AWS Machine Learning Engineer Nanodegree Capstone Project

By Araz Sharma 22nd February 2022

I. DEFINITION

Project Overview:

This Project is on **"Inventory Monitoring at Distribution Centres"**.
This problem has originated from the popular field of Robotics, and its applications in the industry, like that of Inventory Monitoring and Supply Chain Functions. A lot of Corporations, which handle physical cargo and deal with supply chain of any kind of goods have tried to bring in automation to make these processes more efficient and accurate. A great example of this is Amazon, who is one of the biggest hubs of delivery of all kinds of goods. These goods are often stored in big warehouses. Since the quantity of these items are in a huge amount, to physically do inventory monitoring would require both large and intelligent human resources, which are both expensive and prone to errors.

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.

Problem Statement:

As mentioned above in the domain, distribution centres often have robots which carry objects. These objects are present in bins, and for our problem, each bin can contain 1-5 objects.

The problem we aim to tackle in this project is to count the number of items present in the bin. This is a **Classification task**, of classifying number of items in 1 – 5, given an input image. This is a worthwhile problem to solve, for it has immense real world applications. If we can develop 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!

Datasets and Input:

The dataset used in this problem is the open source Amazon Bin Image Dataset. This dataset has 500,000 images of bins containing one or more objects present in it. Corresponding to each image, is a metadata file, which 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 image. An example of an image & the corresponding metadata file is shown as below: (Source **Reference [1]**)

The “EXPECTED_QUANTITY” field tells us the total number of objects in image.


```
{
    "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"
}
```

Solution Strategy:

To solve our problem statement, we will use the task of Computer Vision, to come up with a Machine Learning Model, which given an image from our dataset, can identify the number of objects present in it. Essentially, we would be using **Multi-Class Image Classification**, with Number of Objects from 1-5 as an individual class.

To do this, we can make use of **Convolutional Neural Networks**, which are a State of the Art technique for Image Recognition tasks. We will leverage Pre-trained models with Transfer Learning to solve the problem.

The end solution should be a model, which can take in an input image from the Amazon Bin Image dataset, and accurately output the number of objects it thinks are present in that image (from 1-5).

Metrics for Evaluation:

To evaluate our model, we need some good metrics, which align with the problem statement. The metrics must be mathematically sound and we should be able to optimise our model for them.

Since we have a Classification Task, we can use Accuracy, Recall, Precision and F1 scores as our metrics. These can be for the overall data, and also class-wise, to identify if a model is doing better on a particular class, or has a high bias for one of them.

Since we have Multi-Class classification, the definition of Precision & Recall must be clearly understood. Let's assume we have 3 classes: A, B & C.

With Multi-Class classification, we have precision and recall for each individual class. For example, metrics for Class A will be defined as follows:

Precision: (‘#’ stands for Number of)

#Correctly Predicted Class A Instance / #All Instance predicted as Class A

Recall: (‘#’ stands for Number of)

#Correctly Predicted Class A Instances / #Total Class A Instances

F1 Score: 2*Precision*Recall / (Precision + Recall)

So to evaluate our model, we will see per class metrics of Precision, Recall and F1 Score, and see the overall accuracy of the model, which includes all classes, and is defined as:

Accuracy: #Total Correctly Predicted Instances / #Total Instances

Reference [2] for Precision, Recall and F1 for multi-class classification

II. Analysis

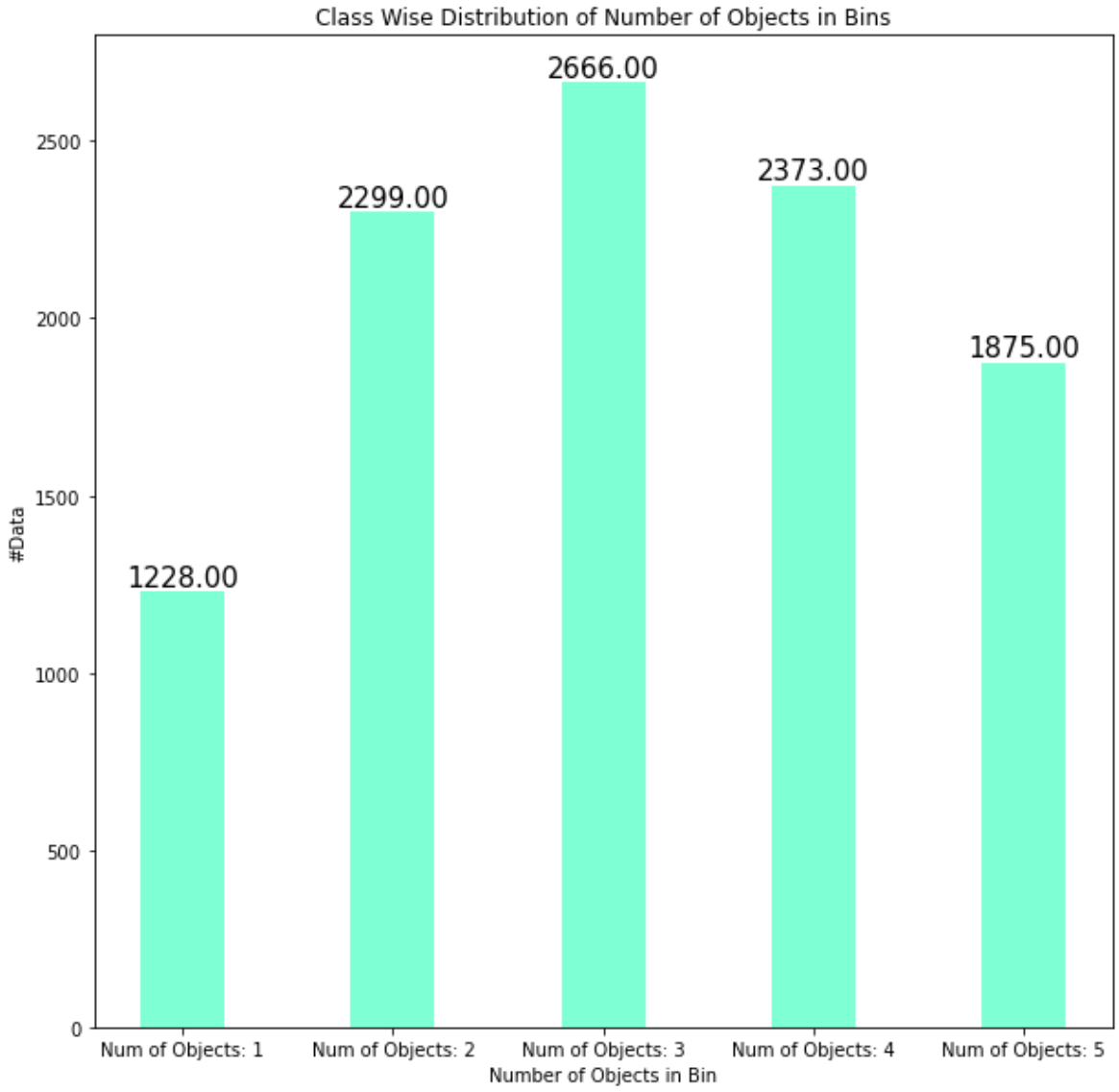
Data & Distribution Exploration:

The Notebook ‘Create_Data_Capstone.ipynb’ implements Data Exploration & Visualisation for our Amazon Bin Image Dataset.

The ‘file_list.json’ provided by Udacity, has a subset of the Amazon Bin Image Dataset.

This subset of data has 5 classes, corresponding to number of objects present in the bin: 1, 2, 3, 4 & 5. The total number of images in this subset are: 10,441

The Class Wise Distribution is as follows: (Plotted with code)

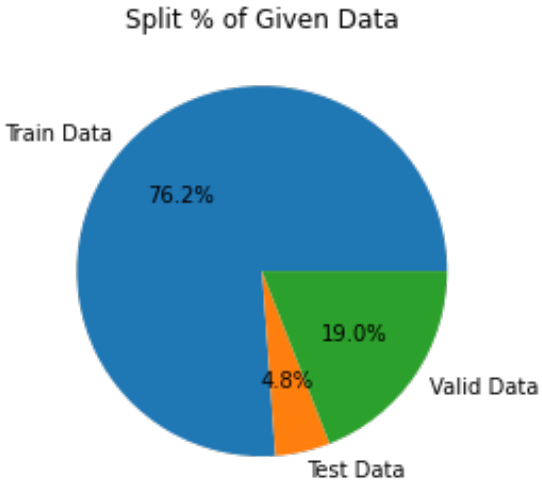


From this subset, we took a Train-Test-Validation Split as follows:

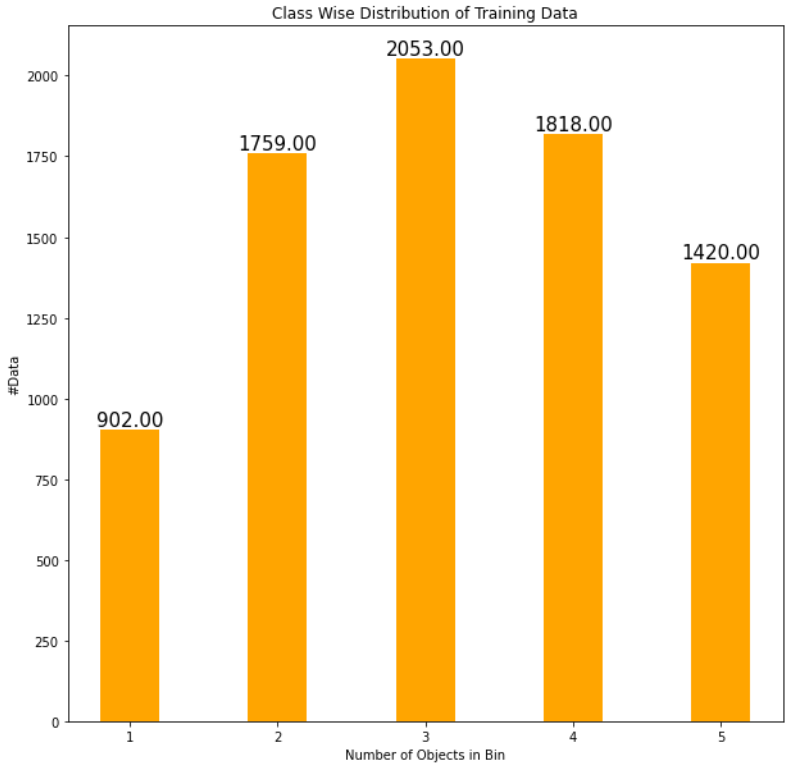
1. We randomly sample 100 Images from each class as Test Data

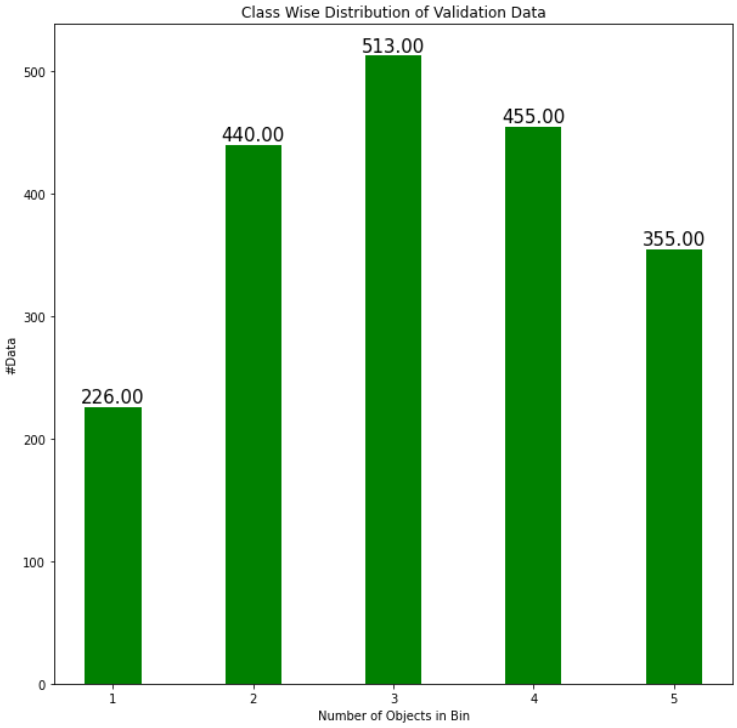
2. For the remaining data, we do a 80-20 Train-Validation Stratified Split. Stratification is important to preserve the same distributions in both sets.

After this split we see:



We also explore the Class Wise Distribution in Training and Validation Sets:



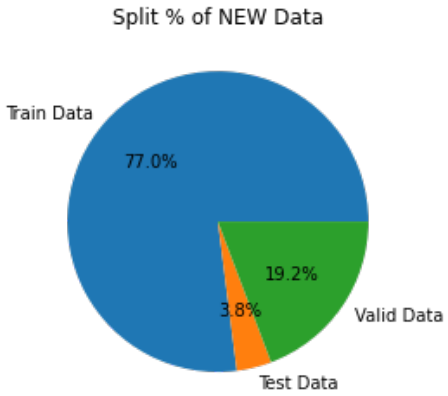


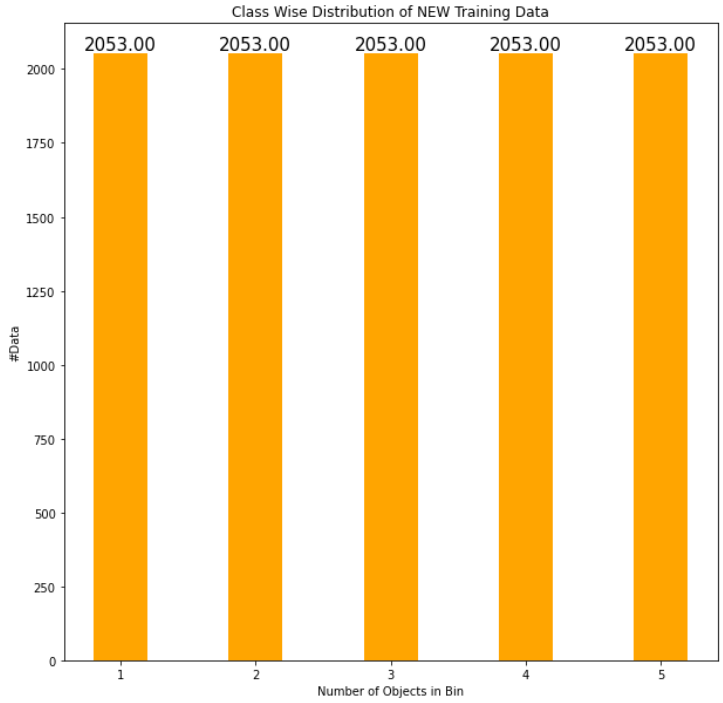
However, we see a big imbalance in Class Distribution! From the Above Distributions, we see a big difference in the Data in Class 3 vs All other classes

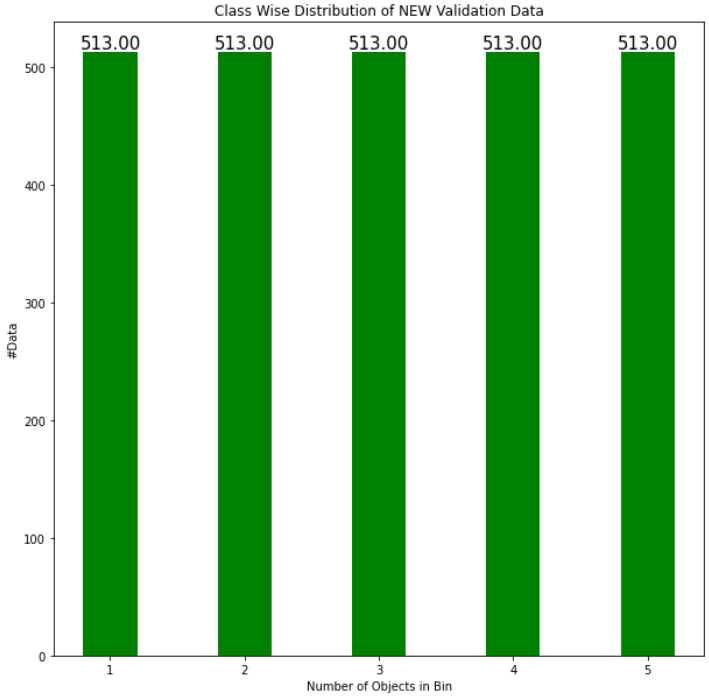
Since there is an Imbalance of Classes, there is a high chance for a model to learn to only predict Majority Class. We need to test this data with a basic model, to see if the model is able to learn other classes or not. So we fine-tuned a pre-trained Resnet50 model for 5 epochs and observed:

1. Imbalance is leading to Model only predicting Class 3
2. Loss decreases since Class 3 is dominating, but final test accuracy is poor, since all classes balanced (100 each)
3. We need to balance out the data

Using the 'Get_More_Data_Script.ipynb' we balance out the data (check in Create_Data_Capstone for implementation details). The new distributions are:







Now the dataset class distribution is not biased towards a single class. This dataset is now used for Model Training and Evaluation. Further transformations of data are covered in data pre-processing steps ahead.

Visualizations:

In the Notebook, we have taken 3 examples per class to check for:

1. To validate that Image has that many objects as the class.
2. Try to analyse to see what kind of transformations can be applied here.
3. Can we as humans identify them with ease? If this is a difficult task for Humans itself to analyse, then Models may not perform as well as we expect them to.

Class 1: One Object in the Bucket

- We see that Class 1 Images are Valid
- **They seem easy to identify for Humans**
- The Bin has a net, which may make it difficult to see the objects sometimes

Class 2: Two Objects in the Bucket

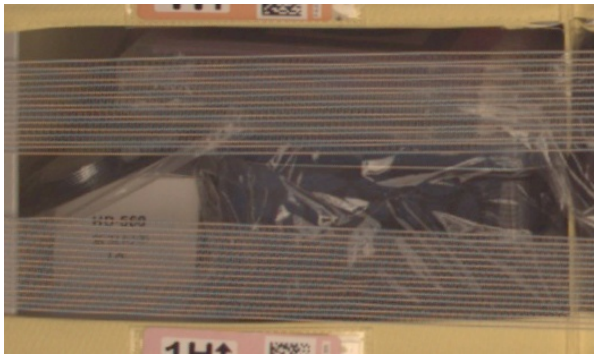
- We see that Class 2 Images are Valid
- **There are cases when 2 Objects are very close to each other and it seems they are one object**
- The Bin has a net, which may make it difficult to see the objects sometimes

Class 3: Three Objects in the Bucket

- After carefully scrutinizing the Images, we see Class 3 Images are Valid
- **They are difficult to identify for even Humans, and Model may not perform well on this Class**
- The Bin has a net, which may make it difficult to see the objects sometimes

Class 4: Four Objects in the Bucket

- Needed to check images very carefully, and had to inspect more images to confirm that the Class 4 Images are Valid
- **They are not easy to verify by Humans, and Model may not perform well on them**
- The Bin has a net, which may make it difficult to see the objects sometimes



Class 5: Five Objects in the Bucket

- Images appear very cluttered with objects and it is difficult to make out if there are 5 objects always
- **Extremely hard to classify by Humans**
- The Bin has a net, which may make it difficult to see the objects sometimes

- Here there's a possibility **Model may directly learn for very cluttered to predict 5, but not actually from that it detects 5 separate objects**

From all pictures, we see that we can resize them and horizontally flip them.

Benchmark:

As discussed in the Proposal, since this particular task has no implementations, we created 2 Benchmarks, as implemented and showed in the Benchmark_Model.ipynb notebook.

Benchmark I: Custom CNN Architecture
Batch Size 32 and Learning Rate: 0.005 Epochs = 5

```
# Custom CNN Architecture for Benchmark
class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv4 = nn.Conv2d(64, 128, 3, padding=1)
        self.conv5 = nn.Conv2d(128, 256, 3, padding=1)
        self.pool = nn.MaxPool2d(2,2)
        self.fc1 = nn.Linear(14*14*256, 2048)
        self.fc2 = nn.Linear(2048, 1024)
        self.fc3 = nn.Linear(1024, 128)
        self.fc4 = nn.Linear(128, 5)


    def forward(self, x):

        x = F.relu(self.conv1(x))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.conv4(x)))
        x = self.pool(F.relu(self.conv5(x)))

        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)
        return x
```

Since we had limited amount of resources from the Udacity AWS Gateway, the resource constraints made training a CNN from scratch difficult & posed memory and cost issues, and the Benchmark wasn't very good –

Class Wise Metrics

Class	Precision	Recall	F1 Score
1	0.2	1	0.33
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0

Overall Test Set Accuracy = 0.2 (20%) ¶

So we also trained a second model for a Benchmark

Benchmark II: Finetuning a Pre-Trained Resnet50 CNN Architecture
Hyper Parameters: Batch Size 64 and Learning Rate: 0.001 Epochs = 5

This resulted in a good Benchmark we wanted to beat:

Class Wise Metrics

Class	Precision	Recall	F1 Score
1	0.432	0.86	0.575
2	0.294	0.23	0.258
3	0.180	0.13	0.151
4	0.303	0.1	0.150
5	0.381	0.45	0.412

Overall Test Set Accuracy = 0.354 (35.4%)

Algorithms and Techniques Planned to Use:

1. **Fine Tuning a Pre-Trained Resnet50 CNN Model with Hyper Parameters from Intuition** – Since we know that this is a Multi-Class Image Classification, leveraging transfer learning can help produce good results

2. **Using Hyper Parameter Tuner in SageMaker** – We can leverage SageMaker's HP Tuner too find good parameters to train with

3. **Training Model with Best HP from Tuner** – Once we get the best parameters, we can train on them for a longer duration

III. METHODOLOGY:

Data Pre-Processing:

Once the data is uploaded to S3 Buckets, the Data Loaders help in fetching them, and we can do the transformations as seen fit from Data Analysis with them.

We apply the following Transformations (Training) as seen from the code:

- Resize Image to 224 x 224
- Apply Random Horizontal Flips
- Convert to Tensor
- Shuffle before feeding in Model

```
def create_data_loaders(data, batch_size):
    train_data_path = os.path.join(data, 'train_data')
    test_data_path = os.path.join(data, 'test_data')
    validation_data_path=os.path.join(data, 'valid_data')

    train_transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        ])

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

    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=True)

    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=True)

    return train_data_loader, test_data_loader, validation_data_loader
```

Implementation:

'Final_Model_Train_Inference.ipynb' notebook has the detailed implementation, which is well documented, for the Algorithms, Techniques and Stand out Suggestions tried for the Project.

The overall Implementation was consisting of designing and writing the code for the Algorithms and Techniques, and using SageMaker's Library to use them to train, test and deploy the Models. This was also an iterative process, since any errors in the code would lead to a failed training job, which had to be debugged from the CloudWatch Logs.

Steps for Implementation:

I Implementing Training and Evaluation Code in model_train.py
- Loss and Optimiser to use
- Data Loaders and Transformations
- Metrics for Evaluation
- Enable GPU Training

II Deciding First Model & Parameters to Train

III Refinement: Hyperparameters Search

IV Refinement: Training Model with Best Hyperparameters

V Generating Profiler Report

VI Deploying Endpoint

VII Inference for each Class

I. Implementing Training and Evaluation Code in model_train.py

- We used the CrossEntropyLoss and the Adam Optimiser
- The Data Loader is defined as shown in Data Pre-processing
- The Metrics defined in definition are coded
- GPU support is added in the entire code

II. Deciding First Model & Parameters to Train

To begin, we try fine tuning a Pre-Trained Resnet50 Model, as defined in model_train.py on our Data
From intuition we try the following Hyper Parameters:

- Batch Size: 64
- Learning Rate: 0.005
- Number of Epochs: 5

```python
def net():
    model = models.resnet50(pretrained=True) #Using a Resnet 50 Pre-Trained Model

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

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

THIS IS THE MODEL WE ARE USING IN ALL THE STEPS

III. Refinement: Hyperparameters Search on Model

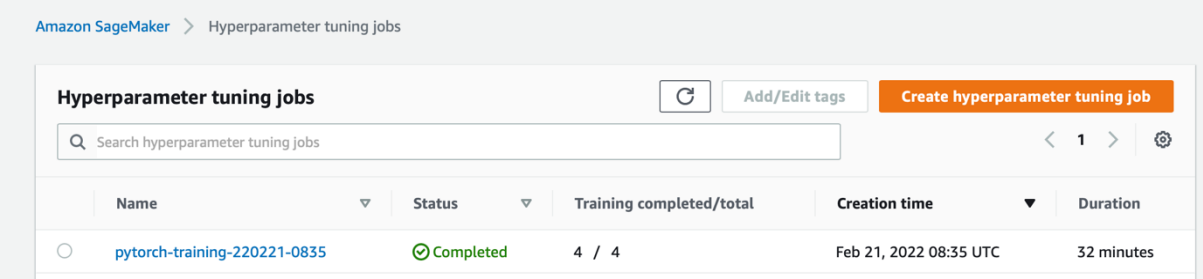
We search for the Batch Size and Learning Rate, as these Hyper Parameters are the most difficult to choose manually

In [3]:
```python
hyperparameter_ranges = {
    "learning_rate": ContinuousParameter(0.001, 0.1),
    "batch_size": CategoricalParameter([16, 32, 64, 128, 256]),
}

role = sagemaker.get_execution_role()

objective_metric_name = "Valid Loss"
objective_type = "Minimize"
metric_definitions = [{"Name": "Valid Loss", "Regex": "Final Validation Loss: ([0-9\\.]+)"}]
```

We launch 4 HP Jobs, with 2 Jobs in Parallel, so the Tuner can learn from the first 2 Jobs, to try better Hyper Params in the next 2 Jobs

In [6]:
```python
estimator = PyTorch(
    entry_point="model_train.py",
    base_job_name='test_bins',
    role=role,
    framework_version="1.4.0",
    instance_count=1,
    instance_type="ml.p3.2xlarge",
    py_version='py3'
)

tuner = HyperparameterTuner(
    estimator,
    objective_metric_name,
    hyperparameter_ranges,
    metric_definitions,
    max_jobs=4,
    max_parallel_jobs=2,
    objective_type=objective_type
)
```

We launch the Jobs as shown here:



The Jobs succeed in telling the best HP: Batch Size 32, LR: 0.002

The HP Search has Succeeded!

We see that the Model first tried Batch sizes of 128 & 64, and since 64 gave a better result, it tried smaller batch sizes next

Note: The Objective Metric is the Final Validation Loss

In [8]:
```
from sagemaker.analytics import HyperparameterTuningJobAnalytics

exp = HyperparameterTuningJobAnalytics(
  hyperparameter_tuning_job_name='pytorch-training-220221-0835')

jobs = exp.dataframe()

jobs.sort_values('FinalObjectiveValue', ascending=0)
```

Out[8]:

	batch_size	learning_rate	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTime	TrainingElapsedTimeSeconds
3	"128"	0.059663	pytorch-training-220221-0835-001-61ba29e9	Completed	196.0	2022-02-21 08:38:19+00:00	2022-02-21 08:49:44+00:00	685.0
2	"64"	0.007783	pytorch-training-220221-0835-002-a00a4de7	Completed	88.0	2022-02-21 08:37:21+00:00	2022-02-21 08:49:45+00:00	744.0
0	"64"	0.001545	pytorch-training-220221-0835-004-7dba57e9	Completed	87.0	2022-02-21 08:51:59+00:00	2022-02-21 09:04:24+00:00	745.0
1	"32"	0.001948	pytorch-training-220221-0835-003-c9d3e5a8	Completed	44.0	2022-02-21 08:54:34+00:00	2022-02-21 09:06:05+00:00	691.0

IV. Refinement: Training Model with Best Hyperparameters

We perform training on the model, with the best HP found above, for 10 epochs. We also add debugger rules and hooks to the Model_Train.py code to get the Profiler Output and so we can learn from the Profiler Report.

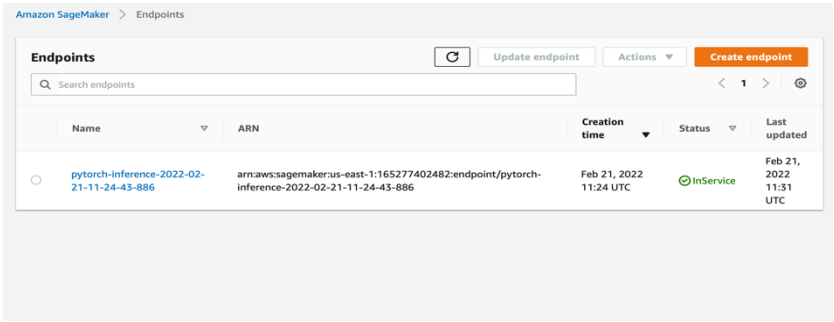
V. Generating Profiler Report

We see that the Low GPU Utilisation and Batch Size Rules were triggered.

	Description	Recommendation	Number of times rule triggered	Number of datapoints	Rule parameters
LowGPUUtilization	Checks if the GPU utilization is low or fluctuating. This can happen due to bottlenecks, blocking calls for synchronizations, or a small batch size.	Check if there are bottlenecks, minimize blocking calls, change distributed training strategy, or increase the batch size.	12	2384	threshold_p95:70 threshold_p5:10 window:500 patience:1000
BatchSize	Checks if GPUs are underutilized because the batch size is too small. To detect this problem, the rule analyzes the average GPU memory footprint, the CPU and the GPU utilization.	The batch size is too small, and GPUs are underutilized. Consider running on a smaller instance type or increasing the batch size.	12	2383	cpu_threshold_p95:70 gpu_threshold_p95:70 gpu_memory_threshold_p95:70 patience:1000 window:500

VI. Deploying Endpoint

We deploy an endpoint which can be used for Inference



VII. Inference for Each Class

We use new images for each class, to test from the endpoint, and see the results and reasons for them being classified correctly or not. Check 'Final_Model_Train_Inference.ipynb' for the detailed results.

Refinement Techniques:

For our Project, we do refinement for results in 2 cases:

1. **Correcting Imbalance of Classes in Dataset:** Since the dataset was imbalanced, it would lead to model predicting the majority class. To avoid that, we wrote a script to fetch new data for Classes 1, 2, 4 and 5 & balanced out the data, for refinement in predictions

2. **From Intuition to Hyper Parameter Search:** We started by guessing hyper parameters from Intuition. Then to refine our results, we used Hyper Parameter Tuner search, to find the good HPs and use them to train the final model.

Initial Solution from Intuition:

Class Wise Metrics

Class	Precision	Recall	F1 Score
1	0.690	0.49	0.573
2	0.301	0.35	0.324
3	0.219	0.16	0.184
4	0.297	0.7	0.417
5	0.2	0.01	0.019

Overall Test Set Accuracy = 0.342 (34.2%)

Class Wise Metrics

Class	Precision	Recall	F1 Score
1	0.615	0.72	0.663
2	0.361	0.26	0.302
3	0.254	0.14	0.180
4	0.258	0.31	0.281
5	0.419	0.57	0.483

Overall Test Set Accuracy = 0.4 (40%)

THUS WE SEE AN IMPROVEMENT IN OUR SOLUTION

IV. RESULTS:

Model Evaluation & Validation:

The Metrics we determined for evaluating our Models were:

1. Final Test Accuracy
2. Class wise Precision
3. Class wise Recall
4. Class wise F1 Score

Our Final Model is a pre-trained Resnet50 (as shown below) fine-tuned for 10 epochs with Batch Size 32 and Learning Rate 0.002

```
def net():
    model = models.resnet50(pretrained=True) #Using a Resnet 50 Pre-Trained Model

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

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

The Metrics for the Final Model are:

Class Wise Metrics

Class	Precision	Recall	F1 Score
1	0.615	0.72	0.663
2	0.361	0.26	0.302
3	0.254	0.14	0.180
4	0.258	0.31	0.281
5	0.419	0.57	0.483

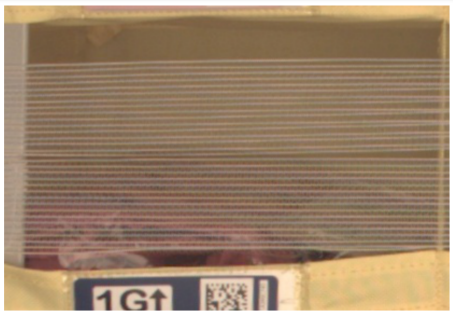
Overall Test Set Accuracy = 0.4 (40%)

According to our Metrics:

1. Model has the highest F1 score on Class 1, so it would perform the best in Inference for Class 1.

2. This holds true from the Inference results we have in 'Final_Model_Train_Inference.ipynb', where model gets 6/10 Images correct for Class 1, as shown below

In [63]:
```
amazon_bin_data = "https://aft-vbi-pds.s3.amazonaws.com/bin-images/"
print("CLASS 1 IMAGES")
class_1_correct = 0
for i in range(10): #Since we have 10 Pictures for Inference
    request_dict={ "url": amazon_bin_data + Inference_Images[1][i]}
    img_bytes = requests.get(request_dict['url']).content
    display(Image.open(io.BytesIO(img_bytes))) #Display Inference Picture
    response=predictor.predict(img_bytes, initial_args={"ContentType": "image/jpeg"})
    class_predicted = np.argmax(response, 1)[0]
    print("CLASS PREDICTED AS: ",class_predicted + 1)
    if(class_predicted==0):
        class_1_correct+=1
```



```
CLASS PREDICTED AS:  1
```

In [64]:
```
print(f"Model got {class_1_correct}/10 Images right")
```

```
Model got 6/10 Images right
```

3. This also makes sense according to the Data Analysis we had done, where we expected the model to do well on this class, as humans had done well

4. Model has the lowest F1 score on Class 3, so it should perform the worst in Inference for Class 3

5. Even in Data Analysis phase, we saw that Class 3 Images were hard for Humans also to classify, and further obstacles like the net of the bin, dim lighting conditions, heavy packaging and objects behind one another made it seem difficult for the model to perform well

6. This holds true from the Inference results we have in ‘Final_Model_Train_Inference.ipynb’, where model gets 2/10 Images correct for Class 3, as shown below

Checking for Class 3 Images

In [69]:
```
amazon_bin_data = "https://aft-vbi-pds.s3.amazonaws.com/bin-images/"
print("CLASS 3 IMAGES")
class_3_correct = 0
for i in range(10): #Since we have 10 Pictures for Inference
    request_dict={ "url": amazon_bin_data + Inference_Images[3][i]}
    img_bytes = requests.get(request_dict['url']).content
    display(Image.open(io.BytesIO(img_bytes))) #Display Inference Picture
    response=predictor.predict(img_bytes, initial_args={"ContentType": "image/jpeg"})
    class_predicted = np.argmax(response, 1)[0]
    print("CLASS PREDICTED AS: ",class_predicted + 1)
    if(class_predicted==2):
        class_3_correct+=1
```

```
CLASS PREDICTED AS:  4
```

In [70]:
```
print(f"Model got {class_3_correct}/10 Images right")
```

```
Model got 2/10 Images right
```

From the Inference and Metrics, we see that the Model is able to identify Classes 1 and 5 well, is decent for Class 2, but not doing well for Classes 3 & 4

Justification with Benchmark:

For our Analysis, since F1 score is the combination of Precision and Recall (both with equal weights), we can compare two models based on the Final Test Accuracy and the Class wise F1 scores.

If Model A has a higher Test Accuracy than Model B and Majority of Classes in Model A have a higher F1 score than Classes in Model B, then we can declare that Model A is better than Model B

So now we compare the metrics of:

- Benchmark Model, a Fine Tuned Resnet50 for Batch Size of 64 and Learning Rate 0.001, trained for 5 epochs

- Final Model, a Fine Tuned Resnet50, but different hyper params with Batch size 32, Learning Rate 0.002 and trained for 10 Epochs

The Metrics for our Best Benchmark vs. Best Model:

Class	F1 for Benchmark	F1 for Best Model
1	0.575	0.663
2	0.258	0.302
3	0.151	0.180
4	0.150	0.281
5	0.412	0.483

Accuracy for Benchmark	Accuracy for Best Model	% Increase in Accuracy
0.354 = 35.4%	0.4 = 40%	0.046 = 4.6%

Thus, we can justify that the Model we trained beat the Benchmark we had. We can also say that the trained Model is robust for Images of Class 1 and 5, but not so much for others.

The solution we wanted was to have a model, which when fed Image input is able to accurately classify the number of objects in them (from 1 – 5). We have met this solution, in the limits of the performance defined by our benchmarks.

V. CONCLUSION:

Free Form Visualization:

One important takeaway from this problem is from the Dataset we have. Analyzing the Amazon Image Bin Dataset for 1 – 5 number of objects, we find that **most of the images are taken from the front of the bin**. This means that a lot of times even humans cannot accurately tell the number of objects in the bin, **as we count well from the top view.** So, this shows that this task is a complex one, and a lot of data would be needed to create a model robust enough to perform with a State-of-the-Art accuracy!



Reflection:

In this project, we have accomplished & learnt the following things:

1. Learnt about “Inventory Monitoring at Distribution Centres”, and the domain behind this
2. Learnt about the Amazon Bin Image Dataset and its properties
3. Learnt how to use boto3 and AWS offered libraries to access S3 with code to download or upload data
4. Learnt how to visualise the distributions of data
5. Implemented a Train-Test-Validation split that is stratified
6. Learnt how to identify Class Imbalance
7. Learnt how class imbalance can bias a model to predict majority class
8. Learnt how to write a script to check against meta data and download more training data to balance classes
9. Learnt how to analyze image data and get insights on the classes
10. Learnt how to create your own Benchmarks
11. Implemented a custom CNN Architecture and trained it from scratch
12. Learnt how to fine tune a pre-trained model
13. Implemented coding metrics for a Multi-Class Classification task
14. Implemented training a model on AWS SageMaker
15. Implemented training on SageMaker with a GPU instance
16. Implemented Hyper Parameter Training Jobs and learnt how to get better Hyper Params in SageMaker with Multiple HP Jobs
17. Implemented hooks in training code and creating rules for debugger
18. Generating a Profiler Debugger Report
19. Deployed a Model on an Endpoint and perform Inference from that
20. Created a Model which beat the Benchmark and can be used for inferencing with a decent accuracy

Thoughts on Improvement:

- The model can be improved with more training on bigger resources
- The model can be improved with more input data from Dataset
- We can try more complex architectures, with more resources to learn more features from inputs, and improve performance

VI. References

[1] https://github.com/awslabs/open-data-docs/tree/main/docs/aft-vbi-pds
[2] https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2