**AWS Machine Learning Engineer Nanodegree Capstone Project**

By Araz Sharma 22nd February 2022

1. **DEFINITION**

**Project Overview:**

# This Project is on “Inventory Monitoring at Distribution Centres”. This problem has originated from the popular field of Robotics, and its applications in the industry, like that of Inventory Monitoring and Supply Chain Functions. A lot of Corporations, which handle physical cargo and deal with supply chain of any kind of goods have tried to bring in automation to make these processes more efficient and accurate. A great example of this is Amazon, who is one of the biggest hubs of delivery of all kinds of goods. These goods are often stored in big warehouses. Since the quantity of these items are in a huge amount, to physically do inventory monitoring would require both large and intelligent human resources, which are both expensive and prone to errors.

# This is where robots come in to help in Inventory Monitoring. They can be trained with Machine Learning Models, to perform tasks like Object Detection, Outlier & Anomaly Detection and much more. Once trained, these models are scalable, and can be deployed at a low cost for usage in actual warehouses and distribution centres on industry level robots.

**Problem Statement:**

# As mentioned above in the domain, distribution centres often have robots which carry objects. These objects are present in bins, and for our problem, each bin can contain 1-5 objects.

# The problem we aim to tackle in this project is to count the number of items present in the bin. This is a Classification task, of classifying number of items in 1 – 5, given an input image. This is a worthwhile problem to solve, for it has immense real world applications. If we can develop a model, which can take in a picture of a bin, and accurately return the number of objects present in that, we could solve & thus fully automate one crucial step in the Inventory Management process!

# Datasets and Input:

# The dataset used in this problem is the open source Amazon Bin Image Dataset. This dataset has 500,000 images of bins containing one or more objects present in it. Corresponding to each image, is a metadata file, which contains information about the image, like the number of objects it has, the dimensions and type of objects. For our problem statement, we only need the total count of objects in the image. An example of an image & the corresponding metadata file is shown as below: (Source [Reference [1]](https://github.com/awslabs/open-data-docs/tree/main/docs/aft-vbi-pds))

# The “EXPECTED\_QUANTITY” field tells us the total number of objects in image.

# A picture containing text, indoor Description automatically generatedTable Description automatically generated

# Solution Strategy:

# To solve our problem statement, we will use the task of Computer Vision, to come up with a Machine Learning Model, which given an image from our dataset, can identify the number of objects present in it. Essentially, we would be using Multi-Class Image Classification, with Number of Objects from 1-5 as an individual class.

# To do this, we can make use of Convolutional Neural Networks, which are a State of the Art technique for Image Recognition tasks. We will leverage Pre-trained models with Transfer Learning to solve the problem.

# The end solution should be a model, which can take in an input image from the Amazon Bin Image dataset, and accurately output the number of objects it thinks are present in that image (from 1-5).

# Metrics for Evaluation:

# To evaluate our model, we need some good metrics, which align with the problem statement. The metrics must be mathematically sound and we should be able to optimise our model for them.

# Since we have a Classification Task, we can use Accuracy, Recall, Precision and F1 scores as our metrics. These can be for the overall data, and also class-wise, to identify if a model is doing better on a particular class, or has a high bias for one of them.

# Since we have Multi-Class classification, the definition of Precision & Recall must be clearly understood. Let’s assume we have 3 classes: A, B & C.

# With Multi-Class classification, we have precision and recall for each individual class. For example, metrics for Class A will be defined as follows:

# Precision: (‘#’ stands for Number of)

# #Correctly Predicted Class A Instance / #All Instance predicted as Class A

# Recall: (‘#’ stands for Number of)

# #Correctly Predicted Class A Instances / #Total Class A Instances

# F1 Score: 2\*Precision\*Recall / (Precision + Recall)

# So to evaluate our model, we will see per class metrics of Precision, Recall and F1 Score, and see the overall accuracy of the model, which includes all classes, and is defined as:

# Accuracy: #Total Correctly Predicted Instances / #Total Instances

# [Reference [3]](https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2) for Precision, Recall and F1 for multi-class classification

# Analysis

# Data & Distribution Exploration:

# The Notebook ‘Create\_Data\_Capstone.ipynb’ implements Data Exploration & Visualisation for our Amazon Bin Image Dataset.

# The ‘file\_list.json’ provided by Udacity, has a subset of the Amazon Bin Image Dataset.

# This subset of data has 5 classes, corresponding to number of objects present in the bin: 1, 2, 3, 4 & 5. The total number of images in this subset are: 10,441

# The Class Wise Distribution is as follows: (Plotted with code)

# Chart, bar chart Description automatically generated

# From this subset, we took a Train-Test-Validation Split as follows:

# We randomly sample 100 Images from each class as Test Data

# For the remaining data, we do a 80-20 Train-Validation Stratified Split. Stratification is important to preserve the same distributions in both sets.

# After this split we see:

# Chart, pie chart Description automatically generated

# We also explore the Class Wise Distribution in Training and Validation Sets:

# Chart, bar chart Description automatically generated

# Chart, bar chart Description automatically generated

# However, we see a big imbalance in Class Distribution! From the Above Distributions, we see a big difference in the Data in Class 3 vs All other classes

# Since there is an Imbalance of Classes, there is a high chance for a model to learn to only predict Majority Class. We need to test this data with a basic model, to see if the model is able to learn other classes or not. So we fine-tuned a pre-trained Resnet50 model for 5 epochs and observed:

# Imbalance is leading to Model only predicting Class 3

# Loss decreases since Class 3 is dominating, but final test accuracy is poor, since all classes balanced (100 each)

# We need to balance out the data

# Using the ‘Get\_More\_Data\_Script.ipynb’ we balance out the data (check in Create\_Data\_Capstone for implementation details). The new distributions are:

# Chart, pie chart Description automatically generated

Chart, bar chart

Description automatically generated

Chart, bar chart

Description automatically generated

Now the dataset class distribution is not biased towards a single class. This dataset is now used for Model Training and Evaluation. Further transformations of data are covered in data pre-processing steps ahead.

**Visualizations:**

In the Notebook, we have taken 3 examples per class to check for:

1. To validate that Image has that many objects as the class.
2. Try to analyse to see what kind of transformations can be applied here.
3. Can we as humans identify them with ease? If this is a difficult task for Humans itself to analyse, then Models may not perform as well as we expect them to.

A picture containing text

Description automatically generated

Class 1: One Object in the Bucket

- We see that Class 1 Images are Valid

- **They seem easy to identify for Humans**

- The Bin has a net, which may make it difficult

to see the objects sometimes

Class 2: Two Objects in the Bucket

* We see that Class 2 Images are Valid
* **There are cases when 2 Objects are very close to each other and it seems they are one object**
* The Bin has a net, which may make it difficult to see the objects sometimes

A picture containing text, indoor

Description automatically generated

Class 3: Three Objects in the Bucket

* After carefully scrutinizing the Images, we see Class 3 Images are Valid
* **They are difficult to identify for even Humans, and Model may not perform well on this Class**
* The Bin has a net, which may make it difficult to see the objects sometimes

A picture containing text, indoor

Description automatically generated

Class 4: Four Objects in the Bucket

* Needed to check images very carefully, and had to inspect more images to confirm that the Class 4 Images are Valid
* **They are not easy to verify by Humans, and Model may not perform well on them**
* The Bin has a net, which may make it difficult to see the objects sometimes

A picture containing text, indoor, shelf

Description automatically generated

Class 5: Five Objects in the Bucket

* Images appear very cluttered with objects and it is difficult to make out if there are 5 objects always
* **Extremely hard to classify by Humans**
* The Bin has a net, which may make it difficult to see the objects sometimes
* Here there's a possibility **Model may directly learn for very cluttered to predict 5, but not actually from that it detects 5 separate objects**

A picture containing text, cage, train

Description automatically generated

**From all pictures, we see that we can resize them and horizontally flip them.**

**Benchmark:**

As discussed in the Proposal, since this particular task has no implementations, we created 2 Benchmarks, as implemented and showed in the Benchmark\_Model.ipynb notebook.

**Benchmark I:** Custom CNN Architecture

Batch Size 32 and Learning Rate: 0.005 Epochs = 5

**Text

Description automatically generated**

Since we had limited amount of resources from the Udacity AWS Gateway, the resource constraints made training a CNN from scratch difficult & posed memory and cost issues, and the Benchmark wasn’t very good –

Graphical user interface

Description automatically generated with low confidence

So we also trained a second model for a Benchmark

**Benchmark II:** Finetuning a Pre-Trained Resnet50 CNN Architecture

Hyper Parameters: Batch Size 64 and Learning Rate: 0.001 Epochs = 5

This resulted in a good Benchmark we wanted to beat:

Graphical user interface

Description automatically generated with low confidence

**Algorithms and Techniques Planned to Use:**

1. **Fine Tuning a Pre-Trained Resnet50 CNN Model with Hyper Parameters from Intuition** – Since we know that this is a Multi-Class Image Classification, leveraging transfer learning can help produce good results
2. **Using Hyper Parameter Tuner in SageMaker** – We can leverage SageMaker’s HP Tuner too find good parameters to train with
3. **Training Model with Best HP from Tuner** – Once we get the best parameters, we can train on them for a longer duration
4. **METHODOLOGY:**

**Data Pre-Processing:**

Once the data is uploaded to S3 Buckets, the Data Loaders help in fetching them, and we can do the transformations as seen fit from Data Analysis with them.

We apply the following Transformations (Training) as seen from the code:

* Resize Image to 224 x 224
* Apply Random Horizontal Flips
* Convert to Tensor
* Shuffle before feeding in Model

Graphical user interface, text, application

Description automatically generated

**Implementation:**

‘Final\_Model\_Train\_Inference.ipynb’ notebook has the detailed implementation, which is well documented, for the Algorithms, Techniques and Stand out Suggestions tried for the Project.

The overall Implementation was consisting of designing and writing the code for the Algorithms and Techniques, and using SageMaker’s Library to use them to train, test and deploy the Models. This was also an iterative process, since any errors in the code would lead to a failed training job, which had to be debugged from the CloudWatch Logs.

**Steps for Implementation:**

I Implementing Training and Evaluation Code in model\_train.py

- Loss and Optimiser to use

- Data Loaders and Transformations

- Metrics for Evaluation

- Enable GPU Training

II Deciding First Model & Parameters to Train

III Refinement: Hyperparameters Search

IV Refinement: Training Model with Best Hyperparameters

V Generating Profiler Report

VI Deploying Endpoint

VII Inference for each Class

1. **Implementing Training and Evaluation Code in model\_train.py**

* We used the CrossEntropyLoss and the Adam Optimiser
* The Data Loader is defined as shown in Data Pre-processing
* The Metrics defined in definition are coded
* GPU support is added in the entire code

1. **Deciding First Model & Parameters to Train**

To begin, we try fine tuning a Pre-Trained Resnet50 Model, as defined in model\_train.py on our Data

From intuition we try the following Hyper Parameters:

* Batch Size: 64
* Learning Rate: 0.005
* Number of Epochs: 5

Graphical user interface, application

Description automatically generated

We see that we are close to our Benchmark scores, by less than ~ 1.2%

1. **Refinement: Hyperparameters Search**

Graphical user interface, text, application, email

Description automatically generated

We launch the Jobs as shown here:

Graphical user interface, application

Description automatically generated

The Jobs succeed in telling the best HP: Batch Size 32, LR: 0.002

Table

Description automatically generated

1. **Refinement: Training Model with Best Hyperparameters**

We perform training with the best HP found above, for 10 epochs. We also add debugger rules and hooks to the Model\_Train.py code to get the Profiler Output and so we can learn from the Profiler Report.

1. **Generating Profiler Report**

We see that the Low GPU Utilisation and Batch Size Rules were triggered.

Graphical user interface, text

Description automatically generated

1. **Deploying Endpoint**

We deploy an endpoint which can be used for Inference

**Graphical user interface, text, application, email

Description automatically generated**

1. **Inference for Each Class**

We use new images for each class, to test from the endpoint, and see the results and reasons for them being classified correctly or not. Check ‘Final\_Model\_Train\_Inference.ipynb’ for the detailed results.

**Refinement Techniques:**

For our Project, we do refinement for results in 2 cases:

1. **Correcting Imbalance of Classes in Dataset:** Since the dataset was imbalanced, it would lead to model predicting the majority class. To avoid that, we wrote a script to fetch new data for Classes 1, 2, 4 and 5 & balanced out the data, for refinement in predictions
2. **From Intuition to Hyper Parameter Search:** We started by guessing hyper parameters from Intuition. Then to refine our results, we used Hyper Parameter Tuner search, to find the good HPs and use them to train the final model.

**Initial Solution from Intuition:**

**Graphical user interface, application

Description automatically generated**

**Final Solution after HP Search:**

**Graphical user interface, text, application, chat or text message

Description automatically generated**

**THUS WE SEE AN IMPROVEMENT IN OUR SOLUTION**

1. **RESULTS:**