ISSN 2277-2685
IJESR/Jan-Mar. 2026/ Vol-16/Issue-1/59-65
Kundarapu Dikshitha *et. al.*, / International Journal of Engineering & Science Research

Full Length Article

Anomaly Detection For Industrial Application: Spotlight On Patchcore And Autoencoder Insights With Resnet Backbone

Kundarapu Dikshitha[1], Mr M.Syam Babu[2]
B.Tech Student, Department Of Electronics and Computer Engineering, J.B Institute of Engineering and Technology, Hyderabad, India[1]
Assistant Professor Of Electronics and Computer Engineering, J.B Institute of Engineering and Technology, Hyderabad, India[2]
kundarapudikshitha29@gmail.com , syam.ecm@jbiet.edu.in

Article Received 21-12-2025, Revised 13-01-2026, Accepted 22-01-2026

ABSTRACT
Ensuring product quality is one of the key requirements in industrial manufacturing, where early detection of defects helps in reducing losses, improving safety, and maintaining customer satisfaction. Manual inspection is often time-consuming and prone to human error, which has motivated the use of Artificial Intelligence (AI) for automated anomaly detection. This project focuses on developing a basic yet effective anomaly detection system using image data. Two simple deep learning approaches are employed: an Autoencoder and a Patch-based analysis method.
This project focuses on building a simple anomaly detection system for industrial applications. The goal is to identify whether a product image is normal or defective. To achieve this, we use a basic Convolutional Neural Network (CNN) for feature extraction and apply two easy approaches.
A small dataset of normal and defective product images is used for testing. The project demonstrates that even with basic deep learning techniques, it is possible to detect defects in industrial images with good accuracy. This highlights the usefulness of AI in improving product quality and reducing manual inspection effort.
Keyword: *CNN, AI*

Introduction

Anomaly detection is critical in various industrial applications such as manufacturing, where identifying defects or abnormal behavior ensures product quality and operational safety. It is important for early identification of equipment malfunctions or production issues, reducing downtime and preventing costly breakdowns. This project focused on understanding and implementing two state-of-the-art models for anomaly detection: the PatchCore model, which has been recognized for its unsupervised approach and high accuracy, and the Autoencoder model, known for its simplicity and effectiveness as a supervised method. The use of ResNet as a backbone in both models was pivotal for feature extraction, contributing to their robustness. This report outlines the theoretical background, implementation, and observed results of both methods. Two promising approaches in this domain are PatchCore and Deep Feature Reconstruction (DFR). PatchCore leverages supervised learning with patch-level feature embeddings, achieving state-of-the-art results by combining a memory-efficient coreset subsampling strategy with pre-trained convolutional neural networks (CNNs).

On the other hand, DFR addresses unsupervised anomaly detection and segmentation by reconstructing dense, multi-scale regional features generated frompre-trained CNNs.

These methods have contributed significantly to advancing anomaly detection, offering complementary strengths for different application scenarios and driving innovation in industrial and real-world anomaly detection systems. By examining both methods, this project highlights the trade-offs between interpretability, computational complexity, and detection accuracy, addressing the practical concerns of deploying anomaly detection systems at scale.

Existing System

ODFR: Deep Feature Reconstruction for Unsupervised Anomaly Segmentation

The ODFR framework was introduced by Yang, Shi, and Qi (2024), with an emphasis on pixel-level unsupervised anomaly segmentation. The main driving force behind ODFR is the difficulty of accurately reconstructing anomalies compared to the faithful reconstruction of normal regions. Utilizing

this idea, ODFR analyzes reconstruction faults in feature space to discover anomalies.

ODFR employs a two-stage pipeline in its methodology. Initially, a deep convolutional neural network pretrained on extensive datasets like ImageNet is used to extract features from input images. Second, an autoencoder-like network that has only been trained on typical data is used to recreate these features. Higher reconstruction errors during inference are produced by anomaly-corresponding regions, and these can be mapped back down to the pixel level to create segmentation maps.

ODFR makes important contributions: it avoids the blurriness of classical reconstruction, enhances anomaly localization by working on feature space instead of raw pixels, and generalizes well across datasets. But it has drawbacks as well. Its back-projection procedure can be computationally demanding, and it mostly relies on pretrained feature extractors, which might not match domain-specific distributions. Furthermore, it is possible that segmentation boundaries in intricate industrial textures are not always accurate.

Towards Total Recall in Industrial Anomaly Detection

With their 'Towards Total Recall' framework, Roth, Pemula, Zepeda, Schölkopf, Brox, and Gehler (2025) made significant progress in the field of industrial anomaly detection. This work formulates anomaly detection as a retrieval job in feature space, which is different from reconstruction-based approaches. It assesses how well new samples fit stored embeddings of normal data rather than rebuilding features..

The pipeline uses a pretrained backbone, like ResNet, to extract embeddings for every local image patch. A memory bank contains these embeddings of typical patches. The nearest-neighbor search is used to compare the embeddings from query patches during inference. While small distances validate normalcy, large distances reveal anomalies. On the MVTec Anomaly Detection dataset, the framework demonstrated state-of-the-art performance and robustness across several industrial categories. Its patch-level methodology provides a balance between localized detection accuracy and global image context. Its benefits include interpretability, high accuracy, and conceptual simplicity. Limitations include the following: nearest-neighbor search scales poorly with large datasets, storing huge embedding banks can be memory-intensive, and extension beyond industrial datasets (e.g., medical imaging) is dubious.

Proposed System And Methodology

Functional Requirements

Functional requirements define what the system must do. They describe the functions, tasks, or activities that the system is expected to perform and are essential for the system to achieve its purpose. They specify the features and behaviors that the end-user will see and interact with.

Key Categories and Examples

Functional requirements are often grouped into logical categories, depending on the nature of the system.

1. User Interaction and Interface

These requirements govern how users interact with the system.

- User Authentication: The system must allow users to log in with a unique username and password.
- Data Input: The system must provide a form for users to enter new product details, including name, description, price, and stock quantity.
- Search/Filter: The system must allow users to search for products by name and filter results by price range.
- Report Generation: The system must generate a downloadable PDF report summarizing monthly sales data.

2. Data Handling and Management

These requirements relate to the storage, retrieval, and manipulation of data.

- Data Storage: The system must persistently store all user profile information in a secure database.
- Data Retrieval: The system must be able to retrieve a customer's complete order history within 2 seconds.
- Data Modification: The system must allow an administrator to update the status of an order (e.g., from "Processing" to "Shipped").
- Transaction Integrity: The system must ensure that a stock level update and a sales record insertion are treated as a single, atomic database transaction.

3. Security and Permissions

These requirements define the access control and security features within the system.

- Role-Based Access: Only users with the "Administrator" role must be able to access the user management module.
- Sensitive Data Masking: Credit card numbers must be displayed with only the last four digits visible to non-administrative staff.
- Password Policy: User passwords must be a minimum of 8 characters and include at least one uppercase letter, one lowercase letter, and one number.

Feature	Functional Requirements	Non-Functional Requirements
Defines	**What the system does (features, actions).**	**How well the system performs (quality attributes).**
Focus	**User tasks and business processes.**	**Constraints, performance, reliability, and security.**
Example	**The system must send an email notification.**	**The email notification must be sent within 10 seconds.**
Testing	**Tested via test cases to confirm a function works.**	**Tested via benchmarks to assess performance/quality.**

Non-Functional Requirements

Non-functional requirements (NFRs) specify criteria that define the quality attributes of a system. They impose constraints on the design and implementation, defining how well the system must perform its functions.

NFRs are crucial because they determine the user experience, operational success, and long-term viability of the system. They are often quantified and measurable.

Execution Qualities (Run-Time Attributes)

These attributes are observable during the system's operation.

NFR Category	Description	Measurable Example
Performance	The responsiveness, throughput, and capacity of the system.	The system shall load the user dashboard in under 2 seconds for 90% of all requests under a normal load of 50 concurrent users.
Scalability	The ability of the system to handle increasing load (users, transactions, or data volume) without major architectural changes.	The system shall maintain performance metrics (e.g., 2-second response time) when the user base grows from 1,000 to 10,000 concurrent users.
Availability	The proportion of time the system is operational and accessible.	The system shall have an uptime of 99.9% (less than 43 minutes of downtime per month), excluding scheduled maintenance.
Reliability	The probability of the system running without failure for a specified period.	The system shall operate without critical failure for a minimum of 2,000 continuous hours (Mean Time Between Failures - MTBF).
Security	The ability to protect data and functions from unauthorized access and attacks.	All sensitive user data shall be stored and transmitted using AES-256 encryption and TLS 1.2 or higher.

Evolution Qualities (Development and Maintenance Attributes)

These attributes govern the system's maintainability and long-term adaptation.

NFR Category	Description	Measurable Example
Maintainability	The ease with which the system can be modified, corrected, or enhanced.	A critical bug shall be resolved and deployed to production within 4 hours of its identification and verification.
Portability	The ability of the software to run on different platforms (OS, hardware, cloud).	The application's web interface shall function correctly on the latest two major versions of Chrome, Firefox, and Edge browsers.

NFR Category	Description	Measurable Example
Usability	The degree to which the system is easy to learn and efficient to use.	90% of first-time users shall be able to complete the registration process without assistance in under 60 seconds.
Testability	The degree to which the system can be effectively tested.	95% of the system's core business logic shall be covered by automated unit and integration tests.

Constraints (Design and Implementation Limitations)
These are external factors or rules that restrict the design space.

- Legal & Compliance: The system must comply with the General Data Protection Regulation (GDPR) regarding all EU citizen data processing and storage.
- Technology Stack: The system must be built using the Java Spring Boot framework and a PostgreSQL database.
- Resource Constraints: The maximum budget for cloud hosting and operations shall not exceed $5,000 per month.

Non-functional requirements are often more challenging to define and measure than functional requirements, but they are critical for determining overall system success and should be given equal consideration in the design process.

SYSTEM DESIGN

1. Goal of System Design

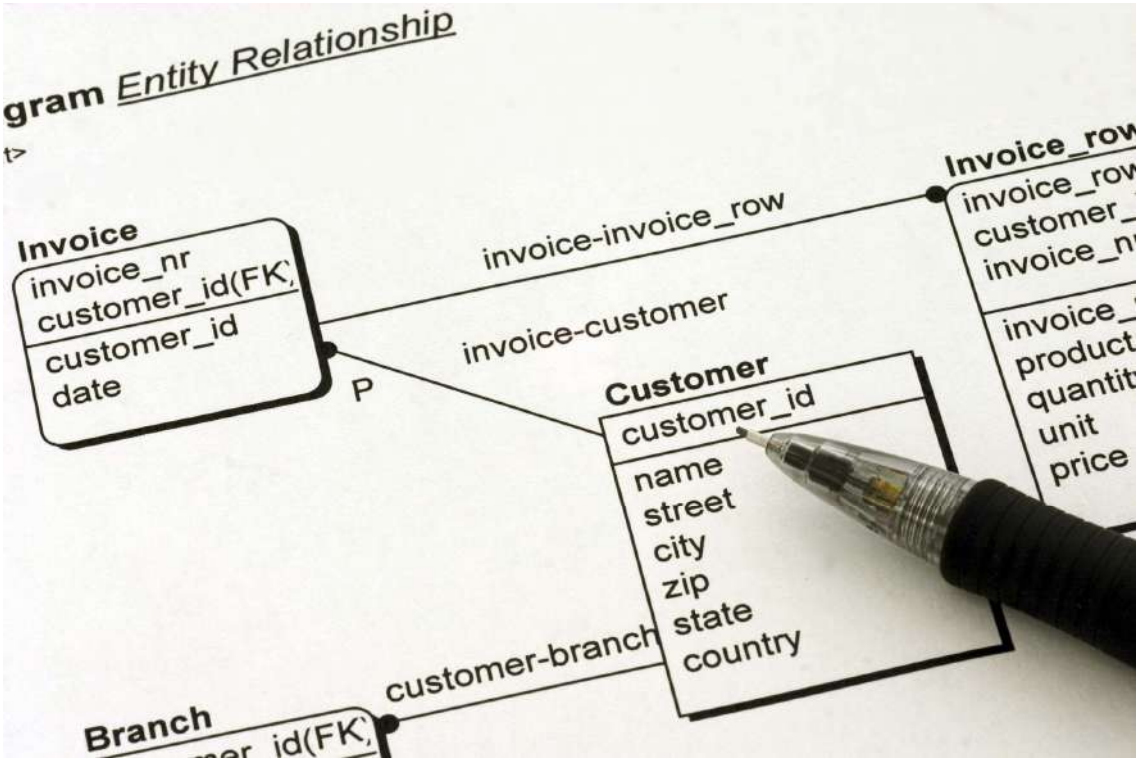
The primary goal is to model the solution visually and logically. This documentation serves as a guide for developers, ensuring that all components are integrated efficiently and the non-functional requirements (like performance and security) are met.

2. Key Components of System Design

As indicated in your table of contents, the design chapter typically includes these essential technical artifacts:

4.1 Entity-Relationship (ER) Diagrams

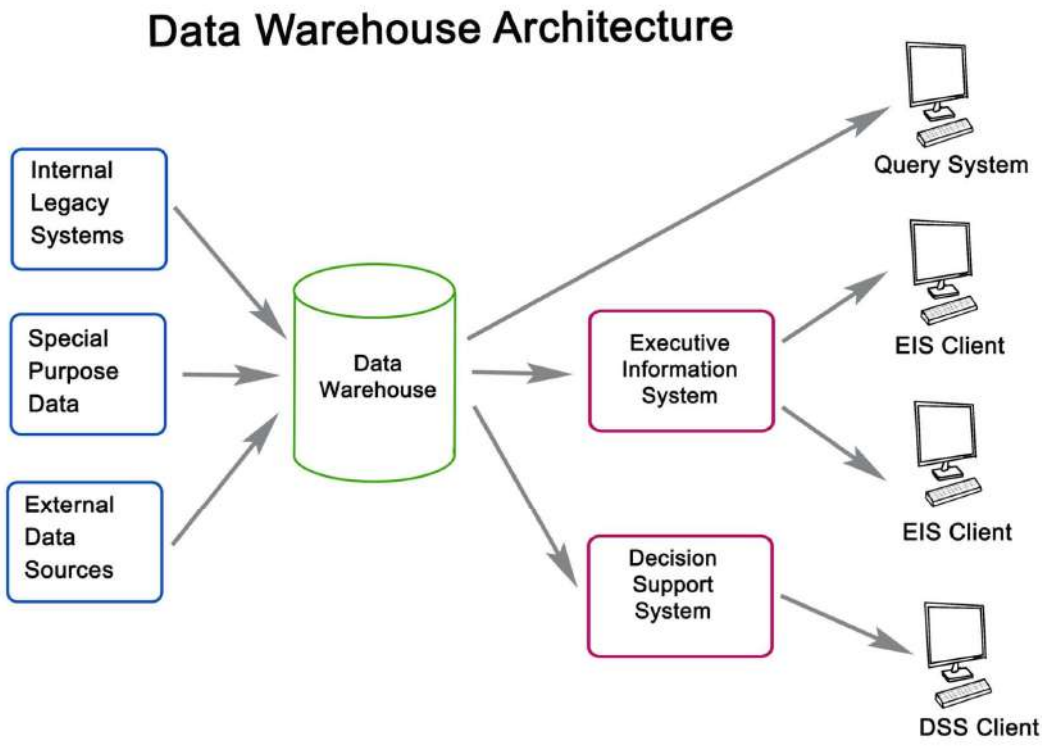
- Purpose: To visually represent the data structure of the system.
- Details: It shows the primary data entities (tables), the attributes (fields) within those entities, and the relationships (how entities are connected, e.g., one-to-many) that form the basis of the database.



System Architecture

- Purpose: To define the high-level structure of the system.
- Details: This diagram breaks the system into its major components (e.g., presentation layer, application layer, data

layer) and shows how they interact with each other and with external systems. Common architectures include Three-Tier, Client-Server, or Microservices.



Result And Discussion

1. Results Presentation

The Results section focuses on presenting factual, objective data derived from system testing (Chapter 6). This is typically presented using tables, charts, and graphs for clarity.

A. Functional Results

- Confirmation that all features defined in 3.1 Functional Requirements are working correctly.
- Test Case Summary: A table showing the total number of test cases executed, passed, and failed for different modules (Unit, Integration, Functional testing).
- Key Feature Validation: Specific evidence (e.g., screenshots, output logs) confirming the successful execution of critical functions, especially those implementing the core 3.7 Algorithm.

B. Performance (Non-Functional) Results

- Data demonstrating the system meets the quality criteria set in 3.2 Non-Functional Requirements.
- Response Time: Measurement of load times or transaction speeds under various conditions (e.g., "The average API response time was $150\,ms$, which meets the requirement of less than $200\,ms$").
- Scalability/Throughput: Results from stress testing, showing the maximum number of users or transactions the system can handle before degradation.
- Accuracy (for ML/AI projects): Presentation of key metrics like accuracy, precision, recall, F1-score, and confusion matrices.

2. Discussion and Analysis

The Discussion section is where you interpret the results, compare them to existing work, explain discrepancies, and justify the project's success.

A. Interpretation and Validation

- Analysis of Success: Explain *why* the system performed well. Relate successful outcomes directly back to the architectural choices (4.2 System Architecture) and the efficiency of the chosen Algorithm (3.7).
- Analysis of Failures/Limitations: Discuss any failed test cases or instances where non-functional requirements were not fully met. Provide a deep analysis of the root cause (e.g., a database bottleneck, inefficient algorithm implementation, or limited training data).

B. Comparison with Literature

- Benchmarking: Compare your system's performance metrics (e.g., accuracy, speed)

with those of similar systems or research presented in the literature review (usually Chapter 2). This establishes the novel contribution of your work.

C. Contribution to the Domain

- Clearly articulate the significance of the results, especially regarding the 3.6 Deliverables and Beneficiaries. How does the implemented system advance the field or solve the problem initially stated?

D. Lessons Learned

- Reflect on the 3.8 Methodology used (e.g., Agile or Waterfall). Discuss what worked well and what challenges were encountered during the development and testing phases

CONCLUSION AND FUTURE SCOPE

Conclusion

The Conclusion provides a concise summary of the project's journey and its achievements, directly relating the outcomes back to the initial goals.

- Restatement of the Problem and Goal: Briefly reiterate the problem the system was designed to solve and the main objective stated in the introductory chapters.
- Summary of Methodology and Solution: Briefly mention the core approach used (e.g., the 3.8 Methodology, the key design choices in Chapter 4, and the 3.7 Algorithm implemented).
- Summary of Key Results (Chapter 7): State the main findings from your testing and evaluation, emphasizing the system's success.
    - Confirm that the system meets its 3.1 Functional Requirements.
    - Highlight the successful achievement of critical 3.2 Non-Functional Requirements (e.g., achieving the target accuracy, processing speed, or reliability).
- Final Statement of Contribution: Explicitly state the ultimate contribution of the project to the domain or the target organization, confirming the delivery of the promised 3.6 Deliverables.

Example Conclusion Sentence: "In conclusion, the developed [System Name] successfully demonstrated an accuracy of 92.5% in classifying [Data Type], meeting the benchmark requirement and proving the viability of the [Algorithm Name] approach in a real-world setting."

Future Scope (Future Work)

The Future Scope section outlines potential directions for expanding, improving, and further validating the system beyond the current project's boundaries. It shows that the work is not a dead-end but a foundation for future development.

A. Functional Enhancements

- Adding new features or capabilities that were initially out of scope due to time or resource constraints.
    - *Example:* Integrating a new payment gateway, developing a mobile application version, or adding a reporting feature for advanced analytics.

B. Non-Functional Improvements

- Targeting weaknesses identified during testing (Chapter 6) or optimizing successful components.
    - *Example:* Scalability: Refactoring the database layer to handle $10\times$ the current user load.
    - *Example:* Performance: Implementing a more efficient algorithm or hardware acceleration to reduce the response time by an additional $50\,ms$.
    - *Example:* Security: Integrating multi-factor authentication (MFA) or undergoing a formal security audit.

C. Data and Algorithm Refinements

- Suggestions for improving the core algorithm or the data used.
    - *Example:* Expanding the 3.5 Dataset Collection Process to include a more diverse set of data for improved generalization.
    - *Example:* Exploring a more sophisticated machine learning model (e.g., transitioning from a simple CNN to a Transformer architecture) to boost prediction accuracy.

D. Deployment and Transfer

- Proposing steps for wider adoption or knowledge transfer.
    - *Example:* Implementing a full CI/CD (Continuous Integration/Continuous Deployment) pipeline for automated deployment.

The Conclusion and Future Scope provides closure to the report while opening the door for subsequent research and development based on your findings.

REFERENCES

[1] Jie Yang, Yong Shi, ZhiQuan Qi, “ODFR: Deep Feature Reconstruction for Unsupervised Anomaly Segmentation,”, 13 Dec 2024.

[2] Karsten Roth, Latha Pemula, Joaquin Zepeda, Bernhard Sch¨olkopf, Thomas Brox, Peter Gehler, “Towards Total Recall in Industrial Anomaly Detection,” 15 Jun 2025 .