Capstone Two

Final Presentation

by Sheik Mohammad Boni Sadar

10-May, 2025

Defective Jar Lid Identification Using CNN

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.

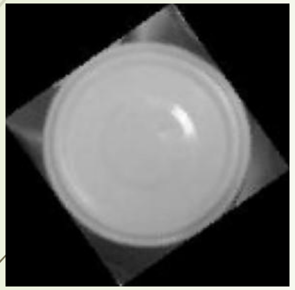
This project focuses on automating the inspection of jar lids by developing a **Convolutional Neural Network (CNN)** model to **classify defective versus non-defective jar lids.**

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.

Data augmentation

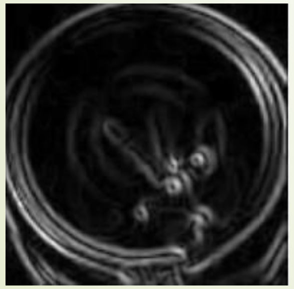
- ❑ **Rotations** (±35°, 90°)



- ❑ **Flipping** (horizontal)

- ❑ **Brightness adjustments**

- ❑ **Edge detection** (Sobel)



- ❑ **X-ray style inversion**

- ❑ **Sharpening**

Training Strategy

❑ **Hyperparameter tuning:**

- Find the optimal batch size (128/256).
- Selecting the optimal filter number combination.
- Finding most optimal learning rate.
- Finding most effective Activation Functions.

❑ **Epochs:** up to 50

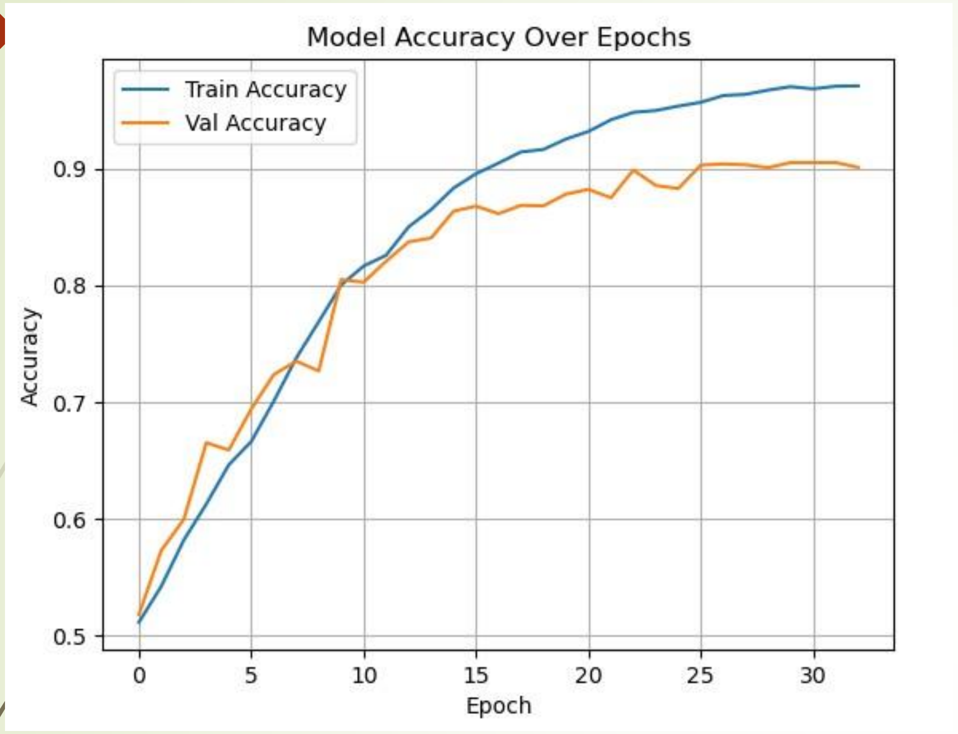
❑ **Early Stopping** on validation accuracy

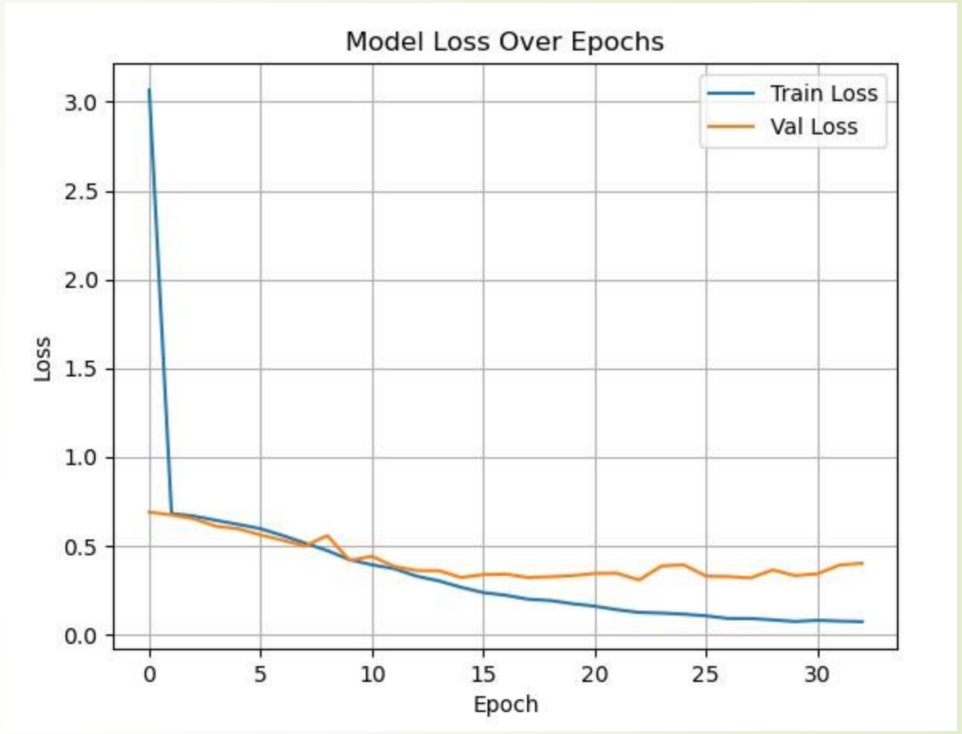
Final Model Architecture

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

Training Curves





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

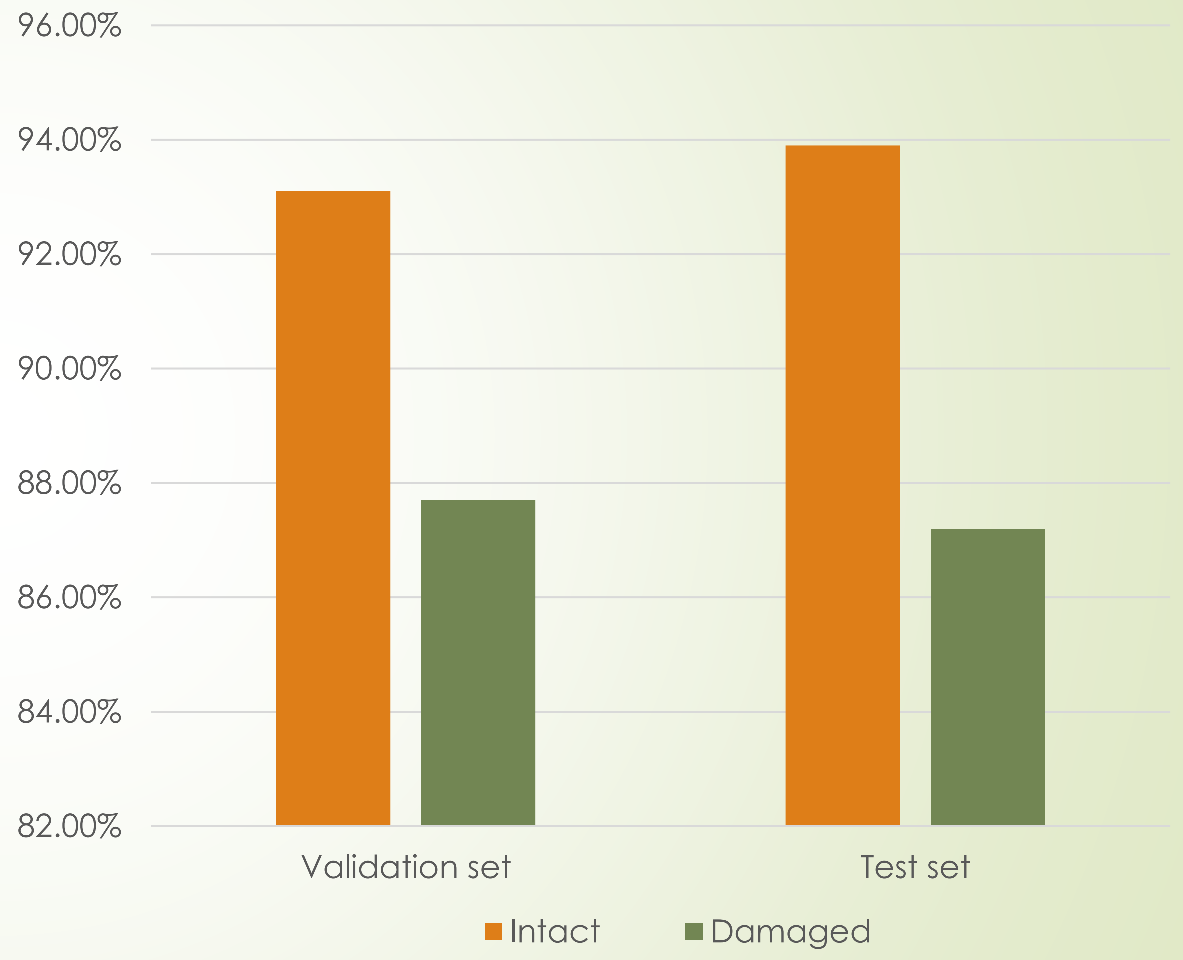
- ❑ **Train Loss** shows a steep drop early on, decreasing consistently to ~0.05.
- ❑ **Validation Loss** decreases initially but plateaus and slightly fluctuates around ~0.35 after epoch 10–15.
- ❑ A widening gap emerges between train and validation loss in later epochs.

Insights

- 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.

CNN Classification Performance Report

- Performance is consistent with both sets.
- Slightly better at classifying intact images.
- The accuracy drop for damaged may suggest more intra-class variation.



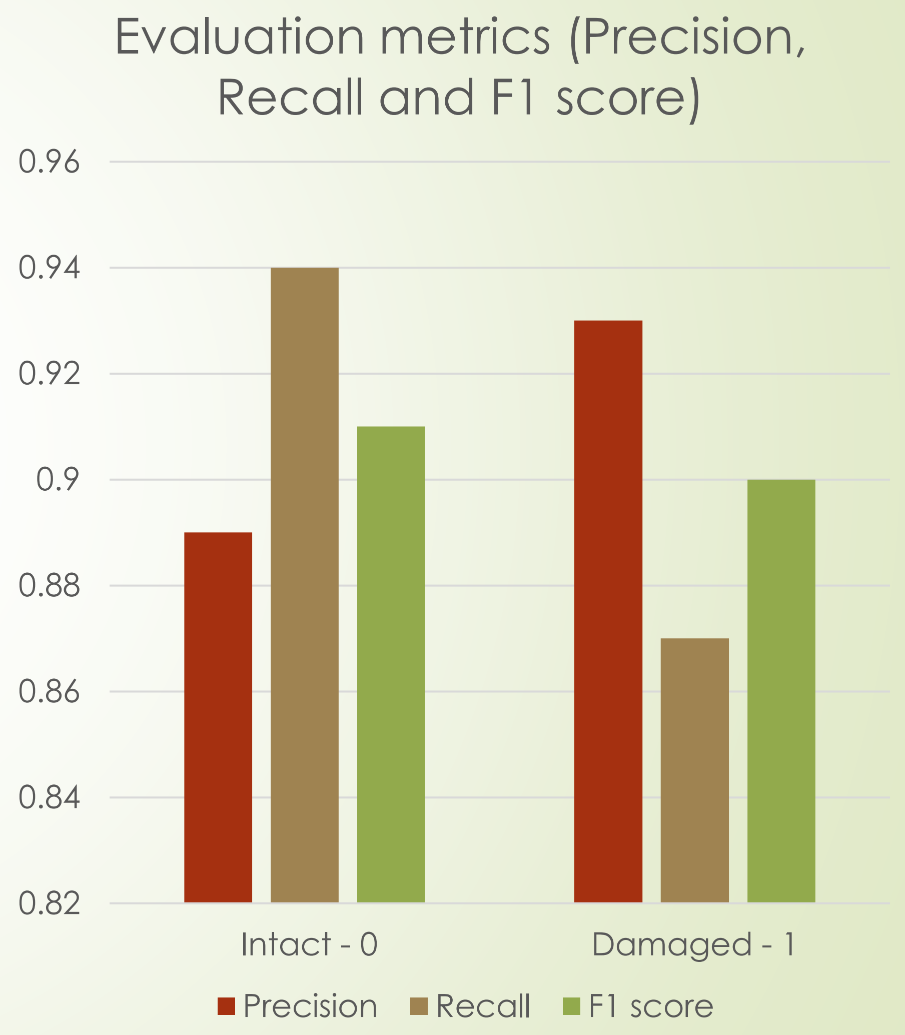
Confusion matrix plot - validation and test set

Precision, Recall and F1 score

- The model is **slightly better at identifying intact lids** (high recall).

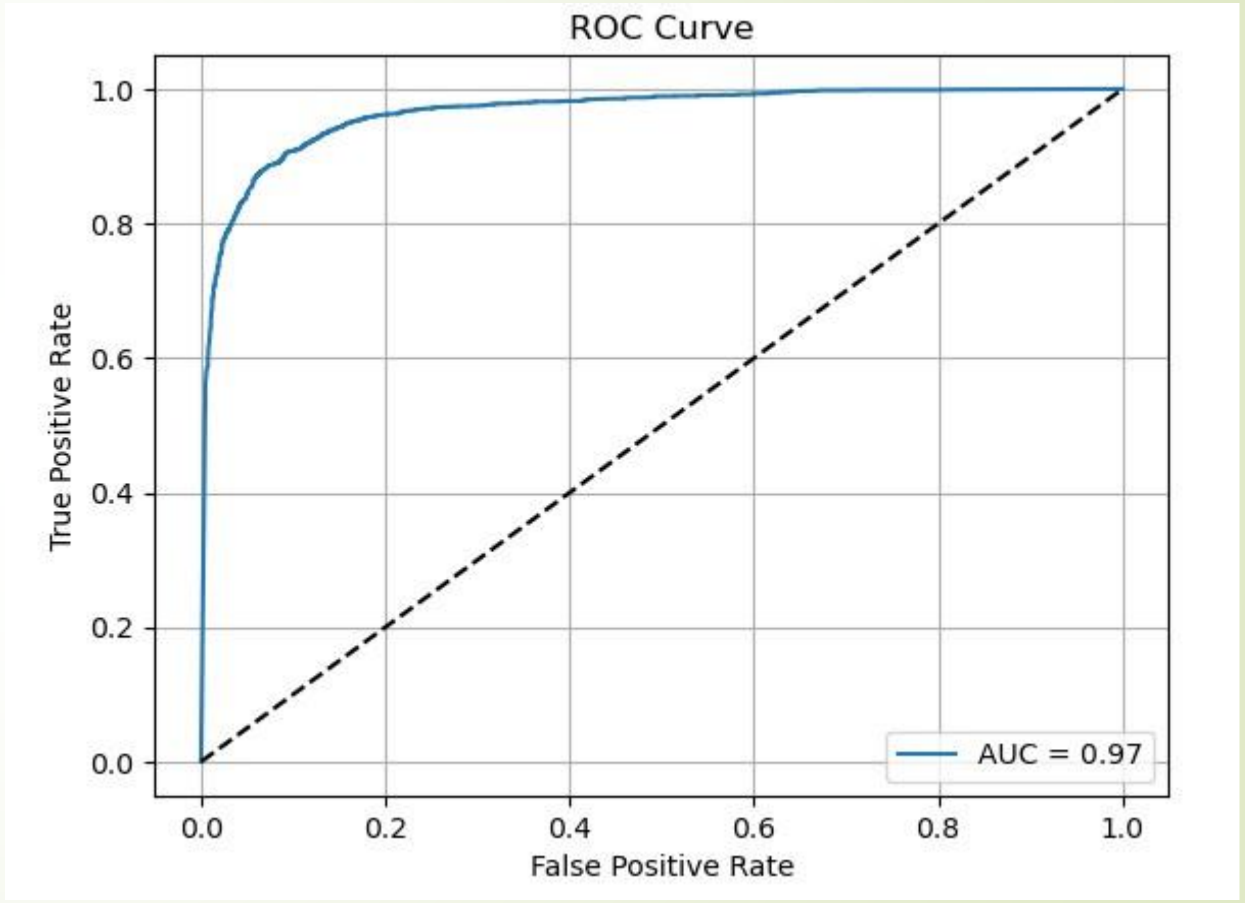
- It's **more conservative when labeling damaged lids**—prioritizing precision, possibly to avoid false alarms.

- 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.



Summary

- High performance across both validation and test sets (≈ 90% accuracy).
- AUC of 0.97 demonstrates strong capability to distinguish defective vs. intact jar lids.
- Slight overfitting after ~20 epochs, but not performance-breaking.

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

Future Improvements

- ❑ **Leverage Pre-trained Models**
  - Utilize transfer learning with well-established architectures such as VGG16, ResNet50 or EfficientNet.
- ❑ **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.
- ❑ **Data Augmentation Strategies**
  - Continue expanding the dataset with advanced augmentation (e.g., elastic deformation, random occlusion, noise injection) to simulate real-world variability and further improve generalization.

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.

Thank You All