On Local Overfitting and Forgetting in Deep Neural Networks

Uri Stern, Tomer Yaacoby and Daphna Weinshall

School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem 91904, Israel
ustern@gmail.com, tomer.yaacoby@mail.huji.ac.il, daphna@mail.huji.ac.il

Abstract

The infrequent occurrence of overfitting in deep neural networks is perplexing: contrary to theoretical expectations, increasing model size often enhances performance in practice. But what if overfitting does occur, though restricted to specific sub-regions of the data space? In this work, we propose a novel score that captures the forgetting rate of deep models on validation data. We posit that this score quantifies *local overfitting*: a decline in performance confined to certain regions of the data space. We then show empirically that *local overfitting* occurs regardless of the presence of traditional overfitting. Using the framework of deep over-parametrized linear models, we offer a certain theoretical characterization of forgotten knowledge, and show that it correlates with knowledge forgotten by real deep models. Finally, we devise a new ensemble method that aims to recover forgotten knowledge, relying solely on the training history of a single network. When combined with knowledge distillation, this method will enhance the performance of a trained model without adding inference costs. Extensive empirical evaluations demonstrate the efficacy of our method across multiple datasets, contemporary neural network architectures, and training protocols.

1 Introduction

Overfitting a training set is considered a fundamental challenge in machine learning. Theoretical analyses predict that as a model gains additional degrees of freedom, its capacity to fit a given training dataset increases. Consequently, there is a point where the model becomes too specialized for a particular training set, leading to an increase in its generalization error. In deep learning, one would expect to see *increased generalization error* as the number of parameters and/or training epochs increases. Surprisingly, even vast deep neural networks with billions of parameters seldom adhere to this expectation, and overfitting as a function of epochs is almost never observed (Liu et al. 2022). Typically, a significant increase in the number of parameters still results in enhanced performance, or occasionally in peculiar phenomena like the double descent in test error (Annavarapu 2021), see Section 3. Clearly, there exists a gap between our classical understanding of overfitting and the empirical results observed when training modern neural networks.

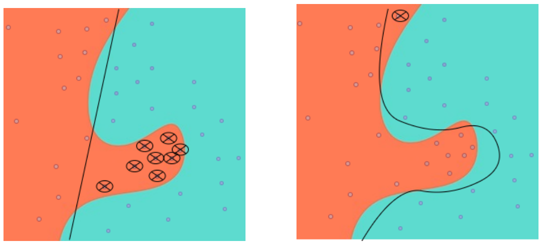


Figure 1: Local overfitting and forgetting in a binary problem, where blue and orange denote the different classes, and circles mark the validation set. The initial (left) and final (right) separators of a hypothetical learning method are shown, where ⊗ marks prediction errors. Clearly the final classifier has a smaller generalization error, but now one point at the top is ’forgotten’.

To bridge this gap, we present a fresh perspective on overfitting. Instead of solely assessing it through a decline in *validation accuracy*, we propose to monitor what we term *the model’s forget fraction*. This metric quantifies the portion of test data (or validation set) that the model initially classifies correctly but misclassifies as training proceeds (see illustration in Fig. 1). Throughout this paper we term the decline in test accuracy as ”forgetting”, to emphasize that the model’s ability to correctly classify portions of the data is reduced. In Section 3, we investigate various benchmark datasets, observing this phenomenon even in the absence of overfitting as conventionally defined, i.e., when test accuracy increases throughout. Notably, this occurs in competitive networks despite the implementation of modern techniques to mitigate overfitting, such as data augmentation and dropout. Our empirical investigation also reveals that forgetting of patterns occurs alongside the learning of new patterns in the training set, explaining why the traditional definition of overfitting fails to capture this phenomenon.

Formal investigation of the phenomenon of forgotten knowledge is challenging, particularly in the context of deep neural networks which are not easily amenable to formal analysis. Instead, in Section 4 we adopt the framework of over-parameterized deep linear networks. This framework involves non-linear optimization and has previously offered valuable insights into the learning processes of practical deep networks (Fukumizu 1998; Saxe, McClelland, and

Ganguli 2014; Arora, Cohen, and Hazan 2018; Arora et al. 2019; Hu, Xiao, and Pennington 2020). Within such models, employing gradient descent for learning reveals a straightforward and elegant characterization of the model's evolution (Hacohen and Weinshall 2022).

We expand upon this analysis, deriving an analytical description of the data points forgotten at each gradient descent step. As this analysis pertains specifically to deep linear models, it's crucial to correlate its findings with the forgotten knowledge in competitive neural networks. Intriguingly, when comparing these findings with the same image datasets utilized in our experiments, we observe significant overlap between the sets. This implies that the proposed theoretical characterization might offer partial insight into the phenomenon of forgotten knowledge and the underlying causes of local overfitting.

Based on the empirical observations reported in Section 3, we propose in Section 5 a method that can effectively reduce the forgetting of test data, and thus improve the final accuracy and reduce overfitting. More specifically, we propose a new prediction method that combines knowledge gained in different stages of training. The method delivers a weighted average of the class probability output vector between the final model and a set of checkpoints of the model from mid-training, where the checkpoints and their weights are chosen in an iterative manner using a validation dataset and our forget metric. The purpose is two-fold: First, an improvement upon the original model by our method will serve as another strong indication that models indeed forget useful knowledge in the late stages of training. Second, to provide a proof-of-concept that this lost knowledge can be recovered, even with methods as simple as ours.

In Section 6 we describe the empirical validation of our method in a series of experiments over image classification datasets with and without label noise, using various network architectures, including in particular modern networks over Imagenet. The results indicate that our method is universally useful and generally improves upon the original model, thus fulfilling its two mentioned goals. When compared with alternative methods that use the network's training history, our method shows comparable or improved performance, while being more general and easy to use (both in implementation and hyper-parameter tuning). Unlike some methods, it does not depend on additional training choices that require much more time and effort to tune the new hyper-parameters.

Our main contributions. (i) A novel perspective on overfitting, capturing the notion of *local overfitting*. (ii) Empirical evidence that overfitting occurs **locally** even without a decrease in overall generalization. (iii) Analysis of the relation between forgetting and PCA. (iv) A simple and effective method to reduce overfitting, and its empirical validation.

2 Related Work

Study of forgetting in prior work. Most existing studies examine the forgetting of training data, where certain training points are initially memorized but later forgotten. This typically occurs when the network cannot fully memorize the training set. In contrast, our work focuses on the *forgetting of validation points*, which arises when the network successfully memorizes the entire training set. Building on Arpit et al. (2017), who show that networks first learn "simple" patterns before transitioning to memorizing noisy data, we analyze the later stages of learning, particularly in the context of the double descent phenomenon. Another related but distinct phenomenon is "catastrophic forgetting" (McCloskey and Cohen 1989), which occurs in *continual learning* settings where the training data evolves over time—unlike the static training scenario considered here.

Ensemble learning. Ensemble learning has been studied extensively (see Polikar 2012; Ganaie et al. 2022; Yang, Lv, and Chen 2023). Our work belongs to a line of works called "implicit ensemble learning", in which only a single network is learned in a way that "mimics" ensemble learning (Srivastava et al. 2014). Utilizing checkpoints from the training history as a 'cost-effective' ensemble has also been considered. This was achieved by either considering the last epochs and averaging their probability outputs (Xie, Xu, and Chuang 2013), or by employing exponential moving average (EMA) on all the weights throughout training (Polyak and Juditsky 1992). The latter method does not always succeed in reducing overfitting, as discussed in (Izmailov et al. 2018).

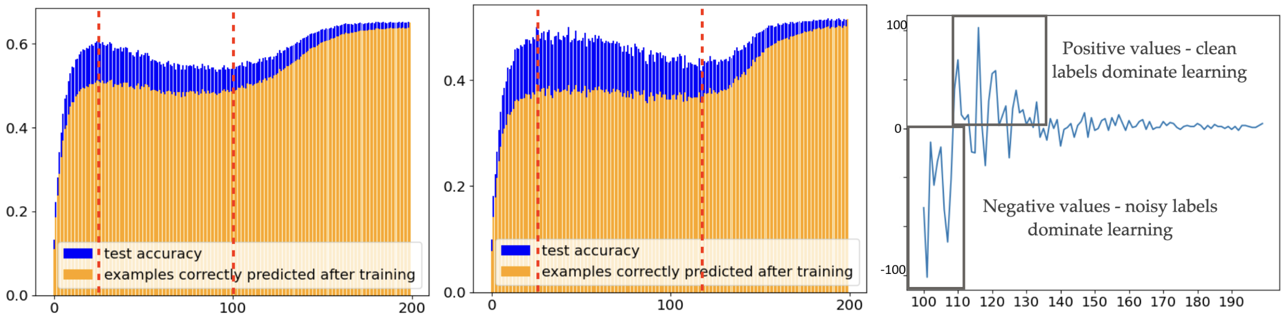
Several methods (Izmailov et al. 2018; Garipov et al. 2018; Huang et al. 2017) modify the training protocol to converge to multiple local minima, which are then combined into an ensemble classifier. While these approaches show promise (Noppitak and Surinta 2022), they add complexity to training and may even hurt performance (Guo, Jin, and Liu 2023). Our comparisons (see Table 3) demonstrate that our simpler method either matches or outperforms these techniques in all studied cases.

Ensemble methods can impose significant computational demands during inference, especially for large ensembles. Knowledge distillation (Hinton, Vinyals, and Dean 2015) addresses this challenge by training a single student model to replicate the ensemble's predictions, effectively eliminating the increased computational costs. This approach typically maintains the ensemble's performance and, in high-noise scenarios, may outperform the ensemble itself (Jeong and Chung 2024; Stern, Shwartz, and Weinshall 2024).

Studies of overfitting and double descent. Double descent with respect to model size has been studied empirically in (Belkin et al. 2019; Nakkiran et al. 2021), while epoch-wise double descent (which is the phenomenon analyzed here) was studied in (Stephenson and Lee 2021; Heckel and Yilmaz 2020). These studies analyzed when and how epoch-wise double descent occurs, specifically in data with label noise, and explored ways to avoid it (sometimes at the cost of reduced generalization). Essentially, our research identifies a similar phenomenon in data without label noise. It is complementary to the study of "benign overfitting", e.g., the fact that models can achieve perfect fit to the train data while still obtaining good performance over the test data.

3 Overfitting Revisited

The textbook definition of overfitting entails the co-occurrence of increasing train accuracy and decreasing gen-



(a) Cifar100, 20% symmetric noise

(b) TinyImagenet, 20% symmetric noise

(c) Cifar100, 20% symmetric noise

Figure 2: (a)-(b): Blue denotes test accuracy (Y-axis) as a function of epoch (X-axis). Among those correctly recognized in each epoch e, orange denotes the fraction that remains correctly recognized at the end. The test accuracy (blue) shows a clear double ascent of accuracy, which is much less pronounced in the orange curve. During the decrease in test accuracy - the range of epochs between the first and second dashed red vertical lines - the large gap between the blue and orange plots indicates the fraction of test data that has been correctly learned in the first ascent and then forgotten, without ever being re-learned in the later recovery period of the second ascent. (c): The difference between the number of clean and noisy datapoints at each epoch during the second ascent of test accuracy (the epochs after the second dashed red vertical line), counting datapoints with large loss only. Positive (negative) value indicates that clean (noisy) datapoints are more dominant in the corresponding epoch.

eralization. Let $acc(e, S)$ denote the accuracy over set S in epoch e - some epoch in mid-training, E the total number of epochs, and T the test[1] dataset. Using test accuracy to approximate generalization, this implies that overfitting occurs at epoch e when $acc(e, T) \geq acc(E, T)$.

We begin by investigating the hypothesis that portions of the test data T may be forgotten by the network during training. When we examine the 'epoch-wise double descent', which frequently occurs during training on datasets with significant label noise, we indeed observe that a notable forgetting of the test data coincides with the memorization of noisy labels. Here, forgetting serves as an objective indicator of overfitting. When we further examine the training of modern networks on standard datasets (devoid of label noise), where overfitting (as traditionally defined) is absent, we discover a similar phenomenon (though of weaker magnitude): *the networks still appear to forget certain sub-regions of the test population*. This observation, we assert, signifies a significant and more subtle form of overfitting in deep learning.

Local overfitting. Let M_e denote the subset of the test data *mislabeled* by the network at some epoch e. We define below two scores L_e and F_e:

$$F_e = \frac{acc(e, M_E) \cdot |M_E|}{|T|}, \; L_e = \frac{acc(E, M_e) \cdot |M_e|}{|T|} \quad (1)$$

The *forget fraction* F_e represents the fraction of test points correctly classified at epoch e but misclassified by the final model. L_e represents the fraction of test points misclassified at epoch e but correctly classified by the final model. The relationship $acc(E, T) = acc(e, T) + L_e - F_e$ follows[2]. In line with the classical definition of overfitting, if $L_e < F_e$, overfitting occurs since $acc(E, T) < acc(e, T)$.

But what if $L_e \geq F_e \; \forall e$? By its classical definition *overfitting does not occur* since the test accuracy increases continuously. Nevertheless, there may still be local overfitting as defined above, since $F_e > 0$ indicates that data has been forgotten even if $L_e \geq F_e$.

Reflections on the epoch-wise double descent. Epoch-wise double descent (see Fig. 2) is an empirical observation (Belkin et al. 2019), which shows that neural networks can improve their performance even after overfitting, thus causing *double descent in test error* during training (or *double-ascent in test accuracy*). This phenomenon is characteristic of learning from data with label noise, and is strongly related to overfitting since the dip in test accuracy co-occurs with the memorization of noisy labels.

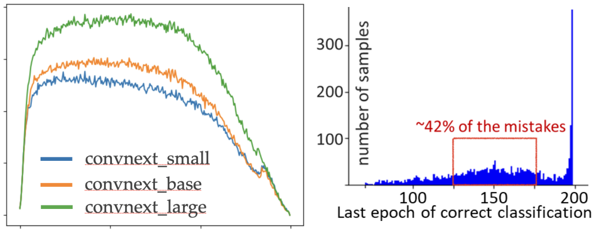
We examine the behavior of score F_e in this context and make a novel observation: when we focus on the fraction of data correctly classified by the network during the second rise in test accuracy, we observe that the data newly memorized during these epochs often differs from the data forgotten during the overfitting phase (the dip in accuracy). In fact, most of this data has been previously misclassified (see Figs. 2a-2b). Fig. 2c further illustrate that during the later stages of training on data with label noise, the majority of the data being memorized is, in fact, data with clean labels, which explains the second increase in test accuracy. It thus appears that epoch-wise double descent is caused by the *simultaneous learning* of general (but hard to learn) patterns from clean data, and irrelevant features of noisy data.

Forgetting in the absence of label noise When training deep networks on visual benchmark datasets without added label noise, double descent rarely occurs, if ever. In contrast, we observe that local overfitting, as captured by our new score F_e, commonly occurs.

To show this, we trained various neural networks (ConvNets: Resnet, ConvNeXt; Visual transformers: ViT, MaxViT) on various datasets (CIFAR-100, TinyImagenet,

[1] Below, 'test set' and 'validation set' are used interchangeably.

[2] $acc(E, T) - L_e = acc(e, T) - F_e$ is the fraction of test points correctly classified in both e and E.



(a) ImageNet (b) TinyImagenet, resnet 18

Figure 3: (a) The F_e score (1) of ConvNeXt trained on Imagenet (Y-axis) as a function of epoch (X-axis), showing 3 network sizes: small → blue, base → orange and large → green. Accuracy remained consistent across all network sizes, while F_e increases with network size. (b) Within the set of wrongly classified test points after training, we show the last epoch in which an example was classified correctly.

Imagenet) using a variety of optimizers (SGD, AdamW) and learning rate schedulers (cosine annealing, steplr). In Fig. 3a we report the results, showing that all networks forget some portion of the data during training as in the label noise scenario, even if the test accuracy never decreases. Fig. 3b demonstrates that this effect is not simply due to random fluctuations: many test examples that are incorrectly classified post training have been correctly classified during much of the training. These results are connected to overfitting in Fig. 3a: when investigating larger models and/or relatively small amounts of train data, which are scenarios that are expected to increase overfitting based on theoretical considerations, we see larger *forget fraction* F_e (see Figs. 5-6 in App. A[3] for additional results).

In summary, we see that neural networks can, and often will, "forget" significant portions of the test population as their training proceeds. In a sense, the networks *are* overfitting, but this only occurs at some limited sub-regions of the world. The reason this failing is not captured by the classical definition of overfitting is that the networks continue to learn new general patterns simultaneously. In Section 5 we discuss *how we can harness this observation to improve the network's performance*.

4 Forgotten Knowledge: Theory & Exps

To gain insight into the nature of knowledge forgotten while training a deep model with Gradient Descent (GD), we analyze over-parameterized deep linear networks trained by GD. These models are constructed through the concatenation of linear operators in a multi-class classification scenario: $\boldsymbol{y} = W_L \cdot \ldots \cdot W_1\mathbf{x}$, where $\mathbf{x} \in \mathbb{R}^d$. For simplicity, we focus on the binary case with two classes, suggesting that similar qualitative outcomes would apply to the more general multi-class model. Accordingly, we redefine the objective function as follows:

$$\min_{W_1,\ldots,W_L} \sum_{i=1}^{n} \|W_L \cdot \ldots \cdot W_1\mathbf{x}_i - y_i\|^2 \qquad (2)$$

Above the matrices $\{W_l\}_{l=1}^L$ represent the $2D$ matrices corresponding to L layers of a deep linear network, and points $\{\mathbf{x}_i\}_{i=1}^n$ represent the training set with labeling function $y_i = \pm 1$ for the first and second classes, respectively. Note that $\boldsymbol{w} = \prod_{l=L}^1 W_l$ is a row vector that defines the resulting separator between the classes. The classifier is defined as: $f(\mathbf{x}) = \text{sign}\left(\prod_{l=L}^1 W_l\mathbf{x}\right)$ for $\mathbf{x} \in \mathbb{R}^d$.

Preliminaries. Let $\boldsymbol{w}^{(n)} = \prod_{l=L}^1 W_l^{(n)}$ represent the separator after n GD steps, where $\boldsymbol{w}^{(n)} \equiv [w_1^{(n)}, \ldots, w_d^{(n)}] \in \mathbb{R}^d$. For convenience, we rotate the data representation so that its axes align with the eigenvectors of the data's covariance matrix. Hacohen and Weinshall (2022) showed that the convergence rate of the j^{th} element of $\boldsymbol{w}$ with respect to n is exponential, governed by the corresponding j^{th} eigenvalue:

$$w_j^{(n)} \approx \lambda_j^n w_j^{(0)} + [1 - \lambda_j^n]w_j^{opt}, \qquad \lambda_j = 1 - \gamma s_j L \quad (3)$$

Here, $\boldsymbol{w}^{(0)}$ denotes the separator at initialization, $\boldsymbol{w}^{opt}$ denotes the optimal separator (which can be derived analytically from the objective function), s_j represents the j^{th} singular value of the data, and γ is the learning rate. Notably, while $\boldsymbol{w}^{opt}$ is unique, the specific solution at convergence $\{W_l^{(\infty)}\}_{l=1}^L$ is not.

4.1 Forget Time in Deep Linear Models

Let Λ denote diag$(\{\lambda_j\})$ - a diagonal matrix in $\mathbb{R}^{d \times d}$, and I the identity matrix. It follows from (3) that

$$\boldsymbol{w}^{(n)} \approx \boldsymbol{w}^{(0)}\Lambda^n + \boldsymbol{w}^{opt}[I - \Lambda^n] \qquad (4)$$

We say that a point is forgotten if it is classified correctly at initialization, but not so at the end of training. Let $\mathbf{x}$ denote a forgotten datapoint, and let N denote the number of GD steps at the end of training. Since by definition $f(\mathbf{x}) = \text{sign}(\boldsymbol{w}^{(n)}\mathbf{x})$, it follows that $\mathbf{x}$ is forgotten iff $\{\boldsymbol{w}^{(0)}y\mathbf{x} > 0\}$ and $\{\boldsymbol{w}^{(N)}y\mathbf{x} < 0\}$.

Let us define the forget time of point $\mathbf{x}$ as follows:

Definition 1 (Forget time)**.** *GD iteration $\hat{n}$ that satisfies*

$$\begin{aligned} &\boldsymbol{w}^{(\hat{n})}y\mathbf{x} \leq 0 \\ &\boldsymbol{w}^{(n)}y\mathbf{x} > 0 \qquad \forall n < \hat{n} \end{aligned} \qquad (5)$$

Claim 1. *Each forgotten point has a finite forget time $\hat{n}$.*

Proof. Since $\{\boldsymbol{w}^{(0)}y\mathbf{x} > 0\}$ and $\{\boldsymbol{w}^{(N)}y\mathbf{x} < 0\}$, (5) follows by induction. □

Note that Def 1 corresponds with the *Forget time* seen in deep networks (cf. Fig. 3b). The empirical investigation of this correspondence is discussed in App. B (see Fig. 7).

To characterize the time at which a point is forgotten, we inspect the rate with which $F(n) = \boldsymbol{w}^{(n)}y\mathbf{x}$ changes with n. We begin by assuming that the learning rate γ is infinitesimal, so that terms of magnitude $O(\gamma^2)$ can be neglected.

[3] All references to appendices are to be found in the complete archived version (Stern, Yaacoby, and Weinshall 2024).

Using (4) and the Taylor expansion of λ_j from (3)

$$
\begin{aligned}
F(n) &\approx \left(\boldsymbol{w}^{(0)} - \boldsymbol{w}^{opt}\right) \Lambda^n y\mathbf{x} + \boldsymbol{w}^{opt} y\mathbf{x} \\
&= \boldsymbol{w}^{opt} y\mathbf{x} + \sum_{j=1}^{d} (w_j^{(0)} - w_j^{opt}) \lambda_j^n y x_j \\
&= \boldsymbol{w}^{opt} y\mathbf{x} + \sum_{j=1}^{d} (w_j^{(0)} - w_j^{opt}) [1 - n\gamma s_j L + O(\gamma^2)] y x_j \\
&= \boldsymbol{w}^{(0)} y\mathbf{x} - n\gamma L \sum_{j=1}^{d} (w_j^{(0)} - w_j^{opt}) y s_j x_j + O(\gamma^2)
\end{aligned}
$$

It follows that

$$
\frac{dF(n)}{dn} = -\gamma y L \sum_{j=1}^{d} (w_j^{(0)} - w_j^{opt}) s_j x_j + O(\gamma^2) \qquad (6)
$$

Discussion. Recall that $\{s_j\}$ is the set of singular values, ordered such that $s_1 \geq s_2 \geq \cdots \geq s_d$, and x_j is the projection of point $\mathbf{x}$ onto the j^{th} eigenvector. From (6), the rate at which a point is forgotten, if at all, depends on vector $[s_j x_j]_j$, in addition to the random vector $\boldsymbol{w}^{(0)} - \boldsymbol{w}^{\text{opt}}$ and label y. All else being equal, a point will be forgotten faster if the length of its spectral decomposition vector $[x_j]$ is dominated by its first components, indicating that most of its mass is concentrated in the leading principal components.

4.2 Spectral Properties of Forgotten Images

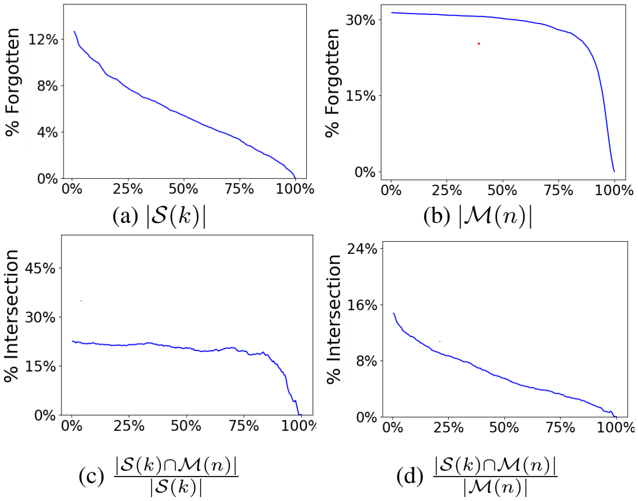
When working with datasets of natural images, where it has been shown that the singular values decrease rapidly at an approximately exponential rate (Hyvärinen, Hurri, and Hoyer 2009), the role of the singular values becomes even more pronounced. Hacohen and Weinshall (2022) argued that in the limiting case, the components of the separating hyperplane $\boldsymbol{w}^{opt}$ will be learned sequentially, one at a time. In essence, the model first captures the projection of $\boldsymbol{w}^{opt}$ onto the data's leading eigenvector, then onto the subsequent eigenvectors in order. For similar considerations, this reasoning also holds in the multi-class scenario.

This analysis suggests that PCA of the raw data governs the learning of the linear separator. We therefore hypothesize that forgotten points with substantial projections onto the leading principal components are more likely to be forgotten early, and vice versa. To empirically test this prediction, we must first establish some key definitions.

Let $\boldsymbol{W}^{opt} \in \mathbb{R}^{c \times d}$ denote the optimal solution of the multi-class linear model with c classes and the L_2 loss. Let $\boldsymbol{W}(k)$ denote the projection of $\boldsymbol{W}^{opt}$ on the first k principal components of the raw data.

Definition 2. *Let $\mathcal{S}(k)$ denote the set of points that are correctly classified by $\boldsymbol{W}(k')$ for some $k' > k$, but incorrectly classified by $\boldsymbol{W}^{opt}$. Similarly, let $\mathcal{M}(n)$ denote the set of points correctly classified by the trained deep model after $n' > n$ epochs, but incorrectly classified by the final model.*

To empirically investigate the prediction above, we correlate the two sets $\mathcal{S}(k)$ and $\mathcal{M}(n)$ after establishing correspondence $n = \alpha k + \beta$ between the ranges of indices k and n. We examined this correlation using the CIFAR100 dataset, a linear model trained using the images' RGB representation, and the corresponding deep model from the experiments reported in Section 6. Interestingly, the respective sets $\mathcal{S}(k)$ and $\mathcal{M}(n)$ show significant correlation, as seen in Fig. 4. Since deep networks also learn a representation, we repeated the experiment with alternative learned feature spaces, obtaining similar results (see Fig. 8 in App. B).



(a) $|\mathcal{S}(k)|$ (b) $|\mathcal{M}(n)|$

(c) $\frac{|\mathcal{S}(k) \cap \mathcal{M}(n)|}{|\mathcal{S}(k)|}$ (d) $\frac{|\mathcal{S}(k) \cap \mathcal{M}(n)|}{|\mathcal{M}(n)|}$

Figure 4: Empirical results, correlating the sets of examples forgotten during the training of a DNN and those forgotten during the training of a deep linear network. Note in (d) that early on, roughly $\frac{1}{6}$ of the points to be forgotten by our deep model are also forgotten by the deep linear model.

5 Recover Forgotten Knowledge: Algorithm

In Section 3 we showed that neural networks often achieve better performance in mid-training on a subset of the test data, even when the test accuracy is monotonically increasing with training epochs. Here we aim to integrate the knowledge obtained in mid- and post-training epochs, during inference time, in order to improve performance. To this end we must determine: (i) which versions of the model to use; (ii) how to combine them with the post-training model; and (iii) how to weigh each model in the final ensemble.

Choosing an early epoch of the network. Given a set of epochs $\{1, \ldots, E\}$ and corresponding forget rates $\{F_e\}_e$, we first single out the model n_A obtained at epoch $A = argmax_{e \in \{1,\ldots,E\}} F_e$. This epoch is most likely to correctly fix mistakes of the model on "forgotten" test data.

Combining the predictors. Next, using validation data we determine the relative weights of the two models - the final model n_E, and the intermediate model n_A with maximal forget fraction. Since the accuracy of n_E is typically much higher than n_A, and in order not to harm the ensemble's performance, we expect to assign n_E a higher weight.

Improving robustness. To improve our method's robustness to the choice of epoch A, we use a window of epochs around A, denoted by $\{n_{A-w}, \ldots, n_A, \ldots, n_{A+w}\}$. The vectors of probabilities computed by each checkpoint are averaged before forming an ensemble with n_E. In our experiments, we use a fixed window $w = 1$, achieving close to optimal results as verified in the ablation study (see App. G.9).

Iterative selection of models. As we now have a new predictor, we can find another alternative predictor from the training history that maximizes accuracy on the data misclassified by the new predictor, in order to combine their knowledge as described. This can be repeated iteratively, until no further improvement is achieved.

Choosing hyper-parameters. In order to compute F_e and assign optimal model weights and window size, we use a validation set, which is a part of the labeled data not shown to the model during initial training. This is done **post training** as it has no influence over the training process, and thus *doesn't incur additional training cost*. We follow common practice, and show in App. G.1 that after optimizing these hyper-parameters, it is possible to retrain the model on the complete training set while maintaining the same hyper-parameters. The performance of our method thus trained is superior to alternative methods trained on the same data.

Pseudo-code for our method. We name our method **K**nowledge**F**usion (KF), and provide its pseudo-code in Alg 1. There, we call functions that: (i) calculate the forget value per epoch on some validation data, given the predictions at each epoch (**calc_early_forget**); and (ii) calculate the probability of each class for a given example and a list of predictors (**get_class_probs**).

Algorithm 1: Knowledge Fusion (KF)

```
Input: Checkpoints of trained model $\{n_0,...,n_E\}$, w, test-pt $x$
Output: prediction for $x$
$\{A_1,...,A_k\}$, $\{\varepsilon_1, ..., \varepsilon_k\} \leftarrow$ calc_early_forget($\{n_0,...,n_E\}$)
$prob \leftarrow \mathbf{get\_class\_probs}[E]$
for $i \leftarrow 1$ to $k$ do
    $prob_A \leftarrow mean(\mathbf{get\_class\_probs}[A_i - w : A_i + w])$
    $prob \leftarrow \varepsilon_i * prob_A + (1 - \varepsilon_i) * prob$
end for
$prediction \leftarrow \mathbf{argmax}(prob)$
Return $prediction$
```

Knowledge distillation post-processing. The proposed method enhances the performance of any trained model with a minor increase in training costs. However, ensemble classifiers incur higher inference costs. To address this, knowledge distillation can be employed with a further increase in training costs, to deliver a single model that *achieves performance comparable to the ensemble while maintaining inference costs comparable to the original model*.

6 Empirical Evaluation

6.1 Main Results

In this section we evaluate the performance of our method as compared to the original predictor, i.e. the network after training, and other baselines. We use various image classification datasets, neural network architectures, and training schemes. The main results are presented in Tables 1-3, followed by a brief review of our extensive ablation study and additional comparisons in Section 6.2. All references to appendices below are to be found in the complete archived version of this paper (Stern, Yaacoby, and Weinshall 2024).

Specifically, in Table 1 we report results while using multiple architectures trained on CIFAR-100, TinyImagenet and Imagenet, with different learning rate schedulers and optimizers. For comparison, we report the results of both the original predictor and some baselines. Additional results for scenarios connected to overfitting are shown in Table 2 and App. F, where we test our method on these datasets with injected symmetric and asymmetric label noise (see App. E), as well as on a real label noise dataset (Animal10N). Note that, as customary, the label noise exists only in the train data while the test data remains clean for model evaluation.

In Table 3 and App. F we compare our method to additional methods that adjust the training protocol itself, using both clean and noisy datasets. We employ these methods using the same network architecture as our own, after a suitable hyper-parameter tuning.

In each experiment we use half of the *test data* for validation, to compute our method's hyper-parameters (the list of alternative epochs and $\{\varepsilon_i\}$), and then test the result on the remaining test data. The accuracy reported here is only on the remaining test data, averaged over three random splits of validation and test data, using different random seeds. In App. G.1 we report results on the original train/test split, where a subset of the training data is set aside for hyper-parameter tuning. As customary, these same parameters are later used with models trained on the full training set, demonstratively without deteriorating the results.

Baselines Our method incurs the training cost of a single model, and thus, following the methodology of (Huang et al. 2017), we compare ourselves to methods that require the same amount of training time. The first group of baselines includes methods that do not alter the training process:

- **Single network**: the original network, after training.
- **Horizontal ensemble** (Xie, Xu, and Chuang 2013): this method uses a set of epochs at the end of the training, and delivers their average probability outputs (with the same number of checkpoints as we do).
- **Fixed jumps**: this baseline was used in (Huang et al. 2017), where several checkpoints of the network, equally spaced through time, are taken as an ensemble.

The second group includes methods that *alter* the training protocol. While this is not a directly comparable set of methods, as they focus on a complementary way to improve performance, we report their results in order to further validate the usefulness of our method. This group includes *Snapshot ensemble* (Huang et al. 2017), *Stochastic Weight Averaging* (SWA) (Izmailov et al. 2018) and *Fast Geometric Ensembling* (FGE) (Garipov et al. 2018), see details in App. D. Comparisons to additional baselines that are relevant to resisting overfitting, including early stopping and test time augmentation, are discussed in App. G.5. Full implementation details are provided in App. E.

6.2 Ablation Study

We conducted an extensive ablation study in order to investigate the limitations, and some practical aspects, of our

Method/**Dataset**	**CIFAR-100**	**TinyImagenet**	**Imagenet**			
architecture	Resnet18	Resnet18	Resnet50	ConvNeXt large	ViT16 base	MaxViT tiny
single network	78.07 ± .28	64.95 ± .24	75.74 ± .14	82.92 ± .11	79.16 ± .1	82.51 ± .15
horizontal (i)	78.15 ± .17	64.89 ± .18	**76.46 ± .14**	**83.13 ± .1**	79.11 ± .1	82.77 ± .1
fixed jumps (i)	78.04 ± .23	66.54 ± .35	75.5 ± .09	82.37 ± .1	78.67 ± .08	**83.38 ± .1**
KF (ours) (i)	**78.33 ± .08**	**66.98 ± .37**	75.88 ± .14	**83.18 ± .16**	**79.93 ± .11**	**83.34 ± .04**
horizontal (∞)	78.23 ± .17	65.11 ± .3	**76.42 ± .1**	83.02 ± .06	79.53 ± .13	82.93 ± .14
fixed jumps (∞)	**79.17 ± .08**	68.24 ± .38	75.72 ± .18	**83.86 ± .06**	79.11 ± .13	**83.78 ± .15**
KF (ours) (∞)	**79.13 ± .14**	**68.5 ± .36**	**76.52 ± .16**	**83.96 ± .09**	**80.34 ± .08**	**83.81 ± .14**
improvement	**1.05 ± .14**	**3.54 ± .14**	**.78 ± .04**	**1.03 ± 13**	**1.17 ± .08**	**1.29 ± .02**

Table 1: Mean (over random validation/test splits) test accuracy (in percent) and standard error on image classification datasets, comparing our method and baselines described in the text. The last row shows the improvement of the best performer over the single network. Suffixes: (i) denotes a limited budget scenario, in which we use our method in a non-iterative manner; (∞) denotes the unlimited budget scenario, where we use our full iterative version. In each case, the baselines employ the same number of checkpoints as our method.

Method/**Dataset**	**Animal10N**	**CIFAR-100 asym**		**CIFAR-100 sym**		**TinyImagenet**	
% label noise	8%	20%	40%	20%	40%	20%	40%
single network	85.9 ± .3	67.1 ± .5	49.4 ± .3	65.4 ± .3	56.9 ± .1	56.2 ± .2	49.8 ± .3
fixed jumps (∞)	87.1 ± .4	73.9 ± .1	59.9 ± .6	72.8 ± .1	66.5 ± .1	60.0 ± .8	54.16 ± .3
horizontal (∞)	86.3 ± .3	73.4 ± .1	58.5 ± .1	71.1 ± .38	65.2 ± .1	59.3 ± .3	51.7 ± .2
KF (ours) (∞)	**87.8 ± .4**	**74.2 ± .1**	**62.1 ± .5**	**72.8 ± .1**	**67.0 ± .1**	**62.8 ± .2**	**57.0 ± .5**
improvement	**1.9 ± .4**	**7.1 ± .6**	**12.6 ± .2**	**7.4 ± .4**	**10.1 ± .1**	**6.6 ± .1**	**7.2 ± .1**

Table 2: Mean test accuracy (in percent) and standard error of Resnet 18, comparing our method and the baselines on datasets with large label noise and significant overfitting. We include the Animal10N dataset, which has innate label noise.

Method/**Dataset**	**CIFAR-100**	**Animal10N**	**CIFAR-100 asym**		**CIFAR-100 sym**	
% label noise	0%	8%	20%	40%	20%	40%
FGE (∞)	78.9 ± .4	86.5 ± 0.6	67.1 ± .2	48.1 ± .3	66.5 ± .1	52.1 ± .1
SWA (∞)	78.8 ± .1	**88.1 ± .2**	66.6 ± .1	46.9 ± .2	65.6 ± .4	50.0 ± .1
snapshot (∞)	78.4 ± .1	86.8 ± .3	72.1 ± .4	52.8 ± .6	70.8 ± .5	63.8 ± .2
KF (ours) (∞)	**79.3 ± .2**	**87.8 ± .4**	**74.2 ± .1**	**62.1 ± .5**	**72.8 ± .1**	**67.0 ± .1**

Table 3: Mean test accuracy of Resnet18, using for baseline methods that alter the training procedure.

method. Due to space limitation, we only provide here a brief overview of the results, and postpone the full description to App. G. The results can be summarized as follows:

(i) §G.1 shows that a separate validation set is not really necessary for the method to work well. (ii) §G.2 investigates how many checkpoints are needed for the method to be effective, showing that only $5 - 10\%$ of the past checkpoints are sufficient. (iii) §G.3 investigates the added value of our method when using only a partial hyper-parameter search, which leads to sub-optimal training. Interestingly, our method is shown to be even more beneficial in the sub-optimal regime, with a smaller gap between the optimal and sub-optimal networks. (iv) §G.4 shows that our method is effective in a transfer learning scenario, when using a pre-trained network. (v) §G.5 shows that our method outperforms Exponential-Moving-Average (EMA), early stopping and test time augmentation. (vi) §G.6 shows that our method's benefit increases as the number of parameters grows. (vii) §G.7 shows that much of the improvement of a regular ensemble of independent networks can often be obtained by using our method at a much lower cost. (viii) §G.8 shows that our method does not have negative effects on the model's fairness. (ix) §G.9 shows that using a window of size w=1 is both necessary and near optimal.

7 Summary and Conclusions

We revisited the problem of *overfitting* in deep learning, proposing to track the forgetting of validation data in order to detect local overfitting. We connected our new perspective with the *epoch wise double descent* phenomenon, empirically extending its scope while demonstrating that a similar effect occurs in benchmark datasets with clean labels. Inspired by these new empirical observations, we constructed a simple yet general method to improve classification at inference time. We then empirically demonstrated its effectiveness on many datasets and modern network architectures. The method improves modern networks by around 1% accuracy over Imagenet, and is especially useful in some transfer learning settings where its benefit is large and its overhead is very small. Most importantly, the success of the method to improve upon the original model shows that indeed models forget useful knowledge at the late stages of learning, and serves as a proof of concept that recovering this knowledge can be useful to improve performance.

Acknowledgments

This work was supported by grants from the Israeli Council of Higher Education and the Gatsby Charitable Foundation.

References

Annavarapu, C. S. R. 2021. Deep learning-based improved snapshot ensemble technique for COVID-19 chest X-ray classification. *Applied Intelligence*, 51: 3104–3120.

Arora, S.; Cohen, N.; Golowich, N.; and Hu, W. 2019. A Convergence Analysis of Gradient Descent for Deep Linear Neural Networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Arora, S.; Cohen, N.; and Hazan, E. 2018. On the Optimization of Deep Networks: Implicit Acceleration by Overparameterization. In *International Conference on Machine Learning*, 244–253.

Arpit, D.; Jastrzebski, S.; Ballas, N.; Krueger, D.; Bengio, E.; Kanwal, M. S.; Maharaj, T.; Fischer, A.; Courville, A.; Bengio, Y.; et al. 2017. A closer look at memorization in deep networks. In *International conference on machine learning*, 233–242. PMLR.

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*, 116(32): 15849–15854.

Fukumizu, K. 1998. Effect of batch learning in multilayer neural networks. *Gen*, 1(04): 1E–03.

Ganaie, M. A.; Hu, M.; Malik, A.; Tanveer, M.; and Suganthan, P. 2022. Ensemble deep learning: A review. *Engineering Applications of Artificial Intelligence*, 115: 105151.

Garipov, T.; Izmailov, P.; Podoprikhin, D.; Vetrov, D. P.; and Wilson, A. G. 2018. Loss surfaces, mode connectivity, and fast ensembling of dnns. *Advances in neural information processing systems*, 31.

Guo, H.; Jin, J.; and Liu, B. 2023. Stochastic weight averaging revisited. *Applied Sciences*, 13(5): 2935.

Hacohen, G.; and Weinshall, D. 2022. Principal components bias in over-parameterized linear models, and its manifestation in deep neural networks. *Journal of Machine Learning Research*, 23(155): 1–46.

Heckel, R.; and Yilmaz, F. F. 2020. Early stopping in deep networks: Double descent and how to eliminate it. *arXiv preprint arXiv:2007.10099*.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the Knowledge in a Neural Network. arXiv:1503.02531.

Hu, W.; Xiao, L.; and Pennington, J. 2020. Provable Benefit of Orthogonal Initialization in Optimizing Deep Linear Networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Huang, G.; Li, Y.; Pleiss, G.; Liu, Z.; Hopcroft, J. E.; and Weinberger, K. Q. 2017. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*.

Hyvärinen, A.; Hurri, J.; and Hoyer, P. O. 2009. *Natural image statistics: A probabilistic approach to early computational vision.*, volume 39. Springer Science & Business Media.

Izmailov, P.; Podoprikhin, D.; Garipov, T.; Vetrov, D.; and Wilson, A. G. 2018. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*.

Jeong, H.; and Chung, H. W. 2024. Understanding Self-Distillation and Partial Label Learning in Multi-Class Classification with Label Noise. *arXiv preprint arXiv:2402.10482*.

Liu, Z.; Mao, H.; Wu, C.-Y.; Feichtenhofer, C.; Darrell, T.; and Xie, S. 2022. A ConvNet for the 2020s. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

McCloskey, M.; and Cohen, N. J. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, 109–165. Elsevier.

Nakkiran, P.; Kaplun, G.; Bansal, Y.; Yang, T.; Barak, B.; and Sutskever, I. 2021. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12): 124003.

Noppitak, S.; and Surinta, O. 2022. dropCyclic: snapshot ensemble convolutional neural network based on a new learning rate schedule for land use classification. *IEEE Access*, 10: 60725–60737.

Polikar, R. 2012. Ensemble learning. *Ensemble machine learning: Methods and applicannavarapu2021deepations*, 1–34.

Polyak, B. T.; and Juditsky, A. B. 1992. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4): 838–855.

Saxe, A. M.; McClelland, J. L.; and Ganguli, S. 2014. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1): 1929–1958.

Stephenson, C.; and Lee, T. 2021. When and how epochwise double descent happens. *arXiv preprint arXiv:2108.12006*.

Stern, U.; Shwartz, D.; and Weinshall, D. 2024. United We Stand: Using Epoch-wise Agreement of Ensembles to Combat Overfit. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38(13), 15075–15082.

Stern, U.; Yaacoby, T.; and Weinshall, D. 2024. On Local Overfitting and Forgetting in Deep Neural Networks. *arXiv preprint arXiv:2412.12968*.

Xie, J.; Xu, B.; and Chuang, Z. 2013. Horizontal and vertical ensemble with deep representation for classification. *arXiv preprint arXiv:1306.2759*.

Yang, Y.; Lv, H.; and Chen, N. 2023. A survey on ensemble learning under the era of deep learning. *Artificial Intelligence Review*, 56(6): 5545–5589.