MACHINE LEARNING ENGINEER CAPSTONE

Inventory Monitoring at Distribution Centers

By Eric Moreno

DEFINITION

Domain Background

A supply chain is a network between a business and the suppliers that make and distribute the products, supplies, or services the business needs to operate. There can be many players in a supply chain, including warehouses, trucking companies, retailers, distribution centers, and producers.

Regardless of a company's supply chain being large or small, inventory management is vital to a company's health because it balances supply with demand by ensuring that product is available at the right time by tracking product up and down the supply chain. Too much stock costs money and reduces cash flow and too little stock could lead to unfilled customer orders and lost sales.

Problem Statement

The largest operator of distribution centers is Amazon, which operates more than 175 fulfillment centers worldwide, with more than 150 million square feet of space. Amazon sells more than 12 million products and relying on manual processes to manage inventory for such a large operation would not be feasible and would be extremely costly. The advent of digital transformation has allowed many industries to benefit from artificial intelligence and this project aims to discover how artificial intelligence can be incorporated to improve inventory monitoring.

Datasets and Inputs

The dataset used for this project will be the Amazon Bin Image Dataset, which consists of 500,000 images of bins containing one or more objects from Amazon Fulfillment Centers. Each image is in JPEG format and contains corresponding JSON metadata files which describe the items in each bins.

Evaluation Metrics

Counting the number of items in a bin will be evaluated using accuracy.

Project Design

Below is an outline of the project design:

Data Preparation

- Download data and store in S3 bucket
- Split training data into train, test, and validation sets

Exploratory Data Analysis

- Explore the distribution of classes of images
- Identify any class imbalance

Model Training

- A training script will be created for data pre-processing and hyperparameter tuning, and will load a pre-trained ResNet50 model
- Data augmentation will be performed on the training set
- Hyperparameters for consideration are learning rate and batch size
- A training estimator will be set up

Model Evaluation

- Model performance will be evaluated using accuracy and RMSE

Deployment

- The model will be deployed to an endpoint
- A lambda function will be created which will connect to the endpoint and make inferences using a test image

ANALYSIS

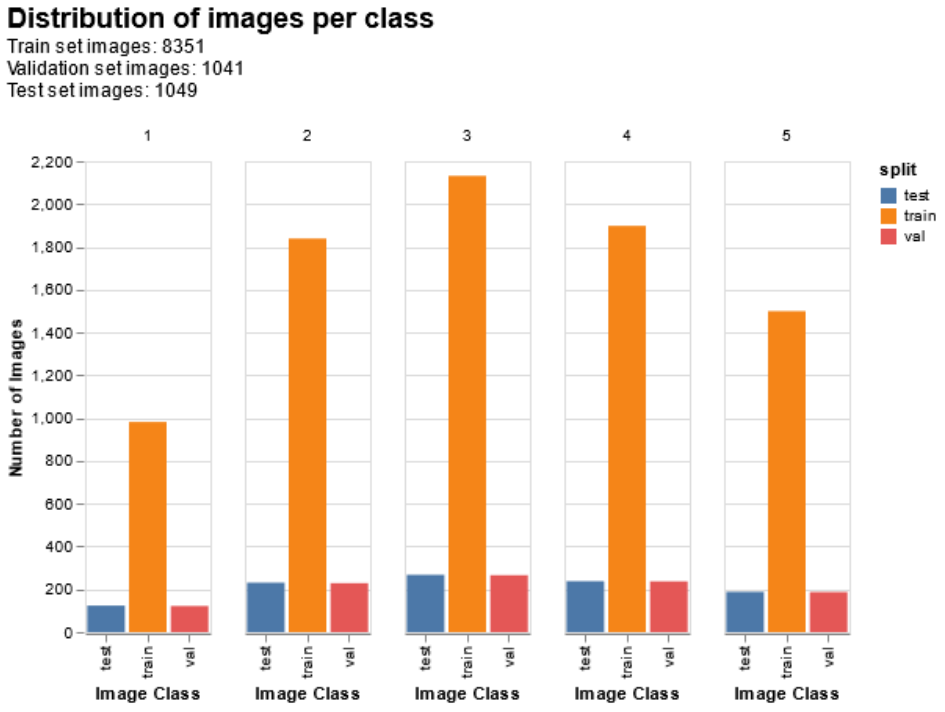
Data Exploration

When performing classification, it's good practice to identify class imbalance. We can see that there is some minor class imbalance, primarily within images with the label 1. Label 1 images make up only 12% of the dataset whereas the other labels make up anywhere from 18 to 26%.

	image_class	number_of_images	percentage
0	1	1228	0.12
1	2	2299	0.22
2	3	2666	0.26
3	4	2373	0.23
4	5	1875	0.18

Train-Validation-Split

The data was split into train, validation, and test sets using an 80/10/10 split, respectively. The visual below displays the number of images within each class for each split.



Algorithms and Techniques

A pre-trained ResNet50 model will be used, which will have a fully connected layer attached to the end of the model, followed by a ReLu activation function, followed by another Linear layer, and one final ReLu activation function.

```
model.fc = nn.Sequential( nn.Linear( num_features, 256),
                    nn.ReLU(inplace = True),
                    nn.Linear(256, 5),
                    nn.ReLU(inplace = True)
                    )
```

The reason why the ResNet50 model was chosen over other popular model types such as the VGG-16 model is because it is a 50-layer deep convolutional network. The RestNet50 model beats the limitation of the VGG-16 model by solving the issue of diminishing gradient. The ResNet architecture uses the concept of skip connections, allowing inputs to skip some convolutional layers and the result is a significant reduction in training time and improved accuracy.

Benchmark Model

The Amazon Bin Image Dataset (ABID) Challenge was conducted in 2017 and serves as a benchmark. Below are the results on the validation split performed by the initial challenge.

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

Bin 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

METHODOLOGY

Data Preprocessing

Minor data augmentation was performed on the training images by performing a random horizontal flip along with resizing and cropping. No augmentation was performed on the validation and test images, only resizing and cropping.

```
training_transform = transforms.Compose([
        transforms.RandomHorizontalFlip(p=0.5), # perform augmentation
        transforms.Resize(256),
        transforms.RandomResizedCrop((224, 224)),
        transforms.ToTensor() ])

testing_transform = transforms.Compose([
        transforms.Resize(256),
        transforms.RandomResizedCrop((224, 224)),
        transforms.ToTensor() ])
```

Implementation

The pre-trained ResNet50 model was implemented using the Torchvision library and below is the code used for the model, which can be found in the train.py and train_and_debug_model.py scripts. At the end of the pre-trained model there are two linear layers attached followed a ReLu activation function.

```
def net():
    model = models.resnet50(pretrained=True) # instantiate pretrained resnet50 model

    for param in model.parameters():
        param.requires_grad = False # freeze the convolutional layers of the pretrained layers by setting to false

    # find the number of features present in the pretrained model
    num_features = model.fc.in_features

    # add a fully connected layer to the end of our model
    model.fc = nn.Sequential( nn.Linear( num_features, 256),
                              nn.ReLU(inplace = True),
                              nn.Linear(256, 5), # set to 5 classes
                              nn.ReLU(inplace = True)
                             )
    return model
```

After creating the model, training was conducted using SageMaker Studio using the sagemaker.ipynb notebook and ml.t3.medium for the instance type.

Refinement

Hyperparameter Tuning

Instead of relying on specified hyperparameters, hyperparameter tuning was performed to search across the below search space. There seems to be a consensus among many that low batch sizes perform better than large batch sizes, which is why I opted for batch sizes of 32 and 64.

```
hyperparameter_ranges = {
        "lr": ContinuousParameter(0.001, 0.1),
        "batch_size": CategoricalParameter([32, 64]),
        "epochs": CategoricalParameter([8,12,15])
}
```

In order to speed up training, max_jobs was set to 4 and max_parallel_jobs was set to 2 along with using ml.g4dn.xlarge as the instance type.

Training Job Objective Metric

The tuning job created via the HyperparameterTuning object in SageMaker allowed for the use of objective metrics, which were used during tuning. The objective during training was to **minimize the Test Loss** metric.

```
objective_metric_name = "Test Loss"
objective_type = "Minimize"
metric_definitions = [{"Name": "Test Loss", "Regex": "Testing Loss: ([0-9\\.]+)"}]
```

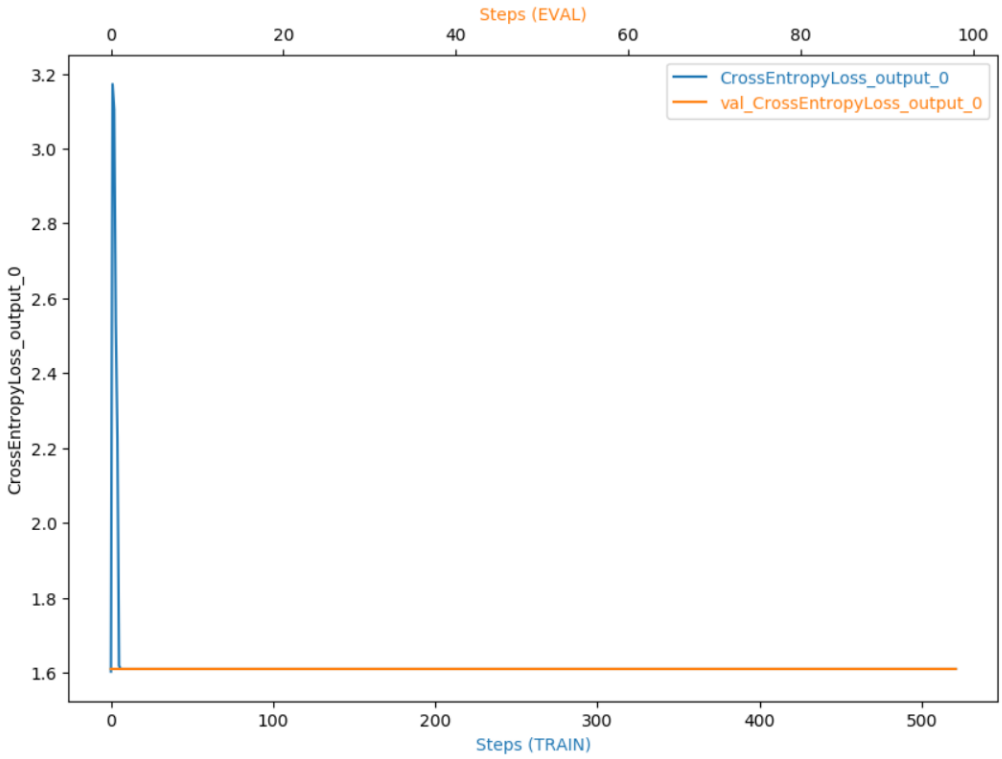
After the training jobs completed, the below hyperparameters were found to be optimal and were used to train the model before it was deployed to the endpoint.

```
{'batch_size': 32, 'lr': 0.003309487406915597, 'epochs': 12}
```

RESULTS

Model Evaluation and Validation

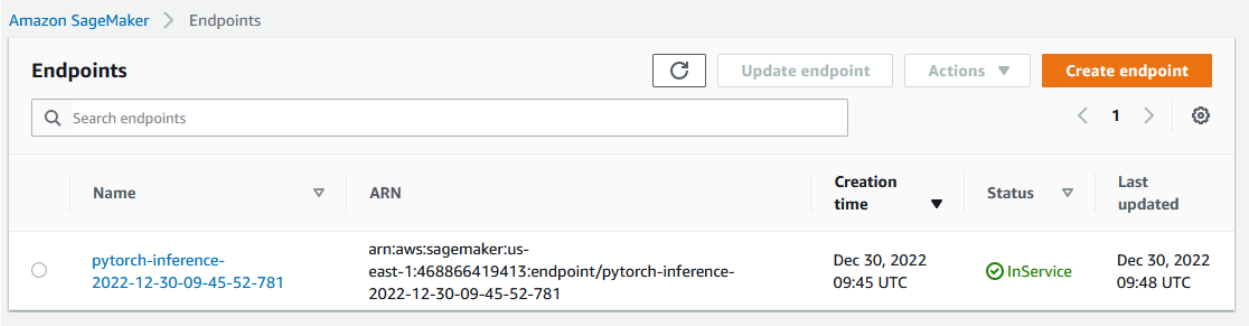
Unfortunately, despite using a pre-trained ResNet50 model and hyperparameter tuning, accuracy for this model was only 11.8%. This is significantly lower than the benchmark of 55.67%. It could be the case that I added too many fully connected layers at the end of the model, which I will explore down the road.



Model Deployment

The model was deployed to an endpoint and randomly sampled images from the test set were used for testing. Below is an example of a prediction generated from the endpoint. The randomly sampled image has a label of 1 but the predicted label was zero. I believe that with refinement to the CNN model and improved training that the results can be improved.

```
Sample image selected for image class: 1
Image path: data/test/1/102821.jpg
Image label: 1
Predicted label: 0
```



Sources

- https://www.bigrentz.com/blog/amazon-warehouses-locations
- https://www.netsuite.com/portal/resource/articles/inventory-management/inventory-management.shtml
- https://registry.opendata.aws/amazon-bin-imagery/
- https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
- https://datagen.tech/guides/computer-vision/vgg16/