Capstone Project Report

Inventory_Monitoring_at_Fullfillment_Centers_using_Sagemaker

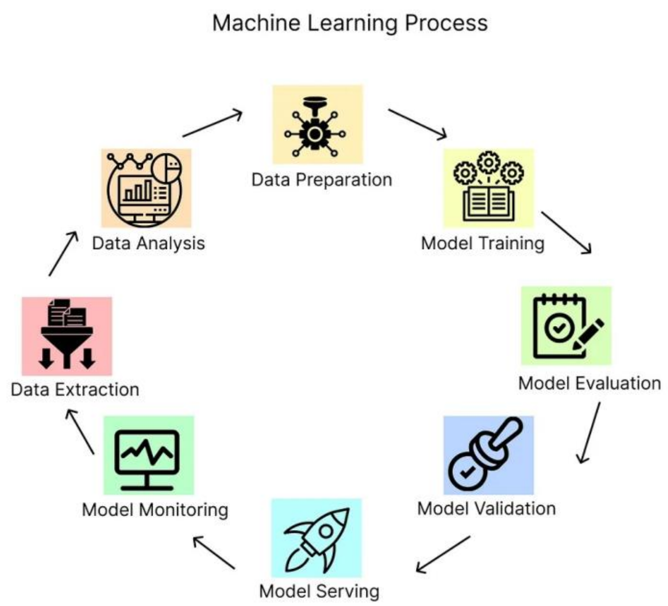
1. **Domain Background:**

   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

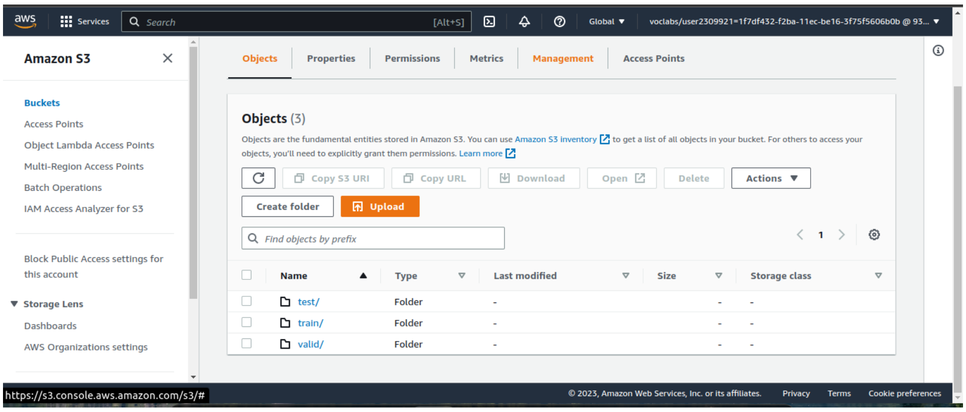
3.1 Data Collection:

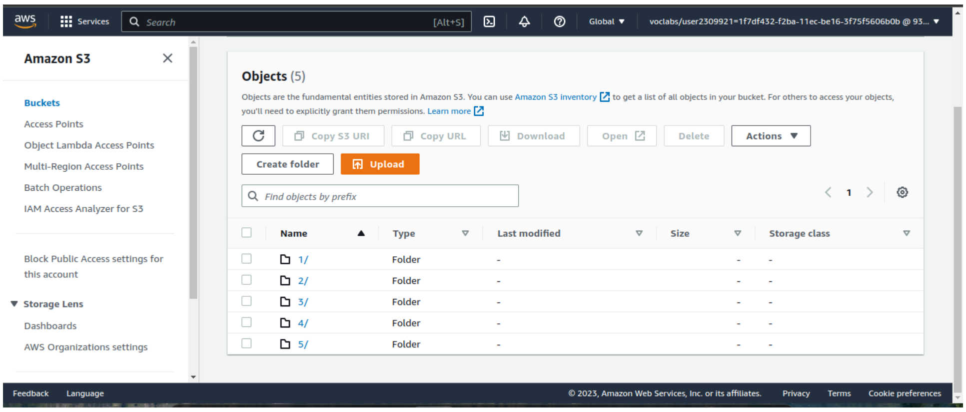
Data Collection is preliminary and crucial step in machine learning pipeline. It is crucial because it is a tiresome process and the efficacy of model also depends on data being collected in this step. Anyway, for this problem data was already collected so I did not have to go down that road.

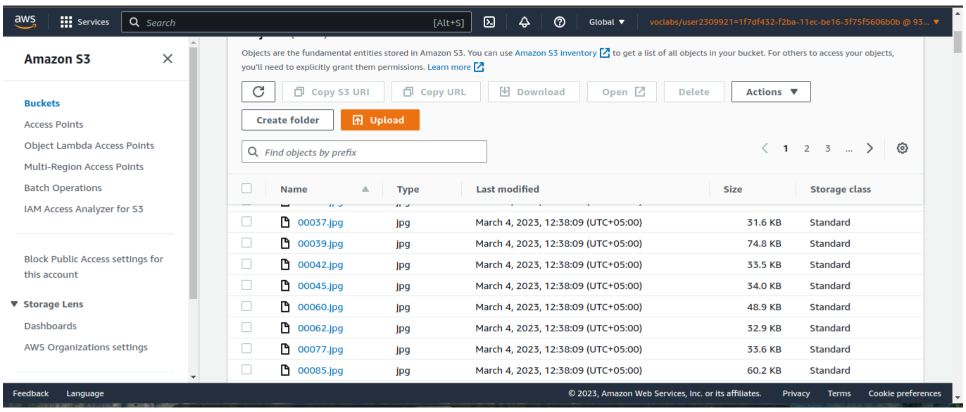
3.2 Data Pre-processing:

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.

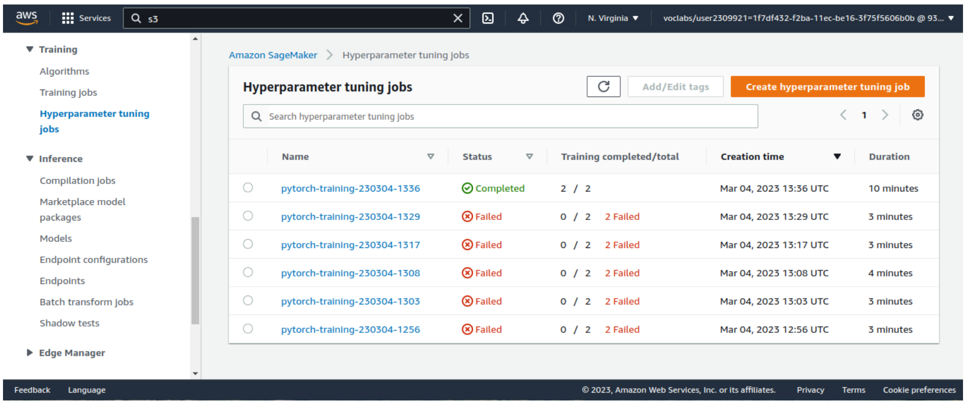
```
train_transform = transforms.Compose([
            transforms.Resize((224, 224)),
            transforms.RandomHorizontalFlip(),
            transforms.RandomRotation(10),
            transforms.RandomResizedCrop(size=224,scale=(0.8, 1.0)),
            transforms.ToTensor(),
            transforms.Normalize(mean = [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])
            ])
```

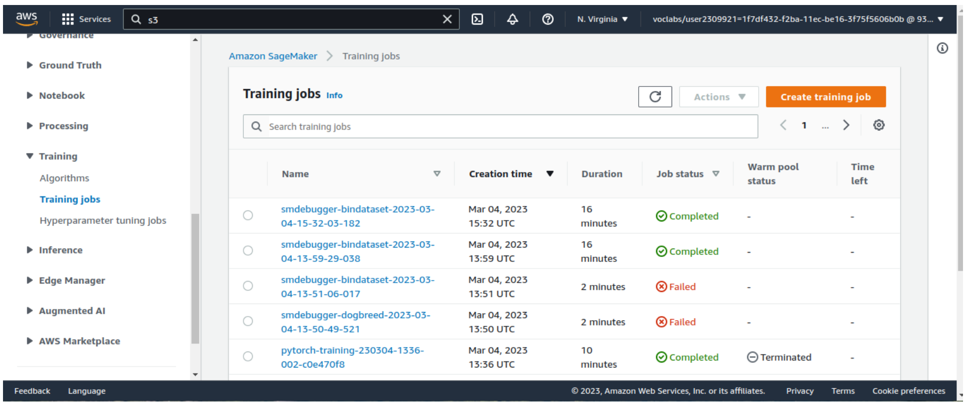
3.3 Model Training:

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.

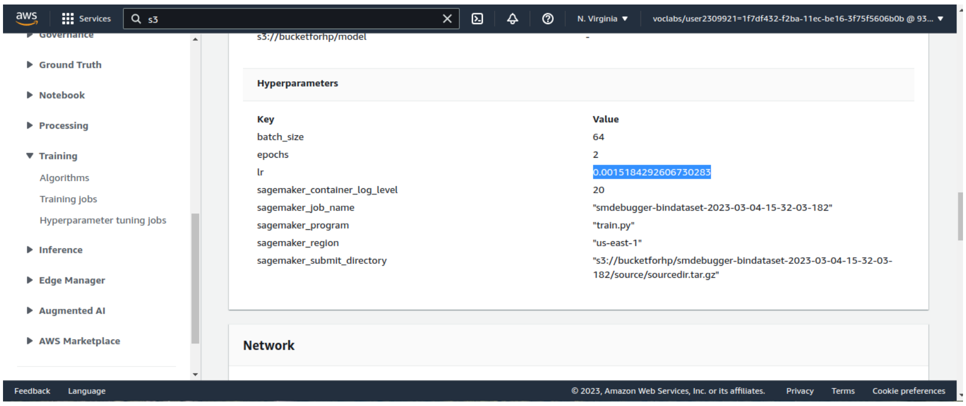
Below image shows hyperparameter tuning jobs.



Below image shows training jobs

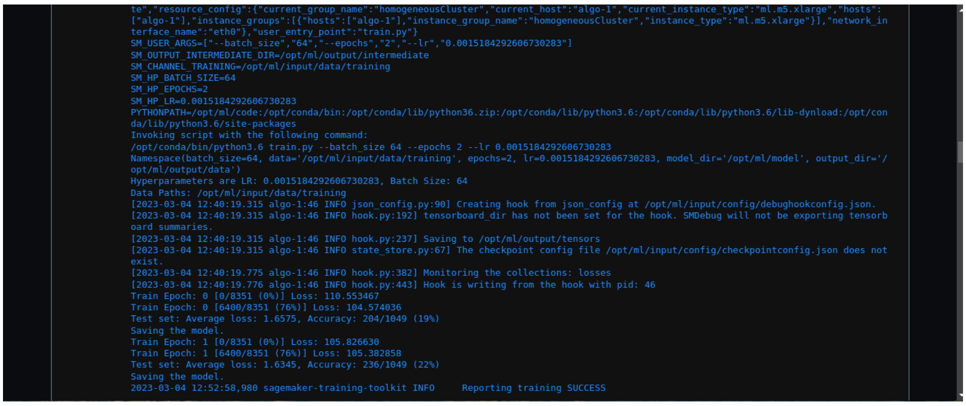


Below image shows hyperparameters inside the training job.



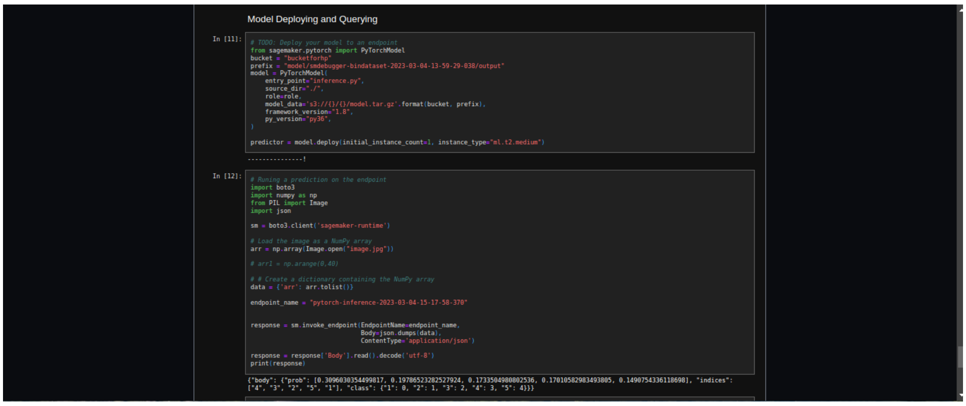
3.4 Model Testing:

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.

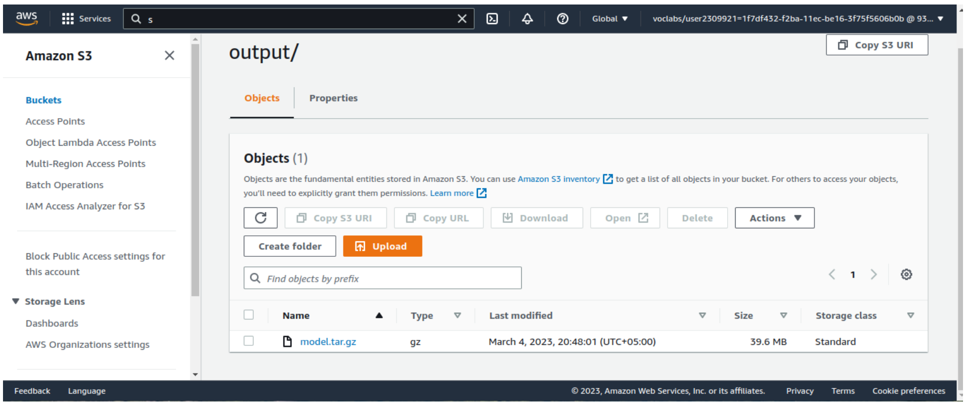


3.5 Model Deployment:

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.



For model to make inference we need mode 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 enpoint.

```python
def input_fn(request_body, request_content_type):
    """
    Deserialize and prepare the prediction input
    """

    if request_content_type == "application/json":

        deserialized_data = json.loads(request_body)

        plt.imsave("image.png",deserialized_data['arr'])

        data = Image.open("image.png").convert('RGB')

        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])
            ])


        train_inputs = test_transform(data)

        return train_inputs
```

```python
def net():
    '''
    This function takes zero parameters and returns a Network

    Parameters:
        None

    Returns:
        Untrained Image Classification Model

    '''
    pretrained_model = models.resnet18(pretrained=True)

    # Freezing Pretrained Weights
    for param in pretrained_model.parameters():
        param.requires_grad = False

    # Append Fully_Connected layer
    num_ftrs = pretrained_model.fc.in_features
    pretrained_model.fc = nn.Linear(num_ftrs, 5)

    model_ft = pretrained_model.to(device)

    return model_ft



def model_fn(model_dir):
    model = net()
    with open(os.path.join(model_dir, 'model.pth'), 'rb') as f:
        checkpoint = torch.load(f)
        model.load_state_dict(checkpoint['model_state_dict'])
        class_to_idx = checkpoint['class_to_idx']

    return {"model":model,"class":class_to_idx}
```

```python
def predict_fn(input_data, model):
    """
    Apply model to the incoming request
    """
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model['model'].to(device)
    input_data = torch.unsqueeze(input_data,0).to(device)
    model['model'].eval()
    with torch.no_grad():
        return {"prediction": model['model'](input_data),"class": model['class']}


def output_fn(prediction_output, response_content_type):
    if response_content_type == "application/json":
        result = nn.functional.softmax(prediction_output['prediction'],dim=1)

        prob = torch.topk(result, 5)[0][0].tolist()
        indices = torch.topk(result, 5)[1][0].tolist()

        for i in range(len(indices)):
            for key, val in prediction_output['class'].items():
                if indices[i] == val:
                    indices[i] = key


        temp = {"prob":prob,"indices":indices,"class":prediction_output['class']}


        data = {'body': temp}

        # Serialize the data using the JSONSerializer
        serialized_data = json.dumps(data)

        return serialized_data
```

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.