Inventory Monitoring at Distribution Centers

Capstone Project Report

1. Definition

1.1 Project Overview

Inventory monitoring is an essential process at distribution centers. It involves counting inventory objects and ensures that the correct quantities are maintained to meet supply chain demands. There has been a great need to automate the process to meet the growing increase in consumer demand and eliminate errors.

Distribution centers use robots to move objects in bins that can contain multiple objects. Convolutional Neural Networks (CNNs) have achieved excellent results for object detection in images [1-4], text detection [5] vehicle detection [6] and many other applications.

The aims of this project were to:

1. Use machine learning techniques to build a model that can classify the number of objects in each bin to assist in automating the inventory monitoring process [16].
2. Demonstrate a machine learning engineering pipeline.

The Amazon Bin Image Dataset (ABID) [18] was used to build the model. The system can be used to track inventory and ensure that delivery consignments have the correct number of items [16].

1.1.1 Domain background

Inventory monitoring has evolved throughout the years to save time, lower costs and increase efficiency. Ancient methods involved using tally sticks (50,000 years ago) and clay tokens (4000 years ago) to count objects. Herman Hollerith developed the first modern automatic computation machine (1880). It recorded information using punch cards. The barcode was invented by Norman Woodland (1940). Lasers were later used to scan barcodes to track inventory (1960). Inventory tracking became even more efficient with more advanced computers and software (1980s and 1990s). Barcode readers could instantly update business databases therefore it wasn't necessary to input data by hand (2000s). RFID (Radio-frequency identification) tags (patented in 1970s) became widely used in warehouses, factories, and retail stores (2000s) [7, 8].

Research in computer vision and machine learning, seeks to improve and further automate inventory monitoring. Approaches to count objects in images include using shape information [9], background subtraction in vehicle detection and counting systems [10], fuzzy color histograms [11], visual features, clustering and artificial neural networks [12], support vector machines [13] and CNNs [13-15, 19].

1.2 Problem statement

Inventory monitoring systems face the following problems: employee errors, stock shortages, excess inventory, misplaced inventory and lack of optimization [15, 17]. Physically counting or scanning objects is a tedious and time - consuming process that may lead to human error, fatigue, reduced efficiency and ultimately monetary losses.

The automation of inventory monitoring systems by leveraging computer vision and machine learning techniques will assist to minimize these problems and eliminate human error. This will in turn assist in saving costs, time and increase efficiency in inventory monitoring.

The problem of classifying the number of objects could be replicated by using the ABID [18]. The use of CNNs to solve this problem is a growing research area [13-15, 19] that has shown promising results. A pre-trained CNN model was fine-tuned and trained to classify the number of objects in each bin image. A list of probabilities of the classes was returned where the maximum was the predicted number of objects in the bin.

The following steps were required to complete the project:

1. Data was obtained from a database and preprocessed.
2. An initial model was trained with fixed hyperparameters to count the number of objects in an image and its accuracy and RMSE were measured.
3. Hyperparameter tuning was performed to find the best hyperparameters to improve the model.
4. A final model was trained with the best hyperparameters to count the number of objects in an image and its accuracy and RMSE were measured.
5. Endpoints were deployed and queried with images to make predictions.
6. Performance of the model was evaluated and validated by comparing it to a benchmark.
7. Multi-instance GPU training was performed.

1.3 Metrics

Accuracy and Root Mean Square Error (RMSE) are standard metrics used to evaluate model performance. Accuracy and RMSE are defined in Figure 1, where *1* is an indicator function, *p* is the prediction, *g* is the ground truth and *N* is the number of samples in the dataset [19].

$$\text{Accuracy: } \frac{1}{N}\sum_{i=1}^{N} 1[p_i == g_i] \qquad \text{RMSE: } \sqrt{\frac{1}{N}\sum_{i=1}^{N}(p_i - g_i)^2}$$

Figure 1: Accuracy and RMSE equations [19].

2. Analysis

2.1 Data Exploration

The ABID [18] will be used to classify the number of objects in each bin. It contains over 500 000 JPEG images and the corresponding JSON metadata from bins of a pod in an Amazon Fulfillment Center (AFC). The bin images are captured as robots transport pods during normal AFC operations [18].

Images are found in the bin-images directory, and metadata for each image is found in the metadata directory. Images and their corresponding metadata share numerical unique identifiers [20]. An example image and its corresponding metadata are shown in Figures 2 and 3, respectively.

The metadata contains information about the objects in the bin such as the Amazon Standard Identification Number (ASIN), name, normalized name, quantity, weight and size. The images and their corresponding metadata will serve as inputs to the training algorithm. Individual instances of objects will be counted separately therefore two of the same objects in a bin, will be counted as two [19].

A random storage scheme is used where objects are stored in accessible bins with available space, therefore each bin's contents is random. Each bin image may show only one type of object or many different types of objects. If objects are misplaced the contents of some bin images may not match the inventory record [20].

The tapes in front of the bins prevent objects from falling out of the bins and occasionally reduce visibility of the objects. Objects may be greatly occluded by other objects or limited image viewpoint [19].

Figure 2: Bin image 777.jpg [22].

```
{    "BIN_FCSKU_DATA":
  {       "B0067EQF9I":
   {           "asin": "B0067EQF9I",
    "height": {              "unit": "IN",              "value": 2.2          },
    "length": {              "unit": "IN",              "value": 4.9          },
    "name": "PUR Gum Aspartame Free Wintergreen Gum, 9 Count (Pack of 12)",
    "quantity": 3,
    "weight": {              "unit": "pounds",              "value": 0.6          },
    "width": {              "unit": "IN",              "value": 3.8          }       }    },
    "EXPECTED_QUANTITY": 3,
    "image_fname": "777.jpg"}
```


Figure 3: JSON metadata for image 777.jpg [23].

Statistics of the data are shown in Table 1. Figure 4 shows the distribution of quantity of objects in a bin. About 90% of the images have less than ten objects in a bin [19].

Table 1: Data statistics [19].

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

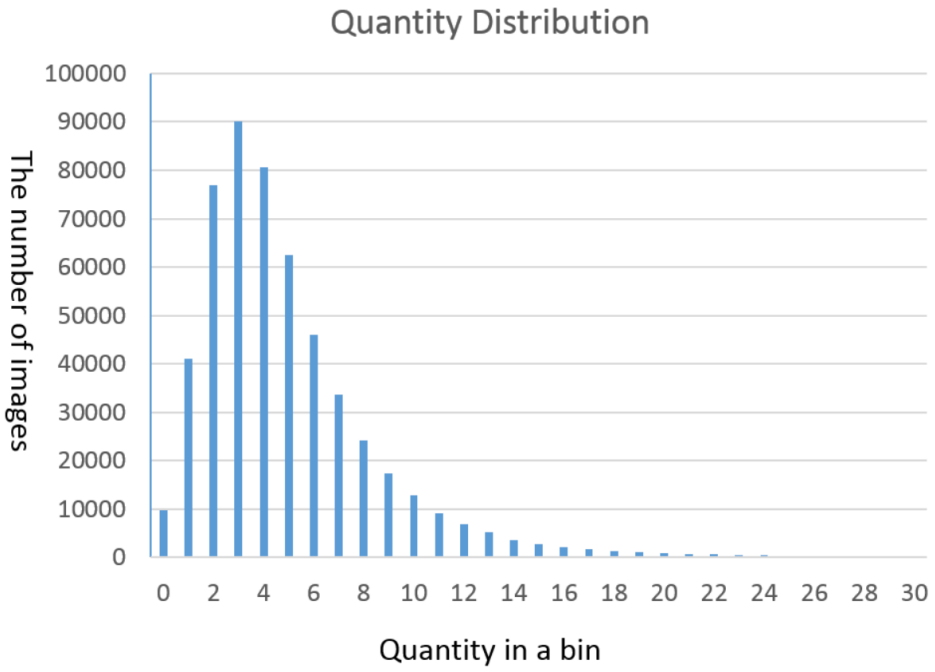

Figure 4: Distribution of quantity of objects in a bin [19].

2.2 Exploratory Visualization

Due to the large size of the data, a small subset of the data will be used to stay within the allocated budget. The data subset consists of folders where the name of each folder is equal to the number of objects in each bin image. For example, in folder 5, each image has 5 objects. The data subset contains 10441 bin images. The distribution of the number of products per bin in the data subset is shown in Figure 5. The range of the number of products is from 1 to 5, therefore there are 5 classes. The most common number of products per bin is 3.

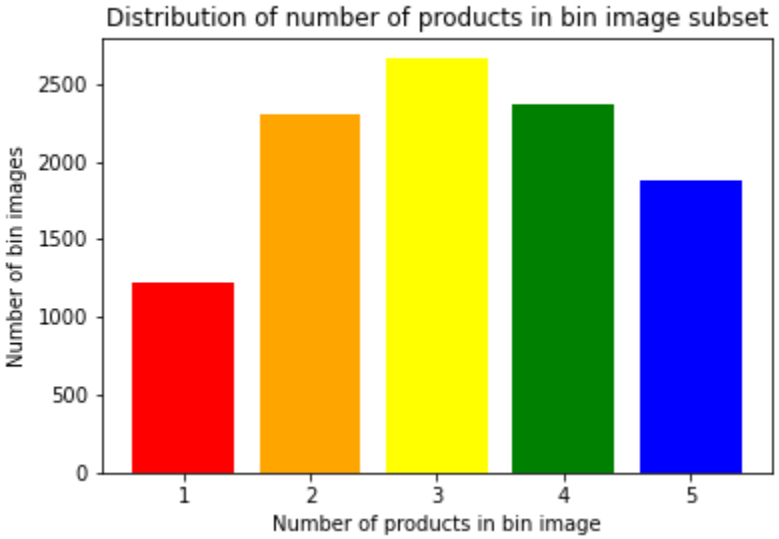

Figure 5: Distribution of the number of products in bin image subset.

2.3 Algorithms and Techniques

The following algorithms and techniques were used to solve the multi-class counting classification task.

2.3.1 ResNet

ResNet [4] has achieved excellent results for image recognition tasks. ResNets are trained on huge image datasets like ImageNet which assists to them to learn general features therefore they can be applied on various datasets and tasks [28].

The ResNet50 [4] pre-trained PyTorch model [25] was used for the classification task because it has the best accuracy and speed, and a small model size when compared to other models [26] as shown in Figure 6. ResNet50 consists of 48 convolution layers, 1 max pooling layer and 1 average pooling layer. It has 3.8 billion FLOPs and is extensively used [4, 27]. The images and their corresponding metadata (labels) will serve as inputs to the training algorithm.

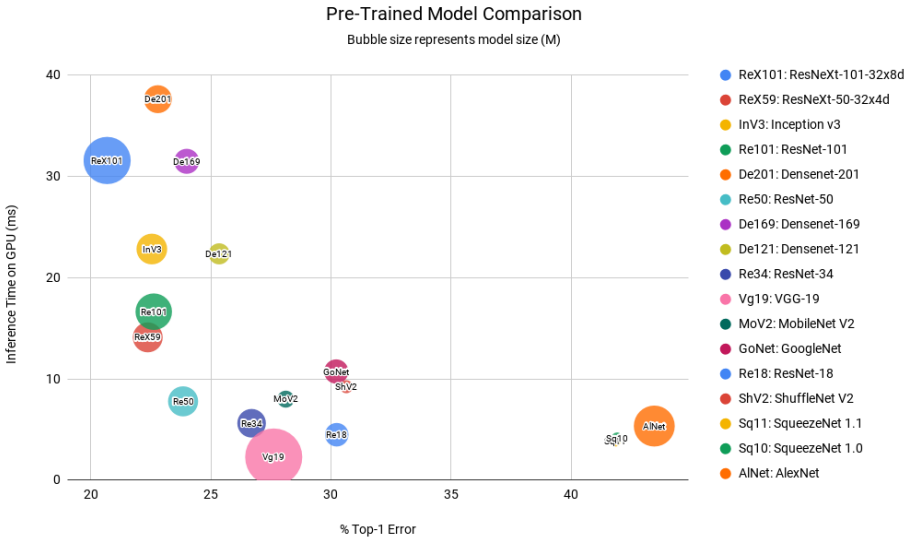

Figure 6: Pre-trained model comparison [26].

2.3.2 Adaptive Moment Estimation

Adaptive Moment Estimation (Adam) [31] is an optimization algorithm that can be used in place of Stochastic gradient descent. It combines the advantages of AdaGrad and RMSProp algorithms therefore it can cope with sparse gradients on noisy tasks. It is simple to configure and the default hyperparameters perform well on most tasks therefore it usually requires minimal tuning [32]. Adam is recommended as the best choice when compared to other optimization algorithms [33].

PyTorch uses the default parameters for Adam: *learning rate*=0.001, *beta1*=0.9, *beta2*=0.999, *epsilon*=1e-8, where *beta1* is the exponential decay rate for the first moment estimates, *beta2* is the exponential decay rate for the second-moment estimates, and *epsilon* is a minute number that avoids division by zero [32].

2.3.3 Cross - Entropy Loss Function

The Cross - entropy loss function is widely used to optimize classification models. It results in quicker training and better generalization as compared to sum-of-squares for classification tasks [29, 30]. The image predictions and labels are inputs to the Cross - entropy loss function.

2.3.4 Transfer Learning

Transfer Learning involves fine-tuning pre-trained models on new datasets for new tasks. To fine-tune a pre-trained CNN the following steps are taken:

1. The convolutional layers are frozen.
2. Fully connected layers are added

3. The model is trained.

During training, the forward pass runs for the whole network, and the backward pass to update weights runs for only the fully connected layers, because the convolutional layers are frozen. This allows for faster training with the improved accuracy of using a CNN for predictions [28].

2.3.5 Hyperparameter Tuning

Hyperparameter tuning with SageMaker Tuner involves searching a hyperparameter space to find the best hyperparameters. The hyperparameters used were:

1. Learning rate: Controls how much to adjust the weights of the network with respect to the loss function [34].
2. Batch size: The number of training samples used in one iteration [35].

2.3.5 Multi-Instance Training

Multi-Instance training is needed for large models or data. The model or data is split between different devices during training. There are two methods in PyTorch to split models and data across multiple GPUs: torch.nn.DataParallel (DP) and torch.nn.DistributedDataParallel (DDP). Although DP is simpler to use, DDP has the following advantages:

1. The devices can all be on the same instance (node) or spread across multiple instances.

2. Only gradients are passed between the devices therefore network communication has a lower bottleneck [36].

The DDP class [37] implements distributed data parallelism. The container parallelizes the implementation of the model. The input is split across the defined devices by chunking in the batch dimension. The model is copied on each machine and each device, and each copy processes part of the input. In the backwards pass, the gradients from each instance are averaged.

The NVIDIA Collective Communication Library (NCCL) backend is the fastest and provides the best distributed GPU training performance [37]. The SageMaker distributed training library optimizes communication between AWS ML compute instances, resulting in greater device utilization and quicker training times [38].

2.4 Benchmark

The ABID Challenge by Eunbyung Park [19] will be used as a benchmark to compare results. ResNet34 was trained and an accuracy of 55.67% and RMSE of 0.930 was obtained.

3. Methodology

3.1 Data Preprocessing

Data was obtained from a database and uploaded to Amazon S3 for training the model. The data was randomly split into training (60%), validation (20%), and testing (20%) sets in the create_data_loaders function in the training scripts (section 3.2.3).

```
trainset, testset, validset = torch.utils.data.random_split(dataset, lengths)
```

The data was preprocessed. In the training scripts, images were resized to 224×224, randomly flipped, converted to a to tensor and normalized before being passed the model.

Low quality images such as images with occlusions, unclear or blurry images and images where the contents of the bin images do not match the inventory record, could be removed to improve performance. For example, in image 777.jpg it seems like there are 2 objects in the image but the metadata expected quantity is 3 (Figures 2 and 3). However, 1 object could be occluded and not visible to the untrained eye.

3.2 Implementation

The project involved the following:

3.2.1 Software and platforms

AWS SageMaker, Amazon S3, Amazon CloudWatch, Jupyter notebook, Python, PyTorch, various software libraries and good machine learning engineering practices were used.

3.2.2 Jupyter Notebook

The sagemaker.ipynb notebook was used to:

1. Download the data with the file_list.json file, explore it, and upload it to S3.
2. Define estimators, the tuner, training containers, hyperparameters for hyperparameter tuning, rules and hooks for the debugger and profiler.
3. Perform training jobs and hyperparameter tuning.
4. Display the debugger profiling report and plot a debugger output.
5. Deploy endpoints and make predictions.

3.2.3 Script files

The training scripts are:

hpo.py was used for hyperparameter tuning. Different hyperparameters were used to train the model to find the best hyperparameters, which gave the lowest loss for the model.

train.py was used to train the model with the best hyperparameters and perform debugging and profiling.

train_multi.py was used to perform multi-instance GPU model training. with the NCCL backend, DDP and the best hyperparameters.

The inference script is:

inference.py was used to deploy the trained model to an endpoint. The image was deserialized, processed (resized, converted to a tensor, normalized and unsqueezed) and used to make a prediction.

3.2.4 Instance selection

In AWS SageMaker studio, the ml.t3.medium instance was used. It is low cost and has 2 vCPUs and 4 GiB memory therefore it has sufficient computing power for the project. It also has fast launch [39, 40].

The ml.g4dn.xlarge instance was used for faster performance to train the models in a container environment and model deployment. It has 1 GPU, 4 vCPUs and 16 GiB memory therefore it has sufficient computing power for training. It costs the lowest compared to the other GPU instances [39].

3.2.5 Transfer learning

To fine-tune the pre-trained PyTorch ResNet50 model the steps in section 2.3.4 were taken. Two fully connected layers (nn.Linear) and a rectified linear unit (nn.ReLU) layer were added. The output number of classes was set to 5 for the counting classification task. The changes are shown below.

```
num_classes = 5
model.fc = nn.Sequential(
                  nn.Linear(2048, 128),
                  nn.ReLU(inplace=True),
                  nn.Linear(128, num_classes))
```

3.2.6 Adam

Adam was implemented with the default parameters in PyTorch (section 2.3.2**)** and the specified learning rate.

3.2.7 Cross - entropy loss function

Cross - entropy loss function was implemented with the default parameters in PyTorch [41].

3.2.8 Model training

The initial and final model training is described in sections 3.3.1 and 3.3.3, respectively.

3.2.9 Hyperparameter Tuning

Hyperparameter tuning is described in section 3.3.2.

3.2.10 Debugger and profiler

SageMaker debugger and profiler were used to provide insights for improvements and log metrics for the final model. Hooks were included in the train.py script to enable debugging and profiling. The Cross - entropy loss was logged.

3.2.11 Model deployment

The initial and final models were each deployed to SageMaker endpoints, with the inference.py script. Each endpoint was queried with images to make predictions.

3.2.12 Performance evaluation

The accuracy and RMSE of the model were measured during training, validation and testing in the training scripts (section 3.2.3), in the train and test functions. The number of samples, predictions and data labels were inputs to the metrics.

3.2.13 Multi-Instance training

Multi-Instance GPU model training was performed with the train_multi.py script. The NCCL backend, DDP and the best hyperparameters were used. Two instances of ml.g4dn.xlarge were used to save costs.

3.3 Refinement

3.3.1 Initial model

The initial model was trained with the hpo.py script with the default hyperparameters shown in Table 2. These hyperparameters are commonly used. The testing performance is shown in Table 3.

Table 2: Initial model hyperparameters

Learning rate	0.01
Batch size	64

Table 3: Initial model performance

Loss	99.75794517632687
Accuracy	16
RMSE	1.4317821063276353

An endpoint was deployed with the initial model with the inference.py script. It was queried with images to make predictions. The first image is shown in Figure 7.

Figure 7: Bin image 923.jpg [42].

The expected quantity in the corresponding metadata is 4 [43]. The results for the first image are shown below. The maximum is in bold (index=2). The model could not predict the correct result (index=3).

```
[-0.48211732506752014,
  0.08623732626438141,
  0.21010656654834747,
  0.14803989231586456,
  -0.0743698999285698]
```

The second image is shown in Figure 2. The expected quantity in the corresponding metadata is 3 [23] but it looks like there are 2 objects in the image. The results for the second image are shown below. They are the same as the results for the first image.

```
[-0.48211732506752014,
  0.08623732626438141,
  0.21010656654834747,
  0.14803989231586456,
  -0.0743698999285698]
```

The maximum is in bold (index=2). The model predicted the correct result as per the metadata but if the number of objects is taken as 2 (index=1), the result would be incorrect. The poor performance could be due to the small size of the dataset and/or hyperparameters that need to be further tuned.

3.3.2 Hyperparameters tuning

Hyperparameters tuning was performed with SageMaker to find the best hyperparameters and refine the model. The hyperparameter spaces in Table 4 were used. The best hyperparameters are shown in Table 5 and were used to train the final model.

Table 4: Hyperparameter spaces

Learning rate	0.001 to 0.1
Batch size	[32, 64, 128, 256, 512]

Table 5: Best hyperparameters

Learning rate	0.00371599290323504
Batch size	32

3.3.3 Final model

The testing performance is shown in Table 6. The final model's loss and RMSE improved but the accuracy decreased as compared to the initial model's performance in Table 3.

Table 6: Final model performance

Loss	45.05898605693471
Accuracy	10
RMSE	0.8660254037844386

SageMaker debugger and profiler were used to train the final model. The debugger profiling report recommended the following:

LowGPUUtilization and Batch size rules:

Use a smaller instance type or increase the batch size because the training job underutilized the instance.

4. Results

4.1 Model Evaluation and Validation

An endpoint was deployed with the final model. It was queried with the same images as the initial model to make predictions [22, 42]. The results for the first image [22] are shown below. The maximum is in bold (index=2). The model could not predict the correct result (index=3).

```
[-1.8109267950057983,
  0.2871607542037964,
  0.66116863489151,
  0.5736399292945862,
  0.3940010368824005]
```

The results for the second image [42] are shown below.

```
[2.741820812225342,
  1.6072783470153809,
  -0.2160869836807251,
  -2.114128589630127,
  -3.8471338748931885]
```

The maximum is in bold (index=0). The model could not predict the correct result (index=2). If the number of objects is taken as 2 (index=1), the result would be also be incorrect.

The final model is not robust enough to solve the problem presently and needs further improvement (section 5.3) as it does not produce the correct results. The hyperparameters of the model may need further tuning. The final model was tested with various images and does not generalize well to unseen data.

4.2 Justification

The performance for the benchmark and final model is compared in Table 7.

Table 7: Performance comparison

	Benchmark	Final model
Accuracy	55.67	10
RMSE	0.930	0.866

The final model has lower accuracy and RMSE compared to the benchmark. The pipeline to build the model follows the standard procedure [44]. However, the model is unable to accurately classify the number of objects in an image therefore it needs further improvement (section 5.3).

5. Conclusion

5.1 Free-Form Visualization

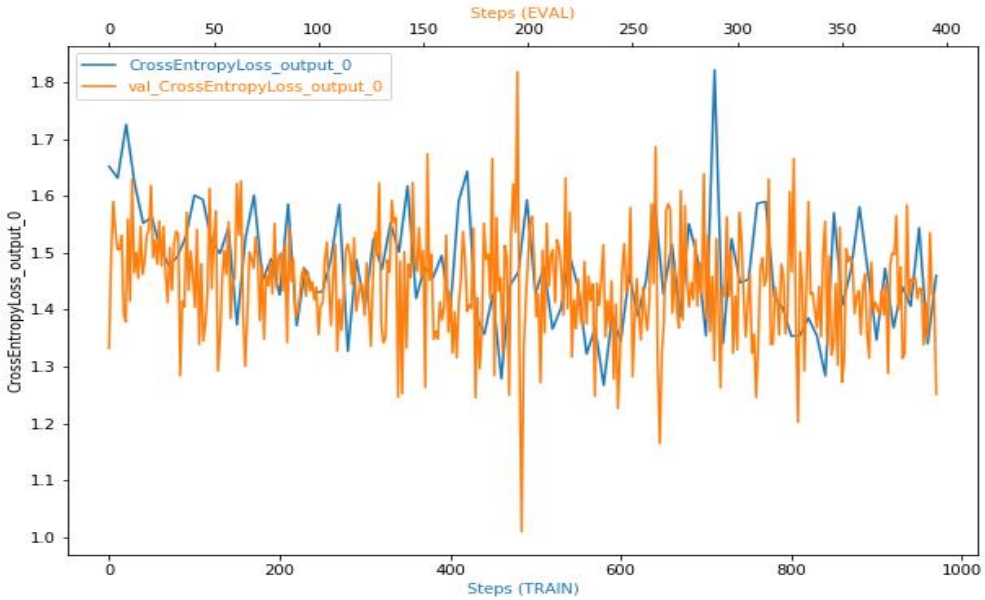

Figure 8: Cross - entropy loss.

The Cross - entropy loss is shown in Figure 8. It is difficult to determine if the model is learning because of the numerous oscillations. These are likely due to the small batch size [45] or outliers [46]. If a larger batch size is used and the outliers are removed, the training curve may smooth out, and show a decrease in loss, which indicates the model is learning. This maybe the case, as the training loss decreases from about 1.65 to 1.45, and the cross validation loss decreases from about 1.33 to 1.25.

However, if the training curve is still flat, it indicates that the model is not learning and the learning rate needs to be changed. The number of input and output features, in the fully connected layers that were added on, could be modified. If the training curve shows an increase in loss, it indicates that the model is diverging and a smaller learning rate needs to be used.

5.2 Reflection

Inventory monitoring is an essential process at distribution centers. It involves counting inventory objects and ensures that the correct quantities are maintained to meet supply chain demands. There has been a great need to automate the process to meet the growing increase in consumer demand and eliminate errors.

The aims of this project were to:

1. Use machine learning techniques to build a model that can classify the number of objects in each bin to assist in automating the inventory monitoring process [16].
2. Demonstrate a machine learning engineering pipeline.

The following steps were required to complete the project:

1. Methods to solve the problem were researched.
2. AWS SageMaker, Amazon S3, CloudWatch, Jupyter notebook, Python, PyTorch and other libraries were used to provide a solution.
3. A subset of the ABID was obtained from a database and preprocessed.
4. A pre-trained CNN (ResNet50) was selected for the counting classification task.
5. An initial model was trained with fixed hyperparameters to count the number of objects in an image and its accuracy and RMSE were measured.
6. Hyperparameter tuning was performed to find the best hyperparameters to improve the model.
7. A final model was trained with the best hyperparameters to count the number of objects in an image and its accuracy and RMSE were measured.
8. Endpoints were deployed and queried with images to make predictions.
9. Performance of the model was evaluated and validated by comparing it to a benchmark.
10. Multi-instance GPU training was performed.

The use of a CNN to classify the number of objects in an image is an interesting research area. However, it was challenging to get high accuracy for the model. (The tuner was run twice but it produced similar results each time.) The model needs further improvement in order to be used in general settings to solve this type of problem.

5.3 Improvement

Low quality images such as images with occlusions, unclear or blurry images and images where the contents of the bin images do not match the inventory record, could be removed to improve performance. This would have been very time - consuming if performed manually. An automated method would be better.

Training on a larger dataset may improve performance. Tuning with more training jobs may result in better hyperparameters. Large batch sizes may prevent oscillations. The network architecture that was added onto the pre-trained model for fine-tuning could be modified. The number of input and output features, in the fully connected layers and the number of layers could be modified. Alternative pre-trained models such as EfficientNet, Inception and VGG [47] could be tried.

The hpo.py and train.py scripts have code in the test function that saves text files of the predictions and corresponding ground truth if the parameter evaluate = True. These text files are inputs to the evaluate.py script that allows one to compute the per class accuracy and RMSE. The text files can only be saved when training locally. When training on SageMaker in a container environment, alternative methods such as PutObject [48] and custom resources [24] could be used to save similar objects that can be used as inputs to evaluate.py. This will allow one to compute the per class accuracy and RMSE.

References

[1] Redmon, J. & Farhadi, A. 2018. YOLOv3: An incremental improvement. arXiv preprint, arXiv1804.02767.

[2] Fu, C-Y., Liu, W., Ranga, A., Tyagi, A. & Berg, A.C. 2017. DSSD: Deconvolutional single shot detector. arXiv preprint, arXiv:1701.06659.

[3] https://github.com/ultralytics/yolov5, Accessed 18/1/22.

[4] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778.

[5] He, T., Huang, W., Qiao, Y. and Yao, J., 2016. Text-attentional convolutional neural network for scene text detection. IEEE transactions on image processing, 25(6), pp.2529-2541

[6] Chen, X., Xiang, S., Liu, C.L. and Pan, C.H., 2014. Vehicle detection in satellite images by hybrid deep convolutional neural networks. IEEE Geoscience and remote sensing letters, 11(10), pp.1797-1801.

[7] https://www.citycleanandsimple.com/2018/07/16/history-of-inventory-management-technology/#:~:text=The%20earliest%20form%20of%20inventory,%E2%80%9Ctally%20sticks%E2%80%9D%20to%20count.&text=Over%20time%2C%20inventory%20management%20developed,ancient%20Greek%20and%20Egyptian%20societies, Accessed 18/1/22.

[8] https://www.barcodedirect.com/the-evolution-of-inventory-management/, Accessed 18/1/22.

[9] Wählby, C., Sintorn, I.M., Erlandsson, F., Borgefors, G. and Bengtsson, E., 2004. Combining intensity, edge and shape information for 2D and 3D segmentation of cell nuclei in tissue sections. Journal of microscopy, 215(1), pp.67-76.

[10] Mandellos, N.A., Keramitsoglou, I. and Kiranoudis, C.T., 2011. A background subtraction algorithm for detecting and tracking vehicles. Expert Systems with Applications, 38(3), pp.1619-1631.

[11] Verma, N.K., Goyal, A., Chaman, A., Sevakula, R.K. and Salour, A., 2015, June. Template matching for inventory management using fuzzy color histogram and spatial filters. In 2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA), pp. 317-322.

[12] Sharma, T., Rajurkar, S.D., Molangur, N., Verma, N.K. and Salour, A., 2019. Multi-faced object recognition in an image for inventory counting. In Computational Intelligence: Theories, Applications and Future Directions-Volume II, pp. 333-346.

[13] Rodriguez Bertorello, P., Sripada, S. and Dendumrongsup, N., 2018. Amazon Inventory Reconciliation Using AI. Available at SSRN: https://ssrn.com/abstract=3311007, Accessed 18/1/22.

[14] Verma, N.K., Sharma, T., Rajurkar, S.D. and Salour, A., 2016, October. Object identification for inventory management using convolutional neural network. In 2016 IEEE Applied Imagery Pattern Recognition Workshop (AIPR), pp. 1-6.

[15] Kalahiki, C.B., 2020. Computer Vision for Inventory Management, Master of Science Thesis, College of Engineering & Science Louisiana Tech University.

[16] https://classroom.udacity.com/nanodegrees/nd189/parts/cd0549/modules/864d3e12-dc8d-47c6-b443-d01d6c7aedde/lessons/d6ec6005-e421-455c-9b79-ebc2df44e1ac/concepts/ad3d3503-f9a9-4c18-86ae-c03d6fbb1470, Accessed 18/1/22.

[17] https://bizfluent.com/info-8596246-common-problems-inventory-systems.html, Accessed 18/1/22.

[18] https://registry.opendata.aws/amazon-bin-imagery/, Accessed 18/1/22.

[19] https://github.com/silverbottlep/abid_challenge, Accessed 18/1/22.

[20] https://github.com/awslabs/open-data-docs/tree/main/docs/aft-vbi-pds, Accessed 18/1/22.

[21] https://pytorch.org/vision/stable/models.html, Accessed 18/1/22.

[22] https://aft-vbi-pds.s3.amazonaws.com/bin-images/777.jpg, Accessed 22/1/22.

[23] https://aft-vbi-pds.s3.amazonaws.com/metadata/777.json, Accessed 22/1/22.

[24] https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-custom-resources.html, Accessed 19/2/22.

[25] https://pytorch.org/vision/stable/models.html, Accessed 19/2/22.

[26] https://learnopencv.com/pytorch-for-beginners-image-classification-using-pre-trained-models/, Accessed 19/2/22.

[27] https://www.google.com/amp/s/iq.opengenus.org/resnet50-architecture/amp/, Accessed 19/2/22.

[28] https://classroom.udacity.com/nanodegrees/nd189/parts/cd0387/modules/44f321a8-8d42-43a6-a682-9d3b6b0e5bc5/lessons/aa5128b4-d31a-4456-8ee6-70d5262ab97d/concepts/32d6bbb8-b6de-4818-b1ab-96b9850a5169, Accessed 19/2/22.

[29] Bishop, C.M. and Nasrabadi, N.M., 2006. Pattern recognition and machine learning, vol. 4, p. 738. New York: Springer.

[30] https://machinelearningmastery.com/cross-entropy-for-machine-learning/, Accessed 19/2/22.

[31] Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[32] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/, Accessed 19/2/22.

[33] Ruder, S., 2016. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

[34] https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10, Accessed 19/2/22.

[35] https://radiopaedia.org/articles/batch-size-machine-learning#:~:text=Batch%20size%20is%20a%20term,iteration%20and%20epoch%20values%20equivalent , Accessed 19/2/22.

[36] https://yangkky.github.io/2019/07/08/distributed-pytorch-tutorial.html, Accessed 19/2/22.

[37] https://pytorch.org/docs/stable/generated/torch.nn.parallel.DistributedDataParallel.html#torch.nn.parallel.DistributedDataParallel, Accessed 19/2/22.

[38] https://docs.aws.amazon.com/sagemaker/latest/dg/distributed-training.html, Accessed 19/2/22.

[39] https://aws.amazon.com/SageMaker/pricing/, Accessed 19/2/22.

[40] https://docs.aws.amazon.com/SageMaker/latest/dg/notebooks-available-instance-types.html, Accessed 19/2/22.

[41] https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html#torch.nn.CrossEntropyLoss, Accessed 19/2/22.

[42] https://aft-vbi-pds.s3.amazonaws.com/bin-images/923.jpg, Accessed 19/2/22.

[43] https://aft-vbi-pds.s3.amazonaws.com/metadata/923.json, Accessed 19/2/22.

[44] https://classroom.udacity.com/nanodegrees/nd189/parts/cd0387/modules/44f321a8-8d42-43a6-a682-9d3b6b0e5bc5/lessons/a822f040-5481-4d51-8e34-950ee5e76ae9/concepts/0aff4cec-3ada-489f-8ea6-df74decfde1e, Accessed 19/2/22.

[45] https://stackoverflow.com/questions/68529287/why-does-my-learning-curves-shows-spikes-or-fluctuations, Accessed 10/1/22.

[46] https://stats.stackexchange.com/questions/303857/explanation-of-spikes-in-training-loss-vs-iterations-with-adam-optimizer, Accessed 10/1/22.

[47] https://towardsdatascience.com/4-pre-trained-cnn-models-to-use-for-computer-vision-with-transfer-learning-885cb1b2dfc, Accessed 19/2/22.

[48] https://docs.aws.amazon.com/AmazonS3/latest/API/API_PutObject.html, Accessed 19/2/22.