The Hebrew University of Jerusalem
The Rachel and Selim Benin School of Computer Science and Engineering

Observing and Correcting Overfit in Deep Neural Networks

Uri Stern

Thesis submitted in partial fulfillment of the requirements
for the Master of Sciences degree
in Computer Science

Under the supervision of **Prof. Daphna Weinshall**

December 2023

Abstract

The absence of overfit in modern deep neural networks is confusing. On one hand, theory predicts that larger models should suffer more from overfit, leading to worse performance at inference time. On the other hand, larger and larger models are being used today, showing better and better performance. How can we explain the gap?

In this work, we analyze the effect of overfit on deep neural networks in two novel ways: we begin by developing new metrics for overfit detection, showing that overfit can happen "locally" in sub-regions of the data population as the training continues even if the general performance of the model is improving, as a result of simultaneous learning of useful and harmful information from the training data. Then, we show that overfit increases diversity between independently trained models, and show how this phenomenon can be used for overfit detection at training time, as well as wrong predictions detection at inference time.

Our first observation raises the need for methods that counteract the damages of overfit, both on the general and local levels, while enabling the model to acquire useful features of the data learned at the late stages of training (even in the overfitting phase). Accordingly, we propose two such methods: KnowledgeFusion (KF) and Max Agreement Prediction (MAP): the first uses an ensemble over checkpoints from the training history of a single model, and the second uses the diversity inside an ensemble - caused by overfitting - to detect and correct wrong predictions. We evaluate our methods on multiple image classification datasets with modern neural networks and show significant improvement compared to simple baselines and competitive methods from the literature.

Acknowledgements

I would like to thank my supervisor Prof. Daphna Weinshall for her great support and guidance over the last two years, and for pushing me to do this interesting and exciting research. Daniel Schwartz, my colleague for a large part of my research, deserves a special thank you for teaching me a lot of what I learned along the way - from doing research to numpy. I would also like to thank my friends at the lab for their helpful conversations and advises over these last two years.

Lastly, I would like to thank my family, who taught me that having fun is the most important part of this (and every other) work; and to my wonderful partner, that listened patiently to every part of this work, and supported me as much as possible with making this research and thesis.

In memory of my grandfather, Dr. Abraham Suhami, a man of science.

Contents

List of Figures

List of Tables

1 Introduction

Deep supervised learning has achieved exceptional results in various image recognition tasks in recent years. This impressive success is largely attributed to two factors - ready access to very large annotated datasets, and powerful architectures involving a huge number of parameters, thus capable of learning very complex functions from large datasets. However, when training very large models, the classical theory of machine learning suggests we will face the risk of *overfit* - when models are fine-tuned to irrelevant details in the training set, which implies poorer generalization as training proceeds.

Theoretical analysis predicts that as a model acquires additional degrees of freedom, its ability to fit certain training data increases. Accordingly, in deep learning, we expect to see *increased generalization error* as the number of parameters and/or training epochs increases. Surprisingly, even vast deep neural networks with many billions of parameters rarely fulfill this expectation. In fact, even in larger models that do exhibit slightly inferior performance [28] with increased size, we do not see overfit as a function of epochs. More typically, even a substantial increase in the number of parameters will still lead to improved performance, or to more bizarre behaviors like the double descent in test error [1, see Section 3.1]. Clearly, there is a gap between our classical understanding of overfit and the empirical results obtained when using modern neural networks.

To bridge this gap, we introduce a novel perspective on overfit. Instead of gauging it solely through a reduction in *test accuracy*, we propose to track what we term *the model's forget fraction*. This metric represents the portion of test data[1] that the model initially classifies correctly but misclassifies as training proceeds. In Section 3.1 we examine various benchmark datasets, where we measure this phenomenon even if no overfit is observed using the traditional definition, namely, when test accuracy increases as learning proceeds. Interestingly, we observe that models can suffer from a significant level of forgetting, which indicates that our score measures something new. Importantly, this occurs even after the deployment of modern methods to reduce overfit, such as data augmentation, while using competitive networks.

Using this new perspective with its corresponding scores and to better understand the phenomenon, we analyze in Section 3.1 the curious phenomenon of "epoch-wise double descent". Here, models trained over data with label noise show *two distinct periods of descent in test error* (i.e., improvement in generalization), the second of which occurs *after* the model has memorized the noisy labels. Our empirical analysis shows that the second period of descent is caused by learning *new patterns* in the data, rather than relearning old ones forgotten at the overfit stage. This phenomenon, we show empirically, is caused by the *simultaneous learning* of general patterns learned from clean data (which enhances performance continuously as learning progresses), and irrelevant specific patterns learned from noisy data (which degrades performance). A similar phenomenon, we hypothesize, happens even when no overfit is seen via the validation accuracy. This hypothesis is strengthened by the observation that for most of

[1] We use the common terminology, though for estimation purposes, the various scores are evaluated on validation data.

the ”forgotten” validation data, the last epoch in which the data was last correctly predicted is roughly the same - the data was ”forgotten” in a dense period of epochs, in the late stages of training (similarly to the double descent case).

Overfit, simply put, means that a model becomes to specialized on the training data. As neural networks have enormous degrees of freedom, different models are able to memorize the training data in a different way, which might result in *different overfitting models* (see illustration in Figure. 3.5) that predict differently during inference time. We thus continue our study of overfit with new lens: ensemble's inter-model agreement.

In the context of neural network ensembles, Recent empirical findings [14, 36] show that when the predictions of networks are tracked through time, almost all the networks will either predict correctly the correct label or almost all will fail. This is true in all the epochs, for both the training and the test sets, irrespective of architecture or learning parameters. These findings imply that deep networks demonstrate high agreement per epoch in their predictions of correct labels. But while all the networks are shown to succeed together, they may still fail in different ways, as visualized in Figure 3.5. Here we pursue the hypothesis that *such failures can be exploited to distinguish between false predictions and correct ones*, most effectively when overfit occurs (as different models can overfit to the training data in different ways).

In Section 3.2 we investigate the hypothesis mentioned above, where we study networks' agreement over false predictions both theoretically and empirically. We start with the theoretical analysis of a regression model (Section 3.2.2), which reinforces the intuition that *overfit should increase variance in prediction between linear models*. Accordingly, we make two conjectures: (i) ensembles become more effective when overfit occurs; (ii) the agreement between deep neural networks is reduced when overfit occurs. These conjectures are verified empirically in Sections 3.2.3-3.2.4, while evaluating deep networks in common use. We see that ensemble classifiers indeed improve performance when overfit occurs. However, overfit is *not* eliminated when ensembles are used. Conjecture (ii) is also shown to be valid, and we report in Section 3.2.5 a *new empirical result*: the variance of correct predictions during inference is much smaller than the variance of false predictions.

Our empirical findings show that useful information can be learned *simultaneously* with harmful one by the neural network, which raises two problems: useful information over sub-regions of the data can be lost (”forgotten”) during training even if the overall validation accuracy improves, and using ”early stopping” to avoid overfitting misses the opportunity to teach the model more useful information in a longer training. In the rest of the paper, we propose two post-training methods to be used in inference time, one for each of those problems.

In Section 4 we propose a method that can effectively reduce the forgetting of test data, both when overfit (in its traditional definition) is present and when it is missing. It does so by combining knowledge gained both in mid-training and after training. More specifically, 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 method is outlined in Alg. 1, see Section 4.

In Section 5 we propose a method that can effectively use information learned by the model *after overfit occurs* without suffering from the overfit, usually showing better performance than early stopping. Our method relies on our new empirical result, which implies that correct predictions at inference time can be distinguished from false ones by looking at the *variance in predictions between multiple networks throughout the training epochs*. This is used to construct a new algorithm for ensemble-based prediction, which is much more resistant to overfit (see Section 5.1.1).

In Section 4.1.3 we perform a series of experiments over image classification datasets on our first method (KF), testing it both when general overfit occurs and when it is missing. Compared to the best epoch of the model, our method often shows significant improvement, which implies successful retrieval of "forgotten" knowledge.

In Section 5.1.2 we describe the empirical validation of our second method (MAP) on settings in which overfit is evident, mostly using datasets with label noise in the training data. Our method shows two positive effects: overfit is eliminated in almost all the experiments (except in extreme cases of label noise), showing that the method is *robust against overfit*; and the performance is better than that of the ensemble, even in its best epoch (obtained by early stopping).

When compared with alternative methods that use the network's training history or ensembles of different models, our methods show comparable or better performance, while being more general and easy to use (both in implementation and hyper-parameter tuning). Specifically, in contrast with other methods, our methods do not depend on additional training choices that require much more time and effort to tune the new hyper-parameters. Another advantage of our methods is that they are complementary to other measures of improving performance (such as special ensemble training methods, overfit reduction methods, etc), where the overall performance is improved compared to using only those measures.

Overall, our contribution is the following: (i) We provide new insights on the overfit of deep neural networks and show how overfit can be detected even when the validation accuracy does not decrease. (ii) We present a method for combining knowledge "forgotten" by the model, even when its validation accuracy keeps on increasing (iii) We present a method that counteracts negative effects of overfitting in the late stages of training, thereby empowering the network to acquire beneficial features learned in those stages and enhance its general performance, even in the presence of overfitting.

2 Previous Work

Studies of overfit and double descent Overfit in deep neural networks, and specifically the double descent and epoch-wise double descent phenomena has garnered increasing attention in recent years [see recent review by 11]. Double descent with respect to model size has been studied empirically in [4, 30], while epoch-wise double descent (which is the phenomenon analyzed here) was studied in [16, 48]. 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).

Not surprisingly, given the gravity of this issue, a variety of measures have been developed through the years to combat overfit. This previous work can be largely divided into two groups, which are **not** mutually exclusive. The first group, which includes the majority of works, focuses on attempting to modify *the learning process* in order to prevent or reduce overfit. These include various forms of data augmentation [43], regularization techniques such as dropout [47], weight decay [22], batch normalization [18] and "early stopping" - terminating the training before overfit occurs.

The second group, to which our methods belong, comes into play after learning is concluded. Such methods involve the design of post-processing algorithms to be invoked during *inference time* (e.g., [25]). Relevant work, when dealing with label noise, includes [3, 58]. An ensemble classifier is used in [41] to combat overfit due to insufficient training data, using network confidence to filter out erroneous predictions.

One benefit of the second approach is that it doesn't require additional time for training, e.g., by way of re-training or data-specific hyper-parameter tuning, which is a major drawback of the first approach. Additionally, methods from the second group can be incorporated with any training scheme, which is useful when methods from the first group fail to completely eliminate overfit.

Study of forgetting in prior work Most relevant studies focus on the forgetting of training data, namely, the fact that some training points are memorized early on, but are then forgotten. This may occur when the network is not able to memorize all the training set. [49], for example, analyzed the forgetting of points in the *train data*, which is then used to score the importance of individual data points for training. In contrast, our work focuses on the forgetting of test points, which cannot be verified during training since the label of these points is not known. Another phenomenon, which may be confused with the one we are discussing, is "catastrophic forgetting" [29, 39]. This occurs in a *continual learning* scenario when the training data changes with time and the network does not have access to training data encountered at the beginning of training. In the scenario considered here, this problem does not exist, since data does not expire.

Ensemble learning Ensemble learning has been studied in machine learning for decades [37], including many recent works that employ deep neural networks ensembles, see [11, 57] for recent surveys. This paradigm is effective when the members of the ensemble generate different predictions, namely, there is diversity (or variability) among the classifiers. To generate such variability, members of the ensemble may be trained with different training sets, as in the popular methods of bagging [5] and boosting [9]. In the context of deep learning, ensemble methods attempt to increase the diversity between the networks in a variety of ways, as can be seen in recent reviews [10, 26]. In [55], for example, a pool of models with different weights initialization is trained with different hyper-parameters for each weight, from which the best models for the ensemble are chosen.

As ensembles are expensive, many works attempt to reduce their cost, specifically their training cost [11, 54]. This line of work, to which one of our methods (KF) belongs, is called "implicit ensemble learning", in which only a single network is learned in a way that "mimics" ensemble learning. A notable work in this field is dropout [47] and its variants, where some parts of the network are dropped at random during training, creating multiple "independent" networks inside the network.

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 [56], or by employing exponential moving average (EMA) on all the weights throughout training [38]. While the latter method has demonstrated some success in reducing overfit, it has been reported to fail in some cases [19].

A number of methods [12, 17, 19] adopted a somewhat different approach that involves changing the training protocol of the network, such that it will converge to several local minima throughout training. These multiple solutions are then combined to form an ensemble classifier. These methods show great promise, impacting a range of fields such as medical applications [1, 31], fault diagnosis [53], attack detection [40] and land use classification [33]. However, these methods should be used with care, as they require the tuning of two inter-connected training schemes (the old and the new), and sometimes prolong the training time substantially at a potentially large financial cost in practical settings. Additionally, the new training scheme is not guaranteed to produce good and diverse local minima, and can even hurt performance as reported by Guo et al. [13].

Noisy labels in the training data One important cause of overfit in deep learning is the existence of false labels in the training dataset (also termed "noisy labels"), which the network nevertheless is able to memorize. Empirically, such noisy labels are known to be memorized late by deep networks [2], resulting in overfit - errors that occur later in training. As deep learning requires large annotated datasets, the problem of false labels often arises in real applications, medical data being an important example [20], where the reliable labeling of data is expensive. Moreover, cheap alternatives such as crowd-sourcing or automatic labeling typically introduce

noisy labels to the dataset [32]. It is thus important to develop additional tools for combating overfit, especially heavy overfit caused by noisy labels, which is one of our goals in this paper. Many works were focused specifically on dealing with label noise in various ways (see [46] for a recent review), mostly by altering the training data/process, under the assumption that it contains label noise. We do not compare ourselves to those methods, as (a) we do not alter the training process, and (b) our methods focus on dealing with overfit in general, and do not assume the existence of label noise in the training data.

3 Overfit Revisited

What is overfit? In previous sections, we discussed why overfit is still a very relevant problem nowadays, and why our current lens of it might be too limited. Here, we propose two different ways of looking at overfit: in Section 3.1 we use alternative metrics for overfit detection to show that overfit can occur in sub-regions of the data population, even if the model generally improves as training proceeds. in Section 3.2 we focus on the effect of overfit on linear and non-linear models, showing that overfit intensifies diversity between independently trained models, which can be useful in both (a) identifying the occurrence of overfit (at training time) and (b) avoiding the mistakes caused by it (at inference time).

3.1 When and how "Local" overfit occurs in DNNs

The textbook definition of overfit entails the co-occurrence of elevated train accuracy and reduced generalization. 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 dataset. Using test accuracy to approximate generalization, this implies that overfit occurs at epoch e when $acc(e, T) \geq acc(E, T)$. However, test accuracy is a global measure that may obscure more subtle expressions of overfit, for example, when overfit only occurs in sub-regions of a continually evolving feature space. In this section, we attempt to expand our view of overfit and develop a more sensitive metric that can capture the second form of overfit even in the absence of the first one.

We begin with the observation that portions of the test data T may be 'forgotten' by the network during training. We argue that this phenomenon indicates a form of local overfit, which can persist and negatively impact specific sub-regions of the dataset even as overall test accuracy continues to improve. Based on this observation, we propose to expand the definition of overfit. This broader definition gains further validation from our examination of the 'epoch-wise double descent' phenomenon, which frequently occurs during training on datasets that contain significant label noise in the training set. In such cases, a notable forgetting of the test data coincides with the memorization of noisy labels, serving as an objective indicator of overfit. The primary implication of this analysis leads to a surprising revelation: even when training modern networks on standard datasets (devoid of label noise), where overfit (as traditionally defined) does not manifest, *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 overfit.

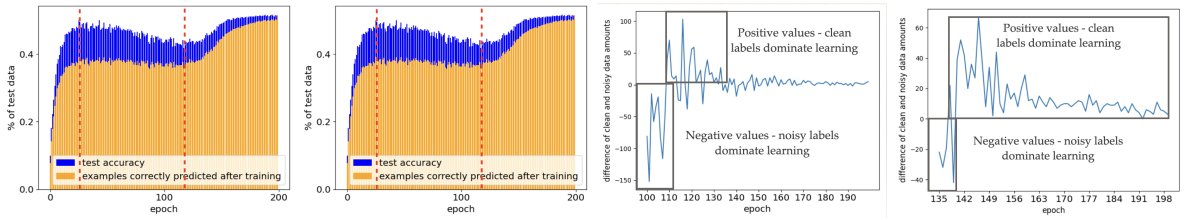
Looking at overfit in a new way Let M_e denote the subset of test data mislabeled by the network at some epoch e. We now define two scores - *learning* L_e and *forgetting* F_e - as follows:

$$F_e = acc(e, M_E) \times \frac{|M_E|}{|T|}, \quad L_e = acc(E, M_e) \times \frac{|M_e|}{|T|} \tag{3.1}$$

F_e, referred to henceforth as the *forget fraction*, represents the fraction of the test data that is correctly classified at epoch e but incorrectly classified at the end of training (epoch E). This subset of the test data is known at epoch e, but later forgotten. On the other hand, L_e denotes the fraction that will be known post-training but is misclassified at epoch e. Clearly $acc(E,T) = acc(e,T) + L_e - F_e$. Therefore, in line with the classical definition of overfit, if $L_e < F_e$, overfit indeed occurs sometime after epoch e.

What happens in the absence of classical overfit? If $L_e \geq F_e \ \forall e$, then by its classical definition *overfit does not occur* since the test accuracy doesn't ever decrease. Nevertheless, there might still be numerous test examples that the final model misclassifies, even though they were classified correctly at some intermediate stage of training. This happens if at some epoch e $L_e > F_e$ is still true, but F_e is nevertheless large. As we show later on, this phenomenon is frequent among neural networks, more so than the traditionally defined overfit, which makes our definition useful in capturing this type of ill behavior.

Reflections on the epoch-wise double descent phenomenon Epoch-wise double descent (see Figure 3.1) is an empirical observation [4], 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 phenomenon is characteristic of learning from data with label noise and is strongly related to overfit since the dip in test accuracy co-occurs with the memorization of noisy labels.



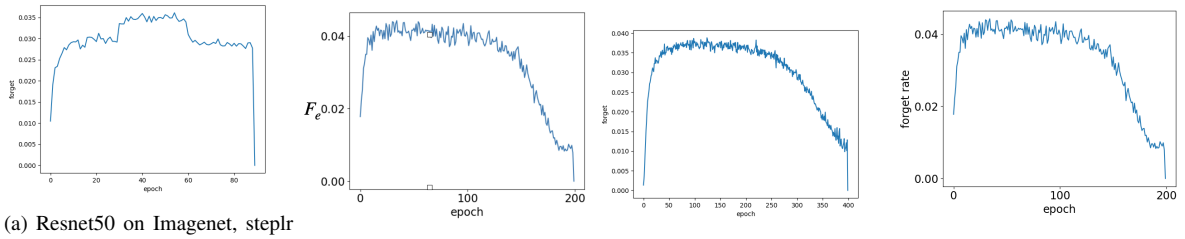
(a) Cifar100, 20% sym noise (b) TinyImagenet, 20% sym noise (c) Cifar100, 20% sym noise (d) TinyImagenet, 20% sym noise

Figure 3.1: (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) 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)-(d): Comparison between the amounts of the clean and noisy data with large loss (above a fixed threshold) at each epoch just before and during the second ascent of test accuracy (the range of epochs after the second dashed red vertical line). A positive value indicates that there are more clean data examples with large loss at this epoch, indicating that the model will now learn more general and correct patterns than wrong patterns caused by the noisy labels.

We examine the behavior of the novel score F_e in this context and make a novel revelation: when we focus on the fraction of data acquired 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 overfit phase (the dip in accuracy). In fact, most of this data has been previously misclassified (refer to Figures 3.1a and 3.1b). To bolster this observation, Figures 3.1c and 3.1d further illustrate that during the later stages of training on data with label noise, when the second increase in test accuracy occurs, the majority of the data being memorized is, in fact, data with clean labels.

What happens when there is no label noise? When training deep networks on visual benchmark datasets without added label noise, double descent rarely occurs, if ever.

What about our new score F_e? To answer this question we trained various neural networks (ConvNets: Resnet, Densenet, ConvNeXt; Visual transformers: ViT, MaxViT) on various datasets (CIFAR100, TinyImagenet, Imagenet) using a variety of optimizers (SGD, AdamW) and learning rate schedulers (cosine annealing, steplr). In Figure 3.2 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. In Figures 3.3a-3.3b we analyze the number of times a mistake of the last epoch was previously classified correctly, and show that for many examples, the correct classification was not a random effect but rather a consistent trend in significant parts of training.



(a) Resnet50 on Imagenet, steplr scheduler

(b) Densenet121 on cifar100

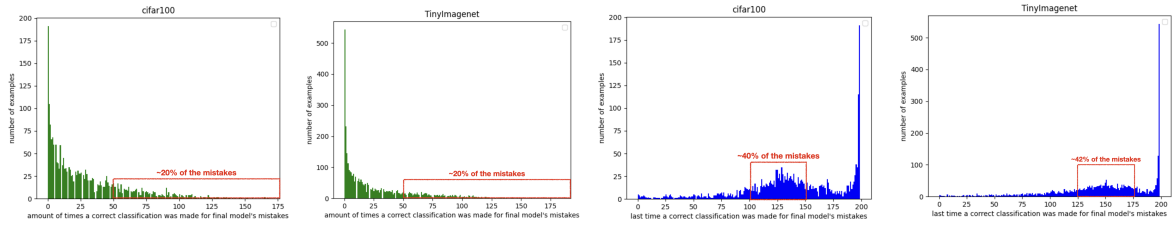
(c) MaxViT tiny on Imagenet

(d) Densenet121 on TinyImagenet

Figure 3.2: The forget fraction F_e, as defined in (3.1), of common neural networks trained on image classification datasets.

To further validate our results, we examined in Figures 3.3c-3.3d a different definition of "forget", in which we look at the last epoch in which an example was classified correctly. Interestingly, we see a dense span of epochs in which many examples are being classified correctly for the last time, which hints that subregions of the data population were "forgotten" in those epochs.

Finally, in Figures 3.4a and 3.4b we connect our results to overfit. We show that when investigating either larger models or relatively small amounts of train data, which are scenarios that



(a) cifar100 (b) TinyImagenet (c) cifar100 (d) TinyImagenet

Figure 3.3: On the left: histogram of the number of epochs a correct prediction was made for test data misclassified post-training; One can see that a large fraction of the mistakes were correctly predicted for over 25% of the training, showing this correct prediction did not happen by random. Notably, both datasets are without label noise, and show no signs of overfit in the validation accuracy. On the right: histogram of the last epoch in which a test data example misclassified post training is correctly classified. One can see that within a span of 50 epochs, a large fraction of those data examples is being "forgotten", as opposed to a random "forgetting" time that might have been expected.

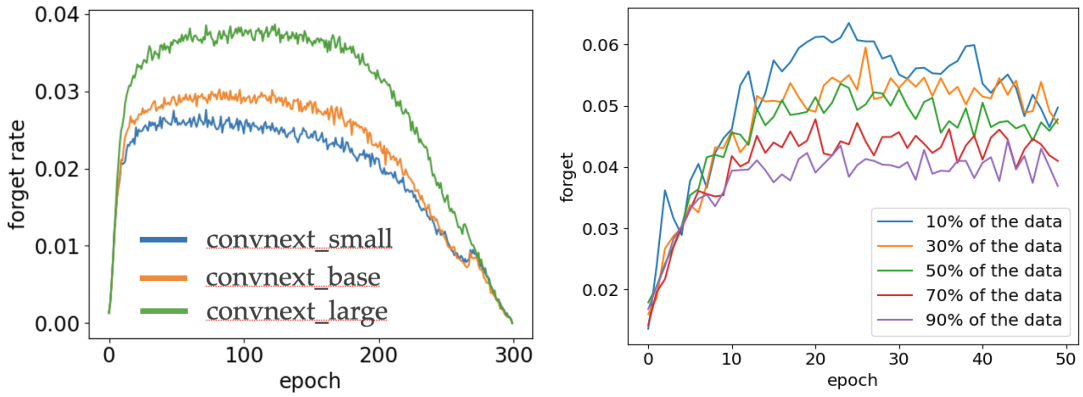
are expected to increase overfit based on theoretical considerations, both are associated with larger *forget fraction* F_e.

Interim summary Our observations reveal 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 subregions of the world. The reason that overfit is not seen using the classical definition, it appears, is that at the same time, the networks still learn general patterns about the data from some correct - but hard to learn - training data, which allows for the test accuracy to keep improving.

3.2 Overfit and ensemble classifiers

In this section we study, theoretically and empirically, ensembles' dynamics when overfit occurs.

We begin with a theoretical analysis of a regression model of linear classifiers (Section 3.2.2), showing that the agreement between such classifiers decreases when overfit occurs. In Section 3.2.3 we empirically study the relevance of this result to deep learning, showing that the same phenomenon occurs also with deep neural networks. In Section 3.2.4 we show that tracking the ensemble agreement through time can often be used for an accurate detection of overfit, using only unlabeled test data. We conclude in Section 3.2.5 with the analysis of the onset time of correct and false predictions. This empirical analysis shows that false predictions caused by overfit are associated with lower agreement scores (see Section 3.2.1) as compared to the correct predictions, implying that false predictions are less common throughout the training than



(a) ImageNet

(b) Cifar100

Figure 3.4: (a) Comparison of the F_e score of ConvNeXt trained on Imagenet, evaluated in three network sizes: small $\rightarrow$ blue, base $\rightarrow$ orange, and large $\rightarrow$ green. Clearly, F_e increases with the size of the network. (b) The F_e score of Resnet18 trained on $10/30/50/70/90\%$ of the train data in cifar100 (purple/red/green/yellow/blue line, respectively) in the first 50 epochs of training (after which the score decreases); F_e is significantly larger for the smallest set of only 10%.

the corresponding correct predictions. Later in Section 5.1.1, these results guide the construction of a new ensemble-based prediction algorithm, which is resistant to overfit.

3.2.1 Methodology

Notations Let N denote the number of networks in an ensemble of deep neural networks trained independently on a training dataset with C classes for E epochs using Stochastic Gradient Descent (SGD), and tested after each epoch of training on the test set $\mathbb{T}$. Let $f_i^e(x)$ denote the prediction of network i for $\mathbf{x} \in \mathbb{T}$ in epoch e. $N^e(x)$ denotes the ensemble prediction in each epoch e, defined as:

$$N^e(\mathbf{x}) = \underset{c \in C}{\operatorname{argmax}} \sum_{i=1}^{N} \mathbb{1}_{[f_i^e(\mathbf{x})=c]} \tag{3.2}$$

Label noise boosts the level of detected overfit As deep models are able to memorize any data distribution, false labels in the training set (also termed "noisy labels") lead networks to learn irrelevant features while memorizing them, which may lead to a large decrease in their test accuracy. As noisy labels are memorized towards the end of the training, this means that training with noisy labels is usually accompanied by overfit. We thus expose our models to data with label noise in order to evaluate robustness to heavy overfit. We evaluate our method on: (i) datasets with synthetic label noise, (ii) real-world datasets with label noise crafted from the

internet by web-labeling (Clothing1M, Webvision), and (iii) a dataset with inherent label noise since it is by design confusing to annotators (Animal10N).

Injecting label noise To generate data with synthetic label noise, we use two standard noise models [34]:

1. **Symmetric noise:** a fraction $p \in \{0.2, 0.4, 0.6\}$ of the labels is selected randomly. Each selected label is switched to any other label with equal probability.
2. **Asymmetric noise:** a fraction p of the labels is selected randomly. Each selected label is switched to another label using a deterministic permutation function.

Inter-Model Agreement In order to measure agreement between different models during inference time, we define below in (3.3) an Agreement score. This score measures the average fraction of networks in the ensemble that predict class c at point $\mathbf{x}$ in a set of training epochs $\mathcal{E}$.

$$Agr(\mathbf{x}, c) = \frac{1}{N|\mathcal{E}|} \sum_{e \in \mathcal{E}} \sum_{i=1}^{N} \mathbb{1}_{[f_i^e(\mathbf{x})=c]} \tag{3.3}$$

Additionally, we also define here the **average per epoch maximal agreement** score - for a single epoch e, test set $\mathbb{X}$ and possible classes $\{1, ..., C\}$ it is defines as:

$$AverageMaxAgreement(\mathbb{X}, e) = \frac{1}{N|\mathbb{X}|} \sum_{x \in \mathbb{X}} max_{c \in \{1,...,C\}} \sum_{i=1}^{N} \mathbb{1}_{[f_i^e(\mathbf{x})=c]} \tag{3.4}$$

3.2.2 Overfit implies increased variance: theoretical result

Since deep learning models are difficult to analyze theoretically, common practice invokes simple models (such as linear regression) that can be theoretically analyzed, whose analysis may shed light on the behavior of actual deep models. Accordingly, it is shown in [44] that using a large ensemble of linear networks, trained on subsets of the training data, can improve performance. We take a different approach, theoretically analyzing an ensemble of linear regression models trained using gradient descent via the perspective of their agreement.

Our analysis culminates in a theorem, which states that the agreement between linear regression models decreases when overfit occurs in all the models, namely when the generalization error in all the models increases.

Here is a brief sketch of the proof:

1. We measure *Disagreement* by the empirical variance over models of the error vector at each test point, averaged over the test examples.

2. We prove the following (intuitive) Lemma 1: *Overfit occurs in a model iff the gradient step of the model, which is computed from the training set, is negatively correlated with a vector unknown to the learner - the gradient step defined by the test set.*

3. We show that under certain asymptotic assumptions, the disagreement is approximately minus the sum of the correlation between each network's gradient step and its "test gradient step". It now readily follows from Lemma 1 that if overfit occurs in all the models then the aforementioned disagreement score increases.

The full proof goes as follows: we start by introducing the model and some notations, followed by our main result (Prop. 3.2.2): the occurrence of overfit at time s in all the models of the ensemble implies that the agreement between the models decreases.

Model. We analyze the agreement between an ensemble of Q models, computed by solving the linear regression problem with Gradient Descent (GD) and random initialization. In this problem, the learner estimates a linear function $f(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}$, where $\mathbf{x} \in \mathbb{R}^d$ denotes an input vector and $y \in \mathbb{R}$ the desired output. Given a training set of M pairs $\{\mathbf{x}_m, y_m\}_{m=1}^M$, let $X \in \mathbb{R}^{d\times M}$ denote the training input - a matrix whose m^{th} column is $\mathbf{x}_m \in \mathbb{R}^d$, and let row vector $\boldsymbol{y} \in \mathbb{R}^M$ denote the output vector whose m^{th} element is y_m. When solving a linear regression problem, we seek a row vector $\hat{\boldsymbol{w}} \in \mathbb{R}^D$ that satisfies

$$\hat{\boldsymbol{w}} = \text{argmin}_{\boldsymbol{w}} L(\boldsymbol{w}), \quad L(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}X - \boldsymbol{y}\|_F^2 \tag{3.5}$$

To solve (3.5) with GD, we perform at each iterative step $s \geq 1$ the following computation:

$$\begin{aligned} \boldsymbol{w}^{s+1} =& \boldsymbol{w}^s - \mu\Delta\boldsymbol{w}^s \\ \Delta\boldsymbol{w}^s =& \frac{\partial L(\mathbb{X})}{\partial \boldsymbol{w}}\bigg|_{\boldsymbol{w}=\boldsymbol{w}^s} = \boldsymbol{w}^s\Sigma_{XX} - \Sigma_{YX} \\ & \Sigma_{XX} = XX^\top,\ \Sigma_{YX} = \boldsymbol{y}X^\top \end{aligned} \tag{3.6}$$

for some random initialization vector $\boldsymbol{w}_0 \in \mathbb{R}^M$ where usually $\mathbb{E}[\boldsymbol{w}_0] = 0$, and learning rate μ. Henceforth we omit the index s when self evident from context.

As a final remark, when we use the notation $\|A\|$ below, it denotes the operator norm of the symmetric matrix A, namely, its largest singular value.

Additional notations

- Index $i \in [Q]$ denotes a network instance, and t denotes the test data. For simplicity and with some risk of notation abuse, let Q and Q' also denote sets of indices, either training or test. Specifically, $Q = [1, \ldots, Q]$ and $Q' = [1, \ldots, Q, t]$.

- We use function notation, where $\{X(i), y(i)\}$ is the training set of network i and $\{X(t), y(t)\}$ is the test set. Thus

$$\Sigma_{XX}(j) = \mathbb{X}(j)\mathbb{X}(j)^\top, \Sigma_{YX}(j) = \boldsymbol{y}(j)\mathbb{X}(j)^\top \quad j \in Q'$$

- Similarly, $\boldsymbol{w}(i) \in \mathbb{R}^d$ is the model learned by network i, and $\Delta\boldsymbol{w}(i)$ is the gradient step of $\boldsymbol{w}(i)$, where

$$\Delta\boldsymbol{w}(i) = \boldsymbol{w}(i)\Sigma_{XX}(i) - \Sigma_{YX}(i) \qquad i \in Q$$

- $\boldsymbol{e}(i,j)$ denotes a function, which maps indices $i \in Q, j \in Q'$ to the cross error of model i on data j - the classification error vector when using model $\boldsymbol{w}(i)$ to estimate $y(j)$. Let $M' = M$ if $j \in Q$ is a training index, and $M' = N$ if $j \in \{t\}$ .Then we can write

$$\begin{aligned} &\boldsymbol{e}(i,j) : Q \times Q' \to \mathbb{R}^{M'} \quad \boldsymbol{e}(i,j) = \boldsymbol{w}(i)\mathbb{X}(j) - \boldsymbol{y}(j) \\ &\implies \quad \Delta\boldsymbol{w}(i) = \boldsymbol{e}(i,i)\mathbb{X}(i)^\top \end{aligned}$$

  Note that in this notation, $e(i,t)$ is the classification error vector when using model i, which is trained on data $\mathbb{X}(i)$, to estimate the desired outcome on the test data - $y(t)$. $\|e(i,t)\|_F$ is the test error, estimate of the generalization error, of classifier i.

- Let $\Delta(i,j)$ denote the cross gradient:

$$\begin{aligned} \Delta(i,j) = \boldsymbol{e}(i,j)\mathbb{X}(j)^\top &= \boldsymbol{w}(i)\Sigma_{XX}(j) - \Sigma_{YX}(j) \\ &\implies \quad \Delta\boldsymbol{w}(i) = \Delta(i,i) \end{aligned} \tag{3.7}$$

After each GD step, the model and the error are updated as follows:

$$\begin{aligned} \tilde{\boldsymbol{w}}(i) &= \boldsymbol{w}(i) - \mu\Delta\boldsymbol{w}(i) \\ \tilde{\boldsymbol{e}}(i,j) &= \tilde{\boldsymbol{w}}(i)\mathbb{X}(j) - \boldsymbol{y}(j) = \boldsymbol{e}(i,j) - \mu\Delta(i,i)\mathbb{X}(j) \end{aligned}$$

We note that at step s and $\forall i, j \in Q$, $\tilde{\boldsymbol{w}}(i)$ is a random vector in $\mathbb{R}^d$, and $\tilde{\boldsymbol{e}}(i,j)$ is a random vector in $\mathbb{R}^M$. If $j \in \{t\}$, then $\tilde{\boldsymbol{e}}(i,j) = \tilde{\boldsymbol{e}}(i,t)$ is a random vector in $\mathbb{R}^N$.

Test error random variable. Let N denote the number of test examples. Note that $\{\boldsymbol{e}(i,t)\}_{i=1}^{Q}$ is a set of Q test errors vectors in $\mathbb{R}^N$, where the n^{th} component of the i^{th} vector $\boldsymbol{e}(i,t)_n$ captures the test error of model i on test example n. In effect, it is a sample of size Q from the random variable $\boldsymbol{e}(*,t)_n$. This random variable captures the error over test point n of a model computed from a random sample of M training examples. The empirical variance of this random variable will be used to estimate the agreement between the models.

Overfit. Overfit occurs at step s if

$$\|\tilde{\boldsymbol{e}}(i,t)\|_F^2 > \|\boldsymbol{e}(i,t)\|_F^2 \tag{3.8}$$

Measuring inter-model agreement. For our analysis, we need a score to measure agreement between the predictions of Q linear functions. This measure is chosen to be the variance of the test error among models. Accordingly, we will measure *disagreement* by the empirical variance of the test error random variable $\tilde{\boldsymbol{e}}(*,t)_n$, average over all test examples $n \in [N]$.

More specifically, consider an ensemble of linear models $\{w(i)\}_{i=1}^{Q}$ trained on set $\mathbb{X}$ to minimize (3.5) with s gradient steps, where i denotes the index of a network instance and Q the number of network instances. Using the test error vectors of these models $e(i,t)$, we compute the empirical variance of each element $\mathrm{var}[\boldsymbol{e}(*,t)_n]$, and sum over the test examples $n \in [N]$:

$$\begin{aligned}\sum_{n=1}^{N} \sigma^2[e(*,t)_n] &= \sum_{n=1}^{N} \frac{1}{2Q^2} \sum_{i=1}^{Q}\sum_{j=1}^{Q} |e(i,t)_n - e(j,t)_n|^2 \\ &= \frac{1}{2Q^2} \sum_{i=1}^{Q}\sum_{j=1}^{Q} \|e(i,t) - e(j,t)\|^2\end{aligned}$$

Definition 1 (Inter-model DisAgreement.)**.** *The disagreement among a set of Q linear models $\{w(i)\}_{i=1}^{Q}$ at step s is defined as follows*

$$DisAg(s) = \frac{1}{2Q^2} \sum_{i=1}^{Q}\sum_{j=1}^{Q} \|e(i,t) - e(j,t)\|^2 \tag{3.9}$$

We first prove Lemma 1, which has the following intuitive interpretation: overfit occurs in model i iff the gradient step of model i (denoted $\Delta \boldsymbol{w}(i)$), which is computed using the training set, is negatively correlated with the 'correct' gradient step - the one we would have obtained had we known the test set (this unattainable vector is denoted $\Delta(i,t)$).

Lemma 1. *Assume that the learning rate μ is small enough so that we can neglect terms that are $O(\mu^2)$. Then in each gradient descent step s, overfit occurs iff the gradient step $\Delta \boldsymbol{w}(i)$ of network i is negatively correlated with the cross gradient $\Delta(i,t)$.*

Proof. Starting from (3.8)

$$\begin{aligned}\text{(overfit)} &\iff \|\tilde{\boldsymbol{e}}(i,t)\|_F^2 > \|\boldsymbol{e}(i,t)\|_F^2 \\ &\iff \|\tilde{\boldsymbol{e}}(i,t)\|_F^2 - \|\boldsymbol{e}(i,t)\|_F^2 > 0 \\ &\iff \|\boldsymbol{e}(i,t) - \mu\Delta(i,i)X(t)\|_F^2 - \|\boldsymbol{e}(i,t)\|_F^2 > 0 \\ &\iff -2\mu\Delta(i,i)X(t)\boldsymbol{e}(i,t)^\top + O(\mu^2) > 0 \\ &\iff \Delta(i,i)\cdot\Delta(i,t) < 0 \\ &\iff \Delta \boldsymbol{w}(i)\cdot\Delta(i,t) < 0\end{aligned}$$

□

Lemma 2 claims that if the magnitude of the gradient step μ is small enough, then the operator norm of matrix $I - \mu\Sigma_{XX}$ is smaller than 1. The implication is that a geometric sum of this matrix converges, a technical result which will be used later.

Lemma 2. *For any invertible covariance matrix Σ_{XX} there exists $\hat{\mu} > 0$, such that $\mu < \hat{\mu} \implies \|I - \mu\Sigma_{XX}\| < 1$.*

Proof. Since Σ_{XX} is positive-definite, we can write $\Sigma_{XX} = USU^\top$ for orthogonal matrix U and the diagonal matrix of singular values $S = diag\{s_i\}$. It follows that $I - \mu\Sigma_{XX} = Udiag\{1 - \mu s_i\}U^\top$, a matrix whose largest singular value is $1 - \mu s_d$. Since by assumption $s_d > 0$, the lemma follows. □

Our last Lemma 3 claims that eventually, after sufficiently many gradient steps, the expected value of the solution is exactly the closed-form solution of the vector that minimizes the loss.

Lemma 3. *Assume that $\|I - \mu\Sigma_{XX}\| < 1$ and Σ_{XX} is invertible. If the number of gradient steps s is large enough so that $\|I - \mu\Sigma_{XX}\|^s$ can be neglected, then*

$$\mathbb{E}[\boldsymbol{w}^s] \approx \Sigma_{YX}\Sigma_{XX}^{-1} \tag{3.10}$$

Proof. Starting from (3.6), we can show that

$$\boldsymbol{w}^s = \boldsymbol{w}^0(I - \mu\Sigma_{XX})^{s-1} + \mu\Sigma_{YX}\sum_{k=1}^{s-1}(I - \mu\Sigma_{XX})^{k-1}$$

Since $\mathbb{E}(\boldsymbol{w}^0) = 0$

$$\begin{aligned}\mathbb{E}(\boldsymbol{w}^s) &= \mathbb{E}(\boldsymbol{w}^0)(I - \mu\Sigma_{XX})^{s-1} + \mu\Sigma_{YX}\sum_{k=1}^{s-1}(I - \mu\Sigma_{XX})^{k-1} \\ &= \mu\Sigma_{YX}\sum_{k=1}^{s-1}(I - \mu\Sigma_{XX})^{k-1}\end{aligned}$$

Given the lemma's assumptions, this expression can be evaluated and simplified:

$$\begin{aligned}\mathbb{E}(\boldsymbol{w}^s) &= \mu\Sigma_{YX}[I - (I - \mu\Sigma_{XX})]^{-1}[I - (I - \mu\Sigma_{XX})^{s-1}] \\ &= \Sigma_{YX}\Sigma_{XX}^{-1} - \Sigma_{YX}\Sigma_{XX}^{-1}(I - \mu\Sigma_{XX})^{s-1} \\ &\approx \Sigma_{YX}\Sigma_{XX}^{-1}\end{aligned}$$

□

From (3.9) it follows that a decrease in inter-model agreement at step s, which is implied by increased test variance among models, is indicated by the following inequality:

$$\begin{aligned}\mathbb{C} =& DisAg(s) - DisAg(s-1) \\ =& \frac{1}{2Q^2}\sum_{i,j=1}^{Q} \|\tilde{\boldsymbol{e}}(i,t) - \tilde{\boldsymbol{e}}(j,t)\|^2 - \\ & \frac{1}{2Q^2}\sum_{i,j=1}^{Q} \|\boldsymbol{e}(i,t) - \boldsymbol{e}(j,t)\|^2 \; > \; 0\end{aligned} \tag{3.11}$$

Theorem. *Assume that all models see the same training set, denoted as $X(i) = X \; \forall i \in [Q]$, and that the training data covariance matrix Σ_{XX} is full rank.*

We make the following asymptotic assumptions, which are loosely phrased but can be rigorously defined with additional notations:

1. *The learning rate μ is small enough so that $\|I - \mu\Sigma_{XX}\| < 1$ (from Lemma 2), and additionally we can neglect terms that are $O(\mu^2)$.*
2. *The number of gradient steps s is large enough so that $\|I - \mu\Sigma_{XX}\|^s$ can be neglected.*
3. *The number of models Q is large enough so that using the law of large numbers, we get $\frac{1}{Q}\sum_{i=1}^{Q} \boldsymbol{w}(i) \approx \mathbb{E}[\boldsymbol{w}]$.*

Finally, we assume that overfit occurs at time s in all the models of the ensemble. In other words, at time s the generalization error does not decrease in all the models.

When these assumptions hold, the agreement between the models decreases.

Proof. (3.11) can be rearranged as follows

$$\begin{aligned}\mathbb{C} =& \frac{1}{2Q^2}\sum_{i,j=1}^{Q} \|[\boldsymbol{e}(i,t) - \mu\Delta(i,i)X(t)] - [\boldsymbol{e}(j,t) - \\ & \mu\Delta(j,j)X(t)]\|^2 - \frac{1}{2Q^2}\sum_{i,j=1}^{Q} \|\boldsymbol{e}(i,t) - \boldsymbol{e}(j,t)\|^2 \\ =& \frac{1}{Q^2}\sum_{i,j=1}^{Q} -\mu[\boldsymbol{e}(i,t) - \boldsymbol{e}(j,t)] \cdot [\Delta(i,i)X(t)] - \\ & \Delta(j,j)X(t)] + O(\mu^2) \\ =& \frac{\mu}{Q^2}\sum_{i,j=1}^{Q} [\Delta(i,i) \cdot \Delta(j,t) + \Delta(j,j) \cdot \Delta(i,t)] - \\ & [\Delta(i,i) \cdot \Delta(i,t) + \Delta(j,j) \cdot \Delta(j,t)] + O(\mu^2)\end{aligned}$$

where the last transition follows from $\boldsymbol{e}(i,t)X(t)^\top = \Delta(i,t)$. Using assumption 2

$$\mathbb{C} = \mu(\mathbb{C}' - \mathbb{C}'') + O(\mu^2) \approx \mu(\mathbb{C}' - \mathbb{C}'') \tag{3.12}$$

where

$$\begin{aligned}\mathbb{C}'' &= \frac{1}{Q^2}\sum_{i,j=1}^{Q}\left[\Delta(i,i)\cdot\Delta(i,t) + \Delta(j,j)\cdot\Delta(j,t)\right] \\ &= \frac{2}{Q}\sum_{i=1}^{Q}\Delta(i,i)\cdot\Delta(i,t)\end{aligned} \tag{3.13}$$

and

$$\begin{aligned}\mathbb{C}' =& \frac{1}{Q^2}\sum_{i,j=1}^{Q}\left[\Delta(i,i)\cdot\Delta(j,t) + \Delta(j,j)\cdot\Delta(i,t)\right] \\ =& \frac{1}{Q}\sum_{i=1}^{Q}\Delta(i,i)\cdot\frac{1}{Q}\sum_{j=1}^{Q}\Delta(j,t) + \\ & \frac{1}{Q}\sum_{j=1}^{Q}\Delta(j,j)\cdot\frac{1}{Q}\sum_{i=1}^{Q}\Delta(i,t) \\ =& \frac{1}{Q}\sum_{i=1}^{Q}\Delta(i,i)\cdot\frac{2}{Q}\sum_{j=1}^{Q}\Delta(j,t)\end{aligned} \tag{3.14}$$

Next, we prove that $\mathbb{C}'$ is approximately 0. We first deduce from assumptions 1 and 4 that

$$\begin{aligned}\frac{1}{Q}\sum_{i=1}^{Q}\Delta(i,i) &= \frac{1}{Q}\sum_{i=1}^{Q}\boldsymbol{w}(i)\Sigma_{XX}(i) - \Sigma_{YX}(i) \\ &= \left(\frac{1}{Q}\sum_{i=1}^{Q}\boldsymbol{w}(i)\right)\Sigma_{XX} - \Sigma_{YX} \approx \mathbb{E}[\boldsymbol{w}]\Sigma_{XX} - \Sigma_{YX}\end{aligned}$$

From assumption 3 and Lemma 3, we have that $\mathbb{E}[\boldsymbol{w}] \approx \Sigma_{YX}\Sigma_{XX}^{-1}$. Thus

$$\begin{aligned}\frac{1}{Q}\sum_{i=1}^{Q}\Delta(i,i) &\approx \mathbb{E}[\boldsymbol{w}]\Sigma_{XX} - \Sigma_{YX} \\ &\approx \Sigma_{YX}\Sigma_{XX}^{-1}\Sigma_{XX} - \Sigma_{YX} = 0\end{aligned}$$

From this derivation and (3.14) we may conclude that $\mathbb{C}' \approx 0$. Thus

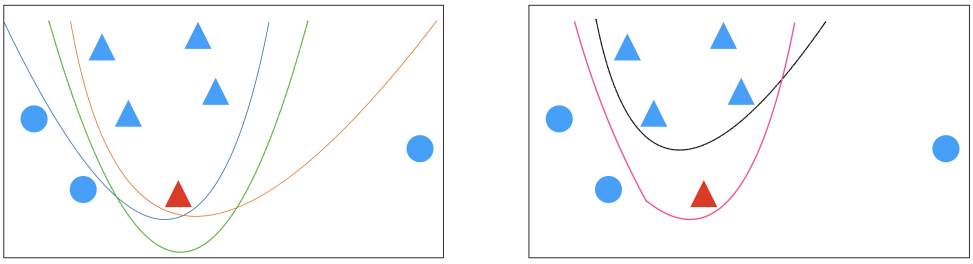
$$\mathbb{C} \approx -\mu\mathbb{C}'' = -\mu\frac{2}{Q}\sum_{i=1}^{Q}\Delta(i,i)\cdot\Delta(i,t) \tag{3.15}$$

If overfit occurs at time s in all the models of the ensemble, then $\mathbb{C} > 0$ from Lemma 1 and (3.15). From (3.11) we may conclude that the inter-model agreement decreases, which concludes the proof.

□

3.2.3 Overfit implies increased variance: empirical result

We discuss in the introduction known findings concerning ensembles of deep models, which imply that all networks are likely to succeed or fail together in their prediction. However, it is still possible that they fail in different ways, where each network predicts a different false label, particularly when overfitting (as each model might memorize the training data differently, see illustration in Figure 3.5).



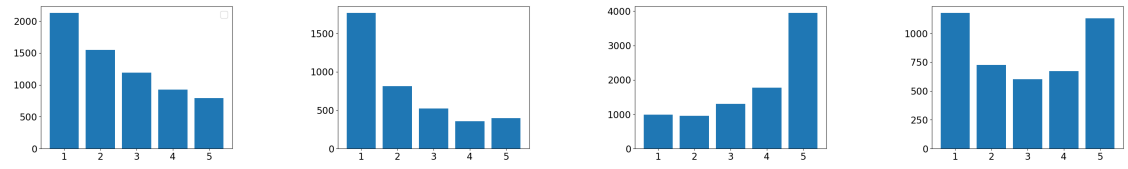
(a) 3 decision boundaries (b) Ensemble classifier

Figure 3.5: (a) A $2D$ classification example, with a single false label (in red) and an ensemble of 3 models, where each model overfitted the false label in a different way. (b) The black line marks the correct decision boundary of the data's empirical distribution. The decision boundary of an ensemble classifier, based on the majority vote of the 3 instances in (a), is shown in pink. It is clearly closer to the correct decision boundary than any of the 3 members of the ensemble.

We therefore start our empirical investigation by analyzing the variance of false predictions (or errors) in such ensembles. We use datasets injected with label noise in this study, in order to amplify the prevalence of overfit. The diversity in the ensembles has 2 sources: (i) different random initialization, and (ii) different random mini-batches within each epoch, where all networks train on the full dataset.

More specifically, we begin by training an ensemble of N DNNs on noisy labeled datasets and then isolate the test points with erroneous prediction in the last epoch. Using this set of points, we compute the *Error Consensus Score (ECS)*, which measures the number of networks that agree on each erroneous prediction (from 1 to N). In Figure 3.6 we plot some empirical histograms of *ECS*.

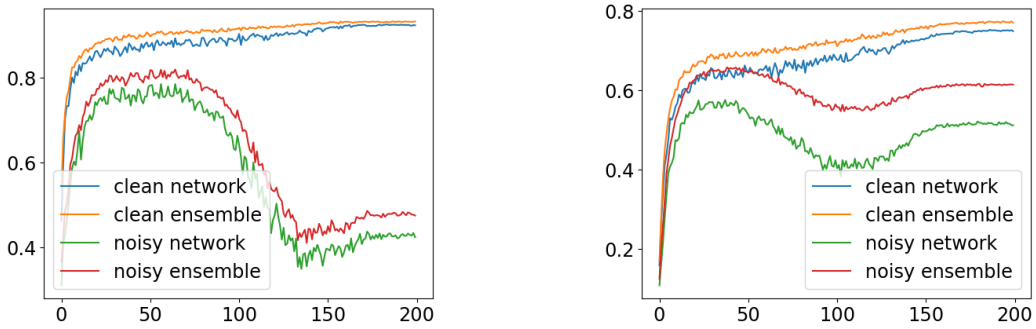
The results in Figure 3.6 indicate that when erroneous predictions are concerned, their distribution seems to have a large variance. In fact, in many of our study cases, the minority value



(a) cifar10 40% SN (b) cifar10 20% AsN (c) cifar100 60% SN (d) cifar100 10% AsN

Figure 3.6: Histograms of *ECS* - the number of networks that agree on each erroneous prediction, for an ensemble of 5 networks. 'SN' denotes symmetric noise, and 'AsN' asymmetric noise.

($ECS < \frac{N}{2}$) dominates the distribution. We also see that in more difficult settings (e.g., cifar100), more mistakes are made by the majority, making the ensemble unable to correct them. We therefore conjecture that ensemble classifiers will be effective in correcting erroneous predictions in cases of overfit. In addition, we conjecture that with minor overfit, ensembles will only make a little difference.



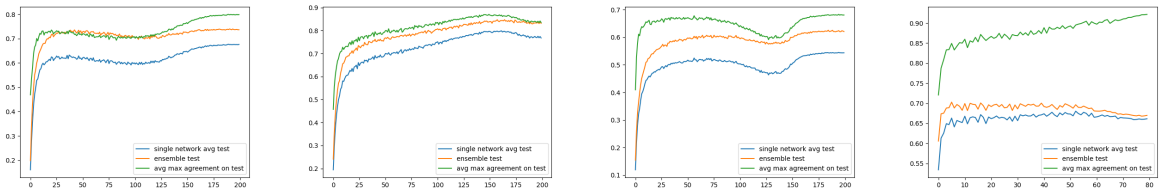
(a) cifar10, 60% symmetric noise (b) cifar100, 40% sym noise

Figure 3.7: Test accuracy over the epochs of a single network (green) and ensemble (red) trained on a dataset with label noise. For comparison, we also show the test accuracy of a single network (blue) and ensemble (orange) trained on the clean data only..

These conjectures are supported by our empirical results, in experiments involving various datasets and noise settings, as shown in Figure 3.7 and Table 5.1. In particular, Figure 3.7 clearly shows that the benefit of the ensemble classifier is much larger when there is significant overfit, which is seen when the training data has significant label noise. However, Figure 3.7 also shows that **while ensembles improve accuracy when there is overfit, they do not eliminate the phenomenon** - there is still performance deterioration as training proceeds. The "rebound" in test accuracy of the single network is due to the "epoch-wise double descent" phenomenon, see details in [30, 48].

3.2.4 Tracking ensemble's variance over time to detect overfit

As our previous analysis shows that overfit increases diversity between independent models, it is interesting to observe whether one can identify overfitting using solely the predictions of different models of an ensemble over some test data, without using its true labels. In Figure 3.8 we measure the *per epoch maximal agreement* (3.4) of the ensemble at each epoch of training. Interestingly, we see that this score and the average test accuracy of a model show a strong connection: when the first decreases, it is a clear sign of overfitting, which allows for early stopping without validation data (although there are instances where overfitting occurs without such a decrease, such as Figure 3.8d).



(a) cifar100, 20% asym noise (b) Imagenet100, 20% sym noise (c) TinyImg, 20% symmetric noise (d) Clothing1M

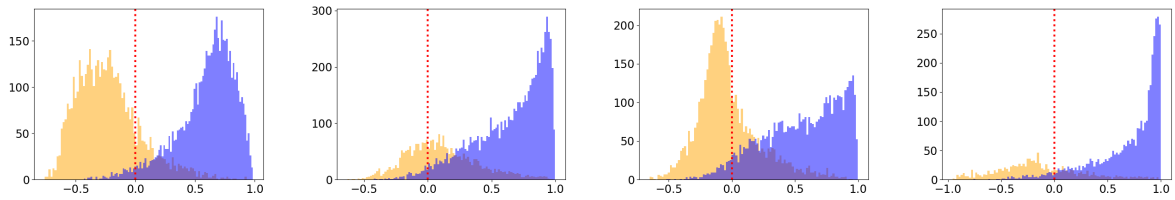
Figure 3.8: tracking the average max agreement score (3.4) over time (green), comparing with the average test accuracy (blue) of a single model and the test accuracy of the ensembles (orange).

3.2.5 Remember your history and avoid overfit

As we have seen earlier, prolonging the training can be useful to learn more informative features of the data, but harmful to the model predictions on some sub-regions of the population, and sometimes even harmful for its predictions on the general level. As ensembles' agreement shows a special connection with overfitting, it is thus curious to see whether it is possible to use ensemble's inter-model agreement to avoid the damage of overfitting at inference time, while still allowing a longer training (not using early stopping) to teach the model some useful late-learned features from the training data.

In this section, we build upon the regular ensemble discussed in Section 3.2.3 and inspect another potential source of variability to that of the different models - a prediction's persistence. Following our theoretical and empirical analysis and the agreement score definition in (3.3), we hypothesize that since erroneous predictions on the test examples have large variance, correct predictions will have larger persistence (agreement) than erroneous ones, amplifying the difference between the two.

In order to test this conjecture, we use the agreement score (3.3) defined in Section 3.2.1. More specifically, we compare the agreement score of the ensemble's final prediction in epoch E $N^E(x)$ with the agreement score of the most agreed upon label, which is different from the



(a) cifar10, 60% symmetric noise (b) cifar100, 20% asym noise (c) TinyImg, 40% symmetric noise (d) Imagenet100, 40% sym noise

Figure 3.9: Histograms of the margin score (3.16), where blue indicates correct predictions and orange erroneous predictions.

final prediction:

$$\begin{aligned} AgrMargin(\mathbf{x}, y) &= Agr(\mathbf{x}, y) - \max_{c \neq y} Agr(\mathbf{x}, c) \\ y &= N^E(\mathbf{x}) \end{aligned} \tag{3.16}$$

Figure 3.9 shows histograms of score (3.16), separated to correct and erroneous predictions. The phenomenon is rather general, shown in a variety of conditions in which overfit is enhanced by injecting label noise. Clearly, most of the predictions with negative margin scores are false, while the vast majority of the correct predictions have a positive score. This result will be used in Section 5 to build a new classifier method from the ensemble, that allows prolonged (and better) training with less danger of overfitting.

4 Algorithm I: KF boosts performance using a single training process

Following our empirical observations on "local" overfit, we construct here a method that aims to reduce its effects during inference time, using an ensemble of checkpoints from a single training process. We describe the method in Section 4.1 and evaluate it on various databases, models, and training schemes. In Section 4.2 we explore various aspects related to our method, including its effectiveness in practical settings, additional overfit-related settings known in the literature, and the general fairness of the final predictor. We conclude in Section 4.3 with a discussion of our method's advantages and limitations.

4.1 Method: recovering forgotten knowledge

In Section 3.1 we showed that neural networks often show 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 much weight to assign the post-training model, whose overall performance is usually better.

We present our answers to those questions and our final algorithm pseudocode in Section 4.1.1. In Section 4.1.2 we present setups for multiple empirical evaluations done in Section 4.1.3 which show the method's usefulness and generality.

4.1.1 Algorithm

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 correct mistakes of the model on "forgotten" test data.

Combining the predictors How to combine the two models, n_A where the forget fraction is maximal, and n_E where train accuracy is maximal? A common practice is to average models' output (probability for each class). However, since the performance of n_E is typically better than that of n_A, we use a weighted average instead, giving n_E a larger weight. This guarantees that our method will not harm the general performance, as it can always give zero weight to the early checkpoint n_A.

Improving robustness To improve our method's robustness to the choice of epoch A, we use a span 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 width $w = 1$.

Working in an iterative manner As we now have a new predictor, we can now find another alternative predictor from the training history that maximizes accuracy on the data misclassified by the new predictor, and combine their knowledge as described. This can be done iteratively until no further improvement is achieved.

Choosing hyper-parameters In order to compute F_e (for the early epoch choice) and to find the best weights and epoch span, 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 costs over the training time*. We follow the common practice, and show in Section 4.2.1 that after finding the best hyper-parameters it is possible to retrain the model on the complete training set and validation set and use our method *without hurting performance* (using the same hyper-parameters calculated earlier), and while maintaining our superiority over alternative methods also trained on the full data.

We call our method **K**nowledge**F**usion (KF) and evaluate it in Section 4.1.3.

In Alg 1 and Alg 2 we show how to implement our method and how to calculate its hyper-parameters, respectively. In the pseudo-codes we call functions that (i) calculate the probability for each class for a given example and a list of predictors (get_class_probabilities) (ii) calculate the forget value per epoch on some validation data, given the predictions at each epoch (calc_forget_per_epoch) (iii) calculate the validation accuracy, to determine the best weights for each epoch (validation_acc).

Algorithm 1: Knowledge Fusion (KF)

Input: A set of checkpoints during training of the neural network $\{n_0,...,n_E\}$, w, test-point x
Output: prediction for x
$\{A_1,...,A_k\}$, $\{\epsilon_1, ..., \epsilon_k\} \leftarrow$ **calc_early_epochs_and_epsilons**$(\{n_0,...,n_E\})$ #use Alg.2
$class\_prob\_per\_checkpoint \leftarrow$ **get_class_probabilities**$(\{n_0,...,n_E\}, x)$
$prob \leftarrow class\_prob\_per\_checkpoint[E]$
for $i \leftarrow 1$ **to** k **do**
  $prob_A \leftarrow$ **mean**$(class\_prob\_per\_checkpoint[A_i - w : A_i + w])$
  $prob \leftarrow \epsilon_i * prob_A + (1 - \epsilon_i) * prob$
end for
$prediction \leftarrow$ **argmax**$(prob)$
Return $prediction$

Algorithm 2: 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
$class\_prob\_per\_checkpoint \leftarrow$ **get_class_probabilities**($\{n_0,...,n_E\}$, V)
$prob \leftarrow class\_prob\_per\_checkpoint[E]$
explore = $\{n_0,...,n_E\}$
Alternative_epochs = $\{\}$
epsilons = $\{\}$
while explore is not empty **do**
  F = **calc_forget_per_epoch**($prob$, $class\_prob\_per\_checkpoint$)
  alt_epoch = **argmax**($F[explore]$)
  Alternative_epochs.**append**(alt_epoch)
  explore.**remove**($alt\_epoch - 1$, alt_epoch, $alt\_epoch + 1$)
  for $epsilon \in \{0, 0.01, ..., 1\}$ **do**
    $prob_A \leftarrow$ **mean**$(class\_prob\_per\_checkpoint[A_i - w : A_i + w])$
    $combined\_prob \leftarrow \epsilon * prob_A + (1 - \epsilon) * prob$
    if validation_acc($combined\_prob$) $>$ **validation_acc**($prob$) **then**
      $best\_prob = combined\_prob$
      $best\_epsilon = epsilon$
    end if
  end for
  $prob = best\_prob$
  epsilons.append(**argmax**($best\_epsilon$))
end while
Return Alternative_epochs, epsilons;

4.1.2 Empirical setup

In our experiments, we use three image classification datasets - Imagenet [6], cifar100 [21] and TinyImagenet [24]. In the Imagenet experiments, we train all networks (Resnet [15], ConvNeXt [28], ViT [8] and MaxViT [51]) using the torchvision code[1] for training [50] with the recommended hyper-parameters, except ConvNeXt, which was trained using the official code[2] with the recommended hyper-parameters, without using exponential moving average (EMA) (see Section 4.2.5 for comparison to using EMA). With cifar100 and TinyImagenet, we train all networks for 200 epochs. For the clean versions of cifar100 and TinyImagenet, we use a batch size of 32, a learning rate of 0.01, SGD optimizer with momentum of 0.9 and weight decay of

[1] https://github.com/pytorch/vision/tree/main/references/classification
[2] https://github.com/facebookresearch/ConvNeXt

5e-4, cosine annealing scheduler, and standard augmentations (horizontal flip, random crops).

We use similar settings for our transfer learning experiments, in which the images are resized to 224×224, the learning rate is set to 0.001 and the network is initialized using Imagenet weights. In training, either the whole network is finetuned or only a new head (instead of the original fully connected layer), which consists of two dense layers, the first with an output size of 100 times the embedding size.

For noisy labels experiments, we train using cosine annealing with warm restarts (restarting the learning rate every 40 epochs), using a larger learning rate of 0.1 and updating it after every batch. We also use a larger batch size of 64 in cifar100 and 128 in TinyImagenet. We use the same method to inject noisy labels as described earlier. In the suboptimal training described in Section 4.2.3, each image was cut before training into its central 224 over 224 pixels (images smaller than this size were first resized such that the smallest dimension was of size 224, then cut into 224 over 224).

To obtain a fair comparison, we train the competitive methods in Table 4.3 from scratch using our network architecture and data. For [17] we train as instructed by the paper, while for [12, 19] we use our training scheme (as these methods are meant to be added to an existing training scheme) and perform hyper-parameter search to optimize the methods' performance in the new setting. Experiments were conducted on a cluster of GPU type A5000.

4.1.3 Empirical evaluation

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 4.1: 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 our method over a 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 iterative version of the method. 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%	10%	20%	40%	20%	40%	20%	40%
single network	85.9±.3	72.1±.1	67.1±.5	49.4±.3	65.4±.3	56.9±.1	56.2±.2	49.8±.3
fixed jumps (∞)	87.1±.4	76.2±.1	73.9±.1	59.9±.6	72.8±.1	66.5±.1	60.0±.8	54.16±.3
horizontal (∞)	86.3±.3	75.4±.3	73.4±.1	58.5±.1	71.1±.38	65.2±.1	59.3±.3	51.7±.2
KF (ours) (∞)	**87.8±.4**	**76.6±.3**	**74.2±.1**	**62.1±.5**	**72.8±.1**	**67.0±.1**	**62.8±.2**	**57.0±.5**
improvement	**1.9±.4**	**4.4±.2**	**7.1±.6**	**12.6±.2**	**7.4±.4**	**10.1±.1**	**6.6±.1**	**7.2±.1**

Table 4.2: 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 overfit. We include a comparison to the Animal10N dataset, which has innate label noise. Note that as is customary, only the train data has label noise while the test data remains clean for a fair evaluation.

Method/**Dataset**	**CIFAR-100**	**TinyImagenet**	**Animal10N**	**CIFAR-100 asym**		**CIFAR-100 sym**	
% label noise	0%	0%	8%	20%	40%	20%	40%
FGE (∞)	78.9±.4	67.7±.1	86.5±0.6	67.1±.2	48.1±.3	66.5±.1	52.1±.1
SWA (∞)	78.8±.1	**69.3±.6**	**88.1±.2**	66.6±.1	46.9±.2	65.6±.4	50.0±.1
snapshot (∞)	78.4±.1	**69.3±.4**	86.8±.3	72.1±.4	52.8±.6	70.8±.5	63.8±.2
KF (ours) (∞)	**79.3±.2**	**69.4±.6**	**87.8±.4**	**74.2±.1**	**62.1±.5**	**72.8±.1**	**67.0±.1**

Table 4.3: Mean test accuracy (in percent) and standard error of Resnet18, comparing our method and baseline methods that alter the training.

Method/**Dataset**	**CIFAR100 sym**	**TinyImagenet**	
% label noise	60%	20%	40%
FGE	38.3±.7	53.8±.1	40.4±.3
SWA	30.5±.7	52.5±.2	39.4±.3
snapshot	55.6±.2	62.6±.1	56.5±.3
KF (ours)	**57.6±.2**	**62.8±.2**	**57.0±.5**

Table 4.4: Mean (over random validation/test split) test accuracy (in percent) and standard error on image classification datasets with injected label noise, comparing our method and baselines.

We now demonstrate the superior performance of our method as compared to the original predictor, i.e. the network after training, as well as other baselines. We evaluate our method using various image classification datasets, neural network architectures, and training schemes. The main results of our empirical evaluation are presented in Tables 4.1-4.4, followed by an extensive ablation study (and additional comparisons) in Section 4.2.

Review of empirical results In Table 4.1 we report the results of our method using multiple architectures trained on cifar100, TinyImagenet, and Imagenet, with different learning rate schedulers and optimizers. For comparison, we report the results of both the original predictor and some simple baselines. We continue with additional experiments on settings connected to

overfit in Table 4.2, where we test our methods on these datasets with injected symmetric and asymmetric label noise (see Section 4.1.2), as well as on 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 Tables 4.3-4.4 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 search (where our result is the best over the different training methods) . Finally, we compare in Fig. 4.2 our method with an ensemble of independent networks, to evaluate how much of the ensemble's performance boost can be gained using our method (without the extra cost of ensemble training), see details in Section 4.1.2.

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 $\{\epsilon_i\}$), 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 Section 4.2.1 we show that when data is limited, we can train a network on a subset of the training data while using the left-out data for hyper-parameter tuning. As customary, these same parameters are later used with models trained on the full data, demonstratively without deteriorating the results.

Baselines Our method incurs the training cost of a single model, and thus, following the methodology of [17], we compare ourselves to methods that require the same amount of training time. Our baselines are from two groups of methods. The first group includes methods that do not alter the training process:

- **Single network**: the original network, after training.
- **Horizontal ensemble** [56]: 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 [17], 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 the following methods:

- **Snapshot ensemble** [17]: in this method, the network is trained in several "cycles", each ending with a large increase in the learning rate that pushes the network away from the local minimum. The network is meant to converge to several different local minima during training, which are used as an ensemble.

- **Stochastic Weight Averaging** (SWA) [19]: in this method, the network is regularly trained for a fixed training budget of epochs, and is then trained using a circular/constant learning rate to converge to several local minima, whose weights are averaged to get the final predictor. To achieve a fair comparison in training budget, we train the network using our training method for 75% of the epochs, followed by their unique training for the remaining 25% epochs.
- **Fast Geometric Ensembling** (FGE) [12]: similar to SWA, except that the final predictor is constructed by averaging the probability outputs of each model, instead of their weights. In this comparison, we match budgets as explained above.

Comparisons to additional baselines that are relevant to resisting overfit, including early stopping and test time augmentation, are discussed in Section 4.2.5.

4.2 Ablation study

In this section, we investigate some limitations and practical aspects of our method. In Section 4.2.1 we show that a separate validation set is not really necessary for the method to work well. In Section 4.2.2 we investigate how many checkpoints are needed for the method to be effective, showing that only $5-10\%$ of the past checkpoints are sufficient. In Section 4.2.3 we investigate the added value of our method when using only partial hyper-parameter search, as is common in real applications, which leads to sub-optimal training. Interestingly, our method is shown to be even more beneficial in the sub-optimal scenario and reduces the gap between the optimal and sub-optimal networks. In Section 4.2.4 we show our method is effective in a transfer learning scenario when the network's initial weights are pre-trained.

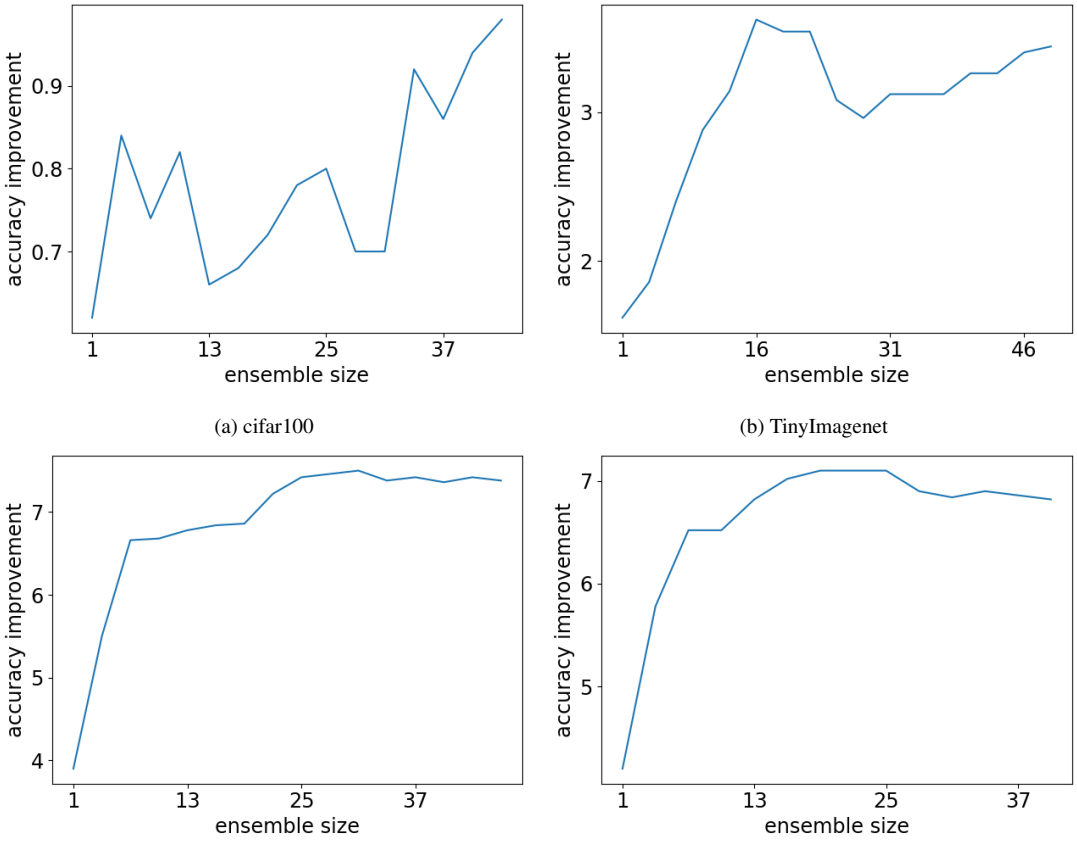
Additional evaluations show that: (i) our method is superior compared to exponential-moving-average (EMA), early stopping and test time augmentation (Section 4.2.5); (ii) our method's improvement can increase as the number of parameters increases (Section 4.2.6); (iii) a large portion the improvement of a regular ensemble of independent networks can often be obtained using our method at a much lower cost (Section 4.2.7); and (iv) our method does not have negative effects on the model's fairness (Section 4.2.8).

4.2.1 Removing the requirement for validation set

In this experiment, we follow a common practice with respect to the validation data: we train our model on cifar100 and TinyImagenet using only 90% of the train data, use the remaining 10% for validation, and finally retrain the model on the full train data while keeping the same hyper-parameters for inference. The results are almost identical to those reported in Table 4.1. This validates the robustness of our method to the (lack of) validation set.

4.2.2 number of checkpoints used

Here we evaluate the cost entailed by the use of an ensemble at inference time. In Fig. 5.2 we report the improvement in test accuracy as compared to a single network when varying the ensemble size. The results indicate that almost all of the improvement can be obtained using only $5-10\%$ of the checkpoints, making our method practical in real life.



(a) cifar100

(b) TinyImagenet

(c) cifar100, 20% asym noise

(d) TinyImagenet, 20% sym noise

Figure 4.1: Improvement achieved by our method when using a different number of checkpoints (shown on the x-axis).

4.2.3 Optimal vs sub-optimal training

In real life, a full search for the optimal training scheme and hyper-parameters is not always possible, leading to sub-optimal performance. Interestingly, our method can be used to reduce the gap between optimal and sub-optimal training, as seen in Table 4.5, where the gap be-

tween optimally and suboptimally trained MaxViT over Imagenet reduces by almost half when applying our method (on both models).

training/**method**	**original network**	**KF**
regular training	82.5±.1	83.8±.1
sub-optimal training	77.3±.1	81.0±.1
improvement	**5.2**	**2.8**

Table 4.5: Mean test accuracy (in percent) and ste, over random validation/test split. MaxViT is trained to classify Imagenet, comparing optimal and sub-optimal training with and without KF.

4.2.4 Transfer learning

Another popular method to improve performance and reduce overfit 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 4.6 we show that our method is complementary to the use of transfer learning in both cases, as it can still improve performance in this scenario. Note that when finetuning only the last layers, our method is **almost free of overhead costs**, as one needs to save and use at inference most of the model only once. Thus, the only overhead involves the memory and inference costs of the different head checkpoints in finetuning, whose size is insignificant compared to the rest of the model.

Method/**Dataset**	**CIFAR-100**	**TinyImagenet**
fully finetuned Resnet18	80.72±.53	75.01±.12
fully finetuned Resnet18 + KF	**81.64±.27**	**75.6±.18**
partially finetuned Resnet18	61.7±.60	54.78±.13
partially finetuned Resnet18 + KF	**65.24±.67**	**59.5±.03**

Table 4.6: Mean test accuracy over random validation/test split. Our method was applied to Resnet18 pre-trained on Imagenet while finetuning the entire model (top) or only the head (bottom).

4.2.5 Comparisons to additional baselines

An alternative method to combine different checkpoints is to perform exponential moving average (EMA) during training, which is known to have some advantages [38] and is used sometimes to reduce overfit [8, 28], although it is not always useful (see [51] for example). In table 4.7 we explore this option for two datasets and a regular Resnet18, showing that our method can be of use when EMA doesn't work or improves the performance much less than our method.

Method/**Dataset**	**CIFAR-100**	**TinyImagenet**
EMA (decay = 0.999)	−0.34 ±.14	0.73 ±.11
EMA (decay = 0.9999)	−0.06 ±.33	2.51 ±.01
KF	**1.05** ±.14	**3.54** ±.14

Table 4.7: 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.

Method/**Dataset**	**CIFAR-100**	**CIFAR-100 asym**			**CIFAR-100 sym**		**TinyImagenet**
% label noise	0%	10%	20%	40%	20%	40%	0%
ES	78.13 ±.4	71.91 ±.2	68.54 ±.3	51.53 ±.3	68.16 ±.4	61.17 ±.2	65.44 ±.3
TTA	79.21 ±.3	72.97 ±.1	71.00 ±.1	54.53 ±.1	70.14 ±.2	63.59 ±.1	65.67 ±.3
ES + TTA	79.11 ±.2	73.14 ±.2	70.46 ±.2	53.93 ±.5	70.21 ±.1	63.57 ±.1	65.74 ±.2
KF (ours)	78.71 ±.2	73.61 ±.1	71.24 ±.5	56.19 ±.8	72.21 ±.3	65.75 ±.1	69.00 ±.1
KF (ours) + TTA	**79.55** ±.3	**74.83** ±.1	**72.65** ±.2	**57.71** ±.3	**72.33** ±.2	**65.71** ±.1	**69.05** ±.3

Table 4.8: Mean test accuracy (in percent) and standard error of resnet 18, comparing our method with Early Stopping (ES) and Test Time Augmentation (TTA) on datasets with and without label noise.

Our analysis focused, for the most part, on overfit that exists even in the scenario in which test accuracy does not decrease as training proceeds - which means that ”early stopping” - culminating the training when performance over validation data decreases - has a minimal effect. Still, we wanted to compare our performance to early stopping (ES) on datasets with and without label noise. We also included in this comparison the test-time augmentation (TTA) method, in which a test example is classified several times, each time with a different augmentation, and given a final classification based on the average class probabilities of the different classifications. The results in Table 4.8 indicate that our method is comparable or better than both methods even when label noise exists in the training dataset (which leads to deteriorating performance as training proceeds) and that it is complementary to test-time augmentation.

4.2.6 model size

Method/**model size**	**small**	**base**	**large**
single network	83.21 ±.01	83.31 ±.15	82.92 ±.09
KF (ours)	83.17 ±.04	**83.57** ±.15	**83.96** ±.09

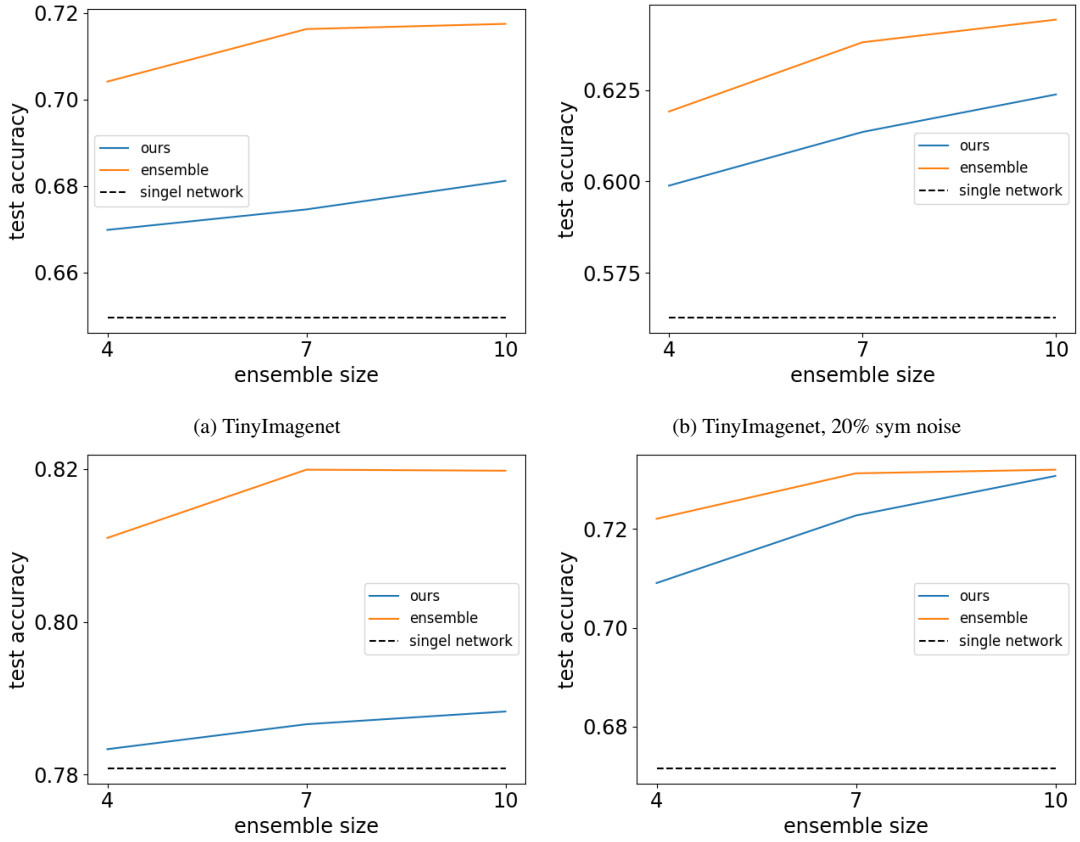
Table 4.9: Mean (over random validation/test split) test accuracy (in percent) and standard error on image classification datasets, comparing our method and the original predictor (ConvNeXt, trained on Imagenet) with varying number of parameters

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 overfit. In Fig. 3.4a we show that indeed larger versions of a model can cause increasing forget fraction, which also improves the benefit of our method (see Table 4.9), making it especially useful when one uses a large model.

4.2.7 Comparison to a regular ensemble

A regular ensemble, unlike ours, requires multiple training of independent networks, which could be unfeasible. A regular ensemble can be seen as an "upper bound" for our method's performance [17]. In Fig. 4.2, we compare our method and a regular ensemble of the same size, showing our method can achieve much of the performance gain provided by the regular ensemble. Notably, when label noise occurs our method can add most, if not all, of the regular ensemble performance gain.



(a) TinyImagenet

(b) TinyImagenet, 20% sym noise

(c) CIFAR100

(d) CIFAR100, 20% asym noise

Figure 4.2: Comparing our method with a limited number of checkpoints and an ensemble of the same size of independent networks

4.2.8 Fairness

In this section, we study our method's effect on the model's *fairness*, i.e. the effect non-relevant features have on the classification of test data examples. We follow [52] and train and test our models on datasets in which they might learn spurious correlations. To create those datasets, we divide the classes into two groups: in each class of the first group 95% of the training images go through a transformation (and the rest remain unchanged), and vice versa for the classes of the second group. The transformations we use are: removing color, lowering the image resolution (by down sampling and up sampling), and replacing images with downsampled images for the same class in Imagenet. We use cifar10 in our evaluation as done in [52], and use also cifar100 with the remove color transformation (the rest of the transformations were less appropriate for this dataset, as it contains similar classes that could actually become harder to separate at a lower resolution). We use the same method as before for our validation data, and thus the validation is of the same distribution as the test data.

Our evaluation uses the following metrics: (i) the test accuracy on two test sets (with/out the transformation), which should be lower if the model learns more spurious correlations, and (ii) the amplification bias defined in [59], which is defined as follows:

$$\frac{1}{|C|}\sum_{c\in C}\frac{max(c_T,c_N)}{c_T+c_N}-0.5 \tag{4.1}$$

When C is the group of classes, c_T is the number of images from the transformed test set predicted to be of class c, and c_N is the number of images from the natural test set predicted to be of class c - we would like those to be as close as possible since the transformation shouldn't change the prediction, and thus the lower the score the better.

The results of our evaluation are presented in Table 4.10. To summarize, our method improves the average performance on both datasets without deteriorating the amplification bias, which indicates that our method has no negative effects on the model's fairness.

Dataset/**Method**	**original model**			**KF**		
evaluation metric	natural	transformed	bias	natural	transformed	bias
cifar10 w/o color	89.07±.48	87.98±.38	0.07±.001	89.90±.40	87.85±.48	0.07±.002
cifar10 center cropped to 28x28	88.45±.31	70.44±.44	0.13±.003	88.92±.32	70.21±.74	0.13±.004
cifar10 downsampled to 16x16	85.43±.32	76.70±.13	0.08±.001	86.55±.27	77.47±14	0.07±.001
cifar10 downsampled to 8x8	80.061±.33	52.03±.49	0.22±.002	81.48±.44	52.99±.49	0.21±.003
cifar10 with Imagenet replacements	88.45±.31	70.44±.44	0.13±.003	88.92±.32	70.44±.44	0.13±.004

Table 4.10: Mean (over random validation/test split) test accuracy and amplification bias (in percent) and standard error on natural and transformed test sets, comparing our method and the original model.

4.3 Discussion and limitations

Our method provides a significant improvement of around 1% on modern neural networks over Imagenet, which typically implies more than 5% reduction in test error. It is often complementary to other methods (such as Test Time Augmentation) that aim to reduce overfit and often succeeds in reducing overfit when such methods (as EMA) fail (see Section 4.2.4 and App 4.2.5). Our method is especially useful in transfer learning settings when only a few layers are finetuned, as it: (i) significantly improves performance; and (ii) **adds very little overhead in inference and memory costs**, since most of the model is saved used at inference time only once.

In more difficult settings, such as a small network for complex data (e.g., Resnet18 over Tiny-Imagenet) or datasets with label noise, our method improves performance even further, leading to a nice error reduction of $\sim$15% in the reasonable setting of 10% asymmetric noise. This may be the case when dealing with complex and confusing natural datasets such as Animal10N [45].

When compared to baselines, we achieve comparable or better results (see, for example, our improvement in ViT16 over Imagenet and Resnet18 over TinyImagenet). Another large advantage of our method is that it is **independent of training choices**. In contrast, the horizontal method seems to show little improvement when the cosine-annealing learning rate scheduler is used (all experiments but Resnet50 over Imagenet), while fixed-jumps with no cycles [17] are known to show little improvement (or none at all) when step-size learning rate scheduler is used (Resnet50 over Imagenet in our experiments). Finally, while the more complicated baselines proved more useful than the other baselines (but not more than our method), this was only made possible by an extensive hyperparameter search.

While being simple and useful, our method has a few limitations as it requires: (i) validation data to tune the hyper-parameters; (ii) using multiple checkpoints, which incurs overhead in inference time and memory usage; (iii) the occurrence of forgetting to have any impact. These limitations can be mitigated, however, as shown in Section 4.2: (i) A subset of the train set can be effectively used for hyper-parameter tuning, followed by a full training (on both train and validation datasets) that provides optimal results. (ii) A few checkpoints can already achieve most of the method's benefit. We note that these checkpoints can run in parallel on multiple GPUs, and the memory overhead is minor compared to the usually huge training datasets. For example, the Imagenet dataset weighs 167GB[3], while a regular resnet50 weighs around 230MB. (iii) The method can revert to the original single network if no improvement on a validation set is seen, a benefit lacking in the methods listed in Table 4.3, as it doesn't require any changes to the learning process. Lastly, our empirical evaluation shows that our method has no negative effects on the model's fairness, making it safe to use.

[3]https://www.kaggle.com/competitions/imagenet-object-localization-challenge/data.

5 MAP: using ensembles's training history to overcome overfit in inference time

Following our observation on the effects of overfit on ensembles, in this section we present a new ensemble-based classifier to avoid some of the damages of overfit during inference time. We present our Algorithm in Section 5.1 and perform extensive evaluations, which prove its usefulness. In Section 5.2 we perform an elaborate ablation study on our method's cost and generality. We conclude in Section 5.3 with a discussion on our method's advantages and limitations.

5.1 Proposed method (MAP)

We present a new ensemble classifier algorithm called Max Agreement Prediction (MAP) in Section 5.1.1. Extensive experiments, demonstrating its superior performance, are described in Sections 5.1.2-5.1.3.

5.1.1 Algorithm

Our empirical results, as seen in Figure 3.9, show that the agreement margin score in (3.16) can be reliably used to identify false predictions. We take this result one step further and propose to use the ensemble statistics of agreement score in order to select the label prediction, instead of the usual practice of selecting $N^E(\mathbf{x})$ from (3.2). Specifically, we propose the following prediction selection rule:

$$y(\mathbf{x}) = \underset{c}{\operatorname{argmax}}\, Agr(\mathbf{x}, c) \tag{5.1}$$

We use this score in Alg. 3 and test it in Section 5.1.3, where its boost in performance is shown.

Algorithm 3: Max Agreement Prediction (MAP)

Input: $f_i^e(x)$ - the prediction of network i for test example $\mathbf{x} \in \mathbb{T}$, $\forall \mathbf{x}, i, e \in \mathcal{E}$
Output: final prediction $\forall \mathbf{x} \in \mathbb{T}$
for x **in** $\mathbb{T}$ **do**
  final_prediction_arr[x] $\leftarrow \underset{c}{\operatorname{argmax}}\, Agr(\mathbf{x}, c)$
end for
Return final_prediction_arr;

5.1.2 Empirical setup

To evaluate our method with different levels of overfit we use image and nlp classification datasets with injected noisy labels (using cifar10/100, TinyImagenet, Imagenet100, MNLI, QNLI, QQP) and with native label noise (Webvision50, Clothing1M, Animal10N). As in all empirical studies and in order to evaluate the generalization error correctly, the test set is kept clean. In Webvision, following [35], we used only the first 50 classes from Flickr and Google - around 100,000 images.

In our experiments, the ensemble contained 5 networks and was trained for 200 epochs (except clothing, on which it was trained for 80 epochs). We used a batch-size of 32, a learning rate of 0.01, an SGD optimizer with momentum of 0.9 and weight decay of 5e-4, a cosine annealing scheduler, and standard augmentations (horizontal flip, random crops). For cifar10/100, TinyImagenet, and Imagenet100 we used DenseNet with a width of 32 and batch normalization layers. For Clothing1M we used Resnet50 pretrained on Imagenet, while for Animal10N and Webvision we trained resnet50 from scratch. For elr [27] we used their implementation and hyperparameters. For HyperEnsembles we used different values of learning rate, ranging from 0.01 to 0.1 at equal distances, using ensembles of 4 networks. For Batchensembles, we used 4 networks of wide-resnet, at the recommended settings. For the NLP classification tasks, we used bert-base-cased [7] with hugging face [42] implementation as our model, with a batch size of 32, learning rate of 2e-5, max sequence length of 128, and 4 training epochs. We tested the networks every $\frac{1}{10}$ epoch. Experiments were conducted on a cluster of GPU type AmpereA10. Each experiment reports the mean and standard error (ste) results over 3 repetitions.

We compare our method's performance (with 5 networks) to the obvious baselines: (i) A single network of the same architecture and similarly trained. (ii) The same ensemble while using the common prediction rule $N^E(\mathbf{x})$ of 'Majority Vote' as defined in (3.2). (iii) The same ensemble while using the popular 'Class probabilities average' rule defined e.g. in [23]. The last 2 classifiers are versions of the 'regular ensemble', which incurs the same training cost as our method. Note that the 'regular ensemble' is chosen as it is a fair comparison in terms of *training time*. Most results are reported in Tables 5.1-5.4. A comparison with comparable *inference time*, using a much larger ensemble that is *almost infeasible to train*, is reported in Table 5.3.

In addition to the final performance, we check the different methods for overfit by looking for deterioration in performance as learning proceeds. All networks are trained with common methods to avoid overfit - data augmentation, batch normalization, and weight decay. This way we are able to test our method's *added value* as compared to these methods.

Additionally, we compare our method to alternative post-processing methods, which also aim to improve classification at inference time. To assure a fair comparison, all the results are obtained using the same network models in the ensemble, which is most relevant to the methods that make use of an ensemble as we do [25, 41]. The method described in [3] is executed several times with random seeds, the results of which are processed by the majority vote as in a

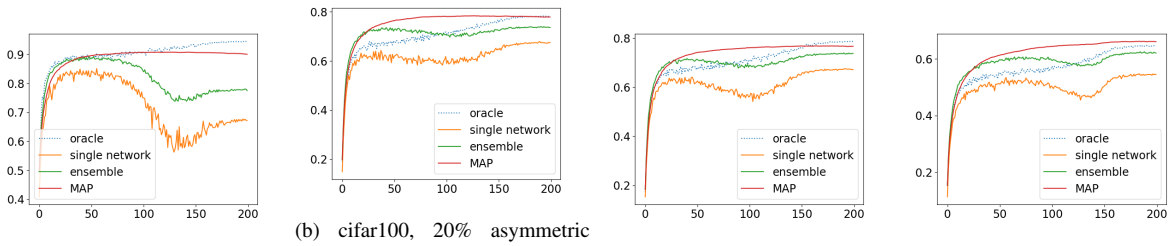
regular ensemble. To the best of our knowledge, no other method uses the training history of an ensemble to improve classification at inference time, to be used as a baseline.

Lastly, we test our method on two distinct methods for training ensembles: batch ensemble [54] and hyper ensemble [55]. The first method aims to reduce costs, whereas the second aims to increase the ensemble's diversity by using different hyper-parameters as well as different initializations.

5.1.3 Empirical results

Figure 5.1 and Table 5.1 summarize the comparison to the basic baselines. When the performance drop becomes more severe (due to increased label noise), our method significantly outperforms the regular ensemble (both majority vote and class probabilities average) at the end of the training. Table 5.2 summarizes the comparison to alternative post-processing methods.

Importantly, note that MAP eliminates the overfit in almost all cases: when inspecting the case studies shown in Figure 5.1, we clearly see that the test accuracy does not deteriorate when MAP is used, unlike a single network and the regular ensemble, making it a superior alternative to an early stopping of the training. Only in severe, unrealistic settings of label noise (such as 40% asymmetric noise) do we see a deterioration in our method's performance in the late stages of the training (though MAP still outperforms the regular ensemble), but such cases are not common.



(a) cifar10, 40% symmetric noise (b) cifar100, 20% asymmetric noise (c) cifar100, 20% symmetric noise (d) TinyImg, 20% symmetric noise

Figure 5.1: Mean test accuracy of our method (MAP, in red) in each training epoch (X-axis), compared with the following baselines: (i) A single network (orange). (ii) An ensemble (green). (iii) An oracle of a *single network* trained on the clean subset of the data (dotted blue). Our method shows significant *and stable* improvement, and **largely eliminates the overfit caused by the noisy labels**. Note that even though in some cases the test accuracy of the network/ensemble improves at the late stages of the training, it is often inferior compared to its past performance, and is *always* inferior to our method's performance.

Method/**Dataset**	**CIFAR-10 sym**			**CIFAR-100 sym**			**Clothing1M**
% label noise	20%	40%	60%	20%	40%	60%	38% (est)
single network	85.4±.1	67.7±.6	43.5±.5	67.0±.2	51.1±.2	32.2±.4	65.1±.1
majority vote ensemble	90.1±.2	78.5±.7	54.0±.4	73.5±.2	61.2±.4	42.8±.2	66.0±.2
probability average ensemble	90.5	79.5	56.6	74.9	64.2	46.3	67.1
MAP	**93.2±.1**	**90.0±.1**	**83.8±.7**	**76.7±.1**	**69.7±.5**	**60.0±.4**	**71.7±.1**

Method/**Dataset**	**TinyImagenet sym**		**Imagenet100 sym**	
% label noise	20%	40%	20%	40%
single network	54.5±.7	40.3±.2	76.4±.5	64.9±.1
majority vote ensemble	62.0±.1	50.3±.4	82.8±.2	75.5±.5
probability average ensemble	64.0	53.2	83.9	77.9
MAP	**66.0±.1**	**60.1±.1**	**83.9±.1**	**80.9±.3**

Method/**Dataset**	**CIFAR-10 asym**			**CIFAR-100 asym**			**Animal10N**
% label noise	10%	20%	40%	10%	20%	40%	8% (est)
single network	90.8±.1	83.1±.2	59.7±.3	74.1±.2	67.5±.2	47.5±.1	86.0
majority vote ensemble	93.5±.1	88.4±.2	64.0±.4	79.0±.1	73.7±.1	53.4±.2	87.3
probability average ensemble	93.4	88.6	63.3	79.8	74.5	52.9	87.9
MAP	**95.2±.1**	**94.4±.2**	**85.6±.2**	**80.6±.08**	**77.7±.1**	**57.5±.2**	87.4

Method/**Dataset**	**mnli**		**qnli**		**qqp**		**Webvision50**
% label noise	20%	40%	20%	40%	20%	40%	20% (est)
single network	79.3±.07	74.5±.06	86.0±.05	74.5±.05	85.7±.04	75.6±.04	78.0±.2
majority vote ensemble	81.6±.2	76.7±.2	87.3±.1	74.6±.4	88.1±.1	76.3±.1	85.3±.1
probability average ensemble	82.1	77.3	87.5	74.3	88.2	77.0	-
MAP	**82.6±.09**	**79.3±.1**	**89.0±.1**	**82.5±.08**	**88.8±.05**	**82.2±.1**	**86.5±.03**

Table 5.1: Mean test accuracy (in percent) and standard error, comparing our method (MAP) and the baselines. In the top 2 tables, we show results with 4 image datasets with injected label noise and 2 image datasets with presumed label noise marked by (est). In the bottom table, we show results with 3 text datasets with injected label noise and 1 image dataset with presumed label noise marked by (est).

Method/**Dataset**	**CIFAR-10 sym**		**CIFAR-100 sym**		**TinyImagenet sym**		**Clothing1M**
% label noise	20%	40%	20%	40%	20%	40%	38% (est)
RoG	87.4	81.8	64.3	55.6	-	-	68.0
consensus	87.4±.3	74.7±.3	71.0±.4	41.9±.1	23.7±.2	18.5±.3	66.9
NPC	89.8	77.5	73.7	61.5	62.1	50.3	70.8
MAP	**93.2±.07**	**90.0±.1**	**76.7±.1**	**69.7±.5**	**66.0±.1**	**60.1±.06**	**71.7±.2**

Table 5.2: Mean test accuracy (in percent) and standard error, comparing our method (MAP) to some of the post-processing methods discussed above. We report 3 image datasets with injected label noise and a single dataset with presumed annotation errors marked by (est). Source of alternative methods results: RoG from [25] and consensus from [41]. NPC [3] results are recomputed using our settings and our own implementation.

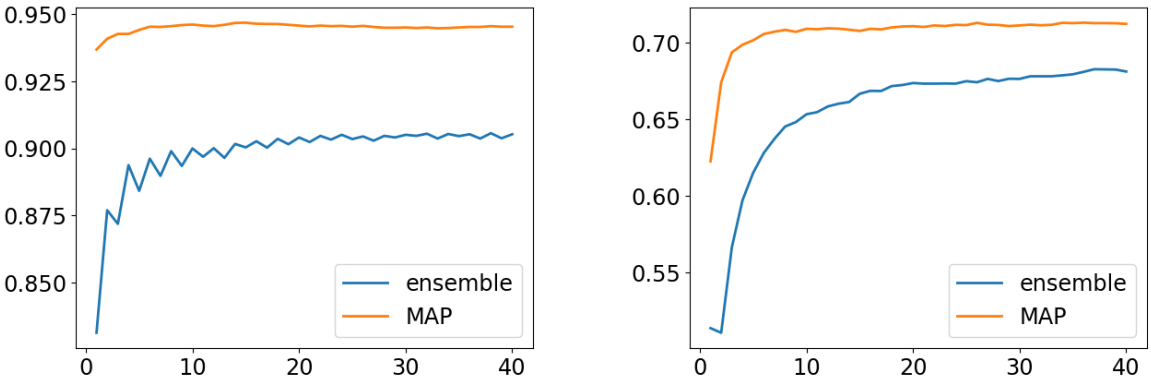
5.2 Ablation study

Our method requires the training of a few networks, which is computationally costly. This is justified by the improvement achieved by ensemble classifiers over a single classifier, as shown

above. In order to evaluate the practical implication of this added complexity, we investigated the effect of the number of networks in the ensemble on the final outcome of a regular ensemble and of our method, see Section 5.2.1.

When compared with alternative ensemble classifiers, our method incurs an additional cost as it requires that we save multiple checkpoints during the training, used to generate a time series of predictions. This is justified by the additional improvement achieved by our method as shown above. To evaluate the practical significance of this factor, we investigated the effect of using only a small subset of the training checkpoints, see Section 5.2.2. Additional results are described in Section 5.2.3 (effect of architecture and training choices) and Section 5.2.4 (performance without label noise).

Summarizing the ablation results, even with *a few networks* and *a few checkpoints for each*, our method achieves optimal or near-optimal performance, making it practical and useful with limited computational resources. Interestingly, our method can be combined with methods for learning with noisy labels, improving performance when such methods fail to eliminate overfit (see Table 5.4). Finally, the network architecture does not change the overall picture - our method maintains its benefit over the alternative deep ensemble.



(a) cifar10, 20% asymmetric noise

(b) cifar100, 40% symmetric noise

Figure 5.2: test accuracy of a regular ensemble and MAP when varying the size of the ensemble, where the x-axis denotes the number of networks in the ensemble.

5.2.1 The effect of ensemble size

In the results shown in Table 5.1, all ensemble classifiers use the same ensemble size (5), which shows the usefulness of adding the networks' history to the prediction at inference. However, to achieve a fair comparison between a regular ensemble and our method, we repeat the study without limiting the number of networks in the ensemble, reporting accuracy at the point where the addition of models does not improve performance, see results in Table 5.3. Notably, our

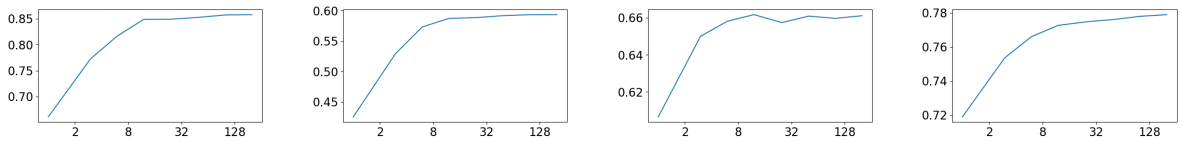
method maintains its superiority. Moreover, we see that with MAP - even a few networks achieve optimal or close to optimal results, making it practical for use (see Figures 5.2a-5.2b).

Method/**Dataset**	**CIFAR-10 sym**		**CIFAR-100 sym**	
% label noise	20%	40%	20%	40%
MAP (5)	93.3	90.0	76.6	70.1
majority vote (opt)	91.8	84.2	77.3	69.1
MAP (opt)	93.5	90.7	77.4	71.3
Method/**Dataset**	**CIFAR-10 asym**		**CIFAR-100 asym**	
% label noise	20%	40%	20%	40%
MAP (5)	94.4	85.7	77.9	57.3
majority vote (opt)	90.5	66.5	76.6	56.1
MAP (opt)	94.5	86.9	78.4	59.1

Table 5.3: Comparing ensembles with optimal size, denoted (opt), beyond which adding more networks does not improve the ensemble performance, to our method with 5 and opt networks.

5.2.2 How many checkpoints are needed?

The agreement score in (3.3) can be estimated from any subset of epochs. Figure 5.3 shows test accuracy when using a subset of the training epochs, equally spaced through time. These results demonstrate that only a few checkpoints are sufficient to achieve the same accuracy (or close to it) as when using all the epochs. The method is quite robust to the selection of epochs: we considered eliminating early epochs but observed that this had no significant effect on performance, and only increased the number of hyperparameters.



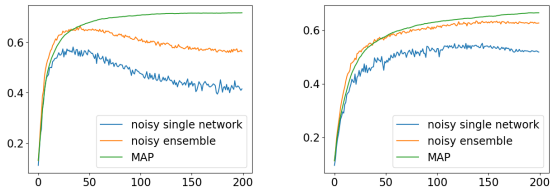
(a) cifar10 40% AsN (b) cifar100 60% SN (c) TinyImg 20% SN (d) cifar100 20% AsN

Figure 5.3: Test accuracy of MAP when using only a fraction of the epochs (equally spaced) per number of epochs. 'SN' denotes symmetric noise, and 'AsN' asymmetric noise.

5.2.3 The role of architecture and hyperparameters

Figure 5.4 shows results from experiments with different network architectures or learning rate schedulers. In all cases, our method still successfully eliminates the overfit.

We have also tested our method on a few other settings of interest:



(a) cifar100, 40% noise, const lr (b) cifar100, 40% noise, resnet32

Figure 5.4: Test accuracy of MAP, different lr (a) and architecture (b).

- We tested our method on ensembles of different network architectures/sizes trained in the same manner, and saw that the method is still effective in its performance improvement, showing comparable improvement to the one presented in Table 5.1.
- Transfer learning - we tested our method when initial weights are taken from pretraining (Imagenet), and saw our method still provides a large improvement in its final performance.
- Using different optimizers - we also tested our method when using a different optimizer (AdamW) - our method still shows a large improvement in performance.
- Having an adversarial network in the ensemble - interestingly, we also saw that due to the usage of multiple networks, our method is robust to the usage of a network that is poorly trained, or even provides random predictions, which is another advantage of our method.
- Using a different ensembling training method - we tested our method on two modern ways of training ensembles, that aim to reduce their training cost [54] and improve their performance [55]. In both cases, our method improved the original performance, showing it is complementary to such methods.
- Using a special training for label noise - we tested our method on an ensemble of networks trained with ELR [27], a special training method for label noise. Our method improves performance even in this case.

The full results appear in Tables 5.4-5.6. All experiments in Table 5.6 were done with densenet121 trained on cifar100 with 40% symmetric noise, except transfer learning, where our network choice was resnet18 pre-trained on Imagenet.

Method/**Dataset**	**TinyImagenet**		
% label noise (sym)	20%	40%	60%
elr	57.5	47.8	25.9
elr ensemble	61.6	54.0	33.3
elr + MAP	**62.3**	**57.4**	**47.2**

Table 5.4: A competitive method for label noise - elr [27], and its performance with and without MAP.

Method/**method**	**BatchEnsemble**		**Hyper-ensemble**	
% label noise (sym)	20%	40%	20%	40%
original accuracy	69.5	52.8	72.1	61.3
MAP	**72.5**	**60.1**	**74.1**	**67.6**
Method/**method**	**BatchEnsemble**		**Hyper-ensemble**	
% label noise (asym)	10%	20%	10%	20%
original accuracy	75.7	67.8	76.9	71.2
MAP	**77.8**	**71.1**	**77.5**	**75.3**

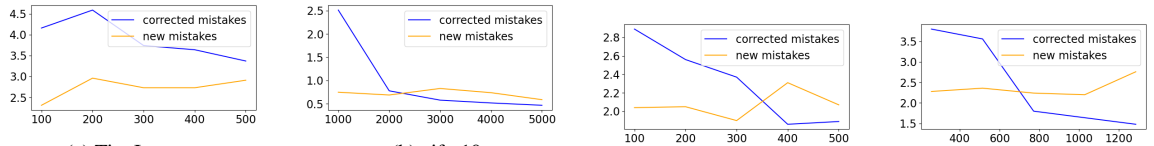
Table 5.5: testing our method on unique ensembles designed to decrease cost/increase diversity, trained on CIFAR100 with injected label noise. Our method is superior compared to those ensembles, both in the unique training setting and in the general one.

Method/**Test**	**Adversarial network**	**Transfer learning**	**Different architectures**	**Different optimizer**
single network	52.0	54.9	52.2	29.4
ensemble	59.9	63.9	61.4	42.6
MAP	**68.7**	**70.1**	**68.9**	**58.5**

Table 5.6: Additional ablation results. In the adversarial network setting, one network provides random predictions. In general, the single network shows the best network in the ensemble.

5.2.4 Performance evaluation on clean datasets

We evaluate our method on clean datasets of different sizes, where overfit can be much smaller or nonexistent. Results are shown in Figure 5.5. Compared to a regular ensemble, MAP is mostly better or comparable. We believe the improvement in performance in smaller datasets is caused by the overfit caused by the small amounts of data (another known source of overfit), making our method useful in such cases as well.



(a) TinyImagenet (b) cifar10 (c) cifar100 (d) Imagenet with 50 classes

Figure 5.5: x-axis: number of data points per class. y-axis: fraction of predictions of a regular ensemble, which MAP changes from true to false (orange) and from false to true (blue). MAP outperforms the ensemble whenever the blue curve lies above the orange curve.

5.3 Discussion and limitations

Our empirical study focused for the most part, though not entirely, on datasets with noisy labels, a real-world condition that creates heavy overfitting, where our method is most useful. In almost

all cases, the method's resistance to overfit is maintained even when the level of noise is very high. In such cases, our method significantly outperforms regular ensembles, as well as other post-processing methods designed specifically to handle noisy labels.

Our method has some practical advantages. Notably, it is easy to implement and can be readily used at post-processing with almost any method, network, and dataset. This is accomplished *without any change to the training process, without prior assumptions on the existence of label noise in the training data and its properties, and without any additional hyper-parameter tuning*. Since our method is only employed at post-processing, when no overfit is suspected it can be completely avoided, without any cost. Lastly, unless extreme overfit occurs, MAP also eliminates the need for a clean validation set, as it does not introduce new hyper-parameters and does not require "early stopping", often even surpassing the early stopping performance. Thus, it serves as a practical tool to combat overfit (with or without label noise in the training set), especially when large amounts of correctly labeled data are difficult to acquire.

Our method has two main limitations: (i) the need for multiple network training, and (ii) the need to save multiple checkpoints of the network during training to be used at inference time. Importantly, using multiple GPUs can mitigate these limitations, as the networks are trained independently and the checkpoints are inspected independently at inference time. Not less importantly, in Sections. 5.2.1-5.2.2 and Table 5.5 we show that training a few networks and saving only a few checkpoints for each one (equally spaced through time), suffice for optimal or almost optimal results; and that ensemble methods that minimize the training cost can go along well with our method, making our method much more practical.

6 Conclusions and Future Work

In this work, we revisited the tradeoff between using extended learning and/or larger models for a more capable model (hopefully achieving better performance) with the danger of ”overfitting” on the training data (potentially getting lower performance). We attempted to bridge the gap between the classical ML theory - i.e. the bias-variance tradeoff - and the empirical results from deep learning, by empirically showing that neural networks *do* overfit as training proceeds and show deteriorated performance on some sub-spaces of the data population even if their general performance still improves. This phenomenon, we believe, is deeply connected with the ”epoch-wise double descent phenomenon”, as both (we hypothesize) are caused by *simultaneous learning of general and specific (wrong?) features from the training data*.

To further boost our analysis, we examined the effect of overfitting on ensembles of neural networks, showing that agreement between different members of the ensemble decreases as overfitting occurs, up to the point where early stopping can often be accurately predicted using solely the average maximum agreement in the ensemble on test data (without using its true labels). We concluded this analysis with the observation that correct predictions (during inference time) can often be distinguished from incorrect ones by their *persistence over time in the ensemble*.

Our results indicate that the common practice of ”early stopping”, i.e. concluding training when the validation accuracy begins to drop, can be suboptimal as useful information is learned by the network even during the ”overfitting” phase; and also that useful information is ”forgotten” by the network even when the model seems to be improving and not overfitting (the validation accuracy is rising). To address these issues, we developed two simple methods that combine multiple checkpoints from one/multiple networks into an ensemble, which proved to: (a) be useful in combating overfit; (b) perform better than early stopping; (c) achieve comparable or superior performance when compared to similar methods, without any need for hyper-parameter tuning for training and without altering the training process.

For future work, we see two interesting directions: on the practical side, one can come up with interesting approaches for combining the two methods, optimize them (better ensembling?), and minimize their inference and memory costs (knowledge distillation from the different checkpoints/ensemble?). On the theoretical/empirical side, we propose further investigation of the subspaces ”forgotten” by the network during training, as their characterization can be useful to further push our understanding of overfit in the modern era of deep learning.

Bibliography

[1] Annavarapu, C. S. R. (2021). Deep learning-based improved snapshot ensemble technique for covid-19 chest x-ray classification. *Applied Intelligence*, 51:3104–3120.

[2] 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*, pages 233–242. PMLR.

[3] Bae, H., Shin, S., Na, B., Jang, J., Song, K., and Moon, I.-C. (2022). From noisy prediction to true label: Noisy prediction calibration via generative model. In *International Conference on Machine Learning*, pages 1277–1297. PMLR.

[4] 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.

[5] Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

[6] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

[7] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

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

[9] Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139.

[10] Ganaie, M. A., Hu, M., et al. (2021). Ensemble deep learning: A review. *arXiv preprint arXiv:2104.02395*.

[11] 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.

[12] 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.

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

[14] Hacohen, G., Choshen, L., and Weinshall, D. (2020). Let's agree to agree: Neural networks share classification order on real datasets. In *Int. Conf. Machine Learning ICML*, pages 3950–3960.

[15] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

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

[17] 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*.

[18] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR.

[19] 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*.

[20] Karimi, D., Dou, H., Warfield, S. K., and Gholipour, A. (2020). Deep learning with noisy labels: Exploring techniques and remedies in medical image analysis. *Medical Image Analysis*, 65:101759.

[21] Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images. *Online*.

[22] Krogh, A. and Hertz, J. (1991). A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4.

[23] Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30.

[24] Le, Y. and Yang, X. (2015). Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3.

[25] Lee, K., Yun, S., Lee, K., Lee, H., Li, B., and Shin, J. (2019). Robust inference via generative classifiers for handling noisy labels. In *International Conference on Machine Learning*, pages 3763–3772. PMLR.

[26] Li, H., Wang, X., and Ding, S. (2018). Research and development of neural network ensembles: a survey. *Artificial Intelligence Review*, 49(4):455–479.

[27] Liu, S., Niles-Weed, J., Razavian, N., and Fernandez-Granda, C. (2020). Early-learning regularization prevents memorization of noisy labels. *Advances in neural information processing systems*, 33:20331–20342.

[28] 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)*.

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

[30] 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.

[31] Nguyen, T. and Pernkopf, F. (2020). Lung sound classification using snapshot ensemble of convolutional neural networks. In *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 760–763. IEEE.

[32] Nicholson, B., Sheng, V. S., and Zhang, J. (2016). Label noise correction and application in crowdsourcing. *Expert Systems with Applications*, 66:149–162.

[33] 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.

[34] Patrini, G., Rozza, A., Krishna Menon, A., Nock, R., and Qu, L. (2017). Making deep neural networks robust to label noise: A loss correction approach. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1944–1952.

[35] Pleiss, G., Zhang, T., Elenberg, E., and Weinberger, K. Q. (2020). Identifying mislabeled data using the area under the margin ranking. *Advances in Neural Information Processing Systems*, 33:17044–17056.

[36] Pliushch, I., Mundt, M., Lupp, N., and Ramesh, V. (2021). When deep classifiers agree: Analyzing correlations between learning order and image statistics. *arXiv preprint arXiv:2105.08997*.

[37] Polikar, R. (2012). Ensemble learning. *Ensemble machine learning: Methods and applicannavarapu2021deepations*, pages 1–34.

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

[39] Ratcliff, R. (1990). Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285.

[40] Rouzbahani, H. M., Bahrami, A. H., and Karimipour, H. (2021). A snapshot ensemble deep neural network model for attack detection in industrial internet of things. *AI-Enabled Threat Detection and Security Analysis for Industrial IoT*, pages 181–194.

[41] Salman, S. and Liu, X. (2019). Overfitting mechanism and avoidance in deep neural networks. *arXiv preprint arXiv:1901.06566*.

[42] Sanh, V., Webson, A., Raffel, C., Bach, S. H., Sutawika, L., Alyafeai, Z., Chaffin, A., Stiegler, A., Scao, T. L., Raja, A., Dey, M., Bari, M. S., Xu, C., Thakker, U., Sharma, S. S., Szczechla, E., Kim, T., Chhablani, G., Nayak, N., Datta, D., Chang, J., Jiang, M. T.-J., Wang, H., Manica, M., Shen, S., Yong, Z. X., Pandey, H., Bawden, R., Wang, T., Neeraj, T., Rozen, J., Sharma, A., Santilli, A., Fevry, T., Fries, J. A., Teehan, R., Biderman, S., Gao, L., Bers, T., Wolf, T., and Rush, A. M. (2021). Multitask prompted training enables zero-shot task generalization.

[43] Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48.

[44] Sollich, P. and Krogh, A. (1995). Learning with ensembles: How overfitting can be useful. *Advances in neural information processing systems*, 8.

[45] Song, H., Kim, M., and Lee, J.-G. (2019). Selfie: Refurbishing unclean samples for robust deep learning. In *International Conference on Machine Learning*, pages 5907–5915. PMLR.

[46] Song, H., Kim, M., Park, D., Shin, Y., and Lee, J.-G. (2022). Learning from noisy labels with deep neural networks: A survey. *IEEE Transactions on Neural Networks and Learning Systems*.

[47] 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.

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

[49] Toneva, M., Sordoni, A., Combes, R. T. d., Trischler, A., Bengio, Y., and Gordon, G. J. (2018). An empirical study of example forgetting during deep neural network learning. *arXiv preprint arXiv:1812.05159*.

[50] TorchVision (2016). Torchvision: Pytorch's computer vision library. https://github.com/pytorch/vision.

[51] Tu, Z., Talebi, H., Zhang, H., Yang, F., Milanfar, P., Bovik, A., and Li, Y. (2022). Maxvit: Multi-axis vision transformer. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXIV*, pages 459–479. Springer.

[52] Wang, Z., Qinami, K., Karakozis, I. C., Genova, K., Nair, P., Hata, K., and Russakovsky, O. (2020). Towards fairness in visual recognition: Effective strategies for bias mitigation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8919–8928.

[53] Wen, L., Gao, L., and Li, X. (2019). A new snapshot ensemble convolutional neural network for fault diagnosis. *Ieee Access*, 7:32037–32047.

[54] Wen, Y., Tran, D., and Ba, J. (2020). Batchensemble: an alternative approach to efficient ensemble and lifelong learning. *arXiv preprint arXiv:2002.06715*.

[55] Wenzel, F., Snoek, J., Tran, D., and Jenatton, R. (2020). Hyperparameter ensembles for robustness and uncertainty quantification. *Advances in Neural Information Processing Systems*, 33:6514–6527.

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

[57] 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.

[58] Zhang, M., Lee, J., and Agarwal, S. (2021). Learning from noisy labels with no change to the training process. In *International Conference on Machine Learning*, pages 12468–12478. PMLR.

[59] Zhao, J., Wang, T., Yatskar, M., Ordonez, V., and Chang, K.-W. (2017). Men also like shopping: Reducing gender bias amplification using corpus-level constraints. *arXiv preprint arXiv:1707.09457*.