Machine Learning Engineer Capstone Project: Inventory Monitoring at Distribution Centers

Counting objects from the images of bins taken by an operating Amazon FC

Engincan Meydan

Table of Contents

A. DEFINITION

Project Overview

In this project, we will train, build and deploy a deep learning model to count the number of objects from the bin-images taken by operating Amazon Fulfillment Center (FC). It is a real-world example as distribution centers would benefit from inventory tracking with correct number of items in delivery consignments.

Domain Background

Distribution centers are the foundation of a supply network which is stocked with goods to be redistributed to retailers, wholesalers or the customers. It is often defined as a warehouse which plays a key role in order fulfillment process. Amazon uses Fulfillment Center (FC) instead of "warehouse" because a standard warehouse keeps inventory until the store, whereas Amazon's warehouses ship items to the customers in addition to storage [1]. These facilities use robots to move objects from one place to another. The objects are carried in bins which can contain multiple objects. As Amazon ships ~1.6M packages a day, automation is key to optimize operations and speed up delivery [2].

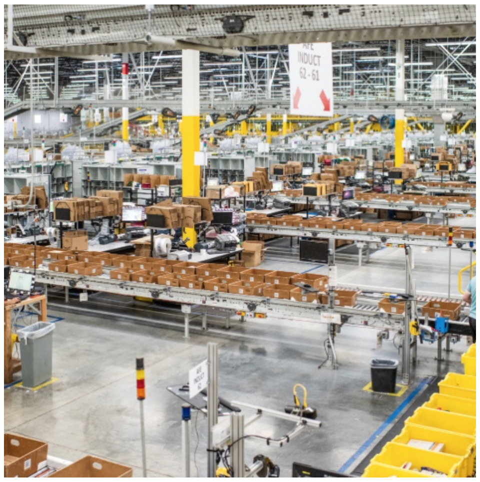


Figure 1: An image from Amazon Fulfilment Center [3]

Problem Statement

Inventory management allows companies to identify which and how much stocks needed at the point of a time to fulfill customers’ requests. However, without using intelligence, the process is highly manual for workers to keep tracking. The main problem we focus on this project is the manual inventory monitoring to track the number of objects being sent or received. Manual classification of how many objects in each bin in the warehouse would take +20sec per bin considering the process of scanning/logging it. For Amazon, it would make 3K hours productivity loss every day (assumption of avg 3 object per bin). This is not only a problem which is related with Amazon but from all industries companies deal with inventory management.

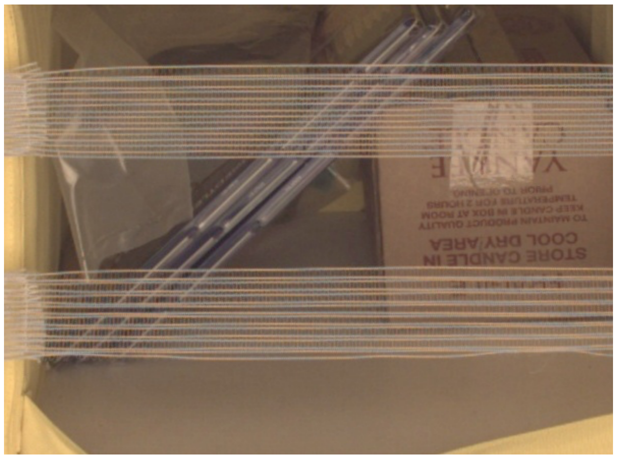


Figure 2: Example of a bin image which contains multiple objects

Our dataset is from "Open Data on AWS" called "Amazon Bin Image Dataset". 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 during Amazon Fulfillment Center operations.

The main purpose is to demonstrate how we could build end-to-end Machine Learning pipeline. Therefore, having smaller dataset will be convenient considering the costs that would raise during AWS services utilization. We will not utilize all those images but build a model with a sample.

Evaluation metrics

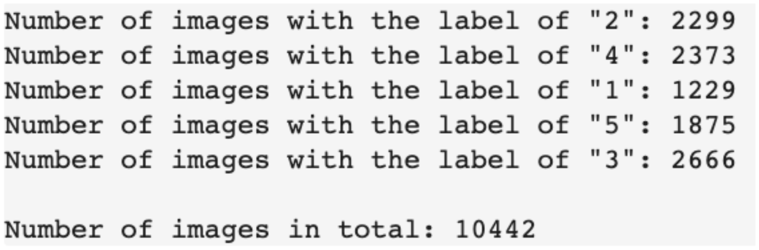
The evaluation metric will be the accuracy. Given the problem we would like to solve, we need to be able to correctly classify how many objects there are in each bin. The calculation of it will be as [Correct Inferences/Total Inferences]. We will also 1/keep our eyes on imbalance in dataset regarding images with number of objects and 2/measure accuracy for each label, such as how accurately our model could predict the images with 2 objects or 3 objects.

B. ANALYSIS

Exploratory Data Analysis

In the Amazon Bin Image Dataset, we have metadata on top of the images. To be able to build a model, we had to divide images into labelled folders such as number of objects: 1. However, Udacity team provided us json file which we could find already labeled 2% of the dataset (10441 images). We have 5 folders, named as 1,2,3,4,5 which are our classes showing how many objects exist in each bin. Due to cost concerns considering main purpose of the project as an end-to-end machine learning pipeline, we will move on with this dataset.

After diving deep into the data, first thing we noticed is that there is a slight imbalance in the dataset with label-1 as ~12% of the dataset whereas label-5 as 25%. One of the options could be balancing all labels by keeping only 1229 images each. However, this time I will go with the data as it is since we have relatively low number of data for this kind of complex task.



```
Number of images with the label of "2": 2299
Number of images with the label of "4": 2373
Number of images with the label of "1": 1229
Number of images with the label of "5": 1875
Number of images with the label of "3": 2666

Number of images in total: 10442
```

Figure 3: Dataset after downloading via provided json file

Second thing we noticed is the different sizes in the images. For image classification, we will resize those images with transform functions. In the early stage, I decided to resize the images before starting to utilize training scripts. However, I have realized that this is not a good idea, as the new images we will receive from Amazon FCs will keep being at different sizes and it will lead wrong result in deployment.

As a third thing, you might notice that images are not clear due to lines on top of the bins. I assume these are to prevent objects to drop while carrying it. However, this increases complexity of image classification.

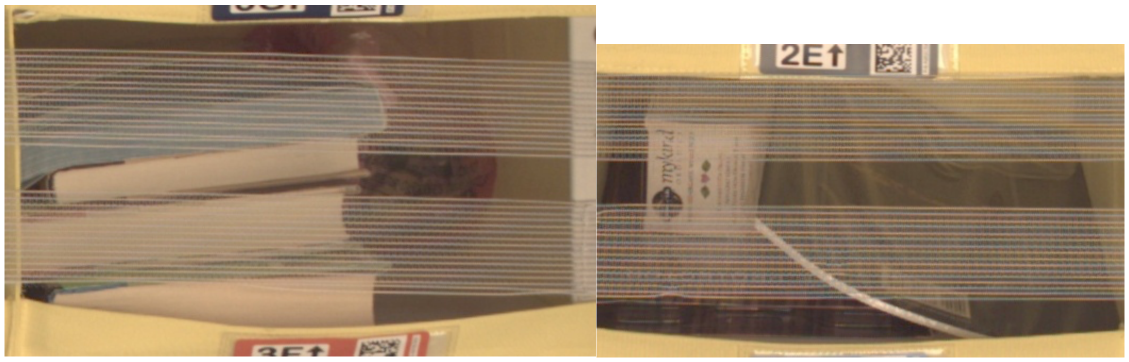
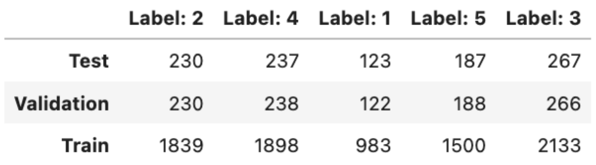


Figure 4: Bin Images with 5 and 3 objects with different sizes pictures

Algorithms and Techniques

After having downloaded data, we will need to do a split as train, test and validation. Due to low number of images, I decided to go with 80-10-10 split. My approach is to use “os” library to create a folder called “amazon_images” and do the split by iterating over the downloaded data which is available under “train_data” folder.



	Label: 2	Label: 4	Label: 1	Label: 5	Label: 3
Test	230	237	123	187	267
Validation	230	238	122	188	266
Train	1839	1898	983	1500	2133

Figure 5: Train, Test, Validation Split

Image Classification from Deep Learning would be the method to achieve this training. As a platform, we could leverage AWS' scalability, reliability, availability and security to build, train and deploy our models. Sagemaker will provide us training jobs, debugging/profiling reports, hyperparameter tuners, endpoints and many more with fully managed environment so we don't need to do updates or patches in the server.

As a model, I am planning to use Resnet50 which is a convolutional neural network (CNN) that is 50 layers deep with higher accuracy than Resnet34. I will utilize pretrained model and support it with few fully connected layers [4]. My activation function will be Relu which is Rectified Liner Unit as one of the most common functions, easier to train and often achieves better performance [5].

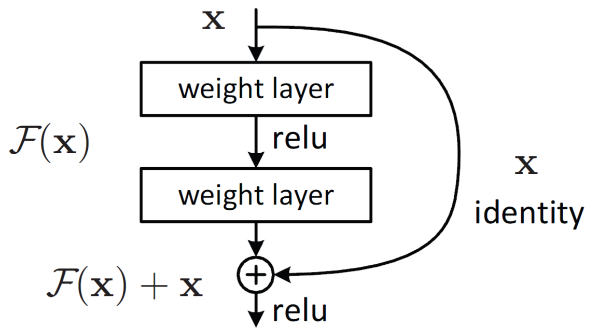
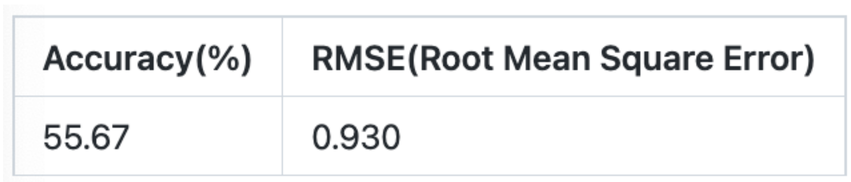


Figure 6: Simplified version of layers that I will utilize with ReLu activation function [6]

As a cost function which is used as a signal to tell how well the neural network is learning and if we are going in the right direction, I will use Cross-Entropy Loss. It is used for multiple classification problems which is exactly what we try to achieve. In our hyperparameters, it will be used to avoid model to skip minimum.

Benchmark

As a benchmark, I will leverage from "usage examples" from Amazon Bin Image Dataset as a publication [7]. Main reason is that project owner also leveraged accuracy metric and followed up a similar approach. However, the hyperparameter, dataset I will use will be more conservative such as (40epochs vs 5 or 500K images vs 10K). As it is shown below, the benchmark model achieved 55.67% accuracy. Due to the size of our dataset and cost-efficient methods we will use, I set up a goal of 25-30% accuracy in my model.



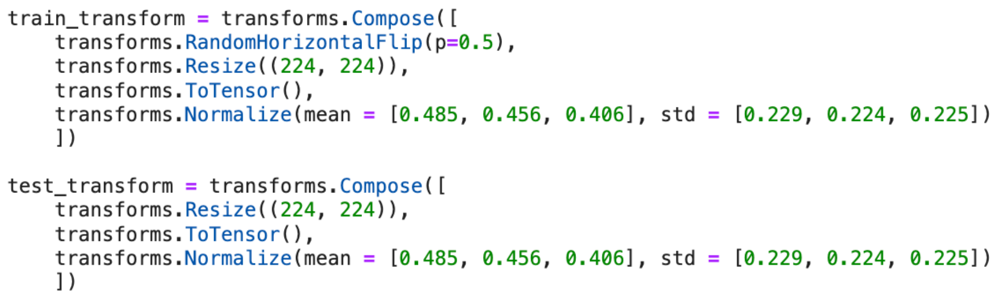
Accuracy(%)	RMSE(Root Mean Square Error)
55.67	0.930

Figure 7: Accuracy of Benchmark model from usage example

C. METHODOLOGY

Data Preprocessing

In our training jobs, one of the most important things was transforming images so we could correctly train the model we would like to achieve. After specifying the paths for train, test and validation; we kicked off the transforming jobs. We started the process with random horizontal flip which is a type of image data augmentation that horizontally flips the image with a give probability in this case it is 0.5. Then as we mentioned above, we are achieving resizing thanks to the “torchvision”. This step is important as we need to tensorize the images before normalization. I had been confused with the order of them so my training job was giving error. Only difference of training transform is random horizontal flip which we do not need in testing. We use data loaders to upload our data by giving the related paths.



```
train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225])
    ])

test_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225])
    ])
```

Figure 8: Transforming process

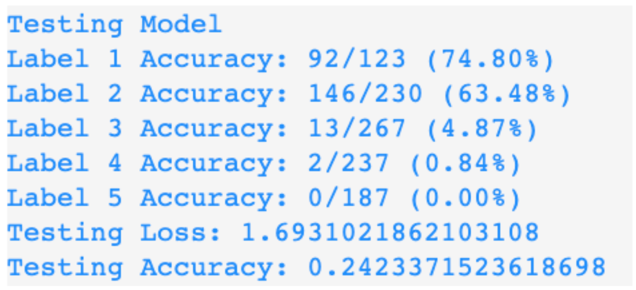
Implementation

S3 Upload

After we downloaded and split the data, it is time to upload it to Amazon S3 bucket, so we could give path to Amazon Sagemaker training jobs. In the jupyter notebook, I provided two different methods to achieve this.

Model Training

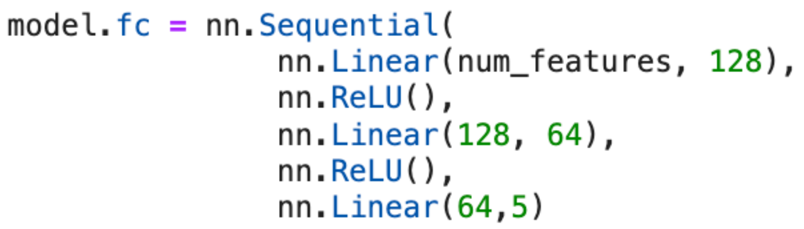
Before any finetuning (as it is also not required for this project), I wanted to give it a chance to the model with hardcoded hyperparameters of learning rate (0.01) and batch size (32). After two epochs, it provided a result of 24% accuracy as stated below. I was happy with initial result as my target was achieving 25-30% accuracy. What I can see that we are not able to predict label: 4, 5 correctly. Due to images with low quality and lines in between, I was expecting to see low accuracy towards high number of objects per bin.



```
Testing Model
Label 1 Accuracy: 92/123 (74.80%)
Label 2 Accuracy: 146/230 (63.48%)
Label 3 Accuracy: 13/267 (4.87%)
Label 4 Accuracy: 2/237 (0.84%)
Label 5 Accuracy: 0/187 (0.00%)
Testing Loss: 1.6931021862103108
Testing Accuracy: 0.2423371523618698
```

Figure 9: First training without having any tuning processes.

To train this model, I leveraged pretrained ResNet50. As stated before, I improved it with few more fully connected layers. My activation function is Relu which is Rectified Liner Unit. You might notice that we don't close the model with Relu function as our loss function of CrossEntropy will apply SoftMax, used to covert outputs to probabilities, internally.



```
model.fc = nn.Sequential(
                   nn.Linear(num_features, 128),
                   nn.ReLU(),
                   nn.Linear(128, 64),
                   nn.ReLU(),
                   nn.Linear(64,5)
```

Figure 10: Connected Layers for image classification

$$S(x) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

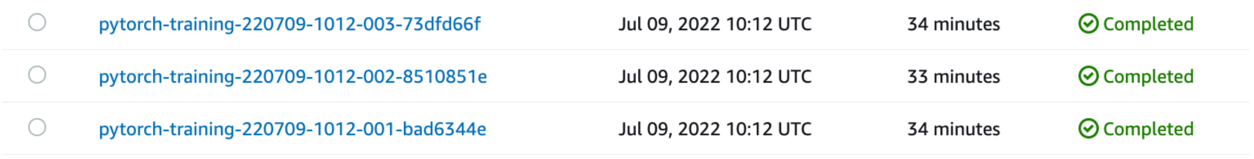
Figure 11: Softmax Function

Refinement - Hyperparameter Tuning

Throughout this course, I learned how convenient it is to run a hyperparameter tuning job via Sagemaker. I chose two parameters to tune:

- Learning Rate between (0.001 and 0.1) as continuous
- Batch Size between (32, 64, 128) as categorical parameter

It is great to be able to trust Sagemaker with objective metric to find the best ones. I did not want to increase values further to avoid further increasing the training duration due to cost concerns. I have decided to pick "Test Lost" as objective metric name. Hyperparameter tuner will track this metric and give us best parameters according to the results. For my estimator, I tried both instances types, with and without GPU - ml.g4dn.xlarge was two times faster than ml.m5.2xlarge. To be able leverage parallel jobs, I went for ml.5.2xlarge as I only have one GPU instance, you need to request via ticket to have more.

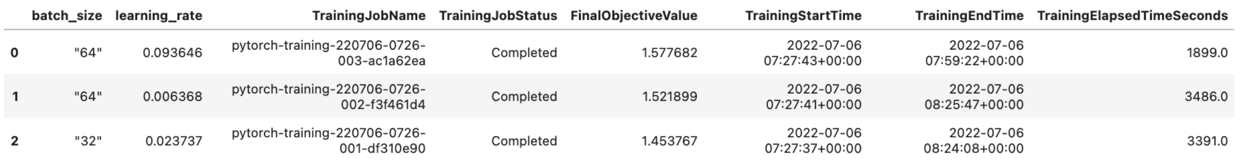


○	pytorch-training-220709-1012-003-73dfd66f	Jul 09, 2022 10:12 UTC	34 minutes	✓ Completed
○	pytorch-training-220709-1012-002-8510851e	Jul 09, 2022 10:12 UTC	33 minutes	✓ Completed
○	pytorch-training-220709-1012-001-bad6344e	Jul 09, 2022 10:12 UTC	34 minutes	✓ Completed

Figure 12: Hyperparameter Tuner Jobs to achieve best estimator

Our second job achieved the lowest test loss with the given parameters below. It is also nice to visualize the jobs with parameters during hyperparameter tunning. We can see the duration, final objective value and more details.

- Best Learning Rate: 0.0015790022970053907
- Best Batch Size: 64

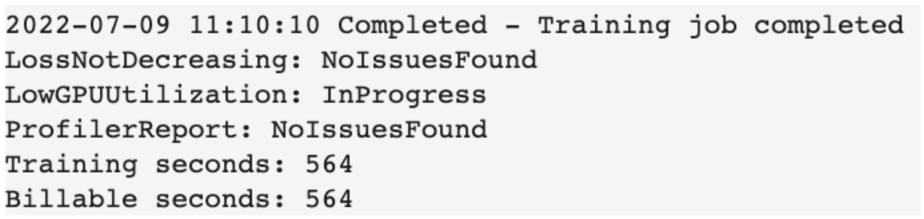


	batch_size	learning_rate	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTime	TrainingElapsedTimeSeconds
0	"64"	0.093646	pytorch-training-220706-0726-003-ac1a62ea	Completed	1.577682	2022-07-06 07:27:43+00:00	2022-07-06 07:59:22+00:00	1899.0
1	"64"	0.006368	pytorch-training-220706-0726-002-f3f461d4	Completed	1.521899	2022-07-06 07:27:41+00:00	2022-07-06 08:25:47+00:00	3486.0
2	"32"	0.023737	pytorch-training-220706-0726-001-df310e90	Completed	1.453767	2022-07-06 07:27:37+00:00	2022-07-06 08:24:08+00:00	3391.0

Figure 13: Hyperparameter Tuner Jobs with different parameters

Debugging/Profiling Job

After defining the correct hyperparameters, I decided to use Sagemaker Debugger/Profiling. As normally we have to read through logs to identify issues about the model, this helps us to understand problems if the training is not working well and generate a simple report called the Profiler Report that gives us an overview of our training job. Key thing here is to add hooks to train and test the model after importing necessary libraries. Creating hook and registering the model will give us ability to select rules such as loss_not_decreasing, low_gpu_utlization and profiler_report. As it is given below, we also did not encounter issue with loss not decreasing.



```
2022-07-09 11:10:10 Completed - Training job completed
LossNotDecreasing: NoIssuesFound
LowGPUUtilization: InProgress
ProfilerReport: NoIssuesFound
Training seconds: 564
Billable seconds: 564
```

Figure 14: Debugging/Profiling Issue Inspection

As we can see below, there is a callout for time sending not focusing on training and evaluation in the profiling report. Downloading data from our dataset is big part of the time spent which can be seen from the logs during the trainings. Another example is the suggestions for batch size. Profiler Report mentions that the batch size is too small and GPUs are underutilized. It gives us opportunity to make the model more cost efficient.

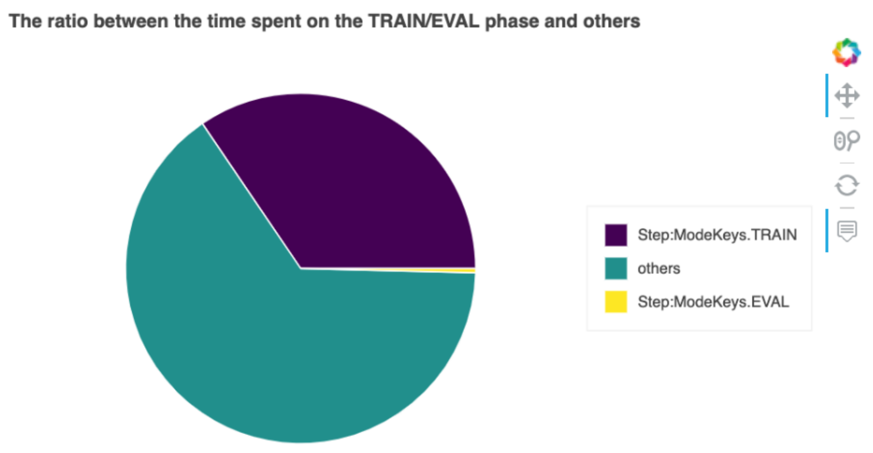
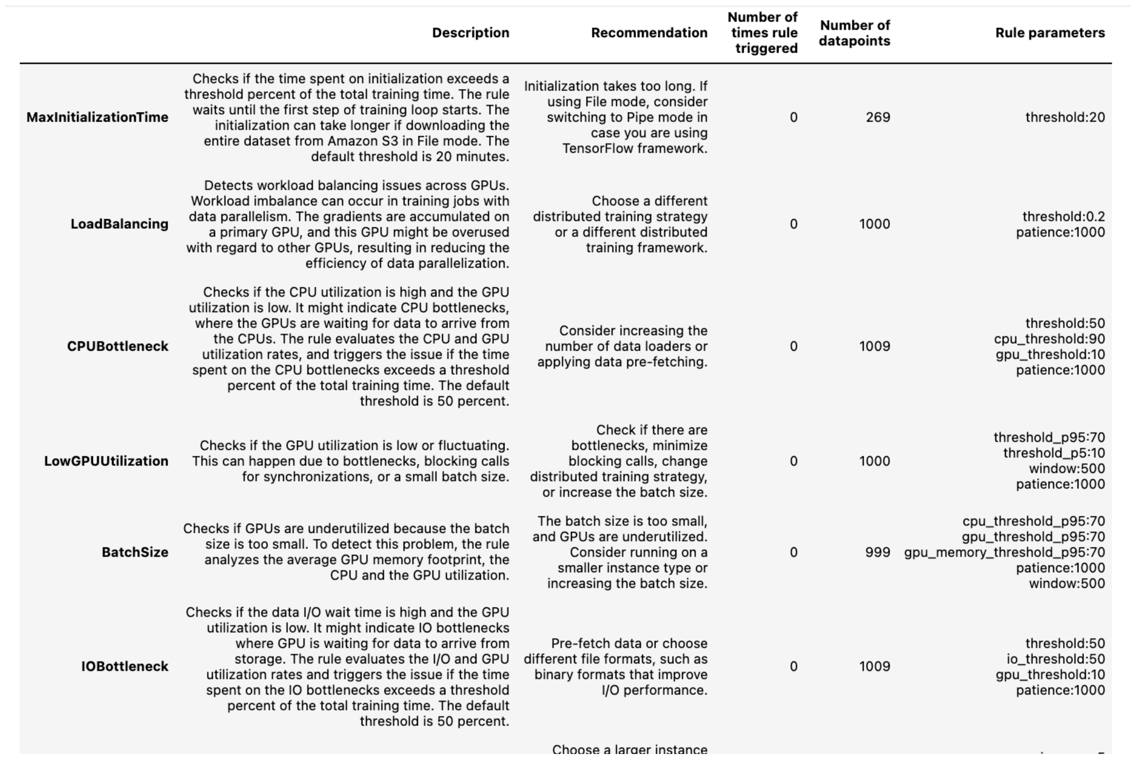


Figure 15: Debugging/Profiling Framework Metrics



	Description	Recommendation	Number of times rule triggered	Number of datapoints	Rule parameters
MaxInitializationTime	Checks if the time spent on initialization exceeds a threshold percent of the total training time. The rule waits until the first step of training loop starts. The initialization can take longer if downloading the entire dataset from Amazon S3 in File mode. The default threshold is 20 minutes.	Initialization takes too long. If using File mode, consider switching to Pipe mode in case you are using TensorFlow framework.	0	269	threshold:20
LoadBalancing	Detects workload balancing issues across GPUs. Workload imbalance can occur in training jobs with data parallelism. The gradients are accumulated on a primary GPU, and this GPU might be overused with regard to other GPUs, resulting in reducing the efficiency of data parallelization.	Choose a different distributed training strategy or a different distributed training framework.	0	1000	threshold:0.2 patience:1000
CPUBottleneck	Checks if the CPU utilization is high and the GPU utilization is low. It might indicate CPU bottlenecks, where the GPUs are waiting for data to arrive from the CPUs. The rule evaluates the CPU and GPU utilization rates, and triggers the issue if the time spent on the CPU bottlenecks exceeds a threshold percent of the total training time. The default threshold is 50 percent.	Consider increasing the number of data loaders or applying data pre-fetching.	0	1009	threshold:50 cpu_threshold:90 gpu_threshold:10 patience:1000
LowGPUUtilization	Checks if the GPU utilization is low or fluctuating. This can happen due to bottlenecks, blocking calls for synchronizations, or a small batch size.	Check if there are bottlenecks, minimize blocking calls, change distributed training strategy, or increase the batch size.	0	1000	threshold_p95:70 threshold_p5:10 window:500 patience:1000
BatchSize	Checks if GPUs are underutilized because the batch size is too small. To detect this problem, the rule analyzes the average GPU memory footprint, the CPU and the GPU utilization.	The batch size is too small, and GPUs are underutilized. Consider running on a smaller instance type or increasing the batch size.	0	999	cpu_threshold_p95:70 gpu_threshold_p95:70 gpu_memory_threshold_p95:70 patience:1000 window:500
IOBottleneck	Checks if the data I/O wait time is high and the GPU utilization is low. It might indicate IO bottlenecks where GPU is waiting for data to arrive from storage. The rule evaluates the I/O and GPU utilization rates and triggers the issue if the time spent on the IO bottlenecks exceeds a threshold percent of the total training time. The default threshold is 50 percent.	Pre-fetch data or choose different file formats, such as binary formats that improve I/O performance.	0	1009	threshold:50 io_threshold:50 gpu_threshold:10 patience:1000
		Choose a larger instance			

Figure 16: Debugging/Profiling Rules Summary

As we used best hyperparameters, we managed to increase our accuracy to 30% with test loss of 1.48. What is noticeable is that now we are able predict 13% of label 5 correctly but missing label 4 still. This is highly related with low number of datasets we have and also imbalance we already noticed.

```
2022-07-09 11:09:39 Uploading - Uploading generated training model Label 1 Accuracy: 37/123 (30.08%)
 Label 2 Accuracy: 39/230 (16.96%)
 Label 3 Accuracy: 215/267 (80.52%)
 Label 4 Accuracy: 0/237 (0.00%)
 Label 5 Accuracy: 25/187 (13.37%)
Testing Loss: 1.4823028173483195
Testing Accuracy: 0.30268198251724243
Printing Log
Test set: Average loss: 1.4823, Accuracy: 316/1044 (30%)
```

Figure 17: Overall model accuracy and per label

In addition to the rules, we could see how many tensor recorded and adjust these to visualize our loss in the diagram. It is important to choose right save_intervals for training and validation jobs in configurations as if you don't have many steps, you might end up with only few recorded points. I set them up as 20 steps interval for training and 5 for validation as we finish around 50 steps for validation. However, this process needs iteration as initially I ended up with only 2-3 recorded points and then more than 100.

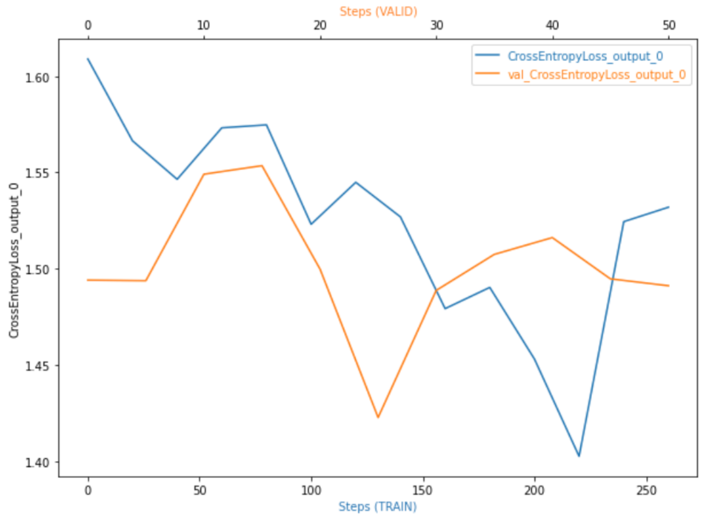
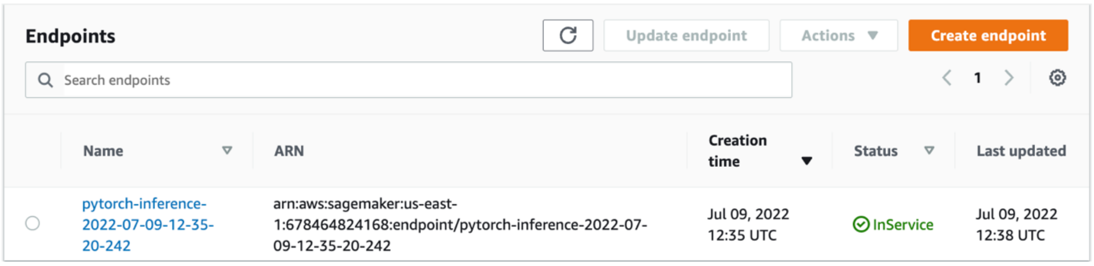


Figure 18: CrossEntropyLoss Output per step

Deployment

After identifying the best model and double check the issues with debugging/profiling reports, now it is time for deployment. First, I provide the model_location which can be easily identified by profiling output path as it is just in the folder of "output" instead of "profiler_output". As we experienced throughout this Udacity course, we need a different entry point py file for deployment. This file includes functions of net, predict_fn, input_fn and model_fn. In addition to this, we need to define predictor_cls, which is a function to call to create a predictor. Our ImagePredictor class will collect endpoint name, serializer, deserializer and also Sagemaker session.

For deployment, I am only using ml.m5.large, because we should not forget that we already trained the model, this is only for deployment. With .deploy, we created Sagemaker endpoint and now we could predict images after opening the files, followed by predictor.predict(). Sagemaker endpoints cost as long as they are active, so if you are on production, delete the endpoints when you are done.



Name	ARN	Creation time	Status	Last updated
pytorch-inference-2022-07-09-12-35-20-242	arn:aws:sagemaker:us-east-1:678464824168:endpoint/pytorch-inference-2022-07-09-12-35-20-242	Jul 09, 2022 12:35 UTC	InService	Jul 09, 2022 12:38 UTC

Figure 19: Active Sagemaker endpoint to create inference

D. RESULTS

Model Evaluation and Validation

We provide the best hyperparameters and after the end of two epochs, we achieved 30% accuracy that we defined as a target. I am overall happy with the result which shows potential for training with high volume of dataset to achieve higher accuracy and lower loss.

Model	Test Loss	Accuracy
No-tuned	1.69	24%
Hyperparameter Tuned	1.48	30%
Benchmark	n/a	56%

By using recently created endpoint, I predicted few images from different classes. Beyond having accuracy metric per each class, seeing the model in production gave me confidence.

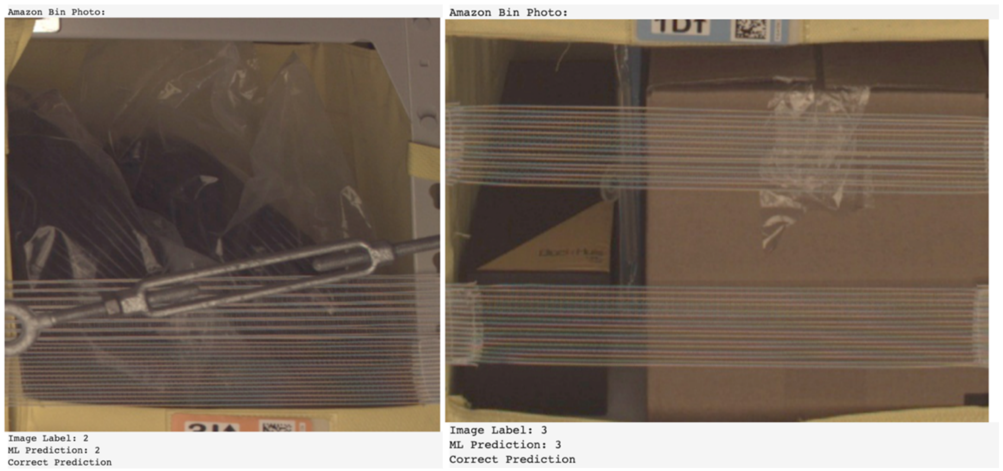
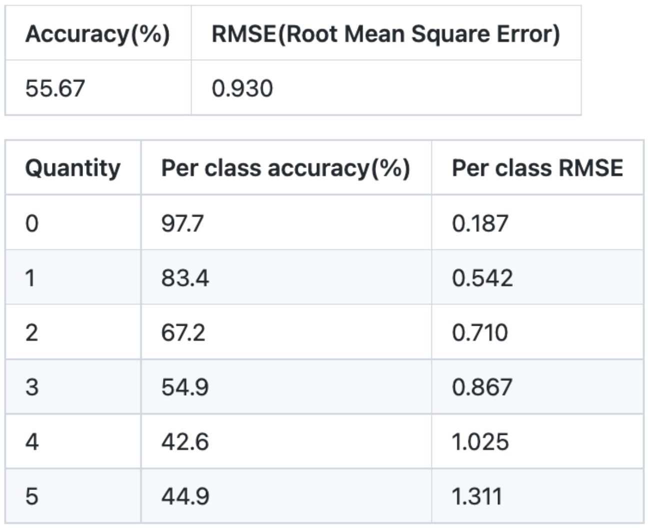


Figure 20: Prediction examples to demonstrate predicted/actual labels

Justification

Our benchmark had achieved 55% accuracy, but due to sample usage in our project, we had put our target accuracy as between 25-30%. This accuracy would not be sufficient to put the model into production for Fulfillment Centers. However, it shows the potential that when we use more dataset, we could increase this accuracy and achieve a model that out into the production. While working with Fulfillment Centers, data collection is really important. Same angle images with high resolution would allow deep learning models to better learn and predict number of objects in each bin.



Accuracy(%)	RMSE(Root Mean Square Error)
55.67	0.930

Quantity	Per class accuracy(%)	Per class RMSE
0	97.7	0.187
1	83.4	0.542
2	67.2	0.710
3	54.9	0.867
4	42.6	1.025
5	44.9	1.311

Figure 18: Benchmark Result

As a next steps, to put this model into the production from Amazon FCs, I would collect more images and balance those to be able efficiently train the model. In addition, if there is any possibility to influence, the angle and resolution of the images can be adjusted.

E. References

1. "Amazon Facilities." *US About Amazon*, US About Amazon, 21 Sept. 2020, https://www.aboutamazon.com/workplace/facilities.
2. "57 Amazon Statistics to Know in 2022." *LandingCube*, 20 June 2022, https://landingcube.com/amazon-statistics/.
3. "Gestión De Centros Logísticos." *Amazon.jobs*, https://www.amazon.jobs/es/teams/fulfillment-center-management.
4. "Deep Network Designer." *ResNet-50 Convolutional Neural Network - MATLAB*, https://www.mathworks.com/help/deeplearning/ref/resnet50.html.
5. Brownlee, Jason. "A Gentle Introduction to the Rectified Linear Unit (ReLU)." *Machine Learning Mastery*, 20 Aug. 2020, https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.
6. "ResNet (34, 50, 101): Residual CNNS for Image Classification Tasks." *Neurohive*, 25 Jan. 2019, https://neurohive.io/en/popular-networks/resnet/.
7. "Amazon Bin Image Dataset." *Amazon Bin Image Dataset - Registry of Open Data on AWS*, https://registry.opendata.aws/amazon-bin-imagery/.
8. Hudgeon, Doug, and Richard Nichol. "Machine Learning for Business: Using Amazon Sagemaker and Jupyter." *Amazon*, Manning Publications, 2020, https://aws.amazon.com/sagemaker/.