AWS Machine Learning Engineer Nanodegree

Capstone Project - Inventory Monitoring at Distribution Centers

Faezeh Siavashi
January 23, 2022

1. Project Definition

Project overview

Keeping the right number of parts and products in a packaging customer orders, is one of the challenges that distribution centers and warehouses face. Inventory monitoring is a process to check the number of products in inventories as well as the number of ordered items in packages. Distribution centers use robots to move items as a part of their operations. Items are carried in “bins” which can contain multiple items. Due to the massive amount of bins that should be delivered every day, computer vision is an excellent automation choice to verify the number of items in each bin before packaging and delivery.
In this project, we solve the challenge of monitoring the number of items in each bin. I use the public dataset from Amazon Bin Image Dataset, which contains images and metadata from more than 500,000 bins. The dataset also is studied in two different projects (Amazon Bin Image Dataset(ABID) Challenge) and (Amazon Inventory Reconciliation using AI, n.d.).

Problem Statement

The problem that we attempted to solve with this project is to ensure that the number of items in each inventory bin is correct. On one hand, if the bins are packed with fewer items than ordered, then the reputation of the service provider will be damaged. On the other hand, if there are extra items in a package, then the service provider will lose profits. To this end, it is required to keep track of the correct number of items in the bins. In order to achieve the goal, we took the following steps:

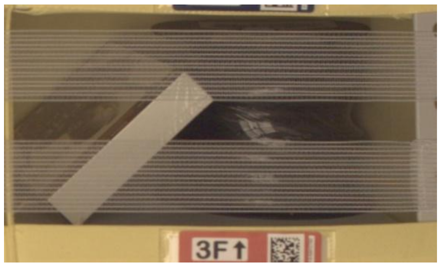
- Downloaded a part of the Amazon inventory bins dataset and classify it.
- Splitted the dataset into train, test and validation folders.
- Implemented and trained ResNet50 machine learning model. Tuned the hyperparameters for a more accurate model.
- Evaluated the model by predicting given bin images with the actual number of items in the bins that is retrieved from the images' metadata.
- Created a benchmark with another machine learning model and compared its accuracy with our model.

Evaluation Metrics

In order to evaluate the accuracy of the model, we use Cross Entropy Loss which is a good metric for image classification problems. We plot cross entropy loss for both training and testing.

2. Data Analysis

The Amazon Bin Image Dataset is a public dataset that can be accessed here. It contains more than 500,000 images and metadata. Examples of the images in the dataset are shown below. As it can be seen, items in some of the images are not clear. Therefore, we expect to have lower accuracy in our train model.



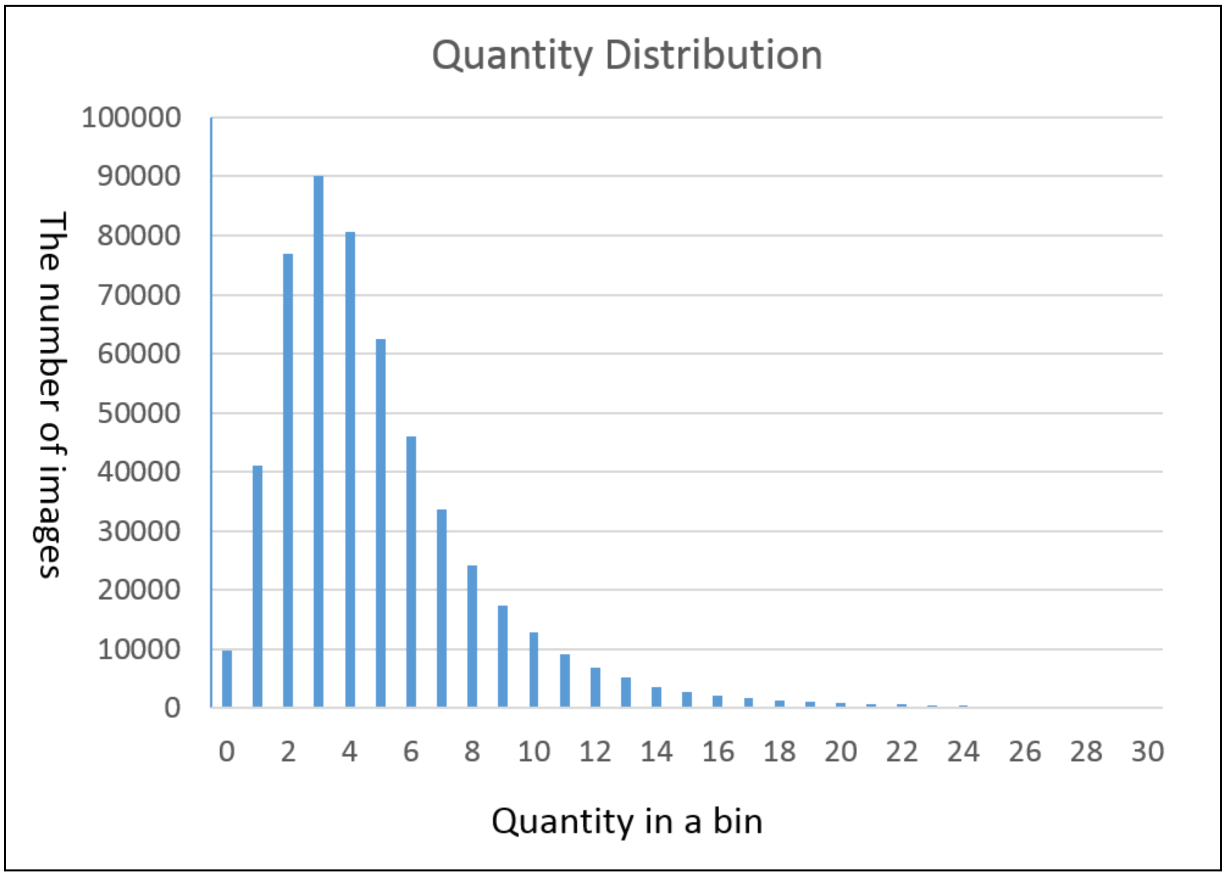
Metadata of each image provides more information about the bin. The tricky part in this image is that there are 3 items inside a clear packed item which is counted as 1 item. As it is shown in the metadata, the expected quantity is 1, although 3 items are detected.

```
{
   "BIN_FCSKU_DATA": {
      "B0029O0BWW": {
         "asin": "B0029O0BWW",
         "height": {
            "unit": "IN",
            "value": 2.3
         },
         "length": {
            "unit": "IN",
            "value": 6.2999999999999999
         },
          "name": "Nature Made Potassium Gluconate 550mg, 100 Tablets (Pack of 3)",
         "quantity": 1,
         "weight": {
            "unit": "pounds",
            "value": 0.35
         },
         "width": {
            "unit": "IN",
            "value": 3.3999999999999995
         }
      }
   },
   "EXPECTED_QUANTITY": 1,
   "image_fname": "302.jpg"
```

}	

Data Visualization

For data visualization, we plot the distribution of the quantity of the items in bins. The diagram below shows that the majority of bins contain between 1 to 5 items. Very small number of bins contain more than 6 items.



In our project, we limit our model train on a smaller dataset on quantities between 1 to 5 items in a bin because the majority of the images have between 1 to 5 items. Therefore, there is enough data for training our model.

Requirements

We utilize the **ResNet50** image classification machine learning model to solve the problem in this project. The reason for choosing this model is that it improves the efficiency of deep neural networks with more neural layers while minimizing the percentage of errors. ResNet50 is one of the best algorithms for solving image classification problems. We tuned the **learning rate** and **batch size** for hyperparameters in our model. We used a tuner and searched for the best combination of the hyperparameters from the training jobs.

Benchmark

Although there are two other publications that could be used for benchmark, since our dataset is smaller and our classification is limited to 5, we implement a simple machine learning model with smaller network layers with the same data processing method.

We trained the model and its benchmark once with **CrossEntropyLoss** as the metrics to evaluate the accuracy of our model. We later refined/improved the model with a different metric, **NLLLoss**, and compared the results one more time with the benchmark. The TestLoss and TestAccuracy are calculated below:

$$Test\ Loss\ = \frac{test\ loss}{number\ of\ test\ images} \qquad Test\ Accuracy\ = \frac{accuracy}{number\ of\ test\ images}$$

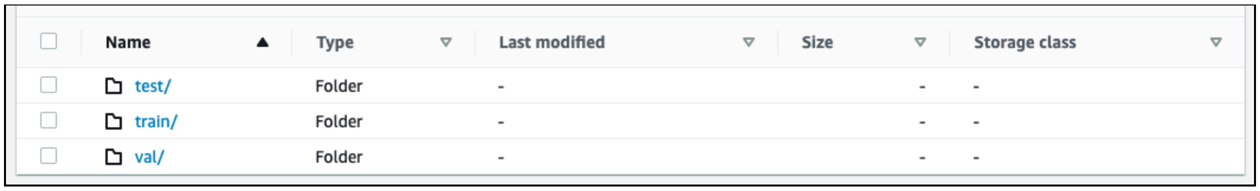
3. Methodology

Data pre-processing

For this project, we used a json file which contained the information of approx. 10k images that are classified into 5 classes, each class represents the number of items in images. This file is used to download and classify the images into 5 different folders, named 1-5. We partitioned our dataset into three distinct sets : a set 75% images to be used for training our machine learning model, a set of 15% of the images for validating the model, and a set of 10% of images for testing. Since the initial dataset that we download is already selected by shuffling, we do not need to shuffle the datasets at this point.

Next, we store the data into a S3 bucket.

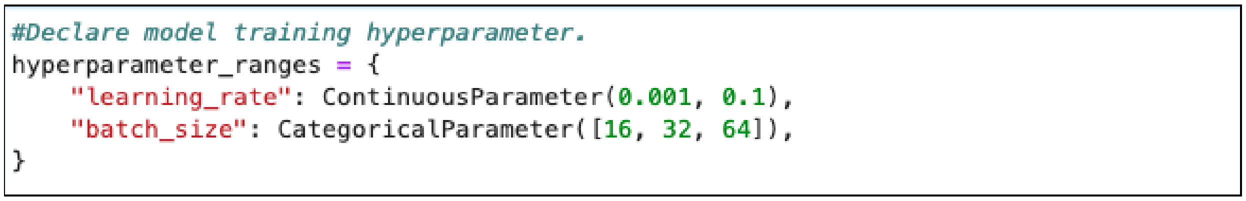
!aws s3 cp images s3://udacityfinalproject/ --recursive



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

Hyperparameters Tuning

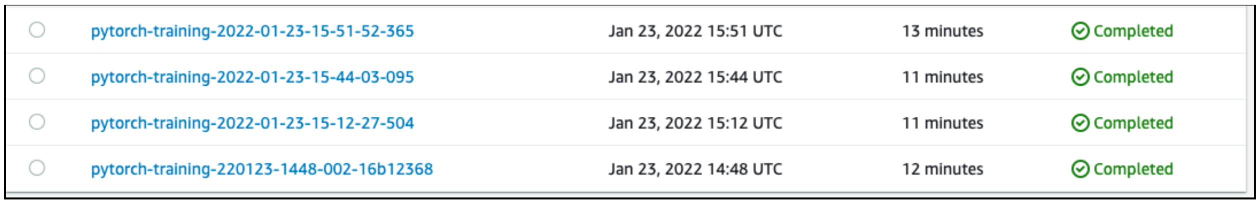
To select the best hyperparameter for our training model, we use a range of hyperparameters and execute multiple training jobs with different hyperparameters. The hyperparameter of the best tuner (with the least loss score) is used for training a classification model.



```
#Declare model training hyperparameter.
hyperparameter_ranges = {
    "learning_rate": ContinuousParameter(0.001, 0.1),
    "batch_size": CategoricalParameter([16, 32, 64]),
}
```

Implementation

One the pre-processing is completed, it is ready for implementing the training model. We use Pytorch, torchvision, and other related machine learning libraries. We selected the **ResNet50** deep learning model with 50 network layers. To improve performance of the model, I use some of the data augmentation techniques such as cropping and flipping the images during the training phase. Pytorch torchvision provides different image augmentation choices. The metrics that we use to evaluate the train model is **Cross Entropy Loss**. The implementation of the training model is provided in the "*train.py*" file. The maximum jobs for the estimator is set to 4. The completed training jobs in Sagemaker are shown below.

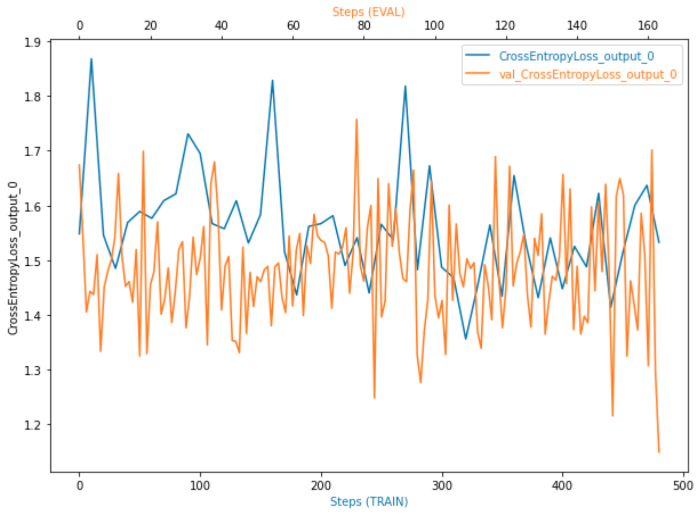


○	pytorch-training-2022-01-23-15-51-52-365	Jan 23, 2022 15:51 UTC	13 minutes	✓ Completed
○	pytorch-training-2022-01-23-15-44-03-095	Jan 23, 2022 15:44 UTC	11 minutes	✓ Completed
○	pytorch-training-2022-01-23-15-12-27-504	Jan 23, 2022 15:12 UTC	11 minutes	✓ Completed
○	pytorch-training-220123-1448-002-16b12368	Jan 23, 2022 14:48 UTC	12 minutes	✓ Completed

After selecting the best hyperparameters, we train the model one more time. To train a classification model, we add a profiler which logs a collection of performance metrics during the training and inference. We also add a debugging configuration to help develop better models by catching anomalies while training models. The trained model will be used for deployment.

After training a model with the best hyperparameters, I plot the train/validation accuracy as well as measure the CrossEntropyLoss. We set a debugger hook to record the Loss

Criterion of the process in both training and validation. The plot below shows the cross entropy loss for the training and validation processes.

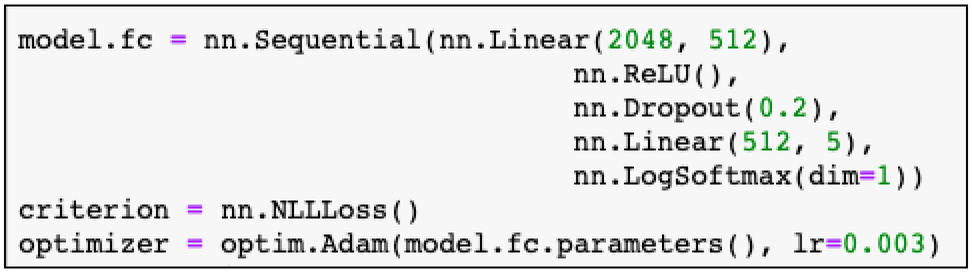


Although the plot shows the trend of having smaller loss, it shows fluctuations on crossentropyloss for both train and evaluation. One of the reasons behind this is the visibility of the images. The number of items in bins are usually difficult to distinguish even with human eyes. Another reason might be the small size of the dataset that we are using for this project.

Refinement / improvement

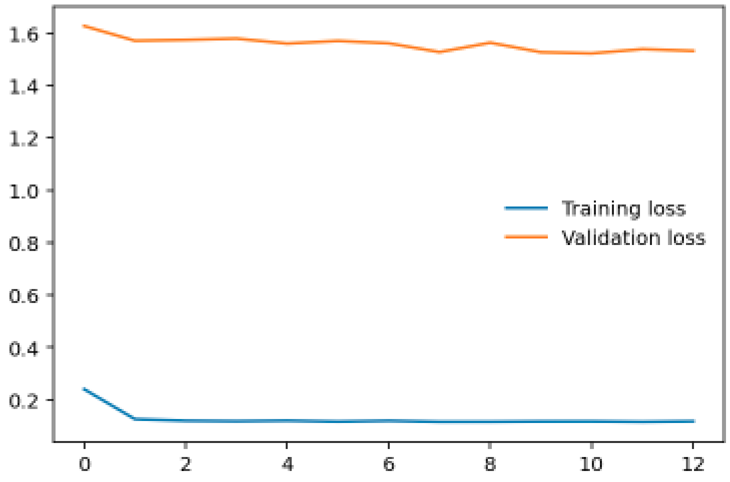
To improve the model there are some options that could be applied, such as changing the architecture of the training algorithm, adding augmentation to the dataset during preprocessing, and adding/tuning hyperparameters. In this project we applied the following changes to the training model:

- Added different additional steps into ResNet50 architecture
- Changed the Metrics from CrossEntropyLoss to NLLLoss



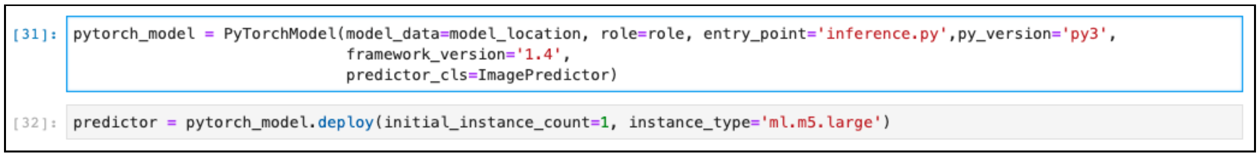
```
model.fc = nn.Sequential(nn.Linear(2048, 512),
                                 nn.ReLU(),
                                 nn.Dropout(0.2),
                                 nn.Linear(512, 5),
                                 nn.LogSoftmax(dim=1))
criterion = nn.NLLLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=0.003)
```

The diagram below shows the model train and test loss after refinement. As it can be seen, the model performance has improved.



Deployment

We use the pytorch deployment approach to deploy an endpoint for our model. The endpoint is created by following code:



```
[31]: pytorch_model = PyTorchModel(model_data=model_location, role=role, entry_point='inference.py',py_version='py3',
                                   framework_version='1.4',
                                   predictor_cls=ImagePredictor)

[32]: predictor = pytorch_model.deploy(initial_instance_count=1, instance_type='ml.m5.large')
```

Model_data is the link to the trained model and can be accessed via S3 bucket. Pytorch needs a script which implements the entry point and other internal processes for the endpoint. **Reference.py** provides the script.

4. Results

Test the model

To check if the model works correctly, a user can communicate with the endpoint by sending a HTTPS request with an image of a bin in bytes format. The response is a list of 5 different float numbers as predictions of 5 different bin classes. The index of highest prediction in the list is used as the index of the list of the bin classes.

Example:

```
!aws s3 cp s3://aft-vbi-pds/bin-images/00454.jpg 00454.jpg --no-sign-request
```

```
img2 = Image.open(r'00454.jpg')
img2.show()
img2_byte = convert_pil_image_to_byte_array(img2)
```

```
response2=predictor.predict(img2_byte, initial_args={"ContentType": "image/jpeg"})
```

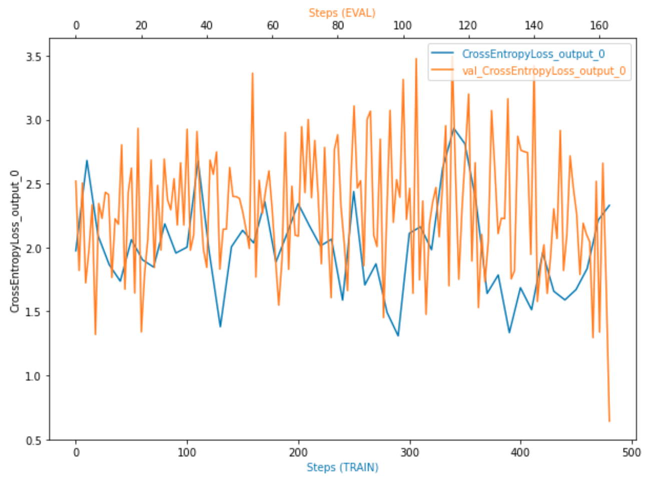
```
np.argmax(response2, 1)
```

```
array([2])
```

Since the model has 5 classes: `item_classes:['1', '2', '3', '4', '5']` and index of the response array is 2, meaning that `items_classes[2]= 3` which is the number of detected items.

Model Evaluation (before refinement)

With the similar dataset, we created another model as a benchmark for our model. The benchmark model is ResNet18, with 18 layers to classify the dataset. The CrossEntropyLoss was used as the metrics. Below shows the result of crossEntropyloss during training and testing the benchmark.



The table below shows a comparison between the results of the benchmark and our model.

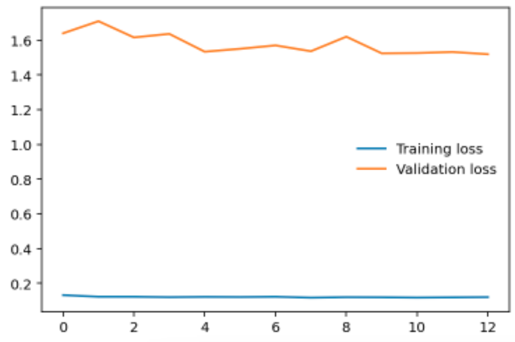
$$Test\ Loss \ = \frac{test\ loss}{number\ of\ test\ images} \qquad Test\ Accuracy \ = \frac{accuracy}{number\ of\ test\ images}$$

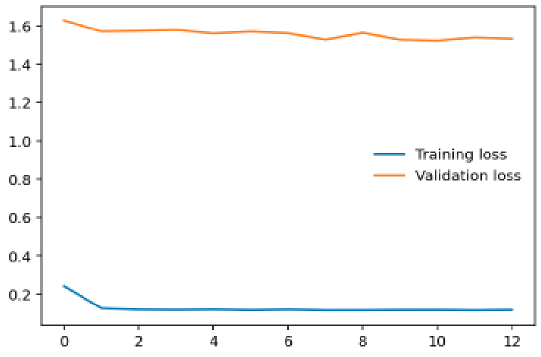
```
Model         Testing Loss    Testing Accuracy
--------  --------------  ------------------
ResNet18              36                   3
ResNet50              23                   4
```

As it can be seen in the table above, our model performed better in Testing loss as well as Testing Accuracy.

Model Evaluation (after refinement)

We used the refinement of the model for the benchmark to have a fair comparison between two different deep learning models with similar settings. Plots below show the Test loss and Test accuracy for both benchmark (left) and our model (right).





As it can be seen, the testing accuracy for both models are almost the same. From this experiment we can conclude that with good hyperparameter tuning and changing architecture, a model with a smaller number of layers can perform the same as a more complex model.

Justification

The plots that are provided for CrossEntropyLoss in both models show that there are great fluctuations in cross entropy loss for both train and evaluations, however, it is possible to reduce this issue with adding larger numbers of images and classify them for training and testing. Although our testing accuracy is small, the model performed well in counting the number of items in images with more visibility. Nevertheless, our model is sufficient for our project purpose.

5. Conclusion

In this project, we encountered some challenges. The first challenge was to acquire, process and partition the dataset. The next challenge was to choose the right machine learning approach. The third challenge was to evaluate the model with a new benchmark. Even Though there were some reports which could work as our benchmark, due to the different approach of data selection and data evaluation, we ended up creating our own simple benchmark.

Despite the challenges, completing this project was very fruitful and we gained a great experience in Machine learning and utilizing AWS tools and services.

Clearly, this project has lots of potential for improvements. Having more time and resources might help to achieve better results. For example, working on dataset preparations and pre-processing could yield a more accurate model.