Inventory Monitoring at Distribution Centers

Definition

Project Overview

Most distribution centers all over the world use robots to move objects from one place to another. These robots use bins which contains multiple objects. Determining the number of objects in each bin can be very valuable in order to check that the process is working as expected.

The main goal of this project is to build a ML model that can count the number of objects in each bin in order to track inventory and check that bins have the appropriate number of items in order to reduce stock mismatches.

Problem Statement

Based on the background, it can be seen that the problem to be resolved here is related to image classification. A ton of images have been provided by our client (Amazon) and a ML model will be built in order to identify the number of objects in each bin.

Metrics

For this case, we will use both, the model accuracy (based on the images which are correctly identified by the model) and the Root Mean Square Error (RMSE) metrics in order to evaluate how the model is performing.

These metrics will be used at the end of each epoch in order to observe how the model is improving its results.

Analysis

Data Exploration

The dataset used for the project will be the Amazon Bin Image Dataset which contains more than 500,000 images and metadata from bins of a pod in an operating Amazon Fulfillment Center. Both, images and metadata, will be used to develop the model.

Images will be used as the main input in the training phase. Metadata will be used in order to arrange the pictures in a way that the ML model will identify the number of possible classes.

Amazon provide us with these images via S3. Specifically, both, images and metadata, are stored on the `aft-vbi-pds` bucket.

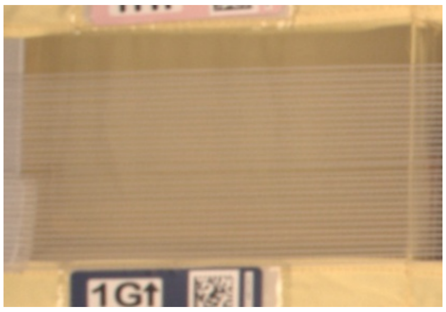
- Images are stored in the `bin-images` folder. Each image is a different `JPG` file whose name is similar to `<X>.jpg` being `<X>` a number.
- Metadata is stored in the `metadata` folder. Each image has a corresponding metadata file whose format is JSON. The most important field of these JSON files is `EXPECTED_QUANTITY` as it contains the

number of objects which can be found on the specific image.

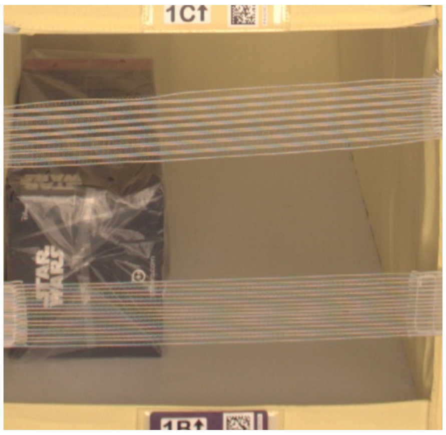
For the scope of this project, we will focus on pictures contain 0 to 5 objects.

Exploratory Visualization

The following picture is an example of a bin with no objects.



While this one is an example of a bin with 1 object:



As can be seen, each picture has different sizes, so it will be critical to resize all these pictures un order the model to work properly.

On the other hand, here is an example of a metadata file. In this case, the JSON is describing a picture with one object (it refers to the previous picture):

```
{
    "BIN_FCSKU_DATA": {
        "B01427MW3O": {
            "asin": "B01427MW3O",
            "height": {
                "unit": "IN",
                "value": 5.511811023622047
            },
            "length": {
                "unit": "IN",
                "value": 9.645669291338583
            },
            "name": "Sphero BB-8 App-Enabled Droid",
            "quantity": 1,
            "weight": {
                "unit": "pounds",
                "value": 2.1737357169469815
            },
            "width": {
                "unit": "IN",
                "value": 6.181102362204724
            }
        }
    },
    "EXPECTED_QUANTITY": 1,
    "image_fname": "301.jpg"
}
```

Algorithms and Techniques

In this project a ML model has been built to identify the number of objects in each image. In order to build this ML model, SageMaker has been used, training a model using a ResNet neuronal network.

ResNet model is widely used for image classification which is pretrained and can be customized in order to categorize images from different use cases. To adapt this pretrained model to our use case, different training jobs will be launched in AWS SageMaker. In addition, hyperparameters tunning jobs has been launched in order to find the most appropriate combination of hyperparameters for our use case.

Benchmark

Others have worked on the same data to build their own models. Specifically, we can find two GitHub projects related to this problem:

1. Amazon Bin Image Dataset (ABID) Challenge by silverbottlep
2. Amazon Inventory Reconciliation using AI by Pablo Rodriguez Bertorello, Sravan Sripada, Nutchapol Dendumrongsup

As can be seen in the conclusions of both projects, the obtained accuracy is 55% approximately with a RMSE of 0.94. In the results sections, a comparison between the results of my model and the results of these project can be found.

Methodology

Data preprocessing

The first step to train a model is to download an process the data which will be used as the input. As stated before, we have decided to focus on pictures with 0 to 5 objects. Each picture will be assigned a class according to the following rule:

- Class 1 for pictures without objects
- Class 2 for pictures with 1 object
- Class 3 for pictures with 2 objects
- Class 4 for pictures with 3 objects
- Class 5 for pictures with 4 objects
- Class 6 for pictures with 5 objects

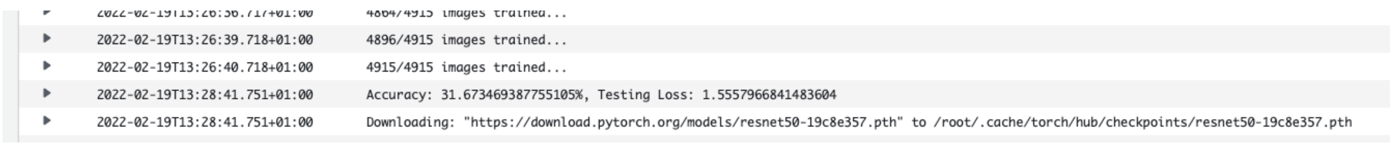
For each class, I decided to download 1,000 images: 800 will be used as input for the training phase, while 200 will be used as input for the validation phase.

In order to download these 6,000 pictures, a Python script has been created. The script can be found at the project Jupyter notebook (file `sagemaker.ipynb`). Specifically, the script will iterate over the JSON files from the Amazon dataset and will download the picture if it contains 0 to 5 objects and the number of objects downloader for the specific class is below 1,000.

Finally, all these pictures were uploaded to S3, as it is the entry point for data for models being trained on AWS.

Implementation

As stated before, I planned to use a ResNet neuronal network to train the model. As a base I used this Python training script, which is the one I implemented for the "Image Classification" project of this course (file `train.py`). I adapted the number of classes (from 133 to 6) and configured the transformation part in order to deal with the new set of images (i.e. resizing). Then I launched this script through the Jupyter Notebook. However, the first results, as can be seen on this screenshot, were not very promising, with a RMSE of 1.55 and an accuracy of 31,60%.



```
▸    2022-02-19T13:26:38.717+01:00    4864/4915 images trained...
▸    2022-02-19T13:26:39.718+01:00    4896/4915 images trained...
▸    2022-02-19T13:26:40.718+01:00    4915/4915 images trained...
▸    2022-02-19T13:28:41.751+01:00    Accuracy: 31.673469387755105%, Testing Loss: 1.5557966841483604
▸    2022-02-19T13:28:41.751+01:00    Downloading: "https://download.pytorch.org/models/resnet50-19c8e357.pth" to /root/.cache/torch/hub/checkpoints/resnet50-19c8e357.pth
```

This first script used a pretrained ResNet18 neuronal network for training. In order to obtain a more accurate model, I tried the same script but with different ResNet networks. I tested with ResNet50 and the results were pretty similar.

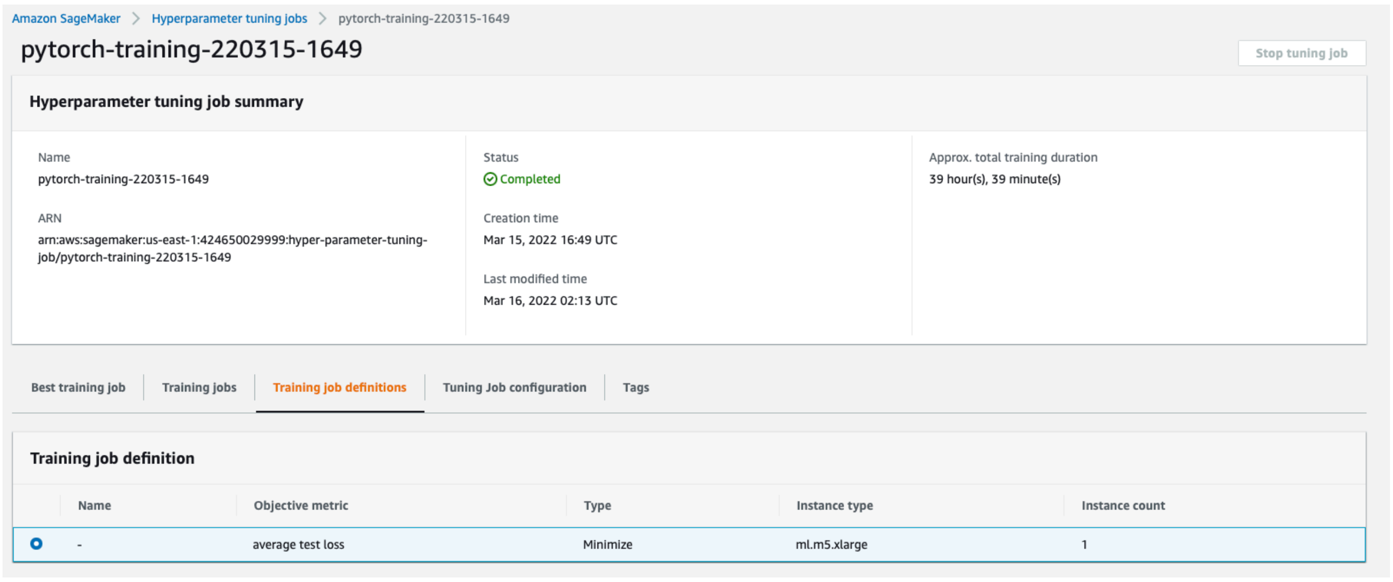
However, reading the project done by Pablo Rodriguez et al., I discovered they were using a ResNet34 non-pretrained network, so I decided to move my project to this implementation (ResNet34) but with a pretrained network. With this implementation, accuracy was 38% and RMSE dropped to 1.38, which was not perfect but better than the results obtained on the first attempt.

Refinement

At this state, I decided it was high time to tune the hyperparameters in order to find a better combination of them which allow me to obtain a more precise model. Specifically, these hyperparameters were tuned:

- The number of epochs
- The batch size (the number of images being trained on each iteration)
- The learning rate (`lr`)

Here is a screenshot of the hyperparameters job launched on AWS:



pytorch-training-220315-1649

Hyperparameter tuning job summary

Name
pytorch-training-220315-1649

ARN
arn:aws:sagemaker:us-east-1:424650029999:hyper-parameter-tuning-job/pytorch-training-220315-1649

Status
Completed

Creation time
Mar 15, 2022 16:49 UTC

Last modified time
Mar 16, 2022 02:13 UTC

Approx. total training duration
39 hour(s), 39 minute(s)

Training job definition

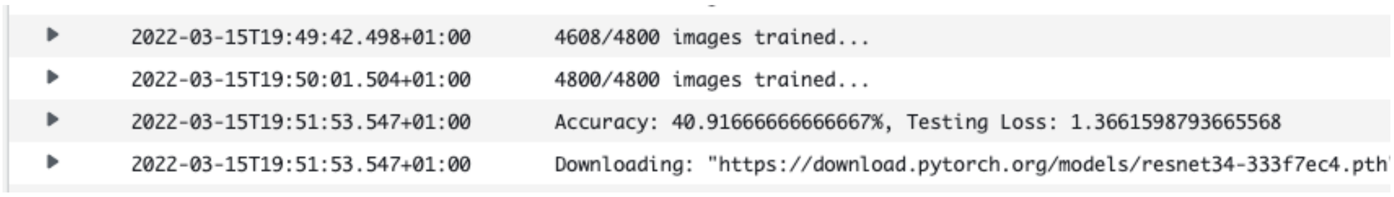
	Name	Objective metric	Type	Instance type	Instance count
●	-	average test loss	Minimize	ml.m5.xlarge	1

After completing, the best hyperparameters combination was the following one:

- Epochs: 12

- Batch Size: 256
- Learning Rate: 0.0055476177746041485

With this combination, testing accuracy spiked to 41% and RMSE was reduced to 1.36, as can be seen on the following screenshot.



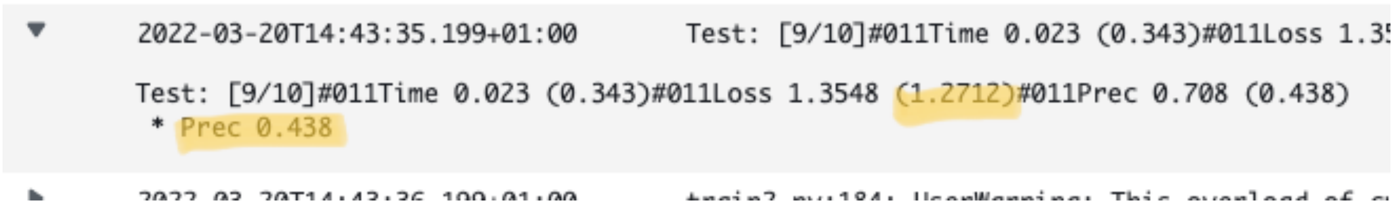
```
▶    2022-03-15T19:49:42.498+01:00      4608/4800 images trained...
▶    2022-03-15T19:50:01.504+01:00      4800/4800 images trained...
▶    2022-03-15T19:51:53.547+01:00      Accuracy: 40.916666666666667%, Testing Loss: 1.3661598793665568
▶    2022-03-15T19:51:53.547+01:00      Downloading: "https://download.pytorch.org/models/resnet34-333f7ec4.pth
```

Results

Model Evaluation and Validation

As can be seen on the previous picture, results were very poor if compared with the ones obtained on other projects such as the one developer by Pablo Rodriguez et al. were a RMSE of 0.94 is obtained on the testing phase. At this point, I decided to launch their training script against my dataset.

To do so, I modified their script so it can be run on AWS with the input being stored on S3 (file: `train_external.py`). Once executed, here is an screenshot of the results I obtained:



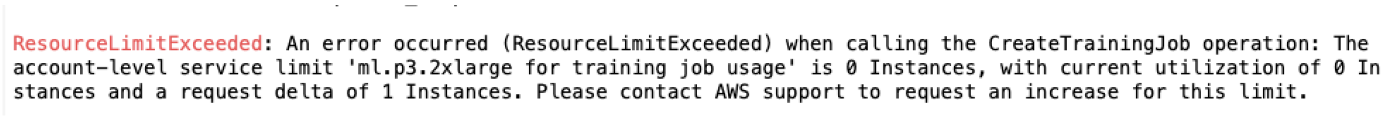
```
▼    2022-03-20T14:43:35.199+01:00      Test: [9/10]#011Time 0.023 (0.343)#011Loss 1.35

     Test: [9/10]#011Time 0.023 (0.343)#011Loss 1.3548 (1.2712)#011Prec 0.708 (0.438)
      * Prec 0.438
```

As can be seen, results with the same input was pretty close to what I obtained, with a RMSE of 1.27 and an accuracy of 43,8%. It's important to note that while their script requires 40 epochs to obtain this results, my script only requires 12.

Justification

As stated before, both training scripts, the one I developed and the one developed by the team of Pablo Rodriguez, generated pretty close results with testing accuracy differing in just 3 points (40,9% to 43,80%).

Based on this, I tried to obtain better results by moving my script to GPUs, so I can get results more quickly with a higher number of epochs. However, when I tried to launch this script using GPUs, I discovered that Udacity has limited my permissions to launch this type of jobs (as can be seen on the attached screenshot), so I decided this was a good time to stop since I have no more options to improve my model.



```
ResourceLimitExceeded: An error occurred (ResourceLimitExceeded) when calling the CreateTrainingJob operation: The
account-level service limit 'ml.p3.2xlarge for training job usage' is 0 Instances, with current utilization of 0 In
stances and a request delta of 1 Instances. Please contact AWS support to request an increase for this limit.
```