UNIVERSITY OF SOUTHAMPTON

Applications of AI and Computer Vision in Agriculture-Fruit recognition, localisation and segmentation

by

Jing Meng(29299675)

Supervisor: Jize Yan

Second Examiner: Nicholas Gibbins

A dissertation submitted in partial fulfilment of the degree of MSc Data Science

in the

Faculty of Engineering and Physical Sciences

Department of Electronics and Computer Science

September 2019

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
DEPARTMENT OF ELECTRONICS AND COMPUTER SCIENCE

by Jing Meng(29299675)
Supervisor: Jize Yan
Second Examiner: Nicholas Gibbins

With the proposal of ReLU, dropout, and the development of GPU, AlexNet was proposed in 2012. From then on, CNNs were boomed, Top-5 error of image classification has reduced quickly in each year ILSVRC context. Till 2015, the Top-5 error of image classification reduced to less than 5%, which is less than human eye recognition error rate. The image classification technology has been well-developed. However, the object detection technology did not perform well, until the advent of R-CNN in 2014. The report studies and explores the state-of-the-art instance segmentation technology: Mask R-CNN start from R-CNN. Then apply the model in practical use: fruit recognition, localisation and segmentation. In addition, evaluate the impact for the result due to the change of parameters and some structures of the algorithm in terms of precision and execution time. For example, heads training with all layers fine-tuning, the backbone ResNet-50 with ResNet-101, from different pre-trained models to train, use the data augmentation technology to train. Based on the results, achieve the conclusion.

Acknowledgements

This project is supervised by Dr Jize Yan. I would like to thank him for his patient guidance, advice and asssistance during the entire work. You provided me with great insights, perspective. I also would like to thank my second examiner Dr Nicholas Gibbins for providing me with some greatful advice, and answering my questions during and after demonstration.

Statement of Originality

- *I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.*
- *I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.*
- *I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.*
- *I have acknowledged all sources, and identified any content taken from elsewhere.*
- ***The project references the open source mature code framework from the GitHub and references some examples associated with the code, how to use the API** [1].*
- *I did all the work myself, and have not helped anyone else.*
- *The material in the report is genuine, and I have included all my data/code/designs.*
- ***I appropriate re-use literature review, project plan and project brief submitted to ELEC6211 Project Preparation.***

- *My work did not involve human participants, their cells or data, or animals.*

Contents

List of Figures

List of Tables

Nomenclature

Average Precision	AP
Area under Curve	AUC
Bounding Box	bbox
Convolutional Neural Networks	CNNs
Fully Connected Layer	FC layer
Fully Convolutional Networks	FCN
Mean Average Precision	mAP
Multiple-Layer Perception	MLP
Non-Maximum Suppression	NMS
Region of Interests	RoIs
Regional with CNN	R-CNN
Support Vector Machine	SVM
Spatial Pyramid Pooling	SPP

Chapter 1

Introduction

1.1 Background and Motivation

With the high development of machine learning and deep learning technologies, computer vision technologies have been greatly developed and applied in practical application, which improves work efficiency. There are three types branches in computer vision technologies: image classification, object detection, and semantic segmentation including instance segmentation. There are progressive connections from image classification to instance segmentation. Convolutional Neural Networks(CNNs) is the architecture applied in these scenes. The project is about the application of computer vision in an agriculture-fruit orchard.

1.2 Aims and Objectives

There is a practical question in fruit orchard: when fruits in the orchard are ripe, they need to be picked. It is quite efficient to use robots to pick fruits based on artificial intelligence technologies. The key step of the process is to identify and locate the fruit using computer vision technologies. Hence, the main aim of the project is to implement the fruit recognition, localization and segmentation captured from the orchard. The secondary aim is the assurance of the accuracy of the algorithm. If the accuracy of the algorithm cannot achieve the accepted confidence interval, it does not make sense for this application. The project is the algorithm study, does not extend to the application software. To achieve these aims, the corresponding objectives can be summarised as follows:

- Investigate and analyse typical models from image classification, object detection to semantic and instance segmentation.

- Study the transfer learning technology.
- Build a deep learning environment.

1.3 Report Structure

The main body of the report is divided into below parts:

- Project Plan: Make a complete and detailed plan for the project, reflect excellent time management considering contingency as well.
- Research Review: Analyse, discuss and evaluate typical models from image classification, object detection to semantic and instance segmentation
- Methodology: Introduce the tools and techniques utilised in the project.
- Code Explanation: Explore and analyse the source code architecture and how to call functions to implement on your own data set.
- Results and Discussion: Based on the results, evaluate the impact of results due to the change of parameters and some structures of the algorithm in terms of accuracy and execution time.

Chapter 2

Project Plan

2.1 Requirements Analysis

As the aims and objectives stated in the introduction, because the project is the algorithm study, does not include the application software part. the requirements of the project are as follows:

2.1.1 Functional Characteristics

- Chose one picture randomly from the validation and test data set respectively, show the classification label, the bbox and the mask on the original image.
- Precision recall curve for the chosen picture.
- Grid of ground truth objects and their predictions for the chosen picture.

2.1.2 Non-functional Characteristics

- Compute mAP of each image on the validation data set then makes the mean of the sum of mAP under the IoU=50 condition, as well as on the test data set.
- Compute the inference time for a per image.

2.2 Processes and Time Management

The project processes can be divided into following parts (Figure 2.1):

Based on the above processes, the project Gantt graph is as follows(Figure 2.2): The

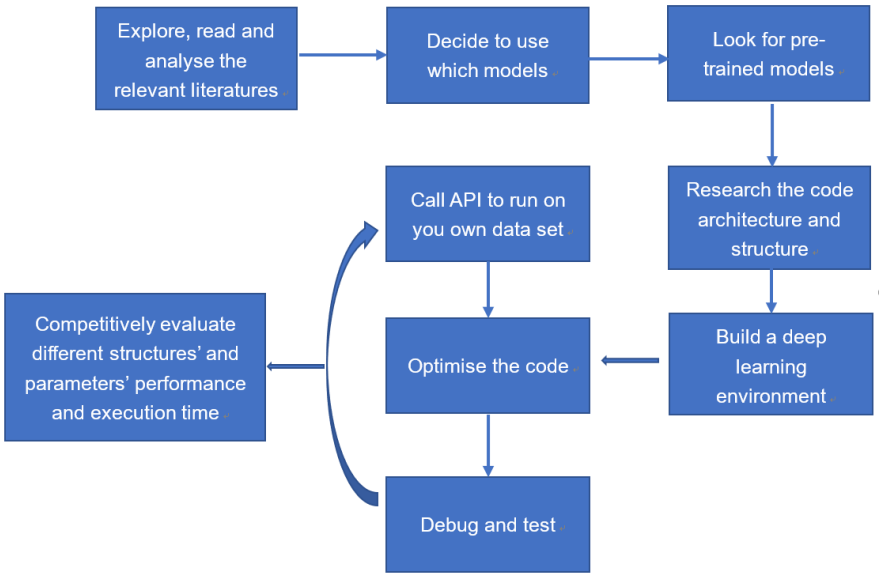


FIGURE 2.1: The Project Processes.

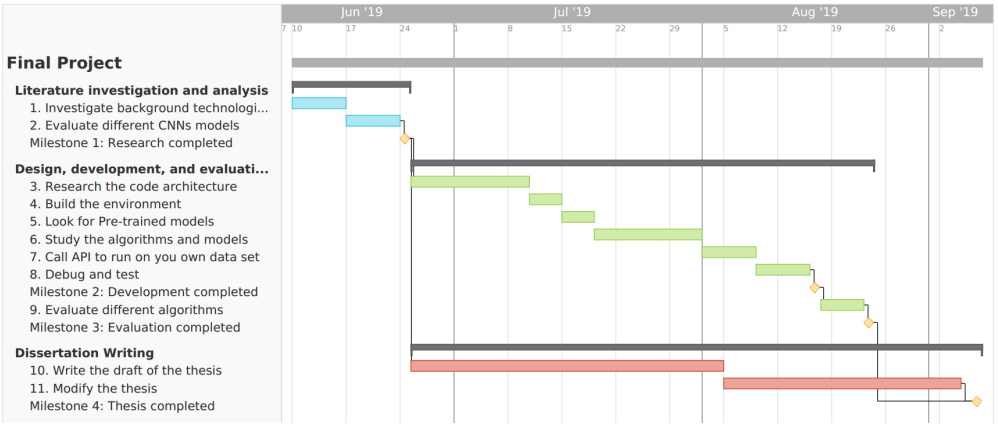


FIGURE 2.2: The Project Gantt Graph.

process of the project can be divided into 3 main groups of activities: Literature investigation, review and analysis; Research, development, debug, test and evaluation; Dissertation writing.

The first group consists of two tasks "Investigate the background technologies in computer vision", "Evaluate the applicability of different CNNs models in instance segmentation". The purpose and the outcome are to understand the computer vision technologies and provide the overview, insight for the whole project.

The second stage is an important part of the project, which is the implementation of the algorithm. Because the algorithm is complicated, there are existing mature source codes, the key step is task 6. Task 7 and Task 8 will follow the software developing principle and abide by the agile methodology to improve efficiency and quality and provide the fundament for the next step. After the three tasks, the next step is to evaluate different structures' and parameters' performance and execution time.

The final stage is thesis writing, the writing will start after literature review finishing to ensure there is enough time for writing and modifying.

In addition, there will be a discussion meeting with the supervisor every week.

2.3 Contingency Planning

Because the project is regarding deep learning technologies involving high volume computation. The training process is time-consuming, which needs strong hardware resource support. Hence, it is most likely to accidentally affect the entire project. As a result, the plan prepares for three backup schemes:

- The google cloud platform Collaboratory which provides Nvidia Tesla K80 GPU free resources.
- High-Performance Cluster Iridis 5 provided by University of Southampton.
- Cloud Servers provided by cloud platform service provider e.g. Google Cloud, Amazon AWS, Vast.ai.

Chapter 3

Research Review

This part will analyse, discuss and evaluate the instance segmentation algorithm: Mask R-CNN which includes image classification, object detection and segmentation. Hence, the following part will introduce the inner connection of three sections step by step.

3.1 Some Related Concepts

Before analysing and discussing the relevant models, the metrics in image classification, object detection and semantic and instance segmentation will be introduced.

3.1.1 Confidence Score

Confidence Score refers to how much probability that an anchor includes an object [2].

3.1.2 Bounding Box

Bounding Box indicates the anchor box than contains an object with highest Foreground class score, i.e. the minimum box containing an object [3].

3.1.3 Intersection over Union

Intersection over Union(IoU) measures "the area of the intersection divided by the area of the union of a predicted bounding box(B_p) and a ground truth box(B_{gt})"(Figure 3.1) [2].

$$IoU = \frac{area\,(B_p \bigcap B_{gt})}{area\,(B_p \cup B_{gt})} \tag{3.1}$$

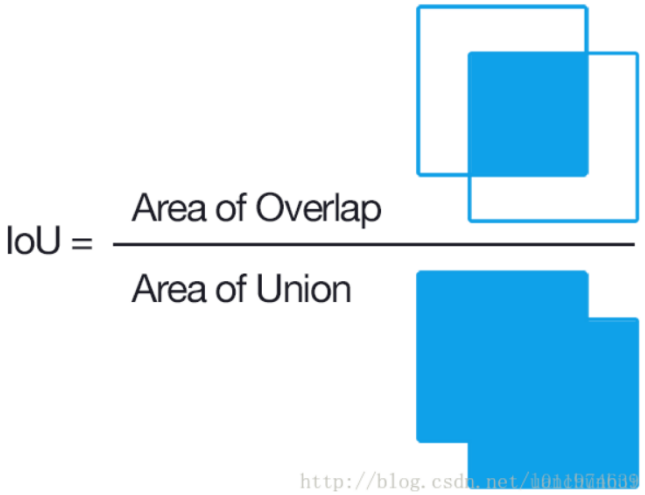


FIGURE 3.1: Intersection over Union.
[4]

3.1.4 Prediction Outcomes

There are four prediction outcomes in instance segmentation:

- True Positive(TP): The predicted object mask has an IoU score which is more than a predefined threshold.
- False Positive(FP): The predicted object mask has an IoU score which is less than a predefined threshold.
- False Negative(FN): There is no associated object mask for a ground-truth object mask.
- True Negative(TN): The predicted object does not belong to the given class. In object detection, this condition is usually not cared. [2] [5]

3.1.5 Precision

Precision is defined that the percentage of ture positive conditions in the positive predictions [5].

$$\frac{TP}{TP + FP} \tag{3.2}$$

3.1.6 Recall

Recall refers to the proportion of positive predictions in the ground truth [5].

$$\frac{TP}{TP + FN} \tag{3.3}$$

3.1.7 Precision Recall Curve

We can get different precision and recall by setting different thresholds. Generally speaking, in practical condition, if the accuracy is high, the recall will be relatively low, and vice versa. Thus, we intend to find the balance between precision and recall. We can take precision as the vertical axis, recall as the horizontal axis, draw the precision recall curve in different thresholds, through computing AUC(area under Curve) to evaluate the performance of the model. The bigger of AUC, the higher performance of a model [2].

3.1.8 Average Precision

In order to decrease the influence of steep of the precision recall curve, the average precision concept is introduced, it interpolates precision at different recall levels. For level r, to find the highest precision for any recall level $r' \geq r$ [2]:

$$p_{interp}(r) = \max_{r' \geq r} p(r') \tag{3.4}$$

There are two methods for interpolation:

1. 11-points-interpolation: equally divide recall values from 0 to 1 into 11 levels (i.e. 0-0.1, 0.1-0.2, ..., 0.9-1.0), calculate the maximum of precision at different levels, then take the mean of the sum of these precision to obtain average precision.

2. all-points-interpolation: choose all recall points instead of above 11 points.

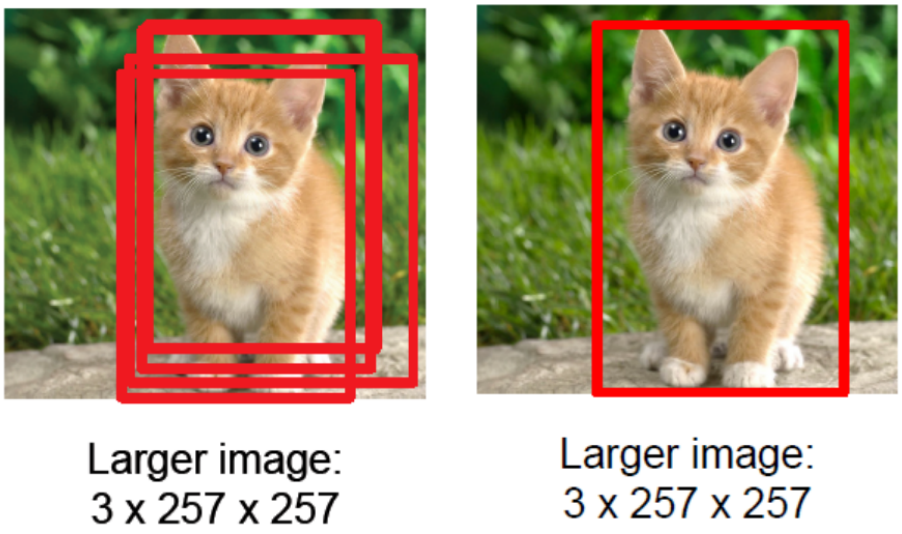
3.1.9 Mean Average Precision

Mean Average Precision refers to the mean of Average Precision for all classes(defined as N) [2].

$$mAP = \frac{\sum_{i=1}^{N} A_i}{N} \tag{3.5}$$

3.1.10 Non-Maximum Suppression

Non-Maximum Suppression indicates to filter out the values which are not maximum by searching for local maxima. In object detection, it means to filter out redundant bounding boxes of an object which has multiple bounding boxes (Figure 3.2) [4].



(a) Before Non-Maximum Suppression (b) After Non-Maximum Suppression

FIGURE 3.2: Non-Maximum Suppression.
[4]

3.2 Image Classification

Image Classification is a study for recognising objects in an image. With the high development of CNNs, there has been developed several typical CNNs architectures for image classification with better performance than traditional technologies. The below will introduce and discuss the corresponding CNNs architectures. Especially, because ResNet is the backbone of the algorithm used in this project, it will be analysed and discussed more detailed than others.

3.2.1 GoogLeNet

The emergence of GoogLeNet reduces model parameters and decreases the overfitting problem in an enlarged network. It applies the FC layer instead of max pooling layer in the final layer to achieve this purpose [6]. The important module introduced in GoogLeNet is inception-v4 [6].

3.2.2 VGGNet

VGGNet stacks 3×3 convolutional layers and 2×2 pooling layers from start to end, which can improve the transfer performance of model [7]. In addition, VGGNet also applies a 1×1 convolutional kernel to enhance the recognition ability of model [7].

3.2.3 ResNet

The purpose of the design of ResNet is to figure out the problem of deeper neural networks are more difficult to train especially gradients vanishing or exploding during the back-propagation process. Under the plan baselines inspired by the philosophy of VGG nets, ResNet creatively introduces the identity shortcut connections as learning residual functions referencing to the previous layers (shown in Figure 3.3) [8]. Furthermore, ResNet baseline model has fewer filters and lower complexity than VGGNet [8].

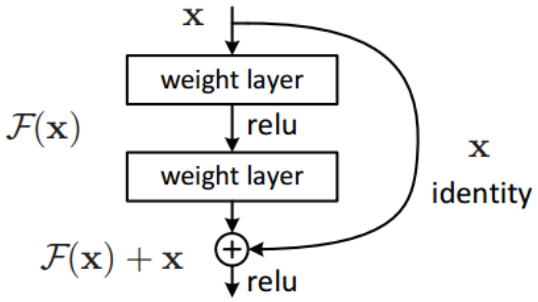
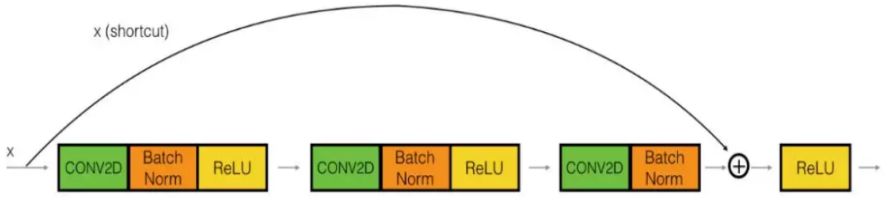


FIGURE 3.3: Residual learning: a building block.
[8]

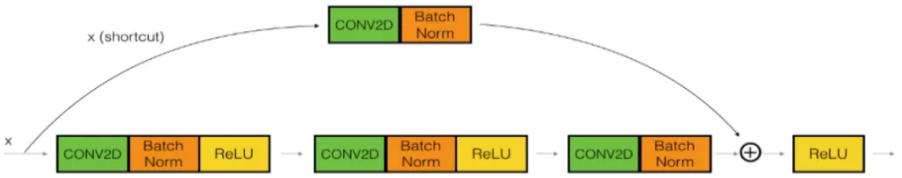
There are two types of shortcut connections, one is identity block used in the condition that channel size of input and output is consistent, the other one is convolutional block for the condition that the channel size of input and output is inconsistent (shown in Figure 3.4). In other words, the effect of a convolutional block is to adjust the channel size of input and output to be the same. In addition, ResNet applied batch normalisation after each convolution and before activation [8].

In addition, there are several typical layers of architecture in practical use, which are 18-layer, 34-layer, 50-layer, 101-layer, and 152-layer (Figure 3.5). The ResNet network can be divided into 5 stages, the first item in square brackets is the kernel size, the second item in square brackets is the channel size, the multiple sign next to the square bracket presents stacking several identity blocks together. The more clear architecture for ResNet-50 is as follows (Figure 3.6). The typical convolutional block contains convolutional layer, batch normalisation, ReLU activation layer and final max pooling layer. There is a point to note that the number of layers such as 50,101 just contains the convolutional layer or FC layer, does not include the activation layer and pooling layer.

For different layers, there are two types of shortcut connection blocks, one is the basic building block (Figure 3.7(a)) applied for low-layer networks , the other one is the bottleneck building block (Figure 3.7(b)). The bottleneck design is designed for parameters reduction in high-layer networks, such as ResNet-50/101/152. The first 1×1 convolutional kernel can decrease channel size from 256 to 64, after 3×3 convolutional kernel, use 1×1 convolutional kernel make channel size back to 256. The overall parameter for



(a) ResNet Identity block



(b) ResNet Convolutional Block

FIGURE 3.4: ResNet Shortcut Connection Types for different dimensions.
[9]

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

FIGURE 3.5: ResNet Architecture.
[8]

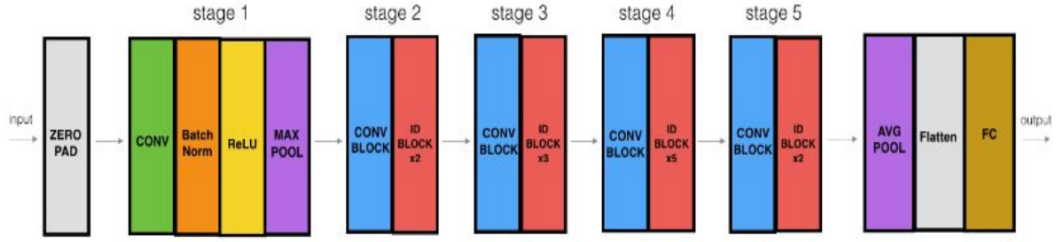
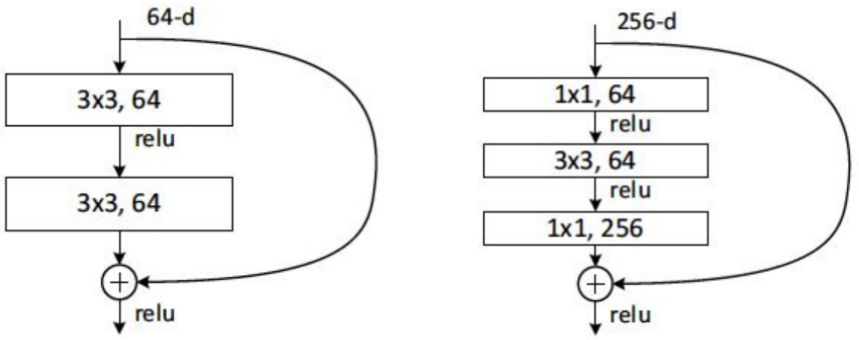


FIGURE 3.6: ResNet-50 Architecture.
[9]

bottleneck block is $1 \times 1 \times 256 \times 64 + 3 \times 3 \times 64 \times 64 + 1 \times 1 \times 64 \times 256 = 69632$, which is around $\frac{1}{17}$ as basic block's parameter ($3 \times 3 \times 256 \times 256 \times 2 = 1179648$).



(a) ResNet Basic Connection (b) ResNet Convolutional Block

FIGURE 3.7: ResNet Shortcut Connection Types for Different Layers. [8]

3.3 Object Detection

Based on image classification, object detection is to classify and localize different objects in an image (Figure 3.8).

The algorithms for object detection can be divided into two types: one-stage and two-stage. The two-stage algorithm as the name implies has two steps, first generate proposals, then classify and localize objects by CNNs. Different from two stages algorithm, one stage algorithm does not use proposals first, classification and localization are applied in one stage. Therefore, obviously, the speed of the one-stage algorithm is quicker than the two-stage algorithm. The two-stage algorithm can be used in real-time application such as video processing. The project uses the Mask R-CNN algorithm which belongs to the two-stage algorithm. Hence, we will mainly discuss the two-stage algorithm in this report.

3.3.1 Two-stage Algorithm

In this section, we will mainly analyse and discuss four typical algorithms step by step: R-CNN, SPP-net, Fast R-CNN, Faster R-CNN.

3.3.1.1 R-CNN

R-CNN is short for Regional with CNN. The shinning point of R-CNN is to adopt selective search methods for generating proposals probably covering target objects. The

steps of the algorithms can be summarised to below four steps:

(1) Extract Region Proposals,

(2) Extract features for generated proposals by CNN,

(3) Classify objects by SVM based on the extracted features,

(4) Refine bounding box via regression. (Figure 3.9) [11]

Selective search method generates around 2000 proposals with different size for next step features extraction [12]. The limitation for features extraction is the input image size must be fixed. Hence, before features extraction, the input image size must be adjusted to the same size .

Based on extracted region proposals, crop and warp each region to the same size for CNN input, then run forward process through CNN, save pool5 features to disk [11]. As a result, it needs a big hard disk to save these features.

R-CNN uses AlexNet as the backbone, the output size of pool5 is 4096, therefore, for 2000 proposals, the dimension of extracted feature vectors is 2000×4096, then train

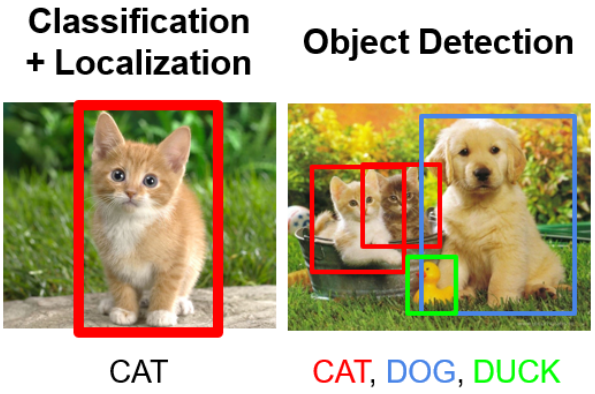


FIGURE 3.8: Object Detection.
[10]

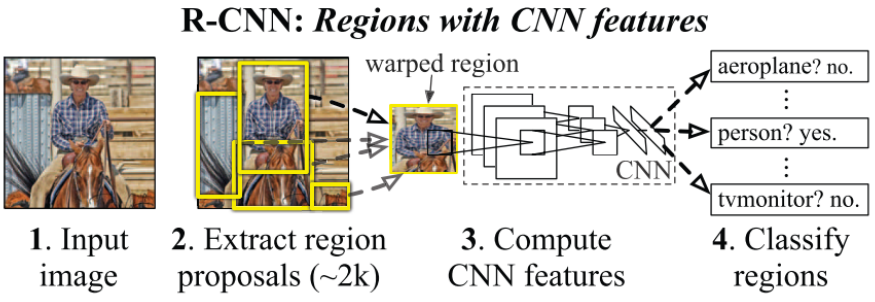


FIGURE 3.9: R-CNN Process.
[11]

one binary SVM per class to classify regions to get weight matrix with $4096 \times N$ (N is types of classification). For training samples, the regions with $IoU > 0.7$ are regarded as positive samples, the regions with $IoU < 0.35$ are regarded as negative samples. After SVM classification, it outputs a $2000 \times N$ score matrix, then applies NMS to obtain the final proposals with classification results. The process can be approximately summarised as follows: (1). Descending sort matrix along a column, (2). Apply the NMS from the maximum down of each column, column by column, (3). Obtain the final proposals based on the threshold [11].

The final step is the bounding box refinement. Since the region proposals after SVM is approximate to the ground-truth, use linear regression between region proposals and ground-truth for final bounding box refinement (shown in Figure 3.10).

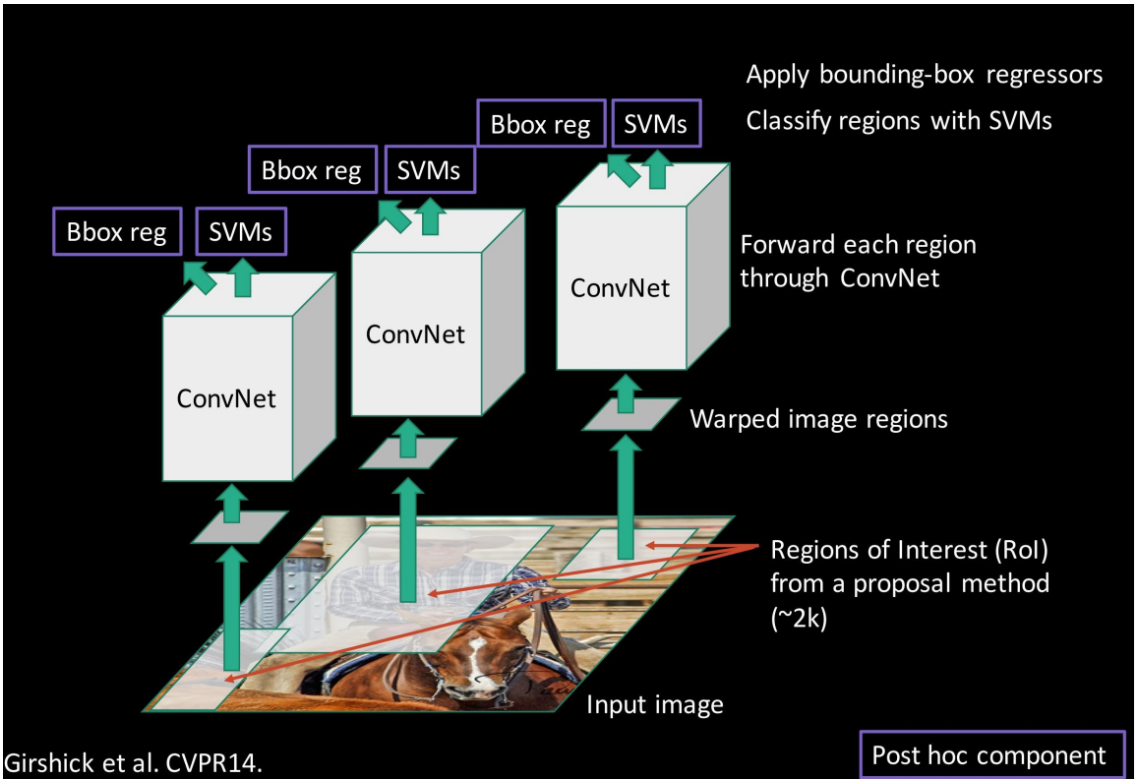


FIGURE 3.10: R-CNN Specific Process.
[13]

3.3.1.2 SPP-net

Although R-CNN has obviously improved the accuracy of object detection with an mAP of 53.5% which is over 30% than the previous best result [11]. There are two main disadvantages of R-CNN. (1). Because FC layer has fixed neuro size, the requirement for each existing CNN is fixed input image size, which needs cropping or warping input images leading to incomplete part from the original image. The consequence is the

geometric distortion of the original input image result in accuracy decrease. (2). Each region proposal needs to extract feature by CNN result in heavy computation. It means 2000 region proposals need 2000 CNNs.

SPP-net came up with a new network structure: spatial pyramid pooling layer lays after feature extraction convolutional layers, which projects, transfers and generates a fixed-length feature vector based on the feature map directly extracted from CNN no matter how image's size or scale is [14] (Figure 3.11). It just uses 1 CNN to extract features on the entire image getting rid of heavy computation. Hence, SPP-net can solve the above two problems.

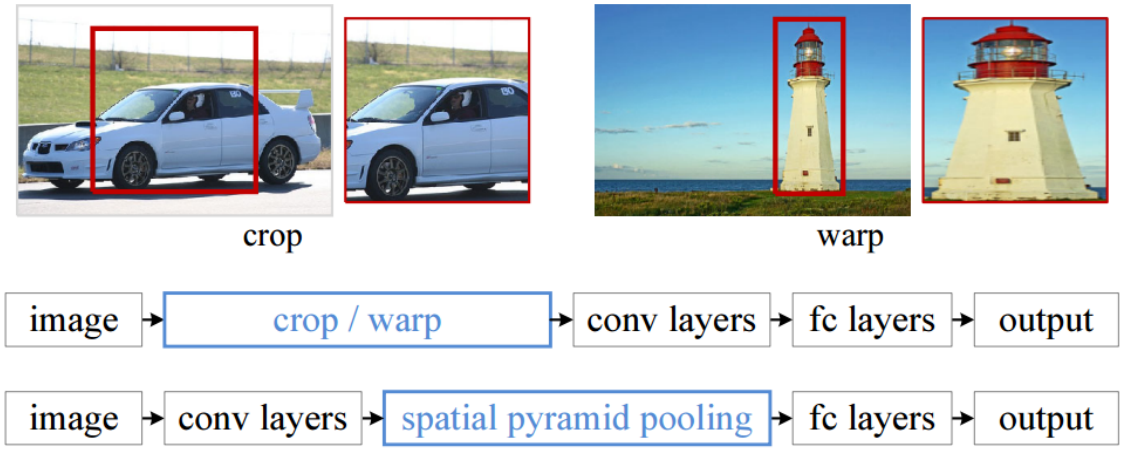


FIGURE 3.11: SPP-net Architecture.
[14]

The specific processes of SPP are shown as follows: It splits a feature map into different scale bins: 1×1, 2×2, 4×4 then makes the max-pooling to each output, afterwards combines them together, and finally, the output is $(16 + 4 + 1) \times 256$ feature vector [14] (Figure 3.12).

3.3.1.3 Fast R-CNN

Although SPP-net figures out the problems of R-CNN, it has defects as well:

(1) The training pipeline is multi-stage training pipeline which includes generating regional proposals, training CNN for feature extraction, training SVM classifier and bounding box regressor. This means each stage is independent which cause the whole process is time-consuming.

(2) Extracted Features need to be saved in the disk, then fetched out again for the SVM classification and Bounding box regression training process, which is space-consuming and time-consuming.

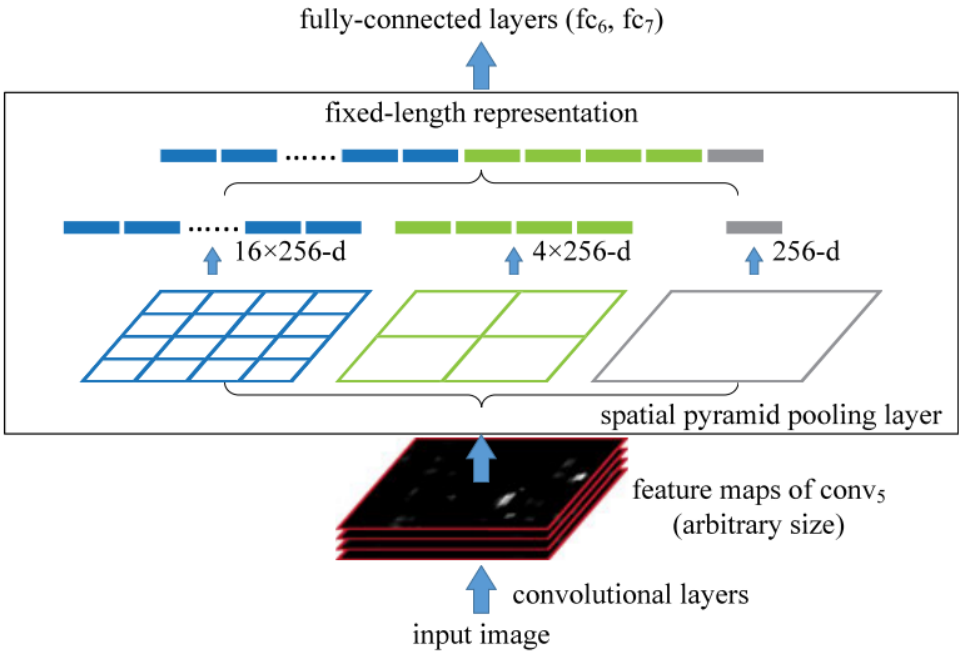


FIGURE 3.12: A network structure with a spatial pyramid pooling layer. [14]

(3) Due to the split multi-stage training pipeline, in the SVM classification and bbox regression training process, CNN parameters cannot be updated, which reduces the final accuracy (Figure 3.13).

These problems also exist in R-CNN.

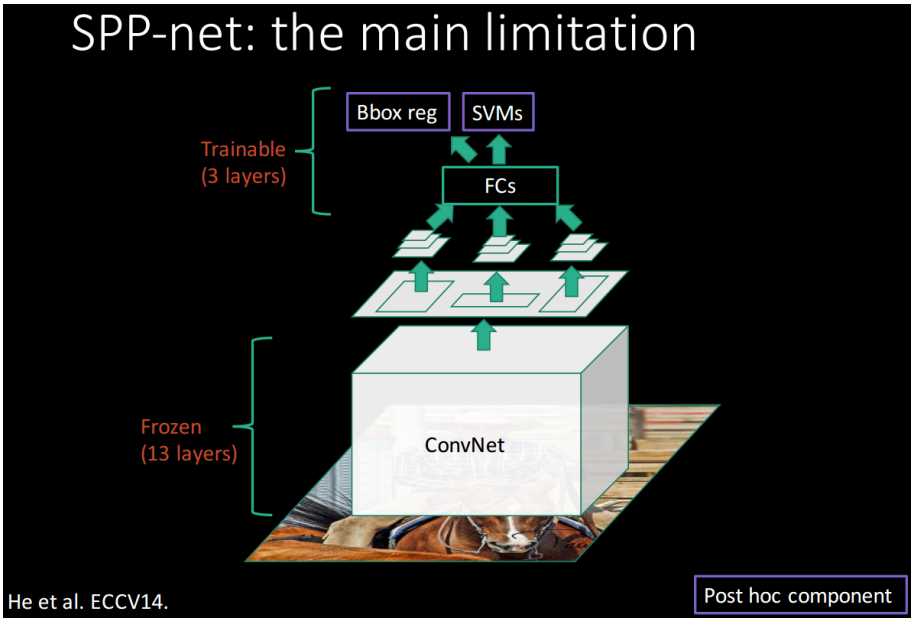


FIGURE 3.13: The limitation for SPP-net. [15]

For these problems, the philosophy of Fast R-CNN is to propose an RoI pooling layer integrating CNN layer, SVM Classification and bbox regression then train them together, make multi-stage training pipeline into single-stage, extracted features do not have to be saved in a disk and therefore save space and improve the training speed (Figure 3.14, Figure 3.15) [16].

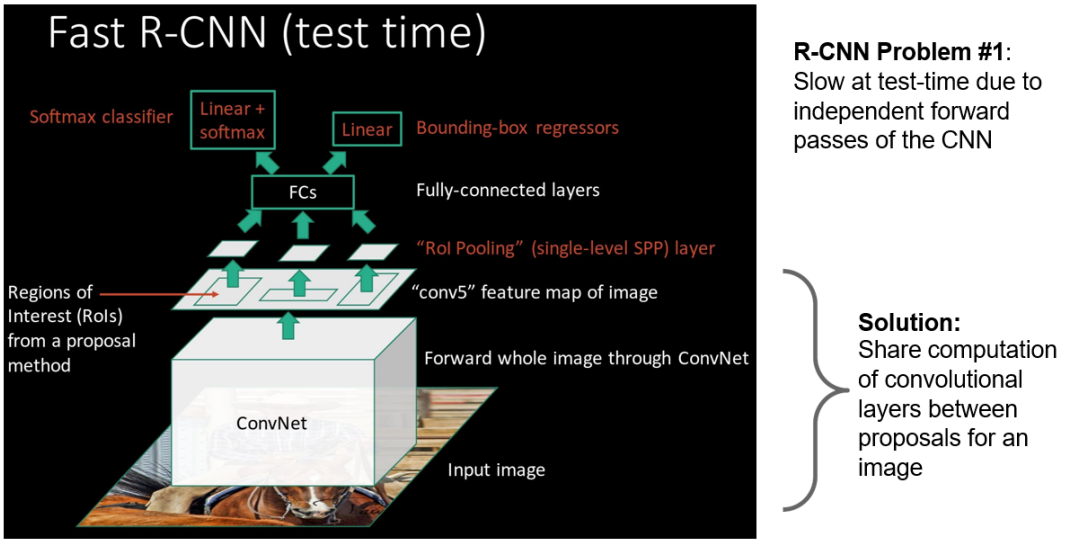


FIGURE 3.14: Fast R-CNN test process.
[13]

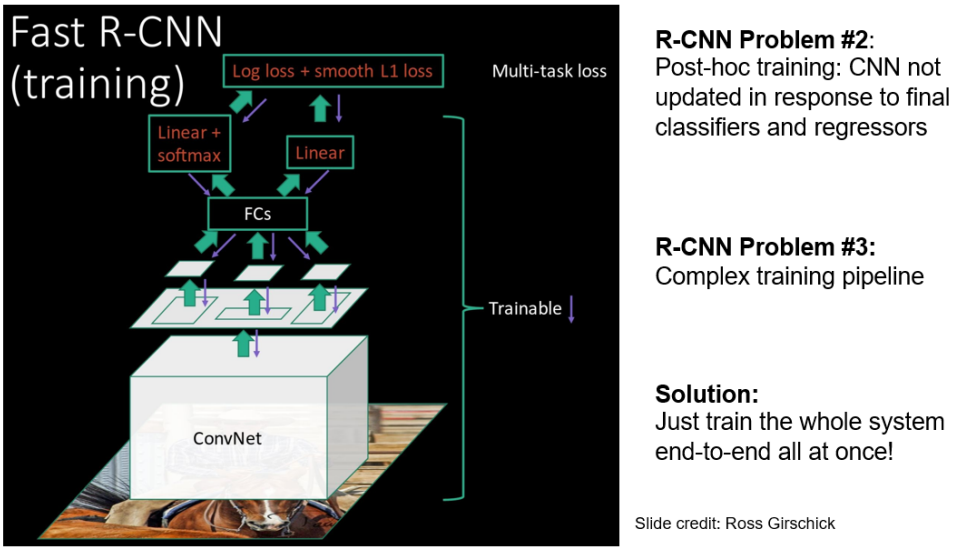


FIGURE 3.15: Fast R-CNN training process.
[13]

The specific processes are as follows:

(1) Extract whole image features through CNN,

(2) Generate RoIs from a proposal method, project each RoI to corresponding feature map patch.

(3) Utilise RoI Pooling layer to turn different scale feature maps extracted from different region proposals into a fixed-length feature vector (Figure 3.16) [16].

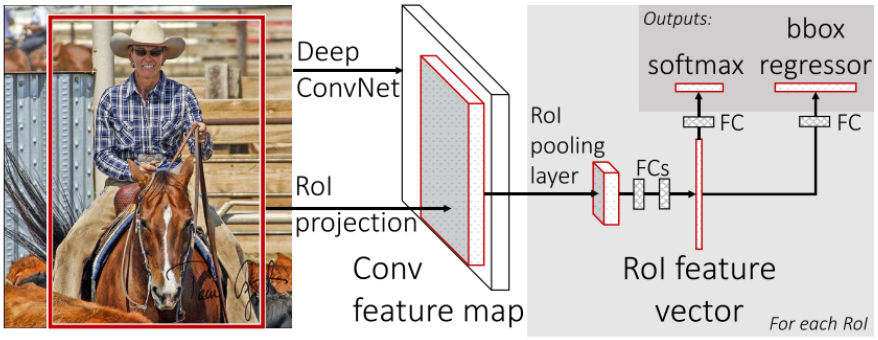


FIGURE 3.16: Fast R-CNN process.
[16]

It is worth pointing out that the RoI Pooling layer is simplified SPP layer: single-level SPP layer, because multi-scale pooling layers cannot contribute much higher accuracy, however, leads to much more computation. After FC layer, the network combines SVM classification loss and bbox regression loss together to form multi-task loss. Thus, RoI pooling layer and multi-task loss can joint CNN layer, SVM Classification and bbox regression together for training.

3.3.1.4 Faster R-CNN

From R-CNN to Fast R-CNN, the most time-consuming part is proposals extraction by selective search method. Faster R-CNN came up with RPN instead of selective search method to generate region proposals. Thus, it can be seen the four stags of object detection: region proposals generation, feature extraction, classification and location refinement are integrated into a deeper network (Figure 3.17), which eliminates most of the computational redundancy and speeds up the training process because most of the training process can be completed in GPU. As a result, Faster R-CNN can be regarded as RPN + Fast R-CNN [17].

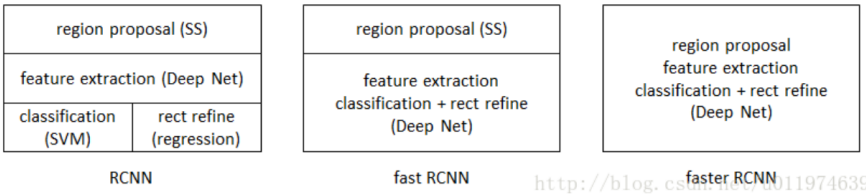


FIGURE 3.17: The Comparison of R-CNN, Fast R-CNN and Faster R-CNN process.
[4]

The specific stages are as follows: (1). Insert an RPN after the last convolutional layer of feature maps extraction. (2). Train RPN to produce region proposals. (3). Use an

RoI Pooling layer to obtain a fixed-length feature vector and then get the classification and bbox through an upstream classifier and bbox regressor (Figure 3.18) [17].

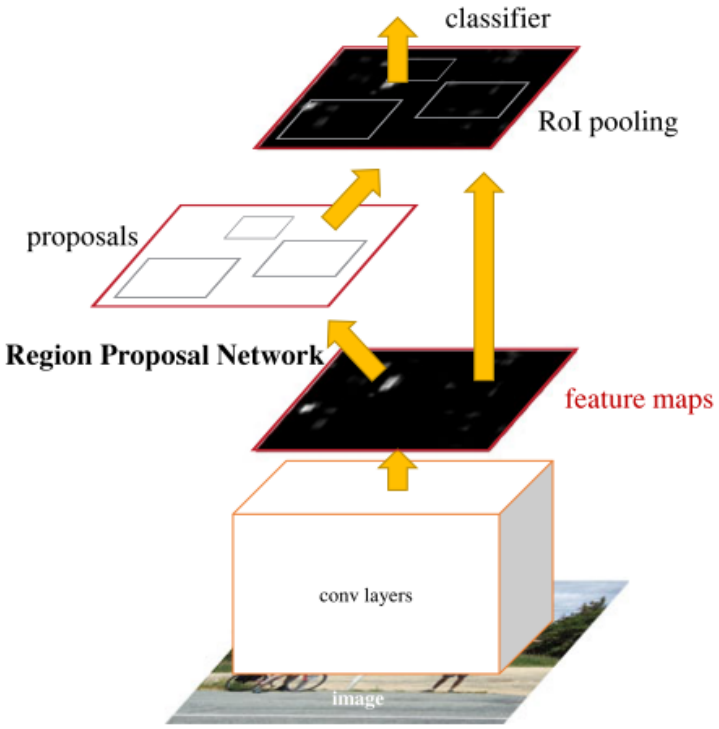


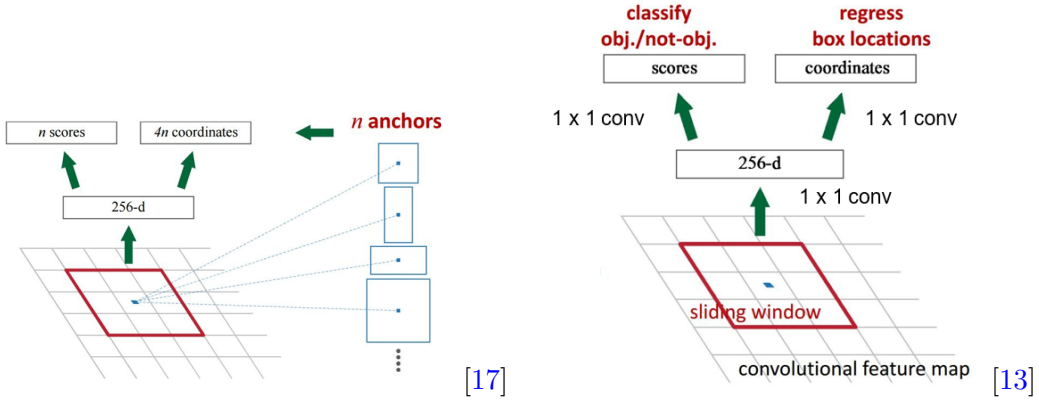
FIGURE 3.18: Faster R-CNN process.
[17]

RPN RPN is the shinning point of faster R-CNN. The basic philosophy of RPN is to use a sliding window on feature maps to extract features then output to two FC layers: box-classification layer and box-regression layer. RPN uses anchors to extract features on feature maps. There is a sliding window configuring N anchors iterating each point on feature maps. The final two outputs are n classification probability scores indicting how regressed anchor shows an object and $4 \times n$ coordinates offsetting from anchor boxes (Figure 3.19(a), Figure 3.19(b)) [17].

Thus, the whole network system has four final losses, RPN classification loss, RPN regression loss, Fast R-CNN classification loss and Fast R-CNN regression, which are shown as follows (Figure 3.20).

Each anchor centres around the centre of the current sliding window with different scales and aspect ratios. In default, there are three sizes and three aspect ratios, therefore, there are 9 anchors for each sliding window. For image $W \times H$ (e.g. 224×224), there are $W \times H \times n$ anchors. The following picture shows the feature dimension for different anchors in each process (Figure 3.21).

The feature map is $51 \times 39 \times 256$, for each location of a picture, there are 9 potential



[17] [13]

(a) RPN Anchors (b) RPN process

FIGURE 3.19: RPN.

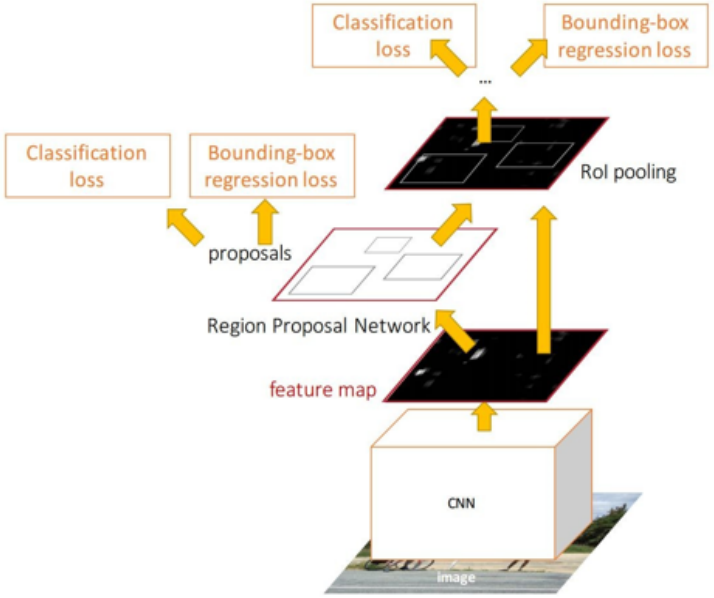


FIGURE 3.20: Faster R-CNN losses.
[13]

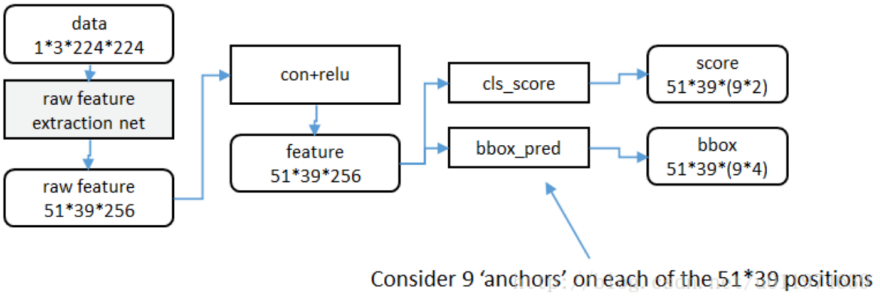


FIGURE 3.21: Feature Dimension for Different Anchors in Each Process.
[4]

anchors with three areas: 128×128, 256×256, 512×512, and three aspect ratios 1:1, 1:2, 2:1. The following image indicates 51×29 anchor centres and 9 anchors (Figure 3.22).

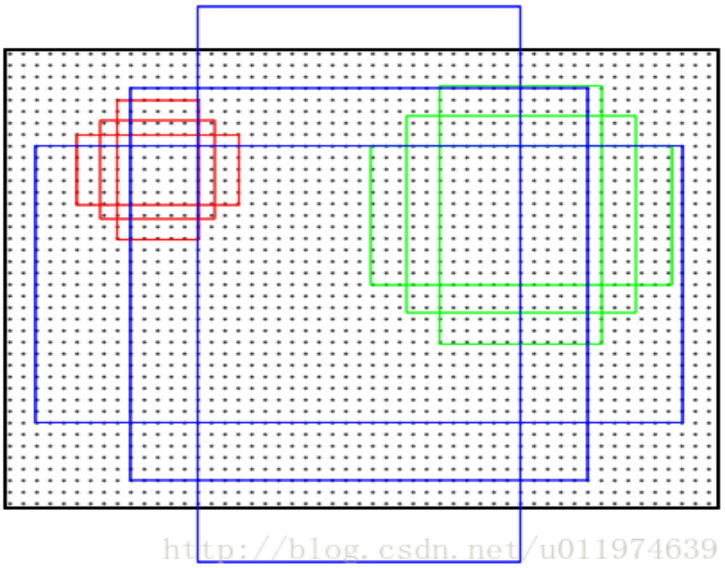


FIGURE 3.22: Anchors for different scales and ratios.
[4]

RPN has two classes (Foreground and Background), the anchors with $IoU \geq 0.7$ are regarded as positive samples, the anchors with $IoU \leq 0.3$ are considered as negative samples. The remaining anchors are neural samples, which are not used for training. The final anchor with the highest score is bbox.

Sharing Features The key character of Faster R-CNN is RPN can share features extracted from CNNs to whole network (Figure 3.23), which accelerates the training speed. There are three approaches for fine-tuning the parameters to meet the two parts' needs, alternating training, approximate joint training and non-approximate training.

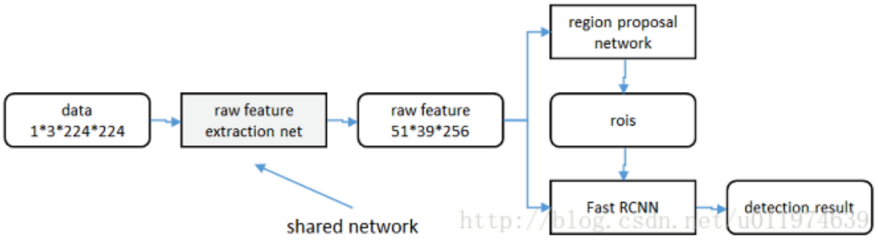


FIGURE 3.23: Share Features Network.
[4]

1. Alternating training The procedure can be summarised as follows:

   (1) Train RPN.

(2) Initialise Fast R-CNN network weights with RPN network weights, use the output of RPN as an input.

(3) Exchange previous two steps, iterate for twice (more iterations cannot contribute better performance).

2. Approximate joint training During the backpropagation process of box classification of RPN, the gradient is used for parameters update, however, for box regression process, the gradient is ignored, because this is beneficial to obtain an analytical solution, hence, this is called "approximate".

3. Non-approximate training. Non-approximate training means it does not ignore gradient during the backpropagation process of box regression.

3.4 Semantic Segmentation

Semantic segmentation refers to distinguish multiple objects belonging to different classes and draw their boundaries in pixel level in an image. The typical network architecture for semantic segmentation is fully convolutional networks, which is end-to-end, pixels-to-pixels prediction (Figure 3.24).

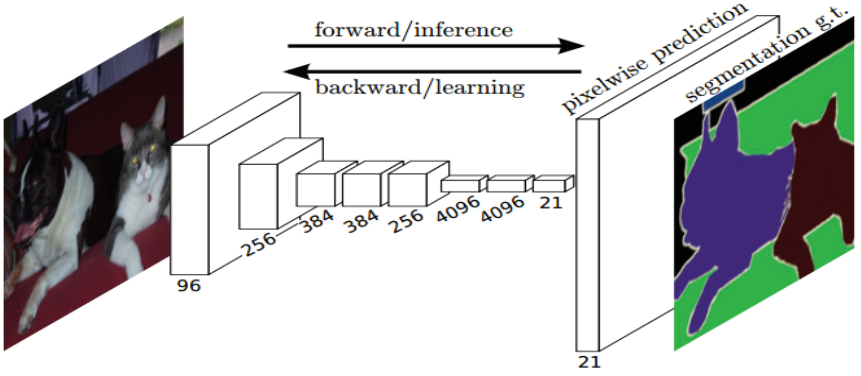


FIGURE 3.24: Fully convolutional networks.
[18]

3.4.1 Fully Convolutional Networks

FCN adopts encoder and decoder architecture and uses AlexNet as the network's backbone. The encoder acts as downsampling which is comprised of convolutional layers and pooling layers. The decoder plays the role of upper sampling which consists of transposed convolutional layers and de-pooling layers so that realise end-to-end, the same resolution output as input. FCN net transforms FC layers to fully convolutional layers which can accept arbitrary size input and output corresponding classification maps. The

feature map of the last convolutional layer output of the encoder is upsampled by transposed convolution until the feature map is restored to the resolution of the input image so that end-to-end pixel-level image segmentation can be achieved (Figure 3.25)[18].

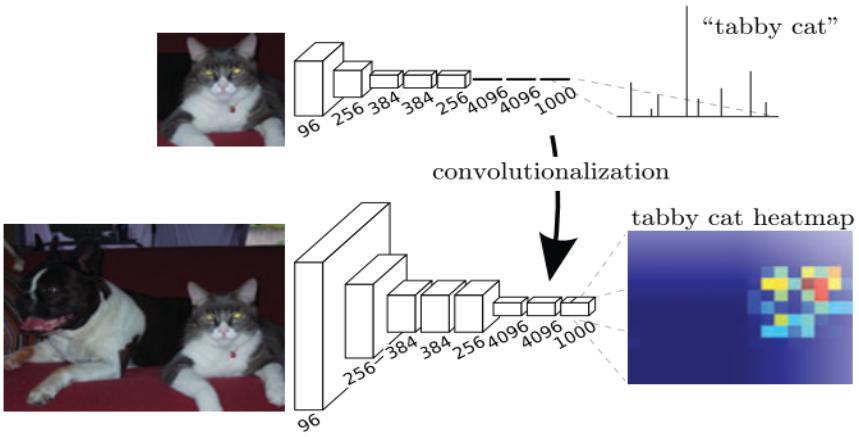


FIGURE 3.25: Transforming fully connected layers into convolution layers. [18]

3.5 Instance Segmentation

Based on the semantic segmentation, instance segmentation extends to distinguish multiple objects belonging to the same class and draw their borders in pixel level in an image (Figure 3.26).

FIGURE 3.26: Instance Segmentation. [13]

3.5.1 Mask R-CNN

From Fast R-CNN, we know the RoI Pooling layer can integrate CNN layer, SVM Classification and bbox regression together, make multi-stage training pipeline into single-stage, speed up the training process. However, in the quantization RoI process, rounding operation introduces misalignments between the extracted features and the RoI, which brings a big negative impact on pixel-level masks [19]. Mask R-CNN introduces RoI Align which applies bilinear interpolation instead of rounding to get the accurate values of the input features [19].

Mask R-CNN adds the branch of FCN layer (Mask layer) for semantic mask prediction on the base of Faster R-CNN, therefore Mask R-CNN can be regarded as Mask R-CNN = Faster R-CNN + FCN + RoIAlign. Hence, Mask R-CNN can combine classification, objection detection and segmentation together. The following picture shows the framework for Mask R-CNN (Figure 3.27).

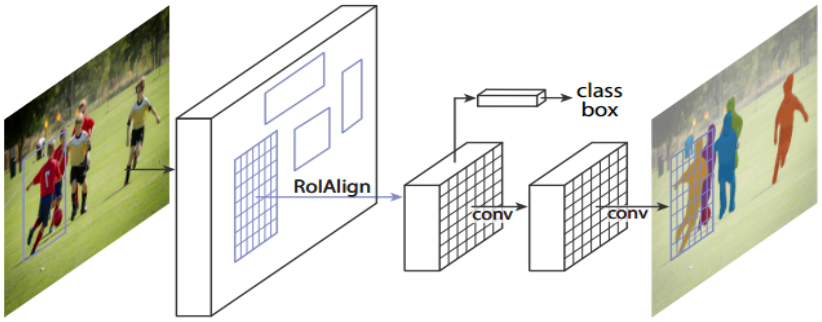


FIGURE 3.27: Mask R-CNN Architecture.
[19]

Let us explore the mask branch. The semantic output of an input image is a segmentation map where each pixel represents a class label whose value is an integer($height * width * 1$) (Figure 3.28) [20].

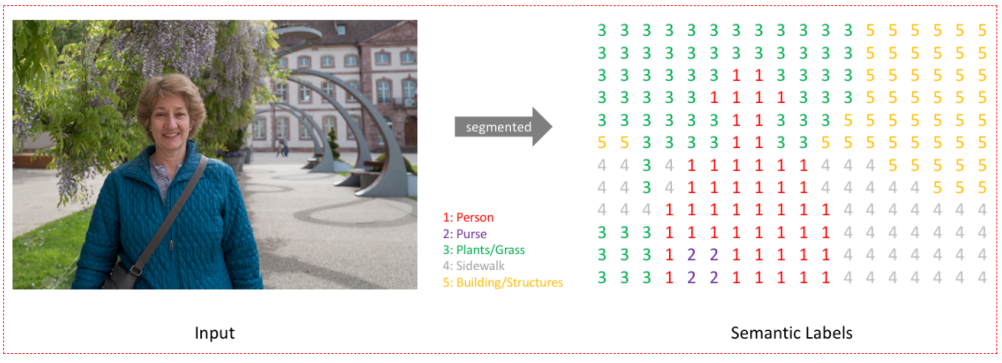


FIGURE 3.28: Semantic Labels.
[20]

If we split the output of each possible class to different channels by on-hot encoding, we can get the below multiple channels output (Figure 3.29) [20]:

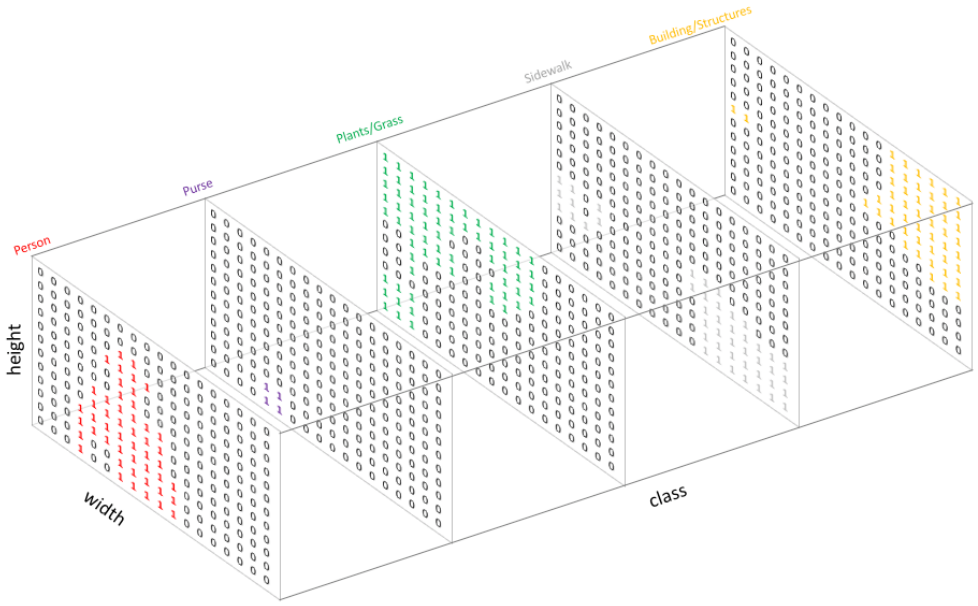


FIGURE 3.29: Multiple Channel Semantic Outputs.
[20]

If we overlay the single channel semantic labels to the original image, the result is a mask which illustrates which a specific class locates in which region (Figure 3.30) [20].

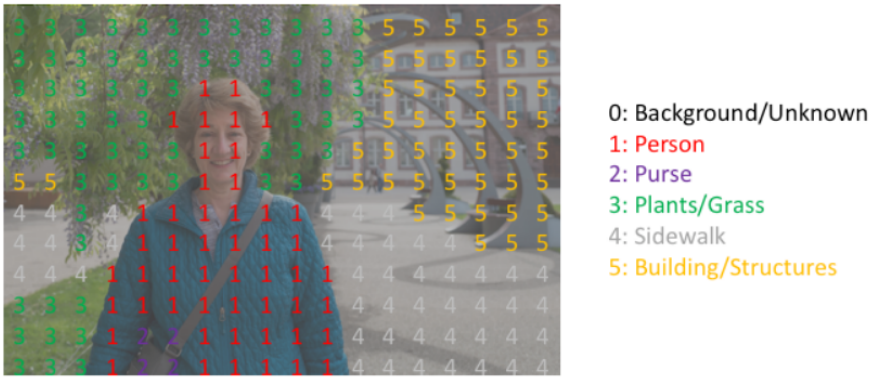
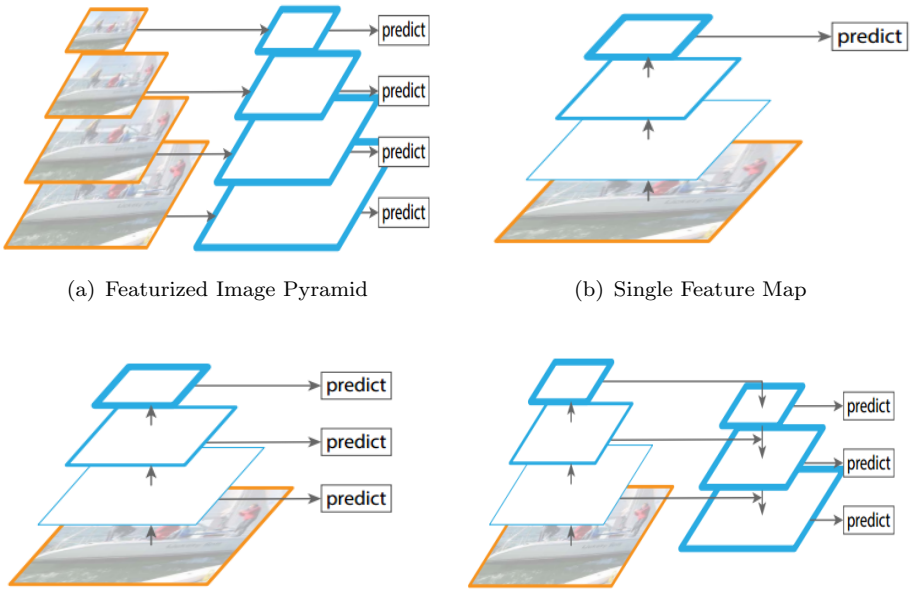


FIGURE 3.30: Mask Output.
[20]

FPN Masker R-CNN uses ResNet as the backbone, after ResNet extracting features, the subsequent part is FPN which is used for constructing multi-scale pyramidal hierarchy structure to obtain each-level semantic features. The next part will introduce the evolution from featurized image pyramid to feature pyramid network.



(a) Featurized Image Pyramid

(b) Single Feature Map

(c) Pyramidal Feature Hierarchy

(d) Feature Pyramid Network

FIGURE 3.31: FPN evolution.
[21]

Featurized Image Pyramid Traditional featurized image pyramid is first to make an image pyramid to different scales image, then extract feature maps from different scales image. Obviously, this is computation and space intensive (Figure 3.31(a)) [21] .

Single Feature Map This architecture uses CNN to extract features in different scales. Single feature map just uses the highest-level feature map. The disadvantage is losing the other all levels semantic features (Figure 3.31(b)) [21] .

Pyramidal Feature Hierarchy Compared to single feature map, pyramidal feature hierarchy use several different scales feature maps, however, it does not take full advantage of all different scales feature maps, especially high-resolution feature maps, which is particularly important for small objects detection. Because high-level feature maps have the characters of strong semantic, large receptive field, and low-level feature maps have high-resolution, and also keep the semantics for small objects (Figure 3.31(c)) [21] .

Feature Pyramid Network FPN makes full use of each level feature maps, upsamples higher-level feature maps to higher solution feature maps, afterwards make a lateral connection with next low-level feature maps, which can reinforce high-level feature and in the meantime assure the adjacent two levels with lateral connections have the same spatial size (Figure 3.31(d)) [21] .

There are two pathways for FPN process:

1. Bottom-up pathway
   The CNN forward process is from bottom to up, the feature maps shrink in multiples through convolutional kernel computation due to pooling layer. The convolutional layers which output the same size feature maps are called a stage. When building feature pyramid, take the feature maps outputted by the last layer of each stage as a pyramid level, because the later layer has stronger features.
2. Top-down pathway
   - Upsample high-level feature by nearest neighbour interpolation method, which causes spatial resolution is magnified twice. The output feature map is denoted map_{up} .
   - The low-level feature maps are through 1×1 convolutional layer to make the channel size of low-level feature maps consistent with high-level's. The output feature map is denoted map_{down}.
   - Make element-wise addition of map_{up} and map_{down}, iterate until generating the finest resolution map.
   - The final feature map is obtained by using 3×3 convolution on each merged feature map above.

Mask R-CNN Architecture The Mask R-CNN architecture is as follows (Figure 3.33), for comparison, we also draw the Faster R-CNN architecture (Figure 3.32). conv1, conv2_x, conv3_x, conv4 and conv5 is the five stages of ResNet, after conv4_x, the output is linked to FPN, then RPN. The difference between Faster R-CNN and Mask R-CNN is RoI Align is used in Mask R-CNN instead of RoI Pooling to get fixed length feature vector. Mask R-CNN has one more output branch mask after RoI Align than Faster R-CNN.

The final output of $conv4\_x$ ($14 \times 14 \times 1024$) is shared for FPN, RPN and RoI Align. The follows are two types Mask R-CNN heads architecture extended from two existing Faster R-CNN heads, the left panel is based on ResNet C4, the right panel is from ResNet FPN (Figure 3.34) [19]. It is obviously indicated that the head from C4 is heavier than from FPN. The project uses ResNet-FPN as a backbone.

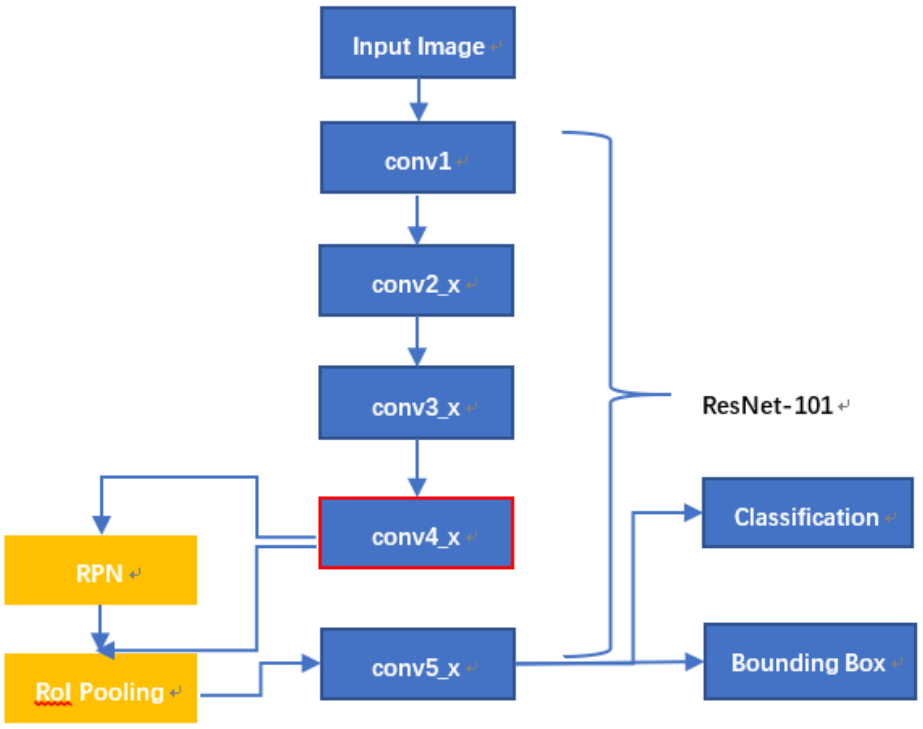


FIGURE 3.32: Faster R-CNN Whole Architecture.

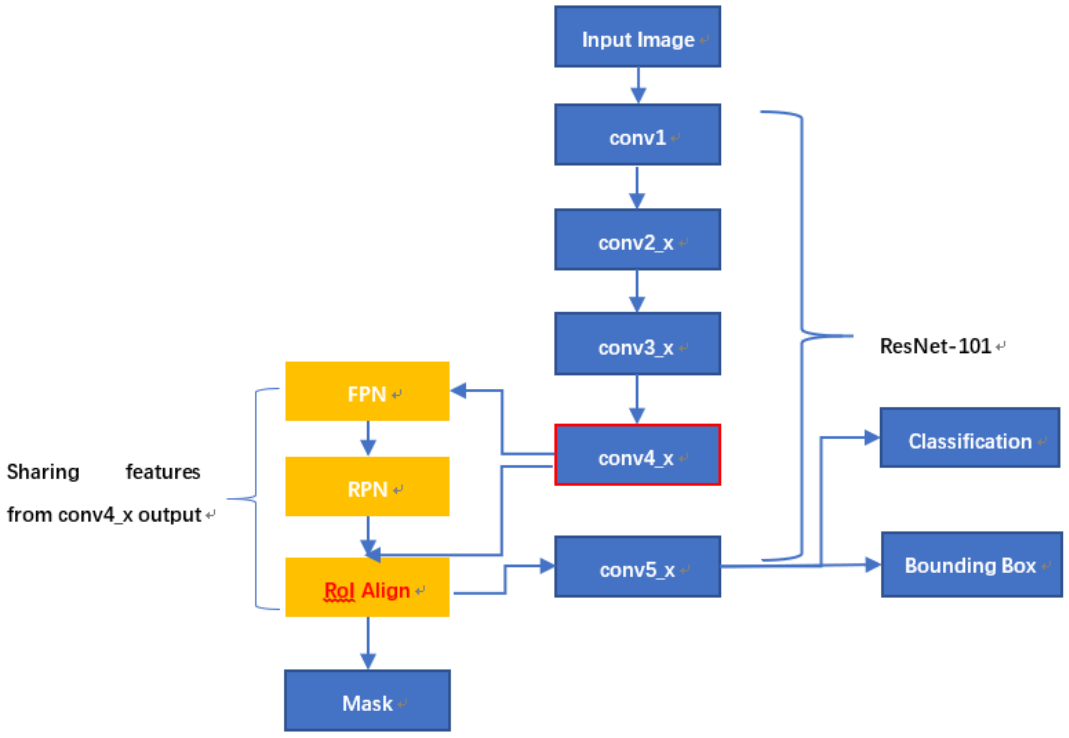


FIGURE 3.33: Mask R-CNN Whole Architecture.

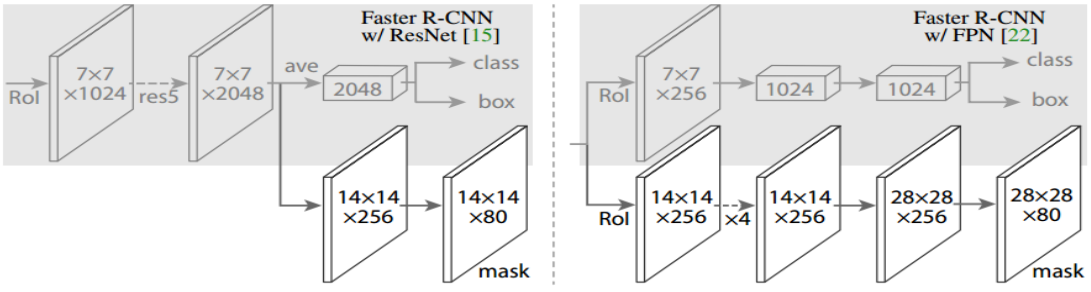


FIGURE 3.34: Mask R-CNN Head Architecture.
[19]

Chapter 4

Methodology

This part will introduce the techniques and tools applied in this project.

4.1 Data Sets

This project uses an open-source data set "acfr-multifruit-2016" gathered by the agriculture team from the University of Sydney. The data set includes images and annotations for three types of fruits: Apples, Mangoes and Almonds. This project applies the Mask R-CNN algorithm on apple data set to achieve apple recognition, localization and segmentation. The apples data set summary of is as follows (Table 4.1):

Fruit	**Image Info**	**Annotation Info**	**Notes**
Apples	1120 Images, PNG Image Data, 308 x 202, 8-bit/color RGB , Sensor: PointGrey Ladybug3	Circle Annotation (x,y,radius), Pixel-wise annotation	Collected at Warburton, Australia Apple varieties: Pink Lady, Kanzi

Table 4.1: The apples data set summary
[22]

The data set structure is as follows Figure 4.1. The images directory lists all images including training, validation and test parts. The annotations directory contains the corresponding annotation of each image. The Sets directory indicates which images are used for training, validation and test. Because the annotation is a circle, the masks for apples utilize circles instead of polygons in this project. In the beginning, we use a piece of python code to automatically read the filename of the corresponding image from the three files "train.txt", "val.txt" and "test.txt" then put the image files and annotation files to three corresponding directories "train", "val" and "test" (code see 3). For the

apple data set, there are 896 images for training, 112 images for validation and 112 images for testing.

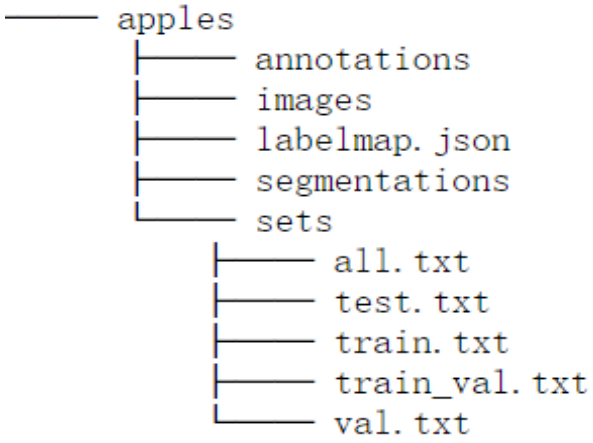


FIGURE 4.1: The Data Set Structure for Apples. [22]

4.2 Label Data Sets

Mask R-CNN model is for supervised learning. Labeling data sets is mandatory. There are a lot of tools for annotating images, e.g. VIA(VGG Image Annotator), LabelMe, RectLabel, LabelBox, COCOUI [3], PychetLabeller [22]. Different labeling tools have different annotations format. The project uses PychetLabeller as the annotation tool. The format of the annotation is CSV.

4.3 Data Augmentation

For deep learning, if the model wants to get a good performance, it needs sufficient size of data set. However, in practical use, there often does not exist enough size of data set, under this circumstance, the data augmentation technology can be applied. The premise of data augmentation is the invariance property of CNNs, i.e. a CNN can be invariant to translation, size, rotation or a combination of the above. Hence, data augmentation uses translations, flips or rotations to increase the data set size [23].

4.4 Transfer Learning

Due to the relatively insufficient size of data set, it is rare to train an entire CNN from scratch. The common method is to pre-train a CNN model on a very large data set(e.g.

ImageNet or COO), and then fine-tuning the parameters to get the final result [24]. The key process is

1. Remove the last fully-connected layer due to the different output dimension, add you own classifier. Take the rest of the CNN as a fixed feature extractor for the new data set.
2. Keep some of the earlier layers fixed, because the earlier features include more generic features. Fine-tune some high-level layers of the network.
3. Use a smaller learning rate for later layers fine-tuning. [24]

4.5 Remote Connection

Because the server is at remote end. It needs to use remote login tools to connect the server. When renting a server, it will generate the parameters: user name, remote IP, and port for remote login. The remote server needs an SSH connection mode. We use putty as the remote login tool. First, use "puTTYgen" to generate an SSH key pair. Second, set the public key on the server. After everything is done, we can use "putty" software to remote login. The operating system of the server is Ubuntu 16.04 LTS, hence, it needs to use some Linux command to run transfer files and program. For convenience, we can use "FileZilla FTP Client" software with SFTP to transfer files.

Chapter 5

Code Explanation

The project references the open-source mature code framework from the GitHub and references some examples associated with the code, how to use the API [1]. The code was developed by Python. The amount of source code is around 5000 lines. It uses TensorFlow as the back end, Keras as the front end. Now, let us explore the source code architecture.

5.1 Code Architecture

The "model.py" is the main code file implementing the whole model architecture. There are six classes in model.py.

1. BatchNorm: The batch normalization layer.
2. ProposalLayer: First read the output of the RPN, remove overlaps of regional proposals based on anchor scores and non-max suppression. Then apply bbox refinement deltas to anchors, and select a subset of anchors to pass as proposals to the next stage.
3. PyramidROIAlign: Implement RoI Align on multiple levels of the feature pyramid.
4. DetectionTargetLayer: Subsample proposals and generate target box refinement, class_ids, and masks for each proposal.
5. DetectionLayer: Return the final detection boxes based on classified proposal boxes and their bounding box deltas.
6. MaskRCNN: Provide various Mask RCNN model functionalities. [3][1]

Next, we introduce some important functions:

- MaskRCNN.build(): Call functions from above classes to build each layer of the model. The calling order of above classes is ProposalLayer → DetectionTarget-Layer → DetectionLayer. The specific calling order for layers and functions is as follows (Figure 5.1). The blue arrow indicates the training condition, whereas the yellow arrow shows the inference condition. After "ProposalLayer", if the training condition, next order is "DetectionTargetLayer", if the inference condition, the next order is fpn_classifier_graph() function. The blue rectangle shows the class, the yellow rectangle represents the function. The green arrow indicates the call function. Except it, other arrows represent the sequential order. In addition, the sequential order of anchors, bbox, RoIs is anchors → RoIs → bbox.

  The model architecture is shown as follows (Figure 5.2). In build function, call the function resnet_graph() to get five stages of ResNet from C1 to C5, then use upsampling to get FPN P2 to P6. P6 is utilised in RPN, rather than in the classifier heads.

- resnet_graph(): Implement the ResNet backbone, return the five stages from C1 to C5.
- rpn_graph(): Construct the computation graph of RPN.
- build_rpn_model(): Wrap the RPN graph for repetitive use of shared weights.
- build_fpn_mask_graph(): Generate the mask branch e.g. mrcnn_mask_conv1.
- fpn_classifier_graph(): Create the classifier and bounding box regressor, generate layers e.g. mrcnn_class_conv1. [3] [1]

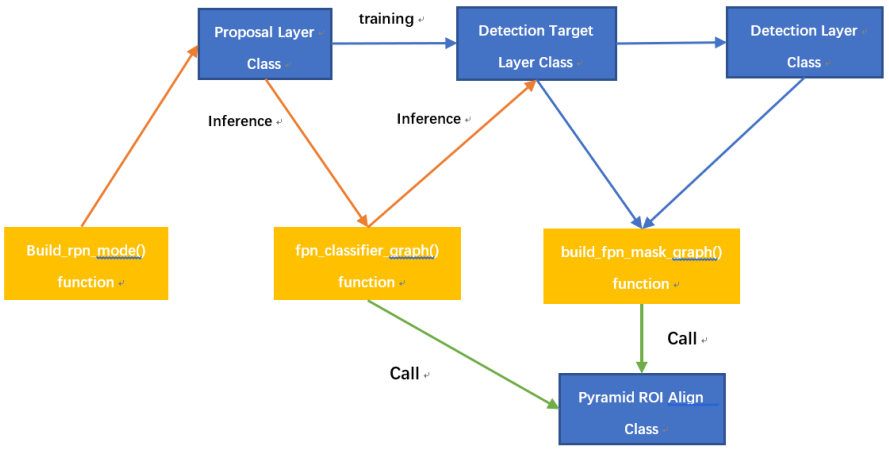


FIGURE 5.1: Code Structure of Mask R-CNN's build function.

5.2 API Call to Run on Your Own Data Set

The calling process to implement on your own data set can be divided into 5 steps:

```
fpn_c5p5                (Conv2D)
fpn_c4p4                (Conv2D)
fpn_c3p3                (Conv2D)
fpn_c2p2                (Conv2D)
fpn_p5                  (Conv2D)
fpn_p2                  (Conv2D)
fpn_p3                  (Conv2D)
fpn_p4                  (Conv2D)
In model:  rpn_model
    rpn_conv_shared         (Conv2D)
    rpn_class_raw           (Conv2D)
    rpn_bbox_pred           (Conv2D)
mrcnn_mask_conv1        (TimeDistributed)
mrcnn_mask_bn1          (TimeDistributed)
mrcnn_mask_conv2        (TimeDistributed)
mrcnn_mask_bn2          (TimeDistributed)
mrcnn_class_conv1       (TimeDistributed)
mrcnn_class_bn1         (TimeDistributed)
mrcnn_mask_conv3        (TimeDistributed)
mrcnn_mask_bn3          (TimeDistributed)
mrcnn_class_conv2       (TimeDistributed)
mrcnn_class_bn2         (TimeDistributed)
mrcnn_mask_conv4        (TimeDistributed)
mrcnn_mask_bn4          (TimeDistributed)
mrcnn_bbox_fc           (TimeDistributed)
mrcnn_mask_deconv       (TimeDistributed)
mrcnn_class_logits      (TimeDistributed)
mrcnn_mask              (TimeDistributed)
```

Figure 5.2: Structure of Mask R-CNN Model.

- **Configurations**: Set some configurations for training and inference, derives from the base Config class and overrides some values.
- **Load Data Set**: Override three functions:
  - **load_fruit**: Load a subset of the fruit data set.
  - **load_mask**: Generate instance masks for an image.
  - **image_reference**: Return the path of the image.
- **Load Model**: Create a model object in inference mode, load pre-trained weights.
- **Run Detection**: Call detect() function to run object detection, and call display_instances() to show results.
- **Evaluation**: Call visualize.plot_precision_recall() function to draw precision-recall curve. Call visualize.plot_overlaps() function to draw grid of ground truth objects and their predictions. Compute mAP @ IoU=50 on Batch of Images, first compute mAP on each image of data set, then compute the mean of all mAP of each image.(code see 3 3) [3] [1]

There are several differences from the original paper:

- **Image Resizing**: Resize all images to the same size by padding an image with zeros to be square in order to implement training multiple images per batch.
- **Bounding Boxes**: Compute the bbox through the mask, pick the smallest box that includes all the pixel of the mask as a bbox in order to scale, rotate and crop images easily.
- **Learning Rate**: 0.02 is high gradient clipping which often results in weights explosion, especially when using a small batch size. Hence, the project utilises the learning rate at 0.001.
- **Mini mask**: When training high-resolution images, the mask of each target also takes up big space. Because most of the values in the mask binary matrix are 0, which wastes much space. Thus, we optimise the store method: store the mask inside the object bbox to reduce 0 value. Although adjusting the mask size to a smaller size will lose some accuracy for objects which are bigger than the selected size. However, most object annotations are not very accurate. Therefore, the loss can be ignored in most case. Afterwards, use expand_mask() function to obtain the full mask of objects. [3] [1]

Chapter 6

Results and Discussion

In this section, we will evaluate the impact for the result due to the change of parameters and some structures of the algorithm in terms of accuracy and execution time. ResNet is utilised as the backbone for the Mask R-CNN model on apple data Set stated before to achieve apple recognition, localization and segmentation. ResNet-50-FPN and ResNet-101-FPN architectures are applied in this project. The Mask R-CNN model is quite complicated, which is time-consuming and therefore needs strong GPUs to support training. The project rents cloud GPU from vast.ai. The parameters for rented GPU is 2*GTX 1080 Ti with 11.2GB Per-GPU memory, CUDA 10.0, 28.1 TFLOPS, 339.3GB/s Per-GPU Memory Bandwidth. The original Mask R-CNN paper uses Nvidia Tesla M40 GPU single 8-GPU machine [19]. It takes one to two days for training on COCO data set, and the models run at about 200ms per image [19]. The result detection is implemented on google cloud platform Collaboratory, whose GPU is Tesla K80.

The following, we will compare, analyse, and discuss the below consequences. We will randomly choose one picture from validation and test data set respectively, compare all models' results reflecting in the same picture. The results will be divided into five parts: ground truth data, prediction data, precision recall curve, grid of ground truth objects and their predictions, and finally list the table of precision, training time and inference time both on validation and test data set for a summary.

- The result of ResNet-101-FPN for heads training and all layers fine-tuning,
- The result of ResNet-50-FPN for heads training both from COCO and ImageNet pre-trained model,
- The result of ResNet-50-FPN for heads training in data augmentation from COCO pre-trained model.

Before comparison, there are two concepts to explain: heads and all layers. The heads refer to the RPN, classifier and mask heads of the network(i.e. the last several layers). All

layers contain all layers of ResNet. For mAP @ IoU=50 on a batch of Images, it means to compute the mAP of each image on validation or test data set then take the mean of the sum of mAP. It is noted that mAP represents the mask mAP, because as analysed before in chapter "code explanation", bounding boxes are generated dynamically from the masks, the box mAP is equal to the mask mAP.

6.1 ResNet-101-FPN for heads training and all layers fine-tuning

This section will compare the performance between heads training and all layers fine-tuning, how much accuracy all layers fine-tuning can improve.

Configuration Below is the configuration of the Mask R-CNN model. In addition, the model uses 15 epochs for heads training, 10 epochs for all layers fine-tuning.

```
Configurations:
BACKBONE                       resnet101
BACKBONE_STRIDES               [4, 8, 16, 32, 64]
BATCH_SIZE                     1
BBOX_STD_DEV                   [0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE         None
DETECTION_MAX_INSTANCES        100
DETECTION_MIN_CONFIDENCE       0.9
DETECTION_NMS_THRESHOLD        0.3
FPN_CLASSIF_FC_LAYERS_SIZE     1024
GPU_COUNT                      1
GRADIENT_CLIP_NORM             5.0
IMAGES_PER_GPU                 1
IMAGE_CHANNEL_COUNT            3
IMAGE_MAX_DIM                  1024
IMAGE_META_SIZE                14
IMAGE_MIN_DIM                  800
IMAGE_MIN_SCALE                0
IMAGE_RESIZE_MODE              square
IMAGE_SHAPE                    [1024 1024    3]
LEARNING_MOMENTUM              0.9
LEARNING_RATE                  0.001
LOSS_WEIGHTS                   {'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0, 'mrcnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0, 'mrcnn_mask_loss': 1.0}
MASK_POOL_SIZE                 14
MASK_SHAPE                     [28, 28]
MAX_GT_INSTANCES               100
MEAN_PIXEL                     [123.7 116.8 103.9]
MINI_MASK_SHAPE                (56, 56)
NAME                           apple
NUM_CLASSES                    2
POOL_SIZE                      7
POST_NMS_ROIS_INFERENCE        1000
POST_NMS_ROIS_TRAINING         2000
PRE_NMS_LIMIT                  6000
ROI_POSITIVE_RATIO             0.33
RPN_ANCHOR_RATIOS              [0.5, 1, 2]
RPN_ANCHOR_SCALES              (32, 64, 128, 256, 512)
RPN_ANCHOR_STRIDE              1
RPN_BBOX_STD_DEV               [0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD              0.7
RPN_TRAIN_ANCHORS_PER_IMAGE    256
STEPS_PER_EPOCH                100
TOP_DOWN_PYRAMID_SIZE          256
TRAIN_BN                       False
TRAIN_ROIS_PER_IMAGE           200
USE_MINI_MASK                  True
USE_RPN_ROIS                   True
VALIDATION_STEPS               50
WEIGHT_DECAY                   0.0001
```

FIGURE 6.1: Configuration of the model.

Ground Truth Data on Validation Data Set and Test Data Set The ground truth data picture on validation data Set and test data Set below contains six apple

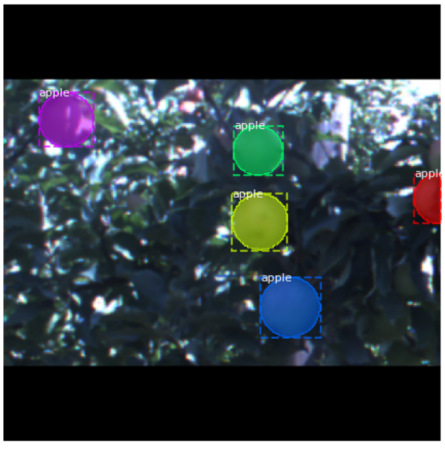
and four apple instances respectively, the circle is the mask of each apple, the rectangle surrounding the mask is bbox (Figure 6.2(a), Figure 6.3(a)). The ground truth information indicates five parts information: original image, image_meta, gt_class_id, gt_bbox, gt_mask. For validation data Set, the resolution of original image is 1024 × 1024 for R,G,B three channels, hence, the shape is 1024 × 1024 × 3. There are six instances in the picture, thus, the first shape of gt_class_id and gt_bbox is six, and the third shape of gt_mask is six as well (Figure 6.2(b)).

(a) Ground Truth Image

```
original_image    shape: (1024, 1024, 3)    min:    0.00000  max:  255.00000  uint8
image_meta        shape: (14,)              min:    0.00000  max: 1024.00000  float64
gt_class_id       shape: (6,)               min:    1.00000  max:    1.00000  int32
gt_bbox           shape: (6, 4)             min:  162.00000  max:  962.00000  int32
gt_mask           shape: (1024, 1024, 6)    min:    0.00000  max:    1.00000  bool
```

(b) Ground Truth Infomation

FIGURE 6.2: Ground Truth Data on Validation Data Set.



(a) Ground Truth Image

```
original_image    shape: (1024, 1024, 3)    min:    0.00000  max:  255.00000  uint8
image_meta        shape: (14,)              min:    0.00000  max: 1024.00000  float64
gt_class_id       shape: (5,)               min:    1.00000  max:    1.00000  int32
gt_bbox           shape: (5, 4)             min:   82.00000  max: 1024.00000  int32
gt_mask           shape: (1024, 1024, 5)    min:    0.00000  max:    1.00000  bool
```

(b) Ground Truth Infomation

FIGURE 6.3: Ground Truth Data on Test Data Set.

6.1.1 Heads Training

Losses and Training Time Below is the various losses and the training time for ResNet-101-FPN Heads training (Figure 6.4). The whole process takes around 15 hours, which is time-consuming. It takes around average 57.43S per image.

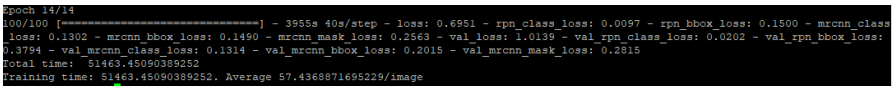


FIGURE 6.4: Losses and Training Time for ResNet-101-FPN Heads Training

6.1.1.1 Validation Data Set

Prediction Data Below is the prediction data for heads training model (Figure 6.5). It can be shown from the prediction image that there are four prediction instances, two instances have not been predicted. The character and number on bbox is the predicted class and the highest score of anchor foreground class. From the prediction information, we can see there are around 20K anchors. Image_metas is a 1D array combining all attributes of an image such as image_id, original_image_shape, window (the area of the image where the real image is), scaling factor applied to the original image, active_class_ids which is a list of class_ids available in the dataset where the image came from [3]. Molded_image is an RGB array of images which minus the mean pixel and is converted to float [3].

(a) Prediction Image

(b) Prediction Infomation

FIGURE 6.5: Prediction Data for ResNet-101-FPN Heads Training on Validation Data Set.

Precision-Recall Curve The precision-recall curve on IoU=50 is shown as follows (Figure 6.6). The recall corresponding to the curve turning point is 0.667.

Grid of ground truth objects and their predictions Below is the grid of ground truth objects and their predictions (Figure 6.7). The vertical axis indicates the prediction with the highest score of anchor foreground class. The horizontal axis shows the ground truth. We can see the highest match score between predictions and the ground truth. There is one empty column because it lacks one prediction for one instance.

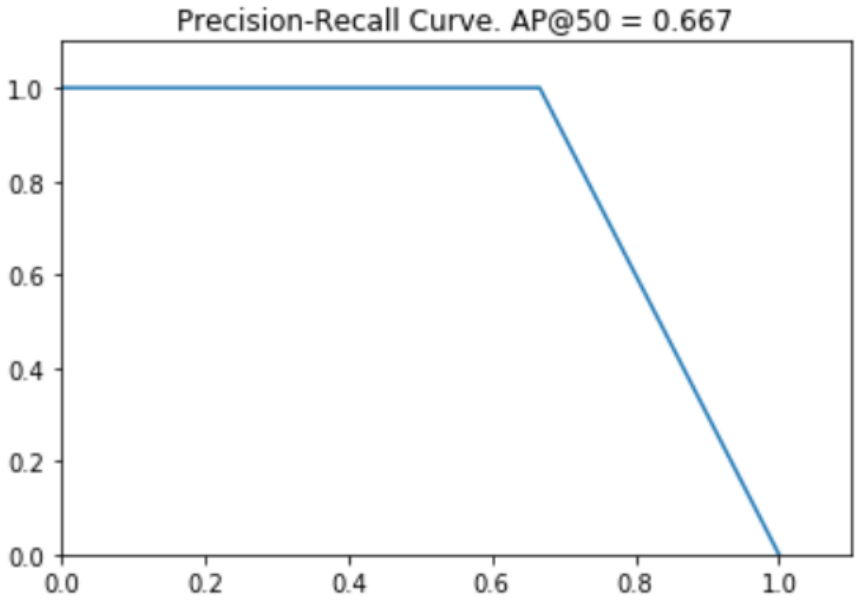


FIGURE 6.6: Precision-Recall Curve for ResNet-101-FPN Heads Training on Validation Data Set.

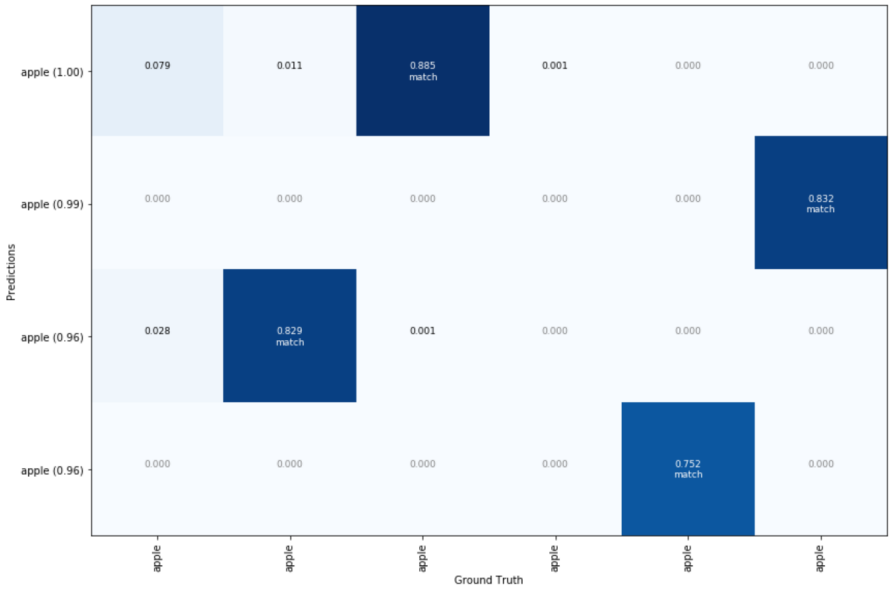


FIGURE 6.7: Grid of ground truth objects and their predictions for ResNet-101-FPN heads training on validation data Set.

mAP @ IoU=50 on Batch of Images The final precision result is 0.681 (Figure 6.8). The prediction time for each image is around 0.63S compared to original Mask R-CNN paper on COCO trainval35k is 0.72s [19].

```
Prediction time: 70.63103461265564s. Average 0.6306342376129968s/image
Total time: 101.36222982406616s
mAP @ IoU=50:  0.6818577837622749
```

FIGURE 6.8: mAP and Inference Time for ResNet-101-FPN Heads Training on Validation Data Set.

6.1.1.2 Test Data Set

This section will validate the corresponding results on the test data set.

Prediction Data The prediction data image and information is as follows (Figure 6.9). The result predicts all instances, but one of them is not corresponding to the label.

(a) Prediction Image

```
image          shape: (1024, 1024, 3)      min:    0.00000  max:  255.00000  uint8
molded_images  shape: (1, 1024, 1024, 3)   min: -123.70000  max:  151.10000  float64
image_metas    shape: (1, 14)              min:    0.00000  max: 1024.00000  int64
anchors        shape: (1, 261888, 4)       min:   -0.35390  max:    1.29134  float32
```

(b) Prediction Infomation

FIGURE 6.9: Prediction Data for ResNet-101-FPN Heads Training on Test Data Set.

Precision-Recall Curve The precision-recall curve on IoU=50 is shown as follows (Figure 6.10). The recall corresponding to the curve turning point is 0.800.

Grid of ground truth objects and their predictions The matching grid is as follows (Figure 6.11). As shown before, one prediction is not corresponding to the label. Thus, there is no match score for the prediction. Besides, there is no prediction for one instance result in one empty column.

mAP @ IoU=50 on Batch of Images The average prediction time for per image is around 0.630S (as shown in Figure 6.12). The mAP for IoU=50 is around 0.681.

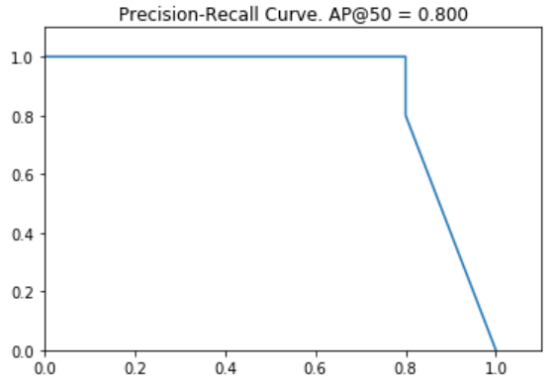


FIGURE 6.10: Precision-Recall Curve for ResNet-101-FPN Heads Training on Test Data Set.

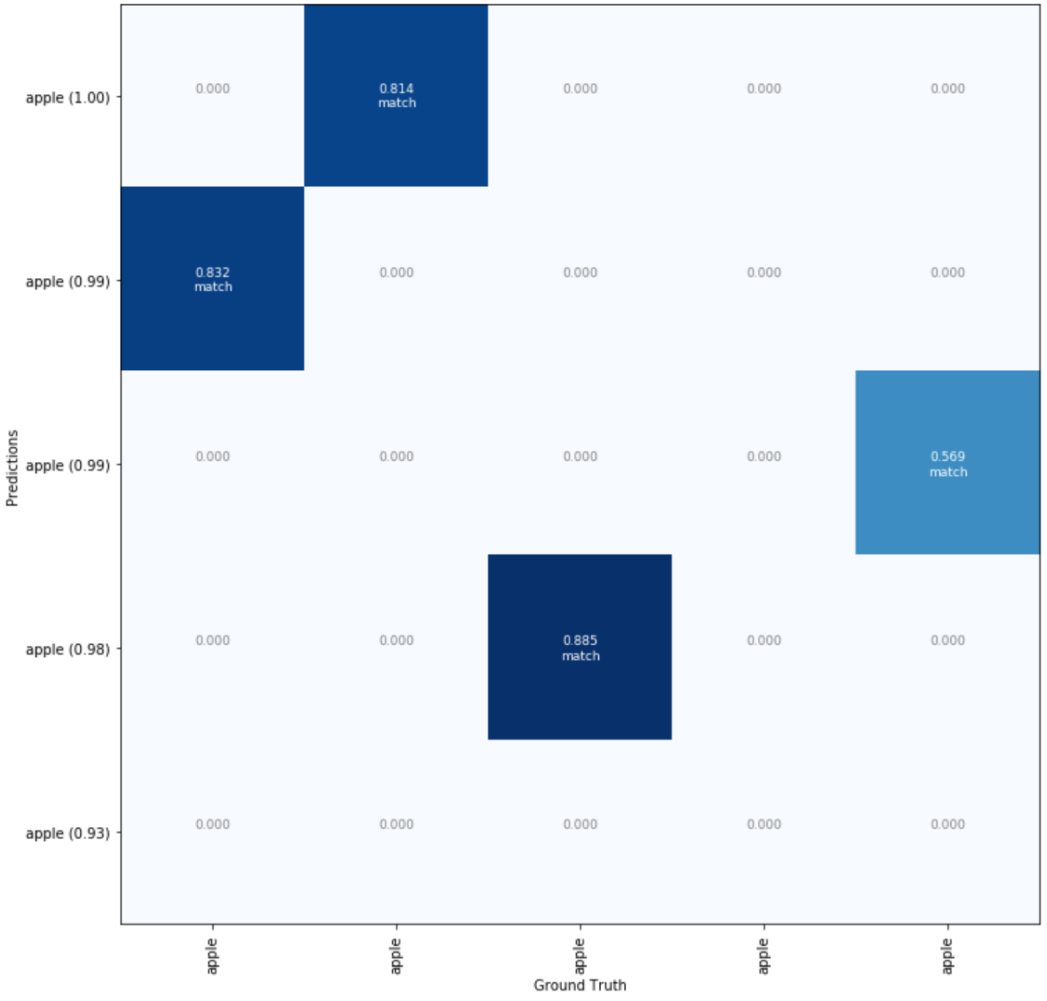


FIGURE 6.11: Grid of ground truth objects and their predictions for ResNet-101-FPN Heads Training on Test Data Set.

```
Prediction time: 70.5929639339447s. Average 0.630294320838792s/image
Total time: 100.95394539833069s
mAP @ IoU=50:  0.6818577837622749
```

FIGURE 6.12: mAP and Inference Time for ResNet-101-FPN Heads Training on Test Data Set.

6.1.2 All Layers Fine-tuning

This section will validate the consequences of all layers fine-tuning for ResNet-101-FPN.

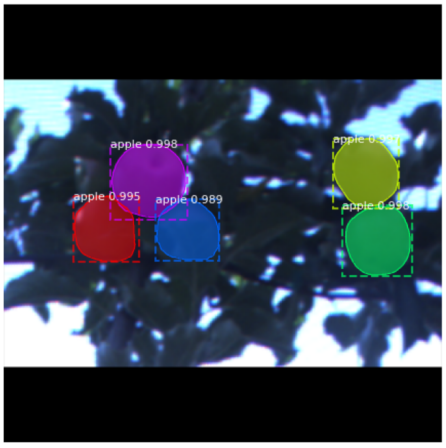
Losses and Training Time Below shows the various losses and training time for all layers fine-tuning (Figure 6.13). The average time is around 62.57S for per image, which is more than heads training (57.43S).

```
Epoch 25/25
100/100 [==============================] - 5085s 51s/step - loss: 0.4616 - rpn_class_loss: 0.0074 - rpn_bb
ox_loss: 0.0802 - mrcnn_class_loss: 0.0990 - mrcnn_bbox_loss: 0.0879 - mrcnn_mask_loss: 0.1871 - val_loss:
 0.6828 - val_rpn_class_loss: 0.0192 - val_rpn_bbox_loss: 0.1096 - val_mrcnn_class_loss: 0.1624 - val_mrcn
n_bbox_loss: 0.1461 - val_mrcnn_mask_loss: 0.2455
Total time:  56068.411068439484
Training time: 56068.411068439484. Average 62.57635163888335/image
```

FIGURE 6.13: Losses and Training Time for ResNet-101-FPN All Layers Fine-tuning

6.1.2.1 Validation Data Set

Prediction Data The prediction image and information on validation data set is as follows (Figure 6.14). The result predicts five instances one more than heads training.



(a) Prediction Image

```
image                    shape: (1024, 1024, 3)       min:    0.00000  max:  255.00000  uint8
molded_images            shape: (1, 1024, 1024, 3)    min: -123.70000  max:  151.10000  float64
image_metas              shape: (1, 14)               min:    0.00000  max: 1024.00000  int64
anchors                  shape: (1, 261888, 4)        min:   -0.35390  max:    1.29134  float32
```

(b) Prediction Infomation

FIGURE 6.14: Prediction Data for ResNet-101-FPN All Layers Fine-tuning on Validation Data Set.

Precision-Recall Curve Below is the corresponding precision-recall curve (Figure 6.15). The recall corresponding to the curve turning point is 0.833, which is higher than heads training(0.667).

Grid of ground truth objects and their predictions The matching score grid is as follows(Figure 6.16). There is one empty column because it lacks one prediction for one instance.

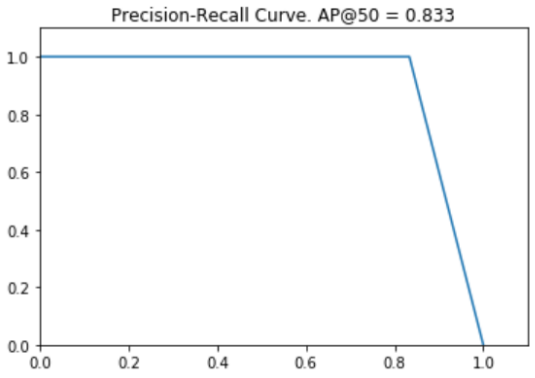


FIGURE 6.15: Precision-Recall Curve for ResNet-101-FPN All Layers Fine-tuning on Validation Data Set.

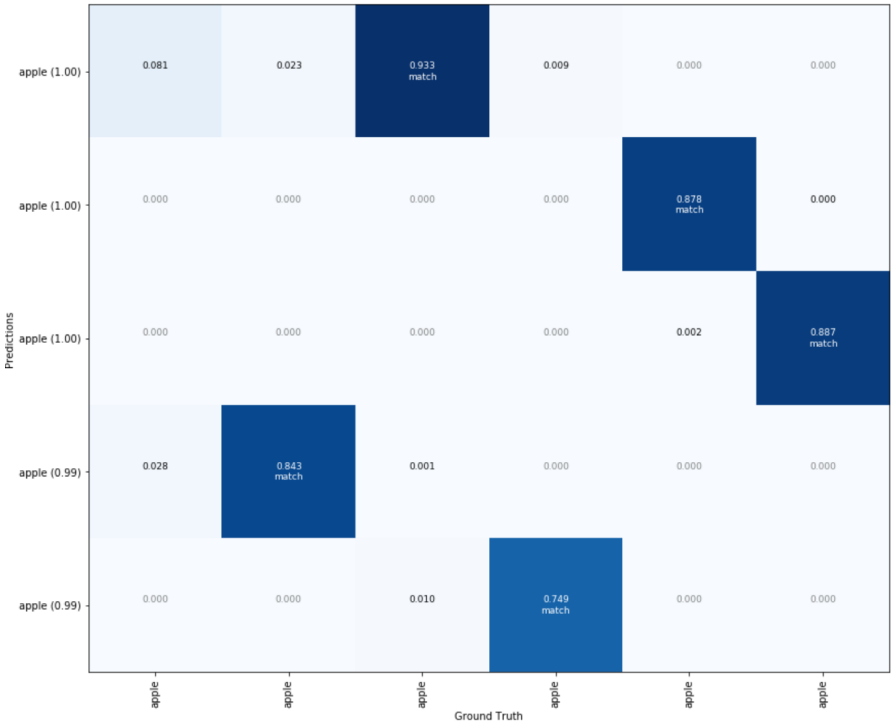


FIGURE 6.16: Grid of ground truth objects and their predictions for ResNet-101-FPN All Layers Fine-tuning on Validation Data Set.

mAP @ IoU=50 on Batch of Images The mAP for all layers fine-tuning is around 0.737 higher than heads training (0.681) (Figure 6.17). The inference time is around average 0.628S per image.

```
Prediction time: 70.41288304328918s. Average 0.6286864557436534s/image
Total time: 100.79533791542053s
mAP @ IoU=50:  0.7379795501922433
```

FIGURE 6.17: mAP and Inference Time for ResNet-101-FPN All Layers Fine-tuning on Validation Data Set.

6.1.2.2 Test Data Set

Prediction Data The prediction data of ResNet-101-FPN for all layers fine-tuning is as follows (Figure 6.18). There are six prediction instances in the picture. We can see one instance which is not pre-labeled.

(a) Prediction Image

```
image                    shape: (1024, 1024, 3)       min:    0.00000  max:  255.00000  uint8
molded_images            shape: (1, 1024, 1024, 3)    min: -123.70000  max:  151.10000  float64
image_metas              shape: (1, 14)               min:    0.00000  max: 1024.00000  int64
anchors                  shape: (1, 261888, 4)        min:   -0.35390  max:    1.29134  float32
```

(b) Prediction Infomation

FIGURE 6.18: Prediction Data for ResNet-101-FPN All Layers Fine-tuning on Test Data Set.

Precision-Recall Curve The corresponding precision-recall curve is as follows (Figure 6.19). The recall corresponding to the curve turning point is 0.933, which is higher than heads training(0.800).

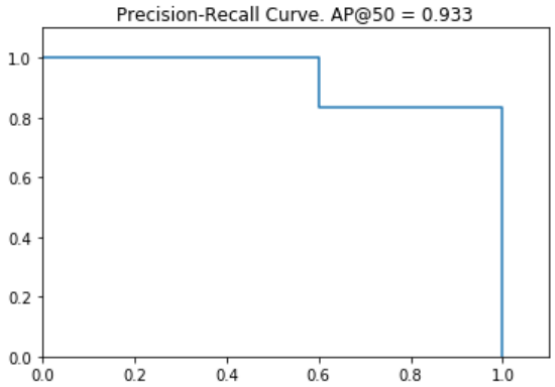


FIGURE 6.19: Precision-Recall Curve for ResNet-101-FPN All Layers Fine-tuning on Test Data Set.

Grid of ground truth objects and their predictions Below is the matching score grid (Figure 6.20). Because there is one prediction instance which is not pre-labeled, there is one empty row in the grid.

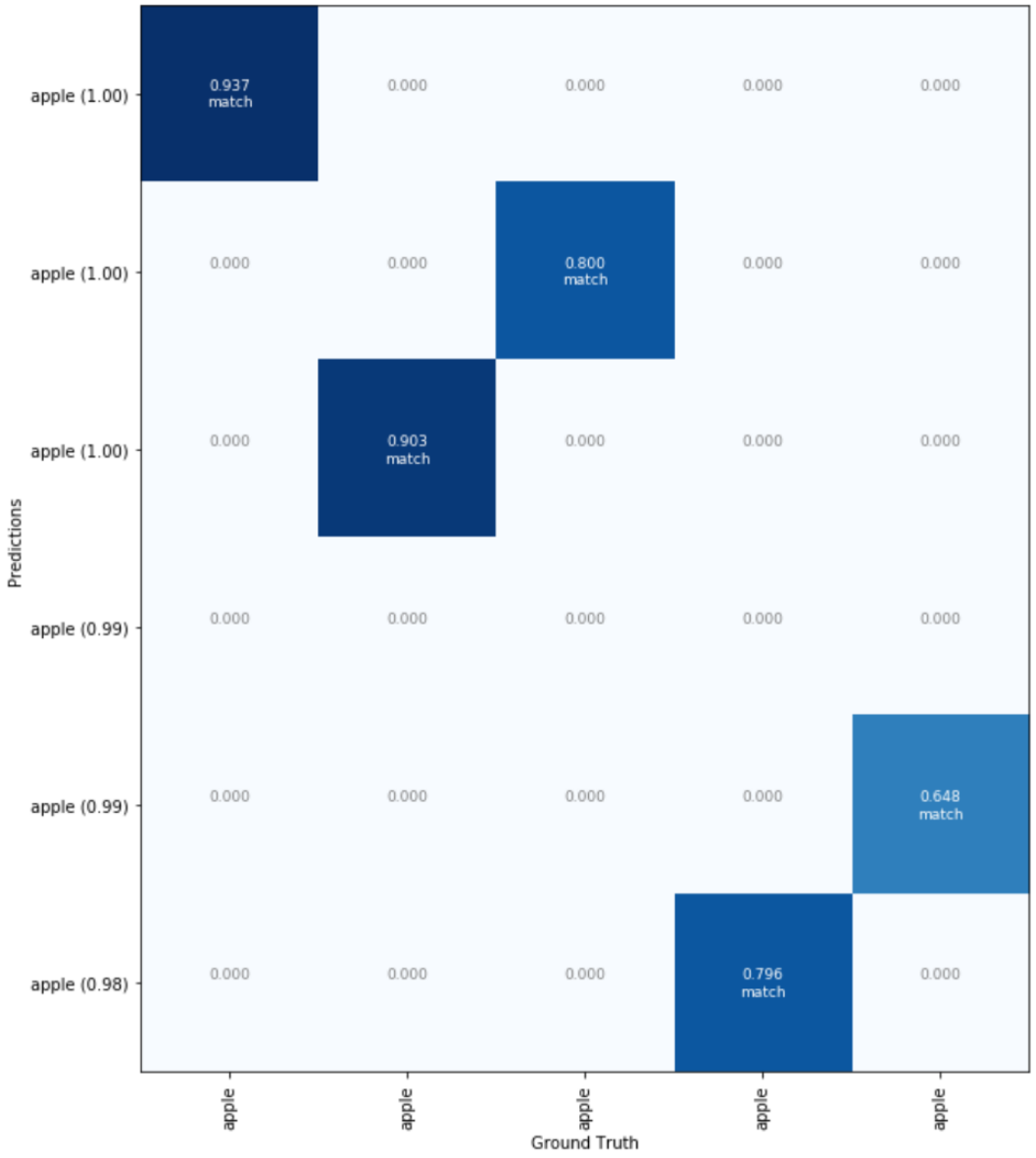


FIGURE 6.20: Grid of ground truth objects and their predictions for ResNet-101-FPN All Layers Fine-tuning on Test Data Set.

mAP @ IoU=50 on Batch of Images We can see the mAP is around 0.737, average prediction time is around average 0.628S per image. (Figure 6.21).

```
Prediction time: 70.4096462726593s. Average 0.6286575560058866s/image
Total time: 100.80939507484436s
mAP @ IoU=50:  0.7379795501922433
```

FIGURE 6.21: mAP and Inference Time for ResNet-101-FPN All Layers Fine-tuning on Test Data Set.

6.2 ResNet-50-FPN for Heads Training

This part will compare ResNet-50-FPN with ResNet-101-FPN architecture, whether the accuracy of Res-50-FPN will be less than ResNet-101-FPN and compare the accuracy from two different pre-trained models. Because the performance of head training and all layers fine-tuning has been evaluated before. This part will just validate head training on the test data set.

6.2.1 From COCO pre-trained model

Losses and Training Time Below shows the various losses and training time for all heads training. (Figure 6.22). The average time is around 61.07S for per image.

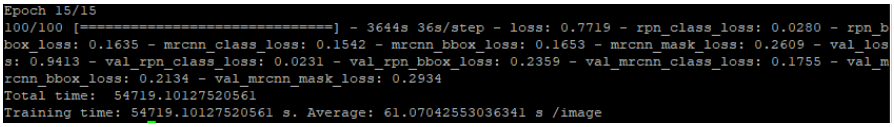


FIGURE 6.22: Losses and Training Time for ResNet-50-FPN Heads Training from COCO pre-trained model.

6.2.1.1 Test Data Set

Prediction Data From the picture, we can see there are five prediction instances, where one is not pre-labeled and the other one is missing. (Figure 6.23).

Precision-Recall Curve The corresponding precision-recall curve is as follows (Figure 6.24). The recall corresponding to the curve turning point is 0.800.

Grid of ground truth objects and their predictions The below is matching score grid (Figure 6.25). Because there is one prediction instance which is not labeled, there is one empty row in the grid. Also, there is one empty column because one instance lacks the corresponding prediction.

mAP @ IoU=50 on Batch of Images We can see the mAP is around 0.692, average prediction time is around 0.509S for per image (Figure 6.26).

6.2.2 From ImageNet pre-trained model

Losses and Training Time Below is the corresponding losses and training time (Figure 6.27). It takes around average 60.92S per image, which is almost the same as from coco pre-trained model (61.07s).

```
image                    shape: (1024, 1024, 3)        min:    0.00000  max:  255.00000  uint8
molded_images            shape: (1, 1024, 1024, 3)     min: -123.70000  max:  151.10000  float64
image_metas              shape: (1, 14)                min:    0.00000  max: 1024.00000  int64
anchors                  shape: (1, 261888, 4)         min:   -0.35390  max:    1.29134  float32
```

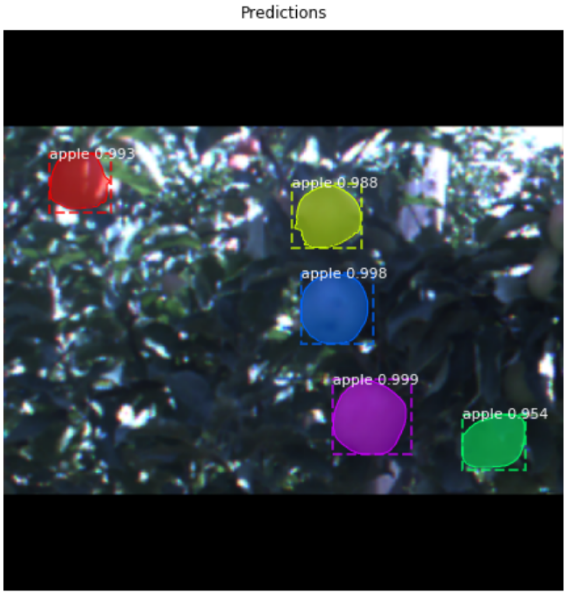


FIGURE 6.23: Prediction Data for ResNet-50-FPN Heads Training from COCO pre-trained model on Test Data Set.

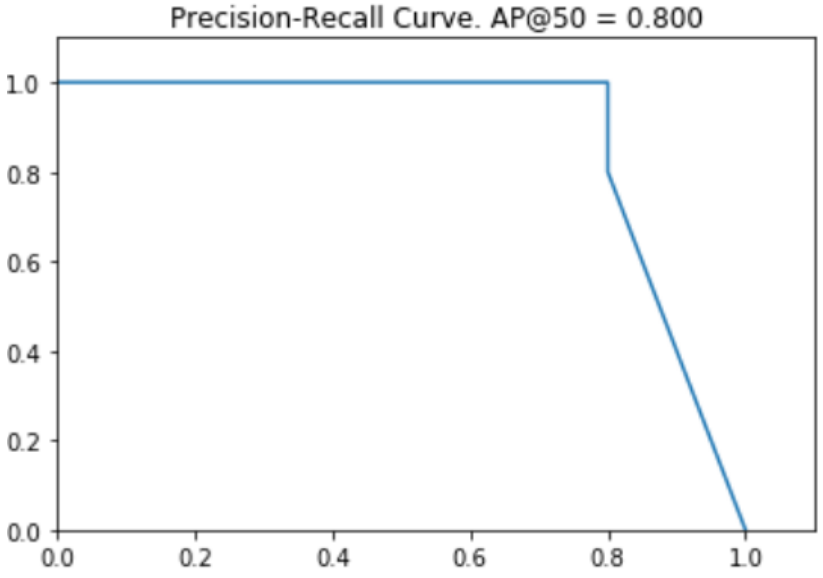


FIGURE 6.24: Precision-Recall Curve for ResNet-50-FPN Heads Training from COCO pre-trained model on Test Data Set.

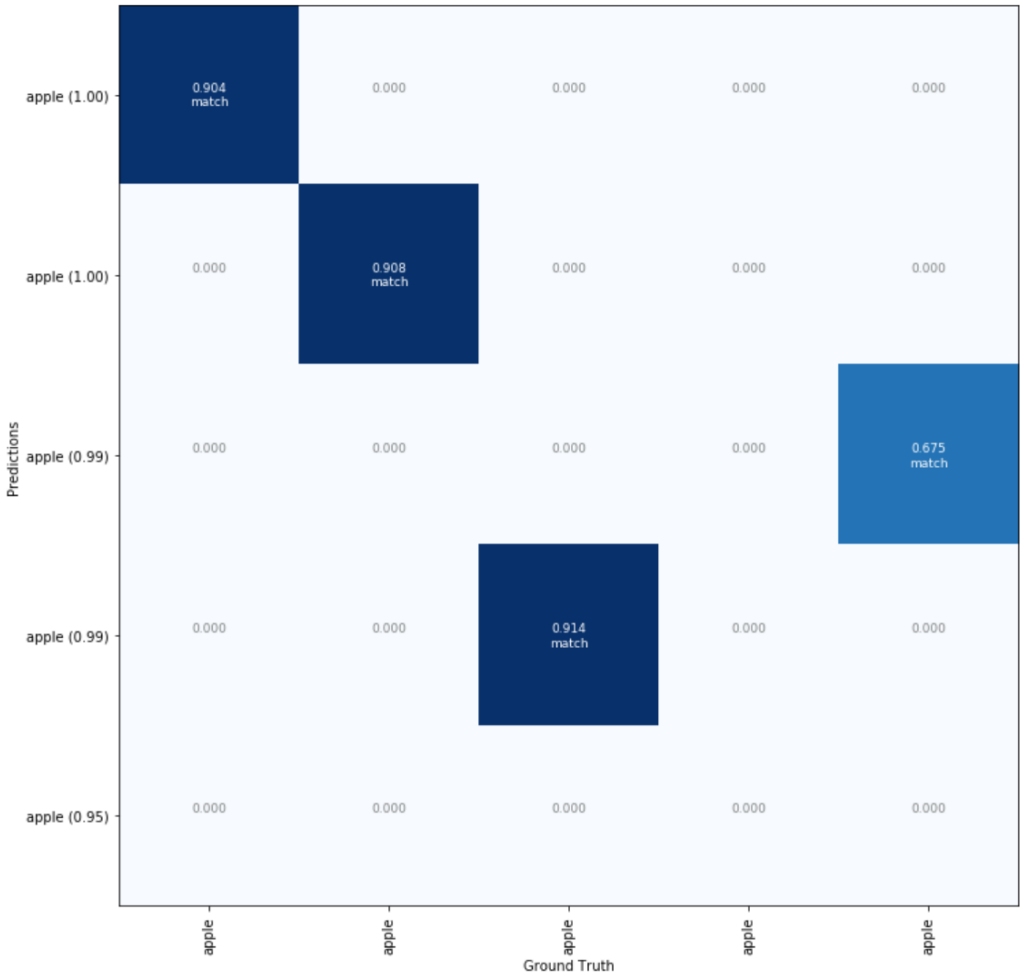


FIGURE 6.25: Grid of ground truth objects and their predictions for ResNet-50-FPN Heads Training from COCO pre-trained model on Test Data Set.

```
Prediction time: 57.03900623321533s. Average 0.509276841367994s/image
Total time: 87.01134967803955s
mAP @ IoU=50:  0.6929838244273063
```

FIGURE 6.26: mAP and Inference Time for ResNet-50-FPN Heads Training from COCO pre-trained model on Test Data Set.

```
Epoch 15/15
100/100 [==============================] - 3638s 36s/step - loss: 0.7523 - rpn_class_loss: 0.0198 - rpn_bbo
x_loss: 0.1441 - mrcnn_class_loss: 0.1322 - mrcnn_bbox_loss: 0.1563 - mrcnn_mask_loss: 0.3000 - val_loss: 0
.9580 - val_rpn_class_loss: 0.0188 - val_rpn_bbox_loss: 0.2637 - val_mrcnn_class_loss: 0.1177 - val_mrcnn_b
box_loss: 0.2456 - val_mrcnn_mask_loss: 0.3122
Total time:  54586.23294329643
Training time: 54586.23294329643 s. Average: 60.92213498135762 s /image
```

FIGURE 6.27: Losses and Training time for ResNet-50-FPN Heads Training from ImageNet pre-trained model.

6.2.2.1 Test Data Set

Prediction Data There are four prediction instances in the picture, where one is not pre-labeled, other two are missing (Figure 6.28).

FIGURE 6.28: Prediction Data for ResNet-50-FPN Heads Training from ImageNet pre-trained model on Test Data Set.

Precision-Recall Curve The corresponding precision-recall curve is as follows (Figure 6.29). The recall corresponding to the curve turning point is 0.600, which is lower than from COCO pre-trained model(0.800).

Grid of ground truth objects and their predictions Below is the matching score grid (Figure 6.30). Because there is one prediction instance which is not pre-labeled, there is one empty row in the grid. Besides, there are two instances which do not have predictions resulting in two empty columns.

mAP @ IoU=50 on Batch of Images The mAP is around 0.536 less than from COCO pre-trained model (0.692) (Figure 6.31). The average prediction time is around 0.256S per image.

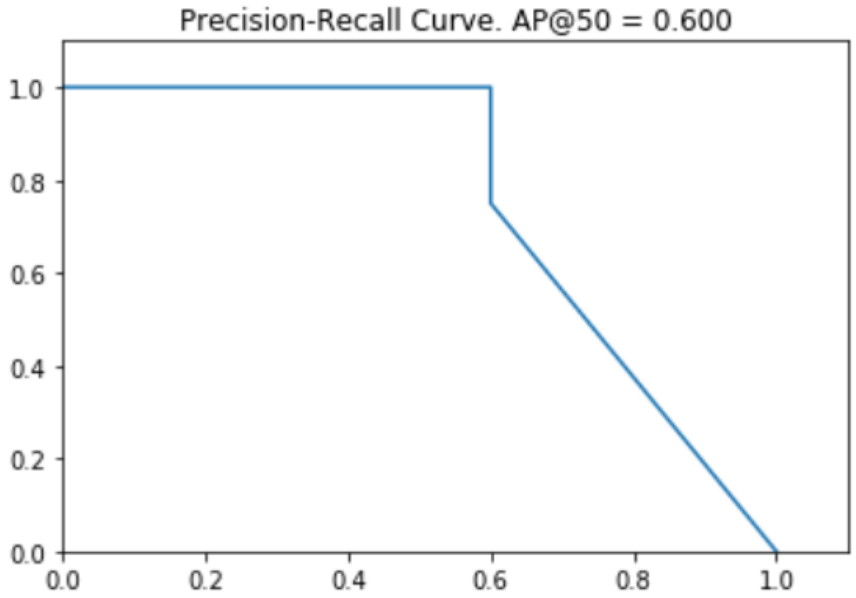


FIGURE 6.29: Precision-Recall Curve for ResNet-50-FPN Heads Training from ImageNet pre-trained model on Test Data Set.

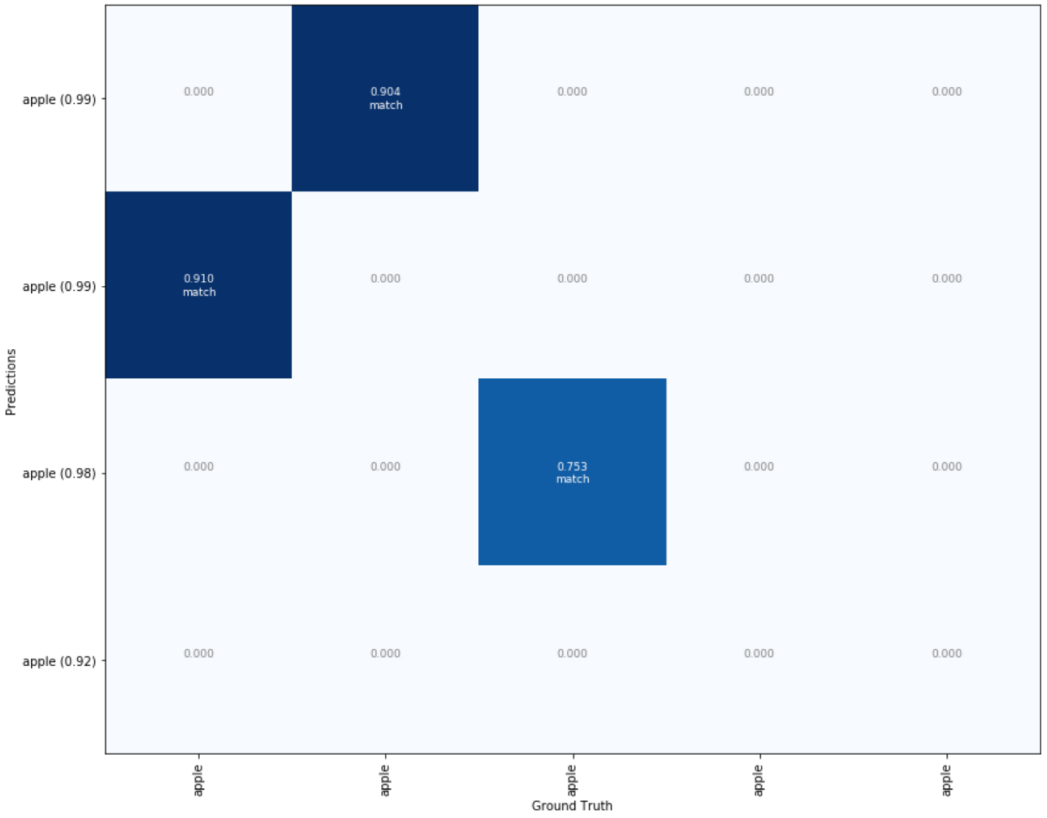


FIGURE 6.30: Grid of ground truth objects and their predictions for ResNet-50-FPN Heads Training from ImageNet pre-trained model on Test Data Set.

```
Prediction time: 28.773500680923462s. Average 0.25690625607967377s/image
Total time: 55.95989489555359s
mAP @ IoU=50:  0.536380917573342
```

FIGURE 6.31: mAP and Inference Time for ResNet-50-FPN Heads Training from ImageNet pre-trained model on Test Data Set.

6.3 ResNet-50-FPN Heads Training for Data Augmentation

This part uses data augmentation for input data to check whether it improves accuracy. It applies the simplest data augmentation method: flips images right or left half of the time. The main purpose is to validate the data augmentation technology. Hence, we just apply the heads training on the test data set from COCO pre-trained mode here.

6.3.1 From COCO pre-trained model

Losses and Training Time The following is the corresponding losses and training time (Figure 6.32). It takes around average 60.27S per image.

```
Epoch 15/15
100/100 [==============================] - 3591s 36s/step - loss: 0.7793 - rpn_class_loss: 0.02
00 - rpn_bbox_loss: 0.1741 - mrcnn_class_loss: 0.1560 - mrcnn_bbox_loss: 0.1533 - mrcnn_mask_lo
ss: 0.2758 - val_loss: 1.1366 - val_rpn_class_loss: 0.0478 - val_rpn_bbox_loss: 0.3084 - val_mr
cnn_class_loss: 0.1808 - val_mrcnn_bbox_loss: 0.2816 - val_mrcnn_mask_loss: 0.3179
Total time:  54008.67997646332
Training time: 54008.67997646332 s. Average: 60.27754461658852 s /image
```

FIGURE 6.32: Losses and Training Time for ResNet-50-FPN Heads Training from COCO pre-trained model

6.3.1.1 Test Data Set

Prediction Data The prediction data is as follows (Figure 6.33). There are seven predicted instances in the picture, two more instances are not pre-labeled.

Precision-Recall Curve From the curve, we can see the recall corresponding to the curve turning point is 0.967 higher than ResNet-50-FPN from the original data set (0.800) (Figure 6.34)

Grid of ground truth objects and their predictions Below is the matching score grid (Figure 6.35). Because there are two prediction instances which are not pre-labeled, there are two empty rows in the grid.

```
image                    shape: (1024, 1024, 3)       min:    0.00000  max:  255.00000  uint8
molded_images            shape: (1, 1024, 1024, 3)    min: -123.70000  max:  151.10000  float64
image_metas              shape: (1, 14)               min:    0.00000  max: 1024.00000  int64
anchors                  shape: (1, 261888, 4)        min:   -0.35390  max:    1.29134  float32
```

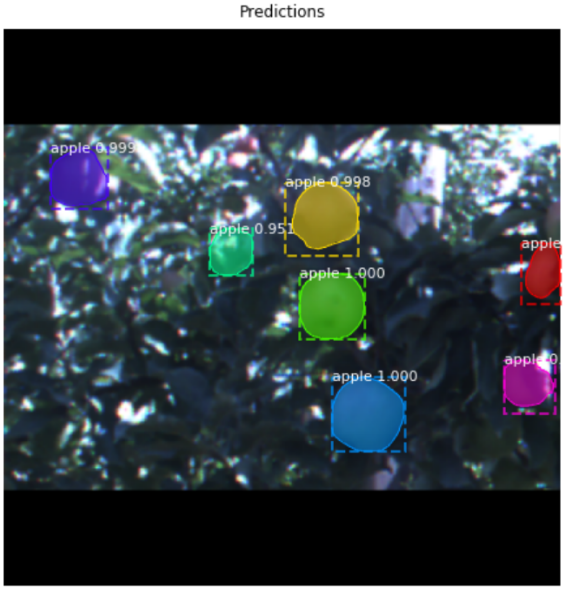


FIGURE 6.33: Prediction Data for ResNet-50-FPN Heads Training from COCO pre-trained model on Test Data Set.

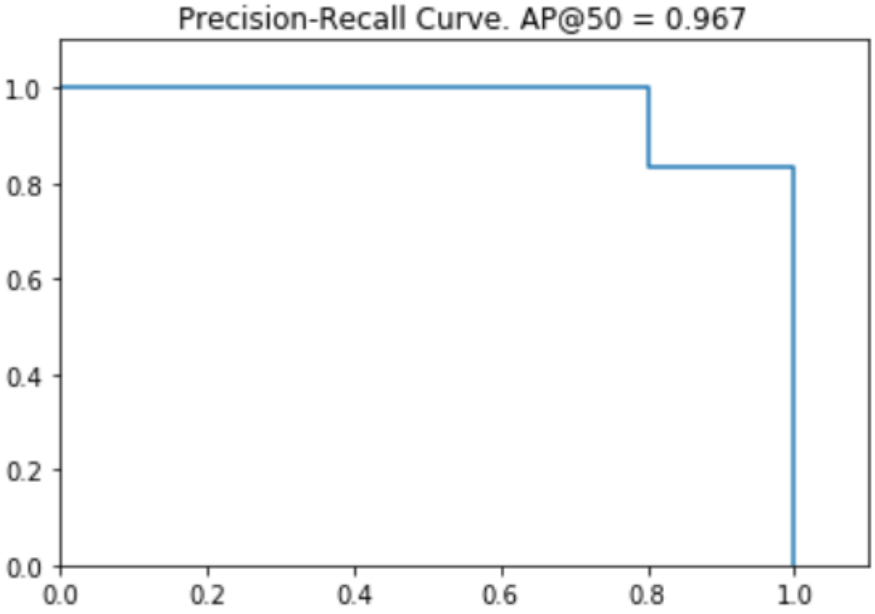


FIGURE 6.34: Precision-Recall Curve for ResNet-50-FPN Heads Training from COCO pre-trained model on Test Data Set.

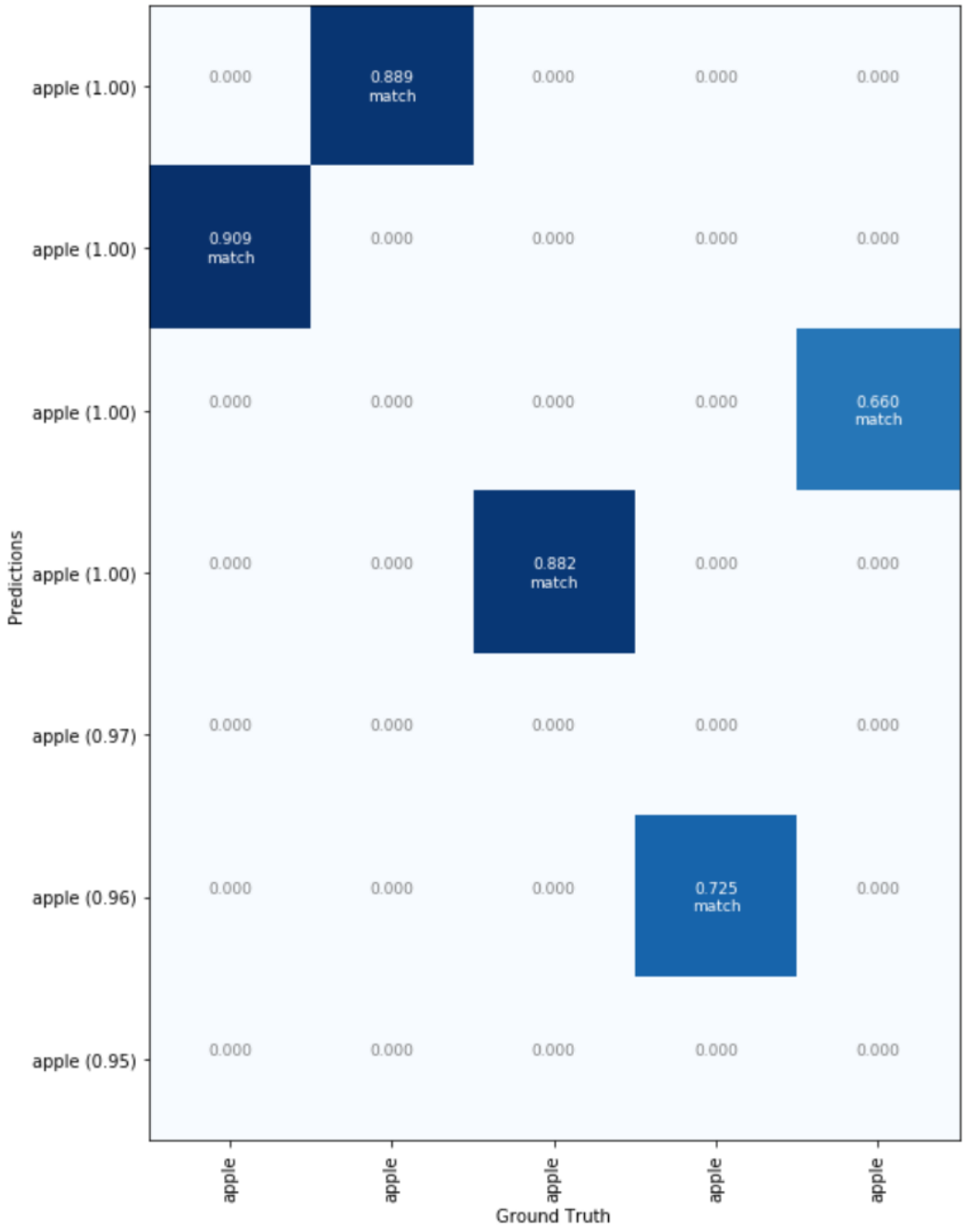


FIGURE 6.35: Grid of ground truth objects and their predictions for ResNet-50-FPN Heads Training from COCO pre-trained model on Test Data Set.

mAP @ IoU=50 on Batch of Images Below shows the corresponding mAP and inference time (Figure 6.36). The average inference time is around 0.260S per image.

```
Prediction time: 29.162858963012695s. Average 0.26038266931261334s/image
Total time: 57.207791328430176s
mAP @ IoU=50:  0.7476583049824688
```

FIGURE 6.36: mAP and Inference Time for ResNet-50-FPN Heads Training from COCO pre-trained model on Test Data Set.

6.4 Lessons Learned

Based on the previous results, this part lists the summary in tables for comparison.

	ResNet-101-FPN		**ResNet-50-FPN Augmentation**
	head	**all**	**head**
Training Time	14.29hours, 57.43S/image	15.57hours, 62.57S/image	15hours, 60.27S/image

TABLE 6.1: The Training Time for Each Model-1

	ResNet-50-FPN		
	COCO		**ImageNet**
	head	**all**	**head**
Training Time	15.19hours, 61.07S/image	11.59hours, 46.6S/image	15.16hours, 60.92S/image

TABLE 6.2: The Training Time for Each Model-2

	Validation		**Test**	
	head	**all**	**head**	**all**
Precision	0.681	0.737	0.681	0.737
Inference Time	0.630S/image	0.628S/image	0.630S/image	0.628S/image

TABLE 6.3: The Summary for ResNet-101-FPN

From the results, it can be shown the blow several points:

1. The performance for all layers fine-tuning is better than heads training. The reason is obvious because heads just contain the last several layers, all layers contain all of ResNet. Fine-tuning all layers based on the trained heads. For ResNet-101-FPN, the accuracy of all layers fine-tuning is more than 5% than heads training but takes more about 15 hours. In a practical problem, it needs to find a trade-off. For simple data set, heads training can meet requirements.

	Validation		**Test**	
	From COCO Pre-trained Model			
	head	**all**	**head**	**all**
Precision	0.692	0.758	0.692	0.758
Inference Time	0.505S/image	0.509S/image	0.509S/image	0.511S/image
	From ImageNet Pre-trained Model			
	head	**all**	**head**	**all**
Precision	0.536	0.716	0.536	0.716
Inference Time	0.250S/image	0.259S/image	0.256S/image	0.259S/image

Table 6.4: The Summary for ResNet-50-FPN

	Validation		**Test**	
	head	**all**	**head**	**all**
Precision	0.747	0.750	0.747	0.750
Inference Time	0.254S/image	0.258S/image	0.260S/image	0.257S/image

Table 6.5: The Summary for ResNet-50-FPN Augmentation

2. The performance for ResNet-50 backbone from COCO pre-trained model is better than from ImageNet, yet the inference time is almost twice than from ImageNet. Because compared to ImageNet data set, the character of COCO data set is the annotation file marks the exact coordinates of each segmentation and bbox.

3. The performance for data augmentation is better than the original data set because data augmentation provides more samples for training.

4. The performance for ResNet-50 backbone is better than ResNet-101 backbone. Theoretically, the more layers for backbone, the more features can be extracted. Because the data set is not complicated, the fewer layers backbone performs better.

Furthermore, we can get more generalised conclusions:

1. For simple data sets, complicated backbones do not make a big difference in improving accuracy.

2. The training process of Mask R-CNN is quite time-consuming.

3. When computing mAP, first compute matches:

   (a) 1-D array for each ground-truth box which has the index of the matched predicted box.

   (b) 1-D array for each predicted box which has the index of the matched ground truth box. [1]

   Thus, if some instances were not pre-labeled, but the result predicted the instances (regarded as True Negative, as shown in before pictures), as the above second

condition, the length of array will reduce, and therefore the final accuracy would be affected to decrease.

Chapter 7

Conclusion and Future Work

This project applies the state-of-the-art technology Mask R-CNN model to implement fruit recognition, localisation and end-to-end pixel-level segmentation. The mAP of a batch of images on the test data set can achieve around 75%. First, we introduce some related concepts in computer vision area. Because Mask R-CNN includes classification, object detection and segmentation. For studying the Mask R-CNN model, we analyse, discuss and evaluate the technologies from classification, object detection to segmentation and the connection between them.

7.1 Constraints

One of the requirements is the accuracy of the algorithm should achieve the accepted confidence interval. Due to the several reasons below, the accuracy cannot arrive at a high level:

- Hardware Limitation: Because of CNNs Algorithms are time-consuming, which needs strong hardware-GPU supports. More training epochs, higher accuracy. For time limitation, the project just uses 15 epochs for heads training, 10 epochs for all layers fine-tuning. In a practical project, more epochs can be used.
- Data Set Size and Quality: The data set size impacts the training effect directly so that impact the final precision. The fruit data set has around 1000 images, 896 images are used for training, the data set size is not big enough. Likewise, the data set quality affects the training effect as well.
- Labelling Data: As analysed before, data that actually exists without labelling will reduce the final precision.
- Parameters Optimization: Some parameters(e.g.learning rate, weight decay, and momentum etc.) can be optimized in further.

7.2 Future Work

Mask R-CNN model is the state-of-the-art technology in instance segmentation area. To study and explore the model more deeply, we can compare and evaluate Mask R-CNN with the previous models and analyse key replacement techniques to previous models.

- Compare and evaluate Mask R-CNN model with Multi-task Network Cascades model and Fully Convolutional Instance-aware Semantic model.
- Replace ResNet backbone to other CNNs architecture(e.g. ResNeXt, DenseNet), evaluate their performance.
- Replace RoI Align back to RoI Pooling(used in Faster R-CNN) to validate how RoI Align can improve the mask accuracy.
- For mask branch prediction, replace FCN to MLP to validate how FCN can improve the mask precision.

Bibliography

[1] W. Abdulla, "Mask r-cnn for object detection and instance segmentation on keras and tensorflow." https://github.com/matterport/Mask_RCNN, 2017.

[2] N. Zeng, "An introduction to evaluation metrics for object detection," December 2018.

[3] W. Abdulla, "Splash of color: Instance segmentation with mask r-cnn and tensorflow," March 2018.

[4] DFann, "Object detection-rcnn sppnet fast-rcnn faster-rcnn," September 2017.

[5] J. Jordan, "Evaluating image segmentation models," May 2018.

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2015.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[9] M. Rizwan, "Residual networks," October 2018.

[10] F.-F. L. Justin Johnson, Andrej Karpathy, "Spatial localization and detection," October 2015.

[11] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580–587, 2014.

[12] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, pp. 154–171, 2013.

[13] G. Erdoğan, "Faster r-cnn," October 2016.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, pp. 1904–1916, 2014.

[15] R. Girshick, "Fast r-cnn," October 2015.

[16] R. B. Girshick, "Fast r-cnn," *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1440–1448, 2015.

[17] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 1137–1149, 2015.

[18] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, 2015.

[19] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask r-cnn," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[20] J. Jordan, "An overview of semantic image segmentation.," May 2018.

[21] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 936–944, 2017.

[22] S. Bargoti and J. Underwood, "Deep fruit detection in orchards," *arXiv preprint arXiv:1610.03677*, 2016.

[23] A. Gandhi, "Data augmentation — how to use deep learning when you have limited data," December 2018.

[24] S. University, "Cs231n convolutional neural networks for visual recognition-transfer learning," March 2018.

Automatically Split Training, Validation, and Test Data Set and Annotations Files

```
import pandas as pd
import numpy as np
import os
import sys
import shutil

dataset_dir="D:\\Google Drive\\Mask_RCNN\\datasets\\apples"
img_dir=os.path.join(dataset_dir,"images")  ####### cannot use "\\images"
ano_dir=os.path.join(dataset_dir,"annotations")

train_dir=os.path.join(dataset_dir,"train")
test_dir=os.path.join(dataset_dir,"test")
val_dir=os.path.join(dataset_dir,"val")

sets_dir=os.path.join(dataset_dir,"sets")
train_path=os.path.join(sets_dir,"train.txt")
val_path=os.path.join(sets_dir,"val.txt")
test_path=os.path.join(sets_dir,"test.txt")

img_names = [name for name in os.listdir(img_dir)
if os.path.isfile(os.path.join(img_dir, name))]
print(img_names[:10])

ano_names = [name for name in os.listdir(ano_dir)
if os.path.isfile(os.path.join(ano_dir, name))]
print(ano_names[:10])

def read_sets_copy(file_path,flag):
   if flag=="train":
      target_dir=train_dir
   elif flag=="val":
```

```
            target_dir=val_dir
        else:
            target_dir=test_dir
        print(target_dir)

        with open(file_path,'rt',encoding='utf-8') as f:
            for line in f:
                file_name1=line.strip()
                if file_name1+".png" in img_names:
                    img_src=os.path.join(img_dir,file_name1+".png")
                    shutil.copy(img_src,target_dir)
                    ano_src=os.path.join(ano_dir,file_name1+".csv")
                    shutil.copy(ano_src,target_dir)

read_sets_copy(train_path,"train")
read_sets_copy(val_path,"val")
read_sets_copy(test_path,"test")
```

Train on the Fruit Data Set

Reference to [1].

```
import os
import sys
import json
import datetime
import time
import numpy as np
import csv
import skimage.draw

# Root directory of the project
# ROOT_DIR = os.path.abspath("../../")
ROOT_DIR = os.path.abspath("./")  # currnet directory

# Import Mask RCNN
sys.path.append(ROOT_DIR)  # To find local version of the library
from mrcnn.config import Config
from mrcnn import model as modellib, utils

# Path to trained weights file
COCO_WEIGHTS_PATH = os.path.join(ROOT_DIR, "mask_rcnn_coco.h5")

# Directory to save logs and model checkpoints, if not provided
# through the command line argument --logs
DEFAULT_LOGS_DIR = os.path.join(ROOT_DIR, "logs")

############################################################
#  Configurations
############################################################

class FruitConfig(Config):
    """Configuration for training on the fruit dataset.
    Derives from the base Config class and overrides some values.
    """
    # Give the configuration a recognizable name
```

```
    NAME = "apple"

    # We use a GPU with 12GB memory, which can fit two images.
    # Adjust down if you use a smaller GPU.
    IMAGES_PER_GPU = 2

    # Number of classes (including background)
    NUM_CLASSES = 1 + 1  # Background + fruit

    # Number of training steps per epoch
    STEPS_PER_EPOCH = 100

    # Skip detections with < 90% confidence
    DETECTION_MIN_CONFIDENCE = 0.9


############################################################
#  Dataset
############################################################

class FruitDataset(utils.Dataset):
    def load_fruit(self, dataset_dir, subset):
        """Load a subset of the fruit dataset.
        dataset_dir: Root directory of the dataset.
        subset: Subset to load: train,val or test
        """
        # Add classes. We have only one class to add.
        self.add_class("apple", 1, "apple")

        # Train, validation or test dataset?
        assert subset in ["train", "val", "test"]
        dataset_dir = os.path.join(dataset_dir, subset)
        print(dataset_dir)

        # Load annotations
        img_file_list = []
        ano_file_list = []
        for name in os.listdir(dataset_dir):
            if name.endswith(".png"):
            img_file_list.append(name)
            elif name.endswith(".csv"):
            ano_file_list.append(name)

        for name in img_file_list:
            index_ = name.index("_")
            pre_name = name[:index_ + 3].strip()
             ano_index=ano_file_list.index(pre_name)
```

```
        if (pre_name + ".csv") not in ano_file_list:
        continue

        r_list = []
        c_list = []
        radius_list = []
        ano_path = os.path.join(dataset_dir, pre_name + ".csv")
        with open(ano_path) as f:  ####
           f_csv = csv.reader(f)
           headers = next(f_csv)
           for row in f_csv:
              r_list.append(row[2])  ##### subscript, row is y, column is x
              c_list.append(row[1])
              radius_list.append(row[3])


        image_path = os.path.join(dataset_dir, name)
        image = skimage.io.imread(image_path)
        height, width = image.shape[:2]
        #print("height:",height,"width:",width)   # 202*308

        self.add_image(
        "apple",
        image_id=name,  # use file name as a unique image id
        path=image_path,
        width=width, height=height,
        r=r_list,
        c=c_list,
        radius=radius_list)  ######

  def load_mask(self, image_id):
     """Generate instance masks for an image.
     Returns:
     masks: A bool array of shape [height, width, instance count] with
     one mask per instance.
     class_ids: a 1D array of class IDs of the instance masks.
     """
     # If not a fruit dataset image, delegate to parent class.
     image_info = self.image_info[image_id]
     if image_info["source"] != "apple":
        return super(self.__class__, self).load_mask(image_id)

     # Convert circles to a bitmap mask of shape
     # [height, width, instance_count]
     info = self.image_info[image_id]
     mask = np.zeros([info["height"], info["width"], len(info["r"])],
```

```
            dtype=np.uint8)
            for index in range(len(info["r"])):
                # Get indexes of pixels inside the circle and set them to 1
                rr, cc = skimage.draw.circle(float(info["r"][index]),
        float(info["c"][index]), float(info["radius"][index]),
                mask.shape)
                mask[rr, cc, index] = 1

            # Return mask, and array of class IDs of each instance. Since we have
            # one class ID only, we return an array of 1s
            return mask.astype(np.bool), np.ones([mask.shape[-1]], dtype=np.int32)

        def image_reference(self, image_id):
            """Return the path of the image."""
            info = self.image_info[image_id]
            if info["source"] == "apple":
                return info["path"]
            else:
                super(self.__class__, self).image_reference(image_id)


    def train(model,epoch,layer):
        """Train the model."""
        # Training dataset.
        dataset_train = FruitDataset()
        dataset_train.load_fruit(args.dataset, "train")
        dataset_train.prepare()

        # Validation dataset
        dataset_val = FruitDataset()
        dataset_val.load_fruit(args.dataset, "val")
        dataset_val.prepare()

        t_start = time.time()
        image_ids = dataset_train.image_ids
        print("len(image_ids):",len(image_ids))
        if layer == 'heads':
            # Training - Stage 1
            print("Training network heads")
            model.train(dataset_train, dataset_val,
            learning_rate=config.LEARNING_RATE,
            epochs=int(epoch),
            layers='heads')  #########

        elif layer == '3+':
            # Training - Stage 3
            # Finetune layers from ResNet stage 2 and up
```

```
        print("Fine tune Resnet stage 3 and up")
        model.train(dataset_train, dataset_val,
        learning_rate=config.LEARNING_RATE,
        epochs=int(epoch)+10,
        layers='3+')

    elif layer == "4+":
        # Training - Stage 4
        # Finetune layers from ResNet stage 4 and up
        print("Fine tune Resnet stage 4 and up")
        model.train(dataset_train, dataset_val,
        learning_rate=config.LEARNING_RATE,
        epochs=int(epoch)+10*2,
        layers='4+')

    elif layer == "all":
        # Fine tune all layers
        print("Fine tune all layers")
        model.train(dataset_train, dataset_val,
        learning_rate=config.LEARNING_RATE / 10,
        epochs=int(epoch),
        layers='all')

    t_train = time.time() - t_start
    print("Total time: ", t_train)
    print("Training time: {} s. Average: {} s /image".format(
    t_train, t_train / len(image_ids)))

############################################################
#  Training
############################################################

if __name__ == '__main__':
    import argparse

    # Parse command line arguments
    parser = argparse.ArgumentParser(description='Train Mask R-CNN to detect
     fruit.')
    parser.add_argument("command",
                        metavar="<command>",
                        help="train")
    parser.add_argument('--dataset', required=False,
                        metavar="/dataset/apples/",
                        help='Directory of the Fruit dataset')
    parser.add_argument('--weights', required=True,
                        metavar="/weights.h5",
                        help="Path to weights .h5 file or 'coco'")
```

```
parser.add_argument('--logs', required=False,
                    default=DEFAULT_LOGS_DIR,
                    metavar="/logs/",
                    help='Logs and checkpoints directory (default=logs/)')
parser.add_argument('--epoch', required=True,
                    help="Epoch of trainning")
parser.add_argument('--layers', required=False, default=
 'heads',choices=['heads','3+','4+','all'],
help="Layer for trainning")
args = parser.parse_args()

# Validate arguments
if args.command == "train":
assert args.dataset, "Argument --dataset is required for training"

print("Weights: ", args.weights)
print("Dataset: ", args.dataset)
print("Logs: ", args.logs)
print("Epoch: ", args.epoch)
print("Layers: ", args.layers)

# Configurations
if args.command == "train":
   config = FruitConfig()
else:
   class InferenceConfig(FruitConfig):
      # Set batch size to 1 since we'll be running inference on
      # one image at a time. Batch size = GPU_COUNT * IMAGES_PER_GPU
      GPU_COUNT = 1
      IMAGES_PER_GPU = 1
      config = InferenceConfig()
config.display()

# Create model
if args.command == "train":
   model = modellib.MaskRCNN(mode="training", config=config,
   model_dir=args.logs)
else:
   model = modellib.MaskRCNN(mode="inference", config=config,
   model_dir=args.logs)

# Select weights file to load
if args.weights.lower() == "coco":
   weights_path = COCO_WEIGHTS_PATH
   # Download weights file
   if not os.path.exists(weights_path):
      utils.download_trained_weights(weights_path)
```

```
elif args.weights.lower() == "last":
    # Find last trained weights
    weights_path = model.find_last()
elif args.weights.lower() == "imagenet":
    # Start from ImageNet trained weights
    weights_path = model.get_imagenet_weights()
else:
    weights_path = args.weights

# Load weights
print("Loading weights ", weights_path)
if args.weights.lower() == "coco":
    # Exclude the last layers because they require a matching
    # number of classes
    model.load_weights(weights_path, by_name=True, exclude=[
    "mrcnn_class_logits", "mrcnn_bbox_fc",
    "mrcnn_bbox", "mrcnn_mask"])
else:
    model.load_weights(weights_path, by_name=True)


# Train or evaluate
if args.command == "train":
    train(model,args.epoch,args.layers)
else:
    print("'{}' is not recognized. "
    "Use 'train' ".format(args.command))
```

Inference on the Fruit Data Set

Reference to [1].

```
import os
import sys
import random
import math
import re
import time
import numpy as np
import tensorflow as tf
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Root directory of the project
# ROOT_DIR = os.path.abspath("../../")
ROOT_DIR = os.path.abspath("./")  # currnet directory
print(ROOT_DIR)

# Import Mask RCNN
sys.path.append(ROOT_DIR)  # To find local version of the library
from mrcnn.config import Config
from mrcnn import utils
from mrcnn import visualize
from mrcnn.visualize import display_images
import mrcnn.model as modellib
from mrcnn.model import log

%matplotlib inline

# directory of dataset
DATASET_DIR = os.path.join(ROOT_DIR,"datasets/apples")
print(DATASET_DIR)

# Directory to save logs and trained model
MODEL_DIR = os.path.join(ROOT_DIR, "logs")
```

```
class FruitConfig(Config):
    """Configuration for training on the fruit dataset.
    Derives from the base Config class and overrides some values.
    """
    # Give the configuration a recognizable name
    NAME = "apple"

    IMAGES_PER_GPU = 1    ####

    # Number of classes (including background)
    NUM_CLASSES = 1 + 1  # Background + fruit

    # Number of training steps per epoch
    STEPS_PER_EPOCH = 100

    # Skip detections with < 90% confidence
    DETECTION_MIN_CONFIDENCE = 0.9

config = FruitConfig()
config.display()

### Override the training configurations with a few changes for inferencing.
class InferenceConfig(config.__class__):
    # Run detection on one image at a time
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1

inference_config = InferenceConfig()
inference_config.display()

DEVICE = "/gpu:0"  # /cpu:0 or /gpu:0

def get_ax(rows=1, cols=1, size=8):
    """Return a Matplotlib Axes array to be used in
    all visualizations in the notebook. Provide a
    central point to control graph sizes.

    Change the default size attribute to control the size
    of rendered images
    """
    _, ax = plt.subplots(rows, cols, figsize=(size*cols, size*rows))
    return ax

### Load validation dataset
dataset_val = FruitDataset()
dataset_val.load_fruit(DATASET_DIR, "val")
```

```
# Must call before using the dataset
dataset_val.prepare()

print("Images: {}\nClasses: {}".format(len(dataset_val.image_ids),
    dataset_val.class_names))

### Load test dataset
dataset_test = FruitDataset()
dataset_test.load_fruit(DATASET_DIR, "test")

# Must call before using the dataset
dataset_test.prepare()

print("Images: {}\nClasses: {}".format(len(dataset_test.image_ids),
    dataset_test.class_names))

### Head Training Model, Create model object in inference mode.
with tf.device(DEVICE):
   model_head = modellib.MaskRCNN(mode="inference", model_dir=MODEL_DIR,
    config=inference_config)

# Get path to saved weights
# Either set a specific path or find last trained weights
# model_path = model.find_last()
model_head_path = os.path.join(ROOT_DIR+"/logs",
    "apple_101_head_epoch15_step100.h5")

# Load trained weights
print("Loading weights from ", model_head_path)
model_head.load_weights(model_head_path, by_name=True)

### Create the whole model object in inference mode.
with tf.device(DEVICE):
   model = modellib.MaskRCNN(mode="inference", model_dir=MODEL_DIR,
    config=inference_config)

# Get path to saved weights
# Either set a specific path or find last trained weights
# model_path = model.find_last()
model_path = os.path.join(ROOT_DIR+"/logs", "apple_101_all_epoch25_step100.h5")

# Load trained weights
print("Loading weights from ", model_path)
model.load_weights(model_path, by_name=True)

image_id = random.choice(dataset_val.image_ids)
```

```
image_id = 83  # 20130320T005805.533702.Cam6_13.png
original_image, image_meta, gt_class_id, gt_bbox, gt_mask =\
modellib.load_image_gt(dataset_val, inference_config, image_id,
    use_mini_mask=False)
info = dataset_val.image_info[image_id]
print("image ID: {}.{} ({}) {}".format(info["source"], info["id"], image_id,
dataset_val.image_reference(image_id)))

'''
Run Detection on Validation Data Set
'''
# ground truth
visualize.display_instances(original_image, gt_bbox, gt_mask, gt_class_id,
dataset_val.class_names, figsize=(8, 8))   ######## dataset_train.class_names

log("original_image", original_image)
log("image_meta", image_meta)
log("gt_class_id", gt_class_id)
log("gt_bbox", gt_bbox)
log("gt_mask", gt_mask)

##### Head training model, Run object detection
results_head = model_head.detect([original_image], verbose=1)

# Display results
ax = get_ax(1)
r_head = results_head[0]
visualize.display_instances(original_image, r_head['rois'], r_head['masks'],
    r_head['class_ids'], dataset_val.class_names, r_head['scores'], ax=ax,
    title="Predictions")

# The whole model, Run object detection
results = model.detect([original_image], verbose=1)

# Display results
ax = get_ax(1)
r = results[0]
visualize.display_instances(original_image, r['rois'], r['masks'],
    r['class_ids'], dataset_val.class_names, r['scores'], ax=ax,
    title="Predictions")

'''
Evaluation on Validation Data Set
'''
#### Head Training Model, Draw precision-recall curve
AP_head, precisions_head, recalls_head, overlaps_head =
    utils.compute_ap(gt_bbox, gt_class_id, gt_mask, r_head['rois'],
```

```
    r_head['class_ids'], r_head['scores'], r_head['masks'])
visualize.plot_precision_recall(AP_head, precisions_head, recalls_head)

# The whole model, Draw precision-recall curve
AP, precisions, recalls, overlaps = utils.compute_ap(gt_bbox, gt_class_id,
    gt_mask, r['rois'], r['class_ids'], r['scores'], r['masks'])
visualize.plot_precision_recall(AP, precisions, recalls)

##### Head Training Model, Grid of ground truth objects and their predictions
visualize.plot_overlaps(gt_class_id, r_head['class_ids'], r_head['scores'],
    overlaps_head, dataset_val.class_names)

#The Whole model, Grid of ground truth objects and their predictions
visualize.plot_overlaps(gt_class_id, r['class_ids'], r['scores'], overlaps,
    dataset_val.class_names)

'''
Compute mAP @IoU=50 on Batch of Images
'''
def compute_batch_ap(image_ids,model,r,dataset):
   t_prediction = 0
   t_start = time.time()
   APs = []

   for image_id in image_ids:
      # Load image
      image, image_meta, gt_class_id, gt_bbox, gt_mask =
    modellib.load_image_gt(dataset_test, inference_config, image_id,
    use_mini_mask=False)

      # Run object detection
      t = time.time()
      results = model.detect([image], verbose=0)
      t_prediction += (time.time() - t)

      r = results[0]
      # Compute AP
      AP, precisions, recalls, overlaps = utils.compute_ap(gt_bbox,
    gt_class_id, gt_mask,
      r['rois'], r['class_ids'], r['scores'], r['masks'])

      flag = False
      for data in recalls:
      if np.isnan(data):
         print(image_id,dataset.image_reference(image_id))
         flag= True
         continue
```

```
        if flag == True:
        continue

        APs.append(AP)

    print("Prediction time: {}s. Average {}s/image".format(t_prediction,
     t_prediction / len(image_ids)))
    print("Total time: {}s ".format(time.time() - t_start))

    return APs

APs = compute_batch_ap(dataset_val.image_ids,model_head,r_head,dataset_val)
print("mAP @ IoU=50: ", np.mean(APs))

APs = compute_batch_ap(dataset_val.image_ids,model,r,dataset_val)
print("mAP @ IoU=50: ", np.mean(APs))
```

Table of Contents for the design archive

- Mask_RCNN: The source code project folder.
- split_train_val_text.ipynb: The code file for Appendix 3
- Readme.md: The description file for how to run the source code project.