AWS ML ENGINEER NANODEGREE - CAPSTONE PROJECT REPORT

Inventory Monitoring via Bin Item Count Classification (Amazon Bin Images)

Author: Brejesh Balakrishnan
LinkedIn: https://www.linkedin.com/in/brejesh-balakrishnan-7855051b9/
Project Type: Computer Vision (Multi-class Image Classification)
Platform: Amazon SageMaker (Training + HPO + DDP + Real-time Endpoint + Lambda Integration)

1. PROJECT OVERVIEW

1.1 Domain Background

Visual recognition systems play a significant role in daily operations by automating tasks like counting objects, estimating quantities, and checking inventory states. This project aims to create an image classification model. The model will predict a specific "bin count" label (classes 1–5) based on an input image. It will then be deployed as a real-time service on Amazon SageMaker. This project demonstrates an end-to-end ML pipeline: **data preparation → baseline training → hyperparameter tuning → distributed training → real-time deployment → Lambda-triggered inference → cleanup**.

The solution uses SageMaker's real-time endpoints for HTTP inference and relates to AWS Lambda. This setup allows external systems to trigger the endpoint using S3 URLs, HTTP URLs, or API Gateway payload formats. For details on deployment mechanics like the model container lifecycle and endpoint health checks, as well as runtime behaviors for endpoint invocation, the implementation follows SageMaker's inference patterns.

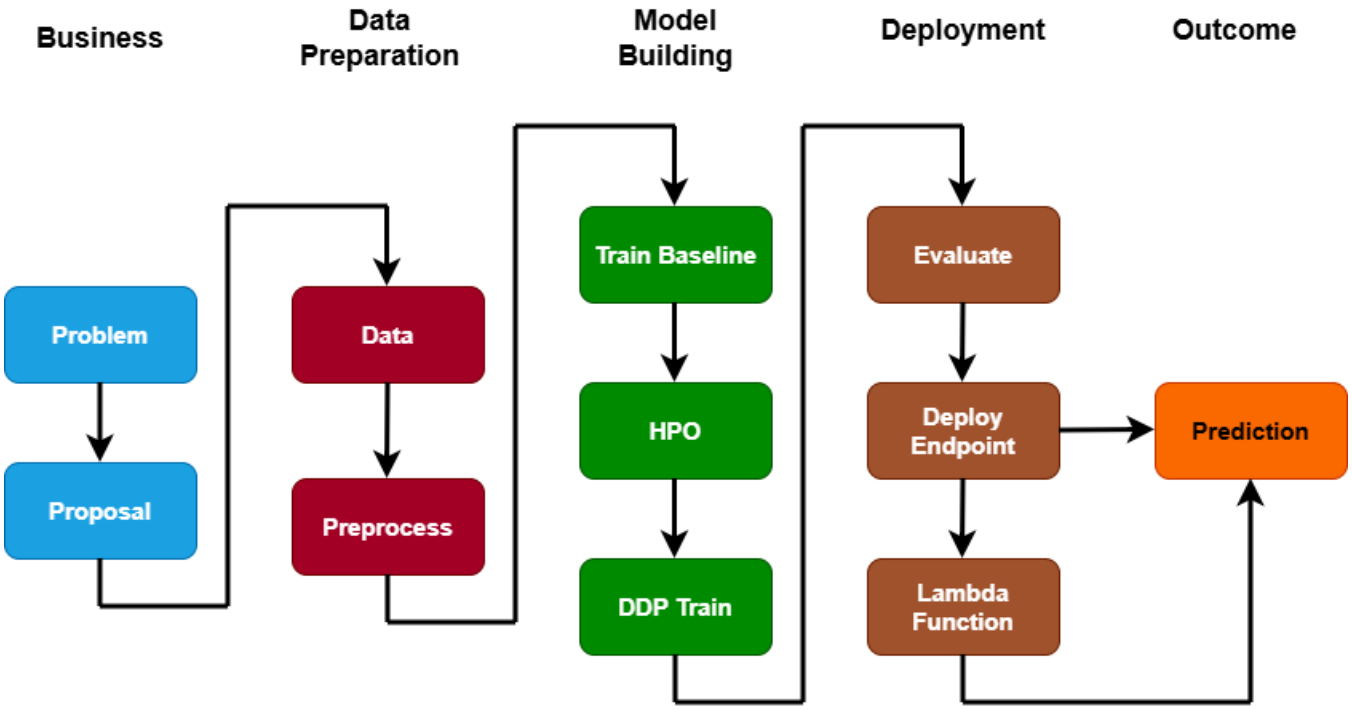

Figure 1: Project Workflow

1.2 Project Origin and Motivation

The motivation for this project is to show production-ready ML engineering skills using AWS. This involves training a deep learning model, improving it step by step, deploying it for real-time inference, and exposing inference through a Lambda integration. It also includes managing operational challenges like timeouts, instance selection, and endpoint cost management.

2. PROBLEM STATEMENT

2.1 PROBLEM DEFINITION

In distribution centers, it is crucial that bins have the expected number of items for reliable fulfillment. Manual counting is costly and does not scale. The aim is to create a model that can sort bin images into item-count categories (1 to 5), allowing for automated inventory checks and handling exceptions.

PROBLEM:
Given an input bin image x, predict the label $y \in \{1,2,3,4,5\}$,

where y represents the **EXPECTED_QUANTITY** (number of objects in the bin).
The dataset provides an EXPECTED_QUANTITY field in metadata as the ground-truth target.

This is a **supervised multi-class classification** problem:

- **Input:** image (JPG)
- **Output:** one of five classes (1–5)

2.2 WHY THIS IS MEASURABLE AND REPLICABLE

The task is measurable using standard classification metrics such as:

- Accuracy
- Macro F1-score (especially useful when class imbalance exists)
- Macro Precision / Macro Recall
- Confusion Matrix

The pipeline is replicable because:

- Dataset splits are deterministic and stored
- Training scripts are parameterized (epochs, lr, batch size, etc.)
- SageMaker training jobs and artifacts are versioned in S3

3. METRICS

3.1 CHOSEN METRICS

This project uses:

1. **Accuracy**

   Definition: How often your model is right, overall. It's the simplest measure—just the total number of correct predictions (both positive and negative) divided by the total number of data points tested. Accuracy is easy to interpret, but it can be misleading when class distributions are imbalanced.

2. **Macro F1-score (primary)**

   Definition: The harmonic mean of precision and recall, calculated independently for each class and then averaged. It treats every class equally, no matter how many examples it has. Macro F1 averages F1 across classes equally, making it a better indicator of performance when some classes are harder or underrepresented. This is why the project used **val_macro_f1** as a primary objective during tuning.

3. **Macro Precision / Macro Recall**
   Macro Precision

   - **Definition: The average of the precision for all individual classes.** Precision tells you, "Out of everything the model predicted as X, how many were actually X?"
   - **Focus:** It emphasizes the model's ability to **avoid false positives** (not incorrectly labeling a sample).

   Macro Recall

   - **Definition: The average of the recall (or sensitivity) for all individual classes.** Recall tells you, "Out of all the data points that *should have been* X, how many did the model actually find?"
   - **Focus:** It emphasizes the model's ability to **avoid false negatives** (not missing a sample that should have been labeled).

   These metrics are standard for multi-class classification evaluation and align with typical ML evaluation approaches.

4. ANALYSIS

4.1 DATA EXPLORATION

4.1.1 DATASET DESCRIPTION

The dataset contains images and metadata JSON files. Metadata includes the EXPECTED_QUANTITY field, which is used as the label. Due to compute/cost constraints (Udacity AWS Gateway environment), the project uses a **subset** of the Amazon Bin dataset:

Observed split sizes (from training logs):

- Total: **10,441 images** in the provided subset
- Train: **8348**
- Validation: **1041**
- Test: **1049**
- Classes: **['1', '2', '3', '4', '5']**

Class mapping:

- {'1': 0, '2': 1, '3': 2, '4': 3, '5': 4}

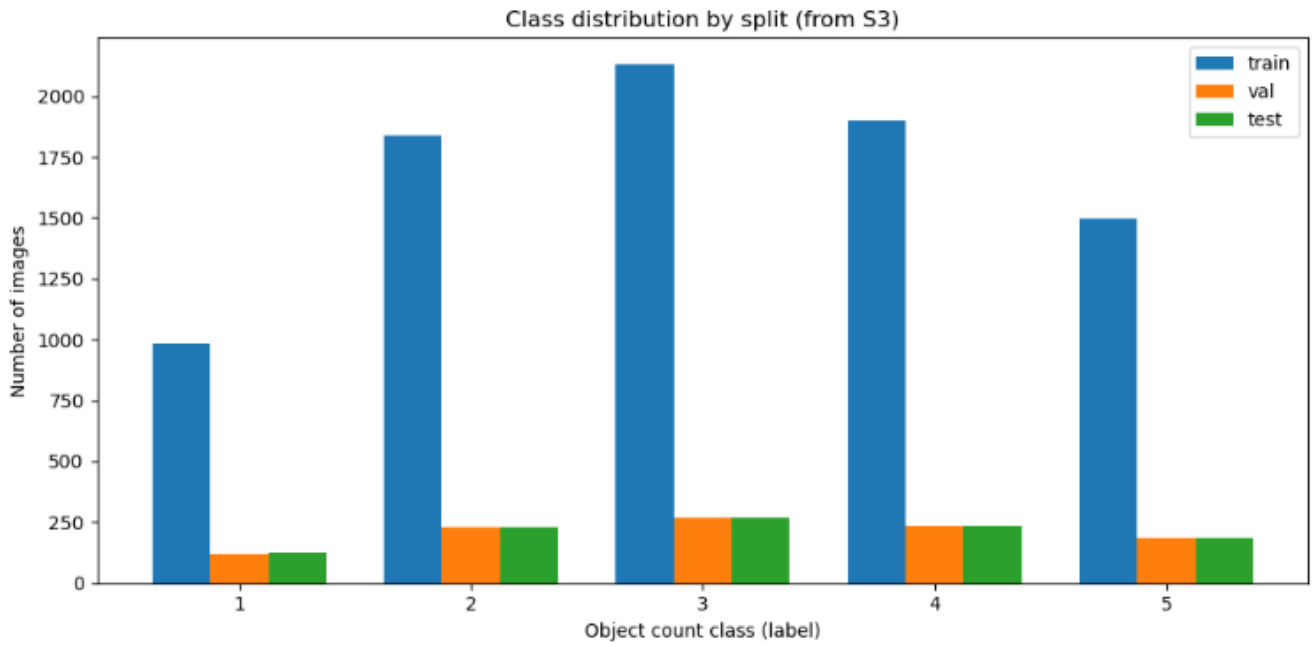

Figure 2: Class Distribution Bar chart

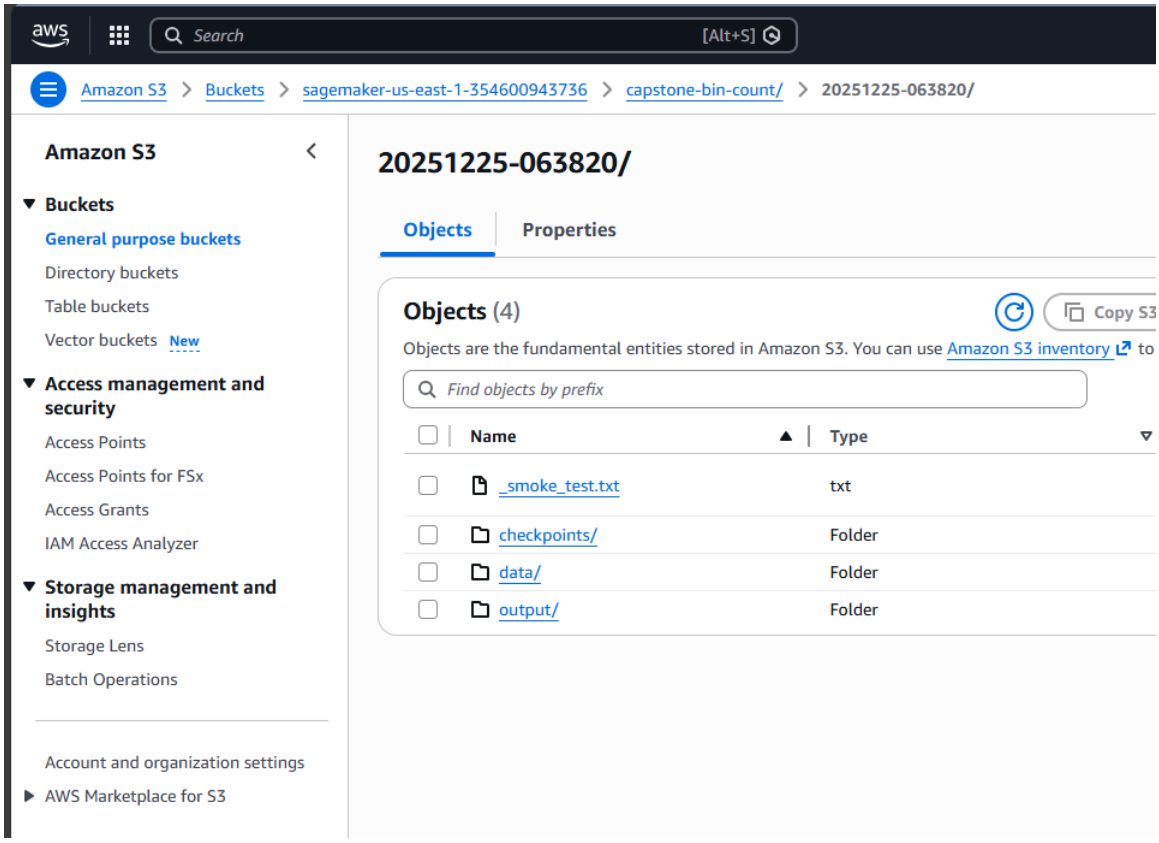

Figure 3: Data stored in S3 after split

4.1.2 DATA VISUALIZATION

Figure 4: Sample images grid by class (2 to 5),
Purpose: qualitative understanding of visual differences between labels

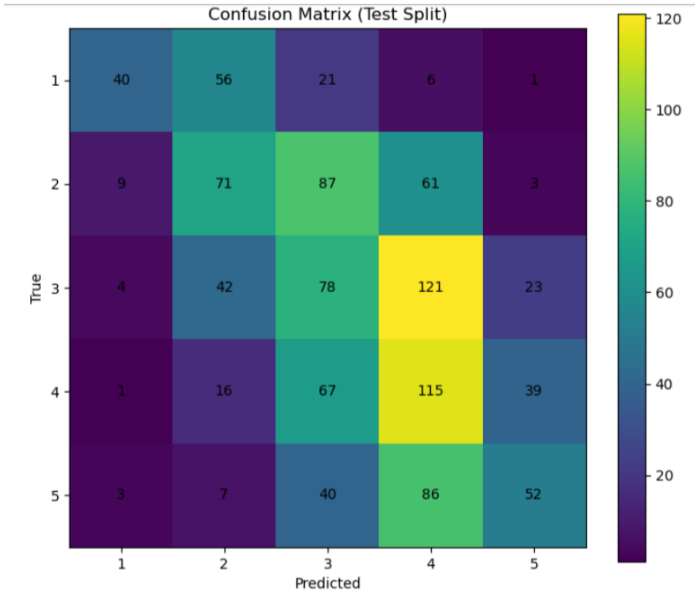

Figure 5: Confusion matrix heatmap (baseline and final),
Purpose: show where misclassifications concentrate

5. METHODOLOGY

5.1 DATA PREPROCESSING

5.1.1 TRANSFORMATIONS

Images were loaded via PyTorch data loaders and transformed to match ResNet input expectations (typical preprocessing includes resize/crop and normalization). The backbone used was **ResNet-50**, initialized with ImageNet weights. (Torchvision has migrated from pretrained=True to weights=... enums; the notebook logs show the related deprecation warnings.)

5.1.2 ABNORMALITIES / CHARACTERISTICS HANDLED

- The project uses **macro metrics** and confusion matrices to detect class-specific failures.
- Early stopping is applied to prevent wasted compute if validation loss stops improving.

5.2 IMPLEMENTATION

5.2.1 PROJECT ARTIFACTS & SCRIPTS

Notebooks:

- **sagemaker.ipynb:** End-to-end orchestration notebook: data prep, training, HPO, DDP training, deployment tests, Lambda integration, and cleanup.

Training scripts

- **train.py: Single-instance training entry point** (baseline + non-distributed training flow).
- **train1.py: Multi-instance / distributed training entry point** (DDP workflow for 2 instances).Includes distributed-safe logging + checkpointing logic appropriate for multi-host.

Inference scripts

- **inference.py**: Initial deployment inference handler (used for earlier endpoint testing).
- **final_inference.py:** Final deployment inference handler (used for the final endpoint + Lambda integration). Accepts image bytes and returns a consistent JSON schema (label, confidence, per-class probabilities).

Lambda script

- **Lambda.py:** Lambda handler and associated helper methods to get Request from user that contains either Image URL, S3 URI or API Gateway, downloads the image, feeds Image bytes to endpoint and returns the response. Includes code for handling multiple possible exceptions.

Model artifacts (S3)

- **Baseline training job output**
  s3://sagemaker-us-east-1-354600943736/capstone-bin-count/20251225-063820/output/capstone-bin-count-baseline-20251225-090040/output/model.tar.gz

- **Best HPO job output**
  s3://sagemaker-us-east-1-354600943736/capstone-bin-count/20251225-063820/output/cbc-hpo-251225095928-003-6006652c/output/model.tar.gz
- **DDP training output**
  s3://sagemaker-us-east-1-354600943736/cbc-ddp-251225173021/output/model.tar.gz

Deployment + Integration

- **SageMaker real-time endpoint:** Created for smoke test and then removed after validation.
- **AWS Lambda function:** Invokes SageMaker Runtime **InvokeEndpoint** and returns model predictions as JSON to upstream callers (HTTP/S3/API Gateway).

Cleanup instructions (what gets deleted)

- SageMaker **Endpoint** (stops hosting & releases endpoint resources).
- SageMaker **EndpointConfig** (deleted explicitly).
- SageMaker **Model** objects (deleted defensively if created during deployment repackaging).

5.2.2 BASELINE TRAINING (SINGLE INSTANCE)

Entry point: train.py
A baseline ResNet-50 is used as a pre-trained feature extractor, fine-tuned for 5-class classification, was trained on SageMaker, producing a model artifact and evaluation metrics. The PyTorch torchvision ResNet-50 pretrained weights provide a strong initialization for visual feature extraction.

Training on SageMaker

The project uses SageMaker training jobs and produces model artifacts stored in S3. The training script includes:

- cross entropy loss
- Adam optimizer (with tuned learning rate)
- early stopping logic
- saving metrics + model weights to /opt/ml/model
- macro F1 computed from confusion matrix
- robust distributed metric aggregation when using DDP

Baseline test results:

- **Test Accuracy: 0.320305**
- **Test Macro F1: 0.316838**
- **Macro Precision: 0.363682**
- **Macro Recall: 0.310963**

Baseline confusion matrix (Test):

[[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]]

This matrix shows strong confusion between adjacent mid/high classes (e.g., 3↔4↔5), indicating the model struggles to separate visually similar count categories.

```
epoch=1 train_loss=1.549969 train_acc=0.260661 val_loss=1.541848 val_acc=0.268012 val_macro_f1=0.164870 elapsed_se
c=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_se
c=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_se
c=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_se
c=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_se
c=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 6: Baseline Model training metrics

5.3 REFINEMENT

5.3.1 HYPERPARAMETER OPTIMIZATION (HPO)

An HPO job was launched with the objective metric: **val_macro_f1** and this job focuses on learning rate, weight size, freeze backbone, early stop patience, batch size, and epochs.

Hyperparameter	**Range**	**Why We Tune It**
Learning Rate	Continuous Logarithmic – 1e-5 to 3e-3	**Controls speed of adaptation.** Needs to be **very small** to gently adjust the pre-trained weights without wiping out the knowledge (catastrophic forgetting). Logarithmic search is necessary because small changes matter vastly more than large ones.

Hyperparameter	Range	Why We Tune It
Weight Decay	Continuous Logarithmic – 1e-7 to 1e-2	**Fights overfitting (regularization).** Penalizes large weights to encourage the model to be simpler and generalize better to the 5 new classes. We search this widely to see if minimal or aggressive regularization is needed for our specific dataset size.
Batch Size	Categorical - 16, 32 or 64	**Balances stability vs. noise.** Smaller batches (16) introduce more gradient noise, sometimes finding better generalizable solutions. Larger batches (64) offer faster, more stable training but can overfit.
Freeze Backbone	Categorical - "true" or "false"	**Determines the training strategy.** True (Freeze) is best for small, similar datasets, only training the new final layers. False (Full Fine-tune) is better for large or diverse datasets, allowing the whole network to adjust slightly.
Early Stop Patience	Continuous Integer – 2 to 5	**Prevents wasted compute and overfitting.** How many epochs to wait for improvement before stopping. A lower value (2) is faster but risky; a higher value (5) is safer but slower.

	TrainingJobName	FinalObjectiveValue
5	cbc-hpo-251225095928-003-6006652c	0.387539
4	cbc-hpo-251225095928-004-268fe9c3	0.381017
7	cbc-hpo-251225095928-001-c6611cb8	0.320202
6	cbc-hpo-251225095928-002-d7e448aa	0.301742
2	cbc-hpo-251225095928-006-87e0f4d5	0.284589
3	cbc-hpo-251225095928-005-675cf88d	0.230475
0	cbc-hpo-251225095928-008-9b4cd195	0.190717
1	cbc-hpo-251225095928-007-bd921a65	0.137914

✅ Best objective (val_macro_f1): 0.38753899931907654
✅ Best job: cbc-hpo-251225095928-003-6006652c

Figure 7: HyperParameter Training jobs

Best HPO objective value (FinalObjectiveValue): 0.387539
Best training job: cbc-hpo-251225095928-003-6006652c

Best hyperparameters (key ones):

- batch_size = **64**
- epochs = **12**
- learning_rate = **2.0128517889116948e-05**
- weight_decay = **1.3422792576662194e-06**
- early_stop_patience = **5**
- freeze_backbone = **false**
- num_workers = **2**

This step improved the objective metric compared to the earlier baseline validation behavior by systematically searching a better configuration rather than using fixed defaults.

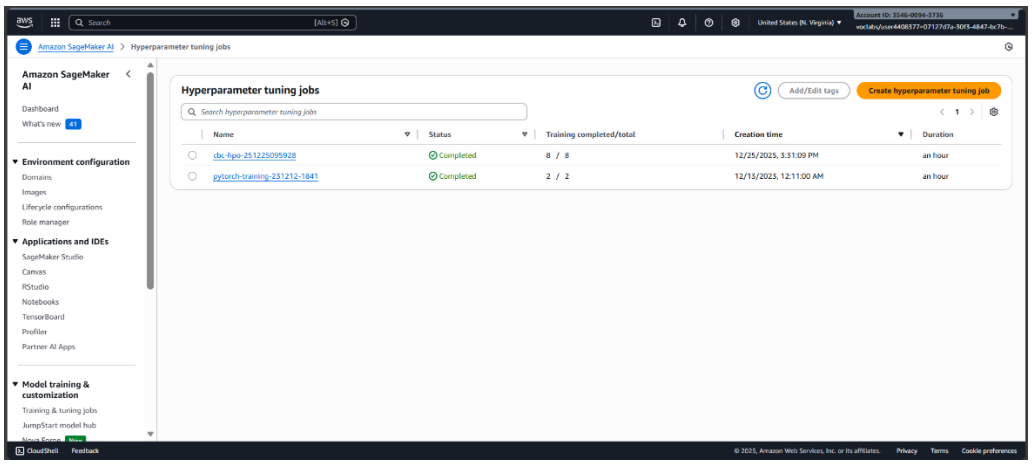

Figure 8: Completed Hyperparameter tuning

5.3.2 DISTRIBUTED TRAINING (DDP)

To reduce training time and support scalability, the project completes a DDP training milestone using multiple GPU instances (e.g., ml.g4dn.xlarge, instance count = 2). SageMaker supports distributed training patterns and distributed data parallel strategies for deep learning workloads.

Entry point: train1.py
The project then scaled training using **SageMaker Distributed Data Parallel**, which coordinates multiple instances for data-parallel training.

Completed DDP training job details:

- Status: **Completed**
- InstanceType: **ml.g4dn.xlarge**
- InstanceCount: **2**
- ModelArtifacts:
    - **s3://sagemaker-us-east-1-354600943736/cbc-ddp-251225173021/output/model.tar.gz**

During DDP development, practical operational issues were encountered (e.g., multi-host initialization, argument parsing pitfalls, and occasional host SSH connectivity retries visible in logs). Addressing these issues is part of real-world distributed ML engineering on managed platforms.

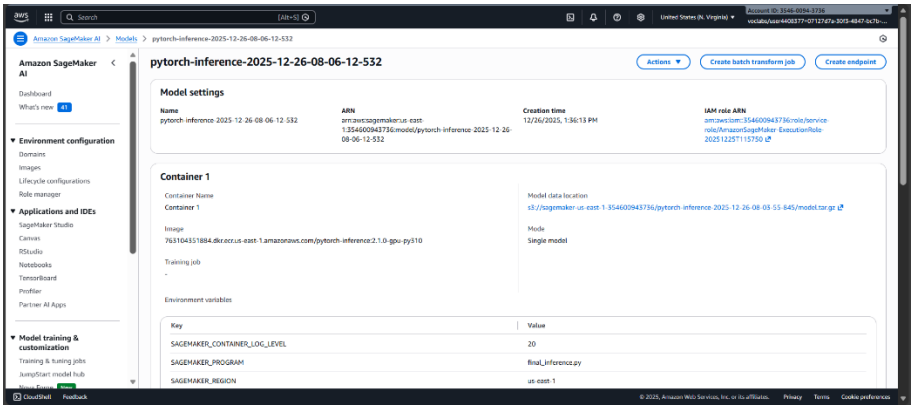

Figure 9: Final Model

5.3.3 MANAGED SPOT TRAINING (COST OPTIMIZATION ATTEMPT)

In addition to Hyperparameter Tuning (HPO) and Distributed Training (DDP), **Managed Spot Training** was also attempted to reduce training cost. Managed Spot Training uses **EC2 Spot instances** and can reduce training cost significantly (often *up to ~90%* vs on-demand), but jobs can be **delayed or interrupted** depending on capacity.

A Spot training job was run using SageMaker's managed spot capability and compared it against our earlier on-demand baseline run.

Baseline (On-demand)

- Instance: ml.g4dn.xlarge (GPU)
- TrainingTimeInSeconds: **482 sec**
- BillableTimeInSeconds: **482 sec**
- Job: capstone-bin-count-baseline-20251225-090040

Spot Experiment (Managed Spot Training)

- Instance: ml.m5.2xlarge (CPU)
- TrainingTimeInSeconds: **5445 sec**
- BillableTimeInSeconds: **1896 sec**
- Managed Spot Training savings shown by SageMaker: **~65.2%**
- Job: cbc-spot-251225133313
- Job ended with status **Stopped** due to **MaxRuntimeExceeded**

Run Type	Job Name	Instance Type	Training Time (sec)	Billable Time (sec)	Outcome
On-demand baseline	capstone-bin-count-baseline-20251225-090040	ml.g4dn.xlarge	482	482	Completed
Managed Spot (experiment)	cbc-spot-251225133313	ml.m5.2xlarge	5445	1896	Stopped (MaxRuntimeExceeded)

Reason for Bad Spot comparison data:

1. **CPU vs GPU mismatch (primary reason for long runtime):** The Spot run used a **CPU instance** (ml.m5.2xlarge) while the baseline used a **GPU instance** (ml.g4dn.xlarge). For CNNs (ResNet50), CPU training is dramatically slower than GPU training, so wall-clock training time increased substantially.
2. **Training job hit the MaxRuntime limit:** SageMaker stopped the Spot job because it exceeded the configured MaxRuntimeInSeconds ("MaxRuntimeExceeded"). This means the job did not finish all intended epochs.

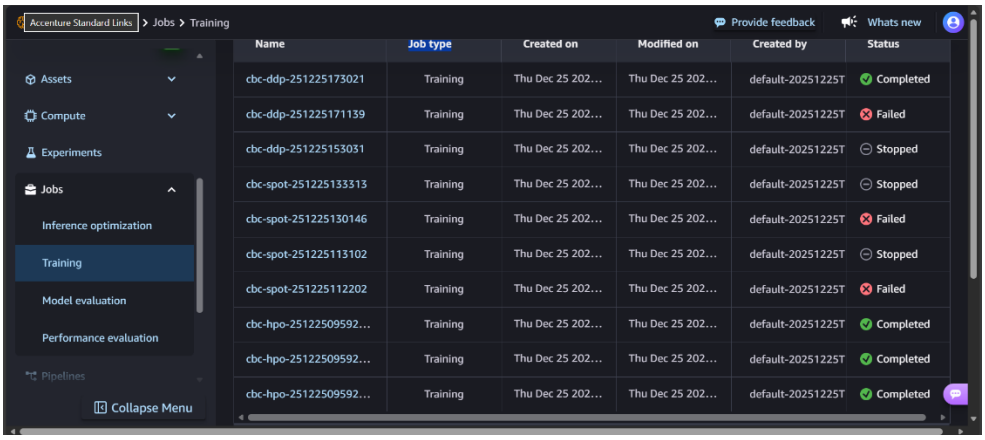

Figure 10: Spot training jobs "Stopped" rather than "Failed"

6. RESULTS

6.1 Model Evaluation and Validation

6.1.1 Real-time endpoint smoke test (DDP-trained model)

The DDP-trained model was deployed as a **real-time SageMaker endpoint** and tested with a random image from the local test split.

Deployment artifact used:

- s3://sagemaker-us-east-1-354600943736/cbc-ddp-251225173021/output/model.tar.gz

Endpoint smoke test output:

- Selected image: local_data/splits/test/3/07590.jpg
- True label: **3**
- Predicted label: **4**
- Confidence: **0.3136216998100281**
- Probabilities: [0.04406448, 0.12882562, 0.23083554, 0.31362170, 0.28265268]
- Class labels: ['1','2','3','4','5']

This result indicates the model is still somewhat uncertain between neighboring classes (3/4/5), which matches the confusion trends observed earlier. However, the deployment validates that the full training→deployment pipeline works reliably and returns structured predictions.

6.2 JUSTIFICATION

```
=== ✅ Milestone 10 Evaluation Metrics ===
test_accuracy: 0.3393708293612965
test_macro_precision: 0.41481308857605015
test_macro_recall: 0.3361547501263552
test_macro_f1: 0.3524913306918365
num_test_samples: 1049
class_labels: ['1', '2', '3', '4', '5']

=== ✅ Classification Report (per class) ===
              precision    recall  f1-score   support

           1       0.70      0.32      0.44       124
           2       0.37      0.31      0.34       231
           3       0.27      0.29      0.28       268
           4       0.30      0.48      0.37       238
           5       0.44      0.28      0.34       188

    accuracy                           0.34      1049
   macro avg       0.41      0.34      0.35      1049
weighted avg       0.38      0.34      0.34      1049
```

Figure 11: Final Model Metrics

What improved the pipeline

- **HPO** improved the objective score by selecting better training hyperparameters rather than relying on defaults.
- **DDP** improved scalability and reduced wall-clock time potential for larger workloads and future dataset growth.

What still limits real-world usability (and why accuracy is not yet "production-grade")

Even after systematic refinement, bin-count classification remains challenging because:

- Neighboring classes can be visually similar
- Images may vary in lighting, angles, occlusion, and clutter
- Potential dataset imbalance can make minority classes harder (macro metrics help reveal this)

6.3 Result Compilation

The following table contains the compilation of all the models built, description of what was done and key output metrics/Evidences to prove the same.

Stage / Milestone	What was done	Key Output Metrics / Evidence
Baseline (Single-instance training + evaluation)	Trained a ResNet50-based classifier and evaluated on the held-out test set	• **Test Acc:** 0.3203 • **Test Macro-F1:** 0.3168 • **Test Macro-Precision:** 0.3637 • **Test Macro-Recall:** 0.3110 • **Test Loss:** 1.4467
HPO (Hyperparameter Optimization)	Ran SageMaker HPO and selected the best job by objective metric	• **Best objective (val_macro_f1):** 0.387539 • **Best training job: cbc-hpo-251225095928-003-6006652c**
DDP Training (Multi-instance distributed)	Trained using **2 instances** (Distributed Data Parallel workflow) and produced deployable artifacts	• **Status:** Completed • **InstanceType:** ml.g4dn.xlarge • **InstanceCount:** 2 • **ModelArtifacts: s3://sagemaker-us-east-1-354600943736/cbc-ddp-251225173021/output/model.tar.gz**
Real-time Deployment + Smoke Test	Deployed endpoint, invoked with a sample image, validated JSON response	• Endpoint reached **InService**, inference returned structured output: **predicted_label**, **confidence**, and full class probabilities
Lambda Integration	Created Lambda function to invoke SageMaker endpoint with **HTTP URL / S3 URL / API Gateway style** inputs	• Verified end-to-end: Lambda → SageMaker Runtime **InvokeEndpoint** → Response JSON returned successfully.

Stage / Milestone	What was done	Key Output Metrics / Evidence
Cleanup Confirmation	Deleted deployed endpoint resources after testing	• Endpoint deleted + EndpointConfig deleted + model resources deleted (defensive cleanup). • Endpoint deletion releases the endpoint's deployed hosting resources; endpoint config must be deleted separately.

7. DEPLOYMENT ARCHITECTURE

7.1 SageMaker Endpoint + Lambda Integration

After validating real-time endpoint inference, the project added an AWS Lambda function that triggers the endpoint using **SageMaker Runtime invoke_endpoint** calls. This allows inference requests from:

- HTTP image URLs
- S3 URLs
- API Gateway event payloads

The Lambda uses the SageMaker Runtime client and forwards payloads to the endpoint, returning JSON predictions. The call pattern is based on the boto3 SageMaker Runtime API.

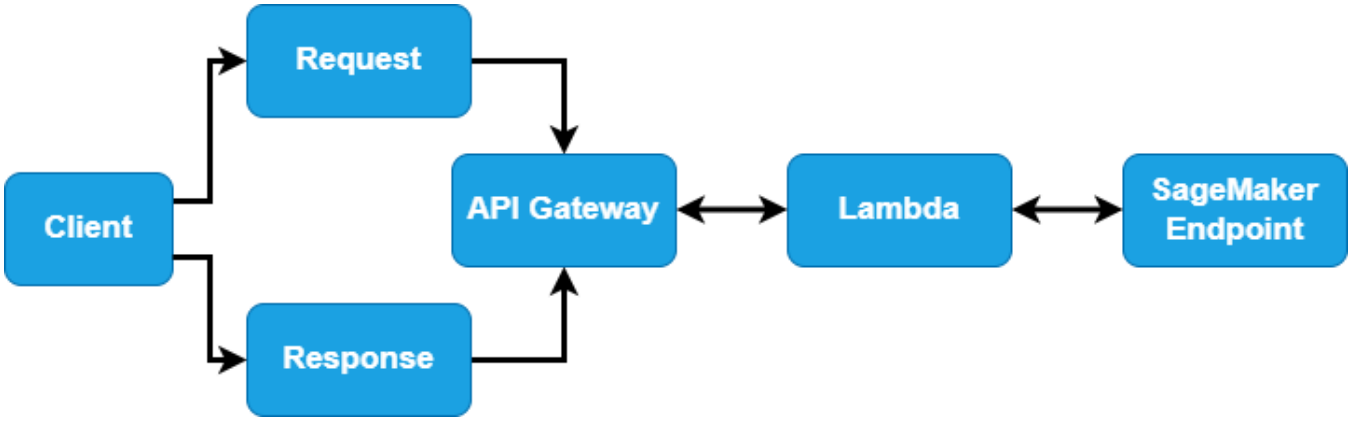

Figure 12: Architecture diagram

8. CONCLUSION

8.1 Reflection

This project demonstrates a complete ML engineering workflow on AWS: training a deep learning classifier, improving it via hyperparameter tuning, scaling with distributed training, deploying to a real-time endpoint,

and integrating inference with Lambda. The most challenging part was ensuring stable distributed training and reliable deployment behavior across different instance types and lifecycle constraints (timeouts, endpoint health checks, and argument parsing in scripted runs).

8.2 Improvement Opportunities

To improve model quality toward real-world usability:

1. **Data improvements**
   - Add more samples for confusing neighboring classes (3/4/5)
   - Add augmentations targeted to real-world variation (blur, occlusion, brightness shifts)

2. **Modeling improvements**
   - Try more modern backbones (EfficientNet / ConvNeXt) or fine-tune longer
   - Use class-weighted loss or focal loss if imbalance exists

3. **Evaluation improvements**
   - Add stratified cross-validation (or repeated holdout) for robustness
   - Track per-class precision/recall/F1 and analyze failure cases by example images

8.3 Cost and Resource Cleanup Confirmation

After validating the deployed model (real-time endpoint smoke test and Lambda-to-endpoint integration), all temporary cloud resources created for inference were cleaned up to prevent ongoing charges. Specifically, the SageMaker endpoint was deleted (releasing endpoint hosting resources) and the endpoint configuration was also removed explicitly as part of defensive cleanup.

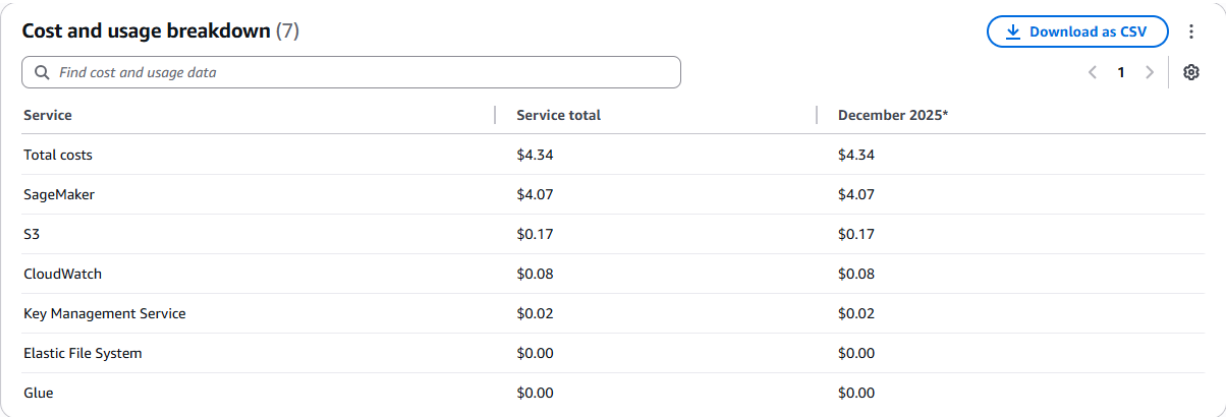

Figure 13: Usage data in Cost Explorer

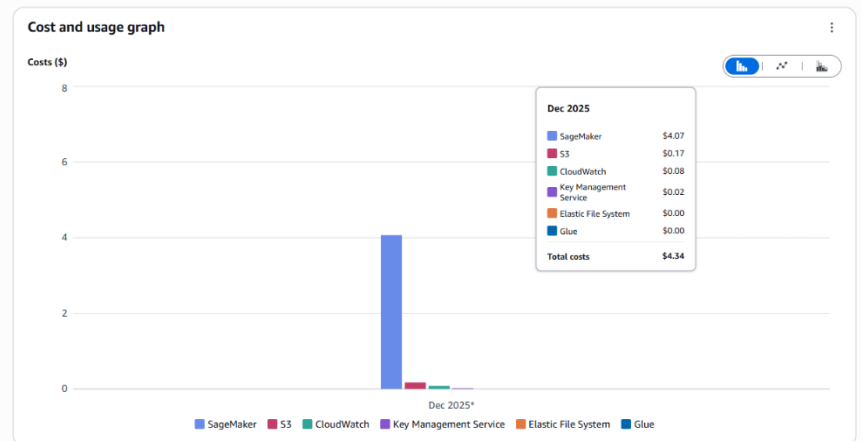

Figure 14: Usage Comparison in Cost Explorer

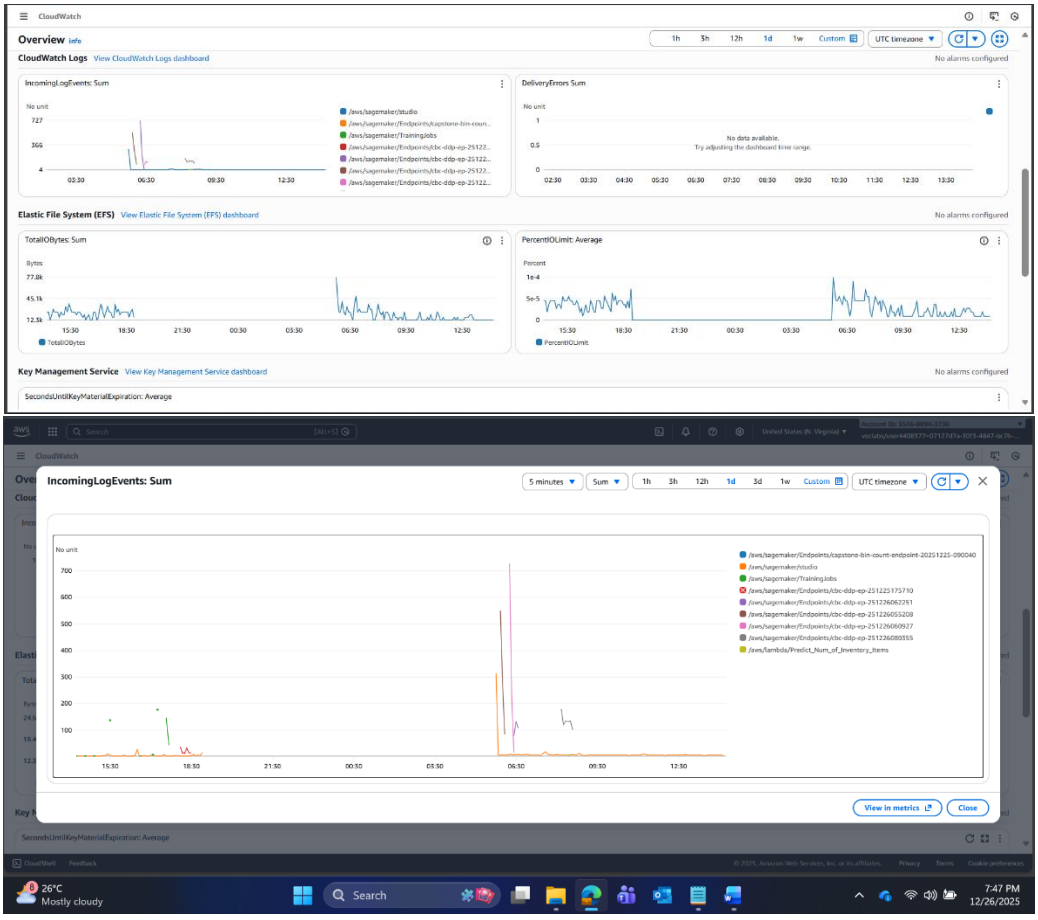

Figure 15: Cloud Watch Monitoring Logs

9. REFERENCES

9.1 AWS SAGEMAKER

- Amazon SageMaker – Distributed training (Developer Guide) - https://docs.aws.amazon.com/sagemaker/latest/dg/distributed-training.html
- Amazon SageMaker Distributed Data Parallel (SMDP) – Overview & Concepts (SageMaker Examples / Docs landing) - https://github.com/aws/amazon-sagemaker-examples
- SageMaker Training Toolkit (how SageMaker runs user scripts, env vars, distributed launching) - https://github.com/aws/sagemaker-training-toolkit
- CreateHyperParameterTuningJob (API Reference) - https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_CreateHyperParameterTuningJob.html
- CreateTrainingJob – EnableManagedSpotTraining / MaxRuntimeInSeconds / MaxWaitTimeInSeconds (API Reference) - https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_CreateTrainingJob.html
- Troubleshoot model deployment (includes ping/health check failures) - https://docs.aws.amazon.com/sagemaker/latest/dg/deploy-model-troubleshoot.html
- boto3 – SageMaker Runtime: invoke_endpoint - https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sagemaker-runtime/client/invoke_endpoint.html
- AWS Lambda – Python handler basics - https://docs.aws.amazon.com/lambda/latest/dg/python-handler.html

9.2 OTHERS

- Torchvision – ResNet model docs - https://pytorch.org/vision/stable/models/resnet.html
- Torchvision – Model weights enums (ResNet50_Weights) and “pretrained → weights” deprecation - https://pytorch.org/vision/stable/models.html
- Torchvision – ImageFolder dataset - https://pytorch.org/vision/stable/generated/torchvision.datasets.ImageFolder.html
- PyTorch – Saving and loading models (state_dict, checkpoints) - https://pytorch.org/tutorials/beginner/saving_loading_models.html
- Amazon CloudWatch Logs – Getting started / concepts - https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/WhatIsCloudWatchLogs.html