UNIVERSITY WEST

May 24, 2024

AI AIDED QUALITY CONTROL AT NOLATO CERBO: AN INVESTIGATIVE STUDY ON THE APPLICATION OF AI IN DETECTING PRODUCT BATCH NUMBER.

OLANIYI SUNDAY JOSEPH AAMIR BHATTI

PROJECT REPORT
PIA700 PROJECT WORK IN ARTIFICIAL INTELLIGENCE AND AUTOMATION.
Department of Engineering Science

Preface

This project, titled "AI Aided Quality Control at Nolato Cerbo: An Investigative Study on the Application of AI in Detecting Product Batch Number," represents a comprehensive exploration into the integration of Artificial Intelligence (AI) within the quality control processes at Nolato Cerbo. Conducted as part of the PIA700 Project Work in Artificial Intelligence and Automation at University West, this project reflects the collaborative efforts of Olaniyi Sunday Joseph and Aamir Bhatti under the supervision of our examiner, Mahmood Khabbazi.

The motivation for this study stems from the critical importance of quality control in industrial manufacturing, where maintaining product consistency, functionality, and adherence to industry standards are paramount. Traditional methods often fall short in efficiency and accuracy, particularly in identifying and verifying batch numbers on small or intricately marked products. This project aims to address these challenges by leveraging AI-driven Machine Vision (MV) systems to automate and enhance the quality control process.

The report details the development and implementation of an MV solution designed to capture 3-dimensional data of plastic tubes, focusing on the accurate detection of batch numbers. Advanced imaging techniques, deep learning algorithms, and transfer learning are employed to achieve high precision and reliability in batch number identification, significantly reducing the potential for human error.

Throughout this project, various Support Vector Machine (SVM) models, both linear and non-linear, along with deep learning feature extractors like VGG16 and ResNet50, were rigorously tested. The results demonstrate the robustness and adaptability of the proposed system, highlighting its potential to revolutionize quality control in industrial settings.

This work not only contributes to the academic field of AI and automation but also offers practical insights and solutions for the manufacturing industry, aligning with the goals of Industry 4.0. We extend our gratitude to the Department of Engineering Science at University West for their support and resources that made this project possible. We also thank Nolato Cerbo for their collaboration and provision of the necessary materials for this study.

We hope this report provides valuable knowledge and sparks further innovations in the application of AI for quality control and beyond.

Date: May 24, 2024

Authors:
Olaniyi Sunday Joseph
Aamir Bhatti

Course: PIA700 Project Work in Artificial Intelligence and Automation

Examiner: Mahmood Khabbazi

University West
Department of Engineering Science

AI AIDED QUALITY CONTROL AT NOLATO CERBO: AN INVESTIGATIVE STUDY ON THE APPLICATION OF AI IN DETECTING PRODUCT BATCH NUMBER.

Abstract

This investigative study explores the application of Artificial Intelligence (AI) in quality control at Nolato Cerbo, focusing on detecting product batch numbers using AI-driven Machine Vision (MV) systems. Quality control in industrial manufacturing is vital for ensuring product consistency, functionality, reliability, and adherence to industry standards. AI enhances these processes by automating inspections, providing advanced image analysis, enabling real-time monitoring, predicting maintenance needs, and integrating with Industry 4.0 technologies.

Our study introduces an MV solution that captures 3-dimensional data to identify batch numbers inscribed on plastic tubes. Utilizing a high-quality industrial camera under controlled lighting, the system employs deep learning algorithms and transfer learning to recognize patterns and accurately identify batch numbers from various orientations. This approach mitigates human errors and improves inspection efficiency, especially for small-sized products or complex batch codes.

The MV system's effectiveness was tested using Support Vector Machine (SVM) models, including both linear and non-linear variants, and deep learning feature extractors like VGG16 and ResNet50. The results showed that both linear and non-linear SVM models achieved perfect performance with clean data, achieving an overall accuracy, precision, recall, and F1-score of 1.00 across all classes. However, when tested with noisy data, the models showed variability in performance. The linear SVM with noisy data achieved an overall accuracy of 85%, with precision and recall metrics ranging from 0.75 to 0.94 across different batches. Non-linear SVMs showed slightly better robustness, achieving an overall accuracy of 87% with noisy data. The addition of deep learning feature extractors, especially ResNet50, further enhanced the performance, with the non-linear SVM + ResNet50 model achieving an overall accuracy of 87% with noisy data.

This study highlights the potential of AI-driven MV systems in revolutionizing quality control processes, offering significant improvements in accuracy, reliability, and efficiency. The integration of advanced imaging techniques and machine learning algorithms presents a promising solution for the industrial manufacturing sector, ensuring high standards of product quality and customer satisfaction.

Date:	May 24, 2024	
Author(s):	Olaniyi Sunday Joseph	Aamir Bhatti
Course:	PIA700 Project Work in Artificial Intelligence and Automation	
Examiner:	Mahmood Khabbazi.	
	University West, Department of Engineering Science	

Contents

Nomenclature

Glossary

AI	= Artificial Intelligence
MV	= Machine vision
SVM	= Support Vector Machine
Linear	= Linear kernel
Non-linear	= Poly kernel

1 Introduction

Quality control is a critical component of industrial manufacturing, encompassing all procedures and tasks designed to ensure products maintain consistent functionality, reliability, and safety, while adhering to industry standards and regulations. Effective quality control is essential for manufacturers to achieve high levels of customer satisfaction, trust, and loyalty. To be successful, these processes must be flexible, highly repeatable, and cost-effective.

AI enhances quality control by automating inspection processes, providing advanced image analysis, enabling real-time monitoring, predicting maintenance needs, integrating with Industry 4.0 technologies, and ensuring high customer satisfaction. These applications collectively improve the efficiency, accuracy, and reliability of quality control systems in industrial manufacturing.

AI-driven Machine Vision (MV) systems have become increasingly significant in both daily applications and industrial operations. In recent years, MV has been pivotal in enhancing various industrial processes, including process control, monitoring, and particularly in automated product inspection during production, post-production, and final inspection stages as part of comprehensive quality control systems. These systems use advanced imaging techniques and deep learning algorithms to detect defects and deviations from standards with high precision.

Engineers have developed state-of-the-art MV techniques to achieve these goals. These techniques include deep learning-based image processing, algorithmic analysis of image dimensions, and the use of neural networks and transfer learning. These advancements help mitigate human errors caused by stress and fatigue from performing repetitive tasks.

Common quality control task in the industry is ability to be able to identify/group product based on their batch of production since industrial products are produced in batches, and some batches within the production time might be defected, identifying the defect batch and being able to inspect and extract them is very important and in most cases an enormous and time consuming when the product especially is small in size or the batch numbers are a series of code. This is a similar task that quality control personnel at Nolato Cerbo engaged in on daily basis.

The AI-driven MV solution investigated and proposed in this report is designed to capture 3-dimensional data for analysis. The MV system captures the image of the lower end of the plastic tube here the batch number is inscribed using a high-quality industrial camera under a controlled lightning condition. With these pixels input, the system was trained using machine learning algorithm and transfer learning to understand patterns of each batch number and identify them.

The solution is highly adaptable and robust with the flexibility of identifying the batch numbers from different orientations to the camera.

2 Brief Review of Application of AI-driven Machine vision Systems for Quality Control.

Machine vision (MV) has proven to have the needed potential for industries to be able to meet up with the task of ensuring that a high-quality product that meets customer demands and satisfaction, standard and other safety regulations guiding such production by aiding in a timely, efficient, and repeatable quality control. As such eliminating human errors due to fatigue from repetitive task and other environmental factors [4]. As such several implementations of Machine vision have been engineered, although there is no any organized, defined framework guiding the implementation of a quality control based Machine vision systems [2].
Attempts have been made in the recent to evaluate an approach for MV based quality control but does not necessarily generalize a wholistic design or method, it rather focuses more on the general components of such system, how individual components influence and relate with each other, and how it can be implemented to ensure less implementation time and ultimately less cost [2, 5].
MV with several other Machine Learning algorithms have been used in quality control [1, 6] , for instance; in inspecting empty glass bottles for abnormality in the bottle mouth, the used technique and algorithm had an accuracy score of 99.41%, also, Huang et al. uses Support Vector Machine (SVM) to study the detection of bottle mouth defects with an accuracy rate of 91.6% [6]. In another study, Huang et al. also use MV with connected domain search algorithm; an algorithm that works by tracking specific defined dimension of the bottle and comparing such with predefine base parameters and thereby calculating average error(s) to classify the bottle mouth [6]. Akundi et al seems to have follow the same approach as above in their studies, by using MV with a dimensional analysis algorithm similar to the connected domain search algorithm [4]. Andriosopoulou *et al* study the recognition of surface defects in die-cast parts using deep neural networks, specifically faster R_CNN and YOLOv5, to improve accuracy and consistency in quality control.

3 Proposed AI-Driven Machine Vision System.

3.1 MV Set up and Image Acquisition.

The developed AI-Driven machine vision system uses a Basler a2A2448-75u cPRO industrial camera, a rectangular white light source, and the sample: plastic tube provided by Nolato Cerbo. Data acquisition was carried out at the Vision lab of Hoskolan Vast. The vision lab provided a unit of control chamber in a box which make it possible to control the lighting and reflection around the lab. The camera provided a wide depth of view with high focusing capabilities. A 3-channel image with shape (2048, 2448, 3) was captured for each provided batch sample.

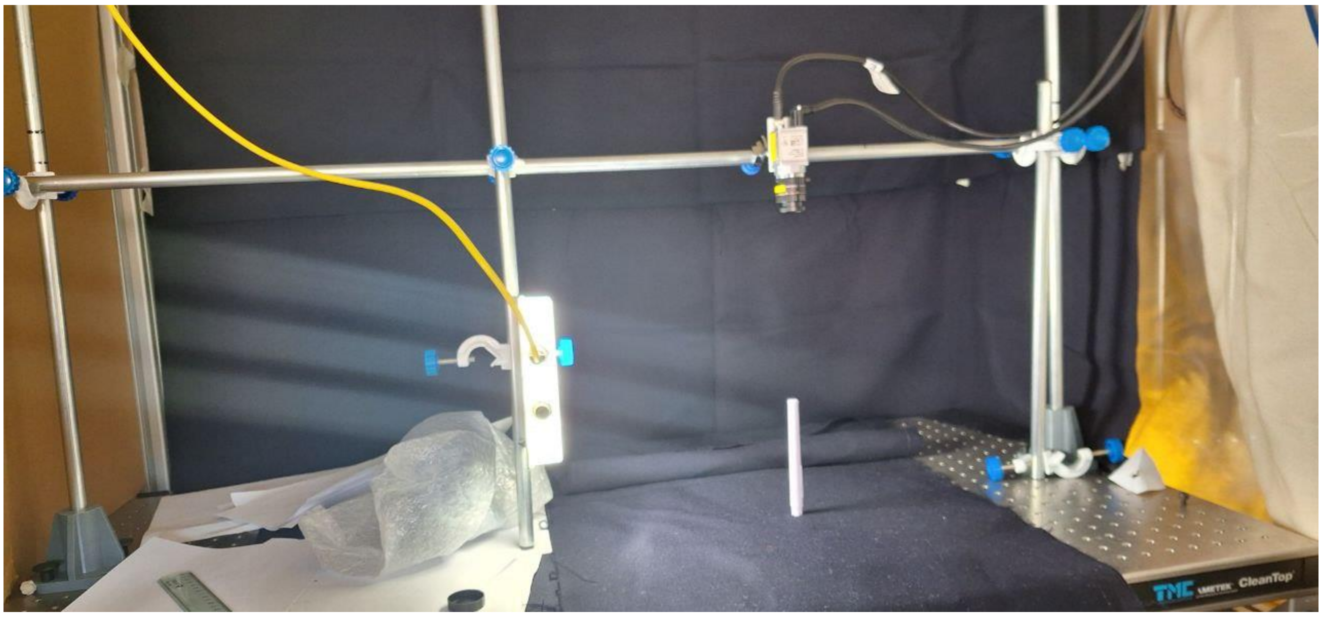
Figure 1. Machine Vision System Set up.

The camera was mounted at about 15.3cm to the object and zoomed to provide effective coverage and focused on the tube tips to be able to capture the batch number effectively. Also, the box is lined with non-reflective material or materials that is capable of absorbing light so that light only travels from the area of interest from the object/sample to the camera. The position of the light also helps to obtain optimum possible image with less light noise in the data. Image for each batch was captured such that the object is centrally placed in the camera view and slightly tilted in different two other directions to have different view of the images from slightly different orientation. See the figures 2 shows the data captured while the object is centrally placed in the camera feed, figure 3 shows the same image while the object is slightly rotated and moved towards the right side of the camera feed and figure 4 shows the image when the object is slightly rotated and moved towards the right side of the camera feed. This is to help have robust data and enrich model learning and performance.

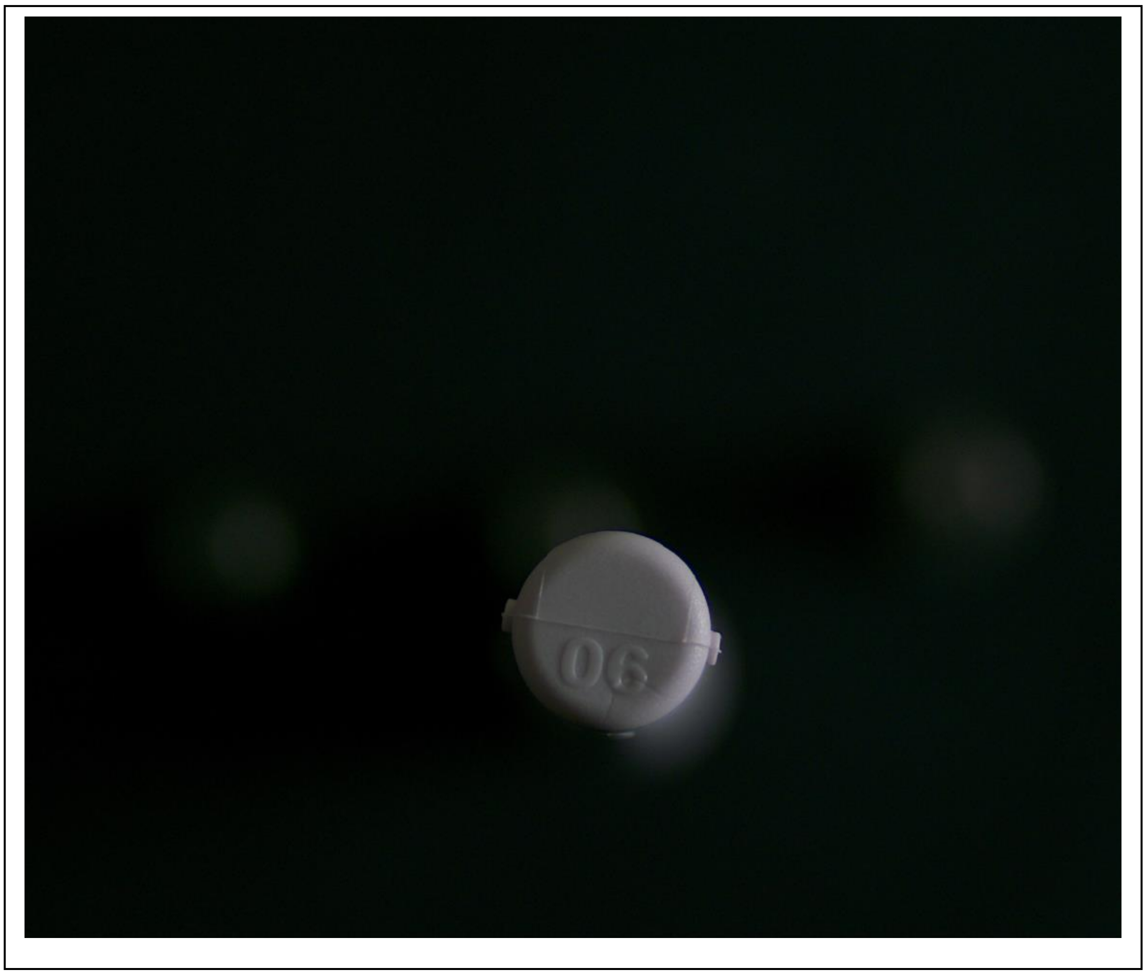

Figure 2: Object placed in the camera feed center.

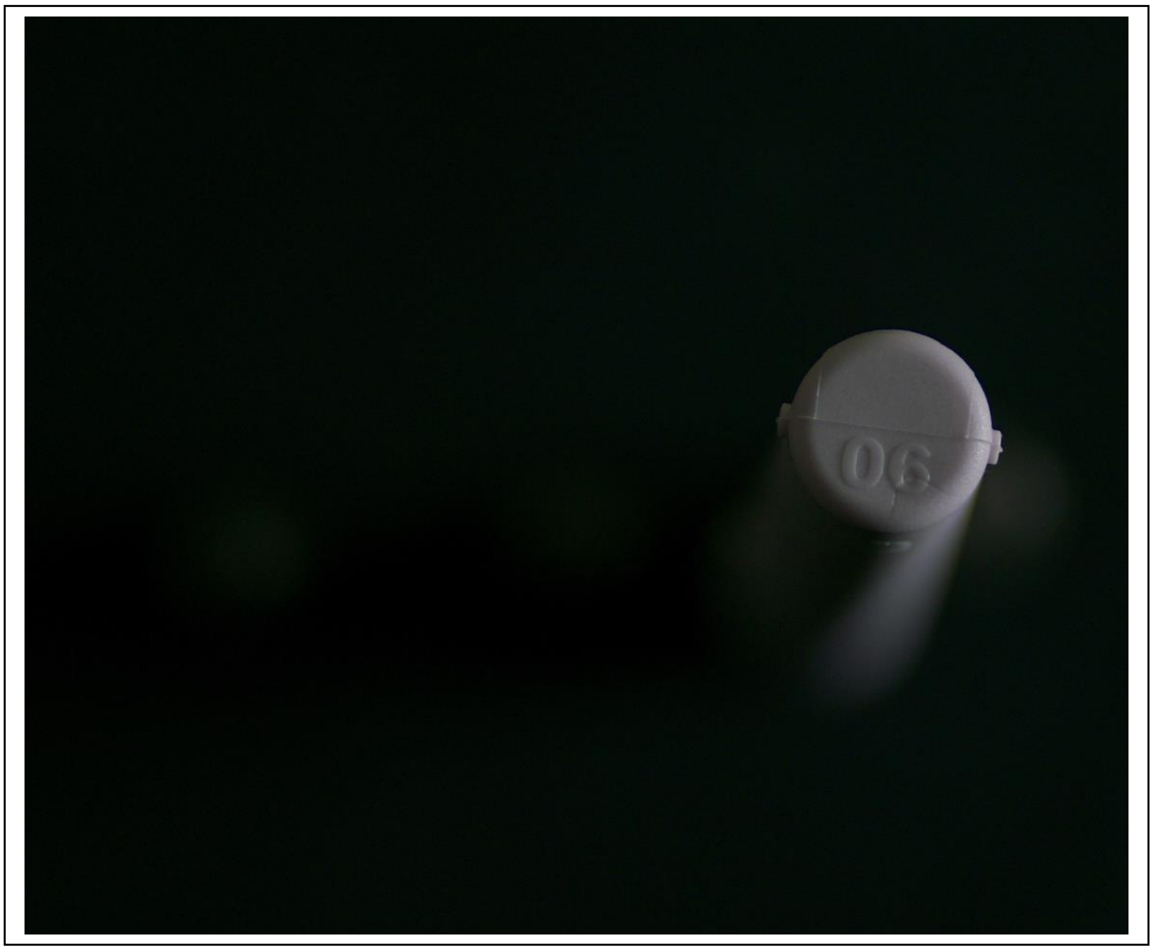

Figure3: Object placed in the camera feed towards right.

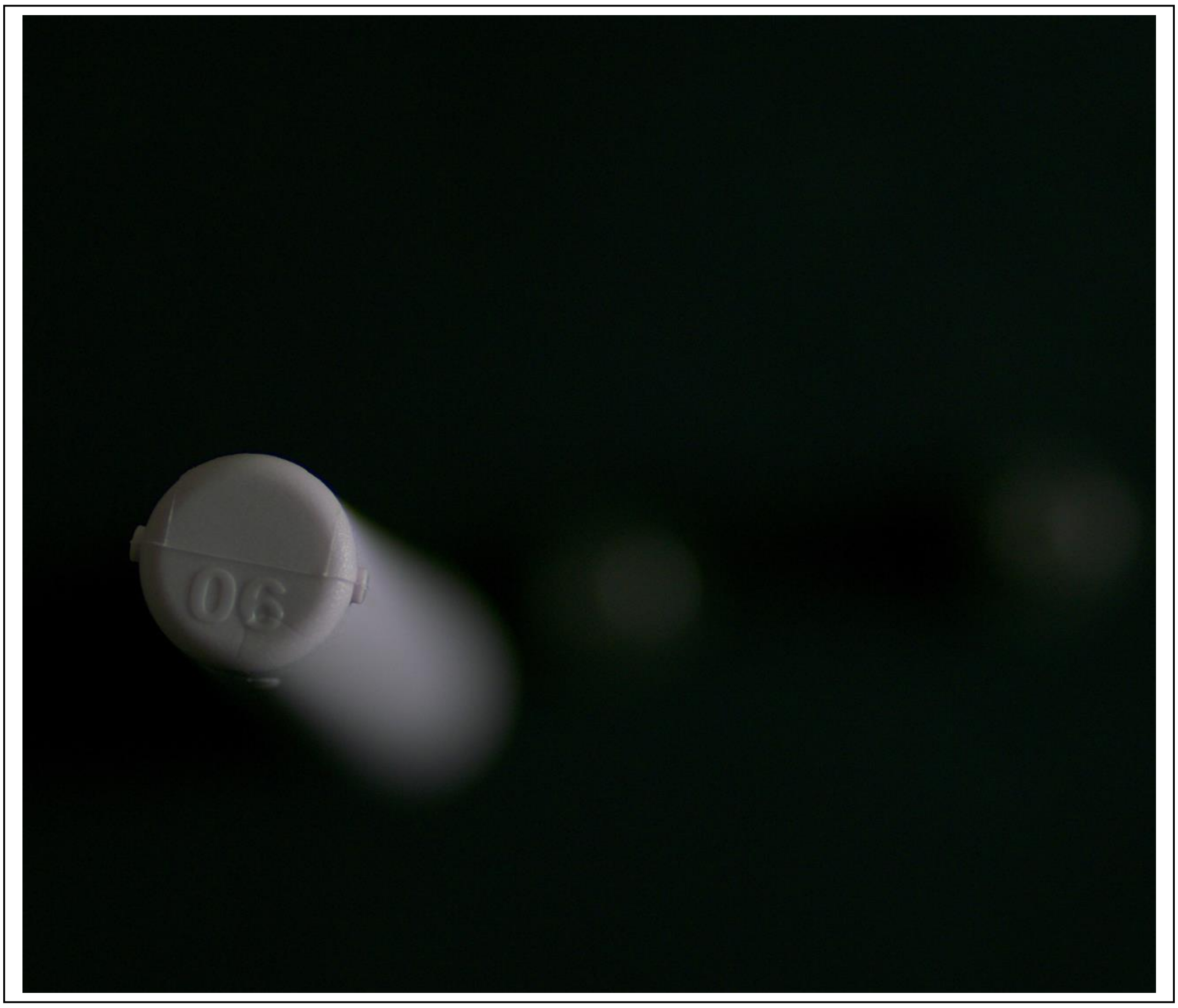

Figure 4: Object placed in the camera feed towards left.

3.2 Data Preprocessing.

A high-quality image preprocessing algorithm is used to process the acquired data from the vision system, using OpenCV and python capabilities for fast processing. The image was normalized, and skew corrected to reduce noise from illumination, to maintain a consistent and standardized data representation and to ensure that clean and more accurate data is passed through the system pipeline, reducing the risk of errors down the line and ultimately help the model algorithm to have a good, well-prepared data. The image normalized image was further rotated at angles 30 and 100 degrees to provide more robustness to the machine learning algorithm and also augmenting the data in the process.

3.3 Machine Learning Algorithm and Training/Classification.

The Support Vector Machine (SVM), a powerful machine learning algorithm was used to perform this task of correctly classifying the batches using a multi-label-multiclass classification technique. SVM offers high accuracy, robustness, and versatility. Their ability to handle high-dimensional data and complex boundaries makes them essential for many real-world applications. It is also scalable to handle large datasets effectively, especially when used with kernel methods. However, the choice of kernel and regularization parameters has the potential of influencing computational efficiency.

This model relies on rigorous mathematical foundation and objective of maximizing the margin and this makes it reliable for many tasks.

The preprocessed data was split into batches and properly labelled. A few variants of the SVM were tested using a total of 900 datasets in order to have a broad study in to the capability of this algorithm. A linear SVM is trained, also a non-linear kernel SVM is trained. The capability of ResNet50 and VGG16 architecture was also explored for feature extraction and used the extracted feature to train an SVM model for classification. All the SVM variants perform effectively well on the prediction with new dataset collected and treated under the same condition as the training data. They all come with different computational requirements and performance. In order to have a comprehensive comparative insight into the model's performance, the dataset is slightly altered by deliberately introducing noise into the dataset by adding some samples of wrong batch into each batch and this is regarded as noisy dataset. Overall, the models used include Linear SVM and Non-Linear SVM.

The preprocessing techniques include the base model (no additional feature extraction), VGG16, and ResNet50, and the data types include clean data and noisy data.

4 Result Presentation and Interpretations.

4.1 Initial SVM Result.

The result of the linear SVM with the clean dataset is presented below:

	Precision	Recall	F1-score	Support
BATCH03	1	1	1	20
BATCH04	1	1	1	22
BATCH05	1	1	1	24
BATCH06	1	1	1	15
BATCH07	1	1	1	11
BATCH08	1	1	1	10
BATCH09	1	1	1	14
BATCH10	1	1	1	19
accuracy			1	135
macro avg	1	1	1	135
weighted avg	1	1	1	135

Table 1: Classification metrics Linear SVM with Clean Data

The report shows metrics for eight classes (labeled from 0 to 7). For each class, the following metrics are reported: precision, recall, f1-score, and support.

For all classes (0 to 7), the precision is 1.00, which means that every instance predicted as a certain class was indeed that class. Recall is the ratio of correctly predicted positive observations to all observations in actual class. For all classes (0 to 7), the recall is 1.00, indicating that the model successfully identified all instances of each class. For all classes (0 to 7), the F1-Score is 1.00, demonstrating perfect balance between precision and recall for each class. The overall accuracy of the model is 1.00, meaning the model correctly classified all samples in the dataset.

Also, all macro average values are 1.00, confirming that, on average, the model performs perfectly across all classes. All weighted average values are 1.00, indicating perfect performance when considering the imbalance in class support. The classification report suggests that the model has achieved perfect classification performance on the test set, with all precision, recall, and F1-Score values being 1.00 for every class.

4.2 Results from other SVM models.

Due to the perfect performance of the first linear SVM model, a noisy dataset was generated as described in section 3, and this dataset is now used to trained extensively with different variants of SVM model to create a robust comparative analysis for the system. The result of the model's performance metrics is presented below.

	Macro Average			**Weighted Average**			
	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Accuracy
Linear SVM + Clean Data	1	1	1	1	1	1	1
Linear SVM + Noisy Data	0.86	0.84	0.85	0.85	0.85	0.85	0.85
Non-Linear SVM + Clean Data	1	1	1	1	1	1	1
Non-Linear SVM + Noisy Data	0.87	0.86	0.86	0.87	0.87	0.87	0.87
Linear SVM + VGG16 + Clean Data	1	1	1	1	1	1	1
Linear SVM + VGG16 + Noisy Data	0.86	0.85	0.86	0.86	0.86	0.86	0.86
Non-Linear SVM + VGG16 + Clean Data	1	1	1	1	1	1	1
Non-Linear SVM + VGG16 + Noisy Data	0.86	0.85	0.85	0.86	0.87	0.86	0.86
Linear SVM + ResNet50 + Clean Data	1	1	1	1	1	1	1
Linear SVM + ResNet50 + Noisy Data	0.87	0.86	0.86	0.87	0.87	0.87	0.87
Non-Linear SVM + ResNet50 + Clean Data	1	1	1	1	1	1	1
Non-Linear SVM + ResNet50 + Noisy Data	0.86	0.85	0.85	0.86	0.85	0.85	0.85

Table 2: Performance metrics for all Model.

Linear SVM on clean data achieves perfect performance with all metrics (macro and weighted averages) being 1. This suggests the model perfectly classifies the clean data without errors. Linear SVM on noisy data performance drops with precision, recall, and F1-score all at 0.85 (macro and weighted averages), and accuracy at 0.85. This indicates the model's performance is affected by noise in the data.

Non-Linear SVM on clean data, similar to linear SVM on clean data, achieves perfect performance with all metrics being 1. Non-Linear SVM on noisy data performs slightly better than the linear SVM on noisy data, with all metrics at 0.87, showing some robustness to noise.

Linear SVM + VGG16 on clean data also achieves perfect performance with all metrics being 1, indicating that VGG16 helps maintain high performance on clean data. Linear SVM + VGG16 on noisy data performance is slightly better than linear SVM on noisy data without VGG16, with metrics at 0.86, suggesting VGG16 provides some noise robustness.

Non-Linear SVM + VGG16 on clean data also achieves perfect performance with all metrics being 1. Non-Linear SVM + VGG16 on noisy data similar to Linear SVM + VGG16 on noisy data, with metrics at 0.86.

Linear SVM + ResNet50 on clean data also achieves perfect performance with all metrics being 1. Linear SVM + ResNet50 on noisy data performs slightly better than the Linear SVM + VGG16 on noisy data, with metrics at 0.87.

Non-Linear SVM + ResNet50 on clean data also achieves perfect performance with all metrics being 1. Non-Linear SVM + ResNet50 on noisy data performs similar to Linear SVM + ResNet50 on noisy data, with metrics at 0.85.

5 Conclusion.

Both Linear and Non-Linear SVMs, regardless of whether they use VGG16 or ResNet50 preprocessing, achieve perfect performance on clean data. Moreso, non-linear SVMs generally perform slightly better than Linear SVMs on noisy data.
The use of VGG16 and ResNet50 helps improve the performance of SVMs on noisy data, with ResNet50 showing a slight edge over VGG16.
Non-Linear SVMs are more robust to noise compared to Linear SVMs. Also, preprocessing with VGG16 and ResNet50 enhances noise robustness, with ResNet50 being marginally better.
Conclusively, the result highlights the effectiveness of combining SVM models with deep learning feature extractors like VGG16 and ResNet50, especially in handling noisy data. While clean data leads to perfect classification across all models, noisy data challenges the models, revealing the superior performance of Non-Linear SVMs and the added benefits of deep learning preprocessing.

References

[1] Abdelfatah Ettalibi, Abdelmajid Elouadi, Abdeljebar Mansour, "AI and Computer Vision-based Real-time Quality Control: A Review of Industrial Applications," Procedia Computer Science, Volume 231, 2024, Pages 212-220.

[2] Tobias Reichenstein, Tim Raffin, Christian Sand, Jörg Franke, "Implementation of Machine Vision based Quality Inspection in Production: An Approach for the Accelerated Execution of Case Studies," Procedia CIRP, Volume 112, 2022, Pages 596-601.

[3] Andriosopoulou G, Mastakouris A, Masouros D, Benardos P, Vosniakos G-C, Soudris D. "Defect Recognition in High-Pressure Die-Casting Parts Using Neural Networks and Transfer Learning". Metals. 2023; 13(6):1104. https://doi.org/10.3390/met13061104

[4] Aditya Akundi, Mark Reyna, "A Machine Vision Based Automated Quality Control System for Product Dimensional Analysis," Procedia Computer Science, Volume 185, 2021, Pages 127-134.

[5] Anand, Sheila, and Loganathan, Priya. (2019). A Guide for Machine Vision in Quality Control. 10.1201/9781003002826.

[6] Bin Huang, Sile Ma, Ping Wang, Huajie Wang, Jinfeng Yang, Xinyi Guo, Weidong Zhang, and Huiquan Wang, "Research and implementation of machine vision technologies for empty bottle inspection systems," Engineering Science and Technology, an International Journal 21 (2018) 159-169.

Appendices

See the following pages.

A: Image Preprocessing Algorithm.

```
"""
Author: @suol0008@student.hv.se
"""
import os
import asyncio
import logging
import cv2 as cv
import numpy as np

logging.basicConfig(
    level = logging.DEBUG,
    format = "%(asctime)s %(levelname)s, %(message)s",
    datefmt = "%Y-%m-%d %H:%M:S",
    filename = "basics.logs"
)

logger = logging.getLogger(__name__)


async def processor(data):
    """ Image Normalization and Adaptive Thresholding. """
    try:
        norm_imag = np.zeros((data.shape[0], data.shape[1]))
        data_im = cv.cvtColor(data, cv.COLOR_BGR2GRAY)
        image = cv.normalize(data_im, norm_imag, 0, 255, cv.NORM_MINMAX)

    # SKEW CORRECTION
        coord = np.column_stack(np.where(image > 0))
        angle = cv.minAreaRect(coord)[-1]
        if angle < -45:
            angle = -(90 + angle)
        else:
            angle = -angle

        (h, w) = image.shape[:2]
        center = (w // 2, h // 2)

        #des_angle = 100    #To be used for different rotations only.
        M = cv.getRotationMatrix2D(center, angle, 1.0)
        rotated = cv.warpAffine(image, M, (w, h), flags = cv.INTER_CUBIC,
                                borderMode = cv.BORDER_REPLICATE)

        gray_img = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
        thresh = cv.adaptiveThreshold(gray_img, 255,
cv.ADAPTIVE_THRESH_GAUSSIAN_C,
                                        cv.THRESH_BINARY, 11, 2)

    except Exception as err:
        logger.error(f"{err}")
        if err:
            print (f"401 Error. \nhttp:goto basics.logs")
    return image, rotated

async def arrange(processed_dir, batch, metadata, data):
    try:
        if not os.path.exists(processed_dir):
            os.makedirs(processed_dir, exist_ok = True)
        batch_folder = batch.__str__()
        batch_path = os.path.join(processed_dir, batch_folder)
        if not os.path.exists(batch_path):
            os.makedirs(batch_path, exist_ok = True)
        output_file = os.path.join(batch_path, metadata)
        cv.imwrite(output_file, data)

    except Exception as err:
        logger.error(f"{err}")
        if err:
            print (f"401 Error. \nhttp:goto basics.logs")



async def process_image(batch, imageData):
    data_path = os.path.join(data_location, batch, imageData)
    metadata = imageData.__str__()
    try:
        image = cv.imread(data_path)
        gray_image, skewed = await processor(image)
        await asyncio.gather(
            arrange("GRAY-DATASETS", batch, metadata, gray_image),
            arrange("SKEWED-DATASETS", batch, metadata, skewed)
        )

    except Exception as err:
        logger.error(f"{err}")
        if err:
            print (f"401 Error. \nhttp:goto basics.logs")

data_location = "R-DATASETS"
async def main():
    for batch in os.listdir(data_location):
        for imageData in os.listdir(os.path.join(data_location, batch)):
            await process_image(batch, imageData)
```

```python

# Run the asyncio event loop
asyncio.run(main())
```

B: Support Vector Machine Algorithm.

```
#!/usr/bin/env python
# coding: utf-8
"""
Author: @suol0008@student.hv.se
"""

import os
import logging
import itertools
import joblib
import tensorflow as tf
import cv2 as cv
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from skimage.io import imread
from skimage.transform import resize
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.metrics import classification_report, confusion_matrix, roc_curve,
roc_auc_score, precision_recall_curve, average_precision_score


# `Defining plotting function with Matplotlib and Seaborn.`

def imagePlotter (image_arr):
    fig, axes = plt.subplots(1, 5, figsize = (15, 15))
    axes = axes.flatten()
    for image, ax in zip(image_arr, axes):
        ax.imshow(image)
        plt.tight_layout()
        plt.axis("off")
        plt.show()

# `Error Handling.`

logging.basicConfig(
        level = logging.DEBUG,
        format = "%(asctime)s %(levelname)s %(message)s",
        datefmt = "%Y-%m-%d %H:%M:%S",
        filename = "basic.logs"
    )

# Create logger
logger = logging.getLogger(__name__)
```

```
def imagePlotter1(image_arr):
    fig, axes = plt.subplots(1, len(image_arr), figsize=(15, 15))
    for i, image in enumerate(image_arr):
        sns.heatmap(image, ax=axes[i], cmap="gray", cbar=False)
        axes[i].axis("off")
    plt.tight_layout()
    plt.show()

# `Load the Dataset.`

data_dir = "../GRAY-DATASETS"

try:
    for batch in os.listdir(data_dir):
        for data in os.listdir(os.path.join(data_dir, batch)):
            data_path = os.path.join(data_dir, batch, data)
            try:
                image = cv.imread(data_path)
            except Exception as err:
                logger.error(f"Error occurred while reading image: {str(err)}")
except Exception as err:
    logger.error(f"{str(err)}")
    if err:
        print(f"401 \n http:Goto basic.logs")

data = tf.keras.utils.image_dataset_from_directory(data_dir)

# Split the datasets to training & validation sets.
train_set = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split = 0.1,
    subset = "training",
    seed = 100,

)

validation_set = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split = 0.1,
    subset = "validation",
    seed = 100
)

# Write training set and validation set to directory.
#
# Base on their classes.
```

```
train_directory = "../DATA/Train"       # Path to Datasets for training.

os.makedirs(train_directory, exist_ok = True)


image_classes = { }

try:
    for data, labels in train_set:
        for i in range(data.shape[0]):
            image = data[i].numpy()
            label = labels[i].numpy()

            str_label = label.__str__()

            class_directory = os.path.join(train_directory, str_label)
            os.makedirs(class_directory, exist_ok = True)

            try:
                if str_label not in image_classes:
                    image_classes[str_label] = [ ]

                image_classes[str_label].append(image)
            except Exception as err:
                logger.error(f"{err}")
except Exception as err:
    logger.error(f"{err}")
    if err:
        print(f"401 \n http:Goto basic.logs")


for str_label, image_list in image_classes.items():
    class_directory = os.path.join(train_directory, str_label)

    for j, image in enumerate(image_list):

        metadata = f"BN0520240{j}.jpg"
        image_path = os.path.join(class_directory, metadata)

        tf.keras.utils.save_img(image_path, image)

validation_directory = "../DATA/Validation"

os.makedirs(validation_directory, exist_ok = True)

image_classes = { }

try:
    for data, labels in validation_set:
        for i in range(data.shape[0]):
            image = data[i].numpy()
            label = labels[i].numpy()

            str_label = label.__str__()
            #print(str_label)

            class_directory = os.path.join(validation_directory, str_label)
            os.makedirs(class_directory, exist_ok = True)

            try:
                if str_label not in image_classes:
                    image_classes[str_label] = [ ]

                image_classes[str_label].append(image)
            except Exception as err:
                logger.error(f"{err}")
                if err:
                    print(f"401 \n http:Goto basic.logs")

except Exception as err:
    logger.error(f"{err}")
    if err:
        print(f"401 \n http:Goto basic.logs")

for str_label, image_list in image_classes.items():
    class_directory = os.path.join(validation_directory, str_label)

    for j, image in enumerate(image_list):

        k = j + 50
        metadata = f"BN0520240{k}.jpg"
        image_path = os.path.join(class_directory, metadata)

        tf.keras.utils.save_img(image_path, image)

# Number of Training Data.

def train_num (Dir):
    data_num = 0
    try:
        for root, dirs, data in os.walk(Dir):
            data_num += len(data)
    except Exception as err:
        logger.error(f"{err}")
        if err:
            print(f"401 \n http:Goto basic.logs")
    return data_num
```

```
train_dir = "../DATA/Train"
total_train_data = train_num(train_dir)
print(f"Total number of Training Set: {total_train_data}")


# Number of Validation Data

def val_num (Dir):
    data_num = 0
    try:
        for root, dirs, data in os.walk(Dir):
            data_num += len(data)
    except Exception as err:
        logger.error(f"{err}")
        if err:
            print(f"401 \n http:Goto basic.logs")
    return data_num

val_dir = "../DATA/Validation"
total_val_data = val_num(val_dir)
print(f"Total number of Validation Set: {total_val_data}")

# Initialize lists to hold images and labels

images = []
labels = []

# Loop through each class folder
try:
    for label, class_folder in enumerate(os.listdir(train_directory)):
        class_folder_path = os.path.join(train_directory, class_folder)
        for image_file in os.listdir(class_folder_path):
            image_path = os.path.join(class_folder_path, image_file)
            image = imread(image_path)
            image = resize(image, (224, 224))  # Resize image to a fixed size
            images.append(image)
            labels.append(label)
except Exception as err:
    logger.error(f"{err}")
    if err:
        print(f"401 \n http:Goto basic.logs")


# Convert lists to numpy arrays

X = np.array(images)
y = np.array(labels)

# Flatten the images for the SVM (each image is now a vector)
```

```
try:
    X_flat = X.reshape((X.shape[0], -1))
except Exception as err:
    logger.error(f"{err}")
    if err:
        print(f"401 \n http:Goto basic.logs")

# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X_flat, y, test_size = 0.2, random_state =
42)

# `Create a Pipeline with Standard scaling and SVM`
#
# * This is a linear SVM.
svm_clf = Pipeline([
    ("scaler", StandardScaler()),
    ("svm", SVC(kernel = 'linear', decision_function_shape = 'ovr', probability = True))
])

# Train the SVM model
svm_clf.fit(X_train, y_train)

# Predict on the test set
y_pred = svm_clf.predict(X_test)

y_pred_proba = svm_clf.predict_proba(X_test)

# `Save the Model`

#modelname = "Model.joblib"

modelname = "Model.h5"


joblib.dump(svm_clf, modelname)


# Evaluate the Model.

print("Classification report:")
print(classification_report(y_test, y_pred))
print("Confusion matrix:")
print(confusion_matrix(y_test, y_pred))

batch_names = ["BATCH03", "BATCH04", "BATCH05", "BATCH06", "BATCH07",
"BATCH08", "BATCH09", "BATCH10"]
```

```
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title="Confusion matrix",
                          cmap=plt.cm.Blues):
    """
    This function plots and prints the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """

    # Compute confusion matrix
    conf_matrix = confusion_matrix(y_true, y_pred)

    if normalize:
        cm = conf_matrix.astype('float') / conf_matrix.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        cm = conf_matrix
        print("Confusion matrix, without normalization")

    print(cm)

    plt.figure(figsize=(10, 10))
    plt.imshow(cm, interpolation="nearest", cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
    plt.show()

    print(classification_report(y_true, y_pred, target_names = classes))

plot_confusion_matrix(y_test, y_pred, classes = batch_names, normalize = True,
                      title = "Confusion matrix, with normalization")


# `Plot ROC curve for each class`
# Calculate ROC curve and AUC for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
n_classes = len(np.unique(y))  # Number of classes

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test == i, y_pred_proba[:, i])
    roc_auc[i] = roc_auc_score(y_test == i, y_pred_proba[:, i])


plt.figure(figsize=(10, 8))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], lw=2, label=f'Class {i} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()


# Plot Precision-Recall Curve.

def plot_precision_recall_curve(y_test, y_pred_proba):
    """
    Plot the Precision-Recall curve for each class.

    Parameters:
    y_test (numpy array): True labels for the test data.
    y_pred_proba (numpy array): Predicted probabilities for the test data.
    """
    precision = dict()
    recall = dict()
    average_precision = dict()
    try:
        n_classes = len(np.unique(y_test))  # Number of classes

        for i in range(n_classes):
            precision[i], recall[i], _ = precision_recall_curve(y_test == i, y_pred_proba[:, i])
            average_precision[i] = average_precision_score(y_test == i, y_pred_proba[:, i])

        # Plot Precision-Recall curve for each class
        plt.figure(figsize=(10, 8))
        for i in range(n_classes):
            plt.plot(recall[i], precision[i], lw=2, label=f'Class {i} (AP =
{average_precision[i]:.2f})')

        plt.xlabel('Recall')
        plt.ylabel('Precision')
        plt.title('Precision-Recall Curve')
        plt.legend(loc='lower left')
        plt.show()

    except Exception as err:
        logger.error(f"{err}")
        if err:
            print(f"401 \n http:Goto basic.logs")


plot_precision_recall_curve(y_test = y_test, y_pred_proba = y_pred_proba)

# Plot some test images with their Prediction labels.

plt.figure(figsize = (24, 24))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.imshow(X_test[i].reshape(224, 224, 3))
    plt.title(f"True: {y_test[i]}, Pred: {y_pred[i]}")
    plt.axis('off')
plt.show()

# Define the figure size
plt.figure(figsize=(30, 30))

# Loop through images from index 30 to 60 (inclusive)
for i, img_idx in enumerate(range(30, 61)):
    # Create a subplot grid with 7 rows and 5 columns
    plt.subplot(7, 5, i + 1)

    # Reshape and display the image
    plt.imshow(X_test[img_idx].reshape(224, 224, 3))

    # Set the title for each subplot
    plt.title(f"True: {y_test[img_idx]}, Pred: {y_pred[img_idx]}")

    # Hide the axes
    plt.axis('off')

# Show the plot
#plt.tight_layout()
plt.show()
```