On Improving the Use of Dedicated Hardware Resources for Deep Learning Methods on the Small Data

Iurii Krak [1,2], Vladislav Kuznetsov [1], Anatoliy Kulias [1] and Andriy Stavrovskyi [1]

[1] *Glushkov Cybernetics Institute, Kyiv, 40, Glushkov ave., 03187, Ukraine*
[2] *Taras Shevchenko National University of Kyiv, Kyiv, 64/13, Volodymyrska str., 01601, Ukraine*

Abstract

This paper addresses the problem of improving deep computing, especially equipment prices, electricity prices, etc., which in turn impact not only research cost, but also experimental reproducibility and consistency. Based on our findings, we propose an idea that should be useful for complex computations such as machine learning on small datasets using various neural network structures, including RCNN and DCGAN, which are notoriously timeconsuming and memory-intensive methods. when applying specific data. The improvements here rely on using our own data processing methods, comparing hardware using known techniques as benchmarking hardware for central processing units (CPUs) and graphics processing units (GPUs). We use this straightforward approach in our experiments, using wellestablished datasets available online to evaluate hypotheses about computing devices and algorithms. This in turn helps us infer device properties based on how the device behaves in the experiment, thereby improving the productivity and quality of our experiments. For our experiments, we used a wide range of libraries such as scikit-learn, TensorFlow, Pytorch, and Direct ML for practical desktop and laptop computing devices that also run CPUs and GPUs. From this study, we found major differences in the behavior of CPUs and CPUs using available practice exercises in each case - deep neural network training. Our results allow us to confidently choose an appropriate device based upon the deep learning task complexity.

Keywords 1

Network architecture, deep learning, training, small data, hardware

1. Introduction

During these years, various new methods and devices for deep learning have been developed. Especially large companies, as well as scientific research institutions, have benefited from it. Instead, individual researchers have discovered a major problem: what is useful in large tasks ("big data") is not always useful in small tasks ("small" data). Problems in performance evaluation for such small tasks can be considered in work of Kim J. et al [1] where they solved a task of feedback supervised learning on small datasets, which can raise attention to the problem of the small data in modern research in deep data field. Reasonably, to optimize research costs, they use all available devices: gaming graphics cards, data center cards, and laptops with integrated graphics; for instance in work published by Mahakalkar N. et al. [2], one can observe a possible application, where the customer grade GPU used for island-based algorithm optimization task.

Discussing the advances in these recent works we can state that there isn't always a solution to this problem, since each task needs own task-specific hardware: heavy-workload graphics cards may seem like overkill for simple tasks, while regular consumer cards may not be able to handle slightly larger

International Scientific Symposium "Intelligent Solutions" (IntSol-2023), September 27-28, 2023, Kyiv-Uzhhorod, Ukraine
EMAIL: iurii.krak@knu.ua (I. Krak); kuznetsow.wlad@gmail.com (V. Kuznetsov); anatoly016@gmail.com; (A.Kulias); andriystavrovskyi@knu.ua (A. Stavrovskyi)
ORCID: 0000-0002-8043-0785 (I. Krak); 0000-0002-1068-769X (V. Kuznetsov); 0000-0003-3715-1454 (A.Kulias); 0009-0000-9750-5571 (A. Stavrovskyi)

tasks due to performance degradation or memory bottlenecks, which may tend to “optimization” of architecture, rather the algorithms.

Why is it so important? In recent years, due to problems with the global supply chain, there has been a severe shortage of semiconductors, especially modern types of video memory and graphics cards, making graphics cards and video memory very expensive. So even if there is, it will cost a lot of money, except for big data centers and corporate cards (new cards that are newly purchased by individuals are not easy to buy). Nowadays, this problem is solved to some extent: there are 3 companies that produce graphics cards for computers: Nvidia, AMD, and Intel, and there are many proprietary and open source libraries for computing with these types of graphics cards (such as CUDA [3], Microsoft DirectML [4] etc.). In one of the works, Martinez P.-A. et. al [5] discussed usage of Intel One API for machine learning tasks, that can serve as an example of multiprocessing. Also, two of these suppliers produce multi-core processors for high-performance computing, which offers the possibility of using them in certain situations where the cards are not available or their use does not make sense. The existence of three suppliers is good, but the question is what should we do if a similar problem occurs again (global supply chain disruption or other reasons)? If this happens again, individuals should consider finding such a solution that either helps them take advantage of obsolete hardware rather than investing in it, although it is a viable solution for now. So, depending on the answer, a solution has to be chosen: either individual researchers have to apply solutions that don't work, or they have to reorganize or improve their computers for specific tasks (shrinking down machine learning models or using them for tasks like learning from small amounts of data).

We believe the last solution is the better one. Therefore, the purpose of this investigation is to find a given solution to a problem. To this end, it is recommended to formulate and address the following complex tasks:

- to suggest an idea to establish and compare the performance of individual devices;
- to find the benchmarking tasks in order to compare behavior of devices in deep learning tasks;
- to establish lowest and highest performance limits for the use of such devices in given cases;
- to suggest an approach to harness the GPU/CPU computing power in a more complex task;
- to consider a solution to selecting an appropriate device, according to sophistication of the task.

2. Related Works and Small Data Problem Solving

Here we discuss the problem approaches to test small data using a range of available hardware, used in data analysis and intelligent data processing.

2.1. The small data problem and the solution

If we compare the availability of HPC solutions from the beginning of applied computer science to today, we see that there were a few paradigm shifts in computing — centralized, decentralized, remote, and local, — which shown a significant change. Today, however, users have the opportunity to choose whether to use remote solutions such as cloud solutions or local services based on desktop computers or small computing servers. The advantage of the latest solution is obvious - complete control over the experiments, since all available resources are dedicated to specific computational tasks. However, despite the variety of possibilities, researchers sometimes have to guess the setup in order to adjust an experimental system for a specific purpose. For instance, in work by authors of the paper (Krak Yu.V. et al.) [6], one can observe, that even in tasks of planning of manipulation systems, there is also a margin of optimization, which, in turn, lies in algorithmic complexity, that supplements modern problems of task sophistication (implementation part) and all possible outcomes in optimization of either the software or hardware to fulfill such a task. In order to achieve this, there must be a way to estimate the capabilities of the current system. By means of well-known methods, it is possible to determine the basic connection between the resources used for small and large tasks. Moreover, when we discuss the most recent advances, we see this problem in detail; for instance in work by Zhao B. et al. [7] one can observe a simple example of benchmarking specific computational task, such as classification of hyperspectral datasets, which can give a hint on developing benchmarks for a specific machine learning task we observe in the current paper in following chapters.

Sometimes such tasks to be solved lay somewhere in-between of smaller tasks and really larger tasks that need to be done on dedicated servers or cloud solutions. We consider these tasks "small data solving tasks" because they are the basis for innovative solutions (big models) and there is room for unknown variables, sometimes one has to measure them using small tasks to obtain the required system parameters, their capabilities and identify the positive and negative parts of the solution used.

Since we're mainly talking about common desktop or laptop solutions here, we'll focus on the features of their architecture - the PC ecosystem and its CPU, GPU (if available), and memory.

In order to find out the unknown part in small data solving tasks, we think that a better approach is to use commonly used datasets related to image recognition, machine learning, and suitable hardware as well, since it has been proven to perform such tasks. For instance, in work by J. Lee at al [8], as well as work by Xu X. et al [9], one can find possible problems, related on optimization of neural network architecture as well as deep neural network inference (neural net activation on unknown data), which gives a roadmap to develop specific algorithms as well as finding a specific hardware that may meet our needs – best efficiency on small data tasks. With this approach, we can be reassured that the proposed hardware not only meets our needs in terms of efficiency, increased power consumption, and own cost, but is also suitable for specific small data solving tasks, as we consider further in the following paper chapters, using the given hardware and suggested methods. To this end, we also agreed on comparing the capabilities of the processor with the GPUs available hands-on in our lab, so that we can draw some important lessons from this comparison as well.

2.2. The general idea to estimate the efficiency of a computational device

At this point we would like to point out that while there are many solutions on the web, we cannot simply rely on them and their metrics: for example, there are many solutions that use benchmarks to estimate the computing power of a computer system, i.e. estimate the system against Performance levels for specific tasks (eg, computing lighting in 3D scenes, encoding video, computing complex mathematical expressions or formulas). As an example of this task one can observe the facial recognition problem that is quite intricate and memory consuming task, as well as time consuming; authors of the paper (Yu. Krak et al) in [10, 11] as well as Porta-Lorenzo M. et al in [12] discussed this task. While the facial recognition not only the task to provide rough performance estimates, there are some others as well; as an analogy of facial expressions one can use sign language representations in either 2dimensional or three-dimensional space. In the works made by authors of the paper (Yu. Krak et al) [13] and Wan J. et al. [14] we can observe applications of these task and establish baseline for performance. For obvious reasons, estimating the performance of 3D rendering does not mean that solving a system of linear equations works the same way as training a deep convolutional network. To do this, we need to develop benchmarks that make our assumptions more reliable.

We may also know about system utilization—underutilization and overutilization—because both affect benchmark results and our assumptions about the efficiency of a computing device (CPU or GPU). That is why the computer acceleration of these task is so important; in work by Afif M. et al [15] one can see the application of GPU acceleration for computer vision tasks, as well as application of NVidia CUDA technology for application of these tasks. It may give a key for possible applications in our paper.

While it may not be an obvious solution, we are free to assume that it may be rather effective for certain tasks that we require. To aim on this goal, we may focus on utilizing various machine learning models and data dimensionality reduction methods, such as singular value decomposition [16], T-stochastic neighborhood embedding [17] and autoencoders. Additionally, our goal is to expand our experiments in order to compare different devices behavior within neural network architectures to assess the most valuable information based on our system performance under various conditions. We believe that this approach will be beneficial for specific tasks that involve small datasets. According to our estimate, we will be able to determine whether we are utilizing the appropriate hardware for the task or suggest a specific one according to the needs. We are studying different techniques to optimize the hyperparameters of the methods we use and hence, to enhance our data. These methods are widely known of their ability of dimensionality reduction, data organization and grouping as well as being able to process small data quantities, what we decided to discuss more in detail further in paper.

3. Experiments with CPU devices and general approach for task solving

Here, in section we discuss the experiments made with the general purpose computing devices for desktop machines in order to implement various tasks for machine learning, such as deep neural network models, by utilizing hands-on equipment currently available in our inventory.

3.1. The general scheme of an experiment

For our experiments, a method was suggested that allows us to scrutinize different devices and roughly estimate their performance. Hereby, we benefit from an environment that depends on the contents of the local or remote desktop computer (according to the task) and allows us all necessary controls to measure resource usage via the standard task manager, device monitor, benchmarking software or device-specific software such as GPU Resource Monitor. This highly benefits since we can determine resource consumption before and under load. Therefore, we are less prone and affected by the background tasks in system, which are likely not related to the experiment.

In order to perform data analysis tasks, we prepare data in our environment; the key is to keep data in system memory (RAM disk, system RAM, video RAM, or RAM and VRAM together), so that we don't have to worry about caching data from disk, but is focused on RAM to VRAM transfers, that would be likely faster than using page file. The input data are pre-extracted features generated by cascade models including: singular value decomposition (SVD) [16], T-stochastic neighborhood embedding (TSNE) [17,18] and K-means clustering. This allows one to focus only on execution time (time elapsed) and factors that may affect deep neural network tasks such as: initialization of the network, training of the network, and testing of the network with test data. As an example of further usage of this data we may consider social network analysis task, discussed in the work by Madhuri K. et al [19]; while it is more connected to other topic it is very important for visual data analysis task.

To document the experiments, we tracked information from Resource Monitor, and execution in the Jupyter environment—the Jupyter server's console (which prints timestamps when jobs are stopped and when autosaves are created), as well as Jupyter Notebook output and Resource Monitor. This allows us to control the flow of the experiment and see if the Python kernel might be stuck or if there is processor or GPU throttling. Theoretically, based on the graph alone, we can determine the boundaries of each load cycle - either for the next epoch or load a new batch of data. This also helps to determine load (in %) and detect underload or overload of the test system we use here.

After the experiment, we look not only at the numerical data but also at the overall performance and note in our workbook if there are any issues with the equipment or the algorithm and why they might have arisen. We will then carefully analyze the behavior of this device several times for possible signs of such behavior. We are considering extending the task so that it can help spot issues.

Since our test system consists of different devices, we will try to benefit from their capabilities if they are available. We also analyzed the behavior of the system, like performance rate per iteration.

Since our scheme affects both processor and GPU devices, we will use it as a general template and customize it according to the characteristics of each device, so they won’t differ gradually. In general, this is quite a straightforward approach, which differs slightly for the processors or GPU devices.

3.2. The theoretical and practical performance of processors and GPUs

Let's discuss in more detail the experimental setup for testing different processors that we have actually tested in a lab environment. First, we would like to emphasize that we tested our device with different algorithms (especially deep neural networks) in experiments. The aim is not to favor one device over another, but to explore its capabilities in the context of different approaches.

In our text environment, we tested two different computers equipped with AMD Zen2 processors [20], including a desktop processor (AMD Ryzen 5 3600X) and a mobile system (AMD Ryzen 7 4800H). The common approach here is based on benchmarking of the test system, so as we base our own approach on one proposed by Kounev S. et al. [21], but modifying the tasks for our needs based on the benchmarking method. The specifications of the processors are shown in Table 1; note that we also use performance metrics using the well known PassMark benchmark and PassMarkScore.

Table 1
Performance of different processors used in experiments

Processor Model	Cores	Threads	Base Clock (GHz)	PassMark Score
AMD Ryzen 5 3600X	6	12	3.8	18259
AMD Ryzen 7 4800H	8	16	2.9	18851

The reasons we make use of this benchmark are simple: it contains many different tests that take into account the different capabilities of the tested system and average across tests. This gives us confidence in the results to compare the benchmark performance with the results of our deep learning-related experiments (e.g. training, testing, and init). To conduct our tests, we use proven and known software and libraries that are widely used in the scientific, computer science and data analysis communities. We benefit from a Jupyter test environment based on the Python distribution Anaconda 3.7 for all of our environments (Windows and Ubuntu). To perform deep learning tasks, we use TensorFlow. This library is widely known and used in many applications, that can be studied from the survey, proposed by Ramchandani M. et al in the work [22]; it gives a view of possible applications of the technology. We also tried using CUDA acceleration in some test systems when enabled (desktop computers have GPUs, but disabled). Because we aimed to compare the effectiveness using widely known and accepted benchmarks, we ran our experiments to compare our lab equipment. We also made an assumption this will give us a better understanding of the results (Table 2).

Table 2
Performance of different procedures on processors and GPUs

Method	Nvidia RTX 2060[1,3]	Nvidia RTX 2060 H[2,3]	Ryzen5 3600X[1]	Ryzen5 3600X[1]
DNN init	0,1945	0,1856	0,251	0,234
DNN train	64,782	67,476	7,27	7,12
DNN test	0,7348	2,674	1,0946	1,0946

The superscript numbers in Table 2 have the following meanings: 1 represents the Windows test environment, 2 represents the Ubuntu test environment, and 3 represents the environment that supports CUDA accordingly. Two numbers, e.g. 1, 3, show CUDA on Windows environment.

We want to underscore that the equipment listed here is not only equipment we tested our methods against; we decided to sort out some obsolete and underperforming equipment like AMD Athlon X2 or Intel Core2 E-series processors because they had no some optimizations (instruction set) to perform deep learning tasks more efficiently (as well as data dimensionality reduction). In further research we will include them to showcase some differences in newer and older architectures for such tasks.

In our experiments, we found that hardware may behave differently than we would normally expect when performing such tasks. First, we found that there is a benefit to using some OS, but it can be considered negligible for deep neural networks - about 1-2% goes to testing and training, which is the most time-consuming task, so using of certain OS is mainly personal preference.

However, when we had tested the both systems, we found some issues with the system performance when using a laptop graphics card: since GPUs are generally considered as ones that superior to CPUs, it also founds out in experiment that compared to desktop processors, the GPU has an obvious disadvantage, and it obviously depends on situation. We need some clarification, as well. To this end we decided to conduct a series of additional experiments. Based on the general performance of such devices in deep learning tasks, one could explain the behavior of the GPU, or why the graphics card may did not work as expected.

4. The problem with graphics card performance and resolution

The section discuss the problems: finding the bottlenecks and situations as well when one or another device are not an appropriate to perform a specific task. To justify this we discuss the area of maximal efficiency within memory and performance range to get the needed level of efficiency.

4.1. Solving the Problem: Libraries, Devices, and GPU Experiments

In the previous experiments, we found out that we had issues when using a laptop GPU RTX 2060 Mobile. Since we were unable to determine the cause of the problem, we made a decision to conduct an experiment. It would allow us, in theory, to exclude conditions that might have or not affected our experiment. To achieve the goal, we decided to use other neural network libraries, to compare them to neural network library (TensorFlow with CUDA) and to use a desktop system based on previously tested AMD Ryzen 3600X and AMD Radeon RX6500XT desktop GPUs as well to take full advantage of our Work. We also chose to rely on publicly available datasets to avoid being overshadowed by some of the results we received. However, for experimental purposes, we may try to use this data for other tasks.

Because of this here we use TensorFlow and PyTorch, with adapter or backend Direct ML. It allows to apply it on the hands on available equipment (AMD Radeon RX6500XT GPU unit). While this GPU may or may not be a fair comparison to the Nvidia RTX 2060, our primary goals are finding potential caveats or, simply, performance hits. They likely ones, that one GPU might experience as other, in the same situation. So here we make use of the data to test the capabilities: of GPUs alone, not GPU-to-GPU comparisons, face-to-face.

To evaluate capabilities of our test system, we will test fixed number of neural network architectures, such as shallow networks, deep autoencoders, convolutional and recurrent convolutional networks, and general adversarial networks as well. This will allow us to sequentially test hardware with different architectures, estimate the efficiency level of these devices, and find the effective usage range of the CPU or GPU we are currently using in the following experiments.

For instance a performance curve can be compared, using open source data on CUDA accelerated or OpenCL accelerated machine learning tasks. Using a baseline performance, we can estimate or infer desired performance in machine learning oriented tasks. This can be observed on figure 1, which compares competitive designs that can be used in our test suite against if we consider improving our performance (the values on the figure are scaled to Intel i3-8100 which has performance 1000 points, which means, for instance, that GTX1080 is 54.5 times more powerful than i3-8100).

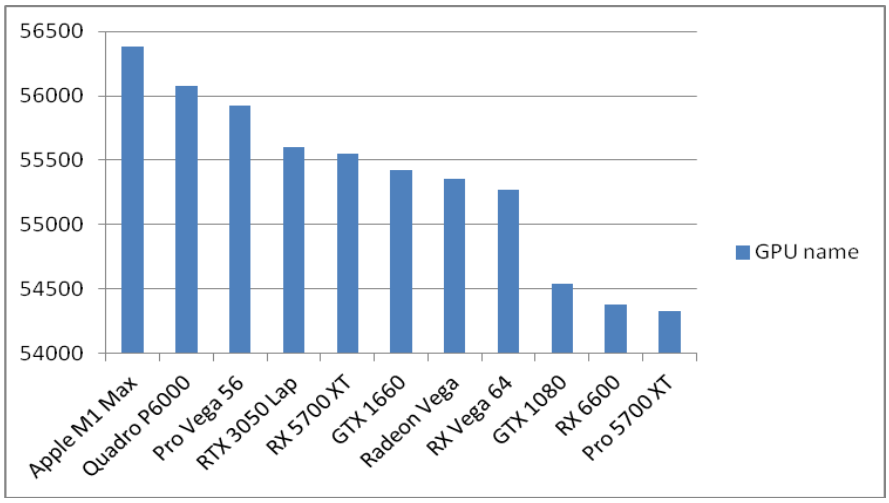

Figure 1: Comparison of possible near peer devices, comparable to our GPU using Passmark suite

As we were able to use these plots, we can then make the following – to infer the baseline performance of our CPU, compare it against our GPU and compare the relative performance of our CPU against i3-8100. This allowed us to test such features, such as plotting the performance curves of a graphics card compared to the CPU we're currently using. However, as we discovered earlier, the GPU is highly dependent on the computation task and its sophistication is not linear, if compared to the task. In order to overcome this we suggested a benchmark suite based on autoencoder optimization procedure, which is based on some suggestions proposed by Osipyan H. et al in [23]. In the next sections we will discuss some experiments we came to, using some publicly available datasets, as well as some optimization procedures to help us to make the equipment to behave better.

4.2. Experiments with deep networks and tiny datasets

We tested various models, including regular networks, convolutional networks, and autoencoders, on tiny datasets (the popular M.N.I.S.T. fashion [24] and M.N.I.S.T. Digits [25]), but found graphics cards to be useless in this case. Due to the low performance of graphics cards, we decided to investigate autoencoders that are more flexible than other models. Overall, our results suggest that using graphics cards may not be as effective at analyzing these datasets as we previously thought, and that other methods, such as exploring more complex models, which may us yield the extra performance as we want here.

4.3. Experiments with autoencoders and the different lambda functions

The first idea is to make an autoencoder structure more flexible and, as well as the possibility of forward and backward transformations (encoding and decoding). Since we opt to change transformations and create optimal transformations (likely many), we can also define a lambda function. For example, this function, describes properties of weights, an objective function etc. By modifying this function, the overall efficiency can be improved and several implementations of the same model can be referred to. Which, in turn, can consume fewer graphics card resources.

We decided to use lambda expressions, which are passed by to the optimization procedure as restrictions on weights. They compute the orthogonality of weights, norm of matrices, and independence of weights; this allows simulating SVD behavior using restriction on weights.

Using all these functions, we can create a linear autoencoder that behaves like a singular value decomposition (table 3). This approach can be observed in the work by Zhang C. et al [26] dedicated to some implementation areas for autoencoders, including encoding some complex functions, that encode input feature space representaton. However, if we don’t decide to use one, we may just create a non-linear autoencoder which creates a transformation in dimensionality reduction.

Here we have interesting results: there is a performance difference using a non-linear (normal) autoencoder, or the processor (here AMD Ryzen 5 3600X) is up to three times more efficient than the graphics card used here (AMD Radeon R.X. 6500 X.T.). Conversely, the difference reversed when more lambda functions were used - more than 5 times if compared to just one single processor. In addition, we can see that the calculation time of the previous graphics card is almost the same (the estimated performance level, according to open data). This indicates a performance bottleneck. When the card overcomes this bottleneck, performance increases. It's also the best explanation for our RTX 2060 mobile card: why it didn't deliver any advantages in our last experiment. However it may be argued, if we were able to change the conditions of the experiment, and adapt it for other type of data or other experimental setup. Additional experiments are needed to test our hypothesis. Now we know that simple jobs don't require a graphics card. However, when we are doing more time-consuming tasks, the GPU will give the best results. Indeed, we consider these results could improve with some tweaks. But in order to achieve them, one have to do what they say: when the equipment behaves accordingly, and achieves the desirable performance, it is really possible to find a performance band (interval), outside of which the device loses its level of efficiency.

4.4. Experiment with convolution networks for human face recognition

The experiments we do here are specifically applicable for face recognition. Here we use data available online - FER 2013 dataset [27] and a convolutional network performing facial expression classification. Since it's not about precision but about performance, this work is a review of equipment. To verify our experiments, we use the AMD Radeon Resource Monitor [28] (Fig. 2) and the Anaconda Jupyter console (Fig. 3), writing the necessary information in a text format.

Notably, we can witness that the test system utilizes the GPU resources via Python and DirectML backend, which is necessary to verify that the video card is used to test, instead of a CPU. The test results can be seen on Fig. 2. Of course, the benchmarking was done by injecting time steps of flags to measure from, as well as internal means, used in the TensorFlow-DirectML library to visualize the

utilization of the GPU time per each step, epoch or iteration of the training algorithm (Figure 4). It gave us a clue, how it behaves on different iterations of learning.

Table 3
Performance of learning procedure on an autoencoder with different parameters passed as lambda expression on each step

Name	CPU time, sec	GPU time, sec
Non-linear encoder and minimum number of iterations	0,46	0,61
Non-linear and normal encoder number of iterations	2,19	6,97
Quasi-linear encoder, weight orthogonality	27,55	7,36
Quasi-linear encoder, orthogonality of weights, norm of matrices	32,98	7,68
Linear encoder (orthogonality, norm, weight independence)	41,11	7,85

Figure 2: AMD Radeon Adrenaline Edition performance monitor during the experiments

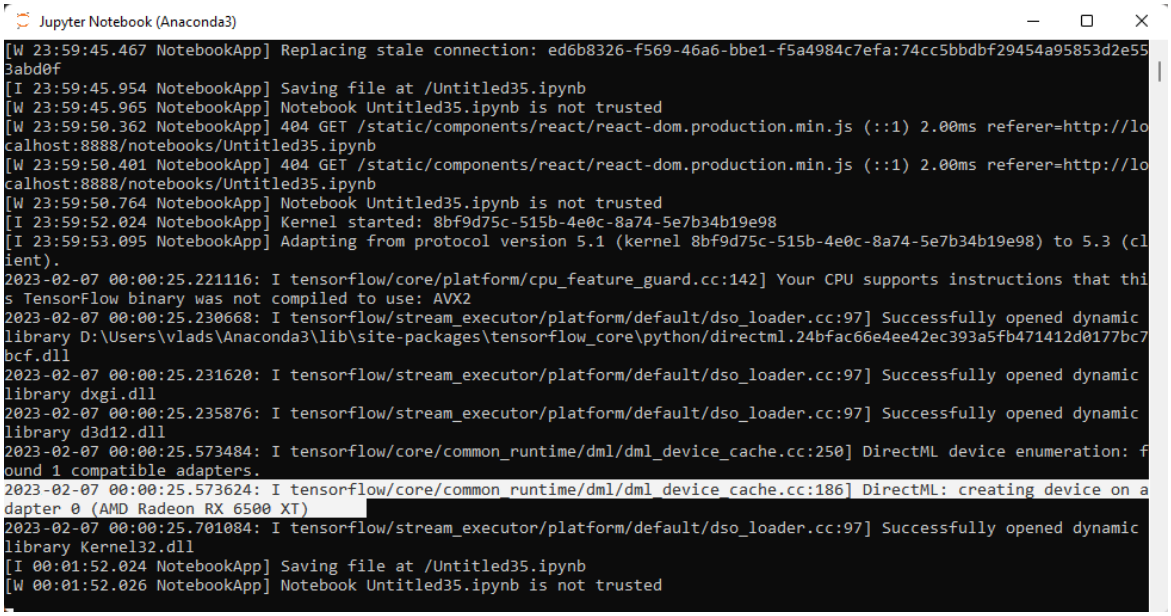

Figure 3: Python interpreter console and Jupyter environment (Anaconda)

As one can observe, the training algorithm may, on the first glance give not the best results. However, it was primarily designed to accommodate the need of novel algorithms and was proposed as competition, as can be found in [26, 27]. This competition was aimed to find a best algorithm for facial emotion recognition based on information crawled throughout Internet as open-source online image database. In the first iterations the results were around 65-70% (out of sample) and as of today in 2023 are known around 77-78% on the same accuracy metric. Since our intentions were to benchmark performance, the accuracy and loss plot are important features, hence they show that each iteration and epoch are quite unique (in terms of data and accuracy and loss values). After that we focused mainly on performance in time. Which, in turn, gave us some important results (Fig. 4).

From here, we found that time is of great importance and that each iteration proves the overall efficiency per step on the long run. Also, according to AMD Monitor used to show the performance, approximately 1.2 GB of VRAM resources were used, which corresponds to the number of hyperparameters of our network and data. Moreover, if we compare the results from epoch 1 and epoch 2, we clearly see that it takes around 80 seconds to send data from RAM memory to video memory. The time required for computation is comparable for small datasets, but performance may

not be as noticeable as tasks become more complex. The performance difference between the graphics card and the processor is about 4.5 to 1 in one iteration, and can be as high as 20 to 1 as the number of iterations increases, indicating that the graphics card also needs a lot of computing time to process and load the data. However, as number of iterations increases, the time also increases, as we expected.

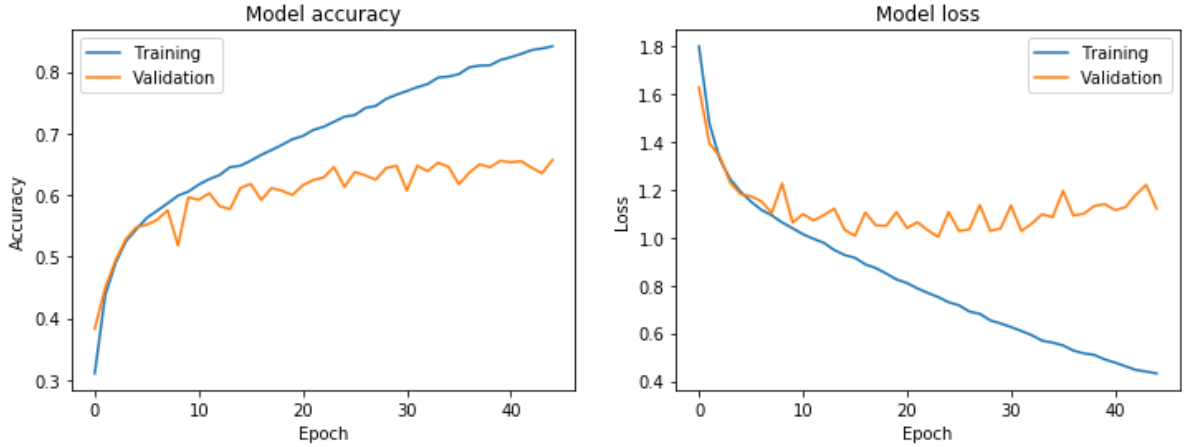

Figure 4: Performance during the experiments. CPU (on the left) and GPU (on the right).

```
Epoch 1/15

448/448[==============================]

449s 1ms/step – loss 1.7944 – acc 0.3116

Epoch 2/15

448/448[==============================]

447s 998ms/step – loss 1.4812 – acc 0.4364
```

```
Epoch 1/15

448/448[==============================]

107s 238ms/step – loss 1.7978 – acc 0.3112

Epoch 2/15

448/448[==============================]

22s 49ms/step – loss 1.4888 – acc 0.4260
```

Figure 5: Performance during the experiments. CPU (on the left) and GPU (on the right).

Based on the library calls in the Jupyter console we inspected (especially during initialization, as shown in Fig. 2), it is possible to assume the Direct ML interface to utilize "tensors", but in the AMD architecture. In order to transfer data between AMD GPUs, the adapter has to use DirectX 12 libraries to convert tensors using Direct ML and DirectX 12 libraries, which of them aren't directly compatible between AMD and Nvidia because they have different memory structures. Since we were working on this problem, we are now better prepared for further experiments, that utilize AMD video cards.

4.5. Additional experiments: if there is not enough memory

Several studies have reported that Recurrent Convolutional Neural Networks (RCNNs) and Deep Convolutional Generative Adversarial Networks (DCGANs) require large amounts of memory to train due to their complex architectures and large number of parameters. We therefore conduct an experiment on these memory-intensive networks, specifically RCNN convolutional networks and generative adversarial networks. To achieve consistent results, we use systems with the same processor and graphics card, but a different network implementation - PyTorch library with Direct ML. We use different techniques to process the two datasets. For image segmentation on the Cityscapes dataset, we use a hidden network RCNN, which can efficiently segment images of pedestrians. For image generation, on the other hand, we used the DCGAN model.

In fact, there in two experiments we conducted, it finds out that there is no sufficient memory. And in some cases it may seem concerning, if the model has to fit the available resources. However, not in

this case, since we are able to mediate the dedicated resources and tweak them in accordance with the task; however, there may be a need for some minimal usage, which may, in some cases not met the requirements to the system, as of one we have currently in our disposal hands-on.

To find a proper solution, we recommend using both the graphics card and the processor. Here's how it works: the model is loaded into the video memory and a system memory, then we do a computation step and compute the gradients using the GPU. While we prefer to compute the entire network weights at once, here we faced some memory constraints, with memory usage reaching about 3.8 GB video memory and 12.8 GB system memory (shared). As one can see, it is not trivial to handle such a large block of memory while using all the VRAM. (Figure 6).

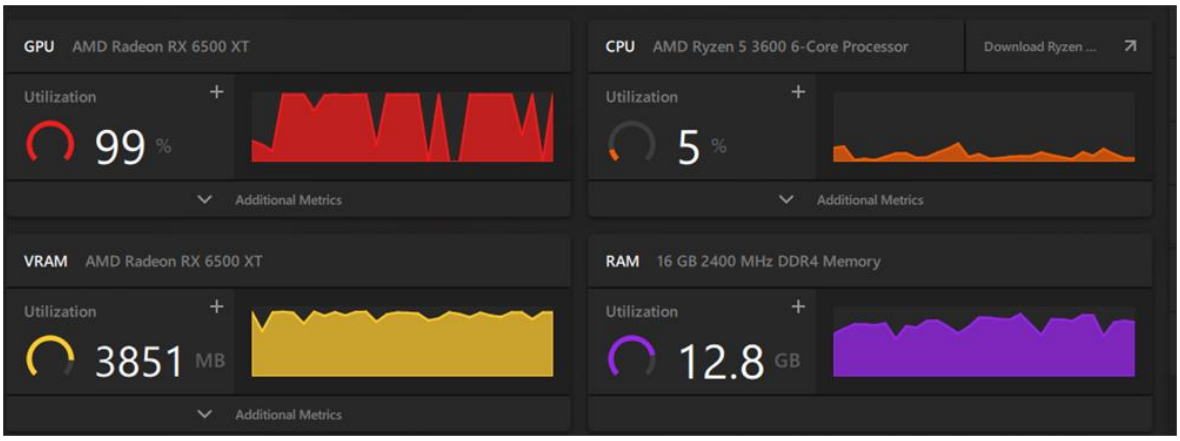

Figure 6: Memory usage, GPU and CPU time

Another effect we could probably see: the graphics card got a speedup with the processor combined by 2.74x if compared to the processor alone. That means the card can come in handy when work needs to be done, even if it may look inefficient at first glance. However, video memory bottlenecks are very problematic because large chunks of memory have to be sent after each iteration or epoch. As can be seen, fine-tuning of the network or data is also required. We also want to underscore that while the DirectML technology is quite new, it is still in development and, hence, it may be possible that some of implementations for TensorFlow or PyTorch with DirectML may work not as good as at native corresponding libraries with CUDA enabled. Because of that an additional study may be needed.

5. Discussion

The aim of this study was to improve the dedicated hardware resource usage in small data tasks, which in our case included fine tuning the algorithms, hardware as well as using different methods. The results showed us that while we have got some results performance wise in terms of either accuracy or time per iteration, based on specific type of the networks or other deep learning algorithm, the theoretical performance curve (which is, in our case time per iteration or time per epoch) show us that there is no simple correlation between baseline performance of a device in some synthetic tests like rendering or OpenCL. In our case we had to develop own tests, that raised a question: what the test is the better to track down the performance of the device? In order to do this, we decided to focus on some not that obvious construction as using synthetic network to imitate behavior of other algorithms. The key feature that the learned target function shoul not be imitated 1 by 1, but mimic its nature. That's why we used linear autoencoder approximation for singular value decomposition. While we may achieve it using some tweaks on neural network structure, instead we used few levels of approximation by lambda functions passed into the training procedure, that defined different levels of complicity of the task. Using them we discovered that performance is quite dependent not only dataset size or batches it processed, but rather in procedure. Hence, it raised the question – how complicated has to be the training procedure (in terms of inner functions) to achieve perfect performance ratio, that shows the difference between different devices?

Lastly, we must focus also on the devices capabilities. As we believe, the AMD GPUs, while being capable in calculations for science, in particular deep learning, they use not fully their

capabilities since non-native code and usage of either RocM or DirectML. This raises again another question – how much we lose if using non native library like TensorFlow oe Pytorch with Direct ML? We believe this can be answered if running the Nvidia GPUs on native CUDA libraries and then – on modified, within DirectML environment, as used here in the experimental study.

6. Conclusion

Overall, the main findings of this paper can be summarized in the following.

We found that one should not rely upon any particular architecture (CPU or GPU, neither vendor). But, as it seems more logical, also study its strengths and weaknesses. One should make informed decisions based on research. As asking -which one is better for the task? For example, we found that the hardware required for complex computer vision tasks (such as RCNN) is quite different from the hardware required to train the network on the MNIST dataset, which often called toy dataset.

The process of fine-tuning isn't easy, but it is worth gaining new insights into the capabilities of specific hardware for specific tasks (such as image recognition and text processing). However, we have also noticed some shortcomings of modern neural networks related to computer vision. And they sometimes have some disadvantages compared with classical computer vision methods.

Therefore, to unleash the full power of these new methods, a more advanced setup may be required, otherwise, the data may be scaled down, with its own known negative effects on accuracy.

It is also worth mentioning that there is considerable competition between different vendors; as some devices may have comparable computing power to similar devices from other vendors.

We also found that devices that perform similarly and produce the same results under ideal conditions (such as computer benchmarks) can sometimes perform quite differently on other tasks. For example, the variance between desktop and laptop processors of the same architecture (Zen2) is much greater than we expected; however, we believe that desktop processors of similar grade and performance may show more consistent and reliable predicted results. To truly achieve research goals, you would ideally need several different devices dedicated to one task, such as a processor for one task and a graphics card for another, or even multiples sharing them. This approach guarantees the most positive qualities that we will use with different devices, and decline the negative ones, balancing the efficiency and performance. We see that the researcher should be open-minded to new ideas and try different methods and laboratory equipment to achieve the research goals. This suggestion may be helpful in our future experiments, dedicated to computer vision, if we decide to expand the results and use other devices for more complex tasks and to improve, as a result, our current results.

7. References

[1] Kim J., Lee W., Kim Y. Feedback-supervised learning with a small dataset. KIISE transactions on computing practices. 2019. Vol. 25, no. 2. P. 130–135. https://doi.org/10.5626/ktcp.2019.25.2.130

[2] Mahakalkar N., Mahajan A. R. Multi-GPU island-based genetic algorithm. International journal of recent advances in engineering & technology. 2020. Vol. 08, no. 02. P. 32–38. URL: https://doi.org/10.46564/ijraet.2020.v08i02.06

[3] CUDA zon-library of resources. NVIDIA Developer. https://developer.nvidia.com/cuda-zone

[4] Online resource on Microsort machine learning using DirectX technology on GitHub. https://github.com/microsoft/DirectML

[5] Martínez, P.-A., Peccerillo, B., Bartolini, S., Garcia, J.-M., Bernabe, G. Applying Intel's oneAPI to a machine learning case study. Concurrency and computation: practice and experience. 2022. Vol. 34. Issue 13. E6917. P.1-15. https://doi.org/10.1002/cpe.6917

[6] Krak, Y. V. Dynamics of manipulation robots: Numerical-analytical method of formation and investigation of computational complexity. Journal of Automation and Information Sciences, 1999, 31(1-3), 121-128. doi:10.1615/JAutomatInfScien.v31.i1-3.

[7] Zhao, B., Ragnarsson, H.I., Ulfarsson, M.O., Cavallaro, G., Benediktsson, J.A. Predicting classification performance for benchmark hyperspectral datasets. IEEE journal of selected topics in applied earth observations and remote sensing. 2022. Vol. 15. P. 4180-4193. https://doi.org/10.1109/jstars.2022.3173893

[8] Lee, J., Rhim, J., Kang, D., Ha, S. SNAS: fast hardware-aware neural architecture search methodology. IEEE transactions on computer-aided design of integrated circuits and systems. 2022. Vol. 41. No. 11. P. 4826-4836. doi: 10.1109/TCAD.2021.3134843
[9] Xu, X., Ding, Y., Hu, S.X., Niemier, M., Cong, J., Hu, Y., Shi, Y. Scaling for edge inference of deep neural networks. Nature electronics. 2018. Vol. 1, no. 4. P. 216–222. https://doi.org/10.1038/s41928-018-0059-3
[10] Kryvonos, I. G., Krak, I. V., Barmak, O. V., Ternov, A. S., Kuznetsov, V. O. Information technology for the analysis of mimic expressions of human emotional states. Cybernetics and Systems Analysis, 2015, 51(1), 25-33. doi:10.1007/s10559-015-9693-1
[11] Krak, Y. V., Barmak, A. V., Baraban, E. M. Usage of NURBS-approximation for construction of spatial model of human face. Journal of Automation and Information Sciences, 2011, 43(2), 71-81. doi:10.1615/JAutomatInfScien.v43.i2.70
[12] Porta-Lorenzo, M, Vázquez-Enríquez, M, Pérez-Pérez, A, Alba-Castro, JL, Docío Fernández, L. Facial Motion Analysis beyond Emotional Expressions. Sensors. 2022; Vol. 22. No 10 P.3839. https://doi.org/10.3390/s22103839
[13] Kryvonos, I. G., Krak, I. V., Barmak, O. V., Bagriy, R. O. New tools of alternative communication for persons with verbal communication disorders. Cybernetics and Systems Analysis, 2016, 52(5), 665-673. doi:10.1007/s10559-016-9869-3
[14] Wan, J., Ruan, Q., Li, W., Deng, S. One-Shot Learning Gesture Recognition from RGB-D Data Using Bag of Features. In: Escalera, S., Guyon, I., Athitsos, V. (eds). Gesture Recognition. The Springer Series on Challenges in Machine Learning. Springer, Cham. 2017. P.329-364. https://doi.org/10.1007/978-3-319-57021-1_11
[15] Afif, M., Said, Y., Atri, M. Computer vision algorithms acceleration using graphic processors NVIDIA CUDA. Cluster computing. 2020. Vol. 23, no. 4. P. 3335–3347. https://doi.org/10.1007/s10586-020-03090-6 [16] Zuniga, C. D. Singular value decomposition for imaging applications. EBOOKS. 2021. 41 p. https://doi.org/10.1117/3.2611523.ch1
[16] Balamurali, M. T-Distributed stochastic neighbor embedding. In: Daya Sagar, B., Cheng, Q., McKinley, J., Agterberg, F. (eds) Encyclopedia of Mathematical Geosciences. Encyclopedia of Earth Sciences Series. Springer, Cham, 2022. P. 1-9. https://doi.org/10.1007/978-3-030-26050-7_446-1
[17] Pezzotti, N., Höllt, T., Lelieveldt, B., Eisemann, E., Vilanova. A. Hierarchical stochastic neighbor embedding. Computer graphics forum. 2016. Vol. 35, no. 3. P. 21–30. https://doi.org/10.1111/cgf.12878
[18] Madhuri, K., Rao, M. K. S. Social media analysis using optimized k-means clustering. International journal of trend in scientific research and development. 2019. Vol. 3, Issueb2. P. 953–957. https://doi.org/10.31142/ijtsrd21558
[19] Suggs, D., Bouvier, D., Clark, M., Lepak K., Subramony, M. AMD “Zen 2”. 2019 IEEE hot chips 31 symposium (HCS), Cupertino, CA, USA, 18–20 August 2019. 2019. PP. 1-24. https://doi.org/10.1109/hotchips.2019.8875673
[20] Kounev, S., Lange, K.-D., Kistowski, J. V. The SPEC CPU benchmark suite. Systems benchmarking. Cham, 2020. P. 231–250. https://doi.org/10.1007/978-3-030-41705-5_10
[21] M. Ramchandani et al. Survey: tensorflow in machine learning.Journal of physics: conference series. 2022. Vol. 2273, no. 1. P. 012008. https://doi.org/10.1088/1742-6596/2273/1/012008
[22] Osipyan, H., Edwards B. I., Cheok A. Deep neural network applications. CRC Press, 2022. 158 p. https://doi.org/10.1201/9780429265686-2
[23] GitHub - zalandoresearch/fashion-mnist: a MNIST-like fashion product database. benchmark. GitHub. https://github.com/zalandoresearch/fashion-mnist
[24] Dataset for images and characters for image recognition. http://yann.lecun.com/exdb/mnist/
[25] Zhang, C., Geng, Y., Han, Z., Liu, Y., Fu H., Hu, Q. Autoencoder in Autoencoder Networks. in IEEE Transactions on Neural Networks and Learning Systems, 2022, P. 1–13. doi: 10.1109/TNNLS.2022.3189239 [27] FER-2013 facial expression dataset for Kaggle. https://www.kaggle.com/c/challenges-in representation-learning-facial-expression-recognition-challenge/data
[26] FER-2013 facial expression dataset for Kaggle. https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data
[27] Advanced Micro Devices software for computer testing. https://www.amd.com/en/technologies/software