Inventory Monitoring at Distribution Centers (Project Report)

Muhammad M. Atout

1. Domain Background:

The complexity and speed requirements of logistics tasks are constantly increasing. Product identification and traceability are increasingly important necessities for many modern producers. Goods are printed, marked, engraved, etched, or labeled with various barcodes, characters, and symbols to identify contents and provide vital manufacturing information. In addition, more and more goods need to be unpacked, inspected, packaged, and distributed in an ever shorter time.

Modern image processing software provides all the necessary methods and technologies for reading characters, codes, and symbols, regardless of how they were applied. Objects can also be identified entirely without codes by visual texture features and color, for example.

Many processes in logistics and warehouse automation can benefit from the use of computer vision. Warehouse processes can be partially and fully automated with computer vision systems.

The scope of this project would focus on a single branch of inventory monitoring which is to automate the process that ensures that every bin contains the correct number of products.

2. Problem Statement:

With the rising cost (salaries, operation cost, human errors) of the workforce that is required for manual inventory management comes the need for more effective and cost-optimized inventory management methods.

Inventory management automation would eliminate a lot of human-related costs (operation/error costs) and one of the most state-of-the-art used techniques in Inventory management automation is Computer vision.

Computer vision enables real-time data collection via images and video collected by drones, phones, robots, and fixed cameras on shelves, in stores, and in warehouses. Computer vision-enabled software tracks stock detects damaged or mislabeled items, analyzes and forecasts peak- and off-peak demand for particular items and stores, and even orders products from suppliers.

For years, computer vision has been transforming business solutions across industries, from self-driving cars to healthcare. Advances in this kind of artificial intelligence, which relies on machine learning (ML) models for object detection and recognition, have made computer vision competitive with human vision.

3. Solution Statement:

As stated before we will use computer vision to help automate the count of the products in each bin.

In this project, we will fine-tune a pre-trained CNN model to produce a robust model that can choose the count of items in each pin by looking at its image.

Project Component:

Dataset or the Image Source

The Amazon Bin Image Dataset is to be used to train the model. It contains over 500,000 images and metadata from bins of a pod in an operating Amazon Fulfillment Center.

Algorithm

We will fine-tune the Resnet50 pretraind CNN. The model hyperparameters will also be tuned to find out the best hyperparameters.

Environment or the platform

We are going to use the AWS platform (Sagemaker - S3).

4. Benchmark Model:

The result of the Amazon Bin Image Dataset (ABID) Challenge will be considered a Benchmark. The author has achieved an accuracy of approximately 55%. This project will try to achieve or exceed this mark.

5. Evaluation Metrics:

The overall **accuracy** of the classification can be used to evaluate the performance of the trained model.

6. Dataset or the Image Source:

The Amazon Bin Image Dataset is to be used to train the model. It contains over 500,000 images and metadata from bins of a pod in an operating Amazon Fulfillment Center. The bin images in this dataset are captured as robot units carrying pods as part of normal Amazon Fulfillment Center operations.

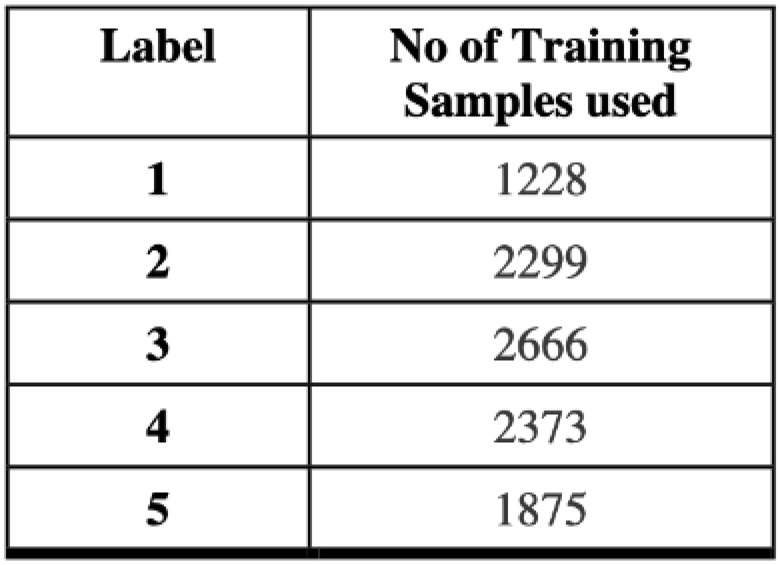
Documentation about the open source dataset can be found here. These are some typical images in the dataset. A bin contains multiple object categories and a various number of instances.

The corresponding metadata exists for each bin image and it includes the object category identification(Amazon Standard Identification Number, ASIN), quantity, size of objects, weights, and so on. The size of bins varies depending on the size of objects in them. The tapes in front of the bins are for preventing the items from falling out of the bins and sometimes it might make the objects unclear. Objects are sometimes heavily occluded by other objects or limited viewpoints of the images.

The Images are extracted from the source which is available in JPEG format and the target or label data is extracted from the corresponding JSON file.

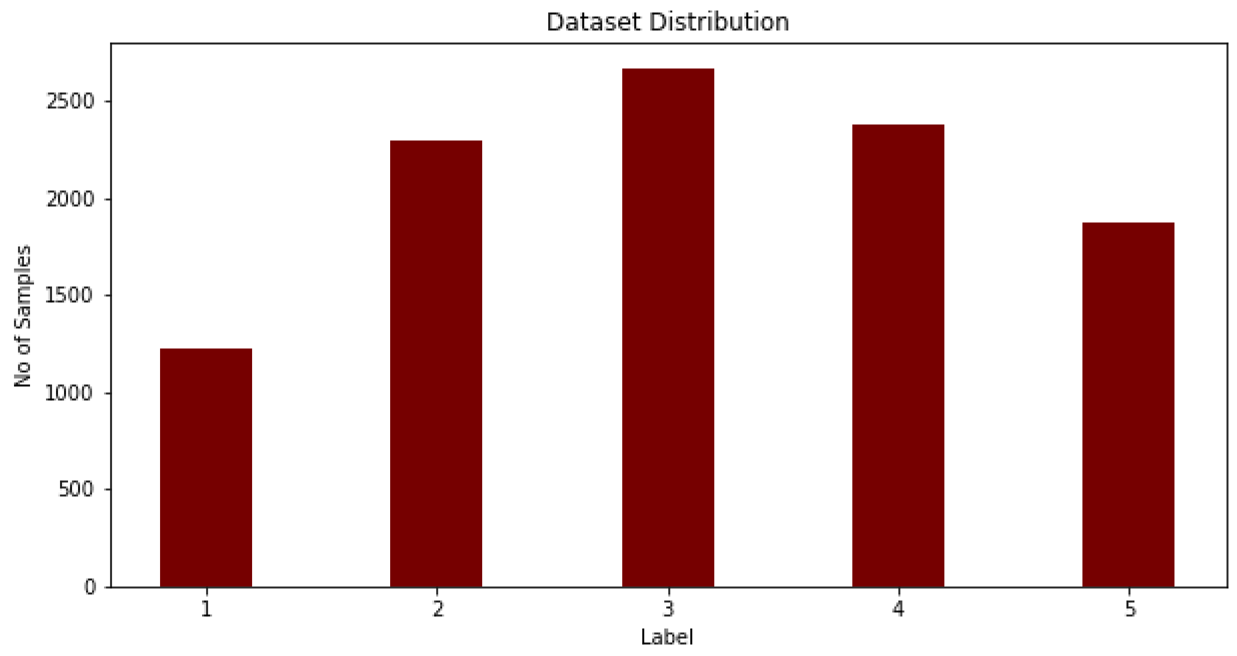
Data exploration:

After performing some data exploration we find that the distribution of the training samples would be:



Label	No of Training Samples used
1	1228
2	2299
3	2666
4	2373
5	1875

We can find a slight imbalance in the number of samples used for the target labels (1, 5). In a future step, the samples for labels (1, 5) should be increased.



<u>Data Pre-Processing:</u>

1. The images are downloaded from the URL. Then every image is converted into bytes.
2. As we need all of the images to be of the same size, All images will be resized to 224x244 pixels using PyTorch transformers.
3. The data loaders are created at a batch size of 32 for training, testing, and validation datasets.

Example of the images used in the training process after being pre-processed

7. Hyperparameter Tuning:

To be able to identify the best hyperparameters to be used in the training process. A hyperparameter tunning phase would be in order.

In this phase, I used the AWS Sagemaker Hyperparameter Tuner tool to perform the training on the model with many combinations of hyperparameters and detect the best-performing combination to be used in the actual training phase.

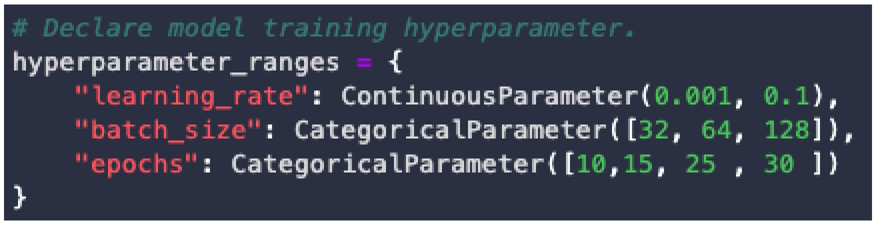
The following hyper-parameters are identified for tunning.

1. Learning Rate - The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated.
2. Batch Size - The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters.
3. Epochs - The number of epochs is a hyperparameter that defines the number of times that the learning algorithm will work through the entire training dataset.

The Learning rate for the Adam optimizer is selected to be optimized as it could help improve the training process rather than the predefined default.

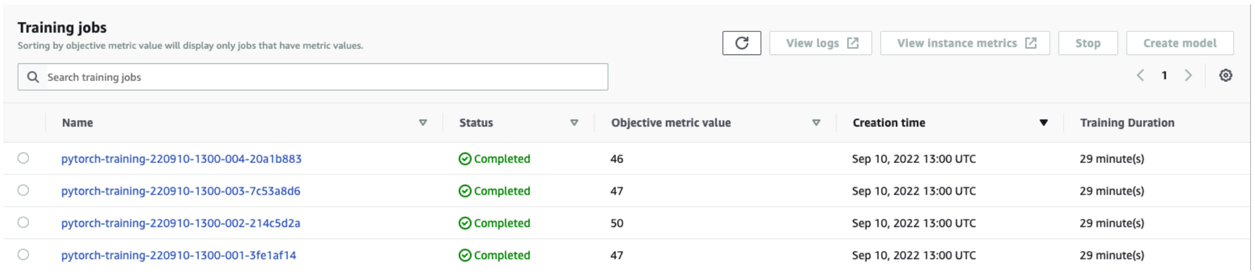
Batch Size was also selected to be optimized as it could help us optimize the computation speed and convergence.

The Number of Epochs could help the model to be better trained over a longer period of time.



```python
# Declare model training hyperparameter.
hyperparameter_ranges = {
    "learning_rate": ContinuousParameter(0.001, 0.1),
    "batch_size": CategoricalParameter([32, 64, 128]),
    "epochs": CategoricalParameter([10,15, 25 , 30 ])
}
```

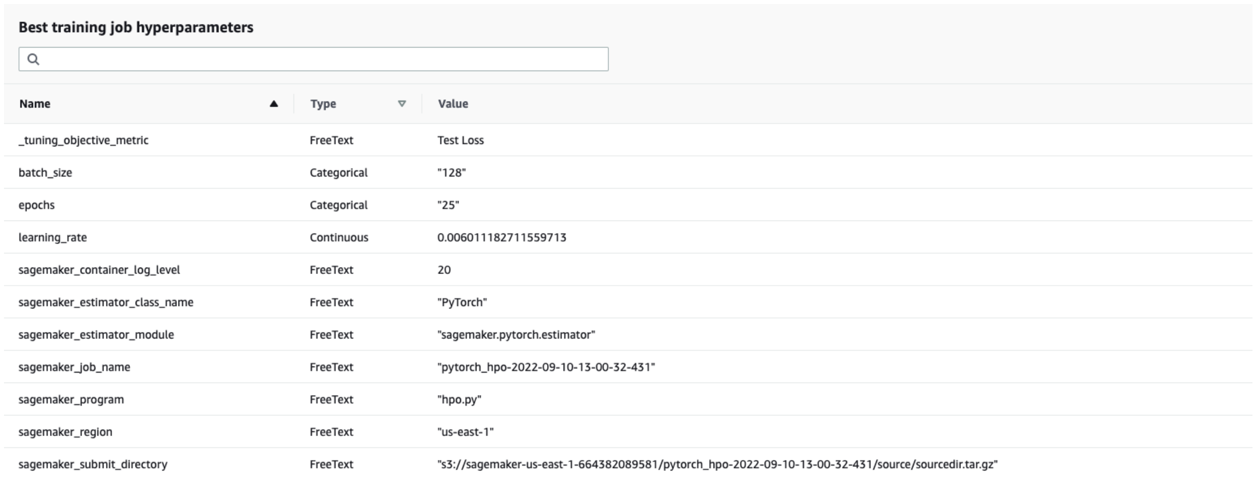
In this phase, 4 training Jobs are created for the tunning and one among them is chosen based on the results ie, an objective metric.



Training jobs

Sorting by objective metric value will display only jobs that have metric values.

	Name	Status	Objective metric value	Creation time	Training Duration
○	pytorch-training-220910-1300-004-20a1b883	✓ Completed	46	Sep 10, 2022 13:00 UTC	29 minute(s)
○	pytorch-training-220910-1300-003-7c53a8d6	✓ Completed	47	Sep 10, 2022 13:00 UTC	29 minute(s)
○	pytorch-training-220910-1300-002-214c5d2a	✓ Completed	50	Sep 10, 2022 13:00 UTC	29 minute(s)
○	pytorch-training-220910-1300-001-3fe1af14	✓ Completed	47	Sep 10, 2022 13:00 UTC	29 minute(s)

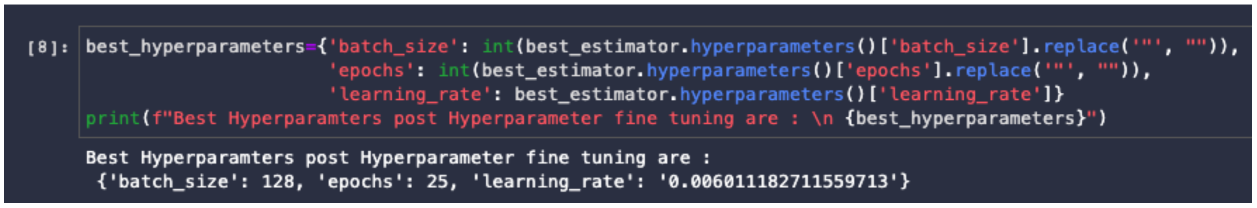


Best training job hyperparameters

Name	Type	Value
_tuning_objective_metric	FreeText	Test Loss
batch_size	Categorical	"128"
epochs	Categorical	"25"
learning_rate	Continuous	0.006011182711559713
sagemaker_container_log_level	FreeText	20
sagemaker_estimator_class_name	FreeText	"PyTorch"
sagemaker_estimator_module	FreeText	"sagemaker.pytorch.estimator"
sagemaker_job_name	FreeText	"pytorch_hpo-2022-09-10-13-00-32-431"
sagemaker_program	FreeText	"hpo.py"
sagemaker_region	FreeText	"us-east-1"
sagemaker_submit_directory	FreeText	"s3://sagemaker-us-east-1-664382089581/pytorch_hpo-2022-09-10-13-00-32-431/source/sourcedir.tar.gz"

8. Model Evaluation:

When the tuning jobs finished, I extracted the best hyperparameters from the tuning job and used them in the actual training phase.



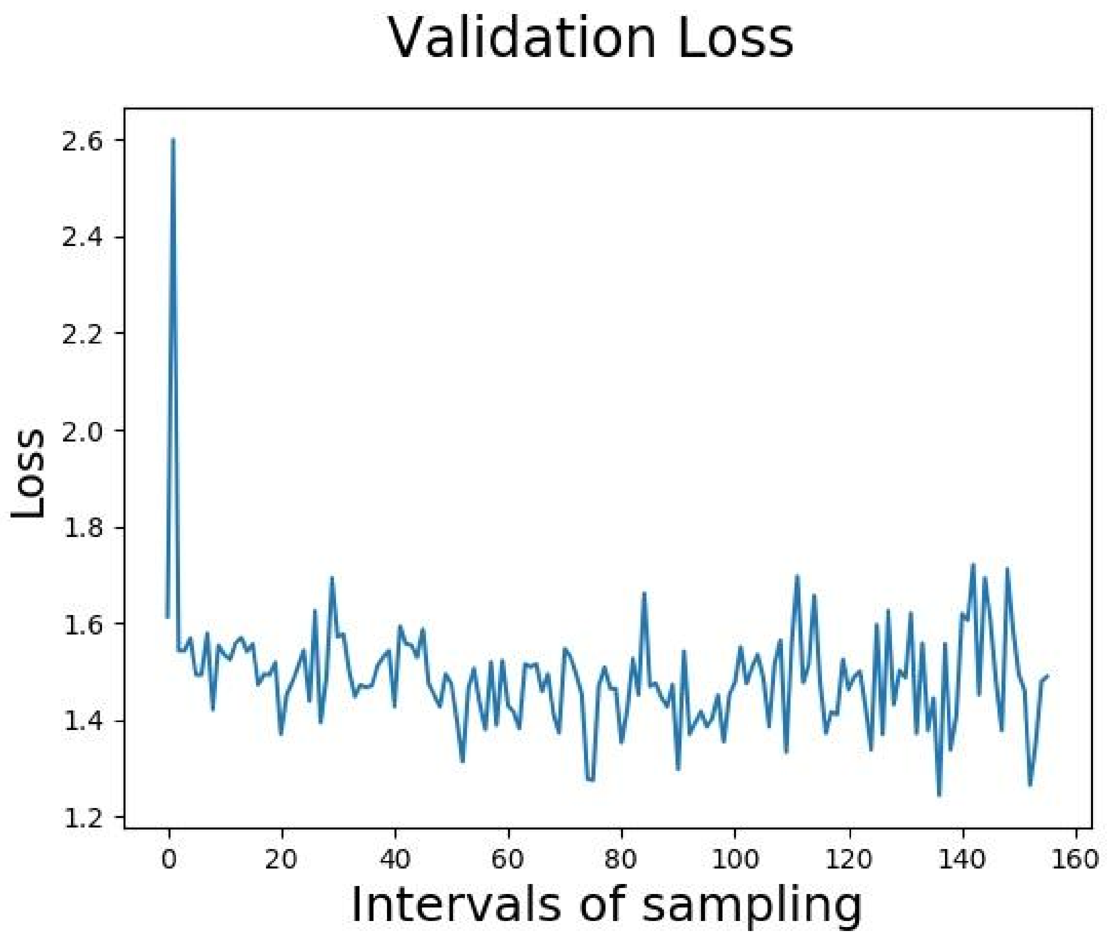
```
[8]: best_hyperparameters={'batch_size': int(best_estimator.hyperparameters()['batch_size'].replace('"', "")),
                          'epochs': int(best_estimator.hyperparameters()['epochs'].replace('"', "")),
                          'learning_rate': best_estimator.hyperparameters()['learning_rate']}
     print(f"Best Hyperparamters post Hyperparameter fine tuning are : \n {best_hyperparameters}")

     Best Hyperparamters post Hyperparameter fine tuning are :
      {'batch_size': 128, 'epochs': 25, 'learning_rate': '0.006011182711559713'}
```

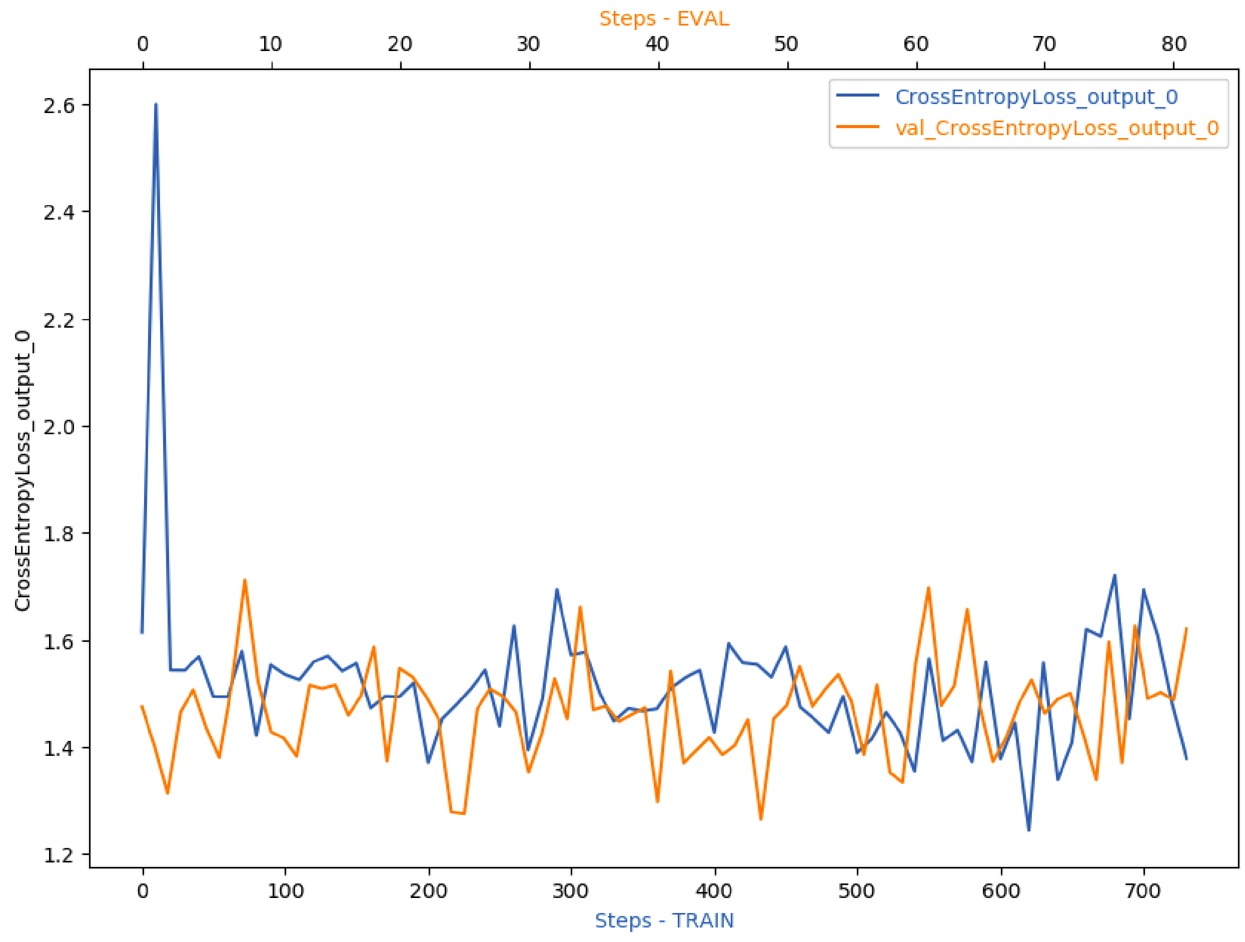
Also, Multi-instance model training had been used to perform a more efficient and time-optimized training phase.

The validation loss of the training process remains almost constant and there are frequent highs and lows, when compared with the training loss, it seems to fit along with it as shown below.

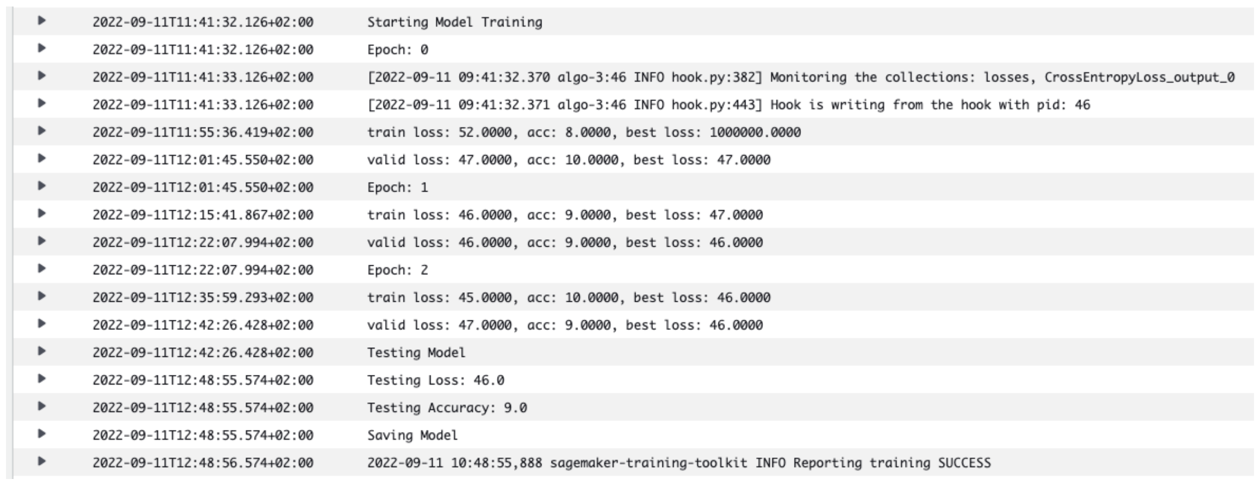
The accuracy of the benchmark model chosen is approximately 56 %, unfortunately, the experiment model didn't achieve the results of the Benchmark. So I think that the training dataset must be increased and training epochs also need to be increased.



Validation Loss



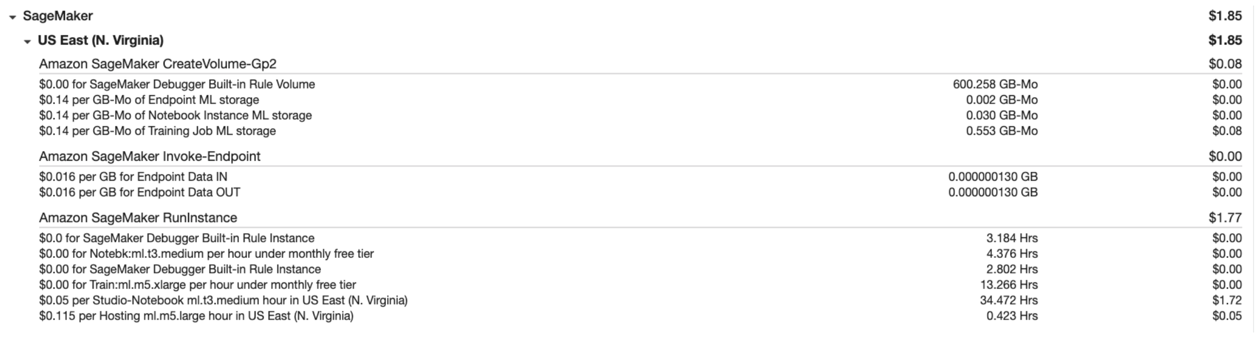
Cross Entropy Loss of Train and Eval



```
▶   2022-09-11T11:41:32.126+02:00    Starting Model Training
▶   2022-09-11T11:41:32.126+02:00    Epoch: 0
▶   2022-09-11T11:41:33.126+02:00    [2022-09-11 09:41:32.370 algo-3:46 INFO hook.py:382] Monitoring the collections: losses, CrossEntropyLoss_output_0
▶   2022-09-11T11:41:33.126+02:00    [2022-09-11 09:41:32.371 algo-3:46 INFO hook.py:443] Hook is writing from the hook with pid: 46
▶   2022-09-11T11:55:36.419+02:00    train loss: 52.0000, acc: 8.0000, best loss: 1000000.0000
▶   2022-09-11T12:01:45.550+02:00    valid loss: 47.0000, acc: 10.0000, best loss: 47.0000
▶   2022-09-11T12:01:45.550+02:00    Epoch: 1
▶   2022-09-11T12:15:41.867+02:00    train loss: 46.0000, acc: 9.0000, best loss: 47.0000
▶   2022-09-11T12:22:07.994+02:00    valid loss: 46.0000, acc: 9.0000, best loss: 46.0000
▶   2022-09-11T12:22:07.994+02:00    Epoch: 2
▶   2022-09-11T12:35:59.293+02:00    train loss: 45.0000, acc: 10.0000, best loss: 46.0000
▶   2022-09-11T12:42:26.428+02:00    valid loss: 47.0000, acc: 9.0000, best loss: 46.0000
▶   2022-09-11T12:42:26.428+02:00    Testing Model
▶   2022-09-11T12:48:55.574+02:00    Testing Loss: 46.0
▶   2022-09-11T12:48:55.574+02:00    Testing Accuracy: 9.0
▶   2022-09-11T12:48:55.574+02:00    Saving Model
▶   2022-09-11T12:48:56.574+02:00    2022-09-11 10:48:55,888 sagemaker-training-toolkit INFO Reporting training SUCCESS
```

9. Cost Analysis:

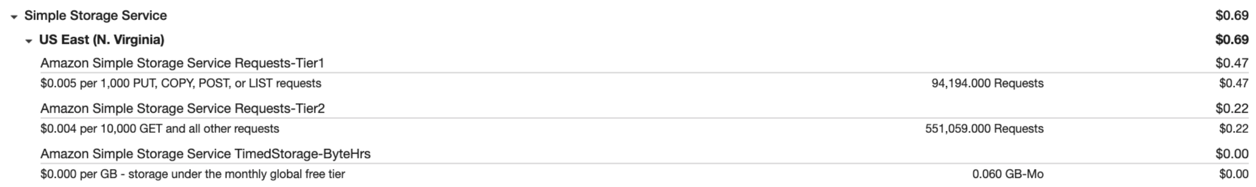
The cost of the training jobs on Sagemaker:



SageMaker		**$1.85**
US East (N. Virginia)		**$1.85**
Amazon SageMaker CreateVolume-Gp2		$0.08
$0.00 for SageMaker Debugger Built-in Rule Volume	600.258 GB-Mo	$0.00
$0.14 per GB-Mo of Endpoint ML storage	0.002 GB-Mo	$0.00
$0.14 per GB-Mo of Notebook Instance ML storage	0.030 GB-Mo	$0.00
$0.14 per GB-Mo of Training Job ML storage	0.553 GB-Mo	$0.08
Amazon SageMaker Invoke-Endpoint		$0.00
$0.016 per GB for Endpoint Data IN	0.000000130 GB	$0.00
$0.016 per GB for Endpoint Data OUT	0.000000130 GB	$0.00
Amazon SageMaker RunInstance		$1.77
$0.0 for SageMaker Debugger Built-in Rule Instance	3.184 Hrs	$0.00
$0.00 for Notebk:ml.t3.medium per hour under monthly free tier	4.376 Hrs	$0.00
$0.00 for SageMaker Debugger Built-in Rule Instance	2.802 Hrs	$0.00
$0.00 for Train:ml.m5.xlarge per hour under monthly free tier	13.266 Hrs	$0.00
$0.05 per Studio-Notebook ml.t3.medium hour in US East (N. Virginia)	34.472 Hrs	$1.72
$0.115 per Hosting ml.m5.large hour in US East (N. Virginia)	0.423 Hrs	$0.05

1.85$ was charged for the hyperparameter tuning and the training jobs. Of course, if a spot instance were used for those jobs the cost would have been significantly lower than the current cost.

The cost of the usage of S3 storage:



Simple Storage Service		**$0.69**
US East (N. Virginia)		**$0.69**
Amazon Simple Storage Service Requests-Tier1		$0.47
$0.005 per 1,000 PUT, COPY, POST, or LIST requests	94,194.000 Requests	$0.47
Amazon Simple Storage Service Requests-Tier2		$0.22
$0.004 per 10,000 GET and all other requests	551,059.000 Requests	$0.22
Amazon Simple Storage Service TimedStorage-ByteHrs		$0.00
$0.000 per GB - storage under the monthly global free tier	0.060 GB-Mo	$0.00

<u>So in total, the whole cost of the project was 2.61$</u>

10. Further Improvements:

1. Increase the training dataset and balance the sample numbers for all of the target labels.

2. Increase the hyperparameters that will be tuned in the hyperparameter tuning job.

3. Implement Data Augmentation Techniques to better transform the data to prevent model overfitting.

4. Try out different combinations of CNN and other models in the stacked architecture.

References:

- Project Benchmark
- Amazon Bin Image Dataset
- Sagemaker -Python SDK for PyTorch
- ResNet Model