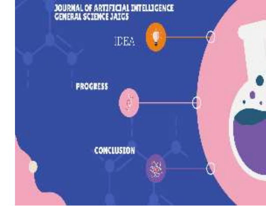
Vol.3, Issue 01, March 2024
Journal of Artificial Intelligence General Science JAIGS



Machine Learning Algorithms Scaling on Large-Scale Data Infrastructure

Harish Padmanaban

Site Reliability Engineering lead and Independent Researcher.

*Corresponding Author: Harish Padmanaban

ARTICLEINFO
Article History:
Received: 05.03.2024
Accepted: 10.03.2024
Online: 02.04.2024
Keyword: *Machine Learning, Algorithms, Scaling, Large-Scale Data Infrastructure, Computational Resources, Parallel Processing, Optimization*

ABSTRACT

Scalability is a critical aspect of deploying machine learning (ML) algorithms on large-scale data infrastructure. As datasets grow in size and complexity, organizations face challenges in efficiently processing and analyzing data to derive meaningful insights. This paper explores the strategies and techniques employed to scale ML algorithms effectively on extensive data infrastructure. From optimizing computational resources to implementing parallel processing frameworks, various approaches are examined to ensure the seamless integration of ML models with large-scale data systems.

© The Author(s) 2024. Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permitsuse, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the originalauthor(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other thirdparty material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the mate-rial. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation orexceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0

Introduction:

When queried about prevalent topics in computer science today, big data and machine learning invariably emerge as prominent contenders. Big data, characterized by high volume, velocity, and variety, necessitates specialized technologies and analytical methods for effective transformation into value. Conversely, machine learning, as defined by Tom Mitchell, involves programs enhancing task performance based on experience, task class, and performance measures.

These technologies synergize seamlessly due to the inherent challenges posed by large-scale, unstructured data in big data problems. Machine learning algorithms autonomously discern logic or uncover novel data connections, enhancing program performance with increasing experience (data). Consequently, training programs becomes a big data problem, benefiting from larger datasets.

Choosing an appropriate machine learning algorithm is paramount for application design, as different algorithms offer varied strengths and weaknesses. Selection hinges on available data and domain expertise. Given the need to process extensive datasets efficiently, consideration of big data techniques becomes imperative. However, not all algorithms lend themselves readily to parallelization and distribution on big data infrastructures, necessitating careful assessment before adoption.

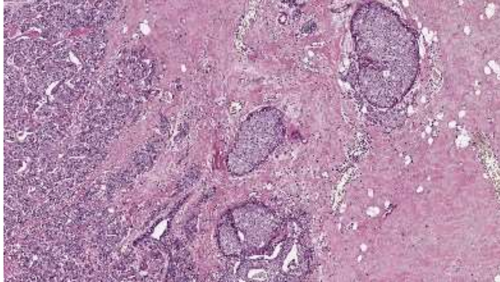
Future research would benefit from guidelines delineating algorithm suitability for implementation on big data infrastructure, aiding researchers in optimizing machine learning applications for large-scale data environments.

Practical Application in Cancer Diagnostics

In medical science, diagnostics heavily relies on various tissue images examined by specialists to detect anomalies and prescribe appropriate treatments. While human judgment is proficient in image recognition tasks, it is susceptible to errors, particularly under fatigue or time constraints. Automating this diagnostic process through a computerized system can not only reduce costs but also assist specialists in their diagnosis. However, replacing specialists entirely requires extensive validation. Instead, automated systems can serve as second opinions or highlight areas for closer examination by specialists.

Image segmentation, a computer vision technique, dissects an image into predefined components, offering better insight into its contents. Specifically, it assigns each pixel to distinct categories, such as different breast tissue types. As medical imaging techniques evolve to produce larger, more detailed images, the accuracy of diagnosis can improve. Utilizing these high-resolution images and augmenting training datasets with more diverse samples can enhance algorithm accuracy and generalization capabilities.

While these improvements promise enhanced diagnostic accuracy, they necessitate increased computational power for training and image segmentation. The latter is especially critical, as prompt results are essential in critical cases. This research aims to explore the acceleration achieved by implementing an existing sequential cancer image segmentation pipeline on the Spark big data infrastructure.



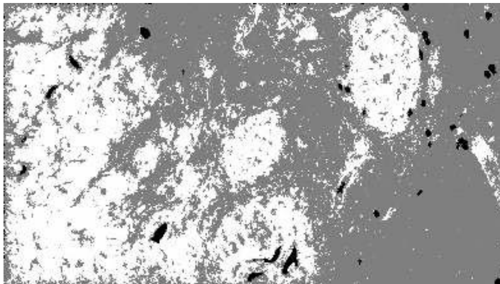
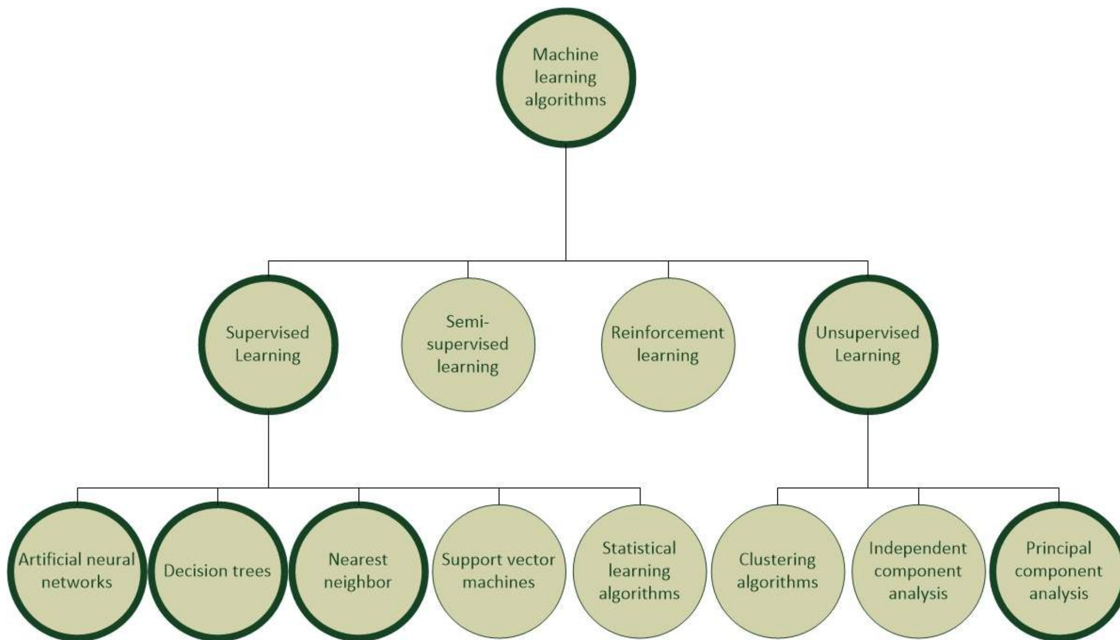


Figure 1.1 illustrates a tissue sample image on the left, accompanied by its segmented result on the right. The segmentation delineates three distinct types of tissue: healthy tissue (gray), cancerous tissue (white), and fat tissue (black).

Figure 1.2 displays a high-resolution tissue sample, showcasing magnifications of 1x, 5x, and 40x from left to right. The red rectangle denotes the specific area depicted in the adjacent picture.

Machine learning

Machine learning encompasses a wide array of algorithms, making it impractical to review and analyze them all comprehensively. Hence, this research focuses on some of the most commonly used algorithms. These algorithms are categorized based on the type of data they learn from, namely supervised learning and unsupervised learning. Although other classes such as semi-supervised learning and reinforcement learning exist, they are not within the scope of this study. Figure 2.1 illustrates the hierarchical structure of these classes, highlighting the algorithms that will be reviewed alongside others that fall outside the scope of this research.



Quantifying Scalability

Before delving into the analysis of various algorithms, it's essential to outline the methodology for quantifying scalability. Each algorithm undergoes training on a dataset comprising numerous data entries. Each entry comprises multiple attributes, representing the properties of the respective data point. Consequently, scalability can be observed along two dimensions: the number of data entries (denoted by N) and the number of attributes (denoted by M). The impact of these dimensions on execution time varies for each algorithm.

For each algorithm, the time complexity of the sequential code relative to these dimensions is analyzed. Subsequently, a scalable version of the machine learning algorithm is proposed. The new time complexity of these scalable versions is determined, accounting for the number of processors used (denoted by P). To quantify scalability, P is set equal to either N or M multiplied by a constant, as illustrated in Equation 2.1. Equations 2.1 to 2.3 are demonstrated for N, but can be similarly applied to M when parallelizing loops over this dimension.

$$P = N * c \qquad (2.1)$$

The effect of increasing P linearly with either N or M is of interest, leading to the following statements:

$O(N/P) = O(1/c) = O(1)$ (2.2)

$O(N) = O(P)$ (2.3)

Essentially, we aim to understand how much the execution time will increase if we scale the input in any dimension while also increasing the number of processors at the same rate. Achieving O(1) would imply that we could expand our input set indefinitely while maintaining a stable execution time, provided we add a sufficient number of processors.

Practical datasets typically contain more data entries than attributes, as data entries represent an upper limit to the number of classes the algorithm can distinguish. Moreover, having more attributes than classes suggests that some attributes can be combined without loss of information. Therefore, we assume that $N > M$.

Supervised Learning

The first category of machine learning algorithms we'll explore is supervised learning. Supervised learning algorithms learn tasks by utilizing labeled training data, where each data point consists of inputs and corresponding outputs. The goal is to find a function that maps inputs to outputs while maintaining the ability to generalize to new inputs. Hence, supervised learning can be viewed as a function approximation technique.

Supervised learning algorithms can be classified into two main types: decision trees and artificial neural networks (ANNs).

Decision Trees

Decision trees are hierarchical structures that facilitate decision-making processes. Each node in a decision tree represents a question about the dataset, and each branch corresponds to a possible answer. The leaves of the tree contain the final decision or classification.

Decision trees are particularly useful for complex decision-making tasks involving multiple parameters. They offer a transparent sequence of steps leading to the final decision.

One commonly used decision tree algorithm is C4.5, developed by Ross Quinlan. C4.5 builds a decision tree based on the entropy of the dataset. The algorithm iteratively selects attributes that result in the greatest reduction in entropy, ultimately forming the decision tree.

The sequential implementation of the C4.5 algorithm involves iterating over attributes and calculating the entropy gain for each split, resulting in a time complexity of $O(N * M)$.

For big data implementation, parallelization strategies can be applied to both the entropy calculation and the attribute split functions. This allows for efficient scalability in terms of both dataset size (N) and attribute count (M). By parallelizing these operations, the time complexity can be reduced to $O(M * (N/P + \log P))$, where P represents the number of processors.

Artificial Neural Networks (ANNs)

Artificial neural networks are inspired by the human nervous system and consist of interconnected nodes or neurons. These networks can perform complex functions by combining individual neurons.

Neural networks can vary in size and complexity, with input and output layers, as well as hidden layers in between. The training of a neural network involves repeatedly feeding training data to the network and adjusting weights based on the observed errors.

Two common training algorithms for neural networks are online learning and batch learning, both of which have a time complexity of $O(N * M^2)$ per training iteration.

For big data implementation, parallelization strategies are employed for both network size and dataset size. Parallelization of the apply and backpropagation algorithms can be effectively utilized on shared memory architectures. By parallelizing operations based on data entry count (N), communication costs can be minimized, leading to efficient scalability.

Overall, both decision trees and artificial neural networks exhibit scalability potential in handling large datasets and complex tasks, making them valuable tools in supervised learning.

K-Nearest Neighbor

Unlike previous algorithms, the K-nearest neighbor (KNN) algorithm does not involve a training phase. Instead, it directly determines its output from the training data by identifying the "closest" points to the point to classify.

To determine proximity, a distance metric between data points is required, such as Euclidean distance or Manhattan distance. Additionally, KNN can utilize multiple nearest points in a voting process for classification tasks or a weighted vote for regression tasks, where each attribute's importance is considered.

Algorithm 2.6 outlines the pseudo-code for a 1-nearest neighbor algorithm. The time complexity of this algorithm is O(N * M), assuming unordered data, where N is the number of data points and M is the number of attributes.

For big data implementation, parallelization can be applied to the loop over the dataset, as each node can independently calculate the K closest points. Once computed, the results are combined by a control function to determine the overall K closest points. While communication costs are relatively high for single classifications due to transmitting the entire dataset, for multiple classifications, the dataset can be retained on the nodes, reducing communication costs to the target point and resulting K distances and classes per node.

By fully parallelizing the loop over the dataset and comparing the results for minimum values, the complexity can be reduced to O((N/P + log P) * M). This approach ensures efficient scalability of the KNN algorithm on big data infrastructures.

Practical scalabilty of machine learning algorithms

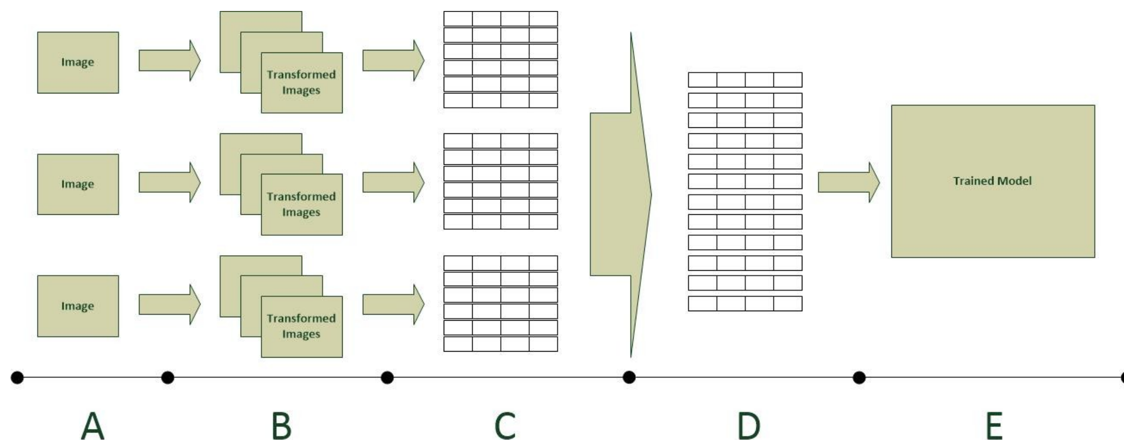
Now that we have established a theoretical measure of scalability for several machine learning algorithms, it is imperative to validate this scalability in practical scenarios. To achieve this, we will utilize an existing cancer diagnostics pipeline, which employs an artificial neural network for supervised learning. Our objective is to scale up the entire pipeline, encompassing both machine learning and non-machine learning components. This chapter initiates with an in-depth analysis of the pipeline, followed by the creation of a design in the subsequent chapter. The design will undergo testing and discussion in the final chapters.

Image Processing Pipeline

The pipeline is designed for image segmentation on breast tissue images [7], a process whereby features within an image are identified and labeled. Each pixel in the image is assigned a label indicating the type of tissue it represents. Although the pipeline can accommodate any number of labels, we will focus on three: fat tissue, stroma (healthy tissue), and carcinogen (cancerous tissue). While the pipeline as a whole performs segmentation, the labeling of individual pixels involves a classification process. To facilitate accurate classification, the pipeline utilizes a collection of images paired with annotation files specifying the class for each pixel.

Written in Matlab, the pipeline comprises three major components: (1) image loading and preprocessing, (2) machine learning algorithm training, and (3) image segmentation using the trained algorithm. Notably, the pipeline integrates both the training and application phases, whereas in real-world scenarios, training and segmentation are often conducted separately. Therefore, we will bifurcate the pipeline into two phases: training (comprising parts 1

and 2) and application (comprising parts 1 and 3). By evaluating the performance of each phase separately, we can establish a more realistic benchmark. In this setup, the application phase cannot reuse loaded and preprocessed images from the training phase, mirroring typical real-world scenarios. We will further dissect these phases into smaller steps, identifying potential performance bottlenecks. Critical steps will be examined in detail to explore options for optimal scalability on multi-node systems.

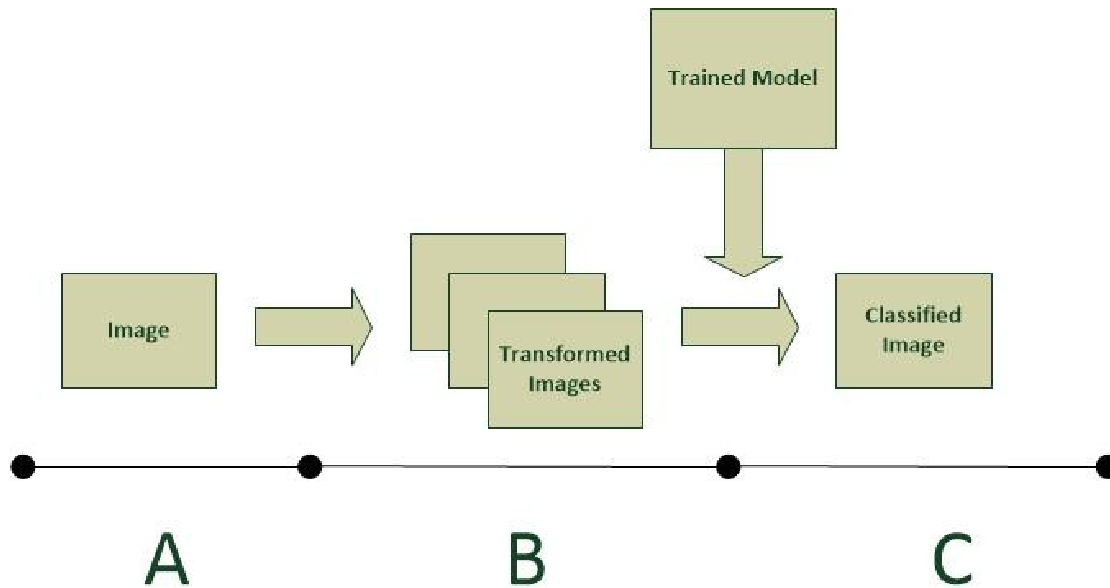


Training phase pipeline: A: Loading and decoding of images from storage. B: Applying multiple distance transformations per image. C: Arranging resulting data per pixel. D: Combining data in one set, remove non-annotated data. E: Training neural network, the resulting network parameters are stored for use in the application phase.

Pipeline Phases

Figure 3.1 illustrates the stages of the training phase pipeline. It commences with the loading and preprocessing of annotated training images. During preprocessing, a Distance Transformation on Curved Space (DTOCS) is applied using predetermined optimal alpha values. Each image undergoes DTOCS with six alpha values, resulting in six transformed images per original image. These transformed images generate an array of six input values and one output value for each pixel, after which normalization is applied. Pixels lacking annotations are excluded from the dataset. The processed dataset serves as input for training an artificial neural network, which learns to classify the images. The best-performing network produced during training is retained.

Figure 3.2 depicts the pipeline for the application phase. This phase is designed for single-image processing, focusing on scalability for large image sizes. The preprocessing step mirrors that of the training phase, excluding annotation processing. Once preprocessed, the entire image is classified using the trained neural network from the training phase. The resulting classifications for individual pixels are aggregated and stored as a complete image.



Application phase pipeline:A: Loading and decoding of image from storage. B: Applying multiple distance transformations on the image.C: Using the trained model from the training phase to classify each pixel and store the segmented image.

Performance Analysis

Profiling the pipeline and evaluating its performance occurs after the Matlab code is converted to Java. This transition is necessary because Matlab is optimized for vector and matrix operations, whereas Java is more efficient for sequential code with conditional statements. Failing to profile the Java code directly may lead to misidentified bottlenecks, as the optimizations needed for the target platform might differ.

The program runs on the cluster designated for the optimized implementations. Table 5.1 provides detailed specifications of the target platform and its constituent machines. As the algorithm is entirely sequential, only a single thread is utilized. Due to the non-deterministic nature of neural network training, a single run's results may not accurately reflect expected execution times. Therefore, the average of five runs is considered. All tests vary image count and size. Image count tests employ 1.17-megapixel images, while image size tests use ten images per trial. Each variation in count or size triggers a full pipeline run, as larger input sets may affect runtimes due to iteration variability.

Figure 3.3 illustrates execution times of phases with varying numbers of small-sized images. The training phase exhibits a linear trend, albeit with some non-linearity due to fluctuating training iteration counts. Figure 3.3 also demonstrates execution time variance with image size, displaying a clear linear increase. However, the benchmark's maximum image size is constrained by default Java decoders. Extrapolating, segmentation of Gigapixel-size images would require over 100 hours for a modest training set of 100 images. Hence, enhancing execution speed and scalability is imperative for practical application.

Figure 3.4 presents execution times for the application phase, averaged per image. Since image count does not impact per-image execution time, only results for different image sizes are depicted. Once again, a clear linear relationship emerges, with a slope indicating a five-second per megapixel dependency. Projecting to Gigapixel-sized images, segmentation of a single image would exceed 80 minutes—an unacceptable delay for critical patient cases.

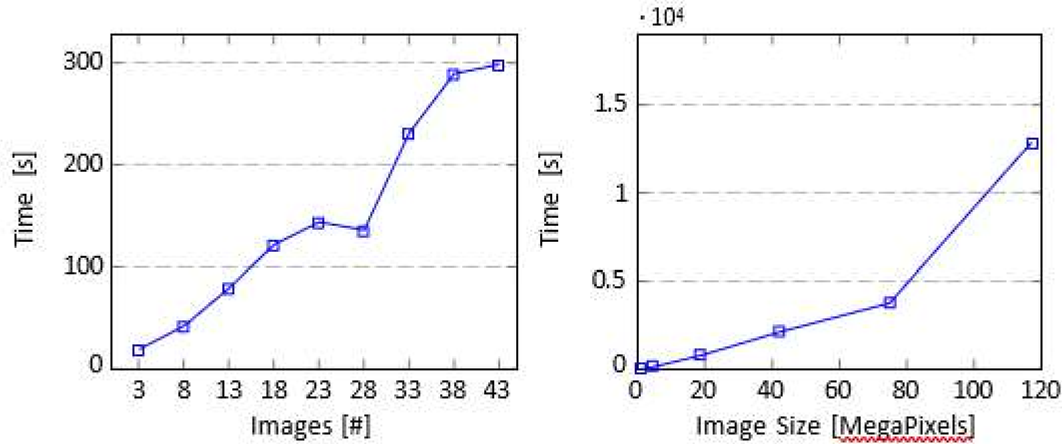


Figure 3.3: Execution times of the pipelines training phase for different image counts (left) and different image sizes(right)

Scalable Pipeline Design

With the insights gained from profiling and analyzing the pipeline steps, we can now formulate a design for a scalable version of the pipeline. Given that both the training and application phases exhibited significant time consumption in the preprocessing and neural network training/application steps, these areas will be the primary focus of our attention in the design phase. Table 3.1 provides an overview of the average contribution of different steps to the total execution time for both phases.

Apache Spark

Apache Spark, henceforth referred to as Spark, is a data processing framework developed in 2010 by the University of California, Berkeley. Designed to efficiently handle big data applications, Spark was conceived as an enhancement over the popular map-reduce model.

Map-reduce facilitated parallel operations on large data sets using clusters of commodity machines, leveraging fault tolerance mechanisms. It aimed to simplify computation on large data sets without necessitating a deep understanding of underlying distributed systems. A map-reduce program consists of pairs of map and reduce functions, with intermediate results stored on disk between each pair. Utilizing the Hadoop Distributed Filesystem (HDFS) for data storage, map-reduce enabled the creation of massively parallel programs.

However, the map-reduce model has drawbacks. Storing intermediate results on disk is slow, and the fixed map-reduce pattern limits flexibility. Spark addresses these issues by storing intermediate results in memory whenever possible, significantly enhancing speed. The in-memory data structure used by Spark across nodes is called a Resilient Distributed Dataset (RDD). RDDs can be created from files in HDFS or by distributing existing List data structures in code. Data distribution occurs via partitioning, where the dataset is divided into smaller chunks distributed evenly across worker nodes.

Spark supports various operations on RDDs, including map and reduce operations, without requiring a strict order. Spark programs run on a driver, serving as the control function. RDDs can be created and manipulated from the driver. Spark classifies operations on RDDs into transformations and actions. Transformations operate on a single node and modify partition content, while actions combine partition results and return data to the driver. Transformations are lazily executed, triggered only when an action is performed on the transformation result.

To implement the cancer diagnostics pipeline steps in Spark, we must consider its limitations and utilize the tools provided by Spark effectively. These limitations are discussed further below.

Limitations:

1. Limited worker node communication: Direct communication between worker nodes is not possible in Spark. Communication can only occur between the driver and worker nodes and vice versa.

2. Inability to address specific nodes: Spark programs are not designed to access specific nodes. Once an RDD is created, additional data needed for transformations can only be sent to partitions, functioning as broadcast variables. This approach ensures that all nodes receive any communicated variable, which may lead to communication overhead if the variable is only used by a single node.

Tools:

1. RDD (Resilient Distributed Dataset): RDDs can be created from List objects on the driver and are automatically distributed over worker nodes.

2. Accumulators: The driver can maintain an accumulator variable that can be updated from nodes during transformations and actions. The merge operation for accumulators can be customized. Care should be taken when using accumulators, as failures may cause transformations or functions to contribute multiple times to the same accumulator.

3. Broadcast variables: Broadcast variables on the driver can be used to send additional data to worker nodes. Each worker node receives an identical copy of the variable. Changes made to the variable on the driver after sending will not be reflected in the worker nodes.

Preprocessing Pipeline Optimization

The optimization of the preprocessing step in the pipeline will primarily target the DTOCS algorithm, identified as the most time-consuming aspect of the training phase.

Given the data-dependent nature of the DTOCS algorithm's inner loop, parallelization presents some challenges. To align with the focus on scalability, a coarse-grain parallelization approach is proposed. This approach, while simpler to implement and likely to incur less overhead due to larger tasks, will scale effectively only when a sufficient number of images are provided to keep each node occupied. However, it does not directly accelerate the individual DTOCS algorithm and is therefore bounded by the number of images provided.

The coarse-grain parallelization strategy involves distributing the DTOCS applications across multiple cores or nodes. One straightforward method is to distribute all necessary data for the DTOCS applications (source image and alpha value) across available nodes and execute them using multiple threads per node. The loading of annotations can follow a similar distribution approach. Subsequently, each node calculates the minimum and maximum values of the result sets required for normalization. While normalization can also be performed on the nodes, synchronization of the minimum and maximum values is necessary before this step.

For efficient data filtering, all six results per pixel must first be collected and combined with their respective annotations. To minimize data transmission, careful consideration should be given to the distribution of DTOCS tasks. Keeping DTOCS applications on the same image within a single node, along with the inclusion of annotation loading tasks, minimizes the need for post-processing data transmission. Additionally, this approach eliminates the need for nodes to load and decode new images for each DTOCS application, as the data can be shared among threads.

Implementation:

1. Coarse-grain parallelization of preprocessing involves initially listing all available images and transmitting only their locations to worker nodes to mitigate networking bottlenecks.

2. Images stored in HDFS can be directly loaded into RDDs by worker nodes without communication through the driver.

3. The application of the six transformations can be executed within a map function. Each map function encompasses all functionality from decoding to filtering. To parallelize individual transformations on a single image, they can be distributed. However, to ensure transformations of the same image reside on the same node and minimize data transmission, ordinary Java threads running on a single node in parallel are preferred. As a result, Spark tasks can utilize up to six threads each for improved efficiency.

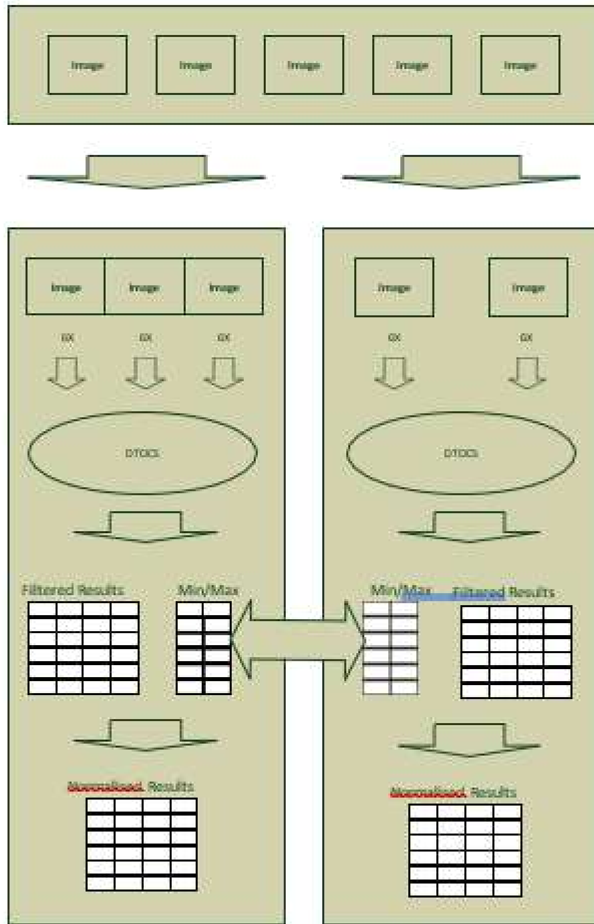


Figure 4.1: Coarse-Grain Parallelization Design for Preprocessing Steps

In this design, the images earmarked for processing are distributed across available nodes, where each node concurrently processes as many DTOCS transformations as feasible. Following the transformations, minimum and maximum values are promptly computed to facilitate early-stage data filtering. These computed minimum and maximum values need to be disseminated across the nodes, enabling each node to normalize its filtered data.

Although not depicted in this diagram, the loading of annotations also occurs on the same node where image transformation takes place, as annotations are integral to the filtering process.

To mitigate excessive network traffic associated with collecting all data from the map function, local min/max calculations are initially performed within each partition. Subsequently, these calculated values are retrieved from each partition, combined, and processed before being distributed back to each partition for normalization. Retrieval is achieved using accumulators, while broadcasting is employed to transmit the max/min results back. As the min/max functions are associative, concerns regarding the order of accumulator updates are mitigated. Furthermore, the risk of a value being added twice to the accumulator due to node failure is inconsequential, as it does not impact the results of the max/min functions. Following the transmission of results, data is filtered to retain only annotated entries. Subsequently, normalization is applied within a final map function.

Given that the resultant data will be utilized by the training algorithm, it can conveniently remain on the nodes for subsequent processing.

Results

To validate the efficiency of our implementation, a series of experiments were conducted using the same input size variations as those used in the profiling outlined in Section 3.2. This ensures that the results can be compared directly on identical input sets.

Experimental Setup

The experiments were conducted on an industrial-grade scalable big data cluster built on the IBM Power architecture. All results were obtained from the same Power7 cluster, the details of which are provided in Table 5.1. The cluster operated on Hadoop 2.5.1, with Spark applications executed on the management node using YARN in client mode. YARN serves as the job scheduler in Hadoop 2.0 and above. By connecting to the YARN scheduler, Spark leverages the pre-configured Hadoop cluster settings instead of relying on its own configuration. The configuration options for the applications are detailed in Table 5.2, with default values used for all other configuration parameters.

Spark operates on the concept of executors and containers. Executors represent the resources over which task partitions are distributed. They run on individual nodes and can consume resources as specified in the configuration. While the number of executors can surpass the number of nodes, each executor can only run on one node and cannot be split across multiple nodes. Balancing the number of executors according to the cluster configuration is crucial. Containers, on the other hand, reside within executors, with each executor housing one container. These containers contain the Java Virtual Machine where transformations and actions are executed, as well as the contents of RDD partitions assigned to the executor.

Despite having the combined resources of four worker nodes theoretically allowing for 256 threads to run in parallel using 512GB of memory, practical considerations must be taken into account. The underlying software layers beneath the executors also require resources. Consequently, we opted to use four executors, one per node. Each node runs the executor atop Hadoop and the operating system, which necessitates some resources. By assigning 62

threads per executor, each node reserves 2 threads for these tasks. Regarding memory allocation, each container requires heap space, typically 7% of the allocated executor memory, along with additional space for headers. This memory is separate from the "executor memory" configuration option (see Table 5.2). Considering these factors, we allocated 100GB for memory usage, leaving 107GB for container heap space and headers, with 21GB reserved for other processes to remain operational.

Nodes	4 workers + 1 management
Processor architecture	Power7 64 bit
Sockets per node	2
Cores per socket	8
Threads per core	4
Processor speed	3.6 GHz
Memory per node	128GB
Network Bandwidth	10Gb/s

Table 5.1: Cluster specifications, all 5 nodes have identical hardware

Master	Yarn client
Number of executors	4
Executor cores	62
Executor memory	100GB
Driver memory	80GB
Spark driver maxResultSize	8GB

Table 5.2: Spark configuration options, other options were left default

To maximize the practical utilization, we allocated 248 threads and 400 GB of memory on the worker nodes.

To mitigate statistical variance, each measurement was averaged over 5 runs. This approach helps compensate for the variability in the number of training iterations and provides more reliable results.

Preprocessing

In Figure 5.1, the left graph illustrates the execution times for the preprocessing steps with varying numbers of input images. Coarse-grain parallelization can only utilize as many processors as there are input images, which ideally would result in a constant execution time until a saturation point, typically beyond 40 input images1. However, in practice, we observe a departure from this expectation, with execution times starting to increase notably beyond 28 images.

This deviation could stem from several factors. The scheduler may encounter difficulties in evenly distributing tasks among processors, leading to uneven workload distribution. Additionally, the communication overhead associated

with accumulating and broadcasting the minimum and maximum values may increase significantly as the number of images grows. Moreover, maximizing thread utilization on a processor might result in slower execution per thread due to resource sharing within the processor.

Another intriguing observation is the higher execution time observed for the lowest number of images. This discrepancy may hint at the presence of caching mechanisms. Given that the experiments were conducted sequentially from smallest to largest image sets, it's plausible that Spark employs some form of lazy initialization, which was not fully captured during the initialization phase of measurement.

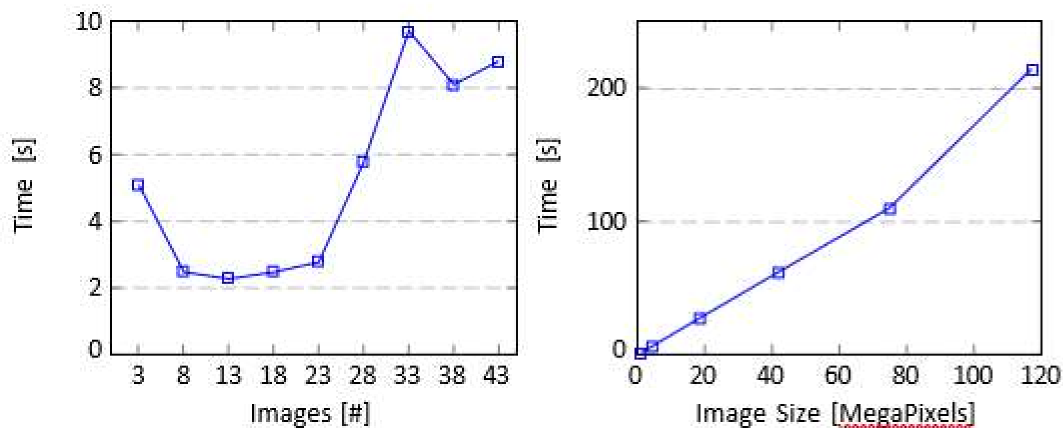


Figure 5.1: Execution times of the DTOCS step in the training phase, left shows the results for different number of images (with image size of 1.17 Megapixel), and right the results for different image sizes (for 10 images)

The results for the experiments on different image sizes are shown in Figure 5.1. These experiments could all utilize 60 threads, so we expect to still see a linearly increasing line, only with reduced execution time. This is exactly what the results show.

Training

Figure 5.2 illustrates the execution times for the delta accumulation and validation functions during the training step. The results exhibit variability depending on the number of training iterations required for training the neural network. To gain deeper insights into scalability, Figure 5.3 presents the same execution times normalized by the number of iterations.

These findings reveal that for smaller datasets2, the execution times remain comparable, with the exception of the smallest dataset, which aligns with observations from the preprocessing step. However, as the dataset size increases, particularly for the largest images, there is a noticeable uptick in execution time per iteration. To delve deeper into this phenomenon, we can leverage Spark's WebGUI, which provides visualizations of logged data for all Spark tasks executed on the cluster, as depicted in Figure 5.4.

Upon examining the breakdown of task execution times during a training iteration, we identify a significant overhead in scheduler time and deserialization time. This overhead contributes to similar execution times for smaller datasets, concealing the actual increase in execution time until it begins to overshadow the Spark-induced overhead.

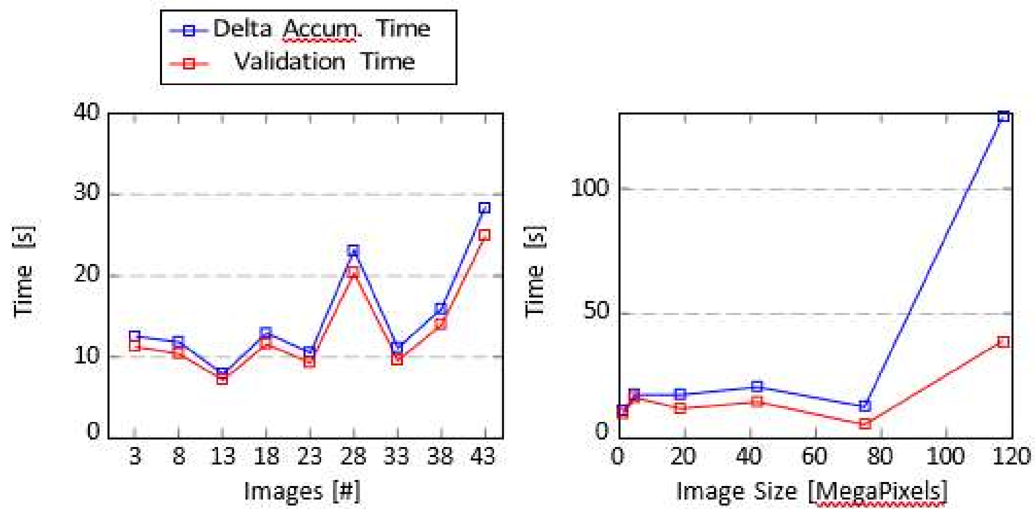


Figure 5.2: Execution times of the delta accumulation and training steps in the training phase, left shows the results for different image counts and right the results for different image sizes

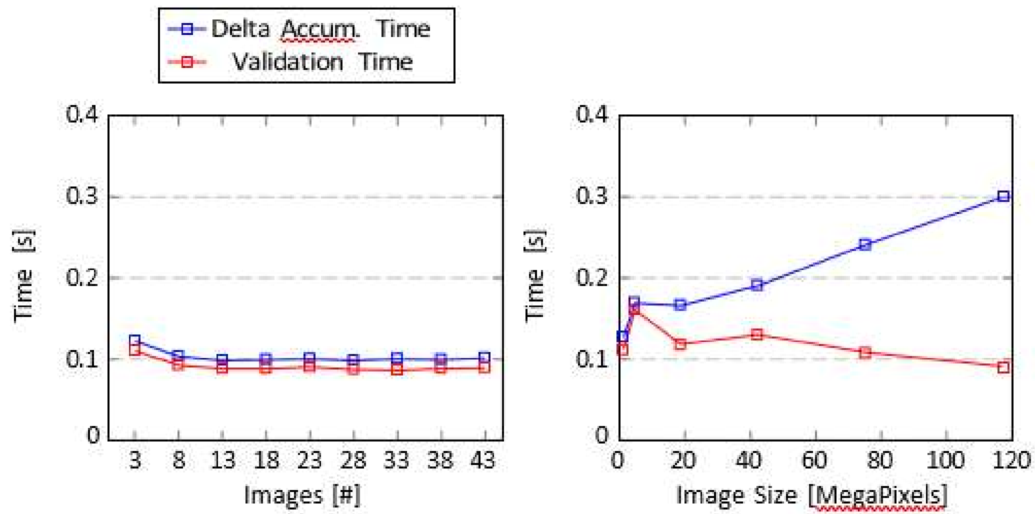


Figure 5.3: Average execution times per iteration of the delta accumulation and validation steps in the training phase, left shows the results for different image counts and right the results for different image sizes

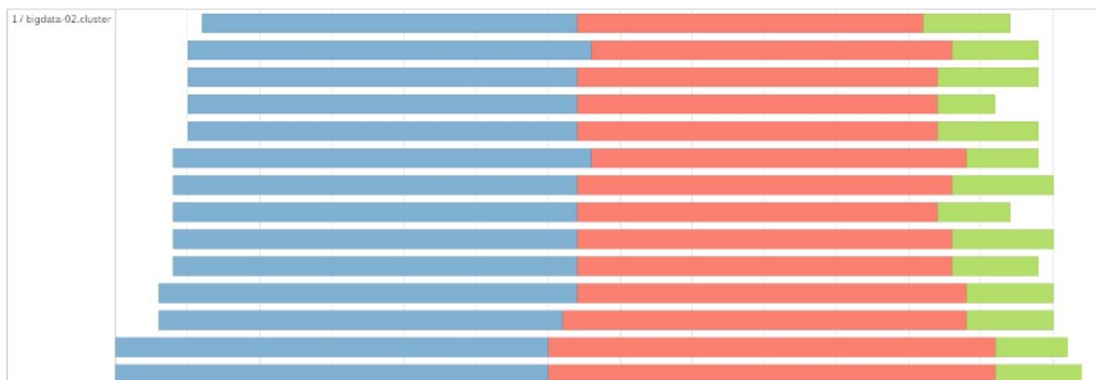


Figure 5.4 displays a screenshot of the Spark WebGUI presenting a breakdown of task execution time for a reduce task within an iteration of the training loop. The blue bar represents scheduler delay, the red bar denotes deserialization time, and the green bar illustrates the actual task body execution time.

Scalability: The preprocessing step was constrained to a number of partitions equal to the number of images, thus reflecting the growth of resources in line with the input data size. This allowed us to assess the algorithm's scalability effectively. However, for the training step, all cluster resources were utilized for calculations. Although this yielded a consistent execution time per iteration, it was primarily due to the dominance of overhead rather than the input/resource ratio.

To explore the scalability of this step further, another test was conducted without increasing the number of partitions after preprocessing. Consequently, the training and validation data were divided over one partition per input image. This reduction in resources for all image counts decreased the percentage of overhead per task. Figure 5.5 illustrates the execution time per training iteration. While the results remained stable, the lower execution time of the validation tasks, which are smaller tasks, indicates that overhead does not dominate these numbers. An intriguing observation is that the execution time per iteration actually improved compared to full resource utilization. This improvement is likely attributed to the scheduler having less workload in assigning tasks to its resources due to the reduced number of tasks.

Conclusion:

In conclusion, the scalability of machine learning algorithms on large-scale data infrastructure is a crucial aspect in the era of big data. As organizations increasingly rely on vast amounts of data to drive insights and decision-making, the ability to effectively scale machine learning algorithms becomes paramount. Throughout this study, we have explored various dimensions of scaling machine learning algorithms, including architectural considerations, performance optimization techniques, and challenges inherent in handling large volumes of data.

One key finding is the importance of designing scalable architectures that can efficiently process and analyze massive datasets. This involves leveraging distributed computing frameworks such as Apache Spark, Hadoop, or specialized platforms like TensorFlow Extended (TFX) and Apache Flink. By parallelizing computation and storage, these frameworks enable the efficient execution of machine learning algorithms across clusters of nodes, thus accommodating the scalability requirements of modern data infrastructure.

Moreover, we have examined different strategies for scaling specific machine learning algorithms, ranging from traditional models like linear regression and decision trees to more advanced deep learning architectures such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs). Techniques such as data partitioning, model parallelism, and parameter servers have been explored to distribute computation and mitigate bottlenecks, thereby enhancing scalability without compromising on performance.

However, scaling machine learning algorithms on large-scale data infrastructure is not without its challenges. Issues such as data skew, communication overhead, and resource contention can arise when dealing with heterogeneous datasets and distributed environments. Addressing these challenges requires a combination of algorithmic optimizations, system-level improvements, and architectural innovations.

Looking ahead, the field of scalable machine learning continues to evolve rapidly, driven by advancements in distributed computing, cloud technologies, and specialized hardware accelerators like GPUs and TPUs. Future research directions may include exploring novel algorithmic paradigms tailored for distributed settings, devising more efficient data processing pipelines, and integrating machine learning seamlessly into real-time, streaming data applications.

In summary, the scalability of machine learning algorithms on large-scale data infrastructure is a multifaceted problem that demands interdisciplinary solutions spanning computer science, statistics, and domain-specific expertise. By addressing the scalability challenges effectively, organizations can unlock the full potential of their data assets and empower data-driven decision-making at scale.

References:

[1]. Rehan, H. (2024). Revolutionizing America's Cloud Computing the Pivotal Role of AI in Driving Innovation and Security. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *2*(1), 189-208. DOI: https://doi.org/10.60087/jaigs.v2i1.p208

[2]. Rehan, H. (2024). AI-Driven Cloud Security: The Future of Safeguarding Sensitive Data in the Digital Age. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *1*(1), 47-66. DOI: https://doi.org/10.60087/jaigs.v1i1.p66

[3]. Li, Z., Huang, Y., Zhu, M., Zhang, J., Chang, J., & Liu, H. (2024). Feature Manipulation for DDPM based Change Detection. *arXiv preprint arXiv:2403.15943*.

https://doi.org/10.48550/arXiv.2403.15943

[4]. Ramírez, J. G. C. (2023). Incorporating Information Architecture (ia), Enterprise Engineering (ee) and Artificial Intelligence (ai) to Improve Business Plans for Small Businesses in the United States. *Journal of Knowledge Learning and Science Technology ISSN: 2959-6386 (online)*, *2*(1), 115-127. **DOI:** https://doi.org/10.60087/jklst.vol2.n1.p127

[5]. Ramírez, J. G. C. (2024). AI in Healthcare: Revolutionizing Patient Care with Predictive Analytics and Decision Support Systems. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *1*(1), 31-37. **DOI:** https://doi.org/10.60087/jaigs.v1i1.p37

[6]. Ramírez, J. G. C. (2024). Natural Language Processing Advancements: Breaking Barriers in Human-Computer Interaction. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *3*(1), 31-39. **DOI:** https://doi.org/10.60087/jaigs.v3i1.63

[7]. Ramírez, J. G. C., & mafiqul Islam, M. (2024). Application of Artificial Intelligence in Practical Scenarios. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *2*(1), 14-19. **DOI:** https://doi.org/10.60087/jaigs.v2i1.41

[8]. Ramírez, J. G. C., & Islam, M. M. (2024). Utilizing Artificial Intelligence in Real-World Applications. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *2*(1), 14-19.

DOI: https://doi.org/10.60087/jaigs.v2i1.p19

[9]. Ramírez, J. G. C., Islam, M. M., & Even, A. I. H. (2024). Machine Learning Applications in Healthcare: Current Trends and Future Prospects. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *1*(1). **DOI:** https://doi.org/10.60087/jaigs.v1i1.33

[10]. RAMIREZ, J. G. C. (2023). How Mobile Applications can improve Small Business Development. *Eigenpub Review of Science and Technology*, *7*(1), 291-305. **https://studies.eigenpub.com/index.php/erst/article/view/55**

[11]. RAMIREZ, J. G. C. (2023). From Autonomy to Accountability: Envisioning AI's Legal Personhood. *Applied Research in Artificial Intelligence and Cloud Computing*, *6*(9), 1-16. https://researchberg.com/index.php/araic/article/view/183

[12]. Ramírez, J. G. C., Hassan, M., & Kamal, M. (2022). Applications of Artificial Intelligence Models for Computational Flow Dynamics and Droplet Microfluidics. *Journal of Sustainable Technologies and Infrastructure Planning*, *6*(12). https://publications.dlpress.org/index.php/JSTIP/article/view/70

[13]. Ramírez, J. G. C. (2022). Struggling Small Business in the US. The next challenge to economic recovery. *International Journal of Business Intelligence and Big Data Analytics*, *5*(1), 81-91. https://research.tensorgate.org/index.php/IJBIBDA/article/view/99

[14]. Ramírez, J. G. C. (2021). Vibration Analysis with AI: Physics-Informed Neural Network Approach for Vortex-Induced Vibration. *International Journal of Responsible Artificial Intelligence*, *11*(3). https://neuralslate.com/index.php/Journal-of-Responsible-AI/article/view/77

[15]. Shuford, J. (2024). Interdisciplinary Perspectives: Fusing Artificial Intelligence with Environmental Science for Sustainable Solutions. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *1*(1), 1-12. **DOI:** https://doi.org/10.60087/jaigs.v1i1.p12

[16]. Islam, M. M. (2024). Exploring Ethical Dimensions in AI: Navigating Bias and Fairness in the Field. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *1*(1), 13-17. **DOI:** https://doi.org/10.60087/jaigs.v1i1.p18

[17]. Khan, M. R. (2024). Advances in Architectures for Deep Learning: A Thorough Examination of Present Trends. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *1*(1), 24-30. **DOI:** https://doi.org/10.60087/jaigs.v1i1.p30

[18]. Shuford, J., & Islam, M. M. (2024). Exploring the Latest Trends in Artificial Intelligence Technology: A Comprehensive Review. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *2*(1). **DOI:** https://doi.org/10.60087/jaigs.v2i1.p13

[19]. Islam, M. M. (2024). Exploring the Applications of Artificial Intelligence across Various Industries. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *2*(1), 20-25. **DOI:** https://doi.org/10.60087/jaigs.v2i1.p25

[20]. Akter, S. (2024). Investigating State-of-the-Art Frontiers in Artificial Intelligence: A Synopsis of Trends and Innovations. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *2*(1), 25-30. **DOI:** https://doi.org/10.60087/jaigs.v2i1.p30

[21]. Rana, S. (2024). Exploring the Advancements and Ramifications of Artificial Intelligence. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *2*(1), 30-35. **DOI:** https://doi.org/10.60087/jaigs.v2i1.p35

[22]. Sarker, M. (2024). Revolutionizing Healthcare: The Role of Machine Learning in the Health Sector. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *2*(1), 35-48.

DOI: https://doi.org/10.60087/jaigs.v2i1.p47

[23]. Akter, S. (2024). Harnessing Technology for Environmental Sustainability: Utilizing AI to Tackle Global Ecological Challenges. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *2*(1), 49-57. **DOI:** https://doi.org/10.60087/jaigs.v2i1.p57

[24]. Padmanaban, H. (2024). Revolutionizing Regulatory Reporting through AI/ML: Approaches for Enhanced Compliance and Efficiency. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *2*(1), 57-69. **DOI:** https://doi.org/10.60087/jaigs.v2i1.p69

[25]. Padmanaban, H. (2024). Navigating the Role of Reference Data in Financial Data Analysis: Addressing Challenges and Seizing Opportunities. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *2*(1), 69-78. **DOI:** https://doi.org/10.60087/jaigs.v2i1.p78

[26]. Camacho, N. G. (2024). Unlocking the Potential of AI/ML in DevSecOps: Effective Strategies and Optimal Practices. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *2*(1), 79-89. **DOI:** https://doi.org/10.60087/jaigs.v2i1.p89

[27]. PC, H. P., & Sharma, Y. K. (2024). Developing a Cognitive Learning and Intelligent Data Analysis-Based Framework for Early Disease Detection and Prevention in Younger Adults with Fatigue. *Optimized Predictive Models in Health Care Using Machine Learning*, 273. https://books.google.com.bd/books?hl=en&lr=&id=gtXzEAAAQBAJ&oi=fnd&pg=PA273&dq=Developing+a+Cognitive+Learning+and+Intelligent+Data+Analysis-Based+Framework+for+Early+Disease+Detection+and+Prevention+in+Younger+Adults+with+Fatigue&ots=wKUZk_Q0IG&sig=WDlXjvDmc77Q7lvXW9MxIh9Iz-Q&redir_esc=y#v=onepage&q=Developing%20a%20Cognitive%20Learning%20and%20Intelligent%20Data%20Analysis-Based%20Framework%20for%20Early%20Disease%20Detection%20and%20Prevention%20in%20Younger%20Adults%20with%20Fatigue&f=false

[28]. Padmanaban, H. (2024). Quantum Computing and AI in the Cloud. *Journal of Computational Intelligence and Robotics*, *4*(1), 14–32. Retrieved from https://thesciencebrigade.com/jcir/article/view/116

[29]. Sharma, Y. K., & Harish, P. (2018). Critical study of software models used cloud application development. *International Journal of Engineering & Technology, E-ISSN*, 514-518. https://www.researchgate.net/profile/Harish-Padmanaban-2/publication/377572317_Critical_study_of_software_models_used_cloud_application_development/links/65ad55d7ee1e1951fbd79df6/Critical-study-of-software-models-used-cloud-application-development.pdf

[30]. Padmanaban, P. H., & Sharma, Y. K. (2019). Implication of Artificial Intelligence in Software Development Life Cycle: A state of the art review. *vol*, *6*, 93-98. https://www.researchgate.net/profile/Harish-Padmanaban-2/publication/377572222_Implication_of_Artificial_Intelligence_in_Software_Development_Life_Cycle_A_state_of_the_art_review/links/65ad54e5bf5b00662e333553/Implication-of-Artificial-Intelligence-in-Software-Development-Life-Cycle-A-state-of-the-art-review.pdf

[31]. Harish Padmanaban, P. C., & Sharma, Y. K. (2024). Optimizing the Identification and Utilization of Open Parking Spaces Through Advanced Machine Learning. *Advances in Aerial Sensing and Imaging*, 267-294. **https://doi.org/10.1002/9781394175512.ch12**

[32]. PC, H. P., Mohammed, A., & RAHIM, N. A. (2023). *U.S. Patent No. 11,762,755*. Washington, DC: U.S. Patent and Trademark Office. https://patents.google.com/patent/US20230385176A1/en

[33]. Padmanaban, H. (2023). Navigating the intricacies of regulations: Leveraging AI/ML for Accurate Reporting. *Journal of Knowledge Learning and Science Technology ISSN: 2959-6386 (online)*, *2*(3), 401-412. DOI: https://doi.org/10.60087/jklst.vol2.n3.p412

[34]. PC, H. P. Compare and analysis of existing software development lifecycle models to develop a new model using computational intelligence.
https://shodhganga.inflibnet.ac.in/handle/10603/487443

[35]. Camacho, N. G. (2024). Unlocking the Potential of AI/ML in DevSecOps: Effective Strategies and Optimal Practices. *Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023*, *2*(1), 79-89. DOI: https://doi.org/10.60087/jaigs.v2i1.p89

[36]. Gehrmann, S., & Rončević, I. (2015). Monolingualisation of research and science as a hegemonial project: European perspectives and Anglophone realities. Filologija, (65), 13-44.

[37]. Singla, A., & Malhotra, T. (2024). Challenges And Opportunities in Scaling AI/ML Pipelines. Journal of Science & Technology, 5(1), 1-21.

[38]. Singla, A., & Chavalmane, S. (2023). Automating Model Deployment: From Training to Production. Journal of Knowledge Learning and Science Technology ISSN: 2959-6386 (online), 2(3), 340-347.

[39]. Roncevic, I. (2021). Eye-tracking in second language reading. Eye, 15(5).

[40]. Latif, M. A., Afshan, N., Mushtaq, Z., Khan, N. A., Irfan, M., Nowakowski, G., ... & Telenyk, S. (2023). Enhanced classification of coffee leaf biotic stress by synergizing feature concatenation and dimensionality reduction. *IEEE Access*.
DOI: https://doi.org/10.1109/ACCESS.2023.3314590

[42]. Irfan, M., Mushtaq, Z., Khan, N. A., Mursal, S. N. F., Rahman, S., Magzoub, M. A., ... & Abbas, G. (2023). A Scalo gram-based CNN ensemble method with density-aware smote oversampling for improving bearing fault diagnosis. *IEEE Access*, *11*, 127783-127799.
DOI: https://doi.org/10.1109/ACCESS.2023.3332243

[43]. Irfan, M., Mushtaq, Z., Khan, N. A., Althobiani, F., Mursal, S. N. F., Rahman, S., ... & Khan, I. (2023). Improving Bearing Fault Identification by Using Novel Hybrid Involution-Convolution Feature Extraction with Adversarial Noise Injection in Conditional GANs. *IEEE Access*.

DOI: https://doi.org/10.1109/ACCESS.2023.3326367

[44]. Rahman, S., Mursal, S. N. F., Latif, M. A., Mushtaq, Z., Irfan, M., & Waqar, A. (2023, November). Enhancing Network Intrusion Detection Using Effective Stacking of Ensemble Classifiers With Multi-Pronged Feature Selection Technique. In *2023 2nd International Conference on Emerging Trends in Electrical, Control, and Telecommunication Engineering (ETECTE)* (pp. 1-6). IEEE.

DOI: https://doi.org/10.1109/ETECTE59617.2023.10396717

[45]. Latif, M. A., Mushtaq, Z., Arif, S., Rehman, S., Qureshi, M. F., Samee, N. A., ... & Al-masni, M. A. Improving Thyroid Disorder Diagnosis via Ensemble Stacking and Bidirectional Feature Selection.
https://doi.org/10.32604/cmc.2024.047621

[46]. Ara, A., & Mifa, A. F. (2024). INTEGRATING ARTIFICIAL INTELLIGENCE AND BIG DATA IN MOBILE HEALTH: A SYSTEMATIC REVIEW OF INNOVATIONS AND CHALLENGES IN HEALTHCARE SYSTEMS. *Global Mainstream Journal of Business, Economics, Development & Project Management*, *3*(01), 01-16.

DOI: https://doi.org/10.62304/jbedpm.v3i01.70

[47]. Bappy, M. A., & Ahmed, M. (2023). ASSESSMENT OF DATA COLLECTION TECHNIQUES IN MANUFACTURING AND MECHANICAL ENGINEERING THROUGH MACHINE LEARNING MODELS. *Global Mainstream Journal of Business, Economics, Development & Project Management*, *2*(04), 15-26.
DOI: https://doi.org/10.62304/jbedpm.v2i04.67

[48]. Bappy, M. A. (2024). Exploring the Integration of Informed Machine Learning in Engineering Applications: A Comprehensive Review. *American Journal of Science and Learning for Development*, *3*(2), 11-21.
DOI: https://doi.org/10.51699/ajsld.v3i2.3459

[49]. Uddin, M. N., Bappy, M. A., Rab, M. F., Znidi, F., & Morsy, M. (2024). Recent Progress on Synthesis of 3D Graphene, Properties, and Emerging Applications.
DOI: https://doi.org/10.5772/intechopen.114168

[50]. Hossain, M. I., Bappy, M. A., & Sathi, M. A. (2023). WATER QUALITY MODELLING AND ASSESSMENT OF THE BURIGANGA RIVER USING QUAL2K. *Global Mainstream Journal of Innovation, Engineering & Emerging Technology*, *2*(03), 01-11.
DOI: https://doi.org/10.62304/jieet.v2i03.64

[51]. Talati, D. (2023). Telemedicine and AI in Remote Patient Monitoring. Journal of Knowledge Learning and Science Technology ISSN: 2959-6386 (online), 2(3), 254-255.

[52]. Talati, D. (2023). Artificial Intelligence (Ai) In Mental Health Diagnosis and Treatment. Journal of Knowledge Learning and Science Technology ISSN: 2959-6386 (online), 2(3), 251-253.

[53]. Talati, D. (2023). AI in healthcare domain. Journal of Knowledge Learning and Science Technology ISSN: 2959-6386 (online), 2(3), 256-262.