Inventory Monitoring at Distribution Centers

Ayesha Mosaddeque

28th March, 2023

Project Definition

Background

Computer Vision is a branch of Artificial Intelligence that focuses on enabling machines to interpret, understand and analyze visual data from the world around them. It involves developing algorithms and techniques that allow computers to perceive, process and extract useful information from digital images, videos and other visual inputs.

The primary objective of Computer Vision is to mimic human visual perception, which involves the ability to recognize patterns, objects, and their relationships in a scene. Computer Vision algorithms use mathematical and statistical models to analyze visual data and identify key features such as edges, shapes, colors, and textures. These features are then used to classify and categorize objects, track their movements, and even reconstruct 3D models of the environment.

Computer Vision has numerous practical applications in various fields, including robotics, autonomous vehicles, surveillance, medical imaging, manufacturing, and entertainment. For example, in the field of robotics, Computer Vision is used to enable robots to perceive their surroundings and interact with objects and humans. In the field of medical imaging, it is used for early detection of diseases and for surgical planning. In the entertainment industry, it is used for special effects and computer-generated imagery. Overall, Computer Vision is a rapidly advancing field with great potential to revolutionize the way machines interact with the world and transform numerous industries.

Problem Description

Accurately counting the number of items in a bin is a significant challenge that Amazon Fulfillment Centers encounter when managing inventory. Bins, which are standard storage units used in Amazon's warehouses to store products, need to be counted precisely.

In the past, counting inventory has been a slow and laborious process that involves manually counting each item in a bin, which is susceptible to mistakes. To address this issue, Amazon began investigating the use of Computer Vision technology to automate the process of counting items in bins. Employing Robotics and Computer Vision in inventory management has significantly enhanced the precision and effectiveness of counting items in bins, resulting in reduced time and resources required for managing inventory. The automated inventory management and logistics introduced by Amazon's solution have the potential to revolutionize the retail and e-commerce industry by offering new possibilities. Bertorello et al. (2018) has academic research done in this domain.

Project Analysis

Data Analysis

For this project I have used 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, its dimension and the type of object. For this task, we will try to classify the number of objects in each bin. The images are available here: *s3://aft-vbi-pds/bin-images/*. Images are located in the bin-images directory, and metadata for each image is located in the metadata directory. Images and their associated metadata share simple numerical unique identifiers. For example, the metadata for the image at https://aft-vbi-pds.s3.amazonaws.com/bin-images/523.jpg is found at https://aft-vbi-pds.s3.amazonaws.com/metadata/523.json. For every image, the corresponding metadata contains information about the quantity. Following is an example image from https://aft-vbi-pds.s3.amazonaws.com/bin-images/526.jpg:

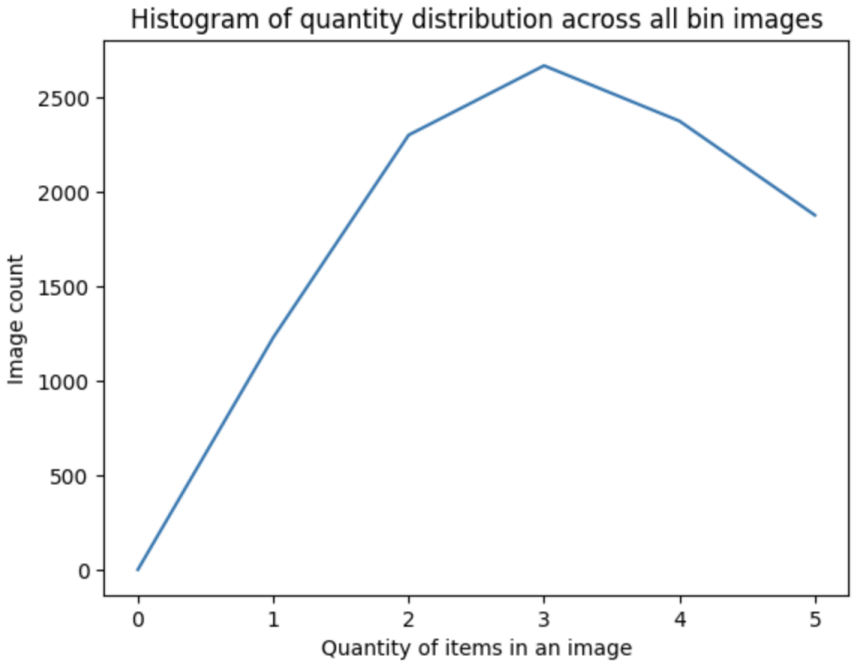
The corresponding metadata retrieved from https://aft-vbi-pds.s3.amazonaws.com/metadata/526.json is as follows:

```
{
   "BIN_FCSKU_DATA": {
      "B00CSWH7G8": {
         "asin": "B00CSWH7G8",
         "height": {
            "unit": "IN",
            "value": 6.500000000000001
         },
         "length": {
            "unit": "IN",
            "value": 9.3
         },
         "name": "Jim Shore Halloween Is At Hand",
         "quantity": 1,
         "weight": {
            "unit": "pounds",
            "value": 3.45
         },
         "width": {
            "unit": "IN",
            "value": 7.8
         }
      },
      "B019NOTJ1O": {
         "asin": "B019NOTJ1O",
         "height": {
            "unit": "IN",
            "value": 2.1000000000000005
         },
         "length": {
            "unit": "IN",
            "value": 9.0
         },
         "name": "Polo Ralph Lauren Unisex Merino Wool Cuff Olive Green Beanie Hat One
Size",
         "quantity": 1,
         "weight": {
            "unit": "pounds",
            "value": 0.15
         },
         "width": {
            "unit": "IN",
            "value": 6.6
         }
      }
   },
   "EXPECTED_QUANTITY": 2,
   "image_fname": "526.jpg"
}
```

I have used a subset of the actual data. For this task I have only considered this subset of data listed in `file_list.json`. The keys of this json file are the count of the objects in a bin and the value is a list of images containing the said amount of objects. This project only considers images with 1 to 5 numbers of objects in them. Following are some exploration of the data. We can see some statistics on this data subset in the following table:

Description	Value
Total number of images	10441
Average number of images in any one bin	2088.2
Total number of images with quantity 1	1228
Total number of images with quantity 2	2299
Total number of images with quantity 3	2666
Total number of images with quantity 4	2373
Total number of images with quantity 5	1875

Following is a histogram of quantity distribution:



I have split the data into 3 parts - training, validation and test data. Training data is 70% of the total data subset. Validation and test data each are 15% of the total data subset. Sharing some similar insight on the 3 partitions of data below.

Train

Description	Value
Total images	7312
Average number of images	1462.4
Total images with quantity 1	860
Total images with quantity 2	1610
Total images with quantity 3	1867
Total images with quantity 4	1662
Total images with quantity 5	1313

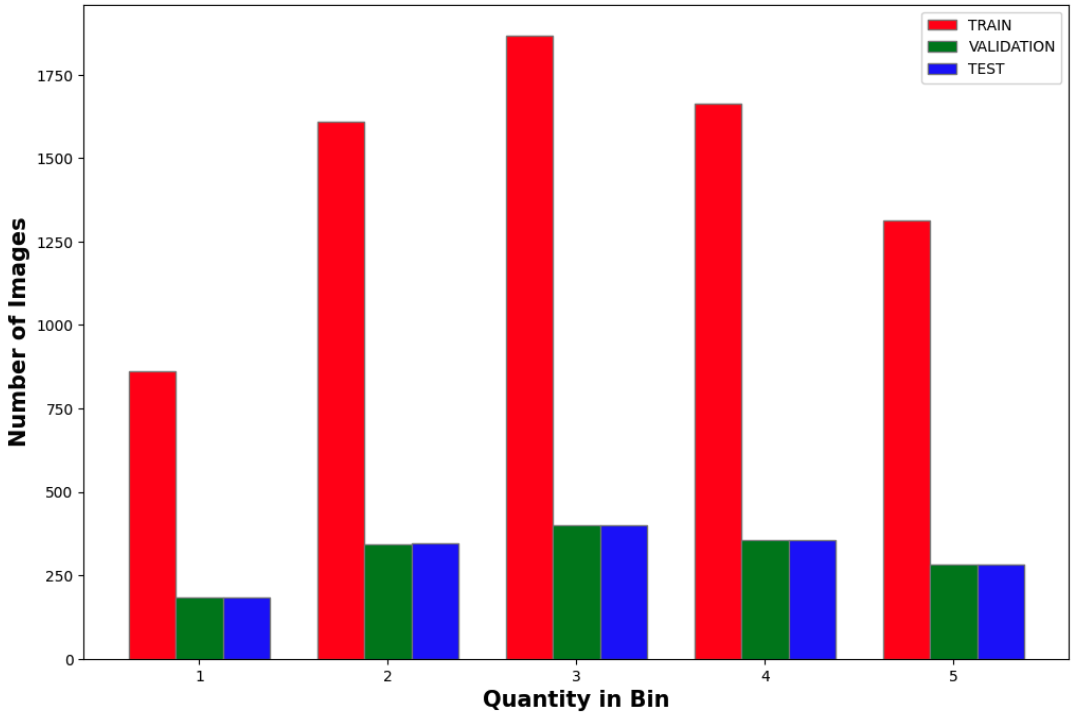
Validation

Description	Value
Total images	1563
Average number of images	312.6
Total images with quantity 1	184
Total images with quantity 2	344
Total images with quantity 3	399
Total images with quantity 4	355
Total images with quantity 5	281

Test

Description	Value
Total images	1566
Average number of images	313.2
Total images with quantity 1	184
Total images with quantity 2	345
Total images with quantity 3	400
Total images with quantity 4	356
Total images with quantity 5	281

A visual representation of the data partitions:



Model Analysis

Convolutional Neural Networks (CNNs) are deep neural networks that are frequently used to classify images. CNNs leverage convolutional layers to examine the image and extract significant features. These layers employ filters on the image to detect patterns and edges that can be utilized for image classification. Following every convolutional layer, a pooling layer is applied to shrink the feature map. This results in a decrease in the number of parameters and computational complexity of the network while retaining crucial features. CNNs are ideal for image classification because they can automatically acquire significant features from the image data and subsequently employ those features to correctly classify the image.

Hence, I have created a simple CNN model from scratch to create a baseline model. The model has 2 Convolutional layers and 3 Linear layers. I have also used a 2X2 Max pooling. Following is my baseline CNN model:

```
class Model(nn.Module):
   def __init__(self):
       super().__init__()
       self.cnnlayer1 = nn.Conv2d(3, 6, 5)
       self.pool = nn.MaxPool2d(2, 2)
       self.cnnlayer2 = nn.Conv2d(6, 16, 5)
       self.linear1 = nn.Linear(16 * 53 * 53, 256)
       self.linear2 = nn.Linear(256, 84)
       self.linear3 = nn.Linear(84, 5)


   def forward(self, x):
       x = self.pool(F.relu(self.cnnlayer1(x)))
       x = self.pool(F.relu(self.cnnlayer2(x)))
       x = torch.flatten(x, 1)
       x = F.relu(self.linear1(x))
       x = F.relu(self.linear2(x))
       x = self.linear3(x)
       return x
```

For baseline training the following were my optimizer and loss function. I have passed 3 hyperparameters.

```
loss_criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=args.lr, momentum=args.momentum)
```

I have used the following hyperparameter values:

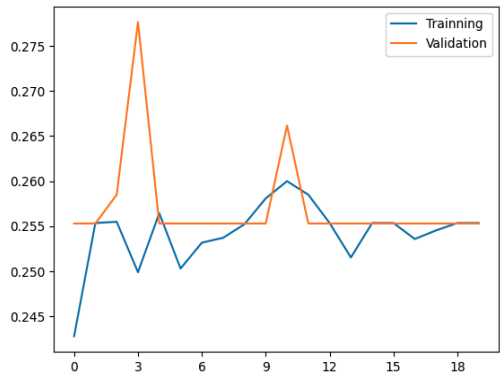
hyperparameters = {
"batch-size": 64,
"epochs": 20,
"lr": 0.08596215306782015,
"momentum": 0.5712572770971469
}

The outcome of the base model is:

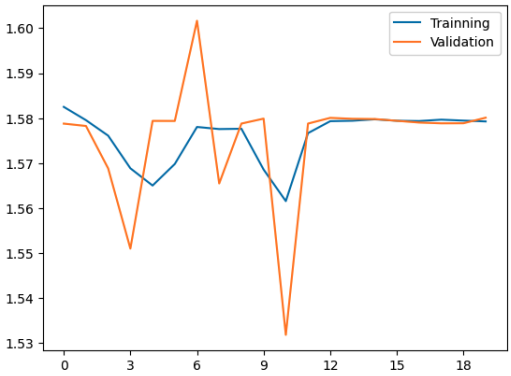
Metric	Value
Average Testing Loss	1.8000916684115376
Testing RMSE	52.449237785110114
Testing Accuracy	24.010217113665387

Baseline model outcome

The accuracy and loss for training and validation sets using baseline model is like so:



Accuracy



Loss

CNNs have the capability to be pre-trained on extensive image datasets like ImageNet, followed by customization for specific image classification tasks. This process enables the network to grasp significant features from a vast dataset, and then employ that knowledge to a new task that has a smaller training dataset. This is known as Transfer Learning. After I have created the

base model, I have used transfer learning method, where I have used a pretrained model *ResNet50* and trained it using our data.

Project Implementation

For implementing this project, I consider following to be the milestone steps:

- Resizing Images
- Split data into train, validation and test partitions
- Train baseline model
- Tune Hyperparameter
- Train a pre-trained model with current data
- Generate Model motoring and profiling output
- Deploy model as Sagemaker endpoint
- Perform prediction

I have resized the images using a python function before splitting them into partitions. The following function was used for this purpose:

```
from PIL import Image
from os import listdir
import os.path



def resize_image():
   dir_path = "./train_data"
   resized_img_dir = "./train_data_resize/"

   if not os.path.exists(resized_img_dir):
       os.makedirs(resized_img_dir)

   for i in ['1', '2', '3', '4', '5']:
       if not os.path.exists(os.path.join(resized_img_dir, i)):
           os.makedirs(os.path.join(resized_img_dir, i))
       for filename in os.listdir(os.path.join(dir_path, i)):
           if os.path.isfile(os.path.join(dir_path, i, filename)):
               image_path = os.path.join(dir_path, i, filename)
               resized_image_path = os.path.join(resized_img_dir,i,filename)
               img = Image.open(image_path).convert('RGB')
               resized_img = img.resize((224,224),Image.BILINEAR)
               resized_img.save(resized_image_path)
```

After resizing and splitting the data I have uploaded them into the AWS S3 bucket ***inventory-monitoring-1*** and defined the data channels to be used by DataLoaders.

After this step, I have trained the model in Sagemaker using a training script *train.py.* The *Baseline model outcome* table in the previous section outlines the model result.

Following on from here, I have used a pre-trained model *ResNet50* in order to perform hyperparameter tuning using the script *hpo.py*. The hyperparameter search space was as follows:

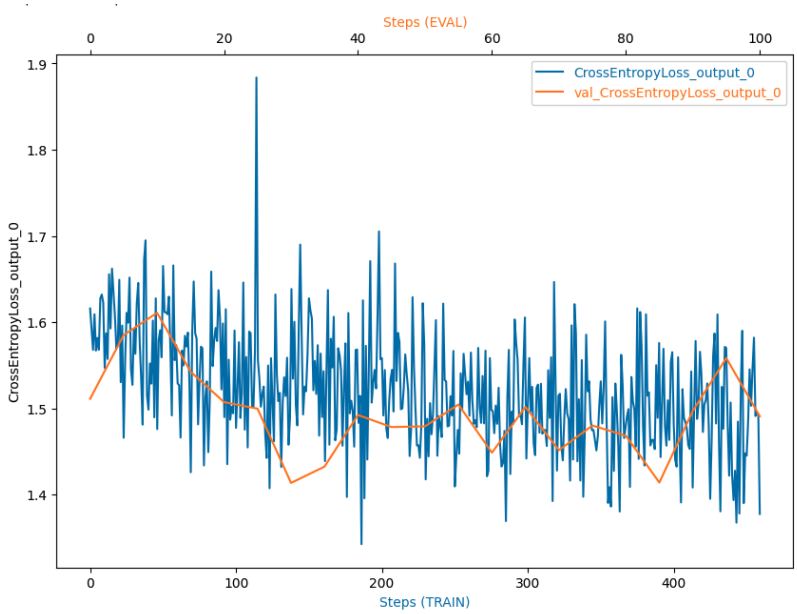
```
hyperparameter_ranges = {
    "lr": ContinuousParameter(0.001, 0.1),
    "momentum": ContinuousParameter(0.0, 1.0),
    "batch-size": CategoricalParameter([32, 64, 128, 256]),
}
```

Following was the best hyperparameter result:

```
{'_tuning_objective_metric': '"average test loss"',
 'batch-size': '"64"',
 'lr': '0.002855165887726784',
 'momentum': '0.8531139470926627',
 'sagemaker_container_log_level': '20',
 'sagemaker_estimator_class_name': '"PyTorch"',
 'sagemaker_estimator_module': '"sagemaker.pytorch.estimator"',
 'sagemaker_job_name': '"pytorch-training-2023-03-22-06-33-14-255"',
 'sagemaker_program': '"hpo.py"',
 'sagemaker_region': '"us-east-1"',
 'sagemaker_submit_directory':
'"s3://sagemaker-us-east-1-885075329292/pytorch-training-2023-03-22-06-33-14-255/source/sourcedir.tar.gz"'}
```

Using these values above, I have trained a ResNet50 model to yield the final model for this project. During this stage of the project I have also included Sagemaker profiling and debugger configuration. The script used for this step is called *train_with_profile.py.*

Following is a visualization of the Model Monitor data outputs:



Project Outcome

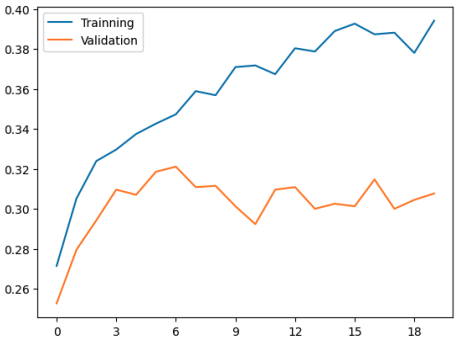
The outcome of the project is not satisfactory. Even after utilizing transfer learning the accuracy was not up to the mark. Following is a statistics of the outcome:

Metric	Value
Average Testing Loss	1.4395427166426014
Testing RMSE	41.65928522369212
Testing Accuracy	33.14176245210728

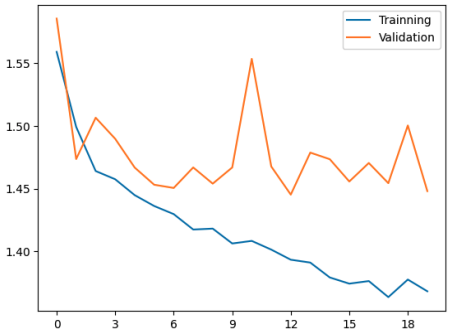
Pretrained model outcome

Where it is evident that performance has gone a little but still only 33% accuracy could be obtained.

The line charts below, depicts the trend of the pre-trained model's accuracy and loss for training and validation datasets:



Accuracy

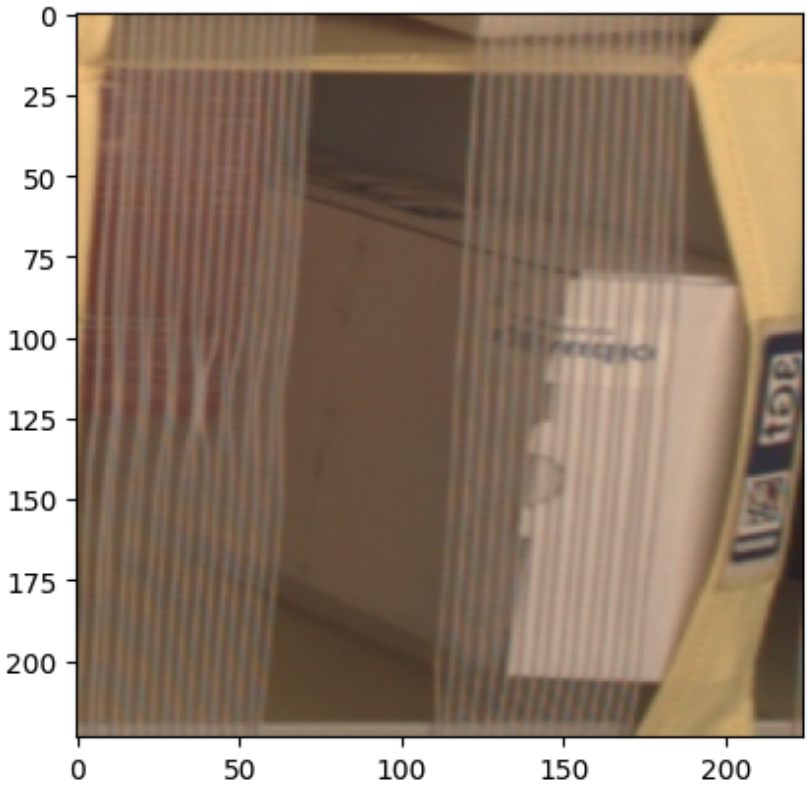


Loss

Following is an example of a successful prediction:

```
Actual Label: 2
Predicted Label: 2
```

[86]: `<matplotlib.image.AxesImage at 0x7f90b9a5e910>`



Conclusion

The images of the bins are not very high quality. The tape in the front of the bins makes it harder for even humans to understand. Better image quality would definitely improve the result.

References:

Bertorello et al. (2018)

Rodriguez Bertorello, Pablo Martin and Sripada, Sravan and Dendumrongsup, Nutchapol, Amazon Inventory Reconciliation Using AI (December 15, 2018). Available at SSRN: https://ssrn.com/abstract=3311007 or http://dx.doi.org/10.2139/ssrn.3311007

Amazon Bin Image Dataset was accessed on 21st March 2023 from https://registry.opendata.aws/amazon- bin-imagery.