AWS Machine Learning Engineer Nanodegree

Capstone Project Report

Lorna Yen 12th Nov, 2024

1. Problem Definition

1.1 Objective

The objective of this project is to build a machine learning model that can accurately count the number of objects in a bin based on image data. This task is challenging because bins may contain varying quantities and configurations of objects, often with overlapping items and similar colors, making it difficult to perform precise counts manually or through basic image processing techniques. Automating this process with a trained model can improve efficiency in distribution centers by providing real-time and accurate object counts for better inventory control.

1.2 Problem Statement

The objective of this project is to build a machine learning model that can accurately count the number of objects in a bin based on image data. This task is challenging because bins may contain different quantities and configurations of objects, and object colors may be ambiguous, making it difficult to perform precise counts manually or through basic image processing techniques. Automating this process with a trained model can improve the efficiency of distribution centers by providing real-time and accurate object counts for better inventory control.

1.3 Goal

The primary goal of this project is to develop an end-to-end machine learning pipeline for automating object counting in bin images. This pipeline, built on Amazon SageMaker, will incorporate data ingestion, preprocessing, hyperparameter tuning, training, debugging, and deployment steps. Although accuracy in object counting is a critical outcome, the main objective is to establish a scalable and efficient pipeline that could be adapted for real-world applications. The target benchmark for accuracy is based on prior work in the Amazon Bin Image Dataset Challenge, where a ResNet34-based model achieved 55.67%. This benchmark serves as a reference to gauge the pipeline's capability in training effective models, but the focus remains on the robustness and flexibility of the pipeline itself.

2. Analysis

2.1 Dataset Description

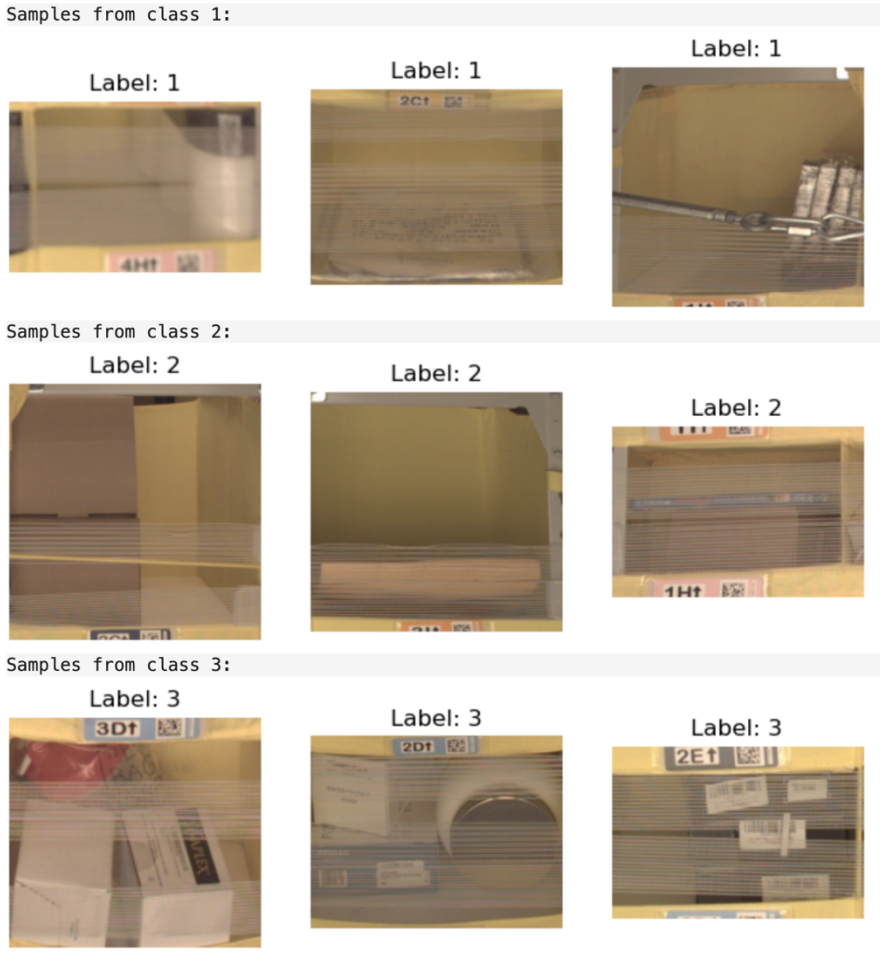
This project utilizes the Amazon Bin Image Dataset[1], which consists of approximately 500,000 images of bins, each containing a variable number of objects. Due to resource constraints, a subset of 10,441 images is used, provided by Udacity for educational purposes. Each image includes metadata indicating the count of objects in the bin, which serves as the label for the classification task. This subset contains images with object counts ranging from 1 to 5, providing diverse configurations for training the model.

We utilize a data subset list, `file_list.json`, provided by Udacity, to download images from the Amazon Bin Image Dataset's S3 storage. This file includes labels and file URLs, structured as follows:

```
{ "1": ["data/metadata/100313.json", "data/metadata/09915.json",
 ...], "2": ["data/metadata/100894.json", ...], ... }
```

Each image is accompanied by metadata for object count and other attributes.

Sample Image Data

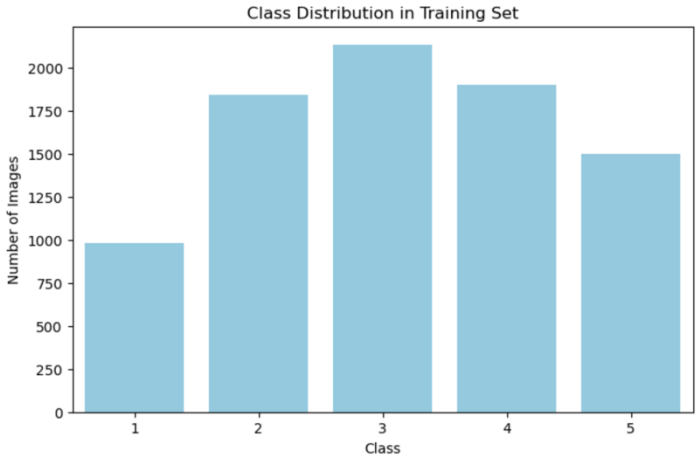


2.2 Data Exploration and Exploratory Visualization

The exploration process started by examining the class distribution, color distribution, and image dimensions within the dataset. This analysis provided insights into potential class imbalances, dominant color patterns, and variations in image dimensions, all of which are essential factors in designing and fine-tuning a model for the object counting task.

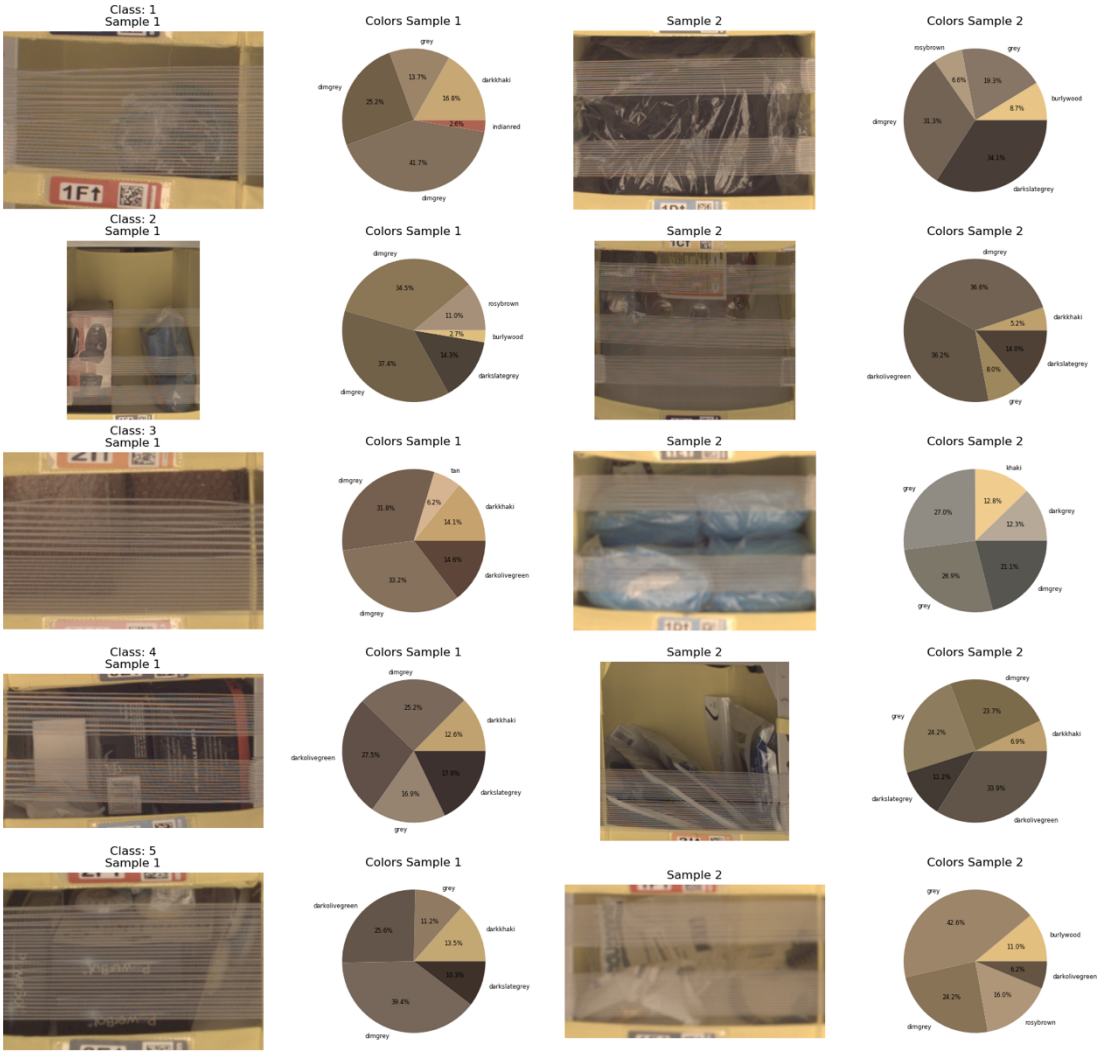
2.2.1 Class Distribution

To understand the distribution of object counts across classes, a bar plot was generated to display the number of images in each class. The bar plot illustrates the distribution of image counts across five object count classes in the training set, labeled 1 through 5. Each bar represents the total number of images available for a specific object count class. Class 3 has the highest representation, with 2132 images, followed closely by Classes 2 and 4, each containing over 2,000 images. Class 1 has the lowest representation, with just around 1,000 images, and Class 5 contains 1,500 images. This uneven distribution indicates that certain object count classes, particularly Class 1, are underrepresented, which may lead to imbalances in model training, potentially affecting the model's ability to generalize across all classes. To address this, stratified sampling will be applied to ensure balanced splits across training, validation, and testing datasets.

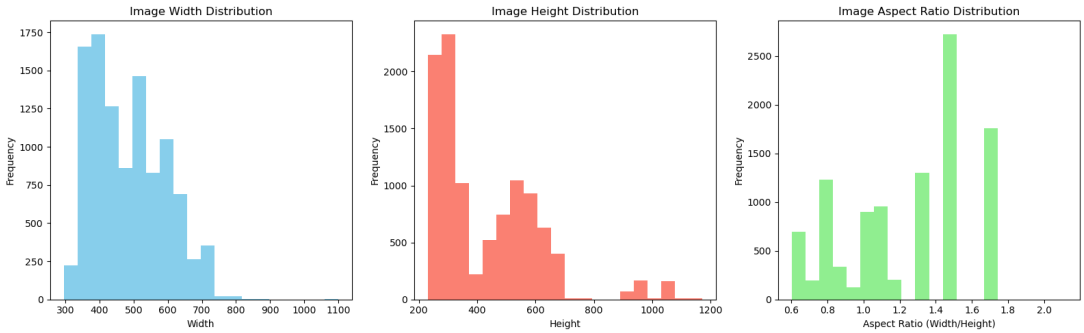


2.2.2 Color Analysis

Using KMeans clustering, we identified the top 5 dominant colors in randomly selected images from each class to assess color distribution. Across all classes, common colors like **dimgray**, **darkolivegreen**, and **grey** were prevalent, with subtle variations in secondary shades such as **darkkhaki** and **burlywood**. This consistency suggests that while color may aid in object differentiation to some extent, other features, like shape or texture, may be more influential for accurate object counting.



2.2.3 Image Dimension Analysis



The dataset's image dimensions vary in width, height, and aspect ratio, as illustrated by the three histograms. The **width distribution** shows a concentration of images between 400 and 500 pixels, with fewer images beyond 600 pixels, indicating that most images are of moderate width. The **height distribution** reveals a similar pattern, with most images falling between 300 and 500 pixels, though there are some outliers at larger dimensions. The **aspect ratio distribution** peaks around 1.4 and 1.6, suggesting that many images have a rectangular format rather than being perfectly square. These findings indicate that

while the dataset has some dimensional consistency, there are enough variations that resizing and aspect ratio preservation will be essential in preprocessing for effective model training.

To enhance model performance, images will undergo preprocessing steps, including resizing to 224x224 pixels and normalization to standardize color channels, to meet the input requirements of our proposed model (ResNet50). Data augmentation techniques, such as horizontal flipping and slight brightness/contrast adjustments, will be applied to improve generalization during training.

2.3 Algorithms and Techniques

This project leverages a convolutional neural network (CNN) based on the ResNet50 architecture, adapted for object counting in bin images. ResNet50, with its 50 layers and residual connections, is effective at capturing complex visual patterns and mitigating the vanishing gradient problem. By fine-tuning the pretrained ResNet50 and modifying its final layers, the model is designed to capture spatial and subtle visual differences crucial for accurate object counts in bins.

To enhance performance, hyperparameter tuning was conducted to optimize key parameters, including learning rate, batch size, and the number of epochs. The model was trained using the Adam optimizer with cross-entropy loss as the objective metric, aiming for high accuracy and robust generalization across different image scenarios.

2.4 Benchmark

This project references a benchmark accuracy of 55.67%, which was achieved in the Amazon Bin Image Dataset Challenge using a ResNet34 model[2]. While the primary focus of this project is to build a scalable machine learning pipeline on AWS rather than enhancing model accuracy, this benchmark provides a useful reference point for evaluating model performance. The benchmark model used accuracy and RMSE (Root Mean Square Error) as evaluation metrics.

3. Methodology

3.1 Data Preprocessing

The dataset was preprocessed with the following steps to enhance model performance:

- **Resizing:** All images were resized to 224x224 pixels to maintain a consistent input size, aligning with the input requirements of the ResNet50 model.
- **Normalization:** Pixel values were standardized across RGB channels to match the ImageNet format, which improves model compatibility and performance.
- **Data Augmentation (Training Set):**

- **Random Horizontal Flip:** Applied to enhance generalization by introducing variations in image orientation.
- **Brightness and Contrast Adjustment:** Slight adjustments were added to improve the model's robustness to varying lighting conditions.

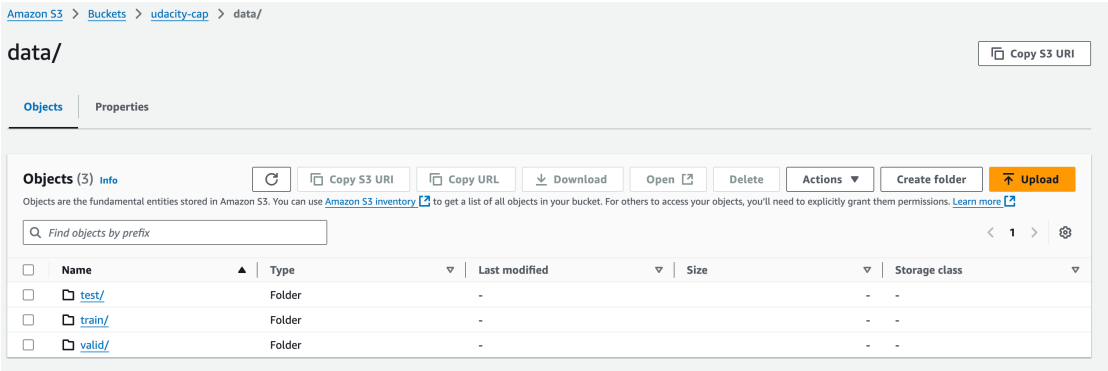
- **Validation and Test Sets:**
  - Applied only resizing and normalization (without augmentation) to ensure stable and reliable evaluation metrics.

3.2 Implementation

We set up a Jupyter Notebook in SageMaker Studio on an **ml.t3.medium** instance to perform our implementation tasks. Additionally, we created three scripts: **hpo.py**, **inference.py**, and **train.py** to automate their respective processes.

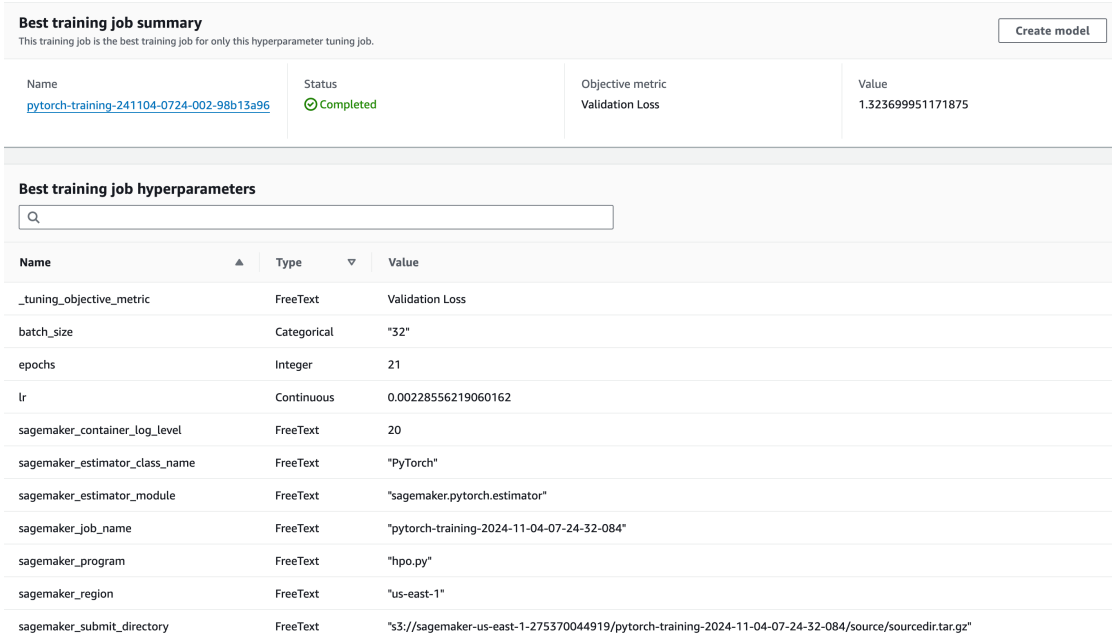
3.2.1 Data Ingestion

First, we download a subset of data from the Amazon Bin Image Dataset's S3 buckets. After downloading, we split the data into training, validation, and test sets with an 80%, 10%, and 10% ratio, respectively. Finally, we upload these splits back to the S3 buckets.



3.2.2 Hyperparameter Tuning

For hyperparameter tuning, we first create a search space in our SageMaker notebook, defining ranges and categories for key parameters such as learning rate, batch size, and epochs. The **hpo.py** script is used to automate the tuning process, running multiple training jobs with various combinations of these parameters to minimize validation loss. We specify an objective metric for optimization and use SageMaker's HyperparameterTuner to evaluate and refine the model, helping identify the best-performing hyperparameters for improved accuracy and generalization. Following is the screenshot of the best tuning job summary:



Best training job summary

This training job is the best training job for only this hyperparameter tuning job.

Create model

Name	Status	Objective metric	Value
pytorch-training-241104-0724-002-98b13a96	Completed	Validation Loss	1.323699951171875

Best training job hyperparameters

Name	Type	Value
_tuning_objective_metric	FreeText	Validation Loss
batch_size	Categorical	"32"
epochs	Integer	21
lr	Continuous	0.00228556219060162
sagemaker_container_log_level	FreeText	20
sagemaker_estimator_class_name	FreeText	"PyTorch"
sagemaker_estimator_module	FreeText	"sagemaker.pytorch.estimator"
sagemaker_job_name	FreeText	"pytorch-training-2024-11-04-07-24-32-084"
sagemaker_program	FreeText	"hpo.py"
sagemaker_region	FreeText	"us-east-1"
sagemaker_submit_directory	FreeText	"s3://sagemaker-us-east-1-275370044919/pytorch-training-2024-11-04-07-24-32-084/source/sourcedir.tar.gz"

3.2.3 Model Profiling and Debugging¶

We set up Amazon SageMaker's Debugger and Profiler with key rules to monitor potential training issues such as vanishing gradients, overfitting, overtraining, poor weight initialization, and stagnant loss, enabling early detection and optimization of model performance. Additionally, Profiler rules tracked GPU utilization and other system metrics, generating a detailed report on CPU, GPU, and memory usage. This comprehensive setup allowed us to monitor and optimize resource efficiency throughout the training process.

```
# Set debugging and profiling rules
rules = [
    Rule.sagemaker(rule_configs.vanishing_gradient()),
    Rule.sagemaker(rule_configs.overfit()),
    Rule.sagemaker(rule_configs.overtraining()),
    Rule.sagemaker(rule_configs.poor_weight_initialization()),
    Rule.sagemaker(rule_configs.loss_not_decreasing()),
    ProfilerRule.sagemaker(rule_configs.LowGPUUtilization()),
    ProfilerRule.sagemaker(rule_configs.ProfilerReport()),
]
```

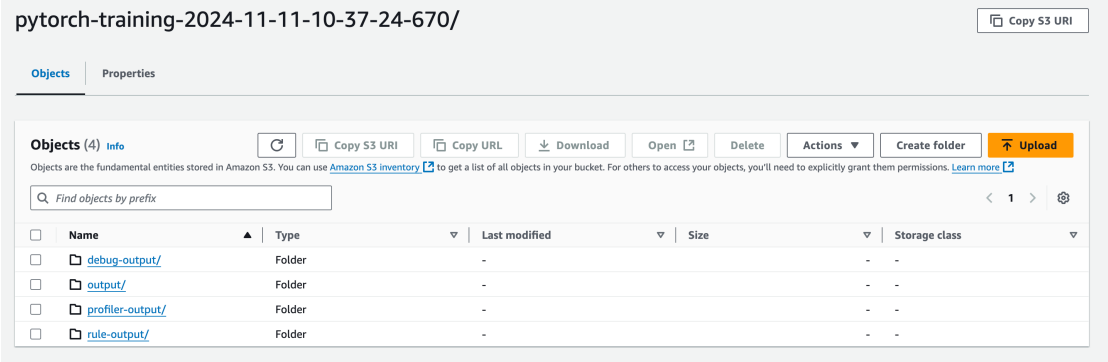
3.2.4 Model Training with the Best Hyperparameters

In this step, we use the **train.py** script with optimal hyperparameters. The script incorporates a debsugging hook to capture training metrics, with all artifacts stored in S3 for later analysis.

- **Training Script**: Initializes a pretrained ResNet50 model, sets up data loaders with preprocessing and augmentation, and includes a SageMaker Debugger hook for monitoring.
- **PyTorch Estimator:** Configured with the train.py script, an ml.p3.2xlarge instance, optimized hyperparameters, and debugging/profiling rules to monitor metrics and detect training issues.

```
estimator = PyTorch(
    entry_point="train.py",
    source_dir="scripts",
    role=role,
    framework_version="1.8.0",
    py_version="py3",
    instance_count=1,
    instance_type="ml.p3.2xlarge",
    debugger_hook_config=hook_config,       # Hook configuration
    rules=rules,                            # Debugging rules
    profiler_config=profiler_config,        # Profiler configuration
    hyperparameters=best_hyperparameters_cleaned, # Use the best hyperparam
eters from tuning job
    output_path="s3://udacity-cap/output/",
)
```

- **Artifact Storage**: Model checkpoints, logs, and profiler reports are saved in S3 for further inspection.



3.2.5 Model Deploying and Inference

We deployed the trained model to a SageMaker endpoint using a PyTorchModel setup, specifying **inference.py** as the entry point to handle prediction requests. The ImagePredictor class managed image serialization and deserialization, and we deployed the model on an ml.m5.large instance. Inference requests were processed by converting images to byte arrays and sending them to the endpoint for prediction. The endpoint returned JSON responses with predicted classes and confidence scores, facilitating real-time image-based inference.

3.3 Refinement

At the beginning of the project, an initial round of hyperparameter tuning was conducted to establish optimal starting parameters while conserving AWS resources.

```
hyperparameter_ranges = {
    "lr": ContinuousParameter(0.001, 0.1),
    "batch_size": CategoricalParameter([32, 64, 128, 256, 512]),
    "epochs": IntegerParameter(10, 40)
}
```

The hyperparameter tuning was conducted using a Bayesian optimization algorithm. After five iterations, the best hyperparameters were identified as follows:

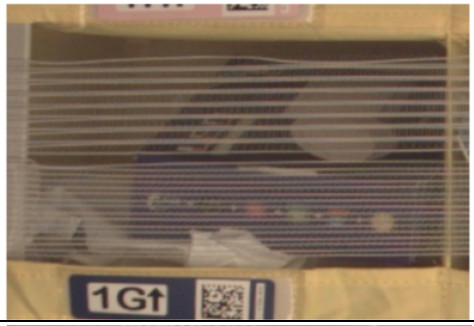
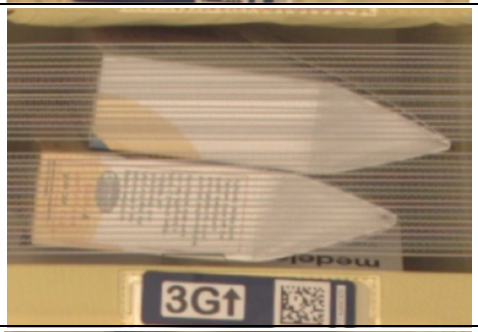
```
Best Hyperparameters: {
'batch_size': 32, 'epochs': 21,
'lr': 0.00228556219060162}
```

4. Results

The primary evaluation metric for this project will be accuracy in predicting the correct object count in each bin. Two rounds of training were performed using the tuned parameters. Although these adjustments led to slight improvements in validation accuracy, the gains were modest, indicating potential constraints in the model's capacity to generalize further with the current dataset and architecture.

Training Round	Epoch	Validation Accuracy
First Run	1	30.07%
	5	33.33%
	10	32.85%
	20	33.72%
Second Run	1	31.80%
	5	33.91%
	10	31.70%
	15	31.32%
	21	35.06%

Finally, we deployed the model as an endpoint on SageMaker to predict object counts in unseen bin images. We randomly selected five images from the Amazon Bin Image Dataset to evaluate the model's performance. Below are the prediction results:

Image	Prediction
	Predicted class: 4, Confidence: 0.33
	Predicted class: 3, Confidence: 0.31
	Predicted class: 3, Confidence: 0.41
	Predicted class: 2, Confidence: 0.35

	Predicted class: 4, Confidence: 0.56

The predictions were not highly accurate, reflecting the limitations observed during the two rounds of training. This suggests that more advanced techniques need to be explored for improved accuracy.

5. Conclusion

This project aimed to develop a machine learning pipeline using a ResNet50-based model for object counting in bin images, leveraging AWS SageMaker for hyperparameter tuning, training, profiling, and deployment. Despite modest improvements observed during refinement, the model's final test accuracy indicated limitations in its ability to generalize effectively to unseen images. This may point to inherent constraints in the dataset, model architecture, or feature extraction techniques.

The deployment of the model as an endpoint provides a functional prototype for real-time inference, but the results indicate a need for further optimization. Color similarity between objects contributed to prediction inaccuracies, suggesting that improvements could include exploring more advanced architectures tailored to object counting tasks, increasing dataset diversity, or implementing more sophisticated data augmentation methods. Overall, this project serves as a foundational step, demonstrating the potential of automated pipelines for model training and deployment, while highlighting areas requiring further development to achieve practical performance goals.

Reference

[1] Amazon, "https://registry.opendata.aws/amazon-bin-imagery/," 2024. [Online]. Available: https://registry.opendata.aws/amazon-bin-imagery/.

[2] silverbottlep, "Amazon Bin Image Dataset(ABID) Challenge," 20 Jully 2017. [Online]. Available: https://github.com/silverbottlep/abid_challenge.