AWS Machine Learning Engineer Nanodegree
Capstone Project Report:
Inventory Monitoring at Distribution Centers

Ebrahim Pichka
April, 2023

1- Project Overview

Inventory monitoring at distribution centers is a critical task to ensure that the correct number of items are in stock and available for delivery. To improve the accuracy and efficiency of this process, machine learning, deep learning, and computer vision technologies can be utilized on AWS Sagemaker. In this project, the focus is on developing a model that can accurately count the number of objects in each bin that is being moved by robots in the distribution center. This requires a robust model that can handle multiple objects in a bin and can detect them even when they are partially obstructed.

The use of deep learning models, such as convolutional neural networks (CNNs), can provide the required accuracy in object detection and recognition. AWS Sagemaker provides a comprehensive set of tools and services for training and deploying machine learning models at scale. It also supports computer vision tasks through Amazon Rekognition, which can help in analyzing and detecting objects in images and videos. By deploying the model on AWS Sagemaker, the system can be easily integrated with existing distribution center infrastructure and be used for real-time inventory monitoring. This would enable distribution centers to accurately track their inventory and ensure that delivery consignments have the correct number of items, leading to improved operational efficiency and customer satisfaction.

2- Problem Statement

For this project, we will design and train a model that will need to classify the number of objects in bins that are being moved by robots, which presents a unique challenge due to the dynamic and unpredictable nature of the environment. The model will need to be trained on a large dataset of images and videos of bins with varying numbers and types of objects, as well as different lighting conditions and backgrounds. Additionally, the model will need to be able to handle occlusions, where objects in the bin may be partially or fully obstructed by other objects or the bin itself.

Once trained, the model can be deployed on AWS Sagemaker and integrated with the distribution center's existing infrastructure. This could involve using cameras or sensors to capture images or video feeds of bins as they move through the distribution center. The captured data is then processed in real-time using the deployed model to count the number of objects in each bin. The results can be displayed on a dashboard or sent to a central inventory management system to keep track of stock levels.

Using machine learning and computer vision technologies for inventory monitoring can provide significant benefits for distribution centers. By automating the counting process, it reduces the chances of human error and saves time and resources. Additionally, by having accurate inventory data in real-time, the distribution center can optimize their operations by ensuring that stock is replenished when needed and reducing the risk of overstocking. Ultimately, this can lead to improved customer satisfaction, reduced waste, and increased profitability for the distribution center.

3- Metrics

For this specific task we will be using Classification Accuracy as evaluation metric, since the data is generally balanced and we approached the problem using classification techniques.

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Number\ of\ Instances} * 100$$

Also, we used Cross Entropy Loss as the training Cost Function for training the model.

$$CELoss = -\log \frac{e^{x_i}}{\sum_{c=1}^{N} e^{x_c}}$$

4- Data Exploration and Exploratory Visualization

The Amazon Bin Image Dataset, containing 500,000 images of bins containing one or more objects (maximum of 5 objects and minimum of 1), will be used for this project. The dataset also includes a metadata file containing information about each image, such as the number of objects, dimension, and object type. The dataset will be preprocessed and cleaned before training the machine learning model.

We are going to use a subset of 10,000 images of this dataset which is then splitted into two partitions for training, and validation with ratios of %85 and %15 respectively. The number of images are balanced among classes, i.e., there will be an equal number of train, and validation images for each class (number of objects in bin). Each section consisting of 5 classes, indicating the number of objects in each bin (from 1 to 5 objects respectively). As shown in image 4-1 the overall distribution of classes are as follows:

1,228 images (%12) in class 1
2,299 images (%22) in class 2
2,666 images (%25) in class 3
2,373 images (%23) in class 4
1,875 images (%18) in class 5

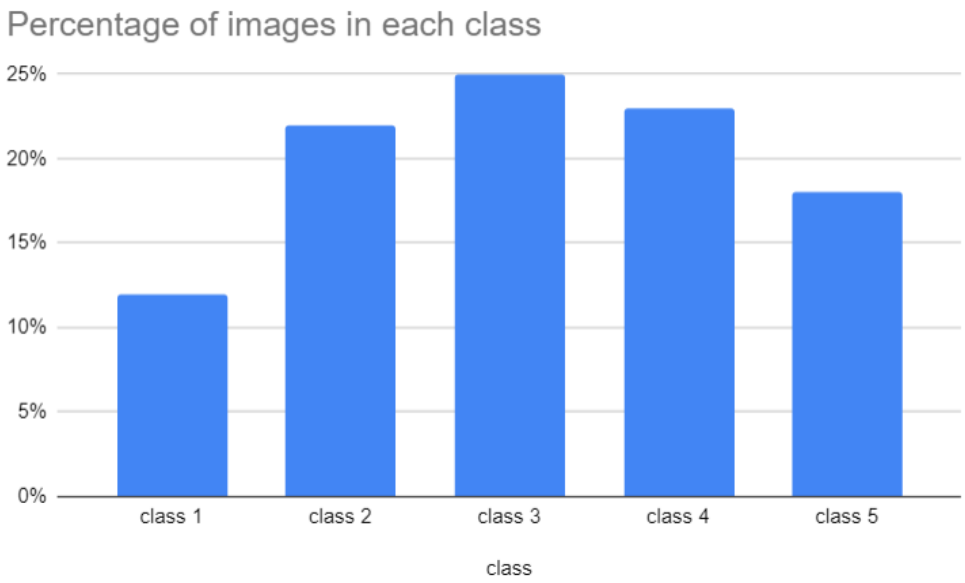

image 4-1: visualization of data balance and distribution

The images in the data set were captured while robots carried around pods during the operations in an amazon fulfilment centre. The bin sizes vary depending on the size of the objects in them. Tapes in front of the bins prevent the objects from falling out of the bins. The tapes are sometimes in the way and might make the objects in the images unclear. The images are in different sizes, so to be able to train the model, we needed to resize them into similar sizes. This step could be included in the data augmentation pipeline in the training phase as well, however, to reduce training time and prevent bottlenecks during training we resized all the images beforehand and saved them in an S3 Bucket. Image 4-2 demonstrates the created S3 bucket containing the train and validation image directories.

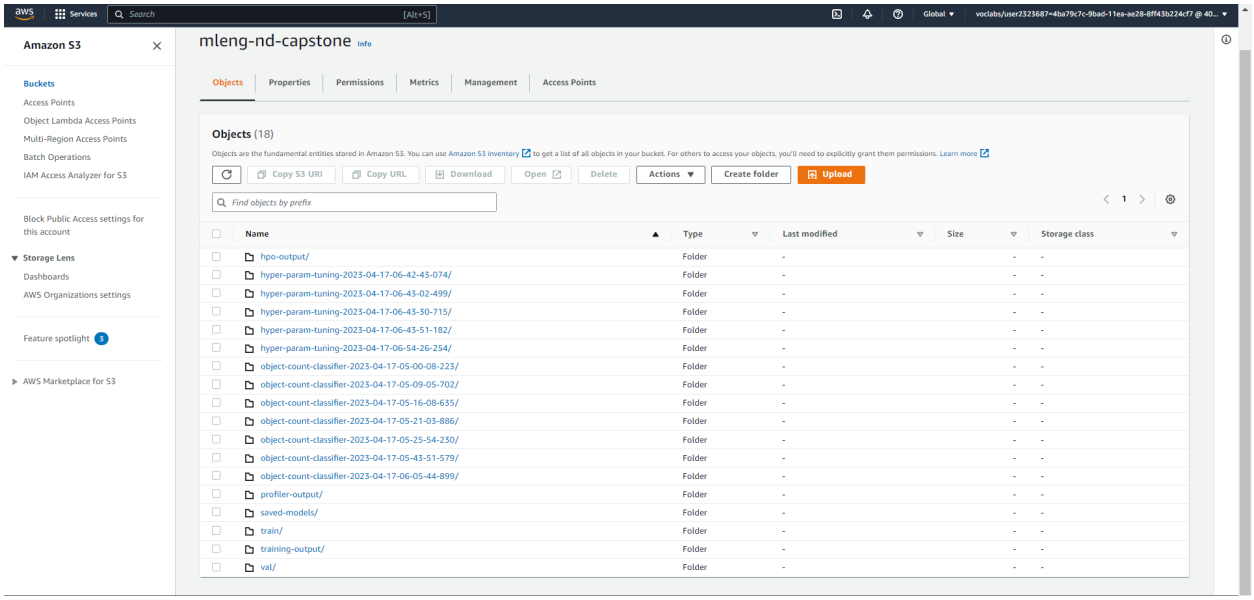

image 4-2: created S3 bucket

Different samples of each class of the training data are illustrated in the image 4-3.

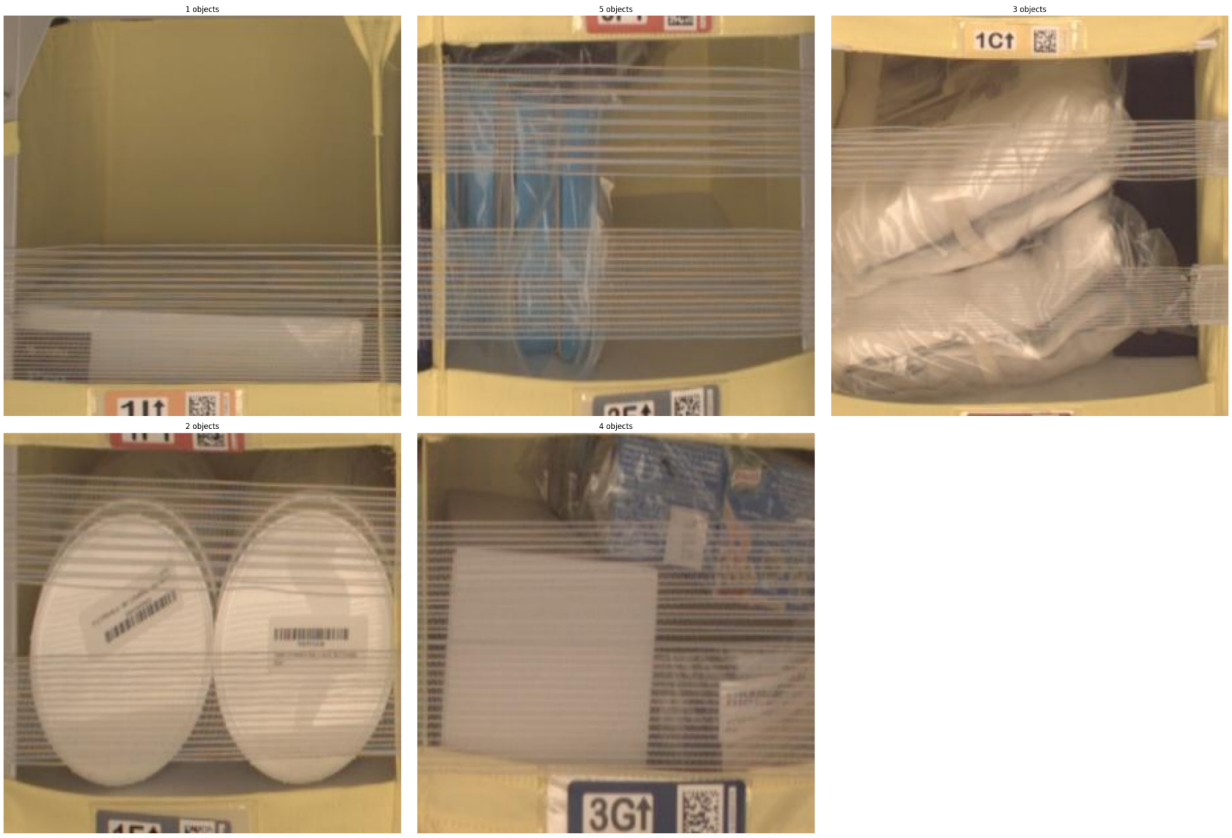

image 4-3: sample images from each class (1 to 5)

5- Solution Statement

The proposed solution is to train a machine learning model using the Amazon Sagemaker, which can accurately count the number of objects in each bin. The model will be trained using SageMaker Notebook instance. Different machine learning architectures can be used for this task, such as pre-trained convolutional neural networks or custom neural network architectures, We will discuss our choice of architecture further in this section. The trained model will be able to predict the number of objects in each bin using the input image.

5-1- Algorithms and Techniques

For this specific task, we are going to use PyTorch (v1.4) framework to design, implement, and train the proposed deep learning model in the AWS Sagemaker notebook instance. We will use ResNet-50 architecture as the network architecture and model configuration, for image classification. Moreover, we use the Adam Optimization algorithm for training the network.

5-2- Implementation

The baseline model will be a simple ResNet-50 model with a last fully-connected layer of size 128 neurons. We did not use any hyperparameter optimization for the baseline model. We trained the baseline model for 10 epochs, with the learning rate of 0.001 and bach size of 128. The model is trained to count the number of objects in each bin using pretrained weights from ImageNet dataset. The performance of this model will be compared to the performance of the more sophisticated model developed in this project to evaluate the effectiveness of the proposed solution.

```
hyperparameters = {
    'lr':0.0001,
    'batch-size':128,
    'test-batch-size':128,
    'epochs':10,

}
```

image 5-1: baseline hyperparameters (manually set)

```
estimator = PyTorch(
    entry_point="train.py",
    base_job_name='object-count-classifier',
    role=get_execution_role(),
    framework_version="1.4.0",
    instance_count=1,
    instance_type="ml.g4dn.xlarge",
    py_version='py3',
    output_path = "s3://mleng-nd-capstone/training-output/",
    hyperparameters=hyperparameters
)
```

image 5-2: baseline model estimator instance

The training pipeline, model architectures, and other specifications is defined in the *`train.py`* script file. This script accepts the required hyperparameters as command arguments and performs training and validation steps based on the given hyperparameters.

The training has been done on a ***`ml.g4dn.xlarge`*** instance to take advantage of the accelerated compute resources (GPUs) to be able to train the models faster, while keeping track of the incurred costs.

5-3- Benchmark

After training the baseline model, the initial validation loss and accuracy is regarded as benchmark performance for further improvements. Image 5-3 illustrates the final validation metrics of the baseline model. The model has reached **1.486** loss and **%30** accuracy for validation set. We will try to improve this results further by hyperparameter optimization, in the next chapters.

```
>> Epoch 5 -> Train loss: 1.5068 | Train acc: %28 | Val loss: 1.4907 | Val acc: 29
    - Train [0/8873 (0%)] || Batch Loss: 1.554913
    - Train [6400/8873 (71%)] || Batch Loss: 1.566855
>> Epoch 6 -> Train loss: 1.5012 | Train acc: %29 | Val loss: 1.5104 | Val acc: 29
    - Train [0/8873 (0%)] || Batch Loss: 1.515097
    - Train [6400/8873 (71%)] || Batch Loss: 1.482148
>> Epoch 7 -> Train loss: 1.4955 | Train acc: %30 | Val loss: 1.4843 | Val acc: 29
    - Train [0/8873 (0%)] || Batch Loss: 1.510413
    - Train [6400/8873 (71%)] || Batch Loss: 1.479071
>> Epoch 8 -> Train loss: 1.4916 | Train acc: %30 | Val loss: 1.4617 | Val acc: 32
    - Train [0/8873 (0%)] || Batch Loss: 1.473720
    - Train [6400/8873 (71%)] || Batch Loss: 1.467149
>> Epoch 9 -> Train loss: 1.4878 | Train acc: %31 | Val loss: 1.4669 | Val acc: 31
    - Train [0/8873 (0%)] || Batch Loss: 1.518061
    - Train [6400/8873 (71%)] || Batch Loss: 1.473588
>> Epoch 10 -> Train loss: 1.4863 | Train acc: %31 | Val loss: 1.4831 | Val acc: 30
Start Model Testing
Test : Average Test loss: 1.4831, Accuracy: 30%
Model Saving
2023-04-17 06:39:20,356 sagemaker-containers INFO     Reporting training SUCCESS

2023-04-17 06:39:47 Uploading - Uploading generated training model
2023-04-17 06:39:47 Completed - Training job completed
Training seconds: 1974
Billable seconds: 1974
```

image 5-3: last 5 epochs of training logs of the baseline model

6- Data Preprocessing

As a part of data preprocessing, first all images were resized to 300 by 300 pixels and then a data augmentatrion pipeline is defined in the `train.py` file to introduce more variations of the data to the model. The Augmentation pipeline follow these steps:

- Random Horizontal Flip
- Random Rotation
- ColorJitter
- ToTensor
- Normalize

7- Refinements

To further improve the results of the baseline model, we had performed hyperparameter optimization to determine two key hyperparameters, namely **learning rate** and the **number of neurons in the final fully-connected layer**. All other hyperparameters had been fixed as before. Images 5-1 and 5-2 demonstrate the corresponding configurations for hyperparameter optimization.

```
hyperparameter_ranges = {
    "lr": ContinuousParameter(0.0001, 0.01),
    "head_size": CategoricalParameter([32, 64, 128, 256, 512]),
}
```

image 7-1: hyperparameter ranges for optimization

```
objective_metric_name = "Average Test loss"
objective_type = "Minimize"
metric_definitions = [{"Name": "Average Test loss", "Regex": "Average Test loss: ([0-9\\.]+)"}]


hpo_estimator = PyTorch(
    entry_point = "hpo.py",
    base_job_name = "hyper-param-tuning",
    role = get_execution_role(),
    instance_count = 1,
    instance_type = "ml.g4dn.xlarge", #"ml.m5.large"
    hyperparameters = hyperparameters,
    framework_version = "1.8",
    py_version = "py36",
    output_path = "s3://mleng-nd-capstone/hpo-output/"
)


tuner = HyperparameterTuner(
    hpo_estimator,
    objective_metric_name,
    hyperparameter_ranges,
    metric_definitions,
    max_jobs = 2,
    max_parallel_jobs = 2,
    objective_type = objective_type,
)
```

image 7-2: hyperparameter optimization instantiation

Hyperparametr optimization has been also done on `***ml.g4dn.xlarge***` instances to take advantage of the accelerated compute resources (GPUs) to be able to train the models faster, while keeping track of the incurred costs.

It uses the *`hpo.py`* as an entry point meaning all training steps and pipelines is defined in this script, same as *`train.py`* script.

Finally, we save the best performing set of hyperparameters with respect to Average Test Loss of the training jobs. The so far best set of hyper params is as follows:

```
{
'batch-size':                '128',
'epochs':                     '6',
'head_size':                '512',
'lr': '0.006747601467379431',
'test-batch-size':           '128'
}
```

8- Model Evaluation and Validation

After obtaining the best set of hyperparameters, we trained the model again using this set, we have also configured a Debugger and Profiler in the training job to better monitor the training. Image 8-1 show the debugger output for training and validation loss. Since the training was only done fir limited number of epochs, the decrease in the loss is subtle, however by continuing the training for further epochs we could reach better performances.

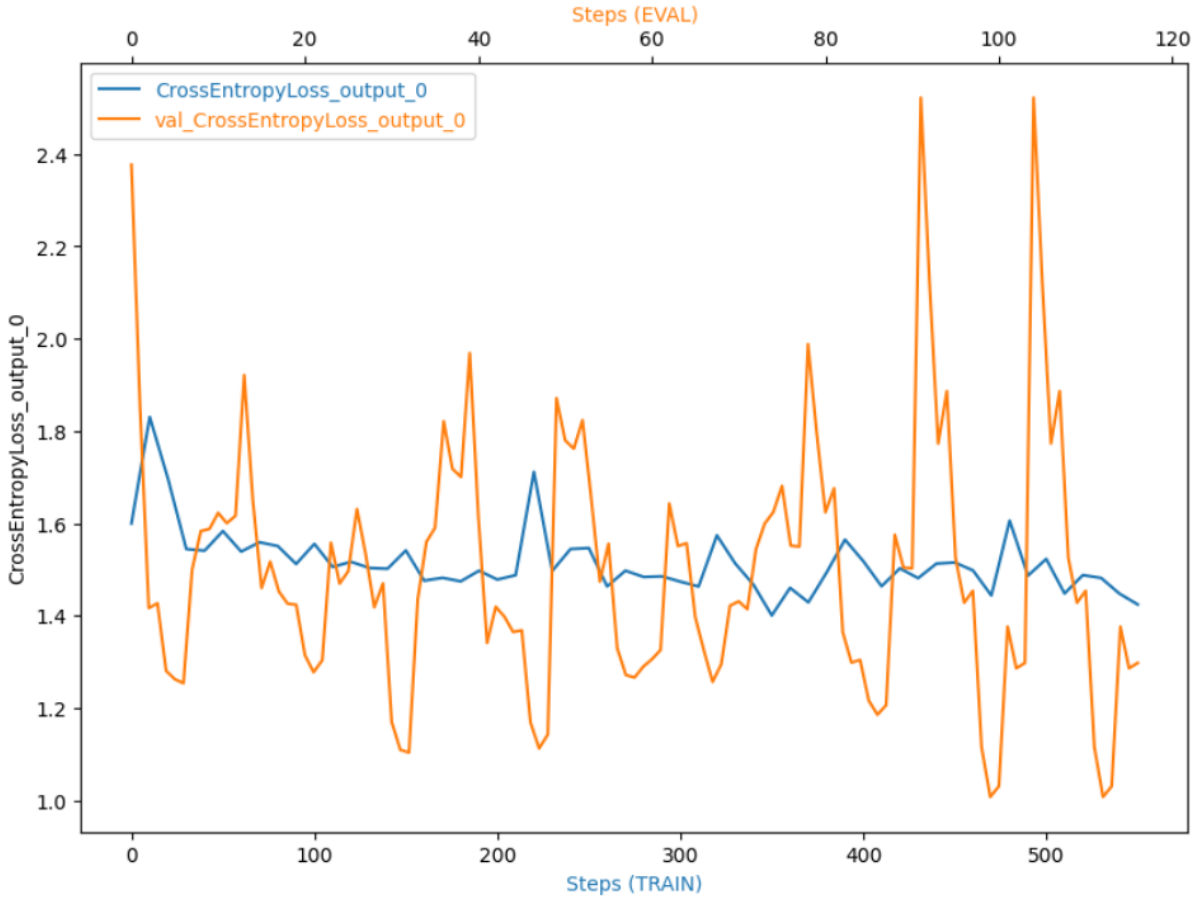

image 8-1: debugger output

The profiler report file is also included in the project submission directory as appendix as well as in the notebook file.

The trained model is then deployed to an endpoint for inference on new images. The inference is done through the *`inference.py`* script file which is written to handle different types of requests (image 8-2) . A sample image from the test set is tested by the endpoint as demonstration. (image 8-3)

```
jpeg_serializer = sagemaker.serializers.IdentitySerializer("image/jpeg")
json_deserializer = sagemaker.deserializers.JSONDeserializer()

class ImagePredictor(Predictor):
    def __init__(self, endpoint_name, sagemaker_session):
        super(ImagePredictor, self).__init__(
            endpoint_name,
            sagemaker_session=sagemaker_session,
            serializer=jpeg_serializer,
            deserializer=json_deserializer,
        )


pytorch_model = PyTorchModel(model_data=model_location, role=get_execution_role(), entry_point='inference.py',py_version='py3',
                             framework_version='1.4',
                             predictor_cls=ImagePredictor)
```

```
predictor = pytorch_model.deploy(initial_instance_count=1, instance_type='ml.m5.large')
```

```
INFO:sagemaker:Creating model with name: pytorch-inference-2023-04-17-08-46-49-990
INFO:sagemaker:Creating endpoint-config with name pytorch-inference-2023-04-17-08-46-50-660
INFO:sagemaker:Creating endpoint with name pytorch-inference-2023-04-17-08-46-50-660
-----!
```

image 8-2: deploying model to an inference endpoint

```
pred = predictor.predict(img_bytes, initial_args={"ContentType": "image/jpeg"})
pred[0]
```

```
[-1.319909691810608,
 -0.04901547729969025,
 0.3014374375343323,
 0.1349669247865677,
 -0.3881283402442932]
```

```
print(f"Actual: 3 | predicted: {np.argmax(pred, 1).item()+1}")
```

```
Actual: 3 | predicted: 3
```

image 8-3: inference demonstration on sample test image

8-1- Justification

By comparing the final results with the benchmark, we could observe that the final metrics reach the same level as the benchmarks in earlier epochs. It is highly plausible that by further training the final model, we would reach much better performances with respect to the benchmark.

9- Conclusion

AWS Sagemaker provides a comprehensive set of tools and services to support machine learning and computer vision tasks. It allows the model to be easily integrated into the distribution center's existing infrastructure, enabling real-time inventory monitoring. The captured data can be analyzed and processed, and the results can be displayed on a dashboard or sent to a central inventory management system, providing the distribution center with a clear overview of their stock levels.

Furthermore, the use of machine learning and computer vision technologies can reduce the chances of human error, which is a common problem in manual inventory counting processes. By automating the process, the distribution center can save time and resources, and have an accurate and efficient inventory monitoring system. This can help optimize their operations and improve profitability, as well as enhance customer satisfaction by ensuring that products are always available in the right quantities.

In this project we trained, analyzed, optimized, and deployed an image classification model to observe and countc the number of objects in the bins in an inventory.

In conclusion, the use of machine learning tools on AWS Sagemaker can improve the inventory management process at distribution centers. By building a robust and accurate model that can handle dynamic and unpredictable environments, distribution centers can optimize their operations, reduce the risk of human error, and improve customer satisfaction. AWS Sagemaker provides a scalable and flexible platform to support machine learning and computer vision tasks, making it an ideal solution for inventory monitoring at distribution centers.