Automated Bin Quantity Classification from Warehouse Bin Images using Deep Learning on AWS SageMaker

Capstone Project Proposal

Student: Brejesh Balakrishnan
Program: AWS Machine Learning Engineer Nanodegree (Capstone)
Date: 26th December 2025

1. **Domain Background:**

Accurate inventory tracking is a primary requirement for modern logistics and fulfillment operations. In such large scale warehouses, each storage bin contains variable quantities of products and incorrect estimates of these products can lead to stockouts, delays and operational inefficiency. Computer vision offers a scalable way to estimate the quantities of items in these bins from the images we get in the warehouse, thereby replacing manual counting.

This Project uses the Amazon Bin Image Dataset, a large scale dataset of warehouse bin images compiled for tasks like classification and quantity understanding. The dataset includes labels such as EXPECTED_QUANTITY and enables building models that can predict the count category from these images.

Modern Deep Learning approaches like CNN that have models pretrained on large scale images like ImageNet that are widely used for efficient transfer learning in Industrial classification tasks. ResNet style architectures are commonly used for such tasks due to its strong performance and stable optimization.

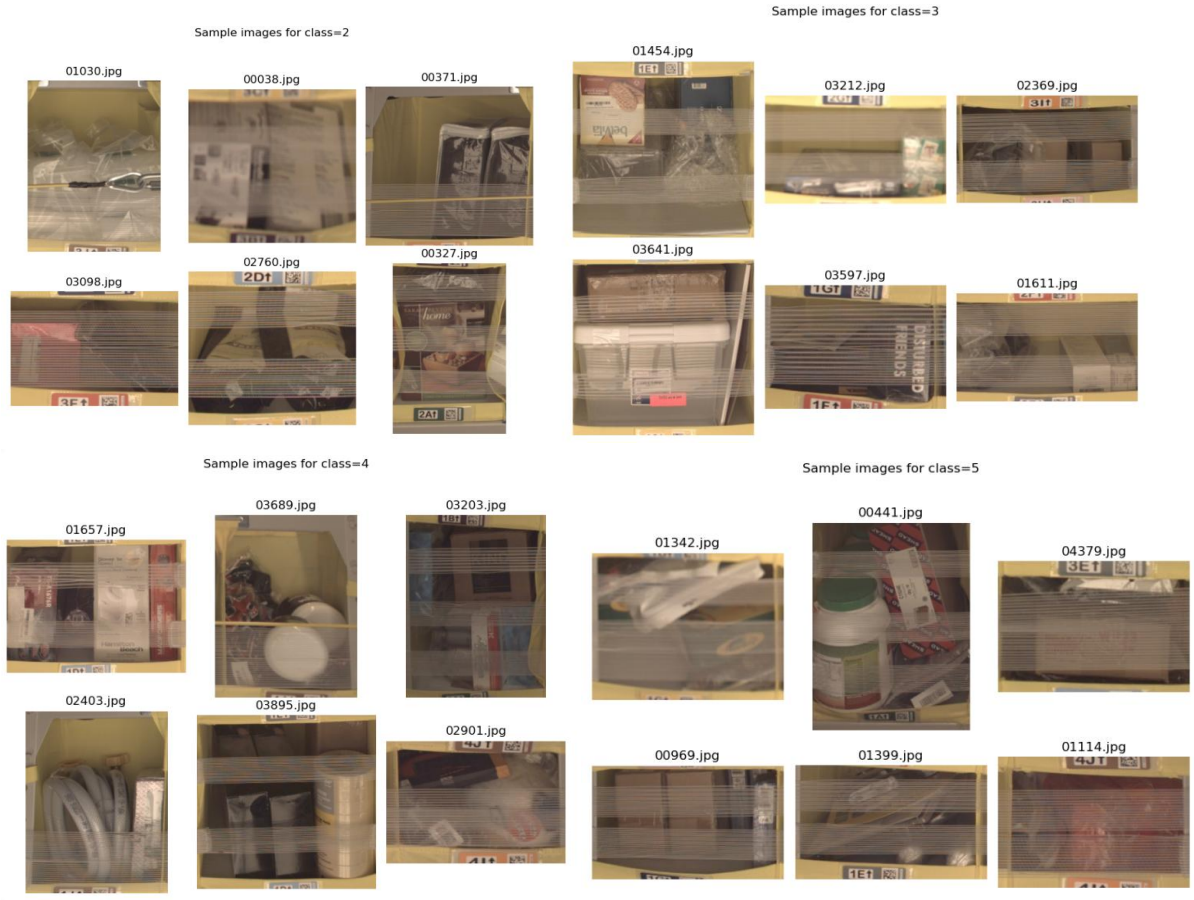

Figure 1: Example bin images for each class (2–5)

2. **Problem Statement:**

Goal: Given an image of a warehouse bin, predict the bin's expected quantity category (multi-class classification). For this project, the task is modified to be a 5-class classification problem

corresponding to a quantity labels {1,2,3,4,5} thereby using a subset of the dataset. Such a classification model is needed because manual bin checks are costly and slow at scale and an automated model can reduce this manual effort, improve accuracy and enable faster workflows.

Defined Structure:

- **Input:** bin image
- **Output:** predicted class label (1–5)
- Repeatable training/evaluation split with consistent preprocessing
- Evaluation with measurable metrics (macro-F1, accuracy)

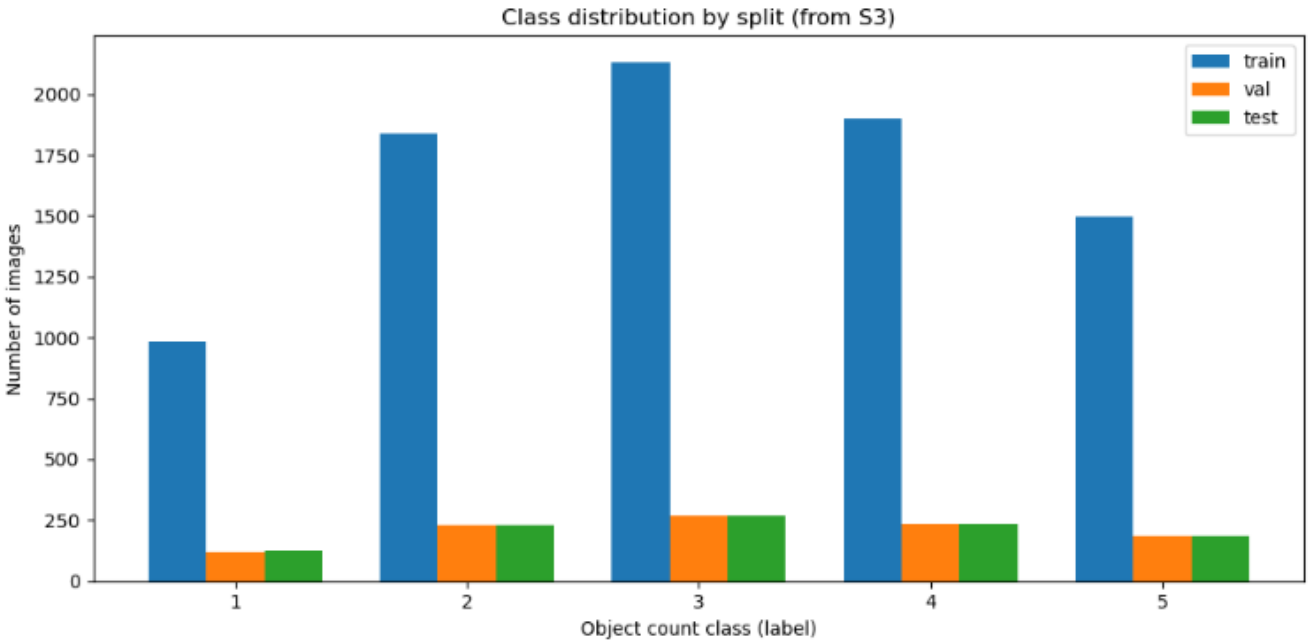

Figure 2: Class distribution of the split dataset

3. **Solution Statement:**

For this project, I propose a deep learning image classification pipeline that predicts expected bin quantity category from images.

Solution steps:

(i) Preprocess image (resize, normalization)

(ii) Train a transfer learning CNN (eg. ResNet50 pretrained on ImageNet) with a new classification head for 5 classes.

(iii) Improve performance using Hyperparameter tuning (learning rate, weight decay, batch size, epochs and freezing backbone) and Distributed training for faster scaling on multiple GPU instances.

(iv) Deploy the trained model to a real time sagemaker endpoint and expose inference through a Lambda function that takes in input of either image URL, S3 URI or API Gateway.

Measurable outputs:

- Improved macro-F1 vs Baseline
- Production ready endpoint that returns the predicted label and its associated confidence

4. **Datasets and Inputs:**

Dataset:

This project uses a subset of the Amazon Bin Image Dataset consisting of 10,441 images across 5 classes (labels 1–5). The dataset is structured into train/validation/test splits and loaded using ImageFolder-style directory layouts.

Data characteristics:

- Warehouse lighting and occlusion
- Visual clutter and multiple objects in a bin
- Potential class imbalance across quantities (1–5)

Data Information:

Train: 8348 images, 491.69 MB
Val: 1041 images, 61.61 MB
Test: 1049 images, 61.09 MB
Counts by class:
Train: {'1': 982, '2': 1839, '3': 2132, '4': 1898, '5': 1497}
Val: {'1': 122, '2': 229, '3': 266, '4': 237, '5': 187}
Test: {'1': 124, '2': 231, '3': 268, '4': 238, '5': 188}
Input during inference:

There will be two types of supported inference input patterns:

1. Raw image bytes used during direct invocation
2. Image URL/S3 URI used during Lambda function Invocation

5. **Benchmark Model:**

A benchmark model is required to establish a minimum performance baseline for objective comparison. After much research, it was established that there are two possible benchmarks that can be implemented for such a multi class problem.

Benchmark A (recommended for clarity):

- **Majority-class baseline** (always predict the most frequent class).
- Strength: simple, replicable, shows impact of imbalance.

Benchmark B (stronger baseline):

- **Transfer learning with frozen backbone**: pretrained ResNet50 feature extractor + linear classifier head (only head trained).
- Strength: a realistic “quick baseline” used in many applied projects.

Target comparison: The final model should exceed baseline macro-F1/accuracy by a meaningful margin.

6. **Evaluation Metrics:**

Because this is a multi-class classification task (with likely imbalance), the primary metric will be:

- **Macro F1-score** (treats all classes equally; robust to imbalance)
- Secondary: **Accuracy**, **Confusion Matrix** for class-wise error interpretation

Sample training result:

epoch=1 train_loss=1.549969 train_acc=0.260661 val_loss=1.541848 val_acc=0.268012 val_macro_f1=0.164870 elapsed_sec=29.92
epoch=2 train_loss=1.497035 train_acc=0.293843 val_loss=1.457621 val_acc=0.316042 val_macro_f1=0.264525 elapsed_sec=29.20
epoch=3 train_loss=1.489978 train_acc=0.295999 val_loss=1.454626 val_acc=0.298751 val_macro_f1=0.308525 elapsed_sec=28.81
epoch=4 train_loss=1.484403 train_acc=0.297676 val_loss=1.437040 val_acc=0.327570 val_macro_f1=0.321426 elapsed_sec=29.23
epoch=5 train_loss=1.475078 train_acc=0.298275 val_loss=1.439733 val_acc=0.312200 val_macro_f1=0.274868 elapsed_sec=28.73
TEST: loss=1.446686 acc=0.320305 macro_f1=0.316838
TEST: macro_precision=0.363682 macro_recall=0.310963
TEST: confusion_matrix=
[[42 59 18 4 1]
[14 87 105 20 5]
[9 75 119 50 15]
[5 36 106 65 26]
[4 27 81 53 23]]

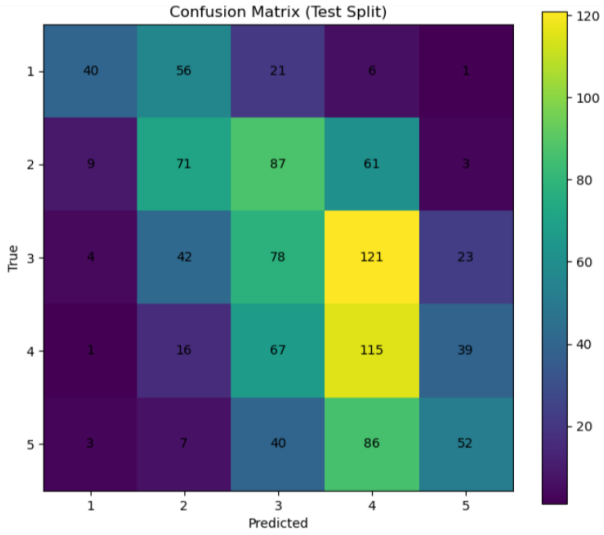

Figure 3: Confusion Matrix

7. **Project Design:**
    1. **Data Ingestion**
        - Download dataset from source / S3.
        - Verify folder structure and splits.
    2. **Exploratory Data Analysis (EDA)**
        - Class distribution, sample visualization, potential imbalance.
    3. **Baseline Training**
        - Train benchmark model.
        - Record baseline metrics (macro-F1, accuracy).
    4. **Model Development**
        - Transfer learning with ResNet50.
        - Augmentations (if needed).
        - Early stopping + checkpointing.
    5. **Hyperparameter Tuning**
        - Run HPO in SageMaker to identify best hyperparameter combination.
    6. **Distributed Training (DDP)**
        - Train on multi-instance GPU using SageMaker distributed data parallel support.
    7. **Deployment**
        - Deploy model to real-time endpoint.
        - Validate with smoke-test inference.
    8. **Lambda Integration**
        - Build Lambda function that accepts URL/S3/API Gateway event → calls endpoint → returns prediction.
    9. **Monitoring & Cleanup**
        - Capture endpoint logs/metrics.
        - Delete endpoints/models/configs to avoid costs.

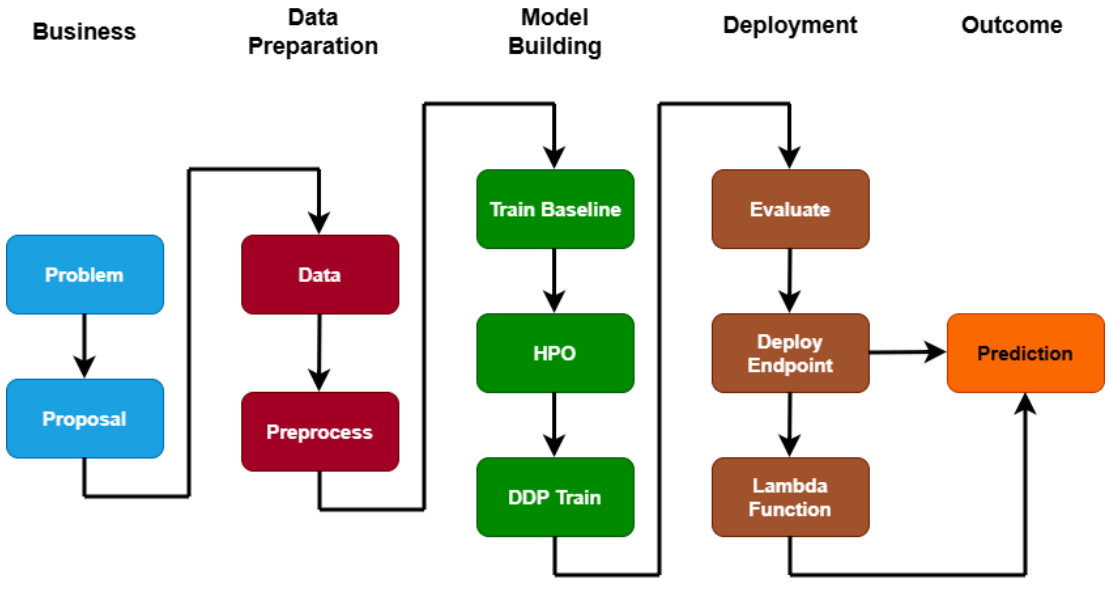

Figure 4: Workflow Diagram

8. **Presentation and Resources:**

Tools & Tech Stack

- **AWS SageMaker** (training, tuning, endpoint deployment)
- **PyTorch** (model training/inference)
- **S3** (dataset + model artifacts)
- **CloudWatch Logs** (endpoint & training logs)
- **AWS Lambda** (serverless invocation)

References:

- AWS Open Data: Amazon Bin Imagery Dataset: registry.opendata.aws
- He et al., "Deep Residual Learning for Image Recognition" (ResNet) arXiv
- Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge" arXiv
- AWS SageMaker Distributed Data Parallel documentation AWS Documentation