Forget Me Not: Fighting Local Overfitting with Knowledge Fusion and Distillation

Uri Stern*, Eli Corn* and Daphna Weinshall
School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem 91904, Israel
Email: {ustern@gmail.com, eli.corn@mail.huji.ac.il, daphna@mail.huji.ac.il}

***Abstract*—Overfitting in deep neural networks occurs less frequently than expected. This is a puzzling observation, as theory predicts that greater model capacity should eventually lead to overfitting – yet this is rarely seen in practice. But what if overfitting does occur, not globally, but in specific sub-regions of the data space? In this work, we introduce a novel score that measures the *forgetting rate* of deep models on validation data, capturing what we term *local overfitting*: a performance degradation confined to certain regions of the input space. We demonstrate that local overfitting can arise even without conventional overfitting, and is closely linked to the double descent phenomenon.**

Building on these insights, we introduce a two-stage approach that leverages the training history of a single model to recover and retain forgotten knowledge: first, by aggregating checkpoints into an ensemble, and then by distilling it into a single model of the original size, thus enhancing performance without added inference cost. Extensive experiments across multiple datasets, modern architectures, and training regimes validate the effectiveness of our approach. Notably, in the presence of label noise, our method – *Knowledge Fusion* followed by *Knowledge Distillation* – outperforms both the original model and independently trained ensembles, achieving a rare win-win scenario: reduced training and inference complexity.

***Index Terms*—Overfitting, Double Descent, Knowledge Distillation, Noisy Labels, Ensemble Learning**

I. 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 at which 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 [1]. 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 [2], see Section III. Clearly, there exists a gap between our classical understanding of overfitting and the empirical results observed when training modern neural networks.

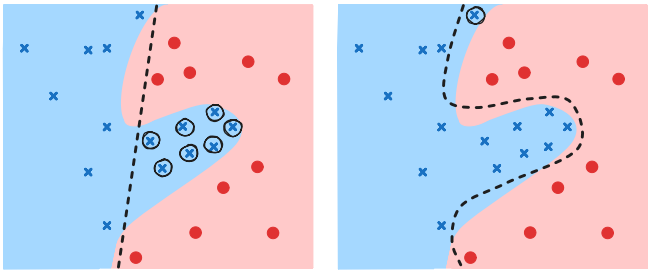


Fig. 1. Local overfitting and forgetting in a binary classification problem, where blue and red denote the two classes. Test samples from the blue class are marked with X's, and test samples from the red class with circles. The initial (left) and final (right) decision boundaries of a hypothetical learning method are shown, with prediction errors circled in black. Although the final classifier achieves lower generalization error, one test point that was initially classified correctly becomes misclassified by the end, illustrating forgetting.

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 III, 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.

Based on the empirical observations reported in Section III, we propose in Sections IV-V a two phase method that can

*Equal contribution

TABLE I
SUMMARY OF METHOD COMPLEXITY AND PERFORMANCE

Method	Complexity		Ranking	
	Training	**Inference**	**Label Noise**	**Clean Data**
Single Model	Low	Low	5	5
Independent Ensemble	High	High	2-3	**1**
Distilled Independent Ensemble	High	Low	2-3	3-4
Knowledge Fusion (KF)	Low	High	4	3-4
Distilled KF Ensemble	**Low**	**Low**	**1**	2

Each method is ranked from 1 (best) to 5 (worst) based on its average performance across multiple datasets (see Table IV). Note that distilled KF offers the best performance-efficiency trade-off, ranking first with label noise and second on clean data, while maintaining low complexity.

effectively reduce the forgetting of test data, thus improving the final accuracy and reduce overfitting, while maintaining inference efficiency similar to a single model.

In phase one (Section IV), we introduce a new prediction method, *Knowledge Fusion (KF)*, that combines knowledge gained in different stages of training. This method delivers a weighted average of the class probability output between the final model and a set of checkpoints of the model from mid-training. The checkpoints and their respective weights are selected iteratively using a validation dataset and our forget metric. The purpose of this phase is two-fold: First, it improves upon the original model, serving as another strong indication that models indeed forget useful knowledge in the late stages of training. Second, it provides a proof-of-concept that this lost knowledge can be recovered.

In phase two (Section V), we address the increased inference complexity introduced by phase 1. Since phase 1 effectively forms an ensemble by aggregating predictions from multiple checkpoints, its memory and compute demands during inference are significantly higher than those of a single model. This overhead can hinder deployment in resource-constrained environments. To solve this, in phase 2 we *condense* the ensemble into a single, compact model through two approaches: (i) **Knowledge Distillation (KD)**, where a student network is trained to mimic the outputs of the KF ensemble; and (ii) **Weight Averaging**, a simpler method that directly averages the weights of selected checkpoints. These approaches aim to retain the performance benefits of KF while restoring the efficiency of inference by a single model.

Empirical results, presented in Section VI, validate the effectiveness of our approach across multiple image classification datasets, with and without label noise. Using various network architectures, including modern networks on ImageNet, the results demonstrate that our method is universally beneficial, consistently improving upon the original model. To ensure a fair comparison, we compare against the original model, which also leverages a validation set to select the best performing checkpoint for early stopping, since this aligns with our method's use of a validation set.

When compared to alternative methods that leverage the network's training history, our approach delivers comparable or superior performance, while retaining lower inference complexity. When compared to alternative ensemble methods of higher training **and** inference complexity, the distilled condensed model retains (with a few exceptions), and in some cases surpasses, the performance of alternative methods. This provides a win-win scenario: reduced training and inference costs without sacrificing generalization.

Table I summarizes these results by ranking the aforementioned core methods across two criteria: performance on noisy and clean datasets, and their training/inference complexity. Rankings (1 = best, 5 = worst) are computed by first ranking the methods separately on each dataset and then averaging those ranks, based on accuracy scores from Table IV of Section VI. Notably, the *distilled KF ensemble* achieves the best performance on noisy datasets and strong performance on clean datasets, while maintaining **low** complexity in both training and inference. In contrast, while the *independent ensemble* performs best on clean data, it incurs significantly higher computational costs. These results underscore the distilled KF ensemble as offering the most favorable trade-off between performance and efficiency across both clean and noisy data settings.

The theoretical investigation of forgotten knowledge is presented in Section VII, which adopts the framework of over-parameterized deep linear networks. In the preliminary conference version of this work [3], it is shown that the theoretical analysis of forgotten points correlates significantly with empirical results, offering insights into local overfitting and the phenomenon of forgotten knowledge.

Our main contributions

(i) A novel perspective on overfitting, formalizing the notion of *local overfitting*. (ii) Empirical evidence that overfitting occurs **locally** even without a decrease in overall generalization. (iii) An effective method to reduce local overfitting, and its empirical validation. This method retains the low training and inference costs of a single model, while preserving superior performance.

II. 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 even when the network successfully memorizes the entire training set. Another related but distinct phenomenon is "catastrophic forgetting" [4], which occurs in *continual learning* settings where the training data evolves over time, differently from the static training scenario considered here.

Ensemble learning

Ensemble learning has been studied extensively see [5]. Our methodology is rooted in "implicit ensemble learning", in which only a single network is trained in a way that "mimics" ensemble learning [6]. 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 [7], or by employing exponential moving average (EMA) on all the weights throughout training [8]. The latter method does not always succeed in reducing overfitting, as discussed in [9].

Several methods modify the training protocol to converge to multiple local minima, which are then combined into an ensemble classifier. While these approaches show promise [10], they add complexity to training and may even hurt performance [11]. Our comparisons (see Table V) demonstrate that our simpler method either matches or outperforms these techniques in all studied cases.

Studies of overfitting and double descent

Double descent with respect to model size has been studied empirically in [12, 13], while epoch-wise double descent (which is the phenomenon analyzed here) was studied in [14, 15]. 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.

Knowledge distillation

Knowledge Distillation (KD) [16] is a technique in which a smaller "student" model is trained to replicate the behavior of a larger "teacher" model by learning from its output predictions. In traditional KD, the teacher model provides "soft targets" for the student to learn from, instead of relying solely on hard labels. This enables the student to capture a more nuanced understanding of the data distribution, which often results in improved performance.

Recall that ensemble learning can significantly increase computational costs at inference time. Knowledge distillation, where the ensemble acts as the teacher and a single model serves as the student, offers a way to reduce these costs, although often with some loss in performance. Interestingly, in certain settings such as those with high label noise, the distilled model can even outperform the ensemble itself [17, 18].

Self-distillation is a technique in which a model learns from its own prior predictions, enhancing performance without the need for a separate teacher model. This technique has been shown to enhance the original model's performance, particularly in noisy environments [19, 20, 21], with several theoretical explanations proposed for its effectiveness.

Somewhat counter-intuitively, recent studies have shown that higher-performing models do not necessarily make better teachers in knowledge distillation. Accordingly, [22] show that an intermediate checkpoint from a teacher's training trajectory can serve as a more effective teacher than the final, fully trained model. They also demonstrate that ensembles built from a random sample of checkpoints, while underperforming independently trained ensembles, are better able to guide student models. [23] expands on this idea by proposing to use a self-attention mechanism to adaptively weight intermediate models during distillation.

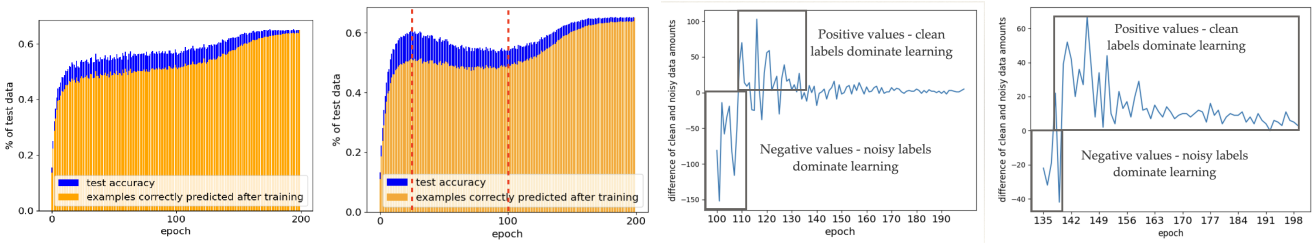
Some of our empirical findings mirror these insights, particularly the superior performance of ensembles composed of intermediate checkpoints over independently trained models in knowledge distillation. While [23] primarily addresses the problem of how to weight each randomly sampled checkpoint, we focus on a complementary challenge - the effective selection of checkpoints - guided by the notion of forgotten knowledge. Their proposed adaptive weighting technique further adds complexity to the training. Consequently, and given the absence of publicly available code, our empirical comparisons are limited to a classical checkpoint sampling method (see Table VIII). Nonetheless, we note that the adaptive weighting of checkpoints can be incorporated into our approach as a complementary enhancement.

III. Overfitting Revisited

The textbook definition of overfitting entails the co-occurrence of increasing train accuracy and decreasing generalization. Let $acc(e, S)$ denote the accuracy over set S in epoch e, where E is the total number of epochs, and T is the test dataset.[1] When approximating generalization by test accuracy, overfitting is said to occur at epoch e if $acc(e, T) \geq acc(E, T)$.

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.

[1]Throughout this paper, the terms "test set" and "validation set" are used interchangeably.



(a) Cifar100, original data (b) Cifar100, 20% sym noise (c) Cifar100, 20% sym noise (d) TinyImageNet, 20% sym noise

Fig. 2. (a)–(b): Blue denotes test accuracy. Among those correctly recognized in each epoch e, orange denotes the fraction that remains correctly recognized at the end. The test accuracy (the blue curve) in (b) 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. This gap also appears in (a), demonstrating that forgetting occurs even without double ascent. (c)–(d): 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) values indicate that clean (noisy) datapoints are more dominant in the corresponding epoch.

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 $L_e \geq F_e$.

Local overfitting vs local underfitting

When $F_e > 0$, two interpretations are possible, depending on how the training error behaves near the forgotten points. If the local training error continues to decrease, the effect is consistent with *local overfitting*. If instead the local training error increases together with the test error, the more appropriate interpretation is *local underfitting*.

To distinguish between these cases, we examined how training and test accuracies evolve around the *forgotten* test points. As described in Appendix E, we embed all samples in a fixed semantic space and, for each forgotten point, collect all training examples within a radius r to form its local neighborhood. We then track accuracy on the forgotten point and on its nearby training samples across epochs. Figure 3 presents the aggregated curves, averaged over all forgotten points and their neighborhoods.

The results show a clear pattern. Accuracy on the neighboring training samples continues to improve, while accuracy on the forgotten points declines. Since local training accuracy does not deteriorate together with local test accuracy, this behavior does not support local underfitting. Instead, it indicates that the model becomes increasingly specialized to nearby training samples and loses generalization to test examples in the same regions. This provides direct evidence that forgetting reflects *local overfitting*.

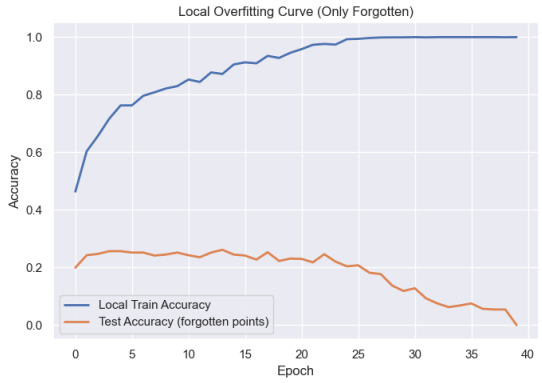


Fig. 3. Aggregated local train and test accuracy around forgotten instances. For each forgotten point, accuracy is computed over training samples within a radius-defined neighborhood and over the forgotten point itself. Across training, accuracy on local training samples rises, while accuracy on the forgotten points decreases, indicating local overfitting (see Appendix E).

Reflections on the epoch-wise double descent

Epoch-wise double descent (see Fig. 2) is an empirical observation [12], which shows that neural networks can improve their performance even after overfitting, thus causing double descent in test error during training (note that we show the corresponding *double-ascent in test accuracy*). This

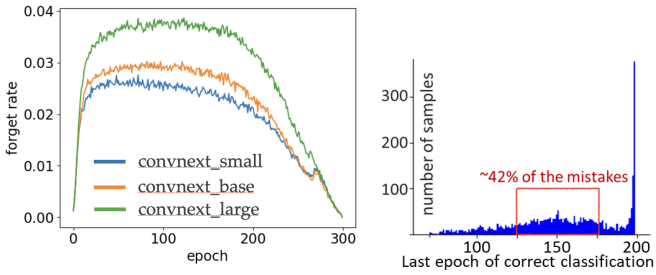
[2] $acc(E,T) - L_e = acc(e,T) - F_e$ is the fraction of test points correctly classified in both e and E.

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 Fig. 2b). Figs. 2c-2d 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, overfitting rarely occurs. In contrast, we observe that local overfitting, as captured by our new score F_e, commonly occurs, as demonstrated in Fig. 2a.



(a) ImageNet (b) TinyImageNet, ResNet18

Fig. 4. (a) F_e score (1) of ConvNeXt trained on ImageNet, 3 network sizes: small → blue, base → orange, and large → green. Accuracy remains consistent across all network sizes, while F_e increases with the network size. (b) Within the set of wrongly classified test points after training, we show the last epoch in which each sample was classified correctly.

To investigate this phenomenon, we trained various neural networks (ConvNets: Resnet, ConvNeXt; Visual transformers: ViT, MaxViT) on various datasets (CIFAR-100, TinyImageNet, ImageNet) using a variety of optimizers (SGD, AdamW) and learning rate schedulers (cosine annealing, steplr). Results on ImageNet are shown in Fig. 3a; the results of additional experiments on other datasets and configurations can be found in Appendix A.

Across all settings, we observe that networks consistently forget a subset of the data during training, in a manner similar to what is observed with label noise, even when test accuracy continues to improve. Fig. 4b (see additional results and analysis in Figs. 2b-2c in Appendix A) 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. 4a: 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, the *forget fraction* F_e is larger.

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 feature space. 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 IV we discuss *how we can harness this observation to improve the network's performance*.

IV. Knowledge Fusion (KF)

In Section III 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. In this section, we introduce an approach to integrate the knowledge obtained in both mid- and post-training epochs during inference time, 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.

We call our method **KnowledgeFusion** (KF). Below, we outline the core components of the approach and provide further implementation details in the following subsections.

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

b) 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.

c) Improving robustness: To improve our method's robustness to the choice of epoch A, we use a window of epochs around A, denoted by $\{n_{A-w}, ..., n_A, ..., n_{A+w}\}$. 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 Section VI.

d) 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.

e) 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*. Following common practice, we show in the ablation study below that after optimizing these hyper-parameters, it is possible to retrain the model on the complete training set while maintaining the same hyper-parameters. T,he performance of KF thus trained is superior to alternative methods trained on the same data.

A. Hyper-parameter Calculation

Algorithm 1 outlines the hyper-parameter search phase of our method, Knowledge Fusion, which is performed once after training. It is responsible for selecting the most relevant epochs and their associated weights.

Algorithm 1 KF - hyper-parameter calculation

Input: all past checkpoints during training of the neural network $\{ n_0,...,n_E \}$, w, and validation data V.
Output: list of alternative epochs and their weights.
$class\_probs \leftarrow$ **get_class_probs**($\{ n_0,...,n_E \}$, V)
$prob \leftarrow class\_probs[E]$
explore = $\{ n_0,...,n_E \}$
Alternative_epochs = $\{\}$
epsilons = $\{\}$
while explore is not empty **do**
  $F \leftarrow$ **calc_forget_per_epoch**($prob$, $class\_probs$)
  $altEp \leftarrow$ **argmax**($F[explore]$)
  Alternative_epochs.**append(altEp)**
  explore.**remove(altEp - 1, altEp, altEp + 1)**
  for $\varepsilon \in \{0, 0.01, ..., 1\}$ **do**
    $prob_A \leftarrow \mathbf{mean}(class\_probs[A_i - w : A_i + w])$
    $combProb \leftarrow \varepsilon \cdot prob_A + (1 - \varepsilon) \cdot prob$
    if validAcc(combProb) $\geq$ **validAcc**(prob) **then**
      $best\_prob \leftarrow combProb$
      $best\_epsilon \leftarrow \varepsilon$
    end if
  end for
  $prob \leftarrow best\_prob$
  epsilons.**append**($best\_epsilon$)
end while
Return Alternative_epochs, epsilons

B. Prediction

Algorithm 2 shows the prediction step of Knowledge Fusion (KF). Using the epochs and weights computed in Algorithm 1, it combines the predictions of selected checkpoints to produce the final output for a test sample. As in Algorithm 1, the **get_class_probs** function is utilized to obtain the class probabilities for a given example and a list of predictors.

Algorithm 2 Knowledge Fusion (KF) Prediction

Input: Checkpoints of trained model $\{ n_0,...,n_E \}$, selected epochs $\{ A_1,...,A_k \}$, associated weights $\{ \varepsilon_1,...,\varepsilon_k \}$, w, test-pt x.
Output: prediction for x.
$prob \leftarrow$ **get_class_probs**$[E]$
for $i \leftarrow 1$ **to** k **do**
  $prob_A \leftarrow$ **mean**(**get_class_probs**$[A_i - w : A_i + w]$)
  $prob \leftarrow \varepsilon_i \cdot prob_A + (1 - \varepsilon_i) \cdot prob$
end for
$prediction \leftarrow$ **argmax**$(prob)$
Return $prediction$

C. Discussion

Empirical results (see Section VI) show that KF consistently performs on par with, or better than, strong baselines. This holds across a range of architectures, datasets, and training conditions.

At the same time, the checkpoint selection step in Algorithm 1 relies on a greedy procedure, which offers no theoretical guarantees of optimality. Although this strategy works well in practice, exploring alternatives such as global optimization or submodular-style selection procedures as potential alternatives may offer promising directions for future work.

A key limitation of our method lies in its computational cost at inference. Since predictions involve aggregating outputs from multiple model checkpoints, runtime and memory usage scale linearly with the number of components in the ensemble. This cost can be prohibitive in resource-constrained deployment settings, making the method less practical despite its accuracy benefits.

To address this, the next section introduces strategies for condensing the KF ensemble into a single, compact model. These strategies aim to preserve the performance advantages of the ensemble while enabling efficient inference, thereby broadening the practical applicability of KF.

V. Condensing Knowledge Fusion

The KF method is designed to enhance model accuracy while successfully mitigating local overfitting, as demonstrated in Section VI. It can be applied as a post-processing step to any trained model with only a minor increase in training costs. However, because KF produces an ensemble classifier, its adoption results in a substantial increase in inference time.

To address this issue, we explore an additional post-processing step that compresses the ensemble classifier into a single model, thereby reducing inference costs to a level comparable with the baseline. Below, we discuss two such

methods that entail a trade-off between implementation complexity and model accuracy, as demonstrated in Section VI.

A. Method 1: Distilled KF

Our first approach leverages Knowledge Distillation (KD). This technique involves training a smaller, more efficient model (the student) to replicate the behavior of a larger, more complex model (the teacher). In the context of Knowledge Fusion, the teacher model is represented by the KF ensemble, while the student is a single model distilled from this ensemble. The goal is to combat local overfitting by leveraging the combined knowledge that KF has accumulated from earlier stages of training, while fully eliminating the memory overhead and increased inference time associated with maintaining a complete ensemble.

In our experiments presented later in this section, the distilled network was designed to match the size of the original single network. This ensures a fair comparison in terms of resource usage during inference and similar training times. We conducted multiple experiments across various settings, including different noise levels and datasets, all of which demonstrated the effectiveness of this approach. Our results indicate that applying Knowledge Distillation to Knowledge Fusion not only enhances computational efficiency but also preserves the high accuracy of the ensemble, confirming its practicality for large-scale tasks.

Brief Overview of Knowledge Distillation We now describe the standard implementation of Knowledge Distillation (KD), introduced earlier in Section II. KD typically relies on two loss terms: a soft target loss that compares teacher and student predictions, and a true label loss computed using ground-truth labels.

The soft target loss is computed by comparing the logits of the teacher model with the logits of the student model. Both sets of logits are scaled by a temperature parameter T, which smooths the logits and generates softened probability distributions. This allows the student to learn from the teacher's more informative, probabilistic outputs.

Neural networks typically produce class probabilities by applying the *softmax* function to the output logits. Given a vector of logits $z \in \mathbb{R}^C$, the temperature-scaled softmax function converts each logit z_c into a probability by comparing it with the other logits:

$$\text{softmax}\left(\frac{z}{T}\right)_c = \frac{e^{z_c/T}}{\sum_{j=1}^{C} e^{z_j/T}}.$$

This results in a smooth probability distribution over the C classes, with higher temperature T producing a more uniform distribution.

Given teacher logits $\mathbf{z}_\text{T}$ and student logits $\mathbf{z}_\text{S}$, we define their soft probability vectors as

$$\hat{\mathbf{p}}_\text{T} = \text{softmax}\left(\frac{\mathbf{z}_\text{T}}{T}\right), \quad \hat{\mathbf{p}}_\text{S} = \text{softmax}\left(\frac{\mathbf{z}_\text{S}}{T}\right).$$

The soft target loss is defined as the Kullback-Leibler (KL) divergence between the teacher and student distributions:

$$\mathcal{L}_\text{soft} = T^2 \cdot \text{KL}\left(\hat{\mathbf{p}}_\text{T} \,\|\, \hat{\mathbf{p}}_\text{S}\right).$$

The factor of T^2 ensures that the gradient magnitudes remain comparable across different values of T, as suggested by [16].

Beyond the soft target loss, there is also the student's original loss, denoted as $\mathcal{L}_\text{label}$, which represents the **true label loss**. In our case, it is the cross-entropy loss computed using the student model's logits and the true labels $\mathbf{y}$:

$$\mathcal{L}_\text{label} = \text{cross\_entropy}(\mathbf{z}_\text{S}, \mathbf{y})$$

The total loss is a weighted sum of the soft target loss and the true label loss with hyperparameter $\alpha \in [0, 1]$:

$$\mathcal{L}_\text{total} = \alpha \cdot \mathcal{L}_\text{soft} + (1 - \alpha) \cdot \mathcal{L}_\text{label} \tag{2}$$

This formulation allows the student model to retain knowledge from the teacher model (via soft targets), while ensuring that it also performs well on the ground-truth labels.

B. Method 2: Averaged KF

As an alternative to augmenting KF with Knowledge Distillation, and a baseline to assess the necessity of distillation, we explored a simpler approach to reducing the complexity of the KF ensemble. This method directly combines the weights of the ensemble models into a single model.

The procedure mirrors the Knowledge Fusion (KF) algorithm, where we iteratively select model checkpoints with high forgetting scores and incorporate them into the current model. However, rather than creating an ensemble of models, we average the network weights of the selected models to produce a single set of weights. This method allows for predictions with only a single forward pass, thus reducing the computational and memory costs compared to an ensemble.

As in the ensemble approach, each model's contribution to the final mechanism is weighted according to its performance. The optimal weight for each model is determined by testing the same set of ε values used in the Knowledge Fusion algorithm on the validation set and selecting the one that yields the best performance.

VI. Empirical Evaluation

In this section, we evaluate the performance of our method, with and without phase 2, against the original predictor (i.e., the early-stopped trained network) and other baselines. Full implementation details are provided in Appendix B.

Our experiments strictly maintain disjoint training, validation, and test partitions. Full details of the protocol and checks ensuring data isolation are provided in Appendix D.

A. Baselines

KF incurs the training cost of a single model, and thus, following the methodology of [24], we begin by comparing our method 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, which in this case refers to the early-stopped version of the model, where the validation set is used to choose the best stopping point.
- **Self-Adaptive Training (SAT)** [25]: a robust-training method designed specifically for noisy-label regimes. SAT gradually replaces the hard labels with soft targets obtained from an exponential moving average of the model's own predictions; to the best of our knowledge, it is the current state-of-the-art approach for the label noise setting.
- **Horizontal ensemble** [7]: 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 [24], 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** [24]: The network is trained in multiple "cycles," each ending with a sharp learning rate increase to escape local minima and promote convergence to diverse solutions for ensembling.
- **Stochastic Weight Averaging** (SWA) [9]: The network is first trained conventionally, then continuing with a constant or cyclic learning rate to reach multiple local minima, whose weights are averaged. For fair comparison, we adopt their setup: using our strategy for 75% of the epochs, followed by SWA for the remaining 25%.
- **Fast Geometric Ensembling** (FGE) [26]: Like SWA, where this method averages the output probabilities of diverse models rather than their weights. Computational budgets are matched as described above.

TABLE II
COMPARISON ON CLEAN DATASETS

Method/**Dataset** architecture	**CIFAR-100** Resnet18	**TinyImageNet** Resnet18
baseline network	$78.85 \pm .32$	$65.30 \pm .36$
SAT	$77.84 \pm .01$	$63.10 \pm .20$
Ensembles		
horizontal	$78.42 \pm .17$	$65.61 \pm .31$
fixed jumps	$78.21 \pm .46$	$67.41 \pm .21$
KF	$79.36 \pm .17$	$68.33 \pm .17$
Condensed Models		
Averaged KF	$78.48 \pm .26$	$65.17 \pm .23$
Distilled KF	$\mathbf{80.29 \pm .16}$	$\mathbf{69.96 \pm .15}$
improvement	$1.44 \pm .36$	$4.66 \pm .39$

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 baseline network. Bolded values indicate the best performer in each column.

B. Results

Main Results Table II reports the performance obtained by all variants of our method on CIFAR-100 and TinyImageNet. For comparison, we report the results of both the original model and the aforementioned baselines. Additional results for scenarios associated with overfitting are shown in Table III, where we test our method on these datasets with injected symmetric and asymmetric label noise (see Appendix B), and CIFAR-100N - a real world dataset with label noise. As is customary, label noise is present only in the training data, while the test data remains clean for evaluation.

The results in Tables II and III reveal a rather surprising finding: the distilled KF model not only matches but often **outperforms** the KF ensemble of phase 1, on both clean and noisy data. While this may seem counterintuitive—since distillation typically aims to mimic a stronger teacher—it aligns with prior findings that student models can occasionally surpass their teachers in accuracy and robustness [19, 20, 21]. In our case, this is likely due to the synergy between two known effects: the strength of checkpoint-based ensembles [see 23, 22], and the regularizing benefits of distillation.

Independent Ensembles Since KF yields an ensemble classifier at inference time, we conclude its evaluation by comparing it to a standard independent ensemble and its distilled variant, see Table IV. We observe that a standard ensemble, which involves training multiple independent networks, often outperforms KF. However, this comes at a substantial increase in training costs, which may render the approach impractical in some scenarios.

After applying distillation to condense the respective ensembles, this performance gap disappears—a single condensed KF model outperforms the condensed standard ensemble. Surprisingly, in the presence of label noise, the condensed KF can even surpass the original, independent ensemble classifier, achieving the best performance with significantly reduced training and inference complexity.

After distillation, a consistent pattern emerges from Table IV: in most cases, distilling the independent ensemble leads to a drop in performance, whereas the KF-based ensemble improves. This not only narrows the performance gap between the two approaches, but often allows the distilled KF ensemble to outperform its independent counterpart. This

TABLE III
COMPARISON ON NOISY DATASETS

Method/**Dataset**	**CIFAR-100N**	**CIFAR-100**				**TinyImageNet**	
architecture / % noise	40 %	20 % asym	40 % asym	20 % sym	40 % sym	20 % sym	40 % sym
baseline network	$54.53 \pm .44$	$67.07 \pm .42$	$49.95 \pm .53$	$65.16 \pm .25$	$59.03 \pm .68$	$56.25 \pm .47$	$49.83 \pm .3$
SAT	$53.18 \pm .43$	$74.43 \pm .07$	$63.26 \pm .13$	$71.97 \pm .10$	$66.67 \pm .09$	$60.59 \pm .17$	$54.01 \pm .20$
Ensembles							
fixed jumps	$60.38 \pm .57$	71.46 ± 1.91	$59.9 \pm .6$	$72.8 \pm .1$	$66.5 \pm .1$	$60 \pm .8$	$54.16 \pm .3$
horizontal	$55.14 \pm .57$	$73.04 \pm .47$	$58.5 \pm .1$	$71.1 \pm .38$	$65.2 \pm .1$	$57.74 \pm .47$	$51.7 \pm .2$
KF	$63 \pm .39$	$74.58 \pm .28$	$63.37 \pm .53$	$72.68 \pm .15$	$67.43 \pm .12$	$\mathbf{62.22 \pm .38}$	$55.66 \pm .37$
Condensed Models							
Averaged KF	$62.67 \pm .48$	$70.72 \pm .44$	$56.4 \pm .8$	$69.08 \pm .53$	$64.16 \pm .37$	$57.08 \pm .43$	$50.39 \pm .38$
Distilled KF	$\mathbf{64.12 \pm .31}$	$\mathbf{74.96 \pm .33}$	$\mathbf{63.4 \pm .47}$	$\mathbf{72.86 \pm .26}$	$\mathbf{68.42 \pm .23}$	$62.04 \pm .47$	$\mathbf{56.28 \pm .24}$
improvement	$9.59 \pm .71$	$7.51 \pm .5$	$13.55 \pm .73$	$7.7 \pm .27$	$9.39 \pm .65$	$5.79 \pm .66$	$6.44 \pm .37$

Mean test accuracy (in percent) and standard error of ResNet18, comparing our method and the baselines on datasets with large label noise and significant overfitting. We include a comparison using the CIFAR-100N dataset, which has innate label noise. The last row shows the improvement of the best performer over the baseline network. Bolded values indicate the best performer in each column.

TABLE IV
COMPARISON WITH INDEPENDENT ENSEMBLES

Method/Dataset	**CIFAR-100**			**TinyImageNet**		
architecture / % noise	0%	20% asym	40% asym	0%	20% sym	40% sym
single network	$78.85 \pm .32$	$67.07 \pm .42$	$49.95 \pm .53$	$65.30 \pm .36$	$56.25 \pm .47$	$49.83 \pm .3$
independent ensemble	$\mathbf{82.13}$	73.89	55.18	$\mathbf{71.88}$	$\mathbf{64.22}$	$\mathbf{57.56}$
distilled independent ensemble	$79.93 \pm .21$	$\mathbf{74.54 \pm .18}$	$\mathbf{56.84 \pm .22}$	$67.58 \pm .22$	$62.54 \pm .25$	$57.24 \pm .33$
KF ensemble (ours)	$79.36 \pm .17$	$74.58 \pm .28$	$63.37 \pm .53$	$68.33 \pm .17$	$\mathbf{62.22 \pm .38}$	$55.66 \pm .37$
distilled KF ensemble (ours)	$\mathbf{80.29 \pm .16}$	$\mathbf{74.96 \pm .33}$	$\mathbf{63.4 \pm .47}$	$\mathbf{69.96 \pm .15}$	$62.04 \pm .47$	$\mathbf{56.28 \pm .24}$
averaged KF ensemble (ours)	$78.48 \pm .26$	$70.72 \pm .44$	$56.79 \pm .92$	$65.17 \pm .23$	$57.08 \pm .43$	$50.32 \pm .45$

A comparison of the Knowledge Fusion ensemble and its two condensed variants with an independent ensemble and its distilled version.

TABLE V
METHODS ALTERING THE TRAINING PROCEDURE

Method/**Dataset**	**CIFAR-100**	**CIFAR-100 asym**		**CIFAR-100 sym**		
% label noise	0%	20%	40%	20%	40%	60%
FGE	$78.9 \pm .4$	$67.1 \pm .2$	$48.1 \pm .3$	$66.5 \pm .1$	$52.1 \pm .1$	$38.3 \pm .7$
SWA	$78.8 \pm .1$	$66.6 \pm .1$	$46.9 \pm .2$	$65.6 \pm .4$	$50.0 \pm .1$	$30.5 \pm .7$
snapshot	$78.4 \pm .1$	$72.1 \pm .4$	$52.8 \pm .6$	$70.8 \pm .5$	$63.8 \pm .2$	$55.6 \pm .2$
KF	$\mathbf{79.36 \pm .17}$	$\mathbf{74.58 \pm .28}$	$\mathbf{63.37 \pm .53}$	$\mathbf{72.68 \pm .15}$	$\mathbf{67.43 \pm .12}$	$\mathbf{57.6 \pm .2}$

Mean test accuracy of Resnet18 on CIFAR100 with and without label noise, comparing KF with baseline methods that alter the training procedure.

pattern is consistent with prior findings on the distillation of dependent versus independent ensembles, see Section II.

Additional Comparisons In Tables V and VI 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 suitable hyper-parameter tuning.

TABLE VI
METHODS ALTERING THE TRAINING PROCEDURE

Method/**Dataset**	**Animal10N**	**TinyImageNet**	
% label noise	8%	20%	40%
FGE	86.5 ± 0.6	$53.8 \pm .1$	$40.4 \pm .3$
SWA	$\mathbf{88.1 \pm .2}$	$52.5 \pm .2$	$39.4 \pm .3$
snapshot	$86.8 \pm .3$	$62.6 \pm .1$	$56.5 \pm .3$
KF	$\mathbf{87.8 \pm .4}$	$\mathbf{62.22 \pm .38}$	$\mathbf{55.6 \pm .39}$

Like Fig. V, for additional datasets.

C. Ablation Study

We conducted an extensive ablation study to investigate the limitations, and some practical aspects, of our method, focusing on the key design choices of the KF ensemble (phase 1). For conciseness, we only provide brief summaries of the specific ablations listed in items (6)-(11) below; the full results and detailed analysis for these ablations are provided in Appendix C.

1) Architecture Choice: To demonstrate the robustness and generality of our method, we evaluate it across a wide range of contemporary neural network architectures. Given the increased representational capacity of these models, we apply them to the ImageNet dataset, which presents a more challenging and large-scale benchmark compared to CIFAR-100 or TinyImageNet. As shown in Table VII, our method consistently outperforms all baselines across all architectures.

TABLE VII
COMPARISON ACROSS ARCHITECTURES ON IMAGENET

Method/Architecture	ResNet50	ConvNeXt-L	ViT-B/16	MaxViT-T
Baseline network	$75.74 \pm .14$	$82.92 \pm .11$	$79.16 \pm .1$	$82.51 \pm .15$
Horizontal	$76.42 \pm .1$	$83.02 \pm .06$	$79.53 \pm .13$	$82.93 \pm .14$
Fixed jumps	$75.72 \pm .18$	$83.86 \pm .06$	$79.11 \pm .13$	$83.78 \pm .15$
KF	$\mathbf{76.52 \pm .16}$	$\mathbf{83.96 \pm .09}$	$\mathbf{80.34 \pm .08}$	$\mathbf{83.81 \pm .14}$
Improvement	$.78 \pm .04$	$1.03 \pm .13$	$1.17 \pm .08$	$1.29 \pm .02$

Mean test accuracy (in percent) and standard error on ImageNet, comparing our method and baselines across several modern architectures. Bolded values indicate the best performer in each column.

TABLE VIII
COMPARISON WITH EMA

Method/**Dataset**	**CIFAR-100**	**TinyImageNet**
EMA (decay = 0.999)	$-0.34 \pm .14$	$0.73 \pm .11$
EMA (decay = 0.9999)	$-0.06 \pm .33$	$2.51 \pm .01$
KF	$\mathbf{1.05} \pm .14$	$\mathbf{3.54} \pm .14$

Mean (over random validation/test split) improvement in test accuracy (in percent) and standard error on image classification datasets, comparing our method and EMA with different decay values. We use the best epoch for EMA, calculated using the validation set.

2) Comparisons to Additional Baselines: In addition to the condensing methods presented in Section V, an alternative method to combining checkpoints is to apply an exponential moving average (EMA) to the model weights during training. This technique is known to have some advantages [8] and is used sometimes to reduce overfitting [1, 27], see [28] for example. In Table VIII, we explore this option for two datasets while using the same ResNet-18 architecture, showing that our method can be of use when EMA isn't effective, and that EMA improves performance much less than our method.

TABLE IX
TRANSFER LEARNING COMPARISON

Method/**Dataset**	**CIFAR-100**	**TinyImageNet**
fully finetuned Resnet18	$80.72 \pm .53$	$75.01 \pm .12$
fully finetuned Resnet18 + KF	$\mathbf{81.64 \pm .27}$	$\mathbf{75.6 \pm .18}$
partially finetuned Resnet18	$61.7 \pm .60$	$54.78 \pm .13$
partially finetuned Resnet18 + KF	$\mathbf{65.24 \pm .67}$	$\mathbf{59.5 \pm .03}$

Mean test accuracy over random validation/test split. Our method uses ResNet18 pre-trained on ImageNet, while finetuning the entire model (top) or only the head (bottom).

3) Transfer Learning: Another popular method to improve performance and reduce overfitting employs transfer learning, in which the model weights are initialized using pre-trained weights over a different task, for example, ImageNet pre-trained weights. This is followed by *fine-tuning* either the entire model or only some of its layers (its head for example). In Table IX, we show that our method is complementary to transfer learning, improving performance in this scenario as well. Notably, when fine-tuning only the final layers, the overhead of KF is **negligible**, as the majority of the model is used only once at inference. The only overhead comes from the memory and inference costs of storing and processing the different checkpoint heads used during fine-tuning, which are small in comparison to the size of the full model.

4) Test Time Augmentation: In TTA, each test sample is classified multiple times using different augmentations, and the final prediction is obtained by averaging the class probabilities from all augmented versions. The results in Table X show that our method is comparable or better than TTA, even in the presence of label noise. Moreover, our approach is complementary to TTA, offering potential benefits when used together.

5) Model Size: A common practice nowadays is to use very large neural networks, with hundreds of millions of parameters, or even more. However, enlarging models does not always improve performance, as a large number of parameters can lead to overfitting. In Fig. 4a we show that indeed the forget fraction increases with the model's size, which also improves the benefit of our approach (see Table XI), making it especially useful while using large models.

6) Using Training Set for Validation (Appendix C.1): The results reported above summarize experiments in which half of the *test data* was used for validation, to evaluate our method's hyper-parameters (the list of alternative epochs and $\{\varepsilon_i\}$). The accuracy reported above was computed over the remaining test data, averaged over three random splits of validation and test data with different random seeds. In our ablation study we repeated these experiments using 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.

7) Number of Checkpoints (Appendix C.2): Knowledge Fusion remains effective when using only a small fraction (5–10%) of the available training checkpoints.

8) Optimal vs Sub-Optimal Training (Appendix C.3): KF is particularly useful under limited hyperparameter tuning; in sub-optimal training regimes, it helps close the gap to optimally trained models.

9) Fairness (Appendix C.4): KF does not negatively affect model fairness across standard fairness benchmarks.

10) Window Size (Appendix C.5): We find that using a window size of $w = 1$ when averaging checkpoints is both necessary and near-optimal for the effectiveness of the method.

11) Knowledge Distillation Hyperparameter Sensitivity (Appendix C.6): To examine the stability of our method with respect to distillation hyperparameters, we conducted an ablation over the loss weight α and the temperature T on the CIFAR–100 dataset with clean labels. The results remain statistically identical across a wide range of values surrounding those reported in the main experiments, indicating that the performance of our method is largely *insensitive* to these hyperparameters.

TABLE X
TEST TIME AUGMENTATION

Method/**Dataset**	**CIFAR-100**	**CIFAR-100 asym**			**CIFAR-100 sym**		**TinyImageNet**
% label noise	0%	10%	20%	40%	20%	40%	0%
TTA	$79.21 \pm .3$	$72.97 \pm .1$	$71.00 \pm .1$	$54.53 \pm .1$	$70.14 \pm .2$	$63.59 \pm .1$	$65.67 \pm .3$
KF	$79.36 \pm .17$	$73.61 \pm .1$	$71.24 \pm .5$	$56.19 \pm .8$	$72.21 \pm .3$	$65.75 \pm .1$	$68.33 \pm .17$
KF + TTA	$\mathbf{79.55 \pm .3}$	$\mathbf{74.83 \pm .1}$	$\mathbf{72.65 \pm .2}$	$\mathbf{57.71 \pm .3}$	$\mathbf{72.33 \pm .2}$	$\mathbf{65.71 \pm .1}$	$\mathbf{69.05 \pm .3}$

Mean test accuracy (in percent) and STE of ResNet-18, comparing KF with Test Time Augmentation (TTA) on datasets with and without label noise.

TABLE XI
COMPARISON OF MODEL SIZE

Method/**model size**	**small**	**base**	**large**
single network	$83.21 \pm .01$	$83.31 \pm .15$	$82.92 \pm .09$
KF	$83.17 \pm .04$	$\mathbf{83.57 \pm .15}$	$\mathbf{83.96 \pm .09}$

Mean (over random validation/test split) test accuracy (in percent) and standard error on image classification datasets, comparing our method and the original predictor (ConvNeXt, trained on ImageNet) with a varying number of parameters.

D. Summary and Discussion

Our method significantly improves performance in modern neural networks. It complements other overfitting reduction methods like EMA and proves effective where these methods fail, as in fine-tuning of pre-trained models. In challenging scenarios, such as small networks handling complex data or datasets with label noise, it further enhances performance, reducing errors by around 15% in cases of 10% asymmetric noise. Our approach outperforms or matches baselines, especially in settings like ViT16 over ImageNet and Resnet18 over TinyImageNet, regardless of training choices. Unlike some horizontal methods and fixed-jump schedules that show limited improvement, our method remains effective without extensive hyper-parameter tuning.

We provide a spectrum of model variants derived from the core KF algorithm, offering different trade-offs between performance, training cost, and inference efficiency. While the distilled model consistently yields the strongest results, simple weight averaging provides clear improvements over the vanilla model, without requiring an additional training phase or the maintenance of multiple networks at inference time. This flexibility allows practitioners to choose the variant that best aligns with their computational and deployment constraints.

VII. FORGOTTEN KNOWLEDGE: THEORETICAL ANGLE

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. While such models do not capture the full complexity of deep nonlinear architectures, they offer a tractable analytical framework that may shed light on the underlying forgetting dynamics. Extending this analysis to nonlinear settings remains a promising direction for future work.

Deep linear 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, with the following objective function:

$$\min_{W_1,\ldots,W_L} \sum_{i=1}^{n} \|W_L \cdot \ldots \cdot W_1 \mathbf{x}_i - y_i\|^2 \tag{3}$$

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 $\boldsymbol{w}^{(n)} = \prod_{l=L}^{1} W_l^{(n)}$ denote 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. [29] 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:

$$w_j^{(n)} \approx \lambda_j^n w_j^{(0)} + [1 - \lambda_j^n] w_j^{opt}, \qquad \lambda_j = 1 - \gamma s_j L \tag{4}$$

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 and $\prod_{l=L}^{1} W_l^{(n)} \xrightarrow[n\to\infty]{} \boldsymbol{w}^{opt}$, the specific solution at convergence $\{W_l^{(\infty)}\}_{l=1}^{L}$ is not unique.

Forget Time in Deep Linear Models. Let Λ denote $\text{diag}(\{\lambda_j\})$ - a diagonal matrix in $\mathbb{R}^{d\times d}$, and I the identity matrix. It follows from (4) that

$$\boldsymbol{w}^{(n)} \approx \boldsymbol{w}^{(0)} \Lambda^n + \boldsymbol{w}^{opt}[I - \Lambda^n] \tag{5}$$

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 $\{\boldsymbol{w}^{(N)} y\mathbf{x} < 0\}$.

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

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

$$\begin{aligned} \boldsymbol{w}^{(\hat{n})} y\mathbf{x} &\leq 0 \\ \boldsymbol{w}^{(n)} y\mathbf{x} &> 0 \qquad \forall n < \hat{n} \end{aligned} \tag{6}$$

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

Proof. Since $\{\boldsymbol{w}^{(0)} y\mathbf{x} > 0\}$ and $\{\boldsymbol{w}^{(N)} y\mathbf{x} < 0\}$, (6) follows by induction. □

Note that Def 1 corresponds with the *Forget time* seen in deep networks (cf. Fig. 4b). The empirical investigation of this correspondence is discussed in [3].

To characterize the time at which a point is forgotten, we inspect the rate at 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. Using (5) and the Taylor expansion of λ_j from (4)

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

It follows that

$$\frac{dF(n)}{dn} = -\gamma y L \sum_{j=1}^{d} (w_j^{(0)} - w_j^{opt}) s_j x_j + O(\gamma^2) \tag{7}$$

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 (7), 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.

VIII. Summary and Conclusions

We revisited the problem of *overfitting* in deep learning by proposing a method to track the forgetting of validation data as a means to detect *local overfitting*. We linked this perspective to the phenomenon of *epoch-wise double descent*, empirically extending its scope and showing that a similar effect arises even in benchmark datasets with clean labels. Motivated by these empirical insights, we introduced a simple yet general method to improve classification performance at inference time. We then demonstrated its effectiveness across a range of datasets and modern network architectures. These results confirm that models do forget useful information in the later stages of training, and provide a proof of concept that recovering this knowledge can lead to improved performance. When combining this method with Knowledge Distillation, we showed enhanced performance accompanied by reduced complexity - particularly in noisy environments - offering a win-win strategy that improves accuracy while reducing both training and inference costs.

Acknowledgments

This work was supported by grants from the Israeli Council of Higher Education, AFOSR award FA8655-24-1-7006, and the Gatsby Charitable Foundation.

References

[1] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," in *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2022.

[2] C. S. R. Annavarapu, "Deep learning-based improved snapshot ensemble technique for covid-19 chest x-ray classification," *Applied Intelligence*, vol. 51, pp. 3104–3120, 2021.

[3] U. Stern, T. Yaacoby, and D. Weinshall, "On local overfitting and forgetting in deep neural networks," in *AAAI Conf. Artif. Intell. (AAAI)*, vol. 39, no. 19, 2025, pp. 20 592–20 600, Appendices – arXiv preprint arXiv:2412.12968.

[4] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*. Elsevier, 1989, vol. 24, pp. 109–165.

[5] Y. Yang, H. Lv, and N. Chen, "A survey on ensemble learning under the era of deep learning," *Artificial Intelligence Review*, vol. 56, no. 6, pp. 5545–5589, 2023.

[6] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

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

[8] B. T. Polyak and A. B. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM journal on control and optimization*, vol. 30, no. 4, pp. 838–855, 1992.

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

[10] S. Noppitak and O. Surinta, "dropcyclic: snapshot ensemble convolutional neural network based on a new

learning rate schedule for land use classification," *IEEE Access*, vol. 10, pp. 60 725–60 737, 2022.

[11] H. Guo, J. Jin, and B. Liu, "Stochastic weight averaging revisited," *Applied Sciences*, vol. 13, no. 5, p. 2935, 2023.

[12] M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine-learning practice and the classical bias–variance trade-off," *Proceedings of the National Academy of Sciences*, vol. 116, no. 32, pp. 15 849–15 854, 2019.

[13] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever, "Deep double descent: Where bigger models and more data hurt," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2021, no. 12, p. 124003, 2021.

[14] C. Stephenson and T. Lee, "When and how epochwise double descent happens," *arXiv preprint arXiv:2108.12006*, 2021.

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

[16] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015. [Online]. Available: https://arxiv.org/abs/1503.02531

[17] H. Jeong and H. W. Chung, "Understanding self-distillation and partial label learning in multi-class classification with label noise," *arXiv preprint arXiv:2402.10482*, 2024.

[18] U. Stern, D. Shwartz, and D. Weinshall, "United we stand: Using epoch-wise agreement of ensembles to combat overfit," in *AAAI Conf. Artif. Intell. (AAAI)*, vol. 38(13), 2024, pp. 15 075–15 082.

[19] Z. Allen-Zhu and Y. Li, "Towards understanding ensemble, knowledge distillation and self-distillation in deep learning," *arXiv preprint arXiv:2012.09816*, 2020.

[20] M. Pham, M. Cho, A. Joshi, and C. Hegde, "Revisiting self-distillation. arxiv," *arXiv preprint arXiv:2206.08491*, 2022.

[21] R. Das and S. Sanghavi, "Understanding self-distillation in the presence of label noise," in *Int. Conf. Machine Learning (ICML)*. PMLR, 2023, pp. 7102–7140.

[22] C. Wang, Q. Yang, R. Huang, S. Song, and G. Huang, "Efficient knowledge distillation from model checkpoints," *Adv. Neural Inform. Process. Syst. (NeurIPS)*, vol. 35, pp. 607–619, 2022.

[23] C. Wang, S. Zhang, S. Song, and G. Huang, "Learn from the past: Experience ensemble knowledge distillation," in *Int. Conf. Pattern Recog. (ICPR)*. IEEE, 2022, pp. 4736–4743.

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

[25] L. Huang, C. Zhang, and H. Zhang, "Self-adaptive training: beyond empirical risk minimization," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020, neurIPS 2020.

[26] T. Garipov, P. Izmailov, D. Podoprikhin, D. P. Vetrov, and A. G. Wilson, "Loss surfaces, mode connectivity, and fast ensembling of dnns," *Adv. Neural Inform. Process. Syst. (NeurIPS)*, vol. 31, 2018.

[27] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[28] Z. Tu, H. Talebi, H. Zhang, F. Yang, P. Milanfar, A. Bovik, and Y. Li, "Maxvit: Multi-axis vision transformer," in *Eur. Conf. Comput. Vis. (ECCV)*. Springer, 2022, pp. 459–479.

[29] G. Hacohen and D. Weinshall, "Principal components bias in over-parameterized linear models, and its manifestation in deep neural networks," *Journal of Machine Learning Research*, vol. 23, no. 155, pp. 1–46, 2022.

Uri Stern received the BSc and MSc degrees in computer science from the Hebrew University of Jerusalem, Israel, where he was a member of the Vision Lab led by Prof. Daphna Weinshall. His research interests include deep learning, representation learning, and generalization in neural networks.

Eli Corn received the BSc and MSc degrees in computer science from the Hebrew University of Jerusalem. His master's research was conducted in the Vision Lab under the supervision of Prof. Daphna Weinshall. He also has several years of industry experience developing deep learning models for real-world applications. His research interests include robustness, learning dynamics, and knowledge distillation in neural networks.

Daphna Weinshall is a Professor of Computer Science at the Hebrew University of Jerusalem and has held visiting positions at MIT, NYU, IBM, and NECI Research Labs. She received her M.S. and Ph.D. in Statistics (population genetics) from Tel Aviv University. Dr. Weinshall has served as an area chair for NeurIPS, CVPR, ICCV, ECCV, and IJCAI, and as an editorial board member for IEEE TPAMI, CVIU, and MVA. She has also served on various international grant review panels, including ERC and NSF panels. Her recent research focuses on deep learning in dynamic settings, with an emphasis on curriculum, continual, and active learning.