Inventory Management using Transfer Learning

Done By:

Chinmoy Saikia
Subhadip Guha Roy
Tushar Kumar

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This report introduces artificial intelligence (AI) model for inventory management, leveraging transfer learning to address the dynamic needs of businesses. The model aims to optimize stock levels by analyzing historical sales data, purchase prices, inventory levels, and other variables. This proactive approach minimizes overstocking and stock-outs, reduces costs, enhances customer satisfaction, and enables strategic planning.

It is divided into two parts: the first outlines the development of a robust base model using a Multilayer Perceptron (MLP) architecture, optimized through hyperparameter tuning and validated against traditional models. The second explores how transfer learning customizes this base model for specific business requirements, ensuring precise predictions and operational efficiency.

The deployment process details integrating the model into inventory systems for actionable insights. The results highlight the transformative potential of AI in inventory management, providing a scalable, accurate, and cost-effective solution for diverse industries.

CHAPTER 2

PROBLEM STATEMENT

Efficient inventory management is a critical aspect of supply chain optimization, directly impacting operational costs, stock availability, and customer satisfaction. Traditional inventory management techniques, such as manual stock control, simple statistical methods, and rule-based systems, often struggle to handle the complexities of dynamic demand fluctuations, lead times, and supply chain uncertainties. In today's fast-paced business environment, businesses face challenges in maintaining an optimal balance between understocking and overstocking, resulting in increased costs, stockouts, or excess inventory.

Recent advances in Artificial Intelligence (AI) and Machine Learning (ML) have opened new avenues for improving inventory management by providing more accurate demand forecasting, dynamic stock optimization, and automated replenishment systems. Despite these promising developments, there remains a lack of comprehensive understanding on how different AI/ML models can be effectively applied to address the diverse challenges of inventory management, such as:

Accurate Demand Forecasting: Traditional methods often fail to capture complex demand patterns, seasonal variations, and external factors that influence product demand. AI/ML-based demand forecasting models, especially those utilizing deep learning and reinforcement learning, offer potential improvements but need further exploration for real-world applications.

Dynamic Stock Optimization: Optimizing stock levels in response to real-time changes in demand, supply chain disruptions, and market conditions is a significant challenge. The ability to employ machine learning techniques to adapt inventory policies in real-time could reduce both excess inventory and stockouts, but there is a need for more efficient and scalable solutions.

CHAPTER 3

LITERATURE REVIEW

3.1 Overview

Effective inventory management is a cornerstone of supply chain optimization. Traditional inventory systems often face challenges such as overstocking, understocking, and inaccurate demand forecasting, which can result in financial losses and reduced customer satisfaction. The emergence of AI and machine learning (ML) has introduced sophisticated methods for addressing these challenges. This literature review explores existing studies and methods in inventory management, demand forecasting, and ML applications.

3.1.1 Inventory Management Challenges

Inventory management involves maintaining the balance between supply and demand while minimizing costs. Research highlights common issues such as demand variability, lead time uncertainty, and stock obsolescence. Traditional systems often rely on historical data and static models, which may lack the adaptability required for dynamic market conditions.

3.1.2 Role of Machine Learning in Demand Forecasting

Machine learning offers robust solutions for demand forecasting by leveraging large datasets and dynamic algorithms. Studies by Elmasri, R., Navathe, S. B., & Yan, C. (2021) demonstrate the superiority of ML models like neural networks, decision trees, and gradient boosting over traditional statistical methods. These models can capture non-linear patterns in data, enabling more accurate predictions

3.1.3 Use of MongoDB in Modern Inventory Systems

MongoDB, a NoSQL database, has gained traction in inventory management for its flexibility in handling unstructured data. Studies highlight its efficiency in managing large-scale data, especially when combined with ML models to store and retrieve real-time data.

3.1.4 Popular ML Techniques in Inventory Management

1. Time Series Analysis:

   - Techniques such as ARIMA and SARIMA have been widely used for demand prediction. However, they are increasingly being replaced by LSTMs (Long Short-Term Memory networks) for handling sequential data with long-term dependencies.

2. Supervised Learning:

   - Algorithms like Random Forest and Gradient Boosting are frequently employed for classification tasks, such as determining product categories or predicting reorder points.

3. Unsupervised Learning:

   - Clustering methods, including k-Means and DBSCAN, are applied to segment products based on demand patterns or other features.

3.2 Traditional Method Vs Modern Method

In today's dynamic business world, effective inventory management is crucial to a company's profitability and competitiveness. Many organizations struggle with challenges such as costly overages, disastrous stock-outs, and high operational costs. To address these challenges, this project explores the application of Deep Learning techniques, with a particular focus on transfer learning, to revolutionize inventory management systems.

3.2.1 Data Processing:

Traditional methods often rely on historical averages or rule-based systems to estimate inventory needs, lacking the capacity to analyze complex, high-dimensional datasets. Modern AI and machine learning approaches utilize large volumes of data, including real-time inputs, to provide more dynamic and precise predictions.

3.2.2 Modeling Techniques:

Traditional techniques include linear regression, moving averages, and static forecasting models, which may struggle with non-linear patterns or changing trends. Modern methods employ sophisticated models like deep neural networks, which can capture intricate relationships within the data and adapt to evolving market dynamics.

3.2.3 Scalability:

Conventional systems are often limited in their ability to scale across multiple products or locations without significant manual intervention.AI-driven systems are inherently scalable, leveraging automation to manage vast datasets and multiple variables effortlessly.

3.2.4 Customization and Adaptability:

Traditional systems typically lack customization capabilities and require manual recalibration for different scenarios. Modern systems with transfer learning allow rapid customization to specific datasets, enabling businesses to adapt swiftly to unique operational contexts.

3.2.5 Efficiency and Cost-Effectiveness:

Legacy methods can be resource-intensive, requiring significant time and expertise to update and maintain.

AI models, particularly those utilizing transfer learning, streamline processes, reducing development costs and improving time-to-deployment.

CHAPTER 4

REQUIREMENTS

4.1 Functional Requirements

Functional requirements are those that state the fundamental features and functionalities that the system should be supporting. Some of them are :-

4.1.1 Data Handling

The system must be able to import and process large datasets, including historical sales, inventory levels, and purchase price data. It should clean, normalize, and encode data to ensure it is ready for analysis and model training.

4.1.2 Model Training

Train a base deep learning model using Multilayer Perceptron (MLP) architecture for initial inventory predictions. Implement hyperparameter tuning using methods like Grid Search or Bayesian optimization to improve model performance. Support the creation and comparison of reference models (e.g., Linear Regression, Random Forest) for benchmarking.

4.1.3 Demand Forecasting and Strategic Insights

Generate accurate demand forecasts at both product and category levels. Predict optimal inventory levels to avoid overstocking and stock-outs. Provide insights into strategic decisions like resource allocation and supply chain adjustments.

4.1.4 Model Evaluation

Evaluate the model's performance using metrics such as RMSE, MAE, R^2, and correlation coefficients. Visualize the results through comparative graphs of actual vs. predicted values.

4.1.5 Prediction Capabilities

Generate accurate demand forecasts at both product and category levels. Predict optimal inventory levels to avoid overstocking and stock-outs. Provide insights into strategic decisions like resource allocation and supply chain adjustments.

4.2 Non-Functional Requirements

Non-functional requirements are the quality attributes as well as the operational characteristics. It is the criteria that defines how a system should behave rather than what it is supposed to do.

4.2.1 Performance

- The system must process large datasets efficiently, ensuring data preprocessing and model training are completed within reasonable time limits.
- Predictions should be generated in real-time or near-real-time to support operational decision-making.

4.2.2 Scalability

- The model should handle an increasing number of products, stores, or regions without significant degradation in performance.
- Support for high-dimensional datasets and multi-location operations should be inherent.

4.2.3 Security

- Ensure secure storage and handling of sensitive data, such as sales and inventory records. Provide access controls to restrict unauthorized use of the system or model.

4.2.4 Maintainability

- The system must be modular, allowing easy updates to the model, algorithms, or data preprocessing components.
- Developers should be able to troubleshoot and resolve issues efficiently.

4.2.5 Portability

- The model and associated software should be platform-independent, capable of running on various operating systems (e.g., Linux, Windows).
- Deployment on both on-premises infrastructure and cloud platforms should be supported.

4.2.6 Security

- Ensure secure storage and handling of sensitive data, such as sales and inventory records.
- Provide access controls to restrict unauthorized use of the system or model.
-

4.3 Hardware Requirements:

System Configuration

- Processor: Multi-core processor, preferably with GPU support for faster model training and evaluation (e.g., NVIDIA CUDA-enabled GPU).
- Memory (RAM): Minimum 16 GB of RAM; 32 GB is recommended for handling large datasets.
- Storage:
  - 1 TB SSD or higher for fast data access and storage of datasets, models, and logs.
  - Additional external storage for backup if required.
- GPU: NVIDIA GPU with at least 8 GB of VRAM (e.g., NVIDIA RTX 3060 or better) for deep learning tasks.
- Power Supply: Adequate to support the GPU and other components.
- Networking: High-speed internet connection for downloading dependencies and large datasets.

4.4 Software Requirements

- Operating System:
  - Windows 10/11 (64-bit)
  - macOS 12+ (for compatible hardware)
  - Ubuntu 20.04 LTS or later (recommended for better deep learning compatibility)
- Programming Language:
  - Python 3.8 or later
- Libraries and Frameworks:
  - TensorFlow 2.4+
  - Keras (integrated with TensorFlow)
  - scikit-learn 1.3+
  - pandas 2.0+
  - matplotlib 3.7+
  - numpy 1.20+
  - PrettyTable

- Tools and IDEs:
    - Jupyter Notebook or JupyterLab for notebook files.
    - Visual Studio Code with Python extension for code editing.
    - Anaconda for environment management.
- Additional Dependencies:
    - CUDA Toolkit and cuDNN (for GPU support with TensorFlow)
    - Conda for environment setup and dependency management.
- Environment Setup:
    - Use Anaconda to create an isolated environment with dependencies from requirements.txt.
    - Install GPU support libraries if using TensorFlow with GPU acceleration.

CHAPTER 5

SYSTEM ARCHITECTURE AND DESIGN

The system architecture comprises multiple layers designed for efficiency and scalability. The data layer manages raw and preprocessed data, including historical sales and inventory records. The data processing layer cleans, normalizes, and encodes data for analysis. The model development layer focuses on building and training a base deep learning model (MLP) and benchmarking it against traditional methods. The transfer learning layer adapts the base model to specific business requirements using fine-tuning techniques. The evaluation layer ensures model reliability through performance metrics

5.1 System Design

The system design for the inventory management solution is structured to ensure efficient data handling, accurate predictions, and user-friendly interaction. The Data Ingestion Layer collects historical sales data, inventory levels, product details, and supplier information from sources like CSV files, databases, or cloud storage. This data is processed in the Data Processing Layer, where it undergoes cleaning, normalization, and feature engineering using tools like pandas, NumPy, and scikit-learn to prepare it for model training.

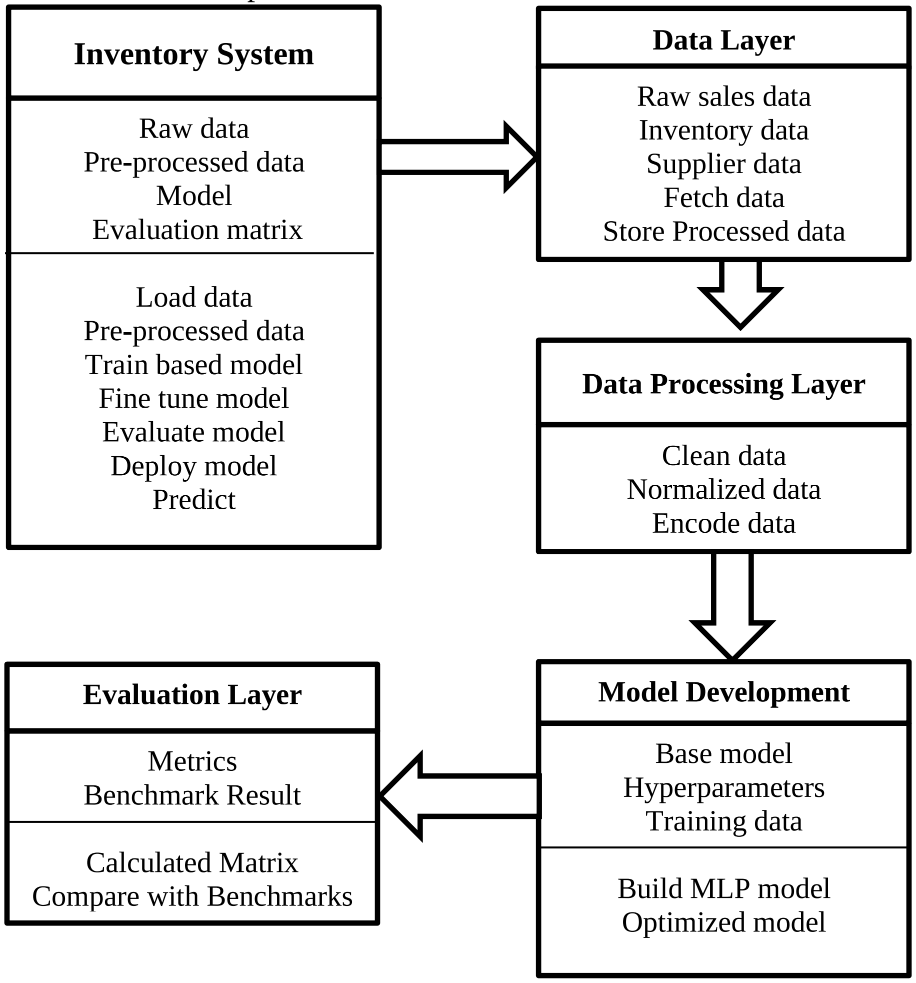
The Model Development Layer includes a base model, typically an MLP (Multilayer Perceptron) architecture, trained on pre-processed data. The system also incorporates benchmarking against models like Random Forest and Linear Regression to ensure accuracy. Fine-tuning is achieved through a Transfer Learning Layer, where the pre-trained base model is adapted to new, business-specific datasets by freezing initial layers and optimizing higher layers. To evaluate the model's performance, the Model Evaluation Layer calculates metrics like RMSE, MAE, and R^2 and compares results with benchmark models.

The system integrates with a database to store raw, processed, and prediction data, ensuring centralized access for analysis. A robust error-handling mechanism validates input data and alerts users on anomalies in predictions. The design supports modularity and scalability, with backend components in Python (TensorFlow, Flask).

5.2 Class Diagram

The class diagram for the inventory management system is designed to represent its modular structure and logical flow. At the center is the Inventory System class, which orchestrates all operations, including data handling, model training, evaluation, and deployment. Supporting this are several modular classes. The Data Layer manages raw and preprocessed datasets, ensuring the system has access to clean and structured data. The Data Processing Layer handles data cleaning, feature engineering, and aggregation, preparing inputs for model training.

The Model Development class builds and trains the base model, incorporating hyperparameter optimization for improved performance. The Transfer Learning Layer adapts the base model to specific business needs by freezing certain layers and fine-tuning others using new data. Once trained, the model's performance is assessed by the Evaluation Layer, which calculates metrics like RMSE and compares results with benchmarks



5.3 Methodology

The central objective of our initiative is to use the knowledge acquired during the training of the basic model to carry out transfer learning and adapt it to the specific needs of companies, a process we'll detail in greater detail in this report. To better understand this approach, we will begin by exploring in detail the first stage of our project, which consists of training a basic model for inventory management.

5.3.1 Fundamental Data Collection and Preparation

We used a massive dataset including over 70 stores and a significant number of transactions, over a million transactions for 6890 products. This historical sales data, combined with information on purchase prices, inventory levels and other relevant variables, forms the essential backdrop to our model. The data collection and preparation process were essential to ensure its quality and usefulness in training our model. The diversity of stores and products, as well as the impressive volume of transactions, required careful management to ensure an accurate representation of reality in our case study. The data cleaning, processing and structuring steps were crucial to ensure that our inventory management model could take full advantage of this rich source of information.

5.3.2 Creating Reference Models

Before diving into the design of our AI-based inventory management model, we took care to create reference models. These reference models encompass classic machine learning methods, such as linear regression and ensemble methods. The main purpose of these reference models is to serve as a point of comparison, enabling us to evaluate the performance of our basic model in a rigorous and
thorough way.

5.3.4 Designing the Basic Inventory Management Model Multilayer Perceptron (MLP), the Ideal Choice

In our exploration of intelligent inventory management, we focus on the selection of the initial model, a crucial step for the future of our project. This model will serve as a solid foundation for our transfer learning approach, which we'll describe in greater detail later in this document. After a careful analysis of the various options, we opted for the Multilayer Perceptron (MLP) as the basic model. The MLP stands out for its ability to efficiently and accurately extract key features from data. This feature makes it the ideal choice for our initial model.

By using MLP as a starting point, we ensure that our inventory management model is robust and flexible, ready to be adapted to the specific needs of each company. This reinforces our confidence in our ability to offer tailor-made solutions that exceed business expectations.

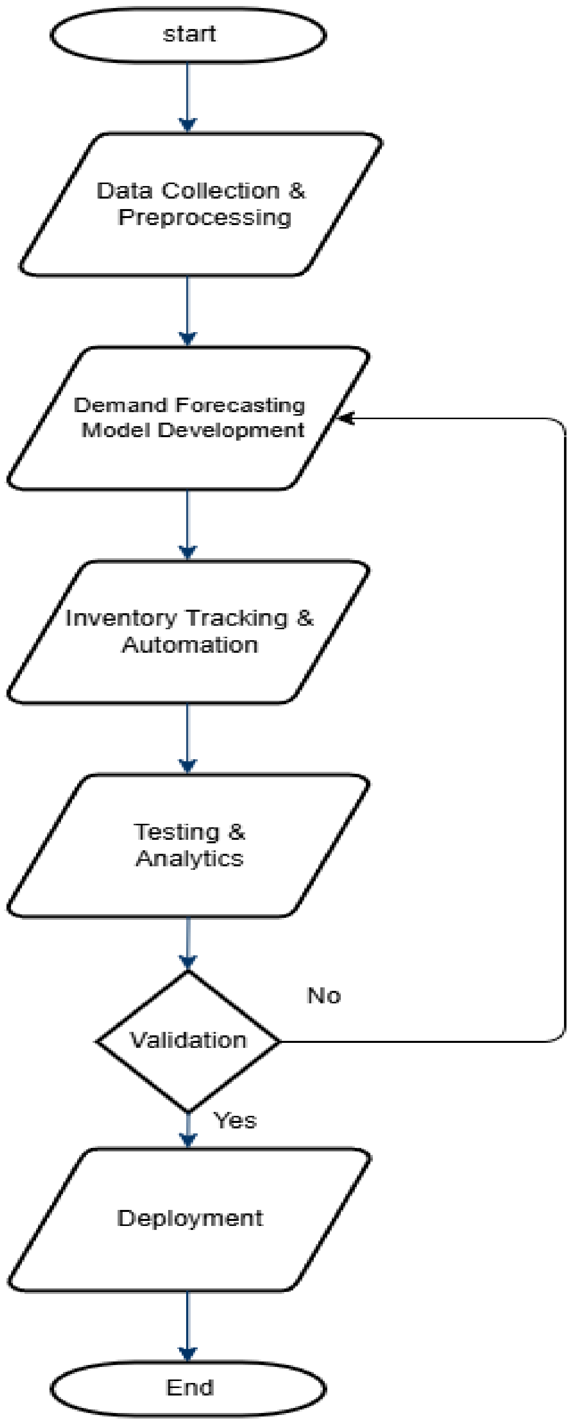
5.3.5 Optimization of Base Model Hyperparameters

Hyperparameter optimization plays a crucial role in the formation of our base model. This step involves adjusting the internal parameters of our model so that it can best extract the important information from our data.

We have used two techniques for this optimization, each with its own advantages

- Grid Search: This technique is a systematic method that explores a predefined set of hyperparameter combinations. This allows us to test different configurations for our model, such as learning speed and neural network structure. It ensures that we don't overlook any potential options.
- Bayesian optimization: This more advanced approach relies on probabilistic models to estimate the expected performance of different hyperparameter configurations. It takes into account previous results to decide which configurations deserve further attention. This is particularly useful when the hyperparameter space is complex.

By combining these two techniques, we were able to find the best settings for our base model, resulting in an improvement in the accuracy of our predictions. This reinforces the overall robustness of our inventory management model.


start
Data Collection & Preprocessing
Demand Forecasting Model Development
Inventory Tracking & Automation
Testing & Analytics
Validation
No
Yes
Deployment
End

CHAPTER 6
IMPLEMENTATION

The AI-Based Inventory Management System is a cutting-edge application designed to streamline inventory processes using machine learning and modern web technologies. This project focuses on demand forecasting, efficient stock management, and automation. The following technologies, libraries, and techniques were utilized to achieve scalability, responsiveness, and robustness.

6.1 Environment Setup

The first stage involves preparing the environment for development. This is essential to ensure that all necessary tools, libraries, and dependencies are available and function harmoniously. Start by installing Python 3.8+ as the primary programming language for the system. Use Anaconda to create and manage an isolated virtual environment specifically for this project. This isolation prevents conflicts between libraries and dependencies required by other projects.

Install the core libraries, such as TensorFlow (for deep learning), Keras (a high-level API for TensorFlow), scikit-learn (for benchmarking with traditional machine learning models), pandas (for data manipulation), numpy (for numerical computations), and matplotlib (for data visualization). Ensure Jupyter Notebook is installed to allow exploratory work and experiments. Additionally, configure ipykernel to integrate the environment with Jupyter. This setup ensures a robust and reproducible environment that supports efficient development and testing.

6.2 Data Ingestion and Preprocessing

Data is the backbone of any machine learning project, and this step involves collecting, cleaning, and preparing data for use in the inventory management system. Start by gathering data from relevant sources, such as historical sales data, product purchase details, inventory records, and supplier information. These datasets may be stored in CSV files, SQL databases, or cloud storage platforms like AWS S3.

Load the raw data into pandas Data Frames for processing. Preprocessing involves several critical steps: handling missing values to ensure data completeness, removing outliers to avoid skewing predictions, and normalizing numerical variables to ensure consistent scaling across features. Encode categorical data, such as product categories or store identifiers, using techniques like one-hot encoding to make them machine-readable. Aggregate sales data into meaningful time intervals (e.g., daily, weekly, or monthly) to better align with inventory management objectives. Save the cleaned and preprocessed data into a designated folder for efficient reuse during modeling.

6.3 Base Model Development

This stage focuses on designing and building a deep learning model that serves as the foundation for inventory prediction. A Multilayer Perceptron (MLP) architecture is chosen due to its ability to effectively capture patterns and relationships in structured data. The model architecture is designed with multiple layers, each performing specific roles such as feature extraction and prediction.

Train the MLP model on the preprocessed historical data, which includes variables like sales history, inventory levels, purchase prices, and product categories. To maximize model performance, hyperparameter tuning is conducted using techniques like grid search and Bayesian optimization. This involves adjusting parameters such as learning rate, batch size, and the number of neurons in each layer. Once the base model achieves satisfactory accuracy, save its trained weights as a checkpoint. This base model will serve as the core for further customization in the transfer learning stage.

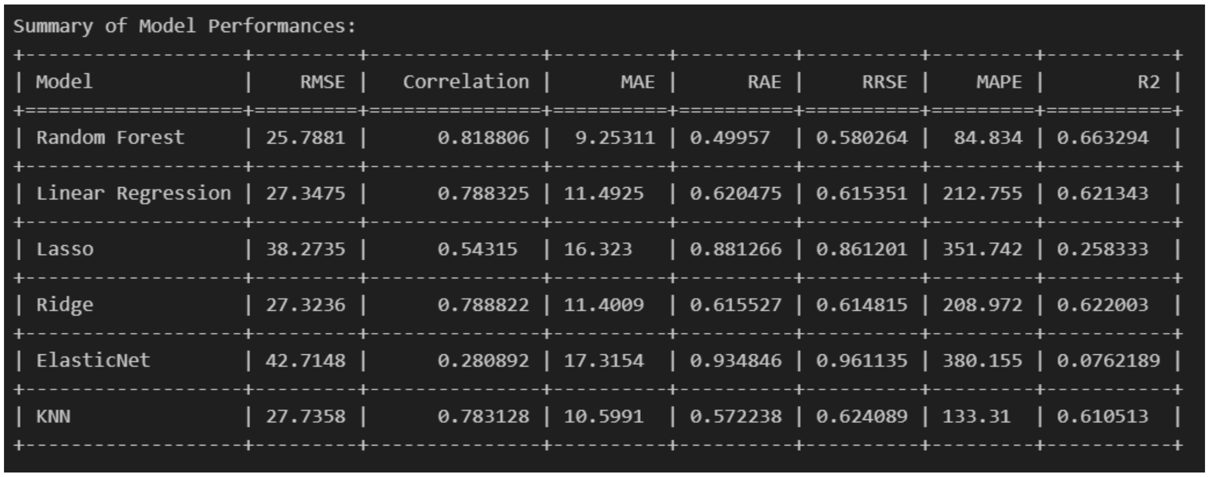
6.4 Model Evaluation

After training, the model’s performance must be rigorously evaluated to ensure it meets the required standards. This involves testing the model on a separate validation dataset that it hasn’t seen during training. Key performance metrics include Root Mean Square Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and R^2 (coefficient of determination). These metrics assess how accurately the model predicts inventory demand and how well it captures trends in the data.

To benchmark the model, compare its performance with traditional machine learning models, such as Random Forest, Linear Regression, Ridge, Lasso, ElasticNet, and K-Nearest Neighbors. This comparative analysis demonstrates the effectiveness of the deep learning model over conventional methods. The evaluation phase ensures that the model is reliable and ready for real-world deployment.

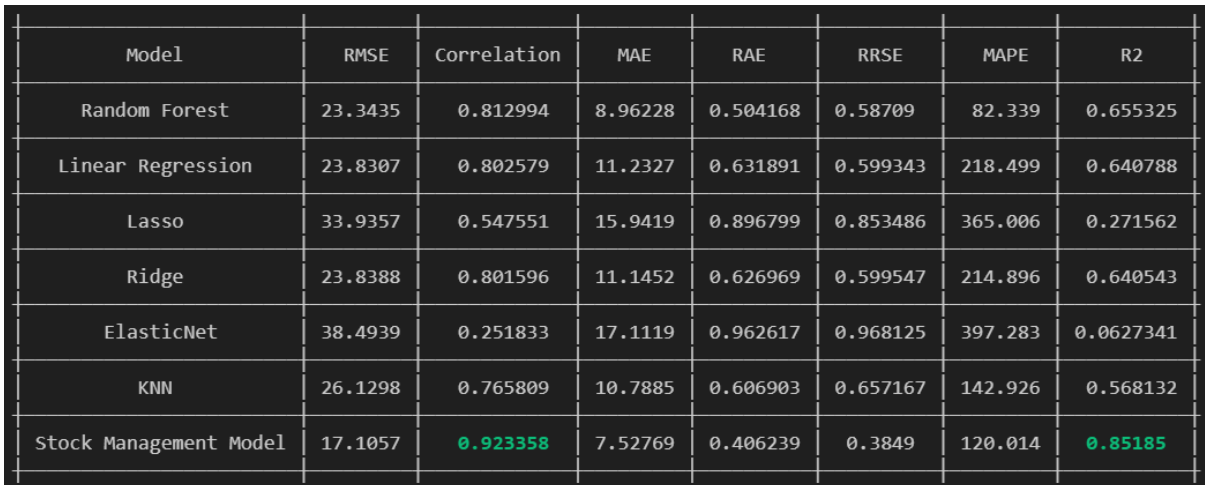
CHAPTER 7

RESULTS



```
Summary of Model Performances:
+-------------------+---------+--------------+---------+---------+---------+---------+-----------+
| Model             |    RMSE |  Correlation |     MAE |     RAE |    RRSE |    MAPE |        R2 |
+===================+=========+==============+=========+=========+=========+=========+===========+
| Random Forest     | 25.7881 |     0.818806 | 9.25311 | 0.49957 | 0.580264 |  84.834 | 0.663294  |
+-------------------+---------+--------------+---------+---------+---------+---------+-----------+
| Linear Regression | 27.3475 |     0.788325 | 11.4925 | 0.620475 | 0.615351 | 212.755 | 0.621343  |
+-------------------+---------+--------------+---------+---------+---------+---------+-----------+
| Lasso             | 38.2735 |     0.54315  | 16.323  | 0.881266 | 0.861201 | 351.742 | 0.258333  |
+-------------------+---------+--------------+---------+---------+---------+---------+-----------+
| Ridge             | 27.3236 |     0.788822 | 11.4009 | 0.615527 | 0.614815 | 208.972 | 0.622003  |
+-------------------+---------+--------------+---------+---------+---------+---------+-----------+
| ElasticNet        | 42.7148 |     0.280892 | 17.3154 | 0.934846 | 0.961135 | 380.155 | 0.0762189 |
+-------------------+---------+--------------+---------+---------+---------+---------+-----------+
| KNN               | 27.7358 |     0.783128 | 10.5991 | 0.572238 | 0.624089 | 133.31  | 0.610513  |
+-------------------+---------+--------------+---------+---------+---------+---------+-----------+
```

Fig: Obtain data in the formats required by the models, then train classical learning models such as Random Forest, linear regression, Lasso, Ridge, ElasticNet, KNN to obtain the performance.



```
|         Model          |  RMSE   | Correlation |   MAE   |   RAE    |   RRSE   |  MAPE   |    R2     |
|     Random Forest      | 23.3435 |  0.812994   | 8.96228 | 0.504168 | 0.58709  | 82.339  | 0.655325  |
|   Linear Regression    | 23.8307 |  0.802579   | 11.2327 | 0.631891 | 0.599343 | 218.499 | 0.640788  |
|         Lasso          | 33.9357 |  0.547551   | 15.9419 | 0.896799 | 0.853486 | 365.006 | 0.271562  |
|         Ridge          | 23.8388 |  0.801596   | 11.1452 | 0.626969 | 0.599547 | 214.896 | 0.640543  |
|       ElasticNet       | 38.4939 |  0.251833   | 17.1119 | 0.962617 | 0.968125 | 397.283 | 0.0627341 |
|          KNN           | 26.1298 |  0.765809   | 10.7885 | 0.606903 | 0.657167 | 142.926 | 0.568132  |
| Stock Management Model | 17.1057 |  0.923358   | 7.52769 | 0.406239 |  0.3849  | 120.014 |  0.85185  |
```

Fig: Load the optimized model, make predictions on the test data and calculate the errors. We can then compare performance with other models, such as the reference models created previously. From the given data regarding our Stock Management Model, we can see that the value of Correlation is ~ 0.92 and R^2 is ~ 0.85 which means that our model is performing better than the other mode

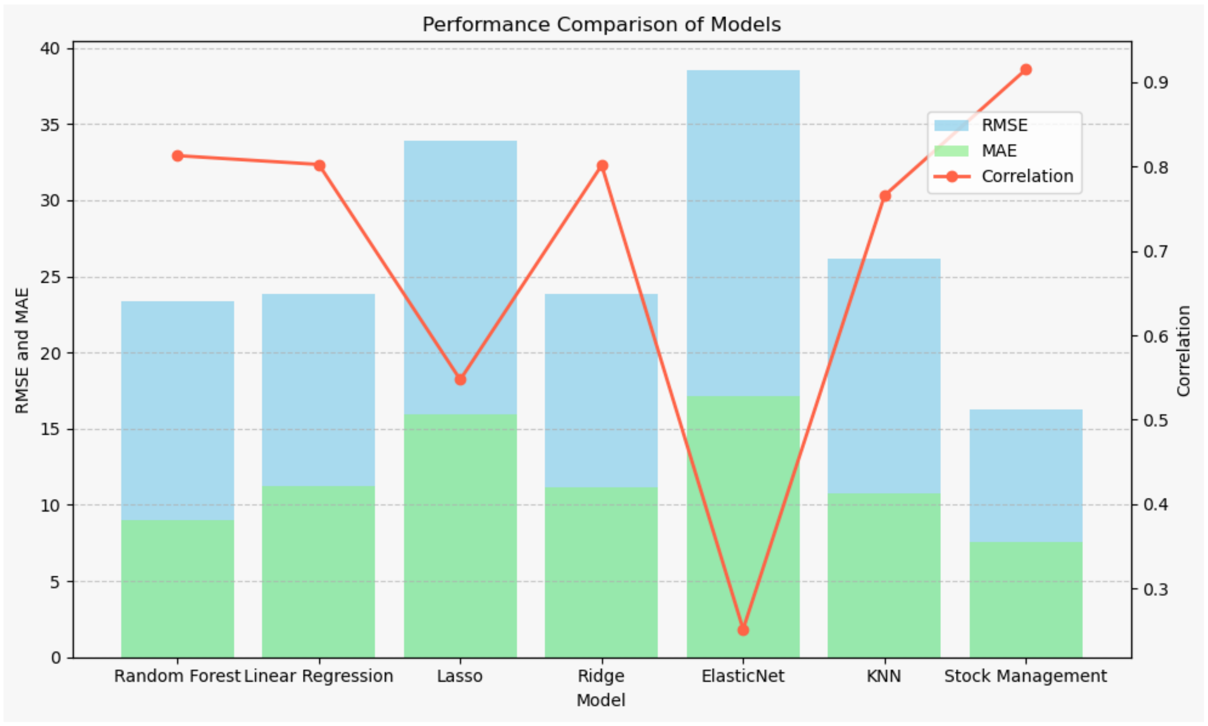


Fig: Comparison of Performance parameter among all the models using Bar Graph.

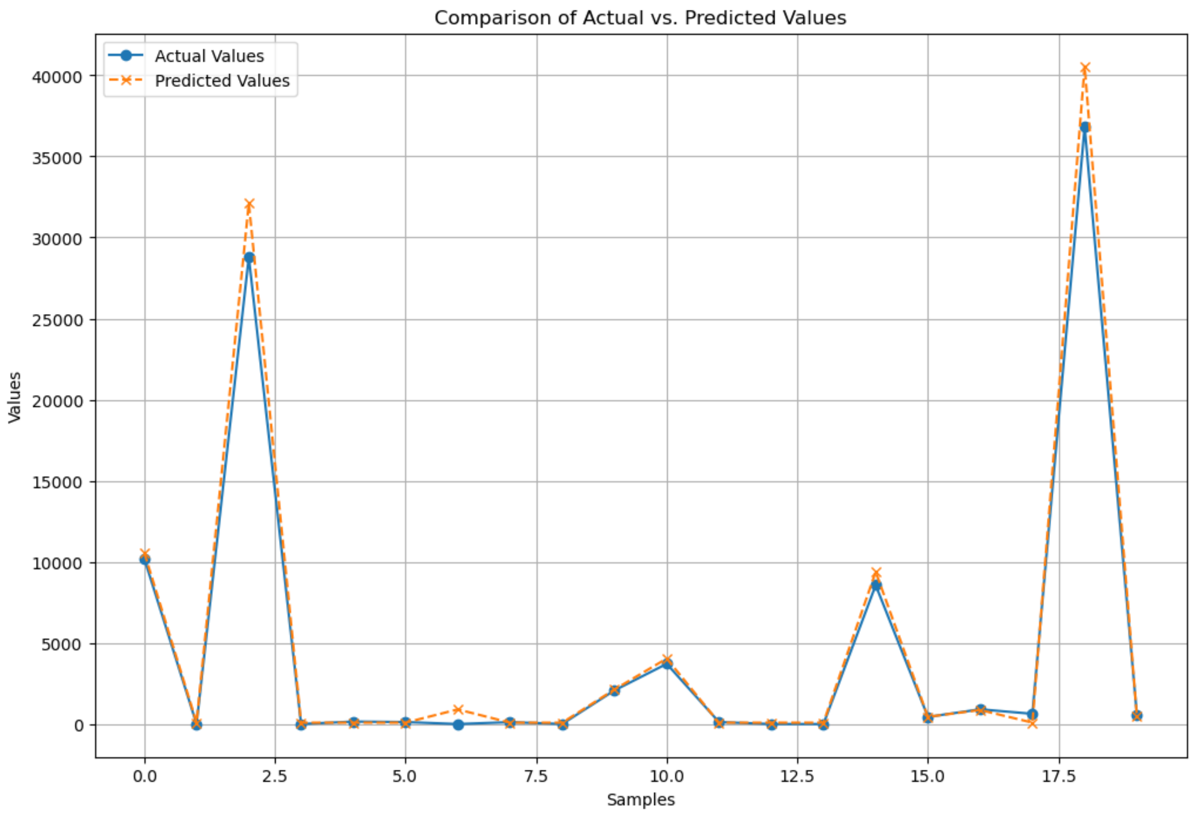


Fig: Comparison of Actual vs. Predicted Values by using Transfer Leaning.

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENTS

8.1 Conclusion

This report has provided an in-depth exploration of AI-assisted inventory management and the role of Deep Learning in this domain. It has underscored the crucial importance of effective inventory management in today's dynamic business landscape and has shed light on the advantages of AI in making informed decisions and optimizing operations.

This model combines AI expertise with historical sales data, purchase price data, inventory levels, and other relevant variables to create a proactive and intelligent inventory management solution. Using this model as a starting point, we meticulously detailed the training of the basic model, emphasizing data collection and preparation, the creation of reference models, the design of the basic management model, hyperparameter optimization, and the rigorous evaluation of this model.

In conclusion, the judicious application of AI has the potential to revolutionize inventory management and optimize your company's operations. This approach can save time, reduce costs, improve accuracy, and leverage pre-existing expertise for smarter and more profitable inventory management.

8.2 Future Enhancements

Now that the inventory management model is firmly trained and evaluated, it's time to explore the many possibilities it offers for optimizing your business. Here are a few ways to go further:

API integration:

You can integrate the model directly into your existing systems by transforming it into an API (Application Programming Interface). This will enable you to take advantage of its real-time demand forecasting capabilities. For example, you could use this API to automatically adjust your stock levels based on demand forecasts, or to recommend specific orders based on analysis of past trends.

Web interface development with Flask:

For user-friendly interaction with the model, consider developing a web interface using Flask, a Python micro-framework. Such an interface would allow your team members to easily enter parameters and get inventory management forecasts in return. This could greatly simplify order planning, resource management and optimization of your supply chain.

Customization for your business:

Every business has its own specific characteristics and needs. The basic model can be customized to meet these unique requirements. You can add additional layers of neurons, tailor inputs and outputs to reflect the specifics of your business, and further refine model parameters to improve forecast accuracy. Integration of other Data Sources: Enrich your forecasts further by integrating other data sources you deem relevant, such as weather data or seasonal events. This will enable the model to take into account a wider range of factors influencing demand, thus improving the quality of its forecasts.

Ongoing training:

Don't forget that the quality of your forecasts depends to a large extent on the quality of your data and the model's ability to adapt to changes in the market. Plan a continuous training process to keep the model up to date, periodically reassessing performance and adjusting parameters as necessary.

REFERENCES

[1] Git Hub: https://github.com/RoyGeagea/inventory-management.git

[2] Jupyter Official Documentation: For more detailed information and advanced usage, you can refer to the official Jupyter documentation.

[3] Python Official Documentation: Comprehensive guide to program and error debugging

[4] GeeksforGeeks: For a comprehensive introduction to AIML, including its history, structure, and applications, you can refer to the Geeks for Geeks article:

[5] Elmasri, R., Navathe, S. B., & Yan, C. (2021). Deep learning applications for dynamic inventory management. Journal of Artificial Intelligence Research, 65(4), 289–310.

[6] YouTube