COVERTYPE` the numbers are respectively 117.9 and 130.

- Potential extensions/limitations: We could add a filter that learn the size of convolutions for convolution layers. Supporting Residual Network would allow us to support more architecture. We could also try to mix try to learn the number of layers such is described in (Meier)

7. Conclusion

A. Appendix

A.1. Proofs

Unless specified, all the proofs consider the Multi-Target linear regression problem

Proposition A.1. $\forall (n,p) \in \mathbb{N}_+^2, \boldsymbol{y} \in \mathbb{R}^n, \boldsymbol{x} \in \mathbb{R}^p \lambda \in \mathbb{R}$

$$\begin{aligned}
&\min_{\mathbf{A}} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \lambda \sum_{j=1}^{p} \left\| \left(A^T\right)_j \right\|_2 \\
=&\min_{\mathbf{A}',\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{A}' diag\left(\boldsymbol{\beta}\right)\mathbf{x}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1 \\
&s.t. \forall j \in [\![1,p]\!], \left\| \left(A'^T\right)_j \right\|_2^2 = 1
\end{aligned}$$

Proof. In order to prove this statement we will show that for any solution $\boldsymbol{A}$ in the first problem, there exists a solution in the second with the exact same value, and vice-versa. We now assume we have a potential solution $\boldsymbol{A}$ for the first problem and we define $\boldsymbol{\beta}$ such that $\boldsymbol{\beta}_j = \left\| \left(\boldsymbol{A}^T\right)_j \right\|_2^2$, and $\boldsymbol{A'} = \boldsymbol{A}\left(\text{diag}\left(\boldsymbol{\beta}\right)\right)^{-1}$. It is easy to see that the constraint on $\boldsymbol{A'}$ is statisfied by construction. Now:

$$\begin{aligned}
&\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \lambda \sum_{j=1}^{p} \left\| \left(A^T\right)_j \right\|_2 \\
=& \|\mathbf{y} - \mathbf{A}'\text{diag}\left(\boldsymbol{\beta}\right)\mathbf{x}\|_2^2 + \lambda \sum_{j=1}^{p} \left\| \left(A'^T\right)_j \beta_j \right\|_2 \\
=& \|\mathbf{y} - \mathbf{A}'\text{diag}\left(\boldsymbol{\beta}\right)\mathbf{x}\|_2^2 + \lambda \sum_{j=1}^{p} |\beta_j| \cdot 1 \\
=& \|\mathbf{y} - \mathbf{A}'\text{diag}\left(\boldsymbol{\beta}\right)\mathbf{x}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1
\end{aligned}$$

Assuming we take an $\boldsymbol{A'}$ that satisfy the constraint and a $\boldsymbol{\beta}$, we can define $\boldsymbol{A} = \boldsymbol{A'}\text{diag}\left(\boldsymbol{\beta}\right)$. We can apply the same operations in reverse order and obtain an instance of the first problem with the same value. We can now see that the two problems must have the same minimum otherwise we would be able to construct a solution to the other with exact same value. □

Proposition A.2.

$$\|\mathbf{y} - \mathbf{A} diag\left(\boldsymbol{\beta}\right)\mathbf{x}\|_2^2$$

is not convex in $\boldsymbol{A}$ and $\boldsymbol{\beta}$.

Proof. To prove this we will take the simplest instance of the problem: with only scalars. We have $f(a,\beta) = (y - a\beta x)^2$. For simplicty let's take $y =$ and $x > 0$. If we take two candidates $s_1 = (0,2)$ and $s_2 = (2,0)$, we have $f(s_1) = f(s_2) = 0$. However $f(\frac{2}{2},\frac{2}{2}) = x > \frac{1}{2}f(0,2) + \frac{1}{2}f(2,0)$, which break the convexity property. Since we showed that a particular case of the problem is non-convex then necessarly the general cannot be convex. □

Proposition A.3.

$$\min_{\mathbf{A},\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{A} diag\left(\boldsymbol{\beta}\right)\mathbf{x}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1$$

has no solution if $\lambda > 0$.

Proof. Let's assume this problem has a minimum $\boldsymbol{A}^*, \boldsymbol{\beta}^*$. Let's consider $2\boldsymbol{A}^*, \frac{1}{2}\boldsymbol{\beta}^*$. Trivially the first component of the sum is identical for the two solutions, however $\lambda \left\|\frac{1}{2}\boldsymbol{\beta}\right\| < \lambda \|\boldsymbol{\beta}\|$. Therefore $\boldsymbol{A}^*, \boldsymbol{\beta}^*$ cannot be the minimum. We conclude that this problem has no solution. □

Proposition A.4. *For this proposition we will not restrict ourselves to single layer but the composition of an an arbitrary large (n) layers as defined individually as $f_{\boldsymbol{A_i},\boldsymbol{\beta}_i,\boldsymbol{b}_i}(x) = a(\boldsymbol{A_i} diag\left(\boldsymbol{\beta_i}\right)\boldsymbol{x} + \boldsymbol{b_i})$. The entire network follows as: $N(\boldsymbol{x}) = \left(\bigcirc_{i=1}^{n} f_{\boldsymbol{A_i},\boldsymbol{\beta_i},\boldsymbol{b_i}}\right)(\boldsymbol{x})$. For $\lambda > 0$, $\lambda_2 > 0$ and $p > 0$ we have:*

$$\min \|\boldsymbol{y} - N(\boldsymbol{x})\|_2^2 + \Omega_{\lambda,\lambda_2,p}^{rs}$$

has at least 2^k global minimum where $k = \sum_{i=1}^{n} \#\boldsymbol{\beta_i}$

Proof. First let's prove that there is at least one minimum to this problem. The two components of the expression are always positive so we know that this problem is bounded by below by 0. Let's assume this function does not have a minimum. Then there is a sequence of parameters $(S_n)_{n>0}$ such that the function evaluated at that point convereges to the infimum of the problem. Since the function is defined everywhere does not have a minimum then this sequence must diverge. Since the entire sequence deverge the there is at least one individual parameter that diverges. First case, the parameter is a component k of some $\boldsymbol{\beta_i}$ for some i. Necessarly $\|\boldsymbol{\beta_i}\|_1$ diverge towards $+\infty$, which is incompatible with the fact that (S_n) converges to the infimum. We can have the exact same argument if the diverging parameter is

in $\boldsymbol{A_i}$ or $\boldsymbol{b_i}$ because $p > 0$. Since there is always a contradiction then our assumption that the function has no global minimum must be false. Therefore, this problem has at least one global minimum.

Let's consider one optimal solution of the problem. For each component k of $\boldsymbol{\beta_i}$ for some i. Negating it and negating the k^{th} column of $\boldsymbol{A_i}$ does not change the the first part of the objetive because the two factors cancel each other. The two norms do not change either because by definition the norm is independant of the sign. As a result these two sets of parameter have the same value and are both global minimum. It is easy to see that going from this global minimum we can decide to negate or not each element in each $\boldsymbol{\beta_i}$. We have a binary choice for each parameter, there are $k = \sum_{i=1}^{n} \#\boldsymbol{\beta_i}$ parameters, so we have at least 2^k global minima.

□

A.2. Multi-Target Linear and Multi-Class Logistic regressions experiments

As we showed, Group sparsity share similarities with our method, and we claim that ShrinkNets are a relaxation of group sparsity. In this experiment we want to compare the two aproaches. We decided to focus on multi-target linear regression because in the single target case, groups in the Group Sparsity problem would have a size of one ($\boldsymbol{A}$ would be a vector in this case).

The evaluation will be done on two datasets `scm1d` and `oes97` (Spyromitros-Xioufis et al., 2016) for linear regressions and we will use `gina_prior2` (Guyon et al., 2007) and the *Gas Sensor Array Drift Dataset* (Vergara et al., 2012) (that we shorten in `gsadd`) for logistic regressions.

For each dataset we fit with different regularization parameters and measure the error and sparsity obtained after convergence. In this context we define sparsity as the ratio of columns that have all their weight under 10^{-3} in absolute value. Regularization parameters were choosed in order to obtain the widest sparsity spectrum. Loss is normalized depending on the problem to be in the $[0, 1]$ range. We summarized the results in fig. 4. From our experiments it is clear that ShrinkNets can fit the data closer than Group Sparsity for the same amount of sparsity. The fact that we are able to reach very low loss demonstrate that even if our objective function is non convex, in practice it works as good or better as convex alternatives.

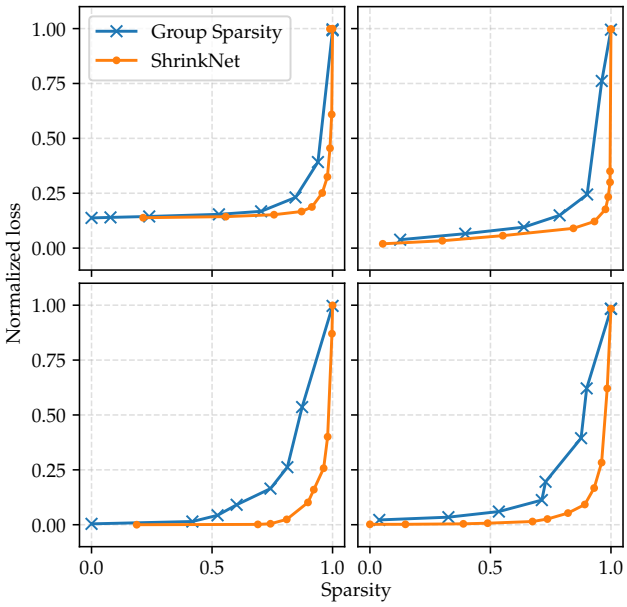

Figure 4. Loss/Sparsity trade off comparison between Group Sparsity and Shrinknet on linear and logistic regression. From top to bottom and left to right we show the results for `scm1d`, `oes97`, `gina_prior2` and `gsadd`.

References

Abadi, Martín, Barham, Paul, Chen, Jianmin, Chen, Zhifeng, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Irving, Geoffrey, Isard, Michael, Kudlur, Manjunath, Levenberg, Josh, Monga, Rajat, Moore, Sherry, Murray, Derek G, Steiner, Benoit, Tucker, Paul, Vasudevan, Vijay, Warden, Pete, Wicke, Martin, Yu, Yuan, Zheng, Xiaoqiang, Brain, Google, Osdi, Implementation, Barham, Paul, Chen, Jianmin, Chen, Zhifeng, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Irving, Geoffrey, Isard, Michael, Kudlur, Manjunath, Levenberg, Josh, Monga, Rajat, Moore, Sherry, Murray, Derek G, Steiner, Benoit, Tucker, Paul, Vasudevan, Vijay, Warden, Pete, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. TensorFlow : A System for Large-Scale Machine Learning. In *OSDI*, 2016. ISBN 9781931971331.

Aghasi, Alireza, Abdi, Afshin, Nguyen, Nam, and Romberg, Justin. Net-Trim: Convex Pruning of Deep Neural Networks with Performance Guarantee. 2016. URL https://papers.nips.cc/paper/6910-net-trim-convex-pruning-of-deep-neural-netwo pdfhttp://arxiv.org/abs/1611.05162.

Bergstra, James and Yoshua, Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13:281–305, 2012. ISSN 1532-4435. doi: 10.1162/153244303322533223. URL http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf.

Blackard, Jock A. *Comparison of Neural Networks and Discriminant Analysis in Predicting Forest Cover Types*. PhD thesis, Fort Collins, CO, USA, 1998. AAI9921979.

Cun, Yann Le, Denker, John S, and Solla, Sara a. Optimal Brain Damage. *Advances in Neural Information Processing Systems*, 2(1):598–605, 1990. ISSN 1098-6596. doi: 10.1.1.32.7223. URL https://papers.nips.cc/paper/250-optimal-brain-damage.pdf.

Guyon, I., Saffari, A., Dror, G., and Cawley, G. Agnostic learning vs. prior knowledge challenge. In *2007 International Joint Conference on Neural Networks*, pp. 829–834, Aug 2007. doi: 10.1109/IJCNN.2007.4371065.

Han, Song, Mao, Huizi, and Dally, William J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Hinton, Geoffrey, Vinyals, Oriol, and Dean, Jeff. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

Jamieson, Kevin and Talwalkar, Ameet. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*, pp. 240–248, 2016.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Krizhevsky, Alex. Learning Multiple Layers of Features from Tiny Images. ... *Science Department, University of Toronto, Tech.* ..., pp. 1–60, 2009. ISSN 1098-6596. doi: 10.1.1.222.9220. URL https://www.cs.toronto.edu/{~}kriz/learning-features-2009-TR.pdfhttp://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:Learning+Multiple+Layers+of+Features+from+Tiny+Images{#}0.

LeCun, Y, Bottou, L, Bengio, Yoshua, and Haffner, P. Gradient-Based Learning Applied to Document Recognition. In *Intelligent Signal Processing*, pp. 306–351, 2001. ISBN 0018-9219. doi: 10.1109/5.726791. URL http://vision.stanford.edu/cs598{_}spring07/papers/Lecun98.pdf.

Li, Lisha, Jamieson, Kevin, DeSalvo, Giulia, Rostamizadeh, Afshin, and Talwalkar, Ameet. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016.

Meier, Benjamin. Going deeper: Infinite deep neural networks. URL https://github.com/kutoga/going_deeper/raw/master/doc/going_deeper.pdf.

Nair, Vinod and Hinton, Geoffrey E. Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning*, (3):807–814, 2010. ISSN 1935-8237. doi: 10.1.1.165.6419. URL http://www.cs.toronto.edu/{~}fritz/absps/reluICML.pdf.

Paszke, Adam, Gross, Sam, Chintala, Soumith, Chanan, Gregory, Yang, Edward, DeVito, Zachary, Lin, Zeming, Desmaison, Alban, Antiga, Luca, and Lerer, Adam. Automatic differentiation in pytorch. 2017.

Philipp, George and Carbonell, Jaime G. Nonparametric Neural Network. In *Proc. International Conference on Learning Representations*, number 2016, pp. 1–27, 2017. URL https://www.cs.cmu.edu/{~}jgc/publication/NonparametricNeuralNetworks.pdfhttps://openreview.net/pdf?id=BJK3Xasel.

Rasmussen, CE and Williams, CKI. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, 1 2006. ISBN 0-262-18253-X.

Romero, Adriana, Ballas, Nicolas, Kahou, Samira Ebrahimi, Chassang, Antoine, Gatta, Carlo, and Bengio, Yoshua. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, Berg, Alexander C., and Fei-Fei, Li. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Scardapane, Simone, Comminiello, Danilo, Hussain, Amir, and Uncini, Aurelio. Group sparse regularization for deep neural networks. *Neurocomputing*, 241:81–89, 2017. ISSN 18728286. doi: 10.1016/j.neucom.2017.02.029. URL https://arxiv.org/pdf/1607.00485.pdf.

Shwartz-Ziv, Ravid and Tishby, Naftali. Opening the Black Box of Deep Neural Networks via Information. *arXiv*, pp. 1–19, mar 2017. URL http://arxiv.org/abs/1703.00810https://arxiv.org/pdf/1703.00810.pdf{%}0Ahttp://arxiv.org/abs/1703.00810.

Snoek, Jasper, Larochelle, Hugo, and Adams, Ryan P. Practical bayesian optimization of machine learning algorithms. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 2951–2959. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf.

Spyromitros-Xioufis, Eleftherios, Tsoumakas, Grigorios, Groves, William, and Vlahavas, Ioannis. Multi-target regression via input space expansion: treating targets as inputs. *Machine Learning*, 104(1):55–98, 2016. ISSN 1573-0565. doi: 10.1007/s10994-016-5546-z. URL http://dx.doi.org/10.1007/s10994-016-5546-z.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15: 1929–1958, 2014. ISSN 15337928. doi: 10.1214/12-AOS1000. URL https://www.cs.toronto.edu/{~}hinton/absps/JMLRdropout.pdf.

Vanschoren, Joaquin, van Rijn, Jan N., Bischl, Bernd, and Torgo, Luis. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198. URL http://doi.acm.org/10.1145/2641190.2641198.

Vergara, Alexander, Vembu, Shankar, Ayhan, Tuba, Ryan, Margaret A., Homer, Margie L., and Huerta, Ramn. Chemical gas sensor drift compensation using classifier ensembles. *Sensors and Actuators B: Chemical*, 166-167:320 – 329, 2012. ISSN 0925-4005. doi: https://doi.org/10.1016/j.snb.2012.01.074. URL http://www.sciencedirect.com/science/article/pii/S0925400512002018.

Xiao, Han, Rasul, Kashif, and Vollgraf, Roland. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. 2017. URL https://arxiv.org/pdf/1708.07747.pdfhttp://arxiv.org/abs/1708.07747.

Yuan, Ming and Lin, Yi. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 68(1):49–67, 2006. ISSN 13697412. doi: 10.1111/j.1467-9868.2005.00532.x. URL http://pages.stat.wisc.edu/{~}myuan/papers/glasso.final.pdf.

Zoph, Barret and Le, Quoc V. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016. URL http://arxiv.org/abs/1611.01578.

Zoph, Barret, Vasudevan, Vijay, Shlens, Jonathon, and Le, Quoc V. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.