Udacity AWS Machine Learning Engineer Nanodegree

Capstone Project Report

INVENTORY MONITORING AT DISTRIBUTION CENTERS

1. DEFINITION

Project Overview

Robots are used in distribution centers for moving objects as a part of their operations and those objects which are carried in bins can consist of a lot of objects. In this project, we are addressing the problem of inventory monitoring at distribution centers by developing a Deep Learning (DL) model that is capble of automatically counting the number of objects in bins and, thus, providing accurate inventory tracking information to decision makers for optimizing their operations.

To develop this Capstone Project, I am going to use AWS SageMaker to develop an end-to-end solution from obtaining a public dataset from its database to training and deploying a DL model using the dataset.

Project Statement

The main objective of this Capstone Project is to count the number of objects in bins via images. In other words, this is a Computer Vision problem to detect multiple objects in images, and then count the number of objects.

Besides, to complete this Project, we will be using the Amazon Bin Image Dataset [1]. So we need to perform the data ingestion, data processing, model training and deployment based on this public dataset to solve the problem.

Metrics

In this Project, the Cross Entropy function is used as the loss function for training and testing the model, defined as below:

$$H(X) = -\Sigma_i P(X=i) log_2 P(X=i)$$

2. ANALYSIS

Data Exploration

As mentioned above, the Amazon Image Bin Dataset will be used in this Capstone Project. This dataset contains more than 500K images (in JPEG format) and their corresponding labels (in JSON format) collected from bins of a pod in an operating Amazon Fulfillment Center.

The data can be accessed by using the below command line with no AWS account required:

```
aws s3 ls --no-sign-request s3://aft-vbi-pds/
```

Exploratory Visualization

Images are located in the `bin-images` directory while the corresponding labels are located in the `metadata` directory. For example, the metadata for the image at https://aft-vbi-pds.s3.amazonaws.com/bin-images/523.jpg (Fig. 1) is found at https://aft-vbi-pds.s3.amazonaws.com/metadata/523.json

Fig. 1. An image sample from Amazon Image Bin Dataset.

In particular, each bin image may show only one or more types of products. In this Project, we have utilized a total of 10441 images with the class distributions as below:

- 1 item in bin: 1228 images
- 2 item in bin: 2299 images
- 3 item in bin: 2666 images
- 4 item in bin: 2373 images
- 5 item in bin: 1875 images

More details on Amazon Image Bin Dataset can be found: https://github.com/awslabs/open-data-docs/tree/main/docs/aft-vbi-pds.

Algorithms and Techniques

To address the problem in this Capstone Project, we will firstly fetch the Amazon Bin Image Dataset from its database, and store the dataset using Amazon Simple Storage Service (Amazon S3). The dataset will be preprocessed and split into training, validation and test sets with a predefined split ratio, which will be ready for training the DL model. In this Project, Amazon SageMaker will be used to build the DL workflow.

Residual Networks, or ResNets [2], are Convolutional Neural Networks (CNNs) that learn residual functions with reference to the layer inputs, instead of learning unreferenced functions. Instead of hoping each few stacked layers directly fit a desired underlying mapping, residual nets let these layers fit a residual mapping. They stack residual blocks ontop of each other to form network. Due to its high performance in image classification, in this Capstone Project, we will employ and fine tune a pretrained ResNet-50 with 50 layers deep (Fig. 2) for detecting objects in the images.

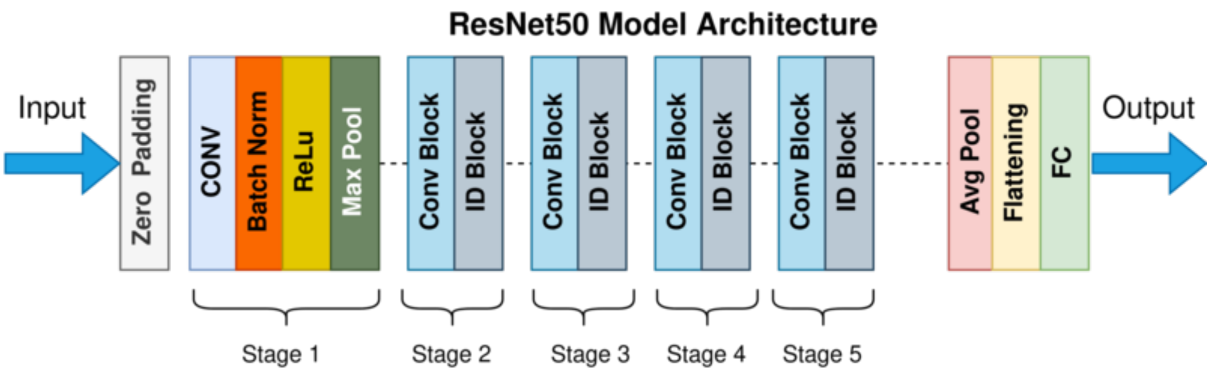


Fig. 2. ResNet-50 Architecture.

Benchmark

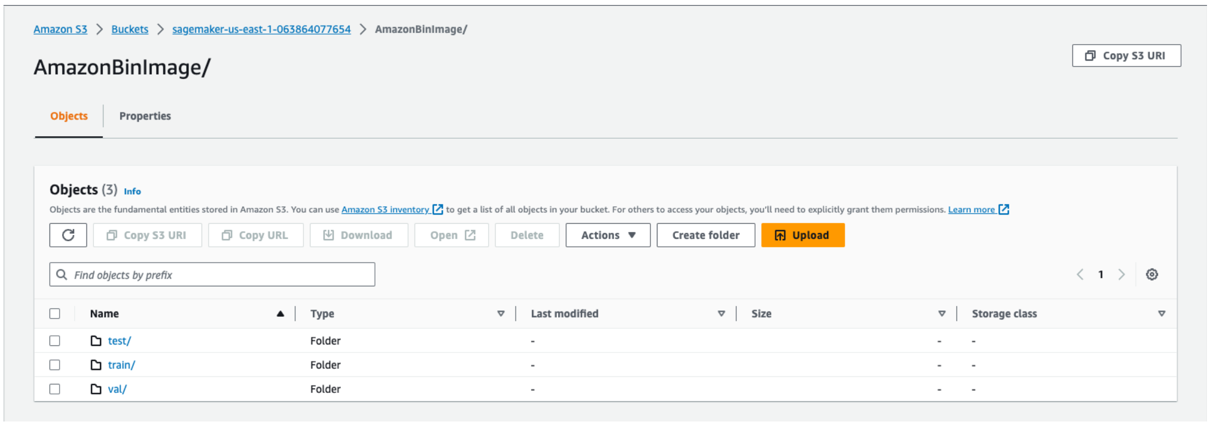
To benchmark the proposed approach, the input dataset will be split into three subsets: training, validation and test sets with a predefined split ratio: 0.6, 0.2 and 0.2. In other words, 6265 images are used for fine-tuning the pretrained ResNet-50 model, 2088 images are used for validating the fine-tuning process while the remaining 2088 images are used for testing the performance of the trained model.

In this Capstone project, we also employ SageMaker Hyperparameter Tuner to perform hyperparameter tuning, so that the model can get the optimal hyperparameters during the fine-tuning process. SageMaker Debugger and Profiler are also applied to detect any anomaly while training the model.

3. METHODOLOGY

Data Preprocessing

- Amazon Image Bin Dataset is fetched from the database *s3://aft-vbi-pds/.*
- The dataset is split into training, validation and test sets, which are used to fine tune the model in the next step. The split ratio is 60% train, 20% validation and 20% test. In other words, 6265 images are used for fine-tuning the pretrained ResNet-50 model, 2088 images are used for validating the fine-tuning process while the remaining 2088 images are used for testing the performance of the trained model.
- The data after preprocessing are uploaded to AWS S3.


Amazon S3 > Buckets > sagemaker-us-east-1-063864077654 > AmazonBinImage/

AmazonBinImage/

Objects (3) Info

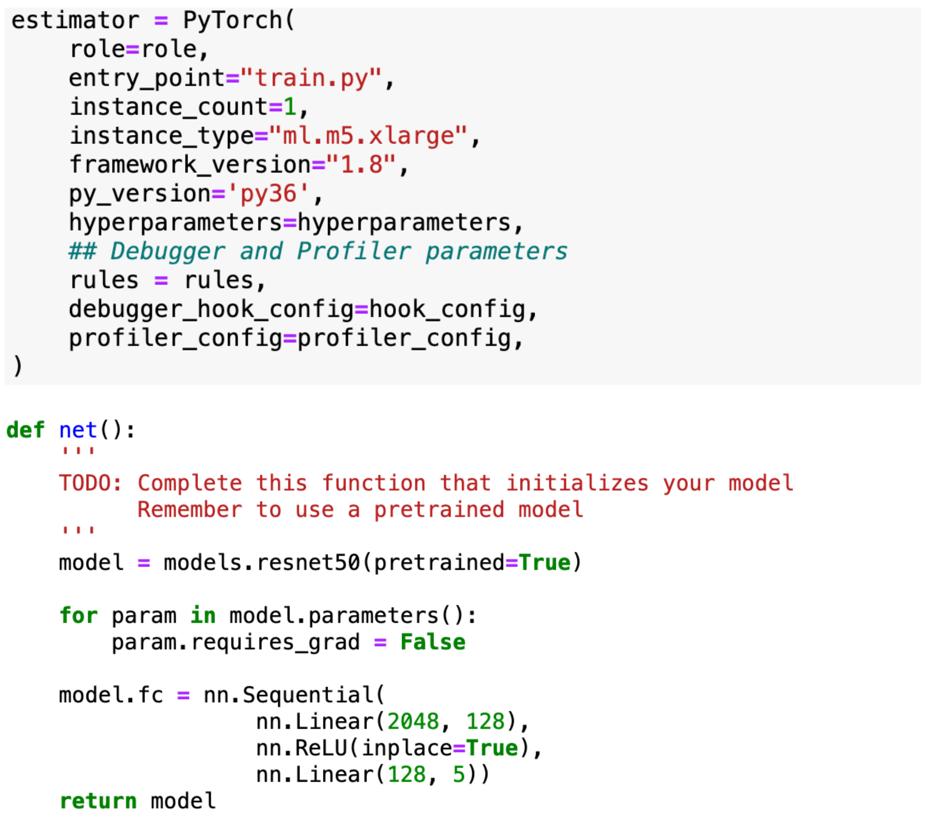
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
test/	Folder	-	-	-
train/	Folder	-	-	-
val/	Folder	-	-	-

Fig. 3. Upload the preprocessed data to AWS S3.

Implementation

- Amazon SageMaker is used to fine tune a pretrained ResNet-50 CNN model.



```
estimator = PyTorch(
    role=role,
    entry_point="train.py",
    instance_count=1,
    instance_type="ml.m5.xlarge",
    framework_version="1.8",
    py_version='py36',
    hyperparameters=hyperparameters,
    ## Debugger and Profiler parameters
    rules = rules,
    debugger_hook_config=hook_config,
    profiler_config=profiler_config,
)
```

```
def net():
    '''
    TODO: Complete this function that initializes your model
          Remember to use a pretrained model
    '''
    model = models.resnet50(pretrained=True)

    for param in model.parameters():
        param.requires_grad = False

    model.fc = nn.Sequential(
                   nn.Linear(2048, 128),
                   nn.ReLU(inplace=True),
                   nn.Linear(128, 5))
    return model
```

Fig. 4. Implementation of ResNet-50 model fine-tuning.

- Cross Entropy is used as the loss function during the model training and testing processes.
- Adam is used as the network optimizer.
- The model hyperparameters including batch_size and learning_rate are optimized using SageMaker Hyperparameter Tuner, mentioned in details in the next section.
- SageMaker Debugger is used to track the training loss and evaluation loss.
- After training, the fine-tuned model with the optimal hyperparameters is then deployed to a SageMaker Endpoint for further testing.

More details can be found in the train.py and sagemaker.ipynb.

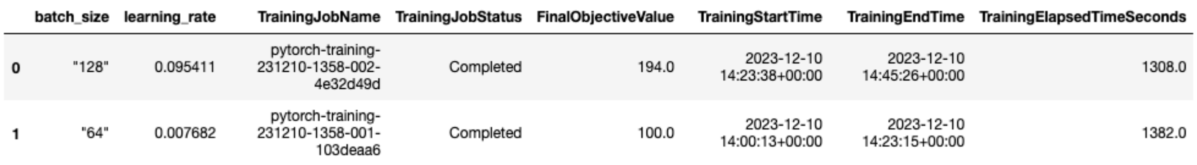
Refinement

Hyperparameter optimization:

- SageMaker Hyperparameter Tuner is used to optimize the model hyperparameters during the fine tuning process, thus improving the accuracy of the trained model.
- Hyperparameter search space:

```
hyperparameter_ranges = {
    "learning_rate": ContinuousParameter(0.001, 0.1),
    "batch_size": CategoricalParameter([32, 64, 128, 256, 512]),
}
```

- The comparison of different model hyperparameter sets in terms of Cross Entropy loss (smaller is better):



	batch_size	learning_rate	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTime	TrainingElapsedTimeSeconds
0	"128"	0.095411	pytorch-training-231210-1358-002-4e32d49d	Completed	194.0	2023-12-10 14:23:38+00:00	2023-12-10 14:45:26+00:00	1308.0
1	"64"	0.007682	pytorch-training-231210-1358-001-103deaa6	Completed	100.0	2023-12-10 14:00:13+00:00	2023-12-10 14:23:15+00:00	1382.0

Fig. 5. Hyperparameter optimization using SageMaker Hyperparameter Tuner.

The best hyperparameters are: {'batch_size': 64, 'learning_rate': '0.007682083391454922'}.

4. RESULTS

Model Evaluation and Validation

- SageMaker Debugger and Profiler are also applied to detect any anomaly while training the model

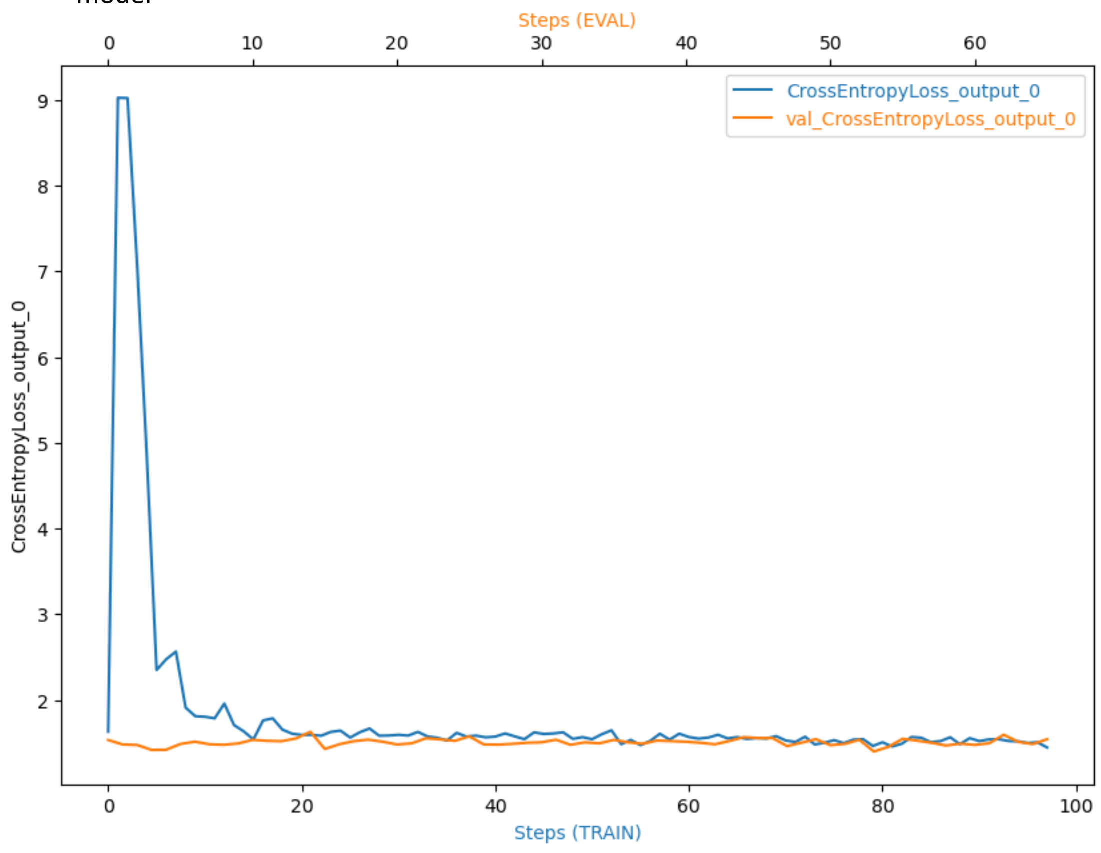
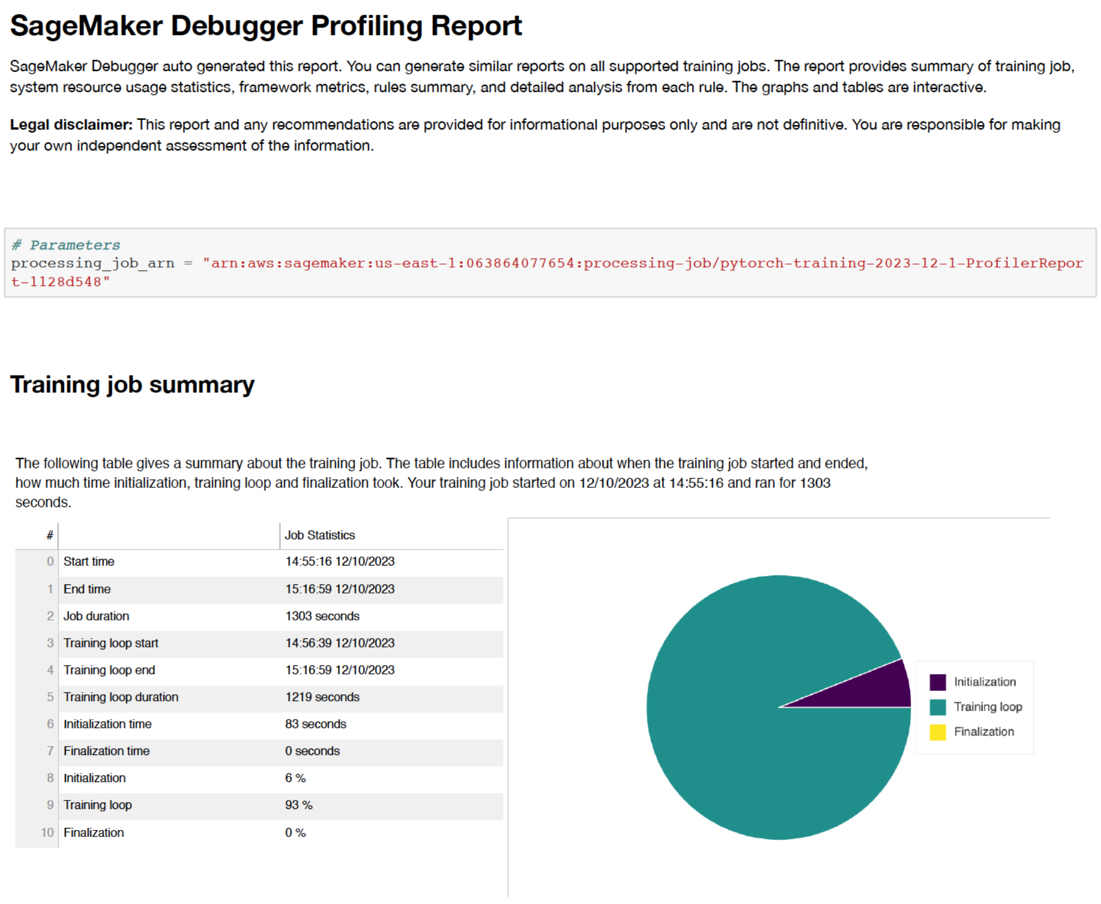


Fig. 6. Training and evaluation debugging using SageMaker Debugger.



SageMaker Debugger Profiling Report

SageMaker Debugger auto generated this report. You can generate similar reports on all supported training jobs. The report provides summary of training job, system resource usage statistics, framework metrics, rules summary, and detailed analysis from each rule. The graphs and tables are interactive.

Legal disclaimer: This report and any recommendations are provided for informational purposes only and are not definitive. You are responsible for making your own independent assessment of the information.

```
In [4]: # Parameters
        processing_job_arn = "arn:aws:sagemaker:us-east-1:063864077654:processing-job/pytorch-training-2023-12-1-ProfilerRepor
        t-1128d548"
```

Training job summary

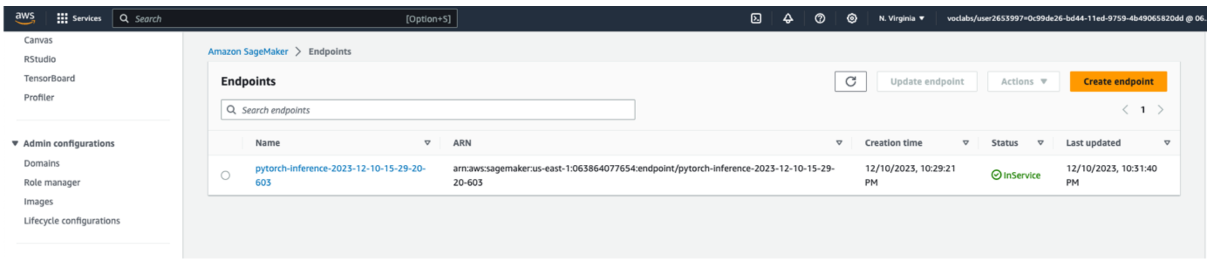
The following table gives a summary about the training job. The table includes information about when the training job started and ended, how much time initialization, training loop and finalization took. Your training job started on 12/10/2023 at 14:55:16 and ran for 1303 seconds.

#		Job Statistics
0	Start time	14:55:16 12/10/2023
1	End time	15:16:59 12/10/2023
2	Job duration	1303 seconds
3	Training loop start	14:56:39 12/10/2023
4	Training loop end	15:16:59 12/10/2023
5	Training loop duration	1219 seconds
6	Initialization time	83 seconds
7	Finalization time	0 seconds
8	Initialization	6 %
9	Training loop	93 %
10	Finalization	0 %

Fig. 7. SageMaker profiling report for the model training.

More details on the Profiler report can be found in the attached profiler-report.html or profiler-report.pdf.

- The fine-tuned model with the optimal hyperparameters is then deployed to a SageMaker Endpoint for further testing



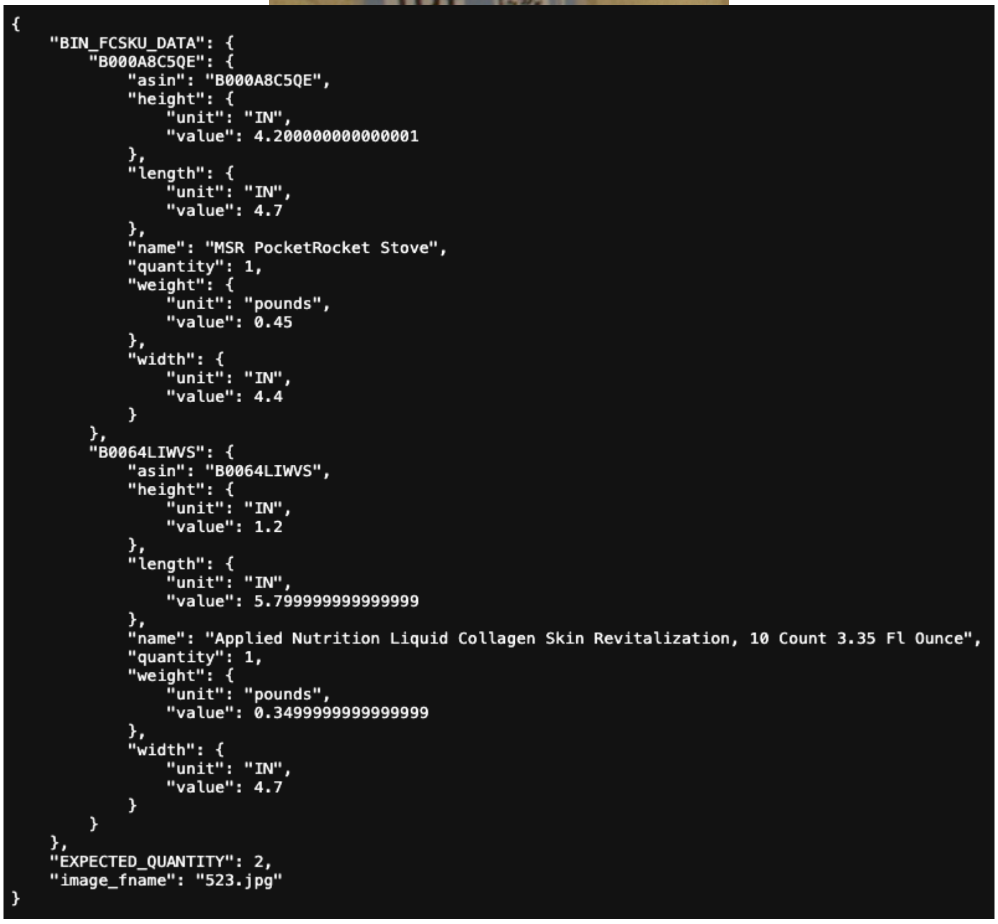
Endpoints

Name	ARN	Creation time	Status	Last updated
pytorch-inference-2023-12-10-15-29-20-603	arn:aws:sagemaker:us-east-1:063864077654:endpoint/pytorch-inference-2023-12-10-15-29-20-603	12/10/2023, 10:29:21 PM	InService	12/10/2023, 10:31:40 PM

Fig. 8. Deploy the trained model with SageMaker Endpoint.

Justification

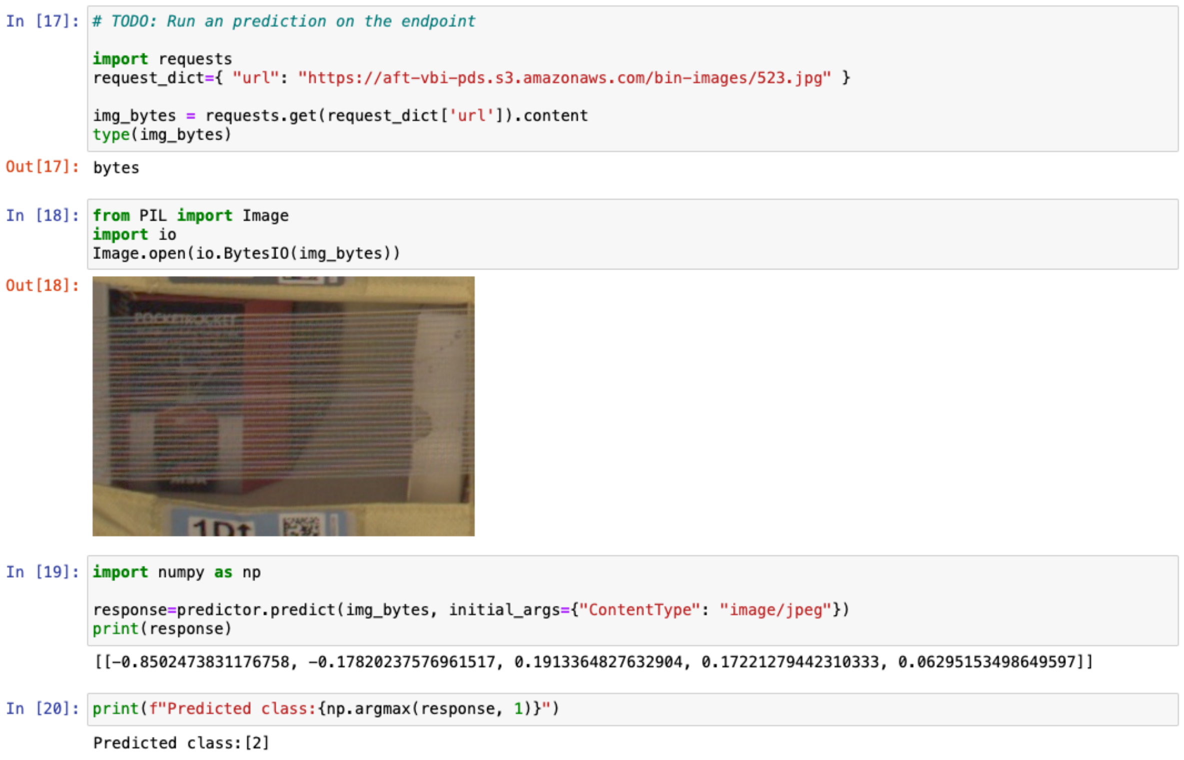
We verify the performance of our trained model using real data samples from the Amazon Image Bin dataset. Below are the image of the 523th sample in the dataset and its corresponding metadata:



```
{
    "BIN_FCSKU_DATA": {
        "B000A8C5QE": {
            "asin": "B000A8C5QE",
            "height": {
                "unit": "IN",
                "value": 4.200000000000001
            },
            "length": {
                "unit": "IN",
                "value": 4.7
            },
            "name": "MSR PocketRocket Stove",
            "quantity": 1,
            "weight": {
                "unit": "pounds",
                "value": 0.45
            },
            "width": {
                "unit": "IN",
                "value": 4.4
            }
        },
        "B0064LIWVS": {
            "asin": "B0064LIWVS",
            "height": {
                "unit": "IN",
                "value": 1.2
            },
            "length": {
                "unit": "IN",
                "value": 5.799999999999999
            },
            "name": "Applied Nutrition Liquid Collagen Skin Revitalization, 10 Count 3.35 Fl Ounce",
            "quantity": 1,
            "weight": {
                "unit": "pounds",
                "value": 0.3499999999999999
            },
            "width": {
                "unit": "IN",
                "value": 4.7
            }
        }
    },
    "EXPECTED_QUANTITY": 2,
    "image_fname": "523.jpg"
}
```

Fig. 9. The image and the corresponding metadata of the 523th sample.

As can be seen, the "EXPECTED_QUANTITY" of this image is 2 items in the bin. Next, let test our trained model by using the deployed Endpoint as below:



```
In [17]: # TODO: Run an prediction on the endpoint

         import requests
         request_dict={ "url": "https://aft-vbi-pds.s3.amazonaws.com/bin-images/523.jpg" }

         img_bytes = requests.get(request_dict['url']).content
         type(img_bytes)

Out[17]: bytes

In [18]: from PIL import Image
         import io
         Image.open(io.BytesIO(img_bytes))

Out[18]:

In [19]: import numpy as np

         response=predictor.predict(img_bytes, initial_args={"ContentType": "image/jpeg"})
         print(response)

         [[-0.8502473831176758, -0.17820237576961517, 0.1913364827632904, 0.17221279442310333, 0.06295153498649597]]

In [20]: print(f"Predicted class:{np.argmax(response, 1)}")

         Predicted class:[2]
```

Fig. 10. Test the deployed Endpoint.

We can see that the predicted class from the deployed Endpoint is 2, yielding the similar result with the expected output in the metadata. This has shown that our fine-tuned model are accurate in counting the number of items in bins, which has addressed the target problem of this Capstone Project.

REFERENCES

[1] Amazon Bin Image Dataset: https://registry.opendata.aws/amazon-bin-imagery/.

[2] He, K., Zhang, X., Ren, S., and Sun, J., “Deep Residual Learning for Image Recognition”, <i>arXiv e-prints</i>, 2015. doi:10.48550/arXiv.1512.03385.