DSD²: Can We Dodge Sparse Double Descent and Compress the Neural Network Worry-Free?

DSD^2: Can We Dodge Sparse Double Descent and Compress the Neural Network Worry-Free?

Victor Quétu, Enzo Tartaglione

LTCI, Télécom Paris, Institut Polytechnique de Paris, France
{name.surname}@telecom-paris.fr

Abstract

Neoteric works have shown that modern deep learning models can exhibit a sparse double descent phenomenon. Indeed, as the sparsity of the model increases, the test performance first worsens since the model is overfitting the training data; then, the overfitting reduces, leading to an improvement in performance, and finally, the model begins to forget critical information, resulting in underfitting. Such a behavior prevents using traditional early stop criteria.

In this work, we have three key contributions. First, we propose a learning framework that avoids such a phenomenon and improves generalization. Second, we introduce an entropy measure providing more insights into the insurgence of this phenomenon and enabling the use of traditional stop criteria. Third, we provide a comprehensive quantitative analysis of contingent factors such as re-initialization methods, model width and depth, and dataset noise. The contributions are supported by empirical evidence in typical setups. Our code is available at https://github.com/VGCQ/DSD2.

1 Introduction

Nowadays, deep neural networks are one of the most employed algorithms when required to solve complex tasks. In particular, their generalization capability allowed them to establish new state-of-the-art performance in domains like computer vision (He et al. 2016; Dosovitskiy et al. 2021) and natural language processing (Vaswani et al. 2017; Brown et al. 2020), showing as well promising capability in very complex hybrid tasks, like text-to-image generation (Ramesh et al. 2022; Saharia et al. 2022). The problem of optimally sizing these models is relevant to the vastly distributed employment of deep neural networks on edge devices (Chen and Ran 2019; Lin et al. 2022), posing questions about power consumption and hardware complexity (Goel et al. 2020; Luo et al. 2022).

It was general knowledge that the more a model is overparametrized, the easier it will overfit the training set, entering the *memorization phase*: the model memorizes the single samples in the training set, learning also a wrong set of features. Such a phenomenon harms the model's generalization, worsening its performance on unseen data (Liu et al. 2020). Recently, a new surprising phenomenon, *double descent* (DD) (Belkin et al. 2019), has been observed for extremely over-parametrized models: beyond the traditional over-fitting regime, while continuing to increase the size of the model, the generalization gap between train and test performance inverts trend, and narrows the more, the larger the model is (Nakkiran et al. 2020).

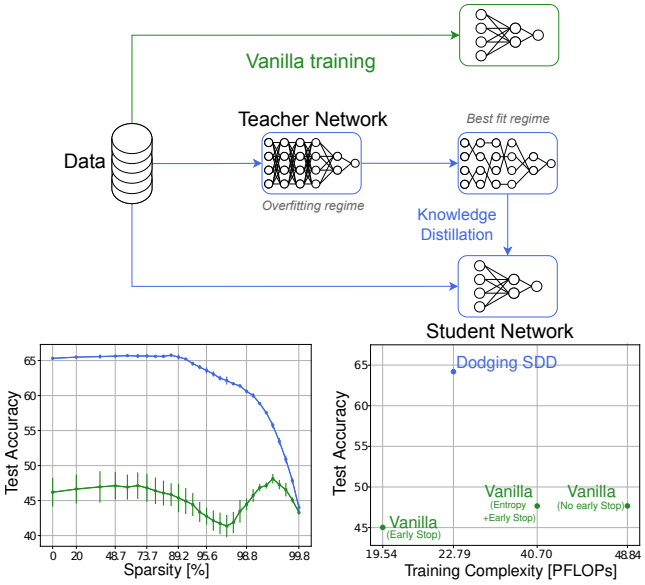


Figure 1: Distilling knowledge from a sparse teacher grants access to solutions (for the student model) where SDD is dodged, also saving computation.

The DD phenomenon raises questions about how to optimally size the model to have the best performance (while having the minimum size). Many approaches have indeed proposed the use of regularization functions to relieve DD in models for regression and classification tasks. However, when moving to real applications, the complexity of optimally tuning the regularization hyper-parameters and learning optimal early-stop discourages their use (Kan, Nagy, and Ruthotto 2020). While DD's typical analysis concerns moving from small models to big ones, a recent work observed a similar phenomenon also moving from an overparametrized model backward to a smaller one (He et al.

This paper has been accepted for publication at the 38th Annual AAAI Conference on Artificial Intelligence (AAAI24).

2022). That is possible thanks to pruning, which iteratively removes parameters from the model. Intuitively, while pruning removes parameters from the model, while at first the performance is enhanced, it will enter a second phase where its inevitable worsening is met (Han et al. 2015; Quétu and Tartaglione 2023). Such effect took the name of *sparse double descent* (SDD): is it inevitable? Can we properly regularize the model toward test performance enhancement? What is the underlying explanation of the SDD phenomenon?
We summarize our contributions as follows.

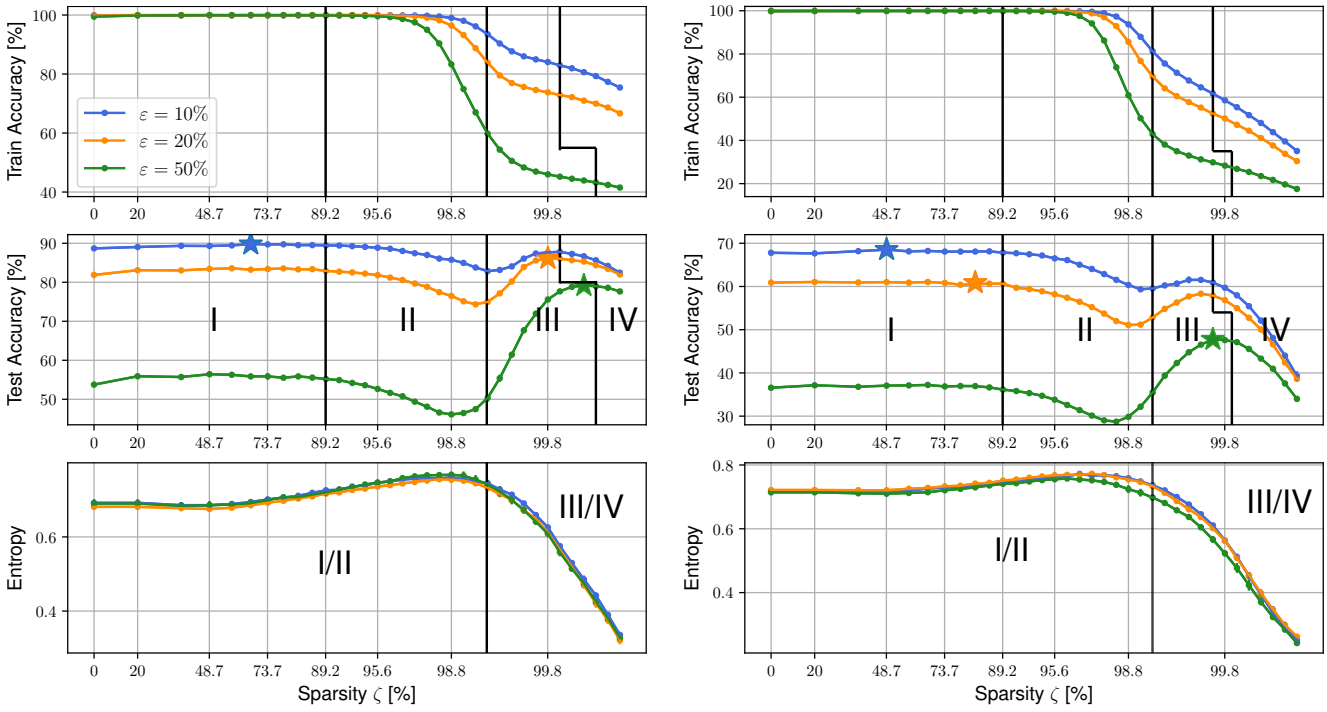
- To the best of our knowledge, we propose the first approach avoiding SDD, and providing a model with a good performance in terms of validation/test accuracy, consistently. More specifically, we regularize a student model distilling knowledge from a sparse teacher (in its best validation accuracy region), observing that the student dodges SDD (Fig. 1). Interestingly, we observe that this happens even when distilling knowledge from a non-pruned teacher.
- We study SDD from the perspective of "neuron's states": we calculate the entropy of the activations in the model, observing a correlation between the *interpolation regime* (where the SDD occurs) and the entropy's flatness. When leaving such a region, thus entering the *classical regime* where the traditional bias/variance trade-off occurs, the entropy monotonically decreases: a check on this measure enables-back the use of early stop criteria, which saves training computational cost (rightmost Fig. 1).
- We propose a quantitative study on some open questions, in particular: i) is DD/SDD still occurring when models increase in depth? ii) do we find the best validation/test performance configuration in models extremely over-parametrized or right before under-fitting? iii) is there a big difference between setting back parameters to their initial value (rewinding), randomly initializing, and not perturbing the model's parameters after pruning?

2 Related works

The real world is noisy In the real world, data acquisition is often noisy (Gupta and Gupta 2019), stemming from data collection or labeling. Concerning the annotation noise, many works proposed solutions to prevent the learning of wrong feature sets: for example, (Li et al. 2017) propose a unified distillation framework by leveraging the knowledge learned from a small clean dataset and semantic knowledge graph to correct the noisy labels. Other works find solutions inspired by the benefits of noise in the nervous system: (Arani, Sarfraz, and Zonooz 2021), for example, show that injecting constructive noise at different levels in the collaborative learning framework enables training the model effectively and distills desirable characteristics in the student model. More specifically, they propose methods to minimize the performance gap between a compact and a large model, to train high-performance compact adversarially-robust models.

Since a single image may belong to several categories, different samples can suffer from varying intensities of label noise. (Xu et al. 2020) proposed a simple yet effective feature normalized knowledge distillation that introduces the sample-specific correction factor to replace the temperature. (Kaiser et al. 2022) developed a teacher-student approach that identifies the tipping point between good generalization and overfits, thus estimating the noise in the training data with Otsu's algorithm. Other works focus more on the label's prediction robustness: (Sau and Balasubramanian 2016), for example, introduced a simple method that helps the student to learn better and produces results closer to the teacher network by injecting noise and perturbing the logit outputs of the teacher. With this setup, the noise simulates a multi-teacher setting and produces the effect of a regularizer due to its presence in the loss layer. To simulate noise in the labels, a typical approach is to manually inject noise in some well-annotated, standard datasets like MNIST and CIFAR-10/100 (Nakkiran et al. 2020; He et al. 2022). AI security works use a similar setup as well, where noise is injected parametrically to analyze the model's robustness against attacks. In a nutshell, these attacks propose adversarial representations of the data and check the model's performance - so this setup is named "adversarial learning" (Miller, Xiang, and Kesidis 2020).

Double Descent in classification tasks. Considering the presence of labeling noise, the occurrence of DD is a real threat. DD has already been reported in various machine learning models, like decision trees, random features (Meng, Yao, and Cao 2022), linear regression (Muthukumar et al. 2020)(Belkin, Hsu, and Xu 2020; Hastie et al. 2022), and deep neural networks (Yilmaz and Heckel 2022). For classification tasks, the test error of standard deep networks, like the ResNet architecture, trained on image classification datasets, consistently follows a double descent curve both when label noise is injected (CIFAR-10), and in some cases, even without any label noise injection (CIFAR-100) (Yilmaz and Heckel 2022). (Nakkiran et al. 2020) show that double descent occurs not just as a function of model size when increasing the model width, but also as a function of the number of training epochs. The double descent phenomenon has been extensively studied under the spectrum of over-parametrization (Nakkiran et al. 2020; Chang et al. 2021).
(He et al. 2022) observe the occurrence of double descent not only in the traditional setups but also when unstructurally pruning a dense model, observing the SDD phenomenon. Working on a related research question, and motivating the importance of further studying the SDD, (Chang et al. 2021) addressed the important question of whether it could be more convenient to train a small model directly, or quite first train a larger one and then prune it. In this work, the authors provide convincing evidence that the latter strategy is winning, toward enhanced model performance in sparsified regimes. Cotter et al. (2021), as a contrast to this work's purpose, exploited the DD phenomenon in a self-supervised setup, to assign pseudo-labels to a large held-out dataset. While this work sought to exploit DD, our goal is different: our objective is to avoid the sparse double descent, towards enhanced performance on the final student model, on the same task and dataset as the teacher. In the next section, we present and show the occurrence of SDD.



(a) (b)

Figure 2: Performance of ResNet-18 with different amount of noise ε on CIFAR-10 (a) and CIFAR-100 (b). **I**: Light Phase. **II**: Critical Phase. **III**: Sweet Phase. **IV**: Collapsed Phase.

3 Model size and sparse double descent

3.1 Background on neural network's pruning

Neural network pruning aims to reduce a large network while maintaining accuracy by removing irrelevant weights, filters, or other structures, from neural networks. All the pruning algorithms removing weights without explicitly considering the neural network's structure are typically named *unstructured* pruning methods. Various unstructured pruning methods exist and can be divided into magnitude-based, where the ranking for the parameters to prune is based on their magnitude (Han et al. 2015; Louizos, Welling, and Kingma 2018; Zhu and Gupta 2018), and gradient-based, where the ranking or the penalty term is a function of the gradient magnitude (or to higher order derivatives) (Lee, Ajanthan, and Torr 2019; Tartaglione et al. 2022). The general comparison between the effectiveness of any of the reported approaches is reported by (Blalock et al. 2020) and, although complex pruning approaches exist, the simple magnitude-based one, in general, is considered a good trade-off between complexity and competitiveness (Gale, Elsen, and Hooker 2019): hence, we will focus on this one.[1]

Training first an over-parametrized model, and then pruning it, leads to an improved generalization. In particular, (Chang et al. 2021) analyzes the beneficial effects of pruning, and then compares the performance achieved by pruned models to shallow vanilla ones. This work motivates the quest for investigating SDD, looking for the highest possible performance on unseen data.

3.2 Pruning exhibits sparse double descent

Setup The trained model $\mathcal{M}$ on the train set $\mathcal{D}_{\text{train}}$ (whose performance is evaluated on the validation set $\mathcal{D}_{\text{val}}$) consists of L layers, having $\boldsymbol{w}^{\mathcal{M}}$ as its set of parameters, and $\boldsymbol{w}_l^{\mathcal{M}}$ indicates those belonging to the l-th layer. When we prune the ζ-th fraction of parameters from the model, the parameters are projected to a parameter sub-space, according to a threshold on the quantile function $\mathcal{Q}_{\mathcal{M}}(\cdot)$, computed on the absolute values for parameters in $\mathcal{M}$.

The overall approach employed to reduce the dimensionality of the trained $\mathcal{M}$ follows these steps. The first step is to train the dense model. Until it has reached the desired sparsity percentage ζ_{wall}, the model is iteratively pruned using some pruning strategy (i.e. magnitude pruning, following (He et al. 2022)), perturbed (weights can be rewound to initialization, randomly re-initialized, or not perturbed at all), and the sparse model is re-trained on $\mathcal{D}_{\text{train}}$. The training follows standard policies: the set of hyper-parameters as well as the algorithm are reported in Appendix. When ζ_{wall} is reached, the model parameters $\boldsymbol{w}_{\text{best}}$, which achieve the best performance on the validation set $\mathcal{D}_{\text{val}}$ are returned.

Experiments As in (He et al. 2022), SDD is, consistently, found in our experiments. In particular, Fig. 2a and

[1](He et al. 2022) showed that magnitude, gradient-based, and random pruning achieve similar performance for the same setup as we consider in this work. Moreover, we also present a study employing structured ℓ_1-pruning in Appendix.

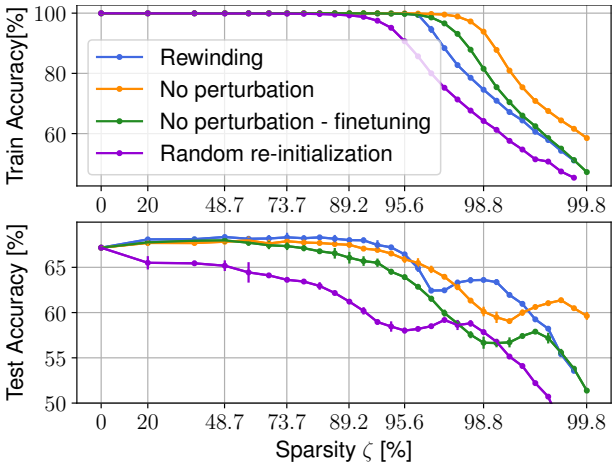


Figure 3: Performance of ResNet-18 on CIFAR-100 with $\varepsilon = 10\%$ when retrained from either the original initialization (lottery ticket), a random re-initialization, or from the last configuration achieved before pruning.

Fig. 2b show double descent of a ResNet-18 model trained on CIFAR-10 and CIFAR-100, respectively. The four phases reported by (He et al. 2022) are displayed in Fig. 2 (first a *light* phase, secondly a *critical* phase, thirdly the *sweet* phase and finally, the *collapsed* phase).

3.3 Better low parametrization or extreme over parametrization?

Currently, there is a big debate about whether some simple techniques, like early stopping, are sufficient to achieve great generalization performance. In particular, (Rice, Wong, and Kolter 2020) studies the extreme overparametrization for adversarially trained deep networks. The authors observe that overfitting the training set harms robust performance to a large degree in adversarially robust training across multiple datasets: this can be fought by simply using early stopping. To mitigate the robust overfitting, (Chen et al. 2021) propose self-training to smoothen the logits, combined with stochastic weight averaging trained by the same model, the other performing stochastic weight averaging (Izmailov et al. 2018). Although these works focus on DD, similar effects can be drawn to SDD: is it always true that the best model is in the sweet phase?

In our results presented in Fig. 2, we observe a correlation between the best model (marked with $\star$) and the noise in the dataset: on CIFAR-10, $\boldsymbol{w}_{\text{best}}$ is located in the sweet phase for $\varepsilon \in \{20\%, 50\%\}$, while on CIFAR-100, $\boldsymbol{w}_{\text{best}}$ is in the sweet phase only for $\varepsilon = 50\%$. Therefore, we empirically observe a correlation between the amount of noise in the training set and the location of $\boldsymbol{w}_{\text{best}}$: for small noise (i.e. $< 15\%$), $\boldsymbol{w}_{\text{best}}$ is located in the Light Phase (I). As ε exceeds 15%, $\boldsymbol{w}_{\text{best}}$ is consistently found in the Sweet Phase (III). A further analysis of this shift is conducted in Appendix.

3.4 Critical phase occurrence

(Frankle and Carbin 2019) conjectured, under the so-called *lottery ticket hypothesis*, that a large network contains smaller sub-networks, which can be trained in isolation without performance loss. Their approach consists of rewinding the parameters to their original value every pruning iteration and is one of the most popular for neural network pruning (Malach et al. 2020; Zhang et al. 2021).

We compare, in Fig. 3, different strategies to introduce perturbations in the model's parameters - random re-initialization, rewinding (in the lottery ticket hypothesis fashion), and not introducing any perturbation-, averaged on three seeds. We observe, in all the above-mentioned cases, the rise of the sparse double descent. However, we notice a gap in the performance between the different perturbation approaches. While randomly re-initializing the model leads to worse results, rewinding and not introducing any perturbation exhibit slightly different behavior, depending on the pruning regime. In particular, rewinding can marginally achieve a better performance, but the performance decays faster than not-perturbing. Moreover, we investigate whether it could be more convenient to let the model remain in the neighborhood of the same local minimum found by the dense model. Towards this end, we propose an experiment where we do not perturb the learned parameters, and we scale down the learning rate. In this scenario, we observe the performance consistently deteriorated. Our finding suggests that with rewinding we are post-posing, in the ζ plane, the critical phase, and without introducing any perturbation we are further post-posing it: we will use a "no perturbation" scheme for all our experiments.

3.5 An entropy-based interpretation to the sparse double descent

Analyzing the SDD phenomenon from a learning dynamics perspective raises questions related to the so-called *information bottleneck theory* (Tishby et al. 1999; Tishby and Zaslavsky 2015). In particular, this approach estimates the mutual information between the information processed by the layers and the input and output variables. Hence, it is possible to calculate optimal theoretical limits and set the bars for the generalization error. Several works have followed this theory, observing differences in the learning dynamics for different activation functions employed (Saxe et al. 2018), improving the estimation approach (Pan et al. 2021), and verifying that such theory admits DD in regression tasks (Ngampruetikorn and Schwab 2022). Inspired by these works, we formulate the following observation.

Observation 1 *As the size of the model, in terms of the number of parameters per neuron, shrinks from the light phase, the entropy of the features inside the model is stationary (as small adjustments in the parameters are needed). As we leave the interpolation regime right after the interpolation threshold, the entropy begins to decrease.*

To empirically verify this observation, we can visualize the entropy of the activations in the model. Considering that our observation is not constrained to any validation/test set, we will perform the measures directly on the training set. We

define the average neuron's entropy in the l-th layer as

$$\bar{\mathcal{H}}_{l|\mathcal{D}_{\text{train}}} = -\frac{1}{N_l}\sum_{i=1}^{N_l}\sum_{\zeta\in\{0;1\}} p\left(s_{l,i}^{\zeta}\right)\log\left[p\left(s_{l,i}^{\zeta}\right)\right], \quad (1)$$

where $p\left(s_{l,i}^{\zeta}\right)$ is the probability (in a frequentism sense, over $\mathcal{D}_{\text{train}}$) the i-th neuron (ReLU-activated) in the l-th layer to be in the negative region ($\zeta = 0$) or in the positive one ($\zeta = 1$), similarly to how done in (Liao et al. 2023; Spadaro et al. 2023). We provide more details on the entropy computation in Appendix. Fig. 2 displays the variation of the entropy, averaged across all the model's layers. For the considered examples, we corroborate our observation, enabling-back the use of early-stop criteria jointly with the entropy. Indeed, the entropy stays stationary and then decreases when the model enters the classical regime (entering the sweet phase). Using traditional early criteria starting from this regime saves training computation as the pruning/-training process is stopped when the performance decreases. We provide more details in Appendix.

3.6 Generalization gap in deep double descent: relationships between DD and SDD

In this section, we analyze the occurrence of the sparse double descent concerning the model's size. To carry out the following experiments, we have defined a multi-configurable "VGG-like" model: we can generate multiple architectures depending on the depth δ (the larger, the deeper the model) and the number of convolutional filters per layer 2^{γ} (the larger, the wider the model). More details on the generated architectures can be found in Appendix.

Results Fig. 4a shows the results at growing depth, with fixed $\gamma = 5$. The sparse double descent becomes more and more evident for growing depths, while for shallow models we can only observe the traditional regime of sweet and collapsed phase. We observe a similar trend also in Fig. 4b, where the depth of the model is set to $\delta = 1$ (two convolutional layers and one fully connected) while the width of the layer is increasing. With a fixed depth, increasing the width of the layers also reveals a double descent phenomenon: with low width, no SDD is observed, but it can be observed as γ is increased.

Typical analyses are performed, in the literature, in terms of the number of samples in the training set, or in terms of the width of the layers (Nakkiran et al. 2020; Chang et al. 2021; Chen, Wang, and Kyrillidis 2021). As the phenomenon also rises as a function of the model's depth, we believe SDD is more related to the number of parameters in a model rather than to the layer's organization and structure. This could motivate the occurrence of the sparse double descent, as the number of parameters is the varying quantity, while the width of the layer might not change.

4 Distilling knowledge to avoid the sparse double descent

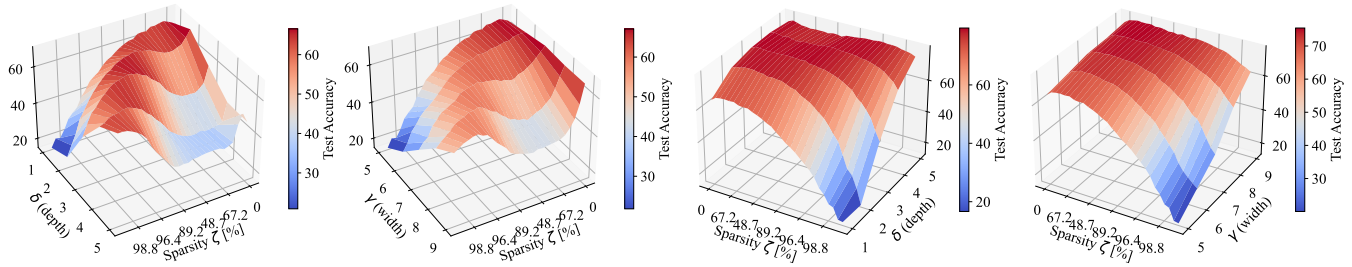
Although a standard ℓ_2 regularization approach can already positively contribute to dodging SDD,[2] it also presents some flaws. (Nakkiran et al. 2021) showed that, for certain linear regression models with isotropic data distribution, optimally-tuned ℓ_2 regularization can achieve monotonic test performance as either the sample size or the model size is grown. However, (Quétu and Tartaglione 2023) highlighted that in some image classification setups, like ResNet-18 on CIFAR datasets, SDD is still noticeable even if ℓ_2 regularization is used. Moreover, this regularization has the big drawback of sacrificing the performance/sparsity trade-off (Quétu, Milovanović, and Tartaglione 2023). Thus, the need to design a custom regularization to prevent SDD, while maintaining good performance, becomes evident. In this section, we present a learning scheme in which a student model is regularized by distilling knowledge from a sparse teacher in its best validation accuracy region (or even with a dense teacher), observing that the student is dodging SDD.

4.1 Background on knowledge distillation

Knowledge distillation (KD) transfers the knowledge from a (deep) teacher to a (shallow) student network. There are several knowledge types: response-based, feature-based, and relation-based. While response-based knowledge uses the output of the last layer of the teacher model as the knowledge (Hinton et al. 2015), the output of intermediate layers, which in convolutional neural networks are the feature maps, is used to supervise the training of the student model for feature-based knowledge (Romero et al. 2014). Relation-based knowledge further explores the relationships between different layers or data samples (Yim et al. 2017). In this work, we will focus on the more flexible and easy-to-deploy response-based knowledge.

Distilling the knowledge from a teacher model, besides enhancing the student's performance, can also implicitly transmit good regularization properties. While (Yuan et al. 2020) observe that good students could also improve the teacher's performance, and (Furlanello et al. 2018) observe improvement in distilling knowledge from students in a "born again" fashion, (Saglietti and Zdeborová 2022) introduce a formal statistical physics framework that allows an analytic characterization of the properties of KD in shallow neural networks. By using Gaussian Mixture Models, it is shown that, without any fine-tuning at the level of the student loss function, KD allows a transfer of the (possibly fine-tuned) regularization properties of the teacher, even if the two models are mismatched, and even if the regularization strategy in the teacher training is not explicitly known. This work motivates us, in the quest for transmitting an implicit teacher's regularization (as being either in the well-generalizing light phase or the sweet phase for high label noise) to a student, in the attempt to escape from the SDD. Indeed, (Saglietti and Zdeborová 2022) highlight a noticeable enhancement in the distillation test error when the trans-

[2] We also present a study on other types of regularization strategies towards avoidance of SDD in Appendix.



(a) (b) (c) (d)

Figure 4: Performance of VGG-like model, vanilla-trained (a, b), distilled from a sparse teacher (c, d), varying the depth δ (a, c) and the width γ (b, d) on CIFAR-10 with $\varepsilon = 50\%$.

ferred outputs are produced by an optimal teacher, with respect to a direct ℓ_2 regularization: the gap with the optimal generalization bound is nearly closed. KD enables the student model to reach a performance unachievable with other regularization approaches, inheriting the teacher's regularization implicitly.

4.2 Combining Knowledge Distillation and Network Pruning

Several works have already explored the combination of KD and pruning. Towards generalization enhancement, in the context of working with reduced size of the train set, (Zhou et al. 2022) propose a "progressive feature distribution distillation", which consists of first collecting a student network by pruning a trained network, and then distilling the feature distribution into the student.

A large slice of work combining the two techniques targets the final model's size reduction. For example, (Cui, Li, and Zhang 2021) combine structured pruning and dense knowledge distillation techniques to significantly compress an original large language model into a deep compressed shallow network, and (Wei, Hao, and Zhang 2022) follow a similar strategy for speech enhancement. Still working on language models, (Kim and Rush 2016) reveal that standard KD applied to word-level prediction can be effective for Neural Machine Translation, and applied weight pruning on top of it to decrease the number of parameters. Image processing-related works are, more applicative and oriented towards the enhancement of the performance on a sparsified model: (Chen et al. 2022) and (Aghli and Ribeiro 2021) are two works in this context. Leveraging on the beneficial, well-generalizing properties of both pruning and KD, (Park and No 2022) provide several applicative examples where the "prune, then distill" pattern is practically very effective.

Motivated by (Saglietti and Zdeborová 2022) and building on top of (Park and No 2022) (although in a very different context), we formulate the following observation, which will drive our approach towards SDD dodging

Observation 2 *In an adversarial learning setup with knowledge distillation, the teacher's response will act as a regularizer for the student model, which makes it avoid the sparse double descent. The student's performance, in a high-noise setup, is further enhanced when the knowledge is distilled from a sparse, well-performing teacher.*

Approach In general, the objective function to be minimized while training a smaller student network in the KD framework is a linear combination of two losses, in the image classification setup: a standard cross-entropy loss $\mathcal{L}_{\text{CE}}$, which uses "hard" ground truth targets, and the Kullback-Leibler divergence loss $\mathcal{L}_{\text{KL}}$, calculated between the teacher's and the student's predictions, eventually scaled by a temperature τ:

$$\mathcal{L} = (1-\alpha)\mathcal{L}_{\text{CE}}(\boldsymbol{y}^{\text{s}}, \hat{\boldsymbol{y}}) + \alpha\mathcal{L}_{\text{KL}}(\boldsymbol{y}^{\text{s}}, \boldsymbol{y}^{\text{t}}, \tau), \qquad (2)$$

where the apex "s" and "t" refer to the student and the teacher respectively, $\hat{\boldsymbol{y}}$ is the ground truth label, and α is the distillation hyper-parameter impacting on the weighted average between the distillation loss and the student loss. We employ the KD formulation loss as in (Hinton et al. 2015; Kim et al. 2021). To distill the knowledge to some students, we require the teacher model to be already in its best-fit regime, achieved through Alg. 1 in Appendix, and using the same sparsification process -but changing the objective function to (2)-, we can train and sparsify the student model.

Results For all the presented experiments here, $\alpha = 0.8$ and $\tau = 10$. An ablation study on these two hyper-parameters and the set of all the other hyper-parameters used for the learning process are presented in Appendix. Fig. 5 shows the results on CIFAR-10 and CIFAR-100 with $\varepsilon \in \{10\%, 20\%, 50\%\}$ using a VGG-like model with $2^{\gamma} = 32$ and $\delta = 5$ as student model, while ResNet-18 as teacher (whose training and performance is consistent to Fig. 2). We observe, consistently for all the investigated noise rates, that the student model, trained with a vanilla training setup, always reveals the sparse double descent phenomenon. Nevertheless, the same student model, trained using a KD setup, can consistently avoid SDD. Furthermore, even if the knowledge is distilled from a dense teacher, SDD is always avoided, either in the case of best performance in the light or the sweet phase. Observing the entropy for the model, it stays stationary and then decreases when the model enters the classical regime. Hence, once the entropy decreases, if the performance drops, the training

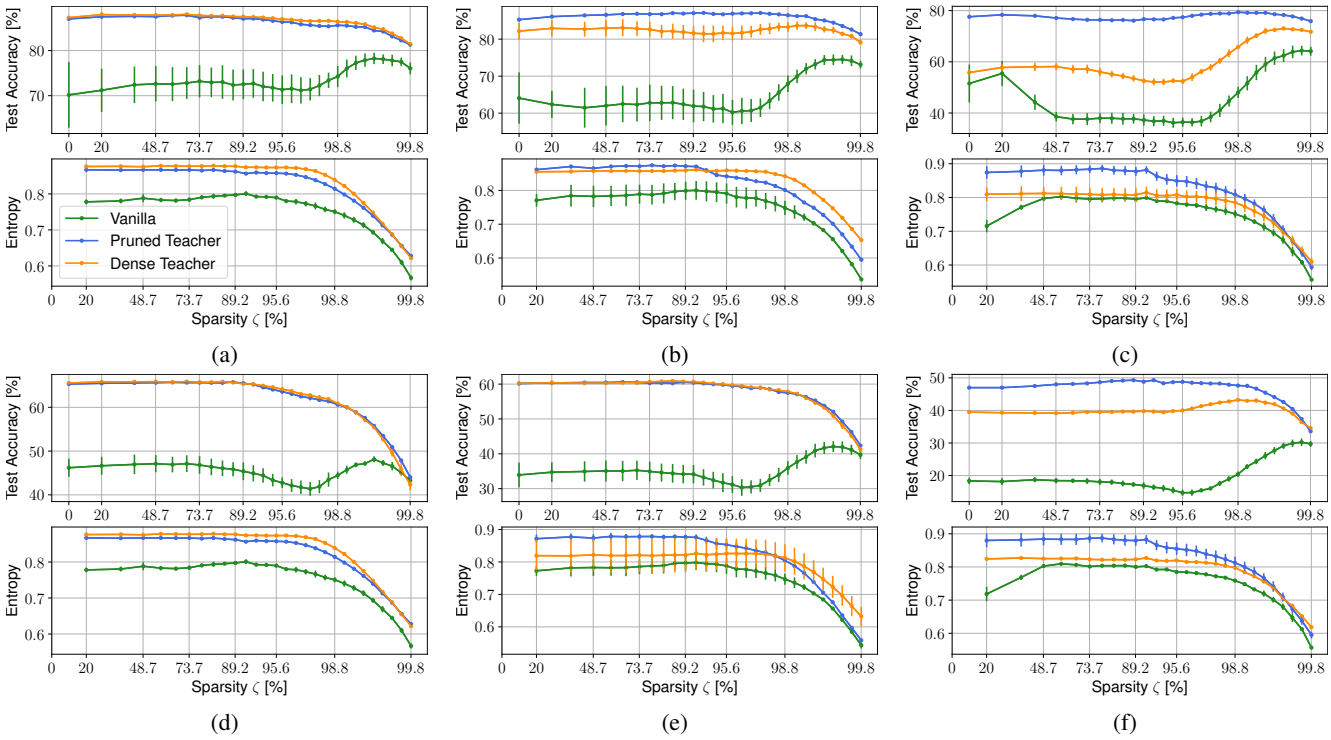


Figure 5: Performance of the VGG-like model on CIFAR-10 (a, b, c) and CIFAR-100 (d, e, f) for different label noises. **Left:** $\varepsilon = 10\%$. **Middle:** $\varepsilon = 20\%$. **Right:** $\varepsilon = 50\%$.

Table 1: Performance achieved and training computational cost of the student model with traditional approaches and our proposed scheme on CIFAR-10 with $\varepsilon = 20\%$. The training cost of the teacher model is reported between parenthesis.

Early stop	Distill	Distill from pr. teach.	Training [PFLOPs](↓)	Test accuracy [%](↑)	Sparsity [%] (↑)
			48.84	74.52 ± 1.20	99.62
✓			4.88	60.23 ± 3.37	36.00
✓	✓		35.82 (+1.63)	81.52 ± 1.85	99.26
✓	✓	✓	35.82 (+47.21)	86.89 ± 0.16	99.26

can be stopped as the model enters the under-fitting regime.[3]

Computation/performance trade-off We compare traditional approaches with our proposed scheme on CIFAR-10 with $\varepsilon = 20\%$ in Table 1 in terms of performance achieved and training computational cost for the student model.

In the vanilla case, early stop results in a model achieving low performance and with low sparsity. In order to extract a better performance, apparently, there is no other choice than pruning the model until all of its parameters are completely removed, which costs more than 48 PFLOPs. Our method achieves a model with high sparsity achieving more than 10% improvement in performance with approximately 25% computation less. We believe this is a core applicative contribution in real applications, where annotated datasets are small and noisy, and SDD can be easily observed. The same comparison for other setups can be found in Appendix.

Experiments with varying depth and width In Fig. 4c and Fig. 4d we report the distillation results on CIFAR-10 for $\varepsilon = 50\%$, with a varying depth and width for the student (results for the other noise values are displayed in Appendix). While in Fig. 4a and Fig. 4b we consistently observe that the vanilla-trained student model exhibits SDD as the width or depth is increased, we persistently notice in Fig. 4c and Fig. 4d, however, that employing KD within the framework, the performance exhibits a monotonic behavior: the sparse double descent is dodged.

Limitations and future work Leveraging a knowledge distillation scheme is a successful approach to transmitting good generalization properties from the teacher model to the student and enabling the dodge of SDD on the student itself. However, this method also presents some limits: in resource-constrained schemes, one might not afford to train a large teacher model. Nevertheless, our work is the first to enable the small model to enter the sweet phase instead of the critical phase, which cannot be achieved with traditional regularization strategies. Therefore, we let the exploration of more efficient approaches as future work.

[3] We have performed experiments on three other datasets, without noise injection: CIFAR-100N, a human-annotated dataset, Flowers-102, a typically small dataset, and ImageNet in Appendix.

5 Conclusion

In this paper, we have studied and proposed a solution to dodge the sparse double descent. This phenomenon poses critical questions about where to find the model with the best performance in the low or the high sparsification regime. We observe an interesting correlation between the occurrence of the critical phase and the entropy of the activation's state for the training set - when not in the classical regime, the entropy is stationary. Although SDD challenges traditional early-stop schemes, the use of the network's entropy, which indicates the region we are navigating on, enables them back. We further observe that by leveraging a knowledge distillation scheme, not only the good generalization property of the teacher is transmitted to the student, but the student itself is no longer subject to SDD, due to the implicit regularization inherited by the teacher. We hope this work will ignite new research toward the improvement of learning strategies for deep, sparse models.

6 Acknowledgements

This work was performed using HPC resources from GENCI–IDRIS (Grant 2022-AD011013930). Also, this research was partially funded by Hi!PARIS Center on Data Analytics and Artificial Intelligence.

References

Aghli, N.; and Ribeiro, E. 2021. Combining weight pruning and knowledge distillation for cnn compression. In *CVPR*.

Arani, E.; Sarfraz, F.; and Zonooz, B. 2021. Noise as a resource for learning in knowledge distillation. In *WACV*.

Belkin, M.; Hsu, D.; Ma, S.; and Mandal, S. 2019. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*.

Belkin, M.; Hsu, D.; and Xu, J. 2020. Two models of double descent for weak features. *SIAM Journal on Mathematics of Data Science*.

Blalock, D.; Gonzalez Ortiz, J. J.; Frankle, J.; and Guttag, J. 2020. What is the state of neural network pruning? *MLSys*.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *NeurIPS*.

Chang, X.; Li, Y.; Oymak, S.; and Thrampoulidis, C. 2021. Provable benefits of overparameterization in model compression: From double descent to pruning neural networks. In *AAAI*.

Chen, J.; and Ran, X. 2019. Deep learning with edge computing: A review. *IEEE*.

Chen, J.; Wang, Q.; and Kyrillidis, A. 2021. Mitigating deep double descent by concatenating inputs. In *CIKM*.

Chen, L.; Chen, Y.; Xi, J.; and Le, X. 2022. Knowledge from the original network: restore a better pruned network with knowledge distillation. *Complex & Intelligent Systems*.

Chen, T.; Zhang, Z. A.; Liu, S.; Chang, S.; and Wang, Z. 2021. Robust Overfitting may be mitigated by properly learned smoothening. In *ICLR*.

Cotter, A.; Menon, A. K.; Narasimhan, H.; Rawat, A. S.; Reddi, S. J.; and Zhou, Y. 2021. Distilling Double Descent. *arXiv preprint arXiv:2102.06849*.

Cui, B.; Li, Y.; and Zhang, Z. 2021. Joint structured pruning and dense knowledge distillation for efficient transformer model compression. *Neurocomputing*.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; and Houlsby, N. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*.

Frankle, J.; and Carbin, M. 2019. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *ICLR*.

Furlanello, T.; Lipton, Z.; Tschannen, M.; Itti, L.; and Anandkumar, A. 2018. Born again neural networks. In *ICML*.

Gale, T.; Elsen, E.; and Hooker, S. 2019. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*.

Goel, A.; Tung, C.; Lu, Y.-H.; and Thiruvathukal, G. K. 2020. A survey of methods for low-power deep learning and computer vision. In *WF-IoT*.

Gupta, S.; and Gupta, A. 2019. Dealing with noise problem in machine learning data-sets: A systematic review. *Procedia Computer Science*.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. *NeurIPS*.

Hastie, T.; Montanari, A.; Rosset, S.; and Tibshirani, R. J. 2022. Surprises in high-dimensional ridgeless least squares interpolation. *Annals of statistics*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*.

He, Z.; Xie, Z.; Zhu, Q.; and Qin, Z. 2022. Sparse Double Descent: Where Network Pruning Aggravates Overfitting. In *ICML*.

Hinton, G.; Vinyals, O.; Dean, J.; et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Izmailov, P.; Podoprikhin, D.; Garipov, T.; Vetrov, D.; and Wilson, A. G. 2018. Averaging weights leads to wider optima and better generalization. *UAI*.

Kaiser, T.; Ehmann, L.; Reinders, C.; and Rosenhahn, B. 2022. Blind Knowledge Distillation for Robust Image Classification. *arXiv preprint arXiv:2211.11355*.

Kan, K.; Nagy, J. G.; and Ruthotto, L. 2020. Avoiding The Double Descent Phenomenon of Random Feature Models Using Hybrid Regularization. *arXiv preprint arXiv:2012.06667*.

Kim, T.; Oh, J.; Kim, N.; Cho, S.; and Yun, S.-Y. 2021. Comparing kullback-leibler divergence and mean squared error loss in knowledge distillation. *IJCAI*.

Kim, Y.; and Rush, A. M. 2016. Sequence-level knowledge distillation. *EMNLP*.

Lee, N.; Ajanthan, T.; and Torr, P. 2019. SNIP: Single-shot network pruning based on connection sensitivity. In *ICLR*.

Li, Y.; Yang, J.; Song, Y.; Cao, L.; Luo, J.; and Li, L.-J. 2017. Learning from noisy labels with distillation. In *ICCV*.

Liao, Z.; Quétu, V.; Nguyen, V.-T.; and Tartaglione, E. 2023. Can Unstructured Pruning Reduce the Depth in Deep Neural Networks? In *ICCVW*.

Lin, J.; Zhu, L.; Chen, W.-M.; Wang, W.-C.; Gan, C.; and song han. 2022. On-Device Training Under 256KB Memory. In *NeurIPS*.

Liu, S.; Niles-Weed, J.; Razavian, N.; and Fernandez-Granda, C. 2020. Early-learning regularization prevents memorization of noisy labels. *NeurIPS*.

Louizos, C.; Welling, M.; and Kingma, D. P. 2018. Learning Sparse Neural Networks through L_0 Regularization. In *ICLR*.

Luo, X.; Liu, D.; Kong, H.; Huai, S.; Chen, H.; and Liu, W. 2022. Lightnas: On lightweight and scalable neural architecture search for embedded platforms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Malach, E.; Yehudai, G.; Shalev-Schwartz, S.; and Shamir, O. 2020. Proving the lottery ticket hypothesis: Pruning is all you need. In *ICML*.

Meng, X.; Yao, J.; and Cao, Y. 2022. Multiple Descent in the Multiple Random Feature Model. *arXiv preprint arXiv:2208.09897*.

Miller, D. J.; Xiang, Z.; and Kesidis, G. 2020. Adversarial Learning Targeting Deep Neural Network Classification: A Comprehensive Review of Defenses Against Attacks. *Proceedings of the IEEE*.

Muthukumar, V.; Vodrahalli, K.; Subramanian, V.; and Sahai, A. 2020. Harmless interpolation of noisy data in regression. *JSAIT*.

Nakkiran, P.; Kaplun, G.; Bansal, Y.; Yang, T.; Barak, B.; and Sutskever, I. 2020. Deep Double Descent: Where Bigger Models and More Data Hurt. In *ICLR*.

Nakkiran, P.; Venkat, P.; Kakade, S. M.; and Ma, T. 2021. Optimal Regularization can Mitigate Double Descent. In *ICLR*.

Ngampruetikorn, V.; and Schwab, D. J. 2022. Information bottleneck theory of high-dimensional regression: relevancy, efficiency and optimality. In *NeurIPS*.

Pan, Z.; Niu, L.; Zhang, J.; and Zhang, L. 2021. Disentangled information bottleneck. In *AAAI*.

Park, J.; and No, A. 2022. Prune your model before distill it. In *ECCV*.

Quétu, V.; Milovanović, M.; and Tartaglione, E. 2023. Sparse Double Descent in Vision Transformers: real or phantom threat? In *ICIAP*.

Quétu, V.; and Tartaglione, E. 2023. Dodging the Double Descent in Deep Neural Networks. In *ICIP*.

Ramesh, A.; Dhariwal, P.; Nichol, A.; Chu, C.; and Chen, M. 2022. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*.

Rice, L.; Wong, E.; and Kolter, J. Z. 2020. Overfitting in adversarially robust deep learning. In *ICML*.

Romero, A.; Ballas, N.; Kahou, S. E.; Chassang, A.; Gatta, C.; and Bengio, Y. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*.

Saglietti, L.; and Zdeborová, L. 2022. Solvable model for inheriting the regularization through knowledge distillation. In *MSML*.

Saharia, C.; Chan, W.; Saxena, S.; Li, L.; Whang, J.; Denton, E.; Ghasemipour, S. K. S.; Gontijo-Lopes, R.; Ayan, B. K.; Salimans, T.; Ho, J.; Fleet, D. J.; and Norouzi, M. 2022. Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. In *NeurIPS*.

Sau, B. B.; and Balasubramanian, V. N. 2016. Deep model compression: Distilling knowledge from noisy teachers. *arXiv preprint arXiv:1610.09650*.

Saxe, A. M.; Bansal, Y.; Dapello, J.; Advani, M.; Kolchinsky, A.; Tracey, B. D.; and Cox, D. D. 2018. On the Information Bottleneck Theory of Deep Learning. In *ICLR*.

Spadaro, G.; Renzulli, R.; Bragagnolo, A.; Giraldo, J. H.; Fiandrotti, A.; Grangetto, M.; and Tartaglione, E. 2023. Shannon Strikes Again! Entropy-Based Pruning in Deep Neural Networks for Transfer Learning Under Extreme Memory and Computation Budgets. In *ICCVW*.

Tartaglione, E.; Bragagnolo, A.; Fiandrotti, A.; and Grangetto, M. 2022. Loss-based sensitivity regularization: towards deep sparse neural networks. *Neural Networks*.

Tishby, N.; Pereira, F.; Bialek, W.; Hajek, B.; and Sreenivas, R. 1999. Proceedings of the 37th Annual Allerton Conference on Communication, Control and Computing.

Tishby, N.; and Zaslavsky, N. 2015. Deep learning and the information bottleneck principle. In *IEEE ITW*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *NeurIPS*.

Wei, J.; Zhu, Z.; Cheng, H.; Liu, T.; Niu, G.; and Liu, Y. 2022. Learning with Noisy Labels Revisited: A Study Using Real-World Human Annotations. In *ICLR*.

Wei, Z.; Hao, L.; and Zhang, X. 2022. Model Compression by Iterative Pruning with Knowledge Distillation and Its Application to Speech Enhancement. In *Interspeech*.

Xu, K.; Rui, L.; Li, Y.; and Gu, L. 2020. Feature normalized knowledge distillation for image classification. In *ECCV*.

Yilmaz, F. F.; and Heckel, R. 2022. Regularization-wise double descent: Why it occurs and how to eliminate it. In *ISIT*.

Yim, J.; Joo, D.; Bae, J.; and Kim, J. 2017. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *CVPR*.

Yuan, L.; Tay, F. E.; Li, G.; Wang, T.; and Feng, J. 2020. Revisiting knowledge distillation via label smoothing regularization. In *CVPR*.

Zhang, Z.; Chen, X.; Chen, T.; and Wang, Z. 2021. Efficient lottery ticket finding: Less data is more. In *ICML*.

Zhou, Z.; Zhou, Y.; Jiang, Z.; Men, A.; and Wang, H. 2022. An Efficient Method for Model Pruning Using Knowledge Distillation with Few Samples. In *ICASSP*.

Zhu, M. H.; and Gupta, S. 2018. To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression.

A Details on the architectures employed

To conduct extensive experiments to evaluate the impact of the width and the depth of the model on double descent, we have defined a model, based on a VGG-like architecture composed of blocks. Each block is composed of a 2D convolutional layer, followed by a ReLU activation function and a batch-normalization layer.

Every VGG-like model is defined iteratively depending on the value of γ and δ. δ stands for the number of (two blocks followed by a max-pooling layer). Hence, increasing δ by one is equivalent to adding two blocks ended by a max-pooling layer. γ is the power of two, setting the width of the convolutional layer. Thus, adding one to γ, is equivalent to multiplying the number of filters in the convolutional layer by 2. The last two layers of the VGG-like model architecture are always an adaptative average pooling layer and a linear layer. A summary of the architecture of the VGG-like model according to γ and δ can be found in Table 2.

Possible depth configurations				
$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta = 5$
Input (RGB image)				
Conv2d -2^{γ}	Conv2d -2^{γ}	Conv2d -2^{γ}	Conv2d -2^{γ}	Conv2d -2^{γ}
ReLU	ReLU	ReLU	ReLU	ReLU
BatchNorm2d	BatchNorm2d	BatchNorm2d	BatchNorm2d	BatchNorm2d
Conv2d - 2^{γ}	Conv2d - 2^{γ}	Conv2d - 2^{γ}	Conv2d - 2^{γ}	Conv2d - 2^{γ}
ReLU	ReLU	ReLU	ReLU	ReLU
BatchNorm2d	BatchNorm2d	BatchNorm2d	BatchNorm2d	BatchNorm2d
	Maxpool			
	Conv2d -$2^{\gamma+1}$	Conv2d -$2^{\gamma+1}$	Conv2d -$2^{\gamma+1}$	Conv2d -$2^{\gamma+1}$
	ReLU	ReLU	ReLU	ReLU
	BatchNorm2d	BatchNorm2d	BatchNorm2d	BatchNorm2d
	Conv2d - $2^{\gamma+1}$	Conv2d - $2^{\gamma+1}$	Conv2d - $2^{\gamma+1}$	Conv2d - $2^{\gamma+1}$
	ReLU	ReLU	ReLU	ReLU
	BatchNorm2d	BatchNorm2d	BatchNorm2d	BatchNorm2d
		Maxpool		
		Conv2d - $2^{\gamma+2}$	Conv2d - $2^{\gamma+2}$	Conv2d - $2^{\gamma+2}$
		ReLU	ReLU	ReLU
		BatchNorm2d	BatchNorm2d	BatchNorm2d
		Conv2d - $2^{\gamma+2}$	Conv2d - $2^{\gamma+2}$	Conv2d - $2^{\gamma+2}$
		ReLU	ReLU	ReLU
		BatchNorm2d	BatchNorm2d	BatchNorm2d
			Maxpool	
			Conv2d - $2^{\gamma+3}$	Conv2d - $2^{\gamma+3}$
			ReLU	ReLU
			BatchNorm2d	BatchNorm2d
			Conv2d - $2^{\gamma+3}$	Conv2d - $2^{\gamma+3}$
			ReLU	ReLU
			BatchNorm2d	BatchNorm2d
				Maxpool
				Conv2d - $2^{\gamma+4}$
				ReLU
				BatchNorm2d
				Conv2d - $2^{\gamma+4}$
				ReLU
				BatchNorm2d
Adaptive Average Pooling				
Fully Connected				

Table 2: The possible configurations used for the "VGG-like model Architecture".

We include in Tab. 3 a summary of the model's initial sizes used in the different experiments. Thanks to this initial size and the sparsity percentage of a model in a given experiment, one is able to have an idea of the total number of remaining/pruned weights.

Architecture	Number of parameters
ResNet-18	11.1 M
ResNet-50	23.5 M
VGG-like ($\delta = 1, \gamma = 5$)	26.0 K
VGG-like ($\delta = 1, \gamma = 6$)	70.3 K
VGG-like ($\delta = 1, \gamma = 7$)	214.4 K
VGG-like ($\delta = 1, \gamma = 8$)	723.7 K
VGG-like ($\delta = 1, \gamma = 9$)	2.6 M
VGG-like ($\delta = 2, \gamma = 5$)	97.3 K
VGG-like ($\delta = 3, \gamma = 5$)	350.6 K
VGG-like ($\delta = 4, \gamma = 5$)	1.3 M
VGG-like ($\delta = 5, \gamma = 5$)	4.9 M

Table 3: Model's initial number of parameters for a given architecture.

B Details on the learning strategies employed

The implementation details used in this paper are presented here.

B.1 Experimental details

Like in (He et al. 2022) set up, for the ResNet-18 network, a modified version of the `torchvision` model is used: the first convolutional layer is set with a filter of size 3×3 and the max-pooling layer that follows has been eliminated to adapt ResNet-18 for CIFAR-10 and CIFAR-100. CIFAR-10 and CIFAR-100 are augmented with per-channel normalization, random horizontal flipping, and random shifting by up to four pixels in any direction. ImageNet is augmented with per-channel normalization, random horizontal flipping, random cropping, and resizing to 224.

In pruning experiments, all weights from convolutional and linear layers are set as prunable no matter the model architecture. Neither biases nor batch normalization parameters are pruned.

The training hyperparameters used in the experiments are presented in Table 4. Our code can be found at https://github.com/VGCQ/DSD2.

Model	Dataset	Epochs	Batch	Opt.	Mom.	LR	Milestones	Drop Factor	Weight Decay	Rewind Iter.
ResNet-18	CIFAR-10	160	128	SGD	0.9	0.1	[80, 120]	0.1	1e-4	1000
VGG-like	CIFAR-10	160	128	SGD	0.9	0.001	[80, 120]	0.1	1e-4	0
ResNet-18	CIFAR-100	160	128	SGD	0.9	0.1	[80, 120]	0.1	1e-4	1000
VGG-like	CIFAR-100	160	128	SGD	0.9	0.001	[80, 120]	0.1	1e-4	0
ResNet-18	CIFAR-100N	160	128	SGD	0.9	0.1	[80, 120]	0.1	1e-4	1000
VGG-like	CIFAR-100N	160	128	SGD	0.9	0.001	[80, 120]	0.1	1e-4	0
ResNet-18	Flowers102	160	64	SGD	0.9	0.001	[80, 120]	0.1	1e-5	0
ResNet-50	ImageNet	90	1024	SGD	0.9	0.1	[30, 60]	0.1	1e-4	0
ResNet-18	ImageNet	90	1024	SGD	0.9	0.1	[30, 60]	0.1	1e-4	0

Table 4: Table of the different employed learning strategies.

B.2 Algorithm

First, we introduce the function PPTE in Alg. 1, which is used in Alg. 1 and Alg. 2.

The function first **P**runes the model $\mathcal{M}$(line 18) using some pruning strategy (we employ magnitude pruning, following (He et al. 2022)), **P**erturb (line 19 - weights can be rewound to initialization, randomly re-initialized, or not perturbed at all), and re-**T**rained on the training set Ξ_{train} (line 20). Then, it **E**valuates the performance of the model on the validation set Ξ_{val} (line 21). Finally, the function returns the model, represented by its weights $\boldsymbol{w}^{\mathcal{M}}$, as well as its performance on the validation set (line 22).

Moreover, we present our iterative pruning algorithm employed to reduce the dimensionality of the trained $\mathcal{M}$ in Alg. 1.

Algorithm 1: Iterative pruning algorithm & PPTE function.

1: **function** Iterative pruning($\boldsymbol{w}^{\mathcal{M}}$, Ξ, ζ, ζ_{wall})
2: $\quad\boldsymbol{w}^{\mathcal{M}} \leftarrow$Train($\boldsymbol{w}^{\mathcal{M}}$, Ξ_{train})
3: $\quad\boldsymbol{w}^{\mathcal{M}}_{\text{best}} \leftarrow \boldsymbol{w}^{\mathcal{M}}$
4: $\quad$best_acc $\leftarrow$ Evaluate(ζ_{current}, Ξ_{val})
5: $\quad\zeta_{\text{current}} \leftarrow \zeta$
6: $\quad$**while** $\zeta_{\text{current}} < \zeta_{\text{wall}}$ **do**
7: $\quad\quad\boldsymbol{w}^{\mathcal{M}}$, this_acc $\leftarrow$ PPTE($\boldsymbol{w}^{\mathcal{M}}$, ζ_{current}, Ξ_{train}, Ξ_{val})
8: $\quad\quad$**if** this_acc>best_acc **then**
9: $\quad\quad\quad\boldsymbol{w}^{\mathcal{M}}_{\text{best}} \leftarrow \boldsymbol{w}^{\mathcal{M}}$
10: $\quad\quad\quad$best_acc $\leftarrow$ this_acc
11: $\quad\quad$**end if**
12: $\quad\quad\zeta_{\text{current}} \leftarrow 1-(1-\zeta_{\text{current}})(1-\zeta)$
13: $\quad$**end while**
14: $\quad$**return** $\boldsymbol{w}^{\mathcal{M}}_{\text{best}}$
15: **end function**
16:
17: **function** PPTE($\boldsymbol{w}^{\mathcal{M}}$, ζ, Ξ_{train}, Ξ_{val})
18: $\quad\boldsymbol{w}^{\mathcal{M}} \leftarrow$ **P**rune($\boldsymbol{w}^{\mathcal{M}}$, ζ)
19: $\quad\boldsymbol{w}^{\mathcal{M}} \leftarrow$ **P**erturb($\boldsymbol{w}^{\mathcal{M}}$)
20: $\quad\boldsymbol{w}^{\mathcal{M}} \leftarrow$**T**rain($\boldsymbol{w}^{\mathcal{M}}$, Ξ_{train})
21: $\quad$this_acc $\leftarrow$ **E**valuate(ζ_{current}, Ξ_{val})
22: $\quad$**return** $\boldsymbol{w}^{\mathcal{M}}$, this_acc
23: **end function**

The first step is to train the dense model (line 2). Until it has reached the desired sparsity percentage ζ_{wall} (line 6), the model is iteratively pruned, perturbed, re-trained on Ξ_{train} and evaluated on the validation set Ξ_{val} (line 7). When ζ_{wall} is reached, the algorithm returns the model parameters $\boldsymbol{w}_{\text{best}}$, which achieve the best performance on the validation set Ξ_{val} (line 14).

C Ablation study for α and τ

In order to choose the set of parameters α and τ which are used in our learning framework, we carried out an ablation study over these 2 parameters. We report in Tab. 5 the results of the validation accuracy achieved by the simple model on CIFAR-10 with $\varepsilon = 10\%$, $\varepsilon = 20\%$ and $\varepsilon = 50\%$ for different sets of α and τ. To choose the final α and τ, we compute the average validation accuracy obtained for the 3 noise ratios. We observe that the best validation accuracy is achieved for $\alpha = 0.8$ and $\tau = 10$.

Noise rate ε	α	Temperature τ	Validation Accuracy
10%	0.5	10	86.77
	0.7		86.06
	0.8		**87.33**
	0.9		86.69
	0.5	20	86.26
	0.7		86.33
	0.8		86.76
	0.9		86.92
20%	0.5	10	84.64
	0.7		**85.36**
	0.8		85.24
	0.9		84.66
	0.5	20	84.51
	0.7		84.90
	0.8		85.27
	0.9		85.28
50%	0.5	10	77.72
	0.7		**79.23**
	0.8		79.09
	0.9		78.91
	0.5	20	76.46
	0.7		77.53
	0.8		78.67
	0.9		78.98
Average on ε values	0.5	10	83.04
	0.7		83.55
	0.8		**83.89**
	0.9		83.42
	0.5	20	82.41
	0.7		82.92
	0.8		83.57
	0.9		83.73

Table 5: Ablation study over α and τ on CIFAR-10 for the VGG-like architecture. The best performance over the 3 noise rates is achieved for $\tau = 10$ and $\alpha = 0.8$.

D Comparison with other regularization approaches towards avoidance of SDD

(Nakkiran et al. 2021) showed in regression tasks that an optimal ℓ_2 regularization can help mitigate DD. However, a recent work (Quétu and Tartaglione 2023) highlights that in image classification, the problem is not easily overcome and this regularization is not sufficient enough to completely lessen the phenomenon: a more complex approach has to be leveraged. To compare our framework with other existing regularization methods, we conduct here a comparison with other regularization approaches: ℓ_1, ℓ_2, dropout, and data augmentation. The results for a ResNet-18 are presented in Fig.6 on CIFAR-100 with $\varepsilon = 50\%$.

The use of any of these regularizations isn't helping to relieve the phenomenon. Indeed, SDD occurs in the experiments using data augmentation (the training dataset is augmented with AutoAugment and CutOut). Moreover, even if a dropout layer before the classifier with a probability of 0.5 is used, SDD still appears. Finally, for ℓ_1 and ℓ_2 regularizations weighted by λ, we show that for $\lambda = 1e-5$ and $\lambda = 1e-4$ respectively, SDD is not mitigated. Moreover, by increasing their value to respectively 1e-4 and 1e-3, the training fails in the case of ℓ_1, and the phenomenon is just shifted to the left (confirming the results shown in (Quétu and Tartaglione 2023)).

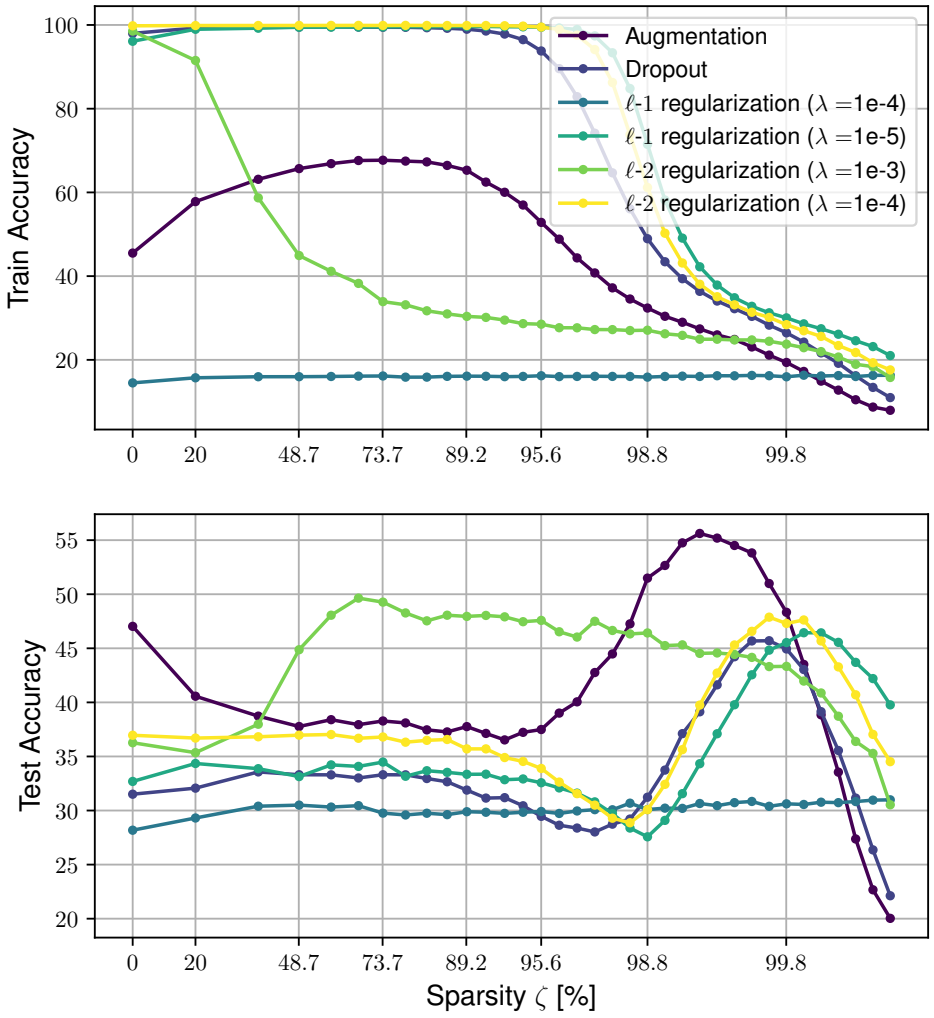


Figure 6: Comparison with other regularization approaches for a ResNet-18 on CIFAR-100 with $\varepsilon = 50\%$.

E Details on the computation resources and the compression/performance tradeoff.

A model is subject to SDD if its over-parametrization is massive (He et al. 2022): it is impossible to effectively stop the learning/compression with any early-stop criteria, as the accuracy curve will invert, at some point, its trend. Guaranteeing the SSD is dodged will avoid such non-monotonic behavior, enabling back early stop approaches. To illustrate the motivation, let us compare traditional approaches with our proposed scheme presented in Alg. 2 on CIFAR-10 and CIFAR-100 with $\varepsilon \in \{10\%, 20\%, 50\%\}$ here below.

Early stop	Distillation	Distillation from pruned teacher	Training FLOPs [PFLOPs](↓)	Test accuracy [%](↑)	Sparsity [%] (↑)
			48.84	76.04 ± 1.30	99.52
✓			48.84	76.04 ± 1.30	99.80
✓	✓		48.84 (+1.63)	81.42 ± 0.15	99.80
✓	✓	✓	48.84 (+9.77)	81.29 ± 0.23	99.80

Table 6: Performance achieved and training computational cost of traditional approaches and our proposed scheme on CIFAR-10 with $\varepsilon = 10\%$.

Early stop	Distillation	Distillation from pruned teacher	Training FLOPs [PFLOPs](↓)	Test accuracy [%](↑)	Sparsity [%] (↑)
			48.84	74.52 ± 1.20	99.62
✓			4.88	60.23 ± 3.37	36.00
✓	✓		35.82 (+1.63)	81.52 ± 1.85	99.26
✓	✓	✓	35.82 (+47.21)	86.89 ± 0.16	99.26

Table 7: Performance achieved and training computational cost of traditional approaches and our proposed scheme on CIFAR-10 with $\varepsilon = 20\%$.

Early stop	Distillation	Distillation from pruned teacher	Training FLOPs [PFLOPs](↓)	Test accuracy [%](↑)	Sparsity [%] (↑)
			48.84	64.37 ± 1.89	99.76
✓			6.51	38.57 ± 1.85	48.80
✓	✓		48.84 (+1.63)	71.72 ± 0.36	99.80
✓	✓	✓	48.84 (+52.09)	75.87 ± 0.18	99.80

Table 8: Performance achieved and training computational cost of traditional approaches and our proposed scheme on CIFAR-10 with $\varepsilon = 50\%$.

Early stop	Distillation	Distillation from pruned teacher	Training FLOPs [PFLOPs](↓)	Test accuracy [%](↑)	Sparsity [%] (↑)
			48.84	48.13 ± 0.68	99.53
✓			19.54	44.93 ± 1.85	91.41
✓	✓		22.79 (+1.63)	64.60 ± 0.20	94.50
✓	✓	✓	22.79 (+6.51)	64.07 ± 0.33	94.50

Table 9: Performance achieved and training computational cost of traditional approaches and our proposed scheme on CIFAR-100 with $\varepsilon = 10\%$.

In the vanilla case (second lines), early stop results in a model achieving low performance and low sparsity. In order to extract a better performance, apparently, there is no other choice than pruning the model until all of its parameters are completely removed, which results in a *big computation overhead* (first lines).

Our method achieves a model with high sparsity resulting in better performance with less computation compared to the vanilla case. We believe this is a core applicative contribution in real applications, where annotated datasets are small and noisy, and SDD can be easily observed.

Early stop	Distillation	Distillation from pruned teacher	Training FLOPs [PFLOPs](↓)	Test accuracy [%](↑)	Sparsity [%] (↑)
			48.84	52.06 ± 1.57	99.62
✓			24.42	30.33 ± 2.39	96.48
✓	✓		30.93 (+1.63)	58.26 ± 0.45	98.56
✓	✓	✓	30.93 (+14.65)	57.73 ± 0.29	98.56

Table 10: Performance achieved and training computational cost of traditional approaches and our proposed scheme on CIFAR-100 with $\varepsilon = 20\%$.

Early stop	Distillation	Distillation from pruned teacher	Training FLOPs [PFLOPs](↓)	Test accuracy [%](↑)	Sparsity [%] (↑)
			48.84	30.24 ± 1.16	99.75
✓			22.79	14.72 ± 0.93	95.60
✓	✓		42.33 (+1.63)	39.06 ± 0.19	99.70
✓	✓	✓	42.33 (+45.58)	40.45 ± 0.21	99.70

Table 11: Performance achieved and training computational cost of traditional approaches and our proposed scheme on CIFAR-100 with $\varepsilon = 50\%$.

F Correlation between $\boldsymbol{w}_{best}$ and ε.

We include below a deeper analysis of the correlation between $\boldsymbol{w}_{best}$ and ε. Given that the shift of the best fitting model, for CIFAR-10, is between $\varepsilon = 10\%$ and $\varepsilon = 20\%$, we propose down here a study including also the new noise levels $\varepsilon = 12.5\%, 15\%, 17.5\%$, following the same experimental setup we use and detailed in Appendix B (ResNet-18).

Noise rate ε [%]	Best Accuracy [%]	Sparsity [%]	Best Accuracy Phase
10	89.66	86.58	Light Phase (I)
12.5	87.57	86.58	Light Phase (I)
15	86.98	99.81	Sweet Phase (III)
17.5	86.29	99.84	Sweet Phase (III)
20	85.72	99.81	Sweet Phase (III)
50	77.12	99.88	Sweet Phase (III)

Table 12: Correlation between noise and $\boldsymbol{w}_{best}$

Empirically, we observe a correlation between the best model and the noise rate in the dataset: on CIFAR-10, $\boldsymbol{w}_{best}$ is located in the Light Phase (I) for small noise rate (i.e. $< 15\%$). Once, the noise rate exceeds 15%, $\boldsymbol{w}_{best}$ is consistently found in the Sweet Phase (III).

G Study employing structured ℓ_1-pruning

We mainly consider unstructured pruning as it is also the one chosen pruning strategy to evidence SDD in (He et al. 2022). However, we propose here below a study employing structured ℓ_1-pruning on CIFAR-10 with $\varepsilon = 50\%$, in the same experimental setup detailed in Appendix B (but with ($\delta = 1$).

# Filters	Vanilla test accuracy [%]	Distillation from pruned teacher test accuracy [%]
512	66.80	73.08
256	67.10	71.54
128	68.43	70.86
64	66.88	67.96
32	65.20	65.70

Table 13: Study employing structured ℓ_1-pruning.

Empirically, we still observe similar behavior as with unstructured pruning: for the vanilla model there is an evident non-monotonic behavior as the number of filters is reduced in the convolutional block, while with our distillation approach the trend is monotonic, and the performance is consistently higher than with the vanilla approach.

H Re-enabling early-stop criteria

We present here an overall approach enabling-back the use of early-stop criteria jointly with the entropy for a given model $\mathcal{M}$ in Alg. 2. Indeed, as highlighted in Fig 2, the entropy stays stationary and then decreases when the model enters the classical regime. Using traditional early criteria starting from this regime can save training computation as the pruning/training process is stopped when the performance decreases.

Algorithm 2: Re-enabling early-stop.

1: **function** Early-stopping($\boldsymbol{w}^{\mathcal{M}}$, Ξ, ζ, $\mathcal{T}_E$, $\mathcal{T}_A$)
2: $\boldsymbol{w}^{\mathcal{M}} \leftarrow$Train($\boldsymbol{w}^{\mathcal{M}}$, Ξ_{train})
3: best_acc $\leftarrow$ Evaluate(ζ_{current}, Ξ_{val})
4: $\eta_0 \leftarrow$ Entropy($\boldsymbol{w}^{\mathcal{M}}$, ζ_{current}, Ξ_{train})
5: $\eta_{\text{current}} \leftarrow \eta_0$
6: $\zeta_{\text{current}} \leftarrow \zeta$
7: **while** η_{current} ¿ $\eta_0 \times \mathcal{T}_E$ **do**
8: $\boldsymbol{w}^{\mathcal{M}}$, this_acc $\leftarrow$ PPTE($\boldsymbol{w}^{\mathcal{M}}$, ζ_{current}, Ξ_{train}, Ξ_{val})
9: best_acc $\leftarrow \max$(this_acc, best_acc)
10: $\eta_{\text{current}} \leftarrow$ Entropy($\boldsymbol{w}^{\mathcal{M}}$, ζ_{current}, Ξ_{train})
11: $\zeta_{\text{current}} \leftarrow 1 - (1 - \zeta_{\text{current}})(1 - \zeta)$
12: **end while**
13: **while** this_acc>best_acc $\times \mathcal{T}_A$ **do**
14: $\boldsymbol{w}^{\mathcal{M}}$, this_acc $\leftarrow$ PPTE($\boldsymbol{w}^{\mathcal{M}}$, ζ_{current}, Ξ_{train}, Ξ_{val})
15: $\zeta_{\text{current}} \leftarrow 1 - (1 - \zeta_{\text{current}})(1 - \zeta)$
16: **end while**
17: **return** $\boldsymbol{w}^{\mathcal{M}}$
18: **end function**

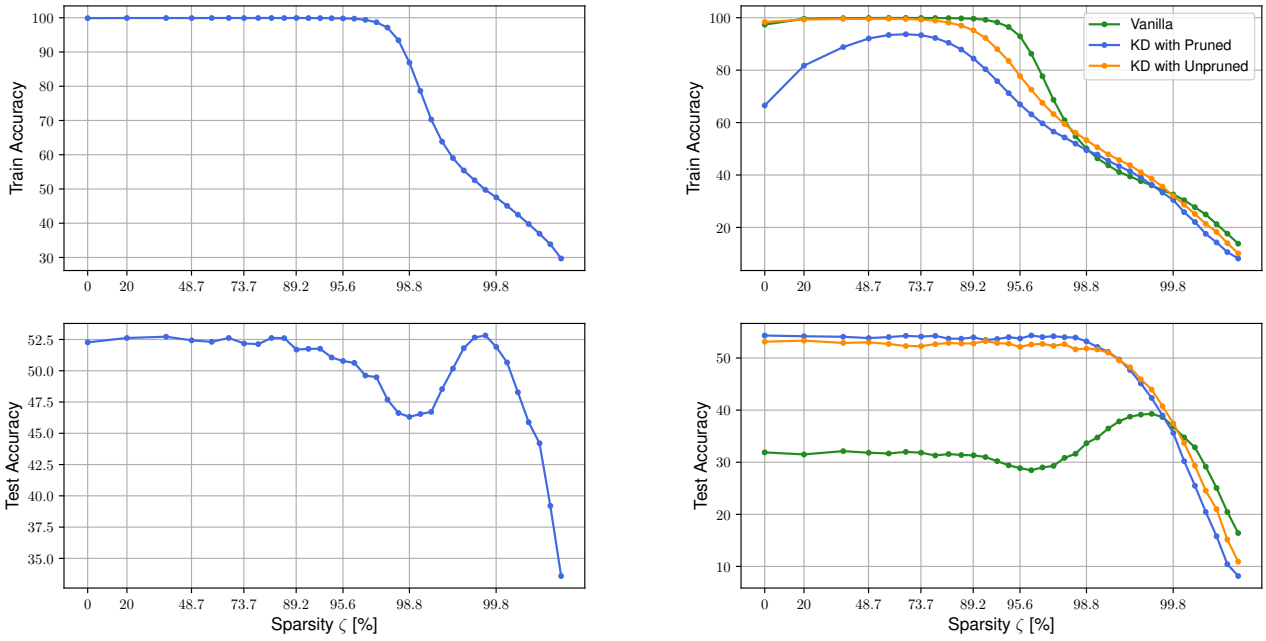
The first step is to train the dense model (line 2). While the entropy η calculated on the training set Ξ_{train} remains stationary, i.e. when $\eta_{\text{current}} > \eta_0 \times \mathcal{T}_E$ (line 7), where $\mathcal{T}_E$ represents a threshold (e.g. 80%) and η_0 the entropy of the model after the first training, the model is iteratively pruned, perturbed and re-trained on Ξ_{train} (line 8) using the function PPTE defined in Appendix B.
Once $\eta_{\text{current}} < \eta_0 \times \mathcal{T}_E$, i.e. when the entropy is decreasing, we re-enable an early-stop criterion. Until the current performance this_acc on the validation set is lower than best_acc $\times \mathcal{T}_A$, where $\mathcal{T}_A$ represents a threshold (e.g. 80%), we continue to prune, perturb and re-train on Ξ_{train} (line 14). When the performance is below best_acc $\times \mathcal{T}_A$, the algorithm returns the model parameters $\boldsymbol{w}^{\mathcal{M}}$ (line 17).

I Experiments on more datasets

In this section we will present some results, with training on-the-wild (or in different terms, without injecting noise and using common and standard learning policies), on two main-stream datasets: Flowers-102 and ImageNet-1k. Moreover, as synthetic noise has clean structures which greatly enabled statistical analyses but often fails to model the real-world noise patterns, we also conducted experiments on CIFAR-100N, a dataset presented by (Wei et al. 2022), which is formed with CIFAR-100 training dataset equipped with human-annotated real-world noisy labels collected from Amazon Mechanical Turk.

I.1 CIFAR-100N

For CIFAR-100N, we used the same learning policy as for CIFAR-10 for the VGG-like architectures as reported in Tab. 4. We have employed a ResNet-18 as teacher model while a VGG-like model with $\gamma = 5$ and $\delta = 5$ as student.



(a) (b)

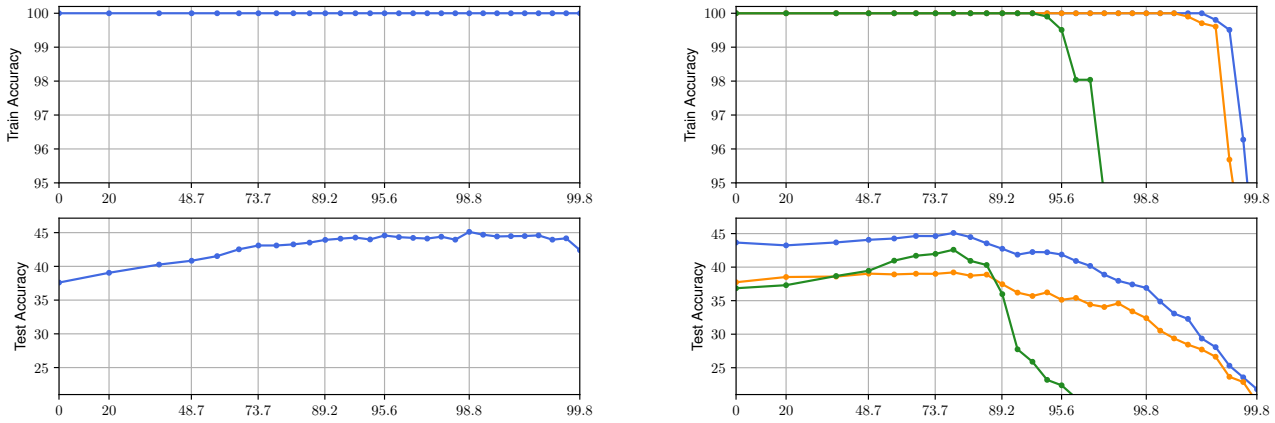
Figure 7: Performance on CIFAR-100N. **Left.** ResNet-18 **Right.** VGG-like model

In Fig. 7 we report the distillation results on CIFAR-100N. Like for CIFAR-10 and CIFAR-100, as already reported in Sec. 3.6, we consistently observe that, when sparsity increases, the student model, trained in a vanilla setup, exhibits the sparse double descent phenomenon. However, for the same architecture, we persistently notice that employing KD within the framework proposed in Sec. 4, whether the teacher is pruned or not (i.e. dense), the performance is enhanced and exhibits a monotonic behavior: the sparse double descent is dodged.
We also note that, in high sparsity regimes, the performance of the student model, trained within the KD framework becomes marginally below the vanilla setup. This behaviour can be explained by the fact that α and τ were not tuned for this dataset.

I.2 Flowers-102

For Flowers-102, we have employed exactly the same strategy as for CIFAR-10 for the VGG-like architectures, as reported in Tab. 4. We have employed for our experiment a ResNet-18 as teacher model while a VGG-like model with $\gamma = 5$ and $\delta = 4$ as student. Given the reduced size of the train set (consisting in 1020 samples, 10 per class), evidently, over-fitting is in general very easy for a sufficiently-large model. In order to further enhance this effect, we have decided not to use any dataset augmentation strategy. According to the results reported in Fig. 8, we clearly observe that, despite an evident overfit from both teacher and student, no double descent is visible, although the generalization gap is huge.[4] In this case, we hypothesize that the

[4]Employing transfer learning strategies it is in general possible to achieve extremely high performance on this performance, above 95%, on the test set. However, this is not the scope of the paper.



(a) (b)

Figure 8: Performance on Flowers-102. **Left.** ResNet-18 **Right.** VGG-like model

noise is irrelevant at the dataset's scale, and the model simply lacks the proper priors to learn the right set of features.

I.3 ImageNet

For ImageNet, we have employed the standard learning policy, consisting in SGD with lr decayed at epoch milestones 30 and 60, trained for 90 epochs with initial learning rate 0.1 and momentum 0.9, using batchsize 128. We have employed for our experiment a ResNet-50 as teacher model while a ResNet-18 as student. The result is reported in Tab. 14. Unsurprisingly, we do not observe a sparse double descent for any of the considered configurations: it is known that even ResNet-50 is an under-parametrized model with respect to ImageNet, and evidently we are already in the collapsed phase.

ζ	ResNet-18 (Vanilla)	ResNet-18 (KD with a dense ResNet-50)	ResNet-18 (KD with a 50% pruned ResNet-50)
0	68.89	69.57	69.75
0.5	69.28	69.82	69.63
0.75	68.98	69.03	68.93
0.875	67.50	66.04	66.08

Table 14: Test accuracy of ResNet-18 on ImageNet.

J Various visualizations for varying width & depth experiments

J.1 Study on the width for CIFAR-10 with $\varepsilon = 50\%$

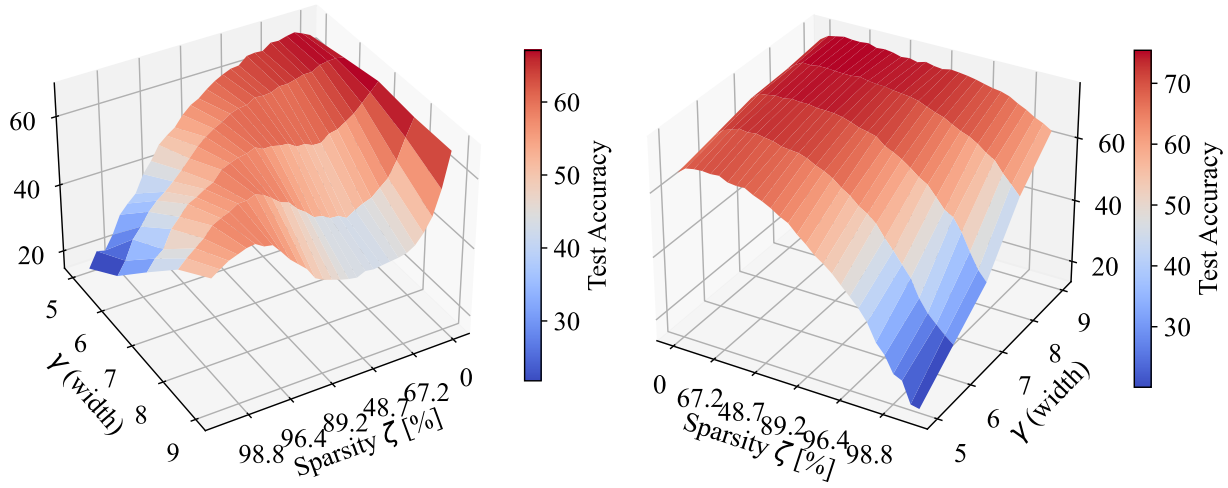


Figure 9: Test accuracy of the VGG-like varying γ on CIFAR-10 with $\varepsilon = 50\%$. **Left:** Vanilla Training. **Right:** Our proposed method.

J.2 Study on the Depth for CIFAR-10 with $\varepsilon = 50\%$

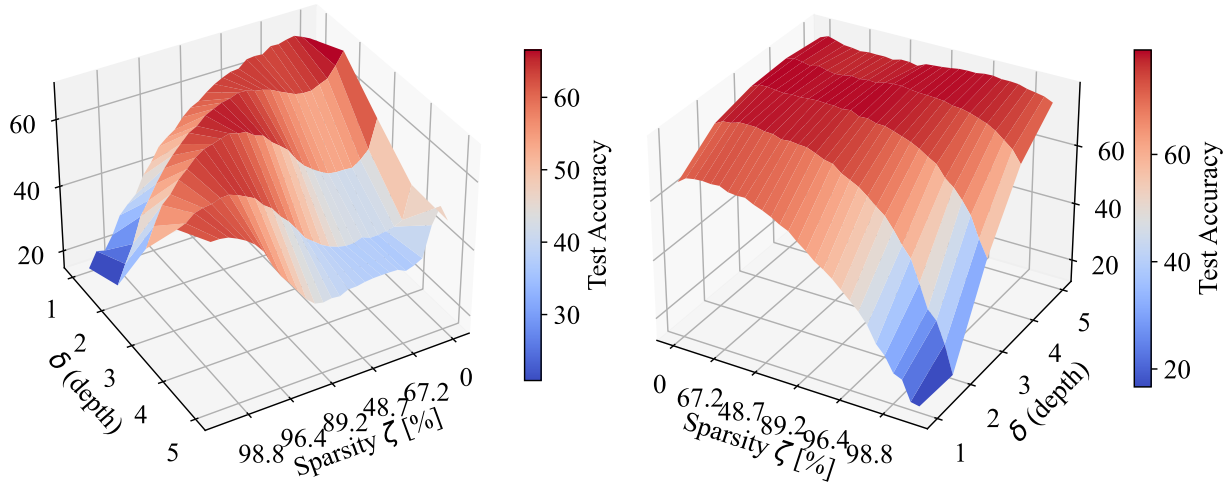


Figure 10: Test accuracy of the VGG-like varying δ on CIFAR-10 with $\varepsilon = 50\%$. **Left:** Vanilla Training. **Right:** Our proposed method.

J.3 Study on the width for CIFAR-10 with $\varepsilon = 20\%$

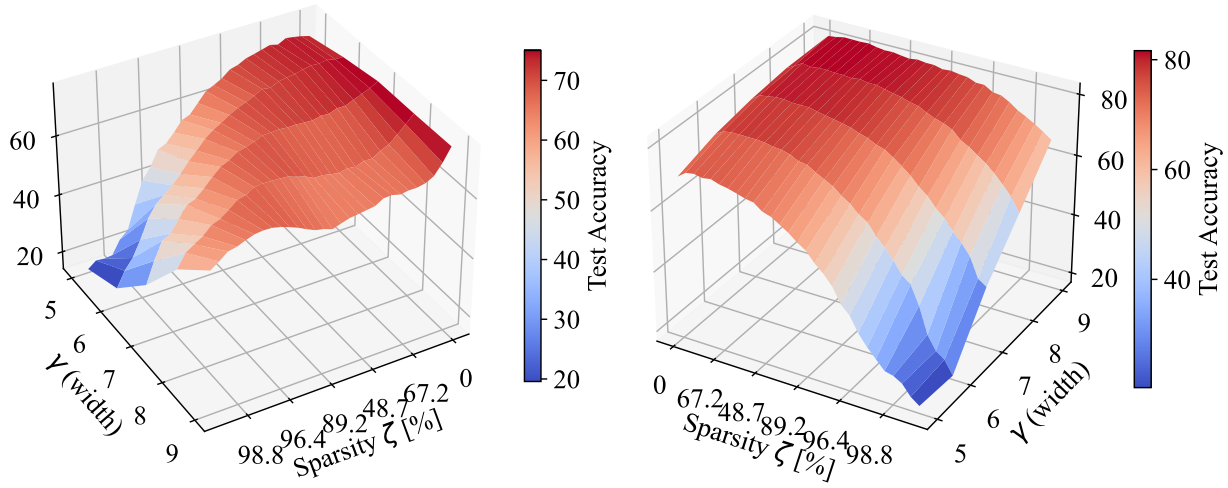


Figure 11: Test accuracy of the VGG-like varying γ on CIFAR-10 with $\varepsilon = 20\%$. **Left:** Vanilla Training. **Right:** Our proposed method.

J.4 Study on the depth for CIFAR-10 with $\varepsilon = 20\%$

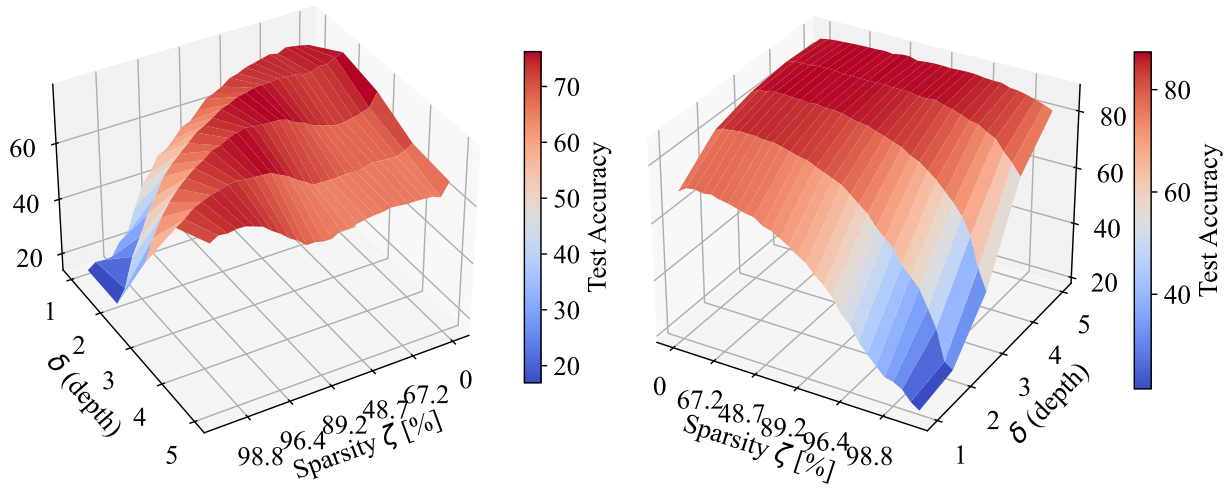


Figure 12: Test accuracy of the VGG-like varying δ on CIFAR-10 with $\varepsilon = 20\%$. **Left:** Vanilla Training. **Right:** Our proposed method.

J.5 Study on the width for CIFAR-10 with $\varepsilon = 10\%$

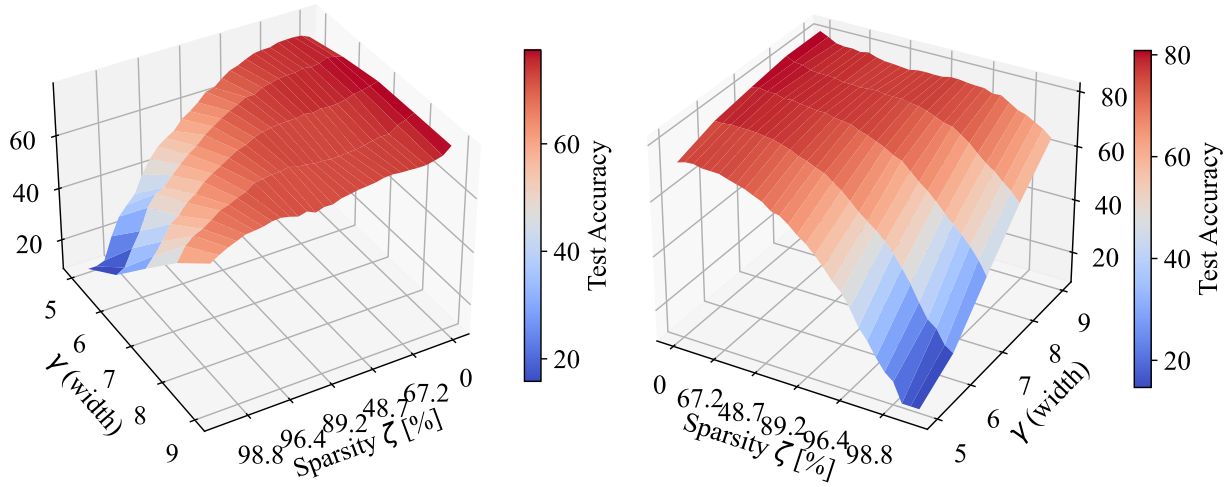


Figure 13: Test accuracy of the VGG-like varying γ on CIFAR-10 with $\varepsilon = 10\%$. **Left:** Vanilla Training. **Right:** Our proposed method.

J.6 Study on the depth for CIFAR-10 with $\varepsilon = 10\%$

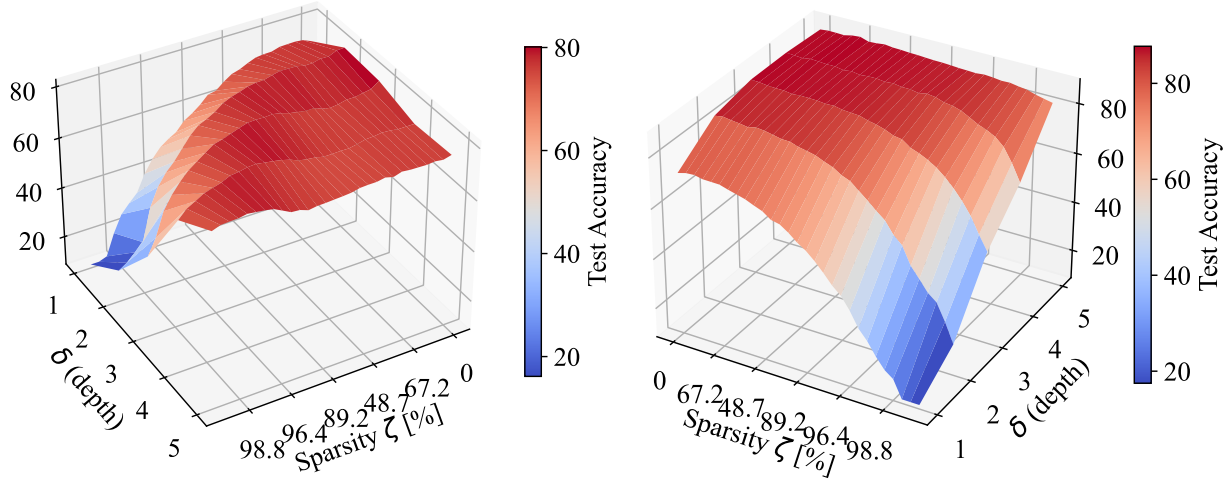


Figure 14: Test accuracy of the VGG-like varying δ on CIFAR-10 with $\varepsilon = 10\%$. **Left:** Vanilla Training. **Right:** Our proposed method.