Energy-Efficient CNN inferencing on GPUs with Dynamic Frequency Scaling

Rolf Drechsler, Christopher A. Metz and Christina Plump

Abstract To ensure that emerging technologies such as autonomous driving and application-specific Internet of Things devices work correctly, fast and accurate calculations must be performed by algorithms like *Machine Learning* (ML). One essential algorithm in these systems is *Convolutional Neural Network* (CNN), which requires a lot of computational resources. Designers often use ML accelerators like *General Purpose Graphic Processing Units* (GPGPUs) to keep up with design requirements, but choosing the right accelerator and accelerator configuration can be time-consuming and difficult. Our research analyzes the power consumption and execution time of CNNs on GPGPUs with different frequency settings. We found that changing the frequency significantly impacted power consumption but only had a marginal effect on computation time. Furthermore, increasing the frequency beyond 1200 MHz shows no improvement in computation time anymore. Therefore, a lower frequency can help create an energy-efficient CNN inference system without sacrificing performance.

Key words: Convolutional Neural Network (CNN), Graphic Processing Unit (GPU), energy-efficient, frequency scaling, inferencing

Rolf Drechsler
University of Bremen/DFKI GmbH, Bremen, e-mail: drechsler@uni-bremen.de

Christopher A. Metz
University of Bremen, Bremen e-mail: cmetz@uni-bremen.de

Christina Plump
DKFI GmbH, Bremen, e-mail: cplump@uni-bremen.de

1 Introduction

The use of *Machine Learning* (ML) is widespread, ranging from self-driving cars [1] to medical imaging like MRI [7]. Even ordinary people can access *Natural Language Processing* (NLP) through platforms like ChatGPT [3]. However, the hardware and framework required for these applications differ significantly. *Artificial Intelligence* (AI) and ML systems can be as small as embedded microprocessors or as large as supercomputers. When designing these systems, one must consider the trade-off between performance and power consumption. Small systems embedded with batteries cannot match the performance of a supercomputer due to power limitations.

To optimize the performance of ML/AI applications, current *General Purpose Graphics Processing Unit* (GPGPU) are explicitly designed for this purpose. However, the power consumption of individual GPGPUs continues to increase, with models like the NVIDIA H100 consuming up to 700 watts [16]. In contrast, the NVIDIA V100 only consumes up to 250 watts. To achieve sustainable ML/AI systems, an energy-efficient design, implementation, and operation are necessary.

When designing IoT or edge devices for ML/AI applications with limited power supply, it is crucial to perform *Design Space Exploration* (DSE) to find a balance between power consumption and performance [15]. Power management techniques like *Dynamic Voltage and Frequency Scaling* (DVFS) or *Dynamic Frequency Scaling* (DFS) are effective ways to achieve this balance. DVFS adjusts the voltage/frequency during application processing, while DFS only modifies the frequency. Both techniques optimize energy consumption and performance. Although DVFS/DFS techniques for CPU-based applications are well-established, research on these techniques for GPGPU only started a decade ago [12, 4].

NVIDIA offers the option to set a frequency cap due to their tool *nvidia-smi*. Studies [13, 18] illustrate that frequency caps can positively affect power consumption. However, they also indicate that adjusting the frequency settings of GPUs does not significantly improve the execution time of *Deep Learning* (DL) applications [13]. Thus, configuring a frequency cap is a promising method to reduce power consumption. As a drawback, those frequency caps result in unconventional and unpredictable behavior regarding the execution time of various applications [18, 13].

In [18] the impact of power and frequency caps are investigated on general high-performance applications but not on the inferencing of neural networks. They pointed out that using the frequency caps configured with *nvidia-smi* results in large variations in the execution time. Consequently, it is not possible to estimate the execution time of an application for multiple executions with the same frequency caps.

[5] analyzes the impact of DVFS on the NVIDA K20. They discovered that when using a GPU for compute-bound tasks requiring high performance and throughput, the system's performance and power consumption are closely

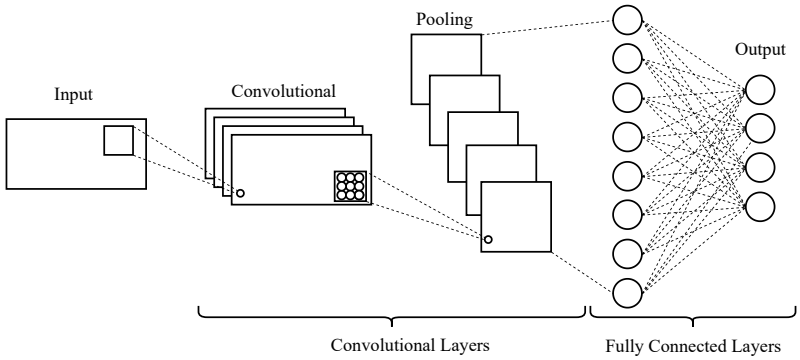

Fig. 1 Overview of CNN architecutre adjusted from [14]

tied to the GPU frequency. This means that increasing the GPU frequency results in improved performance without significantly increasing energy consumption as long as the power limit is not exceeded. This contrasts our experimental results related to the newer NVIDIA V100S. We could spot an increase in power consumption, especially in higher frequencies.

The work in [13] only investigated the impact of DFS on power consumption on CNNs. We extend the analysis to performance and investigate both power consumption and performance of CNN inferencing with DFS. Our main findings can be summarized as follows:

- There is almost no correlation between performance and frequency on most CNN inferencing task.
- There is a strong correlation between power consumption and frequency.
- The power consumption is strongly increasing for frequencies larger than 1200 MHz.
- Lower frequencies do not lead to significant performance loss but reduce power consumption.

This work is structured as follows: Section 2 explains the necessary background to CNNs and GPGPUs. Next, Section 3 illustrates the Methodology. The experimental setup and results are described in section 4. The work closes with a conclusion and future work in Section 5

2 Background

In this section, we explain some background and introductory concepts of CNNs and GPGPUs that are necessary to understand the proposed analysis.

2.1 Convolutional Neural Networks

Convolutional Neural Nrtworks (CNNs) are a specialized type of neural network mainly used for image data but can also be used on time-series data. As the name *Convolutional Neural Network* indicates, these networks use a mathematical operation called *convolution* [6, 7].

When it comes to a convolutional layer, it typically involves three stages - *convolution*, *activation*, and *pooling*, illustrated in Fig. 1. Multiple convolutions are carried out in parallel during the convolution stage, as demonstrated in Fig. 2.

The activation stage involves applying a linear activation function, such as the *Rectified Linear Unit* (ReLU) activation function. Finally, the optional pooling stage employs a pooling function, such as max-pooling [6]. For instance, Alexnet has six convolution layers and three max-pooling layers [10]. The max-pooling function selects the highest value within a $n \times m$ sized kernel or window.

The convolution calculation and conceptual design of CNNs are well-suited for parallel execution and benefit significantly from GPGPU's massive parallelization capabilities [6].

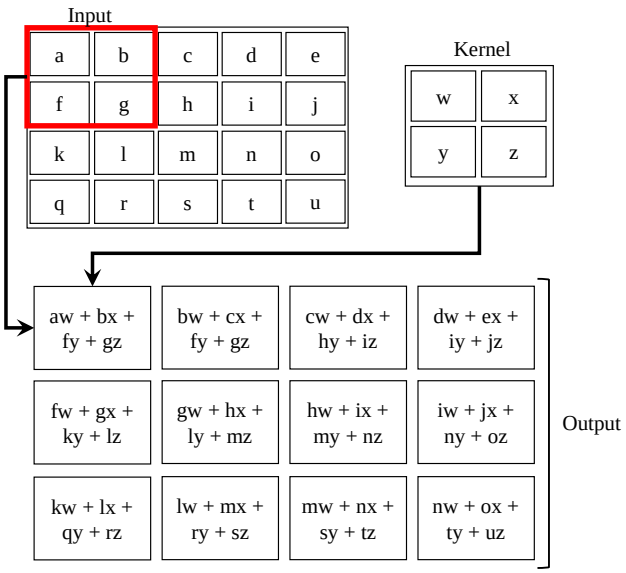

Fig. 2 Convolutional operations for a 2D convolution with a 2x2 Kernel. The figure does not consider any bias used for the convolutional layer based on [6, p. 330].

2.2 General Purpose GPU

The architecture of *Graphics Processing Units* (GPUs) is considerably more intricate than that of conventional *Central Processing Units* (CPUs), involving an adaptable quantity of *Streaming Multiprocessors* (SMs), as depicted in Figure 3 for the NVIDIA V100. Each SM is subdivided into four *Processing Units* (PUs) that work to optimize GPU efficiency. For the NVIDIA V100, each of the four PUs are equipped with an L0 instruction cache, one Warp Scheduler, one dispatch unit, a 64KB register file, 16 *Floating Point* (FP) 32 cores, 8 FP64 cores, 16 INT32 cores, and one tensor core (with mixed-precision tensor cores for deep learning) [8, 17].

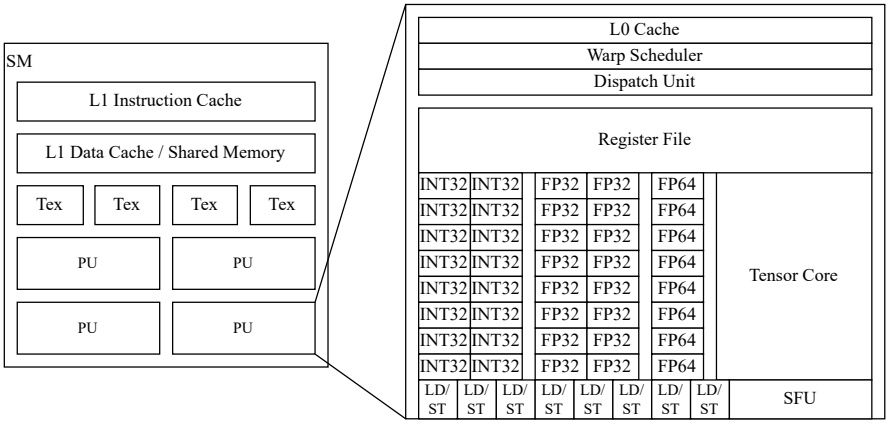

Fig. 3 Overview of streaming multiprocessor architecture adjusted from [17].

The GPGPU application, based on CUDA, is divided into multiple kernels[1]. These kernels are compiled and divided into *Cooperative Thread Arrays* (CTAs) or thread blocks. A CTA is further divided into groups of 32 threads, known as a warp, and all threads in a warp perform the same instruction following the *Single Instruction Multiple Data* (SIMD) principle [19]. However, NVIDIA refers to it as *Single Instruction Multiple Threads* (SIMT). Managing warps may be unfamiliar to some programmers; they can only control the total number of threads, but it is a crucial aspect of the process [9]. Programmers write a program for a single thread and specify the number of parallel executions. The warp scheduler locates the warps on the system without the control of the programmers. The number of warps that can run concurrently on an SM depends on their resource requirements, such as the number of registers or shared memory usage.

[1] Please note that the mentioned kernels are not affiliated with those of the CNN concept.

The warp-synchronous programming technique (i.e., SIMT) is softening with the Volta architecture. Thus, threads can execute different instructions within a warp. This makes the process more efficient and streamlined, enhancing its effective programming method [11, 19]. This results in fewer idle threads generated by divergent branches.

3 Method

We conducted a study to determine how frequency scaling impacts the efficiency of CNN inferencing tasks. Our study carefully examined the power consumption and computation time of inferencing tasks on multiple CNNs with different frequencies. In the following, we describe the data-gathering process and the evaluation methods.

3.1 Data Gathering

For our analysis, we generated different CNNs, same as in [13]. As Table 1 illustrates, the CNNs differ in size (number of hidden layers and neurons) and input image size. However, it should be noted that some CNNs share the same input size. That can be attributed to all these CNNs being trained based on the ImageNet dataset [2].

The CNNs are downloaded pre-trained from TensorflowHub[2] to ensure the correct implementation of each CNN. Afterward, we execute each CNN as a single batch run, with the prediction of one image at a time, on a machine equipped with the NVIDIA V100S.

Before each execution, we use the *nvidia-smi* tool to set the frequency of the NVIDIA V100S to a fixed value. Figure 4 illustrates using the *nvidia-smi* frequency cap. The NVIDIA V100S offers a wide range of frequency settings, from 135 MHz to 1597 MHz. However, the memory frequency remains fixed at 1107 MHz for this particular GPU model and cannot be adjusted during benchmarks. We perform only one CNN per run to ensure accurate results and reset the GPU settings after each execution. This returns the GPU to its default state before starting a new run. Additionally, we repeat each CNN and frequency combination three times to determine the average execution time and power consumption.

To measure the power consumption, we utilize the internal power sensors of NVIDIA V100S. We consistently measure the power consumption and keep track of the highest power consumption during CNN executions. Additionally, we measure the execution time for each CNN and frequency setting,

[2] `https://www.tensorflow.org/hub`

Table 1 An overview of CNN models used in the experiments

CNN	Input Size	Layers	Neurons
m-r50x1	*224* × *224*	50	15,903,016
m-r50x3	*224* × *224*	50	143,111,080
m-r101x3	*224* × *224*	101	25,3408,168
m-r101x1	*224* × *224*	101	28,158,248
m-r154x4	*224* × *224*	154	611,981,544
densenet121	224 × 224	121	49,926,612
densenet169	224 × 224	169	60,094,164
densenet201	224 × 224	201	77,292,244
efficientnetb0	224 × 224	240	25,117,095
efficientnetb1	240 × 240	342	40,150,331
efficientnetb2	260 × 260	342	50,908,981
efficientnetb3	300 × 300	387	87,507,971
efficientnetb4	380 × 380	477	180,088,531
efficientnetb5	456 × 456	579	358,290,427
efficientnetb6	528 × 528	669	605,671,091
efficientnetb7	600 × 600	816	1,046,113,195
inceptionresnetv2	299 × 299	164	81,201,907
inceptionv3	299 × 299	48	32,554,387
mobilenet	224 × 224	28	16,848,248
mobilenetv2	224 × 224	53	21,815,960
nasnetlarge	331 × 331	1041	290,560,171
nasnetmobile	224 × 224	771	27,690,705
resnet101	224 × 224	101	55,886,036
resnet101v2	224 × 224	101	51,261,140
resnet152	224 × 224	152	79,067,348
resnet152v2	224 × 224	152	75,755,220
resnet50v2	224 × 224	50	31,381,204
vgg16	224 × 224	16	15,262,696
vgg19	224 × 224	19	16,567,272
xception	299 × 299	71	62,981,867
alexnet	*227* × *227*	8	650,000

```
nvidia-smi -i 0 -ac ${memory_clock_rate},${core_clock_rate}
```

Fig. 4 Frequency capping with *nvidia-smi*

allowing for a comparison between execution time and frequency behavior. The measured values are stored in a *Comma-Separated Value* (CSV) file.

Technical Setup:

Our technical setup utilizes a SLURM-based HPC cluster, ensuring consistency using the same machine for all experiments. This machine has three NVIDIA V100S 32GB graphics cards, 256GB of memory, and an EPYC

ROME 7272 processor with 2 AMD. The home directory is connected through a 10 GBit/s ethernet connection to a *Network Attached Storage* (NAS) [13].

3.2 Evaluation Process

As already mentioned, we consider the following setup: Our independent variables contain the CNNs used for inferencing $N_i \in \mathcal{N} = \{N_1 = alexnet, N_2 = densenet121, ..., N_{31} = xception\}$ and the varied frequencies $f \in \mathcal{F} \subset \mathbb{R}_{\geq 0}$. Our dependent variables contain the maximum power consumption P and the computation time T. While $\mathcal{N}$ is a discrete set, $\mathcal{F}$ can be easily discretized by sampling the frequencies at interest and ensuring a sensible distribution.

For each CNN inferencing task, we sampled 196 different frequencies (ranging from 135 MHz to 1597 MHz) in approximately uniform distribution and performed $n = 3$ repetitions of these tasks to factor out measurement noise. Hence, for each CNN, we have $196 \cdot 31 = 6076$ data points (each in $n = 3$ repetitions) both in maximum power consumption and computation time.

We use standard statistical measures for evaluating our results: First, we analyze the reliability of the computed data by computing mean $\mu(d)$ and standard deviation $\sigma(d)$ for each data point d in maximum power consumption P and computation time t, as well as the variation coefficient $\frac{\sigma(d)}{\mu(d)}$ for both dependent variables. Second, we compute relative values (on means) $t_{rel}(d) = \frac{\mu_t(d)}{\min_{f \in \mathcal{F}} t(d)}$ for computation time based on the minimum value of computation time. This computation shows for each frequency in each net by what factor the computation time increases compared to the shortest computation time given this CNN. Third, we compute correlation coefficients $\sigma_{xy} = \frac{\mu(xy - \mu(x)\mu(y)}{\sigma(x) \cdot \sigma(y)}$ for each net between both dependent variables, but also power consumption to frequency as well as computation time to frequency, i.e. σ_{Pt}, σ_{Pf}, and σ_{tf} to analyze their influence on one another. This value is normalized to lie between -1.0 and 1.0, where higher absolute values denote a higher linear dependence and lower absolute values (usually below 0.5) indicate no linear correlation.

4 Evaluation

In this section, we describe the results of the above-described evaluation process. Furthermore, we analyze and discuss our results to answer the following evaluation questions:

- **Evaluation Question 1:** How reliable is the technical setup?
- **Evaluation Question 2:** What influence does the frequency have on the computation time of the CNN inferencing task?

- **Evaluation Question 3:** What influence does the frequency have on the power consumption of the CNN inferencing task?
- **Evaluation Question 4:** Which influence is higher on computation time or power consumption, the net's or frequency's influence?

Combining the results of these evaluation questions may allow a sound answer to the original question about the influence of frequency scaling on performance measures.

4.1 Evaluation Results

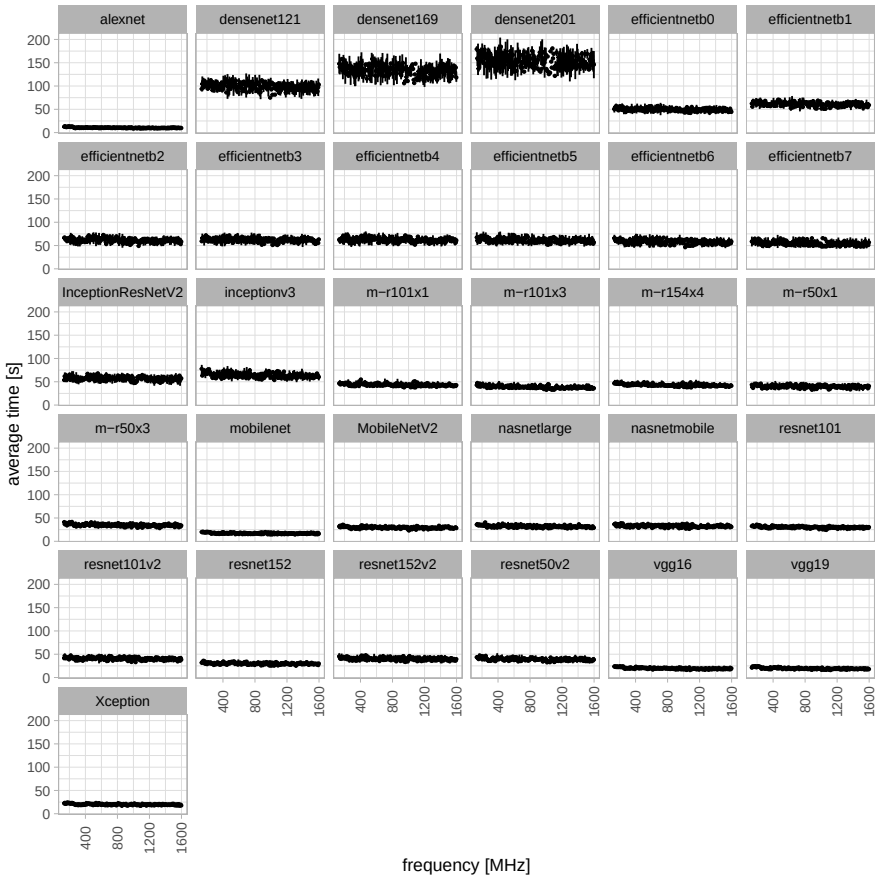

Fig. 5 Average computation time (in seconds) for all nets and frequencies with error bands of one σ

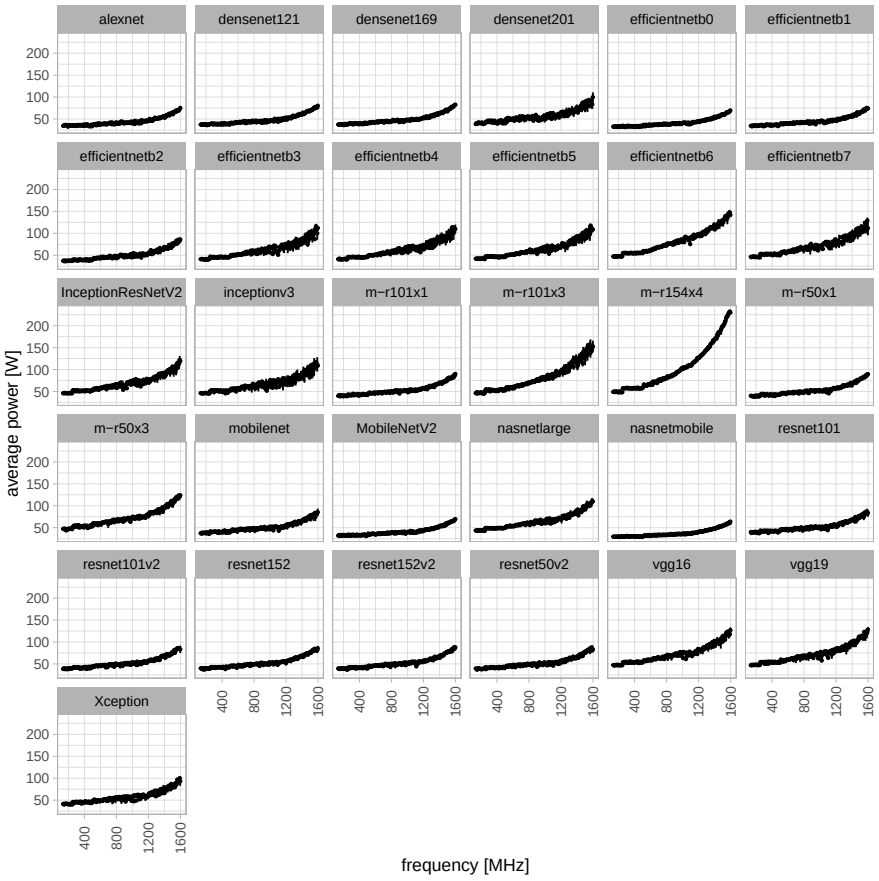

Fig. 6 Average maximum power (in W) for all nets and frequencies with error bands of one σ

Figures 5 and 6 show the results on average computation time and average maximum power consumption. In both figures, facets show the results per net, the x-axis shows different frequencies, and the y-axis shows the average value with an error-band of one standard deviation for time and power, respectively.

In Figure 5, one can see that in most nets, the error band is comparatively narrow, with the obvious exception of the densenet-variants. The inceptionnet-variants and efficientnet-variants also show higher standard deviations of consumption time than the remaining nets. It is noticable that the frequency seems to have, at most, a mild negative effect, for most nets, the trend appears constant at first sight.

In Figure 6, we again see higher standard deviations for most of the efficientnet-variants as well as the inceptionnet-variants, however, not for the densenet-variants. Here, only densenet201 shows high standard devia-

tions. Additionally, the setup results in higher standard deviations for the m-r101x3 net. As for the average maximum power values, the figure shows a progressing curve for all nets, however, with varying steepness, e.g. the net m-r154x4 has the highest increase for high frequencies, whereas nasnetmobile has a comparably low increase, although still an increase. It is, however, remarkable that the increase significantly progresses after passing the 1200 MHz frequencies. For most nets, this is where the most significant increase starts.

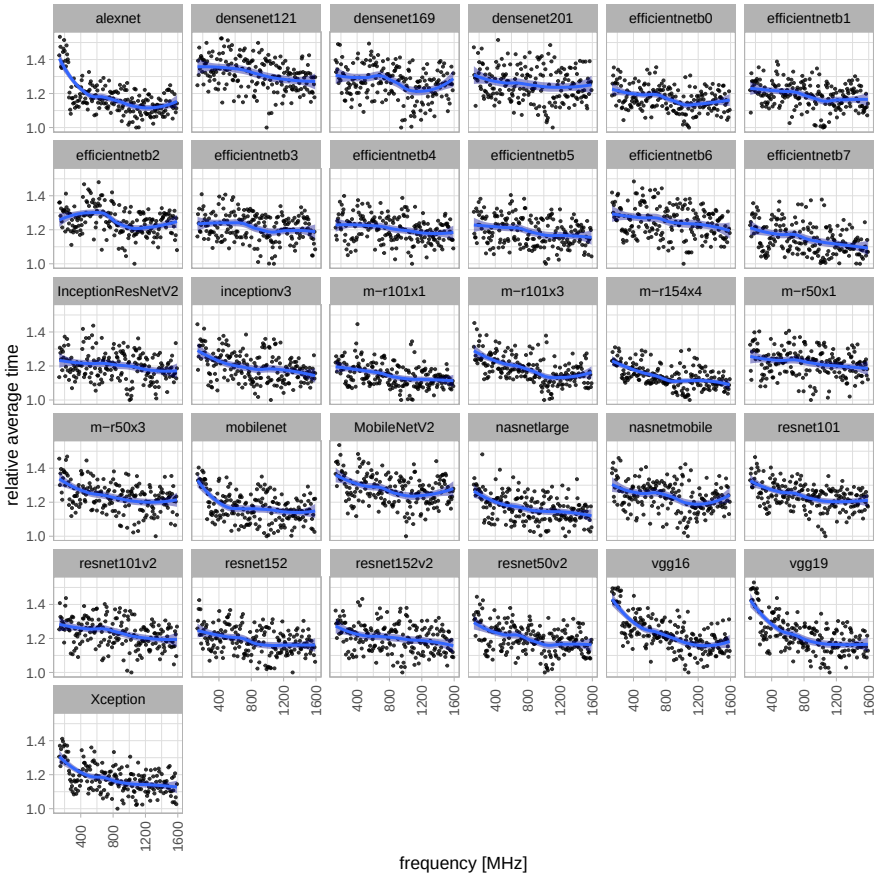

Fig. 7 relative average computation time for all nets and frequencies with a loess-model fitted

To further investigate the only barely visible negative trend for the average computation time, we computed the relative average computation as described in Section 3.2. Figure 7 shows the result. Again, each facet represents a CNN inferencing task (in the same order as both figures before), the

x-axis shows the frequencies, and the y-axis now shows the relative average computing time based on the minimum value (for this net). The blue lines indicate a model fitted via the loess method to aid the eye in visualizing the underlying trend. It cannot be seen as direct model computation. There are several things to observe here: First, the higher variability in the data is not an indicator of a high deviation in terms of the frequency variation as all of the data is now scaled to its perceptive minimum and does still not exceed 1.5, i.e. a 50% increase on the minimum. Second, in this visualization, the negative trend is easier to spot, however, it is still very small. Some nets, e.g., alexnet, vgg16, vgg19, show a rather clear negative trend, while others still show basically none at all, e.g., efficientnetb3, densenet201, or InceptionResNetV2. Third, the minimum value, i.e. 1.0, seems to generally lie between 900 and 1200 MHz, sometimes even once at the beginning of this range and the end, see e.g. InceptionResNetV2. Last but not least, some nets seem to show a change of trend in the middle of their data: It is especially prominent in densenet169 and nasnetmobile, where at around 900 MHz, there seems to be a drop in relative computing time, and then an increase can be seen.

To undermine our visual findings with a quantitative metric, we computed the correlation coefficient for each net between frequency, average computation time, and average maximum power. Figure 8 shows the results, light gray columns show the correlation coefficient results for frequency and average maximum power, and dark gray columns show the correlation coefficient results for frequency and average computing time. As was already visible in Figure 6, the values for the correlation coefficient with average maximum power are very high, sometimes even close to 1.0, i.e. there is a strong positive relationship between both variables. To phrase it differently, a higher average maximum of power can be seen for higher frequencies. The picture is inverse when looking at average computing time. Here, all correlation coefficients are negative, albeit very small. The highest values to be observed are the ones for alexnet, and both vgg16, and vgg19, which matches the conclusions from the last paragraph.

4.2 Evaluation Discussion

For Evaluation Question 1, it is important that the repetitions yield comparable results. A closer look at Figures 5 and 6 shows that although some nets have a higher standard deviation, these are also the ones with higher average values. A computation of the variance coefficient for both maximum power and computation time yields most variance coefficients to be smaller than 0.2, and none exceeding 0.3. These are small values, that indicate that is valid to trust the technical setup to give reproducible values.

In response to Evaluation Question 2, we can detect only a marginal negative impact on the computation time of the CNN inferencing task, i.e. higher

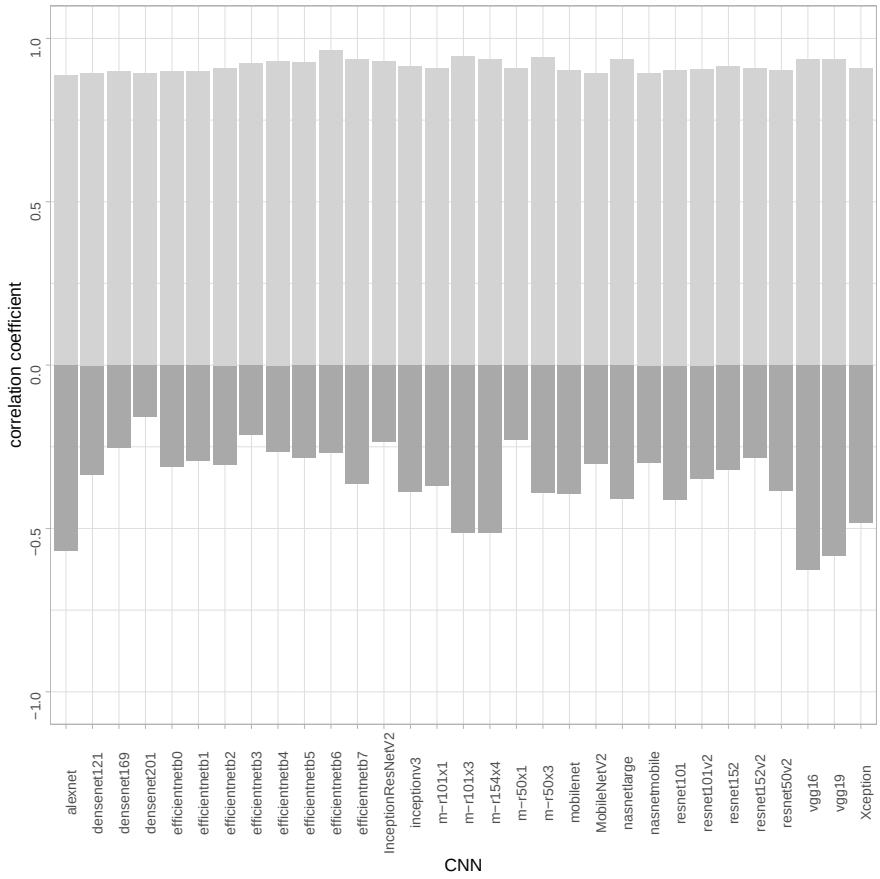

Fig. 8 Correlation coefficients for frequency and computation time as well as frequency and power

frequencies lead only to very small improvements in computation time if at all. This finding aligns with the general direction of the findings in [18].

Concerning Evaluation Question 3, we have observed a significant link between frequency and power usage. The power consumption drastically increases when the frequency exceeds 1200 MHz, the range where overclocking begins, as the base frequency of NVIDIA V100 is 1245 MHz. As a result, overclocking has a greater impact on power consumption, while underclocking has a smaller effect on most CNNs. However, some exceptions exist, such as the NASNetlarge or the NASNetMobile, which suggests that network design also plays a role in power consumption. Overclocking the GPGPU core frequency generally does not result in significantly better performance but increases power consumption dramatically.

The answer to Evaluation Question 4 has two parts: Firstly, the frequency has a significant impact on power consumption, as shown by the strong correlations. There are a few exceptions like the NASNetLarge and NASNetMobile, where the increase of power consumption already starts before the 1200 MHz frequency. Please notice that both CNNs are automatically designed by *Neural Architecture Search* (NAS) techniques and not by humans. Thus, the NAS technique and the implementation of both networks could lead to different behavior. Secondly, while the correlation between frequency and power consumption is generally high, the correlation between computation time and frequency is the opposite, indicating that the application influences computation time more than the frequency does. This effect can also be observed in the findings of [18] for other high-performance applications. In addition, the implementation of the application can also have a significant impact on the computation time.

Overall, lower frequencies lead to lower power consumption for CNNs, while the computation time is never more than 50 % slower as the fastest execution. Consequently, we recommend lowering the frequency of CNNs inferencing tasks to establish sustainable and energy-efficient systems.

5 Conclusion

We present a new experimental results on DVFS for CNN on the NVIDIA V100S showing an interesting behavior of the GPU. As our experiments pointed out, higher frequencies significantly impact power consumption, while small to middle frequencies have a smaller impact. Moreover, we showed that the impact of the frequency on computation time is marginal and can be ignored. Consequently, lower frequency settings can be used to design energy-efficient CNN inferencing systems without significant performance loss.

The new insights can help to design more energy-efficient AI systems, thus leading to more sustainable AI. In future work, we plan to develop automatic optimization methods to automatically adjust the frequency of GPGPUs related to the executed applications. We aim to attain superior frequency adjustment and energy conservation while maintaining optimal performance.

Acknowledgements This work was supported by the Data Science Center of the University of Bremen (DSC@UB).

References

1. Daily, M., Medasani, S., Behringer, R., Trivedi, M.: Self-driving cars. Computer **50**(12), 18–23 (2017)
2. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255 (2009)
3. Deng, J., Lin, Y.: The benefits and challenges of chatgpt: An overview. Frontiers in Computing and Intelligent Systems **2**(2), 81–83 (2022)
4. Ge, R., Vogt, R., Majumder, J., Alam, A., Burtscher, M., Zong, Z.: Effects of dynamic voltage and frequency scaling on a K20 GPU. In: 2013 42nd International Conference on Parallel Processing. pp. 826–833 (2013)
5. Ge, R., Vogt, R., Majumder, J., Alam, A., Burtscher, M., Zong, Z.: Effects of dynamic voltage and frequency scaling on a k20 gpu. In: 2013 42nd International Conference on Parallel Processing. pp. 826–833 (2013)
6. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), http://www.deeplearningbook.org
7. Hoinkiss, D.C., Huber, J., Plump, C., Lüth, C., Drechsler, R., Günther, M.: AI-driven and automated MRI sequence optimization in scanner-independent MRI sequences formulated by a domain-specific language. Frontiers in Neuroimaging **2**, 1090054 (2023)
8. Hong, S., Kim, H.: An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness. SIGARCH Comput. Archit. News **37**(3), 152–163 (2009)
9. Hong, S., Kim, H.: An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness. In: Proceedings of the 36th annual international symposium on Computer architecture. pp. 152–163 (2009)
10. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Pereira, F., Burges, C., Bottou, L., Weinberger, K. (eds.) Advances in Neural Information Processing Systems. vol. 25. Curran Associates, Inc. (2012)
11. Lustig, D., Sahasrabuddhe, S., Giroux, O.: A formal analysis of the nvidia ptx memory consistency model. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. p. 257–270. ASPLOS '19, Association for Computing Machinery, New York, NY, USA (2019)
12. Mei, X., Wang, Q., Chu, X.: A survey and measurement study of GPU DVFS on energy conservation. Digital Communications and Networks **3**(2), 89–100 (2017)
13. Metz, C.A., Goli, M., Drechsler, R.: Towards neural hardware search: Power estimation of CNNs for GPGPUs with dynamic frequency scaling. In: Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD. pp. 103–109 (September 2022)
14. Metz, C.A., Plump, C., Berger, B.J., Drechsler, R.: Hybrid PTX analysis for GPU accelerated CNN inferencing aiding computer architecture design. In: Forum on Specification & Design Languages (FDL). Turin, Italy (2023), (accepted for publication)
15. Milenkovic, M.: Internet of Things: Concepts and System Design. Springer Nature (2020)
16. Nvidia: NVIDIA H100 Tensor Core GPU. https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet, accessed: 2023-08-31
17. Nvidia: Volta architecture whitepaper. https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf, accessed: 2022-01-18

18. Patki, T., et al.: Comparing GPU power and frequency capping: A case study with the MuMMI workflow. In: 2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS). pp. 31–39 (2019)
19. Saiz, A., Prieto, P., Abad, P., Gregorio, J.A., Puente, V.: Top-down performance profiling on nvidia's gpus. In: 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 179–189. IEEE (2022)