THE UNIVERSITY OF HONG KONG

DEPARTMENT OF COMPUTER SCIENCE

COMP4801 2020/21

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

Final Report

18th April 2020

Supervisor	Dr. T.W. Chim	
Report by	Utsav Raj	3035497915
Team Member	Muhammad Sheheryar Naveed	3035493672

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

Abstract

Facing stiff competition from other premier airports in the Middle East, Hong Kong International Airport has introduced the self-service bag drop system. Taking inspiration from this initiative, the objective of this project is to develop a deep machine learning model to identify baggage surfaces (as hard or soft) for this system using a mobile application. The integration of this model into the system would reduce wear and tear on soft baggage during handling as well as in flight by informing the passenger to use a tray. Three model types were used - Classifiers (ResNet50 and Custom ResNet), Mask R-CNN and Yolo (Yolo-V3 and Yolo-V5). These models were trained on a secondary dataset called MVB however, it soon became evident a primary dataset was needed for Mask R-CNN and Yolo. Hence, Baggage Surface Dataset was created to eliminate the limitation of MVB dataset being made up of highly cropped images. After all the training, these models were tested on a hand labelled test set made of challenging image with Custom ResNet model boasting 82% F1-score on the test set while 96.6% on the cropped test set. While Yolo-V5 also got a mAP score of 88% on the test set, only Mask R-CNN lacked behind due to technical difficulties. In order to facilitate the testing process conducted by Hong Kong Airport Authority,, a Smartphone Application was created. While the final decision rests on HKAA, we recommend the usage of Custom ResNet model due to it being faster and more accurate than both of the models - however, it requires cropped images to perform the best. With our Smartphone Application, the project aims to make passenger's life easier by protecting baggage.

Acknowledgements

Our team would like to offer a token of appreciation to our supervisor Dr T.W. Chim for his meticulous 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. Their willingness to give their time so generously has been very much appreciated. Last but not least, A special thanks goes to my teammate, Muhammad Sheheryar Naveed, for finishing this year long endeavour with me.

Table of Contents

Abbreviations

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

COCO	**Common Objects in Context**
FPN	**Feature Pyramid Network**
HKAA	**Hong Kong Airport Authority**
HKIA	**Hong Kong International Airport**
HKU	**The University of Hong Kong**
IEEE	**Institute of Electrical and Electronics Engineers**
R-CNN	**Region Convolutional Neural Network**
ROI	**Regions of Interest**
RPN	**Region Proposal Network**
ResNet	**Residual Neural Network**
SSD	**Single Shot Detector**
VGG	**Visual Geometry Group (VGG-19 is a trained CNN by this group from Oxford University)**

List of Figures

List of Tables

1. Introduction

With over 80 Best Airport Awards[1], Hong Kong International Airport (HKIA) is one of the premier airports in Asia and arguably in the world. However, there is an increasing competition from the state-of-art airports in the Middle East and South Asia. Nearly 70 million passengers[1] enjoy HKIA's services each year but the tough competition and growing number of passengers everyday means keeping up with technology to be able to provide exceptional services day in, day out.

In order to maintain its premier position in the 21st Century, HKIA has introduced projects using the latest technology to enhance the passenger experience and work processes and be transformed into the smart airport of the future. One of the projects that focuses on this smart future is the Self-service bag drop system.

Figure 1. One of the self-service bag drop systems.[2] With over 120 systems, this is one of the largest projects of its kind.

Self-service bag drop system (as seen in figure 1) was initiated in 2016 to shorten the long queues needed for manual handling of bags at the check-in counter. This system not only helps the airport in speeding up the departure process from three minutes to seventy seconds[2] but also reduces the bag handling cost for HKIA.[3] Passengers who have checked-in earlier with either a self-service kiosk or web-based check-in are required to follow these instructions below to self-drop their baggage:

1. Scan their boarding pass barcode and get a luggage tag to place it on their bags [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 [4]
3. Confirm the details on the screen to send the bags to the main conveyor belt [4]

With the current COVID-19 pandemic, contactless self-service bag drop system allows reduced human interaction at the airport, reducing the chances of contracting the virus for passengers and staff alike. One major problem with self-service bag drop systems occurs during step 2 mentioned in the workings of the system. When passengers put any soft material luggage (refer to figure 2) it tends to get damaged (e.g. abrasions) during handling if not put in a tray first. Currently, soft baggage is put into a tray manually by airport staff during handling and any oversight can cause customer inconveniences by damage to the baggage but damage to the HKIA's service quality as well.

Figure 2. Example of hard baggage and soft baggage (with and without abrasions)

Our project aims to plan, design, develop and implement a state-of-the art machine learning model which will use a mobile application to provide an image to the model to identify the baggage surface as soft or hard. If a passenger has soft baggage, the screen would prompt them to use a tray before putting their baggage on the conveyor belt of the self-service bag drop counter. A soft baggage is made with either woven nylon (e.g. cordura, ballistic, or ripstop) or leather while hard baggage is made of plastics (ABS or polycarbonate) or aluminium for the scope of this project (refer to figure 2). The following models will be used and their performances compared - Mask R-CNN, Yolo and Classifiers like Custom ResNet50 and VGG.

The remaining report is organized as follows. This section continues with a subsection on research background of key terms identifying the capabilities of the each model while the Section 2 surveys existing research done on baggage material in the literature. Section 3

provides the implementation approach for the two training dataset, smartphone application and the three model types – Classifiers, Mask R-CNN and Yolo. Then, Section 4 compares the implemented models and tries to make an educated judgement on which model type is the best choice by using F1-Score and mAP. Section 5 discussing the future work of on how the project could be extended or improved and finally conclusion in Section 6 .

1.1 Research Background

The key machine learning techniques separating the three models are listed below with figure 3 providing an easy to understand visualization of the same:

- Object recognition: Computer vision technique to recognize varying classes of objects in an image or in a video[5]. This can be achieved with models like Resnet50.
- Object detection: Identifies an object in an image and specifies its location using a bounding box [5] (often a minimum bounding rectangle). This can be achieved with models like Yolo-v3.
- Semantic segmentation Provides a more detailed approximation of an object by classifying every pixel in an image as a class and then creating a mask (blue mask in figure 3) on that object in an image[5]. This can be achieved with models like Mask R-CNN.

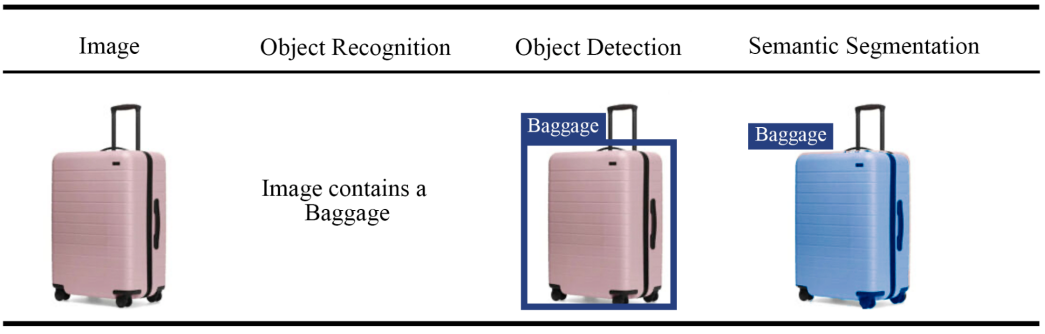


Figure 3. Visualization of three computer vision techniques on a baggage image

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[6], to find mishandled bags by creating an automated database[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[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[6]. Likewise, we will use the pre-trained models, originally trained on the same COCO dataset this study uses except for Yolo to be used instead of RetinaNet.

The second 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[7]. This project 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 raw data collection, dataset creation and augmentation in order to train the three baggage surface material identification models. The block diagram in figure 4 illustrates the high-level flow of the methodology undertaken for this training of the different models for this project. The dataset is discussed in detail in Subsection 3.1, then the model types in Subsection 3.2, followed by mobile application explained in Subsection 3.3.

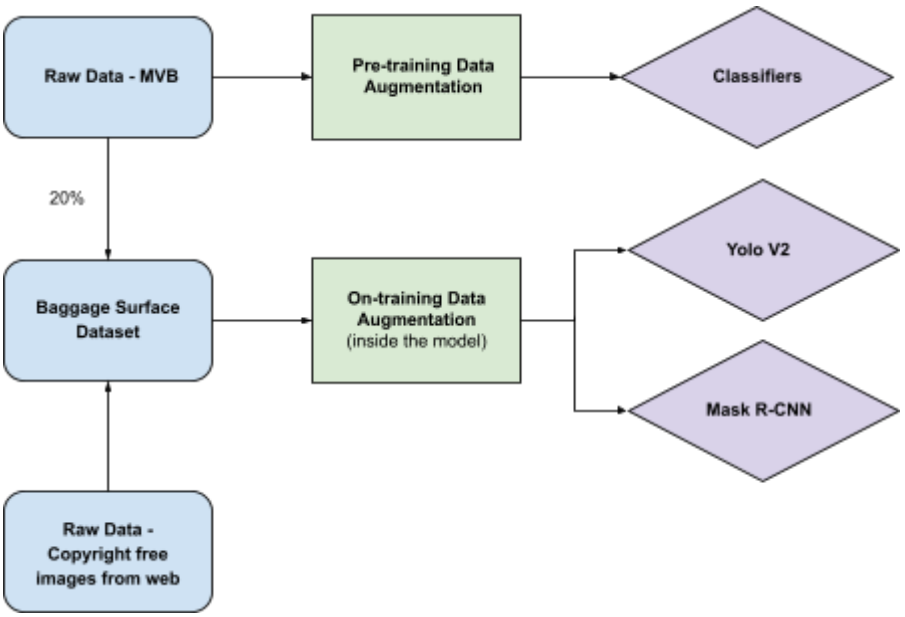


Figure 4. Block Diagram of the datasets and Models

3.1. Dataset

Preferably, the dataset used for training needs images of baggage at different angles to achieve the best possible results. HKIA does not have an existing mechanism to capture baggage images and with privacy concerns during collection, secondary source dataset and primary source dataset were created.

3.1.1. Secondary Source dataset - MVB

The secondary source dataset named MVB dataset, contains 4,500 unique baggage identities with around 22,000 annotated baggage images taken at different angles with multi view cameras [8]. The labelling of the image was stored in VGG format 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).

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.

The dataset was created for baggage re-identification and hence had three material labels - soft, hard and others. Thus, these unwanted 'others' labelled images were removed from the dataset as a part of the data-cleaning stage. As the number of these images were low, the clean dataset had around 18,000 images with the baggage identities remaining around 4,200 as seen in figure 5.

3.1.2. Primary Source dataset - Baggage Surface Dataset

This is a custom hand-annotated dataset specifically made to counter the limitations of the MVB dataset (see Section 4.2.2 for more details) for segmentation as well as object detection - Referred to as Baggage Surface Dataset henceforth. Annotation was done using VGG Image annotator (version 2.0.11) as it is easy to use, does not require any installation or setup and is used by authors of MVB dataset, which helps keep consistency when using both simultaneously. Any images added from MVB dataset to this new dataset were re-annotated.

Images were taken from publicly available dataset such as ImageNet, OIDv6 and ADE20K as well as from free stock image database websites like Freepix, Pixabay, eBay, Flickr and Pinterest. However, images from the COCO dataset are not used because COCO weights are used when training on the Baggage Surface Dataset. Baggage Surface Dataset contains 20% of MVB dataset in order to tackle the limitation of COCO train models in detecting close up images of baggage while containing images with diverse backgrounds like Airport, Beaches, Room and images with hard and soft identities together as seen in figure below. More than 20% of MVB dataset is not added as it starts to suffer similar problems

when using MVB dataset of bigger bounding box and segmentation mask as seen in Section 4.2.2.

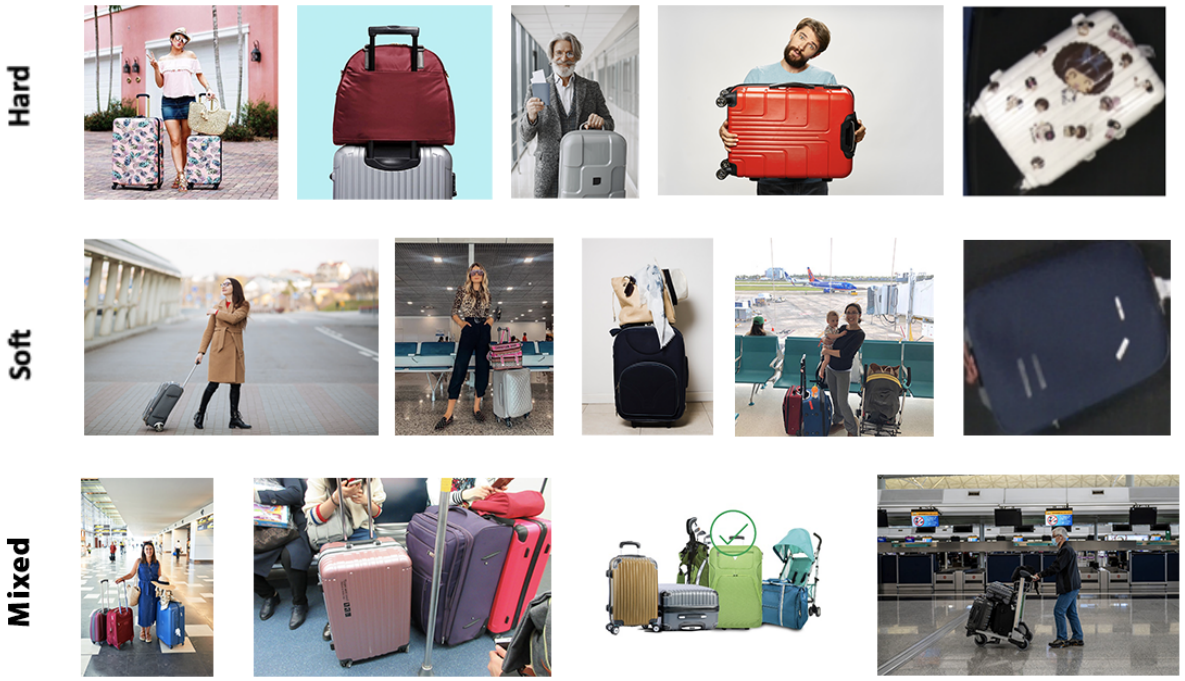


Figure 6. Sample images from Baggage Surface dataset. It contains images with multiple hard or soft baggages or even hard and soft baggages in the same images (Mixed)

Baggage Surface Dataset was increased by 3 stages in total in order to test if increase in images have any significant impact on model performance. The stages themselves are increased in batches as it is best practice to see if increasing the dataset with new images is working properly. These stages are named based on the approximate number of images in the train set - more information can be found in table 1.

Dataset	# of Images	Train/Test Split	# of Hard Identities	# of Soft Identities
Baggage Surface-900	1,099	909/190	836	835
Baggage Surface-1700	1,934	1,744/190	1,416	1,415
Baggage Surface-2000	2,200	1.999/201	1,626	1,632

Table 1. Annotation Statistics of Baggage Surface Dataset

3.1.3. Test set

This set was specifically created to see the performances for Mask R-CNN and Yolo models on never-seen-before challenging images as seen from the figure 7 below.

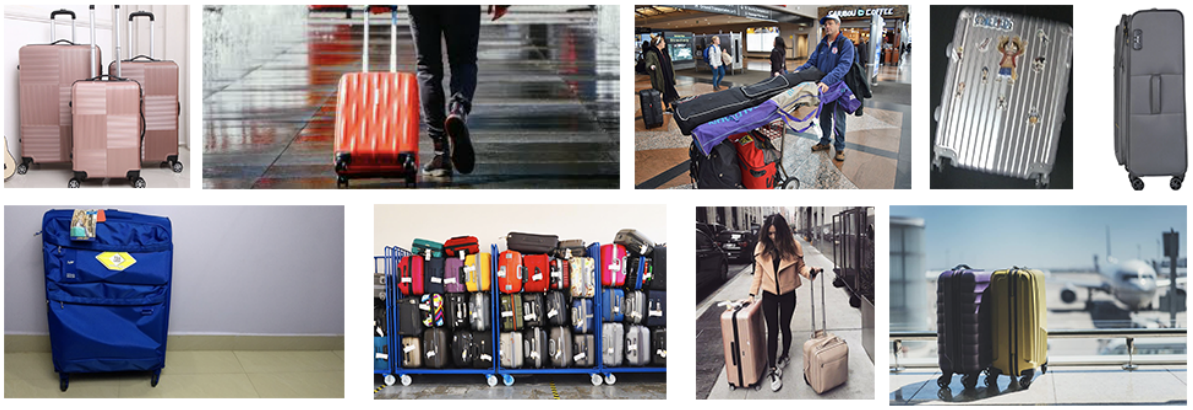


Figure 7. Sample images from the test set

Cropped version of this set is used for the classifiers (see figure below) along with an uncropped version for Mask R-CNN and Yolo to compare their performances to make choices on which model suits which situation better. These cropped images are edited by hand to imitate how a user may crop images once they take an image from their camera.

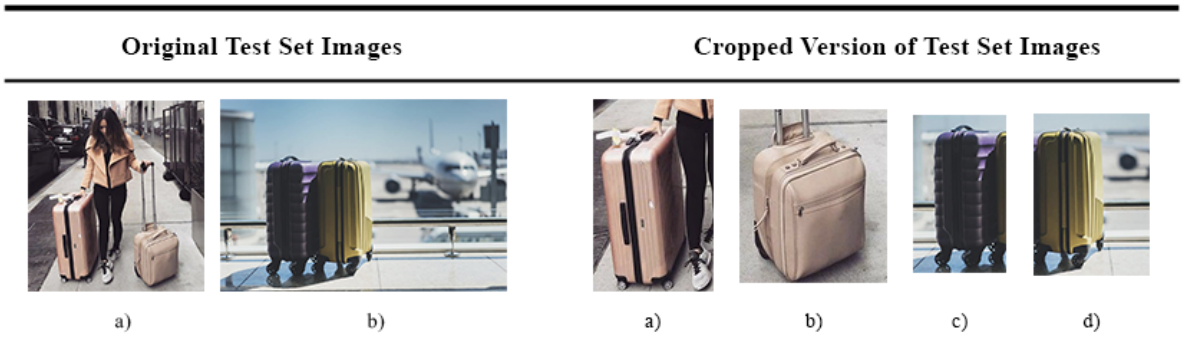


Figure 8. Original test set images a) and b) are cropped to get the only baggage object as new images as seen in the section a), b), c), d) of the cropped version of test set images

More information about the uncropped test set can be found in table 2.

Dataset	# of Images	# of Hard Identities	# of Soft Identities
Test Set	260	218	201

Table 2. Statistics about the test set. Please note that the number of images is equal to the total number of identities for the cropped version of the test set and the number of identities stays the same.

3.1.4. Data Augmentation

Data augmentation is a technique to create new training images artificially from existing training images[25] - increasing images without collecting any new images. This can be achieved by transformation on images such as flipping, rotation, increasing contrast, adding blur and many more. These techniques have been seen to improve the accuracies of the model in the range +2 to 31%[16] Data Augmentation helps reduce overfitting as it helps create images that create a variety of conditions that may not be accounted for in the original images with a limited set of conditions.

Both, the MVB dataset used for the Classifiers as well as the Baggage Surface dataset used for Yolo and Mask R-CNN use different augmentation techniques to improve their performances. However, Mask R-CNN and Yolo both use on-training augmentation when training by selecting random images and augment on-the-fly. This is beneficial as these models need complex calculations involving the mask or bounding box especially for geometric transformations like rotation.

Classifiers are object recognition models and do not rely on bounding boxes or masks and thus receive pre-training augmented images for training purposes - the data is manipulated before the training process and then kept constant for the remainder of the experiment. More details on exact augmentation used for each model are provided in their respective sections.

3.2 Models

There are three main models used during this project. These are:

- Classifiers: Resnets and Custom Model
- Mask R-CNN
- Yolo: Yolo Version 3 and Yolo Version 5

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.

The accuracy of all the models will be tested by determining the F1 scores. F1 score is an effective 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[14]. 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 the predicted mask or bounding box and actual mask or bounding box overlap by 50% or more, it is considered a true positive. For both of these metrics, higher values are more desired.

3.2.1 Classifiers

Two different classification networks namely ResNet50 and Custom Resnet Model were used and compared. More detail about these networks is further discussed in the subsections of 3.2.1 below.

3.2.1.1 Prevention of Overfitting

The classifier models have many parameters, making training these models a challenging task. These state of the art models are designed to classify hundreds of models and may lead to overfitting for the scope of the project which is only soft and hard classes. Then again, these state of the art models can take advantage of transfer learning to extract features from images quite easily. Techniques to limit the effect of over-fitting are mentioned below.

3.2.1.2 Data Augmentation

The MVB dataset was manipulated using the python pillow library with the following methods (refer to figure 10):

- Coloring: Conversion from BGR to RGB was used where R is Red, G is Green and B is Blue. This means that Red is converted to Blue (as seen in Figure 9) and vice versa. Similar design bags with different colours are possible and hence, models should be able to work in these cases.

Figure 9. Color filter comparison - Original image (Left), Altered image (Right)

- Rotation: Images were rotated with maximum ration set to 50%. This allows creating baggage images in different angles than the raw MVB dataset.
- Noise: By adding noise in the image, low quality camera were being emulated, which have sharper edge recognition of the object but lower image quality,
- Lighting: Varied lighting conditions allows for a robust model. These conditions were recreated by increasing or decreasing brightness by 40% and adding linear noise to colour component of each pixel separately
- Blur: Image taken in a hurry or camera moved while taking a picture is a common occurrence which can result in blur. Additionally, blur can simulate the network's performance on small or distant objects that will be captured with low resolution.
- Flipping and Zoom: Flipping the image horizontally and vertically as well as zooming in and out was done.

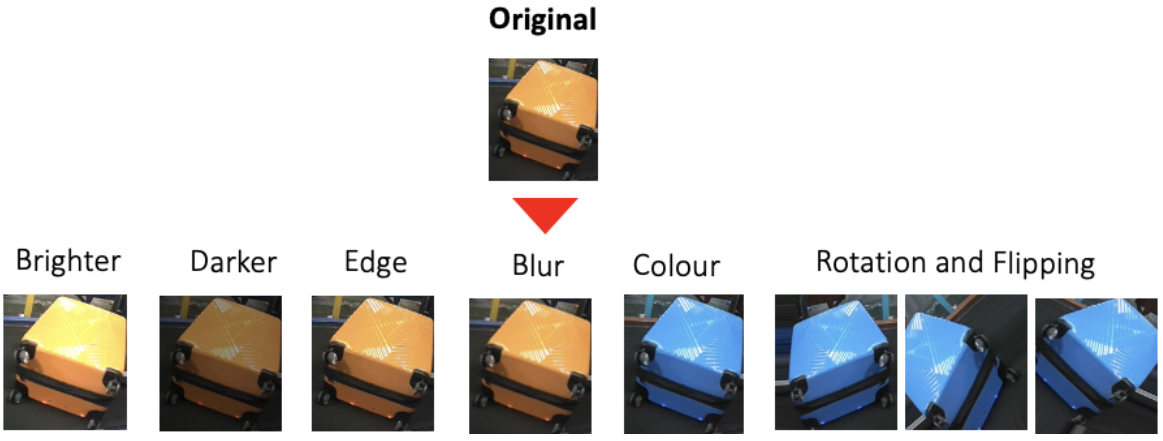


Figure 10. Visualisation of all data manipulation methods

There are over 12k hard images as opposed to only 5k soft images. In order to ensure there is no bias in the model, three new datasets using the mentioned augmentation were created (see table 3):

- Random minority Over Sampling: Randomly choosing the samples from each of the available augmented sets of soft images to increase the total size of soft images to ~11,000.
- Random majority Under Sampling: Randomly choosing 4,500 hundred images from the hard baggage image thus making a total dataset equal to 4,500 training images for both classes.
- Complete Sampling: Keeping the imbalance with the intention of using a cost-sensitive learning[17] explained in the next subsection.

		Over Sampling	Under Sampling	Complete Sampling
Train	**Hard**	11,000	4,500	12,000
	Soft	11,000	4,500	4,000
Validation	**Hard**	1,000	300	501
	Soft	1,000	300	888
Test	**Hard**	500	134	30
	Soft	500	88	34
Total	**Hard**	12,500	4,934	12,531
	Soft	12,500	4,888	4,922

Table 3. Precise summary of datasets used for classifiers

Cost Sensitive Learning:

Cost Sensitive Learning assigns different costs to misclassification of examples from different classes[18]. Aim of this method is to adjust weights in the objective loss function for training samples from different classes. One of the standard approaches is called backward pass of backpropagation algorithm, which assigns weights according to inverse class frequency by assigning higher weights for wrongly predicted samples. Classifier approach adapts 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.

With unequal samples in classes, the performance is measured by average cost per example rather than with the error rate only. It is possible to see a very small error rate but good 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]:

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

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 samples per class), the Cost function would be the following:

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

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

Error rate = Number of incorrectly classified examples / N

Accuracy = 1.0 − Error rate

If the costs for all classes is non-uniform, the average cost would be a true reflection of the model and, hence is a better fit for model performance criterion. Deep neural networks assume equal weightage for all classes by default - a bad case when the sample for each class varies as explained in figure 6. Hence, average cost is a better performance criterion over traditional error rate for this unbalanced case.

Cost[soft, hard] = 0.75 *(For 1 hard, 3 soft)*	mages = 5 (3 hard and 2 soft)
Cost[hard, soft] = 0.25	2 soft incorrectly predicted
# of train soft bag images = 4000	= (0+0+0+0.75+0.75)/5 = 30%
# of train hard bag images = 12000	rror rate = (0+0+0+0.5+0.5)/5 = 20%

Average cost vs error rate for imbalanced class sample sizes

Calculating Class Weights

For implementation of this strategy, different class weights would be set that affects the loss of the two classes separately and thus, affect the overall loss. The class with less samples (for our case, soft) would have a greater impact on the overall loss of a batch as the penalty of wrong prediction of the soft bag would carry a higher penalty. 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 normalisation taken into account.

$$\eta(p) = \frac{\eta \cdot CostVector[class(p)]}{\max_i CostVector[i]} \qquad (A)$$

$$\text{Ratio} = \frac{\#Samples\ of\ soft\ bags}{\#Samples\ of\ hard\ bags} \qquad (B)$$

Each class sample is computed and these results are normalised to ensure that the convergence of the back-propagation model is not affected. Here the cost vector is the expected cost of misclassifying a sample belonging to i^{th} class and is computed as follows, where the function P is the estimate of the prior probability of the example belonging to i^{th} class:

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

Assuming, number of hard samples are 12,000 and number of soft samples are 4,000

$Cost$[soft, hard] = 0.75 *(1 hard – 3 soft = ¾ = 0.75 ~ normalizing)*

$$CostVector[soft] = \frac{1}{1-0.25} \,.\, (0.75) \,.\, 0.75 \;=\; 0.75$$

P(i) here could be visualized as the fraction of i^{th} class out of the total number of samples.

Calculating the Loss After Setting Class Weights

For every training batch, formula (C) shows the traditional loss. 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) seen below.

$$\text{Loss} = \frac{\Sigma\, L(y_i)}{Batch\ Size} \qquad \text{(C)}$$

$$\text{Loss} = \frac{\Sigma\, W_{y_i}\, L(y_i)}{\Sigma\, W_{y_i}} \qquad \text{(D)}$$

Where using data in figure 6, $W_{yi} = 0.75$ for $y_i = soft$

$W_{yi} = 0.25$ for $y_i = hard$

In formula (D), Sum of the weights rather than the batch size is in the denominator - sum of weights would be normalised and comparable to what is deduced in the numerator. The logic behind this strategy of cost effective weighing is by giving a higher weight 'x' to soft baggage images. It is done to create the visualisation for the model that if it wrongly classifies one soft bag, it has misclassified 'x' number of soft bags instead. As higher magnitude of weight of soft baggage in the loss calculation, higher gradients during the gradient descent in the back propagation will be observed.

3.2.1.2.3 ResNet-50

ResNet-50 is a 50 layered classification network which does not suffer from diminishing gradient - a problem where the model fails to learn when the implemented network is deep. This is achieved by skipping the connection and passing the residual to the next layer [16]. With the residual technique, the implementation of deep neural networks

became plausible while eliminating the exploding or diminishing gradient issuem. An explanation on how the residual is added to the next layer is illustrated in the figure 11 below.

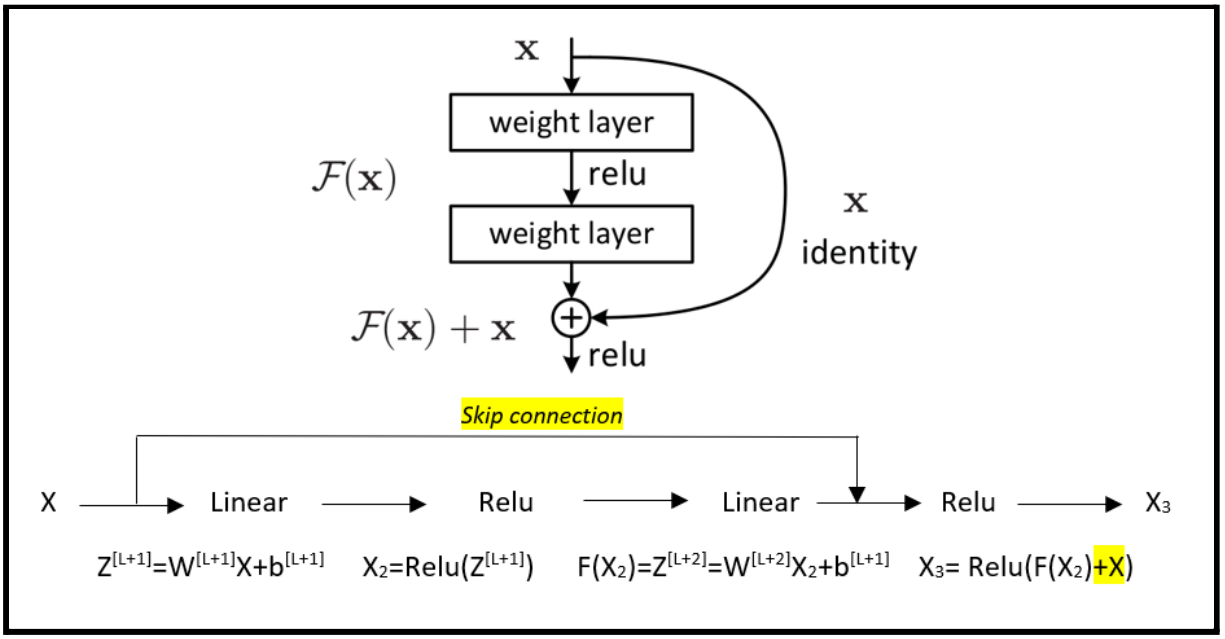
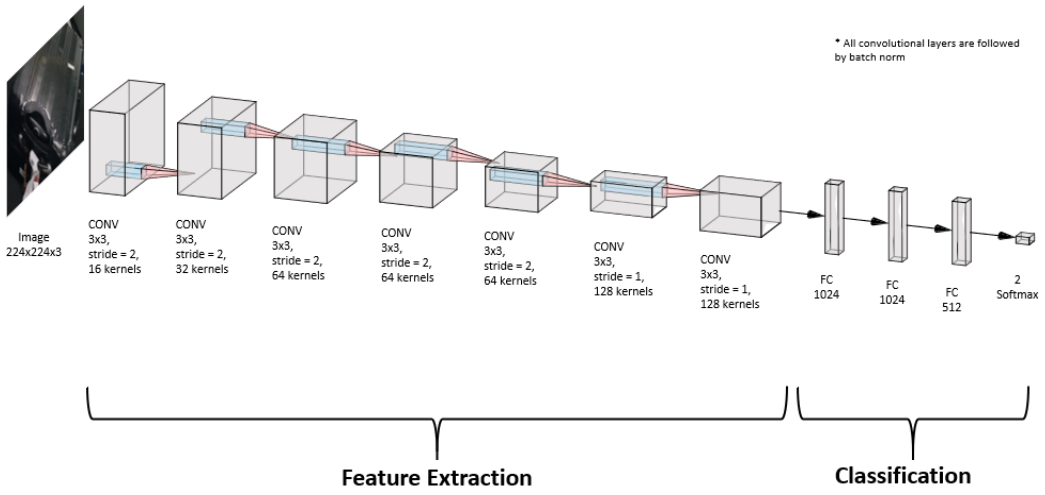


Figure 11. ResNet residual block skip connection implementation

3.2.1.2.4 Custom Model

Apart from the above models, a custom-built model was created. Keeping in mind that only two classes are being classified and the overfitting problem mentioned in Section 3.2.1.1, it was important to create such a model to see if there is any possibility of training our model even better for this specific problem.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 111, 111, 16)      448
_________________________________________________________________
batch_normalization_8 (Batch (None, 111, 111, 16)      64
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 55, 55, 32)        4640
_________________________________________________________________
batch_normalization_9 (Batch (None, 55, 55, 32)        128
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 27, 27, 64)        18496
_________________________________________________________________
batch_normalization_10 (Batc (None, 27, 27, 64)        256
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 13, 13, 64)        36928
_________________________________________________________________
batch_normalization_11 (Batc (None, 13, 13, 64)        256
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 6, 6, 64)          36928
_________________________________________________________________
batch_normalization_12 (Batc (None, 6, 6, 64)          256
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 4, 4, 128)         73856
_________________________________________________________________
batch_normalization_13 (Batc (None, 4, 4, 128)         512
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 2, 2, 128)         147584
_________________________________________________________________
batch_normalization_14 (Batc (None, 2, 2, 128)         512
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 1024)              525312
_________________________________________________________________
dense_5 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_6 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_7 (Dense)              (None, 2)                 1026
=================================================================
Total params: 2,421,602
Trainable params: 2,420,610
Non-trainable params: 992
_________________________________________________________________
```

Figure 12. Summary and Architecture of Custom CNN Network

The custom model architecture is shown in figure 12 - 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.

Several CNNs include pooling layers in their architecture, this is favoured as long as losing positional information is fine by artificially reducing the resolution of the image further after certain processing steps. However, this would result in more computational steps for the previous convolutional layer. These problems are mitigated by using convolutional layer of a stride 2, which stabilises performance based on University of Freiburg study[20] on several key image recognition benchmarks. Hence, strides of (2,2) were kept instead of adding max pooling layers.

In the start, the train f1-score was stuck at 85% even after several epochs with only five convolutional layers with dropouts. To enhance the learning, additional two convolutional layers were added, resulting in better performance after long periods of training with dropouts removed due to no overfitting. These results are elaborated in Section 4.

3.2.2 Mask R-CNN

Mask R-CNN 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.

3.2.2.1 Architecture

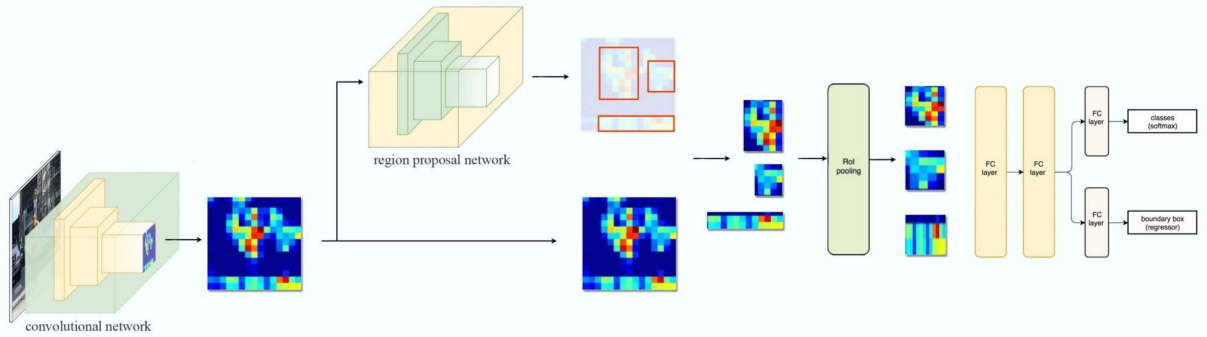


Figure 13. Block diagram of Mask R-CNN architecture. [21] ***It has 4 stages - Backbone, Region Proposal Network, ROI Classifier & Bounding Box Regressor and Segmentation Masks.***

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.

- 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.[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.[22]

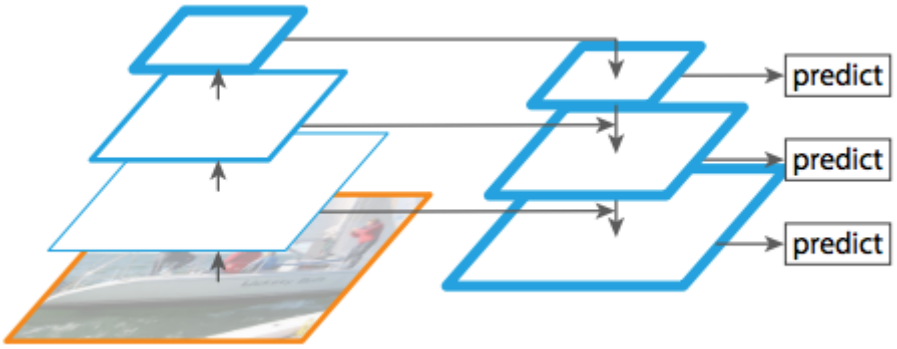


Figure 14. Feature Pyramid Network allowing pyramids to see high level and low level features

- Region Proposal Network (RPN)

The boxes in the figure below are the anchors. As shown on the left, these boxes are distributed over the image area. 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.[22]

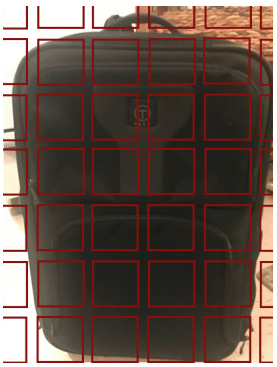


Figure 15. Simplified illustration showing 42 anchor boxes

The Region Proposal Network (RPN) is a light-weight neural network that can quickly scan the image to determine areas (called anchors) containing the objects in a sliding-window fashion. Furthermore, 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]

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).[22] These are called the final proposals (regions of interest) and are passed onto the next stage.

❖ 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:

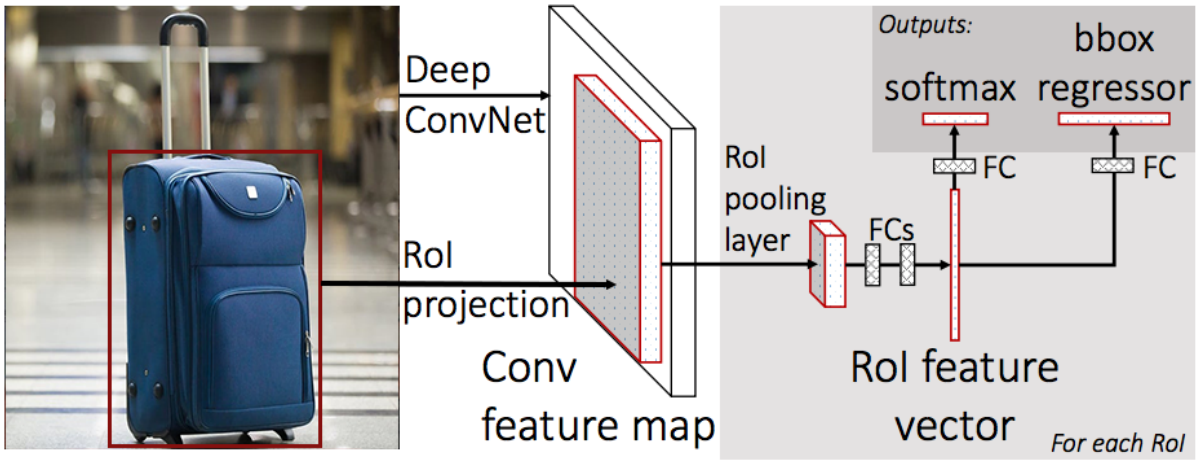


Figure 17. Illustration of stage 3 of Mask R-CNN. The red box represents the ROI.

- **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.

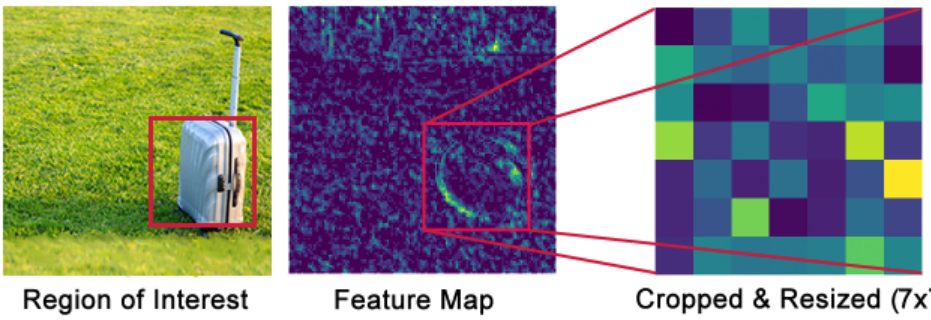


Figure 18. Getting a fixed size feature map from ROI. This feature is a low level layer for ease of understanding

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

- ❖ Segmentation Masks

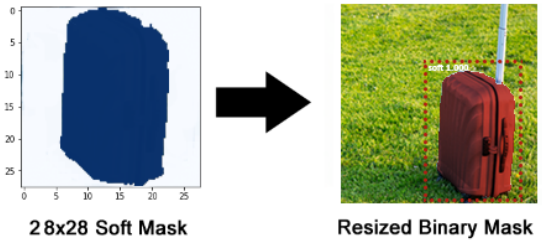


Figure 19. Scaling up the 28x28 soft mask to fit our baggage image

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 s*oft* masks are represented by float numbers, to hold more details than binary masks.[22] 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[21] :

- *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 is done
- *Mrcnn mask loss*: how well the segmentation of classified objects using mask is done.

3.2.2.2 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[23] as seen in Figure 20.

Figure 20. Sample images for class suitcase from COCO 2014 dataset[23]

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.

3.2.3 Yolo

Outperforming top methods like Faster R-CNN with ResNet and SSD[10], Two versions of Yolo (Yolo-V3 and Yolo-V5) were used as an object detection network. Given the research work published under IEEE[11] that the pre-trained Yolo is substantially better in detecting most of the common types of objects including baggages, transfer learning was prioritised. Yolo-v3 was trained on the self-collected stage-1 ‘Baggage Surface-900’ dataset

whereas Yolo-v5 was trained on stage-2 ‘Baggage Surface-1700’ and stage-3 ‘Baggage Surface-2000’ in table-1 described in Section 3.1.2. The Subsection below describes the architecture of Yolo.

3.2.3.1 Architecture of Yolo

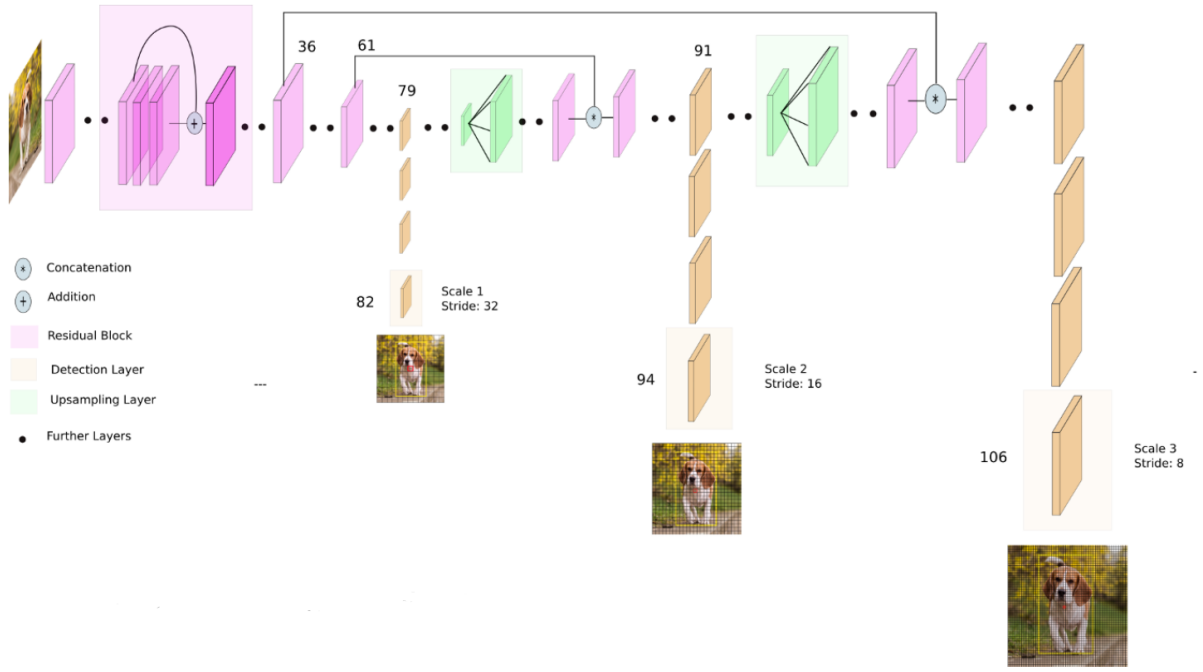


Figure 21. Visualisation of Yolo architecture.

3.2.3.2 Backbone

As seen in figure 21 above, the backbone of Yolo-v3 consists of 106 layers, which boosts its speed by 30% compared to Yolo-v2. Yolo-v3 performs detection of the object at three scales by downsampling the image by slides of 32, 16, 8 - scale corresponding to the size of the object it can detect with bigger slides for larger objects. Stride refers to the ratio by which the layers downsampled the image. Detection depends on the anchor boxes. From the same figure as before, it is noticed that upsampling is performed after each scale. This upsampling concatenated with the previous layers helps preserve the fine grained features required to detect small objects - a feature lacking in previous versions.

3.2.3.3 Non-max Suppression

Based on the calculation for the output of the kernel at three different phases of the network being applied at three different feature maps, there is a possibility that one object be

bounded by multiple bounding boxes covering several different regions partially or fully around the object - visualized in figure 22.

Figure 22. Illustrates how non-max suppression chooses the best bounding box

In order to reduce the excessive bounding boxes, the Non-max suppression algorithm is used. This algorithm removes the bounding boxes whose Intersection over Union score is more than the given threshold (e.g. near other bounding boxes) relative to the bounding box with the highest confidence score. In the beginning, to save computation time, all the bounding boxes with confidence scores lower than a certain probability threshold are already removed. This algorithm generates the highest confidence bounding box per object predicted by Yolo from a wide pool of weaker predictions.

3.3 Smartphone Application

Although, the initial scope of the project was to create a fixed camera setup mounted near self-service bag drop system. However, it was realised after the interim presentation that creating a mobile application would be more beneficial due to the following reasons:

- As Passengers are using their own smartphones, HKAA does not need to buy a camera for each self baggage drop service kiosk, reducing their costs.
- Users can easily adjust their smartphone camera unlike the fixed camera setup.
- Allow Airport Staff to easily test and monitor model's performance using their smartphone in the airport in varying environments.
- The Baggage Identification App can be easily integrated into the existing HKG My Flight application available for iOS and android devices.

Hence, after receiving approval from the HKAA, Mobile Application was created instead of the initial camera setup.

3.3.1 Frontend

The frontend of this application was created in React Native as it is the highest performing cross-platform (iOS and Android) frameworks, integration with Expo allows for live reloading (easier testing) and both of us have experience in creating robust applications using the framework. Please keep in mind that this Application is for testing purposes so that HKAA staff members can try the model predictions - although, they are more than welcome to use elements from this user interface for their final application. The following flowchart outlines the user journey when using the application. More details about the user journey are outlined in the next section.

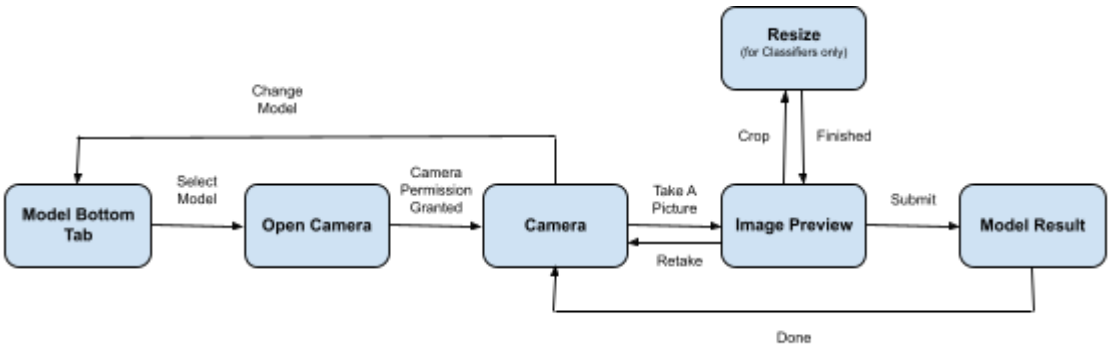


Figure 23: Flowchart of User Journey from opening the app to getting results.

3.3.1.1 Application User Journey

When the user opens the application, by default, they are in the classifier model tab as seen in figure 24. Users are instructed to remove any covers from baggage or otherwise, the surface may be misclassified. Only the classifier model has additional instruction to take a close-up photo or crop the photo as it cannot detect objects and then classify them. In the same figure, on the bottom panel, the user can choose to switch models to Mask R-CNN and YOLO using the bottom-middle and bottom-right buttons respectively.

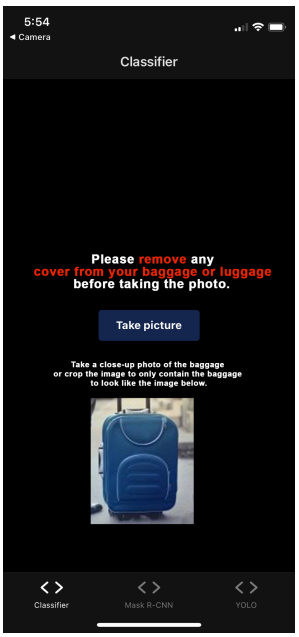


Figure 24: Default first page (classifier model tab) user sees when they open the app.

Figure 25: Page when user presses the button in figure 23 and grants camera permissions.

Referring to figure 25, based on the buttons at the bottom, the user has the ability to either retake the photo which reopens the camera or submit this photo to the cloud. Only in the classifier tab, the option to crop the image is available because unlike Yolo and Mask R-CNN, classifiers only do classification and not object detection and hence tend to take all background objects into account when performing prediction in images seen in example in figure 26. Pressing on the crop menu opens the resize menu as depicted in figure 26. The top

panel has the option from the left to right - go back without cropping, crop, rotate by 90 degrees, flip horizontally and flip vertically. By either pressing Done or Go Back button, the user is returned to screen as seen in figure 25.

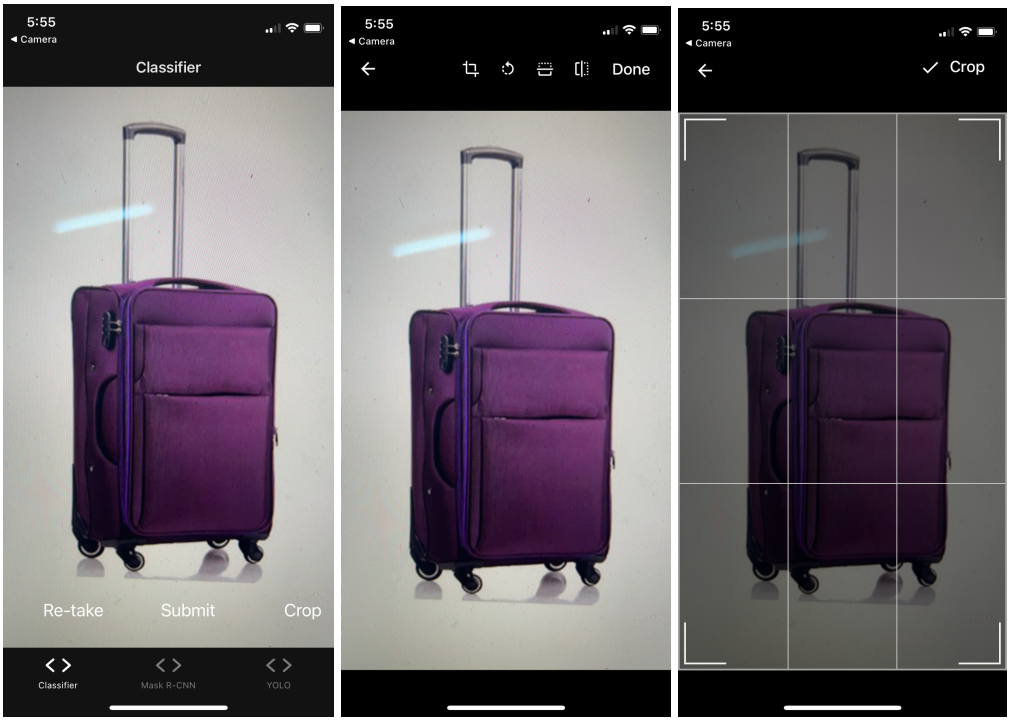


Figure 26: From left to right: Image preview when user takes a photo, when user presses crop button in classifier tab and user manually cropping the image

Once submit button is pressed, as shown in figure 26, a loading page is displayed until the response from the cloud is successfully received we see figure 27 based on the model used before - all of them shows the classification label, confidence score of the label predicted, and time taken by the model to do the prediction (not to be confused with the total time taken for the model in the server to send a response to the application).

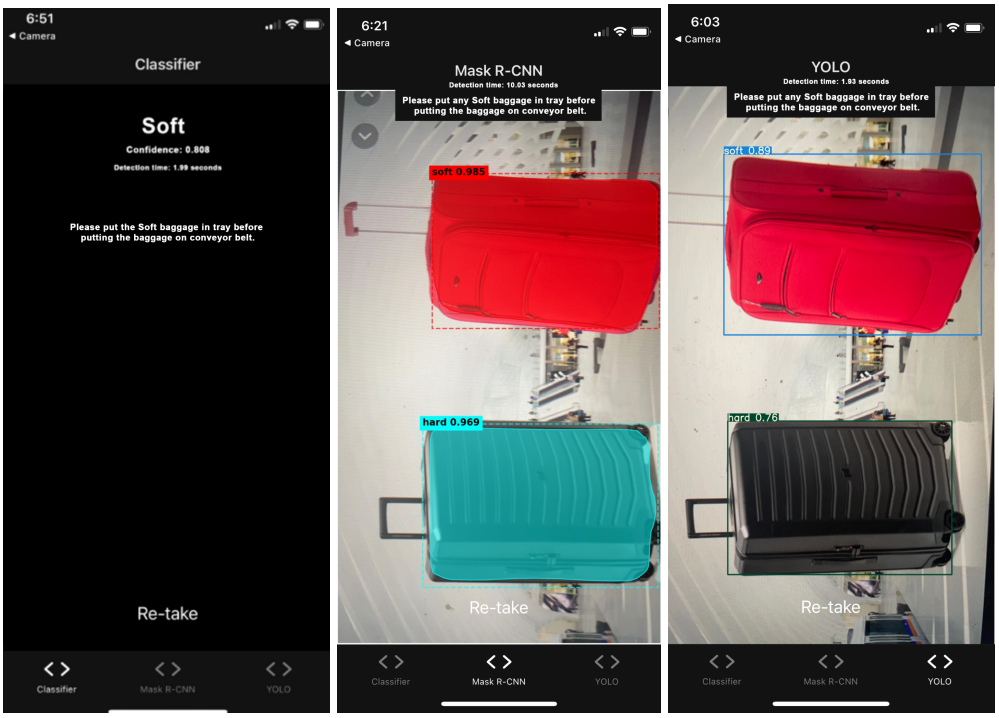


Figure 27 From left to right: Results from each of the three model - Classifier (purple bag image from Figure 26), Mask R-CNN and Yolo-V5

3.3.2 Backend

The backend of the application is made from individual virtual machine instances where each of the models was maintained in its own environment. CPU was used instead of GPU due to the additional pricing required by Amazon Web Services. Hence, there were three instances supporting the best version of each type of the model. Please note that for the model type classifiers, since ResNet was the most optimal model in terms of testing, all of its versions were deployed altogether on a single machine.

Each of the three types of network are deployed on separate instances to mimic the microservice architecture. This was also kept in this way so as to support alteration of physical power (e.g. different versions of GPU and number of CPUs) on the instance as each instance is essentially a Ubuntu linux based virtual machine. While Yolo-V5 and Mask R-CNN are served using a python based framework called ‘Flask’, classifiers are deployed using a pytorch framework called ‘torchserve’ described in figure 28 below. This framework provides inference and management APIs for deploying multiple models and number of workers allocated for each model.

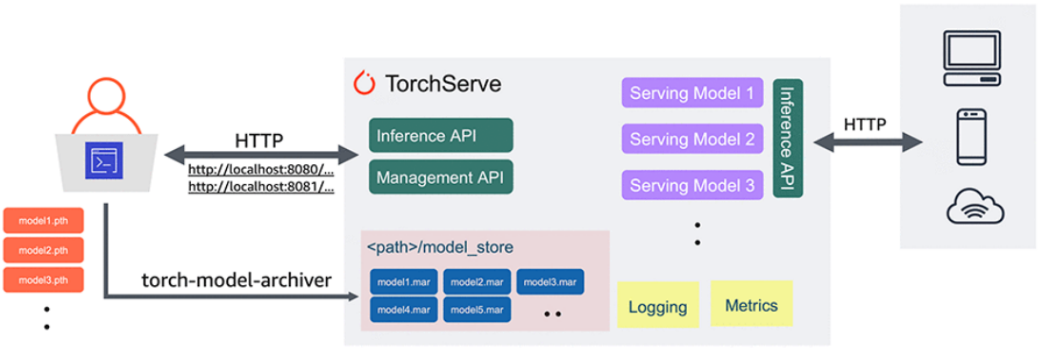


Figure 28. How TorchServe handles Classifiers[30]

The response is returned in JSON format and it consists of the baggage class (hard & soft) probabilities calculated using average score of all the variants and response time for classifiers. Only the response time for Yolo and Mask R-CNN. For Yolo and Mask R-CNN, the server also puts the predicted image with a bounding box or segmented mask in a static directory in the instance hosted as a separate endpoint by flask . This is then fetched by the frontend client after receiving the response to its initial POST request.

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 Classifiers followed by Section 4.2 that describes the Mask R-CNN and Subsection 4.3 that talks about Yolo. Subsection 4.4 compares the model type described before while Subsection 4.5 contains difficulties encountered while working on this project.

4.1 Classifiers Results

For implementation and testing of Classifier and Yolo, 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.

4.1.1 Network Training and Hyperparameter Selection

SGD was used as it is the best optimiser by providing the most generalised model. A callback was created to reduce learning rate if for the next three epochs, the validation loss increases, in order to make learning rate follow validation loss's trend.

Custom Model:

The custom model was trained on a complete sampling dataset by setting class weights. As figure 29 still showed a declining trend, epoch was increased from initial 100 to additional 200 epoch to see how much maximum f1-score could be achieved. The validation f1-score reached 93% while training f1-score reached 94%. Surprisingly, the declining trend in training loss still persisted. Before increasing epoch further, weights of the epoch with the minimum loss were loaded and the model was tested. While the high f1-scores shows promise, Proto-test set shown in table 4 produces f1-score of only 55%. It is important to note that sample size of the image was small and images consisted of multiple objects which may interfere with prediction. Hence, justifying the need of the image being cropped before being sent to the classifier models.

Dataset	# of Images	# of Soft Identities	# of Hard Identities
Proto-Test set	64	30	34

Table 4. Proto-Test set based on web images - test set having 64 images at that point in time

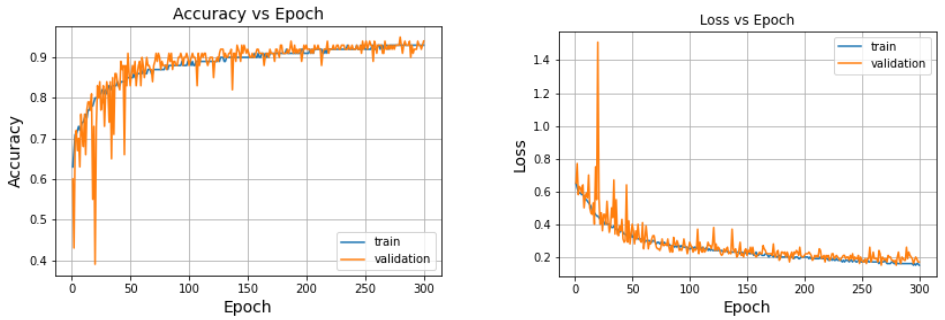


Figure 29. Custom Model on complete sampling data

Due to the long time needed for training using the complete sampling data, the under sampling data was used to see how much training can improve the performance. A similar behaviour was observed where the model was inclined to learn even more if additional epochs were added. As seen in figure 30, not only the training loss was lower than the complete sampling data, the rate of decline of loss was noticeably steeper.

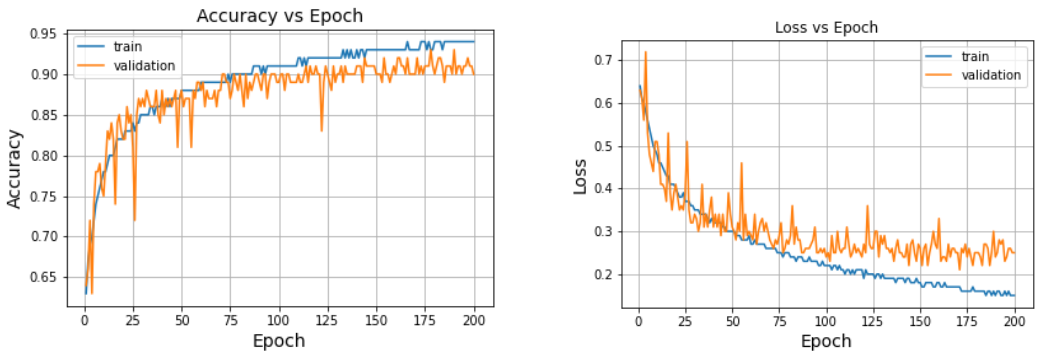


Figure 30. Custom Model on under sampling data

The implication of the two experiments is that the custom-built model performs quite well with high f1-score and that cropping of image is needed for the classifier to get an image in similar vein in the train set to perform better.

ResNet-50:

The biggest challenge faced by ResNet-50 was Overfitting - various experimentation was performed to resolve this issue.

Again, the experiments were performed on the under sampling data to save time and resources. Five experiments were performed as shown in Table 5.

	Architecture	Overfitting mitigation mechanism	Augmented Data Used	(Lowest) Train Loss, F1-Score	(Lowest) Validation Loss, F-1 Score	Behavior
Experiment 1	Last 12 layers unfrozen + 3 Dense layers	Dropout	No	0.0088, 99%	0.2560, 93%	Overfits
Experiment 2	Last 12 layers unfrozen + 3 Dense layers	Gaussian	No	0.0080, 99%	0.3086, 94%	Overfits

Experiment 3	Last 12 layers unfrozen + 3 Dense layers	Gaussian – increased magnitude	Yes	0.0080, 96%	0.7892, 51%	Overfits
Experiment 4	Last 9 layers unfrozen + 3 Dense layers	Dropout – increased magnitude	Yes	0.1815, 93%	0.7069, 49%	Overfits
Experiment 5	All layers frozen + 3 fully connected layers	No Dropout / Gaussian	No	0.044, 98%	0.43, 89%	Overfits

Table 5. ResNet-50 results on experimentation performed on under sampling data

The experiments above use the learnings from the previous experiments to make relevant changes. Even with the addition of noise, overfitting is observed.

The regularisation technique to drop some subsets of nodes in the layers is called Dropout and helps in reducing overfitting and creating a more generalised model. Gaussian dropout uses Gaussian distribution and does not drop the nodes but rather drops or alters their weights during the training time by adding Gaussian noise. As all the nodes are kept intact unlike dropout, nodes in Gaussian dropout are exposed during the testing time and thus lead to lesser computation cost. In Dropout, weights are needed to be scaled due to node dropping in layers. Both types of Dropout achieve the same results in their own manner.

On the complete sampling data, two experiments were done with the addition of class weights:

- **Experiment 1** used dropout with the original dataset as training
- **Experiment 2** had data randomizer added along with Gaussian noisy layers instead of dropouts. Data randomiser is an augmentation of data done randomly as mentioned in 3.1.4 Data Augmentation

The results of these two experiments are shown in figure 31 and figure 32 respectively.

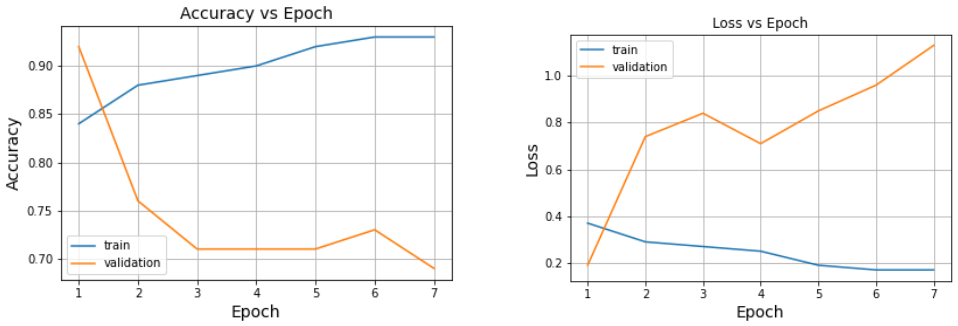


Figure 31. ResNet-50 results on complete sample – without data randomizer

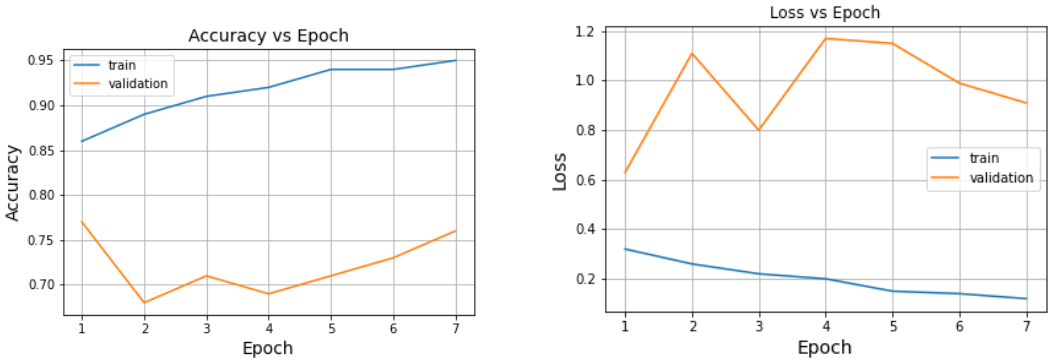


Figure 32. ResNet-50 results on complete sample – with data randomizer

In experiment 1, the loss was 0.19 and the validation f1-score was 92% on the very first epoch and only got worse after each epoch while model accuracy increased to 93% until the seventh epoch. This indicated an overfitting model. To combat this, randomness was added to the data and dropouts were replaced by Gaussian Noisy layers. Again, this did not solve the overfitting validation f1-score decreased to 77% in the first epoch, but training reached 95% in the seventh epoch. Hence, Gaussian noisy layers did not help in the overfitting issues.

As experiments showed overfitting in the model for under sampling data, unfreezing the layers would increase the overfitting problem. Hence, all the layers of ResNet-50 were kept frozen but overfitting still remained - as seen in figure 33 where validation loss only decreased till the third epoch with validation f1-score of and training reaching as high as 97% by the end of seventh epoch.

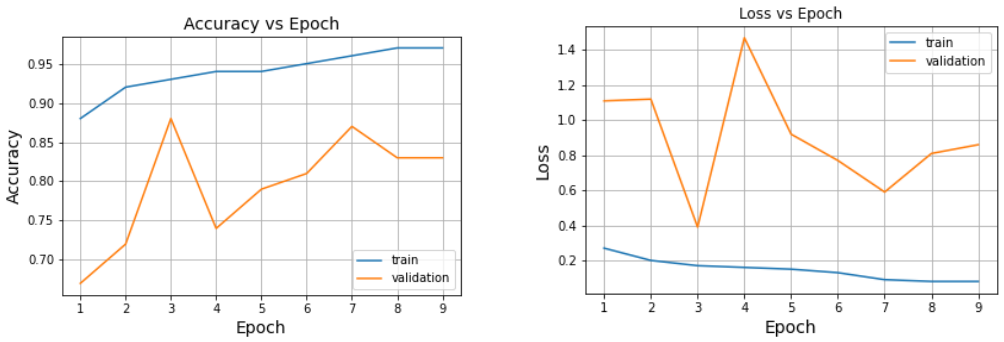


Figure 33. ResNet-50 Training Results on Oversampling Data

With over 92% f1-score on the Proto-test set, transfer learning with ImageNet weights helped the model predict much more challenging images.

Above results show that ResNet-50 is good at predicting unseen images but the accuracy is limited to around 94% at the max. On the other hand, custom models that are trained with no transfer learning mitigates the overfitting issue but can only predict similar images to the train set. Hence, more shallow networks such as VGG-16 or ResNet-32 would be better suited for our use case.

4.1.2 Experimentation on Overfitting Mitigation Strategies

In deep learning literature, adding an outlier class to be trained on the model is an effective technique[24] used for reducing the overfitting. Adding this new class and training on 3 classes instead of the baggage (hard or soft) can help alleviate the overfitting problem. Thus, 2,500 of the 'other' class category were added (images of cardboard boxes etc) to produce a train dataset having 2,500 images for each of three classes.

ResNet50 was trained on three final fully connected layers with dropouts and with the last 12 layers unfrozen. Overfitting issue was only slightly fixed as again validation loss started increasing and the validation f1-score achieved 89% as seen in figure below.

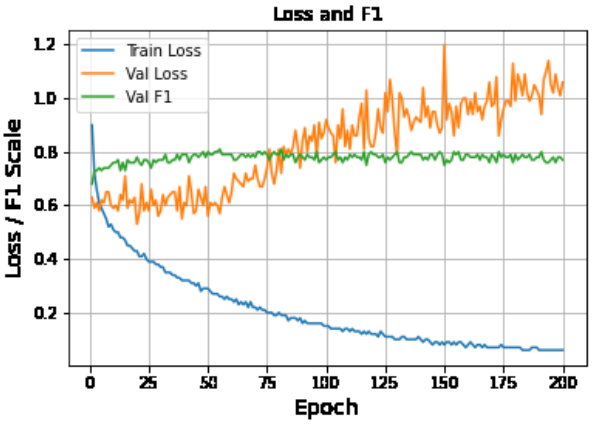


Figure 34. ResNet-50 training on under sampling data with outlier

Since, the cropped test set would not contain objects from the 'other' category, the model chose the second highest class as the label. Please note that in figure 34 that validation accuracy is not the correct indicator of the model performance as it takes into account the prediction of the other category class.

Finally, we use Random Image Cropping Augmentation with other augmentations to try to eliminate overfitting. In Random Image Cropping Augmentation, we create a random section of the image and feed it to the model as training as seen in figure 35.

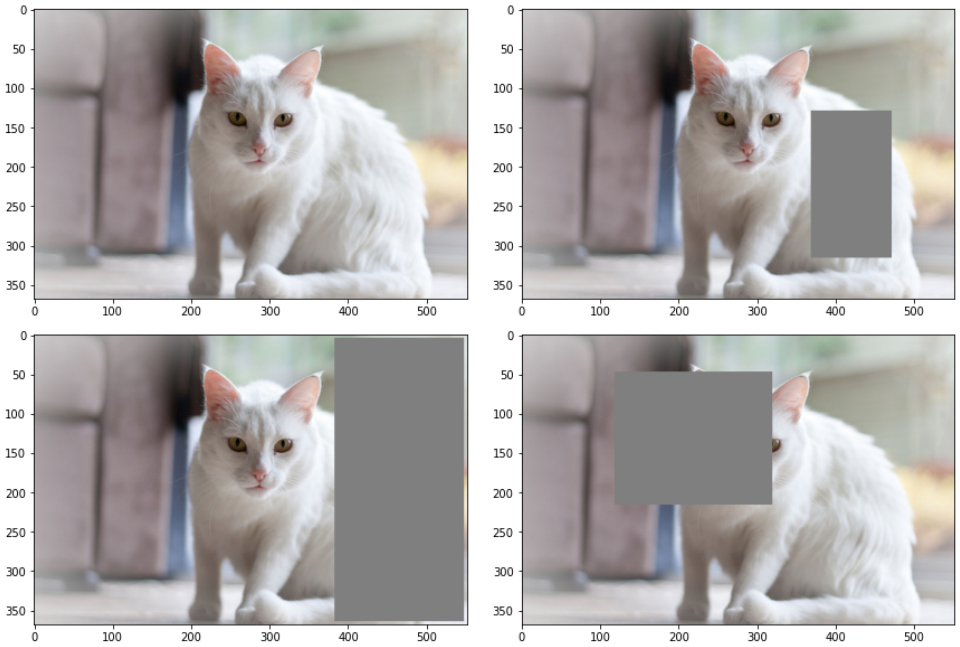


Figure 35. Top Left original image is cropped in three ways and that cropped grey section is fed.

Due to the highly specific images in the MVB dataset, classifiers models have difficulty generalizing resulting in overfitting. This technique enabled the models to learn from the cropped portions which were not given high weightage earlier. This technique allowed us to overcome the overfitting problem finally.

Different versions of resnets were tried with this augmentation and the average overall validation f1-score was 96% without any overfitting.

4.2 Mask R-CNN

For implementation and testing of networks in the Mask R-CNN, Tensorflow GPU 1.15.2 Keras library with Python 3.7 on a Linux based computer system Ubuntu 18.04.5 with 30 GB RAM, Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz, and NVIDIA GeForce GTX 2080 Ti GPU was used.

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 from the MVB dataset called undersampled data. 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 undersampled data, keeping the other hyper-parameter and weights constant.

	Backbone	Epochs	Layers	Loss Weights	Validation Loss
Experiment 1	Resnet-50	100	Heads	1.0	0.8708
Experiment 2	**Resnet-101**				0.3776

Table 6. 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. All layers and stages refer to Resnet Backbone inside 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 4 and Stage 3 are more deeper layers respectively 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.

	Backbone	Epochs	Layers	Loss Weights	Validation Loss
Experiment 1	Resnet-101	100	Heads	1.0	0.3766
Experiment 2			Resnet Stage 4+		0.1744
Experiment 3			Resnet Stage 3+		0.1708
Experiment 4		**40, 80, 40**	**Heads, Stage 4+, Al Layers unfrozen**		0.1162

Table 7. 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 is 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.

	Backbone	Epochs	Layers	Loss Weights	Validation Loss
Experiment 1	Resnet-101	40, 80, 40	Heads, Stage 4+, All	1.0	0.1342

Experiment 2				**Custom weights**	0.1162

Table 8. Validation Loss for different loss weights.

As mentioned above, using the proposed custom weights helps reduce validation loss by around 13%. Hence, these custom weights allow for our model to fit better to the undersampled data.

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.

Backbone	Epochs	Layers	Loss Weights	Validation Loss	F1-Score	mAP@[0.5]
Resnet-101	40, 80, 40	Heads, Stage 4+, All	**Custom weights**	0.1162	0.314	0.017

Table 9. F1-score and mAP@[0.5] form the most successful Mask R-CNN model.

Based on table 9, F1-Score for undersampled data 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 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 36. 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 37), when people are present (see figure 38) and when two or more baggage are present near together (see figure 39).

Figure 37. **Example of a background that mimics baggage material. From left to right, result from COCO weights, training the Heads and training from Resnet Stage 3 onwards.**

As seen in the figure 37, 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.

Figure 38. Example of an image with a person near the baggage. From left to right, result from COCO weights, training the Heads and training from Resnet Stage 3 onwards.

Similarly for figure 38, 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.

Figure 39. Example of two hard baggage overlapping with each other. From left to right, result from training from Resnet Stage 3 onwards and COCO weights

Again, for figure 39, 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 the MVB dataset. As seen in figure 5, MVB 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 MVB dataset is not very 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 37 and figure 38, 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.

4.2.3 Results on Baggage Surface dataset

After creating the new dataset called Baggage Surface-900 dataset, based on the suggestions from Section 4.2.2, we get the following results that justify the creation of the dataset.

Set	F1-Score	mAP@[0.5]
Train	0.789	0.801
Validation	0.692	0.699
Test	0.700	0.720

Table 10. Performance of Mask R-CNN on newly created Baggage Surface-900 dataset

As seen in Table 10, the performance has improved significantly from Table 9, especially in mAP value. This result can be reinforced by the figure below as the mask much accurately covers the baggage.

Figure 40. Mask accurately covers and predicts the baggage as hard from the test set.

As Mask R-CNN receives a mask as input during training, complex geometric transformation of images either needs to be created by annotating each new image which can be time-consuming or use the in-built support of certain geometric transformations using the imgaug library. Based on research done by Google Brain Team, these are top three augmentations to increase mAP score upto 3%: [29]

- **Rotation:** Image rotation. Please note that the bounding boxes get larger relative to the object.
- **Equalize:** Flattens the pixel histogram for the image
- **Bounding Box Movement along Y-axis:** Moves the objects in the bounding box up and down the Y axis (50% odds of up or down)

For the first augmentation, flipping horizontally and flipping vertically is also added.

Set	F1-Score	mAP@[0.5]
Train	0.870	0.891
Validation	0.710	0.723
Test	0.770	0.799

Table 11. Performance of Mask R-CNN on first augmentations on Baggage Surface-900 dataset. This is the best performing model.

As seen in the table above, by at least 2% or more for all F1-score and mAP scores and hence, more augmentation is added to see if performance can be improved.
The following augmentations are added for second augmentations [31]:

- Cutout: Gaussian channel wise noise added to randomly removed sections of image as seen in figure below.

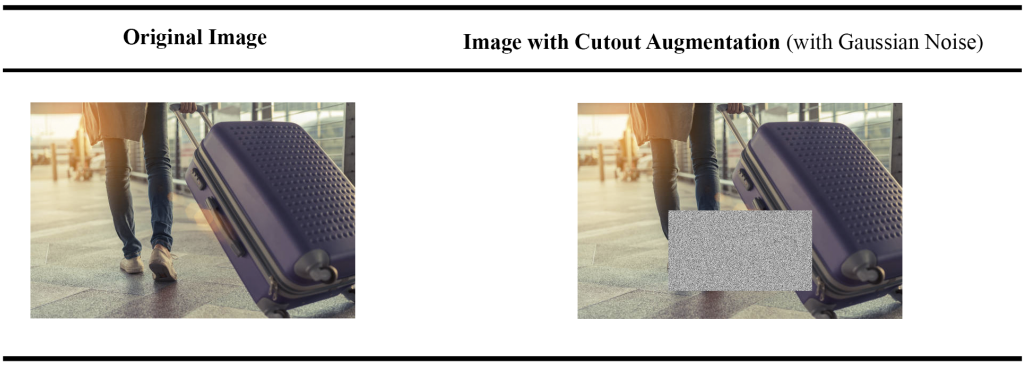


Figure 41. Original image along with cutout augmentation with gaussian noise of the same image

- Motion Blur: Apply motion blur with a kernel size of 15x15 pixels and a blur angle of either -45 or 45 degrees (randomly picked per image)
- Gaussian Noise Injection: Blend the Gaussian Noise (as seen in previous figure) into image.
- Black and White: Using grayscale.

Set	F1-Score	mAP@[0.5]
Train	0.752	0.767
Validation	0.601	0.637
Test	0.633	0.655

Table 12. Performance of Mask R-CNN on second augmentations on Baggage Surface-900 dataset and test set

All scores go down after adding these new augmentations. Further testing revealed that decrease can be attributed to Cutout augmentation. Cutout creates similar results to when Mask R-CNN was trained on the MVB dataset. This is because cutout starts to train models on images with partial sections of baggage and hence, any material which resembles hard or soft baggage structure is masked and classified as well (similar to figure 37). By removing Cutout augmentation, results from the previous table can be achieved again.

Learning rate as a hyper-parameter was also changed with two forms of testing done:

- Increased the Learning rate to 0.02 as mentioned in the Mask R-CNN Report[32]. Matterport Model has a Learning Rate of 0.01 by default.
- Using Leaning Rate of 0.01 for head layers, 0.001 for Resnet 3+ layers and 0.0001 for all layers.

Both of these changes had no effect on the result and actually reinforces that users of the Matterport Mask R-CNN model with the Learning Rate of 0.01 provide the optimal result most of the time. Learning rate of 0.02 may cause weights to explode due to slightly different implementation of Mask R-CNN compared to the original implementation in Caffe[33].

When the Baggage Surface-1700 dataset was created, the environment of the jupyter notebook in the GPU farm where training was performed, was broken due to the upgradation of the farm - more details provided in the difficulties encountered section. After several attempts, it was decided that more effort will be put into improving the performances of the other models while fixing the performance issue after upgradation of the farm would be a lower priority due to lack of time.

One way to potentially combat this issue was to use the best Mask R-CNN model weights as seen in table 11 to train a new model using these weights as the base on the head layers for 100 epochs - results of which can be seen in the table 13.

‘

Set	F1-Score	mAP@[0.5]
Train	0.781	0.800
Validation	0.768	0.765
Test	0.776	0.792

Table 13. Performance of Mask R-CNN on weights from model of table 11 train on Baggage Surface-2000 dataset and test set for 100 epoch

While the performance again decreased with training and due to lack of time, more modifications were abandoned for Mask R-CNN in favour of improving the other model performance. Without the head training, the best weight models results were comparable to table 14 as seen on table below on the Baggage Surface Dataset-2000.

Set	F1-Score	mAP@[0.5]
Train	0.880	0.891
Validation	0.712	0.723
Test	0.771	0.799

Table 14. Performance of Mask R-CNN on weights from model of table 11 on Baggage Surface-2000 dataset and test set

4.3 Yolo Results

SGD optimizer was used as it gave the most generalised results. Initially, Yolo was trained using standard hyperparameters with some layers frozen. Pretrained COCO dataset weights were used similar to Mask R-CNN. It was observed that unfreezing the layers improves performance and hence, all the layers used for training for the rest of the experimentation. All the experiments were carried out on Yolo-v3 and upon obtaining the best hypertuned parameters, these were used on Yolo-v5 later to achieve any better results.

Dataset used Yolo-v3 is Baggage Surface-900 dataset and Baggage Surface-1700 dataset while Yolo-v5 used Baggage Surface-2000 dataset.

Initially results were 74.1% mAP@IoU[0.5] and 63% mAP@IoU[0.5-0.95] on the test set.

4.3.1 Strategy to improve results

The following strategies were applied to improve model performance and are mentioned below.

4.3.1.1 Anchor Box Fitting

Yolo-v3 uses 9 anchor boxes of varying sizes in total and bounds each object that corresponds to the best fitted anchor box. As mentioned earlier in Section 3.2.3.1, three anchor boxes are used in each of the three scaling stages, meaning any object which does not fill in those boxes will have impact on its IoU value and thus, worsening performance. To combat this, K-means algorithm is used to create 9 clusters relative to the sizes of objects present in the training dataset than the standard fixed anchor boxes. The nine clusters formed by K-means algorithm with their sizes used in the final test are shown in figure 42.

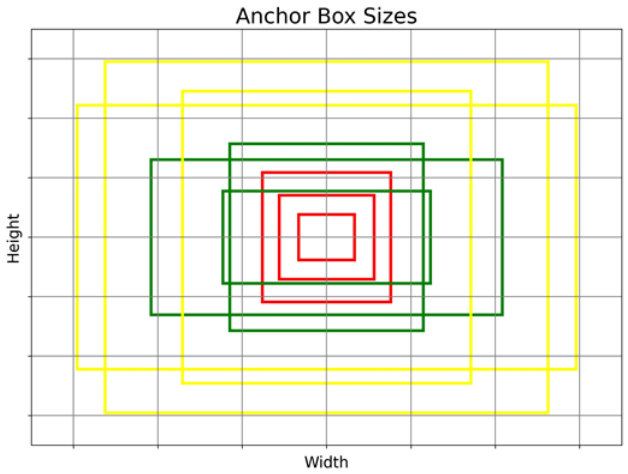


Figure 42. The nine Anchor Box sizes made by K-means algorithm

Keeping all the previous parameters constant, Anchor Box Fitting improved to results around 79% mAP@IoU[0.5] and 67% mAP@IoU[0.5-0.95] on the Baggage Surface-900 dataset.

4.3.1.2 Mixup Augmentation

Mixup is a data augmentation technique that has increased the performance of Yolo drastically. It blends two images to create a new image. This technique helps get better performance with regularization and thus, prevent models from overfitting. The simplicity of this augmentation made the fast and very useful.

4.3.1.3 Additional Data Augmentation

The following augmentations were used:.

- Shear
- Translation
- Horizontal flipping: Vertical flipping was avoided because this would not appear in practice
- Rotation: its magnitude was low due to the same reason of not a real life incident.
- HSV color augmentation: Value (V) defines the brightness of the color, Saturation (S) reflects the depth of purity whereas hue (H) represents chromatic information. Due to its instinctive nature of describing the colors, it distinguishes the colors of images better than RGB. Manipulation of these values needs to be done carefully or it may affect the lighting significantly.

These data augmentation are used to create more data for the model. These augmentation did not improve the model greatly but regularized the Yolo-v3 by converging earlier.

The results of these techniques are mentioned in the table below.

		Experiment-1	Experiment-2	Experiment-3	Experiment-4
Remarks		**Standard Parameters**	**Custom Anchors**	**Mixup Augmentation**	**Additional Data Augmentation**
Validation	**mAP @ IoU[0.5]**	74.1%	79%	80%	80%
	mAP @ IoU[0.5 - 0.95]	63%	67%	70%	70%

Test	**mAP @ IoU[0.5]**	70%	72%	73%	73.3%
Data Augmentation	**translate**	0.1	0.148	0.23	0.245
	scale	0.5	0.472	0.80	0.898
	shear	0.0	0.23	0.50	0.602
	mixup	0.0	0.0	0.213	0.243
	H,S,V	0.015, 0.544, 0.514	0.015, 0.544, 0.514	0.015, 0.544, 0.514	0.0138, 0.664, 0.464

Table 15. Performance of Yolo-V3 on 4 experiments

4.3.1.4 Results from Yolo-v5

Taking the parameters from the experiment that gave the best result, Yolo-v5 network was used. This was done because Yolo-v5 is very similar in architecture to Yolo-v3[26]. Two experiments were performed with different proportions of the dataset as summarized in table 16 below. Yolo-v5 increased mAP values significantly and hence, it was used in production during the mobile app deployment stage for the Yolo tab. Increased performance could be attributed primarily to the increased volume of the dataset as the images in the new dataset were almost double the number of images that Yolo-v3 was trained on previously as the slight modification of architecture in Yolo-v5.

		Experiment-1	**Experiment-2**
Dataset		**Baggage Surface 1700**	**Baggage Surface 2000**
Validation	**mAP @ IoU[0.5]**	88.3%	88%
Test	**mAP @ IoU[0.5]**	87.4%	87.2%

Table 16. Performance of Yolo-V5 with best Yolo-V3 parameters on 2 experiments of different Baggage surface dataset

4.4 Comparison of Best Models

The following best performing models for each type on the test set are mentioned below. Please keep in mind that Classifiers were tested on the altered version of the test set called the cropped test set as they cannot do not object detection.

Model type	Model/s	Modifications from Default model	F1-Score	Avg Detection Time (CPU)	mAP@[0.5]
Classifier	ResNet-18,ResNet-34,ResNet-50,ResNet-101, ResNet-152 - named as Custom Resnet	Augmentation of Random Resized Cropping, (plus, mention all the augmentation from the interim report augmentation section), Unfreezing train layers, cost effective weighing for imbalance class dataset	96.6%	0.18 seconds	-
Mask R-CNN	-	Augmentations of Rotation, Equalize and Bounding Box Movement along Y-axis, Motion Blur and Black and White	-	10.37 seconds	79.9%
Yolo	Yolo-v5	Custom Anchor Box Sizes fitting the dataset, augmentation of mixup, HSV, Random Image Cropping. Unfreezing train layers	-	1.97 seconds	87.2%

Table 17. Best Performing Models from Each Type of Model

Using the table above, it can be observed that Yolo-v5 outperforms Mask R-CNN by 7% and even is 5 times faster at detection due to Mask R-CNN being a computational heavy model. Hence, Yolo-v5 is the recommended model to use in the mobile application if HKAA does not want passengers to crop images and just get results when submitting the image to the cloud. This would make for a much easier and more convenient to use application.

However, it cannot be denied that 96.6% F1-score is an impressive result for the Classifier model in table 15, which means that 9/10 times, the baggage in the image would be correctly identified by manually cropping the image first. By cropping, the user is making the

model focus on one baggage and with being just as fast, this combined model makes an equally compelling case for its usage. It also has 82% F1-Score on uncropped test set images.

Due to high F1-score, faster average detection time to Yolo-v5 and cropping feature reducing the chances of submitting image with too much irrelevant background objects and hence, incorrect detection causing damage to the baggage as well as inconvenience to the customer, this project recommends the usage of combined classifier model for the smart phone application part that HKAA will integrate into their HKG MyFlight App.

Nonetheless, our application provides the HKAA staff to use and check all the models as explained in Section 3.4 to make their own decision on which model in table 15 to use as the Baggage surface identifier for the Self-service Bag drop system.

4.5 Difficulties Encountered

Several difficulties were encountered while 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. Similarly, Yolo required a specific formatting of images and their box coordinates. This transformation from JSON to VIA format was a time consuming task. Not only was the transformation time consuming, during the reformatting stages for the networks, some internal library glitches led to several hours of training wasted. One such annotator's internal library issue during reformatting was to not normalize the values appropriately and the model was not made to handle this exceptional edge case when the dataset consisted of labels going beyond the boundary of the image. The implementation was thoroughly checked and no more errors were encountered later.

The Baggage Surface Dataset was created using copyright free images, which meant the pool of images that could be added to the dataset was limited and was a herculean task to expand this dataset to 2,200 images. It was tough to find soft baggage compared to hard baggage online and hence, was a challenging task to keep the number of object classes at a similar number. Most easy to secure images online had fashion models in images with perfect lighting (see figure 43) which is not realistic when taking images for the application. Hence, more time and effort were put in digging through images to find these typical images.

Figure 43. A fashion model poses with baggage on a set. An unrealistic situation for the models to encounter.

The Mask R-CNN model was functioning properly until HKU GPU Farm 2 was updated without prior notice in March, 2021. Training the model in the previously used virtual environment was not possible as GPU was not being identified and meant 2 hour per epoch.

One way to fix this unreasonable training time was to open the training iPython notebook inside the virtual environment before starting the training. However, this resulted in the performance across all metrics to drop by at least 10% as seen in the table below even with all other parameters as well as the dataset remaining the same.

Set	F1-Score	mAP@[0.5]
Train	0.672	0.673
Validation	0.555	0.576
Test	0.567	0.585

Table 18. Mask R-CNN performance on Baggage Surface-900 Dataset after HKU GPU farm upgradation

The GPU training time was fixed when it was realised that CUDA was updated to 11.2, which is incompatible with the tensorflow version of 11.15.0, the requirement for Matterport Mask R-CNN model. Luckily, CUDA 10.0 was still provided for legacy software and hence, the GPU training time was fixed. Nevertheless, training in the current

environment still resulted in worse performance than before this issue and changing weight parameters or any augmentation made the performance of the model worse.

The issue still persisted even with clean installation of Matterport Mask R-CNN libraries in a newly created virtual environment. Due to time constraints, it was decided to put more effort in improving the application as well as Yolo performance by adding the crop feature and increasing the dataset respectively.

Some of the difficulties were associated with 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. In hindsight, it was disheartening to see this issue ruin several hours of our training process. Likewise, there were some images whose mask coordinates were not present in the metadata JSON file. Such exceptional 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 challenge faced 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.

Mask R-CNN's code was created to work only on one class (excluding background class) so it needed to be extended to handle more classes. However, a limitation that was also reported to the IT team of HKU GPU farm lart, was that none of the available GPUs in the user's quota must be in use before opening the jupyter notebook. The rest of the GPUs must be occupied after opening the jupyter lab and the jupyter lab would not open if the user has already occupied one or more GPUs for some other tasks. There were instances when the jupyter notebook could not be opened for processing because other models were being trained in parallel. 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

Mask R-CNN's testing could not be completed due to breakdown of HKU GPU farm 2 environment owing to the abrupt upgrade. Implementing the TensorFlow object detector implementation of Mask R-CNN, which is the second most popular implementation of it may fix the library issue currently faced by Matterport Mask R-CNN as TensorFlow Object detector Mask R-CNN support TensorFlow Version 2. As measured in Section 4.4, Mask R-CNN is slow during detection and its masks are not refined enough for large objects with complex shapes and especially suffers from stair effects.

Released in October of 2020 [27], SOLOv2 is not only at least a 20% faster model but also performs better in instance segmentation. The implementation of SOLOv2 was done in February of 2021[28] and can be used to replace or compare Mask R-CNN for this project.

As mentioned before, Yolo-v5 is a community effort (not made by the original author of Yolo-v3) and suffers from glitches. There are also arguments that Yolo-v4 (also a community effort) performs better than Yolo-v5, however, there is not enough substantial

proof to back up this claim. Hence, comparison of these two community models' performance can be done to see which model outperforms the other on this project.

Baggage Surface Dataset as well as MVB dataset did not take into account handbag or backpack as a soft surface material as well as small suitcase as hard surface material and was out of the scope for this project. However, in some rare cases, these can be checked in by the passenger and hence, can be a good extension of the current datasets. It is also important to note that any hard baggage with a luggage cover would be misclassified as having a soft surface - one way to make model predictions may be to look at the corners of the baggage as they may not be covered by the luggage cover sometimes. A similar situation is also observed for baggage that is covered in tape for protection. Clearly more research is needed on how this issue can be tackled without asking the passenger to remove the cover.

Figure 44. From left to right: Backpack, small office suitcase and baggage covered with cloth cover (the hard surface can be seen from corners but not true for all covers)

One of the challenging aspects of the Baggage Surface Dataset was collecting more training images. This can be achieved by paying for premium accounts in stock photo websites but also first hand by collecting images on the airport by taking consent of the passengers to take photos of them with their baggage. It is also important to keep in mind that most people would at least use this app first time in the airport by either being directed to download by the smart kiosk and thus, better results can be achieved for our model by collecting the images in the HKIA environment. Although it was always the plan to visit the Airport, the COVID-19 pandemic as well as the busy schedule of HKAA meant we were not able to do so.

5.1 Project Schedule

Date	Deliverables	Status

September 2020	Preparations and Deliverables of Phase 1: ● Detailed Project Plan ● Project Web Page Review: ● All details provided by HKAA on the self-service bag drop system. Research: ● Self-service bag drop system implemented in the airport ● Existing algorithm to achieve our objectives.	**Completed**
October 2020	Design: ● Basic System architecture of the solution Research: ● Implementation of Yolo versions, ResNet50, and Mask R-CNN ● Camera for live feed Implement: ● Implementation of Mask R-CNN	**Completed**
November 2020	Research: ● Implementation of Yolo-v3, ResNet50, VGG-16, Custom Model ● Environmental conditions in the airport where the model will perform Implement: ● Design Custom Model ● Train Yolo-v3, ResNet50, VGG-16, Custom Model ● MVB Data preparation and transformation (Augmentation)	**Completed**
December 2020	Preparations of Phase 2: ● Interim report ● First presentation ● Demo in the first presentation Implement: ● Target the classes sample size imbalance problem ● Evaluate the performance of ResNet50,VGG-16, Custom Model and Mask R-CNN ● Test the top two best performing models in a simulated environment of HKIA.	**Completed**
January 2020	Deliverables of Phase 2: ● Interim report ● First presentation Implement: ● Fine tune the machine learning models ● Trying variants of ResNets ● Mitigate the overfitting issues in ResNets ● Trying variants of mask R-CNN classifiers ● Creating Baggage Surface Dataset-900 and Test set	**Completed**

February 2020	Research: • Camera Effect on Deep Neural Networks • Regularization and hypertuning techniques for Yolo • Research on Backend and Frontend architecture of mobile app Implement: • Fine tune the mask R-CNN • Suitable dataset collection for Yolo-v3 and mask R-CNN training • Implementation of Yolo-v3 and Yolo-v5 • Test the best performing model in HKIA environment	**Completed**
March 2020	Preparations of Phase 3: • Final report • Final presentation • Demo(using mobile app) in the final presentation Implement: • Complete the frontend using React Native • Complete the backend implementation using AWS services • Fine Tuning Yolo-v3 • Create Baggage Surface Dataset-1700 Testing: • Testing Yolo-v3 and Yolo-v5 (same parameters as Yolo-v3) • Choosing the best model version for Yolo and Mask-RCNN • Test and debug the implemented mobile application	**Completed**
April 2020	Implement: • Create Baggage Surface Dataset-2000 • Test all the models on this final Baggage Surface Dataset Deliverables of Phase 3: • Final report • Final presentation • Demo in the final presentation	**Completed**

6. Conclusion

This project aims to create a robust machine learning with fixed camera setup to identify baggage material as hard or soft. Three model types - Classifiers, Mask R-CNN and Yolo were trained on the primary dataset of Baggage Surface which was hand annotated and the secondary dataset called MVB. A test set was created to compare performances which showed that Custom ResNet50 is the fastest model that performs the best with 96.6% F1-score however, it requires cropped images from the test set to achieve this. Yolo-V5 performs the best on the test set with 88% mAP while Mask R-CNN lags behind with 80% mAP due to technical difficulties. A smartphone application containing all the best performing model types was created for HKAA staff to test and make the final decision on

which model they would want to use. This project has the potential to improve the existing architecture by using new state of the art models such as SOLOv2 and address more challenging use cases such as covered luggage. With the smartphone app being integrated into the HKG, MyFlight App once finalised by HKAA, would be one step closer to make HKIA airport of the future.

7. References

[1] “Facts and Figures, HKIA at a Glance.” Hong Kong International Airport, www.hongkongairport.com/en/the-airport/hkia-at-a-glance/fact-figures.page.

[2] “Self Bag Drop Service, Airport Facilities & Services.” Hong Kong International Airport, www.hongkongairport.com/en/passenger-guide/airport-facilities-services/self-bag-drop-service.

[3] “New Self-Service Bag-Drop System Will Cut Check-in Times at Hong Kong International Airport by a Third.” South China Morning Post, 16 Sept. 2015, www.scmp.com/news/hong-kong/economy/article/1858602/new-self-service-bag-drop-system-will-cut-check-times-hong.

[4] Hkairportofficial, A Smart Airport Experience - Smart Check-in Kiosk + Self-Bag Drop Service. Youtube, 6 Nov. 2019, www.youtube.com/watch?v=-a0JYmmSo9A.

[5] Martin ThomaMartin Thoma 2, et al. “What Is the Difference between Object Detection, Semantic Segmentation and Localization?” Computer Science Stack Exchange, 1 June 2016, cs.stackexchange.com/questions/51387/what-is-the-

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

[6] “Better Baggage Handling with SITA.” Filament AI, 24 Aug. 2020, www.filament.ai/2019/04/01/sita-baggage-classifier-data-study/.

[7] Zhang, Zhulin, et al. “MVB: A Large-Scale Dataset for Baggage Re-Identification and Merged Siamese Networks.” ArXiv.org, 26 July 2019, arxiv.org/abs/1907.11366.

[8] VolumeNet. 同方威视箱包再识别技术挑战赛, http://volumenet.cn/#/.

[9] “Common Objects in Context.” COCO, cocodataset.org/.

[10] Redmon, Joseph, and Ali Farhadi. “YOLO9000: Better, Faster, Stronger.” https://Arxiv.org/, 25 Dec. 2015, arxiv.org/pdf/1612.08242.pdf.

[11] Santad, Tossaporn, et al. Application of YOLO Deep Learning Model for Real Time Abandoned Baggage Detection - IEEE Conference Publication. IEEE, 9 Oct. 2018, ieeexplore.ieee.org/document/8574819.

[12] Tzutalin. “LabelImg.” GitHub, github.com/tzutalin/labelImg.

[13] Keras. “ResNet-50.” Kaggle, 12 Dec. 2017, www.kaggle.com/keras/resnet50.

[14] Tharwat, Alaa. (2018). Classification assessment methods. Applied Computing and Informatics. https://doi.org/10.1016/j.aci.2018.08.003

[15] SowmiyaNarayanan, G. “Image Segmentation Using Mask R-CNN.” Medium, Towards Data Science, 12 July 2020, towardsdatascience.com/image-segmentation-using-mask-r-cnn-8067560ed773.

[16] Sarah O'Gara, and Kevin McGuinness. “Comparing Data Augmentation Strategies for Deep Image Classification.” 2019, https://arrow.tudublin.ie/cgi/viewcontent.cgi?article=1003&context=impstwo.

[17] Qi Dong, et al. “Imbalanced Deep Learning by Minority Class Incremental Rectification.” *arxiv.org*, https://arxiv.org/pdf/1804.10851.pdf.

[18] Yunru Liu, et al. “SelectNet: Learning to Sample from the Wild for Imbalanced Data Training.” *arxiv.org*, https://arxiv.org/pdf/1905.09872.pdf.

[19] Matjaz Kukar, and Igor Kononenko. “Cost-Sensitive Learning with Neural Networks.” *citeseerx.ist.psu.edu*, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.8285&rep=rep1&type=pdf.

[20] Jost Tobias Springenberg, et al. “STRIVING FOR SIMPLICITY: THE ALL CONVOLUTIONAL NET.” *arxiv.org*, https://arxiv.org/pdf/1412.6806.pdf.

[21] Bobba, Ravikiran. “Taming the Hyper-Parameters of Mask RCNN.” Medium, Analytics Vidhya, 18 Dec. 2019, medium.com/analytics-vidhya/taming-the-hyper-parameters-of-mask-rcnn-3742cb3f0e1b.

[22] Abdulla, Waleed. “Splash of Color: Instance Segmentation with Mask R-CNN and TensorFlow.” Medium, Matterport Engineering Techblog, 10 Dec. 2018, engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46.

[23] “Common Objects in Context.” *COCO*, cocodataset.org/#explore.

[24] Navid Ghassemi, and Hadi Mahami. “Material Recognition for Automated Progress Monitoring using Deep Learning Methods.” *arxiv.org*, https://arxiv.org/pdf/2006.16344.pdf.

[25] Gondhalekar, Amey. “Data Augmentation- Is It Really Necessary?” *Medium*, Analytics Vidhya, 24 Mar. 2020, medium.com/analytics-vidhya/data-augmentation-is-it-really-necessary-b3cb12ab3c3f#:~:text=Data%20augmen

tation%20is%20a%20technique,data%20from%20existing%20training%20data.&text=It%20helps%20us%20to%20increase,images%20as%20distinct%20images%20anyway.

[26] Solawetz, Jacob. “YOLOv5 New Version - Improvements And Evaluation.” *Roboflow Blog*, Roboflow Blog, 4 Mar. 2021, blog.roboflow.com/yolov5-improvements-and-evaluation/.

[27] SOLOv2: Dynamic and Fast Instance Segmentation, Xinlong Wang, Rufeng Zhang, Tao Kong, Lei Li, Chunhua Shen In: Proc. Advances in Neural Information Processing Systems (NeurIPS), 2020 arXiv preprint (arXiv 2003.10152)

[28] WXinlong. “WXinlong/SOLO.” *GitHub*, github.com/WXinlong/SOLO.

[29] Wright, Less. “State of the Art Object Detection - Use These Top 3 Data Augmentations and Google Brain's Optimal...” *Medium*, Medium, 27 June 2019, lessw.medium.com/state-of-the-art-object-detection-use-these-top-3-data-augmentations-and-google-brains-optimal-57ac6d8d1de5#:~:text=The%20top%203%20augmentations%20used,bounding%20box%20size%20must%20increase.

[30] “TorchServe¶.” *TorchServe - PyTorch/Serve Master Documentation*, pytorch.org/serve/.

[31] Shorten, Connor, and Taghi M. Khoshgoftaar. “A Survey on Image Data Augmentation for Deep Learning.” *Journal of Big Data*, Springer International Publishing, 6 July 2019, journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0#Sec3.

[32] arXiv:1703.06870 [cs.CV].

[33] Matterport. “Learning Rate Decay · Issue #289 · Matterport/Mask_RCNN.” *GitHub*, github.com/matterport/Mask_RCNN/issues/289.