Montserrat Maceda Noda
November 23, 2024

Inventory Monitoring at Distribution Centers (Amazon Bin Image counting)

Definition

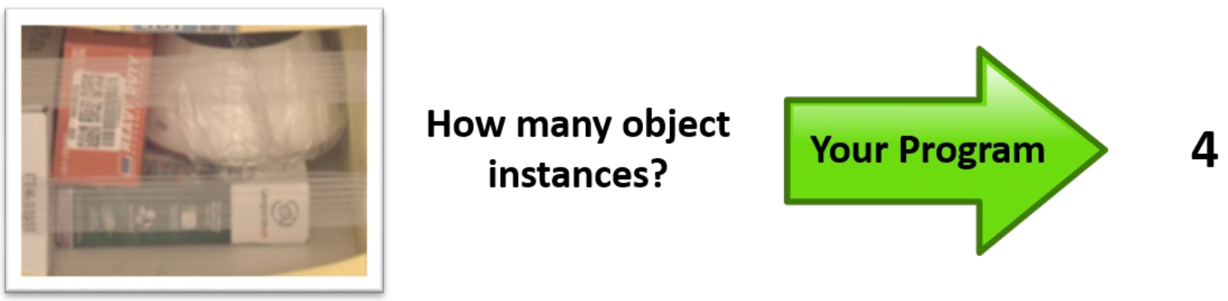
- Project Overview

Distribution centers often use robots to move objects as a part of their operations. Objects are carried in bins which can contain multiple objects. In this project, I will have to build a model that can count the number of objects 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.
To build this project I will use AWS SageMaker and good machine learning engineering practices to fetch data from a database, preprocess it, and then train a machine learning model. This project will serve as a demonstration of end-to-end machine learning engineering skills that I have learned as a part of this nanodegree.

- Problem Statement

This is a image classification problem like other we've seen throught this program, but in this case the result of the image classification must be the total number of different items in a bin image.

Example:



- Metrics

For this project we will use two metrics to evaluate the model: **Accuracy** and **RMSE**(Root Mean Square Error), to evaluate the distance from the real value of number of items in an image to the model predicted one.

Analysis

- Data Exploration

We will be using the **Amazon Bin Image Dataset**. The dataset contains more than 500,000 images of bins containing one or more objects (536435 in total). For each image there is a metadata file in json format containing information about the image like the number of objects, it's dimension and the type of object.

Example:

```
{
   "BIN_FCSKU_DATA": {
      "B00CFQWRPS": {
         "asin": "B00CFQWRPS",
         "height": {
            "unit": "IN",
            "value": 2.399999997552
         },
         "length": {
            "unit": "IN",
            "value": 8.199999991636
         },
         "name": "Fleet Saline Enema, 7.8 Ounce (Pack of 3)",
         "normalizedName": "(Pack of 3) Fleet Saline Enema, 7.8 Ounce",
         "quantity": 1,
         "weight": {
            "unit": "pounds",
            "value": 1.8999999999999997
         },
         "width": {
            "unit": "IN",
            "value": 7.199999992656
         }
      },
      "ZZXI0WUSIB": {
         "asin": "B00T0BUKW8",
         "height": {
            "unit": "IN",
            "value": 3.99999999592
         },
         "length": {
            "unit": "IN",
            "value": 7.899999991942001
         },
         "name": "Kirkland Signature Premium Chunk Chicken Breast Packed in Water, 12.5 Ounce, 6 Count",
         "normalizedName": "Kirkland Signature Premium Chunk Chicken Breast Packed in Water, 12.5 Ounce, 6 Count",
         "quantity": 1,
         "weight": {
            "unit": "pounds",
            "value": 5.7
         },
         "width": {
            "unit": "IN",
            "value": 6.49999999337
         }
      },
      "ZZXVVS669V": {
         "asin": "B00C3WXJHY",
         "height": {
            "unit": "IN",
            "value": 4.330708657
         },
         "length": {
            "unit": "IN",
            "value": 11.1417322721
         },
         "name": "Play-Doh Sweet Shoppe Ice Cream Sundae Cart Playset",
         "normalizedName": "Play-Doh Sweet Shoppe Ice Cream Sundae Cart Playset",
         "quantity": 1,
         "weight": {
            "unit": "pounds",
            "value": 1.4109440759087915
         },
         "width": {
            "unit": "IN",
            "value": 9.448818888
         }
      }
   },
   "EXPECTED_QUANTITY": 3
}
```

For this problem, the only field we've to take into account is the “EXPECTED_QUANTITY”.

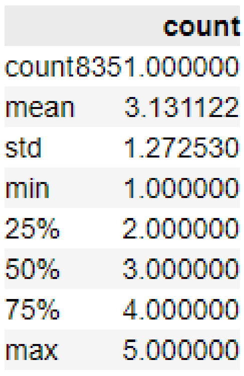
There's two problems found here, the first is the size of the dataset, which can lead to a very time consuming and therefore, costly training process, and the second is the presence of zero items images within the dataset and also images with more than 5 items, which rise the complecity of this problem very much. We also already know for the reports present in explorations of this dataset, like the one found in https://github.com/silverbottlep/abid_challenge, that obtaining a model for the full dataset can be considered a “hard” complecity task and probabily requires additional strategies or algorithms.

For that reason is decided to train the model only for the subset of files of this dataset provided in the starter project in the *file_list.json* file, which contains only 10441 files, distributed in images of bins with 1 to 5 items, properly tagged.

- Exploratory Visualization

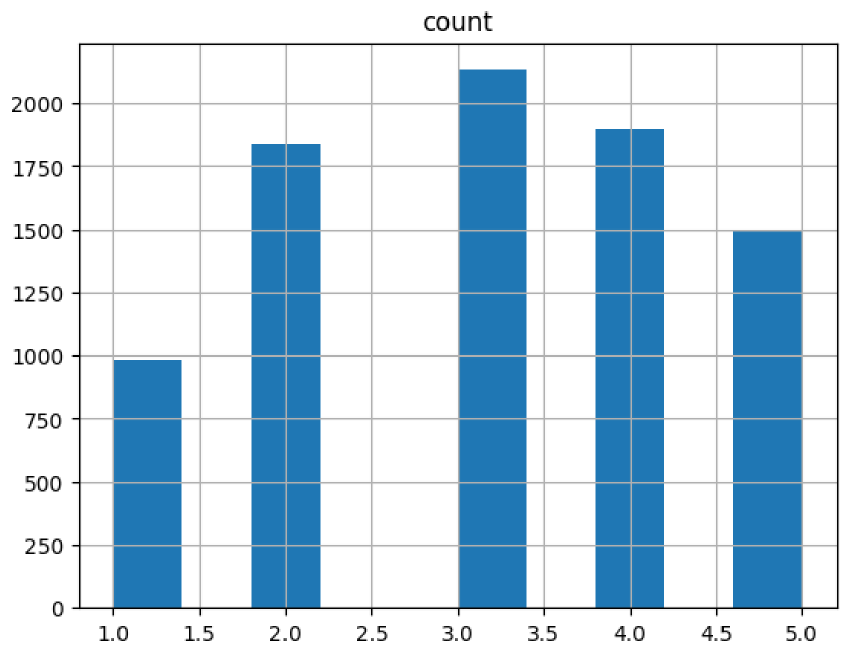
Upon exploration of the subset previosly described, is obtained the basic stats and histogram for the training dataset, after preparing and performing the spliting of the subset into train, validation and test sets with a 80%, 10% and 10% distribution for each.

Stats:



	count
count	8351.000000
mean	3.131122
std	1.272530
min	1.000000
25%	2.000000
50%	3.000000
75%	4.000000
max	5.000000

Histogram:



After checking there is no edge cases and the train set is following a pretty standard gaussian distribution, no further data manipulation/cleaning is performed.

- Algorithms and Techniques

According to the research, the CNN are the best models to solve this problem. I'll use transfer learning over a pre trained image classification model (EfficientNet_B4) and fine tune it for the dataset, applying the techniques learned throught the program.

- Benchmark

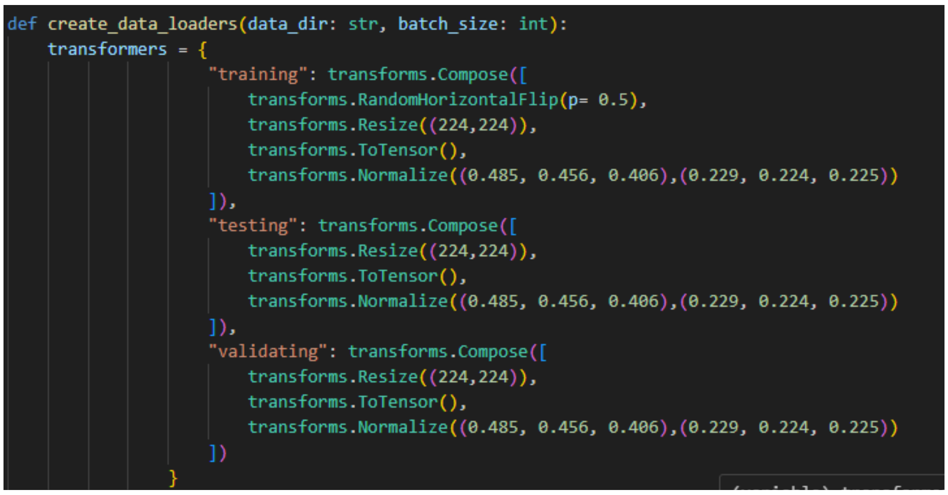
At the proposal is explored two publications about examples usage of this dataset linked in https://registry.opendata.aws/amazon-bin-imagery/, and is stated that with the approach of the subset removing the edge cases, and also taking into account the findings found in the report https://github.com/pablo-tech/Image-Inventory-Reconciliation-with-SVM-and-CNN/blob/master/ProjectReport.pdf, which declared the best model for this problems is using a CNN (a ResNet50 model with Adam optimizer), is achievable to obtain at least a result as good as theirs (about a 50% accuracy), but as we'll see in this report, even taking into account all the recommentations in the report for optimizer, hyperparameters, data aumentation and even with the selection of EfficientNet as the base model, a more advanced model than Resnet, with higher accuracy for smaller datasets, the level of accuracy obtained was 32,79%. I'll discuss later why I think this a decent outcome for the size of the dataset used.

Methodology

- Data Preprocessing

For the training, test and validation sets, is performed a resizing of the images to 224x224 (data aumentation), which is recommended in the report “Image-Inventory-Reconciliation-with-SVM-and-CNN” for better accuracy. For all datasets, also is perfomed a normalization for zero mean and a horizontal flip data aumentation for the training set only.

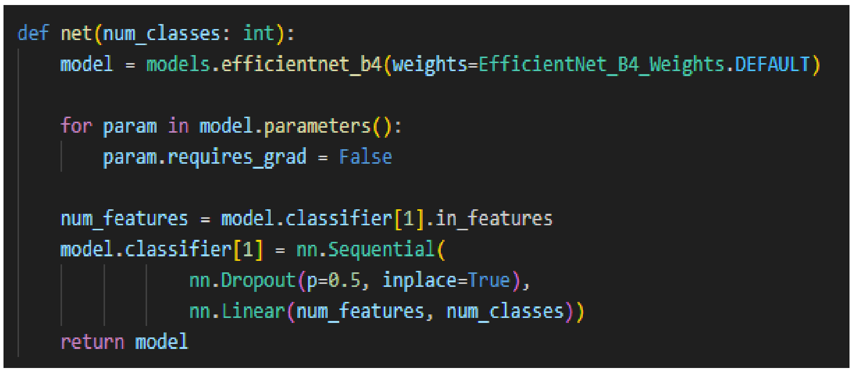
The transformations performed in code are shown below:



```python
def create_data_loaders(data_dir: str, batch_size: int):
    transformers = {
                    "training": transforms.Compose([
                        transforms.RandomHorizontalFlip(p= 0.5),
                        transforms.Resize((224,224)),
                        transforms.ToTensor(),
                        transforms.Normalize((0.485, 0.456, 0.406),(0.229, 0.224, 0.225))
                    ]),
                    "testing": transforms.Compose([
                        transforms.Resize((224,224)),
                        transforms.ToTensor(),
                        transforms.Normalize((0.485, 0.456, 0.406),(0.229, 0.224, 0.225))
                    ]),
                    "validating": transforms.Compose([
                        transforms.Resize((224,224)),
                        transforms.ToTensor(),
                        transforms.Normalize((0.485, 0.456, 0.406),(0.229, 0.224, 0.225))
                    ])
                }
```

- Implementation

The model is implemented as shown in the following code:



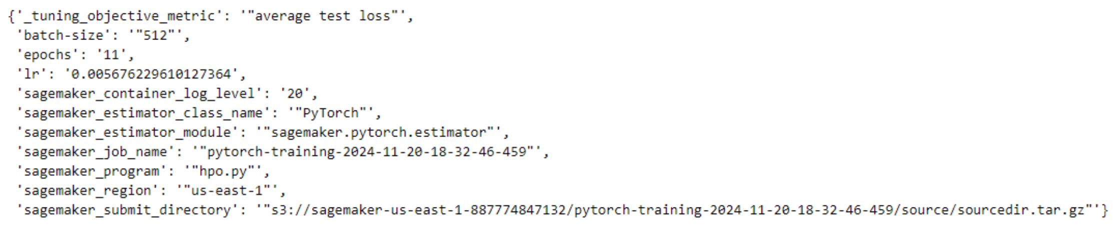
```python
def net(num_classes: int):
    model = models.efficientnet_b4(weights=EfficientNet_B4_Weights.DEFAULT)

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

    num_features = model.classifier[1].in_features
    model.classifier[1] = nn.Sequential(
            nn.Dropout(p=0.5, inplace=True),
            nn.Linear(num_features, num_classes))
    return model
```

The Pytorch implementation of EfficientNet B4 is used, and initialised with the default pre trained weights. We can see then is connected a final layer to the model that classifies the features of the net to the different classes present in our dataset (in this scenario 5).

The optimizer selected is Adam, and the loss function is cross entropy loss (even if we are measuring as metric RSME too, this loss function performs better with this kind of image classification problems).

To fine tune this model, is perfomed a hyperparameter tuning step previous to the training. The hyperparameters chosen are: epochs, batch size and learning rate.

The results of this process are shown below:



```
{'_tuning_objective_metric': '"average test loss"',
 'batch-size': '"512"',
 'epochs': '11',
 'lr': '0.005676229610127364',
 'sagemaker_container_log_level': '20',
 'sagemaker_estimator_class_name': '"PyTorch"',
 'sagemaker_estimator_module': '"sagemaker.pytorch.estimator"',
 'sagemaker_job_name': '"pytorch-training-2024-11-20-18-32-46-459"',
 'sagemaker_program': '"hpo.py"',
 'sagemaker_region': '"us-east-1"',
 'sagemaker_submit_directory': '"s3://sagemaker-us-east-1-887774847132/pytorch-training-2024-11-20-18-32-46-459/source/sourcedir.tar.gz"'}
```

We already know for the papers, that 11 is the best value for epochs, and after that the model begin to overfit, so it seen we have obtained a pretty good values for the params.

Also I've parametrized the value of number of classes, in case I need or want in the future to use this code to perform training with a bigger image dataset, that may include more classes.

After that, the training process is launch using the best hyperparameters found, and multi-instance training for a better performance (2 instances), and making use of the profiler/debugger rules to obtain a full report of the training and be able to detect and analize any problems found. This report can be found in the folder *starter\ProfilerReport\profiler-output* of this project.

- Refinement

This report and project shows the first version, which became the final version too, for obtaining the best results. At first I was worried for the accuracy obtained, and tried to improve the results in several ways:

- Applied several data augmentation techniques to the train images set, like converting to grayscale randomly with *RandomGreyScale* or changing randomly the brightness, contrast, saturation and hue with *ColorJitter.*
- Increased the epoch range to explore in the hyperparameter tuning process.
- Amplified the last layer of the net model, following the example code found here (is not the same problem I'm triyng to solve here, since there only detects empty/not empty bin images, but also works with EfficientNet)

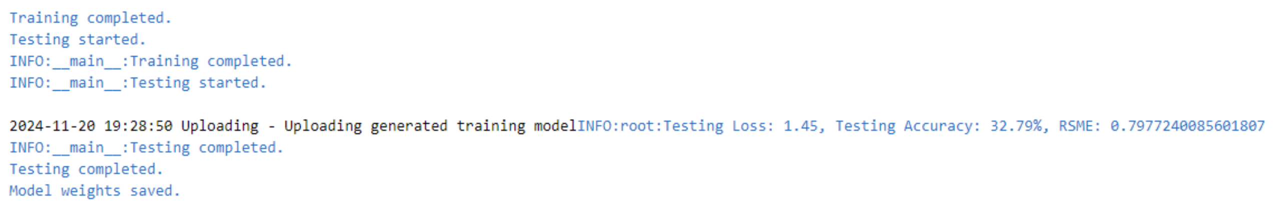
None of these methods worked in the hyperparemeter tuning and training process to improve the accuracy of the model, in fact all runs ended with a lower accuracy to the version presented here (all runs ended with around 30% acurracy).

I've included for reference the scripts used with this changes in the project, but reverted the final version to the previous one.

Results

- Model Evaluation and Validation

The final result of the training are shown below:

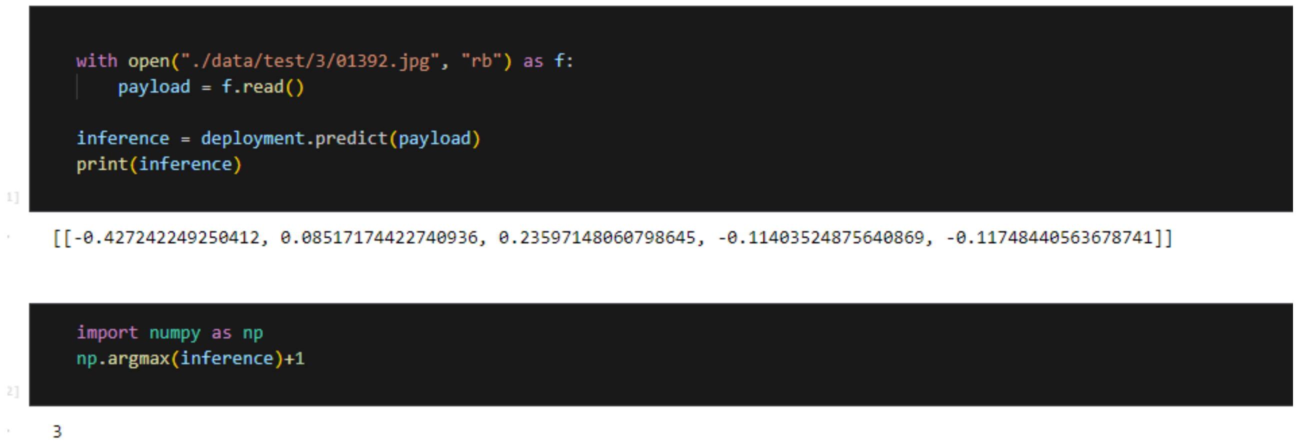


```
Training completed.
Testing started.
INFO:__main__:Training completed.
INFO:__main__:Testing started.

2024-11-20 19:28:50 Uploading - Uploading generated training modelINFO:root:Testing Loss: 1.45, Testing Accuracy: 32.79%, RSME: 0.7977240085601807
INFO:__main__:Testing completed.
Testing completed.
Model weights saved.
```

As we can see our final test accuracy porcentage is **32.79%,** as is quite below from the results obtained by the "Image-Inventory-Reconciliation-with-SVM-and-CNN" project, which obtained around 50-55% of accuracy.
The final **RSME** value obtained is **0,79** though, which is a good result if we compare again the ones obtained in the "Image-Inventory-Reconciliation-with-SVM-and-CNN" project, which obtained around 0.98-0-91.

In the next section I'll try to justify the reasons for that metrics, and why is still a good result for the small size of the dataset used, and how it could be improved.

Also, to validate the model obtained and see if we can perform a inference correctly, an endpoint with it is deployed and tested using one random image from the test set.



```
with open("./data/test/3/01392.jpg", "rb") as f:
    payload = f.read()

inference = deployment.predict(payload)
print(inference)
```

```
[[-0.427242249250412, 0.08517174422740936, 0.23597148060798645, -0.11403524875640869, -0.11748440563678741]]
```

```
import numpy as np
np.argmax(inference)+1
```

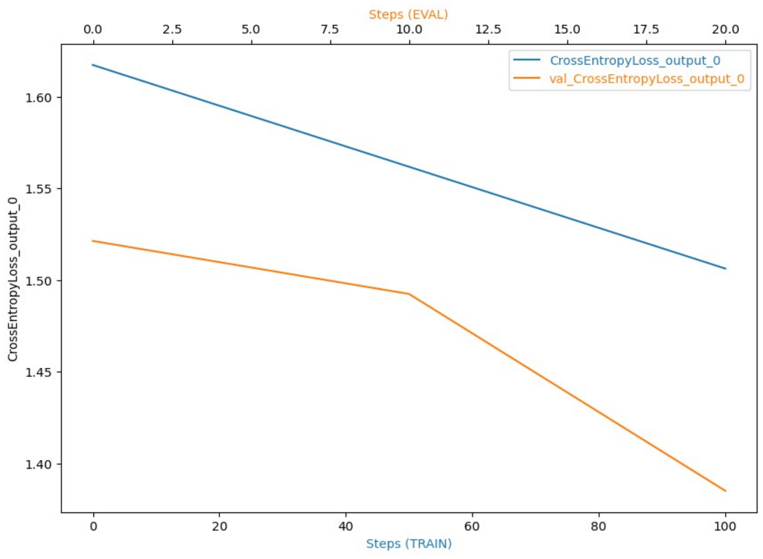
```
3
```

In this case the results of the inference is correct, as the image belongs to the category "3" meaning there's 3 items present in the image and the array of probabilities returned has the bigger value in the third position.

- Justification

In general, though it may seem that the model is not good enough, (at least not good enough still for a real world usage), we have obtained a reasonable accuracy for the dataset used:

1. We've obtained a **32.79%** against the 50-55% desired, but although we've implemented similar techniques to resolve this problem, this “best case scenario” is obtained only with a much bigger dataset that the one used in this problem (150000 images against 10441 used here). That means we've obtained over 17% acurracy less for a 10% of the images. So one of the solutions to obtain a better result is to **train with a larger dataset.**
2. Also the value obtained for the **RSME** metric is actually pretty good (0,79) and improves the ones obtained in the benchmark project (0,98).
3. The plot obtained for the “cross entry loss” during the training process shows the loss is decreasing all the time in a progressive way and is clearly in a decreasing trend still when the process end (it shows no “stagnation” signs or spikes)



4. As stated in the “Image-Inventory-Reconciliation-with-SVM-and-CNN”, many of the images in the dataset shown occluded items or are difficult to classify even for a human. So another solution to improve this is to perform **a cleaning of those images**.