CAPSTONE PROJECT REPORT

Nguyen Phung Hung

Inventory Monitoring at Distribution Centers Capstone Project

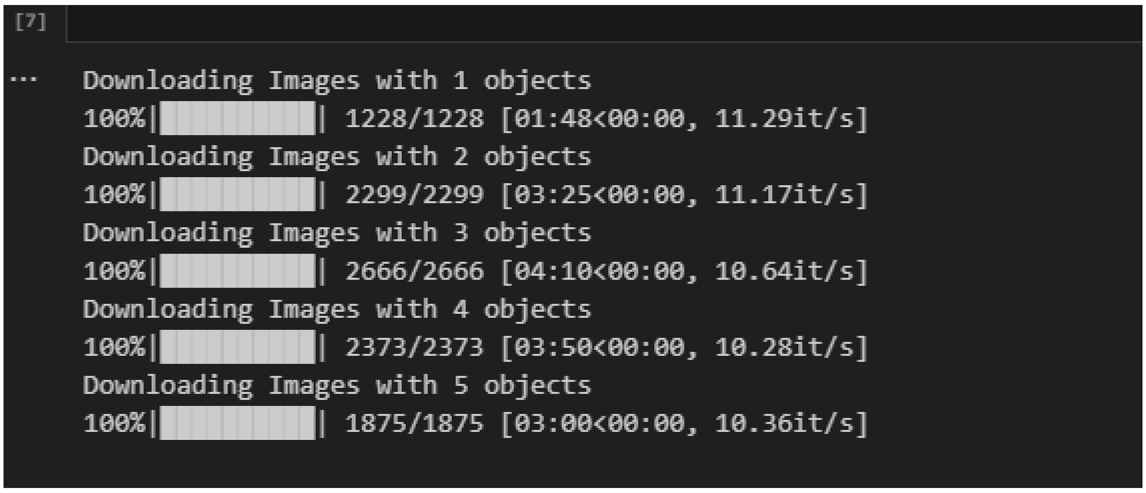
1. Domain Background:

Distribution centers often use robots to move objects as a part of their operations. Objects are carried in bins which can contain multiple objects. In this project, you will have to build a model that can count the number of objects in each bin. A system like this can be used to track inventory and make sure that delivery consignments have the correct number of items.This is where robots come in to help in Inventory Monitoring. They can be trained with Machine Learning Models, to perform tasks like Object Detection, Outlier & Anomaly Detection and much more. Once trained, these models are scalable, and can be deployed at a low cost for usage in actual warehouses and distribution centres on industry level robots.

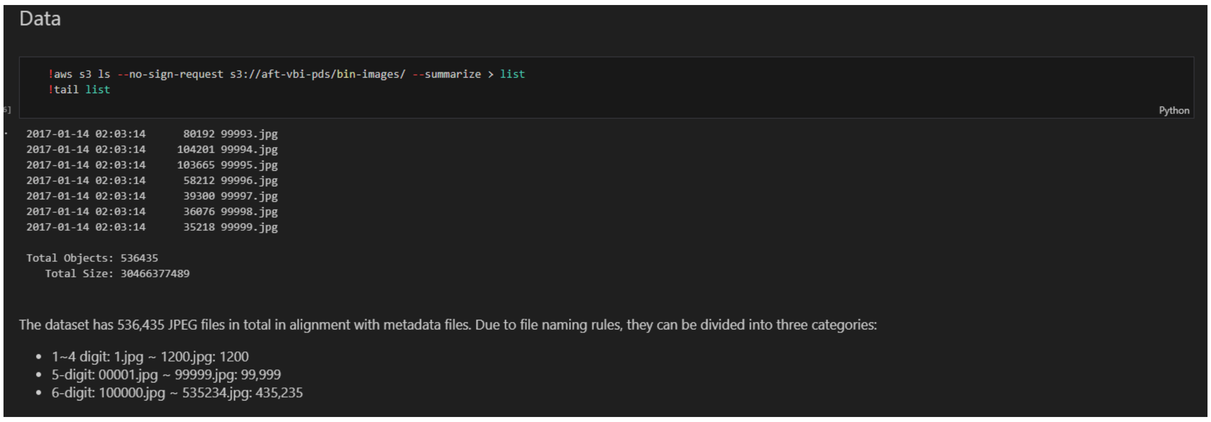
To build this project i will use AWS SageMaker and good machine learning engineering practices to fetch data from a database, preprocess it, and then train a machine learning model. This project will serve as a demonstration of end-to-end machine learning engineering skills that you have learned as a part of this nanodegree.

2. Analysis and Data Processing

Here is some exploratory analysis about the Dataset Amazon Image Bin, the distribution can be captured in histogram of barchart and display



```
[7]
...  Downloading Images with 1 objects
     100%|██████████| 1228/1228 [01:48<00:00, 11.29it/s]
     Downloading Images with 2 objects
     100%|██████████| 2299/2299 [03:25<00:00, 11.17it/s]
     Downloading Images with 3 objects
     100%|██████████| 2666/2666 [04:10<00:00, 10.64it/s]
     Downloading Images with 4 objects
     100%|██████████| 2373/2373 [03:50<00:00, 10.28it/s]
     Downloading Images with 5 objects
     100%|██████████| 1875/1875 [03:00<00:00, 10.36it/s]
```



Data

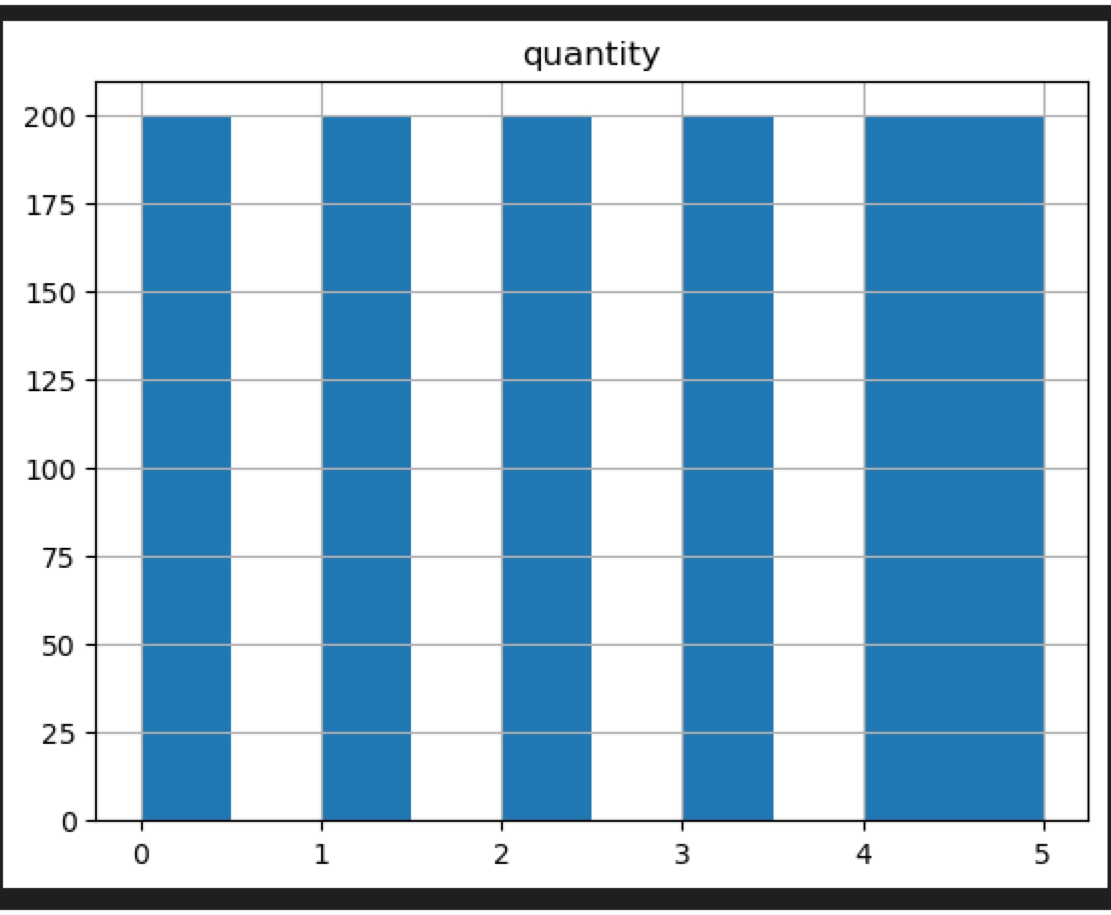
```
!aws s3 ls --no-sign-request s3://aft-vbi-pds/bin-images/ --summarize > list
!tail list
```

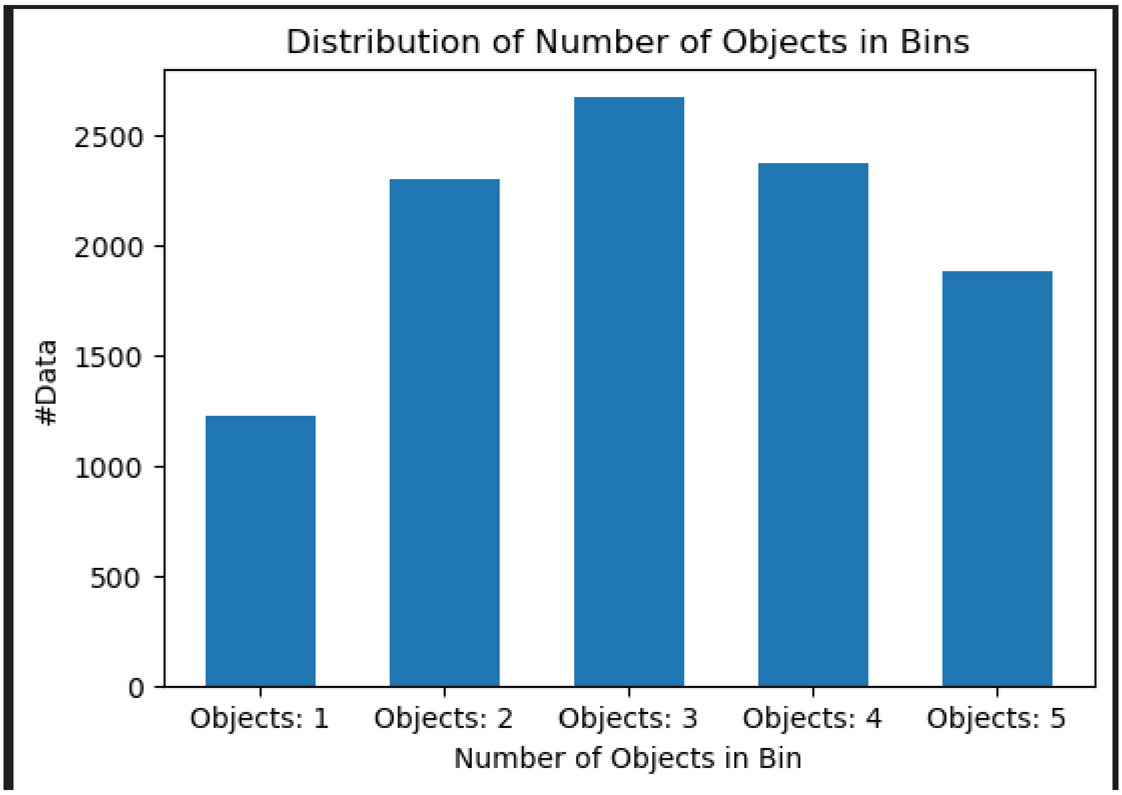
Python

```
2017-01-14 02:03:14      80192 99993.jpg
2017-01-14 02:03:14     104201 99994.jpg
2017-01-14 02:03:14     103665 99995.jpg
2017-01-14 02:03:14      58212 99996.jpg
2017-01-14 02:03:14      39300 99997.jpg
2017-01-14 02:03:14      36076 99998.jpg
2017-01-14 02:03:14      35218 99999.jpg

Total Objects: 536435
   Total Size: 30466377489
```

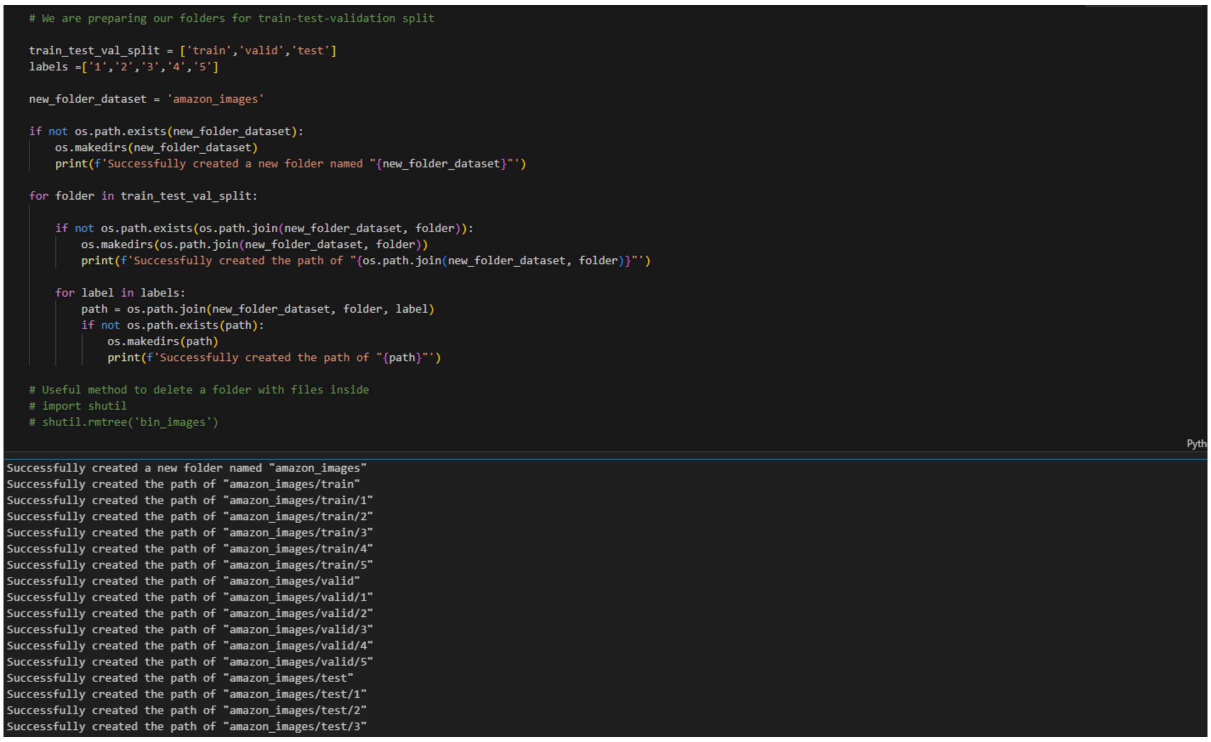
The dataset has 536,435 JPEG files in total in alignment with metadata files. Due to file naming rules, they can be divided into three categories:

- 1~4 digit: 1.jpg ~ 1200.jpg: 1200
- 5-digit: 00001.jpg ~ 99999.jpg: 99,999
- 6-digit: 100000.jpg ~ 535234.jpg: 435,235





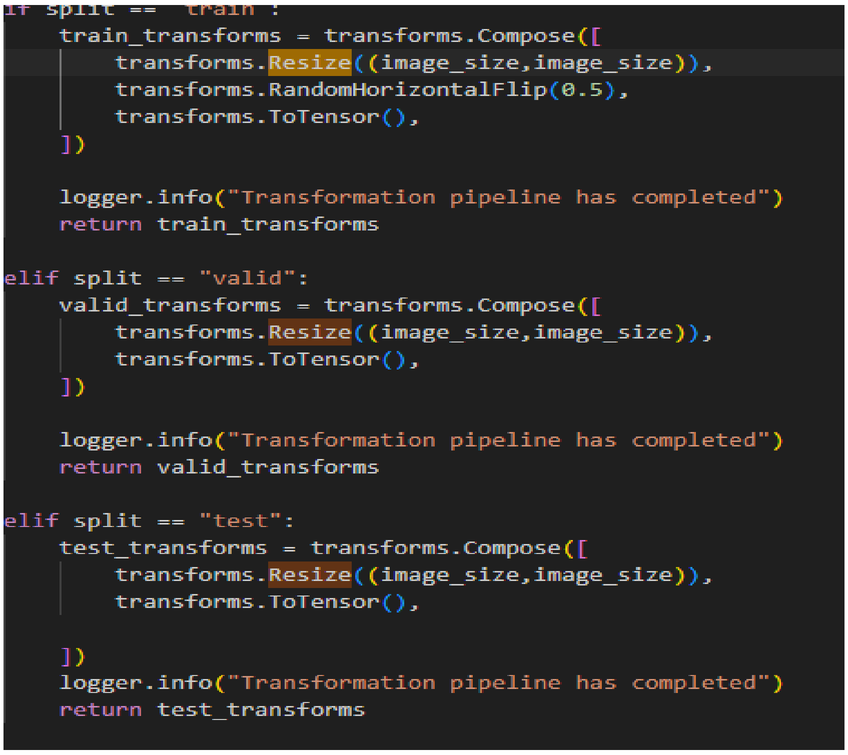
About Data Processing step, when finish download and observer the Dataset Image, we decided to devided dataset to 3 folder – there are, validation , train and test folder. We do that in this step for conduct trained dataset in training step. This ensures that the models can effectively train on the data set, evaluate the training process, compare the training results and the appropriate test results.



```
# We are preparing our folders for train-test-validation split

train_test_val_split = ['train','valid','test']
labels =['1','2','3','4','5']

new_folder_dataset = 'amazon_images'

if not os.path.exists(new_folder_dataset):
    os.makedirs(new_folder_dataset)
    print(f'Successfully created a new folder named "{new_folder_dataset}"')

for folder in train_test_val_split:

    if not os.path.exists(os.path.join(new_folder_dataset, folder)):
        os.makedirs(os.path.join(new_folder_dataset, folder))
        print(f'Successfully created the path of "{os.path.join(new_folder_dataset, folder)}"')

    for label in labels:
        path = os.path.join(new_folder_dataset, folder, label)
        if not os.path.exists(path):
            os.makedirs(path)
            print(f'Successfully created the path of "{path}"')

# Useful method to delete a folder with files inside
# import shutil
# shutil.rmtree('bin_images')
```

```
Successfully created a new folder named "amazon_images"
Successfully created the path of "amazon_images/train"
Successfully created the path of "amazon_images/train/1"
Successfully created the path of "amazon_images/train/2"
Successfully created the path of "amazon_images/train/3"
Successfully created the path of "amazon_images/train/4"
Successfully created the path of "amazon_images/train/5"
Successfully created the path of "amazon_images/valid"
Successfully created the path of "amazon_images/valid/1"
Successfully created the path of "amazon_images/valid/2"
Successfully created the path of "amazon_images/valid/3"
Successfully created the path of "amazon_images/valid/4"
Successfully created the path of "amazon_images/valid/5"
Successfully created the path of "amazon_images/test"
Successfully created the path of "amazon_images/test/1"
Successfully created the path of "amazon_images/test/2"
Successfully created the path of "amazon_images/test/3"
```

Besides, I also resized the image so I could train to fit the model



```
if split == "train":
    train_transforms = transforms.Compose([
        transforms.Resize((image_size,image_size)),
        transforms.RandomHorizontalFlip(0.5),
        transforms.ToTensor(),
    ])

    logger.info("Transformation pipeline has completed")
    return train_transforms

elif split == "valid":
    valid_transforms = transforms.Compose([
        transforms.Resize((image_size,image_size)),
        transforms.ToTensor(),
    ])

    logger.info("Transformation pipeline has completed")
    return valid_transforms

elif split == "test":
    test_transforms = transforms.Compose([
        transforms.Resize((image_size,image_size)),
        transforms.ToTensor(),

    ])
    logger.info("Transformation pipeline has completed")
    return test_transforms
```

All of this step can help other researchers to understand how the data was prepared for machine learning. we can ensure that the data is prepared in a consistent way, which can improve the accuracy of the machine learning results and we identify any potential problems with the data, which can be addressed before the machine learning algorithms are run.

3. **Problem Statement:**

Distribution centres often have robots which carry objects. These objects are present in bins, each bin contain 1-5 objects. The problem in this project is to count the number of items in the bin. We will create a model, which can take in a picture of a bin, and accurately return the number of objects present in that, we could solve & thus fully automate one crucial step in the Inventory Management process!

4. **Solution Statement:**

The identified problem can be solved using Image Classification . By used data to train the machine learning model, a model can be produced.

Components:

- Dataset Image Amazon Bin

- Deep learning Model

- Amazon Web Service

Pllatform:

1.. Sage Maker Studio – to train, tune and deploy the model.

2. S3 – Storage Bucket

5. **Refinement:**

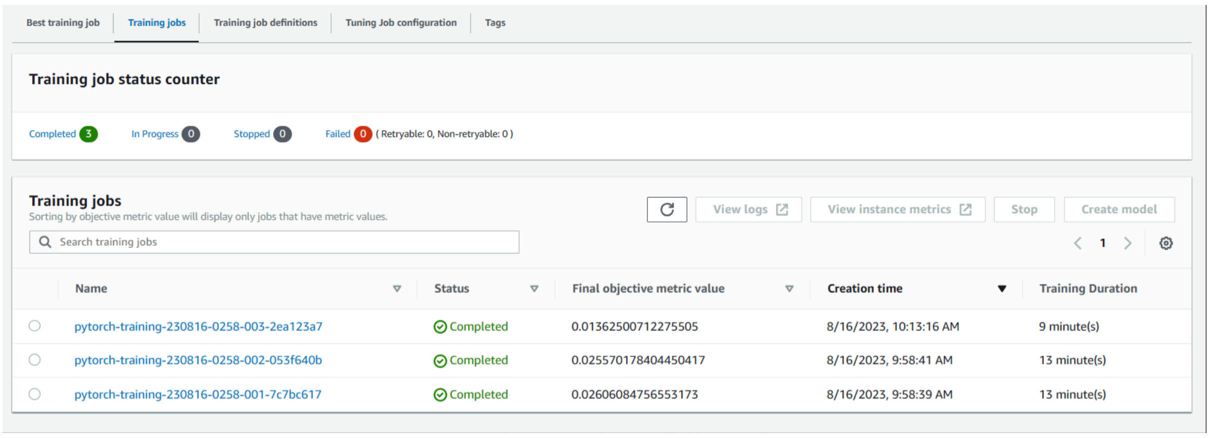
In this section, I will describe about the model I choose and how can I decide the best model for this dataset.

First I chosse all model pre-trained – transfer learning – for this project. They are maybe Resnet, VGG and MobileNet. The common point of these models is that they are lightweight, suitable for training with small data sets. Training time on these models can save a lot of time.

6. **Benchmark Model:**

First I chosse all model pre-trained – transfer learning – for this project. They are maybe Resnet, VGG and MobileNet. The common point of these models is that they are lightweight, suitable for training with small data sets. Training time on these models can save a lot of time.

The metric loss and hyperparameter of MobileNet you can following in this image below



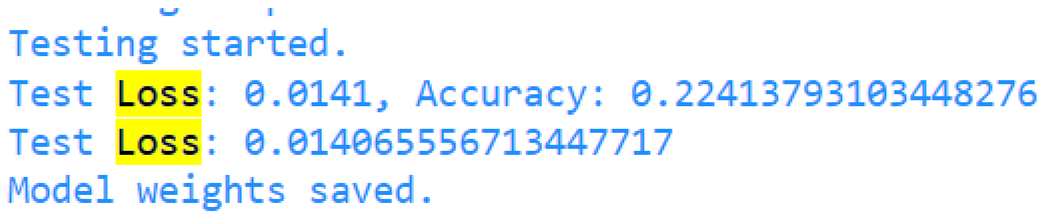
Training job status counter

Completed 3 | 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.

Name	Status	Final objective metric value	Creation time	Training Duration
pytorch-training-230816-0258-003-2ea123a7	Completed	0.01362500712275505	8/16/2023, 10:13:16 AM	9 minute(s)
pytorch-training-230816-0258-002-053f640b	Completed	0.025570178404450417	8/16/2023, 9:58:41 AM	13 minute(s)
pytorch-training-230816-0258-001-7c7bc617	Completed	0.02606084756553173	8/16/2023, 9:58:39 AM	13 minute(s)

And to compare to Resnet18 model you can following the image below.



```
Testing started.
Test Loss: 0.0141, Accuracy: 0.22413793103448276
Test Loss: 0.014065556713447717
Model weights saved.
```

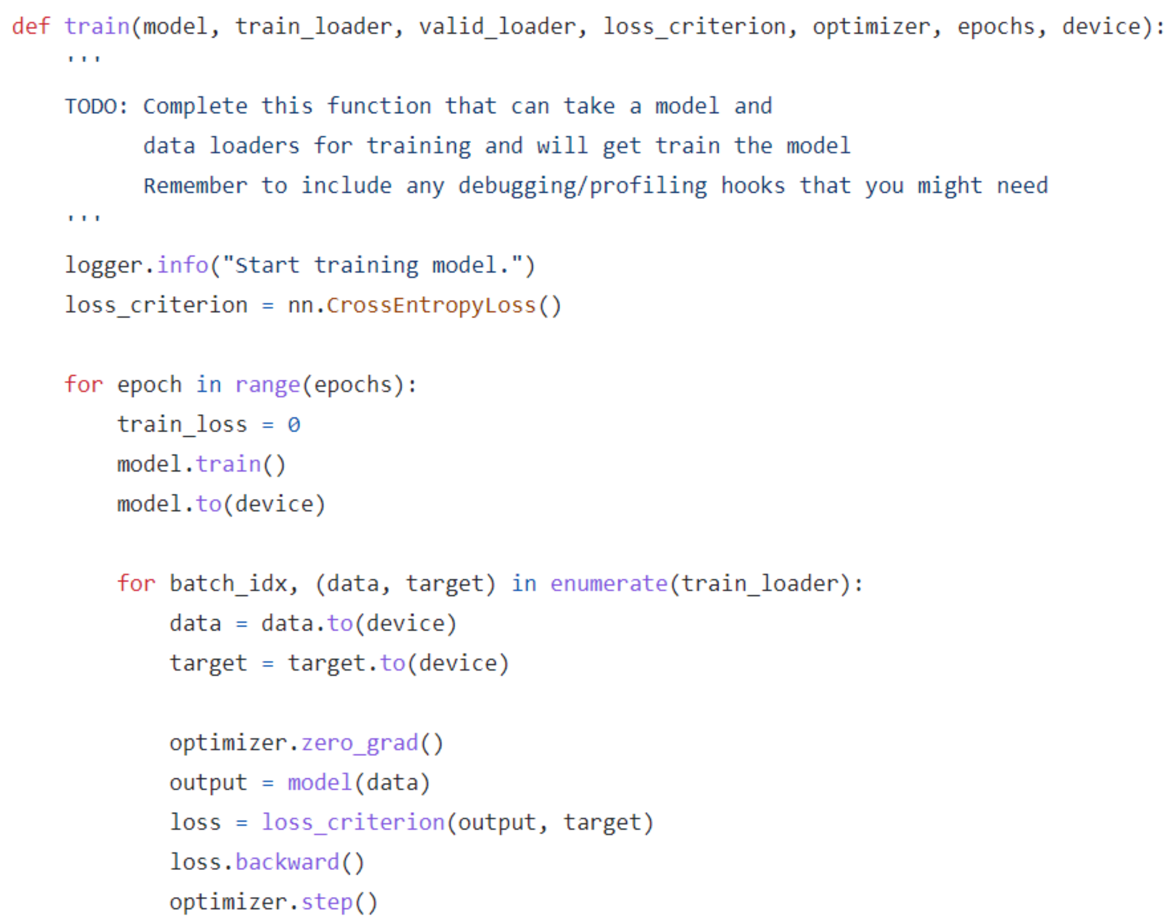
As you can see the result of two model tranning, you can see the Resnet Model is better than Mobilenet little bit, about accuracy and loss metric. The tranning time is short and two model have same time tranning.

7. Implementation

For this part, I will defined all step for implementation model this project. I will have one file to training job name train.py. For this file I will define all thing about training job like the model-pretrained I decide to chosse to training, training job and testing job. The model-pretrained I chosse here is Mobilenet and another is Resnet and I defined it in the net function.

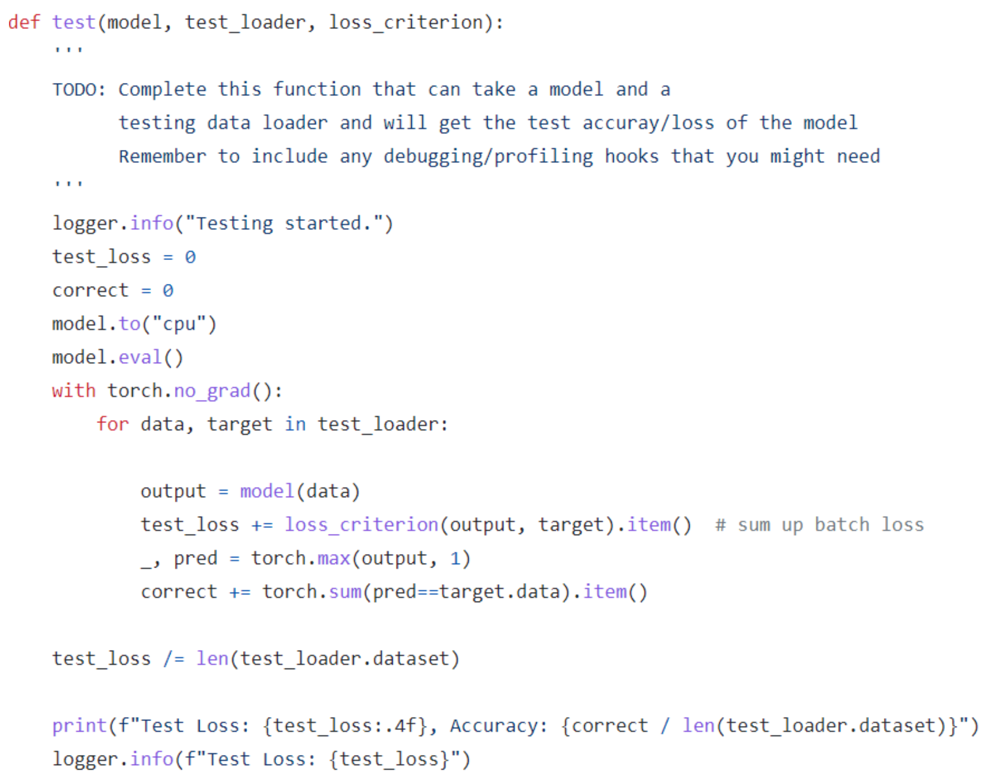
```python
def net(num_classes):
    '''
    TODO: Complete this function that initializes your model
          Remember to use a pretrained model
    '''
    logger.info("Start creating model.")
    model = models.resnet18(pretrained=True)

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

    num_features = model.fc.in_features
    model.fc = nn.Linear(num_features, num_classes)
    logger.info("Model has created.")
    return model
```

The next step I defined the traning job. The traning job have all defined about the number of epoch training, the input data for train, the input data for validation, in this step we used Pytorch Framework to help to define



```python
def train(model, train_loader, valid_loader, loss_criterion, optimizer, epochs, device):
    '''
    TODO: Complete this function that can take a model and
          data loaders for training and will get train the model
          Remember to include any debugging/profiling hooks that you might need
    '''
    logger.info("Start training model.")
    loss_criterion = nn.CrossEntropyLoss()

    for epoch in range(epochs):
        train_loss = 0
        model.train()
        model.to(device)

        for batch_idx, (data, target) in enumerate(train_loader):
            data = data.to(device)
            target = target.to(device)

            optimizer.zero_grad()
            output = model(data)
            loss = loss_criterion(output, target)
            loss.backward()
            optimizer.step()
```

The samething can do for testing job, you can refer in the image below:



```python
def test(model, test_loader, loss_criterion):
    '''
    TODO: Complete this function that can take a model and a
          testing data loader and will get the test accuray/loss of the model
          Remember to include any debugging/profiling hooks that you might need
    '''
    logger.info("Testing started.")
    test_loss = 0
    correct = 0
    model.to("cpu")
    model.eval()
    with torch.no_grad():
        for data, target in test_loader:

            output = model(data)
            test_loss += loss_criterion(output, target).item()  # sum up batch loss
            _, pred = torch.max(output, 1)
            correct += torch.sum(pred==target.data).item()

    test_loss /= len(test_loader.dataset)

    print(f"Test Loss: {test_loss:.4f}, Accuracy: {correct / len(test_loader.dataset)}")
    logger.info(f"Test Loss: {test_loss}")
```

8. **Refinement:**

In data processing part, I devided the data to 3 folder to conduct training. The models I chosse are all lightweight model so the training time is short and I can train model quickly and so that I have manytime to compare the result of model. When conduct training time I use the hyperparameter. The hyperparameter have learning rate, batch size, epoch. In the first time I choose the learning rate is low, I see the result is good and the accuracy is high, but the training time is long, same for number of epoch, but when I conduct to finetune and training again, the result don't have any different and I choose the best parameter – you can see in the ipynb file. About the decide I chosse the model-pretrained, I see the result of metric loss in MobileNet near the same with Resnet18, but Resnet18 have a little bit short training time.

9. **Evaluation Metrics:**

There is a classification problem, the overall accuracy of the classification can be used to evaluate the performance of the trained model.

10. **Dataset or the Image Source:**

The Amazon Bin Image Dataset is to be used to train the model.

The dataset used in this problem is the Amazon Bin Image Dataset. This dataset have 500,000 images of bins and the data have one or more objects present in it. One of each image contains information about the image, like the number of objects it has, the dimensions and type of objects. For our problem statement, we only need the total count of objects in the imageThese are some typical images in the dataset.

```
Number of images with the label of "4": 2373
Number of images with the label of "2": 2299
Number of images with the label of "1": 1228
Number of images with the label of "5": 1875
Number of images with the label of "3": 2666

Number of images in total: 10441
```

The, the data loaders are created at a batch size of 32 for training, testing and validation.

11 **Algorithm:**

- The images is used to train the model, that could learn the parameters to perform classification based on the number of items in the bin.

- Deep Learning Framework – PyTorch

- A corresponding Sage Maker instance will be created, and data will be fed from the S3 bucket.

- The model is also tuned to find out the best hyper-parameters as a model improvement method.

- A pre-trained model ResNet18 is used in this project.

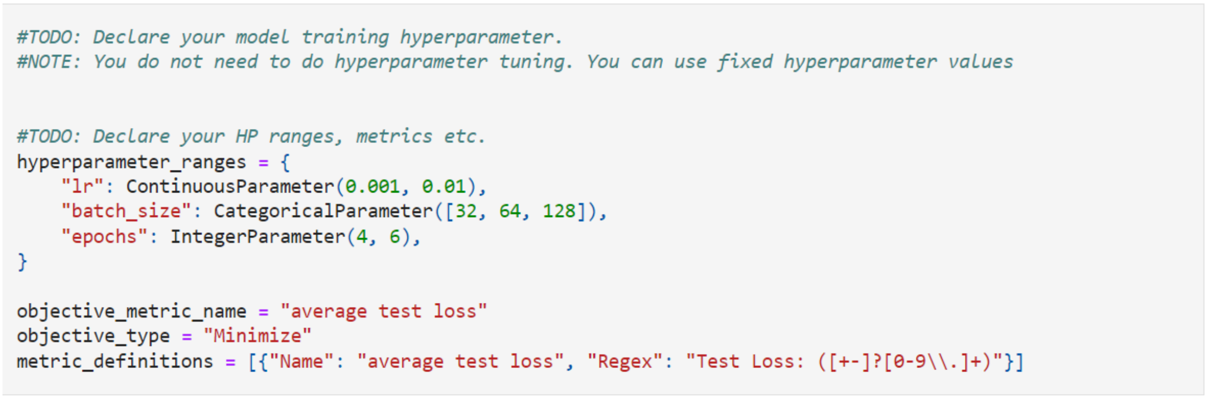
- Hyper-parameter Tunning:

The following hyper-parameters are identified for tunning.

- Learning Rate

- Batch Size

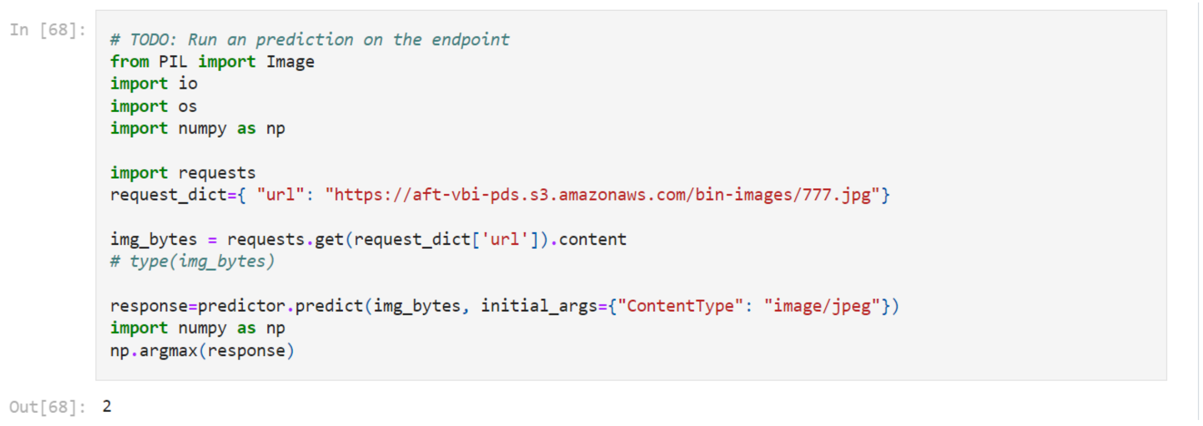
- Epoch The Learning rate for the Adam optimizer is identified as one of the hyper-parameters which could help improve the training process rather than the pre-defined default.



```
#TODO: Declare your model training hyperparameter.
#NOTE: You do not need to do hyperparameter tuning. You can use fixed hyperparameter values


#TODO: Declare your HP ranges, metrics etc.
hyperparameter_ranges = {
    "lr": ContinuousParameter(0.001, 0.01),
    "batch_size": CategoricalParameter([32, 64, 128]),
    "epochs": IntegerParameter(4, 6),
}

objective_metric_name = "average test loss"
objective_type = "Minimize"
metric_definitions = [{"Name": "average test loss", "Regex": "Test Loss: ([+-]?[0-9\\.]+)"}]
```

9. Model Evaluation:

Finnaly we can used the model is deployed like an endpoint to predict the Image to see the result and so that we can evaluation the accuracy



```
In [68]: # TODO: Run an prediction on the endpoint
         from PIL import Image
         import io
         import os
         import numpy as np

         import requests
         request_dict={ "url": "https://aft-vbi-pds.s3.amazonaws.com/bin-images/777.jpg"}

         img_bytes = requests.get(request_dict['url']).content
         # type(img_bytes)

         response=predictor.predict(img_bytes, initial_args={"ContentType": "image/jpeg"})
         import numpy as np
         np.argmax(response)

Out[68]: 2
```