

# Inventory Monitoring at Distribution Centers

Ramy Gendy

February 9th, 2023

## Definition

### Project Overview

Distribution centers often use robots to move objects as a part of their operations. Objects are carried in bins which can contain multiple objects. Since no one to check the number of objects in bin that may cause loss of some of the bin items which will result low quality of service and decrease profit. Have a way to check the count of those items and avoid losing them to ensure quality of service is good practice for any distribution center.

In this project, I built a machine learning model that can count the number of objects in each bin. A system like this can be used to track inventory and make sure that delivery consignments have the correct number of items.

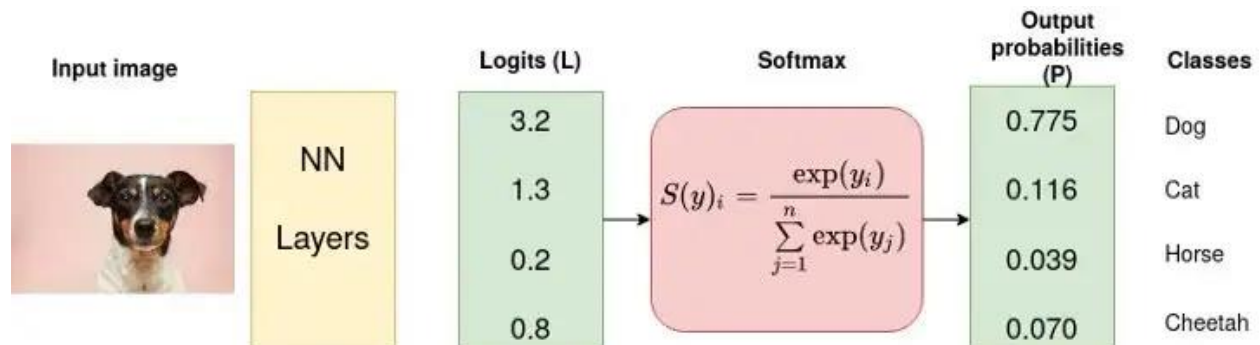
To finish this project, I used *AWS SageMaker* and its services with good machine learning engineering practices to fetch data from a database, preprocess it, then proceed to train, refine, evaluate and validate a machine learning model.

### Problem Statement

Loss of objects at the distribution centers since robots doesn't notice the missing ones while moving them around as robots lack vision which states clearly our problem is related to *computer vision domain*. Where my implemented solution lies in training of machine learning model with computer vision capabilities to check bin and decide the count based on the number of items in it. For that I will need a good source of data to build the model based on it, for that the [Amazon Bin Image Dataset](#) which 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.

## Metrics

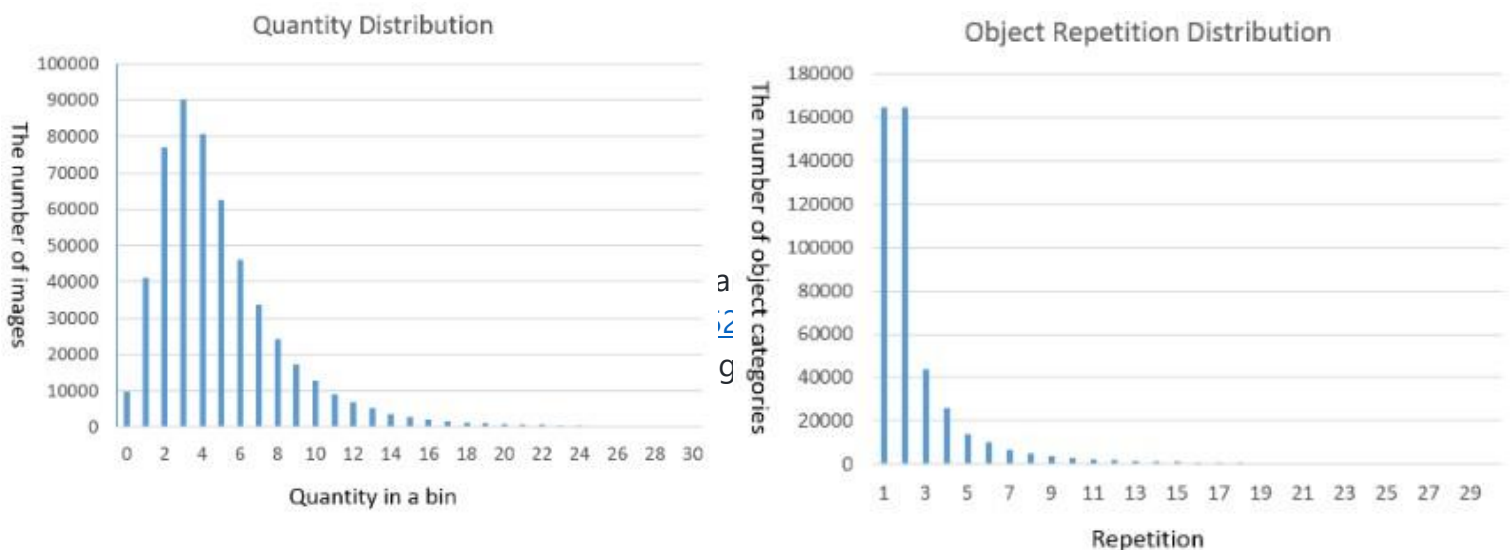
After training my ML model I will use the standard *Cross Entropy Loss function* as my metric to study my training process results along with hyperparameter tuning and optimization for quality and improvements. The reason for this is that *Cross-Entropy* is to take the output probabilities (P) and measure the distance from the truth values (as shown in Figure below) since *Cross-entropy* builds upon the idea of entropy from information theory and calculates the number of bits required to represent or transmit an average event from one distribution compared to another distribution.

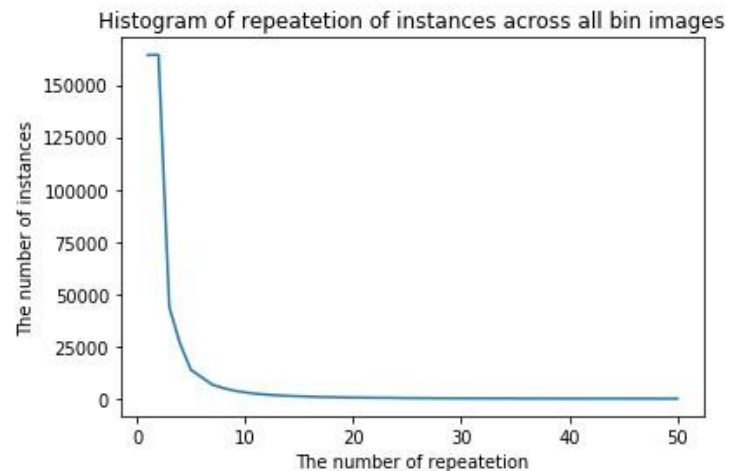
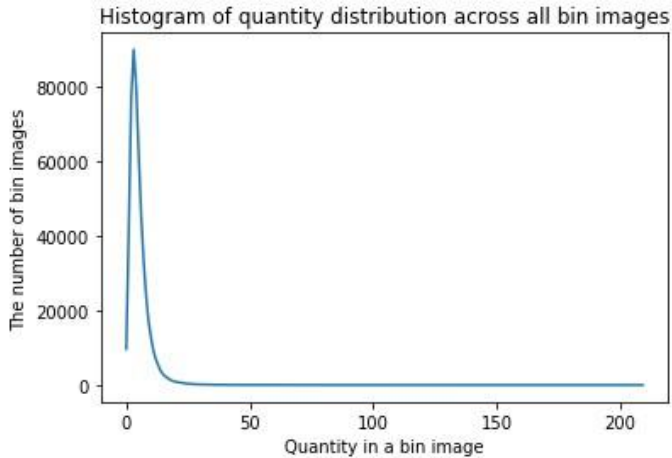


## II. Analysis

### Data Exploration

The Dataset contains over 500,000 images and metadata from bins of a pod in an operating Amazon Fulfillment Center. Images are located in the bin-images directory, and metadata for each image is located in the metadata directory.





Description	Total	Train	Validation
The number of images	535,234	481,711	53,523
Average quantity in a bin	5.1	5.11	5.08
The number of object categories	459,476	441,485	105,003

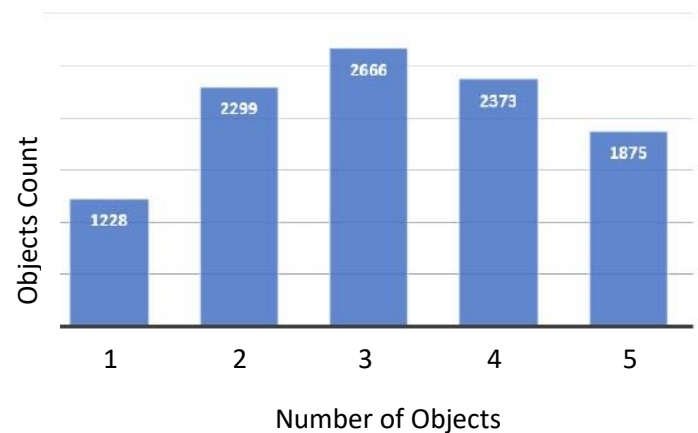
To download data using the AWS Command Line Interface, you can use the "cp" command. For instance, the following command will copy the image named 523.jpg to your local directory:

```
aws s3 cp s3://aft-vbi-pds/bin-images/523.jpg 523.jpg
```

In order to limit training time and AWS costs a JSON file with list of 10441 images divided into 5 classes was used.

Each class has items in range [1, 2, 3, 4, 5]:

- Class 1 with: 1228 images with 1 item in it.
- Class 2 with: 2299 images with 2 items in it.
- Class 3 with: 2666 images with 3 items in it.
- Class 4 with: 2373 images with 4 items in it.
- Class 5 with: 1875 images with 5 items in it.



## Exploratory Visualization

A visualization summary about the data is done by taking a random sample image from the dataset are presented in Figure below.

num items1  
sample102986.jpg



num items2  
sample10151.jpg



num items3  
sample05335.jpg



num items4  
sample101515.jpg



num items5  
sample08007.jpg



## Algorithms and Techniques

For the algorithm used to complete this project is *ResNet50 CNN* it is the best to use pretrained image classification network, which already is able to process and classify some image data, because of similar data structures, high results and fast training time with *PyTorch* models. As for techniques was to train the model after finding the best hyperparameter values, which is done by optimizing the hyperparameters before creating a training job, all that in *AWS Sagemaker* platform.

For that we, I have created AWS Sagemaker notebook instance before starting working on the project with the following setup:

The screenshot displays the AWS Sagemaker console interface. At the top, the navigation bar includes the AWS logo, a search bar, and service icons for Amazon SageMaker, S3, IAM, Lambda, and EFS. The breadcrumb trail indicates the path: Amazon SageMaker > Notebook instances > Object-Count-Tracking-Using-AWS-Sagemaker. The main heading is 'Object-Count-Tracking-Using-AWS-Sagemaker', with action buttons for Delete, Stop, Start, Open Jupyter, and Open JupyterLab. Below this is the 'Notebook instance settings' section, which includes an 'Edit' button and a table of instance details. The table lists the Name, Status (Stopping), Notebook instance type (ml.t3.medium), Platform identifier (Amazon Linux 2, Jupyter Lab 3), ARN, Creation time (Feb 09, 2023 04:41 UTC), Elastic Inference (none), Minimum IMDS Version (2), Last updated (Feb 09, 2023 17:50 UTC), Volume Size (5GB EBS), and Lifecycle configuration (none). The 'Git repositories' section shows a table with one repository: 'Object-Count-Tracking-Using-AWS-Sagemaker - (Default)' with the URL 'https://github.com/RamyGendy/Object-Count-Tracking-Using-AWS-Sagemaker.git' and Type 'Default'. The 'Permissions and encryption' section shows the IAM role ARN, Root access (Enabled), and Encryption key.

Name	Status	Notebook instance type	Platform identifier
Object-Count-Tracking-Using-AWS-Sagemaker	Stopping	ml.t3.medium	Amazon Linux 2, Jupyter Lab 3 (notebook-ai2-v2)

ARN	Creation time	Elastic Inference	Minimum IMDS Version
arn:aws:sagemaker:us-east-1:933845045900:notebook-instance/object-count-tracking-using-aws-sagemaker	Feb 09, 2023 04:41 UTC	-	2

Last updated	Volume Size
Feb 09, 2023 17:50 UTC	5GB EBS

Name	Repository URL	Type
Object-Count-Tracking-Using-AWS-Sagemaker - (Default)	https://github.com/RamyGendy/Object-Count-Tracking-Using-AWS-Sagemaker.git	Default

IAM role ARN	Root access	Encryption key
arn:aws:iam::933845045900:role/service-role/AmazonSageMaker-ExecutionRole-20230130T210045	Enabled	

## Benchmark

In the same domain, I found 2 contributions:

- Title: Amazon Bin Image Dataset Challenge:
  - URL: [https://github.com/silverbottlep/abid\\_challenge](https://github.com/silverbottlep/abid_challenge)
  - Author Name: silverbottlep
  - Author URL: <https://github.com/silverbottlep>

Title: Amazon Inventory Reconciliation using AI:

- URL: <https://github.com/OneNow/AI-Inventory-Reconciliation>
- Author Name: Pablo Rodriguez Bertorello, Sravan Sripada, Nutchapol Dendumrongsup
- Author URL: <https://github.com/pablo-tech>

The resulted accuracy of both contributions is 55% approximately with a RMSE of 0.94. I will be trying to challenge or provide similar accuracy.

### III. Methodology

#### Data Preprocessing

Since I used AWS Sagemaker platform to complete this project, then my steps in data preparation and preprocessing based on my data exploration are:

1. Fetch data from *Amazon* Database.
2. Select the data I will be working with through the help of *JSON* file.
3. Split the result into 3 subsets: Train, Test and Validation sets with ratios 60%, 20% and 20% respectively.
4. Upload the new dataset to *AWS S3 Bucket*.

#### Implementation

Once our data is uploaded, I started with `hpo.py` which executes just a single epoch on a part of training data before the real training job starts to find the best hyperparameters for the training job with the following ranges:

- ✓ Learning rate range: 0.001, 0.1
- ✓ Batch size values: 32, 64, 128, 256, 512

Hyperparameter tuning jobs:

The screenshot displays the Amazon SageMaker console interface. At the top, the navigation bar shows the AWS logo, 'Services', a search bar, and the user's profile 'voclabs/user2074122-5828682428 @ 9338-4504-5900'. The main header indicates the current path: 'Amazon SageMaker > Hyperparameter tuning jobs > pytorch-training-230209-0721'. A 'Stop tuning job' button is visible in the top right corner.

**Hyperparameter tuning job summary**

<b>Name</b> pytorch-training-230209-0721	<b>Status</b> Completed	<b>Approx. total training duration</b> 1 hour(s), 18 minute(s)
<b>ARN</b> arn:aws:sagemaker:us-east-1:933845045900:hyper-parameter-tuning-job/pytorch-training-230209-0721	<b>Creation time</b> Feb 09, 2023 07:21 UTC	
	<b>Last modified time</b> Feb 09, 2023 07:34 UTC	

**Training job status counter**

Completed 10 In Progress 0 Stopped 0 Failed 0 (Retryable: 0, Non-retryable: 0)

**Training jobs**

Sorting by objective metric value will display only jobs that have metric values. [Refresh] [View logs] [View instance metrics] [Stop] [Create model]

Search training jobs

	Name	Status	Objective metric value	Creation time	Training Duration
<input type="radio"/>	pytorch-training-230209-0721-010-b8ea65f5	Completed	1.6596095561981201	Feb 09, 2023 07:21 UTC	9 minute(s)
<input type="radio"/>	pytorch-training-230209-0721-009-e642f64e	Completed	13.012751579284668	Feb 09, 2023 07:21 UTC	8 minute(s)
<input type="radio"/>	pytorch-training-230209-0721-008-74918001	Completed	6.54473352432251	Feb 09, 2023 07:21 UTC	8 minute(s)
<input type="radio"/>	pytorch-training-230209-0721-007-2924c748	Completed	45.24698257446289	Feb 09, 2023 07:21 UTC	8 minute(s)
<input type="radio"/>	pytorch-training-230209-0721-006-d72c50c4	Completed	1.6978278160095215	Feb 09, 2023 07:21 UTC	8 minute(s)
<input type="radio"/>	pytorch-training-230209-0721-005-cdb0e388	Completed	1.697159767150879	Feb 09, 2023 07:21 UTC	7 minute(s)
<input type="radio"/>	pytorch-training-230209-0721-004-dae311c5	Completed	1.6355668306350708	Feb 09, 2023 07:21 UTC	8 minute(s)
<input type="radio"/>	pytorch-training-230209-0721-003-36bb93f6	Completed	1.7978156805038452	Feb 09, 2023 07:21 UTC	7 minute(s)
<input type="radio"/>	pytorch-training-230209-0721-002-789d5561	Completed	1.5889160633087158	Feb 09, 2023 07:21 UTC	7 minute(s)
<input type="radio"/>	pytorch-training-230209-0721-001-03113e74	Completed	1.6834334135055542	Feb 09, 2023 07:21 UTC	7 minute(s)

Feedback Language © 2023, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

After execution, found the best hyperparameters are:

- ✓ Learning rate: 0.005339291333527525
- ✓ Batch size: 32

The screenshot displays the AWS SageMaker console interface. At the top, the navigation bar shows the AWS logo, a search bar, and the user's location (N. Virginia) and account ID. The main header indicates the current page is 'pytorch-training-230209-0721' under 'Hyperparameter tuning jobs'. A 'Stop tuning job' button is visible in the top right corner.

**Hyperparameter tuning job summary**

Name pytorch-training-230209-0721	Status Completed	Approx. total training duration 1 hour(s), 18 minute(s)
ARN arn:aws:sagemaker:us-east-1:933845045900:hyper-parameter-tuning-job/pytorch-training-230209-0721	Creation time Feb 09, 2023 07:21 UTC	
	Last modified time Feb 09, 2023 07:34 UTC	

Below the summary, there are tabs for 'Best training job', 'Training jobs', 'Training job definitions', 'Tuning Job configuration', and 'Tags'. The 'Best training job' tab is selected, showing a 'Best training job summary' with a 'Create model' button. The summary states: 'This training job is the best training job for only this hyperparameter tuning job.' It lists the job name 'pytorch-training-230209-0721-002-789d5561', status 'Completed', objective metric 'Test Loss', and value '1.5889160633087158'.

**Best training job hyperparameters**

A search bar is provided above a table of hyperparameters:

Name	Type	Value
_tuning_objective_metric	FreeText	Test Loss
batch_size	Categorical	"32"
learning_rate	Continuous	0.005339291333527525
sagemaker_container_log_level	FreeText	20
sagemaker_estimator_class_name	FreeText	"PyTorch"
sagemaker_estimator_module	FreeText	"sagemaker.pytorch.estimator"
sagemaker_job_name	FreeText	"Amazon buckets HP search-2023-02-09-07-21-30-880"
sagemaker_program	FreeText	"hpo.py"
sagemaker_region	FreeText	"us-east-1"
sagemaker_submit_directory	FreeText	"s3://sagemaker-us-east-1-933845045900/Amazon buckets HP search-2023-02-09-07-21-30-880/source/sourcedir.tar.gz"

Note: hyperparameter tuning was performed to search the 2D parameter space. After that I started training my model with the best hyperparameters and deploying it to an endpoint on *AWS SageMaker Endpoints*. Where I used "ml.m5.xlarge" as training instance for fast computational power in computer vision applications.

The screenshot shows the AWS SageMaker console for an inference endpoint. The navigation bar is consistent with the previous screenshot. The main header shows the endpoint name 'pytorch-inference-2023-02-09-10-04-02-393' and a 'Delete' button.

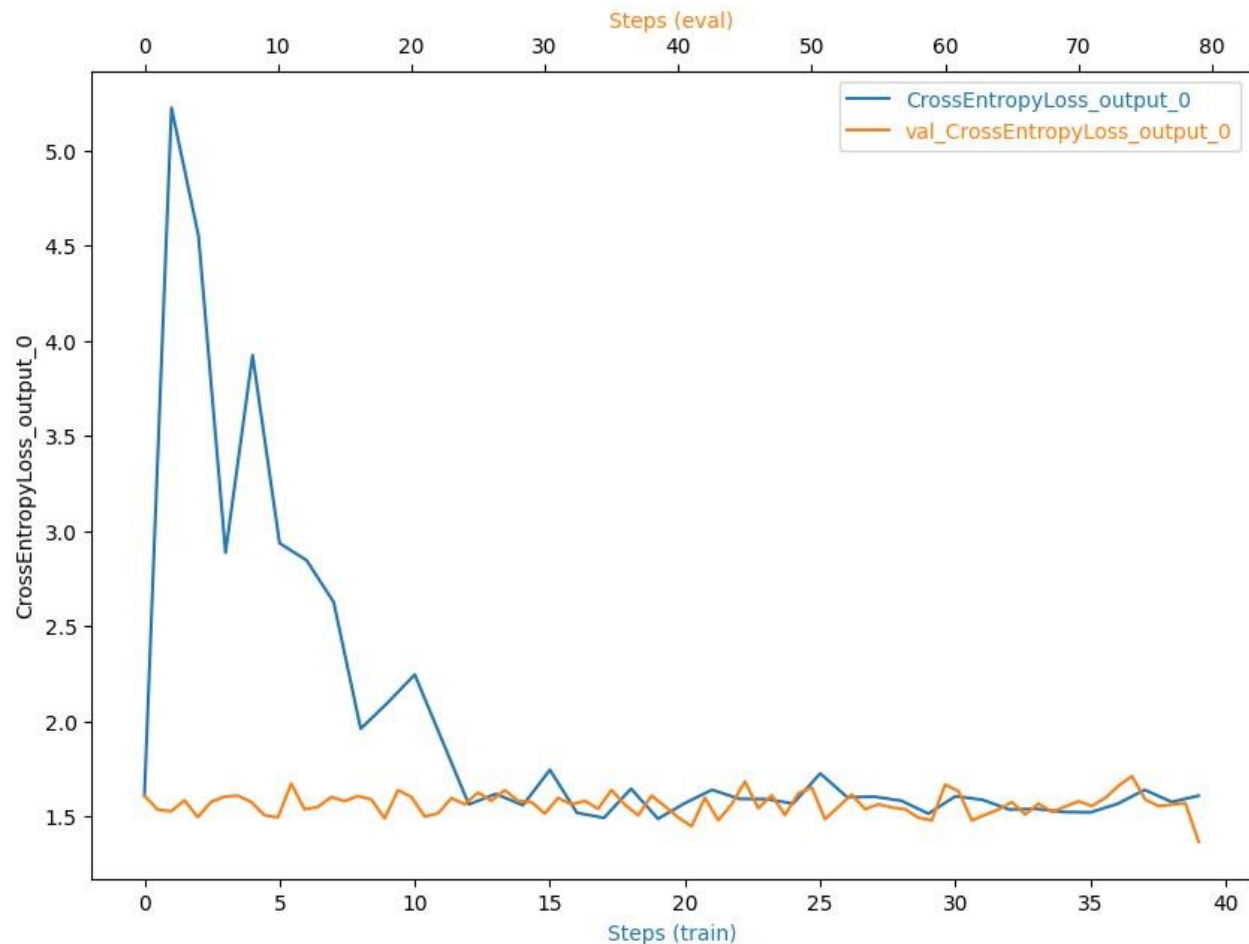
**Endpoint settings**

Name pytorch-inference-2023-02-09-10-04-02-393	Status InService	Type Real-time	URL <a href="https://runtime.sagemaker.us-east-1.amazonaws.com/endpoint/pytorch-inference-2023-02-09-10-04-02-393/invocations">https://runtime.sagemaker.us-east-1.amazonaws.com/endpoint/pytorch-inference-2023-02-09-10-04-02-393/invocations</a> <a href="#">Learn more about the API</a>
ARN arn:aws:sagemaker:us-east-1:933845045900:endpoint/pytorch-inference-2023-02-09-10-04-02-393	Creation time Thu Feb 09 2023 12:04:03 GMT+0200 (Eastern European Standard Time)	Last updated Thu Feb 09 2023 12:06:54 GMT+0200 (Eastern European Standard Time)	



## IV. Results

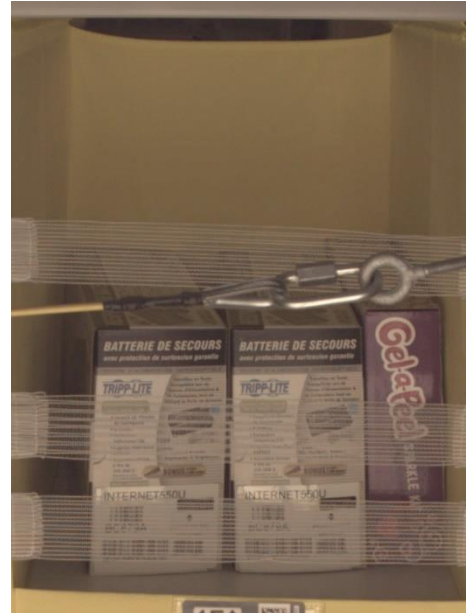
### Model Evaluation and Validation



For the model inference deployment, I have created "inference.py" script, which can be used for creating service functions like *AWS Lambda* or *Sagemaker*. After deploying the model to the endpoint, I started evaluation it with my metrics (Cross Entropy Loss Function) as in the above figure.

Then tested the model operation on the test image below

Where it is clear that the bin has 3 objects.



Once successful pass of the test image to the predictor, I received the below response:

```
labeled_predictions = dict(zip(object_count_range, response[-1]))
print("Labeled predictions: ", labeled_predictions)
```

```
Labeled predictions: {1: 0.019855573773384094, 2: 0.34309279918670654, 3: 0.7109078168869019, 4: 0.6135687828063965, 5: -0.08671106398105621}
```

```
print("Most likely answer: {} with probability {}".format(max(labeled_predictions, key=labeled_pred
```

```
Most likely answer: 3 with probability 0.7109078168869019
```

```
print("Predictor Answer:", max(labeled_predictions, key=labeled_predictions.get))
```

```
Predictor Answer: 3
```

Where the model has successfully predicted the right count.

## V. Conclusion

In conclusion, implementing the hyperparameter tuning step was important and performed properly where the selected hyperparameter values are acceptable for further development even though the pre-trained ResNet50 model suits my data perfectly but the final accuracy of the trained model is about 0.25, which is considered relatively low however the fact that only a small quantity of the Amazon Dataset with the limited the training process finished in about 20 mins with achieved accuracy is acceptable.

Even though the model is not ready for real time application but it can act as a verification tool for other complex networks.

## Improvement

With the current aspect of implementation and designed, I would improve:

- ✓ the training dataset on a bigger part of the dataset should be performed.
- ✓ Comparison with the additional pretrained models and compare performance.
- ✓ As an example, consider ways your implementation can be made more general, and what would need to be modified.
- ✓ The use of multi-instance training would help in decreasing the training time or the capacity of handling the training data size.