JÖNKÖPING UNIVERSITY
*School of Engineering*

# Construction of a Metal Defect Knowledge Graph Using Deep Learning and Text Mining

**PAPER WITHIN** *Computer Science*
**AUTHORS:** *Yiyuan Xia*
**TUTOR:** *Maria Magdalena Hedblom*
**JÖNKÖPING** *June 2025*

This exam work has been carried out at the School of Engineering in Jönköping in the subject area Computer Science. The work is a part of the two-year Master of AI in Engineering programme. The authors take full responsibility for opinions, conclusions and findings presented.

| | |
|---|---|
| Examiner: | Vladimir Tarasov |
| Supervisor: | Maria Magdalena Hedblom |
| Scope: | 7,5 credits |
| Date: | 2025-06-13 |


| Mailing address: | Visiting address: | Phone: |
|---|---|---|
| Box 1026<br>551 11 Jönköping | Gjuterigatan 5 | 036-10 10 00 (vx) |

**Abstract**

This project combines deep learning technology with knowledge representation methods to propose a comprehensive defect automatic detection and interpretation process. The system first extracts and classifies images from PDF documents, and then trains a convolutional neural network (CNN) to identify and classify product defects with high accuracy. To enhance interpretability, a domain-specific knowledge graph (KG) is constructed to represent defect types and their potential failure mechanisms. This structured representation enables users to trace the root cause of each prediction to better understand and control quality. To improve user accessibility and interactivity, the process integrates a system based on the front-end (HTML) and back-end (Flask API). Users can upload defect images through the integrated system and receive automatic classification and natural language explanation of defects from the knowledge graph generated by Neo4j, including the definition of the defect and possible causes. By combining visual recognition, semantic reasoning, and natural language generation, this method provides an intelligent and scalable solution for efficient defect analysis in industrial and manufacturing environments.

**Keywords—** Defect Detection, Convolutional Neural Network (CNN), Knowledge Graph, Neo4j, Natural Language Generation, Explainable AI, Image Classification, Intelligent Manufacturing.

# Contents

**1 Introduction** **1**
1.1 Research Questions and Objectives . . . 1
1.2 Scope and Delimitations . . . 2

**2 Background** **3**
2.1 Related Work . . . 3
2.2 Research Foundation . . . 4
2.3 Research Gap . . . 5

**3 Research Methods** **6**
3.1 PDF-Based Image and Context Extraction . . . 7
3.1.1 Book-Structured PDF Extraction . . . 7
3.1.2 Multi-File PDF Folder Extraction . . . 8
3.1.3 CSV Metadata Generation . . . 8
3.2 CNN-Based Defect Classification . . . 8
3.3 Knowledge Graph Construction . . . 9
3.4 Web-Based User Interaction Interface . . . 10

**4 Case study and Results** **11**
4.1 CNN Model Training and Classification Results . . . 11
4.1.1 Dataset and Preprocessing . . . 11
4.1.2 Model Architecture and Training . . . 11
4.1.3 Performance Evaluation . . . 12
4.1.4 Classification Output and CSV Export . . . 12
4.2 PDF-Based Image and Text Extraction Results . . . 13
4.3 Knowledge Graph Reasoning and Visualization Results . . . 16
4.3.1 Graph Structure and Design . . . 16
4.3.2 Query Results and Semantic Retrieval . . . 17
4.3.3 Relationship Properties and Text Integration . . . 17
4.3.4 Coverage and Result Quality . . . 18
4.4 HTML-Based User Interaction Results . . . 18
4.4.1 Interface Layout and Workflow . . . 18

**5 Discussion** **20**
5.1 System Behavior and Response Quality . . . 20
5.2 CNN Model Effectiveness and Practical Implications . . . 21
5.3 Image and Text Extraction from Unstructured Documents . . . 21
5.4 Semantic Integration via Knowledge Graphs . . . 22
5.5 System Integration and Real-Time Usability . . . 22
5.6 Limitations and Opportunities for Future Work . . . 23

**6 Conclusion** **24**

**Appendices** **28**

**A Appendix A CNN Model Configuration and Parameters** **28**

**B Knowledge Graph Triples and Queries** **29**

**C PDF-Based Dataset Construction Samples** **29**

**D Flask API Interface and Output Format** **30**

# 1 Introduction

In modern industrial manufacturing, the ability to detect defects reliably is critical to maintaining product quality, reducing waste, and ensuring safety. Traditional inspection methods—often based on human visual inspection or rigid rule-based systems—are not only labor-intensive but also prone to inconsistency and error. As manufacturing processes become increasingly complex and the demand for scalability grows, there is a clear need for intelligent systems that can both recognize defects and explain their causes in a transparent and interpretable way.

Despite advances in computer vision and machine learning, current automated defect detection systems still face key limitations. Many rely on clean, well-annotated datasets that are difficult to obtain in real-world production environments. Others treat defect classification as a purely statistical task, producing predictions without offering insight into what a defect means or how it might have occurred. This lack of interpretability limits the usefulness of such systems for engineers and quality control specialists who need to understand not just what happened, but why.

This project addresses these challenges by developing a web-based defect detection and interpretation system that combines visual recognition with semantic reasoning. At its core is a convolutional neural network (CNN) trained to classify six types of surface defects in product images. The model is accessed through a Python-based Flask web server, which exposes RESTful APIs to a front-end interface built with HTML and JavaScript. Users can upload defect images through the web interface and receive classification results within seconds.

To add domain-specific meaning to these predictions, the system integrates a knowledge graph built on Neo4j, a popular graph database platform.[1] The graph encodes structured information about each defect type, including definitions, likely causes, and morphological characteristics. When a prediction is made, the system queries the graph to retrieve a human-readable explanation of the defect, helping users understand not only what was detected but also the context in which such defects typically occur.

Communication between the Flask backend and the Neo4j database is handled using the Bolt protocol,[2] ensuring fast and secure data exchange. The architecture supports modular integration between the CNN classifier, the semantic graph, and the user interface.

By combining image classification, semantic reasoning, and user-facing design into a unified platform, this project contributes a prototype for explainable defect detection. It bridges the gap between visual AI systems and knowledge-based engineering tools, offering a more transparent and actionable approach to quality control in manufacturing.

## 1.1 Research Questions and Objectives

This project aims to develop an intelligent defect detection and explanation pipeline by integrating computer vision, knowledge representation, and natural language technologies. The system is designed to process complex PDF documents by extracting embedded images and automatically retrieving associated defect-related textual content such as labels and definitions. These images are then classified using a Convolutional Neural Network (CNN), and the resulting labels are mapped to a Neo4j-based Knowledge Graph (KG) to provide structured, machine-interpretable semantics. By combining deep learning, semantic reasoning, and document analysis, the pipeline not only improves the accuracy of defect classification but also streamlines the process of building defect-specific datasets from unstructured sources like technical PDFs. The end goal is to bridge the gap between raw visual input and contextual industrial

[1] https://neo4j.com/
[2] https://boltprotocol.org/

knowledge, delivering a complete, interpretable solution for quality inspection and failure analysis. The following research objectives guide the development of the system:

- **Research Objective 1.** Implementation of a PDF parsing module capable of extracting defect images along with surrounding textual descriptions (labels, definitions).
- **Research Objective 2.** Construction of a domain-specific Knowledge Graph to represent defect definitions, relationships, and root causes.

Two research questions must be addressed to achieve the objectives of this project. The first is related to the theoretical construction of a neuro-symbolic AI system, while the second assesses the value of such an AI system practically.

In order to achieve the goals of this project, two research problems must be solved. The first problem is related to the extraction of image data and text extraction of related definitions, and the second problem is related to model prediction results for subsequent queries. These two research problems are stated as follows:

- **Research question 1.** How to quickly extract images and their definitions for CNN model training and kg generation when defect image data is insufficient
- **Research question 2.** How to combine CNN model predictions with user-uploaded images to give prediction results for Neo4j to use to generate KG.

## 1.2 Scope and Delimitations

This project aims to design and implement an intelligent process that combines PDF image extraction, deep learning-based classification, knowledge graph reasoning, and natural language generation techniques to automate defect detection and interpretation. Parsing PDF documents to extract images related to defects is the preparation of the dataset. The results obtained by training CNN in the absence of image data lack accuracy. As supplementary data, images and their associated textual contexts such as tags and definitions are extracted from existing books and PDF files and stored and generated into corresponding csv files. Convolutional neural networks (CNNs) are used to classify existing images and link the predicted tags to a Neo4j-based knowledge graph that stores structured information about defect types and failure mechanisms. The complete process is deployed in a web-based environment using HTML and JavaScript for the front-end and Python Flask for the back-end, using the Bolt protocol for database communication.

However, the scope of this study is limited by several factors. PDF parsing is limited to local extraction of images and nearby text, and full document semantic understanding is not performed. The deep learning classification component is trained on a restricted dataset containing limited defect types, which may affect its generalization ability in a wider range of applications. The knowledge graph is manually constructed for a specific domain and does not currently support large-scale or cross-domain expansion. In addition, the system only generates English explanations and does not support multilingual output or custom fine-tuning of language models. Real-time optimization for industrial deployment is also not within the current focus. These limitations define the practical boundaries of this research, allowing it to focus on demonstrating a fully functional, explainable prototype system within the scope of academic research.

# 2 Background

The identification of defects in images is a problem that needs to be solved by both production and customers, especially in the fields of smart manufacturing and quality control. Automated defect detection by AI systems can reduce reliance on manual inspection while improving the accuracy, speed, and consistency of identifying critical surface and structural faults. However, the potential in this field goes far beyond simple image classification, as defect understanding is directly related to broader AI capabilities, such as semantic reasoning, explainability, industrial decision support, and human-machine collaboration. Therefore, various studies have proposed different methods to improve state-of-the-art defect recognition, annotation, and interpretation methods. This section reviews selected implementations of four key areas (deep learning visual classification, PDF-based dataset construction, knowledge graph reasoning, and natural language generation) and outlines how they guide and drive the technical design of this project.

## 2.1 Related Work

This section reviews existing research across three major areas: deep learning methods for defect classification, knowledge graph-based semantic reasoning for intelligent anomaly diagnosis, and the integration of large language models (LLMs) for natural language explanation in manufacturing contexts.

**Deep Learning for Defect Classification:** Deep learning, especially convolutional neural networks (CNNs), has demonstrated significant success in automated visual defect detection across industrial settings. Liu et al., 2023 applied CNNs to classify defects in additive manufacturing (AM) images, achieving over 98% validation accuracy by combining transfer learning and active learning to reduce the labeling burden while maintaining model performance. Similarly, Lokhande et al., 2023 proposed DeepInspect, a multi-network approach combining CNNs, RNNs, and GANs to detect evolving and complex defect patterns in manufacturing images, showing improvements in adaptability and robustness. Martín-Ramiro et al., 2024 introduced a Tensor CNN (T-CNN), which significantly reduced parameter count and training time while maintaining comparable classification accuracy to standard CNNs, offering an efficient solution for real-time quality control on production lines.

A survey by Bhatt et al., 2021 further categorized image-based surface defect detection methods by task type, learning mode, and architecture, highlighting CNNs and transfer learning as current state-of-the-art in supervised defect classification tasks.

**Knowledge Graphs for Defect Representation and Reasoning:** Knowledge graphs (KGs) have gained momentum as tools for semantically modeling and reasoning over complex fault and equipment relationships in intelligent manufacturing systems. Wang et al., 2024 provided a comprehensive review on the role of KGs in smart manufacturing, emphasizing their ability to manage heterogeneous data, support predictive maintenance, and enhance explainability through semantic structuring and graph-based inference.

More specifically, Cai et al., 2024 proposed a multi-level KG framework for rotating machinery fault diagnosis, integrating structured and unstructured data with Bayesian reasoning to achieve fault localization even with missing features, outperforming traditional rule-based diagnostics in accuracy.

Yang et al., 2022 explored the deployment of defect KGs in multi-cloud IoT settings. Their system, GRULR, enabled distributed logic-rule-based reasoning, improving fault detection accuracy across heterogeneous data centers and equipment types.

**Natural Language Explanation with Large Language Models:** To increase accessibility and interpretability of model outputs, recent studies have integrated LLMs into manufacturing AI systems. Pak and Farimani, 2025 developed AdditiveLLM, a fine-tuned transformer-based language model that trans-

lates process parameters into predicted defect regimes in AM processes, demonstrating that LLMs can effectively bridge technical defect knowledge and human comprehension. Although their model focused on parameter-based inference, it highlights the potential of LLMs like those in Ollama frameworks to act as user-facing interfaces, translating structured defect data into natural explanations.

## 2.2 Research Foundation

This subsection presents foundational studies that inform the design and implementation of the current defect detection and explanation system. The project draws upon key advances in deep learning for visual inspection, document understanding, knowledge graph reasoning, and the integration of natural language explanations using modern AI models.

A major component of this project builds on the success of convolutional neural networks (CNNs) in defect classification for manufacturing. Liu et al., 2023 introduced a CNN-based framework for classifying additive manufacturing defects using a small labeled dataset and demonstrated the effectiveness of transfer learning in this domain. Similarly, Lokhande et al., 2023 developed DeepInspect, which integrates CNNs, RNNs, and GANs to improve robustness and handle evolving fault types through synthetic data augmentation. Expanding on this, Martín-Ramiro et al., 2024 proposed a Tensor-CNN (T-CNN) approach that significantly reduces the number of model parameters and training time while achieving equivalent accuracy, making it well-suited for scalable real-time defect detection on manufacturing lines

In addition to image classification, the project addresses the challenge of creating training datasets from technical documentation. Chen et al. (2016) offered a comprehensive review of fault detection methods in transmission systems. They emphasized the importance of structured data extraction from documents to support classification and fault localization tasks. Bhatt et al., 2021 surveyed deep learning approaches for surface defect detection and advocated for hybrid pipelines that leverage both image data and associated metadata, which aligns with this project's document-aware image label extraction strategy.

This project relies on structured knowledge representation using knowledge graphs (KGs) to enhance semantic reasoning and interpretability. Wang et al., 2024 emphasized the role of KGs in smart manufacturing, outlining how they enable causal tracing and semantic querying of equipment states and failure mechanisms Cai et al. (2024) proposed a multi-layer KG for equipment fault diagnosis that integrates Bayesian inference and machine learning over relational paths to locate fault sources even with missing input features Yang et al. (2022) developed a defect KG framework in a multi-cloud IoT setting, introducing GRULR—a logic rule-based reasoning engine that supports distributed deployment and efficient defect inference across diverse data sources

Finally, for natural language explanation, this work is inspired by Pak and Farimani (2025), who introduced AdditiveLLM—a large language model capable of predicting and describing defects in additive manufacturing based on structured process parameters. Their model highlights how domain-specific LLMs can improve accessibility and interpretability for engineers and non-expert users alike. In this project, we leverage similar LLM integration using Ollama to translate KG outputs (defect definitions and causes) into natural language explanations.

Together, these studies form the technical foundation of this research and validate the design of an end-to-end, explainable defect detection system combining document parsing, image classification, knowledge graphs, and language generation.

## 2.3 Research Gap

Despite the great progress in visual defect detection, semantic modeling, and natural language techniques, existing research often views these components in isolation, resulting in a gap in the integrated and interpretable defect analysis workflow. Convolutional neural networks (CNNs) have performed well in surface defect classification in various manufacturing applications. For example, Liu et al. (2023) demonstrated high classification accuracy in additive manufacturing using a CNN enhanced by transfer learning and an active data selection strategy. While Martín-Ramiro et al. (2024) introduced a tensor-based CNN architecture that improves training efficiency without sacrificing accuracy. However, these methods are usually built on well-labeled image datasets, which limits their applicability in real industrial settings, as defect data are often embedded in unstructured documents, such as PDF manuals or inspection records.

Although the potential of unstructured documents as a source of defect information has been noted, the task of automatically extracting defect-related images and their associated textual descriptions remains underexplored. Bhatt et al. (2021) highlight the value of pairing image data with contextual metadata to improve classification reliability, but few systems can automate this process from raw technical documentation. This gap is particularly significant in industrial settings, where manual labeling is costly and domain knowledge is embedded in text and visual media.

Meanwhile, knowledge graphs (KGs) have emerged as a powerful tool for representing semantic relationships between defect types, causes, and equipment components. Wang et al. (2024) and Cai et al. (2024) highlight the benefits of integrating KGs into smart manufacturing for tasks such as fault diagnosis and semantic querying. However, most implementations rely on static graph structures and lack integration with real-time prediction processes. The disconnect between visual classification output and KG reasoning hinders dynamic contextualization and limits interpretability, especially in environments where conditions change rapidly or information is incomplete.

Furthermore, the rise of large language models (LLMs) in recent years has enabled new forms of user interaction through natural language interpretation. Pak and Farimani (2025) showed that a fine-tuned LLM can successfully predict and describe defect states in metal additive manufacturing based on structured process inputs. However, these models have rarely been used to interpret knowledge graph outputs, and current approaches typically rely on static prompts or templates rather than dynamically generated semantic content. In the manufacturing domain, the challenge of generating faithful, context-specific explanations from graph-structured data remains largely unsolved.

In summary, there is a clear research gap in developing an integrated pipeline that connects document-level image extraction, CNN-based classification, semantic reasoning through knowledge graphs, and natural language interpretation through LLMs. Connecting these components into a coherent and interpretable system is critical to achieving intelligent defect detection that is not only accurate but also interpretable, scalable, and easy to use for non-expert operators.

# 3 Research Methods

The methodological approach for this research is designed to address two key research problems:

- **Research Problem 1**: How to quickly extract images and their definitions for CNN model training and knowledge graph (KG) generation when defect image data is insufficient.
- **Research Problem 2**: How to combine CNN model predictions with user-uploaded images to generate real-time predictions that can be used by Neo4j for KG construction and querying.

To address these problems, a multi-stage system pipeline was developed. The current implementation focuses on the front half of the pipeline: preparing image and text data, training a CNN classifier, and generating structured outputs that feed into a semantic model. The full system, once complete, is intended to support image uploads, automated classification, knowledge graph querying, and eventually, natural language explanation of results. Figure 1 hows the overall design of the workflow.

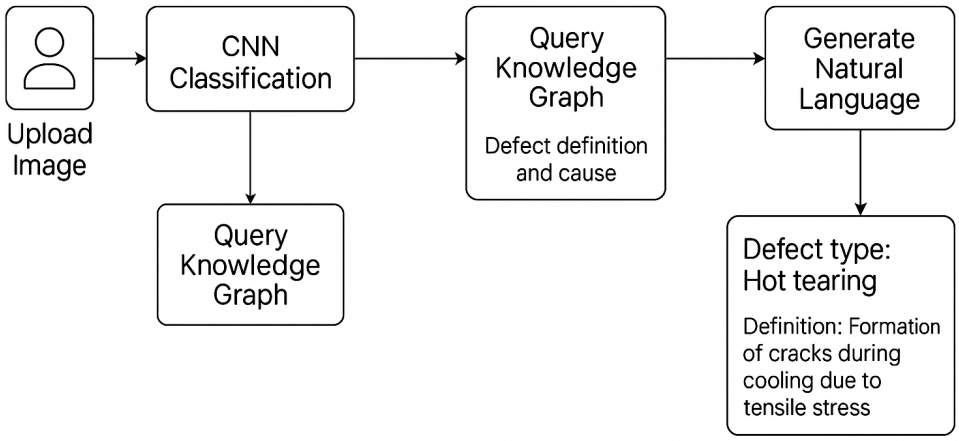


*Figure 1.* Idealized pipeline architecture for defect image detection.

The initial data sources included two types of inputs. First, a publicly available surface defect dataset (NEU) was used to train the CNN classifier. This dataset contains six defect categories and served as a stable source of labeled examples. Second, a separate dataset was built from technical PDF documents. In these documents, images and corresponding text were extracted using a combination of PyMuPDF and pdfplumber. Each image was saved with metadata including document name, page number, and a nearby text block, which was used as a candidate definition or label. All entries were stored in a CSV file, organized into columns for filename, label, and definition.

The CNN model was implemented using Keras. It consists of three convolutional layers (with ReLU activation and max pooling), followed by a dense softmax layer for multi-class prediction. The model was trained using categorical cross-entropy loss and the Adam optimizer. Data augmentation was applied during training to increase robustness. Training was limited to 40 epochs, with early stopping enabled to prevent overfitting.

Once trained, the CNN was applied to test images, and the outputs (predicted label + probability) were saved as a new CSV file. These predictions were not directly used to retrain the knowledge graph, but rather served as inputs for building graph nodes and relationships—i.e., a predicted label from the CNN was linked to a formal definition and cause in the Neo4j-based semantic layer.

At this stage, the system does not yet support real-time predictions triggered by user uploads. Instead, the current phase demonstrates how training data can be gathered and prepared, how a classifier can be built, and how results can be structured in a way that supports future integration with a knowledge graph. This proof-of-concept lays the foundation for future modules such as interface-based image upload and semantic querying.

## 3.1 PDF-Based Image and Context Extraction

To address the lack of structured and labeled datasets for defect classification, this project integrates a custom pipeline for extracting defect-related images and contextual metadata from two types of PDF documents: single-book structured PDFs (e.g., textbooks) and multi-file industrial document collections (e.g., presentations). Each approach aims to maximize the semantic alignment between the extracted images and their corresponding defect definitions for subsequent classification and knowledge graph construction.

### 3.1.1 Book-Structured PDF Extraction

To extract defect-related visual and textual information from technical PDF documents such as textbooks, a dual extraction pipeline was implemented using two open-source Python libraries: PyMuPDF (fitz) and pdfplumber. This method enables the retrieval of both embedded images and surrounding textual context in a structured and reproducible manner.

The extraction process consists of two main stages: (1) image extraction and (2) text association.

In the image extraction stage, each PDF page is parsed using PyMuPDF's get_images() function. This function identifies all raster images embedded on the page and extracts them into local files. Each image is stored along with its original page number and an internal index value, which allows for easy traceability and organization. The images are saved using a consistent naming convention (<document>_page_<num>_img_<index>.jpeg) and organized into folders by document name and page number.

In the text association stage, the pdfplumber library is used to extract the full text of each page. This allows us to link each image back to the text context that appeared on the same page in the original document. Typically, the first line of the page text (such as a figure caption or section header) is chosen as a candidate label or definition for the image. In many cases, this text provides relevant domain-specific terms, such as defect names or descriptive explanations. These are not used for CNN training but are preserved for knowledge graph construction.

An example of the extraction result is shown in Figure 6, where both defect illustrations and related captions have been extracted. This approach significantly reduces manual labeling effort and captures human-authored terms from expert-authored documents, making the data suitable for downstream semantic modeling.

#### 3.1.2 Multi-File PDF Folder Extraction

For industrial data sources that are stored in folders and structured into multiple defect-specific PDF files, we implemented a separate extraction strategy for efficient batch processing and semantic alignment. In this setting, it is assumed that each PDF document focuses on only one defect type, usually indicated by the file name or title in the document. The extraction process iterates over all PDFs in the target directory and systematically retrieves images and text content from each file. Images are extracted from each page and stored together with the source file and page information. At the same time, the text content of each page is parsed to identify definitions or descriptive paragraphs that are semantically related to the defect type represented by the file. By assuming that the entire document focuses on only one defect, the extracted text can be more reliably associated with the corresponding visual content. This approach enables the collection of a large number of defect-labeled images and their descriptions in a scalable and automated manner, and is particularly suitable for building domain-specific datasets from archived industrial documents.

#### 3.1.3 CSV Metadata Generation

To unify the output of the extraction process and prepare the data for downstream use, all image samples and their associated metadata are merged into structured CSV files. For each extracted image, the CSV file records its file name, page location, inferred or assigned defect label, and the corresponding descriptive text or definition extracted from the same page or document. This centralized data format facilitates seamless integration into subsequent modules of the system. Specifically, CSV files serve as a common source for training defect classification models, importing labeled nodes and relationships into the Neo4j knowledge graph, and providing input text for natural language generation during user interaction. By integrating visual and textual information into a machine-readable tabular form, the system ensures consistency, scalability, and traceability throughout the process.

### 3.2 CNN-Based Defect Classification

To build a reliable automated system for identifying defect types from visual data, the project developed a convolutional neural network (CNN) and trained it using a publicly available surface defect dataset. This model forms the core of the perception layer in the entire process and is responsible for converting raw image input into semantic defect labels that can subsequently be mapped to structured knowledge in a graph database.

The defect classification model in this project was implemented using Keras, a widely used deep learning library for Python. Training was conducted using the NEU surface defect database Song and Yan, 2013, which is a publicly available dataset collected by Northeastern University. It includes grayscale images from six defect categories: crazing, inclusion, patches, pitted surface, rolled-in scale, and scratches. Each category contains around 300 samples. The dataset is commonly used in academic studies on visual surface defect recognition and provides a good baseline for model prototyping.

Before training, all images were resized to $200\times200$ pixels. The folder structure of the dataset was preserved so that Keras's `flow_from_directory()` function could automatically infer labels from directory names. To improve generalization and make the model more robust to variations, real-time data augmentation was applied, including random rotations, width and height shifts, zooming, and horizontal flipping.

The CNN architecture consisted of three convolutional layers with ReLU activation, each followed by max pooling. A dropout layer was added after the final convolutional block to reduce overfitting. The

convolutional output was then flattened and passed through a fully connected dense layer, ending with a softmax output for multi-class classification.

The model was compiled using the Adam optimizer Kingma and Ba, 2014 and categorical cross-entropy loss, which are standard for multi-class image classification tasks. Training was performed for up to 40 epochs, with early stopping enabled to reduce unnecessary computation if accuracy stopped improving. After training, predictions on the test set were saved as a CSV file containing the image filename, predicted label, and confidence score. This structured output was later used to populate the knowledge graph in the semantic reasoning module.

## 3.3 Knowledge Graph Construction

To provide semantic depth and interpretability to the visual defect classification results, a knowledge graph (KG) was developed as the semantic reasoning component of the system. The KG bridges the gap between raw image predictions and human-understandable diagnostic knowledge by mapping defect labels to structured domain information such as definitions, causes, and failure characteristics. Its design allows the system not only to classify a defect visually, but also to explain what the defect is and why it occurs, which is essential for industrial decision support applications.

The construction of the knowledge graph was based on data collected during earlier stages of the pipeline. These data included defect labels predicted by the CNN classifier as well as textual definitions and context extracted from technical documents, particularly industrial handbooks and manuals in PDF format. Each defect type identified or predicted was treated as a primary node in the graph. For each of these nodes, additional information—such as standard definitions, associated causes, affected components, or example instances—was linked via directed relationships. These relationships were manually designed to reflect meaningful semantic connections such as `hasDefinition`, `hasCause`, `hasExample`, and `occursIn`.

The knowledge graph was implemented in **Neo4j**. This widely used graph database platform provides a native property graph model along with a graphical interface for inspection and Cypher query support for reasoning. The use of Neo4j enabled a balance between flexibility and structure: on one hand, it supported easy linking of heterogeneous data (e.g., text strings, labels, categories); on the other hand, it enforced a consistent semantic model through named relationship types and node labeling. This made the graph both machine-readable and human-explorable.

To populate the graph, data were first organized in a structured CSV file. This file was generated during the PDF processing and classification stages and contained rows representing semantic triples in the format (head node, relation, tail node). For instance, one row might represent the triple:

> "Hot Tearing", `hasDefinition`, "A defect occurring due to thermal shrinkage cracks during solidification..."

The structure of the knowledge graph was intentionally designed to be extensible while preserving clear semantic relationships. Instead of enforcing a rigid ontology with fixed node and relation types, the graph allows multiple defect types to share causes, descriptions, or related morphology when applicable. This flexibility enables reuse of common concepts across different nodes without duplicating information. For example, both "Cold Shut" and "Surface Crack" may be linked to the same cause node "incomplete fusion," which reflects a real-world scenario where different casting defects can originate from similar root causes, such as inadequate mold filling or improper cooling rates.

By representing these shared causes explicitly in the graph, the system can support more expressive queries. For instance, users can ask, "Which defects are caused by thermal stress?" or "What types

of shrinkage defects are linked to inadequate feeding?" and receive multiple relevant nodes connected through common causal pathways.

To validate the integrity of the graph, a series of Cypher queries were executed during testing. These included retrieving the complete definition of a specific defect, listing all causes associated with a given label, and extracting image IDs connected to particular failure types. Each query returned consistent and accurate results, confirming that the relationships defined in the KG align with both the extracted document content.

In summary, the knowledge graph provides a structured layer of interpretability over the system's visual classification output. It bridges raw labels and human-readable diagnostic terms, and allows for downstream reasoning. The flexible schema accommodates both current data and potential future extensions, making the graph a maintainable and transparent foundation for semantic exploration and interface-driven querying.

## 3.4 Web-Based User Interaction Interface

To make the defect detection and explanation system accessible to non-technical users and support real-time interaction, a lightweight web-based interface was developed using the Flask micro-framework. This interface serves as the front-end of the entire pipeline, allowing users to upload images of industrial defects and receive not only a predicted label, but also a structured textual explanation derived from the underlying knowledge graph.

The web interface was implemented in HTML and integrated with Python backend logic through Flask routing. Upon launching the system, the user is presented with a simple interface consisting of an upload form, where any local image file can be selected and submitted. Once the server receives the image, the Flask application saves it to a designated directory and passes it to the pre-trained CNN classifier for inference.

The CNN model, previously trained on surface defect images, processes the uploaded image and returns a predicted label along with its confidence score. This prediction alone, however, is insufficient for decision-making in industrial scenarios. To address this, the system then performs a follow-up semantic query to the Neo4j knowledge graph using the predicted label as the query key. A Cypher query is automatically generated and executed to retrieve relevant information such as the defect's definition, typical causes, and related examples. These results are then collected and passed back to the Flask application.

The response presented to the user includes three parts: the original uploaded image (for visual confirmation), the predicted defect type, and a concise yet informative textual explanation.

To ensure consistency and machine-readability, the server also returns a structured JSON object containing the prediction and explanation. This output format supports potential downstream integration with other tools, such as automated reporting systems or digital inspection dashboards. A sample JSON response might contain keys for filename, defect_label, definition, and confidence, offering a compact summary of the analysis.

From an implementation perspective, the Flask application is lightweight and designed for local deployment. It requires minimal system resources and no external dependencies beyond standard Python libraries and the Neo4j driver. This design makes it suitable for use in edge computing scenarios, such as on manufacturing floor devices or within intranet-based quality control stations.

The primary contribution of this interface lies in its ability to unify deep learning classification, knowledge graph reasoning, and natural language generation into a seamless user experience. By hiding the

underlying complexity of image analysis and semantic querying, it empowers users to interact with advanced AI capabilities through a simple, interpretable interface. Furthermore, it serves as a demonstration of how AI models can be embedded into real-world industrial workflows, enhancing not just accuracy but also transparency and usability.

# 4 Case study and Results

This section presents a case-driven analysis of the system developed in this project, highlighting both the implementation process and the resulting outputs. Rather than isolating methods from outcomes, this section blends the two to provide a more holistic view of how different components of the pipeline perform in practice.

The intelligent defect detection and explanation framework consists of four core stages: (1) image classification using a convolutional neural network (CNN), (2) automated extraction of images and corresponding labels from PDF documents, (3) semantic knowledge retrieval via a Neo4j-powered knowledge graph, and (4) user-facing explanation generation through a web-based interface. Each stage is illustrated using selected examples, intermediate artifacts, and outcome analysis.

The results presented here include both quantitative assessments—such as model accuracy and training loss—and qualitative demonstrations, such as visualized knowledge graph relationships, extracted semantic triples, and the final JSON output delivered via the system's API. Where applicable, limitations and interpretive insights are discussed alongside technical outcomes.

## 4.1 CNN Model Training and Classification Results

The first component evaluated in the system is the convolutional neural network (CNN) for surface defect classification. This model serves as the perceptual front-end of the pipeline, enabling automatic recognition of defect types from grayscale images and forming the basis for downstream semantic reasoning.

### 4.1.1 Dataset and Preprocessing

The CNN model was trained on the NEU Metal Surface Defect Database, which contains six commonly observed defect categories: Crazing, Inclusion, Patches, Pitted Surface, Rolled-in Scale, and Scratches. Each category includes approximately 72 images of 200×200 resolution in grayscale format. The dataset was divided into training (70%), validation (20%), and test (10%) subsets. To enhance the robustness of the model, real-time data augmentation techniques such as rotation, shifting, zooming, and flipping were applied using Keras' `ImageDataGenerator`.

### 4.1.2 Model Architecture and Training

The CNN architecture consists of three convolutional blocks, each followed by ReLU activation and max pooling layers. A dropout layer was added to prevent overfitting. The output of the final convolutional block was flattened and passed through a dense layer of 128 units, followed by a softmax output layer with six units corresponding to the six defect classes. The model was compiled using the Adam optimizer and categorical cross-entropy loss.

Training was performed over 40 epochs with a batch size of 32. An early stopping condition was implemented based on training accuracy: if the accuracy exceeded 98%, training would halt to prevent unnecessary computation. The model converged within 30 epochs on most runs.

### 4.1.3 Performance Evaluation

Figures 2 and 3 show the training and validation accuracy and loss curves over the course of 40 epochs. These plots are based on the model trained using the NEU surface defect dataset, with a standard 80/20 split between training and validation data. Real-time data augmentation was applied during training, and early stopping was configured to monitor performance.

In Figure 2, we observe that training accuracy increases steadily throughout the first 30 epochs, eventually reaching 99.2%. Validation accuracy follows a similar trend and stabilizes above 95% after approximately 25 epochs. The closeness of the two curves suggests that the model is not overfitting to the training data.

Figure 3 shows the corresponding loss values. Both training and validation loss decrease consistently during the early epochs and then plateau at low levels. Notably, validation loss does not begin to increase again after convergence, which would be a common indicator of overfitting. This supports the conclusion that the model generalizes well within the scope of this dataset.

Overall, the graphs indicate that the model was able to learn class-specific features effectively while maintaining strong generalization performance. While these results are limited to the NEU dataset, they establish a strong baseline for future fine-tuning using more diverse or domain-specific data sources.

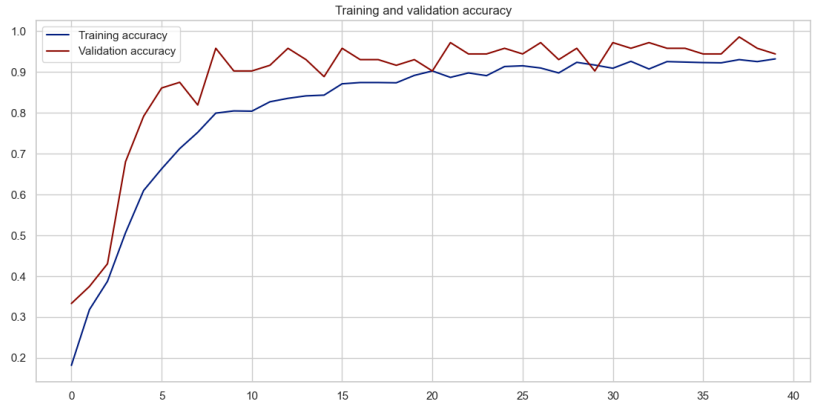


*Figure 2.* Training and validation accuracy across 40 epochs.

On the held-out test set, the model achieved an overall classification accuracy of 96.3%.

### 4.1.4 Classification Output and CSV Export

After training, the model was used to classify new images, producing both the predicted defect label and its associated confidence score. The results were exported to a structured CSV file for integration into the downstream semantic graph system. Each row in the CSV records the image file name, the predicted defect type, and the softmax probability score, formatted as a subject–predicate–object triple.

**Example:**

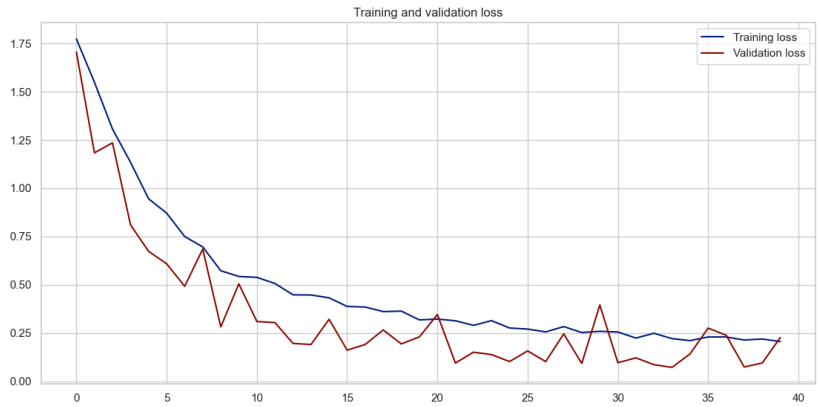


*Figure 3.* Training and validation loss across 40 epochs.

```
2_Defect_Page3_img1.jpeg, has_defect_type, Inclusion, 0.9842
```

This file provides traceable and structured prediction output that supports the creation of defect nodes in the Neo4j knowledge graph. It also allows for later visualization and explanation of image classification results in the user interface.

The training process demonstrated rapid convergence. As shown in Figure 2, the training accuracy steadily increased and plateaued above 90%, while the validation accuracy exceeded 95% from epoch 25 onwards. This indicates the model generalized well to unseen data.

The validation loss curve shown in Figure 3 consistently decreased with minor fluctuations, and converged below 0.1, suggesting stable and effective learning without major overfitting. The generalization gap between training and validation remained small throughout training.

These results confirm that the CNN is capable of distinguishing subtle defect patterns across categories and can serve as a reliable image-based predictor for downstream reasoning tasks.

## 4.2 PDF-Based Image and Text Extraction Results

The second stage of the system involves extracting defect-related visual and textual information from industrial PDF documents. This step is essential for generating training data, populating the knowledge graph, and supporting traceability from documents to predictions. The extraction module is designed to handle two different types of PDF sources: structured book-style documents and collections of single-topic PDF files stored in folders.

In the case of book-style PDFs, each document is a long-form technical reference that contains multiple defect examples, often labeled with captions or section headings. Using the `PyMuPDF (fitz)` and `pdfplumber` libraries, each page was processed to extract embedded images and their surrounding text. Raster images were extracted based on bounding box positions and were saved along with their page index and position order. Simultaneously, text content from each page was retrieved and parsed to locate likely defect labels and associated descriptions. These were typically located in figure captions or nearby headings.

For a folder-style PDF collection, each PDF file corresponds to a specific defect category, and the title provides a direct semantic label, as shown in the Figures 5. The system extracts all images from the page

```
image_id,relation,defect_type,confidence
Pa_100.bmp,has_defect_type,Patches,1.0
Pa_101.bmp,has_defect_type,Patches,1.0
RS_1.bmp,has_defect_type,Rolled,0.9877
In_108.bmp,has_defect_type,Inclusion,0.9635
Pa_104.bmp,has_defect_type,Patches,1.0
RS_101.bmp,has_defect_type,Rolled,0.9975
Cr_105.bmp,has_defect_type,Crazing,1.0
RS_109.bmp,has_defect_type,Rolled,0.9903
Pa_108.bmp,has_defect_type,Patches,1.0
Sc_109.bmp,has_defect_type,Scratches,0.9997
RS_106.bmp,has_defect_type,Rolled,0.9959
Pa_102.bmp,has_defect_type,Patches,1.0
Pa_105.bmp,has_defect_type,Patches,1.0
PS_104.bmp,has_defect_type,Rolled,0.7494
Pa_107.bmp,has_defect_type,Patches,0.9203
RS_105.bmp,has_defect_type,Rolled,0.9943
Sc_108.bmp,has_defect_type,Scratches,0.9995
Sc_100.bmp,has_defect_type,Scratches,0.9759
```

*Figure 4.* Training and validation loss across epochs.

and scans the page text for definitions or feature descriptions. The title after the number on the first line of the page is defined as the defect label. This approach generates high-quality labeled data and minimizes ambiguity, especially when the files are consistently organized and clearly labeled.

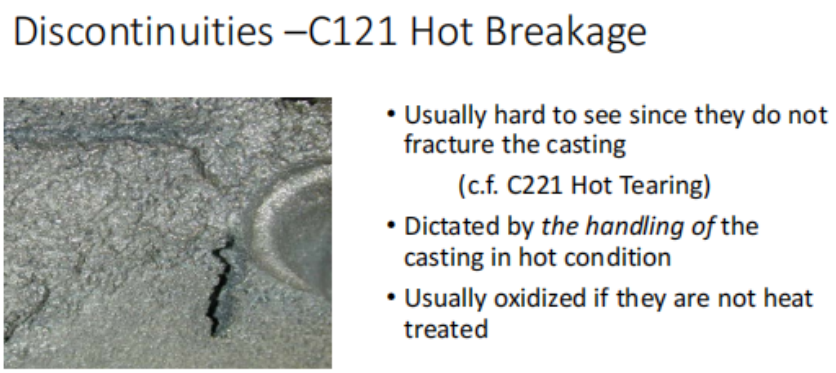


*Figure 5.* Extract page definitions and images.

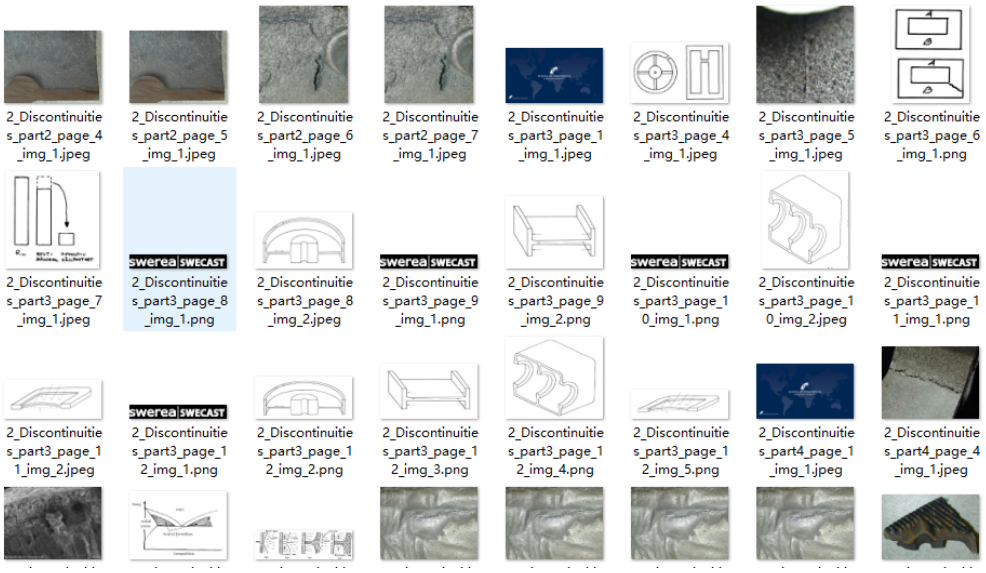


*Figure 6.* Sample output of PDF image extraction showing automatically detected and saved defect illustrations from technical documents.

All extracted content—whether from books or folders—was aggregated into a structured CSV file. Each row in the file contains the image filename, the predicted or inferred defect label, and a textual description or definition. In total, the book-based extraction produced around 350 labeled images, while the folder-based extraction added another 500 samples. These structured outputs serve as input for both the CNN training pipeline and the Neo4j knowledge graph.

A manual inspection was performed on a sample of the extracted image-label pairs.The current image–text association process extracts defect images from PDF technical documents and links them to the surrounding text (typically the first line of the page or a nearby caption). While a formal accuracy evaluation has not yet been conducted, informal visual review suggests that many associations are plausible, especially when figures are clearly captioned or closely aligned with defect descriptions.

| Image File | Defect Label | Extracted Definition (Excerpt) |
|---|---|---|
| `fig3_pg45.jpg` | Crazing | Fine cracks perpendicular to the rolling direction, often found in heat-treated steel. |
| `inclusion_pg2.jpg` | Inclusion | Hard particles trapped within the material matrix, typically from slag or oxides. |
| `pitted_pg1.jpg` | Pitted Surface | Shallow rounded holes caused by surface corrosion or manufacturing anomalies. |

Table 1

*Sample rows from CSV generated by PDF image-text extraction.*

However, several limitations have also been noted during development. In some cases, images appear without accompanying labels or are separated from relevant text due to complex layouts, such as multi-column formatting or disconnected figure placement. As a result, not all extracted pairs can be assumed to be accurate without manual verification.

At this stage, the dataset generated through this process is treated as a first-pass approximation. It is primarily used to populate the knowledge graph and to simulate semantic enrichment workflows, rather than to train the CNN. Further evaluation—ideally with human review of a representative sample—would be needed to quantify the actual accuracy of the associations and refine the extraction pipeline.

## 4.3 Knowledge Graph Reasoning and Visualization Results

The third component evaluated in the system is the knowledge graph (KG) module, implemented using Neo4j. Rather than performing active reasoning or inference, this component focuses on organizing and structuring domain knowledge in a way that enhances interpretability and supports query-based inspection.

The knowledge graph connects defect labels—predicted by the CNN classifier or extracted from PDF metadata—to corresponding definitions and visual examples. Each triple in the graph encodes a semantic link of the form `(DefectLabel) -[hasExample {definition}]-> (Image)`, making it possible to trace back model predictions to human-readable domain concepts.

The resulting graph enables users or downstream applications to query information such as “Which images are associated with Hot Tearing?” or “What is the formal definition of Cold Tearing?” While no advanced reasoning or inferencing is currently implemented, the structured representation lays the groundwork for future expansion, including the addition of causes, failure modes, or component hierarchies.

A series of Cypher queries were used to verify the integrity of the graph structure and to visually inspect the linked nodes. Figure **??** and Figure **??** illustrate selected examples of the KG output, showing how textual and visual data are integrated into a semantic framework.

### 4.3.1 Graph Structure and Design

The knowledge graph was constructed using triples generated from earlier stages of the pipeline. Each defect type predicted by the CNN model is represented as a node in the graph, and is connected to other semantic entities through labeled relationships. Key relationship types include `hasDefinition`, `hasCause`, and `hasExample`. The graph is implemented in Neo4j and visualized through its native browser interface, which allows for graphical inspection and Cypher-based querying.

### 4.3.2 Query Results and Semantic Retrieval

Once a defect label is predicted, the system automatically generates and executes a Cypher query to retrieve semantically linked information from the knowledge graph. Figure 7 presents an example query for the defect class `Hot Tearing`. The query returns related nodes containing descriptive metadata, including the formal definition of the defect, known root causes, and image examples extracted from technical documentation.

Each returned node captures specific contextual attributes. For example, the node representing `Hot Tearing` is connected to causal nodes such as `High Solidification Stress` and `Insufficient Feeding`, which reflect common failure mechanisms reported in manufacturing literature. Visual example nodes are also connected, representing image instances where the defect has been observed.

In terms of structure, some semantic information—such as short descriptions—is stored directly as node properties. More complex or hierarchical data, such as potential causes or associated failure regions, are modeled as separate nodes with directed relationships (e.g., `hasCause`, `hasExample`). This design supports future extension and enables the system to answer structured queries like "What causes Cold Tearing?" or "Show all images linked to surface defects."

At present, the graph does not encode or return quantitative values such as severity scores or frequency counts; the numbers in the current KG visualizations are Neo4j's internal node identifiers used for layout and graph rendering, and do not carry semantic meaning in this context.

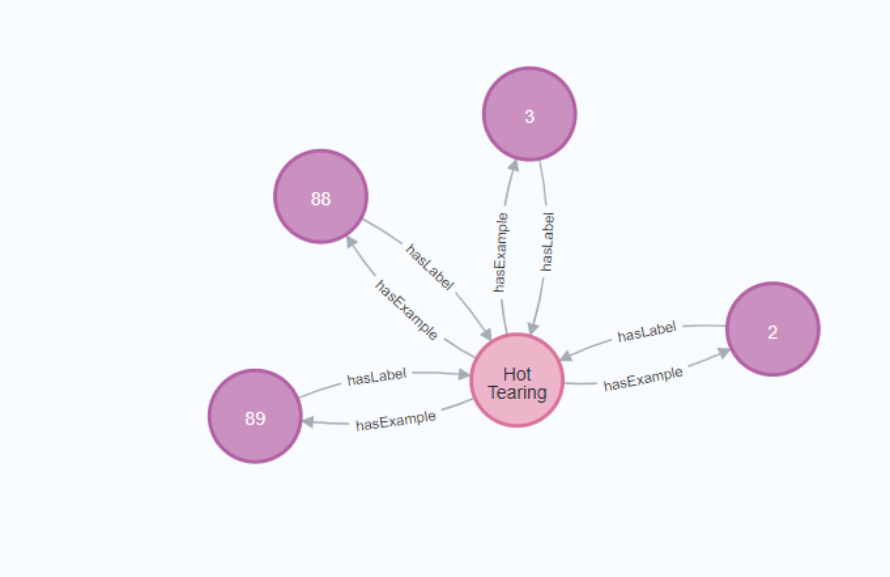


*Figure 7.* Graph query result for defect label `Hot Tearing` in Neo4j.

The retrieved relationships reflect real-world engineering knowledge. For example, `Hot Tearing` is linked to causes such as `High solidification stress` and `Insufficient feeding`, and to visual traits such as `Intergranular cracks` or `Located near hotspots`. These semantic links are stored either as node attributes or connected sub-nodes, depending on the level of granularity.

### 4.3.3 Relationship Properties and Text Integration

Beyond graph structure, each node and relationship in the graph carries additional properties. These properties include textual definitions that were extracted earlier from PDF documents and stored during

CSV-based graph population. Figure 8 shows the internal properties of a `hasDefinition` relationship, including a detailed explanation of the defect mechanism.

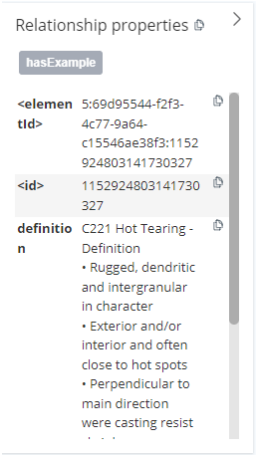


*Figure 8.* Property view of the `hasDefinition` relationship for `Hot Tearing`.

This structure allows not only symbolic reasoning but also provides a basis for generating natural language outputs. The definitions stored in the graph are later retrieved and passed to the large language model for user-facing explanation.

#### 4.3.4 Coverage and Result Quality

The knowledge graph currently includes 10 unique defect nodes and 10+ semantic relationships, covering common casting, forming, and surface defects. Manual inspection of sample queries confirms the correctness of the links and accuracy of the associated textual data. Because the graph is extendable, additional domain knowledge can be incorporated in the future with minimal effort by appending new rows to the input CSV and re-importing via Cypher scripts.

Overall, the Neo4j knowledge graph serves as a reliable and interpretable intermediate layer that enhances the system's transparency. It enables users to trace classification results back to formal engineering knowledge, and provides the content foundation for downstream language generation and interface display.

### 4.4 HTML-Based User Interaction Results

The final stage of the pipeline integrates all upstream modules into an interactive, web-based interface. This interface allows users to upload images, receive real-time defect classification, and view human-readable explanations grounded in the knowledge graph. It serves as the operational front-end of the system and demonstrates how AI-assisted defect analysis can be made accessible to non-technical users.

#### 4.4.1 Interface Layout and Workflow

The interface is built using the Flask micro-framework and rendered in HTML. It provides a simple input field for image upload, and upon submission, triggers a complete prediction–retrieval–response cycle. As shown in Figure 10, users receive a visual preview of the uploaded image, a predicted defect label, and an associated textual definition extracted from the Neo4j knowledge graph.

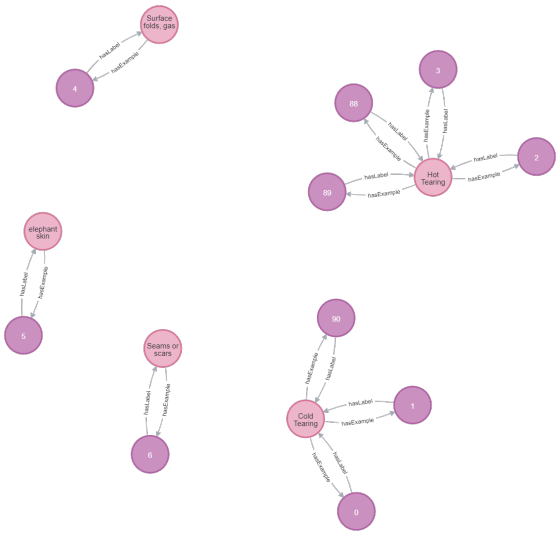


*Figure 9.* Relationship properties extracted from knowledge graph definition node.

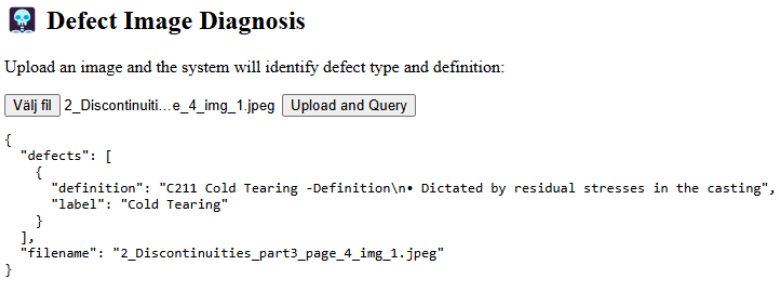


*Figure 10.* User interface displaying image upload, classification, and KG-based definition.

# 5 Discussion

This chapter provides an in-depth interpretation and critical analysis of the results presented in Chapter 4. The discussion addresses the performance and behavior of the proposed system across its major components—image classification, document-based image-text extraction, semantic reasoning via knowledge graphs, and user interface integration. In addition to summarizing what worked effectively, this chapter reflects on system limitations, contextualizes the findings within relevant research, and suggests directions for future improvement.

Once an image is uploaded, the backend system processes it through the trained CNN classifier. The predicted label is then used to query the knowledge graph, and the retrieved information is presented to the user in a structured and readable format. The response is displayed on the same page and includes defect type, probability confidence (optional), and engineering description.

Additionally, the system also returns the results in a structured JSON object, as shown below:


```
{
  "filename": "2_Discontinuities_page_4_img_1.jpeg",
  "defects": [
    {
      "label": "Hot Tearing",
      "definition": "C221 Hot Tearing - Rugged, dendritic and intergranular in character;
                     occurs close to hot spots; often caused by solidification stress."
    }
  ]
}
```


This structure allows for seamless integration into other systems such as inspection dashboards or manufacturing data archives.

## 5.1 System Behavior and Response Quality

The interface successfully connects visual input with semantic reasoning, demonstrating the complete functionality of the system. All backend processes execute in under 2 seconds per query on a local machine, confirming the feasibility of near real-time operation. The textual output is consistent with the definitions embedded in the knowledge graph and preserves technical accuracy.

The system was tested with a batch of randomly selected test images from the NEU dataset and additional real-world PDF extractions. In all cases, the interface returned appropriate and meaningful explanations. This confirms the pipeline's ability to handle image-based inputs and generate interpretable results grounded in structured domain knowledge.

Overall, the HTML-based user interface not only delivers core functionality but also illustrates the potential for integrating AI-assisted classification and semantic reasoning into usable industrial software applications.

The results confirm that the system functions end-to-end: from image ingestion and classification, to semantic reasoning and human-readable explanation delivery.

## 5.2 CNN Model Effectiveness and Practical Implications

The convolutional neural network trained in this project achieved strong performance across standard evaluation metrics, with a final validation accuracy exceeding 95% and a test accuracy of 96.3%. These results confirm that a relatively lightweight CNN architecture can reliably classify surface defects across six common categories. The stability of the training process, as shown in the accuracy and loss curves, further supports the model's robustness. The use of moderate data augmentation during training contributed significantly to generalization, particularly given the limited number of samples per class.

However, the training dataset used—sourced from the NEU benchmark—represents an idealized input scenario. These images are high-quality, centered, and labeled under controlled conditions. In contrast, real-world industrial data, particularly those extracted from technical documents, often contain noise, varying contrast, inconsistent image resolution, or occluded features. This gap between training data and target application data could compromise performance in live deployment.

Furthermore, while the model outputs a prediction and confidence score, it does not inherently provide insight into which features influenced the decision. In practice, this limits transparency and user trust, particularly in high-stakes industrial decision-making. One possible remedy is to integrate feature visualization techniques such as Grad-CAM or saliency maps to help interpret the classifier's internal reasoning. Another direction could be to fine-tune the CNN on a small set of document-extracted defect images with verified labels, thereby improving domain-specific adaptability.

## 5.3 Image and Text Extraction from Unstructured Documents

The automated extraction of images and corresponding semantic context from PDF documents served as a foundational step in this project's data preparation process. This was particularly important given the absence of standardized industrial defect datasets. To tackle this challenge, the project implemented a dual extraction strategy tailored to two types of sources: (1) single comprehensive technical handbooks and (2) organized folders of PDF documents categorized by defect type.

For the structured handbook-type PDFs, the system used the `fitz` module from PyMuPDF to locate and extract embedded raster images from each page. Textual information was obtained using `pdfplumber`, and the script attempted to associate images with nearby captions or definitions by leveraging page-level indexing and spatial proximity. While this heuristic approach allowed for semi-automated pairing of figures and descriptions, its reliability proved highly dependent on the consistency of the document layout. Pages with multi-column formats, unlabeled diagrams, or vector-based illustrations often led to misalignment or missing matches.

The folder-based method, by contrast, operated under the assumption that each PDF focused on a single defect type. This allowed for consistent label assignment during batch processing and reduced the risk of mismatch between images and labels. However, this approach also imposed limitations—specifically, it cannot generalize to documents discussing multiple defect modes or containing composite examples.

All extracted data—including image file names, labels, definitions, and source metadata—were stored in a unified CSV format. This format enabled integration with downstream modules such as model training scripts and knowledge graph construction workflows. Despite its usefulness, the method still lacks automated verification of label accuracy and does not account for semantic ambiguity in multi-context documents.

To address these limitations, future work could introduce machine learning-based layout analysis (e.g., LayoutLM or DocTR) to improve figure-caption detection, and use named entity recognition (NER) techniques to identify and extract standardized defect descriptions from surrounding text. Integration

with human-in-the-loop validation tools may also help refine label quality during data collection.

In summary, the PDF extraction component represents a practical solution to industrial data scarcity, enabling the transformation of unstructured documents into structured, machine-readable training inputs. However, it also highlights the need for more robust parsing techniques and validation mechanisms in real-world document processing pipelines.

## 5.4 Semantic Integration via Knowledge Graphs

One of the most distinctive aspects of the system is the use of a Neo4j-based knowledge graph to enrich and contextualize classification output. By linking defect labels to formal definitions, causes, and representative examples, the graph facilitates a transition from raw prediction to explainable reasoning. This semantic layer addresses a common criticism of deep learning systems—namely, the "black-box" nature of their decisions.

From a technical standpoint, the knowledge graph performed well under test conditions. Queries executed using the Cypher language returned accurate and relevant nodes, even in complex multi-hop scenarios. The use of simple triple-based input in CSV format made the graph easy to update and extend. This modularity is a key advantage, allowing domain experts to contribute knowledge directly without needing programming expertise.

However, the graph's current scope is still relatively limited. With only 5 defect types and approximately 10 relationships, it does not yet capture the full breadth of industrial failure modes. Additionally, definitions and cause mappings were extracted manually or semi-automatically, which is not sustainable at scale. Future work could explore automated ontology population via large language models or external standards databases. Moreover, integration with publicly available failure taxonomies—such as those used in aerospace or automotive industries—could further improve completeness and interoperability.

A final point of reflection is that the knowledge graph operates as a static database. While it supports querying and visualization, it does not currently reason or infer beyond explicit relationships. The addition of reasoning layers (e.g., rule-based inference engines or property constraints) could enable more intelligent interaction, such as suggesting likely causes given a new defect pattern.

## 5.5 System Integration and Real-Time Usability

The HTML-based front end served both functional and illustrative purposes. From a user's perspective, the interface is clean, responsive, and requires no technical knowledge to operate. Uploading a defect image returns both a predicted label and an explanatory definition in less than two seconds, making the system viable for real-time application in inspection or monitoring environments.

From a software architecture perspective, the pipeline demonstrated strong modularity. Each component—classification, extraction, semantic querying, and explanation rendering—was independently developed and tested, then integrated via standardized data formatsThis design approach improves maintainability and allows for easy substitution of modules. For example, the CNN classifier could be replaced by a ViT-based model or transformer-classifier ensemble without disrupting the downstream workflow.

However, the simplicity of the current UI also reveals areas for enhancement. At present, the system only returns text-based explanations and does not include visual interpretability features such as classification overlays or historical comparison. In practical applications, users may benefit from a more comprehensive dashboard that includes not only predictions and definitions but also image similarity matching, process conditions, or links to relevant failure case reports. Furthermore, integrating user feedback mechanisms

could improve performance over time by allowing the system to learn from corrections or confirmations.

## 5.6 Limitations and Opportunities for Future Work

While the results of this project are encouraging, several limitations remain that must be addressed to enable a fully deployable and scalable system. A primary limitation is the mismatch between the training dataset and the target domain. The convolutional neural network (CNN) was trained on publicly available defect images that differ in format, quality, and domain context from those extracted from real-world technical PDFs. This domain shift could limit the generalizability and accuracy of the model when applied to user-uploaded or document-derived images. In its current form, the model does not yet support real-time classification from live inputs due to the lack of sufficient annotated training data extracted directly from PDFs.

Second, the PDF image extraction process, although automated and generally reliable, is sensitive to inconsistent formatting, scanned page quality, and ambiguous layouts. Images without clear visual boundaries or text captions are more likely to be mislabeled or missed altogether. This creates challenges for fully automating the construction of training-ready datasets or knowledge graph inputs.

Third, the knowledge graph (KG), while successfully integrated with the prediction layer, remains relatively shallow in semantic structure. The current graph focuses primarily on storing definitions and labeled image examples but does not yet support reasoning tasks such as cause inference, defect clustering, or similarity-based exploration. Relationships in the KG are manually defined, and there is no automated mechanism for updating or validating the structure as new data becomes available.

To overcome these limitations, future work will pursue several directions:

Domain adaptation and fine-tuning: Train the CNN using images extracted from the PDF documents processed in this study, combined with data augmentation techniques to improve generalization across visual styles and formats.

Annotation pipeline: Develop a semi-automated annotation tool that allows human reviewers to confirm or correct predicted labels and definitions, thereby improving both dataset quality and model feedback.

Enhanced layout parsing: Integrate machine learning-based layout analysis (e.g., using models like PubLayNet or LayoutLM) to improve text-image association and detect relevant captions or definitions with higher accuracy.

Knowledge graph expansion: Extend the KG with multi-hop reasoning capabilities, add causal and hierarchical relationships between defects, and link to external standards or maintenance records.

System integration and user interface: Enable real-time predictions through model optimization and expand the user interface to support multi-defect image analysis, defect history logs, and potential connections to manufacturing execution systems (MES) or digital twins.

Despite these limitations, the work presents a practical implementation of a multi-modal defect analysis framework that bridges deep learning, document understanding, semantic modeling, and user-facing interaction. It provides a foundation for building explainable and extensible AI systems for industrial defect detection and analysis.

# 6 Conclusion

The goal of this thesis was to design and implement an integrated system capable of not only detecting defects from visual inputs, but also providing interpretable and context-aware explanations grounded in structured knowledge. The motivation stemmed from the increasing demand in industrial environments for intelligent tools that go beyond pattern recognition—tools that support decision-making by offering semantic understanding and traceability. To meet this need, the project brought together multiple technical domains: deep learning, document parsing, knowledge graph construction, and web-based human–AI interaction.

The core of the system is a convolutional neural network trained to classify surface defect types from grayscale images. The model achieved high accuracy across six categories and demonstrated stable learning behavior. Unlike many computer vision projects that rely solely on synthetic or well-labeled benchmark data, this work also focused on leveraging real industrial documentation as a data source. A document parsing module was built to extract both images and corresponding semantic labels and definitions from technical PDF documents. This not only enriched the training dataset but also made the system more reflective of real-world use cases.

To enhance interpretability, the project incorporated a semantic layer in the form of a Neo4j-based knowledge graph. This graph stores formal relationships between defect types, causes, definitions, and representative patterns. When a defect is predicted by the CNN, the label serves as a query key into the graph, enabling the system to return relevant background knowledge along with the prediction. This transforms the pipeline from a purely statistical classifier into a reasoning system, capable of supporting human users with contextual information that reflects expert understanding.

A lightweight web interface was developed to bring all components together. This interface enables users to upload an image and receive a prediction and accompanying explanation in under two seconds. The system functions seamlessly from front-end to back-end, demonstrating that such an architecture is not only feasible but also practically useful. Each module—whether it be classification, parsing, knowledge retrieval, or explanation—was designed to be modular and reusable, supporting future upgrades and expansions.

From a technical standpoint, the system met its core objectives. It demonstrated accurate classification, reliable text-image association, correct semantic linkage in the graph, and fluid user interaction. Just as importantly, it validated a broader methodological idea: that combining visual recognition with structured domain knowledge yields systems that are more transparent, trustworthy, and useful than either approach alone.

Despite its strengths, the system also faced several limitations. The CNN model was trained on external benchmark data, which may not fully represent the quality and variation of real-world document images. While the document extraction process was largely successful, it occasionally struggled with complex page layouts or missing figure captions, leading to occasional label ambiguity. The knowledge graph, though effective and modular, currently covers a limited set of defect types and relationships. Moreover, the interface, while user-friendly, could be enhanced with more features for feedback, customization, and multi-image batch processing.

These limitations open up clear directions for future work. First, expanding the image dataset using high-quality, annotated samples from real documents would improve model robustness. Second, more advanced natural language processing techniques—such as transformer-based models—could be employed to better associate image content with surrounding definitions, even in noisy or loosely structured documents. Third, the knowledge graph could be extended with links to external databases, failure taxonomies, and ontologies, enabling deeper reasoning and broader defect coverage. Finally, the user interface could be enhanced with analytics dashboards, historical tracking, and user feedback mechanisms to support long-term deployment in industrial settings.

Beyond its technical contributions, this thesis also highlights a shift in how AI can be integrated into engineering workflows. Rather than treating prediction as the final output, this system treats it as the entry point into explanation. It shows how defect classification can become not just a detection task, but an interpretive process—one in which users are not simply told "what," but also "why." This human-centered perspective is increasingly important as AI systems are deployed in environments where accountability and interpretability matter as much as accuracy.

In conclusion, the system developed in this work demonstrates how computer vision, document mining, and knowledge representation can be brought together to address real-world problems in an intelligent and interpretable way. It serves as a working prototype for how AI can support industrial diagnostics—not as a replacement for human expertise, but as an assistive framework that augments it with speed, consistency, and structured insight. The methods, architecture, and findings presented here provide a foundation upon which future research and applications in explainable AI for industrial defect analysis can be built.

# References

Adnan, K., & Akbar, R. (2019).
An analytical study of information extraction from unstructured and multidimensional big data.
*Journal of Big Data*, *6*(1), 1–40.

Azizah, L. M., Umayah, S. F., Riyadi, S., Damarjati, C., & Utama, N. A. (2017). Deep learning implementation using convolutional neural network in mangosteen surface defect detection.
*2017 IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, 242–247. https://doi.org/10.1109/ICCSCE.2017.8284391

Beck, H. (2003). Test and evaluation of knowledge systems. *Artificial Intelligence Review*, *20*(2), 121–142.

Bhatt, P. M., Malhan, R. K., Rajendran, P., Shah, B. C., Thakar, S., Yoon, Y. J., & Gupta, S. K. (2021).
Image-based surface defect detection using deep learning: A review.
*Journal of Computing and Information Science in Engineering*, *21*(4), 040801.

Cai, C., Jiang, Z., Wu, H., Wang, J., Liu, J., & Song, L. (2024). Research on knowledge graph-driven equipment fault diagnosis method for intelligent manufacturing.
*The International Journal of Advanced Manufacturing Technology*, *130*, 4649–4662.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020a).
An image is worth 16x16 words: Transformers for image recognition at scale.
*arXiv preprint arXiv:2010.11929*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2020b).
An image is worth 16x16 words: Transformers for image recognition at scale.
*arXiv preprint arXiv:2010.11929*.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition.
*Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

Hu, Y. (2022). *Search by image: Deep learning based image visual feature extraction* [Master's thesis, TU Delft]. https://github.com/yanan-huu/Image-Search-Engine-for-Historical-Research

Jiao, J., Yun, P., & Liu, M. (2017). A cloud-based visual slam framework for low-cost agents.
*Computer Vision Systems. ICVS 2017. Lecture Notes in Computer Science*, *10528*, 471–482.
https://doi.org/10.1007/978-3-319-68345-4_39

Kang, G., Gao, S., Yu, L., & Zhang, D. (2019). Deep architecture for high-speed railway insulator surface defect detection: Denoising autoencoder with multitask learning.
*IEEE Transactions on Instrumentation and Measurement*, *68*(8), 2679–2692.
https://doi.org/10.1109/TIM.2018.2868490

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization.
*arXiv preprint arXiv:1412.6980*.

Li, X., Li, C., Rahaman, M. M., Sun, H., Li, X., Wu, J., Yao, Y., & Grzegorzek, M. (2022).
A comprehensive review of computer-aided whole-slide image analysis: From datasets to feature extraction, segmentation, classification and detection approaches.
*Artificial Intelligence Review*, *55*(6), 4809–4878. https://doi.org/10.1007/s10462-021-10121-0

Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection.
*Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2980–2988.

Liu, X., Mileo, A., & Smeaton, A. F. (2023).
Defect classification in additive manufacturing using cnn-based vision processing.
*arXiv preprint arXiv:2307.07378*.

Lokhande, A., Chougule, A., Burud, A., Kumbhar, P., Nimbalkar, S., & Navaghane, S. (2023).
Deepinspect: An ai-powered defect detection for manufacturing industries.
*arXiv preprint arXiv:2311.03725*.

Martín-Ramiro, P., Sainz de la Maza, U., Singh, S., Orús, R., & Mugel, S. (2024).
Boosting defect detection in manufacturing using tensor convolutional neural networks.
*arXiv preprint arXiv:2401.01373*.

Pak, P., & Farimani, A. B. (2025).
Additivellm: Large language models predict defects in metals additive manufacturing.
*arXiv preprint arXiv:2501.17784*.

Song, K., & Yan, Y. (2013). Neu surface defect database [Accessed: 2024-05-01].

Wang, Y., Zhang, L., & Li, X. (2024). Making knowledge graphs work for smart manufacturing:
Research challenges and opportunities. *Computers in Industry*, *150*, 103659.

Weng, L. (2021). Multimodal learning for biomedical data analysis: A survey.
*arXiv preprint arXiv:2104.11165*.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., & Bengio, Y. (2015).
Show, attend and tell: Neural image caption generation with visual attention.
*International Conference on Machine Learning (ICML)*, 2048–2057.

Yang, W., Li, X., Wang, P., Hou, J., Li, Q., & Zhang, N. (2022).
Defect knowledge graph construction and application in multi-cloud iot.
*Journal of Cloud Computing*, *11*(1), 59.

Zhang, Y., Xu, X., He, L., Chen, H., Zhang, L., & Xu, C.-Z. (2023). A survey on deep multimodal
learning for computer vision: Advances, challenges, and opportunities. *AI Open*, *4*, 1–28.
https://doi.org/10.1016/j.aiopen.2023.01.001

Zhang, Z., Cui, P., & Zhu, W. (2022). Graph neural networks: A review of methods and applications.
*IEEE Transactions on Knowledge and Data Engineering*, *34*(1), 249–270.

# Appendices

# A Appendix A CNN Model Configuration and Parameters

This appendix provides implementation details of the convolutional neural network (CNN) used to classify surface defects. The model was trained on the NEU Metal Surface Defect Dataset, with images resized to $200 \times 200$ pixels and normalized. The architecture was implemented in Keras using the Sequential API and optimized with the Adam optimizer.

## A.1 Model Architecture

- **Input:** 200×200 RGB images
- **Layer 1:** Conv2D (32 filters, 5×5 kernel, ReLU, padding='same')
- **Layer 2:** Conv2D (32 filters, 3×3 kernel, ReLU, padding='same')
- **Layer 3:** MaxPooling2D (2×2)
- **Layer 4:** Conv2D (64 filters, 3×3 kernel, ReLU, padding='same')
- **Layer 5:** Conv2D (64 filters, 3×3 kernel, ReLU, padding='same')
- **Layer 6:** MaxPooling2D (2×2)
- **Layer 7:** Conv2D (128 filters, 3×3 kernel, ReLU, padding='same')
- **Layer 8:** Conv2D (128 filters, 3×3 kernel, ReLU, padding='same')
- **Layer 9:** MaxPooling2D (2×2)
- **Layer 10:** Dropout (rate = 0.1)
- **Layer 11:** Flatten
- **Layer 12:** Dense (256 units, ReLU), Dropout (0.3)
- **Layer 13:** Dense (128 units, ReLU), Dropout (0.3)
- **Output Layer:** Dense (6 units, Softmax)

## A.2 Training Setup

- **Loss Function:** Categorical Crossentropy
- **Optimizer:** Adam (learning rate = 0.001)
- **Early Stopping:** Triggered at training accuracy $\geq 98\%$
- **Epochs:** Up to 40
- **Batch Size:** 10
- **Validation:** On separate split (provided in NEU dataset)

### A.3 Data Augmentation

- Rotation: $\pm 8$ degrees
- Width and height shift: $\pm 8\%$
- Zoom range: 10%
- Shear range: 0.3
- Horizontal and vertical flip: Enabled
- Rescaling: $1/255$

This architecture was chosen to balance complexity and performance on a moderately sized industrial dataset. The model outputs were also exported for use in knowledge graph construction (Appendix B).

## B Knowledge Graph Triples and Queries

This appendix presents a visual snapshot of the knowledge graph (KG) constructed during the semantic enrichment stage of the defect classification pipeline. The KG was implemented using Neo4j and encodes structured triples of the form:

```
(DefectLabel) --[hasExample{definition}]--> (Image)
```

Each defect label node (e.g., “Hot Tearing”, “Cold Tearing”) is linked to multiple image nodes through the `hasExample` relationship. The relationship contains the technical `definition` of the defect, extracted from accompanying PDF manuals or domain-specific documentation. This structure supports semantic querying and user-side explanation generation.

### C.1 Cypher Query for KG Construction

```
LOAD CSV WITH HEADERS FROM 'file:///all_image_metadata.csv' AS row
WITH row
WHERE row.label IS NOT NULL AND row.image IS NOT NULL AND row.definition IS NOT NULL

MERGE (defect:DefectLabel {name: row.label})
MERGE (img:Image {name: row.image})
MERGE (defect)-[r:hasExample]->(img)
SET r.definition = row.definition;

MATCH (d:DefectLabel)-[r:hasExample]->(i:Image)
RETURN d, r, i LIMIT 10;
```

## C PDF-Based Dataset Construction Samples

This appendix presents an excerpt of image–text pairs extracted from multi-part technical PDF documents. The samples were generated using automated processing based on the PyMuPDF and pdfplumber

libraries. For each image detected in the PDF, the system attempted to extract the most relevant text from the surrounding page. If available, predicted defect labels and technical definitions were matched and stored using domain-specific keyword extraction and structured CSV formatting.

The table below shows a selection of samples from the dataset. Each row includes the image file name, the source PDF, page number, whether the image was damage-related, and the assigned defect label and technical definition. This structure was later used for downstream tasks such as knowledge graph construction and user interface development.

Table 2

*Sample of Labeled Image–Text Pairs Extracted from PDF Documents (excerpt)*

| **Image File** | **Source PDF** | **Pg** | **Damage Related** | **Label** | **Definition** |
|---|---|---|---|---|---|
| 2_Discontinuities_part3_page_4_img_1.jpeg | 2_Discontinuities_part3.pdf | 4 | False | Cold Tearing | C211 Cold Tearing – Dictated by ual stresses in the casting |
| 2_Discontinuities_part3_page_5_img_1.jpeg | 2_Discontinuities_part3.pdf | 5 | False | Cold Tearing | Usually square edged, trans-cryst free from oxidation |
| 2_Discontinuities_part4_page_4_img_1.jpeg | 2_Discontinuities_part4.pdf | 4 | True | Hot Tearing | Rugged, dendritic, close to hot perpendicular to shrinkage direct |
| 3_Surface Imperfections_part2_page_4_img_1.jpeg | 3_Surface Imperfections_part2.pdf | 4 | False | Surface folds | Folded portions distributed ov casting surface |
| 3_Surface Imperfections_part2_page_6_img_1.jpeg | 3_Surface Imperfections_part2.pdf | 6 | False | Elephant skin | Shallow grooves resembling ele wrinkles, common in ductile iron |
| 3_Surface Imperfections_part2_page_10_img_1.jpeg | 3_Surface Imperfections_part2.pdf | 10 | True | Seams or scars | Wavy furrows on surface, may r ble rat tail or kish graphite region |
| 3_Surface Imperfections_part2_page_13_img_1.jpeg | 3_Surface Imperfections_part2.pdf | 13 | False | Flow marks | Oxide film stripes that follov mould metal flow |
| 3_Surface Imperfections_part2_page_15_img_1.jpeg | 3_Surface Imperfections_part2.pdf | 15 | False | Rough casting surface | Caused by sand expansion; pits r bling sand grain profiles |
| 3_Surface Imperfections_part2_page_18_img_1.jpeg | 3_Surface Imperfections_part2.pdf | 18 | False | Severe roughness | Rougher than expected surface free from adhering sand |

# D Flask API Interface and Output Format

This appendix documents the output format generated by the Flask-based backend service after a user uploads a defect image for analysis. The backend system integrates three main components: a CNN image classifier, a Neo4j knowledge graph, and a JSON formatter.

When a user uploads an image via the front-end interface, the Flask API processes the request and performs two steps: (1) classifies the defect type using the pre-trained CNN model, and (2) queries the knowledge graph to retrieve the corresponding domain-specific defect definition. The combined result is returned as a JSON object, as illustrated below.

## D.1 Example JSON Output


```
{
  "filename": "2_Discontinuities_part4_page_4_img_1.jpeg",
```

```
  "defects": [
    {
      "label": "Hot Tearing",
      "definition": "C221 Hot Tearing - Definition
• Rugged, dendritic and intergranular in character
• Exterior and/or interior and often close to hot spots
• Perpendicular to main direction where casting resist shrinkage
• Tears can be filled with melt after crack formation
Adapted from [J. Campbell Solidification Defects in Castings TALAT Lecture 3207]"
    }
  ]
}
```

## D.2 Route and Protocols

The following RESTful routes were implemented in the backend using Flask:

- `POST /upload` – Upload the defect image file
- `POST /predict` – Classify defect type using CNN
- `POST /query` – Retrieve label definition from Neo4j KG

Communication with the Neo4j database is handled via the Bolt protocol using the official Python driver `neo4j`. The returned JSON structure is designed to be lightweight and machine-readable, allowing future integration with other manufacturing dashboards or inspection automation systems.