Capstone Proposal - Inventory Monitoring at Distribution Centres

Betty R Mtengwa

4 November 2024

Background

This capstone project proposal is part of the fulfilment of AWS Machine Learning Engineer Nanodegree with Udacity. The aim of the project is to serve as a demonstration of how an end-to-end machine learning model would look like.

A traditional way of managing inventory has proved to have several challenges as the systems often struggle to adapt quickly to changes in demand and the system provide insufficient information on real-time stock levels (StoreFeeder, 2024). As the business is not able to track its stock and information is not up to date, this can result in loss of sales. In addressing these challenges businesses have turned to distributed inventory management which is a modern way of managing inventory stock. Amazon distribution centres have adopted the modern way of managing inventory and this helps to track products from obtaining the stock to customer shipment. One of the modern ways they have adopted is by using robots to move objects as part of their operations in a day-to-day business. However, they is need to check how many items have been placed in each bin to track inventory and making sure that delivery consignments have the correct number of items. This is carried out using a pre-trained convolutional neural network machine learning model. Amazon Bin Image Dataset is a dataset that contains images of bins showing items that have been placed by the robot. The data is publicly accessible. The dataset is large, and it is time consuming to physically assess each picture, hence we are going to train a machine learning model and this also saves time.

To complete this project, I used AWS SageMaker to train and test the model.

Problem Statement

The aim of the project is to build a model that can count the number of objects that are placed in each bin by the robots that helps to move objects around the factory. The model should be able to accurately predict the number of objects in a bin and also identify false prediction.

Dataset Description

The Amazon Bin Image dataset contains almost over 500, 000 images of bins in a pod in an operating Amazon Fulfilment Centre. Amazon Fulfilment Centres are known for delivering millions of products to customers worldwide with the help of robots and computer vision technologies. The images were captured when the robots where in operation. As the dataset is quite large, a sample of 10, 441 images will be used. The data will be further divided into three portions 60% for training, 20% for testing and the remaining 20% for validation.

Data Preparation

The first step was to set the root directory ‘final_project’ where downloaded images were going to be stored. The dataset was downloaded and placed in each folder based on the number of objects in a picture. The model mainly focused on pictures that had 1 to 5 objects in them as follows (and as shown in the screenshot below);

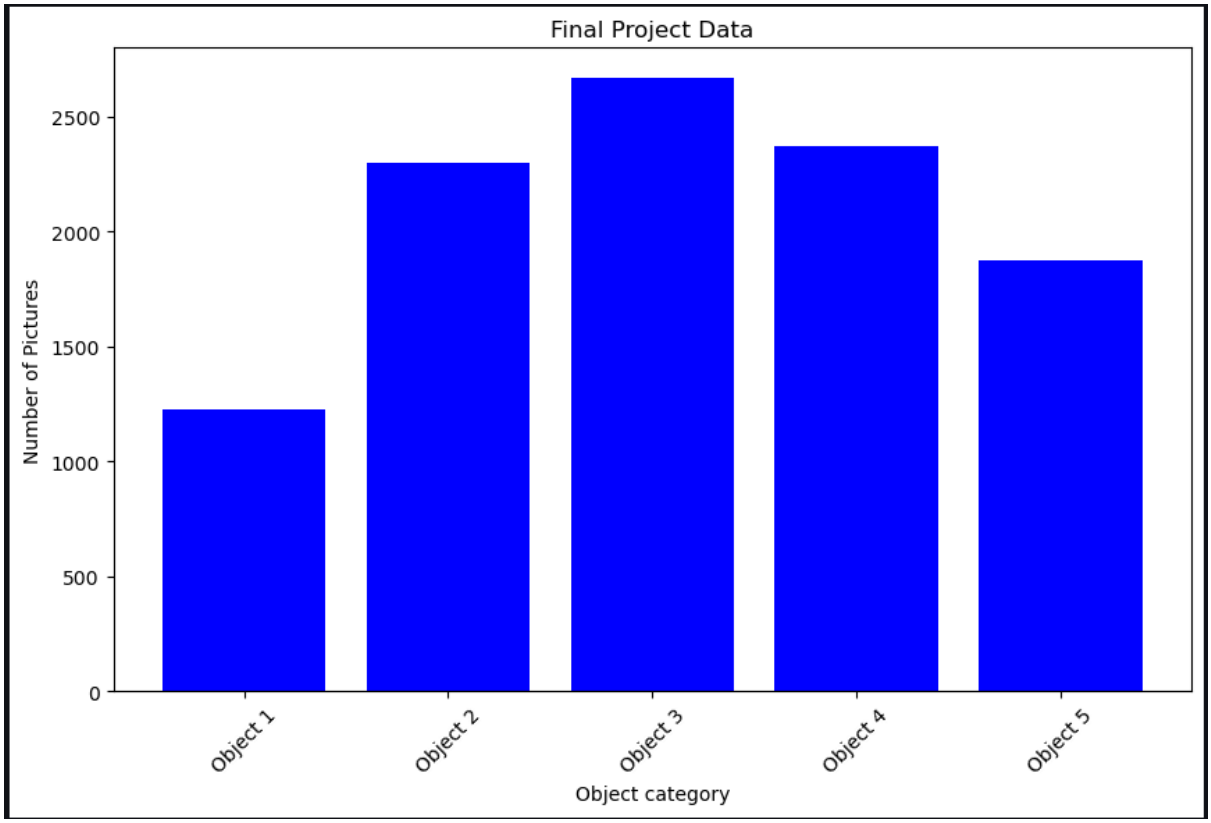
Class 1 had pictures with 1 object and there were 1228 pictures

Class 2 had pictures with 2 objects and there were 2299 pictures

Class 3 had pictures with 3 objects and there were 2666 pictures

Class 4 had pictures with 4 objects and there were 2373 pictures

Class 5 had pictures with 5 objects and there were 1875 pictures.



The data was further divided into train, test and validation with a ratio of 0.60, 0.20,0.20 respectively. Screenshot below details how data was further was divided within each object container.

```
num items 1: Test: 736, Train: 1200, Valid: 245
num items 2: Test: 1379, Train: 1200, Valid: 459
num items 3: Test: 1599, Train: 1200, Valid: 533
num items 4: Test: 1423, Train: 1200, Valid: 474
num items 5: Test: 1125, Train: 1200, Valid: 375
```

The pictures were then uploaded to s3 bucket for the model to be trained on AWS.

Resizing the Image

For benchmaking the model, I resized the images by 224 *224. I resized the images so that they will all have consistent size and same dimensions. I converted the images to tensorflow format as it enables efficient data handling when training the model. The benchmark is implemented as shown below;

```
train_transform = transforms.Compose([
    transforms.RandomResizedCrop((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    ])

test_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    ])

validation_transform = transforms.Compose([
    transforms.RandomResizedCrop((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    ])
```

Model Development

Hyperparameter tuning

During the model training process, I decided to tune hyperparameters. Hyperparameters are used to control the learning process and define model parameters that the algorithm will used during learning. I set the values before I begin the model training. The hyperparameters were tuned as follows;

- learning_rate : 0.005
- batch_size: 6
- epochs: 5

Standout Suggestions

As standout suggestions I tried different hyperparameters so that I can find the best combination of them. This allows one to get a more accurate model. The hyperparameters were tuned as follows,

- learning_rate: ContinuousParameter(0.001, 0.1),
- batch_size : CategoricalParameter([32, 64, 128, 256])

Model Profiling and Debugging

The main reason why its important to go through model profiling and debugging is because we are able to establish the following,

- Able to understand how our model is training
- Being able to identify training issues such as overfitting.

I firstly imported Amazon Sagemaker Debugger and added hooks to the train and test model. I created Hooks and registered the model as follows,

```
import smdebug.pytorch as smd
from sagemaker.pytorch import PyTorch
from sagemaker import get_execution_role
from sagemaker.debugger import (
    Rule,
    DebuggerHookConfig,
    rule_configs,
)
from sagemaker.debugger import Rule, ProfilerRule, rule_configs
```

```
# TODO: Create and fit an estimator
rules = [
    Rule.sagemaker(rule_configs.vanishing_gradient()),
    Rule.sagemaker(rule_configs.overfit()),
    Rule.sagemaker(rule_configs.overtraining()),
    Rule.sagemaker(rule_configs.poor_weight_initialization()),
    ProfilerRule.sagemaker(rule_configs.ProfilerReport()),
]
```

```
hook_config = DebuggerHookConfig(
    hook_parameters={"train.save_interval": "100", "eval.save_interval": "10"}
)
```

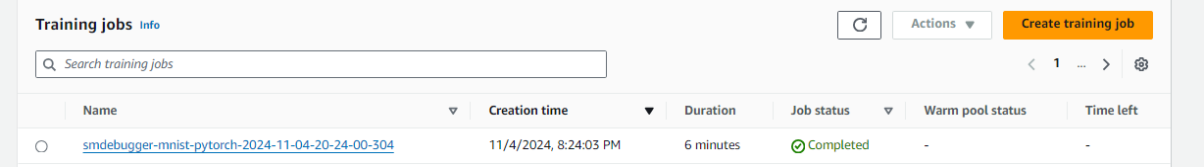
Next we will specify the hyperparameters and create our estimator. In our estimator, we will additionally need to specify the debugger rules and configs that we created before.

```
hyperparameters = {"epochs": "2", "batch-size": "32", "test-batch-size": "100", "lr": "0.001"}
```

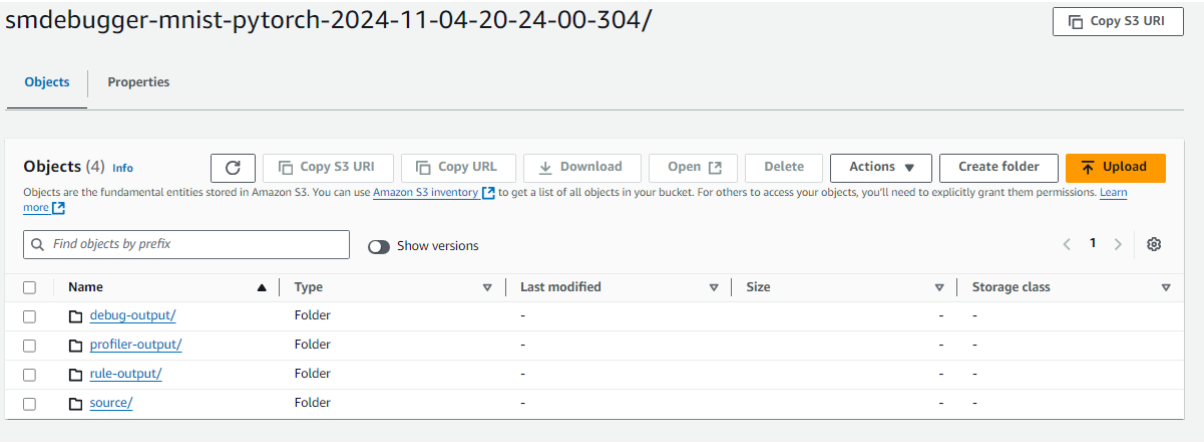
```
estimator = PyTorch(
    entry_point="pytorch_mnist.py",
    base_job_name="smdebugger-mnist-pytorch",
    role=get_execution_role(),
    instance_count=1,
    instance_type="ml.m5.large",
    hyperparameters=hyperparameters,
    framework_version="1.8",
    py_version="py36",
    ## Debugger parameters
    rules=rules,
    debugger_hook_config=hook_config,
)
```

```
estimator.fit(wait=True)
```

As shown in the screenshot below, debugger training was successfully completed.



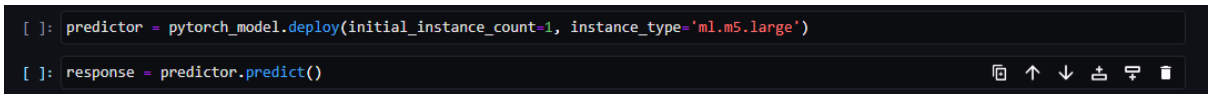
Below is a screenshot of profiler and debugger report that was successfully created,



Model Deployment

The first step is to verify that the model is performing as expected. For model evaluation and validation I used root mean square error (RMSE). RMSE was used to measure the difference between predicted values from the model and the actual observed values in the dataset. My model did not perform as I expected.

The final stage of our model is to deploy the model and query it to get results. This means that the model will be available for use in a production environment. The sagemaker was used to deploy the model using instance type ml.m5.large. The model was deployed as shown in screenshot below,



The endpoint has been deleted to avoid further costs.