SNAPSHOT ENSEMBLES: TRAIN 1, GET M FOR FREE

Gao Huang*, **Yixuan Li***, **Geoff Pleiss**
Cornell University
{gh349, yl2363}@cornell.edu, geoff@cs.cornell.edu

Zhuang Liu
Tsinghua University
liuzhuangthu@gmail.com

John E. Hopcroft, Kilian Q. Weinberger
Cornell University
jeh@cs.cornell.edu, kqw4@cornell.edu

ABSTRACT

Ensembles of neural networks are known to give far more robust and accurate predictions compared to any of its individual networks. However, training multiple deep networks for model averaging is computationally expensive. In this paper, we propose a method to obtain the seemingly contradictory goal to obtain ensembles of *multiple neural network at no additional training cost*. We achieve this goal by letting a single neural network converge into several local minima along its optimization path and save the model parameters. To obtain repeated rapid convergence we leverage recent work on cyclic learning rate schedules. The resulting technique, which we refer to as *Snapshot Ensembling*, is surprisingly simple, yet effective. We show in a series of experiments that our approach is compatible with diverse network architectures and learning tasks. It consistently yields significantly lower error rates than state-of-the-art single models at no additional training cost, and almost matches the results of (far more expensive) independently trained network ensembles. On Cifar-10 and Cifar-100 our DenseNet Snapshot Ensembles obtain error rates of 3.4% and 17.4% respectively.

1 INTRODUCTION

Stochastic Gradient Descent (SGD) (Bottou, 2010) and its accelerated variants (Kingma & Ba, 2014; Duchi et al., 2011) have become the de-facto approaches for optimizing deep neural networks. The popularity of SGD is typically attributed to its ability to avoid and even escape spurious saddle-points and local minima (Dauphin et al., 2014). Although this ability is generally considered positive, in this paper we argue that there may in fact be some value to these local minima and that simply leaving them on the way-side may be outright wasteful.

Although deep networks typically never converge to a global minimum, there is a notion of "good" and "bad" local minima in terms of generalization. Keskar et al. (2016) argue that a local minimum generalizes well if it has a flat basin. SGD can avoid sharper spurious local minima because its gradients are computed from small mini-batches and are therefore inexact (Keskar et al., 2016). If the learning-rate is sufficiently large, there will be enough intrinsic random motion across gradient steps that the optimizer will not be trapped in any of the sharper local minima along its optimization path. If however the learning rate is sufficiently small, then the model will converge into the closest local minimum. These two very different behaviors of SGD are typically exploited in different phases of the optimization (He et al., 2015). Initially the learning rate is kept high to move into the general vicinity of a flat local minimum. Once this search has reached a stage in which the optimization makes no more progress, the learning-rate is dropped (once or twice), which triggers a descent and ultimately a convergence into the final local minimum.

It is well established (Kawaguchi, 2016) that the number of possible local minima grows exponentially with the number of parameters—of which modern neural networks can have millions. It is therefore not surprising that two identical architectures optimized from different initializations or

*Authors contribute equally.

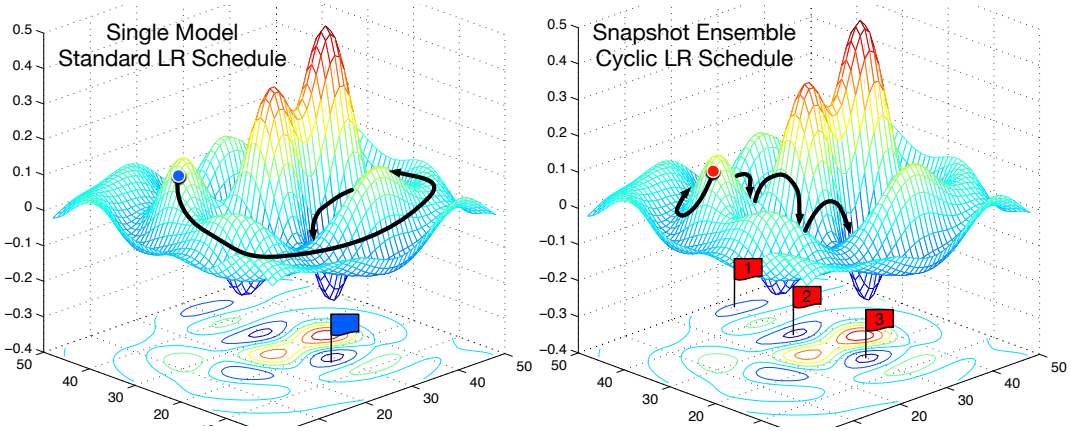

Figure 1: **Left:** Illustration of SGD optimization with a typical learning rate schedule. The model converges to a minimum at the end of training. **Right:** Illustration of Snapshot Ensembling optimization. The model undergoes several learning rate annealing cycles, converging to and escaping from multiple local minima. We take a snapshot at each minimum for test time ensembling.

even just with a different order of minibatches will converge to different solutions. Although different local minima often lead to very similar error rates, the corresponding neural networks tend to make very different mistakes. This diversity can be exploited through ensembling, in which multiple neural networks are trained from different initializations and then combined with majority voting or averaging (Caruana et al., 2004). Ensembling often leads to drastic reductions in error rates. In fact, most high profile competitions, e.g. Imagenet (Deng et al., 2009) or Kaggle[1], are won by ensembles of deep learning architectures. Despite its obvious advantages, the use of ensembling for deep networks is not nearly as wide-spread as for other algorithms. One likely reason for this lack of adaptation may be the additional training cost incurred by ensembling. Training deep networks is computationally expensive and can last for weeks, even on high performance hardware with GPU acceleration. As the training cost for ensembles increases linearly, ensembles can quickly becomes uneconomical for most researchers without access to industrial scale computational resources.

In this paper we propose a method to achieve the seemingly contradictory goal to obtain an ensemble of multiple neural networks *without incurring any additional training costs*. Our approach leverages the non-convex nature of neural networks and the ability of SGD to converge into and escape from local minima on demand. Most importantly, it is compellingly simple and straight-forward to implement. Instead of training M neural networks independently from scratch, we let SGD converge M times into local minima along its optimization path. Each time the model converges we save the weights and add the corresponding network to our ensemble. We then restart the optimization with a large learning rate to escape the current local minimum and continue the optimization. To achieve this cyclic learning, we adopt the aggressive cosine annealing procedure suggested by Loshchilov & Hutter (2016), which lowers the learning rate continuously following a cosine function. Because our final ensemble consists of snapshots of the optimization path, we refer to our approach as *Snapshot Ensembling*. Figure 1 presents a high-level overview of this method.

In contrast to traditional ensembles, the training time for the entire ensemble is identical to the time required to train a *single* traditional model. During testing time, one can evaluate and average the last (and therefore most accurate) m out of M models. Our approach is naturally compatible with other methods to improve the accuracy, such as data augmentation, stochastic depth (Huang et al., 2016b), or batch normalization (Ioffe & Szegedy, 2015). In fact, Snapshot Ensemble can even be ensembled, if for example parallel resources are available during training. In this case, an ensemble of K Snapshot Ensembles yields $K \times M$ models at K times the training cost.

We evaluate the efficacy of Snapshot Ensemble on three popular state-of-the-art deep learning architectures for object recognition: ResNet (He et al., 2016), Wide-ResNet (Zagoruyko & Komodakis, 2016), and DenseNet (Huang et al., 2016a). We show across four different data sets that Snapshot

[1] `www.kaggle.com`

Ensemble almost always leads to substantial reductions in error rates at the same training costs. For example, on Cifar-10 and Cifar-100 it obtains error rates of 3.44% and 17.41% respectively.

2 Related Work

Neural networks ensembles have been widely studied and applied in machine learning (Hansen & Salamon, 1990; Krogh et al., 1995). However, most existing works focuses on improving the generalization performance, while few of them address the cost of training ensembles.

As an alternative to traditional ensembles, so-called "implicit" ensembles have high efficiency during both training and testing (Srivastava et al., 2014; Wan et al., 2013; Huang et al., 2016b; Singh et al., 2016; Krueger et al., 2016). The Dropout (Srivastava et al., 2014) technique creates an ensemble out of a single model by "dropping" — or zeroing — random sets of hidden nodes during each mini-batch. At test time, no nodes are dropped, and each node is scaled by the probability of surviving during training. Srivastava et al. claim that Dropout reduces overfitting by preventing the co-adaptation of nodes. An alternative explanation is that this mechanism creates an exponential number of networks with shared weights during training, which are then implicitly ensembled at test time. DropConnect (Wan et al., 2013) uses a similar trick to create ensembles at test time by dropping connections (weights) during training instead of nodes. The recently proposed Stochastic Depth technique (Huang et al., 2016b) randomly drops layers during training to create an implicit ensembles of many networks with varying depth at test time. Stochastic Depth has a strong regularization effect and tends to outperform Dropout on very deep networks (Huang et al., 2016b). Finally, Swapout (Singh et al., 2016) is a stochastic training method which generalizes Dropout and Stochastic Depth. From the perspective of model ensemble, it provides more diversified network structures for model averaging. Our proposed method similarly trains only a single model; however, the resulting ensemble is explicit in that the models do not share weights. It is also compatible with implicit techniques and the approaches can be used in conjunction.

Several recent publications focus on reducing *test time cost* of ensembles, by transferring the "knowledge" of cumbersome ensembles into a single model (Bucilu et al., 2006; Hinton et al., 2015). Hinton et al. (2015) propose to use an ensemble of multiple networks as the target of the single (smaller) network. In order to produce more informative guidance, a high temperature is used to soften the probability distribution over classes. Our proposed method is complementary to these works as we aim to reduce the *training* cost of ensembles rather than the test time cost.

There are several recently proposed training mechanisms to improve the power of explicit ensembles. Laine & Aila (2016) propose a temporal ensembling method for semi-supervised learning, which achieves consensus among models trained with different regularization and augmentation conditions. Recently, Moghimi et al. show that boosting can be applied to convolutional neural networks to create strong ensembles.

Our work is inspired by recent work of Loshchilov & Hutter (2016) and Smith (2016), who show that cyclical learning rates can be effective for training convolutional neural networks. The authors show that each cycle produces models which are (almost) competitive to those learned with traditional learning rate schedules in a fraction of training iterations. Although the model performance temporarily suffers when the learning rate cycle is restarted, the performance eventually surpasses that of the previous cycle after annealing the learning rate. The authors suggest that cycling perturbs the parameters of a converged model, which allows the model to find a better local minimum. Our work builds upon these recent findings in non-linear function optimization, but we are not concerned with speeding up or improving the training of a single model but instead try to extract an ensemble of classifiers while following the optimization path of the final model.

3 Snapshot Ensembling

Snapshot Ensembling produces an ensemble of accurate and diverse models from a single training process. At the heart of Snapshot Ensembling is an optimization process which visits several local minima before converging to a final solution. We take model snapshots at these various minima, and average softmax predictions from each of them at test time.

Ensembles work best if their individual members have low test error and have little overlap in the set of examples they still misclassify. Almost all weight assignments along the optimization path of a neural network do not lead to low error models. In fact, it is commonly observed that the validation error drops significantly only after the learning rate has been reduced at least once, which is typically done after several hundred epochs. Our approach is inspired by the observation that training neural networks for fewer epochs—dropping the learning rate earlier—often increases the final error by only a small amount (Loshchilov & Hutter, 2016). This seems to suggest, that the local minima along the optimization path are already starting to become promising in terms of generalization error after only several epochs.

Cyclical Cosine Annealing. In our search for viable local minima we follow a cyclic annealing schedule as proposed by Loshchilov & Hutter (2016). We lower the learning rate at a very fast pace, encouraging the model to converge towards its first local minimum after as few as 50 epochs. The optimization is then continued at a larger learning rate, perturbing the model to snap out of its local minimum and the annealing cycle starts over. Formally, the learning rate α has the form:

$$\alpha(t) = f\left(\mathrm{mod}\left(t-1, \lceil T/M \rceil\right)\right), \qquad (1)$$

where t is the iteration number, T is the total number of training iterations, and f is a monotonically decreasing function. In other words, we split the training process into M cycles, each of which starts with a large learning rate value. Each cycle then undergoes a full annealing process before raising the learning rate at the beginning of the next cycle. The large learning rate $\alpha = f(0)$ in each stage gives the model enough energy to escape from a critical point, and the small learning rate $\alpha = f(T/M)$ will drive the model to a well behaved local minimum. The learning rate function in (1) can be any monotonically decreasing function. We adopt the shifted cosine function proposed by Loshchilov & Hutter (2016):

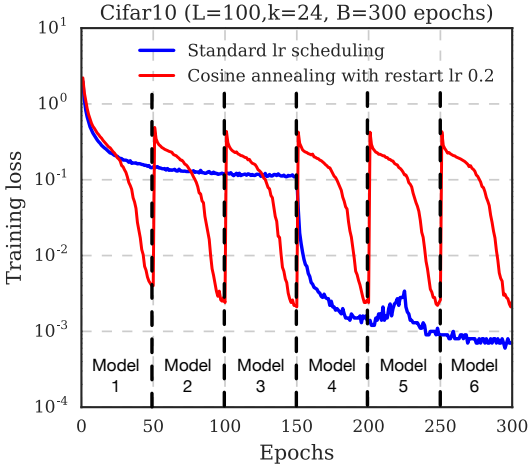

Figure 2: Training loss of 100-layer DenseNet on CIFAR10 when using standard learning rate (blue) and $M = 6$ cosine annealing cycles (red). The intermediate models, denoted by the dotted lines, are ensembled together at the end of training.

$$\alpha(t) = \frac{\alpha_0}{2}\left(\cos\left(\frac{\pi \mathrm{mod}(t-1, \lceil T/M \rceil)}{\lceil T/M \rceil}\right) + 1\right), \qquad (2)$$

where α_0 is the initial learning rate. Intuitively, this function anneals the learning rate from its initial value $\alpha_0$to $f(T/M) = 0$ over the course of a cycle. Note that the learning rate is updated at each iteration rather than at every epoch as suggested by Loshchilov & Hutter (2016). This improves the convergence of short cycles, even when a large initial learning rate is used .

Snapshot Ensembling. Figure 2 depicts the training process using cyclic and traditional learning rate schedules. At the end of each training cycle, it is apparent that the model reaches a local minimum with respect to training the loss. Thus, before raising the learning rate, we take a "snapshot" of the model weights (indicated as vertical dashed black lines). After training M cycles, we have M model snapshots, $f_1 \dots f_M$, each of which will be used in the final ensemble. It is important to highlight that the total training time of the M snapshots is the same as training a model with a standard schedule (indicated in blue). It is also fair to point out that the training loss under the standard schedule is eventually lower—however, as we will show in the next section, ensembling the M models more than compensates for that.

Ensembling at test-time. At test time, we compute the ensemble prediction by averaging the softmax outputs of the last m models. Let $\mathbf{x}$ be a test sample and let $f_i(\mathbf{x})$ be the softmax score of snapshot i. The output of the ensemble is a simple average of the last m models: $f_{Ensemble} = \frac{1}{m}\sum_0^{m-1} f_{M-i}(\mathbf{x})$. We always ensemble the last m models because the generalization error of the models tends to monotonically decrease.

	Method	C10	C100	SVHN	Tiny ImageNet
ResNet-110	Single model	5.52	28.02	1.96	46.50
	NoCycle Snapshot Ensemble	5.49	26.97	1.78	43.69
	Snapshot Ensemble ($\alpha_0 = 0.1$)	5.73	25.55	**1.63**	40.54
	Snapshot Ensemble ($\alpha_0 = 0.2$)	**5.32**	**24.19**	1.66	**39.40**
Wide-ResNet-32	Single model	5.43	23.55	1.90	39.63
	Dropout	4.68	22.82	1.81	36.58
	NoCycle Snapshot Ensemble	5.18	22.81	1.81	38.64
	Snapshot Ensemble ($\alpha_0 = 0.1$)	**4.41**	**21.26**	1.64	35.45
	Snapshot Ensemble ($\alpha_0 = 0.2$)	4.73	21.56	**1.51**	**32.90**
DenseNet-40	Single model	5.24*	24.42*	1.77	39.09
	Dropout	6.08	25.79	1.79*	39.68
	NoCycle Snapshot Ensemble	5.20	24.63	1.80	38.51
	Snapshot Ensemble ($\alpha_0 = 0.1$)	4.99	23.34	**1.64**	37.25
	Snapshot Ensemble ($\alpha_0 = 0.2$)	**4.84**	**21.93**	1.73	**36.61**
DenseNet-100	Single model	3.74*	19.25*	-	-
	Dropout	3.65	18.77	-	-
	NoCycle Snapshot Ensemble	3.80	19.30	-	-
	Snapshot Ensemble ($\alpha_0 = 0.1$)	3.57	18.12	-	-
	Snapshot Ensemble ($\alpha_0 = 0.2$)	**3.44**	**17.41**	-	-

Table 1: Error rates (%) on CIFAR-10 and CIFAR-100 datasets. All methods in the same group use the same amount of training time and cost. Results of our method are colored in blue, and best result for each network and each dataset is shown in **bold**. * indicates numbers directly taken from Huang et al. (2016a).

4 EXPERIMENTS

We demonstrate the effectiveness of Snapshot Ensembles on several benchmark datasets and compare it with competitive baselines. We run all experiments with Torch 7 (Collobert et al., 2011)[2].

4.1 DATASETS

CIFAR. The two CIFAR datasets (Krizhevsky & Hinton, 2009) consist of colored natural scene images sized at 32×32 pixels. CIFAR-10 (C10) and CIFAR-100 (C100)images are drawn from 10 and 100 classes respectively. For each dataset, there are 50,000 training images and 10,000 images reserved for testing. A standard data augmentation scheme is used as in (Lin et al., 2013; Romero et al., 2014; Lee et al., 2015; Springenberg et al., 2014; Srivastava et al., 2015; He et al., 2015; Huang et al., 2016b; Larsson et al., 2016): the images are first zero-padded with 4 pixels on each side, then randomly cropped to again produce 32×32 images; half of the images are then horizontally mirrored.

SVHN. The Street View House Numbers (SVHN) dataset (Netzer et al., 2011) contains 32×32 colored digit images from Google Street View, with 10 different classes for each digit. There are 73,257 images in the training set and 26,032 images in the test set. Following common practice (Sermanet et al., 2012; Goodfellow et al., 2013; Huang et al., 2016a), we withhold 6,000 training images for validation, and train on the remaining images without data augmentation.

Tiny ImageNet. The Tiny ImageNet dataset[3] consists of a subset of ImageNet ILSVRC images (Deng et al., 2009), with 200 classes. Each class has 500 training images, 50 validation images and 50 test images. Each image is resized to 64×64, with standard data augmentation including random crops, horizontal mirroring, and altering RGB intensities (Krizhevsky et al., 2012).

4.2 TRAINING SETTING

Architectures. We benchmark on several state-of-the-art architectures including residual networks (*ResNet*) (He et al., 2016), *Wide ResNet* (Zagoruyko & Komodakis, 2016) and *DenseNet* (Huang

[2]Code to reproduce results is available at `https://github.com/gaohuang/SnapshotEnsemble`

[3]`https://tiny-imagenet.herokuapp.com`

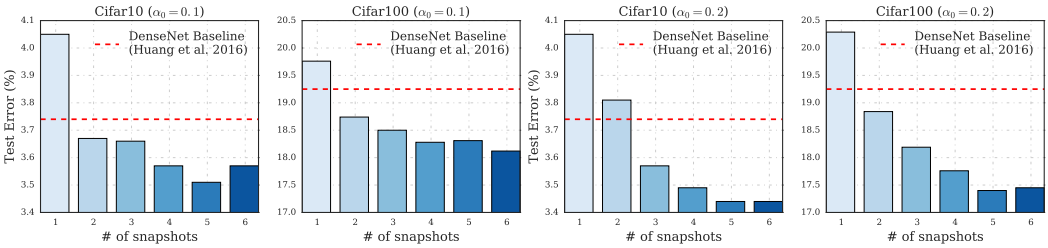

Figure 3: Snapshot Ensemble performance on CIFAR-10 and CIFAR-100 when trained on DenseNet-100 with restart learning rate 0.1 (left two) and 0.2 (right two) with $M = 6$ annealing cycles (50 epochs per each).

et al., 2016a). For ResNet, we use the original 110-layer network introduced by He et al. (2015). The Wide-ResNet) is a 32-layer ResNet with each of its convolutional layer four times wider (in terms of the number of output channels) than standard ResNets. For DenseNet, our large model in use is consistent with the original setup in (Huang et al., 2016a), with depth $L = 100$, growth rate $k = 24$. In addition, we also evaluate our method on a small DenseNet, with depth $L = 40$ and $k = 12$. For Tiny ImageNet, we add a stride of 2 to the first convolution of each model to downsample the images to a 32×32 size.

We use a batch size 64 for all the architectures and on all datasets, except the ResNet-110 and Wide-ResNet are trained with batch size 128 on Tiny ImageNet.

Baselines. Snapshot Ensembles incur the training cost of a single model; therefore, we compare with baselines that require the same amount of training. Firstly, we compare against a **Single Model** trained with a standard learning rate schedule, dropping the learning rate from 0.1 to 0.01 halfway through training, and then to 0.001 when training is at 75%. To compare against implicit ensembling methods, we test against a single DenseNet model trained with **Dropout**. This baseline uses the same learning rate as above, and drops nodes during training with a probability of 0.2.

We then test variants of the **Snapshot Ensemble** algorithm trained with the cyclic cosine cycle as described by Equation (2). We test models with the max learning rate α_0 set to 0.1 and 0.2. In both cases, we divide the training process into contingent learning rate cycles. We take a model snapshot at the end of each training cycle. Additionally, we train a Snapshot Ensemble with a non-cyclic learning rate schedule. This **NoCycle Snapshot Ensemble**, which uses the same schedule as the Single Model and Dropout baselines, is meant to highlight the impact of cyclic learning rates for our method. To accurately compare with the cyclical Snapshot Ensembles, we take six snapshots equally spaced throughout the training process. During test time, we feed samples through each model and average the softmax outputs.

Training Cost. The DenseNet-40 and DenseNet-100 baseline models are trained for a total of $B = 300$ epochs (150 for Tiny ImageNet). Snapshot variants were trained with $M = 6$ cycles of $B/M = 50$ epochs each (25 for Tiny ImageNet). Following the original training budget in (He et al., 2015; Zagoruyko & Komodakis, 2016), we train ResNet and Wide ResNet models for 200 epochs (150 for Tiny ImageNet). Snapshot variants of ResNet and Wide ResNet were trained with 5 cycles of 40 epochs each (6 cycles of 25 epochs for Tiny ImageNet).

On SVHN dataset, we consistently train all the models using a total training budget of $B = 40$ epochs. Snapshot variants in this case are trained with 5 cycles of 8 epochs each.

4.3 Snapshot Ensemble Results

Accuracy. The main results are summarized in Table 1. In all but one experiment, Snapshot ensembles achieve lower error than any of the baseline methods. Most notably, Snapshot Ensembling yields an error rate of 17.41% on CIFAR-100 using large DenseNets, far outperforming the state-of-the-art record of 19.25% under the same training cost and architecture (Huang et al., 2016a). Our method has the most success on CIFAR-100 and Tiny ImageNet, which is likely due to the complexity of these datasets. The softmax outputs for these datasets are high dimensional due to the large number of classes, making it unlikely that any two models make the same predictions. Nevertheless, Snapshot Ensembling is also capable of improving the competitive baselines for CIFAR-10 and SVHN, reducing error by up to 1% and 0.4% on Wide ResNet.

The NoCycle Snapshot Ensemble generally has little effect on performance, and in some instances even *increases* the test error. This highlights the need for a cyclical learning rate to discover solutions for useful ensembling.

Ensemble Size. In some applications, it may be beneficial to vary the size of the ensemble *dynamically* at test time depending on available resources. Figure 3 displays the performance of CIFAR DenseNets as the effective ensemble size, m, is varied. Each ensemble is comprised of snapshots from later cycles, as these snapshots have received the most training and converge to better minima. Although ensembling more models generally gives better performance, we observe significant drops in error when the second and third models are added to the ensemble. In most cases, an ensemble of two models outperforms the baseline model.

M	**Test Error (%)**
2	22.92
4	22.07
6	21.93
8	21.89
10	22.16

Table 2: Error rates (%) on CIFAR-100 using DenseNet-40 with varying number of cycles.

Restart Learning Rate. The effect of restart learning rate can be observed in Figure 3. The left two plots show the performance when using restart learning rate of $\alpha_0 = 0.1$ at the beginning of each cycle, and the bottom panel shows that of $\alpha_0 = 0.2$ restart learning rate. In most cases, we find that larger restart learning rate can lead to better ensemble performance, presumably due to larger diversity introduced by the stronger perturbation inbetween cycles.

Varying Number of Cycles Given a fixed training budget, there is a trade-off between the number of learning rate cycles and their length. Therefore, we investigate how the number cycles M affects the ensemble performance, give a training budget of $B = 300$ epochs. We train a 40-layer DenseNet on CIFAR100 dataset with an initial learning rate of $\alpha_0 = 0.2$. As shown in Table 2, our method is relatively robust with respect to different values of M. At the extremes, $M = 2$ and $M = 10$, we find a slight degradation in performance, as the cycles are either too few or too short. In practice, we find setting M to be $4 \sim 8$ works reasonably well.

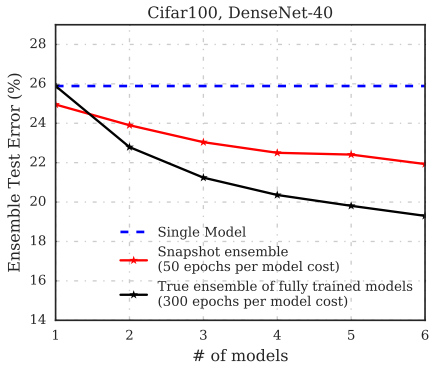

Figure 4: Performance comparison of Snapshot Ensembles with true ensembles.

Comparison with True Ensemble. We compare Snapshot Ensemble with the expensive traditional ensemble method. Figure 4 shows the test rates of DenseNet-40 on CIFAR100. The true ensemble method averages models that are trained with 300 full epochs, each with different weight initializations. Given the same number of models at test time, the error rate of the true ensemble can be seen as an lower bound of our method. Our method achieves with comparable performance of ensembling 3 independant models, but with the training cost of one model.

4.4 Diversity of Model Ensembles

Parameter Space. We hypothesize that the cyclical learning rate schedule creates snapshots which are not only accurate but also diverse with respect to model predictions. We qualitatively measure this diversity by visualizing the local minima they fall into. For that purpose we linearly interpolate snapshot models, as described by Goodfellow et al. (2014). Let $J(\theta)$ be the test error of a model using parameters θ. Given θ_1 and θ_2 — the parameters from models 1 and 2 respectively — we can compute $J(\theta)$ for various combinations of these parameters: $\theta = \lambda(\theta_1) + (1 - \lambda)(\theta_2)$, where λ is a mixing coefficient. Setting λ to 1 results in a parameters that are entirely θ_1 while setting λ to 0 give the parameters θ_2. By sweeping the values of λ, we can examine an linear slice of the parameter space. Two models with similar convergence point will have smooth parameter interpolations, whereas the interpolation for models at different minima will likely be non-convex and show a "bump" around $\lambda = 0.5$.

In Figure 5, we draw interpolations between the final model (sixth snapshot) and all intermediate snapshots. The left two of the plots denote Snapshot Ensembles trained with a cyclical learning rate, while the right two plots denote NoCycle Snapshot Models. $\lambda = 0$ represents a model which is

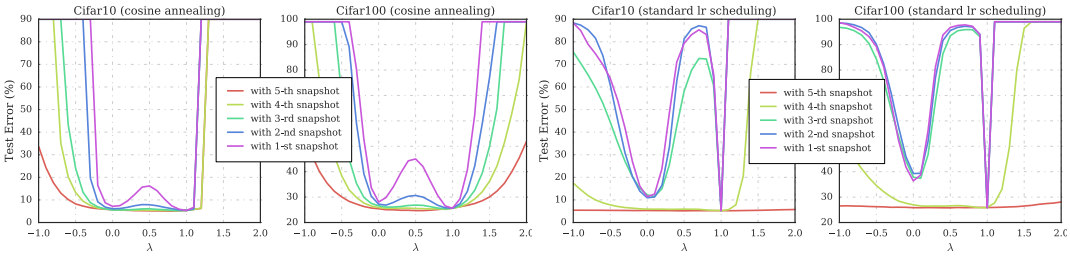

Figure 5: Interpolations in parameter space between the final model (sixth snapshot) and all intermediate snapshots. $\lambda = 0$ represents an intermediate snapshot model, while $\lambda = 1$ represents entirely final model parameters. **Left:** Snapshot Ensemble, with cosine annealing cycles ($\alpha_0 = 0.2$ every $B/M = 50$ epochs). **Right:** NoCycle Snapshot Ensemble, (two learning rate drops, snapshots every 50 epochs).

entirely snapshot parameters, while $\lambda = 1$ represents a model which is entirely the final parameters. From this figure, it is clear that there are differences between cyclical and non-cyclical learning rate schedules. Firstly, all of the cyclical snapshots achieve roughly the same error as the final cyclical model, as the error is similar for $\lambda = 0$ and $\lambda = 1$. Additionally, it appears that most snapshots do not lie in the same minimum as the final model. Thus the snapshots are likely to misclassify different samples. Conversely, the first three snapshots achieve much higher error than the final model. This can be observed by the sharp minima around $\lambda = 1$, which suggests that mixing in any amount of the snapshot parameters will worsen performance. While the final two snapshots achieve low error, the figures suggests that they lie in the same minimum as the final model, and therefore likely add limited diversity to the ensemble.

Activation space. To further explore the diversity of models, we compute the pairwise correlation of softmax outputs for every pair of snapshots. Figure 6 displays the average correlation for both for cyclical snapshots and for non-cyclical snapshots. Firstly, there are large correlations between the last 3 snapshots of the non-cyclical training schedule (right). These snapshots are taken after dropping the learning rate, suggesting that each snapshot has converged to the same minimum. Though there is more diversity amongst the earlier snapshots, these snapshots have much higher error rates and are therefore not ideal for ensembling. Conversely, there is less correlation between all cyclical snapshots (left). Because all snapshots have similar accuracy (as can be seen in Figure 5), these difference in predictions can be exploited to create effective ensembles. As expected, correlations are strongest between consecutive snapshots.

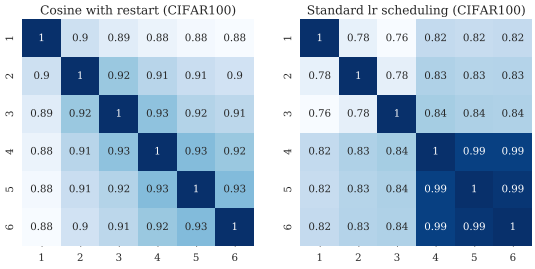

Figure 6: Pairwise correlation of softmax outputs between any two snapshots for small DenseNet. **Left:** Snapshot Ensemble, with cosine annealing cycles (restart with 0.2 every 50 epochs). **Right:** NoCycle Snapshot Ensemble, (two learning rate drops, snapshots every 50 epochs).

5 Discussion

We introduce Snapshot Ensemble, a simple method to obtain ensembles of neural networks without any additional training cost. Our method exploits the ability of SGD to converge to and escape from local minima, which allows the model to visit several accurate solutions over the course of training. We harness this power with cyclical learning rate schedule proposed by Loshchilov & Hutter (2016), saving snapshots of the models at each point of convergence. We show in several experiments that all acquired snapshots are accurate, yet produce different predictions from one another, and therefore are ideal for test-time ensembles. Ensembles of these snapshots significantly improve the state-of-the-art on the CIFAR-10, CIFAR-100 and SVHN datasets. Future work will explore combining Snapshot Ensembles with traditional ensembles. In particular, we will investigate how to balance growing an ensemble with new models (with random initializations) and refining existing models with further training cycles under a fixed training budget.

REFERENCES

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pp. 177–186. Springer, 2010.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Cristian Bucilu, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541. ACM, 2006.

Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 18. ACM, 2004.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pp. 2933–2941, 2014.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232, 2001.

Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544*, 2014.

Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12:993–1001, 1990.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *arXiv preprint arXiv:1603.05027*, 2016.

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

Gao Huang, Zhuang Liu, and Kilian Q Weinberger. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016a.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep networks with stochastic depth. *arXiv preprint arXiv:1603.09382*, 2016b.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Kenji Kawaguchi. Deep learning without poor local minima. *arXiv preprint arXiv:1605.07110*, 2016.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Anders Krogh, Jesper Vedelsby, et al. Neural network ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems*, volume 7, pp. 231–238. MORGAN KAUFMANN PUBLISHERS, 1995.

David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, Aaron Courville, et al. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.

Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*, 2016.

Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. 2015.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Mohammad Moghimi, Mohammad Saberian, Jian Yang, Li-Jia Li, Nuno Vasconcelos, and Serge Belongie. Boosted convolutional neural networks.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning, 2011. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

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

Pierre Sermanet, Soumith Chintala, and Yann LeCun. Convolutional neural networks applied to house numbers digit classification. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pp. 3288–3291. IEEE, 2012.

Saurabh Singh, Derek Hoiem, and David Forsyth. Swapout: Learning an ensemble of deep architectures. *arXiv preprint arXiv:1605.06465*, 2016.

Leslie N. Smith. No more pesky learning rate guessing games. *CoRR*, abs/1506.01186, 2016. URL `http://arxiv.org/abs/1506.01186`.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 1058–1066, 2013.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.