Cardiff University

Group G9 Project



Object Localisation

*Supervisor:* Dr. Yukan Lai April 2020

**Contents**

1. [Introduction 2](#_TOC_250017)
2. [Description of the task and dataset 3](#_TOC_250016)
3. [Methodology 5](#_TOC_250015)
   1. [Convolutional Neural network 5](#_TOC_250014)
   2. [Mask R-CNN 9](#_TOC_250013)
      1. [Faster R-CNN 9](#_TOC_250012)
      2. [Mask R-CNN, Theory 10](#_TOC_250011)
      3. [Mask R-CNN, Impementation 11](#_TOC_250010)
4. [Experimental setting 13](#_TOC_250009)
5. [Results 18](#_TOC_250008)
   1. [Mean Average Precision Metric 18](#_TOC_250007)
   2. [Model Results 18](#_TOC_250006)
6. [Analysis 22](#_TOC_250005)
7. [Literature review / Related work 27](#_TOC_250004)
   1. [Deep Neural Networks for Object Detection 27](#_TOC_250003)
   2. [Advantages of Mask R-CNN 28](#_TOC_250002)
   3. [Implementation of Mask R-CNN 28](#_TOC_250001)
8. [Conclusion and future work 29](#_TOC_250000)

# Introduction

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 if locating the objects in the image. In this project we focus mainly on the localisation aspect, however, the models used will perform both parts of object detection.

# 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. For this project we use a subset of the Ima- geNet Object Localization dataset from Kaggle’s ImageNet Object Local- ization competition. 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.

The 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. 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.

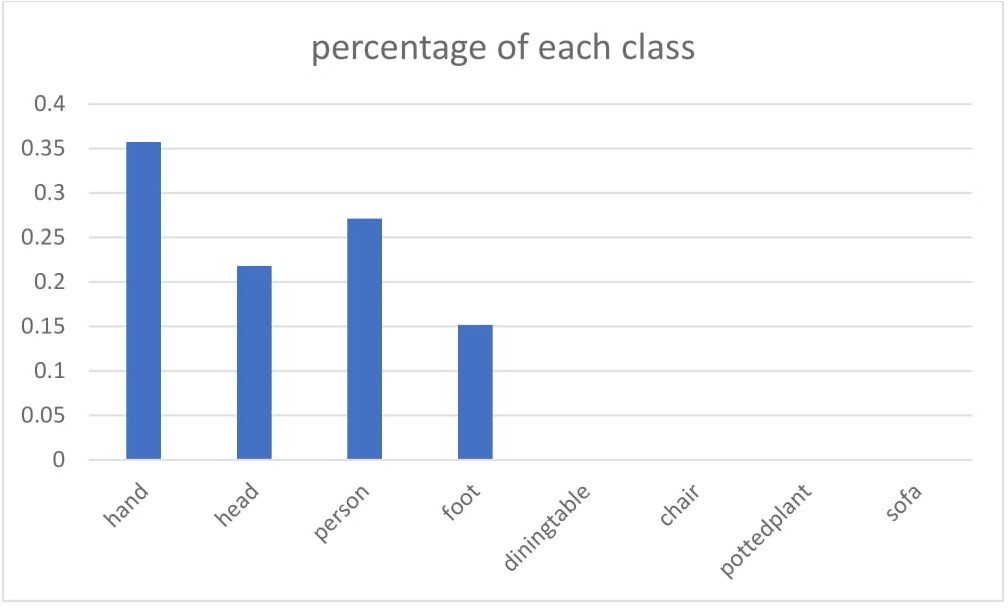


Figure 1: percentage representation of each class out of the total amount of objects

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 ob- jects across these images. Because of this, 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 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 the data has a mean of 6 and a standard deviation of 4. From this 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 IQR and Q1 1*.*5 IQR, where IQR is the inter-quartile range and Q1, Q3 and the first and third quartile respectively. We have Q1 = 4 and Q3 = 8, hence there are no outliers below Q1 as 4 (1*.*5 (8 4)) = 2 *<* 3, (3 being the

*− ·*

*·*

*− · − −*

smallest number of objects). However, all values above 8+(1*.*5 (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, however, any model may be more likely to make errors when performing object detection on these images.

* *−*

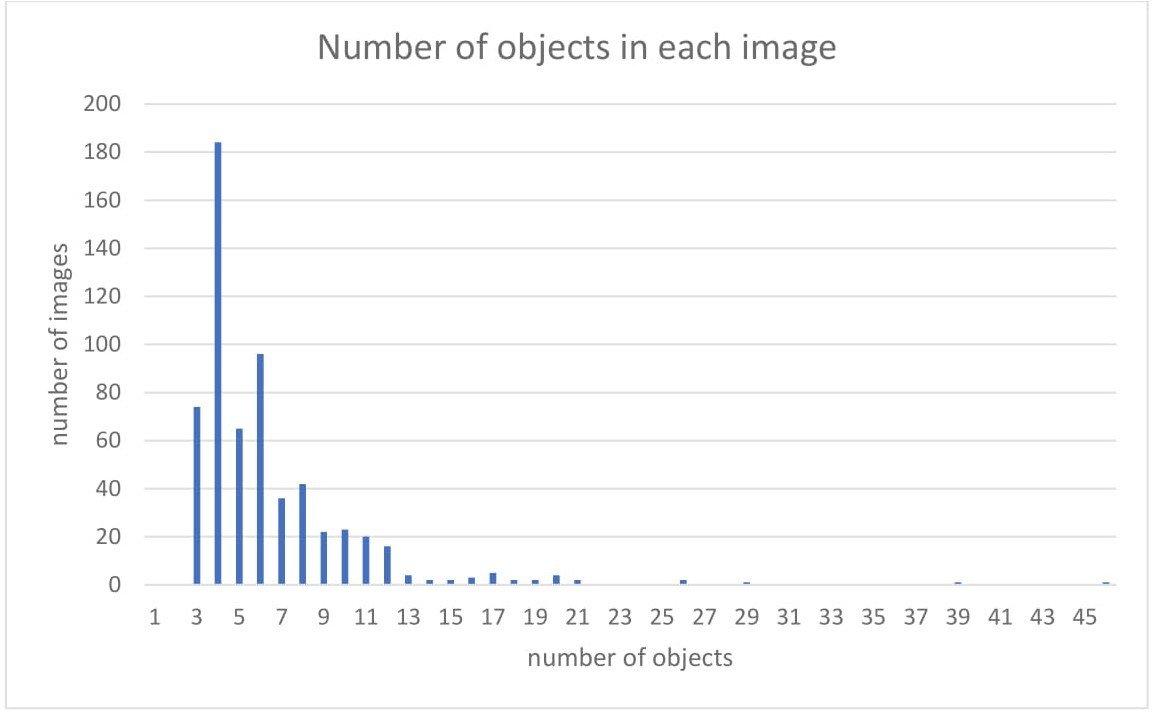


Figure 2: Graph showing the number of images containing different numbers of objects.

# Methodology

For this project we chose 2 approaches to attempt to locate 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 ob- ject 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 implemen- tation.

## Convolutional Neural network

As a starting point for the project, it was important to understand the imple- mentation 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.

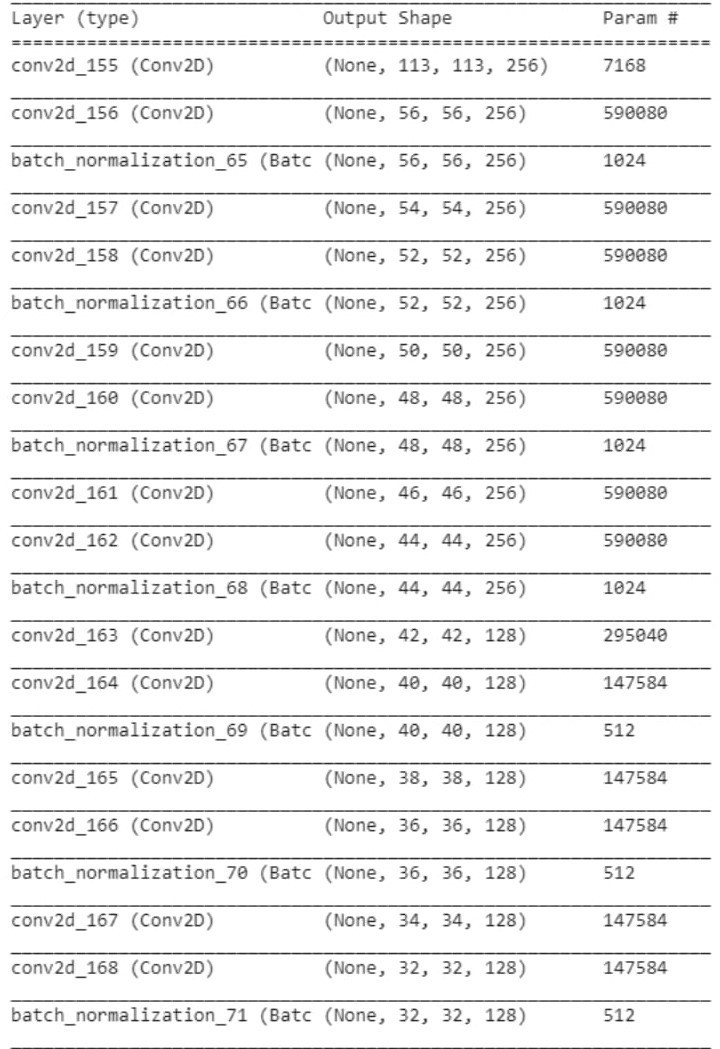
In order to create this implementation we started by forming a data-set using the images specified in the text file supplied with the original data. From this data-set, a csv file was created that contained information about each of the images. The information contained in the csv file was, the filename, width and height of each image, along with the object name and 4 co-ordinates or each object in each image. This information was gathered using the anno- tation files corresponding to each image in the data-set and a custom script utilising the ”ElementTree” module from ”xml.etree” in Python.

From our data-set, a subset was extracting by taking images that only con- tained one person. This subset would become the set for training and test images for this model. The images were turned into a numpy array, resized to 228x228x3 and had their values normalised by dividing by 255. The ground truth labels were then created, where each label was a set of 4 numbers cor- responding to the co-ordinates of the bounding box describing the person in the image.

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 dense 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. 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. hence for this reason we look to more complex model architectures to tackle our object detection problem.



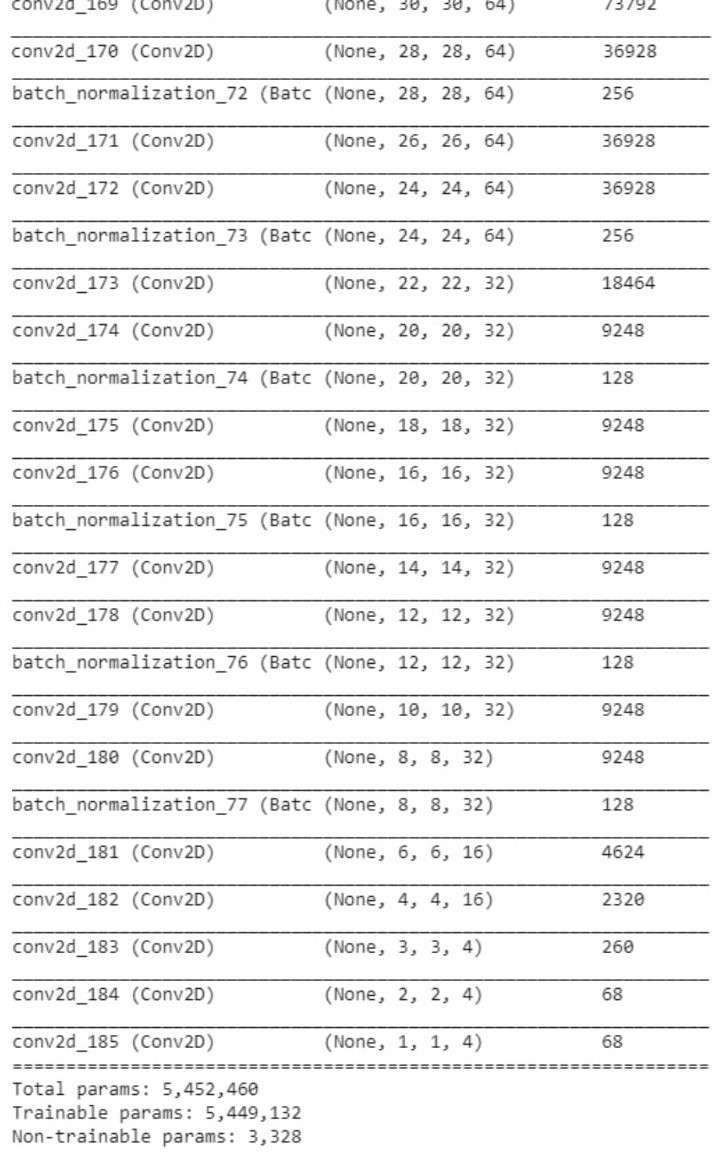


Figure 3: Simple Convolutional Neural Network Architecture

## Mask R-CNN

Mask R-CNN, (Region Convolutional Neural Network), is a method for ob- ject detection and instance segmentation, which is built of off faster R-CNN, hence, before we discuss the ideas behind this method, we shall first take a look at the fundamentals of Faster R-CNN.

### 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.

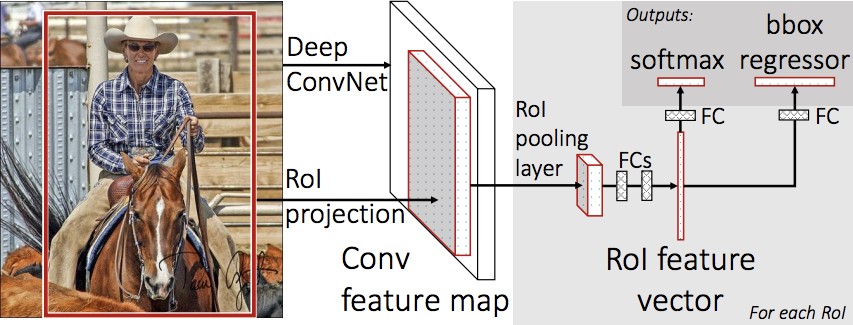


Figure 4: Fast R-CNN architecture

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, [1]. With this, the model gives two final outputs for each of the region proposals, a softmax to classify the

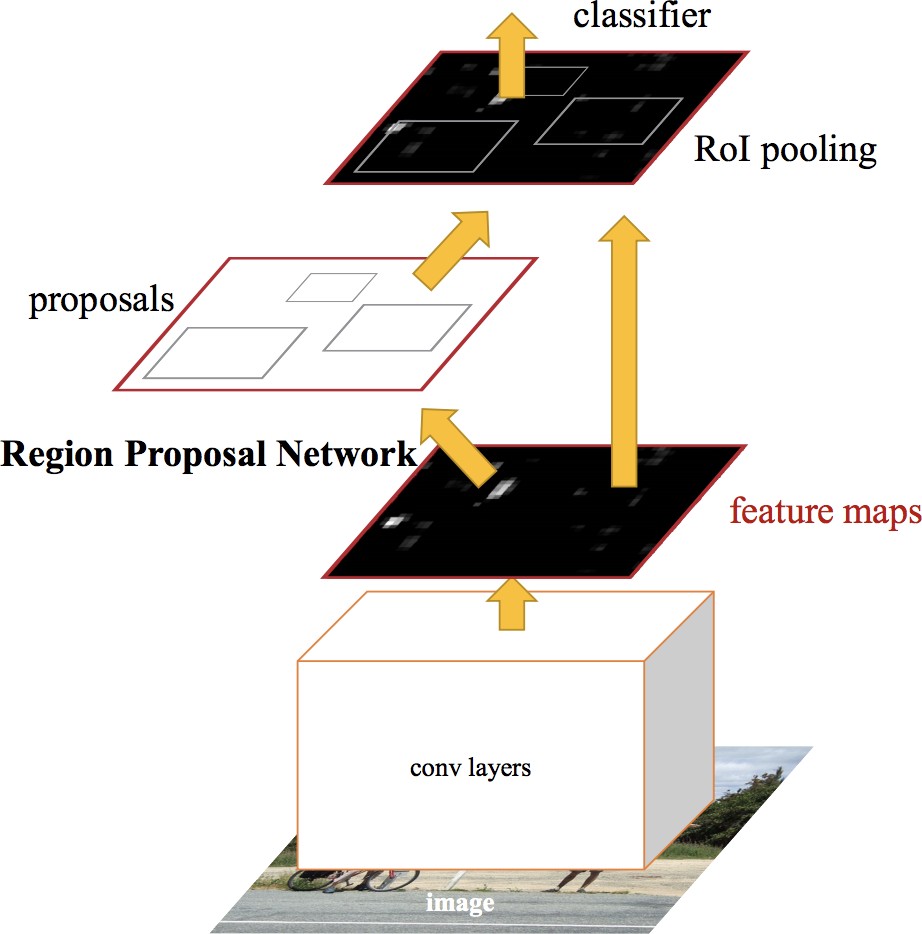


Figure 5: Faster R-CNN architecture

object and a regressor to determine the size and position of the bounding box.

### Mask R-CNN, Theory

Mask R-CNN works in exactly the same way as Faster R-CNN except it out- puts a 3rd object, an object mask, in addition to the class label and bounding box. 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, [2]. 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 [2] 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.

### Mask R-CNN, Impementation

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

* + - 1. Installation of the Mask R-CNN Library.
      2. Create csv file containing annotations of each image.
      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 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* ***×*** *m* image we create an array with the same dimensions for each object in the image, which 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 architec- ture. 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 us- ing transfer learning from the pre-trained weights, mask rcnn coco. The model outputs multiple losses when training, we take note of the ones la- belled 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 for 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 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.

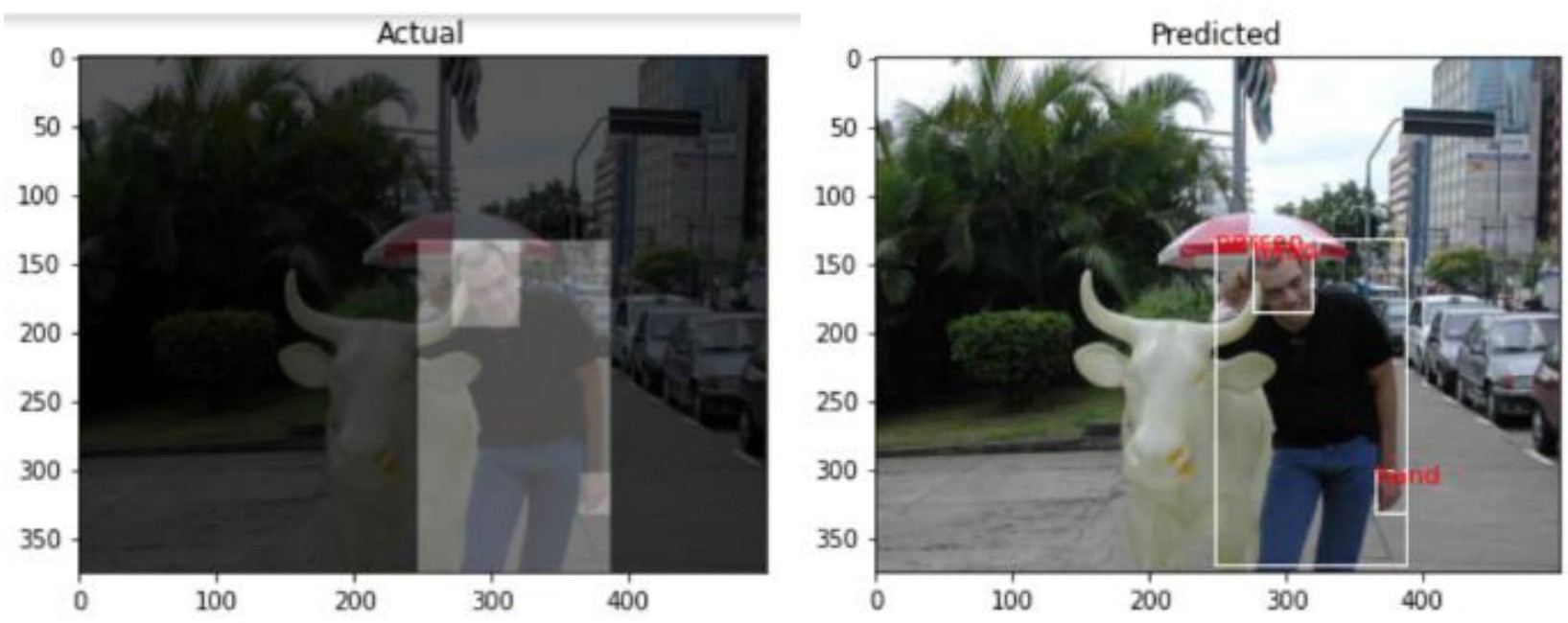


Figure 6: Left image: Ground truth bounding boxes for image in test set. Right image: Predicted bounding boxes and labels by baseline model.

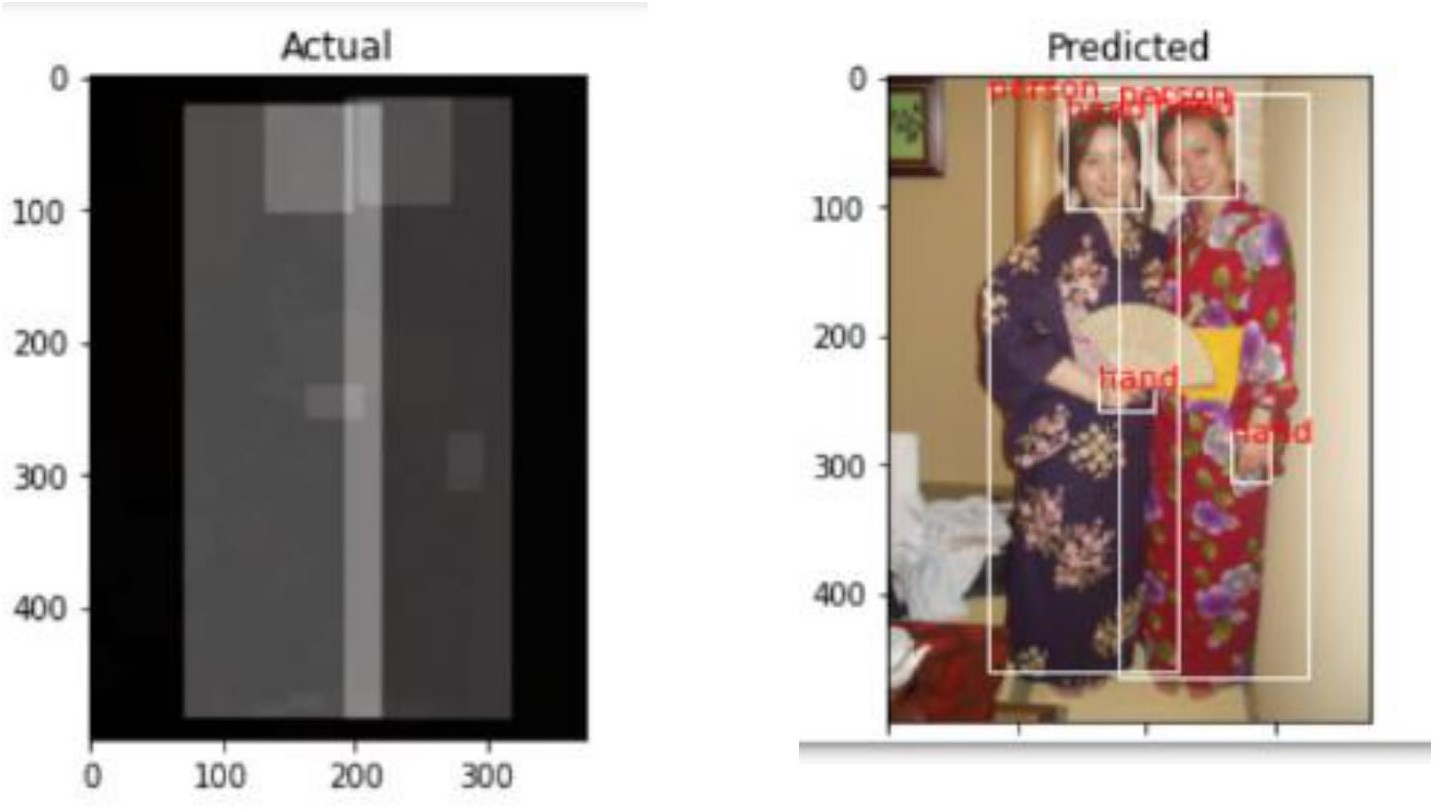


Figure 7: Left image: Ground truth bounding boxes for image in test set. Right image: Predicted bounding boxes and labels by baseline model.

# 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, [3]. 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 mAP of 0.784 on the train set and 0.660 on the test set, comparing to the baseline model which had a map of 0.780 and 0.651 on the train and test set 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 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 decrease improvement training time does not outweigh the deterioration of the performance.

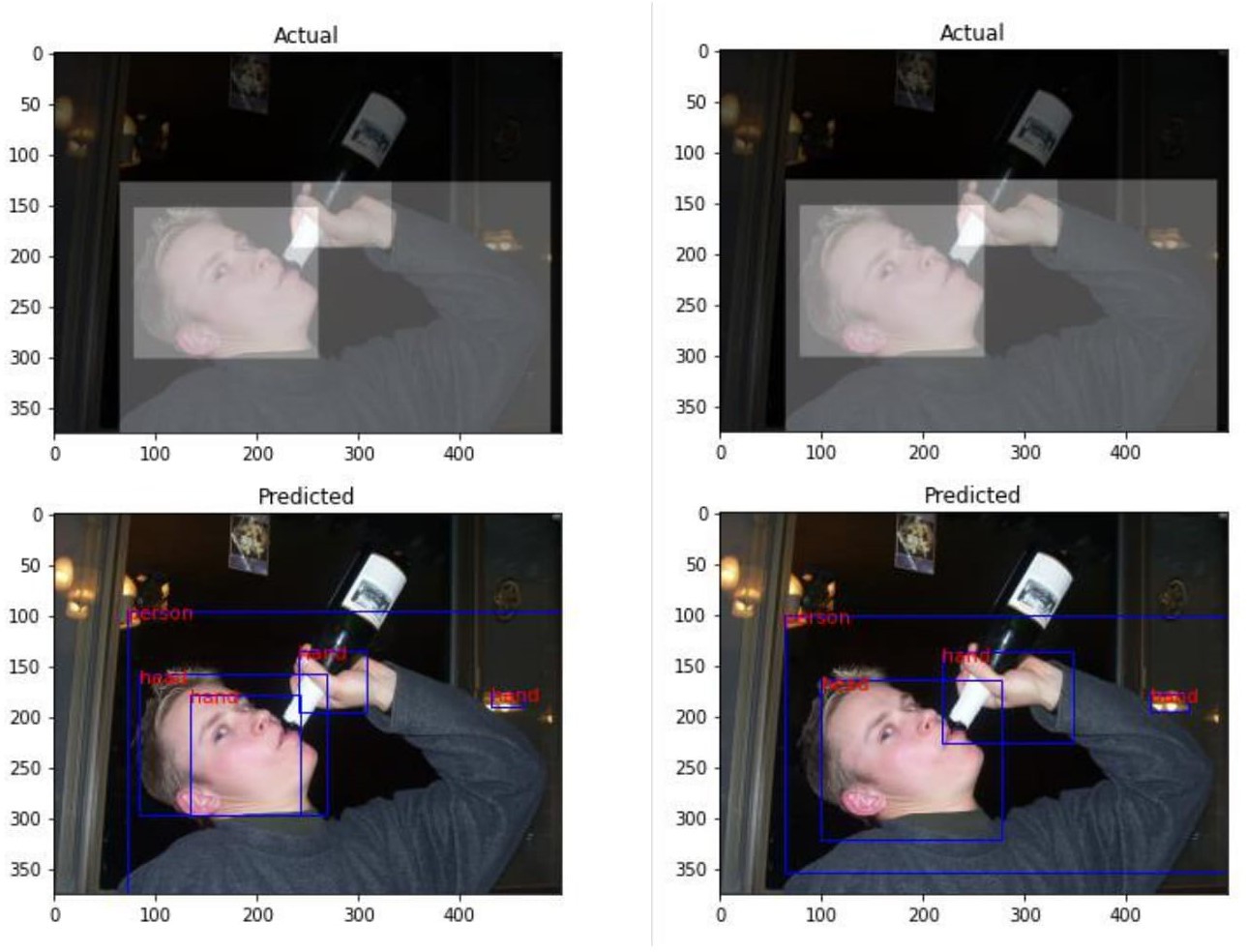


Figure 8: Left images: Actual and predicted bounding boxes and labels for baseline model. Right images: Actual and predicted bounding boxes and labels for tuned loss weights model.

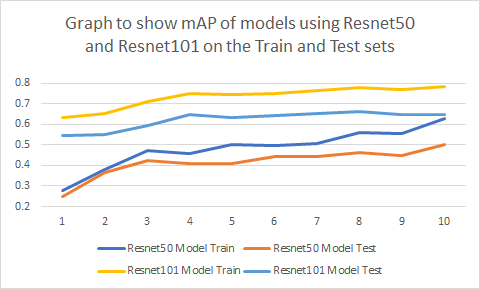


Figure 9: Graph showing the mAP of two models using Resnet50 and Resnet101 as the backbone over 10 epochs.

The final steps in 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.

# 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.

## 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.

Precision =

Recall =

TP

*,*

TP + FP

TP

*.*

TP + FN

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.

Reference: <https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52>

## 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.

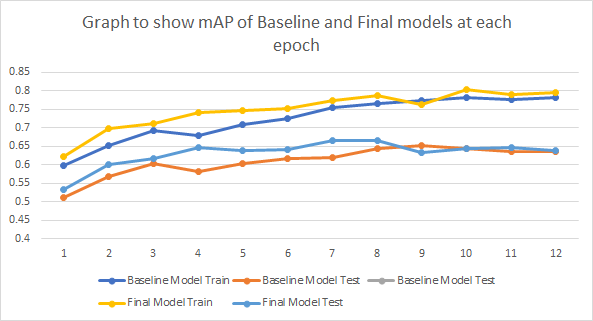


Figure 10: Graph showing the mAP of the baseline and final models on each of the train and test sets over 12 epochs

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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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

Increasing the threshold value can have benefits in removing any predictions that may be decent, but not precise. This can help remove things like mul- tiple detentions of the same object, but usually it will remove more useful predictions than non useful one. Figures 11 and 12 below, show some pre- dictions 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.

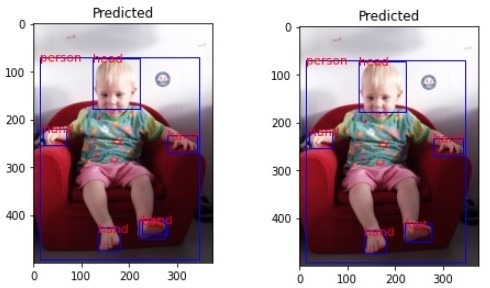


Figure 11: Left image: Predicted bounding boxes and labels by final model for IoU = 0.5. Right image: Predicted bounding boxes and labels by final model for IoU = 0.6.

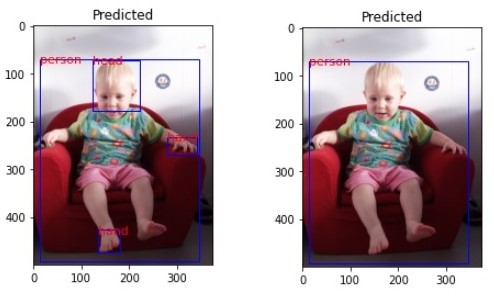


Figure 12: Left image: Predicted bounding boxes and labels by final model for IoU = 0.75. Right image: Predicted bounding boxes and labels by final model for IoU = 0.9.

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 increase. 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, however, of the baseline model 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.

# 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.

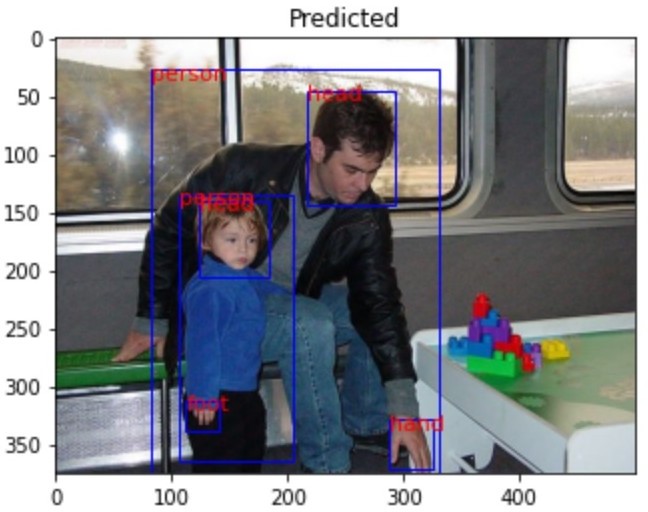


Figure 13: Final model predictions on an image in the test set

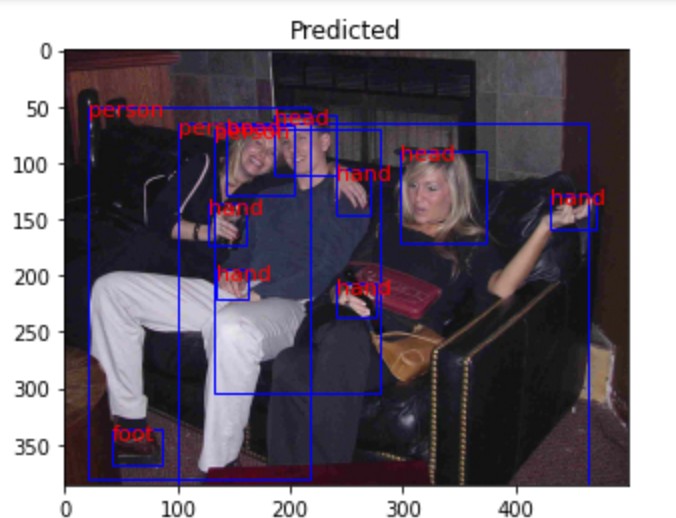


Figure 14: Final model predictions on an image in the test set

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 results 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.

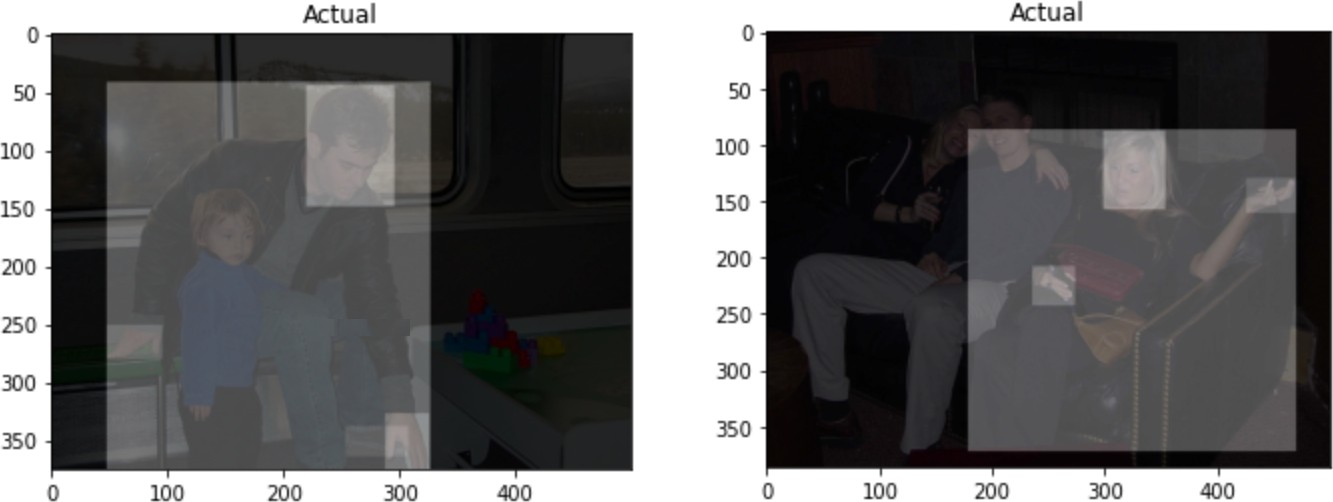


Figure 15:

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 number in predictions then the Recall will increase and the Precision will decrease. To prove this, we recall the definitions of Precision and Recall:

Precision =

Recall =

TP

*,*

number of predictions

TP

*.*

number of ground truths

Let *G* be the number of ground truths and *P* be the number of predictions, then we have:

Precision =

TP TP

*,* Recall = *. P G*

If we increase the number of predictions as stated above, i.e *P → P* + *s* , *s >* 0, then TP, the number of True Positives increase, i.e, TP *→* TP + *δ* , 0 *≤ δ ≤ s*. Hence, we have,

and

Recall =

TP + *δ*

TP TP + *δ*

*,*

*→*

*G G*

TP

*G ≥ G* , as *δ ≥* 0*,*

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

Precision =

TP TP + *δ*

*.*

*→*

*P P* + *s*

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

Rearranging, we get,

TP + *δ* TP

*<*

*P* + *s P*

*P* (TP + *δ*) *<* TP(*P* + *s*)*,*

(*P* )*δ <* TP(*s*)*,*

*δ* TP

*<*

*s P*

= Precision*.*

Thus the Recall will increase and the Precision will decrease, if the ratio between the increase in the number of predictions, *s* 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 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.

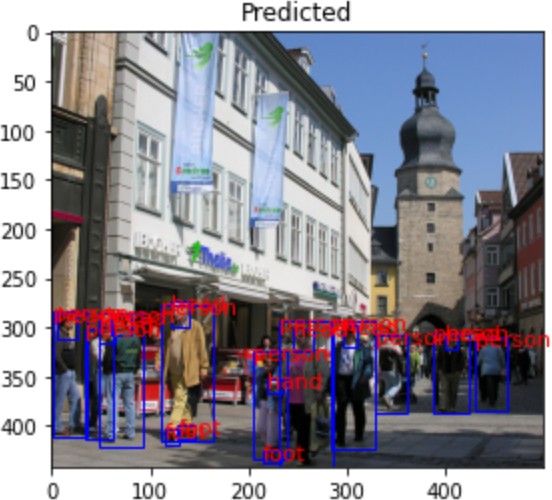


Figure 16:

# 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 is 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 introduce how to get started and a step by step procedure for object detection using Matterport’s Mask R-CNN implementation (2017). And Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun(2015) tells 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). At the same time, Google ’s Inception series models also perform well on this task. For exampleChollet Fran¸cois(2016) gives the method to develop the Inception model, which can be called as Xception model. All of these articles are talking about different viewpoints around convolutional neural network and how can neural network work for our VOC2012, which is worthy of referencing in our group project.

## Deep Neural Networks for Object Detection

To do 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). 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.

## Advantages of Mask R-CNN

It has been determined that deep neural networks can be used well for ob- ject 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. More- over, the pose can be uncertain and can appear anywhere in the picture along with objects possibly being from multiple diffrent categories. There are too many methods based on DNN for solving these problems, what kind of object detection algorithm is perfect for our VOC2012 dataset? Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross B. Girshick(2017) think 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 perform- ing 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.

## Implementation of Mask R-CNN

Their article does not explain in detail how to implement mask r-cnn. Mat- terport(2017) implements a Mask R-CNN framework for Object Detection and Segmentation, which provide us a lot of visualizations and allow running the model step by step to inspect the output at each point. 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).

# 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...

# References

1. 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.
2. Kaiming He, Georgia Gkioxari, Piotr Doll´ar, and Ross B. Girshick. Mask r-cnn. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.
3. 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].