Inventory Monitoring at Distribution Centers

AWS Machine Learning Engineer Nanodegree Program

Capstone Project

Matthias Luttmann
21.02.2022

1 Definition

1.1 Project Overview

The Amazon Fulfilment Centers are not built to store products only but also fulfil million orders from customers every day. Inventory Monitoring helps to keep the orders of customers correct. Therefore, robots are used in these distribution centers to automatically track products while carrying them in bins to lower costs and raise efficiency. During this process images of the bins were taken and the publicly available “Amazon Bin Image Dataset” with over 500.000 images were created [1].

1.2 Problem Statement

The process of collecting and packaging items by robots can be error prone. If an item is missing from a package, the customer will be dissatisfied. This will probably result in an additional package being sent with the missing item, causing additional costs. If more items are packed than ordered, profit will be lost also. Therefore, checking the number of items in a bin is a good idea. This can be solved with a training model for image classification.

In this project, Amazon AWS and PyTorch is used to train machine learning model. The strategy to solve this problem consists of the following tasks:

- Download and classify a subset of the “Amazon Bin Image Dataset”.
- Split the dataset into parts for training, validation and testing and upload them into an Amazon S3 bucket.

- Perform transfer learning on a pretrained convolutional neural network called ResNet50. Optimal hyperparameters are found by tuning job beforehand.
- Investigate the achieved result by profiling.
- Deploy an end point and predict an example image.

Different models have already been trained in the studies “Amazon Inventory Reconciliation Using AI” [2] and “Amazon Bin Image Dataset Challenge” [3]. One of these models can serve as benchmark for comparison in a later step.

1.3 Metrics

The data will be split into subsets for training, validation and testing. For evaluation while training the model, the average classification accuracy for the validation dataset is calculated and observed for multiple iterations. The training process stops as soon as the value stagnates. For the final evaluation, the accuracy is calculated for the testing dataset. Splitting up the data into three parts ensures that the final metric is not distorted because it is calculated on a data set that is not seen by the model before.

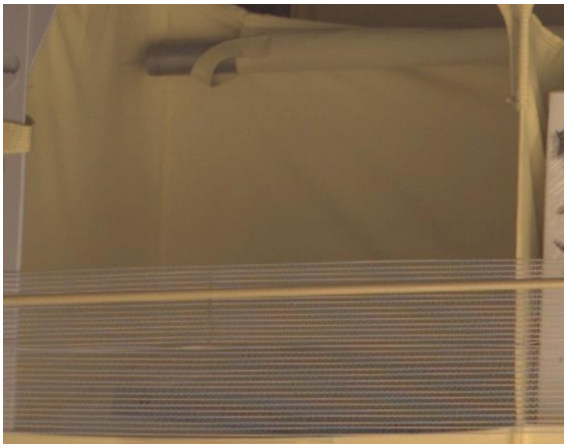
For this problem, precision would also be a good metric because finding the bins that are misclassified by the model is more important. But since the benchmark model is using accuracy as their evaluation metric, this project uses it too so the comparison between with the trained model is more meaningful.

2 Analysis

2.1 Data Exploration

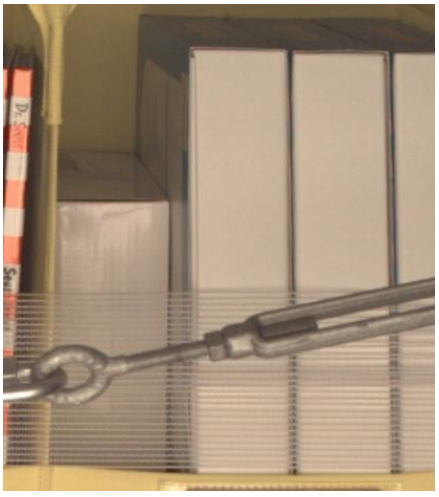
As a part of their normal operation, the robots take pictures of the products that they carry inside their bins. These images belong to the “Amazon Bin Image Dataset” and are store as JPEG files in an S3 container. Currently, this dataset consists of over more than 500,000 images. Each image has a JSON metafile that describes the content of the image, for example the number of elements in the image [1]. To reduce the cost of model training with AWS SageMaker, only a subset of 10,441 images is used. The selection of images can be investigated in a JSON file [4] provide by Udacity, Inc. These images contain one to five items.

For some images in the dataset, prediction can be very hard, like for the collection of bins in Figure 1. Items can be covered by tape (see Figure 1a) or chains (see Figure 1c). It can also happen that image appear blurry like in Figure 1b. Sometimes the image shows an item of a neighbor bin too (see Figure 1c). Also, it is important to note that items can be misplaced so the contents of the data may not match the recorded inventory [5] which can be seen in Figure 1d. In this picture the missing items could also be covered by the visible ones.



a) Items covered by a tape (05243.jpg)

b) Image is blurred (102984.jpg)



c) Item from neighbor bin visible (105077.jpg)

d) Bin classified with 4 items (00462.jpg)

Figure 1: Example images

Without manually searching for those images to remove or crop them, lower accuracy can be expected.

2.2 Exploratory Visualization

Figure 2 shows the distribution of the quantity of items per image including the split into training, validation and testing datasets. Out of subset of 10,441 images the most were

classified with 3 items. The images containing only one item are at least contained in the dataset.

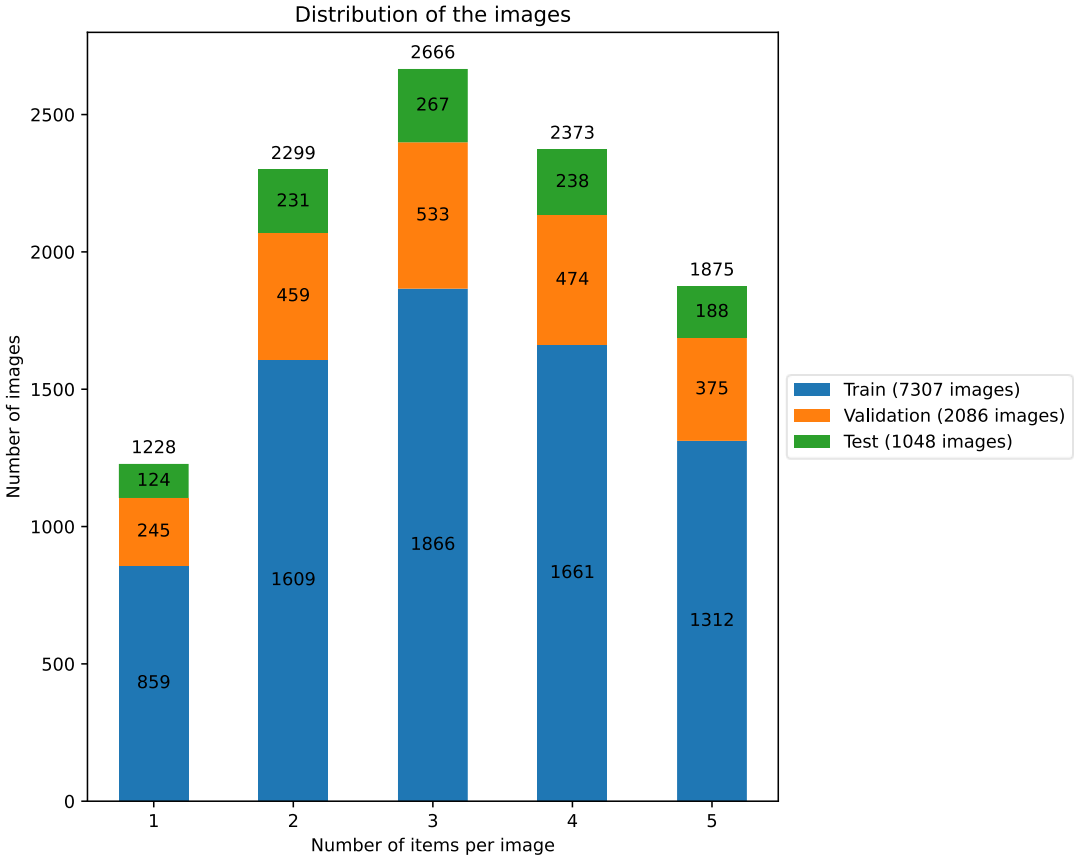


Figure 2: Distribution of the data subset

The statics analysis done in [6] shows a quantity distribution of the whole dataset. 90% of the images contains less than 10 objects. It can also be seen that the distribution shows a positive skew.

2.3 Algorithms and Techniques

To determine how well the model is performing by calculating the error, a cost function is used. Very common for multi classification problems is the cost function cross-entropy loss which is a combination of the cost function negative log-likelihood loss and the logarithmic softmax function. The calculation of the loss is done by the PyTorch. The actual algorithms of this loss function can be found in [7].

With an optimizer which updates the weights of our model, it can be specified how the model should learn. A good optimizer which works well out of the box for multi classification and give decent perform quickly is the Adam optimizer. This optimizer is also used for the

benchmark model described in the next chapter. The implementation for the optimizer is provided by PyTorch and the algorithms are defined in [8].

The chosen evaluation metric is accuracy which is calculated as follows:

$$accuracy = \frac{number\ of\ correct\ predicted\ images}{number\ of\ images}$$

2.4 Benchmark

As a benchmark, the results from the publication “Amazon Inventory Reconciliation Using AI” [2] will be used which is also mentioned by the “Amazon Bin Image Dataset” [1]. For the classification problem, the authors used support vector machines as well as convolutional neural networks which are of greater interest for this project.

The study used different models for training. One of the best models was a ResNet50 model with an Adam optimizer. The model was trained with over 320.000 images and achieved an accuracy of 61.2 % on the training dataset and an accuracy of 55.2 % on the test dataset.

3 Methodology

3.1 Data Preprocessing

To improve the training process, the data will be split roughly into 70 % for training, 20 % for validation and 10 % for testing. After downloading, a file is saved in a directory which represents its subset and furthermore its classification. The structure for the dataset is displayed in Figure 3.

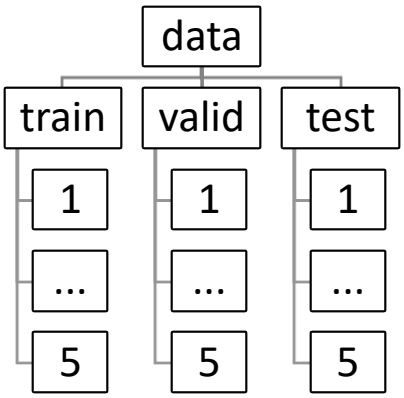


Figure 3: Data structure

All pretrained models from PyTorch expect the input data in a specific way [9]. An image must

- have a height (H) and width (W) of at least 224 pixels.
- be a 3-channel RGB image in form 3xHxW.
- be loaded as a tensor.
- be normalized with a mean of [0.485, 0.456, 0.406] and standard deviation of [0.229, 0.224, 0.225].

These transformations are made during the training process by the PyTorch data loaders. To add a bit of diversity while iterating over the datasets, a random portion of an image is cropped before resizing. Moreover, an image may also be flipped horizontally with a probability of 50 %.

3.2 Implementation

In the following, the steps to prepare the AWS instances, find the best hyperparameters, train a model and deploy an end point are described.

3.2.1 Preparing AWS Instances

First, an AWS instance for a Jupyter notebook needs to be created. It is sufficient to choose “ml.t2.medium” instance type since the model training is done by training jobs on other instances. Then, the kernel “conda_pytorch_latest_p36” must be chosen since it brings the required modules like the SageMaker SDK, PyTorch or torchvision. Alternatively, the notebook can be executed on a local machine on which the mentioned modules are installed.

In order to provide images for the training jobs, an empty S3 bucket must be created where the images can be uploaded to.

Additionally, to provide access to the S3 bucket, an appropriate policy must be attached to the execution role that is used by the SageMaker instance.

By executing the first cells of the notebook, the images are downloaded into their specific directory and then uploaded to defined S3 bucket.

3.2.2 Hyperparameter Tuning

A hyperparameter tuning job is created to find optimal parameters for model training. The entry point for the training was the “hpo.py” Python script. To get optimal results, the tuning

took place on instances of type “ml.g4dn.xlarge”. The range of hyperparameters are chosen by experience and displayed in Table 1.

Table 1: Hyperparameter range

Parameter	Type	Values
Learning rate	Continuous	0.0001 to 0.01
Batch size	Categorical	16, 64, 256, 512
Number of epochs	Integer	5 to 15

Table 2 shows four training jobs that are started by the tuning job. The highest accuracy is achieved by the second job with a batch size of 256, number of epochs of 5 and a learning rate of 0.000111.

Table 2: Training jobs for hyperparameter tuning

	batch-size	epochs	lr	TrainingJobName	TrainingJobStatus	FinalObjectiveValue
2	"256"	5.0	0.000111	inventory-hpo-tuning-220219-1737-002-08a7d6c5	Completed	29.389313
3	"64"	10.0	0.000319	inventory-hpo-tuning-220219-1737-001-4390c2a0	Completed	27.576336
0	"256"	6.0	0.000425	inventory-hpo-tuning-220219-1737-004-88c868bb	Completed	27.099237
1	"256"	12.0	0.000559	inventory-hpo-tuning-220219-1737-003-61d48e06	Completed	27.003817

3.2.3 Model Training

With the best hyperparameters found during tuning, a new model is trained with the “train.py” Python script. As with tuning, an instance of type “ml.g4dn.xlarge” was chosen. This script includes code for debugging to investigate bugs in the SageMaker Console as well as code for profiling to provide some visual output and performance metrics and to get notified if the model is under- or overfitting.

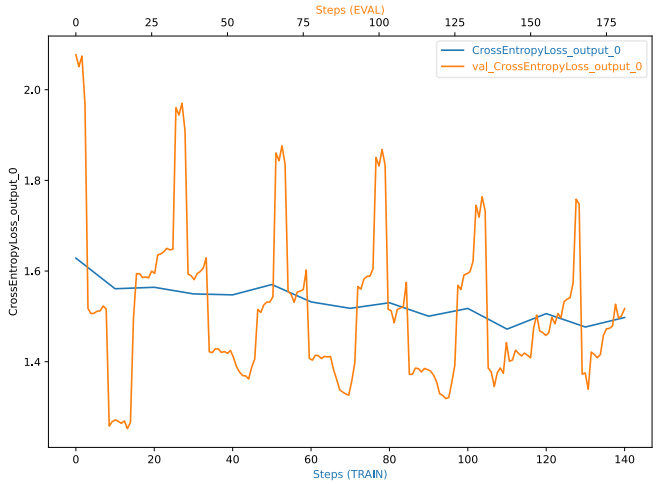
The model used for training is the ResNet50 model. In order to perform transfer learning, the weights of the convolutional layers are frozen and the replaced fully connected layer with an output of 5 is optimized only.

For the process of model training, the mini-batch gradient descent algorithm is used which is composed of the following tasks. The model is trained with the whole data set for training and validation over multiple iterations called epochs. For each epoch, the datasets are split into batches. For the forward pass, the model classifies the images from one batch before the cross-entropy loss function calculates the error. For the backward pass, the fully connected

layers are then optimized by the Adam algorithm. The training scripts provides a break condition so the training process stops after an epoch if the accuracy does not improve furthermore.

Once training is done, the final accuracy and loss is calculated by classifying the images from the testing dataset. After that, the model is saved to the S3 bucket of the SageMaker instance.

Figure 4 shows the loss during the training process on the left side. On the right side, the achieved accuracy of 29.68 % on the testing set can be seen. The graph shows that the cross-entropy loss does not stagnate after five epochs which is discussed in chapter 3.3.




```
Epoch 5, Phase valid
  Images [64/2086 (3%)] Loss: 1.75 Accuracy: 5/64 (7.81%)
  Images [704/2086 (34%)] Loss: 1.44 Accuracy: 264/704 (37.50%)
  Images [1344/2086 (64%)] Loss: 1.50 Accuracy: 449/1344 (33.41%)
  Images [1984/2086 (95%)] Loss: 1.54 Accuracy: 602/1984 (30.34%)
Start testing ...
Testing Loss: 1.4797724075899779
Testing Accuracy: 29.67557251908397
Saving model to '/opt/ml/model/model.pth' ...
```

Figure 4: Cross-entropy loss and accuracy

3.2.4 Deployment of an End Point

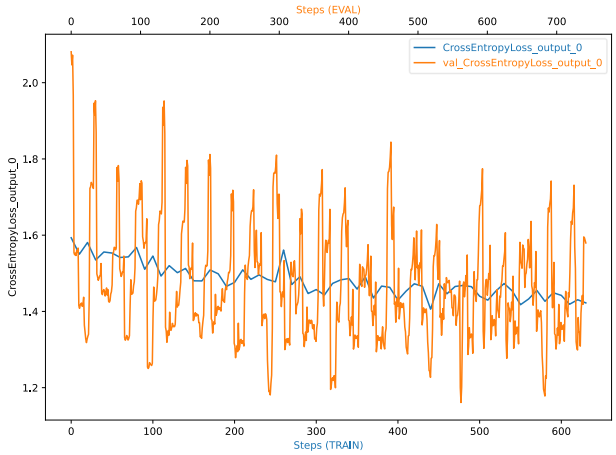
The trained model stored in S3 is deployed with the “inference.py” Python script as an entry point. Since the computational load is not as heavy as while training, a cheaper instance of type “ml.m5.large” was chosen.

Prediction can be made either by providing an image as byte blob or by specifying an URL to an image in Json format.

3.3 Refinement

Because the loss described in chapter 3.2.3 did not stagnate, a second model was trained with a number of 30 epochs. The break condition was also changed so that the training stops after the tenth time instead of the first time when the accuracy does not improve. This resulted in cross-entropy loss shown in Figure 5. As with the model before, the loss does not stagnate. The training stops after 22 epochs. The accuracy on the testing set improved to a value of

32.54 %. It may be that the accuracy could be improved even further by deactivating or refactoring the breaking condition.




```
Epoch 22, Phase valid
  Images [64/2086 (3%)] Loss: 1.46 Accuracy: 24/64 (37.50%)
  Images [704/2086 (34%)] Loss: 1.42 Accuracy: 301/704 (42.76%)
  Images [1344/2086 (64%)] Loss: 1.45 Accuracy: 493/1344 (36.68%)
  Images [1984/2086 (95%)] Loss: 1.66 Accuracy: 640/1984 (32.26%)
Start testing ...
Testing Loss: 1.431278185080026
Testing Accuracy: 32.538167938931295
Saving model to '/opt/ml/model/model.pth' ...
```

Figure 5: Cross-entropy loss and accuracy after 22 epochs

4 Results

4.1 Model Evaluation and Validation

To check if the model works correctly, the end point is queried with a Jpeg byte blob and with a Json object for the image 524.jpg, both with the SageMaker SDK as shown in Figure 6. The output shows that the model works and prediction was in that case correct.

```
image_number = 524
request_dict={ "url": f"https://aft-vbi-pds.s3.amazonaws.com/bin-images/{image_number}.jpg" }
metadata = requests.get(f"https://aft-vbi-pds.s3.amazonaws.com/metadata/{image_number}.json").content
expected_quantity = json.loads(metadata)["EXPECTED_QUANTITY"]
```

```
# Jpg request
img_bytes = requests.get(request_dict['url']).content
response_jpg = predictor.predict(img_bytes, initial_args={"ContentType": "image/jpeg"})

print(f"Expected number of items: {expected_quantity} ({request_dict['url']})")
print("Predicted number of items: " + classes[np.argmax(response_jpg, 1)[0]])
```

```
Expected number of items: 2 (https://aft-vbi-pds.s3.amazonaws.com/bin-images/524.jpg)
Predicted number of items: 2
```

```
response_json=predictor.predict(json.dumps(request_dict), initial_args={"ContentType": "application/json"})

print(f"Expected number of items: {expected_quantity} ({request_dict['url']})")
print("Predicted number of items: " + classes[np.argmax(response_jpg, 1)[0]])
```

```
Expected number of items: 2 (https://aft-vbi-pds.s3.amazonaws.com/bin-images/524.jpg)
Predicted number of items: 2
```

Figure 6: Requesting prediction at the end point

Nevertheless, the measured accuracy of 32.54 % on the testing set for the second trained model means that only every third image is classified correctly. As shown in the graph for the

cross-entropy loss in Figure 5, the model could improve more when training over more epochs. A better improvement would be to train the model on multiple instances with a larger dataset instead of 10,441 images of 500.000 images (2 %).

4.2 Justification

Compared to the benchmark model introduced in chapter 2.4, the model from this project does not perform very well. The benchmark model achieved an accuracy of 55.2 % on the testing set while the accuracy of the model from this project was nearly 20 % lower. This could be due to large amount of data the benchmark mark model is trained with. Another reason could be that features were extracted from the images for the benchmark model. Despite the fact that both models were optimized with the Adam algorithm, the parameters could differ.

5 Conclusion

Overall, this classification task was very challenging. While the trained model did not achieve similar results as the benchmark model, some aspects were pointed out how to get higher accuracy for this model. Thus, with more resources and time better results could be achieved. Regardless of the challenging task, or perhaps because of it, this project really instructive and helped to consolidate the knowledge gained during the nano degree course.

6 Bibliography

[1] Amazon.com, Inc., "Amazon Bin Image Dataset," [Online]. Available: https://registry.opendata.aws/amazon-bin-imagery/. [Accessed 20 02 2022].

[2] P. R. Bertorello, S. Sripada and N. Dendumrongsup, "Amazon Inventory Reconciliation using AI," [Online]. Available: https://github.com/pablo-tech/Image-Inventory-Reconciliation-with-SVM-and-CNN/blob/master/ProjectReport.pdf. [Accessed 20 02 2022].

[3] E. Park, "Amazon Bin Image Dataset(ABID) Challenge," [Online]. Available: https://github.com/silverbottlep/abid_challenge. [Accessed 20 02 2022].

[4] Udacity, Inc., "Project Overview: Inventory Monitoring at Distribution Centers," [Online]. Available: https://github.com/udacity/nd009t-capstone-starter/blob/master/starter/file_list.json. [Accessed 20 02 2022].

[5] Amazon.com, Inc., “Amazon Bin Image Dataset - Documentation - Github,” [Online]. Available: https://github.com/awslabs/open-data-docs/tree/main/docs/aft-vbi-pds#about-the-data. [Accessed 22 02 2022].

[6] E. Park, "Dataset statistics - Amazon Bin Image Dataset(ABID) Challenge," [Online]. Available: https://github.com/silverbottlep/abid_challenge#22-dataset-statistics. [Accessed 20 02 2022].

[7] Torch Contributors, "Docs - CROSSENTROPYLOSS," [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html. [Accessed 20 02 2022].

[8] Torch Contributors, "Docs - ADAM," [Online]. Available: https://pytorch.org/docs/stable/generated/torch.optim.Adam.html#torch.optim.Adam. [Accessed 20 02 2022].

[9] Torch Contributors, "Docs - TORCHVISION.MODELS," [Online]. Available: https://pytorch.org/vision/stable/models.html. [Accessed 22 02 2022].