

# **Capstone project**

## **Machine Learning Engineer Nanodegree**

### **Inventory Monitoring at Distribution Centers**

#### **Definition**

##### **Project overview:**

Distribution centers often use robots to move objects as a part of their operations. Objects are carried in bins which can contain multiple objects. The task of this project is to build a model able to classify the object class 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.

##### **Problem statement:**

The goal is to build a model that can count or classify the number of objects in each bin into one of 5 classes; class 1, class 2, class 3, class 4 and class 5. The tasks involved are:

- 1- Download and process a subset of Amazon Bin Image Dataset and upload it to S3 bucket.
- 2- Train the model using AWS sagemaker.

- 3- Fine tune the model to improve its performance by using the extracted best hyperparameters of hyperparameters tuning.
- 4- Perform a prediction on the model.

## Metrics:

Test loss and test accuracy are the metrics used to measure the model performance which directly reflected on how well the used solution. Both matrices have a vice versa relationship, low test loss means high test accuracy.

$$\text{Accuracy} = \frac{\text{model prediction} - \text{actual prediction}}{\text{total dataset size}}$$

$$\text{Loss} = \text{loss\_criterion}(\text{model prediction}, \text{actual prediction})$$

*loss\_criterion*, this criterion computes the cross entropy loss between input logits and target [1].

The *target* that this criterion expects should contain either:

- Class indices in the range  $[0, C)$  where  $C$  is the number of classes; if *ignore\_index* is specified, this loss also accepts this class index (this index may not necessarily be in the class range). The unreduced (i.e. with *reduction* set to 'none') loss for this case can be described as:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^C \exp(x_{n,c})} \cdot 1\{y_n \neq \text{ignore}\}.$$

Where  $x$  is the input,  $y$  is the target,  $w$  is the weight,  $C$  is the number of classes, and  $N$  spans the minibatch dimension as well as  $1, \dots, d_1, \dots, d_k$  for the  $K$ -dimensional case[1]

# Analysis

## Data Exploration

Amazon Bin Image Dataset contains over 500,000 images and metadata from bins of a pod in an operating Amazon Fulfillment Center. The bin images in this dataset, a sample given in Fig.1, are captured as robot units carry pods as part of normal Amazon Fulfillment Center operations [2]. Amazon Bin Image Datasets are available in the aft-vbi-pds S3 bucket in the us-east-1 AWS Region [3].

A small subset of Amazon Bin Image Dataset will be downloaded and used in this project as recommended. The subset data is about 10446 bin images.



Fig1. Bin object image.

## Exploratory visualization:

The following plot in Fig2 show the subset of Amazon Bin Image Dataset that will be used in the project it's clear that dataset amount per class is not equal.

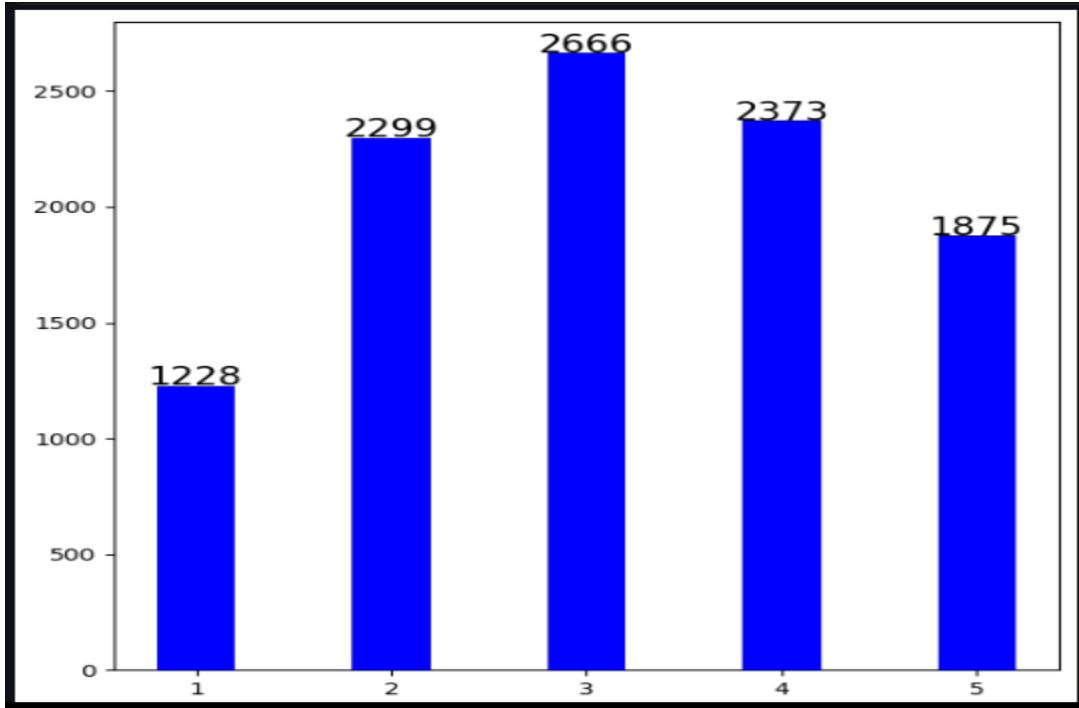


Fig2. Subset of Amazon Bin Image Dataset.

The Images dataset in Fig.2 is not divided into training, testing or validation sets. So Images dataset split randomly over each individual object class as shown in Fig.3 and divided into 80% for training set, 5% for validation set and 15% for testing set.

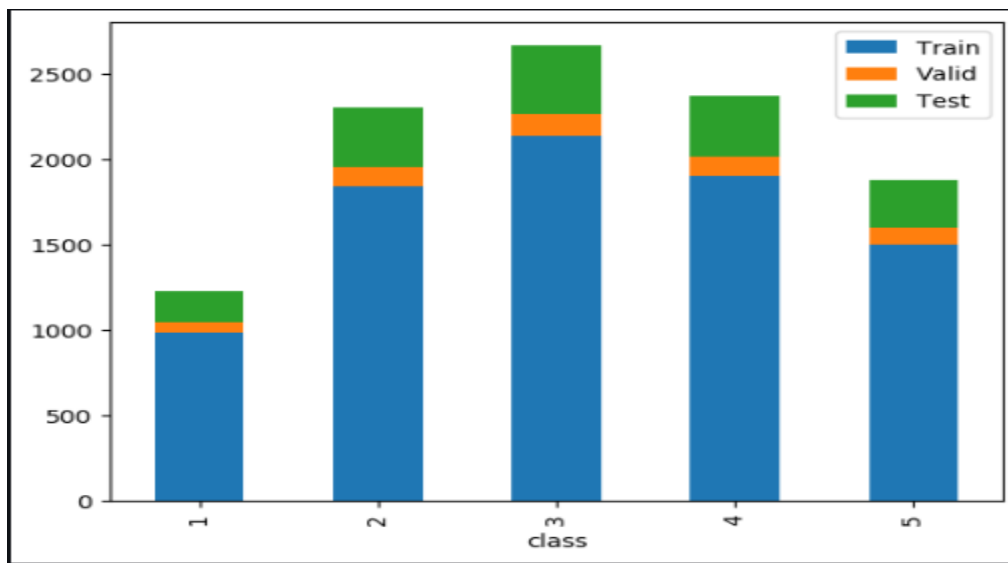


Fig3. Training, validation and testing dataset.

## Algorithms and techniques:

- 1- A model with large neural network architecture will be used where this type of models show excellent performance when dealing with image classification. A pretrained Resnet50 model is trained using randomly chosen fixed hyperparameters; epochs, learning rate and batch-size.
- 2- To improve model performance a hyperparameters tuning is performed to extract best hyperparameters.
- 3- Train the model using best hyperparameters and monitor its performance using model debugging and profiling.

## Benchmark model:

Since distribution centers are considered to be part of a supply chain network, I am going to use one of algorithms used for Supply Chain, Convolution Neural Networks (CNNs) are a type of algorithm that usually deals with image recognition. CNN is one of the most widely used algorithms in supply chain management [4]. CNNs is used to recognize the category of a given image. It's quite handy in the supply chain, as it can classify different products in an instance, and separate them accordingly [4].

## Benchmark model:

- CNN model source is training\_a\_cnn\_solution.py given at Udacity course>Common Model Architecture Types and Fine-Tuning> Exercise: Fine-Tuning a CNN Model [5].
- The model is used as its but with some modification in it, create the following functions: create\_data\_loaders () function and main () function.
- Model is finetuned by pre-trained model resnet18 model, model output layer is modified to equal 5.

- The used hyperparameters to train the model are same as the ones in the original model; epochs=2, learning rate=0.001 and batch-size=10.
- Loss function is CrossEntropyLoss () and used optimizer is Adam optimizer.
- The model will be trained using AWS sagemaker studio through benchmarak\_model.ipynb jupyter notebook.
- Benchmark model is trained on the same preprocessing dataset with the predefined parameters earlier, and the model training steps is the same as the one that mentioned in step 1 in implementation section.
- Test loss and test accuracy of trained benchmark model is shown below in Fig.4.where test accuracy= 27.7884% and test loss = 1.5113.

The screenshot displays the terminal output of a Jupyter notebook session in AWS SageMaker Studio. The output shows the download of a PyTorch model, followed by training progress for Epoch 0 and Epoch 1. The final line of the output, which is circled in red, reports the testing accuracy and loss: "INFO: \_\_main\_\_:Testing Accuracy: 27.78840025493945, Testing Loss: 1.5113466369490323".

```

2023-04-19T13:50:20.185+02:00      Downloading: "https://download.pytorch.org/models/resnet18-5c106cde.pth" to /root/.cache/torch/hub/...
2023-04-19T13:50:20.185+02:00      #015 0% | 0.00/44.7M [00:00<?, ?B/s]#015 78% [██████████] | 34.8M/44.7M [00:00<00:00, 365MB/s]#015100...
#015 0% | 0.00/44.7M [00:00<?, ?B/s]#015 78% [██████████] | 34.8M/44.7M [00:00<00:00, 365MB/s]#015100% [██████████]
44.7M/44.7M [00:00<00:00, 375MB/s]
Copy

2023-04-19T13:50:20.185+02:00      INFO: __main__:Epoch 0, Phase valid
2023-04-19T13:50:20.185+02:00      INFO: __main__:Epoch 1, Phase train
2023-04-19T13:50:20.185+02:00      INFO: __main__:Epoch 1, Phase valid
2023-04-19T13:50:20.185+02:00      INFO: __main__:Testing Model on Whole Testing Dataset
2023-04-19T13:50:20.185+02:00      INFO: __main__:Testing Accuracy: 27.78840025493945, Testing Loss: 1.5113466369490323
2023-04-19T13:50:20.185+02:00      INFO: __main__:saving the model.
2023-04-19T13:50:20.185+02:00      2023-04-19 11:50:19,592 sagemaker-training-toolkit INFO Reporting training SUCCESS
Back to top ^

```

Fig.4. benchmark model accuracy and loss.

# Methodology

## Data preprocessing:

- 1- Dataset used in the project is divided into training, validation and testing sets, split process performed randomly through each individual object class. After that training, validation and testing datasets uploaded to S3 bucket so sagemaker can use it to train the model.
- 2- Different transformation will be applied on the images in train.py script, these transformation divided on two types:
  - Transformation applied on images of training set; random resized crop into (224,224) and random horizontal flip.
  - Transformation applied on images of both validation and testing sets; resize into (224, 224).

## Implementation:

Stages of implementation are:

- 1- Model is trained on sagemaker jupyter notebook by running train.py script, the script implemented as follow:
  - Identify fixed value for hyperparameters; epochs, learning rate and batch size.
  - Implement `create_data_loaders()`; loads training, validation and testing dataset and preprocess them as mentioned earlier in data preprocessing.
  - Identify the used model; pretrained resnet50 model is used and its output layer is modified to equal 5.

- Identify loss function; cross entropy loss.
- Identify optimizer; Adam optimizer.
- Implement train () function to train the model, logging the accuracy and loss for both training and validation.
- Implement test () function, logging test accuracy and loss.
- Save the trained model

2- Perform hyperparameters tuning; epochs, learning rate and batch-size:

- Identify hyperparameters ranges.
- Identify objective metric name, its type and definition.
- Create training estimator and tuner.

3- Training model with extracted hyperparameters from hyperparameters tuning. Monitor model performance using model debugging and profiling:

- Apply the following steps on training script debug\_model.py:
  - a- Import Amazon SageMaker Debugger client library.
  - b- In the train () function, add the *SMDebug* hook for PyTorch with TRAIN mode.
  - c- In the test () function, add the *SMDebug* hook for PyTorch with EVAL mode.
  - d- In the main () function, create the *SMDebug* hook and register to the model.
  - e- In the main () function, pass the *SMDebug* hook to the train () and test ().
  - f- Configure Debugger Rules and Hook Parameters.
- Create an estimator to execute the debug\_model.py.
- Plot the debug output.



- Open profiler report on jupyter notebook to check system metrics and Rule.

#### 4- Deploy the model to an endpoint.

- Train the model using best hyperparameters, by running `train.py` script, since deployment instance is not dedicated for debugging.
- Create `inference.py` script to get a prediction by implementing the following functions:
  - a- `input_fn()` and `output_fn()` to process input and output.
  - b- `model_fn()` calls the loaded model to get a prediction.
  - c- `predict_fn()` customize how the model server gets predictions from the loaded model.
- Create a Predictor class that can easily load model and have a method to classify an image that is in the form of a file object.
- Create a `PyTorchModel` object and identify the following parameters; trained model location, `inference.py` as entry script, framework version and role.
- Call `deploy ()` method of `PyTorchModel` object to deploy the model for inference and identify instance type and instance count.

#### 5- Query the deployed endpoint to get prediction:

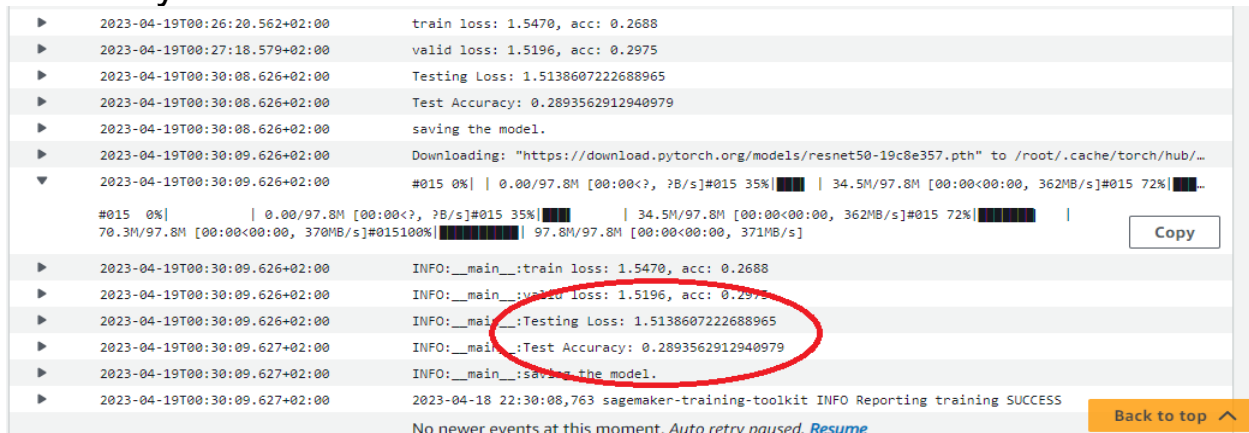
- Randomly chose images that represent the different classes from testing set.
- Call `predict ()` method of predictor, its output is an array representing the probability value for each class.
- Use `numpy.argmax()` function on the probabilities array to get the index of the max probability and adding 1 to

the output of np.argmax and that is it the image or object class.

## Refinement:

The predefined resnet50 model trained at first using fixed hyperparameters; epochs= 4, learning rate = 0.001 and batch size= 64.

Test accuracy and loss is shown below in Fig.5, where test accuracy = 28.9356% and loss = 1.5138.



▶	2023-04-19T00:26:20.562+02:00	train loss: 1.5470, acc: 0.2688
▶	2023-04-19T00:27:18.579+02:00	valid loss: 1.5196, acc: 0.2975
▶	2023-04-19T00:30:08.626+02:00	Testing Loss: 1.5138607222688965
▶	2023-04-19T00:30:08.626+02:00	Test Accuracy: 0.2893562912940979
▶	2023-04-19T00:30:08.626+02:00	saving the model.
▶	2023-04-19T00:30:09.626+02:00	Downloading: "https://download.pytorch.org/models/resnet50-19c8e357.pth" to /root/.cache/torch/hub/...
▼	2023-04-19T00:30:09.626+02:00	#015 0%   0.00/97.8M [00:00<?, ?B/s]#015 35%   34.5M/97.8M [00:00<00:00, 362MB/s]#015 72%   70.3M/97.8M [00:00<00:00, 370MB/s]#015 100%   97.8M/97.8M [00:00<00:00, 371MB/s]
▶	2023-04-19T00:30:09.626+02:00	INFO:__main__:train loss: 1.5470, acc: 0.2688
▶	2023-04-19T00:30:09.626+02:00	INFO:__main__:valid loss: 1.5196, acc: 0.2975
▶	2023-04-19T00:30:09.626+02:00	INFO:__main__:Testing Loss: 1.5138607222688965
▶	2023-04-19T00:30:09.627+02:00	INFO:__main__:Test Accuracy: 0.2893562912940979
▶	2023-04-19T00:30:09.627+02:00	INFO:__main__:saving the model.
▶	2023-04-19T00:30:09.627+02:00	2023-04-18 22:30:08,763 sagemaker-training-toolkit INFO Reporting training SUCCESS

Fig.5 Test accuracy and loss of trained model with fixed hyperparameters.

Test accuracy of model is higher with 1.1472% Compared to benchmark model accuracy, while loss of model is higher to 0.0025 compared with benchmark loss.

To improve the model performance hyperparameters tuning is performed on the model.

Test accuracy and loss is shown below in Fig.6, where test accuracy = 30.2103% and loss = 1.4891.

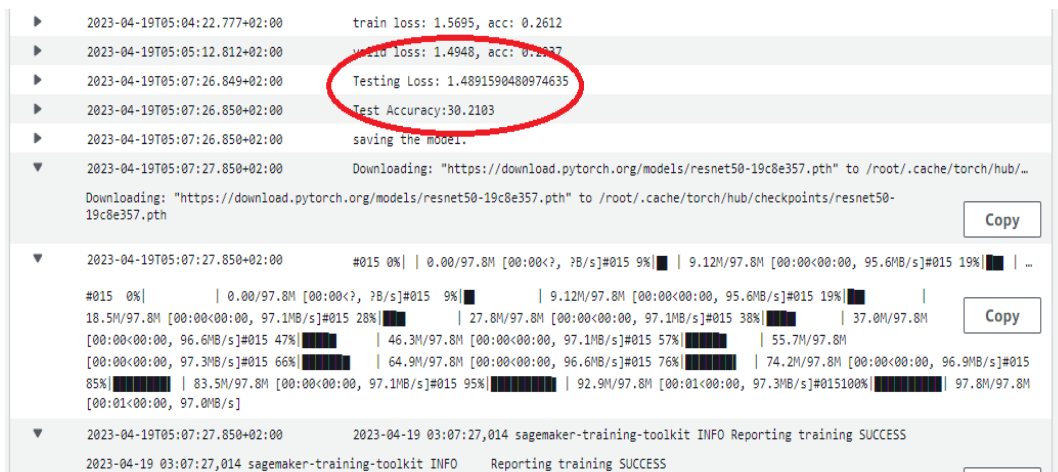


Fig.6 Test accuracy and loss of trained model with best hyperparameters

Test accuracy of model is higher with 2.4219% Compared to benchmark model accuracy, and loss of model is lower with 0.0222 compared to benchmark loss.

## Results

### Model evaluation and validation:

A validation dataset was use to evaluate the model.

Final hyperparameters used to train the model are:

- Epochs = 2.
- Learning rate = 0.005482862483508994.
- Batch-size = 32.

To verify the model trained with final hyperparameters, the model is deployed to an endpoint as shown in Fig.7.

Amazon SageMaker > Endpoints

Endpoints ↻ Update endpoint Actions ▾ Create endpoint

Q Search endpoints < 1 > ⚙

	Name ▾	ARN	Creation time ▾	Status ▾	Last updated
<input type="radio"/>	pytorch-inference-2023-04-19-04-58-20-532	arn:aws:sagemaker:us-east-1:027003389477:endpoint/pytorch-inference-2023-04-19-04-58-20-532	4/19/2023, 6:58:21 AM	<span>✔ InService</span>	4/19/2023, 7:00:57 AM

Fig.7 endpoint of deployed model.

The endpoint is tested by 5 random samples of each class of test dataset. Some of the results shown in the figures below from Fig.8 to Fig.13:



Fig.8

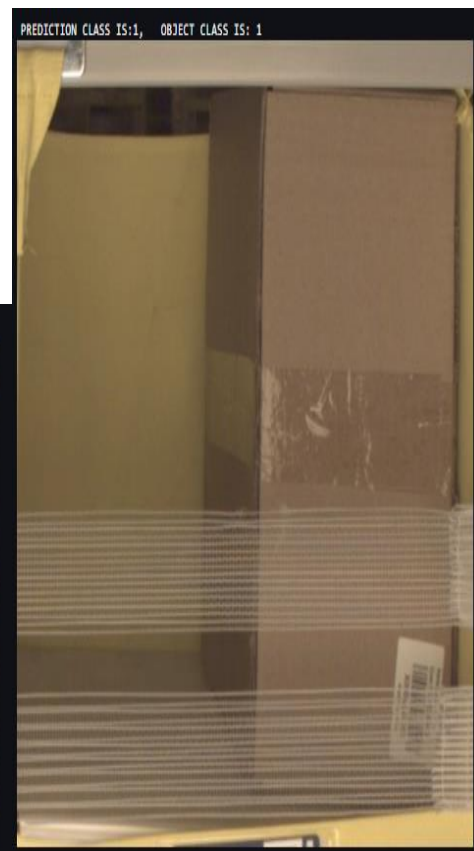


Fig.9



Fig.10

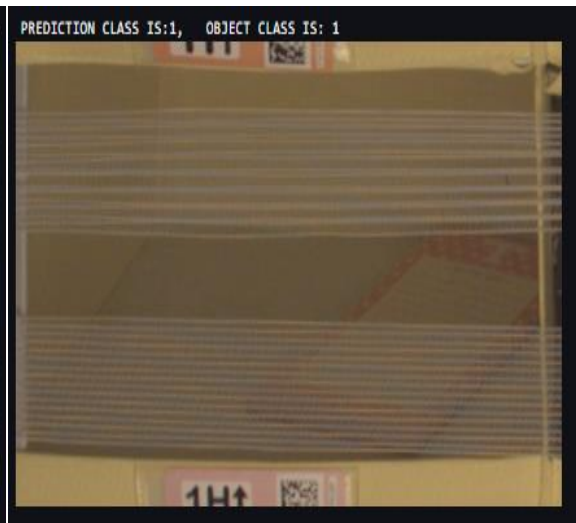


Fig.11



Fig.12



Fig.13

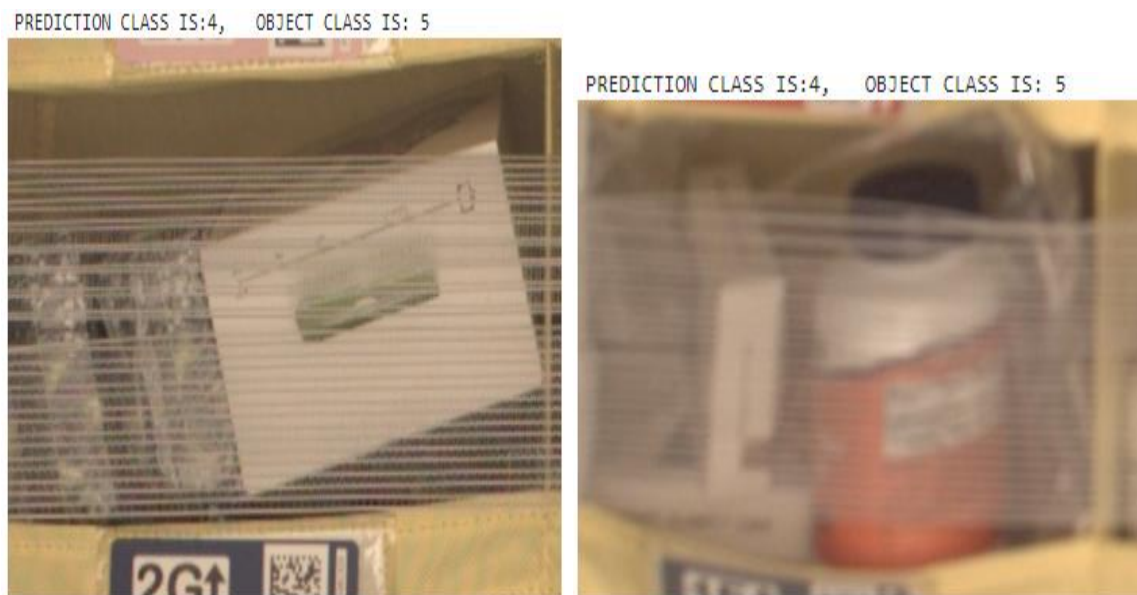
It's obvious from the previous test results that Fig.8, Fig.10 and Fig.13 that the model fails to classify the object class. While Fig.9, Fig.11 and Fig.12 the model success to classify the object class.

## Justification:

The results were as expected due to low model accuracy 30%, but we're restricted to the recommended subset of Amazon Bin Image dataset. So using larger dataset in training the model beside best tuned hyperparameters the results would be promising.

## Conclusion

### Object captured images:



The above two sample images are a good example to show why the model may fail to predict the object class correctly:

- Object image is foggy.
- Number of object in the image isn't clear even by human.

## Reflection:

The process used for the project summary:

- 1- Real-life project, with available real dataset.
- 2- Data subset downloaded and preprocessed and uploaded to S3 bucket.
- 3- AWS sagemaker used to train pretrained model using the uploaded dataset.
- 4- The model is trained using the best tuned hyperparameters, and model debug and profile used to monitor the model performance.
- 5- Model was deployed to an endpoint and tested by image samples.

## Improvement:

To improve the model accuracy:

- Use large dataset to training the model beside best tuned hyperparameters the results would be promising.
- Provide more object images captured from different sides or views, in turn more reliable model.

## References:

- 1- <https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html>
- 2- <https://registry.opendata.aws/amazon-bin-imagery/>
- 3- <https://github.com/awsmlabs/open-data-docs/tree/main/docs/aft-vbi-pds>
- 4- <https://neptune.ai/blog/use-cases-algorithms-tools-and-example->
- 5- [https://9f84d623cee9420aae7fbcf3e3c2bfea.udacity-student-workspaces.com/edit/finetune\\_a\\_cnn\\_solution.py](https://9f84d623cee9420aae7fbcf3e3c2bfea.udacity-student-workspaces.com/edit/finetune_a_cnn_solution.py)

