Identification of Surface Material of Baggage for Self-service bag drop system

Intermediate Report

24th January 2020

Supervisor  Dr. T.W. Chim
Members  Utsav Raj  3035497915
Muhammad Sheheryar Naveed  3035493672

https://wp.cs.hku.hk/fyp20030/
Abstract

To accommodate high volume of Hong Kong airport activity without compromising the service quality, self-service bag drop exercise has intensified but it should be completely automated by integrating a deep learning material (hard or soft) detection model. Taking inspiration from this initiative, our project aims to provide a robust solution not only dehumanizing the self-bag drop procedure but also helping reduce the wear and tear on soft baggage in-flight and during handling by using a tray. The exploration phase involves two potential approaches to help distinguish different types of surface materials of baggage. The pipelined approach uses Yolo-v3, ResNet50 and Inception-ResNet-v2 to perform the object recognition and classification as cohesive units whereas similar operations could be performed all together using Mask R-CNN in the aggregated approach. With experimentations performed on the pipelined approach, classification produced promising results but the first component in the stack, Yolo-v3 could not be tested due to the limitation of the dataset where each image is a minimum bounding box of the object itself. Mask R-CNN mostly produced visually pleasing results for image segmentation, but performs poorly when new objects (eg. persons or background) are introduced in the image, clearly demonstrating the novelty of the dataset that lacks such variable parameters. There is still work that needs to be done to get the comprehensive view of the discipline; tuning hyper-parameters for Mask R-CNN and new data collection. Hoping Mask R-CNN to be successful in segmenting objects accurately, Yolo-v3 training would no longer be required as the segmented images could be fed to the already highly accurate classifier thus achieving an overall optimal performance. Keeping in view the costs and user convenience, testing the performance of the two approaches is highly dependent on the camera installation setup, therefore would be our immediate next task in terms of priorities.
Acknowledgements

We would like to offer a token of appreciation to our supervisor Dr TW Chim for his thorough guidance throughout the project, which immensely contributed to the completion of this paper. We would also like to express our sincerest gratitude to Cezar Cazan from the Center of Applied English Studies for providing us insightful recommendations which proved extremely useful while drafting this paper. Both of their willingness to give their time so generously has been very much appreciated.
# Table of Contents

1. Introduction 10
   1.1 Research Background 12

2. Related Work 12

3. Methodology 13
   3.1 Data Collection 14
   3.2 Camera Setup: 15
   3.3 Algorithm and Model Design 16
      3.3.1 Pipelined Approach 17
         3.2.1.1 Object Segmentation and Cropping 17
         3.2.1.2 Object Classification Models 18
         3.2.1.2.1 Prevention of Overfitting 18
         3.2.1.2.2 Data Augmentation 19
         3.2.1.2.3 ResNet-50 25
         3.2.1.2.4 Custom Model 26
      3.3.2 Aggregated Approach 28
         3.3.2.1 Architecture 29
         3.3.2.2 COCO weights 33
   3.4 Comparison of two model approaches 34

4. Results 35
   4.1 Pipeline Approach 35
      4.1.1 Network Training and Hyperparameter Selection 35
      4.1.2 Consequences of Varying Conditions (Data Randomness and Outliers) 41
   4.2 Mask R-CNN 42
      4.2.1 Network Training and Hyperparameter Selection 42
      4.2.2 Investigation of Performance 44
   4.3 Difficulties Encountered: 47

5. Future Work 50
   5.1 Project Schedule 53

6. Conclusion 54

7. References 56
## Abbreviations

List of all abbreviations used in this paper in alphabetical order.

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COCO</td>
<td>Common Objects in Context</td>
</tr>
<tr>
<td>FPN</td>
<td>Feature Pyramid Network</td>
</tr>
<tr>
<td>HKAA</td>
<td>Hong Kong Airport Authority</td>
</tr>
<tr>
<td>HKIA</td>
<td>Hong Kong International Airport</td>
</tr>
<tr>
<td>HKU</td>
<td>The University of Hong Kong</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>R-CNN</td>
<td>Region Convolutional Neural Network</td>
</tr>
<tr>
<td>ROI</td>
<td>Regions of Interest</td>
</tr>
<tr>
<td>RPN</td>
<td>Region Proposal Network</td>
</tr>
<tr>
<td>ResNet</td>
<td>Residual Neural Network</td>
</tr>
<tr>
<td>SSD</td>
<td>Single Shot Detector</td>
</tr>
<tr>
<td>VGG</td>
<td>Visual Geometry Group (VGG-19 is a trained CNN by this group from Oxford University)</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>One of the self-service bag drop systems.</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>Example of hard baggage and soft baggage (with and without abrasions)</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>Visualization of three computer vision techniques on a baggage image</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>Block Diagram of proposed Model Approaches</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>Visualization of the secondary dataset.</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>Different camera configurations</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>Different sized images being resized or padded</td>
<td>17</td>
</tr>
<tr>
<td>8</td>
<td>Color filter comparison</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>Visualisation of all data manipulation methods</td>
<td>21</td>
</tr>
<tr>
<td>10</td>
<td>Average cost vs error rate for imbalanced class sample sizes</td>
<td>23</td>
</tr>
<tr>
<td>11</td>
<td>ResNet residual block skip connection implementation</td>
<td>26</td>
</tr>
<tr>
<td>12</td>
<td>Summary of Custom CNN Network</td>
<td>27</td>
</tr>
<tr>
<td>13</td>
<td>Block diagram of Mask R-CNN architecture.</td>
<td>29</td>
</tr>
<tr>
<td>14</td>
<td>Feature Pyramid Network allowing pyramids to see high level and low level features</td>
<td>30</td>
</tr>
<tr>
<td>15</td>
<td>Simplified illustration showing 42 anchor boxes</td>
<td>30</td>
</tr>
<tr>
<td>16</td>
<td>3 anchor boxes (dotted) and the shift/scale applied to them to fit the object precisely (solid).</td>
<td>31</td>
</tr>
<tr>
<td>17</td>
<td>Illustration of stage 3 of Mask R-CNN.</td>
<td>32</td>
</tr>
<tr>
<td>18</td>
<td>Getting a fixed size feature map from ROI.</td>
<td>32</td>
</tr>
<tr>
<td>19</td>
<td>Scaling up the 28x28 soft mask to fit our baggage image</td>
<td>33</td>
</tr>
<tr>
<td>20</td>
<td>Sample images for class suitcase from COCO 2014 dataset</td>
<td>33</td>
</tr>
<tr>
<td>21</td>
<td>Custom Model on complete sampling data</td>
<td>36</td>
</tr>
<tr>
<td>22</td>
<td>Custom Model on under sampling data</td>
<td>37</td>
</tr>
<tr>
<td>23</td>
<td>ResNet-50 results on complete sample – without data randomizer</td>
<td>39</td>
</tr>
<tr>
<td>24</td>
<td>ResNet-50 results on complete sample – with data randomizer</td>
<td>39</td>
</tr>
<tr>
<td>25</td>
<td>ResNet-50 Training Results on Oversampling Data</td>
<td>40</td>
</tr>
</tbody>
</table>
Figure 26. ResNet-50 training on under sampling data with outlier

Figure 27. Trained model working well on segmentation of baggage with focused image of baggage.

Figure 28. Example of a background that mimics baggage material.

Figure 29. Example of an image with a person near the baggage.

Figure 30. Example of two hard baggage overlapping with each other.

Figure 31. Effect of varying lighting on bag’s appearance

Figure 32. Showcasing what images the new dataset may contain

Figure 33. MTR Airport Express containing the two scenarios

Figure 34. Different environments in Hong Kong International Airport
List of Tables

Table 1. Precise summary of datasets 22
Table 2. Comparison of two model approaches - aggregated and pipelined 34
Table 3. Real Dataset based on web images 36
Table 4. ResNet-50 results on experimentation performed on under sampling data 38
Table 5. Validation Loss for different Resnet backbone 42
Table 6. Validation Loss for different layers trained 43
Table 7. Validation Loss for different loss weights 44
Table 8. F1-score and mAP@[0.5] form the most successful Mask R-CNN model 45
1. Introduction

Hong Kong International Airport, with over 80 Best Airport Awards\textsuperscript{[1]}, currently enjoys its position as a premier airport in Asia and arguably in the world. Stiff challenges from other airports in the Middle East have made it difficult for HKIA to maintain its attractive standing. Entertaining 71.5 million passengers in 2019\textsuperscript{[1]}, HKIA has realized the ever-coming problem of handling a plethora of passengers while maintaining its unique position. With alternative transport means such as the inclusion of High-Speed rail, the scope of the problem has become broader than ever.

Provided the ever-increasing importance passengers place in their experience and strong competition to attract various passengers, HKIA plans on using the latest technologies such as machine learning to transform it into a so-called smart airport that leverages such technologies to enhance the customer experience and work operations. One such smart initiative is a self-service baggage drop system.

![Figure 1. One of the self-service bag drop systems.\textsuperscript{2} With over 120 systems, this is one of the largest such projects in the world.](image)

Introduced in 2016, self-service baggage drop (as seen in figure 1) entails the purpose of mollifying the long queues at the check-in counters and increasing the efficiency of the departure process while also reducing the handling cost for HKIA\textsuperscript{[3]}. Passengers who have checked-in earlier either using a self-service kiosk or web-based check-in are just required to follow these instructions to self-drop their bag and thus enjoy a hassle-free experience:
1. Scan their boarding pass barcode and get a luggage tag to place it on their bags\textsuperscript{[4]}

2. Put a soft bag in a tray and hard bag directly onto the belt as per the directive of the airport staff on duty\textsuperscript{[4]}

3. Confirm the details on the screen to send the bags to the main conveyor belt\textsuperscript{[4]}

Incorporation of this system led to baggage check-in time being reduced by a large extent – three minutes to seventy seconds\textsuperscript{[2]}.

The objective of this project is to plan, design, develop and implement a state-of-the-art machine learning model, accompanied by a fixed camera setup to provide live feed to the model, in order to identify a bag as soft (made up of either leather or woven nylon – cordura, ballistic, or ripstop) or hard (made up of aluminium or plastics – ABS or polycarbonate). The screen would then prompt the passenger to use a tray in case the bag is classified as soft. This would allow them to follow the right practice to comply with the standard procedures of HKIA while also avoiding any potential damage to soft bags, in turn improving customer experience. Besides protection against scraping as shown in figure 2, this seamless model would also offer reduced human interaction between the passengers and the airport staff which is much demanded with the Covid-19 in place. The following models will be used, and their performances compared - Mask R-CNN and Yolo-v3 with ResNet50 or Yolo-v3 with Inception-ResNet-v2. This project is not just limited to the baggage detection and material identification but needs to propose an overall working prototype to HKIA and hence, requires the understanding of not only the algorithms but also the hardware components such as camera configuration. This paper provides a basis for sound understanding regarding the well-suited techniques for performing such computer vision tasks and their applicability at a preliminary stage.

<table>
<thead>
<tr>
<th>Hard Baggage</th>
<th>Soft Baggage</th>
<th>Soft Baggage (with abrasions)</th>
</tr>
</thead>
</table>

\textit{Figure 2. Example of hard baggage and soft baggage (with and without abrasions)}
The remaining paper is formulated as follows. This section continues with a subsection on research background of key terms identifying capabilities of each model. Section 2 discusses the existing research done on the baggage material in the literature. Then, Section 3 provides the methodology adopted for treating our dataset which is necessary for the training stage, then the camera setup followed by the two approaches - Pipelined and Aggregated, including the algorithm designs under these as well as their comparison from a practical standpoint. Section 4 explains the experiments performed to evaluate both the classification models in pipelined approach and aggregated mask R-CNN implemented, interprets the results and findings, while also considering the difficulties encountered, and finally concludes by discussing the future work to be completed in Section 5.

1.1 Research Background

Object recognition is a computer vision technique to recognize varying classes of objects in an image/video\textsuperscript{5}. Object detection provides more detailed information by identifying an object in an image and specifying its location using a bounding box\textsuperscript{5} which is often shown in the form of a minimum bounding rectangle as shown in figure 3. Semantic segmentation is similar to object detection but provides a more detailed approximation of an object by classifying every pixel in an image as a class and then drawing a mask (blue colored mask in figure 3) on that object in an image\textsuperscript{5}.

![Figure 3. Visualization of three computer vision techniques on a baggage image](image)

2. Related Work

While a lot of efforts are being directed towards the recognition of material properties of an image in the stream of Computer Vision, there is scant focus on the detection of the surface materials of luggage items. There are two studies that have been carried out that are associated with the work we are undergoing.
The first study was performed by Filament, an applied AI business\cite{6}, to find mishandled bags by creating an automated database\cite{6}. For each mishandled bag, this database stores a code generated based on the visual properties (i.e. bag type, colors, patterns, wheels, etc.) of the bag using a deep learning model. When a passenger provides the characteristics of the missing bag, the mishandled baggage database is used to find the bag matching the given description. RetinaNet, originally trained on COCO\cite{9} dataset, was used as a basis for training this deep learning model achieving an accuracy as high as 90% including the labelling prediction of both hard and soft bag surfaces\cite{6}. Likewise, we will use the pre-trained models, originally trained on the same COCO dataset this study uses except for Yolo-v2 to be used instead of RetinaNet.

The last study focuses on the re-identification of the same bags to be performed as these bags pass different checkpoints on a baggage handling system. The Siamese network was being used to match the same bag identity images taken at different angles and the network was slightly modified to take into effect the hard and soft bag material surface\cite{7}. Our work will complement the same approach by using an existing network and changing the architecture to fit our use case.

3. Methodology

This section outlines the data collection and modification process in order to train the baggage surface material identification model in the two approaches of Pipeline and Aggregated. The block diagram in figure 4 illustrates the high-level flow of the methodology undertaken for this project. The block diagram consists of three major branches i.e. initial data refinement, approach 1 and approach 2. The first data cleansing stage is discussed in Subsection 3.1, then the camera setup design in Subsection 3.2, followed by the two approaches that are explained in Subsection 3.3. This methodology section is finally concluded by the last Subsection 3.3 that compares the two approaches.
3.1. Data Collection

Ideally, we want many baggage images taken at different angles with multiple images per bag identity to achieve the best results. Owing to privacy concerns that HKIA must comply with, and lack of presence of an inbuilt mechanism to capture such images, data of HKIA passengers is not available and we are relying on secondary sourced data that was collected for another study readily available for use. This dataset consists of 4,500 different baggage identities with around 22,000 annotated baggage images taken at different angles with multi view cameras\textsuperscript{[8]}. All the information about the dataset was stored in a JSON file with the file name of each image (primary key – unique identifier for each image), material label (hard, soft, or others) and polygon (x and y coordinates for each point in the mask). With this dataset, we have a firm base to create an altered dataset making it in sync with the environment of HKIA for our deep learning models.
Figure 5. Visualization of the secondary dataset. It has 12,000 hard baggage images and only around 5,000 soft baggage images. Multiple views of the same baggage is present in the dataset as well.

Since, the study from which this dataset was borrowed focused on three different types of luggage items, the raw data consisted of some unwanted labelled images not containing bags. Such images were identified and removed from the dataset as a part of our data-cleansing stage. The number of such images were relatively low yet, and the refined dataset still consisted of around 18,000 images with the baggage identities remaining to be the same as around 4,200 because only non-bag object images were omitted.

In order to make the data mimic the real environment of HKIA, it needs to be augmented by applying various filters on it. Also, the model may have a lower propensity to recognize certain materials that change color at varying lighting. To counteract this problem, color distortion augmentation was done by changing the parameters such as randomizing the brightness and changing the contrast and the saturation. Further augmentation involving flipping and rotating of the image as well as color filter was performed in order to make the model focus on the surface of the baggage and ignore the background. This transformation step aims at not only depicting the true environment but also to multiplicate the training data with higher complexity. More about this data augmentation step is discussed in the section 3.2.1.2.2.

3.2 Camera Setup:

The plan is to test two camera configurations (a single camera and two multi-view cameras) to see which one better fits the model as seen in figure 6. The reason to choose as well as prioritize one camera configuration is that if the performance of a single camera setup suffices the problem, HKAA does not need to buy two cameras for each self-service bag drop
system, reducing cost of the setup along with maintenance. For the two camera configurations, we will install two cameras at different angles to capture a multi-view visualization of the certain bag object. Two images will nullify any faulty image on one camera that could impact the model performance otherwise if a single camera is used. An algorithm will be written to analyze the two images and give a final prediction of the surface material based on external factors such as lighting. Each of the two phases will be properly tested while considering productivity, ease-of-use and economic feasibility.

![Image of camera configurations](image)

**Figure 6. Different camera configurations - Top-down view (Red), Front view (Green)**

### 3.3 Algorithm and Model Design

There are essentially two approaches that were adopted for this project. Either there should be an aggregated image segmentation model to be trained on augmented data to be produced as discussed in Section 3.1 or there could be a split pipelined model consisting of two networks in place, where the first one detects and crops the baggage object image while the second one specializes in identifying the material of the bag in a clean image without any random background.

Pipelined approach, as shown in the block diagram in figure 4 above, consists of numerous stages along with a condition of the availability of the existing models. This section consists of two subsections where Subsection 3.3.1 thoroughly discusses each of the individual components of the pipelined approach followed by Subsection 3.3.2 that describes the aggregated approach.
3.3.1 Pipelined Approach

3.2.1.1 Object Segmentation and Cropping

Outperforming state-of-the-art methods like Faster R-CNN with ResNet and SSD\textsuperscript{[10]}, Yolo-v3 will be used as an object detection network. Given the research work published under IEEE\textsuperscript{[11]} that the pre-trained Yolo-v3 is substantially better in detecting most of the common types of objects including bags, it is going to be prioritized over retraining on a novel dataset. Labelling\textsuperscript{[12]} would be done (approach-1 ‘no’ condition – block diagram plan shown in figure 4) from scratch if Yolo-v3 transfer learning does not produce expected results. However, Retraining the entire model from scratch is not a priority because of the extra resources and time needed to be invested. The detected objects will be cropped and passed to the second network for classification.

![Different sized images being resized or padded. Either of the alterations to image allows it to match the dimensions of the training data.](image)

Due to varying dimensions of images, a standard dimension needs to be set, and the effective padding or cropping must be performed for some images to conform to standard. A limitation of padding is, however, increased features for the neural network to learn and thus might take extra computation and increased training time. Resizing is an alternate solution as shown in figure 5, it can distort the image yet. A trade-off approach will be adapted between the two potential methods and then resize, when the difference between the target size and given size is small (a threshold to be set) or pad otherwise.

Putting Yolo-v3 before the classifier in the pipeline brings us several benefits. Yolo-v3 can help us detect multiple bags in an image too. This makes our approach more
scalable by extending this project further when passengers just put all their bags at one place rather than putting one after the other. The classifier if trained on just the bag object itself without the background as passed by Yolo-v3, would never fail and become independent of any environmental conditions affecting the background thus making our model more robust once high accuracy for individual classification model is achieved.

It is to be noted that due to our data consisting of highly cropped images, Yolo-v3 cannot be trained on such data. Therefore, the Yolo-v3 part is skipped before substantial data with images consisting of sufficient background proportion to create a minimum bounding rectangle around the bag becomes available.

3.2.1.2 Object Classification Models

This is the final step in the pipelined approach that would output the label of the bag present in the original image. Up to this point, a cropped image with background scraped off containing only the bag present in the original image would be available and fed to the model that would be used in this step.

Two different classification networks namely ResNet50 and Inception-ResNet-v2 were used and compared. More detail about these networks is further discussed in the subsections below.

The accuracy of the two models will be tested by determining the F1 scores. F1 score is a good means of comparison of predictive abilities of different models as it counteracts the imbalance between the target classes by giving equal weightage to majority and minority class\textsuperscript{[14]}. The better performing model (higher F1 score) would be used.

3.2.1.2.1 Prevention of Overfitting

The models used in our study have a plethora of parameters making it hard to train such models. These are state of the art models that are usually designed to classify thousands, if not hundreds, of classes and using these to classify into two categories (hard or soft) might be an overkill and eventually could lead to overfitting. On the other hand, taking the advantage of transfer learning, enable these state-of-the-art models to extract the features from the images quite accurately making the overfitting more conceivable. Several steps were taken to ensure that DNNs do not overfit.
3.2.1.2.2 Data Augmentation

Data Augmentation techniques are some of the exemplar techniques to solve the overfitting issue. Not only these are used to increase the training data set but are used widely in the Deep Learning activities. These techniques have been seen to improve the accuracies of the DNN in the range +2 to 31%.[16]

Two different approaches were adapted to implement this technique:

1. **Pre-training Data Augmentation**: In this approach the data is manipulated before the training process and then kept constant for the remainder of the experiment. As the number of augmentation methods increases, this scheme becomes challenging to implement due to increased memory required to store the multiplied data in relation to the storage constraints. However, this approach gives us a fair comparison of results of different networks by keeping sizes of different variants of augmentation fixed.

2. **Post-training Data Augmentation**: This approach does the data augmentation on-the-fly by randomly selecting different batches of images for each distortion variant thus leading to increased combinations of augmented data. Although this method optimizes the storage requirements by achieving a parallelization performed in the CPU, it adds an additional layer of burden for the network to learn the new features every time the data is fed into it for training.

For our use case, mainly the second approach was adopted because availability of resources (large-run time memory and high-power GPUs). This approach helped us to fairly compare the performance of different types of model with keeping such factors constant. The dataset was manipulated with the following methods:

1. **Coloring**: It changes blue to red color filters and vice versa only. As illustrated in figure 8, the technique used is effective in changing the design color as well which is well-suited to a real-life situation where multiple color bags are available for the same bag. Model must be robust enough to work in such situations. Python library OpenCV’s color conversion module ‘BGR to RBG’ (R=Red, G=Green, B=Blue) was used to achieve the color change.
2. Rotation: The rotation range was kept at 50%. While rotation would help when the camera captures the bag completely vertically or horizontally, for images taken at different angles, too much rotation might not depict a real-life scenario. A more precise rotation will be produced once the camera configuration and testing is finalized.

3. Noise: Noise may result from using low quality camera sensors. Similarly, some cameras automatically perform the edge contrast. This is done to make the edges appear more prominent thus making the object easily distinguishable. This added noise was achieved by adding an edge enhancement filter by a python Pillow library.

4. Lighting: Lighting could vary depending on the environment and background. Some cameras might have more contrast as well. This noise can be modeled as linear noise added to each color component of each pixel separately. Pillow library functions were used and the brightness was increased and decreased by 40% to achieve the lighting and darkness effect.

5. Blur: Blur can result when a camera is not focused properly on the object of interest. Additionally, blur can simulate the network’s performance on small or distant objects that will be captured with low resolution. To add a blur effect, ‘Blur’ image filter by python Pillow library was used.

6. Flipping and Zoom: This was done particularly to multiplicate the data without much realistic component being linked to it as the plan is to fix the zoom of the camera to a certain level.
An important observation is the difference in sizes of the dataset for each class. While the number of hard images is ~12,000, soft images are only limited to one-third the size of hard ~5000. If the model is trained on this entire dataset without balancing off the difference of the training set of two classes, a bias of the model towards one of the two classes is imminent. To counteract this issue (minority_class.pdf in research paper folder), three different datasets were prepared in the following manner:

1. Random minority Over Sampling: Manipulating the data using the above-mentioned methods and randomly picking the samples from each of the available augmented sets of soft images to increase the total size of soft images to ~11,000.

2. Random majority Under Sampling: Randomly selecting 4,500 hundred images from a pool of ~12,000 images for hard bags thus making a total dataset equal to 4,500 training images for each of the hard and soft bags.

3. Complete Sampling: Keeping the dataset as it is with an imbalance. This type of dataset is produced with the intention of using a cost-sensitive learning[17] explained in the next subsection.

A complete summary of the number of images for each of the above dataset is described in table 1.
Table 1. Precise summary of datasets

<table>
<thead>
<tr>
<th></th>
<th>Over Sampling</th>
<th>Under Sampling</th>
<th>Complete Sampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>Hard</td>
<td>11,000</td>
<td>4,500</td>
</tr>
<tr>
<td></td>
<td>Soft</td>
<td>11,000</td>
<td>4,500</td>
</tr>
<tr>
<td>Validation</td>
<td>Hard</td>
<td>1,000</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>Soft</td>
<td>1,000</td>
<td>300</td>
</tr>
<tr>
<td>Test</td>
<td>Hard</td>
<td>500</td>
<td>134</td>
</tr>
<tr>
<td></td>
<td>Soft</td>
<td>500</td>
<td>88</td>
</tr>
<tr>
<td>Total</td>
<td>Hard</td>
<td>12,500</td>
<td>4,934</td>
</tr>
<tr>
<td></td>
<td>Soft</td>
<td>12,500</td>
<td>4,888</td>
</tr>
</tbody>
</table>

**Cost Sensitive Learning:**

This method assigns different costs to misclassification of examples from different classes\[18\]. These aim at adjusting the weights in the objective loss function for training samples from different classes. Various methods are applicable to achieve such an approach. Popular methodology is assigning weights according to inverse class frequency by assigning higher weights for wrongly predicted samples. It can also be referred to as a backward pass of backpropagation algorithm. Here we adapt the neural network to mercurial weights in the calculation of loss for individual classes based on the fraction of their sample size in the original training set.

When the number of samples in the classes are not equal, the performance is measured by average cost per example rather than just the error rate. It is possible to see a small error rate but poor overall performance. The average cost, where total testing sample size is N and Cost function is the misclassification cost for a sample I of a particular class, can be represented as follows\[19\]:

\[
Average\ cost = \frac{1}{N} \sum_{i=1}^{N} Cost[actual\ class(i),\ predicted\ class(i)]
\]

Cost\[i, j\] = cost of misclassifying an example from “class i” as “class j”

Cost\[i,i\] = 0 (cost of correct classification).
For uniform case (equal number of samples per class), the Cost function would be:

$$\forall i, j : Cost[i, j] = \begin{cases} 
1, & i \neq j \\
0, & i = j 
\end{cases}$$

When all the cost weights for each class are uniformly distributed (equal number of samples per class), the error rate becomes a special case for the average cost.

$$Error \ rate = \frac{No. \ of \ incorrectly \ classified \ examples}{N}$$

$$Accuracy = 1.0 - Error \ rate$$

If the costs for each class is non-uniform, the average cost would reflect a true reflection of the model and, hence is a better fit for model performance criterion. In short, a DNN by default assumes each class has an equal weightage and this could be erroneous when the sample size varies, as explained in figure 6. It can be noticed that average cost is a better performance criterion over traditional error rate when it comes to imbalanced class sample sizes.

<table>
<thead>
<tr>
<th>Cost[soft, hard] = 0.75 (For 1 hard, 3 soft)</th>
<th># of testing images = 5 (3 hard and 2 soft)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost[hard, soft] = 0.25</td>
<td>Prediction = 2 soft incorrectly predicted</td>
</tr>
<tr>
<td># of train soft bag images = 4000</td>
<td>Average cost = (0+0+0+0.75+0.75)/5 = 30%</td>
</tr>
<tr>
<td># of train hard bag images = 12000</td>
<td>Traditional Error rate = (0+0+0+0.5+0.5)/5 = 20%</td>
</tr>
</tbody>
</table>

*Figure 10 – Average cost vs error rate for imbalanced class sample sizes*

**Calculating Class Weights**

To implement this strategy different class weights would be set that directly impact the loss for each of the two classes differently thus affecting the overall loss. The high cost examples (samples with high expected misclassification cost – soft in our case) would have a greater impact on the overall loss of a batch. In other words, it penalizes the incorrect prediction of soft bags with a larger magnitude as compared to hard bags. Thus, the relative
weight of each class is impacted in the calculation of the objective function. The ratio by
which weight imbalance is calculated below where formula ‘A’ can be simplified as formula
‘B’ without normalization taken into account.

\[ \eta(p) = \frac{\eta \cdot CostVector[class(p)]}{\max_i CostVector[i]} \]  \hspace{1cm} (A)

\[ \text{Ratio} = \frac{\# \text{Samples of soft bags}}{\# \text{Samples of hard bags}} \]  \hspace{1cm} (B)

The fraction of each class sample is computed, and the results are normalized to
ensure that the convergence of the backpropagation model is not affected. Here the cost
vector is the expected cost of misclassifying a sample belonging to i\textsuperscript{th} class and is computed
as follows, where the function P is the estimate of the prior probability of the example
belonging to i\textsuperscript{th} class:

\[ CostVector[i] = \frac{1}{1-P(i)} \sum_{j \neq i} P(j)Cost[i, j] \]

Assuming, \#samples(hard) =12,000, \#samples(soft) = 4,000

\[ \text{Cost}[\text{soft, hard}] = 0.75 \quad (1 \text{ hard} - 3 \text{ soft} = \frac{3}{4} = 0.75 \text{ - normalizing}) \]

\[ CostVector[soft] = \frac{1}{1-0.75} \cdot (0.75) \cdot 0.75 = 0.75 \]

\( P(i) \) here could be visualized as the fraction of i\textsuperscript{th} class out of the total number of
samples. This method helps us fully utilize the available dataset without any changes to the
original data and hence avail the complete original training set.

**Calculating the Loss After Setting Class Weights**

For every training batch, the traditional loss is shown by formula (C). The new loss
after applying weights for the batch is shown in formula (D). The error rate calculation shown
in figure 9 is slightly different from formula (D).

\[ \text{Loss} = \frac{\sum L(y)}{\text{Batch Size}} \]  \hspace{1cm} (C)

\[ \text{(D)} \]
\[
\text{Loss} = \frac{\sum W_y L(y)}{\sum W_y}
\]

*Where using data in figure 6, \( W_{yi} = 0.75 \) for \( y_i = \text{soft} \)*

\( W_{yi} = 0.25 \) for \( y_i = \text{hard} \)

In formula (D), the sum of weights is in the denominator not the batch size. It makes more sense to divide by total sum of weights than by batch size because the sum of weights would be normalized and comparable to what is deduced in the numerator. The idea behind this whole strategy of cost effective weighing by giving a higher weight ‘x’ to soft bag images is to make the model visualize that whenever it mis-classifies one soft bag, it has misclassified ‘x’ number of soft bags instead. Due to higher magnitude of weight of soft in the loss calculation, the gradients during the gradient descent in the back propagation are also going to be higher.

### 3.2.1.2.3 ResNet-50

ResNet-50 is a 50 layered state of the art classification network. Another network traditionally used is called VGG19 however, it has a problem of diminishing gradient making it fail to learn when the implemented network is deep. Having said that VGG-9 we still plan on using VGG19 in one of the tests to see the performance of the shallow network for our two-class computer vision problem.

ResNet50 here is an enhanced version of VGG19 which solves the problem of diminishing gradient\(^{[13]}\) by skipping the connection and passing the residual to the next layer, thus is the reason we prioritize it over VGG19. With the residual technique the implementation of deep networks became plausible eliminating the otherwise exploding/diminishing gradient problem. An explanation on how the residual is added to the next layer is illustrated in the figure 11 below.
3.2.1.2.4 Custom Model

Apart from the existing state-of-the-art models, we created a custom-built model to address the classification problem. Given that we are only supposed to classify two categories and as mentioned earlier in ‘3.2.2.2.1 Prevention of Overfitting’, about using the pretrained models typically used for very large number of classes as opposed to the two class problem might be needless, it was imperative to create such a model to see if there’s any possibility of training our model even better for this specific problem. Additionally, this gave us an additional testing model to counter the overfitting issues the transfer learning brought as discussed in the results section later on.

Figure 11 – ResNet residual block skip connection implementation
Figure 12. Summary of Custom CNN Network

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv2d_8 (Conv2D)</td>
<td>(None, 111, 111, 16)</td>
<td>448</td>
</tr>
<tr>
<td>batch_normalization_8</td>
<td>(None, 111, 111, 16)</td>
<td>64</td>
</tr>
<tr>
<td>conv2d_9 (Conv2D)</td>
<td>(None, 55, 55, 32)</td>
<td>4640</td>
</tr>
<tr>
<td>batch_normalization_9</td>
<td>(None, 55, 55, 32)</td>
<td>128</td>
</tr>
<tr>
<td>conv2d_10 (Conv2D)</td>
<td>(None, 27, 27, 64)</td>
<td>18496</td>
</tr>
<tr>
<td>batch_normalization_10</td>
<td>(None, 27, 27, 64)</td>
<td>256</td>
</tr>
<tr>
<td>conv2d_11 (Conv2D)</td>
<td>(None, 13, 13, 64)</td>
<td>36028</td>
</tr>
<tr>
<td>batch_normalization_11</td>
<td>(None, 13, 13, 64)</td>
<td>256</td>
</tr>
<tr>
<td>conv2d_12 (Conv2D)</td>
<td>(None, 6, 6, 64)</td>
<td>36028</td>
</tr>
<tr>
<td>batch_normalization_12</td>
<td>(None, 6, 6, 64)</td>
<td>256</td>
</tr>
<tr>
<td>conv2d_13 (Conv2D)</td>
<td>(None, 4, 4, 128)</td>
<td>73856</td>
</tr>
<tr>
<td>batch_normalization_13</td>
<td>(None, 4, 4, 128)</td>
<td>512</td>
</tr>
<tr>
<td>conv2d_14 (Conv2D)</td>
<td>(None, 2, 2, 128)</td>
<td>147584</td>
</tr>
<tr>
<td>batch_normalization_14</td>
<td>(None, 2, 2, 128)</td>
<td>512</td>
</tr>
<tr>
<td>flatten_1 (Flatten)</td>
<td>(None, 512)</td>
<td>0</td>
</tr>
<tr>
<td>dense_4 (Dense)</td>
<td>(None, 1024)</td>
<td>525312</td>
</tr>
<tr>
<td>dense_5 (Dense)</td>
<td>(None, 1024)</td>
<td>1049600</td>
</tr>
<tr>
<td>dense_6 (Dense)</td>
<td>(None, 512)</td>
<td>524800</td>
</tr>
<tr>
<td>dense_7 (Dense)</td>
<td>(None, 2)</td>
<td>1026</td>
</tr>
</tbody>
</table>

Total params: 2,421,682
Trainable params: 2,420,619
Non-trainable params: 992
The architecture of the model we created is shown in figure 12. There are a total of seven convolutional layers with ‘relu’ activation and each followed by a batch normalization layer while putting four fully connected layers at the top.

Many CNNs even include pooling layers in their architecture, which artificially reduce the resolution further after certain processing steps. This is usually considered a good idea as long as losing positional information is acceptable. But using a pooling layer would increase the number of computations for the previous convolutional layer. According to a study\cite{20}, when pooling layers are replaced with a convolutional layer of a stride 2, performance is seen to be stabilized without loss in accuracy on several image recognition benchmarks. Strides would not only eliminate the need to add an additional pooling layer to down sample the image but also reduce computation in the convolutional layers in parallel. It can be observed that instead of adding max pooling layers, strides of (2,2) were kept.

Initially, only five convolutional layers were added along with dropouts after each of the fully connected layers, but the results indicated that it was quite hard to train the model where the train accuracy was stuck at 85% even after several epochs. To enhance the network learning further, an additional two convolutional layers were added which gave promising results after long periods of training. Dropouts were also removed because there was no sign of overfitting seen. These results are discussed in detail in section 4.

### 3.3.2 Aggregated Approach

This approach relies on using the dataset produced in Section 3.2.1.2.2 with a semantic segmentation model called Mask R-CNN to perform both object detection and identification. It creates a high quality segmented mask around the bag object and at the same time also is able to predict the material class that the baggage belongs to. The results (bounding box coordinates and label) of the Mask R-CNN are created on the images using OpenCV.
3.3.2.1 Architecture

Figure 13. Block diagram of Mask R-CNN architecture.\textsuperscript{[21]} It has 4 stages - Backbone, Region Proposal Network, ROI Classifier & Bounding Box Regressor and Segmentation Masks.

The block diagram above represents the Mask R-CNN architecture. Mask R-CNN is a two stage framework wherein image is scanned to generate proposals (areas where object is most likely present) and the proposals are classified while generating the bounding boxes and masks.

Stage 1: Backbone

Mask R-CNN’s backbone contains a standard convolutional neural network (for our model, ResNet50 or ResNet101) seen in the start of figure 18 that serves as a feature extractor, and Feature Pyramid Network. The early layers of Resnet detect low level features (edges and corners), and later layers successively detect higher level features (e.g. car, dog).

Passing through the backbone network, the image is converted from 1024x1024px x 3 (RGB) to a feature map of shape 32x32x2048. This feature map becomes the input for the next stages.\textsuperscript{[22]}

The Feature Pyramid Network (FPN) is used as it can better represent objects at multiple scales. FPN improves the standard feature extraction pyramid by adding a second pyramid. This pyramid takes the high level features from the first pyramid and passes them down to lower layers, allowing features at every level to have access to both, lower and higher level features.\textsuperscript{[23]}
Stage 2: Region Proposal Network (RPN)

The boxes in the figure below are the anchors. Which are boxes distributed over the image area, as shown on the left. This is a simplified view, though. In practice, there are about 200K anchors of different sizes and aspect ratios, and they overlap to cover as much of the image as possible.\textsuperscript{[22]}

The Region Proposal Network (RPN) is a light-weight neural network that can fastly scans the image to determine areas (called anchors) containing the objects in a sliding-window fashion. Further, the RPN does not scan over the image directly (even though the anchors are drawn on the image in figure 15 for illustration) but it scans over the...
backbone feature map. This is because scanning over the backbone feature map allows the RPN to reuse the extracted features efficiently and avoid duplicate calculations. \[22\]

![Image](image.png)

**Figure 16.** 3 anchor boxes (dotted) and the shift/scale applied to them to fit the object precisely (blue - solid). Several anchors can map to the same object.

The RPN generates two outputs for each anchor:

- **Anchor Class:** foreground (positive anchor/FG) or background (negative anchor/BG). The foreground class implies that there is probably an object in that box.
- **Bounding Box Refinement:** A foreground anchor (positive anchor) might not be centered perfectly over the object. So the RPN estimates a delta to refine the anchor box so it fits the object more closely. \[22\]

Using the RPN predictions, the top anchors that are likely to contain objects are picked and their location and size refined (as seen in figure 16). If many anchors overlap too much, the one with the highest foreground score is kept and others are discarded (Non-max Suppression). \[21\] These are called the final proposals (regions of interest) and are passed onto the next stage.

**Stage 3: ROI Classifier and Bounding Box Regressor**

This stage runs on the regions of interest (ROIs) proposed by the RPN. It also generates two outputs for each ROI:
Class: The class of the object in the ROI. Unlike the RPN, which has two classes (FG/BG), the deeper network has the capacity to classify regions to specific classes (soft, hard etc.). It can also generate a background class, and any ROI containing that is discarded.[22]

Bounding Box Refinement: Similar to the RPN, further refinement of the location and size of the bounding box to encapsulate the object is done.[22]

Classifiers require a fixed input size. But, the bounding box refinement step in the RPN, leads to ROI boxes having different sizes.

ROI Pooling is cropping a part of a feature map and resizing it to a fixed size (seen in figure 18). It’s similar in principle to cropping part of an image and then resizing it (but there are differences in implementation details).[22]
Stage 4: Segmentation Masks

The mask branch is a convolutional network that takes the positive regions selected by the ROI classifier and generates masks for them. The generated masks are low resolution (e.g. 28x28 pixels) and are soft to keep branch mask light. These soft masks are represented by float numbers, to hold more details than binary masks. During training, the ground-truth masks are scaled down to 28x28 to compute the loss, and during inferencing, the predicted masks are scaled up to the size of the ROI bounding box to give the final masks, one per object.

Mask R-CNN uses a complex loss function which is calculated as the weighted sum of different losses at each and every state of the model. Loss for Mask R-CNN is made up of sum of five components:

- **Rpn class loss**: how well the RPN separates background with objects.
- **Rpn bbox loss**: how well the RPN localizes each object.
- **Mrcnn class loss**: how well the classification of the localized objects proposed by Region Proposal.
- **Mrcnn bbox loss**: how well the localization of the each identified class of the objects
- **Mrcnn mask loss**: how well the segmentation of classified objects using mask is done.

### 3.3.2.2 COCO weights

The key to training the model both faster and better, especially with limited data, is transfer learning. The COCO 2014 dataset is a huge corpus of images, containing 2,507 images of suitcase as seen in Figure 20.

![Figure 20. Sample images for class suitcase from COCO 2014 dataset]
Therefore, initialize the weights of our Resnet101 backbone model to weights pre-trained on COCO. This will improve the accuracy of the feature maps we obtain, and therefore the overall model. By using later layers of Mask R-CNN, We can allow ourselves to freeze and never fine-tune the first layers because we can reuse the weights the model learned to extract features from natural images. This is because the COCO dataset would allow the model to already have extracted global features and training the model further on our dataset would allow it to finetune its weight.

### 3.4 Comparison of two model approaches

This section compares the strength and weakness of both approaches mentioned above in four criteria. While reliability and decoupling of the approaches focuses on the long term maintainability, resources and complexity focuses on price to performance ratio. A more detailed analysis would be conducted once all approaches are implemented and optimised.

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Pipelined</th>
<th>Aggregated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resources</td>
<td>More resources (training time &amp; computation power &amp; effort) are required to train two networks individually.</td>
<td>Fewer resources are required as both recognition and classification are done in parallel.</td>
</tr>
<tr>
<td>Complexity</td>
<td>Two models work independently but failure of object detection model implies definite failure of classification model</td>
<td>Less complex as only one component is involved.</td>
</tr>
<tr>
<td>Reliability</td>
<td>Less reliability as the individual simple classification model might not learn the size of the objects as it is doing just the classification and not the recognition.</td>
<td>More reliability because it does recognition at the same time as classification, so it does learn and consider the factors i.e. detect the sizes of objects when doing the classification.</td>
</tr>
<tr>
<td>Decoupling</td>
<td>As there are two components, one can be easily replaced to accommodate a new dataset</td>
<td>As object detection and identification of surfaces is done as a single component, any new dataset introduced may lead to training the model from scratch.</td>
</tr>
</tbody>
</table>

**Table 2. Comparison of two model approaches - aggregated and pipelined**

To summarize, our methodology consists of a data preprocessing stage where the data is cleansed, re-formatted as per the input requirements of the model and multiplied with filters to counterbalance any external factors making the overall model trained on this data
more sustainable. The processed data is then used in two different approaches. The pipelined approach consists of several layers performing tasks such as bag detection and bag labelling independently using Yolo-v3, and ResNet50, Inception-ResNet-v2 or Custom-build model respectively whereas aggregated approach performs both these tasks all together using a single state-of-the art mask R-CNN model. A practical analysis would then be performed on both the approaches, once these are finalized and optimized, to compare these apart from their accuracies in predicting the baggage material.

4. Results

This section discusses the experimentations performed on the networks described in the methodology Section 3 and the results achieved. Subsection 4.1 details the pipelined approach followed by Section 4.2 that describes the Mask R-CNN forming a part of the aggregated approach.

4.1 Pipeline Approach

For implementation and testing of networks in the pipelined approach, Tensorflow GPU 1.15.2 Keras library with Python 3.7 on a Linux based computer system Ubuntu 18.04.5 with 28 GB RAM, Xeon (R) Silver 4108 CPU @ 1.80GHz , and GeForce GTX 1080 Ti GPU was used. All models used had the same f1 score as accuracy due to similar samples per class so accuracy is used in the context for the sake of simplicity.

4.1.1 Network Training and Hyperparameter Selection

For all the models, different optimizers were tried, and SGD loss appeared to be the most suitable for the Custom model and ResNet-50. Adam was next in line in terms of performance. RMSprop was also tried because of its unique ability of dynamic learning rate but the SGD still took a lead by producing better results. In order to make the learning rate changeable according to the validation loss’s trend, a callback was added to reduce the learning rate if the validation loss increases consecutively in three epochs.

Custom Model:

The custom model was trained on a complete sampling dataset by setting class weights. Initially the number of epochs was set to 100 as an upper limit but as shown in
figure 10, the train loss was still declining thereafter. The model was trained on an additional 200 epochs to see how much maximum accuracy could be achieved. The training accuracy went as high as 94% and validation reached 93%. Surprisingly the declining training loss trend still shows the sign of additional training to obtain improved results. Before spending more resources on it, the weights of the epoch with the minimum loss were loaded and the model was tested. While the high validation accuracy was a good indicator for us to evaluate the model’s performance already, the model was tested on some images scraped from the internet – ‘Real Dataset’ shown in table 3. The accuracy was just limited to 55%. Not to mention that this dataset only consisted of 64 images and most of the images consisted of multiple objects rather than just the bag as it was in the training set.

<table>
<thead>
<tr>
<th>Internet Based Images</th>
<th>Hard</th>
<th>Soft</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>34</td>
<td>30</td>
<td>64</td>
</tr>
</tbody>
</table>

Table 3. Real Dataset based on web images

![Accuracy vs Epoch](image1.png) ![Loss vs Epoch](image2.png)

Figure 21. Custom Model on complete sampling data

Owing to the long training time needed for the complete sampling data, the under sampling data was used to see how far the training can go. A similar behavior was observed where the model was inclined to learn even more if additional epochs were added. Apparently, as shown in figure 11, the training loss (<0.15) even went lower than that in the complete sampling data. Also, the rate of decline of loss is noticeably steeper than complete sampling’s.
Figure 22. Custom Model on under sampling data

The results of these two experiments imply that the custom-built model performs quite well, and it is possible to achieve a quite high accuracy of >94% with more training. Also, it is noticed that the model would only work well when tested on images that are similar to the ones used in training.

**ResNet-50:**

Overfitting was the most prevalent issue that limited the performance of ResNet-50. Various experimentation was performed, and steps taken to resolve this issue.

Initially the experiments were performed on the under sampling data because of its smaller sized nature helping us achieve results quicker and enabling us to make relevant changes that are best suited for our dataset. Five experiments were performed, and their description and the results are listed in chronological order in the table 4 below.
These experiments were performed in conjunction with what was realized from the results of previous experiments and then making that relevant change with the hope to produce better results. Overfitting is clearly observed despite the addition of noise.

Dropout is a regularization technique to drop some subsets of nodes in the layers. Doing so can help us prevent the model from overfitting. It can also be viewed as a method to prevent the model from adapting to the given dataset too much so that it is better at generalizing. Gaussian dropout here refers to the Gaussian(normal) distribution being used. In most DNN dropouts, Bernoulli’s Gate is used. Contrarily, Gaussian dropout doesn’t drop the nodes but rather drops or alters their weights during the training time by adding Gaussian noise. All the nodes are kept intact. Unlike the normal dropout, these nodes are exposed during the testing time and thus leads to lesser computation cost as compared to normal dropout where the weights need to be scaled for testing due to dropped nodes in layers. In short, both the methodologies do the same thing but in a different manner.

Table 4. ResNet-50 results on experimentation performed on under sampling data

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Architecture</th>
<th>Overfitting mitigation mechanism</th>
<th>Augmented Data Used</th>
<th>(Lowest) Train Loss, F1-Score</th>
<th>(Lowest) Validation Loss, F1-Score</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment 1</td>
<td>Last 12 layers unfrozen + 3 Dense layers</td>
<td>Dropout</td>
<td>No</td>
<td>0.0088, 99%</td>
<td>0.2560, 93%</td>
<td>Overfits</td>
</tr>
<tr>
<td>Experiment 2</td>
<td>Last 12 layers unfrozen + 3 Dense layers</td>
<td>Gaussian</td>
<td>No</td>
<td>0.0080, 99%</td>
<td>0.3086, 94%</td>
<td>Overfits</td>
</tr>
<tr>
<td>Experiment 3</td>
<td>Last 12 layers unfrozen + 3 Dense layers</td>
<td>Gaussian – increased magnitude</td>
<td>Yes</td>
<td>0.0080, 96%</td>
<td>0.7892, 51%</td>
<td>Overfits</td>
</tr>
<tr>
<td>Experiment 4</td>
<td>Last 9 layers unfrozen + 3 Dense layers</td>
<td>Dropout – increased magnitude</td>
<td>Yes</td>
<td>0.1815, 93%</td>
<td>0.7069, 49%</td>
<td>Overfits</td>
</tr>
<tr>
<td>Experiment 5</td>
<td>All layers frozen + 3 fully connected layers</td>
<td>No Dropout / Gaussian</td>
<td>No</td>
<td>0.044, 98%</td>
<td>0.43, 89%</td>
<td>Overfits</td>
</tr>
</tbody>
</table>

38
Two experiments were performed on the complete sampling data with the addition of class weights. The first experiment used dropout with the original dataset as training whereas data randomizer was added to the second experiment along with Gaussian noisy layers instead of dropouts. Data randomizer is basically the first data augmentation approach listed in 3.2.2.2.2 Data Augmentation. A tensorflow keras library module was used that added flip, rotation and zoom-in to multiplicate the dataset on-the-fly during the training. The results of these two experiments are shown in figure 23 and figure 24 respectively.

![Figure 23. ResNet-50 results on complete sample – without data randomizer](image1)

![Figure 24. ResNet-50 results on complete sample – with data randomizer](image2)

For the first experiment the validation accuracy was 92% and loss 0.19 on the very first epoch and these metrics got worse thereafter. On the other hand, the model accuracy increased to 93% until the seventh epoch. Thus, validation loss increased, and training loss decreased with this divergence indicating an overfitted model. As a result, randomness was added to the data and dropouts were replaced by Gaussian Noisy layers. The overfitting issue however was not fixed with the randomness in the data in that the validation accuracy got even worse reaching 77% in the very first epoch but training reaching 95% in the seventh
epoch. Not much difference was seen and adding Gaussian noisy layers did not help much on overfitting issues.

Owing from the results of experiments performed on under sampling data where the model was overfitting, unfreezing the layers would mean to fine tune the model even better for learning aiding the overfitting behavior, all the layers of ResNet-50 were kept frozen. Even with this measure in effect, for the oversampling data, the model was overfitting with validation loss only decreasing till the third epoch (87% accuracy) and training reaching as high as 97% by the end of seventh epoch. The results of these experiments can be viewed in figure 25 below.

![Graphs showing accuracy and loss vs epoch](image)

**Figure 25. ResNet-50 Training Results on Oversampling Data**

ResNet-50 was evaluated on the web-based real images shown in table 3. The accuracy of ResNet for predicting such types of images was higher (~92%) than the custom model. This is because of the transfer-learning component of ResNet-50 being utilized as the image-net weights were loaded prior to training. ResNet is better at predicting the global images because of pre-training on image-net images.

With the above results, it is realized that ResNet-50 is quite good at predicting the images it has never seen but the accuracy is just limited to ~94%. Provided that this model needs to put in production, a practical figure of accuracy of at least 97% is required. On the other hand, a custom model has the potential of predicting even better than ResNet-50 without any overfitting issues but is only confined at making good predictions for the type of data it is trained on. With this implication, ResNet-50 seems to be clearly an overkill for our purpose and a shallow state-of-the-art network just like a custom model such as VGG-16 or a shallow version of ResNet might be a better fit and is worth trying.
4.1.2 Consequences of Varying Conditions (Data Randomness and Outliers)

In studying the deep learning literature, an effective technique[^24] used for reducing the overfitting behavior apart from adding the randomness to the dataset is to add an outlier to a class. Adding this foreign agent and training on three classes instead of just the bags might help reduce the overfitting problem. Owing to this approach, an additional 2,500 images of boxes or other types of luggage items were added to the training set. This produced a dataset consisting of 7,500 images, each class having 2,500 images in the train set. The prediction was now made in 3 classes.

ResNet50 was trained with the last 12 layers unfrozen and three final fully connected layers with dropouts. Figure 26 shows the results of this experiment where it can be observed that the overfitting issue was only slightly relieved (more epochs now being trained) but still the training was stopped after validation loss started increasing and the maximum validation accuracy achieved was 89%.

![Accuracy vs Epoch](image)

**Figure 26. ResNet-50 training on under sampling data with outlier**

Since, the testing dataset would not consist of the images from the alien category, if the model classified an image as ‘other’ the class having the next higher probability would be chosen as the correct prediction by the model. It is to be noted in figure 15 that validation accuracy is not the correct indicator of the model performance as it takes into account the prediction of the outlier class which would not be the case on the testing data. Running this trained model on the testing data is yet to be performed but with a lesser expectation of improved performance as the model still overfits.
4.2 Mask R-CNN

4.2.1 Network Training and Hyperparameter Selection

All the experiments for Mask R-CNN are tested on the loss calculated on the validation set. This set is created from choosing 2,000 random images of 1,000 hard and soft baggage each. Training set was the oversampled dataset. SGD optimiser was used as it gave the most generalised model.

Mask R-CNN has the following hyperparameter to be tuned:

**Backbone:**

As mentioned before, Backbone of Mask R-CNN consists of Resnet-50/Resnet-101 and FPN. Hence, the convolutional neural network can change to see how it impacts the performance on our dataset, keeping the other hyper-parameter and weights constant.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Backbone</th>
<th>Epochs</th>
<th>Layers</th>
<th>Loss Weights</th>
<th>Validation Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Resnet-50</td>
<td>100</td>
<td>Heads</td>
<td>1.0</td>
<td>0.8708</td>
</tr>
<tr>
<td>2</td>
<td>Resnet-101</td>
<td></td>
<td></td>
<td></td>
<td>0.3776</td>
</tr>
</tbody>
</table>

*Table 5. Validation Loss for different Resnet backbone*

As seen in the table, Resnet-101 has half the loss (0.3776 vs 0.8708) compared to Resnet-50. This better performance can be attributed to the fact that Resnet extracts class and bounding box from regions of interest, which are similar to focusing on the cropped image of the object. The deeper layers of Resnet-101 allows extract deeper features of hard and soft baggages, allowing for more less loss.

**Training Layer:**

Training of Mask R-CNN can be done on different layers of Mask R-CNN. Heads refers to the layers after ROI Pooling and will learn only the low level features from the dataset. This will allow Mask R-CNN to use global features from the COCO dataset while learning the low level features from our dataset. Similarly, Resnet-50’s Stage 3 and Stage 4 are more deeper layers of Resnet but still allow us to keep the global features from COCO.
Experiment 4 hence focuses on first increasing these global weights first before training deeper using Stage 3, 4 of Resnet and All of the layers. The model is not trained from All layers from the start as our images are highly cropped and just focuses on Baggage and hence, acts more like ROI then a proper dataset.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Backbone</th>
<th>Epochs</th>
<th>Layers</th>
<th>Loss Weights</th>
<th>Validation Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment 1</td>
<td>Resnet-101</td>
<td>100</td>
<td>Heads</td>
<td>1.0</td>
<td>0.3766</td>
</tr>
<tr>
<td>Experiment 2</td>
<td>Resnet Stage 4+</td>
<td></td>
<td>Resnet Stage 4+</td>
<td>0.1744</td>
<td></td>
</tr>
<tr>
<td>Experiment 3</td>
<td>Resnet Stage 3+</td>
<td></td>
<td></td>
<td>0.1708</td>
<td></td>
</tr>
<tr>
<td>Experiment 4</td>
<td>40, 80, 40</td>
<td>Heads, Stage 4+, All</td>
<td>0.1162</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 6. Validation Loss for different layers trained. Important to note that from experiment 1 to 3, the loss had plateaued and further epoch training would result in no lower validation loss.*

As seen from table above, Experiment 4 of using Heads, Stage 4+ and All layers perform much better with validation loss of only 0.1162. This is due to the fact that the model first learns the important features of baggage using the COCO pretrained weights using heads and moves understand much more higher level features by moving deeper into the network.

**Loss:**

Mask R-CNN uses a complex loss function which is calculated as the weighted sum of different losses at each and every state of the model. The loss weight hyper parameters correspond to the weight that the model should assign to each of its stages.[21]

- **RPN class loss:** This corresponds to the loss that is assigned to improper classification of anchor boxes (presence or absence of any object) by RPN. This should be increased when multiple objects are not being detected by the model in the final output.
- **RPN bbox loss:** This corresponds to the localization accuracy of the RPN. This is the weight to tune if the object is being detected but the bounding box needs to be corrected
• **MRCNN class loss**: This corresponds to the loss that is assigned to improper classification of objects that are present in the region proposal. This is to be increased if the object is being detected from the image, but misclassified.

• **MRCNN bbox loss**: This is the loss, assigned on the localization of the bounding box of the identified class. It is to be increased if correct classification of the object is done, but localization is not precise.

• **MRCNN mask loss**: This corresponds to masks created on the identified objects. If identification at pixel level is of importance, this weight is to be increased.

Based on the information provided above, we are going to modify the following loss weights (referred to as custom weights as now on):

• **RPN class loss**: Increased to 1.2 as fewer objects are detected based on our dataset.

• **MRCNN class loss**: Increased to 1.5 as objects are detected but misclassified.

• **MRCNN mask loss**: Increased to 1.5 so masks are clearer.

• **Others**: Reduced to 0.7.

<table>
<thead>
<tr>
<th></th>
<th>Backbone</th>
<th>Epochs</th>
<th>Layers</th>
<th>Loss Weights</th>
<th>Validation Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Experiment 1</strong></td>
<td>Resnet-101</td>
<td>40, 80, 40</td>
<td>Heads, Stage 4+, All</td>
<td>1.0</td>
<td>0.1342</td>
</tr>
<tr>
<td><strong>Experiment 2</strong></td>
<td></td>
<td></td>
<td></td>
<td>Custom weights</td>
<td>0.1162</td>
</tr>
</tbody>
</table>

*Table 7. Validation Loss for different loss weights.*

As mentioned above, using the proposed custom weights help reduce validation loss by around 13%. Hence, These custom weights allow for our model to fit better to our current dataset.

**4.2.2 Investigation of Performance**

Based on the results of our previous experiment, we calculate the F1-Score as well as mean average precision of IOU threshold 0.5. Precision is the ratio of true positive to all positives - how many correct predicted classes (e.g. soft or hard) are compared to all the actual classes. The mean average precision (mAP) for object detection is the average of the AP calculated for all the classes. IOU for threshold 0.5 means that if predicted mask and actual mask overlap by 50% or more, it is considered a true positive.
Based on table, F1-Score for our given dataset is only 0.314 and 0.017 for mAP@[0.5]. These very low scores and investigation needs to be done using the test set (made of random copy-right free images from net) to understand the difficulties the model is facing. Higher F1-score compared to mAP means the model is better at detecting baggage than classifying the baggage the model has detected although both values are extremely low.

By going through the test set, we can observe the following about our trained model:

- Good at segmenting baggage in a cropped/focused image as seen in figure below.

[Figure 27. Trained model working well on segmentation of baggage with focused image of baggage. This is similar to the dataset as seen in figure 5.]

This is because these images closely match our dataset and hence, our model performs well.

- Struggles with a background that mimics baggage material (see figure 28), when people are present (see figure 29) and when two or more baggage are present near together (see figure 30).
As seen in the figure 28, COCO weights are easily able to segment the person as well as the suitcase however, our model (with trained heads or trained from Resnet Stage 3 onwards) not only misses the baggage altogether but focuses on the bigger background instead. This background is similar to the design pattern of the vertical stripes on the hard baggage and hence, confuses our model. As most of the ROI is covered by the metal background, the classification is incorrectly made as hard.

Similarly for figure 29, COCO weights are easily able to segment the person as well as the suitcase however, our model (with trained heads or trained from Resnet Stage 3 onwards) focuses on the baggage as well as the person in the image. As the person is wearing clothes which are similar to the design of a soft baggage and most of the ROI is covered by it, the classification is incorrectly made as soft.
Again, for figure 30, COCO weights are able to segment the two different hard baggages (although, not perfectly). However, our trained model segments both baggage as one hard baggage. It is important to note that reducing RPN class loss weight still gave the result seen above.

All of the problems can be attributed to the design of our current dataset. As seen in figure 5, our current dataset is highly zoomed in images of a single baggage. This means even when training heads of the model using the COCO weights, our model is focused on the large baggage in the middle and material in the large baggage. It is also important to note that our dataset is not much similar to the COCO dataset of suitcase and therefore, the full advantage of transfer learning is not being taken. Hence, for situations as seen in figure 26 and figure 27, our model is focused on the whole image more than the actual baggage itself. A new dataset with clear baggage and different environments can help alleviate these problems.

4.3 Difficulties Encountered:

Several difficulties were encountered when using the dataset and during the experimentation and implementation of the algorithms for the aggregated approach outlined in this report. In particular, the metadata of our dataset was in JSON format contrary to VIA format that our pre-trained Mask R-CNN model needed. This transformation from JSON to VIA format was a time consuming task. Conforming to the correct type of input required and reformatting was necessary to get the advantage of the additional characteristics of this
chosen Mask R-CNN code including but not limited to being able to perform automated padding for varying sizes of images, and calculating multiple losses (bounding boxes, mask, labels etc.).

Some of the difficulties were linked to the glitches present in the non-augmented dataset in 3.1 used for training. There was a corrupted image which could not be used in training, however, for most of the time there were alternate images available for the same identity and hence the removal of such images had limited impact on the actual training and testing. These were identified when we ran the algorithm for the first time. Before it was figured out, it was problematic to see this issue ruin our several hours training process. Likewise, there were some images whose mask coordinates were not present in the metadata JSON file. Such exception issues were properly handled thereafter by making minor adjustments in the algorithm.

Initially while constructing our model, we used the traditional keras library for constructing and running our model. The version we used has an issue in its accuracy calculation which wasted quite a lot of our time understanding why the accuracy number quoted by the model was abnormal. This bug was long time undetected for hours of training that needed to be redone. The issue was fully fixed by using the tensorflow.keras most stable version.

Another highlight of a faced challenge was to make use of a keras library function called train_on_batch(). This function is normally not used in practice because of other available functions such as fit() and fit_generator() which automate most of the training process requiring the user to give only a few arguments. As discussed about the cost-effective weighing technique in section ‘Cost Sensitive Learning’, one potential way to implement was to give the ‘class_weights’ argument in the fit_generator() but looking at it more precisely, the training is always done in batches so it makes more sense to give class weights depending on the sample sizes for each class in the selected batch than on the overall data. In order to apply this, train_on_batch() function was used and the whole algorithm for training was written manually.
Another important risk that we can currently foresee is the system-setup. It is possible that the inbuilt camera resolution and the other environmental variables affect our model’s performance, or the training data fed to our model might not overlap completely with the real conditions, even after data augmentation, when the model is put in production. An example of such a problem is illustrated in figure 31 showing how much the lighting setting of different cameras could affect the data and hence may lead to erroneous results. The likelihood of facing this issue is decently high and we intend to make the model bypass such issues by potentially changing some parameters, altering the dataset as much as possible to mimic the real environment by possibly capturing some sample images at the Hong Kong airport, deciding specific camera settings, or making other relevant changes, if necessary.

Additionally, pre-trained ResNet-50 which starts to overfit after achieving 94% accuracy as shown under experiment 2 in table 4, performs poorly when the image contains a person. This is solely due to it being trained on the dataset with a limitation of highly zoomed-in images. This would no longer be a problem once Mask-RCNN or Yolo-v3 is ready to be deployed as these networks would just pass the exact object’s image (minimum bounding rectangle) to ResNet-50 or any classifier model as a part of the pipelined approach.

Mask R-CNN’s code was made to work only on one class (excluding background class) so it needed to be extended to handle more classes. While using the HKU GPU Farm (set up mentioned above), Mask R-CNN was not using GPU due to having incompatible CUDA and CUNN versions. This led to a lot of usage of time in training as each epoch took around an hour to complete. Hence, Mask R-CNN was also required to be downgraded to Tensorflow GPU 1.15 to match versions of CUDA and CNN. This helped bring down the epoch time to 5 minutes per epoch. Both of these tasks not only required a good
understanding of the model but also attention to detail to make sure any modifications did not affect the calculations of the model. Finally, F1-Score and mAP scores could not be calculated by using the keras library and had to be created using a custom algorithm which ran on the training and validation set after training the model.

5. Future Work

As discussed in the results section 4, the performance of most of the studied models is promising on the validation set but models are generally biased towards either hard or soft when it comes to classifying the images with multiple objects(for example humans). Keeping in view this obstacle in the deep learning literature where DNNs are usually too specific data and task dependent, and as discussed earlier, we plan on transforming our dataset that gives a more holistic picture of the airport environment.

A new dataset (called baggage detection dataset) must be created for both Aggregated and Pipelined Approach. This new dataset can have images similar to the COCO dataset to take full advantage of the transfer learning and global features already learned by the mask R-CNN model with COCO weights. This dataset would be multi-view as our previous dataset but have multiple baggage, more complex background containing people and unique environments. Baggage detection dataset would be made up of around 500 images with 80% public dataset (e.g. OIDv6, ImageNet) and copyright free images from the internet and 20% of the current dataset to get a more comprehensive view for our model of what a soft or hard bags would look like.

![Figure 32. Showcasing what images the new dataset may contain](image)

Baggage detection dataset would also be used to train the YOLO in the pipeline approach. While training on data depicting real environments is imperative, if Yolo-v3 or Mask R-CNN gives sufficient results for detection or segmentation respectively, random backgrounds is no longer a problem and therefore collection of images at the airport would
no longer be a requirement. This is because the custom model has already been trained on the current dataset (zoomed in images) and has the capacity to predict images very accurately (>96%).

The two special cases that would be needed to keep in mind for our dataset is that people tend to put handbags/jackets on top of their baggage and also use trolleys (seen in figure 33). Handbags and jackets are made up of clothes and are hence, soft in nature. This can confuse the model to incorrectly identify a hard baggage as soft when a jacket or handbag is present. Similarly, trolley is used by people to move many luggage around. As trolley is made up of metal or alloy, our model can incorrectly classify soft baggage as hard. This problem can be eliminated to a high degree by using images representing these two scenarios.

![Figure 33. MTR Airport Express containing the scenario of a) a person using a trolley on left b) people putting their handbag on their baggage on right.](image)

Outlier class as a mitigation strategy of overfitting issue has been implemented and the model is trained as such whereas it still needs to be monitored on the testing phase in order to analyse its performance. Likewise, another technique normally used for testing in classification problems as suggested in [21] is to make the model predict multiple variations of the original image. For example, given an image, augment the image by zooming in, setting a shear level and making other necessary changes to multiplicate and image into five different transformations and predicting the class of each of those images. This technique is also in line with the multiple camera view images discussed in section 3.2.
Remaining Work

After completing all the models, testing of various camera configurations (as mentioned in Methodology section) with two approaches would be done to evaluate which model performs better in most of the situation. The evaluation metric for this comparison for real life performance still needs research. It is only possible to know the exact image background, quality and other useful metrics once the type and configuration of cameras are finalized. By this time, we would already possess workable models and we are optimistic that with just some hyper tuning, the models would get adapted to the refined dataset. In order to test our model, we would visit the HKIA and a sample of the self-service bag drop system will be borrowed and testing would be done at the airport environment as seen in figure 34.

If the results from these testing are not satisfactory, our dataset will be modified to contain baggage in these environments by visiting the airport and collecting the images. Again, the idea is that transfer learning would help us by reducing the amount of images for the model to work in the airport environment successfully.

Figure 34. Different environments in Hong Kong International Airport our model

In terms of the project schedule set in our detailed project plan (see Section 5.1 below), we have achieved all the deliverables mentioned till January 2020.
## 5.1 Project Schedule

<table>
<thead>
<tr>
<th>Date</th>
<th>Deliverables</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>September 2020</td>
<td>Preparations and Deliverables of Phase 1:</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>● Detailed Project Plan</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Project Web Page</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Review:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● All details provided by HKAA on the self-service bag drop system.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Research:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Self-service bag drop system implemented in the airport</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Existing algorithm to achieve our objectives.</td>
<td></td>
</tr>
<tr>
<td>October 2020</td>
<td>Design:</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>● Basic System architecture of the solution</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Research:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Implementation of Yolo-v2, ResNet50, Inception-ResNet-v2 and Mask R-CNN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Camera for live feed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Implement:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Implementation of Mask R-CNN</td>
<td></td>
</tr>
<tr>
<td>November 2020</td>
<td>Research:</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>● Implementation of Yolo-v2, ResNet50, Inception-ResNet-v2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Environmental conditions in the airport where the model will perform</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Implement:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Train Yolo-v2, ResNet50, Inception-ResNet-v2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Data preparation and transformation (Augmentation)</td>
<td></td>
</tr>
<tr>
<td>December 2020</td>
<td>Preparations of Phase 2:</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>● Interim report</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● First presentation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Demo in the first presentation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Implement:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Target the classes sample size imbalance problem</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Evaluate the performance of ResNet50, Inception-ResNet-v2 and Mask R-CNN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Test the top two best performing models in a simulated environment of HKIA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>in HKU.</td>
<td></td>
</tr>
<tr>
<td>January 2020</td>
<td>Deliverables of Phase 2:</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>● Interim report</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● First presentation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Demo in the first presentation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Implement:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Fine tune the machine learning models</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Mitigate the overfitting issues in classification model - Pipeline approach</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Trying variants of mask R-CNN classifiers</td>
<td></td>
</tr>
</tbody>
</table>
6. Conclusion

This project aims to create a robust machine learning with fixed camera setup to identify baggage material as hard or soft. Two approaches were taken, namely the Pipelined model having Yolo-v3 with ResNet50 or Inception-ResNet-v2 and Aggregated approach of Mask R-CNN. Due to highly zoomed in/cropped images as a limitation of our dataset, Yolo-v3 training could not be performed due to lack of sufficient background for a minimum bounding rectangle to be drawn. Currently, only classifiers of pipelined approach and Mask R-CNN were implemented and tested on the custom multi-view Baggage dataset. Mask R-CNN was successful in predicting the segmentation area for baggage for images similar to the dataset but struggled when people or complex background in the images were incorporated. ResNet variants are mostly overfitting despite the mitigation strategies of data randomness (e.g. blurring, color filter, edge enhancement etc.) and outlier classes. A shallow custom model built from scratch taking the inspiration from VGG-16 shows promising results with the f1 score increasing(above 96%) even after 300 epoch training with the class weighing being used to handle the large class imbalance for hard and soft baggage images. A better suited dataset and tuning hyperparameters would help achieve image segmentation as well as detection for mask R-CNN and Yolo-v3, which remains to be implemented. With
proper comparisons and investigation completed in the future, the implementation phase will subsequently use the results to make informed decisions on the selection of approach to use with the camera setup and make HKIA a smarter airport.
7. References


difference-between-object-detection-semantic-segmentation-and-local.


