

# **AWS Machine Learning Engineer Nanodegree**

## **Capstone Project**

By Mervat Khaled 1st July 2023

### **1- Definition**

#### **Project Overview**

Artificial intelligence in inventory management is yielding hefty and impressive improvements for the companies that are utilizing it like Amazon and others. We are on the verge of a major upheaval in the way inventory is managed. This revolution is a result of the availability of the huge amounts of real-time data that are now routinely generated on the internet and through the interconnected world of enterprise software systems and smart products. In order to make effective use of this new data and to stay competitive, managers will need to redesign their supply-chain processes. Amazon, for example, implemented artificial intelligence throughout their inventory operations, at an unprecedented scale. In almost every aspect of their operations, A.I methodologies such as time series prediction and reinforcement learning systems are being deployed. User demand, supplier backorders, warehouse optimization, stock levels are all being guided by either machine learning or more complex artificial intelligence systems. [1]

#### **Problem Statement**

As we mentioned above about how inventory management is critical to Amazon's success. Thus, in distribution centers Amazon often uses robots to move objects as a part of their operations, In this project, we will have to build a model that can count the number of objects in (**Amazon Bin Images Dataset**) [2]. A system like this can be used to track inventory and make sure that delivery consignments have the correct number of items.

#### **Dataset**

To complete this project we will be using the Amazon Bin Image Dataset. The dataset contains 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, we will try to classify the number of objects in each bin.

An example of an image and the corresponding metadata file is shown as below: (Source [3])

```
{
  "BIN_FCSKU_DATA": {
    "B000A8C5QE": {
      "asin": "B000A8C5QE",
      "height": {
        "unit": "IN",
        "value": 4.2000000000000001
      },
      "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"
}
```



The “EXPECTED\_QUANTITY” field tells us the total number of objects in image.

However, since this dataset is too large to use as a learning project, and due to cost limitations on the Udacity AWS Portal, we will be using a subset of the data provided to us by Udacity itself. ( [2] )

## **Solution Statement**

The proposed solution involves applying deep learning techniques that have proved to be successful for computer vision. We can use convolutional neural network with a pre-trained model (ResNet Model), and leverage this pre-trained model with transfer learning to solve the problem.

As it is a multi-classification problem; we will adopt the cross-entropy loss function to maximize prediction accuracy.

Besides accuracy, we will evaluate the performance of the model with (Precision, Recall, F1 Score) to identify if a model is doing better on a particular class, or has a high bias for one of them.

**Precision: True positives (correctly predicted class) / Total predicted positives**

**Recall: True predicted positives / Total actual positives**

**F1 Score (Harmonic Mean):  $2 * \text{precision} / (\text{Precision} + \text{Recall})$**

**F1 score: is a way to computing an average of sorts that pays more attention to whichever is lower. Also it is useful for unbalanced classes, as we have here with Amazon Bin Images.**

## **Evaluation Matrix**

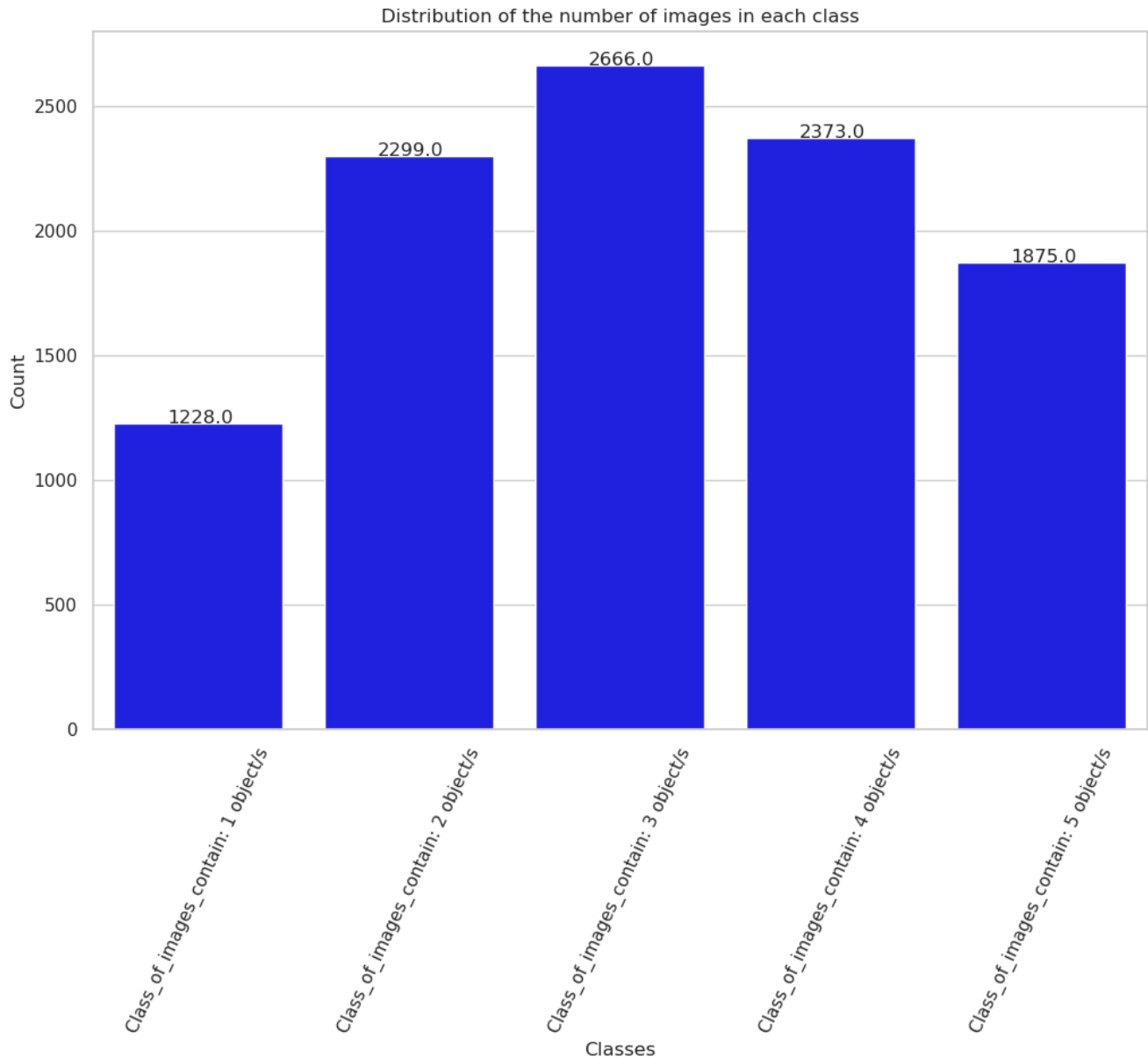
Since we have a Classification Task, and we aim to identify if a model is doing better on a particular class; we can use Accuracy, Recall, Precision and F1 scores as our metrics.

## **2- Analysis**

### **Data Preparation And Splitting**

We worked on a small sample of Amazon Bin Images dataset, which is provided by Amazon in (file\_list.json), This subset of data has 5 classes, corresponding to a number of objects present in each bin: 1, 2, 3, 4 & 5. The total number of images in this subset is 10,441. But we have an unbalanced data issue, thus we have downloaded more data from the original file that contains 500,0000 images, and to avoid downloading an existing image we compared the downloaded image metadata with the subset data in (file\_list.json). Figure 1 shows the distribution of classes before getting more data.

**Figure 1: Distribution of classes**



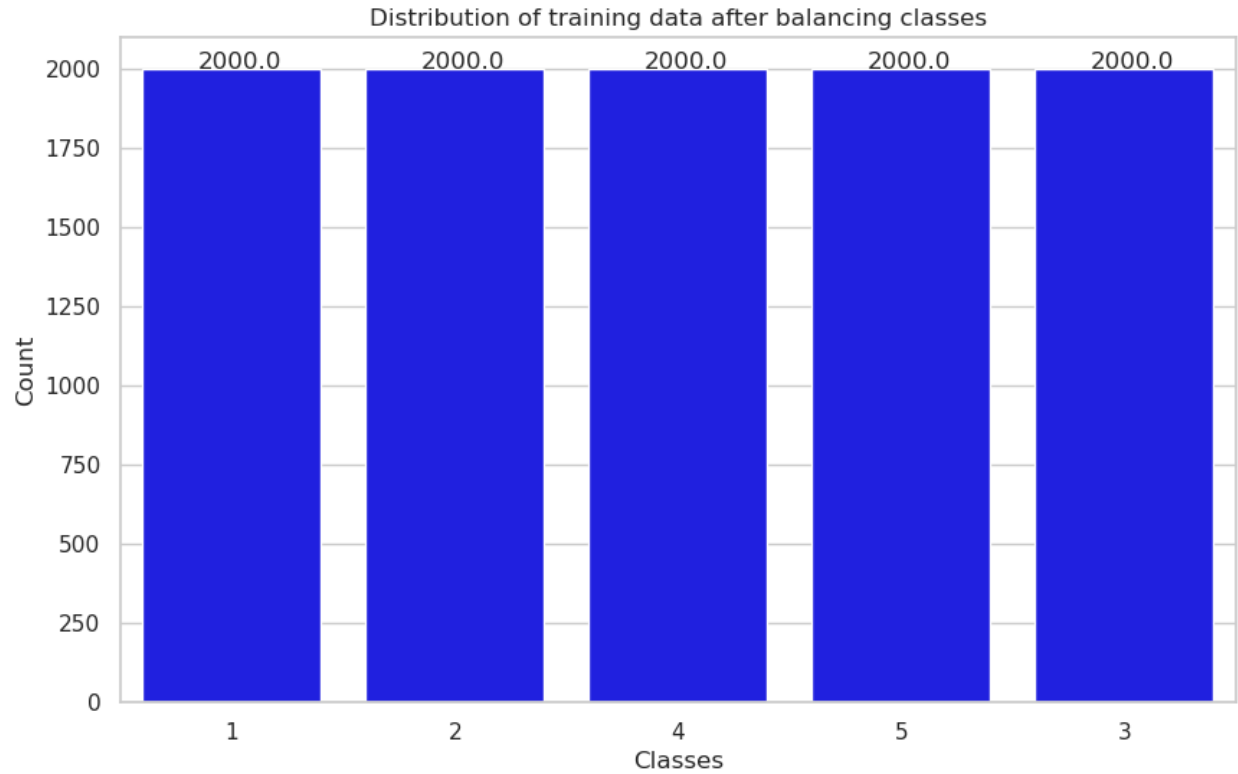
After downloading the needed images for unbalanced classes, we had 13194 images, we took 75% for training data and 15% for testing and validation data.

**Data split as follows:**

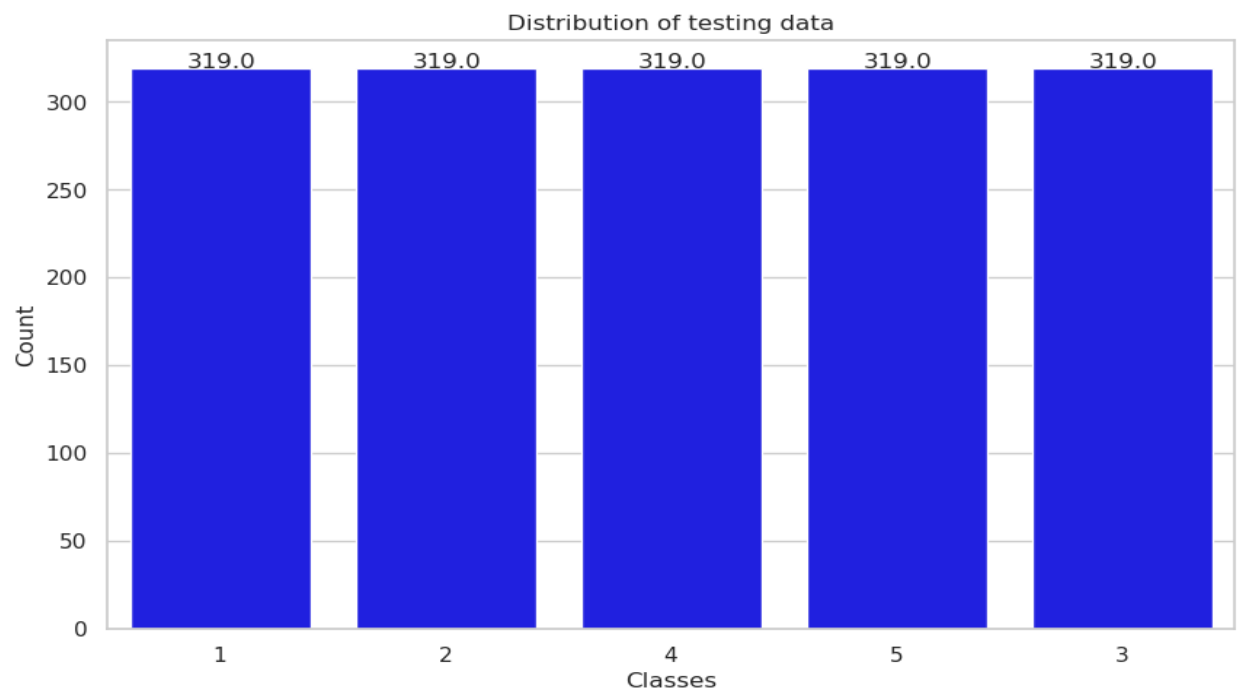
- \* Total samples of training data: 10000 samples, 2000 samples for each class
- \* Total samples of testing data: 1597 samples, 319 samples for each class
- \* Total samples of validation data: 1597 samples, 319 samples for each class

Figure 2,3 and 4 shows the distribution of classes after balancing data

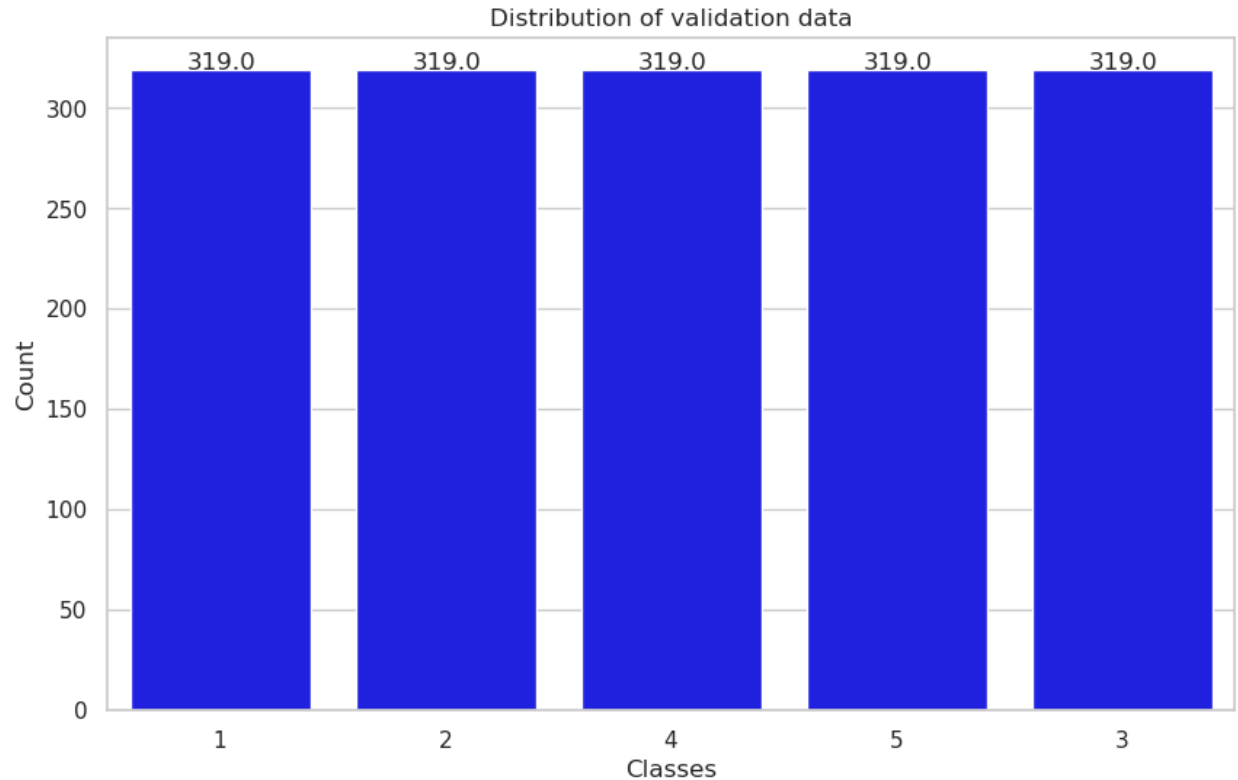
**Figure 2: Distribution of training data after balancing classes**



**Figure 3: Distribution of testing data after balancing classes**



**Figure 4: Distribution of validation data after balancing classes**



## Visualization

We have taken 16 random examples per class to check shape, size, and color exist in the dataset, and we found the following aspects:

- \* Items in each class have a different shape and size and color, which makes generalization hard for any model.
- \* Some images are blurred or smeared.
- \* Items within the bins might be partially or completely occluded by other items or the tape that Amazons uses to prevent items from falling.
- \* Images with one item (in class one) is more clear than in other classes, when the items increasing in images it is difficult even for humans to count them, because they might be very close to each other.

All those aspects have a bad indication that the model might have poor generalization.

**Figure 5: random examples from class 1(images that have one item)**



**Figure 6: random examples from class 2(images that have two items)**





**Figure 7: random examples from class 3(images that have three items)**



**Figure 8: random examples from class 4(images that have four items)**



**Figure 9: random examples from class 5(images that have five items)**



## Algorithms and Techniques

- 1- Fine-tuning a pre-trained model, which is ResNet50, with ranges of hyper parameters to search of best of them to leverage model performance.
- 2- Training the model with best hyper parameters.

## Benchmark

Since we had limited amount of resources from the Udacity AWS Gateway, the resource constraints made training a CNN from scratch, therefore we fine-tuned a pre-trained ResNet50, Convolutional Neural Networks architecture, with hyper parameters: batch size 32, learning rate 0.006 and epoch 10. Table 1 shows the result of the model metrics.

**Table 1: model metrics**

Class	Precision	Recall	F1 Score
1	0.436	0.811	0.567
2	0.266	0.272	0.269
3	0	0.0	0
4	0.275	0.235	0.253
5	0.355	0.448	0.396

---

**Test set Accuracy= 0.353 (35.3%)**

## 3- Methodology

### \* Data Pre-processing

- 1- The data set split to training, testing and validation, each containing 10000, 1597, 1597 images.
- 2- For consistency, we first re-sized the images to (224,224).
- 3- During training, We augmented the images with horizontal flip.

## **\* Implementation**

amazon\_bin\_image\_model\_training\_deploying.ipynb notebook has the detailed implementation, which is well documented, for algorithms and techniques are used in the project.

I have used SageMaker's library to train, test and deploy the model.

## **Steps for implementation**

### **1- Implementing training and evaluation code in (train.py) file.**

- \* Loss and optimizer to use
- \* Data loaders and transformations
- \* Metrics for evaluation
- \* Enable GPU training

### **2- Refinement: Hyperparameters Search**

we have searched for the Batch size and Learning rate, as those hyper parameters are the most difficult to choose manually.

```
In [18]: hyperparameter_ranges = {  
    "learning_rate": ContinuousParameter(0.001, 0.1),  
    "batch_size": CategoricalParameter([32, 64, 128]),  
}  
  
role = sagemaker.get_execution_role()  
  
objective_metric_name = "Valid Loss"  
objective_type = "Minimize"  
metric_definitions = [{"Name": "Valid Loss", "Regex": "Final Validation Loss: ([0-9\\.]+)"}]
```



```
In [19]: estimator = PyTorch(
    entry_point="train.py",
    base_job_name='amazon-bins-hpo',
    role=role,
    framework_version="1.4.0",
    instance_count=1,
    instance_type="ml.m5.2xlarge",
    py_version='py3'
)

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

```
In [20]: os.environ['SM_CHANNEL_TRAINING'] = 's3://capstone-project-amazon-bin-images/data/'
os.environ['SM_MODEL_DIR'] = 's3://capstone-project-amazon-bin-images/model/'
os.environ['SM_OUTPUT_DATA_DIR'] = 's3://capstone-project-amazon-bin-images/output/'
tuner.fit({"training": "s3://capstone-project-amazon-bin-images/data/"}, wait=False)
```

The job completed successfully and the best hyper parameters are: Batch size 32, learning rate 0.0058

```
In [5]: from sagemaker.analytics import HyperparameterTuningJobAnalytics

exp = HyperparameterTuningJobAnalytics(
    hyperparameter_tuning_job_name='pytorch-training-230630-0747')

jobs = exp.dataframe()

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

```
Out[5]:
```

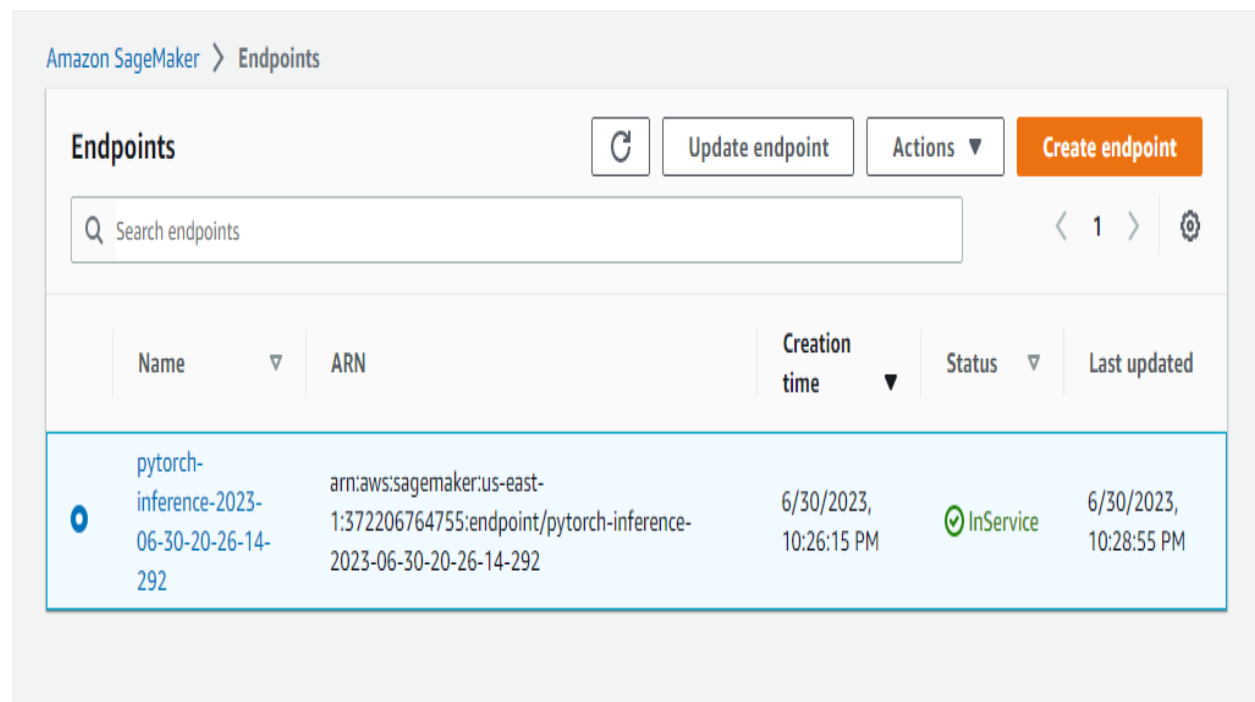
	batch_size	learning_rate	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTime	TrainingElapsedTimeSeconds
2	"128"	0.032647	pytorch-training-230630-0747-002-e8bdb36c	Completed	197.0	2023-06-30 07:49:03+00:00	2023-06-30 09:00:30+00:00	4287.0
0	"128"	0.005163	pytorch-training-230630-0747-004-d5005802	Completed	174.0	2023-06-30 09:01:04+00:00	2023-06-30 10:10:42+00:00	4178.0
3	"32"	0.042857	pytorch-training-230630-0747-001-7a9d2d4f	Completed	51.0	2023-06-30 07:49:03+00:00	2023-06-30 08:55:50+00:00	4007.0
1	"32"	0.005811	pytorch-training-230630-0747-003-a412d263	Completed	44.0	2023-06-30 08:58:58+00:00	2023-06-30 10:04:19+00:00	3921.0

### 3- Refinement: Training Model with Best Hyperparameters

We trained the model with best hyper parameters for 10 epochs.

### 4- Deploying Endpoint

We deployed an endpoint which could be used for inference.



The screenshot shows the Amazon SageMaker Endpoints console. At the top, there's a breadcrumb 'Amazon SageMaker > Endpoints'. Below it, a header bar contains the title 'Endpoints', a refresh button, an 'Update endpoint' button, an 'Actions' dropdown, and a prominent orange 'Create endpoint' button. A search bar labeled 'Search endpoints' is positioned below the header. The main content area features a table with columns: Name, ARN, Creation time, Status, and Last updated. A single endpoint is listed with the name 'pytorch-inference-2023-06-30-20-26-14-292', an ARN 'arn:aws:sagemaker:us-east-1:372206764755:endpoint/pytorch-inference-2023-06-30-20-26-14-292', a creation time of '6/30/2023, 10:26:15 PM', a status of 'InService' (indicated by a green checkmark icon), and a last updated time of '6/30/2023, 10:28:55 PM'.

Name	ARN	Creation time	Status	Last updated
pytorch-inference-2023-06-30-20-26-14-292	arn:aws:sagemaker:us-east-1:372206764755:endpoint/pytorch-inference-2023-06-30-20-26-14-292	6/30/2023, 10:26:15 PM	InService	6/30/2023, 10:28:55 PM

## 5- Inference for each Class

We randomly chose images from test data to visualize how the model acts to predict each class.

Check out `amazon_bin_image_model_training_deploying.ipynb` for detailed result.

## Refinement

\* Correcting imbalanced classes in the dataset: the imbalanced classes would lead to model biased in prediction. Although balancing classes didn't help that much because there are challenges with the images themselves, as we mentioned before.

## 3- Results

### Model Evaluation and Validation

The metrics we determined for evaluating our model were:

Precision, recall, F1 Score and test accuracy.

Class	Precision	Recall	F1 Score
1	0.436	0.811	0.567
2	0.266	0.272	0.269
3	0	0.0	0
4	0.275	0.235	0.253
5	0.355	0.448	0.396

Test Accuracy: 35.3

**Figure 9: The model logs from amazon cloudwatch**

▼	2023-06-30T20:18:21.447+02:00	Precision Computed: {0: 0.43602693602693604, 1: 0.26605504587155965, 2: 0, 3: 0.2757352941176471, 4: 0...	
		Precision Computed: {0: 0.43602693602693604, 1: 0.26605504587155965, 2: 0, 3: 0.2757352941176471, 4: 0.35572139303482586}	Copy
▼	2023-06-30T20:18:21.447+02:00	Recall Computed: {0: 0.8119122257053292, 1: 0.2727272727272727, 2: 0.0, 3: 0.23510971786833856, 4: 0.4...	
		Recall Computed: {0: 0.8119122257053292, 1: 0.2727272727272727, 2: 0.0, 3: 0.23510971786833856, 4: 0.4482758620689655}	Copy
▼	2023-06-30T20:18:21.447+02:00	F1 Computed: {0: 0.5673603504928806, 1: 0.26934984520123845, 2: 0, 3: 0.2538071065989848, 4: 0.3966712...	
		F1 Computed: {0: 0.5673603504928806, 1: 0.26934984520123845, 2: 0, 3: 0.2538071065989848, 4: 0.39667128987517336}	Copy
▶	2023-06-30T20:18:21.447+02:00	Test Len: 1595	
▶	2023-06-30T20:18:21.447+02:00	564.0	
▶	2023-06-30T20:18:21.447+02:00	Testing Loss: 45.0	
▶	2023-06-30T20:18:21.447+02:00	Testing Accuracy: 0.35360501567398117	
▶	2023-06-30T20:18:21.447+02:00	Saving Model	
▶	2023-06-30T20:18:21.447+02:00	2023-06-30 18:18:21,215 sagemaker-training-toolkit INFO Reporting training SUCCESS	

we can conclude from model metrics that:

\* the model has the highest F1 Score in class, thus it performed the best when we called random examples from test data, we got 12 correct prediction of 16.

## **Actual Class: 1, Predicted Class: 1**



\* the model has the lowest F1 score on class 3, and this is logic because it is even hard for humans to enumerate the items in this class.

## **Actual Class: 3, Predicted Class: 5**



### **Justification**

The result of our model is not bad according to the quality of images in the dataset, also it is not good enough. thus If we had more time and resources to fit the model with more data, it would perform much better.

### **Conclusion**

In this project, we explored the object counting problem for images in the (Amazon Bin Image Dataset), we experimented with ResNet50



model and the result wasn't bad, but the model result would be more accurate, If we implanted filter techniques and a good pre-processing to in hence the quality of the images, and definitely fitting the model with more data increases the model performance.

## References

- [1] [Online]. Available: <https://medium.com/@RemiStudios/artificial-intelligence-for-inventory-management-c8a9c0c2a694>
- [2] [Online]. Available: <https://registry.opendata.aws/amazon-bin-imagery/>
- [3] [Online]. Available: <https://github.com/aws-labs/open-data-docs/tree/main/docs/aft-vbi-pds>