Summary

Problem Definition

In this section, the problem will be defined, possible solutions will be presented and a discussion about performance metrics for evaluating the solution will be made.

Problem Introduction

Information always played a great role in management processes, since decisions tend to be as good as the basis upon which they were built. Given the great value of this resource, it was only natural that data utilization would be vastly explored and expanded with time.

One of the main uses of data is the training of several Machine Learning models. Examples of these models can include Natural Language Processing, which allowed the creation of chat bots and translation tools, Computer Vision, responsible for automatic object and person identification and pattern recognition, and many others model types for different purposes.

The problem I intend to tackle is linked to Computer Vision and stock management. Since logistic planning is a hard task by itself, one of the ways data can improve operations is by providing vision to robots used in bin transportation in warehouses. Using Computer Vision algorithms and images provided by Amazon to train a classification machine learning model, we can reduce errors, automate operations and optimize stock placement.

In short, the main problem studied in this project is the creation and training of a Computer Vision model capable of identifying the number of items being carried in transportation bins. That is, to classify images into different classes that are divided between the number of items present in the picture.

Potential Solutions

Solutions outside the realm of Machine Learning can become very time costly, e.g., human employees checking each transport bin throughout the day.

Taking that into consideration, the potential solutions for the problem can be seen as the ones that actually reduce costs and improve efficiency.

In this academic context, we will only look at solutions related to Computer Vision, that is, the different types of image classification model architectures we can use as base for our model.

These architectures differentiate between themselves by how the convolutional and fully connected layers were employed in order to improve a metric of choice (e.g., accuracy). Additionally, we can modify some sections of these pre-built architectures to fit it to our needs.

As PyTorch will be used as base package for model creation and training, the possible model architectures are the ones contained on Torchvision (PyTorch's library with useful tools for Computer Vision).

The model architectures available for utilization are:

- AlexNet
- VGG
- ResNet
- Wide ResNet
- ResNeXt
- SqueezeNet
- DenseNet
- Inception
- GoogleNet
- ShuffleNet
- MobileNet
- MNASNet
- EfficientNet
- RegNet

- VisionTransformer
- ConvNeXt

Details on these architectures' utilization can be found here[1].

Performance Metrics

Performance metrics are values that give us the measure of how well our model is predicting whatever it was trained for.

There are different metrics for different kinds of problems: if our model is configured for doing regression, the metrics used for classification cannot be used and vice-versa. And since the problem presented previously is a classification into classes, the metrics considered for evaluating the model were:

- Accuracy
- Precision
- Recall
- F1-Score

In the following subsections these metrics will be detailed, but first the concept of Confusion Matrix needs to be introduced.

Confusion Matrix

In a classification problem, the training process consists in fitting a model to make predictions that will be compared to a label. This label is the target class that we want our model to output.

That idea might seem quite simple when we are dealing with binary classification, e.g., if an image is showing a dog or a cat. For this binary example, we will define that dog predictions are 1 (positive for dog) and cat predictions are 0 (negative for dog, therefore cat).

The possible outcomes in this scenario are:

- Prediction for a dog picture is 1: **true positive (TP)**, the model predicted that the image **was** a dog and got it right.
- Prediction for a cat picture is 0: **true negative (TN)**, the model predicted that the image **was not** a dog and got it right.
- Prediction for a cat picture is 1: **false positive (FP)**, the model predicted that the image was a dog when it was a cat.
- Prediction for a dog picture is 0: **false negative (FN)**, the model predicted that the image was a cat when it was a dog.

Having these predictions and labels at hand, it is possible to display the results of true and false positives and negatives as a Confusion Matrix. For this binary case, this matrix will have the following form:

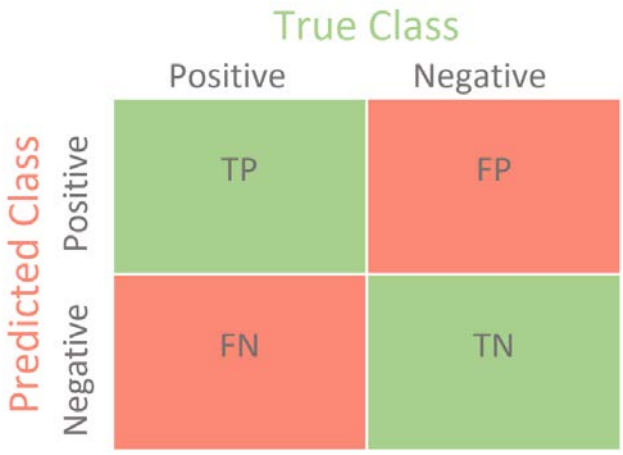

Figure 1 - Binary Confusion Matrix.

source: https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826

In multi-class classification problems, it is not that simple. We will use another example to better show the creation of this matrix for more than two classes.

Consider an image classification problem where we want to predict if an image is of an apple, orange or mango. Our classification method will output one of these classes for each image it analyzes, that is, it will say that an image is **positive for a class**. Consequently, this image is a **negative for the other classes**.

When assembling the Confusion Matrix for this problem, one possible outcome is:

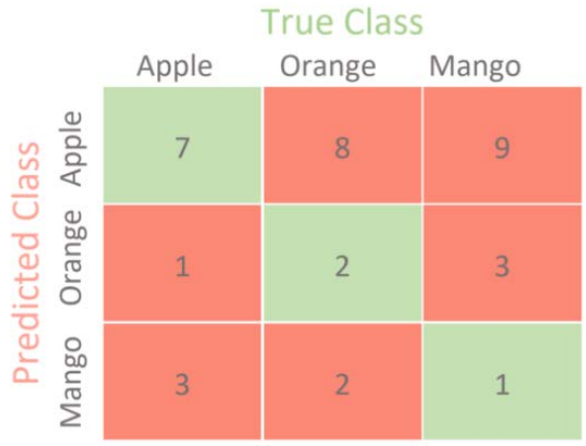

Figure 2 - Multi-Class Confusion Matrix

source: https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826

In order to transpose the idea of TP, TN, FP and FN we will take the different classes into consideration.

Looking at the Apple class, we have that:

- The number of TP is 7.

- The number of TN is 8 (5 Orange predictions + 3 Mango predictions). This number includes all the times the model said an image was not an apple and got it right. Since we are looking only at the Apple class, it does not matter whether the predictions for Orange and Mango were correct, the point is that they were **negative** for Apple.
- The number of FP is 17 (8 Orange images + 9 Mango images)
- The number of FN is 4 (1 Orange prediction + 3 Mango predictions)

The method of segregating each class during performance metric evaluation is called One-Vs-Rest (OVR) approach.

It is noticeable that the OVR method will return a value *x* of scores for a given metric, where *x* is the number of classes in the classification problem. This is not ideal, since the objective of performance metrics is to provide a single value that give us the idea of how well the model is doing as a whole.

With that in mind, we use averaging techniques for metrics in multi-class classification:

- Macro: An arithmetic mean of all class metrics. A good choice for problems where the classes are balanced (similar number of samples in each class).
- Weighted: A weighted arithmetic mean where the number of samples serve as weights to account for imbalances between them.
- Micro: A simple division between correct predictions and all the predictions made by the model. It is similar to accuracy.

The concepts of TP, TN, FP and FN for multi-class classification methods will serve as building blocks for the performance metrics presented in the next subsections.

Accuracy

The accuracy of a model is defined as the percentage of correct predictions in the total number of predictions, that is:

$$acc = \frac{correct\ predictions}{dataset\ size} * 100$$

It is the one of the simplest and most intuitive performance metrics and portrays the overall hit rate of the classification method.

The concept of accuracy does not change for binary or multi-class classification problems.

Precision

Precision focus on showing a number related to the predicted positives output by a model. In mathematical terms, it is translated as:

$$precision = \frac{TP}{TP + FP}$$

By optimizing a model for precision, we are saying that the **false positives** have significance on the result aimed by performing predictions. That is, predicting a negative as a positive will have relevant consequences.

Recall

Recall focus on showing a number related to the actual positives analyzed by a model. Its mathematical form is:

$$recall = \frac{TP}{TP + FN}$$

By optimizing a model for recall, we are saying that the **false negatives** have significance on the result aimed by performing predictions. That is, predicting a positive as a negative will have relevant implications.

F1-Score

F1-Score is a metric that combines both precision and recall scores in order to measure a model. Its mathematical form is:

$$F1 = 2 * (precision * recall) / (precision + recall)$$

Problem Analysis

This section presents information about the dataset used during the development of the problem solution and the planning behind the proposed solution.

Data Exploration

This subsection aims to present the problem through data visualization and exploration in order to better understand the details surrounding the problem and how to better solve it.

The dataset is divided into 5 classes ranging from the number of items presented in each image.

Training Dataset Sizes

One important aspect of classification problems is the balancing of classes. To better understand how the classes are divided, the sizes of the training datasets is presented in the following figure and table.

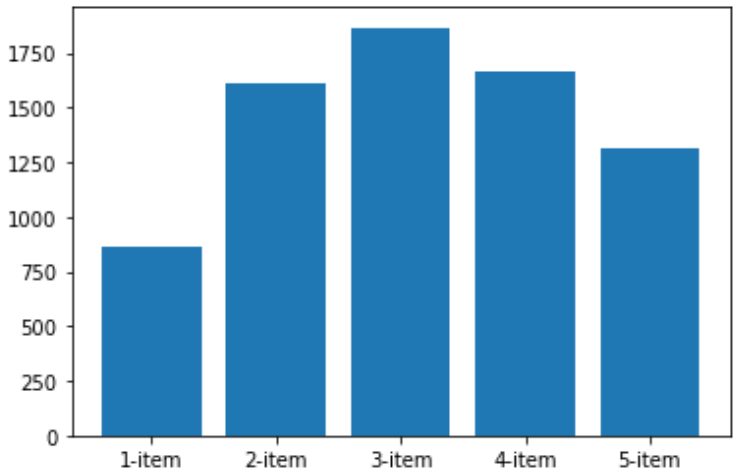

Figure 3 - Training Sample Sizes (Visual Representation)

Table 1 – Training Sample Sizes (Detailed Values)

Class	Training Sample Size
1-item images	859 images
2-item images	1609 images
3-item images	1866 images
4-item images	1661 images
5-item images	1312 images

Except for the class of 1 item per image, the classes are fairly balanced.

Images Details

Since we are dealing with image preprocessing and analysis, it is relevant to know how they are formatted and have an idea of how they look like.

As expected, the images in the training dataset are not formatted to be fed directly to the model. The image sizes ranges are:

Table 2 - Image Size Ranges

	Minimum Value	Maximum Value
Width	295 pixels	1100 pixels
Height	231 pixels	1171 pixels

Examples of training images:

- 1-Item class:

Figure 4 - Example of image with 1 item

- 2-Item class:

Figure 5 - Example of image with 2 items

- 3-Item class:

Figure 6 - Example of image with 3 items

- 4-item class:

Figure 7 - Example of image with 4 items

- 5-Item class:

Figure 8 - Example of image with 5 items

Chosen Performance Metric

Given the nature of the problem, there is not a considerable difference in importance when false positives and false negatives are compared. The main objective of the developed solution must be the correct overall prediction.

Therefore, **accuracy** was chosen as the target performance metric to evaluate the model. The simplest and most intuitive metric is well suited to show how well the model performs given the project's objective.

Solution Implementation

This section aims to show the steps taken in the development of the classification model.

The platform used for these processes was Amazon Web Services (AWS), more specially Amazon Sagemaker and S3.

A diagram of the planned implementation of the solution can be seen in the following figure.

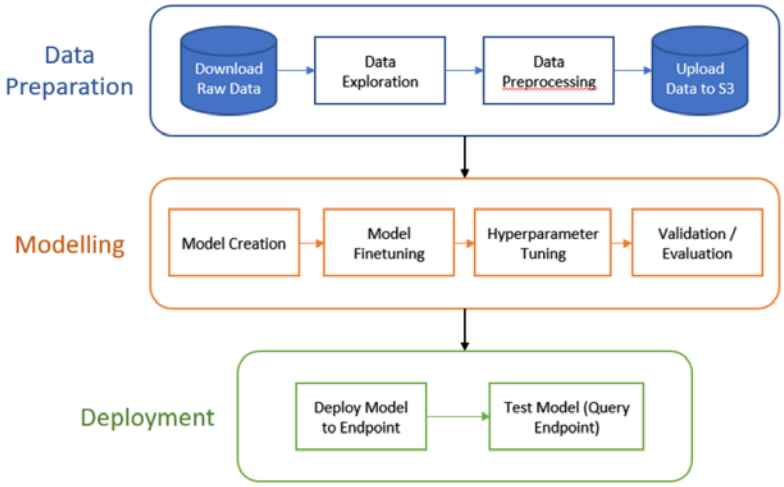

Figure 9 - Proposed Solution Flowchart

Data Preprocessing

The raw dataset (Amazon Bin Image Dataset) provided Amazon holds 500,000 images and metadata from bins transported by robots in an operating Amazon Center.

The images are not divided or labelled in any way. In order to use them a prior division of train images would be necessary using the metadata JSON files by Amazon in the dataset S3 folder. That step is not necessary for this project since Udacity has kindly provided a JSON file that labelled 10,441 images into 5 different folders ranging from 1 item to 5 items in each picture. Therefore, it was decided that the files to be used for training, validation and testing would be the ones listed in the forementioned JSON file.

After downloading the data in the JSON files and performing some exploration, the train, evaluation and test datasets were chosen as:

- Train set: 70% of the complete dataset
- Evaluation set: 15% of the complete dataset
- Test set: 15% of the complete dataset

The first step of the process of data preprocessing was then to split the images

Therefore, the first step of preprocessing was to split the data following the percentages defined previously. That division led to the training sample sizes presented in previous sections of this report.

As observed during data exploration, images in the provided dataset had different sizes, making them not ideal to be directly analyzed by the classification model. Hence, during the creation of the data loaders for training, evaluation and testing phases, the images were resized to the required format for the chosen classification algorithm (to be detailed in the following subsections).

The resizing process reformatted all images to 224 x 224 pixels, which was the classification algorithm required format. Additionally, in an attempt to improve the training process, all images had a 50% chance of horizontal flipping.

Lastly, as the final step of the data loaders creation process, the resulting images were transformed to tensors and normalized afterwards.

Model Implementation

Due to utilization in previous similar projects, the chosen architecture for this project was ResNet50.

The ResNet architecture was a response to a problem that appeared as Convolutional Neural Networks (CNNs) were rising in popularity in the 2010s: as more studies were performed where the number on convolutional layers was being gradually increased, it was observed that the training and testing errors were growing instead of diminishing.

Research has found out that this was caused by vanishing/exploding gradients, whereas the gradients are back propagated to earlier layers, the repeated multiplications performed could turn said gradient either too small (vanishing gradient) or too big (exploding gradient).

The ResNet introduced the concept of the Residual Block, where the technique called skip connections allowed skipping training from a few layers and connecting directly to the output. Therefore, instead of learning from the underlying mapping, the network would benefit from fitting the residual mapping.

The advantage of this approach is to allow skipping by regularization of any layer that would hurt the network performance by generating vanishing/exploding gradients.

The good results obtained with this architecture throughout the community motivated its utilization in this project, as well as my personal experience using it.

Initially, a ResNet50 benchmark model was used in order to provide a comparison baseline for future attempts of model optimization using the test average loss as a comparison tool. The hyperparameters used for this model were:

- Train batch size: 16
- Learning Rate: 0.1
- Epochs: 4

The choices for optimizer and loss functions were Adagrad and Cross Entropy Loss, respectively.

The results obtained using the benchmark model were:

- Accuracy: 30%
- Test Average Loss: 1.4751

Model Refinement

This subsection details attempts at improving both the accuracy and the test average loss for the initial classification model.

Model Finetuning

After taking the first steps creating and training the benchmark model, what naturally follows is an attempt to improve the ResNet50's architecture in order to fit it better to the dataset (model finetuning).

Due to limited budget, just one attempt at model finetuning was performed to showcase the author's knowledge in these types of practices.

Since it is not in our best interest to lose the pretraining done in the convolutional layers, model finetuning will focus solely on the fully connected layers of the architecture.

The hyperparameters used for the benchmark model were not changed from the last step since the main objective was to test how the modifications in the architecture would affect results.

The fully connected network that follows the convolutional layers in ResNet50 were changed to the layer designed shown in the following figure.

```
model.fc = nn.Sequential(
    nn.Linear(num_features, 128),
    nn.ReLU(),
    nn.Linear(128,256),
    nn.ReLU(),
    nn.Linear(256,64),
    nn.ReLU(),
    nn.Linear(64,5)
)
```

Figure 10 - Fully Connected Layers in Model Finetuning

It is important to mention that the network does not end with an activation function (e.g., SoftMax) due to the loss function (Cross Entropy Loss) applying SoftMax internally.

The results for this attempt of model finetuning were:

- Accuracy: 26.3%
- Test Average Loss: 1.5788

It became clear that this type of optimization would not be ideal. As a next step, hyperparameter optimization was attempted.

Hyperparameter Optimization

The second attempt to improve the model was using the benchmark model as baseline and perform hyperparameter optimization by searching the best set of hyperparameter values in a defined search space.

The target hyperparameters used for this process were learning rate, epochs and train batch size. The metric used as comparison to define the best model was the average loss for the evaluation dataset.

Amazon Sagemaker provides a tool for hyperparameter tuning, all that was necessary was to define the model evaluation metric, define the hyperparameter search ranges and set the maximum number of attempts (tries with different hyperparameter values).

In order to maintain the operational costs within the budget of the project, only 6 jobs were performed.

The search spaces for hyperparameter tuning were defined as:

- Learning Rate Range (Continuous Parameter): Between 0.001 and 0.1
- Epoch Range (Integer Parameter): Between 3 and 5
- Train Batch Size (Categorical Parameter): 16, 32 or 64

The best model had the following hyperparameters:

- Learning Rate: 0.001378
- Epochs: 3
- Train Batch Size: 32

The results for this optimization attempt were:

- Accuracy: 38%
- Test Average Loss: 1.3365

Results

This section aims summarize and discuss the results obtained throughout the project execution.

Model Evaluation

The main metrics for model evaluation is the comparison of accuracy and test average loss. These values are summarized in the tables below.

Model	Learning Rate	Epochs	Train Batch Size
Benchmark	0.1	4	16
Finetuning	0.1	4	16
Hyperparameter Opt.	0.001378	3	32

Table 3 - Result Comparison

Model	Accuracy	Test Average Loss
Benchmark	30%	1.4751
Finetuning	26.3%	1.5788
Hyperparameter Opt.	38%	1.3365

Some observations are due in this point of the project:

- As expected, the accuracy increased as the test average loss decreased (this must be confirmed either mathematically or through experimentation with more samples).
- Contrary to initial intuition, a larger number of epochs is not responsible for better results: although the range for this hyperparameter was between 3 and 5, the best model passed through 3 epochs.

Conclusions

Finally, you will construct **conclusions** about your results, and discuss whether your implementation adequately solves the problem.

Lastly, the conclusions and discussions about the project will be presented here.

Taking accuracy as the most important metric in this project, its values throughout the development of the solution were not high enough to be considered a final solution for the initial problem.

The possible attempts to solve this low-accuracy issue are:

- Application of more preprocessing techniques, such as vertical flipping, to increase the variety of images used to train the model.
- Utilization of a different optimizer, such as Adadelta or Adam.
- Additional attempts at hyperparameter optimization with a greater range of search.
- Implementation of a different pretrained architecture.

Although the accuracy values were not ideal for real-world implementation, it was valuable to increase my Machine Learning skills.

Despite the achieved levels of accuracy not being enough to solve the proposed problem adequately, the method used during this project and the possible solutions presented in this section might be enough to improve the model to a level where it becomes a complete solution.

References

[1] *Models and Pre-Trained Weights.* (n.d.). PyTorch. https://pytorch.org/vision/stable/models.html

[2] Bex, T. *Comprehensive Guide to Multiclass Classification Metrics* (Jun 9, 2021). Towards Data Science. https://towardsdatascience.com/comprehensive-guide-on-multiclass-classification-metrics-af94cfb83fbd

[3] Faria, H. *Project: Inventory Monitoring at Distribution Centers.* (n.d.).