Mask R-CNN - Pedestrian Detection : Final Report

Anita Chia | Bernadine Lye | Woojin Park
Introduction to Artificial Intelligence, Spring 2020
Heinz College at Carnegie Mellon University

Problem Statement **2**
Importance of Problem 2
Data **2**
Data source 2
Data description 3
Exploratory Data Analysis **3**
Our Approach : Mask R-CNN **5**
Current Approaches vs. Our Model 7
Model Input 7
Model Output 7
Model Evaluation **8**
Loss 8
Manual Inspection 9
Baseline 9
Our Procedure **10**
Results **28**
Definitions 28
Actual vs. Predicted Images 28
Misclassification Examples 33
Summary 34
Comparing our results 34
Reflections **35**
Lesson-Learned 35
Findings 36
Retrospect 37
Next Steps 38
Project Work Responsibilities **39**
Appendix **39**
Exploratory Data Analysis Code 40
MRCNN Code 42

Problem Statement

Autonomous vehicles need to identify human features accurately to ensure safe self-driving in real time. When autonomous vehicles are able to identify human features or even yet, accurately classify the type of feature, it will be able to make better decisions quickly.

But as the types of scenarios of transportation are diversified, we are increasingly in need of more sophisticated human detection techniques applying advanced 'Instance segmentation' deep learning techniques called Mask R-CNN.

Importance of Problem

On March 18, 2018, the first pedestrian fatality because of a self-driving car was recorded in Arizona.[1] The autonomous vehicle was operated by Uber and had an emergency human driver behind the wheel. Both the vehicle and the driver did not notice a pedestrian jaywalking with her bicycle. This incident shocked many across the world and shattered the public's trust in self-driving vehicles. It also resulted in an issue of liability - who is to blame for the accident?

Being able to accurately identify partially-visible or visually-obstructed people can aid in more accurate human detection and potentially avoid any future fatalities. We hope to increase safety for both drivers and pedestrians and reduce liability for autonomous vehicle manufacturers. A stronger ability to detect humans may also improve the public's trust in autonomous vehicles.

Data

Data source

We utilize the WiderPerson dataset[2], which provides a large number of highly diverse and dense bounding box annotations for pedestrian detection.

Data description

We have a total of 13,382 image dataset with 9,000 images annotated, and a total of 399,786 annotations *i.e.*, 44.42 annotations per image.Our dataset contains dense human features with various kinds of occlusions such as pedestrians, riders, partially-visible persons, ignoring regions and crowds.

---

[1] Schmelzer, R. (2019, September 26). What Happens When Self-Driving Cars Kill People? Retrieved May 05, 2020, from https://www.forbes.com/sites/cognitiveworld/2019/09/26/what-happens-with-self-driving-cars-kill-people/

[2] WiderPerson: A Diverse Dataset for Dense Pedestrian Detection in the Wild. Retrieved from http://www.cbsr.ia.ac.cn/users/sfzhang/WiderPerson/

Exploratory Data Analysis

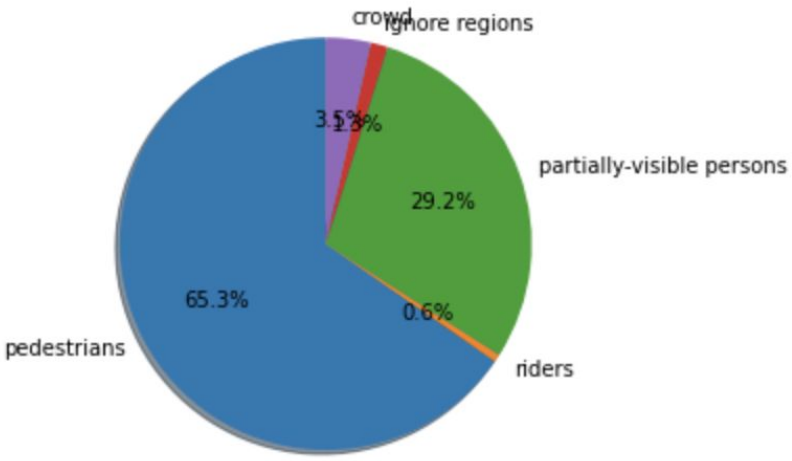


Fig.1.1

Based on our exploratory data analysis, we noticed that there is a sharp imbalance of classes. This is seen in Fig 1.1 where slightly more than 65% of class annotations are classified as pedestrians.

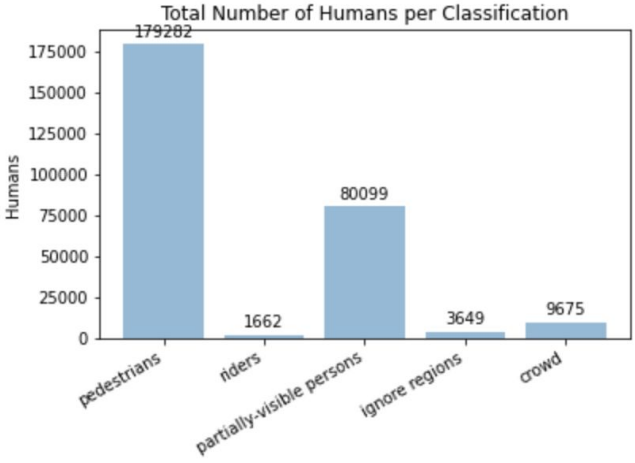


Fig 1.2

Upon further exploration, we've found that there are at least 1500 data points for each class, with riders being the smallest class followed by ignore-regions at 3649. There is on average, 0.18 and 0.4 ignore regions per image as seen in Fig 1.3. We also noted that there is a high density of pedestrian and partially-visible persons at 19.07 and 8.88 respectively per image. We were worried that an imbalance of classes may affect our model's ability to carry out multi-class classification effectively.

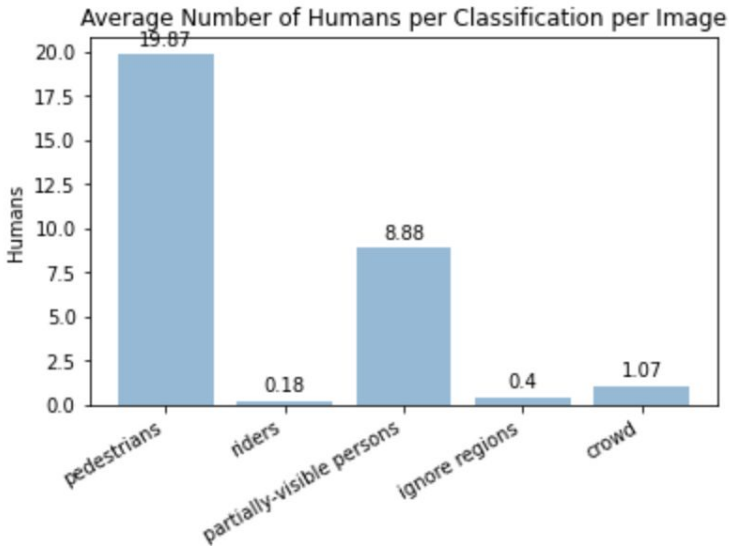


Fig 1.3

Once we've taken a look at a sample of 30 images, we noticed that most images do not contain ignore-regions or riders specifically. We've also noted that images may not be annotated accurately. For example, an image with a lot of ignore-regions may not have the relevant bounding boxes to represent it. Similarly, we noted that some pedestrians or partially-visible persons were not captured in the annotations as seen in Fig 1.4.

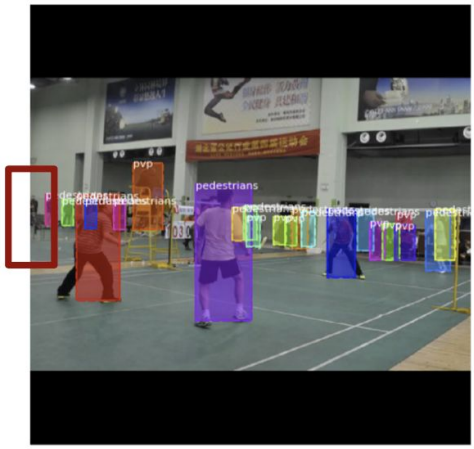


Fig 1.4

With a severely imbalanced dataset, we were considering an under-sampling of over represented classes. By doing so, we can balance out the number of records for each class. But as seen in Fig 1.2 and Fig 1.3, the imbalance between the largest and smallest represented class is huge, with a difference of over 100,000 records. It did not make sense for us to remove such a large amount of valuable records just for the sake of having balanced classes.

After much deliberation, we prioritized that our model should identify all humanoid figures within an image, regardless of class assigned to each record. It is important for autonomous vehicles

to be able to identify and avoid human features while driving. Accurate classification of the human feature would definitely be a bonus that we can consider in future.

Our Approach : Mask R-CNN

We apply a general framework for object ‘Instance Segmentation’ and the model we implement in this project is called ‘Mask R-CNN’. Our approach efficiently detects objects in an image while simultaneously generating a high-quality segmentation mask for each instance. Instance segmentation is challenging because it requires the correct detection of all objects in an image while also precisely segmenting each instance. It therefore combines elements from the classical computer vision tasks of object detection, where the goal is to classify individual objects and localize each using a bounding box, and semantic segmentation, where the goal is to classify each pixel into a fixed set of categories without differentiating object instances.

When we first started reviewing the computer vision technology to do our pedestrian detection project, we found out that there are currently two of the most ‘state-of-art’ and ‘popular’ algorithms called ‘Faster R-CNN’ and ‘Mask R-CNN’. In our initial research, we focused on understanding the technical differences between these two algorithms and decided to choose ‘Mask R-CNN’ which is a more sophisticated and advanced algorithm.

First, Mask R-CNN efficiently detects objects in an image while simultaneously generating a high-quality segmentation mask for each instance.

The method, called Mask R-CNN, extends Faster R-CNN by adding a branch for predicting

1) ‘Object mask’ in parallel with the existing branch for
2) ‘Bounding box’ recognition and the
3) ‘Confidence score’ which is the probability that an anchor box contains an object.

More in detail, our model calculates the ‘Confidence score’ by applying ‘Intersection over Union (IOU)’ calculation defined as the area of the intersection divided by the area of the union of a predicted bounding box(Bp) and a ground-truth box(Bgt)

$$IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})}$$

Both confidence score and IoU are basically used as the criteria that determine whether a detection is a true positive or a false positive.

The detection is considered a 'true positive (TP)' only if it satisfies three conditions[3]:

1. Confidence score > Threshold
2. Satisfies the predicted class matches the class of a ground truth
3. The predicted bounding box has an IoU greater than a threshold (e.g., 0.5) with the ground-truth.

Second, our learning algorithm Mask R-CNN enables pixel-to-pixel alignment to have high-quality segmentation masks for each instance. On the other hand, Faster R- CNN generates the bounding box as an output of object detection and was not designed to capture the pixel-to-pixel alignment.

The general steps the approach follows[4] is depicted below:

1. **Backbone model:** a standard convolutional neural network that serves as a feature extractor. For example, it will turn a1024x1024x3 image into a 32x32x2048 feature map that serves as input for the next layers.

2. **Region Proposal Network (RPN):** Using regions defined with as many as 200K anchor boxes, the RPN scans each region and predicts whether or not an object is present. One of the great advantages of the RPN is that it does not scan the actual image, the network scans the feature map, making it much faster.

3. **Region of Interest Classification and Bounding Box:** In this step the algorithm takes the regions of interest proposed by the RPN as inputs and outputs a classification (softmax) and a bounding box (regressor).

4. **Segmentation Masks:** In the final step, the algorithm the positive ROI regions are taken in as inputs and 28x28 pixel masks with float values are generated as outputs for the objects. During inference, these masks are scaled up.

---

[3] (2018). *An Introduction to Evaluation Metrics for Object Detection.* NICKZENG. Retrieved from https://blog.zenggyu.com/en/post/2018-12-16/an-introduction-to-evaluation-metrics-for-object-detection/

[4] Garza, G. (2019). *Mask R-CNN for Ship detection & Segmentation*. Towards Data Science. Retrieved from https://towardsdatascience.com/mask-r-cnn-for-ship-detection-segmentation-a1108b5a083

Current Approaches vs. Our Model

Comparing the Faster R-CNN to Mask R-CNN model's main work procedure, firstly,, Faster R-CNN's image detection approaches consist of two main stages. The first stage, called a Region Proposal Network (RPN), proposes candidate object bounding boxes. The second stage, which is in essence Fast R-CNN, extracts features using RoIPool from each candidate box and performs classification and bounding-box regression[5].

Our model Mask R-CNN adopts the same two-stage procedure, with an identical first stage (which is RPN). The first stage, called a Region Proposal Network (RPN), proposes candidate object bounding boxes and then projects these values into the original image once again to draw anchor boxes.

After projection Mask R-CNN applies 'Non-max-suppression' to delete all other anchor boxes except the highest score anchor box. Finally the model uses ROI align to re-scaling these anchor boxes and pass the anchor box through 'Classification', 'Bbox regression' and 'Mask' branches respectively.

Model Input

The WiderPerson dataset is a pedestrian detection benchmark dataset in the wild, of which images are selected from a wide range of scenarios, no longer limited to the traffic scenario.
The input for our project is 'image' and corresponding 'annotation' text file.
We have an object instance per row which consists of [class_label, x1, y1, x2, y2], and we have a total of 5 classes in the entire dataset.

The class label definition is:

class_label =1: pedestrians
class_label =2: riders
class_label =3: partially-visible persons
class_label =4: ignore regions
class_label =5: crowd

Model Output

Our model generates bounding boxes and segmentation masks for each instance of an object in the image after training. Therefore we will have a model validation output as a bounding box, class label, segmentation mask and detection confidence level per each object in the picture. Our model also outputs the loss for each epoch of a training run.

---

[5] He, K., Gkioxari, G., Dollar, P. & Girshick, R. (2018). Mask R-CNN. Retrieved from https://arxiv.org/pdf/1703.06870.pdf

Model Evaluation

Loss

The MRCNN model is accompanied with evaluation metrics that can be used to evaluate our model. Our key evaluation metrics are the training and validation loss of the model. Various types of losses (Classification, Mask, Bounding Box) were considered in computing the overall loss. Our main criteria for success was thus to optimise loss. The lower the loss, the better the model

Classification Loss

Classification loss values are dependent on the confidence score of the true class. Hence, it reflects how confident the model is in prediction class labels/ how close the model is to predicting the correct class. For the MRCNN model, there are 2 types of classification losses metrics.

- **mrcnn_class_loss:** It covers all the object classes and evaluates how well the Mask RCNN recognises each class of object.
- **rpn_class_loss:** How well the Region Proposal Network separates background with objects

Bounding Box Loss

Bounding Box loss values reflect the distance between the true box parameters (x/y coordinates, width/height) and the predicted ones. It is a regression loss that penalizes larger absolute differences.

- **rpn_bbox_loss:** How good the model is at locating objects within the image
- **mrcnn_bbox_loss:** How good the model is at precisely predicting the area(s) within an image corresponding to the different objects that are present

Mask Loss

Mask loss penalises wrong per-pixel binary classifications (foreground/background), with respect to the true class label, and is calculated differently for each of the regions of interest. MRCNN encodes a binary mask per class for each of the RoIs, and the mask loss for the specific RoI is calculated based only on the mask corresponding to its true class, which prevents the mask loss from being affected by class predictions. For the MRCNN model, the evaluation metric for mask loss is “mrcnn_mask_loss”.

- **mrcnn_mask_loss :** How well the model segment objects.

Note: The definitions for these losses, with respect to the Mask R-CNN algorithm, gathered references, including code comments[6], stack overflow[7] and the original Mask R-CNN paper[8].

Manual Inspection

We also did manual inspection to determine our model's performance across each training run. We manually inspected the actual and predicted bounding boxes, masks and labels on the images (Fig 1).

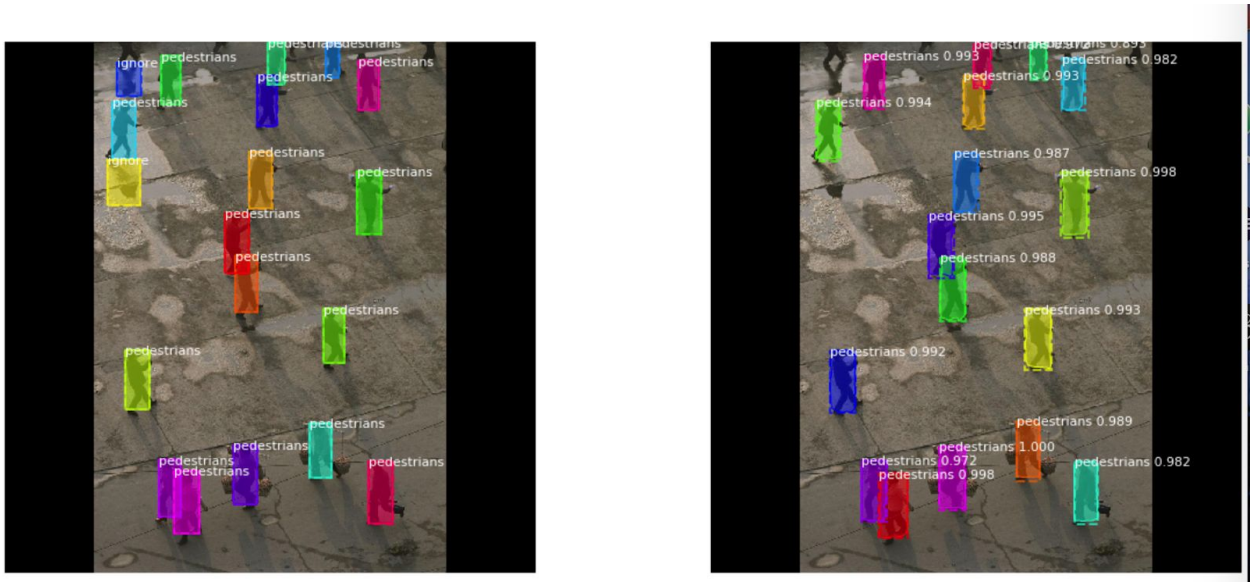


Fig 1: Actual (Left) vs. Predicted Images (Right)

Baseline

We consider the Faster R-CNN model as our baseline. That is, the technical improvement of image classification from the Faster R-CNN model, and the ability of the Mask R-CNN model to locate the exact pixels of each image (vs. just the bounding box).
We also consider the loss improvement across training from losses in smaller datasets in our discovery and training phases as our baseline. We also consider the loss across epochs within a training run. We consider loss improvements across training and validation steps. The smaller the losses obtained, the better the model.

---

[6] (2019). Mask_RCNN Model Github Code. Retrieved from: https://github.com/matterport/Mask_RCNN/blob/master/mrcnn/model.py
[7] Sizigia. (2019). *What exactly are the losses in Matterport Mask-R-CNN?* Stack Overflow. Retrieved from: https://stackoverflow.com/questions/55360262/what-exactly-are-the-losses-in-matterport-mask-r-cnn
[8] He, K., Gkioxari, G., Dollar, P. & Girshick, R. (2018). Mask R-CNN. Retrieved from https://arxiv.org/pdf/1703.06870.pdf

Our Procedure

Before starting our training runs on AWS, we ran a few training runs on our local computers. This was our primary discovery stage of the model - which allowed us to better understand how the model parameters affected the model. We tuned model parameters on a small subset of about 200 pictures to get a gauge on the appropriate learning rate for the model,

Training Run: 1

In our first training run, we trained a model on a smaller subset of the images - 1500 training, 500 validation, and a smaller number of epochs - 5.

Model Parameters

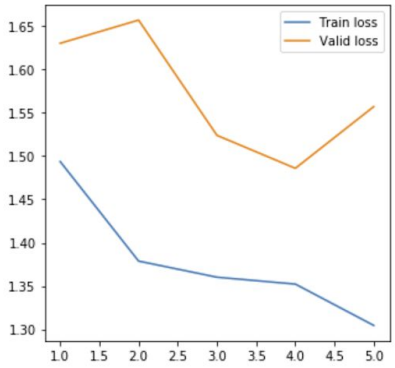
Parameters	# Number of training steps per epoch
	STEPS_PER_EPOCH = 1500
	VALIDATION_STEPS = 500
	
	# Learning rate
	LEARNING_RATE=0.001
	
	# Skip detections with < 90% confidence
	DETECTION_MIN_CONFIDENCE = 0.7

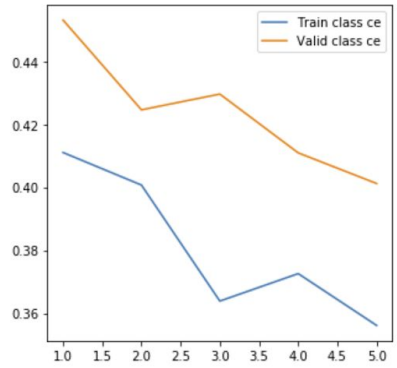
# Epoch	5

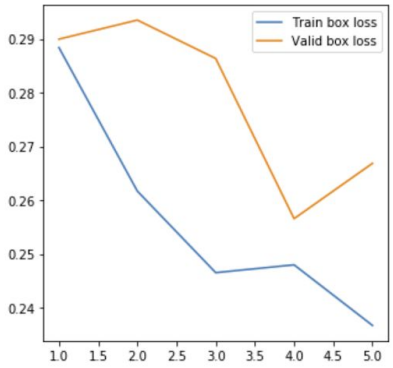
Model Evaluation

	val_loss	val_rpn_class_loss	val_rpn_bbox_loss	val_mrcnn_class_loss	val_mrcnn_bbox_loss	val_mrcnn_mask_loss	loss
1	1.629824	0.144842	0.407264	0.453233	0.289963	0.334520	1.493486
2	1.656550	0.137521	0.479651	0.424682	0.293509	0.321185	1.378811
3	1.523703	0.115011	0.365398	0.429692	0.286373	0.327227	1.360232
4	1.485726	0.109323	0.391793	0.410987	0.256583	0.317038	1.352294
5	1.556819	0.125472	0.444537	0.401243	0.266867	0.318697	1.304669

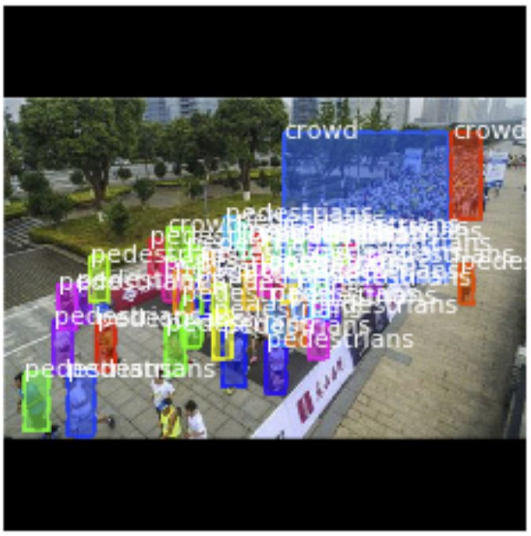
rpn_class_loss	rpn_bbox_loss	mrcnn_class_loss	mrcnn_bbox_loss	mrcnn_mask_loss
0.089085	0.365681	0.411098	0.288403	0.339217
0.069863	0.322737	0.400749	0.261713	0.323747
0.080839	0.353801	0.363836	0.246540	0.315215
0.071855	0.344774	0.372570	0.247994	0.315099
0.072063	0.329096	0.356091	0.236728	0.310688







Manual Inspection Sample Images (Actual vs. Predicted)





Parameter Tuning for the next training run

We observed that the training loss seems to continue on a downward trend after Epoch 5. We also looked at the detection confidence level for the instances in each image and realised a few of the numbers were in the low 0.7s. Hence, in our next training run, we wanted to test if lowering the detection confidence level would improve our results.

Training Run: 2

In our second training run, we wanted to test if lowering the detection confidence level would improve our results. Hence, only the detection confidence level was changed for this training run.

Model Parameters

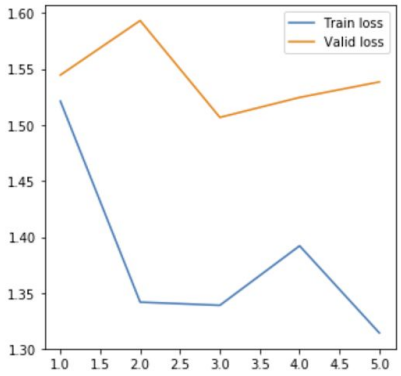
Parameters	# Number of training steps per epoch
	STEPS_PER_EPOCH = 1500
	VALIDATION_STEPS = 500
	
	# Learning rate
	LEARNING_RATE=0.001
	
	# Skip detections with < 90% confidence
	DETECTION_MIN_CONFIDENCE = 0.6

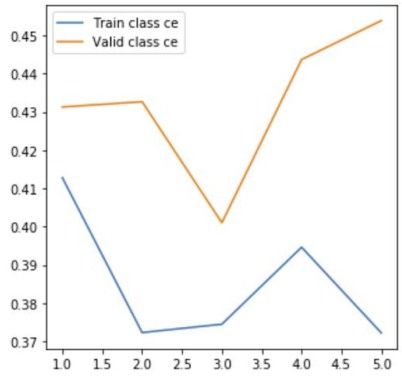
# Epoch	5

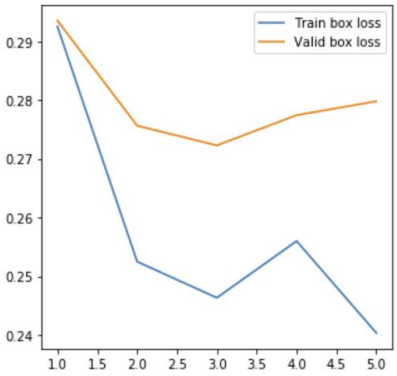
Model Evaluation

	val_loss	val_rpn_class_loss	val_rpn_bbox_loss	val_mrcnn_class_loss	val_mrcnn_bbox_loss	val_mrcnn_mask_loss	loss
1	1.544513	0.108856	0.388183	0.431258	0.293557	0.322657	1.521179
2	1.593065	0.131596	0.434043	0.432636	0.275653	0.319134	1.341884
3	1.506710	0.119779	0.389085	0.401065	0.272299	0.324480	1.339058
4	1.524527	0.102433	0.382467	0.443639	0.277449	0.318537	1.392115
5	1.538347	0.123936	0.350959	0.453807	0.279811	0.329831	1.314397

rpn_class_loss	rpn_bbox_loss	mrcnn_class_loss	mrcnn_bbox_loss	mrcnn_mask_loss
0.094445	0.381284	0.412788	0.292562	0.340098
0.076500	0.321728	0.372396	0.252504	0.318754
0.073741	0.331617	0.374575	0.246347	0.312775
0.075802	0.350140	0.394666	0.256011	0.315493
0.067628	0.322457	0.372298	0.240359	0.311653



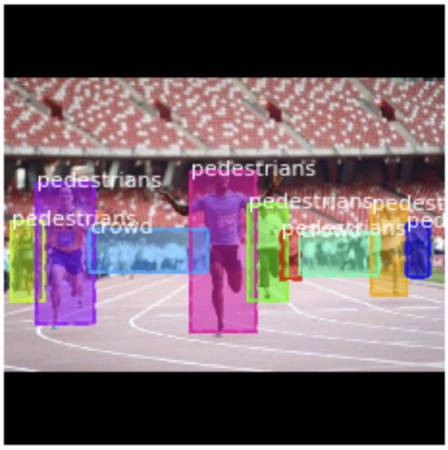


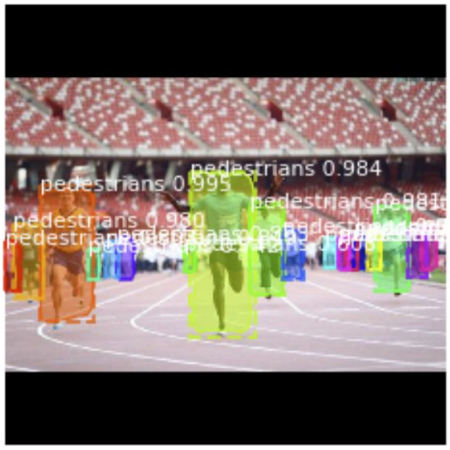


Manual Inspection Sample Images (Actual vs. Predicted)











Predicted Image

Parameter Tuning for the next training run

We observed that the training loss also seems to continue on a downward trend after Epoch 5. From the evaluation results, lowering the minimum detection confidence level caused our model to have a poorer performance with a higher minimum training loss of 1.314 vs. 1.304 (run 1) and a higher minimum validation loss of 1.506 vs. 1.485 (run 1). However, from manual inspection of the individual images detection confidence levels for detected instances, we realised that a few of the correctly-labelled instances lied between the 0.64 to 0.7 range. Hence, in order not to miss the results in this range of confidence levels, we decided on a minimum detection confidence level of 0.65.

Training Run: 3

In training run 3, our objective was to test if increasing the number of training (STEPS_PER_EPOCH) and validation data sets (VALIDATION_STEPS) would increase the model's performance. Hence, keeping minimum detection confidence at 0.6, we increased the number of training data to 5000 and validation data to 1500.

Model Parameters

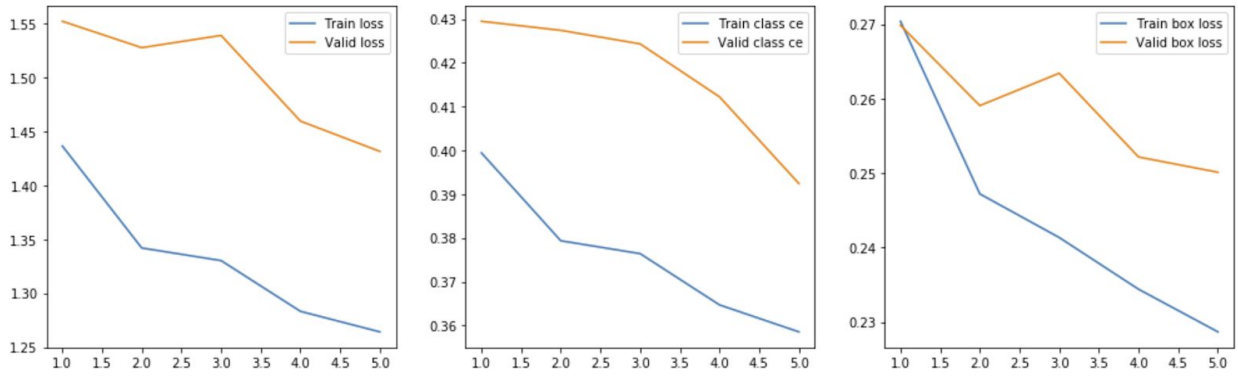
Parameters	# Number of training steps per epoch
	STEPS_PER_EPOCH = 5000
	VALIDATION_STEPS = 1500
	
	# Learning rate
	LEARNING_RATE=0.001
	
	# Skip detections with < 90% confidence
	DETECTION_MIN_CONFIDENCE = 0.6

# Epoch	5

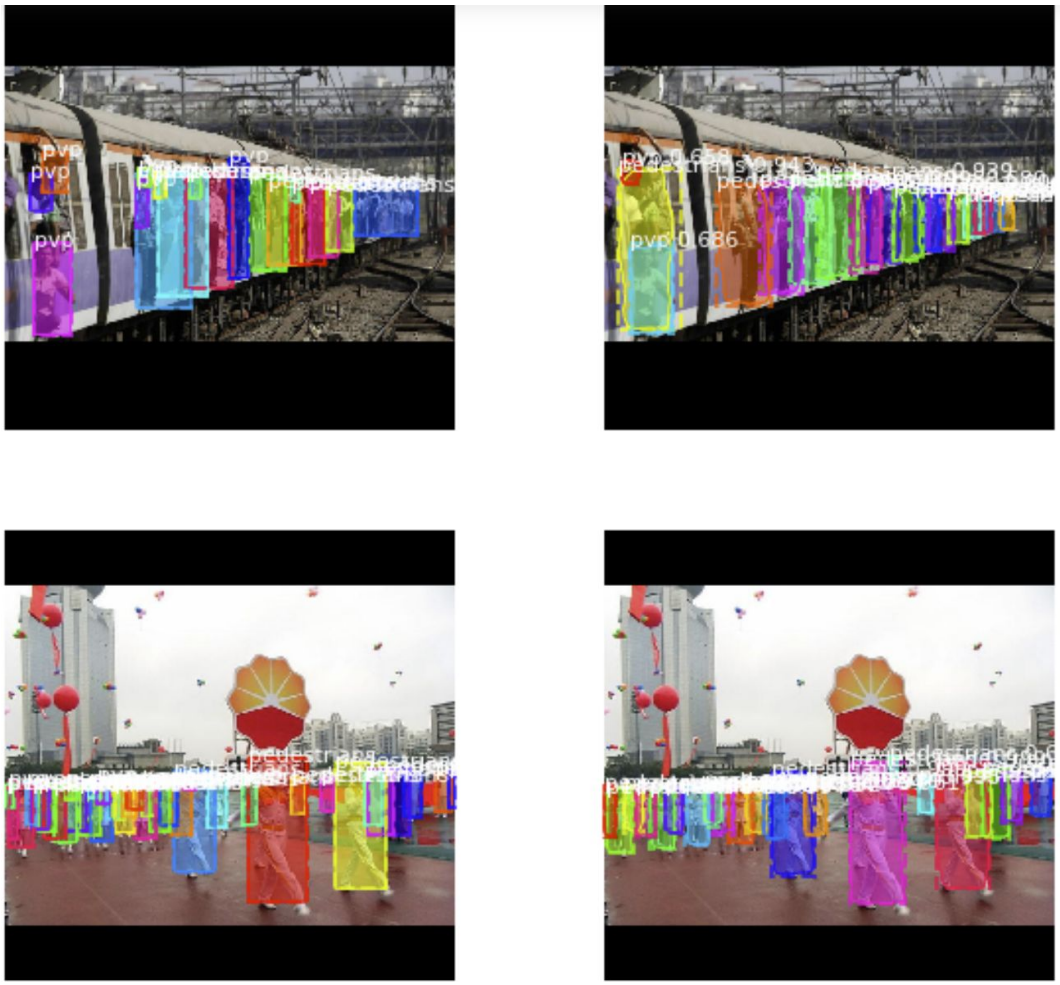
Model Evaluation

	val_loss	val_rpn_class_loss	val_rpn_bbox_loss	val_mrcnn_class_loss	val_mrcnn_bbox_loss	val_mrcnn_mask_loss	loss
1	1.551978	0.131706	0.399328	0.429504	0.269828	0.321610	1.436400
2	1.527538	0.113694	0.404254	0.427434	0.259057	0.323095	1.341978
3	1.538877	0.131849	0.399971	0.424334	0.263399	0.319318	1.330303
4	1.459498	0.103284	0.384316	0.412238	0.252163	0.307488	1.283290
5	1.431461	0.099422	0.374480	0.392424	0.250113	0.315013	1.264275

rpn_class_loss	rpn_bbox_loss	mrcnn_class_loss	mrcnn_bbox_loss	mrcnn_mask_loss
0.083007	0.355981	0.399428	0.270369	0.327614
0.075572	0.328135	0.379351	0.247211	0.311705
0.073411	0.331637	0.376405	0.241361	0.307484
0.065888	0.315623	0.364692	0.234422	0.302658
0.069477	0.306707	0.358530	0.228701	0.300853



Manual Inspection Sample Images (Actual vs. Predicted)





Predicted Image: Could detect “crowd”

Parameter Tuning for the next training run

Increasing the data set numbers definitely did decrease the overall performance of the model. Our model managed to obtain lower validation and training losses overall. Furthermore, we noticed that with 5 epochs, the loss of the model still continues on a downward trend. Hence, in our next training run, we wanted to see if increasing the number of epochs would improve the model's performance.

Training Run: 4

Deciding on our ideal minimum detection confidence level of 0.65 in Training Run 2, we used it in this training run with 8 epochs.

Model Parameters

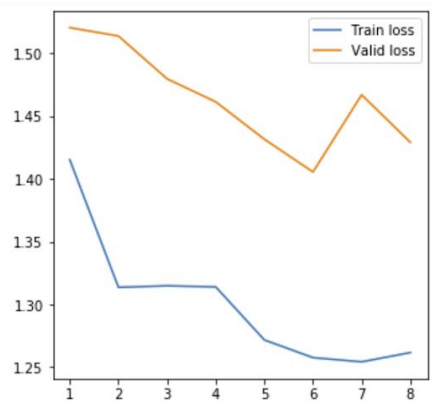
Parameters	# Number of training steps per epoch
	STEPS_PER_EPOCH = 5000
	VALIDATION_STEPS = 1500
	
	# Learning rate
	LEARNING_RATE=0.001
	
	# Skip detections with < 90% confidence
	DETECTION_MIN_CONFIDENCE = 0.65

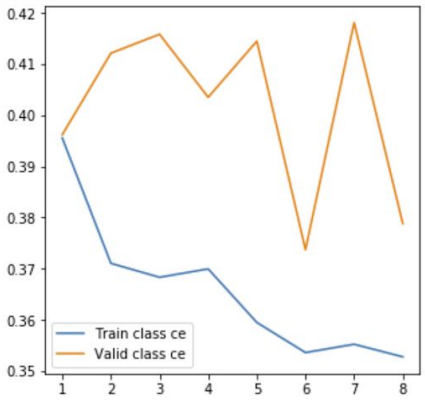
# Epoch	8

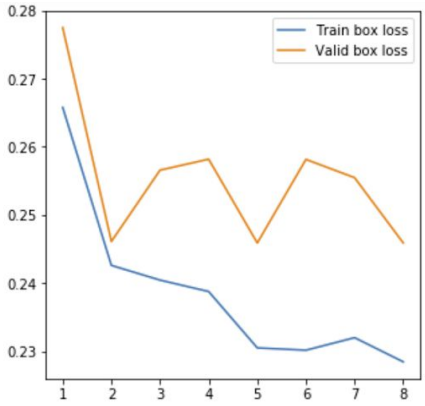
Model Evaluation

	val_loss	val_rpn_class_loss	val_rpn_bbox_loss	val_mrcnn_class_loss	val_mrcnn_bbox_loss	val_mrcnn_mask_loss	loss
1	1.520535	0.118176	0.398359	0.396175	0.277486	0.330337	1.415239
2	1.513870	0.117438	0.425717	0.412095	0.246108	0.312509	1.313626
3	1.479754	0.095153	0.401611	0.415787	0.256576	0.310621	1.314840
4	1.461439	0.100921	0.383402	0.403484	0.258187	0.315437	1.313974
5	1.431629	0.093178	0.370319	0.414444	0.245893	0.307785	1.271689
6	1.405552	0.096232	0.371222	0.373711	0.258156	0.306221	1.257595
7	1.467014	0.115602	0.371440	0.418055	0.255473	0.306432	1.254266
8	1.429199	0.112758	0.386190	0.378824	0.245900	0.305512	1.261623

rpn_class_loss	rpn_bbox_loss	mrcnn_class_loss	mrcnn_bbox_loss	mrcnn_mask_loss
0.078403	0.348945	0.395521	0.265778	0.326591
0.070268	0.320813	0.371041	0.242604	0.308897
0.073115	0.325737	0.368320	0.240446	0.307218
0.072108	0.326769	0.369957	0.238768	0.306365
0.066470	0.313208	0.359496	0.230499	0.302007
0.066044	0.307265	0.353613	0.230157	0.300505
0.064587	0.302476	0.355241	0.231998	0.299952
0.064115	0.318032	0.352784	0.228451	0.298227



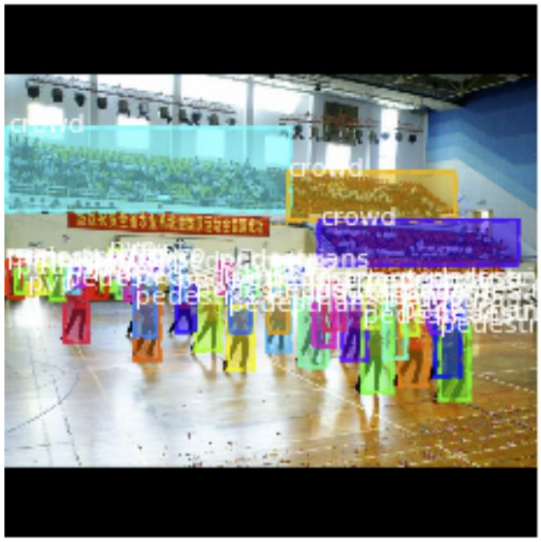




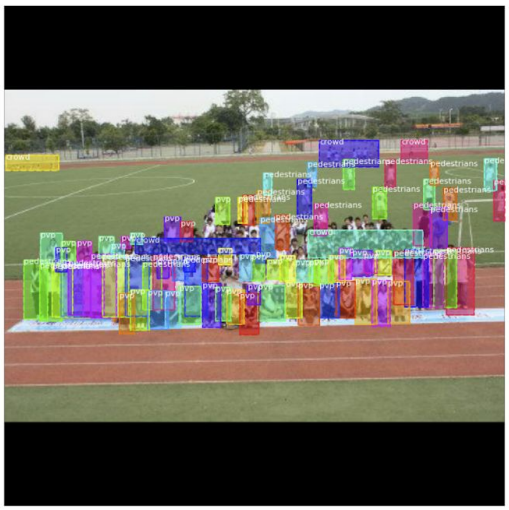
Manual Inspection Sample Images (Actual vs. Predicted)













Parameter Tuning for the next training run

Increasing the number of epochs did allow us to get better (lower) validation and training losses than the previous runs. Hence, we wanted to try to increase the number of epochs to investigate how the losses vary across epochs.

Training Run: 5

In this training run, we hoped to investigate the effect of the number of epochs on the losses obtained. To do so, we used a smaller dataset (200 training, 50 validation) with a larger number of epochs.

Model Parameters

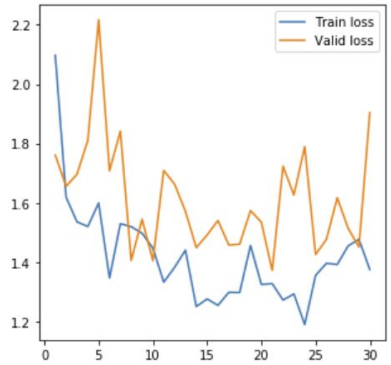
Parameters	# Number of training steps per epoch
	STEPS_PER_EPOCH = 200
	VALIDATION_STEPS = 50
	
	# Learning rate
	LEARNING_RATE=0.001
	
	# Skip detections with < 90% confidence
	DETECTION_MIN_CONFIDENCE = 0.65

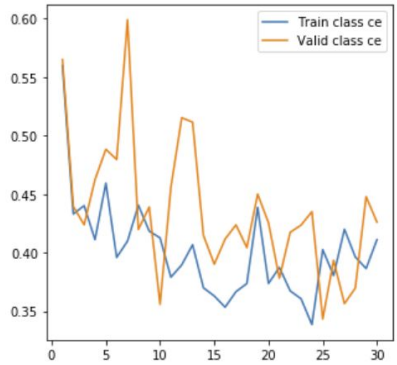
# Epoch	30

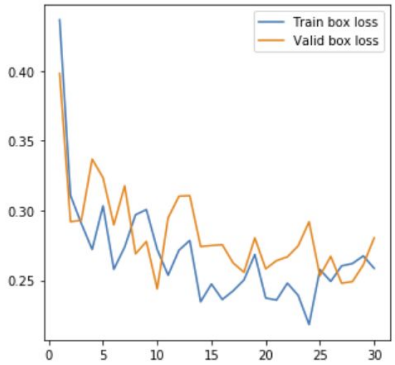
Model Evaluation

	val_loss	val_rpn_class_loss	val_rpn_bbox_loss	val_mrcnn_class_loss	val_mrcnn_bbox_loss	val_mrcnn_mask_loss	loss
1	1.760751	0.083208	0.376438	0.565045	0.398272	0.337787	2.096595
2	1.656365	0.162479	0.405763	0.439602	0.292004	0.356516	1.619460
3	1.696741	0.178114	0.473535	0.423944	0.292934	0.328211	1.536896
4	1.810716	0.089414	0.553781	0.462479	0.336827	0.368214	1.521075
5	2.216305	0.362677	0.702477	0.488405	0.323552	0.339193	1.600308
6	1.708241	0.074193	0.517036	0.479674	0.289739	0.347597	1.347874
7	1.841844	0.189738	0.409961	0.599015	0.317665	0.325463	1.530723
8	1.406996	0.099314	0.307855	0.419987	0.268921	0.310918	1.520570
9	1.545589	0.084509	0.418528	0.439111	0.277772	0.325667	1.498594
10	1.406436	0.078517	0.410948	0.356036	0.243754	0.317179	1.448426
11	1.709900	0.115796	0.514123	0.456022	0.294642	0.329315	1.334059
12	1.663577	0.096991	0.395815	0.515370	0.310208	0.345192	1.383923
13	1.572500	0.082269	0.324342	0.511556	0.310612	0.343718	1.441778
14	1.449968	0.111614	0.338656	0.414866	0.274180	0.310651	1.251977
15	1.492683	0.118369	0.402998	0.390290	0.274898	0.306127	1.277502
16	1.541559	0.087271	0.454334	0.411798	0.275413	0.312741	1.256158
17	1.458917	0.072479	0.372684	0.423850	0.262367	0.327535	1.299780
18	1.461707	0.100738	0.391467	0.404454	0.255720	0.309327	1.299375
19	1.574949	0.156989	0.370025	0.450122	0.280346	0.317464	1.457135
20	1.535699	0.151097	0.389306	0.425580	0.258183	0.311531	1.326694
21	1.373558	0.074702	0.335667	0.378249	0.264064	0.320874	1.329151
22	1.723790	0.279735	0.450607	0.417526	0.266725	0.309194	1.273860
23	1.627096	0.117949	0.477507	0.423649	0.274654	0.333335	1.294493
24	1.789874	0.142874	0.584602	0.435136	0.291889	0.335371	1.191501
25	1.426990	0.104172	0.416323	0.343490	0.253193	0.309810	1.356633
26	1.477909	0.120890	0.389472	0.393624	0.267140	0.306781	1.397499
27	1.617862	0.144886	0.547765	0.356600	0.247846	0.320762	1.393598
28	1.515251	0.113638	0.469515	0.369828	0.249021	0.313247	1.455890
29	1.452287	0.073242	0.358436	0.447949	0.261028	0.311630	1.478078
30	1.904068	0.314457	0.552078	0.426163	0.280321	0.331046	1.376346

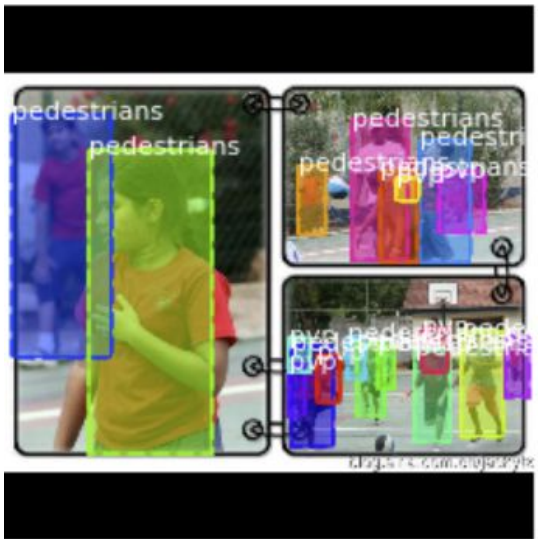
rpn_class_loss	rpn_bbox_loss	mrcnn_class_loss	mrcnn_bbox_loss	mrcnn_mask_loss
0.140923	0.523282	0.559917	0.436719	0.435753
0.104882	0.422771	0.433103	0.310866	0.347836
0.090989	0.372907	0.440299	0.290672	0.342027
0.088041	0.414767	0.411255	0.272054	0.334957
0.092583	0.389475	0.459514	0.303286	0.355448
0.070537	0.302399	0.396063	0.257830	0.321044
0.096994	0.415358	0.410031	0.273839	0.334499
0.089309	0.355548	0.440658	0.296934	0.338120
0.090106	0.345984	0.418530	0.300629	0.343344
0.067594	0.369826	0.412734	0.272040	0.326230
0.068924	0.316191	0.379174	0.253573	0.316196
0.074581	0.321360	0.389699	0.271498	0.326783
0.075934	0.354111	0.406889	0.278523	0.326320
0.052459	0.287544	0.370092	0.234539	0.307342
0.062370	0.291821	0.363110	0.247281	0.312919
0.068670	0.285706	0.353563	0.236170	0.312048
0.070982	0.304636	0.366804	0.242389	0.314968
0.066090	0.293638	0.373732	0.250227	0.315687
0.072733	0.348538	0.438802	0.268443	0.328618
0.075126	0.333595	0.373804	0.237240	0.306928
0.071227	0.323414	0.387528	0.235784	0.311196
0.062659	0.282256	0.367454	0.247908	0.313582
0.057896	0.321968	0.360755	0.239090	0.314781
0.073100	0.263004	0.338721	0.218215	0.298459
0.063125	0.310189	0.402697	0.257649	0.322971
0.080056	0.370162	0.380556	0.249141	0.317583
0.069358	0.323243	0.420091	0.260285	0.320618
0.089473	0.385297	0.396478	0.261972	0.322668
0.082105	0.423722	0.386558	0.267414	0.318276
0.074987	0.315766	0.411232	0.258517	0.315841







Manual Inspection Sample Images (Actual vs. Predicted)









Parameter Tuning for the next training run

We were unsure if this smaller dataset could be a good gauge for the optimal number of epochs, but could derive a generalised conclusion that we needed to increase the number of epochs to obtain a better model.

Training Run: 6

In our final training run, after obtaining our ideal configuration parameters, we wanted to get the optimal number of epochs such that we were not under-training or over-training our model. We trained our model using a larger dataset of 6000 training images and 1500 validation images. Based on training run 5's results, we chose to train our model for 15 epochs as it was one of the points where the losses started to increase after the 15th epoch.

Model Parameters

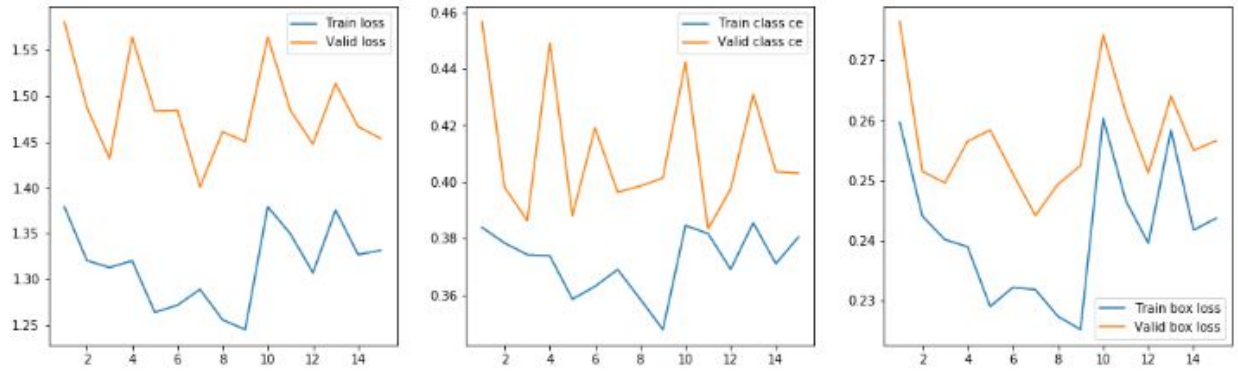
Parameters	# Number of training steps per epoch
	STEPS_PER_EPOCH = 6000
	VALIDATION_STEPS = 1500
	
	# Learning rate
	LEARNING_RATE=0.001
	
	# Skip detections with < 65% confidence
	DETECTION_MIN_CONFIDENCE = 0.65

# Epoch	15

Model Evaluation

epoch	val_loss	val_rpn_class_loss	val_rpn_bbox_loss	val_mrcnn_class_loss	val_mrcnn_bbox_loss	val_mrcnn_mask_loss	loss
1	1.581092	0.113114	0.407088	0.456547	0.276372	0.327968	1.379078
2	1.487379	0.138506	0.383365	0.397988	0.251507	0.316008	1.320369
3	1.431519	0.115713	0.378553	0.386207	0.249592	0.301449	1.312679
4	1.564533	0.146404	0.403594	0.448934	0.256499	0.309093	1.320107
5	1.483617	0.127337	0.404669	0.388084	0.258379	0.305137	1.263842
6	1.484293	0.133516	0.378658	0.419019	0.251136	0.301951	1.271485
7	1.400907	0.087174	0.361749	0.396301	0.244151	0.311516	1.288857
8	1.460896	0.104687	0.395162	0.398505	0.249350	0.313173	1.255608
9	1.450154	0.113099	0.374545	0.401329	0.252447	0.308715	1.244931
10	1.564358	0.122073	0.406494	0.442295	0.274255	0.319238	1.379108
11	1.484478	0.119886	0.407364	0.383330	0.261282	0.312613	1.349501
12	1.447503	0.111305	0.375901	0.397436	0.251213	0.311642	1.306913
13	1.513756	0.116285	0.396300	0.430974	0.264033	0.306161	1.375375
14	1.466527	0.105965	0.389383	0.403574	0.254987	0.312615	1.326762
15	1.453656	0.122315	0.363906	0.403046	0.256593	0.307791	1.331548

rpn_class_loss	rpn_bbox_loss	mrcnn_class_loss	mrcnn_bbox_loss	mrcnn_mask_loss
0.077179	0.336080	0.383837	0.259619	0.322361
0.070796	0.316364	0.378327	0.244066	0.310813
0.069344	0.322319	0.374240	0.240152	0.306619
0.073525	0.330474	0.373769	0.238956	0.303374
0.065482	0.310062	0.358499	0.229014	0.300774
0.063786	0.311062	0.363030	0.232198	0.301397
0.073492	0.316552	0.368962	0.231829	0.298008
0.066649	0.307245	0.358501	0.227385	0.295811
0.067287	0.307244	0.347745	0.225188	0.297448
0.076493	0.335943	0.384531	0.260321	0.321818
0.074041	0.336622	0.381669	0.246625	0.310540
0.066505	0.325192	0.369064	0.239543	0.306604
0.076371	0.333067	0.385514	0.258373	0.322049
0.074157	0.330846	0.371011	0.241724	0.309021
0.072222	0.324855	0.380466	0.243714	0.310285



By obtaining the losses for each epoch and plotting the corresponding loss graphs, we found that the loss for epoch 9 was the lowest overall (considering both training and validation loss). Even though Epoch 7 had the lowest validation loss, its training loss was a peak, and decreased to the lowest loss in epoch 9. Epoch 9 had a lower validation loss than epoch 8. Thus, we chose the model in epoch 9 as our best model.

Manual Inspection Sample Images (Actual vs. Predicted)





Epoch 9 results

Val_ loss	Val_ Rpn_ Class_ loss	Val_ Rpn_ bbox_loss	Val_ mrcnn_cl ass_loss	Val_ mrcnn_b box_loss	Val_ mrcnn_m ask_loss	loss	Rpn_ class_loss	Rpn_ bbox_loss	Mrcnn_c lass_ loss	mrcnn_bb ox_loss	Mrcnn_m ask_ loss
1.540154	0.113099	0.374545	0.401329	0.252447	0.308715	1.244931	0.067287	0.307244	0.347745	0.225188	0.297448

Validation Loss

- **Overall:** 1.540154
- **rpn_bbox_loss:** How good the model is at locating objects within the image
  (0.375/ Accuracy of 63%)
- **Mrcnn_bbox_loss:** How good the model is at precisely predicting the area(s) within an image corresponding to the different objects that are present
  (0.252447/ Accuracy of 75%)
- **mrcnn_mask_loss :** How well the model segment objects.
  (0.308715/ Accuracy of 70%)
- **mrcnn_class_loss:** It covers all the object classes and evaluates how well the Mask RCNN recognises each class of object.
  (0.401329/ Accuracy of 60%)
- **rpn_class_loss:** How well the Region Proposal Network separates background with objects
  (0.113099/ Accuracy of 90%)

Training Loss

- **Overall:** 1.244931
- **rpn_bbox_loss:** How good the model is at locating objects within the image
  (0.307244/ Accuracy of 70%)
- **Mrcnn_bbox_loss:** How good the model is at precisely predicting the area(s) within an image corresponding to the different objects that are present
  (0.225188/ Accuracy of 78%)
- **mrcnn_mask_loss :** How well the model segment objects.
  (0.297448/ Accuracy of 70%)
- **mrcnn_class_loss:** It covers all the object classes and evaluates how well the Mask RCNN recognises each class of object.
  (0.347745/ Accuracy of 66%)
- **rpn_class_loss:** How well the Region Proposal Network separates background with objects
  (0.067287/ Accuracy of 94%)

Results

Definitions

True Positive: Bounding box in actual and predicted are the same
True Negative: Bounding box not in actual is not in the predicted as well
False Positive: Bounding box does not exist in actual, but exists in predicted
False Negative: Bounding box in actual, but not in predicted

Class Labels: Pedestrian, Pvp (Partially-Visible Persons), Ignore, Crowd, Riders

Actual vs. Predicted Images

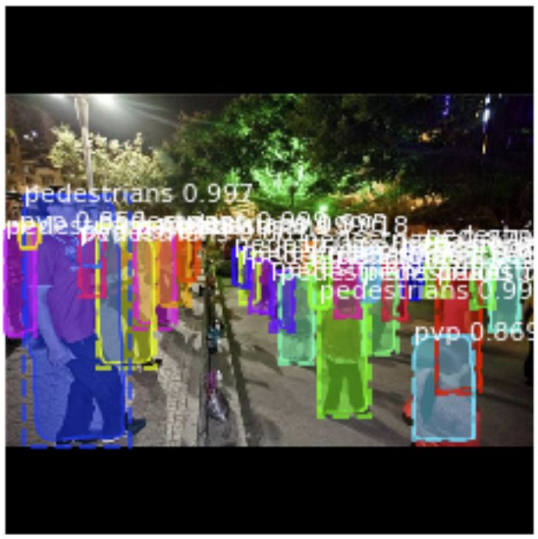
Below are some images comparing our result labels with the actual picture labels.
Note: Images on the left are ground truth, images on the right are predicted





True Positive: Comparing the actual image to the predicted, our model could accurately pick out all of the “pedestrians” in the picture.This image thus achieved a good result of 100% prediction accuracy.

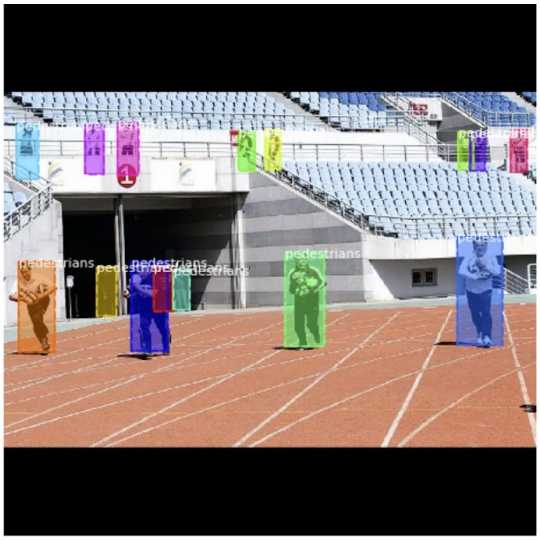


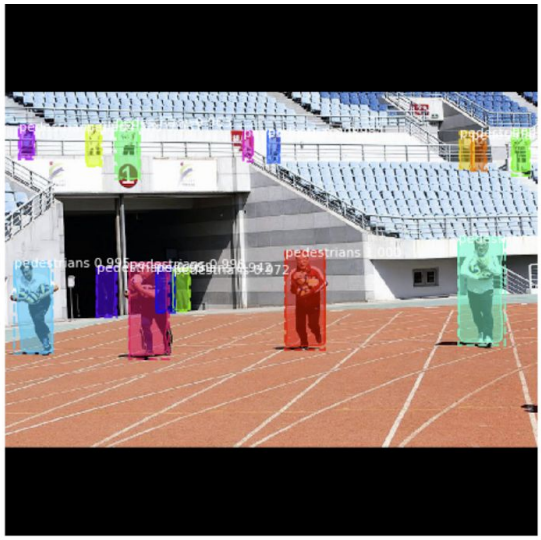


True Positive: Comparing the actual image to the predicted, our model could accurately pick out all of the “pedestrians” in the picture.

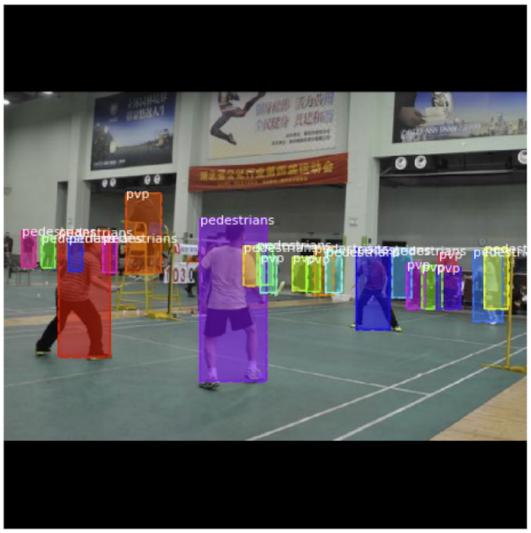
False Negative: Our model could not detect the “crowd” instance in the center of the picture. This is probably due to the fact that we did not have sufficient training data in this class.

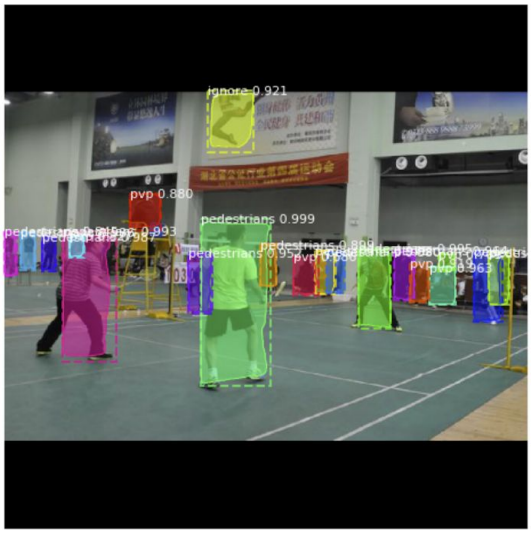
However, our model managed to accurately pick out an additional “pvp” instance that was not labelled in the actual picture.





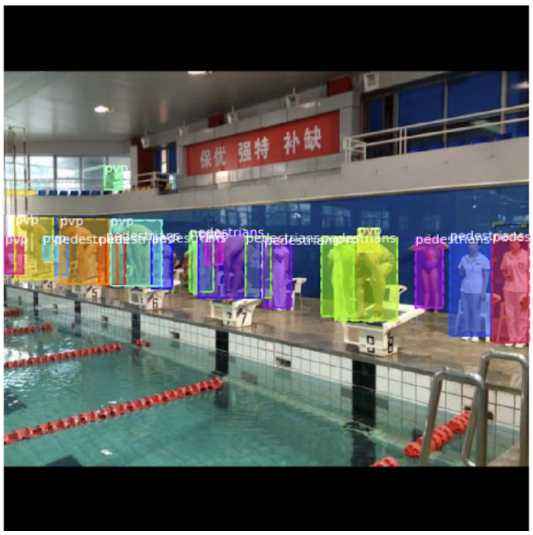
True Positive: Comparing the actual image to the predicted, our model could accurately pick out all of the “pedestrians” in the picture.This image thus achieved a good result of 100% prediction accuracy.

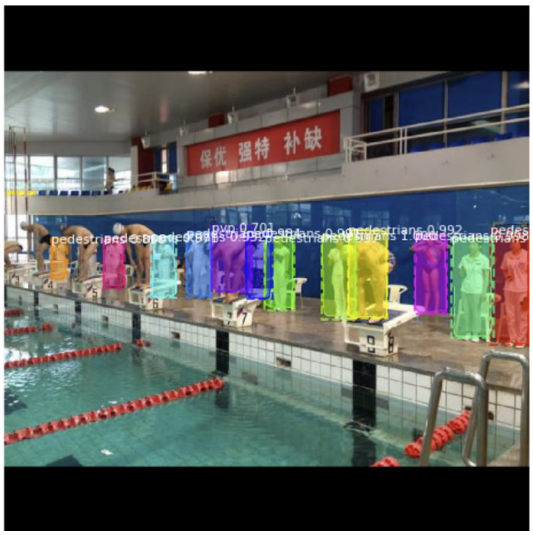




True Positive: Comparing the actual image to the predicted, our model could accurately pick out most of the “pedestrians” and “pvp” in the picture. It managed to detect the umpire in the centre of the pictures as a “pvp”.

Our model also managed to pick up the human figure in the background poster and labelled it as “ignore”. It also managed to pick up an additional pedestrian on the left of the picture (in pink). This was accurately labelled, even though it was not labelled in the actual picture.





Comparing the actual image to the predicted, some of the swimmers that had their bodies arched were labelled as “pvp” in the actual image. However, they were not picked up in the predicted images at all. It was also interesting to note that those swimmers that also had their bodies arched but were labelled as “pedestrians” were accurately labelled. This could be due to the imbalance of classes, where there were insufficient examples of “pvp” for the model to pick up these instances that had a different posture from standing “pedestrians” or “pvp”.







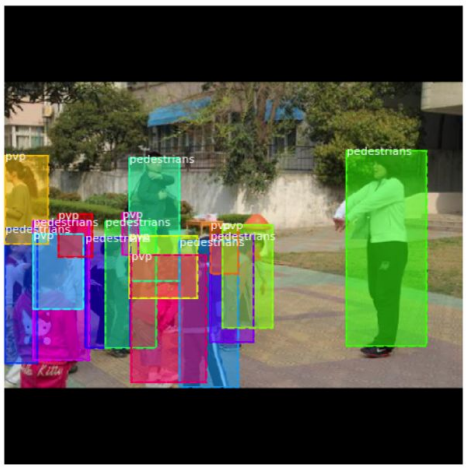
True Positive: Comparing the actual image to the predicted, the labels for “pedestrian” in the predicted picture is accurate.

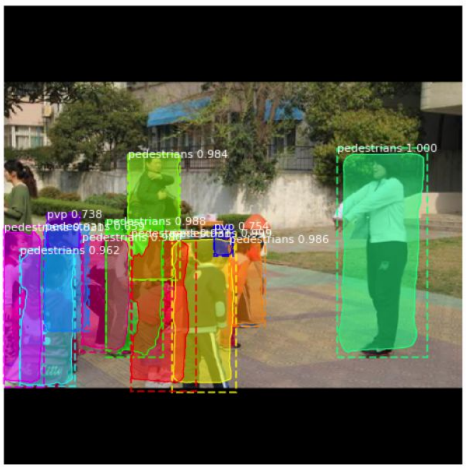
False Negative: Our model could not detect some “pedestrian” instances on the left of the picture. These pedestrians were very far in the background, and were quite small - which could have led to the model missing them out.

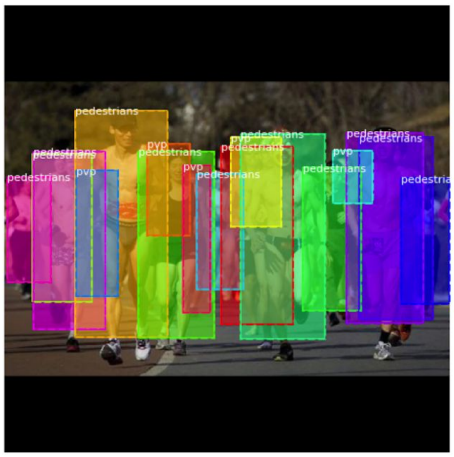
However, our model managed to accurately pick out an additional “pvp” instance in the middle of the picture, that was not labelled in the actual picture.

Misclassification Examples

Comparing the actual image with predicted labels, our model's misclassification happens when the objects seem in between 'Pedestrian' and 'PvP'. Here are a few examples of our model's misclassifications.









Summary

Overall, based on manual inspection, our model did well in classifying the "pedestrian" class, and did a decent job for the "pvp" class. Our model had difficulty picking out instances for the "Ignore", "Riders" and "Crowd" classes due to the lack of data in these classes. However, it could even pick out additional instances that were not depicted in the ground truth labels, but were accurately labelled. This may have affected the loss calculations for the evaluation of our model.

After reviewing the results, we feel that Mask R-CNN is a good technique to use in object detection, with its ability to detect the main class - ‘pedestrians’ pretty well. Furthermore, its ability for pixel-to-pixel segmentation allows edge detection, which is good in getting the rough outlines of the human figures and their postures (i.e. standing/ jumping/ bending down), etc..

Comparing our results

We compared our results with the journal ‘WiderPerson: A Diverse Dataset for Dense Pedestrian Detection in the Wild’ which applied Faster R-CNN to detect human features in the Wider Person Dataset. Initially our group referred to their previous research about pedestrian detection, and focused on making improvements attributed to variations in pose, scale and occlusion, which need to be addressed while building pedestrian detection algorithms.

As we observe the result from Faster R-CNN in Fig1, the model was able to detect most of the human features in the images in various crowded scenarios with highly occulede pedestrians.

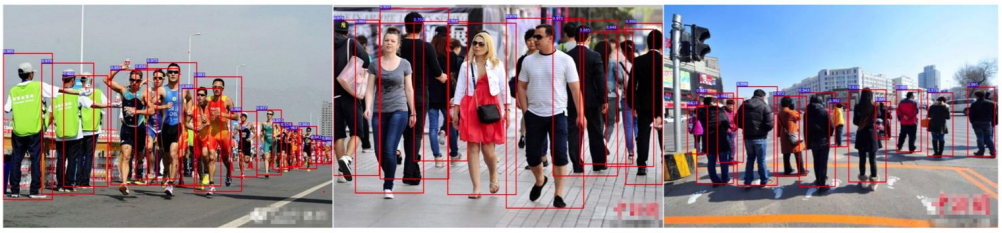


Fig1. Faster R-CNN Model result of Wider Person Journal

However, our group hoped to go one step further and locate exact pixels of each object instead of just bounding boxes. Therefore we decided to launch the ‘Image Segmentation’ techniques by adding a branch to Faster R-CNN that outputs a binary mask that says whether or not a given pixel is part of an object. As we can observe the output in Fig2, our Mask R-CNN model, extends Faster R-CNN, enables pixel-to-pixel alignment to have high-quality segmentation masks for each instance. In all, we found Mask R-CNN to be a good fit for pedestrian detection, especially with its ability for pixel-to-pixel segmentation.

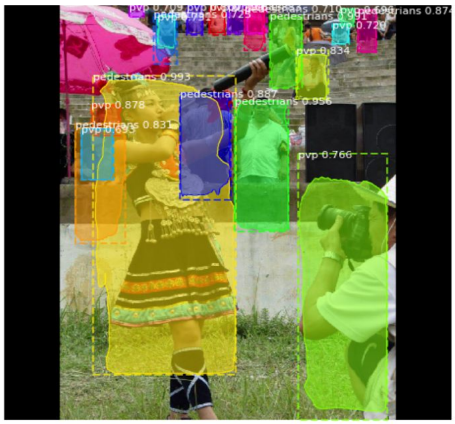


Fig 2. Image Segmentation result of our Mask R-CNN Model

Reflections

Lesson-Learned

1) Multi-class classifier requires a balanced number of training data across classes to detect the classes. As our data set was imbalanced, its multi-class classification performance was thus affected, only detecting dominant classes like “pedestrian” and “pvp” with a high recall and accuracy.
2) Manual image annotations may not always be accurate. Our model managed to learn and pick up instances that were accurate, but were not labelled in the actual picture. This may have negatively impacted the accuracy of the model.
3) We also had hands-on experience with parameter tuning for computer vision techniques - which was a tedious but extremely rewarding process. We learnt more about the current state of art technologies in Computer Vision, and obtained a better understanding of Instance segmentation and Object Detection. Object Detection focuses on classifying the object, while instance segmentation segments each picture with high resolution (i.e. pixel-by-pixel).

Findings

Firstly, our model’s results did surprise us.

Though some of the pedestrians may be overlapped, be partially visible, or may not even be standing upright, our model could still pick out those pedestrians. In one of our test cases, the “partially-visible person” was upside down mid-flip, and our model managed to accurately detected that person (Fig 1). Being able to pick out partially visible persons, and persons not standing upright is interesting and could expand its applications in human detection.

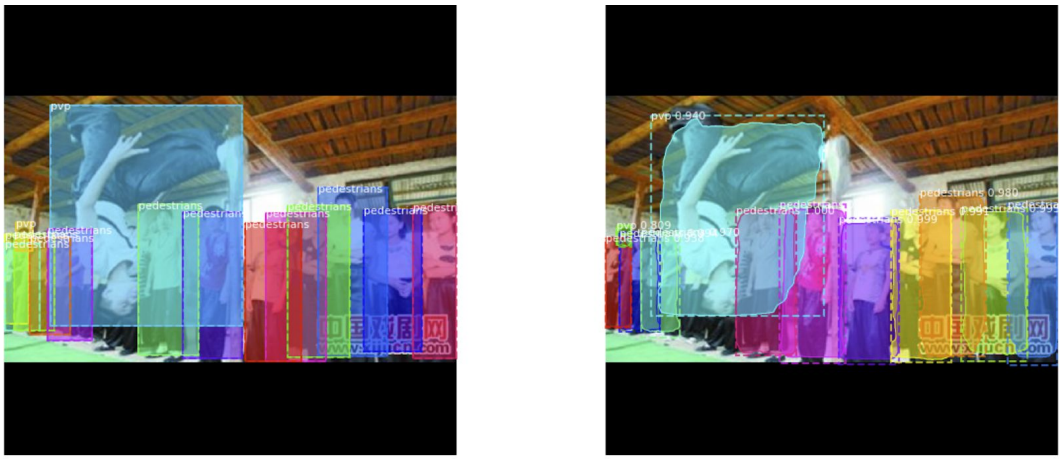


Fig 1

Our model was also able to pick out non-humans - classified in the ignore class (Fig 2). The mascot was not tagged as a pedestrian - which was accurate as it is technically not a human.



Fig 2

We also underestimated the computing power required as we were working with thousands of images. We started working on Google Collab, but was quickly deprioritised and it would not support our large number of images. We tried using our local, but the environment settings were unfavourable, especially as NVIDIA's CUDA was needed. Thus we had to move on to Amazon Web Services for higher computing power. To support our model, we used AWS's EC2 P2.XL Instance with Deep Learning AMI, with 1 GPU. Moving to AWS really allowed us to understand the power and efficiency of the GPU. On our local machine's CPU, 1 epoch with about 1000 images took about 6 hours to run, while 1 epoch on a GPU took about 2 hours to run for 6000 images. However, the costs involved with obtaining such computing capabilities was definitely not cheap (Figure 3).

Amazon Elastic Compute Cloud running Linux/UNIX		*$132.98*
$0.9 per On Demand Linux p2.xlarge Instance Hour	*147.750 Hrs*	*$132.98*

Fig 3: Our AWS Bill

Retrospect

Fortunately, our data was well-labelled and relatively clean (i.e. the pictures were re-sized to the same size, etc.). Hence, there was minimal preprocessing that we had to do.
Detecting all 5 different class pedestrians in the proposed dataset was extremely challenging in the very beginning of our training stage due to large variations in the scenario and our imbalanced dataset. The raw numbers of data for each class varied across classes, with "pedestrian" being the dominant class. However, as our data consisted of images, balancing the class labels across the images was difficult as each picture depicted real life environments,

which we had no control over. Due to the limited time, we did not manage to balance the classes before training our model - hence resulting in some classes (i.e. crowd, ignore) not being detectedFind by our model.

Our computational power was also limited to train the M-RCNN model with thousands of images. This made training our model extremely time consuming. It also limited us in our abilities to discover the best model within the time constraints.

Finding the optimal number of epochs, and tuning the model configuration parameters was also a challenge. We tried to do multiple runs on different parameters with smaller datasets before moving on to bigger data sets - but the correlation between smaller datasets and larger ones were not definite. Eventually, after studying smaller datasets, we decided on a minimum detection confidence of 0.65, as a majority of the detection confidence levels were between 0.

Collaborating across different time zones was also challenging at first, as we were working with 12 hour time differences - in Singapore and in Pittsburgh. However, with effective work distribution and settling on a fixed meeting time each week, we managed to work it out and successfully complete the project.

Next Steps

To improve our model, more data for the other classes should be obtained in order to better perform multi class classification. Due to the time constraints, we could not continue to train the model to achieve even better results. To obtain a better model, a larger training set could be used and more parameter tuning could be done.

Our model is currently implemented on still images, and could be expanded to video detection, such as in live images to get frame by frame detection. We could also look towards exploring other computer vision techniques or object detection models to solve our problem, and comparing them with our model to ultimately find the best model for our use case of autonomous vehicles.

And as mentioned in our exploratory data analysis, the WiderPerson dataset has a severe imbalance between classes. This may have affected our model's ability to accurately classify each identified record of human features. We could have applied Synthetic Minority Over-sampling Technique (SMOTE)[9] to synthetically increase the number of under-represented classes in our dataset.

We would also have used a more expensive AWS instance with higher computational power. This will reduce the amount of time needed to train our model and we can work with a larger subset or the entire dataset to increase our model's performance.

---

[9] He, H., & Garcia, E. (2009). Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering,21*(9), 1263-1284. doi:10.1109/tkde.2008.239

Project Work Responsibilities

Anita	Exploratory Data Analysis, Attempted YOLO Modelling, AWS Setup
Bernadine	MRCNN Model, AWS Configurations & Setup
Woojin	MRCNN Model, Model Evaluations Research
All	Research different methodologies for Pedestrian Identification

Appendix

Exploratory Data Analysis Code

```
import pandas as pd
# Method to load a single img txt file
def EDA_load_single_img_data(file_name) :
    classes = [0,0,0,0,0,0]

    file_name_path = "drive/My Drive/AI Project/Data/WiderPerson/Annotations/" + file_name

    #Open image file
    with open(file_name_path) as file:
        for line in file:
            if classes[0] == 0:
                #Adds to total sum of humanCount
                classes[0] += int(line.strip())

            else:
                class_label = int(line[0])
                #print(class_label)
                classes[class_label] +=1

    return classes
```

```
#Method loads all image and image innotation
#Method returns a list of:
#[Total_humans, total_class1, total_class2, total_class3, total_class4, total_class5]
def EDA_load_all_img_data() :
  human_data = []
  total_hc = 0
  numPics = 0

  #Path to folder with annotation
  folder = "drive/My Drive/AI Project/Data/WiderPerson/Annotations"
  files = os.listdir(folder);

  for file_name in files:
    numPics+=1
    #Find image based on annotation file name
    row = EDA_load_single_img_data(file_name)
    human_data.append(row)

  return human_data, numPics

human_data,numPics = EDA_load_all_img_data()
df = pd.DataFrame(human_data)
df.columns= ['totalhc', 'hc1', 'hc2', 'hc3', 'hc4', 'hc5']
```

```
totalhc = df['totalhc'].sum()
hc1 = df['hc1'].sum()
hc2 = df['hc2'].sum()
hc3 = df['hc3'].sum()
hc4 = df['hc4'].sum()
hc5 = df['hc5'].sum()

avgHuman = totalhc / numPics
avgClass = [round(hc1/numPics, 2), round(hc2/numPics, 2),  round(hc3/numPics, 2),  round(hc4/numPics, 2),  round(hc5/numPics, 2), ]
```

```
import matplotlib.pyplot as plt
import seaborn as sns

print("Density of Humans per Image: ", avgHuman)

#Method labels bar chart
def autolabel(rects):
    """Attach a text label above each bar , displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

#Data for respective axis
y_pos = ['pedestrians', 'riders', 'partially-visible persons', 'ignore regions', 'crowd' ]
x_pos = [hc1, hc2, hc3, hc4, hc5]

fig, ax = plt.subplots()

ax1 = ax.bar(y_pos, x_pos , align='center', alpha= 0.5)

plt.ylabel('Humans')
plt.title('Total Number of Humans per Classification')


autolabel(ax1)
fig.autofmt_xdate()
plt.show()
```

```
Density of Humans per Image:  30.4041445035461
```

```
#Class Percentage breakdown for pie chart
perc = [hc1/totalhc, hc2/totalhc, hc3/totalhc, hc4/totalhc, hc5/totalhc]

fig1, ax1 = plt.subplots()
ax1.pie(perc, labels=y_pos, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()
```

```
y_pos = ['pedestrians', 'riders', 'partially-visible persons', 'ignore regions', 'crowd' ]

fig, ax = plt.subplots()

ax1 = ax.bar(y_pos, avgClass , align='center', alpha= 0.5)

plt.ylabel('Humans')
plt.title('Average Number of Humans per Classification per Image')

autolabel(ax1)
fig.autofmt_xdate()
plt.show()
```

MRCNN Code

```
# Gets the GPU configurations and information for the project
UseAllowGrowth = True
UseGPUnum = "0"

import os
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"   # see issue #152
os.environ["CUDA_VISIBLE_DEVICES"]= str(UseGPUnum)
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '0'

if(UseAllowGrowth):
    from tensorflow.compat.v1 import ConfigProto
    from tensorflow.compat.v1 import InteractiveSession
    config = ConfigProto()
    config.gpu_options.allow_growth = True
    session = InteractiveSession(config=config)

# Show GPU info
from tensorflow.python.client import device_lib
device_lib.list_local_devices()
```

```
# Core import statements necessary for the project
import mrcnn
from PIL import Image
from mrcnn.utils import Dataset
from mrcnn.model import MaskRCNN
from mrcnn.config import Config
from mrcnn import model as modellib
from mrcnn import visualize

import os
```

Class PedestrianDataset

```
import numpy as np
from numpy import zeros
from numpy import asarray
import colorsys
import argparse
import imutils
import random
import os
import time
from matplotlib import pyplot
from matplotlib.patches import Rectangle
from keras.models import load_model
%matplotlib inline
from os import listdir


class PedestrianDataset(Dataset):
    # load the dataset definitions
    def load_dataset(self, dataset_dir, is_train=True):

        # Add classes.
        # class_label =1: pedestrians
        # class_label =2: riders
        # class_label =3: partially-visible persons
        # class_label =4: ignore regions
        # class_label =5: crowd
        self.add_class("dataset", 1, "pedestrians")
        self.add_class("dataset", 2, "riders")
        self.add_class("dataset", 3, "pvp")
        self.add_class("dataset", 4, "ignore")
        self.add_class("dataset", 5, "crowd")

        # define data locations for images and annotations
        images_dir = dataset_dir + '/Images/'
        annotations_dir = dataset_dir + '/Annotations/'

        # Iterate through all files in the folder to
        #add class, images and annotaions
        count = 0
        for filename in listdir(images_dir):

            # extract image id
            image_id = filename.split(".")[0]

            # setting image file
            img_path = images_dir + filename

            # setting annotations file
            ann_path = annotations_dir + image_id + '.jpg.txt'
            try:
              file = open(ann_path)
              count+=1
              # skip all images after 1500 if we are building the train set
              if is_train and count >= 6001:
                  continue
              # skip all images before 150 if we are building the test/val set
              if not is_train and (count < 6001 or count >= 7501):
                continue
              # adding images and annotations to dataset
              self.add_image('dataset', image_id=image_id, path=img_path, annotation=ann_path)
            except FileNotFoundError:
              print("File with: "+ ann_path + " not found")
```

```
# extract bounding boxes from an annotation file
def extract_boxes(self, image_id):

    # extract each bounding box
    boxes = list()
    class_ids = list()

    annot_name_path = self.image_info[image_id]['annotation']
    img_name_path = self.image_info[image_id]['path']
    print(img_name_path)
    count = 0
    with open(annot_name_path) as file:
      for line in file:
          if count == 0:
              count+=1
          else:
              class_ids.append(line.strip().split(" ")[0])
              coors = line.strip().split(" ")[1:5]
              coors = [int(x) for x in coors]
              boxes.append(coors)

    im = Image.open(img_name_path)
    width, height = im.size

    return class_ids, boxes, width, height

# load the masks for an image
"""Generate instance masks for an image.
   Returns:
    masks: A bool array of shape [height, width, instance count] with
        one mask per instance.
    class_ids: a 1D array of class IDs of the instance masks.
 """
def load_mask(self, image_id):
    # get details of image
    info = self.image_info[image_id]
    print(info)

    # define anntation  file location
    # path = info['annotation']

    # load bounding boxes
    class_ids, boxes, w, h = self.extract_boxes(image_id)

    # create one array for all masks, each on a different channel
    masks = zeros([h, w, len(boxes)], dtype='uint8')

    # create masks
    for i in range(len(boxes)):
        box = boxes[i]
        row_s, row_e = box[1], box[3]
        col_s, col_e = box[0], box[2]
        masks[row_s:row_e, col_s:col_e, i] = 1

    return masks, asarray(class_ids, dtype='int32')

# load an image reference
"""Return the path of the image."""
def image_reference(self, image_id):
    info = self.image_info[image_id]
    print(info)
    return info['path']
```

Preparing the Train & Test Set

```
# prepare train set
train_set = PedestrianDataset()
train_set.load_dataset("WiderPerson", is_train=True)
train_set.prepare()

# prepare test/val set
test_set = PedestrianDataset()
test_set.load_dataset("WiderPerson", is_train=False)
test_set.prepare()

print("Train: %d" % len(train_set.image_ids))
print("Test: %d" % len(test_set.image_ids))
```

Model Configurations

```
# Configurations for the model
class myMaskRCNNConfig(Config):
    # give the configuration a recognizable name
    NAME = "MaskRCNN_config"

    # set the number of GPUs to use along with the number of images
    # per GPU
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1

    # number of classes (we would normally add +1 for the background)
     # 5 + BG
    NUM_CLASSES = 5+1

    # Number of training steps per epoch
    STEPS_PER_EPOCH = len(train_set.image_ids) #size of training set
    VALIDATION_STEPS = len(test_set.image_ids) # size of test set

    # Learning rate
    LEARNING_RATE=0.001

    # Skip detections with < 65% confidence
    DETECTION_MIN_CONFIDENCE = 0.65

    # setting Max ground truth instances
    MAX_GT_INSTANCES=1000

config = myMaskRCNNConfig()
config.display()
```

Training the Model

```
import tensorflow as tf
import sys

print("Loading Mask R-CNN model...")
model = modellib.MaskRCNN(mode="training", config=config, model_dir="ModelCheckpoints/")
#Load the weights for COCO, exclude layers for training
model.load_weights('.\\mask_rcnn_coco.h5',
                   by_name=True,
                   exclude=["mrcnn_class_logits", "mrcnn_bbox_fc","mrcnn_mask"])

# train heads with higher lr to speedup the learning
model.train(train_set, test_set, learning_rate=2*config.LEARNING_RATE, epochs=15, layers="heads")
history = model.keras_model.history.history

import time
# save the  model
model_path = 'PedestrianModel/mask_rcnn_'  + '.' + str(time.time()) + '.h5'
model.keras_model.save_weights(model_path)
```

Prediction Model

```
# Detection Model

from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array

model_path = 'ModelCheckpoints/maskrcnn_config20200426T0504/mask_rcnn_maskrcnn_config_0009.h5'

#Loading the model in the inference mode
model = modellib.MaskRCNN(mode="inference", config=config, model_dir='ModelCheckpoints/')
# loading the trained weights on the custom dataset
model.load_weights(model_path, by_name=True)
img = load_img("WiderPerson/Images/015298.jpg")
img = img_to_array(img)
# detecting objects in the image
result= model.detect([img])
result
```

Evaluation: Loss Graphs

```
# Plotting loss graphs for epoch

import pandas as pd
# history = model.keras_model.history.history
# epochs = range(1,len(next(iter(history.values())))+1)
# pd.DataFrame(history, index=epochs)

epochs_val = pd.read_csv('output_val/output_val_15e.csv')
epochs=range(1,len(epochs_val)+1)

import matplotlib.pyplot as plt
plt.figure(figsize=(17,5))

plt.subplot(131)
plt.plot(epochs, epochs_val["loss"], label="Train loss")
plt.plot(epochs, epochs_val["val_loss"], label="Valid loss")
plt.legend()
plt.subplot(132)
plt.plot(epochs, epochs_val["mrcnn_class_loss"], label="Train class ce")
plt.plot(epochs, epochs_val["val_mrcnn_class_loss"], label="Valid class ce")
plt.legend()
plt.subplot(133)
plt.plot(epochs, epochs_val["mrcnn_bbox_loss"], label="Train box loss")
plt.plot(epochs, epochs_val["val_mrcnn_bbox_loss"], label="Valid box loss")
plt.legend()

plt.show()
```

Evaluation: Comparing Actual vs Predicted images

```
import matplotlib.pyplot as plt
dataset = test_set
fig = plt.figure(figsize=(20, 60))

for i in range(5):
    # Get a random image from the data set
    image_id = random.choice(dataset.image_ids)

    # Comparing ACTUAL vs PREDICTED
    original_image, image_meta, gt_class_id, gt_bbox, gt_mask =\
        modellib.load_image_gt(test_set, config,image_id, use_mini_mask=False)

    print(original_image.shape)
    plt.subplot(6, 2, 2*i + 1)
    visualize.display_instances(original_image, gt_bbox, gt_mask, gt_class_id,
                                dataset.class_names,ax=fig.axes[-1])

    plt.subplot(6, 2, 2*i + 2)
    results = model.detect([original_image]) #, verbose=1)
    r = results[0]
    visualize.display_instances(original_image, r['rois'], r['masks'], r['class_ids'],
                                dataset.class_names, r['scores'], ax=fig.axes[-1])
```