Project Report for Inventory Monitoring at Distribution Centers

Vinamra Sareen , vsareen24@gmail.com

Project Definition

Overview

Distribution centres often use robots to move objects as a part of their operations. Objects are carried in bins which can contain multiple objects. Sometimes items get misplaced and to prevent mismatch while maintaining inventory records an efficient system can be put in place.

In this project, we will have to build a model that can count the number of objects in each bin. Creating a system like this can be used to track inventory and make sure that delivery consignments have the correct number of items.

Problem Statement

As described above in the domain, robots carry out the job of moving objects in distribution centres and items might get misplaced. To tackle the problem of mismanagement we are introducing an object counting task. Developing a solution for this problem also has some real world applications such as crowd surveillance, traffic monitoring, wildlife conservation and inventory management.

For this task, we have images of bin and we know each bin can contain objects in range of 1-5. If we develop a model that can take a picture of the bin and accurately predict the number of objects present in the bin, we can solve a crucial problem in the inventory management system.

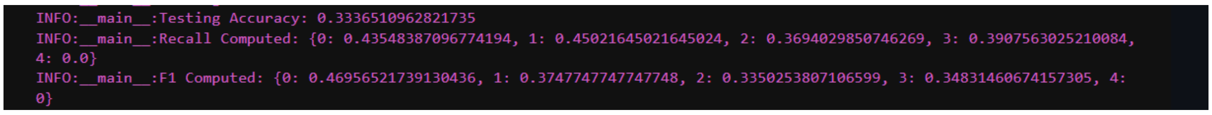
Evaluation Metrics

The below metric have been used for this task:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{F1 Score} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$



```
INFO:__main__:Testing Accuracy: 0.3336510962821735
INFO:__main__:Recall Computed: {0: 0.43548387096774194, 1: 0.45021645021645024, 2: 0.3694029850746269, 3: 0.3907563025210084,
4: 0.0}
INFO:__main__:F1 Computed: {0: 0.46956521739130436, 1: 0.3747747747747748, 2: 0.3350253807106599, 3: 0.34831460674157305, 4:
0}
```

	1	2	3	4	5
Recall	.43	.45	.36	.39	0
F1 Score	0.46	0.37	0.33	0.34	0

The model was not able to identify images with more than 4 objects in testing.

Analysis

Dataset Exploration and Visualisation

We used the data provided by amazon for our model training and inference. The info about dataset can be found here:

Amazon Bin Image Dataset - Registry of Open Data on AWS ("Amazon Bin Image Dataset")

A bit of overview of data:

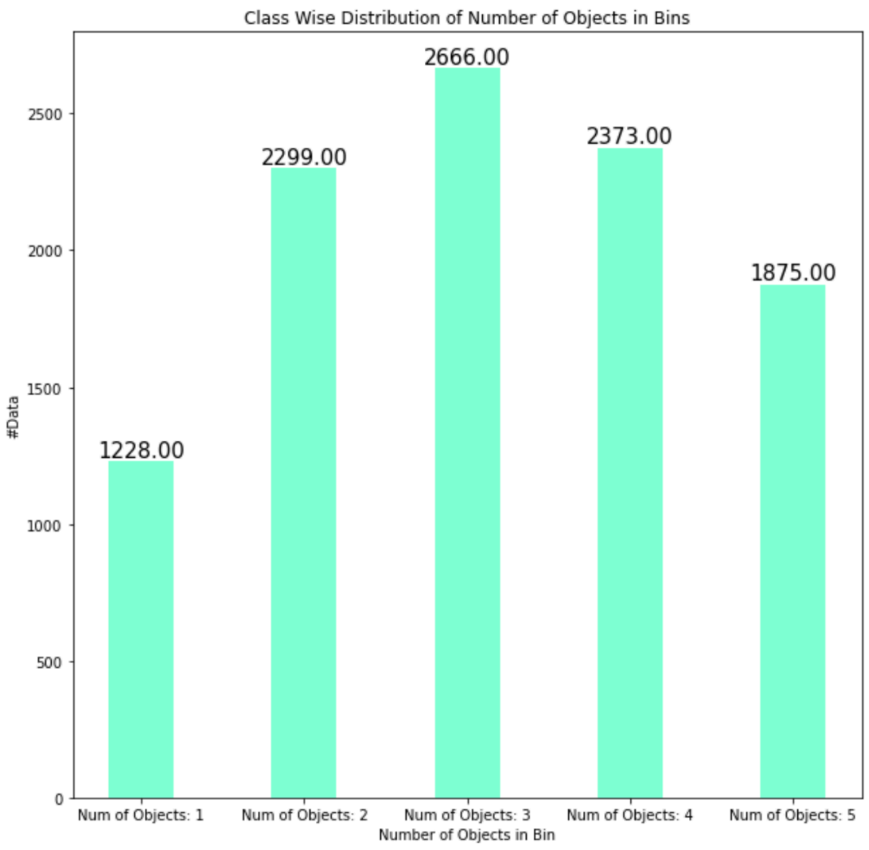
The Amazon Bin Image Dataset contains over 500,000 images and metadata from bins of a pod in an operating Amazon Fulfilment Centre. The bin images in this dataset are captured as robot units carry pods as part of normal Amazon Fulfilment Centre operations.

Example of metadata json file:

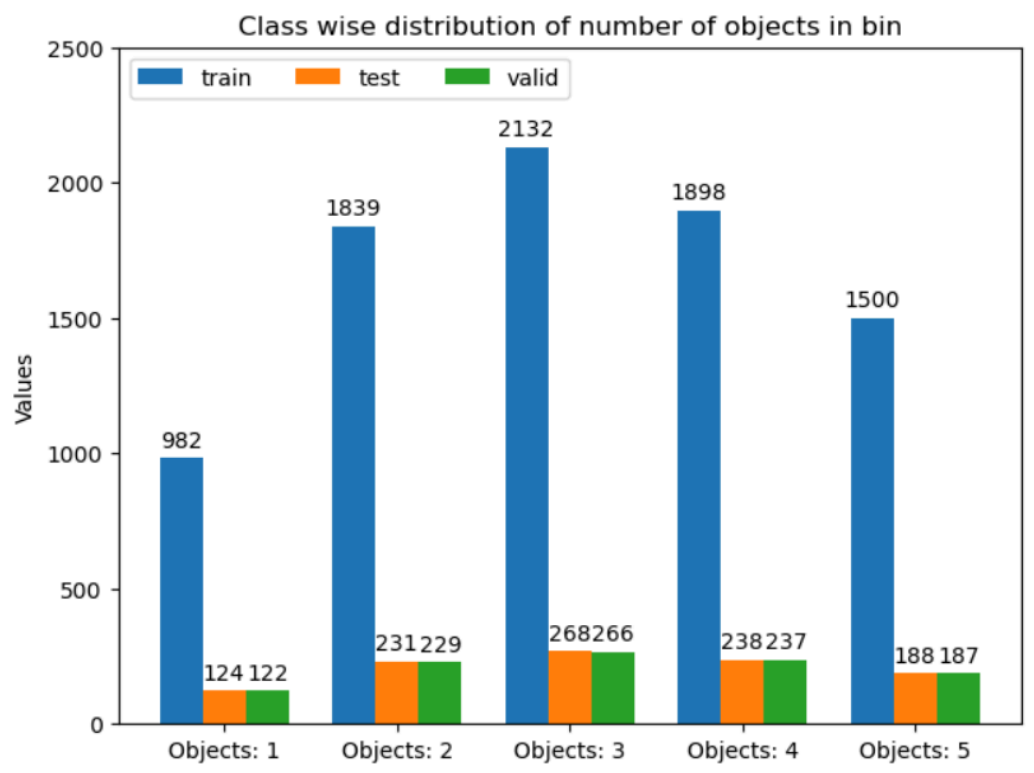
```
{
   "BIN_FCSKU_DATA": {
      "B000A8C5QE": {
         "asin": "B000A8C5QE",
         "height": {
            "unit": "IN",
            "value": 4.200000000000001
         },
         "length": {
            "unit": "IN",
            "value": 4.7
         },
         "name": "MSR PocketRocket Stove",
         "quantity": 1,
         "weight": {
            "unit": "pounds",
            "value": 0.45
         },
         "width": {
            "unit": "IN",
            "value": 4.4
         }
      },
      "B0064LIWVS": {
         "asin": "B0064LIWVS",
         "height": {
            "unit": "IN",
            "value": 1.2
         },
         "length": {
            "unit": "IN",
            "value": 5.799999999999999
         },
         "name": "Applied Nutrition Liquid Collagen Skin Revitalization, 10 Count 3.35 Fl Ounce",
         "quantity": 1,
         "weight": {
            "unit": "pounds",
            "value": 0.3499999999999999
         },
         "width": {
            "unit": "IN",
            "value": 4.7
         }
      }
   },
   "EXPECTED_QUANTITY": 2,
   "image_fname": "523.jpg"
}
```

Example of image for above metadata file:

We have been provided with a **file_list.json (Udacity)** that has about 10,441 image json objects. Each being grouped with keys 1-5, representing labels for images. We will be preferring the **file_list.json (Udacity)** over the whole dataset of 500k+ records as it will be computationally expensive. Below is class wise distribution of



images.
The below image represents splitting of the whole dataset into train, test and validation dataset wrt to labels.



Algorithms and Techniques

I used a ResNet50 model using transfer learning to implement a solution for the problem statement. I performed hyperparameter tuning and chose the best estimator values and based on them training was performed. I can further enhance the accuracy of the model by using the whole dataset with balanced distribution.

Benchmark

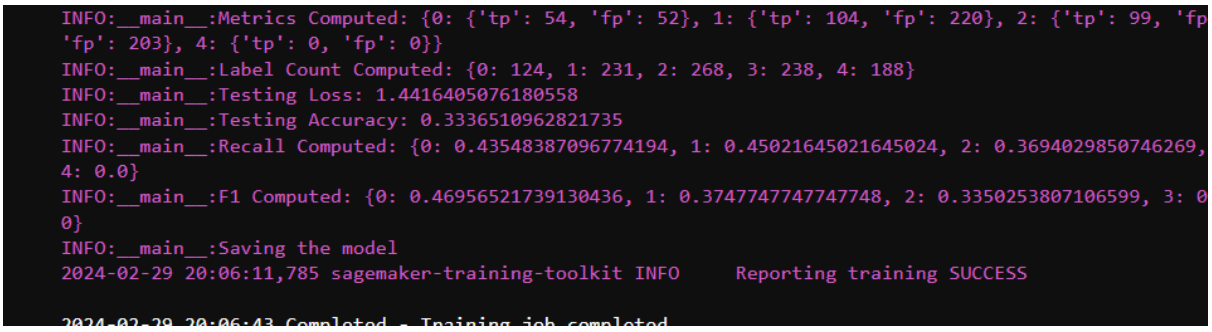
Benchmarks are used to compare performance of models. It can help us improve our model and see if it is close or how much far away from the established accuracy achieved so far in the same domain/problem.

The github repo from amazon challenge has an accuracy provided for the same problem, the accuracy specified there is about 55%.

GitHub - silverbottlep/abid_challenge: Amazon Bin Image Dataset Challenge (silverbottlep)

The above challenge was performed on a whole dataset of 535,234 images.We have a benchmark accuracy of 55% achieved so far.

After performing training on our model, we were able to reach an accuracy of 33.3% on 10,441 image dataset.



```
INFO:__main__:Metrics Computed: {0: {'tp': 54, 'fp': 52}, 1: {'tp': 104, 'fp': 220}, 2: {'tp': 99, 'fp
'fp': 203}, 4: {'tp': 0, 'fp': 0}}
INFO:__main__:Label Count Computed: {0: 124, 1: 231, 2: 268, 3: 238, 4: 188}
INFO:__main__:Testing Loss: 1.4416405076180558
INFO:__main__:Testing Accuracy: 0.3336510962821735
INFO:__main__:Recall Computed: {0: 0.43548387096774194, 1: 0.45021645021645024, 2: 0.3694029850746269,
4: 0.0}
INFO:__main__:F1 Computed: {0: 0.46956521739130436, 1: 0.3747747747747748, 2: 0.3350253807106599, 3: 0
0}
INFO:__main__:Saving the model
2024-02-29 20:06:11,785 sagemaker-training-toolkit INFO     Reporting training SUCCESS

2024-02-29 20:06:43 Completed - Training job completed
```

Methodology

Data Preprocessing

We split the data in three parts:

- train (80%)
- test (10%)
- validation (10%).

The data was uploaded to s3 using the boto3 library and using a data loader from torchvision we fetched the data for training.

```
def create_data_loaders(args, batch_size):
    train_data_path = os.path.join(args.data)
    test_data_path = os.path.join(args.test_dir)
    validation_data_path = os.path.join(args.valid_dir)

    train_transform = transforms.Compose([
        transforms.RandomResizedCrop((224, 224)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

    test_transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

    train_data = torchvision.datasets.ImageFolder(
        root=train_data_path,
        transform=train_transform
    )
    train_data_loader = torch.utils.data.DataLoader(
        train_data,
        batch_size=batch_size,
        shuffle=True,
    )

    test_data = torchvision.datasets.ImageFolder(
        root=test_data_path,
        transform=test_transform
    )
    test_data_loader = torch.utils.data.DataLoader(
        test_data,
        batch_size=batch_size,
        shuffle=False,
    )

    validation_data = torchvision.datasets.ImageFolder(
        root=validation_data_path,
        transform=test_transform,
    )
    validation_data_loader = torch.utils.data.DataLoader(
        validation_data,
        batch_size=batch_size,
        shuffle=False,
    )

    return train_data_loader, test_data_loader, validation_data_loader
```

We can see below transformations that are applied for training data.

- Randomly resize by cropping to dimensions of 224, 224.
- Random Horizontal Flip
- Convert To Tensor
- Normalise the image
- Shuffle before feeding to model

```
        transforms.RandomResizedCrop((224, 224)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
```

Implementation

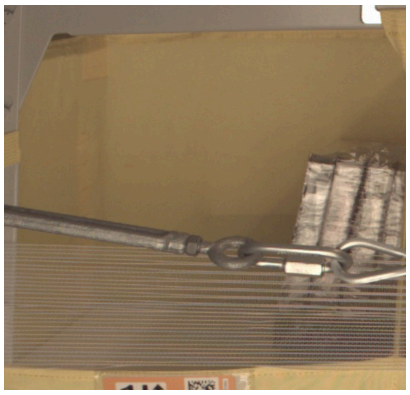
Follow **sagemaker.ipynb** for detailed implementation of this project. I followed all the project rubrics guidelines that were outlined as well as performed some standout suggestions.

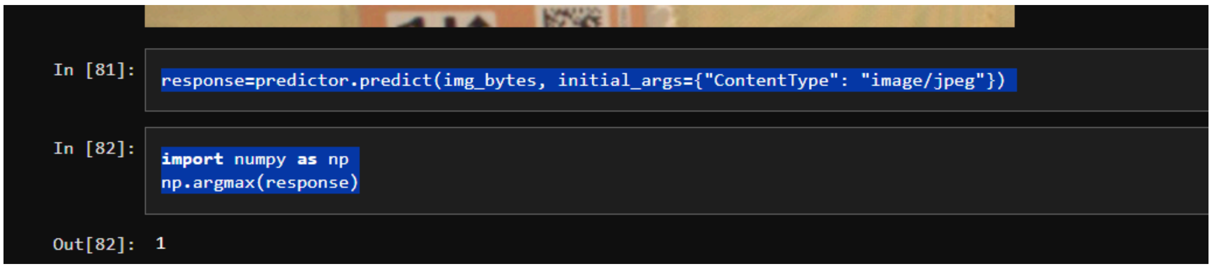
1. Implemented Hyperparameter Optimization code in **hpo.py**
    - Used ResNet50 pretrained model.
    - CrossEntropyLoss as loss function
    - Adam as optimiser
    - GPU support
    - Data loader and transformer
    - Hyperparameters:
        i. Learning rate: 0.003
        ii. Batch size: 128

2. Implemented model train script in **train.py** where profiler and debugging hooks were set up and all evaluation metrics were calculated and logged.
    - Used ResNet50 pretrained model.
    - CrossEntropyLoss as loss function
    - AdamW as optimiser
    - GPU support
    - Data loader and transformer
    - Metrics calculation
    - Hyperparameters were chosen from best values after hyperparameter tuning.

3. Dataset
    - Downloaded dataset from https://registry.opendata.aws/amazon-bin-imagery/ s3 bucket mentioned.
    - Visualised distribution of dataset split.
    - Used data loader and transform to handle data for model training.

4. Generated Profiling report to analyse enhancement strategies.
5. Deployed Model on instance type **ml.m5.large**.
6. Performed Inference on deployed endpoint.





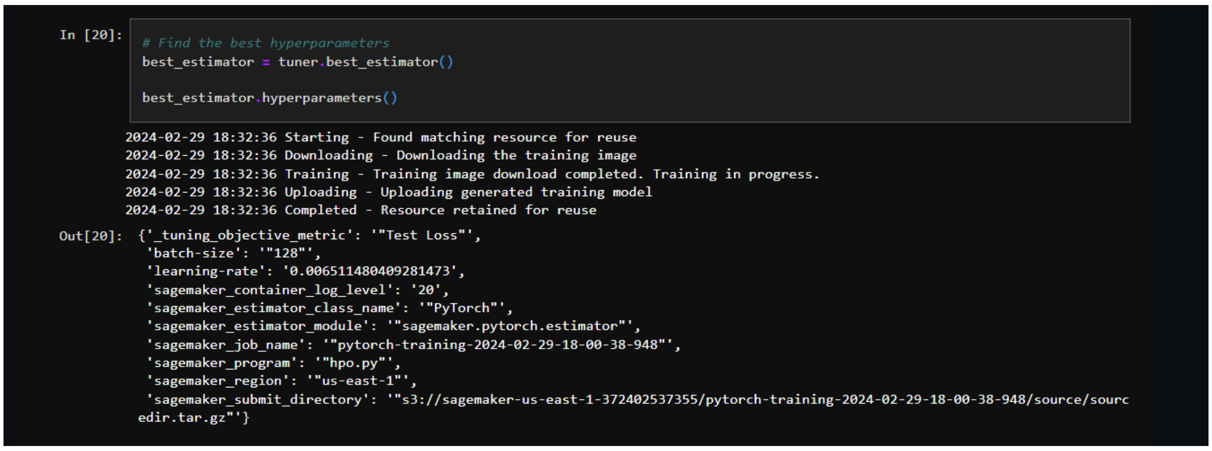
```
In [81]:  response=predictor.predict(img_bytes, initial_args={"ContentType": "image/jpeg"})

In [82]:  import numpy as np
          np.argmax(response)

Out[82]:  1
```

Refinement

I used hyperparameter tuning to get the best estimator values to perform training on our model. We performed hyperparameter tuning for 5 epochs which gave these as best values:



```
In [20]:  # Find the best hyperparameters
          best_estimator = tuner.best_estimator()

          best_estimator.hyperparameters()

         2024-02-29 18:32:36 Starting - Found matching resource for reuse
         2024-02-29 18:32:36 Downloading - Downloading the training image
         2024-02-29 18:32:36 Training - Training image download completed. Training in progress.
         2024-02-29 18:32:36 Uploading - Uploading generated training model
         2024-02-29 18:32:36 Completed - Resource retained for reuse

Out[20]:  {'_tuning_objective_metric': '"Test Loss"',
           'batch-size': '"128"',
           'learning-rate': '0.006511480409281473',
           'sagemaker_container_log_level': '20',
           'sagemaker_estimator_class_name': '"PyTorch"',
           'sagemaker_estimator_module': '"sagemaker.pytorch.estimator"',
           'sagemaker_job_name': '"pytorch-training-2024-02-29-18-00-38-948"',
           'sagemaker_program': '"hpo.py"',
           'sagemaker_region': '"us-east-1"',
           'sagemaker_submit_directory': '"s3://sagemaker-us-east-1-372402537355/pytorch-training-2024-02-29-18-00-38-948/source/sourc
          edir.tar.gz"'}
```

Using these mentioned values we performed training for 10 epochs.

Results

Model Evaluation and Validation

The metrics we used for evaluating our model:

1. Test Accuracy
2. F1 Score
3. Recall

These are the class wise scores of model evaluation:

	1	2	3	4	5
Recall	.43	.45	.36	.39	0
F1 Score	0.46	0.37	0.33	0.34	0

The overall accuracy of the test is 33.3 % which is far below than the benchmark model with 55% accuracy.

I also performed inference on a model deployed, which was able to predict some images from class 1-4 but for class 5 no image was correctly predicted which makes sense as we see score for recall and f1 score.

Justification

From analysis performed above, we can see scores of both f1 and recall. We were not able to reach near benchmark model which I believe can be improved by using below approaches:

- Adding more data to the train model.
- We are Randomly cropping image to fit dimension of 224*224, instead simply resizing makes more sense.
- Trying more hyperparameter combinations.

One of the reasons class 5 objects were not being classified by models could be heavy packaging on bin, which can obstruct the view for the model to learn features.

Reflection

What I learned from this lesson:

1. Learned about a domain and similar real word problems that can be tackled from this solution.
2. Learned boto3 for data uploading and matplotlib for visualisation.
3. Identify class imbalance.
4. Learned benchmarks, what they are and how they can be used for guidance.
5. Using pre trained models and fine tuning them on custom dataset.
6. Implemented training model on AWS utilising services such as Sagemaker Studio, S3.
7. Performed Hyperparameter tuning and how to get best parameter values.
8. Implemented hooks in training code and debugger.
9. Generated a profiler report.
10. Deployed model and performed inference on it.