Joni Jokela

Optimizing Inventory Management: Leveraging CNNs and Conventional Grouping Methods

A Strategy for Excess Inventory Reduction and Dynamic Grouping

School of Technology and Innovations
Master's Thesis in Economics and Business Administration
Master's Programme in Industrial Management

Vaasa 2024

UNIVERSITY OF VAASA
School of Technology and Innovations
Author: Joni Jokela
Title of the Thesis: Optimizing Inventory Management: Leveraging CNNs and Conventional Grouping Methods : A Strategy for Excess Inventory Reduction and Dynamic Grouping
Degree: Master of Science in Economics and Business Administration
Discipline: Industrial Management
Supervisors: Ahm Shamsuzzoha & Petri Helo
Year: 2024 **Pages:** 100

ABSTRACT :

This thesis addresses gaps in existing literature by exploring the optimization of inventory management processes through the integration of convolutional neural networks and conventional grouping methods. Against the backdrop of increasing complexity in inventory management, the research aims to reduce excess inventory and enhance decision-making in warehouse operations. Utilizing a combination of convolutional neural networks and conventional grouping methods, the study investigates their effectiveness in classifying inventory items, identifying patterns, and improving overall inventory management efficiency.

The methodology involves training an image classification convolutional neural network model to classify items into categories based on their characteristics, thereby improving inventory management accuracy. Concurrently, conventional grouping methods such as ABC-XYZ analysis are applied independently to categorize items according to their value and demand variability. The integration of these approaches aims to provide a comprehensive understanding of inventory dynamics and enable more informed decision-making. The study data for the CNN database was collected by photographing products and naming them according to product reference codes. Additionally, 8 years of sales data were utilized for the ABC-XYZ analysis to compare it with the currently stocked inventory products.

The findings suggest that integrating convolutional neural networks and conventional grouping methods can enhance inventory management processes by improving accuracy in inventory classification and prioritization. This leads to reduced excess inventory, as well as improved resource allocation and decision-making in warehouse operations. By leveraging CNNs' ability to accurately classify items based on their characteristics and conventional grouping methods' effectiveness in categorizing items by value and demand variability, companies can optimize inventory levels and allocation strategies, ultimately resulting in more efficient warehouse operations.

This research contributes to the increasing literature on inventory management optimization and offers practical implications for businesses seeking to improve their inventory management practices. By leveraging advanced technologies such as CNNs and traditional methods like ABC-XYZ analysis, organizations can achieve greater efficiency, cost savings, and competitive advantage in today's dynamic marketplace. However, it is important to note that limitations such as data availability constraints and model generalizability may impact the applicability of findings to specific industry contexts.

KEYWORDS: Deep learning, inventory control, automation, classification, enterprise resource planning, image processing, spare parts

VAASAN YLIOPISTO
Tekniikan ja innovaatiojohtamisen yksikkö
Tekijä: Joni Jokela
Tutkielman nimi: Varastonhallinnan optimointi: CNN:n ja perinteisten ryhmittelymenetelmien hyödyntäminen : Ylimääräisen varaston vähentämisstrategia ja dynaaminen ryhmittely
Tutkinto: Kauppatieteiden maisteri
Oppiaine: Tuotantotalous
Työn ohjaajat: Ahm Shamsuzzoha & Petri Helo
Valmistumisvuosi: 2024 **Sivumäärä:** 100

TIIVISTELMÄ :

Tässä tutkielmassa puututaan olemassa olevan kirjallisuuden aukkoihin tutkimalla varastonhallintaprosessien optimointia yhdistämällä konvolutiivisia neuroverkkoja ja perinteisiä ryhmittelymenetelmiä. Varastonhallinnan monimutkaistumisen myötä tutkimuksen tavoitteena on vähentää ylijäämävarastoja ja tehostaa päätöksentekoa varastotoiminnoissa. Tutkimuksessa käytetään konvolutiivisten neuroverkkojen ja perinteisten ryhmittelymenetelmien yhdistelmää, ja siinä tutkitaan niiden tehokkuutta varastokohteiden luokittelussa, mallien tunnistamisessa ja varastonhallinnan kokonaistehokkuuden parantamisessa.

Menetelmässä koulutetaan kuvien luokitteluun tarkoitettu konvoluutio-neuroverkkomalli, joka luokittelee nimikkeet luokkiin niiden ominaisuuksien perusteella ja parantaa siten varastonhallinnan tarkkuutta. Samanaikaisesti sovelletaan perinteisiä ryhmittelymenetelmiä, kuten ABC-XYZ-analyysia, itsenäisesti erien luokittelemiseksi niiden arvon ja kysynnän vaihtelun mukaan. Näiden lähestymistapojen yhdistämisen tavoitteena on tarjota kattava käsitys varaston dynamiikasta ja mahdollistaa tietoon perustuva päätöksenteko. CNN-tietokannan tutkimusaineisto kerättiin valokuvaamalla tuotteita ja nimeämällä ne tuoteviitekoodien mukaisesti. Lisäksi ABC-XYZ-analyysissä hyödynnettiin 8 vuoden myyntitietoja, jotta niitä voitiin verrata tällä hetkellä varastossa oleviin varastotuotteisiin.

Tulokset viittaavat siihen, että konvolutiivisten neuroverkkojen ja perinteisten ryhmittelymenetelmien yhdistäminen voi tehostaa varastonhallintaprosesseja parantamalla tarkkuutta varaston luokittelussa ja priorisoinnissa. Tämä johtaa ylijäämävarastojen vähenemiseen sekä resurssien jakamisen ja päätöksenteon parantamiseen varastotoiminnoissa. Hyödyntämällä CNN:ien kykyä luokitella nimikkeet tarkasti niiden ominaisuuksien perusteella ja perinteisten ryhmittelymenetelmien tehokkuutta nimikkeiden luokittelussa arvon ja kysynnän vaihtelun mukaan yritykset voivat optimoida varastotasoja ja jakamisstrategioita, mikä johtaa lopulta tehokkaampiin varastotoimintoihin.

Tämä tutkimus täydentää varastonhallinnan optimointia koskevaa lisääntyvää kirjallisuutta ja tarjoaa käytännön vaikutuksia yrityksille, jotka haluavat parantaa varastonhallintakäytäntöjään. Hyödyntämällä CNN:n kaltaisia kehittyneitä teknologioita ja ABC-XYZ-analyysin kaltaisia perinteisiä menetelmiä yritykset voivat saavuttaa suurempaa tehokkuutta, kustannussäästöjä ja kilpailuetua nykypäivän dynaamisilla markkinoilla. On kuitenkin tärkeää huomata, että rajoitukset, kuten tietojen saatavuuden rajoitukset ja mallin yleistettävyys, voivat vaikuttaa tulosten soveltuvuuteen tiettyihin toimialaympäristöihin.

AVAINSANAT: Syväoppiminen, varastonvalvonta, automaatio, luokitus (toiminta), toiminnanohjausjärjestelmät, kuvankäsittely, varaosat

Contents

Figures

Tables

Algorithms

Abbreviations

SME	Small and Medium Enterprise
LE	Large Enterprise
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
CNN	Convolutional Neural Network
CAM	Class Activation Mapping
JIT	Just-In-Time
ReLU	Rectified Linear Unit

1 Introduction

Before delving into the specifics of the case company, it is important to understand the pervasive needs of inventory management, which encompass systematic categorization and visual representations of inventory items. Effective inventory management is essential for businesses to optimize resource allocation, minimize excess inventory, and enhance decision-making processes. To achieve these goals, various item classification methods are available, ranging from traditional grouping methods to emerging technologies like computer vision image processing. These methods serve a crucial role in streamlining inventory management processes and improving overall efficiency.

Now, transitioning to the case company, a small and medium enterprise (SME) specializing in maintenance activities, it encounters similar challenges in managing its diverse inventory due to limited human capital resources and storage space. The current inventory management practices lack systematic categorization and visual representations, resulting in excess inventory and inefficient resource allocation. Addressing these challenges requires enhancing transparency, automation, and organization within the inventory systems.

1.1 Purpose and Research Question

This research aims to improve inventory management practices at the case company by leveraging convolutional neural networks (CNNs) and traditional grouping methods. The goal is to automate and optimize inventory classification, particularly spare parts, while reducing manual intervention. The research question asks: *How the combination of CNNs, and conventional grouping methods can optimize inventory management processes to reduce excess inventory and costs, and enhance decision-making in warehouse operations?*

1.2 Research Approach and Limitations

A mixed research method was employed, combining a qualitative understanding of technological adaptation challenges with quantitative analysis using CNNs and ABC-XYZ. Deep supervised learning techniques were initially used for model training, followed by deep unsupervised learning for generalization evaluation. Class activation mapping (CAM) was utilized to interpret the model's decision-making process, emphasizing transparency and insight into the neural network's inner workings. The ABC-XYZ grouping method was employed to compare sales and stocked inventory data, aiming to gain a deeper understanding of their composition. Subsequently, recommendations were tailored for each category based on the findings.

As for the limitations, while the number of classes was limited to 20 for the model, it consisted mainly of all the active groups, thereby encompassing most inventory categories. Additionally, while generally well-established practices were followed during CNN construction, it is important to note that the architecture and hyperparameters are dependent on the complexity of the task and dataset size. Furthermore, the limited timeframe of sales data evaluation in the ABC-XYZ analysis may have impacted the comprehensiveness of the study.

1.3 Background and Summary of Study Gaps

Inventory management serves as a cornerstone of efficient supply chain operations, ensuring the seamless flow of goods from point of origin to final destination while balancing supply and demand dynamics. However, SMEs encounter unique challenges in managing their inventory due to limited resources and storage space constraints. These challenges often result in inefficient resource allocation, excess inventory, and suboptimal decision-making processes.

In response to these challenges, the integration of advanced technologies with traditional inventory management methods has emerged as a promising solution. CNNs, a form of artificial intelligence, have garnered attention for their potential to revolutionize inventory management processes by enabling automated classification, pattern recognition, and decision support. By leveraging CNNs alongside conventional inventory management strategies, SMEs can potentially enhance transparency, automation, and organization within their inventory systems.

While the potential benefits of integrating CNNs into inventory management practices are evident, existing literature reveals several gaps and limitations that warrant further investigation. Prior research has highlighted the need for a comprehensive analysis of inventory management policies, particularly focusing on item grouping strategies and technological adoption challenges faced by SMEs. Additionally, the implementation of CNNs in inventory management poses its own set of challenges and limitations, including issues related to data availability, model generalization, and resource constraints.

Despite the growing interest in leveraging CNNs for inventory management optimization, there remains a lack of empirical studies that systematically examine the effectiveness of CNNs in real-world SME contexts. Furthermore, existing literature predominantly focuses on theoretical frameworks and simulation models, with limited empirical evidence on the practical implications of integrating CNNs with traditional inventory management methods.

Therefore, this study seeks to address these gaps by conducting a comprehensive investigation into the integration of CNNs and conventional inventory management strategies within the context of SMEs. By examining the implementation challenges, performance outcomes, and practical implications of CNNs in real-world inventory management settings, this research aims to contribute to the existing literature on inventory management optimization and offer valuable insights for businesses seeking to enhance their inventory management practices.

In summary, this study aims to bridge the gap between theoretical frameworks and practical applications in inventory management optimization, with a specific focus on the role of CNNs and ABC-XYZ method in improving efficiency, reducing costs, and enhancing decision-making processes for SMEs.

1.4 Structure of the Thesis

This thesis adopts a structured methodology, commencing with Chapter 1, which establishes the study's background, research questions, objectives, and limitations. It also provides an overview of the thesis framework.

Chapter 2 conducts a comprehensive review of relevant academic literature, focusing on spare part management, conventional grouping methods, technological adoption for SMEs, and deep learning, particularly CNN architecture. The chapter explores challenges and limitations, establishing a robust theoretical foundation.

Chapter 3 shifts focus to research methodology, data collection, and pre-processing, involving two distinct phases. It leverages 4,000 captioned images, later enhanced to 20,000, to build a model following established practices. To conclude, a process flow is depicted, detailing the workflow from the warehouse operator to the inventory management system and CNN.

Chapter 4 similarly relies on research methodology and data collection, focusing on inventory stock and sales data spanning three years. Through comprehensive analysis, it seeks to understand price composition and sales history, laying the groundwork for strategic recommendations based on item classification and sales viability.

Chapter 5 discusses the managerial implications of the research. The CNN model enhances workflow productivity, while the ABC-XYZ classification simplifies decision-

making in spare parts management, optimizing inventory and reducing costs. These practices position companies for sustained success and competitive advantage.

The concluding Chapter 6 offers summarized findings and conclusions, addresses study limitations, identifies potential areas for future research, and reassesses the reliability and validity of the research and its outcomes.

2 Literature Review

The literature review begins by delving into inventory management, drawing insights from Singh and Verma (2018) and Williams and Tokar (2008). It elucidates the historical evolution of inventory management, emphasizing the importance of balancing supply and demand through efficient planning and execution. Traditional methods, originating in the 1920s, focused on determining economical purchase quantities, while recent research highlights the integration of logistics activities and collaborative inventory management practices.

Moving on to the management of spare parts inventory, Bacchetti and Saccani (2012) shed light on the challenges inherent in spare parts inventory management. They underscore the significance of classification analyses in prioritizing critical items and reducing costs associated with excess stock or system downtime.

Transitioning to the challenges of technology adoption in SMEs, insights from Vaaland and Heide (2007) reveal the disparities in technology adoption between SMEs and larger enterprises. Despite their agility, SMEs often lag behind in embracing technology-driven management methods, potentially risking their competitiveness.

Shifting focus to AI's impact on SMEs, Kopka and Fornahl (2024) and Albayrak Ünal et al. (2023) highlight the transformative potential of AI, ML, and DL technologies in enhancing productivity within SMEs. These technologies offer promising avenues for optimizing inventory management processes and improving overall operational efficiency.

In the realm of deep learning, Alzubaidi et al. (2021) demonstrate the versatility of DL techniques, showcasing their efficacy in scenarios where human expertise is limited, or explanations are elusive. They also elaborate on the architecture of CNNs, highlighting their effectiveness in tasks such as image processing and pattern recognition.

Lastly, addressing the challenges and limitations of CNN, insights from Alzubaidi et al. (2021) underscore the challenges encountered by CNNs, such as overfitting, vanishing gradients, and underspecification. Various regularization techniques and optimization algorithms are proposed to mitigate these challenges and enhance the robustness of CNN models.

2.1 Inventory Management

Inventory management, a vital component of supply chain management, involves strategically coordinating the movement, storage, and delivery of goods, services, and related information from point of origin to final destination, ensuring timely fulfillment of customer needs. It involves continuous planning, organization, and control of inventory to minimize investment while maintaining a balance between supply and demand. The process entails overseeing the procurement, storage, and accessibility of items to ensure an optimal supply without unnecessary stockpiling (Singh & Verma, 2018, p. 3868).

In Whitin's (1952) assessment on *Inventory Control in Theory and Practice*, the emphasis on inventory control gained traction due to the recognition of the significance of inventory control systems focused on determining economical purchase and reorder quantities, a formula independently arrived at by several authors in the 1920s.

Several factors contributed to the development of such formulas. Firstly, the expansion of business establishments, particularly large-scale enterprises, highlighted the potential savings achievable through improved inventory management. Secondly, the proliferation of business training, including courses in high schools, and the growth of business administration colleges; have enhanced awareness of optimization opportunities among business practitioners. Additionally, trade publications covering diverse business topics contributed to a deeper understanding of relevant challenges.
Thirdly, the increasing presence of engineers in business introduced a scientific approach to management, encompassing methods such as production control and cost accounting.

Lastly, the "inventory depression" of 1921 underscored the risks of excessive inventory accumulation, further driving research into inventory control methods. These developments led to the derivation of formulas for determining economic purchase quantities, involving elementary differential calculus applications to inventory management.

In their review of inventory management research spanning from 1976 to 2008, Williams and Tokar (2008) identified two predominant themes in the literature.

1. **Integration of Traditional Logistics Activities:** This theme mainly resorts to simulation and analytical models with the purpose of investigating compromises of warehouse management, storage, and logistics. While the continuous review inventory system approach is commonly assumed, periodic review inventory control models receive limited attention in the logistics literature (Williams & Tokar, 2008).
2. **Collaborative Inventory Management:** This theme explores the effect of cooperative inventory initiatives where customer satisfaction and inventory dedication effectiveness are concerned. Scholars advocate for increased focus on coordination and collaboration in logistics research, reflecting an evolving trend towards collaborative inventory management practices (Davis-Sramek & Fugate, 2007; Williams & Tokar, 2008).

Singh and Verma (2018) addressed several contemporary issues in inventory management, including maintenance policies, transportation costs, **holding costs**, and ordering costs. To alleviate these issues, performance improvement, system structure enhancement, and **technological integration** were proposed.

2.1.1 Management of Spare Parts Inventory

From operational and support perspectives, spare part demands are typically uncertain, and maintaining excess stock is costly. The provisioning of spare parts often encounters fluctuations throughout a system's life cycle. Inadequate provisioning of critical items can lead to expensive system downtime, while surplus stock can also incur significant costs (Yang & Du, 2004). Therefore, effective logistics strategies for spare parts serve a significant role in enhancing efficiency and reducing the costs of service processes. However, the ongoing challenge lies in the traditional procurement, storage, and provisioning of all spare parts based solely on intuitive assessment, thereby overlooking the individual characteristics of each spare part. Consequently, classification analyses are increasingly applied in practical contexts (Stoll et al., 2015).

Bacchetti and Saccani (2012) highlighted similar challenges in their research on spare part classification and demand forecasting for stock control. In which several factors contribute to the complexity of demand and inventory management for spare parts: the management of a high quantity of parts, intermittent or lumpy demand patterns, the need for high responsiveness to minimize downtime costs for customers, and the risk of stock obsolescence.

To address these challenges, Syntetos et al. (2009) proposed the categorization of SKUs to streamline decision-making processes. This categorization enables the selection of appropriate forecasting and stock control methods while allowing managers to prioritize their focus on the most critical stock-keeping units, however defined. Despite its importance, this issue has been largely overlooked in academic literature. Yet, it represents a significant opportunity for enhancing spare parts availability and reducing inventory costs.

2.2 Inventory Classification Methods

In Altay Guvenir's and Erel's (1998) study on *Multicriteria inventory classification using a genetic algorithm*, they examined the origins of the ABC classification method. Developed at General Electric in the 1950s, the conventional ABC classification is the most extensively utilized framework for classifying products. It is rooted in the Pareto principle, which posits that a small percentage of items (typically 5–20%) contribute to the majority of the overall value, while a larger portion (20–30%) contributes moderately, and the remaining items (50–75%) contribute minimally.

Under the classical ABC classification, items are ranked by their annual usage values, calculated by multiplying annual usage quantities by average unit prices. Class A comprises the top-ranking items that contribute the most to total usage value, while class C consists of lower-ranking items that contribute less. Items in between are classified as class B. The delineation of these classes is subjective, and organizations may adjust the number of classes based on their desired level of control differentiation.

In their comprehensive study, Stoll et al. (2015) emphasized the importance of extending the ABC categorization method into a more comprehensive model by incorporating additional criteria such as XYZ, FSN, or VED. These methods provide further insights into various aspects of inventory management, allowing for a more nuanced categorization approach tailored to specific needs and objectives.

XYZ analysis classifies items based on their demand variability. Items are categorized into "X" for low variability, "Y" for moderate variability, and "Z" for high variability. Whereas FSN represents "Fast-moving, Slow-moving, Non-moving." This analysis categorizes items based on their rate of movement in inventory. Items are labeled as "F" for fast-moving, "S" for slow-moving, and "N" for non-moving or rarely used. While VED analysis categorizes items based on their criticality for production or service delivery. In which products are denoted as "V" for vital, "E" for essential, and "D" for desirable but not critical. Figure 1 presents an integrated classification method for ABC-XYZ-VED.

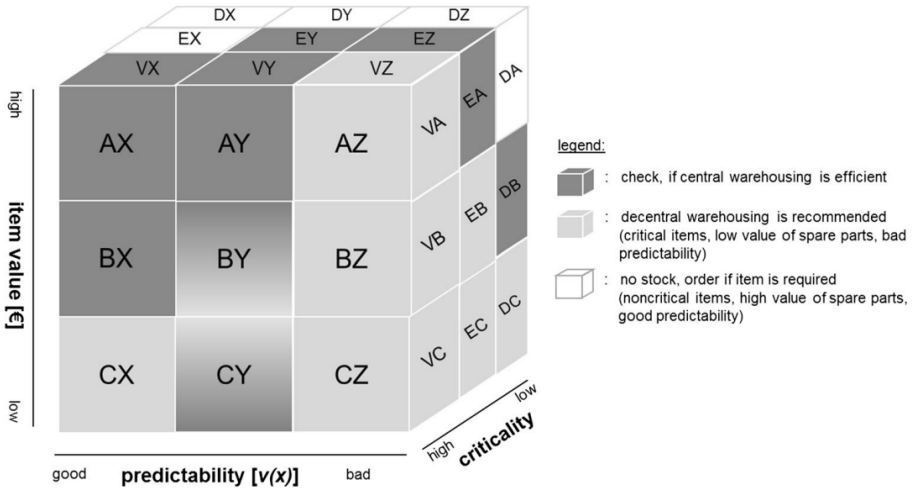


Figure 1. Integrated classification method comprising ABC-XYZ-VED (Stoll et al., 2015).

Different inventory policies are to be applied based on the dimensionality. With the following proposed by Stoll et al. (2015):

1. **X-parts**: These parts have high prediction accuracy and are recommended to be stored using a just-in-time (JIT) ordering strategy, as they are needed urgently.
2. **A-parts**: Akin to X-parts, A-parts should also be procured as needed to mitigate the capital expenditure.
3. **D-parts**: These parts present a minimal risk during manufacturing procedure and can be promptly obtained due to general availability in the sector. Therefore, they should not be stored.
4. **A–X–D-parts**: This combination of parts is particularly suitable for JIT ordering.
5. **B-parts**: These parts, which consist mainly of carry-over parts and standard machine components, are suitable for central warehousing due to their value and usability across different locations.
6. **E-parts**: Although E-parts necessitate storage owing to their critical nature, they do not warrant mandatory on-site storage. Therefore, it is recommended to centrally store them.

7. **Y-parts**: These components exhibit demand patterns that fluctuate regularly, rendering them appropriate for storage. However, their consistent demand profile makes them unideal for decentralized warehousing. Therefore, they should be stored in a central warehouse.
8. **B–Y–E-parts**: This combination of parts is ideal for the central warehouse storing.

2.3 Challenges of Technology Adoption in Small and Medium Enterprises

Vaaland and Heide (2007) observed in a cross-sectional study involving 200 SMEs and large enterprises (LE) from Norway that SMEs tend to allocate less attention to planning and control methods compared to their larger counterparts, LEs. One plausible explanation for this difference could be attributed to the organizational structures commonly found in larger enterprises. These arrangements frequently facilitate greater specialization, enabling distinct roles for supply chain management. Such specialization is deemed essential for the effective development, operation, and maintenance of planning and control systems.

Conversely, SMEs are often recognized for their agility and reduced bureaucracy. While it was anticipated that their entrepreneurial orientation and flexible organizational framework would foster an anticipatory approach favouring e-business implementations, the findings indicate a lack of focus on electronic-based methods among SMEs. Given the fundamental role of integrating actors and enhancing transaction efficiency in sustaining a competitive supply chain, there is concern that SMEs may encounter significant obstacles by ignoring technological enhancements on their control and planning systems. An implementation problem to address, as SMEs have their limitations within organizational, financial, and human resources.

In general, larger companies anticipate a more technology-driven future for their businesses, whereas SMEs' expectations are to a lesser extent. As a result, SMEs tend to lag in adopting and implementing critically deemed technology for sustainable supply chain

management operations. Consequently, SMEs are at significant risk of losing their competitive advantage. Similar findings were reported by Quayle (2003) in *a study of supply chain management practices in UK industrial SMEs*. The research found that new technology adaptation was deemed of the least significance. The study employed random stratified sampling, gathering 280 responses from various sectors including manufacturing, electrical, engineering, and construction.

2.3.1 AI's Impact on SMEs

Digitalization is pervasive across industries, with AI technology now at the forefront. While AI research dates back decades, contemporary progress within machine learning (ML) and deep learning (DL) have fueled its resurgence since 2013. This surge has led to the development of general-purpose ML tools, simplifying AI applications across various domains. AI, ML, and DL are not only perceived as emerging general-purpose technologies, in addition they represent a novel approach to innovation (Kopka & Fornahl, 2024, p. 63).

While AI's impact on inventive output is expected to be greater in larger firms due to their capacity for extensive investment in formal research and development (Rogers, 2004, pp. 142–143). Conversely, smaller firms, with their agile and less hierarchical structures, excel at recognizing and integrating opportunities (Goode & Stevens, 2000). This enables them to leverage AI implementation more effectively, facilitating productivity growth and narrowing the gap with larger firms. AI thus serves as an external catalyst for entrepreneurship (Obschonka & Audretsch, 2020, p. 10).

Kopka and Fornahl (2024) conclude that while the hypothesis suggesting productivity gains for small firms investing in AI application patents lacks confirmation in two-way interactions, the positive moderation effect of the distance to the frontier supports this hypothesis. Moreover, a pronounced leapfrogging effect is observed for firms distant from the technological frontier, particularly small ones with flat internal structures and

high flexibility. These firms are encouraged to aggressively invest in AI application patents to leverage the maximum benefits. However, small businesses may encounter imbalanced increase of costs due to their investments in highly skilled human capital, influencing their productivity outcomes. Conversely, larger firms often exhibit contrasting patterns due to their slower adaptation resulting from hierarchical structures.

In Albayrak Ünal et al.'s (2023) systematic review on AI applications in inventory management, advancements in artificial intelligence, driven by improvements in computer hardware, are revolutionizing inventory management processes. Sub-branches of AI, such as ML and DL, are central to this transformation. Recent research highlights a growth in interest in ML and DL techniques, enabling swift analysis of extensive datasets and refining demand forecasting accuracy. The integration of AI techniques into inventory management fosters enhanced efficiency, flexibility, cost-effectiveness, and rapid customer response.

2.4 Deep Learning

Recently, deep learning has increasingly become the predominant computational approach in machine learning, delivering exceptional results across various complex cognitive tasks; and often surpassing human performance. One key advantage of DL is its capacity to effectively process vast amounts of data. The field of DL has experienced rapid growth and has been widely utilized to address numerous traditional applications. Importantly, DL has demonstrated superior performance compared to established ML techniques spanning various domains with applications such as pattern recognition of images, audio, and text. The relationship between DL, AI, and ML is illustrated in Figure 2 (Alzubaidi et al., 2021, pp. 1–5).

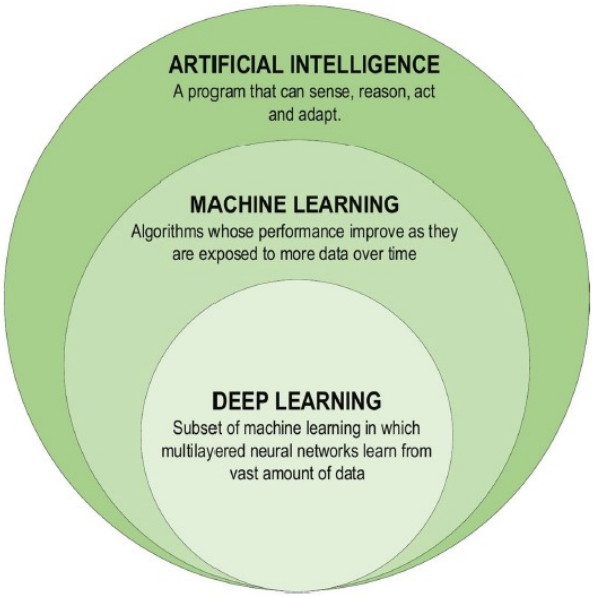


Figure 2. Dependency of DL on ML and AI (Alzubaidi et al., 2021, p. 7).

According to Alzubaidi et al. (2021, pp. 8–10), the application of deep learning is warranted in various scenarios where machine intelligence can match or even surpass human expertise. This indicates that deep learning can solve issues such as:

- When the expertise of humans is not available.
- Instances in which humans struggle in articulating the rationale behind their decisions, such as linguistical comprehension, medical diagnoses, and voice recognition.
- Scenarios necessitating a solution capable of adapting and evolving over time, such as in forecasting prices, stock preferentialism, weather, and tracking.
- Situations demanding personalized solutions, such as in biometrics or customization.
- Situations involving extremely large datasets that exceed human capacity for analysis, such as in sentiment analysis, targeted advertising on social media platforms, and calculating webpage rankings.

Deep learning's versatility allows it to excel across a wide range of application domains, earning it the reputation of universal learning. Unlike traditional methods, deep learning

techniques autonomously learn optimized features tailored to specific tasks, resulting in robust performance even with changes in input data. Deep learning models can also be applied across different data types and applications through transfer learning, making them valuable in scenarios with limited data availability. Deep learning architectures, such as ResNet, can scale to thousands of layers, enabling their deployment at a supercomputing scale.

Deep learning techniques are categorized followingly:

- Supervised learning: Training using labeled data, with preassigned input features and output labels.
- Semi-supervised learning: Combining labeled and unlabeled data during training, with the model learning from labeled data in a supervised manner while utilizing unlabeled data to improve performance.
- Unsupervised learning: Learning patterns and structures directly from input data without explicit supervision from labeled examples, aiming to discover inherent patterns or representations within the data for various tasks (Alzubaidi et al., 2021).

2.4.1 DL Networks

As for the network types, Alzubaidi et al. (2021, pp. 11–14) list three variants:

1. Recursive neural networks are categorized as neural networks tailored for processing structured data such as trees and graphs by recursively applying the same set of weights to child nodes.
2. Recurrent neural networks are categorized as neural networks tailored to process sequential data by introducing loops within the network, allowing information to persist over time.

3. Convolutional neural networks are categorized as deep neural networks designed for processing grid-like data, such as images, by leveraging convolutional layers to automatically learn hierarchical representations.

In the realm of deep learning, CNN stands out as the most widely utilized algorithm, primarily attributed to its capability to autonomously identify relevant features without requiring supervision. Figure 3 depicts the human error rate on ImageNet's dataset, which includes 14,197,122 images across 21,841 classes, known as synsets (ImageNet, 2021).

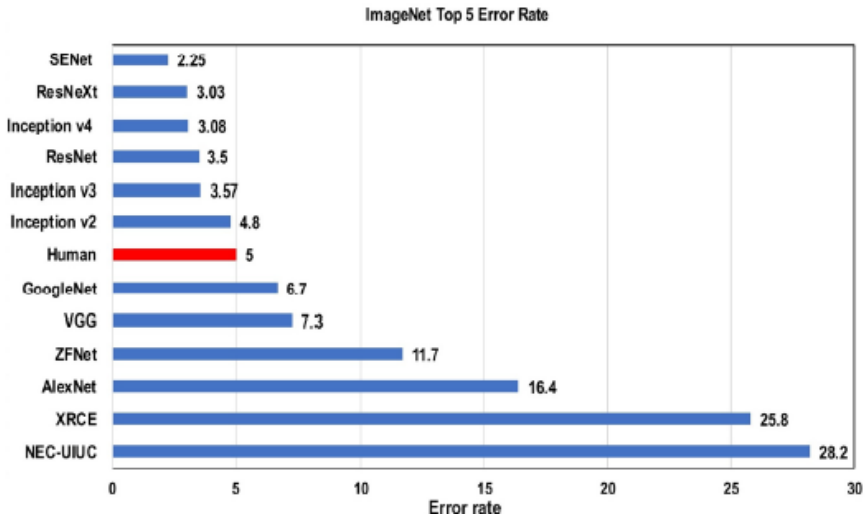


Figure 3. The performance of DL models and human recognition on the ImageNet dataset (Alzubaidi et al., 2021, p. 8).

In this context, accuracy is determined by the top 5 predictions. When dealing with numerous classes or specialized labeling, models tend to achieve higher accuracy. The illustrated models have undergone training using colored (RGB) images with input sizes ranging from 32x32 to 320x320. (Alzubaidi et al., 2021. p. 27). With the networks depicted, the following chapter specifically explores the architecture of a CNN.

2.5 CNN Architecture

A frequently employed CNN type comprises multiple convolution layers followed by subsampling (pooling) layers, with fully connected layers at the end. In a CNN model, the input *x* of a layer is three dimensional: height, width, and depth, denoted as *m* * *m* * *r*, where *m* represents both height and width. The depth, referred to as the channel number, is *r*. Colored (RGB) images require three channels, whereas grayscale images require only one.

Each convolutional layer contains multiple kernels (filters), denoted as *k*, with dimensions *n* * *n* * *q*, resembling the input image, where the dimension *n* is smaller than *m*, and the depth *q* equals to or is less than the channel value. These kernels generate feature maps h^k by convolving with the input. The convolutional layer calculates dot products between its input and weights, processing smaller regions of the original image. Following this, a nonlinearity, also known as an activation function, is applied to the convolutional layer's output to produce the final result as depicted in (1) (Alzubaidi et al., 2021, pp. 15–20).

$$h^k = \mathrm{f}\left(W^k * \mathrm{x} + \mathrm{b}^k\right) \tag{1}$$

Where:

b^k represents the bias term,

W^k denotes the weight term.

The subsequent phase involves downsampling each feature map within the pooling layers, which reduces network parameters and accelerates training while mitigating overfitting. Adjacent areas of size p * p undergo either max or average pooling functions, where p represents the kernel size. Subsequently, dense layers handle mid- and low-level features, extracting high-level abstractions akin to those in traditional neural networks. Finally, the last-stage layers produce classification scores via support vector machines or softmax activation, with each score indicating the probability of a given class for a particular instance. Cross-entropy, also known as the softmax loss function, is a popular

choice for assessing the performance of CNNs. Here, (2) illustrates the class probability distribution, while (3) represents the cross-entropy loss function (Alzubaidi et al., 2021, pp. 15–20).

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e_k^a} \tag{2}$$

Where:

e^{ai} denotes the output of the previously unadjusted layer,

N denotes the output layer's quantity of neurons.

$$H(p, y) = -\sum_i y_i \log(p_i) \quad where \quad i \in [1, N] \tag{3}$$

CNN's architecture typically comprises the following layers:

1. **Convolutional Layer:** This layer applies filters (kernels) to the input data to detect features such as edges, textures, or patterns. Each filter produces a feature map by convolving across the input image.
2. **Activation Layer:** Following convolution, an activation function (such as ReLU) is applied to integrate non-linearity, enabling the network to learn complex patterns and relationships from the data.
3. **Pooling Layer:** Its purpose is to downsample the features extracted from the convolutional layers, decreasing dimensionality whilst preserving important features. Common pooling methods include max pooling and average pooling.
4. **Fully Connected Layer:** Also known as the dense layer, its purpose is to connect the adjacent layer's neurons, allowing the network to learn high-level features and make predictions.
5. **Output Layer:** The final layer of the network produces the desired output, such as class probabilities in classification tasks. Activation functions such as softmax

are often used to normalize the output into a probability distribution (Alzubaidi et al., 2021, pp. 15–20).

Figure 4 illustrates the convolution operation applied to an RGB image using a 3x3 kernel size.

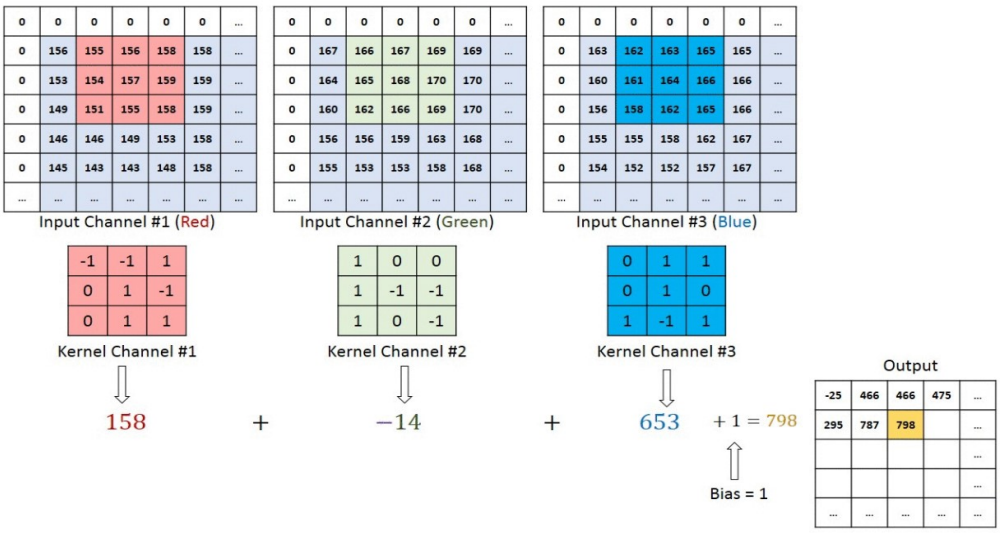


Figure 4. Convolution operation with RGB channel & 3x3 kernel size (Saha, 2018).

Thus, the scanning process iterates until the entire image has been traversed with a stride value of one, unless specifically defined. From each scan range, the input values are multiplied with respect to the values of the kernel channel; thereafter, they are summed together along with the bias and passed to the convolved feature. An uneven kernel size is preferred, as the surrounding pixels from the previous layer are symmetrically positioned around the output pixel, as demonstrated in Figure 5.

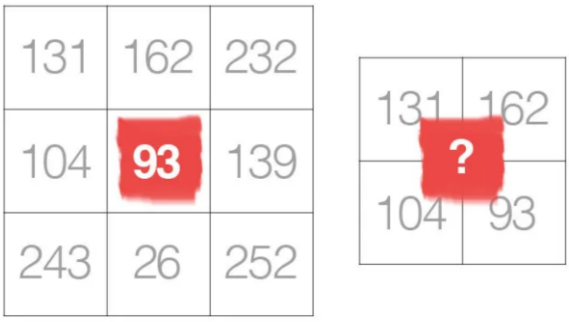


Figure 5. Elaboration of uneven and even-sized kernels (Sahoo, 2022).

The prevalence of traditional 3x3 filter sizes is acknowledged in a study by Wu et al. (2019), which aimed to evaluate the fundamental challenges associated with even-sized kernels. As for the activation layer, Rectified Linear Unit (*ReLU*) is the most widely utilized function in CNNs, which transforms all input values to positive numbers. Its primary advantage over other functions is its lower computational burden. Mathematically, it is represented as shown in (4) (Alzubaidi et al., 2021, p. 18).

$$f(x)_{ReLU} = max(0, x) \tag{4}$$

The outcome of a pooling layer depends on whether a max pooling or average pooling is defined, with Figure 6 depicting the differences.

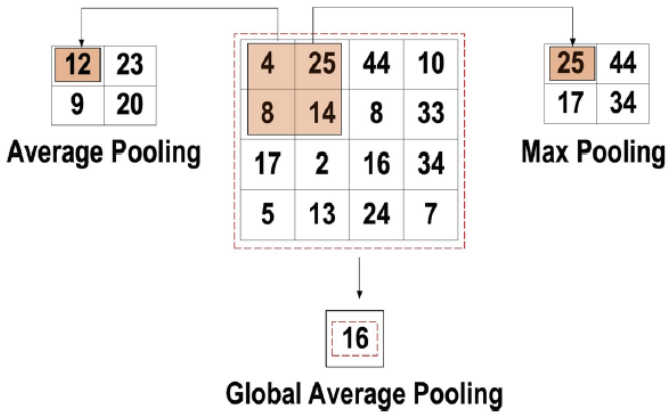


Figure 6. Function of max and average pooling (Alzubaidi et al., 2021, p. 17).

As can be inferred from the names, max pooling is responsible for passing forward the maximum value of the concerned section. Average pooling passes the average value of the respective section, whereas global average pooling passes the average value of the given block. Prior to dense layer implementation, a common practice is to introduce a flattening layer that simply converts a 3D tensor into a 1D tensor, i.e., a 3x3 tensor is transformed into a 9x1 tensor. The dense layer connects neurons between adjacent layers, with the final dense layer linking to the output layer, which returns class probabilities (Alzubaidi et al., 2021, pp. 15–19). Figure 7 illustrates the complete CNN structure.

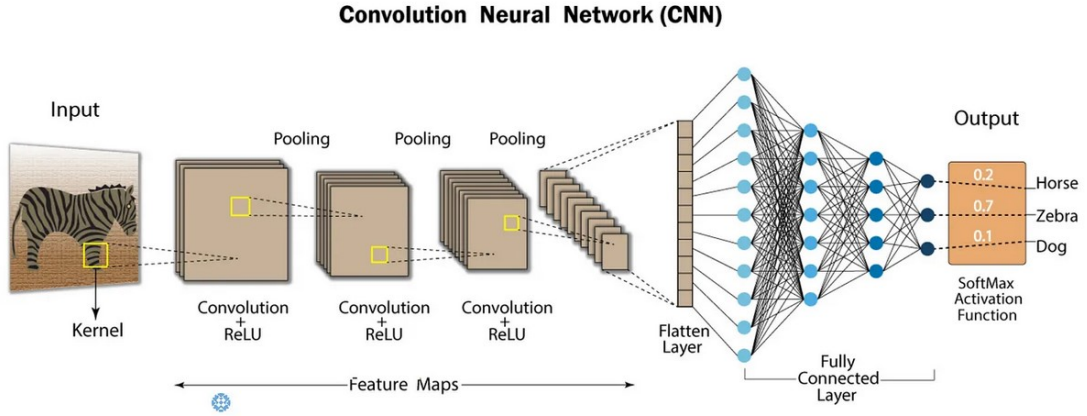


Figure 7. Composition of a CNN (Wen et al., 2020).

While a dropout layer is missing from the figure, it is an optional, yet a recommended regularization technique to be introduced to convolutional or dense layers with the purpose of mitigating overfitting. Which brings us to the next topic.

2.5.1 Challenges and Limitations of CNN

In CNN models, overfitting poses a significant challenge for achieving robust generalization. Overfitting arises when the model exhibits high accuracy on the training dataset but struggles with unlabeled test data. Conversely, an underfit model fails to adequately learn from the training data. A model that performs well on both training and testing

data is considered "well-fitted". To provide a form of regularization, the following methods can be applied:

- **Dropout**: A regularization method employed during neural network training to mitigate overfitting by randomly dropping (setting to zero) a certain proportion of neurons.
- **Drop-weights**: This technique closely resembles dropout. Instead of dropping neurons, drop-weights involve dropping the connections between neurons (weights) in each training epoch.
- **Data augmentation**: This method artificially enhances the training data by employing various transformations to the existing data. These transformations include, but are not limited to, color shifts, blurring, rotation, scaling, and cropping.
- **Batch normalization**: Is a technique used to improve the training speed and stability of neural networks by normalizing the inputs of each layer, usually after the activation function. It helps alleviate issues such as internal covariate shift and vanishing gradient, while also allowing for faster convergence during training (Alzubaidi et al., 2021, pp. 20–21).

The learning process in neural networks involves two main considerations: firstly, selecting an appropriate learning algorithm (optimizer), and secondly, employing various enhancements such as AdaDelta, Adam, and momentum to improve performance.
Loss functions are fundamental in supervised learning, aiming to minimize the error between predicted and actual outputs by adjusting learnable parameters such as biases and weights.

Gradient-based learning techniques are commonly used in CNN, ensuring parameter updates across all training epochs to seek locally optimized solutions and minimize errors. The learning rate, determining the step size of parameter updates, plays a crucial role in the training process. Careful selection of the learning rate is essential to avoid adverse

effects on learning, despite being a hyperparameter. Each training epoch involves a complete iteration through the entire training dataset, emphasizing the need for strategic learning rate choices to optimize the learning process effectively.

Adam is a cutting-edge optimization technique for deep learning, designed specifically for neural network training. It offers memory-efficient and computationally light advantages by adapting learning rates for each parameter in the model. Combining features of Momentum and RMSprop, Adam scales learning rates using squared gradients like RMSprop and employs a moving average of gradients akin to Momentum, with its function depicted in (3) (Alzubaidi et al., 2021. pp. 22–24).

$$w_{ijt} = w_{ijt-1} - \frac{\eta}{\sqrt{\widehat{E[\delta^2]^t} + \in}} * \widehat{E[\delta^2]^t} \tag{3}$$

To enhance model performance, the author suggests employing data augmentation techniques or transfer learning when dealing with limited dataset sizes. Additionally, extending training duration by adjusting batch size, increasing model depth or width through additional convolutional or dense layers, applying regularization techniques, and fine-tuning hyperparameters are recommended strategies.

Regarding hyperparameters, selecting an optimal split ratio for the datasets is crucial for model evaluation, with each serving a distinct purpose:

- **Training set**: Used to train or fit the parameters.
- **Validation set**: During validation, the model's performance is assessed using a separate dataset, ensuring an unbiased evaluation without influencing training.
- **Test set**: Used to objectively evaluate how well a final model fits the training dataset. This set is not used in any part of the model fitting or parameter tuning process (Ripley, 2005, p. 354).

CNNs encounter several limitations, including but not limited to challenges posed by limited dataset size, which can hinder model generalization. Transfer learning offers a solution by leveraging pre-trained models on larger datasets. Imbalanced data distribution can skew model performance, necessitating techniques such as data augmentation and class balancing. Model compression is also crucial for resource-constrained environments. Whereas more notable issues are:

1. **Vanishing gradient problem**: This phenomenon occurs during training when the gradients of the loss function diminish significantly as they traverse backward through the network layers. It particularly affects deep networks with many layers, causing early layers to receive very small gradient updates, leading to slow or stalled learning.
2. **Underspecification**: This refers to the phenomenon where multiple parameter configurations can adequately fit the training data, resulting in ambiguity in the learned model. Underspecification can lead to poor generalizations on practical implementations (Alzubaidi et al., 2021, pp. 41–50).

Alzubaidi et al. (2021, p. 63) conclude that DL relies on substantial datasets, preferably labeled, for predicting unseen data and training models. Real-time data processing and limited datasets pose significant challenges for this task. Additionally, the performance of CNNs is heavily influenced by hyperparameter selection, where even slight alterations in hyperparameter values can significantly impact overall CNN performance. Thus, careful parameter selection is crucial during the optimization. Furthermore, effective training of CNNs demands impressive and robust hardware resources such as GPUs. These resources are not only essential for training CNNs effectively but also for exploring the potential efficiency of using CNNs on automated and embedded platforms.

2.6 Identified Gaps in the Literature Review

Based on the literature review, four prevalent themes were identified:

Integration of Deep Learning Techniques in Inventory Management: While the literature acknowledges the potential of deep learning, particularly CNNs, in enhancing inventory management processes, there is a lack of comprehensive studies that specifically address the integration of these techniques into real-world inventory systems, especially within SMEs. Existing research tends to focus on theoretical frameworks and proof-of-concept studies rather than practical implementations and evaluations in operational settings.

Consequently, a gap exists in understanding the practical challenges, limitations, and effectiveness of implementing deep learning techniques, such as CNNs, particularly within SMEs facing limited resources and expertise in advanced technologies.

Optimal Item Classification Methods for Spare Parts Management: While various item classification methods, such as ABC-XYZ analysis, have been proposed and widely used in inventory management, there is limited research on identifying the most suitable classification methods for spare parts management, particularly in industries with complex and diverse spare parts inventories. Existing literature provides insights into the application of traditional classification methods but lacks comprehensive evaluations of their effectiveness in addressing the unique challenges of spare parts management, such as intermittent demand patterns, high inventory variability, and criticality considerations.

Thus, a gap persists in research specifically aimed at identifying and evaluating optimal item classification methods tailored to spare parts management's characteristics and requirements, which could potentially enhance inventory control and resource allocation.

Technology Adoption Challenges in SMEs: Despite the growing importance of technological adoption for enhancing competitiveness and operational efficiency, particularly in

SMEs, there is limited research on understanding the specific challenges and barriers faced by SMEs in adopting and integrating advanced technologies, such as deep learning and artificial intelligence, into their inventory management practices. Existing studies often overlook the unique organizational structures, resource constraints, and risk aversion tendencies prevalent in SMEs, which significantly impact their ability to adopt and implement complex technological solutions.

Therefore, a gap exists in research aimed at identifying and addressing technology adoption challenges specific to SMEs, which could offer valuable insights and practical recommendations for successfully integrating advanced inventory management technologies into SMEs.

Evaluation of Deep Learning Models in Real-world Inventory Systems: While several studies have proposed and evaluated deep learning models, such as CNNs, for inventory management applications, there is a lack of comprehensive evaluations of these models in real-world inventory systems, particularly in diverse industrial settings with varying inventory characteristics and operational requirements. Existing research often relies on simulated or laboratory-based experiments, which may not fully capture the complexities and uncertainties present in real-world inventory environments.

Consequently, a gap persists in research focusing on conducting rigorous evaluations of deep learning models in real-world inventory systems, offering valuable insights into their practical feasibility, performance, and scalability across different industries and operational contexts.

These identified study gaps highlight the need for further research to address specific challenges and limitations in inventory management, particularly in the context of integrating advanced technologies, optimizing item classification methods, understanding technology adoption dynamics in SMEs, and evaluating the practical effectiveness of deep learning models in real-world inventory systems. Addressing these gaps could

contribute to the development of more robust and effective inventory management strategies, ultimately leading to improved operational efficiency, cost savings, and competitive advantage for organizations. Considering these insights, the study now transitions to the practical aspect, where the process of constructing and optimizing the convolutional neural network model begins.

3 Development of CNN Models

In this chapter, the initial requirements, data preparation, and established practices are presented to ensure the development of a robust convolutional neural network model capable of generalizing to unseen data and integration into the inventory management system.

3.1 Requirements Analysis

Depending on the pre-existing computing capabilities, enterprises may or may not have access to suitable specifications. However, the cloud-based Google Colaboratory enables users to utilize the Jupyter Notebook to execute Python code, and any potential library dependencies without having to install them locally.

In this specific scenario, the primary user is typically a warehouse operator. However, depending on the organizational structure, the user applications may vary. As the model's purpose is to assign labels to products based on their features, it streamlines the receiving process by minimizing the need for the operator to manually assign a category during the identification and classification phase. The closest benefiting stakeholders include the purchasing and sales teams, data management specialists, and inventory managers. The forthcoming chapters detail the methodologies and practices employed to construct a model that fulfilled the requirements outlined in Table 1.

Table 1. Requirements of the CNN model.

Functional Requirements	Non-functional Requirements
Input processing	Performance
Feature extraction	Reliability
Classification	Accuracy
Evaluation	Usability
Deployment	Transparency

The model begins by preprocessing images, which involves resizing, normalizing, auto-tuning, and applying data augmentation techniques to enhance the quality and diversity of the dataset. In the feature extraction phase, the architecture of the model is optimized by determining the types and quantities of layers, as well as the sizes of kernels, to extract meaningful features from the images. The classification phase is responsible for configuring the fully connected layers, specifying the number of neurons in each layer, setting the output layer to match the total number of classes, and defining the loss function used to compute the error between predicted and true labels. During evaluation, the model's accuracy was assessed using appropriate metrics. Upon deployment, predictions for unseen data were generated by the model, and the results were output to an Excel file, including the file name, class integer, and confidence percentage for each prediction.

For the non-functional requirements, the architecture is designed to exhibit robust performance in computational time and scalability. It can handle a fivefold increase in dataset size while maintaining efficiency. Accuracy encompasses classification accuracy, generalization capability to unseen data, and prevention of overfitting. In terms of reliability, Python and TensorFlow generate warning and error messages for debugging measures. Usability is ensured through the ease of use of the model, complemented by comprehensive training documentation. The entire model can be compiled into an executable file, simplifying the process of training and predictions. Finally, class activation mapping is employed to evaluate the model's progress and emphasize relevant features during classification tasks. This enhances transparency and interpretability, ensuring that the model progresses in the correct direction.

In the evaluation of the camera for image capture, a generic cell phone with 48-, 50-, and 8-megapixel cameras was chosen. Despite downsizing the images to 512x512 pixels and smaller, the significance of initial image quality is to be considered, taking into account factors such as noise, color profiles, lighting, and blur. Additionally, downsizing can introduce its own effects depending on the interpolation method used. TensorFlow

defaults to bilinear interpolation for resizing, unless the images are already at the desired resolution. For scenarios involving a stationary camera, occasional variation of the zoom range is to be considered to enhance the model's ability to generalize features. Regarding the inventory system, a bespoke solution has been developed using Visual Basic, operating on Microsoft SQL Server. This system encompasses purchase and sales order data, product information, storage locations, as well as details of suppliers and customers, as depicted in Figure 8.

	Date of order	Purchasing account	Purchase order number	Product reference	Purchase volume	Unit	Cost price	Storage location
7	26.3.2024	Vendor 1	69124695	9999	1,00			
8	26.3.2024	Vendor 1	69124695	111	1,00			
9	18.3.2024	Vendor 2	6995442	36363	1,00			
10	18.3.2024	Vendor 2	6995442	IEC213	2,00			
11	23.1.2024	Vendor 3	313405	MK3-1UF	1,00			
12	22.3.2024	Vendor 4	22040364	INAKH1630..	2,00			
13	21.3.2024	Vendor 4	22040363	1LE1001-0...	2,00			
14	7.3.2024	Vendor 4	22040256	193144	2,00			
15	27.3.2024	Vendor 5	672428	10072	1,00			
16	27.3.2024	Vendor 5	672428	FN2090-10-..	1,00			
17	27.3.2024	Vendor 5	672428	HE00EJ6D...	1,00			
18	27.3.2024	Vendor 5	672428	HVE23AA...	1,00			
19	27.3.2024	Vendor 5	672428	HVPS06A...	1,00			
20	27.3.2024	Vendor 5	672428	EM1-C-M-4...	1,00			
21	27.3.2024	Vendor 5	672428	ED1F-PN-0..	1,00			
22	19.3.2024	Vendor 6	347079401	1006	1,00			
23	19.3.2024	Vendor 6	347079401	8760021	1,00			

Figure 8. Inventory management system layout**.**

As there is no pre-existing categorization in place, the model's purpose is to generate the class integer and name into the inventory management system with two additional columns allocated. The next chapter covers the preparation phase for compiling the model.

3.2 Database Compiling and Labeling

To initialize, the quantity of classes should be determined, as the complexity of the model is directly affected. In this case, 20 classes were chosen for items that can be considered active presently or in the foreseeable future. While there is no universally optimal minimum quantity of images a class should contain due to varying applications, ImageNet's synsets typically consist of 500–1,000 images (Deng et al., 2009, p. 249).

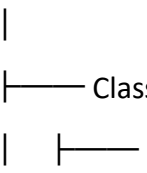
Initial testing commenced with 200 images per class to evaluate the generalization potential. With the weights normalized between classes, the model is unbiased against any specific class due to even samples.

Inherent to the classification task are possibilities of mislabeling, stemming either from human pre-processing procedures or machine learning errors during the training process; a simplified example is depicted in Table 2.

Table 2. Confusion Matrix for classification of X and Y (Adapted from *Google for Developers*, 2024).

	Predicted X	Predicted Y
Actual X	True Positive	False Negative
Actual Y	False Positive	True Negative

As each image was chosen to adhere to the class quota, human error is thus effectively ruled out, provided that the images are correctly archived in their respective categories. Where possible, subjects were photographed on white paper to provide a standardized method and minimize background noise that could potentially interfere with the learning process of the model. In addition, each product was captured from five different angles and with varying zoom, so that the model focuses on distinctive features prominent to each class. As the model aims to automate the categorization process of an inventory during material data creation, it is reliant on reference codes. Hence, an Excel file with individual tabs was created for each category, housing the corresponding identifiers. Later, it was used to match the images with their respective identifiers through a script presented in Appendix 1. For this specific script, the following folder structure was used:

Main folder


```
│   ├──── image-2.jpg
│   └──── image-3.jpg
│
├──── Class 2
│   ├──── image-1.jpg
│   ├──── image-2.jpg
│   └──── image-3.jpg
│
└──── Class …
    ├──── image-1.jpg
    ├──── image-2.jpg
    └──── image-3.jpg
```

This is to ensure consistency, as the model relies on the hierarchy to load images for further processing. Since the images retained their original resolutions, they were resized to 256x256 pixels. The resizing can be performed separately before loading the database into the model or by defining the width and height ratio as part of the model. It should be stressed that opting for a lower or higher resolution has its trade-offs. Lower resolution not only reduces the computation time; but also reduces prominent features that can be vital for effective learning of the model. While higher resolution retains the features better, the computing time increases; opting from 256x256 to 512x512 quadruples the resource requirements.

For example, the CIFAR-10 dataset with 60,000 images; and 32x32 pixels has achieved a state-of-the-art accuracy of 99.5% with certain models (Dosovitskiy et al., 2021, p. 6), although the minimal resolution size would greatly impact the human classification judgement. For this specific purpose and the potential confusion between the classes, a higher resolution size is preferred. Figure 9 depicts the varying resolutions of the two class comparisons, which have been rescaled to a uniform size to illustrate differences in quality.

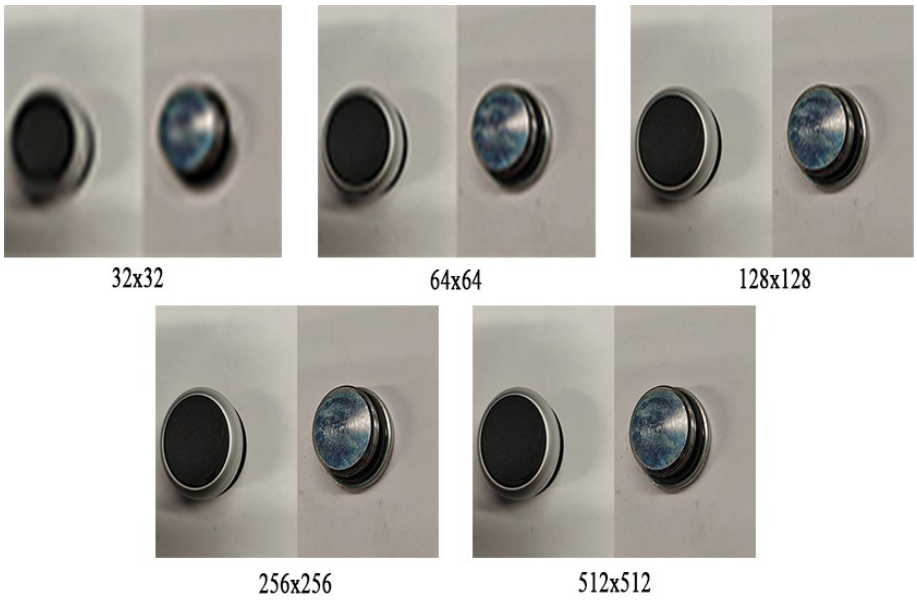


Figure 9. Impact of image resolution on classification.

In the case of the 32x32 resolution, it is particularly noticeable that the lighting conditions alone produce an aura of noise around the actual item. This could lead the model to mistakenly associate it as an outline during feature extraction. In a recent study by Thambawita et al. (2021) on a model's achieved accuracies of medical image resolutions, it can be noted that 256x256 and 512x512 resolutions performed the best, with marginal differences in-between the MCC scores. Bearing in mind that the increase in resolution size is directly affecting the computing time, and previously adjusted hyperparameters may not apply anymore due to the increased complexity.

3.3 Initial Model

While the common consensus is that AI applications necessitate a GPU equipped with a substantial number of tensor cores—specialized cores optimized for mixed precision training—this prerequisite does not uniformly apply across all deep learning models. Akin to Occam's razor, a simpler model in terms of total parameters is often preferred, given that it can produce similar accuracy.

TensorFlow's CPU variant was used to generate this initial model, followed by a GPU variant model to handle the extra workload of the enhanced database. The system specifications used for conducting the testing included an AMD Ryzen 5 7600X for the CPU, 32 GB of RAM, and an NVIDIA GeForce RTX 4090 for the GPU. Given the absence of an industrial-specific dataset containing a wide array of mechanical and electrical components, the utilization of transfer learning was deemed infeasible.

3.3.1 Database Preprocessing

Prior to loading the database, the initial parameters are defined along with the directory (Adapted from TensorFlow, 2024a):

```
batch_size = 32
img_height = 256
img_width = 256
data_dir = r'path:\to\main\folder\location'
seed = 123
```

Establishing a seed value ensures a certain degree of reproducibility in the obtained results. Minor differences naturally occur due to the randomness of potential data augmentation means, hardware and software differences, and the chosen optimization method. Batch size by default is 32 and has proven to yield the most optimal learning capabilities for this model. Altering the batch size is likely subject to alter the initial learning rate to accommodate for the change. As in, batch size 16 can suffice with a lower learning rate, whereas batch size 64 and upwards requires a higher learning rate along with increased memory requirements due to parallel processing. According to studies by Kandel and Castelli; (2020) and Masters and Luschi; (2018), it was noted that batch sizes 32 and lower provided the most optimal performances.

As the initial model employs a train–validation–test split of 70%–20%–10% respectively, a slight adjustment must be applied to process the splits into TensorFlow datasets due

to the lack of a built-in function for a three-way split. Lines 18–81 in Appendix 2 depict the loading and preprocessing phases. To ensure that no duplicates are present in the splits, the paths can be called to return the contained images with the following script (Adapted from Python, n.d.):

```
print("Filenames in x_paths:")
for filepath in x_paths:
    print(filepath)
```

Whereas x is to be replaced either with *train*, *val*, or *test* for this example. In the provided script, it can be noted that the dataset is shuffled with respect to the seed number, indicating that it would be reproduced in an exact manner if reloaded into the model. It is a common practice to ensure a wide representation of class respective items. In a hypothetical scenario, consider a class 'X' comprising 100 images processed in chronological order. The initial 70 images exhibit similar characteristics despite being different variants, while the remaining 30 images possess distinct features compared to the first 70. However, due to the splitting of these images, the model may struggle to accurately classify images within the same class.

To confirm that the labels were correctly extracted from subfolder names and assigned, the following script can be used (Adapted from TensorFlow, 2024a):

```
class_names = train_ds.class_names
print(class_names)
num_classes = len(class_names)
```

Whereby 'num_classes' equals 20, derived from the length of the array containing subfolder titles. It should be noted that during the creation of TensorFlow datasets, the channel value is specified. A channel value of 1 denotes the model's expectation of grayscale images, while a channel value of 3 indicates the expectation of RGB images. The RGB channel is deemed non-essential for this study, given the wide range of colors present within most classes, indicating that the model should not directly associate a specific color with a certain class. Despite this, the incorporation of the RGB channel

enhanced the model's learning performance, although it directly contributed to the complexity of the model. By plotting images from any of the splits, it can be noted that the pixel values are in the range of 0–255, as is evident in Figure 10.

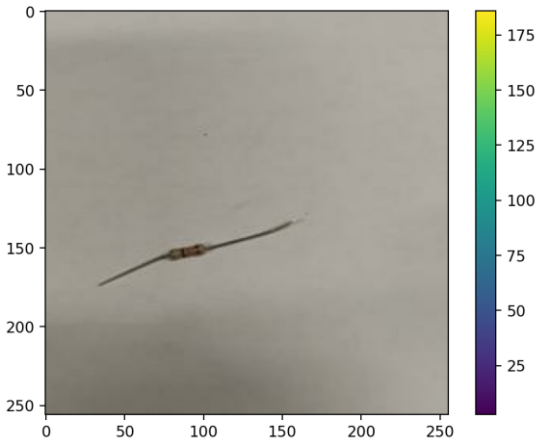


Figure 10. Initial pixel value of an image.

A common practice is to normalize the pixel values in the range of 0–1, which can be implemented by simply redefining the dataset to account for the division of 255, as such:

```
train_ds = train_ds / 255.0
```

Replotting the image confirms the change, as shown in Figure 11.

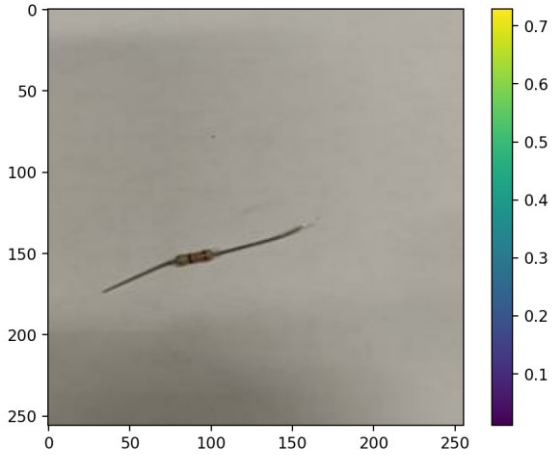


Figure 11. Normalized pixel value range of 0–1.

This implementation reduces the model's proneness to bias and facilitates convergence (Zhang, 2019). The division method may need adjustment depending on the dataset loading procedure. A straightforward approach is to implement the normalization function during model creation, as detailed in Chapter 3.2.3.

3.3.2 Data Augmentation

While the utilization of data augmentation in the training split is optional, it is recommended when the overall size of the dataset is limited. A dataset comprising 4,000 images can be considered small-scale where convolutional neural networks are concerned. After conducting multiple experiments and adhering to the study on image data augmentation by Shorten and Khoshgoftaar (2019), the following configuration demonstrated the highest level of optimization (Adapted from TensorFlow, 2024b):

```
data_augmentation = Sequential([
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2, fill_mode='nearest'),
    layers.RandomZoom(0.1),
    layers.RandomContrast(0.2),
    layers.RandomBrightness(0.2),
    layers.GaussianNoise(0.1),
])
```

This augmentation process creates additional temporary variants from the original 2,800 images in the training set. The exact number of generated images is challenging to evaluate due to the inherent randomness involved. However, a rough estimate suggests the total could range in the tens of thousands. Within this structure, images are randomly rotated by a maximum of 20%, zoomed inward randomly by a maximum of 10%, contrast and brightness values are adjusted randomly by a maximum of 20%, and finally, Gaussian noise is added with a strength of 10%. In Figure 12, each function is represented by a single-generated example, showcasing the outputs of the augmentation techniques.

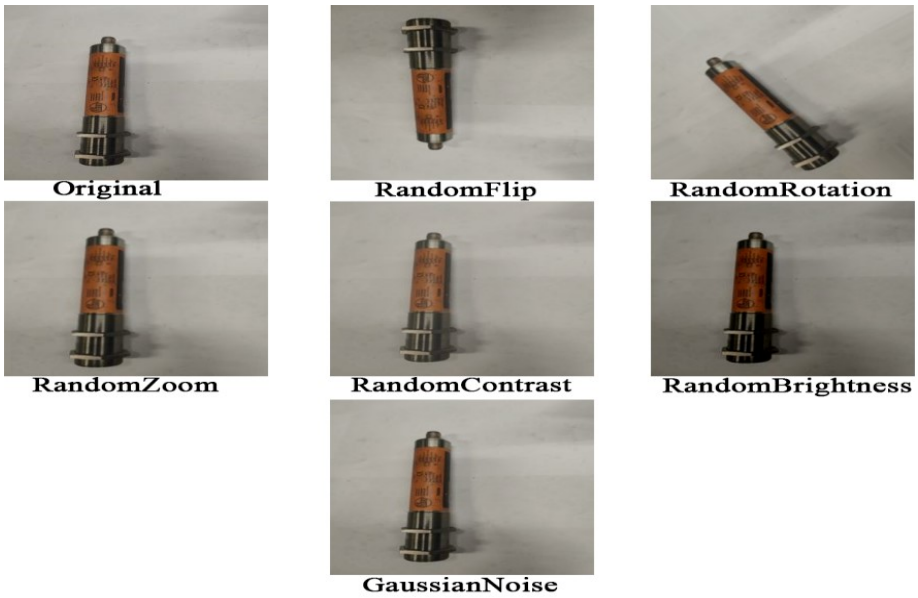


Figure 12. Examples of data augmented images.

As can be noted from the above examples, the generated images are not deviating drastically from the original, which is intended. Slight; yet realistic environmental alterations generally provide the model with richer data to interpret a class. However, exceptions are applicable. Depending on the verification task, a flipped image can have a detrimental effect should it alter the original intention, i.e., a number labeling of 6 vs. 9. To verify that the data-augmented images are correctly generated and realistic, the following script can be used to plot the images with the matplotlib dependency (Adapted from TensorFlow, 2024a):

```
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
plt.show()
```

In this example, the first nine data-augmented images are taken from the indexing. Thereafter, the enhanced data-augmented training dataset was concatenated with the model.

3.3.3 Optimization of Layers

After conducting numerous experiments with hyperparameter tuning, the subsequent model structure of Algorithm 1 was deemed as the most optimal.

```
model = Sequential([
    data_augmentation,
    layers.Rescaling(1./255),
    layers.Conv2D(8, 3, padding='same', activation='relu'),
    layers.Conv2D(8, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(128,     3,     padding='same',     activa
tion='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.1),
    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.05),
    layers.Dense(num_classes, name="outputs")
```

```
])
```

Algorithm 1. CNN model's optimized script (Adapted from TensorFlow, 2024a).

It can be inferred from the model's structure; that the previously defined data augmentation and pixel value normalization have been incorporated into the model. Utilizing twice the number of convolutional layers, each with 8 and 16 channels respectively, resulted in improved accuracy across both validation and test datasets. Whereas the rest of the convolutional layers extracted the larger details, differentiating classes from each other.

Experimentation was also conducted with higher initial channel sizes of 16 or 32; while they may capture slightly larger features, they proved suboptimal versus the initial channel size of 8, with increased computational time in addition. The results were mutual, with an additional larger channel size of 256 introduced after the convolutional layer with 128 channels. On occasion, further additions of convolutional layers may require a dense layer with an increase in neurons. An instance of compromise, as the current dense layer solely is responsible for 2,097,408 / 2,203,836 of total parameters.

As previously stated, the classes contain a mix of simple and complex features. If the initial channel size was altered, the model performed inadequately in distinguishing the simpler features. Whereas increasing the dense layer size or by introducing further convolutional layers, the model was prone to not learning adequately due to the increase in parameters. In addition, the computing times per epoch are significantly affected during the training phase.

The model includes two dropout layers: one applied to the final convolutional layer and another to the initial dense layer. Common practice is to apply a dropout layer for just the initial dense layer, as it attributes for most of the parameters and is hence equally prone to overfit. The values are minimal but proved to retain the consistency between the training and validation accuracies and losses. The activation function ReLU, serves to

introduce non-linearity into the network, thereby facilitating the model's ability to learn complex patterns (Adapted from TensorFlow, 2024d).

Since the final layer does not have a specifically defined activation function, the model defaults to using an equivalent linear function. The model's total parameters can be inspected with (TensorFlow, 2024a):

```
Model.summary()
```

Which prints the complete structure as follows in Algorithm 2.

```
Layer (type)                 Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 256, 256, 3)       0

 rescaling (Rescaling)       (None, 256, 256, 3)       0

 conv2d (Conv2D)             (None, 256, 256, 8)       224

 conv2d_1 (Conv2D)           (None, 256, 256, 8)       584

 max_pooling2d (MaxPooling2D  (None, 128, 128, 8)      0
 )

 conv2d_2 (Conv2D)           (None, 128, 128, 16)      1168

 conv2d_3 (Conv2D)           (None, 128, 128, 16)      2320

 max_pooling2d_1 (MaxPooling  (None, 64, 64, 16)       0
 2D)

 conv2d_4 (Conv2D)           (None, 64, 64, 32)        4640

 max_pooling2d_2 (MaxPooling  (None, 32, 32, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 32, 32, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 conv2d_6 (Conv2D)           (None, 16, 16, 128)       73856

 max_pooling2d_4 (MaxPooling  (None, 8, 8, 128)        0
 2D)
```

```
 dropout (Dropout)            (None, 8, 8, 128)         0

 flatten (Flatten)            (None, 8192)              0

 dense (Dense)               (None, 256)              2097408

 dropout_1 (Dropout)          (None, 256)               0

 outputs (Dense)              (None, 20)                5140
=================================================================
Total params: 2,203,836
Trainable params: 2,203,836
Non-trainable params: 0
```

Algorithm 2. CNN model's total parameters summary.

Considering the total number of parameters with regards to the total dataset size, the quantity of images is low. However, as the images are mainly devoid of background noise, the generalization capability of the model can be considered satisfactory.

3.3.4 Compiling the Model

It is recommended to incorporate early stopping and learning rate scheduling before training commences. These techniques serve as precautionary measures, intervening if the monitored metric fails to improve or decrease after a specified number of epochs. For this classification purpose, the early stopping's metric to monitor was defined as validation accuracy (Adapted from TensorFlow, 2024e):

```
early_stopping    =    EarlyStopping(monitor='val_accuracy',
mode='max',              patience=10,              verbose=1,
restore_best_weights=True)
```

In this example, the training was stopped if the validation accuracy does not improve for ten consecutive epochs, regardless of whether the training is set to commence for a longer period. The verbose value of one indicates that a message is displayed to notify of action taken during the concerned epoch. While 'restore_best_weights' is set to true,

the system retrieves the best performing weights from the corresponding epoch (TensorFlow, 2024e).

As for the learning rate scheduling, the metric to monitor was set as validation loss. Contrary to the validation accuracy, the validation loss is expected to diminish as the training progresses (Adapted from TensorFlow, 2024f):

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.4,
patience=5, min_lr=1e-6, verbose=1)
```

In the above example, the initial learning rate reduces by a factor of 0.4 after five consecutive epochs with no decrease in validation loss, resulting in a 60% reduction. The minimum learning rate lower boundary is set at 1e-6; however, typically, the system implements early stopping or training concludes before reaching this threshold. Experimentation with hyperparameter values is essential for achieving an optimal balance. This function proves particularly beneficial during later epochs, especially as the model achieves a validation accuracy exceeding 70%. At this stage, the current learning rate may be insufficient for the model to advance without risking overfitting.

As the training progresses, instances may arise where the model begins to overfit, characterized by a faster increase in accuracy for the training split compared to the validation split. Hence in Appendix 2, lines 123–139 a custom callback has been implemented to save only such weights where the threshold of a 5% difference between accuracies is not exceeded. Due to TensorFlow's built-in functionalities, compiling the model is straightforward, involving the selection of an optimizer, a loss function, and metrics (Adapted from TensorFlow, 2024a):

```
model.compile(optimizer=Adam(learning_rate=0.00095),
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_log
its=True),
metrics=['accuracy'])
```

Presently, TensorFlow lists a selection of thirteen optimizers, among which Adam stands out as a prominent choice due to its adaptive learning rate algorithm which combines the benefits of both momentum and RMSprop techniques (TensorFlow, 2024h). This results in an efficient convergence rate while being computationally efficient. The commonly used initial learning rate range is 0.001–0.01, depending on various factors such as the complexity of the model, the complexity of the classes, and the defined batch size.

The loss function relies on the activation function of the final dense layer, if defined. In this example, the 'SparseCategoricalCrossentropy' loss function expects labels with integer values; when 'from_logits=True' is set, the model outputs raw logits. Although no softmax activation function was explicitly defined, it is internally applied during training (TensorFlow, 2024i). This implies that the images retain their integer values, allowing for plotting with probabilities without the need to convert from confidence values. The metric is set to accuracy by default, with the purpose of calculating the frequency of correct predictions of corresponding labels as depicted in (5) (Alzubaidi et al., 2021, p. 60).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

Where:

TP: True Positive,

TN: True Negative,

FP: False Positive,

FN: False Negative.

3.3.5 Training and Evaluation of the Model

To initiate the training process, the setup expects both training and validation sets as a reference. As the epochs progress, historical logs of their performances are stored and displayed (Adapted from TensorFlow, 2024a):

```
history = model.fit(
     train_ds,
     validation_data=val_ds,
     epochs=epochs,
     callbacks=[early_stopping,                    reduce_lr,
best_weights_callback]
)
```

The definition of an epoch varies depending on the specific task at hand. For tasks involving distinct classes and a saturation of images, the model may efficiently learn within just ten epochs. Conversely, other tasks may require training to extend beyond a hundred epochs. The previously set callbacks must be referred to in the respective row for them to be accounted for during training. The training was initiated for 75 epochs, with the logs of interest included in Algorithm 3.

```
Epoch 34/75
88/88 [==============================] - ETA: 0s - loss:
0.7004 - accuracy: 0.7621
Epoch 34: ReduceLROnPlateau reducing learning rate to
0.00038000000640749934.
88/88 [==============================] - 40s 457ms/step -
loss: 0.7004 - accuracy: 0.7621 - val_loss: 0.9745 -
val_accuracy: 0.7113 - lr: 9.5000e-04
Epoch 63/75
88/88 [==============================] - ETA: 0s - loss:
0.3630 - accuracy: 0.8764Epoch 63: Saved best weights with
validation accuracy: 0.83%
88/88 [==============================] - 40s 458ms/step -
loss: 0.3630 - accuracy: 0.8764 - val_loss: 0.6070 -
val_accuracy: 0.8350 - lr: 3.8000e-04
Epoch 73/75
88/88 [==============================] - ETA: 0s - loss:
0.3008 - accuracy: 0.8975Restoring model weights from the end
of the best epoch: 63.
88/88 [==============================] - 40s 453ms/step -
loss: 0.3008 - accuracy: 0.8975 - val_loss: 0.5623 -
val_accuracy: 0.8325 - lr: 3.8000e-04
```

```
Epoch 73: early stopping
```

Algorithm 3. Training progress and optimization logs.

During epoch 34, the learning rate was adjusted to 3.8e-4, as the validation loss did not improve over five consecutive epochs. Despite nearing completion of the 75 epochs, early stopping was triggered as validation accuracy failed to improve for 10 consecutive epochs. Consequently, the best weights were retrieved from epoch 63. Given that this model utilizes a CPU, computational times were considerably longer compared to those achieved with a GPU, totaling 48 minutes for training. A validation accuracy of 84% can be deemed satisfactory, given the limited dataset size. However, if integrated into an inventory management system, additional advancements would be necessary to reduce the reliance on human intervention.

As the splits were arranged into training–validation–test sets, with 70–20–10 ratios respectively, the model's performance on the test set can be evaluated with (Adapted from TensorFlow, 2024a):

```
test_loss, test_accuracy = model.evaluate(test_ds)
13/13 [==============================] - 1s 79ms/step - loss:
0.4483 - accuracy: 0.8725
```

In this instance, the test set achieved an accuracy of 87.25% on unseen data. Training and validation losses during training can be plotted followingly with the matplotlib dependency (Adapted from TensorFlow, 2024a):

```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'],     label='Validation
Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Which generates the plot as follows in Figure 12:

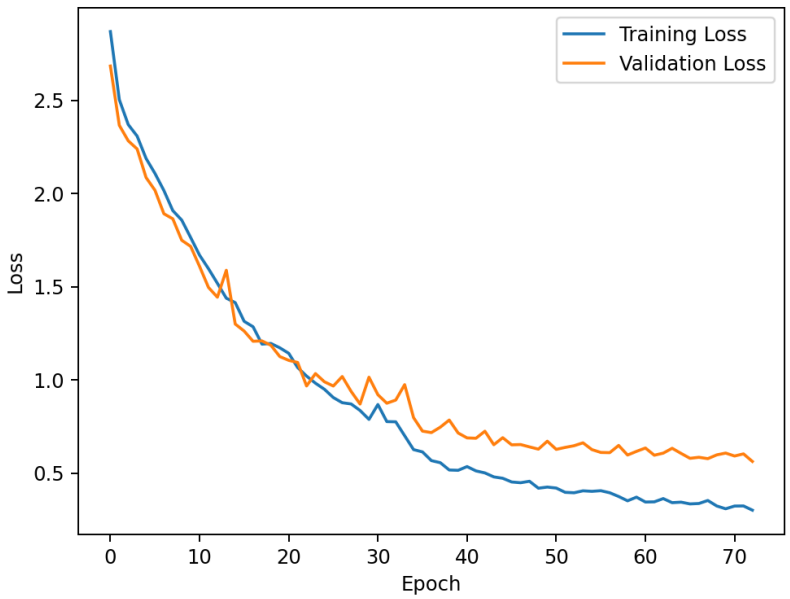


Figure 13. Training and validation losses for 75 epochs.

The losses exhibit similarity up to 20 epochs; however, for this instance, a more substantial adjustment in the learning rate should have been implemented, as no significant improvement can be perceived beyond the adjusted epoch of 34. In a similar manner, the training and validation accuracies are plotted with (Adapted from TensorFlow, 2024a):

```
plt.plot(history.history['accuracy'], label='Training Accu-
racy')
plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Which generates a plot as depicted in Figure 13:

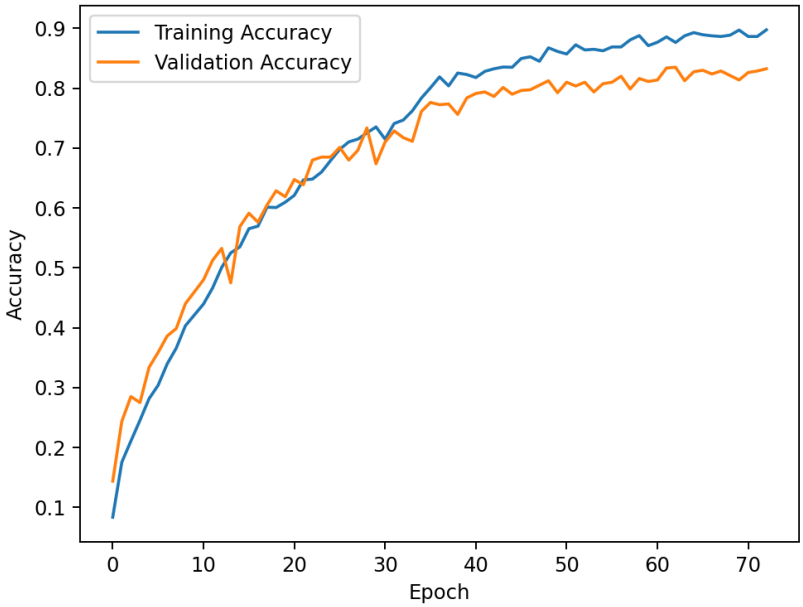


Figure 14. Training and validation accuracies for 75 epochs**.**

As inferred from the figure, the model exhibited a faster learning pace until the 30 epochs. Despite a subsequent adjustment during epoch 34, the model continued to learn, albeit at a slower pace, ultimately peaking at epoch 63, from which the best weights were restored. To compute the top-k accuracies, the labels are initially transformed into one-hot encoding, as the subsequent functions rely on this format. Since the integer value of the first class is 0, a one-hot-encoded array on the contrary contains a value of 1 at index 0, followed by zeros for the remaining classes out of the total number of classes as demonstrated in (6) (TensorFlow, 2024j):

$$[1.\ 0.\ 0.\ 0.\ 0.\ 0.\ 0.\ 0.\ 0.\ 0.\ 0.\ 0.\ 0.\ 0.\ 0.\ 0.\ 0.\ 0.\ 0.\ 0.\ 0.] \quad (6)$$

The process to generate the top-k accuracies is depicted in lines 165–202 in Appendix 2. The softmax function is initially applied to convert logits into probabilities, which are utilized during the generation of the confusion matrix. Then, the labels are converted to one-hot encoding. Finally, the one-hot encoded labels and the softmax probabilities are used to calculate the k-accuracy function, where k represents the desired number of top predictions.

For this instance, a top-3 accuracy of 96% and a top-5 accuracy of 99% were recorded. The inclusion of top-20 accuracy serves purely as a sanity check to ensure the correctness of the conversion process, yielding an intended accuracy of 100%. While the validation accuracy remains within the 60%–90% range, it is ideal to plot the top-5 predicted probabilities for images, as well as a confusion matrix of the entirety of classes to evaluate the feasibility of the model. Mislabeling of classes with similar traits is anticipated and can indicate that the model is learning effectively. The confusion matrix can be plotted as demonstrated in Appendix 2, lines 206–223.

The execution of the script generates a confusion matrix with true labels represented on the y-axis and predicted labels on the x-axis. Additionally, class-wise probabilities are displayed, as depicted in Figure 15.

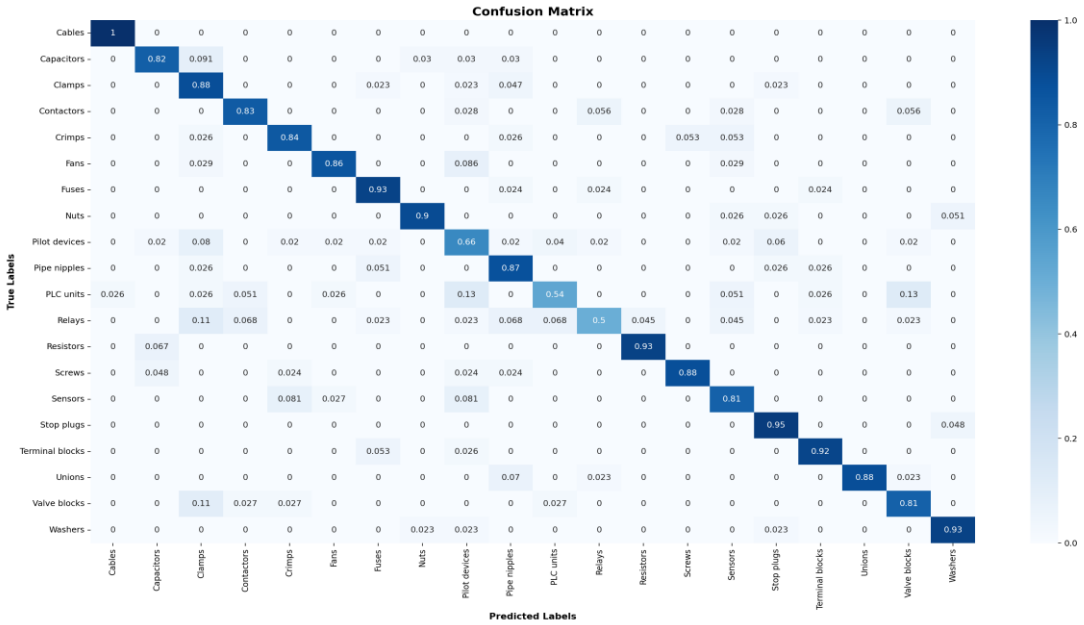


Figure 15. Confusion matrix with true vs. predicted labels**.**

To interpret the results, class ‘Cables’ was perfectly classified even though a risk is present with associating them with the ‘Sensors’ class, which also contains sensors with integrated cables. The model has learned to distinguish features between various cable

states, including being on a reel, wrapped, or tangled, rather than solely relying on the outer shape. In this instance, the three lowest-performing classes were 'Relays,' 'PLC units,' and 'Pilot devices,' which consistently recur even with minor hyperparameter tuning. Due to the similar shapes and numerous variants within classes, the model is prone to misclassifying between these categories. Increasing the number of images per class, as being conducted with the secondary model, could be beneficial.

To plot the top-5 predictions of an indexed image, the function must be initialized as described in Appendix 2, lines 227–270. The script initializes the top 5 predicted labels along with their corresponding class integers. The true label is colored blue, while incorrect predictions are colored red. In Appendix 2, lines 276–296, the plot of image 4 from the test set is generated.

As depicted in Figure 16, the model demonstrates satisfactory generalization, as evidenced by the initial prediction being correct with a probability of 97.1%.

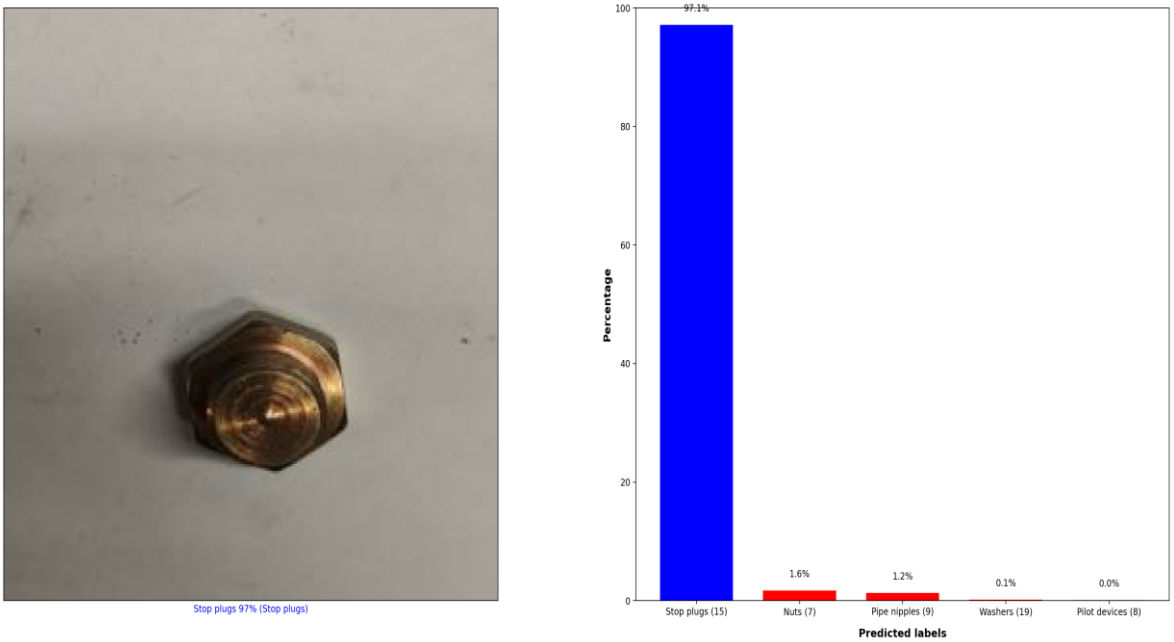


Figure 16. Top-5 probability predictions for an image from the 'Stop plugs' class.

The three incorrect probabilities suggest that while hexagonal edges are shared between classes, the model has learned to account for textural differences and other distinguishing features, such as the presence of a hole in the middle of a nut. On the contrary, class ‘PLC units’ is confused as a pilot device or contactor as exhibited in Figure 17.

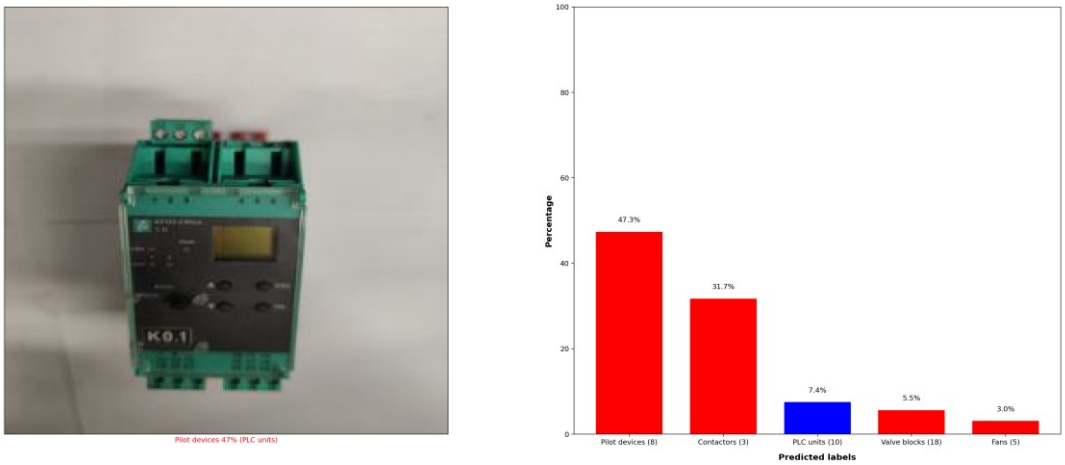


Figure 17. Incorrect predictions for an image from the ‘PLC units’ class.

The example depicted above could also pose difficulty for humans to classify accurately based solely on visual interpretation. In this phase, it is essential to assess the impact of applied data augmentations and variations in image angles on the model's learning process. The objective is to determine whether these adaptations positively or negatively influence the model's performance.

Experimentation is required in addition to batch size, convolutional and dense layers, regularization techniques, and scheduled learning rate adjustments. The next chapter covers the secondary model with an enhanced dataset, as the current indication is that the model’s learning is constrained by the limited dataset size.

3.4 Secondary Model

The second model was trained and validated with a dataset comprising 20,000 images, a substantial increase from the previous 4,000. Given the time-consuming nature of photographing images, the existing dataset was augmented through duplication, employing techniques such as horizontal and vertical flipping, as well as zooming inwards and outwards by 10% on the images. Therefore, data augmentation was omitted from the model, as it was already inherent within the dataset. This time, the splits were reorganized, omitting the test split. The training split encompassed 80% of the data, leaving the remaining 20% for validation. The script encompassing the secondary model is provided in Appendix 3.

The model's architecture has also been adjusted slightly. It now features single convolutional layers with 8 and 16 channels, followed by the addition of two higher-level convolutional layers at the end with 256 and 512 channels, respectively. A single dropout layer has been introduced for the initial dense layer which contains 256 neurons. Following this, a new dense layer with 128 neurons has been added, while the remaining hyperparameters remain unchanged. With these modifications, the training process was initiated, and the differences in convergence and computing time were significant, as indicated by the excerpt from the log:

```
Epoch 35: Saved best weights with validation accuracy: 0.96%
500/500 [==============================] - 5s 10ms/step - loss:
5.4348e-04 - accuracy: 0.9999 - val_loss: 0.2694 - val_accuracy:
0.9597 - lr: 6.0800e-05
```

Validation accuracy reached 91% by the 10th epoch; for this instance, it took just under 3 minutes to train the model with fivefold the dataset size as opposed to the CPU variant’s 48 minutes with lesser validation accuracy. The difference in learning rate is exhibited in Figure 18.

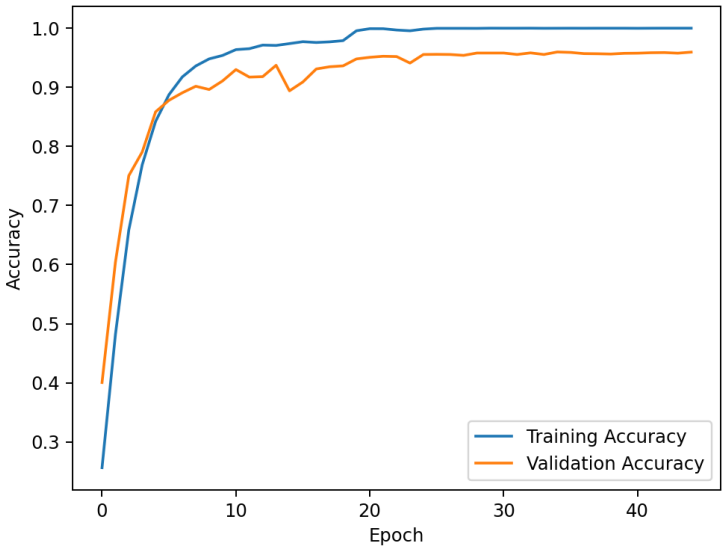


Figure 18. Training and validation accuracy on 20,000 images.

In general, the model is capable of achieving a validation accuracy between 91% and 94% with the initial learning rate, whereas higher accuracies necessitate an adjustment in the learning rate. For this instance, the model attained a top-3 validation accuracy of 99.5% and a top-5 validation accuracy of 99.8%. By replotting the confusion matrix, the results are as depicted in Figure 19.

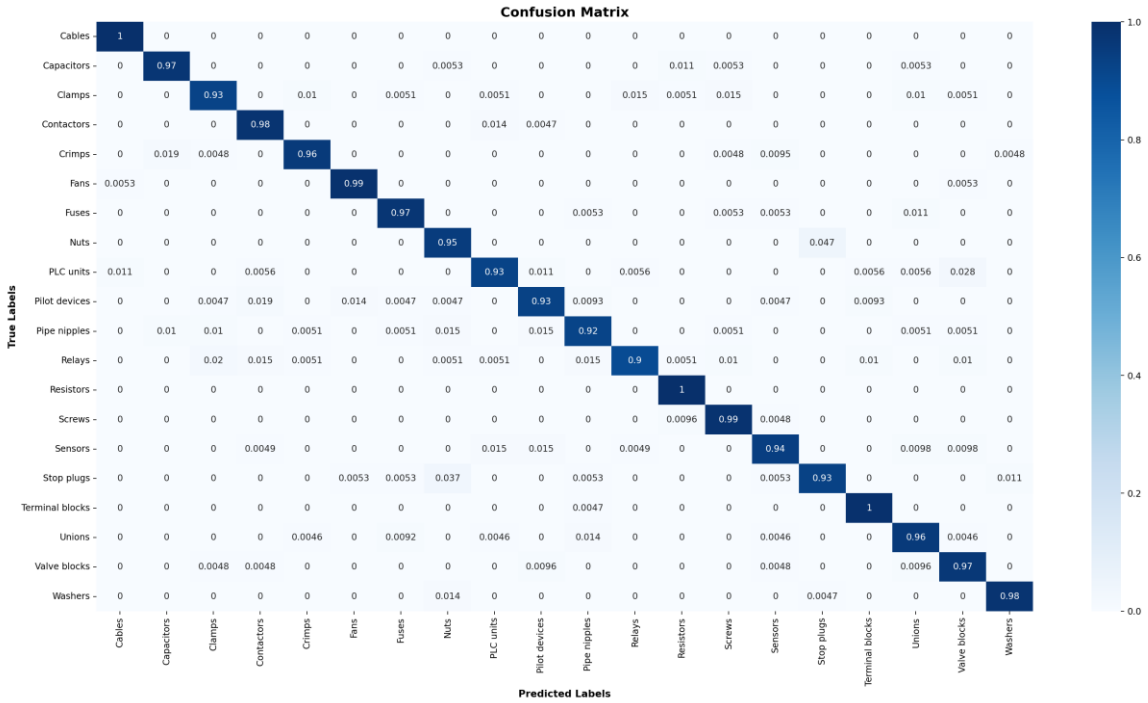


Figure 19. Confusion matrix of a model trained on 20,000 images.

All the class-wise comparisons have achieved an accuracy of 90% at the minimum. With 'Relays' class being an underperformer. This is attributed to the wide variety of features contained in the dataset. Consequently, these classes can inherently overlap with other classes due to the insufficient number of distinguishable features that can be extracted from certain images. In such scenarios, increasing the resolution size, for example, to 512x512, could potentially benefit the model. However, it is important to note that adjusting the model's structure and hyperparameters would likely be necessary and may require further tuning to achieve optimal performance.

Since the model attained satisfactory validation accuracy, the whole model was saved with the following script (Adapted from TensorFlow, 2024m):

```
model.save('model_name.keras')
```

With this example, the whole structure of the model is saved along with its best weights. By reloading the model, the former layer structure is present. The pre-existing model can

be utilized to fine-tune or retrain the final layers. While the initial layers retain their weights, the final layers are adjusted to accommodate the new class, considering that the final activation layer neurons match the number of classes. Depending on the similarity between the existing and new classes, it may be necessary to retrain the model from scratch. However, maintaining records of the hyperparameters and setups for the best model ensures that the time difference between the options is negligible. For instance, the time to caption 200 images of a single class typically ranges from an hour to two; extending this to 1000 images takes only a matter of minutes due to scripting. If the new class is distinct from the existing classes, hyperparameter adjustments may not be necessary.

The upcoming chapter involves loading the model and integrating its predictions with the inventory management system.

3.5 Integration to Inventory Management System

To load the model architecture and its best weights for making predictions, the following script was used (Adapted from TensorFlow, 2024m):

```
loaded_model = tf.keras.models.load_model('model_name.keras')
```

Since the model can provide either confidence or probability values for each class, utilizing confidence values is preferable for this purpose. This is because, with the current model, confidence values typically fall within the range of 97% to 100%. To generate the confidence values, along with the file names (reference code), class integers, and class names in an Excel file, the utilization of the Pandas library in conjunction with OpenPyXL is necessary. These libraries can process Excel formats by reading, editing, and generating Excel files. The script for replicating the testing on unseen data and generating the Excel file is provided in Appendix 4, with Table 3 exhibiting the formatting and results for the first seven predictions.

Table 3. Confidence percentages of a test set.

File Name	CI	Class	Confidence (%)
Cables 1 (Product 1)	0	Cables	100
Cables 2 (Product 2)	0	Cables	99,999
Cables 3 (Product 3)	0	Cables	100
Cables 4 (Product 4)	0	Cables	100
Cables 5 (Product 5)	0	Cables	99.999
Capacitors 1 (Product 6)	1	Capacitors	99.995
Capacitors 2 (Product 7)	1	Capacitors	100

These specific images have been labeled with a class in their file names, which facilitates the verification process of the test accuracy. Out of 113 test samples, the model predicted 106 correctly, which equals an accuracy of 93.8%. Each class had a minimum of 5 samples. Despite encountering variations with no prior feature extraction, such as a sealing washer with an integrated sealant within the inner circle, the model demonstrated robustness. In the future, exposure to such variants in datasets could lead to more efficient interpretation by the model.

Establishing a lower threshold for the confidence percentage is subjective and varies based on user preferences. However, for accurate integration of data into the database, it is advisable to implement conditions where the reference codes from an Excel file are cross-referenced against the database. Subsequently, if the database contains the relevant product, the confidence value is then assessed. For instance, only class integers corresponding to items with confidence percentages equal to or higher than 95% would be integrated into their respective entries. The complete workflow process between the dependencies of the three actors is depicted in Appendix 5, with the rectangular boxes denoting the domains. Each actor's domain is highlighted in a distinct color to visually differentiate them.

It is important to acknowledge the potential for false positives, as the model may exhibit high confidence in certain predictions despite misclassifying the item. While the correction of incorrect labels can be effortlessly accomplished during the annual inventory revision using a drop-down table, as opposed to manually defining each item individually.

When the model has achieved a high level of validation or test accuracy, it is ideal to assess its class activation mapping. This can be accomplished using the Grad-CAM dependency, which highlights sections of the image with a heatmap where the model places the most emphasis when making predictions for a particular class; blue indicates minimal emphasis, while red represents maximal emphasis. With the implementation outlined in Appendix 6. In Figure 20, class activation mapping is demonstrated for an image of a sensor, where the model predicts for both the 'Sensors' and 'Cables' classes.

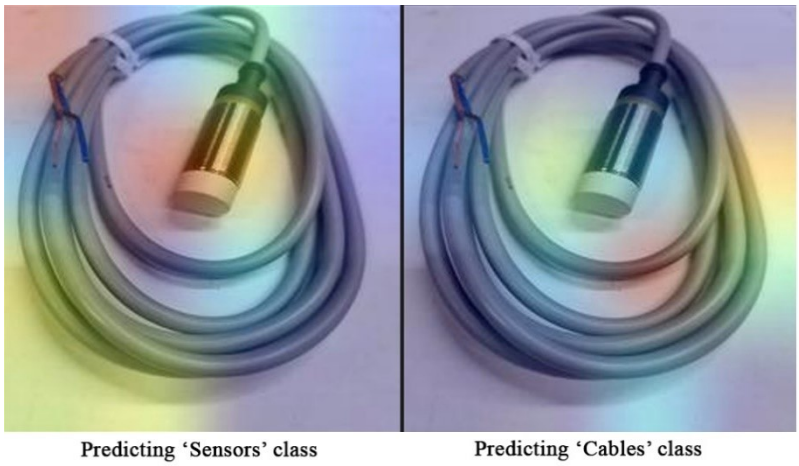


Figure 20. Class activation maps for 'Sensors' and 'Cables' classes during predictions.

From the above example, it can be inferred that when an image of an actual sensor is being predicted, the emphasis lies on the sensor region and end connection. On the contrary, when evaluating whether this sensor belongs to a 'Cables' class, inference is drawn from features typically associated with cables, such as diameter, texture, and even the presence of a gap in the middle if wrapped.

Gradients from the last convolutional layer are typically used for this purpose, as during the initial layers, the model primarily learns abstract features that may not be interpretable by humans. As the depth of convolutional layers increases, the model becomes more susceptible to vanishing gradients. Consequently, the tool inherently serves a dual purpose, as it can also be utilized for the removal of excessive convolutional layers.

4 Evaluation of Inventory

This chapter aims to conduct ABC-XYZ analysis on sales data from the past three years and compare it with the currently stocked items to understand the status of the inventory use rate. Concluding with stocking recommendations.

4.1 Implementing ABC Analysis on Stocked Items

While ABC-XYZ analysis is typically applied to sales data, categorizing the value of stocked items into groups is advantageous. This approach considers limited sales activity and prioritizes products deserving greater attention. After merging the values of duplicate items and removing outliers, the current inventory consists of 3,848 unique items.

To prepare the data for ABC categorization, the items are organized based on their cost price contribution, starting from the largest to the smallest. Additionally, the cumulative cost price is considered. These figures are then compared to the total value of the inventory to determine the individual and cumulative percentage contribution of the value, with Table 4 exhibiting the layout of the first three items.

Table 4. Implementation of ABC categorization on stocked items**.**

Reference Code	Product Description	Contribution %	Cumulative %	ABC Category
1	Product 1	2.76	2.76	A
2	Product 2	1.93	4.69	A
3	Product 3	1.63	6.32	A

In this example, items that contributed 80% of the total value of the inventory were categorized as A, followed by items contributing the subsequent 15% categorized as B, and the remaining 5% categorized as C. Cell coloring is implemented using conditional formatting, which correlates inventory data with sales data spanning the past eight years.

Items that have been sold at least once are highlighted with a green cell color, while items without any sales data are marked with colors such as red, orange, or yellow, corresponding to their ABC group. With the categories assigned, the allocation of the items is depicted in Table 5.

Table 5. Distribution of items belonging to ABC groups.

Total Inventory Items	3842
Belonging to Group A	555 (14.4%)
Belonging to Group B	892 (23.2%)
Belonging to Group C	2401 (62.4%)

While the distributions assist in evaluation, they become more beneficial once coupled with the conditional formatting of sales data. After the extraction of such items which are represented in both stocked items and sales data, the conditional formatting was prepared, with Table 6 revealing that only a marginal quantity of products have recorded sales.

Table 6. Sales distribution based on ABC groups.

Total Unique Items Sold	357
From Group A	107
From Group B	82
From Group C	168

Special attention should be directed towards items categorized as group A, particularly those without any sales data at present. Since this subset of 555 items represents 80% of the total inventory value. The subsequent chapter focuses on conducting ABC-XYZ analysis on the limited sales data.

4.2 Implementing ABC-XYZ Analysis Based on Sales Data

As with the stocked inventory data, the sales data must first be sanitized to remove any records that do not represent products or contribute to the sales price. While the principle of conducting ABC grouping remains consistent with the previous chapter, XYZ grouping is dependent on sales dates. Therefore, a specific number of rows must be allocated based on the chosen timeline, which, in this case, spans three years.

Since spare parts inventory typically moves slowly, the review interval is interpreted in months. During the 36-month review interval, sales records were found for 242 unique items. Whereas only 24 items exhibited multiple sales instances, attributed to the enterprise's main revenue generation from make-to-order and engineer-to-order projects.

After setting the relevant products and time interval slots, the data is arranged based on sales value, starting from the largest to the smallest. Thereafter, the same practice was followed with the contribution and cumulative percentages compared against the total sales value for the given review interval. An example of how to conduct the XYZ analysis on a single item is depicted in Table 7.

Table 7. Implementation of ABC-XYZ analysis on sales data.

Months	01/2021	<x>	12/2023	Contribution %	Cumulative %	ABC-XYZ	Standard deviation	Coefficient variation
Product 1	1	0	1	14.11	14.11	AZ	0.23	412 %

On the above table, '<x>' denotes the rest of the months between the first and last review interval. Whereas population standard deviation is calculated as depicted in (7) (Adapted from Stojanović & Regodić, 2017, p .36).

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i-\mu)^2}{N}} \tag{7}$$

Where:

- *N* denotes the total number of data points.

- x_i denotes each individual data point.
- μ is the mean (average) of the data points.

And the coefficient of variation (CV) is derived from the population standard deviation as follows: (8) (Adapted from Everitt & Skrondal, 2010, p. 89).

$$CV = \frac{\sigma}{\mu} \tag{8}$$

In determining XYZ rankings using the CV, universally accepted boundaries do not exist. Instead, classification adhered to the thresholds proposed by Stojanović and Regodić (2017, p. 36) for integrating ABC-XYZ into inventory management processes. Accordingly, items with a CV of 10% or under was classified as X, those with a CV between 10% and 25% as Y, and those with a CV exceeding 25% as Z. Out of all the evaluated items, the lowest coefficient of variation was noted at 262%, while the highest was observed at 592%. To interpret these findings, it is notable that most items were sold only once. Meanwhile, items with multiple sales records often still experienced extended periods of zero demand, suggesting intermittent demand patterns. Of the 242 sold items, 21 were categorized as AZ, 49 as BZ, and 172 as CZ. The limited number of sales instances over a 36-month period provides insufficient data points to establish reliable trends or patterns necessary for forecasting.

4.3 Inventory Stocking Considerations

The analysis of inventory and sales data reveals that most sales are attributed to one-time purchases, resulting in minimal accumulation of inventory. However, out of the 3,848 stocked items, only 357 have recorded sales, leaving 91% of the inventory without sales data. Table 8 is introduced as an evaluation tool for managing items based on their ABC classification and sales history (Adapted from Stoll et al., 2015).

Table 8. Inventory stocking evaluation by ABC class and sales history.

Description	ABC Class	Sales History	Recommendations
High value	A	Yes	Regular monitoring Strategic stocking
High value	A	No	Regular monitoring Re-evaluate demand
Moderate value	B	Yes	Moderate monitoring Flexible stocking
Moderate value	B	No	Consider bundling or promotions
Low value	C	Yes	Minimal monitoring Just-in-time stocking
Low value	C	No	Consider discontinuation

A Items (80% of total inventory cost):

- Regular monitoring: Monitor closely, despite sales history.
- Re-evaluate demand: Assess reasons for lack of sales and adjust stocking levels accordingly.
- Strategic stocking: Balance stock levels based on proven demand while minimizing excess.

B Items (15% of total inventory cost):

- Moderate monitoring.
- Flexible stocking: Adjust levels based on demand patterns, starting conservatively.
- Consider bundling or promotions: Combine with popular items or arrange promotions to boost sales.

C Items (Remaining 5% of total inventory cost):

- Minimal monitoring: Track changes in demand patterns.
- Just-in-time stocking: Maintain minimal levels and replenish as needed.
- Consider discontinuation: Assess cost-effectiveness for items with no sales history.

While C items with no prior sales may not contribute significantly to the total inventory value, their sheer quantity of 2,233 items constrains inventory space. It is advisable to begin minimizing their presence. Similarly, A items with no prior sales may face technological obsolescence, prompting the need to evaluate their compatibility for future projects. The final chapter briefly summarizes these findings together.

5 Managerial Implications

The CNN model streamlines workflow processes, enhancing resource productivity, especially as the number of identifiable products and product categories grows. Moreover, technological adaptation and skilled personnel foster innovative applications, such as precise demand forecasting in periods of zero demand, provided the enterprise maintains bureaucratic flexibility.

Conversely, the ABC-XYZ classification method simplifies decision-making by categorizing products based on importance and redundancy. This classification is particularly valuable for spare parts, where numerous items are non-moving. Consequently, making informed decisions regarding inventory reduction and maintenance becomes more manageable. By leveraging both practices, companies can develop a comprehensive understanding of product groups' overall status. This facilitates the evaluation of non-conforming products and guides corrective actions, optimizing inventory through reduced excess inventory, holding costs, and space limitations.

By adopting advanced technologies such as the CNN model and implementing efficient classification methods such as ABC-XYZ, companies not only streamline their operations and reduce costs in the short term but also position themselves for sustained success and competitive advantage in the long term.

6 Conclusions and Implications

In conclusion, the study has demonstrated the effectiveness of CNN and conventional grouping methods in automating the classification process and enhancing decision-making in warehouse operations. The secondary model achieved a validation accuracy of 96%, along with top-3 and top-5 validation accuracies of 99.5% and 99.8%, respectively. Although these results are promising, it is important not to overlook the potential issue of underspecification, as discussed by Alzubaidi et al. (2021, p. 50). To ensure the robustness and generalization capability of the model, it underwent evaluation against unseen data in an unsupervised manner. Out of 113 test samples, with each class represented by a minimum sample size of five, the model correctly labeled 106 instances, resulting in a 93.8% accuracy rate.

Class activation mapping was employed to analyse the model's preferences when predicting images. The model emphasized certain sections akin to a human's approach in determining a product's group. However, the current architecture faces challenges in predicting instances of classes with predominantly plain features, such as rectangular shapes and evenly metallic textures. As a result, it relied solely on integrated parts that belonged to another class. This highlights the presence of a mixture of products with varying levels of complexity in the classes. Correctly labeling products with complex features assumes expertise in mechanical and electrical attributes, while also acknowledging the potential for human error.

Given the satisfactory error rate of the model, it can be effectively employed to automate the product grouping process. Any occasional misclassifications can be identified and addressed during the annual inventory review. The accuracy of the model can be improved by increasing the size of the dataset, as CNNs rely on extensive data for effective convolution. Larger datasets provide more opportunities. for the model to learn and extract relevant features, thereby enhancing its performance. For the company in question, the experimentation phase incurred no financial risk due to the absence of accumulated costs. However, to assess the return on investment, potential expenses for computer and

camera setup, as well as the total cost of external consultation or internal expertise for database compilation and model training, need to be considered. These costs are then divided by the total savings that would have otherwise been realized through manual labeling by an employee. If utilizing Google Colab with Jupyter Notebook and existing cell phone cameras, the equation simplifies to the time required for compilation and model training cost versus the cost savings achieved for given duration.

As observed by Roger (2004), resource constraints, particularly in terms of human capital skills and assets, pose significant barriers to technology adoption. These patterns were noted to be present, where the time-consuming nature of initial planning and testing, often without guarantee of success, can deter SMEs during risk assessment. However, given the promising results of the model and the knowledge gained, the enterprise is now keen on further adapting CNN for manufacturing defect detection. Enterprises considering the adoption of CNN should initially explore available datasets, particularly those suitable for transfer learning. Many datasets are globally crowdsourced, encouraging collaboration among enterprises in similar domains. For instance, enterprises with common items that don't involve proprietary information, could collaborate in compiling a comprehensive spare part dataset.

While conventional grouping methods provide a static snapshot of current inventory status, they enable the enterprise to take corrective actions on redundant and expensive products with no prior sales. These grouping methods will be integrated into the system to dynamically present only relevant product data and exclude data entries that do not contribute to the total inventory value. Ideally, forecasting based on sales data would have been preferable. However, it became evident that the majority of products were purchased only once, and the most 'active' products still experienced zero periods of demand for 86% of the total three-year duration span.

6.1 Future Research and Study Limitations

For future research, further automation capabilities are to be evaluated in which reference codes would be taken directly from purchase orders and arranged into a data frame in a chronological order thereafter to be matched after the stationary camera has captioned such products that contain a unique identifier not found in the database currently. In this approach, the prediction process could be automated by scanning for new entries within the image database folder and cross-referencing them with those already processed. As a potential avenue for future research, experimenting with non-linear mixing of images for the most complex products, which exhibit high potential for confusion among other classes, could be undertaken, as depicted in Figure 21.

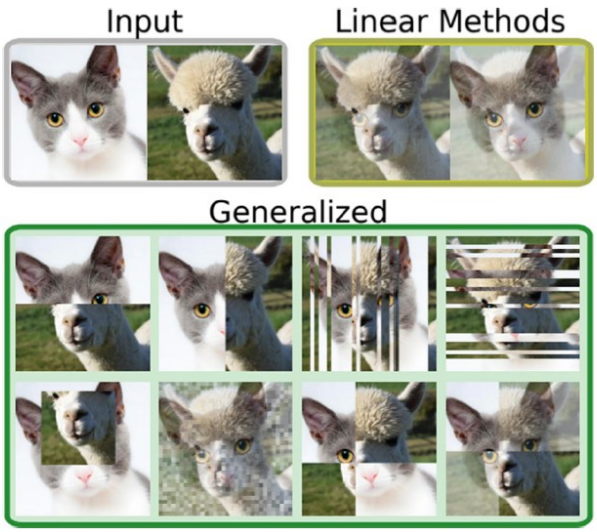


Figure 21. Non-linear mixing of images (Summers & Dinneen, 2019).

While counterintuitive for humans, Summers and Dinneen (2019) demonstrated in their study a reduction in error rates through this practice. If expanding the number of classes results in decreased prediction accuracy to sub-optimal levels, the utilization of skip connections, initially introduced by He et al. (2016) warrants consideration. These connections enhance gradient flow and facilitate the integration of simple and complex features, potentially improving the network's learning and generalization capabilities. Moreover, expanding the dimensionality of the ABC-XYZ analysis to include VED analysis and

assessing criticality through the analytic hierarchy process is a consideration for the future. However, this step should be taken after the inventory has been streamlined to include only relevant products.

References

Albayrak Ünal, Ö., Erkayman, B., & Usanmaz, B. (2023). Applications of Artificial Intelligence in Inventory Management: A Systematic Review of the Literature. *Archives of Computational Methods in Engineering*, *30*. https://doi.org/10.1007/s11831-022-09879-5

Altay Guvenir, H., & Erel, E. (1998). Multicriteria inventory classification using a genetic algorithm. *European Journal of Operational Research*, *105*(1), 29–37. https://doi.org/10.1016/S0377-2217(97)00039-8

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, *8*(1), 53. https://doi.org/10.1186/s40537-021-00444-8

Bacchetti, A., & Saccani, N. (2012). Spare parts classification and demand forecasting for stock control: Investigating the gap between research and practice. *Omega*, *40*(6), 722–737. https://doi.org/10.1016/j.omega.2011.06.008

Classification: True vs. False and Positive vs. Negative | Machine Learning. (n.d.). Google for Developers. Retrieved March 25, 2024, from https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative

Davis-Sramek, B., & Fugate, B. S. (2007). STATE OF LOGISTICS: A VISIONARY PERSPECTIVE. *Journal of Business Logistics*, *28*(2), 1–34. https://doi.org/10.1002/j.2158-1592.2007.tb00056.x

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (n.d.). *ImageNet: A Large-Scale Hierarchical Image Database*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* (arXiv:2010.11929). arXiv. http://arxiv.org/abs/2010.11929

Everitt, B. S., & Skrondal, A. (2010). *The Cambridge Dictionary of Statistics*.

Goode, S., & Stevens, K. (2000). An analysis of the business characteristics of adopters and non-adopters of World Wide Web technology. *Information Technology and Management*, *1*, 129–154. https://doi.org/10.1023/A:1019112722593

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. https://doi.org/10.1109/CVPR.2016.90

ImageNet. (2021). *ImageNet*. https://www.image-net.org/

Kandel, I., & Castelli, M. (2020). The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express*, *6*(4), 312–315. https://doi.org/10.1016/j.icte.2020.04.010

Keras. (2021). *Grad-CAM class activation visualization*. https://keras.io/examples/vision/grad_cam/

Kopka, A., & Fornahl, D. (2024). Artificial intelligence and firm growth—Catch-up processes of SMEs through integrating AI into their knowledge bases. *Small Business Economics*, *62*(1), 63–85. https://doi.org/10.1007/s11187-023-00754-6

Masters, D., & Luschi, C. (2018). *Revisiting Small Batch Training for Deep Neural Networks* (arXiv:1804.07612). arXiv. http://arxiv.org/abs/1804.07612

Obschonka, M., & Audretsch, D. (2020). Artificial Intelligence and Big Data in Entrepreneurship: A New Era Has Begun. *Small Business Economics*, *55*. https://doi.org/10.1007/s11187-019-00202-4

Python. (n.d.). *os.path—Common pathname manipulations*. Python Documentation. Retrieved March 30, 2024, from https://docs.python.org/3/library/os.path.html

Quayle, M. (2003). A study of supply chain management practice in UK industrial SMEs. *Supply Chain Management: An International Journal*, *8*(1), 79–86. https://doi.org/10.1108/13598540310463387

Ripley, B. D. (2005). *Pattern recognition and neural networks* (8. print). Cambridge Univ. Press.

Rogers, M. (2004). Networks, Firm Size and Innovation. *Small Business Economics*, *22*(2), 141–153.

Saha, S. (2018, December 15). *A Guide to Convolutional Neural Networks—The ELI5 way | Saturn Cloud Blog*. https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/

Sahoo, S. (2022, September 26). *Deciding optimal filter size for CNNs*. Medium. https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363

Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, *6*(1), 60. https://doi.org/10.1186/s40537-019-0197-0

Singh, D., & Verma, A. (2018). Inventory Management in Supply Chain. *Materials Today: Proceedings*, *5*(2), 3867–3872. https://doi.org/10.1016/j.matpr.2017.11.641

Stack Overflow. (2019a, May 16). *How to rename all the images in a directory with name coming from an excel cell values?* [Forum post]. Stack Overflow. https://stackoverflow.com/q/56167372

Stack Overflow. (2019b, September 23). *Saving prediction results to CSV*. Stack Overflow. https://stackoverflow.com/a/58064634

Stojanović, M., & Regodić, D. (2017). The Significance of the Integrated Multicriteria ABC-XYZ Method for the Inventory Management Process. *Acta Polytechnica Hungarica*, *14*(5), 29–48. https://doi.org/10.12700/APH.14.5.2017.5.3

Stoll, J., Kopf, R., Schneider, J., & Lanza, G. (2015). Criticality analysis of spare parts management: A multi-criteria classification regarding a cross-plant central warehouse strategy. *Production Engineering*, *9*(2), 225–235. https://doi.org/10.1007/s11740-015-0602-2

Summers, C., & Dinneen, M. J. (2019). Improved Mixed-Example Data Augmentation. *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 1262–1270. https://doi.org/10.1109/WACV.2019.00139

Syntetos, A. A., Keyes, M., & Babai, M. Z. (2009). Demand categorisation in a European spare parts logistics network. *International Journal of Operations & Production Management*, *29*(3), 292–316. https://doi.org/10.1108/01443570910939005

TensorFlow. (2024b). *Data augmentation | TensorFlow Core*. TensorFlow. https://www.tensorflow.org/tutorials/images/data_augmentation

TensorFlow. (2024a). *Image classification | TensorFlow Core*. https://www.tensorflow.org/tutorials/images/classification

TensorFlow. (2024h). *Module: Tf.keras.optimizers | TensorFlow v2.16.1*. TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

TensorFlow. (2024m). *Save and load models | TensorFlow Core*. TensorFlow. https://www.tensorflow.org/tutorials/keras/save_and_load

TensorFlow. (2024d). *Tf.keras.activations.relu | TensorFlow v2.16.1*. https://www.tensorflow.org/api_docs/python/tf/keras/activations/relu

TensorFlow. (2024e). *Tf.keras.callbacks.EarlyStopping | TensorFlow v2.16.1*. TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping

TensorFlow. (2024f). *Tf.keras.callbacks.ReduceLROnPlateau | TensorFlow v2.16.1*. TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ReduceLROnPlateau

TensorFlow. (2024i). *Tf.keras.losses.SparseCategoricalCrossentropy | TensorFlow v2.16.1*. TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy

TensorFlow. (2024j). *Tf.one_hot | TensorFlow v2.16.1*. TensorFlow. https://www.tensorflow.org/api_docs/python/tf/one_hot

Thambawita, V., Strümke, I., Hicks, S. A., Halvorsen, P., Parasa, S., & Riegler, M. A. (2021). Impact of Image Resolution on Deep Learning Performance in Endoscopy Image Classification: An Experimental Study Using a Large Dataset of Endoscopic Images. *Diagnostics*, *11*(12), 2183. https://doi.org/10.3390/diagnostics11122183

Vaaland, T. I., & Heide, M. (2007). Can the SME survive the supply chain challenges? *Supply Chain Management: An International Journal*, *12*(1), 20–31. https://doi.org/10.1108/13598540710724374

Wen, J., Thibeau-Sutre, E., Diaz-Melo, M., Samper-González, J., Routier, A., Bottani, S., Dormont, D., Durrleman, S., Burgos, N., & Colliot, O. (2020). Convolutional neural networks for classification of Alzheimer's disease: Overview and reproducible evaluation. *Medical Image Analysis*, *63*, 101694. https://doi.org/10.1016/j.media.2020.101694

Whitin, T. M. (1952). Inventory Control in Theory and Practice. *The Quarterly Journal of Economics*, *66*(4), 502. https://doi.org/10.2307/1882101

Williams, B. D., & Tokar, T. (2008). A review of inventory management research in major logistics journals: Themes and future directions. *The International Journal of Logistics Management*, *19*(2), 212–232. https://doi.org/10.1108/09574090810895960

Wu, S., Wang, G., Tang, P., Chen, F., & Shi, L. (2019). *Convolution with even-sized kernels and symmetric padding*.

Yang, S., & Du, Z. (2004). Criticality evaluation for spare parts initial provisioning. *Annual Symposium Reliability and Maintainability, 2004 - RAMS*, 507–513. https://doi.org/10.1109/RAMS.2004.1285498

Zhang, X. (2019, May 6). *ML Studio (classic): Normalize Data - Azure*. https://learn.microsoft.com/en-us/previous-versions/azure/machine-learning/studio-module-reference/normalize-data

Appendices

Appendix 1. Script to Automate Image Renaming with Reference Code (Adapted from Stack Overflow, 2019a)

```
import os
import glob
import pandas as pd
import shutil

def sanitize_filename(filename):
    # Convert to string if the input is not already a string
& replace special characters with a blank
    if not isinstance(filename, str):
        filename = str(filename)
    return filename.replace('/', '').replace('"', '')

def rename_files_in_subfolders(main_folder, excel_file):
    # Get the list of subfolders
    subfolders = [f.path for f in os.scandir(main_folder) if
f.is_dir()]
    # Read the Excel file
    df = pd.read_excel(excel_file, sheet_name=None)
    # Iterate through each sub-folder
    for subfolder in subfolders:
        print(f"Processing subfolder: {subfolder}")
        # Get the corresponding sheet name (which should
match the sub-folder name)
        sheet_name = os.path.basename(subfolder)
        # Check if the sheet exists in the Excel file
        if sheet_name not in df:
            print(f"Sheet '{sheet_name}' not found in the
Excel file. Skipping.")
            continue
        # Get the reference codes from the Excel sheet
        reference_codes = df[sheet_name]['Reference
code'].tolist()
        # Get the list of JPEG files in the sub-folder
        jpeg_files = sorted(glob.glob(os.path.join(subfolder,
'*.jpg')))
        # Verify whether the number of images matches the
number of rows in the Excel sheet
        if len(jpeg_files) != len(reference_codes):
            print(f"Number of images does not match the
number of rows in '{sheet_name}'. Skipping.")
            continue
        # Create a dictionary to map filenames to reference
codes
        file_ref_mapping = {}
        for i, ref_code in enumerate(reference_codes):
```

```
            sanitized_ref_code = sanitize_filename(ref_code)
            file_ref_mapping[jpeg_files[i]]                =
sanitized_ref_code
        # Sort the files based on reference codes
        sorted_files  =  sorted(jpeg_files,  key=lambda  x:
file_ref_mapping[x])
        # Rename the JPEG files
        renamed_files = set()  # To keep track of renamed
files
        for i, old_path in enumerate(sorted_files):
            ref_code = file_ref_mapping[old_path]
            old_name = os.path.basename(old_path)
            new_name = f"{ref_code}.jpg"
            # Assign (x) to a duplicate file
            if new_name in renamed_files:
                # If the file name has already been renamed,
add a suffix
                count = 2
                while   f"{ref_code}   ({count}).jpg"   in
renamed_files:
                    count += 1
                new_name = f"{ref_code} ({count}).jpg"
            # Construct the new path
            new_path = os.path.join(subfolder, new_name)
            # Rename the file
            shutil.move(old_path, new_path)
            print(f"Renamed '{old_name}' to '{new_name}'")
            # Add the new name to the set of renamed files
            renamed_files.add(new_name)

if __name__ == "__main__":
    main_folder = r'Path:\to\dataset\folder'
    excel_file   =   r'Path:\to\Excel\file\Reference   code
list.xlsx'
    rename_files_in_subfolders(main_folder, excel_file)
```

Appendix 2. Script for Training and Evaluating CPU Variant Model (Adapted from TensorFlow 2024a–l)

```
# Import dependencies
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import pandas as pd
import seaborn as sns

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import regularizers
from sklearn.metrics import confusion_matrix


# Define parameters for loader
batch_size = 32
img_height = 256
img_width = 256
data_dir = r'Path:\to\dataset\folder'
seed = 123 # Seed for shuffling and splitting

# Get list of all image paths
image_paths = []
class_names = []
for root, dirs, files in os.walk(data_dir):
    for subdir in dirs:
        class_names.append(subdir)  # Capture class names
from subdirectories
        subdir_path = os.path.join(root, subdir)
        for file in os.listdir(subdir_path):
            if file.endswith(".jpg") or
file.endswith(".png"):

image_paths.append(os.path.join(subdir_path, file))

# Sort the image paths to ensure reproducibility
image_paths.sort()

# Shuffle the list of image paths with seed
np.random.seed(seed)
np.random.shuffle(image_paths)

# Calculate the split indices
```

```
  45 total_images = len(image_paths)
  46 train_split = int(0.7 * total_images)
  47 val_split = int(0.2 * total_images)
  48
  49 # Split the image paths
  50 train_paths = image_paths[:train_split]
  51 val_paths = image_paths[train_split:train_split+val_split]
  52 test_paths = image_paths[train_split+val_split:]
  53
  54 # Create TensorFlow datasets from the paths
  55 def create_dataset(paths, labels):
  56     return tf.data.Dataset.from_tensor_slices((paths,
labels)).map(
  57         lambda x, y: (tf.io.read_file(x), y)
  58     ).map(
  59         lambda x, y: (tf.image.decode_jpeg(x, channels=3),
y),
  60         num_parallel_calls=tf.data.experimental.AUTOTUNE
  61     ).map(
  62         lambda x, y: (tf.image.resize(x, (img_height,
img_width)), y),
  63         num_parallel_calls=tf.data.experimental.AUTOTUNE

64     ).batch(batch_size).prefetch(tf.data.experimental.AUTOTUN
E)
  65
  66 # Create labels for each image
  67 class_to_label = {class_name: i for i, class_name in
enumerate(class_names)}
  68 train_labels =
[class_to_label[os.path.dirname(path).split(os.path.sep)[-1]]
for path in train_paths]
  69 val_labels =
[class_to_label[os.path.dirname(path).split(os.path.sep)[-1]]
for path in val_paths]
  70 test_labels =
[class_to_label[os.path.dirname(path).split(os.path.sep)[-1]]
for path in test_paths]
  71
  72 # Create TensorFlow datasets with labels
  73 train_ds = create_dataset(train_paths, train_labels)
  74 val_ds = create_dataset(val_paths, val_labels)
  75 test_ds = create_dataset(test_paths, test_labels)
  76
  77 # Confirm class labels are assigned
  78 print(class_names)
  79
  80 # Define number of classes based on class labels length
  81 num_classes = len(class_names)
  82
  83
  84 # Implement data augmentation to enhance training set size
  85 data_augmentation = Sequential([
```

```
  86     layers.RandomFlip("horizontal_and_vertical",
input_shape=(img_height, img_width, 3)),
  87     layers.RandomRotation(0.2, fill_mode='nearest'),
  88     layers.RandomZoom(0.1),
  89     layers.RandomContrast(0.2),
  90     layers.RandomBrightness(0.2),
  91     layers.GaussianNoise(0.1),
  92 ])
  93
  94 # Define layers and add dropout as a regularizer
  95 model = Sequential([
  96     data_augmentation,
  97     layers.Rescaling(1./255),
  98     layers.Conv2D(8, 3, padding='same',
activation='relu',input_shape=(img_height, img_width, 3)),
  99     layers.Conv2D(8, 3, padding='same', activation='relu'),
 100     layers.MaxPooling2D(),
 101     layers.Conv2D(16, 3, padding='same',
activation='relu'),
 102     layers.Conv2D(16, 3, padding='same',
activation='relu'),
 103     layers.MaxPooling2D(),
 104     layers.Conv2D(32, 3, padding='same',
activation='relu'),
 105     layers.MaxPooling2D(),
 106     layers.Conv2D(64, 3, padding='same',
activation='relu'),
 107     layers.MaxPooling2D(),
 108     layers.Conv2D(128, 3, padding='same',
activation='relu'),
 109     layers.MaxPooling2D(),
 110     layers.Dropout(0.1),
 111     layers.Flatten(),
 112     layers.Dense(256, activation='relu'),
 113     layers.Dropout(0.05),
 114     layers.Dense(num_classes, name="outputs")
 115 ])
 116
 117 # Define early stopping
 118 early_stopping = EarlyStopping(monitor='val_accuracy',
mode='max', patience=10, verbose=1, restore_best_weights=True)
 119
 120 # Define ReduceLROnPlateau callback for learning rate
scheduling
 121 reduce_lr = ReduceLROnPlateau(monitor='val_loss',
factor=0.4, patience=5, min_lr=1e-6, verbose=1)
 122
 123 # Define the custom callback class
 124 class SaveBestWeights(tf.keras.callbacks.Callback):
 125     def __init__(self, filepath, threshold=5):
 126         super(SaveBestWeights, self).__init__()
 127         self.filepath = filepath
 128         self.threshold = threshold
```

```
        self.best_val_acc = -1  # Initialize with a
negative value
    def on_epoch_end(self, epoch, logs=None):
        train_acc = logs.get('accuracy')
        val_acc = logs.get('val_accuracy')
        if train_acc is None or val_acc is None:
            return
        diff = np.abs(train_acc - val_acc) * 100  #
Calculate the absolute difference in percentage
        if val_acc > self.best_val_acc and diff <
self.threshold:
            self.best_val_acc = val_acc
            self.model.save_weights(self.filepath,
overwrite=True)
            print(f"Epoch {epoch + 1}: Saved best weights
with validation accuracy: {val_acc:.2f}%")

# Define the custom callback to save best weights
best_weights_callback =
SaveBestWeights(filepath='best_weights.h5', threshold=5)

# Compile the model with adjusted learning rate
model.compile(optimizer=Adam(learning_rate=0.00095),

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=T
rue),
              metrics=['accuracy'])

# Train the model with early stopping
epochs = 75
history = model.fit(
     train_ds,
     validation_data=val_ds,
     epochs=epochs,
     callbacks=[early_stopping, reduce_lr,
best_weights_callback]
)

# Save the entire model
model.save('best_model.keras')

# Load the entire model
loaded_model =
tf.keras.models.load_model('best_model.keras')

## Evaluating and plotting
# Calculate top-k accuracies
# Softmax function to convert logits to probabilities
def softmax(logits):
    exp_logits = np.exp(logits)
    return exp_logits / np.sum(exp_logits, axis=1,
keepdims=True)

```

```
171 # Define function to calculate top-k accuracy
172 def top_k_accuracy(y_true, y_pred, k=5):
173     true_indices = np.argmax(y_true, axis=1)  # Get index
of true label
174     top_k_indices = np.argsort(y_pred, axis=1)[:, -k:]  #
Get indices of top-k predictions
175     top_k_correct = np.any(top_k_indices == true_indices[:,
np.newaxis], axis=1)
176     return np.mean(top_k_correct)
177
178 # Convert single integer labels to one-hot-encoding
179 num_classes = len(class_names)
180 def create_one_hot(labels):
181     return np.eye(num_classes)[labels]
182
183 # Function to predict probabilities
184 def predict_probabilities(model, dataset):
185     probabilities = model.predict(dataset)
186     return softmax(probabilities)
187
188 # Convert validation labels to one-hot encoding
189 val_labels = np.concatenate([y for x, y in val_ds], axis=0)
190 val_labels_one_hot = create_one_hot(val_labels)
191
192 # Get predicted probabilities on validation set
193 pred_prob_val = predict_probabilities(model, val_ds)
194
195 # Calculate top-k accuracy
196 top_3_acc_val = top_k_accuracy(val_labels_one_hot,
pred_prob_val, k=3)
197 top_5_acc_val = top_k_accuracy(val_labels_one_hot,
pred_prob_val, k=5)
198 top_20_acc_val = top_k_accuracy(val_labels_one_hot,
pred_prob_val, k=20)
199
200 print('Validation Top-3 accuracy:', top_3_acc_val)
201 print('Validation Top-5 accuracy:', top_5_acc_val)
202 print('Validation Top-20 accuracy (Sanity check):',
top_20_acc_val)
203
204 # Confusion matrix
205 # Function to create confusion matrix
206 def plot_confusion_matrix(y_true, y_pred, class_names):
207     cm = confusion_matrix(y_true, y_pred, normalize='true')
208     plt.figure(figsize=(10, 10))
209     sns.heatmap(cm, annot=True, cmap='Blues',
xticklabels=class_names, yticklabels=class_names)
210     plt.xlabel('Predicted Labels', fontsize= 11, fontweight
= 'bold')
211     plt.ylabel('True Labels', fontsize= 11, fontweight =
'bold')
212     plt.title('Confusion Matrix', fontsize= 15, fontweight
= 'bold')
```

```
 213     plt.tight_layout()
 214     plt.show()
 215
 216 # Get predicted labels for validation set
 217 pred_labels = np.argmax(predict_probabilities(model,
val_ds), axis=1)
 218
 219 # Get true labels
 220 true_labels = np.concatenate([y for x, y in val_ds],
axis=0)
 221
 222 # Plot confusion matrix
 223 plot_confusion_matrix(true_labels, pred_labels,
class_names)
 224
 225 ## Plot top 5 predicted labels
 226 #  Function to plot the image with the predicted and true
labels
 227 def plot_image(predictions_array, true_label, img,
class_names):
 228     plt.grid(False)
 229     plt.xticks([])
 230     plt.yticks([])
 231     # Ensure the image is displayed in its original quality
 232     plt.imshow(img.astype('uint8'))
 233     predicted_label = np.argmax(predictions_array)
 234     if predicted_label == true_label:
 235         color = 'blue'
 236     else:
 237         color = 'red'
 238     plt.xlabel("{} {:2.0f}%
({})".format(class_names[predicted_label],
 239
100*np.max(predictions_array),
 240
class_names[true_label]),
 241                color=color)
 242
 243 # Function to plot the bar chart of top 5 predicted labels
and their percentages
 244 def plot_value_array(predictions_array, true_label,
class_names):
 245     plt.grid(False)
 246     predictions_percent = predictions_array * 100
 247
 248     # Find the indices of the top 5 predictions
 249     top_indices = np.argsort(predictions_array)[::-1][:5]
 250     top_percentages = predictions_percent[top_indices]
 251     # Combine label and integer
 252     top_class_names = [f"{class_names[i]} ({i})" for i in
top_indices]
 253     plt.xticks(range(5), top_class_names)  # Display top 5
combined labels on x-axis
```

```
    # Set y-axis display range 0 to 100, name y- and x-axis
respectively
    plt.ylim([0, 100])
    plt.ylabel('Percentage', fontsize=12,
fontweight='bold')
    plt.xlabel('Predicted labels', fontsize=12,
fontweight='bold', labelpad=12)
    thisplot = plt.bar(range(5), top_percentages,
color="#777777", width=0.7)
    # Find the position of the true label among the top 5
predictions
    true_label = int(true_label)
    true_label_index = np.where(top_indices ==
true_label)[0]
    if true_label_index.size > 0:
        thisplot[true_label_index[0]].set_color('blue')
    # Color bars corresponding to incorrect predictions in
red
    for i in range(len(thisplot)):
        if i != true_label_index[0]:
            thisplot[i].set_color('red')
    # Assign probability percentages for classes
    for i, label in enumerate(top_indices):
        plt.text(i, top_percentages[i] + 2,
f'{top_percentages[i]:.1f}%', ha='center', va='bottom',
color='black')

# Plot predictions where i = x:
i = 4 # Change index value to display different image from
test dataset.

# Predictions for the test set
probability_model = tf.keras.Sequential([model,
tf.keras.layers.Softmax()])
predictions = probability_model.predict(test_ds)

# Extract images and labels from test_ds
test_images = []
test_labels = []
for image_batch, label_batch in test_ds:
    test_images.append(image_batch.numpy())
    test_labels.append(label_batch.numpy())

test_images = np.concatenate(test_images)
test_labels = np.concatenate(test_labels)

plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plot_image(predictions[i], test_labels[i], test_images[i],
class_names)
plt.subplot(1, 2, 2)
```

```
 294 plot_value_array(predictions[i], test_labels[i],
class_names)
 295 plt.tight_layout()
 296 plt.show()
```

Appendix 3. Script for Training and Evaluating GPU Variant Model (Adapted from TensorFlow 2024a–l)

```
# Import dependencies
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import pandas as pd
import seaborn as sns

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import regularizers
from sklearn.metrics import confusion_matrix

# Loading and preprocessing
# Define parameters for loader
batch_size = 32
img_height = 256
img_width = 256
data_dir = r'Path:\to\dataset\folder'

# Create training set with 80% split
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

# Create validation set with 20% split
val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

# Confirm class labels are assigned
class_names = train_ds.class_names
print(class_names)

# Define number of classes based on class labels length
```

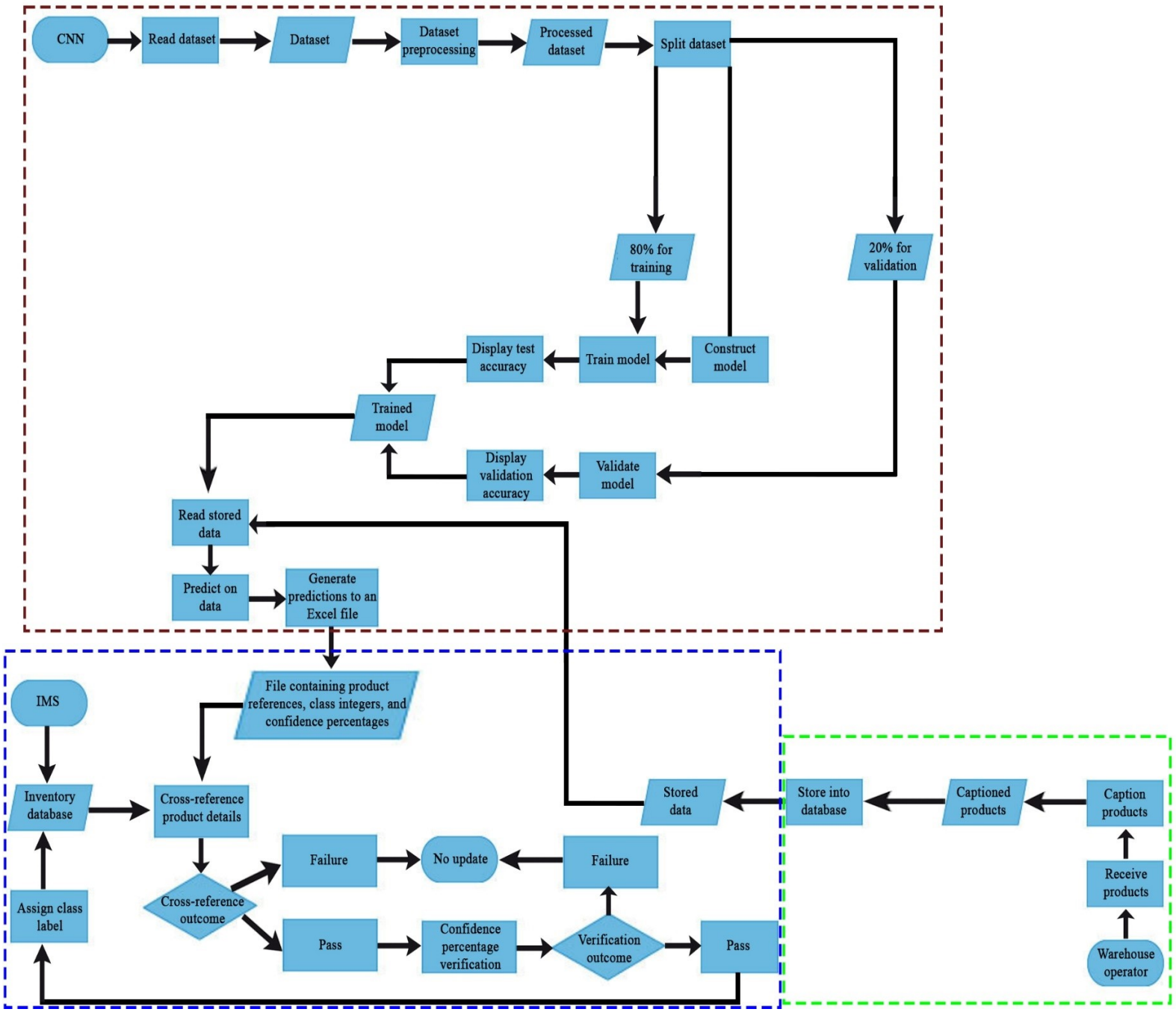
```
num_classes = len(class_names)

#Configure dataset for performance
AUTOTUNE = tf.data.AUTOTUNE
train_ds =
train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

# Define layers and add dropout as a regularizer
#Create the model with dropout regularization
model = Sequential([
    layers.Rescaling(1./255),
    layers.Conv2D(8, 3, padding='same', activation='relu',
input_shape=(img_height, img_width, 3)),
    layers.MaxPooling2D(),
    layers.Conv2D(16, 3, padding='same',
activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same',
activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same',
activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(128, 3, padding='same',
activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(256, 3, padding='same',
activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(512, 3, padding='same',
activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.1),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes, name="outputs")
])

# Define early stopping
early_stopping = EarlyStopping(monitor='val_accuracy',
mode='max', patience=10, verbose=1, restore_best_weights=True)

# Define ReduceLROnPlateau callback for learning rate
scheduling
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
factor=0.4, patience=5, min_lr=1e-6, verbose=1)

# Define the custom callback class
class SaveBestWeights(tf.keras.callbacks.Callback):
    def __init__(self, filepath, threshold=5):
        super(SaveBestWeights, self).__init__()
```

```
        self.filepath = filepath
        self.threshold = threshold
        self.best_val_acc = -1  # Initialize with a
negative value
    def on_epoch_end(self, epoch, logs=None):
        train_acc = logs.get('accuracy')
        val_acc = logs.get('val_accuracy')
        if train_acc is None or val_acc is None:
            return
        diff = np.abs(train_acc - val_acc) * 100  #
Calculate the absolute difference in percentage
        if val_acc > self.best_val_acc and diff <
self.threshold:
            self.best_val_acc = val_acc
            self.model.save_weights(self.filepath,
overwrite=True)
            print(f"Epoch {epoch + 1}: Saved best weights
with validation accuracy: {val_acc:.2f}%")

# Define the custom callback to save best weights
best_weights_callback =
SaveBestWeights(filepath='best_weights.h5', threshold=5)

# Compile the model with adjusted learning rate
model.compile(optimizer=Adam(learning_rate=0.00095),

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=T
rue),
              metrics=['accuracy'])

# Train the model with early stopping
epochs = 75
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs,
    callbacks=[early_stopping, reduce_lr,
best_weights_callback]
)

# Save the entire model
model.save('best_model.keras')

# Load the entire model
loaded_model =
tf.keras.models.load_model('best_model.keras')
```

Appendix 4. Script for Generating Test Set Predictions to an Excel File (Stack Overflow, 2019b; TensorFlow, 2024a)

```
# Import dependencies
import os
import pandas as pd
import numpy as np

# Path to the folder containing test images
folder_path = r'Path:\to\test\dataset\folder’

# List all files in the folder
image_files = os.listdir(folder_path)

# Define image dimensions
img_height, img_width = 256, 256

# Initialize an empty DataFrame to store all predictions
all_predictions_df    =    pd.DataFrame(columns=['Filename',
'Class Integer', 'Class', 'Confidence (%)'])

for image_file in image_files:
    # Load and preprocess the image
    img_path = os.path.join(folder_path, image_file)
    img          =          tf.keras.utils.load_img(img_path,
target_size=(img_height, img_width))
    img_array = tf.keras.utils.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0) # Create a batch
    # Perform prediction
    predictions = model.predict(img_array)
    score = tf.nn.softmax(predictions[0])
    # Get predicted class, its integer and confidence
    predicted_class_index = np.argmax(score)
    predicted_class = class_names[np.argmax(score)]
    confidence_percentage = 100 * np.max(score)
    # Append predictions to DataFrame
    all_predictions_df = all_predictions_df._append({
        'Filename': image_file,
        'Class Integer': predicted_class_index,
        'Class': predicted_class,
        'Confidence (%)': confidence_percentage
    }, ignore_index=True)

# Save the predictions to an Excel file
all_predictions_df.to_excel(r'Path:\to\save\predictions\Pre
dictions.xlsx', index=False)
```

Appendix 5. Integrated Process Flow for Classification Process and Inventory Management System



Appendix 6. Script for Generating Class Activation Mapping Images (Adapted from Keras, 2021)

```
# Presuming a model has been trained or loaded according to
Appendices 2–3

# Import dependencies
import numpy as np
import tensorflow as tf
import keras

from IPython.display import Image, display
import matplotlib as mpl
import matplotlib.pyplot as plt

# Define the Grad-CAM function
def grad_cam(model, img, class_index, layer_name):
    with tf.GradientTape() as tape:
        last_conv_layer = model.get_layer(layer_name)
        iterate   =   Model([model.inputs],   [model.output,
last_conv_layer.output])
        model_out, last_conv_layer = iterate(img)
        class_out = model_out[:, class_index]
    grads = tape.gradient(class_out, last_conv_layer)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    last_conv_layer_output = last_conv_layer[0]
    heatmap                                                =
tf.reduce_mean(tf.multiply(last_conv_layer_output,
pooled_grads), axis=-1)
    heatmap = np.maximum(heatmap, 0) / np.max(heatmap)
    return heatmap

# Identify the last convolutional layer name
last_conv_layer_name  =  'conv2d_20'  #  model.summary()  to
display the layers, change to last convolutional layer

# Prepare image
def get_img_array(img_path, size):
    img = keras.utils.load_img(img_path, target_size=size)
    array = keras.utils.img_to_array(img)
    array = np.expand_dims(array, axis=0)
    return array

# Make Grad-CAM heatmap
def          make_gradcam_heatmap(img_array,          model,
last_conv_layer_name):
    grad_model = keras.models.Model(
        [model.inputs],
[model.get_layer(last_conv_layer_name).output,  model.output]
    )
```

```
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        pred_index = tf.argmax(preds[0])
        class_channel = preds[:, 14] # Change integer to
interpret for class x
    grads = tape.gradient(class_channel,
last_conv_layer_output)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    heatmap = last_conv_layer_output[0] @ pooled_grads[...,
tf.newaxis]
    heatmap = tf.squeeze(heatmap)
    heatmap = tf.maximum(heatmap, 0) /
tf.math.reduce_max(heatmap)
    return heatmap.numpy()

# Load an image for visualization
img_path = r'Path:\to\image\image.jpg' # Change to preferred
image
img_array = get_img_array(img_path, size=(img_height,
img_width))

# Generate Grad-CAM heatmap
heatmap = make_gradcam_heatmap(img_array, model,
last_conv_layer_name)
# Display heatmap
plt.matshow(heatmap)
plt.show()

def save_and_display_gradcam(img_path, heatmap,
cam_path="cam.jpg", alpha=0.4): # Adjusting alpha value
affects the overlay strength of coloring
    # Load the original image
    img = keras.utils.load_img(img_path)
    img = keras.utils.img_to_array(img)
    # Rescale heatmap to a range 0-255
    heatmap = np.uint8(255 * heatmap)
    # Use jet colormap to colorize heatmap
    jet = mpl.colormaps["jet"]
    # Use RGB values of the colormap
    jet_colors = jet(np.arange(256))[:, :3]
    jet_heatmap = jet_colors[heatmap]
    # Create an image with RGB colorized heatmap
    jet_heatmap = keras.utils.array_to_img(jet_heatmap)
    jet_heatmap = jet_heatmap.resize((img.shape[1],
img.shape[0]))
    jet_heatmap = keras.utils.img_to_array(jet_heatmap)
    # Superimpose the heatmap on original image
    superimposed_img = jet_heatmap * alpha + img
    superimposed_img =
keras.utils.array_to_img(superimposed_img)
    # Save the superimposed image
    superimposed_img.save(cam_path)
    # Display Grad CAM
```

```
    display(Image(cam_path))
cam_path = r'Path:\where\generated\image is saved\image.jpg'
# Preferred save location
print("Before saving:", cam_path)
save_and_display_gradcam(img_path, heatmap, cam_path)
```