Capstone Project Proposal for Inventory Monitoring at Distribution Centres

Tan Nguyen
AI Engineer at FPT Software AI Center
TanNP3@fpt.com

1. Definition

Project Overview

In the realm of Robotics, the genesis of our problem arises, particularly within its application in industrial domains such as Inventory Monitoring and Supply Chain Operations. Corporations engaged in managing physical cargo and overseeing supply chains have been actively pursuing automation to enhance efficiency and precision. Take Amazon, for instance, a colossal hub for diverse goods delivery. Within its expansive warehouses, the vast quantities of stored items necessitate inventory management on a grand scale. Manual inventory checks, given the enormity of these quantities, demand a significant investment in human resources and are susceptible to errors.

Enter robots, the champions of Inventory Monitoring. Equipped with Machine Learning Models, these robots adeptly perform tasks such as Object Detection, Outlier & Anomaly Detection, among others. Once trained, these models are easily scalable, offering a cost-effective solution applicable to warehouses and distribution centers, facilitated by industry-grade robots.

Transitioning to distribution centers, robots play an integral role in object transportation, often utilizing bins to carry multiple items. Yet, occasional mishandling can lead to discrepancies between recorded bin inventory and actual contents. A pivotal need emerges for a system that not only ensures precise inventory tracking but also guarantees the complete delivery of consignments by accurately counting items in each bin.

In the pursuit of this objective, our project endeavors to develop a robust model capable of performing item counts within individual bins. Such a system holds the potential to revolutionize inventory tracking, thereby safeguarding accurate delivery of consignments while maintaining item integrity.

The Problem Statement

As noted earlier in the domain description, distribution centers commonly employ robots to transport items. These items are housed within bins, with each bin having the capacity to hold between 1 and 5 objects.

The central challenge that this project endeavors to address involves the accurate enumeration of items contained within these bins. The significance of resolving this challenge cannot be

understated, as it holds substantial practical applications. By crafting a model capable of processing images of bins and providing an accurate count of enclosed objects, we have the potential to offer a transformative solution. This accomplishment would lead to the full automation of a pivotal stage within the Inventory Management process.

Metrics

As for model evaluation, the following metrics will be calculated:

- F1 $= \frac{2*Precision*Recall}{Precision+Recall}$

Where:

- $Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$

- $Recall \frac{True\ Positive}{True\ Positive + False\ Negatie}$

Given the presence of class distribution disparities within the dataset, relying solely on accuracy as a performance metric is inadequate. Instead, the F1 Score has been adopted to provide a more comprehensive evaluation of model effectiveness, particularly in addressing the repercussions of imbalanced classes. While accuracy is not a suitable measure when facing dataset imbalances, there are instances where certain subsets of the data exhibit minimal imbalances. In such cases, it becomes plausible to overlook the adverse impacts of the imbalance and proceed with utilizing accuracy as a metric of assessment.

2. Analysis

Data Exploration

The Amazon Bin Image Dataset contains 536,434 images and metadata from bins of a pod in an operating Amazon Fulfillment Center. The bin images in this dataset are captured as robot units carry pods as part of normal Amazon Fulfillment Center operations. This dataset has many images and the corresponding medadata.

The image files have three groups according to its naming scheme.

- A file name with 1~4 digits (1,200): 1.jpg ~ 1200.jpg
- A file name with 5 digits (99,999): 00001.jpg ~ 99999.jpg
- A file name with 6 digits (435,235): 100000.jpg ~ 535234.jpg

Sample metadata file:

```
{
    "BIN_FCSKU_DATA": {
        "B00PLKV5H6": {
            "asin": "B00PLKV5H6",
            "height": {
                "unit": "IN",
                "value": 6.799999999999999
            },
            "length": {
                "unit": "IN",
                "value": 7.0
            },
            "name": "HBD Thermoid NBR/PVC SAE30R6 Fuel Line Hose, 5/16\" x 25' Length, 0.3125\" ID, Black",
            "quantity": 1,
            "weight": {
                "unit": "pounds",
                "value": 3.0
            },
            "width": {
                "unit": "IN",
                "value": 7.0
            }
        },
        "B00WTI3SGO": {
            "asin": "B00WTI3SGO",
            "height": {
                "unit": "IN",
                "value": 1.2
            },
            "length": {
                "unit": "IN",
                "value": 7.6
            },
            "name": "The Witcher 3: Wild Hunt - PC",
            "quantity": 1,
            "weight": {
                "unit": "pounds",
                "value": 0.6
            },
            "width": {
                "unit": "IN",
                "value": 5.4
            }
        }
    },
    "EXPECTED_QUANTITY": 2,
    "image_fname": "500.jpg"
}
```

The "EXPECTED_QUANTITY" field is 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.

Within this specific dataset segment, there exist 5 distinct categories, each corresponding to the quantity of items contained within a bin: 1, 2, 3, 4, and 5. The cumulative count of images within this particular subset amounts to 10,441.

Image illustrate distribution of sub-dataset which Udacity provide

Number of Classes in Each File
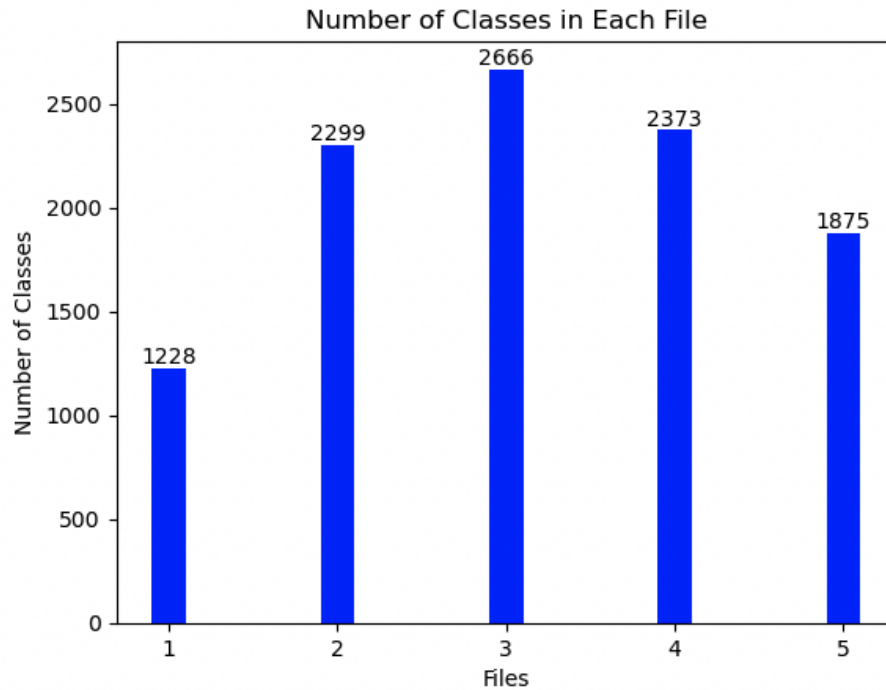
We conducted a data split in three parts, specifically a Train-Test-Validation Split, using the following approach. Validation is choose to best model in training progress and Test measure best model in total trained model with difference methods:

1. We randomly selected 100 images from each class and designated them as our Test Data.

2. For the remaining data, we performed an 80-20 split into Training and Validation sets. Importantly, we applied a Stratified Split, ensuring that the distribution of classes remained consistent in both the Training and Validation sets to maintain data balance.

Exploratory Visualization

Num of Objects	Data characteristics	Sample
1	-It is evident that the images belonging to Class 1 are deemed valid -These images appear to be straightforward for human recognition -Occasionally, the presence of a net in the container might pose challenges in clearly discerning the objects.	

2	- It's evident that the Class 2 Images are considered valid. - There are instances where two objects are in such proximity that they appear as a single object. - The presence of a net in the container occasionally poses challenges in clearly discerning the objects.	
3	- Upon meticulous examination of the images, we confirm the validity of Class 3 Images. - These images present a challenge even for human observers, and it's likely that the model may struggle to perform well on this particular class. - The presence of a net within the container occasionally hampers clear visibility of the objects.	
4	- It required meticulous scrutiny of images and a larger sample size to establish the validity of Class 4 Images. - These images are challenging to authenticate even by human observers, and it's anticipated that the model might face difficulties in performing well on this class. - The presence of a net within the container intermittently obstructs clear object visibility.	
5	- The images exhibit significant clutter due to the presence of multiple objects, making it challenging to consistently discern the presence of five objects. - Human classification of these images is exceptionally difficult. - The existence of a net within the container intermittently obstructs clear object visibility.	

Algorithms and Techniques

In this section, we present our solution which tries to integrate the pros of previous solutions
In general, the steps we did can be listed as follows:
1. Use transfer learning to fine-tune a pre-trained BeIT model on our dataset.
2. Hyperparameter tuning
3. Training and evaluation

Benchmark

Method: Finetuning a Pre-Trained BeIT Model

```
hyperparameters={'epochs': 5,              # number of training epochs
                 'train_batch_size': 32,   # batch size for training
                 'eval_batch_size': 64,    # batch size for evaluation
                 'learning_rate': 5e-5,    # learning rate used during training
                 'model_id':'microsoft/beit-base-patch16-224', # pre-trained model
                 'warmup_ratio': 0.1
                 }
```

Benchmark Hyperparameter

F1 Score:

Validation Score:

```
'eval_f1': 0.45326633165829144,
```

Test Score:

```
epoch = 5.0
eval_accuracy = 0.48
eval_f1 = 0.472
eval_loss = 1.2205156087875366
```

3. Methodology

In this section, we introduce the experiment details including the data preprocessing, implementation procedure, improvements and refinements.

Data Preprocessing

We exclusively utilize a processor supplied by HuggingFace, which functions by resizing images to a standardized resolution of 224x224.

```
    # load processor
    image_processor = AutoImageProcessor.from_pretrained(args.model_id)

    # preprocess function
    def preprocess_function(examples):
        return image_processor(examples["image"],return_tensors="pt")

    # preprocess dataset
    train_dataset = dataset['train'].map(preprocess_function,batched=True)
    val_dataset = dataset['validation'].map(preprocess_function,batched=True)
    test_dataset = dataset['test'].map(preprocess_function,batched=True)

    train_dataset = train_dataset.remove_columns(["image"])
    val_dataset = val_dataset.remove_columns(["image"])
    test_dataset = test_dataset.remove_columns(["image"])
```

Implementation

The benchmark model is implemented as shown in the following snippet code:

```
model = BeitForImageClassification.from_pretrained("microsoft/beit-base-patch16-224", ignore_mismatched_sizes=True, num_labels=num_labels)

# define training args
training_args = TrainingArguments(
    output_dir=args.model_dir,
    num_train_epochs=args.epochs,
    per_device_train_batch_size=args.train_batch_size,
    per_device_eval_batch_size=args.eval_batch_size,
    warmup_ratio=args.warmup_ratio,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    logging_dir=f"{args.output_data_dir}/logs",
    learning_rate=float(args.learning_rate),
    load_best_model_at_end=True,
    save_total_limit = 1,
    metric_for_best_model="f1",
)

# create Trainer instance
trainer = Trainer(
    model=model,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    data_collator=default_data_collator,
)

# train model
trainer.train()

# evaluate model
eval_result = trainer.evaluate(eval_dataset=test_dataset)
```

We also prepared the possibility of using debugger and profiler to have a comprehensive look at computations on our machines.

These implementations are also done for our improved model which we'll talk about later.

Refinement

The benchmark model underwent a process of hyperparameter tuning, focusing on the hyperparameters "batch size," "epochs," and "learning rate." Through this tuning process, the model arrived at the following configuration as the optimal choice.

```
[28]: best_estimator = tuner.best_estimator()

      best_estimator.hyperparameters()

      2023-08-28 08:57:43 Starting - Preparing the instances for training
      2023-08-28 08:57:43 Downloading - Downloading input data
      2023-08-28 08:57:43 Training - Training image download completed. Training in progress.
      2023-08-28 08:57:43 Uploading - Uploading generated training model
      2023-08-28 08:57:43 Completed - Resource released due to keep alive period expiry
[28]: {'_tuning_objective_metric': '"eval_f1"',
       'epochs': '6',
       'learning_rate': '6.301966492587569e-05',
       'sagemaker_container_log_level': '20',
       'sagemaker_estimator_class_name': '"HuggingFace"',
       'sagemaker_estimator_module': '"sagemaker.huggingface.estimator"',
       'sagemaker_job_name': '"huggingface-beit-2023-08-28-06-48-48-2023-08-28-06-48-50-561"',
       'sagemaker_program': '"train.py"',
       'sagemaker_region': '"us-east-1"',
       'sagemaker_submit_directory': '"s3://sagemaker-us-east-1-379150537365/huggingface-beit-2023-08-28-06-48-48-2023-08-28-06-48-50-561/source/sourcedir.tar.gz"',
       'train_batch_size': '"16"'}
```

Considering the benchmark model's performance on our dataset, we recognized the importance of enhancing the model. As part of the improvement and refinement process, we have implemented data augmentation. In this improvement step, we augment the training data using the following configuration:

```python
# load processor
image_processor = AutoImageProcessor.from_pretrained(args.model_id)

_transforms = Compose([
        ColorJitter(brightness=0.5, hue=0.5),
        RandomGrayscale(p=0.1),
        Resize((224,224)),
        ToTensor(),
        Normalize(mean=image_processor.image_mean, std=image_processor.image_std)
    ])


# preprocess function
def transforms(examples):
    examples["pixel_values"] = [_transforms(image.convert("RGB")) for image in examples["image"]]
    return examples


# preprocess dataset
train_dataset = dataset['train'].map(transforms,batched=True)
val_dataset = dataset['validation'].map(transforms,batched=True)
```

4. Result

Model Evaluation and Validation

	batch size	epoch	learning rate	augmentation	F1-Val	F1 Test
Benchmark	32	5	5e-5	False	0.453	0.472
HPO	16	6	6.3e-5	False	0.45	0.462
HPO + Augmentation	16	6	6.3e-5	True	0.417	0.472

Justification

Benchmark model:

Improved model

Confusion Matrix

	precision	recall	f1-score	support
1 Object	0.90	0.64	0.75	100
2 Objects	0.42	0.45	0.44	100
3 Objects	0.33	0.53	0.41	100
4 Objects	0.34	0.34	0.34	100
5 Objects	0.53	0.33	0.41	100
accuracy			0.46	500
macro avg	0.51	0.46	0.47	500
weighted avg	0.51	0.46	0.47	500
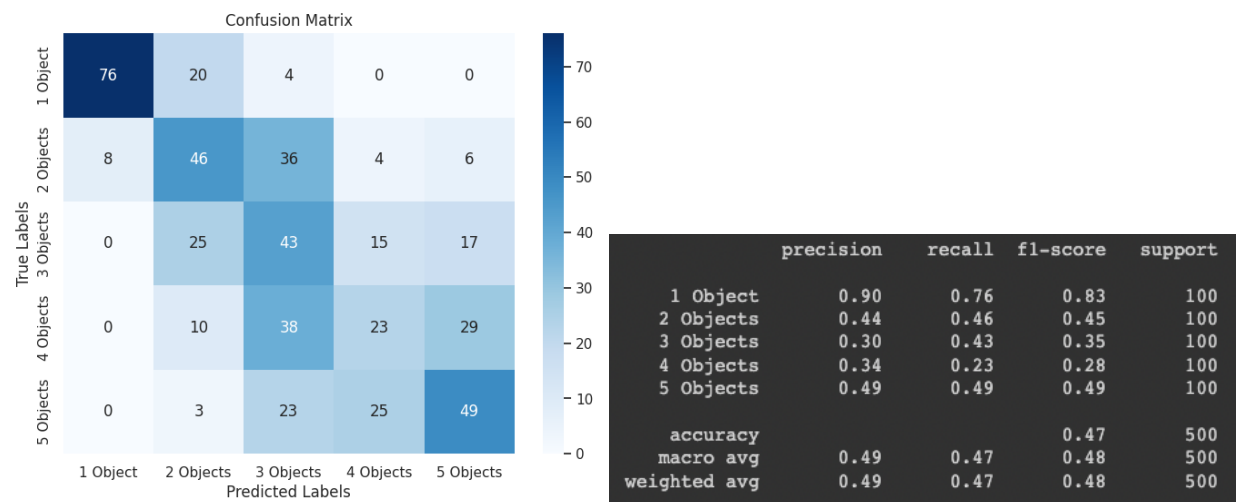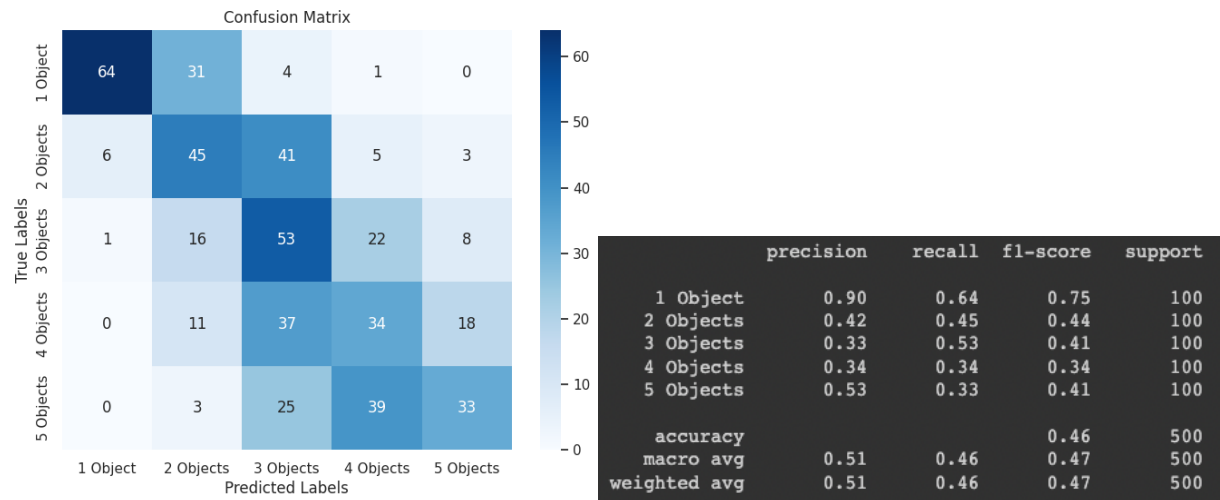
	1	2	3	4	5
Benchmark model	0.83	0.45	0.35	0.28	0.49
Improved model	0.75	0.44	0.41	0.34	0.41

Result F1 score metric for benchmark and improved models per classes

5. Conclusion

During this project, we conducted exploratory data analysis (EDA), selected the most appropriate pre-trained model, and explored innovative approaches for improvement. We maximized the utility of the provided dataset by employing data augmentation techniques, generating various versions of existing data. Notably, our model demonstrated enhancements, particularly in classifying objects within categories 3 and 4, effectively aligning with the project's objectives. Nevertheless, it's worth noting that addressing the issue of data imbalance remains a challenge for future endeavors.