

AWS Machine Learning Engineer Nano degree Capstone Project – Inventory Monitoring at a Distribution Center

Folajimi Adekoya

March 22nd, 2023

1. Domain Background

Most e-commerce brands, wholesaler and retailers need to keep track of their inventory. This has not always been easy in the past but these days there are technologies, systems and processes that help companies to manage their supply chain. Inventory control systems are used by most companies to keep track of their inventory. These systems are important because they help to keep track of a company inventory levels in one warehouse or in warehouses spread across several locations. An important part of the inventory control system is inventory monitoring which is how we keep track of physical inventory. Machine learning provides a way to improve the accuracy of inventory monitoring through the use of image recognition and image classification. This project aims to implement an inventory monitoring system that uses Machine learning.

2. Problem statement

At Amazon distribution centres, robots are often used to move objects around as part of the operations. Objects are moved around in bins which can hold multiple objects. This project aims to build a model that can count the number of objects in each bin. When the model is integrated with the system, it will help to track inventory and make sure that delivery consignments have the correct number of items. The efficiency of this solution can be evaluated using metrics such as accuracy etc. This makes the solution measurable. We can measure the efficiency by measuring the quantity of the goods handled per day.

3. Metrics

As stated, this is a classification problem, classification problems are usually evaluated using accuracy as a metric.

The classification accuracy is defined as the number of correct predictions divided by the total number of predictions, multiplied by 100.

Classification accuracy = (No of correct predictions/ total predictions) * 100

4. Data Exploration

The amazon bin image dataset [1] was used to train the model. The dataset contains 535,234 images and corresponding metadata from bins of a pod in an operating Amazon Fulfillment Center. The images in the data set were captured while robots carried around pods during the operations in an amazon fulfilment centre. The bin sizes vary depending on the size of the objects in them. Tapes in front of the bins prevent the objects from falling out of the bins. The tapes are sometimes in the way and might make the objects in the images unclear. Some objects are obstructed by other objects, which results in a limited view of the images. A subset of the dataset will be used. It will be divided into the test, training and validation sets. The dataset will be saved in an S3 bucket from where it will be used.

Figure 1. sample Dataset images



Each image comes with metadata containing the number of objects in the image, the class labels for each object, and the object's dimensions.

Below you will find a sample of the metadata.

Figure 2. image metadata. From https://github.com/silverbottle/abid_challenge

```
{
  "BIN_FCSKU_DATA": {
    "B00CFQWRPS": {
      "asin": "B00CFQWRPS",
      "height": {
        "unit": "IN",
        "value": 2.399999997552
      },
      "length": {
        "unit": "IN",
        "value": 8.199999991636
      },
      "name": "Fleet Saline Enema, 7.8 Ounce (Pack of 3)",
      "normalizedName": "{Pack of 3} Fleet Saline Enema, 7.8 Ounce",
      "quantity": 1,
      "weight": {
        "unit": "pounds",
        "value": 1.8999999999999997
      },
      "width": {
        "unit": "IN",
        "value": 7.199999992656
      }
    },
    "ZZXI0WUSIB": {
      "asin": "B00T0BUKW8",
      "height": {
        "unit": "IN",
        "value": 3.99999999592
      },
      "length": {
        "unit": "IN",
        "value": 7.899999991942001
      },
      "name": "Kirkland Signature Premium Chunk Chicken Breast Packed in Water, 12.5 Ounce, 6 Count",
      "normalizedName": "Kirkland Signature Premium Chunk Chicken Breast Packed in Water, 12.5 Ounce, 6 Count",
      "quantity": 1,
      "weight": {
        "unit": "pounds",
        "value": 5.7
      },
      "width": {
        "unit": "IN",
        "value": 6.49999999337
      }
    },
    "ZZXVVS669V": {
      "asin": "B00C3WXJHY",
      "height": {
        "unit": "IN",
        "value": 4.330708657
      },
      "length": {
        "unit": "IN",
        "value": 11.1417322721
      },
      "name": "Play-Doh Sweet Shoppe Ice Cream Sundae Cart Playset",
      "normalizedName": "Play-Doh Sweet Shoppe Ice Cream Sundae Cart Playset",
      "quantity": 1,
      "weight": {
        "unit": "pounds",
        "value": 1.4109440759087915
      },
      "width": {
        "unit": "IN",
        "value": 9.448818888
      }
    }
  },
  "EXPECTED_QUANTITY": 3
}
```

It can be seen that the “EXPECTED_QUANTITY” attribute contains the value that represents the number of objects in a bin.

5. Exploratory Visualization

We see in Table 1 the image stats, such as the total number of images, the average quantity of Objects per bin and the total number of object categories in the Amazon Bin Dataset. A subset of the images is used to train due to Amazon web service's budget constraints.

Table 1. No of Bins and objects. From https://github.com/silverbottlep/abid_challenge

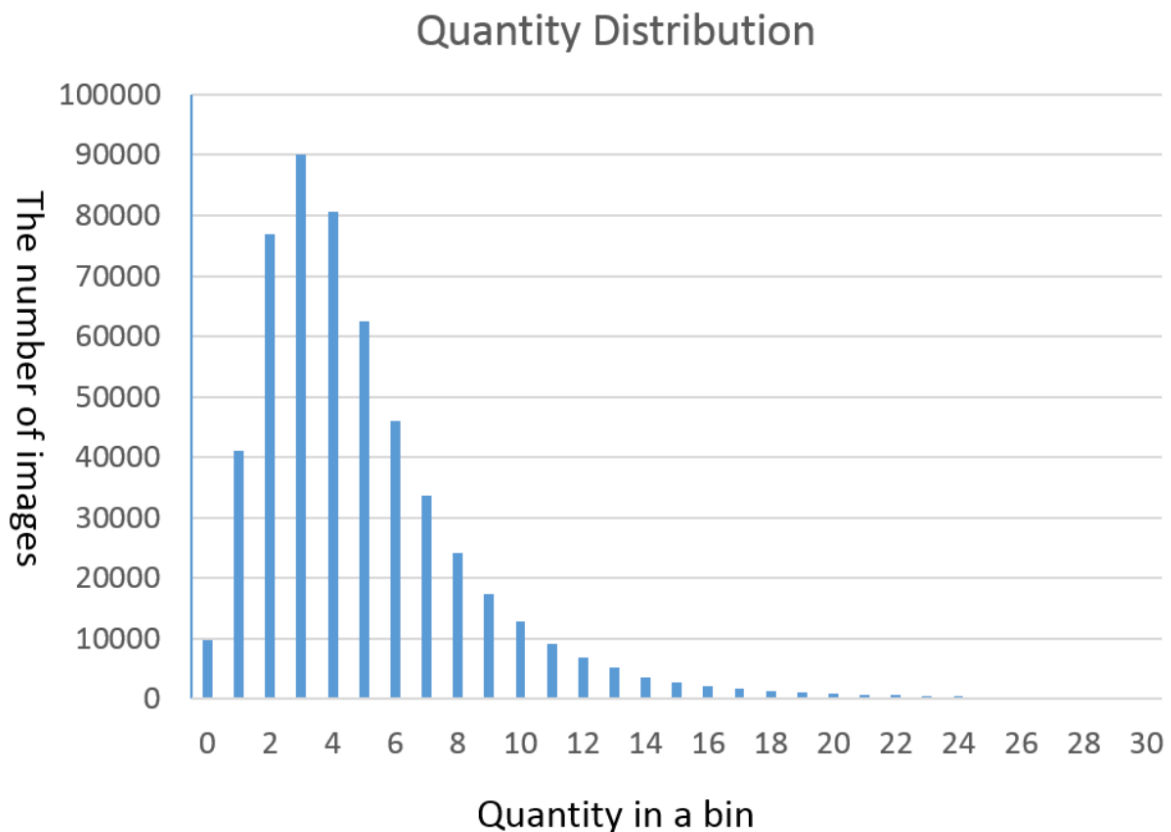
Description	Total
The number of images	535,234
Average quantity in a bin	5.1
The number of object categories	459,476

In figure 3, we see in the graph that approximately 90% of the bin images contain less than ten objects. We can also see that the class distribution is not balanced, which can be a problem for object counting in image classification.

This is because when the class distribution is not balanced, it can cause problems for object counting in image classification as it can bias the model's performance towards the majority class. In other words, the model may focus more on predicting the most frequent category. As a result, it may perform poorly in the minority classes.

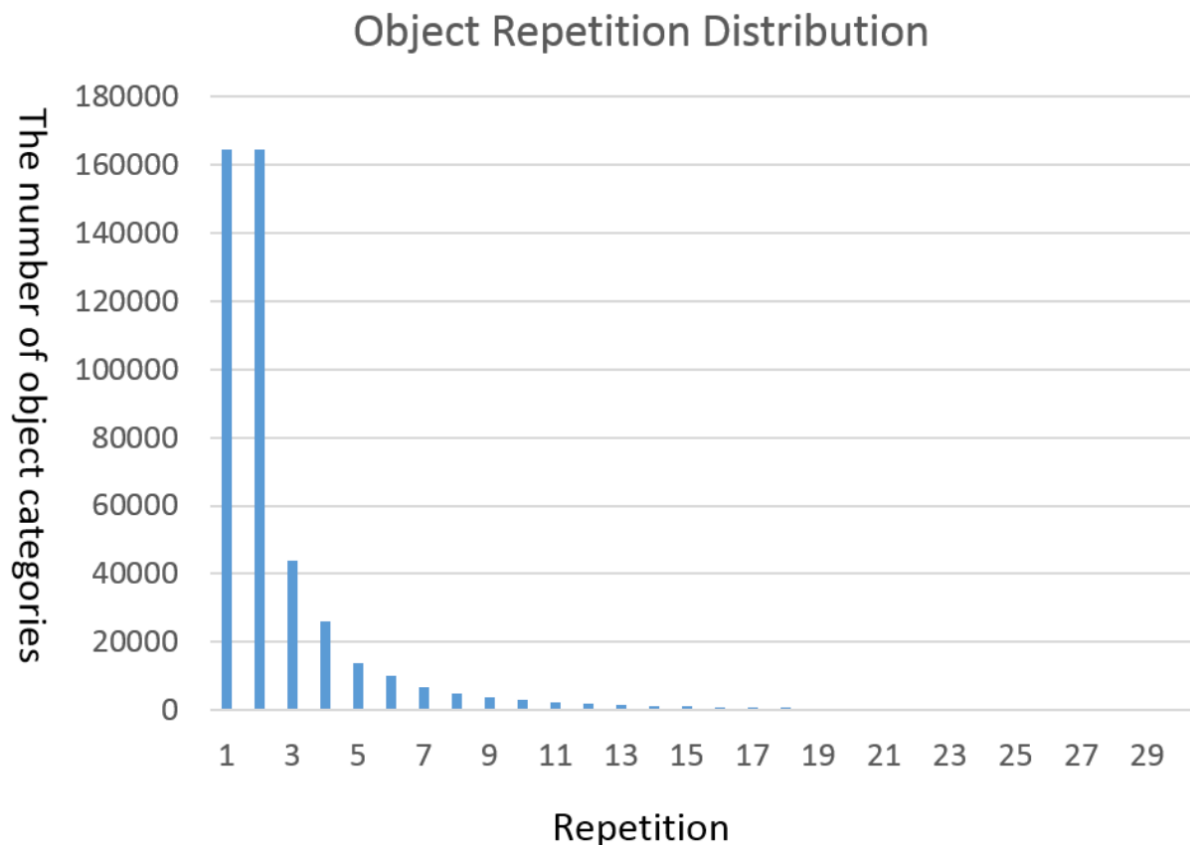
We must remember this when working on the algorithm and preprocessing the data.

Figure 3. Image class distribution



In figure 4, we see that for the distribution of object repetition in the dataset, 164255 object categories appear once in the dataset and 164356 object categories occur twice. This implies that about 71 % of the data contains unique objects.

Figure 4. Object Repetition distribution



6. Algorithms and Techniques

The ResNet50 model was fine-tuned via transfer learning using the Amazon Bin image Dataset to improve the accuracy of the object counting model worked on.

Transfer learning is employed since it is an efficient way of improving the performance of image classification models. The learned features can be adapted to a specific problem domain.

In this project, the model's performance was optimised using Amazon's hyperparameter tuning job called the Amazon SageMaker Hyperparameters Tuner.

Hyperparameters such as the batch size, learning rate (lr) and epochs were tuned to find the best combination to give the highest accuracy. The combination found was used for training my model.

To make the dataset available to SageMaker, a subset of the dataset was downloaded to an S3 bucket in three partitions, i.e. the test set, the training set and the validation set in a ratio of 6:2:2.

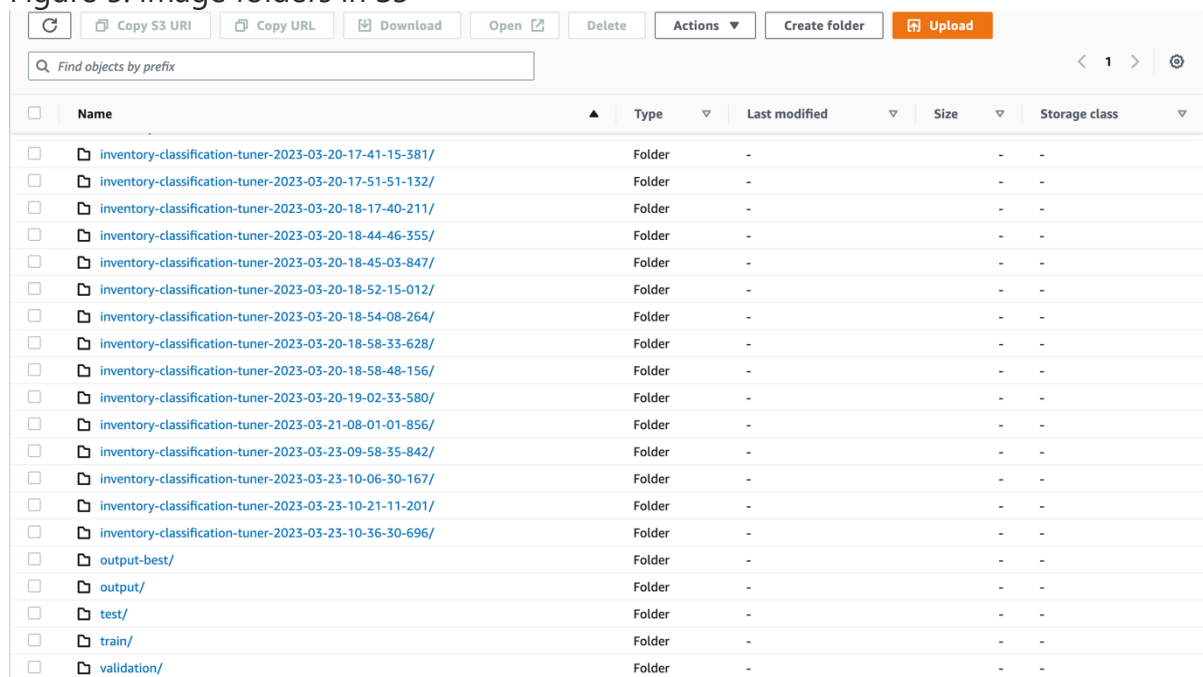
7. Benchmark

The benchmark model used is a deep convolutional classification network that uses the ResNet-50 architecture and is trained with the Amazon bin image dataset. The model was trained to classify images into six categories. This category represents the number of objects in each bin. The model was initially trained with ten epochs, a batch size of 128 and a learning rate of 0.001.

8. Data Preprocessing

As stated earlier, the dataset set is divided into the train, test and validation sets, where 60% of the data is for train, 20% for test and 20% for validation. The images are saved in folders with names train, test and validation in the S3 bucket.

Figure 5. image folders in S3



Name	Type	Last modified	Size	Storage class
inventory-classification-tuner-2023-03-20-17-41-15-381/	Folder	-	-	-
inventory-classification-tuner-2023-03-20-17-51-51-132/	Folder	-	-	-
inventory-classification-tuner-2023-03-20-18-17-40-211/	Folder	-	-	-
inventory-classification-tuner-2023-03-20-18-44-46-355/	Folder	-	-	-
inventory-classification-tuner-2023-03-20-18-45-03-847/	Folder	-	-	-
inventory-classification-tuner-2023-03-20-18-52-15-012/	Folder	-	-	-
inventory-classification-tuner-2023-03-20-18-54-08-264/	Folder	-	-	-
inventory-classification-tuner-2023-03-20-18-58-33-628/	Folder	-	-	-
inventory-classification-tuner-2023-03-20-18-58-48-156/	Folder	-	-	-
inventory-classification-tuner-2023-03-20-19-02-33-580/	Folder	-	-	-
inventory-classification-tuner-2023-03-21-08-01-01-856/	Folder	-	-	-
inventory-classification-tuner-2023-03-23-09-58-35-842/	Folder	-	-	-
inventory-classification-tuner-2023-03-23-10-06-30-167/	Folder	-	-	-
inventory-classification-tuner-2023-03-23-10-21-11-201/	Folder	-	-	-
inventory-classification-tuner-2023-03-23-10-36-30-696/	Folder	-	-	-
output-best/	Folder	-	-	-
output/	Folder	-	-	-
test/	Folder	-	-	-
train/	Folder	-	-	-
validation/	Folder	-	-	-

In Figure 3, it can be inferred from the class distribution plot that our dataset suffers from a severe class imbalance problem, which was addressed before the training stage by applying Pytorch's ImageFolder loader. This recognizes the subfolder name as a label automatically. This is to avoid any biases in my model.

As part of preprocessing the data before training, all the images were resized to (224, 224), and a horizontal flip with a probability of 0.5 was applied, hence preparing my data for training, ensuring accurate results and helps to reduce biasies and inaccuracies. The data preprocessing step is necessary to ensure the reliability and accuracy of the model.

9. Implementation

In the second part of the project, Amazon SageMaker's hyperparameter tuning functionality was used to optimize the model's hyperparameters. The tuned hyperparameters include Learning Rate (lr), batch size and epochs.

After tuning the values of the hyperparameters were

Lr= 0.09886747439639956

Batch size = 32

Epochs = 77

The hyperparameters were used for training the model, excluding the epoch. The training used ten epochs instead to save on costs. The pre-trained ResNet50 was with a fully connected layer that had its output mapped to six classes for the 0-5 object counting problem. The fully connected layer used can be seen in figure 6.

Figure 6. Fully connected layer

```
model = models.resnet50(pretrained=True)

for param in model.parameters():
    param.requires_grad = False

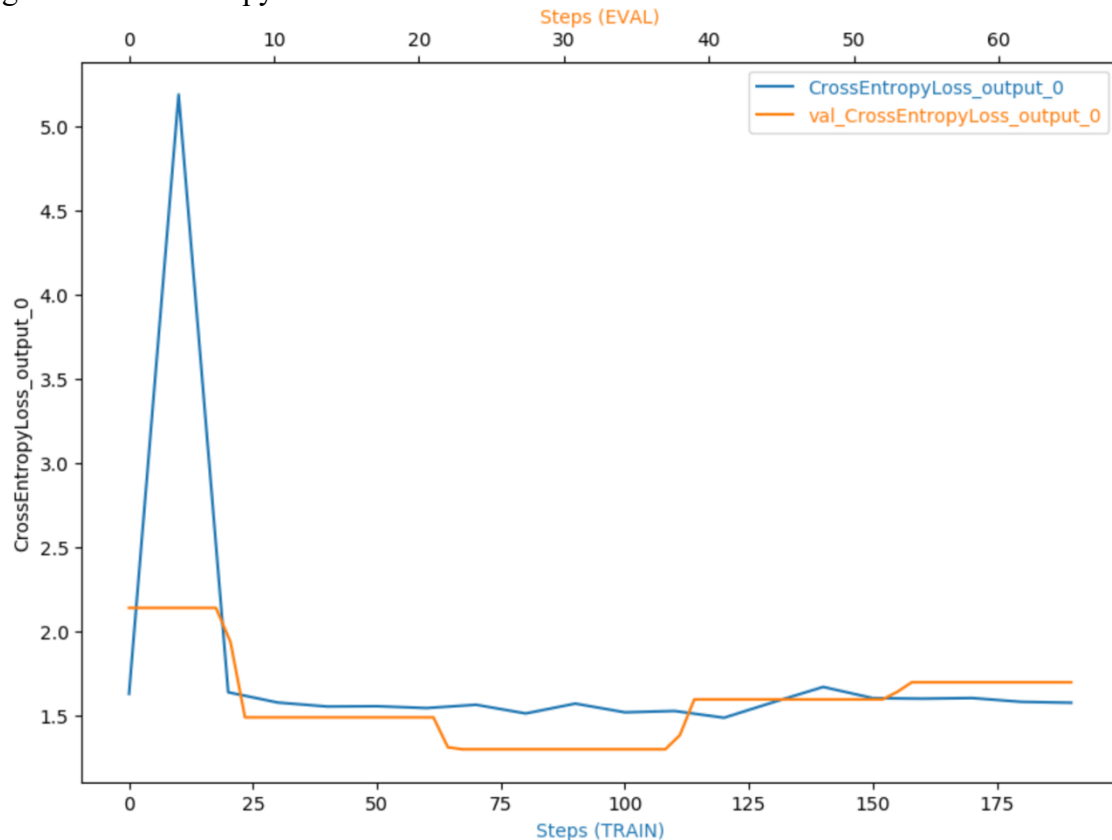
num_features=model.fc.in_features

model.fc = nn.Sequential(
    nn.Linear(num_features, 128),
    nn.ReLU(inplace=True),
    nn.Linear(128, 5))
```

A debugger and profiler were set up to detect overfitting, overtraining, and low GPU failures. During the training, none of these three had any issues.

The cross-entropy loss over the training and validation data can be seen in figure 7. The model converged, and it did not overfit or overtrain. This is because it was trained on a diverse subset of data.

Figure 7. CrossEntropyLoss



10. Refinement

A deep learning workflow using AWS SageMaker was used to train an image classification model on the Amazon bin image dataset. As is, the training model can be applied to a larger dataset to get a better model performance.

In my first attempt at training, I used the following hyperparameters.

Figure 8. Hyperparameters

```
hyperparameters = {
    'lr':0.001,
    'batch-size':128,
    'epochs':10,
    'test-batch-size':100
}
```

This resulted in the following average loss and Accuracy.

Test: Average loss: 1.5363, Accuracy: 571/2089 (27%)

After that, I applied hyperparameter tuning using the following ranges.

Figure 9. Hyperparameter ranges

```
hyperparameter_ranges = {
    'epochs': IntegerParameter(32, 128),
    'lr': ContinuousParameter(0.001, 0.1),
    'batch-size': CategoricalParameter([32, 64, 128, 256, 512]),
}
```

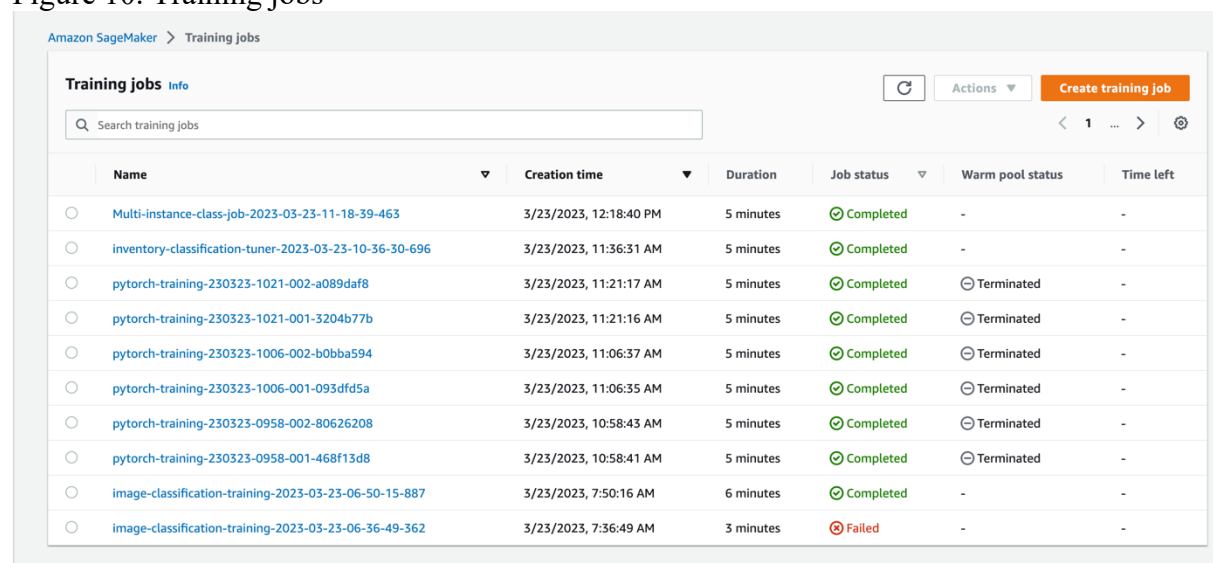

Which then found the following as the best-performing set of hyperparameters
 'batch-size': "32",
 'epochs': '77',
 'lr': '0.09886747439639956',

I used about an epoch of 10 instead of 32 when training to save cost, resulting in the following accuracy and average loss.

Train: Average Test loss: 7.5817, Accuracy: 1514/6262 (24%)

I could have gotten a better result by increasing the number of epochs to 32.

Figure 10. Training jobs



Amazon SageMaker > Training jobs

Training jobs Info

Search training jobs

	Name	Creation time	Duration	Job status	Warm pool status	Time left
<input type="radio"/>	Multi-instance-class-job-2023-03-23-11-18-39-463	3/23/2023, 12:18:40 PM	5 minutes	Completed	-	-
<input type="radio"/>	inventory-classification-tuner-2023-03-23-10-36-30-696	3/23/2023, 11:36:31 AM	5 minutes	Completed	-	-
<input type="radio"/>	pytorch-training-230323-1021-002-a089daf8	3/23/2023, 11:21:17 AM	5 minutes	Completed	Terminated	-
<input type="radio"/>	pytorch-training-230323-1021-001-3204b77b	3/23/2023, 11:21:16 AM	5 minutes	Completed	Terminated	-
<input type="radio"/>	pytorch-training-230323-1006-002-b0bba594	3/23/2023, 11:06:37 AM	5 minutes	Completed	Terminated	-
<input type="radio"/>	pytorch-training-230323-1006-001-093dfd5a	3/23/2023, 11:06:35 AM	5 minutes	Completed	Terminated	-
<input type="radio"/>	pytorch-training-230323-0958-002-80626208	3/23/2023, 10:58:43 AM	5 minutes	Completed	Terminated	-
<input type="radio"/>	pytorch-training-230323-0958-001-468f13d8	3/23/2023, 10:58:41 AM	5 minutes	Completed	Terminated	-
<input type="radio"/>	image-classification-training-2023-03-23-06-50-15-887	3/23/2023, 7:50:16 AM	6 minutes	Completed	-	-
<input type="radio"/>	image-classification-training-2023-03-23-06-36-49-362	3/23/2023, 7:36:49 AM	3 minutes	Failed	-	-

Figure 11. Training job using tuned hyper parameter

```
args to train_model.py : Namespace(batch_size=32, epochs=10, lr=0.09886747439639956, model_dir='/opt/ml/model', test_batch_size=100, test_data='/opt/ml/input/data/test', train_data='/opt/ml/input/data/train', val_data='/opt/ml/input/data/val')
Running on Device cuda:0
[2023-03-23 10:39:39.306 algo-1:45 INFO json_config.py:90] Creating hook from json_config at /opt/ml/input/config/debughookconfig.json.
[2023-03-23 10:39:39.307 algo-1:45 INFO hook.py:192] tensorboard_dir has not been set for the hook. SMDebug will not be exporting tensorboard summaries.
[2023-03-23 10:39:39.307 algo-1:45 INFO hook.py:237] Saving to /opt/ml/output/tensors
[2023-03-23 10:39:39.307 algo-1:45 INFO state_store.py:67] The checkpoint config file /opt/ml/input/config/checkpointconfig.json does not exist.
[2023-03-23 10:39:39.707 algo-1:45 INFO hook.py:382] Monitoring the collections: losses, relu_input, CrossEntropyLoss_output_0, gradients
[2023-03-23 10:39:39.707 algo-1:45 INFO hook.py:443] Hook is writing from the hook with pid: 45
Train [0/6262 (0%)]#011Loss: 1.630950
Train [320/6262 (5%)]#011Loss: 5.191901
Train [640/6262 (10%)]#011Loss: 1.639941
Train [960/6262 (15%)]#011Loss: 1.579055
Train [1280/6262 (20%)]#011Loss: 1.555296
Train [1600/6262 (26%)]#011Loss: 1.557095
Train [1920/6262 (31%)]#011Loss: 1.546237
Train [2240/6262 (36%)]#011Loss: 1.565822
Train [2560/6262 (41%)]#011Loss: 1.514079
Train [2880/6262 (46%)]#011Loss: 1.572720
Train [3200/6262 (51%)]#011Loss: 1.520813
Train [3520/6262 (56%)]#011Loss: 1.528937
Train [3840/6262 (61%)]#011Loss: 1.488661
Train [4160/6262 (66%)]#011Loss: 1.581635
Train [4480/6262 (71%)]#011Loss: 1.671689
Train [4800/6262 (77%)]#011Loss: 1.605119
Train [5120/6262 (82%)]#011Loss: 1.602055
Train [5440/6262 (87%)]#011Loss: 1.605774
Train [5760/6262 (92%)]#011Loss: 1.583258
Train [6080/6262 (97%)]#011Loss: 1.577867
Train : Average Test loss: 7.5817, Accuracy: 1514/6262 (24%)
Start Model Testing
2023-03-23 10:41:18 Uploading - Uploading generated training modelTest : Average Test loss: 1.5808, Accuracy: 533/2089 (26%)
Model Saving
2023-03-23 10:41:15,522 sagemaker-containers INFO Reporting training SUCCESS
2023-03-23 10:41:47 Completed - Training job completed
Training seconds: 243
Billable seconds: 243
```

11. Model Evaluation and Validation

The model deployed used the best-performing hyperparameters suggested by the hyperparameter tuner. The model was deployed to an endpoint, and I then queried the endpoint with the `predict()` method

to get a prediction.

This correctly predicted I had two images in the test image, as seen in Figure 12.

Figure 12. Prediction

```
In [53]: from PIL import Image
import io
Image.open(io.BytesIO(img_bytes))

Out[53]: 
```

```
In [54]: response = predictor.predict(img_bytes, initial_args={"ContentType": "image/jpeg"})

In [55]: response
Out[55]: [[-0.44875672459602356,
 0.20184704661369324,
 0.3914235234260559,
 0.09492100775241852,
 -0.006683466490358114]]

In [56]: import numpy as np
arg_max = np.argmax(response, 1)
arg_max
Out[56]: array([2])
```

Although my model accuracy was low, the prediction was correct. When I tested my endpoint with other images, it gave the correct prediction.

I plan to spend more time on the project later, so I can increase the epochs to get better performance, increase the dataset used in the model gets more varied training and also explore other pre-trained models known for great performance in image classification problems.

12. Justification

Comparing the accuracy of the proposed solution with the benchmarked accuracy, we can see that the benchmarked accuracy was 55.67%. In comparison, the proposed solution had an accuracy of 24%. There are many possible reasons for this discrepancy.

For example, the benchmark model was trained on all the data sets. In contrast, the proposed solution was trained on 10% of the dataset using a pre-trained model.

Since this project aims to demonstrate the use of Amazon SageMaker in solving the problem, it is safe to say the result satisfies what was aimed for.

13. Conclusion

In conclusion, it has been demonstrated that AWS SageMaker provides a robust platform for developing machine-learning models. Amazon makes the debugging and profiling libraries available for debugging and analyzing the model training process. These tools are beneficial in debugging issues encountered.

The model's performance can be significantly improved by increasing the training time and using a larger dataset and more complicated architectures with different layer combinations. Investing more time in experimenting can help figure out a better-performing model.

References

[1] - *Amazon Bin Image Dataset - Registry of Open Data on AWS*. (n.d.). [Amazon](https://registry.opendata.aws/amazon-bin-imagery/).

<https://registry.opendata.aws/amazon-bin-imagery/>

[2] - S. (n.d.). *GitHub - silverbottlep/abid_challenge: Amazon Bin Image Dataset Challenge*.

GitHub. https://github.com/silverbottlep/abid_challenge