Inventory Monitoring at Distribution Centers

1. Definition

1.1 Project Overview

A warehouse management system is a set of policies and processes intended to organize the work of a warehouse, and ensure that such a facility can operate efficiently and meet its objectives (Wikipedia 2024a). Today, warehouses are constantly growing in size and complexity and it's important to develop efficient processes that can accurately track the inventory. A major player in the field of warehouse management is Amazon. I'm interested in how a company, such as Amazon, has been able to build global warehouse management systems, and I want to look into how to streamline and improve processes in these systems. The project will use the Amazon Bin Image Dataset. This dataset is publicly available and contains 500,000 images of bins containing one or more items. The bin images are captured while the robots move shelf units around the warehouse.

1.2 Problem Statement

Industrial robots are used in warehouses to move items as part of operations. These items are carried in bins stacked on moveable shelf units. Each bin contains one or more items. The aim of this project is to build a machine learning model that can count the number of items in each bin. A system like this can be used for inventory tracking and reconciliation. As we are dealing with counting the number of items per bin, the problem is quantifiable and measurable.

The solution is to use a **supervised machine learning algorithm** to classify these images according to the number of items in each bin. In the model, each class corresponds to the number of items detected in the image. A pre-trained convolutional neural network can be used or a customized neural network can be developed. A suitable place to train and evaluate this model is by means of a jupyter notebook in Amazon SageMaker.

1.3 Metrics

A suitable evaluation metric in this project is the **test accuracy**. As the name suggests, the test accuracy is measured on the images reserved for testing only.

For a binary classification problem, accuracy is measured by means of following formula (Wikipedia 2024b):

$$Accuracy = (TP + TN) / (TP + TN + FP + FN)$$

where,

TP are true positives
TN are true negatives
FP are false positives
FN are false negatives

This method can be expanded to measure the accuracy of a multiclass classification problem. In our case, there are 5 classes, where each class corresponds to the number of items per bin. Also, there are other types of evaluation metrics that could be used in this project. For example, the loss, which provides an estimate of how well the trained model can be generalized to new data (Wikipedia 2024c).

2. Analysis

2.1 Data Exploration

The **Amazon Bin Image Dataset** has been used in this project. This dataset is publicly available and contains 500,000 images of bins containing one or more items (Amazon 2024). The bin images are captured while the robots move shelf units around a warehouse. This forms part of the normal operations in an Amazon Fulfillment Center. In the dataset, each bin image has a corresponding metadata file, containing information about the image, such as the number of items, its dimension and object type. Below are sample images from the dataset.

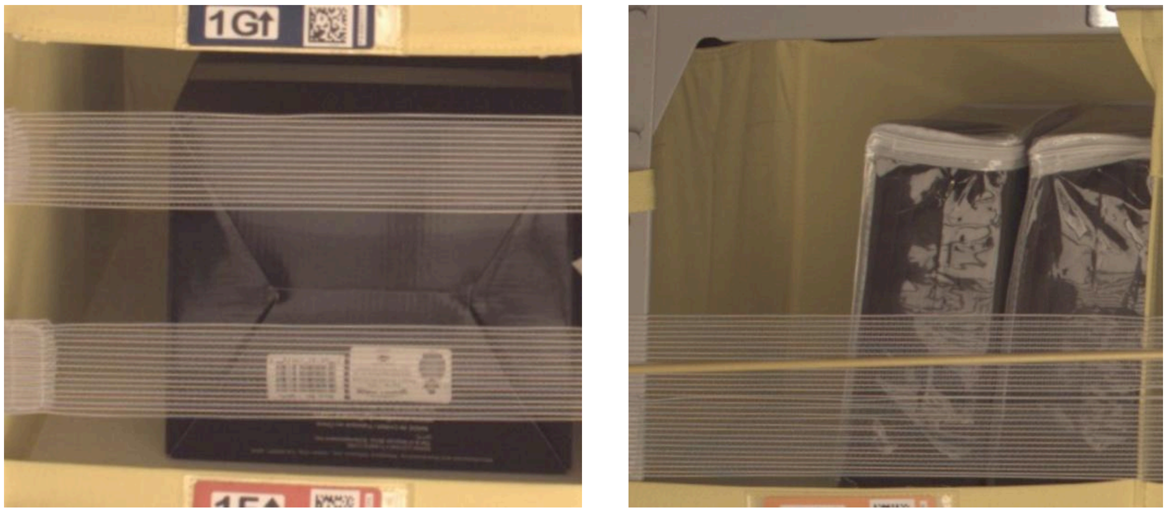


Figure 1: Sample images from the Amazon Bin Image Dataset

2.2 Exploratory Visualization

To reduce the cost of training, a subset of the full dataset has been used. This subset has 10,441 images, which represents about 2% of the total number of images. These images have been divided into 5 classes, where each class represents the number of items in the image. The distribution of images is as follows: 1228 (1 item), 2299 (2 items), 2666 (3 items), 2373 (4 items) and 1875 (5 items). Below is bar chart of the subset of images used in the project:

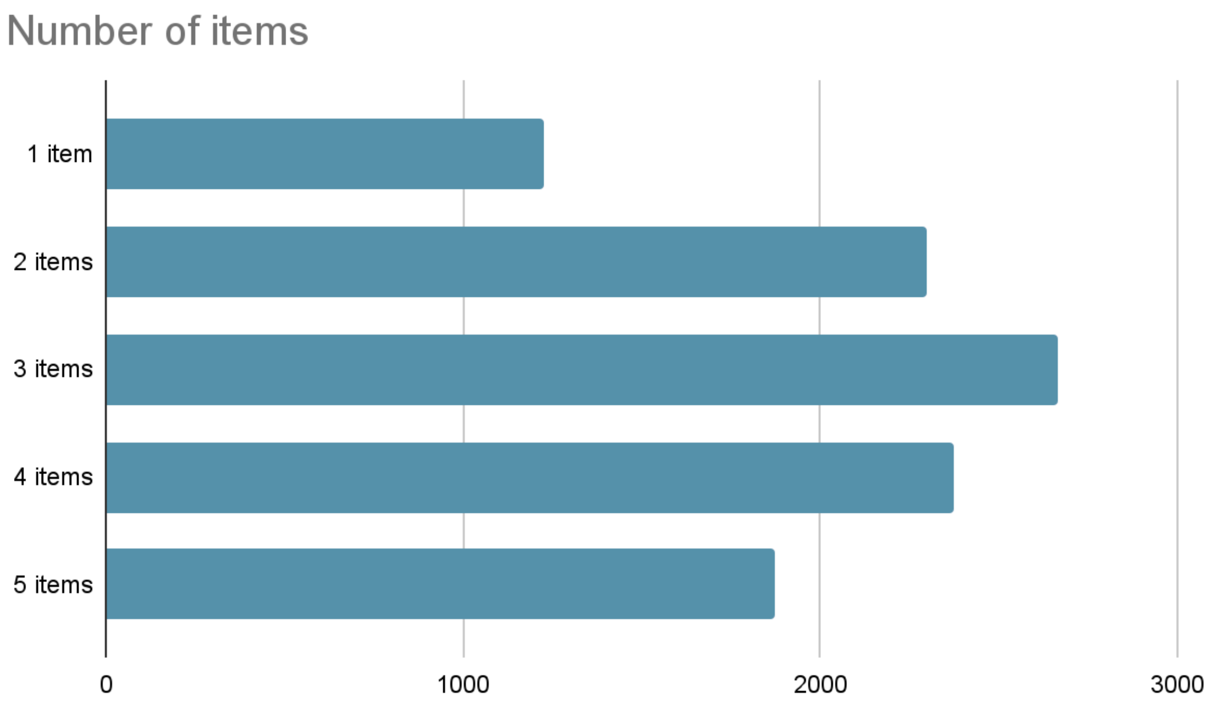


Figure 2: Bar chart showing distribution of images in the subset

The data has a slightly skewed distribution with a mean of about 3 items per image. According to the distribution, the model will be trained more heavily on bins with 3 items, followed by bins with 2 or 4 items and then 1 or 5 items. This can affect the resulting accuracy of the model. Also, the images have imperfections. In some of the images, ít can be difficult to detect the exact number of items with the naked eye, meaning the model might struggle as well.

2.3 Algorithms and Techniques

A suitable benchmark model is **ResNet** (Residual Network). It is a convolutional neural network model which contains so-called residual blocks. The model was developed in 2015 for image classification. It won that year's ImageNet Large Scale Visual Recognition Challenge (Wikipedia 2024d). ImageNet is a large visual database designed for use in visual object recognition software. It serves as a dataset benchmark and contains more than 14 million hand-annotated images and more than 20,000 classes (Wikipedia 2024e).

The model comes in 5 versions: ResNet18, ResNet34, ResNet50, ResNet101 and ResNet152 (where the number indicates the depth in terms of layers). As mentioned, the models use residual blocks. In a traditional neural network, each layer feeds into the next layer. In a neural network with residual blocks, each layer feeds into the next layer and directly into the layers 2-3 hops away (He et al. 2016).

In Pytorch, model builders are available for all versions. These model builders can be instantiated with or without pretrained weights (Pytorch 2024).

2.4 Benchmark

In the project, the ResNet models will be used for both benchmark and training/evaluation.

The benchmark are the results achieved for the Amazon Bin Image Dataset by a Stanford group in 2019. These results are mentioned on the Amazon Bin Image Dataset page provided by AWS. The Stanford team obtained the following results (Amazon 2024; Bertorello et al. 2019a):

Algorithm	Epochs	Test Accuracy
ResNet18 (SGD)	20	50.4
Resnet34 (SGD)	22	51.2
ResNet34 (SGDR)	36	53.8

Table 1: Benchmark results (Bertorello et al. 2019a)

The team used a dataset size of 150,000 images which represent about 30% of the total number of images. In the table, SGD stands for Stochastic Gradient Descent and SGDR stands for Stochastic Gradient Descent with Restarts. They are both optimization algorithms.

For the training and evaluation, transfer learning will be used to reduce time and cost of the training. In transfer learning, the weights in a pre-trained version of the model are frozen. This is followed by adding an output layer to network with the correct number of classes (in our case 5). Only the weights in this output layer will be trained. Two model versions have been evaluated in this project: ResNet50 and ResNet101.

3. Methodology

The project uses script mode in SageMaker. In script mode, training has been carried out as follows:

Submission script
This script specifies, amongst other things, the hyperparameters you want to use, how to create the estimator for the framework and information about the training instances.

Model training script
This script contains the model training code and should be able to read command line arguments.

The files used in this project are as follows:

1) **sagemaker.ipynb** - in our case, the *submission script* is a jupyter notebook rather than a standalone Python file to make things easier (it contains dataset preparation, estimator fitting, hyperparameter tuning, model profiling and debugging and deployment)

2) **train.py** - script implementing the training and testing in PyTorch; this includes model initialization, creation of data batches and hooks for model debugging

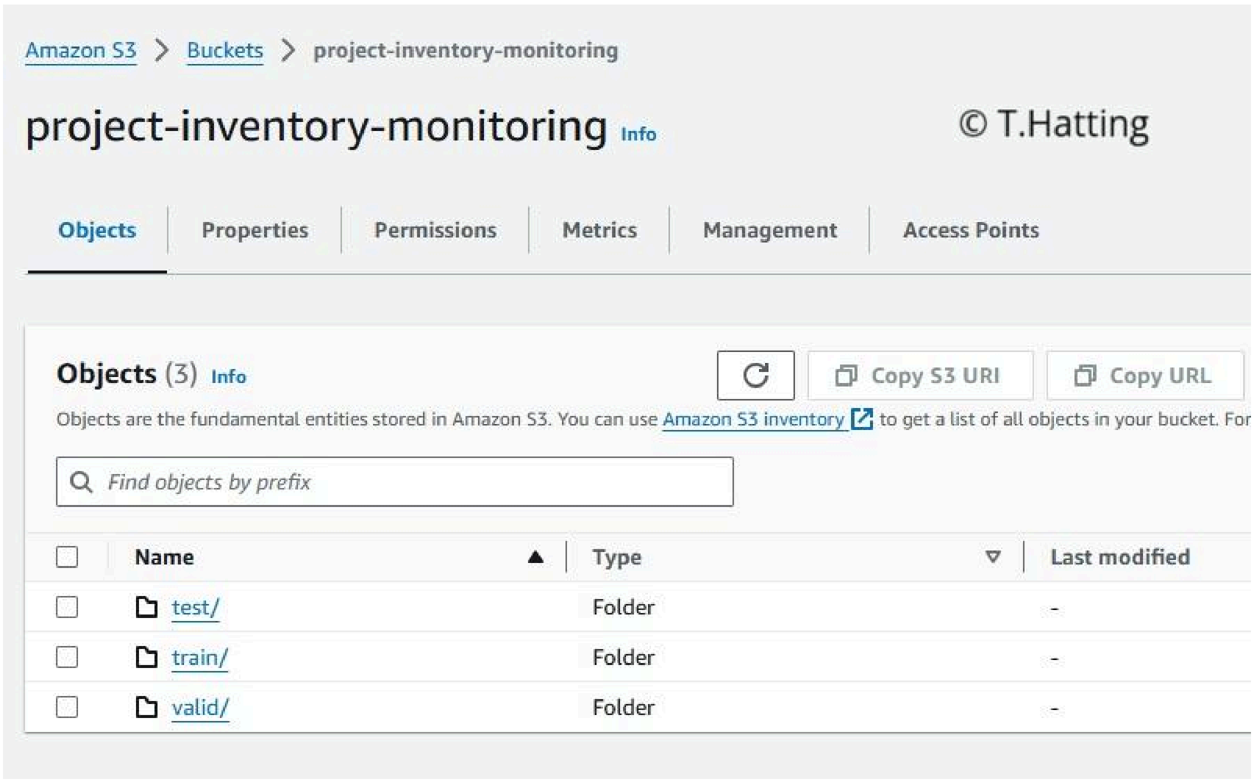
3) **hpo.py** - script similar to train.py and used for hyperparameter tuning

4) **inference.py** - script similar to train.py and used for inference

As can be seen above, in this project, the *model training script* is actually three separate Python files: training, hyperparameter tuning and inference.

3.1 Data Preprocessing

As mentioned earlier, to reduce the cost of training, a subset of the full dataset has been used. This subset contains 10,441 images, which represents about 2% of the total number of images. These images have been uploaded to Amazon S3 and divided into 3 categories: training, validation and testing. The training category contains 90% of images, the validation category contains 5% of images and the testing category contains 5% of images. The data splitting was done manually in Amazon S3.



Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For

Name	Type	Last modified
test/	Folder	-
train/	Folder	-
valid/	Folder	-

Figure 3: Splitting of data in Amazon S3

Below is a code snippet of preprocessing done in files train.py and hpo.py:

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

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

The images used for training are resized to 224 x 224 pixels, flipped horizontally for data augmentation and then converted to a tensor. Also, the images used for validation and testing are resized to 224 x 224 pixels and then converted to a tensor.

3.2 Implementation

The implementation steps below include not only the model training but also hyperparameter tuning, model profiling and debugging, deployment, use of spot instances and multi-instance training. Code snippets from the jupyter notebook (sagemaker.ipynb) have been provided showing how the estimator created in PyTorch has been modified to implement these characteristics.

3.2.1 Model Training

The creation of an estimator for model training is shown in the code snippet below (sagemaker.ipynb):

```
from sagemaker.pytorch import PyTorch

hyperparameters = {'batch_size': 128, 'learning_rate': 0.0005, 'epochs': 5}

estimator = PyTorch(
    entry_point='train.py',
    base_job_name='inventory-pytorch',
    role=role,
    instance_count=1,  # single-instance training
    instance_type='ml.g4dn.xlarge',
    framework_version='1.4.0',
    py_version='py3',
    hyperparameters=hyperparameters,
)
estimator.fit({"training": "s3://project-inventory-monitoring/"}, wait=True)
```

First, the PyTorch class is imported from SageMaker. Then, the 3 chosen hyperparameters are defined as a dictionary. The estimator object is then created as an instance of the PyTorch class using following arguments:

entry_point : script that carries out the training (in our case, *train.py*)

base_job_name : name used for the training job

role : sagemaker execution role in IAM (Identity and Access Management)

instance_count : EC2 instance count (in this case, only 1 instance is used)

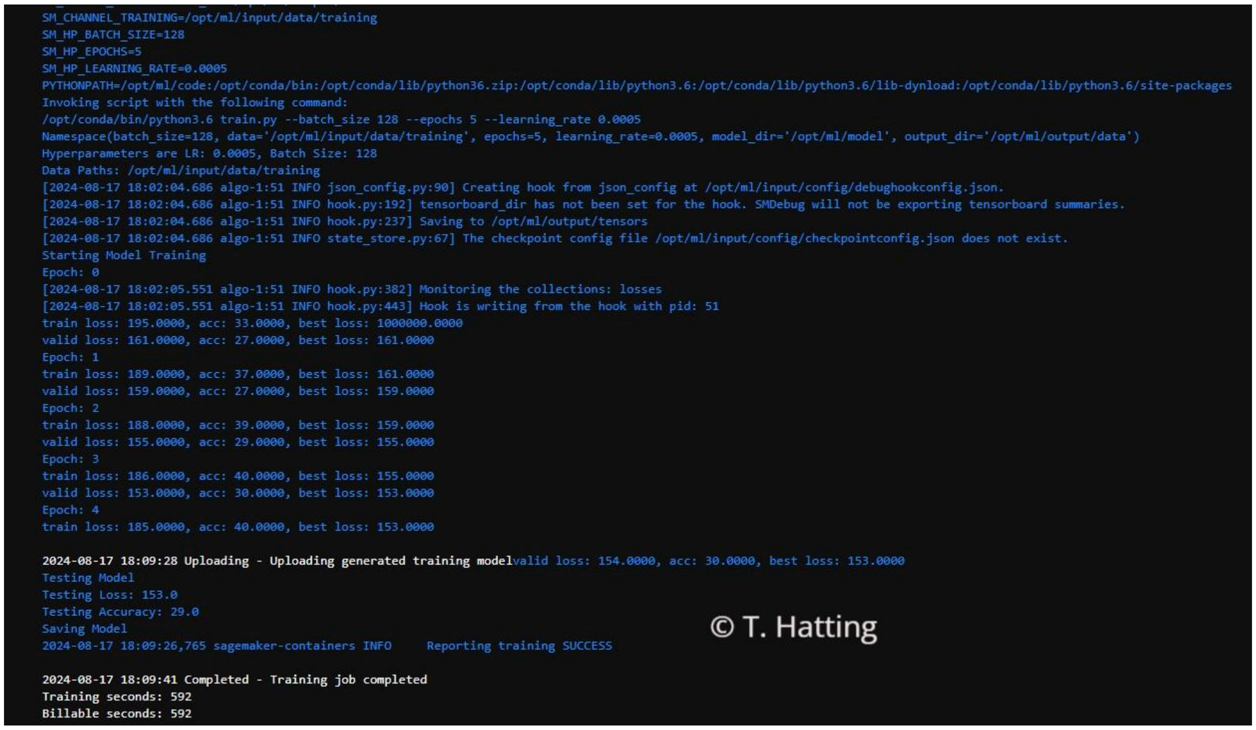
instance_type : EC2 instance type (in this case, the instance type is set to ml.g4dn.xlarge; this is a GPU-based instance as we want to offload the model and data batches to the GPU)

framework_version : PyTorch version for executing model training code

py_version : Python version for executing model training code

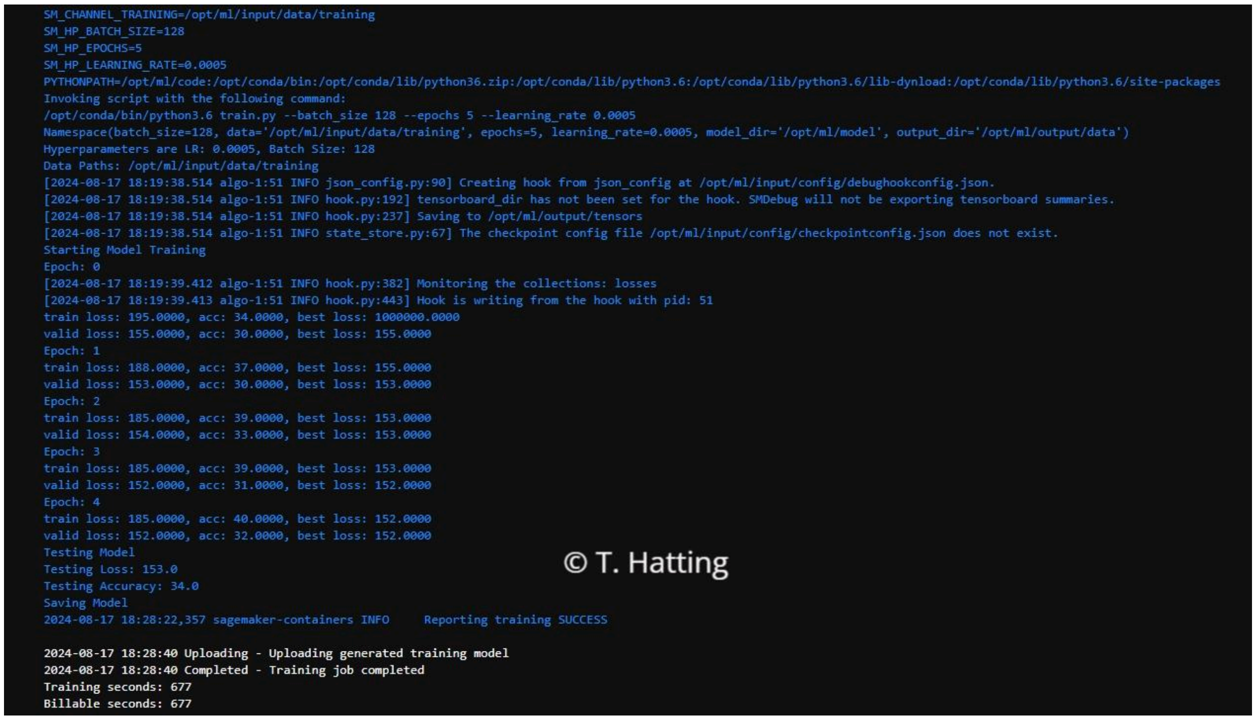
hyperparameters: hyperparameters used for training

Training sessions carried out with resnet50 and resnet101 have been shown in figures 4 and 5 below:



```
SM_CHANNEL_TRAINING=/opt/ml/input/data/training
SM_HP_BATCH_SIZE=128
SM_HP_EPOCHS=5
SM_HP_LEARNING_RATE=0.0005
PYTHONPATH=/opt/ml/code:/opt/conda/bin:/opt/conda/lib/python36.zip:/opt/conda/lib/python3.6:/opt/conda/lib/python3.6/lib-dynload:/opt/conda/lib/python3.6/site-packages
Invoking script with the following command:
/opt/conda/bin/python3.6 train.py --batch_size 128 --epochs 5 --learning_rate 0.0005
Namespace(batch_size=128, data='/opt/ml/input/data/training', epochs=5, learning_rate=0.0005, model_dir='/opt/ml/model', output_dir='/opt/ml/output/data')
Hyperparameters are LR: 0.0005, Batch Size: 128
Data Paths: /opt/ml/input/data/training
[2024-08-17 18:02:04.686 algo-1:51 INFO json_config.py:90] Creating hook from json_config at /opt/ml/input/config/debughookconfig.json.
[2024-08-17 18:02:04.686 algo-1:51 INFO hook.py:192] tensorboard_dir has not been set for the hook. SMDebug will not be exporting tensorboard summaries.
[2024-08-17 18:02:04.686 algo-1:51 INFO hook.py:237] Saving to /opt/ml/output/tensors
[2024-08-17 18:02:04.686 algo-1:51 INFO state_store.py:67] The checkpoint config file /opt/ml/input/config/checkpointconfig.json does not exist.
Starting Model Training
Epoch: 0
[2024-08-17 18:02:05.551 algo-1:51 INFO hook.py:382] Monitoring the collections: losses
[2024-08-17 18:02:05.551 algo-1:51 INFO hook.py:443] Hook is writing from the hook with pid: 51
train loss: 195.0000, acc: 33.0000, best loss: 1000000.0000
valid loss: 161.0000, acc: 27.0000, best loss: 161.0000
Epoch: 1
train loss: 189.0000, acc: 37.0000, best loss: 161.0000
valid loss: 159.0000, acc: 27.0000, best loss: 159.0000
Epoch: 2
train loss: 188.0000, acc: 39.0000, best loss: 159.0000
valid loss: 155.0000, acc: 29.0000, best loss: 155.0000
Epoch: 3
train loss: 186.0000, acc: 40.0000, best loss: 155.0000
valid loss: 153.0000, acc: 30.0000, best loss: 153.0000
Epoch: 4
train loss: 185.0000, acc: 40.0000, best loss: 153.0000

2024-08-17 18:09:28 Uploading - Uploading generated training modelvalid loss: 154.0000, acc: 30.0000, best loss: 153.0000
Testing Model
Testing Loss: 153.0
Testing Accuracy: 29.0
Saving Model
2024-08-17 18:09:26,765 sagemaker-containers INFO     Reporting training SUCCESS

2024-08-17 18:09:41 Completed - Training job completed
Training seconds: 592
Billable seconds: 592
```

Figure 4: Model training with resnet50



```
SM_CHANNEL_TRAINING=/opt/ml/input/data/training
SM_HP_BATCH_SIZE=128
SM_HP_EPOCHS=5
SM_HP_LEARNING_RATE=0.0005
PYTHONPATH=/opt/ml/code:/opt/conda/bin:/opt/conda/lib/python36.zip:/opt/conda/lib/python3.6:/opt/conda/lib/python3.6/lib-dynload:/opt/conda/lib/python3.6/site-packages
Invoking script with the following command:
/opt/conda/bin/python3.6 train.py --batch_size 128 --epochs 5 --learning_rate 0.0005
Namespace(batch_size=128, data='/opt/ml/input/data/training', epochs=5, learning_rate=0.0005, model_dir='/opt/ml/model', output_dir='/opt/ml/output/data')
Hyperparameters are LR: 0.0005, Batch Size: 128
Data Paths: /opt/ml/input/data/training
[2024-08-17 18:19:38.514 algo-1:51 INFO json_config.py:90] Creating hook from json_config at /opt/ml/input/config/debughookconfig.json.
[2024-08-17 18:19:38.514 algo-1:51 INFO hook.py:192] tensorboard_dir has not been set for the hook. SMDebug will not be exporting tensorboard summaries.
[2024-08-17 18:19:38.514 algo-1:51 INFO hook.py:237] Saving to /opt/ml/output/tensors
[2024-08-17 18:19:38.514 algo-1:51 INFO state_store.py:67] The checkpoint config file /opt/ml/input/config/checkpointconfig.json does not exist.
Starting Model Training
Epoch: 0
[2024-08-17 18:19:39.412 algo-1:51 INFO hook.py:382] Monitoring the collections: losses
[2024-08-17 18:19:39.413 algo-1:51 INFO hook.py:443] Hook is writing from the hook with pid: 51
train loss: 195.0000, acc: 34.0000, best loss: 1000000.0000
valid loss: 155.0000, acc: 30.0000, best loss: 155.0000
Epoch: 1
train loss: 188.0000, acc: 37.0000, best loss: 155.0000
valid loss: 153.0000, acc: 30.0000, best loss: 153.0000
Epoch: 2
train loss: 185.0000, acc: 39.0000, best loss: 153.0000
valid loss: 154.0000, acc: 33.0000, best loss: 153.0000
Epoch: 3
train loss: 185.0000, acc: 39.0000, best loss: 153.0000
valid loss: 152.0000, acc: 31.0000, best loss: 152.0000
Epoch: 4
train loss: 185.0000, acc: 40.0000, best loss: 152.0000
valid loss: 152.0000, acc: 32.0000, best loss: 152.0000
Testing Model
Testing Loss: 153.0
Testing Accuracy: 34.0
Saving Model
2024-08-17 18:28:22,357 sagemaker-containers INFO     Reporting training SUCCESS

2024-08-17 18:28:40 Uploading - Uploading generated training model
2024-08-17 18:28:40 Completed - Training job completed
Training seconds: 677
Billable seconds: 677
```

Figure 5: Model training with resnet101

The test accuracy of resnet101 (34%) is slightly higher compared to resnet50 (29%). This is likely due to the fact that resnet101 has a deeper architecture. In the sections below, the training will be carried out with resnet101. Furthermore, hyperparameter tuning will be used to achieve an even higher test accuracy.

3.2.2 Hyperparameter Tuning

Hyperparameter tuning allows adjusting model performance for optimal results. This process is a useful part of machine learning. In this project, following hyperparameters have been chosen for the tuning process:

Learning rate - hyperparameter that determines the step size in each iteration when moving toward a minimum for the loss

Batch size - number of images which are processed together before the model parameters are updated

Epoch - means one complete pass of the entire training dataset

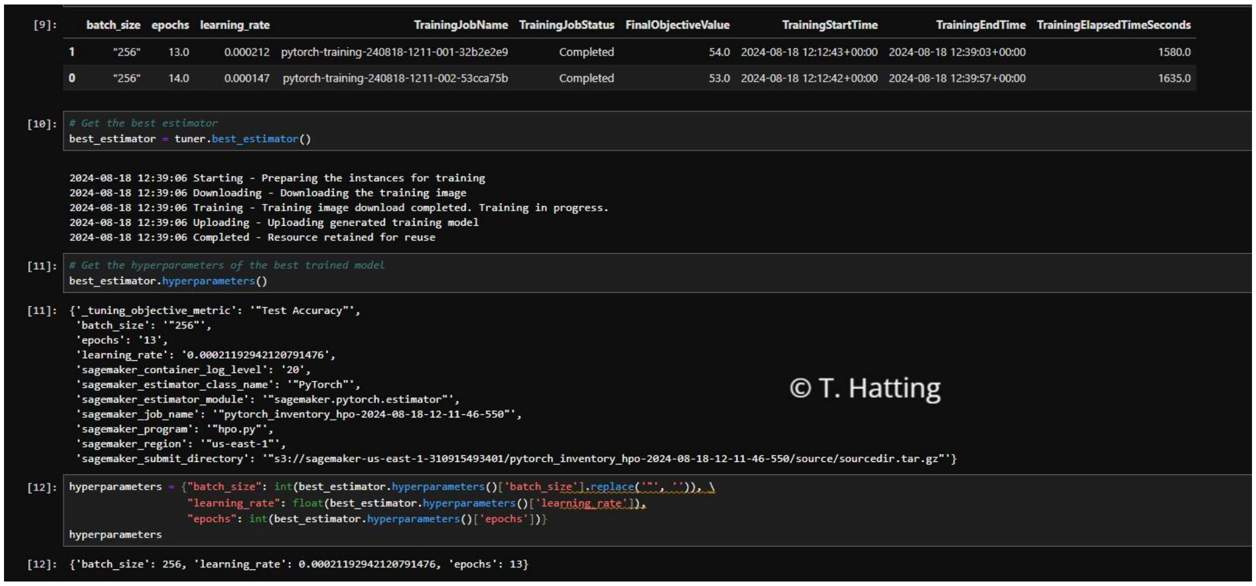
It is important to choose suitable value ranges for the hyperparameters in the tuning process.

The way to create an estimator and tuner object is shown in the code snippet below (sagemaker.ipynb):

```
hyperparameter_ranges = {
    "learning_rate": ContinuousParameter(0.00001, 0.01),
    "batch_size": CategoricalParameter([32, 64, 128, 256]),
    "epochs": IntegerParameter(10, 20)
}


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


estimator = PyTorch(
    entry_point="hpo.py",
    base_job_name='pytorch_inventory_hpo',
    role=role,
    framework_version="1.4.0",
    instance_count=1,
    instance_type="ml.g4dn.xlarge",
    py_version='py3'
)

# Create hyperparameter tuner

tuner = HyperparameterTuner(
    estimator,
    objective_metric_name,
    hyperparameter_ranges,
    metric_definitions,
    max_jobs=2,
    max_parallel_jobs=2,
    objective_type=objective_type
)

os.environ['SM_CHANNEL_TRAINING']='s3://project-inventory-monitoring/'
os.environ['SM_MODEL_DIR']='s3://project-inventory-monitoring/model/'
os.environ['SM_OUTPUT_DATA_DIR']='s3://project-inventory-monitoring/output/'
tuner.fit({"training": "s3://project-inventory-monitoring/"})
```

The parameter **hyperparameter_ranges** has been defined as a dictionary with objects representing the value ranges (continuous, categorical or an integer). Furthermore, the target metric during the tuning process has been defined by setting the parameter **objective_metric_name** to “Test Accuracy” and the parameter **objective_type** is to “Maximize”.

A **tuner** object is then created from the class **HyperparameterTuner** with the hyperparameter ranges and target metrics. Moreover, the tuner object is set to enable 2 tuning jobs to run concurrently. In figure 6, the results of the hyperparameter tuning is shown.



[9]:

	batch_size	epochs	learning_rate	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTime	TrainingElapsedTimeSeconds
1	"256"	13.0	0.000212	pytorch-training-240818-1211-001-32b2e2e9	Completed	54.0	2024-08-18 12:12:43+00:00	2024-08-18 12:39:03+00:00	1580.0
0	"256"	14.0	0.000147	pytorch-training-240818-1211-002-53cca75b	Completed	53.0	2024-08-18 12:12:42+00:00	2024-08-18 12:39:57+00:00	1635.0

```
[10]: # Get the best estimator
      best_estimator = tuner.best_estimator()

      2024-08-18 12:39:06 Starting - Preparing the instances for training
      2024-08-18 12:39:06 Downloading - Downloading the training image
      2024-08-18 12:39:06 Training - Training image download completed. Training in progress.
      2024-08-18 12:39:06 Uploading - Uploading generated training model
      2024-08-18 12:39:06 Completed - Resource retained for reuse

[11]: # Get the hyperparameters of the best trained model
      best_estimator.hyperparameters()

[11]: {'_tuning_objective_metric': '"Test Accuracy"',
       'batch_size': '"256"',
       'epochs': '13',
       'learning_rate': '0.00021192942120791476',
       'sagemaker_container_log_level': '20',
       'sagemaker_estimator_class_name': '"PyTorch"',
       'sagemaker_estimator_module': '"sagemaker.pytorch.estimator"',
       'sagemaker_job_name': '"pytorch_inventory_hpo-2024-08-18-12-11-46-550"',
       'sagemaker_program': '"hpo.py"',
       'sagemaker_region': '"us-east-1"',
       'sagemaker_submit_directory': '"s3://sagemaker-us-east-1-310915493401/pytorch_inventory_hpo-2024-08-18-12-11-46-550/source/sourcedir.tar.gz"'}

[12]: hyperparameters = {"batch_size": int(best_estimator.hyperparameters()['batch_size'].replace('"', '')), \
                         "learning_rate": float(best_estimator.hyperparameters()['learning_rate']),
                         "epochs": int(best_estimator.hyperparameters()['epochs'])}
      hyperparameters

[12]: {'batch_size': 256, 'learning_rate': 0.00021192942120791476, 'epochs': 13}
```

Figure 6: Optimal hyperparameter values obtaining from tuning with test accuracy as the target metric

3.2.3 Model Profiling and Debugging

In this project, model profiling and debugging have been included. The modifications to the estimator to allow for this is shown in the code snippet below (sagemaker.ipynb):

```
rules = [
    Rule.sagemaker(rule_configs.loss_not_decreasing()),
    ProfilerRule.sagemaker(rule_configs.LowGPUUtilization()),
    ProfilerRule.sagemaker(rule_configs.ProfilerReport()),
    Rule.sagemaker(rule_configs.vanishing_gradient()),
    Rule.sagemaker(rule_configs.overfit()),
    Rule.sagemaker(rule_configs.overtraining()),
    Rule.sagemaker(rule_configs.poor_weight_initialization()),
]

hook_config = DebuggerHookConfig(
    hook_parameters={
        "train.save_interval": "1",
        "eval.save_interval": "1"
    }
)

profiler_config = ProfilerConfig( system_monitor_interval_millis=500,
framework_profile_params=FrameworkProfile(num_steps=1)
)

estimator = PyTorch(
    entry_point='train.py',  # replaced hpo.py with train_model.py"
    base_job_name='inventory-pytorch',
    role=role,
    instance_count=instance_count,
    instance_type='ml.g4dn.xlarge',
    framework_version='1.4.0',
    py_version='py3',
    hyperparameters=hyperparameters,
    ## Debugger and Profiler parameters
    rules = rules,
    debugger_hook_config=hook_config,
    profiler_config=profiler_config,
    use_spot_instances=use_spot_instances,
    max_run = max_run,
    max_wait = max_wait,
)

estimator.fit({"training": "s3://project-inventory-monitoring/"}, wait=True)
```

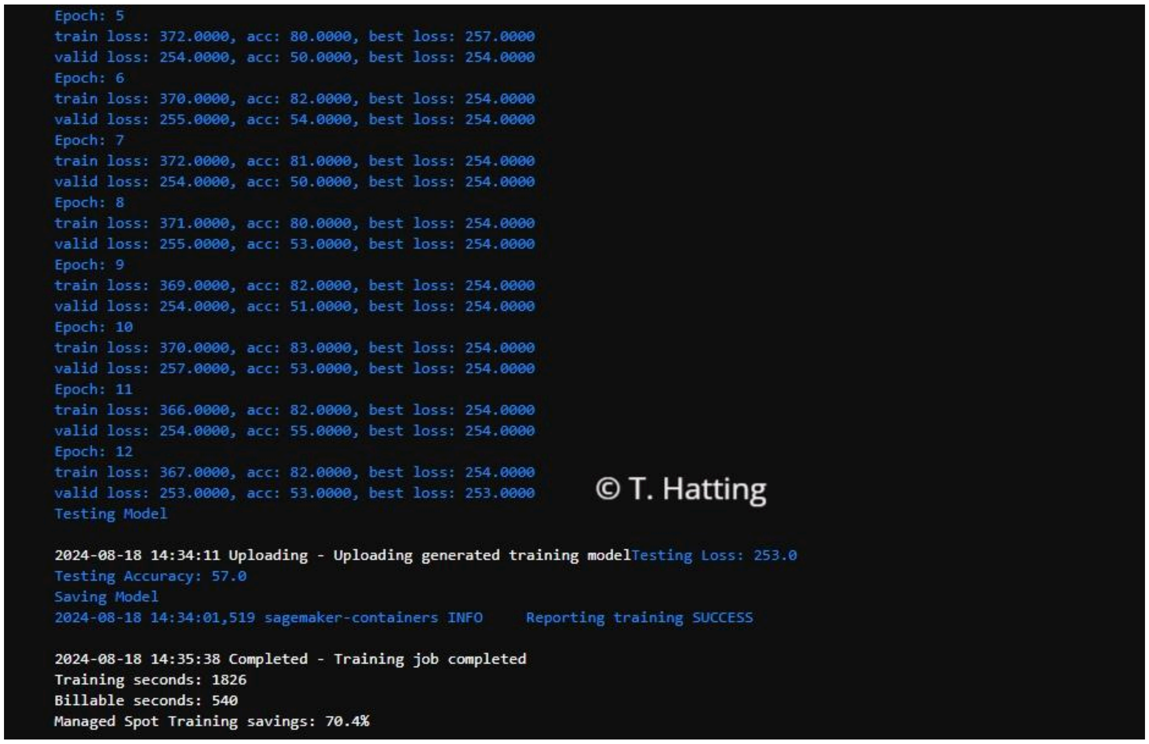
To configure profiling and debugging following variables should be defined:

rules : rules used in the debugging process such as overfitting, overtraining and poor weight initialization

hook_config : hook configuration object for debugging

profiler_config : configuration object for profiling

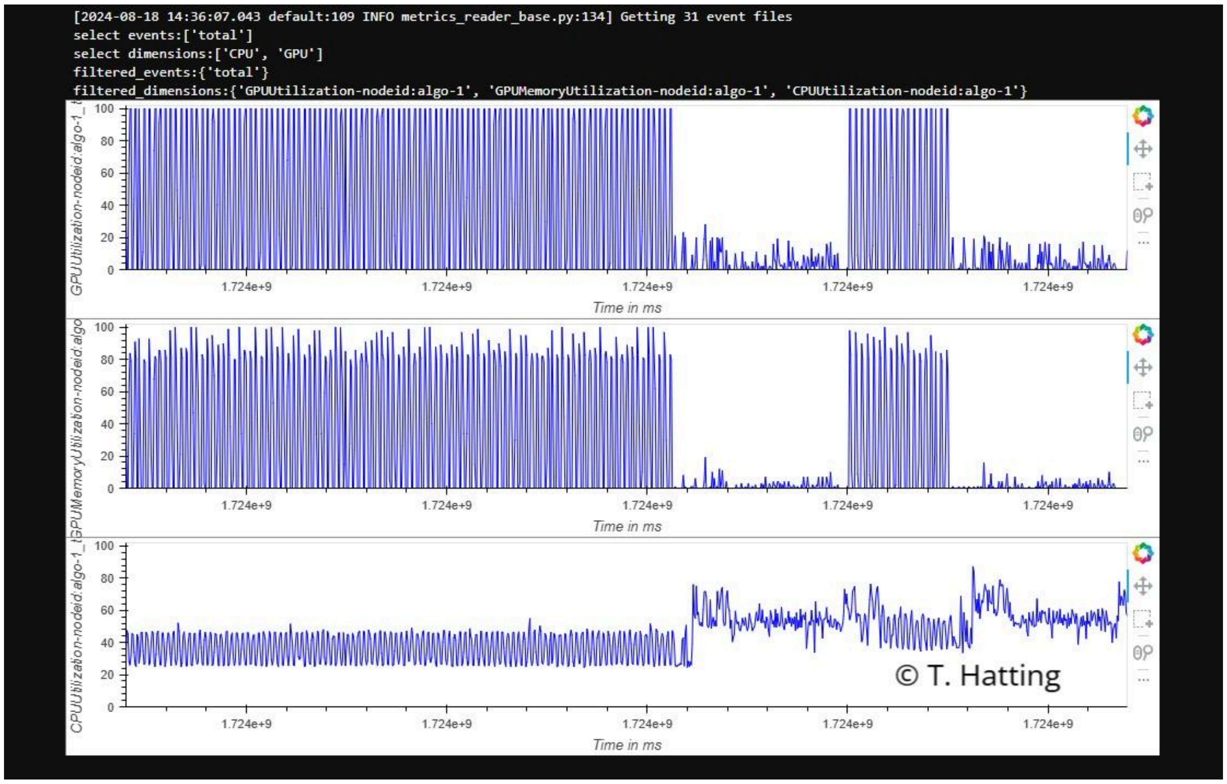
In figure 7, model training has been shown with the optimal hyperparameters obtained during tuning. As can be seen, the test accuracy is 57%, which is higher than the previous run. Also, there seems to be some overfitting, as the training and validation accuracy are quite far apart. The overfitting could be due to the fact that the dataset is relatively small. Also, note that spot instances have been used to reduce the training costs.



```
Epoch: 5
train loss: 372.0000, acc: 80.0000, best loss: 257.0000
valid loss: 254.0000, acc: 50.0000, best loss: 254.0000
Epoch: 6
train loss: 370.0000, acc: 82.0000, best loss: 254.0000
valid loss: 255.0000, acc: 54.0000, best loss: 254.0000
Epoch: 7
train loss: 372.0000, acc: 81.0000, best loss: 254.0000
valid loss: 254.0000, acc: 50.0000, best loss: 254.0000
Epoch: 8
train loss: 371.0000, acc: 80.0000, best loss: 254.0000
valid loss: 255.0000, acc: 53.0000, best loss: 254.0000
Epoch: 9
train loss: 369.0000, acc: 82.0000, best loss: 254.0000
valid loss: 254.0000, acc: 51.0000, best loss: 254.0000
Epoch: 10
train loss: 370.0000, acc: 83.0000, best loss: 254.0000
valid loss: 257.0000, acc: 53.0000, best loss: 254.0000
Epoch: 11
train loss: 366.0000, acc: 82.0000, best loss: 254.0000
valid loss: 254.0000, acc: 55.0000, best loss: 254.0000
Epoch: 12
train loss: 367.0000, acc: 82.0000, best loss: 254.0000
valid loss: 253.0000, acc: 53.0000, best loss: 253.0000
Testing Model

2024-08-18 14:34:11 Uploading - Uploading generated training modelTesting Loss: 253.0
Testing Accuracy: 57.0
Saving Model
2024-08-18 14:34:01,519 sagemaker-containers INFO     Reporting training SUCCESS

2024-08-18 14:35:38 Completed - Training job completed
Training seconds: 1826
Billable seconds: 540
Managed Spot Training savings: 70.4%
```

Figure 7: Model training with “best” hyperparameters, including profiling and debugging

In figure 8, there is a screenshot of CPU/GPU utilization obtained from the profiling. To speed up the training, the model and data batches have been offloaded to the GPU (CUDA) in the EC2 instance. This shows up clearly in the graph.



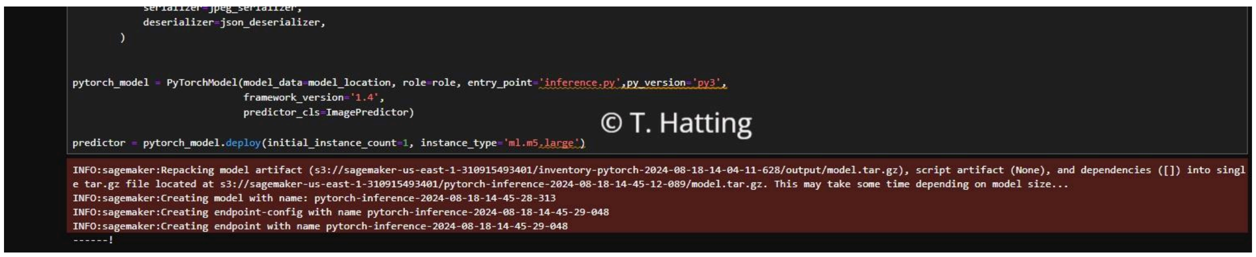
```
[2024-08-18 14:36:07.043 default:109 INFO metrics_reader_base.py:134] Getting 31 event files
select events:['total']
select dimensions:['CPU', 'GPU']
filtered_events:{'total'}
filtered_dimensions:{'GPUUtilization-nodeid:algo-1', 'GPUMemoryUtilization-nodeid:algo-1', 'CPUUtilization-nodeid:algo-1'}
```

GPUUtilization-nodeid:algo-1_t
GPUMemoryUtilization-nodeid:algo
CPUUtilization-nodeid:algo-1_t
Time in ms
1.724e+9
© T. Hatting

Figure 8: CPU/GPU utilization obtained from profiling

3.2.4 Model Deployment and Querying

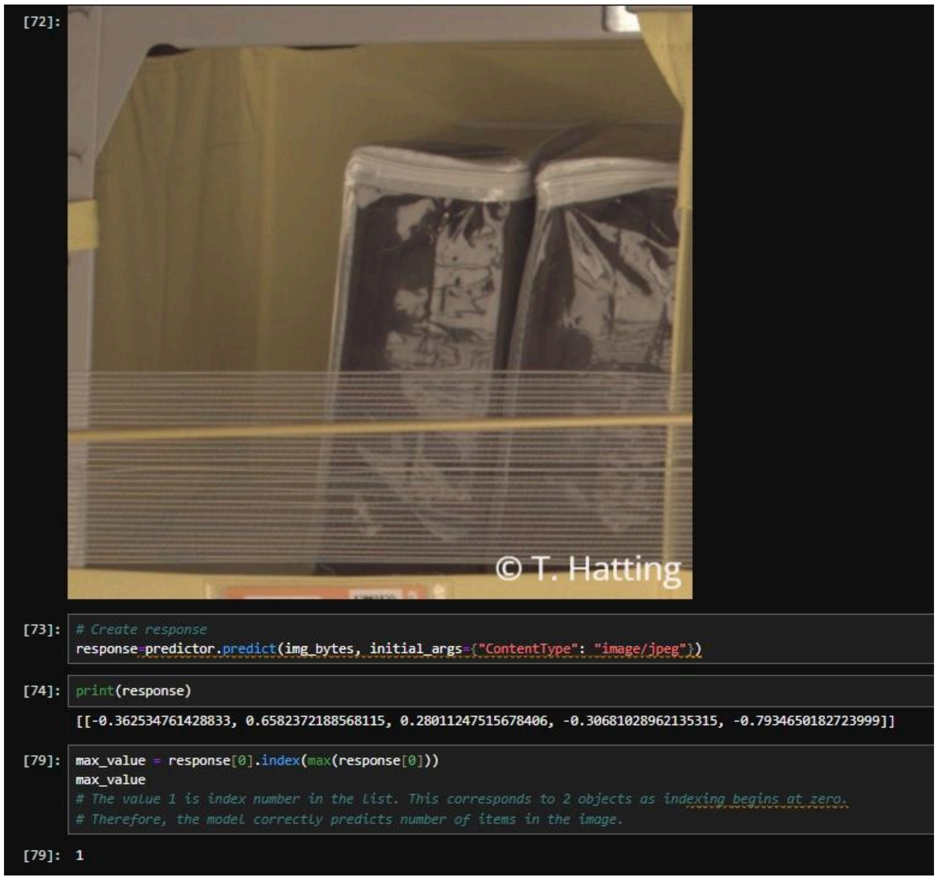
In figures 9 and 10 below, the endpoint has been created in SageMaker and inference has been carried out on a sample test image.



```
            serializer=jpeg_serializer,
            deserializer=json_deserializer,
        )


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

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

```
INFO:sagemaker:Repacking model artifact (s3://sagemaker-us-east-1-310915493401/inventory-pytorch-2024-08-18-14-04-11-628/output/model.tar.gz), script artifact (None), and dependencies ([]) into singl
e tar.gz file located at s3://sagemaker-us-east-1-310915493401/pytorch-inference-2024-08-18-14-45-12-089/model.tar.gz. This may take some time depending on model size...
INFO:sagemaker:Creating model with name: pytorch-inference-2024-08-18-14-45-28-313
INFO:sagemaker:Creating endpoint-config with name pytorch-inference-2024-08-18-14-45-29-048
INFO:sagemaker:Creating endpoint with name pytorch-inference-2024-08-18-14-45-29-048
------!
```

Figure 9: Creation of an active endpoint in SageMaker



```
[72]:
```

```
[73]: # Create response
      response=predictor.predict(img_bytes, initial_args={"ContentType": "image/jpeg"})

[74]: print(response)

      [[-0.362534761428833, 0.6582372188568115, 0.28011247515678406, -0.30681028962135315, -0.7934650182723999]]

[79]: max_value = response[0].index(max(response[0]))
      max_value
      # The value 1 is index number in the list. This corresponds to 2 objects as indexing begins at zero.
      # Therefore, the model correctly predicts number of items in the image.

[79]: 1
```

Figure 10: Inference using a sample test image

A *predictor* (endpoint) is created using the deploy() method on a Pytorch instance of the model. A test image (.jpg file) is then passed to the endpoint, in this case an image of a bin. A response is generated using *predictor.predict()*. The response returns a vector of probabilities. The index with the highest positive value indicates the correct label. In this case, the index is 1 (indexing starts at zero). In the bin image dataset stored in S3, index 1 corresponds to folder 2/ i.e.

bin images with two items. Therefore, the inference result is correct! This was achieved with a test accuracy of just 57% (which is relatively low).

3.2.5 Cheaper Training

The code snippet below shows changes to the estimator to allow for cheaper training by means of spot instances instead of on-demand EC2 instances (sagemaker.ipynb):

```
use_spot_instances = True
max_run=600
max_wait = 1200 if use_spot_instances else None

estimator = PyTorch(
    entry_point='train.py',  # replaced hpo.py with train_model.py"
    base_job_name='inventory-pytorch',
    role=role,
    instance_count=instance_count,
    instance_type='ml.g4dn.xlarge',
    framework_version='1.4.0',
    py_version='py3',
    hyperparameters=hyperparameters,
    ## Debugger and Profiler parameters
    rules = rules,
    debugger_hook_config=hook_config,
    profiler_config=profiler_config,
    ## Spot instances
    use_spot_instances=use_spot_instances,
    max_run = max_run,
    max_wait = max_wait,
)
```

```
estimator.fit({"training": "s3://project-inventory-monitoring/"}, wait=True)
```

Three parameters need to be set:

use_spot_instances : boolean value specifying whether to use SageMaker managed spot instances for training

max_run : maximum length of time, in seconds, that a training job can run

max_wait : maximum length of time, in seconds, that a managed spot training has to complete. It is the amount of time waiting for spot infrastructure to become available plus the amount of time the training job can run.

An AWS EC2 **spot instance** is an unused instance which is available for a much lower price than an on-demand instance. At the time or writing, the on-demand price of instance-type **g4dn.xlarge** was 0.526 (dollars) per hour whereas the spot price of the same instance-type was 0.1559 (dollars) per hour (note: this assumes that the instances are set up with Linux and not Windows). In conclusion, spot instances provide a significant saving.

3.2.6 Multi-Instance Training

To use multi-instance training is relatively easy in SageMaker compared to other systems. It is done by setting the parameter **instance_count** to the desired number instances when creating the estimator. SageMaker then figures out how to do the rest. See code snippet below (sagemaker.ipynb):

```
use_spot_instances = False
max_run=600
max_wait = 1200 if use_spot_instances else None
instance_count = 2

estimator = PyTorch(
    entry_point='train.py',  # replaced hpo.py with train_model.py"
    base_job_name='inventory-pytorch',
    role=role,
    instance_count=instance_count,
    instance_type='ml.g4dn.xlarge',
    framework_version='1.4.0',
    py_version='py3',
    hyperparameters=hyperparameters,
    ## Debugger and Profiler parameters
    rules = rules,
    debugger_hook_config=hook_config,
    profiler_config=profiler_config,
    use_spot_instances=use_spot_instances,
    max_run = max_run,
    max_wait = max_wait,
)

estimator.fit({"training": "s3://project-inventory-monitoring/"}, wait=True)
```

The main reason for using multiple instances is to speed up the training. However, in this project, the training is relatively fast when using just one EC2 instance. There seems to be no major benefit to using multiple instances. This is because the training dataset is relatively small (only 2% of the full dataset). Also, the training has been speeded up by offloading the model and data batches to the GPU. It is also possible to have multi-instance training with distributed data i.e. the data is sharded (split) between multiple instances. However, again as the dataset used for training is relatively small, it does not seem to be beneficial to use sharded data.

3.3 Refinement

Initially, a training session was carried out with suitable values for learning rate, batch size and epochs. The training was then improved by means of hyperparameter tuning as mentioned above. Different ranges for the hyperparameters were tried out until following ranges were chosen:

learning_rate: 0.00001 to 0.01 (continuous parameter)
batch_size: 32, 64, 128, 256 (categorical parameter)
epochs: 10 to 20 (integer parameter)

Learning rates outside the given ranges were either too low or high. Also, the next batch size up, 512, gave an out-of-memory error, so only the above 4 batch sizes were used. Also, the epoch range seemed suitable in terms of cost and duration.

4. Results

4.1 Model Evaluation and Validation

After hyperparameter tuning, the test accuracy reached 57%. This was achieved with batch_size=256, learning_rate=0.00021 and epochs=13. The corresponding training accuracy was 82% and validation accuracy was 53%. In other words, there seems to be a problem with overfitting. One way to overcome overfitting is through data augmentation. However, this requires a much larger dataset which would increase the cost of the project. Also, the images have imperfections. In some of the images, it can be difficult to detect the exact number of items with the naked eye, meaning the model might struggle as well.

Note: Each time hyperparameter tuning is carried out, slightly different hyperparameter values will be achieved with the goal of maximizing the test accuracy. For example, the tuning process might pick a lower batch number and match it with a slightly different learning rate. This means that the achieved test accuracy can vary somewhat from tuning session to tuning session. However, the above mentioned test accuracy represents a typical value that can be achieved by the model.

4.2 Justification

The results achieved compared well to the benchmark (see section 2.4 for more information about the benchmark). In the benchmark results, the Stanford team achieves a test accuracy of 53.8% using ResNet34 with epochs=36 and a dataset representing 30% of the total number of images (Bertorello et al. 2019a). In this project, a test accuracy of 57% was achieved using ResNet101 with epochs=13 and a dataset representing 2% of the total number of images. So, it appears beneficial to use a significantly deeper version of ResNet (101 layers deep rather than 34 layers deep). **However, having said all this, the test accuracy is still relatively low.** It is still only slightly above the 50% mark. Ideally, in machine learning projects, the test accuracy should be significantly higher (greater than 90%). The issue faced in the project is that only a small subset of the total dataset has been used in order to limit the cost of training. Also, the images are of varying quality as mentioned earlier. The issue could perhaps be overcome with further image augmentation in the data preprocessing stage.

References

Amazon (2024), “Amazon Bin Image Dataset”, Available online [html document]: https://registry.opendata.aws/amazon-bin-imagery/

Bertorello, P.R., Sripada, S. and Dendumrongsup, N. (2019a) “Amazon Inventory Reconciliation using AI”. Available online [html document]: https://github.com/pablo-tech/Image-Inventory-Reconciliation-with-SVM-and-CNN

Bertorello, P.R., Sripada, S. and Dendumrongsup, N. (2019b) “Amazon Inventory Reconciliation using AI”. Available online [pdf document]: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3311007

He, K., Zhang, X., Ren, S., Sun, J. (2016) “Deep Residual Learning for Image Recognition”. Available online [pdf document]: https://openaccess.thecvf.com/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf

PyTorch 2024, “ResNet: Model Builders”, Available online [html document]: https://pytorch.org/vision/main/models/resnet.html

Wikipedia 2024a, “Warehouse management systems”. Available online [html document]: https://en.wikipedia.org/wiki/Warehouse_management_system

Wikipedia 2024b, “Precision and recall”. Available online [html document]: https://en.wikipedia.org/wiki/Precision_and_recall

Wikipedia 2024c, “Loss functions for classification”. Available online [html document]: https://en.wikipedia.org/wiki/Loss_functions_for_classification

Wikipedia 2024d, “Residual Neural Network”. Available online [html document]: https://en.wikipedia.org/wiki/Residual_neural_network

Wikipedia 2024e, “ImageNet”. Available online [html document]: https://en.wikipedia.org/wiki/ImageNet