

Capstone Project Report

Inventory Monitoring at Fullfillment Centers using Sagemaker

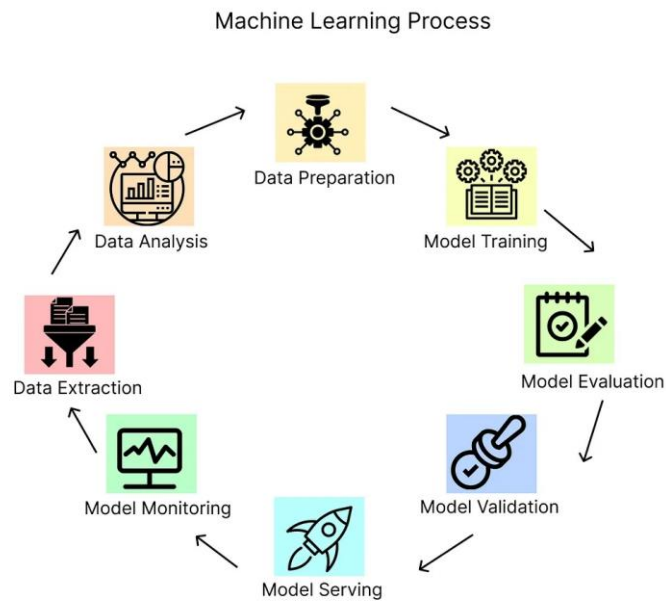
1. Domain Background:

Fulfillment Centers have bin locations for storing goods as it simplifies inventory management processes. Robots often move objects as part of their operations. Objects are carried to bin locations which can contain more than one object. Whole purpose of these bin locations is to organise the objects so they can be easily retrieved when searched. However, the problem is, there is still some manual work being done i.e., someone still needs to count the number of objects in a particular bin location and it can become troublesome as number of objects are constantly changing in bin locations so count of objects will also change.

2. Problem Statement:

As mentioned earlier, one way to go around this problem is to count each object manually in the bin or another way might be to keep track of the count every time an object is removed or placed in the bin. However, a much simpler way of go about this problem might be to use a machine learning model that can count number of objects in each bin.

3. Machine Learning Pipeline:



Machine Learning pipeline typically include these steps:

1. Data Collection
2. Data Pre-processing
3. Model Training
4. Model Testing
5. Model Deployment
6. Model Monitoring

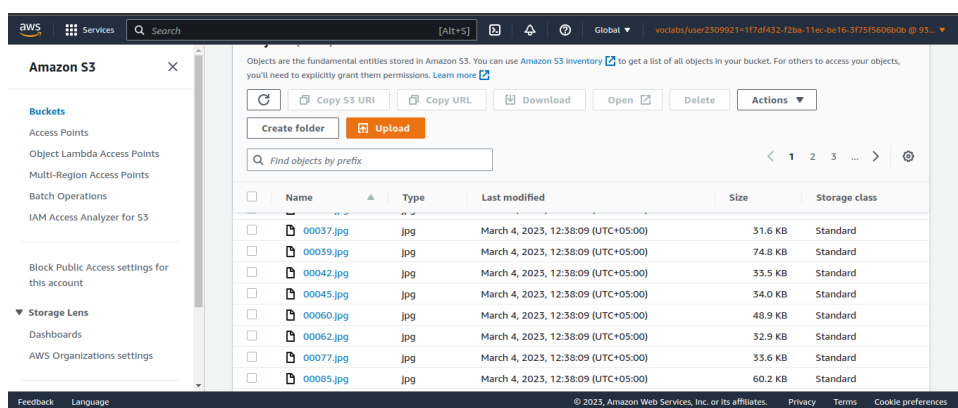
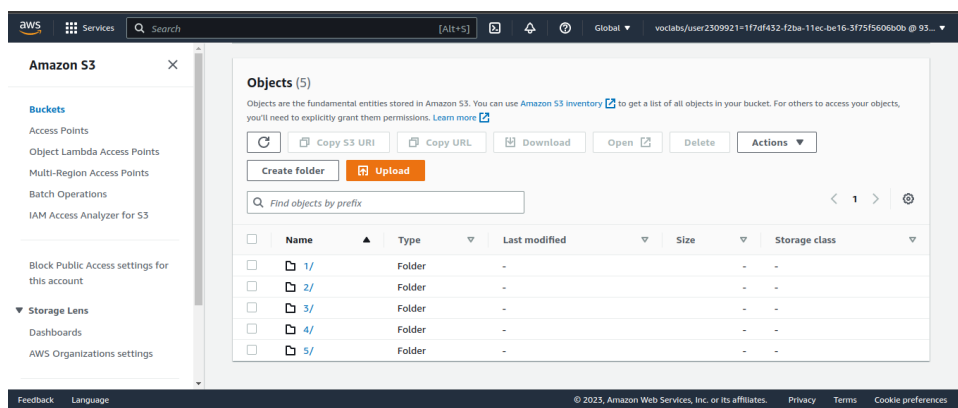
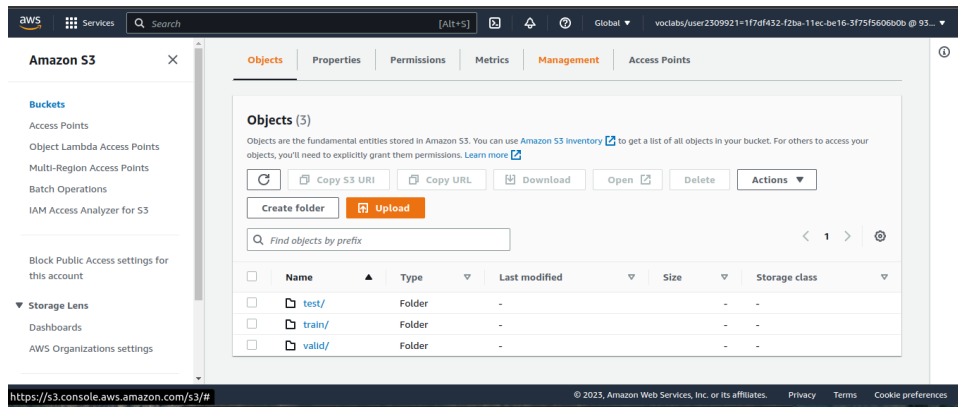
3.1 Data Collection:

Data Collection is preliminary and crucial step in machine learning pipeline. It is crucial because it is a tiresome process and the efficacy of model also depends on data being collected in this step. Anyway, for this problem data was already collected so I did not have to go down that road.

3.2 Data Pre-processing:

Next step of Machine Learning Pipeline is to Data pre-processing. Data is pre-processed before it is sent to the model for training.

In this project, I first downloaded subset of the dataset form s3 bucket where data was hosted. Then, I copied the downloaded data from current directory to my own s3 bucket. Once, it is copied to s3 it implemented training script. In that training script, I defined transformations for the data.



These transformations include:

For training:

- a. **Resize:** Resize image to (224,224) because model takes this size as input.
- b. **Random_Rotation:** for diversifying dataset
- c. **Random_horizontal_flip:** for diversifying dataset.
- d. **Random_Resized_Crop:** for generalizing model and diversifying dataset
- e. **ToTensor:** for converting dtype of image data to tensor.
- f. **Normalize:** Normalizing image data so all data lie in same range

For testing:

- a. **Resize:** Resize image to (224,224) because model takes this size as input.
- b. **ToTensor:** for converting dtype of image data to tensor.
- c. **Normalize:** Normalizing image data so all data lie in same range

Below image show transformations and order in which they were applied.

```

170
171     train_transform = transforms.Compose([
172         transforms.Resize((224, 224)),
173         transforms.RandomHorizontalFlip(),
174         transforms.RandomRotation(10),
175         transforms.RandomResizedCrop(size=224, scale=(0.8, 1.0)),
176         transforms.ToTensor(),
177         transforms.Normalize(mean = [0.485, 0.456, 0.406],
178                               std = [0.229, 0.224, 0.225])
179     ])
180
181     test_transform = transforms.Compose([
182         transforms.Resize((224, 224)),
183         transforms.ToTensor(),
184         transforms.Normalize(mean = [0.485, 0.456, 0.406],
185                               std = [0.229, 0.224, 0.225])
186     ])
187

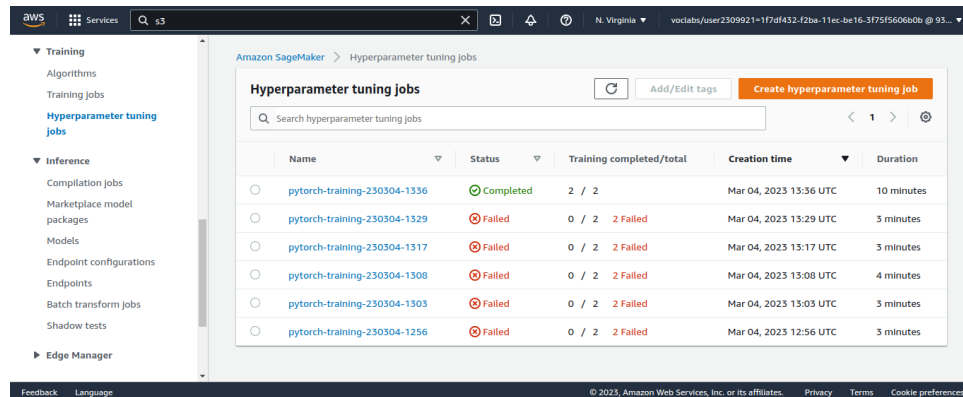
```

3.3 Model Training:

Once Data Pre-processing step is done, we write script for model training. Here we define model architecture. I used pretrained resnet18 model. One of many advantages of using pretrained model is, it takes less time for transfer learning i.e., training on custom data. I used pretrained weights of resnet18. I only changed number of neurons in output layer because this is the prediction layer. I only used 5 neurons in output layer because my dataset had only five classes. I trained my model for 2 epochs. Batch_size I used for training was 64 i.e., 64 image are passed

through the network in one go and learning rate was set to (0.0015184292606730283)
I.e., rate at which weights are updated.

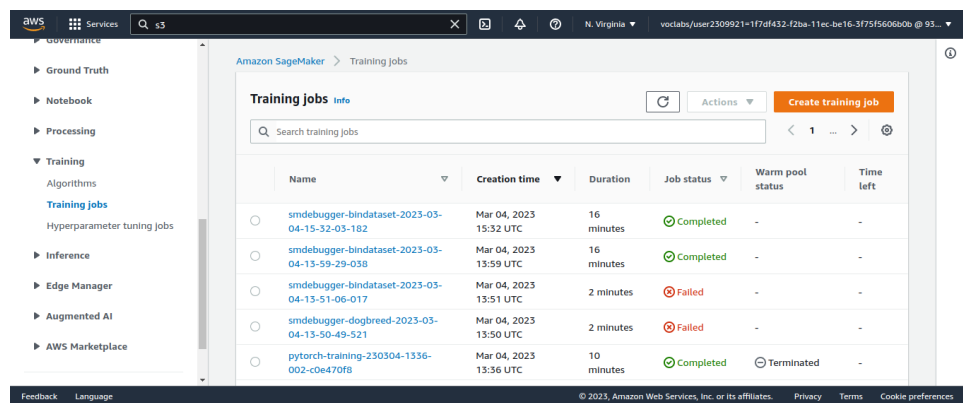
Below image shows hyperparameter tuning jobs.



The screenshot shows the Amazon SageMaker console for Hyperparameter tuning jobs. The left sidebar lists various SageMaker services, with 'Hyperparameter tuning jobs' selected. The main panel displays a table of jobs with columns for Name, Status, Training completed/total, Creation time, and Duration. One job is completed, and five others have failed.

Name	Status	Training completed/total	Creation time	Duration
pytorch-training-230304-1336	Completed	2 / 2	Mar 04, 2023 13:36 UTC	10 minutes
pytorch-training-230304-1329	Failed	0 / 2 2 Failed	Mar 04, 2023 13:29 UTC	3 minutes
pytorch-training-230304-1317	Failed	0 / 2 2 Failed	Mar 04, 2023 13:17 UTC	3 minutes
pytorch-training-230304-1308	Failed	0 / 2 2 Failed	Mar 04, 2023 13:08 UTC	4 minutes
pytorch-training-230304-1303	Failed	0 / 2 2 Failed	Mar 04, 2023 13:03 UTC	3 minutes
pytorch-training-230304-1256	Failed	0 / 2 2 Failed	Mar 04, 2023 12:56 UTC	3 minutes

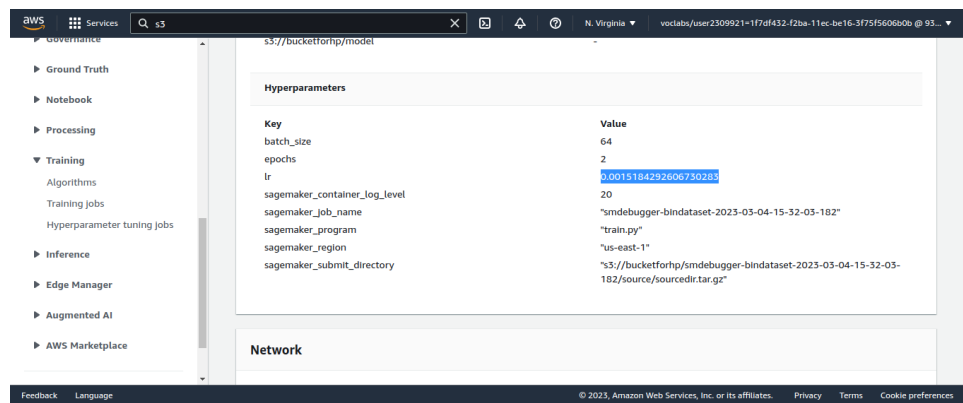
Below image shows training jobs



The screenshot shows the Amazon SageMaker console for Training jobs. The left sidebar lists various SageMaker services, with 'Training jobs' selected. The main panel displays a table of jobs with columns for Name, Creation time, Duration, Job status, Warm pool status, and Time left. Five jobs are listed, with three completed, two failed, and one terminated.

Name	Creation time	Duration	Job status	Warm pool status	Time left
smdebugger-bindataset-2023-03-04-15-32-03-182	Mar 04, 2023 15:32 UTC	16 minutes	Completed	-	-
smdebugger-bindataset-2023-03-04-15-59-29-038	Mar 04, 2023 15:59 UTC	16 minutes	Completed	-	-
smdebugger-bindataset-2023-03-04-13-51-06-017	Mar 04, 2023 13:51 UTC	2 minutes	Failed	-	-
smdebugger-dogbreed-2023-03-04-13-50-49-521	Mar 04, 2023 13:50 UTC	2 minutes	Failed	-	-
pytorch-training-230304-1336-002-c0e470f8	Mar 04, 2023 13:36 UTC	10 minutes	Completed	Terminated	-

Below image shows hyperparameters inside the training job.



3.4 Model Testing:

Once Training is done model is tested on test data to check how well the model is performing. I tested my model on test data after every epoch to see whether model accuracy was increasing. After first epoch, model accuracy was 19% and after second epoch model accuracy was 22%. If I were to let it train for more epochs its accuracy will further improve.

```
to": "resource_config":{"current_group_name":"homogeneouscluster","current_host":"algo-1","current_instance_type":"ml.m5.xlarge","hosts":["algo-1"],"instance_groups":[{"hosts":["algo-1"],"instance_group_name":"homogeneouscluster","instance_type":"ml.m5.xlarge"}],"network_interface_name":"eth0"},"user_entry_point":"train.py"}
SM_USER_ARGS="--batch_size","64","--epochs","2","--lr","0.0015184292606730283"}
SM_OUTPUT_INTERMEDIATE_DIR=/opt/ml/output/intermediate
SM_CHANNEL_TRAINING=/opt/ml/input/data/training
SM_HP_BATCH_SIZE=64
SM_HP_EPOCHS=2
SM_HP_LR=0.0015184292606730283
PYTHONPATH=/opt/ml/code:/opt/conda/bin:/opt/conda/lib/python3.6:/opt/conda/lib/python3.6/site-packages
Invoking script with the following command:
/opt/conda/bin/python3.6 train.py --batch_size 64 --epochs 2 --lr 0.0015184292606730283
Namespace(batch_size=64, data='/opt/ml/input/data/training', epochs=2, lr=0.0015184292606730283, model_dir='/opt/ml/model', output_dir='/opt/ml/output/data')
Hyperparameters are LR: 0.0015184292606730283, Batch Size: 64
Data Paths: /opt/ml/input/data/training
[2023-03-04 12:40:19.315 algo-1:46 INFO json_config.py:98] Creating hook from json_config at /opt/ml/input/config/debughookconfig.json.
[2023-03-04 12:40:19.315 algo-1:46 INFO hook.py:192] tensorboard_dir has not been set for the hook. SMDebug will not be exporting tensorboard summaries.
[2023-03-04 12:40:19.315 algo-1:46 INFO hook.py:237] Saving to /opt/ml/output/tensors
[2023-03-04 12:40:19.315 algo-1:46 INFO state_store.py:67] The checkpoint config file /opt/ml/input/config/checkpointconfig.json does not exist.
[2023-03-04 12:40:19.775 algo-1:46 INFO hook.py:382] Monitoring the collections: losses
[2023-03-04 12:40:19.776 algo-1:46 INFO hook.py:443] Hook is writing from the hook with pid: 46
Train Epoch: 0 (0/8351 (0%)) Loss: 110.553467
Train Epoch: 0 (6400/8351 (76%)) Loss: 104.574036
Test set: Average loss: 1.6375, Accuracy: 204/1049 (19%)
Saving the model.
Train Epoch: 1 (0/8351 (0%)) Loss: 105.826630
Train Epoch: 1 (6400/8351 (76%)) Loss: 105.302850
Test set: Average loss: 1.6345, Accuracy: 236/1049 (22%)
Saving the model.
2023-03-04 12:52:58,980 sagemaker-training-toolkit INFO Reporting training SUCCESS
```

3.5 Model Deployment:

Once the model is trained and accuracy of model is satisfactory it can be used to make predictions on new data. I deployed my trained model to an endpoint once the model was trained. Once it is deployed it can be invoked for inference.

```
Model Deploying and Querying

In [11]: # boto3 client for s3
from sagemaker.pytorch import PyTorchModel
bucket = "bucketforlog"
prefix = "model/estimator-bundataest-2023-03-04-13-59-29-03/output"
model = PyTorchModel()
entry_point = "inference.py",
source_dir = ".",
role = role,
model_data = s3://{{{model_tar_gz}}}/model.tar.gz:format(bucket, prefix),
framework_version = "1.8",
py_version = "py36",
}

predictor = model.deploy(initial_instance_count=1, instance_type="ml.t2.medium")

.....

In [12]: # Running a prediction on the endpoint
import boto3
import numpy as np
from PIL import Image
import json

s3 = boto3.client('sagemaker-runtime')

# Load the image as a NumPy array
arr = np.array(Image.open('image.jpg'))

# arr2 = np.arange(0, arr)

# # Create a dictionary containing the NumPy array
data = {'arr': arr.tolist()}

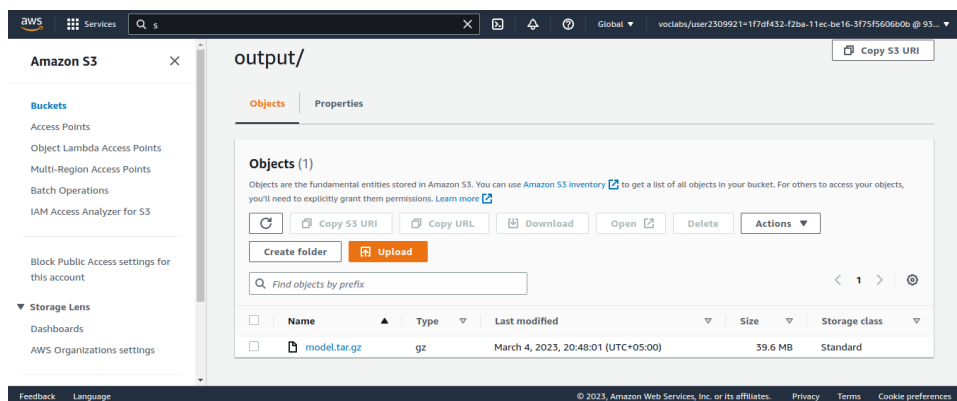
endpoint_name = "pytorch-inference-2023-03-04-15-17-38-379"

response = s3.invoke_endpoint(endpoint_name=endpoint_name,
                             body=json.dumps(data),
                             ContentType='application/json')

response = response['Body'].read().decode('utf-8')
print(response)

{"body": [{"prob": [9.389663015449961, 0.10786523282527924, 0.1731564900002536, 0.17810502363403005, 0.1490754336118698], "indices": [1, 4, 2, 3, 5, 13], "class": [1, 2, 8, 2, 1, 2, 2, 4, 1, 5, 4]}]}
```

For model to make inference we need model artifact file where trained weights are stored. This file was created when estimator.fit() was called and trained weights were stored in this file.



This file plays a significant role in deploying an endpoint because this file contains weights and to make prediction we need to load these weights and return prediction.

Other than model artifact file, we also need an entry point. This entry point is a python script. This script contains all the necessary function needed to make prediction. When endpoint is invoked, model_fn is called. This model_fn creates model object and loads the trained weights into the model. After calling model_fn, input_fn is called. This function takes input data and perform necessary pre-processing on it. Once pre-processing is done predict_fn is called. This function predicts the class of input data and returns the prediction. Then output_fn is called which convert prediction to json object and return the json object.

Below images show inference.py which acts as an entry point for PytorchModel object for deployment of endpoint.

```
26
27 def input_fn(request_body, request_content_type):
28     """
29     Deserialize and prepare the prediction input
30     """
31
32     if request_content_type == "application/json":
33
34         deserialized_data = json.loads(request_body)
35
36         plt.imshow("image.png", deserialized_data['arr'])
37
38         data = Image.open("image.png").convert('RGB')
39
40         test_transform = transforms.Compose([
41             transforms.Resize((224, 224)),
42             transforms.ToTensor(),
43             transforms.Normalize(mean = [0.485, 0.456, 0.406],
44                                 std = [0.229, 0.224, 0.225])
45         ])
46
47         train_inputs = test_transform(data)
48
49         return train_inputs
50
51
```

```
90 def net():
91     """
92     This function takes zero parameters and returns a Network
93
94     Parameters:
95         None
96
97     Returns:
98         Untrained Image Classification Model
99     """
100
101     pretrained_model = models.resnet18(pretrained=True)
102
103     # Freezing Pretrained Weights
104     for param in pretrained_model.parameters():
105         param.requires_grad = False
106
107     # Append Fully Connected layer
108     num_fcfs = pretrained_model.fc.in_features
109     pretrained_model.fc = nn.Linear(num_fcfs, 5)
110
111     model_ft = pretrained_model.to(device)
112
113     return model_ft
114
115
116 def model_fn(model_dir):
117     model = net()
118     with open(os.path.join(model_dir, 'model.pth'), 'rb') as f:
119         checkpoint = torch.load(f)
120         model.load_state_dict(checkpoint['model_state_dict'])
121         class_to_id = checkpoint['class_to_id']
122
123     return ("model": model, "class": class_to_id)
124
```

```
52 def predict_fn(input_data, model):
53     """
54     Apply model to the incoming request
55     """
56     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
57     model['model'].to(device)
58     input_data = torch.unsqueeze(input_data, 0).to(device)
59     model['model'].eval()
60     with torch.no_grad():
61         return [{"prediction": model['model'](input_data), "class": model['class']}]
62
63
64
65 def output_fn(prediction_output, response_content_type):
66     if response_content_type == "application/json":
67         result = nn.functional.softmax(prediction_output['prediction'], dim=-1)
68
69         prob = torch.topk(result, 5)[0][0].tolist()
70         indices = torch.topk(result, 5)[1][0].tolist()
71
72         for i in range(len(indices)):
73             for key, val in prediction_output['class'].items():
74                 if indices[i] == val:
75                     indices[i] = key
76
77         temp = {"prob": prob, "indices": indices, "class": prediction_output['class']}
78
79         data = {'body': temp}
80
81
82         # Serialize the data using the JSONSerializer
83         serialized_data = json.dumps(data)
84
85         return serialized_data
86
87
```

3.6 Model Monitoring:

Once the model is deployed, it is monitored for any bottlenecks. Bottlenecks usually occur because of lambda functions or endpoints so we can set concurrency for lambda functions and autoscaling for endpoint so not bottlenecks occurs and ensuring user experience.

4. Results:

Results can be improved by either increasing the number of epochs, or using a different pretrained model or appending more fully connected layers before the output layer. Anyway, with current setting I got an accuracy of 22%.

5. Conclusion:

In conclusion, this capstone project aimed to solve the problem of inventory monitoring at fulfillment centres by using a machine learning model. The model was trained using a subset of a pre-existing dataset, and the data was pre-processed and transformed for training and testing. The model used was a pre-trained ResNet18 model, and the number of neurons in the output layer was changed to accommodate the number of classes in the dataset. The model was trained for 2 epochs with a batch size of 64 and a learning rate of (0.0015184292606730283).

After training, the model was tested on the test data, and the accuracy of the model was found to be 19% after the first epoch and 22% after the second epoch. The model was then deployed to an endpoint, and inference was made by invoking the endpoint. The model performed well in making predictions on new data.

Overall, this project demonstrated the efficacy of using machine learning models for inventory monitoring at fulfillment centres, and it can be further improved by collecting more data, increasing the number of epochs for training, and fine-tuning the hyperparameters.