Classification of X-Ray Images of the Chest Using Convolutional Neural Networks

Lesia Mochurad [a,*], Andrii Dereviannyi [a], Uliana Antoniv [b]

[a] *Artificial intelligence Department, Lviv Polytechnic National University, Lviv, 79013, Ukraine; lesia.i.mochurad@lpnu.ua, andriidereviannyi@gmail.com*

[b] *Department of Specialized Computer Systems, Lviv Polytechnic National University, Lviv, 79013, Ukraine; Uliana.s.antoniv@lpnu.ua*

***** *Correspondence lesia.i.mochurad@lpnu.ua; Tel.: +380-32-258-2404*

Abstract

A proven way to detect various injuries: from fractures to heart failure, is an X-ray. However, because this examination method depends on the doctor's visual analysis, it can lead to misdiagnosis, that is, the case when the early stage of pneumonia will not be recognized and treatment will be ineffective. This study proposes using a convolutional neural network to classify chest X-rays to solve this problem. To do this, we analyzed the materials on the classification using neural networks for different areas of computer vision. In particular, convolutional neural networks for medical use are considered. The classification model of images on a database that included 112 thousand captions and 30 thousand unique patients is trained. High accuracy values of 0.93 and completeness of 0.99 models were obtained. An analysis of the literature on the acceleration, parallelism, and synchronization of convolutional neural networks was performed. Their shortcomings are taken into account, and a new optimization approach is proposed. The classification results were compared with a parallel approach on a GPU and a sequential on a CPU. The model trains on the GPU is 6.13 times faster than on the CPU based on the proposed algorithm.

Keywords 1

Computer vision, image classification model, parallelization, acceleration, GPU, CPU.

1. Introduction

When it comes to identifying images, it is straightforward for us humans to recognize and distinguish the different features of the depicted objects. All because our brains are constantly subconsciously training on the same set of data, we can easily distinguish between various entities. In contrast, the computer looks at the world around it differently: it is an array of numerical values that form the critical aspects of an image or video that it tries to recognize. The principle by which the system interprets the picture is radically different from how people do it. To vision, a computer needs image recognition algorithms to analyze and understand what is happening in an image or its sequence. An excellent example is the identification of pedestrians and vehicles, which is possible due to the preliminary categorization and sorting of millions of prints – data provided by users [1, 2].

Medicine is a clear favorite for areas that require a reliable image identification system while generating a large amount of data on which you can train the same recognition [3]. However, the biggest challenge in collecting medical data is practical analysis and processing for their further use [4]. There are many methods of organizing the data obtained. We will consider one of them, namely the classification, because it is widely used in the medical field, for example, to detect the disease's symptoms.

IDDM-2021: 4th International Conference on Informatics & Data-Driven Medicine, November 19–21, 2021 Valencia, Spain;
EMAIL: lesia.i.mochurad@lpnu.ua (L. Mochurad); andriidereviannyi@gmail.com (A. Dereviannyi); Uliana.s.antoniv@lpnu.ua (U. Antoniv).
ORCID: 0000-0002-4957-1512 (L. Mochurad); 0000-0003-1456-1303 (A. Dereviannyi); 0000-0002-6792-043X (U. Antoniv).

The classification task is for a computer to analyze an image and assign it to an appropriate class, usually setting a label to a particular image. For us, the classification of images is elementary, but it is also a great example of the Moravec paradox – that simple for humans is difficult for artificial intelligence [5].

Early image classification depended on pure pixels – the computer split the image provided into individual pixels. The problem with this approach is that two photos of the same object may look different. Different backgrounds, angles, poses, and many other factors made it difficult for computers to recognize and categorize images. This problem is designed to correct deep learning [6]. The latter involves the use of computer systems known as neural networks.

Unfortunately, the task of classification is quite resource-intensive and when using weak computing power or a consistent algorithm to obtain the result can take a long time. In such cases, optimization or the possibility of using better resources is usually considered.

The aim of this study is to propose a parallel algorithm and analyze the advantages obtained in solving the problem of classifying X-ray images to detect pneumonia using the CPU and GPU.

2. Literature review

Classification of images in training often involves the use of convolutional neural networks (CNN). The latter proved their power when AlexNet won the ImageNet competition by a wide margin from other, more traditional neural networks. Since then, CNN has become one of the most promising machine learning algorithms. It is widely used to solve problems involving large-scale datasets. However, learning deep convolutional neural networks on large datasets is a highly intensive computational task and requires much time to learn. That is why the algorithm is subjected to parallelization, which reduces the load on one core, dividing the work between several [7-10]. Due to this, the use of the algorithm does not require spending days or even weeks to learn it.

There are two approaches: model parallelism, i.e., the model is divided between several computing nodes and trained on the same data, and data parallelism, if the data is distributed on several nodes and the same model is used for learning. Hybrid approaches using both parallelisms have also been proposed. Examples of hybrid systems are papers [11] and [12]. In hybrid approaches, a small number of nodes are grouped to teach the model using model parallelism. The data set is divided into groups to be processed simultaneously using data parallelism. They use the master-managed model, and the main task of the server is to update parameters centrally. The disadvantage of this approach is that because all groups access the same server, which ensures their interaction, a delay is created, which reduces performance.

One of CNN's most popular learning algorithms is stochastic gradient descent (SGD). It was demonstrated in the article [13]. The algorithm works iteratively: the model parameters are updated until they become optimal. Due to the dependence of the data on the model parameters between any two sequential iterations, parallel SGD can suffer due to the expensive interprocess cost of communication [14]. Recently, researchers have made many efforts to improve the scaling of the similar SGD algorithm [15-16]. Traditional synchronous SGD guarantees optimal parameter updates due to low parallel efficiency, which is achieved through frequent synchronization. Asynchronous SGD has been designed to address such performance vulnerabilities. This approach is quite popular, as can be seen from [17] and [18]. However, asynchronous SGD also has disadvantages. It requires more iterations to match the same accuracy, so for many processes, this increases the learning time.

In paper [19], the accuracy of classification of different models based on chest images is investigated. At the same time, the authors managed to achieve high accuracy – 96.81%. However, no studies have been conducted to process large-scale input data on the time of execution of such classifications, and, accordingly, no parallelization was performed.

Another example of an article related to medicine and the parallelism of CNN is [20]. But the method used in this work is a hybrid in which there is CNN and RNN.

Therefore, in the analysis of publications, no work was found that would demonstrate the parallelization of CNN on the CPU and GPU to classify pneumonia.

Some authors argue that parallelism of large-scale data is harmful and leads to deterioration of results; others say it is beneficial [21-23]. The absolute competitiveness of parallelization can show today, as we can obtain large amounts of data in free access, allowing more free experiments. Hence,

it is not surprising that the literature remains ambiguous because all of the above methods have pros and cons.

3. Materials and Methods

Deep convolutional neural networks are used in many tasks, such as image classification, work with sound, etc. [24-27]. Although CNN was developed in 2012, it is gaining real popularity right now. The main factors for CNN's success are the availability of large data sets and the high performance of modern computer systems.

For high results, this neural network requires large amounts of data. For example, one of the earliest CNNs, LeNet-5 [28], was taught to recognize handwritten numbers using the MNIST dataset, containing about 60 000 images in a training set and 10 000 images in a test set. The CIFAR-10 dataset consists of 60 000 32x32 color images, including 50 000 training images and 10 000 test images. We took an NIH Chest X-rays database for our research, including 112 thousand high-resolution pictures of 30 thousand different patients [29].

3.1. Database description

The database, which is considered in this paper, contains images of chest X-ray examinations, one of the most common medical examinations. One of the main obstacles to creating large data sets of X-ray images is the lack of resources to label many photos. Before releasing this dataset, Openi [30] was the largest publicly available source of chest X-rays, with 4 143 images available.

The NIH chest X-ray data set consists of 112 120 X-rays with disease labels of 30 805 unique patients [29].

Thus, the neural network's task is to divide all images into two classes: pneumonia and healthy, based on notions of chest X-rays (see Figure 1).

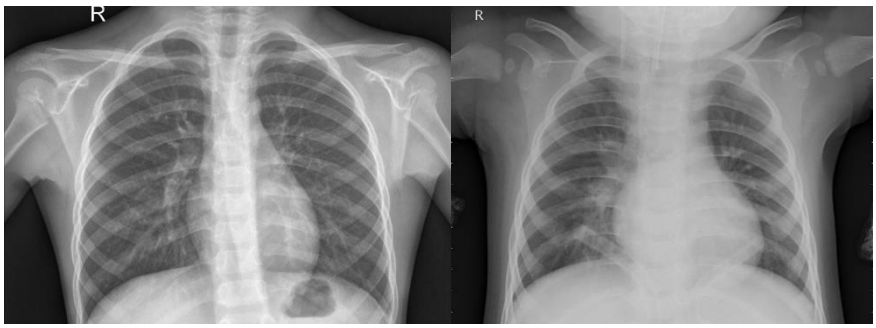

Figure 1: X-rays of the chest, left - without pneumonia, right – with pneumonia

Since this paper will compare the proposed parallel approach with the standard use of CNN, we first consider the operation of the convolutional neural network algorithm.

3.2. Cunvolutional Neural Network (CNN)

In tightly coupled neural networks, neurons are divided into groups that form successive layers. In them, each unit is connected to each neuron from neighboring layers. An example of this network is shown below in Figure 2:

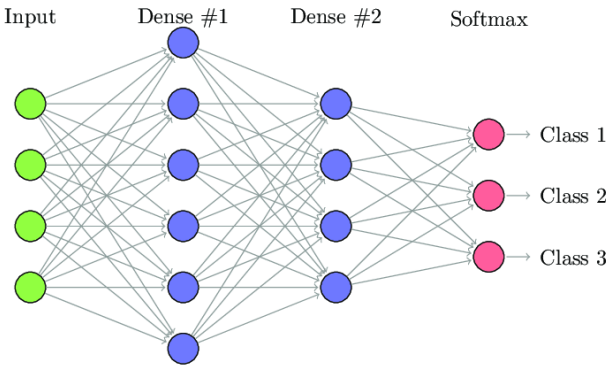

Figure 2: An example of a tightly coupled neural network

This approach works well to solve a classification problem based on a limited set of defined characteristics. But the situation is complicated when the data to be classified are images. We could transmit the brightness of each pixel as separate units to the input of our dense network, but then for it to work, it must contain tens or even hundreds of millions of neurons. One way to solve this problem and reduce the network is to reduce the scale of the photo itself, but then we lose information. Therefore, convolutional neural networks are used to solve this problem.

To begin with, to analyze the operation of convolutional networks, you need to describe the data structure with which they will work. For convenience, the image is stored in a 2d matrix, each of which characterizes a particular image pixel. The image consists of three such matrices for the RGB model, each of which corresponds to a specific channel - red, green, and blue. If the image is black and white, then one matrix is used. Each of the digits is in the range from 0 to 255.

The main element in the work of CNN is a matrix, which is a filter or core. To process it, we pass our image matrix and convert it based on filter values. The following formula calculates the value of the object map. The input of the image is denoted by f and the kernel by h. The indices of the rows and columns of the result matrix are denoted by m and n, respectively.

$$G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[j,k] f[m-j, n-k].$$

After placing our filter over the selected pixels, we take each value and multiply it by the corresponding value from the kernel. Summarize all the results and write in the appropriate place on the map of the initial characteristics. The filter is above pixels that have tens and zeros. Let each pixel multiply by the corresponding one in the filter and add all the results. We write down this result in a matrix. We move the kernel and repeat the previous steps. In the end, we get a matrix with new data. Because the image decreases with each subsequent iteration, the number of image convolutions is limited. Also, following the movement of the filter, the influence of pixels located on the outskirts is observed. They are much smaller than the center of the image. In this way, part of the information present in the picture is lost.

A frame is added to the image matrix, mainly filled with zeros, to solve this problem. Depending we use fillings or not, we deal with two types of convolutions - original and same. Valid – the original image is used. Same – uses a frame around the original image so that when convolving, the output gives a matrix of the same size as the original image. For the second case, the width of the frame must be equal to the following value:

$$p = \frac{f-1}{2},$$

where p – filling, and f – size of the filter (usually this value is odd).

One of the essential hyperparameters of the convolutional layer is the length of the step along which the filter should move. For the CNN architecture, this parameter is vital. If you want to overlap the receptive fields less or the spatial dimensions of the function map to perform the exercise, you need to increase the step. The following function is used to calculate the size of the output matrix:

$$n_{out} = \left[\frac{n_{in} + 2p - f}{s} + 1\right].$$

Convolution over volume – is an essential concept because it allows you to work with color images and apply multiple filters within one layer. But keep in mind that the number of channels contained in the filter and the images must match. To use various filters in the same image, we collapsed filters separately, and the results are combined at the end. We can find the size of the tensor we obtain using the following formula:

$$[n, n, n_C] * [f, f, n_C] = \left[\left[\frac{n_{in}+2p-f}{s} + 1\right], \left[\frac{n_{in}+2p-f}{s} + 1\right], n_f\right],$$

where n is the size of the image, f is the size of the filter, n_C is the number of channels in the image, p is the fill used, s is the step used, n_f is the number of filters.

Forward propagation consists of two steps. The first is the calculation of intermediate values of Z, which we obtain by convolving the input data from the previous layer by the tensor W and then adding the offset b. The second is the application of the nonlinear activation function to our intermediate value, which is denoted by the letter g. The following equations can demonstrate these steps:

$$Z^{[i]} = W^{[i]} \cdot A^{[i-1]} + b^{[i]} \quad A^{[i]} = g^{[i]}(Z^{[i-1]}).$$

Now let's move on to the vital attributes in the complex layers. First, all neurons in convolutional layer are interconnected. Second, some neurons have the same weight. This shows that the convolution has reduced the parameters to be studied. It is also worth noting that one value from the filter affects each element of the object map, which is crucial in backpropagation.

Consider the algorithm of the network in reverse propagation. This algorithm aims to calculate the derivatives and then use them to update the values of the parameters in a process called gradient descent. We want to assess the impact of parameter changes on the resulting feature map and the final result.

The problem of inverse propagation is to calculate the partial derivatives of cost functions: $\frac{\partial L}{\partial W^{[i]}}$ and $\frac{\partial L}{\partial b^{[i]}}$ – which are derivatives related to the parameters of the current layer, as well as values $\frac{\partial L}{\partial A^{[i-1]}}$ which will be transferred to the previous layer. At the entrance we have $\frac{\partial L}{\partial A^{[i]}}$. The first step is to obtain the intermediate value $\frac{\partial L}{\partial Z^{[i]}}$ by applying the activation functions to the input tensor $\frac{\partial L}{\partial Z^{[i]}} = \frac{\partial L}{\partial A^{[i]}} * g'(Z^{[i]})$. According to the chain rule, the result of this operation will be used later.

Then the matrix operation is applied - full convolution. For this operation, we use a core that is rotated 180 degrees. As a result, we have:

$$\frac{\partial L}{\partial A^{[i]}} = \sum_{m=0}^{n_h} \sum_{n=0}^{n_w} W \cdot \frac{\partial L}{\partial Z^{[i]}}[m, n],$$

where W – filter, and $\frac{\partial L}{\partial Z^{[i]}}[m, n]$ is a scalar that belongs to the partial derivative obtained from the previous layer.

In addition to convolution layers, CNN very often uses so-called aggregation layers. They are mainly used to reduce the size of the tensor and speed up calculations. These layers are superficial – we need to divide our image into different regions and then perform a specific operation for each of these parts. For example, for the highest layer of the pool, we select the maximum value from each area and place it in the appropriate place at the output. As in the case of the convolutional layer, we have two hyperparameters available – filter size and pitch. Last but not least, if you are merging for a multi-channel image, the merging for each channel should be done separately.

As we can see, CNN learning can be pretty slow due to the number of calculations required for each iteration. Therefore, to speed up the work, it would be advisable to carry out parallel computing.

3.3. Parrallelization on the GPU

In parallel calculations, the problem is divided into independent smaller subtasks that run simultaneously. The results obtained are recombined or synchronized to formulate the impact of the initial count. The number of tasks into which computations can be divided depends on the number of

cores. Which in turn are contained in the equipment. Unlike CPUs, which process operations sequentially, computationally tricky tasks on a GPU are distributed among thousands of processors, allowing you to perform calculations much faster [31].

Keras and TensorFlow technologies will be used to parallelize the algorithm. These technologies use the capabilities of parallel processing of the graphics processor. Due to the CUDA, computational tasks are performed sequentially on the CPU due to the C ++ software interface or in parallel on the GPU.

As mentioned above, a filter matrix with dimension n*n is used for a classical convolutional neural network. This process can be seen in Figure 3. This matrix is multiplied by a matrix of image pixels. These calculations are performed sequentially and independently of each other, which means that no analysis depends on the results of any other count. This lets us see that the convolution operation can be accelerated using a parallel programming approach and GPUs. This operation takes the most time in this algorithm and is the most voluminous because the image size is 500x500.

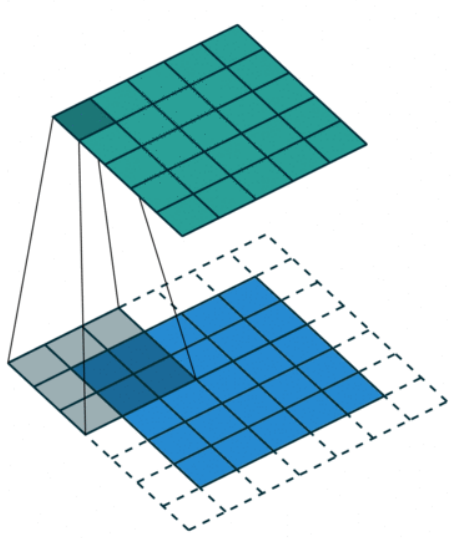

Figure 3: The image matrix multiplies the matrix of the kernel or filter

We can conclude that it is better to perform parallel calculations using graphics processors. To test this hypothesis, we will conduct experiments and test the results and learning time of the convolutional neural network sequentially on the CPU and in parallel on the graphics.

4. Results

Intel (R) Core (TM) i7-10750H CPU with the following characteristics is used for numerical experiments:

Base speed: 2.59 GHz;
Sockets: 1;
Cores: 6;
Logical processors: 12;
Virtualization: Enabled;

and GPU NVIDIA GeForce GTX 1650 Ti:

Dedicated GPU memory 4.0 GB;
Shared GPU memory 7.9 GB;
GPU Memory 11.9 GB.

In both cases, the same CNN model is used, which is presented in Figure 4.

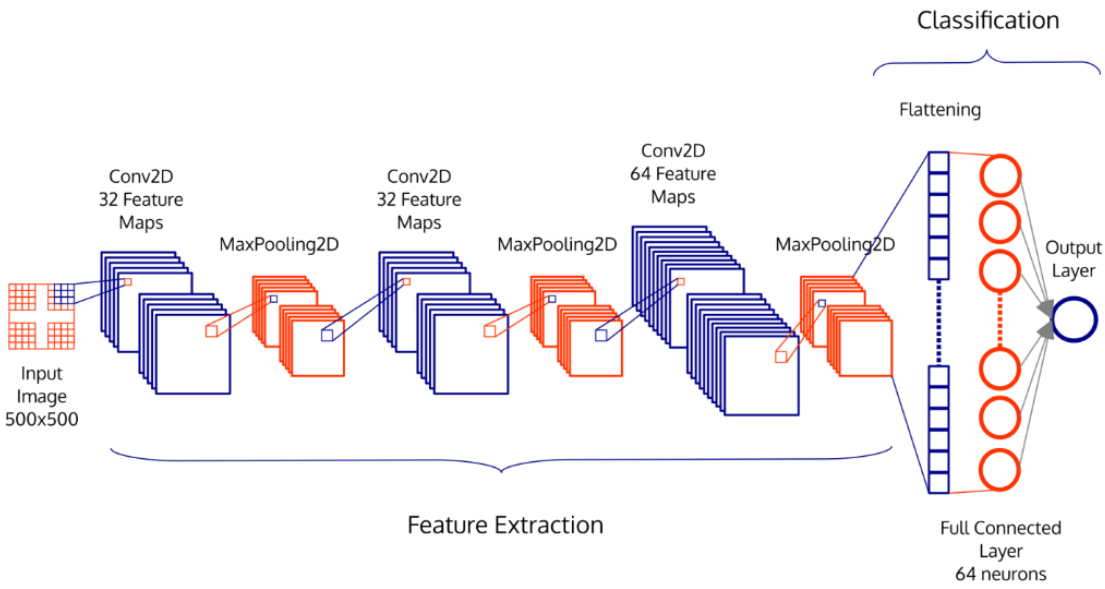

Figure 4: Schematic representation of the CNN model used

After training, the following results were obtained:

Table 1
The result of training CNN model using GPU

Epoch	Time, sec/epoch	Time, ms/step	Loss function	Accuracy	Validation loss function	Validation accuracy
1	91	344	0.5762	0.7623	0.2828	0.9308
2	91	346	0.3177	0.8887	0.2724	0.9192
3	93	353	0.2737	0.9093	0.2852	0.9279
4	93	353	0.2462	0.9206	0.2363	0.9375
5	91	345	0.2341	0.9229	0.2397	0.9288
6	90	343	0.2227	0.9289	0.2056	0.9558
7	91	345	0.1896	0.9359	0.1843	0.9519
8	91	345	0.1816	0.9446	0.1696	0.9529
9	90	343	0.1897	0.9405	0.2008	0.9452
10	92	349	0.178	0.9419	0.2312	0.9423

Table 2
The result of training CNN model using CPU

Epoch	Time, sec/ epoch	Time, sec/step	Loss function	Accuracy	Validation loss function	Validation accuracy
1	578	2	0.4927	0.7721	0.2566	0.9058
2	590	2	0.2711	0.8881	0.183	0.9337
3	597	2	0.2048	0.9152	0.1583	0.9423
4	590	2	0.2066	0.9197	0.1347	0.9538
5	579	2	0.1636	0.9292	0.1498	0.9452
6	553	2	0.1861	0.9252	0.1527	0.9423

7	535	2	0.1751	0.932	0.1405	0.9481
8	526	2	0.14	0.9466	0.1479	0.95
9	524	2	0.1366	0.9481	0.1292	0.95
10	524	2	0.1511	0.9464	0.1512	0.9538

Table 1 and Table 2 show the main indicators by which we will compare the training of the model using the CPU and GPU, namely:

1. Epoch - the number of times the algorithm passes through the entire dataset
2. Time, sec/epoch – time in seconds spent on epoch training
3. Time, sec/step – time in seconds spent training one step
4. Loss function – neural network prediction error
5. Accuracy – a metric for evaluating the classification of the model
6. Validation function of losses – the function of losses during the passage of the algorithm through the validation dataset
7. Validation accuracy – accuracy during the passage of the algorithm through the validation dataset

When training a model, it is important that it coincides, ie finds the best option, which is why we consider two indicators that can be used to monitor changes in the model. The first is the loss function. Since the task is to minimize it, ideally to reduce it to zero, it is important that with the number of epochs it decreases.

Table 3

Changing the function of losses with the change of training epochs

Epoch	GPU		CPU	
	Train	Val	Train	Val
1	0.5762	0.2825	0.4927	0.2566
2	0.3177	0.2724	0.2711	0.183
3	0.2737	0.2852	0.2048	0.1583
4	0.2462	0.2363	0.2066	0.1347
5	0.2341	0.2397	0.1636	0.1498
6	0.2227	0.2056	0.1861	0.1527
7	0.1896	0.1843	0.1751	0.1405
8	0.1816	0.1696	0.14	0.1479
9	0.1897	0.2008	0.1366	0.1292
10	0.178	0.2312	0.1511	0.1512

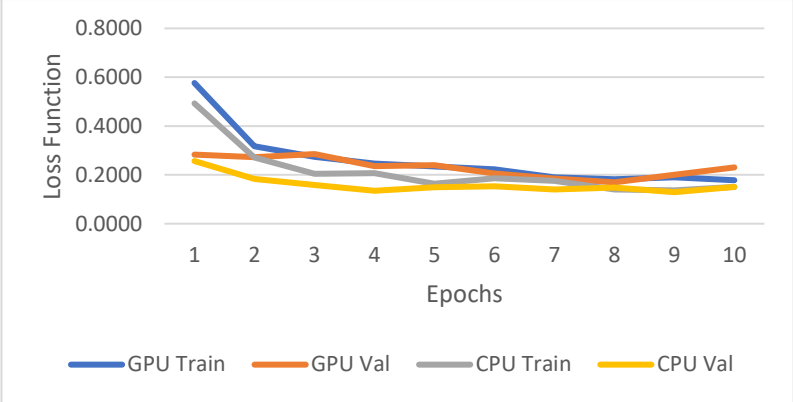

Figure 5: Change of loss function with change of training epochs

As shown from Table 3 and Figure 3, the losses in the training and validation samples decrease at the same rate in both the GPU and CPU versions. This means that according to the first indicator, the loss function, there is no difference in the models to use.

The second indicator that can be used to monitor the change in the model is its accuracy. The task is to maximize it, ideally to reduce it to 1, i.e., 100% accuracy.

Table 4

Changing accuracy with changing training epochs

Epoch	GPU		CPU	
	Train	Val	Train	Val
1	0.7623	0.9308	0.7721	0.9058
2	0.8887	0.9192	0.8881	0.9337
3	0.9093	0.9279	0.9152	0.9423
4	0.9206	0.9375	0.9197	0.9538
5	0.9229	0.9288	0.9292	0.9452
6	0.9289	0.9558	0.9252	0.9423
7	0.9359	0.9519	0.932	0.9481
8	0.9446	0.9529	0.9466	0.95
9	0.9405	0.9452	0.9481	0.95
10	0.9419	0.9423	0.9464	0.9538

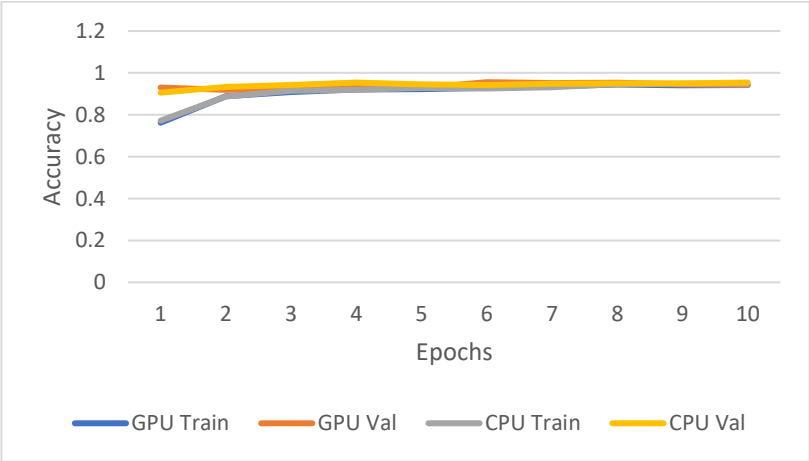

Figure 6: Accuracy change chart with changing training epochs

As shown from Table 4 and Figure 6, the accuracy of the training and validation samples increases at the same rate in both the GPU and CPU versions. This means that for the second indicator, accuracy, there is no difference in the models to use.

Given the same rate of convergence of the model on both the CPU and GPU, an indicator that becomes important is the time spent on training one era, which is equal to 263 steps.

Table 5
Time spent training one epoch/step

Epoch	GPU		CPU	
	sec/epoch	sec/step	sec/epoch	sec/step
1	91	0.34	578	2
2	91	0.35	590	2
3	93	0.35	597	2
4	93	0.35	590	2
5	91	0.35	579	2
6	90	0.34	553	2
7	91	0.35	535	2
8	91	0.35	526	2
9	90	0.34	524	2
10	92	0.35	524	2

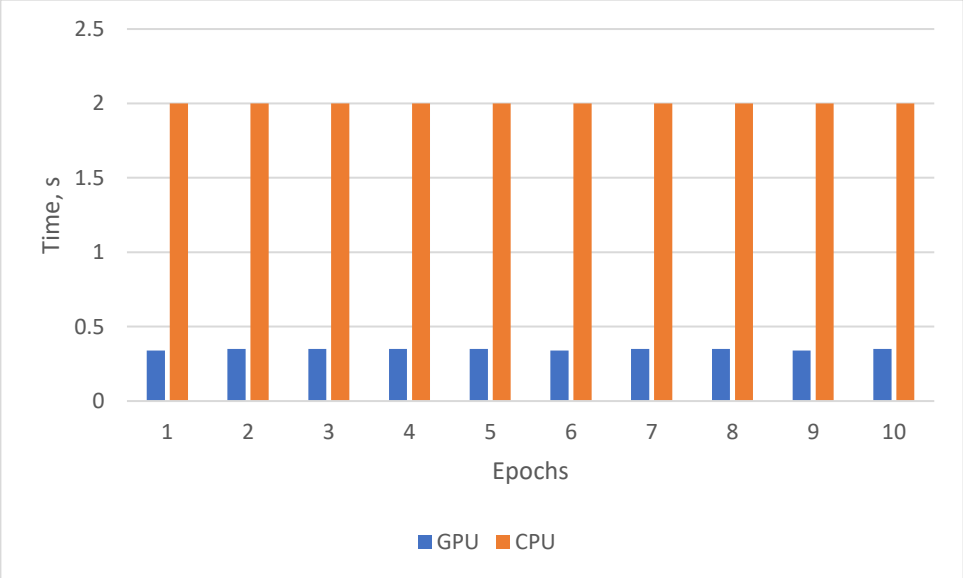

Figure 7: Change of average time for 1 step with a change of training epochs

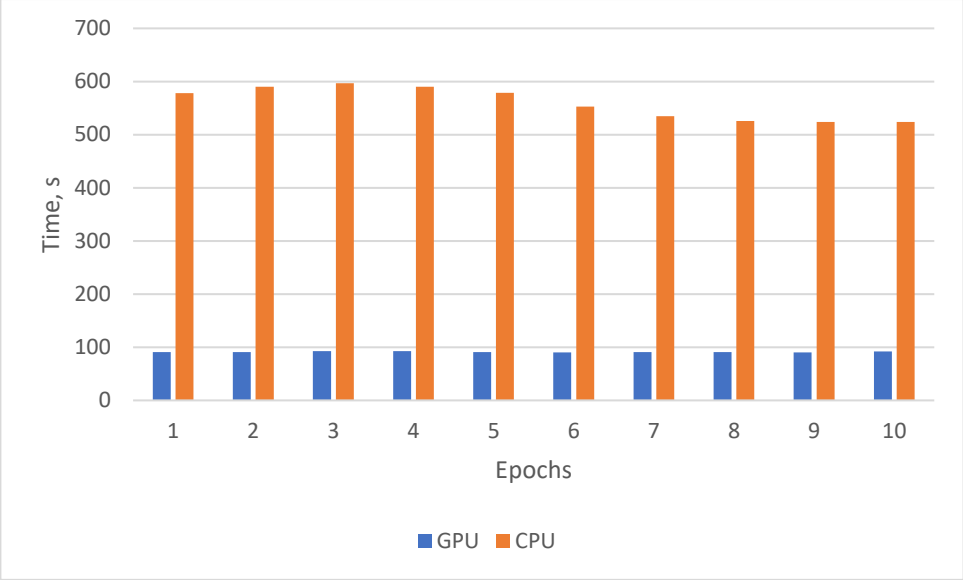

Figure 8: Change of time for 1 training epoch

From Table 5, Figure 7-8, the average time spent on training 1 step with a GPU - 0.347 sec/step; CPU – 2 sec/step, the average time spent on training 1 era with GPU - 91.3 sec/epoch, and CPU - 559.6 sec/epoch. This means that the model trains on the GPU 6.13 times faster than on the CPU.

One of the tasks is to train the model to classify chest X-rays to detect pneumonia. To prepare the model for 10 epochs using a GPU, because as shown above, it gives the same result 6.13 times faster.

Table 6
Matrix of discrepancies

	Predicted Normal	Predicted Pneumonia
Actual Normal	191	43
Actual Pneumonia	3	387

As shown in Table 6, the model correctly classified 191 patients in whom pneumonia was not detected (True Negative - TN), 397 patients in whom pneumonia was detected (True Positive - TP), incorrectly classified 43 patients with no pneumonia was detected. pneumonia (False Positive - FP), 3 patients with pneumonia (False Negative - FN).

Thus, the model showed the following metrics:

1. Accuracy = (TP + TN) / (TP + FP + FN + TN) = 0.93
2. Precision = TP / (TP + FP) = 0.90
3. Recall = TP / (TP + FN) = 0.99
4. F1 score = 2 * precision * recall / (precision + recall) = 0.94

Of the 400 patients with pneumonia, only 3 were classified as having none. Of the 234 patients without pneumonia, 43 were classified as having pneumonia. The medical field needs to keep the number of patients to a minimum, even if this means that there may be more patients of the second type, as it is better to misclassify a healthy patient than to miss a patient.

5. Discussion of results

The results presented in the previous section showed the advantage of parallel computing on the GPU overusing a serial algorithm on the CPU. Using the GPU for training took an average of 91.3 sec/epoch, and the CPU was 559.6 sec/epoch, which is 6.13 times faster for each period than the time it took the CPU to perform the same calculations. At the same time, the losses in the training and validation samples decrease at the same rate, and the accuracy in both instances increases at the same

rate regardless of which processor was used, i.e., the model was equally well trained on both CPU and GPU.

It can also be concluded that the time required to move data from the CPU to the GPU, recombination or synchronization of data obtained from parallel computations to form the result of the initial calculation is not a significant threat to the speed of the program when processing large amounts of data. Fully compensated during the counts - so much faster they run on the GPU than on the CPU.

Figure 9 shows the result of the classification of some cases taken from the dataset. Regarding the work of the model itself, it is worth noting once again its high metrics of accuracy, precision, completeness, and the weighted average value of accuracy and recall (f1 score). As already mentioned, the use of classification in medicine requires or not the highest possible indicators. We can assume that our model meets this need, showing an 18% error in the absence of pneumonia and only 0.77% when lung damage was still present on X-ray.

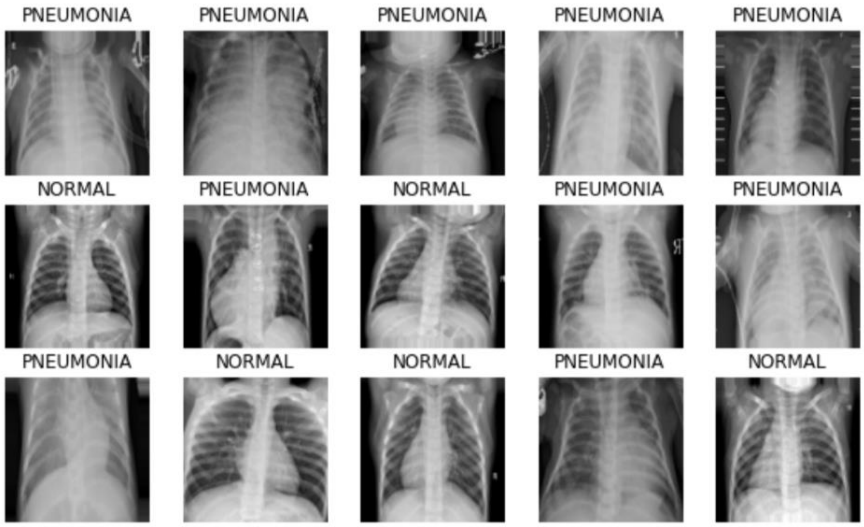

Figure 9: Classification results

However, there is room for improvement. In particular, the next step may be to reduce the error in the absence of pneumonia. By increasing the dataset or creating a more complex model, this can be done.

6. Conclusion

By performing convolutional neural network training to classify chest X-ray images on the CPU, and in combination with the GPU, it was shown that the training on the GPU is faster. In this particular case, training on the NVIDIA GeForce GTX 1650 Ti GPU was 6.13 times faster than on the Intel (R) Core (TM) i7-10750H CPU only. And while setting up a GPU requires extra time, reducing the training time of a convolutional neural network becomes very significant in real learning scenarios, when training a single neural network can take days. In this case, training on the CPU can last a week, when the use of the GPU will reduce this time to one day.

A convolutional neural network was constructed to classify chest X-rays to detect pneumonia, and an accuracy of 0.93 was obtained, with a recall of 0.99. Only three patients with pneumonia are misclassified, which is essential in the medical field because it is better to misclassify a healthy patient and pay more attention to him than to miss a patient who may get complications because of it.

References

[1]. Klette, Reinhard. Concise Computer Vision. An Introduction into Theory and Algorithms. Springer, London, 429 p., (2014), doi.org/10.1007/978-1-4471-6320-6.

[2]. Krishna, Srinivasan, Karthik, Raman, Jiecao, Chen, Michael, Bendersky, Marc, Najork. WIT: Wikipedia-based Image Text Dataset for Multimodal Multilingual Machine Learning. SIGIR Resource Track, Virtual. arXiv:2103.01913, 16 p., (2021).
[3]. Borad, Anand. Healthcare and Machine Learning: The Future with Possibilities. E Infochips : URL: https://www.einfochips.com/blog/healthcare-and-machine-learning-the-future-with-possibilities/?utm_source=EIBlog&utm_medium=BlogPostShubham&utm_campaign=related-blog.
[4]. Mochurad, Lesia, Yatskiv, Mariia. Simulation of a Human Operator's Response to Stressors under Production Conditions. Proceedings of the 3rd International Conference on Informatics & Data-Driven Medicine. Växjö, Sweden, November 19 - 21, pp. 156-169, (2020).
[5]. Moravec's paradox // Wikipedia : веб-сайт. URL: https: //en.wikipedia.org/ wiki/ Moravec%27 s_paradox.
[6]. What is image classification in deep learning? // ThinkAutomation : веб-сайт. URL: https://www.thinkautomation.com/eli5/eli5-what-is-image-classification-in-deep-learning/.
[7]. Mochurad, L., Shakhovska, K., Montenegro, S. Parallel Solving of Fredholm Integral Equations of the First Kind by Tikhonov Regularization Method Using OpenMP Technology. In: Shakhovska N., Medykovskyy M. (eds) Advances in Intelligent Systems and Computing IV. CCSIT 2019. Advances in Intelligent Systems and Computing, vol 1080. Springer, Cham, pp. 25-35,(2020) doi.org/10.1007/978-3-030-33695-0_3.
[8]. Zhou, J., Chen, W., Peng, G., Xiao, H., Wang, H., Chen, Z. Parallelizing convolutional neural network for the handwriting recognition problems with different architectures. 2017 International Conference on Progress in Informatics and Computing (PIC), pp. 71-76, (2017), doi.org/10.1109/PIC.2017.8359517.
[9]. Lee, S., Jha, D., Agrawal, A., Choudhary, A. and Liao, W. Parallel Deep Convolutional Neural Network Training by Exploiting the Overlapping of Computation and Communication. 2017 IEEE 24th International Conference on High Performance Computing (HiPC), pp. 183-192, (2017), doi.org/10.1109/HiPC.2017.00030.
[10].Christopher J. Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, George E. Dahl. Measuring the Effects of Data Parallelism on Neural Network Training. 20(112):1−49, (2019).
[11].Dean, J., Corrado, G., Monga, R., et al. Large scaledistributed deep networks. In Advances in neural informationprocessing systems, pp. 1223–1231, (2012).
[12]. Das, D., Avancha, S., Mudigere, D., Vaidynathan, K., Srid-haran, S., Kalamkar, D., Kaul, B. and Dubey, P. Distributeddeep learning using synchronous stochastic gradient descent. arXiv preprint arXiv:1602.06709, (2016).
[13]. Robbins, Herbert, Monro, Sutton. A stochastic approximation method. The Annals of Mathematical Statistics, Vol. 22, No. 3. (Sep., 1951), pp. 400-407.
[14]. Sunwoo, Lee, Ankit, Agrawal, Prasanna, Balaprakash, Alok, Choudhary, Wei-keng, Liao. Communication-Efficient Parallelization Strategy for Deep Convolutional Neural Network Training. 2018 IEEE/ACM Machine Learning in HPC Environments (MLHPC), pp. 47-56 (2018).
[15]. Xiangrui, Li and Deng, Pan and Xin, Li and Dongxiao, Zhu. Improve SGD Training via Aligning Mini-batches. arXiv preprint arXiv: 2002.09917, 10 p., (2020).
[16].Yiming, Chen & Kun, Yuan & Yingya, Zhang & Pan, Pan. Accelerating Gossip SGD with Periodic Global Averaging. Proceedings of the 38th International Conference on Machine Learning, PMLR 139:1791-1802, (2021).
[17]. Zhao, Shen-Yi & Li, Wu-Jun. Fast Asynchronous Parallel Stochastic Gradient Decent, pp. 1-15, (2015), arXiv:1508.05711v1 [stat.ML] 24 Aug 2015.
[18]. S.-Y., Zhao and W.-J., Li. Fast asynchronous parallel stochas-tic gradient descent: A lock-free approach with convergenceguarantee. In AAAI, pp. 2379–2385, (2016).
[19]. Moujahid, Hicham & Cherradi, Bouchaib & el Gannour, Oussama & Bahatti, Lhoussain & Terrada, Oumaima & Hamida, Soufiane. Convolutional Neural Network Based Classification of Patients with Pneumonia using X-ray Lung Images. Adv. Sci. Technol. Eng. Syst. J. 5(5), 167-175 (2020); doi.org/10.25046/aj050522.

[20]. Yao, Hongdou et al. Parallel Structure Deep Neural Network Using CNN and RNN with an Attention Mechanism for Breast Cancer Histology Image Classification. Cancers vol. 11, 12 1901, (2019), doi.org/10.3390/cancers11121901.
[21]. Elnashar, Alaa. To Parallelize or Not to Parallelize, Speed Up Issue. International Journal of Distributed and Parallel Systems (IJDPS) Vol. 2, № 2, March 2011, pp. 14-28, (2011).
[22]. Parallel computing and its advantage and disadvantage, 2018. https://www.geekboots.com/story/parallel-computing-and-its-advantage-and-disadvantage.
[23]. Martinovic, Goran, Zdravko Krpic and Snjezana Rimac-Drlje. Parallelization Programming Techniques: Benefits and Drawbacks. (2010).
[24]. Lee, S., Agrawal, A., Balaprakash, P., Choudhary, A., & Liao, W. K. Communication-Efficient Parallelization Strategy for Deep Convolutional Neural Network Training. In Proceedings of MLHPC 2018: Machine Learning in HPC Environments, Held in conjunction with SC 2018: The International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 47-56, (2019), doi.org/10.1109/MLHPC.2018.8638635.
[25]. Bird, Jordan & Faria, Diego & Manso, Luis & Ayrosa, Pedro & Ekárt, A. A study on CNN image classification of EEG Signals represented in 2D and 3D. Journal of Neural Engineering, 18(2), (2021), doi.org/10.1088/1741-2552/abda0c.
[26]. Sharma, Atul & Phonsa, Gurbakash. Image Classification Using CNN. SSRN Electronic Journal. Proceedings of the International Conference on Innovative Computing & Communication (ICICC) 2021, 5 p., (2021).
[27]. Palanisamy, Kamalesh & Singhania, Dipika & Yao, Angela. Rethinking CNN Models for Audio Classification. 2020, 8 p., arXiv: 2007.11154v2 [cs.CV] 13 Nov 2020.
[28]. Yann, LeCun, Leon, Bottou, Yoshua, Bengio, and Patrick, Haffner. Gradient-Based Learning Applied to Document Recognition. Proc. of the IEEE, pp. 1-46, (1998).
[29]. NIH Chest X-ray Dataset. URL: https://www.kaggle.com/nih-chest-xrays/data.
[30]. Openi, chest X-ray collection. URL: https://openi.nlm.nih.gov/.
[31]. Mochurad, L.I. Optimization of numerical solution of model problems on the basis of parallel calculations. Chapter 1: monograph. Lviv: PE “BONA Publishing House”, 208 p., (2021).