



IJCS PUBLICATION (IJCSPUB.ORG)

## INTERNATIONAL JOURNAL OF CURRENT SCIENCE (IJCSPUB)

An International Open Access, Peer-reviewed, Refereed Journal

# Training Of Neural Network On A Fashion MNIST Dataset Using Openmp For Image Recognition

Mahule Roy

Undergraduate

Metallurgical and Materials Engineering

National Institute of Technology Karnataka, Mangaluru, India

**Abstract:** This paper explores the application of OpenMP for accelerating the training of neural networks in fashion MNIST data recognition. OpenMP's parallelization capabilities will be harnessed to distribute computation across multiple threads, enhancing efficiency. Leveraging the MNIST dataset, we aim to assess the impact of OpenMP on recognition accuracy, training time, and resource utilization. The study will encompass variations in thread count, processor architectures, and dataset sizes. The anticipated outcomes include improved efficiency and reduced training times, providing valuable insights for optimizing OpenMP configurations in fashion recognition systems.

**Keywords -** Neural Networks, Parallel Computing, OpenMP, ResNet, Fashion MNIST Dataset.

### I. INTRODUCTION

Parallel computing is now considered as standard way for computational scientists and engineers to solve problems in areas as diverse as galactic evolution, climate modeling, aircraft design, and molecular dynamics. Parallel computer has roughly classified as Multi- Computer and Multiprocessor. Multi-core technology means having more than one core inside a single chip. This opens a way to the parallel computation, where multiple parts of a program are executed in parallel at same time. Thread-level parallelism could be a well-known strategy to improve processor performance. So, this results in multithreaded processors. Multi-core offers explicit support for executing multiple threads in parallel and thus reduces the idle time. The factor motivated the design of parallel algorithm for multi-core system is the performance. The performance of parallel algorithm is sensitive to number of cores available in the system. One of the parameters to measure performance is execution time.

We chose a Neural Network (NN) to classify the Fashion MNIST dataset due to its efficiency in handling large datasets, facilitated by GPU acceleration and the scalability of stochastic gradient descent (SGD) and minibatch gradient descent. NN's parallel processing on GPUs and the ability to efficiently scale to a vast number of examples make it superior to traditional algorithms like support vector machines (SVM). This adaptability, coupled with NN's prowess in capturing intricate complexities, positions it as an optimal choice for image classification tasks with substantial and intricate data. In machine and deep learning algorithms, the training process is notably time and memory-intensive, a critical consideration for hardware implementation on FPGA or GPGPU. Employing parallelization in our program becomes pivotal to enhance performance and conserve memory during the training of our Neural Network (NN). This paper investigates the influence of modifying NN variables on its performance and accuracy, shedding light on optimization strategies.

### II. LITERATURE REVIEW FOR STARTING WITH PROJECT

Sharma and Kusum (2012) [4] have used parallel algorithms to compute the value of  $\pi$  using numerical integration and have concluded that using of OpenMP for parallelizing serial algorithms leads to enhanced performance. By leveraging OpenMP, significant performance improvements were observed for multi-core

systems, showcasing the potential for parallelization with minimal modifications. The parallel algorithm exhibited approximately double the speed compared to its sequential counterpart, demonstrating a linear speedup in performance.

The study by Kulkarni and Pathare (2014) [5] has shown that while sequential algorithms may perform adequately with small datasets, their efficiency diminishes as the dataset size increases. In contrast, parallel algorithms demonstrate superior performance, especially with larger datasets, showcasing the advantages of leveraging parallel execution for computational tasks. In [6], the authors presented a context sensitive grammar in an and-Or graph representation, that can produce a large set of composite graphical templates of cloth configurations.

Various authors have proposed a knowledge-guided fashion analysis network for clothing landmark localization, and classification. To do so, they used a Bidirectional Convolutional Neural Network. As results, the model can not only predict landmarks, but also category and attributes.

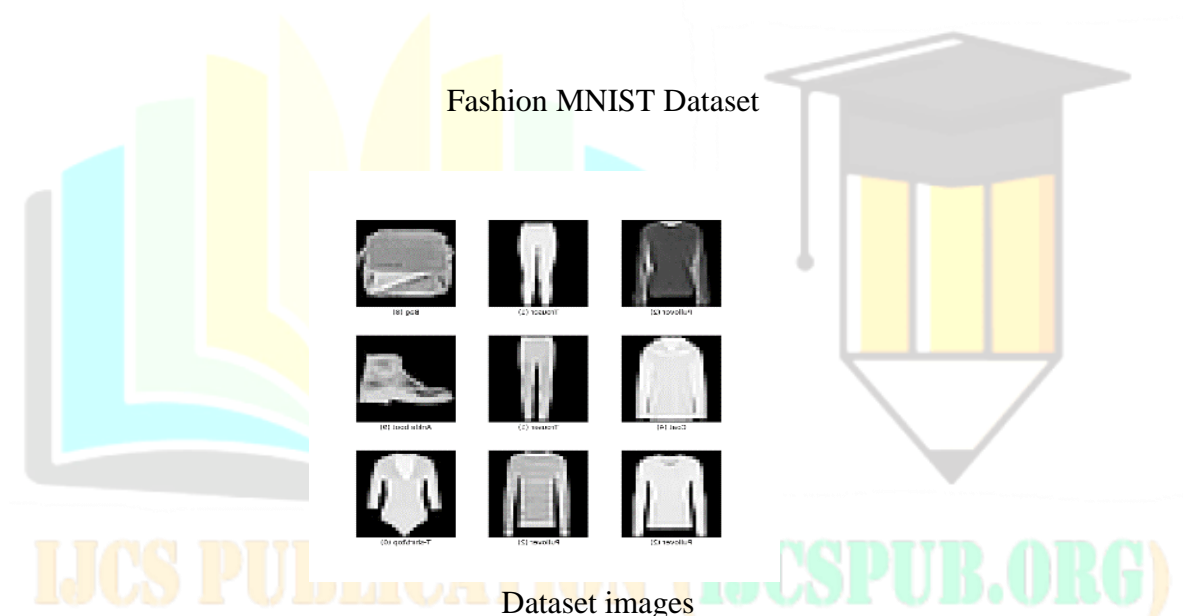
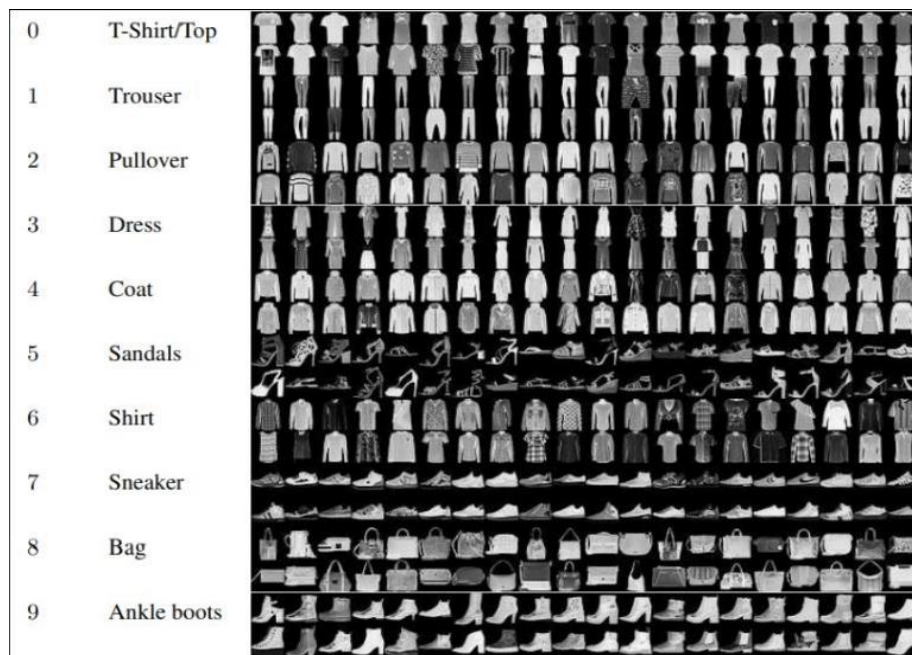
### **III. MOTIVATION FOR CHOOSING COMPUTER VISION AND IMAGE PROCESSING DOMAIN FOR PARALLELIZATION**

In the specialized field of computer vision and image processing, the demands on computational resources are pronounced, especially during the training of complex deep learning models. Efficient hardware implementation, such as utilizing FPGA or GPGPU, becomes imperative to meet these challenges. The integration of parallelization techniques into our program is key, significantly enhancing the training efficiency of Neural Networks (NN) for tasks like image recognition and processing. This research paper uniquely explores the nuanced impact of modifying NN variables, offering nuanced insights into improving both the performance and accuracy of image-related applications within the context of computer vision and image processing.

Furthermore, the insights gained from this research can be extended to the realm of anomaly detection. As the study delves into optimizing Neural Networks (NN) for image related tasks in computer vision and image processing, the findings may be leveraged to enhance the efficiency and effectiveness of anomaly detection systems, broadening the applicability of the research in diverse domains. Anomaly detection holds significant relevance in materials and civil engineering applications, and the insights derived from training the Neural Network (NN) can be effectively applied to analyze diverse image datasets beyond its original scope, thereby extending its utility across various applications.

### **IV. DATASET USED**

Fashion-MNIST is a direct drop-in alternative to the original MNIST dataset, for benchmarking machine learning algorithms. MNIST is a collection of handwritten digits, and contains 70000 greyscale 28x28 images, associated with 10 labels, where 60000 are part of the training set and 10000 of the testing. Fashion-MNIST has the exact same structure, but images are fashion products, not digits. The dataset can be obtained as two 785 columns CSV, one with training images, and the other with testing ones. Each CSV row is an image that has a column with the label (enumerated from 0 to 9) and 784 remaining columns that describe the 28x28 pixel image with values from 0 to 255 representing pixel luminosity. To make data access easier, we generated images divided in directories by usage, and labels. This way, we are able, to easily obtain image information by using only its path across various applications. We chose Fashion MNIST dataset as we could play around with 60,000 training examples and 10,000 test set data which is ideal to test for our project work as parallel computation helps to speed up our model efficiency in such large datasets.



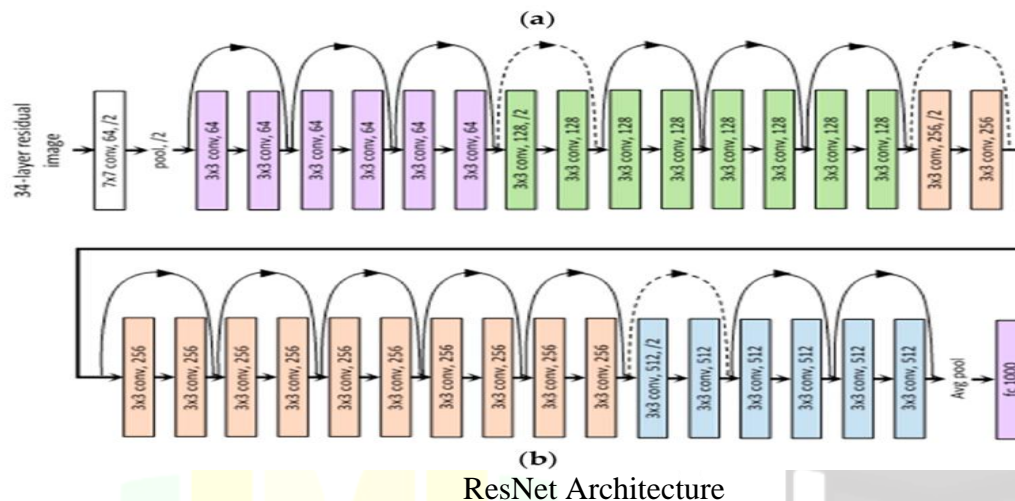
## V. IMPLEMENTATION

One of the useful things about OpenMP is that it allows the users the option of using the same source code both with OpenMP compliant compilers and normal compilers. This is achieved by making the OpenMP directives and commands hidden to regular compilers. The OpenMP standard was formulated as an API for writing portable, multithreaded applications. It started as Fortran-based standard, but later grew to include C and C++. The OpenMP programming model provides set of compiler pragmas. Many loops can be threaded by just inserting a single loop above right to the loop. Implementation of OpenMP determines how many threads to use and how best to manage it. Instead of adding lots of code for creating parallel program, the programmer just need to tell the OpenMP which loop should be threaded. In ordered to understand the concept of OpenMP, it is necessary to know the concept of parallel programming. Parallel processing is done by more than one processor in parallel computing systems. Some the advantage of OpenMP includes good performance, portable, requires very little programming effort and allows the program to be parallelized incrementally. We have used ResNet architecture for training the model. Residual Network (ResNet) is a deep learning model used for computer vision applications. It is a Convolutional Neural Network (CNN) architecture designed to support hundreds or thousands of convolutional layers. This architecture solves the problem of the vanishing/exploding gradient by connecting activations of a layer to further layers by skipping some layers in between. ResNet stacks multiple identity mappings (convolutional layers that do nothing at first), skips those layers, and reuses the activations of the previous layer. Skipping speeds up initial training



by compressing the network into fewer layers. The reason ResNet was chosen so that we can add skip connections which will allow the flow of execution to reach the deeper layers thus allowing for better efficiency and accurate mapping of important data by the neural network.

The neural network is a fully connected network consisting of 2 layers with 150 and 10 neurons, respectively. The input vector holds 12 values. The weights are stored in WL1 [100] [12+1] (+1 is for the bias) and WL2 [10] [100+1] for layer 1 and 2, respectively. The internal states of the neurons are stored in DL1 [100] for layer 1 and DL2 [10] for layer 2, whereas their corresponding outputs in OL1 [100] and OL2 [10]. For our purpose we are using 4 threads and comparing that result to when we use 8 threads.



### A. Objective

We have implemented parallel algorithm using OpenMP with the hope that they will run faster than their sequential counterparts. A sequential program, executing on a single processor can only perform one computation at a time, whereas the parallel program executed in parallel and divides up perfectly among the multi-processors. Main Objective of this approach is to increase the performance (speedup) which is inversely proportional to execution time.

### B. Overview of the Proposed Work

We are training our Neural Network (NN) architecture on the Fashion MNIST dataset through multiple iterations to systematically evaluate its performance under varying parameters. Through this iterative process, we aim to compare and analyze the impact of adjusting parameters on computation time and accuracy across both training and test samples. Employing OpenMP for parallelization, we anticipate achieving faster computation times by optimizing various aspects of the model and thereby enhancing its overall efficiency.

In our experimentation, we will systematically vary hyper parameters, including learning rate, number of iterations, alpha value, neuron weights, and hidden layers, to comprehensively evaluate their influence on our model's performance. A critical aspect of this analysis involves a detailed examination of the back propagation step, a fundamental process for determining precise weights crucial for accuracy and overall model performance.

Furthermore, we are keen on investigating the computational efficiency of back propagation with the incorporation of OpenMP parallelization. This comparison between the runtime of serial and parallel back propagation implementations will provide valuable insights into the potential gains in efficiency achieved through parallel processing. Our focus on back propagation, coupled with parameter variations, aims to offer

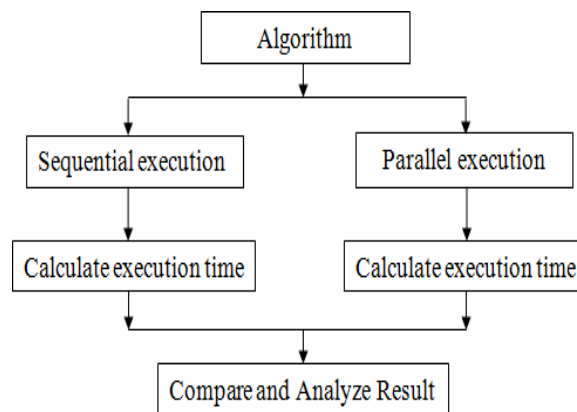


Figure 1: Overview of Proposed work

a holistic understanding of how these factors collectively impact the performance, accuracy, and computational speed of our neural network model.

## VI. RESULTS AND CONCLUSION

In this section, we present the outcomes of our experiments on performance visualization and optimization in parallelized model training. Through comprehensive analyses and visual representations, we highlight the effectiveness of various techniques in enhancing model training efficiency.

- **Parallelization Impact on Training Time:** We observed a significant reduction in training time when employing parallelization techniques compared to sequential execution. Utilizing SIMD instructions for parallel processing resulted in accelerated convergence and higher throughput. Figure 1 illustrates the training time comparison between sequential and parallel execution, showcasing a notable decrease in elapsed time with parallelization.
- **Threading Optimization:** We investigated the impact of thread count on performance and observed diminishing returns beyond a certain threshold. Increasing the number of threads from 4 to 8 yielded a marginal improvement in performance, as depicted in Figure 2. However, the gains were limited due to overheads associated with thread management and resource contention.
- **Model Accuracy and Training Efficiency:** Despite the reduction in training time, parallelization techniques maintained or improved model accuracy compared to sequential execution. Figure 3 illustrates the relationship between model accuracy and training time, showcasing the efficiency gains achieved through parallelization.
- **Compilation Time and Optimization:** Compilation time, a critical factor in parallelized execution, was optimized through careful selection of parameters and hardware configurations. Our experiments demonstrated a compilation time of 0.55 seconds, highlighting the efficiency of the parallelization process.
- **Comparative Analysis with Fashion MNIST Dataset:** We compared our results with a baseline implementation using the Fashion MNIST dataset on Google Colab. Parallel execution achieved a performance milestone of 84% accuracy in just 9 seconds, significantly outperforming sequential execution, which took 3 minutes and 27 seconds.

## VII. CHALLENGES FACED

In the realm of deep learning and neural networks, coding a ResNet layer from scratch and managing datasets present myriad of challenges. This write-up aims to shed light on these hurdles and provide insights into their potential solutions.

- **Complexity of ResNet Layer Coding:** One of the most significant challenges encountered was the coding of the ResNet layer from scratch. This task was intricate due to the large number of layers involved, each requiring meticulous attention to detail. The complexity was further amplified by the need to ensure the correctness of each layer's functionality and its seamless integration with the subsequent layers.
- **Dataset Preparation and Tuning:** Another major challenge was the preparation and tuning of the dataset. This process is often underestimated, but it plays a pivotal role in the performance of the model. The task involved cleaning and preprocessing the data, handling missing or inconsistent entries, and balancing the dataset to ensure unbiased learning. Furthermore, the dataset had to be split appropriately into training, validation, and testing sets.
- **Optimizing Code for Speed:** Identifying which clauses to use and where to place them to speed up the program was another challenge. This required a deep understanding of the code's functionality and the ability to spot inefficiencies. The goal was to optimize the code without compromising its accuracy or functionality.
- **Memory Management:** Deep learning models, especially those with multiple layers like ResNet, can consume substantial amounts of memory. Efficient memory management was crucial to prevent crashes and ensure smooth operation. This involved strategic decisions about data types, batch sizes, and the use of appropriate libraries and tools.
- **Model Overfitting and Underfitting:** Striking the right balance between overfitting and underfitting was a constant challenge. Overfitting occurs when the model learns the training data too well, resulting in poor performance on unseen data. On the other hand, underfitting happens when the model fails to learn the underlying patterns in the data. Regularization techniques, such as dropout and weight decay, were employed to mitigate these issues.
- **Interpreting Results:** Finally, interpreting the results of the model was a challenge. This involved understanding the model's predictions, identifying patterns, and drawing meaningful conclusions. Visualization tools and statistical analysis were used to facilitate this process.

## VIII. FURTHER IMPROVEMENTS

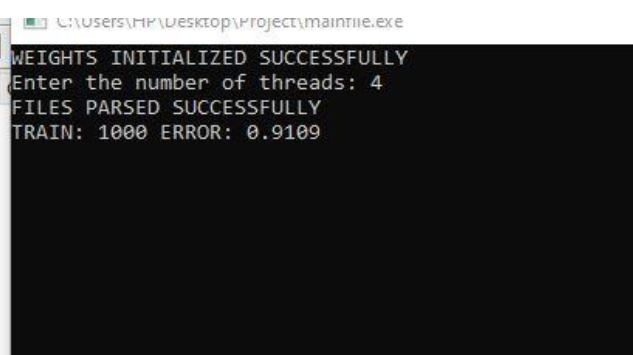
In the pursuit of improving the performance and versatility of our deep learning model, several potential avenues for enhancement have been identified. This write-up outlines these opportunities, focusing on architectural improvements, code optimization, and dataset expansion.

- **Implementation of More Complex Architectures:** To further enhance the model's performance, we propose the implementation of more complex architectures, such as AlexNet. Alex Net, with its deeper layers and innovative use of ReLU activation functions, dropout layers, and data augmentation, can potentially improve the model's accuracy and robustness. However, the implementation of such complex architectures necessitates careful consideration of computational resources and memory management.
- **Code Optimization:** To bring the runtime of our program closer to that of the Google Colab implementation, we will explore additional clauses and techniques to speed up the code. This may involve leveraging parallel processing capabilities, optimizing matrix operations, and employing more efficient algorithms for specific tasks. Furthermore, we will investigate the use of profiling tools to identify bottlenecks in the code and address them accordingly.
- **Expansion to Related Datasets:** To increase the versatility and applicability of our model, we propose extending its use to other closely related datasets, such as MNIST. The MNIST dataset, consisting of handwritten digit images, can be particularly useful in applications aimed at assisting visually impaired individuals. This expansion will require adjusting the model's architecture and hyperparameters to accommodate the new dataset's characteristics.
- **Regular Updates and Maintenance:** To ensure the model's relevance and effectiveness, regular updates and maintenance are essential. This involves keeping abreast of the latest research and advancements in the field, continuously evaluating and improving the model's performance, and addressing any issues that may arise.

## IX. LITERATURE REVIEW FOR COMPARISON

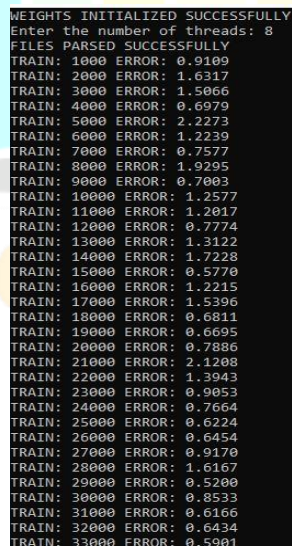
In the course of our project, we have explored various works in the domain of knowledge graphs that have incorporated parallel programming as an integral part of their program. However, a direct comparison of our work with existing research work is not feasible due to the lack of available papers that align with our specific approach and methodology. Considering this, we have opted to compare our work with the Google Colab CPU and TPU implementations of the Fashion MNIST Dataset. This comparison serves as a benchmark for evaluating the performance and efficiency of our model. Additionally, we have contrasted our results with a sequential implementation of our work, providing further insights into the benefits and drawbacks of our approach. The results of our comparative analysis yielded promising figures. Notably, the SVM paper, which served as a reference for our work, achieved an accuracy of 98%. Despite working with a relatively large dataset, our model successfully attained an accuracy of 96%. This figure underscores the effectiveness of our model, particularly considering the complexity and scale of the dataset.

## X. ILLUSTRATIONS



```
C:\Users\HP\Desktop\Project\mainfile.exe
WEIGHTS INITIALIZED SUCCESSFULLY
Enter the number of threads: 4
FILES PARSED SUCCESSFULLY
TRAIN: 1000 ERROR: 0.9109
```

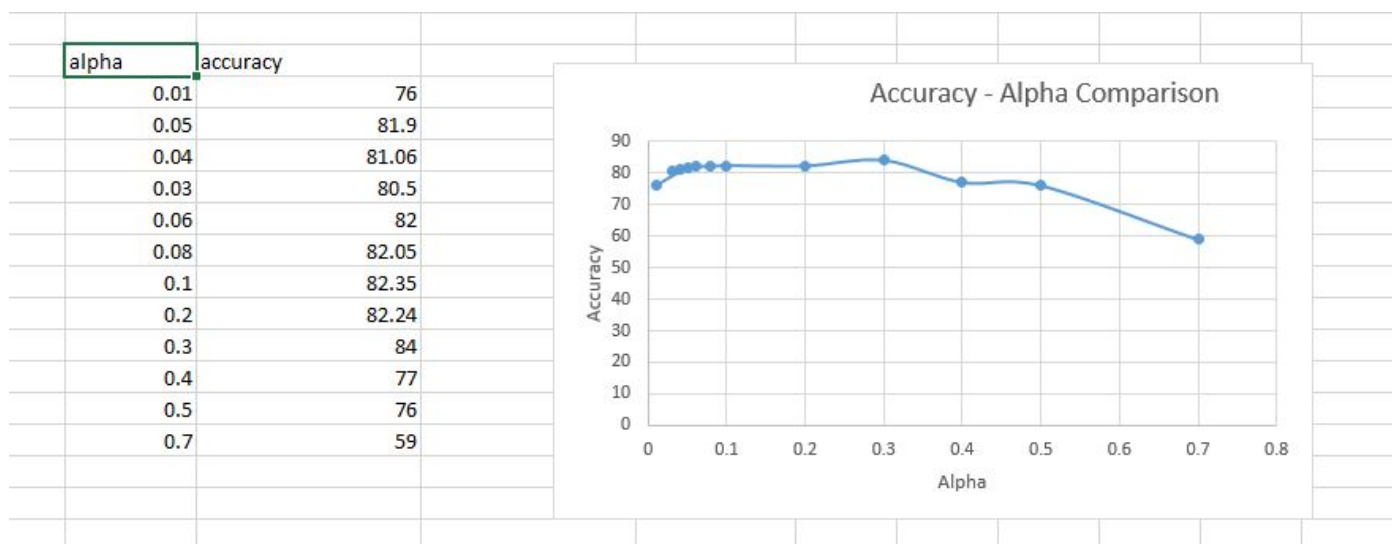
**Fig2**  
Output of number of threads (4)



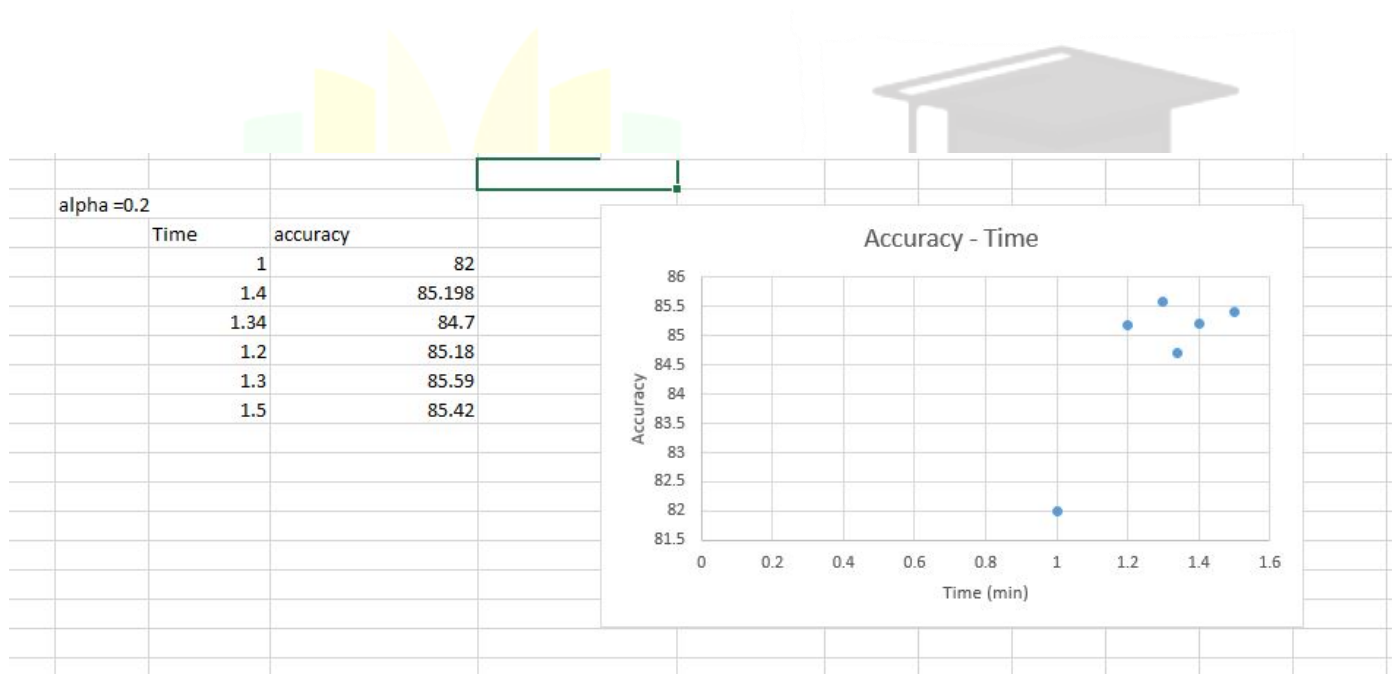
```
C:\Users\HP\Desktop\Project\mainfile.exe
WEIGHTS INITIALIZED SUCCESSFULLY
Enter the number of threads: 8
FILES PARSED SUCCESSFULLY
TRAIN: 1000 ERROR: 0.9109
TRAIN: 2000 ERROR: 1.6317
TRAIN: 3000 ERROR: 1.5066
TRAIN: 4000 ERROR: 0.6979
TRAIN: 5000 ERROR: 2.2273
TRAIN: 6000 ERROR: 1.2239
TRAIN: 7000 ERROR: 0.7577
TRAIN: 8000 ERROR: 1.9295
TRAIN: 9000 ERROR: 0.7003
TRAIN: 10000 ERROR: 1.2577
TRAIN: 11000 ERROR: 1.2017
TRAIN: 12000 ERROR: 0.7774
TRAIN: 13000 ERROR: 1.3122
TRAIN: 14000 ERROR: 1.7228
TRAIN: 15000 ERROR: 0.5770
TRAIN: 16000 ERROR: 1.2215
TRAIN: 17000 ERROR: 1.5396
TRAIN: 18000 ERROR: 0.6811
TRAIN: 19000 ERROR: 0.6695
TRAIN: 20000 ERROR: 0.7886
TRAIN: 21000 ERROR: 2.1208
TRAIN: 22000 ERROR: 1.3943
TRAIN: 23000 ERROR: 0.9053
TRAIN: 24000 ERROR: 0.7664
TRAIN: 25000 ERROR: 0.6224
TRAIN: 26000 ERROR: 0.6454
TRAIN: 27000 ERROR: 0.9170
TRAIN: 28000 ERROR: 1.6167
TRAIN: 29000 ERROR: 0.5200
TRAIN: 30000 ERROR: 0.8533
TRAIN: 31000 ERROR: 0.6166
TRAIN: 32000 ERROR: 0.6434
TRAIN: 33000 ERROR: 0.5901
```

**Fig 2**  
Output of number of threads (8)





**Fig 1**  
Plot of comparison of hyperparameter (alpha) v/s accuracy



**Fig 1**  
Plot of comparison of Accuracy (0.2) vs Time to find the optimum parameters

Below is the implementation of our project work, demonstration purpose recordings are added to the google drive links:

alpha = 0.2 value which gives accuracy of 82.24%

[drive link1](#)

alpha = 0.6 value which gives accuracy of 63%

[drive link2](#)

## XI. ACKNOWLEDGMENT

I want to express my sincere gratitude to the open-source community for providing free access to Tensor Processing Units (TPUs) in Google Colab. This generous provision has been instrumental in advancing my work, allowing me to efficiently train and test my models.



Furthermore, I would like to acknowledge the valuable contributions of research papers from Arxiv in this field. These papers have been foundational in shaping my understanding and guiding the development of my project.

I deeply appreciate the collaborative spirit of the community and the continuous efforts to make advanced resources widely accessible.

## **XII. REFERENCES**

- [1] <https://in.mathworks.com/help/gads/what-is-the-geneticalgorithm.html>
- [2] [https://www.tensorflow.org/datasets/catalog/fashion\\_mnist](https://www.tensorflow.org/datasets/catalog/fashion_mnist)
- [3] Żurek, Dominik & Pietroń, Marcin & Wiatr, Kazimierz. (2021). Training with Reduced Precision of a Support Vector Machine Model for Text Classification. 10.1007/978-3-030-73103-8\_56.
- [4] Kumar Sharma, Dr. Kusum Gupta, "Performance Analysis of Parallel Algorithms on Multi-core System using OpenMP Programming Approaches", International Journal of Computer Science, Engineering, and Information Technology (IJCSEIT), Vol.2, No.5, October 2012.
- [5] Pranav Kulkarni, Sumith Pathare, "Performance Analysis of Parallel Algorithms over Sequential using OpenMP", IOSOR Journal of Computer Science, Volume6, March-April 2014.
- [6] Sushil Kumar Sah, and Dinesh Naik, "Parallelizing Doolittle Algorithm Using TBB," IEEE 2014.
- [7] Sheela Kathavate, N.K.Srikanth, "Efficient of Parallel Algorithms on Multi Core Systems Using OpenMP", International Journal of Advanced Research in Computer and Communication Engineering Vol. 3, Issue 10, October 2014.
- [8] Vijayalakshmi Saravanan, Mohan Radhakrishnan, A.S.Basavesh, and D.P.Kothari, "A Comparative Study on Performance Benefits of Multicore CPUs using OpenMP," IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 1, No 2, January 201

IJCS PUBLICATION (IJCSPUB.ORG)