THE REUSABILITY PRIOR IN DEEP LEARNING MODELS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

AYDIN GÖZE POLAT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

MAY 2023

Approval of the thesis:

THE REUSABILITY PRIOR IN DEEP LEARNING MODELS

submitted by **AYDIN GÖZE POLAT** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Prof. Dr. Ferda Nur Alpaslan
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Sinan Kalkan
Computer Engineering, METU _____

Prof. Dr. Ferda Nur Alpaslan
Computer Engineering, METU _____

Assoc. Prof. Dr. Çiğdem Turhan
Computer Engineering, Atılım University _____

Assoc. Prof. Dr. Kasım Öztoprak
Computer Engineering, Konya Food and Agriculture University _____

Assoc. Prof. Dr. Emre Akbaş
Computer Engineering, METU _____

Date:22.05.2023

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Aydın Göze Polat

Signature :

ABSTRACT

THE REUSABILITY PRIOR IN DEEP LEARNING MODELS

Polat, Aydın Göze
Ph.D., Department of Computer Engineering
Supervisor: Prof. Dr. Ferda Nur Alpaslan

May 2023, 175 pages

Various choices can affect the performance of deep learning (DL) models. For instance, repetitions in a model via cross-layer parameter sharing, using convolutional layers, and relying on skip connections affect the reusability of components in DL models, impacting parameter efficiency. In this work, three different approaches are proposed to investigate how different design choices in terms of such repetitions affect model performance. First, a new library, Revolver, is proposed to analyze reusable modules or model components while training a population of DL models. Reusing modules across models enabled training an entire population of models on a single GPU and collecting statistics about top scoring shared modules. Second, the reusability prior is proposed as follows: model components are forced to function in diverse contexts not only due to the training data, augmentation, and regularization choices but also due to the model design itself. Based on this prior, a counting-based graph analysis approach that can quantify the number of *contexts* for each learnable parameter is proposed. In the experiments, this approach was able to correctly predict

the ranking of several analyzed models in terms of top-1 accuracy without relying on any training. Third, a generalized framework inspired by statistical mechanics is proposed, where the context-based counting approach describes models with absolute temperature $T = -1$. The generalized framework allowed going beyond the proposed counting approach by encoding the constraints and assumptions in the form of energy at the parameter level. Overall, these approaches may open up avenues for research on model analysis and comparison or lead to practical applications for neural architecture search.

Keywords: deep learning, parameter sharing, genetic algorithms, reusability, statistical mechanics

ÖZ

DERİN ÖĞRENME MODELLERİNDE YENİDEN KULLANİLABİLİRLİK ÖNSELİ

Polat, Aydın Göze
Doktora, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Ferda Nur Alpaslan

Mayıs 2023 , 175 sayfa

Çeşitli seçimler, derin öğrenme (DÖ) modellerinin performansını etkileyebilir. Örneğin, katmanlar arası parametre paylaşımı, evrişimli katmanlarö ve atlama bağlantıları yoluyla bir modelde elde edilen tekrarlamalar, DL modellerindeki bileşenlerin yeniden kullanılabilirliğini etkileyerek parametre verimliliğini etkiler. Bu çalışmada, bu tür tekrarlamalar açısından farklı tasarım seçimlerinin model performansını nasıl etkilediğini araştırmak için üç farklı çözüm önerildi. İlk olarak, bir DÖ modelleri popülasyonunu eğitirken yeniden kullanılabilir modülleri veya model bileşenlerini analiz etmek için yeni bir kitaplık olan Revolver önerildi. Deneyler sırasında modeller arasında modüllerin yeniden kullanılması, tüm model popülasyonunun tek bir GPU üzerinde eğitilmesini ve en yüksek puanı alan paylaşılan modüller hakkında istatistiklerin toplanabilmesini sağladı. İkincisi, yeniden kullanılabilirlik önseli şu şekilde önerildi: model bileşenleri, yalnızca eğitim koşulları ve düzenlileştirme seçenekleri

nedeniyle değil, aynı zamanda model tasarımı nedeniyle de farklı bağlamlarda çalışmaya zorlanır. Bu önsele dayanarak, öğrenilebilir her parametre için *bağlam* sayısını ölçebilen saymaya dayalı bir grafik analizi önerildi. Deneylerde, bu yöntem, herhangi bir eğitime dayanmadan, ilk-1 doğruluk açısından analiz edilen birçok modelin sıralamasına ilişkin performansı doğru bir şekilde tahmin edebildi. Son olarak, istatistiksel mekanikten ilham alan genelleştirilmiş bir yaklaşım önerildi; burada bağlama dayalı sayma yönteminin, $T = -1$ mutlak sıcaklığına sahip modelleri tanımladığı anlaşıldı. Genelleştirilmiş yaklaşım, kısıtlamaları ve varsayımları parametre düzeyinde enerji şeklinde kodlayarak önerilen sayma yönteminin ötesine geçmeye izin verdi. Sonuç olarak, önerilen bu çözümler, model analizi ve karşılaştırması üzerine araştırmalara veya nöral mimari araştırması için pratik uygulamalara imkan sağlayabilir.

Anahtar Kelimeler: derin öğrenme, parametre paylaşımı, genetik algoritma, tekrar kullanılabilirlik, istatistiksel mekanik

To all the fathers, mothers, and daughters in my life

ACKNOWLEDGMENTS

I thank Ferda Nur Alpaslan, Emre Akbaş, Kasım Öztoprak, Selim Temizer, and Uğur Halıcı for their constructive input. Special thanks to my wife Georgina Romo Olivares and our parents for their support. Also, my daughter Iris Luna Polat, thank you for the most wonderful smiles in the world. A single one is worth all the sleepless nights I spent writing this work while you slept on my chest.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

List of Algorithms

LIST OF ABBREVIATIONS

BN	Batch Normalization
CAP	Credit Assignment Path
CNF	Convolutional Neural Fabrics
CNN	Convolutional Neural Network
DAG	Directed Acyclic Graph
DAN	Deep Adaptation Network
DBN	Deep Belief Network
DL	Deep Learning
ENAS	Efficient Neural Architecture Search
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
LSTM	Long Short Term Memory Network
ML	Machine Learning
MLP	Multilayer Perceptron
NAS	Neural Architecture Search
NLP	Natural Language Processing
PBT	Population Based Training
PCA	Principal Component Analysis
RBF	Radial Basis Function
RBM	Restricted Boltzmann Machine
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent

SOM	Self Organizing Maps
STN	Spatial Transformer Network
SVM	Support Vector Machine
ViT	Vision Transformer
WRN	Wide Residual Network

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

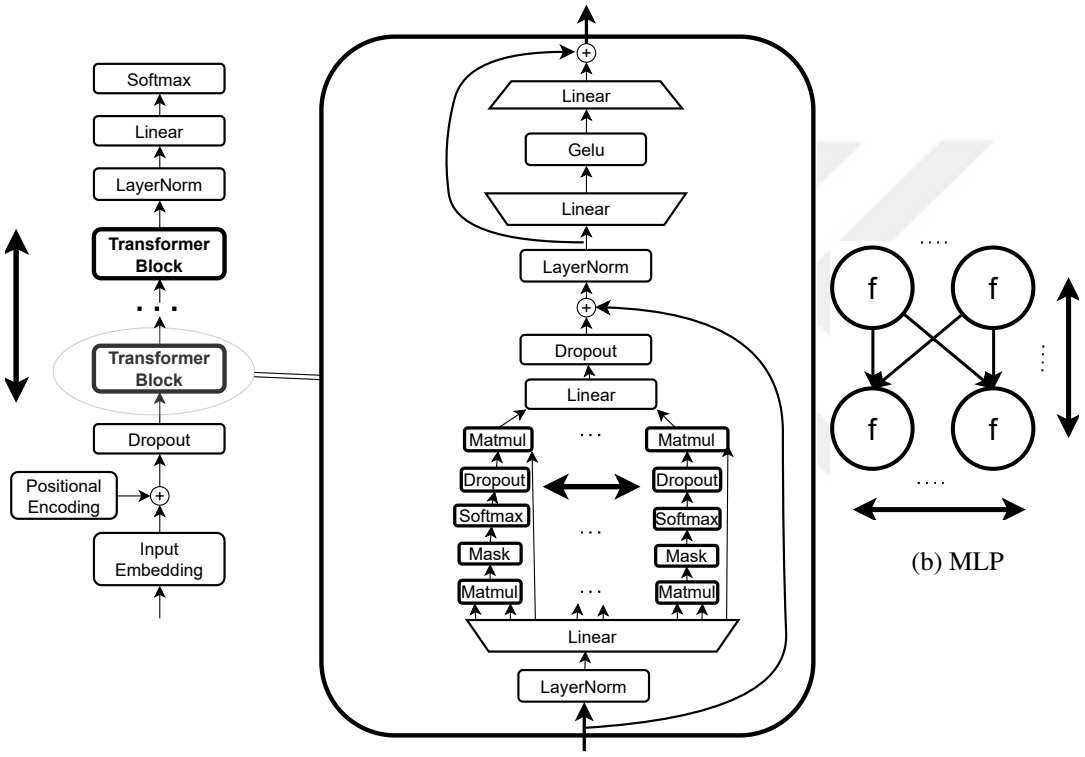
Deep learning (DL) often relies on scaling up models and data for improved performance (Section 2.4). The literature on the progression of DL breakthroughs highlights the importance of learning reusable representations and improving parameter as well as compute efficiency. Despite being large, high-performing DL models are generally simple in terms of architecture. They rely on various forms of repetitions in design (e.g. Figure 1.1), leading to feature and parameter sharing. How do different design choices in terms of such repetitions, i.e. reusability, affect performance? In this work, this question is analyzed by introducing various new approaches.

1.1.1 What is reusability?

Reusability is a term borrowed from software engineering and computer science. It is about utilizing existing solutions for new problems. In the context of this thesis work, reusability analogously refers to the level of repetition for learnable components that are shared either directly via tying weights (e.g. cross-layer parameter sharing) or indirectly by using the hidden outputs (e.g. feature reuse) in multiple places.

In Figure 1.1a, within each transformer block the given two skip connections essentially reuse the hidden features. For instance, the *LayerNorm* output is used as an

input for the second last *Linear* layer. After a nonlinearity (*Gelu*) the LayerNorm output is also summed with the output from the last Linear layer. This is an example of feature reuse (e.g. [12] and [13]). In the same figure, if the architecturally equivalent Transformer blocks have their weights tied to the same learnable parameters, this would be an example of parameter sharing (e.g. [3] and).



(a) Transformer blocks

(b) MLP

Figure 1.1: Scaling up deep learning models vertically and horizontally (bold arrows) leads to repetitions. In larger language models (e.g. [2]), transformer blocks are repeated for increased depth and attention heads are repeated for increased width (a). Note that architectural repetitions can be accompanied with cross layer parameter sharing (e.g. [3]). Increasing the width of multilayer perceptron (MLP) leads to the output from each node to be used in more nodes (b).

1.2 Proposed Methods and Models

To better understand the question of how reusing components affect performance, first, a new library, Revolver, is introduced for searching reusable modules while simultaneously training populations. After Revolver, a graph analysis-based framework is introduced for comparing or ranking DL models *without any training*. Finally, alongside a theoretical understanding and relevant proofs, an energy-based generalization is introduced in terms of comparing models.

1.2.1 Searching for Reusable Modules with Revolver

Revolver is a new library that relies on model blueprints to describe modules as well as entire models in a hierarchical manner. In the library, a genetic algorithm that can treat the blueprints as genotypes is provided alongside a random search approach. The blueprints include scope names to be able to share modules within and between models during training. Parameter sharing in this manner allows hyperparameter search while simultaneously training a fairly large population of DL models (e.g. 80 models) using only a single GPU. This approach keeps track of the fitness of not only the top individuals, but also the modules in the population.

A compact genetic toolbox may have evolved for the nervous system by reusing older genes for new purposes (co-option) as discussed in Section 3.2.2.1. Analogously, Revolver is implemented to reuse DL model components or modules in various combinations within and across different models in a population. Alongside a blueprinting approach that allows parameter sharing, other optimizations exist such as compute budget assigned based on performance and genotype fitness that is assigned before any training as a crude approximation of model performance.

Revolver relies on a search pipeline (Section 3.4.1.1). It provides two search algorithms: random search, and genetic algorithm. Both search algorithms rely on the genetic fitness for comparing models described in Section 3.5.5.2. The random search

algorithm picks width and depths from an allowed range for different groups of components to sample new models. When tested on the CIFAR-10 benchmark [14], the genetic algorithm consistently performed better than the random search algorithm in multiple experiments (Table 3.2).

1.2.2 Comparing DL Models Without Training

All mainstream DL models already share parameters directly or indirectly to various degrees, and this affects their parameter efficiency. In A.2, a simple horizontal unrolling approach is introduced for making parameter sharing explicit for all directed acyclic graphs (DAG).

With the understanding that DL models always share parameters, this work aims to disentangle the intrinsic design and training choices that change how parameters are shared and how a model's performance is affected. For this purpose, the reusability prior is introduced as follows: *model components are forced to function in diverse contexts not only due to the training data, augmentation, and regularization choices, but also due to the model design itself.*

Experiments conducted with cross-layer parameter sharing provided supporting evidence for the viability of the reusability prior. For instance, some models that have cross-layer parameter sharing outperformed the baselines that have at least 60% more parameters. The graph analysis-based method correctly ranked these models higher in terms of the estimated performance than baseline models with more parameters. The published work is available here [15]. To derive model-level quantities and predict performance, the graph analysis approach introduces a precisely defined notion, a *context*.

1.2.3 What is a Context?

Informally, a *context* is a path connecting an output node and a model parameter represented as weight and associated with a given hidden input at a given location within a computational graph of a model. Figure 4.1 depicts what a context is, there is also a formal definition given in Section 4.1.2.

A simple counting approach of contexts for each learnable parameters is used for model comparison. By treating relative frequencies derived from the number of all possible contexts per parameter as a probability distribution, it becomes possible to define quantities for the comparison of model graphs. Performance estimation approaches are also introduced for comparing models without training.

1.2.4 Major Results from the Experiments

Multiple EfficientNetv2 [1] models were trained to analyze the effect of aggressive cross layer parameter sharing strategy. This strategy relies on roughly treating weight matrices of the same shape as the same. In the experiments, a consistent increase in the parameter efficiency was observed for the V2-S models with aggressive parameter sharing compared to V2-B0 which has significantly more parameters, as given in Table 4.2. Overall, at the cost of additional FLOPs, the parameter efficiency was improved for all three benchmarks.

Then graph analysis based experiments were conducted without relying on any training. When training the EfficientNetv2 models, V2-S-shared models consistently performed better than the V2-B0 models, despite having a significantly lower number of parameters. In Table 4.3 the results from the Imagenet-1K experiments are compared with the results from graph-based analysis that relied on counting the number of contexts for each learnable parameter using the computational graphs of V2-B0 and V2-S models with and without parameter sharing. Unlike a naive prediction which would directly correlate the number of parameters with performance, the graph anal-

ysis assigned a higher score for the *V2-S-shared* model. Furthermore, in Table 4.4 the estimated performance and expected spread of ResNet is lower, while its number of parameters are significantly higher than V2-B3 and V2-S models. Overall, the graph analysis method was able to correctly rank all compared models in terms of performance without relying on any training, while the naive approach based on the number of learnable parameters failed to do so.

1.2.5 Encoding New Constraints with Energy Priors

There is an analogy between the idea of contexts and the idea of microstates from statistical mechanics. This analogy leads to a more generalized framework for model comparison. Instead of the reusability prior, new priors can be integrated by assigning energy to each learnable parameter in a given model.

Counting the number of contexts is a methodology that relies on the reusability prior, and ignores all hardware and practical constraints. To come up with a more generalized framework, a statistical mechanics-based framework is introduced for model comparison. This framework can still express the reusability prior, but it introduces additional advantages. For instance, it is possible to predict the performance behavior of multilayer perceptrons (MLP) with constant width *without relying on any graph analysis* (Section 5.3.5.5). Another advantage is that this allows the understanding and tools of statistical mechanics to be adapted for model comparison. Finally, this framework brings additional flexibility in representation. For instance defining a more arbitrary energy prior is possible by encoding relevant constraints into the energy and how the probabilities for each learnable parameter is defined (Section 5.4).

1.3 Contributions and Novelties

Overall, the major contributions of this thesis work are threefold:

- A new library, Revolver, for analyzing the reusability of modules while simultaneously training a population for hyperparameter search is provided. The GitHub repo for Revolver is available here: `https://github.com/gozepolat/revolver`. Revolver may be extended or directly used by researchers for hyperparameter or neural architecture search, as well as analyzing which shared modules are more reusable across models for a given benchmark.

- The reusability prior, a generalized notion of reusability that ties training, design, and data aspects together, alongside a new methodology based on graph analysis for deriving model-level quantities for comparison is proposed. This work is published in [15]. The code to reproduce the analysis results is available at [16] and the GitHub repo at: `https://github.com/gozepolat/priors/tree/main/reusability`. This may lead to further research and important predictions on how deep learning models may be affected by different design, augmentation, and training choices.

- By relying on statistical mechanics, a new theoretical framework for energy-based priors and relevant proofs for a generalization of the model comparison approach are introduced. This framework may open up new avenues of research for the quantifiable analysis and comparison of DL models with different practical constraints and assumptions in mind.

1.4 The Outline of the Thesis

The relevant background on the progression of DL breakthroughs and the importance of learning reusable representations as well as improving parameter and compute efficiency is given in Chapter 2. In Chapter 3, the major algorithms and experiment results for Revolver are given. Then, alongside supporting empirical evidence, the reusability prior and the methodology for comparing models without training are introduced in Chapter 4. After an analysis of how this methodology behaves in the extremes (Section 5.2, in Chapter 5, a theoretical framework for energy-based pri-

ors and relevant proofs for a generalization of the model comparison approach from Chapter 4 are introduced. Finally, in Chapter 6 a summary, major outcomes, and conclusions of this thesis work are given.

CHAPTER 2

BACKGROUND

2.1 Introduction

Minsky describes the *fundamental credit assignment problem* as the question of what components determine success and failure to improve a system with modifiable components [17]. From the perspective of machine learning (ML), these components mainly correspond to representation, evaluation, and optimization. In the last decade, research on deep learning (DL) revealed that achieving a good model-level representation of the data is a critical problem.

According to Bengio et al., the success of a machine learning algorithm depends on the question of whether it can create a good representation of the data and take advantage of it or not [18]. Furthermore, the number of training samples and learnable parameters are highly relevant for model performance and seem to go hand in hand [19].

2.1.1 What is Deep Learning?

Schmidhuber distinguishes deep and shallow architectures according to the depth of their *credit assignment paths* (CAP) [20]. CAP is a chain of possibly learnable causal links between actions and effects. The smallest depth for any credit assignment path corresponds to the problem's depth. A deep architecture has multiple levels of non-

linear operations (i.e. depth > 2). Shallow architectures can still be used as universal approximators, (i.e. they can approximate any function). Yet decreasing the depth can result in an exponential increase in the size of a model as in deep circuits in [21].

Delalleau and Bengio define deep architectures as function families that are equivalent to deep circuits [22]. Even though increasing the depth can reduce the size of a model, training a deep architecture without additional measures can lead to poor performance. For example, the backpropagation algorithm [23, 24, 25] with random initialization of weights (that is commonly used for shallow feed-forward networks) can have a poor performance in deep architectures, because increased depth creates sharp nonlinearities and this results in the exploding or vanishing gradient problem described in [26].

Deep learning (DL) comprises machine learning techniques that take advantage of learning multiple levels of abstraction hierarchically to efficiently train deep architectures [27]. Currently, DL models have state-of-the-art performance in diverse areas such as computer vision [28, 29, 30, 31], natural language processing [2], and protein folding [32].

In this chapter, after describing the early research and fundamental ideas that led to the DL paradigm shift (Section 2.2), the literature on deep representations and representation learning is investigated (Section 2.3). The progression of approaches shows that there is a clear pattern in DL advancements that relied on scaling up model and data size. The main conclusion is that as model size and training data continue growing (Section 2.4), learning reusable representations and improving compute efficiency as well as sample and parameter efficiency will stay relevant if not become increasingly important.

2.2 Early Research

The artificial neuron model suggested by McCulloch and Pitts in 1943 [33] and Rosenblatt's perceptron learning algorithm in 1958 [34] sparked researchers' interest in learning models. Backpropagation algorithm in 1960s and 1970s [23, 24, 25], Neocognitron in 1980 [35], Hopfield networks [36], principal component analysis (PCA) [37], and self-organizing maps (SOM) [38] in 1982, recurrent neural networks (RNN) and Boltzmann machines in 1985 [39], then in 1986 the restricted Boltzmann machines (RBM) [40] and multilayer perceptrons (MLP) [41], after that in 1988 radial basis function (RBF) networks [42], autoencoders in 1989 [43], nonlinear generalization of PCA by Kramer in 1991 [44], sigmoid belief networks in 1992 [45], support vector machines (SVM) in 1995[46], sparse coding in 1996 [47], and in 1997 long short term memory networks (LSTM) [48] can be considered among some of the earlier research. Aside from being the first deep learning example, Neocognitron introduced the notions for convolutional neural networks (CNN) in 1980 [35], then LeCun et al. demonstrated the convolutional layer's potential in their LeNet models [49, 50], and in 2003 Benkhe introduced a generalization [51].

2.2.1 Neocognitron

Fukushima proposed the very first deep learning architecture in history in 1979 [52], and a year later he introduced Neocognitron, which is designed for visual pattern recognition [35]. His model was bioinspired by the visual system and its hierarchy model suggested by Hubel and Wiesel [53]. In his study, several precursory research ideas were applied. For instance, he introduced a CNN-like model. He used unsupervised learning to train a deep architecture in which abstract features that are robust against noise emerged. He described the unsupervised training of the network to acquire translation invariance: "The network is self-organized by 'learning without a teacher', and acquires an ability to recognize stimulus patterns based on the geometrical similarity (Gestalt) of their shapes without affected by their positions."[52]

2.2.2 Recognizing Digits with Backpropagation and LeNet

In 1989, LeCun et al. demonstrated a model that can recognize digits by using the backpropagation algorithm on a model that can directly learn from normalized digit images instead of feature-engineered input [49]. They constrained the network weights and architecture by hand, according to the digit recognition domain, to achieve more efficient training.

A relevant example for CNNs was given by LeCun et al. in 1998 [50]. After a review and comparison of relevant research on handwritten character recognition, they used their CNN model, LeNet-5, to demonstrate that it could achieve state-of-the-art performance, giving better results than all the other reviewed models. Unlike the traditional shallow models which used a classifier on top of a single feature extractor layer, LeNet-5 consisted of several subsequent convolution and subsampling layers, effectively creating a deep architecture. Their research contained important conclusions about feature learning. For instance, they presented the idea that, instead of early segmentation of objects in images using features found by experts (i.e. feature extraction as a fixed transform), a heuristic algorithm can be used in a data-driven way for each layer (i.e. *representation learning instead* of feature engineering).

2.2.3 A Paradigm Shift

The idea of taking advantage of large amounts of data with unsupervised training to learn representations and to construct a deep initial representation was present in early research. For instance, the very first example of deep learning was around for decades [52]. However, the explosion of deep learning research was after the suggestion of practical techniques that performed particularly well.

2.2.3.1 Pretraining and Stacking Layers

In 2006 Hinton and Salakhutdinov described an efficient way to train deep autoencoders [54]. An autoencoder is an MLP that is trained to reconstruct its input. It has encoder and decoder parts. The encoder transforms the input and the decoder decodes the transformed representation back to the input. By introducing constraints in the loss function or nonlinearities right after the encoding, an encoder can learn the representation with desired property using backpropagation. Hinton and Salakhutdinov first pointed out that gradient descent performed well only if there was already a good solution close to the initialized weights. Then they described pretraining, a way to initialize weights in a more sensible way than random assignment. In pretraining, a stack of restricted Boltzmann machines (RBM) (i.e. each with a single layer of feature detectors) was trained. The output of one trained RBM was used as an input for the next RBM for training again. After all the RBMs were trained, they were unrolled to construct a deep autoencoder with the same weights. Finally, the stochastic activities were converted to real probabilistic values and fine-tuned with the backpropagation algorithm. This procedure effectively created a deep autoencoder. This method of dimensionality reduction outperformed PCA [55]. Hinton et al. 's greedy algorithm of layerwise pretraining described an alternative way to achieve better and better layerwise representations of the input for Deep Belief Networks (DBN) [56]. Shortly after Hinton, Ranzato et al. came up with an energy-based model that can learn sparse and overcomplete representations via unsupervised learning [57]. Sparsity was previously ensured with an additional loss term which would require a normalization procedure [58]. Ranzato et al. claimed that adding a nonlinearity between the encoder and decoder also helped in learning sparse representations [59]. The same year, Bengio et al. pointed out that Hinton et al.'s strategy of layerwise transformation of the input to achieve better internal representations can be applied beyond DBNs [60].

2.2.3.2 Emergence of Simpler Activation Functions

In 2010, Glorot and Bengio showed that it was easier to train deep MLPs with alternative ways of initialization and more suitable activation functions [61]. Compared to the greedy layerwise pretraining algorithm that was proposed in 2006 [56], directly applying gradient descent algorithms to a randomly initialized feedforward neural network used to give poor results. To better understand this problem, Glorot and Bengio analyzed the standard gradient descent on randomly initialized deep neural networks [61]. Their first observation was that the logistic sigmoid function was not a good activation function for deep networks with random initialization. It turned out that, with the logistic sigmoid function, the mean value of a random initialization would saturate the top hidden layers (i.e. vanishing gradient problem [26]. Introducing another type of nonlinearity that would not saturate would improve the performance of the gradient descent algorithm on a randomly initialized deep neural network. Indeed Nair et al. showed that a rectified linear unit (ReLU) was such a type of nonlinearity [62]. Overall, these developments and their clear practical success in the benchmarks resulted in a high amount of interest and new research in DL. Finally, DL research and applied focus accelerated after Krizhevsky et al.'s iconic success with a CNN model, AlexNet, in the ImageNet large-scale visual recognition challenge (ILSVRC) in 2012 [63].

2.3 Deep Representations

Hinton discusses why there must be a basic learning algorithm for feature extraction that can be applied to "richly structured high-dimensional sensory data" [64]. He believes that basic, uniform architecture in the cortex and its high adaptability and plasticity to early damage supports the idea that there is such an algorithm. He then talks about strategies that can be combined to create algorithms that are suitable for deep architectures with millions of connections similar to the cortex. For another perspective on why learning deep representations work, an analysis of what features

unsupervised deep learning captures and how such representations emerge given by Paul and Venkatasubramanian can be relevant [65]. They make use of group theory to create a framework to analyze layerwise pretraining and explain why invariant features emerge within the higher layers. However, their framework can not be used for the supervised case. Supporting the idea from [21], Bengio and LeCun provide mathematical and empirical evidence that nonparametric learners such as kernel methods can not always efficiently learn high dimensional functions [66]. This, they argue, is because kernel machines are shallow architectures since there is one large layer of simple template matchers. Having only a single layer of trainable coefficients and shallow architectures can be very inefficient due to the number of computational elements and examples required. Empirical results in their study, where kernel methods are compared with deep architectures, show that deep architectures can capture features beyond immediate neighbors; therefore, it is easier to capture abstract features with deep architectures. Deep architectures with a width constraint can still work well. When exponentially deep, even if limited by the width of the input layer, sigmoid belief networks, RBMs and DBNs can all approximate any distribution over binary vectors to arbitrary accuracy [67]. The required number of parameters to approximate any distributions in RBMs is equal to that of DBNs. In another relevant study, Roux and Bengio show that increasing the number of units in RBMs always improves the modeling power and RBMs can approximate any discrete distribution as long as the number of hidden units can be increased [68].

2.3.1 Learning Input Representation

In their greedy layerwise pretraining approach, Hinton et al. first start with training an RBM and use the hidden layer which learned a set of features from the input as the visible layer of another RBM, and so on. After that, they stack each trained hidden layer to construct a deep network. If the backward-directed weights in the final network are not removed, the resulting network is a DBM and it can be used as a classifier. Training the DBM with the label units results in learning a joint distribution

for labels and inputs. Thus a DBN can be used for reconstructing the input together with a label. To achieve that, label units can be added to the layer before the final hidden layer (i.e. add the label units to the visible layer of the top RBM). An analysis of deep neural networks suggests that slight changes in a test case, which will be imperceptible to the human visual system, can have big effects on a deep architecture, indicating that fairly discontinuous input-output mappings can be learned by them. It is claimed that such slight changes with dramatic effects can be found for virtually every test case (i.e. adversarial examples) [69]. Another related observation made by Nguyen et al. is that the reverse is also true. That is, the images unrecognizable to humans (e.g. white noise) can be "recognized" (e.g. lion) by the models with very high confidence [70]. Currently, it appears that even if a deep neural network and the human visual system have certain similarities, the success of current models should not be taken as a clear indication of human-like representation capabilities. Goodfellow et al. propose generative adversarial nets (GAN) [71]. GANs rely on training a generative model that learns the input distribution and a discriminative model that classifies whether the given input is from the generative model or a real sample. Yet GANs do not necessarily learn the true input distribution, rather the generator learns to create samples that seem realistic to the discriminator model.

2.3.2 Learning Embeddings

A manifold is a topological space equipped with an atlas of local coordinate systems (i.e. charts). In a small neighborhood around every point, it behaves like a Euclidean space [72]. Latent factors of variation in images such as pose, facial expression, and morphology are abstract features that can be important to disentangle. Reed et al. propose to disentangle such factors of variations by learning their manifold coordinates and modeling their joint interaction [73]. Their proposed model is a higher-order Boltzmann machine where hidden unit groups have multiplicative interactions and each group learns to encode a different latent factor. Embeddings are relevant since it is hypothesized that deep learning models stretch and deform manifolds to disentan-

gle abstract factors. Therefore good embeddings of different factors are supposed to be easy to separate by a classifier. Salakhutdinov and Hinton show a way to transform the input space into a low-dimensional feature space suitable for K-nearest neighbor classification. Before fine-tuning, their method has a pretraining phase that can make use of the unlabeled data to come up with a better nonlinear transformation, therefore, it performs well even if the labeled data is limited [74].

2.3.3 Residual models

Srivastava et al. introduce Highway Networks [75] that incorporate information highways to make training easier in a similar manner to long short-term memory networks (LSTM) [48] by regulating the information flow from lower layers. Residual neural networks (ResNet) [12], demonstrate the practical benefits of reusing features via skip connections instead, which lead to a state-of-the-art performance in several computer vision benchmarks such as CIFAR-10 [14], CIFAR-100 [76], ImageNet-1K [77], Pascal VOC [78], and COCO [79]. ResNet variants [12, 13, 80, 81, 82], transformer-based models [83, 84, 85, 86], and more recent MLP based variants without attention modules [87, 88, 89] *all* share features (i.e. outputs from previous modules or components) via identity connections, also called skip connections or shortcuts. These models use *residual blocks* which consist of multiple layers of varying types, and a direct shortcut from the input of the block to the output of the block's final layer for a summation operation. Skip connections allow ResNet variants to learn a simpler residual function instead of the underlying representation that is being approximated [12]. Combined with the vertical dropout technique suggested for stochastic depth [90], ResNet architectures deeper than 1200 layers can be trained without being significantly affected by diminishing gradients and high training error. Moreover, faster training is achieved because vertical dropout bypasses large chunks of layers, in effect, allowing an effective stochastic depth in training time that is much shorter than the original network that would be used in test time. In practice, most modern architectures rely on skip connections.

2.3.3.1 Regularization and Architectural Priors

Aside from lower stochastic depth, architectural priors and regularization mechanisms against overfitting are used in classification tasks [91, 92, 93, 94]. For instance, the *join layers* in FractalNet which take the average of incoming paths or fractals of different depth, as well as the *path drop* method used in training alleviate overfitting related issues and allow training a fractal-like CNN architecture with a large number of parameters. Moreover, averaging over alternative paths of various depths has the additional benefit of reducing the *effective depth* [92]. Inspired by the path drop method, shake-shake regularization [93] used the weighted average of residual blocks with stochastic softmax weights. Later, Devries and Taylor introduced an augmentation technique called *cutout* that randomly masks square portions of the input image, after normalizing the dataset (i.e. per channel mean subtraction) [94].

2.3.3.2 Equivalence of ResNet Flavors

Alongside the fact that vertical dropout works well with ResNet and the observation that the visual cortex does not require 1000+ layers, Liao and Poggio draw attention to the similarities between ResNet and RNNs. They frame ResNet as a *homogeneous dynamical system*. Then they show that, with *parameter sharing*, when posed as a *time invariant* system, the recurrent form looks like an unrolled multi-state RNN, which gives 14% test error for CIFAR-10 with a dramatically reduced number of parameters (40K only). After a series of experiments inspired by the recurrent connections observed in the visual cortex, they conjecture that "the ability to approximate recurrent computations" may have an important role in the success of deep feedforward architectures [95]. Chen et al. proposed Dual Path Networks (DPN) [96], where they also pointed out that topologically, ResNet and many of its flavors were special cases of DenseNet, and when the blocks are reused, they could be represented as Higher Order RNN (HORNN) [97].

2.3.3.3 Implicit models

Deep learning models often transform features at each explicitly defined layer based on a given architecture. For ResNets, which consist of architecturally repetitive residual blocks, this transformation can be similar to RNNs with cross-layer parameter sharing, as discussed in [95]. An alternative strategy to cross-layer parameter sharing is the usage of implicit models. Well-known examples are neural ordinary differential equation solvers (ODEs) [98, 99], variants of deep equilibrium models (DEQ), and multiscale DEQs (MDEQ) [8, 9]. By satisfying an analytical condition, these models replace the discrete notion of depth with a continuous one. Similar to cross-layer parameter sharing, with a constant number of parameters, it is possible to simulate a model that is arbitrarily deep. That is, implicit models do not rely on an architecture with a set number of layers. A similar but discrete outcome can be explicitly achieved by relying on cross-layer parameter sharing, where arbitrarily deep models can be constructed by repeating the same weights in multiple layers so that the number of parameters stays constant while the effective depth grows.

2.3.4 Transfer learning

Deep learning techniques transform input representation into more relevant forms after a series of nonlinear operations. A portion of learned operations is often *reusable* or easily adaptable for other purposes in various contexts. For instance in models such as VGGNet [100], AlexNet [63], and GoogLeNet [101], the initial layers emerge as edge detectors. This makes them useful for a wide range of computer vision tasks. Moreover, there is an efficiency (i.e. lower number of parameters) and accuracy boost that comes with simultaneously learning embeddings of different domains. For instance learning embeddings of images of faces, objects, cats, digits, and sketches, all at once, as in [102] is more efficient than learning each embedding separately. Transfer learning is tightly related to the notion of reusability. To illustrate, Bilen and Vedaldi show that a high degree of parameter sharing between various visual classifi-

cation tasks can reduce the average error rate of all tasks [102]. In particular, in their *deep sharing experiment* where all layers but the final one are shared for ten different classification tasks, they observe a 1% improvement on the average error rate, where the number of parameters is significantly decreased (1/10th of the original number of parameters). Learning embeddings for a wide range of objectives such as word, sentence, document, or multi-relational graph representation, text classification, ranking, and collaborative filtering [103] is another example where the learned modules are efficiently reused. Fernando et al. trained PathNet, by training a *pathway* on one task and freezing and reusing portions of it for another [104]. Reusing parts of a very large network in such a manner allowed training various tasks on the same network faster than training a different network for each task from scratch. Another example is Finn et al.'s work on meta-learning. They proposed a model agnostic algorithm to learn training models that are "easy to fine-tune", making it easier to later reuse the learned representations for new tasks [105]. Anderson et al. have a related approach where portions of trained networks are treated as modules [106]. Similar to the progressive networks [107], they increase the size of the network by adding new modules to the pretrained ones, *while freezing the weights*. They show that this surpasses simple transfer learning approaches that only rely on fine-tuning. Unlike progressive networks, they focus on learning from small data on object classification and sentiment analysis tasks by stitching together the pretrained and new modules. To stitch together the existing modules with the new ones, Anderson et al. insert a small network between each existing layer. This way, the new modules are "interwoven with the original network" [106]. Overall transfer learning allows reusing portions of large models for other settings. This is achieved because models learn deep representations abstract enough for other problems as well.

2.3.5 Learning Equivariant and Invariant Representations

A CNN is in essence a bioinspired design of MLP for visual input. Each channel in CNN corresponds to a neuron in MLP. The kernels repeat spatially for each chan-

nel. This type of parameter sharing is suitable for the input of different sizes, and input data that has spatial consistency. There are early examples of using CNNs as feature extractors. For instance, Ranzato et al. propose a feature extractor that consists of convolution filters, sigmoid nonlinearity, and finally, a feature max-pooling layer [108]. Using two feature extractors in a sequence by giving the features obtained from the first extractor to the second as an input, they combine a layer-wise unsupervised training approach with CNNs. Their method is suited for problems with very few labeled training samples as CNNs often have a significantly lower number of learnable parameters compared to MLPs. CNNs have translation equivariance. That is, the neighborhood information is preserved after a convolutional layer, and distortions to the input in terms of translation cause the output to vary equally. Invariance instead maps all distortions into the same output. When the input domain is not necessarily Euclidean, some symmetries can be learned with graph neural networks (GNN) [109] by relying on the principles of group theory (i.e. geometric deep learning) [110]. There are multiple approaches to learning deep representations based on group theory. For instance, Thiede et al. provide a general theory for the permutation equivariant neural networks [111]. In general, with a low number of parameters, domain-specific symmetries can be captured with various approaches as in [112, 113, 114, 115, 116, 117].

2.3.6 Learning to Attend

Badhanau et al. 's work of jointly learning to align and translate text describes a type of attention layer that is later referred to as "Badhanau attention" [118]. An encoder-decoder architecture that utilizes this type of attention layer can keep the encoder output verbose, as the output dimensionality from the encoder no longer needs to be fixed. The attention layer in the decoder would assign weights to the encoder outputs at each time step. Spatial transformer networks (STN) [119] learn to generate a transformation matrix from the input to sample and transform the input features. Transformer architectures [83], generalizes a similar notion to "Bahdanau attention"

for *self-attention* layers, which linearly transform the current input to *key, value*, and *query*, and elementwise multiply *value* with the softmax of matrix multiplication of key and query. In other words, self-attention creates a similarity matrix for key and query, then multiplies this matrix with value. That is, for each item in a sequence, there is a corresponding row in the similarity matrix that describes how similar the current item is to the rest of the items in the sequence. Transformer architectures currently hold the state-of-the-art in natural language processing (NLP) [2], protein folding [32], and computer vision [120].

2.3.7 Representation Learning

Learning features can be achieved in an unsupervised, supervised, or semi-supervised way. However, since unlabeled data is easily available, unsupervised learning is commonly used for representation learning, thus representation learning is also often referred to as "unsupervised feature learning" [18]. Rooyen and Williamson suggest a theoretical framework about when it is possible to learn a feature in an unsupervised manner [121]. They describe a method for representation learning that is not sensitive to loss. They claim that learning generic features boil down to a *manifold assumption*, that is, it is possible to concentrate high dimensional data into a lower dimensional space, thanks to the low variability of data. Moreover, they suggest using a lossy data compression scheme and its generalizations to evaluate the quality of features. Representation learning is closely connected to DL because learning a deep representation where abstract features emerge at higher levels of a feature hierarchy is often desired. This is because abstract features are often invariant to low-level changes and robust against noise. For instance, multimodal representations [122], embeddings [123], and distributed representations [124] are often achieved with DL.

2.3.7.1 Biological Inspiration

Olshausen and Field connect the biological description of "localized, oriented and bandpass" spatial receptive fields for simple cells in the visual system with basis functions of wavelet transforms and *sparse coding* [125]. Their proposed theory is based on the assumption that the visual system can create an efficient representation of natural input by extracting statistically independent structures in images. They reason that, since sparse coding can reduce the number of elements with statistical dependencies in the representation, it can create an efficient representation. Moreover, the receptive fields that emerge from their algorithm are fairly similar to the ones observed in the primary visual cortex. Finally, they predict that their ideas can be used hierarchically to construct more elaborate models (that can potentially explain higher regions of the visual system). Similarly inspired by the mammalian visual system, Anselmi et al. later describe representation learning as an unsupervised acquisition of invariant representations that can *lower* the sample complexity for the supervised learning stage [126]. The sample complexity corresponds to how easy it is to differentiate between samples. For instance, rectifying images of objects of different orientations reduces sample complexity, and makes it easier to categorize them [127, 128].

2.3.7.2 Priors of Representation

In his overview of unsupervised feature learning techniques, Bengio gives an analysis of priors for representation [18]. For example, multiple explanatory factors (or distributed representations) allow the representation to achieve high expressiveness, hierarchical organization of explanatory factors allows both learning invariant features and disentangling latent variables, shared factors across tasks allow learned parameters to be reusable, natural clustering or well separated $P(X|Y = i)$ for different i allows achieving easier classification, manifolds allow high dimensional natural data (that in general has a much lower variation than what is possible) to be mapped into a much lower dimensionality, and sparsity allows variably sized representations as well

as reducing the number of relevant factors for a given sample.

2.4 Scaling Up

Apparent benefits of deep learning and representation learning are the ability to extract features directly from the data without any feature engineering and the ability to utilize large amounts of data for performance gain. As DL models get larger, an improved sample efficiency may introduce a computational benefit as well. Furthermore, since there is an exponential increase in the size of unlabeled data every year [129], the task of utilizing, labeling, or curating big data for training can be important for scaling up. DL and representation learning approaches transform the input representation into a more useful form. This enables working with a very high dimensional input. Hence, it becomes possible to directly take advantage of big data. Scaling up is useful for three major reasons:

1. The higher the number of useful extracted features is, the higher accuracy a model will achieve, independent of the algorithm [130].

2. Accuracy of the learners can continue to improve as the training set gets larger [131].

3. Accuracy of DL models can grow most optimally if the size of the training set is properly increased alongside the model size [19].

2.4.1 Increasing Model Size

Increasing depth has its unique problems such as overfitting, high training error, and diminishing feature reuse [12, 132]. When such adverse effects are avoided, however, increased model capacity often improves the test accuracy until the training data size becomes the next bottleneck. Aside from well-known techniques such as dropout

[133], batch normalization [134], weight normalization [135], and layer normalization [136], there are examples of techniques that enable training larger networks in CNNs. These techniques focus on avoiding overfitting and high training errors. For instance Highway Networks [75], ResNet [12], FractalNet [92], and DenseNet [13] utilize connections that skip layers, reducing the effective training depth. DenseNet models are claimed to be *compact* due to the concatenation of previous features at each subsequent layer. Models can have a lower number of parameters despite having a larger architecture via forms of parameter sharing. A strategy that relies on depthwise separable convolutions for compactness used by Xception, or "Extreme Inception" [137] with a faster training time than the Inception architecture [101] while outperforming it on ILSVRC [138]. Inspired by the depthwise separable convolutions used in Xception, MobileNet [139] focuses on maximizing the 1x1 convolution operations, which corresponds to 95 % of all operations in the model. Note that 1x1 convolutions are equivalent to dense layers that are shared for the depth and width dimensions (e.g. shared for each pixel for the input features). Achieving compactness via soft sharing [140], student teacher architectures [141, 142], pruning and quantization [143, 144] are alternative approaches. Very large models achieve state-of-the-art in computer vision [145, 146, 147, 148, 149, 150, 28, 151, 152, 29, 30, 31] and natural language processing [153, 154, 155, 156, 157, 2]. In theory, shallow models could match the performance of their deeper counterparts if the size and compute limitations were irrelevant. Coates et al. give an analysis of the changes to receptive field size, number of hidden nodes (features), step size (stride) between features and the effect of whitening [130]. They claim that reducing the step size (stride) while increasing the number of features can monotonically improve the accuracy, even though increasing the receptive field size can have negative effects since a larger receptive field can require more parameters to learn and more data to train [130]. Note that deeper models are not always desirable, or even compute-efficient, and this leads to a trade-off between the depth and width of state-of-the-art models. Furthermore, Levine et al. theoretically predict that there is an "optimal depth-to-width allocation for a given self-attention network size". They recommend significantly wider models

as the model size increases and passes a certain threshold [158].

2.4.2 Increasing Data Size

Before DL, feature engineering was commonly used as a way to compensate for the lack of automated mechanisms to transform the data into a form that is easier to disentangle. Now, however, efficiently learning good data representations without any feature engineering is common, especially for the types of problems where data are abundant [18]. Representation learning consists of techniques that allow learning transformations of the data to extract useful features more efficiently [18]. Representation learning is useful in the application areas such as computer vision, audio, speech, and natural language processing, robotics as well as neuroscience for various purposes such as classification, segmentation, compression, and denoising. Moreover, the application areas continue to grow, since representation learning can be applied to any area where there is a large amount of unlabeled data. Erhan et al. believe that unsupervised pretraining plays a role in optimization and regularization [159]. They reason that at the beginning of training with stochastic gradient descent (SGD), it is more probable for the weight changes to be larger and more dramatic compared to the later phases, in which the trajectory of SGD will follow a path trapped in a much smaller region and eventually towards a basin of attraction. Even though this late behavior is desired for finding a minimum, this introduces an imbalance regarding the influence of each example in the training set. That is, the main trajectory of SGD and the final basin of attraction heavily depend on initial parameters and early training examples. As they show, dramatic effects early on can cause overfitting and the problem can persist even with large amounts of data given later in the training. Thus early training often has a powerful and disproportionate effect on the trajectory of SGD. They believe that initializing weights via pretraining or transfer learning is effective because, in the above setting, it can find a good starting point for the SGD and allow the trajectory to move more easily toward a basin of attraction that corresponds to the parameters that proved useful in multiple settings (i.e. more reusable

representations).

2.5 Conclusions

In this chapter, some of the background research in DL was highlighted. The early research on theoretical simplifications, and pretraining ideas, as well as scaling up data and model size using practical advancements led to a paradigm shift in ML. Furthermore, modern architectures slowly replaced models such as RNN, DBN, and RBM. Combined with a significant increase in the number of learnable parameters and training samples, more recent concepts such as attention layers in transformers turned out to apply to a diverse set of problems including computer vision, NLP, and protein folding. Transformer-based models such as vision transformers (ViT) still hold the state-of-the-art in many benchmarks in computer vision that previously belonged to CNNs. Yet learning good representations in deep architectures has been necessary for utilizing more data. Therefore pretraining has always stayed relevant. Furthermore, learning multimodal representations, taking advantage of the symmetries in the data (e.g. geometric DL), and other domain-specific simplifications are likely to stay relevant for a long time. Overall, there has been a clear pattern in DL advancements that relied on scaling up model and data size. It is likely that as this trend continues, some areas in DL will require prohibitive amounts of compute resources for significant advancements in performance. To alleviate diminishing returns, learning reusable representations, and improving compute efficiency, as well as sample and parameter efficiency, stay important for DL research.

CHAPTER 3

A SEARCH FOR REUSABLE MODULES

3.1 Introduction

Feature and module reuse can be an important underlying pattern when training compact models or improving parameter efficiency. For instance, convolutional and recurrent neural networks have architectural priors that constrain how trained modules are used through space and time. Implicitly, RNNs reuse recurrent modules through time [48] and CNNs reuse convolutional kernels through space [50]. Furthermore, there are meta-learning and neural architecture search (NAS) related approaches that rely on reusing [160] or inheriting [161] parameters.

A potential outcome of a deeper analysis of parameter sharing in model types is that it can lead to equivalence classes of various architectures. Nontrivial relationships between the complexity of various architectures from the literature can be revealed after specific forms of parameter sharing are applied. This is discussed in more detail in C.1.1.

The major contribution of this chapter is a new Python library: *Revolver*. [1] Revolver is implemented to analyze parameter sharing and search for reusable modules while training a population of models. Revolver can train a relatively large population (e.g. 80 DL models) of models simultaneously by relying on parameter sharing, and it can rank the top reused modules in the population. Revolver allows model components to

[1] The GitHub repository for Revolver is available here: `https://github.com/gozepolat/revolver`

be:

- described hierarchically using a *blueprinting approach* (Section 3.3.1),
- shared within a model (e.g. cross-layer parameter sharing), between models, and across generations in a population by relying on *scopes* (Section 3.3.3),
- modified by search algorithms (Section 3.4).

3.2 Background

In this section, first, the relevant literature on meta-learning and neural architecture search (NAS) is reviewed. Then the background on how the nervous system evolved is discussed. The notion of a reusable genetic toolbox discussed in Section 3.2.2.1 points towards the idea that different combinations of the same genetic code can be reused in new settings. Inspired by this, Revolver treats neural architectures as combinations of similar genetic codes (expressing similar functions), adapted for new purposes in phenotypes (models that are trained).

3.2.1 Meta Learning, Metaheuristics, and NAS

It is possible to search, train, and optimize very large and deep architectures that can achieve state-of-the-art results with little human intervention via meta-learning. Although the idea of meta-learning has been around for several decades [162, 163, 164], traditionally, individual tasks are solved separately and hyperparameter search is left to the researcher or done via grid search or random search. Hyperparameter optimization and meta-learning approaches aim to change this by learning priors and hyperparameters for various sets of tasks [165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176].

There may exist a subset of possible internal representations that can be quickly en-

hanced via the addition of new parameters and/or fine-tuning. There are some hints that compact and reusable representations are such subsets [104, 177, 105]. For instance, Long et al. propose deep adaptation networks (DAN) which explicitly focus on making features more transferable [178]. Kaiser et al. concurrently train a model on a large set of seemingly irrelevant tasks from different modalities such as image captioning, speech recognition, and translation. [177]. The reused modules in the model prove crucial for achieving good performance on tasks with small data, while not significantly affecting the overall performance of tasks with large data. This gives supporting evidence that reusable components have better sample efficiency. Yet the advantage likely disappears with larger training sets.

PathNet, introduced by Fernando et al., allows sharing weights and paths for multitask learning within a giant neural network [104]. Agents embedded in PathNet discover reusable portions of paths that can be accommodated from an existing task for which the path is trained to a new task. Agents can continuously work in parallel, *share and learn how to update parameters* for transfer learning. This allows learning subsequent tasks more easily (i.e. faster convergence and high accuracy achieved with smaller sample size etc.) Similar to convolutional neural fabrics (CNF) [179], modules in PathNet have weighted interconnections. Paths are evolved using a tournament selection-based genetic algorithm. Weights of optimal paths get frozen to avoid catastrophic forgetting. The rest of the network is reinitialized. They report that unless the connections that are not in the path are reinitialized, PathNet does not perform better than fine-tuning. PathNet does not require the whole network to be used during training but the main network which contains all the trained paths can be reused for an exponential number of solutions.

Zoph et al. use meta-learning to *find architectural building blocks* for CNNs, for image classification tasks [165]. After finding a good "cell" architecture using neural architecture search on the CIFAR-10 [14] dataset, they stack together the same cell to achieve competitive performance on ImageNet-1K [77]. To generate model architectures with improved accuracy they utilize an RNN that is trained with reinforcement

learning (RL). Efficient neural architecture search (ENAS) instead uses a globally shared Directed Acyclic Graph (DAG) from which their sampler model creates new architectures by potentially reusing components from DAG that were previously parts of other models [160]. Similar to NAS, the controller (i.e. also an RNN) uses RL. Real et al. start with a smaller NASNet search space, and scale models using an evolutionary algorithm [180]. Their tournament selection-based algorithm relies on the concept of age to prioritize younger individuals in the population.

While meta-learning is an active area of research, the literature on NAS and relevant metaheuristics can have issues with reproducibility, potentially due to the limitations in compute resources [181]. For instance, Li et al. claim that random search with a simple budget scheme can surpass these approaches [182, 183]. Yet NAS [165, 184] combined with *compound model scaling* gives state-of-the-art results with EfficientNet models [185, 1] for various benchmarks [186].

The methodology presented in this chapter bears resemblance to ENAS [160] as well as Lamarckian evolution in neural architecture search proposed by Elsken et al. [161] in terms of *weight inheritance*. The difference is the presented evolutionary algorithm operates on *blueprinted* architectures where the parameter-sharing scheme can hierarchically change alongside model architectures. In this way, parameters can be shared between models, across generations, in the same generation, or even within each model (i.e. cross layer).

Population-based training (PBT) [187] relies on the idea of computational budget for different individuals. It searches for a good evolutionary chain of hyperparameters i.e. a "schedule of hyperparameter settings" and it is used for NLP and deep reinforcement learning. An important difference is that the models do not share parameters, even though it is allowed to overwrite weights as an exploitation step to benefit from information gained from training incrementally.

3.2.2 A Brief History of the Evolution of Nervous System

Some of the ideas in this chapter are inspired by the evolution of the nervous system. A likely scenario discussed in [188] starts with self-replicating molecules [189]. This is followed by the evolution of unicellular organisms, leading to a "compact genetic toolbox used for several simple tasks such as cell metabolism, cell division, chemotaxis, phototaxis, and action potential". An example of the uses of excitability in unicellular organisms is “Paramecium caudatum”. It uses electrical excitability as a steering mechanism. A local voltage change occurs when there are significant physical stimuli, and triggers global cilia movement for a short duration; as a result, the organism changes its direction [190]. The evolution of cell adhesion molecules, extracellular matrix, and inter-cellular activity such as collaboration and competition enable the first multicellular life forms, which is followed by cell specification and the emergence of a primitive sensorimotor system. There are still some organisms (e.g. plankton) that maintain generic cells with phototaxis ability. These cells can collaborate with other cells for slightly different tasks, i.e. pigmented cells exist around the cells with cilia, yet task specification is still minimal [191].

The scenario above continues with refinements to cell-cell interactions, further cell specification, and differentiation between layers of cells. While the outer layer specializes in sensory tasks, motor tasks are achieved by the inner layer(s). Cellular communication is slow and limited, yet cells can start benefiting from action potentials (for electrical synapses and state change mechanisms for other purposes). The benefits would be even more pronounced with the emergence of (electrochemical) synapses. This is plausible because all components are co-opted or *reused* (i.e. action potential first used as a steering mechanism and chemical receptors first used in “chemotaxis” and cell collaboration, now are *reused* to enable electrochemical synapses). Efficient usage of “morphogens”, cell adhesion molecules, and extracellular matrix enables cellular migration, differentiation, and axon guidance. Note that growth cone-like structures and cell adhesion molecules used in cell migration is now *reused* for axon guidance [192, 193]. Adhesion molecules used for neural circuitry

are already mostly evolved in the earlier phases since collaboration and stability are required from the start of the multicellular phase. With the refinement of morphogens, the emergence of segmented body parts becomes possible. Segmented body parts allow morphogenesis and evolution/development of each segment separately, leading to a co-evolving neural circuitry. With the emergence of more elaborate nervous systems and body parts, perception and reaction speeds become key for survival. Evolution of a compact but relatively sophisticated nervous system results in the *domination* of other species (i.e. by an "urbilaterian" ancestor [194], the last common ancestor of all higher animals). Then finally, as in the transition from single-celled organisms to metazoans, a compact set of the genetic toolbox that belonged to the urbilaterian ancestor is *reused* in diverse ways. This is supported by research seeking homologies between insect and vertebrate brains [195].

3.2.2.1 A Reusable and Compact Genetic Toolbox

Animals share a common urbilaterian ancestor and a common genetic toolbox for a primitive nervous system [195]. These common genes are used by both vertebrates and insects in the development and later functions of the nervous system. Pallium or cortex in vertebrates and mushroom bodies in insects have therefore a single evolutionary origin.

It is necessary to have novel proteins and genes to explain the extent of genetic diversity. Yet the diversity of phenotype is achieved via more subtle ways than adding new building blocks in genotype. Novelties may be more in the form of new combinations of the common genes and the timing of their expression in development. This may be because of the nature of a primitive system which gradually increases its complexity to gain new benefits while mostly keeping the previous advantageous structures (since there are no jumps in evolution). Therefore novelties depend on and build upon the ancient toolbox. This is why, in general, ancient genes have more than one functionality. They are refined, well-tested, and used more commonly [188]. In other words,

they are *reusable*. That is, the same functionality can take roles in different *contexts*. This idea is later generalized in Section 4.1.2.

3.3 A Blueprinting Approach

It is possible to describe models with a *blueprinting* approach to share parameters or modify components in runtime. This relies on hierarchically naming all components so that modules that are shared have the same scope name.

The described ideas in this study were implemented in Python [196]. The resulting library, Revolver, has a global dictionary that keeps track of all modules, where the reusable modules are persistently stored even after the current models using them are deleted. A random search algorithm randomly picks and combines building blocks based on a randomly selected model skeleton, or a genetic algorithm with hierarchical mutation, crossover, and copyover operators treats the blueprints as genotypes and modifies them.

3.3.1 What is a Blueprint?

A *blueprint* is a simple dictionary-like object that contains hierarchical information about a scoped object:

- **Scope name:** a string that is described using a prefix, suffix, and a module name that contains the shape of its learnable parameters. For aggressive sharing, prefixes and suffixes can be empty.
- **Constructor information:** contains the constructor type and the arguments to call to create a scoped object from scratch.
- **Children:** *blueprints* of all elements or submodules (e.g. for instance for a residual block, the children would consist of batch normalization, activation,

optionally regularization, and convolution layers).

- **Shape:** input and output shape of the module (e.g. when the current object is replaced, only a blueprinted object with a matching shape will be used).
- **Meta information:** container for any other information such as a *score* for the blueprinted object.
- **Mutables:** a list of sub-modules that are allowed to be modified at runtime.

A concrete example in practice is given in C.4.

3.3.2 Motivation for Blueprints

Having blueprints has the following benefits:

- Blueprints allow a complete, hierarchical description of a model so that a model can be partially or fully constructed.
- Modules(i.e. model components) can be modified algorithmically. Choosing, controlling, and logging which parameters to share is fairly simple.
- Sharing hierarchical portions of a model with other models is simple.
- At runtime, it is possible to replace any shared module with one that is not shared (i.e. a unique module).
- At runtime. a portion of a given model can be completely modified, while the rest retains its current weight.

3.3.3 Parameter Sharing with Scopes

A *scope name* is simply a string assigned to a blueprinted object. Based on where the given object in the hierarchy of other modules is, an optional prefix can be provided.

The prefix can contain the current scope combined with a name that often describes the learnable parameters and other differentiating factors of a given submodule. It can directly impact how an object must be shared. An optional suffix is used for differentiating modules for uniqueness so that they will not be shared for the given experiment,[2] or it can denote a general shape of the outer scope.

During runtime, a global dictionary keeps track of all blueprinted objects and ensures that objects with the same scope name are the same. This way, when necessary, reconstructing models with modifications is fairly cheap.

Aside from PyTorch, the scope-based parameter-sharing approach was used for experiments with Tensorflow [197]. The relevant results for *aggressive parameter sharing* by majorly sharing all convolutions of the same shape and stride are given in Section 4.5. Furthermore, for the CIFAR-10 and Imagenet-1K benchmarks, with a modern CNN model such as EfficientNetv2 [1], this approach can outperform implicit models [98, 99, 8, 9, 198, 7, 11, 10] and ResNet models with parameter sharing as discussed in B.

3.3.4 Rules of Parameter Sharing

Revolver allows parameter sharing by relying on scopes and following strict rules for hierarchical objects. In large hierarchies of scoped objects, this ensures consistency of shared components and garbage collection of unique components.

1. A scoped object's scope name dictates whether it will be shared or not. For instance, convolutional layers of the same scope name (and consequently the same parameter shape) will be shared.

2. Multiple blueprints can describe the same object in memory. When blueprints with the same scope name are used for object creation, they must all refer to the same object.

[2] In such a scenario, the suffix would simply be a unique string.

3. A scoped object can be either *unique* or *shared*. A shared object can take roles in multiple architectures or within the same architecture in multiple layers. A unique object must only belong to its parent object in a hierarchy.

4. A scoped object can only be shared if none of its scoped components is unique. In a hierarchy, if a scoped submodule is made unique, all its parents must be made unique as well.

5. When there is an attempt to revoke the uniqueness of a scoped object (e.g. via a mutation), in a hierarchical order, the attempt must be repeated for all of its scoped components. All components that can be shared must have their uniqueness revoked. If there is any component that can not be made unique, these components and the scoped object will remain unique while the rest is shared.

6. When there is a blueprint and an associated object in memory, duplicating that blueprint and marking one of its scoped components as unique must refresh and change the names of all unique scoped components. This is to ensure that they will be different from the original object's unique components.

7. When a scoped object is deleted (i.e. unregistered from the global dictionary), all of its unique components are unregistered.

3.3.5 Describing Hierarchical Building Blocks

Revolver has a set of allowed building block types described in blueprints with varying levels of complexity:

- activation[3]
- dropout
- pooling

[3] While the blueprinting framework can easily allow different types, only ReLU was used in the experiments.

- batch normalization [4]
- fully connected layers
- vanilla convolutional layers
- depthwise separable convolutions
- convolutions followed by deconvolutions
- *convolutional units* with convolutional and optionally activation, dropout, batch normalization, and pooling layers
- *residual blocks* with convolutional units of a similar number of channels and skip connections
- *bottleneck blocks* which convolutional units with different channel sizes
- *dense concatenation groups* where blocks' output gradually are concatenated
- *dense sum groups* where blocks' output is gradually summed and instead concatenated together at the output level [5]
- *ensembles* of various hierarchical blocks with the same output and input shapes
- *ResNet and DenseNet schemes* which include the style of building the models (e.g. for reducing the image size DenseNet uses pooling layers, vs ResNet uses strided convolutions)
- *training sessions*, which are higher-level blueprints that include model blueprints alongside optimization-related hyperparameters such as learning schedule information, optimizer type, loss criteria, etc.

The listed building blocks were used in the experiments described in Section 3.5.6. Note that each hierarchy can further be modified algorithmically. For instance, the

[4] The experiments were suitable for large batch sizes, therefore alternatives such as layer normalization were skipped.

[5] Dense sum groups are invented/proposed by the author for compactness

hierarchical mutation operator (Section 3.5.5.3) can take a convolutional layer from a residual block and replace it with a bottleneck block. In practice, this gradually leads to models of increasing complexity.

3.4 Training a Population

It is possible to use Revolver for neural architecture search while simultaneously training a population of models. Unlike other approaches mentioned in Section 3.2, Revolver assigns each component a score in a hierarchical manner using genotype fitness (Section 3.5.5.2). There is a coefficient adjusted at each generation so that the genotype fitness is closer to the phenotype fitness (i.e. a score based on validation loss). This allows training only a small portion of the population at each generation (e.g. 10% to 20%) and comparing blueprints with fully-fledged models during the search process.

Another use case for Revolver is module search and analysis. Since each module described in a blueprint is assigned a separate score, after the population is trained for several generations (e.g. 110-250 generations), it is possible to rank the highest scored modules (e.g. Table C.1) or save, analyze, and reuse them for various purposes.

3.4.1 Search Strategies

For neural architecture search, Revolver provides a random search-based algorithm and a genetic algorithm inspired by tournament selection. Algorithm 1 describes a search pipeline that can be integrated with custom search algorithms. For this study, the genetic algorithm and random search demonstrate how neural architecture search can be achieved by combining the blueprinting approach with parameter sharing. More advanced techniques can be integrated into future work.

3.4.1.1 The Search Pipeline

The search pipeline first initializes a large number of candidate architectures by randomly sampling from a subset of possible building blocks discussed in Section 3.3.5. [6] The resulting architectures are assigned scores based on their *genotype fitness* (i.e. Algorithm 2). Then a portion of them (e.g. top 50%) is used for the first generation of a population while the rest are discarded.

For each generation, two relatively low-performing individuals from the population are randomly selected, and a search algorithm tries to find new individuals for replacement based on their scores. The search algorithms may fail, so this is potentially repeated a maximum of three times.

The search algorithms are given a score and they try to find a new individual with a better genotype fitness. They may fail to find any individual. In such a scenario, the main search pipeline reduces the genotype fitness coefficient a small amount so that next time there is more chance to find an individual. If an individual was found, the genotype fitness coefficient is instead increased by a small amount. The main reason for this is that the genotype fitness is based on a performance estimation which is fast but fairly imperfect. During the score update phase, some of the individuals in the population are trained and their scores are updated with the validation loss. This hybrid approach allows a reasonable initialization for individuals without training, and the validation loss updates their score towards a more precise direction.

Overall, at each generation, models from the population are constructed, potentially reusing modules from previous generations that were updated during the training that happens score update phase. The models in the current generation are then trained for a short while for the next score update phase. The validation loss is converted into a utility score and used for updating all relevant module scores. As multiple models may have the same parameters, some parameters get the opportunity to be trained more. Eventually, despite training for a short period at a time, the constructed

[6] This step is equivalent to the sampling for the random search algorithm.

models start converging due to achieving a good initialization point by reusing the majority of the previous modules. When there is not much difference in the top score of the population for a reasonably long enough time, the population is assumed to be converged. This approach allows searching for a compact model or ranking module types based on their reusability.

3.4.1.2 Random Search

Random search is given a score and iteration budget and it tries to find a randomly generated individual with a genotype fitness better than the given score. It may fail to find any individual within the allowed number of iterations.

3.4.1.3 Genetic Algorithm

A simple genetic algorithm is implemented in Revolver. The implementation is inspired by the tournament selection strategy [199]. To work on blueprints, and handle hierarchies, the algorithms for the hierarchical mutation and crossover operators are provided in Section 3.5.5.

The genetic algorithm first takes a target score. Two individuals are selected from the population either randomly or based on their scores. Then their offspring are generated with potential crossover and mutation operations. The offspring are evaluated for genotype fitness and either offspring is returned or with some chance, the offspring are eliminated without any training due to having genotype fitness worse than the given score. In the second case, the algorithm fails to find any individual.

To add diversity to the population, an idea of *immigration* is incorporated into the genetic algorithm as well. Based on a small probability that gets smaller with time, if any random architecture can be found with a genotype performance than the top two individuals, then they will be immediately returned as the new individual.

3.4.2 Optimizations

New model blueprints, i.e. new genotypes, are given a genotype fitness. Genotypes with bad fitness are discarded without training (e.g. models with more than 16 M parameters). There is a compute budget so that models that have better scores get more batches to train and improve. Further, in the cases where multiple crossover operations cause a large model, such models likely receive a lower score since for the phenotype fitness there is a penalizing term for size. This way, smaller models are prioritized initially, and as the competition grows, they leave their place to the deeper models. The deeper models can still inherit the weights from some of the components from smaller models since they are globally shared. This aligns well with the ideas from Section 3.2.2.

3.4.3 Exploration vs Exploitation

Picking phenotypes to train for a given generation gives the picked individuals a bit of an advantage compared to the rest of the population. By default, the selection process is half random for diversity, and half based on the scores of each individual. Higher-scored individuals have a higher chance to be trained even further. The trade-off between exploration and exploitation can be adjusted by changing the probability of random selection and allowing genotype scores to be modified with a coefficient. When picking which new genotypes to add to the population, the described optimizations also indirectly allow prioritizing modules from previous generations that were part of successful models by gradually discarding the worst-performing models.

3.5 Methodology and Experiments

The experiments were conducted with PyTorch [200] using the two object classification benchmarks given in Table 3.1, majorly focusing on CIFAR-10 for comparison

of various hyperparameters of Revolver. The training set in the benchmarks was randomly split by 10% for the validation set for the population training, where models with lower validation loss received better phenotype fitness. To avoid fitting the test set, the test set was reserved for evaluating the final performance of the population training.

3.5.1 Hyperparameters

In the experiments, the genetic algorithm was compared to the random search for the MNIST, and CIFAR-10 datasets with various matching hyperparameters. The population size was fixed. As a measure of success for each population, the test accuracy of the top individual from each individual (without any fine tuning) and the mean values of the last 200 validation accuracies were given. For all experiments, after the first 100 generations, the initial learning rate (.35) was exponentially reduced until it reached the minimum learning rate (.00085). For MNIST a batch size of 512, and for CIFAR-10 128 was used. The full set of hyperparameters alongside the instructions to reproduce results similar to the experiments are available at `https://github.com/gozepolat/revolver`. Furthermore, in C.2 the full details of the hyperparameters used for each experiment are given.

3.5.2 Parameter Sharing

Convolutional layers of the same shape were *aggressively shared*. That is, convolutions constructed with the same arguments such as kernel size, stride, etc. were treated as the same object unless marked unique. Furthermore, models with repeating architectures had cross-layer parameter sharing unless their layers were mutated to become unique and never shared.[7] Finally, different models reused and updated a portion of the same parameters during the population training.

[7] Weight inheritance from unique modules was not explored. For future work, high-performing unique modules may be duplicated and used as shared components for their descendants instead.

3.5.3 Warm up Phase

There was an optional *warm up* phase applied in some of the experiments for comparison. Without any training, a search based on genotype fitness was conducted for a given number of generations. The warm-up relied on the same search algorithms namely the random search and genetic algorithm except for the score update phase where phenotype fitness is calculated. Also, the genotype fitness coefficient was held constant.

3.5.4 Model Construction Schemes

The main model construction schemes that were compared were DenseNet [13], and Wide Residual Network (WRN) [132]. The repetitive nature of residual blocks was suitable for testing parameter sharing and the versatility of the blueprinting approach in practice. DenseNet concatenates layer outputs from earlier layers as output to the next layer, and the shape of the convolutions can vary more than WRN. The bottleneck blocks and concatenations were commonly used in the top models found during population training.

Table 3.1: Details on MNIST, and CIFAR-10,datasets

Dataset	Train	Validation	Test	Classes	Image
MNIST[201]	54000	6000	10000	10	28
CIFAR-10[14]	45000	5000	10000	10	32

3.5.5 Algorithms Used

Revolver relies on a search pipeline (Section 3.4.1.1). It provides two search algorithms: random search, and genetic algorithm. Both search algorithms rely on the

genetic fitness for comparing models described in Section 3.5.5.2. The random search algorithm picks a width and depth from an allowed range and creates a new model. Each time there is a reduction in resolution, the width of the model increases. The genetic algorithm has the hierarchical mutation, crossover, and copyover operators. The copyover is similar to the hierarchical crossover operator where the only difference is, instead of a swap operation, it overwrites a portion of the target blueprint.

3.5.5.1 The Search Pipeline Algorithm

Algorithm 1 SearchPipeline: Search & replace individuals for the next generation

Input: $Population$, $SearchAlgorithm$, N, P

1: $discarded \leftarrow Population.pickSubset(N, P)$
2: $replaced \leftarrow []$
3: **for** $index$ in $discarded$ **do**
4: $\quad score \leftarrow Population.genotypes[index].score$
5: $\quad blueprint \leftarrow searchAlgorithm(Population, score, exclude = discarded)$
6: $\quad$ **if** $blueprint$ is $null$ **then**
7: $\quad\quad reduceGenotypeCoefficient()$
8: $\quad$ **else**
9: $\quad\quad Population.replaceIndividual(index, blueprint)$
10: $\quad\quad replaced.append(index)$
11: $\quad\quad increaseGenotypeCoefficient()$
12: $\quad$ **end if**
13: **end for**
14: **return** $replaced$

Algorithm 1 describes what was mentioned in Section 3.4.1.1. For the sake of simplicity, the initialization of the population is skipped as well as the retry loop for the cases where no individual was replaced in a single pass.

3.5.5.2 Fitness Functions

Revolver introduces two different fitness functions, for genotype (i.e. blueprint) and phenotype (i.e. DL model). The genotype fitness given in Algorithm 2 relies on the input and output shapes of the convolutional layers, the number of learnable parameters, and several weights calculated from a model blueprint. Note that the number of weights can be larger than the number of learnable parameters as there can be cross-layer parameter sharing. It is possible to quickly search a subset of architectures by only relying on this fitness function without any training, though it is likely not very accurate.

An optional but important phase is the *score update phase*. This is done by training a small subset of the population for a duration budget based on their fitness (i.e. maximum an epoch and minimum 96 batches) and updating the existing scores with a small weight using the validation loss for scoring (i.e. the phenotype fitness in Algorithm 3).

3.5.5.3 Hierarchical Mutation

Mutating a blueprint (i.e. genotype) can be done hierarchically. In Algorithm 4 a mutable element from the current hierarchy of a given genotype can be mutated (i.e. Algorithm 6), or one that is lower in the hierarchy can be randomly selected for the mutation (i.e. Algorithm 5).

When an element lower in the hierarchy is selected, this often leads to a smaller, and less dramatic change, as a finer-grained component of the genotype is targeted for modification. For instance, at the top level, an entire block can be modified (e.g. marked for uniqueness to avoid parameter sharing as in line 5 in Algorithm 6), when lower in the hierarchy, a mutation can instead occur at a specific component such as a convolutional layer.

Algorithm 2 GenotypeFitness: estimate model fitness based on its blueprint

Input: $Blueprint$, $GenotypeFitnessCoefficient$

$flattenedModuleList \leftarrow collectAllModules(Blueprint)$
$roughScore \leftarrow 1$
for $moduleBlueprint$ in $flattenedModuleList$ **do**
  if $moduleBlueprint['type']$ is $Convolution$ **then**
    $inputShape \leftarrow moduleBlueprint['inputShape']$
    $outputShape \leftarrow moduleBlueprint['outputShape']$
    $layerScore \leftarrow \log_2(\sqrt{prod(inputShape) * prod(outputShape)})$
    $roughScore \leftarrow roughScore + layerScore$
  end if
end for
$numWeights, numParams \leftarrow getSizeStats(Blueprint)$
$roughUtility \leftarrow \log_2(numParams * roughScore/numWeights)$
$utility \leftarrow clip(roughUtility, 0)$
$fitness \leftarrow GenotypeFitnessCoefficient/(utility + 1)$
return $fitness$

Algorithm 3 PhenotypeFitness: estimate model fitness based on validation loss

Input: $Blueprint$, $NumBatches$

$session \leftarrow makeTrainingSession(Blueprint)$
$validationLoss \leftarrow session.trainBatches(numBatches)$
$numParameters \leftarrow session.getNumModelParams()$
$sizePenalization \leftarrow penalize(numParameters)$
$fitness \leftarrow validationLoss * (1 + sizePenalization)$
return $fitness$

Algorithm 4 Mutate: mutating a blueprint (i.e. genotype)

Input: $Genotype$, Key, P, $ChoiceFn$

if Key is $null$ **then**
  $Key \leftarrow ChoiceFn(Genotype)$
end if
if $random() > P$ or Key not in $Genotype.mutables$ **then**
  return $MutateSubElement(Genotype, Key, P)$
end if
return $MutateElement(Genotype, Key, P)$

Algorithm 5 MutateSubElement: mutating a blueprint sub element

Input: $Genotype$, Key, P

$value \leftarrow Genotype[Key]$
if $isSubClass(type(element), Blueprint)$ **then**
  if $len(element.mutables) > 0$ **then**
    $randomKey \leftarrow randomPick(element.mutables)$
    $Mutate(element, randomKey, P, randomPick)$
  end if
end if

Algorithm 6 MutateElement: mutating a blueprint element

Input: $Genotype$, Key

$domain \leftarrow Genotype.mutables[Key]$
$newValue \leftarrow domain.getRandom()$
$adjustMutation(Genotype, Key, newValue)$
if $random() < Genotype[Key].toggleP$ **then**
  $toggleUniqueness(Genotype, Key)$
end if

3.5.5.4 Hierarchical Crossover

Similar to the mutation operator, a crossover of two blueprints (i.e. genotypes) can be done hierarchically. That is, the granularity of the operation can be picked probabilistically. An additional complexity for the crossover operator is the compatibility of the sections that will be swapped between two blueprints. No matter where in the hierarchy, each blueprint is aware of its input and output shape. The crossover operator takes advantage of this. With a given probability, it tries to swap convolutional or fully connected layers in the current level of the hierarchy. Otherwise, for the lower levels in the hierarchy, it probabilistically calls another crossover operator that takes the children of each blueprint and detects locations that are compatible in terms of input and output shapes. Then it swaps a portion of the children for the parent blueprints. With a probability, it may also try this swap not on the children but even lower in the hierarchy (e.g. grandchildren x grandchildren or children x grandchildren). For the sake of simplicity, the algorithm that picks at which level in the hierarchy the crossover will apply is described in Algorithm 7 and the full pipeline that handles finding compatible locations for crossover, as well as the vanilla algorithm for a regular crossover on sequences, are omitted. The full code is available here: `https://github.com/gozepolat/revolver`

3.5.6 Experiment Results

The results in Table 3.2 show that for the simpler MNIST benchmark, the genetic algorithm did not introduce any benefit, while for CIFAR-10, it consistently scored better than the random search both in terms of the top-1 test accuracy of the best individual in the population and in terms of the mean validation accuracy from the last 200 scores in the population. For the the population trained in *Evolve* mode (the last row in Table 3.2), in Figure 3.1 the moving average of the validation accuracy shows that the population converges in around 150 generations to 90% validation

Algorithm 7 Crossover: swap arbitrary portions of two blueprints (i.e. genotypes)

Input: $Genotype_1$, $Genotype_2$, P

$children_1 \leftarrow Genotype_1.children$
$children_2 \leftarrow Genotype_2.children$
if $random() < P$ **then**
    $isCrossed \leftarrow VanillaCrossover(children_1, children_2)$
    if $isCrossed$ **then**
        return True
    end if
end if
if $random() < P$ **then**
    $progeny_1 \leftarrow randomPick([children_1, Genotype_1])$
    $progeny_2 \leftarrow randomPick([children_2, Genotype_2])$
    $isCrossed \leftarrow FullCrossover(progeny_1, progeny_2, P)$
    if $isCrossed$ **then**
        return True
    end if
end if
return False

accuracy. [8]

Increasing the number of samples for the score update phase (i.e. training) from 10% to 20% (i.e. from 8 to 16 samples) increased the overall performance for CIFAR-10. The *Gen.* column in the table shows the generation where the peak test accuracy was achieved. For the phenotype score, validation loss was combined with a penalizing term for model size. This forced the search algorithms to penalize models that are too large, or in other words models that have low parameter efficiency. For instance, in Table 3.2, the largest top-performing model had 3.1 M parameters.

In C.3 some of the top-ranked shared modules are given. The hyperparameters regarding each experiment are available in C.2. Moreover, additional observations and details regarding the experiments are included in C.3.0.1. Finally in C.5 architecture and blueprint details for a mutated top-performing model (i.e. 89.6% in CIFAR-10 after 147 generations) are exemplified, where one of the convolutional layers was mutated into a higher level container unit (i.e. conv unit) which contained an ensemble of three convolutional layers.

3.5.7 Comparison with ENAS

As discussed in Section 3.2, ENAS [160] provided improvements to NAS with parameter sharing to achieve results on par with the state-of-the-art. An important distinction between the results shared in Table 3.2 and the results reported by ENAS is that Pham et al. directly used the test set during the optimization of the RNN controller. This has the risk of the hyperparameters overfitting the test set, potentially leading to biased results. Hence in the experiments in this work, instead the training set was split and a validation set was used for the search algorithms. The test set was used for the reported results without being used for any type of optimization. Another distinction is that Pham et al. trained the top model architecture from scratch after the population training is done. They trained the top model for 310 epochs for macro

[8] For the moving average, to reduce the noise in the plot, a sliding window of size 700 was used.

Table 3.2: Random search vs. genetic algorithm on MNIST and CIFAR-10 datasets without additional training or fine-tuning. There was no significant difference for MNIST except for the mean validation accuracy of the population (bold). Yet, for CIFAR-10, the genetic algorithm (bold) consistently outperformed the random search, regardless of the warm-up method (**RandWarmEvolve**: random warm up then search with genetic algorithm, **RandWarmup**: random warm up then random search). Increasing the sample size per generation from 8 to 16 improved the top-1 accuracies by around 1% and seemed to be more efficient than training for more generations.

Dataset	Mode	Gen.	Sample Sz.	Model Sz.	Top Test Acc.	Mean Val.
MNIST[201]	Evolve	130	8	1.6M	99.6	99.4
MNIST	Random	117	8	1.7M	99.6	**99.5**
CIFAR-10[14]	RandWarmup	146	8	620K	88.5	86.7
CIFAR-10	**RandWarmEvolve**	147	8	790K	**89.6**	**89.1**
CIFAR-10	RandWarmup	238	8	2.0M	88.8	70.4
CIFAR-10	**RandWarmEvolve**	250	8	1.6M	**90.5**	**71.1**
CIFAR-10	Random	139	16	3.1M	90.8	89.4
CIFAR-10	**Evolve**	133	16	2.2M	**91.9**	**89.8**

search, and 630 epochs for micro search. Unlike ENAS, the reported results are directly from population training. With ENAS, with similar amounts of search time and training conditions, the top model before training from scratch has test accuracy lower than the reported results in this chapter.

3.6 Limitations and Future Work

There are several limitations and improvement points for Revolver. The genotype fitness algorithm in Section 3.5.5.2 can be improved with a more advanced performance estimation technique. In Section 4.4.5 more advanced approaches for estimating model performance without training are discussed. Compared to these al-

ternatives, Algorithm 2 is highly crude but significantly faster. For future work, the advanced alternatives will be tested with larger models and datasets.

Since there is a significant amount of parameter sharing, catastrophic forgetting at the later stages may become an issue for harder benchmarks. A potential future solution is freezing the modules that have the top scores so that they can not change after a certain time or threshold. As an alternative, a reasonable amount of redundancy in modules may help. The amount of redundancy can be currently adjusted in Revolver with a hyperparameter. For future work, less aggressive ways of parameter sharing can be tested alongside weight inheritance from unique modules.

While the introduced blueprinting technique is flexible enough to be used for various purposes, the searching approach described in this chapter is a metaheuristic, and it relies on several hyperparameters that introduce a large search space. Since this makes ablation studies extremely compute intensive, it is hard to know where the simplifications are in the described methodology. For the sake of objectiveness, the random search algorithm was rigorously tested and optimized with the available resources. For consistency and interpretability, the same pipeline and hyperparameters were used when comparing the random search with the genetic algorithm. In a more comprehensive future study, the number of experiments can be increased to address the lack of ablation studies.

The major bottleneck of the described methods in this chapter is the GPU budget and memory. As the models in a population grow in size and become more competitive, the GPU memory that is needed for the training steps becomes significant. Thus, the number of experiments was limited due to the compute requirements of training large populations. Further analysis of hyperparameters regarding the search algorithms, the building blocks that are used for the random search, and for initializing the population, population size, learning rate, and schedule may reveal a more comprehensive picture. Using the blueprinting framework, metaheuristics other than genetic algorithms or bilevel optimization approaches such as RL-based generation of the encoding of some of the building blocks in the blueprints similar to the idea of *cells* introduced in [165,

160] can be used for further analysis as well.

Small practical issues are as follows: While they allow a high level of customization and versatility, blueprints can be hard to read and interpret for large models due to the high level of detail they contain. A visualization tool was implemented using Tkinter [202] library and included in Revolver. Further improvements in interpretability can be made by additional visualization options. Another practical issue is that currently, the population training has no checkpointing feature. For future work, it may be possible to save the state of the global dictionary that stores all modules and reload the whole population.

Finally, as the discussed evolutionary algorithm is majorly about finding a good model design and reusable components, to address the compute limitations and GPU bottleneck, an alternative approach based on graph analysis is introduced in the next chapter. This allows comparing models *without relying on GPUs*. Further, in the next chapter, using the idea of scopes, experiments with cross-layer parameter sharing with the significantly larger ImageNet-1K dataset are given, providing more justifications for the reusability idea discussed in Section 3.2.2.1.

3.7 Conclusions

In this chapter, a new library, Revolver was introduced to analyze parameter sharing and search for reusable modules while training a population. After discussing the relevant research and the background on how a reusable and compact genetic toolbox may have evolved for the nervous system, a blueprinting approach was proposed for a precise description of model components including meta information such as whether they are unique or shared, as well as their reusability or performance scores. Then strategies and optimizations for training a population of DL models were discussed. Finally, in the methodology section, the relevant algorithms were given alongside the experiment results. The hierarchical genetic operators namely hierarchical mutation and crossover were able to treat the blueprints as genotypes and allow searching new

model architectures as well as various hyperparameters while training a population. In less than 135 generations, models evolved with 99.6 % and 91.9% top-1 accuracies for MNIST and CIFAR-10 respectively. For instance, the top evolved model for MNIST contained an ensemble of convolutions, and convolution layers followed by deconvolutions. Moreover, during population training, the meta information on the blueprints was tracked by Revolver, leading to a ranking of shared modules in the population with top scores. Revolver allowed the training of a population of 80 individuals using only a single GPU. Overall, the capabilities and use cases of the blueprinting approach and Revolver were illustrated in the experiments. Revolver may be extended or directly used by researchers for hyperparameter or neural architecture search, as well as analyzing which shared modules are more reusable across models for a given benchmark.

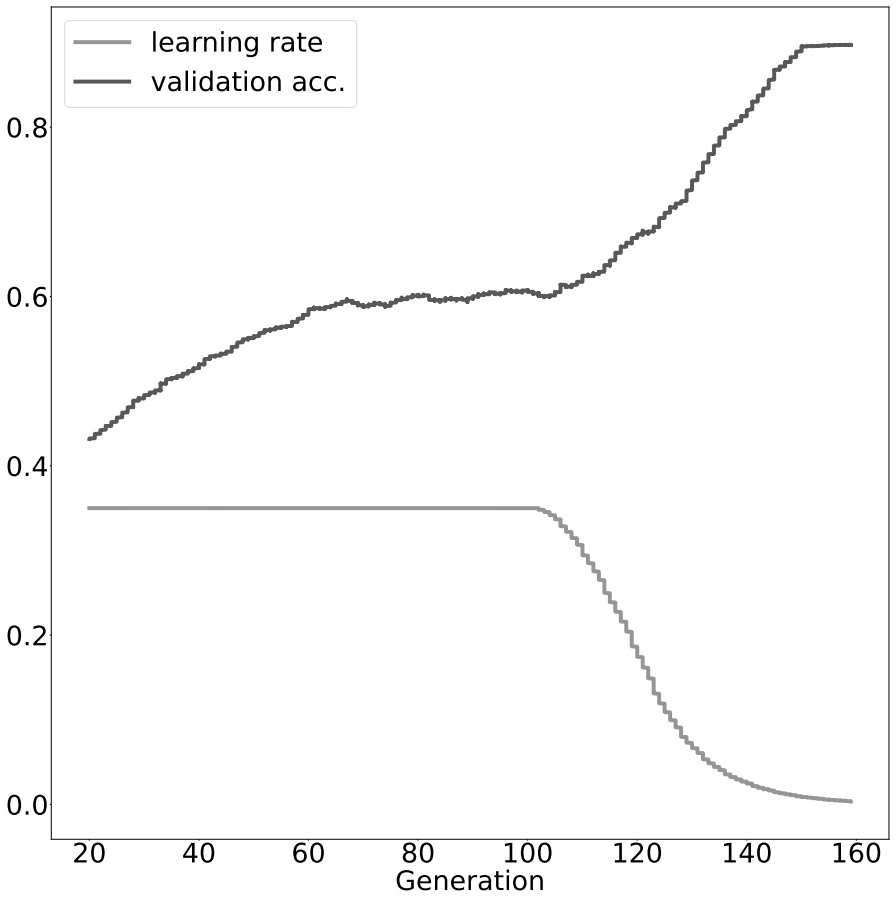


Figure 3.1: With the genetic algorithm the population converges to a validation accuracy of 90% in roughly 150 generations. Note that the convergence speed can be adjusted by changing the learning rate schedule.

CHAPTER 4

THE REUSABILITY PRIOR

4.1 Introduction

In this chapter *the reusability prior* is described [15]. Design, data, and training choices can affect deep learning models. Differences in the number of contexts for model components during training are conjectured to be critical for model design. This notion can be generalized by defining the reusability prior as follows: *model components are forced to function in diverse contexts not only due to the training data, augmentation, and regularization choices but also due to the model design itself.* For the design aspect, a graph-based methodology is introduced to estimate the number of contexts for each learnable parameter. This allows a comparison of models without requiring any training. Supporting evidence with experiments using cross-layer parameter sharing on CIFAR-10, CIFAR-100, and ImageNet-1K benchmarks are provided. Examples of models are given for parameter sharing, outperforming the baselines that have at least 60% more parameters. The introduced graph analysis-based quantities for the reusability prior align well with the results, including at least two important edge cases. Overall the reusability prior provides a viable research direction for model analysis based on a straightforward idea: counting the number of *contexts* for model parameters.

4.1.1 Improving Model Performance

Model performance is often tied to the number of learnable parameters. For example, training large models is a way of achieving the state-of-the-art in computer vision [145, 146, 147, 148, 149, 150, 28, 151, 152, 29, 30, 31] and natural language processing [153, 154, 155, 156, 157]. Yet the number of learnable parameters can not be taken as the sole predictor of performance. It is possible to train models that have significantly lower numbers of learnable parameters with a comparable or increased performance (i.e. higher parameter efficiency). More specifically, additional unlabeled data or curation of larger datasets reduce error in a predictable manner [203]. Hoffmann et al. provide example of models outperforming the previous state-of-the-art models with significantly more compact ones by relying on more training data. Their systematic analysis shows that increasing the number of parameters alongside the size of the training set in a balanced manner leads to more optimal results in terms of the efficient usage of compute resources [19].

Aside from the number of parameters and training samples, different design, augmentation, and regularization choices can impact the performance as well. The main conjecture given in this work is that model performance is tied to not only the number of parameters but also how these parameters are *reused* within various *contexts* that can arise due to different training samples, augmentation and regularization choices, as well as model design itself.

4.1.2 What is a Context?

In Figure 4.1, there is an informal description of *context*.The set of all contexts for a given parameter can be defined as follows:

Definition 1. *Let f_{iw} be a function node that at least takes a given parameter node w and an associated input h_i; let $\mathcal{O}_k$ be any node from the set of all output or leaf nodes $\mathcal{L}$ in graph $\mathcal{G}$, and p a path that connects f_{iw} to $\mathcal{O}_k$. The set of all contexts for w, $\mathcal{C}_w$*

is given by

$$\mathcal{C}_w = \bigcup_{f_{iw} \in \mathcal{G}} \{\, p \mid f_{iw}((h_i, w), ...) \overset{p}{\rightsquigarrow} \mathcal{O}_k \quad \forall \mathcal{O}_k \in \mathcal{L}\} \tag{4.1}$$

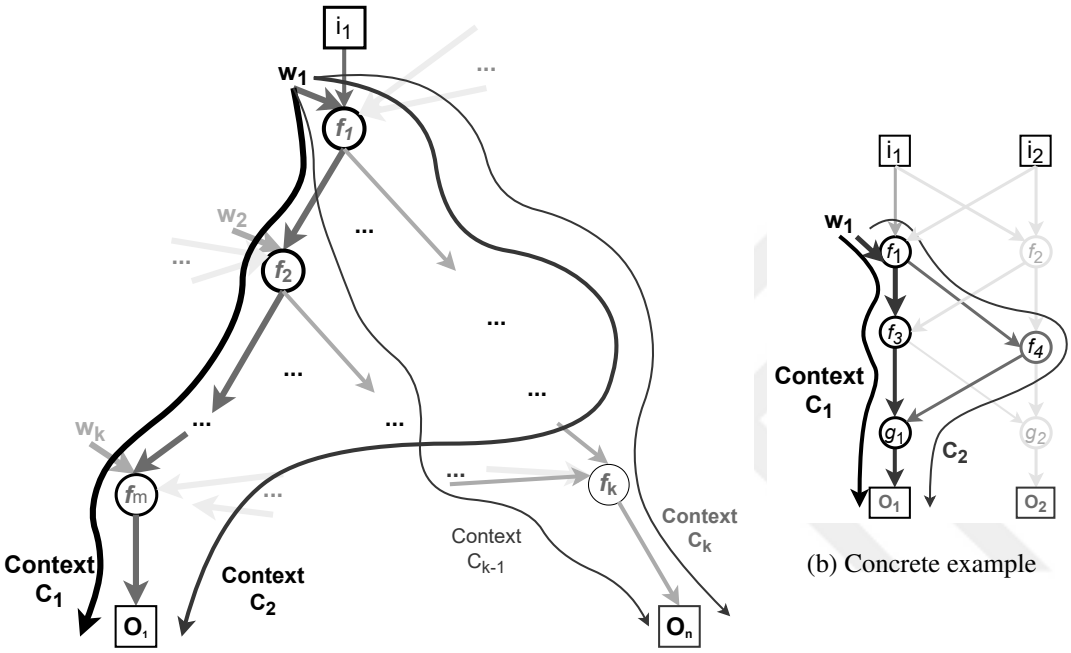
Note that f_{iw} can be any function node within the model graph $\mathcal{G}$ that at least takes parameter w and some hidden or input feature h_i. Parameter w can be used in multiple places within the model with separate inputs leading to different contexts (i.e. $f_{jw}(h_j, w, ...)$). In other words, $\mathcal{C}_w$ is the union of all contexts associated with w, regardless of where w is used within the model or whether w is explicitly shared.

An important assumption is that f_{iw} is mostly nonlinear in a similar manner to the mainstream deep learning models. If all function nodes were instead equivalent to a given linear function, the model could be collapsed into a single layer where each input is associated with a single learnable parameter. Such a model can not learn functions more complicated than a linear map (i.e. a single matrix multiplication).

Overall, each parameter has its own set of contexts which is the union of all paths contributing to an output node, where h_i is part of the *computational graph* rather than the value of an actual output feature from a previous layer. This definition of context is used for analyzing model architectures and defining other relevant concepts. In Figure 4.1, some of the possible contexts that can arise inside a model graph are given. In Figure 4.1b the computational graph of a model with three layers is described. How can model design change the number of possible contexts? An example in Figure 4.2 compares two configurations that yield different numbers of contexts for the same number of parameters.

4.1.3 Parameter Sharing

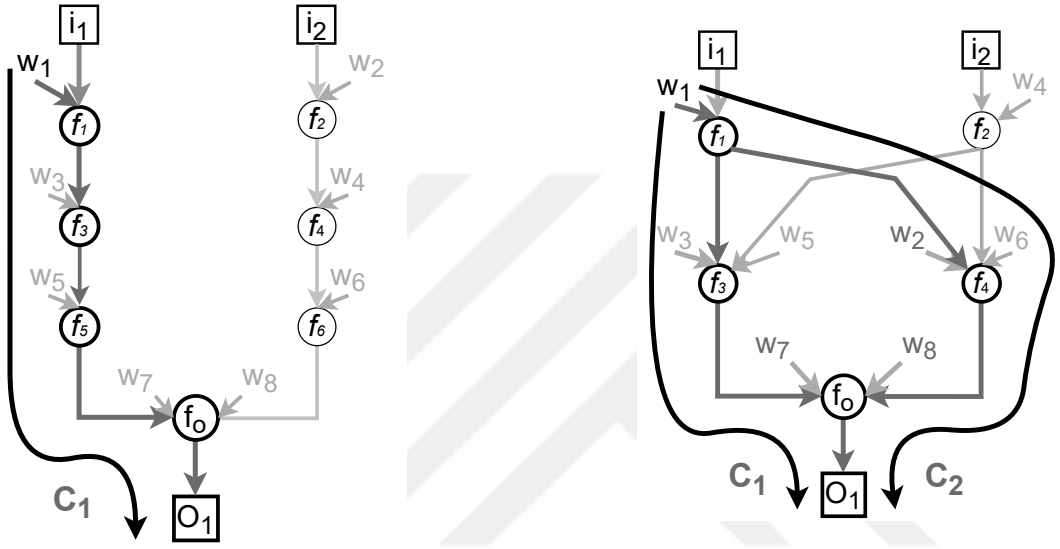
Small models often hit an upper bound early on in terms of performance for large-scale benchmarks, yet parameter sharing is ubiquitous in deep learning. For in-



(a) Arbitrary depth and width

(b) Concrete example

Figure 4.1: A context is a path from a parameter associated with an input to an output. **(a)** C_1 is the bold path from w_1 associated with i_1 through functions f_1, f_2, .., f_m contributing to the output O_1. Note that w_1 can contribute to O_1 through multiple contexts (e.g. C_1 and C_2). **(b)** A concrete analogy for C_1 and C_2 given on a small architecture.



(a) Less reusable

(b) More reusable

Figure 4.2: Given the same number of parameters, model design can change parameter efficiency: (a) w_1 contributes to a single context C_1 (b) w_1 contributes to an additional context C_2. The reusability prior suggests that w_1 is likely to be forced to function for both contexts and hence will become more reusable. Furthermore, by relying on the repetition of w_1 in both contexts, more complex functions can be succinctly described with the same number of parameters.

stance, convolutional kernels [35, 50] repeat spatially (i.e. sliding window), while recurrent modules [204, 205, 206, 207, 208] (or any autoregressive model such as [209, 210, 211]) repeat temporally. Domain-specific benefits and being able to work with different input or output sizes are fair motivations for parameter sharing; ultimately model capacity is recovered by scaling to larger models at the cost of more floating point operations per second (FLOPS).

An explicit form of parameter sharing is cross layer parameter sharing which ties the weights of architecturally repeating layers together. The literature shows that, at the cost of more FLOPs, cross layer parameter sharing can sometimes improve parameter efficiency. Once a model with cross layer parameter sharing is scaled back to a reasonable capacity, it has a good chance to outperform the original baseline model [5, 6, 212, 3, 84, 85]. In Section 4.5.6 a similar outcome with new examples from this study on some of the EfficientNetv2 models [1] is provided.

4.1.4 Improving parameter efficiency

All mainstream DL models already share parameters directly or indirectly to various degrees, and this affects their parameter efficiency. In Section 4.4.1, a simple horizontal unrolling approach is introduced for making parameter sharing explicit for all directed acyclic graphs (DAG).

The generalized notion *reusability prior* is given to disentangle the intrinsic design and training choices that affect a model's performance in Section 4.4. A consequence of the reusability prior is that for models of similar size and capacity, designs that maximize the expected number of contexts are more likely to improve parameter efficiency. The new experiments on EfficientNetv2 as well as the literature on cross layer parameter sharing provide supporting evidence for the reusability prior.

A simple counting approach for model comparison is introduced for graph analysis. By treating relative frequencies derived from the number of all possible contexts per

parameter as a probability distribution, it becomes possible to define quantities for the comparison of model graphs, including *entropy*, *expected spread*, and *total surprisal*. Formal proof is given for when the total number of contexts is held constant, increasing the expected spread reduces the entropy. Performance estimation approaches are also introduced for comparing models without training. In Table 4.3 and Table 4.4 graph analysis and experiment results aligned well. The major focus on the model design aspect is described in Section 4.2.2.3. Note that the data and training aspects can be taken into account when considering the number of possible contexts as well. This is left for future work.

In summary, the major contributions in this chapter are twofold:

- The reusability prior (Section 4.4.2) is introduced alongside a methodology based on graph analysis (Section 4.4.3). To the author's knowledge this work is the first to introduce a generalized notion of reusability that ties training, design, and data aspects together. For the design aspect, graph analysis based quantities are defined and derived from counting the number of contexts for each learnable parameter.[1] Overall, the introduced quantities allow comparing arbitrary DAGs or model architectures without relying on any training. The graph analysis based quantities align well with the results, including at least two important edge cases in Section 4.5 in Tables 4.3 and 4.4.

- It is empirically shown that it is possible to achieve higher parameter efficiency by aggressively sharing parameters in EfficientNetv2 models. In the experiments, even though the numbers of parameters are at least 60% larger for the original baseline EfficientNetv2-b0 models, EfficientNetv2-S models with aggressive parameter sharing consistently outperformed the baselines on CIFAR-10 [14], CIFAR-100 [76], and ImageNet-1K [77] image classification bench-

[1] To the author's knowledge, the majority of the definitions, lemmas, theorems, and proofs are original, except for what is borrowed from the information theory, namely entropy, surprisal, expected value, and an existing identity regarding Kullback-Leibler divergence used in 4.4.4.1. Despite not being able to find them in other relevant studies, the author suspects that the horizontal unrolling approach and the idea of a uniform graph introduced in Section 4.4.1 likely already exist in other disciplines due to their simplicity.

marks.

4.2 Background

Fundamentally, convolutional and recurrent neural networks (RNN) rely on architectural priors based on reusing model components through space and time. Implicitly, recurrent neural networks share recurrent modules through time [48] and convolutional neural networks (CNN) share convolutional kernels through space. Moreover, it is possible to share 3d convolution kernels spatiotemporally [213]. Reusing components in various ways is an important underlying pattern as larger and larger architectures are adopted [63, 100, 101, 214, 215, 216, 217, 3, 85], searched [217, 165, 166, 167, 168, 169, 170, 171, 218], skip connections are introduced [12, 13, 91, 219], domain specific symmetries are captured [112, 113, 114, 115, 116, 117], model size is reduced or manipulated [220, 221, 5, 222, 6, 4, 223, 89], or parameters are globally shared or transferred from different domains or models [224, 179, 104, 225, 105, 103, 217, 226, 86, 227].

DenseNet models are claimed to be *compact* due to the concatenation of previous features at each subsequent layer [13]. Another strategy for compactness is used by Xception, or "Extreme Inception" [137]. It minimizes the parameters from 3x3 convolutions by using *depthwise* 3x3 convolution layers, i.e. one independent convolution kernel per channel. Then it relies on 1x1 convolutions which allow reusing the depthwise convolution outputs by taking the linear combinations of the outputs. 1x1 convolutions are an important form of parameter sharing as they are equivalent to fully connected layers shared in the height and width dimensions. MobileNet models [228, 229] also focus on maximizing the 1x1 convolution operations. Achieving compactness via soft sharing [140], student-teacher architectures [141, 142], pruning and quantization [143] are other relevant approaches.

There is some research on the analysis of model architecture as well as entropy based approaches to analyze how training data is utilized. For instance, Peer et al. intro-

duce *batch entropy* regularization that allows training very deep models without skip connections. Wickstrøm et al. analyze different training phases of deep models with information plane theory using Rényi's entropy [230]. Levine et al. theoretically predict that there is an "optimal depth-to-width allocation for a given self-attention network size". They recommend significantly wider models as the model size increases and passes a certain threshold [158]. Bu et al. investigate the topological entropy of neural networks with ReLU activations, providing an upper bound of $O(d \log w))$ where d is the depth and w is the width or number of neurons at each layer [231]. The work described here diverges from Bu et al. as in this work Shannon entropy [232] was used alongside a methodology that can work with any DAG without requiring constant width or a specific model. Furthermore, the focus on parameter efficiency in this work, led to a probability distribution that is based on the relative frequencies derived from the number of contexts for each parameter.

4.2.1 Cross-Layer Parameter Sharing

SharesNet [5] demonstrates that for wide residual networks (WRN) [132], it is possible to surpass the original model's performance with parameter sharing and then scaling (i.e. increasing depth and/or width). Similar to lookup based CNNs [233], Savarese and Maire train a model that learns to take a linear combination of a shared pool of kernels [6]. Then the coefficients for the linear combination of kernels are also used for constructing a similarity matrix so that similar layers can be shared. This approach outperforms the original WRN baseline on Imagenet-1K and CIFAR-10 with a similar or less number of parameters. Atom-coefficient decomposed convolution (ACDC) [212], inspired from [234], first decomposes convolutional kernels into bases and coefficients, and then shares the coefficients across layers. This leads to improved parameter efficiency for very deep convolutional nets (VGG) [235], ResNet, and WRN baselines. Shapeshifter networks (SSN) used for neural parameter allocation search (NPAS) do not make architectural assumptions such as repeated layers, but instead, learn to reuse parameters from a limited pool by transforming them into

weights for any architecture [4].

Aside from the computer vision domain, a lite BERT, bidirectional encoder representations from transformers (ALBERT) [3] uses cross layer parameter sharing to first reduce the number of parameters and then scale up to a capacity comparable to the original model [236]. This surpasses the original model's performance for language understanding tasks. Other relevant examples share attention weights [84], speed up training by parameter sharing and then unsharing [237], and explore sandwich style weight sharing for generative transformers [85].

The usage of implicit models can be considered a continuous form of cross layer parameter sharing. Well known examples are neural ordinary differential equation solvers (ODEs) [98, 99], variants of deep equilibrium models (DEQ), and multiscale DEQs (MDEQ) [8, 9].

4.2.2 Maximizing the Number of Contexts

Deep learning literature has various strategies for improving performance. Intrinsically, they often seem to maximize the number of contexts for model components. In Section 4.1.2, and Figure 4.1 more details are given for what is meant by *context*.

4.2.2.1 Diversity in input affecting the number of contexts

Aside from increasing the number of samples, a performance gain can be observed with data augmentation techniques and related approaches that rely on diversifying individual training samples (e.g. cutout [94]), or combining multiple samples (e.g. mixup [238], cutmix [239] etc.).

4.2.2.2 Diversifying the roles of model components during training

Regularization such as dropout [133], stochastic depth [90], and block drop [219] increase the total number of contexts that can arise from the same training data.

4.2.2.3 Design choices impacting the role and scope of each model component

Decisions such as network depth, and width, having CNN layers [35, 49] recurrent modules [48], residual connections [12], attention layers [240], training input size, and cross layer parameter sharing [5] impact the expected number of contexts within a model's computational graph.

Overall, the literature hints at an important pattern that can improve the parameter efficiency of deep learning models. This notion is generalized as *the reusability prior* in Section 4.4.2.

4.3 Overview

A generalized notion of reusability and the reusability prior that encompasses training, data, and model design aspects in deep learning in Section 4.4 is proposed. Then by focusing on the model design aspect, a graph analysis-based methodology for model comparison is given. In addition to the supporting evidence from the literature discussed in Section 4.2, new experiments are provided by applying aggressive cross layer parameter sharing to some EfficientNetv2 [1] models. The results from this strategy on image classification benchmarks are available in Section 4.5. Finally, the analysis of the computational graphs of EfficientNetv2 models and comparison of the quantities based on the reusability prior with the empirical results are shared in Sections 4.5.7 and 4.5.8.

4.4 The Notion of Reusability

Maximizing the number of *contexts* for some learnable parameters is critical. All mainstream DL models directly and/or indirectly share parameters, because reusing the output of a component in more than one place within a deep architecture is an indirect way of parameter sharing and increasing the number of contexts. Intrinsically, any deep learning model can be converted to a functionally equivalent form where parameter sharing is made explicit.

4.4.1 Horizontal Unrolling

It is possible to horizontally duplicate parameters to remove multiple output edges in a given model graph so that each node has a single output edge. This would reproduce all node outputs from scratch, eliminating indirect parameter sharing (i.e. replacing feature sharing with direct parameter sharing). The resulting unrolled models would be functionally equivalent. This is illustrated in Figure 4.3. The nodes in earlier layers are duplicated more as they play more diverse roles. A simple recursive algorithm for horizontal unrolling is available in A.2.

Horizontal unrolling can convert any mainstream DL model into a form where parameter sharing is always direct, and each parameter is duplicated as many times as the number of their *contexts* that can arise due to the computational graph. To the author's knowledge, no mainstream DL model exponentially grows in terms of the number of parameters with depth. Hence they all correspond to an unrolled graph that shares parameters. This would be less general and require exponentially fewer parameters than a graph exemplified in Figure 4.3c. This work refers to such graphs as *uniform graph* which can be formally defined as follows:

Definition 2. *Let $\mathcal{G}$ be a model graph. $\mathcal{G}$ is uniform if and only if $|\mathcal{C}_{w_i}| = 1 \ \forall w_i \in \mathcal{G}$, where $|\mathcal{C}_{w_i}|$ is the cardinality of the set of all contexts for parameter w_i.*

Note that for the horizontal unrolling to work, any cyclic graph would need to be vertically unrolled into a DAG first. This already happens in practice during training. Additionally, for differentiable approaches that transform a set of learnable parameters to weights, the parameter transformation and the originally shared parameters would need to be included in the graph (i.e. every w_i would be replaced by the DAG that generates it).

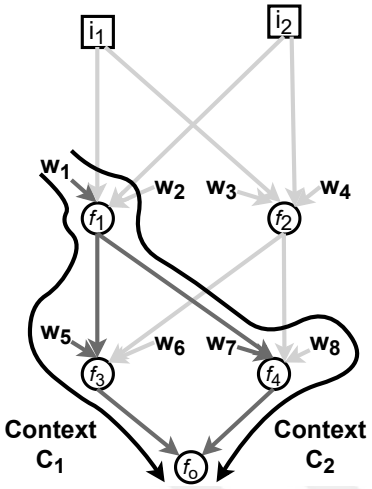
There is a connection between parameter efficiency and parameter sharing. By estimating what the number of *contexts* in the unrolled form would be for each parameter, it is possible to quantify expected spread, total surprisal, and entropy for computational graphs of models.

4.4.2 The Reusability Prior

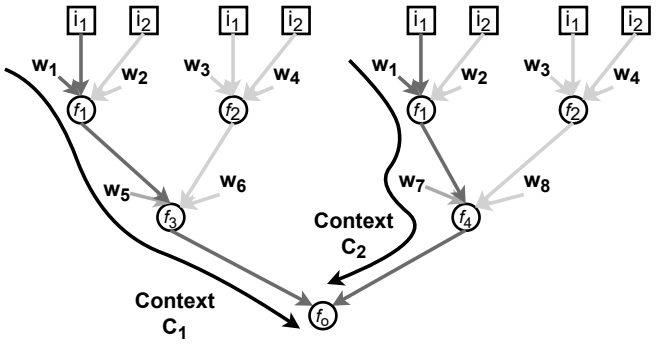
The expected number of *contexts* for model components may be the major reason for differences in model performance. Based on this conjecture, the reusability prior can be introduced as follows:

Model components are forced to function in diverse contexts not only due to the training, data, augmentation, and regularization choices but also due to the model design itself. These aspects explicitly or implicitly impact the expected number of contexts for model components. Until model capacity is reached, maximizing this number improves parameter efficiency for models of similar size and capacity. By relying on the repetition of reusable components, a model can learn to describe an approximation of the desired function more efficiently with fewer parameters.

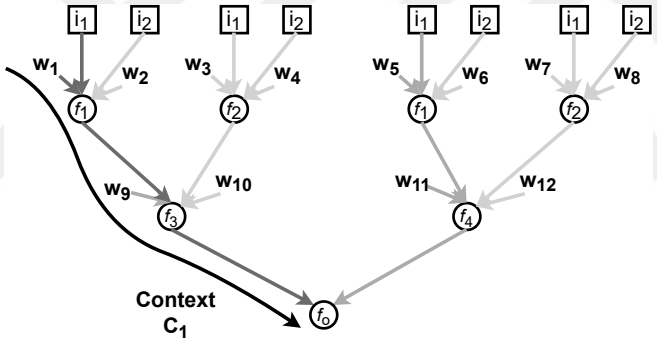
Justifications from the literature are given in Section 4.2, definitions are in Sections 4.1.2, and 4.4.3, finally supporting evidence from new experiments is provided in Section 4.5.



(a) Original graph



(b) Horizontally unrolled graph



(c) Uniform graph

Figure 4.3: Horizontal unrolling reproduces all shared node outputs from scratch, eliminating indirect parameter sharing or feature reuse and instead replacing it with direct parameter sharing with duplicated parameters. (a) w_1 in the original graph has contexts C_1 and C_2 (b) w_1 is duplicated as many times as the number of its contexts in the horizontally unrolled graph (i.e. twice in this example) (c) A uniform graph has a single context for every parameter. It can be considered a non-shared generalization of any model graph, as there is neither feature nor parameter reuse.

4.4.3 Quantities for Model Comparison

Based on the reusability prior and the notion of context, the following quantities are derived: the expected spread, entropy, and total surprisal. Then two different ways of estimating model performance are introduced based on total surprisal and expected spread.

Definition 3. *For a given model graph $\mathcal{G}$, the expected spread is given by*

$$\mathrm{E}[\![\log_2 |\mathcal{C}|]\!] = \sum_{i=1}^{\mathrm{N}_\mathcal{G}} \mathrm{p}(w_i) \log_2 |\mathcal{C}_{w_i}| \tag{4.2}$$

where $|\mathcal{C}_{w_i}|$ is the cardinality of the set of all contexts for w_i, $\mathrm{N}_\mathcal{G}$ the number of learnable parameters, and $\mathrm{p}(w_i)$ is the relative frequency:

$$\frac{|\mathcal{C}_{w_i}|}{\mathrm{N}_\mathcal{C}} \tag{4.3}$$

where $\mathrm{N}_\mathcal{C} = \sum_{w_j \in \mathcal{G}} |\mathcal{C}_{w_j}|$ the total number of contexts in $\mathcal{G}$.

In Section 4.4.4.1 a proof is given for the fact that the expected spread is equal to the relative entropy or Kullback-Leibler divergence [241] of $\mathrm{P}(W)$ from the discrete uniform distribution.

Overall, this quantity prescribes a design maximizing the expected number of contexts. [2] Note that $\mathrm{N}_\mathcal{C}$ and $\mathrm{N}_\mathcal{G}$ grow differently. In 4.2, similar to $\mathrm{N}_\mathcal{C}$, $|\mathcal{C}_{w_i}|$ grows exponentially with depth for the mainstream deep architectures.

Definition 4. *Let w_i be a parameter in graph $\mathcal{G}$. The entropy of the discrete probability distribution for the parameters of $\mathcal{G}$ based on the number of contexts is given*

[2] When all other conditions are fixed, expected spread can be considered a measure of descriptive reusability i.e. parameter efficiency. When comparing models with a similar number of learnable parameters and graph size, models with a higher expected spread can describe more complicated functions.

by

$$\mathrm{H}(W) = -\sum_{i=1}^{\mathrm{N}_{\mathcal{G}}} \mathrm{p}(w_i) \log \mathrm{p}(w_i) \tag{4.4}$$

where $\mathrm{p}(w_i) = |\mathcal{C}_{w_i}|/\mathrm{N}_{\mathcal{C}}$ *is the relative frequency.*

Note that $\mathrm{p}(w_i)$ is also used in the calculation of expected spread in 4.3.

Definition 5. *Let* w_i *be a parameter with relative frequency* $\mathrm{p}(w_i)$ *in graph* $\mathcal{G}$*. The total surprisal is given by*

$$\mathcal{S}_{\mathcal{G}} = -\sum_{i=1}^{\mathrm{N}_{\mathcal{G}}} \log \mathrm{p}(w_i) \tag{4.5}$$

Since the multiplicative $\mathrm{p}(w_i)$ term is removed, this is no longer the expected surprisal, i.e.entropy in 4.4. Consequently, this quantity gives the surprisal of each parameter equal weight.[3]

4.4.4 Maximizing the Expected Spread Minimizes the Entropy

If parameters that affect smaller portions of the model (i.e. with a smaller *spread*) are considered more specific or *surprising*, then for optimal encoding of the horizontally unrolled graph expected spread would encourage assigning shorter bit lengths for more repeated components. For instance, parameters of the mainstream deep learning models have an exponential distribution where parameters in the earlier layers have exponentially larger numbers of contexts. For large models with the same number of parameters, this results in a much lower entropy compared to a uniform distribution which has the maximum possible entropy $\log_2 \mathrm{N}_{\mathcal{G}}$.

[3] Total surprisal can be considered a measure of descriptive ability. That is, models that have higher total surprisal can describe more complicated functions.

Overall, when other conditions such as model size, capacity, training data etc. are similar, the reusability prior encourages increasing the expected spread. This may lead to an improvement in parameter efficiency, as the entropy, i.e. the expected bit length, is reduced. In other words, unrolled graphs have smaller description lengths when there is a lot of repetition in the earlier layers: to reuse is to simplify.

4.4.4.1 Proofs for the Connections Between the Quantities for Model Comparison

Lemma 1. *Let $P(W)$ be the discrete probability distribution of parameters based on their number of contexts in graph G. Kullback-Leibler (KL) divergence $\mathrm{D_{KL}}(\mathrm{P}(W)||\mathrm{P}_U(W))$ from the discrete uniform distribution $\mathrm{P}_U(W)$ is equivalent to the expected spread.*

Proof. Let the relative frequency of w_i be $\mathrm{p}(w_i) = c_i/\mathrm{N}_\mathcal{C}$ where $c_i = |\mathcal{C}_{w_i}|$ and $\mathrm{N}_\mathcal{C} = \sum_{w_j \in \mathcal{G}} |\mathcal{C}_{w_j}|$. Then the KL divergence of the probability distribution of parameters based on the number of contexts $\mathrm{P}(W)$ from uniform distribution $\mathrm{P}_U(W)$ where $\mathrm{p}_U(w_i) = 1/\mathrm{N}_\mathcal{C}$ is given by

$$\mathrm{D_{KL}}(\mathrm{P}(W)||\mathrm{P}_U(W)) = \sum_{i=1}^{n} \mathrm{p}(w_i) \log_2(\mathrm{p}(w_i)/\mathrm{p}_U(w_i))$$

$$= \sum_{i=1}^{n} \mathrm{p}(w_i) \log_2(c_i/\mathrm{N}_\mathcal{C}/(1/\mathrm{N}_\mathcal{C})) = \sum_{i=1}^{n} \mathrm{p}(w_i) \log_2(c_i)$$

$$= \mathrm{E}[\![\log_2 |\mathcal{C}|]\!].$$

□

Note that, if $\mathrm{P}(W)$ is derived from the graph in Figure 4.3a by directly counting the frequencies from its unrolled version in Figure 4.3b, $\mathrm{P}_U(W)$ can correspond to the uniform graph in Figure 4.3c. In general, the cases where $\mathrm{p}(w_j) = 0$ and $\mathrm{p}_U(w_j) = 1/\mathrm{N}_\mathcal{C}$ do not change the summation.

Theorem 1. *Let $\mathrm{N}_\mathcal{C} = \sum_{w_j \in \mathcal{G}} |\mathcal{C}_{w_j}|$ be the total number of all contexts. $\forall \mathcal{G}_i$ when $\mathrm{N}_\mathcal{C}$ is held constant, maximizing the expected spread minimizes the entropy.*

Note that $\mathrm{N}_{\mathcal{C}}$ is equivalent to the number of parameters in G's horizontally unrolled *uniform* graph as in Figure 4.3c and 2. For graphs with identical unrolled uniform graphs, maximizing the expected spread is equivalent to parameter sharing [4]. For different architectures, as long as $\mathrm{N}_{\mathcal{C}}$ is held constant, the theorem still holds. The formal proof is as follows:

Proof. For any discrete probability distribution, it is already known that:

$$\mathrm{H}_{\mathrm{U}}(X) = \mathrm{H}(X) + \mathrm{D}_{\mathrm{KL}}(\mathrm{P}(X)||\mathrm{P}_U(X)).$$

Therefore, from lemma 1 it follows that KL divergence is always:

$$\mathrm{D}_{\mathrm{KL}}(\mathrm{P}(W)||\mathrm{P}_U(W)) = \mathrm{E}[\![\log_2 |\mathcal{C}|]\!]$$

Therefore:

$$\mathrm{H}_{\mathrm{U}}(W) = \mathrm{H}_{\mathcal{G}}(W) + \mathrm{D}_{\mathrm{KL}}(\mathrm{P}(W)||\mathrm{P}_U(W)).$$

This leads to the following equation that shows the relationship between the expected spread and entropy:

$$\mathrm{H}_{\mathcal{G}}(W) + \mathrm{E}[\![\log_2 |\mathcal{C}|]\!] = \log_2 \mathrm{N}_{\mathcal{C}}. \tag{4.6}$$

□

thus when $\mathrm{N}_{\mathcal{C}}$ is constant, reducing the entropy would increase the expected spread and vice versa.

4.4.5 Estimating Model Performance

Model performance is majorly associated with the number of parameters and the size of the training data. For CNNs the training image size is relevant as well. Yet it is often unclear how exactly some model designs consistently outperform others when

[4] If one takes into account the full scope of the reusability prior, e.g. the literature in Section 4.2.2, then diversifying input and diversifying the role of model components with regularization would also increase the expected spread. This is left for future work.

the number of parameters is similar as well as the training data and conditions. A consequence of the reusability prior is that by estimating the relative frequencies of each parameter, it is possible to quantify how the model design itself impacts the entropy, expected spread, and total surprisal. For the scenario where the training data and strategies such as regularization etc. are unchanged, the author proposes using total surprisal to predict model performance, with the assumption that when other conditions are similar, a model with a higher descriptive ability would perform better. This quantity is normalized for different model sizes and multiplied by the input size as follows:

Definition 6. *Let $\mathcal{S}_\mathcal{G}$ be the total surprisal of graph $\mathcal{G}$, N_I the total number of input nodes and $|\mathcal{G}|$ the summation of the total number of input, output and weight nodes. The estimated performance is given by*

$$\mathcal{P}_\mathcal{G} = \log_2\left(\mathcal{S}_\mathcal{G}\frac{\mathrm{N}_I}{|\mathcal{G}|}\right) \tag{4.7}$$

Note that the number of weight nodes can be larger than the number of parameters $\mathrm{N}_\mathcal{G}$ when explicitly sharing parameters.

4.4.5.1 An Alternative Estimation of Model Performance

As a potential alternative to the total surprisal in 4.7, expected spread multiplied by the number of learnable parameters $\mathrm{N}_\mathcal{G}$ can be used (i.e. replacing $\mathcal{S}_\mathcal{G}$ with $\mathrm{N}_\mathcal{G}\mathrm{E}[\![\log_2|\mathcal{C}|+1]\!]$). This alternative estimation is given by:

$$\mathcal{P'}_\mathcal{G} = \log_2\left(\mathrm{N}_\mathcal{G}\mathrm{E}[\![\log_2|\mathcal{C}|+1]\!]\frac{\mathrm{N}_I}{|\mathcal{G}|}\right) \tag{4.8}$$

Main justification for using expected spread is given in Section 4.4.2. For models with a comparable number of parameters and model size, models with a higher expected spread (i.e. with the ability to approximate more complicated functions) would likely

perform better after training until convergence. This can be observed in Table 4.2 for multiple experiments and datasets.

Overall, by relying on the graph based analysis and counting approach, two crude ways to estimate the performance of models without any training are proposed.[5] In practice, both have different strengths and weaknesses that are discussed in Section 4.6.

4.5 Methodology and Experiments

The computational graphs of EfficientNetv2[1] models are analyzed, without training, to estimate the quantities described in Section 4.4.3. For training models from scratch in the experiments, the same EfficientNetv2 models are adopted to explore the effects of aggressive parameter sharing. Then a comparison of empirical results with the results from the graph analysis based quantities is given. Tensorflow [197] was used for the experiments and a new Python [196] library was implemented for the graph based analysis [16].

4.5.1 Analyzing Computational Graphs

The predictions from the reusability prior for model graphs can be quantified as follows:

1. Estimate the relative frequencies of the learnable parameters in the horizontally unrolled computational graph of a given model (e.g. for the original graph in Figure 4.3a it is possible to estimate the relative frequencies from its unrolled form in Figure 4.3b. Please see A.3 for the full example.).
2. The resulting probability distribution allows deriving model-level quantities that are described in Section 4.4.3.

[5] Both estimation results are shared in Tables 4.3 and 4.4 as "P_G." and "P'_G"

3. Use total surprisal and expected spread for estimating model performance as described in Section 4.4.5.

In practice, since the EfficientNetv2 and ResNet-50 models have convolutional layers, the *full computational graph* needs to be taken into account. This depends on aspects such as image size, strided convolutions, batch normalization, and pooling layers. Thus in the calculations all learnable parameters were taken into account from convolutional kernels, batch normalizations, and final fully connected layers for classification which also have bias weights. The Python implementation of the graph analysis relies on imitating the computational graph of EfficientNetv2 and ResNet-50 models with layers that, instead of inference, count the aggregated number of contexts. This does not need to create a horizontally unrolled graph which would have been exponentially more expensive. Yet this approach still allows gathering the total number of contexts for each learnable parameter in a precise manner and calculating model-level quantities for comparison. The Github repo for the library is available here `https://github.com/gozepolat/priors/blob/main/reusability`.

4.5.2 EfficientNetv2

EfficientNetv2 models have a relatively high parameter efficiency, and they have competitive performance in image classification benchmarks. Would applying aggressive parameter sharing to an already compact model still improve parameter efficiency? To answer this question, multiple experiments on EfficientNetv2 models were conducted. The major focus was comparing EfficientNetv2-B0 and EfficientNev2-S models. For cross layer parameter sharing, the experiments revealed results in agreement with the literature discussed in Section 4.2.1. To make it easier to minimize confounding variables due to data and training strategies, the models that were trained from scratch were limited to only EfficientNetv2. The training was conducted in a controlled setting, spanning three different benchmarks. This helped eliminating the differences due to hardware and hyperparameters for the selected models.

4.5.3 Aggressive Parameter Sharing

For the EfficientNetv2 experiments, the official Tensorflow implementation [242] was modified to be able to optionally apply aggressive parameter sharing. This strategy relies on roughly treating weight matrices of the same shape as the same. One exception is that the batch normalization weights were not shared. This helped the training stay stable.

During the initialization of models with parameter sharing, a global *scope* dictionary kept track of all shared parameters. This way, convolutions of the same scope were only created once. Convolutional layers were represented as four-dimensional matrices where their dimensionality was defined by channel and kernel sizes. More precisely, each shared convolution was mapped to a scope name that is constructed by combining the number of input channels, number of output channels, kernel size, and strides.

4.5.4 Hyperparameters

For cross layer parameter sharing, a new hyperparameter which aggressively shares convolutions but uses separate batch normalization layers was introduced: `model.weight_sharing=all_but_bn`. *The only change* that was made to the hyperparameters when comparing an original model with one that shares convolutional layers is that `model.weight_sharing=None` for the original one. The default hyperparameters for EfficientNetv2 given by Tan et al [1] were closely followed, except for the following changes:

- All models were trained from scratch with CIFAR-10, instead of transfer learning. The models were trained for 300 epochs without any training stages or augmentation. The following hyperparameters were used:

  ```
  train_epoch=300, batch_size=512, data.ibase=32,
  ```

```
train.lr_warmup_epoch=5, train.lr_sched=exponential,
data.mixup_alpha=0, data.cutmix_alpha=0,
train.lr_base=0.016, model.bn_momentum=0.99.
```

- For all models trained with CIFAR-100, the same scenario and hyperparameters were followed, except for the data augmentation:

```
train_epoch=300, batch_size=512, data.ibase=32,
train.lr_sched=exponential, train.lr_warmup_epoch=5,
data.augname=autoaug, train.lr_base=0.016,
model.bn_momentum=0.99.
```

- For all models trained with ImageNet-1K, the following hyperparameters were used:

```
train_epoch=352, batch_size=400,
train.stages=4, train.lr_sched=exponential,
model.dropout_rate=0.075, train.lr_warmup_epoch=5,
data.ram=2, train.ema_decay=0.9999,
train.lr_base=0.025, model.bn_momentum=0.99,
data.augname=randaug.
```

The changes in the batch size and learning rate are necessary to be able to train the models with smaller GPUs (i.e. reduce the batch size and learning rate based on the same ratio). Note that, in the described setting, when trained from scratch, lower performance for both the EfficientNetv2-B0 and EfficientNetv2-S models was observed compared to [1]. Larger batch sizes and training longer can sometimes improve performance but a substantial gap in performance was observed between V2-S-shared vs V2-B0-original models regardless. In the previous experiments with ImageNet-1K, different hyperparameters such as gclip, batch size, as well as a larger number of training epochs were tested. Moreover, for CIFAR-100 models were also compared using no augmentation. None of these changes in the hyperparameters changed the ranking of the models in terms of top-1 accuracy given in Table 4.2.

4.5.5 Datasets

In the experiments, the models were trained *from scratch* using the following well known object classification benchmarks for visual recognition as given in Table 4.1, which consists of CIFAR-10, CIFAR-100, and Imagenet Large Scale Visual Recognition Challenge (ILSVRC2012, also referred to as Imagenet-1K).

Table 4.1: Details on CIFAR and ImageNet-1K datasets

Dataset	Train	Validation	Classes
CIFAR-10[14]	50000	10000	10
CIFAR-100[76]	50000	10000	100
ImageNet-1K [243][201]	1,28M	50000	1000

4.5.6 Experiment Results from Aggressive Parameter Sharing

When EfficientNetv2 models are trained from scratch, a consistent increase was observed in the parameter efficiency for the V2-S models with aggressive parameter sharing compared to V2-B0 which has significantly more parameters, as given in Table 4.2.

Overall, at the cost of additional FLOPs, parameter efficiency was improved for all three benchmarks. Alongside the research regarding parameter sharing in the literature discussed in Section 4.2.1, and combined with the graph analysis results in Table 4.3, these results provide supporting evidence for the reusability prior.

4.5.7 Predictions from Graph Analysis vs Experiment Results

In the experiments with EfficientNetv2 models, V2-S-shared models consistently performed better than the V2-B0 models, despite having a significantly lower number of

Table 4.2: EfficientNetV2 [1] trained from scratch on CIFAR-10, CIFAR-100, and ImageNet-1K. At the cost of more FLOPs, V2-S-shared models that aggressively apply cross layer parameter sharing (bold) achieves better top-1 accuracy scores than V2-B0 models that have at least 60% more parameters.

Model	Dataset	Params	Score	Augm.	Batch	Epoch	FlOPS
V2-B0-shared	CIFAR-10	1.7 M	95.3	None	512	300	0.7B
V2-B0-original	CIFAR-10	5.9 M	95.6	None	512	300	0.7B
V2-S-shared	CIFAR-10	**3.1 M**	**96.0**	None	512	300	8.8B
V2-B0-shared	CIFAR-100	1.8 M	78.8	AutoAug[244]	512	300	0.7B
V2-B0-original	CIFAR-100	5.9 M	80.6	AutoAug	512	300	0.7B
V2-S-shared	CIFAR-100	**3.2 M**	**81.5**	AutoAug	510	300	8.8B
V2-B0-shared	ImageNet-1K	3.0 M	73.7	RandAug[245]	400	352	0.7B
V2-B0-original	ImageNet-1K	7.1 M	76.9	RandAug	400	352	0.7B
V2-S-shared	ImageNet-1K	**4.4 M**	**78.3**	RandAug	400	352	8.8B

parameters. In Table 4.3 the results from Imagenet-1K experiments were compared with the results from the analysis of the computational graphs of V2-B0 and V2-S models with and without parameter sharing. Unlike a naive prediction which would directly correlate the number of parameters with performance, the proposed graph analysis method assigned a higher score for the *V2-S-shared* model.

4.5.8 Graph Analysis Results for Larger Models

The graph analysis based quantities were also tested for larger models that were trained by Tan et al. in [1].[6] In Table 4.4 the estimated performance and expected spread of ResNet is lower, while its entropy and number of parameters are significantly higher than V2-B3 and V2-S models. Coincidentally, for the given models, the performance estimations were roughly $1/3$rd of the actual top-1 accuracies, but

[6] Note that, compared to the experiments shared in this work, V2-S model has a higher top-1 accuracy of 83.6-83.9 in their experiments. This does not change the comparison.

Table 4.3: Analysis without training vs the new experiment results from ImageNet-1K. Note that a naive prediction would correlate the given top-1 accuracy scores directly with the number of parameters. The introduced performance estimations based on total surprisal and expected spread (P_G and P'_G respectively) instead correctly favor V2-S-shared (bold) compared to V2-B0.

Model	Score	P_G	P'_G	Exp. spread	Entropy	Params	Img.
V2-B0-shared	73.7	25.00	25.44	746.608+	10.2	3.0 M	224
V2-B0	76.9	26.28	26.72	746.608	10.2	7.1M	224
V2-S-shared	**78.3**	**26.33**	**26.97**	1505.547	9.8	**4.4M**	384
V2-S	83.2	28.72	29.28	1505.546	9.8	21.6M	384

unlike the top-1 accuracy, the performance estimation scores are not bounded (e.g. P_G can be negative).

4.6 Limitations and Future Work

The reusability prior encompasses design, training, and data aspects in deep learning. In this work, however, the major focus was on the design aspect. Rather than considering *concrete* contexts that can arise due to the concrete samples from the training data itself, and the contexts were counted based on the computational graph. That is, when counting the number of contexts, only what can arise due to the computational graph of the model as well as the input size was considered. Generalizing the idea of context to take into account data and training techniques such as regularization or comparing different definitions of context is a future research direction. Another direction is to investigate potential analogies between *microstates* from statistical mechanics and the definition of context that was given in this chapter.

For larger models, additional experiments with cross layer parameter sharing were

Table 4.4: Analysis without training vs some of the top-1 accuracy scores for larger models [1] for ImageNet-1K. Note that a naive prediction would correlate the given top-1 accuracy scores directly with the number of parameters. The graph analysis based estimations instead assign estimated performances based on total surprisal and expected spread (P_G and P'_G respectively) to ResNet-50 (bold) lower than V2-B3 and V2-S models which both have a smaller number of parameters.

Model	Score	P_G	P'_G	Exp. spread	Entropy	Params	Img.
V2-B1	79.80	26.75	27.20	913.4	10.2	8.2 M	240
ResNet-50	**80.30**	**27.25**	**27.60**	**465.4**	13.3	**25.6 M**	380
V2-B3	82.10	27.71	28.20	1167.2	10.6	14.5 M	300
V2-S	83.2	28.72	29.28	1505.546	9.8	21.6M	384
V2-M	85.10	29.9+	30.4+	2171.1	9.8	54.4 M	480
V2-L	85.70	30.5+	30.9+	3095.4	10.3	119.0 M	480

omitted. From the perspective of reusability prior, all models already explicitly (e.g. 1x1 convolutions) or implicitly (e.g. skip connections) share parameters. Therefore, a comparison of existing scores from [1] combined with predictions from the graph based analysis in Table 4.4 was still able to demonstrate the predictive power of the graph based analysis.

An important research direction is to test the reusability prior with neural architecture search. In practice, additional experiments will be likely necessary to first derive a utility function that aligns well with major practical concerns that may be task specific. For instance, the introduced performance estimation approaches do not necessarily optimize for compute resources. For the same number of parameters, both approaches are biased towards deeper and narrower models, since the proposed counting approach does not use any discount factor for the path length of a given context. Each context is counted as one, regardless of the distance between a given parameter and a target node. During graph analysis, incrementing the total number of contexts

by a fractional number instead of incrementing by one may be an important improvement. Furthermore, total surprisal based performance estimation would penalize very deep models with cross layer parameter sharing for not being descriptive enough for their large size. For shallower but exponentially larger models, the expected spread based estimation would severely penalize them for not reusing parameters enough. Overall, concerns including compute limitations, model size, FLOPs, and latency may be relevant for finding the most appropriate utility function. If such a function can be created by relying on the quantities or the graph analysis based framework, then *without any training*, searching viable model designs by solely relying on graph analysis can be an interesting direction.

4.7 Conclusion

Not all performance can be explained directly with the number of parameters and the training data. The reusability prior was introduced to point towards a deeper reason. First it was demonstrated that, either explicitly or implicitly, all mainstream deep learning models reuse parameters. Then a generalized notion of reusability was introduced. It encompassed aspects such as training, data, and model design that affect the number of *contexts* with which model components have to function. Focusing on the model design aspect, for model comparison, the model level quantities were defined, namely entropy, expected spread, and total surprisal which rely on analyzing the computational graph of a model. Formal proof showed that maximizing the expected number of contexts for model components minimizes the entropy when the total number of available contexts is the same.

To test the described approach in practice, two crude performance estimation approaches were proposed based on total surprisal and expected spread. Then EfficientNetv2 models were compared by training them from scratch with and without cross layer parameter sharing. A naive approach would have correlated models with a lower number of parameters with lower performance. A counter-example was given

where EfficientNetv2-S models with parameter sharing outperformed the baseline EfficientNetv2-b0 models which have at least 60% more parameters. Another edge case with ResNet-50 showed that despite having significantly more parameters, it underperformed compared to EfficientNetv2-B3 and V2-S models. The introduced graph based estimations of performance gave appropriate scores for both cases, correctly ranking ResNet-50 below these models in terms of performance.

In the model analysis based experiments, the counting approach allowed calculating model-level quantities for comparison, consequently correctly predicting the rank of all models in terms of top-1 accuracy. In contrast, the naive approach of relying on the number of learnable parameters failed to correctly rank the models of varying parameter efficiency (i.e. Tables 4.3 and 4.4). In practice, as discussed in Section 4.6, the reusability prior and the proposed framework may lead to new approaches for neural architecture search, or help researchers improve their model design in the right direction before any training is done.

Future work with further experiments and analysis will reveal whether the graph based approach for estimating performance is generalizable to more models. Yet, for the models that were investigated, this approach was able to correctly delineate at least two important edge cases in Tables 4.3 and 4.4. Furthermore, the introduced quantities based on the reusability prior aligned well with the experiment results as well as the existing results from the literature for larger models. For estimating these quantities the model graphs were sufficient, and *no training* was required.

In conlcusion, the reusability prior provides a viable research direction for connecting different aspects of deep learning under the same framework that is majorly based on a very simple idea: counting the number of *contexts* for model parameters. This may lead to further research and important predictions on how deep learning models may be affected by different design, augmentation, and training choices.

4.8 Data Availability Statement

The data that support the findings of this study are openly available at [16].

CHAPTER 5

ENERGY PRIORS: A STATISTICAL MECHANICS FRAMEWORK FOR MODEL COMPARISON

5.1 Introduction

Entropy appears in a diverse set of disciplines such as thermodynamics, *statistical mechanics*, and information theory. In the context of machine learning, various forms of entropy are used for the analysis (e.g. Shannon entropy [232]) and comparison (e.g. relative entropy [241]) of probability distributions.

The major contribution of this chapter is a deeper analysis of the notion of reusability, previously discussed in Section 4.4, leading to a generalization based on a connection between microstates from statistical mechanics and *contexts* described in Section 4.1.2. The term "energy prior" is introduced to describe energy in the form of $\ln |C'_w|$ assigned to each learnable parameter. Note that $|C'_w|$ is a custom score which is a generalization of the number of contexts that was previously utilized in Section 4.4.5. The energy prior-based understanding of model comparison allowed proving the convergence of entropy (previously defined in Section 4.4) for MLPs with constant width, as well as describing the behavior of how the performance estimation grows with depth and width *without relying on any graph analysis*.

In this chapter, after understanding the behavior of the context-based performance estimation by comparing the extreme cases of depths vs. widths (Section 5.2.1) as well as arbitrary models with and without cross-layer parameter sharing (Section 5.2.2),

a generalization with energy priors is introduced (Section 5.3). After giving relevant lemmas, theorems, and proofs for the analogy between energy priors and the reusability prior, a further generalization that went beyond the reusability prior is introduced. The representation of a hypothetical prior in terms of energy using this generalization is exemplified in Section 5.4.

5.2 Understanding the Extremes

The previous chapter provided supporting evidence for the reusability prior by analyzing real and high-performing DL models designed for the object classification problem (i.e. Section 4.5.6). In this section, instead, hypothetical models are used for understanding the general behavior of the proposed graph analysis approach in the more extreme cases.

5.2.1 Depth vs. Width Trade off

Vanilla MLP and CNN models were analyzed for depth vs. width trade-off in Tables 5.1 and 5.3 by holding the number of parameters constant. For the sake of simplicity, no batch normalization or bias parameters were included. The model designs consisted of layers of the same width for a given model. This allowed having a constant number of parameters when comparing models of different depths and layer widths. Note that the graph analysis approach ignores practical constraints such as the limitations of floating point representations in a narrow layer, or the computational inefficiency of deeper models. The results showed that both performance estimation approaches majorly favored *deeper models* for both MLP and CNN models.

5.2.2 Cross Layer Parameter Sharing with Batch Normalization

The described graph analysis approach in the previous chapter (Section 4.4) can be used for comparing and analyzing arbitrary model DAGs where the number of parameters is not fixed. The estimated performance scores can be converted into parameter efficiency as follows:

$$PE_G = P_G/N_G \tag{5.1}$$

where P_G is a performance estimation described in Section 4.4.5, and N_G is the number of learnable parameters. For MLPs, compute efficiency can be roughly estimated as:

$$CE_G = P_G/W_G \tag{5.2}$$

where W_G is the number of weight nodes in the model.[1] Note that parameter and compute efficiencies for high performance models would naturally drop due to diminishing returns.

For vanilla models with repeating layers and no batch normalization, cross-layer parameter sharing seems to negatively impact the parameter efficiency of MLP and CNNs. [2] Adding batch normalization changed this prediction in a nontrivial manner. In Table 5.2, some of the MLP models with cross-layer parameter sharing have high parameter efficiency, while others do not (e.g. first and last rows respectively). This is because the learnable parameters from batch normalization led to a depth-width ratio where parameter efficiency is maximized. That is, if the model was too deep, the number of learnable parameters in the batch normalization layers started to

[1] Note that, when there is no cross-layer parameter sharing $N_G = W_G$, therefore compute efficiency would be equivalent to the parameter efficiency. For this reason, in Table5.1 compute efficiency is skipped.

[2] Note that cross-layer parameter sharing is merely an explicit form of parameter sharing. All DL models implicitly share parameters as mentioned in Section 4.4. Furthermore, the parameter efficiency for high performance models would naturally drop due to diminishing returns.

become significant and negatively impacted parameter efficiency. If the model was too shallow, the estimated performance was reduced. In larger or other types of models, similar depth vs. width trade-offs seem to exist (e.g. [158], as well as Table 5.4 where the models with the highest *parameter efficiency* estimation are not the deepest ones with the top estimated performance. Intuitively, as depth is increased, there should be a point where parameter efficiency significantly drops due to diminishing returns. For instance to increase the performance of a model for a small percentage, the model size may have to be doubled after a certain threshold of performance is reached.)

5.2.3 Limitations of the Given Counting Approach

The counting approach introduced in the previous chapter may have certain limitations in practice. For instance, there is a strong bias toward deeper and narrower models when the number of parameters is fixed (Section 5.2). In terms of expressive power, compared to their shallow counterparts with the same number of parameters, shallower models indeed may perform worse (e.g. [21]). The expected spread (4.2) similarly benefits depth more than layer width. In fact, for an MLP-like model with constant layer width, the number of contexts for a given node grows with $O(width^{depth})$ as discussed in A.2), therefore the expected spread would likely grow linearly with depth and logarithmically with width (i.e. $\ln |\mathcal{C}_{\sqsupseteq_\rangle}| = depth \ln width$). This is demonstrated in Figure 5.1 for MLPs of various depths and widths. The performance estimation approaches given in Section 4.4.5 both favor very narrow and deep models. This is demonstrated in Table 5.1 for MLP and Table 5.3 for CNN models. In Figure 5.2 the performance estimation grows sublinearly within the given depth and width intervals where the top model is the widest and the deepest. Depth seems to increase the performance estimation faster. In practice, there can be *compute* and parameter budgets to consider. A very narrow model is not compute efficient, and may not be optimal for performance either. Due to floating point limitations, precision and channel noise can become a problem. Another issue with very deep DL

models is that they are harder and slower to train as there are more sequences of matrix multiplications.

5.3 A Generalization with Energy Priors

As discussed in Sections 4.6 and 5.2, the introduced methodology for graph analysis may have too much bias towards depth since very deep DL models have practical limitations. In its given form, the counting approach does not allow differentiating between contexts of different path lengths. Moreover, very deep DL models may be overly susceptible to the design of the first few layers.

For a given set of assumptions and constraints (i.e. *energy prior*), it is possible to directly assign a *score* $|C'_{w_i}|$ to each learnable parameter w_i (e.g. potentially by modifying the original counting approach), and interpret $\ln |C'_{w_i}|$ as energy E_{w_i}. In such a scenario, an analogy between microstates from statistical mechanics and contexts can be established.

5.3.1 Microstates vs. Contexts

Microstates in statistical mechanics are used similarly to *contexts* defined in Section 4.1.2. Statistical mechanics predict the macrostates of a given system by relying on the total number of microstates for different energy levels. In this way, estimation of macrostates such as entropy and the expected energy level of the whole system becomes possible. Moreover, in thermodynamics, this can be used for the prediction of physical quantities such as pressure, temperature, and volume. Analogously, by relying on the number of contexts (microstates) for each learnable parameter (energy level), an estimation of various model level quantities (macrostates) was made possible in the previous chapter (i.e. Section 4.4). To understand this analogy more rigorously, first, it is necessary to refer to the partition function from statistical mechanics.

5.3.2 The Partition Function

In statistical mechanics, for a classical and discrete system, the partition function is often defined as follows:

$$\mathrm{Z} = \sum_{i=1} \mathrm{e}^{-\beta \mathrm{E}_i} \tag{5.3}$$

where E_i corresponds to the energy, and β a constant which is equivalent to the inverse temperature $1/T$ for a "canonical ensemble" [246] in thermal equilibrium. For the *degenerate* case where $E_i = E_k$ it is often given as:

$$\mathrm{Z} = \sum_{j=1} g_j \mathrm{e}^{-\beta \mathrm{E}_j} \tag{5.4}$$

where g_j is the number of duplicate energy levels. The majority of the modern DL models can be described using the degenerate case.

5.3.2.1 The Partition Function for MLP

For a vanilla MLP architecture with depth D and only fully connected layers, the partition function for the *original counting approach* (Section 4.4) can be written as:

$$\mathrm{Z} = \sum_{d=1}^{\mathrm{D}} g_d \mathrm{e}^{-\beta \ln |\mathcal{C}_{w_d}|} \tag{5.5}$$

where g_d is the number of learnable parameters for the layer at depth d (i.e. the same energy level), and $|\mathcal{C}_{w_d}|$ is the number of contexts for any parameter w at the given depth d. If there is no bias term, and the width ω of the layers is constant, $g_d = \omega^2$, and $|\mathcal{C}_{w_d}| = \omega^d$. When the bias term is taken into account $g_d = \omega(\omega + 1)$, the number of contexts for parameter w is unchanged.

5.3.3 Spread as Energy

Based on how the *expected* spread[3] was introduced in 4.2, *spread* for a learnable parameter can be defined as:

Definition 7. *For a given model graph $\mathcal{G}$'s learnable parameter w_i, the spread is given by*

$$\mathrm{E}_i = \ln |\mathcal{C}'_{w_i}| \tag{5.6}$$

where $|\mathcal{C}'_{w_i}|$ is a score assigned to w_i.

Note that, for the original counting approach described in the previous chapter (Section 4.4) the score would be equivalent to $|\mathcal{C}_{w_i}|$ which is the cardinality of the set of all contexts for the learnable parameter w_i.

5.3.4 Neural Networks with Negative Temperature, T=-1

If spread is directly used as E_i in z (5.4), the following proposed lemma holds:

Lemma 2. *Let E_j be the spread of a learnable parameter w_j in graph $\mathcal{G}$ and g_j be the number of parameters with the same score. Then the total score $\mathrm{N}_{\mathcal{C}'} = \sum_j g_j |\mathcal{C}'_{w_j}|$, is equivalent to the partition function z in 5.4 with $\beta = -1$.*

Proof. Since $e^{\ln x} = x$, plugging $E_i = \ln |\mathcal{C}'_{w_i}|$ and $\beta = -1$ into the partition function $z = \sum_{j=1} g_j \mathrm{e}^{-\beta \mathrm{E}_j}$ leads to $z = \sum_{j=1} g_j |\mathcal{C}'_{w_j}|$ which is equivalent to $\mathrm{N}_{\mathcal{C}'}$. □

Note that negative absolute temperature is consistent with equilibrium thermodynamics, and is a real phenomenon "for some physical systems as incompressible fluids,

[3] Note that compared to the previous chapter, for the sake of mathematical convenience $\log_2$ was replaced by $\ln$, but this only leads to a constant conversion factor in practice.

nuclear magnetic chains, lasers, cold atoms and optical waveguides" [247]. In statistical mechanics, the partition function z is used for deriving the probabilities as follows in a Boltzmann distribution [248]:

$$p(w_i) = \frac{1}{z} e^{-\beta E_i} \tag{5.7}$$

where E_i is the energy. This leads to the following proposed theorem:

Theorem 2. *For any relative frequency $p(w_i)$ which is derived from using a score $|C'_{w_i}|$, assigning an energy in the form of $E_i = \ln |C'_{w_i}|$ to a learnable parameter w_i in graph $\mathcal{G}$ results in a Maxwell-Boltzmann distribution (5.7) with $\beta = -1$,*

Proof. For any arbitrary counting approach, the total summation is calculated by $z = \sum_{j=1} g_j |C'_{w_j}|$. From Lemma 2 this is equivalent to the partition function z, and $p(w_j)$ is estimated as the relative frequency $1/z|C'_{w_j}|$. Replacing $|C'_{w_j}|$ with $e^{\ln |C'_{w_j}|}$ gives $p(w_j) = 1/z e^{-\beta E_j}$ where $E_j = \ln |C'_{w_j}|$ and $\beta = -1$. □

Corollary 1. *When the expected spread from 4.2 is redefined with* $\ln$ *instead of* $\log_2$ *with a custom scoring $|C'|$ as:*

$$\mathrm{E}[\![\ln |\mathcal{C}'|]\!] =< \ln |\mathcal{C}'| >= \sum_{i=1}^{\mathrm{N}_\mathcal{G}} \mathrm{p}(w_i) \ln |\mathcal{C}'_{w_i}| \tag{5.8}$$

the expected spread becomes equivalent to the expected energy $< E >=< \ln |\mathcal{C}'| >$ since $E_i = ln|\mathcal{C}'_{w_i}|$ and $< E >= \sum_{i=1}^{\mathrm{N}_\mathcal{G}} \mathrm{p}(w_i) E_i$.

Note that the introduced theorem and assigning energies in the form of $\ln |C'_{w_j}|$ to learnable parameters allow utilizing the tools of statistical mechanics for the analysis of computational graphs of DL models. To demonstrate the advantage of such an approach in practice, in the next section, formulas for entropy, expected spread, and performance estimation of simple MLP models with constant width are derived.

5.3.5 Statistical Mechanics of MLP Graphs with Constant Width

Lemma 3. *For MLPs of depth* D *and constant layer width* ω*, the total number of contexts* $\mathrm{N}_{\mathcal{C}}$ *with the original counting approach is equivalent to the following:*

$$\mathrm{N}_{\mathcal{C}mlp} = g\frac{\omega^{\mathrm{D}} - 1}{\omega - 1} = \sum_{d=1}^{\mathrm{D}} g_d \mathrm{e}^{-\beta \ln |\mathcal{C}_{w_d}|} = z \tag{5.9}$$

where $g_d = g$ *since the width is constant, and* $g = (\omega + 1)\omega$ *with bias. And the probabilities are:*

$$p(w_i)_{mlp} = \frac{1}{z}\omega^{d_i} \tag{5.10}$$

Proof. The equivalence to z follows from Lemma 2 and from 5.5. The probabilities are:

$$p(w_i)_{mlp} = \frac{\omega^{d_i}(\omega - 1)}{(\omega^{\mathrm{D}} - 1)g} = \frac{1}{\mathrm{N}_{\mathcal{C}mlp}}\omega^{d_i} \tag{5.11}$$

where d_i is the path length from parameter w_i to the output layer. When the next node is the output layer, $d_i = 0$. Since $|\mathcal{C}_{w_d}| = \omega^{d_i}$, the same probability can be derived using the Maxwell-Boltzmann distribution instead by plugging $E_i = \ln |\mathcal{C}_{w_i}| = d_i \ln \omega$ and $\beta = -1$. Overall, from Lemma 2 it follows that $p(w_i) = e^{-\beta E_i}/z = e^{\ln \omega^{d_i}}/z = \omega^{d_i}/\mathrm{N}_{\mathcal{C}mlp}$. □

5.3.5.1 Expected Energy of MLP with Constant Width

The following theorem is introduced by the author for the expected energy of MLP with constant width:

Theorem 3. *For an MLP with constant width* ω *and depth* D*, when the original counting approach is used, the expected energy (i.e. the expected spread) is given by:*

$$< \ln |\mathcal{C}| >_{mlp} = \left(\frac{\mathrm{D}\omega^{\mathrm{D}-1}}{\omega^{\mathrm{D}} - 1} - \frac{1}{\omega - 1} \right) \omega \ln \omega. \tag{5.12}$$

Proof. Plugging the probabilities from Lemma 3 into the expected energy in 5.8 using the original counting approach results in: $< \ln |\mathcal{C}| >= \sum_{i=1}^{\mathrm{N}_\mathcal{G}} \mathrm{p}(w_i) \ln |\mathcal{C}_{w_i}| = \sum_{i=1}^{\mathrm{N}_\mathcal{G}} \omega^{d_i} / (g \frac{\omega^{\mathrm{D}}-1}{\omega-1}) \ln \omega^{d_i}$. Since $\mathrm{N}_\mathcal{G} = g\mathrm{D}$ for a vanilla MLP, this leads to the following equation for the expected energy of an MLP with constant width:

$$< \ln |\mathcal{C}| >_{mlp} = \sum_{j=1}^{\mathrm{D}} g\omega^{j-1} / (g \frac{\omega^{\mathrm{D}} - 1}{\omega - 1}) \ln \omega^{j-1} = \frac{\omega - 1}{\omega^{\mathrm{D}} - 1} \omega \ln \omega \sum_{j=1}^{\mathrm{D}} (j-1)\omega^{j-2} \tag{5.13}$$

Notice that $z/g = \mathrm{N}_{\mathcal{C}mlp}/g = \sum_{j=1}^{\mathrm{D}} \omega^{j-1}$, therefore:

$$\sum_{j=1}^{\mathrm{D}} (j-1)\omega^{j-2} = \frac{\partial (z/g)}{\partial \omega}. \tag{5.14}$$

Furthermore:

$$z/g = \sum_{j=1}^{\mathrm{D}} \omega^{j-1} = \frac{\omega^{\mathrm{D}} - 1}{\omega - 1}.$$

Thus the partial derivative is equal to:

$$\frac{\partial (z/g)}{\partial \omega} = \frac{(1 - \omega^{\mathrm{D}})}{(\omega - 1)^2} + \frac{\mathrm{D}\omega^{\mathrm{D}-1}}{(\omega - 1)}.$$

Plugging this result from the partial derivative in place of 5.14 into 5.13 gives the expected energy:

$$< E >=< \ln |\mathcal{C}| >_{mlp} = \left(\frac{\mathrm{D}\omega^{\mathrm{D}-1}}{\omega^{\mathrm{D}} - 1} - \frac{1}{\omega - 1} \right) \omega \ln \omega$$

□

Note that in 5.12 replacing $\ln$ with $\log_2$ would exactly match the definition of expected spread from the previous chapter (4.2). A potential limitation to notice is this quantity is *independent* from g. For instance, bias terms do not change the expected energy.

5.3.5.2 Growth of the Expected Energy with Depth

Figure 5.1 demonstrates that the expected energy (i.e. expected spread) should roughly grow linearly with depth, and logarithmically with width. The results were from the graph analysis-based approach. Yet, this can be already predicted without any graph analysis.

Theorem 4. *Let depth* D *of an MLP grow and width* ω *stay constant. The expected energy (given in 5.12) converges to* $(\mathrm{D}-1)\ln\omega$.

Proof.

$$< E >= \left(\frac{\mathrm{D}\omega^{\mathrm{D}-1}}{\omega^{\mathrm{D}}-1} - \frac{1}{\omega-1}\right)\omega\ln\omega \approx \frac{(\mathrm{D}-1)}{\omega}\omega\ln\omega = (\mathrm{D}-1)\ln\omega \tag{5.15}$$

□

5.3.5.3 Entropy of MLP with Constant Width

Corollary 2. *The following identity is well known in statistical mechanics (e.g. [249]):*

$$H(W) = \ln z + \beta < E > \tag{5.16}$$

where $H(W)$ *is the entropy,* **<E>** *the expected energy, and* z *the partition function. By plugging* $\beta = -1$ *(Lemma 2), partition function* z *for MLPs with constant width (Lemma 3), and the expected energy for MLPs with constant width (Theorem 3), the entropy for MLP with constant width follows as:*

$$H(W)_{mlp} = \ln\left(g\frac{\omega^{\mathrm{D}}-1}{\omega-1}\right) - \left(\frac{\mathrm{D}\omega^{\mathrm{D}-1}}{\omega^{\mathrm{D}}-1} - \frac{1}{\omega-1}\right)\omega\ln\omega \tag{5.17}$$

Note that, unlike the expected energy, the entropy changes when there is a bias term. Moreover, it is important to note the equivalence between 5.16 and 4.6 since the expected spread and expected energy are the same.

5.3.5.4 Convergence of the Entropy with Depth

The following theorem is introduced by the author to demonstrate that entropy given in 5.17 stays constant as an MLP with constant width goes deeper:

Theorem 5. *Let g be the number of learnable parameters per layer, ω the layer width, and D depth. Let D grow and ω stay constant. The entropy given in 5.17 approximately converges to a constant:* $\ln g + \ln w - \ln(\omega - 1)$.

Proof. As depth becomes much greater than width ($\mathrm{D} >> \omega$), the expected energy converges to $(\mathrm{D}-1)\ln\omega$ (Theorem 4), therefore entropy in 5.17 takes the following form:

$$H(W)_{mlp} \approx \ln\big(g(\omega^{\mathrm{D}} - 1)/(\omega - 1)\big) - (\mathrm{D}-1)\ln\omega.$$

Approximating $\ln(\omega^{\mathrm{D}} - 1)$ as $\mathrm{D}\ln\omega$ results in:

$$H(W)_{mlp} \approx \ln g + D\ln\omega - \ln(w-1) - (\mathrm{D}-1)\ln\omega \approx \ln g + \ln w - \ln(w-1). \tag{5.18}$$

Since $\ln g + \ln w - \ln(w-1)$ does not depend on depth the entropy approximately converges to this constant. □

Note that for a vanilla MLP architecture with bias $g = (\omega+1)\omega$, and the converged constant becomes: $2\ln\omega + \ln(\omega+1) - \ln(\omega-1)$.

5.3.5.5 Expected Energy Based Performance Estimation Score for MLP with Constant Width

For MLPs with constant width ω, plugging the expected energy $< E >$ given in 5.12 to the previously proposed estimation approach in 4.8 with $N_{\mathcal{G}} = g\mathrm{D}$, $N_I = \omega$ and

$N_O = \omega$ results in the following estimation:

$$\mathcal{P}'_{\mathcal{G}_{mlp}} = \ln\left((g\mathrm{D} < E > +1)\frac{\omega}{2\omega + g\mathrm{D}}\right). \tag{5.19}$$

Since for large depths $< E > \approx (\mathrm{D}-1)\ln\omega$ (Theorem 4), for deeper models this approximately becomes:

$$\mathcal{P}'_{\mathcal{G}_{mlp}} = \ln\left((g\mathrm{D}(\mathrm{D}-1)\ln\omega + 1)\frac{\omega}{2\omega + g\mathrm{D}}\right) \approx \ln D + \ln\omega + \ln(\ln\omega). \tag{5.20}$$

Overall, the described quantities were fairly easily derived with insights from statistical mechanics. The approximations provided predictions about how the described quantities grow with depth without any need for graph analysis. In practice, these quantities were compared against the graph analysis method and they match all sufficiently large models. For future work, the same approach can be extended to other models such as CNNs. The main limitation of this approach is that it would be hard to work without simplifications or ideal conditions such as constant width. Yet, when the analyzed models have constant width, these equations provide a highly efficient way of estimation.

5.4 Arbitrary Energy Priors and Temperatures

The counting approach given in the previous chapter was based on the reusability prior, and to estimate the performance of a model, the main approach relied on essentially assigning scores to each learnable parameter by calculating their number of contexts and taking the logarithm, i.e. $\ln|\mathcal{C}_{w_i}|$. In this chapter, an analogy from the perspective of statistical mechanics was introduced, and $\ln|\mathcal{C}_{w_i}|$ was interpreted as energy E_i with temperature $T = 1/\beta = -1$. Additional insights from statistical mechanics were used for a deeper understanding and simplifications. For instance, utilizing a partition function and treating probabilities as $p(w_i) = 1/ze^{-\beta E_i}$ made the

theorems and their proofs easier to derive. A deeper analysis of MLP models with constant width in this manner eliminated the need for any graph analysis for such models.

The proposed counting approach can be modified or E_i can be calculated with various practical constraints in mind such as compute limitations and limitations in floating point precision. If encoding the relevant assumptions and constraints in E_i (i.e. arbitrary energy prior) is possible, the statistical mechanics-based framework introduced in this chapter for graph comparison can be reused. More precisely, the new partition function based on E_i can be calculated, and expected energy can be integrated into the original performance estimation approach.

5.4.1 Comparing Models with a Hypothetical Prior

In this section, a simple arbitrary modification to the given methodology is exemplified. The temperature for the previously described approaches was always -1. One major reason for this is the associated probabilities were not inversely proportional to the energy. A hypothetical situation can be given for a new case for MLPs with constant width where a new prior was used so that the probability assigned to a learnable parameter was instead proportional to ω^{-j}. Then the framework described in this chapter with the new prior can be reused as follows

- Probability: $p(w_j) = e^{-\beta E_j}/z = \omega^{-j}/z$.
- Partition function: $z = \sum_i e^{-\beta E_i} = \sum_{j=0}^{\mathrm{D}-1} g\omega^{-j}$.
- Temperature: since $\omega^{-j} = e^{-\beta E_j}$ due to how $p(w_j)$ was defined, $e^{-j\ln\omega} = e^{-\beta E_j}$, and E_j must have a term with j for non uniform distributions, therefore, if $E_j = j$ then $T = 1/\ln\omega$, and $\beta = \ln\omega$.
- Expected energy: $< E >= \sum_i p(w_i)E_i = \sum_{j=0}^{\mathrm{D}-1} je^{-j\ln\omega}g/z = \sum_{j=0}^{\mathrm{D}-1} jw^{-j}g/z$.

- Entropy: $H(W) = \ln z + \beta < E >= \ln z + \ln \omega \sum_{j=0}^{\mathrm{D}-1} j w^{-j} g / z$.
- Performance estimation (for $N_I = \omega = N_O$): $\ln(g\mathrm{D} < E > +1)\omega/(2\omega + g\mathrm{D})$.

Overall, the same methodology with a different prior was exemplified. The performance estimation would allow comparison between models.[4] For a more general approach beyond MLPs, it is possible to utilize the graph analysis method introduced in the previous chapter for arbitrary DAGs and integrate a different counting function, encoding a new prior.

5.5 Discussion and Future Work

The described approaches focus on the computational graph of models or assigning scores to learnable parameters for model-level analysis and comparison. Moreover, the performance estimation relies on the number of input and output nodes as an ad-hoc solution to the fact that the properties of the training data are not taken into account.

The path normalized counting approach is an immediate next step for future work. Design, training, and data aspects in DL can also be integrated into this framework. Finally, applying the author's current level of understanding of performance estimation given in this chapter to the methods previously described in the neural architecture search and population training approaches is another important research direction.

5.6 Conclusions

In this chapter, the extreme scenarios for the context-based graph analysis were investigated, and models of various width, depth, and layer types (i.e layers that share pa-

[4] Note that, the quality of the ranking would not necessarily be high for arbitrary priors.

rameters, fully connected, convolutional, and batch normalization layers) were compared to each other in terms of the model level quantities introduced in Chapter 4. After discussing the practical limitations, a generalization of the context-based graph analysis approach was introduced to address them. This generalization was inspired by an analogy between contexts and microstates (Section 5.3). Assigning energy in the form of $\ln |C'_w|$ to each learnable parameter for model comparison revealed that from the perspective of statistical mechanics, the context-based approach describes models with temperature $T = -1$. The energy prior-based understanding of model comparison allowed a fairly easy proof of the convergence of entropy (previously defined in Section 4.4) for MLPs with constant width, as well as describing the behavior of how the performance estimation grows with depth and width *without relying on any graph analysis*. Finally, a generalized methodology based on statistical mechanics for encoding assumptions other than the reusability prior in E_i (i.e. arbitrary energy prior) for model comparison was exemplified.

Overall, in this chapter, a theoretical framework for energy-based priors and relevant proofs for a generalization of the model comparison approach from Chapter 4 were introduced. This framework may open up new avenues of research for the quantifiable analysis and comparison of DL models with different practical constraints and assumptions in mind.

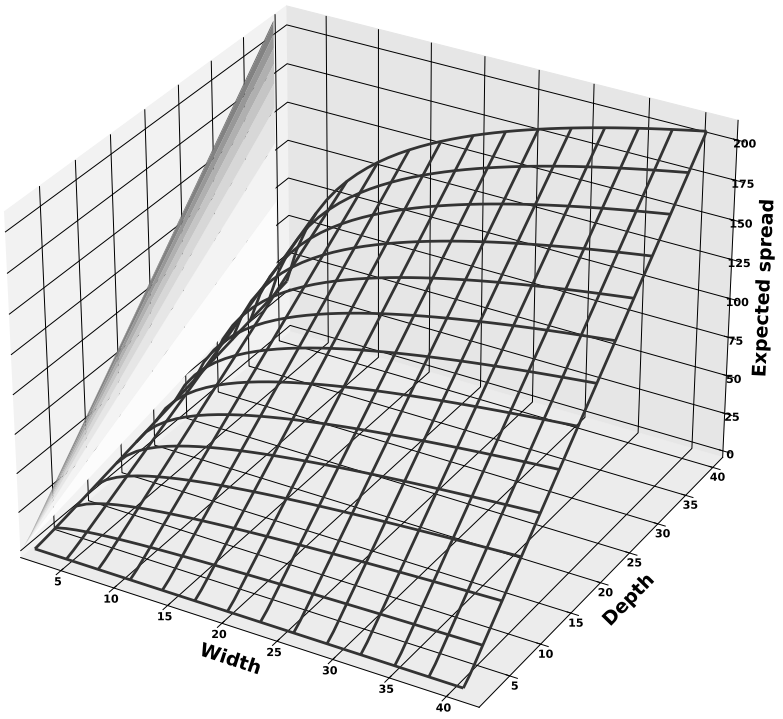


Figure 5.1: For MLP models of different widths and depths, the expected spread grows linearly with depth, and logarithmically with width. The wireframe in the figure corresponds to MLP models with layers of constant width selected from a grid of different depth and width configurations. The number of input nodes is kept constant ($N_I = 3$). The model with $(depth = 40, width = 40)$ has the highest expected spread.

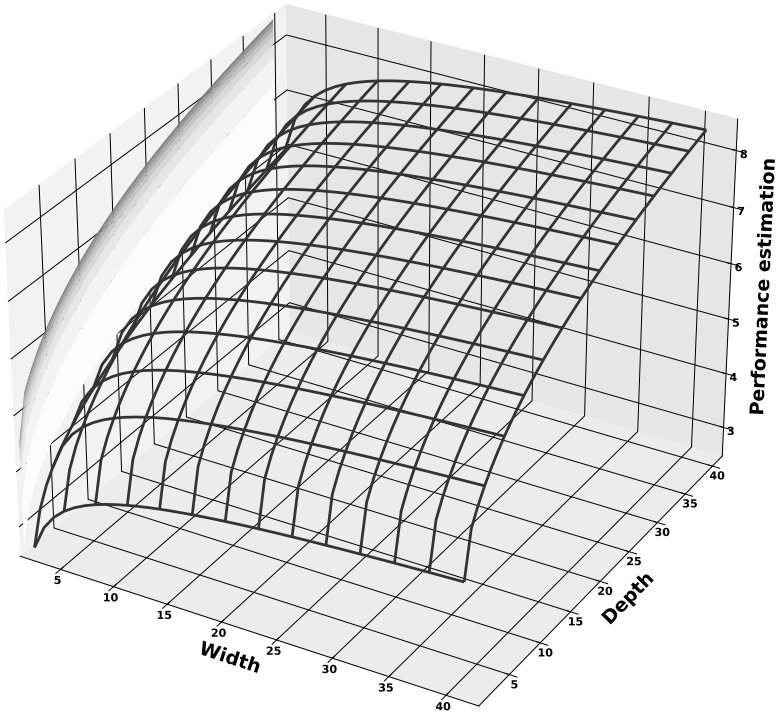


Figure 5.2: For MLP models of different widths and depths, the estimated performance estimation grows sublinearly for both depth and width, where it seems to benefit from depth more. The wireframe in the figure corresponds to MLP models with layers of constant width selected from a grid of different depth and width configurations. No batch normalization layer is used. The number of input nodes is kept constant ($N_I = 3$). Varying the number of input channels based on model width did not significantly change the figures. The deepest and the widest model (i.e. $(depth = 40, width = 40)$) has the highest estimated performance. Note that unlike Table5.1 the number of parameters is not held constant.

Table 5.1: MLP depth vs width trade-off with a parameter budget. No bias or batch norm is included so that the number of parameters exactly matches. All layers in a model have the same width, including the input layer (i.e. $N_I = width$). Deeper models are majorly favored (bold). P_G, and P'_G are quantities for performance estimation discussed in Section 4.4.5. Parameter efficiency estimation PE_G is is given in 5.1 by P_G/N_G where $N_G = 14400$. Note that parameter efficiency for higher performance models should eventually drop due to diminishing returns.

Params	Width	Depth	P_G	P'_G	PE_G
14400	120	1	10.683	6.895	7.42e-4
14400	60	4	10.272	10.120	7.13e-4
14400	40	9	10.316	10.759	7.16e-4
14400	30	16	10.448	11.122	7.26e-4
14400	24	25	10.588	11.375	7.35e-4
14400	20	36	10.718	11.568	7.44e-4
14400	15	64	10.938	11.853	7.60e-4
14400	12	100	11.113	12.058	7.72e-4
14400	10	144	11.253	12.215	7.81e-4
14400	8	225	11.418	12.393	7.93e-4
14400	6	400	11.610	12.595	8.06e-4
14400	5	576	11.715	12.705	8.14e-4
14400	4	900	**11.819**	12.812	**8.21e-4**
14400	**2**	**3600**	11.815	**12.813**	**8.21e-4**

Table 5.2: Cross-layer parameter sharing in MLP vs. no cross-layer parameter sharing. The number of input nodes is held constant ($N_I = 3$). All layers except the first layer in a model have the same width. Batch normalization layers are not shared. The deepest model has the highest performance estimations P_G and P'_G (bold), but the model that is predicted to have the top parameter efficiency PE_G is in the last row (bold). Furthermore, the model with the highest compute efficiency CE_G (bold) does not share parameters. It is neither the deepest nor the widest model. Note that parameter and compute efficiencies for high performance models naturally drop due to diminishing returns.

Params	Width	Depth	P_G	P'_G	PE_G	CE_G	Shared
28810	**2**	**3600**	**11.82**	**12.81**	4.10e-4	2.74e-4	True
21560	8	225	10.01	10.98	4.64e-4	4.64e-4	False
20090	10	144	9.52	10.48	4.74e-4	4.74e-4	False
19092	12	100	9.12	10.06	4.78e-4	**4.78e-4**	False
19209	3	1600	11.09	12.09	5.77e-4	3.30e-4	True
18060	15	64	8.62	9.53	4.77e-4	4.77e-4	False
16940	20	36	7.99	8.83	4.71e-4	4.71e-4	False
14428	4	900	10.41	11.40	**7.21e-4**	3.61e-4	True

Table 5.3: CNN depth vs width trade-off with a parameter budget, with fixed image size 8x8, kernel size 3, and no cross-layer parameter sharing. No bias or batch norm is included so that the number of parameters exactly matches. All layers in a model have the same number of channels, including the input layer (i.e. $N_I = channels$). The deepest model is favored the most (bold). P_G, and P'_G are quantities for performance estimation discussed in Section 4.4.5. Parameter efficiency estimation PE_G is is given in 5.1 by P_G/N_G where $N_G = 129600$. Note that parameter efficiency for higher performance models should eventually drop due to diminishing returns.

Params	Channels	Depth	P_G	P'_G	PE_G
129600	60	4	16.69	16.93	1.29e-04
129600	40	9	16.86	17.49	1.30e-04
129600	120	1	16.91	15.58	1.30e-04
129600	30	16	17.07	17.86	1.32e-04
129600	24	25	17.27	18.14	1.33e-04
129600	20	36	17.45	18.36	1.35e-04
129600	15	64	17.74	18.69	1.37e-04
129600	12	100	17.98	18.95	1.39e-04
129600	10	144	18.18	19.16	1.40e-04
129600	8	225	18.42	19.41	1.42e-04
129600	6	400	18.73	19.72	1.44e-04
129600	5	576	18.92	19.91	1.46e-04
129600	4	900	19.15	20.15	1.48e-04
129600	3	1600	19.44	20.44	1.50e-04
129600	**2**	**3600**	**19.83**	**20.83**	**1.53e-04**

Table 5.4: Cross-layer parameter sharing in CNN vs. no cross-layer parameter sharing. The number of input channels is held constant ($N_I = 3$), as well as the image size (8x8), and kernel size (3). All layers except for the first layer in a model have the same width. Shared models aggressively share all layers except for batch normalization. The top model with the performance estimations P_G and P'_G (bold) does not have cross-layer parameter sharing, unlike the model that is predicted to have the top parameter efficiency PE_G in the second row (bold). Further, the model with the highest estimated compute efficiency CE_G (bold) does not have cross-layer parameter sharing and it is not the deepest model. Note that parameter and compute efficiencies for high performance models naturally drop due to diminishing returns.

Params	Channels	Depth	P_G	P'_G	PE_G	CE_G	Shared
127464	24	25	14.29	15.15	1.12e-04	1.12e-04	False
4881	3	400	14.49	15.51	**2.97e-03**	3.89e-04	True
11880	5	576	14.57	15.61	1.23e-03	1.03e-04	True
129420	20	36	14.73	15.63	1.14e-04	1.14e-04	False
15590	5	64	15.04	16.00	9.65e-04	9.65e-04	False
14652	4	900	15.41	16.44	1.05e-03	1.07e-04	True
131820	15	64	15.43	16.38	1.17e-04	1.17e-04	False
133428	12	100	15.99	16.96	1.20e-04	1.20e-04	False
35190	5	144	16.19	17.17	4.60e-04	4.60e-04	False
9918	2	225	16.40	17.39	1.65e-03	**1.65e-03**	False
134730	10	144	16.45	17.42	1.22e-04	1.22e-04	False
19281	3	1600	16.49	17.49	8.55e-04	1.11e-04	True
35964	4	225	16.74	17.73	4.65e-04	4.65e-04	False
136440	8	225	17.01	17.99	1.25e-04	1.25e-04	False
139038	6	400	17.73	18.72	1.28e-04	1.28e-04	False
28890	2	3600	17.96	18.96	6.22e-04	1.13e-04	True
141030	5	576	**18.18**	**19.18**	1.29e-04	1.29e-04	False

CHAPTER 6

CONCLUSIONS

There has been a clear pattern in DL advancements that relied on scaling up model and data size. The theoretical and practical advancements were often accompanied by simplifications in DL (Chapter 2). As the compute resources turn into a bigger bottleneck, learning reusable representations and improving compute, sample, and parameter efficiency will likely stay important for DL research. In this work, various forms of reusability within and across DL models were investigated.

6.1 Revolver: A Search for Reusable Modules

In Chapter 3, a new library, Revolver, was introduced to analyze parameter sharing and searching for reusable modules while training a population. Some of the ideas in Revolver were bioinspired. A compact genetic toolbox may have evolved for the nervous system by reusing older genes for new purposes (co-option) as discussed in Section 3.2.2. Analogously, Revolver was implemented to be able to reuse DL model components or modules in various combinations within and across different models in a population. To test this idea in practice, a blueprinting approach was introduced for a precise description of modules and hyperparameters. Revolver treated blueprints as genotypes that described shared and unique modules (Section 3.3). In practice, this allowed simultaneously training a population of at least 80 models using a single GPU (Nvidia GeForce RTX 3090) for MNIST and CIFAR-10 benchmarks. Since some modules were shared across models and layers (cross-layer) this

minimized the required GPU memory and allowed new models to reuse the modules from previous generations. In less than 135 generations, models evolved with 99.6 % and 91.9% top-1 accuracies for MNIST and CIFAR-10 respectively. For CIFAR-10 one of the top models contained an ensemble of convolutions as a mutation (C.5). Moreover, during the experiments, the meta information on modules was tracked by Revolver, enabling a ranking of shared modules (C.3) for each population. The main limitation of Revolver was the compute resources for larger benchmarks. Therefore parallelization or larger GPUs would be necessary for larger experiments. Revolver may be extended or directly used by researchers for analysis, hyperparameter, or neural architecture search. The GitHub repo for this work is available at: `https://github.com/gozepolat/revolver`.

6.2 The Reusability Prior: Comparing DL Models without Training

While Revolver demonstrated the viability of a module-level (e.g. convolutional layer) reusability across models within a population, the GPU resources continued to be a bottleneck for neural architecture search for large benchmarks. To address this, a methodology based on information theory and graph analysis was proposed in Chapter 4, and in a publication [15]. Compared to Revolver which focused on a practical setting of reusing entire modules or layers while training a population of models, this work focused on establishing a theoretical foundation for a generalized notion of reusability. The *reusability prior* that ties training, design, and data aspects together was proposed as follows:

Model components are forced to function in diverse contexts not only due to the training, data, augmentation, and regularization choices but also due to the model design itself. These aspects explicitly or implicitly impact the expected number of contexts for model components. Until model capacity is reached, maximizing this number improves parameter efficiency for models of similar size and capacity. By relying on the repetition of reusable components, a model can learn to describe an approximation

of the desired function more efficiently with fewer parameters.

A precise definition of context was given in Section 4.1.2. Based on this definition, a simple counting approach, and consequently, various model level quantities were introduced for analysis and comparison of the computational graphs of DL models (Section 4.4) including two different quantities to estimate performance or ranking models without relying on any training. In the model analysis-based experiments, the rank of EfficientNetv2 and ResNet-50 models were correctly predicted in terms of top-1 accuracy for the ImageNet-1K dataset. In contrast, the naive approach of relying on the number of learnable parameters failed to correctly rank them (i.e. Tables 4.3 and 4.4). The main limitation of this work was that the counting approach did not allow any computational or other practical constraints to be included when comparing models. In practice, the reusability prior and the proposed framework may lead to new approaches for neural architecture search, or help researchers improve their model design in the right direction before any training is done. The code to reproduce the analysis results is available at [16] and the GitHub repo at: `https://github.com/gozepolat/priors/tree/main/reusability`.

6.3 Energy Priors: A Statistical Mechanics Framework For Model Comparison

The counting-based approach introduced in Chapter 4 majorly ignored practical constraints including compute efficiency and floating point precision. A generalization was introduced to address these practical limitations. The generalization was inspired by an analogy between contexts and microstates (Section 5.3). Assigning energy in the form of $\ln |C'_w|$ to each learnable parameter for model comparison revealed that from the perspective of *statistical mechanics*, the context-based approach described models with temperature $T = -1$. Furthermore, the energy prior-based understanding of model comparison allowed a fairly easy proof of the convergence of the entropy (previously defined in Section 4.4) for MLPs with constant width, as well as

describing the behavior of how the performance estimation grows with depth and width *without relying on any graph analysis*.

Investigating the extreme scenarios for the original counting-based approach for models of various width, depth, and layer types (i.e layers that share parameters, fully connected, convolutional, and batch normalization layers) confirmed that, for models with the same number of parameters, the performance estimation approaches had a bias towards deeper and narrower models. To address this, a generalized methodology based on statistical mechanics for encoding assumptions other than the reusability prior in E_i (i.e. arbitrary energy prior) for model comparison was exemplified.

Overall, inspired by statistical mechanics, a theoretical framework and relevant proofs for a generalization of the model comparison approach from Chapter 4 were introduced. This framework may open up avenues of research for the quantifiable analysis and comparison of DL models with different practical constraints and assumptions in mind. For neural architecture search, this methodology may allow encoding practical constraints or new priors in a fitness function.

6.4 The Big Picture and Major Outcomes

In conclusion, the major outcomes were as follows:

- To reuse is to simplify. Reusing modules led to relatively efficient training of a population of models on a single GPU for analysis and neural architecture search in Chapter 3.
- All DL models reuse parameters and this affects their parameter efficiency. A counting approach that can quantify the number of *contexts* for each learnable parameter led to correct performance predictions regarding the ranking of several analyzed models in terms of top-1 accuracy in Chapter 4.
- Constraints and assumptions may be encoded as energy E_i at the parameter

level. A statistical mechanics framework for model comparison in Chapter 5 revealed that the previous context-based counting approach described models with absolute temperature $T = -1$. This framework allowed fairly easy proofs regarding the behavior of performance for MLPs with constant width and the convergence of the entropy. As a generalization, it also allowed going beyond the reusability prior by encoding new energy priors (and different absolute temperatures) for model comparison.

Overall, DL models will likely keep growing in size if not in sophistication. This makes the analysis of models in terms of the reusability of components an important topic, as not all performance can be explained directly with the number of parameters and the training examples. Some of the deeper reasons that were explored in this work led to the direction of model analysis and comparison. This direction is currently a relatively green area. The proposed practical and theoretical approaches may open up avenues of research on model analysis and comparison as well as neural architecture search.

REFERENCES

[1] M. Tan and Q. Le, "Efficientnetv2: Smaller models and faster training," in *International Conference on Machine Learning*, pp. 10096–10106, PMLR, 2021.

[2] OpenAI, "Gpt-4 technical report," 2023.

[3] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A lite BERT for self-supervised learning of language representations," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, 2020.

[4] B. A. Plummer, N. Dryden, J. Frost, T. Hoefler, and K. Saenko, "Neural parameter allocation search," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, OpenReview.net, 2022.

[5] A. Boulch, "Reducing parameter number in residual networks by sharing weights," *Pattern Recognit. Lett.*, vol. 103, pp. 53–59, 2018.

[6] P. Savarese and M. Maire, "Learning implicitly recurrent CNNs through parameter sharing," in *International Conference on Learning Representations*, 2019.

[7] M. Li, Y. Wang, X. Xie, and Z. Lin, "Optimization inspired multi-branch equilibrium models," in *International Conference on Learning Representations*, 2021.

[8] S. Bai, J. Z. Kolter, and V. Koltun, "Deep equilibrium models," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[9] S. Bai, V. Koltun, and J. Z. Kolter, "Multiscale deep equilibrium models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5238–5250, 2020.

[10] S. Bai, V. Koltun, and J. Z. Kolter, "Stabilizing equilibrium models by jacobian regularization," in *ICML*, 2021.

[11] S. W. Fung, H. Heaton, Q. Li, D. Mckenzie, S. J. Osher, and W. Yin, "Jfb: Jacobian-free backpropagation for implicit networks," 2021.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, vol. 2016-Decem, pp. 770–778, 2016.

[13] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 2261–2269, IEEE Computer Society, 2017.

[14] A. Krizhevsky, V. Nair, and G. Hinton, "The CIFAR-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

[15] A. POLAT and F. Alpaslan, "The reusability prior: comparing deep learning models without training," *Machine Learning: Science and Technology*, 2023.

[16] A. G. Polat, "gozepolat/priors: The reusability prior," Apr. 2023.

[17] M. Minsky, "Steps toward artificial intelligence," *Proceedings of the IRE*, vol. 49, no. 1, pp. 8–30, 1961.

[18] Y. Bengio, A. C. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, 2013.

[19] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland,

K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, and L. Sifre, "Training Compute-Optimal Large Language Models," *ArXiv*, vol. abs/2203.1, 2022.

[20] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[21] J. T. Håstad, *Computational limitations for small-depth circuits*. MIT press, 1987.

[22] Y. Bengio and O. Delalleau, "On the expressive power of deep architectures," in *Algorithmic Learning Theory: 22nd International Conference, ALT 2011, Espoo, Finland, October 5-7, 2011. Proceedings 22*, pp. 18–36, Springer, 2011.

[23] H. J. Kelley, "Gradient theory of optimal flight paths," *Ars Journal*, vol. 30, no. 10, pp. 947–954, 1960.

[24] A. E. Bryson Jr, W. F. Denham, and S. E. Dreyfus, "Optimal programming problems with inequality constraints," *AIAA journal*, vol. 1, no. 11, pp. 2544–2550, 1963.

[25] S. Linnainmaa, *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. PhD thesis, Master's Thesis (in Finnish), Univ. Helsinki, 1970.

[26] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," 1991.

[27] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[28] M. Ding, B. Xiao, N. C. F. Codella, P. Luo, J. Wang, and L. Yuan, "DaViT: Dual Attention Vision Transformers," *ArXiv*, vol. abs/2204.0, 2022.

[29] J. Yu, Z. Wang, V. Vasudevan, L. Yeung, M. Seyedhosseini, and Y. Wu, "CoCa: Contrastive Captioners are Image-Text Foundation Models," *ArXiv*, vol. abs/2205.0, 2022.

[30] M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith, and L. Schmidt, "Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time," *ArXiv*, vol. abs/2203.0, 2022.

[31] Z. Dai, H. Liu, Q. V. Le, and M. Tan, "CoAtNet: Marrying Convolution and Attention for All Data Sizes," *ArXiv*, vol. abs/2106.0, 2021.

[32] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[33] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[34] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[35] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and cooperation in neural nets*, pp. 267–285, Springer, 1982.

[36] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities.," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[37] E. Oja, "Simplified neuron model as a principal component analyzer," *Journal of mathematical biology*, vol. 15, pp. 267–273, 1982.

[38] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[39] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[40] P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," tech. rep., Colorado Univ at Boulder Dept of Computer Science, 1986.

[41] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[42] D. S. Broomhead and D. Lowe, "Radial basis functions, multi-variable functional interpolation and adaptive networks," tech. rep., Royal Signals and Radar Establishment Malvern (United Kingdom), 1988.

[43] P. Baldi and K. Hornik, "Neural networks and principal component analysis: Learning from examples without local minima," *Neural networks*, vol. 2, no. 1, pp. 53–58, 1989.

[44] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.

[45] R. M. Neal, "Connectionist learning of belief networks," *Artificial intelligence*, vol. 56, no. 1, pp. 71–113, 1992.

[46] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, pp. 273–297, 1995.

[47] B. A. Olshausen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.

[48] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[49] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[50] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, pp. 2278–2324, 1998.

[51] S. Behnke, "Discovering hierarchical speech features using convolutional non-negative matrix factorization," in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, vol. 4, pp. 2758–2763, IEEE, 2003.

[52] K. Fukushima, "Neural network model for a mechanism of pattern recognition unaffected by shift in position-neocognitron," *IEICE Technical Report, A*, vol. 62, no. 10, pp. 658–665, 1979.

[53] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, p. 106, 1962.

[54] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[55] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 61, no. 3, pp. 611–622, 1999.

[56] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[57] M. Ranzato, C. Poultney, S. Chopra, and Y. Cun, "Efficient learning of sparse representations with an energy-based model," *Advances in neural information processing systems*, vol. 19, 2006.

[58] B. A. Olshausen, "13 sparse codes and spikes," *Probabilistic models of the brain*, p. 257, 2002.

[59] M. Ranzato, Y.-L. Boureau, Y. Cun, *et al.*, "Sparse feature learning for deep belief networks," *Advances in neural information processing systems*, vol. 20, 2007.

[60] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, 2006.

[61] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, JMLR Workshop and Conference Proceedings, 2010.

[62] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[63] 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*, NIPS'12, (USA), pp. 1097–1105, Curran Associates Inc., 2012.

[64] G. E. Hinton, "To recognize shapes, first learn to generate images," *Progress in brain research*, vol. 165, pp. 535–547, 2007.

[65] A. Paul and S. Venkatasubramanian, "Why does deep learning work?-a perspective from group theory," *arXiv preprint arXiv:1412.6621*, 2014.

[66] Y. Bengio, Y. LeCun, *et al.*, "Scaling learning algorithms towards ai," *Large-scale kernel machines*, vol. 34, no. 5, pp. 1–41, 2007.

[67] N. Le Roux and Y. Bengio, "Deep belief networks are compact universal approximators," *Neural computation*, vol. 22, no. 8, pp. 2192–2207, 2010.

[68] N. Le Roux and Y. Bengio, "Representational power of restricted boltzmann machines and deep belief networks," *Neural computation*, vol. 20, no. 6, pp. 1631–1649, 2008.

[69] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[70] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 427–436, 2015.

[71] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[72] M. H. Freedman and F. Quinn, *Topology of 4-Manifolds (PMS-39), Volume 39*, vol. 46. Princeton University Press, 2014.

[73] S. Reed, K. Sohn, Y. Zhang, and H. Lee, "Learning to disentangle factors of variation with manifold interaction," in *International conference on machine learning*, pp. 1431–1439, PMLR, 2014.

[74] R. Salakhutdinov and G. Hinton, "Learning a nonlinear embedding by preserving class neighbourhood structure," in *Artificial intelligence and statistics*, pp. 412–419, PMLR, 2007.

[75] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *arXiv preprint arXiv:1505.00387*, 2015.

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

[77] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern*

Recognition, 2009. CVPR 2009. IEEE Conference on, pp. 248–255, IEEE, 2009.

[78] M. Everingham, L. Van∼Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, jun 2010.

[79] T.-Y. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," *CoRR*, vol. abs/1405.0, 2014.

[80] S. Targ, D. Almeida, and K. Lyman, "Resnet in Resnet: generalizing residual architectures," *arXiv preprint arXiv:1603.08029*, 2016.

[81] Q. Zhao, T. Sheng, Y. Wang, Z. Tang, Y. Chen, L. Cai, and H. Ling, "M2det: A single-shot object detector based on multi-level feature pyramid network," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 9259–9266, 2019.

[82] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated Residual Transformations for Deep Neural Networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5987–5995, 2017.

[83] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[84] T. Xiao, Y. Li, J. Zhu, Z. Yu, and T. Liu, "Sharing attention weights for fast transformer," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 5292–5298, International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[85] M. Reid, E. Marrese-Taylor, and Y. Matsuo, "Subformer: Exploring weight sharing for parameter efficiency in generative transformers," in *Findings of the*

Association for Computational Linguistics: EMNLP 2021, (Punta Cana, Dominican Republic), pp. 4081–4090, Association for Computational Linguistics, nov 2021.

[86] J. H. Tan, Y. H. Tan, C. S. Chan, and J. H. Chuah, "ACORT: A compact object relation transformer for parameter efficient image captioning," *Neurocomputing*, vol. 482, pp. 60–72, 2022.

[87] H. Touvron, P. Bojanowski, M. Caron, M. Cord, A. El-Nouby, E. Grave, G. Izacard, A. Joulin, G. Synnaeve, J. Verbeek, and H. J'egou, "ResMLP: Feedforward networks for image classification with data-efficient training," *ArXiv*, vol. abs/2105.0, 2021.

[88] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, D. Keysers, J. Uszkoreit, M. Lucic, and A. Dosovitskiy, "MLP-Mixer: An all-MLP Architecture for Vision," in *NeurIPS*, 2021.

[89] C. Tang, Y. Zhao, G. Wang, C. Luo, W. Xie, and W. Zeng, "Sparse MLP for image recognition: Is self-attention really necessary?," in *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pp. 2344–2351, AAAI Press, 2022.

[90] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *European Conference on Computer Vision*, pp. 646–661, Springer, 2016.

[91] K. Zhang, M. Sun, T. X. Han, X. Yuan, L. Guo, T. Liu, X. Han, X. Yuan, L. Guo, and T. Liu, "Residual networks of residual networks: multilevel residual networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 6, pp. 1303–1314, 2017.

[92] G. Larsson, M. Maire, and G. Shakhnarovich, "FractalNet: Ultra-Deep Neural Networks without Residuals," *arXiv preprint arXiv:1605.07648*, 2016.

[93] X. Gastaldi, "Shake-Shake regularization," *arXiv preprint arXiv:1705.07485*, 2017.

[94] T. DeVries and G. W. Taylor, "Improved Regularization of Convolutional Neural Networks with Cutout," *arXiv preprint arXiv:1708.04552*, 2017.

[95] Q. Liao and T. A. Poggio, "Bridging the Gaps Between Residual Learning, Recurrent Neural Networks and Visual Cortex," *CoRR*, vol. abs/1604.0, 2016.

[96] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng, "Dual Path Networks," *arXiv preprint arXiv:1707.01629*, 2017.

[97] R. Soltani and H. Jiang, "Higher order recurrent neural networks," *arXiv preprint arXiv:1605.00064*, 2016.

[98] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in neural information processing systems*, vol. 31, 2018.

[99] E. Dupont, A. Doucet, and Y. W. Teh, "Augmented neural odes," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[100] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[101] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[102] H. Bilen and A. Vedaldi, "Universal representations: The missing link between faces, text, planktons, and cat breeds," *ArXiv*, vol. abs/1701.07275, 2017.

[103] L. Y. Wu, A. Fisch, S. Chopra, K. Adams, A. Bordes, and J. Weston, "Starspace: Embed all the things!," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018* (S. A. McIlraith and K. Q. Weinberger, eds.), pp. 5569–5577, AAAI Press, 2018.

[104] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra, "Pathnet: Evolution channels gradient descent in super neural networks," *arXiv preprint arXiv:1701.08734*, 2017.

[105] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, pp. 1126–1135, PMLR, 2017.

[106] A. Anderson, K. Shaffer, A. Yankov, C. D. Corley, and N. O. Hodas, "Beyond Fine Tuning: A Modular Approach to Learning on Small Data," *arXiv preprint arXiv:1611.01714*, 2016.

[107] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[108] M. Ranzato, F. J. Huang, Y.-L. Boureau, and Y. LeCun, "Unsupervised learning of invariant feature hierarchies with applications to object recognition," in *2007 IEEE conference on computer vision and pattern recognition*, pp. 1–8, IEEE, 2007.

[109] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *IEEE Data Eng. Bull.*, vol. 40, no. 3, pp. 52–74, 2017.

[110] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[111] E. H. Thiede, T. Hy, and R. Kondor, "The general theory of permutation equivarant neural networks and higher order graph variational encoders," *CoRR*, vol. abs/2004.03990, 2020.

[112] A. Zhou, T. Knowles, and C. Finn, "Meta-learning symmetries by reparameterization," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, OpenReview.net, 2021.

[113] N. Dehmamy, R. Walters, Y. Liu, D. Wang, and R. Yu, "Automatic Symmetry Discovery with Lie Algebra Convolutional Network," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[114] P. de Haan, T. S. Cohen, and M. Welling, "Natural graph networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 3636–3646, 2020.

[115] T. Cohen, M. Weiler, B. Kicanaoglu, and M. Welling, "Gauge equivariant convolutional networks and the icosahedral CNN," in *International conference on Machine learning*, pp. 1321–1330, PMLR, 2019.

[116] K. S. Tai, P. Bailis, and G. Valiant, "Equivariant transformer networks," in *International Conference on Machine Learning*, pp. 6086–6095, PMLR, 2019.

[117] R. A. Yeh, Y.-T. Hu, M. Hasegawa-Johnson, and A. Schwing, "Equivariance Discovery by Learned Parameter-Sharing," in *International Conference on Artificial Intelligence and Statistics*, pp. 1527–1545, PMLR, 2022.

[118] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[119] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," *Advances in Neural Information Processing Systems*, vol. 2015-Janua, pp. 2017–2025, 2015.

[120] X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer, "Scaling vision transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12104–12113, 2022.

[121] B. van Rooyen and R. C. Williamson, "A theory of feature learning," *arXiv preprint arXiv:1504.00083*, 2015.

[122] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 689–696, 2011.

[123] J. Weston, S. Bengio, and N. Usunier, "Large scale image annotation: learning to rank with joint word-image embeddings," *Machine learning*, vol. 81, pp. 21–35, 2010.

[124] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States* (C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, eds.), pp. 3111–3119, 2013.

[125] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by v1?," *Vision research*, vol. 37, no. 23, pp. 3311–3325, 1997.

[126] F. Anselmi and T. A. Poggio, "Representation learning in sensory cortex: A theory," *IEEE Access*, vol. 10, pp. 102475–102491, 2022.

[127] F. Anselmi, L. Rosasco, and T. Poggio, "On invariance and selectivity in representation learning," *Information and Inference: A Journal of the IMA*, vol. 5, no. 2, pp. 134–158, 2016.

[128] F. Anselmi and T. Poggio, "Representation learning in sensory cortex: a theory," tech. rep., Center for Brains, Minds and Machines (CBMM), 2014.

[129] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east," *IDC iView: IDC Analyze the future*, vol. 2007, no. 2012, pp. 1–16, 2012.

[130] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223, JMLR Workshop and Conference Proceedings, 2011.

[131] M. Banko and E. Brill, "Scaling to very very large corpora for natural language disambiguation," in *Proceedings of the 39th annual meeting of the Association for Computational Linguistics*, pp. 26–33, 2001.

[132] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016* (R. C. Wilson, E. R. Hancock, and W. A. P. Smith, eds.), BMVA Press, 2016.

[133] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[134] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, pmlr, 2015.

[135] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Advances in Neural Information Processing Systems*, pp. 901–909, 2016.

[136] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[137] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recogni-*

tion, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017, pp. 1800–1807, IEEE Computer Society, 2017.

[138] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[139] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[140] S. J. Nowlan and G. E. Hinton, "Simplifying Neural Networks by Soft Weight-Sharing," *Neural Computation*, vol. 4, pp. 473–493, 1992.

[141] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541, 2006.

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

[143] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2016.

[144] M. Sharma, S. Tripathi, A. Dubey, S. Guruju, N. Goalla, and Others, "Smaller Models, Better Generalization," *arXiv preprint arXiv:1908.11250*, 2019.

[145] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, "Big transfer (bit): General visual representation learning," in *ECCV*, 2020.

[146] K. He, X. Chen, S. Xie, Y. Li, P. Doll'ar, and R. B. Girshick, "Masked autoencoders are scalable vision learners," *ArXiv*, vol. abs/2111.06377, 2021.

[147] H. Bao, L. Dong, and F. Wei, "Beit: Bert pre-training of image transformers," *ArXiv*, vol. abs/2106.08254, 2021.

[148] C. Jia, Y. Yang, Y. Xia, Y.-T. Chen, Z. Parekh, H. Pham, Q. V. Le, Y.-H. Sung, Z. Li, and T. Duerig, "Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision," in *ICML*, 2021.

[149] A. Brock, S. De, S. L. Smith, and K. Simonyan, "High-Performance Large-Scale Image Recognition Without Normalization," *ArXiv*, vol. abs/2102.0, 2021.

[150] L. Yuan, D. Chen, Y.-L. Chen, N. C. F. Codella, X. Dai, J. Gao, H. Hu, X. Huang, B. Li, C. Li, C. Liu, M. Liu, Z. Liu, Y. Lu, Y. Shi, L. Wang, J. Wang, B. Xiao, Z. Xiao, J. Yang, M. Zeng, L. Zhou, and P. Zhang, "Florence: A New Foundation Model for Computer Vision," *ArXiv*, vol. abs/2111.1, 2021.

[151] C. Riquelme, J. Puigcerver, B. Mustafa, M. Neumann, R. Jenatton, A. S. Pinto, D. Keysers, and N. Houlsby, "Scaling Vision with Sparse Mixture of Experts," in *NeurIPS*, 2021.

[152] X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer, "Scaling Vision Transformers," *ArXiv*, vol. abs/2106.0, 2021.

[153] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. A. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. A. Zaharia, "Efficient large-scale language model training on gpu clusters using megatron-lm," *SC21: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14, 2021.

[154] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter,

C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 1877–1901, Curran Associates, Inc., 2020.

[155] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *Journal of Machine Learning Research*, vol. 23, no. 120, pp. 1–39, 2022.

[156] N. Du, Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, Y. Xu, M. Krikun, Y. Zhou, A. W. Yu, O. Firat, B. Zoph, L. Fedus, M. P. Bosma, Z. Zhou, T. Wang, E. Wang, K. Webster, M. Pellat, K. Robinson, K. Meier-Hellstern, T. Duke, L. Dixon, K. Zhang, Q. Le, Y. Wu, Z. Chen, and C. Cui, "GLaM: Efficient scaling of language models with mixture-of-experts," in *Proceedings of the 39th International Conference on Machine Learning* (K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, eds.), vol. 162 of *Proceedings of Machine Learning Research*, pp. 5547–5569, PMLR, 17–23 Jul 2022.

[157] X. Zou, D. Yin, Q. Zhong, H. Yang, Z. Yang, and J. Tang, "Controllable generation from pre-trained language models via inverse prompting," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '21, (New York, NY, USA), p. 2450–2460, Association for Computing Machinery, 2021.

[158] Y. Levine, N. Wies, O. Sharir, H. Bata, and A. Shashua, "Limits to depth efficiencies of self-attention," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), 2020.

[159] D. Erhan, A. Courville, Y. Bengio, and P. Vincent, "Why does unsupervised pre-training help deep learning?," in *Proceedings of the thirteenth international*

conference on artificial intelligence and statistics, pp. 201–208, JMLR Workshop and Conference Proceedings, 2010.

[160] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient Neural Architecture Search via Parameters Sharing," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 4095–4104, PMLR, 2018.

[161] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," in *ICLR (Poster)*, OpenReview.net, 2019.

[162] D. Tcheng, B. Lambert, S. C. Y. Lu, and L. Rendell, "Building robust learning systems by combining induction and optimization," *Urbana*, vol. 100, p. 61801, 1989.

[163] S. Thrun and L. Pratt, "Learning to learn: Introduction and overview," in *Learning to learn*, pp. 3–17, Springer, 1998.

[164] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artificial Intelligence Review*, vol. 18, no. 2, pp. 77–95, 2002.

[165] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 4095–4104, PMLR, 10–15 Jul 2018.

[166] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google Vizier: A Service for Black-Box Optimization," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1487–1495, ACM, 2017.

[167] Y. Chen, M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, and

N. de Freitas, "Learning to learn for global optimization of black box functions," *CoRR*, vol. abs/1611.03824, 2016.

[168] R. Negrinho and G. J. Gordon, "Deeparchitect: Automatically designing and training deep architectures," *CoRR*, vol. abs/1704.08792, 2017.

[169] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, "Neural optimizer search with reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, pp. 459–468, PMLR, 2017.

[170] V. K. Garg and A. T. Kalai, "Meta-unsupervised-learning: A supervised approach to unsupervised learning," *CoRR*, vol. abs/1612.09030, 2016.

[171] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems*, pp. 3981–3989, 2016.

[172] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving Deep Neural Networks," *arXiv preprint arXiv:1703.00548*, 2017.

[173] K. Li and J. Malik, "Learning to optimize," *arXiv preprint arXiv:1606.01885*, 2016.

[174] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *arXiv preprint arXiv:1611.02167*, 2016.

[175] T. Munkhdalai and H. Yu, "Meta Networks," *arXiv preprint arXiv:1703.00837*, 2017.

[176] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. Le, and A. Kurakin, "Large-scale evolution of image classifiers," *arXiv preprint arXiv:1703.01041*, 2017.

[177] L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit, "One Model To Learn Them All," *arXiv preprint arXiv:1706.05137*, 2017.

[178] M. Long, Y. Cao, J. Wang, and M. Jordan, "Learning transferable features with deep adaptation networks," in *International Conference on Machine Learning*, pp. 97–105, 2015.

[179] S. Saxena and J. Verbeek, "Convolutional neural fabrics," in *Advances in Neural Information Processing Systems*, pp. 4053–4061, 2016.

[180] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, pp. 4780–4789, 2019.

[181] E. Galván and P. Mooney, "Neuroevolution in deep neural networks: Current trends and future challenges," *IEEE Transactions on Artificial Intelligence*, vol. 2, no. 6, pp. 476–493, 2021.

[182] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *UAI*, vol. 115 of *Proceedings of Machine Learning Research*, pp. 367–377, AUAI Press, 2019.

[183] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, J. Ben-Tzur, M. Hardt, B. Recht, and A. Talwalkar, "A system for massively parallel hyperparameter tuning," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 230–246, 2020.

[184] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2820–2828, 2019.

[185] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional

neural networks," in *International conference on machine learning*, pp. 6105–6114, PMLR, 2019.

[186] D. Baymurzina, E. Golikov, and M. Burtsev, "A review of neural architecture search," *Neurocomputing*, vol. 474, pp. 82–93, 2022.

[187] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, *et al.*, "Population based training of neural networks," *arXiv preprint arXiv:1711.09846*, 2017.

[188] A. G. Polat, *Modeling neurons that can self organize into building blocks and hierarchies : an exploration based on visual systems*. Master thesis, Middle East Technical University, 2012.

[189] R. Dawkins, "selfish gene," 1976.

[190] G. Miller, "On the origin of the nervous system," 2009.

[191] D. Arendt, H. Hausen, and G. Purschke, "The 'division of labour'model of eye evolution," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 364, no. 1531, pp. 2809–2817, 2009.

[192] C. Mason, "The development of developmental neuroscience," *Journal of Neuroscience*, vol. 29, no. 41, pp. 12735–12747, 2009.

[193] N. H. Wilson and E. T. Stoeckli, "Sonic Hedgehog regulates Wnt activity during neural circuit formation," in *Vitamins and Hormones*, vol. 88, pp. 173–209, Elsevier, 2012.

[194] F. Hirth, L. Kammermeier, E. Frei, U. Walldorf, M. Noll, and H. Reichert, "An urbilaterian origin of the tripartite brain: developmental genetic insights from Drosophila," *Development*, vol. 130, no. 11, pp. 2365–2373, 2003.

[195] A. Wirmer, S. Bradler, and R. Heinrich, "Homology of insect corpora allata and vertebrate adenohypophysis?," *Arthropod structure & development*, vol. 41, no. 5, pp. 409–417, 2012.

[196] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[197] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[198] A. Pal, A. Edelman, and C. Rackauckas, "Mixing implicit and explicit deep learning with skip deqs and infinite time neural odes (continuous deqs)," *ArXiv*, vol. abs/2201.12240, 2022.

[199] B. L. Miller, D. E. Goldberg, *et al.*, "Genetic algorithms, tournament selection, and the effects of noise," *Complex systems*, vol. 9, no. 3, pp. 193–212, 1995.

[200] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.

[201] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[202] F. Lundh, "An introduction to tkinter," *URL: www. pythonware. com/library/tkinter/introduction/index. htm*, 1999.

[203] J. Hestness, S. Narang, N. Ardalani, G. F. Diamos, H. Jun, H. Kianinejad,

M. M. A. Patwary, Y. Yang, and Y. Zhou, "Deep Learning Scaling is Predictable, Empirically," *ArXiv*, vol. abs/1712.0, 2017.

[204] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[205] J. Schmidhuber, "Habilitation thesis: System modeling and optimization," *Page 150 ff demonstrates credit assignment across the equivalent of 1,200 layers in an unfolded RNN*, 1993.

[206] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," 1999.

[207] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," in *Proceedings of the Second International Conference on Learning Representations (ICLR 2014)*, 2014.

[208] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," *ArXiv*, vol. abs/1412.3, 2014.

[209] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are rnns: Fast autoregressive transformers with linear attention," in *Proceedings of the 37th International Conference on Machine Learning*, ICML'20, JMLR.org, 2020.

[210] Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized Autoregressive Pretraining for Language Understanding," in *NeurIPS*, 2019.

[211] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio," in *SSW*, 2016.

[212] Z. Wang, X. Cheng, G. Sapiro, and Q. Qiu, "ACDC: Weight Sharing in Atom-Coefficient Decomposed Convolution," *ArXiv*, vol. abs/2009.0, 2020.

[213] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *Computer Vision (ICCV), 2015 IEEE International Conference on*, pp. 4489–4497, IEEE, 2015.

[214] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," *CoRR*, vol. abs/1605.07678, 2016.

[215] S. Singh, D. Hoiem, and D. Forsyth, "Swapout: Learning an ensemble of deep architectures," in *Advances in neural information processing systems*, pp. 28–36, 2016.

[216] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu, "Cnn-rnn: A unified framework for multi-label image classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2285–2294, 2016.

[217] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient Neural Architecture Search via Parameter Sharing," in *ICML*, 2018.

[218] G. Lan, L. de Vries, and S. Wang, "Evolving efficient deep neural networks for real-time object recognition," in *IEEE Symposium Series on Computational Intelligence, SSCI 2019, Xiamen, China, December 6-9, 2019*, pp. 2571–2578, IEEE, 2019.

[219] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. S. Feris, "BlockDrop: Dynamic Inference Paths in Residual Networks," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 8817–8826, Computer Vision Foundation / IEEE Computer Society, 2018.

[220] C. Louizos, K. Ullrich, and M. Welling, "Bayesian compression for deep learning," in *Advances in Neural Information Processing Systems*, pp. 3290–3300, 2017.

[221] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.

[222] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019.

[223] E. Malach, G. Yehudai, S. Shalev-Schwartz, and O. Shamir, "Proving the lottery ticket hypothesis: Pruning is all you need," in *International Conference on Machine Learning*, pp. 6682–6691, PMLR, 2020.

[224] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Advances in neural information processing systems*, pp. 3320–3328, 2014.

[225] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019.

[226] J. Ma, Z. Zhao, J. Chen, A. Li, L. Hong, and E. H. Chi, "SNR: sub-network routing for flexible parameter sharing in multi-task learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 216–223, 2019.

[227] C. Murdock, Z. Li, H. Zhou, and T. Duerig, "Blockout: Dynamic Model Selection for Hierarchical Deep Networks," in *CVPR 2016*, 2016.

[228] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 4510–4520, Computer Vision Foundation / IEEE Computer Society, 2018.

[229] A. Howard, R. Pang, H. Adam, Q. V. Le, M. Sandler, B. Chen, W. Wang, L. Chen, M. Tan, G. Chu, V. Vasudevan, and Y. Zhu, "Searching for mobilenetv3," in *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pp. 1314–1324, IEEE, 2019.

[230] K. Wickstrøm, S. Løkse, M. Kampffmeyer, S. Yu, J. C. Príncipe, and R. Jenssen, "Information plane analysis of deep neural networks via matrix-based renyi's entropy and tensor kernels," *CoRR*, vol. abs/1909.11396, 2019.

[231] K. Bu, Y. Zhang, and Q. Luo, "Depth-width trade-offs for neural networks via topological entropy," *ArXiv*, vol. abs/2010.07587, 2020.

[232] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[233] H. Bagherinezhad, M. Rastegari, and A. Farhadi, "LCNN: Lookup-Based Convolutional Neural Network," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 860–869, 2017.

[234] Q. Qiu, X. Cheng, A. R. Calderbank, and G. Sapiro, "Dcfnet: Deep neural network with decomposed convolutional filters," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018* (J. G. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 4195–4204, PMLR, 2018.

[235] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[236] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019,*

Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers) (J. Burstein, C. Doran, and T. Solorio, eds.), pp. 4171–4186, Association for Computational Linguistics, 2019.

[237] S. Yang, L. Hou, X. Song, Q. Liu, and D. Zhou, "Speeding up Deep Model Training by Sharing Weights and Then Unsharing," *ArXiv*, vol. abs/2110.0, 2021.

[238] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *arXiv preprint arXiv:1710.09412*, 2017.

[239] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, "Cutmix: Regularization strategy to train strong classifiers with localizable features," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6023–6032, 2019.

[240] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA* (I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, eds.), pp. 5998–6008, 2017.

[241] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[242] Google, "Efficientnetv2 official implementation," 2022.

[243] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[244] E. D. Cubuk, B. Zoph, D. Mané, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning augmentation strategies from data," in *IEEE Conference on Com-*

puter Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019, pp. 113–123, Computer Vision Foundation / IEEE, 2019.

[245] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, "Randaugment: Practical automated data augmentation with a reduced search space," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 3008–3017, 2019.

[246] J. W. Gibbs, *Elementary principles in statistical mechanics: developed with especial reference to the rational foundations of thermodynamics*. C. Scribner's sons, 1902.

[247] M. Baldovin, S. Iubini, R. Livi, and A. Vulpiani, "Statistical mechanics of systems with negative temperature," *Physics Reports*, vol. 923, pp. 1–50, 2021.

[248] L. Boltzmann, "Studien uber das gleichgewicht der lebenden kraft," *Wissenschafiliche Abhandlungen*, vol. 1, pp. 49–96, 1868.

[249] R. C. Tolman, *The principles of statistical mechanics*. Courier Corporation, 1979.

APPENDIX A

ADDITIONAL DETAILS FOR THE GRAPH ANALYSIS

A.1 Introduction

Clarifications for how the number of contexts would grow exponentially with depth for a mainstream DL model are provided with an illustrative algorithm. Moreover, a practical example of how the model level quantities are calculated is given.

A.2 Horizontal unrolling algorithm

For illustrative purposes, we provide an algorithm for the horizontal unrolling approach we described. Algorithm 8 takes the original DAG input, creates a new DAG with a root node, and calls $UnrollNode$ for each leaf node. In $UnrollNode$, (Algorithm 9), for each source node (i.e. each node that has an output to the current node), it recursively unrolls until the root node is reached while adding a duplicate of the current node for each source node. This results in a horizontally unrolled graph where the relative frequencies can be directly calculated by counting the duplicates for each learnable parameter. The relative frequencies are treated as a discrete probability distribution. Based on this probability distribution, we derive the model-level quantities for comparison.

Due to the simplicity of Algorithms 8 and 9, similar approaches may already exist for other domains; therefore despite not being able to find a relevant study, we suspect

that we reinvented horizontal unrolling for a new use case. Our main use of the algorithm is to illustrate our point that all DAGs can be converted to a functionally equivalent form where all parameter sharing is made explicit, and in this form, the number of contexts can be directly counted.

For modern deep learning models, the horizontal unrolling algorithm would leave an unrolled graph with an exponentially large number of duplicated components. In practice, for our graph analysis, we do not apply horizontal unrolling to graphs at all due to the complexity. For instance, for MLPs, the time and space complexity would both be in the order of $O(width^{depth})$. We instead use an optimized counting approach. Please see our Github repo for the details: `https://github.com/dlessence/priors/tree/main/reusability`

Algorithm 8 HorizontalUnroll: illustrating how horizontal unrolling can work on any DAG

Input: $Graph$

$unrolledDAG \leftarrow DAG('unrolled')$
$leafNodes \leftarrow GetLeafNodes(Graph)$
for $node$ in $leafNodes$ **do**
    $UnrollNode(node, unrolledDAG)$
end for
return $unrolledDAG$

A.3 An illustration of how we estimate model performances

Step by step, we analyze the graph depicted in Figure 4.3a as an example. For large models, we use an optimized algorithm but for the sake of simplicity, we rely on Algorithm 8 in this example.

1. Horizontally unroll the graph using Algorithms 8 and 9. This results in the graph in Figure 4.3b.

Algorithm 9 UnrollNode: unrolling a specific node in a DAG

Input: $Node, Graph$

1: $duplicate \leftarrow Node(Node.name)$
2: $leafNodes \leftarrow GetLeafNodes(Graph)$
3: **for** $source$ in $Node.sources$ **do**
4: **if** $isRoot(source)$ **then**
5: $root.addTarget(duplicate)$
6: $continue$
7: **end if**
8: $UnrollNode(source, unrolledDAG).addTarget(duplicate)$
9: **end for**

2. For each learnable parameter directly count the repetitions in the unrolled graph, i.e. collect the frequencies: $w_1 = 2, w_2 = 2, w_3 = 2, w_4 = 2, w_5 = 1, w_6 = 1, w_7 = 1, w_8 = 1$

3. Estimate the probabilities as relative frequencies: $p(w_1) = 2/12$, $p(w_2) = 2/12$, $p(w_3) = 2/12$, $p(w_4) = 2/12$, $p(w_5) = 1/12$, $p(w_6) = 1/12$, $p(w_7) = 1/12$, $p(w_8) = 1/12$. We use these probabilities for the calculations of the total surprisal, entropy, and expected spread.

4. **Total surprisal:** $-log_2(2/12) - log_2(2/12) - log_2(2/12) - log_2(2/12) - log_2(1/12) - log_2(1/12) - log_2(1/12) - log_2(1/12) = -4 * (log_2(2/12) + log_2(1/12)) = 24.68$

5. **Entropy:** $-4 * (2/12 * log_2(2/12) + 1/12 * log_2(1/12)) = 2.92$

6. **Expected spread:** $4 * (2/12 * log_2(2) + 1/12 * log_2(1)) = 0.67$

7. **Total surprisal based performance estimation:** input nodes $N_I = 2$ and model size $|G| = 2+1+8 = 11$ and total surprisal=24.68 so $P_G = log_2(24.68 * 2/11) = 2.17$.

8. **Expected spread based performance estimation:** there are 8 learnable pa-

rameters and the expected spread=0.67 so $P'_G = log_2((0.67 + 1) * 8 * 2/11) = 1.28$.

In our GitHub repository, we share multiple examples of how we derive the model-level quantities including the graphs from Figures 4.2a, 4.2b, 4.3a, and 4.3c. The examples are available here: `https://github.com/dlessence/priors/tree/main/reusability`

APPENDIX B

ADDITIONAL EXPERIMENTS WITH PARAMETER SHARING

B.1 Introduction

Some of the experiments on parameter sharing on EfficientNetv2 models were not given in Section 4.5 due to the consistency required in hyperparameters for the graph analysis. The remaining experiments are reported here for comparison with other cross-layer parameter-sharing approaches and implicit models.

B.2 Comparison with Other Cross Layer Parameter Sharing Approaches

In Figure B.1 some of the previous parameter-sharing approaches were compared to the aggressive parameter sharing in Efficientv2 models that rely on the ideas of scope discussed in Sections 3.3.3 and 4.5.3. Note that the difference in performance can be due to multiple reasons such as the model and hyperparameter choices, that are not consistent. Therefore the results should be taken with a grain of salt.

B.3 Comparison with Implicit Models

Implicit models [98, 99, 8, 9, 198, 7, 11, 10] can be considered a form of parameter sharing. In Section 2.3.3.3 this is discussed in more detail. Relatively modern CNN-based models with cross-layer parameter sharing seem to outperform them. Note that

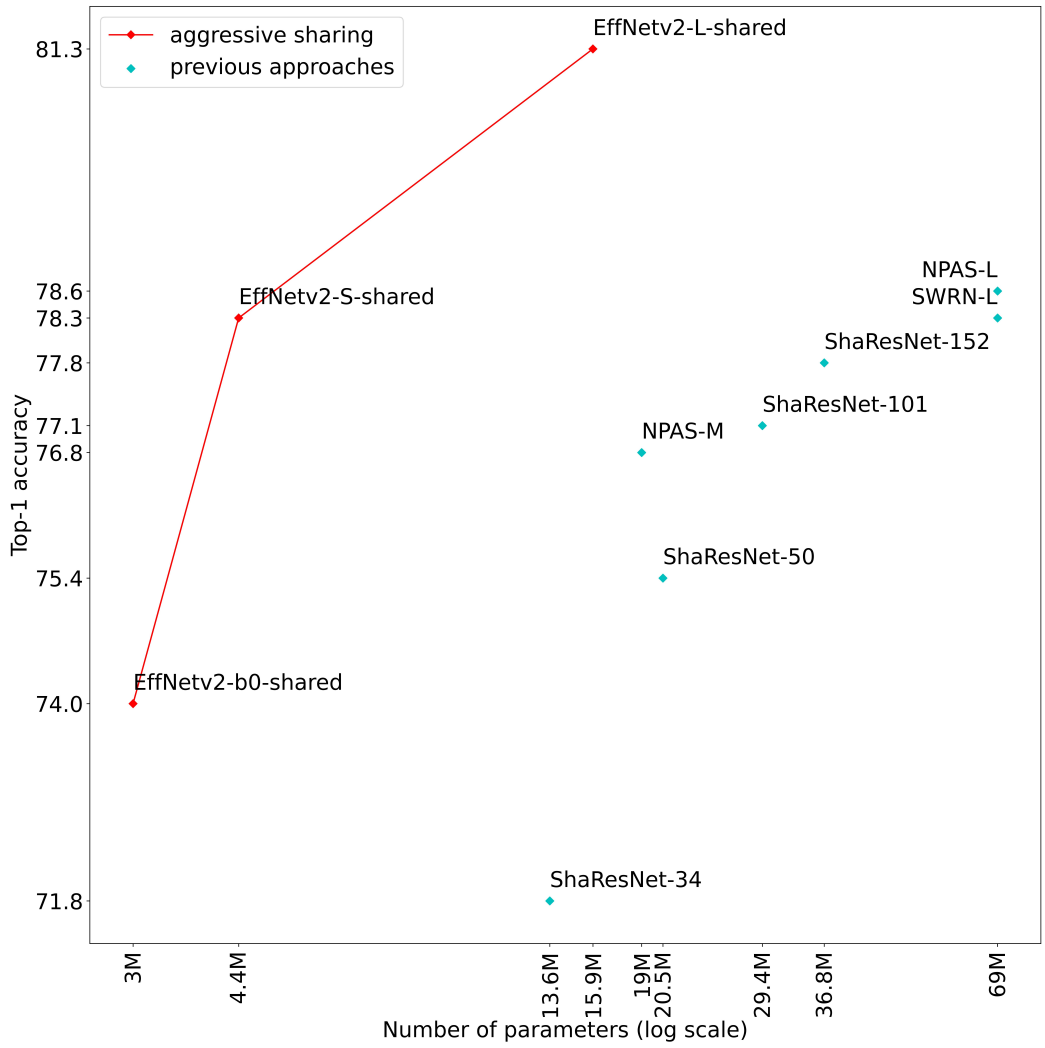


Figure B.1: EfficientNetv2 [1] models with aggressive parameter sharing vs the top-1 accuracy of models from other parameter sharing approaches on ImageNet-1K namely neural parameter allocation search (NPAS) [4], ShaResNet[5], and implicitly recurrent CNNs i.e. SWRN-L [6].

the batch normalization layers are not shared for cross-layer parameter sharing, and this may be an advantage in expressive power as discussed in Chapter 5.

For the CIFAR-10 and Imagenet benchmarks, in Figures B.2 and B.3 where some of the experiment results from EfficientNetv2 [1] is given, it is shown that models with cross-layer parameter sharing with a smaller number of parameters have significantly higher top-1 accuracy than implicit models. Note that the results depict the top-performing models. The results in Table 4.2 shared in Section 4.5 for some of the models are slightly lower as the hyperparameters used in the relevant experiments needed to be fully consistent.

It is possible that this comparison is not fair since implicit models are not studied and optimized as frequently as explicit models. There is a chance that the difference in performance is due to the differences in the quality of hyperparameters or augmentation techniques currently used for the implicit models compared to the relatively well-optimized EfficientNetv2 models.

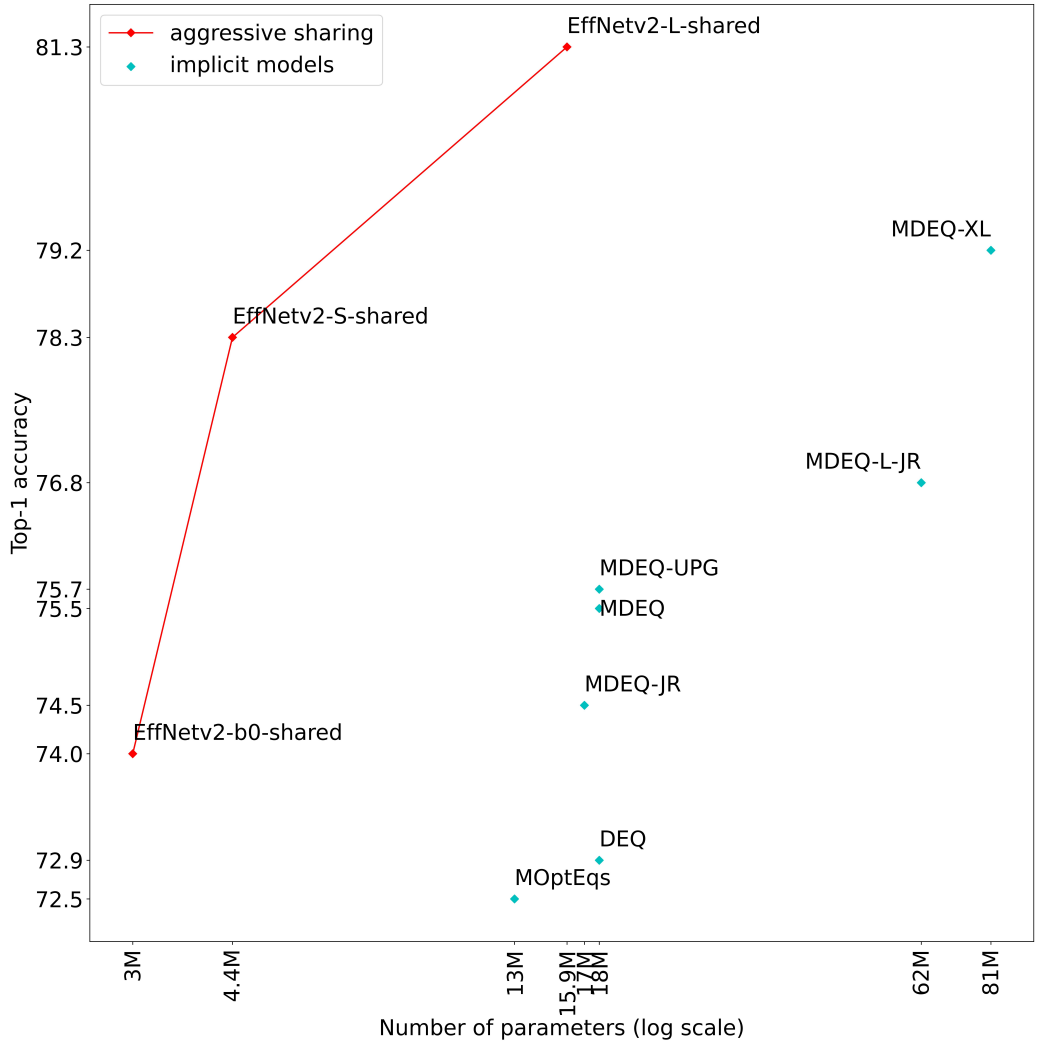


Figure B.2: EfficientNetv2 [1] models with aggressive parameter sharing vs the top-1 accuracy of implicit models on ImageNet-1K namely multi branch equilibrium models MOPTEqs [7], deep eqiulibrium models [8], multi-scale deep equilibrium models (MDEQ) [9], and MDEQ-JR / MDEQ-L-JR [10].

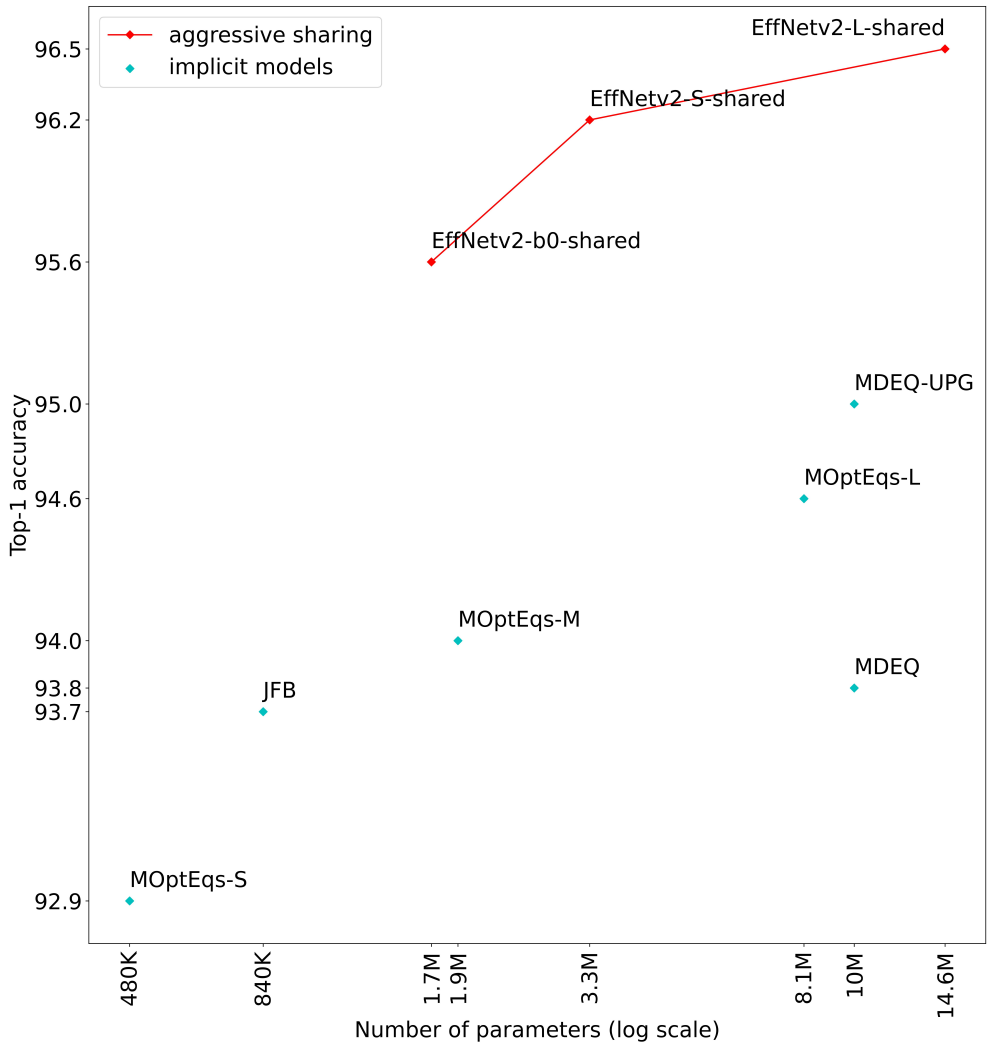


Figure B.3: EfficientNetv2 [1] models with aggressive parameter sharing vs the top-1 accuracy of implicit models on CIFAR-10 namely multi branch equilibrium models MOPTEqs [7], jacobian free backpropagation for implicit models, i.e. JFB [11], and multi-scale deeep equilibrium models (MDEQ) [9].

APPENDIX C

ADDITIONAL DETAILS FOR REVOLVER AND PARAMETER SHARING

C.1 Introduction

Some of the additional details regarding Revolver and parameter sharing are discussed here. The blueprints described in Section 3.3 were extensively in the population training experiments described in Section 3.5. Furthermore, the scope name-based parameter sharing was used in the cross-layer parameter sharing experiments given in B and Section 4.5.6.

In the conducted experiments described in Section 3.5, Revolver allowed tracking the shared modules' scores in the populations. For instance, in Section C.3 some of the top-ranked modules are shared for the top-ranking population for CIFAR-10. In general, 20% to 35% of all shared convolutions had a kernel size of 3x3, while the rest of the convolutions had 1x1. Extensive experiments may reveal whether this is a conclusive pattern about the reusability of convolutions. This is left for future work.

C.1.1 A Complexity Analysis with Parameter Sharing

Parameter sharing may have important repercussions in terms of the model and architecture design. FractalNet [92], ResNet [12], and DenseNet [13] models are analyzed in this regard.

The *join layers* in FractalNet take an average of incoming paths or *fractals* of differ-

ent depths. Larson et al. also introduce the *path drop* method to alleviate overfitting-related issues while training a fractal-like CNN architecture with a large number of parameters [92]. Averaging over alternative paths of various depths has similarities with *block drop* [219] that is used with ResNets. Chen et al. [96] point out that topologically, ResNet and many of its flavors were special cases of DenseNet, and when the blocks are reused, they could be represented as Higher Order RNN (HORNN) [97].

From the perspective of parameter sharing, the below properties can be revealed for FractalNet, ResNet, and DenseNet:

- A FractalNet block with aggressive parameter sharing leads to an exponential compression ratio (i.e. $O(2^n)$ when n is the fractal depth or half the size of the deepest path in the fractal block).
- A ResNet block consisting of architecturally identical layers with aggressive parameter sharing achieves $O(d)$ compression where d is the number of layers in the block.
- When parameters are reused in a specific manner in a DenseNet that uses summation instead of concatenation, it becomes exactly equivalent to ResNet.
- When FractalNet with a different expansion rule is used where at each equivalent depth, the convolutional layers are shared, it becomes equivalent to DenseNet.

C.2 Hyperparameters for the Experiments

In the experiments, various hyperparameters were used for testing Revolver. For instance, `sample_size` was used for denoting the number of individuals to be shortly trained per generation, `warmup_x` first allowed generating several individuals larger than the `population_size` when initializing the population and then selecting

the individuals with the top genotype fitness to fill the population. `warmup_epoch` allowed modifying how many generations would be used for pure genotype search without any score updates from the validation loss. Additional details and hyperparameters on how each experiment for population training was run are given as follows:

- The experiment with the top test accuracy of 99.6 (MNIST Random):

```
--batch_size 512 --search_mode random --sample_size
8--epochs 160 --width 2 --lr 0.35 --min_lr .00085
--genotype_cost 32 --warmup_x 2 --warmup_epoch 30
--population_size 80 --dataset MNIST
```

- The experiment with the top test accuracy of 99.6 (MNIST Evolve):

```
--batch_size 512 --search_mode evolve --sample_size
8--epochs 160 --width 2 --lr 0.35 --min_lr .00085
--genotype_cost 32 --warmup_x 2 --warmup_epoch 30
--population_size 80 --dataset MNIST
```

- The experiment with the top test accuracy of 88.5:

```
--batch_size 128 --search_mode random_warmup
--sample_size 8 --epochs 160 --lr 0.35 --min_lr
.00085 --genotype_cost 32 --warmup_x 2
--warmup_epoch 30 --population_size 80 --dataset
CIFAR10
```

- The experiment with the top test accuracy of 89.6:

```
--batch_size 128 --search_mode random_warmup_evolve
--sample_size 8 --epochs 160 --lr 0.35 --min_lr
.00085 --genotype_cost 32 --warmup_x 2
--warmup_epoch 30 --population_size 80 --dataset
CIFAR10
```

- The experiment with the top test accuracy of 88.8:

```
--batch_size 128 --search_mode random_warmup
--sample_size 8 --epochs 600 --lr 0.35 --min_lr
.00085 --genotype_cost 32 --warmup_x 2
--warmup_epoch 30 --population_size 80 --dataset
CIFAR10
```

- The experiment with the top test accuracy of 90.5:

```
--batch_size 128 --search_mode random_warmup_evolve
--sample_size 8 --epochs 600 --lr 0.35 --min_lr
.00085 --genotype_cost 32 --warmup_x 2
--warmup_epoch 30 --population_size 80 --dataset
CIFAR10
```

- The experiment with the top test accuracy of 90.8:

```
--batch_size 128 --search_mode random --sample_size
16 --epochs 160 --lr 0.35 --min_lr .00085
--genotype_cost 32 --warmup_x 2 --warmup_epoch 30
--population_size 80 --dataset CIFAR10
```

- The experiment with the top test accuracy of 91.9:

```
--batch_size 128 --search_mode evolve --sample_size
16 --epochs 160 --lr 0.35 --min_lr .00085
--genotype_cost 32 --warmup_x 2 --warmup_epoch 30
--population_size 80 --dataset CIFAR10
```

C.3 Scores of the Shared Modules in a Population

Revolver keeps a score dictionary of the shared modules for later analysis. In Table C.1, the top-ranked convolutional layers from the top-performing population for

the CIFAR-10 experiments are shared (i.e. from Table 3.2). Note that further experiments would be necessary for more conclusive results regarding the nature of the most reusable components, but in the conducted experiments, a general observation was that the converged population consisted of a significantly larger number of shared 1x1 convolutions compared to 3x3 convolutions and there seemed to be more shared layers that reduced dimensionality than not.

C.3.0.1 Practical Observations and Notes

The top performing individual in MNIST evolved a mean ensemble of convolutions, convolution followed by deconvolution, and utilized dense concatenation groups. Diversity of components was also achieved to some extent by random search. The random search seemed to be a fair baseline that was relatively easier to implement. Certain crossover and mutation operations on dense concatenation and dense sum groups were not allowed for the sake of simplicity, as changing them often caused the concatenated shape to change.

Revolver's population training is more compute-intensive than a single training session but fairly more efficient than training 80 models sequentially. Training 80 models with CIFAR took from 12 hours to a day based on hyperparameters for Revolver such as how larger the models in the population are, the number of generations, the sample size for training at each generation, batch size, etc. For instance, when the sample size was doubled per generation, the training time was also roughly doubled, taking in between 12-24 hours. The warm-up with the genetic algorithm changed the training time significantly, leading to large initial models and longer training times per batch. They were not included in the reported results.

In the experiments, if all models were trained equally, each model would only get 9 to 18 minutes for training. Yet some models in the population reached 99.6 % top-1 test accuracy for MNIST and 91.9 % for CIFAR in less than 160 generations. Note that these models were not manually designed, but found by the search pipeline during

the population training (Section 3.5.5.1). There were three reasons why some models were able to perform beyond a certain performance threshold:

1. The training budget that was based on model fitness allocated compute resources more efficiently.

2. High-performing models were selected more often for training and score updates.

3. Parameter sharing allowed some models to start from a better initialization point.

In Revolver, it is manually possible to train individual models with cross-layer parameter sharing and achieve a reasonable top-1 accuracy. For instance in B relatively high-performing experiment results were provided for cross-layer parameter sharing with more modern EfficientNetv2 architectures and the implementation of scope from Revolver used in Tensorflow. For population training, however, aggressive cross-layer parameter sharing was likely too susceptible to catastrophic forgetting. The majority of the top performing evolved or randomly selected models had more diverse layer shapes (e.g. bottleneck layers of different shapes) and relatively less amount of cross-layer parameter sharing.

During population training, the learning rate was reduced gradually after the first 100 generations to match the final learning rate by the end of training. For instance, the learning rate may have started from 0.15 and ended at 0.00085, based on the particular hyperparameters used for a given experiment. It is likely that for some experiments the peak test accuracy was achieved relatively early due to the learning rate dropping too much.

The effect of warm-up for the genetic algorithm was that the initial population after the warm-up consisted of larger models compared to a random initialization, with a more chance of higher test scores early on in the training. This did not necessarily

translate to a big difference in the final test accuracy of the top model in the population.

C.4 A Concrete Example of a Blueprint

The following is a description of an arbitrary ResNet-like model with 2 output nodes. The first convolutional layer has a kernel size of 7, and a stride of 2. It does not have a unique suffix in its name, therefore it can be shared with other models, or within the model if it is modified by an algorithm to repeat this layer. The *mutables* can contain the allowed modifications for evolutionary algorithms. Input and output shapes are useful during the actual construction of the model as well as for crossover and copyover operations (Section 3.5). The residual groups can contain residual blocks as children which can contain more convolutional layers. For the sake of brevity, this is skipped in the description (i.e. $'children' : [...]$).


```
{
"args": "[]",
"kwargs": "{'blueprint': 'self',
 'kernel_size': 3, 'stride': 1,
 'padding': 1, 'dilation': 1,
 'groups': 1, 'bias': False}",
"type": "<class 'ScopedResNet'>",
"unique": "True",
"freeze": "False",
"prefix": "ResNetd4w1c2",
"suffix": "",
"parent": null,
"mutables": "{}",
"meta": "{}",
"input_shape": "(1, 3, 32, 32)",
```

```
"output_shape": "(1, 2)",
"name": "ResNetd4w1c2+c3f",
"callback": "<function all_to_none>",
"act": {
"args": "[]",
"kwargs": "{’inplace’: True}",
"type": "<class ’ScopedReLU’>",
"unique": "False",
"freeze": "False",
"prefix": "ResNetd4w1c2/act",
"suffix": "64",
"mutables": "{}",
"meta": "{}",
"input_shape": "None",
"output_shape": "None",
"name": "ResNetd4w1c2/act64",
"parent": "ResNetd4w1c2+c3f",
"children": []
},
"conv": {
"args": "[]",
"kwargs": "{’blueprint’: ’self’,
’kernel_size’: 7, ’stride’: 2,
’padding’: 3, ’dilation’: 1,
’groups’: 1, ’bias’: False}",
"type": "<class ’ScopedConvUnit’>",
"unique": "True",
"freeze": "False",
"prefix": "ResNetd4w1c2/conv",
"suffix": "3_16_7_1_1_1_1_0",
```

```
"mutables": "{}",
"meta": "{}",
"input_shape": "(1, 3, 32, 32)",
"output_shape": "(1, 16, 8, 8)",
"name": "ResNetd4w1c2/conv3_16_7_1_1_1_1_0+b1c",
"module_order": "('conv', 'bn', 'act', 'pool')",
"callback": "<function all_to_none>",
"conv": {
"args": "[]",
"kwargs": "{'in_channels': 3,
'out_channels': 16, 'kernel_size': 7,
'stride': 2, 'padding': 3, 'dilation': 1,
'groups': 1, 'bias': False}",
"type": "<class 'ScopedConv2d'>",
"unique": "False",
"freeze": "False",
"prefix": "ResNetd4w1c2/cnv/conv",
"suffix": "3_16_7_2_3_1_1_F",
"mutables": "{}",
"meta": "{}",
"input_shape": "(1, 3, 32, 32)",
"output_shape": "(1, 16, 16, 16)",
"name": "ResNetd4w1c2/cnv/conv3_16_7_2_3_1_1_F",
"mutation_p": "0.8",
"toggle_p": "0.02",
"parent": "ResNetd4w1c2/cnv3_16_7_1_1_1_1_0+b1c",
"children": []
},
"bn": {
"args": "[]",
```

```
"kwargs": "{’num_features’: 16, ’eps’: 1e-05,
’momentum’: 0.1, ’affine’: True}",
"type": "<class ’ScopedBatchNorm2d’>",
"unique": "True",
"freeze": "False",
"prefix": "ResNetd4w1c2/cnv/bn",
"suffix": "16_16_7_2_3_1_1_F",
"mutables": "{}",
"meta": "{}",
"input_shape": "(1, 16, 16, 16)",
"output_shape": "(1, 16, 16, 16)",
"name": "ResNetd4w1c2/cnv/bn16_16_7_2_3_1_1_F+e89",
"mutation_p": "0.8",
"parent": "ResNetd4w1c2/cnv3_16_7_1_1_1_1_0+b1c",
"children": []
},
"act": {
"args": "[]",
"kwargs": "{’inplace’: True}",
"type": "<class ’ScopedReLU’>",
"unique": "False",
"freeze": "False",
"prefix": "ResNetd4w1c2/cnv/act",
"suffix": "16_16_7_2_3_1_1_F",
"mutables": "{}",
"meta": "{}",
"input_shape": "None",
"output_shape": "(1, 16, 16, 16)",
"name": "ResNetd4w1c2/cnv/act16_16_7_2_3_1_1_F",
"parent": "ResNetd4w1c2/cnv3_16_7_1_1_1_1_0+b1c",
```

```
"children": []
},
"pool": {
"args": "[]",
"kwargs": "{'kernel_size': 3, 'stride': 2,
'padding': 1}",
"type": "<class 'ScopedMaxPool2d'>",
"unique": "False",
"freeze": "False",
"prefix": "ResNetd4w1c2/cnv/pool",
"suffix": "3_2_1",
"mutables": "{}",
"meta": "{}",
"input_shape": "(1, 16, 16, 16)",
"output_shape": "(1, 16, 8, 8)",
"name": "ResNetd4w1c2/cnv/pool3_2_1",
"parent": "ResNetd4w1c2/cnv3_16_7_1_1_1_1_0+b1c",
"children": []
},
"mutation_p": "0.8",
"toggle_p": "0.02",
"parent": "ResNetd4w1c2+c3f",
"children": []
},
"depth": "3",
"shortcut_index": "-1",
"linear": {
"args": "[]",
"kwargs": "{'in_features': 16,
'out_features': 2}",
```

```
"type": "<class 'ScopedLinear'>",
"unique": "True",
"freeze": "False",
"prefix": "ResNetd4w1c2/linear",
"suffix": "16_2",
"mutables": "{}",
"meta": "{}",
"input_shape": "(1, 16)",
"output_shape": "(1, 2)",
"name": "ResNetd4w1c2/linear16_2+a52",
"parent": "ResNetd4w1c2+c3f",
"children": []
},
"mutation_p": "0.8",
"children": [
{
"args": "[]",
"kwargs": "{'blueprint': 'self',
'kernel_size': 3, 'stride': 1,
'padding': 1, 'dilation': 1,
'groups': 1, 'bias': False}",
"type": "<class 'ScopedResGroup'>",
"unique": "False",
"freeze": "False",
"prefix": "ResNetd4w1c2/group",
"suffix": "16_16_3_1_1_1_1_0",
"mutables": "{}",
"meta": "{}",
"input_shape": "(1, 16, 8, 8)",
"output_shape": "(1, 16, 8, 8)",
```

```
"name": "ResNetd4w1c2/g16_16_3_1_1_1_1_0",
"drop_p": "0.0",
"callback": "<function all_to_none>",
"depth": "0",
"mutation_p": "0.8",
"toggle_p": "0.02",
"parent": "ResNetd4w1c2+c3f",
"children": [...]
},
{
"args": "[]",
"kwargs": "{'blueprint': 'self',
'kernel_size': 3, 'stride': 2,
'padding': 1, 'dilation': 1,
'groups': 1, 'bias': False}",
"type": "<class 'ScopedResGroup'>",
"unique": "False",
"freeze": "False",
"prefix": "ResNetd4w1c2/group",
"suffix": "16_32_3_2_1_1_1_0",
"mutables": "{}",
"meta": "{}",
"input_shape": "(1, 16, 8, 8)",
"output_shape": "(1, 16, 8, 8)",
"name": "ResNetd4w1c2/g16_32_3_2_1_1_1_0",
"drop_p": "0.0",
"callback": "<function all_to_none>",
"depth": "0",
"mutation_p": "0.8",
"toggle_p": "0.02",
```

```
"parent": "ResNetd4w1c2+c3f",
"children": [...]
},
{
"args": "[]",
"kwargs": "{'blueprint': 'self',
 'kernel_size': 3, 'stride': 2,
 'padding': 1, 'dilation': 1,
 'groups': 1, 'bias': False}",
"type": "<class 'ScopedResGroup'>",
"unique": "False",
"freeze": "False",
"prefix": "ResNetd4w1c2/group",
"suffix": "16_64_3_2_1_1_1_0",
"mutables": "{}",
"meta": "{}",
"input_shape": "(1, 16, 8, 8)",
"output_shape": "(1, 16, 8, 8)",
"name": "ResNetd4w1c2/g16_64_3_2_1_1_1_0",
"drop_p": "0.0",
"callback": "<function all_to_none>",
"depth": "0",
"mutation_p": "0.8",
"toggle_p": "0.02",
"parent": "ResNetd4w1c2+c3f",
"children": [...]
}
],
"bns": [
{
```

```
"args": "[]",
"kwargs": "{'num_features': 16}",
"type": "<class 'ScopedBatchNorm2d'>",
"unique": "True",
"freeze": "False",
"prefix": "ResNetd4w1c2/bn",
"suffix": "16",
"mutables": "{}",
"meta": "{}",
"input_shape": "None",
"output_shape": "None",
"name": "ResNetd4w1c2/bn16+a47",
"parent": "ResNetd4w1c2+c3f",
"children": []
}
]
}
```


C.5 Tracking The Mutation History

A sample model found with the genetic algorithm in one of the CIFAR-10 experiments with 790 K parameters and 89.6% top-1 accuracy is given as an example of how the mutation history of a model component can be tracked and how a mutation can change the hierarchy of a component from a single layer to multiple layers. The following architecture belongs to the actual PyTorch model:

```
ScopedDenseNet(
(container): ModuleList(
...
```

```
)
(conv): ScopedConvUnit(
--(sequence): Sequential(
----(conv): ScopedEnsembleMean(
------(container): ModuleList(
--------(0-2): 3 x ScopedConv2d(3, 32,
--------------------------------kernel_size=(3, 3),
--------------------------------stride=(1, 1),
--------------------------------padding=(1, 1),
--------------------------------bias=False)
------------)
--------)
----)
)
(bn): ...
(act): ...
(linear): ...
)
```

This model's blueprint reveals that the given convolutional container unit was previously a single convolutional layer. Then it mutated to become a convolutional unit that contained an ensemble of three convolutional layers which is visible inside the outermost `conv` item. The previous one-layer state is visible in `mutation_history` inside the `meta` item:

```
...
"meta": {
-mean_score: 0.29296087378436786,
-score: 0.2929455813112225,
-conv: {
--mutation_history: [
```

```
----{
------args: [],
------kwargs: {...},
------type: <class 'ScopedConv2d'>,
------unique: True,
------freeze: False,
------input_shape: (128, 3, 32, 32),
------output_shape: (128, 32, 32, 32)
------...
----}
--]
-}
}
...
"type": <class 'ScopedConvUnit'>,
"conv": {
-type: <class 'ScopedEnsembleMean'>,
...
children: [{...}, {...}, {...}]
}
```

Table C.1: Top scoring (lower is better) convolutional layers shared across models from the top performing population for CIFAR-10

Input Channels	Output Channels	Kernel Size	Score
512	128	1x1	0.52
128	32	1x1	0.56
256	64	3x3	0.59
128	32	3x3	0.59
128	128	1x1	0.60
256	64	1x1	0.61
64	16	3x3	0.65
64	16	1x1	0.66
80	64	1x1	0.69
1024	256	1x1	0.69
136	128	1x1	0.69
128	32	3x3	0.70
96	16	1x1	0.70
160	128	1x1	0.70
16	16	1x1	0.71

VITA

AYDIN GÖZE POLAT

MACHINE LEARNING ENGINEER

SUMMARY

I am a machine learning engineer who is experienced in deep learning-based solutions for real-world problems for computer vision and natural language processing. I am interested in constant learning and research with a real-world impact in mind.

EMPLOYMENT

Square, Machine Learning Engineer / Modeler, Toronto, Feb 2020 - Jan 2022

I created new models based on cutting-edge deep learning (DL) research, particularly for Cash App, and helped people learn & adopt new DL approaches.

Dessa, Machine Learning Engineer, Toronto, Oct 2018 - Feb 2020

I created an end to end deep learning model from scratch that improved the accuracy of OCR on checks from 72% to 98.5% (deep learning model).

BuddyGuard, Machine Learning Engineer, Berlin, May 2016-Jan 2018

TUBITAK, Researcher, Ankara, Oct 2015 -Apr 2016

EDUCATION

Middle East Technical University

Ph.D. Computer Engineering / Machine Learning, 2023

M.S. Cognitive Science, 2012

B.S. Computer Engineering, 2010