PARALLEL PROGRAMMING FOR NEURAL NETWORK: DIGIT RECOGNITION

Ruimin Zhang
University of Oregon
ruiminz@uoregon.edu

Navya Swetha Daggubati
University of Oregon
navyaswe@uoregon.edu

Abstract

This project delves into the potential of parallel programming to accelerate the training of neural networks for hand-written digit recognition. It explores various parallel programming techniques, including OpenMP and CUDA, to parallelize the computationally intensive tasks involved in training a neural network on hand-written digits. By analyzing the forwardpropagation and backpropagation algorithms and leveraging parallel implementations on different hardware architectures, this project aims to achieve significant speedup compared to traditional sequential training methods. Furthermore, it will compare the performance of these implementations in terms of training time and accuracy, analyzing their scalability with respect to processing cores and dataset size. Additionally, optimization strategies like data pre-fetching, thread scheduling, and memory optimization will be explored to further enhance performance. Finally, the project will be implemented and evaluated using the MNIST dataset, a well-known benchmark for hand-written digit recognition, providing valuable insights into the effectiveness of parallel programming for training neural networks and contributing to the development of efficient and scalable algorithms for digit recognition tasks.

1 Introduction

Handwritten digit recognition (HDR) has emerged as a fundamental problem in the field of computer vision and machine learning, with applications ranging from postal services and financial transactions to educational software and security systems. For a number of years, HCR systems have used the Hidden Markov Model(HMM) for the transcription task –[3], but a short time ago, through Deep Learning, the CNN approach has been used to overcome some limitations of HMM –[4]. In recent years, neural networks (NNs) have become the dominant approach for HDR, achieving impressive accuracy rates. Most powerful networks that focus on handwritten character recognition are based on the inverse expansion principle, which is a systematic approach for training.

However, the training process of NNs can be computationally expensive, particularly for deep neural networks with millions of parameters.

Fortunately, the advent of parallel programming techniques offers a promising solution to accelerate NN training and improve HDR performance. By leveraging the processing power of multiple cores and Graphics Processing Units (GPUs), parallel programming enables simultaneous execution of computational tasks, leading to significant speedup.

Several research efforts have explored the potential of parallel programming for HDR. In [1] the authors demonstrated the effectiveness of a backpropagation network for digit recognition, while [2] pioneered the application of convolutional neural networks (CNNs) to achieve state-of-the-art results. More recently, [3] explored the use of CUDA and OpenMP for parallelizing the training process of NNs for HDR.

The field of parallel programming continues to evolve, offering new opportunities for further enhancing the performance of NN-based HDR systems. This project aims to make significant contributions to the field of NN-based HDR by investigating and implementing advanced parallel programming techniques for NN training, such as OpenMP and CUDA. Evaluating the performance of parallel implementations and comparing them to traditional sequential methods in terms of accuracy, training time, and scalability. Developing efficient and scalable algorithms for HDR that leverage the power of parallel computing platforms.

By building upon the existing research and harnessing the power of parallel programming, this project seeks to push the boundaries of NN-based HDR and contribute to the development of next-generation systems for accurate and efficient digit recognition.

2 Motivation and Background

Handwritten digit recognition (HDR) has long been a fundamental problem in computer vision and machine learning. Its applications are vast and impactful, ranging from postal automation and banking systems to educational software and security solutions. The pursuit of ever-increasing accuracy and efficiency in HDR has driven the exploration of various techniques, with parallel programming emerging as a promising solution.

Neural networks (NNs) have established themselves as the pioneers for HDR, consistently achieving impressive accuracy rates. However, the training process of NNs, particularly deep architectures with millions of parameters, can be computationally expensive. This necessitates the exploration of efficient training methods, and parallel programming presents a compelling solution.

Pioneering research by [2] laid the groundwork for NN-based HDR with their groundbreaking work on convolutional neural networks (CNNs). This breakthrough established a foundation for further exploration and development of NN-based HDR systems. Following this trajectory, [1] investigated the effectiveness of backpropagation networks for digit recognition, demonstrating its potential for HDR applications. Recognizing the limitations of traditional sequential training methods, [3] explored the use of parallel programming techniques, specifically CUDA and OpenMP, to accelerate the training process of NNs for HDR. This opened up exciting avenues for significantly improving training speed and efficiency.

Further investigations by [4] confirmed the feasibility of parallel programming across diverse hardware architectures. Their work demonstrated the potential of tailoring parallel algorithms to specific hardware platforms, paving the way for maximizing performance on different computing systems. Additionally, [5] explored the scalability of parallel backpropagation on multiple GPUs, showcasing the promise of harnessing the power of parallel computing for large-scale HDR tasks.

These research efforts collectively provide a compelling case for the effectiveness of parallel programming in accelerating NN training and enhancing HDR performance. They highlight the potential for significant speedup in training time, paving the way for the development of faster and more efficient HDR systems.

2.1 Neural Networks

Neural networks have revolutionized the field of handwritten digit recognition by offering unparalleled accuracy, robustness, and versatility. Inspired by the intricate structure and function of the human brain, these computational models consist of interconnected nodes called neurons, arranged in layers. Each neuron receives inputs from others, applies a mathematical function to them, and produces an output that propagates further through the network. This interconnected architecture enables NNs to learn complex patterns and relationships from data, making them ideal for tasks like HDR.
High Accuracy: NNs consistently achieve state-of-the-art accuracy rates, exceeding 99% in many cases. This level of accuracy is crucial for ensuring reliable performance in real-world applications [2][3].
Robustness: NNs demonstrate remarkable robustness to noise and variations in handwriting styles. This is critical for handling real-world data, which often contains imperfections and inconsistencies [4].
Scalability: NNs can be readily scaled to handle large datasets and complex tasks. This makes them suitable for adapting to evolving handwriting styles and increasing data volumes [5].
Flexibility: NNs are not limited to recognizing digits. They can be adapted to recognize other handwritten characters and symbols, expanding their potential applications [2].

2.2 Covolutional Neural Networks

The work of [2] marked a significant milestone in the application of NNs for HDR. They introduced the use of convolutional neural networks (CNNs), a specialized type of NN designed specifically for image recognition. CNNs excel at the below mentioned characteristics making them ideal for processing handwritten digits.
Extract spatial features: Handwritten digits are characterized by specific spatial patterns like lines, curves, and loops. CNNs are uniquely designed to extract these features from the input images, enabling accurate recognition.
Reduce dimensionality: Real-world images can have a high dimensionality, making them computationally expensive to process. CNNs automatically reduce the dimensionality of the input data through pooling layers, improving computational efficiency.
Learn hierarchical features: CNNs learn features in a hierarchical manner, starting with simple features like edges and progressing to more complex features like shapes and characters. This hierarchical learning process allows the network to efficiently represent the complex patterns in handwritten digits.
Handle variations in data: Handwriting styles can vary significantly between individuals and even for the same individual over time. CNNs are robust to these variations and can learn to generalize well to new data, making them ideal for real-world applications.
Overall, CNNs represent a significant advancement in the field of HDR. Their unique architecture and ability to learn spatial features allow them to achieve high accuracy, robustness, and scalability. This makes them the dominant approach for HDR and paves the way for developing next-generation systems with even greater capabilities.

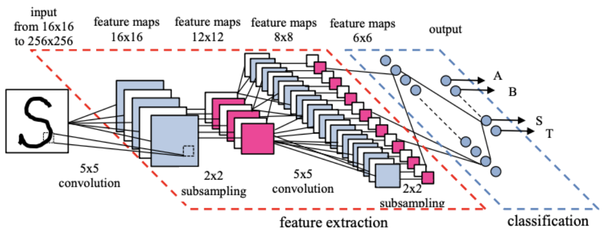

Figure 1: CNN Architecture for Digit Recognition

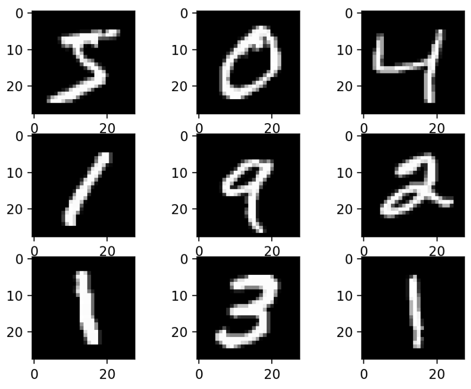

Figure 2: MNIST database images

3 Dataset: MNIST

The dataset used in this project is the Modified National Institute of Standards and Technology database(MNIST) database of handwritten digits. It comprises a training set with 60,000 examples and a test set with 10,000 examples. The dataset is commonly employed for machine learning tasks, particularly for digit recognition. The data is provided in four files, each in the IDX format:

1. `train-images-idx3-ubyte.gz`: Training set images (9912422 bytes)
2. `train-labels-idx1-ubyte.gz`: Training set labels (28881 bytes)
3. `t10k-images-idx3-ubyte.gz`: Test set images (1648877 bytes)
4. `t10k-labels-idx1-ubyte.gz`: Test set labels (4542 bytes)

The MNIST database has long been a cornerstone in the development and evaluation of image recognition models. It comprises a vast collection of 28x28 pixel grayscale images, each depicting a handwritten digit from 0 to 9. The dataset's simplicity, coupled with its accessibility, has made it an instrumental tool for researchers and practitioners alike. Introduced in the late 1990s, the MNIST dataset emerged as a pioneering resource for researchers addressing handwritten digit recognition challenges.

Its introduction facilitated the comparison of different algorithms on a standardized platform, fostering collaboration and benchmarking within the machine learning community. Over the years, MNIST has played a pivotal role in the advancement of image recognition technologies, becoming a de facto standard for evaluating the performance of various models.

The MNIST dataset comprises 60,000 training images and 10,000 testing images, providing a robust foundation for training and evaluating machine learning models. Each image is accompanied by a corresponding label indicating the handwritten digit it represents. The grayscale nature of the images simplifies preprocessing and allows researchers to focus on the intricacies of handwritten patterns. The uniform size of 28x28 pixels standardizes input dimensions for training algorithms.

4 Methodology

Our primary objective is to enhance the efficiency and speed of neural network computations by leveraging serial programming techniques. This unfolds in three key stages: the convolutional neural network (CNN) implementation, a single hidden layer neural network, and a double hidden layer neural network.

The choice of the MNIST dataset as the basis for this project stems from its historical significance and its role as a benchmark dataset in the field of image recognition. The MNIST dataset comprises a vast collection of handwritten digits, each labeled with its corresponding numeric value, making it an ideal candidate for training and evaluating the performance of the CNN.

4.1 Libraries

`"conv-network.h"` - Custom header file containing declarations and definitions for the convolutional neural network (CNN) in the project.
`"activations.h"` - Includes declarations and definitions for activation functions in the neural network.
`<Eigen/Dense>` - Handles matrices and linear algebra operations in the neural network.
`"../matrix/ops.h"` - Includes declarations and definitions for matrix operations from the "ops.h" header file in the "../matrix/" directory.

`<random>` - Provides a random number generator for initializing neural network parameters.
`<chrono>` - Includes functions working with time and date, potentially used for measuring training time in the neural network.
`<fstream>` - Allows file processing operations, potentially used for reading or writing data in the neural network.

4.2 Serial Implementation

4.2.1 Convolutional Neural Network

The CNN stage focuses on image processing and feature extraction, employing convolutional layers, pooling layers, and fully connected layers. In our serial implementation, we perform sequential computations involving convolutions and matrix multiplications. The parallelization strategy will explore both data parallelism, distributing batches of training data across processors, and model parallelism, distributing layers across processors.

Algorithm:
We implemented an algorithm for training a convolutional neural network (CNN) using the backpropagation algorithm and gradient descent for optimization.
Forward Propagation:
The forward_propfunction takes input matrices X, W1, B1, W2, and B2, representing the input, weights, and biases of the neural network layers. It computes the state (Z) and activation (A) for both the hidden layer (Layer 1) and the output layer (Layer 2) using specified activation functions like *tanh*, *ReLU*, or *leaky ReLU*. The final output is a structure (states_and_activations) containing the computed states and activations.

Backward Propagation:
The back_prop function is responsible for computing derivatives during the backpropagation phase, which is crucial for updating the weights and biases during training. It takes input matrices representing the input (X), target output

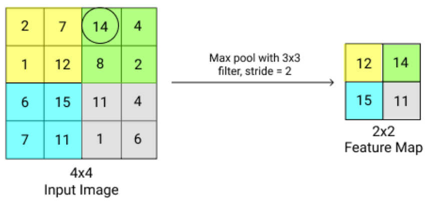

Figure 3: Pooling with filter 3x3 & stride=2

(Y), and the intermediate states and activations (Z1, A1, Z2, A2). The computed derivatives (dZ1, dW1, dB1, dZ2, dW2, dB2) are essential for adjusting the neural network parameters to minimize the error.

Gradient Descent:
The function gradient_descent orchestrates the training process by iterating through mini-batches of training data. It calculates derivatives for each mini-batch using forward and backward propagation. The accumulated derivatives are then used to update the parameters, and the process repeats for a specified number of epochs.

Convolution and Pooling Operations:
The two operations performs on input matrices, which are essential for feature extraction in CNNs. The convolutional filters (vertical_filter, horizontal_filter, neg_diag_filter, pos_diag_filter) are applied to the input images to capture different features. Pooling involves downsampling by selecting the maximum value from patches, contributing to the reduction of spatial dimensions in the data(refer to Fig.1).

4.2.2 Single Hidden Layer

In the forward propagation stage, the code computes the state (Z) and activation (A) of the neurons in each layer using the given weights (W), biases (B), and input data (X). The matrix multiplication and addition operations follow the formula $Z = W \cdot X + B \cdot \mathbf{1}$, where $\cdot$ denotes matrix multiplication and $\mathbf{1}$ is a column vector of ones. The activation function $f(Z)$ is then applied to the computed state to obtain the final activation A.

The backpropagation stage involves computing derivatives for gradient descent. The code supports different activation functions such as tanh, relu, and leaky_relu. The derivatives are calculated with respect to the states and weights of each layer, allowing for the iterative adjustment of parameters during the training process. states and weights of each layer, allowing for the iterative adjustment of parameters during the training process.

The training itself is conducted in an epoch-based manner, iterating through the entire training dataset in mini-batches. The gradients are accumulated over the batches, and the weights and biases are updated using the gradient descent algorithm. This process is repeated for a specified number of epochs, contributing to the overall learning of the neural network. The training mechanism facilitates the network's ability to adjust its parameters systematically, enabling it to learn and improve its performance on the given task over multiple iterations.

Hyper-parameters	Value
Training images	60,000
Learning rate	0.1
Hidden layer neurons	250
Training epochs	5,000
Accuracy	94.0%

Table 1: Accuracy of Single Hidden Layer NN

4.2.3 Double Hidden Layer

In the forward propagation stage, the neural network computes states (Z) and activations (A) for each hidden layer. For a given layer L, the state Z_L is calculated by performing a weighted sum of the activations from the previous layer (A_{L-1}), adding the layer's biases (B_L), and applying an activation function f such as tanh, ReLU, or Leaky ReLU. Mathematically, $Z_L = W_L \cdot A_{L-1} + B_L$, where W_L represents the layer's weights. The computed state Z_L is then passed through the activation function to obtain the final activation A_L.

In the backpropagation stage, derivatives are calculated with respect to the states and weights of each layer, facilitating the iterative adjustment of parameters during the training process. This involves updating the weights and biases based on the computed derivatives.

Hyper-parameters	Value
Training images	60,000
Learning rate	0.5
First hidden layer neurons	200
Second hidden layer neurons	50
Training epochs	5,000
Accuracy	92.7%

Table 2: Accuracy of Double Hidden Layer NN

The training process is conducted in an epoch-based manner, iterating through the entire dataset and updating the network's parameters to minimize the difference between predicted and actual outputs. This iterative learning approach enables the neural network to gradually improve its performance over multiple epochs, ultimately enhancing its ability to accurately capture complex patterns in the training data.

4.3 OpenMP Parallelization

The conversion process of images to matrices involves thirty distinct character datasets employed for the training of the neural network. To enhance computational efficiency and decrease digital processing time, we decided to use the augmentation of processor acceleration by optimizing the utilization of processor resources and developing novel parallel algorithms [17].

In the domains of image recognition and processing, particularly in pattern recognition, Convolutional Neural Networks (CNNs) play a pivotal role. Automatic pattern recognition poses a common challenge: the need to categorize an entire set of images into finite classes referred to as patterns. The input image vector is directed into the CNN layers to extract relevant features. OpenMP technology is ideal for developers who want to parallelize their programs with large parallel cycles for architectures with shared memory [16]. To reduce the computational time on CPU, one can implement feature extraction module for the CNNs using OpenMP, which can help to concurrently process multiple data with single instruction on multi-core CPU [15].

A four-layer neural network is employed for the recognition of handwritten character images, with the input image being directed into the Convolutional Neural Network (CNN) layers. These layers undergo training to extract pertinent features from the input image. The most efficient procedure, leading to maximum acceleration, involves the extraction of relevant features from the image, with a dominant role played by matrix-matrix multiplication operations. The utilization of modern Central Processing Units (CPUs) and Graphics Processing Units (GPUs) facilitates substantial productivity growth in addressing various scientific and technical challenges, such as Convolutional Neural Networks (CNNs), where optimal performance is attained by executing similar actions on a multitude of processed data units [18]. Matrix multiplication operations encompass actions of the same type on a considerable volume of processed data units, enabling the parallelization of the CNN training procedure.

The critical procedure for achieving optimal performance in a handwritten character recognition system is feature extraction. This involves extracting specific types of information from the character image. The filter multiplies its values with the overlapping values of the image while sliding over it, summing them up to output a single value for each overlap until the entire image is traversed. This enables us to compute each value in parallel for each unit of overlap on different CPU cores. Examining Figure 3 , the computation of each output matrix is independently and concurrently performed by allocating threads on different CPU cores using the OpenMP directive. To perform convolution for a single kernel, we employ a matrix-vector multiplication, where the vector represents the unrolled form of the kernel. The OpenMP is used to generate threads corresponding to the number of processor cores, facilitating the parallelization of the computation of the output matrix values.

Training a neural network involves calculating the weight coefficients that connect neurons from one layer to neurons in another layer. Training algorithms are categorized into supervised, unsupervised, and semi-supervised learning. Supervised learning algorithms involve using reference values as the output for the network. In unsupervised learning, the output values are real values calculated when

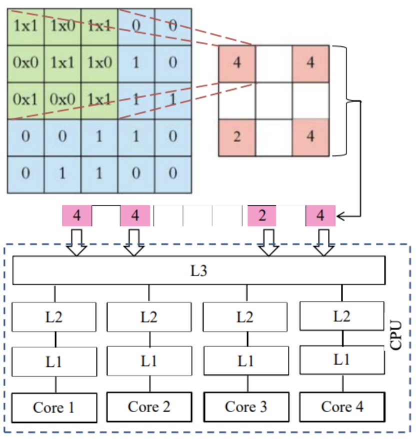

Figure 4: Scheme of allocating threads on different CPU cores

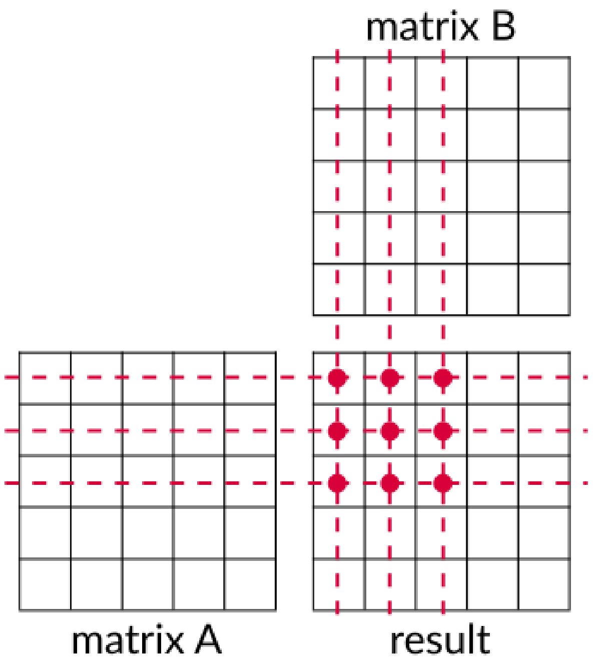

Figure 5: CUDA Matrix Multiplication

the input image is fed. A mixed approach combines supervised and self-learning, with part of the weights determined through supervised learning and the rest through self-learning [12]. However, training neural networks on large datasets is a time-consuming task [13] that requires high accuracy. Parallel processing can be beneficial in saving time during CNN training [14].

Usually in image processing and pattern recognitions contrary to the graphics area, CPU should generate raw feature data for GPU processing as much as possible to effectively utilize GPU performance [15]. If one does not have a suitable GPU, one can train on one or more CPU cores instead.

4.4 CUDA Implementation

GPUs provide a compelling alternative to CPUs for deep learning tasks due to their massive parallel processing capabilities.

It is designed to tackle the power of parallel processing on GPUs to accelerate the computations involved in training and utilizing neural networks for digit recognition [19]. Firstly, the code initializes various parameters and resources, including integer and float pointers for matrix dimensions, denominators, and variables needed for computations. It also sets up a cuBLAS handle, an essential component for utilizing cuBLAS functionalities, which is a GPU-accelerated linear algebra library.

The code implementation defines several GPU kernel functions, each catering to a specific neural network operation. These operations include matrix addition, ReLU activation function, softmax activation function, and backpropagation-related computations such as computing the derivative of the softmax function and the ReLU function.

To achieve parallelism, the code utilizes GPU threads through CUDA, a parallel computing platform and programming model developed by NVIDIA. Kernel functions are executed concurrently by multiple threads, and the resulting parallelism significantly accelerates matrix operations and neural network training procedures.

The core of our methodology lies in parallelizing the CNN training process using CUDA. We exploit the parallel architecture of the GPU to accelerate matrix operations, convolution, and backpropagation. Key CUDA concepts, such as kernels and threads, are employed to distribute computations across the GPU cores efficiently.

The `cuda_matrix_mul` function leverages cuBLAS to perform matrix multiplication efficiently. It takes three matrices, performs the multiplication of the second and third matrices, and stores the result in the first matrix.

Memory management is crucial in GPU programming, and the code handles it by using managed memory allocated through `cudaMallocManaged`. This enables seamless data transfer between the CPU and GPU.

The MNIST dataset consists of 60,000 training images and 10,000 testing images, each representing a handwritten digit between 0 and 9. These grayscale images are 28 x 28 pixels in size. Prior to training, the images were preprocessed to normalize their intensity values between 0 and 1. Additionally, data augmentation techniques like random cropping and horizontal flipping were applied to artificially increase the training data size and improve model generalizability.

For parallelization Strategies, we utilize various CUDA features and techniques to parallelize critical operations within the CNN. This includes:

- Thread Blocks and Grids: We divide the workload among multiple threads organized into blocks and grids, allowing concurrent execution on the GPU.
- Shared Memory: We use shared memory to optimize data access for frequently used data within a block, avoiding redundant global memory reads.
- Atomics: We employ atomic operations to ensure thread-safe access to shared resources, preventing data corruption during concurrent updates.

5 Acknowledgement

Both authors have a basic familiarity with C/C++, acquired through a seminar. Acknowledging the constraints of our limited knowledge, we are aware of the challenges we face in comprehending and excelling in this course. Despite these limitations, we are putting forth our best efforts in tackling this project. While the topic is engaging and enriching to learn, the programming demands have significantly impacted our personal development.

The project remains unfinished, and two months is a relatively short timeframe. Nevertheless, we are committed to persisting with the project. Additionally, we plan to dedicate time to further enhance our understanding of C/C++ during the winter break, given the substantial reliance on these languages within our institution. Our goal is to proactively acquire additional knowledge independently. Looking ahead, we anticipate taking a parallel processing course next quarter, aspiring to gain a more comprehensive understanding of parallelization.

References

[1] Devireddy, Srinivasa Kumar, and Settipalli Appa Rao. "Hand written character recognition using back propagation network." Journal of Theoretical and Applied Information Technology 5.3 (2009).

[2] LeCun, Y., et al. "Handwritten digit recognition with a back-propagation network, 1989." Neural Information Processing Systems (NIPS).

[3] Sharma, Ankit, and Dipti R. Chaudhary. "Character recognition using neural network." International Journal of Engineering Trends and Technology (IJETT) 4.4 (2013): 662-667.

[4] Jang, Honghoon, Anjin Park, and Keechul Jung. "Neural network implementation using cuda and openmp." Computing: Techniques and Applications, 2008. DICTA'08. Digital Image. IEEE, 2008.

[5] Zhang, Shunlu, Pavan Gunupudi, and Qi-Jun Zhang. "Parallel backpropagation neural network training technique using CUDA on multiple GPUs." Numerical Electromagnetic and Multiphysics Modeling and Optimization (NEMO), 2015 IEEE MTT-S International Conference on. IEEE, 2015.

[6] Gupta, Roopali, and Neeraj Shukla. "Character Recognition using Back Propagation Neural Network." International Journal of Digital Application & Contemporary research (2012).

[7] Babu, U. Ravi, Y. Venkateswarlu, and Aneel Kumar Chintha. "Handwritten digit recognition using K-nearest neighbour classifier." Computing and Communication Technologies (WCCCT), 2014 World Congress on. IEEE, 2014.

[8] S. Singh, A. Paul and M. Arun, ”Parallelization of digit recognition system using Deep Convolutional Neural Network on CUDA,” 2017 Third International Conference on Sensing, Signal Processing and Security (ICSSS), Chennai, India, 2017, pp. 379-383, doi: 10.1109/SSPS.2017.8071623.

[9] K. He, ”Handwritten Digit Recognition Based on Convolutional Neural Network,” 2023 IEEE 3rd International Conference on Electronic Technology, Communication and Information (ICETCI), Changchun, China, 2023, pp. 16-19, doi: 10.1109/ICETCI57876.2023.10176680.

[10] K. T. Islam, G. Mujtaba, R. G. Raj and H. F. Nweke, ”Handwritten digits recognition with artificial neural network,” 2017 International Conference on Engineering Technology and Technopreneurship (ICE2T), Kuala Lumpur, Malaysia, 2017, pp. 1-4, doi: 10.1109/ICE2T.2017.8215993.

[11] Patel, Ishani, Virag Jagtap and Ompriya V. Kale. “A Survey on Feature Extraction Methods for Handwritten Digits Recognition.” International Journal of Computer Applications 2014.

[12] Goodfellow I., Bengio Y., Courville A. Deep Learning (Adaptive Computation and Machine Learning series) // Cambridge: The MIT Press, 2016. – 800 p.

[13] Huqqani, A. A., Schikuta, E., Ye, S., & Chen, P., Multicore and GPU parallelization of neural networks for face recognition. In Procedia Computer Science, Vol. 18, 2013. – pp. 349–358.

[14] M. Sharif, Osman Gursoy, Parallel Computing for Artificial Neural Network Training using Java Native Socket Programming, Periodicals of Engineering and Natural Sciences, Vol. 6, No. 1, 2018. – pp. 1-10.

[15] Honghoon Jang, Anjin Park, Keechul Jung, Neural Network Implementation Using CUDA and OpenMP. Digital Image Computing: Techniques and Applications, DICTA 2008. – pp. 155-161.

[16] Fazliddinovich, Rakhimov Mekhriddin and Berdanov Ulug’bek Abdumurodovich. “Parallel processing capabilities in the process of speech recognition.” 2017 International Conference on Information Science and Communications Technologies (ICISCT) (2017): 1-3.

[17] Rashid Nasimov, Nigorakhon Nasimova, Karimov Botirjon, and Munis Abdullayev. 2023. Deep Learning Algorithm for Classifying Dilated Cardiomyopathy and Hypertrophic Cardiomyopathy in Transport Workers. In Internet of Things, Smart Spaces, and Next Generation Networks and Systems: 22nd International Conference, NEW2AN 2022, Tashkent, Uzbekistan, December 15–16, 2022, Proceedings. Springer-Verlag, Berlin, Heidelberg, 218–230.

[18] I. Khujayorov and M. Ochilov, ”Parallel Signal Processing Based-On Graphics Processing Units”, 2019 International Conference on Information Science and Communications Technologies (ICISCT), Tashkent, Uzbekistan, 2019, pp. 1-4

[19] Parallelization of digit recognition system using Deep Convolutional Neural Network on CUDA Srishti Singh1, Amrit Paul1, M. Arun1 03 May 2017

[20] Mizukami, Yoshiki, Katsumi Tadamura, Jonathan H. Warrell, Peng Li and Simon Prince. “CUDA Implementation of Deformable Pattern Recognition and its Application to MNIST Handwritten Digit Database.” 2010 20th International Conference on Pattern Recognition (2010): 2001-2004.