AWS Machine Learning Engineer Nanodegree Program

Capstone Project

Inventory Monitoring at Distribution Centers

Capstone Proposal review link: https://review.udacity.com/#!/reviews/3331836

I have organized my report to address the five major project development stages:

a) Define the problem and investigate potential solutions and performance metrics
b) Analyze the problem through visualizations and data exploration
c) Implement algorithms and metrics
d) Collect results about the performance
e) Construct conclusions

Define the problem

Domain Background

Distribution centers often use robots to move objects as a part of their operations. Objects are carried in bins which can contain multiple objects. Occasionally, the robots may pick the wrong item or the wrong quantity. This causes customer dissatisfaction and it is difficult to track inventory. Due to the large volume of orders, it is time consuming for workers to double-check each order manually.

This is considered an image classification problem as the model should be able to detect different items in each bin. Below, I will provide a brief history of how modern computer vision, powered by Convolutional Neural Networks (CNNs), came to be.

Back in 1959, two neurophysiologists – David Hubel and Torsten Wiesel – proposed that there are simple and complex neurons in the primary visual cortex. Both types of cells are used in pattern recognition. Later in 1962, they proposed that simple detectors can be summed to create more complex detectors. This forms the basis of CNNs. In the 1980s, Dr. Kunihiko Fukushima proposed using simple (first layer) and complex (second layer) mathematical operations for visual pattern recognition.

Modern CNNs were invented in 1990s when Yann LeCun demonstrated using a CNN model for handwriting recognition. This model was trained using the MNIST dataset. More recently in 2012, a CNN called AlexNet achieved high accuracy labelling pictures in ImageNet. Thereafter, various CNNs achieved better accuracy with different datasets such as CIFAR-10, VisualGenome and medical images (such as chest x-rays).

Problem Statement

This is an image classification problem. The model should count every object instance in the bin. If there are two same objects in the bin, the model should count them as two.

Solution Statement

CNNs are commonly used for image classification tasks. I will fine tune a pre-trained CNN, such as ResNet, to count the number of items in each bin.

The success of my model will be measured by how it correctly predicts the testing dataset. Accuracy = number of correct predictions / number of predictions

Benchmark Model

I will fine tune a ResNet model for this project. ResNet models has been pre-trained on millions of images. This helps them learn general features that can come in useful for a variety of datasets.

Firstly, I will freeze all the convolutional layers and add my own fully connected layer. Next, I will train the model using my training dataset. This entails running the forward pass for the whole network and perform the backward pass only for the fully connected layer.

Since I am only training the fully connected layer, training can be done quickly and easily. However, I am using the whole network when performing prediction, so I can get the improved accuracy of using a CNN.

Evaluation Metrics

I will use accuracy to evaluate my model.

For each image, my model will output the probability of each class. I will take the largest probability as the predicted class. Accuracy = number of correct predictions / total number of predictions

If my model is working well, I should see that accuracy increases with more epochs.

Project Design

I will execute my project in the following steps:

Step 1: Create training and testing datasets

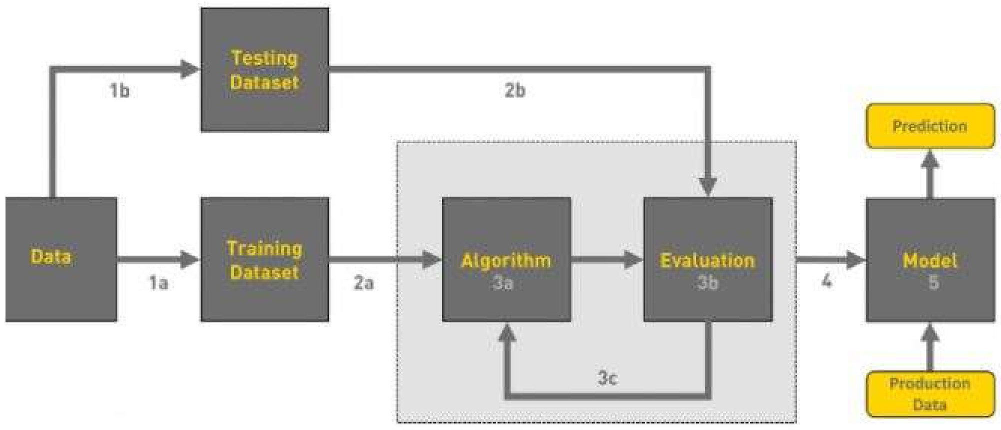
Step 2: Use training dataset to train the model, use testing dataset to evaluate the model

Step 3: Execute multiple epochs to train the model and evaluate the results

Step 4: Deploy the trained model to an endpoint

Step 5: Use the endpoint to infer an image

This workflow is depicted in the diagram below



Overview of the Workflow of ML

Analyze the problem

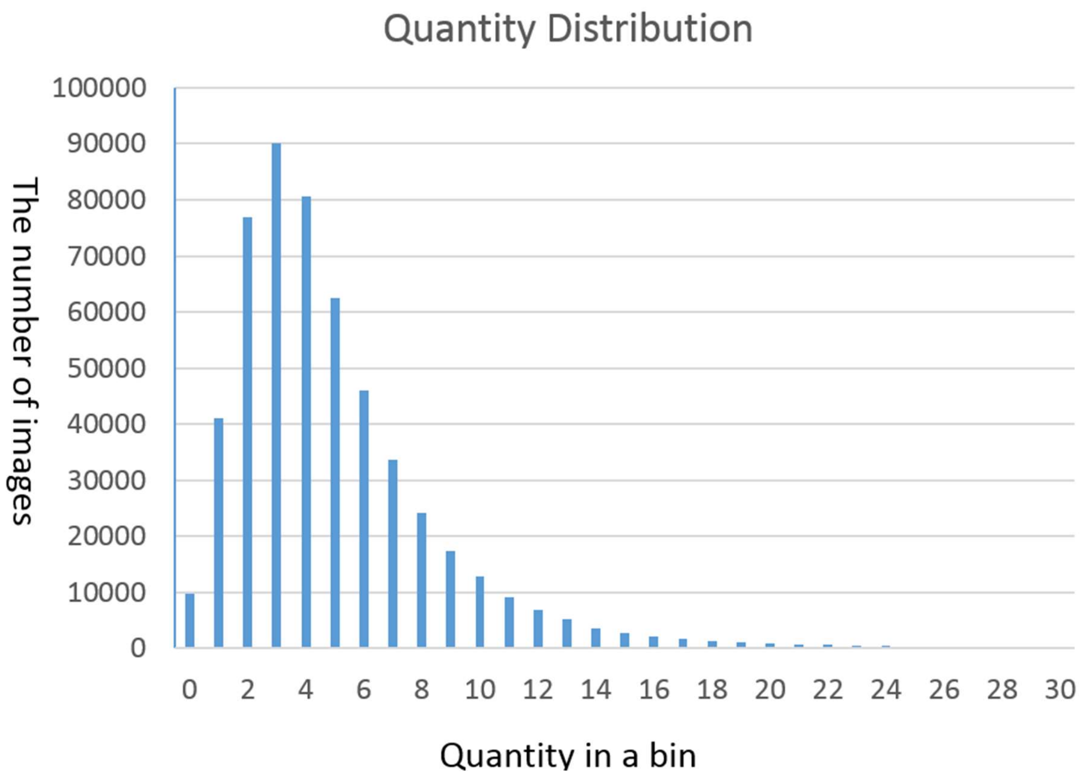
Dataset and Input

I will be using the Amazon Bin Image Dataset which contains 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 carry pods as part of normal Amazon Fulfillment Center operations.

The dataset contains more than 500,000 images of bins containing one or more objects. For each image there is a metadata file containing information about the image like the number of objects, it's dimension and the type of object. For this task, I will try to classify the number of objects in each bin.

Data Exploration

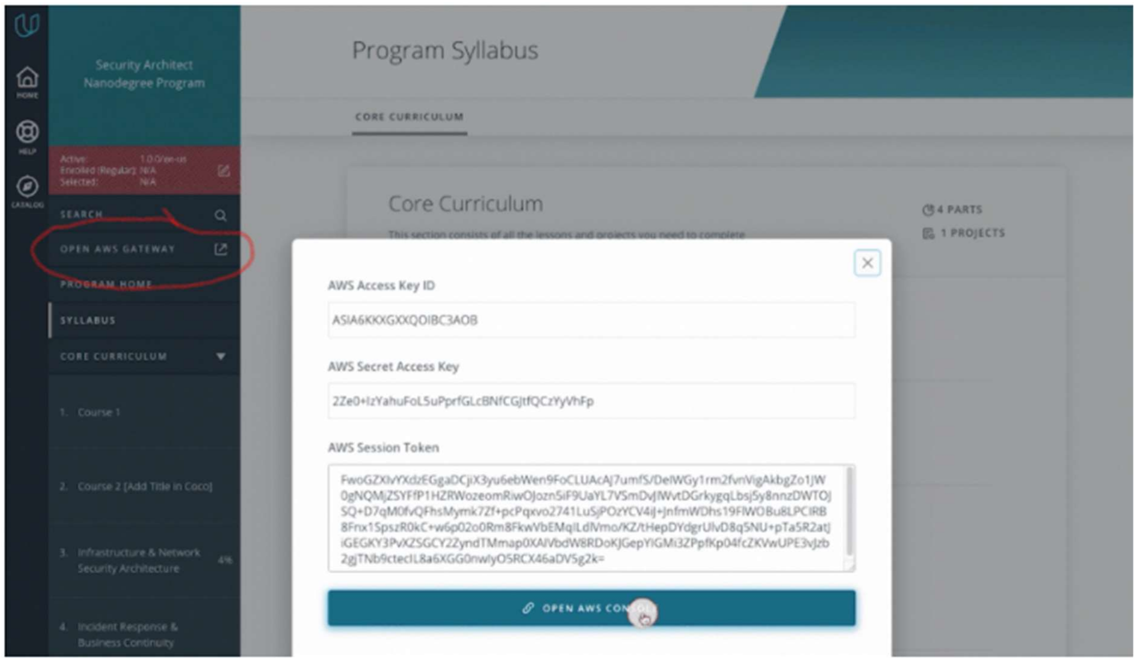
The histogram below shows the distribution of quantity in a bin. More than 90% of the bin images contain less than 10 items. In this project, I will restrict my dataset to images with 5 or less items. To avoid exceeding the course budget, I will only use a subset of images for training and validation purposes.



Implement algorithms and metrics

Project Setup

Step 1: Enter AWS through the gateway in the course and open SageMaker.



AWS Access Key ID

ASIA6KKXGXXQOIBC3AOB

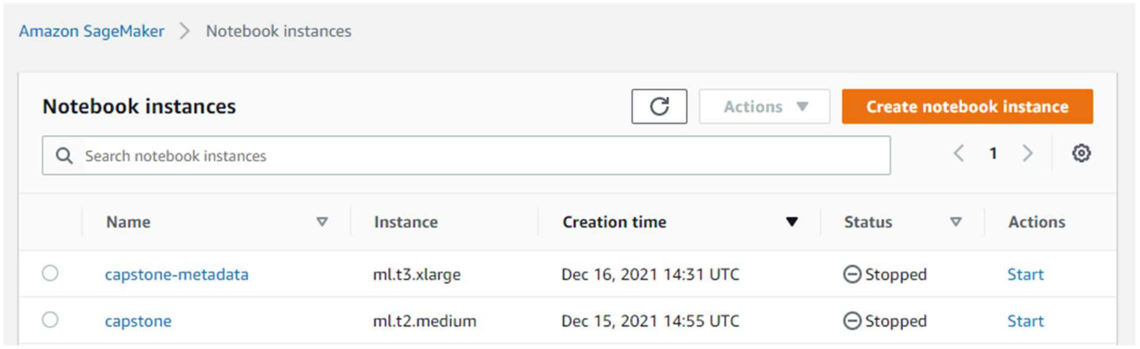
AWS Secret Access Key

2Ze0+IzYahuFoL5uPprfGLcBNfCGJtfQCzYyVhFp

AWS Session Token

```
FwoGZXIvYXdzEGgaDCjiX3yu6ebWen9FoCLUAcAJ7umfS/DeIWGy1rm2fvnVigAkbgZo1JW
0gNQMjZSYFfP1HZRWozeomRiwOJozn5iF9UaYL7VSmDvJIWvtDGrkygqLbsj5y8nnzDWTOJ
SQ+D7qM0fvQFhsMymk7Zf+pcPqxvo2741LuSjPOzYCV4iJ+JnfmWDhs19FlWOBu8LPCIRB
8Fnx1SpszR0kC+w6p02o0Rm8FkwVbEMqILdlVmo/KZ/tHepDYdgrUlvD8q5NU+pTa5R2atJ
iGEGKY3PvXZSGCY2ZyndTMmap0XAlVbdW8RDoKJGepYIGMi3ZPpfKp04fcZKVwUPE3vJzb
2gjTNb9ctecIL8a6XGG0nwlyO5RCX46aDV5g2k=
```

Step 2: Create a Notebook instance


Amazon SageMaker > Notebook instances
Notebook instances
Actions
Create notebook instance
Search notebook instances

Name	Instance	Creation time	Status	Actions
capstone-metadata	ml.t3.xlarge	Dec 16, 2021 14:31 UTC	Stopped	Start
capstone	ml.t2.medium	Dec 15, 2021 14:55 UTC	Stopped	Start

Step 3: Download the starter files from https://github.com/udacity/nd009t-capstone-starter

Step 4: Upload the starter files into the workspace

Data Preparation

As part of the starter files, "file_list.json" contains a dictionary with labels as keys and list of image paths as values. There is a function provided to read this dictionary and download images into subfolders, where each subfolder corresponding to the image quantity. For example, subfolder "1" contains images with one item.

I wanted to create similar files as "file_list.json" for testing and validation datasets. This is challenging as I do not wish to download all the 500,000 metadata files. Reading through the Amazon Bin Image Dataset documentation, I found a link to a project where all the image metadata has been consolidated into a single file.

With this metadata file, I picked 1500 sample images for testing and validation. After filtering out images with more than 5 items, I stored the remaining images in "file_list_test.json" and "file_list_val.json". These 2 files are written in the same format as "file_list.json"

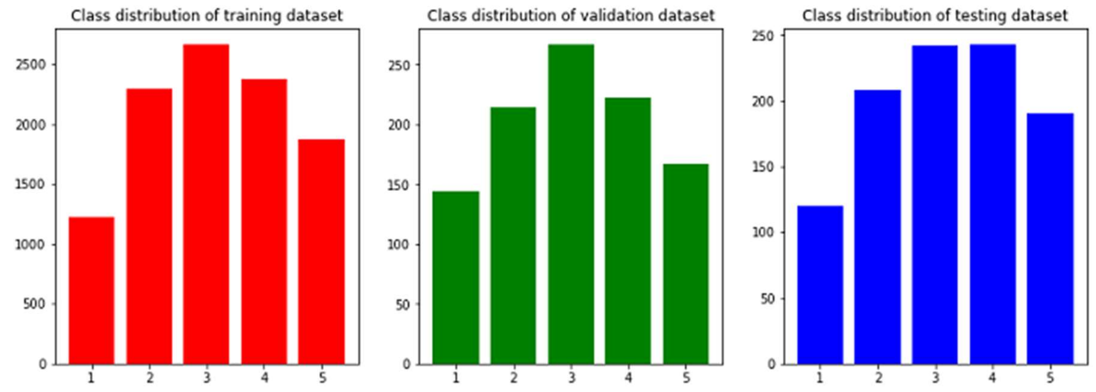
Using the provided function to download and arrange data, I traversed through the 3 json files to download the images into my workspace.

Dataset Statistics

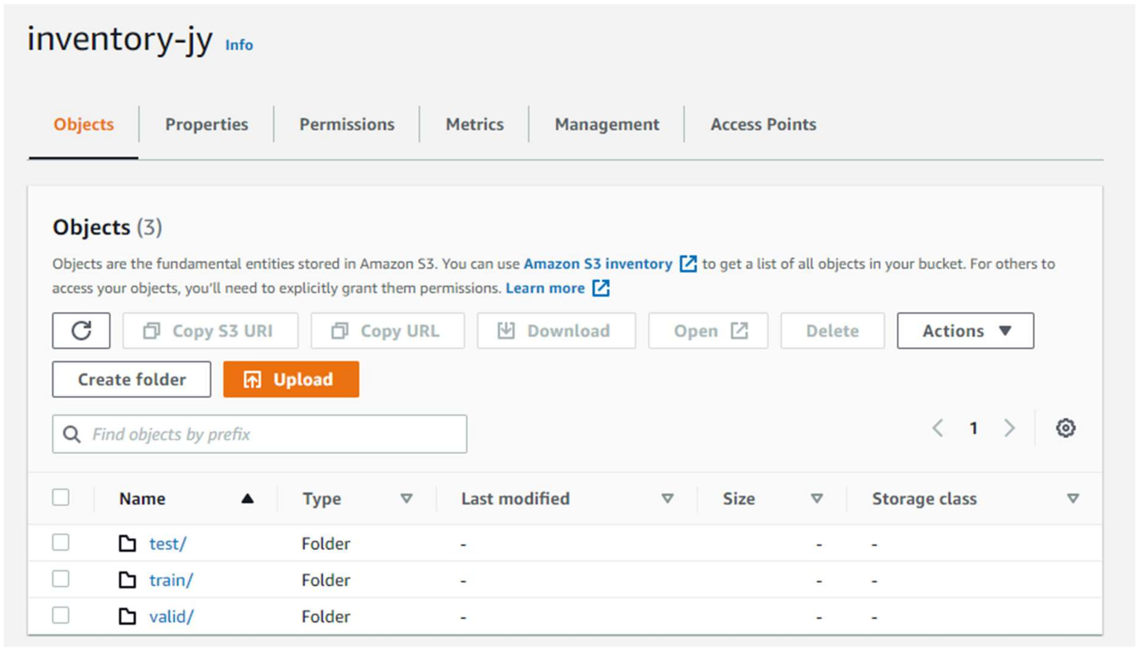
After downloading the images, I analysed the dataset sizes.

```
Number of images in training dataset: 10441
Number of images in validation dataset: 1015
Number of images in testing dataset: 1003
```

I plotted the histogram of training, validation and testing datasets to ensure they have similar data distribution. I notice that the most common label is "3" items.



Finally, I uploaded the training, validation and testing dataset to my S3 bucket so that SageMaker can use them for training. I checked that the images are uploaded correctly:



Objects (3)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
test/	Folder	-	-	-
train/	Folder	-	-	-
valid/	Folder	-	-	-

Model Training Script

Once the data is uploaded to S3, I prepared a training script "train.py" to read, load and preprocess my training, testing and validation data.

Firstly, I initialize the hyperparameters such as epochs, batch size, learning rate and momentum to specific values. I also set the default values for container environment, such as model directory and data directories.

Next, I set up a function net() to call a pre-trained ResNet-18 model. The convolutional layers are frozen so that the weights will not be updated during training. I also added a fully connected layer with 5 classes at output since we are predicting 1 to 5 item quantites.

In the main function, I specify my loss criterion as Cross Entropy Loss and optimizer as Adadelta function. I also preprocessed the input images by resizing and normalizing according to the documentation. Finally, I passed the datasets into the data loaders for training, validation and testing.

Train using SageMaker

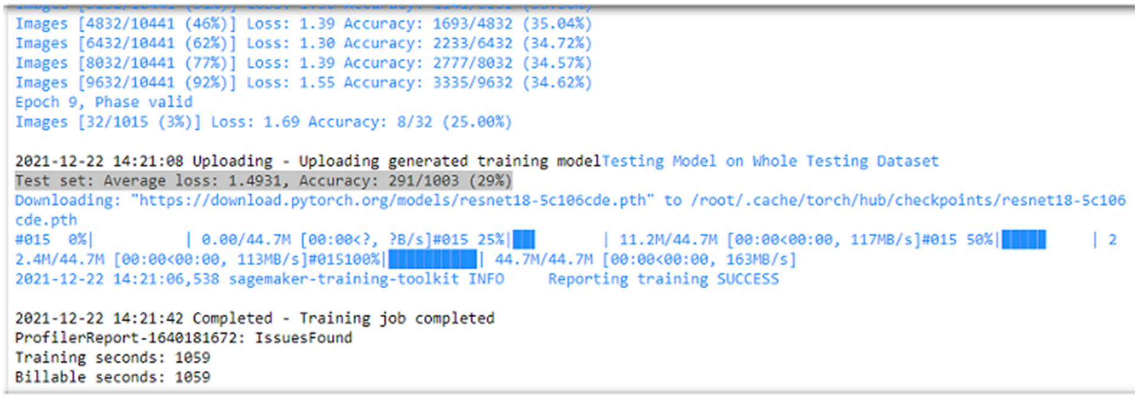
After creating the training script "train.py", I completed the remaining steps in the Jupyter notebook "sagemaker.ipynb". I fixed the values of learning rate and batch size as these 2 hyperparameters. These 2 factors affect the gradient descent of my model. Increasing the learning rate speeds up the learning of my model, yet risks overshooting its minimum loss. Smaller batch size takes up less computational memory but adds more noise to convergence.

Next, I created my training estimator by specifying the entry point, instance count, instance type and hyperparameters. I chose to use "ml.g4dn.xlarge" as my instance type since it uses GPU to speed up the training job. It takes about 20 mins to complete 10 epochs.

Finally, I fit my estimator by passing my S3 bucket directory. The estimator will retrieve my training, validation and testing data from my S3 bucket when running the training job.

Collect results

After training the ResNet-18 model for 10 epochs, it was able to correctly predict 291 out of 1003 images in the testing dataset. This represents an accuracy of 29%.



```
Images [4832/10441 (46%)] Loss: 1.39 Accuracy: 1693/4832 (35.04%)
Images [6432/10441 (62%)] Loss: 1.30 Accuracy: 2233/6432 (34.72%)
Images [8032/10441 (77%)] Loss: 1.39 Accuracy: 2777/8032 (34.57%)
Images [9632/10441 (92%)] Loss: 1.55 Accuracy: 3335/9632 (34.62%)
Epoch 9, Phase valid
Images [32/1015 (3%)] Loss: 1.69 Accuracy: 8/32 (25.00%)

2021-12-22 14:21:08 Uploading - Uploading generated training modelTesting Model on Whole Testing Dataset
Test set: Average loss: 1.4931, Accuracy: 291/1003 (29%)
Downloading: "https://download.pytorch.org/models/resnet18-5c106cde.pth" to /root/.cache/torch/hub/checkpoints/resnet18-5c106
cde.pth
#015  0%|          | 0.00/44.7M [00:00<?, ?B/s]#015 25%|██        | 11.2M/44.7M [00:00<00:00, 117MB/s]#015 50%|█████     | 2
2.4M/44.7M [00:00<00:00, 113MB/s]#015100%|██████████| 44.7M/44.7M [00:00<00:00, 163MB/s]
2021-12-22 14:21:06,538 sagemaker-training-toolkit INFO     Reporting training SUCCESS

2021-12-22 14:21:42 Completed - Training job completed
ProfilerReport-1640181672: IssuesFound
Training seconds: 1059
Billable seconds: 1059
```

Construct conclusion

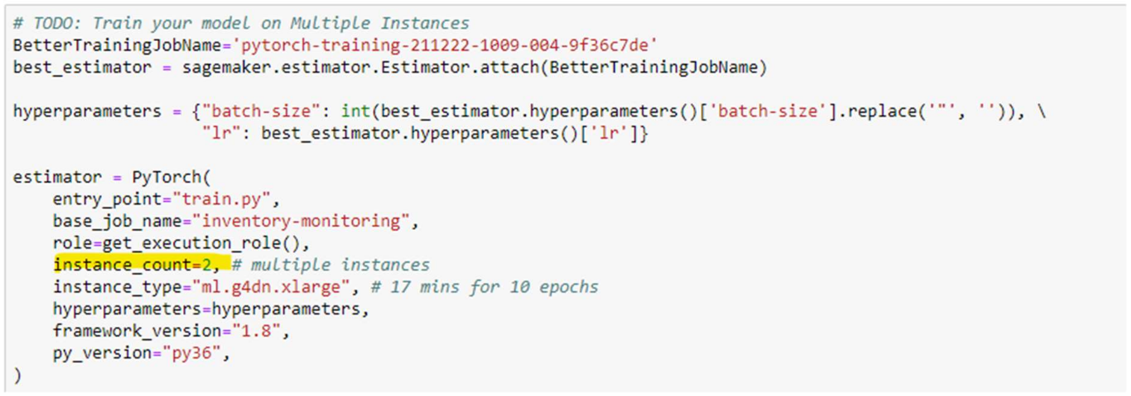
By fine tuning a pre-trained ResNet-18 model with about 10,000 images, I was able to achieve 29% accuracy after 10 epochs. This is not high, possibly due to the small dataset and the quality of the images.

For further improvement, I would conduct debugging and profiling. By analysing the loss value across the training steps, I can determine whether the model is overfitting. I would also read the profiler report to identify any resource bottlenecks and improve system utilization.

Standout Suggestions

Multi-instance training

I was able to execute multi-instance training by increasing the instance count when defining the estimator. This can accomplish training jobs faster than single-instance training.

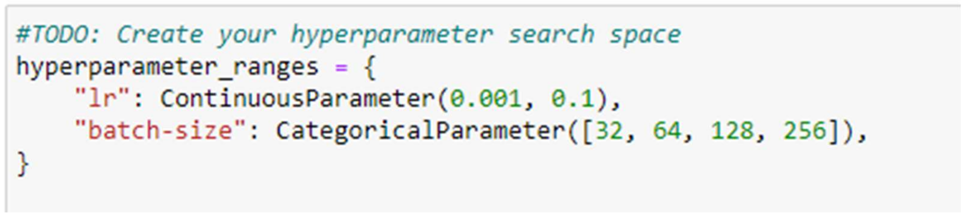


```
# TODO: Train your model on Multiple Instances
BetterTrainingJobName='pytorch-training-211222-1009-004-9f36c7de'
best_estimator = sagemaker.estimator.Estimator.attach(BetterTrainingJobName)

hyperparameters = {"batch-size": int(best_estimator.hyperparameters()['batch-size'].replace('"', '')), \
                   "lr": best_estimator.hyperparameters()['lr']}

estimator = PyTorch(
    entry_point="train.py",
    base_job_name="inventory-monitoring",
    role=get_execution_role(),
    instance_count=2, # multiple instances
    instance_type="ml.g4dn.xlarge", # 17 mins for 10 epochs
    hyperparameters=hyperparameters,
    framework_version="1.8",
    py_version="py36",
)
```

Hyperparameter Tuning

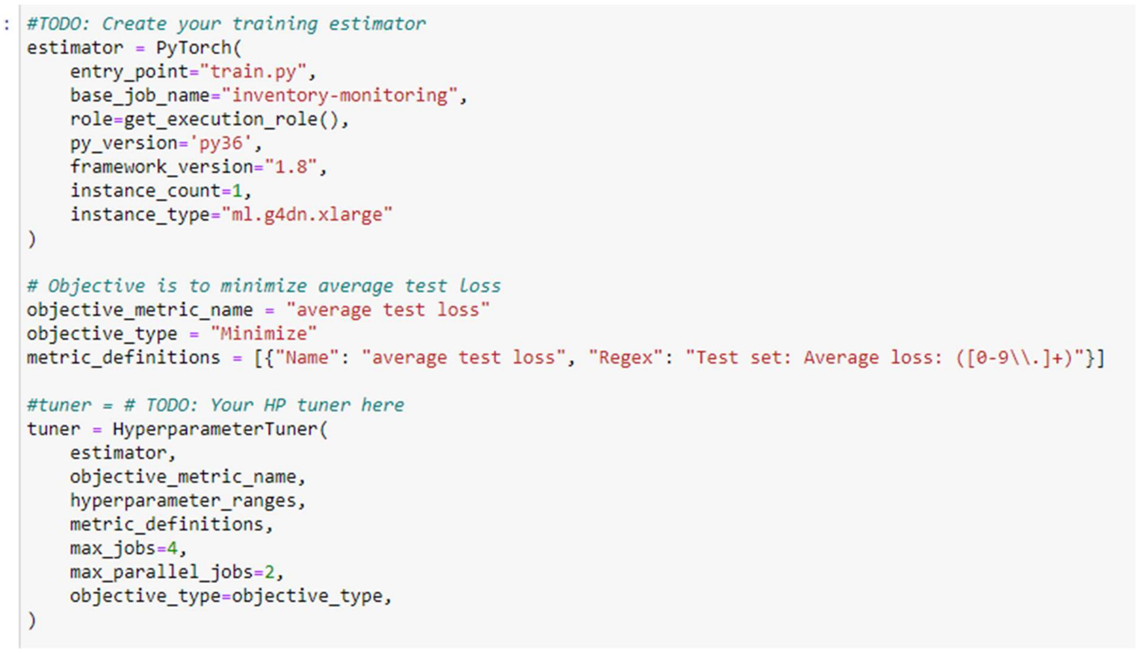
Previously, the learning rate and batch size are fixed at 0.05 and 32 respectively. Using hyperparameter tuning, I am able to identify the best values to train my model.

Firstly, I initialize learning rate as a continuous parameter from 0.01 to 0.1 and batch size as categorical parameter.



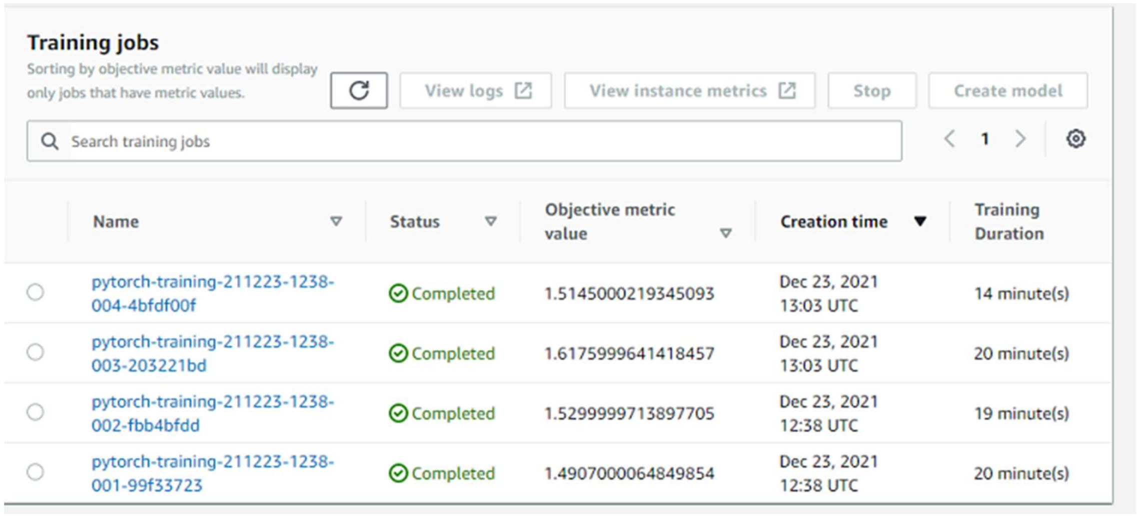
```
#TODO: Create your hyperparameter search space
hyperparameter_ranges = {
    "lr": ContinuousParameter(0.001, 0.1),
    "batch-size": CategoricalParameter([32, 64, 128, 256]),
}
```

Next, I defined the objective of the tuning job to minimize the loss function. I also defined the estimator with the instance count and instance type. Finally, I defined the tuner with the objective_metric_name, hyperparameter_ranges and number of parallel jobs.



```
: #TODO: Create your training estimator
estimator = PyTorch(
    entry_point="train.py",
    base_job_name="inventory-monitoring",
    role=get_execution_role(),
    py_version='py36',
    framework_version="1.8",
    instance_count=1,
    instance_type="ml.g4dn.xlarge"
)

# Objective is to minimize average test loss
objective_metric_name = "average test loss"
objective_type = "Minimize"
metric_definitions = [{"Name": "average test loss", "Regex": "Test set: Average loss: ([0-9\\.]+)"}]

#tuner = # TODO: Your HP tuner here
tuner = HyperparameterTuner(
    estimator,
    objective_metric_name,
    hyperparameter_ranges,
    metric_definitions,
    max_jobs=4,
    max_parallel_jobs=2,
    objective_type=objective_type,
)
```

Using SageMaker, I could view the training jobs and the loss value for each job.

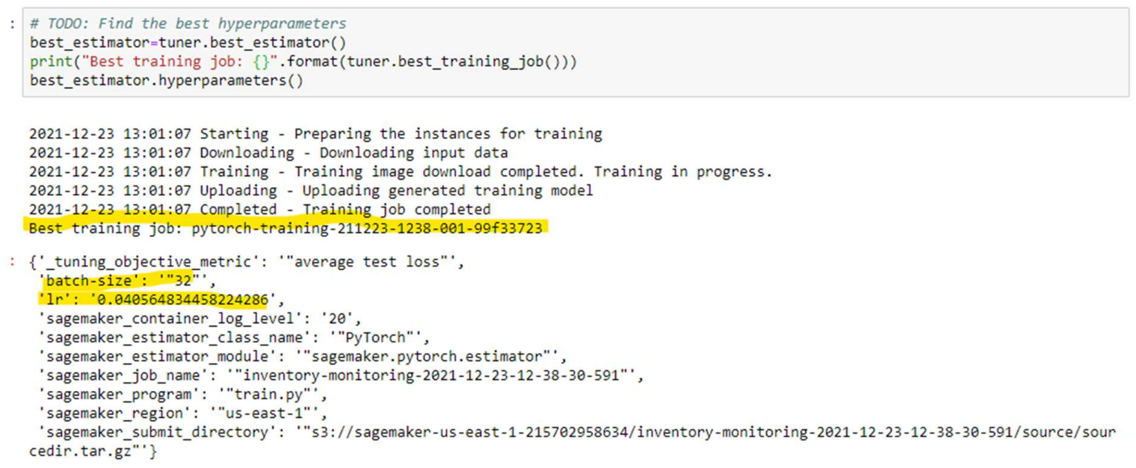


Training jobs

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

Name	Status	Objective metric value	Creation time	Training Duration
pytorch-training-211223-1238-004-4bfdf00f	Completed	1.5145000219345093	Dec 23, 2021 13:03 UTC	14 minute(s)
pytorch-training-211223-1238-003-203221bd	Completed	1.6175999641418457	Dec 23, 2021 13:03 UTC	20 minute(s)
pytorch-training-211223-1238-002-fbb4bfdd	Completed	1.5299999713897705	Dec 23, 2021 12:38 UTC	19 minute(s)
pytorch-training-211223-1238-001-99f33723	Completed	1.4907000064849854	Dec 23, 2021 12:38 UTC	20 minute(s)

I retrieved the best training job and its corresponding hyperparameters. These values will be used in my subsequent training job.



```
: # TODO: Find the best hyperparameters
  best_estimator=tuner.best_estimator()
  print("Best training job: {}".format(tuner.best_training_job()))
  best_estimator.hyperparameters()

  2021-12-23 13:01:07 Starting - Preparing the instances for training
  2021-12-23 13:01:07 Downloading - Downloading input data
  2021-12-23 13:01:07 Training - Training image download completed. Training in progress.
  2021-12-23 13:01:07 Uploading - Uploading generated training model
  2021-12-23 13:01:07 Completed - Training job completed
  Best training job: pytorch-training-211223-1238-001-99f33723

: {'_tuning_objective_metric': '"average test loss"',
   'batch-size': '"32"',
   'lr': '0.040564834458224286',
   'sagemaker_container_log_level': '20',
   'sagemaker_estimator_class_name': '"PyTorch"',
   'sagemaker_estimator_module': '"sagemaker.pytorch.estimator"',
   'sagemaker_job_name': '"inventory-monitoring-2021-12-23-12-38-30-591"',
   'sagemaker_program': '"train.py"',
   'sagemaker_region': '"us-east-1"',
   'sagemaker_submit_directory': '"s3://sagemaker-us-east-1-215702958634/inventory-monitoring-2021-12-23-12-38-30-591/source/sour
  cedir.tar.gz"'}
```

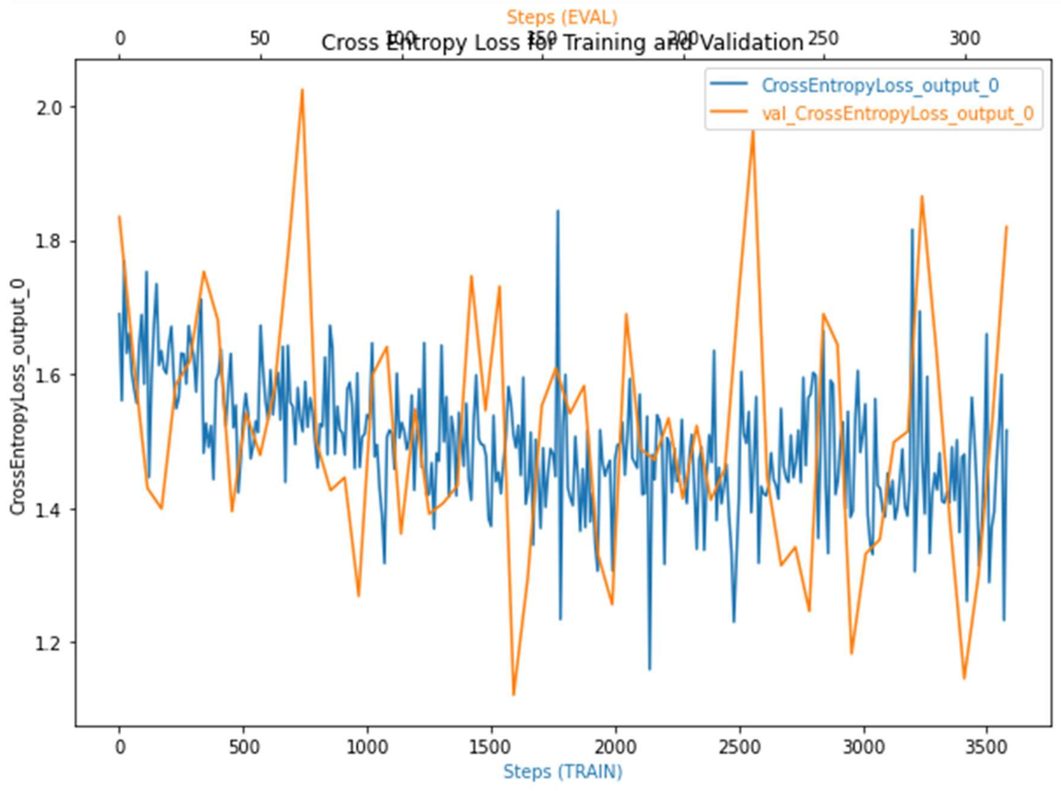
Debugging and Profiling

I executed debugging and profiling to find out if the training is working well and analyse system utilization.

Firstly, I added debugging hooks in my training script. Next, I defined the debugger and profiler rules and configuration. Finally, I passed these rules and configuration into my estimator.

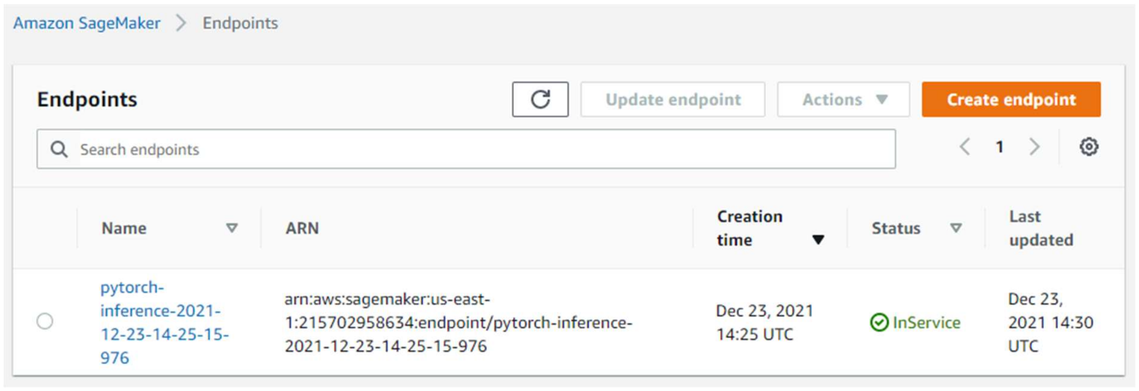
Once training is completed, I plotted the loss value for both training (blue) and validation (orange). I noticed that the training loss value decreases gradually and does not improve after 2000 steps. There is no clear trend in the validation loss.

This could be due to overfitting as the training dataset is small (10,000 images). I would increase the training dataset and re-plot the graph to analyse if there is any improvement.



Deployment

Using the saved model, I created my own inference script for deployment.



Name	ARN	Creation time	Status	Last updated
pytorch-inference-2021-12-23-14-25-15-976	arn:aws:sagemaker:us-east-1:215702958634:endpoint/pytorch-inference-2021-12-23-14-25-15-976	Dec 23, 2021 14:25 UTC	InService	Dec 23, 2021 14:30 UTC

To test my endpoint, I read the JPEG image and passed it to the predictor. The return value is an array where the largest value corresponds to the predicted class.

```
# TODO: Run an prediction on the endpoint

test_file = "./test_data/1/04309.jpg"

with open(test_file, "rb") as f:
    payload = f.read()
response=predictor.predict(payload, initial_args={"ContentType": "image/jpeg"})

# find the position of largest value which corresponds to predicted class
np.argmax(response, 1)
```