CEUR-WS.org/Vol-3513/paper10.pdf

Hardware Usage Improvement for Small Data Problem Solving by Deep Learning Methods

Iurii Krak[a,b], Olexander Barmak[c], Vladislav Kuznetsov[a], Serhii Kondratiuk[a,b], Oleg Stelia[b], Veda Kasianiuk[b]

[a] *Glushkov Cybernetics Institute, Kyiv, 40, Glushkov ave., 03187, Ukraine*
[b] *Taras Shevchenko National University of Kyiv, Kyiv, 64/13, Volodymyrska str., 01601, Ukraine*
[c] *Khmelnytskyi National University, 11, Institutes str., Khmelnytskyi, 29016, Ukraine*

Abstract
This article dedicated to the question of improvement of deep calculations, particularly, the price of appliances, the price of electricity and others, which may, in turn, affect not only the price of research but also repeatability and consistency of the experiments. According to the research, we propose an idea, which should be useful for sophisticated calculations, such as small data machine learning using different constructs of neural networks, including RCNNs, DCGANs, which are known to be very time and memory consuming methods if applied on a particular data. Here, the improvement underlies the usage of our own approaches to process data, to compare hardware using well known techniques to benchmark hardware as central processing units (CPUs) as well as graphic processing units (GPUs). We use this straightforward approach in our experiments in order to assess the hypotheses about computational devices as well as algorithms, using proven and online available datasets. This in turn, helped us to infer the device properties based upon their behavior in experiments and, hence, enhance 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 the hands-on available desktop and laptop computing devices running CPUs and GPUs as well. According to the study, we figured the main differences in behavior of CPUs and CPUs based upon available hands-on in the particular situation – training of deep neural networks. Our findings gave us the possibility, with some confidence, to choose appropriate devices for the deep learning tasks based on their sophistication.

Keywords 1
Deep learning, training, hardware, network architecture, Pytorch, Tensorflow, DirectML.

1. Introduction

In these years, a various number of approaches to process a data were developed, including, for instance new methods of deep learning, as well as computational devices. This development, in particular, was good for large enterprises, as well as big international scientific institutions. However, it didn't fit well for all of the tasks. As a result of this, some individual researchers have found that a large scale ("big") problem that needed a lot of resources may need some specific set of software, opposing to ones that were small, more intricate and hence very applied tasks of a small scale ("small") problem. Hence, one can formulate this as follows: the set of methods, used for large jobs (large datasets, data with big variety, volume, velocity, or known as "big data") need quite a different set of

ICST-2023: Information Control Systems & Technologies, September 21-23, 2023, Odesa, Ukraine.
EMAIL: yuri.krak@gmail.com (I. Krak); alexander.barmak@gmail.com (O. Barmak); kuznetsow.wlad@gmail.com (V. Kuznetsov); sergey.kondrat1990@gmail.com (S. Kondratiuk); oleg.stelya@gmail.com (O. Stelia); veda.kasianiuk@gmail.com (V. Kasianiuk);
ORCID: 0000-0002-8043-0785 (I. Krak); 0000-0003-0739-9678 (O. Barmak); 0000-0002-1068-769X (V. Kuznetsov); 0000-0002-5048-2576 (S. Kondratiuk); 0000-0002-1453-501X (O. Stelia); 0000-0003-3268-303X (V. Kasianiuk);
© 2023 Copyright for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org) Proceedings

methods or approaches for much smaller problem solving (so-called "small data") [1,2]. It means, as a result, that individual researchers, before using some cloud solutions as well as investing in more expensive hardware, make use all of the devices they have available hands-on, like gaming graphics cards, cards for data centers and built-in video cards for laptops [3]. As a result of this, there has to be a stage for a computation cost estimation before upgrading on more advanced hardware. However, one may argue, that the misconception about using much more advanced hardware may not always be useful and hence economically feasible: the graphic cards for large workloads (for big datacenters, for instance) may have seem an overkill for the simple job, and regular low tier consumer-grade hardware may not work intermediate jobs, since on some steps of calculation, there may appear more clearly some memory bottlenecks, that may in turn create performance decline.

Why is it so important? During the last years, in the 2020-2021, because of COVID-19 pandemic, there was a significant shortage of semiconductors, in particularly consumer grade graphics cards, used so widely in research. This caused drastic effects in global supply chain, not only in advanced semiconductors, but also other types hardware as well; which in turn made this hardware very expensive – so the cost of research too. The availability of the low or entry level hardware, was also questionable. While the desktop processors were not that hardly affected by the costs (not more than 30% increase in price), in contrary, the graphics processors were hugely affected, when the prices raised twofold or more. It created a dilemma: on one hand they were available, on the other hand they cost a lot of money, so the researcher had to decide the priority – being able to study the problem in detail in large scale or postponing the research. One may argue that some graphic cards were less affected by the price change like graphic cards for data centers and for businesses (Nvidia Quadro and AMD FirePro series) and one could take a hand on one to continue the research, however, they were likely part time solutions. After few years of advancements in technology and restoration of global supply chain these effects were highly mitigated. As a result, as of today, the problem has slightly resolved. Firstly, there are now three companies that make graphic cards for computers: Nvidia, AMD and Intel, that are available for the research purposes. Secondly, now there are many proprietary and open-source libraries that can available to use the graphic cards of any of 3 vendors available on the market – for instance libraries such as CUDA [4] (Nvidia), Microsoft DirectML [5] (AMD and mobile devices), Intel One API [6] (Intel graphic cards and computing devices). Also, as a promising feature – there is quite a big availability of multi-core processors, made by 2 of these vendors, which gives the hope to use them for specific cases. However, we must not underestimate the cases when such a disruption may occur in a global supply chain again. Even now there seem some possible situations, when this problem may occur again in larger scale because of some economical or geo-political problems. So the presence of the three vendors is good, but we have to put the question: what would we do, if similar problem happens again? If such a problem may have happened again, individual researchers should consider finding such solutions, that either may help them utilize outdated hardware instead of investing in it when it is not a feasible solution. Discussing such a problem in a retrospective, we want to address again such a problem: when a researcher faces a problem in buying some hardware for his research, because of external effects, affecting his solution, he has to choose to either to apply ineffective solution or to refit or fine-tune the hardware for a specific job (scaling down the machine learning model or use it for tasks like small data learning), so as the research could have been continued in place, and not being affected by some external factors.

We believe that the solution, that makes one to adapt is a better one.

Therefore, the purpose of this investigation is to find given hypothetical problem solution. For this purpose, we proposed to formulate and solve the following set of problems:

- to propose an idea for testing and comparing the performance of individual devices;
- to propose tasks for comparing (benchmarking) devices in deep learning tasks;
- to find the lower and upper limits (performance margin) of the use of such devices;
- to find an opportunity to use more computing power for a more complex task;
- to provide a simple solution for selecting the appropriate device for the job.

According to these problems, we split up our paper in sections as follows: in chapter 2 we discuss the related works and possible solutions, in chapter 3 we discuss the experiments on benchmarking, in chapter 4 we address some problems that may occur in experiments, in chapter 5 we discuss the results of our experimental study and in 6 – we write some key insights about the study.

2. Related Works and Proposed Solution

This section discusses the problem of small data and the approaches to process it with a range of hardware available for data analysts and scientists in artificial intelligence area.

2.1. The small data problem and the solution

If we compare the availability a high-performance computation solution from the start of computing to a nowadays, we see quite many shifts of paradigm of computing – centralized, decentralized, remote and local, which made a big turn. Nowadays, however the users have a possibility to choose whether to use remote solutions, like cloud services or to use local services based upon the desktop computer or small computational server. The benefits of the latest solution are obvious – full control on the experiment, since all the available resources are dedicated to a specific computational task. However, despite such wide range of possibilities, sometimes the researchers have to guess the setup, in order to build up the experimental system for a specific purpose [7]. In order to do so, there must be a measure to estimate the capabilities of current system. Using well-known methods, one can identify the key relationships between the used resources on a small- and the big-scale task, based on a test data.

Sometimes, it may happen, that such tasks to be solved, lay just between the area of small-scale tasks and really bigger tasks, which have to be made on dedicated servers or cloud solutions. We consider naming these tasks "small data solving" tasks, since they lay behind cutting edge solutions (big models) and there is a margin for unknown, where in order to obtain the parameters of the desired system, sometimes is better to use a small data task in order to estimate the capabilities, to seek the positive and negative parts of the used solution.

Since, here we discuss mainly on commonly available desktop or laptop solutions, we will focus on the capabilities of their architecture – CPU, GPUs (if available) and memory of the system.

To figure out the unknown part, given a small data solving task, we think is a better way is to use commonly used datasets, related to image recognition, machine learning, as well as proper hardware that was proven suitable perform such a task [8, 9]. Using such an approach, we can be reassured, that not only the proposed hardware answers our needs in terms of efficiency, raised power consumption and its own costs, but also suitability for a given small data solving task, which to be discussed further in the paper, using given hardware and proposed methods. We propose, in order to do so, also compare the capabilities of the processors against the graphic processing units, available in our laboratory, so as we can infer some key insights from this comparison as well.

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

We want to address here that despite the fact a lot of a solutions are available online, we can't just easy rely upon them as well, as their metrics: for instance, there are a plenty of solution to estimate the calculation capability of a computer system, using a benchmarks to estimate a performance level of a system in one specific task (for instance, calculating the lighting in the 3D scene, encoding the video, calculating some complex mathematical expression or formula [10-12]), in general these solutions may give a raw estimate for a performance [13-15]. For obvious reasons, the estimate of performance in a 3D rendering does not mean it works the same way to solve a system of linear equations, as well as training a deep convolution network. For these purposes, we must develop our benchmarks, in order to be more sure about our assumptions. We also may be aware of utilization of the system – both under-utilization, as well as overload, since in both cases they may affect the results of the benchmarking and our assumption about efficiency of a certain computational device.

However, this may be helpful enough to assume either it is suitable for specific tasks we need. For these specific purposes we will focus on different machine learning models, as well data dimensionality reduction methods, such as singular value decomposition [16], T-stochastic neighbor embedding [17] and autoencoders. We also consider widening our experiments to compare the behavior of different devices on deep neural network constructs, so as we can leverage most valuable information of behavior of our test systems in different conditions. We hope this may be very helpful for a specific task, related to a small data problem solving. Using our estimates, we can check either we use a proper hardware for a given task, or it has to be upgraded; we also consider some techniques in order to enhance our data or tweak the hyperparameters of the methods we are using, which may be helpful for dimensionality reduction, data classification and clustering tasks, which are tightly connected to small data and its processing, as we will discuss more in detail further in paper.

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

This section discusses mostly the experiments made with the general purpose CPU and CPU devices for home desktop machines to implement different tasks for machine learning, in particular training deep neural network models, using our hands-on equipment available in the laboratory.

3.1. The general scheme of an experiment

In order to proceed an experiment, we propose an approach, that allows us carefully to study different devices and make a raw estimate about their performance. In order to do so, we benefit from a controlled environment, which contents of a local or remote desktop machine (depending on the task), which allow all necessary controls to measure the utilization of resources, using either standard task manager (or device monitor), or the specific software, dedicated to a device (for instance GPU resource monitor). We benefit from this data, since we can establish usage of the resources before and under load. Since, our experiment would be likely less affected by background tasks.

In order to perform a data analysis task, we carefully prepare the data into our environment; the key is to keep the data within the system memory (either RAM, video RAM or RAM and VRAM combined), so as we would not be affected by caching the data from the disk, but focus on RAM-VRAM transfers, that would be likely faster than using page file. The data are pre-extracted features, generated by cascading model, which includes: singular value decomposition (SVD) [16], T-stochastic neighbor embedding (T-SNE) [17], and K-means clustering [18]. This allows us for being *focused solely on execution time and the factors that may affect it in deep neural network tasks*, such as initialization of the network, training of the network, as well as testing the network on the test data.

In order to document an experiment, we will follow the info from the resource monitor, as well as execution in the Jupyter environment – the console of the Jupyter server (it prints timestamps when the work stopped and autosave was created), as well as Jupyter notebook output and the resource monitor. Using such a means, we can control the flow of the experiment and identify if the Python kernel may have stopped or there is throttling of the processor or the GPU. In theory, based solely on the graph, we could identify the boundaries of each cycle of load – either next epoch or loading a new batch of the data. This may also help to figure the load (in %) and identify under load or overload.

After the experiment we focus not only on numerical data, but general performance, making some notes in our workbook if there were some issues with some device or algorithm and why they may have happened. Then, we will carefully analyze the behavior of this device few more times to find the possible hints of such behavior. We consider scaling the task, so it may help identify problems.

Since our test systems consist of different devices, we will try to benefit from their features, if there are available. We also analyze the behavior of the system, e.g. performance rate per iteration.

Since our scheme touches both processors and GPU devices, we will use it as a general template and adapt it, with respect to properties of each device, so they won't differ significantly. 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 more in detail our experimental setup to test different processors, we tested hands-on in our laboratory environment. It comprised of set of desktop and laptop computers, based on Intel and AMD processors such as: AMD A10-9620P, Intel Core i5-6600k, AMD Ryzen 5 3600X, AMD Ryzen 7 4800H using Windows 10 and Linux (Ubuntu 18 and OpenSuse 15) computing environment.

We decided to set up our systems to lower computational boundary or 16 GB of RAM, because some of our experiments over exceeded the 8 GB memory in the most of the systems we had hands on. Since we decided to focus our experiments on a specific pieces of hardware that satisfied this minimal requirement (system memory size). This requirement is caused by the fact, that we tested our equipment against different algorithms, like deep neural networks, some of them were computationally intense and needed more than 8 GB of memory. Hence, the intents of our experiments were not to prefer one device over another, as well as some vendor over another, but explore their capabilities, in context of different state-of-art methods of deep learning. We hope that after some tweaks, some of these devices and systems could pass the most experiments in the future.

In our text environment, we tested 2 different computers with AMD Zen2 [19] processors, including 1 desktop processor (AMD Ryzen 5 3600X) and one for mobile systems (AMD Ryzen 7 4800H). The specifications of the processors are depicted in the Table 1; note, we used also performance metric, using a well known PassMark benchmark and PassMarkScore [20].

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 reason we are using this benchmark quite straightforward: this text includes a range of different tests, that involve the different capabilities of the test system and averaged between the tests. This makes us confident about the results to compare the benchmark performance with our experimental results related to deep learning (e.g training, testing and initialization).

In order to perform our tests, we make use of verified and well-known software and libraries, being widely used in scientific and data analyst community. We benefit from Jupyter test environment, based on Anaconda 3.7 Python distribution for all of our environments (Windows and Ubuntu). In order to perform deep learning tasks, we utilize TensorFlow [21]. We also tried to use CUDA acceleration in some test systems, if it was enabled (the desktop machine had a GPU, but it was disabled).

Since our intent is to compare validity of usage of known benchmarks, we performed our own experiments to benchmark our laboratory equipment. We also hope that this will give us better understanding of results (Table 2).

The numbers in table 2 in superscript denote following: 1 is for Windows test environment, 2 is for Ubuntu test environment, 3 is for environment with enabled CUDA. If there are two numbers, for instance, 1 and 3 – it means usage of Windows environment with CUDA enabled.

During our experiments we figured that the hardware may behave differently from what we generally expecting in such tasks. Firstly, *we found out, that usage of some OS may give advantage, but for deep neural networks it may be considered insignificant – around 1-2% for testing and training,* which are the most time-consuming tasks, so usage of OS is mostly the personal preference.

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

However, when we tested two systems, we found out that the performance of the system, using the laptop graphic card had shown us some problems: since the GPU in general are considered superior in comparison to CPUs, we found out, that in this experiment, the GPU had significant disadvantage if comparing to desktop processor. Since we were hoping that it needs some clarification. To do so, we decided to make a series of extra experiments, that will be able to explain, why the graphics card does not work as we expected, according to general performance of such devices in deep learning tasks.

4. The problem with graphics card performance and resolution

This section discusses the problem of finding bottlenecks and situations when the GPUs are not an appropriate device to perform a deep learning task and hence discover an area of maximal efficiency within memory and calculation range to get the desirable level of GPU efficiency.

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

According to previous experiments, we figured out that we had some problems using a laptop graphics card (RTX 2060 mobile) on our custom dataset. Since we are unable to track which was a cause of the problem, we decided to perform a controlled experiment, excluding some conditions, that may affect our experiment. In order to do so, we decided to use different libraries for neural networks instead of we used to (TensorFlow with CUDA), to use a desktop system based on previously tested AMD Ryzen 3600X and desktop GPU AMD Radeon RX6500XT to leverage our tasks. We also decided to *rely upon the publicly available datasets in order to not be obscured by some results we got*. However, for experimental purposes, we may try to use this data for other tasks.

Since, here we were using a TensorFlow and PyTorch with its adapter (backend) Direct ML, which allows us to use it with our Radeon RX6500XT GPUs. While it may not be a fair comparison against Nvidia RTX 2060 Mobile, our *goal is to find some possible caveats or losses of performance as we faced with our GPU, which would likely occur with other GPU in same circumstances*. So here we would be testing GPU ability against data, not GPU vs GPU.

In order to estimate the capabilities of our test system we will be used to test fixed number of neural network architectures, as shallow networks, deep autoencoders, convolution and recurrent convolution networks, as well as general adversarial networks. This would allow us to test hardware against different architectures one by one and estimate the efficiency level for these devices and to find an effective range of usage for a CPU or GPU we are currently using in the experiments below.

We also proposed an experiment, which may, in theory, allow as to find the performance curve of a graphic card, in comparison to a CPU we are currently using. The suggested approach is quite simple and straightforward – to simulate the linear dimensionality reduction by exploiting the lambda functions into an optimization procedure of an autoencoder [22]. We hope that this simple task would help us to answer the key problems we faced in previous experiments.

4.2. Experiments with deep networks and tiny datasets

We tested various models including normal networks, convnets, and autoencoders on tiny datasets (fashion M.N.I.S.T. [23] and M.N.I.S.T. numbers [24]) but found that the graphics cards were not useful in this scenario. Due to the low graphics card performance, we decided to study autoencoders which are more flexible than other models. Overall, our findings give us a hint, that use of graphics cards in the analysis of these datasets may not be as effective, as we thought before, and that other approaches, such as exploring more sophisticated models, may gain additional performance we are targeting here.

4.3. Experiments with autoencoders and the different lambda functions

The first idea, which makes autoencoders more flexible, is the possibility of forward and reverse transformations. Since we can change the transformation and create the best transformations (actually an infinite number), we can also send a lambda function. For example, this function describes the property of weights, a target function, etc. If this function is changed, it is possible to increase the overall efficiency and to a few realizations of the same model, which may use less resources of the graphic card (memory, cores, etc.).

We propose to use these lambda functions, which do this: calculate the level of orthogonality of the weights; calculate the norm of the matrices; calculate the independence of weights. If we use all these functions, we can build a linear autoencoder [25], which looks like the singular value decomposition. But, if we do not use any, we create a non-linear autoencoder, which creates a transformation in a reduced dimension. The performance with different functions demonstrates in Table 3.

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

Here we have interesting results: the use of a non-linear (normal) autoencoder gives the difference in performance, or a processor (here AMD Ryzen 5 3600X) is the most 3 times more efficient than a graphics card (here AMD Radeon R.X. 6500 X.T.). But, on the contrary, when using more lambda functions, this difference was inverse - the graphics card had more than five times more than a processor. Also, we can see that before a while, the calculation times of a graphics card were practically the same.

This gives an idea, that there is a performance bottleneck, and therefore, if the card has passed this bottleneck, the performance gets increased. Also, this gives the best explanation with our Nvidia RTX 2060 mobile card: why in the last experiment it does not give an advantage. To verify our hypotheses, additional experiments are needed. Now we know that in simple jobs there is no need for a graphics card. But, when we do more time-consuming job, the GPUs start giving the best results. Indeed, we think these results could improve. But to think like that, you have to do what the experiments say: when the cards are the most efficient and also if we can find another border, when the card can lower its level of efficiency and thus, we can find an interval of effectiveness.

4.4. Experiment with convolution networks for human face recognition

The experiment we're doing here, dedicated to human face recognition. Here, we use data available online - FER2013 [26] and a convolution network that does the classification of face expressions. Because here, we are not looking for precision, but performance, this work is a work of checking devices. To verify our work, we use the AMD Radeon resource monitor [27] (Figure 1), and used the Anaconda console that writes the necessary information (Figure 2).

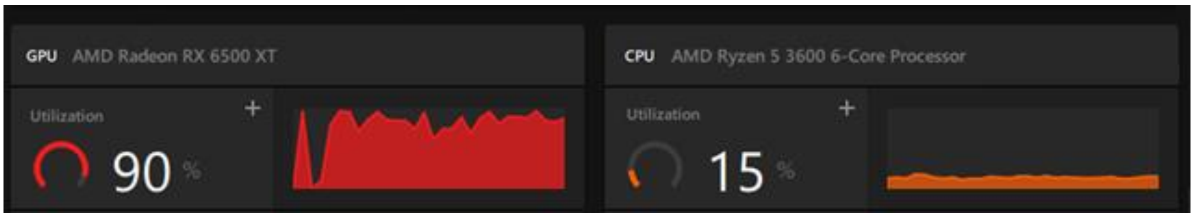
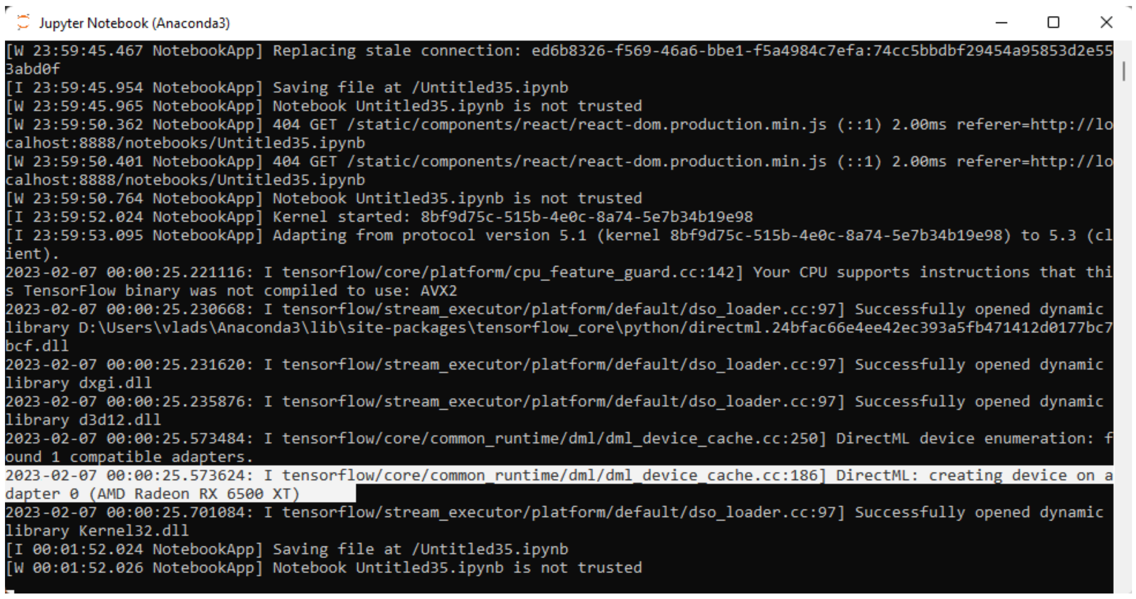

GPU AMD Radeon RX 6500 XT
Utilization
90 %
CPU AMD Ryzen 5 3600 6-Core Processor
Utilization
15 %

Figure 1: AMD Radeon Adrenaline Edition performance monitor during the experiments



```
[W 23:59:45.467 NotebookApp] Replacing stale connection: ed6b8326-f569-46a6-bbe1-f5a4984c7efa:74cc5bbdbf29454a95853d2e55
3abd0f
[I 23:59:45.954 NotebookApp] Saving file at /Untitled35.ipynb
[W 23:59:45.965 NotebookApp] Notebook Untitled35.ipynb is not trusted
[W 23:59:50.362 NotebookApp] 404 GET /static/components/react/react-dom.production.min.js (::1) 2.00ms referer=http://lo
calhost:8888/notebooks/Untitled35.ipynb
[W 23:59:50.401 NotebookApp] 404 GET /static/components/react/react-dom.production.min.js (::1) 2.00ms referer=http://lo
calhost:8888/notebooks/Untitled35.ipynb
[W 23:59:50.764 NotebookApp] Notebook Untitled35.ipynb is not trusted
[I 23:59:52.024 NotebookApp] Kernel started: 8bf9d75c-515b-4e0c-8a74-5e7b34b19e98
[I 23:59:53.095 NotebookApp] Adapting from protocol version 5.1 (kernel 8bf9d75c-515b-4e0c-8a74-5e7b34b19e98) to 5.3 (cl
ient).
2023-02-07 00:00:25.221116: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that thi
s TensorFlow binary was not compiled to use: AVX2
2023-02-07 00:00:25.230668: I tensorflow/stream_executor/platform/default/dso_loader.cc:97] Successfully opened dynamic
library D:\Users\vlads\Anaconda3\lib\site-packages\tensorflow_core\python/directml.24bfac66e4ee42ec393a5fb471412d0177bc7
bcf.dll
2023-02-07 00:00:25.231620: I tensorflow/stream_executor/platform/default/dso_loader.cc:97] Successfully opened dynamic
library dxgi.dll
2023-02-07 00:00:25.235876: I tensorflow/stream_executor/platform/default/dso_loader.cc:97] Successfully opened dynamic
library d3d12.dll
2023-02-07 00:00:25.573484: I tensorflow/core/common_runtime/dml/dml_device_cache.cc:250] DirectML device enumeration: f
ound 1 compatible adapters.
2023-02-07 00:00:25.573624: I tensorflow/core/common_runtime/dml/dml_device_cache.cc:186] DirectML: creating device on a
dapter 0 (AMD Radeon RX 6500 XT)
2023-02-07 00:00:25.701084: I tensorflow/stream_executor/platform/default/dso_loader.cc:97] Successfully opened dynamic
library Kernel32.dll
[I 00:01:52.024 NotebookApp] Saving file at /Untitled35.ipynb
[W 00:01:52.026 NotebookApp] Notebook Untitled35.ipynb is not trusted
```

Figure 2: Python interpreter console and Jupyter environment (Anaconda)

After we experimented, we figured some important results (Figure 3).

```
Epoch 1/15                                        Epoch 1/15

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

449s 1ms/step – loss 1.7944 – acc 0.3116          107s 238ms/step – loss 1.7978 – acc 0.3112

Epoch 2/15                                        Epoch 2/15

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

447s 998ms/step – loss 1.4812 – acc 0.4364        22s 49ms/step – loss 1.4888 – acc 0.4260
```

Figure 3: Performance during the experiments. CPU (on the left) and GPU (on the right).

According to this test, we found that the times are very important and with each iteration, the total efficiency is increased. Also, according to the AMD monitor, approximately 1.2 GB of VRAM resources have been used, which are the weights of our network and of the data. Besides, if we compare epoch one and epoch two results, we find that it takes approximately 80 seconds to send data from RAM memory to video memory. The time required for computations is comparable for small datasets, but when the task becomes more complex, the performance may not be as prominent. During one iteration, the performance difference between the graphics card and processor was approximately 4.5 to 1, and for larger numbers of iterations, the difference could be as much as 20 to 1, demonstrating that graphics cards also require significant computation time to process and load the data. However, as the number of iterations increases, the overall time can also increase dramatically.

Based on study of library calls in Jupyter console (in particular, library calls during the initialization process as shown on Figure 2), one can explore the Direct ML interface to leverage tensors in AMD architecture. In order to transmit data from and into AMD GPU, the adapter has to use DirectX 12 library to transform data using Direct ML and DirectX 12 library to transform tensors, which are not directly compatible between the AMD and Nvidia, since they have different memory structure. Because we address this problem, now we can prepare better the further experiments.

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) are known to require a significant amount of memory for training due to their complex architectures and large number of parameters. Therefore, we conducted an experiment focusing on these memory-intensive networks, specifically RCNN convolutional networks and generative adversarial networks. In order to get a consistent result, we use the system with the same processor and video card, but another realization of the networks - the PyTorch library with Direct ML. We utilized different techniques to process two datasets. To perform image segmentation for the "Cityscapes Dataset", we employed the hidden network RCNN which effectively segmented images of pedestrians. On the other hand, for image generation, we leveraged the D.C.G.A.N. model.

Indeed, in two experiments, it is found that there was not enough memory.

To find the solution, we suggest using the graphics card and the processor at the same time. How it works: the model was loaded into the video memory and system memory, we do the calculations (loss), and little, we use the card to calculate the gradient. Although we would prefer to calculate the entire network at once, we faced memory limitations, with the memory usage reaching approximately 3.8 GB of video memory and 12.8 GB of system memory (shared). As you can see, processing such a large memory chunk while utilizing all VRAM was not an easy task. (Figure 4).

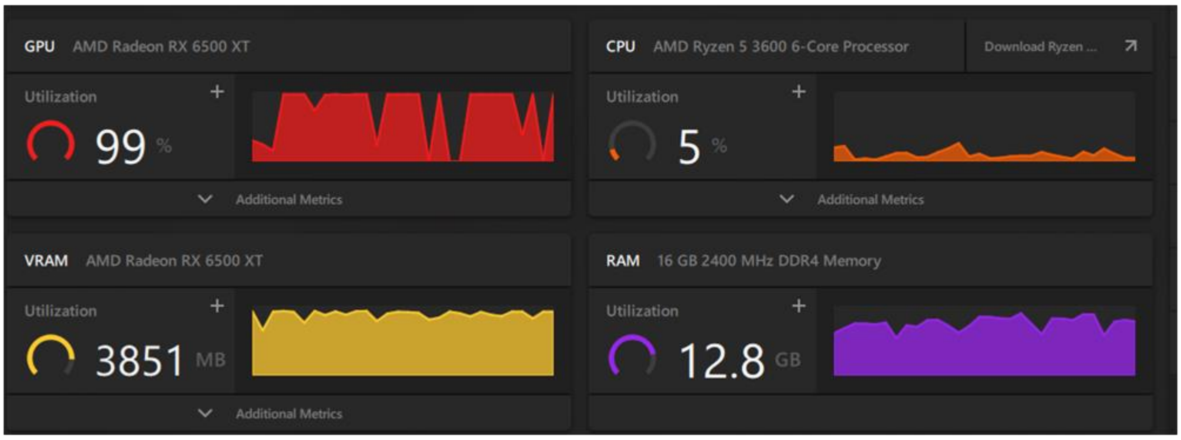


Figure 4: Memory usage, GPU and CPU time

Another effect that we could not find: the graphics card, if we used it with a processor, improved results by 2.74 times, if we would compare it with processor alone. This means that the card may be useful when the work needs to be finished. But the problem with video memory bottlenecks is very big, because huge chunks of memory have to be sent after each iteration or epoch. Also, as can be seen, there are needs of fine tuning of network or data.

5. Discussion

During the experimental runs in different conditions, we figured some problems to discuss. Foremost, we must reassure, that we use a proper devices and techniques to solve our tasks; either it can be done using different tweaks, optimizations or other approaches, either related to data, specifications of the test system as well as algorithms and their particular implementations.

Using such an approach, we can be confident not only in our results, but also, we have proper performance, regarding work time and energy efficiency.

As we said in introductory chapters, *it is very important to take into consideration which task has to be solved – to train a network, or doing such efficiently*. Second, but not least, *scaling up and down the tasks may make us discover that the used hardware answers our needs*; sometimes for a complex tasks one need very expensive workload, but in others, like “toy datasets” one can rely on a processor.

We think that solution finding is a thoughtful process which may need to take in consideration many factors. Firstly, one has *to identify the problematic parts of the system, that may not work properly or at least show a behavior, different from one should expect from*. By using this knowledge, we can be confident about the test system, as well as to predict behavior if it was scaled up or down.

We see also, that *scaling up and down does not give proportional gain and loss of performance*. In fact, this process is non-linear and we may expect the desirable performance in certain circumstances and certain conditions. For instance, if we scale down the dimensions of the network, according to the size of the input vector, we can find that some information may be lost, since not all the features would be present, as of architecture of a certain size. Hence, *it would be a good idea to follow the guidelines related to the data and network, as it was tested by many researchers in the area*.

We also must consider the time needed to expect the results; since there are general rules of how many iterations are needed to get a proper result for a specific piece of a hardware, we may expect that this time may be higher, than we considered before.

So as happens with relative fast tasks, that may have a purpose to establish the most appropriate scenario and hyperparameters in series of tests.

We also found very helpful *to conduct an experiment in a controlled environment, where we can be reassured that there are any factors (e.g., user applications), that may affect the experiment and hence make the experiment incorrect*. This environment must also allow one to identify the problems and to fix them on the fly, optimizing for speed, accuracy or price, including hardware.

In overall, we see the task of *improving the hardware usage for a specific deep learning task some sort of "optimization", where we not just optimize an algorithm, but its behavior in a test environment, in order to enhance the overall efficiency in complex*. If we thoughtfully identify key parameters of our setup, that affect efficiency, then we likely would use our setup in smart way, so it would answer needs in term of performance of models but also in terms of resource optimization.

6. Conclusion

In overall the main results from the article would be summarized as follows.

We have figured that *one should not only rely upon a specific architecture, but study its good sides and weaknesses, and based upon the study decide which of them is more appropriate for the task*. We found, for instance, that hardware, needed to use on complex computer vision tasks, like for instance in RCNN is quite different from that is needed to train a network on a MNIST dataset.

The process of tweaking is not quite straightforward, but rewarding in *new knowledge about possibilities of a certain hardware to perform on specific tasks*, like, for instance, image recognition. However, we also found some drawbacks of modern computer vision related neural networks, that sometimes may have some disadvantages, if compared to a classic computer vision methods.

Thus, to unleash all the power of these novel methods, *one may need much more advanced setup or, if that not possible, scale down the data, with its own known negative effects on accuracy*.

It is also worth noting that the competition between the different vendors is quite high; since, the computational capabilities of certain devices may par with a similar one but from other vendors. We also figured out, that sometimes the devices having similar scores in ideal conditions like computer benchmarks and having the same score, may behave totally differently in other tasks; for instance, the difference between the desktop and laptop processors of same architecture (Zen2) happened much bigger than we may have expected; we believe, though, that desktop processors of a similar class and performance may have shown much more consistent and much more predictable results.

To really achieve the research goals, *one need ideally a set of a different devices, dedicated to a task, for instance processors for one tasks and graphic cards for another or even more, use them in conjunction*. Such an approach may guarantee us exploiting most positive characteristics of 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 expand the results and use other devices for more complex tasks and to improve, as a result, our current results.

7. References

[1] A. P. Adedigba, S. A. Adeshina, A. M. Aibinu, Performance evaluation of deep learning models on mammogram classification using small dataset, Bioengineering, 9 4 (2022) 161-181. doi:10.3390/bioengineering9040161

[2] M. McBride, N. Persson, E. Reichmanis, M.A. Grover, Solving materials' small data problem with dynamic experimental databases, Processesб 6 7 (2018) 79-96. doi:10.3390/pr6070079

[3] H. Gong, Y.Liu, X. Chen, et al., Scene optimization of GPU-based back-projection algorithm, The journal of supercomputing, 79 (2022) 4192–4214. doi:10.1007/s11227-022-04785-w

[4] S. Nangla, GPU Programming using NVIDIA CUDA. International journal for research in applied science and engineering technology, 6 6 (2018) 79–84. doi: 10.22214/ijraset.2018.6016

[5] A. Raman, C. Hoder, S. Bisson, M. Branscombe, Azure AI services at scale for cloud, mobile, and edge: building intelligent apps with azure cognitive services and machine learning. O'Reilly Media, Incorporated, 2022.

[6] M. Krainiuk, M. Goli, V. R. Pascuzzi, OneAPI open-source math library interface, in: International workshop on performance, portability and productivity in HPC (P3HPC), St. Louis, MO, USA, November 14-19 2021, 2021, pp. 22-32. Doi:10.1109/p3hpc54578.2021.00006.

[7] Y. V. Krak, Dynamics of manipulation robots: Numerical-analytical method of formation and investigation of computational complexity, Journal of Automation and Information Sciences, 31 3 (1999) 121-128. doi:10.1615/JAutomatInfScien.v31.i1-3.170

[8] J. Lee, J. Rhim, J. Kang, D. Ha, S3NAS: fast hardware-aware neural architecture search methodology, IEEE transactions on computer-aided design of integrated circuits and systems. 41 11 (2021) 4826-4836. Doi:10.1109/tcad.2021.3134843

[9] J. Anderson, Y. Alkabani, T. El-Ghazawi, Towards energy-quality scaling in deep neural networks, IEEE design & test, 38 4 (2021) 27-36. Doi:10.1109/mdat.2019.2952328

[10] I. G. Kryvonos, I. V. Krak, O. V. Barmak, A. S. Ternov, V. O. Kuznetsov, Information technology for the analysis of mimic expressions of human emotional states, Cybernetics and Systems Analysis, 51 1 (2015) 25-33. doi:10.1007/s10559-015-9693-1

[11] Y. V. Krak, A. V. Barmak, E. M. Baraban, Usage of NURBS-approximation for construction of spatial model of human face, Journal of Automation and Information Sciences, 43 2 (2011) 71-81. doi:10.1615/JAutomatInfScien.v43.i2.70

[12] I. G. Kryvonos, I. V.Krak, O. V. Barmak, R. O. Bagriy, New tools of alternative communication for persons with verbal communication disorders, Cybernetics and Systems Analysis, 52 5 (2016) 665-673. doi:10.1007/s10559-016-9869-3

[13] H. H. Holm, A. R. Brodtkorb, M. L. Sætra, Performance and energy efficiency of CUDA and opencl for GPU computing using python, Parallel computing: technology trends 36 (2020) 593 – 604. Doi:10.3233/apc200089.

[14] W. Varghese, N. Wang, D. Bermbach, et al., A survey on edge performance benchmarking. ACM computing surveys, 54 3 (2021) 1–33. Doi:10.1145/3444692.

[15] Y. Lin, A. Barker, J. Thomson, Modelling VM latent characteristics and predicting application performance using semi-supervised non-negative matrix factorization, in: 2020 IEEE 13th international conference on cloud computing (CLOUD), Beijing, 19-23 October 2020, (2020), P. 470-474. Doi:10.1109/cloud49709.2020.00069.

[16] Z. Fang, F.Qi, Y. Dong, Y. Zhang, S. Fenget, Parallel tensor decomposition with distributed memory based on hierarchical singular value decomposition Concurrency and computation: practice and experience, 2021, Doi:10.1002/cpe.6656

[17] M. Balamurali, T-Distributed stochastic neighbor embedding, Encyclopedia of mathematical geosciences. Cham, 2021. Doi:10.1007/978-3-030-26050-7_446-1

[18] M. Alian, G.Al-Naymat, Questions clustering using canopy-K-means and hierarchical-K-means clustering, International journal of information technology, 14 (2022) 3793-3802. Doi:10.1007/s41870-022-01012-w

[19] D. Suggs, M. Subramony, D. Bouvier, The AMD "zen 2" processor, IEEE micro, 40 2 (2020) 45-52. Doi:10.1109/mm.2020.2974217

[20] Y. Wang, V.Lee, G.-V. Wei, D. Brooks, Predicting new workload or CPU performance by analyzing public datasets, ACM transactions on architecture and code optimization, 15 4 (2019) 1–21. Doi:10.1145/3284127

[21] M. Ramchandani, et al., Survey: tensorflow in machine learning, Journal of physics: conference series, 2273 1 (2022) 1-12. Doi:10.1088/1742-6596/2273/1/012008

[22] O. Fontenla-Romero, B.Pérez-Sánchez, B. Guijarro-Berdiñas, LANN-SVD: a non-iterative SVD-based learning algorithm for one-layer neural networks, IEEE transactions on neural networks and learning systems, 29 8 (2018) 3900–3905. Doi:10.1109/tnnls.2017.2738118

[23] A. S. Henrique, et al., Classifying Garments from Fashion-MNIST Dataset Through CNNs, Advances in science, technology and engineering systems journal, 6 1 (2021) 989–994. Doi:10.25046/aj0601109

[24] A. Baldominos, Y.Saez, P. Isasi, A survey of handwritten character recognition with MNIST and EMNIST, Applied sciences, 9 15 (2019) 3169-3185. Doi:10.3390/app9153169

[25] H. Bourlard, S. H. Kabil, Autoencoders reloaded, Biological cybernetics, 2022. Doi:10.1007/s00422-022-00937-6.

[26] P. Giannopoulos, I. Perikos, I. Hatzilygeroudis, Deep learning approaches for facial emotion recognition: A case study on FER-2013, Advances in hybridization of intelligent methods. Cham, 85 (2018) 1-16. Doi:10.1007/978-3-319-66790-4_1

[27] J. Peddie, The GPU environment-software extensions and custom features, in: The history of the GPU - eras and environment. Cham, 2022. P. 251-281. Doi:10.1007/978-3-031-13581-1_7