Artificial Intelligence Review (2024) 57:349
https://doi.org/10.1007/s10462-024-10943-8

A framework for measuring the training efficiency of a neural architecture

Eduardo Cueto-Mendoza[1,2] · **John Kelleher**[2]

Accepted: 10 September 2024 / Published online: 28 October 2024

Abstract
Measuring Efficiency in neural network system development is an open research problem. This paper presents an experimental framework to measure the training efficiency of a neural architecture. To demonstrate our approach, we analyze the training efficiency of Convolutional Neural Networks and Bayesian equivalents on the MNIST and CIFAR-10 tasks. Our results show that training efficiency decays as training progresses and varies across different stopping criteria for a given neural model and learning task. We also find a non-linear relationship between training stopping criteria, training Efficiency, model size, and training Efficiency. Furthermore, we illustrate the potential confounding effects of overtraining on measuring the training efficiency of a neural architecture. Regarding relative training efficiency across different architectures, our results indicate that CNNs are more efficient than BCNNs on both datasets. More generally, as a learning task becomes more complex, the relative difference in training efficiency between different architectures becomes more pronounced.

Keywords Deep learning · Efficiency · Deep neural networks · Hyperparameters

Eduardo Cueto-Mendoza and John Kelleher contributed equally to this work.

✉ Eduardo Cueto-Mendoza
eduardo.cuetomendoza@tudublin.ie

John Kelleher
john.kelleher@tcd.ie

[1] School of Computer Science, TU Dublin, Grangegorman, Dublin 7 D07H6K8, Co. Dublin, Ireland

[2] ADAPT Research Centre, School of Computer Science and Statistics, Trinity College Dublin, Dublin 2, Co. Dublin, Ireland

1 Introduction

Artificial Intelligence is predicted to be a critical enabling technology for many of the 17 Sustainable Development Goals (SDGs). However, its current dependency on massive datasets and computer power means that it will also inhibit the attainment of some SDGs, particularly SDG7 (Affordable and Clean Energy) and SDG 13 (Climate action) (Vinuesa et al. 2020). Modern Artificial Intelligence (AI) uses data-driven methods like deep learning. It is primarily driven by trends of ever larger datasets, larger models, and more powerful computers with the sole concern of improving model accuracy (Kelleher 2019). This dynamic resulted in a 300,000x increase between 2012 and 2018 in the computation required to train a competitive DL model [3] (this trend far exceeds Moore's Law). Indeed, it has recently been estimated that training one AI model generated the CO2 emissions equivalent to driving 700,000 km (DeWeerdt 2020).

The environmental challenge posed by AI's growing energy needs and associated carbon emissions has been recognized in recent years. For example, researchers in AI Ethics have highlighted this challenge (Bender et al. 2021) and have called for more research on "sustainable methods of AI" (van Wynsberghe 2021). In response to these calls, there is a growing trend within AI research to move beyond systems evaluations solely based on accuracy. Recent research tends to report hardware details and training time alongside accuracy, and some papers report FLOPS. However, time and FLOPS are not sufficient to characterize Efficiency. There is a growing body of work (e.g., Schwartz et al. 2020; Strubell et al. 2020; Li et al. 2020; Sze et al. 2020; Li and John 2003) that shows that more data is required to understand the energy and resource trade-off of deep neural networks. Consequently, a critical step in developing sustainable AI is the development of measures for Efficiency that can be integrated into the development process of an AI system.

This paper directly addresses the need for a measure to characterize the Efficiency of a neural network architecture on a specific hardware and learning task. A natural efficiency ratio of interest for a neural architecture is the ratio between the accuracy of a neural model and the energy consumed to achieve this accuracy. Accuracy is usually measured using an appropriate measure for the task and dataset distribution (e.g., F1, AUC-ROC, etc.). However, several recent results highlight a non-linear relationship between the accuracy of a neural model and the size of the model (Nakkiran et al. 2021). This suggests that there is likely a non-linear relationship between the training efficiency of an architecture and the size of the model instantiating the architecture. At the same time, there is a gap in the research literature in terms of how the training efficiency of a neural architecture varies across training. Understanding the dynamics of training efficiency is crucial as it informs decisions relating to the stopping criterion for training. Consequently, in this work, we set out an experimental methodology for comparing the relative Efficiency of different neural architectures in terms of their efficiency dynamics as training progresses and the changes in Efficiency as the size of the models instantiating the architectures vary. This experimental methodology includes both a measure of Efficiency and an experimental framework for capturing the necessary data for the efficiency measure.

In order to test and demonstrate the usefulness of our efficiency measure, we use our experimental framework to analyze the relative Efficiency of two different neural architectures, a CNN network (LeNet) and a Bayesian Convolutional Network (BCNN), on the MNIST and CIFAR-10 tasks. BCNNs are an interesting case study because, although they

have produced better results than LeNet on MNIST and CIFAR-10 (Gal and Ghahramani 2015), their Efficiency relative to standard frequentist networks has yet to be assessed. Furthermore, given that the outcomes obtained by training a frequentist LeNet architecture with backpropagation and its BCNN counterpart trained using approximate variational inference—implemented via dropout—are very different (training the frequentist LeNet results in a point estimate in parameter space, whereas training the BCNN returns a probability distribution over a parameter space), it is likely that there will be differences in terms of Efficiency between these two architectures.

In summary, the key contributions of this research are: (1) we propose a measure of the training efficiency of a neural architecture on a given task; (2) we present a case study analyzing the efficiency dynamics of CNNs and BCNNs on multiple tasks across training; and (3) we analyze the overall Efficiency of CNN versus BCNN architectures. Our results indicate that CNNs are more efficient than BCNNs for training. Also, the Efficiency of both architectures varies across training. For both architectures, there is a non-linear relationship between training efficiency stopping criteria and between training efficiency and model size. Furthermore, we highlight and illustrate the confounding effect that overtraining can have on measuring the Efficiency of a neural architecture. Finally, as the learning task becomes more complex, the relative difference in training efficiency between different architectures becomes more pronounced.

2 Related work

Research on Efficiency in AI can broadly be categorized into four research streams: architectures, compression, training regimes, and metrics. The first of these streams focuses on developing more computationally efficient neural architectures. For example, improving the Efficiency of the attention mechanism in transformer models (Vaswani et al. 2017) has frequently been a target for this type of research. This is due to the popularity of transformer models and the high complexity in time and space $O(n^2)$—of the standard attention mechanism. Within this category of work, the Reformer (Kitaev et al. 2020) proposes an efficiency improvement (in terms of computation and memory) to the standard transformer that replaces the regular dot-product attention mechanism with one that uses locality-sensitive hashing, and the Linformer Wang et al. (2020) replaces the transformer attention mechanism and approximates it by a low-rank matrix which reduces the complexity of the attention layer to $O(n)$. A recent survey of work on improving efficiency in transformers is presented in Tay et al. (2020). Also, although research on neural architecture search has traditionally focused on optimizing for a single objective (such as accuracy), recently, there has been a growing interest in multi-objective neural architecture search which considers Efficiency (frequently hardware efficiency to enable edge deployment) as part of the optimization problem (see e.g., Zeng et al. 2020; White et al. 2023; Chen et al. 2023; Lu et al. 2024).

A second stream of research has focused on improving Efficiency by reducing model size. Some of this work trades extra computation during initial model training for smaller, more efficient models at inference. For example, the EfficientNet (Tan and Le 2019) and EfficientNet v2 (Tan and Le 2021) papers propose model scaling methods that seek to maximize model efficiency during inference (by attempting to minimize the final model depth, width, and resolution) while preserving accuracy at the cost of extra computation during

training. Similarly, the training methodology proposed in Cai et al. (2019) uses pruning during training to reduce model depth, width, kernel size, and resolution. Another example of this type of work is the Lottery Ticket Hypothesis (Frankle and Carbin 2018) methodology, which focuses on finding small subnetworks that can fit into different hardware platforms and generalize better. Some research focused on reducing model size is designed to work on pre-trained models. For example, NetAdapt uses empirical measures to reduce several hyperparameters in order to fulfill a certain resource budget (Yang et al. 2018), and DistilBERT uses model distillation techniques to generate smaller models from a complete BERT transformer (Sanh et al. 2019). Zhou and Quan (2023) provides a recent review of work on compressing deep neural networks that cover the four main approaches found in the literature (pruning, quantization, factorization, and distillation) and conclude that optimization approaches that combine these different compression approaches are an emerging area of research.

The third stream of research focuses on improving the training regime's Efficiency. Work in this stream generally focuses on modifying one or more of the following components of the training regime: the ordering of (i.e., curriculum learning) or the selection of the training data presented to the model (Jiang et al. 2019; Mindermann et al. 2022; Xie et al. 2023; Wang et al. 2023; Yang et al. 2023; Wang et al. 2024); dynamically modifying the architecture of the model as part of the training process (Gong et al. 2019; Zhang and He 2020; Pan et al. 2023; Ding et al. 2023); modifying the objective function (Anil et al. 2020; Goldfarb et al. 2020; Eschenhagen et al. 2023); and improving the optimization algorithm (Liu et al. 2023; Chen et al. 2023).[1] (Kaddour et al. 2023) reports a recent empirical study of the effectiveness of several of these efficient training approaches against a baseline training regime that used the Adam optimizer with a fully decayed learning rate. These experiments used a fixed computation budget based on wall time (calculated by multiplying the number of iterations of training by the time per iteration for that architecture and training regime on a reference hardware system) as the criterion for stopping training. Three budgets were used for each experiment: 6 hours, 12 hours, and 24 hours. The results indicate that the tested training modifications did not statistically outperform the baseline in most experiments. When they did, this improvement was reduced as the computing budget increased.

The fourth stream of research is focused on developing measures and methodologies for assessing the performance or Efficiency of an AI solution for a given problem. One focus within this stream of research has been on hardware efficiency, see, e.g., Davis et al. (2009); Sze et al. (2020). Another focus for this stream of research is on performance or Efficiency during inference. Frequently, this work focuses on pruning models during training to improve Efficiency at inference, see, e.g., Liu et al. (2017) and Han et al. (2015), which both use the reduction in floating point operations per inference as a measure of how their pruning approaches improve network efficiency. Examples of work in this area that are relevant to this work include Canziani et al. Canziani et al. (2016), and Jurj et al. Jurj et al. (2020). Both of these works propose measures of Efficiency during inference, and what is particularly relevant for this work is that they use a direct measure of energy consumed (rather than FLOPs) as a measure of resource usage (work done) when calculating Effi-

[1] We note that within the research on improving optimization algorithms the concept of training efficiency is often framed in terms of the convergence rate achieved by the algorithm for a fixed architecture on a learning task (see e.g. Kingma and Ba 2014; Ying et al. 2024). By contrast, in this work, we are focused on measuring the training efficiency of a neural architecture (rather than an optimization algorithm) on the task.

ciency. Similarly, Desislavov et al. (2021) examines the trends in computational and energy costs associated with deep learning model inference and assesses whether the exponential growth in model parameters translates into a proportional increase in energy consumption. Their analysis considers algorithmic improvements and hardware advancements to understand their impact on energy consumption. We conclude that algorithmic advancements and hardware specialization have significantly improved the energy efficiency of DNNs.

The work most relevant to this research is focused on Efficiency during model training. As noted in Schwartz et al. (2020), in the research model, training occurs much more frequently than post-deployment inference, so understanding Efficiency during training is in and of itself an important topic. Indeed, Schwartz et al. (2020) reviews several different measures for Efficiency or work done during training (including, *carbon emissions*, *electricity usage*, *elapsed real time*, *number of parameters*, and *floating point operations (FLOPS)*) and argue that FLOPS is the fairest measure to use to compare different approaches. They attribute two properties to FLOPS in support of this argument: (a) FLOPS directly measures the work done when running a specific instance of a model and, therefore, is related to the energy consumed, and (b) it is agnostic to the hardware on which the model is run. However, metrics based on counts of operations performed by a neural network require hardware profiling, and this is computationally expensive to perform (Mills et al. 2021). Consequently, developing a metric for training efficiency that does not require hardware profiling is desirable. Bartoldson et al. (2023) presents a recent review of the most commonly used metrics in efficiency research, including training time, FLOPs, number of model parameters, electricity usage, carbon emissions, and operand sizes. Overall, they found that all these metrics have significant limitations in either not directly measuring the factors of interest or being dependent on confounding factors such as hardware, time, etc. Finally, we note that all of the metrics discussed above (be it FLOPs, CO_2 emissions (Strubell et al. 2020) or using wall time as a measure (Li et al. 2020)) do not consider model accuracy on a task and so do not measure efficiency *per se* but rather are an estimate of work done. We propose a novel efficiency metric considering the relationship between accuracy and work/resource usage.

However, we looked for alternative energy consumption and Efficiency measures during training to avoid the hardware profiling challenges associated with FLOPS measures. Li et al. Li et al. (2016) explore the power behavior and energy consumption of several CNN architectures on both CPUs and GPUs, with a particular focus on characterizing the energy consumption of different layer types (convolution, pooling, ReLU, and so on) during training. Similar to Li et al. Li et al. (2016) (and Canziani et al.'s work on inference efficiency (Canziani et al. 2016), and Strubell et al.'s work on predicting CO_2 emissions (Strubell et al. 2020)), we propose using energy consumed rather than FLOPS as our measure of work done/resource usage. Also, like Canziani et al. Canziani et al. (2016), we are interested in measuring Efficiency, that is, the relationship between performance (e.g., accuracy) and resource usage (e.g., energy consumed). However, we are focused on the training phase rather than on inference. Furthermore, like Strubell et al. Strubell et al. (2020) and Li et al. Li et al. (2016), we focus on the training phase. However, we go beyond measuring the energy consumed in training a specific model and propose a measure of the relative Efficiency of a neural architecture (distinct from a specific model) on a given task. We compare the LeNet CNN architecture against a Bayesian Convolutional Network (BCNN) as a test case for our efficiency measure. We chose this comparison because BCNNs are not trained

with backpropagation, and we conjecture that this comparison may reveal exciting interactions between training regimes and model efficiency.

3 Defining an efficiency measure for deep neural networks

The concept of Efficiency is fundamental to this work:

Definition 1 Efficiency measures a system's capacity to achieve a goal (measured by a metric) with a given amount of resources.

When considering the training efficiency of a neural network on a learning task, it is natural to consider how the accuracy of the network architecture varies as the energy consumed for training changes. This is the efficiency ratio that equation 1 defines and that Figure 1 illustrates (in this figure, the arrow represents an efficiency calculation—in the form of Equation 1—where the arrow points from the denominator to the numerator).

$$Efficiency \propto \frac{Accuracy}{Energy} \tag{1}$$

However, it is difficult to directly calculate a general estimate of the ratio of energy to accuracy for a given neural architecture on a task because the ratio is dependent on measures used to measure energy and accuracy and is sensitive to hyperparameter decisions (e.g., network size), and training regime decisions (e.g., convergence criteria). Consequently, in this section, we set out a methodology for calculating this efficiency ratio by averaging across a sequence of experiments that allow for hyperparameters and training regime variations. Then, we used these results to compute our final measure.

3.1 Metrics for Energy and Accuracy

Deciding what system components to report energy consumption over is not trivial. For example, although the CPU, GPU, and memory are natural system components to consider when tracking energy consumption during the training of a network, other parts of the system, such as fans, buses, and transistors, also consume energy related to training (Huang et al. 2019; Li and John 2003). However, due to the difficulties in measuring the energy consumption of these secondary or satellite components, we have decided to focus our analysis on the energy consumed during our experiments from the GPU, CPU, and RAM.

We could use several different measures to measure these components' energy usage. For example, one family of energy measures often used for neural network research is those based on counting the number of computational operations; for example, Schwartz et al.

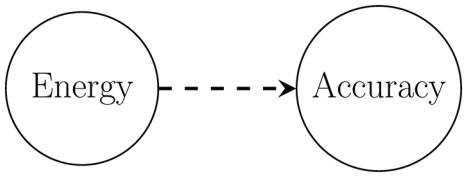

Fig. 1 Network training efficiency visualized as the ratio of accuracy to energy

suggest using the number of FLOPS (Schwartz et al. 2020). FLOPS, however, is one of many types of operation that can be considered. Data movement operations can be much more expensive regarding energy consumption (Horowitz 2014; Sze et al. 2020). However, one of the challenges with tracking energy consumption by counting operations is that the energy consumed by an operation is affected by the sparsity of the data being processed and the data representation being used (Zheng and Mazumder 2019). For example, switching from 32 to 16-bit floating point reduces the energy cost of FLOP operations (and in some cases, this can be done with negligible impact of model accuracy (Micikevicius et al. 2017)) and also reduces energy consumption by reducing data movement (i.e., reduced memory bandwidth) and reduced energy per memory access (due to smaller memories).

In our experiments[2] the hardware used was a Tesla T4 with 15109 MiB memory, from Google Colab (driver version 470.63 and CUDA version: 11.2) and energy collection for the GPU was done using NVIDIA System Management Interface version 460.39 and for recording the energy consumed by the CPU and RAM during training we use the powertop[3] system interface which is a Unix native system tool. We used these tools in each experiment to repeatedly sample and record the energy consumption rate by the GPU, CPU, and RAM access components as each network is being trained. We then calculate the Efficiency of the trained model as the ratio between the performance obtained by the model and the total energy consumed [4] to train the model, as follows:

$$\text{Eff}\ (Acc, W, i = epoch) = \frac{Acc_i}{\sum_{n=0}^{i}[W_n]} \tag{2}$$

where Acc_i is the accuracy obtained on that epoch of training of the model, $\sum[W_n]$ is the sum of the energy samples obtained up to that epoch of training, and $W_n = W_n^{GPU} \bigoplus W_n^{CPU} \bigoplus W_n^{RAM}$, $\bigoplus$ is the concatenation operation.

The selection of the appropriate measure for model performance depends on the task type (e.g., classification, regression, segmentation, and so on) and factors such as the distribution of class labels within the data (Kelleher et al. 2020). In the experiments we report in this paper, the tasks are classification tasks with balanced label distributions, so we have chosen to use simple accuracy for the task. Specifically, we report a model's accuracy (Acc) on the test set after training has converged. In experiments where we use a hold-out test set methodology, Acc is simply the accuracy of the trained model on the test set. In experiments where we use a *k* cross-fold validation methodology, Acc is the mean accuracy across the *k* validation folds.

Figure 2 illustrates the relationship between these measures. As seen above, in this figure, the arrows represent efficiency calculation where the arrow points from the denominator to the numerator. The dashed arrow highlights the overall efficiency calculation we wish to calculate, $\text{Acc} \,/\, \text{W}$, the average amount of task accuracy obtained per unit of energy (Watt) expended in training.

[2] To demonstrate the applicability of our methodology across different hardware platforms, we replicate the experiments reported in the main body of the paper on different hardware, more details on these experiments are found in A.

[3] https://01.org/powertop

[4] measured in terms of Joules per second (Watts)

3.2 Allowing for hyperparameter variations: model size

To experimentally control for the effect of model size[5] we propose to run each experiment multiple times for each neural architecture using a different size model in each run, and for each model size, record both the total energy consumed during training $\sum_{samples}[W]$ and the accuracy obtained by the model. We then calculate the Efficiency for each model on an experimental task as the ratio of accuracy to the total energy consumed to train it. Finally, we calculate the Efficiency of a neural architecture on an experimental task as the mean Efficiency of the models implementing that architecture on the task. Figure 3 illustrates how model size is included in the experimental design, and Equation 3 defines how we integrate model size into the calculation of the training efficiency of a network architecture.

$$\text{Eff}\ (arch, j = size) = \mathop{\mathbb{E}}_{n=1}^{j}\ [\ \text{Eff}\ (Acc, W, i = epoch)_j] \tag{3}$$

3.3 Training regime variations: convergence criteria

The training efficiency of a network (accuracy/energy) is likely to vary as training progresses; in other words, the gain in model accuracy per unit of energy expended is likely to change between the early epochs of training and the later epochs of training. At the same time, the amount of time a network is trained for will vary depending on the convergence criteria used to stop training. To control for this, we define four different convergence criteria and run each experiment with each of these criteria (in combination with our *N* model size variations, we will run each experiment *N* times for each of the four convergence criteria). We then calculate the overall training efficiency of network architecture on a task by first calculating the network efficiency for each convergence criterion using Equation 3 and then calculating the expected value across these efficiency scores.

The four convergence criteria we define are:

1. train for a preset number of epochs, in our experiments, we set **Epochs==50**
2. train until the model achieves a preset accuracy on a validation set; in our experiments, we set the accuracy target to **Accuracy==99**
3. use **early stopping** as the training convergence method, i.e., we track model accuracy on the validation set across consecutive training epochs. Training stops if accuracy does not increase across a preset number of epochs (known as the **patience** parameter). In our experiments, we used a level of patience of 3.
4. stop training after a preset energy (W) budget has been consumed, for our experiments, we set the energy budget to be **Energy==100kW**Figure 4 illustrates how these convergence criteria are integrated into the experimental setup, and Equation 4 defines how we calculate an overall mean training efficiency for a network architecture that accounts for both model size and convergence criteria.

$$\text{Eff}\ (arch, k = convergence) = \mathop{\mathbb{E}}_{k}[\ \text{Eff}\ (arch, j = size)_k] \tag{4}$$

[5] We use the term model to denote a particular instantiation of a neural architecture.

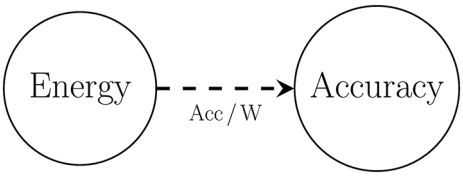

Fig. 2 Visualisation of relationships between variables tracked in the experiments

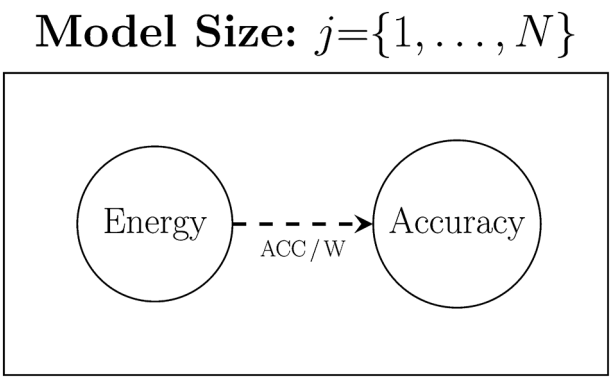

Fig. 3 Visualisation of how model size is integrated into the experimental methodology

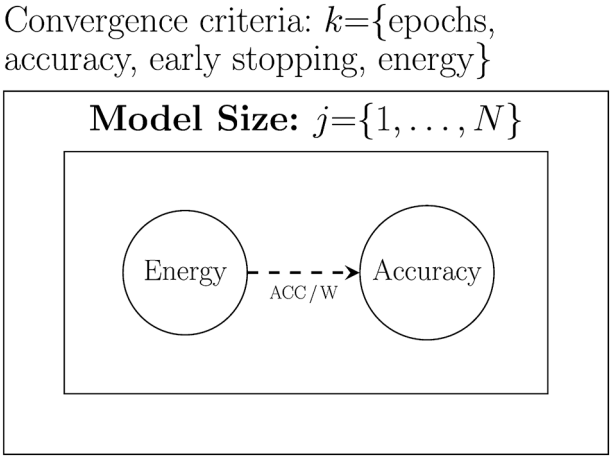

Fig. 4 Visualisation of how convergence criteria are integrated into the experimental methodology

where in the case of Equation 4, $\text{Eff}\ (arch, size)_k$ is computed as in Equation 3.

4 Case study: convolutional and bayesian convolutional architectures

In this case study, we demonstrate the use of our efficiency framework by comparing the Efficiency of a CNN network (LeNet) with that of a Bayesian Convolutional Network (BCNN). The BCNN network is trained using approximate variational inference, which is implemented via dropout. Similar to the experiments reported in the original BCNN paper (Gal and Ghahramani 2015), we use the LeNet-5 architecture from Lecun et al. (1998)

Table 1 LeNet hyperparameters

Architecture	LeNet-5
epochs	50
learning rate	0.001
num workers	4
batch size	256
activation	soft plus
loss	cross-entropy
optimiser	ADAM
initialization	Normal (mean:0, variance:1)

Table 2 BCNN hyperparameters

Architecture	LeNet-5 (Bayesian filters)
epochs	50
learning rate	0.001
num workers	4
batch size	256
activation	soft plus
loss	cross-entropy
optimiser	ADAM
sample size	10^{-25}
train ensemble	1
test ensemble	1
β	0.1
prior μ	0.0
prior σ	0.1
posterior μ_{init}	(0,0.1), (mean, std)
posterior ρ_{init}	(-5,0.1),

as the baseline architecture for our experiments. Following (Gal and Ghahramani 2015), the corresponding Bayesian version of the LeNet baseline was created by applying a dropout with a probability of 0.5 after all convolution and weight layers (i.e., this is the model called "lenet-all" in Gal and Ghahramani (2015)). Tables 1 and 2 report the hyperparameters used to train the LeNet and BCNN models (note: we use the same hyper-parameter settings as reported for experiments performed by Gal and Ghahramani (2015)).

The two models described above are baseline versions of the models used in our experiments. However, in each of our experiments, we vary the model size and different convergence criteria to explore and contrast the efficiency trade-offs for each architecture between size and accuracy and size and Efficiency. In the Bayesian case, two ways of approximating the posterior probability distribution exist: Variational Inference (VI) and Markov Chain Monte Carlo (MCMC). In most cases, (VI) performs excellently but is not a great estimator. While MCMC can be computationally expensive but is an excellent estimator (Charnock et al. 2020), in our experiments, we estimate the posterior using Variational Inference. The best strategy (depth versus width) for scaling a model is an open research challenge. However, because both architectures we consider here are convolutional networks, we decided

to scale the models by increasing the filters used in each layer. In other words, we scaled the width of the models, and we did this by multiplying the number of filters in each layer by multiples from $\times 1$ up to $\times 5$ the original baseline size. This means that in our experiments, we test five versions of the LeNet architecture: LeNet-1, the baseline architecture is the same as reported in Lecun et al. (1998), LeNet-2 has twice the number of filters in each layer as LeNet-1, LeNet-3 has three times the number of filters, and so on up to LeNet-5 with five times the number of filters. Similarly, BCNN-1 is the baseline Bayesian architecture from Gal and Ghahramani (2015) and has the same size as LeNet-1, and BCNN-2 through BCNN-5 are scaled to match their corresponding LeNet-X counterparts in size and structure.

All four efficiency experiments were performed on the MNIST [64] and CIFAR-10 [65] datasets. The same hyperparameters were used for the architectures on both the MNIST and CIFAR-10 datasets[6]. Both of these datasets are based on the task of handwritten numeric digit recognition images, with each image containing a single handwritten digit between 0 and 9. The MNIST dataset contains 10,000 images across ten classes (0-9), each being a 28×28 pixel gray-scale image. The CIFAR-10 also has ten classes with 6000 images per class, each color image being 32×32 pixels. The original experiments with MNIST and CIFAR-10 used different experimental methods: MNIST used a single training and test split, whereas CIFAR-10 used a six-fold cross-validation methodology. In our experiments, we follow the same experimental methodology for each dataset as was reported in the original experiments. Consequently, for the MNIST dataset, a simple split was performed with the training set consisting of 60,000 handwritten digits and our test set of 10,000, and the label distributions in both the training and test sets are balanced across all ten digits. So, in all of our experiments, when we report an accuracy on the MNIST data, this is the accuracy obtained by the model on the single hold-out test set. By contrast, for the CIFAR-10 dataset, we use a 6-fold cross-validation methodology in each experiment, where each fold contains exactly 1000 randomly selected images from each class, and the reported accuracy is the average accuracy of an architecture across these folds.

4.1 Results from the case study

This section presents the results for the 50 epoch, early-stopping, energy-bound, and accuracy-bound experiments. For each experiment, dataset, and neural architecture, we present a table showing the efficiency calculation across model size for each architecture under the convergence criteria specified in that experiment (using Equation 3). Note that in supplementary material we present, for each experiment, plots of the training and test accuracy by training epoch for each model.[7]

4.1.1 50 epoch experiment

In this first experiment, the stopping criterion for training was set at 50 epochs. For each architecture (LeNet and BCNN), the experiment is run a total of 10 times per architecture: once for each of the 5 model sizes (LeNet-1 to LeNet-5, and BCNN-1 to BCNN-5) on

[6] https://unix-talk.com/TastyPancakes/bayesiancnn.git, has all the author's code for the experiments.

[7] All data in the tables from Section 4, and Section 5, is released at: Open Science Foundation

Table 3 MNIST compute for the 50 epochs experiment

Model	Epochs	Acc_i	$\sum_{samples}[W]$	Eff $(Acc, W, epoch)$	Eff $(arch, size)$
BCNN-1	50	0.97×10^{-1}	2.08×10^{5}	4.67×10^{-6}	2.59×10^{-6}
BCNN-2	50	9.78×10^{-1}	4.18×10^{5}	2.34×10^{-6}	
BCNN-3	50	9.77×10^{-1}	4.64×10^{5}	2.11×10^{-6}	
BCNN-4	50	9.77×10^{-1}	5.00×10^{5}	1.95×10^{-6}	
BCNN-5	50	9.76×10^{-1}	5.14×10^{5}	1.90×10^{-6}	
LeNet-1	50	9.91×10^{-1}	0.97×10^{5}	10.15×10^{-6}	8.09×10^{-6}
LeNet-2	50	9.93×10^{-1}	0.90×10^{5}	11.02×10^{-6}	
LeNet-3	50	9.94×10^{-1}	1.45×10^{5}	6.83×10^{-6}	
LeNet-4	50	9.95×10^{-1}	1.52×10^{5}	6.52×10^{-6}	
LeNet-5	50	9.94×10^{-1}	1.67×10^{5}	5.94×10^{-6}	

Table 4 CIFAR compute for the 50 epochs experiment

Model	Epochs	Acc_i	$\sum_{samples}[W]$	Eff $(Acc, W, epoch)$	Eff $(arch, size)$
BCNN-1	50	4.35×10^{-1}	2.87×10^{5}	1.51×10^{-6}	1.54×10^{-6}
BCNN-2	50	4.93×10^{-1}	3.09×10^{5}	1.59×10^{-6}	
BCNN-3	50	5.12×10^{-1}	3.29×10^{5}	1.55×10^{-6}	
BCNN-4	50	5.24×10^{-1}	3.41×10^{5}	1.54×10^{-6}	
BCNN-5	50	5.24×10^{-1}	3.44×10^{5}	1.52×10^{-6}	
LeNet-1	50	6.35×10^{-1}	0.78×10^{5}	8.13×10^{-6}	7.78×10^{-6}
LeNet-2	50	7.18×10^{-1}	0.84×10^{5}	8.47×10^{-6}	
LeNet-3	50	7.71×10^{-1}	0.89×10^{5}	8.66×10^{-6}	
LeNet-4	50	7.88×10^{-1}	0.99×10^{5}	7.91×10^{-6}	
LeNet-5	50	7.96×10^{-1}	1.39×10^{5}	5.72×10^{-6}	

both the datasets (MNIST and CIFAR). During each run of the experiment, we repeatedly recorded the energy being consumed and the amount of memory (GPU and RAM) being used (recorded as model size (MiB) size in RAM and GPU memory).

Table 3 and Table 4 show the efficiency calculation using a convergence criterion of 50 epochs. Note that for the CIFAR dataset, we use a six-fold cross-validation methodology. So for this dataset, the accuracy reported for each model size i (Acc_i) in Table 4 is the average accuracy for that model size across the six validation sets after training has converged.

4.1.2 Early-stopping experiment

This experiment has the same design as the 50 epoch experiment presented above, with a single change in the convergence criteria used for training; in this experiment, we use early-stopping criteria for accuracy.

For the MNIST dataset, Table 5 lists the efficiency calculation using Equation 3. For CIFAR Table 6 presents the efficiency calculation using Equation 3.

Table 5 MNIST compute for the early-stopping experiment

Model	Epochs	Acc_i	$\sum_{samples}[W]$	Eff $(Acc, W, epoch)$	Eff $(arch, size)$
BCNN-1	65	9.67×10^{-1}	9.19×10^{5}	1.05×10^{-6}	1.00×10^{-6}
BCNN-2	21	9.42×10^{-1}	6.31×10^{5}	1.49×10^{-6}	
BCNN-3	37	9.61×10^{-1}	9.26×10^{5}	1.04×10^{-6}	
BCNN-4	53	9.64×10^{-1}	12.21×10^{5}	0.79×10^{-6}	
BCNN-5	65	9.67×10^{-1}	15.40×10^{5}	0.63×10^{-6}	
LeNet-1	16	9.75×10^{-1}	0.75×10^{5}	12.83×10^{-6}	8.77×10^{-6}
LeNet-2	12	9.79×10^{-1}	0.59×10^{5}	16.40×10^{-6}	
LeNet-3	56	9.93×10^{-1}	2.69×10^{5}	3.68×10^{-6}	
LeNet-4	28	9.91×10^{-1}	2.12×10^{5}	4.65×10^{-6}	
LeNet-5	20	9.90×10^{-1}	1.58×10^{5}	6.26×10^{-6}	

Table 6 CIFAR compute for the early-stopping experiment

Model	Epochs	Acc_i	$\sum_{samples}[W]$	Eff $(Acc, W, epoch)$	Eff $(arch, size)$
BCNN-1	61	4.39×10^{-1}	3.58×10^{5}	1.23×10^{-6}	1.02×10^{-6}
BCNN-2	41	4.23×10^{-1}	3.68×10^{5}	1.15×10^{-6}	
BCNN-3	41	4.40×10^{-1}	4.66×10^{5}	0.94×10^{-6}	
BCNN-4	21	3.86×10^{-1}	2.94×10^{5}	1.31×10^{-6}	
BCNN-5	81	4.92×10^{-1}	1.068×10^{5}	0.46×10^{-6}	
LeNet-1	56	5.93×10^{-1}	1.64×10^{5}	3.60×10^{-6}	4.93×10^{-6}
LeNet-2	40	6.53×10^{-1}	1.67×10^{5}	3.90×10^{-6}	
LeNet-3	24	6.45×10^{-1}	0.92×10^{5}	6.96×10^{-6}	
LeNet-4	24	6.65×10^{-1}	1.18×10^{5}	5.63×10^{-6}	
LeNet-5	28	7.18×10^{-1}	1.57×10^{5}	4.56×10^{-6}	

4.1.3 Energy bound experiment

In this experiment, the convergence criterion used to stop training was when the energy samples recorded for a training run on an architecture cumulatively summed up to 100,000 W. Apart from this, the design of the experiment is the same as those reported in the previous two sections.

Mirroring the results from the previous experiments, for the MNIST dataset, Table 7 lists the efficiency calculation using Equation 3. Similarly, for CIFAR, Table 8 presents the efficiency calculation using Equation 3. Note that some of the values for total energy listed in the results for this experiment are above the training convergence criterion of 100,000W. These values are correct values from the experiment. The reason for these values is that although we sample throughout the training process (the average sampling rate for energy was 973 per second for the NVIDIA system and 1052 samples per second for the AMD system), we perform the check of the cumulative amount of energy consumed during training at the end of each epoch. Consequently, the energy consumed during a training run exceeds the stropping threshold if the process crosses that threshold during an epoch.

Table 7 MNIST compute for the energy bound experiment

Model	Epochs	Acc_i	$\sum_{samples}[W]$	Eff $(Acc, W, epoch)$	Eff $(arch, size)$
BCNN-1	19	9.44×10^{-1}	1.46×10^{5}	6.43×10^{-6}	6.18×10^{-6}
BCNN-2	13	9.33×10^{-1}	1.62×10^{5}	5.73×10^{-6}	
BCNN-3	10	9.19×10^{-1}	1.57×10^{5}	5.83×10^{-6}	
BCNN-4	08	9.06×10^{-1}	1.43×10^{5}	6.32×10^{-6}	
BCNN-5	06	8.83×10^{-1}	1.33×10^{5}	6.60×10^{-6}	
LeNet-1	43	9.87×10^{-1}	1.16×10^{5}	8.47×10^{-6}	8.48×10^{-6}
LeNet-2	39	9.90×10^{-1}	1.16×10^{5}	8.46×10^{-6}	
LeNet-3	27	9.89×10^{-1}	1.15×10^{5}	8.58×10^{-6}	
LeNet-4	23	9.89×10^{-1}	1.15×10^{5}	8.60×10^{-6}	
LeNet-5	21	9.89×10^{-1}	1.19×10^{5}	8.29×10^{-6}	

Table 8 CIFAR compute for the energy bound experiment

Model	Epochs	Acc_i	$\sum_{samples}[W]$	Eff $(Acc, W, epoch)$	Eff $(arch, size)$
BCNN-1	19	1.40×10^{-1}	1.49×10^{5}	0.94×10^{-6}	2.21×10^{-6}
BCNN-2	14	3.59×10^{-1}	1.36×10^{5}	2.64×10^{-6}	
BCNN-3	11	3.25×10^{-1}	1.14×10^{5}	2.85×10^{-6}	
BCNN-4	09	3.07×10^{-1}	1.31×10^{5}	2.33×10^{-6}	
BCNN-5	07	2.71×10^{-1}	1.19×10^{5}	2.28×10^{-6}	
LeNet-1	39	5.72×10^{-1}	1.18×10^{5}	4.83×10^{-6}	5.17×10^{-6}
LeNet-2	36	6.44×10^{-1}	1.15×10^{5}	5.59×10^{-6}	
LeNet-3	32	6.77×10^{-1}	1.16×10^{5}	5.83×10^{-6}	
LeNet-4	25	6.80×10^{-1}	1.17×10^{5}	5.77×10^{-6}	
LeNet-5	21	6.71×10^{-1}	1.75×10^{5}	3.83×10^{-6}	

4.1.4 Accuracy bound experiment

The convergence criteria used in these experiments was to stop training when a model obtained a specified accuracy threshold. For the MNIST dataset this accuracy threshold was set at 99% on the training set, and on the CIFAR dataset (where we used a six-fold cross-validation methodology) for each fold the training was stopped when the model had obtained an accuracy threshold of 50% on the training data for that fold[8]. Our reason for using a lower accuracy threshold for CIFAR was that an accuracy threshold $> 50\%$ required training to proceed for more time than our Collab account allowed, and if this time threshold was exceeded, then the training was interrupted, and results were lost.

For MNIST Table 9 lists the efficiency calculation using Equation 3. Similarly, for CIFAR, Table 10 presents the efficiency calculation using Equation 3.

[8] See the final paragraph of Section 4 for details of the training and test split used for MNIST and the six-fold cross-validation methodology used for CIFAR.

Table 9 MNIST compute for the accuracy bound experiment

Model	Epochs	Acc_i	$\sum_{samples}[W]$	Eff $(Acc, W, epoch)$	Eff $(arch, size)$
BCNN-1	72	9.70×10^{-1}	13.32×10^{5}	0.73×10^{-6}	1.74×10^{-6}
BCNN-2	69	9.71×10^{-1}	14.05×10^{5}	0.69×10^{-6}	
BCNN-3	69	9.70×10^{-1}	14.81×10^{5}	0.66×10^{-6}	
BCNN-4	77	9.72×10^{-1}	1.60×10^{5}	6.06×10^{-6}	
BCNN-5	80	9.72×10^{-1}	17.06×10^{5}	0.57×10^{-6}	
LeNet-1	12	9.70×10^{-1}	0.57×10^{5}	16.86×10^{-6}	26.10×10^{-6}
LeNet-2	08	9.73×10^{-1}	0.44×10^{5}	22.02×10^{-6}	
LeNet-3	06	9.74×10^{-1}	0.36×10^{5}	26.73×10^{-6}	
LeNet-4	06	9.75×10^{-1}	0.36×10^{5}	26.47×10^{-6}	
LeNet-5	04	9.75×10^{-1}	0.25×10^{5}	38.44×10^{-6}	

Table 10 CIFAR compute for the accuracy bound experiment

Model	Epochs	Acc_i	$\sum_{samples}[W]$	Eff $(Acc, W, epoch)$	Eff $(arch, size)$
BCNN-1	51	4.24×10^{-1}	5.78×10^{5}	0.73×10^{-6}	0.68×10^{-6}
BCNN-2	37	4.17×10^{-1}	4.98×10^{5}	0.84×10^{-6}	
BCNN-3	30	4.14×10^{-1}	5.32×10^{5}	0.78×10^{-6}	
BCNN-4	36	4.21×10^{-1}	8.07×10^{5}	0.52×10^{-6}	
BCNN-5	33	4.24×10^{-1}	8.32×10^{5}	0.51×10^{-6}	
LeNet-1	7	4.22×10^{-1}	0.60×10^{5}	7.00×10^{-6}	10.35×10^{-6}
LeNet-2	4	4.29×10^{-1}	0.37×10^{5}	11.55×10^{-6}	
LeNet-3	4	4.63×10^{-1}	0.43×10^{5}	10.65×10^{-6}	
LeNet-4	3	4.42×10^{-1}	0.33×10^{5}	13.24×10^{-6}	
LeNet-5	3	4.68×10^{-1}	0.50×10^{5}	9.30×10^{-6}	

5 Analysis of experimental data

This section presents the analysis of the data obtained from our experiments regarding how Efficiency behaves as training progresses, the relationship between model size and Efficiency, and the relative overall Efficiency of the LeNet and BCNN architectures.

5.1 Efficiency as training progresses

Figure 5 and Figure 6 plot for each of the models trained (LeNet sizes 1–5, and BCNN sizes 1–5) how the Efficiency of the model changes across epochs as training progresses. We base this analysis solely on the results from the 50 epoch experiment because, in this experiment, we have collected the same number of epochs for all sizes and both architectures. As a result, the x-axis, which records the training epochs, goes from 0 to 50 in both figures. The y-axis in the graph plots the Efficiency of a model at a given epoch as defined by Equation 2. This definition of *Efficiency* is the ratio of a model's performance on a validation set after epoch i of training to the cumulative energy expended in training the model up to that point in training.

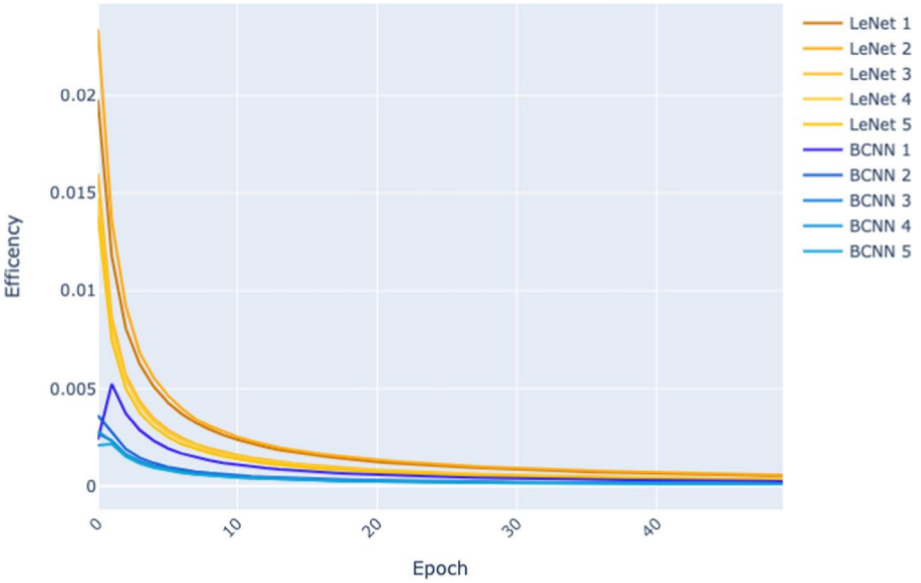

Fig. 5 Efficiency per epoch (MNIST dataset) of the 50 epoch experiment

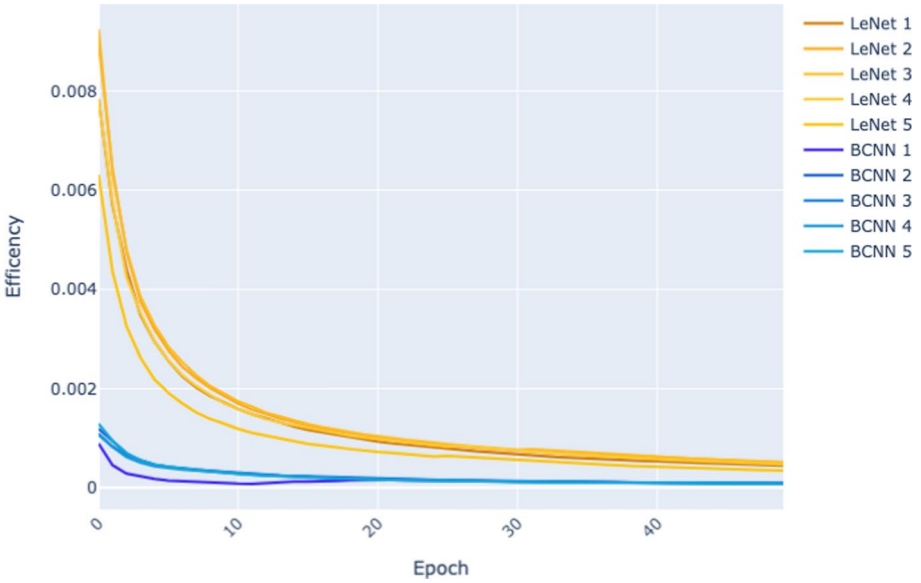

Fig. 6 Efficiency per epoch (CIFAR dataset) of the 50 epoch experiment

For these results, we observe that Efficiency decreases as time progresses. These plots show that although we would expect the performance of a model to improve as training progresses, the rate of improvement tends to decrease as training progresses. After a certain amount of training (epochs), performance plateaus and further training result in energy

being expended. Notice that the plots in Figure 5 drop more steeply than those in Figure 6. This reflects the fact that on the more straightforward MNIST dataset, model performance saturates very early on, whereas, on the more complex CIFAR dataset, it takes more epochs for the models to reach this performance saturation point.

The relative difficulty of the two datasets is also reflected in the differences in the y-axis scales between Figure 5 and Figure 6. The maximum Efficiency recorded for any models at any epoch on MNIST is above 0.02, whereas on CIFAR, it is below 0.01. The primary driver of this difference is that on MNIST, the models achieved accuracies of 0.97–0.99 (see Table 3), whereas on CIFAR, the range of accuracies of the BCNN models is 0.43–0.53 and the LeNet models 0.66–0.80 (see Table 4).

Finally, comparing Figure 5 with Figure 6, it is apparent that the gap between the plots for the LeNet models and the BCNN models is more significant in Figure 6. This suggests that as a learning task becomes more complex, differences in Efficiency become more pronounced.

5.2 Relationship between stopping criteria and efficiency, and model size and efficiency

The results presented in Tables 3– 10 reveal significant variation in architecture efficiency across different stopping criteria. Note that this analysis considers the variation in Efficiency by model size. This variation is particularly noticeable in the MNIST dataset. Table 11 summarises (from Tables 3, 5, 7 and 9) the efficiency results for both architectures across the four stopping criteria on the MNIST dataset. Examining the results for LeNet, the maximum Efficiency (0.00002610) is obtained using an accuracy bound stopping criterion, and the minimum Efficiency (0.00000809) is recorded using the 50 epoch criterion. This means that LeNet is, averaging across model sizes, approximately 3.22 times more efficient on MNIST when the accuracy bound criterion is applied compared to the 50 epoch criterion. A similar variation in Efficiency across stopping criteria is observable for the BCNN architecture. However, the criteria that result in the maximum and minimum values differ. For the BCNN architecture on MNIST, using an energy bound stopping criterion gives the maximum Efficiency of 0.00000618 compared to the minimum Efficiency of 0.00000100 using early stopping, a variation in Efficiency of 6.18 times. More generally, we observe a complex non-linear interaction across architectures and convergence criteria, as shown in Figure 7, which plots the LeNet versus BCNN efficiency scores by convergence criteria. The within-architecture efficiency variation across stopping criteria and the complex interactions across architectures and stopping criteria highlight the need to include multiple stopping criteria within the efficiency framework.

Analyzing the relationship between stopping criteria and Efficiency in more detail, Figure 8 and Figure 9 visually summarizes the efficiency analysis results from across the 50

Table 11 MNIST mean efficiency scores for LeNet and BCNN by stopping criteria

	LeNet	BCNN
50 Epoch	8.09×10^{-6}	2.59×10^{-6}
Early Stopping	8.77×10^{-6}	1.00×10^{-6}
Energy Bounded	8.48×10^{-6}	6.18×10^{-6}
Acc. Bounded	26.10×10^{-6}	1.74×10^{-6}

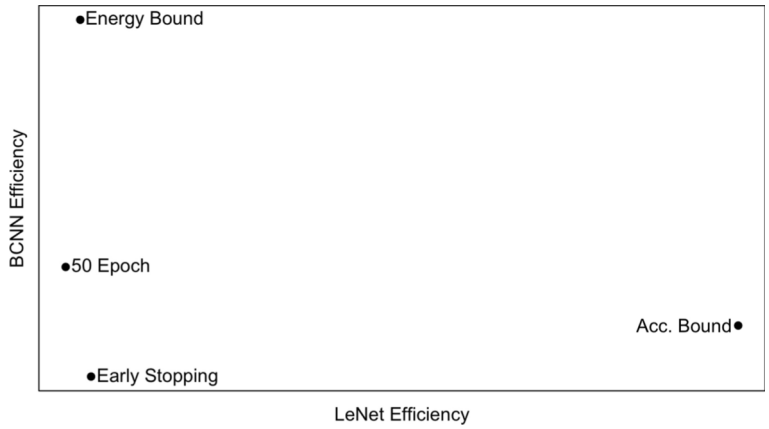

Fig. 7 LeNet versus BCNN efficiency on MNIST by stopping criteria

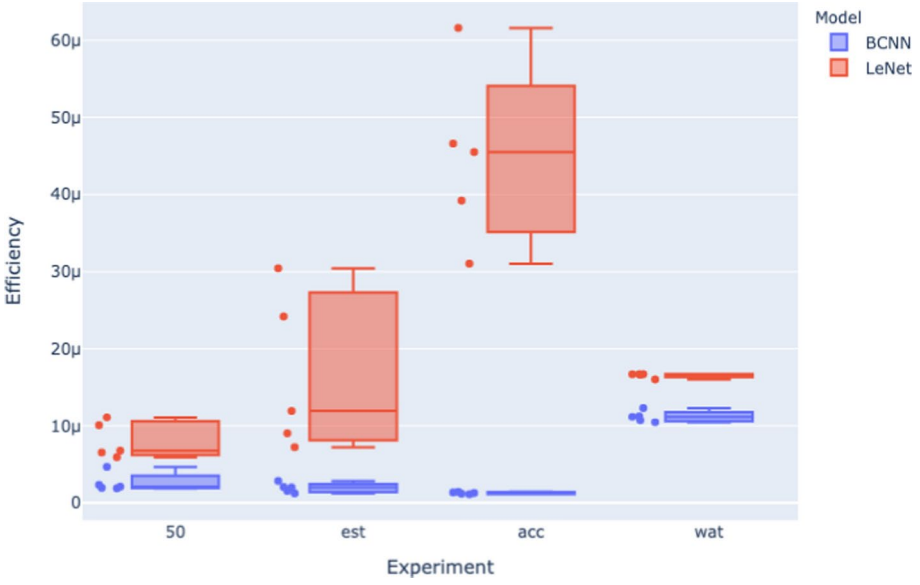

Fig. 8 Efficiency for the 4 experiments (MNIST dataset)

epoch (50), early stopping (est), energy bound (wat), and accuracy bound (acc) experiments. In these figures, the x-axis indicates the stopping criteria of the models being assessed, the y-axis is the efficiency results per model size obtained from Equation 3, there are five model sizes for each architecture in each experiment, and so each box plot contains five efficiency results, and one box plot per stopping criteria.

Both Figure 8 and Figure 9 show that different stopping criteria profoundly influence Efficiency. Variations in stopping criteria affect both the width of the distributions of efficiencies for each architecture and also the distance between these distributions. For example, stopping criteria that bound energy—the energy bound (wat) and the 50 epoch experiments—

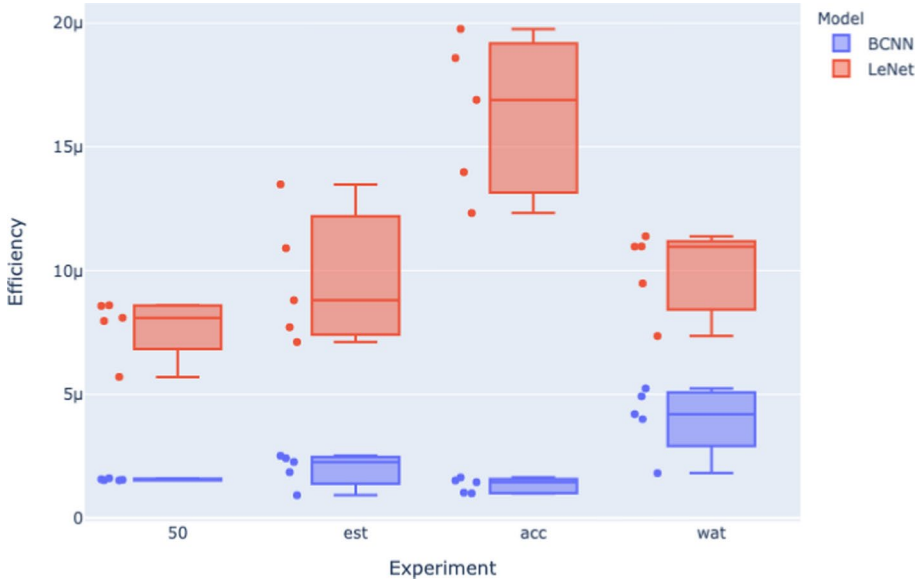

Fig. 9 Efficiency for the 4 experiments (CIFAR dataset)

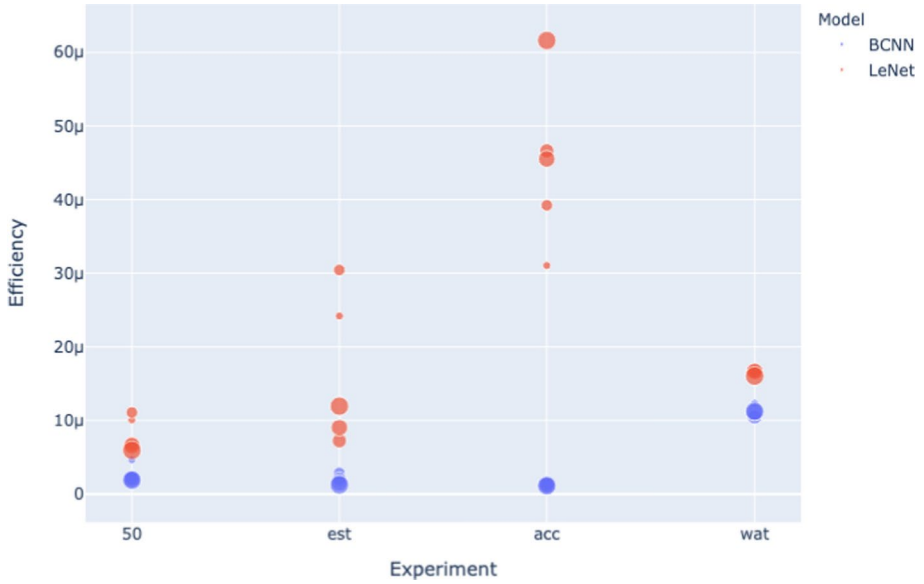

Fig. 10 Efficiency for the 4 experiments (MNIST dataset)

appear to squash the distributions of the model efficiencies of each architecture, whereas stopping criteria based on accuracy bounds—the early stopping (est) and accuracy bound (acc) experiments—the efficiency distributions are wider, particularly for the LeNet model. This energy bound versus accuracy bound categorization of stopping criteria is also predic-

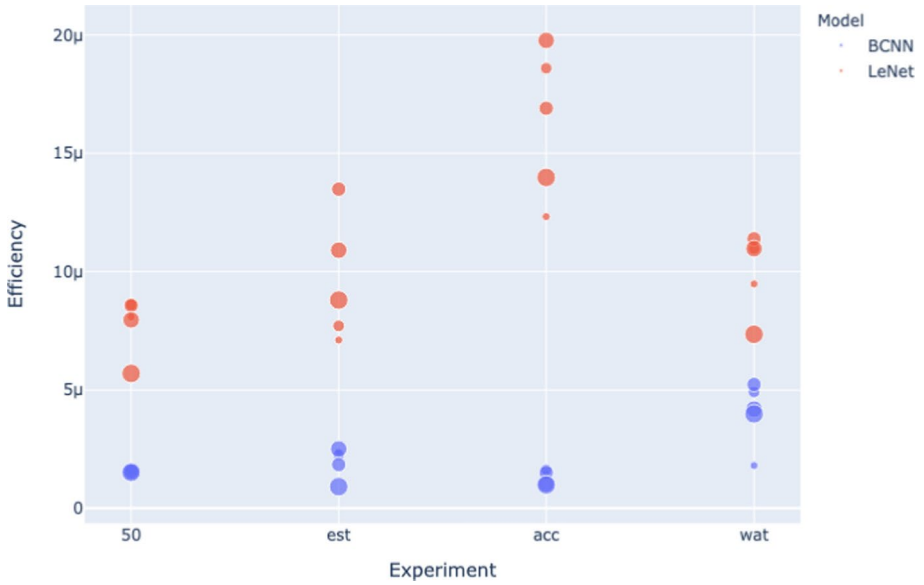

Fig. 11 Efficiency for the 4 experiments (CIFAR dataset)

tive in terms of the gap between the LeNet and BCNN distributions, with accuracy bound experiments (est and acc) exhibiting a more significant gap between the distributions for the architectures as compared with the energy bound (wat and 50 epoch) experiments.

This suggests a trade-off between these two categories of stopping criteria for measuring architecture efficiency. Energy-bound experiments generate narrow efficiency distributions across model sizes, resulting in narrow confidence intervals around the mean Efficiency for a given architecture based on these experiments. However, this relatively more robust confidence is offset by the smaller gap between the efficiency distribution for each architecture. By contrast, the accuracy-bound experiments are more sensitive to differences between architectures in terms of Efficiency. However, the broader distribution per architecture results in wider confidence intervals around the mean Efficiency. In order to balance this trade-off, we suggest using both types of stopping criteria when measuring Efficiency (as done by Equation 4).

The squashing of the distributions when energy-bound stopping criteria are used suggests that for each architecture, a fixed amount of energy per unit of accuracy is obtained independent of model size. In other words, when the stopping criteria are bound to energy, varying model size will not impact the overall architecture efficiency on a learning task. However, when the stopping criteria are based on accuracy, varying the model size will significantly impact the overall architecture efficiency of a learning task.

Figure 10 and Figure 11 show how different model sizes compare to each other. The Efficiency of the LeNet architecture is particularly sensitive to size. However, there is no clear trend between size and Efficiency. The Efficiency of the BCNN architecture is less sensitive to size variation, so the visualizations are less helpful for this architecture. However, examining efficiency results reported in Tables 3, 5, 7, 9, and Tables 4, 6, 8, 10 reveals that there is no apparent trend between model size and Efficiency.

5.3 Efficiency of the LeNet architecture against BCNN architecture

Table 12 presents the overall efficiency calculations for the LeNet and BCNN architectures on the MNIST and CIFAR datasets. These efficiencies are the mean Efficiency of architecture on a dataset across the multiple model sizes and convergence criteria (see Equation 4). On both datasets, LeNet is more efficient than the BCNN architecture.

Comparing the Efficiency of each architecture across the datasets, we can see that both architectures are more efficient on MNIST than on CIFAR. This is due to the relative complexity of CIFAR versus MNIST. In order to check how the Efficiency of an architecture varies across datasets, we take the ratio between the Efficiency of an architecture on one dataset to its Efficiency on another. In this calculation, we take Efficiency on MNIST as the numerator because this is the dataset on which both architectures have the highest Efficiency. For LeNet, this calculation is $12.86/7.06 = 1.822$, i.e., the LeNet architecture is 1.822 times more efficient on MNIST than on CIFAR. For BCNN, this calculation gives us $2.88/1.36 = 2.116$, i.e., the BCNN architecture is 2.116 times more efficient on MNIST than CIFAR. These two ratios are close. However, the ratio for LeNet is smaller than for BCNN $1.822 < 2.116$, indicating that the LeNet architecture has a smaller decrease in Efficiency between MNIST and CIFAR than BCNN. Another perspective on these results is to take the ratio between the two architectures on each dataset. In this case, we use the Efficiency of the LeNet architecture as the numerator because this architecture has the highest Efficiency on both datasets. For MNIST, this calculation is $12.86/2.88 = 4.466$, and for CIFAR, this calculation is $7.06/1.36 = 5.185$. These calculations indicate that on MNIST, LeNet is 4.466 times more efficient than BCNN, whereas on CIFAR, LeNet is 5.185 times more efficient than BCNN. In other words, as the dataset becomes more complex (moving from MNIST to CIFAR), the difference in Efficiency between LeNet and BCNN becomes larger ($5.185 > 4.466$).

To summarise, the CIFAR dataset is the more complex dataset, LeNet is the more efficient architecture on both datasets, and when the learning task switches to a more complex dataset, the relative differences in Efficiency between the architectures become more extensive (the less efficient architecture has a more considerable relative drop in Efficiency, and the ratio between the efficiencies of the architectures increases as the task becomes more difficult). This observation relating the difficulty of the task and changes in the Efficiency of an architecture aligns with what can be observed in Figure 5 and Figure 6 where there is a more significant gap between the LeNet and BCNN plots lines on CIFAR as compared to the plot lines on MNIST.

This comparison of Bayesian Convolutional Neural Networks (BCNNs) and Convolutional Neural Networks (CNNs) highlights a trade-off in training efficiency. BCNNs seek to enhance generalizability by learning a distribution over models rather than fitting a single model to the data (thereby reducing the risk of overfitting) (MacKay 1995). However, learn-

Table 12 Efficiency ($\mathrm{Eff}\ (arch, convergence)$) of BCNN and LeNet architectures on the MNIST and CIFAR datasets

	MNIST	CIFAR	MNIST/CIFAR
LeNet	12.86×10^{-6}	7.06×10^{-6}	1.82
BCNN	2.88×10^{-6}	1.36×10^{-6}	2.11
LeNet/BCNN	4.46	5.18	

ing this distribution requires the repeated sampling of weights during training, which incurs an extra cost in terms of energy. For BCNNs to achieve greater Efficiency than CNNs, their generalization improvement must outweigh the increased energy costs incurred during training. Our findings indicate that, for the tasks we have examined, this trade-off results in BCNNs being less efficient than CNNs in terms of accuracy versus energy.

5.4 On the risks of over-training (over-fitting)

As discussed in Section 5.1, the Efficiency of a neural architecture tends to decay as training progresses; this trend is evident in Figure 5 and Figure 6 where for both architectures on both datasets efficiency consistently reduces as training progresses. This trend reflects that as training progresses, model performance saturates after a certain point, and further training expends more energy with no gain in performance. An implication of this is that if a neural model is trained for an extreme number of epochs, then the training efficiency of that architecture will tend to zero, and furthermore, in such a scenario, comparing the Efficiency of different neural architectures is no longer sensible because all architectures will have an efficiency of zero. Put another way, the measurement of the training efficiency for a neural architecture only makes sense when models are not overtrained.

The most direct definition of overtraining is epochs of training that do not improve model performance. Another complementary way of identifying when overtraining has occurred is through the concept of over-fitting. Overfitting occurs when a model learns to perform well on the training data but fails to generalize to unseen data, compromising its Efficiency. Overfitting can be checked for by comparing the divergence between a model's performance on training data versus non-training data. To illustrate both overfitting and the impact of overtraining training efficiency, we extend our 50-epoch experiment to 100 epochs. We then perform two levels of analysis. First, we check whether the models trained for 100 epochs exhibit overtraining (compared to those trained for 50 epochs). Then, we calculate the Efficiency of both architectures using the results from the 100-epoch experiment in order to understand how overtraining can affect training efficiency.

We examine two measures to check whether extending training from 50 to 100 results in overtraining a model. First, we check whether the extra training resulted in an appreciable increase in model performance on the test set; if there is no increase in test set performance between the 50th and 100th epoch, then we deem the 100 epoch model to be overtrained. Second, suppose a model exhibits an increase in test set performance between the 50th and 100th epochs. We check for overfitting by comparing the model's performance on the training data and the test set. The intuition behind this analysis is that the more significant the drop in the performance between the training data and a test set, the more likely the model will be overfitted (and hence overtrained). In more detail, we calculate the difference between a model's training and test performance after 50 epochs of training and after 100 epochs of training and then calculate the delta between these differences. This delta in the differences reveals the extent of divergence between training and test performance caused by the extra 50 epochs of training. Using this delta metric, we deem a model to be overtrained if the delta is of a comparable scale to the increase in the test set performance of the model between the 50th and 100th epochs.

Table 13 presents the performance results used in this analysis. For the 50 and 100 epoch results, the table presents the model performance on the training set, the test set, and the

Table 13 An analysis of model over-fitting after 100 epochs. Column A lists the per-model increase in test set performance between 50 and 100 epochs (Test accuracy after 100 epochs minus Test accuracy after 50 epochs). Column B lists the per model delta in the training and test difference between 50 and 100 epochs (Difference at 100 minus Difference at 50)

	50 epoch			100 epoch				
MNIST	Train	Test	Difference	Train	Test	Difference	A	B
LeNet-1	0.99071102	0.98506103	0.00564999	0.99524102	0.98694648	0.00829454	0.00	0.00
LeNet-2	0.99438373	0.98787305	0.00651068	0.99708797	0.98906119	0.00802678	0.00	0.00
LeNet-3	0.99525017	0.99059342	0.00465675	0.99761469	0.99154647	0.00606822	0.00	0.00
LeNet-4	0.99565284	0.99153448	0.00411836	0.99782642	0.99276667	0.00505975	0.00	0.00
LeNet-5	0.99629737	0.99057988	0.00571749	0.99814869	0.99128859	0.00686010	0.00	0.00
BCNN-1	0.95748047	0.96715924	-0.00967877	0.97338431	0.97560624	-0.00222193	0.01	0.01
BCNN-2	0.96626704	0.97242924	-0.0061622	0.97819731	0.97928500	-0.00108769	0.01	0.01
BCNN-3	0.96513256	0.97038611	-0.00525355	0.97763173	0.97727584	0.00035589	0.01	0.01
BCNN-4	0.96491689	0.96950655	-0.00458966	0.97758249	0.97698637	0.00059612	0.01	0.01
BCNN-5	0.96142537	0.96888013	-0.00745476	0.97537774	0.97666603	-0.00128829	0.01	0.01
CIFAR	Training	Testing	Difference	Training	Testing	Difference		
LeNet-1	0.59485669	0.57332227	0.02153442	0.65716212	0.61330469	0.04385743	0.04	0.02
LeNet-2	0.70195860	0.64035352	0.06160508	0.76755101	0.66876953	0.09878148	0.03	0.04
LeNet-3	0.76978155	0.68373047	0.08605108	0.83391819	0.70827051	0.12564768	0.02	0.04
LeNet-4	0.80343700	0.69496875	0.10846825	0.86194093	0.71430078	0.14764015	0.02	0.04
LeNet-5	0.82096686	0.69295508	0.12801178	0.88013734	0.71263281	0.16750453	0.02	0.04
BCNN-1	0.33502488	0.33920898	-0.0041841	0.43809589	0.43243262	0.00566327	0.09	0.01
BCNN-2	0.44060957	0.43411328	0.00649629	0.50035355	0.48572559	0.01462796	0.05	0.01
BCNN-3	0.45429289	0.44414062	0.01015227	0.52243183	0.50286328	0.01956855	0.06	0.01
BCNN-4	0.45937550	0.45482617	0.00454933	0.53284086	0.51709375	0.01574711	0.06	0.01
BCNN-5	0.46070014	0.45722070	0.00347944	0.53233031	0.51748633	0.01484398	0.06	0.01

difference between these results. The rightmost two columns of the table (columns A and B) list the difference in test performance between 50 and 100 epochs (calculated as test performance at 100 epochs minus test performance at 50 epochs) and the delta in the differences between training and test performance between 50 and 100 epochs (calculated as the difference between training and test performance at 100 epochs minus the difference between training and test performance at 50 epochs). In order to highlight meaningful differences in columns A and B, we round the results in these columns to two decimal places. If we examine column A, we see that on the MNIST dataset, none of the LeNet models obtain a meaningful increase in test performance between the 50th and 100th epochs. As a result, we consider the LeNet 100 epoch models to be overtrained. The BCNN models on MNIST exhibit a slight increase (≈ 0.01 for all models) in test set performance between the 50th and 100th epoch. However, this is accompanied by a comparable increase in the divergence between training and test set performance, so we also deem these BCNN models to be overtrained. Switching focus to the CIFAR dataset, all of the LeNet models exhibit an increase in test performance between the 50th and 100th epoch. However, this is accompanied by a comparable (and in 4 out of 5 cases more prominent) increase in divergence between training and test performance, so we deem these 100 epoch LeNet CIFAR models to be overtrained. Finally, the BCNN models on the CIFAR dataset all exhibit a relatively significant increase in test performance between the 50th and 100th epoch, accompanied by a comparably slight increase in divergence between training and test performance, so we

deem these models not to be overtrained. In summary, our analysis of overtraining after 100 epochs categorized all the LeNet and BCNN MNIST models, the LeNet CIFAR models as overtrained, and the BCNN CIFAR models as not overtrained.

To analyze how overtraining can affect the measurement of training efficiency, we used Equation 3 to calculate the Efficiency of both architectures on both datasets based solely on the results of the 100 epoch experiment. The results of these calculations are presented in Table 14. We are comparing these results with those listed in Table 12; a consistent finding across both sets of results is that LeNet is more efficient than BCNN on both datasets. Also, for three out of the four categories of models (LeNet and BCNN on MNIST, and LeNet on CIFAR), the training efficiency drops as compared with Table 12, this is in line with what would be expected from the trends exhibited in Figure 5 and Figure 6 discussed in Section 5.1. The one exception to this trend is the BCNN architecture on CIFAR, which slightly increases Efficiency. This exception aligns with the findings of our overtraining analysis presented above. It suggests that if we were to use the efficiency scores presented in Table 14 to compare the efficiency scores of LeNet and BCNN, we would be comparing overtrained LeNet models against BCNN models, some of which are overtrained (i.e., BCNN MNIST) and some of which are not (i.e., BCNN CIFAR). If we run this (incorrect) comparison through to see how overtraining can affect the overall analysis, we get very different conclusions from those we reached from analyzing Table 12. For example, let us compare the efficiency ratio for each architecture across the two datasets (i.e., MNIST/CIFAR). We see that in Table 14 for LeNet, this ratio (1.042) is greater than the BCNN ratio (0.367). Similarly, if we compare the efficiency ratio between the two architectures on each dataset (LeNet/BCNN), we see that this ratio is more significant for MNIST (3.108) than for CIFAR (1.095). In both cases, the relative size of these ratios has flipped as compared with the results reported in Table 12. Taking the ratios in Table 14 at face value, we would (erroneously) conclude that as the learning task becomes more complex (MNIST→ CIFAR), the more efficient architecture (LeNet) has a more significant drop in Efficiency and that the difference in Efficiency between the two architectures becomes smaller. However, the underlying phenomenon driving these results is overtraining. Consequently, when assessing the training efficiency of neural architecture, it is essential to consider overtraining as a factor in the analysis and to be cognizant that overtraining can occur at different points in training for different models on a given training task. One strategy to mitigate the risk of overtraining impacting efficiency analysis is to average over multiple convergence criteria, as we have done in this work.

Table 14 Efficiency ($\mathrm{Eff}\ (arch, convergence)$) of BCNN and LeNet architectures on the MNIST and CIFAR datasets for models trained for 100 epochs

	MNIST	CIFAR	MNIST/CIFAR
LeNet	2.05×10^{-6}	1.97×10^{-6}	1.04
BCNN	0.66×10^{-6}	1.80×10^{-6}	0.36
LeNet/BCNN	3.10	1.09	

6 Conclusions

We present a framework for measuring the training efficiency of a neural architecture on a learning task. This framework involves running multiple experiments but does not require hardware profiling. Moreover, the framework enables a multifaceted analysis of the training efficiency of a neural architecture, including the analysis of how the Efficiency of a model varies across training epochs (Equation 2), how the Efficiency of a neural architecture varies with model size (Equation 3) and the overall Efficiency of a neural architecture on a learning task taking into account variations in model size and stopping criteria (Equation 4). Furthermore, the ability to calculate an overall efficiency for a neural architecture on a learning task enables the analysis of the relative Efficiency of different neural architectures on a learning task and how the relative Efficiency of neural architectures varies across learning tasks.

Applying the framework to the case study comparing CNNs with BCNNs on MNIST and CIFAR, we find that the Efficiency of both architectures on both learning tasks changes substantially as training progresses (see Section 5.1), with all models exhibiting a drop in Efficiency across epochs. The analysis in Section 5.2 reveals a non-linear relationship between stopping criteria and training Efficiency and model size and training Efficiency. We observed significant variation in training efficiency across different stopping criteria for both architectures. This variation across stopping criteria illustrates the need for multiple stopping criteria within the efficiency framework. Moreover, including multiple convergence criteria within the framework mitigates the risk of overtraining affecting the analysis of the training efficiency of neural architectures (see Section 5.4). More generally, we believe that the potential confounding effect of overtraining on neural training efficiency research is not given sufficient attention in the literature. To take a recent example, Kaddour et al. (2023) report, as a key finding, that the efficiency improvements obtained by several training regime modifications vanished when the compute budget allowed for training increases. However, in their analysis, the authors did not consider that this finding may result from overtraining occurring at different points under different training regimes. Indeed, the more efficient a training regime is, the earlier in the training process overtraining will begin, in which case, using a fixed compute budget as a convergence criterion is likely to result in more efficient training regimes overtraining for longer. So, the extra overtraining will negate the efficiency benefits of these regimes. This example illustrates how neglecting the impact of overtraining can directly undermine conclusions drawn from an experiment focused on training efficiency. Regarding the relationship between model size and training Efficiency, we find that intermediate-size models have the best Efficiency for both architectures and learning tasks. This variation in Efficiency with respect to model size highlights the need to include model size within the efficiency frameworks.

In terms of overall neural architecture training efficiency on a learning task, we find that CNNs are more efficient than BCNNs on both MNIST and CIFAR and that the difference in Efficiency becomes more prominent as the learning task becomes more complex (see Section 5.3). To test for interactions with hardware, we replicated our experiments and analysis on a second hardware setup. The description of the hardware and the results are presented in A. The same trends are evident in the results obtained from these other experiments. Overall, we argue that to measure the training efficiency of neural architectures, it is important to consider efficiency variation across model size, the stopping criterion used, and the learning task. In future work, we will explore the application of the framework to

Table 15 Hardware characteristics

AlmaLinux 9.2 (Turquoise Kodkod) x86_64
Kernel: 5.14.0284.11.1.el9_2.x86_64
CPU: AMD Ryzen 9 5900HX with Radeon Graphics (16) @ 3.300GHz
GPU: AMD ATI Radeon Vega Series / Radeon Vega Mobile Series
GPU: AMD ATI Radeon RX 6700/6700 XT/6750 XT/6800M/6850M XT
Memory: 3251 MiB / 31496 MiB
Driver version: 6.1.5
ROCm version: 5.4.2
Python version: 3.9.16
Pytorch version: 2.0.1
powerstat version: 0.03.03
radeontop version: 1.00

other neural architectures and training paradigms. For example, there is a growing body of work exploring parameter-efficient fine-tuning, and applying this framework to these methods could reveal important interactions between the neural architecture and the training regimen. Another potential area of future work emerges from our findings that training efficiency and model size have a non-linear relationship. Given this finding, it may be helpful to consider how Efficiency, model size, and model compression methods interact[9].

[9] Supplementary material is available at the Open Science Foundation.

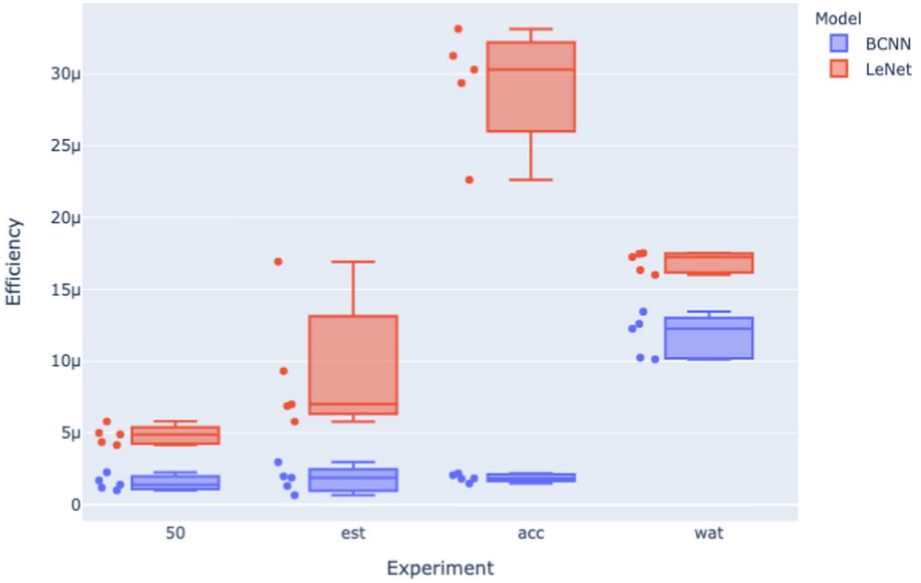

Fig. 12 Box plot for Efficiency per size four experiments (MNIST dataset)

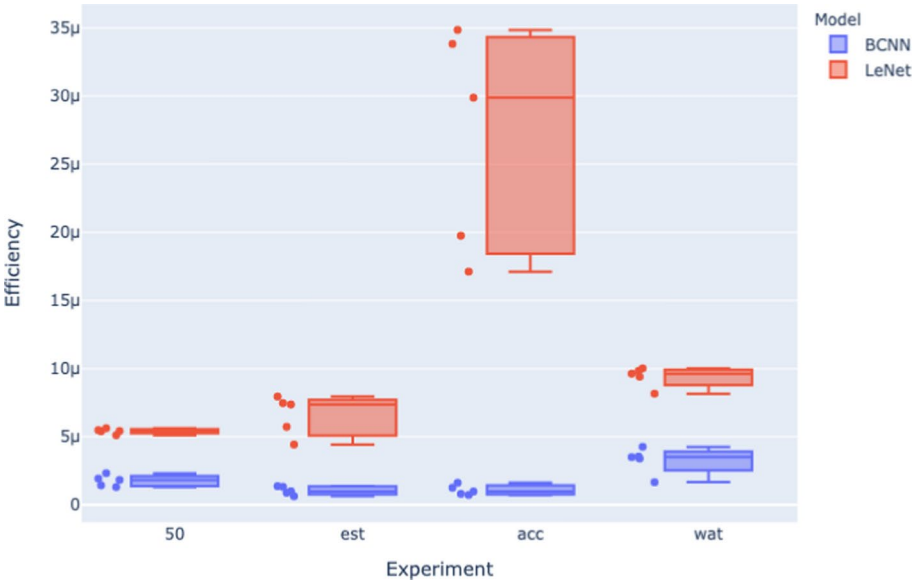

Fig. 13 Box plot for Efficiency per size four experiments (CIFAR dataset)

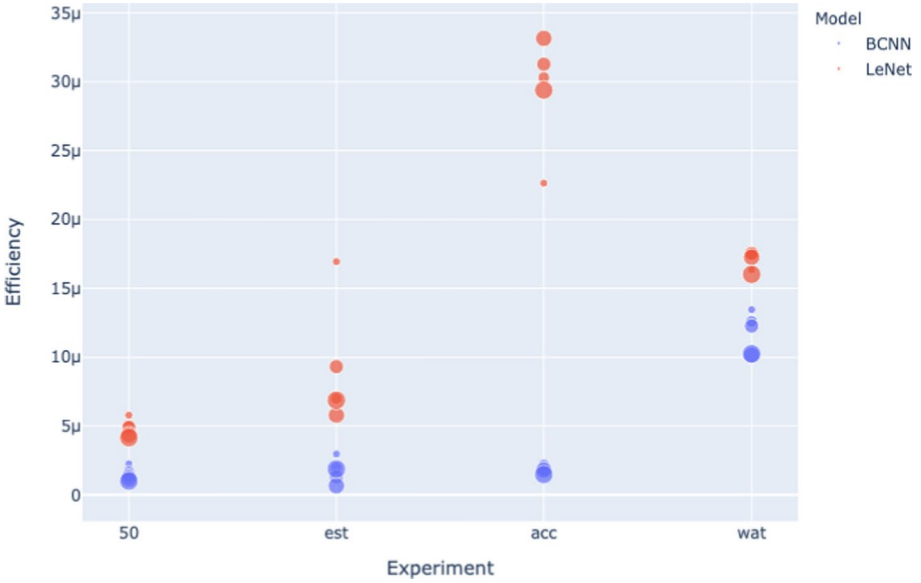

Fig. 14 Scatter plot for the Efficiency 4 experiments (MNIST dataset)

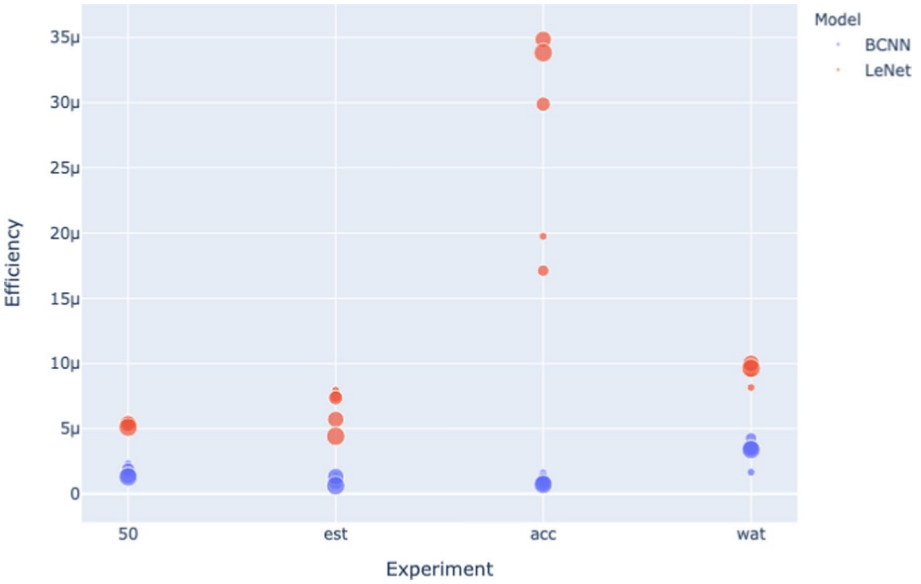

Fig. 15 Scatter plot for the Efficiency 4 experiments (CIFAR dataset)

Table 16 Efficiency ($\mathrm{Eff}\ (arch, convergence)$) of BCNN and LeNet architectures on the MNIST and CIFAR datasets, with AMD hardware

	MNIST	CIFAR	MNIST/CIFAR
LeNet	8.91×10^{-6}	19.30×10^{-6}	0.46
BCNN	2.66×10^{-6}	1.18×10^{-6}	2.25
LeNet/BCNN	3.35	16.41	

Appendix

Hardware comparison

We replicated our experiments on a second hardware setup to demonstrate our framework's generalizability and findings. Table 15 shows the characteristics of this second (AMD) hardware platform. Due to the smaller capabilities of this hardware platform, the training regime was modified for the CIFAR dataset; instead of using six-fold validation, we used a single 70-30 split on the data. This modification allows the training to be completed on this AMD hardware without any memory overflow. Apart from this modification, the same training regimen, architectures, and hyperparameters as described in Section 4 were used in these experiments.

The experimental data was processed in the same manner as in Section 4.1, obtaining the following results:

The results from the data collected are similar to the ones presented in Section 5.

Table 16 shows that our results over the MNIST dataset and CIFAR dataset, for both neural architectures, across both hardware manufacturers seem consistent, i.e., they follow a similar trend and clearly show that the LeNet architecture is more efficient overall than the BCNN architecture, similar to Section 5.3.

Figures 12 and Figure 13 follow along the analysis presented in Section 5.2, with Figure 14 and Figure 15, following a similar trend. These results validate that the Efficiency reported and the analysis presented are consistent across hardware platforms.

Acknowledgements This work was conducted with the financial support of the Science Foundation Ireland Centre for Research Training in Digitally-Enhanced Reality (d-real) under Grant No. 18/CRT/6224.

Declarations

Competing interests The authors declare no competing interests.

References

AI O AI and compute. https://openai.com/index/ai-and-compute/

Anil R, Gupta V, Koren T, Regan K, Singer Y. (2020) Scalable Second Order Optimization for Deep Learning. arXiv. Version Number: 2 https://doi.org/10.48550/ARXIV.2002.09018 . https://arxiv.org/abs/2002.09018

Bartoldson BR, Kailkhura B, Blalock D (2023) Compute-efficient deep learning: algorithmic trends and opportunities. J Mach Learn Res 24(122):1–77

Bender EM, Gebru T, McMillan-Major A, Shmitchell S (2021) On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? In: Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency, pp. 610–623. ACM, Virtual Event Canada https://doi.org/10.1145/3442188.3445922

Cai H, Gan C, Wang T, Zhang Z, Han S (2019) Once-for-All: Train One Network and Specialize it for Efficient Deployment. arXiv. Version Number: 5 https://doi.org/10.48550/ARXIV.1908.09791 . https://arxiv.org/abs/1908.09791

Canziani A, Paszke A, Culurciello E (2016) An Analysis of Deep Neural Network Models for Practical Applications. arXiv. Version Number: 4 https://doi.org/10.48550/ARXIV.1605.07678 . https://arxiv.org/abs/1605.07678

Charnock T, Perreault-Levasseur L, Lanusse F (2020) Bayesian Neural Networks. In: Artificial Intelligence for High Energy Physics, pp. 663–713. WORLD SCIENTIFIC, ??? . https://doi.org/10.1142/9789811234033_0018 . https://www.worldscientific.com/doi/abs/10.1142/9789811234033_0018

Chen A, Dohan D, So D (2023) EvoPrompting: Language Models for Code-Level Neural Architecture Search. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems, vol. 36, pp. 7787–7817. Curran Associates, Inc., ??? https://proceedings.neurips.cc/paper_files/paper/2023/file/184c1e18d00d7752805324da48ad25be-Paper-Conference.pdf

Chen X, Liang C, Huang D, Real E, Wang K, Pham H, Dong X, Luong T, Hsieh C-J, Lu Y, Le QV (2023) Symbolic Discovery of Optimization Algorithms. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems, vol. 36, pp. 49205–49233. Curran Associates, Inc., ??? https://proceedings.neurips.cc/paper_files/paper/2023/file/9a39b4925e35cf447ccba8757137d84f-Paper-Conference.pdf

Davis DM, Lucas RF, Wagenbreth G, Tran JJ, Agalsoff J, Gottschalk TD (2009) Flops per watt: Heterogeneous-computing's approach to dod imperatives. In: the Proceedings of the Interservice/Industry Simulation, Training and Education Conference, Orlando, Florida, USA, pp. 1–10

DeWeerdt S. (2020) The carbon footprint of artificial intelligence is growing https://www.anthropocenemagazine.org/2020/11/time-to-talk-about-carbon-footprint-artificial-intelligence

Desislavov R, Martínez-Plumed F, Herná¡ndez-Orallo J(2021) Compute and Energy Consumption Trends in Deep Learning Inferencehttps://doi.org/10.48550/ARXIV.2109.05472 . Publisher: arXiv Version Number: 2

Ding N, Tang Y, Han K, Xu C, Wang Y (2023) Network Expansion for Practical Training Acceleration, pp. 20269–20279 https://openaccess.thecvf.com/content/CVPR2023/html/Ding_Network_Expansion_for_Practical_Training_Acceleration_CVPR_202_paper.html

Eschenhagen R, Immer A, Turner R, Schneider F, Hennig P (2023) Kronecker-Factored Approximate Curvature for Modern Neural Network Architectures. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems, vol. 36, pp. 33624–33655. Curran Associates, Inc., ??? https://proceedings.neurips.cc/paper_files/paper/2023/file/6a6679e3d5b9f7d5f09cdb79a5fc3fd8-Paper-Conference.pdf

Frankle J, Carbin M (2018) The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks https://doi.org/10.48550/ARXIV.1803.03635 . Publisher: arXiv Version Number: 5

Gal Y, Ghahramani Z (2015) Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference. arXiv. Version Number: 6 https://doi.org/10.48550/ARXIV.1506.02158 . https://arxiv.org/abs/1506.02158

Goldfarb D, Ren Y, Bahamou A (2020) Practical Quasi-Newton Methods for Training Deep Neural Networks. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems, vol. 33, pp. 2386–2396. Curran Associates, Inc., ??? https://proceedings.neurips.cc/paper_files/paper/2020/file/192fc044e74dffea144f9ac5dc9f3395-Paper.pdf

Gong L, He D, Li Z, Qin T, Wang L, Liu T (2019) Efficient Training of BERT by Progressively Stacking. In: Proceedings of the 36th International Conference on Machine Learning, pp. 2337–2346. PMLR, ??? ISSN: 2640-3498. https://proceedings.mlr.press/v97/gong19a.html

Han S, Pool J, Tran J, Dally W (2015) Learning both weights and connections for efficient neural network. Advances in neural information processing systems **28**

Horowitz M (2014) 1.1 Computing's energy problem (and what we can do about it). In: 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), pp. 10–14. IEEE, San Francisco, CA, USA (2014). https://doi.org/10.1109/ISSCC.2014.6757323 . http://ieeexplore.ieee.org/document/6757323/

Huang Y, Cheng Y, Bapna A, Firat O, Chen D, Chen M, Lee H, Ngiam J, Le QV, Wu Y, Chen z (2019) GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. In: Wallach, H., Larochelle, H., Beygelzimer, A., AlchÃ-Buc, F.d., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 32. Curran Associates, Inc., ??? . https://proceedings.neurips.cc/paper_files/paper/2019/file/093f65e080a295f8076b1c5722a46aa2-Paper.pdf

Jiang AH, Wong DL-K, Zhou G, Andersen DG, Dean J, Ganger GR, Joshi G, Kaminksy M, Kozuch M, Lipton ZC, Pillai P (2019) Accelerating Deep Learning by Focusing on the Biggest Losers. arXiv. Version Number: 1 https://doi.org/10.48550/ARXIV.1910.00762 . https://arxiv.org/abs/1910.00762

Jurj SL, Opritoiu F, Vladutiu M (2020) Environmentally-Friendly Metrics for Evaluating the Performance of Deep Learning Models and Systems. In: Yang, H., Pasupa, K., Leung, A.C.-S., Kwok, J.T., Chan, J.H., King, I. (eds.) Neural Information Processing, pp. 232–244. Springer, Cham https://doi.org/10.1007/978-3-030-63836-8_20

Kaddour J, Key O, Nawrot P, Minervini P, Kusner MJ (2023) No Train No Gain: Revisiting Efficient Training Algorithms For Transformer-based Language Models. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems, vol. 36, pp. 25793–25818. Curran Associates, Inc., ??? https://proceedings.neurips.cc/paper_files/paper/2023/file/51f3d6252706100325ddc435ba0ade0e-Paper-Conference.pdf

Kelleher JD (2019) Deep Learning. MIT Press, ??? Google-Books-ID: b06qDwAAQBAJ

Kelleher JD, Namee BM, D'Arcy, A (2020) Fundamentals of Machine Learning for Predictive Data Analytics, Second Edition: Algorithms, Worked Examples, and Case Studies. MIT Press, ??? . Google-Books-ID: UM_tDwAAQBAJ

Kingma DP, Ba J (2014) Adam: A Method for Stochastic Optimization. arXiv. Version Number: 9 https://doi.org/10.48550/ARXIV.1412.6980 . https://arxiv.org/abs/1412.6980

Kitaev N, Kaiser L, Levskaya A (2020) Reformer: The Efficient Transformer. arXiv. Version Number: 2 https://doi.org/10.48550/ARXIV.2001.04451 . https://arxiv.org/abs/2001.04451

Krizhevsky A. Learning Multiple Layers of Features from Tiny Images

LeCun Y, Cortes C, Burges C MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. https://yann.lecun.com/exdb/mnist/

Lecun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proceed IEEE 86(11):2278–2324

Li T, John LK (2003) Run-time modeling and estimation of operating system power consumption. In: Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp. 160–171. ACM, San Diego CA USA https://doi.org/10.1145/781027.781048

Li D, Chen X, Becchi M, Zong Z (2016) Evaluating the Energy Efficiency of Deep Convolutional Neural Networks on CPUs and GPUs. In: 2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom), pp. 477–484 . https://doi.org/10.1109/BDCloud-SocialCom-SustainCom.2016.76 . https://ieeexplore.ieee.org/abstract/document/7723730

Li Z, Wallace E, Shen S, Lin K, Keutzer K, Klein D, Gonzalez J (2020) Train Big, Then Compress: Rethinking Model Size for Efficient Training and Inference of Transformers. In: Proceedings of the 37th International Conference on Machine Learning, pp. 5958–5968. PMLR, ??? ISSN: 2640-3498. https://proceedings.mlr.press/v119/li20m.html

Liu H, Li Z, Hall D, Liang P, Ma T (2023) Sophia: A Scalable Stochastic Second-order Optimizer for Language Model Pre-training. arXiv. Version Number: 4 https://doi.org/10.48550/ARXIV.2305.14342 . https://arxiv.org/abs/2305.14342

Liu Z, Li J, Shen Z, Huang G, Yan S, Zhang C (2017) Learning Efficient Convolutional Networks Through Network Slimming, pp. 2736–2744 https://openaccess.thecvf.com/content_iccv_2017/html/Liu_Learning_Efficient_Convolutional_ICCV_2017_paper.html

Lu Z, Cheng R, Jin Y, Tan KC, Deb K (2024) Neural architecture search as multiobjective optimization benchmarks: problem formulation and performance assessment. IEEE Trans Evol Comput 28(2):323–337. https://doi.org/10.1109/TEVC.2022.3233364

MacKay DJC (1995) Bayesian neural networks and density networks. Nucl Inst Methods Phys Res Sect A: Accel Spectrom Detect Assoc Equip 354(1):73–80. https://doi.org/10.1016/0168-9002(94)00931-7

Micikevicius P, Narang S, Alben J, Diamos G, Elsen E, Garcia D, Ginsburg B, Houston M, Kuchaiev O, Venkatesh G, Wu H (2017) Mixed Precision Training. arXiv. Version Number: 3 . https://doi.org/10.48550/ARXIV.1710.03740

Mills KG, Han FX, Zhang J, Changiz Rezaei SS, Chudak F, Lu W, Lian S, Jui S, Niu D (2021) Profiling Neural Blocks and Design Spaces for Mobile Neural Architecture Search. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management. CIKM '21, pp. 4026–4035. Association for Computing Machinery, New York, NY, USA . https://doi.org/10.1145/3459637.3481944

Mindermann S, Brauner JM, Razzak MT, Sharma M, Kirsch A, Xu W, Höltgen B, Gomez AN, Morisot A, Farquhar S, Gal Y (2022) Prioritized Training on Points that are Learnable, Worth Learning, and not yet Learnt. In: Proceedings of the 39th International Conference on Machine Learning, pp. 15630–15649. PMLR, ??? ISSN: 2640-3498. https://proceedings.mlr.press/v162/mindermann22a.html

Nakkiran P, Kaplun G, Bansal Y, Yang T, Barak B (2021) Sutskever I (2021) deep double descent: where bigger models and more data hurt*. J Stat Mech Theory Exp 12:124003. https://doi.org/10.1088/1742-5468/ac3a74

Pan Y, Yuan Y, Yin Y, Xu Z, Shang L, Jiang X, Liu Q (2023) Reusing Pretrained Models by Multi-linear Operators for Efficient Training. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems, vol. 36, pp. 3248–3262. Curran Associates, Inc., ??? https://proceedings.neurips.cc/paper_files/paper/2023/file/09d9a13f7018110cfb439c06b07940a2-Paper-Conference.pdf

Sanh V, Debut L, Chaumond J, Wolf T (2019) DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv. Version Number: 4 https://doi.org/10.48550/ARXIV.1910.01108 . https://arxiv.org/abs/1910.01108

Schwartz R, Dodge J, Smith NA, Etzioni O (2020) Green AI. Commun ACM 63(12):54–63. https://doi.org/10.1145/3381831

Strubell E, Ganesh A, McCallum A (2020) Energy and policy considerations for modern deep learning research. Proceed AAAI Conf Artif Intell 34(09):13693–13696. https://doi.org/10.1609/aaai.v34i09.7123

Sze V, Chen Y-H, Yang T-J, Emer JS (2020) How to evaluate deep neural network processors: TOPS/W (alone) considered harmful. IEEE Solid-State Circ Mag 12(3):28–41. https://doi.org/10.1109/MSSC.2020.3002140

Sze V, Chen YH, Yang TJ, Emer JS (2020) Efficient processing of deep neural networks. Synths Lect Comput Archit 15(2):1–341. https://doi.org/10.2200/S01004ED1V01Y202004CAC050

Tan M, Le QV (2019) EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks https://doi.org/10.48550/ARXIV.1905.11946 . Publisher: arXiv Version Number: 5

Tan M, Le Q (2021) EfficientNetV2: Smaller Models and Faster Training. In: Proceedings of the 38th International Conference on Machine Learning, pp. 10096–10106. PMLR, ??? ISSN: 2640-3498. https://proceedings.mlr.press/v139/tan21a.html

Tay Y, Dehghani M, Bahri D, Metzler D (2020) Efficient Transformers: A Survey. arXiv. Version Number: 3 https://doi.org/10.48550/ARXIV.2009.06732

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention Is All You Need. arXiv. Version Number: 7 https://doi.org/10.48550/ARXIV.1706.03762

Vinuesa R, Azizpour H, Leite I, Balaam M, Dignum V, Domisch S, Felländer A, Langhans SD, Tegmark M, Fuso Nerini F (2020) The role of artificial intelligence in achieving the sustainable development goals. Nat Commun 11(1):233. https://doi.org/10.1038/s41467-019-14108-y

Wang S, Li BZ, Khabsa M, Fang H, Ma H (2020) Linformer: Self-Attention with Linear Complexity. arXiv. Version Number: 3 https://doi.org/10.48550/ARXIV.2006.04768

Wang Y, Yue Y, Lu R, Liu T, Zhong Z, Song S, Huang G (2023) EfficientTrain: Exploring Generalized Curriculum Learning for Training Visual Backbones, pp. 5852–5864 https://openaccess.thecvf.com/content/ICCV2023/html/Wang_EfficientTrain_Exploring_Generalized_Curriculum_Learning_for_Training_Visual_Backbones_ICCV_2023_paper.html

Wang Y, Yue Y, Lu R, Han Y, Song S, Huang G (2024) EfficientTrain++: Generalized Curriculum Learning for Efficient Visual Backbone Training. IEEE Transactions on Pattern Analysis and Machine Intelligence 1–18. https://doi.org/10.1109/TPAMI.2024.3401036

White C, Safari M, Sukthanker R, Ru B, Elsken T, Zela A, Dey D, Hutter F (2023) Neural Architecture Search: Insights from 1000 Papers. arXiv. Version Number: 2 https://doi.org/10.48550/ARXIV.2301.08727 . https://arxiv.org/abs/2301.08727

Wynsberghe A (2021) Sustainable AI: AI for sustainability and the sustainability of AI. AI Ethics 1(3):213–218. https://doi.org/10.1007/s43681-021-00043-6

Xie SM, Santurkar S, Ma T, Liang PS (2023) Data Selection for Language Models via Importance Resampling. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems, vol. 36, pp. 34201–34227. Curran Associates, Inc., ??? https://proceedings.neurips.cc/paper_files/paper/2023/file/6b9aa8f418bde2840d5f4ab7a02f663b-Paper-Conference.pdf

Yang T-J, Howard A, Chen B, Zhang X, Go A, Sandler M, Sze V, Adam H (2018) NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications, pp. 285–300 https://openaccess.thecvf.com/content_ECCV_2018/html/Tien-Ju_Yang_NetAdapt_Platform-Aware_Neural_ECCV_2018_paper.html

Yang Y, Kang H, Mirzasoleiman B (2023) Towards Sustainable Learning: Coresets for Data-efficient Deep Learning. In: Proceedings of the 40th International Conference on Machine Learning, pp. 39314–39330. PMLR, ??? ISSN: 2640-3498. https://proceedings.mlr.press/v202/yang23g.html

Ying H, Song M, Tang Y, Xiao S, Xiao Z (2024) Enhancing deep neural network training efficiency and performance through linear prediction. Sci Rep 14(1):15197. https://doi.org/10.1038/s41598-024-65691-0

Zeng S, Sun H, Xing Y, Ning X, Shan Y, Chen X, Wang Y, Yang H (2020) Black Box Search Space Profiling for Accelerator-Aware Neural Architecture Search. In: 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 518–523. IEEE, Beijing, China https://doi.org/10.1109/ASP-DAC47756.2020.9045179

Zhang M, He Y (2020) Accelerating Training of Transformer-Based Language Models with Progressive Layer Dropping. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems, vol. 33, pp. 14011–14023. Curran Associates, Inc., ??? https://proceedings.neurips.cc/paper_files/paper/2020/file/a1140a3d0df1c81e24ae954d935e8926-Paper.pdf

Zheng N, Mazumder P (2019) Learning in Energy-Efficient Neuromorphic Computing: Algorithm and Architecture Co-Design. John Wiley & Sons, ??? . Google-Books-ID: IvC0DwAAQBAJ

Zhou R, Quan P (2023) Optimization ways in neural network compression. Procedia Comput Sci 221:1351–1357. https://doi.org/10.1016/j.procs.2023.08.125