Define:

Domain Background:

The domain background for the Amazon inventory monitoring project is related to supply chain management and inventory control. Amazon, which is one of the world's largest and most technologically advanced corporations, inventory monitoring is crucial to ensure that products are available for customers to purchase and that there is no excess or shortage of inventory.

If there is any company that is capable of deploying robotics at scale it is Amazon. Over the long run this could generate billions of dollars in increased revenue and reduced cost. Amazon has a long history of being a leader in AI and Machine Learning so it is very realistic a model like this could be deployed at Amazon.

Problem Statement:

Specifically the problem is to use computer vision to count the number of items in a small to mid sized cardboard box. In the training data set given in the starter notebook for this project the amount of items can be 1,2,3,4 or 5. The ability of a Deep Learning based Computer Vision model to correctly classify the amount of items in a box is quantifiable, measurable, and replicable.

Benchmark Model:

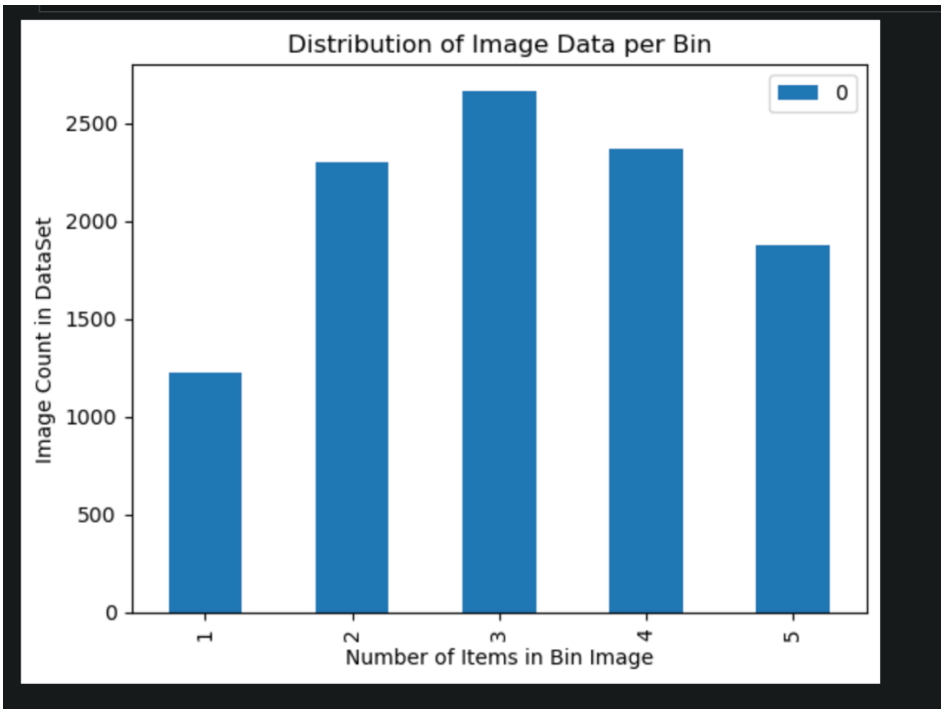
Given we have 5 classes we can start with a simple baseline of 20% accuracy from random guessing. This is a standard practice in classification problems. We could also use the resnet50 as a baseline and use a more modern and powerful model like the EfficientNet as another option. Given the resources for this project we can only run a small amount of training runs so we will use 20% from random guessing as the baseline.

Evaluation Metrics:

We will measure the success of the computer vision model using the % accuracy to predict each class label: 1,2,3,4 or 5.

Analyze:

The notebook features sections on Data Exploration and Exploratory Visualization. Below is a screenshot of the distribution of bin item counts here below:



Algorithms and Techniques:

In this project we use a ResNet50 deep learning model.

ResNet-50 is a deep convolutional neural network architecture that was introduced by Microsoft Research in 2015. It is a variant of the ResNet architecture, which stands for "Residual Network." ResNet-50 is widely used for image classification tasks and has achieved state-of-the-art performance on various benchmark datasets.

The main idea behind ResNet-50 is to address the problem of vanishing gradients in deep neural networks. As the network becomes deeper, the gradients can become very small, making it difficult for the network to learn effectively. ResNet-50 introduces a concept called "skip connections" or "identity mappings" to overcome this issue.

In ResNet-50, the network is composed of multiple residual blocks. Each residual block consists of a series of convolutional layers followed by batch normalization and ReLU activation. The key innovation in ResNet-50 is the introduction of skip connections that allow the network to learn residual mappings.

A skip connection allows the input of a layer to be directly added to the output of a later layer. This means that the network can learn to make small adjustments to the input rather than trying to learn the entire mapping from scratch. These skip connections help in propagating the gradients effectively, enabling the network to learn deeper and more complex representations. ResNet-50 has 50 layers, hence the name. It has a specific architecture with convolutional layers, pooling layers, and fully connected layers. The network starts with a convolutional layer followed by max-pooling. Then, it has four stages, each containing multiple residual blocks. The final stage ends with global average pooling and a fully connected layer for classification. ResNet-50 has been successfully used in various image classification tasks, including the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where it achieved top performance. It has also been used in transfer learning scenarios, where the pre-trained ResNet-50 model is fine-tuned on a smaller dataset for a specific task, allowing for faster and more accurate training.

Implement:

For the data preprocessing and implementation code we really on the SageMaker API, Pytorch and the Torchvision package. Specifically, we implement the below features from Torchvision:

1. **Loading the Dataset**: In the project we use the **torchvision.datasets.ImageFolder**(root=train_data_path, transform=train_transforms)

   In the train2.py script to load are data from S3 for training.

2. **Data Transformation**: Once the dataset is loaded, you can apply various transformations to preprocess the images. In the create_data_loaders function we perform the below.

```
# Define transformations for the training, validation, and test sets
  train_transforms = transforms.Compose([
    transforms.RandomResizedCrop((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    #transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
  ])
```

3. **Creating Data Loaders**: The torch.utils.data.DataLoader class can be used to create data loaders from the preprocessed dataset. In our script we create the final data loaders using the ImageFolder object which incorporates our data from S3 and our transforms.

```
train_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size, shuffle=True)
```

4. **Using Pretrained Models**: In our script the net() function creates an instance of a pretrained resnet model and also adds some additional layers to the neural network:

Results:

Training loss on the initial training run prior to using Hyperparameter Tuning:

```
Model training beginning

Epoch 0, Phase train

[2024-05-05 18:31:29.516 algo-1:27 INFO hook.py:413] Monitoring the collectio
ns: losses

[2024-05-05 18:31:29.520 algo-1:27 INFO hook.py:476] Hook is writing from the
hook with pid: 27

train loss: 1.9968, acc: 0.2478, best loss: 1000000.0000

Epoch 0, Phase valid
```

Training Run using best Hyperparameters from Hyperparameter Tuning Job:

```
train loss: 1.6898, acc: 0.2630, best loss: 1000000.0000

Epoch 0, Phase valid

valid loss: 1.5586, acc: 0.2920, best loss: 1.5586

Epoch 1, Phase train

train loss: 1.5092, acc: 0.2925, best loss: 1.5586

Epoch 1, Phase valid

valid loss: 1.5018, acc: 0.3026, best loss: 1.5018

Model testing beggining

Testing Model on Whole Testing Dataset

Testing Accuracy: 0.3087248322147651, Testing Loss: 1.4945584300
151835

Downloading: "https://download.pytorch.org/models/resnet50-19c8e357.pth" to /
root/.cache/torch/hub/checkpoints/resnet50-19c8e357.pth
```

```
#015  0%|          | 0.00/97.8M [00:00<?, ?B/s]#015  9%|▉         | 8.50M/97.
8M [00:00<00:01, 89.1MB/s]#015 18%|▏▏        | 17.2M/97.8M [00:00<00:00, 90.4
MB/s]#015 27%|▏▏▏       | 26.0M/97.8M [00:00<00:00, 91.0MB/s]#015 35%|▏▏▏▏
| 34.7M/97.8M [00:00<00:00, 91.2MB/s]#015 44%|▏▏▏▏▏     | 43.4M/97.8M [00:00<0
0:00, 91.3MB/s]#015 53%|▏▏▏▏▏▏    | 52.2M/97.8M [00:00<00:00, 91.5MB/s]#015 62
%|▏▏▏▏▏▏▏   | 60.9M/97.8M [00:00<00:00, 91.6MB/s]#015 71%|▏▏▏▏▏▏▏▏  | 69.7M/97.
8M [00:00<00:00, 91.8MB/s]#015 80%|▏▏▏▏▏▏▏▏▏  | 78.5M/97.8M [00:00<00:00, 91.9
MB/s]#015 89%|▏▏▏▏▏▏▏▏▏▏ | 87.3M/97.8M [00:01<00:00, 92.0MB/s]#015 98%|▏▏▏▏▏▏▏▏
▏▏▏| 96.1M/97.8M [00:01<00:00, 91.9MB/s]#015100%|▏▏▏▏▏▏▏▏▏▏| 97.8M/97.8M [00:
01<00:00, 91.4MB/s]

2024-05-05 17:55:33,006 sagemaker-training-toolkit INFO     Reporting trainin
g SUCCESS
```

We can see that through hyperparameter tuning we are able to increase accuracy from 25% training accuracy to 30% testing accuracy on the full out of sample dataset.

Conclusions:

Unfortunately we were only able to have 30% accuracy on this dataset even after completing extensive analysis, data preprocessing and hyperparameter tuning. But with more resources I think there are many steps we could take to works towards a stronger production quality model.

In the Udacity guidelines for the Image Bin Classification project the below is emphasized:

“The cell below creates a folder called train_data, downloads training data and arranges it in subfolders. Each of these subfolders contain images where the number of objects is equal to the name of the folder. For instance, all images in folder 1 has images with 1 object in them. Images are not divided into training, testing or validation sets. If you feel like the number of samples are not enough, you can always download more data (instructions for that can be found here).

However, **we are not accessing you on the accuracy of your final trained model, but how you create your machine learning engineering pipeline.**”

Do to the resources offered for this project it is meant to be judged based on pipeline design not on final accuracy.

Insufficient Training Data: To reduce AWS credits usage we only stored and trained on 10,000 samples of the full 500,000 sample dataset. We would expect significant improvement with using the full dataset as 10,000 images in quite a small sample in a deep learning based computer vision project. In general more and higher quality data is a key to better performance in machine learning and specifically deep learning.

More Epochs on Finetuning:

To reduce training time we used 2 epochs of finetuning. We would likely to continue to see improvement with 4-5 epochs.

Test Different models:

With more time and resources I would choose to test a variety of different pretrained computer vision models.

References:

This final report references my Project Proposal which is also found in the github repo, Udacity course material, previous projects completed as part of this nanodegree and uses Udacity GPT.