Capstone Project
Machine Learning Engineer
Nanodegree

Khang Hoang Tran
July 24th, 2024

Inventory Monitoring at Distribution Centers

I. Definition

Project Overview

In the logistics domain, efficient inventory management and accurate order fulfillment are critical for maintaining operational efficiency and customer satisfaction. Many activities in warehouse these days involve much human resource. For large-scaling business, instantly increasing labor force might produce a greate cost with a potential risk of staff management.

As a result, the rapid growth of e-commerce and the increasing complexity of supply chains have heightened the need for advanced technological solutions to streamline these processes. One such solution has been popular adopted is leveraging technologies like machine learning techniques to automate and enhance inventory monitoring, recognizing and sorting tasks.

Problem Statement

In Amazon warehouse scenario, robots are used to move objects as a part of their operations. Objects are carried in bins which can contain multiple objects. For each customer's order, we need to ensure that delivery consignments have the correct number of items. Amazon robotics, scanning machines, and computer systems in fulfilment centres can track millions of items in a day. A tracking inventory system with only manual checks are labor-intensive.

This project focuses on utilizing the "Amazon Bin Image Dataset" to develop a machine learning models capable of recognizing and specifying the number of objects in each bin. The dataset comprises a diverse collection of images captured in a warehouse setting, depicting various items placed in bins. By applying advanced image recognition and classification algorithms, the goal is to create a robust system that can accurately identify

products, manage inventory levels, and optimize order picking processes. Also, the output model aims to reduce manual labor, minimize errors, increase overall efficiency; and most important, simulate a full machine learning pipeline in a logistic data-processing job.

Metric

In the field of image classification, the accuracy metric is a fundamental measure used to evaluate the performance of a model. Accuracy is defined as the ratio of the number of correctly predicted instances to the total number of instances. It is expressed as a percentage, indicating how often the classifier correctly identifies the label of the input image. A high accuracy value indicates that the model is performing well in correctly classifying the images in general.

Formula for Accuracy

The accuracy metric is calculated using the following formula:

$$Accuracy = \frac{True\ positives + True\ Negatives}{True\ positives + False\ positives + True\ Negatives + False\ Negatives}$$

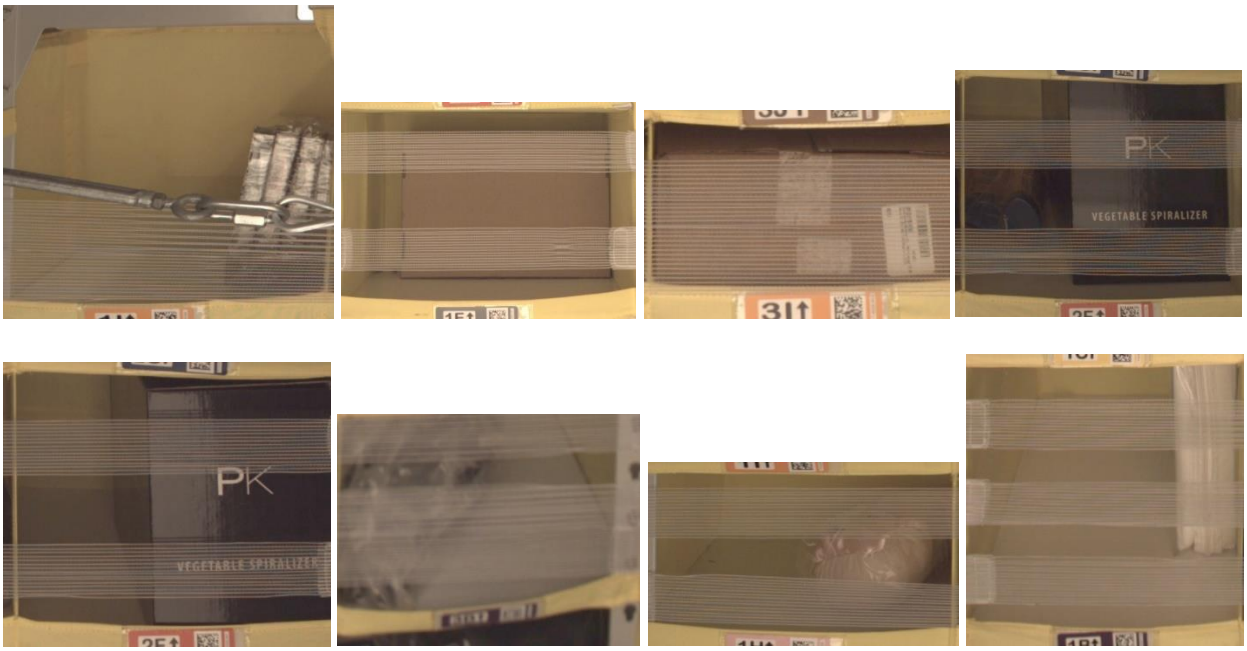
II. Analysis

Data Exploration

The Amazon Bin Image Dataset 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.

As for a large dataset, we plan to use only a subset of **10441** samples from the original dataset which is served as experiment tasks to examine the efficiency of the model. The subset only contains bin images that store the number of items between '1' and '5'.

Sample of the images.



The subnet data have 5 subfolders corresponding to 5 labels, each subfolder contains images. In regard to build the model, we divide our dataset into train, test and validation set. Our final dataset structure looks like this

```
+---bin-images
    +---train
    |   +---1
    |   |       00014.jpg
    |   |       00024.jpg
    |   |       ...
    |   +---2
    |   |       00056.jpg
    |   |       00112.jpg
    |   |       ...
    #   ...
    |   +---5
    |           00006.jpg
```

```
    |           00058.jpg
    +---test
    ... (same as above)
    +---validation
    ... (same as above)
```

For each image there is a metadata file containing information about the image like the number of objects, it's dimension and the type of object. An image, i.e *00014.jpg* will have metadata file *00014.json*.

Sample of the *metadata*


```
{
    "BIN_FCSKU_DATA": {
        "B0123H9HME": {
            "asin": "B0123H9HME",
            "height": {
                "unit": "IN",
                "value": 3.49999999643
            },
            "length": {
                "unit": "IN",
                "value": 9.599999990208
            },
            "name": "Organix Grain Free Lamb & Peas Recipe, 4 lb",
            "normalizedName": "Organix Grain Free Lamb & Peas Recipe,
4 lb",
            "quantity": 1,
            "weight": {
```

```
                "unit": "pounds",
                "value": 4.549999996184413
            },
            "width": {
                "unit": "IN",
                "value": 8.099999991737999
            }
        }
    },
    "EXPECTED_QUANTITY": 1
}
```


We have got one of the problems dealing with this dataset. Some small boxes are inside other larger boxes which can disrupt our predictions. So for an object recognition task, we will need to clean the dataset. Since we already have *file_list.json*, we are allowed to proceed further without spending time with initial data. So the metadata is just for references or if we want to collect more samples for our dataset.

Exploratory Visualization

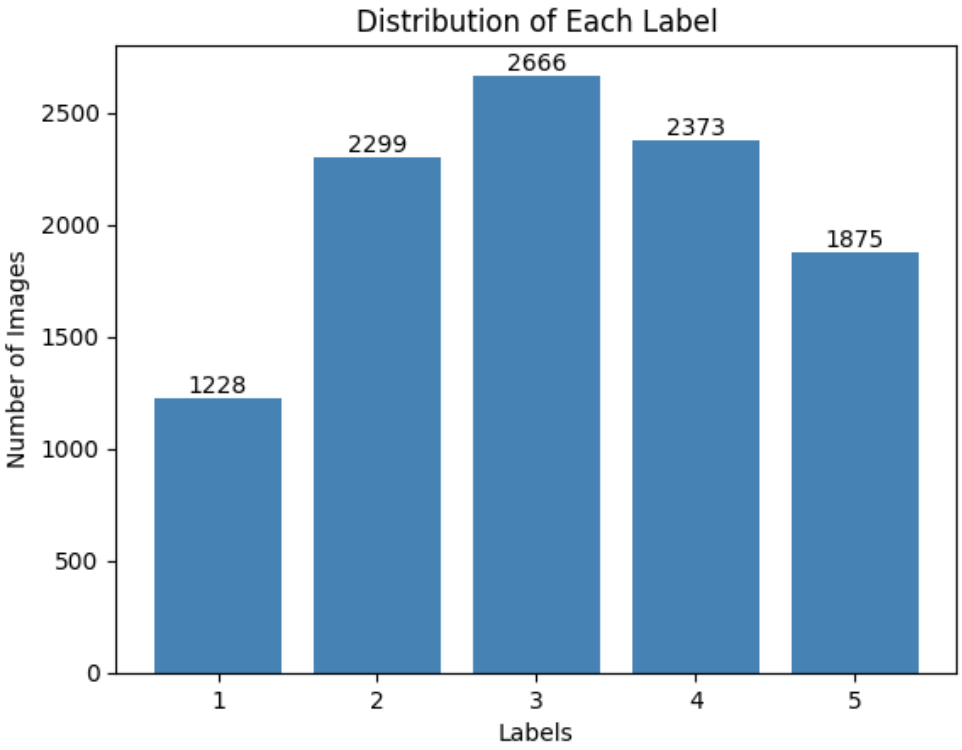
Amazon bin image dataset

The table below shows the average number of objects in a bin(round the result to two decimal places) , note that our target number is in range between 1 and 5.

Table statistic:

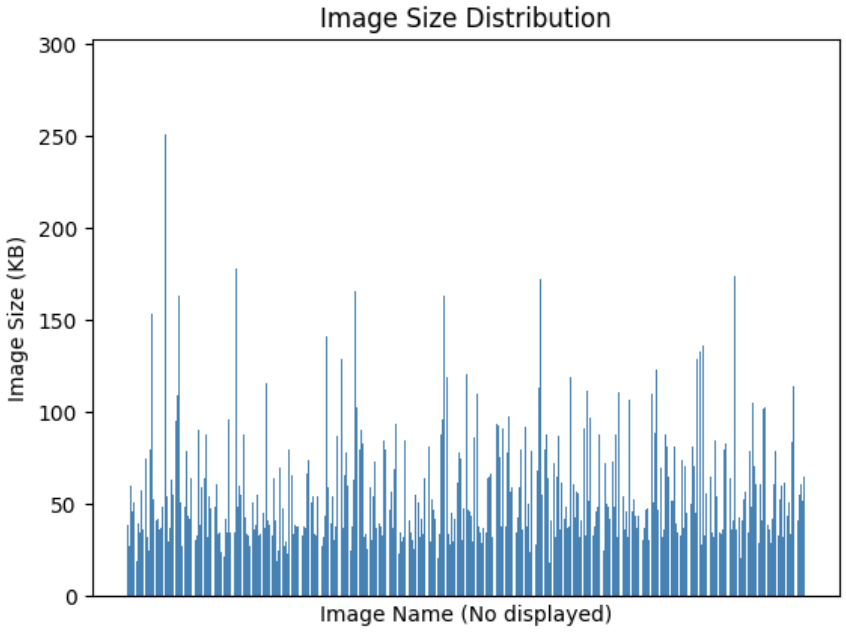
Description	Total
The total of images in dataset	10441
The average number of objects in each bin	3.13

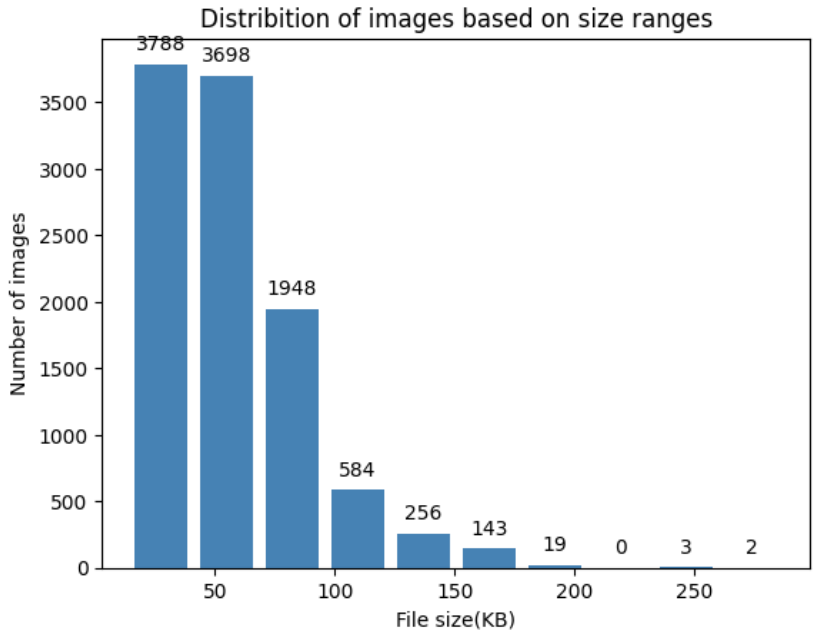
The bar chart below shows the distribution of images in each target labels



The mean value ‘3.13’ and the chart shows that the distribution of our data is moderately even. The label ‘3’ has the highest figures and those of other labels are slightly lower. With this data, we found it appropriate for the model to train and evaluate the result.

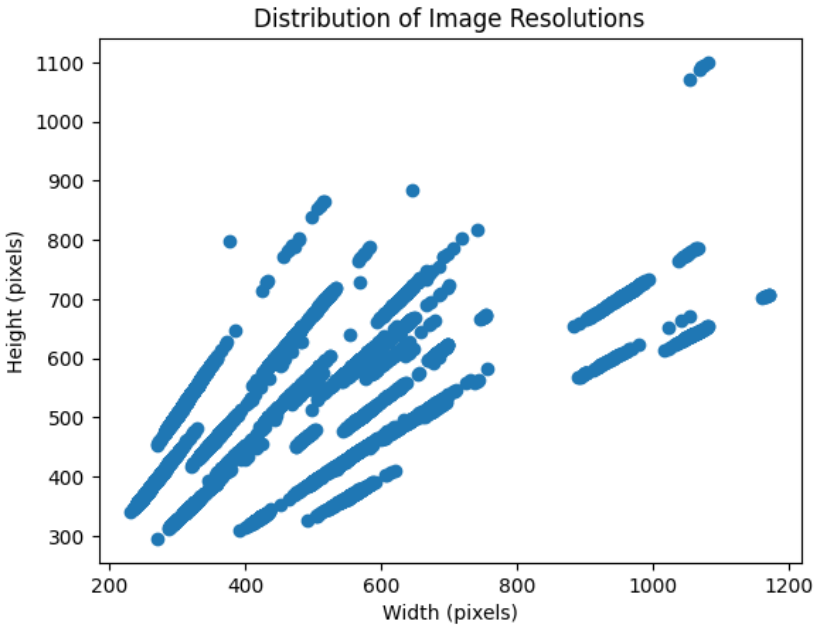
The bar chart below summarizes the distribution of image’s size.





From the chart, we observe that there is significant difference between in size among images, also it follows certain patterns. These sizes are divided into multiple ranges, most of images are about above 50KB and below 100KB in ranges, a few of images are even over 250KB. As the result, we should resize these images into smaller size for model training.

A distribution chart of image's resolution for our dataset insight



The bin images are shaped fairly squarely. Before feeding data to model, we should resize our images into fixed resolution. Ideally, we often choose the same for width and height.

Algorithms and Techniques

ResNet50, a 50-layer deep convolutional neural network, has a robust architecture that excels in handling large-scale image data with high accuracy. It functions similarly to a highly skilled image analyst who is able to analyze a photograph, recognize the scenes and items it contains, and classify them appropriately.

ResNet architectures come in various depths, such as ResNet-18, ResNet-32, and so forth, with ResNet-50 being a mid-sized variant. For this project, we adopt pretrained model **Resnet50** to fine-tune on the amazon dataset.

Since it is a pretrained model, it has similar rules to other kinds. The training paradigm below will describe step-by-step instructions:

1. Create and load the model from library.
2. Freeze all the convolutional layers, skip gradient adjustments for backpropagation in each epoch.
3. Add a custom fully-connected layers and then train it.

With all these steps, our model will skip update weight for backwarding process in layers in pretrained model. It only take actions with both forwarding and backwarding in our customized fully-connected layers.

Our hyperparameter tuning procedure uses the below parameters:

- Number of epochs: The number of iterations for model learning.
- Batch size: The number of images are taken at once into model for each training process.
- Learning rate: How quickly model could learn.

As a image classification problem, we use **CrossEntropyLoss** for loss function and **Adam** for optimizer.

Benchmark

The project will be benched marked using accuracy as we determined since it is a classification problem. Our target is to achieve an accuracy of above **55.67%** on validation

set **using Deep Learning models** which is the final and accepted value in the Amazon Bin image dataset challenge by silverbottlep

III. Methodology

Data Preprocessing

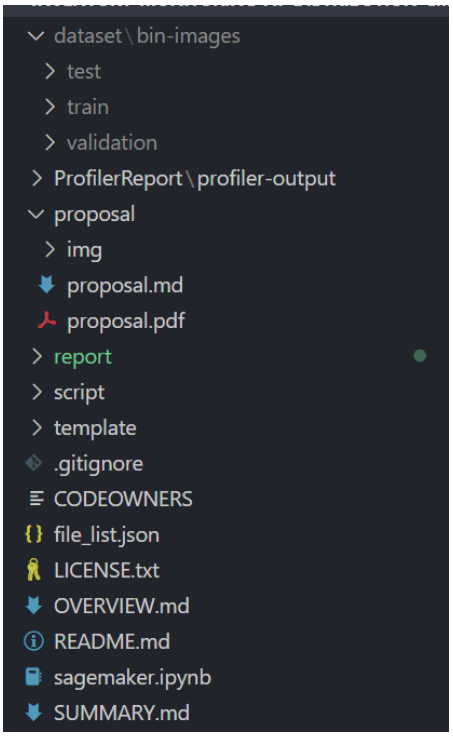
We split our dataset into train, test and validation set, with the ratio of **6:2:2** respectively.

From the illustrated chart, we do the normalization on each image before pushing those into our training model. We transform an arbitrary image into **224x224** resolution.

Implementation

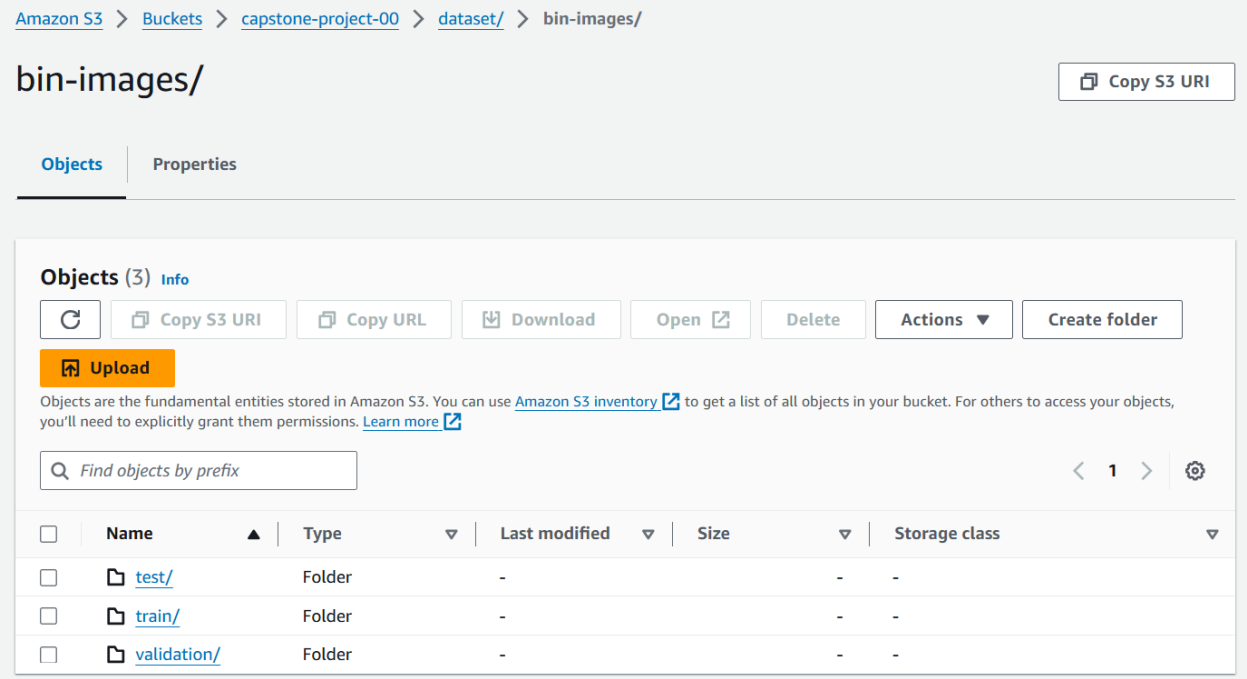
Folder structure

Developing space has structure like this:

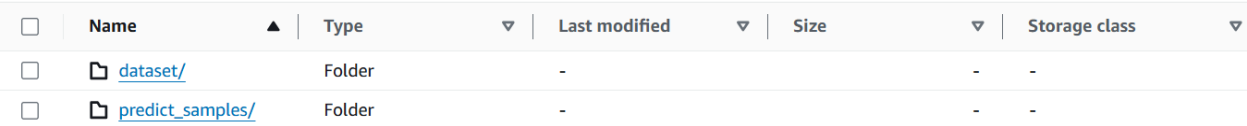


Dataset

For dataset storage, we upload our Amazon images to **Amazon S3 buckets**.



We also upload a few of image for predictions.



For publicly access, we adjust bucket policy for project bucket **capstone-project-00.** So we can predict testing image via HTTPS URL.

Jupyter development environment

Regard to development environment, we use *Jupyter Notebook* in **Sagemaker Studio**. To create a workspace for develop project, we perform the following action:

1. Create Sagemaker domain with quickly setup, specific user and specific role.
2. Open Studio, in JupyterLab, create a space with the default specs:

Name	Value
Instance	ml.t3.medium
Image	Sagemaker Distribution 1.9

Storage	5(GB)
Lifecycle Configuration	No script
Attach custom EFS filesystem	None

3. Open space then clone the starter Git repository, we are able to develop our script.

Python script files

There are three scripts:

- **hpo.py**: For hyperparameter tuning job, focus on training only, no redudant libraries or functions.
- **train.py**: Focus on training process with the best suite of parameters from tuning process above that reach the max value of defined metric. Also there are some integration with debugging and profiling hook to get the insight data from training.
- **Inference.py**: Used to deploy endpoint for inferences with multiple necessary functions that its structure are defined by Sagemaker. We handle and process data in this file.

Submission notebook (sagemaker.ipynb)

The final Jupyter notebook has numerous cells that would interact with Sagemaker and submit jobs to run. There are also output while developing process that can be used for referencing. Below is the main tasks which are conducted:

- Preconfig
    - Install & import modules
    - Configure environment and initialize variables
- Data preparation
    - Download and split dataset
    - Upload to S3 bucket
    - Sync data between local and cloud for local training job(for debug purpose only)
- Data visualization
    - Visualize data with multiple charts
    - Deliver data insight
- Hyperparameter tuning

    - Define target metric definition
    - Define tuning parameters
    - Run hyperparameter job, get the best suite
- Model training
    - Configure hook to **train.py**. Submit training job
    - Add an optional choice for multi-instance training
- Model profiling and debugging
    - Show charts, statistics and information for training debug
    - Show performance profile report for training process.
- Model deploying and querying
    - Deploy model as endpoint with **inference.py**
    - Get sample data image, view image, infer that image and get the result

Refinement

In this project, we create a pipeline workflow using AWS Sagemaker to improve the repeated process. The tuning, training and deploying process run with minimal code changes. We can apply this process for larger dataset and develop further for large-scale enterprise environments. As a result, we can achieve better performance in model.

```
hyperparameter_ranges = {
    "lr": ContinuousParameter(0.001, 0.1),
    "batch-size": CategoricalParameter([16, 32, 64, 128, 256, 512]),
    "epochs": IntegerParameter(10, 20)
}
```

The best hyperparameters are chosen by Bayesian algorithm:

```
{'lr': '0.0011353714970396942', 'batch-size': 128, 'epochs': 19}
```

Model prediction with **numpy.argmax()**

```
array([3])
```

**The same inferenced result is returned regardless of the number of the objects in the bin

IV. Results

Model Evaluation and Validation

The final output model has been developed with the best performing hyperparameters. We had utilized pretrained Pytorch model to serve our goal.
After successfully deploying the endpoint for incoming requests, we indirectly call the **predict()** method from the fully-trained model.

A screenshot from endpoint query:

[34]:
```
from PIL import Image
import io
Image.open(io.BytesIO(img_bytes))
```

[34]:

[35]:
```
response = predictor.predict(img_bytes, initial_args = {"ContentType": "image/jpeg"})
```

View list of probabilities of each label

[36]:
```
response
```

[36]:
```
[[3.5001468658447266,
  4.805016040802002,
  4.875925064086914,
  4.884450912475586,
  3.5118863582611084,
  -6.081723213195801,
  -5.4884724617004395,
  -7.477519512176514,
  -6.24439001083374,
  -7.1445393562316895,
  -7.50557279586792,
  -6.855061054229736,
  -7.080506801605225,
  -6.566812038421631,
  -7.166322231292725,
  -6.736094951629639,
  -6.855122089385986,
  -6.974879264831543]]
```

[37]:
```
np.argmax(response, 1)
```

[37]:
```
array([3])
```

The model prediction result for this image from **test set** is incorrect, the accuracy percentage is low. When querying with different images, all of the images are received the same response result.

Justificatiion

Trained on small subset from original dataset. The model reachs the accuracy of approximately **32%** on test set

```
Testing Model
Test set: Average loss: 1.4684, Accuracy: 675/2089 (32%)
Testing Total Loss: 143.000, Testing Accuracy: 32.000%
Saving Model
2024-07-24 06:48:50,396 sagemaker-training-toolkit INFO     Reporting training SUCCESS

2024-07-24 06:49:21 Uploading - Uploading generated training model
2024-07-24 06:49:21 Completed - Training job completed
Training seconds: 6995
Billable seconds: 6995
```

Although the general idea in our workflow seems to be robust, the current model has limitations when performing with a small dataset like this. Also we need to explore, research and reach for a more suitable CNN models along with appropriate hyperparameters

V. Conclusion

Sagemaker provides a vigorous platform for developing machine learning pipelines. There are some prominent features like debugger and profiler in its supported SDK to analyse the model training process. We also can store features in centralized place after extracting features. We could integerate AWS services to define a complete pipeline, force to train model after more dataset added to S3 bucket. Besides, there are separate sub-services like storing model for reuse and redeploy in the future, manage deployed endpoints, training job and more.

Along with Sagemaker, we also have solutions from various cloud providers such as Azure or GCP. The option depends on the scenarios and context we are applying and on which ecosystem we are focusing, the benefit we gain is from services that come along with the same cloud provider.

Also, to build an ML model, some key factors could be realized when having many experiment with this project are that the quantity of the dataset and the applied model architecture. The choices of loss functions, the optimizer, parameters and the final metrics

to be evaluated. These are the key points that play important roles to assess the accomplished model.

Resources and references:

1. Amazon bin image dataset distribution center: Amazon Bin Image Dataset
2. Dataset Challenge 1: Amazon Inventory Reconciliation using AI by pablo-tech
3. Dataset Challenge 2: Amazon Bin Image Dataset(ABID) Challenge by silverbottlep
4. Metrics to evaluate classification models
5. Set up hook for debugging: sagemaker-debugger/docs/pytorch.md at master · awslabs/sagemaker-debugger (github.com)
6. Set up spot instance for saving cost: How to Enable Spot Instances for Training on Amazon SageMaker (missioncloud.com)

Thank you