The Power of Training: How Different Neural Network Setups Influence the Energy Demand

Daniel Geißler, Bo Zhou, Mengxi Liu, Sungho Suh, Paul Lukowicz

German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany
RPTU, Kaiserslautern, Germany

Abstract

This work examines the effects of variations in machine learning training regimes and learning paradigms on the corresponding energy consumption. While increasing data availability and innovation in high-performance hardware fuels the training of sophisticated models, it also supports the fading perception of energy consumption and carbon emission. Therefore, the goal of this work is to create awareness about the energy impact of general training parameters and processes, from learning rate over batch size to knowledge transfer. Multiple setups with different hyperparameter initializations are evaluated on two different hardware configurations to obtain meaningful results. Experiments on pretraining and multitask training are conducted on top of the baseline results to determine their potential towards sustainable machine learning.

Introduction

The connection between the complexity of a task being solved by a deep neural network and the accompanying energy requirement from powerful hardware to train and deploy such models is a well-known issue of machine learning algorithms. Improving availability and access to more data, which forms the basis for training complex models, supports the trend towards enhanced hardware power and energy consumption further. Next to the increase in computational demands, there is a growing awareness of a less visible but increasingly significant aspect: the carbon footprint for training machine learning models. (Henderson et al. 2020; Patterson et al. 2022)

Multiple projects like carbontracker (Anthony, Kanding, and Selvan 2020) or eco2AI (Budennyy et al. 2022) apply to this problem statement, aiming to track the energy consumption and the related carbon footprint for training a model by linking into the hardware power utilization logging during the model training. Although these tools can monitor energy consumption, they are unable to compare the efficiency of the training process as well as suggesting recommendations for improvements.

In this work, we aim to create awareness about the significance of general adjustments in the training process and their major effects on energy consumption. Moreover, a goal of this work is to strive for a general analysis that can be applied to other projects as well. Additionally, we examine the effects of advanced learning paradigms to project our baseline findings from standard training regimes into state-of-the-art practices by comparing their energy consumption on our test setups.

Even though the innovation in terms of efficiency and sustainability of machine learning algorithms lags behind quantitative growth of hardware resources, small improvements in the low percentage range already help to reduce the carbon footprint if such methods are scaled properly within the landscape of machine learning applications. (Patterson et al. 2022)

Related Works

Carbon tracking software tools are usually designed to capture as much power information about the system as possible by building an additional layer between the system's hardware configuration and the user's model training process. Generally speaking, the power consumption of the GPU is the largest part of the training process as it performs the core work with the parallel processing of mathematical tasks. Following that, usually, the CPU is the second largest consumer of power.

There is a growing list of software available with most of them utilizing the same approach to gather the power consumption data from the hardware manufacturers' utility logging. The energy consumption is then calculated from the current power consumption and the polling time interval set in the software. Works like Carbontracker (Anthony, Kanding, and Selvan 2020), eco2Ai (Budennyy et al. 2022) and Green Algorithms (Lannelongue, Grealey, and Inouye 2021) utilize this approach to gather data from CPU, GPU, and even the RAM if supported. Nevertheless, there is a lack of tracking the full systems' energy consumption. High-performance setups may consume more power than present tools can track, for instance, if the cooling, which can account for a non-negligible proportion, is not taken into account. Therefore, software like Carbontracker (Anthony, Kanding, and Selvan 2020) multiplies its results with an efficiency constant of 1.55 to incorporate untracked secondary power needs and efficiency losses.

With an extended focus on user experience, projects like Cloud Carbon (cloud-carbon footprint 2023) or CodeCarbon (Code-Carbon 2023) extend the gathered knowledge and present it in analytic-based dashboards. Based on the calculated energy consumption and the user's location, the average local energy mix from fossil and renewable energy sources is utilized to estimate the carbon emissions in kilograms or even tons. (Lacoste et al. 2019) On top of that, since the carbon emissions are difficult to visualize or imagine, the conversion into kilometers driven by car, flights with a plane, or the number of phones charged is a standard practice to make the user aware of the generated carbon emissions amount.

Methodology

Experiment Setup

In order to provide a meaningful correlation between the training setup of a model and its power consumption, we chose two application scenarios that determine the structure of this work: computer vision and sensor-based activity recognition. We used two commonly used benchmark datasets from two disciplines: for a computer vision task we used the CelebA dataset (Liu et al. 2015) and for the sensor-based activity recognition, we utilized the PAMAP2 dataset (Reiss and Stricker 2012) as a basis.

We run each experiment two times on two different hardware configurations to reduce hardware-specific implications from our results. The already introduced Carbontracker (Anthony, Kanding, and Selvan 2020) is used to track the GPU power consumption on a workstation powered by an Nvidia RTX 6000 Ada with 48GB VRAM and AMD Ryzen 9 7950X CPU. A 16-inch Apple Macbook Pro with M1 Max chip with 32-core GPU and 64GB unified memory was used as a second device. The Nvidia GPU workstation runs on CUDA 12.3 and the Apple device runs on metal performance shader (MPS) for neural network training acceleration with pytorch. We implemented a custom tracker based on the *powermetrics* tool of Mac OS to generate the same data structure as the Carbontracker. Due to Carbontracker's limitations on tracking CPU power consumption for Intel CPUs only, we distinguished the gathering of GPU and CPU power consumption on the Apple setup as well. As a result, both systems were set up to gather the power consumption of the utilized GPU during the training process in milliwatts precision with a sampling rate of one second. Additionally, the trackers stored the corresponding timestamp and the current epoch to properly evaluate the data. The efficiency constant multiplication was disabled for out testings to achieve comparable measurements.

Since each hardware setup has deviating power capabilities and the power is not representative in terms of energy consumption, we decided to calculate the epoch-wise energy consumption as the baseline following the equation:

$$\forall\, k \in [0, T] \quad E_k = \frac{\sum_{i=0}^{n} power(k, n)}{n} * \frac{time(k)}{3600} \tag{1}$$

Throughout the whole experiment and different hardware setups, we kept the same model and training script versions. Moreover, the changed parameters across training runs were induced as external arguments. For training, we used early stopping techniques based on validation loss for the encoder and the validation accuracy for the classifier as a standard metric.

Training regime

Our test procedure can be separated into two tasks which we applied to the two application scenarios, the training regime and the learning paradigm.

A common problem when training a model is the proper initialization of hyperparameters that fit the task or problem. The two most prominent ones are the batch size and the learning rate (He, Liu, and Tao 2019). To determine the influence of hyperparameters, we created an encoder plus classifier architecture for the CelebA to classify the attributes and identities. The encoder is ResNet18 from the TIMM python package with pre-trained ImageNet weights. For the PAMAP2, we build a similar convolutional classifier architecture inspired by (Suh, Rey, and Lukowicz 2023) to classify the activities. For the training regime, we trained the models with variations in batch size and learning rate, resulting in a total of 16 different setups as shown in table 1.

Setup	Batch Size	Learning Rate
1	64	0.1
2	64	0.01
3	64	0.001
4	64	0.0001
5	256	0.1
6	256	0.01
7	256	0.001
8	256	0.0001
9	1024	0.1
10	1024	0.01
11	1024	0.001
12	1024	0.0001
13	4096	0.1
14	4096	0.01
15	4096	0.001
16	4096	0.0001

Table 1: Training Regime Hyperparameter Setup List

Learning Paradigm

On top of the training regime tests, we extended the experiments to explore the energy consumption of different learning paradigms focusing on knowledge sharing and reusing. In principle, these learning paradigms can potentially reduce the energy cost compared to the baseline of training a randomly initialized, black-box model for each dataset or task. Instead, the research community focuses on the generalization abilities or accuracy improvements of these learning paradigms; the actual energy consumption has not yet been evaluated.

With the PAMAP2 dataset, we evaluate the **pretraining** learning paradigm, shown in figure 1. We train an autoencoder and use its encoder as a pre-trained part to train a classifier on its latent features and compare it to the training of the encoder and classifier architecture sequentially. This practice is typically used to first force the feature encoder to learn information that can be used to reconstruct the input, thus the features are unique to different characteristics of the input data. This could reduce the likelihood of over-fitting on the ground truth labels of the training data and thus provide a better generalized model. We also inspect the differences between freezing and unfreezing the pre-trained encoder weights during the usage for generating the latent features.

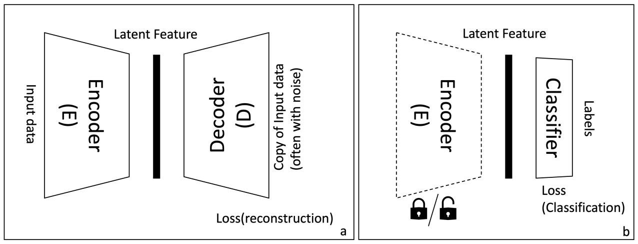


Figure 1: Pretraining an autoencoder and utilizing the encoder in frozen and unfrozen mode to generate latent features for the training of a classifier.

With the CelebA dataset, we can evaluate **multi-task learning** (Yin and Liu 2017), since the datasets have different aspects of ground truth for the same input (identity and facial attributes). Opposed to training a complete black-box-like neural network for each task, multi-task learning shares the feature encoder with multiple downstream tasks (e.g. different classifiers). The procedure is visualized in figure 2 a and b for the single training setup and the multitask setup in c by connecting the two classifiers via the weight coefficient α within the loss function.

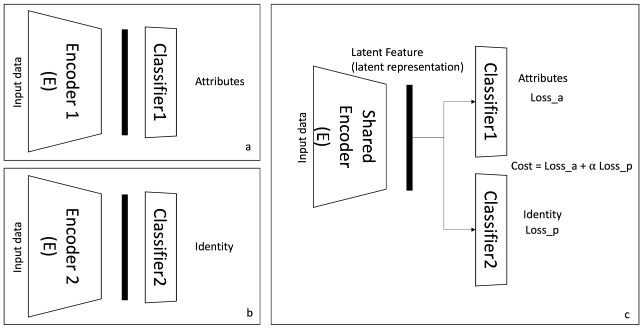


Figure 2: Comparing the single training setup with the multitask setup (combined loss function).

Training Regime Results

For testing the training regime on the two application scenarios, the standard versions of their tasks are utilized by excluding the advanced setups so far. For PAMAP2, this is the training of an encoder and a decoder in one run without pretraining, whereas for the CelebA scenario, a single classifier is trained to predict the identities only. Each scenario is trained with 16 different hyperparameter combinations and run two times on the two different hardware setups.

	Learning Rate			
Batch Size	0.1	0.01	0.001	0.0001
64	109.63	110.42	107.50	108.37
256	42.34	40.20	40.19	38.41
1024	30.84	30.97	30.87	30.81
4096	38.57	37.13	37.44	38.22

(a) PAMAP2 on CUDA (mWh)

	Learning Rate			
Batch Size	0.1	0.01	0.001	0.0001
64	38.40	39.25	38.73	38.10
256	39.03	40.19	39.91	39.34
1024	43.52	43.98	44.26	42.98
4096	43.00	43.97	44.01	43.09

(b) CelebA on CUDA (mWh)

	Learning Rate			
Batch Size	0.1	0.01	0.001	0.0001
64	15.40	15.63	15.96	15.66
256	16.83	16.79	16.76	25.35
1024	25.31	25.35	25.45	25.57
4096	27.33	27.17	27.11	27.24

(c) PAMAP2 on MPS (mWh)

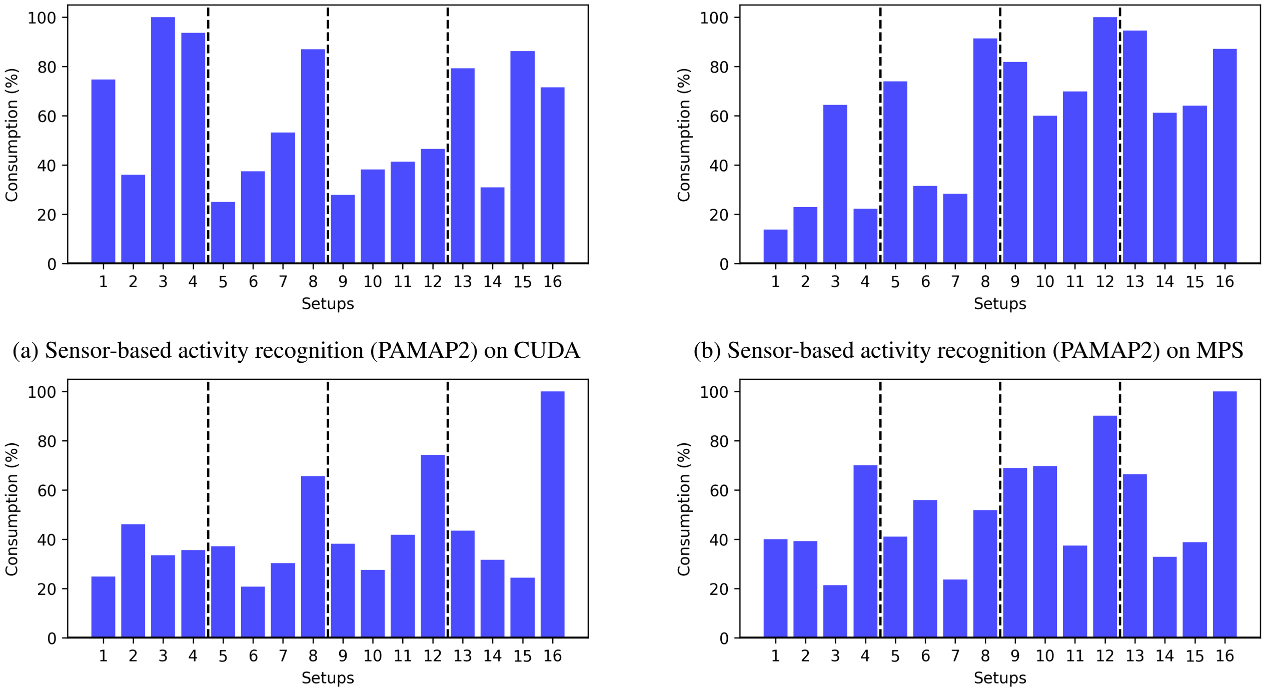
	Learning Rate			
Batch Size	0.1	0.01	0.001	0.0001
64	22.81	22.78	22.75	22.79
256	22.29	22.28	22.31	22.28
1024	22.37	22.34	22.25	22.39
4096	22.39	22.29	22.28	22.37

(d) CelebA on MPS (mWh)

Table 2: Average Energy per Epoch (mWh) across all setup combinations.

The average energy that was demanded during the training is shown in table 2, calculated through the introduced equation 1. As a general rule, the level of energy used can be linked to the efficiency and utilization of the GPU. Lower values mean less energy consumed for training one epoch, which results in improved GPU efficiency.

For each matrix shown in table 2, the batch size choice has the most impact on the energy consumption. There needs to be a balance between the dataset size and the GPU capabilities. On CUDA (matrix 2a and 2b), the PAMAP2 dataset performs best for bigger batch sizes due to its slimmer data structure size whereas the bigger image dataset CelebA performs best for smaller batches. This is mainly ruled by the GPU utilization, but influenced by the memory bandwidth limitations as well. Similar results can be found for the same runs on MPS (matrix 2c and 2d), keeping in mind the less powerful system. Therefore, the best batch size choice moves towards the smaller batch sizes for the PAMAP2 sce-



(a) Sensor-based activity recognition (PAMAP2) on CUDA

(b) Sensor-based activity recognition (PAMAP2) on MPS

(c) Identification from images (CelebA) on CUDA

(d) Identification from images (CelebA) on CUDA

Figure 3: Average consumption of each setup across all epochs by using the worst performing setup as baseline.

nario and even stays constant for the bigger CelebA dataset due to the system resource limitations.

The choice of learning rate does not have a recognizable influence in this analysis. There is little to no change in the energy per epoch across the four learning rate settings. The reason for that is the independence of the training duration, namely the number of epochs until the model converges and stops the training process. Influences through the learning rate change can therefore be seen in figure 3. Across the 16 setups, grouped into the 4x4 setup alignment through the dashed lines as introduced in table 1, the energy consumption in percentage is visualized, assuming the worst run to be at 100%. We can derive a slight trend of learning rate settings towards 0.01 and 0.001, mainly for the bigger batch sizes. This can be due to the issue of learning rate 0.1 being set too large to converge properly whereas learning rate of 0.0001 increases the number of epochs needed until convergence. Additionally, the batch size seems to have a slight influence on the total energy consumption as well. For instance in figure 3b and 3c, the combination of a small batch size of 64 and the biggest learning rate of 0.1 outperforms the other setups using bigger batch sizes.

Overall, the range between the worst and best-performing setup is wide, with the best model usually consuming only around 20% of the worst model. As a guideline to achieve this much of an improvement, the energy per epoch and the number of epochs necessary need both to be minimized.

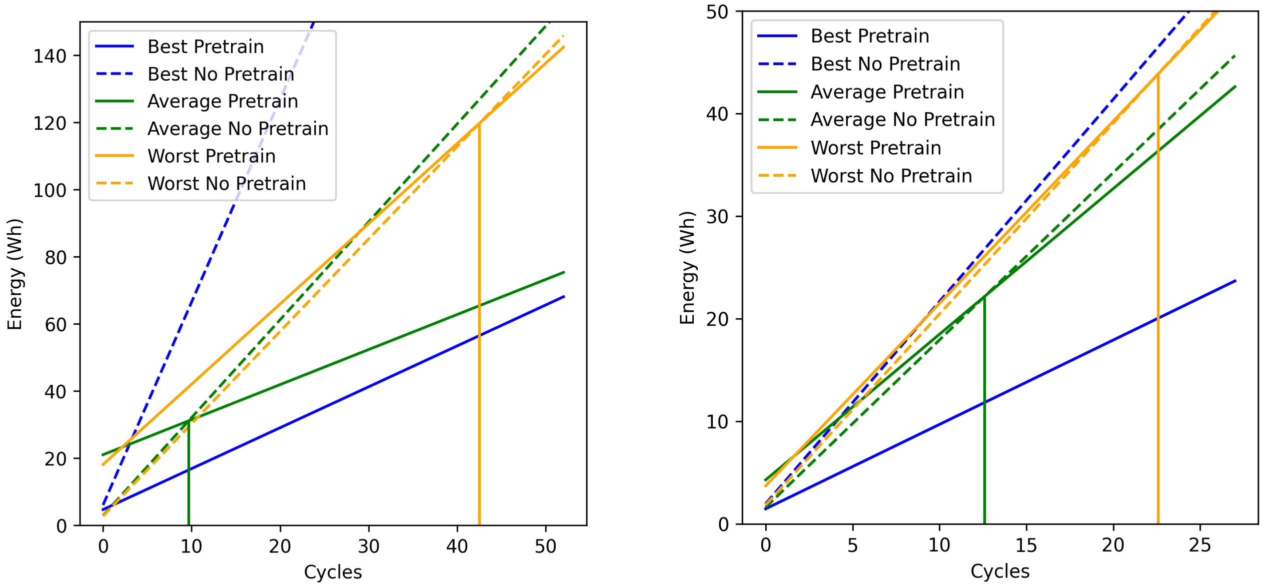
Pretraining Learning Paradigm Results

For the pretraining scenario of PAMAP2, the autoencoder's energy consumption for training the encoder was tracked first. On top of that, the consumption for training a classifier based on the latent features was tracked as well. As a comparison, we trained the encoder and classifier architecture together without pretraining. In total, we trained each model again for all 16 setups.

To compare the results, we generated equation 2 to calculate the breakeven point on how many times the pre-trained model needs to be recycled to compensate its footprint compared to training the full architecture every time.

$$\begin{aligned} &E_{Enc} + x * E_{Class} = x * E_{Enc+Class} \\ &x = \frac{E_{Enc}}{(E_{Enc+Class} - E_{Class})} \\ &\text{with x = Number of Cycles till Compensation} \end{aligned} \tag{2}$$

The results are shown in figure 4 for CUDA and MPS training. In this case, the results are solely based on the frozen encoder, so the optimization process does not consider the encoder anymore. The consumption for the pretraining run is shown in solid lines whereas the basic training without encoder pretraining is a dashed line. The crossing of two lines of the same color indicates the break-even point, which means with how many complete downstream-task training cycles, the investment in the pretraining starts to yield better energy savings in the longer perspective. The



(a) PAMAP2 on CUDA

(b) PAMAP2 on MPS

Figure 4: Break-even points, highlighting the amount of cycles until the pre-taining energy investment is compensated.

ideal situation is that the break-even point appears as early as possible. From the 16 setups utilized, we extracted the best, worst, and average scenarios. As we can see, the blue lines for CUDA and MPS both show a scenario where the pretraining already outperforms the standard training after one recycling step. On the other hand, for the worst case, visualized with orange, it takes up to 43 cycles for the CUDA run and up to 23 cycles on MPS until the spent pretraining energy is compensated. As we can derive from equation 2, the compensation depends on the increased energy consumption of the encoder pretraining and the ratio between training the classifier alone or with the encoder together. On average, the energy is compensated after around 10 to 13 recycling steps of the pre-trained encoder.

Additionally, we investigated the average energy consumption between freezing and unfreezing the encoder after its pretraining run. (Marcelino 2018) Since the optimizer does not update the weights during the backpropagation to finetune the encoder, there is less energy necessary to train the classifier. As shown in figure 3, there is on average an energy saving of 61.5 % for the CUDA experiment and 46.1% on MPS across all 16 setup combinations. Since proper pretraining ensures the required encoders' behavior, the freezing of weights can be a helpful measure to reduce energy consumption.

Multitask Learning Paradigm Results

For the multitask scenario trained on CelebA, we utilize the ResNet18 architecture pre-trained on ImageNet as the encoder. Due to ImageNets size and required training time, we decided to recycle an already existing, pre-trained model. As a side note, the best option to save energy in machine learning is the usage of already existing, pre-trained models to compensate their energy consumption further.

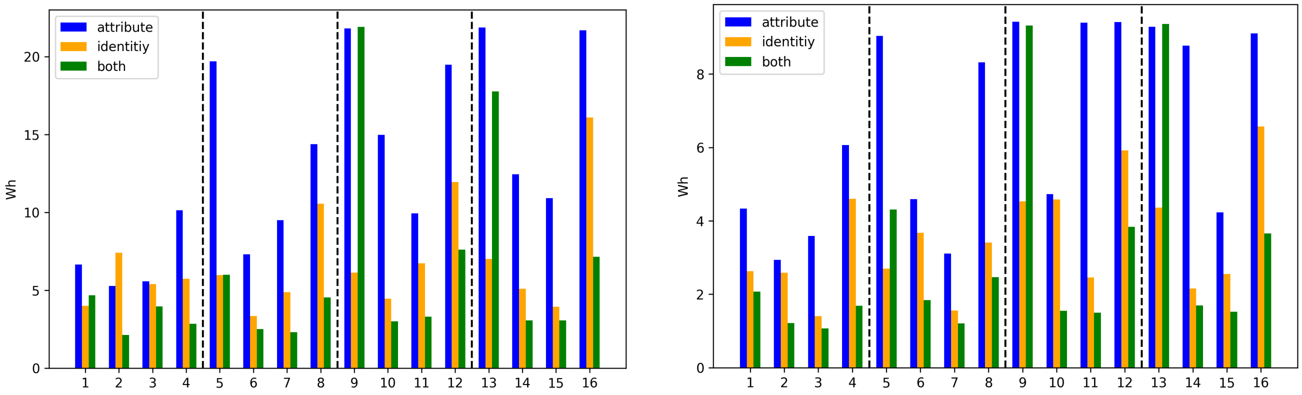
	Hardware	
Encoder	CUDA	MPS
Unfreeze	4.26	1.42
Freeze	1.64	0.76
Savings	61.5%	46.1%

Table 3: Average Energy consumption (Wh) for utilizing the pre-trained encoder in frozen or unfrozen mode.

For our experiment, the nature of the CelebA dataset with multiple properties labeled per image, we focus on training a multitask classifier and measuring the energy consumption including the pre-trained encoder. More specifically, we train a classifier to classify 40 facial attributes and the 40 most represented person identities of a CelebA image.

The experiment was conducted through the same procedure as the pretraining scenario, training the three models (attribute, identity, and both together) with the 16 different setups. The results are shown in figure 5. For all tested setups, the green bar for training both classifiers in one go is considerably lower than the cumulative energy for training the attribute and identity separately. Even more, the multitask training results in a lower energy consumption than training one single task classifier.

In order to interpret the results, we further calculated the average energy per epoch across the three scenarios as shown in figure 4. Since the average energy per epoch is almost identical across training attributes, identity, or both, no matter if trained on the CUDA or MPS system, we further calculated the average length of the training in number of epochs. As we can see, for this experiment the number of



(a) CelebA on CUDA

(b) CelebA on MPS

Figure 5: Energy Consumption for training attribute and identity classifier separately and training it through the multitask model.

epochs rules the overall energy consumption whereas the efficiency from energy per epoch is negligible. Since the loss functions are connected during the training in the multitask scenario, the classifiers of attribute and identity are able to cross-regulate each other, which results in a quicker convergence. The result manifests our previous findings for minimizing either one or at best both, the energy per epoch and the overall number of epochs used till convergence.

	Hardware	
Type	CUDA	MPS
Attribute	22.47 / 368	40.87 / 359
Identity	22.46 / 150	41.09 / 161
Both	22.44 / 142	41.36 /145

Table 4: Average Energy per Epoch (mWh) / Average number of Epochs

Conclusion

In conclusion, we investigated the energy consumption of two application scenarios on two different hardware configurations. Across all experiments, we followed the approach of training the selected architectures with variations in batch size and learning rates. For the 16 selected setups, we first evaluated their influence on the efficiency through energy per epoch and further the training duration through the overall energy consumption until the early stopping monitor criteria were met. As a result, the batch size influences the energy efficiency whereas the learning rate determines the training duration. The optimal configuration has to maximize the efficiency per epoch and minimize the training duration by setting the optimal batch size and learning rate. If this is the case, such optimizations can lead to energy savings of up to 80% in our tests.

For the learning paradigm testings, the calculated break-even point is an indicator of how and at which recycling iteration the initial energy investment for the pretraining is compensated. A long recycling process can be necessary if too much energy is invested into the pretraining or if the savings are too weak for each recycling step. Lastly, for the multitask training, we not only showed that training two classifiers in one training process is less energy-consuming, we additionally showed that fewer epochs are necessary due to the mutual training support of the classifiers.

References

Anthony, L. F. W.; Kanding, B.; and Selvan, R. 2020. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. *arXiv preprint arXiv:2007.03051*.

Budennyy, S. A.; Lazarev, V. D.; Zakharenko, N. N.; Korovin, A. N.; Plosskaya, O.; Dimitrov, D. V.; Akhripkin, V.; Pavlov, I.; Oseledets, I. V.; Barsola, I. S.; et al. 2022. Eco2ai: carbon emissions tracking of machine learning models as the first step towards sustainable ai. In *Doklady Mathematics*, volume 106, S118–S128. Springer.

cloud-carbon footprint. 2023. Cloud Carbon Footprint. https://github.com/cloud-carbon-footprint/cloud-carbon-footprint.

Code-Carbon. 2023. Code Carbon. https://github.com/mlco2/codecarbon.

He, F.; Liu, T.; and Tao, D. 2019. Control batch size and learning rate to generalize well: Theoretical and empirical evidence. *Advances in neural information processing systems*, 32.

Henderson, P.; Hu, J.; Romoff, J.; Brunskill, E.; Jurafsky, D.; and Pineau, J. 2020. Towards the systematic reporting of the energy and carbon footprints of machine learning. *The Journal of Machine Learning Research*, 21(1): 10039–10081.

Lacoste, A.; Luccioni, A.; Schmidt, V.; and Dandres, T. 2019. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*.

Lannelongue, L.; Grealey, J.; and Inouye, M. 2021. Green algorithms: quantifying the carbon footprint of computation. *Advanced science*, 8(12): 2100707.

Liu, Z.; Luo, P.; Wang, X.; and Tang, X. 2015. Deep Learning Face Attributes in the Wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.

Marcelino, P. 2018. Transfer learning from pre-trained models. *Towards data science*, 10: 23.

Patterson, D.; Gonzalez, J.; Hölzle, U.; Le, Q.; Liang, C.; Munguia, L.-M.; Rothchild, D.; So, D. R.; Texier, M.; and Dean, J. 2022. The carbon footprint of machine learning training will plateau, then shrink. *Computer*, 55(7): 18–28.

Reiss, A.; and Stricker, D. 2012. Introducing a new benchmarked dataset for activity monitoring. In *2012 16th international symposium on wearable computers*, 108–109. IEEE.

Suh, S.; Rey, V. F.; and Lukowicz, P. 2023. TASKED: Transformer-based Adversarial learning for human activity recognition using wearable sensors via Self-KnowledgE Distillation. *Knowledge-Based Systems*, 260: 110143.

Yin, X.; and Liu, X. 2017. Multi-task convolutional neural network for pose-invariant face recognition. *IEEE Transactions on Image Processing*, 27(2): 964–975.