Cardiff University

Group G9 Project

Object Localisation

Students:

Alex Westcott : 1432644

Tianbai Peng : 1981034

Shuai Han : 1964610

Yuting Nieh : 1965066

Aman Anand : 1980286

Deepika Velusamy : 1966924

Supervisor:

Dr. Yukan Lai

April 2020

Contents

0 Introduction

Nowadays, object detection techniques have become more and more popular for distinguishing objects in images and videos. Object detection in images is usually split up into two sections, classification, which concerns itself with identifying what each object in an image is and localisation which is the task of locating objects in the image. Object detection is challenging because it requires both, the correct detection of all objects in a picture and great precision in describing their locations. Here Mask R-CNN is a very good solution which provides us with a method for object detection and instance segmentation. We can see that Mask R-CNN integrates many previous excellent research results and it has been widely accepted in industry and in academia.

In this project, we focus mainly on the theory and implementation of Mask R-CNN for object detection. We start by performing careful analysis of our chosen data-set and giving some insights into the limitations of less complex models. We end with outlining the numerical results and visual errors made by our trained model, giving analysis into why these errors occur. We believe the flexibility and accuracy of our model meets all the requirements of a desired object detection model.

1 Description of the task and dataset

The aim of this project was to create a model which could detect and locate certain objects in images. The objective of the created model would be to take an image as input and output co-ordinates which would describe rectangles inscribing each object in the image. For this project we use the VOC2012 data-set from the 2012 Visual Object Classes Challenge. This data-set comes with images containing objects from a variety of classes, however, for this project we use a subset of these images, specifically those containing people. Our choice was to select only a small number of object classes to create a model that could perform object detection very well, rather than one that could detect many classes but with a lower performance.

Our data-set consisted of a set of images containing 1 or more people and a set of annotation files describing each object in the image. From the annotations we can see that the images contained objects from the following classes, person, head, hand, foot, diningtable, chair, potted plant and sofa. To analyse the data-set we first extracted the contents of the annotation files to a csv file. This included the image name, the dimensions of the image, the labels and co-ordinates of each object in the image. From this, we could then collect all the different object labels and calculate the percentage representation of each label out of the total amount of objects.

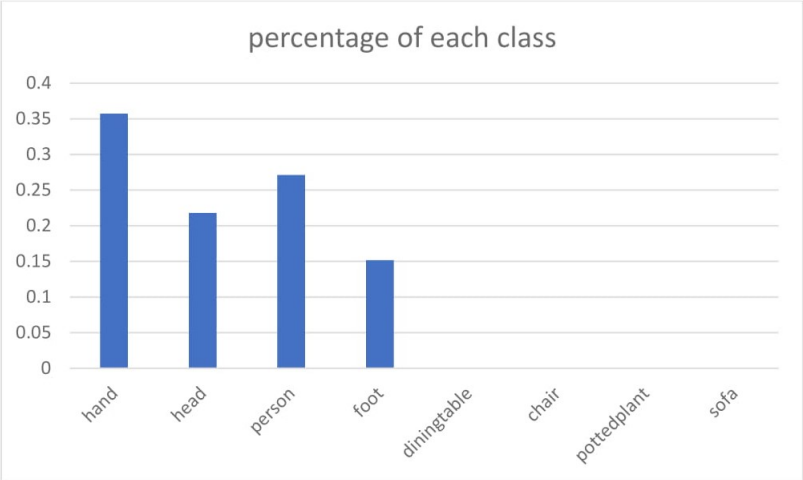


Figure 1: percentage representation of each class out of the total amount of objects

From Figure 1 we can see that the labels 'chair', 'diningtable', 'pottedplant' and 'sofa' make up a very small percentage, less than 0.1% each, of all objects across these images. A model trained from this set of images is not going to be able to learn to classify and locate any of these 4 aforementioned classes, due to the number of instances being so low. For this reason, any annotations for these classes were removed from our data-set.

We then looked at the number of objects in each image to get a feel for the distribution throughout the data-set. Figure 2 shows the number of images containing different numbers of objects. The distribution of this data has a mean of 6.4 and a standard deviation of 4.2. From these values and figure 2, it is clear that the majority of the images contain between 3 and 10 objects. Using the inter-quartile range we can locate the outliers, by the formulas $Q3 + 1.5 \cdot IQR$ and $Q1 - 1.5 \cdot IQR$, where IQR is the inter-quartile range and Q1, Q3 are the first and third quartile respectively. We have $Q1 = 4$ and $Q3 = 8$, hence there are no outliers below Q1, as $4 - (1.5 \cdot (8 - 4)) = -2 < 3$, (3 being the smallest number of objects). However, all values above $8 + (1.5 \cdot (8 - 4)) = 14$, are outliers, giving 27 images classified as such. These images should not have much impact in affecting the training of any model on the data-set, although, any model may be more likely to make errors when performing object detection on these images.

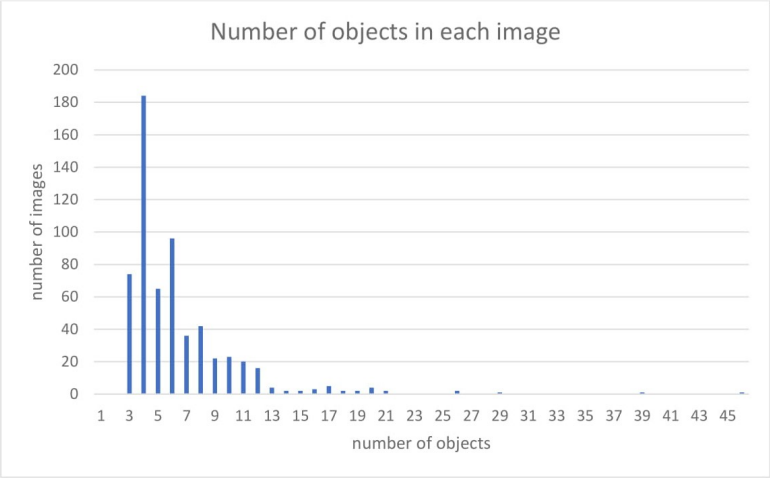


Figure 2: Graph showing the number of images containing different numbers of objects.

2 Methodology

For this project we chose 2 approaches to locate and classify people in images. The first of these was a simple Convolutional Neural Network designed to locate a single person given an image containing only one person. The other implementations was Mask R-CNN a fairly new technique used for both object detection and instance segmentation. Although this project only focuses on the detection part, the multi-task learning used by Mask R-CNN improves the precision of the model when performing object detection. Hence, it was this reason that made Mask R-CNN a good choice for our main implementation.

2.1 Convolutional Neural network

As a starting point for the project, it was important to understand the implementation and limitations of the underlying techniques that could be used to build more complex models that are used in the real world. Using Keras we built a CNN which would take an image containing one person and output 4 numbers corresponding to the co-ordinates of a bounding box describing the size and position of the person in the image.

In order to create this implementation we started by forming a new data-set using images containing 1 person. From this data-set, a csv file was created that contained information about each of the images. The information contained in the csv file was, the filename, width and height of each image, along with the object name and 4 co-ordinates for the object location in each image. This information was gathered using the xml annotation files corresponding to each image in the data-set and a custom script utilising the "ElementTree" module from "xml.etree" in Python.

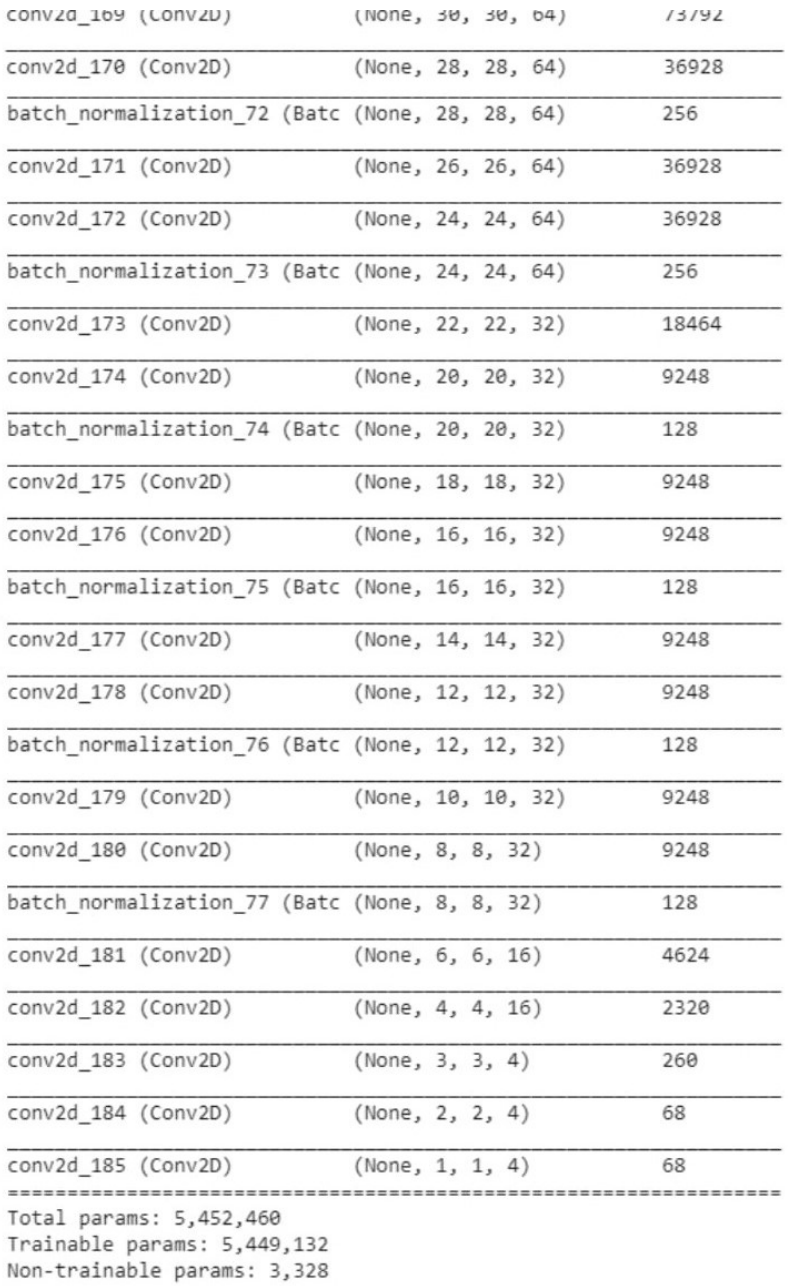
The second part of pre-processing involved turning the images were into numpy arrays, resizing them to 228x228x3 and normalising their values by dividing by 255. The ground truth labels were then created, where each label was a set of 4 numbers corresponding to the co-ordinates of the bounding box describing the person in the image. Finally the data-set was split into two smaller sets for training and validating, a split of 75%/25% was used for the sets respectively.

With the pre-processing completed the CNN model was then created. The model consisted of a series of convolutional layers and batch normalisation (Figure 3). Instead of using fully connected layers for the output, convolutional layers were used instead to create an output volume of 1x1x4, which would be equivalent to a dense layer containing 4 nodes.

This model was then trained using the training set and the optimizer was set to be Stochastic Gradient Decent, (SGD). The loss function used was a custom function using the sum of the mean squared error and the intersection over union. These two losses were chosen as IoU is a great measure of how alike the predicted and actual bounding boxes are and MSE helps penalise the predicted co-ordinates which are very dissimilar to the actual values.

After training the model on 100 epochs and using IoU as the metric for accuracy, the model achieved an accuracy value of 0.53. This value shows that the model was indeed able to learn to some degree how to locate people in images. The accuracy could be improved slightly by tuning the different hyper-parameters of the SGD optimiser, however, the model will never be able to achieve a fantastic result, showing the limitations of such a simple approach. For this reason, we look to more complex model architectures to tackle our object detection problem.

Layer (type)	Output Shape	Param #
conv2d_155 (Conv2D)	(None, 113, 113, 256)	7168
conv2d_156 (Conv2D)	(None, 56, 56, 256)	590080
batch_normalization_65 (Batc	(None, 56, 56, 256)	1024
conv2d_157 (Conv2D)	(None, 54, 54, 256)	590080
conv2d_158 (Conv2D)	(None, 52, 52, 256)	590080
batch_normalization_66 (Batc	(None, 52, 52, 256)	1024
conv2d_159 (Conv2D)	(None, 50, 50, 256)	590080
conv2d_160 (Conv2D)	(None, 48, 48, 256)	590080
batch_normalization_67 (Batc	(None, 48, 48, 256)	1024
conv2d_161 (Conv2D)	(None, 46, 46, 256)	590080
conv2d_162 (Conv2D)	(None, 44, 44, 256)	590080
batch_normalization_68 (Batc	(None, 44, 44, 256)	1024
conv2d_163 (Conv2D)	(None, 42, 42, 128)	295040
conv2d_164 (Conv2D)	(None, 40, 40, 128)	147584
batch_normalization_69 (Batc	(None, 40, 40, 128)	512
conv2d_165 (Conv2D)	(None, 38, 38, 128)	147584
conv2d_166 (Conv2D)	(None, 36, 36, 128)	147584
batch_normalization_70 (Batc	(None, 36, 36, 128)	512
conv2d_167 (Conv2D)	(None, 34, 34, 128)	147584
conv2d_168 (Conv2D)	(None, 32, 32, 128)	147584
batch_normalization_71 (Batc	(None, 32, 32, 128)	512



conv2d_169 (Conv2D)	(None, 30, 30, 64)	73792
conv2d_170 (Conv2D)	(None, 28, 28, 64)	36928
batch_normalization_72 (Batc	(None, 28, 28, 64)	256
conv2d_171 (Conv2D)	(None, 26, 26, 64)	36928
conv2d_172 (Conv2D)	(None, 24, 24, 64)	36928
batch_normalization_73 (Batc	(None, 24, 24, 64)	256
conv2d_173 (Conv2D)	(None, 22, 22, 32)	18464
conv2d_174 (Conv2D)	(None, 20, 20, 32)	9248
batch_normalization_74 (Batc	(None, 20, 20, 32)	128
conv2d_175 (Conv2D)	(None, 18, 18, 32)	9248
conv2d_176 (Conv2D)	(None, 16, 16, 32)	9248
batch_normalization_75 (Batc	(None, 16, 16, 32)	128
conv2d_177 (Conv2D)	(None, 14, 14, 32)	9248
conv2d_178 (Conv2D)	(None, 12, 12, 32)	9248
batch_normalization_76 (Batc	(None, 12, 12, 32)	128
conv2d_179 (Conv2D)	(None, 10, 10, 32)	9248
conv2d_180 (Conv2D)	(None, 8, 8, 32)	9248
batch_normalization_77 (Batc	(None, 8, 8, 32)	128
conv2d_181 (Conv2D)	(None, 6, 6, 16)	4624
conv2d_182 (Conv2D)	(None, 4, 4, 16)	2320
conv2d_183 (Conv2D)	(None, 3, 3, 4)	260
conv2d_184 (Conv2D)	(None, 2, 2, 4)	68
conv2d_185 (Conv2D)	(None, 1, 1, 4)	68

Total params: 5,452,460
Trainable params: 5,449,132
Non-trainable params: 3,328

Figure 3: Simple Convolutional Neural Network Architecture

2.2 Mask R-CNN

Mask R-CNN, (Region Convolutional Neural Network), is a method for object detection and instance segmentation, which is an extension of the Faster R-CNN method. Therefore, before we discuss the ideas behind this method, we shall first take a look at the fundamentals of Faster R-CNN.

2.2.1 Faster R-CNN

Faster R-CNN is an implementation designed to improve on both R-CNN and Fast R-CNN. These two methods use an algorithm called selective search to obtain region proposals, which the models will then use to attempt to identify objects. This Algorithm is the bottleneck for the time efficiency of the Fast R-CNN model, hence, Faster R-CNN was created to remove the need for selective search, [1].

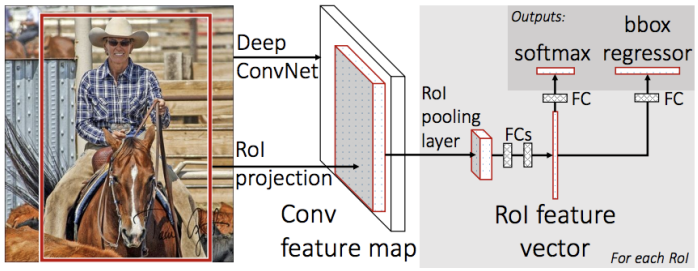


Figure 4: Fast R-CNN architecture, [2].

Faster R-CNN works by first using a convolutional neural network to output a convolutional feature map from an inputted image. The difference with Fast R-CNN from here on, is that instead of selective search to identify region proposals, a separate fully convolutional network is used to predict them instead, (figure 5). Faster R-CNN is composed of two modules, a Region Proposal Network, (RPN) and a Fast R-CNN detector, which combine to make a single network for object detection, [3]. With this, the model gives two final outputs for each of the region proposals, a softmax to classify the object and a regressor to determine the size and position of the bounding box.

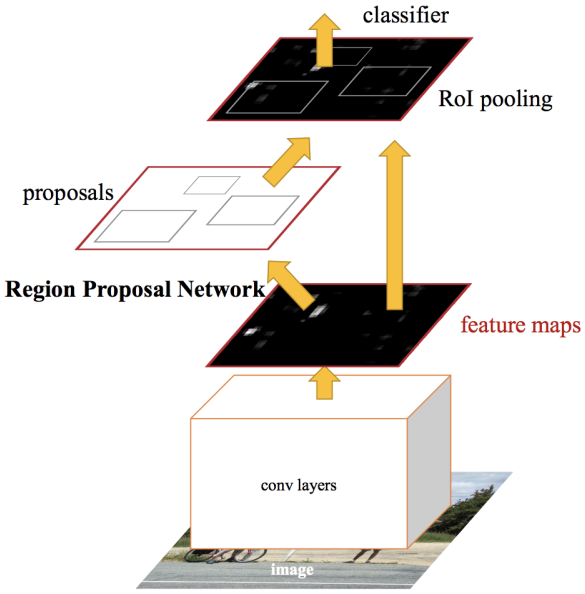


Figure 5: Faster R-CNN architecture

2.2.2 Mask R-CNN, Theory

Mask R-CNN works in exactly the same way as Faster R-CNN except it outputs an additional 3rd object, an object mask. This mask encodes each pixel in the original image using one value for each that does not contain the object and one for each that does. Therefore the mask and the input image will both have the same dimensions. Mask R-CNN is an excellent choice for performing object segmentation, however, although it is similar to Faster R-CNN, it outperforms all base-variants of it's previous state-of-the-art models when performing object detection, [4]. This fact is due to two aspects, firstly Mask R-CNN uses a technique called RoIAlign, which improves on the previous RoIpool, (we refer to [4] for more details). Secondly, due to the multi-task learning of both the bounding box detection and object segmentation using masks, the precision of object detection is improved.

2.2.3 Mask R-CNN, Implementation

To implement Mask R-CNN we use the open source Mask R-CNN project by Matterport. The implementation involves the following steps:

1. Create a csv file containing annotations of each image.
2. Installation of the Mask R-CNN Library.
3. Define data-set object.
4. Create masks for each image.
5. Create Train / test data splits.
6. Training of the Mask R-CNN model.
7. Model evaluation.
8. Prediction of test images.

We start by installing the Mask R-CNN Library from,
https://github.com/matterport/Mask_RCNN.git , which allows for easy use of Mask R-CNN. From here we create a csv file containing the annotations for each image in the same way that was used for the original CNN. In order to use the mask-rcnn library we need to create a new data-set class which will contain functions to load the data and create the masks. The load data-set function will define the classes and the images in the data-set, along with choosing which images will be for testing and training.

In order to train the model, we need to define the labels, bounding boxes and masks for each object in the images. The csv file contains both the bounding boxes and labels for each object, hence, only the masks need to be created.To create the Masks for each image we use the bounding boxes from the csv file. For an $n \times m$ image we create an array with the same dimensions for each of the objects in the image, these will be the masks. Each array consists of 0's and 1's where every pixel inside the bounding box describing the object for that mask will be a 1 and 0 otherwise. From here we then use the load data-set function to load the images and annotations and create the train and test sets, of splits 75/25.

The Mask R-CNN model that we will be training uses resnet101 architecture. We define the configuration for the model, which includes information such as the target number of classes, the learning rate/momentum of the SGD optimiser, which we keep as the default values and the number of steps

per epoch. We train just the heads of the baseline model on 5 epochs using transfer learning from the pre-trained weights, mask_rcnn_coco. The model outputs multiple losses when training, we take note of the ones labelled mrcnn_class_loss and mrcnn_bbox_loss which will be the ones we want to optimise later.

With the model trained, we made some predictions on images from the test set. The images below show some elements from the test set with the ground truth bounding boxes and the predicted boxes. From figure 6 we can see that for an image containing only a few objects the model performs excellently in locating and classify all the objects. However, from figure 7 we see that the model struggles to locate harder objects, as it does not locate and classify the feet, which in this instance are more difficult as they are partially covered. The base model still performs very well on this image and shows that it is a great starting point for detecting people in our data-set.

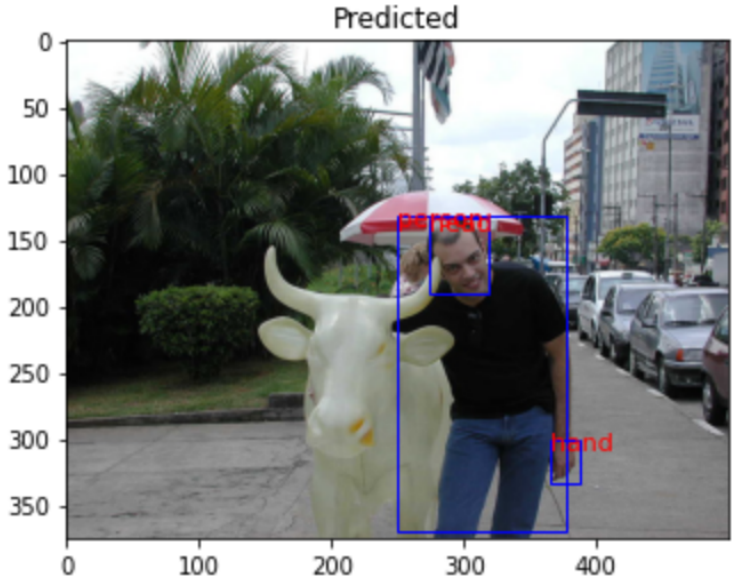
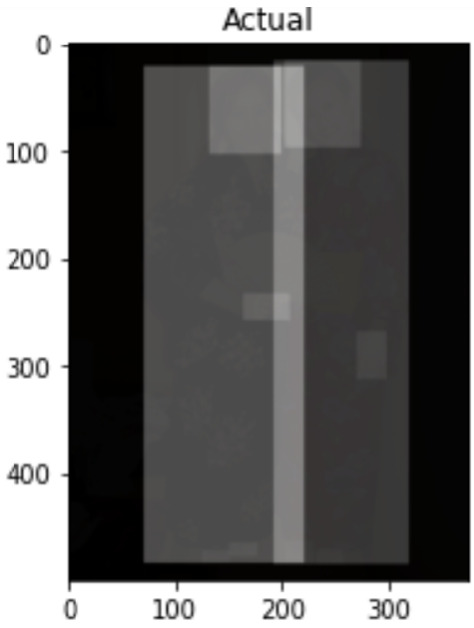


Figure 6: First image: Ground truth bounding boxes for image in test set. Second image: Predicted bounding boxes and labels by baseline model.



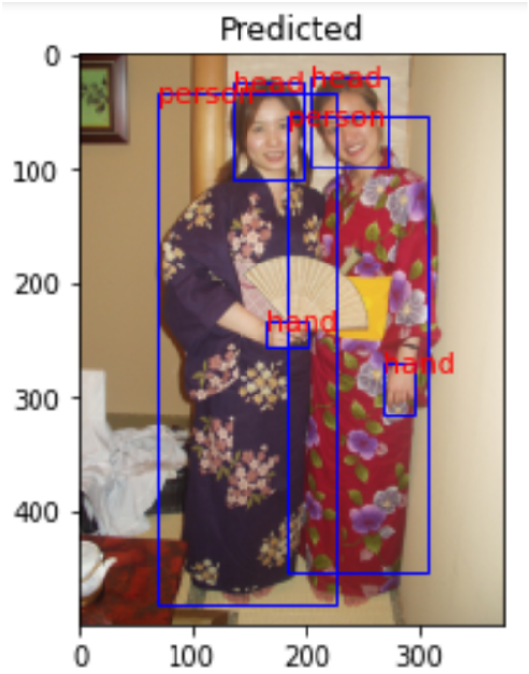


Figure 7: First image: Ground truth bounding boxes for image in test set. Second image: Predicted bounding boxes and labels by baseline model.

3 Experimental setting

To attempt to maximise the performance of the model we trained the model on the train set using a variation of settings and evaluated its performance using the test set. We use mean average precision, (mAP), as the metric for calculating the performance of the models on each of the data-sets. This will be discussed in more detail in section 4. The following items are some of the settings the model was trained under.

- Loss Weights
  - rpn_class_loss
  - rpn_bbox_loss
  - mrcnn_class_loss,
  - mrcnn_bbox_loss
  - mrcnn_mask_loss

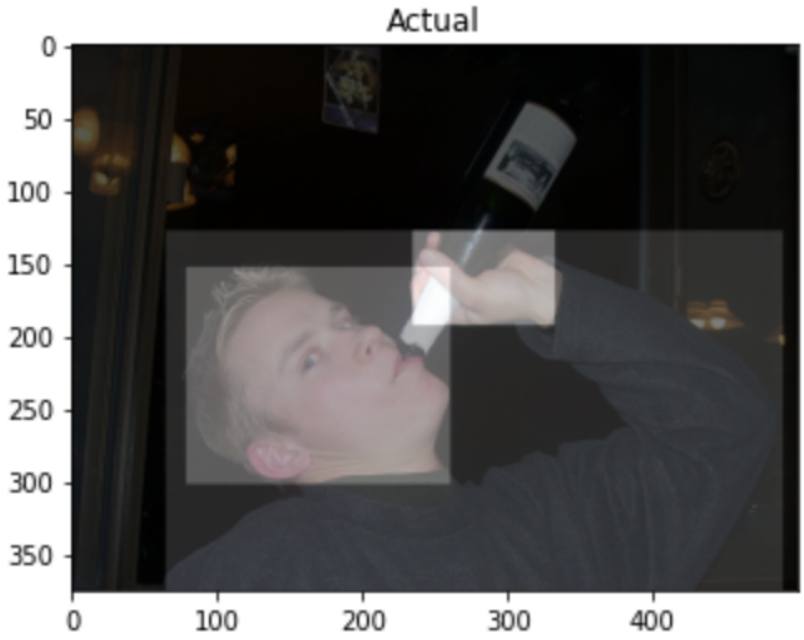
- Learning rate
- Momentum
- Regularisation parameter
- Tuned Model Layers
  - Heads: The RPN, classifier and mask heads of the network
  - 3+: Resnet stage 3 and up
  - 4+: Resnet stage 4 and up
  - 5+: Resnet stage 5 and up
  - All
- Model Backbone
  - Resnet101
  - Resnet50

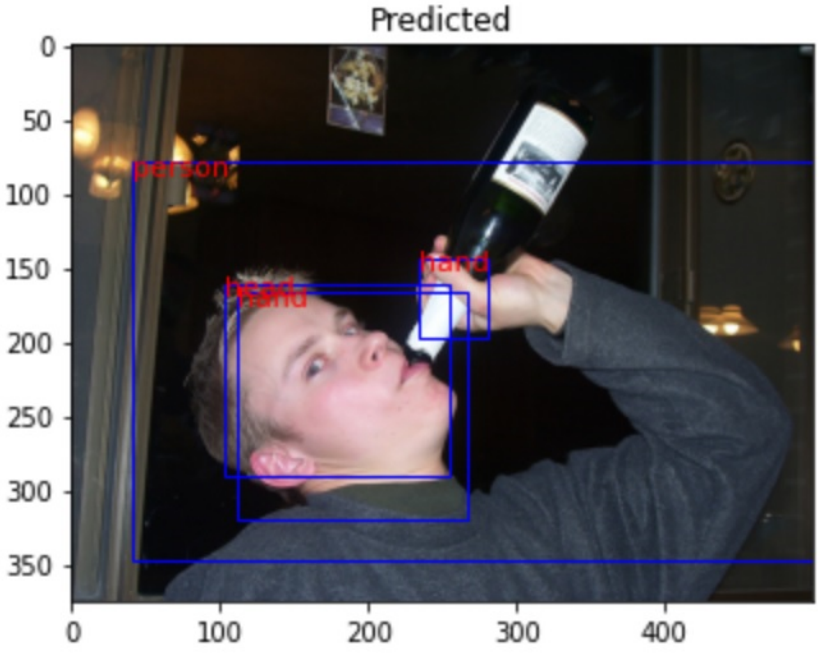
Mask R-CNN uses a loss function which is composed of the sum of different individual losses. These losses each have a hyper-parameter that determines the weight these losses should be assigned during the training. The optimizer used for Mask R-CNN can only make changes depending on the total loss, hence, to optimise an individual loss we must increase the size of it's weight relative to the others. By doing this, the loss will contribute more to the total sum, which will cause the model to focus more on that aspect. The trained baseline model gave the following set of losses after training on 5 epochs,

- rpn_class_loss: 0.0247
- rpn_bbox_loss: 0.2644
- mrcnn_class_loss: 0.2272
- mrcnn_bbox_loss: 0.1854
- mrcnn_mask_loss: 0.2549

The losses here that we are most interested in are the mrcnn_class_loss and the mrcnn_bbox_loss. The weight for the mrcnn_class_loss should be increased when objects are detected, but misclassified and the mrcnn_bbox_loss should be increased if objects are correctly classified, but the bounding box is not accurate enough, [5]. We test a range of values for these two losses by training the model on 10 epochs and validating with the test set. Each model is saved after each epoch of training, so that the model with the best performance on the test set can be used and consequently we can avoid using a model that has over fitted. The model with the best performance out of the set of weight values, $[1, 2, 2.5, 3, 3.5, 4]$, was the one using 3.5 for each of the loss weights this gave a best mAP value of 0.784 on the train set and 0.660 on the test set, comparing to the baseline model which had its best map value of 0.780 and 0.651 on the train and test sets respectively. Although the change in mAP between the this tuned model and the baseline is relatively small there were some noticeable visual differences between its predictions on images in the test set. Figure 8 shows that the baseline model both classifies correctly and incorrectly a head and shows that this is corrected when using the tuned model on this example.

Using these weights we then tested the different hyper-parameters of the optimizer. SGD is the default optimizer used by the Mask R-CNN model, with a learning rate of 0.001 and momentum of 0.9. We again train our model over 10 epochs using learning rate values in the range [0.0005,0.01] and momentum values in the range [0.7,0.95]. The parameters that gave the best validation mAP on the test set were, a learning rate of 0.005 and a momentum of 0.7. From here we then trained a model using the resnet50 backbone instead of resnet101. Figure 9 shows the graph of the mean average precision over 10 epochs of both the resnet50 and resnet 101 models on both the train and test set. From this we can see that the architecture containing more layers, resnet101, gives a much better performance than resnet50 and hence it will be the backbone of choice for our model. It is important to take note that the model using resnet50 was much more efficient in the time taken to train per epoch, however, the improvement in the training time does not outweigh the deterioration of the performance.


Actual
0
50
100
150
200
250
300
350
0
100
200
300
400


Predicted
person
hand
hand
0
50
100
150
200
250
300
350
0
100
200
300
400

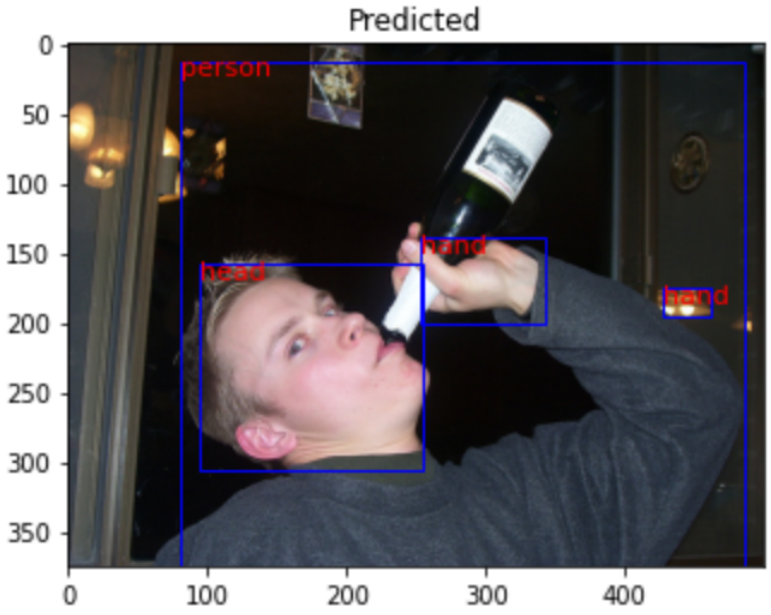


Figure 8: First image: Actual bounding boxes. Second image: Predicted bounding boxes and labels by the baseline model. Third image: Predicted bounding boxes and labels by the tuned loss weights model.

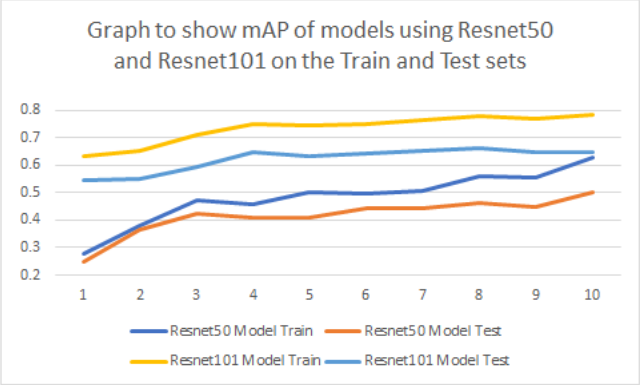


Figure 9: Graph showing the mAP of two models over 10 epochs, using Resnet50 and Resnet101 as the backbone.

The final steps of the tuning process was to experiment training different layers in the model rather than just the heads, along with trying different values of the weight decay parameter used for regularization. The results of training the different layers of the model showed a decrease in precision the more layers that were being tuned. Given this, we decided to keep the heads as the only layers to be tuned during training. Finally different values of the weight decay parameter in the domain [0.00005,0.005] were tested. Although all values gave similar results, the best performing model was the one using 0.00005 as it's weight decay.

4 Results

Before discussing the results of both the baseline and final models, we first explain how the metric, mean average precision , that was used for evaluating the performance is calculated.

4.1 Mean Average Precision Metric

Mean average precision (mAP), has become a common and accepted way of evaluating models for object detection. It can evaluate the performance for multiple objects at one time and helps simplify various evaluation dimensions to give a single result.

The first step in calculating mAP for a set of images is to calculate the precision and recall for each individual object in each image. To calculate these values we use the definitions of precision and recall below.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\text{TP}}{\text{number of predictions}},$$
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{number of ground truths}}.$$

For each detection made for the image we calculate the intersection over union, (IoU) between it and the ground truth for that prediction. If the intersection is above than a threshold, usually 0.5, then the prediction is classed as a true positive (TP). Conversely, if the intersection is less than 0.5

or there is duplicated bounding box prediction, then the detection is classed as false positive (FP). When the prediction is wrong classification or there is no prediction at all, then the detection is classed as false negative (FN).

With precision and recall, the precision-recall (PR) curve is then plotted, and the AP is calculated by taking the area under the curve. The higher the mAP is, the better the model prediction is, [6].

4.2 Model Results

Using the parameters tested in the experimental setting, the details of the final model chosen are:

- Loss Weights
  - mrcnn_class_loss: 3.5
  - mrcnn_bbox_loss: 3.5
- Optimiser: SGD
  - Learning rate: 0.005
  - Momentum: 0.7
- Weight Decay: 0.00005
- Tuned Layers: Heads
- Model Backbone: Resnet101

We trained this model and the baseline model over 12 epochs, calculating the Mean average Precision at each epoch using a minimum IoU threshold of 0.5. Figure 10 below shows the results of mean average precision at each epoch for both the Baseline and final model on the train and test set respectively. From the graph we can see that the final model outperforms the baseline at almost every epoch on both data sets, only falling lower at epoch 9. The highest mAP score for the final model on the test set was 0.665, which was an increase of just over 2%, over the highest mAP for the baseline model, which was 0.651.

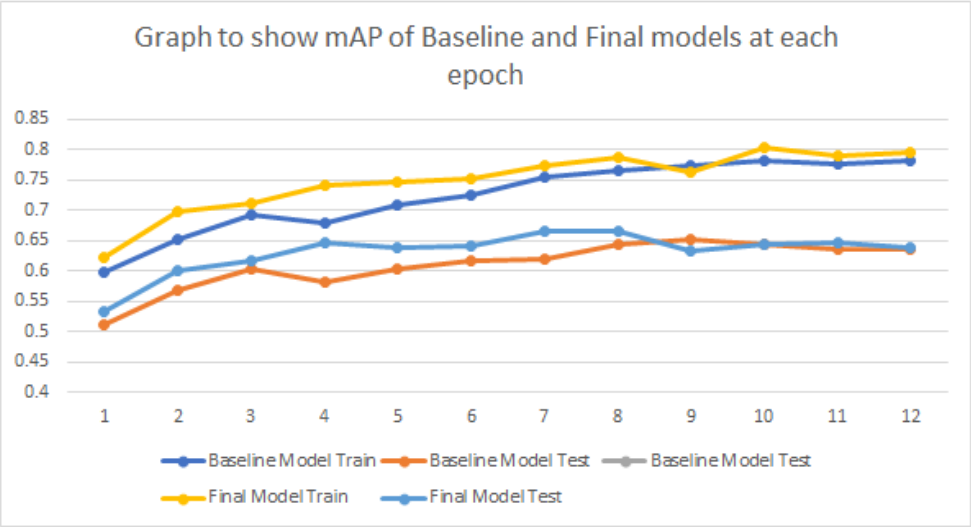


Figure 10: Graph showing the mAP of the baseline and final models on each of the train and test sets over 12 epochs

The mean average precision was then calculated for both models using a range of IoU values in the domain [0.5,0.95]. As expected the table below shows the average precision decreases the higher the IoU threshold. This is due to more predictions being discounted as they do not satisfy the threshold value and hence, the less number of true positive results relative to the number of ground truth instances.

Increasing the threshold value can have benefits in removing any predictions that may be decent, but not precise. This can help remove things like multiple detentions of the same object, but usually it will remove more useful predictions than non useful one. Figures 11 and 12 below, show some predictions by the final model on a test image using various thresholds. Clearly as the threshold increases, less objects are being marked with a location, however, between 0.5 and 0.6 IoU threshold value an incorrect prediction for a foot is removed.

IoU value	Baseline Model		Final Model	
	Train	Test	Train	Test
0.50	0.765	0.645	0.788	0.665
0.55	0.741	0.617	0.762	0.63
0.60	0.706	0.566	0.732	0.593
0.65	0.651	0.505	0.688	0.54
0.70	0.58	0.439	0.624	0.464
0.75	0.473	0.326	0.52	0.373
0.80	0.328	0.21	0.383	0.268
0.85	0.159	0.096	0.204	0.128
0.90	0.031	0.015	0.055	0.027
0.95	0	0	0.001	0

Table 1: table showing the mean average precision of the baseline and final models on the train and test sets at various IoU thresholds

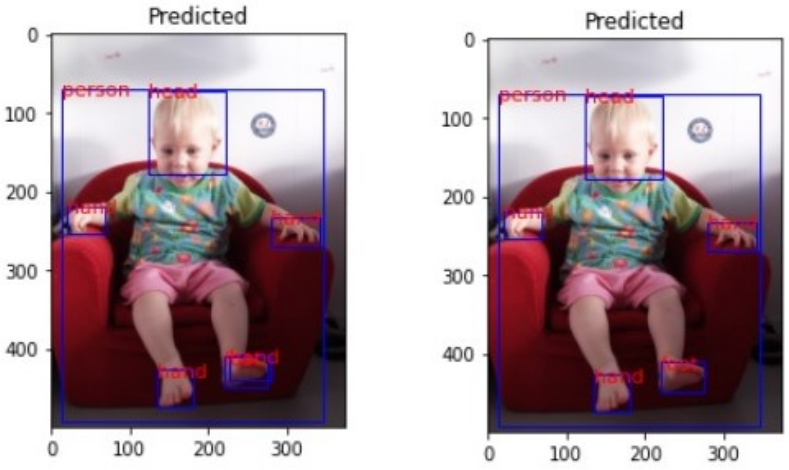


Figure 11: Left image: Predicted bounding boxes and labels by final model for IoU = 0.5. Right image: Predicted bounding boxes and labels by final model for IoU = 0.6.

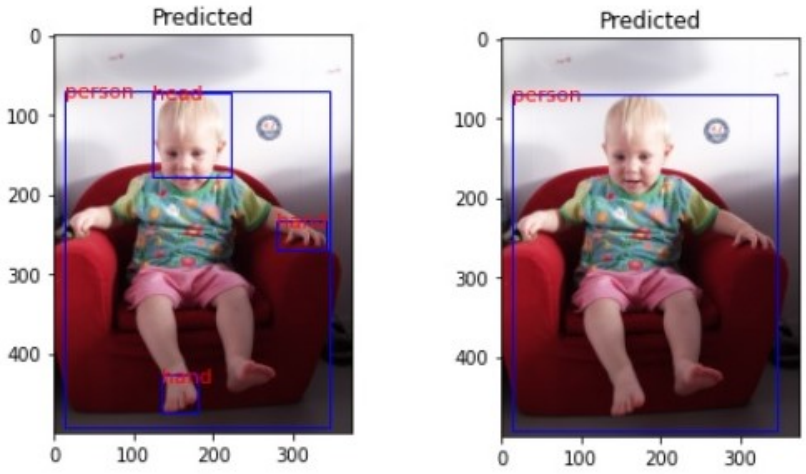


Figure 12: Left image: Predicted bounding boxes and labels by final model for IoU = 0.75. Right image: Predicted bounding boxes and labels by final model for IoU = 0.9.

Finally we calculate the Recall Precision and F1 scores for each of the models on the test set. As shown in table 2 the precision, recall and f1 scores all decrease as the IoU threshold increasea. The recall for the final model is better than the recall for the baseline, showing that it correctly predicts more ground truth boxes than the baseline does. The Precision of the baseline model however, is generally better than the final model, showing that it makes less predictions that are under the IoU threshold than the final model.

IoU value	Baseline Model			Final Model		
	Recall	Precision	F1	Recall	Precision	F1
0.50	0.775	0.701	0.736	0.799	0.635	0.708
0.60	0.706	0.634	0.668	0.728	0.570	0.639
0.70	0.569	0.509	0.537	0.589	0.515	0.515
0.75	0.477	0.427	0.451	0.498	0.390	0.438
0.90	0.072	0.062	0.067	0.100	0.082	0.090

Table 2: Recall, Precision and F1 score of the baseline and final models on the test set, at different IoU thresholds.

5 Analysis

The numerical results of the models show that the model has indeed been able to learn to detect and classify objects. However, the mean average precision obtained is not as high as one might expect or desire from an object detection model. Upon visual inspection of the predictions made by the model on images in the test set, we can see that the model seems to be performing excellently. Figures 13 and 14 below shows some of these instances where the model has made some excellent predictions, being able to detect almost all objects in the images.

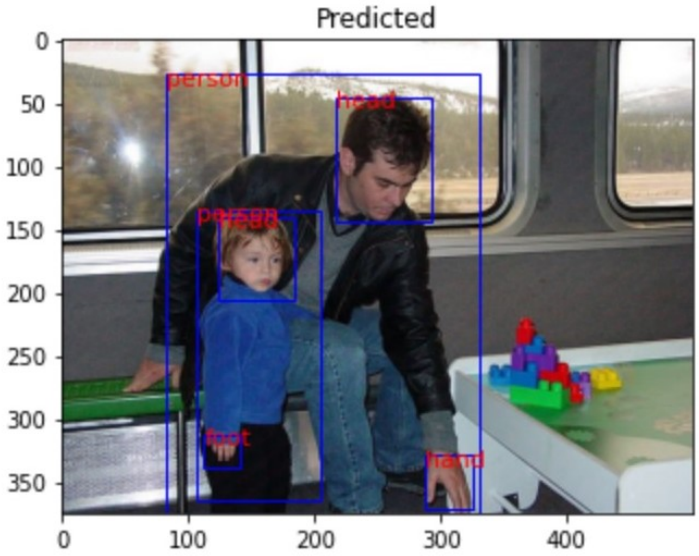


Figure 13: Final model predictions on an image in the test set

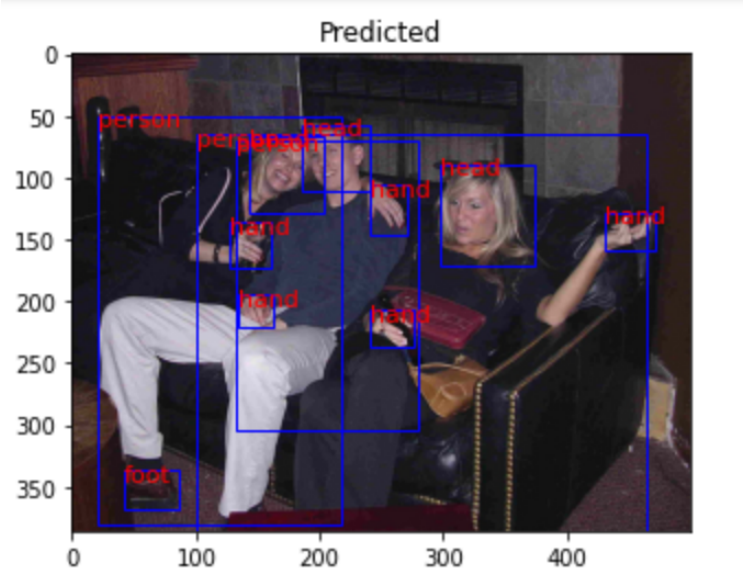


Figure 14: Final model predictions on an image in the test set

On further inspection of these images, we see that the ground truth labels differ quite a lot of from the predicted ones. Figure 15 shows the same test images with their corresponding ground truth labels. From this we see that the true labels only describe one person and their features in the images, even though we can clearly see multiple people. Thus, the reason why the numerical and visual results do not seem to line up, is due to the fact that although the model labels almost all all objects in the images, it is calculated as having a poorer average precision, due to the predictions being marked as false positives.

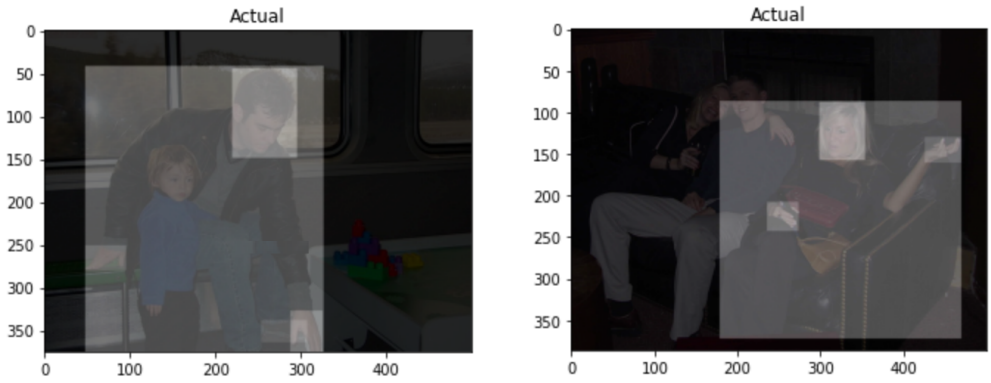


Figure 15:

Furthermore, these images with ground truth labels that do not describe all the objects in the image can cause a wide variation in the mAP of trained models. When creating the train and test sets ,to train and evaluate the model, we used the same images in each of these sets for each different model created. By doing this, it allowed for models to be compared to each other in terms of their performance. However, if the train and test sets are created containing different images from the data-set, then the calculated precision on each of the sets changes. This fact is due to the variation in the number of images with "bad" labels in each of the two sets. If the test set contains less of the poorly labelled images, then the mAP increases. This causes the train mAP to decrease, as the train set will contain more of the poorly labelled images. Overall the annotations for the images are a bottleneck for getting an accurate reading of the performance of the model, and hence, any future

work should aim to fix this problem before anything else.

From the results in Table 2 we can also see that the Precision score for the final model is less than that of the baseline model. Although it is not conclusive why this may have occurred, it is possible that if the final model gives an increase in the number of predictions then the Recall will increase and the Precision will decrease. To prove this, we recall the definitions of Precision and Recall:

$$\text{Precision} = \frac{\text{TP}}{\text{number of predictions}},$$

$$\text{Recall} = \frac{\text{TP}}{\text{number of ground truths}}.$$

Let G be the number of ground truths and P be the number of predictions. Then we have:

$$\text{Precision} = \frac{\text{TP}}{P}, \ \text{Recall} = \frac{\text{TP}}{G}.$$

If we increase the number of predictions as stated before, i.e $P \to P + \epsilon$, $\epsilon > 0$, then TP, the number of True Positives increase, i.e, $\text{TP} \to \text{TP} + \delta$, $0 \leq \delta \leq \epsilon$. Hence, we have,

$$\text{Recall} = \frac{\text{TP}}{G} \to \frac{\text{TP} + \delta}{G},$$

and

$$\frac{\text{TP} + \delta}{G} \geq \frac{\text{TP}}{G} \text{ , as } \delta \geq 0,$$

meaning that Recall will be the same or greater if the number of predictions increases. Furthermore,

$$\text{Precision} = \frac{\text{TP}}{P} \to \frac{\text{TP} + \delta}{P + \epsilon}.$$

We want to show that Precision can decrease as the number of predictions increases, therefore, we must show that,

$$\frac{\text{TP} + \delta}{P + \epsilon} < \frac{\text{TP}}{P}$$

Rearranging, we get,

$$P \cdot (\mathrm{TP} + \delta) < \mathrm{TP} \cdot (P + \epsilon),$$

$$(P) \cdot \delta < \mathrm{TP} \cdot (\epsilon),$$

$$\frac{\delta}{\epsilon} < \frac{\mathrm{TP}}{P} = \text{Precision}.$$

Thus the Recall will increase and the Precision will decrease, if the ratio between the increase in the number of predictions, ϵ, and the increase in the number of True Positives, δ, is less than the original precision value.

Finally we will analyse some of the errors that the model makes on images in the test set. One of the main types of errors that occurred were feet being classified as hands and vice versa. This was particularly noticeable in images of younger children or babies, where the visual difference between their feet and hands are less than that of an adult. The model also had difficulty locating hands and feet that were covered partially by clothing, however, this is to be expected as it can be more difficult for even a human to locate them in certain images.

Some images contained many instances of people, such as figure 16 below, as stated in section 1 of this report. These types of images were classed as outliers when looking at the data-set as a whole and hence, were the images that were thought to cause the model more trouble. Although the model performed very well in locating many of the people, it struggled with detecting their features, head, hands and feet. Again this does not come as much of a surprise, as generally the more people in an image the smaller each person must appear, hence, the individual body parts can be very small and difficult to detect. Overall, errors for these types of images were minimal, relative to the number of objects that were correctly located and classified.

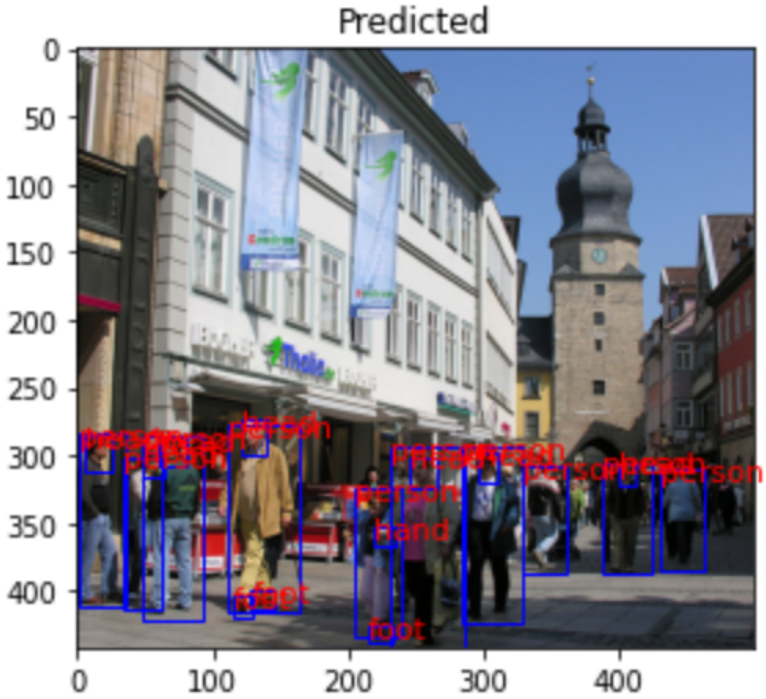


Figure 16:

6 Literature review / Related work

The aim of our project was to create a convolutional neural network model which could detect and locate certain objects in images. A common paradigm to address this problem is to train object detectors which operate on a sub-image and apply these detectors in an exhaustive manner across all locations and scales, suggests by Christian Szegedy, Alexander Toshev and Dumitru Erhan(2013). Therefore, before starting the project, it was important for us to focus on some important articles which closely relate to the object detection problem. The first one, which is the main focus of this report is Mask R-CNN, where we introduced some theory and a step by step procedure for object detection using Matterport's Mask R-CNN implementation (2017). Secondly, Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun (2015), [3] give a good introduction about the theory of Faster r-cnn towards real-

time object detection which give us some ideas about the project. The third provides an example of the use of Mask R-CNN for real-world applications which is from Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross B. Girshick(2017), [4]. At the same time, Google 's Inception series models also perform well on this task. For exampleChollet François(2016), [7] gives the method to develop the Inception model, which is named the Xception model. All of these articles are talking about different viewpoints around convolutional neural network and how can neural network work for our VOC2012 data-set, which are all worthy of referencing in this project.

6.1 Deep Neural Networks for Object Detection

To perform object Detect we may need to think about using Deep Neural networks, DNN. Using deep learning and DNN for target detection is unlike traditional shallow neural networks. Deep neural networks can learn more complex models, not only for classification, but also for accurate positioning. You can predict a bounding box in a given image, and more precisely, it will also generate a binary mask in the box, which is proposed by Christian Szegedy, Alexander Toshev and Dumitru Erhan(2013), [8]. Therefore, for the problem of target detection, it is necessary to use a deep neural network. In past experiments, DNNs have achieved good results in the VOC challenge, Christian Szegedy, Alexander Toshev and Dumitru Erhan have shown this approach can have excellent performance on the VOC2011 data-set. They also performed training on the larger VOC2012 training set, which showed a performance similar to that of most of the state-of-the art models.

6.2 Advantages of Mask R-CNN

It has been determined that deep neural networks can be used well for object detection. However, this problem is not so easy to solve. The size of the objects and the angle the objects are placed can vary widely. Moreover, the pose can be uncertain and can appear anywhere in the picture along with objects possibly being from multiple different categories. There are many methods based on DNN for solving these problems and knowing what kind of object detection algorithm is best for our VOC2012 dataset is not such an task. Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross B. Girshick (2017), [4], explain that Mask R-CNN adopts the same two-stage procedure, with an identical first stage (which is RPN). In the second stage,

in parallel to predicting the class and box offset, the multi-task learning of both the bounding box detection and object segmentation using masks, increase the precision of object detection. The great performance made Mask R-CNN an excellent choice for performing both object detection and object segmentation. Furthermore, it is very similar to Faster R-CNN(Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun(2015)). In the field of Faster R-CNN, Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun have done many experiments on PASCAL VOC, where they show high performance and low cost on each of the layers from their table and figure. In both of their studies, they show that Mask R-CNN is very fast, with better speed/accuracy trade-offs achieved.

6.3 Implementation of Mask R-CNN

Their article does not explain in detail how to implement Mask R-CNN. Matterport(2017), [9], implements a Mask R-CNN framework for Object Detection and Segmentation, which provide us the ability to produce a lot of visualizations and allow running the model step by step to inspect the output at different points. Matterport also give some example projects that made by extension Mask R-CNN model with other datasets. Our project is mainly based on the Mask R-CNN library by Matterport. But at the same time, it is inseparable from the theory from Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross B. Girshick (2017).

7 Conclusion and Future work

Overall the Mask R-CNN model that we created and trained performed very well. Although the numerical results were below expected, the visual results seemed very promising...
Overall, in this project, we mostly have created and trained a Mask R-CNN model with the specific VOC2012 dataset. We have also completed some experiments, mainly to demonstrate the efficiency of trained model. Before starting this evaluation, we have carefully analyzed the data-set and choose a variation of model settings with the aim to improve, the time efficiency of training and testing this model without impacting the performance, and also the precision of the model when performing object detection.

The results prove the conclusion that our Mask R-CNN model performs well and the visual results seem to be very promising. We also show how the variation in IoU threshold changes whether objects are being correctly marked. However, the mean average precision obtained is not as high as expected from an object detection model. Furthermore, it also produces some consistent types of errors when making predictions on images in the test set. After some careful analysis we reveal that the errors for these types of images were minimal, relative to the number of objects that were correctly located and classed.

The next step of this project would be to attempt to improve the accuracy of the model. One of the main issues that needs to be addresses is the poorness of the ground truth labels of the data-set. This could be easily adjusted, however, the process would be time consuming and hence has been omitted from the report. Finally, Mask R-CNN is only one of many possible techniques that can be used for object detection, hence, future work should include exploration of more methods and there benefits and hindrances over Mask R-CNN.

References

[1] Fei-Fei Li Justin Johnson Serena Yeung. Lecture 11: Detection and Segmentation. `http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf`, 2017. [Online; accessed 03 April 2020].

[2] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[3] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1137–1149, 2015.

[4] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask r-cnn. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[5] Ravikiran Bobba. Taming the Hyper-Parameters of Mask RCNN. `https://medium.com/analytics-vidhya/taming-the-hyper-parameters-of-mask-rcnn-3742cb3f0e1b`, 2019. [Online; accessed 06 April 2020].

[6] Ren Jie Tan. Breaking Down Mean Average Precision (mAP). `https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52`, 2019. [Online; accessed 15 April 2020].

[7] Francois Chollet. Deep learning with separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.

[8] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In *Advances in neural information processing systems*, pages 2553–2561, 2013.

[9] Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow. `https://github.com/matterport/Mask_RCNN`, 2017. [Online; accessed 02 March 2020].