Machine Learning Project

Machine Learning Engineer Nanodegree

Carlos Alberto Villarreal Campos

Jul 4th, 2023

1. DEFINITION

1.1 Project Overview

Knowing exactly the count of a physical inventory has been a key issue in all industries that are involved in manufacturing and retail processes. Historically, this is a job that warehouse managers and supply chain professionals have performed through *cycle counting*, an audit process for managing inventory counts. Cycle counting always requires strategic planning to achieve high accuracy in stock recording without distracting staff from their essential tasks [1].

While this type of procedure might be valid for some particular industries, especially where the primary goal is to attain a balance between physical inventory counts and the inventory records (which are typically maintained on ERP platforms) there are other industries where time planning and staff time is prohibitive, especially in systems where a high level of automation has been implemented (i.e., distribution centers that use robots in their operations)

In these high-tech environments, a solution is required that can directly interact with robots, allowing for real-time counting of objects. Even such a solution could also be of great help in less technical environments with a proper interface that updates inventory records in ERP platforms. The benefits of an accurate inventory can be seen in [2]

1.2 Problem Statement

In this project, a solution for object counting problem is presented. The solution uses the Pytorch framework available from AWS SageMaker. The goal is to deploy a SageMaker endpoint that will query images to get predictions about the number of objects in the submitted image.

1.3 Metrics

The performance of the model obtained will be measured through the following metric:

$$\text{Accuracy: } \frac{1}{N}\sum_{i=1}^{N} 1[p_i == g_i]$$

Where, 1 is an indicator function (returns 1 when the event occurs and 0 otherwise), and *p* and *g* is prediction and ground truth respectively. [3]

2. ANALYSIS

2.1 Data Exploration

For this project, the Amazon Container Image dataset is used, which contains over 500,000 container images and metadata from a pod in a working Amazon fulfillment center. Container images in this dataset are captured when robot units transport pods as part of normal Amazon Fulfillment Center operations [4]. Since this is a large dataset, in order not to exceed the budget available for the project, a subset of the dataset has been used.

The subset consists of 10.441 images which were split into train and test datasets (80% of data were used for training and 20% of data for testing). The final distribution of classes is presented in table 1.

Class	Description	Total number of images	Images for training	Images for testing
1	Bins with one object	1228	982	245
2	Bins with two objects	2299	1839	459
3	Bins with three objects	2666	2132	533
4	Bins with four objects	2373	1898	474
5	Bins with five objects	1875	1500	375

Table 1. Class distribution

In the image below there is an example of a bin with one object.

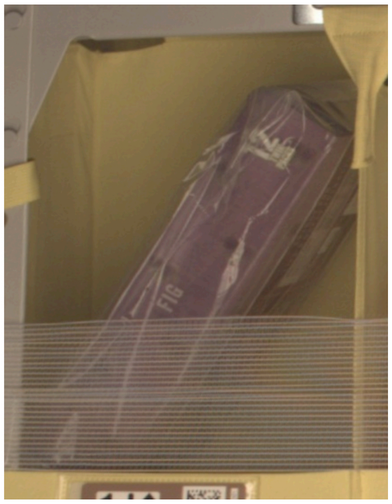

Figure 1. Bin with one object

2.2 Algorithms and Techniques

The first objective was to build an Image Classification Model that can detect the number of objects in the images presented. For this, the Pytorch framework available from AWS SageMaker was used, which generates an Amazon-built Docker container that executes functions defined in the supplied *entry_point* script within a SageMaker Training Job.[5]

Given that, a small subset is used for training purposes, a pre-trained Resnet50 Convolutional Neural Network was used for the model.

The loss function selected was Cross Entropy and as optimizer the SGD algorithm was defined.

2.3 Benchmark

The results obtained in this project will be compared with the accuracy obtained in the Amazon Bin Image Dataset (ABID) Challenge [3] where a 34-layer Resnet architecture trained from scratch was used.

3. METHODOLOGY

3.1 Data Preprocessing

As mentioned earlier, the subset used in the project were split into train and test datasets. In order to make data available for SageMaker, the train and test sets were upload to a S3 bucket.

Furthermore, some image transformations were necessary so that the data available in the train and test datasets had a format compatible with the training algorithms. Several transformations available in *torchvision.transforms* were used (i.e: *Normalize, ToTensor, Resize*)

3.2 Implementation

Once data was proper processed, three Training Scripts (python files) and a Submission Script (a jupyter notebook) were configured.

The three Training Scripts were:

- *train.py:* Used for training the first model (fixed hyperparameters)
- *hpo.py:* Used for executing the hyperparameter tuning
- *trained_model.py:* Used for training the model with the best hyperparameters, adding debugging and profiling features

The three Training Scripts have the same general structure, and the following tasks were executed:

- To read hyperparameters (defined in the Submission Script) and additional information required, that is passed as environment variables.

- To transform and load the data available in the S3 bucket in a format that can be understood by the train and test functions implemented.

- To create and configure the model that will be trained. In this case a ResNet50 model

- To define the optimizer used during the training process.

- To perform the training process according to the parameters defined in the Submission Script.

On the other hand, in the Submission Script the following tasks will be performed:

- To specify the hyperparameters for training and fine-tuning.

- To create and configure the estimator that uses the training script, defined earlier, as entry point.

- To define the infrastructure required for the training and endpoint deploying processes.

- To submit the training and deploying jobs.

Finally, an endpoint was deployed in order to query predictions to the model from the images submitted. In this case, as entry model the *deployment.py* script was used.

3.3 Refinement

The initially trained model obtained an accuracy of 26%. To obtain a better performance a hyperparameter training was carried out, which delivered the following hyperparameters:

- Learning rate: 0.0769169840479485
- Epochs: 10
- Batch-size: 64

By retraining the model with these hyperparameters, a slight improvement in accuracy was achieved. The new accuracy was 28%.

4. RESULTS

4.1 Model Evaluation and Validation

Due to the poor performance of the model, some debugging and profiling were configured. From the figure 2, where the train and test losses are plotted, it is clear that it was not possible to obtain an adequate training of the model since neither of the two plots shows a significant decay as the steps increase.

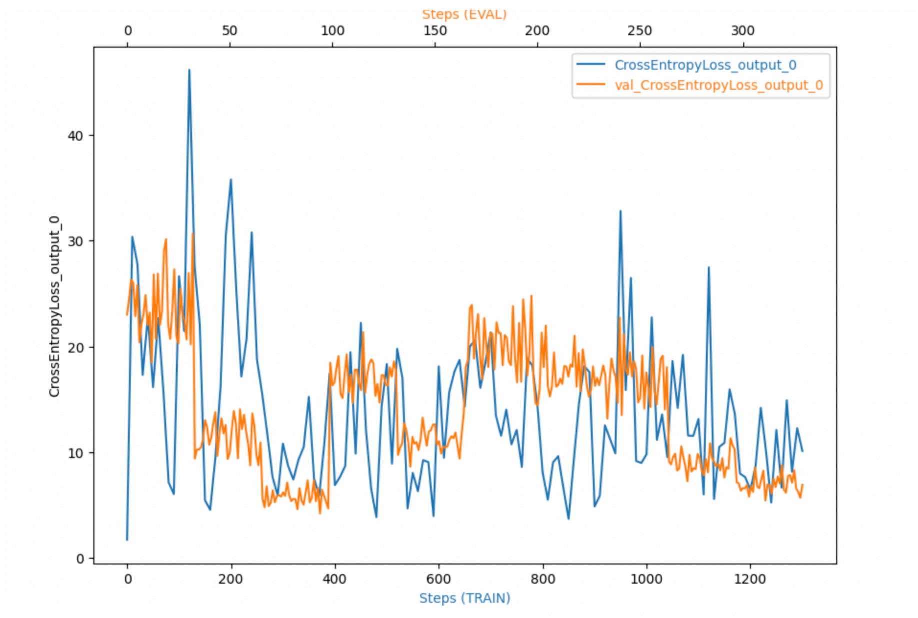

Figure 2. Train and test losses

In addition, the profiling report also shows some infrastructure issues that could affect model performance.

- Significant time is spent in phase “others”. Should be checked what is happening in between the steps. See figure 3.

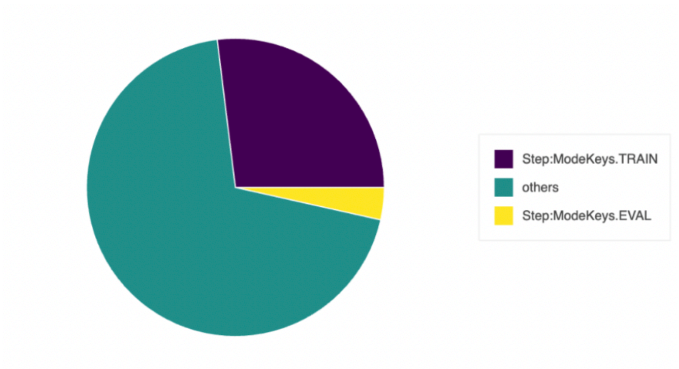

Figure 3. Time spent on the TRAIN/EVAL phases and others

- The *GPUMemoryIncrease* rule was triggered 244 times, which could indicate that a larger instance type should be selected.

- The *LowGPUUtilization* rule was triggered 10 times, which could indicate the existence of bottlenecks and the need for adjust batch sizing or include distribute training.

- The *StepOutlier* rule was triggered 5 times, which indicates also the existence of bottlenecks or system stall.

The whole profiler report can be seen in the file *profiler-report.pdf*

4.2 Justification

Compared to the 55.6% accuracy obtained in the Amazon Bin Image Dataset (ABID) Challenge [3], the 28% accuracy obtained in the present project seems quite low. That result is somehow expected, given that this exercise only uses 2% of the data used in the Amazon Bin Image Dataset (ABID) Challenge.

However, the main objective of this project was achieved, and a complete pipeline for the Inventory Monitoring at Distribution Centers was built, which includes the follow stages:

- Data Fetching
- Data Preprocessing
- Model Training
- Hyperparameter Tuning
- Debugging and Profiling
- Deployment

Using that pipeline, it will be possible to implement additional model trainings with more data in order to get better model performances. Given the results of the profiler-report it will be also interesting try larger instances and multi-instance training.

REFERENCES

[1] Inventory Cycle Counting 101: Best Practices & Benefits. https://www.netsuite.com/portal/resource/articles/inventory-management/using-inventory-control-software-for-cycle-counting.shtml

[2] Physical Inventory. https://www.wallstreetmojo.com/physical-inventory-count

[3] https://github.com/silverbottlep/abid_challenge

[4] https://registry.opendata.aws/amazon-bin-imagery/

[5]https://sagemaker.readthedocs.io/en/stable/frameworks/pytorch/sagemaker.pytorch.
html