Capstone project report – Inventory Monitoring at Distribution Centers

Abdelrhman Elruby

April 7, 2023

1 Project hyperlinks

Useful links for the project description:

- Github repository: Udacity-AWS-ML-project-5
- Project proposal document: project_proposal/project proposal.pdf
- Proposal review: Review #3984555

2 Project Overview

Delivery and logistics systems are essential to many sectors of the economy today. The COVID-19 outbreak demonstrated the serious effects of even minor disruptions in the logistical system. As a result, large warehouses need automated systems that can sort and distribute goods based on a variety of factors, including delivery address and material type. In these warehouses, shipments and other products are typically transported in boxes, each of which can accommodate many items. The automated delivery system's job would be to categorize the package and choose where to deliver it. The system first has to know how many items are in each delivery box, and the proposed Capstone project will focus on that.

To count the number of objects in a box, I'd like to use the Amazon Bin Image Dataset, which contains a photo of the box's contents as well as other object features such as the number of objects. The implemented solution will estimate the number of objects based on this image.

3 Problem Definition

This section will identify the issue at hand, look at various fixes, and discuss performance indicators.

3.1 Goal of the project

The project's major objective is to develop a method for counting the number of items in each bin based on a picture of the contents of the bin. To forecast future false detection rates, the provided model should have known accuracy.

A machine learning model that can count the items in a box based on a picture of the box's contents will be the project's answer. To determine the quality of the given model, accuracy will be computed.

3.2 The datasets and inputs

I utilised the Amazon Bin Image Dataset for the research. Only about 10,000 photos tagged with the number of elements in a bin from this dataset were used. After post-processing, the dataset will be split into train/valid and test subsets.

The dataset is described by Amazon as having "nearly 500,000 photos and metadata from bins of a pod in an active Amazon Fulfillment Center. The bin pictures in this dataset were shot inside an Amazon Fulfillment Center while robot units relocated pods as part of routine operations.

A chosen subset of around 10,000 items is separated into three subgroups for training, testing, and validation with matching set ratios of 60%-20%-20%. The number of photos per class in the chosen datasets will be balanced, i.e., there will be an equal number of test, train, and validation images for each class (number of objects in bin). In order to reduce the bias effect in the result network, classes must be balanced.

3.3 Strategy and realization plan

The project calls for categorising the input photographs into various types. I could be accomplished via traditional image processing and segmentation techniques, where each object is identified, or by utilising an image classification network trained using machine learning. I will use the latter choice since this Udacity course is focused on ML application.

For machine learning to train the network for the task, a lot of resources are needed. I can perform calculations on a home computer or on a cloud platform like Amazon. The network could be trained locally, and cloud computing could be used for the final training steps in this example, but because the nanodegree I'm pursuing concentrates on AWS cloud applications, I'd like to test my talents in that area. Cloud applications make it easier to deploy and integrate models by giving users access to a wide range of computing devices. However, it appears like the cloud interface is always changing. Many of the tutorials utilised in the Udacity course are already outmoded. because AWS changed its user interface and included some new functionalities.

A trained network model file that I might use to count the number of things in the delivery bin will be the project's anticipated outcome.

3.4 Baseline model selection

I want to utilise the resnet50 image classification network as a baseline model. A convolutional neural network with 50 layers is called ResNet-50. A network that has been pretrained using more than a million photos from the ImageNet database is available on the AWS cloud. A number of animals, a keyboard, a mouse, and a pencil are among the 1000 different object categories that the pretrained network can classify images into. The network has therefore acquired rich feature representations for a variety of images. The network accepts images with a resolution of 224 by 224.

The usage of a pretrained network is justified since it reduces the amount of time required to obtain reliable results and because it has a higher chance of generalising a solution and operating on new, untested data.

3.5 Evaluation Metrics

Cross Entropy Loss function and precision will be used to assess the training process's effectiveness (accuracy). The solution will be measured and quantified using these parameters, and their repeatability will enable subsequent advancements. To make sure that the training technique will produce the desired results and won't over-fit the dataset, a hyper parameter optimisation may be required.

4 Data analysis

In this section, I'll give a brief summary of the dataset and use visualisations and data exploration to analyse the issue and determine what characteristics and algorithms would be most useful for fixing it.

4.1 Dataset overview

Around 500,000 photos and related metadata from bins of a pod in an active Amazon Fulfillment Center are included in the Amazon Bin Image Dataset. I utilised the given JSON file with a list of 10441 photos grouped into 5 classes with a range of 1, 2, 3, and 4 items in order to reduce training time and Amazon costs.

Sample images from the dataset in Figure 1.

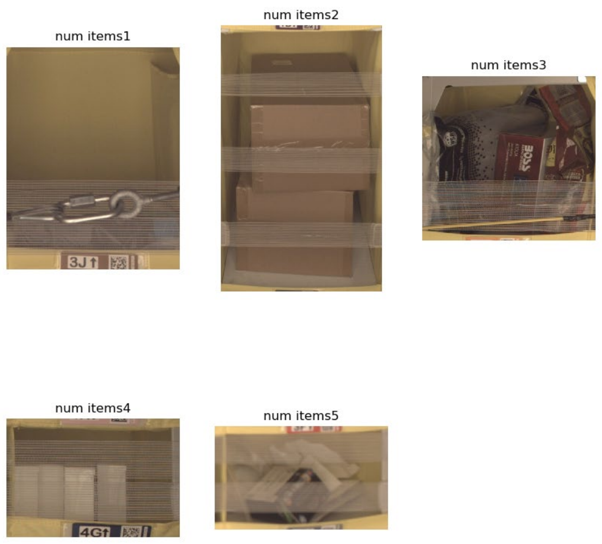


Figure 1: Sample images from the dataset

As there are just 5 classes in the project, the distribution of samples can be shown as a list:

- Single object – 1228 images – 12% of the dataset,
- Two objects – 2299 images – 22% of the dataset,
- Three objects – 2666 images – 25% of the dataset,
- Four objects – 2373 images – 23% of the dataset,
- Five objects – 1875 images – 18% of the dataset,

However, for more classes, it would be ideal to display the datased distribution as a histogram, as seen in Figure 2.

Although there are fewer photographs of a single object and five objects in a bin than there are of other images, each class makes a significant contribution to the dataset. It is essential to guarantee that the model will run consistently accurately no matter how many objects are in the bin. The network might not generalise sufficiently to distinguish each class in the dataset if the numbers varied significantly more.

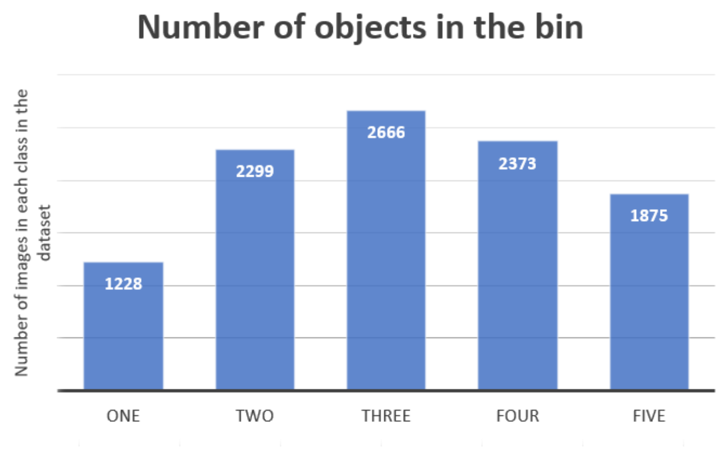


Figure 2: Histogram of the class distribution in used dataset.

4.2 Model selection

The project's objective is to classify the images. The pretrained image classification network, which has already been trained and is capable of processing some picture data, is the ideal choice for completion. Because I am familiar with the input and output data structures of the resnet50 network and because it is readily available in pytotch models, I chose this network. There may be other image classification networks that are suitable for the task.

5 Model implementation

Using Sagemaker notebooks in the AWS domain, the project was established with certain choices to control network training and future development costs. The following are the primary project implementation steps:

1. Data preparation
   - fetching data from a database,
   - pre-process the data to separate it into subsets for testing, training, and validation
   - upload the data to S3 container
2. Model tuning
   - tune hyperparameters of the model
   - train a machine learning model and check for training abnormalities.
   - measure KPI metrics, plot training results
3. Model deployment
   - confirm that the model is performing as anticipated
   - observe the quality of the outcome object

Each part of this procedure will be described in this section.

5.1 Udacity project requirements

This is a quotation from the Udacity course's project specification that was finished for the current project.

> You will need to complete the following tasks in order to complete this assignment:
>
> 1. Upload Training Data: First you will have to upload the training data to an S3 bucket.

2. Writing a script to train a model on that dataset is required once you have completed step one.
3. Use SageMaker to train your model: Finally, you must utilise SageMaker to execute that training script

5.2 Data preparation

It was necessary to separate the 10441 downloaded photos into train, test, and validation subsets. I used the 60-20-20 formula to divide the photos for this project:

- Train: 60%
- Test: 20%
- Valid: 40%

The network accepts only RGB images with resolution 224 × 224 pixels therefore each image has to be resized and vectorized to match the input layer of the network.

5.3 Data augmentation – preprocessing steps

Bins may be rotated in distribution centres, but the network should still be able to count the items independent of box rotation. So that the mirrored images produce the same outcomes in the network, I randomly flip the photos during the training phase.

Also, the network needs to count both smaller and larger objects in the trash. So that different sized photographs will provide the same results, I employ random image cropping during the resize process.

Preprocessing steps applied:

- Random image cropping – from the original image randomly crop smaller image with size of 224 × 224px. This allows to artificially extend the training dataset and ensure that differently scaled images will present the same results.
- Random image flipping – flip image horizontally, hence the network should be proof against different orientation of the bin or the objects within
- Data serialization – 2D image had to be converted into 1D tensor of data in order to properly pass them to the model

I purposefully disabled colour space normalisation for the photos since I was interested in seeing how the model would perform and determine the normalisation factor on its own. This task was made achievable by the fact that photographs from the Amazon Bin Image Dataset were gathered under similar lighting conditions and are generally comparable in terms of colour space. It would be ideal to calculate the mean and standard deviation for the photographs on a full dataset, which I did not want to download because of the per-project charges associated with using AWS.

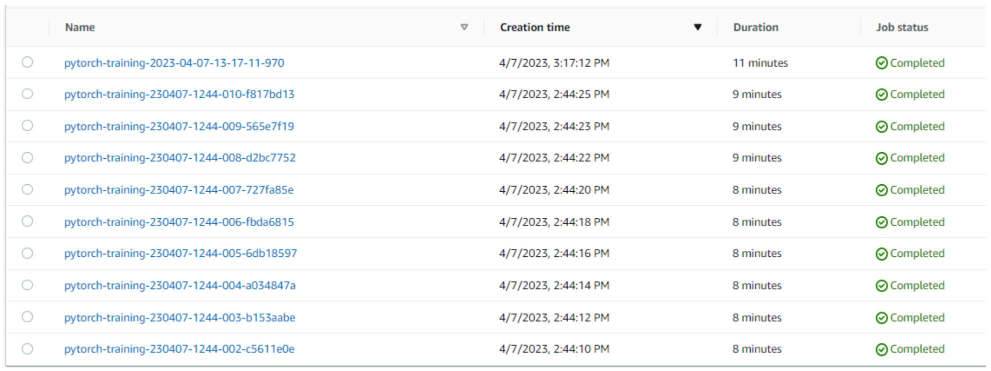
5.4 Hyperparameters tuning

I began the tuning process with the streamlined script "hpo.py," which runs just one epoch on a subset of the training data. The following ranges of learning rate and batch size hyperparameters were adjusted using this script:

- Learning rate was tuned for range: $(0.001, 0.1)$ - found optimal value is 0.0015230721261324884.
- Batch size was tuned for values: $\{32, 64, 128, 256, 512\}$ - found optimal value is 32.

To search the 2D parameter space in the aforementioned ranges, hyper-parameter tuning was used. Figure 3 shows a screenshot of the list of finished AWS jobs together with their metric.

Based on the fact that each task represents a particular set of hyper-parameters, we may portray the dependence of each hyper-parameter against the metric value as a mesh, as shown in Figure 4.



Name	Creation time	Duration	Job status
pytorch-training-2023-04-07-13-17-11-970	4/7/2023, 3:17:12 PM	11 minutes	✓ Completed
pytorch-training-230407-1244-010-f817bd13	4/7/2023, 2:44:25 PM	9 minutes	✓ Completed
pytorch-training-230407-1244-009-565e7f19	4/7/2023, 2:44:23 PM	9 minutes	✓ Completed
pytorch-training-230407-1244-008-d2bc7752	4/7/2023, 2:44:22 PM	9 minutes	✓ Completed
pytorch-training-230407-1244-007-727fa85e	4/7/2023, 2:44:20 PM	8 minutes	✓ Completed
pytorch-training-230407-1244-006-fbda6815	4/7/2023, 2:44:18 PM	8 minutes	✓ Completed
pytorch-training-230407-1244-005-6db18597	4/7/2023, 2:44:16 PM	8 minutes	✓ Completed
pytorch-training-230407-1244-004-a034847a	4/7/2023, 2:44:14 PM	8 minutes	✓ Completed
pytorch-training-230407-1244-003-b153aabe	4/7/2023, 2:44:12 PM	8 minutes	✓ Completed
pytorch-training-230407-1244-002-c5611e0e	4/7/2023, 2:44:10 PM	8 minutes	✓ Completed

Figure 3: List of hyper-parameter tuning jobs with output cross entropy metric value

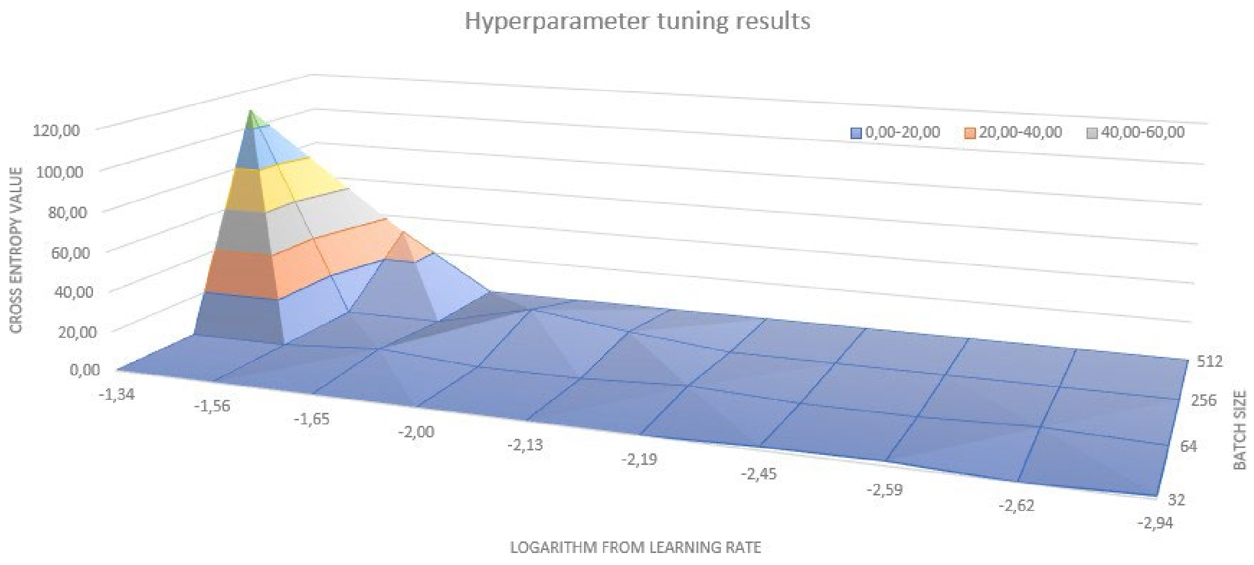


Figure 4: Hyper-parameter value dependency to the training metric. Lower value the better.

For the majority of the examined samples, the hyper-parameter tuning plot (Figure 4) shows that the curve is rather flat. That leads us to the conclusion that any learning rate value below 0.01 for any batch size would produce a successful model training operation.
I chose the best job's hyperparameters as a starting point for the training:

- Learning rate: 0.001
- Batch size: 32

5.5 Model training procedure

After determining which hyperparameters might be the best, I started the training process. The "train.py" file contains the training's programming code. The file is set up to run as a standalone script on your computer or on inexpensive spot instances, or from within a Sagemaker notebook (sample usage in "sagemaker.ipynb"). After 2 hours of execution, the training for the 10441 files was finished in 3 epochs.

6 Model training results

The "train.py" file has been set up such that it can be executed from both SageMaker and Spot instances in AWS. We have access to a variety of debugging tools in Sagemaker to make sure the training is proceeding properly, but the equipment utilised there is expensive.

6.1 Sagemaker Instance training

I only execute the training in Sagemaker for a portion of the dataset, and Figure 6 shows the Cross Entropy loss function.

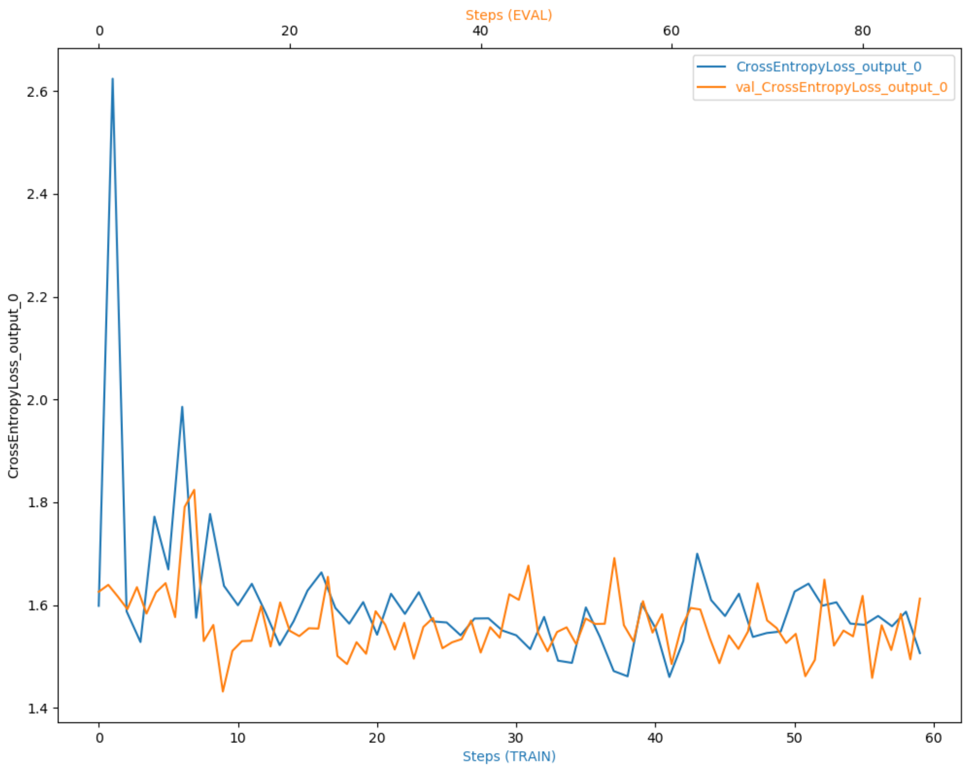


Figure 6: Cross entropy from Sagemaker training

6.2 Step Instance training

To keep the expense of the training process to a minimum, I chose the "Step instances" for the entire training. I produced debug printouts in the "train.py" file that allowed me to see the training process as it was finally completed. With the AWS CloudWatch service, SageMaker users can view the same logs. Figures 7 and 8 exhibit the loss function and running accuracy graphs, respectively.

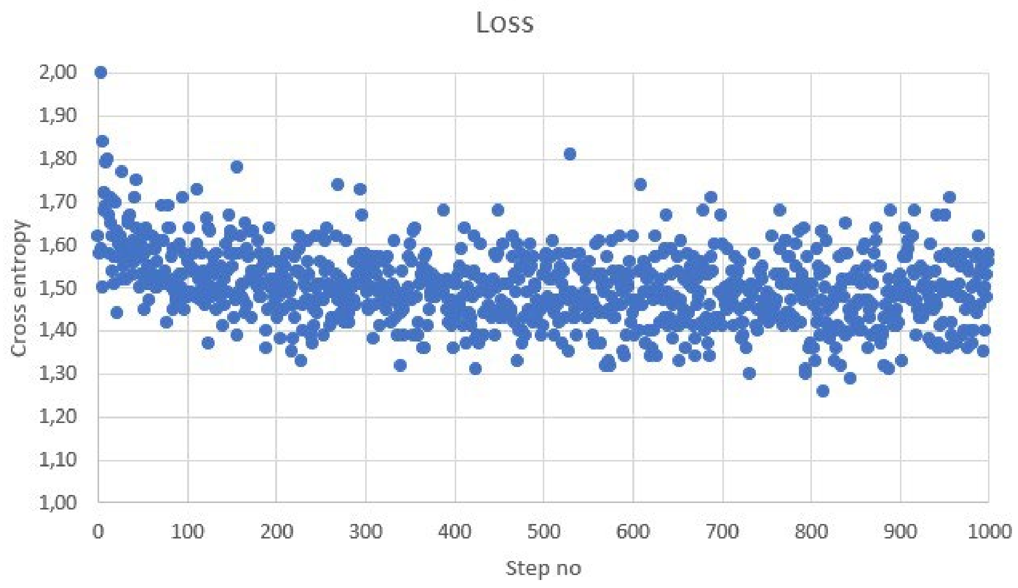


Figure 7: Cross entropy loss function changes during the training process.

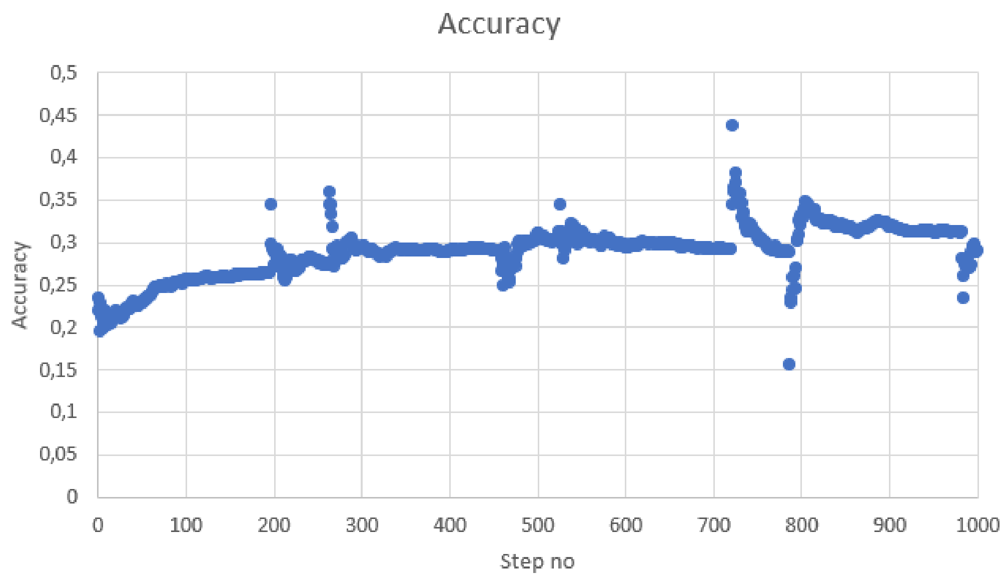


Figure 8: Running accuracy of the model through the training

6.3 Model deployment

Following the creation of the trained model, I deployed it as a Sagemaker Endpoint and evaluated its performance on a test picture. I developed the "inference.py" script for the model inference, which can be used from Lambda or Sagemaker in the AWS cloud. In the "sagemaker.ipynb" notebook, example code showing how to deploy the project and run sample inference is included.

6.4 Comparison of different pre-trained models

I trained the network using several hyperparameter configurations in order to compare results between different jobs and evaluate how different models could behave.

For comparison, I trained network using:

- Learning rate: 0.2
- Batch size: 32

Given that the tuning technique rejected learning rates greater than 0.01 based on the hyperparameters training job, I may anticipate that this model will perform poorly.

With the error that the model loss started rising between epochs, the trained model collapsed after just two iterations. In Figure 9, the accuracy plot is displayed.

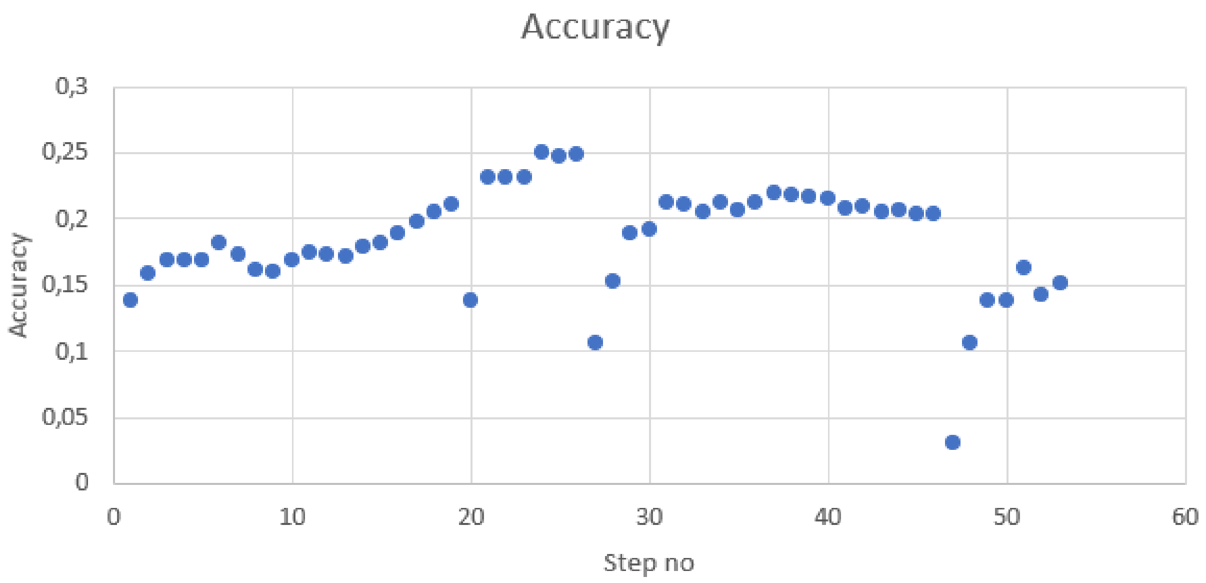


Figure 9: Running accuracy of the additional model for comparison

Compared to the reference model with tweaked hyperparameters, the results demonstrate that this model actually produces less accurate results (see Figure 8 for comparison).

7 Conclusions

The trained model's ultimate accuracy is roughly 0.3, which is not the ideal outcome. The achieved accuracy is reasonable, though, given that I only used 5% of the Amazon Bin Images Dataset and kept the training process to a minimal. But, it should be improved by performing training on a larger portion of the dataset.

The hyperparameter tuning process was carried out correctly, and the chosen hyperparameter values are suitable for continued development, as demonstrated by comparison with the extra models. This further emphasises the significance of the hyperparameter tuning step and suggests that even limited search could result in noticeably improved results for the trained model.

However, the created model for counting objects in a bin is insufficient to resolve the stated issue. Only 30% of the measurements produced by the model with an accuracy of 0.3 will be accurate, and 70% of the bins will be wrongly categorised. This model may serve as a verification tool for additional complicated networks that characterise more specific object properties, such as colours, sizes, labels, etc., as stated in the project specification. In order to cross-check other system components, the desired model accuracy should be relatively high for that reason. I would recommend training the model on a larger portion of the dataset and possibly enabling training even while the overall loss function is expanding in order to improve the model.

An approach used in optimisation approaches known as "annealing" may momentarily raise object entropy, allowing it to decline more than previously.

During this project I learned how to use AWS cloud to deploy ML solutions following steps:

1. Download dataset, preprocess it and store in S3,
2. Train the network using Sagemaker and Spot istances
3. Deploy a model as an endpoint and run inference on the model.