The reusability prior: comparing deep learning models without training

IOP Publishing Mach. Learn.: Sci. Technol. 4 (2023) 025011 https://doi.org/10.1088/2632-2153/acc713

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
5 December 2022

REVISED
20 March 2023

ACCEPTED FOR PUBLICATION
23 March 2023

PUBLISHED
20 April 2023

PAPER

The reusability prior: comparing deep learning models without training

Aydın Göze Polat* and **Ferda Nur Alpaslan**

Department of Computer Engineering, Middle East Technical University, Ankara 06800, Turkey
* Author to whom any correspondence should be addressed.

E-mail: goze.polat@metu.edu.tr

Keywords: entropy, deep learning, parameter efficiency, reusability

Abstract

Various choices can affect the performance of deep learning models. We conjecture that differences in the number of contexts for model components during training are critical. We generalize this notion 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. We focus on the design aspect and introduce a graph-based methodology to estimate the number of contexts for each learnable parameter. This allows a comparison of models without requiring any training. We provide supporting evidence with experiments using cross-layer parameter sharing on CIFAR-10, CIFAR-100, and Imagenet-1K benchmarks. We give examples of models that share parameters outperforming baselines that have at least 60% more parameters. The graph-analysis-based quantities we introduced for the reusability prior align well with the results, including at least two important edge cases. We conclude that the reusability prior provides a viable research direction for model analysis based on a very simple idea: counting the number of *contexts* for model parameters.

1. Introduction

Artificial neural networks with larger numbers of parameters often outperform their smaller counterparts. For instance, larger models achieve state-of-the-art in various benchmarks in computer vision [1–12] and natural language processing [13–17]. Yet there are exceptions to this pattern where some models 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 [18]. Hoffmann *et al* prove that it is possible to outperform previous state-of-the-art models with significantly more compact ones by training with more data. They provide a systematic analysis that reveals that increasing the number of parameters and increasing the size of the training set are equally important for optimal usage of compute resources for training efficient models [19].

Model performance is affected from not only the number of parameters and training samples, but also the design, augmentation, and regularization choices. We conjecture that in essence these choices impact the overall number of concrete *contexts* for model parameters.

1.1. What is a context?

In figure 1, there is an informal description of *context*. We define the set of all contexts for a given parameter 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

$$\mathcal{C}_w = \bigcup_{f_{iw} \in \mathcal{G}} \{p \mid f_{iw}((h_i, w), \ldots) \overset{p}{\rightsquigarrow} \mathcal{O}_k \quad \forall \mathcal{O}_k \in \mathcal{L}\}. \tag{1}$$

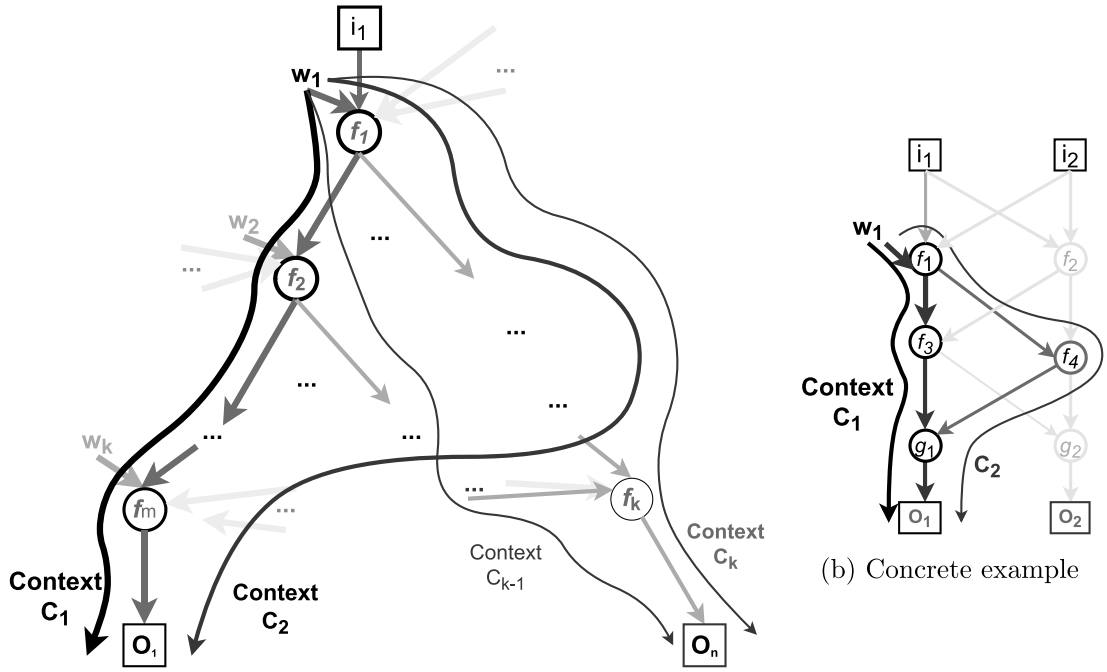

Figure 1. A context is a path from a parameter associated with an input to an output. (a) C_1 is the bold path from w_1 associated with i_1 through functions $f_1, f_2, \ldots f_m$ contributing to the output O_1. Note that w_1 can contribute to O_1 through multiple contexts (e.g. C_1 and C_2). (b) A concrete analogy for C_1 and C_2 given on a small architecture.

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. $f_{jw}(h_j, w, \ldots)$). 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 we make 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 a part of the *computational graph* rather than an instance of an actual output feature from a previous layer. We use this definition of context for analyzing model architectures and defining other relevant concepts. In figure 1, we demonstrate some of the possible contexts that can arise inside a model graph. In figure 1(b) the computational graph of a model with three layers is described. How can model design change the number of possible contexts? We provide an example in figure 2, comparing two configurations that yield different numbers of contexts for the same number of parameters.

1.2. 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 instance, convolutional kernels [20, 21] repeat spatially (i.e. sliding window), while recurrent modules [22–26] (or any autoregressive model such as [27–29]) 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 reasonable chance to outperform the original baseline model [30–35]. In section 5.6 we confirm a similar outcome with our examples on some of the EfficientNetv2 models [36].

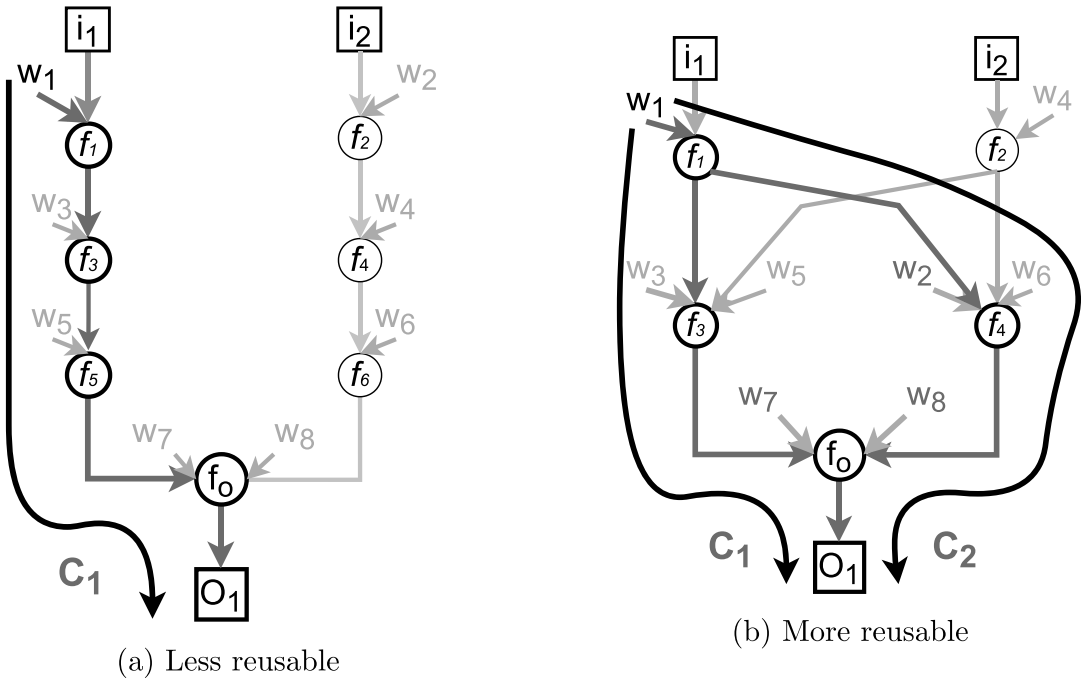

Figure 2. Given the same number of parameters, model design can change parameter efficiency: (a) w_1 contributes to a single context C_1 (b) w_1 contributes to an additional context C_2. The reusability prior suggests that w_1 is likely to be forced to function for both contexts and hence will become 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.

1.3. Improving parameter efficiency

We claim that all mainstream DL models already share parameters directly or indirectly to various degrees, and this affects their parameter efficiency. In section 4.1, we introduce a simple horizontal unrolling approach that can make parameter sharing explicit for all directed acyclic graphs (DAG).

We propose the generalized notion *reusability prior* to disentangle the intrinsic design and training choices that affect a model's performance in section 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. Our experiments on EfficientNetv2 as well as the literature on cross-layer parameter sharing provide supporting evidence for the reusability prior.

We introduce a simple counting approach for model comparison. By treating relative frequencies derived from the number of all possible contexts per learnable parameter as a probability distribution, we define quantities for the comparison of model graphs, including *entropy*, *expected spread*, and *total surprisal*. We give formal proof that when the total number of contexts is held constant, increasing the expected spread reduces the entropy. We also introduce approaches to estimate performance for directly comparing models without training. We compare analysis and experiment results in tables 3 and 4. We scope our work by only focusing on the model design aspect described in section 2.2.3. The data and training aspects can be taken into account as well when considering the number of possible contexts. We leave this for future work.

In summary, our major contributions are twofold:

- We introduce the reusability prior (section 4.2), and provide a methodology based on graph analysis (section 4.3). To our knowledge, we are the first to introduce a generalized notion of reusability that ties training, design, and data aspects together. For the design aspect, we introduce graph analysis based quantities derived from counting the number of contexts for each learnable parameter[1]. Overall, the quantities we introduce allow us to compare arbitrary DAGs or model architectures without relying on any training.

[1] To our knowledge, the majority of our definitions, lemmas, theorems, and proofs are original, except for what we borrowed from the information theory, namely entropy, surprisal, expected value, and an existing identity regarding Kullback–Leibler divergence used in 4.4.1. Despite not being able to find them in other relevant studies, we suspect that the horizontal unrolling approach and the idea of a uniform graph we introduced in section 4.1 likely already exist in other disciplines, and we reinvented them for our use case.

Our graph analysis based quantities aligned well with the empirical results, including at least two important edge cases in section 5 in tables 3 and 4.

- We empirically confirm that it is possible to achieve higher parameter efficiency by aggressively sharing parameters in EfficientNetv2 models. In our experiments, even though the numbers of parameters were at least 60% larger for the original baseline EfficientNetv2-b0 models, we observed that EfficientNetv2-S models with aggressive parameter sharing consistently outperformed the baselines on CIFAR-10 [37], CIFAR-100 [38], and Imagenet-1K [39] image classification benchmarks.

2. Background

Fundamentally, convolutional and recurrent neural networks (RNNs) rely on architectural priors based on reusing model components through space and time. Implicitly, RNNs share recurrent modules through time [40] and convolutional neural networks (CNNs) share convolutional kernels through space. Moreover, it is possible to share 3d convolution kernels spatiotemporally [41]. Reusing components in various ways is an important underlying pattern as larger and larger architectures are adopted [33, 35, 42–48], searched [48–56], skip connections are utilized [57–60], domain specific symmetries are captured [61–66], model size is reduced or manipulated [30, 31, 67–72], and parameters are either globally shared or transferred from different domains or models [48, 73–81].

DenseNet models are claimed to be *compact* due to the concatenation of previous features at each subsequent layer [58]. Another strategy for compactness is used by Xception, or 'Extreme Inception' [82]. It minimizes the parameters from 3×3 convolutions by using *depthwise* 3×3 convolution layers, i.e. one independent convolution kernel per channel. Then it relies on 1×1 convolutions which allow reusing the depthwise convolution outputs by taking the linear combinations of the outputs. 1×1 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 [83, 84] also focus on maximizing the 1×1 convolution operations. Achieving compactness via soft sharing [85], student-teacher architectures [86, 87], pruning and quantization [88] 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* introduce *batch entropy* regularization that allows training very deep models without skip connections [89]. Wickstrøm *et al* analyze different training phases of deep models with information plane theory using Rényi's entropy [90]. 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 [91]. 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 [92]. Our work diverges from Bu *et al* as we use Shannon entropy [93] and provide a methodology that can work with any DAG without requiring constant width or a specific model. Furthermore, we focus on parameter efficiency, thus we opt for a probability distribution that is based on the relative frequencies derived from the number of contexts for each parameter.

2.1. Cross-layer parameter sharing

SharesNet [30] demonstrates that for wide residual networks (WRNs) [94], 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 [95], Savarese and Maire train a model that learns to take a linear combination of a shared pool of kernels [31]. 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 [32], inspired from [96], 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 [42], ResNet, and WRN baselines. Shapeshifter networks used for neural parameter allocation search 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 [69].

Aside from the computer vision domain, a lite BERT, bidirectional encoder representations from transformers [33] uses cross-layer parameter sharing to first reduce the number of parameters and then scale up to a capacity comparable to the original model [97]. This surpasses the original model's performance for language understanding tasks. Other relevant examples share attention weights [34], speed up training by parameter sharing and then unsharing [98], and explore sandwich style weight sharing for generative transformers [35].

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 [99, 100], variants of deep equilibrium models (DEQ), and multiscale DEQs [101, 102].

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 1.1, and figure 1 we provide more details about what is meant by *context*. Essentially, the number of contexts is affected from data, augmentation, training, regularization, and model design choices.

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 [103]), or combining multiple samples (e.g. mixup [104], cutmix [105] etc).

2.2.2. Diversifying the roles of model components during training

By diversifying the hidden representations, regularization such as dropout [106], stochastic depth [107], and block drop [60] increase the total number of contexts that can arise from the same training data.

2.2.3. Design choices impacting the role and scope of each model component

Decisions such as network depth and width, having CNN layers [20, 108], recurrent modules [40], residual connections [57], attention layers [109], training input size, and cross-layer parameter sharing [30] 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. We generalize this notion as *the reusability prior* in section 4.2.

3. Overview

We propose a generalized notion of reusability and the reusability prior that encompasses training, data, and model design aspects in deep learning in section 4. We then focus on the model design aspect to analyze and compare models with graph analysis. In addition to the supporting evidence from the literature discussed in section 2, we conduct our own experiments by applying aggressive cross-layer parameter sharing to some EfficientNetv2 [36] models. We share the results from this strategy on image classification benchmarks in section 5. We analyze the computational graphs of EfficientNetv2 models and compare the quantities based on the reusability prior with the empirical results in sections 5.7 and 5.8.

4. The notion of reusability

We claim that maximizing the number of *contexts* for learnable parameters is critical. For instance, 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. Intrinsically, any deep learning model can be converted to a functionally equivalent form where parameter sharing is made explicit.

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. We illustrate our point in figure 3. The nodes in earlier layers are duplicated more as they play more diverse roles. We introduce a simple recursive algorithm for horizontal unrolling in appendix A.

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 their number of *contexts* that can arise due to the computational graph. To our knowledge, no mainstream DL model exponentially grows in terms of the number of parameters with depth. Hence their unrolled graphs share parameters. This manner of parameter sharing in the unrolled graph uses exponentially fewer parameters than a more general graph exemplified in figure 3(c) that does not share any parameters. We call such graphs a *uniform graph* and formally define them as follows:

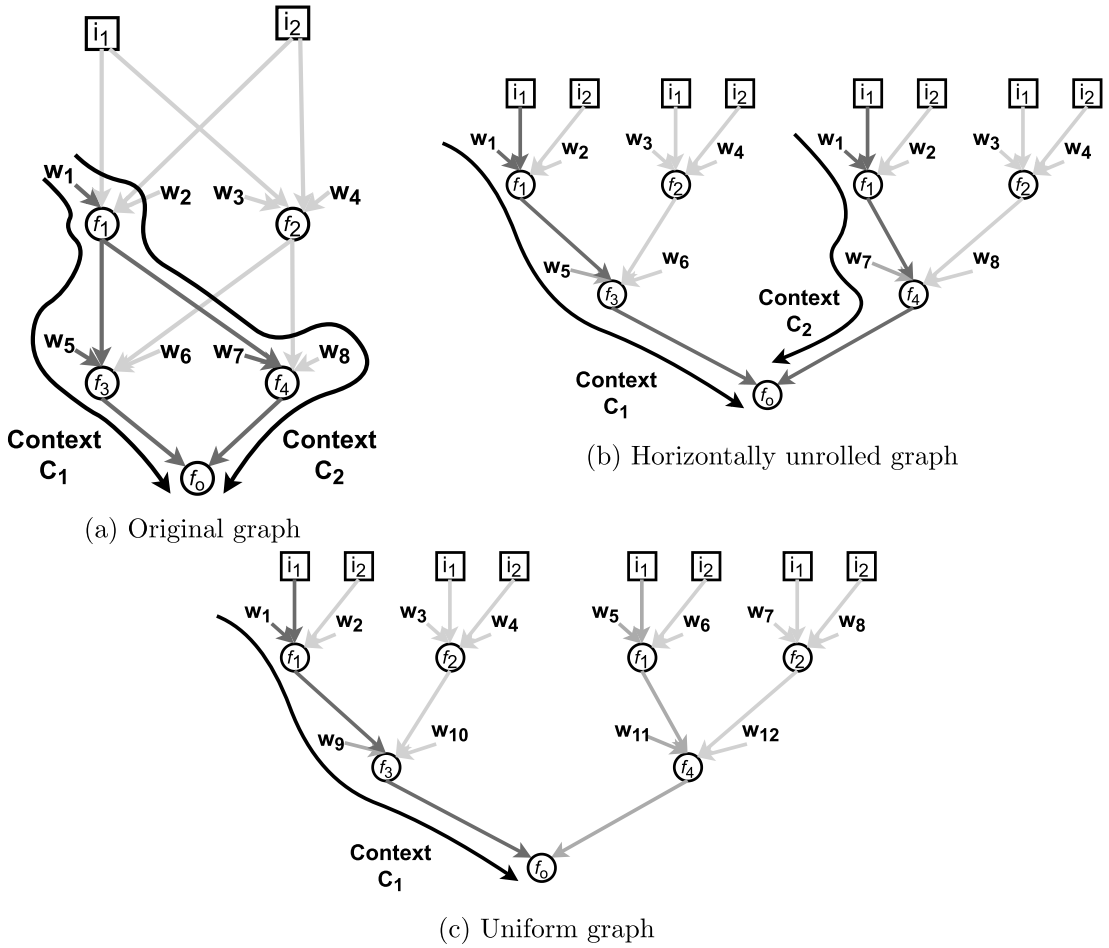

Figure 3. Horizontal unrolling reproduces all shared node outputs from scratch, eliminating indirect parameter sharing or feature reuse and instead replacing it with direct parameter sharing with duplicated parameters. (a) w_1 in the original graph has contexts C_1 and C_2 (b) w_1 is duplicated as many 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.

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, we quantify expected spread, total surprisal, and entropy for computational graphs of models.

4.2. The reusability prior

We conjecture that the expected number of *contexts* for model components is the major reason for differences in model performance. We introduce 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. 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.

We provide justifications from the literature in section 2, definitions in sections 1.1, and 4.3, finally supporting evidence from our experiments in section 5.

4.3. Quantities for model comparison

Based on the reusability prior, and the notion of context, we define the expected spread, entropy, and total surprisal. Then we provide two different ways of estimating model performance based on total surprisal and expected spread.

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

$$E[[\log_2 |\mathcal{C}|]] = \sum_{i=1}^{N_\mathcal{G}} p(w_i) \log_2 |\mathcal{C}_{w_i}| \tag{2}$$

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

$$\frac{|\mathcal{C}_{w_i}|}{N_\mathcal{C}} \tag{3}$$

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

In section 4.4.1 we prove that the expected spread is equal to the relative entropy or Kullback–Leibler divergence [110] of $P(W)$ from the discrete uniform distribution.

Overall, this quantity prescribes a design maximizing the expected number of contexts[2]. Note that $N_\mathcal{C}$ and $N_\mathcal{G}$ grow differently. In 2, similar to $N_\mathcal{C}$, $|\mathcal{C}_{w_i}|$ 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 by

$$H(W) = -\sum_{i=1}^{N_\mathcal{G}} p(w_i) \log p(w_i) \tag{4}$$

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

Note that $p(w_i)$ is also used in the calculation of expected spread in 3.

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

$$\mathcal{S}_\mathcal{G} = -\sum_{i=1}^{N_\mathcal{G}} \log p(w_i). \tag{5}$$

Since the multiplicative $p(w_i)$ term is removed, this is no longer the expected surprisal, i.e.entropy in 4. Consequently, this quantity gives the surprisal of each parameter equal weight[3].

4.4. Maximizing the expected spread minimizes the entropy

If we consider parameters that affect smaller portions of the model (i.e. with a smaller *spread*) 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 $\log_2 N_\mathcal{G}$.

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.

[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.

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

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 $D_{\mathrm{KL}}(P(W)||P_U(W))$ from the discrete uniform distribution $P_U(W)$ is equivalent to the expected spread.*

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

$$D_{\mathrm{KL}}(P(W)||P_U(W)) = \sum_{i=1}^{n} p(w_i)\log_2(p(w_i)/p_U(w_i)) = \sum_{i=1}^{n} p(w_i)\log_2(c_i/N_{\mathcal{C}}/(1/N_{\mathcal{C}})) = \sum_{i=1}^{n} p(w_i)\log_2(c_i)$$
$$= E[[\log_2|\mathcal{C}|]].$$

□

Note that, if $P(W)$ is derived from the graph in figure 3(a) by directly counting the frequencies from its unrolled version in figure 3(b), $P_U(W)$ can correspond to the uniform graph in figure 3(c). In general, the cases where $p(w_j) = 0$ and $p_U(w_j) = 1/N_{\mathcal{C}}$ do not change the summation.

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

Note that $N_{\mathcal{C}}$ is equivalent to the number of parameters in G's horizontally unrolled *uniform* graph as in figures 3(c) 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 $N_{\mathcal{C}}$ is held constant, the theorem still holds. The formal proof is as follows:

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

$$H_U(X) = H(X) + D_{\mathrm{KL}}(P(X)||P_U(X)).$$

Therefore, from lemma 1 we show that KL divergence is always:

$$D_{\mathrm{KL}}(P(W)||P_U(W)) = E[[\log_2|\mathcal{C}|]].$$

Thus, we can write:

$$H_U(W) = H_{\mathcal{G}}(W) + D_{\mathrm{KL}}(P(W)||P_U(W)) = H_{\mathcal{G}}(W) + E[[\log_2|\mathcal{C}|]] = \log_2 N_{\mathcal{C}}.$$

□

Hence, when $N_{\mathcal{C}}$ is constant, reducing the entropy would increase the expected spread and vice versa.

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 the number of parameters as well as the training data and conditions are similar. 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, we propose 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. We normalize this quantity for different model sizes and multiply it 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

$$\mathcal{P}_{\mathcal{G}} = \log_2\left(\mathcal{S}_{\mathcal{G}}\frac{N_I}{|\mathcal{G}|}\right). \tag{6}$$

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

[4] If one takes into account the full scope of the reusability prior, e.g. the literature in section 2.2, then diversifying input and diversifying the role of model components with regularization would also increase the expected spread. We leave this for future work.

4.5.1. An alternative estimation of model performance

As a potential alternative to the total surprisal in (6), expected spread multiplied by the number of learnable parameters $N_{\mathcal{G}}$ can be used (i.e. replacing $\mathcal{S}_{\mathcal{G}}$ with $N_{\mathcal{G}} E[\![\log_2 |\mathcal{C}| + 1]\!]$). This alternative estimation is given by:

$$\mathcal{P}'_{\mathcal{G}} = \log_2 \left(N_{\mathcal{G}} E[\![\log_2 |\mathcal{C}| + 1]\!] \frac{N_I}{|\mathcal{G}|} \right). \tag{7}$$

Our main justification for using expected spread is given in section 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. We observe this in table 2 for multiple experiments and datasets.

Overall, by relying on our graph-based analysis and counting approach, we proposed two crude ways to estimate the performance of models without any training[5]. In practice, both have different strengths and weaknesses that we discuss in section 6.

5. Methodology and experiments

We analyzed the computational graphs of EfficientNetv2 [36] models, without training, to estimate the quantities described in section 4.3. For training models from scratch in our experiments, we adopted the same EfficientNetv2 models to explore the effects of aggressive parameter sharing. We then compared the empirical results with the results from our graph analysis based quantities. We used Tensorflow [111] for the experiments and our own Python [112] library for the graph-based analysis.

5.1. Analyzing computational graphs

We quantify the predictions from the reusability prior for model graphs as follows:

(i) We 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 3(a) it is possible to estimate the relative frequencies from its unrolled form in figure 3(b). Please see appendix B for the full example.).
(ii) The resulting probability distribution allows deriving model-level quantities that we described in section 4.3.
(iii) We use total surprisal and expected spread for estimating model performance as described in section 4.5.

In practice, since the EfficientNetv2 and ResNet-50 models have convolutional layers, we took into account the *full computational graph* that would depend on aspects such as image size, strided convolutions, batch normalization, and pooling layers. Thus we included all learnable parameters from convolutional kernels, batch normalizations, and final fully connected layers for classification which have bias weights. The Python implementation of our 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. Our full code release to reproduce the analysis results is available at [113].

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, we conducted multiple experiments on EfficientNetv2 models. We focused on comparing EfficientNetv2-B0, and EfficientNev2-S models. For cross-layer parameter sharing, our experiments revealed results in agreement with the literature discussed in section 2.1. To make it easier to minimize confounding variables due to data and training strategies, we limited the models we trained from scratch to only EfficientNetv2. We conducted our training in a controlled setting, spanning three different benchmarks. This helped us eliminate differences due to hardware and hyperparameters for the selected models.

5.3. Aggressive parameter sharing

For the EfficientNetv2 experiments, we modified the official Tensorflow implementation [114] to be able to optionally apply aggressive parameter sharing. This strategy relies on roughly treating weight matrices of the

[5] We share both estimation results in tables 3 and 4 as 'P_G.' and 'P'_G.'.

Table 1. Details on CIFAR and Imagenet datasets.

Dataset	Train	Validation	Classes
CIFAR-10 [38]	50 000	10 000	10
CIFAR-100 [38]	50 000	10 000	100
Imagenet-1K [115]	1.28M	50 000	1000

same shape as the same. One exception is that we did not share the batch normalization weights to help the training stay stable.

During the initialization of models with parameter sharing, we keep a global *scope* dictionary. Convolutions of the same scope are only created once. Convolutional layers are represented as four-dimensional matrices where their dimensionality is defined by channel and kernel sizes. More precisely, we map each shared convolution to a scope name that is constructed by combining the number of input channels, number of output channels, kernel size, and strides.

5.4. Hyperparameters

For cross-layer parameter sharing, we used a new hyperparameter which aggressively shares convolutions but uses separate batch normalization layers: `model.weight_sharing = all_but_bn`. *The only change* we made to the hyperparameters when comparing an original model with one that shares convolutional layers is we used `model.weight_sharing = None` for the original one. We closely followed the default hyperparameters for EfficientNetv2 given by Tan and Le [36], except for the following changes:

- For all models trained with CIFAR-10, instead of transfer learning, we trained from scratch. We used no augmentation and trained for 300 epochs without any training stages. We used the following hyperparameters:
  `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, we followed the same scenario and hyperparameters, 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, we used the following hyperparameters:
  `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, we observed lower performance for both the EfficientNetv2-B0 and EfficientNetv2-S models compared to [36]. Larger batch sizes and training longer can sometimes improve performance but we observed a substantial gap in performance between V2-S-shared vs V2-B0-original models regardless. In our previous experiments with Imagenet-1K, we tested different hyperparameters such as gclip, batch size, as well as a larger number of training epochs. Moreover, for CIFAR-100 we tested 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 2.

5.5. Datasets

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

5.6. Experiment results from aggressive parameter sharing

When EfficientNetv2 models are trained from scratch, we observed a consistent increase 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 2.

Table 2. EfficientNetV2 [36] 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) achieve 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 [116]	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 [117]	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

Table 3. Analysis without training vs our experiment results from Imagenet-1K. Note that a naive prediction would correlate the given top-1 accuracy scores directly with the number of parameters. Our 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	7.1 M	224
V2-S-shared	**78.3**	**26.33**	**26.97**	1505.547	9.8	**4.4 M**	384
V2-S	83.2	28.72	29.28	1505.546	9.8	21.6 M	384

Table 4. Analysis without training vs some of the top-1 accuracy scores for larger models [36] for Imagenet-1K. Note that a naive prediction would correlate the given top-1 accuracy scores directly with the number of parameters. Our 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	83.6+	28.72	29.28	1505.5	9.8	21.6 M	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

Overall, at the cost of additional FLOPs, we observed improved parameter efficiency for all three benchmarks. Alongside the research regarding parameter sharing in the literature discussed in section 2.1, and combined with our graph analysis results in table 3, these results provide supporting evidence for the reusability prior.

5.7. Predictions from graph analysis vs experiment results

In our experiments with EfficientNetv2 models, V2-S-shared models consistently performed better than the V2-B0 models, despite having a significantly lower number of parameters. In table 3, we compared the results from our Imagenet-1K experiments with the results from our 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, our graph analysis assigned a higher score for the *V2-S-shared* model.

5.8. Graph analysis results for larger models

We compared our graph analysis based quantities for larger models that were trained by Tan *et al* in [36].[6] In table 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 1/3rd of the actual top-1 accuracies, but unlike the top-1 accuracy, our performance estimation scores are not bounded (e.g. P_G can be negative).

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

6. Limitations and future work

The reusability prior we defined encompasses design, training, and data aspects in deep learning. In this work, however, we majorly focused on the design aspect. We ignored *concrete* contexts that can arise due to the concrete samples from the training data itself. When counting the number of contexts, we only considered what can arise due to the computational graph of the model as well as the input size. 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 our definition of context.

For larger models, we omitted additional experiments with cross-layer parameter sharing. From the perspective of reusability prior, all models we analyzed already explicitly (e.g. 1×1 convolutions) or implicitly (e.g. skip connections) share parameters. Therefore, we instead provided a comparison of existing scores from [36] combined with predictions from our graph-based analysis in table 4.

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 performance estimation approaches we introduced do not necessarily optimize for compute resources. For the same number of parameters, both approaches are biased towards deeper and narrower models, since our 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 we introduced, then *without any training*, searching viable model designs by solely relying on graph analysis can be an interesting direction.

7. Conclusion

Not all performance can be explained directly with the number of parameters and the training data. We introduced the reusability prior to point towards a deeper reason. We first demonstrated that, either explicitly or implicitly, all mainstream deep learning models reuse parameters. We then introduced a generalized notion of reusability that encompasses 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, we defined quantities namely entropy, expected spread, and total surprisal which rely on analyzing the computational graph of a model. We gave formal proof 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 our approach in practice, we proposed two crude performance estimation approaches based on total surprisal and expected spread. We then compared EfficientNetv2 models 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. We demonstrated a counter-example where EfficientNetv2-S models with parameter sharing outperformed the baseline EfficientNetv2-b0 models which have at least 60% more parameters. We gave another edge case with ResNet-50 where despite having significantly more parameters, it underperformed compared to EfficientNetv2-B3 and V2-S models. Our 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, our 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 3 and 4). In practice, as discussed in section 6, the reusability prior and our 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 our graph-based approach for estimating performance is generalizable to more models. Yet, for the models we investigated, our approach was able to correctly delineate at least two important edge cases in tables 3 and 4. Furthermore, the quantities we introduced 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 we only relied on the model graphs *without any training*.

We conclude that 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.

Data availability statement

The data that support the findings of this study are available upon reasonable request from the authors.

Acknowledgments

The numerical calculations reported in this paper were partially performed at TUBITAK ULAKBIM, High Performance and Grid Computing Center (TRUBA resources). We appreciate the GPU resources that were allocated for a portion of this research.

We thank Ugur Halıcı, Emre Akbas, Selim Temizer, Kasım Öztoprak, the anonymous reviewers for their constructive input, and Georgina Romo Olivares for her support.

Conflicts of interest

The authors declare no conflict of interest.

Appendix A. Horizontal unrolling algorithm

For illustrative purposes, we provide an algorithm for the horizontal unrolling approach we described. Algorithm 1 takes the original DAG input, creates a new DAG with a root node, and calls *unrollNode* for each leaf node. In *unrollNode*, 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 algorithm 1, 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 $O(\text{width}^{\text{depth}})$. We instead use an optimized counting approach. Please see our Github repo for additional details: https://github.com/gozepolat/priors/tree/main/reusability.

Algorithm 1. Horizontal Unrolling.

```
Function unrollNode(node, root) is
    duplicate : = Node(node.name)
    for source in node.sources do
        if isRoot(source) then
            root.addTarget(duplicate)
            continue
        end
        unrollNode(source).addTarget(duplicate)
    end
end
Function horizontalUnroll(graph) is
    unrolled : = DAG('unrolled')
    leafNodes : = getLeafNodes(graph)
    for node in leafNodes do
        unrollNode(node, unrolled)
    end
    return unrolled
end
```

Appendix B. An illustration of how we estimate model performances

Step by step, we analyze the graph depicted in figure 3(a) as an example. For large models, we use an optimized algorithm but for the sake of simplicity, we rely on algorithm 1 in this example.

(a) Horizontally unroll the graph using algorithm 1. This results in the graph in figure 3(b).
(b) For each learnable parameter directly count the repetitions in the unrolled graph, i.e. collect the frequencies: $w_1 = 2, w_2 = 2, w_3 = 2, w_4 = 2, w_5 = 1, w_6 = 1, w_7 = 1, w_8 = 1$
(c) 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.
(d) **Total surprisal:** $-\log_2(2/12) - \log_2(2/12) - \log_2(2/12) - \log_2(2/12) - \log_2(1/12) - \log_2(1/12) - \log_2(1/12) - \log_2(1/12) = -4 \times (\log_2(2/12) + \log_2(1/12)) = 24.68$
(e) **Entropy:** $-4 \times (2/12 \times \log_2(2/12) + 1/12 \times \log_2(1/12)) = 2.92$
(f) **Expected spread:** $4 \times (2/12 \times \log_2(2) + 1/12 \times \log_2(1)) = 0.67$
(g) **Total surprisal based performance estimation:** input nodes $N_I = 2$ and model size $|G| = 2 + 1 + 8 = 11$ and total surprisal $= 24.68$ so $P_G = \log2(24.68 \times 2/11) = 2.17$.
(h) **Expected spread based performance estimation:** there are 8 learnable parameters and the expected spread $= 0.67$ so $P'_G = \log2((0.67 + 1) \times 8 \times 2/11) = 1.28$.

In our Github repository, we share multiple examples of how we derive the model-level quantities including the graphs from figures 2(a), (b) and 3(a), (c). The examples are available here: https://github.com/gozepolat/priors/tree/main/reusability.

ORCID iDs

Aydın Göze Polat https://orcid.org/0000-0002-0853-5750
Ferda Nur Alpaslan https://orcid.org/0000-0002-9806-1543

References

[1] Kolesnikov A, Beyer L, Zhai X, Puigcerver J, Yung J, Gelly S and Houlsby N 2020 Big transfer (BiT): general visual representation learning *ECCV*
[2] Jia C, Yang Y, Xia Y, Chen Y T, Parekh Z, Pham H, Le Q V, Sung Y H, Li Z and Duerig T 2021 Scaling up visual and vision-language representation learning with noisy text supervision *ICML*
[3] Bao H, Dong L and Wei F 2021 BEiT: BERT pre-training of image transformers (arXiv:2106.08254)
[4] He K, Chen X, Xie S, Li Y, Doll'ar P and Girshick R B 2021 Masked autoencoders are scalable vision learners (arXiv:2111.06377)
[5] Brock A, De S, Smith S L and Simonyan K 2021 High-performance large-scale image recognition without normalization (arXiv:2102.06171)
[6] Yuan L *et al* 2021 Florence: a new foundation model for computer vision (arXiv:2111.11432)
[7] Ding M, Xiao B, Codella N C F, Luo P, Wang J and Yuan L 2022 DaViT: dual attention vision transformers (arXiv:2204.03645)

[8] Riquelme C, Puigcerver J, Mustafa B, Neumann M, Jenatton R, Pinto A S, Keysers D and Houlsby N 2021 Scaling vision with sparse mixture of experts *NeurIPS*
[9] Zhai X, Kolesnikov A, Houlsby N and Beyer L 2021 Scaling vision transformers (arXiv:2106.04560)
[10] Yu J, Wang Z, Vasudevan V, Yeung L, Seyedhosseini M and Wu Y 2022 CoCa: contrastive captioners are image-text foundation models (arXiv:2205.01917)
[11] Wortsman M *et al* 2022 Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time (arXiv:2203.05482)
[12] Dai Z, Liu H, Le Q V and Tan M 2021 CoAtNet: marrying convolution and attention for all data sizes (arXiv:2106.04803)
[13] Narayanan D *et al* 2021 *SC21: Int. Conf. for High Performance Computing, Networking, Storage and Analysis* pp 1–14
[14] Brown T *et al* 2020 Language models are few-shot learners *Advances in Neural Information Processing Systems* vol 33, ed H Larochelle M Ranzato, R Hadsell, M Balcan and H Lin (Curran Associates, Inc.) pp 1877–901
[15] Fedus W, Zoph B and Shazeer N 2022 *J. Mach. Learn. Res.* **23** 1–39
[16] Du N, Huang Y *et al* 2022 GLaM: efficient scaling of language models with mixture-of-experts *Proc. 39th Int. Conf. on Machine Learning (Proc. of Machine Learning Research)* vol 162 ed K Chaudhuri, S Jegelka, L Song, C Szepesvari, G Niu and S Sabato (PMLR) pp 5547–69
[17] Zou X, Yin D, Zhong Q, Yang H, Yang Z and Tang J 2021 Controllable generation from pre-trained language models via inverse prompting *Proc. 27th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining (KDD' 21)* (New York: Association for Computing Machinery) pp 2450–60
[18] Hestness J, Narang S, Ardalani N, Diamos G F, Jun H, Kianinejad H, Patwary M M A, Yang Y and Zhou Y 2017 Deep learning scaling is predictable, empirically (arXiv:1712.00409)
[19] Hoffmann J *et al* 2022 Training compute-optimal large language models (arXiv:2203.15556)
[20] Fukushima K and Miyake S 1982 Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition *Competition and Cooperation in Neural Nets* (Berlin: Springer) pp 267–85
[21] LeCun Y, Bottou L, Bengio Y and Haffner P 1998 *Proc. IEEE* **86** 2278–324
[22] Rumelhart D E, Hinton G E and Williams R J 1986 *Nature* **323** 533–6
[23] Schmidhuber J 1993 *Page 150 ff demonstrates credit assignment across the equivalent of 1,200 layers in an unfolded RNN*
[24] Gers F A, Schmidhuber J and Cummins F 1999 Learning to forget: continual prediction with LSTM *IET* **2** 850–5
[25] Pascanu R, Gulcehre C, Cho K and Bengio Y 2014 How to construct deep recurrent neural networks *Proc. 2nd Int. Conf. on Learning Representations (ICLR 2014)*
[26] Chung J, Gulcehre C, Cho K and Bengio Y 2014 Empirical evaluation of gated recurrent neural networks on sequence modeling *NIPS 2014 Workshop on Deep Learning December 2014*
[27] Katharopoulos A, Vyas A, Pappas N and Fleuret F 2020 Transformers are rnns: fast autoregressive transformers with linear attention *Proc. 37th Int. Conf. on Machine Learning ICML'20* (JMLR.org)
[28] Yang Z, Dai Z, Yang Y, Carbonell J G, Salakhutdinov R and Le Q V 2019 XLNet: generalized autoregressive pretraining for language understanding *NeurIPS*
[29] van den Oord A, Dieleman S, Zen H, Simonyan K, Vinyals O, Graves A, Kalchbrenner N, Senior A W and Kavukcuoglu K 2016 WaveNet: a generative model for raw audio *SSW*
[30] Boulch A 2018 *Pattern Recognit. Lett.* **103** 53–59
[31] Savarese P and Maire M 2019 Learning implicitly recurrent CNNs through parameter sharing *Int. Conf. on Learning Representations*
[32] Wang Z, Cheng X, Sapiro G and Qiu Q 2020 ACDC: weight sharing in atom-coefficient decomposed convolution (arXiv:2009.02386)
[33] Lan Z, Chen M, Goodman S, Gimpel K, Sharma P and Soricut R 2020 Albert: a lite BERT for self-supervised learning of language representations *8th Int. Conf. on Learning Representations (ICLR 2020) (Addis Ababa, Ethiopia, April 26–30 2020)* (OpenReview.net)
[34] Xiao T, Li Y, Zhu J, Yu Z and Liu T 2019 Sharing attention weights for fast transformer *Proc. 28th Int. Joint Conf. on Artificial Intelligence (IJCAI-19) (Int. Joint Conferences on Artificial Intelligence Organization)* pp 5292–8
[35] Reid M, Marrese-Taylor E and Matsuo Y 2021 Subformer: exploring weight sharing for parameter efficiency in generative transformers *Findings of the Association for Computational Linguistics: Emnlp 2021* (Punta Cana, Dominican Republic: Association for Computational Linguistics) pp 4081–90
[36] Tan M and Le Q 2021 Efficientnetv2: smaller models and faster training *Int. Conf. on Machine Learning* (PMLR) pp 10096–106
[37] Krizhevsky A, Nair V and Hinton G 2014 (available at: http://www.cs.toronto.edu/kriz/cifar.html)
[38] Krizhevsky A, Hinton G *et al* 2009 Learning multiple layers of features from tiny images
[39] Deng J, Dong W, Socher R, Li L J, Li K and Fei-Fei L 2009 Imagenet: a large-scale hierarchical image database *IEEE Conf. on Computer Vision and Pattern Recognition 2009 (CVPR 2009)* (IEEE) pp 248–55
[40] Hochreiter S and Schmidhuber J 1997 *Neural Comput.* **9** 1735–80
[41] Tran D, Bourdev L, Fergus R, Torresani L and Paluri M 2015 Learning spatiotemporal features with 3D convolutional networks *2015 IEEE Int. Conf. on Computer Vision (ICCV)* (IEEE) pp 4489–97
[42] Simonyan K and Zisserman A 2015 Very deep convolutional networks for large-scale image recognition *3rd Int. Conf. on Learning Representations (ICLR 2015) (San Diego, CA, 7–9 May 2015) (Conf. Track Proc.)* ed Y Bengio and Y LeCun
[43] Krizhevsky A, Sutskever I and Hinton G E 2012 ImageNet Classification with deep convolutional neural networks *Proc. 25th Int. Conf. on Neural Information Processing Systems NIPS'12* (*Redhook, NY*: Curran Associates Inc.) pp 1097–105
[44] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V and Rabinovich A 2015 Going deeper with convolutions *Proc. IEEE Conf. on Computer Vision and Pattern Recognition* pp 1–9
[45] Canziani A, Paszke A and Culurciello E 2016 An analysis of deep neural network models for practical applications (arXiv:1605.07678)
[46] Singh S, Hoiem D and Forsyth D 2016 Swapout: learning an ensemble of deep architectures *Advances in Neural Information Processing Systems* pp 28–36
[47] Wang J, Yang Y, Mao J, Huang Z, Huang C and Xu W 2016 Cnn-rnn: a unified framework for multi-label image classification *Proc. IEEE Conf. on Computer Vision and Pattern Recognition* pp 2285–94
[48] Pham H, Guan M Y, Zoph B, Le Q V and Dean J 2018 Efficient neural architecture search via parameter sharing *ICML*
[49] Pham H, Guan M, Zoph B, Le Q and Dean J 2018 Efficient neural architecture search via parameters sharing *Proc. 35th Int. Conf. on Machine Learning (Proc. of Machine Learning Research)* vol 80 ed J Dy and A Krause (PMLR) pp 4095–104

[50] Golovin D, Solnik B, Moitra S, Kochanski G, Karro J and Sculley D 2017 Google vizier: a service for black-box optimization *Proc. 23rd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining* (ACM) pp 1487–95
[51] Chen Y, Hoffman M W, Colmenarejo S G, Denil M, Lillicrap T P and de Freitas N 2016 Learning to learn for global optimization of black box functions (arXiv:1611.03824)
[52] Negrinho R and Gordon G J 2017 DeepArchitect: automatically designing and training deep architectures (arXiv:1704.08792)
[53] Bello I, Zoph B, Vasudevan V and Le Q V 2017 Neural optimizer search with reinforcement learning *Proc. 34th Int. Conf. on Machine Learning, (ICML 2017) (Sydney, NSW, Australia, 6–11 August 2017) (Proc. of Machine Learning Research)* vol 70 ed D Precup and Y W Teh (PMLR) pp 459–68
[54] Garg V K and Kalai A T 2016 Meta-unsupervised-learning: a supervised approach to unsupervised learning (arXiv:1612.09030)
[55] Andrychowicz M, Denil M, Gomez S, Hoffman M W, Pfau D, Schaul T and de Freitas N 2016 Learning to learn by gradient descent by gradient descent *Advances in Neural Information Processing Systems* pp 3981–9
[56] Lan G, de Vries L and Wang S 2019 Evolving efficient deep neural networks for real-time object recognition *IEEE Symp. Series on Computational Intelligence (SSCI 2019) (Xiamen, China, 6–9 December 2019)* (IEEE) pp 2571–8
[57] He K, Zhang X, Ren S and Sun J 2016 Deep residual learning for image recognition *Proc. IEEE Conf. on Computer Vision and Pattern Recognition* vol 2016-Decem pp 770–8
[58] Huang G, Liu Z, van der Maaten L and Weinberger K Q 2017 Densely connected convolutional networks *2017 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2017) (Honolulu, HI, USA, 21-26 July 2017)* (IEEE Computer Society) pp 2261–9
[59] Zhang K, Sun M, Han T X, Yuan X, Guo L, Liu T, Han X, Yuan X, Guo L and Liu T 2017 *IEEE Trans. Circuits Syst. Video Technol.* **28** 1303–14
[60] Wu Z, Nagarajan T, Kumar A, Rennie S, Davis L S, Grauman K and Feris R S 2018 BlockDrop: dynamic inference paths in residual networks *2018 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2018) (Salt Lake City, UT, USA, 18–22 June 2018)* (Computer Vision Foundation/IEEE Computer Society) pp 8817–26
[61] Zhou A, Knowles T and Finn C 2021 Meta-learning symmetries by reparameterization *9th Int. Conf. on Learning Representations (ICLR 2021) (Virtual Event, Austria, 3–7 May 2021)* (OpenReview.net)
[62] Dehmamy N, Walters R, Liu Y, Wang D and Yu R 2021 *Advances in Neural Information Processing Systems* vol 34
[63] de Haan P, Cohen T S and Welling M 2020 *Advances in Neural Information Processing Systems* vol 33 pp 3636–46
[64] Cohen T, Weiler M, Kicanaoglu B and Welling M 2019 Gauge equivariant convolutional networks and the icosahedral CNN *Int. Conf. on Machine Learning* (PMLR) pp 1321–30
[65] Tai K S, Bailis P and Valiant G 2019 Equivariant transformer networks *Int. Conf. on Machine Learning* (PMLR) pp 6086–95
[66] Yeh R A, Hu Y T, Hasegawa-Johnson M and Schwing A 2022 Equivariance discovery by learned parameter-sharing *Int. Conf. on Artificial Intelligence and Statistics* (PMLR) pp 1527–45
[67] Louizos C, Ullrich K and Welling M 2017 Bayesian compression for deep learning *Advances in Neural Information Processing Systems* pp 3290–300
[68] Ullrich K, Meeds E and Welling M 2017 Soft weight-sharing for neural network compression *5th Int. Conf. on Learning Representations (ICLR 2017) (Toulon, France, 24-26 April 2017) (Conf. Track Proc.)* (OpenReview.net)
[69] Plummer B A, Dryden N, Frost J, Hoefler T and Saenko K 2022 Neural parameter allocation search *The Tenth Int. Conf. on Learning Representations (ICLR 2022) (Virtual Event, 25–29 April 2022)* (OpenReview.net)
[70] Tang C, Zhao Y, Wang G, Luo C, Xie W and Zeng W 2022 Sparse MLP for image recognition: is self-attention really necessary? *36th AAAI Conf. on Artificial Intelligence (AAAI 2022) 34th Conf. on Innovative Applications of Artificial Intelligence (IAAI 2022) 12th Symp. on Educational Advances in Artificial Intelligence (EAAI 2022) (Virtual Event, 22 February–1 March 2022)* (AAAI Press) pp 2344–51
[71] Malach E, Yehudai G, Shalev-Schwartz S and Shamir O 2020 Proving the lottery ticket hypothesis: pruning is all you need *Int. Conf. on Machine Learning* (PMLR) pp 6682–91
[72] Frankle J and Carbin M 2019 The lottery ticket hypothesis: finding sparse, trainable neural networks *7th Int. Conf. on Learning Representations (ICLR 2019) (New Orleans, LA, USA, 6–9 May 2019)* (OpenReview.net)
[73] Ma J, Zhao Z, Chen J, Li A, Hong L and Chi E H 2019 SNR: sub-network routing for flexible parameter sharing in multi-task learning *Proc. AAAI Conf. on Artificial Intelligence* vol 33 pp 216–23
[74] Tan J H, Tan Y H, Chan C S and Chuah J H 2022 *Neurocomputing* **482** 60–72
[75] Murdock C, Li Z, Zhou H and Duerig T 2016 Blockout: dynamic model selection for hierarchical deep networks *CVPR 2016*
[76] Yosinski J, Clune J, Bengio Y and Lipson H 2014 How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems* pp 3320–8
[77] Saxena S and Verbeek J 2016 Convolutional neural fabrics *Advances in Neural Information Processing Systems* pp 4053–61
[78] Fernando C, Banarse D, Blundell C, Zwols Y, Ha D, Rusu A A, Pritzel A and Wierstra D 2017 Pathnet: evolution channels gradient descent in super neural networks (arXiv:1701.08734)
[79] Wu L Y, Fisch A, Chopra S, Adams K, Bordes A and Weston J 2018 Starspace: embed all the things! *Proc. 32nd AAAI Conf. on Artificial Intelligence, (AAAI-18), the 30th Innovative Applications of Artificial Intelligence (IAAI-18) and the 8th AAAI Symp. on Educational Advances in Artificial Intelligence (EAAI-18) (New Orleans, Louisiana, USA, 2–7 February 2018)* ed S A McIlraith and K Q Weinberger (AAAI Press) pp 5569–77
[80] Finn C, Abbeel P and Levine S 2017 Model-agnostic meta-learning for fast adaptation of deep networks *Proc. 34th Int. Conf. on Machine Learning (ICML 2017) (Sydney, NSW, Australia, 6-11 August 2017) (Proc. of Machine Learning Research)* vol 70 ed D Precup and Y W Teh (PMLR) pp 1126–35
[81] Liu H, Simonyan K and Yang Y 2019 DARTS: differentiable architecture search *7th Int. Conf. on Learning Representations (ICLR 2019) (New Orleans, LA, USA, 6-9 May 2019)* (OpenReview.net)
[82] Chollet F 2017 Xception: deep learning with depthwise separable convolutions *2017 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2017) (Honolulu, HI, USA, 21–26 July 2017)* (IEEE Computer Society) pp 1800–7
[83] Sandler M, Howard A G, Zhu M, Zhmoginov A and Chen L 2018 Mobilenetv2: inverted residuals and linear bottlenecks *2018 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2018) (Salt Lake City, UT, USA, 18–22 June 2018)* (Computer Vision Foundation/IEEE Computer Society) pp 4510–20
[84] Howard A *et al* 2019 Searching for mobilenetv3 *2019 IEEE/CVF Int. Conf. on Computer Vision, (ICCV 2019) (Seoul, Korea (South), 27 October–2 November 2019)* (IEEE) pp 1314–24
[85] Nowlan S J and Hinton G E 1992 *Neural Comput.* **4** 473–93
[86] Hinton G, Vinyals O and Dean J 2015 Distilling the knowledge in a neural network (arXiv:1503.02531)

[87] Buciluǎ C, Caruana R and Niculescu-Mizil A 2006 Model compression *Proc. 12th ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining* pp 535–41
[88] Han S, Mao H and Dally W J 2016 Deep compression: compressing deep neural network with pruning, trained quantization and huffman coding *4th Int. Conf. on Learning Representations (ICLR 2016) (San Juan, Puerto Rico, 2–4 May 2016) (Conf. Track Proc.)* ed Y Bengio and Y LeCun
[89] Peer D, Keulen B, Stabinger S, Piater J and Rodriguez-sanchez A 2022 Improving the trainability of deep neural networks through layerwise batch-entropy regularization (arXiv:2208.01134)
[90] Wickstrøm K, Løkse S, Kampffmeyer M, Yu S, Príncipe J C and Jenssen R 2019 Information plane analysis of deep neural networks via matrix-based Renyi'ls entropy and tensor kernels (arXiv:1909.11396)
[91] Levine Y, Wies N, Sharir O, Bata H and Shashua A 2020 Limits to depth efficiencies of self-attention *Advances in Neural Information Processing Systems 33: Annual Conf. on Neural Information Processing Systems 2020* (6–12 December 2020) ed H Larochelle, M Ranzato, R Hadsell, M Balcan and H Lin
[92] Bu K, Zhang Y and Luo Q 2020 Depth-width trade-offs for neural networks via topological entropy (arXiv:2010.07587)
[93] Shannon C E 1948 *Bell Syst. Tech. J.* **27** 379–423
[94] Zagoruyko S and Komodakis N 2016 Wide residual networks *Proc. British Machine Vision Conf. 2016* (*York, UK,* 19–22 September 2016) ed R C Wilson, E R Hancock and W A P Smith (BMVA Press)
[95] Bagherinezhad H, Rastegari M and Farhadi A 2017 *2017 IEEE Conf. on Computer Vision and Pattern Recognition* (CVPR) pp 860–9
[96] Qiu Q, Cheng X, Calderbank A R and Sapiro G 2018 Dcfnet: deep neural network with decomposed convolutional filters *Proc. 35th Int. Conf. on Machine Learning (ICML)* StockholmsmäSsan, Stockholm (*10–15 July 2018*) vol 80 ed J G Dy and A Krause (PMLR) pp 4195–204
[97] Devlin J, Chang M, Lee K and Toutanova K 2019 BERT: pre-training of deep bidirectional transformers for language understanding *Proc. 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2019) (Minneapolis, MN, 2–7 June 2019) (Volume 1 Long and Short Papers)* ed J Burstein, C Doran and T Solorio (Association for Computational Linguistics) pp 4171–86
[98] Yang S, Hou L, Song X, Liu Q and Zhou D 2021 Speeding up deep model training by sharing weights and then unsharing (arXiv:2110.03848)
[99] Chen R T Q, Rubanova Y, Bettencourt J and Duvenaud D K 2018 *Advances in Neural Information Processing Systems* vol 31
[100] Dupont E, Doucet A and Teh Y W 2019 *Advances in Neural Information Processing Systems* vol 32
[101] Bai S, Kolter J Z and Koltun V 2019 *Advances in Neural Information Processing Systems* vol 32
[102] Bai S, Koltun V and Kolter J Z 2020 *Advances in Neural Information Processing Systems* vol 33 pp 5238–50
[103] DeVries T and Taylor G W 2017 Improved regularization of convolutional neural networks with cutout (arXiv:1708.04552)
[104] Zhang H, Cisse M, Dauphin Y N and Lopez-Paz D 2017 mixup: beyond empirical risk minimization (arXiv:1710.09412)
[105] Yun S, Han D, Oh S J, Chun S, Choe J and Yoo Y 2019 Cutmix: regularization strategy to train strong classifiers with localizable features *Proc. IEEE/CVF Int. Conf. on Computer Vision* pp 6023–32
[106] Srivastava N, Hinton G, Krizhevsky A, Sutskever I and Salakhutdinov R 2014 *J. Mach. Learn. Res.* **15** 1929–58
[107] Huang G, Sun Y, Liu Z, Sedra D and Weinberger K Q 2016 Deep networks with stochastic depth *European Conf. on Computer Vision* (Berlin: Springer) pp 646–61
[108] LeCun Y, Boser B, Denker J S, Henderson D, Howard R E, Hubbard W and Jackel L D 1989 *Neural Comput.* **1** 541–51
[109] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A N, Kaiser L and Polosukhin I 2017 Attention is all you need *Advances in Neural Information Processing Systems 30: Annual Conf. on Neural Information Processing Systems 2017 (Long Beach, CA, USA, 4–9 December 2017)* ed I Guyon, U von Luxburg, S Bengio, H M Wallach, R Fergus, S V N Vishwanathan and R Garnett pp 5998–6008
[110] Kullback S and Leibler R A 1951 *Ann. Math. Stat.* **22** 79–86
[111] Abadi M *et al* 2015 Tensorflow: large-scale machine learning on heterogeneous systems software available from tensorflow.org (available at: http://tensorflow.org/)
[112] Van Rossum G and Drake F L 2009 *Python 3 Reference Manual* (Scotts Valley, CA: CreateSpace)
[113] Polat A 2023 gozepolat/priors: the reusability prior (https://doi.org/10.5281/zenodo.7805346)
[114] Google 2022 Efficientnetv2 official implementation (available at: https://github.com/google/automl/tree/master/efficientnetv2)
[115] Russakovsky O *et al* 2015 *Int. J. Comput. Vis.* **115** 211–52
[116] Cubuk E D, Zoph B, Mané D, Vasudevan V and Le Q V 2019 Autoaugment: learning augmentation strategies from data *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2019) (Long Beach, CA, USA, 16–20 June 2019)* (Computer Vision Foundation/IEEE) pp 113–23
[117] Cubuk E D, Zoph B, Shlens J and Le Q V 2019 *2020 IEEE/CVF Conf. on Computer Vision and Pattern Recognition Workshops (CVPRW)* pp 3008–17