Inventory Monitoring at Distribution Centers

CAPSTONE PROJECT REPORT

Sathyapriyan Kannan

Udacity's AWS Machine Learning Engineer Nanodegree Program

Inventory Monitoring at Distribution Centers

Capstone Project - Report

1. Domain Background:

Inventory Monitoring is a critical process in a Distribution Center. It is the process of ensuring the right number of products or items present in a particular bin. It is an integral part of the supply chain management, that ensures the flow of goods from manufacturing venue to the warehouse or a distribution center and to the sale location.

Long before the industrial age, keeping track of things consisted of manually counting and tallying items. The earliest form of inventory management dates back over 50,000 years in which people used “tally sticks” to count.

The empowerment of the distribution centers to automate its process through a digital workplace is the need of the hour to manage the demand and supply chain. During the time of the pandemic, there is a huge demand for the logistics services.

2. Problem Statement:

Manual Inventory Management requires a huge workforce, and it is also error prone. So, there is a need for an effective inventory management.

Effective Inventory Management enables the business to save time and increase productivity. The automation of this process would eliminate the manual errors, thus saving cost and time. For larger organization, the process is much complicated.

With the recent advancement in the technology, we could solve the above problem with machine automated tasks. The Computer Vision Process aids us in solving the defined problem.

With the huge rise in the demand for inventory management, scalability is a major concerned that affects the long term run of the business. With the automated workflow, scalability becomes much simpler and cost efficient.

The solution can be replicated to different objects with respective datasets for training and the outcomes are measurable with the proposed evaluation metrics.

The problem is identified as a classification problem and upon given an input image from the live camera, the count of the objects in the bin can be predicted using the machine learning model.

3. Solution Statement:

The identified problem can be solved using Computer Vision Techniques. By feeding appropriate data to train the machine learning model, a robust model can be produced.

Components:

1. Dataset / Image Source
2. Algorithm
3. Environment or the platform

Environment or the platform:

1. Amazon Web Services
2. Sage Maker Studio – to train, tune and deploy the model.
3. S3 – Storage Bucket

4. Benchmark Model:

The result of the Amazon Bin Image Dataset (ABID) Challenge from this Repository is considered as a Benchmark.

The author has achieved the accuracy of 55 percent (Approx.). The project tends to achieve or exceed this mark.

5. Evaluation Metrics:

Since it is a classification problem, the **overall accuracy** of the classification can be used to evaluate the performance of the trained model.

6. Dataset or the Image Source:

The Amazon Bin Image Dataset is to be used to train the model.

It contains over 500,000 images and metadata from bins of a pod in an operating Amazon Fulfillment Center. The bin images in this dataset are captured as robot units carry pods as part of normal Amazon Fulfillment Center operations.

Documentation about the open-source dataset can be found here.

These are some typical images in the dataset. A bin contains multiple object categories and various number of instances. The corresponding metadata exist for each bin image and it includes the object category identification (Amazon Standard Identification Number, ASIN), quantity, size

of objects, weights, and so on. The size of bins is various depending on the size of objects in it. The tapes in front of the bins are for preventing the items from falling out of the bins and sometimes it might make the objects unclear. Objects are sometimes heavily occluded by other objects or limited viewpoint of the images.

The Images are extracted from the source which are available in JPEG format and the target or label data is extracted from the corresponding JSON file.

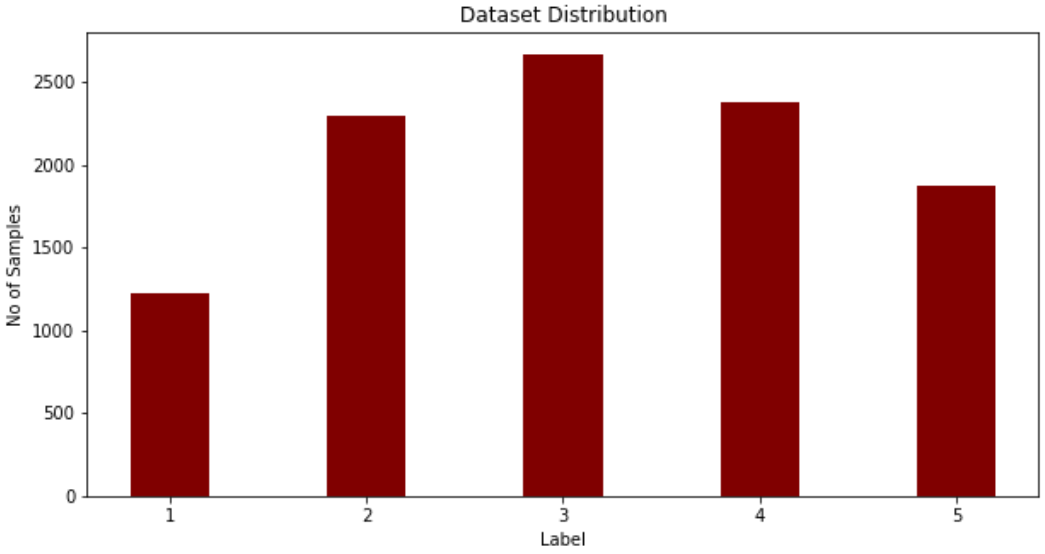
Data Extraction:

Example JPEG : https://aft-vbi-pds.s3.amazonaws.com/bin-images/523.jpg

Example JSON : https://aft-vbi-pds.s3.amazonaws.com/metadata/523.json

Label	No of Training Samples used
1	1228
2	2299
3	2666
4	2373
5	1875

We could find a slight imbalance in the no of samples used for target label – 1. As for the futuristic step the samples for label – 1 could be increased.



Visualization of the Dataset Distribution

```
  0%|          | 0/1228 [00:00<?, ?it/s]
Downloading Images with 1 objects
100%|██████████| 1228/1228 [02:18<00:00,  8.88it/s]
  0%|          | 1/2299 [00:00<06:07,  6.25it/s]
Downloading Images with 2 objects
100%|██████████| 2299/2299 [04:25<00:00,  8.67it/s]
  0%|          | 0/2666 [00:00<?, ?it/s]
Downloading Images with 3 objects
100%|██████████| 2666/2666 [05:21<00:00,  8.30it/s]
  0%|          | 0/2373 [00:00<?, ?it/s]
Downloading Images with 4 objects
100%|██████████| 2373/2373 [04:35<00:00,  8.62it/s]
  0%|          | 2/1875 [00:00<02:42, 11.52it/s]
Downloading Images with 5 objects
100%|██████████| 1875/1875 [03:29<00:00,  8.95it/s]
```

tqdm Progress of the Data Extraction

Data Pre-Processing:

The images are downloaded from the URL. Every image is converted into image bytes. In order to maintain the images are of same size, image re-sizing is performed at this stage. All images are resized to 224x244 pixels using PyTorch transformers.

The, the data loaders are created at a batch size of 32 for training, testing and validation.

Example Image used in the project after Pre Processing

7. Algorithm:

1. The Algorithm is built using the Convolution Neural Network (CNN) architecture.

2. The CNN follows a hierarchical model which works on building a network, like a funnel, and finally gives out a fully connected layer where all the neurons are connected to each other and the output is processed. Hence CNN is chosen for this project to perform the image classification.

3. The images are fed to train the model, that could learn the parameters to perform classification based on the number of items in the bin.

4. Deep Learning Framework – PyTorch

5. A corresponding Sage Maker instance will be created, and data will be fed from the S3 bucket.

6. The model is also tuned to find out the best hyper-parameters as a model improvement method.

7. A pre-trained ResNet50 Model is used.

7. Hyper-parameter Tunning:

The following hyper-parameters are identified for tunning.

1. Learning Rate
2. Batch Size
3. Epoch

The Learning rate for the Adam optimizer is identified as one of the hyper-parameters which could help improve the training process rather than the pre-defined default.

Batch Size could help us optimize the computation speed and convergence.

The Number of Epochs could help the model to better train the model over a longer period of time and different possibilities can be checked for tunning.

For the experiment 2 Jobs are crated for the tunning and one among them is chosen based on the results ie, objective metric.

	batch_size	epochs	learning_rate	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTime	TrainingElapsedTimeSeconds
1	"32"	"30"	0.098655	pytorch-training-220119-1607-001-437562e8	Completed	50.0	2022-01-19 16:12:59+00:00	2022-01-19 16:35:31+00:00	1352.0
0	"32"	"25"	0.001293	pytorch-training-220119-1607-002-8514b643	Completed	47.0	2022-01-19 16:10:08+00:00	2022-01-19 16:31:22+00:00	1274.0

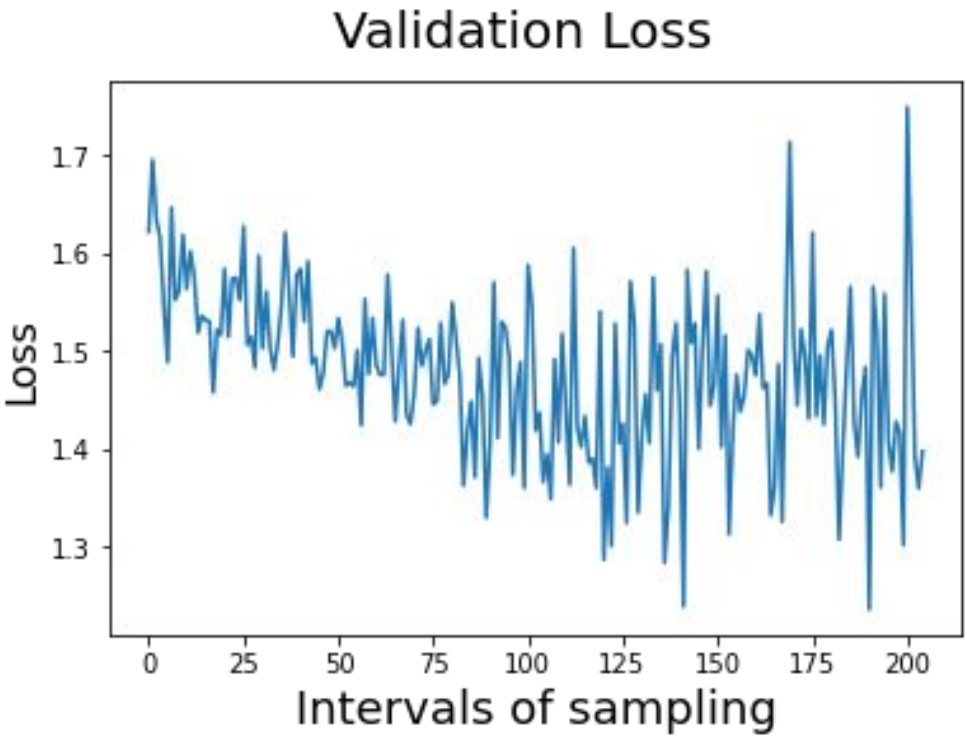
Hyper-parameter Tunning Jobs

8. Model Evaluation:

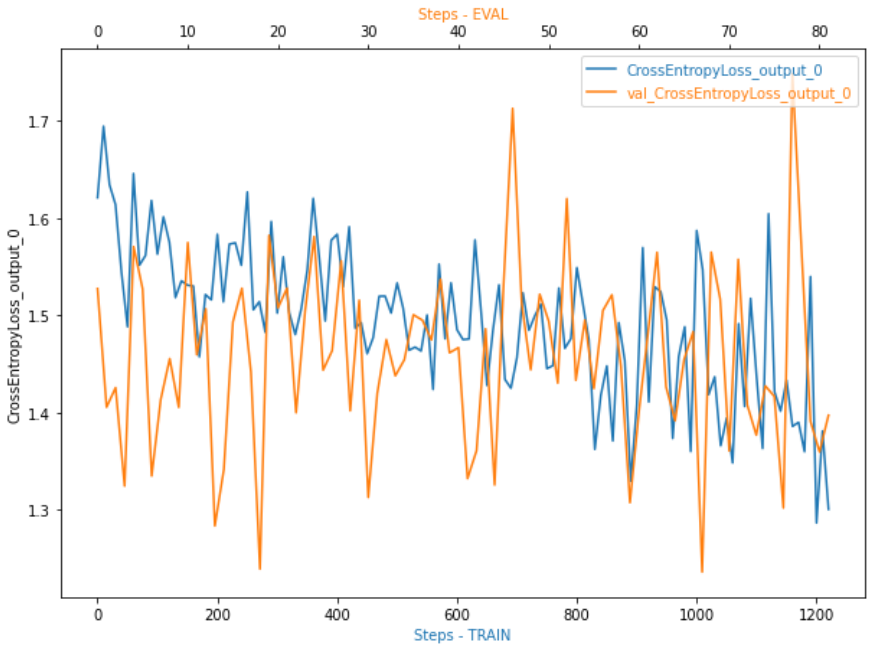
Hyperparameter tunning is performed to find out the best hyperparameters to train the model.

The validation loss of the training process remains almost constant and there are frequent highs and lows, when compared with the training loss, it seems to fit along with it as shown below.

The accuracy of the benchmark model chosen is 56 % (Approx), the experiment model didn't achieve the results of the Benchmark. So as next steps, must work more on the data and its transformation.



Validation Loss



Cross Entropy Loss of Train and Eval

Metrics form the trained model:

```
2022-01-23T10:45:48.824+05:30    Starting Model Training
2022-01-23T10:45:48.824+05:30    Epoch: 0
2022-01-23T10:45:49.825+05:30    [2022-01-23 05:15:48.853 algo-1:45 INFO hook.py:382] Monitoring the collections: CrossEntropyLoss_output_0, losses, relu_input
2022-01-23T10:45:49.825+05:30    [2022-01-23 05:15:48.854 algo-1:45 INFO hook.py:443] Hook is writing from the hook with pid: 45
2022-01-23T10:55:42.162+05:30    train loss: 50.0000, acc: 7.0000, best loss: 1000000.0000
2022-01-23T11:00:08.302+05:30    valid loss: 49.0000, acc: 8.0000, best loss: 49.0000
2022-01-23T11:00:08.302+05:30    Epoch: 1
2022-01-23T11:09:53.675+05:30    train loss: 48.0000, acc: 8.0000, best loss: 49.0000
2022-01-23T11:14:29.791+05:30    valid loss: 48.0000, acc: 8.0000, best loss: 48.0000
2022-01-23T11:14:29.791+05:30    Epoch: 2
2022-01-23T11:24:11.177+05:30    train loss: 47.0000, acc: 9.0000, best loss: 48.0000
2022-01-23T11:28:43.298+05:30    valid loss: 47.0000, acc: 9.0000, best loss: 47.0000
2022-01-23T11:28:43.298+05:30    Epoch: 3
2022-01-23T11:38:22.756+05:30    train loss: 46.0000, acc: 9.0000, best loss: 47.0000
2022-01-23T11:42:56.868+05:30    valid loss: 46.0000, acc: 10.0000, best loss: 46.0000
2022-01-23T11:42:56.868+05:30    Epoch: 4
2022-01-23T11:52:38.296+05:30    train loss: 45.0000, acc: 10.0000, best loss: 46.0000
2022-01-23T11:57:14.477+05:30    valid loss: 46.0000, acc: 9.0000, best loss: 46.0000
2022-01-23T11:57:14.477+05:30    Testing Model
2022-01-23T12:01:57.828+05:30    Testing Loss: 46.0
2022-01-23T12:01:57.828+05:30    Testing Accuracy: 9.0
2022-01-23T12:01:57.828+05:30    Saving Model
2022-01-23T12:01:58.829+05:30    2022-01-23 06:31:57,995 sagemaker-training-toolkit INFO Reporting training SUCCESS
```

We could infer from the figure that the Entropy Loss gradually decreases over the increase in the steps of the training process.

9. Cost Analysis:

The cost incurred for the Sage Maker is shown in the below image.

SageMaker		$3.30
US East (N. Virginia)		**$3.30**
Amazon SageMaker CreateVolume-Gp2		$0.02
$0.00 for SageMaker Debugger Built-in Rule Volume	78.383 GB-Mo	$0.00
$0.14 per GB-Mo of Endpoint ML storage	0.066 GB-Mo	$0.01
$0.14 per GB-Mo of Training Job ML storage	0.080 GB-Mo	$0.01
Amazon SageMaker Invoke-Endpoint		$0.00
$0.016 per GB for Endpoint Data IN	0.000000130 GB	$0.00
$0.016 per GB for Endpoint Data OUT	0.000000120 GB	$0.00
Amazon SageMaker RunInstance		$3.28
$0.0 for SageMaker Debugger Built-in Rule Instance	1.354 Hrs	$0.00
$0.00 for SageMaker Debugger Built-in Rule Instance	0.590 Hrs	$0.00
$0.05 per Studio-Notebook ml.t3.medium hour in US East (N. Virginia)	2.876 Hrs	$0.14
$0.115 per Hosting ml.m5.large hour in US East (N. Virginia)	12.297 Hrs	$1.41
$0.23 per Training ml.m5.xlarge hour in US East (N. Virginia)	1.996 Hrs	$0.46
$0.7364 for SageMaker Studio Notebook Instance ml.g4dn.xlarge per hour	1.716 Hrs	$1.26

3.28 USD is charged for the model training. The instance used is Amazon E2 – PyTorch 1.4 GPU Optimized.

If the spot instances are used for the training, the cost would have been significantly lesser compared to the current incurred costs.

Cost incurred for the S3 is as below:

Simple Storage Service		$0.29
US East (N. Virginia)		**$0.29**
Amazon Simple Storage Service Requests-Tier1		$0.21
$0.005 per 1,000 PUT, COPY, POST, or LIST requests	42,684.000 Requests	$0.21
Amazon Simple Storage Service Requests-Tier2		$0.08
$0.004 per 10,000 GET and all other requests	192,708.000 Requests	$0.08
Amazon Simple Storage Service TimedStorage-ByteHrs		$0.00
$0.023 per GB - first 50 TB / month of storage used	0.185 GB-Mo	$0.00

10. Future Steps:

1. Include More Data for Label – 1 to ensure better balance in the data

2. Better Hyper parameter tuning with more parameters

3. Implement Data Augmentation Techniques to better transform the data to help the model to learn easily from the input.

4. Try out different combinations of CNN and other models in the stacked architecture.

11. References:

1. Sagemaker -Python SDK for PyTorch - https://sagemaker.readthedocs.io/en/stable/overview.html

2. Benchmark - https://github.com/silverbottlep/abid_challenge

3. Amazon Bin Image Dataset - https://github.com/awslabs/open-data-docs/tree/main/docs/aft-vbi-pds

4. ResNet - https://pytorch.org/hub/pytorch_vision_resnet/