Building a CNN model to identify defective jar lids

Project Overview

In modern manufacturing, automation plays a critical role in enhancing efficiency, consistency, and quality control. One key area where automation can add significant value is in defect detection during the production process. This project focuses on automating the inspection of jar lids by developing a Convolutional Neural Network (CNN) model to classify defective versus non-defective jar lids.

Manual inspection is not only time-consuming but also prone to human error and inconsistency—making it unsuitable for large-scale operations. By leveraging deep learning and computer vision techniques, we aim to streamline the quality control process, reduce inspection time, and increase the accuracy of defect identification.

Our CNN-based model is trained on labeled image data to learn distinguishing features between acceptable and defective jar lids, enabling real-time and scalable quality assessment on the production line.

This solution supports the broader goal of smart manufacturing and Industry 4.0 by reducing reliance on manual labor and ensuring higher product reliability.

Data overview

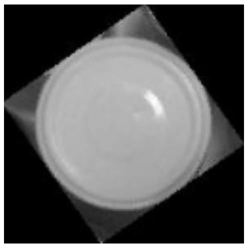
- Dataset: Labeled images of **damaged** vs. **intact** jar lids obtained from Kaggle.
- Images converted to **128×128 grayscale.**
- Original + Augmented images for better generalization.

Methodology

To improve the model's performance and generalizability, we began by augmenting the dataset using a variety of image transformation techniques. The goal was to artificially increase the number and diversity of training examples to help the model better learn relevant features. Our approach involved the following steps:

Data Augmentation

We applied several augmentation techniques to increase the size and variability of the dataset:
- Rotations: Images were rotated at angles of 90°, +35°, and -35° to simulate different orientations of jar lids.
- Flipping: Horizontal flip was used to expose the model to mirrored versions of the jar lids.
- Lighting Adjustments: Brightening and darkening were applied to help the model learn under varied lighting conditions.


Rotated

Flipped

Flipped

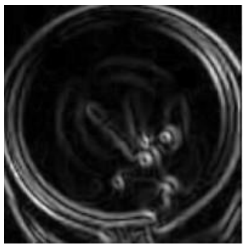
Darkened

Brightened

Image Manipulation for Feature Enhancement

To help the CNN learn meaningful and discriminative features, we incorporated preprocessing techniques that emphasize key visual patterns:

- X-ray Style Transformation: Inverted color channels to highlight internal contrasts.
- Edge Detection (Sobel Filter): Applied to capture contours and edges critical for detecting defects.
- Image Sharpening: Enhanced edges and textures, making defect features more distinguishable.


Edge detection (Sobel)

X-ray style inversion

Sharpening

These preprocessing and augmentation steps were essential in creating a robust and comprehensive training set, ensuring the model learns not just from the limited original dataset, but also from a wide range of visually altered variations.

Training Strategy

To optimize model performance, the following hyperparameters were systematically explored:

- Batch Size: Tested values of 128 and 256 to identify the optimal trade-off between speed and stability.
- Filter Combinations: Experimented with different numbers of filters in convolutional layers to enhance feature extraction.
- Learning Rate: Tuned to find the most effective value for stable and efficient convergence.
- Activation Functions: Compared several (e.g., ReLU, Leaky ReLU) to determine which yielded the best learning dynamics.

Training was conducted for up to 50 epochs, with Early Stopping based on validation accuracy to prevent overfitting and reduce unnecessary computation.

Model Architecture

Our final model is a deep Convolutional Neural Network (CNN) built using TensorFlow/Keras, specifically tailored for binary classification of defective vs. non-defective jar lids.

Input Layer: (128, 128, 1)

↓

Conv2D: 32 filters, 5x5, 'same' padding → initializer 'he_normal' → LeakyReLU(0.1) → MaxPooling2D (2x2)

Output: (64, 64, 32)

↓

Conv2D: 64 filters, 5x5, 'same' padding → initializer 'he_normal' → LeakyReLU(0.1) → MaxPooling2D (2x2)

Output: (32, 32, 64)

↓

Conv2D: 128 filters, 5x5, 'same' padding → initializer 'he_normal' → LeakyReLU(0.1) → MaxPooling2D (2x2)

Output: (16, 16, 128)

↓

Flatten

Output: 32768

↓

Dense Layer: 256 units, ReLU, Initializer 'he_normal'

↓

Dropout (rate=0.3)

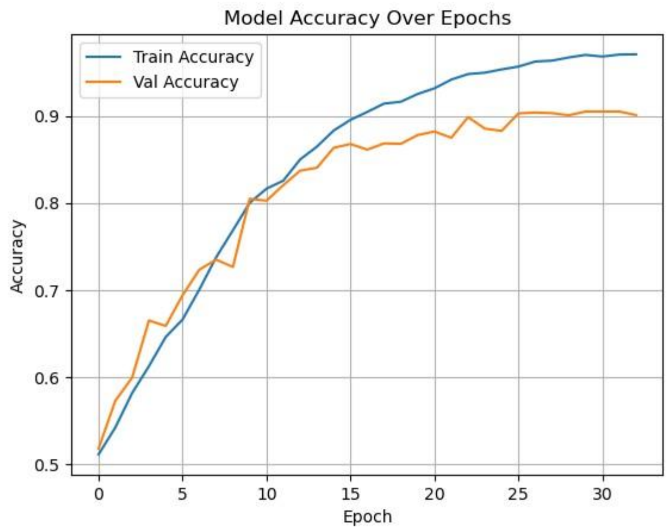
↓

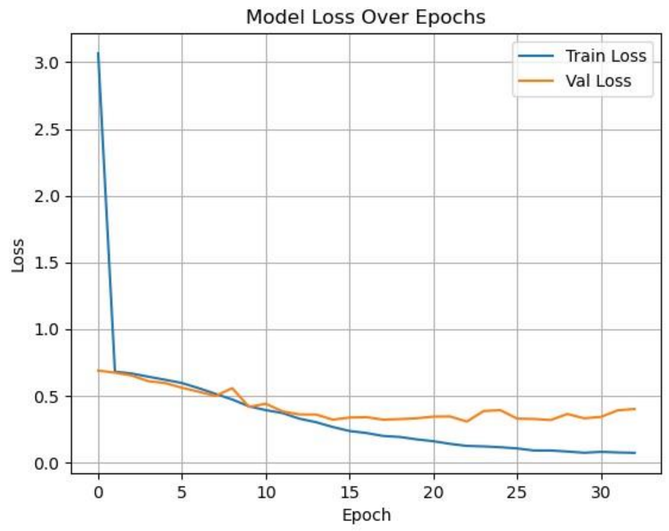
Dense Layer: 2 units (logits for binary classification)

Optimizer: Adam | **Loss:** Sparse Categorical Crossentropy

This architecture was selected after extensive experimentation with hyperparameters and activation functions. It effectively balances model complexity with generalization capability, achieving high accuracy on both validation and unseen test data.

Training Curves





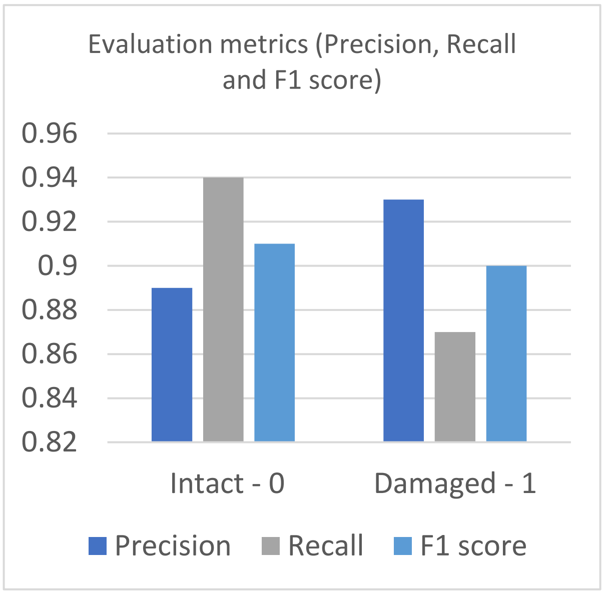
- ❑ **Training accuracy:** ↑ to ~96%
- ❑ **Validation accuracy:** Plateaus ~90%

- ❑ **Train Loss** shows a steep drop early on, decreasing consistently to ~0.05.
- ❑ **Validation Loss** decreases initially but plateaus and slightly fluctuates around ~0.35 after epoch 10–15.
- ❑ A widening gap emerges between train and validation loss in later epochs.

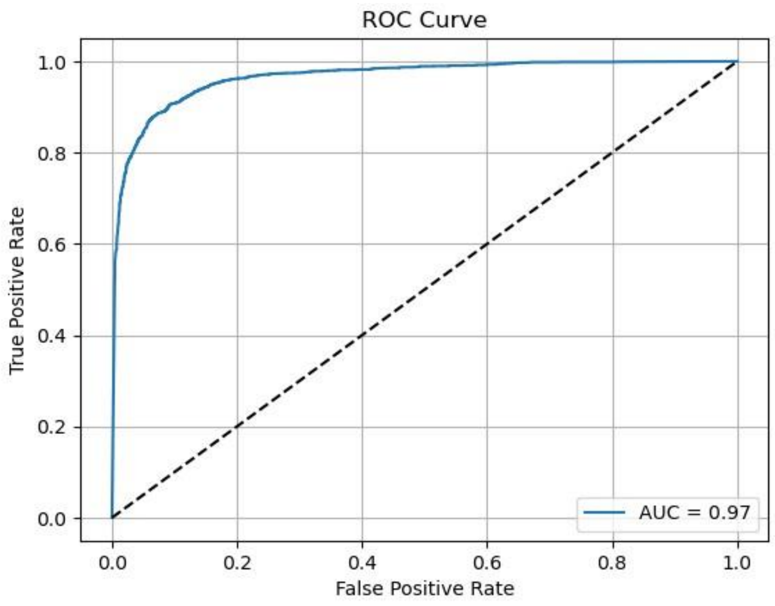
- The pattern for training curve is typical of overfitting, where the model performs significantly better on training data than on unseen validation data.
- However, the validation accuracy remains fairly high and stable, suggesting moderate rather than severe overfitting.
- The divergence in loss curves, with validation loss no longer decreasing while training loss continues to drop, reinforces the overfitting indication.
- The model continues to optimize on training data but fails to generalize better to validation data beyond a certain point.
- High final accuracy (~90%) on validation data indicates a well-performing model.

Evaluation Metrics (Precision, Recall and F1 score)

- ❑ The model is slightly better at identifying intact lids (high recall).
- ❑ It's more conservative when labeling damaged lids—prioritizing precision, possibly to avoid false alarms.
- ❑ The balanced F1 scores (~0.90–0.91) indicate the model performs very well on both classes, though improving recall on the "Damaged" class could further enhance defect detection reliability.



ROC Curve & AUC



- ❑ AUC = **0.97** → A high AUC implies good generalization to unseen data.
- ❑ Confirms robustness across thresholds.
- ❑ The steep initial rise and long plateau suggest the model confidently classifies many positives before significant false positives occur.

This evaluation confirms the CNN model is reliable, generalizes well, and is well-suited for deployment in an automated defect detection pipeline.

Future Improvements

To further enhance the model's performance and robustness, several future directions can be explored:

1. Leverage Pre-trained Models

- ➨ Utilizing transfer learning with well-established architectures such as VGG16, ResNet50 or EfficientNet. These models, pre-trained on large-scale datasets like ImageNet, can provide powerful feature extraction capabilities with minimal training data.

2. Deeper Architectures

- ➨ Experimenting with deeper CNNs that include additional convolutional and pooling layers. This may help the model capture more abstract, high-level features crucial for identifying subtle defects.

3. Mixed Activation Functions

- Explore using different activation functions (e.g., ReLU, Leaky ReLU, PReLU, Swish) across various layers instead of a single activation type. This can introduce non-linear diversity and improve model expressiveness.

4. Inception-based Architectures

- Inception modules, which combine multiple filter sizes within the same layer to capture features at different scales. This may be especially effective for capturing both fine and coarse structural details in jar lid images.

5. Data Augmentation Strategies

- Continuing expanding the dataset with advanced augmentation (e.g., elastic deformation, random occlusion, noise injection) to simulate real-world variability and further improve generalization.

By implementing these improvements, future iterations of the model can become more accurate, resilient, and suitable for real-time deployment in industrial settings.

Incorporating into an industrial automation pipeline

- **Model Serving via REST API**
  - Packaging the model using TensorFlow Serving, FastAPI, or Flask.
  - Creating a lightweight REST API that the factory control system can query with images for real-time results.
- **Real-time Inference Integration**
  - Deploying the model on edge devices (e.g., NVIDIA Jetson, Raspberry Pi with Coral TPU) near the production line. Creating a lightweight REST API that the factory control system can query with images for real-time results.
  - Use a real-time video feed or frame-based capture to inspect jar lids on a conveyor belt.
  - Automatically flag or sort defective items via mechanical actuators (e.g., robotic arms, air nozzles).
- **Integration with PLC Systems**
  - Connecting the model's output to a Programmable Logic Controller (PLC) using middleware like MQTT or OPC UA.
  - Automate reject mechanisms or stop production if defect thresholds are exceeded.