Cardiff University

Group G9 Project

Object Localisation

Supervisor:

Dr. Yukan Lai

April 2020

Contents

0 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.

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. For this project we use a subset of the ImageNet Object Localization dataset from Kaggle's ImageNet Object Localization 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 and the labels and co-ordinates of each object in the image. From this we could then find the different object labels and calculate the percentage representation of each label out of the total amount of objects.

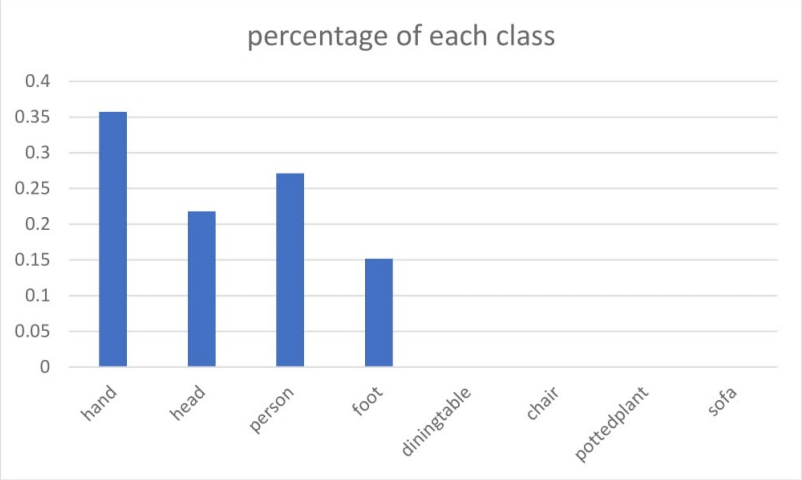

Figure 1: percentage representation of each class out of the total amount of objects

From Figure 1 we can see that the labels 'chair', 'diningtable', 'pottedplant' and 'sofa' make up a very small percentage of all objects across these images, less than 0.1% each. Because of this, an 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 $4 - 1.5 * (8 - 4) = -2 < 3$, the smallest number of objects. However, all values above $8 + 1.5 * (8 - 4) = 14$ are outliers, giving 27 images to be 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.

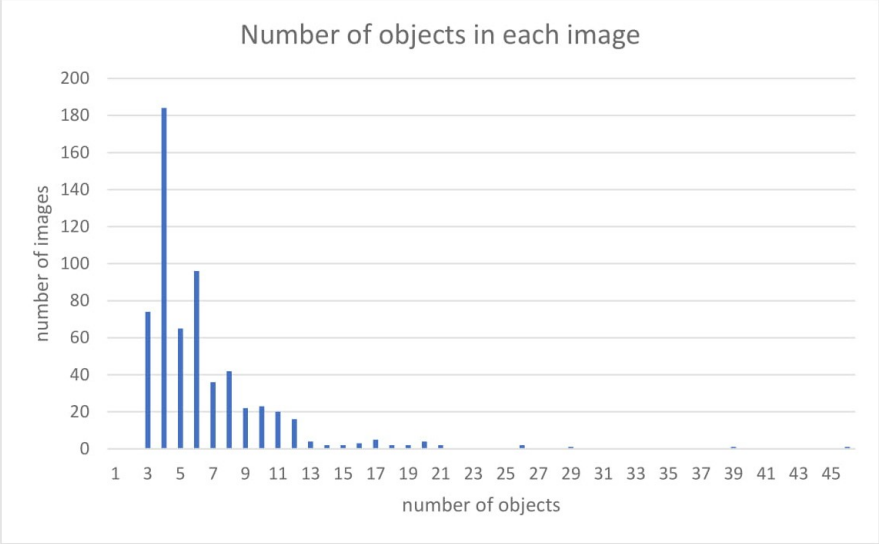

Figure 2:

2 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 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 techniques behind the more complex models 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 creating a data-set using the images specified in the text file supplied with the original data. For these images 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 annotation files corresponding to each image in the data-set and using "ElementTree" module from "xml.etree" in Python.

From this set of images, a subset was extracting by taking images that only contained one person. This subset would become the set for training and test images for the 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 corresponding 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 of size 4.

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.

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

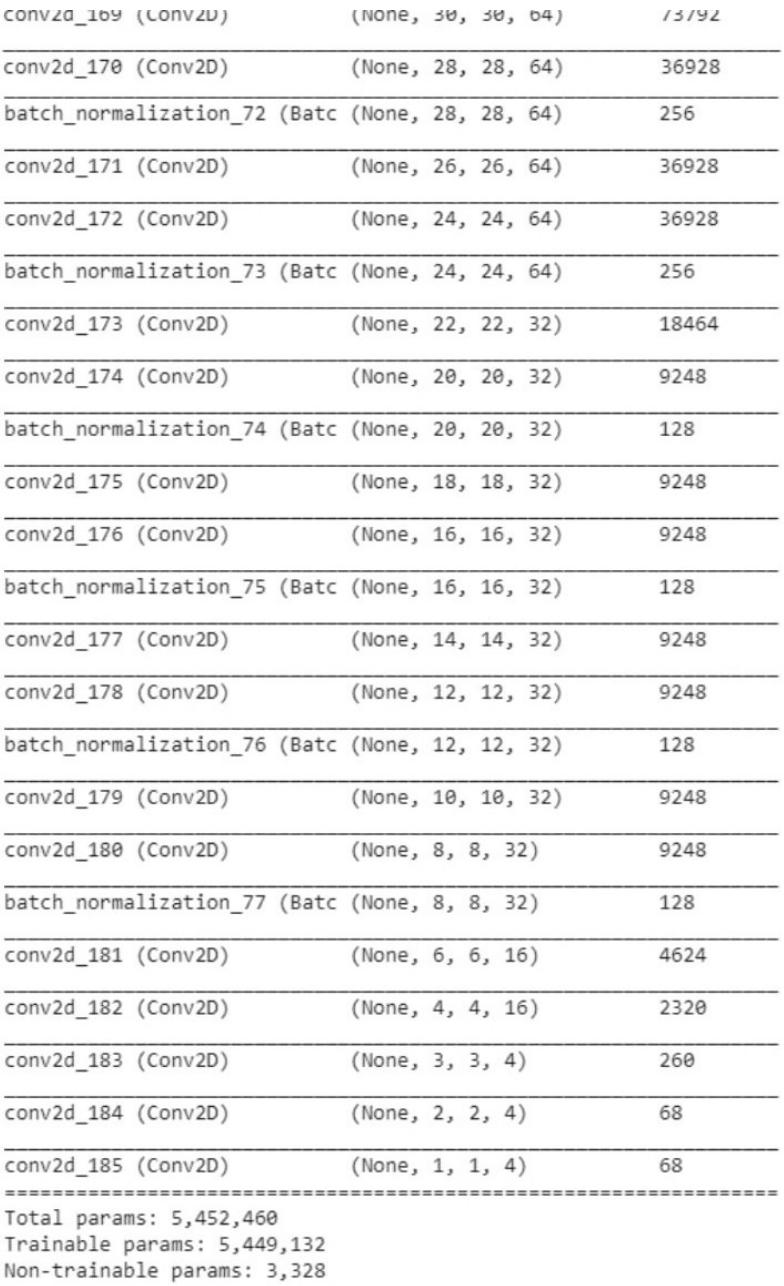

Figure 3: Convolutional Neural Network

2.2 Mask R-CNN

Mask R-CNN, (Region Convolutional neural network), is a method for object 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.

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.

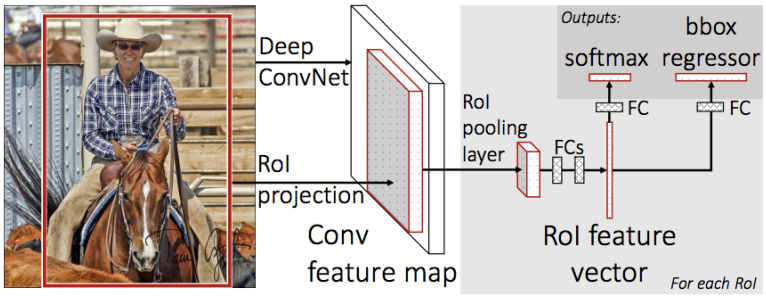

Figure 4: Fast R-CNN architecture

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

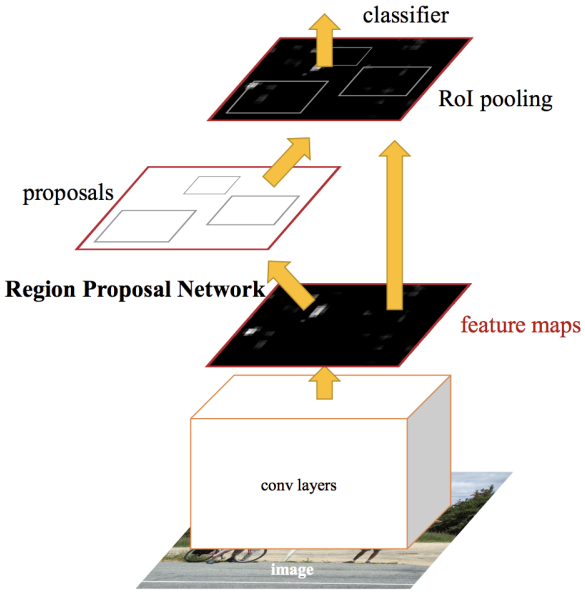

Figure 5: Faster R-CNN architecture

2.2.2 Mask R-CNN, Theory

Mask R-CNN works in exactly the same way as Faster R-CNN except it outputs 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.

2.2.3 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.

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 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. ....

3 Experimental setting

To attempt to maximise the performance of the model we trained the model on the training set using a variation of settings and evaluated its performance using the validation set. The following items are some of the settings the model was trained under.

- Optimiser
  - SGD
  - Adam
- Learning rate
- Momentum
- Regularisation
- Model Layers
  - Heads
  - All

4 Results

5 Analysis

6 Literature review / Related work

7 Conclusion and future work

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 Dollár, and Ross B. Girshick. Mask r-cnn. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.