ON USING SIMULATED ANNEALING IN TRAINING DEEP NEURAL NETWORKS

Amirmohammad Sarfi

A thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Science (Computer Science) at
Concordia University
Montréal, Québec, Canada

April 2023

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: Amirmohammad Sarfi

Entitled: **On Using Simulated Annealing in Training Deep Neural Networks**

and submitted in partial fulfillment of the requirements for the degree of

Master of Science (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Adam Kryzak — Chair

Dr. Adam Krzyzak — Examiner

Dr. Tiberiu Popa — Examiner

Dr. Eugene Belilovsky — Co-supervisor

Dr. Sudhir P. Mudur — Supervisor

Approved by

Dr. Leila Kosseim, Graduate Program Director
Department of Computer Science and Software Engineering

20

Dr. Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

On Using Simulated Annealing in Training Deep Neural Networks

Amirmohammad Sarfi

In deep learning, overfitting is a major problem that makes it difficult for a neural network to perform well on new data. This issue is especially prevalent in low-data regimes, or when training for too many epochs. Iterative learning methods have been devised to improve the generalization performance of neural networks when trained for a prolonged duration. These techniques periodically reduce the training accuracy of a network which is called forgetting. The primary objective of the forgetting stage is to allow the network to learn more from the same data and surpass its previous performance over the long run.

In this thesis, we propose a new forgetting technique motivated by simulated annealing. Although simulated annealing is a powerful tool in optimization, its application in deep learning has been overlooked. In our study, we highlight the potential of this method in deep learning and illustrate its usefulness through experiments. Essentially, we select a subset of layers to undergo brief periods of gradient ascent, followed by gradient descent. In the first scenario, we utilize Simulated Annealing in Early Layers (SEAL) during the training process. Through extensive experiments on the Tiny-ImageNet dataset, we demonstrate that our method has a much better prediction depth, in-distribution, and transfer learning performance compared to the state-of-the-art works in iterative training. In the second scenario, we expand the application of simulated annealing beyond the realms of classification and computer vision, by employing it in text-to-3D generative methods. In this scenario, we apply simulated annealing to the entire network and illustrate its effectiveness compared to normal training. These two scenarios collectively demonstrate the potential of simulated annealing as a valuable tool for optimizing deep neural networks and emphasize the need for further exploration of this technique in the literature.

Acknowledgments

The past two years of my Master's program have been an exceptional learning journey for me. I am grateful to have this opportunity to express my sincere appreciation and thanks to all those who have contributed to this incredible journey.

First and foremost, I would like to thank my supervisors Drs. Sudhir Mudur and Eugene Belilovsky, for their unwavering support, insightful guidance, and constructive feedback throughout my studies. Their expertise, encouragement, and mentorship have been invaluable to me, and I am immensely grateful for their contribution to my academic and personal growth.

My heartfelt thanks go to Dr. Mirco Ravanelli for his valuable support, insights, and his constructive feedback. Additionally, I am deeply grateful to Dr. Pierre Poulin for his exceptional course on image synthesis and his wonderful support during my academic journey. I am also thankful to Dr. Hadis Mohseni and Dr. Mahdi Eftekhari, my supervisors and mentors during my bachelor's studies, for their incredible guidance and support.

I must also thank my close collaborators with whom I have had the privilege of working and learning from. In particular, I express my sincere gratitude to Zahra Karimpour, whose unique perspectives and critical thinking have been a source of inspiration to me. I am also grateful to Nasir Khalid and Muawiz Chaudhary for their help with my research and their valuable inputs.

I would like to extend my gratitude to Nader Asadi, Soroush Saryazdi, Ali Pourganjalikhan, Farzad Salajegheh, Mohammadmahdi Moradi, Armaghan Khoshaeen, Rucha Shende, and Bobae Jeon for their valuable insights and friendship. Their feedback has played a significant role in shaping my thoughts and approach toward research.

Finally, I want to express my gratitude to all of my close friends and family who have believed in me. I also want to thank my brother, whose unwavering belief in me has been a great source of inspiration. And last but not least, I want to extend a heartfelt thank you to my parents, who have supported and educated me throughout

my life, including their invaluable support of my decision to study abroad. Making them proud has been one of my strongest motivations during my studies.

Contents

List of Figures

List of Tables

List of Abbreviations

BANN Born Again Neural Network

CD-FSL Cross Domain Few Shot Learning Benchmark

CLIP Contrastive Language-Image pre-training

CNN convolutional neural network

DDPM Denoising Diffusion Probabilistic Model

DPM Diffusion Probabilistic Model

FSL few shot learning

G Generation

GAN Generative Adversarial Network

IMP Iterative magnitude pruning

KE Knowledge Evolution

KELS kernel-level Splitting

KNN K Nearest Neighbor

LDM Latent diffusion model

LLF Later Layer Forgetting

MCMC Markov chain Monte Carlo

NeRF Neural Radiance Fields

RIFLE Re-Initializing the Fully-connected LayEr

SA Simulated Annealing

SA-GD simulated annealing for gradient descent

SDS Score Distillation Sampling

SEAL Simulated annealing in EArly Layers

SFN-PC Saddle Free Newton with Positive Curvature

WELS weight-level Splitting

Chapter 1

Introduction

1.1 Overview

Overfitting is a common problem in machine learning where a model is trained to fit the training data too closely, which leads to poor generalization performance on unseen data. This occurs when a model becomes too complex or has too many parameters relative to the amount of training data, and it starts to capture noise or random fluctuations in the data instead of the underlying patterns. Overfitting can be thought of as a form of memorization, where the model learns the training data by heart without being able to generalize to new, unseen data. This can result in a model that performs very well on the training data but poorly on the test data or real-world applications.

Simulated Annealing is a stochastic optimization algorithm that iteratively generates new solutions by randomly perturbing the current solution, and accepts new solutions with a certain probability, even if they have worse objective function values. This probability is determined by a temperature parameter, which decreases over time according to a cooling schedule. Simulated Annealing is particularly useful for optimization problems where the search space is large and the objective function is noisy or non-convex, and escaping local optimums.

In this thesis, we incorporate simulated annealing into the training process of neural networks, to prevent overfitting and improve the generalization performance of the model. Our experiments show that simulated annealing can effectively help the model overcome overfitting, escape local and sub-optimal solutions, and eventually, produce better-performing neural network models on a variety of tasks.

1.2 Contributions and Outline

The outline of this master's thesis revolves around the utilization of simulated annealing in deep learning. The thesis includes a brief background in Chapter 2 and several sections of chapters 3 and 4 that delve into specific scenarios and methods where the use of simulated annealing has proven beneficial.

Our main contributions are:

Simulated Annealing in Early Layers Leads to Better Generalization: Recently, several iterative learning methods have been introduced to improve generalization. These typically rely on training for longer periods of time in exchange for improved generalization. LLF (later-layer-forgetting) is a state-of-the-art method in this category. It strengthens learning in early layers by periodically re-initializing the last few layers of the network. Our principal innovation in this work is to use Simulated annealing in EArly Layers (SEAL) of the network in place of re-initialization of later layers. Essentially, later layers go through the normal gradient descent process, while the early layers go through short stints of gradient ascent followed by gradient descent. Extensive experiments on the popular Tiny-ImageNet dataset benchmark and a series of transfer learning and few-shot learning tasks show that we outperform LLF by a significant margin. We further show that, compared to normal training, LLF features, although improving on the target task, degrade the transfer learning performance across all datasets we explored. In comparison, our method outperforms both LLF and normal training across the same target datasets by a large margin. We also show that the prediction depth of our method is significantly lower than that of LLF and normal training, indicating on average better prediction performance (in chapter 3). This work in this chapter has led to a conference publication:

- Amir Sarfi, Zahra Karimpour, Muawiz Chaudhary, Nasir M. Khalid, Mirco Ravanelli, Sudhir Mudur, and Eugene Belilovsky. "Simulated Annealing in Early Layers Leads to Better Generalization" In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2023 [5].

This paper is the result of the collaboration and contributions of several individuals. Amir Sarfi proposed the idea and developed the codebase. Zahra Karimpour contributed to the implementation of the codebase and provided valuable insights into the development of the idea. Muawiz Chaudhary conducted the few-shot learning experiments. Nasir Khalid extended the work to spiral data and provided

valuable insights regarding the internal representation learning of the system. I would like to thank all of these individuals for their valuable contributions to this work.

Simulated Annealing in Text to 3D Generation Chapter 3 delved into the topic of Simulated Annealing (SA) in computer vision, specifically in classification tasks that involve long and generational training. Our findings revealed that SA outperforms other iterative training methods [2–4, 6] in terms of producing more generalizable neural networks in both in-distribution and transfer learning scenarios. Cai et al. [7] recently introduced Simulated Annealing for Gradient Descent (SA-GD), where they also only focus on image classification tasks. Therefore, it remains uncertain whether simulated annealing can be applied to other tasks (in chapter 4).

In this chapter, we present a use case for SA in text-to-3D generative methods. Our study demonstrates that traditional training methods fail to optimize an existing NeRF and that a simplified version of SA provides a simple solution to the issue. This work in this chapter is currently in progress and is to be submitted to a conference.

Chapter 2

Background

2.1 Deep Learning

The reader is assumed to have a basic understanding of deep learning concepts. For more detailed information, the reader can refer to the incredible Michigan EECS 498.008/598.008 online course [1] and the Deep Learning book [8]. In this chapter, we will provide a brief overview of the background information directly related to this thesis.

2.1.1 Models

ResNet A ResNet, or Residual Network, is a type of convolutional neural network (CNN) that utilizes residual connections to help address the problem of vanishing gradients in very deep networks. The architecture of a ResNet typically consists of a series of layers, where each layer is a residual block. A residual block contains one or more convolutional layers, followed by batch normalization and a non-linear activation function, and is designed to learn the residual mapping between the input and output of the block. By adding the input to the output of each residual block through the use of a shortcut connection, the network is able to more easily propagate gradients throughout the entire network, enabling the training of deeper networks with improved performance.

U-Net A U-Net [9], is a type of CNN that is commonly used for image-to-image mapping. The architecture of a UNet is characterized by its symmetrical, "U" shape, which is formed by a series of contracting and expanding layers.

[1] https://web.eecs.umich.edu/ justincj/teaching/eecs498/WI2022/

The contracting layers, also known as the encoder, are composed of a series of convolutional layers that gradually reduce the spatial resolution of the input image while increasing the number of feature maps. The expanded layers, also known as the decoder, are composed of a series of transposed convolutional layers that gradually increase the spatial resolution of the feature maps while decreasing the number of feature maps.

The U-Net architecture allows the network to learn both local and global features of the input image by concatenating the high-resolution feature maps from the contracting layers with the corresponding low-resolution feature maps from the expanding layers. The concatenation of these feature maps helps the network to preserve the spatial information of the input image while learning more abstract features, which improves the performance of the network in tasks such as image denoising.

Clip CLIP [10] (Contrastive Language-Image Pre-training) is a pre-training method that uses contrastive learning to train a neural network to predict the relationship between images and text. CLIP is based on transformer architecture and is trained on an enormous dataset of image-text pairs.

During the pre-training process, the model is presented with an image and a text and is trained to predict if the text describes the image or not which is called the contrastive task. The contrastive task helps the model to learn the relationship between the visual and textual features and to extract meaningful representations from both modalities. CLIP models have been especially used in image generative methods conditioned on textual prompts which we will explain in further detail in section 2.4.

2.2 Simulated Annealing

Simulated Annealing (SA) [11] is a probabilistic optimization technique that is used for solving problems in combinatorial optimization. It is inspired by the annealing process of slowly cooling a material to reduce its defects and increase its structural stability.

Mathematically, SA can be defined as a Markov chain Monte Carlo (MCMC) method that generates a sequence of solutions to a given optimization problem, where each solution is generated by randomly perturbing the current solution. The probability of accepting a new solution is determined by a probability distribution

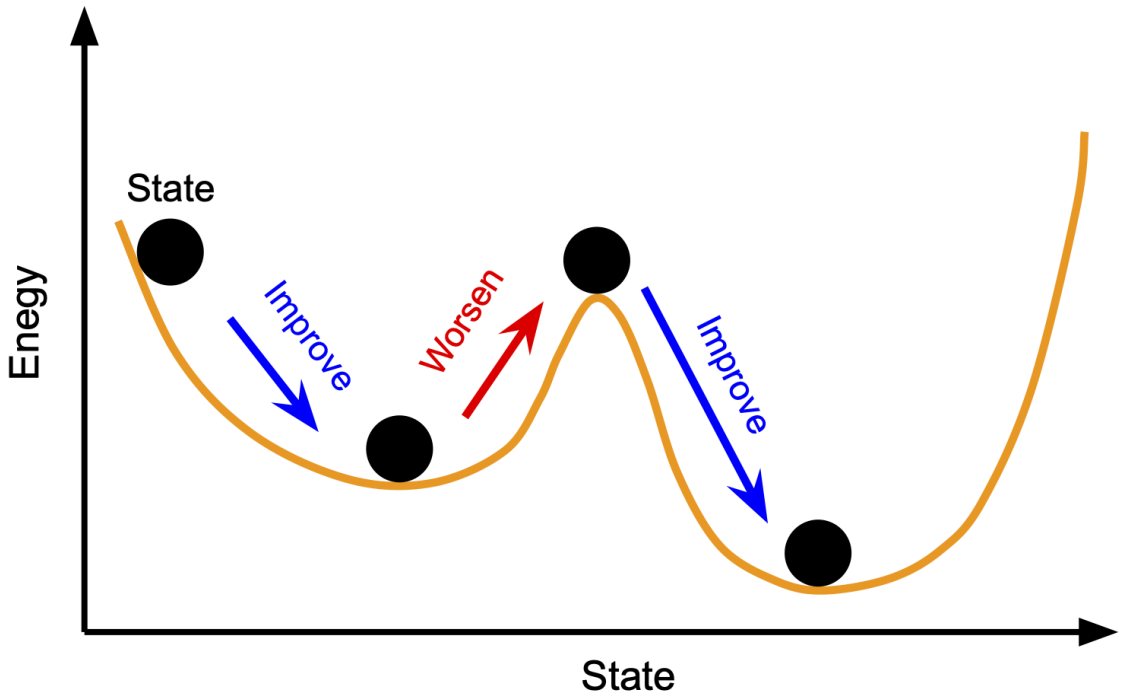

Figure 1: Starting from "state", optimizing with normal gradient descent leads to a sub-optimal solution. Simulated annealing helps with escaping the local minima and finding a better global solution.[2]

called the "acceptance function", which is typically based on the Boltzmann distribution. The acceptance function depends on a control parameter called the "temperature", which is gradually decreased over time, simulating the cooling process in physical annealing.

The goal of SA is to find the global optimum of a given optimization problem by generating a sequence of solutions that converge to the global optimum. The SA algorithm starts with a high temperature, which allows for large changes in the solutions, and gradually decreases the temperature, which makes the solutions converge to a more optimal solution. In figure 1, a typical case in which simulated annealing helps in finding a better solution is demonstrated.

We find SA to be especially useful in helping a neural network to escape a local optima or saddle point and move towards the global optima. In the context of neural networks, the loss function's value is analogous to the "temperature" in SA. During the process of optimization, we wish to minimize (cool down) this temperature.

[2]Source: `https://www.documentation.jijzept.com/docs/jijzept_docs/sa/`

2.3 Iterative Training Methods

Training neural networks for many (e.g., 2,000) epochs is a very challenging task due to the existence of sub-optimal solutions.

1. **Overfitting**: As the model is further trained on the same dataset, it starts to memorize the data, rather than learning more generalizable features. This phenomenon is referred to as overfitting which leads to poor performance on unseen data. This issue is especially prevalent when working with smaller datasets.

2. **Loss Landscape**: Another common reason that leads to a neural network to be stuck in sub-optimal solutions is the shape of the loss landscape. **Local minimas** which are points on the loss surface where the gradients of the loss function are zero, but the model's parameters are not making progress toward a better solution. This means that the model has found a solution that is good enough, but it is not the global minimum, thus the model's performance on unseen data can be suboptimal. **Plateaus** are flat regions in the loss surface where the gradients of the loss function are close to zero, making it difficult for the model to make progress. This can cause the optimization process to get stuck while the model's performance on unseen data can be suboptimal. **Saddle points** are points in the loss surface where the gradients of the loss function are close to zero in some directions but non-zero in others, making it difficult for the model to converge to a local minimum. This can cause the optimization process to oscillate between different suboptimal solutions, leading to poor performance on unseen data.

Recently, iterative training methods have been proposed to enhance the generalization ability of neural networks. These techniques enable neural networks to be trained for multiple epochs while enhancing their ability to generalize and avoid sub-optimal solutions [1–3, 6, 12].

Iterative training methods rely on the notion of a "generation", where a model is optimized to reach a local minimum within a single generation. Then, in the next generation, the objective function is altered, or the network is altered to have a higher loss, which necessitates another round of optimization to reach a minimum. Due to these modifications, it is hoped that the model will escape the sub-optimal solution, and will move towards the optimal solution in the new generation.

The initial studies on this subject concentrated on **self-distillation** [2, 3]. Self-distillation is a technique for training neural networks that is based on the idea of transferring knowledge from a pre-trained model (the teacher) to a new model with the same architecture (the student). In self-distillation, the teacher network is first trained on a dataset using traditional methods. The student network is then initialized with the same architecture as the teacher network and fine-tuned using the same dataset but with a different objective. During training, the student network is presented with the same input data as the teacher network, but instead of using the true labels, the student network is trained to predict the output of the teacher network on the same input data. This is done by using the softmax output of the teacher network as the "soft target" for the student network. The process is then repeated with the student network becoming the new teacher, and a new student network is trained on the same dataset to mimic the output of the new teacher network. This process is repeated for multiple generations. This technique has been shown to improve the generalization of neural networks by regularizing the model and preventing overfitting. Additionally, it allows the model to converge to a better solution by starting from a good initialization, avoiding poor local minima.

Subsequently, other methods have been introduced, which are referred to by Zhou et al. [6] as variations of the "forget-and-relearn" strategy. In this strategy, the authors define forgetting as any process that degrades the accuracy of a model during training. The relearn refers to the normal training behavior during each generation. In the forget-and-relearn approach, the forgetting stage typically takes place at the start of each generation. During this stage, some of the weights of the model are altered (e.g., by removing [12, 13] or re-initializing them [1, 6]). After that, the network is normally trained for the remainder of the generation.

To utilize simulated annealing in training neural networks, we periodically perform gradient ascent on a subset (or all) of the parameters of the network which decreases the training accuracy of the neural network for a few epochs. This process can be categorized under the forget-and-relearn scheme. The gradient ascent phase can be thought of as the forgetting stage and the cooling phase of simulated annealing can be considered the relearn stage. We formally explain our method from an iterative training perspective in chapter 3.

2.3.1 Self-Distillation

Knowledge Distillation: Teacher-student networks, also known as knowledge distillation [14], is a popular technique in deep learning where a small neural network is trained to mimic the behavior of a larger and more complex "teacher" network. The idea is to transfer the knowledge and expertise learned by the teacher network to the student network, with the goal of improving the student network's performance and efficiency.

The process of teacher-student learning involves two phases: training the teacher network and then using it to teach the student network. The teacher network is typically a larger and more complex model that has been trained on a given task, and it is used to generate "soft targets" or probability distributions over the output classes for each training example. The student network, on the other hand, is a smaller and simpler model that is trained to match the soft targets generated by the teacher network, rather than the actual labels. By training the student network to mimic the behavior of the teacher network, we can transfer the knowledge and representations learned by the teacher network to the student network. If the student model is much smaller than the teacher network, doing so can result in a smaller and more efficient model that performs just as well as the larger teacher network, which is referred to as model compression. In addition to improving performance and efficiency, teacher-student networks can also help to regularize the student network by enforcing consistency between the soft targets generated by the teacher network and the outputs of the student network.

Teacher-student networks have been applied successfully to a wide range of tasks, including image classification [14], object detection [15], and natural language processing [16]. They have also been used to compress and optimize large neural networks for deployment on mobile and embedded devices, and real-time applications [15–17]. For instance, Sandler et al. [17] used a teacher-student network to compress a large-scale object detection network for deployment on mobile devices, resulting in a significant reduction in model size and improved computational efficiency. Tang et al. [16] applied teacher-student networks to natural language processing tasks, demonstrating that a smaller network can be trained to achieve state-of-the-art performance by learning from the output of a larger and more complex network. Chen et al. [15] proposed a teacher-student framework for object detection to improve the efficiency and performance of the student network, making it more suitable for real-time applications.

Self-Distillation: Self-distillation involves training a single neural network to both generate predictions for a given task and to learn from its own predictions by treating them as "soft targets" during training. The idea is to encourage the network to learn from its own mistakes and refine its own predictions, which can lead to improved performance and efficiency. The difference between self-distillation and knowledge distillation is that knowledge distillation involves using a separate, pre-trained "teacher" network to generate soft targets for a smaller "student" network. The student network is then trained to mimic the behavior of the teacher network, rather than to learn from its own predictions.

Born Again Neural Networks [2]: Born Again Neural Networks (BANNs) is a self-distillation technique where the idea is to encourage neural networks to "forget" any noisy or irrelevant information that they may have learned during previous training iterations, and to focus instead on the most relevant features of the data. At each generation, it involves three main steps. First, train a neural network on a given task until it reaches a plateau in its performance. Then, randomly initialize a network with the exact same architecture (the student). Finally, train the student network to mimic the softmax output distribution of the previous model, and repeat for the next generation. Born Again Neural Networks is an effective technique for improving the performance and generalization of neural networks. Yang et al. [3] demonstrated that a more tolerant teacher, meaning that a teacher that has a higher threshold for what it considers to be correct predictions, leads to better performance. The paper argues that this approach is particularly effective for deep neural networks, which can easily become overfit to the training data and fail to generalize to new examples.

Pham et al. [4] observed that in most of the previous self-distillation methods, the teacher networks were not properly trained, with a gap between their capacity and their performance. Therefore, they raise the question that does self-distillations only work when the teacher network is far from its capacity, and hence they help with closing this gap, or are they useful even when the teacher is at near capacity? Seeking for an answer, they trained the teacher networks with state-of-the-art techniques to much higher accuracy and performed self-distillation using the properly-tuned teachers. They observe that even with a very well-performing teacher, self-distillation still improves the accuracy of the network by a considerable margin.

Born again neural networks [2, 4] iteratively perform self-distillation on the same network and are therefore considered as an iterative training method. In Chapter 3, we assess our proposed approach against self-distillation methods and exhibit its

superiority.

2.4 Diffusion Models

Diffusion models, short for Diffusion probabilistic models, originate from the framework of nonequilibrium thermodynamics in physics [18]. In this framework, the generative process of a diffusion model is viewed as a physical system that is driven away from equilibrium by a random force. The goal of the model is to learn the underlying dynamics of this system and use it to generate new samples.

The application of diffusion models in deep unsupervised learning has led to the development of new latent variable generative models that can learn more complex and realistic data distributions and can be applied to a wide range of tasks, especially, image synthesis [19–22].

Ho et al. [19] introduced Denoising Diffusion Probabilistic Models (DDPM) for image synthesis. DDPMs are a variant of Diffusion Probabilistic Models (DPMs) that are designed to handle noisy data. DPMs are based on the idea of simulating a random walk through a high-dimensional space, where each step of the walk corresponds to a change in the generated image. DDPMs introduce a denoising mechanism to the DPMs by adding a denoising step to the generative process.

The idea behind DDPMs is to learn a generative model that can remove the noise from the input data while keeping the important features of the data. This is done by the introduced denoising function that maps the noisy input data to a clean version of the data. The denoising function is trained to minimize the difference between the noisy input data and the clean output data. Formally, the generative process in DDPMs is composed of two main steps: a forward process (diffusion process) and a reverse process (denoising step).

What makes diffusion models unique is the forward process $q(x_i)$ which uses a fixed Markov chain to slowly add Gaussian noise to the data based on a schedule of variances $(\beta_1, \beta_2, ..., \beta_T)$. For simplicity, DDPM sets all variances to a constant value although they can be learned by reparameterization. Therefore, the forward process of DDPM does not involve any learnable parameters.

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}), \qquad q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I}) \tag{1}$$

An important property of the forward process is that sampling x_t at any timestep t can be done using a closed-form solution without the need of computing above the

multiplication every time. We refer the reader to the DDPM paper for further details.

By gradually adding Gaussian noise to the original data x_0, we eventually end up with a normal distribution $p(x_T) = \mathcal{N}(x_T; 0, \mathbf{I})$ at time step T. The intuition is that, if we can train a neural network that can gradually remove noise from the generated noisy data, then we can start from any randomly generated normal distribution and generate new samples. The reverse process is designed to do exactly that. More specifically, in the reverse process $p_\theta(x_i)$, we wish to gradually recover the original data from the noisy images. We do so by performing multiple small denoising steps $p_\theta(x_{t-1}|x_t)$, instead of trying to directly recover the original data. The reverse process can also be formally defined as a Markov chain with learned Gaussian transitions initiating from a random normal distribution $p(x_T)$:

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t), \qquad p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{2}$$

The primary task of DDPM is to learn the reverse process. In particular, we need to model $\mu_\theta(x_t, t)$ and $\Sigma_\theta(x_t, t)$, as they are the only learnable parameters in $p_\theta(x_{t-1}|x_t)$. First, DDPM does not learn the covariance matrix $\Sigma_\theta(x_t, t)$ and instead sets it to $\Sigma_\theta(x_t, t) = \sigma_t^2 \mathbf{I}$ where σ_t^2 is a function of $\beta_{0:t}$, and hence it is a time dependant parameter.

To represent the mean $\mu_\theta(x_t, t)$, DDPM models the forward process posterior means through a reparameterization. Specifically, they optimize a neural network $f_\phi(x)$ with the following objective:

$$L = ||\epsilon_t - f_\phi(x_t)||^2 \tag{3}$$

Please note that the above formulation is a simplified version of the actual objective. We refer the reader to the DDPM paper for further detail and the complete formulation. DDPM uses a U-Net for $f_\phi(x_t)$ and operates in the pixel space. Diffusion models that operate in the pixel space are known for their high computational demands and complexity. Also, at inference time, generating new data from them is quite expensive [21].

Latent diffusion models (LDMs) [21] were proposed to remedy the computational complexity of diffusion models. A latent diffusion model works by performing the same diffusion process in the latent space of strong auto-encoders. This means that instead of working on the images x, LDM works on the encoded representation of the image $z = E(x)$. Therefore, in the forward process, LDM adds Gaussian noise to the

latent representation of the image z, and in the reverse process, LDM learns to remove noise from the noisy latent representations. Finally, to synthesize images, LDM simply starts from a random normal distribution with the shape of the latent representations $p(z_T) = \mathcal{N}(z_T; 0, \mathbf{I})$, and performs the same process as DPMs to generate denoised latent representations. Finally, they decode the latent representations to generate the images using the auto-encoder's decoder.

In addition to unconditional or class-conditional generation, diffusion models have demonstrated very strong conditional generation as well. Specifically, Rombach et al. [21] have developed a conditional method based on cross-attention that enables multi-modal training. Using this multi-modal training, they have trained the LDM that can perform text-to-image generation. They condition the LDM using clip embeddings of the textual prompts. We use diffusion models in chapter 4 as one of the main building block of text-to-3D generation.

Chapter 3

Simulated Annealing in Early Layers Leads to Better Generalization

3.1 Introduction

Overfitting is a crucial challenge in supervised deep learning, which prevents a neural network from performing well on unseen data. This problem is particularly pervasive in smaller dataset regimes. Classical machine learning approaches to address this problem typically rely on explicit regularization added as part of the optimization objective [23]. A number of implicit approaches are often applied in training deep networks. These implicit methods are usually easier to design than explicit terms added to the objective function. For example, early stopping [24] and dropout [25] are classical examples of implicit regularization methods. It can be shown that these techniques can be shown to be linked directly to explicit regularization terms. Consider the case of linear regression, early stopping can be directly seen as Tikhonov regularization [24]. Indeed the use of modified optimization strategies to improve generalization is becoming pervasive in deep learning [26].

Recently, *iterative training* methods have been introduced to improve generalization in deep networks. These allow neural networks to be trained for many epochs while progressively improving generalization [1–3,6,12]. These works typically rely on a notion of a *generation*, in which a model is optimized towards a local minimum in a single generation. Subsequently, in a new generation, the objective function is modified, or the network is perturbed towards a high loss, requiring

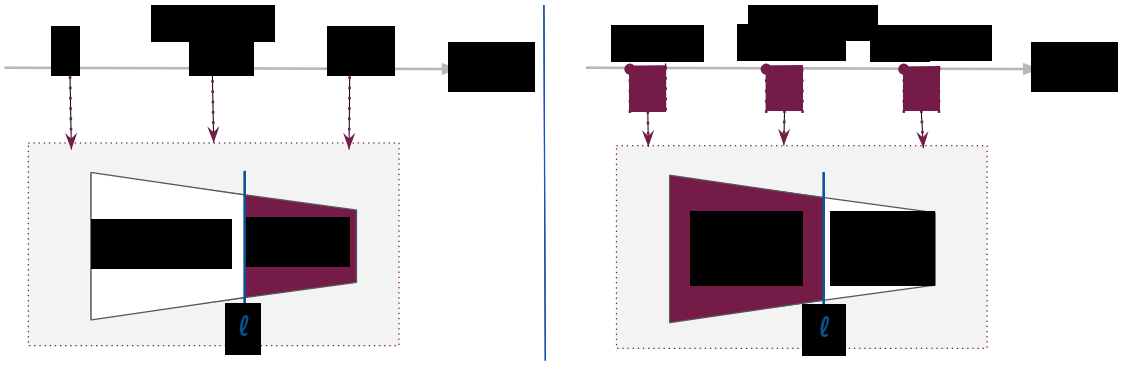

Figure 2: Our iterative training method (SEAL) compared to LLF. The model is shown with the input on the right and output on the left. E denotes the epochs in a generation. LLF re-initializes the top layers right before a new generation begin. In our approach, we do not re-initialize but perform gradient ascent in the first k epochs of a generation, only on the early layers of the network.

a new generation of optimization towards a minimum. The earliest work on this topic focused on self-distillation [2,3], where, in each successive generation, a student network with the same architecture was initialized and trained to mimic the softmax output distribution of the previous model. This procedure was theoretically studied by [27] and provided the formal link to its explicit regularization effects.

A number of other techniques were subsequently proposed. These have been characterized by Zhou et al. [6] as instances of a forget-and-relearn scheme, where they define forgetting as any process that worsens the training accuracy of a model. In forget-and-relearn, the forgetting stage happens at the beginning of each generation where some part of the weights are perturbed (e.g., by removing [12, 13] or by re-initialization [1,6]), and the network is normally trained for the rest of the generation. They further introduce a simple forgetting method called later-layer-forgetting (LLF), where they re-initialize the last few layers of the network. The intuition behind LLF comes from the concept of prediction depth introduced by Baldock et al. [28]. The prediction depth of a sample refers to the earliest layer, after which the layer-wise k-nearest neighbor (KNN) probe of all layers is the same as the model's prediction. Zhou et al. [6] empirically show that LLF improves the prediction depth.

We hypothesize that by constantly re-initializing the later layers, LLF pushes high-level information to the early layers, which explains its stronger generalization as compared to normal training. However, when it comes to transfer learning, Zhao et al. [29] have stated that it is mainly the low- and mid-level representations that are transferred. Following this insight, we analyzed features learned by LLF in a transfer learning setting. We observed that LLF performs weaker than normal

training, which affirms our hypothesis. This is discussed further in the Experimental Results section 3.4.2.

Simulated annealing is a method for solving unconstrained and bound-constrained optimization problems. It models the physical process of heating a material and then slowly lowering the temperature to decrease defects, thus minimizing the system energy. During training, gradient ascent mimics increase in defects (temperature increase), while gradient descent emulates the cooling process to reach a better minimum [11]. Cai [7] adapted the classical simulated annealing for gradient descent (SA-GD) to improve the optimization by escaping local minimums and saddle points. In SA-GD, this is done by probabilistically doing gradient ascent during training. Their method, however, is not used for iterative learning and unlike our work is not adapted to take advantage of the specific architecture biases of deep networks. Based on the hypothesis that LLF primarily benefits from pushing high-level information into the early layers (and thereby the prediction depth), in this work, we adapt the SA idea as an alternative to LLF, and we do not reset the later layers. Instead, we perform an intermittent gradient ascent procedure just on the early layers, coaxing them to find better solutions through multiple generations of training.

Our main contribution is a new iterative training method that performs gradient ascent on the initial layers of the network, for a few epochs, at the beginning of every generation to induce forgetting. By performing the ascent on only a subset of layers, we ensure that the model retains information from the previous generations, rendering it suitable for long training. The intuition behind gradient ascent on the early layers is that it prohibits them from being overly specific to the task, avoiding the encoding of high-level semantics, unlike what happens in LLF. We have carried out extensive experiments on the Tiny-ImageNet [30] dataset and transfer learning to Flower [31], CUB [32], Aircraft [33], Dogs [34], and MIT [35] datasets, and Cross-Domain Few-Shot Learning (CD-FSL) [36]. Our experiments show that our method outperforms LLF and normal training in both in-distribution and transfer learning settings (Tables 1, 2, 6). Our method also provides better prediction depth (Fig. 3). By analyzing the statistics of Hessian eigenvalues, we observe that our method has a lower max eigenvalue and no negative eigenvalues, suggesting SEAL reaches a flatter local minima and avoids saddle points (Table 10). The complete code for our experiments is provided in the supplementary materials.

3.2 Background and Related Work

Iterative learning Consider a neural network f parameterized by $\mathbf{\Theta}$. Following [1] and [6], let us define a mask M that splits a neural network's weights into fit hypothesis $\mathbf{H}_{fit}$ and forgetting hypothesis $\mathbf{H}_{forget}$ as:

$$\mathbf{H}_{fit} = M \odot \mathbf{\Theta} \quad \text{and} \quad \mathbf{H}_{forget} = (1 - M) \odot \mathbf{\Theta} \tag{4}$$

Forget-and-relearn iterative training methods perturb the weights in the forgetting hypothesis $\mathbf{H}_{forget}$ at the beginning of each generation and retrain the neural net for E epochs. This process is repeated for G generations; hence it is called iterative training. We now formally cast the previous iterative training methods into this framework.

Knowledge evolution in neural networks (KE) [1]: In this work, two different masking strategies were proposed. One strategy splits the network's weights based on their location in the convolutional kernels, namely kernel-level (KELS) splitting. The other strategy splits the weights randomly, namely weight-level splitting (WELS). At the beginning of every generation, the weights in the forgetting hypothesis $\mathbf{H}_{forget}$ are randomly re-initialized, and the fit hypothesis $\mathbf{H}_{fit}$ is kept the same. Then, the network is trained (for E epochs of the next generation) from this new initialization. The masking framework in this work is determined prior to the training, meaning that it does not depend on the state of the weights by the end of each generation. Furthermore, the masks are created given a split rate of weights s_r in each layer. Taha et al. [1] hypothesize that this method (KE) has a resemblance to dropout in that it encourages neurons to become more independent. However, Zhou et al [6] believe that it is difficult to apply the intuition behind dropout to a single subnetwork. Therefore, they propose another possible explanation regarding why KE works (by comparing with ResNet): KE can create a zero mapping of the reset hypothesis without decreasing capacity. However, this zero-mapping only happens after adequate normal training, and it is not clear how it would enhance generalization.

Iterative magnitude pruning (IMP) [12,13]: In this work, the weights are split into forgetting and fits hypotheses based on their state by the end of each generation (unlike KE where the masking strategy was fixed before training). In particular, they focus on the magnitude of the weights at the end of each generation. The idea is that, if a weight has a relatively lower magnitude, it was probably not contributing much to the final decision-making of the system, and hence can be removed. Therefore, in this

work, the forgetting hypothesis $\mathbf{H}_{forget}$ consists of weights that had low magnitude, and the rest of the weights are considered in the fit hypothesis $\mathbf{H}_{fit}$. At the beginning of each generation, the weights in the forgetting hypothesis $\mathbf{H}_{forget}$ are removed from the network (by setting them to zero and freezing them), and the fit hypothesis $\mathbf{H}_{fit}$ weights are rewound to their initial values. Thus, in each generation, a percentage of the weights are removed. The authors discovered that in many neural networks, there is a small subset of neurons that are critical to the network's performance. They refer to the resulting lightweight network as "lottery ticket".

The lottery ticket hypothesis in deep learning is the idea that there exists a sparse neural network within a larger, dense neural network that can be trained to perform as well as the original network, but with significantly fewer parameters. The term "lottery ticket" refers to this sparse neural network, which is like winning a lottery because it can provide significant computational savings without compromising performance. The implications of the lottery ticket hypothesis are significant, as it suggests that neural networks can be compressed without sacrificing performance, leading to faster and more efficient training and deployment. This has important applications in areas such as mobile and edge computing, where computational resources are limited.

Deconstructing Lottery Tickets [37]: Zhou et al. [37] studied the critical components of the Lottery Ticket hypothesis, trying to explain why the resulting lottery tickets (lightweight versions of the networks) usually outperform the original higher capacity network. They further define new masking strategies with a very unique attribute; at the beginning of each generation and immediately after the re-initialization phase stage (after pruning the low-magnitude weights and rewinding the rest of the network using their super masks), they show that the resulting lightweight network performs much stronger than a randomly initialized network, even though it has less capacity. Considering this observation, [6] et al. categorizes IMP under the forget-and-relearn strategy with an additional sparsity constraint. By rewinding and pruning the weights, IMP decreases the accuracy at the beginning of each generation (forgetting), and since the resulting model performs much better than chance, it retains information from the past generations and hence, relearns during the new generation.

Re-Initializing the Fully-connected LayEr (RIFLE) [38]: Li et al. [38] proposes that periodically resetting the final layer during fine-tuning (in transfer

learning) can help to prevent the network from getting stuck in a local minimum and enable more substantial updates to earlier layers. This is because the final layer is closely connected to the earlier layers, and by resetting it, the earlier layers can adapt to the new task more effectively. This technique can facilitate more meaningful updates to the low-level features of the earlier layers, leading to improved performance. In other words, resetting the final layer can be thought of as a way of forgetting the previous training, allowing for new learning to take place. They demonstrate that RIFLE leads to much stronger transfer learning performance compared to normal training using this simple iterative training. To be specific, the forgetting hypothesis $\mathbf{H}_{forget}$ consists of weights in the final fully-connected layer, and the rest of the weights are considered as the fit hypothesis $\mathbf{H}_{fit}$.

Later Layer Forgetting (LLF) [6]: Zhou et al. [6] modifies the KE forget-and-relearn strategy (in a very similar way to RIFLE [38]) to improve the prediction depth of the network [28]. LLF considers a layer threshold L, and the weights in all layers before L are put into the forgetting hypothesis $\mathbf{H}_{forget}$, and the rest of the network (the last half of the network) is considered the fit hypothesis $\mathbf{H}_{fit}$. Then, similar to KE and RIFLE, they re-initialize the forgetting hypothesis $\mathbf{H}_{forget}$ and do not modify the fit hypothesis $\mathbf{H}_{fit}$. They demonstrate that doing so enhances the prediction depth of the network and leads to better in-distribution generalization.

Simulated Annealing: Cai [7] proposes a novel optimization algorithm by adapting the classical simulated annealing for gradient descent (SA-GD) to improve the optimization by escaping local minimums and saddle points. The authors begin by explaining that while gradient descent is a widely used optimization method in machine learning, it can get stuck in local minima and suffer from slow convergence. To address these issues, the authors introduce simulated annealing, a well-known optimization method that is inspired by the annealing process in metallurgy. Simulated annealing allows the algorithm to explore a wider range of solutions by randomly accepting worse solutions, especially at the earlier stages of training.

The SA-GD algorithm works by applying gradient descent with simulated annealing during each training epoch. Specifically, at each epoch, the algorithm computes the gradient of the loss function with respect to the parameters and updates them using a learning rate. However, before accepting the updated parameters, the algorithm performs a simulated annealing step that randomly perturbs the

parameters and accepts the new solution with a probability that depends on the temperature and the difference in the loss function. The authors show that SA-GD is effective in improving the convergence speed and robustness of traditional gradient descent methods on a variety of machine learning tasks, including image classification, regression, and natural language processing. Furthermore, they demonstrate that SA-GD can generalize better than other optimization methods, achieving higher accuracy on test data while using fewer training epochs. However, the performance gains are not significant and they introduce many hyper-parameters compared to normal training.

Dauphing et al. [39] make the argument that many difficulties in optimization arise from saddle points and not local minima. The saddle points occur when the gradient of a function is zero but the Hessian matrix (which describes the curvature of the function) has both positive and negative eigenvalues. Saddle points can be problematic for optimization algorithms because they are not strict local minima or maxima, and the gradient direction does not point toward the optimal solution.

Jin et al. [40] develop a perturbed stochastic gradient descent procedure to deal with saddle points. In particular, the authors propose a new optimization algorithm called "Saddle Free Newton with Positive Curvature" (SFN-PC), which is based on the idea that saddle points can be escaped more efficiently if the algorithm searches for directions with positive curvature. They compute the Hessian matrix at each iteration and then project it onto a subspace that is spanned by eigenvectors with positive curvature. This subspace is then used to compute a new search direction that is guaranteed to have positive curvature, allowing the algorithm to escape saddle points more efficiently.

Few Shot Learning Traditionally, machine learning models require large domain-specific labeled datasets in order to correctly classify [41–43]. The goal of few shot learning (FSL) [44] is the following: given a limited number of labeled data, learn a model that rapidly generalizes to new, novel classes. Few Short Learning has been considered recently in the literature, from the point of view of meta learning and transfer approaches [45], [46], [47], [44], [48]. Typical FSL tasks are based on having an initial large dataset from which a model is either pre-trained (transfer based approaches) or meta-trained (meta-learning based approaches).

Early FSL works have focused on using natural image datasets, with another natural image dataset provided for base model training (via meta-learning or pre-training) [44, 48]. On the other hand, many cases of more general interest require

learning target datasets that are not natural image datasets, and moreover are distant in terms of the domain. The recently proposed Cross-Domain Few-Shot Learning (CD-FSL) [36] benchmark is aimed at providing an investigative setting for such large domain shifts from source to target training data. Four datasets of decreasing similarity to natural images are included, [49], [50], [51], [52], [53]. These include images of plant disease, aerial photos, and medical data not resembling natural images at all (skin lesions, Chest X-rays). In each dataset, we wish to predict some novel classes, whether it be rare skin diseases, airplanes, or crop diseases. A number of proposals for this challenging scenario have been suggested. [54] proposed a self-supervised learning objective coupled with a teacher-student method. They also observed that traditional meta learning techniques fail on this task. [55] demonstrated that efficient manipulation of the batch norm layer during training of models can lead to improved performance under such extreme domain shifts. The results of using SEAL for FSL are provided in a later section.

3.3 Proposed Method

In this work, we split the network into fit $\mathbf{H}_{fit}$ and forgetting hypotheses $\mathbf{H}_{forget}$, using a layer threshold, say, L. All weights prior to L are considered in the forgetting hypothesis $\mathbf{H}_{forget}$, and weights in layers $> L$ as the fit hypothesis $\mathbf{H}_{fit}$. This is the opposite of LLF [6], since their fit and forgetting hypotheses are swapped.

To induce forgetting, we perform gradient ascent on the forgetting hypothesis $\mathbf{H}_{forget}$ for k epochs. During the gradient ascent phase of the forgetting hypothesis $\mathbf{H}_{forget}$, we train the fit hypothesis $\mathbf{H}_{fit}$ normally (with gradient descent). This can be categorized under the high-level definition of forgetting that Zhou et al. [6] provide, in that it drops the training accuracy of the network to completely random. This is inspired by the simulated annealing algorithm to enhance the optimization of the network and to introduce a more definitive forgetting mechanism. Our method is different from the prior methods as they either remove [12, 13] or re-initialize [6] [1] the forgetting hypothesis $\mathbf{H}_{forget}$ (at once) before the first epoch of a new generation. We set k to $\frac{1}{4}$ of total epochs E in the generation.

We adjusted the sign of the weight decay for the layers that perform gradient ascent; so as to avoid the weights being encouraged to have a higher norm, and causing the network to diverge. However, we found that even this is not enough to stop the divergence. We noted that using the same learning rate for the ascending phase was the reason for this. Hence we toned down the ascent learning rate using a

Method	Tiny-ImageNet	Flower	CUB	Aircraft	MIT	Dogs
Normal	54.37	34.31	6.49	6.24	25.67	8.99
Normal (long)	49.27	26.96	8.07	6.30	24.85	11.53
LLF	56.92	22.84	5.33	4.65	23.8	8.69
SEAL (Ours)	**59.22**	**45.68**	**8.49**	**9.81**	**35.37**	**12.61**

Table 1: Transferring tiny-imagenet learned features to other datasets using linear probe. Normal, and Normal (long) refer to $G = 1$ and $G = 10$ generations of training, respectively. LLF and SEAL were trained for $G = 10$ generations. Transfer accuracy of LLF after $1,600$ epochs is significantly lower than normal training with both 160 and $1,600$ epochs; our method after $1,600$ epochs surpasses normal training by a large margin. This demonstrates that our method learns much more generalizable features compared to Normal training and LLF.

factor S:

$$\eta_{\mathrm{a}} = S \times \eta \tag{5}$$

Where η is the learning rate, and η_{a} is the ascent learning rate. We empirically fix $S = 0.01$. We summarize the whole process as follows:

$$\begin{aligned} \boldsymbol{\Theta}^{e,t+1}_{forget} &= \begin{cases} \boldsymbol{\Theta}^{e,t}_{forget} + \eta_{\mathrm{a}} \nabla J(\boldsymbol{\Theta}^{e,t}_{forget}), & \text{if } \; e \,\%\, E < k \\ \boldsymbol{\Theta}^{e,t}_{forget} - \eta \nabla J(\boldsymbol{\Theta}^{e,t}_{forget}), & \text{otherwise} \end{cases}, \\ \boldsymbol{\Theta}^{e,t+1}_{fit} &= \boldsymbol{\Theta}^{e,t}_{fit} - \eta \nabla J(\boldsymbol{\Theta}^{e,t}_{fit}) \end{aligned} \tag{6}$$

Where $J(\boldsymbol{\Theta})$ is the objective function, $\boldsymbol{\Theta}^{e,t}_{forget}$ and $\boldsymbol{\Theta}^{e,t}_{fit}$ refer to parameters in the forgetting and fit hypotheses, respectively, and e, t refers to the iteration t during epoch e. Again, E refers to epochs in each generation and is fixed for all generations. Figure 2 illustrates this for LLF and our proposed method (SEAL).

3.4 Experimental Results

3.4.1 Implementation Details

We use Tiny-ImageNet [30] to train models and then evaluate on both the Tiny-ImageNet test set and a wide set of downstream transfer learning tasks, including popular few shot learning benchmarks. Following LLF, we use ResNet50 and train using SGD optimizer with a momentum of 0.9 and weight decay of 5e$-$4. We train

$G = 10$ generations for $E = 160$ epochs using a batch size of 32. As in LLF, we use cross entropy loss with label-smoothing [56, 57] with $\alpha = 0.1$. We also use cosine learning rate decay [58] with an initial learning rate of 0.01. For data augmentations, we perform horizontal flip and random crop with a padding of 4. We use layer threshold $L = 23$ for both LLF and our method (the third block in ResNet50). This means that in LLF, the first two blocks are considered the fit hypothesis $\mathbf{H}_{fit}$, while the fit hypothesis in our method is the last two blocks. In all experiments, normal training refers to $G \times E$ epochs of training with the same optimization settings.

For our few shot learning evaluation, we evaluated our models in an episodic fashion. Each episode has a train set of 5 classes with K examples each (5-way K-shot) and a test set with 15 examples for each class. These sets are sampled from a target dataset. Models are fine tuned on the train set and then used to produce predictions on the test set. The accuracies are reported over 600 episodes. Cross Domain Few Shot Learning Benchmark (CD-FSL) [36] was selected as the dataset for the task, which includes data from four different data sets, namely CropDiseases [49], EuroSAT [50], ISIC2018 [51, 52], and ChestX [53].

Overall we compare our method with the following three approaches:

Normal denotes the standard training with 160 epochs (corresponding to $G = 1$ generations under the conventions of iterative training).

Normal (long) refers to training the model with the standard settings for $1,600$ epochs (corresponding to $G = 10$ generations without any forgetting).

LLF refers to fortuitous forgetting [6] where at the beginning of each generation the later layers of the network are re-initialized.

SEAL refers to our proposal which performs a gradient ascent and subsequent descent phase during a generation.

Generation	**Normal**	**LLF**	**Ours**
Gen=1	54.37	-	-
Gen=3	51.16	56.12	**58.25**
Gen=10	49.27	56.92	**59.22**

Table 2: Comparison of our method with normal training and LLF on Tiny-ImageNet. Please note that the behavior of the first generation for all methods is the same. We significantly outperform standard long training and LLF.

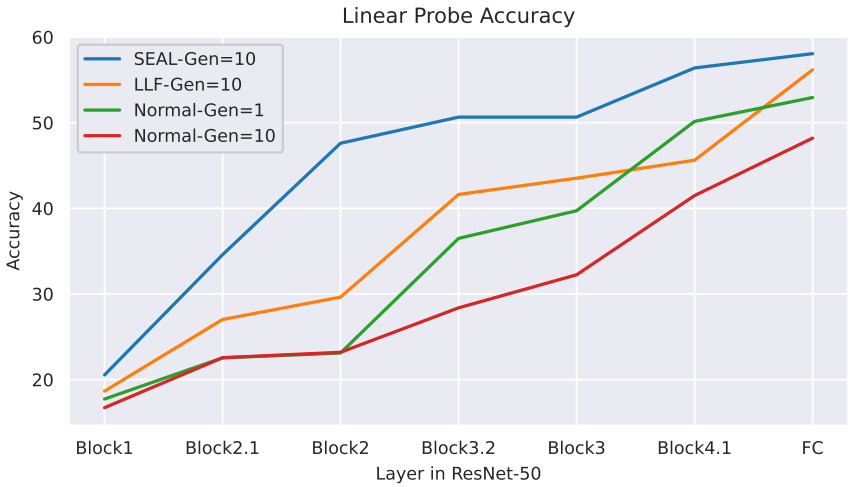

Figure 3: Comparison of layer-wise prediction depth. SEAL gives comparably much stronger predictions early on in the network. Note that block.X.Y denotes the output activations of intermediate layer Y in residual block X. This indicates that our method encourages learning the difficult examples using conceptually simpler and more general features of the early layers. This leads to better overall performance as we progress deeper into the network.

Datasets Tiny-ImageNet consists of 200 classes and has $100,000$ training and $10,000$ validation images selected from the ImageNet dataset. These images are downsized to 64x64 colored images. For our transfer learning evaluations we use the natural image datasets Flower, CUB, Aircraft, MIT, and Stanford Dogs, the statistics of these datasets are provided in Table 3.

	num_classes	**Train**	**Valid**	**Test**	**Total**
Flower [31]	102	1,020	1,020	6,149	8,189
CUB [32]	200	5,994	N/A	5,794	11,788
Aircraft [33]	100	3,334	3333	3,333	10,000
MIT [35]	67	5,360	N/A	1,340	6,700
Dogs [34]	120	12,000	N/A	8,580	20,580
CIFAR-100 [59]	100	50,000	N/A	10,000	60,000

Table 3: Summary of the datasets used in tables 2 and 1, adopted from Taha et al. [1].

For the few-shot learning, we utilized the 4 datasets from the CD-FSL benchmark, which include ChestX, ISIC, CropDisease, and EuroSAT. From these, we sample training and evaluation sets with 5 classes and a varying number of samples per class.

3.4.2 Evaluation

We now present our primary evaluations of SEAL for both improved generalization, transfer, and few-shot transfer learning.

In-Distribution Generalization: Table 2 compares the in-distribution accuracy of our method with the state-of-ther-art iterative training method, namely, LLF. We note that LLF outperforms Normal training (as reported in [6]). Our method significantly outperforms both LLF and normal training in this setting. We conduct our experiment using the same hyper-parameters used in LLF to ensure fairness.

Furthermore, we compare our method with both classical self-distillation methods such as classical born-again neural networks [2, 3], and a state-of-the-art self-distillation method [4] in an in-distribution setting.

Yang et al. [3] conducted experiments on CIFAR-100 [59] using a 110-layers ResNet. CIFAR-100 consists of $60,000$ RGB images of 32×32 resolution, with $50,000$ training examples and $10,000$ test samples. They choose this dataset over CIFAR-10 since CIFAR-10 does not contain fine-level classes, and thus self-distillation methods do not bring significant gains. For training, they use SGD with a weight decay of 1e$-$4, and a momentum of 0.9, using batch sizes of 128. Each generation is trained for $E = 164$ epochs with a base learning rate of 0.1 which is then divided by 10 after 150 and 225 epochs. They also use standard data augmentation techniques during training such as padding of four pixels, and random cropping, and horizontal flipping with a probability of 50% (without any augmentation during the test phase). Using this exact setting, we compare our method with their method in table 4. We observe that compared to Furlanello et al. [2], our method outperforms them consistently by the end of each generation. Furthermore, we compare our best model after 10 generations with that of [3] (each generation is trained for the same number of epochs in both works). Our method demonstrates 1.71% higher accuracy compared to [3] under this setting.

Pham et al. [4] observed that the hyper-parameter settings used in previous self-distillation methods are not properly tuned, and hence there is a gap between their teachers' performance and their capacity. Therefore, they question whether self-distillation methods only work when the teacher is far from their capacity, or do they always enhance the teacher, even when the teacher is properly trained. To this end, they trained a teacher with well-tuned hyperparameters using ResNet-18 on the CIFAR-100 dataset. Precisely, they use SGD with a weight decay of 3e$-$4, and a Nestrov momentum of 0.9, using batch sizes of 96 with standard data

Generation	Furlanello et al. [10]	Yang et al. [41]	Ours
Gen=0	71.55	–	–
Gen=1	71.41	–	**72.83**
Gen=2	72.30	–	**73.53**
Gen=3	72.26	–	**73.88**
Gen=4	72.52	–	**74.18**
Gen=10 (Best)	72.61	73.72	**75.43**

Table 4: Comparison with [2, 3] on CIFAR100 using ResNet-110 with inferior hyper-parameter settings. The last row lists the best accuracy of each method throughout 10 generations. The hyperparameters and baseline accuracies are adopted (and not changed) from [3] to ensure fairness.

augmentation techniques. Using the same hyperparameter setting, SEAL outperforms self-distillation techniques by a 1.18% margin (refer to table 5.

Therefore, our method outperforms self-distillation methods in both an inferior hyper-parameter setting [2, 3] and a well-tuned one [4].

Generation	Pham et al. [4]	SEAL (Ours)
Gen=0 (Teacher)	76.30	76.15
Gen=Last (Student)	77.32	**78.50**

Table 5: Comparison with Pham et al. [4] on CIFAR100 using ResNet-18 with a well-tuned hyper-parameter setting. We train our method for the same number of epochs and under the same hyper-parameter setting as [4] to ensure fairness.

Transfer Learning We now turn to evaluate the transfer learning properties of our method and that of LLF. We begin by studying the transfer learning from the Tiny-ImageNet pretrained models to 5 different image datasets. Specifically, CUB-200-2011 (CUB) [32] contains images of 200 wild bird species, Flower102 (Flower) [31] contains images from 102 flower categories, FGVC-Aircraft (Aircraft) [33] consists of 100 aircraft model variants, and Stanford Dogs dataset contains images of 120 breeds of dogs for fine-grained classification. MIT Indoor 67 (MIT) [35] is an indoor scene recognition dataset that includes 67 different scene classes.

We train linear models on top of pretrained models from each method to measure their transfer learning properties. Specifically, we re-initialize the last linear layer of the network and freeze the rest. Then, we train the linear head using the train set of

Updates	Base Model	ChestX	ISIC	EuroSAT	CropDisease
		5-way, 1-shot			
Linear	Normal	21.01 ± 0.35	25.7 ± 0.47	53.41 ± 0.92	50.31 ± 1.03
	Normal (long)	20.40 ± 0.23	22.59 ± 0.35	36.60 ± 0.75	27.50 ± 0.69
	LLF	20.38 ± 0.24	24.61 ± 0.46	36.58 ± 0.86	26.28 ± 0.68
	SEAL (ours)	**21.67 ± 0.36**	**27.93 ± 0.55**	**57.75 ± 0.88**	**63.64 ± 0.96**
LA	Normal	21.14 ± 0.35	26.71 ± 0.56	47.36 ± 1.26	64.75 ± 1.13
	Normal (long)	20.90 ± 0.37	26.41 ± 0.54	45.51 ± 1.25	63.73 ± 1.05
	LLF	20.24 ± 0.24	23.28 ± 0.45	34.02 ± 1.35	35.12 ± 1.61
	SEAL (ours)	**21.3 ± 0.39**	**29.14 ± 0.57**	**55.68 ± 1.05**	**67.87 ± 0.54**
		5-way, 5-shot			
Linear	Normal	22.79 ± 0.36	33.28 ± 0.49	71.74 ± 0.75	77.05 ± 0.80
	Normal(long)	21.00 ± 0.28	28.93 ± 0.48	53.40 ± 0.77	57.10 ± 1.07
	LLF	22.03 ± 0.33	29.60 ± 0.48	60.69 ± 0.87	53.55 ± 1.12
	SEAL (ours)	**24.42 ± 0.42**	**37.58 ± 0.54**	**74.61 ± 0.65**	**85.00 ± 0.61**
LA	Normal	20.82 ± 0.26	28.57 ± 0.78	53.35 ± 1.74	63.22 ± 2.35
	Normal(long)	21.59 ± 0.34	29.67 ± 0.82	55.79 ± 1.67	71.55 ± 2.00
	LLF	20.44 ± 0.19	22.01 ± 0.44	30.83 ± 1.45	28.06 ± 1.47
	SEAL (ours)	**22.98 ± 0.36**	**39.64 ± 0.79**	**73.83 ± 0.93**	**88.24 ± 0.54**

Table 6: Few-shot transfer results for the CFSDL benchmark (extreme distribution shift) for 1 and 5 shots. All methods make use of a ResNet50 backbone trained on Tiny-ImageNet evaluated over 600 episodes. We consider finetuning both the linear layer (Linear) and the linear layer and affine parameters (LA), the best performer in both categories highlighted in red. We observe that SEAL outperforms standard training even for a very long number of epochs, while LLF under-performs.

the target datasets and evaluate on their test sets. For training on the target dataset, we again use SGD with a momentum of 0.9, weight decay of 1e−4, and flat learning rates of [1e−1, 1e−2, 1e−3] for 120 epochs and report the highest accuracy.

Table 1 demonstrates the transfer accuracy of LLF, normal training, and our method (SEAL). Even though LLF outperforms normal training with a 2.55% margin in the in-distribution setting, we observe that in transfer learning, LLF's performance is substantially lower than normal training for 1/10 of epochs ($G = 1$), as well as normal training for the same number of epochs ($G = 10$). On the other hand, our method not only dominates in the in-distribution setting, but it also has a much stronger transfer learning performance than both LLF and normal training across all of the target datasets in our experiments.

Few-Shot Transfer Learning We now consider evaluating our approach for a challenging distant transfer learning task studied in [54, 55]. Here we are presented

Updates	Base Model	ChestX	ISIC	EuroSAT	CropDisease
		5-way, 20-shot			
Linear	Normal	25.16 ± 0.37	41.24 ± 0.50	79.14 ± 0.67	86.74 ± 0.57
	Normal (long)	21.79 ± 0.27	32.85 ± 0.50	60.95 ± 0.81	70.20 ± 1.01
	LLF	22.54 ± 0.29	32.26 ± 0.48	67.79 ± 0.85	68.43 ± 1.07
	SEAL (ours)	**27.44 ± 0.40**	**46.96 ± 0.53**	**82.45 ± 0.53**	**92.47 ± 0.39**
LA	Normal	22.91 ± 0.36	49.40 ± 1.14	81.52 ± 1.38	87.43 ± 1.67
	Normal (long)	23.38 ± 0.37	47.36 ± 1.25	80.03 ± 1.53	89.63 ± 1.53
	LLF	21.10 ± 0.25	30.54 ± 1.22	46.55 ± 2.42	39.80 ± 2.30
	SEAL (ours)	**26.99 ± 0.46**	**55.12 ± 0.77**	**87.70 ± 0.52**	**95.67 ± 0.28**
		5-way, 50-shot			
Linear	Normal	26.55 ± 0.36	46.29 ± 0.47	82.20 ± 0.61	90.51 ± 0.45
	Normal (long)	22.56 ± 0.29	36.57 ± 0.53	67.14 ± 0.77	80.60 ± 0.78
	LLF	23.78 ± 0.31	35.59 ± 0.5	73.76 ± 0.79	79.67 ± 0.78
	SEAL (ours)	**29.78 ± 0.40**	**51.46 ± 0.50**	**84.99 ± 0.52**	**94.91 ± 0.29**
LA	Normal	24.2 ± 0.40	60.27 ± 1.10	88.09 ± 1.30	93.80 ± 1.33
	Normal (long)	24.62 ± 0.45	56.98 ± 1.32	85.17 ± 1.58	89.8 ± 1.9
	LLF	22.56 ± 0.34	41.53 ± 1.65	58.69 ± 2.76	51.35 ± 2.92
	SEAL (ours)	**27.13 ± 0.45**	**60.59 ± 1.11**	**91.94 ± 0.53**	**97.91 ± 0.40**

Table 7: Low-shot transfer results for the CFSDL benchmark (extreme distribution shift) for 20 and 50 shots. We fine-tuned both the linear layer (Linear) and the linear layer along with affine parameters (LA). We observed that SEAL outperforms standard training and that LLF severely under-performs in this case. Tuning the affine parameters and linear layer provides consistent performance gains for both Normal training models and SEAL.

with a few shot learning problem on multiple datasets from medical imaging to satellite images, with only one dataset of natural images available for pre-training. Several approaches to this problem exist, some utilizing meta-learning methods [60] and others focused on transfer learning [55]. It has been shown in multiple works that for this distant few shot learning task, transfer learning approaches exceed meta-learning [55]. We use our Tiny-ImageNet models from the previous section as base model for FSL transfer. In transfer we train a linear head as in the standard protocol [54, 55], we also consider jointly training linear head and affine parameters as suggested by [55]. Results of our evaluations for 1, 5, 20, and 50 shots are shown in Table 6 and Table 7. We observe that in this challenging benchmark LLF substantially underperforms standard training, suggesting features learned by LLF do not generalize well. We note that the training accuracies on Tiny-Imagenet of all the suggested models are 100% and the testing accuracies are the ones shown in Table 2.

Although LLF has higher in-distribution performance than normal training, its FSL transfer properties are much worse. On the other hand, SEAL features generalize both in-distribution and to the distant FSL tasks.

Evaluating on Smaller Models: In this experiment, we evaluate both the in-domain and transfer learning performance of SEAL, LLF, and normal training using ResNet-18 on Tiny-ImageNet. We do not modify any of the hyper-parameters that were used for ResNet-50. We observe that SEAL outperforms both LLF and Normal training on this model as well (Table 8).

Generation	**Normal**	**LLF**	**Ours**
Gen=1	50.47	-	-
Gen=3	48.32	52.36	**53.69**
Gen=10	46.66	53.64	**54.75**

Table 8: Comparison of our method with normal training and LLF on Tiny-ImageNet with ResNet-18. Please note that the behavior of the first generation for all methods is the same. We outperform standard long training and LLF on ResNet-18 as well.

Furthermore, we transfer the models trained on ResNet-18 to multiple datasets. We observe the same performance improvements as we saw with ResNet-50 with this smaller model as well (Table 9).

Method	**Tiny-ImageNet**	**Flower**	**CUB**	**Aircraft**	**MIT**	**Dogs**
Normal	50.47	31.47	7.47	7.14	28.20	11.85
Normal (long)	46.66	19.11	5.36	4.80	21.71	8.17
LLF	53.64	31.66	7.19	6.09	25.67	11.64
SEAL (Ours)	**54.75**	**40.68**	**9.87**	**8.85**	**33.65**	**14.61**

Table 9: Transferring features learned from Tiny-ImageNet with ResNet-18 to other datasets using linear probe. Normal, and Normal (long) refer to $G = 1$ and $G = 10$ generations of training, respectively. LLF and SEAL were trained for $G = 10$ generations. Our method, after $1,600$ epochs, surpasses both LLF and normal training by a large margin. This demonstrates that our method learns much more generalizable features as compared to Normal training or LLF.

3.5 Analysis and Ablation Studies

In this section we present additional analysis of our method, specifically we study the effects on the prediction depth as well as on the eigenvalues of the Hessian at the end of model training. Finally, we perform ablation studies to demonstrate that the early layers indeed benefit the most from SEAL.

Analysis of Hessian Eigenvalues We first study the eigenvalue spectra of the Hessian for the different proposed models. We utilize the same process for estimating the eigenvalues suggested in [61]. The results for the 4 models are summarized in Table 10 where we report both the maximum eigenvalues and the percentage of negative eigenvalues. We observe that the maximum eigenvalues are smaller for SEAL than for normal long training and also for LLF, suggesting a flatter minimum. Flatter minima have been previously associated with improved generalization [62].

We further observe that SEAL has no negative eigenvalues. This suggests that SEAL obtains some of its advantages by helping to avoid saddle points during training. This is consistent with prior uses of simulated annealing [7]. Following [61] we hypothesize the absence of negative eigenvalues can indicate a more robust solution.

Method	**Max Eigenval**	**Negative % Eigenval**
Normal	889.06	4.25%
Normal (long)	2353.74	0%
LLF	1027.21	14.63%
SEAL (Ours)	847.89	0%

Table 10: In this table, we demonstrate the statistics of the Hessian eigenvalues. We observe that our method has a lower max eigenvalue which suggests flatter local minima. Furthermore, our method has no negative eigenvalues, suggesting SEAL can avoid saddle points.

Prediction Depth: Zhou et al. [6] proposed LLF to specifically enhance the prediction depth of the model. Their intuition was that periodically resetting the final layers would decrease the prediction depth. Following Zhou et al. [6], we approximate the prediction depth using the K Nearest Neighbor (KNN) probe (with $K = 5$) on different layers of the network. To do so, for every image in the test set, we use all of the images in the train set for the KNN.

We affirm that the prediction depth of LLF is improved over normal training (Figure 3). However, with SEAL, we achieve a much stronger prediction depth. For instance, the KNN accuracy of our method is more than 18.54% stronger than LLF and 25.04% stronger than normal training on the outputs of the second block of the network. This is the layer threshold L used in our method and LLF. The layer-wise accuracy of the other layers indicates the superiority of our method across all layers.

Baldock et al. [28] demonstrated that example difficulty is correlated with prediction depth; decreasing the prediction depth corresponds to lower example difficulty, which is desired. They show this correlation by analyzing the speed of learning, the input and output margin, and the adversarial input margin for each data point. In Figure 4, we measure the prediction depth evolution of the three methods over different generations ($G = [1, 2, 4, 10]$). For normal training, the prediction depth gets worse over time, which explains its poor in-distribution performance. This suggests that in normal training, the early layers are becoming weaker after $G = 1$ and more samples are being classified by the later layers.

LLF slightly improves the prediction depth of the model. However, this comes with a deterioration of the performance of the later layers. For instance, the KNN probe on the activations of Block 4.1 in $G = 1$ has 50.15% accuracy which decreases to 45.62 in $G = 10$. On the other hand, our method does a better job of pushing more examples to be classified in the early layers than normal training and LLF. This implies that SEAL promotes relearning the more difficult samples using the simpler and more general features of the early layers. Furthermore, the later layers of the network improve over time with our method.

Ablation Study: We now investigate different strategies for the fit hypothesis $\mathbf{H}_{fit}$ during the forgetting phase. By default, during this phase, we perform gradient descent on the fit hypothesis and gradient ascent on the forgetting hypothesis. In "Ours+Reinit", following LLF [6], at the beginning of the forgetting phase, we reinitialize the fit hypothesis and during this phase, we perform gradient descent on these layers. We observe that not using the re-initialization leads to higher accuracy. Further, we demonstrate that freezing the final layers during the forgetting phase has a negative impact on the training. Finally, in "Ours+Reverse", we swap the fit and forgetting hypotheses, where we perform the gradient ascent on the later layers. We observe that performing simulated annealing in later layers fails drastically. This shows the importance of promoting the early layers and affirms the observations of Baldock et al. [28].

Gen	Normal	SEAL+Freeze	SEAL+Reinit	SEAL+Reverse	SEAL+Descent
Gen1	54.37	-	-	-	-
Gen3	51.16	52.45	56.82	50.25	**58.25**
Gen10	49.27	51.17	58.87	41.05	**59.22**

Table 11: Fitting hypothesis $\mathbf{H}_{fit}$ ablation study. While performing gradient ascent on the early layers during forgetting, we freeze, reinitialize, and perform gradient descent on the later layers. In reverse, we swap the fit and forgetting hypotheses. We observe that doing gradient descent on the fitting hypothesis during the forgetting phase leads to the best performance.

Furthermore, we demonstrate that **SEAL is not sensitive to forgetting frequencies**. To this end, in an experiment, we evaluate the sensitivity of SEAL to different forgetting frequencies. For this, we test multiple values for the number of epochs per generation E. The fewer the number of epochs in a generation more is the number of forgetting stages. We do not modify any other hyper-parameter. As summarized in Table 12, we observe that our method is not sensitive to the forgetting frequency and significantly improves over Normal training with any forgetting frequency. Please note that in this experiment, the maximum number of training epochs is different as each model is trained for E epochs for 10 generations.

Gen	E=160(default)	E=60	E=70	E=80	E=90	E=100	E=120	E=200
Gen1	54.37	53.33	53.62	53.59	53.66	53.84	53.92	54.07
Gen3	58.25	54.00	55.36	57.04	57.8	57.21	57.59	57.78
Gen10	**59.22**	**56.56**	**57.49**	**58.35**	**58.37**	**59.36**	**59.50**	**59.47**

Table 12: Dependency of SEAL on forgetting frequency in ResNet-50. Numbers in the columns indicate the number of epochs per generation E. Every E epochs, we perform gradient ascent for $k = \frac{E}{4}$ epochs. Each model is trained for $G = 10$ generations. We can see that our method has a significant positive impact on every forgetting frequency.

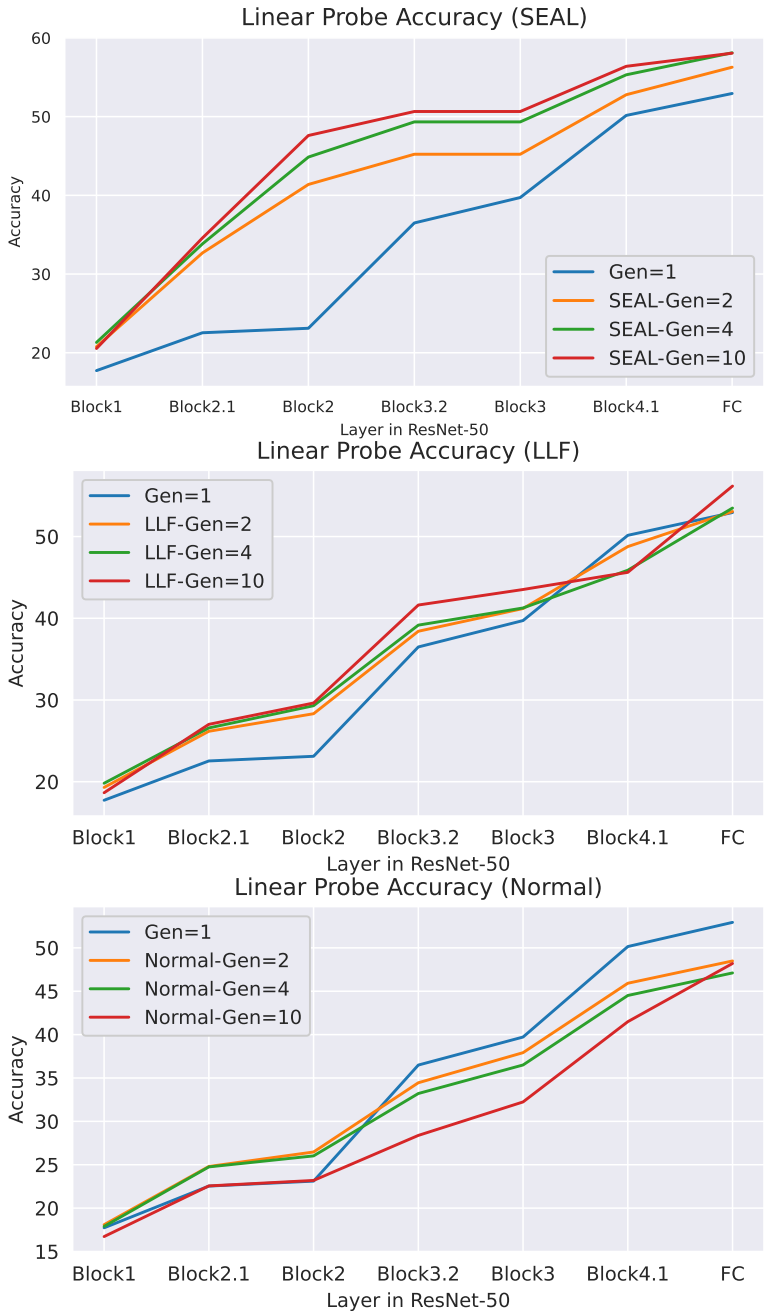

Figure 4: Evolution of Prediction Depth over Epochs for LLF, SEAL, and Normal training. Normal training worsens the prediction depth after the first generation which explains its poor in-distribution performance. LLF slightly improves the prediction depth of the model, however, it hurts the performance of the later layers of the network. SEAL shows the most significant improvement in prediction depth, while the later layers are improving over time.

3.6 Conclusion

In this chapter, we used the simulated annealing concept (intermittent heating and gradual cooling) in iterative training. We perform intermittent gradient ascent on the early layers for a few epochs. Following the iterative training literature, we do not perform gradient ascent on all the network parameters to ensure that the model maintains information from its previous state. This allowed us to obtain another perspective on the recently introduced later layer forgetting and the need to reset layers. We show that our method, SEAL, performs better than the state-of-the-art iterative training method, LLF, in an in-distribution setting. Moreover, we observed promising transfer learning performance in both natural image data and popular cross-domain few shot learning benchmarks. Investigating our approach illustrated it can greatly improve network prediction depth. Finally, we demonstrated that current iterative learning methods can have very poor generalization under transfer learning.

Chapter 4

Simulated Annealing in Text to 3D Generation

In chapter 3 we studied Simulated Annealing (SA) in computer vision, and more specifically, classification tasks that involve iterative (long) training. We demonstrated that SA helps to train more generalizable neural networks than other iterative training methods [2–4, 6] in both in-distribution and transfer learning scenarios. Furthermore, Cai et al. [7] introduced Simulated Annealing for Gradient Descent (SA-GD), where they adapted SA in gradient descent optimization and demonstrated the strength of SA in training neural networks. Their experiments also mainly focused on computer vision and were limited to the classification task. Therefore, the following question still remains: Is simulated annealing useful outside computer vision and classification tasks? In this chapter, we demonstrate a use case of SA in text-to-3D generative methods. We demonstrate a use case for SA in text-to-3D generative methods, showing how normal training fails and how a simplified version of simulated annealing provides a simple solution that solves the problem.

4.1 Introduction

The demand for 3D digital content has been increasing in various industries, such as gaming, entertainment, architecture, and robotics simulation. However, creating professional 3D content requires specialized skills and knowledge in 3D modeling, which can take a significant amount of time and effort to acquire. This makes it a challenging field for beginners. By using natural language to generate 3D content, text-to-3D generative models can democratize 3D content creation, making

it accessible to more people, including novices. Furthermore, the use of text-to-3D generative models can also improve the workflow of experienced artists, who currently spend a great deal of time and expertise in manually designing 3D assets for simulators, video games, and movies. By automating this process, text-to-3D generative models have the potential to improve the efficiency of experienced artists and make 3D content creation more accessible and scalable across a wide range of applications.

Neural networks that generate 3D data can be trained on explicit representations such as point clouds [63–65] and voxels [66, 67]. However, 3D data is much scarcer than 2D data, which makes it difficult to train these models to achieve adequate performance. Furthermore, Generative Adversarial Networks (GANs) have been used to learn controllable 3D generators from photos of a single object category by applying an adversarial loss on 2D image renderings of the resulting 3D object or scene [68–70]. While GANs have shown impressive results in generating specific object categories, such as faces, there is currently no evidence of their ability to generate 3D models from arbitrary text prompts [71].

Neural Radiance Fields (NeRF) [72] is a technique for inverse rendering that involves combining a volumetric raytracer with a neural network. The network maps spatial coordinates to color and volumetric density, making NeRF a crucial tool for neural inverse rendering [73]. Originally, NeRF was utilized in 3D reconstruction tasks where, given multiple images of a scene along with their camera information, the task is to understand the underlying structure of that scene in an implicit representation to synthesize novel and unobserved views. Recently, numerous 3D generative techniques have achieved positive outcomes by integrating NeRF-inspired models as a fundamental component within a more extensive generative framework [74–76]. Dream Fields [77] is an example of such an approach, which employs pre-trained image-text joint embedding models from CLIP and an optimization-driven methodology to train NeRFs. The study indicated that pre-trained 2D image-text models could be utilized for 3D synthesis, but the resulting 3D objects generated through this method often lack realism and precision. Clip-Mesh [78] uses the same loss function but optimizes a mesh representation directly, leading to much faster generation.

With the success of diffusion models in text-to-2D image generation tasks [19], DreamFusion [71] utilizes a novel loss function called Score Distillation Sampling (SDS). The technique uses pre-trained text-to-2D diffusion models as guidance to optimize a NeRF representation based on a text prompt. The generated 3D models

have much better quality compared to CLIP-driven methods [77,78]. However, due to optimizing a NeRF representation, DreamFusion is much slower than Clip-Mesh [78]. In this chapter, we mainly use DreamFusion for text-to-3D generation.

All of the mentioned models typically start from a randomly initialized representation and optimize to represent the desired textual prompt. The slowness is due to the number of steps required to optimize this representation. Nevertheless, if the 3D representation is initialized from a more suitable representation for the text prompt, optimization can become orders of magnitude faster. For instance, if optimizing from random initialization requires 200 epochs to become a "red rabbit", a "white rabbit" can become "red" in only 20 epochs. Furthermore, since "white rabbits" are much more common in the datasets that the text-to-2D diffusion models were trained on compared to "red rabbits", the quality of generated "white rabbits" is much better than that of the red ones. Therefore, we find that initializing from a "white rabbit" to a "red rabbit" is not only faster but results in much better quality than directly generating the "red rabbit". Hence, being able to optimize from a good initialization to a target text prompt (rather than always using a random initialization) is an important task.

However, when a 3D representation is optimized for a source text prompt and is then optimized for a new target text prompt that is similar to the source text prompt, the optimization often results in suboptimal solutions. For example, if we start with a white rabbit with pink ears and ask the model to generate a "purple rabbit", the model makes the minimum effort to do so. That is, it only changed the color of ears from "pink" to "purple". We hypothesize that this is because the loss becomes very small, and the network gets stuck in a local meaning. This prohibits the model to change the 3D representation substantially and generate a proper "purple rabbit" (refer to figure 5). In this chapter, we utilize a very simplified version of the Simulated Annealing method proposed in chapter 3 to DreamFusion to resolve this issue. in section 4.4, we qualitatively show multiple examples where normal optimization fails drastically, and our simplified simulated annealing approach is able to generate proper results.

In what follows, we will first provide a brief background on the DreamFusion paper and the SDS loss in section 4.2. We will then introduce our proposed method in section 4.3 a and compare it to previous simulated annealing techniques, such as SA-GD [7] and SEAL (chapter 3) in. Finally, in section 4.4, we demonstrate the importance of our method by showcasing qualitative examples.

4.2 Background

In this section, we define the Score Distillation Sampling (SDS) method proposed in DreamFusion [71], as it is the core loss function we use in our work.

Text-to-image diffusion models (e.g., stable diffusion [21]) are trained such that given a realistic image I_{real} and a randomly generated noise ϵ added to the image I_{real}, the model is able to differentiate between the real image and the noise. Now, let us consider a realistic image I_{real} and a fake (generated) image I_{fake}. We denote noisy versions of these images as I^{ϵ}_{real} and I^{ϵ}_{fake}, respectively. The intuition behind SDS is that if the network is properly trained, given the I^{ϵ}_{real}, the network would be easily able to tell I_{real} from ϵ, whereas for I^{ϵ}_{fake}, it would struggle. How well the network can differentiate between the noise and the image provides a signal to update the input image toward becoming more realistic. If the image is generated from differentiably rendering a 3D representation, we can use this loss to update any 3D representation to produce more realistic images.

Formally, SDS is based on probability density distillation, which involves minimizing the KL divergence between a group of Gaussian distributions that share means based on the forward process of diffusion and the score functions learned by a pre-trained diffusion model. This leads to the development of the SDS method, which allows for optimization in differentiable image parameterizations and facilitates sampling. The following equation provides the SDS signal:

$$\nabla_{theta} f(x = g_{\theta}(S)) = \nabla_{\theta}\, \mathbb{E}_t \left(C_t \mathrm{KL}\Big(q(z_t|g_{\theta}(S); y, t) || p(z_t|y, t) \Big) \right) \tag{7}$$

Here, g_{θ} is the differentiable process with learnable parameters θ that generates an image x from the 3D representation S. q represents the forward process of the text-to-image diffusion model, t is the noise timestep for the diffusion model, and C_t is a coefficient for this time step. p is the backward process of the diffusion model, and y is the text prompt. Finally, f refers to the SDS loss.

4.3 Proposed Method

DreamFusion [71] start from an initialized 3D NeRF representation S_{init} and iteratively optimizes this representation in multiple epochs using the SDS loss based on a text prompt t_1. We denote $S^k_{t_1}$ as the optimized NeRF representation for the t_1 text prompt after k epochs.

(a) Base Rabbit

(b) Purple rabbit from scratch

(c) resume SA

(d) resume Normal

Figure 5: (a) refers to a NeRF trained on t_1 = a rabbit for k_1 epochs from random initilization (top-left). (b) is trained on t_2 = a purple rabbit for k_1 epochs from random initialization (top-right). The bottom images refer to resuming from (a) trained once with our method ("resume SA", bottom-left (c)), and once with normal training ("resume normal", bottom-right (d)), for k_2 epochs on t_2. We observe that normal training on t_2 starting from both a random initialization and a good initialization fails, while our method is able to properly present t_2.

Typically, DreamFusion randomly initializes the NeRF representation S_{init}. Doing so is inefficient as it requires many epochs to properly optimize for the target text prompt. In this work, we focus on starting from a good initialization $S^k_{t_1}$ for a text prompt t_1 and study how it optimizes to a new text prompt t_2 which is close to t_1. Let us note k_2 as the number of optimization epochs for the new text prompt t_2,

where the resulting NeRF representation is given by $S_{t_2}^{k+k_2}$.

First, we normally optimize the NeRF representation $S_k^{t_1}$ for k_2 epochs on the new text prompt t_2, using gradient-descent:

$$\mathbf{\Theta}_{NeRF}^{k,i+1} = \mathbf{\Theta}_{NeRF}^{k,i} - \eta \nabla f(\mathbf{\Theta}_{NeRF}^{k,i}, t_2) \tag{8}$$

Where $\mathbf{\Theta}_{NeRF}^{k,i}$ refers to the trainable parameters of the NeRF representation $S_{t_2}^{k,i}$ at epoch k, iteration i, and $f(.)$ refers to the SDS loss derived from the new text prompt t_2 and the current NeRF representation. However, doing so results in minimal changes meaning that the difference between $S_{t_1}^{k_1}$ and $S_{t_2}^{k_1+k_2}$ is negligible if t_1 and t_2 are close. We believe that this is due to the small magnitude of the loss signal $f(.)$, and that the system is stuck in a flat local minimum for t_2.

To resolve the issue, we use a very simplified version of simulated annealing. In particular, to optimize from the initialization $S_{t_1}^{k}$ based on the text prompt t_2, we first perform gradient ascent for a few epochs k_a. Following our observations in SEAL (chapter 3), we reduce the ascent-phase learning rate using a factor S. We perform normal gradient descent for the rest of the training ($k_2 - k_a$ epochs):

$$\mathbf{\Theta}_{NeRF}^{k,i+1} = \begin{cases} \mathbf{\Theta}_{NeRF}^{k,i} - \eta \nabla f(\mathbf{\Theta}_{NeRF}^{k,i}, t_1), & \text{if} \quad k < k_1 \\ \mathbf{\Theta}_{NeRF}^{k,i} + \eta_{\text{a}} \nabla f(\mathbf{\Theta}_{NeRF}^{k,i}, t_2), & \text{if} \quad k_1 < k < k_a \\ \mathbf{\Theta}_{NeRF}^{k,i} - \eta \nabla f(\mathbf{\Theta}_{NeRF}^{k,i}, t_2), & \text{if} \quad k_a < k < k_2 \end{cases} \tag{9}$$

Where similar to SEAL, $\eta_{texta} = S \times \eta$, is the reduced learning rate used during the gradient ascent phase, and f(.) is the SDS loss function. Similar to SEAL, we fix $S = 0.01$ and $k_a = \frac{1}{4}k_2$.

This method can be categorized under iterative training methods with the forget-and-relearn scheme proposed in [6]. In particular, we train the model in two generations. We perform SEAL-style forgetting on the network. Instead of breaking the network into exclusive parts, we consider the whole network as both the fit and forgetting hypotheses. This is similar to the simulated annealing utilized in SA-GD [7]. However, SA-GD uses a temperature with multiple hyper-parameters to decide whether to ascend or not. We simplify this (similar to SEAL) and do the ascending at fixed intervals (for a fixed number of epochs).

In figures 5 and 6, we observe our method resolves the issue; the model is able to escape the local minima and find much better solutions for the new text prompt t_2.

4.4 Experiments

4.5 Implementation Details

We follow the implementation of DreamFusion [71] in this work. The only difference is that DreamFusion uses Imagen [20] as their text-to-image diffusion model. However, Imagen is not publicly available. Instead, we make use of stable-diffusion [21] which is public. To optimize the NeRF representation, we need to render the NeRF from a random camera view and use the SDS loss to update the NeRF parameters Θ_{NeRF}. Following DreamFusion:

1. Sampling Camera. During each iteration, the camera position is randomly sampled. The camera's position is chosen randomly in spherical coordinates, with the elevation angle within the range of -10° to 90°, $\phi_{cam} \in [-10, 90]$, the azimuth angle within the range of 0° to 360°, $\theta \in [0, 360]$, and the distance from the origin between 1 to 1.5 units. To form a camera pose matrix, we also randomly select a "look-at" point and an "up" vector. Additionally, we randomly choose a focal length multiplier λ_{focal} from a uniform distribution ranging from 0.7 to 1.35, where the focal length is λ_{focal} times the image width of 64 pixels.

2. Rendering. After obtaining the camera pose, we use it to render the NeRF model at a resolution of 64x64. All the rendering is done on albedo without shading.

3. Diffusion loss and view conditioning. The text prompts only describe the object of interest and do not take into account the camera position. This is problematic because objects usually do not look the same from all angles. For instance, consider the 3D representation of a human's head. The front view should be optimized to demonstrate the face, the side view should mainly represent ears and hair, and the back view should focus on hair. Optimizing all angles of a NeRF to fit a single text prompt "A human's head" usually fails, as different sides of the generated object may optimize to represent the same thing. For example, the back view of the object may show "face" instead of "hair". To counter this, Poole et al. [71] discovered that it is advantageous to augment the text prompt with view-dependent text (e.g., appending ", side view" to the end of the original text prompt).

Unlike DreamFusion, which uses Imagen [20] as their text-to-image model, we use stable-diffusion [21] since it is open-source. We use a very high guidance weight

$w = 100$ in stable diffusion, following DreamFusion. Finally, we use SDS loss to derive the diffusion loss and update NeRF parameters using this loss.

4. optimization. To optimize the NeRFs, we typically train for $k_1 = 200$ epochs ($20,000$ iterations) on the source text prompt t_1. On the new text prompt t_2, we perform $k_a = 0$ epochs of ascent for normal training and $k_a = 10$ epochs ($1,000$ iterations) of ascent for our approach. Finally, we optimize the new text prompt for $k_a + k_2 = 50$ epochs ($5,000$ iterations). Note that the number of iterations for both normal training and our method (both forgetting, and relearning phase) is the same for fairness.

4.6 Qualitative Results

The goal of this work is to take a NeRF that is optimized on t_1 for k_1 epochs and optimize it on a new text prompt t_2 (which is close to t_1). This is especially helpful as the number of epochs needed to optimize on the new text prompt is far lower than optimizing on it from scratch $k_2 << k_1$, and that sometimes, directly optimizing on k_2 leads to worse quality.

Therefore, we first optimize a NeRF using DreamFusion on t_1 = a rabbit for $k_1 = 200$ epochs. Then, the goal is to optimize it on a new text prompt t_2 = a purple rabbit, which is very close to t_1. To optimize on t_2, we investigate the following three methods:

Method 1: In this method, we directly optimize for k_1 epochs on the new text prompt t_2 from random initialization. We observe that if the text prompt is unusual (e.g., “a purple rabbit"), sometimes the generated NeRF does not look good compared to first training on a more conventional prompt (like “a rabbit"), and then optimizing on t_2. Furthermore, always starting from random initialization is much slower than optimizing from existing NeRFs.

Method 2: In this method, we take the pre-optimized $S_{t_1}^{k_1}$ and optimize it (normally) on t_2 for $k_2 = 40$ epochs. We observe that the model makes the least amount of effort to adapt to the new text prompt, leading to a representation $S_{t_2}^{k_1+k_2}$ that does not properly represent t_2 where the difference between $S_{t_1}^{k_1}$ and $S_{t_2}^{k_1+k_2}$ is negligible.

(a) Base Gas Can (b) resume Normal (c) resume SA

Figure 6: (a) refers to a NeRF trained on t_1 = a gas can for k_1 epochs from random initilization. (b) and (c) refer to resuming from (a) trained once with our method ("resume SA", (c)), and once with normal training ("resume normal", (d)), for k_2 epochs on t_2 = an ebony gas can. Similar to the rabbit experiment, we can see that normal training fails while our method is able to optimize well on the new text prompt.

Method 3 (ours): In this method, we first perform $k_a = 10$ epochs of gradient ascent on the pre-optimized NeRF representation $S_{t_1}^{k_1}$ using t_2 to obtain the perturbed representation $S_{t_2}^{k_1+k_a}$. Then, we train normally on the new text prompt for $k_2 - k_a$ epochs and obtain $S_{t_2}^{k_1+k_2}$. We observe that the new representation properly represents t_2 and is substantially different from $S_{t_1}^{k_1}$.

In figure5, we observe that while normal training on t_2 fails drastically both when trained from random initialization and from $S_{t_1}^{k_1}$, our method is able to properly represent the new text prompt t_2.

We repeat this experiment on t_1 = a gas can and t_2 = an ebony gas can and observe very similar results, summarized in figure 6.

4.7 Conclusion

In this chapter, we extended SEAL-style forgetting to a task other than classification and computer vision, namely, text-to-3D generation. We observed that normal optimization can lead to sub-optimal solutions due to getting stuck in local minima. To address this issue, we introduced a simplified version of simulated annealing that enables escaping local minima and finding more suitable solutions.

Chapter 5

Conclusions

In our study, we utilized the concept of simulated annealing, which involves intermittent heating and gradual cooling, in deep learning. To counter-act overfitting when training a network for many epochs under iterative training benchmarks, we proposed a novel forgetting technique inspired by simulated annealing. Specifically, we proposed a simplified simulated annealing on the early layers of the network. We perform simulated annealing only on the early layers of the network to enhance prediction depth of the model, and to ensure that the model retains information from its previous knowledge. This approach allowed us to gain another perspective on later layer forgetting, which was the state-of-the-art in iterative training, as well as the need to re-initialize layers in iterative training. Our method, SEAL, was demonstrated to outperform the state-of-the-art iterative training method, LLF, in in-distribution settings. Our investigations also demonstrated that our method (SEAL) can significantly enhance the network's prediction depth. Finally, we showed that existing iterative learning methods can exhibit poor generalization in transfer learning situations.

In addition, we applied simulated annealing to the task of text-to-3D generation, with the aim of extending its use beyond classification and computer vision tasks. Our experiment focused on investigating the behavior of existing text-to-3D models when given a NeRF representation that accurately represents a source text prompt and optimizing it for a new target text prompt that is similar to the previous one. We found that these models frequently failed, becoming trapped in local minima and producing suboptimal results. To address this problem, we introduced a new and simplified version of simulated annealing, which enabled the network to escape local minimas and uncover more appropriate solutions.

This thesis showcases the practical applications of simulated annealing in deep

learning across various tasks and data modalities. We encourage further investigation into the potential of simulated annealing for training deep neural networks in future research.

References

[1] A. Taha, A. Shrivastava, and L. S. Davis, "Knowledge evolution in neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12843–12852, 2021. x, 9, 10, 16, 17, 19, 23, 26

[2] T. Furlanello, Z. Lipton, M. Tschannen, L. Itti, and A. Anandkumar, "Born again neural networks," in *International Conference on Machine Learning*, pp. 1607–1616, PMLR, 2018. x, 5, 9, 10, 12, 16, 17, 27, 28, 37

[3] C. Yang, L. Xie, S. Qiao, and A. L. Yuille, "Training deep neural networks in generations: A more tolerant teacher educates better students," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 5628–5635, 2019. x, 5, 9, 10, 12, 16, 17, 27, 28, 37

[4] M. Pham, M. Cho, A. Joshi, and C. Hegde, "Revisiting self-distillation," *arXiv preprint arXiv:2206.08491*, 2022. x, 5, 12, 27, 28, 37

[5] A. Sarfi, Z. Karimpour, M. Chaudhary, N. Khalid, M. Ravanelli, S. Mudur, and E. Belilovsky, "Simulated annealing in early layers leads to better generalization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 4

[6] H. Zhou, A. Vani, H. Larochelle, and A. Courville, "Fortuitous forgetting in connectionist networks," *arXiv preprint arXiv:2202.00155*, 2022. 5, 9, 10, 16, 17, 19, 20, 21, 23, 25, 27, 32, 33, 37, 42

[7] Z. Cai, "Sa-gd: Improved gradient descent learning strategy with simulated annealing," *arXiv preprint arXiv:2107.07558*, 2021. 5, 18, 21, 32, 37, 39, 42

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. 6

[9] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015. 6

[10] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, *et al.*, "Learning transferable visual models from natural language supervision," in *International Conference on Machine Learning*, pp. 8748–8763, PMLR, 2021. 7

[11] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983. 7, 18

[12] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin, "Stabilizing the lottery ticket hypothesis," *arXiv preprint arXiv:1903.01611*, 2019. 9, 10, 16, 17, 19, 23

[13] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018. 10, 17, 19, 23

[14] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015. 11

[15] G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker, "Learning efficient object detection models with knowledge distillation," *Advances in neural information processing systems*, vol. 30, 2017. 11

[16] R. Tang, Y. Lu, L. Liu, L. Mou, O. Vechtomova, and J. Lin, "Distilling task-specific knowledge from bert into simple neural networks," *arXiv preprint arXiv:1903.12136*, 2019. 11

[17] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018. 11

[18] J. Kurchan, "Fluctuation theorem for stochastic dynamics," *Journal of Physics A: Mathematical and General*, vol. 31, no. 16, p. 3719, 1998. 13

[19] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020. 13, 38

[20] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. Denton, S. K. S. Ghasemipour, B. K. Ayan, S. S. Mahdavi, R. G. Lopes, *et al.*, "Photorealistic text-to-image diffusion models with deep language understanding," *arXiv preprint arXiv:2205.11487*, 2022. 13, 43

[21] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, 2022. 13, 14, 15, 40, 43

[22] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," *arXiv preprint arXiv:2204.06125*, 2022. 13

[23] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 1999. 16

[24] Y. Yao, L. Rosasco, and A. Caponnetto, "On early stopping in gradient descent learning," *Constructive Approximation*, vol. 26, pp. 289–315, 2007. 16

[25] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014. 16

[26] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, "Sharpness-aware minimization for efficiently improving generalization," *arXiv preprint arXiv:2010.01412*, 2020. 16

[27] H. Mobahi, M. Farajtabar, and P. Bartlett, "Self-distillation amplifies regularization in hilbert space," *Advances in Neural Information Processing Systems*, vol. 33, pp. 3351–3361, 2020. 17

[28] R. Baldock, H. Maennel, and B. Neyshabur, "Deep learning through the lens of example difficulty," *Advances in Neural Information Processing Systems*, vol. 34, pp. 10876–10889, 2021. 17, 21, 33

[29] N. Zhao, Z. Wu, R. W. Lau, and S. Lin, "What makes instance discrimination good for transfer learning?," *arXiv preprint arXiv:2006.06606*, 2020. 17

[30] Y. Le and X. Yang, "Tiny imagenet visual recognition challenge," *CS 231N*, vol. 7, no. 7, p. 3, 2015. 18, 24

[31] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pp. 722–729, IEEE, 2008. 18, 26, 28

[32] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The Caltech-UCSD Birds-200-2011 Dataset," Tech. Rep. CNS-TR-2011-001, California Institute of Technology, 2011. 18, 26, 28

[33] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi, "Fine-grained visual classification of aircraft," *arXiv preprint arXiv:1306.5151*, 2013. 18, 26, 28

[34] A. Khosla, N. Jayadevaprakash, B. Yao, and F.-F. Li, "Novel dataset for fine-grained image categorization: Stanford dogs," in *Proc. CVPR workshop on fine-grained visual categorization (FGVC)*, vol. 2, Citeseer, 2011. 18, 26

[35] A. Quattoni and A. Torralba, "Recognizing indoor scenes," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 413–420, IEEE, 2009. 18, 26, 28

[36] Y. Guo, N. C. Codella, L. Karlinsky, J. V. Codella, J. R. Smith, K. Saenko, T. Rosing, and R. Feris, "A broader study of cross-domain few-shot learning," ECCV, 2020. 18, 23, 25

[37] H. Zhou, J. Lan, R. Liu, and J. Yosinski, "Deconstructing lottery tickets: Zeros, signs, and the supermask," *Advances in neural information processing systems*, vol. 32, 2019. 20

[38] X. Li, H. Xiong, H. An, C.-Z. Xu, and D. Dou, "Rifle: Backpropagation in depth for deep transfer learning through re-initializing the fully-connected layer," in *International Conference on Machine Learning*, pp. 6010–6019, PMLR, 2020. 20, 21

[39] Y. N. Dauphin, R. Pascanu, Ç. Gülçehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," *CoRR*, vol. abs/1406.2572, 2014. 22

[40] C. Jin, R. Ge, P. Netrapalli, S. M. Kakade, and M. I. Jordan, "How to escape saddle points efficiently," *CoRR*, vol. abs/1703.00887, 2017. 22

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International*

Conference on Neural Information Processing Systems - Volume 1, NIPS'12, (Red Hook, NY, USA), p. 1097–1105, Curran Associates Inc., 2012. 22

[42] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. 22

[43] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, jan 2016. 22

[44] O. Vinyals, C. Blundell, T. P. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," *CoRR*, vol. abs/1606.04080, 2016. 22

[45] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *CoRR*, vol. abs/1703.03400, 2017. 22

[46] J. Snell, K. Swersky, and R. S. Zemel, "Prototypical networks for few-shot learning," *CoRR*, vol. abs/1703.05175, 2017. 22

[47] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *ICLR*, 2017. 22

[48] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. S. Torr, and T. M. Hospedales, "Learning to compare: Relation network for few-shot learning," *CoRR*, vol. abs/1711.06025, 2017. 22

[49] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection," *Frontiers in plant science*, vol. 7, p. 1419, 2016. 23, 25

[50] P. Helber, B. Bischke, A. Dengel, and D. Borth, "Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 12, no. 7, pp. 2217–2226, 2019. 23, 25

[51] P. Tschandl, C. Rosendahl, and H. Kittler, "The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions," *Scientific data*, vol. 5, no. 1, pp. 1–9, 2018. 23, 25

[52] N. Codella, V. Rotemberg, P. Tschandl, M. E. Celebi, S. Dusza, D. Gutman, B. Helba, A. Kalloo, K. Liopyris, M. Marchetti, *et al.*, "Skin lesion analysis toward melanoma detection 2018: A challenge hosted by the international skin imaging collaboration (isic)," *arXiv preprint arXiv:1902.03368*, 2019. 23, 25

[53] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers, "Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2097–2106, 2017. 23, 25

[54] C. P. Phoo and B. Hariharan, "Self-training for few-shot transfer across extreme task differences," 2021. 23, 29, 30

[55] M. Yazdanpanah, A. A. Rahman, M. Chaudhary, C. Desrosiers, M. Havaei, E. Belilovsky, and S. E. Kahou, "Revisiting learnable affines for batch norm in few-shot transfer learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9109–9118, June 2022. 23, 29, 30

[56] R. Müller, S. Kornblith, and G. E. Hinton, "When does label smoothing help?," *Advances in neural information processing systems*, vol. 32, 2019. 25

[57] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016. 25

[58] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016. 25

[59] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," tech. rep., University of Toronto, 2009. 26, 27

[60] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*, pp. 1126–1135, PMLR, 2017. 30

[61] N. Park and S. Kim, "How do vision transformers work?," *arXiv preprint arXiv:2202.06709*, 2022. 32

[62] S. Hochreiter and J. Schmidhuber, "Flat minima," *Neural computation*, vol. 9, no. 1, pp. 1–42, 1997. 32

[63] L. Zhou, Y. Du, and J. Wu, "3d shape generation and completion through point-voxel diffusion," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5826–5835, 2021. 38

[64] R. Cai, G. Yang, H. Averbuch-Elor, Z. Hao, S. Belongie, N. Snavely, and B. Hariharan, "Learning gradient fields for shape generation," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pp. 364–381, Springer, 2020. 38

[65] G. Yang, X. Huang, Z. Hao, M.-Y. Liu, S. Belongie, and B. Hariharan, "Pointflow: 3d point cloud generation with continuous normalizing flows," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4541–4550, 2019. 38

[66] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling," *Advances in neural information processing systems*, vol. 29, 2016. 38

[67] K. Chen, C. B. Choy, M. Savva, A. X. Chang, T. Funkhouser, and S. Savarese, "Text2shape: Generating shapes from natural language by learning joint embeddings," in *Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part III 14*, pp. 100–116, Springer, 2019. 38

[68] P. Henzler, N. J. Mitra, and T. Ritschel, "Escaping plato's cave: 3d shape from adversarial rendering," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9984–9993, 2019. 38

[69] T. Nguyen-Phuoc, C. Li, L. Theis, C. Richardt, and Y.-L. Yang, "Hologan: Unsupervised learning of 3d representations from natural images," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7588–7597, 2019. 38

[70] R. Or-El, X. Luo, M. Shan, E. Shechtman, J. J. Park, and I. Kemelmacher-Shlizerman, "Stylesdf: High-resolution 3d-consistent image and geometry

generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13503–13513, 2022. 38

[71] B. Poole, A. Jain, J. T. Barron, and B. Mildenhall, "Dreamfusion: Text-to-3d using 2d diffusion," *arXiv preprint arXiv:2209.14988*, 2022. 38, 40, 43

[72] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021. 38

[73] A. Tewari, J. Thies, B. Mildenhall, P. Srinivasan, E. Tretschk, W. Yifan, C. Lassner, V. Sitzmann, R. Martin-Brualla, S. Lombardi, *et al.*, "Advances in neural rendering," in *Computer Graphics Forum*, vol. 41, pp. 703–735, Wiley Online Library, 2022. 38

[74] K. Schwarz, Y. Liao, M. Niemeyer, and A. Geiger, "Graf: Generative radiance fields for 3d-aware image synthesis," *Advances in Neural Information Processing Systems*, vol. 33, pp. 20154–20166, 2020. 38

[75] J. Gu, L. Liu, P. Wang, and C. Theobalt, "Stylenerf: A style-based 3d-aware generator for high-resolution image synthesis," *arXiv preprint arXiv:2110.08985*, 2021. 38

[76] Z. Liu, Y. Wang, X. Qi, and C.-W. Fu, "Towards implicit text-guided 3d shape generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 17896–17906, 2022. 38

[77] A. Jain, B. Mildenhall, J. T. Barron, P. Abbeel, and B. Poole, "Zero-shot text-guided object generation with dream fields," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 867–876, 2022. 38, 39

[78] N. Mohammad Khalid, T. Xie, E. Belilovsky, and T. Popa, "Clip-mesh: Generating textured meshes from text using pretrained image-text models," in *SIGGRAPH Asia 2022 Conference Papers*, pp. 1–8, 2022. 38, 39