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

Student
Muhammad Sheheryar Naveed
UID: 3035493672

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

Abstract

Increased activity at Hong Kong airport has intensified the use of self-service bag drop systems and must be automated, completely eradicating the involvement of airport staff in the procedure. Taking inspiration from this initiative, the project aims at implementing a deep learning material detection model (hard or soft) to both dehumanize the self-service bag drop procedure as well as to avoid the abrasions on check-in bags(soft bags added in a tray for distinguishing) in-flight thus improving customer experience and quality assurance. The exploration phase consists of three types of neural networks - Classifiers, Yolo(v3 & v5) and Mask R-CNN. VGG-16, ResNets and a custom built network classifier were tested. ResNets performed much better but suffered overfitting which was alleviated by using specialized data augmentation techniques like random image cropping. Twenty percent of secondary source dataset suitable for classifiers consisting of highly zoomed in-cropped images, was added to a hand-annotated web-based self collected dataset suitable for training Yolo and Mask R-CNN networks. With experimentations performed in accordance with the comprehensive view of discipline; tuning hyper-parameters and using specialized techniques to fit the model to the data, Yolo and Mask R-CNN both produces visually pleasing results with Yolov5 performing better on test images than Mask R-CNN whose train accuracy is better however. Mask R-CNN is theoretically better in terms of performance than Yolo but has a significantly lesser inference time(10.37s for Mask R-CNN and 1.97s for Yolo-v5 on CPU based device) as compared to Yolo which, unlike Mask R-CNN, is a single stage model. Classifiers are the fastest, producing response in 0.18s with 96% F1 score on cropped images. Cross platform(IOS & Android) smartphone application, with features including but not limited to response time, editing image for classifiers, has been developed and the best performing versions of the three networks deployed using AWS EC2 service, TorchServe and flask, for better visualization and testing of the three types of networks by the stakeholders and easier integration into the existing HKIA mobile app.

Acknowledgements

I would like to offer a token of appreciation to my supervisor Dr TW Chim for his thorough guidance throughout the project, which immensely contributed to the completion of this paper. I would also like to express my sincerest gratitude to Cezar Cazan from the Center of Applied English Studies for providing me insightful recommendations which proved extremely useful while drafting this paper. Both of their willingness to give their time so generously has been very much appreciated. Moreover, credits to my fellow batchmate Utsav Raj for his participation in the work outlined in this paper.

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)**
NMS	**Non Max Suppression**
mAP	**Mean Average Precision**

List of Figures

List of Tables

1. Introduction

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

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

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

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

1. Scan their boarding pass barcode and get a luggage tag to place it on their bags[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]

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

The objective of this project is to plan, design, develop and implement a state-of-the art machine learning model, accompanied by deployment on a mobile app for the purpose of visualization, in order to identify a bag as soft (made up of either leather or woven nylon – cordura, ballistic, or ripstop) or hard (made up of aluminium or plastics – ABS or polycarbonate). The screen would then prompt the passenger to use a tray in case the bag is classified as soft. This would allow them to follow the right practice to comply with the standard procedures of HKIA while also avoiding any potential damage to soft bags, in turn improving customer experience. Besides protection against scraping as shown in figure 2, this seamless model would also offer reduced human interaction between the passengers and the airport staff which is much demanded with the Covid-19 in place. The following models will be used, and their performances compared - Mask R-CNN and Yolo-v5 & Yolo-v3, Classifiers(ResNets, VGG and Custom-built). This project is not just limited to the baggage detection and material identification but needs to propose an overall working prototype to HKIA and hence, requires the understanding of not only the algorithms but also the hardware components such as camera configuration and device processing. This paper provides a basis for sound understanding regarding the well-suited techniques for performing such computer vision tasks and their applicability at a preliminary stage.

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

The remaining paper is formulated as follows. This section continues with a subsection on the research background of key terms identifying capabilities of each model and the terminologies used for the comparison of their performance. Section 2 discusses the existing research done on the baggage material in the literature followed by the methodology adopted for this project explaining the data sets collected, the camera factors that could impact performance and finally the three types of networks tested as the scope of this project - Classifiers(ResNets, VGG-16 & Custom Built), Yolo(v3 & v5) and Mask R-CNN, including the algorithm designs under these as well as their comparison from a practical standpoint. Section 4 details the experiments performed to evaluate the performance of three types of networks and reasons the results and findings as such, while also considering the difficulties encountered, and finally concludes by discussing the future work that should be carried out as next steps, in Section 5.

1.1 Research Background

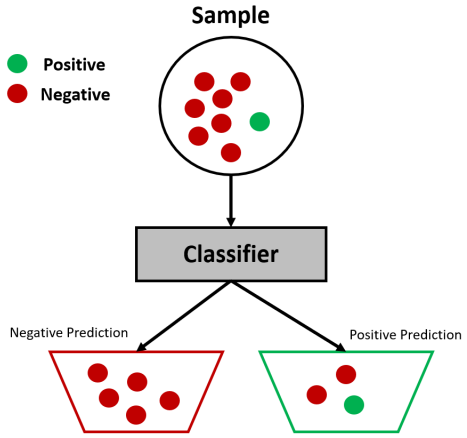
Object recognition is a computer vision technique to recognize varying classes of objects in an image/video[5]. Object detection provides more detailed information by identifying an object in an image and specifying its location using a bounding box[5] which is often shown in the form of a minimum bounding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

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

Apart from the general metrics used to compare the performance and preferences for DNN, certain metrics are used to evaluate the models in terms of their accuracy. While accuracy is a general term that could help in simple scenarios, it is not a robust metric to compare a model's performance against other models. For evaluating stability and consistency of models and comparing these with one another better, there are various

evaluation metrics available, but the popular ones include F1 score, AUC(Area Under ROC - precision-recall) curve and mAP (mean Average Precision).

F1 score is a good means of comparison of predictive abilities of different models as it counteracts the imbalance between the target classes by giving equal weightage to majority and minority class[14]. It takes into account precision and recall. Precision is the measure of how precise a model is in terms of prediction. It is a measure of the total true positives (eg. let's say dogs) out of all the objects model predicted of that class (i.e. total predictions model made as dog regardless of these were dogs(true positive) or not(false positives)). Recall on the other hand measures the total true positives out of the actual true positives(i.e. true positives which were labelled by the model as dogs and other dog objects which might have been labelled as other class but were indeed dogs). Precision and recall calculation in terms of object detection could be visualized in figure 4. It is observed that accuracy could be largely contributed by true negatives and if false positives and false negatives have some intrinsic value in the use case, then F1 is a better parameter for comparison. As mentioned earlier, it also contributes towards uneven class distribution.



		Real	
		Positive	Negative
Predicted	Positive	1(tp)	2(fp)
	Negative	0(fn)	5(tn)

$$\textbf{Accuracy} = \frac{tp+tn}{tp+fp+tn+fn} = \frac{6}{8} = 0.75$$

$$\textbf{Precision} = \frac{tp}{tp+fp} = \frac{1}{3} = 0.333$$

$$\textbf{Recall} = \frac{tp}{tp+fn} = \frac{1}{1} = 1$$

$$\textbf{F-1 Score} = \frac{2*(precision*recall)}{precision+recall} = 0.50$$

Figure 4. Accuracy in DNN: Precision, Recall, F-1

Another commonly used term for comparing the performance of detection models is called inference time[25]. Inference time could be associated with the processing time of the model when deployed in production or when processing the live feed of images. This is essential owing to the rate at which the model might need to detect. Some models have lower inference time but lower accuracy as well. Certain models depending on the complexity of their network are compared on this metric.

Intersection over Union(IoU) is commonly used in image segmentation and detection models. This is because when determining the position of an object, as opposed to just the prediction, the difference in the ground truth position and the prediction position needs to be accounted for. IoU parameter is used to find out how close the model predicted the bounding box/mask in comparison to the ground truth. For example, IOU for threshold 0.5 means that if predicted bounding box/mask and actual bounding box/mask overlap by 50% or more, it is considered a true positive. Figure 5 illustrates the calculation below.

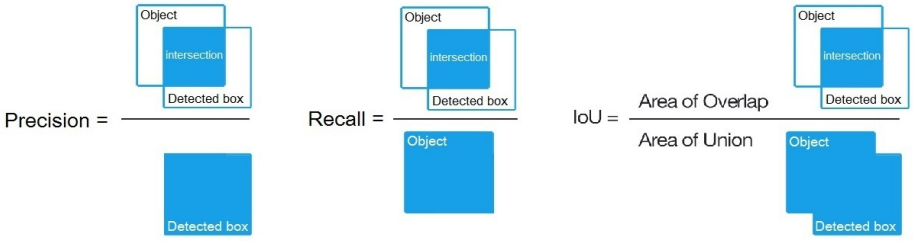


Figure 5. Visualization of Precision, Recall and IoU calculation[26]

mAP is derived from the AP(Average Precision) parameter, which is used to evaluate the performance of the model at various confidence levels whereas F1 score is measured from precision and recall at a fixed confidence interval. For local positioning detection models, confidence intervals play an important role, therefore AP is used as an evaluator. In general, AP is the measure of the area under precision recall curve (also called area under AUC). Mean AP of all the classes being predicted is called mAP. Figure 6 illustrates how confidence scores could impact recall and precision and then mAP. Precision is nearly 1 at a high confidence threshold because the model detects objects very carefully but the predictions it makes are quite precise whereas recall is quite low because it fails to detect objects that were indeed present in the image probably because their confidence score was slightly lower than the set threshold.

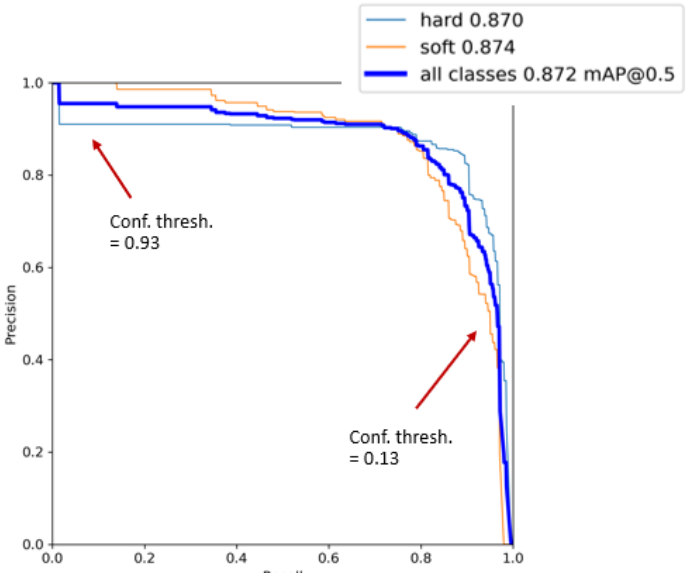


Figure 6. Precision Recall Curve and mAP for hard and soft bags

2. Related Work

Although a lot of research in the field of Computer Vision is being directed towards the recognition of material properties of an image, detection of the surface material of baggage has been sparsely focused. Two studies were carried out that are associated with the work outlined in this paper.

The first study is conducted by an applied AI business[6] called Filament to find mishandled bags using an automated database[6]. Every mishandled bag's images are automatically fed to a deep learning model which extracts the unique visual properties of the bag(i.e. bag type, colors, patterns, wheels, etc.) into an unique identifier code. When a passenger provides the characteristics of the missing bag, the database is used to find the bag matching the characteristics in the given description. By using the basis of a RetinaNet, originally trained on COCO[9] dataset, the study shows that 90% accuracy was achieved in predicting the bag type including hard and soft surfaces[6]. Similarly, pretrained models, which were originally trained on the COCO dataset that this study used, were also used in our project with the exception that Yolo-v3,v5 and Mask R-CNN were used instead of RetinaNet.

The last study focuses on re-identification of the same bags that pass through the different checkpoints on the baggage handling system in an airport using the Merged Siamese networks. The Siamese network was used to match the images of the same bag identities taken at different angles. Similar to the first study, the network was slightly modified to take

into effect the hard and soft bag material surface[7]. Likewise, our study imitated the same approach by using an existing network and changing the parameters and architecture to fit our use case.

3. Methodology

This section outlines the data collection and modification process in order to train the baggage surface material identification, the three different networks and their variants tested, and the procedure adopted for the development of Smartphone application. The block diagram in figure 7 illustrates the high-level flow of the methodology(apart from Smartphone app) undertaken for this project. The block diagram consists of the initial data preparation stage involving two types of datasets discussed in section 3.1 followed by the testing of networks for which the methodology adopted is discussed in section 3.3. Details of the smartphone application are discussed in subsection 3.4. Methodology section is concluded by discussion on the comparison between Yolo and Mask R-CNN based on the design, implementation and practicability of these algorithms in subsection 3.5.

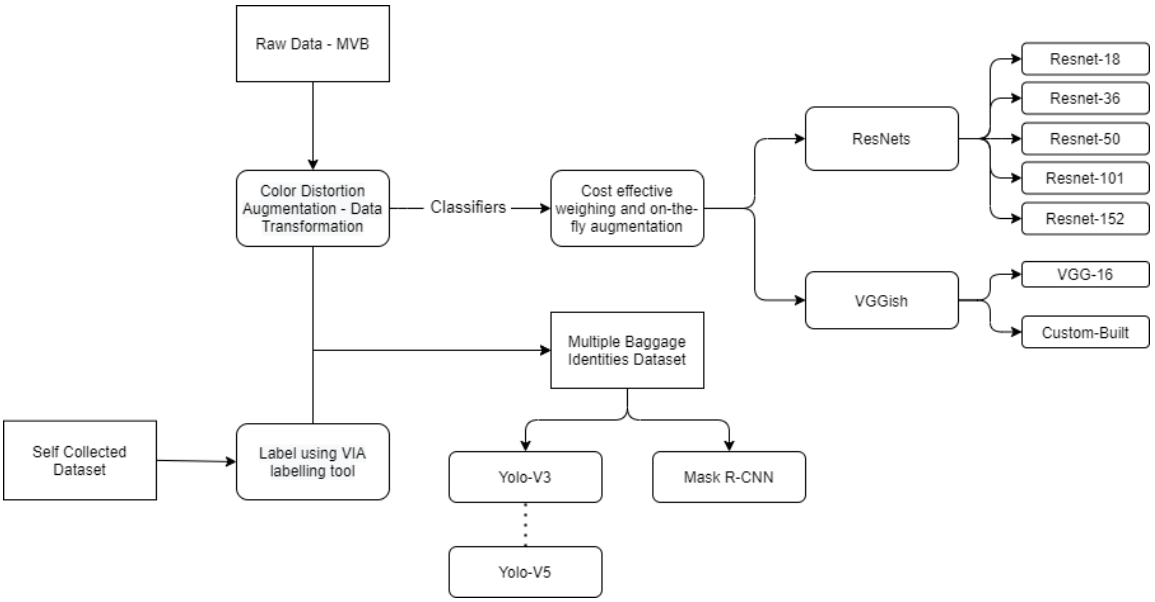


Figure 7. Block Diagram of proposed Networks Testing

3.1. Data Collection

Ideally, we want many baggage images taken at different angles with multiple images per bag identity to achieve the best results. Owing to privacy concerns that HKIA must comply with, and lack of presence of an inbuilt mechanism to capture such images, data of

HKIA passengers' baggage is not available. Thus, two different types of data were collected as a part of this study, described below in subsection 3.1.1 and 3.1.2.

3.1.1. Multi View Baggage (MVB) Dataset

This is a secondary source data that was collected for another study readily available for use. This dataset consists of 4,500 different baggage identities with around 22,000 annotated baggage images taken at different angles with multi view cameras[8]. All the information about the dataset was stored in a JSON file with the file name of each image (primary key – unique identifier for each image), material label (hard, soft, or others) and polygon (x and y coordinates for each point in the mask). With this dataset, we have a firm base to create an altered dataset making it in sync with the environment of HKIA for our deep learning models.

Figure 8. Visualization of the secondary dataset. It has 12,000 hard baggage images and only around 5,000 soft baggage images. Multiple views of the same baggage is present in the dataset as well.

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

In order to make the data mimic the real environment of HKIA, it needs to be augmented by applying various filters on it. Also, the model may have a lower propensity to recognize certain materials that change color at varying lighting. To counteract this problem,

color distortion augmentation was done by changing the parameters such as randomizing the brightness and changing the contrast and the saturation. Further augmentation involving flipping and rotating of the image as well as color filter was performed in order to make the model focus on the surface of the baggage and ignore the background. This transformation step aims at not only depicting the true environment but also to multiplicate the training data with higher complexity. More about this data augmentation step is discussed in the section 3.3.1.3.2.

A few characteristics of this dataset includes single baggage identity per image and highly enlarged bag object with minimal background. Such features make this dataset suitable for only classification tasks and not the object detection, which requires multiple objects in an image and detecting each of those objects and classifying them as an appropriate class.

3.1.2. Multiple Baggage Identities Dataset

While the MVB dataset in section 3.1.1 would work for classifiers, it is not suitable for detection and segmentation purposes. This hand-annotated dataset was self-collected primarily to counteract the limitations of MVB dataset. VGG Image annotator (version 2.0.11) was used for annotation not only because of its easy installation and setup but also because it was used by authors of MVB dataset, which helped preserve consistency when using these two datasets together simultaneously. Any images taken from MVB and added to this new dataset were re-annotated.

Images were collected from publicly available sources including but not limited to Imagenet, ADE20K, Freepix, Pixabay, eBay, Flickr and Pinterest. Instead of integrating COCO dataset images, we rather used its weights in the pretrained model on which these new datasets were trained. Apart from the self-collected images, this dataset was composed of 20% of MVB dataset randomly chosen in order to overcome the ineptitude of COCO trained models to handle closeup images of diverse backgrounds such as Room, Beaches and Airport and images containing both classes of objects together as seen in figure 9 below. It was refrained from adding more than 20% MVB dataset because this dataset consists of bigger bounding boxes and segmentation masks leading to the similar issue as seen with MVB.

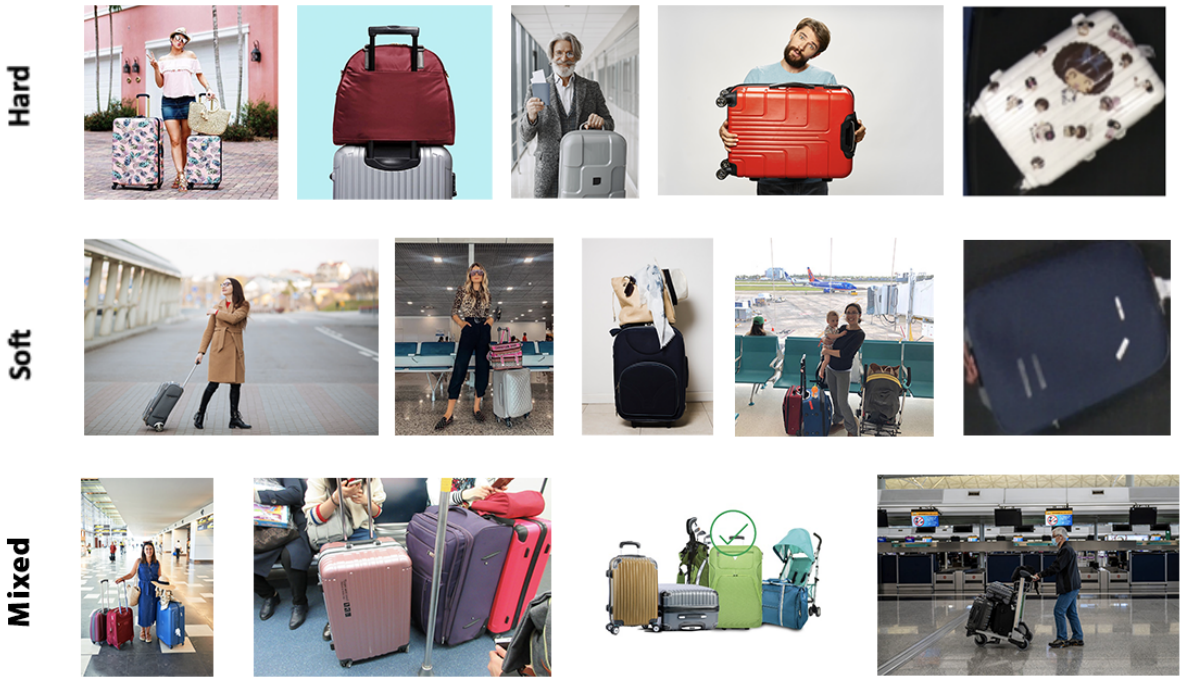


Figure 9. Images in Multiple Baggage Identities Dataset

Relatively disparate images, as shown in figure 10, were added to the test set intentionally, as compared to validation set to check both the transfer learning element of detection and segmentation models as well as their ability to generalize.

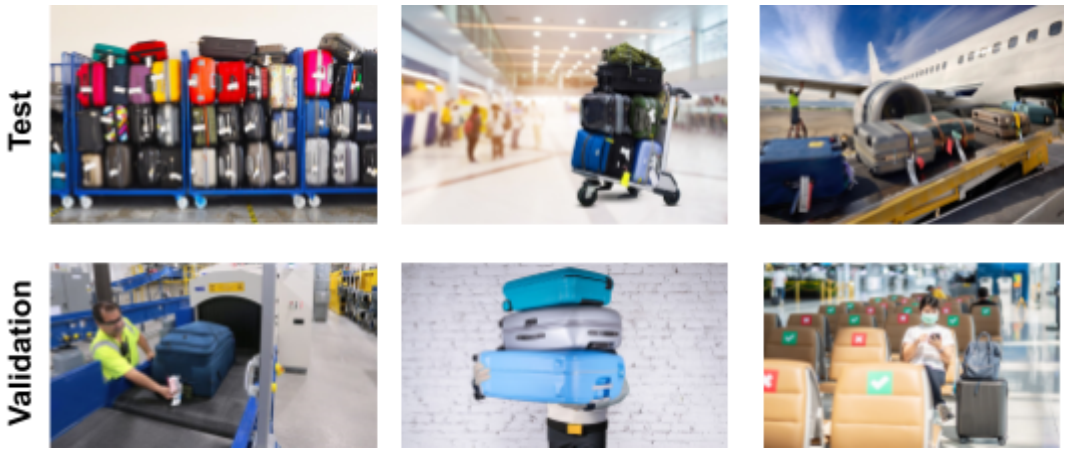


Figure 10. Comparison of Test and Validation Set Images

This dataset was multiplicated in three stages to test the impact of increasing the magnitude of training images on the model performance. This dataset was increased in batches to comply with the industry best practice of checking if larger dataset with new

images produced any significant improvement. As shown in table 1, three stages are named based on the approximate number of images in the training set.

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
Test Set	260		218	201

Table 1. Stages of Multiple Baggage Identities Dataset

3.2 Suitable Camera Types for DNNs

DNNs heavily rely on the input data and in turn on the camera's properties. Most of the cameras depend on encoding techniques (H264 and H265 being the famous ones). Data loss usually occurs whenever data is compressed between the nodes. Encoding and then decoding is an additional step in the process between the nodes in the network communication which could lead to an additional latency. Type of connection between the edge device hosting the model and camera could be wired or wireless and protocols(UDP, TCP being the popular ones) used in communication need to be considered keeping in view the latency, security of the data and data loss. Gstreamer[27] is a popular tool used to handle the incoming media data and could be used for deployment of this application.

Camera that should be chosen for this task depends on the chosen DNN's characteristics such as the processing time. Analysis on the most suitable camera features(such as frames per second) with respect to performance constraints, required for each type of network is detailed in the subsection 3.5.

3.3 Algorithm and Model Design

There are essentially three different types of ML techniques that were adopted for this project. The first technique focuses on object classification, whereas the other two, mask R-CNN and Yolo perform segmentation and detection respectively. Subsection 3.3.1 discusses the first technique adopted whereas subsection 3.3.2 describes the other two specialized techniques.

3.3.1 Classifiers

Classifier network is used to predict the label of the object contained in the image fed to it. Classifiers, however, would fail to distinguish between multiple objects of the different classes present in an image. Owing to this, the MVB dataset described in 3.1.1 was used for training the classifiers. Two different classification networks types namely ResNets and VGG and different variants of these two types of networks were explored. The models were compared on the basis of their validation and test set F1 scores discussed in section 1.1. More detail about these two network types is further discussed in the subsection 3.3.1.1 and 3.3.1.2. Subsection 3.3.1.3 describes the remedies used for preventing the overfitting followed by the data preparation in subsection 3.3.1.4.

3.3.1.1 ResNets and VGG

ResNet is a state of the art classification network that uses a technique allowing the development of very deep networks. Another network type traditionally used is called VGG however, it has a problem of diminishing gradient making it fail to learn when the implemented network is deep. Having said that, we still tested two variants of VGG namely, VGG-16 and a custom-built model using VGG's design paradigm. These models helped us determine the performance of the shallow networks for our two-class computer vision problem.

ResNet here is an enhanced version of VGG which solves the problem of diminishing gradient[13] by skipping the connection and passing the residual to the next layer, thus it is prioritized over VGGish design. With the residual technique the implementation of deep networks became plausible eliminating the otherwise exploding/diminishing gradient problem. An explanation on how the residual is added to the next layer is illustrated in the figure 11 below. All the variants starting from the most shallow type (Resnet-18) to the most deep network (Resnet-152) were tested.

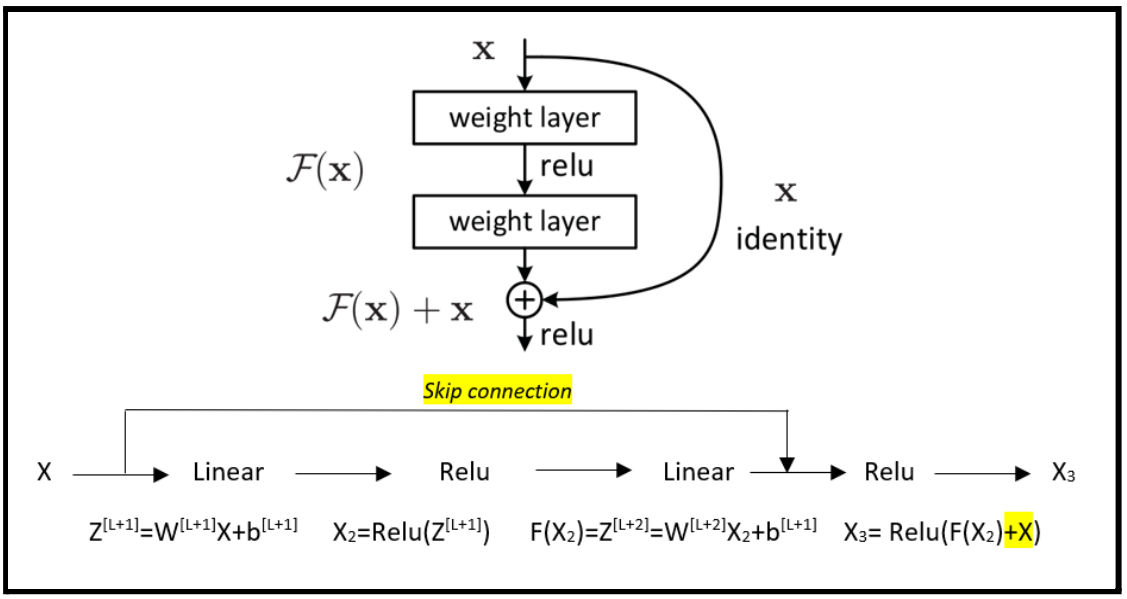
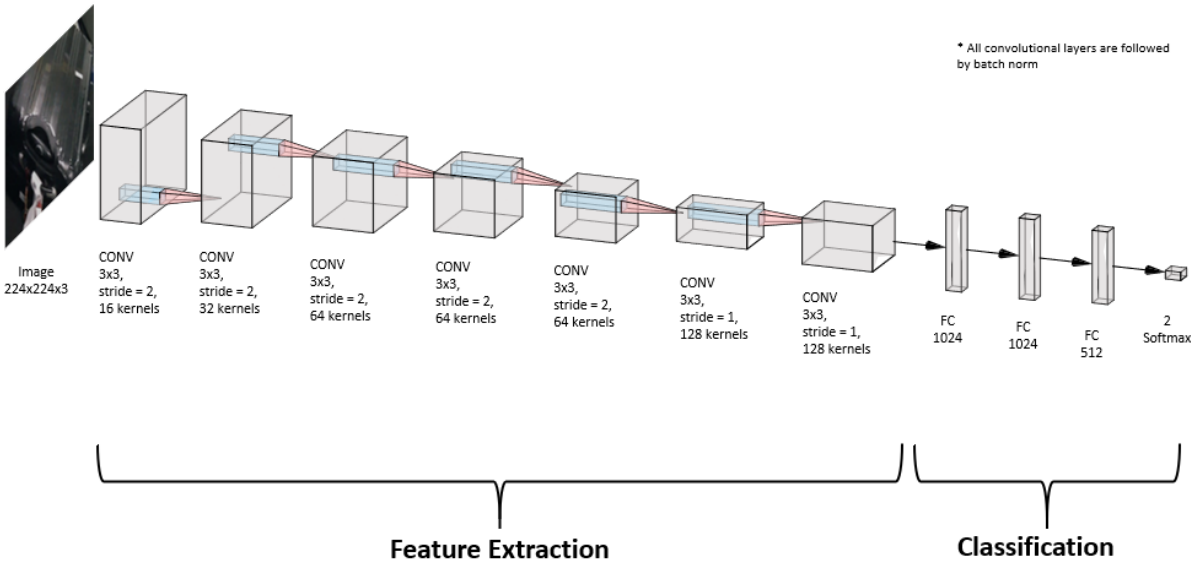


Figure 11 – ResNet residual block skip connection implementation

3.3.1.2 Custom-built Model

Apart from the existing state-of-the-art networks, a custom-built model was created to address the classification problem. Given that only two categories need to be classified and that the pretrained models are typically used for a very large number of classes, using these for the two class problem might be needless. It was imperative to create such a model to see if there's any possibility of training our model even better for this specific problem. Moreover, this gave us an additional testing model to counter the overfitting issues the transfer learning brought as discussed in the results section later on.

The architecture of the model we created is shown in figure 12. There were a total of seven convolutional layers with 'relu' activation and each followed by a batch normalization layer while putting four fully connected layers at the top.




```
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 of Custom CNN Network

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

Initially, only five convolutional layers were added along with dropouts after each of the fully connected layers, but the results indicated that it was quite hard to train the model where the train F1 score was stuck at 85% even after several epochs. To enhance the network learning further, an additional two convolutional layers were added which gave promising results after long periods of training. Dropouts were also removed because there was no sign of overfitting seen. These results are discussed in detail in section 4.

3.3.1.3 Prevention of Overfitting

The state of the art extensively deep models, specifically, 50 or more layered Resnets, have a plethora of parameters making it hard to train such models. Originally designed to classify thousands, if not hundreds, of classes, using these to classify two categories (hard or soft) might be an overkill and eventually could lead to overfitting. On the other hand, taking the advantage of transfer learning, enable these state-of-the-art models to extract the features from the images quite accurately making the overfitting more conceivable. Shallow variants of ResNet having fewer than 50 layers were included in the exploration phase. Several mitigation strategies were applied to ensure that DNNs do not overfit. These include using shallow networks (seen in section 3.3.1.2), introducing an outlier-class and putting randomness in the dataset along with various augmentation techniques. Details on the rest of the two strategies are detailed in the subsections 3.3.1.3.1 and 3.3.1.3.2.

3.3.1.3.1 Outlier Class

In studying the deep learning literature, an effective technique[24] used for reducing the overfitting behavior is to add an outlier to a class. Adding this foreign agent and training on three classes instead of just the bags might help reduce the overfitting problem. Owing to this approach, an additional alien category of cardboard boxes, shown in figure 13, were added to the training set. The prediction was now made in 3 classes.

Figure 13. Outlier Class Sample Images

Since, the testing dataset would not consist of the images from the alien category, if the model classified an image as 'other' the class having the next higher probability would be chosen as the correct prediction by the model.

3.3.1.3.2 Data Augmentation and Randomness

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

Two different approaches were adapted to implement this technique:

1. Pre-training Data Augmentation: In this approach the data is manipulated before the training process and then kept constant for the remainder of the experiment. As the number of augmentation methods increases, this scheme becomes challenging to implement due to increased memory required to store the multiplied data in relation to the storage

constraints. However, this approach gives us a fair comparison of results of different networks by keeping sizes of different variants of augmentation fixed.

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

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

1. Coloring: It changes blue to red color filters and vice versa only. As illustrated in figure 14, the technique used is effective in changing the design color as well which is well-suited to a real-life situation where multiple color bags are available for the same bag. Model must be robust enough to work in such situations. Python library OpenCV's color conversion module 'BGR to RBG'(R=Red, G=Green, B=Blue) was used to achieve the color change.

Figure 14. Color filter comparison - Original image (Left), Altered image (Right)

2. Rotation: The rotation range was kept at 50%. While rotation would help when the camera captures the bag completely vertically or horizontally, for images taken at different angles, too much rotation might not depict a real-life scenario. A more precise rotation will be produced once the camera configuration and testing is finalized.

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

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

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

6. Flipping and Zoom: This was done particularly to multiplicate the data without much realistic component being linked to it as the plan is to fix the zoom of the camera to a certain level.

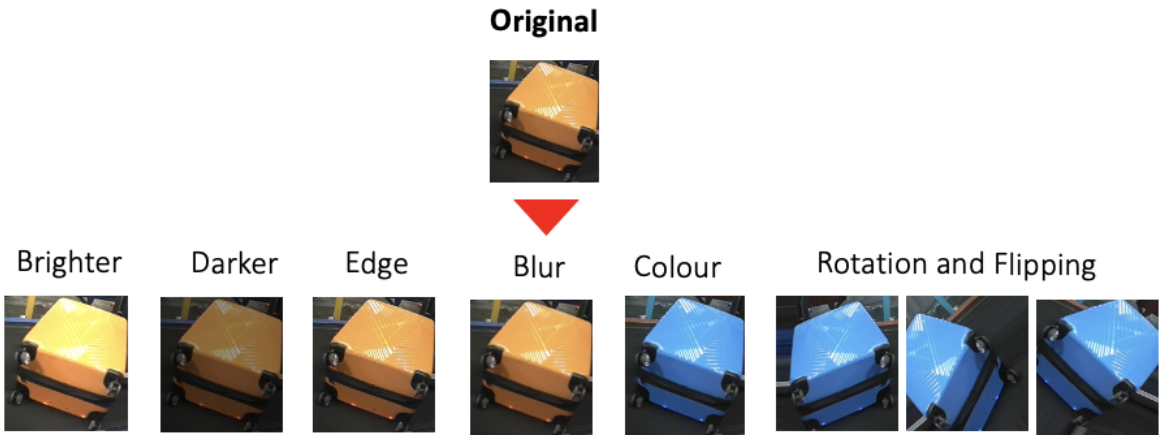


Figure 15. Visualisation of all data manipulation methods

3.3.1.4 Data Preparation

An important observation in MVB dataset (section 3.1.1) is the difference in sizes of the dataset for each class. While the number of hard images is ~12,000, soft images are only limited to one-third the size of hard ~5000. If the model is trained on this entire dataset

without balancing off the difference of the training set of two classes, a bias of the model towards one of the two classes is imminent. To counteract this issue[17], three different datasets were prepared in the following manner:

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

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

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

A complete summary of the number of images for each of the above dataset is described in table 2.

		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**	30	30	30
	Soft	34	34	34
Total	**Hard**	12,030	4,830	12,531
	Soft	12,034	4,834	4,922

Table 2. Precise summary of datasets

The dataset above was divided into two proportions - test and validation. While validation accuracy is a fair indicator for us to evaluate the model's training phase, the model needs to be tested on some images that are somewhat different from the training dataset. A small dataset was scraped from the internet consisting of only 64 images (30 hard bags, 34 soft bags as shown in table-1) and some of these images consisted of multiple objects, but belonging to the same class, as well. Classifiers would be tested on this dataset for comparison. Type of images are shown in figure 16.

Figure 16. Test Dataset for Classifiers (Hard - two Images on left, Soft-two Images on right)

Cost Sensitive Learning:

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

When the number of samples in the classes are not equal, the performance is measured by average cost per example rather than just the error rate. It is possible to see a small error rate but overall poor 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 number of samples per class), the Cost function would be:

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

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

Error rate = No. of incorrectly classified examples / N

Accuracy = 1.0−Error rate

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

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

Figure 17. Average cost vs error rate for imbalanced class sample sizes

Calculating Class Weights

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

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

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

The fraction of each class sample is computed, and the results are normalized to ensure that the convergence of the backpropagation model is not affected. Here the cost vector is the expected cost of misclassifying a sample belonging to i^{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, *#samples(hard) =12,000, #samples(soft) = 4,000*

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

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

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

Calculating the Loss After Setting Class Weights

For every training batch, the traditional loss is shown by formula (C). The new loss after applying weights for the batch is shown in formula (D). The error rate calculation shown in figure 17 is slightly different from formula (D).

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

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

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

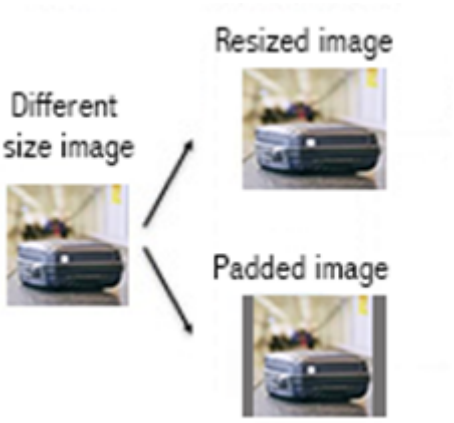


Figure 18. Different sized images being resized or padded. Either of the alterations to image allows it to match the dimensions of the training data.

Due to varying dimensions of images, a standard dimension needs to be set, and the effective padding or cropping must be performed for some images to conform to standard. A limitation of padding is, however, increased features for the neural network to learn and thus might take extra computation and increased training time. Resizing is an alternate solution as shown in figure 18, it can distort the image yet. However, for the images in MVB dataset, the sizes were nearly the same therefore resizing approach was adopted.

3.3.2 Detection and Segmentation

Yolo and mask R-CNN are state of the art ML techniques used for the purpose of both detecting(drawing bounding box) or segmenting(creating a mask) an object and then classifying that object as one of the appropriate classes just like classifiers. The two networks are further detailed in sections 3.3.2.2 and 3.3.2.3.

3.3.2.1 Yolo

Outperforming state-of-the-art methods like Faster R-CNN with ResNet and SSD[10], Yolo was 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 bags, it was prioritized over retraining on a novel dataset. 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 following subsection describes the architecture of Yolo.

3.3.2.1.1 Architecture

Since its evolution, various versions of Yolo have been released with the most recent one being Yolo-v5. Yolo-v2 was the most optimal version of the first release but Yolo-v3 was the final version released by the original author, later introduced with the capability of changing the processing speed by changing the model size without the need for any retraining. Yolo-v4 was later introduced by the deep learning community as an enhanced version of Yolo-v3 and was the final darknet based version. Thereafter, Yolo-v5 was introduced and was claimed to be better than any other versions released so far. Nevertheless, there has been several controversies[28] and failing to draw a fine distinct line between these two final versions, hence it was decided to fully explore Yolo-v3 and hypertune it, thus carrying out most of the work on fitting it to our dataset and then use the same

parameterizations along with the last two stages of Multiple Identities dataset on Yolo-v5 to see if the results can be improved.

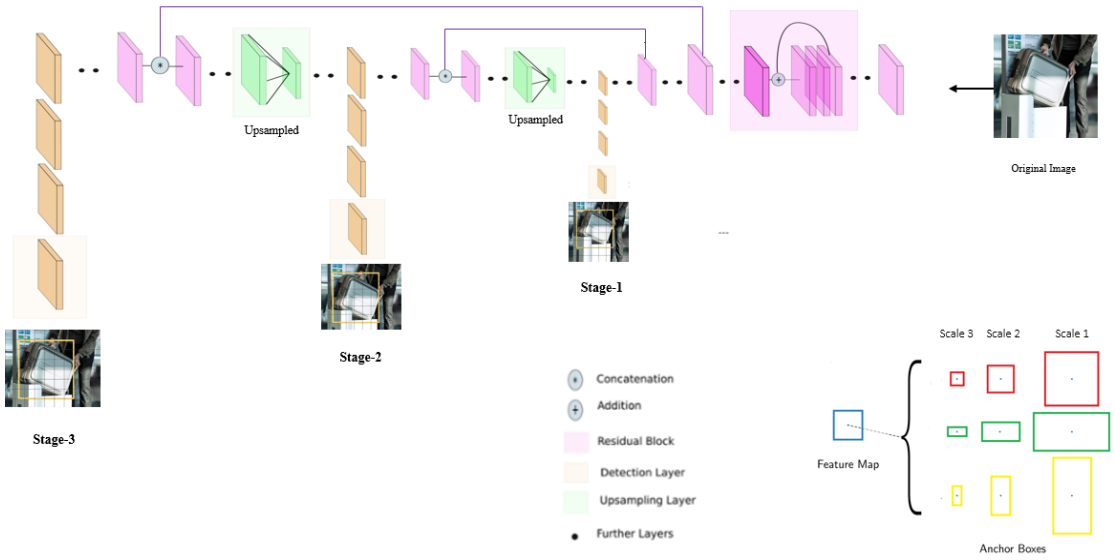


Figure 19. Block Diagram of Yolo-v3 architecture [29] ***- three stages***

The backbone of the Yolo-v3 is shown in figure 19 . Yolo-v3 consists of 106 layers as compared to the second version of Yolo with a 30-layered network and explains why the speed reduced from 45 to 30 fps. The following components forms the part of this yolo network:

Component-1 - Three Scale Detection:

Yolo performs a detection of objects at three scales where each scale typically corresponds to the size of objects it can detect. The original image is downsampled by strides of 32, 16 and 8 respectively where the first scale is responsible for detecting large objects, second for medium sized while the final scaling is used for detecting smaller objects. Detection depends on the anchor boxes. Stride here means the ratio by which the layers downsampled the image. From the architecture shown in figure 19, 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. This technique is similar to residual skip connections discussed in section 3.3.1.1 for ResNets. The previous

yolo version lacks this technique and is often criticized for poor ability to detect small objects.

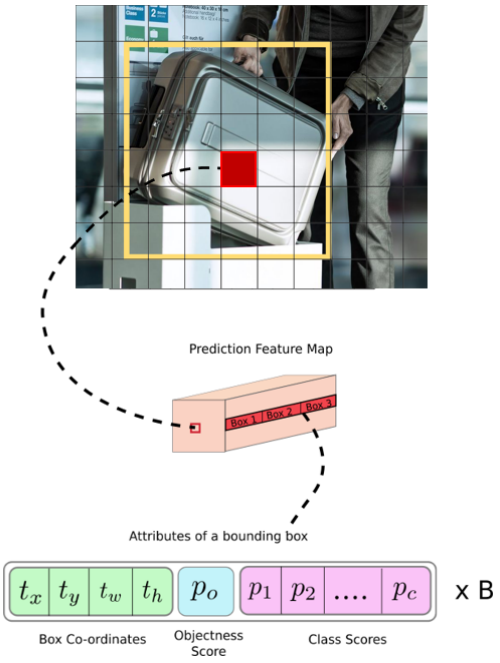


Figure 20. Dimensions of one feature map prediction in Yolo-v3

Yolo's output is generated by applying a 1x1 kernel on feature maps of three different sizes by applying at three scales as mentioned earlier. Output of each kernel results in a 1x1x(B+5+C) as shown in figure 20, where B is the number of bounding box which is 3 for yolo-v3 per scale and 5 includes 1 field for objectness score and 4 fields for the box coordinates of each of the box.

Component-2 - 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, it is possible that one object could result in multiple bounding boxes covering several different regions around the object partially or fully. This can be visualized in figure 21.

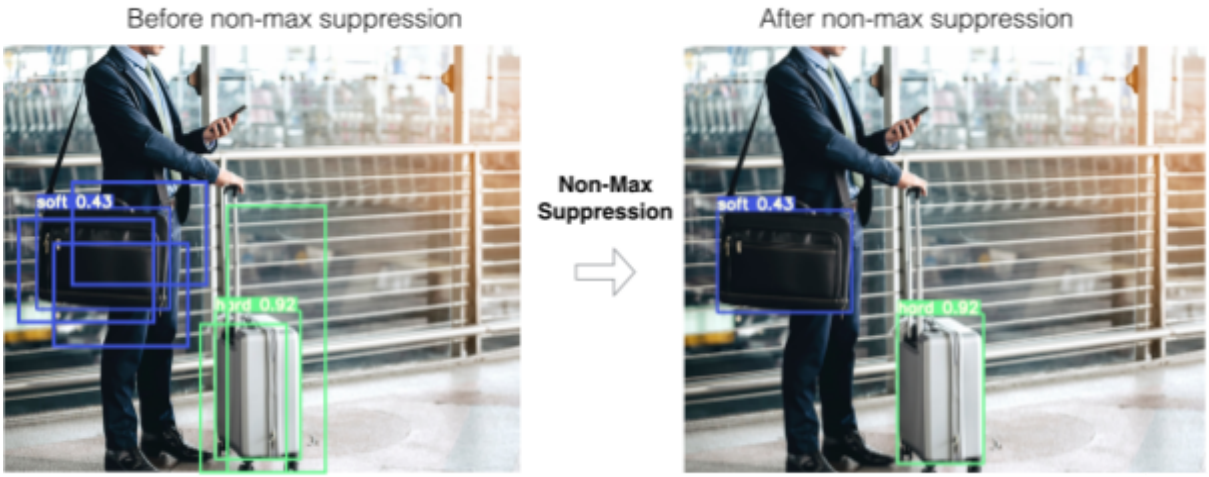


Figure 21. Non-max suppression algorithm results in Yolo Network

In order to handle such cases, yolo incorporates a non-max suppression algorithm that removes the bounding boxes whose Intersection over Union(IoU) score is more than the given threshold(i.e. near bounding boxes) relative to the bounding box with the highest confidence score. Initially, all the bounding boxes with confidence scores lower than a certain probability threshold are already removed, which helps reduce the computation time. This algorithm generates the highest confidence bounding box per object predicted by Yolo from the wide pool of loose predictions. The pseudocode of the algorithm is detailed in figure 22.

Algorithm 1 Non-Max Suppression

procedure NMS(B,c)
  $B_{nms} \leftarrow \emptyset$ Initialize empty set
  for $b_i \in B$ **do** => Iterate over all the boxes
    $discard \leftarrow$ False Take boolean variable and set it as false. This variable indicates whether b(i) should be kept or discarded
    for $b_j \in B$ **do** Start another loop to compare with b(i)
      if same$(b_i, b_j) > \boldsymbol{\lambda_{nms}}$ **then** If both boxes having same IOU
        if score$(c, b_j) >$ score(c, b_i) **then**
          $discard \leftarrow$ True Compare the scores. If score of b(i) is less than that of b(j), b(i) should be discarded, so set the flag to True.
    if not $discard$ **then** Once b(i) is compared with all other boxes and still the discarded flag is False, then b(i) should be considered. So add it to the final list.
      $B_{nms} \leftarrow B_{nms} \cup b_i$
  return B_{nms} Do the same procedure for remaining boxes and return the final list

Figure 22. Non-max suppression algorithm pseudocode[30]

Component-3a - Classification Loss Function:

In the previous version of Yolo, the loss function consisted of traditional squared errors. These are replaced with cross entropy loss in Yolo-v3. This applies to the object score calculation and the class prediction scores as discussed in component-1 above. With cross-entropy loss, these predictions are now made using logistic regression[31].

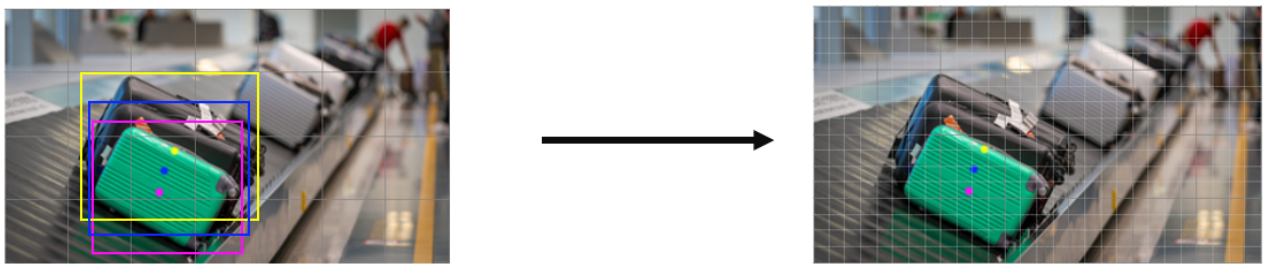
Component-3b - Mutually Exclusive Classification:

The classifiers in section 3.3.1 used softmax function for predicting the final probabilities. Softmax function relies on the assumption that classes are mutually exclusive i.e. an object could be either a dog or an animal but this is not true if correlated classes exist in the dataset. Previous Yolo's versions relied on the softmax function as well. Yolo-v3 no longer rests on such an assumption and uses logistic regression for prediction and the class with the highest probability is assigned as the class of that box.

Yolo is a single shot detection model which means that it performs both the localization search positioning of the object as well as predicts its class using a single flow of the network as opposed to region based CNNs that process regions of objects resulting in multiple flows of the same network, hence consuming more time.

3.3.2.1.2 Yolo Limitations

While Yolo seems to suffice the task, there are certain limitations. Once such a limitation that one needs to be aware of is called multiple objects within a grid box. It is possible that a grid cell would contain the center of bounding box of multiple objects with similar shapes in an image. Yolo atmost supports only 3 bounding boxes per grid cell. As shown in figure 23, having more than three objects in the same grid cell would not be handled by Yolo in such a case because there are only 3 anchor boxes per grid cell. However, even in a 19x19 grid, the probability of more than 3 similar sized objects' midpoints falling in the same grid happening seems quite low and has not been much of a concern lately.



Low Resolution - Intermediate *High Resolution - Real*

Figure 23. Yolo's limitation of multiple mid point centers in one grid box

Just like when objects are put closed together as seen above, another limitation of Yolo in this special scenario revolves around the execution of a non-max suppression algorithm because it could potentially remove some bounding boxes whose IoU might appear greater than threshold due to their position despite these being valid distinct objects. This problem is illustrated in figure 24 below.

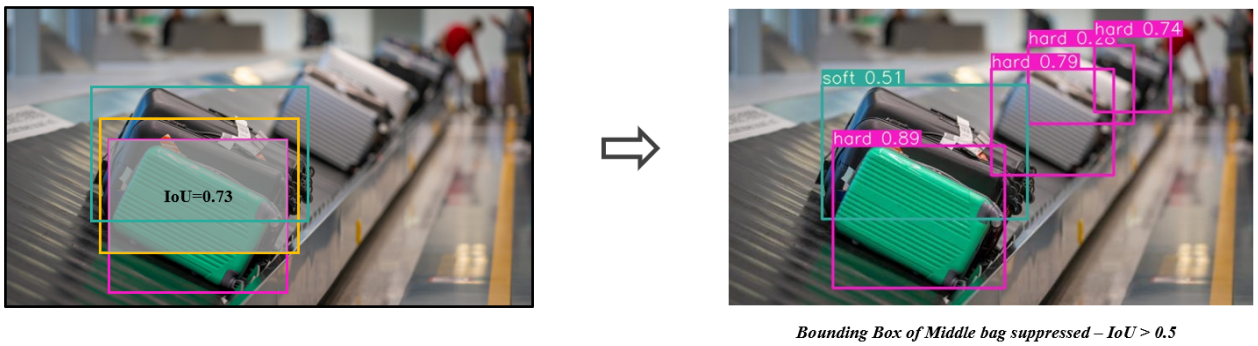


Figure 24. Example of Normal NMS algorithm suppressing valid object's bounding box

3.3.2.2 Mask R-CNN

Mask R-CNN is a state-of-the-art model for image segmentation. It detects the object, segments it by creating a mask as well as classifies it to one of the available categories. The results (bounding box coordinates and label) of the Mask R-CNN are created on the images using OpenCV. The following subsections describe the architecture of this network. This model's training was performed on MVB dataset initially and then 'Baggage Surface-900' dataset in 3.1.2.

3.3.2.2.1 Architecture

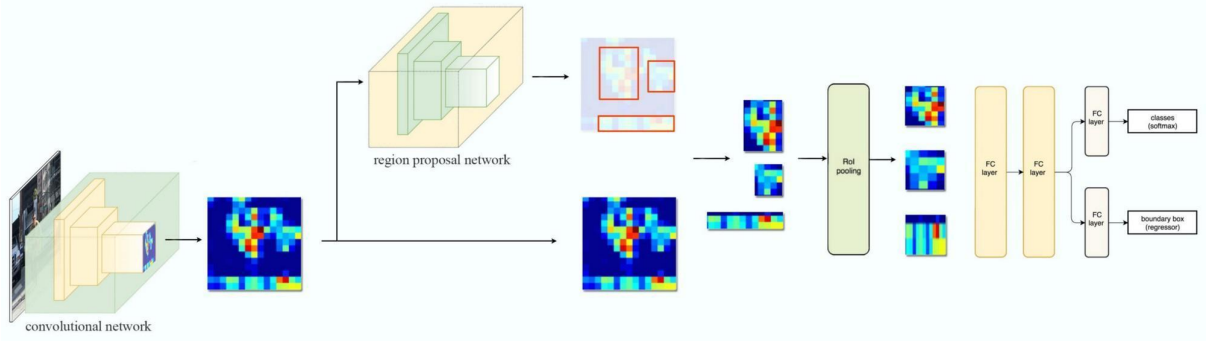


Figure 25. Block diagram of Mask R-CNN architecture. [21] ***It has 4 stages - Backbone, Region Proposal Network, ROI Classifier & Bounding Box Regressor and Segmentation Masks.***

As shown in the block diagram above, Mask R-CNN is a double shot detection model where it first performs the regional proposals of the scanned image to determine the position of the object and then these are classified while generating the bounding boxes and drawing the masks. The two stages are further splitted into two components per stage for better explanation.

Stage 1: Backbone (Shot-1)

Mask R-CNN contains a standard convolutional neural network that is often regarded as its backbone as it is responsible for feature extraction and feature pyramid network. For the purpose of experimentation on Mask R-CNN, ResNet50 and ResNet101 were tested. Lower level features such as edges are detected by early layers whereas later layers are responsible for high level features that eventually represent objects.

The backbone processing down samples an image from 1024x1024px x 3 (RGB) to a feature map of shape 32x32x2048 which becomes the input for the next stages.[22]

In order to support object representation at multiple scales, Feature Pyramid Network(FPN) is used as illustrated by figure 26. A second pyramid is added that inputs high level features from the first pyramid and passes it down to the second pyramid to allow access to diverse features down the stages[22], thus improving the standard feature extraction pyramid. This technique is similar to the upsampling results concatenated with previous layers results to preserve features for smaller objects as seen in Yolo's version 3(this technique is often not used in SSD models) specifically and the skip connections in ResNets.

Mask R-CNN applies the same technique but at a much larger scale and hence requires more processing time.

Figure 26. Feature Pyramid Network allowing pyramids to see high level and low level features

Stage 2: Region Proposal Network (RPN) (Shot-1)

The processed extracted feature from the previous stage of FPN is fed to RPN which uses a CNN with a lightweight binary classifier to scan through the processed image in a sliding window fashion and identify if the region of the image contains an object It does so by using different sized anchor boxes. In principle, 200K varying sizes anchor boxes are used to identify the region of interest. Figure 27 illustrates a simplistic view of RPN applicability on using anchors with sliding windows.

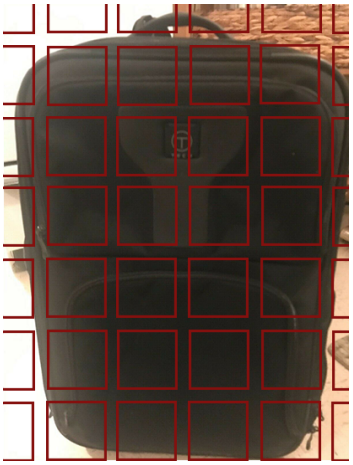


Figure 27. Simplified illustration showing 42 anchor boxes

For each anchor, RPN outputs both the anchor classroom determine the foreground(positive- object detected) or background(negative) connotation for the presence

of object and the bounding box refinement so that the box fits more closely with the object. Just like Yolo, a non-max suppression algorithm is applied to keep the high confidence regions. These are called the final proposals (regions of interest) and are passed onto the next stage. Figure 28 reflects the overall result after drawing anchors just before non-max suppression.[22]

Figure 28. 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.

Stage 3: ROI Classifier and Bounding Box Regressor (Shot-2)

Following stage-2, this stage performs the final analysis and predicts both the class as well as the bounding box for each region of interest pointed out. The two outputs are further detailed below:

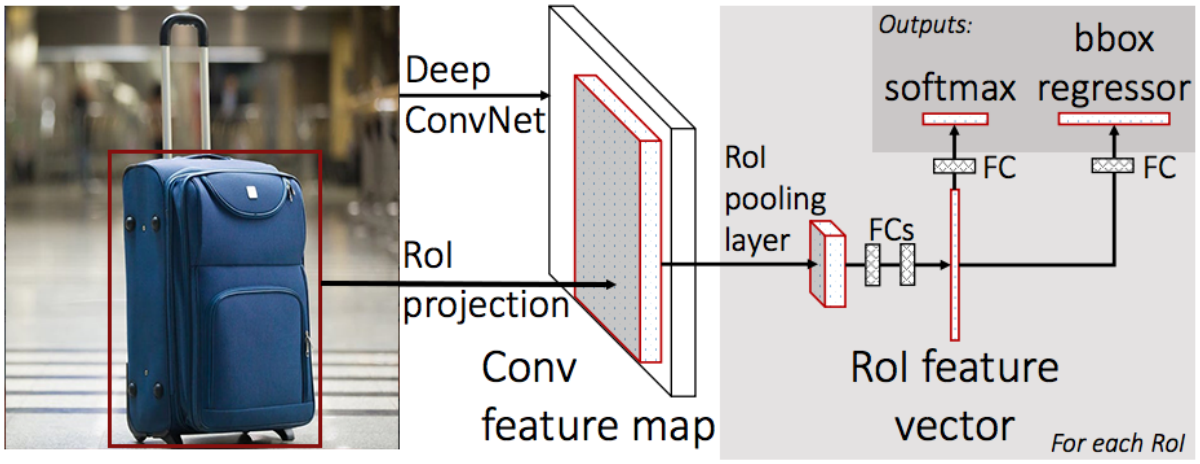


Figure 29. Illustration of stage 3 of Mask R-CNN. The red box represents the ROI

- **Class/Label:** Contrary to RPN which identified the region of interest and gave only two classes(FG/BG), warped features are finally fed to the fully connected layers which classify regions to specific classes (eg. soft, hard) using softmax. A background class might also be generated during the classification and such ROIs are discarded.[22]
- **Final Bounding Box:** Mask R-CNN outputs a bounding box for each object of relevant class. The regression model allows further refinement of the size and location of the bounding box relative to the object[22]

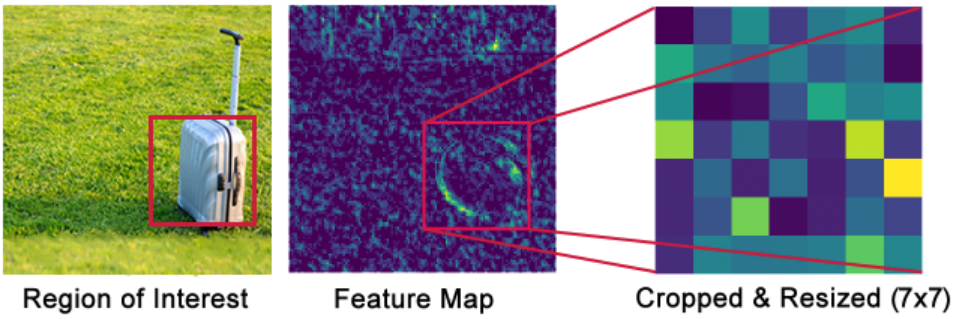


Figure 30. Getting a fixed size feature map from ROI. This feature is a low level layer for ease of understanding

Input images fed to classifiers must be of fixed dimensions but it is noticed that during the RPN bounding box refinement stage, it is possible to produce ROI boxes of varying sizes. To counteract this, ROI Pooling is used which, as shown in figure 30, crops a part of a feature map and resizes it to a fixed size.[22]

Stage 4: Segmentation Masks (Shot-2)

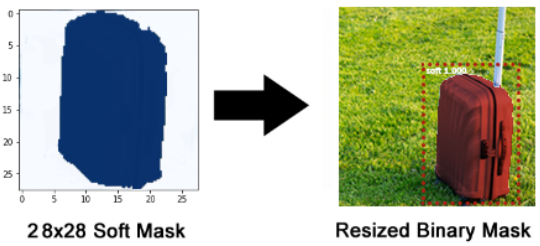


Figure 31. Scaling up the 28x28 soft mask to fit our baggage image

Warped features are also fed to the mask classifier which consists of CNNs to generate marks for the positive regions only. These are low resolution soft masks (eg. 28x28)

represented by float numbers, holding more detail than the binary masks.[22] Ground-truth maska are scaled down to 28x28 during training for the purposes of computing loss and these are scaled up to the size of ROI box to predict the final mask for each object

Mask R-CNN's loss is calculated by weighted sum of different losses at every state of model and consist of a total of five components:[21] :

- ➔ *Rpn class loss*: distinguish objects against the backgrounds
- ➔ *Rpn bbox loss*: localization information of each object
- ➔ *Mrcnn class loss*: classification of the localized objects by ROI
- ➔ *Mrcnn bbox loss*: localization of identified object of a specific category
- ➔ *Mrcnn mask loss*: segmentation of classified objects using mask.

3.3.2.3 COCO weights

With the limited available training data constraint, transfer learning was used by adopting the weights of a Mask R-CNN model trained on COCO 2014 dataset for, which is a huge corpus of images, containing 2,507 images of suitcase[23] as seen in Figure 32.

Figure 32. Sample images for class suitcase from COCO 2014 dataset[23]

The ResNet backbone's weights were initialized to the ones pre-trained on COCO. Initial layers of the backbone were frozen because the backbone has already learnt how to extract the low level features very well. Fine Tuning the weights of the later layers would improve the accuracy of the feature maps hence that of the entire model. The element of transfer learning helps model generalize on the global features enabling these to perform accurate predictions on the image sets not included in the training set. Although Yolo doesn't come with a backbone provided its single shot nature, COCO(2017 dataset) weights were also used for Yolo for enhanced performance without having to spend additional resources.

3.4 Smartphone Application

While the initial scope of the project was to design a fixed camera setup mounted near the self-service bag drop system just next to the conveyor belt, it was later realized that creating a mobile application would be more advantageous and suffice the purpose due to the following reasons:

- HKAA does not need to buy and install the cameras for each self-service bag drop kiosk, reducing both the initial and maintenance costs
- HKAA would be able to monitor model performance using their smart phones at varying environments of airport before putting this into production
- Unlike fixed camera setup, passengers would be able to change the view or adjust camera easily using their smart devices.
- The mobile app can be easily integrated into existing HKAA HKG My Flight[49] mobile app

Hence, the mobile app was implemented after seeking approval from HKAA.

3.4.1 Frontend

The frontend of the application was built using the popular React Native framework (supported on both IOS and Android platforms), using Expo to allow for live hot reloading, because both of us have experience in building robust applications using this framework. The flowchart in figure 33 outlines the user journey of the frontend.

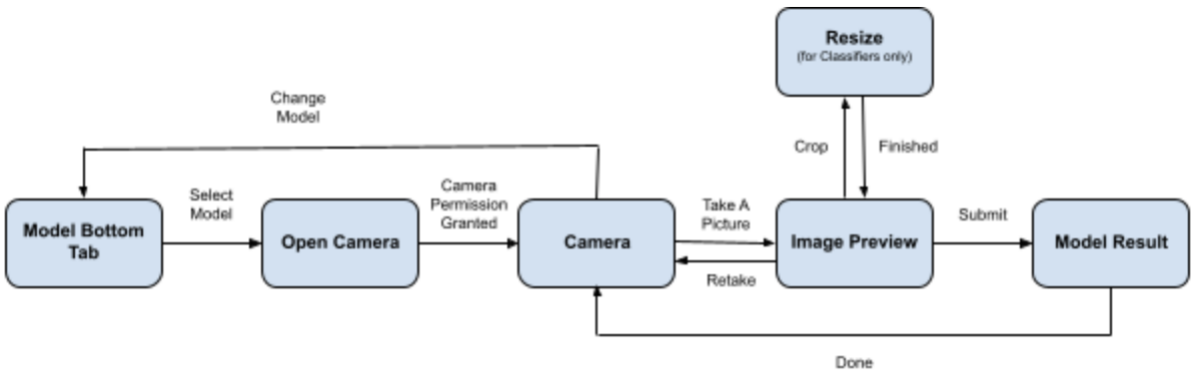


Figure 33. Flowchart describing the user journey from App start to the viewing of results

Upon opening the application, the use by default lands on the classifier tab as depicted in figure 34. Users can switch to the other two tabs - Yolo and Mask-RCNN. After taking the snap, users are prompted to confirm the capture of their image before it is being sent to be processed by the model. Camera inbuilt has both the ability to switch to front-facing as well along with the option of flash light to control the lighting of the environment.

The classifier tab has the option to pre-process the image before being sent to the neural network for prediction. This includes cropping, flipping horizontally and vertically and rotation by 90 degrees. This option has not been added to the rest of the two tabs because unlike classifiers Yolo and Mask R-CNN networks are capable of detecting multiple entities in the same images. Once the user confirms the captured or post edited image, the image is sent to the backend server where models endpoints are hosted and the user is shown the loading screen in the meanwhile. The details of the response shown are discussed in the next section. A pictorial representation of the user journey of the app as described in this section is illustrated in figure 34 below.

Figure 34. Pictorial representation of the user journey. Process of taking the picture and editing/cropping for classifiers (above), Results of Mask R-CNN and Yolo (below)

3.4.2 Backend architecture

The backend of the mobile application consisted of individual virtual machine instances where each model was maintained in its own environment. GPU was not supported for processing the image fed in these virtual machines due to the additional pricing required by AWS, thus only CPU power has been added. There were a total of three instances supporting the best version of each type of the model. For classifiers, since ResNet was chosen as the most optimal model in terms of testing, all its versions were deployed altogether on a single machine. The backend architecture of the mobile application is shown in figure below.

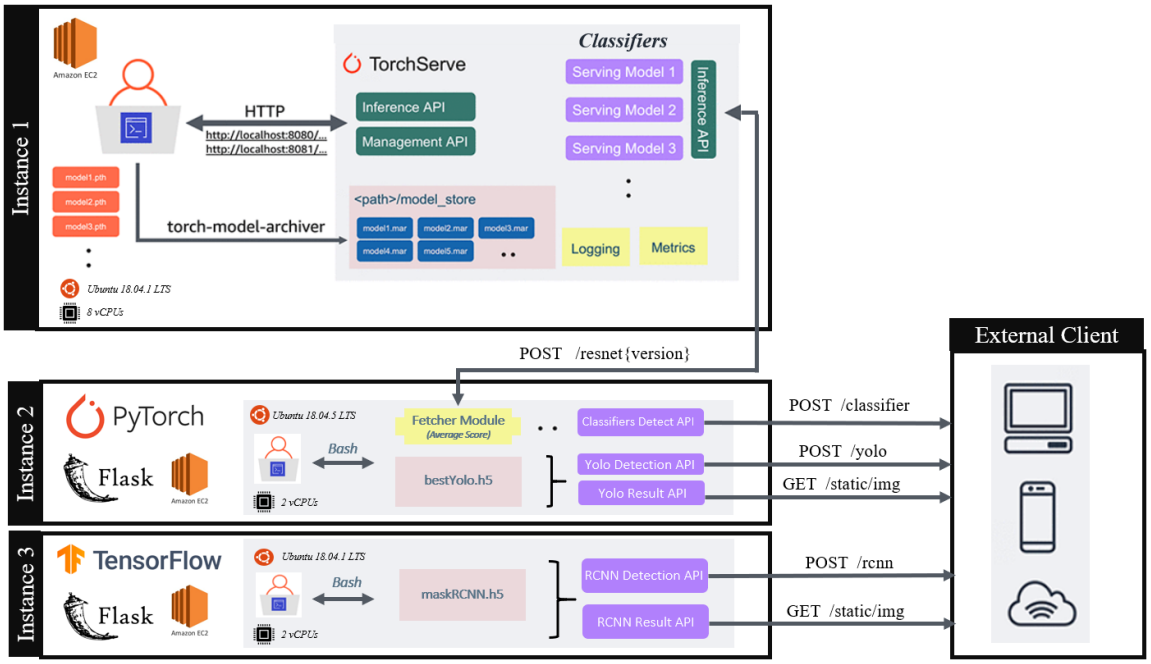


Figure 35. Backend Architecture of Mobile App - consist of three individual virtual machines

As shown in the figure above, 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(for e.g. different versions of GPU and number of CPUs) on the instance as each instance is essentially a linux(Ubuntu) based virtual machine. While Yolo and mask RCNN are served using a python based framework called 'Flask', classifiers are deployed using a pytorch framework called 'torchserve'[32]. It provides inference and management APIs for deploying multiple models and number of workers allocated for each model. Torchserve framework was used to deploy multiple variants of ResNet classifiers and the average score of all these were sent as a response for each image

input to the classifiers endpoint as depicted by instance-1 in figure 35. It could also be seen that classifier endpoints are hosted in instance-1 but the endpoints are requested through instance-2 which is responsible for Yolo network. This is primarily due to the average score being aggregated in instance-2 by sending multiple fetch requests to instance-1 so that the burden of load of several fetch requests is not put on the frontend.

The response is returned in JSON format and it consists of the class(hard & soft) probabilities and response time for classifiers and just the response time for Yolo and Mask R-CNN. For Yolo and Mask R-CNN the server puts the predicted image with bounding box or segmented mask in a static directory in the instance hosted as '/static/img' endpoint by flask which is then fetched by the frontend client after receiving the response to its initial POST request.

The rest of the preliminary details of the instance configuration and the environments are shown in figure 35. Torchserve architecture used for deployment is further described in figure 36 below. Pytorch allows easier management using its Management API through which users can register, deregister, determine model's status, scale workers and determine description of deployed model configuration using simple commands. Some services of Inference APIs include the user to determine the endpoints health status, make predictions using hosted endpoints and get explanations from the served models on their predictions.

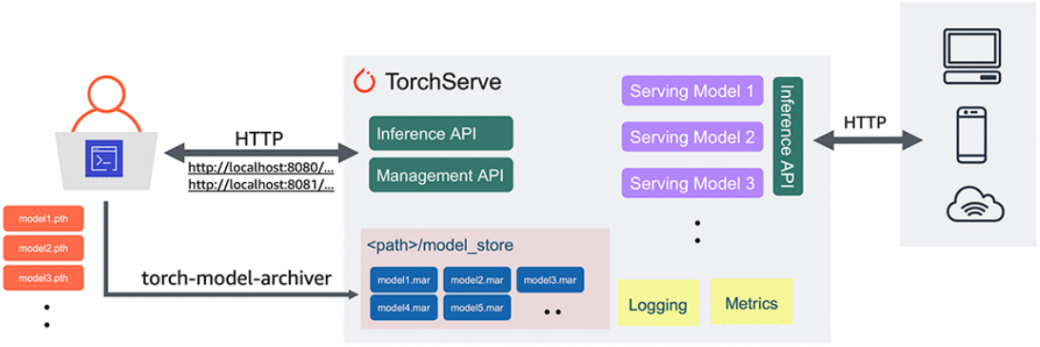


Figure 36. PyTorch framework Architecture

3.5 Comparison of Models

This section compares the strengths and constraints of the three different types of networks discussed above with respect to their deployment. Comparison needs to be drawn between Mask R-CNN and Yolo only as these two models provide a complete solution to the problem. Classifiers are just an individual component of these models and classifiers alone

does not suffice the problem without only one baggage object per image. Table 3 summarizes the general distinguishing features of Mask R-CNN and Yolo.

	Mask R-CNN	Yolo
Detection Type	Localization and segmentation: Determines the position as well as the masks of any regular or irregular object	Localization and bounding boxes: Determines the position and draws a minimum bounding rectangle around that object
Network Architecture	Two Shot Model: Detection and Classification(Uses Resnet as classifier) done in two individual phases	Single Shot Model: Detection and Classification performed in a single phase
Speed	Owing to complex architecture and heavy processing, the speed is dropped to 4-5fps	Light architecture results in speed being as high as 45fps in yolo-v2 and 30fps in yolo-v3 with more than double the layers in yolov2.
Performance	Performance is better than single shot detection models due to deeper architecture and additional processing	Performance is fair but theoretically poorer as compared to region based CNNs

Table 3. Comparison of mask-RCNN and Yolo

Comparison of Yolo with other single and two stage models on MS COCO dataset has been made a part of yolo-v3's paper[31] and is shown in figure 37 below. The table and diagram compares the performance of Yolo-v3 with other competitor one stage models and the high performing two stage networks. RetineNet-101-FPN was found to be performing better than Yolo-v3 but the inference time of Yolo-v3 is significantly better. It is also observed that Yolo-v3's performance is not very far behind from the two stage models despite its lesser processing.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [3]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [6]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [4]	Inception-ResNet-v2 [19]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [18]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [13]	DarkNet-19 [13]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [9, 2]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [2]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [7]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [7]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

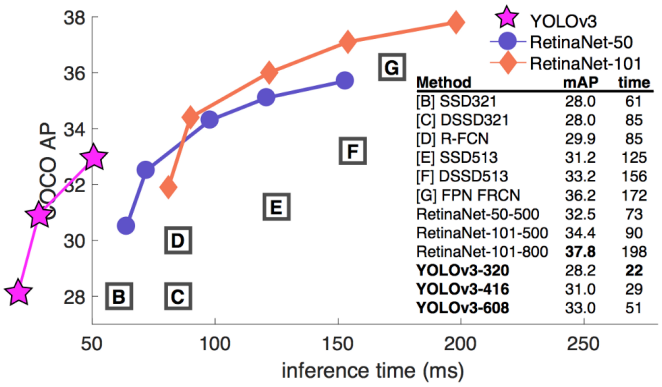


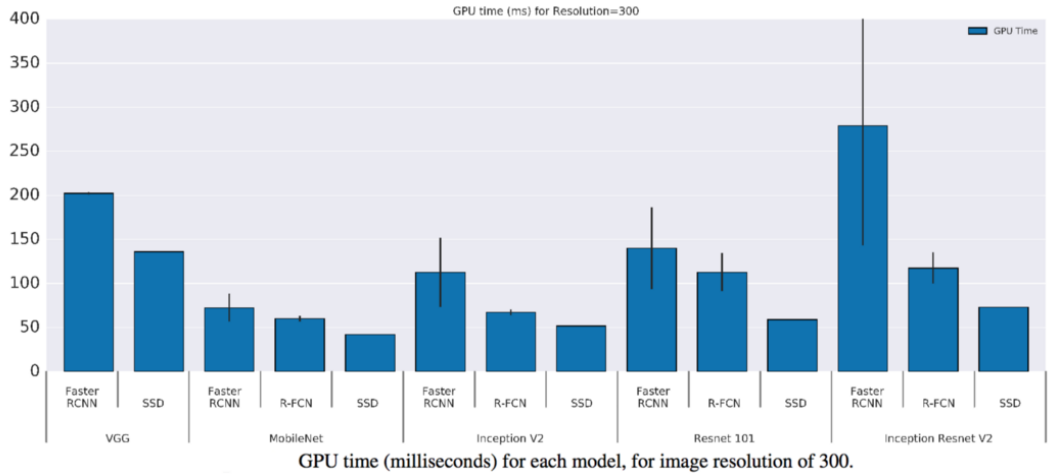
Figure 37. Comparison of Yolo-v3 with single and two stage models on inference time and AP

Results shown in figure 37 could not be compared with the results of figure 38 because these might have been performed with different configurations. For a fair comparison, research conducted by Google[33] could be taken into account. Although yolo and mask-RCNN were not included in the study, other single stage and two stage networks were compared which could give some idea on the speed performance trade-off.

	backbone	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{bb}_{S}	AP^{bb}_{M}	AP^{bb}_{L}
Faster R-CNN+++ [19]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [27]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [21]	Inception-ResNet-v2 [41]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [39]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
Faster R-CNN, RoIAlign	ResNet-101-FPN	37.3	59.6	40.3	19.8	40.2	48.8
Mask R-CNN	ResNet-101-FPN	38.2	60.3	41.7	20.1	41.1	50.2
Mask R-CNN	ResNeXt-101-FPN	**39.8**	**62.3**	**43.4**	**22.1**	**43.2**	51.2

Figure 38. Comparison of Mask R-CNN with Faster RCNN-another two stage model

The figure 39 shows the GPU time vs overall mAP. Evidently, single shot detector SSD has the highest mAP on real-time processing but Faster RCNN with 1fps gives the highest mAP with 300 proposals. The study also shows that Inception-ResNet-v2 as the feature extractor has the highest feature extraction accuracy but two shot detection models performed significantly better with it. SSD struggled with small sized objects and the performance decreased with decrease in input image resolution for both types of networks. The GPU time and memory consumption requirements which are integral for the hardware components used in production are shown in figure 39 below:



GPU time (milliseconds) for each model, for image resolution of 300.

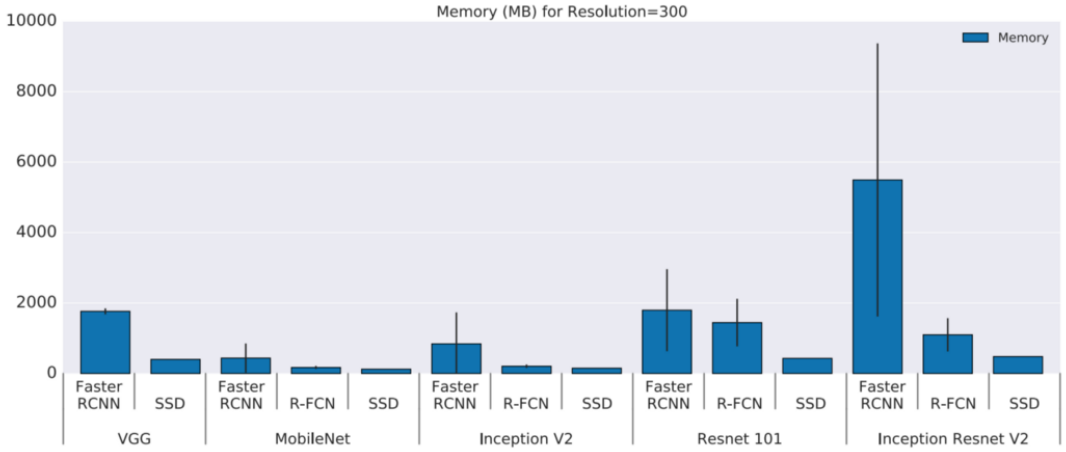


Figure 39. GPU time(above) and Memory Consumption(Below) of Single and Two Stage Models according to Google's study

To summarize, our methodology section consists of a data collection stage where two different types of data were collected for the networks tested as a part of the scope of this project. Suitable camera types and various underlying factors to consider for images captured for training the DNNs are explained and these must be considered during the deployment stage. The three types of models were tested - classifiers, Yolo and Mask-RCNN. Classifiers are primarily used for single entity images whereas the other two networks are used for detection and segmentation purposes. Yolo is regarded as a single stage network while Mask R-CNN is theoretically better in terms of performance but slower when compared to Yolo. A mobile app was included in the scope of the project both as an alternative option as well as better visualization of the results of the studies conducted as a part of this project.

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 results on classifiers followed by Section 4.2 and 4.3 that describes the Yolo and Mask R-CNN respectively. Environments used for implementation and testing of networks are listed in table 4 below.

Networks	GPU Library	Python Version	Linux Type	RAM	CPU Type	GPU Type
Mask R-CNN	Tensorflow GPU 1.15.2 Keras	3.7	Ubuntu 18.04.5	28 GB	Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz	NVIDIA GeForce GTX 2080 Ti
Yolo and Classifiers	PyTorch version 1.7.1	3.7	Ubuntu 18.04.5	28 GB	Xeon (R) Silver 4108 CPU @ 1.80GHz	NVIDIA GeForce GTX 1080 Ti
Custom Model, VGG-16 and ResNet-50 (Initial)	Tensorflow GPU 1.15.2 Keras	3.7	Ubuntu 18.04.5	30 GB	Xeon (R) Silver 4108 CPU @ 1.80GHz	NVIDIA GeForce GTX 1080 Ti

Table 4. Environments - GPU/CPU/OS types, RAM capacity and library versions used for each network

- 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.
- Yolo and Classifiers: GPU 1.15.2 PyTorch library version 1.7.1 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.

Note that Custom Model and initially ResNet-50 was built on the following tensorflow environment before shifting to PyTorch:

- 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 Classifiers

4.1.1 Network Training and Hyperparameter Selection

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

Custom Model and VGG:

The custom model was trained on a complete sampling dataset by setting class weights. The model was trained on 300 epochs and the training F1 score went as high as 94% and validation reached 93%. Surprisingly the declining training loss trend still shows the sign of additional training to obtain improved results.

Owing to the long training time needed for the complete sampling data, the under sampling data was used to see how far the training can go. A similar behavior was observed where the model was inclined to learn even more if additional epochs were added. As shown in figure 40, the training loss (<0.15) even went lower than that in the complete sampling data. Also, the rate of decline of loss is noticeably steeper than complete sampling's.

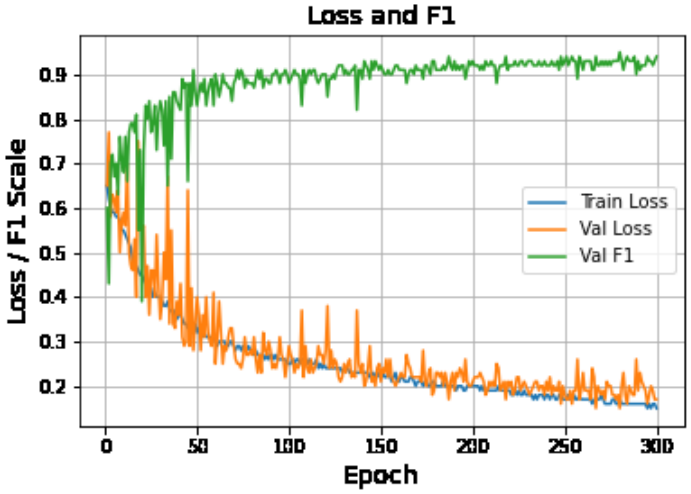


Figure 40. Custom Model on under sampling data

Before spending more resources on it, the weights of the epoch with the minimum loss were loaded and the model was tested on the undersampling dataset in table 2. The test F1 score was just limited to 34%. These results imply that the custom-built model performs quite well, and it is possible to achieve a quite high accuracy of >94% with more training, however, it is noticed that the model would only work well when tested on images that are similar to the ones used in training. This could be attributed to the lack of element of transfer learning in this model trained from scratch.

We ought to introduce the transfer learning element with the VGG-16 model, whose design was mimicked to create a custom model, using the same dataset. VGG-16 though performed almost identically to the custom model, showed early signs of overfitting even after on-the-fly-data augmentation of flipping, zooming, rotation and range shift. The best test F1 score (~48%) output by the experiment had the configuration and results shown in figure 41 and table 5. It is also noticed that even with transfer learning, while VGG is better at generalizing that custom-built model, it is far from the expected results.

Model	Layers	Learning Rate	Epochs Trained	Best Validation F1	Best Train F1
VGG-16	All unfrozen	0.00001	500	96.52%	98.8%

Table 5. VGG Experiment Configuration

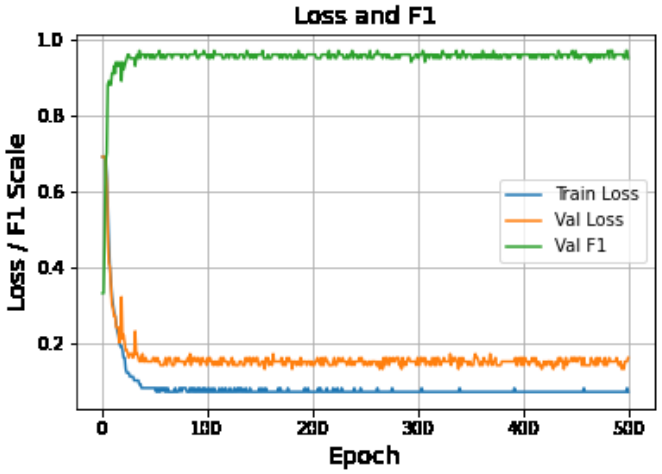


Figure 41. VGG results

ResNets:

From the available resnets, ResNet-50 was chosen as a standard i.e. not too deep or too shallow. All initial experimentations were done on this network. Overfitting was the most prevalent issue that limited the performance of ResNet-50. Various experimentation was performed, and steps taken to resolve this issue.

Initially the experiments were performed on the under sampling data because of its smaller sized nature helping us achieve results quicker and enabling us to make relevant changes that are best suited for our dataset. Five experiments were performed, and their description and the results are listed in chronological order in the table 6 below.

	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 6. ResNet-50 results on experimentation performed on under sampling data

These experiments were performed in conjunction with what was realized from the results of previous experiments and then making that relevant changes with the hope to produce better results. Overfitting is clearly observed despite the addition of noise.

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

Upon evaluating the ResNet50 on test data, the test F1 score was higher (~76%) than the VGG's and custom model's. Besides skip connections being used for this deeper network, this is because of the transfer-learning component of ResNet-50 being utilized as the image-net weights were loaded prior to training. ResNet is better at predicting the global images because of pre-training on image-net images.

With the above results, it is realized that ResNet-50 is quite good at predicting the images it has never seen but the validation F1 score is just limited to ~94%. Provided that this model needs to put in production, a practical figure of validation F1 score of at least 96% is required. On the other hand, a custom model has the potential of predicting even better than ResNet-50 without any overfitting issues but is only confined at making good predictions for the type of data it is trained on. With this implication, ResNet is clearly a better fit for our use case in terms of the results and steps must be taken to alleviate the overfitting issue.

4.1.2 Experimentation on Overfitting Mitigation Strategies

<u>Outlier Class</u>:

Owing to the outlier addition approach discussed in section 3.3.1.3.1, an additional 2,500 alien images were added to the training set. This produced a dataset consisting of 7,500 images, each class having 2,500 images in the train set.

ResNet50 was trained with the last 12 layers unfrozen and three final fully connected layers with dropouts. Figure 42 shows the results of this experiment where it can be observed that the overfitting issue was only slightly relieved(more epochs now being trained) but still the training was stopped after validation loss started increasing and the maximum validation F1 achieved was just 80%.

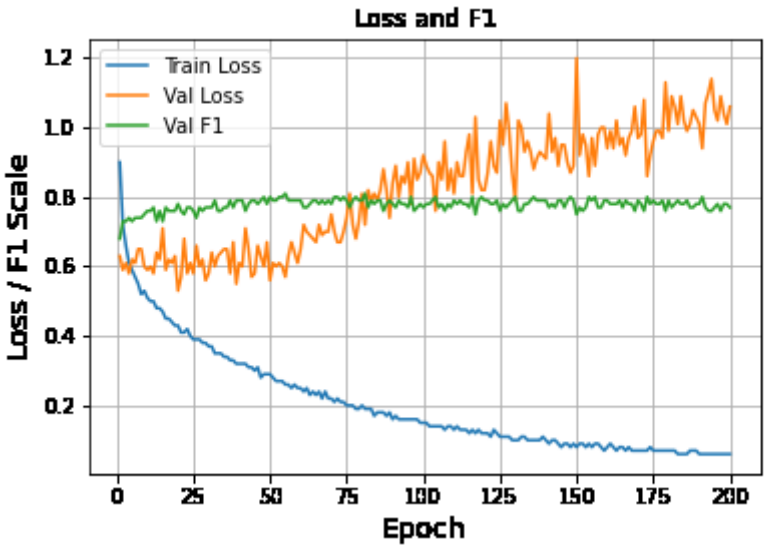


Figure 42. ResNet-50 training on under sampling data with outlier

The model achieved an overall test F1 score of just 49% with this technique. Upon analysing the performance, it was observed that the model had a recall of 90% for hard bags showing that it was quite biased towards hard bags. Upon analysing the validation data, there were instances of outlier-class prediction but the next highest probability for such samples were always hard showing the model's bias. This technique did not help much so we do not plan to move forward with it as it added another layer of complexity without helping the model to learn much.

Post Train Image Regional Multiplication:

This technique involves zooming in at the four corners of the image and then resizing the image to the original dimensions. An example of how this was done is shown in figure 43 below. Instead of one original image, four subsets of the image are fed and the average of all the probabilities is taken before outputting the final result. This idea is an absolute simple version of how prediction is done in region based networks. This technique did create some space of overcoming guessed predictions or novel images, but it did not help much with the overfitting issue.

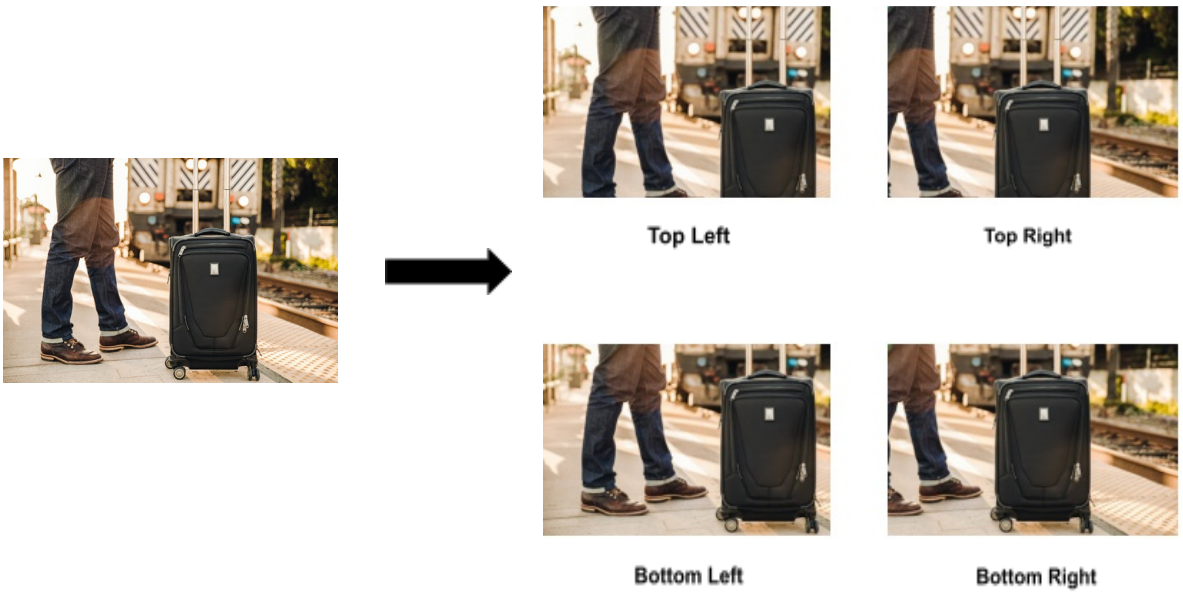


Figure 43. Depiction of Image Regional Multiplication. It consists of 4 images from each corner of the original image

Random Resized Cropping:

This is another augmentation technique that is similar to 'Post Train Image Regional Multiplication' mentioned above except that it involves augmentation on the fly during training epochs. For each image set fed to an epoch, a portion of image is randomly chosen and then the image is scaled to the original dimension before being finally input to the training. An example illustrating this technique is shown in figure below.

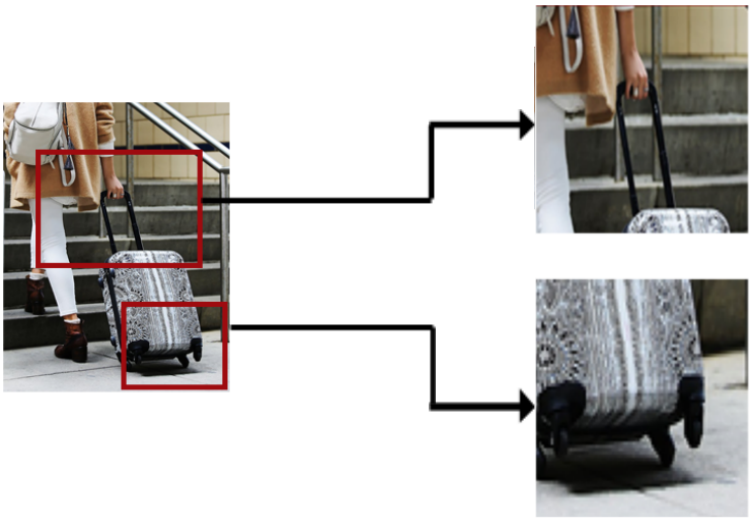


Figure 44. Two possible options for random resized cropping

The inspiration of testing this technique was taken from another very similar yet advanced version of the same technique called RICP(Random Image Cropping and Padding)[34]. In addition to random cropping of one image, several images are cropped and joined together to form a complete image of irregular sizes of child images. An example has been illustrated in figure 45 below. The amalgamated image has its label probabilities marked according to the proportion of the sizes of each child images. A study[35] shows drastic improvement in mitigating overfitting and overall training using this technique.

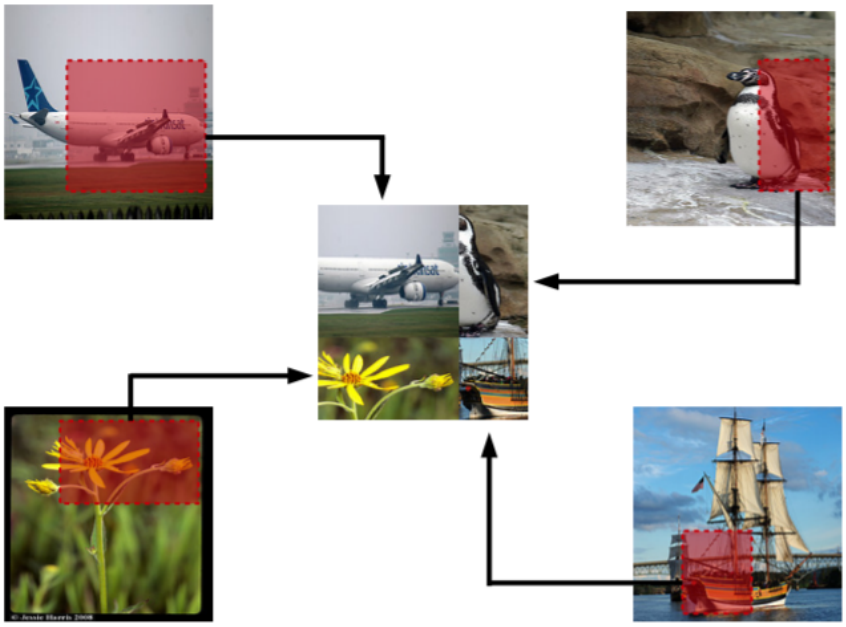


Figure 45. Illustration of RICP and formulating a new image from a set of images

Fortunately, inclusion of this technique solved the overfitting issue. This can be thought in terms of neural networks sometimes tend to learn recognition shortcuts when datasets are biased. It was realized that due to the highly specific images in the MVB dataset, models had difficulty generalizing and learned such patterns resulting in overfitting. With the overfitting now resolved, other variants of Resnets were tested out and the comparison between their performances in terms of test data and validation data is summarized in following subsection 4.1.3.

4.1.3 Comparison of Classifiers

All the experiments were performed on the undersampling dataset and maximum number of epochs set to 300. For each version of a network, layers were adjusted by adding some stacks on the head, freezing and unfreezing networks partially or fully as well as hypertuning. Only the best models were chosen and their metrics are displayed in figure 46.

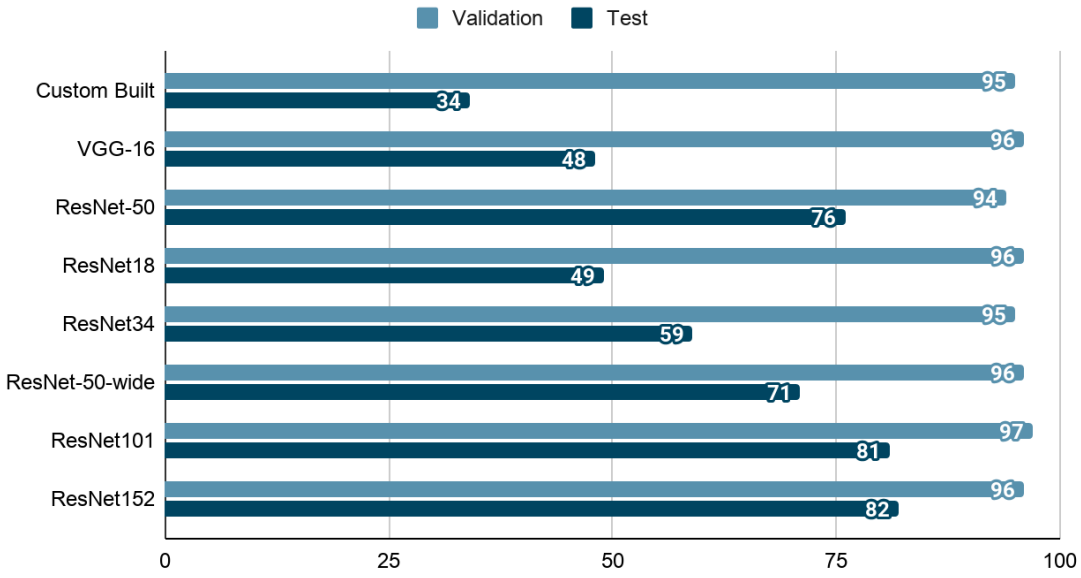


Figure 46. Model convergence on seen vs unseen data

From the results, it is clear that in terms of validation, all the models performed quite well. Nevertheless, when the test set images were fed, ResNets outperformed the other two networks. And primarily, ResNet-50 or ResNet101 could be chosen as the most optimal one, provided their high F1 test score. If processing time is not the bottleneck, ResNet152 should be prioritized.

It is to be noted that the test set just contained 64 quite different, as mentioned earlier as well, images from which the model was trained. Validation dataset consisted of zoomed-in images just like the nature of images in MVB dataset. Furthermore, as mentioned in the frontend user story of section 3.4.1 that for classifiers testing, the user has the option of cropping the bag object before sending it to the model for classification, this would make work for classifiers even more easier and, the validation F1 score would be a better reflection of the classifiers' performance in this realistic scenario. An example in figure 47 depicts how the average score of classifiers(ResNet-18, Resnet-34, ResNet-50-wide, ResNet101, ResNet152) performance improves once the same image is cropped and is completely focussed on the bag.

Figure 47. Classifier performance on original vs cropped images

It could be seen in figure 47 that the confidence score of the prediction of a hard bag(leftmost image) increases by 17% when the same image is cropped to focus on the bag alone. For the soft bag(in the middle), it could be seen that the score increased by cropping out and the model was robust enough to not get confused by the hard surface on which the bag is being placed on. While it could be thought of the person's clothes making it easier for the model to correctly classify this bag as soft, however, removing the person in the cropped image, the score increased even further. For the right-most image, the uncropped version of the soft bag image was misclassified as hard and that too with the high score. This could be due to multiple objects being present in the image and most importantly the bags behind this subject bag(for eg. the red -colored bag) are hard which might have confused the model. This reasoning is strengthened by the model's correct classification of this bag as soft after the cropped version is tested.

Thus the highest 82% F1 score of ResNet-152 on the test set(original uncropped images) is quite remarkable. Moreover, 64 images is a very small sample size to judge the performance but it gives an idea of the performance and ability of models to converge on unseen data.

4.2 Yolo

4.2.1 Network Training and Hyperparameter Selection

All the experiments done on Yolo were trained on 400 epochs each. SGD optimizer was used as it gave the most generalised results. Initially, Yolo was trained using standard hyperparameters with some layers frozen. To compare the performance of different versions, mAP@IoU value was used. It was observed that unfreezing the layers and hypertuning these even further improves performance. Thereafter, all the layers were kept unfrozen for the rest of the experimentation. All the experiments were carried out on Yolo-v3 and upon obtaining the best hypertuned parameters in addition to the new batch of dataset as mentioned earlier, these were used on Yolo-v5 later to achieve any better results. The subsection 4.2.1.1 describes the experimentations performed steps taken to improve performance on Yolo-v3 network followed by subsection 4.2.1.2 that discusses significant improvements after running the most optimal batch of parameters on Yolo-v5.

4.2.1.1 Yolo-v3 Experimentation

Apart from individual versions of Yolo, multiple subserversion exists. These were primarily built to support Yolo on different platforms. For Yolo-v3, three subserversions exist shown in figure 48 along with their associated constraints. Smaller networks like Yolo-v3-tiny occupy less memory with reduced performance but are suitable to be employed in the app's bundle. For our testing, we used Yolo-v3, being the most optimal in terms of performance out of all three.

Model	AP^{val}	AP^{test}	AP_{50}	$Speed_{GPU}$	FPS_{GPU}		params	FLOPS
YOLOv3	43.3	43.3	63.0	4.8ms	208		61.9M	156.4B
YOLOv3-SPP	**44.3**	**44.3**	**64.6**	4.9ms	204		63.0M	157.0B
YOLOv3-tiny	17.6	34.9	34.9	**1.7ms**	**588**		8.9M	13.3B

Figure 48. Comparison of subversions of Yolo-v3[36]

Initially, 74.1% mAP@IoU[0.5] and 63% mAP@IoU[0.5-0.95] was achieved. Several strategies have been applied to bump these values, describes as follows:

Anchor Box Fitting:

Yolo-v3 uses 9 anchor boxes of varying sizes in total and grasps each object that corresponds to the best fitted anchor box. As mentioned earlier in section 3.3.2.1.1, three anchor boxes are used in each of the three scaling stages, this means that if there's a certain object that does not fit any anchor box, the IoU value would be impacted and hence affecting performance. To counteract this, instead of using the standard anchor box sizes, we used a popular clustering algorithm K-means[37] to create 9 clusters as relative to the sizes of objects present in the training dataset. K-means clustering works by randomly generating 'K' centroids initially and then placing each coordinate to the cluster closest squared distance from its centroid. This process goes on and the final average of all the points in a cluster is taken. The nine clusters formed by K-means algorithm and the sizes of the nine bounding boxes used in the final test are shown in figure 49.

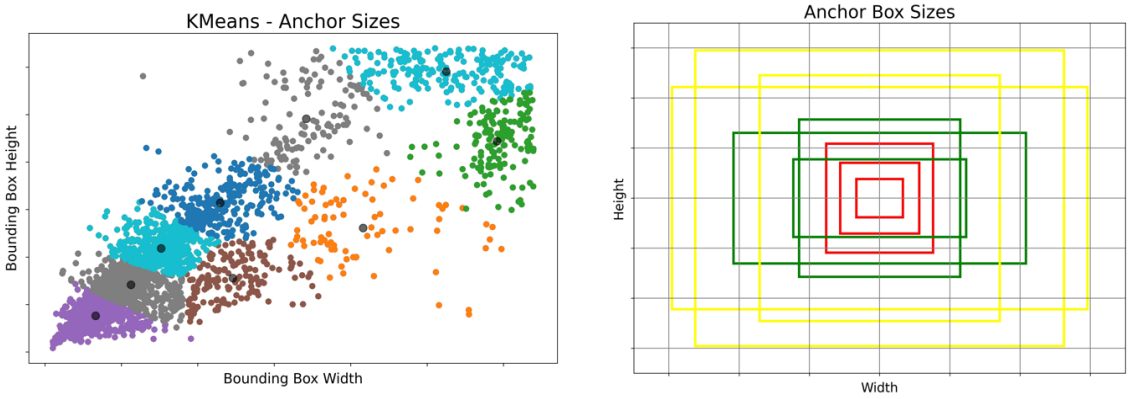


Figure 49. Results of K-means clustering and the final anchor box sizes

An experiment was performed keeping all the previous parameters constant and performance was improved significantly to ~79% mAP@IoU[0.5] and 67% mAP@IoU[0.5-0.95] on validation dataset.

Data Augmentation- Mixup:

Mixup[38] is a simple yet very powerful data augmentation technique that has increased the performance of Yolo drastically. This technique simply involves blending two images to create a new one. The target of the new image is the proportion of the original targets as shown in figure 50 below. The key idea is that custom loss function needs to be defined without marking labels using one hot encoding as done in the classifiers.

```
mixedImage = alpha * firstImage + (1-alpha) * secondImage
mixedTarget = alpha * firstImageLabel + (1-alpha) * secondImageLabel
```

Figure 50. Formulae for computing the mixed up image and new target class

_____This technique helps achieve improved performance with regularization. Hence, it prevents models from learning too specific features that could result in overfitting and thus generalize quicker. Furthermore, due to the simplicity of the process in producing the images, the overall process was fast. Figure 51 illustrated the example of this technique.

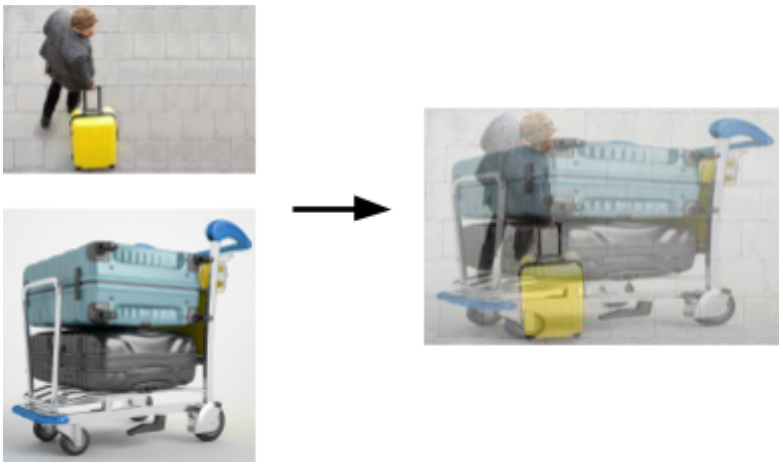


Figure 51. Example of how two images are mixed up with alpha = 0.5

Data Augmentation- Miscellaneous:

Varying magnitudes of popular data augmentation techniques were tested. These include, shear, translation and flipping(horizontally only), and rotation. Vertical flipping was avoided because this would not appear in practice. Rotation's magnitude was also kept low due to the same reason. This is to multiplicate the data as much as possible but also to avoid abnormal data being fed to the network. Another type of augmentation in terms of color other than RGB is called HSV[39] (hue, saturation, value). It distinguishes the colors of images better than RGB space because of its instinctive nature of describing the colors. Here, value defines the brightness of the color, saturation reflects the depth of purity whereas hue represents chromatic information. Figure 52 and 53 illustrates the HSV and miscellaneous augmentation together respectively.

Figure 52. Illustration of HSV augmentation

Figure 53. Miscellaneous augmentations all together during the training phase of Yolo-v3

The magnitude of hue, saturation and value would often depend on the background variable i.e. environment and lighting and changing these significantly could backfire. Miscellaneous augmentation did not bring any massive improvement but evidently regularized our model by converging earlier. All the experiments performed in the above and their configurations are summarized in table 7 below.

		Experiment-1	Experiment-2	Experiment-3	Experiment-4
Remarks		**Standard Parameters**	**Custom Anchors**	**Mixup Augmentation**	**Miscellaneous Augmentation**
	mAP @ IoU[0.5]	74.1%	79%	80%	80%

Validation	**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 7. Results of experimentations on data augmentation

4.2.1.2 Yolo-v5 Supplementary Test

Just like Yolo-v3, three subservions of Yolo-v5 exist. These are compared according to their processing speeds in figure below.

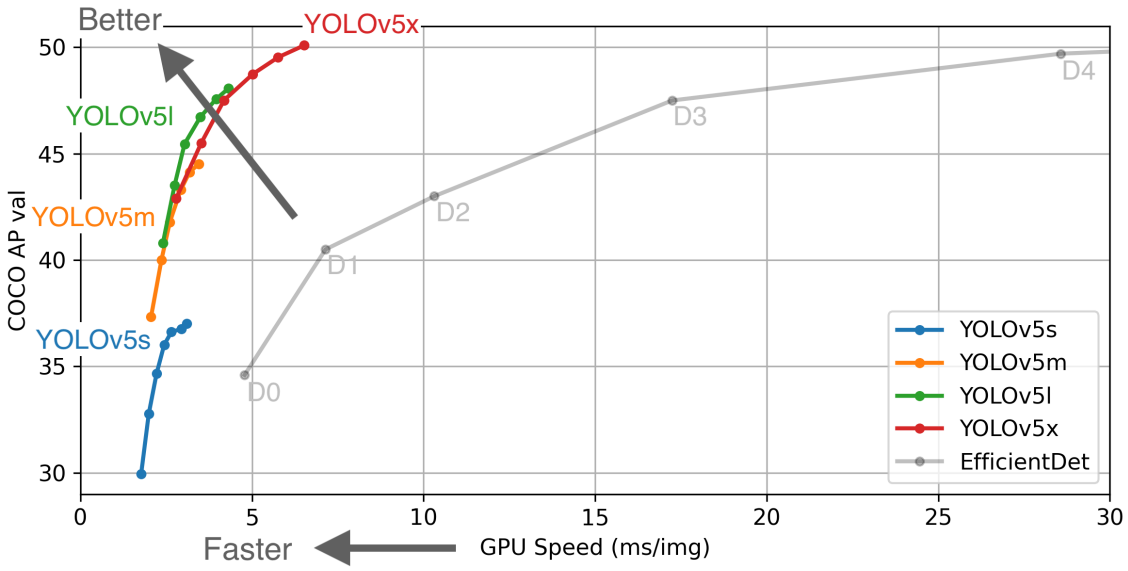


Figure 54. GPU Speed vs AP for subs versions of Yolo-v5[40]

Taking the same parameters as described in the previous subsection, Yolo-v5x network was used because of its high performance without any significant impact on the overall time. Two experiments were performed with different proportions of the dataset as summarized in table 8 below. Yolo-v5 improved the performance significantly.

		Experiment-1	Experiment-2
Dataset		**Baggage Surface-1700**	**Baggage Surface-2000**
Validation	**mAP @ IoU[0.5]**	88.3%	88%
	mAP @ IoU[0.5 - 0.95]	84%	83%
Test	**mAP @ IoU[0.5]**	87.4%	87.2%

Table 8. Results of Experimentation on Yolo-v5 with additional datasets

Yolo-v5 was able to not only predict the classes better but was also able to identify small objects which Yolo-v3 seemed to struggle with a little. The improved performance could be primarily attributed to the increased dataset as the number of training images almost doubled.

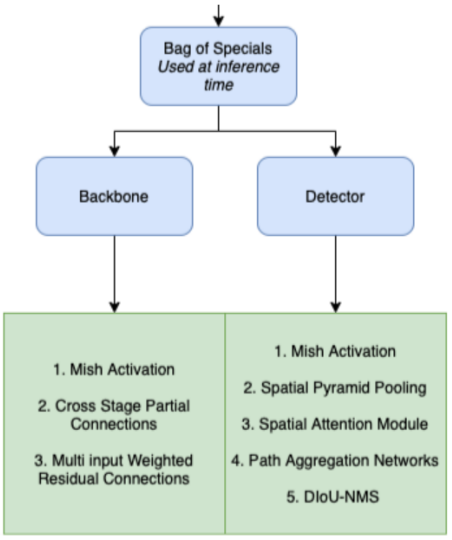


Figure 55. Bag of Specials techniques - Yolo-v4

One of the major improvements after Yolo-v3 that was added in Yolo-v4 was the inclusion of Bag-of-freebies and Bag-of-specials. Bag-of-freebies correspond to the data augmentation techniques(DropOut, DropConnect, DropBlock) that are adapted on the fly by model including but not limited to specialized regularization techniques, various IoU calculations(e.g. distance IoU, generalized IoU). Bag of specials corresponds to the post processing techniques that can increase the performance drastically without much increase in inference time An example of such techniques is illustrated in figure 55 above. In addition, the next version Yolo-v5 has been improved with an additional ability to perform mosaic data augmentation and automatic anchor box calculation. Due to similar architecture to Yolo-v5 and lack of time, it was not hypertuned to a large extent.

Since Yolo-v5 performed better than Yolo-v3, it was used in production during the mobile app deployment stage. Evidence of Yolo-v5's improved performance when compared to Yolo-v3's is illustrated in the figure below.

Figure 56. Comparison of results on Yolo-v3 and Yolo-v5

4.3 Mask R-CNN

4.3.1 Network Training and Hyperparameter Selection

While the new dataset was being collected, MVB dataset was used to perform initial testing on Mask R-CNN. Oversampling dataset in table 2 was used and a validation set consisting of randomly chosen 2,000 images with 1,000 images each of hard and soft baggage was created. SGD was used as the most optimal optimizer in terms of its ability to generalise. Hypertuning performed was in relation to the backbone, training layers and loss adjustment. Once the new dataset was prepared, the tuned hyper parameters were applied on its training.

Backbone:

Since, there were multiple candidates for the backbone, which as mentioned earlier is essentially used for feature extraction through region of interest, Resnet-50 and Resnet-101 were tried. Rest of the hyper parameters were kept constant for fair comparison. Details of these experiments are shown in the table below.

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

Table 9. Validation Loss for different Resnet backbone

Resnet-101 reduced the loss by around 43%. The better performance can be attributed to the deeper layers of Resnet-101, which produced better results than Resnet-50 as it was evident in the experiments done for the classifiers earlier as well. This is due to the more trainable parameters for deeper networks without the loss of generality. More precisely, this dataset was also easier to process for the model due to the nature of the dataset as the localization, which is performed on the basis of the feature extraction by backbone, was quite easier due the highly cropped images in MVB dataset.

Training Layer:

One possible way for dealing with the limitation of MVB dataset is to freeze some of the trainable parameters that were already trained on valid dataset and unfreeze the later layers to finetune these according to the new dataset. Stage 3+ and Stage 4+ correspond to the layers of ResNet which is used in the feature extraction and appears before the ROI classification part where Stage 3+ appears before Stage 4+. Heads refer to the layers responsible for ROI classification and segmentation and bounding box prediction after extracting the region of interest(i.e. all layers in shot-2). Region of Interest are in general the candidates for localized objects but due to the zoomed in nature of MVB, it seemed logical to freeze all the layers up to the Heads because the dataset already mimics this ROIs scenario due to a single entity fully covering the image nature of MVB. Additionally, pre-trained Mask R-CNN on COCO dataset has already learnt to extract features very well and by freezing these layers, the advantage of trained global features could be taken. The later layers focus on the more definitive features of the image because these are more specific to the actual 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+, All**		0.1162

Table 10. 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 then understands 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 initial weights that the model should assign to each of its stages.[21] The following loss weights were customized as follows:

- RPN *class loss*: This corresponds to the loss that is assigned for improper classification of anchor boxes. This must be increased when multiple objects are not detected by the models hence was increased to 1.2 as fewer objects are detected based on our dataset.
- MRCNN *class loss*: This corresponds to the improper classification of the object after detection and must be given higher weightage if the model misclassified a lot of objects. It was increased to 1.5 as objects were observed to be detected correctly but misclassified.
- MRCNN *mask loss* : This corresponds to the pixel level identification of the object and was increased to 1.5 so masks are clearer.
- *Others(RPN bbox loss & MRCNN bbox loss)*: These correspond to the localization accuracy of RPN in the first shot and then MRCNN in the second shot. It was 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 11. Validation Loss for different loss weights.

As shown in table 10, the proposed custom weights helped in reducing the validation loss by around 13% allowing the model to fit better to our current dataset.

4.3.2 Investigation of Performance

As seen in table 12, the F1 score and mAP@[0.5] for the model is quite low on the test set.While it could be attributed to the non-symmetric test and train sets, other reasons must be investigated by visualizing the model's results on the test set.

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

There were certain limitations and certain strengths of the model that were observed after going through the test test as follows:

- Performs segmentation fairly well on the cropped/zoomed-in images as seen in figure below, as these images closely match the MVB dataset on which the model was trained on.

Figure 57. Trained model working well on segmentation of baggage with focused image of baggage. This is similar to the dataset as seen in figure 5.

- Struggles with a background that mimics baggage material, when people are present (see figure 58) and when two or more baggage are present near together (see figure 59).

Figure 58. **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.**

COCO weights were able to segment both the person and the suitcase however, the trained model (with trained heads or trained from Resnet Stage 3 onwards) missed the baggage object and focussed more on the enlarged object and the background in this image. Note that this background is similar to the design pattern of vertical stripes in hard baggage which confuses the model and since most of the ROI is covered by the meta background, model misclassifies it as hard.

Figure 59. 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 59, COCO weights again were able to segment the person as well as the suitcase however, the trained model (with trained heads or trained from Resnet Stage 3 onwards) focuses on both the baggage and the person as one object. Since, persons clothes

mimic a soft material that covered most of the ROI, the classification was incorrectly made as soft.

Figure 60. 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 60, 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 imperative to note that reducing RPN class loss weight did not impact the result seen above.

All of the problems can be attributed to the design MVB dataset. Due to zoomed-in images, even when training heads of the model using the COCO weights, the model focuses on the center of the image and material of the background. Moreover, since the MVB dataset is not much similar to the COCO dataset of suitcase, the full advantage of transfer learning could not be taken. This is particularly the reason for poor results for situations shown in figure 58 and figure 59 A new dataset with clear baggage and different environments was required to help alleviate these problems and is discussed in the next subsection.

4.3.3 Experimentation on Self-Collected Dataset

Results were improved significantly, as seen in table 13 when compared to the statistics shown in table 12, after testing Mask R-CNN on the self collected Baggage Surface-900 dataset. This can be reinforced by the figure 61 as the mask much more accurately covers the baggage.

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

Table 13. Results of training on self-collected dataset

Figure 61. Mask R-CNN's segmentation on test image after training on Baggage Surface-900 dataset

Complex geometric transformation augmentation requires the re-annotation of the image. Doing this manually is extensively time consuming and the implemented model did not have the built-in support for such transformations. Therefore, imgaug[41] library was used for this purpose. According to a study by Google Brain team[42], the following augmentation helped improve the mAP by 3%:

- **Rotation**: Rotation of image by a certain degree. Note that the bounding boxes get larger relative to the object. Vertical and horizontal flipping is also performed in addition to rotation
- **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)

Set	F1-Score	mAP@[0.5]
Train	0.870	0.891

Validation	0.710	0.723
Test	0.770	0.799

Table 14. Results of training on self-collected dataset with augmentations - rotation, equalize, bounding box movement along y-axis

Both F-1 score and mAP increased by at least 2% as seen in the table above. Owing to the improved performance due to increased augmentation, more augmentation was performed to see if performance can be improved. Another experiment was performed with the following augmentations[43]:

- **Cutout**: gaussian channelwise noise added to randomly removed sections of image as seen in figure 62 below.
- **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**
- **Black & White using grayscale**

The addition of these augmentations decreased the scores as shown in table 14 above. Further testing shows that decreased performance was due to the cutout augmnentation which produces data similar to the MVB dataset and upon removing the cutout augmentation, results similar to table 15 could be achieved.

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

Table 15. Results of training on self-collected dataset with additional augmentations - cutout, motion blur, gaussian noise injection and black & white(grey scale)

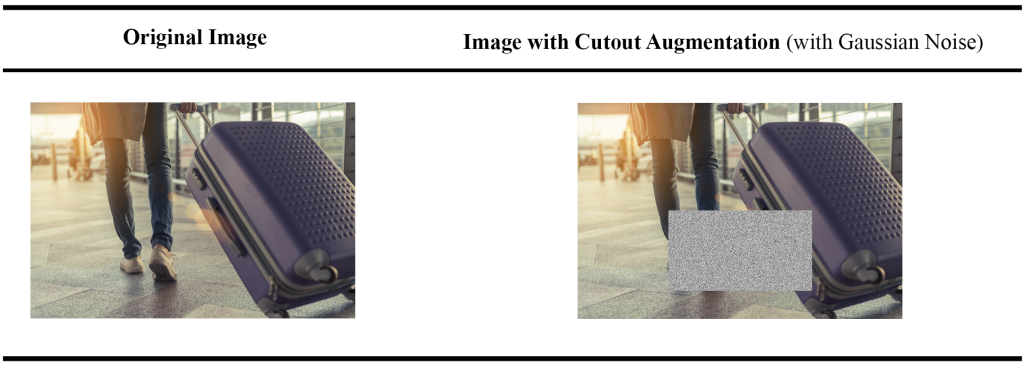


Figure 62. Illustration of cutout augmentation on train images

For the experimentations performed, the learning rate was fixed to 0.01. Although the users of matter plot Mask R-CNN claims that learning rate of 0.01 produces optimal result most of the time and that 0.02 might explode[44] the weights due to slightly different implementation of Mask R-CNN as compared to the one in caffe, the increased learning rate was tried with two experiments done with the following configurations:

- Increased the Learning rate to 0.02 as mentioned in the Mask R-CNN Report[45]. 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.

The results of these experimentations did not produce any significant improvement reinforcing the claims of the users of matter plot Mask R-CNN.

When the Baggage Surface-1700 dataset was created, the maintained environment for Msk R-CNN in the GPU farm, where training was performed, was disrupted due to the upgradation of the farm and the internal libraries.

With some attempts, GPU training was fixed and the model was able to use the GU for training again but the library version issue that was degrading the model's performance still persisted. One potential way to combat the latter issue was to try using the best-performing Mask R-CNN model weights, which produced the results as seen in table 14, to train a new model using these weights in the new environment. This experiment was performed with the base on the head layers for 100 epochs and the results can be seen in the table 16.

Set	F1-Score	mAP@[0.5]
Train	0.781	0.800

Validation	0.768	0.765
Test	0.776	0.792

Table 16. Results of training on the upgraded environment with base on head layers for 100 epochs

The model was also tested, to see how much the performance was affected due to this issue, using the best model weights without any heads training and the 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 17. Results of Model on broken environment without heads training

With the performance again decreased with training, after several trials, more modifications for Mask R-CNN were abandoned and more effort was directed towards the improvement of other models. More detail regarding this issue is discussed in the difficulties encountered section below.

4.4 Comparison of Best Models:

The best performing models for Mask R-CNN, Yolo and the best versions for all the variants of ResNet classifiers, which were deployed on the mobile app, on the test set of their respective dataset are compared in the table below. Note that classifiers were tested on the 64 hard images and the validation accuracy of the classifiers is a reflection of its performance on the cropped dataset.

Model type	Model/s	Modifications from Default	F1-Score	Avg Detection Time (CPU)	mAP@[0.5]

Classifier	ResNet-18,ResNet-34,ResNet-50,ResNet-101, ResNet-152, VGG-16, Custom Built	Augmentation of Random Resized Cropping, Normalized images, Rotation, Flip, Color, Unfreezing train layers, Cost effective weighing for imbalance class dataset	96.6% [validation - cropped], 82% [test - uncropped]	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, RICP, HSV, Unfreezing train layers	-	1.97 seconds	87.2%

Table 18. Comparison of best Models in terms of their performance and modifications

Note that the response time shown in the above table need not be compared with the statistics discussed in section 3.5 because the environments in which the above statistics were collected was different. Most importantly, as mentioned earlier due to AWS educate account limitation, CPU is being used for detention in the deployed backend using the low-tier cloud service. From table 18, Yolo-v5 clearly outperforms Mask R-CNN by 7% and due to its lightweight design as compared to computationally heavy Mask R-CNN, is also observed to be faster than Mask R-CNN. Hence, Yolo-v5 should be prioritized over Mask R-CNN and is recommended if HKAA does not want the passengers to crop images and obtain results faster or if video processing is to be used instead of images, thus providing ease of use and convenience to the customers.

Classifiers on the other hand, proved quite effective in classification. An exceptionally high F1 score of 96.6% F1-score is quite impressive which means it can easily classify 9/10 baggage correctly if a cropped image is fed to the model. The ability of the model to generalize on uncropped images(with random background) is also notable given the fact that it was trained on highly cropped and zoomed in images. By manually cropping using the mobile app, the time taken for detection is saved(as in Yolo-v5 and Mask R-CNN) and

classification step is quite fast with cropping making the overall compelling use case for classifiers.

Nonetheless, with the above statistics and the general comparison drawn in section 3.5, HKAA staff could use the app to test the models and make their own decision on which model to use as the Baggage surface identifier for the Self-service Bag drop system.

4.5 Difficulties Encountered:

Several difficulties were encountered during the implementation and the experimentation of the algorithms outlined in the paper as well as the mobile application. Particularly, it was challenging to optimize the algorithms and hand hyper parameterization was done which is time consuming hence slowed down the project's progress significantly. This is true for both Yolo and Mask R-CNN for which excessive literature was read regarding hyper parameterization technique to improve the performance. Since, automation was not achieved, future work may be carried out in this subject to address this issue.

Likewise, MVB dataset had limitations that further posed difficulties for us to tune the classifiers. It was difficult to solve the overfitting issue as most of the techniques were already exhausted before the final augmentation approaches(Random Resized Cropping) fixed this issue. Multiple Baggage Identities dataset was hand annotated and copyright free images were manually collected, which meant that the pool of images that could be added was limited and was indeed a herculean task to expand this dataset to 2,200 images. Furthermore, finding soft baggage images was challenging when compared to the hard baggage online making it tough to keep the number of samples per class similar. It was discovered that most of the images online had fashion models with intense lighting(as seen in figure 63) in the background but this does not reflect the real scenario when taking images for the application. Hence, more effort was put in digging through these images to find the typical images, which was highly time consuming and demanding thus delayed the overall process.

Figure 63. Intense lighting in images with fashion models

Another issue related to datasets was their reformatting according to the network used. Yolo required a specific formatting of images and their box coordinates, whereas Mask R-CNN required the metadata to be in a specific format different from JSON format that MVB dataset consisted of. 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. Reformatting was necessary for the chosen existing models to obtain the advantage of additional characteristics of these implemented models such as being able to perform automated padding for varying sizes of images, calculating multiple losses (bounding boxes, mask, labels etc.), some non trivial on-the-fly augmentation etc.

Numerous technical difficulties were faced while creating the backend of the mobile app. Initially, it was proposed to use AWS service(AWS SageMaker) specifically designed for implementing and deploying machine learning models. After several hours of effort to migrate the models to cloud and setting up the relevant servers for each model, it was prompted by AWS admin at the execution that AWS educate accounts are not eligible for that service thus several hours of effort was wasted. Similarly, due to fewer credits granted, most of the testing was done on the local device before moving the architecture to cloud which was indeed time consuming as well. Moreover, it was observed during the testing that some of the label boxes on the predicted image by Yolo were too small. For Mask R-CNN, the default

model used a jupyter notebook to show images and build the model after change. Both these changes required gradual understanding of the internal files and making relevant changes with great attention to avoid causing any changes to the calculations of the model and hence prolonged the setup.

Implementing some of the regularization and hyper parameterization techniques were a challenge. One such difficulty was the use of a less-commonly used keras library function called train_on_batch() for classifiers. Normally, other existing functions such as fit() and fit_generator() are used which automate most parts of training. In order to use the cost effective weighing technique for imbalance class dataset as seen in MVB, one technique was to input the overall ratio in fit_generator(). This would precisely not solve the problem however due to different sizes of images fed to the epoch in each batch and given that these batches are selected randomly. In order to calculate the ratios per epoch after randomly choosing the batch, train_on_batch() function was used and the whole training algorithm was written manually.

Some difficulties were encountered while setting up the environment on HKU GPU farm. Since, multiple classifier models were tested, these were trained in parallel to save time. 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. For Mask R-CNN some CUDA and CUNN versions needed to be downgraded in order to support the use of GPU. This became imminent when the model was taking several minutes for a 5 minutes epoch. This was later fixed with attention to detail of the internal HKU GPU farm files.

As mentioned earlier that Mask R-CNN training was halted due to upgradation issues in HKU GPU Farm 2 in March 2021 , some methods were tried to fix the issue. This is because due to the upgrade, the previously built virtual environment was not identifying GPU which meant 2 hour training per epoch. One potential fix was to open the training iPython notebook inside the virtual environment before starting the training, however, this caused the performance across all the metrics to drop by approximately 10% as seen in the table below with all other parameters still the same.

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

Table 19. Results of Mask R-CNN in the broken environment - performance dropped by approximately 10%

It was later realised that the root cause of the problem was the incompatibility of the TensorFlow version of 11.15.0(requirement for Matterport Mask) in the previous environment with the updated CUDA version 11.2. It was noticed that the previous CUDA 10.0 still persisted for legacy software and hence, the GPU training time was fixed. Nonetheless, model training performance was not fixed and still resulted in worse performance than before this issue. Even with changing weight parameters or any augmentation did not have any impact and rather made the performance of the model even worse.

The issue was not even fixed with the clean installation of Matterport Mask R-CNN libraries in a newly created virtual environment. With the time constraints, thereafter more effort was put into the enhancement of SmartPhone application by adding the crop functionality and Yolo's performance by increasing the dataset.

5. Future Work

In view of the major obstacle in being the inability of the the detection and segmentation networks to achieve very high mAP, preferably 95%, and given that most of the techniques to improve the performance have been exhausted and exploited to a vast extent, other existing competitor models using inspiration from other advance ideas must be tried. For Yolo, one such model is Solo which can be seen to tackle some of the issues with Yolo. For example, one of the Yolo's limitation as mentioned in section 3.3.2.1.2 regarding its use of simple non-max suppression algorithm leading to elimination of distinct objects that are put close together or overlaps, could be relieved by Solo, which claims to alleviate this issue without compromising the running time by using a special non-max suppression(NMS)

algorithm called Matrix NMS. Matrix NMS as seen in figure 64 below, is better in terms of average precision and inference time than most other NMS algorithms that exist.

Method	Iter?	Time(ms)	AP
Hard-NMS	✓	9	36.3
Soft-NMS	✓	22	36.5
Fast NMS	✗	< 1	36.2
Matrix NMS	✗	< 1	**36.6**

Figure 64. Comparison of various NMS algorithms[46]

Similarly, since, the deep learning community holds difference in opinion between the performance of Yolo-v4 and Yolo-v5 and there has been some experiments showing situations where Yolo-v4 could perform better, Yolo-v4 must also be tried once to see any improvement in performance if inference time is not a limitation factor.

Likewise, Solo network performs image segmentation and the recent SOLOv2 paper[46] released in October 2020 states that not only SOLOv2 is at least 20% faster model but also performs better in instance segmentation. It was implemented in February of 2021[48] and can be used as a replacement of Mask R-CNN. It can be thought of an intermediate model for both Mask R-CNN and Yolo networks that are tested in this paper.

Moreover, Mask R-CNN can also be thought of a concatenation of another network called Faster R-CNN and Fully Convolutional Network (FCN). Faster R-CNN outputs just the bounding box and class labels whereas FCN component of Mask R-CNN outputs the segmentation mask and has a separate pipeline from box and class prediction as discussed in its architecture. Therefore, it needs to be considered whether the segmentation mask is actually needed and fellow models should be tried to avoid the burden of improving redundant features. Thus, there are other models which can be seen as competitor models with constraints different from Mask R-CNN and Yolo. Keeping in view those constraints and the desired output, such models must also be tested for better comparison.

It is worth noting that Mask R-CNN's testing could not be completed due to breakdown of GPU farm environment owing to the abrupt upgrade. Hence, if Mask R-CNN is to be pursued for this problem, it must be tested on the Baggage Surface-1700 and Baggage Surface-2000 dataset.

In order to make any network more robust against abnormal input, an additional background class should be added apart from hard and soft classes. This is imperative because it is possible for these models to detect a non bag object and by virtue of restricting the class set to just two, the model needs to choose one of the two even if the probabilities for both is lower(class probabilities in Yolo are mutually exclusive as opposed to softmax). This setup penalizes the classification stage of the model by default if the detection is not accurate. Thus the model's classification stage could be used as a validation step by adding a third background class in case the model fails to localize objects well in the initial phase. This is true for both Mask R-CNN and Yolo. This phenomenon is popular in Mask R-CNN design.

Since, building the most optimal backend design architecture was tried but not achieved due to the AWS educate account limitations, after researching the most suitable deployment architecture uses an AWS service called AWS SageMaker[47] that is specifically designed for creating and maintaining ML models on cloud(using jupyter notebooks) as well as deploying these as endpoints. The current architecture is good enough in terms of testing and training the models as there are suitable environments for each model and the processing power could be changed as well but the pricing and other domains in which AWS sagemaker is better are as follows:

- From 2 core and 4GB machines to 96 core and 786GB RAM machine where each machine comes with the environments(eg. tensorflow and related libraries installed) configured
- Link to the git repository such that any changes would be published automatically
- Unlike AWS EC2 instance currently used, charged on the start and stop time basis, the user only gets charged for the time(seconds) during which the model was trained
- Deploy multi model endpoints to host multiple models on the same endpoint with the request load to be managed by AWS SageMaker
- A separate pipeline for logging information to monitor the model and its endpoints

In order to support live video processing on Yolo or Mask R-CNN, these models must be added to the app bundle to avoid the network latency.

Although self-collected dataset was collected manually and was altered to the best possible efforts in terms of fitting the model, data exploration is important as there might be some images in the test set that were too different from yet the ones that model was trained

on. Investigations must be performed to see the images that the model finds troublesome. Adequate testing is also necessary rather than just relying on a small test sample size that was used. This is also true for classifiers where the number of images were just limited to 64. Both the magnitude of the dataset(precisely the images taken at HKAA) and experiments performed must be increased for better analysis and comparison before drawing the final conclusion in terms of the performance. The mobile app can provide a basis for easier testing and monitoring the performance and could also be used as a source of data collection in this regard.

Some special cases must be catered with to avoid model failure in such extreme cases. Two such cases include people putting their handbags/jackets/plastic wrap on top of their luggage and sometimes bags are put on steel trolleys as seen in figure 65. While jackets on hard bags could hinder the view and might confuse the model and result in wrong prediction, steel trolleys can make the model suffer when soft bags are put on it. Although some images have been added to tackle the former case, in order to eradicate this issue to a high degree, substantial amounts of such images depicting the two scenarios must be added to the train set.

Figure 65. MTR Airport Express containing the scenario of a) a person using a trolley on left b) people putting their handbag on their baggage on right.

Likewise, there are certain types of bags(shown in figure 66) and some with different shapes which are also used by the passengers but these were not included in the dataset on which the models outlined in this paper were trained. In order to make the models more scalable, these types of baggage must be added to the dataset.

Figure 66. Special types of bags not included in the dataset - School bags, Suitcases

The project is complete and major components are summarized in the table below. Note that several changes have been done over the course of the project in terms of its scope. In terms of the project schedule set in our detailed project plan (see Section 5.1 below), all the items mentioned upto mid April 2020 have been completed.

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	**Completed**

	• MVB Data preparation and transformation (Augmentation)	
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 • Create 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	**Completeds**
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**
	Preparation for the project exhibition: • Poster • Video	**Complete**

6. Conclusion

This project aims to develop a robust machine learning model application for HKAA to classify the bags as either hard or soft. Three types of networks, simple single entity classifiers(ResNets and VGG), Yolo and Mask R-CNN, were studied in relation to the concerned objective. Due to the highly closeup images as a limitation in the initial secondary source dataset(MVB), while it was suitable for classifiers, another data set was manually collected from the web to allow multiple entities to support training on Yolo-v3 and Mask R-CNN. Overfitting in the classifiers was imminent due to the specificity of MVB dataset, it was yet solved with several advanced data augmentation techniques which have also been useful for Yolo and Mask R-CNN including but not limited to random resized cropping, mixup, hue, saturation and value. Several experiments with hypertuning and regularization techniques with varying configurations have been performed on both Yolo-v3 and Mask R-CNN. Increasing the dataset has shown massive improvement when tested on Yolo-v5 which could be attributed to both the enhancements in Yolo-v5 and additional training volume. The results show that Yolo-v5 is better at generalizing than Mask R-CNN which learns better with high training mAP but relatively struggles at predicting. Although little can be said before Mask R-CNN is fully tested on the increased dataset, Yolo-v5 should be prioritized over Mask R-CNN if inference speed is a bottleneck in the media input processing. For better accuracy on images, Mask R-CNN would be an ideal candidate however. Taking inspiration from the VGG model, the custom model was built from scratch for the two class problem and experimented along with VGG-16. Deep networks, with skip connection technique, in ResNets enable them to learn better than traditional VGGish models and hence were seen to perform better than the custom built model. ResNet-152 performed

the best, achieving an impressive ~96% F-1 score on cropped dataset and ~82% on original uncropped test set, whose images were intentionally kept quite dissimilar from the validation and train set. Smartphone application was developed with the frontend functionality of cropping/editing images for classifier models to make use of the least inference time(0.18s) that classifier offers relative to the other two networks as well as its notable performance on cropped images. Increasing the dataset(both training and testing) and trying competitor models including but not limited to Solov2 and Faster R-CNN, after studying their constraints, should be the next steps before drawing the final interpretation on the best model. Mobile app with frontend implemented in React Native(cross platform - Android & IOS) the backend, consisting of the best performing version of Mask R-CNN and Yolo and the best variants of ResNets, was deployed using AWS EC2, TorchServe and flask. It could be used for visualization and testing, as well as data collection via passengers and is a one step further towards being integrated with the existing HKIA app hence making HKIA a smarter airport.

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] hazelcast. What Is Machine Learning Inference? Retrieved from HazelCast: https://hazelcast.com/glossary/machine-learning-inference/

[26] Blengino, A. (2020, August 20). How to calculate mAP for detection task for the PASCAL VOC Challenge? Retrieved from https://datascience.stackexchange.com/questions/25119/how-to-calculate-map-for-detection-task-for-the-pascal-voc-challenge

[27] GStreamer. gstreamer.freedesktop.org. Retrieved from gstreamer: https://gstreamer.freedesktop.org/features/index.html

[28] Supeshala, C. (2020, August 23). YOLO v4 or YOLO v5 or PP-YOLO? Retrieved from https://towardsdatascience.com/yolo-v4-or-yolo-v5-or-pp-yolo-dad8e40f7109

[29] Alderliesten, K. (2020, May 28). YOLOv3 — Real-time object detection. medium.com. Retrieved from https://medium.com/analytics-vidhya/yolov3-real-time-object-detection-54e69037b6d0

[30] K, S. (2019, October 1). Non-maximum Suppression (NMS). https://towardsdatascience.com/. Retrieved from https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c

[31] Joseph Redmon, A. F. (2018, April 8). https://arxiv.org/. Retrieved from https://arxiv.org/pdf/1804.02767.pdf

[32] TorchServe, PyTorch.. Retrieved from https://pytorch.org/serve/

[33] Research, G. (2017, April 25). Speed/accuracy trade-offs for modern convolutional object detectors. Retrieved from https://arxiv.org/: https://arxiv.org/pdf/1611.10012.pdf

[34] Ryo Takahashi, T. M. (2018). RICAP: Random Image Cropping and Patching. Retrieved from http://proceedings.mlr.press/: http://proceedings.mlr.press/v95/takahashi18a/takahashi18a.pdf

[35] Ryo Takahashi, T. M. (2019, August 27). Data Augmentation using Random Image Cropping. Retrieved from arxiv.org: https://arxiv.org/pdf/1811.09030.pdf

[36] ultralytics. Yolo-V3. Retrieved from https://github.com/ultralytics/yolov3

[37] scikit-learn.org. KMeans. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[38] Hongyi Zhang, M. C.-P. (2018, April 27). mixup: BEYOND EMPIRICAL RISK MINIMIZATION. Retrieved from https://arxiv.org/: https://arxiv.org/pdf/1710.09412v2.pdf

[39] Eun Kyeong Kim, H. L. (2020, May 28). Data Augmentation Method by Applying Color Perturbation of Inverse PSNR and Geometric Transformations for Object Recognition Based on Deep Learning. https://www.mdpi.com/. Retrieved from https://www.mdpi.com/2076-3417/10/11/3755/htm

[40] ultralytics. Yolo-V5. Retrieved from https://github.com/ultralytics/yolov5

[41] imgaug. Retrieved from imgaug.readthedocs.io: https://imgaug.readthedocs.io/en/latest/

[42] Wright, L. (2019, June 28). State of the Art Object Detection — use these top 3 data augmentations and Google Brain's optimal policy for training your architecture. medium.com. Retrieved from https://lessw.medium.com/state-of-the-art-object-detection-use-these-top-3-data-augmentations-and-google-brains-optimal-57ac6d8d1de5

[43] Connor Shorten, T. M. (2019, July 6). A survey on Image Data Augmentation. Retrieved from https://journalofbigdata.springeropen.com/: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0#Sec3

[44] matterport. Learning Rate Decay. Retrieved from https://github.com/matterport/Mask_RCNN/issues/289

[45] Kaiming He, G. G. (2018, January 24). Mask R-CNN. Retrieved from arxiv.org: https://arxiv.org/pdf/1703.06870.pdf

[46] Xinlong Wang, R. Z. (2020, October 23). SOLOv2: Dynamic and Fast Instance Segmentation. Retrieved from arxiv.org: https://arxiv.org/pdf/2003.10152.pdf

[47] Amazon Web Services. Amazon SageMaker. Retrieved from https://aws.amazon.com/sagemaker/

[48] WXinlong. SOLO. Retrieved from https://github.com/WXinlong/SOLO

[49] Google Play Store. HKG My Flight (Official). Retrieved from https://play.google.com/store/apps/details?id=com.hkia.myflight&hl=en_ZA