Green AI

An empirical study

T.E.R. Yarally

Software Engineering Research Group (SERG)
Delft University of Technology
Master Computer Science

Green AI

An empirical study

by

T.E.R. Yarally

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday July 1, 2022 at 10:00 AM.

Student number:	4586107	
Project duration:	November 1, 2021 – July 1, 2022	
Thesis committee:	Prof. dr. A. Van Deursen	TU Delft
	L. Miranda da Cruz	TU Delft
	M. Weinmann	TU Delft
Supervision:	L. Miranda da Cruz	TU Delft
	D. Feitosa	RU Groningen

This thesis is confidential and cannot be made public until July 1, 2022.

TU Delft

Abstract

In this work, we look at the intersection of Sustainable Software Engineering and AI engineering known as Green AI. AI computing is rapidly becoming more expensive, calling for a change in design philosophy. We consider both training and inference of neural networks used for image vision; to reveal energy-efficient practices in an exploratory fashion.

First of all, we examine a modern algorithm for hyperparameter optimisation and compare this to two baseline methods. We find that the baseline algorithms perform considerably worse despite their wide usage and argue that they should not be used when training large models. Furthermore, we look at the layer structure of convolutional networks and conclude that the convolutional layers have the largest influence on the total consumption. We report increases of up to 95% with only marginal improvements in accuracy. Therefore we recommend developers to reduce their network architectures as long as the performance stays within a reasonable margin.

Second, we present a study focused on the inference phase of the deep learning pipeline. We look at the effect of batching for image classification requests. To facilitate the data collection, we make use of a simulated queue and the Pytorch framework. We find that batching has a significant impact on the energy consumption, but the magnitude of this impact can vary a lot for different models. Our recommendation is to treat the batch size as an inference parameter that needs to be tuned first. Additionally, we highlight how the energy consumption of image vision networks has evolved over the past decade. Presenting the findings together with the performance of these networks shows a steady, upward energy trend accompanied by a decreasing slope for the accuracy. The only exception is the model ShuffleNetV2. We mention the design principles that went into the development of this network and present it as a start for future research.

Preface

To Whom It May Concern:

For the time being, I hereby present my final academic contribution. As a temporary member of the Software Engineering Research Group, I started my thesis on the topic of Green AI, in November of 2021. During the following eight months, I would be working towards this finished product with the goal of graduating from the Computer Science Master at Delft's University of Technology.

When I first met Luìs, who became my weekly supervisor and does research in the fields of AI engineering and sustainable software engineering, I did not have a great deal of knowledge on either of those topics. For the first couple of weeks, I would work through the available literature while considering different research directions for my study. My initial indifference towards this topic, naturally shifted to fascination as I discovered the many research opportunities and how much appreciation there is for this field.

The game that we play in the field of Green AI does not only involve accumulating favourable results to present to the scientific community. We must also try to change the rigid habits and mindsets that have dominated AI engineering for the longest time. To open up this conversation and look at previously published results through a different set of glasses, is also a responsibility of Green AI engineers, and I find these methods to be quite charming.

Writing a thesis during the pandemic is nothing short of difficult. Most of the work is done from home and all communication is online. There are also many uncertainties: The research you perform is novel and the way ahead has not been paved for you. Despite all of this, I was in the fortunate position of being surrounded by two amazing supervisors. I want to thank Luìs Miranda da Cruz and Daniel Feitosa for their continuous support, insights and encouragement. Our meetings really helped me get through these months. Arie van Deursen also deserves some special thanks for his flexibility regardless of his busy schedule. Thank you for signing my documents almost literally on the fly and listening to my presentation when you did not even know I would be attending that meeting. I would also like to mention Michael Weinmann. When I was desperately trying to find a second assessor for my thesis committee, you swiftly responded on a Sunday morning of all days. It may seem small, but that relieved me from a lot of stress.

Finally, I want to thank my mother, who has been fighting a very unfair battle. Even when most would succumb to their own sadness and anger, you never let it get to you. You cry, you think, you move on. In your condition, to be able to say, "There are people who have it worse", that is the level of strength that I strive to reach. Knowing this, I could not allow myself to feel disheartened or bitter. Your positivity fuels mine and together we persevere.

Yours faithfully,

T.E.R. Yarally
Delft, June 2022

Contents

1 Introduction

When we think about saving energy, trivial matters come to mind first: turn off the lights when the sun is out, wash your clothes with cold water and take shorter showers. We like to think that every small bit helps, but in reality, the consumption of the individual is often overshadowed by that of large corporations. In 2020, Microsoft announced a new supercomputer in collaboration with OpenAI. The build was a single system containing 10,000 GPUs[1]. An average European household consumes 1.3 toe (tonne of oil equivalent) of energy in a year[2], which is around 15,000 kWh. Solely considering the GPUs used in the supercomputer and assuming that they are V100s with a lower bound power usage of 250 watts, the estimated energy cost of this system adds up to 21,900,000 kWh or just over 1,450 households.

AI practices are expensive and can have a significant environmental impact. That is not surprising since an important challenge within the AI community is improving the accuracy of previously reported systems [1]. Now, a new field is emerging to address this problem: Green AI, with its roots planted deep into the discipline of Sustainable Software Engineering. The software engineering community has increasingly studied the energy efficiency of software systems by developing energy estimation models [2], [3]; developing code analysis and optimisation tools to improve energy efficiency [4]–[7]; studying practices that lead to green software [8]–[10] and so on. Recently, a new trend is calling for software engineering approaches that consider 'data as the new code', challenging practitioners with new software systems that ship AI-based features. This intersection between Green Software Engineering and AI Engineering is where we find the origin of Green AI. The initial contributions in this field consist of positional papers that are calling for a new research agenda [1], [11], [12]. This involves the measurement and reporting of energy consumption next to accuracy, but also the appreciation of research efforts that do not necessarily rely on enterprise-sized data or training budgets.

This thesis covers two separate studies that focus on different parts of the deep learning pipeline. The first study in Chapter 3, involves a thorough analysis of efficient practices during the training phase. We look at baseline algorithms for hyperparameter optimisation and quantify how much energy could be spared if we use more sophisticated algorithms to lower the required amount of training epochs. Additionally, we examine the energy efficiency of different network architectures and perform a trade-off analysis with the accuracy of those networks. For the second study, as presented in Chapter 4, the attention is shifted to the inference phase. Here we consider a more practical setting. We look at the effect of different batching strategies during inference and perform a trade-off analysis between energy consumption and response time. These studies have a series of implications that are relevant to various stakeholders. We first highlight these implications in Chapter 5 and then expand to discuss opportunities for further research in Chapter 6. Finally, we present our general conclusion in Chapter 7.

[1] https://blogs.microsoft.com/ai/openai-azure-supercomputer/
[2] https://www.enerdata.net/publications/executive-briefing/households-energy-efficiency.html

2

Related Work

There is a general consensus that developing energy-efficient software systems is far from trivial [13], [14]. Hence, the discipline of green software engineering has long been studying how to realise this. Previous work proposes a catalogue of design patterns aiming to reduce the energy consumption of mobile applications [8]. The authors mine 1,700+ mobile applications to collect code changes introduced by developers to improve energy efficiency. The code changes are then studied and grouped in different patterns, resulting in a catalogue of 22 energy patterns. However, since the study is targeted at mobile applications, it is not clear which of these patterns would fit an AI system.

Other works aim at comparing the energy consumption of different development environments. Pereira et al. showcase the energy consumption of 27 different programming languages [15]. They use different benchmarks for validation and compare results with memory efficiency and time efficiency. Overall, programming languages such as C, Rust, Ada and C++ consistently yield high energy efficiency. In contrast, Python – one of the most popular programming languages in AI projects – scored very low when compared to other modern languages. However, the benchmark used in this study is not representative of common AI systems which typically rely on API calls to fully optimised frameworks. Hence, most energy consumption in AI is likely to stem from external libraries and frameworks that are not necessarily written in the same language of the system – e.g. NVIDIA Cuda python.

Within the green software community, AI has been used to provide more accurate estimations models of software energy consumption [2]. In their work, Chowdbury et al. found that CPU usage is a good proxy for energy consumption in software applications. While AI is a powerful tool to address energy efficiency in traditional software systems, we are interested in the opposite: we apply an empirical software engineering approach to address energy efficiency in AI systems. Moreover, it is not clear to what extent previous results still apply to AI software.

Contributions to green software have also scoped a more holistic approach within software organisations. Hankel et al. have studied the factors influencing the adoption of green processes within ICT organisations [16]. Verdecchia et al. propose short-term solutions that lead to an impact on energy efficiency [17] – examples include opting for green energy sources and utilising AI-specific hardware. Our work addresses green technologies at a lower level of abstraction – for instance, by addressing energy efficiency at the level of designing the AI pipeline.

Given the particularities of different types of software systems, green software contributions span across multiple sub-fields of computer science: mobile computing [2], [8], Web [18], robotics [19], and so on. In our work, we challenge the green software engineering field to expand to AI systems. To the best of our knowledge, related research in Green AI is still preliminary and does not yet follow the scientific method that drives the research in green software. We pinpoint below the most relevant related contributions in Green AI.

Schwartz et al. [1] present an elegant introductory article into this field of research. The authors introduce two novel terms to guide future conversations: Green AI, which refers to AI research that considers computational cost as a primary metric next to accuracy; and Red AI, the most common form of AI research that seeks to improve accuracy without any regards for the computational resources required. Ultimately, Schwartz et al. call for a research agenda that aims to reduce carbon emissions and make the deep learning field more accessible to everyone. Our work takes a preliminary step

towards this goal, by presenting empirical results focused on different parts of the AI pipeline that can lead to energy-efficiency gains on a larger scale.

Strubell et al. [12] look into the quantity of energy consumption in the domain of Natural Language Processing (NLP). The authors present preliminary results showing that the accuracy of trained NLP models has improved substantially at the expense of a serious amount of energy. Our study aims to provide scientifically proven advice to help design energy-efficient AI systems, including NLP.

Li, Chen et al. [20] address a similar problem specifically targeted toward applications of convolutional neural networks (CNNs). In their work, the authors compare a set of well-known CNN models in terms of energy efficiency. They also assess to what degree different types of layers contribute to the overall energy consumption. As such, they provide percentages for the convolutional layers, fully connected (or linear) layers, pooling layers and ReLU layers. Apart from finding the energy efficiency of these layer types, our study also includes a trade-off analysis where we compare changes in energy consumption to those in accuracy.

Yang et al. [21] propose a new pruning method that they named energy-aware pruning. They report a reduction in the overall consumption by a factor between 1.6x and 3.7x, with an insignificant loss in accuracy. Our approach to optimise the network architecture is much more coarse-grained compared to the techniques described in this paper. Rather than focusing on fine-tuning the weights inside the layers, we investigate the factor of redundancy of those layers and provide advice that is more relevant from a design-level perspective. Both philosophies could be used together.

More in the same theme as the second study presented in this thesis, Canziani et al. [22] present an analysis of different deep learning metrics in a practical setting. Amongst other results, the authors compare the relation between batch size and power consumption of state-of-the-art models. The study reports that no such relation could be found. Upon closer inspection, however, we notice that for most neural networks, there are no data points for batch sizes larger than 16. Furthermore, the method for determining the power consumption likely considers the system as a whole rather than just the GPU. In our study, we attempt to correct for these shortcomings. We perform experimentation with a larger range of batch sizes and, because image classification tasks are mostly delegated to the graphics card, we focus solely on the power consumption of the GPU.

3

First study - Efficient practices during training

3.1. Introduction

This study focuses on deep learning, a subset of machine learning and the driver behind many AI applications and services. All experiments are performed with rudimentary neural networks that comprise the building blocks of more complex models. We train these networks on two popular image vision problem sets: FashionMNIST [23] and CIFAR-10 [24]. We adopt the idea of designing neural networks with energy consumption as one of the main metrics. Specifically, we direct our attention to the early phases of the deep learning pipeline and formulate the following research questions:

1. RQ_1^{α}: Between Bayesian optimisation, random optimisation and grid search; which strategy is the most energy-efficient for training a neural network?

2. RQ_2^{α} Can the complexity of a neural network be reduced such that it consumes less energy while maintaining an acceptable level of accuracy?

First, we analyse Bayesian optimisation, random optimisation and grid search, three popular optimisation strategies, to identify best practices in terms of energy efficiency. Classically, grid search has served as the most popular baseline optimisation strategy in the context of hyperparameter tuning [25]. Nonetheless, there have been studies that present random search as an alternative baseline that competes with or even exceeds grid search in multi-dimensional optimisation problems [25]–[27]. Bayesian optimisation is a more powerful strategy that is also more difficult to implement and parallelise. Apart from comparing these three strategies, we demonstrate that further optimisation attempts past a specific point are met with diminishing returns in performance that might not be worth the additional cost of training. Training times can vary greatly depending on the workload and network architecture and there are no rules that state how many optimisation rounds one should perform. This is where the potential opportunity for energy savings lies.

Second, we quantify the effect of a network's architecture in terms of layers, on the actual energy consumption of the GPU. In a similar fashion, we show how more complex models see diminishing returns in their performance, while the energy consumption keeps increasing at a steady rate. By analysing the accuracy of a neural network together with its energy consumption, the perspective of what is currently considered 'the best' model could see a dramatic shift. We believe it is the job of the Green AI community to make such data and observations available to the public so that software engineers can make more informed trade-offs, and more sustainable decisions.

The concepts mentioned throughout this paper are discussed in Section 3.2. The methodology and resources necessary for reproduction purposes are presented in Section 3.3. Section 3.4 details the design of two experiments that were constructed around the research questions. Section 3.5 collects and parses all the results which we then discuss in Section 3.6. Finally, we elaborate on the threats to the validity of our study in Section 3.7 and finish with the summary in Section 3.8.

3.2. Background

This section introduces the FashionMNIST and CIFAR-10 datasets; elaborates on grid search, random search and Bayesian optimisation and establishes a basic understanding of linear, convolutional and ReLU layers inside a neural network.

3.2.1. Datasets

The original MNIST dataset consists of many grey-scale images of handwritten digits. MNIST has been used excessively as a benchmark to validate many different models. However, with modern technology, the MNIST problem set has become too trivial. Because most networks can achieve near-perfect accuracy on the set, researchers from Zalando have proposed the use of **FashionMNIST** as a direct drop-in-replacement [23]. As such, the dataset is comprised of 28×28 grayscale images of 70,000 different fashion products. Just like in the original MNIST set, the products are separated into 10 categories. Because both datasets are shaped identically, FashionMNIST is immediately compatible with any machine learning package that works with MNIST.

The **CIFAR-10** dataset is a subset of the tiny images dataset [28]. It is composed of 60,000 32×32 RGB images divided into 10 classes [24]. CIFAR-10 presents a challenge that is very similar to FashionMNIST, however, the larger image size and two additional layers increase the complexity of the task significantly.

3.2.2. Optimisation Strategies

Grid search is a traditional optimisation strategy that applies an exhaustive search over the hyperparameter space. For discrete variables, this means that the algorithm considers the Cartesian product of all the values. For continuous variables, it is necessary to select a distribution first. One could for example choose a uniform or log-uniform distribution to map the continuous space to a discrete one. The computational complexity of grid search is exponential in the number of parameters, therefore it quickly becomes impractical to calculate it all the way through. Nevertheless, because the search space is determined at the beginning, the workload can very easily be parallelised, somewhat offsetting this drawback.

In a **random search**, the well-defined structure of the grid is replaced by random selection. Because every drawn sample is completely independent, parallelisation of this algorithm is as trivial as with grid search.

In the context of hyperparameter tuning, the **Bayesian optimisation** algorithm creates and refines a probabilistic regression model of a function $f(x)$ that can be exploited to return the predicted accuracy and corresponding standard deviation. An acquisition function is then used to determine the next most promising set of input variables. For the probabilistic model, Gaussian Processes (GP) are the most popular choice amongst many studies [29]–[31]. Bayesian optimisation is especially effective in scenarios where the true value of $f(x)$ is hard to compute, which is the case with neural network training. This is because the probabilistic model needs to evaluate a sufficiently large quantity of samples.

The purpose of the acquisition function is to determine the most promising sample from a set of randomly selected input variables. There are many possible choices that can be considered:

- Probability of Improvement (PI) [32]
- Expected Improvement (EI) [33]
- Upper Confidence Bound (UCB) [34]
- Entropy Search (ES) [35]
- Predictive Entropy Search (PES) [36]
- Knowledge Gradient (KG) [37]

We now elaborate on the PI acquisition function, which is also the function that we use in all of the experiments.

$$PI(x) = P(f(x) \geq f(x_{best})) = \Phi(\frac{\mu(x) - f(x_{best})}{\sigma(x) + \epsilon}) \tag{3.1}$$

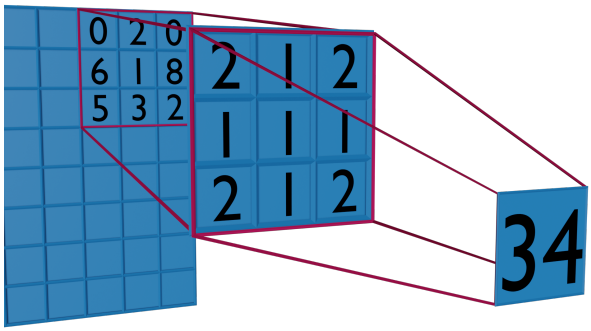


Figure 3.1: Convolution operation on an input layer using a 3×3 kernel.

$$PI(x) = P(f(x) \leq f(x_{best})) = \Phi(\frac{f(x_{best}) - \mu(x)}{\sigma(x) + \epsilon}) \tag{3.2}$$

Equations 3.1 and 3.2 are used to calculate the probability of improvement for maximisation and minimisation problems respectively. Since we are interested in maximising the accuracy of a neural network, we will use equation 3.1. Here, Φ refers to the cumulative density function of a normal distribution; μ and σ are the predicted value and standard deviation retrieved from the Gaussian regressor, and $f(x_{best})$ is the highest actual accuracy found so far. Algorithm 1 displays an example implementation of a single PI iteration.

Algorithm 1 Bayesian Optimisation - Probability of Improvement

$y = max(Y)$
$Candidates \leftarrow N$ random input samples
x', pi'
for $x \in Candidates$ **do**
  $\mu, \sigma = predict(x)$
  $pi = \Phi(\frac{\mu - y}{\sigma + \epsilon})$
  if $pi > pi'$ **then**
    $x' = x, pi' = pi$
  end if
end for
$X \leftarrow x', Y \leftarrow f(x')$
$fit(X, Y)$

3.2.3. Neural Network Layers

Every layer inside a neural network performs some transformation on an input vector x. The obtained output is then passed on to the next layer. **Linear**, or **fully connected layers**, calculate an output by applying a linear transformation through a matrix of weights W [38]. The values of W are optimised and updated during training. The term fully connected comes from the fact that every element of x is mapped to every other element in the output by the matrix multiplication $W^T x$.

Inside a **convolutional layer**, a kernel is used to calculate a weighted summation of the elements of the input layer. The kernel slides across the input layer, considering all elements and their neighbours. A convolutional operation is defined by stride, kernel size and zero padding[39]. The stride determines how many places the kernel slides after each calculation; the kernel size represents the dimensions of the filter and zero padding adds zero's to the outer edges of the input layer. Generally speaking, the output layer is always smaller than the input layer, limiting the maximum number of convolutional layers that can be implemented. However, by applying zero padding, one can prevent this shrinking behaviour if desired. The convolution operation is shown graphically in Figure 3.1.

The **Rectified Linear Unit (ReLU) layer** introduces an activation function that applies non-linearity to the input. ReLU is the most common form of non-linearity in CNN's [39]. The function is very simple: An element is deactivated (set to 0) if it is negative, otherwise the value remains the same.

3.3. Research Methods

The goal of this study is to identify trade-off points with regard to the energy consumption during the training phase of the deep learning pipeline.

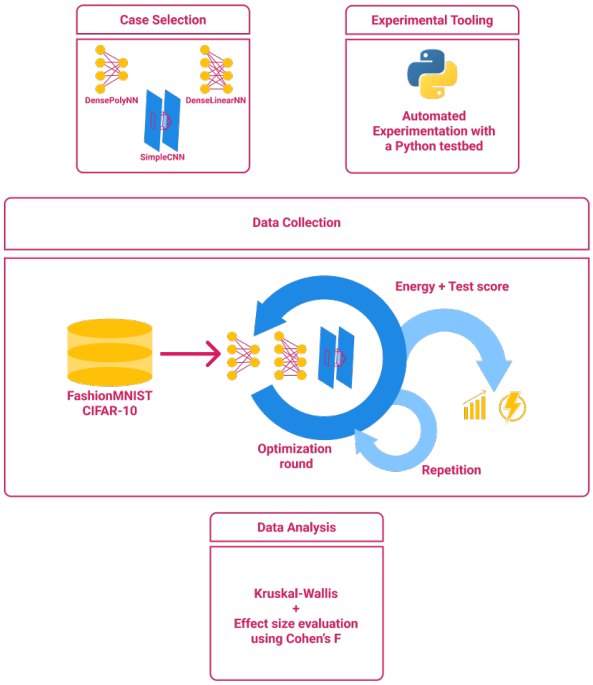


Figure 3.2: Diagram of the methodology

3.3.1. Case Selection

Achieving state-of-the-art accuracy results on challenging data sets is not the main focus. For this reason, we will be working with rudimentary networks architectures that can be trained using consumer-grade hardware. The simplicity of the models facilitates the design of more intricate experimentation and encourages inclusivity. We choose to direct our efforts to image recognition problems. Image recognition is a canonical problem that can be solved with neural networks and there are a plethora of easily accessible data sets available.

All experiments are performed on three neural networks written with the Pytorch framework[1], which are trained on a single GeForce GTX-1080[2] GPU. During every optimisation round mentioned in this study, an optimisation algorithm chooses a set of hyper-parameter values. An optimisation round lasts for N repetitions during which we use the same configuration of hyperparameters. After the N repetitions, a new optimisation round starts. For the actual training, the networks undergo 25 training epochs[3]. The structure of the different networks is as follows:

- **DenseLinearNN**: N linear layers where the number of neurons in each layer scales down linearly towards the number of problem classes.
- **DensePolyNN**: N linear layers where every layer has half the number of neurons as the layer before it.
- **SimpleCNN**: M convolutional layers, each followed by a BatchNorm2d, ReLU and MaxPool2d layer, and N linear layers where every layer has half the number of neurons as the layer before it.

[1] https://pytorch.org
[2] https://www.nvidia.com/nl-nl/geforce/10-series/
[3] During one optimisation round with two repetitions, a network undergoes $25 + 25$ training epochs with the same set of hyper-parameter values

3.3.2. Experimental Tooling

To facilitate and standardise the data collection, we develop a test suite that automates the execution of the experiments. This test suite is available online[4] and contains the implementations of the aforementioned neural networks that can be trained on all the visual problem datasets provided by Pytorch[5]. The testbed is designed to be modular and we encourage other researchers to add additional neural network designs or different hyperparameter optimisation functions. All of the results in this study have been accumulated with the aid of this testbed.

3.3.3. Data Collection

To answer RQ_1^{α}, we compare the convergence rate of three different hyperparameter tuning strategies: Grid search, random optimisation and Bayesian optimisation with the PI acquisition function. Because grid search is an exhaustive method, it quickly becomes infeasible to train a network on every configuration. To fairly compare grid search to the other strategies, we first generate the complete search space and then proceed to pick random samples from that space until we reach the desired amount of optimisation rounds. Variables with continuous ranges are divided into five uniformly-distributed values. Although this is a partial grid search, the most important difference with random search remains intact: because we select samples from the grid, a limited number of values are considered for every hyperparameter.

For RQ_2^{α}, we first examine the effect of the neural network's architecture on the absolute energy consumption. To obtain the power usage of the GPU, we query the NVIDIA System Management Interface[6] every 100 milliseconds. We use this to compute the total energy consumption of a training iteration and then factor out the idle energy consumption of the GPU.

3.3.4. Data Analysis

To accurately analyse the effect of the neural network architecture in relation to the energy consumption, first, we would like to show that the hyperparameter configuration does not contaminate the results. We do so by calculating the coefficient of variance (CV) of the energy consumption and showing that it is very low (< 0.01). The CV is calculated as the standard deviation of a sequence divided by its average.

Prior to any further in-depth analysis, we need to assess whether the energy results obtained in the experiments follow a normal distribution. After a visual inspection of the quantile-quantile (Q-Q) plot, followed by the Shapiro-Wilk test[7], we conclude that our data is not normally distributed. Hence, we opt for a non-parametric analysis and apply the Kruskal-Wallis test to indicate the significance of our independent variables, i.e. whether we may conclude that the layer types have a statistically meaningful impact on the energy consumption. Additionally, we calculate the η^2 as the effect size. We evaluate these effect sizes based on the rules of thumb for Cohen's f [40], which is calculated as $f = \sqrt{\frac{\eta^2}{1-\eta^2}}$. Cohen suggests that the values 0.10, 0.25 and 0.40 convey a small, medium and large effect size respectively. By finding the inverse of the function we obtain the effect thresholds for η^2: 0.01, 0.06 and 0.14.

3.4. Experiments

In this section, we present the design of two different experiments, each related to one of the research questions. Section 3.4.1 describes the experiment to compare the hyperparameter optimisation strategies. The second experiment, to investigate the relationship between neural network architectures and energy consumption, is described in Section 3.4.2.

3.4.1. Hyperparameter Optimisation

Given that the response function of a hyperparameter optimisation problem $f(x_1, ..., x_n)$ has a *low effective dimensionality* [25], meaning that the function can be approximated by another function $g(x_1, ..., x_{n-i})$ with less variables, the hypothesis for RQ_1 is that random search will converge faster than grid search, because it does not consider two identical values more than once. Given enough time, Bayesian op-

[4] https://anonymous.4open.science/r/green-ai-46E7
[5] https://pytorch.org/vision/0.8/datasets.html
[6] https://developer.nvidia.com/nvidia-system-management-interface
[7] https://www.statskingdom.com/shapiro-wilk-test-calculator.html

Table 3.1: Comparison of optimisation strategies.

Strategy	Network	hyperparameters
Bayesian	DensePolyNN	α, β_1, β_2
	DenseLinearNN	α, β_1, β_2
	SimpleCNN	α, β_1, β_2
	DensePolyNN	$\alpha, \beta_1, \beta_2, \epsilon, w$
	DenseLinearNN	$\alpha, \beta_1, \beta_2, \epsilon, w$
	SimpleCNN	$\alpha, \beta_1, \beta_2, \epsilon, w$
Random	DensePolyNN	α, β_1, β_2
	DenseLinearNN	α, β_1, β_2
	SimpleCNN	α, β_1, β_2
	DensePolyNN	$\alpha, \beta_1, \beta_2, \epsilon, w$
	DenseLinearNN	$\alpha, \beta_1, \beta_2, \epsilon, w$
	SimpleCNN	$\alpha, \beta_1, \beta_2, \epsilon, w$
Grid	DensePolyNN	α, β_1, β_2
	DenseLinearNN	α, β_1, β_2
	SimpleCNN	α, β_1, β_2
	DensePolyNN	$\alpha, \beta_1, \beta_2, \epsilon, w$
	DenseLinearNN	$\alpha, \beta_1, \beta_2, \epsilon, w$
	SimpleCNN	$\alpha, \beta_1, \beta_2, \epsilon, w$

timisation should outperform the other two strategies. However, with a limited run budget, we might observe that the Bayesian strategy performs worse because it chooses to exploit suboptimal solutions rather than explore better ones.

The setup of the experiment, as is depicted in Table 3.1, involves 18 different configurations. Each optimisation strategy is applied twice to the DensePolyNN, DenseLinearNN and SimpleCNN mentioned in section 3.3. The *hyperparameters* column in Table 3.1 shows how many parameters are optimised during a run. The five hyperparameters refer to the learning rate (α), beta's (β_1, β_2), epsilon (ϵ) and weight decay (w) of the ADAM optimiser provided by PyTorch[8]. The entire experiment is repeated for both the FashionMNIST and CIFAR-10 datasets.

For every row in Table 3.1, a network is trained on 64 different hyperparameter settings with 8 repetitions for each setting, amounting to 512 training iterations. After each set of repetitions, the optimisation function provides a new set of values for the hyperparameters. A trained model is evaluated and the results are logged. In total we run 18 configurations $\times$ 64 optimisation rounds $\times$ 8 repetitions $\times$ 2 data sets $=$ 18,432 training iterations.

3.4.2. Network Architecture

The second experiment aims to answer RQ_2 by collecting empirical data that shows the relation between the structure of a neural network and its energy consumption. We present a full factorial design in Table 3.2. The results of this experiment highlight the energy efficiency or lack thereof for the linear, convolutional and ReLU layers. The interesting point for discussion will be whether reducing the network complexity has a significant, positive influence on the energy efficiency, without too heavily compromising on the accuracy.

For every row in Table 3.2, the SimpleCNN model from Section 3.3 is trained on 8 different hyperparameter settings with 24 repetitions for each setting, using the random optimisation strategy. Again, the experiment is repeated for both the FashionMNIST and CIFAR-10 datasets. Because accuracy is not the main metric for this experiment, we are less interested in finding different hyperparameter settings as opposed to the first experiment. For this reason, we reduce the number of optimisation rounds and increase the number of repetitions. In total, we run 8 configurations $\times$ 8 optimisation rounds $\times$ 24 repetitions $\times$ 2 data sets $=$ 3072 training iterations.

[8] https://pytorch.org/docs/stable/generated/torch.optim.Adam.html

Table 3.2: Relation between model architecture and energy consumption.

Linear layers	Convolutional layers	ReLU layers
3	1	0
3	1	1
3	4	0
3	4	4
7	1	0
7	1	1
7	4	0
7	4	4

Table 3.3: Summary of the results for the optimisation experiment. Values are reported as x | y, where x represents the results with 5 hyperparameters (i.e. $\alpha, \beta_1, \beta_2, \epsilon, w$), and y those with 3 (i.e. α, β_1, β_2).

		CIFAR-10		FashionMNIST	
		Accuracy	Optimisation rounds	Accuracy	Optimisation rounds
DensePolyNN	Random	0.33 | 0.32	56 | 27	0.826 | 0.83	56 | 27
	Grid	0.37 | 0.40	42 | 55	0.81 | 0.81	40 | **19**
	Bayesian	**0.40** | **0.41**	**27** | **11**	**0.833** | **0.85**	**29** | 46*
DenseLinearNN	Random	0.35 | 0.38	50 | 63	0.81 | 0.84	52 | **23**
	Grid	**0.40** | **0.40**	53 | **35**	0.81 | 0.81	30 | 54
	Bayesian	0.37 | 0.39	**4** | 38	**0.84** | **0.85**	**18** | 47*
SimpleCNN	Random	0.60 | 0.62	40 | **4**	0.**8879** | 0.879	29 | 39
	Grid	0.58 | 0.59	17 | 55	0.86 | 0.875	**14** | **20**
	Bayesian	**0.68** | **0.66**	**16** | 26	0.8876 | 0.**884**	20 | 36

3.5. Results

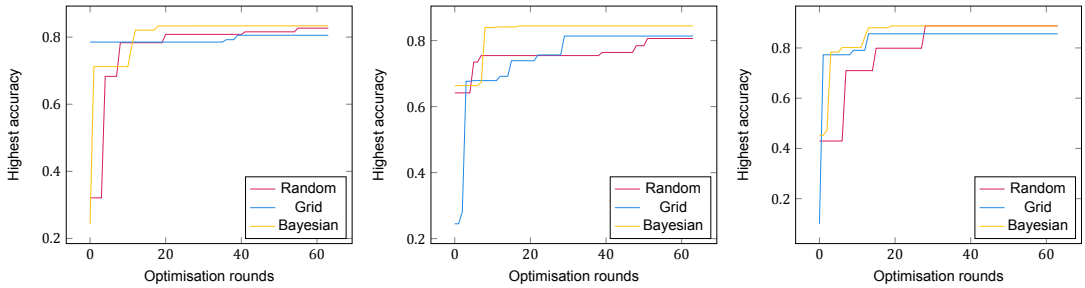
In this section, we report the results of the experiments formulated in Sections 3.4.1 and 3.4.2.

3.5.1. Hyperparameter Optimisation

The line graphs in Figures 3.3 and 3.4 show the highest achieved accuracy by the number of optimisation rounds for all the settings with 5 hyperparameters (i.e., $\alpha, \beta_1, \beta_2, \epsilon$ and w) on both the FashionMNIST and CIFAR-10 datasets. Figures 3.5 and 3.6, display the results for all settings with 3 parameters (i.e., α, β_1 and β_2). The figures are separated into subfigures to distinguish between the results for the DensePolyNN (a), DenseLinearNN (b) and SimpleCNN (c) that were introduced in section 3.3. The total runtime of the hyperparameter optimisation experiment (Section 3.4.1) amounts to ± 85 hours.

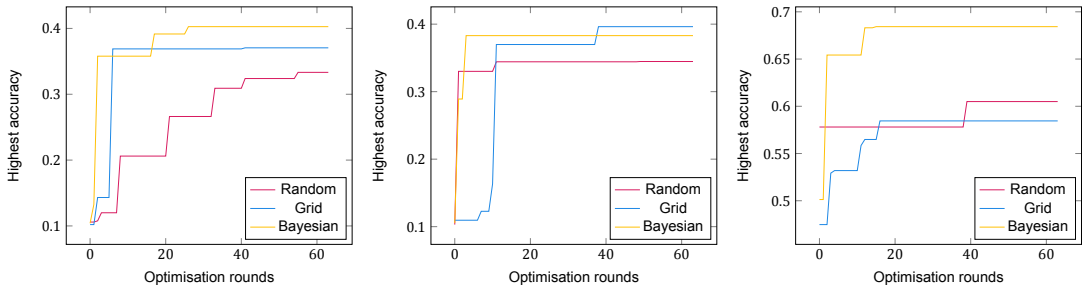
With an initial visual assessment, a few observations can be made. First, Bayesian optimisation proves to be the most effective strategy when compared with random and grid search. Regardless of the network or workload, it consistently outperforms the other strategies, only being overtaken slightly by grid search twice (3.4b and 3.6b) and narrowly matched by random search three times (3.3c, 3.5b and 3.5c). Second, between grid search and random search, there is no definitive winner. Random search performed better than grid search 5 out of 6 times on the FashionMNIST dataset and 2 out of 6 times on CIFAR-10. We have also summarised this data in Table 3.3. This table presents the highest accuracy for every experimental configuration (i.e. network $\times$ optimisation strategy $\times$ dataset $\times$ #hyperparameters) together with the number of optimisation rounds it took to achieve that accuracy.

Finally, notice that the Bayesian optimisation strategy converges to an optimum within 27 optimisation rounds on average. The two outliers with regards to this rule are marked by an asterisk (*) in Table 3.3. Nonetheless, a quick inspection of the corresponding graphs (3.5a and 3.5b) shows that there is only a very slight increase compared to the accuracy that was achieved after 27 optimisation rounds. The same cannot exactly be said for random optimisation. Most of the graphs follow a much more gradual incline with bigger jumps in accuracy. Overall, this strategy takes longer to converge. Grid search, on the other hand, does seem to converge rapidly. The numbers in Table 3.3 might sug-



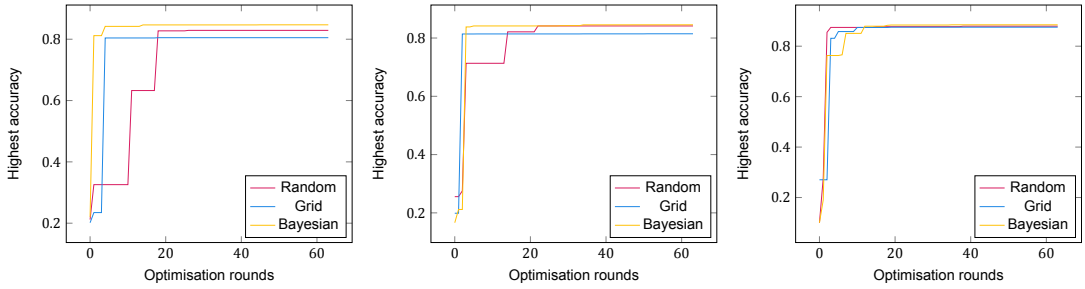
(a) DensePolyNN (b) DenseLinearNN (c) SimpleCNN

Figure 3.3: Convergence graphs for the hyperparameter optimisation experiment with 5 parameters on FashionMNIST



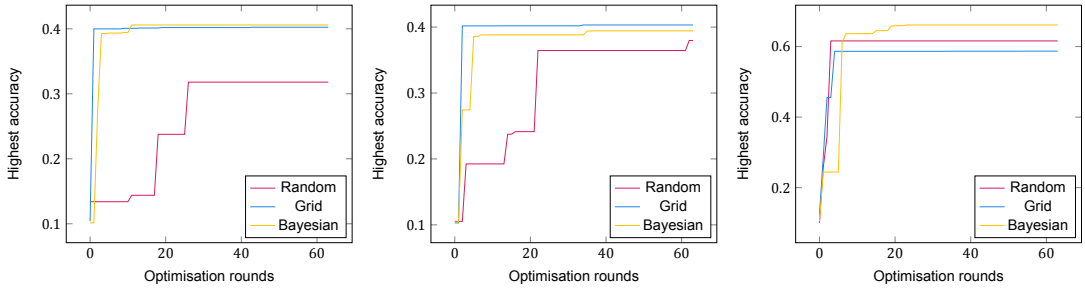
(a) DensePolyNN (b) DenseLinearNN (c) SimpleCNN

Figure 3.4: Convergence graphs for the hyperparameter optimisation experiment with 5 parameters on CIFAR-10



(a) DensePolyNN (b) DenseLinearNN (c) SimpleCNN

Figure 3.5: Convergence graphs for the hyper-parameter optimisation experiment with 3 parameters on FashionMNIST



(a) DensePolyNN (b) DenseLinearNN (c) SimpleCNN

Figure 3.6: Convergence graphs for the hyper-parameter optimisation experiment with 3 parameters on CIFAR-10

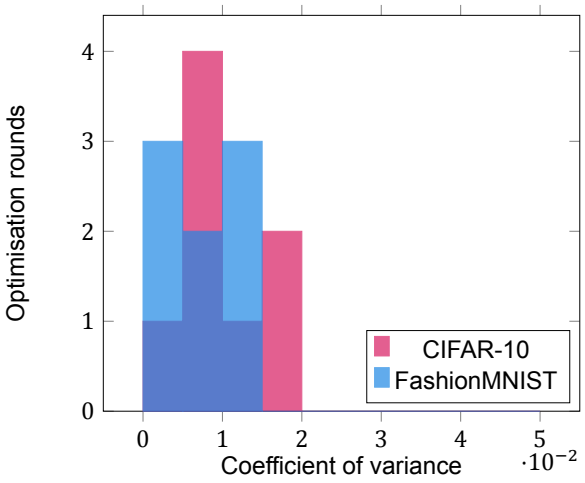


Figure 3.7: Histogram of the coefficient of variance for each run on the CIFAR-10 and FasionMNIST datasets

gest otherwise, but similar to what we observe with Bayesian optimisation, the increases in accuracy past 27 optimisation rounds are very small.

3.5.2. Network Architecture

The total runtime for all the different configurations of the network architecture experiment (Section 3.4.2) approximately amounts to 46 hours. The purpose of this experiment is to quantify the relation between the network architecture and the amount of energy that is being consumed during training.

To reinforce the validity of our results, we first show that the values of the hyperparameters, as chosen by the random optimisation function, do not significantly impact the energy consumption. The coefficient of variance (CV) is a metric that explains the relative size of the standard deviation to the mean. Because we assume that the hyperparameter setting has little to no influence on the energy consumption, we expect a very small CV (>1%) for all the optimisation rounds of a network. The histogram in Figure 3.7 depicts the CVs for every row in Table 3.2 on both the FashionMNIST and CIFAR-10 datasets. Every data point is a calculation of 8 optimisation rounds including 24 repetitions. We find an average CV of 0.009 and a maximum value of 0.018.

Now that we have shown that the energy consumption of a training iteration is independent of the hyperparameter settings in this experiment, we can analyse the network architecture in isolation. Because the data is not normally distributed, a conclusion made following the procedure described in Section 3.3.4, we perform the non-parametric Kruskal-Wallis test to identify if the energy consumed to train the network architectures can be distinguished statistically. Table 3.4 presents the corresponding p-values and effect sizes (η^2). Notice that out of the three layer types, convolutional layers and linear layers have a large degree of influence on the energy consumption, while the influence of ReLU layers is small. An additional post hoc comparison shows that all combinations of independent variables are significant as well. To put these statistics into perspective we compare the increase in average energy consumption by fixing each layer type. We use the notation **x|y** to distinguish results on the FashionMNIST dataset (x) from those on the CIFAR-10 dataset (y). The presence of ReLU layers contributes an average increase of **2.7%|2.9%**. For the linear layers, the jump from 3 to 7 layers accounts for an increase of **4.9%|6.6%**. The convolutional layers are the largest sources of energy usage. Introducing 3 additional layers on top of the first one increases the overall consumption by **95.3%|66.4%**.

Moreover, we carry out a trade-off analysis with respect to the energy consumption of a neural network and its achieved accuracy on the problem set. This comparison is visualised in Figure 3.8. The scatter plots in this figure highlight the relation of the energy consumption in Joules against the achieved accuracy on the test sets of FashionMNIST (a) and CIFAR-10 (b). In both scatter plots we can discern two clusters; one spread around a higher energy consumption which we will refer to as E^+; the other spread around a lower energy consumption, we call this cluster E^- (notice that the energy axis is reversed). All data points in E^- correspond to network architectures with a single convolutional layer, while the E^+ cluster contains all the networks with four convolutional layers.

Table 3.4: Kruskal-Wallis test results. From top to bottom, the tables refer to the experiments on the FashionMNIST and CIFAR-10 datasets respectively.

Factor (layer type)	Statistic	p	η^2	magnitude
Linear	481.799	$< .001$	0.155	large
Convolutional	2303.250	$< .001$	0.749	large
ReLU	176.545	$< .001$	0.055	small
Factor (layer type)	Statistic	p	η^2	magnitude
Linear	496.807	$< .001$	0.160	large
Convolutional	2303.250	$< .001$	0.749	large
ReLU	106.655	$< .001$	0.033	small

Table 3.5: Low energy performance compared against high energy performance.

	FashionMNIST		CIFAR-10	
	E^+	E^-	E^+	E^-
Average energy	1674 J	857 J	4588 J	2758 J
Average accuracy	0.872	0.864	0.639	0.572
Max accuracy	0.889	0.887	0.725	0.609
Std accuracy	0.011	0.010	0.056	0.021

Table 3.5 parses the most important data from the scatter plots into numerical values. The first two columns show the average energy consumption, average accuracy, maximum accuracy and the standard deviation of the accuracy for the E^+ and E^- clusters on the FashionMINST dataset. The latter two columns show the same information on the CIFAR-10 set. Notice that the average and maximum accuracy for both clusters on the FashionMNIST dataset are particularly close together, only varying by less than 1%. For CIFAR-10, which is a more computationally complex set, this difference is more significant. A little over 6% for the average and almost 12% for the maximum accuracy.

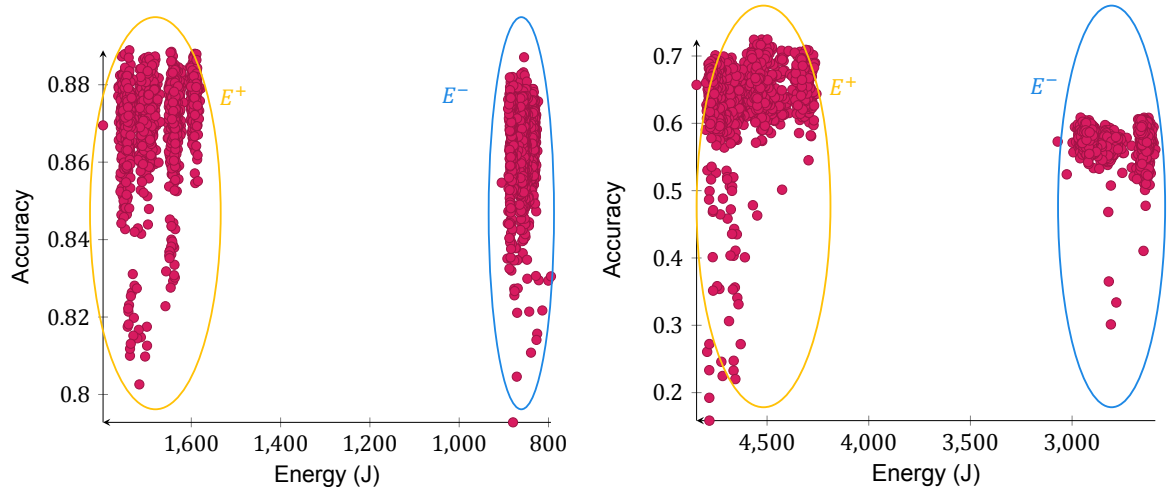
3.6. Discussion

This empirical study aims to provide insights into possible improvements for deep learning pipelines out of environmental considerations. In this section, we answer both research questions by analysing the results of the experiments.

3.6.1. Hyperparameter Optimisation

The conclusion to RQ_1^α is that *Bayesian optimisation* is the most energy-efficient strategy during the training phase of a machine learning model. Out of all three strategies, Bayesian optimisation consistently finds hyperparameter configurations that result in the highest accuracy and it does so within the least amount of optimisation rounds ($\pm$27). Because Bayesian optimisation requires the storage and constant fitting of a probabilistic model, one downside is the difficulty of parallelisation, but even that is not impossible [30]. Based on our results, there seems to be no good argument to choose one of the other methods.

Nevertheless, we cannot deny the presence of grid search and random search within the deep learning field. Both algorithms are easy to understand and implement and could serve a purpose during early exploration or calibration. In this context, does one of the algorithms dominate the other? Solely based on our results, we cannot make any decisive claims. We can, however, assess the practicality of both solutions. Grid search is an exhaustive method with a search space that increases exponentially by the number of hyperparameters. Considering every configuration in that search space is not feasible and goes against our philosophy of energy-efficient training. As a consequence, we can only consider a portion of the complete search space, which defeats the purpose of the grid search. By randomly selecting samples from the search space, grid search devolves into a random search with a finite number of options. Furthermore, as Bergstra and Bengio [25] explain: hyperparameter optimisation problems in high-dimensional spaces have a low effective dimensionality. What this entails



(a) FashionMNIST (b) CIFAR-10

Figure 3.8: Scatter plots of the energy consumption vs the achieved accuracy

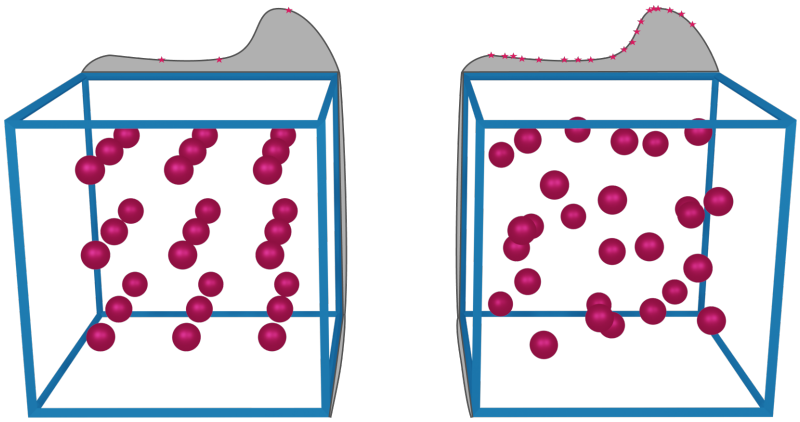


Figure 3.9: Grid search (left) vs Random search (right) on a problem with a low effective dimensionality.

in our context is that some parameters will have a much larger influence on the accuracy than others. Figure 3.9 illustrates how random search exploits this property more effectively than grid search. The cubes in the image represent a three-dimensional problem where only one parameter has a significant influence on the function value. With the grid search (left), although we consider 27 distinct samples, only 3 values of the important parameter are tested. On the contrary, the random search (right) tests a new value for every sample.

Furthermore, random search facilitates the job of AI engineers because it does not require any human guidance apart from selecting the bounds. For these reasons, we recommend the use of random search over grid search for the early stages of the training. Our results show that random search is not worse than grid search for problems with ≤ 5 hyperparameters. Additionally, random search should remain a valid baseline strategy if the number of parameters is increased, while grid search will fall short due to the expanding search space.

3.6.2. Network Architecture

To answer RQ_2^{α}: reducing the layer complexity of a neural network is a valid option to lower the energy consumption of a deep learning pipeline. Besides computer vision, we believe that our results can be generalised to other fields such as speech recognition or natural language processing as well. The computational complexity of different layer types is a constant that will present similar effects on the energy consumption in a different context. The observation of diminishing performance is also not a very

bold claim. However, whether the loss in accuracy resulting from architecture simplifications is acceptable, depends largely on the context. As the visuals in Figure 3.8a and the corresponding summarised data in Table 3.5 make apparent, the accuracy gain for introducing complexity on a relatively simple problem (FashionMINST) is very small. In this case, we would argue that the diminishing return in accuracy is not worth doubling the number of expended Joules. For the CIFAR10 problem set, although there are still diminishing returns, the difference in accuracy we observe is quite significant. Ultimately, what it comes down to is how much error is acceptable for the application in question. To aid this decision, it is important that researchers monitor the performance slope of their model and that they report some metric that relates to the energy efficiency throughout the pipeline, such as Joules or FLOPs. By combining the accuracy and energy trends, we can make more considerate design choices, reduce the layer complexity, and improve the efficiency of the pipeline. As we have shown, these changes could lower the overall energy consumption of a training pipeline by up to 95%. Many state-of-the-art models could also benefit from this philosophy. Following the current trend, new models are becoming exponentially larger and more costly, while the performance only sees marginal increases. If all these models would also report their energy consumption, it would vastly change the perspective of which one is 'the best' and give rise to new research efforts that focus on energy-efficient design.

3.6.3. *Extra:* optimise GPU load

While collecting energy measurements and analysing results, we were faced with a natural follow-up question: do we really need to measure energy consumption or could we simply rely on time efficiency? While looking at our data, we noticed that different experiments yield a different GPU load. Hence, a model that trains faster might be using the GPU more efficiently, but one cannot immediately draw conclusions w.r.t. the pipeline's total energy consumption.

Nevertheless, we know from previous research that AI systems that optimise GPU usage can reduce their energy consumption by a factor of 10 [41]. When we look at our experiments, model training resulted in a GPU load that ranged between 40% and 60%. Similar values have been reported by a study conducted at Facebook AI [41]: a large portion of machine learning model experimentation only utilise their GPUs at 30–50%. This shows that an important step for Green AI engineering is to monitor and optimise GPU acceleration in pipelines. One should also consider the embodied carbon from the GPU hardware. This is a serious problem because underutilising models require more GPUs than what should be theoretically sufficient [42].

Hence, we argue that AI frameworks ought to feature GPU-enabled operations out of the box. Tools should be refactored or improved to support both AI practitioners and software engineers to monitor and optimise the GPU usage of their AI systems and associated pipelines. Developing AI systems is already a trans-disciplinary field that requires expertise across different domains. Hence, making energy consumption a first-class citizen for designing these systems will facilitate communication across disciplines and boost Green AI efforts substantially.

3.7. Threats to Validity

In this section, we go through potential threats to the internal, external and construct validity, as well as the reliability.

3.7.1. Internal validity

It could be argued that our method of measuring energy for RQ_2 does not provide an unbiased value. Different tasks running in the background could introduce noise to our measurements. To reduce the influence of this threat, the experiment was performed on a clean installation of Ubuntu 20.04. The only redundant program that might have had a slight impact on the measurements was a running instance of TeamViewer[9] that was used for periodic monitoring. Every optimisation round included 24 repetitions to drown out this effect. Moreover, when calculating the effect sizes of the layer types, we omit the hyperparameter configurations. It is possible that different hyperparameter settings change the overall energy consumption of a neural network, however, in Section 3.5.2 we calculate the coefficient of variance to show that this effect is negligible.

[9] https://www.teamviewer.com/nl/

3.7.2. External validity
During the experiments, we did not consider the optimal utilisation of the GPU. This might have a negative impact on the generalisability because the relation between utilisation and power is not necessarily linear. Kistowski et al. [43] find that for CPUs, there is a steep increase in power output starting at around 80% utilisation.

3.7.3. Construct validity
Because we solely consider the power usage of the GPU and ignore the contributions of other components, such as memory access, we do not capture the actual energy consumption for training a model. Nevertheless, we specifically selected GPU-heavy workloads and made sure to factor out the idle energy consumption. The results are therefore still valuable to compare relative to each other.

3.7.4. Reliability
To increase the reliability, we made an effort to assemble a complete reproduction package. The source code for training the neural networks, along with the results of the experiments and the statistical analysis, are all available online[10]. These components were created by a single developer, but all the involved authors reviewed and approved the entire process. The statistical analysis was replicated by one of the authors to confirm the findings.

3.8. Summary
In this study, we have expanded the horizon of green software to the realm of AI applications. Our empirical study shows that Bayesian optimisation can find the most optimal set of hyperparameters within the least amount of iterations, where 27 should be sufficient in most instances (RQ_1^{α}). Grid search and random search have their purposes as baselines algorithms. If the parameter bounds are chosen with care, neither strategy significantly dominates the other. Nevertheless, we advocate the use of random optimisation since the exhaustive nature of grid search often implies that one cannot consider the complete search space anyway. Additionally, because the function of hyperparameters has a low effective dimensionality, it is more reasonable to introduce randomness to the search space.

Furthermore, we have investigated the impact of a neural network's architecture on its energy consumption, followed by a trade-off analysis with regards to the accuracy (RQ_2^{α}). We found that for a substantial increase in energy consumption, the increases in accuracy see diminishing returns. We advise reducing the number of convolutional layers to a point where the accuracy is still within a reasonable margin. This entirely depends on the project in question and should be evaluated case by case.

[10] https://anonymous.4open.science/r/green-ai-46E7

4

Second study - Batching during inference

4.1. Introduction

In this study, we focus on the inference phase of the deep learning pipeline. During the early phase of neural networks development, untrained models are tweaked and tuned by repeatedly passing large datasets through them. We have seen that this process requires many repetitions and can become very costly. When a model is fully trained, it may be used for inference. In this phase, the model probably will not process the same excessive quantity of data anymore. However, fully trained models can be deployed to a huge number of independent devices that collectively do process a lot of data. From a study by Facebook AI [41], we learn that at least 50% of the operational carbon cost for training and inference of large machine learning tasks can be attributed to inference.

Another problem is that the devices that act as hosts to the neural networks, do not necessarily have the same computational power as the machine used for training. In the case of mobile devices, battery life also becomes a factor. For these use-cases, efficiency is essential. Some studies specifically focus on developing computation-efficient models because of this [44], [45].

These reasons compel us to argue that one should not only optimise for training and development; but consider the complete life-cycle of a neural network. The batch size is one of the most important hyperparameters to tune during the training phase. It has implications on the model accuracy and generalisability [46], training times and parallelisability [47], etc. During inference, however, there is no dataset available that can be divided into batches. Instead, the incoming stream of requests depends on some external factor that provides input. Because of this, any attempt to process data in batches of a specific size inadvertently introduces a form of delay to the response. This is an important difference from the training phase because the GPU always exerts some amount of power even when idle. In this study, we analyse this two-way optimisation problem between the energy consumption of response time. In addition, we present a timeline of state-of-art neural networks and compare them in terms of their energy consumption. In doing so, we strive to answer the following two research questions:

- RQ_1^{β} How does batch inference affect the energy consumption of image vision tasks under different frequencies of incoming requests?

- RQ_2^{β} How has the energy efficiency of image vision models evolved in the last decade?

The methods and tools used in this study are accounted for in Section 4.2. In Section 4.3, we go over the experiment that was devised to collect this delay and the energy consumption for different experimental settings. Further down in the same section, the data collection for the neural network energy timeline is explained. The results of the experiment are presented in Section 4.4 and we analyse and elaborate on these in Section 4.5. Finally, we elaborate on the threats to our study in Section 4.6 and wrap up our findings and recommendations with the summary in Section 4.7.

4.2. Research Methods

The methods we use are similar to the ones described in Section 3.3. We use Pytorch to manage neural networks and we query the NVIDIA System Management Interface to obtain the power output of the GPU.

4.2.1. Case Selection

An important difference with the first study is that we opt to use pre-trained networks that are considered *state-of-the-art*. We compile a list of five different networks provided by Pytorch[1]:

- **AlexNet** (2014) [48]
- **DenseNet** (2016) [49]
- **ShuffleNetV2** (2018) [45]
- **VisionTransformer** (2020) [50]
- **ConvNext** (2022) [51]

The reason we choose these five networks in particular; is because their initial publication dates are spread out evenly in the past decade. This not only provides a good variety of different network designs, but it also facilitates RQ_2^{β}, where we attempt to compare the energy consumption of modern neural networks to their predecessors. It should also be noted that these models are designed for image classification. We choose this problem space because image recognition is a canonical deep learning challenge and because it helps to relate our findings to the study from Chapter 3.

All experiments are performed on a single GeForce GTX-1080 GPU[2]. The stop condition for any run mentioned in this study is a fixed amount of processed image classification requests. Because inference is reliant on external providers for incoming requests, the time it takes to receive a certain amount of requests can vary a lot. To make a fair comparison of the differences in energy consumption, we assume regular streams of incoming requests in this study.

4.2.2. Experimental Tooling

We develop a testbed in Python[3] to automate the data collection. The software provides a simulated queue that creates image classification tasks at a frequency that can be configured manually. Requests are then pulled from the queue and collected in a batch with configurable size. These batches are fed to the neural networks. Apart from the five networks mentioned in Section 4.2.1, our tool is immediately compatible with any image vision model that Pytorch provides or any custom model that is built using the same framework. We highly encourage experimenting with different architectures and reporting the results.

4.2.3. Data Collection

For this study, we are interested in two quality metrics: the average energy usage per image classification and the maximum response time, meaning the time between a user submitting a task and receiving an answer. As mentioned before, we assume that the stream of incoming requests is about constant. This entails that the time between any two requests will be roughly the same for a fixed frequency.

We obtain the power usage of the GPU by querying the NVIDIA System Management Interface[4] every 10 milliseconds. The total energy consumption can then be computed as a factor of time and the average power. Finally, this amount is divided by the total number of images to calculate the desired metric. For this study, we do *not* factor out the idle consumption of the GPU, because idling is an important part of the experiment. By increasing the batch size, we inherently increase idle times as well. The experiment is meant to show whether this improves the efficiency or not.

The maximum response time is determined by providing each incoming classification task with a timestamp. This timestamp is resolved as soon as the request is handled and the program keeps track of the longest time in memory.

[1] https://pytorch.org/vision/stable/models.html
[2] https://www.nvidia.com/nl-nl/geforce/10-series/
[3] https://anonymous.4open.science/r/green-ai-46E7
[4] https://developer.nvidia.com/nvidia-system-management-interface

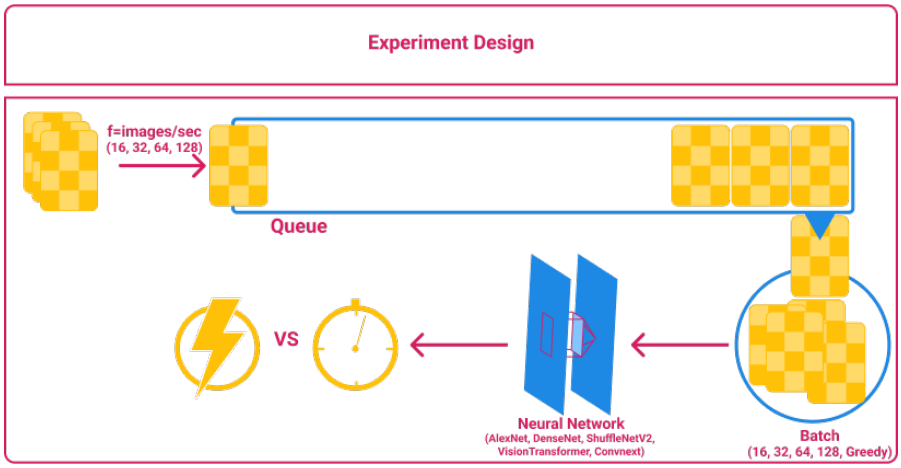


Figure 4.1: Inference experiment diagram

4.3. Experiments

In the following section, we describe the setup of the inference experiment in detail. We use the results of this single experiment to answer both research questions (RQ_1^β & RQ_2^β).

4.3.1. Batching During Inference

During the image vision training phase, a neural network processes thousands upon thousands of images. To parallelise this task and employ more of the available GPU power, these images are often processed in batches. During inference, however, when we look at image classification in a practical setting, the usual dataset is replaced by an irregular stream of incoming requests. We refer to the number of images that come in per second as the *frequency*. If we choose to perform inference in larger batch sizes, depending on the frequency, we might have to wait for a batch to fill up before passing it on to the network and this increases the response time to the user. In a nutshell, this is the game that we attempt to optimise: the trade-off between energy consumption and wait time.

To carry out this experiment, we simulate a queue that receives incoming image classification requests. These images are then passed to a neural network using some batching strategy. The setup is as follows: we compare four different frequencies (16, 32, 64 & 128); the five different networks mentioned in Section 4.2 and five batching strategies (16, 32, 64, 128 & Greedy). This amounts to 100 different experimental configurations. The greedy batching strategy is the baseline to which the other batch sizes are compared. Greedy in our simulation means all the images in the queue are passed to the network as soon as it becomes available[5]. The flowchart in Figure 4.1 displays this entire process in a graphical format.

Each configuration triplet `<frequency, network model, batching strategy>` comprises one *run*, which continues until 2^{13} image classification tasks have been requested and processed. Note that because we use the request count as the termination criterion, we cannot compare settings with different frequencies to each other in terms of energy consumption. This is because as long as the GPU can keep up with the incoming stream, the frequency will determine for how long the simulation will continue and during idle periods, the GPU will still exert power. Therefore, we expect that low-frequency simulations will consume more energy absolutely than high-frequency simulations. For this reason, we compare only the results that were accumulated using the same frequency with each other when formulating our answers to RQ_1^β.

4.3.2. Image Vision Energy Timeline

To answer the second research question (RQ_2^β), we calculate the average energy consumption per image over all five batching strategies for each combination of neural network and simulation frequency. This amounts to four values per model or 20 data points in total. We present these results in a bar chart in Section 4.4.

[5]The maximum greedy batch size is set to 128 to avoid out-of-memory issues

4.4. Results

In this section, we present the results from the inference batching experiment.

4.4.1. Batching During Inference

For each of the five image vision models (i.e. AlexNet, DenseNet, ShuffleNetV2, VisionTransformer & ConvNext), we perform a trade-off analysis concerning the average energy consumption per processed image and the maximum wait time in the queue. The results are visualised in the scatter plots from Figure 4.2. In these plots, the x-axis represents the average energy consumption in Joules for processing a single image. The y-axis shows the maximum time from when a user submits an image until he receives a response. Furthermore, there are four different classes that each correspond to a different setting of the simulation. For a class f=X, X represents the frequency of the incoming image requests, i.e. the class f=32 will have 32 image requests every second. Finally, the labels on top of each data point correspond to the size of the batches, where *G* refers to the greedy batching strategy.

Table 4.1: GPU peak power in Watts (W) differences for small and large batch sizes

Model	Batch size 1-2	Batch size 128	% difference
AlexNet	$\pm$65W	87.3W	$\pm$34.3%
DenseNet	72.5W	163.1W	124.9%
ShuffleNetV2	$\pm$65W	87.1W	$\pm$33.9%
VisionTransformer	76.2W	185.8W	144.0%
ConvNext	93.1W	166.4W	78.6%

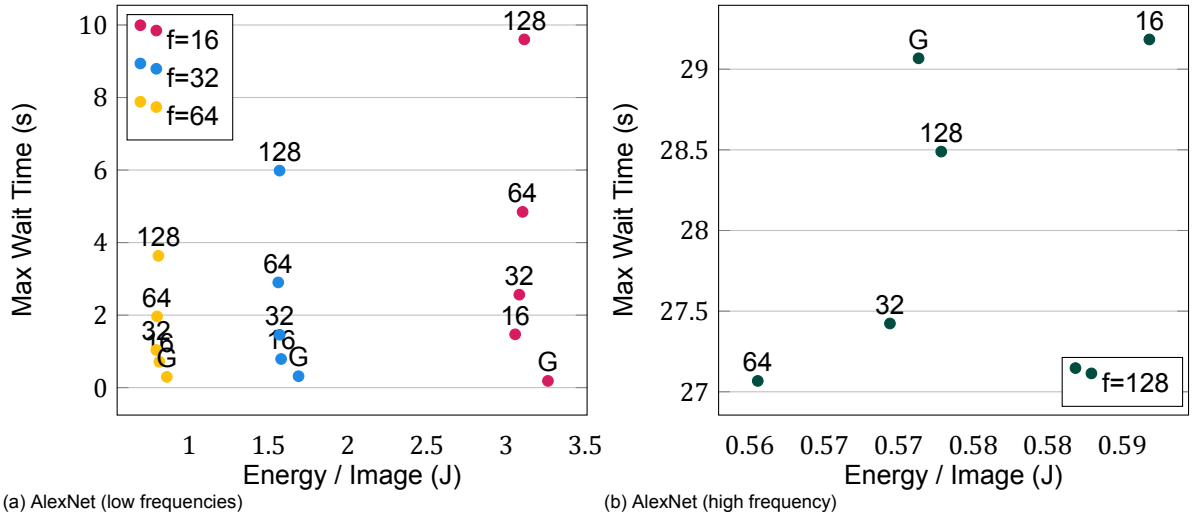
There are several things that we can observe from these scatter plots. First of all, every network responds to batching differently. AlexNet (Figures 4.2a and 4.2b) and ConvNext (Figures 4.2i and 4.2j) both clearly benefit from batching as the greedy strategy is almost always the least energy efficient. Two exceptions are the frequency 32 simulation for ConvNext and the frequency 128 simulation in general. The latter is easy to explain if we consider the average batch size of the greedy strategy. For frequencies 16, 32 and 64, this ranges from 1 to 7 images per batch, whereas for the 128 frequency simulation, the average batch size lies around 122. Given the positive effect of larger batch sizes, we can understand why the greedy strategy would perform better in high-frequency scenarios.

The VisionTransformer (Figures 4.2g and 4.2h) also generally runs more efficiently for larger batch sizes. For this model, the exception can be found in the frequency 16 simulation. Here we find that the greedy strategy, with an average batch size of 1.0004, is the most energy-efficient.

For another interesting observation we direct our attention to the scatterplot for ShuffleNetV2 in Figures 4.2e and 4.2f. We find that there is virtually no horizontal spread in the points, which suggests that the model's efficiency does not depend on the batch size. This belief is enforced if we also consider the average peak power of the GPU while processing a batch of images. For all the other models, there is a large difference in peak power for processing a small batch of images versus a large one. This does not hold for ShuffleNetV2, which can be seen in Table 4.1. This table shows that not only ShuffleNetV2, but also AlexNet have a relatively small change in peak power for different batch sizes.

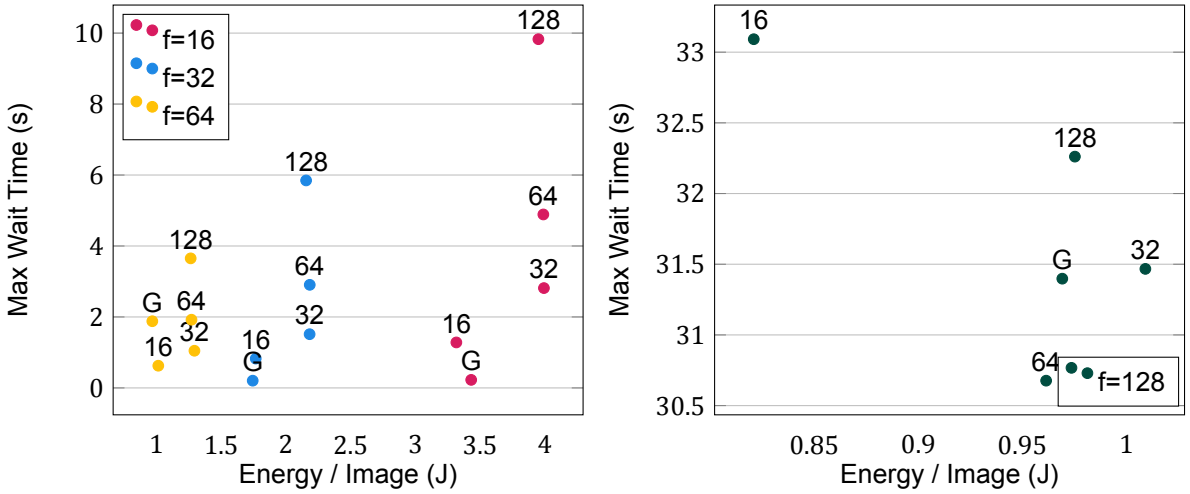
Finally we look at the results for DenseNet in Figures 4.2c and 4.2d. Out of all five networks, the observed behaviour for DenseNet is the most contradictory. We find that it performs the most efficiently for smaller batch sizes regardless of the simulation frequency. Because the greedy strategy takes very small batch sizes (1-2) for frequencies 16-64, we find that greedy is actually a very energy-efficient strategy for DenseNet.

Nevertheless, we cannot make any design decisions based on these results without considering the second metric in the scatter plots. Although the greedy strategy is often the least energy-efficient, we see that it consistently achieves the lowest wait times. For the frequencies 16 through 64 simulations, the graphs show that the greedy strategy results in near-instant response times while increasing the batch size introduces a maximum delay between 1 and 15 seconds. For the high-frequency simulation, we observe a shift in this trend. Since the GPU is not quite able to process all the images as soon as they enter the queue, a bottleneck is formed. This results in higher wait times in general and we find that the smallest batch size of 16 is the least favourable in this case. Across all models, the batch size of 64 is the most optimal with regard to the maximum wait time.



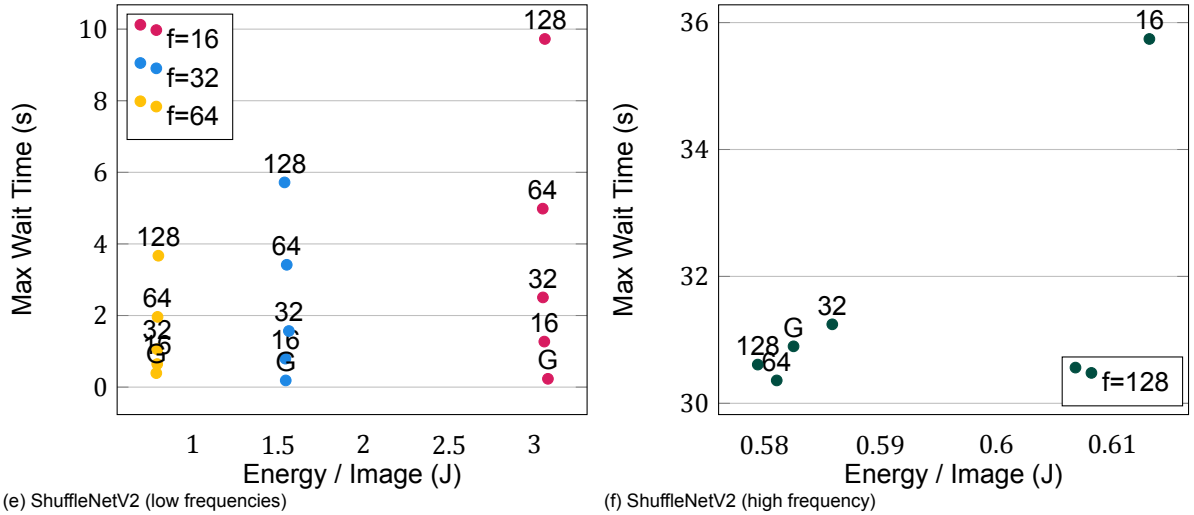
(a) AlexNet (low frequencies)

(b) AlexNet (high frequency)



(c) DenseNet-121 (low frequencies)

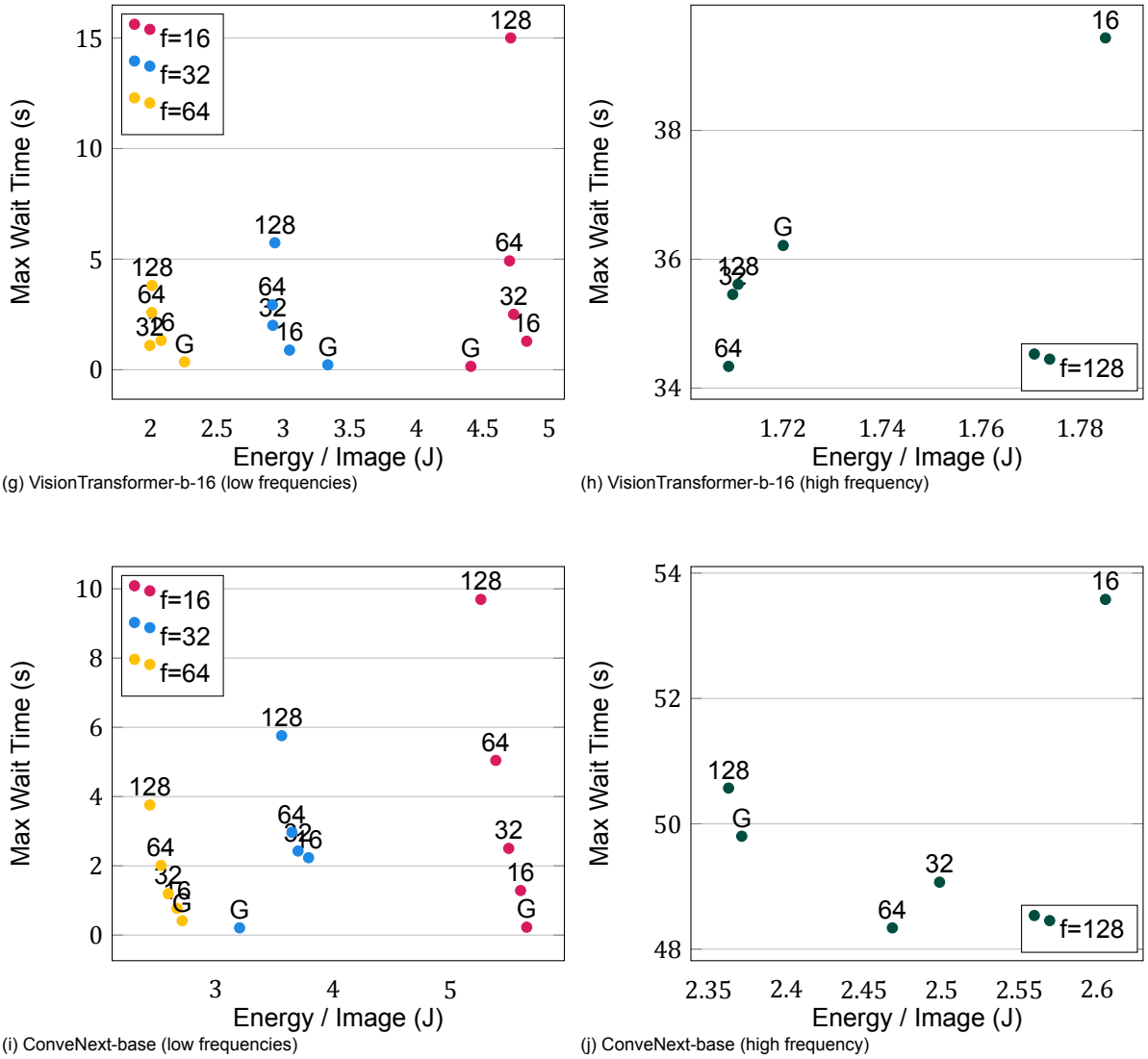
(d) DenseNet-121 (high frequency)



(e) ShuffleNetV2 (low frequencies)

(f) ShuffleNetV2 (high frequency)

Figure 4.2: Energy consumption per unit inference vs the maximum response time. Low frequency simulations (16, 32, 64) are shown on the left and high frequency simulations (128) on the right.



(g) VisionTransformer-b-16 (low frequencies)

(h) VisionTransformer-b-16 (high frequency)

(i) ConveNext-base (low frequencies)

(j) ConveNext-base (high frequency)

Figure 4.2: Energy consumption per unit inference vs the maximum response time. Low frequency simulations (16, 32, 64) are shown on the left and high frequency simulations (128) on the right.

4.4.2. Image Vision Energy Timeline

For the second part of this experiment, we take a step back to compare the overall energy consumption of the five models to each other. Figure 4.3 shows the average energy required to process a single image in four different simulations. This average comes from the summation of the energy consumption for all the batch sizes for one such simulation. The models on the x-axis are in a specific order, which is not necessarily an increasing one in terms of energy efficiency. The models on the left and their respective papers were published before the models on the right. This creates an intuition for how energy efficiency evolves over time. The chart shows that there is a positive linear relationship between energy consumption and publication date. The exception to this trend is ShuffleNetV2, which, in terms of energy efficiency, is on the same level as AlexNet.

It would not be fair to look at this graph without considering the improvements in accuracy that the newer models achieve. In Figure 4.4, we highlight the relative changes in accuracy and energy consumption from every network compared to AlexNet, which was published first. The energy consumption is based on the results from this study and the accuracy refers to the achieved top 1 accuracy on the ImageNet dataset[6]. From this graph we can conclude that since 2012, the energy consumption has

[6]https://paperswithcode.com/sota/image-classification-on-imagenet

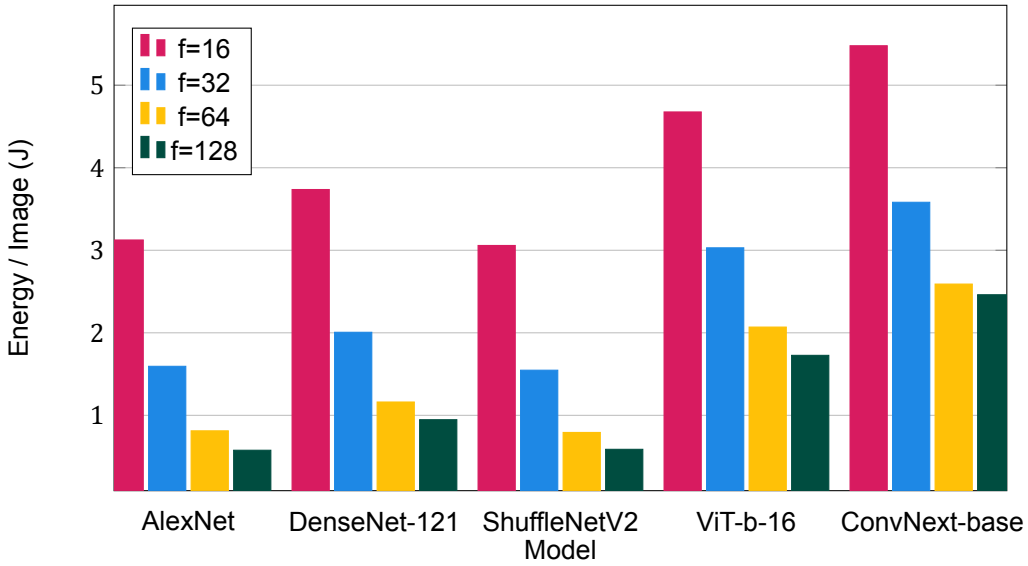


Figure 4.3: Energy comparison of different image vision models throughout the years

seen a steep increase of 131% and this trend does not start to fall off. Accuracy, on the other hand, has improved by 35%. Also notice that despite ShuffleNetV2's energy efficiency, it does not seem to sacrifice anything in terms of performance.

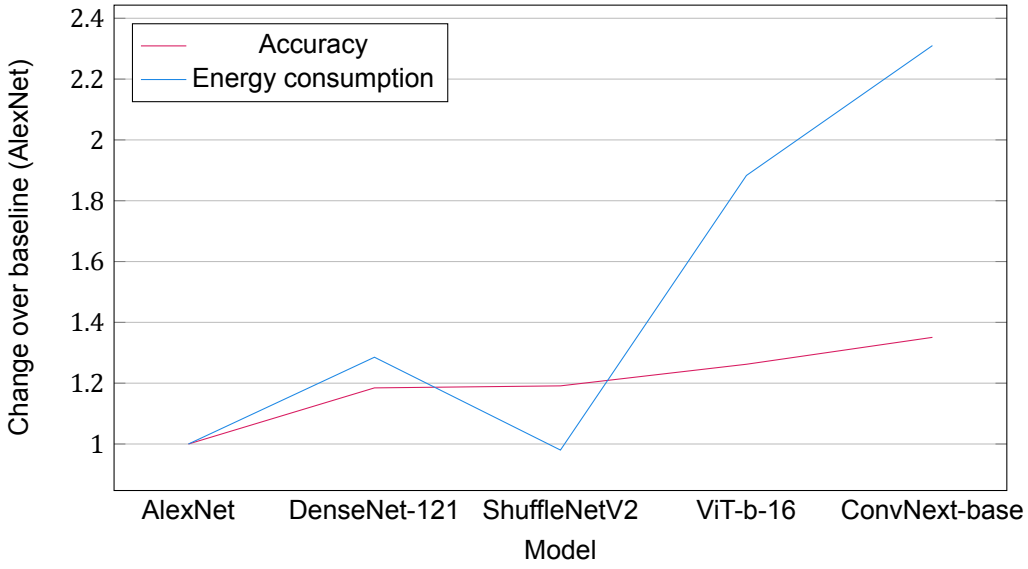


Figure 4.4: Relative changes in accuracy and energy consumption compared to AlexNet

4.5. Discussion

In this section, we reflect and elaborate on the results as presented in Section 4.4. First, we consider the trade-off between energy and wait time to formulate an answer to RQ_1^{β}. After that, to answer RQ_2^{β}, we examine the collected energy consumption of the five networks. To further explain our findings, we look at the inner mechanisms and design principles of the five image vision models.

4.5.1. Batching During Inference

What we learn from the results might be somewhat unexpected. We did not find one recommended batch size or even an indication that reduces energy consumption in all cases. Instead, we find that

each network behaves differently under varying batch sizes. Nonetheless, for some of the networks, the potential gain in energy efficiency cannot be ignored.

AlexNet was published in 2012 and at that time it revolutionised the field of image vision. The architecture of the model is simple, with only five convolutional layers [48]. ConvNext is a more modern CNN that incorporates design choices from classical CNNs like AlexNet and ResNet, that have been presented in the past decade [51]. Even the base model is quite a bit more complex than AlexNet, containing four different blocks for a total of 36 convolutional layers[7]. Nonetheless, both of these models are pure CNNs that do not rely on any special tricks. If we compare the results now, we find that there is a bias towards larger batch sizes as opposed to the small batch sizes of the greedy strategy in the low-frequency simulations. For the high-frequency simulation, where the inference becomes the bottleneck, the greedy strategy will process larger batch sizes, decreasing the energy consumed per image. Because we do not expect large data centres to experience this bottleneck, generally speaking, we can conclude that purely convolution-based models will benefit from performing inference in fixed batches (>16) rather than using a greedy strategy. For AlexNet, a batch size of 32 seems ideal because it limits the maximum wait time and the difference in energy consumption with the larger batch sizes is very small. For ConvNext, the largest batch size is nearly always the most energy-efficient, therefore we recommend a batch size of 128 (or even larger) when solely considering energy consumption. The increase in wait time should be evaluated per use case.

The VisionTransformer is the only network that does not rely on convolution. Nevertheless, we find that batching almost always results in lower energy consumption when compared to the greedy strategy. Again we recommend a batch size of 32 for the same reason as with AlexNet. The difference in energy consumption with the largest batch sizes is small, so here we can afford to optimise for the wait time. Because we do not evaluate any other transformers in this study, it cannot be guaranteed that these results will generalise. However, since the VisionTransformer was designed to closely resemble the architecture of a “standard transformer” as used in NLP [50], we can make an educated guess that this will be the case.

ShuffleNetV2 and DenseNet are the anomalies in this experimentation. For ShuffleNet, we find that the energy consumption is completely invariant from the size of the batches. The wait times still scale as expected, therefore greedy batching is the most optimal inference strategy for this network. The energy consumption of DenseNet does differ per batching strategy, but this time with a bias towards smaller batch sizes. This network seems to consume the least amount of energy with batches of size $\leq$16. This means that for the low-frequency simulations, the greedy strategy is optimal. To get an intuition as to why this may be the case, we look at the architecture of DenseNet. In regular CNNs, the output of one layer is passed on only to the next layer. In DenseNet, all the layers are *densely connected*, which means that any layer receives the output from all the preceding layers [49]. All the layers remain occupied until an image has been completely processed by the network. One can imagine that this translates poorly to the parallel processing of multiple images.

Now that we have established how each network responds differently to batching and why that makes it difficult to provide recommendations, we move to answer to RQ_1^{β}:

“How does batch inference affect the energy consumption for image vision tasks under different frequencies of incoming requests?”

We find that in some cases, batching of the requests has a positive effect on the energy consumption of a neural network. However, there are strong exceptions to this observation. Our recommendation to AI practitioners is therefore as follows: When preparing newly trained networks for practical application, one should consider the batch size as an optimisation parameter that needs to be tuned. First, we establish whether the network runs more efficiently on small or larger batch sizes and then we tweak the batch size to lower values until the system adheres to the tolerated response time for the use case.

4.5.2. Image Vision Energy Timeline

In terms of accuracy and energy consumption, the timeline that we have presented in Figure 4.3 looks consistent and predictable. It also presents a critical problem: Although the innovations between 2012 and the present have led to impressive advancements in our neural networks and their precision, the

[7] https://pytorch.org/vision/main/_modules/torchvision/models/convnext.html

potential gains in this regard are starting to diminish. We formulate our answer for RQ_2^{β}:

"How has the energy efficiency of image vision models evolved in the last decade?"

Modern image vision models consume more than twice as much energy as earlier iterations and although these models demonstrate better performance, the gains in accuracy are limited. It is not surprising that we find this to be the case. Accuracy (or a similar measure) is the metric that currently defines what is "state-of-the-art" [1], in fact, many challenges and benchmarks only request a submission of the top-1 or top-5 accuracy. Some leaderboards do focus on cost or energy consumption[8], but these are far and few between. If more challenges would accept submissions of new models where energy consumption is considered as a primary objective alongside accuracy, we can create opportunities for Green AI research. The proof that competitive models can also be efficient is already there. We established before that the ShuffleNetV2 architecture manages to break the increasing energy trend without bowing down to its predecessors in terms of accuracy. We look at the 2018 publication of ShuffleNetV2 to find out how this was accomplished [45]. The authors mention that most neural network design is guided by an indirect metric of the computational complexity: the number of floating-point operations (FLOPs). However, FLOPs only account for a part of the equation. The direct metric, speed, is also influenced by other processes like memory access. ShuffleNetV2 was designed with this mindset, to optimise for the direct metric of computational complexity rather than an indirect one. This goal of designing a fast network coincidentally resulted in a network that is also energy-efficient. In the same paper, the authors present a collection of four guidelines for efficient network design:

1. Equal channel width minimizes memory access cost (MAC)
2. Excessive group convolution increases MAC
3. Network fragmentation reduces degree of parallelism
4. Element-wise operations are non-negligible

Many of these guidelines focus on the reduction of memory access during classification tasks. This could be an interesting starting point for future research in Green AI.

4.6. Threats to Validity

In this section, we go through potential threats to the internal, external and construct validity, as well as the reliability.

4.6.1. Internal validity

During early experimentation, we noticed that the GPU was idling on a higher power output for the first few minutes. Because this influenced the average energy consumption for some of the configurations, we introduced a warm-up phase. Before starting a new simulation and logging the energy consumption, we allowed the GPU to "warm up" by passing 256 batches of 32 images through the respective model. This factors out most of the inconsistencies.

4.6.2. External validity

We mentioned before that the results collected in this study do not grant opportunities for firm recommendations and guidelines. Because we found a strong deviation in how different neural networks are influenced by batch inference, we can hardly claim that our findings will generalise well to other types of models. As such, our main contribution is not on the empirical results, but on the finding that a correct batching strategy will improve the overall energy efficiency and should therefore be tuned accordingly.

4.6.3. Construct validity

The main factor that hurts the construct validity is how accurately our simulation mirrors a real scenario. For the experiments, we assumed a constant workload with little to no deviation. In practice, one would expect a more erratic stream of incoming requests, with some periods of complete downtime. Naturally,

[8] https://dawn.cs.stanford.edu/benchmark/

it is undesirable to hold an unfilled batch while nothing new is coming in, so there should be some maximum time since the last request to avoid that. Nevertheless, our focus was not on optimising this simulation, but on investigating the energy efficiency of different batching strategies. Even in a more realistic scenario, the deviations in energy consumption that we observed should remain the same.

4.6.4. Reliability

A single developer worked on accumulating the results presented in this study, but all the involved authors reviewed and approved the entire process. The complete reproduction package is available online[9]. This repository contains the source code that can be run to reproduce the results for any of the models from Section 4.3 or a different one provided by the Pytorch library[10].

4.7. Summary

In this study, we examined the energy efficiency of different neural networks that have been presented in the past decade. We simulated how these networks could be employed in a practical setting and extracted the optimal batching strategies for each. We learned that there is no one size fits all solution for recommending a batching strategy (RQ_1^{β}).

AlexNet and ConvNext both operate more efficiently when using fixed batch sizes as opposed to greedy batching. Our results suggest a batch size of 32 for AlexNet and 128 (or larger) for ConvNext. Because of their classical architecture, we expect these results to generalise well to other pure CNN-based models.

For the VisionTransformer, we find a similar result. A batch size of 32 appears to be the sweet spot in terms of GPU utilisation. A smaller batch size hurts the energy efficiency and a larger one does not provide any improvements. For future research, it would be interesting to repeat this experiment and evaluate more transformer-based models to see if these results generalise well.

The graphs from ShuffleNetV2 show little to no deviation in the energy consumption for different batching strategies. Based on these results we draw the conclusion that this neural network is batch size invariant with regard to the energy consumption. As such, the greedy strategy is the most optimal because it limits the maximum response time.

Finally, the results for DenseNet highlight why we chose to evaluate each network separately. Larger batch sizes actively hurt the energy efficiency of this model, therefore the greedy strategy is the most optimal one.

Furthermore, we presented an energy efficiency timeline in Figure 4.3. In general, we find that the energy consumption of modern neural networks has increased steadily in the last ten years. ConvNext, the most recent publication, consumes more than twice as much energy as the revolutionary AlexNet from 2012. Nevertheless, our timeline has an irregularity that holds a great opportunity. ShuffleNetV2 is the only model in our timeline that does not adhere to the increasing energy trend. Additionally, when compared to its predecessors AlexNet and DenseNet, we find that ShuffleNetV2 does not perform any worse. We looked at the design principles that were considered when developing this network and argue that future work should incorporate the views and guidelines presented in the corresponding publication.

[9] https://anonymous.4open.science/r/green-ai-46E7
[10] https://pytorch.org/vision/stable/models.html

5

Implications

In this chapter, we acknowledge different kinds of stakeholders in AI research and practices. We highlight what implications can be drawn from our results for each of these parties.

5.1. To AI Practitioners

First and foremost, practitioners should be aware of the differences between Green AI and Red AI and the energy-efficient practices that we have laid bare in this thesis. As such, when developing and tuning new deep learning models, developers should look beyond the realm of baseline optimisation strategies and opt for more advanced techniques such as Bayesian optimisation. Another valid approach is to outsource this part of the training pipeline and implement existing solutions such as the population-based training algorithm from Ray Tune[1]. When working with fully-trained models, practitioners should consider the batch size that they want to use for inference. We have shown that different batch sizes can greatly influence the energy consumption and response time of a system.

5.2. To Software Engineers

Software engineers are already part of trans-disciplinary AI teams to enable the productionisation of AI models. We argue that the role of software engineers goes even further and is quintessential to realise energy-aware AI pipelines. One cannot ask regular AI practitioners or data scientists to engineer the collection of energy-efficiency metrics – software engineers must have the right knowledge and experience to help include energy as an important factor when developing AI pipelines.

5.3. To AI tool developers

Our results show that AI frameworks have to provide green alternatives. For example, there are not many options when selecting hyperparameter strategies. Moreover, there is no information regarding the energy efficiency of these alternatives. Hence, our results call for more energy-efficient options and better documentation with sustainability tips.

5.4. To Researchers

In the past four years, several works have emerged that call for a research agenda that considers energy efficiency in AI [1], [11], [12]. Past these positional papers, the number of hands-on studies is still very limited. Researchers should answer the call by building on our results w.r.t. hyperparameter optimisation and efficient network architectures, or explore new areas of energy-efficient practices. Documentation of empirical results is a valuable contribution. In the energy timeline from Section 4.5, we have shown that the consumption of our models is rising drastically, which calls for Green AI innovation, but we also shed light on existing work that could serve as a point of reference for further research.

[1] https://docs.ray.io/en/latest/tune/tutorials/tune-advanced-tutorial.html

5.5. To Tech Organisations

Large corporations are the biggest consumers in the field of AI. In this study, we have shown that the energy consumption of a deep learning model rises at a much faster pace than the performance. Tech organisations should make an effort to measure and report their energy consumption as a metric of equal importance to accuracy. This will change how we evaluate state-of-the-art deep learning models and encourage the development of Green AI. Since these large parties are the ones that distribute their models in masses, it is extra important to make sure that inference is as efficient as possible. A good starting point is to select the most optimal batch size for any specific network while considering the maximum response time as a secondary metric.

6

Future Research

In this chapter, we propose ideas for future work based on the observations presented in this thesis. The first study exists as a standalone research paper that was submitted to a conference. The reviewers were not convinced that the results presented with regards to the energy efficiency of different layers, would generalise well to neural networks other than CNNs. In our rebuttal, we argue that most network architectures consist of basic layers like the ones that we have explored, therefore we expect our findings to partially inform practitioners about the energy consumption of such architectures. Nonetheless, we did not investigate all the different building blocks. An interesting direction for future research would be to expand our experimentation and quantify the energy efficiency of other kinds of layers and cells. To give an example, we cannot make educated guesses on the consumption of layers that use memory cells, since these work differently from the studied structures. By improving our collective understanding of the energy consumption of these building blocks, we create opportunities for more efficient architecture to be introduced based on well-founded techniques.

Apart from extending and diversifying, upscaling the experiments would also a valuable contribution. All the data we gathered came from a single-machine setup with one GPU. In a more realistic setting, one can expect a large cluster of machines dedicated to deep learning tasks. These data centres are the largest source of energy consumption in the deep learning field. Therefore it is critical to learn whether the results that we found hold true and generalise well. To give an example, we recommend investigating the effect of different batching strategies on the energy consumption of an array of GPUs performing image classification tasks.

A final idea is to pursue neural network architecture design with energy consumption as a secondary quality metric next to accuracy. We have shown that modern network architectures consume much more energy than older models. In trying to obtain better results, these models become increasingly larger and computationally complex. Nevertheless, we think that this evolution is avoidable. By considering energy efficiency as a metric, we can lay bare the flaws in the current network architectures and provide space for new design principles to emerge.

7

Conclusion

This marks the end of my empirical study in the research field of Green AI. The work presented in this thesis adopts ideas, techniques and proposals from the limited collection of available literature, to uncover and emphasize energy-efficient practices in deep learning.

We first looked at rudimentary image vision networks during their training phase, with a focus on hyperparameter optimisation and architectural design. Software engineers in the multidisciplinary AI teams are responsible for the qualitative characteristics of the deep learning pipeline, such as its energy efficiency. Nevertheless, AI research is often conducted by large players with plenty of resources. Because of this, little regard is given to the kind of optimisation algorithms that are employed for tuning, which leads to unnecessary run times for training. Baseline algorithms such as random search and grid search are still very popular because they are comprehensible and easy to implement. Random search also has the additional benefit of being parallelisable out of the box. We compared these two algorithms to a classical variant of the Bayesian optimisation algorithm and found that the latter will consistently find more optimal hyperparameter values in less time. We conclude that baseline algorithms should exist for the purpose of early exploration, but should not be used when training power-hungry models.

Second, we mapped out the relation between different types of layers found in convolutional networks and the energy consumption. We found that both linear layers and ReLU layers have a statistically significant impact on the energy usage of a network. Nonetheless, this effect is largely overshadowed by that of the convolutional layers. The added computational complexity increases the utilisation of the GPU and the time it takes to process a single image. We reported an increase of up to 95% energy consumption for networks with four convolutional layers as opposed to networks with only one layer. Additionally, we observed that the deviation in accuracy follows a much more gentle slope. Because of this, our advice to the software engineering community is to evaluate the desired performance of their models on a case-by-case basis. They should then tune the complexity of these models, starting with the expensive convolutional layers, until the accuracy is within a reasonable margin.

In the next part of this thesis, we stepped away from the training phase and advanced to another part of the deep learning pipeline: the inference phase. We developed a simulation consisting of an image queue and a stream of incoming classification requests with variable frequency. Using this simulation, we visualised the three-way relation between batch size, energy consumption and wait time for five state-of-the-art neural networks. In contrast to the findings of Canziani et al. [22], our results generally show that batch size has a noticeable effect on the energy consumption. However, since all networks (AlexNet, DenseNet, ShuffleNetV2, VisionTransformer & ConvNext) respond very differently to the varying batch sizes, we can't give any concrete recommendations. Practitioners should consider our results when tuning their models to discover if they should use a small (1-31) or large batch size ($\geq$32). With regards to the wait time, we found that the relationship with the batch size is positive. This means that the batch size can't be too large if response time is important.

Finally, we use the results of this same experiment to highlight the deep learning energy trend of the last decade. The initial publication dates of the five networks we chose are all spread apart neatly to facilitate this analysis. Unsurprising, but significant nonetheless, we showed that the most recent model drains more than twice the amount of energy that AlexNet does. More generally, we presented a line plot that proves that this increase is mostly consistent and goes accompanied by very

marginal improvements in accuracy. The only exception to these findings comes from ShuffleNetV2. The developers of this model proposed an architectural design that optimises for speed and they largely succeeded in doing so without sacrificing performance. This shows that results do not necessarily have to come at the cost of energy efficiency and it paints an optimistic lead for future research.

Bibliography

[1] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, “Green ai,” *Communications of the ACM*, vol. 63, no. 12, pp. 54–63, 2020.

[2] S. Chowdhury, S. Borle, S. Romansky, and A. Hindle, “Greenscaler: Training software energy models with automatic test generation,” *Empirical Software Engineering*, vol. 24, no. 4, pp. 1649–1692, 2019.

[3] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, “Mining energy-greedy api usage patterns in android apps: An empirical study,” in *Proceedings of the 11th working conference on mining software repositories*, 2014, pp. 2–11.

[4] L. Cruz and R. Abreu, “Performance-based guidelines for energy efficient mobile applications,” in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, IEEE, 2017, pp. 46–57.

[5] L. Cruz, R. Abreu, and J.-N. Rouvignac, “Leafactor: Improving energy efficiency of android apps via automatic refactoring,” in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, IEEE, 2017, pp. 205–206.

[6] A. Banerjee and A. Roychoudhury, “Automated re-factoring of android apps to enhance energy-efficiency,” in *Proceedings of the International Conference on Mobile Software Engineering and Systems*, 2016, pp. 139–150.

[7] M. Linares-Vásquez, C. Bernal-Cárdenas, G. Bavota, R. Oliveto, M. Di Penta, and D. Poshyvanyk, “Gemma: Multi-objective optimization of energy consumption of guis in android apps,” in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, IEEE, 2017, pp. 11–14.

[8] L. Cruz and R. Abreu, “Catalog of energy patterns for mobile applications,” *Empirical Software Engineering*, vol. 24, no. 4, pp. 2209–2235, 2019.

[9] S. Chowdhury, S. Di Nardo, A. Hindle, and Z. M. J. Jiang, “An exploratory study on assessing the energy impact of logging on android applications,” *Empirical Software Engineering*, vol. 23, no. 3, pp. 1422–1456, 2018.

[10] D. Feitosa, R. Alders, A. Ampatzoglou, P. Avgeriou, and E. Y. Nakagawa, “Investigating the effect of design patterns on energy consumption,” *Journal of Software: Evolution and Process*, vol. 29, no. 2, 2017.

[11] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, “On the dangers of stochastic parrots: Can language models be too big?” In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 2021, pp. 610–623.

[12] E. Strubell, A. Ganesh, and A. McCallum, “Energy and policy considerations for deep learning in nlp,” *arXiv preprint arXiv:1906.02243*, 2019.

[13] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, “What do programmers know about the energy consumption of software?” *PeerJ PrePrints*, vol. 3, e886v1, 2015.

[14] C. Sahin, L. Pollock, and J. Clause, “How do code refactorings affect energy usage?” In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ACM, 2014, p. 36.

[15] R. Pereira, M. Couto, F. Ribeiro, *et al.*, “Ranking programming languages by energy efficiency,” *Science of Computer Programming*, vol. 205, p. 102 609, 2021.

[16] A. Hankel, G. Heimeriks, and P. Lago, “Green ict adoption using a maturity model,” *Sustainability*, vol. 11, no. 24, p. 7163, 2019.

[17] R. Verdecchia, P. Lago, C. Ebert, and C. De Vries, “Green it and green software,” *IEEE Software*, vol. 38, no. 6, pp. 7–15, 2021.

[18] J. De Macedo, R. Abreu, R. Pereira, and J. Saraiva, “On the runtime and energy performance of webassembly: Is webassembly superior to javascript yet?” In *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, IEEE, 2021, pp. 255–262.

[19] I. Malavolta, K. Chinnappan, S. Swanborn, G. A. Lewis, and P. Lago, “Mining the ros ecosystem for green architectural tactics in robotics and an empirical evaluation,” in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, IEEE, 2021, pp. 300–311.

[20] D. Li, X. Chen, M. Becchi, and Z. Zong, “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)*, IEEE, 2016, pp. 477–484.

[21] T.-J. Yang, Y.-H. Chen, and V. Sze, “Designing energy-efficient convolutional neural networks using energy-aware pruning,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5687–5695.

[22] A. Canziani, A. Paszke, and E. Culurciello, “An analysis of deep neural network models for practical applications,” *arXiv preprint arXiv:1605.07678*, 2016.

[23] H. Xiao, K. Rasul, and R. Vollgraf. “Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms.” arXiv: `cs.LG/1708.07747 [cs.LG]`. (Aug. 28, 2017).

[24] A. Krizhevsky, G. Hinton, *et al.*, “Learning multiple layers of features from tiny images,” 2009.

[25] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *Journal of machine learning research*, vol. 13, no. 2, 2012.

[26] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, “Algorithms for hyper-parameter optimization,” *Advances in neural information processing systems*, vol. 24, 2011.

[27] P. Liashchynskyi and P. Liashchynskyi, “Grid search, random search, genetic algorithm: A big comparison for nas,” *arXiv preprint arXiv:1912.06059*, 2019.

[28] A. Torralba, R. Fergus, and W. T. Freeman, “80 million tiny images: A large data set for non-parametric object and scene recognition,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 11, pp. 1958–1970, 2008.

[29] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, “Fast bayesian optimization of machine learning hyperparameters on large datasets,” in *Artificial Intelligence and Statistics*, PMLR, 2017, pp. 528–536.

[30] J. Snoek, H. Larochelle, and R. P. Adams, “Practical bayesian optimization of machine learning algorithms,” *Advances in neural information processing systems*, vol. 25, 2012.

[31] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, and S.-H. Deng, “Hyperparameter optimization for machine learning models based on bayesian optimization,” *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 26–40, 2019.

[32] H. J. Kushner, “A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise,” 1964.

[33] J. Mockus, V. Tiesis, and A. Zilinskas, “The application of bayesian methods for seeking the extremum,” *Towards global optimization*, vol. 2, no. 117-129, p. 2, 1978.

[34] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, “Gaussian process optimization in the bandit setting: No regret and experimental design,” *arXiv preprint arXiv:0912.3995*, 2010.

[35] P. Hennig and C. J. Schuler, “Entropy search for information-efficient global optimization.,” *Journal of Machine Learning Research*, vol. 13, no. 6, 2012.

[36] J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani, “Predictive entropy search for efficient global optimization of black-box functions,” *arXiv preprint arXiv:1406.2541*, 2014.

[37] W. Scott, P. Frazier, and W. Powell, “The correlated knowledge gradient for simulation optimization of continuous parameters using gaussian process regression,” *SIAM Journal on Optimization*, vol. 21, no. 3, pp. 996–1026, 2011.

[38] W. Ma and J. Lu, “An equivalence of fully connected layer and convolutional layer,” *arXiv preprint arXiv:1712.01252*, 2017.

[39] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a convolutional neural network,” in *2017 international conference on engineering and technology (ICET)*, Ieee, 2017, pp. 1–6.

[40] J. Cohen, *Statistical power analysis for the behavioral sciences*. Routledge, 2013.

[41] C.-J. Wu, R. Raghavendra, U. Gupta, *et al.*, “Sustainable ai: Environmental implications, challenges and opportunities,” *arXiv preprint arXiv:2111.00364*, 2021.

[42] G. Yeung, D. Borowiec, A. Friday, R. Harper, and P. Garraghan, “Towards {gpu} utilization prediction for cloud deep learning,” in *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*, 2020.

[43] J. v. Kistowski, H. Block, J. Beckett, K.-D. Lange, J. A. Arnold, and S. Kounev, “Analysis of the influences on server power consumption and energy efficiency for cpu-intensive workloads,” in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, 2015, pp. 223–234.

[44] X. Zhang, X. Zhou, M. Lin, and J. Sun, “Shufflenet: An extremely efficient convolutional neural network for mobile devices,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[45] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, “Shufflenet v2: Practical guidelines for efficient cnn architecture design,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 116–131.

[46] I. Kandel and M. Castelli, “The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset,” *ICT express*, vol. 6, no. 4, pp. 312–315, 2020.

[47] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, “Don’t decay the learning rate, increase the batch size,” *arXiv preprint arXiv:1711.00489*, 2017.

[48] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, 2012.

[49] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[50] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” *arXiv preprint arXiv:2010.11929*, 2020.

[51] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, “A convnet for the 2020s,” *arXiv preprint arXiv:2201.03545*, 2022.