AWS Machine Learning Engineer Nanodegree Capstone Project Report.

Inventory Monitoring at Distribution Centers.

17.04.2023

—

Henok Wubet.

Definition

Project Overview

Computer vision research dates back 60 years, with computer scientists continuously exploring novel approaches to enable computers to extract meaningful information from input images.

Computer vision algorithms involve various techniques for acquiring, processing, analyzing, and comprehending digital images to extract data from the real world. This interdisciplinary field aims to enable computers to fully understand digital images, emulating human vision.

In recent years, there has been a surge in the development of deep learning techniques and technologies, especially since 2010. With high-performance computers and GPUs, deep learning models can be trained and improved over time, providing businesses with powerful tools to analyze images.[5]

The retail industry has seen rapid advancement in recent years as automation and machine learning are increasingly integrated into various operations. One such area is optimizing warehouse product storage and retrieval, which is essential for efficient order fulfillment. The Amazon Bin Image Dataset is a valuable resource for exploring machine-learning solutions to object counting and identification within bins.

The research on counting objects in images has significant practical implications across various domains. Convolutional Neural Networks (CNNs) have shown outstanding performance in detecting and counting objects. This project aims to leverage the progress made in this field and use it to count things inside Amazon bins. The ultimate aim is to improve product storage and retrieval accuracy and efficiency in warehouse settings.

Problem Statement

Distribution centers often use robots to move objects as part of their operations. Things are carried in bins that can contain multiple objects. Accurately tracking inventory and ensuring that delivery consignments have the correct number of items is crucial for the smooth operation of the distribution centers. To solve this problem, we propose building a Classifier ML model that counts the number of objects in each bin (items per bin is always between 1 and 5). This project will use the Amazon Bin Image Dataset and AWS SageMaker to preprocess data and train the model. The project will demonstrate end-to-end machine learning engineering skills acquired during the nano degree.

Metrics

The evaluation metric for this problem is the Accuracy Score. Accuracy measures the proportion of correctly predicted object counts to the total number of object counts in the validation set. It is computed as

$$Accuracy = \frac{Number\ of\ Correct\ prediction}{Total\ number\ of\ prediction}$$

$$accuracy(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(\hat{y}_i = y_i)$$

Analysis

Data Exploration

To accomplish this project we will utilize the Amazon Bin Image Dataset, which comprises 500,000 images depicting bins that contain one or multiple objects. Each image in the Dataset is accompanied by a metadata file containing information such as the number of objects, dimensions, and the object type. Our objective is to classify the number of objects present in each bin.[1]

- ❖ This Dataset consists of about 535,000 images in jpg format of Bins alongside corresponding metadata of the items contained in each Image in JSON format.
- ❖ Each image has a unique numerical identifier and the corresponding metadata pair. For example, the image named 1000.jpg will have its metadata named 1000.json
- ❖ The number of items in each bin varies across the bins.
- ❖ The metadata(JSON) file contains information like the associated Image name, expected quantity of items, dimensions, and units of each item.
- ❖ We used only a **tiny subset** of the original Amazon bin image dataset to avoid cloud cost.
- ❖ The dataset has been split into **80% training**, **10% validation**, and **10% test** datasets

Fig 1—sample Images from Amazon Bin Image Datasets

Exploratory Visualization

The Amazon Bin Image Dataset's basic statistics, including the total number of images, the average number of objects per bin, and the total number of object categories, are presented in **Table 1**. With its vast collection of over 500,000 images, a smaller subset of the dataset must be chosen due to budget limitations on Amazon Web Services. **Table 1** showcases the total number of images, the average number of objects per bin, and the total number of object categories found in the dataset.

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

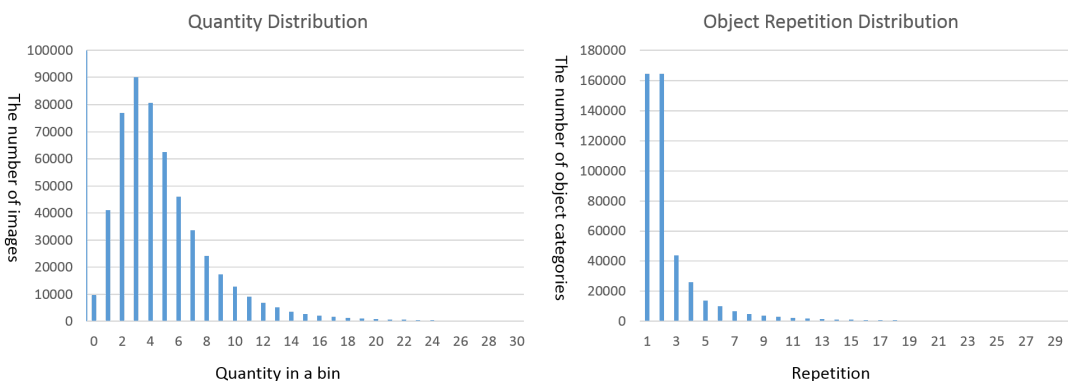Table 1. Amazon Bin Image Dataset Statistics

Fig 2. Amazon bin image dataset stat

The figure on the left displays the frequency distribution of the number of objects in a bin. Specifically, it illustrates that in 90% of the bin images, fewer than ten object instances are present. On the other hand, the figure on the right illustrates the distribution of how often objects are repeated. Out of 459,475 object categories, 164,255 appeared only once in the entire dataset, while 164,356 appeared twice. Additionally, 3038 object categories appeared ten times.

As mentioned in the previous section, We used only a **tiny subset** of the original Amazon bin image dataset to avoid cloud cost. The following figure shows the dataset distribution of the small subset of data we used.
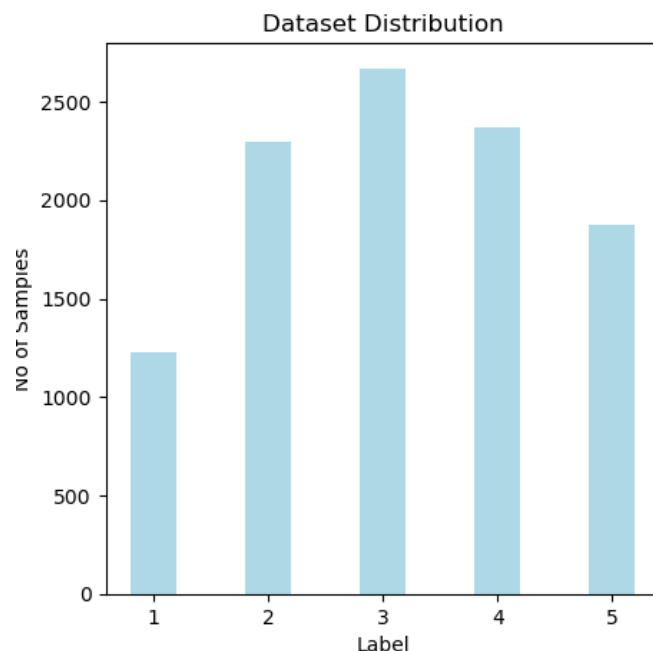

Fig 3. Small subset dataset distribution

Algorithm & Techniques

Convolutional neural networks (CNNs) are better suited to image classification tasks than multi-layer perceptrons (MLPs), which only use fully connected layers. MLPs use lots of parameters, and the computational complexity of the network can become very large. Another disadvantage is that they discard the 2D information of the pixels as they flatten the input image to a 1D vector.

CNNs take advantage of the spatial proximity of a group of pixels by using sparsely connected layers and accepting a 2D matrix as an input. Pixels close to each other are relevant to extracting patterns in the image. Each group of pixels close to each other shares a common group of weights (parameters), the idea being that different regions within the image can share the same information.

The 2D convolution operation is simple: We start with a kernel (or filter), a small matrix of weights the same size as the convolutional window. This kernel slides over the 2D input matrix of image pixels (nodes), performing an elementwise multiplication with the part of the image it is currently on and then summing up the results into a single output node. a collection of such output nodes is called a convolutional layer.

To define a CNN, the following hyperparameters can be tuned:

- **kernel size:** Represents the size of the convolutional window.
- **Number of filters:** Each filter detects a single pattern, so we need to define more filters to detect more patterns.
- **Stride**: An integer specifies the steps that the sliding convolutional window moves.
- **Padding:** Padding the input such that the output has the same length as the original input.

The project's primary objective is to develop a machine-learning model that can accurately count the number of objects in each bin in distribution centers.[3]

The solution to this problem involves training a deep learning model using the **ResNet-50** architecture to predict the object counts in an image from 1 - 5. Using transfer learning with pre-trained Torch models, the ResNet-50 model will be fine-tuned on the given dataset. We trained the model and tuned the hyperparameters using Amazon Sagemaker. The model was evaluated on the validation set using **Accuracy** metrics. The final model has been deployed and tested on a test set to evaluate its performance.

We chose transfer learning because it has been shown to improve the performance of models on image classification tasks, especially in scenarios with limited training data. By starting with a pre-trained model, we can utilize its learned features and modify them to suit our particular problem domain. This approach can minimize training time and enhance accuracy.

We will initiate a hyperparameter tuning task using Amazon SageMaker Hyperparameters Tuner to enhance our model's performance. We will tune the hyperparameters for learning rate (LR) and batch size (batch_size) to identify the optimal combination with the highest accuracy. We will employ the best hyperparameters obtained to train our model upon completing the tuning task.

By using these techniques and algorithms, we aim to improve the accuracy of our object-counting solution.

BenchMark

In this project, we will use an open-access model as the benchmark to compare the performance of our proposed solution. The benchmark model, which can be downloaded from here, is a deep convolutional classification network based on the ResNet-34 architecture. It is trained from scratch on the Amazon Bin Image Dataset to categorize each image into one of six categories (0-5) based on the number of objects in the bin. The training script has a batch size of 128 and runs for 40 epochs, with the learning rate decaying by a factor of 0.1 every ten epochs.

The model attained its highest validation accuracy of 55.67% and an RMSE of 0.930 at 21 epochs during training. However, it began to overfit after 21 epochs. The metadata format used for the training and validation sets is a list of [image IDX, count] pairs.

We will compare our proposed solution's accuracy with the benchmark model to evaluate the effectiveness of our approach.

Methodology

Data Preprocessing

Data preprocessing is critical to any machine learning project, mainly when dealing with large and complex datasets. In this project, we encounter a massive dataset and must extract a representative subset of data for our task.

Our code downloads the training data and organizes it into subfolders, each containing images corresponding to the number of objects represented by the folder name. We only consider data with labels from 1 - 5 to ensure a fair comparison with our benchmark solution (Fig 3).

For baseline methods, we resized all images into **224x224** for convenient training.

Afterward, we partition the extracted dataset into **train**, **test**, and **validation** sets, allocating 10% for testing, 10% for validation, and 80% for training. We then migrate the images for each partition from the source bucket to our own bucket, creating subfolders with label-based names corresponding to the number of objects in the image.
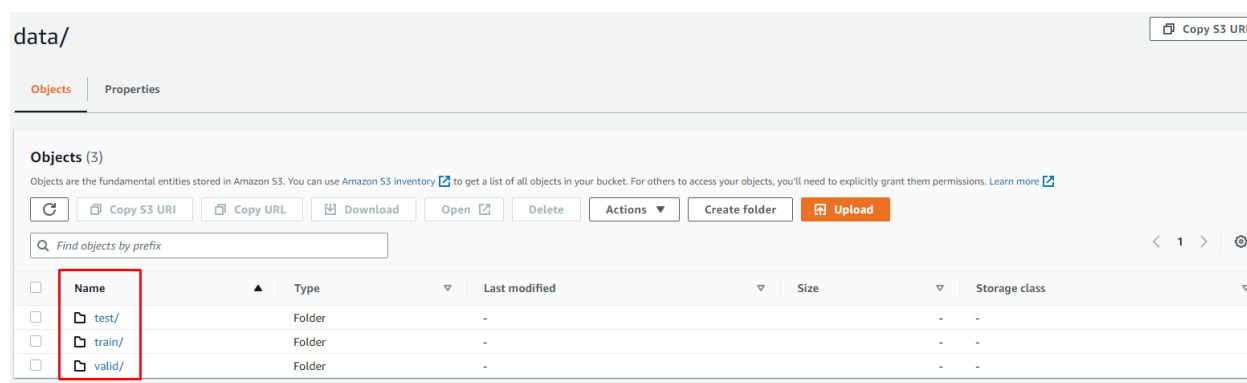
Fig 4. A split dataset in s3.

Implementation

In this project, we utilize AWS Sagemaker to train an already pre-trained Resnet50 model capable of image classification on the given dataset for dog breed classification. Additionally, we incorporate Sagemaker profiling, debugger, hyperparameter tuning, and other commendable practices in machine learning engineering.

This project was developed and tested on AWS SageMaker. It was created from a starter file provided by Udacity here. And The dataset provided is the dog breed classification dataset, accessible through this link.

A. Hyperparameter Tuning

We utilize the pre-trained **ResNet50** model from PyTorch, a convolutional neural network, to enable transfer learning. We then fine-tune this model using transfer learning techniques to classify the number of objects in images.

In this project, we focus on tuning two key parameters - the **learning_rate** and **batch_size** - as these impact both the model's accuracy and speed of conversion. The learning_rate falls within 0.001 to 0.1, while the batch_size can take on one of five values (32, 64, 128, 256, or 512).

To achieve the best results, we execute a hyperparameter tuning job that selects parameters from the search space, runs a training job, and then makes predictions. The primary objective of this process is to improve the Test Loss metric.

Ultimately, the optimal training hyperparameters will minimize the Test Loss metric.
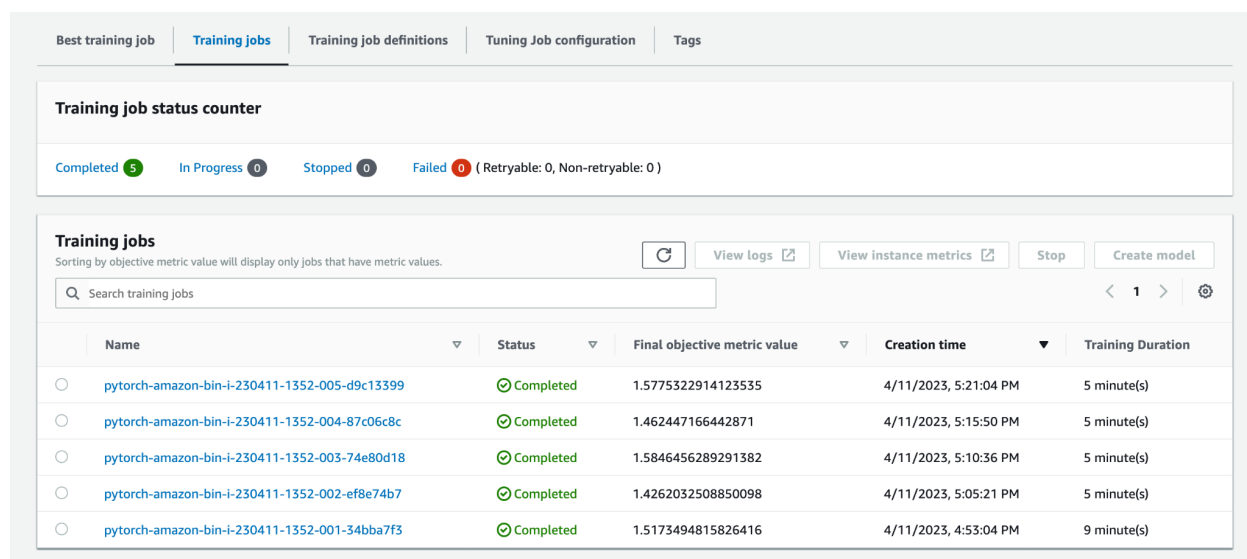
Best training job	**Training jobs**	Training job definitions	Tuning Job configuration	Tags	

Training job status counter

Completed **5**   In Progress **0**   Stopped **0**   Failed **0** (Retryable: 0, Non-retryable: 0)

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

Search training jobs

	Name	Status	Final objective metric value	Creation time	Training Duration
○	pytorch-amazon-bin-i-230411-1352-005-d9c13399	⊘ Completed	1.5775322914123535	4/11/2023, 5:21:04 PM	5 minute(s)
○	pytorch-amazon-bin-i-230411-1352-004-87c06c8c	⊘ Completed	1.462447166442871	4/11/2023, 5:15:50 PM	5 minute(s)
○	pytorch-amazon-bin-i-230411-1352-003-74e80d18	⊘ Completed	1.5846456289291382	4/11/2023, 5:10:36 PM	5 minute(s)
○	pytorch-amazon-bin-i-230411-1352-002-ef8e74b7	⊘ Completed	1.4262032508850098	4/11/2023, 5:05:21 PM	5 minute(s)
○	pytorch-amazon-bin-i-230411-1352-001-34bba7f3	⊘ Completed	1.5173494815826416	4/11/2023, 4:53:04 PM	9 minute(s)

Fig 5. Completed hyperparameter tuning job

Best training job	Training jobs	Training job definitions	Tuning Job configuration	Tags	

Best training job summary
This training job is the best training job for only this hyperparameter tuning job.

Create model

Name	Status	Objective metric	Value
pytorch-amazon-bin-i-230411-1352-002-ef8e74b7	⊘ Completed	Test Loss	1.4262032508850098

Best training job hyperparameters

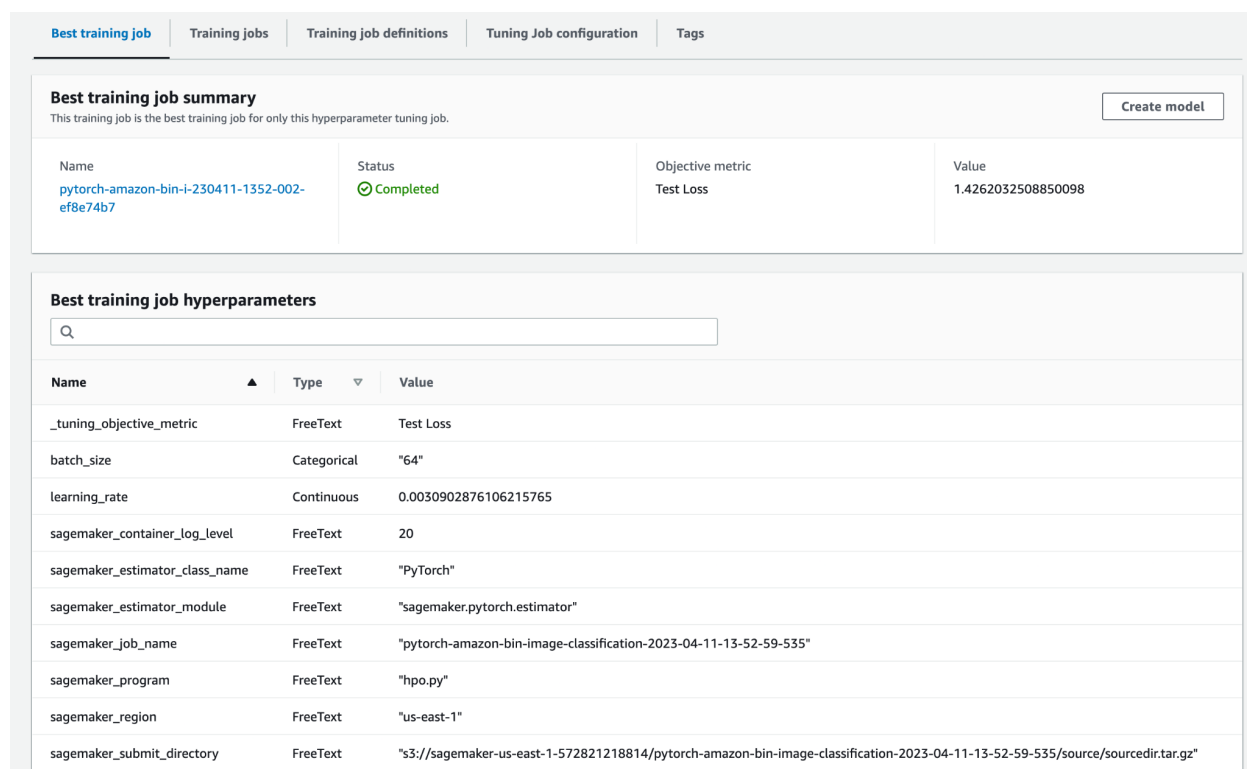| Name ▲	Type	Value
_tuning_objective_metric	FreeText	Test Loss
batch_size	Categorical	"64"
learning_rate	Continuous	0.0030902876106215765
sagemaker_container_log_level	FreeText	20
sagemaker_estimator_class_name	FreeText	"PyTorch"
sagemaker_estimator_module	FreeText	"sagemaker.pytorch.estimator"
sagemaker_job_name	FreeText	"pytorch-amazon-bin-image-classification-2023-04-11-13-52-59-535"
sagemaker_program	FreeText	"hpo.py"
sagemaker_region	FreeText	"us-east-1"
sagemaker_submit_directory	FreeText	"s3://sagemaker-us-east-1-572821218814/pytorch-amazon-bin-image-classification-2023-04-11-13-52-59-535/source/sourcedir.tar.gz"

Fig 6. Summary of the best training job

Upon tuning, we found the best hyperparameters to be:

```
Batch Size: 64
Learning rate: 0.0030902876106215765
```

We employed these hyperparameters to train our model, utilizing transfer learning of a pre-trained **ResNet50** model from PyTorch to accelerate training and enhance accuracy. We froze the parameters of all layers except for the last ones and mapped the output to 5 classes to tackle our object counting problem from 1-5.

```python
def net():
    '''
    TODO: Complete this function that initializes your model
          Remember to use a pre-trained model
    '''
    model = models.resnet50(pretrained=True)

    for param in model.parameters():
        param.requires_grad = False

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

B. Debugging and Profiling

We employed the SMDebug client library from Amazon SageMaker to facilitate model debugging and profiling. The Sagemaker debugger allows us to monitor our machine learning model's training performance, record training and evaluation metrics, and plot learning curves. Additionally, it can detect potential problems such as overfitting, overtraining, poor weight initialization, and vanishing gradients.

Underfitting occurs when the validation score does not improve over time, suggesting that the model is not learning enough from the data. On the other hand, overfitting refers to a situation in which the training curve keeps improving while the validation curve worsens. Both of these issues can be addressed by tuning the hyperparameters or by collecting more data samples.
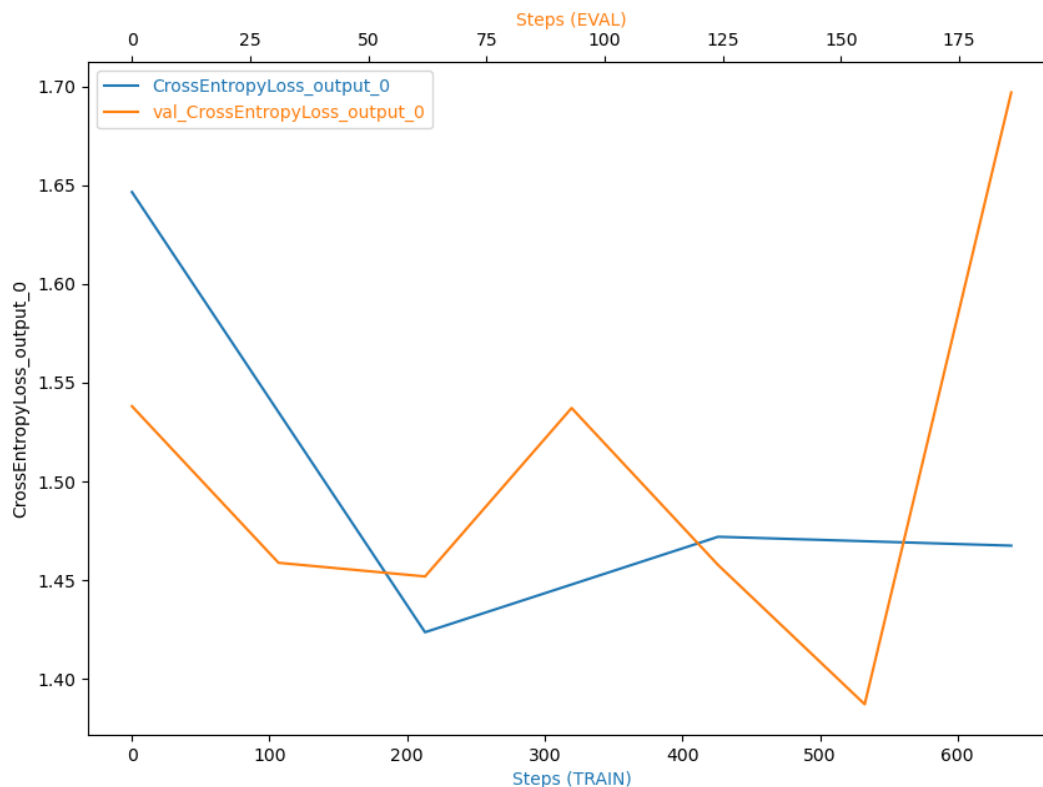
Fig 7. Debugger & Profiler Output

Multi-Instance Training

By modifying the parameter `instance_count=4`, we created a multi-instance training job to train the model on four instances simultaneously.

```python
# TODO: Train your model on Multiple Instances
###in this cell, create and fit an estimator using multi-instance training
estimator_multi_instance = PyTorch(
    source_dir="scripts",
    entry_point="train.py",
    base_job_name='pytorch-multi-instance-amazon-multi-instance-bin-image-classification',
    role=ROLE,
    instance_count=4,
    instance_type='ml.m5.2xlarge',
    framework_version='1.9',
    py_version='py38',
    hyperparameters=hyperparameters,
    output_path=f"s3://{BUCKET}/output_multi_instance/",
    ## Debugger and Profiler parameters
    rules = rules,
    debugger_hook_config=debugger_config,
    profiler_config=profiler_config,
)
```

EC2 Training

To minimize costs, one can consider using a special type of EC2 instance known as a Spot Instance. AWS has a vast supply of computing resources, made available to users upon request. Instances are reserved for individual users, but sometimes, users reserve instances and leave them idle. AWS makes these idle resources temporarily available to others, known as "**Spot Instances.**"

Spot Instances can be started and used at any time. However, since a Spot Instance is a computing resource that someone else has reserved, it might be turned off at any time, such as when the original user decides to use or turn off the resource. Therefore, using or creating a Spot Instance involves a risk of the instance being turned off at any moment.

We used `t3.2xlarge` instances, low-cost burstable general-purpose instance types that provide a baseline level of CPU performance with the ability to burst CPU usage anytime for as long as required.

As for the training image, We have used Deep Learning `AMI GPU PyTorch 1.13.1 (Ubuntu 20.04) 20230326` to train the model.
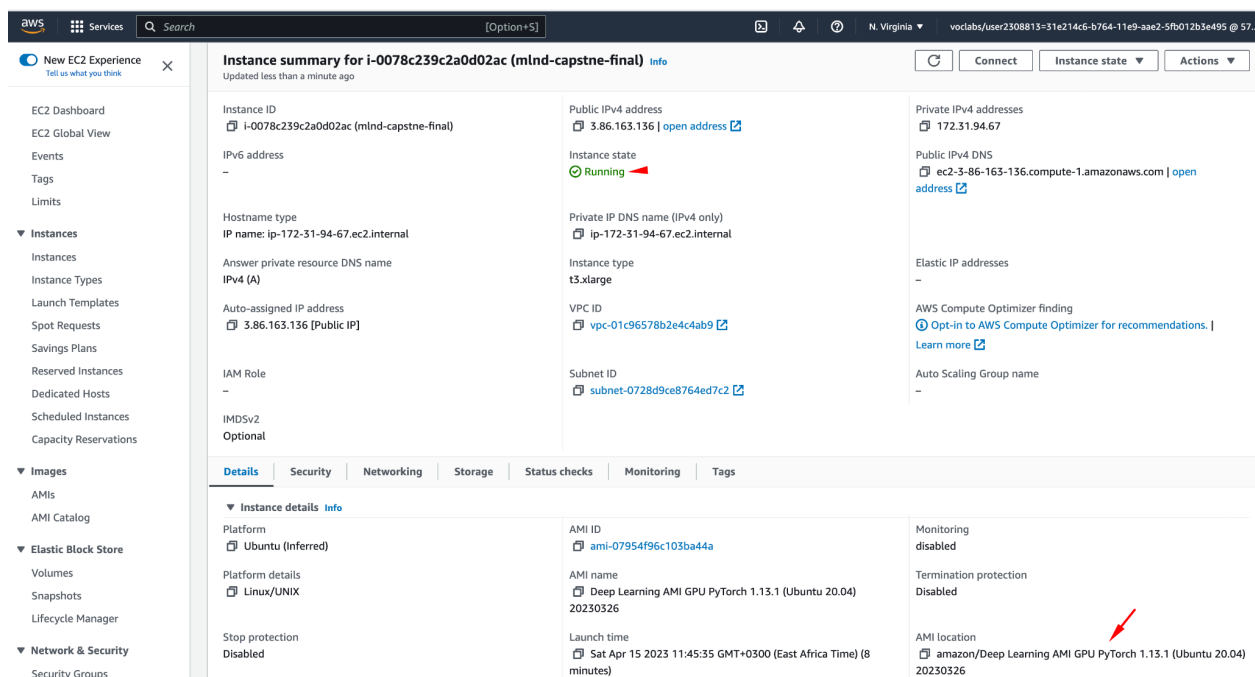
Fig 8. EC2 Instance summary

After connecting to the EC2 instance, We downloaded the data & created the model output directory:

```
wget https://mlnd-capstone-inventory-monitoring.s3.amazonaws.com/EC2/amazon_bin_image.zip
unzip dogImages.zip
mkdir TrainedModels
```

Created a blank Python file by running the following command in your EC2 Terminal:

```
vim solution.py
```

Paste all of the code from **/scripts/ec2train.py** into **solution.py** and Activated the PyTorch environment that we will use to train our model:

```
Source activate pytorch
```

And Run the code.

```
python solution.py
```


Fig 9. EC2 Training

There are significant differences between the EC2 training and Sagemaker approaches. When the /scripts/ec2train.py file is executed directly, the training occurs locally on the same computing machine. On the other hand, in the Sagemaker notebook, a new training job instance is created, and all the training parameters are passed to it. The hyperparameters in the Sagemaker starter script, `hpo.py`, are explicitly passed to the script and recorded in the instance environment variables. In Sagemaker, the output trained `model.pth` is saved to the training job compute instance in `SM_MODEL_DIR=/opt/ml/model,` compressed with the source code `hpo.py`, and automatically transferred to an output directory in the S3 bucket. In contrast, during EC2 training, the model is saved locally inside the `./TrainedModels` directory, and the user is responsible for transferring the model to S3 using the aws s3 cp command. Another difference is that Sagemaker can use the training

script to spawn multiple instances and perform distributed training. In contrast, an EC2 instance is limited to running the script on a single instance.
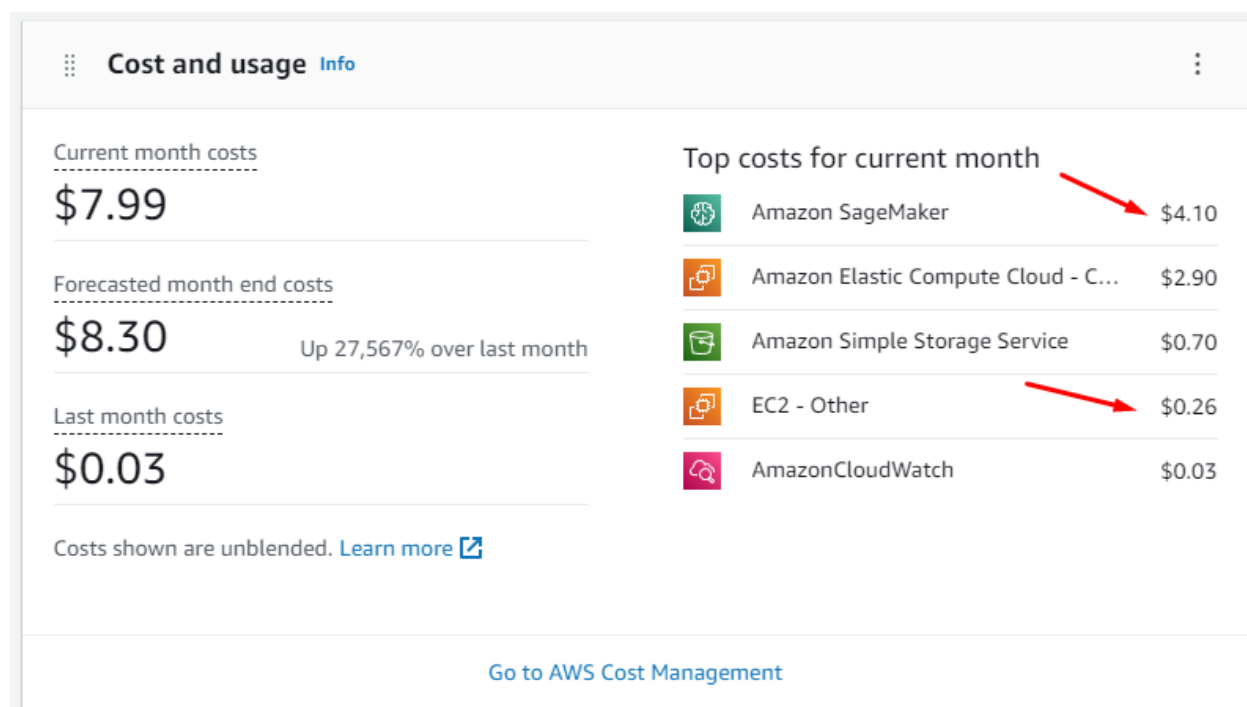

Fig 10. Comparing the Cost of Training on Sagemaker vs. EC2 Instances.

Results

Model Evaluation & Validation

To enable inference, the model was deployed to a Sagemaker endpoint using an `ml.m5.large` instance, and The inference script is designed to accept an image URL as input.
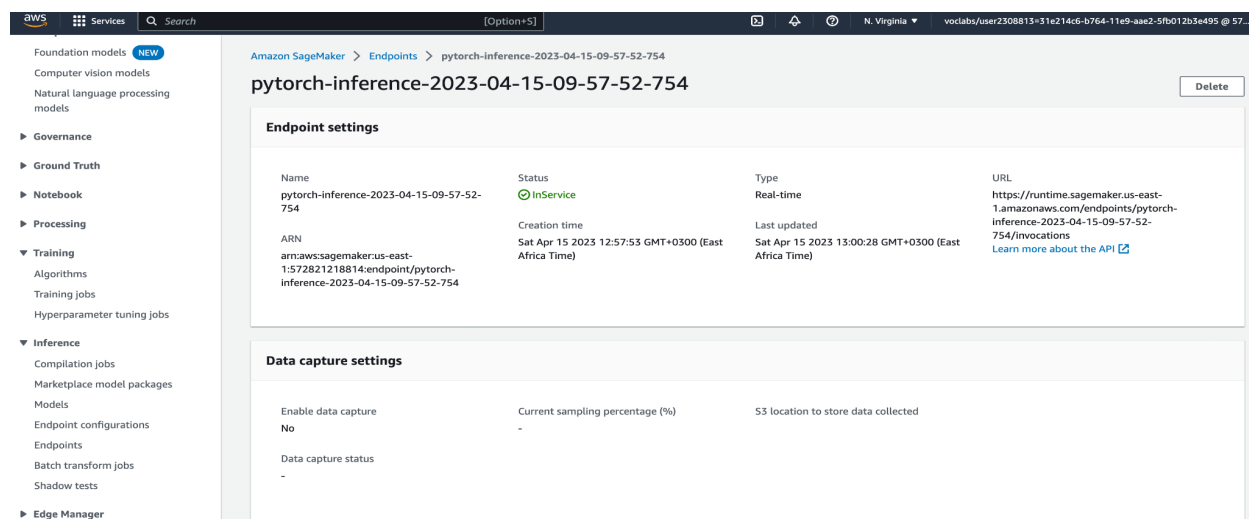
Fig 11. Deployed Endpoint

As we stated in section 1.3. The evaluation metric for this problem is the Accuracy Score. And it can be calculated with the following mathematical formula.

$$Accuracy = \frac{Number\ of\ Correct\ prediction}{Total\ number\ of\ prediction}$$

$$accuracy(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(\hat{y}_i = y_i)$$

Final Evaluation Result Obtained:

METRIC	VALUE
Accuracy (%)	31.5 %

Furthermore, to test the functionality of our trained model, We then queried an image from the endpoint and found that the endpoint returned the correct object counting result for the queried image.

```
[17]:  predict("./splited_dataset/test/2/00558.jpg")
```

Response: [0.36589158 1.07819462 0.58115435 0.15872678 -0.74548155]
Number of objects in the Bin: 2
Probability: 0.37608065466256047

Fig 12. Queried endpoint result.

Justification

This section compares the proposed solution's performance to the benchmark model. The benchmark model was trained on the complete dataset, while the proposed model was trained on only 10% of the dataset using a pre-trained model. The benchmark model achieved an accuracy of 55.67%, while our proposed solution achieved an accuracy of 31.5%.

METRIC	Benchmark result	Solution result
Accuracy (%)	55.67%	31.5 %

From the results, it is evident that the benchmark model surpasses our proposed solution in accuracy. Nonetheless, it is crucial to acknowledge that the main aim of this project is to showcase the application of Amazon SageMaker and not to create the most accurate machine learning model. Thus, the outcomes presented in this project may not reflect the optimum performance attainable with the dataset and the selected model.

Conclusion

We followed a standard machine learning workflow in this project, starting with exploring and preprocessing the data, then building and training a model, and finally evaluating its performance. Working with the AWS SageMaker platform was one of the most interesting aspects of this project, as it provided several powerful tools and services that made the process much more efficient and streamlined.

We developed the final model and solution for this project and are satisfied with the overall outcome. The model achieved a reasonable level of accuracy on the test set, and we successfully deployed it using AWS SageMaker hosting services. However, we acknowledge that there is always room for improvement, and we identified several potential areas for future work.

Reflection

- ➔ The entire end-to-end problem solution can be summarized as the following:
- ➔ A challenge problem and an available public dataset were found.
- ➔ A suitable metric was found and implemented.
- ➔ The data was downloaded and split into training, validation, and testing sets.
- ➔ The data was prepared and preprocessed to be used as input for the classification model.
- ➔ A simple convolutional neural network benchmark model was implemented and tested.
- ➔ Transfer-learning solution models ResNet50 were implemented
- ➔ The solution model classifier was tested on unseen images.

An interesting aspect of this project was that transfer-learning models achieved great performance in a short training time. Their performance was better than the benchmark model, which took much longer to train.

Convolutional neural networks, in general, and transfer-learning techniques, in particular, are the best for image classification problems (up to date), as we have seen in this project implementation. They are highly recommended for such problems and similar problems.

Improvement (Future Work)

For the improvement of our solution model:

→ Allowing the model to train for longer times on more powerful hardware with capable GPUs. That would speed up experimenting with and testing different solution models in shorter times.

→ Another possible improvement is to use the whole Amazon bin image dataset, training the model on a larger count of images and augmenting the available data to increase the count of images per class.

→ Tune more hyperparameters

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

Reference

[1] https://github.com/awslabs/open-data-docs/tree/main/docs/aft-vbi-pds

[2] https://github.com/silverbottlep/abid_challenge/tree/master

[3]https://towardsdatascience.com/comprehensive-guide-on-multiclass-classification-metrics-af94cfb83fbd

[4] https://www.fast.ai/

[5] Pablo Rodriguez Bertorello, Sravan Sripada, & Nutchapol Dendumrongsup. "Amazon Inventory Reconciliation Using AI." https://github.com/pablo-tech/AI-Inventory-Reconciliation/blob/master/ProjectReport.pdf