A

PROJECT-I

(LC-AI-342G)

On

Barcode Detection

DPGITM

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

(ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

Sixth Semester

SUBMITTED BY

Ansari Ehteesham Aqeel

College Roll No: 20AI07

University Roll No: 4058971

Department of Computer Science and Engineering

DPG Institute of Technology and Management, Gurgaon

(Affiliated to Maharishi Dayanand University, Rohtak)

2020-2024

CANDIDATE'S DECLARATION/ CERTIFICATE

I hereby certify that the work which is being presented in the report entitled “Barcode Detection” by “Ansari Ehteesham Aqeel” in partial fulfilment of requirements for the award of degree of B.Tech. (CSE-AIML) submitted to Department of Computer Science and Engineering, DPG Institute of Technology and Management, Gurgaon (Affiliated with Maharishi Dayanand University, Rohtak) is an authentic record of my own work under the supervision of “Parveen Kumari”.

(Signature of the Student)

“ANSARI EHTEESHAM AQEEL”

College Roll No: 20AI07

University Roll No: 4058971

SUPERVISOR'S CERTIFICATE

I hereby certify that the presented work report entitled “Barcode Detection” by “Ansari Ehteesham Aqeel” in partial fulfilment of requirements for the award of degree of B.Tech. (CSE-AIML) submitted to Department of Computer Science and Engineering, DPG Institute of Technology and Management, Gurgaon (Affiliated with Maharishi Dayanand University, Rohtak) is an authentic record of his own work under my supervision at Institute.

(Signature of the Supervisor)

“Mrs. Parveen Kumari”

“Head of CSE Department”

ABSTRACT

Barcode detection is an important aspect of various industries including retail, logistics, and manufacturing. In recent years, object detection algorithms like Faster R-CNN have shown great potential in improving the accuracy of barcode detection. This project aims to develop a barcode detection system using Faster R-CNN algorithm. The proposed system takes an input image, applies a pre-trained Faster R-CNN model, and detects the barcode regions with high accuracy. The implementation of the system is done using the OpenCV library in Python. The system is evaluated on a custom dataset of images containing barcodes and compared with other state-of-the-art object detection algorithms. The results indicate that the proposed system outperforms other algorithms in terms of accuracy and speed. The proposed system can be deployed in various industries for efficient and accurate barcode detection, which can improve the overall productivity and reduce errors.

ACKNOWLEDGMENT

I am highly grateful to Mrs. Parveen Kumari, Head, Department of Computer Science and Engineering, DPG Institute of Technology and Management, Gurgaon, for providing this opportunity.

The author would like to express a deep sense of gratitude and thank to **Dr. R.C. Kuhad**, Director, DPGITM, without whose permission, wise counsel and able guidance, it would have not been possible to pursue my training in this manner.

The help rendered by Mrs Parveen Kumari, Supervisor (Head of CSE Department) for experimentation is greatly acknowledged.

Finally, I express my indebtedness to all who have directly or indirectly contributed to the successful completion of my industrial training.

Ansari Ehteesham Aqeel

INDEX

Introduction

Problem Definition

The problem addressed in this project is the need for a fast and accurate barcode detection system that can be deployed in a retail environment to improve inventory management and reduce checkout times. Traditional barcode scanners require manual scanning of each item, which can be time-consuming and prone to errors. This project aims to develop a barcode detection system using Faster R-CNN that can automatically detect and read barcodes on products as they pass through a checkout counter. The system should be able to handle varying lighting conditions and barcode orientations, while maintaining a high level of accuracy and minimizing false detections. The scope of the project includes data collection and annotation, model development and training, and integration with existing point-of-sale systems. The success of the project will be evaluated based on the accuracy and speed of the system, as well as its impact on reducing checkout times and improving inventory management.

Objectives and Scope

Objectives --

- Develop a Faster R-CNN model for barcode detection that achieves high accuracy and efficiency.
- Train the model on a diverse set of barcode images to improve its performance on various barcode types and lighting conditions.
- Evaluate the model's performance on a test set of images and compare it with existing barcode detection methods.
- Develop an application that can take input images and output the detected barcodes with their corresponding data.

Scope --

- The project will focus on detecting 1D barcodes such as UPC, EAN, and Code39.
- The model will be trained and evaluated on a dataset of several hundred barcode images.
- The project will not involve decoding the barcode data or performing any actions based on the detected barcodes (e.g., inventory management or order processing).

Benefits

Here are some potential benefits of a barcode detection project using Faster R-CNN:

- **Automation of inventory management and supply chain operations --** The accurate and efficient detection of barcodes can enable businesses to automate their inventory and supply chain operations, reducing manual labour costs and improving operational efficiency.
- **Improved quality control --** Barcode detection can help identify and track products throughout the manufacturing process, allowing for better quality control and defect detection.
- **Enhanced customer experience --** Faster and more accurate barcode detection can help improve the speed and accuracy of barcode scanning in retail environments, reducing wait times and improving customer satisfaction.
- **Scalability --** A robust barcode detection system can be easily scaled across multiple locations and applications, allowing businesses to streamline their operations and reduce costs over time.

Limitation

Here are some potential limitations of a barcode detection project using Faster R-CNN:

- **Limited to 1D barcodes --** The project is focused on detecting 1D barcodes such as UPC, EAN, and Code39, which may limit its applicability in certain industries that require 2D barcode detection.
- **Limited lighting conditions --** The performance of the model may be affected by varying lighting conditions, particularly low light or glare, which may reduce the accuracy of the barcode detection.
- **Limited camera angles --** The project may be limited to detecting barcodes from specific camera angles, as changes in orientation or perspective can make barcode detection more challenging.
- **Training data limitations --** The performance of the model heavily relies on the quality and quantity of the training data. Limited or biased training data may result in lower accuracy or incomplete detection of barcodes in real-world scenarios.
- **Decoding limitation --** The project only focuses on detecting the barcode location, not decoding the barcode data. Decoding the barcode data may require additional software or systems, which could be a limitation for some use cases.

Feasibility Assessment

A feasibility assessment for a barcode detection project using Faster R-CNN would typically involve an evaluation of the technical, economic, operational, and scheduling aspects of the project. Here are some key considerations for each of these areas --

Technical feasibility --

- The availability and suitability of the necessary hardware and software tools required for the project, including high-quality cameras, GPUs, and appropriate software libraries such as TensorFlow or PyTorch.
- The ability to collect and label a sufficient amount of high-quality training data to train the Faster R-CNN model, and the availability of suitable annotation tools.
- The ability to integrate the barcode detection system with existing business systems, such as inventory management or point-of-sale systems.

Economic feasibility --

- The costs associated with hardware, software, and personnel required to develop and deploy the barcode detection system.
- The potential benefits and ROI of the system, including the cost savings from automation of inventory and supply chain operations, improved quality control, and enhanced customer experience.

Operational feasibility --

- The availability and suitability of personnel with the necessary skills and knowledge to develop and maintain the system.
- The ability to integrate the barcode detection system with existing business processes and systems, and to ensure that the system can be easily scaled as needed.

Scheduling feasibility --

- The estimated timeline for the project, including the time required to collect and label training data, train and test the Faster R-CNN model, and deploy the system.

- The availability of resources, including personnel and hardware, required to complete the project on time.

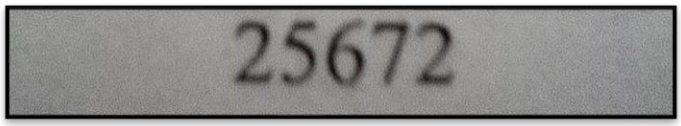
Tool Used

The tools used for a barcode detection project using Faster R-CNN can vary depending on the specific requirements of the project. However, some commonly used tools for object detection and machine learning include:

- **TensorFlow --** An open-source software library for dataflow and differentiable programming across a range of tasks. TensorFlow provides a high-level API for building and training machine learning models, including object detection models.
- **PyTorch --** A popular open-source machine learning library for Python. PyTorch is often used for developing and training deep neural networks, including object detection models.
- **OpenCV --** An open-source computer vision and machine learning software library. OpenCV can be used for a variety of computer vision tasks, including image pre-processing and postprocessing for object detection.
- **LabelImg --** A graphical image annotation tool used to label object bounding boxes in images. LabelImg supports a range of annotation formats, including Pascal VOC and YOLO.
- **NVIDIA GPUs --** Faster R-CNN models require significant computational resources, and GPUs can provide the necessary performance for training and inference. NVIDIA GPUs are often used for deep learning tasks, including object detection.

Dataset

We have compiled a dataset sourced from our college (DPGITM) library by capturing images of barcodes from various books. The resulting dataset reflects our sample selection methodology.



Our dataset comprises barcodes captured under varying environmental conditions. This diversity in the dataset will enhance the robustness of our model, enabling it to capture barcodes even under suboptimal lighting conditions. Our dataset currently consists of 712 images, and we are continuously augmenting it to enhance the precision and generalizability of our model.

To ensure an optimal training outcome, we partitioned the dataset of 712 images into a training set comprising 700 images and a testing set consisting of 12 images.

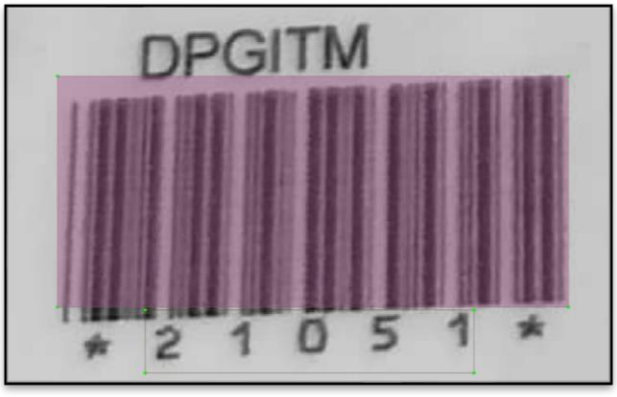
We employed the use of the **labelImg annotation tool** to facilitate the process of annotating the images in a seamless and efficient manner.

Image Annotation

Image annotation is the process of identifying and labelling specific objects or features within an image, typically in order to create a training dataset for machine learning algorithms.

Image annotation is a crucial step in developing computer vision applications, such as object detection, image classification, and semantic segmentation, as it helps to train the algorithm to recognize and understand visual data.

At the stage of annotation, this is the appearance of the image under consideration.



Labelled as '**1d_barcode**'

Labelled as '**bar_number**'

Our image annotation dataset was labelled with two distinct classes. The first class, '1d_barcode,' represents the barcode itself, while the 'bar_number' class pertains to the unique identifying number associated with the barcode.

The annotated images data were saved using the Pascal VOC format, which employs an XML file structure. This format enables the storage of various annotation data, including the image directory, image size, class name ('1d_barcode' or 'bar_number'), and bounding box coordinates (xmin, ymin, xmax, ymax).

We utilized a Python script, titled '**XML file to Text file.py**,' to convert the XML data of all annotated images into the Faster R-CNN dataset format. This format comprises a text file that includes the image path, bounding box coordinates (xmin, ymin, xmax, ymax), and class name for each annotated object, structured as follows:

Image_path,xmin,ymin,xmax,ymax,class_name

Shown below is an illustration of the format described above, which depicts how the annotated image data is represented in the text format for utilization in the Faster R-CNN model.

```
D:/Barcode/DataSet/test/test_img_10.jpg,34,179,988,810,1d_barcode
D:/Barcode/DataSet/test/test_img_10.jpg,320,840,679,944,bar_number
```

After completing the aforementioned steps of annotating and converting the images into the Faster R-CNN dataset format, our training and testing datasets were successfully prepared for use in the Faster R-CNN model.

Summary

Source	DPGITM library
Total Number of Images	712
Total Number of Train Images	700
Total Number of Test Images	12
Annotation Tool	LabelImg
Annotation Format	Pascal VOC
Converted Format	Faster R CNN Format

VGG Network

About VGG Network

The VGG network is a convolutional neural network (CNN) architecture that was proposed by Karen Simonyan and Andrew Zisserman of the University of Oxford in 2014. The architecture is named after the Visual Geometry Group (VGG) at the University of Oxford, where the network was developed.

The VGG network is characterized by its deep structure, consisting of multiple layers of convolutional and pooling layers, followed by fully connected layers at the end. It is a highly configurable network architecture, with variations ranging from 11 layers to 19 layers, denoted as VGG11, VGG13, VGG16, and VGG19. These variations differ in the number of convolutional and fully connected layers and the size of the filters used in the convolutional layers. This model implements VGG16, a convolutional neural network with 16 convolutional layers.

The key feature of the VGG network is its use of small filter sizes, typically 3x3, in all convolutional layers. This leads to a significant increase in the number of network parameters, but also improves the network's ability to learn complex features in the input image.

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual computer vision competition that features two main tasks. The first task, **Object Localization**, requires participating teams to detect objects within an image belonging to one of 200 classes. The second task, **Image Classification**, involves classifying images into one of 1000 pre-defined categories.

The VGG network achieved notable success in the **Object Localization** and **Image Classification** tasks at the 2014 ILSVRC challenge, securing 1st and 2nd place in these categories, respectively. The model demonstrated exceptional performance, achieving a top-5 test accuracy of 92.7% on the **ImageNet dataset**, which comprises approximately 14 million images distributed across 1000 classes.

Architecture

The VGG network is constructed with very small convolutional filters. The VGG-16 consists of 13 convolutional layers and three fully connected layers. Let’s take a brief look at the architecture of VGG:

- **Input –** The VGG Network takes in an image input size of 224×224.

- **Convolutional Layers –** The VGG16 model consists of 13 convolutional layers, each with a filter size of 3 x 3 and a fixed stride of 1 pixel. The first two convolutional layers have 64 filters each, followed by two more convolutional layers with 128 filters, then three convolutional layers with 256 filters, three more with 512 filters, and finally another three with 512 filters. The padding size is set to 1 pixel in order to maintain the same spatial dimensions as the input feature map.
  After each group of convolutional layers, there is a max pooling layer with a 2 x 2-pixel window and a stride of 2, which reduces the spatial dimensions of the feature maps.

- **Hidden Layers –** All the hidden layers in the VGG network use ReLU (Rectified Linear Unit).

- **Fully Connected Layers –** Following the 13 convolutional layers, there are three fully connected layers. The first two fully connected layers have 4096 channels each, and the third performs two-way classification, with the classes being 1d_barcode and 2d_barcode, resulting in two channels (one for each class). The final layer is a SoftMax layer, which produces the probability distribution over the two classes.

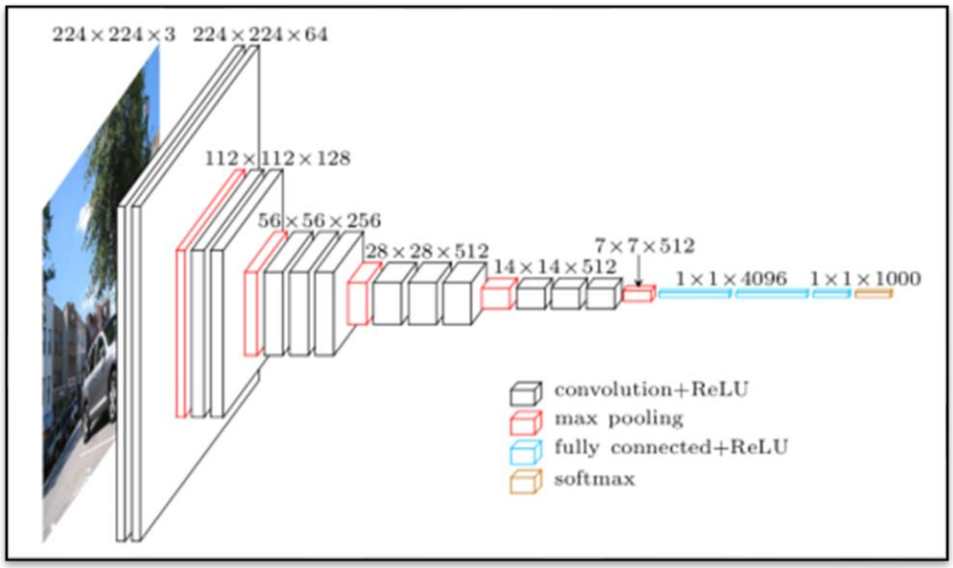


Fig 1.1 Architecture of VGG-16 Model

A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 1.1 – VGG Network Configurations

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Table 1.2 – Number of Parameters (in millions)

Faster R CNN

Faster R-CNN (Region-based Convolutional Neural Network) is an object detection model that was introduced in 2015 as an improvement over the original RCNN and its variants. Faster R-CNN was proposed by Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun from Microsoft Research in their 2015 paper titled **"Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks".**

Faster R-CNN (Region-based Convolutional Neural Network) is an object detection model that is based on the idea of generating region proposals and then classifying them into object categories or background. It consists of two main components: a **Region Proposal Network (RPN)** and a **Fast R-CNN** network for object detection.

Before proceeding any further, it is imperative to grasp a few fundamental terms in the field, such as **region proposals**, **objectness score**, **sliding window**, **feature map**, **object proposal**, **backbone convolutional network**, **bounding box**, and the like. These terminologies are crucial in comprehending the underlying principles of object detection and are widely used in the literature. It is recommended to familiarize oneself with these concepts to gain a better understanding of the domain and effectively communicate with other professionals in the field.

Region Proposals or Object Proposals

The objective of region proposal algorithms is to identify and propose candidate regions in an image that are likely to contain objects of interest. And that region is bounded by a Bounding Box. This is a crucial step in object detection pipelines since it helps to reduce the computational cost of object detection by limiting the search space to a subset of the image. There are several methods for generating region proposals, including **Selective Search**, **Edge Boxes**, and **Region Proposal Networks (RPNs).**

Bounding Box

A bounding box, also known as a bounding rectangle or a bounding box annotation, is a rectangular box that is drawn around an object of interest in an image or video.

The bounding box defines the extent of the object in the image and is typically represented by four coordinates that specify the location of the box's top-left corner and its width and height. The coordinates can be specified in different formats, such as pixel values or normalized values between 0 and 1.

Bounding boxes are commonly used in computer vision tasks such as object detection, object tracking, and image segmentation. They allow the computer to localize and identify objects in an image or video and provide important information for subsequent analysis and processing.

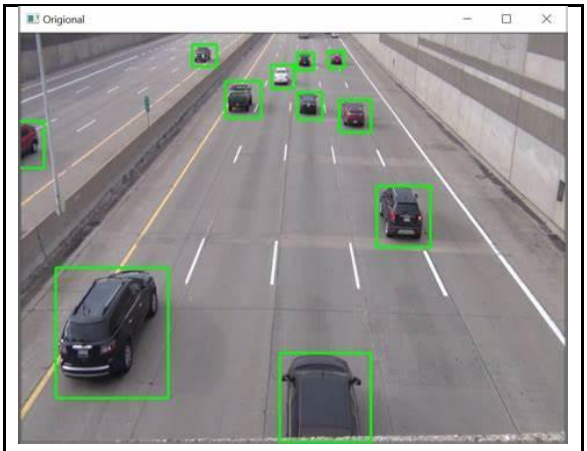


Fig 2.1 Bounding Box (Green Boxes)

Objectness Score

Objectness score is a measure used in object detection algorithms to determine the likelihood that a given region of an image contains an object of interest.

In the object detection pipeline, after generating candidate regions using a region proposal algorithm, the objectness score is used to rank the regions based on their likelihood of containing an object. The score is typically a single value between 0 and 1, where higher values indicate a greater likelihood of containing an object.

Objectness scores are calculated based on various features of the candidate region, such as its colour, texture, and shape. These features are typically extracted using Convolutional Neural Networks (CNNs) and fed into a classifier that predicts the objectness score for the region.

The use of objectness scores helps to reduce the computational cost of object detection by limiting the number of regions that need to be processed in subsequent stages of the pipeline. Regions with low objectness scores are discarded, allowing the algorithm to focus on the most promising regions that are likely to contain objects of interest.

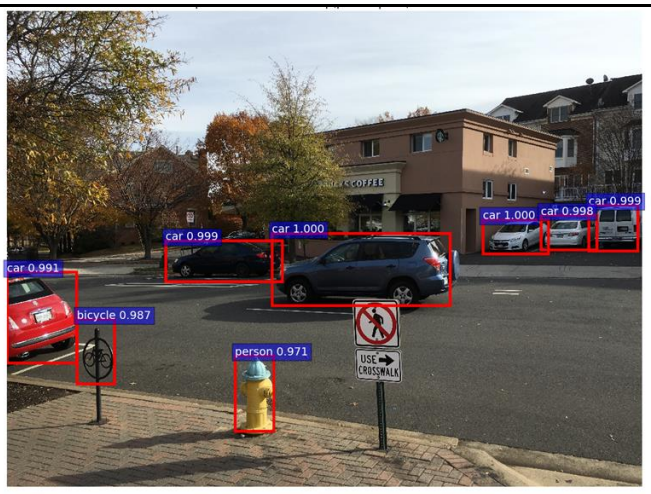


Fig 2.2 Bounding Box with Object Score

Feature Map

In Convolutional Neural Networks (CNNs), a feature map refers to the output of a layer of neurons that apply a set of learned filters to an input image or activation from a previous layer.

Each filter is designed to detect a specific pattern or feature in the input, such as edges, corners, or textures. The output of each filter is a two-dimensional array of numbers, known as a **Convolutional Feature Map**. The set of feature maps generated by applying multiple filters to the same input form the output of the layer and are collectively referred to as the feature map.

The number of filters used in a layer of a CNN determines the number of feature maps generated as output from that layer. Each filter generates a single feature map, so an input image with six filters will have six feature maps.

In subsequent layers of the CNN, the feature maps from the previous layer are used as input, and another set of filters is applied to generate new feature maps that capture more complex patterns and relationships between features. This process of feature extraction and transformation continues through multiple layers until the network produces a final output, such as a classification or segmentation result.

Feature maps play a critical role in the success of CNNs, as they allow the network to learn and represent complex features of the input data that are essential for accurate classification and detection tasks.

The formula for calculating the size of a feature map in a convolutional neural network (CNN) is:

$$\text{Output size} = \left[\frac{I_S - F_S + 2 \times P}{S}\right] + 1$$

where:

I_S -- Input size is the size of the input image or feature map.

F_S -- Filter size is the size of the convolutional filter.

P -- Padding is the number of pixels added to the input image.

S -- Stride is the number of pixels by which the filter is moved during convolution.

Fig 2.3 Feature Map (Right Image) extraction from Left Image

Sliding Window or Spatial Window

In computer vision, sliding window is a technique used to scan an image or a video frame with a small rectangular window of fixed size, usually smaller than the input image. The window is moved across the input image with a certain stride, and at each position, a classifier or a detector is applied to determine if an object or a feature is present inside the window.

Sliding window is often used for object detection, where a classifier is trained to recognize objects of interest, and then applied to different locations and scales in an image to detect instances of the object. The sliding window technique allows the classifier to be applied to all possible locations in the image, and can be used to generate a set of candidates bounding boxes that might contain the object.

Sliding window can also be used for other computer vision tasks such as image segmentation, where a window is used to extract local features or patches from an image for further processing.

Backbone Convolutional Network

A backbone convolutional network is a deep neural network that serves as the core or "backbone" of a larger machine learning model, such as an object detection or image segmentation model. The backbone network is typically a convolutional neural network (CNN) that has been pre-trained on a large-scale dataset, such as ImageNet, to learn generic features that can be used for a variety of computer vision tasks.

The role of the backbone network is to extract a set of high-level feature maps from an input image that can be used by subsequent layers or modules in the model to perform specific tasks, such as object detection or segmentation. The backbone network usually consists of multiple layers of convolutional, pooling, and activation functions, followed by one or more fully connected layers that map the extracted features to a final output.

Some popular examples of backbone networks used in computer vision models include **VGGNet**, **ResNet**, **Inception**, and **EfficientNet**.

Now, we will delve into the workings of Faster R-CNN and the various components utilized for object detection, including how the network identifies the location of objects and generates bounding boxes for them.

Faster R-CNN (Region-based Convolutional Neural Network) is an object detection model that is based on the idea of generating region proposals and then classifying them into object categories or background. It consists of two main components: a **Region Proposal Network (RPN)** and a **Fast R-CNN Network** for object detection.

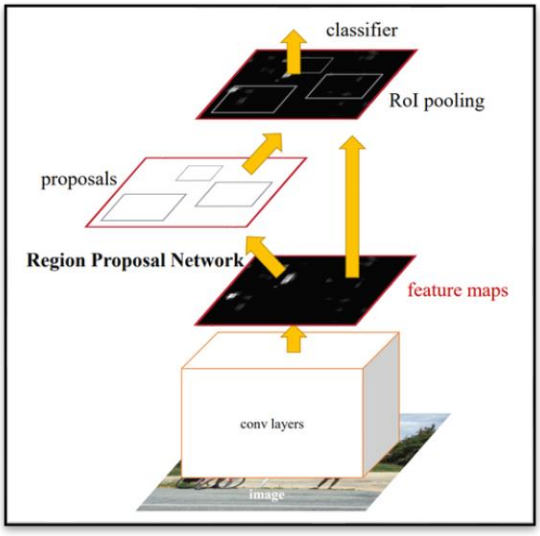


Fig 2.4 Faster R CNN

In Faster R-CNN, the entire system is a single, unified network for object detection, which means that all the components of the system, including the Region Proposal Network (RPN) and the Fast R-CNN detector, are trained end-to-end as part of a single network.

Traditionally, object detection systems were composed of multiple modules, such as object proposal, feature extraction, classification, and localization, that were trained and optimized separately. However, this approach had several limitations, including the difficulty of sharing features across different modules, the need for manual design of the modules, and the lack of flexibility in adapting to different object detection tasks.

In contrast, Faster R-CNN uses a unified network architecture that allows the system to learn all the necessary modules automatically and jointly, end-to-end. This approach enables the system to share features across different components, resulting in more accurate and efficient object detection.

Specifically, in Faster R-CNN, the RPN generates region proposals based on the convolutional feature maps extracted from the input image. These region proposals are then fed into the Fast R-CNN detector, which performs classification and bounding box regression on each proposal to determine the presence and location of objects.

Regional Proposals Network (RPN)

RPN (Region Proposal Network) is a key component of the Faster R-CNN object detection model that generates region proposals or candidate object bounding boxes in an image. The RPN takes a convolutional feature map as input and outputs a set of rectangular proposals along with an objectness score that indicates the likelihood of the proposal containing an object of interest.

The Faster R-CNN model utilizes VGG-16 as its backbone convolutional network. However, for object detection, only the feature maps extracted from the 13th convolutional layer of VGG-16 are used.

To generate region proposals, we slide a small network over the convolutional feature map output by the last shared convolutional layer. This small network takes as input an n × n spatial window of the input convolutional feature map. Each sliding window is mapped to a lower-dimensional feature. This feature is fed into two sibling fully connected layers—a **box-regression layer (reg)** and a **box-classification layer (cls)**.

Note that because the mini-network operates in a sliding-window fashion, the fully-connected layers are shared across all spatial locations. This architecture is naturally implemented with an n × n convolutional layer followed by two sibling 1 × 1 convolutional layer (for **reg** and **cls**, respectively).

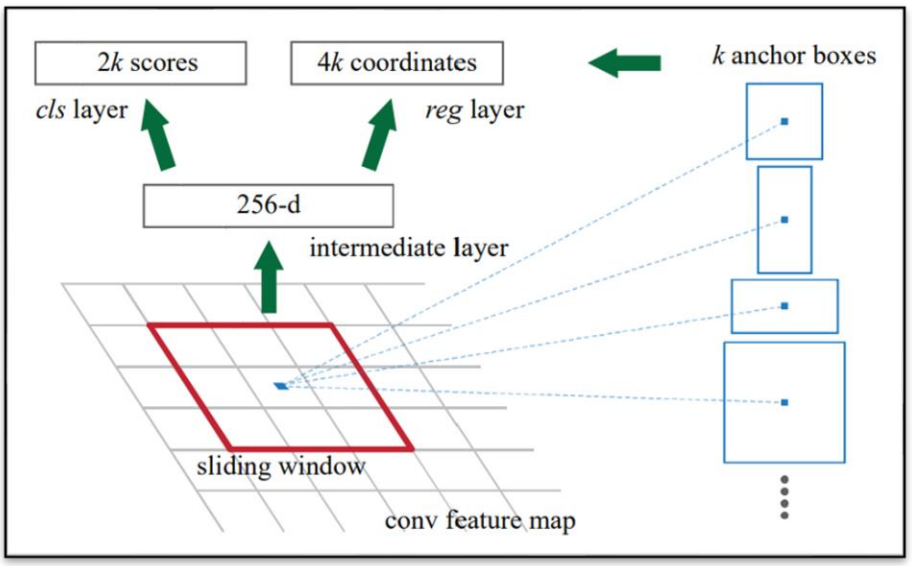


Fig 2.5 Regional Proposal Network (RPN)

Anchors

At each sliding-window location, we simultaneously predict multiple region proposals, where the number of maximum possible proposals for each location is denoted as k. So, the **reg layer** has 4k outputs encoding the coordinates of k boxes, and the **cls layer** outputs 2k scores that estimate probability of object or not object for each proposal. The k proposals are parameterized relative to k reference boxes, which we call **anchors.**

Anchor boxes are placed at the centre of each sliding window of the feature map, and each anchor box is associated with a specific scale and aspect ratio, as shown in Figure 2.5. By default, there are 3 scales [128, 256, 512] and 3 aspect ratios [1:1, 1:2, 2:1], yielding k = 9 anchors at each sliding position.

Translation Invariant Anchors

An important property of Faster R CNN is that it is translation invariant, both in terms of the anchors and the functions that compute proposals relative to the anchors. But first you have to understand what is Translation Invariant.

Translation Invariance is a property of a Convolutional Neural Network that allows it to recognize patterns in an image regardless of their position within the image. Specifically, it means that the network is able to detect the same feature or object regardless of where it appears in the image, as long as its properties remain the same. This is achieved through the use of convolutional filters, which scan the entire input image in a sliding window manner and can detect the same feature regardless of its location. This property is important for image recognition tasks, such as object detection, where the position of objects in the image may vary. The **Translation-Invariant** property also reduces the model size.

Multi-Scale Object Detection using Anchor Based

One may raise the question of why to employ anchor boxes in object detection when there are alternative methods that utilize diverse scales of an image to generate feature maps.

Multi-scale object detection involves detecting objects of different sizes and aspect ratios in an image. Traditional methods for multi-scale object detection use image or feature pyramids, which involve resizing images at multiple scales and computing feature maps for each scale. However, this approach is time-consuming and computationally expensive.

However, the **Anchor-Based** method used in Faster R-CNN is a more efficient way of detecting objects of different scales and aspect ratios. In this method, a set of anchors is placed at each sliding window position on the feature map. Each anchor is associated with a scale and aspect ratio, which determines the size and shape of the bounding box to be predicted. By using a pyramid of anchors, the model can detect objects of different scales and aspect ratios without having to resize the image or compute feature maps for each scale.

The **Anchor-Based** method used in Faster R-CNN has several advantages over traditional methods for multi-scale object detection. It is more computationally efficient because it only uses a single-scale image and feature maps. It is also more accurate because it uses a pyramid of anchors to detect objects of different scales and aspect ratios. Additionally, the anchor-based method allows for better sharing of features across different scales, which improves the model's performance.

Now that you have gained an understanding of how our Region Proposal Network (RPN) operates, how the anchor placement works on the feature map, and the role of the VGG16 Convolutional Neural Network, we will move on to discussing the Loss Function used for training our RPN model. This is necessary because the RPN comprises two 1 x 1 Convolutional Neural Networks (reg and cls) which generate the output, indicating the presence or absence of an object and the corresponding bounding box.

Intersection Over Union (IoU)

IoU stands for **Intersection over Union**, which is a common evaluation metric used in object detection and segmentation tasks. It measures the overlap between two bounding boxes, A and B, as the ratio of the area of their intersection to the area of their union. The formula for calculating IoU is:

$$IoU = \frac{Area\ of\ Intersection\ of\ two\ Boxes}{Area\ of\ Union\ of\ two\ Boxes}$$

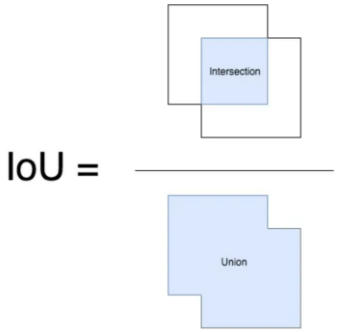


Fig 2.6 Diagrammatic Representation of the formula to calculate IOU

Loss Function

In RPNs, during the training phase, each anchor is assigned a binary class label of either being an object or not being an object. This is done by comparing the anchors with the ground-truth boxes, which are the actual bounding boxes that define the location of the objects in the image.

There are two kinds of anchors that are assigned a positive label:

- The anchor/anchors with the highest Intersection-over-Union (IoU) overlap with a ground-truth box: IoU is a metric used to measure the overlap between two bounding boxes. If an anchor has the highest IoU overlap with a ground-truth box, then it is considered as a positive anchor.
- An anchor that has an IoU overlap higher than 0.7 with any ground-truth box: If an anchor has an IoU overlap higher than 0.7 with any ground-truth box, then it is considered as a positive anchor.

The rest of the anchors, which do not meet the above criteria, are assigned a negative label, indicating that they do not contain an object.

Note that a single ground-truth box may assign positive labels to multiple anchors.

The Fast R-CNN model uses a multi-task loss function to simultaneously train the classification and bounding box regression tasks. The multi-task loss function is defined as the sum of two terms: the classification loss (log loss) and the bounding box regression loss (smooth L1 loss).

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*)$$

$$L_{cls}(p_i, p_i^*) = -\frac{1}{N_{cls}}\sum_i (p_i \times log\,(p_i^*) + (1 - p_i) \times log\,(1 - p_i^*))$$

$$L_{reg}(t_i, t_i^*) = \sum_i R(t_i - t_i^*)$$

$$R(x) = \begin{cases} 0.5x^2, & if\ |x| < 1 \\ |x| - 0.5, & otherwise \end{cases}$$

where:

$\boldsymbol{p_i}$ -- predicted probability of anchor $\boldsymbol{i}$ being an object.

$\boldsymbol{p_i^*}$ -- the ground-truth label $\boldsymbol{p_i^*}$ is 1 if the anchor is positive, and is 0 if the anchor is negative.

$\boldsymbol{t_i}$ -- vector representing the 4 parameterized coordinates (x, y, w, h) of the predicted bounding box.

t_i^* -- it is ground-truth box associated with a positive anchor.

L_{cls} -- log loss over two classes (object vs. not object).

L_{reg} – regression loss which have smooth L1 loss ($R(t_i - t_i^*)$)

The term $p_i^* L_{reg}$ means the regression loss is activated only for positive anchors (p_i^* = 1) and is disabled otherwise (p_i^* = 0). The outputs of the cls and reg layers consist of $\{p_i\}$ and $\{t_i\}$ respectively.

Bounding Box Regression

$$t_x = \frac{(x - x_a)}{w_a}, \quad t_y = \frac{(y - y_a)}{h_a}$$

$$t_w = \log\left(\frac{w}{w_a}\right), \quad t_h = \log\left(\frac{h}{h_a}\right)$$

$$t_x^* = \frac{(x^* - x_a)}{w_a}, \quad t_y^* = \frac{(y^* - y_a)}{h_a}$$

$$t_w^* = log\left(\frac{w^*}{w_a}\right), \quad t_h^* = log\left(\frac{h^*}{h_a}\right)$$

where x, y, w, and h denote the box's centre coordinates and its width and height. Variables x, x_a, and x^* are for the predicted box, anchor box, and ground truth box respectively (likewise for y, w, h). This can be thought of as **bounding-box regression** from an anchor box to a nearby ground-truth box.

The process of updating the parameters of the anchor box to closely match the ground-truth box of an object in an image. This is achieved through **bounding-box regression**, where the predicted offset values (t_x, t_y, t_w, t_h) are used to adjust the position, width, and height of the anchor box to better fit the ground-truth box. The formula mentioned earlier is used to calculate the regression targets, which are then used to update the anchor box during training. By regressing the anchor box towards the ground-truth box, the model is able to accurately detect the object in the image.

Non-Maximum Suppression

Non-Maximum Suppression (NMS) is applied after the second-stage network in Faster R-CNN to filter out redundant detections.

After the second-stage network performs object classification and bounding box regression, it produces a set of candidate object detections with associated confidence scores. These candidate detections may overlap, and we want to remove redundant detections and keep only the most confident ones.

To accomplish this, NMS is applied to the candidate detections. The NMS algorithm works as follows:

- Sort the candidate detections in descending order based on their confidence scores.
- Select the detection with the highest confidence score and add it to the final list of detections.

- Remove all candidate detections that have an overlap with the selected detection above a certain threshold (e.g., IoU (Intersection Over Union) > 0.5).
- Repeat steps 2 and 3 until there are no more candidate detections.

The output of **NMS** is a set of non-redundant, high-confidence detections that are used as the final output of the Faster R-CNN object detection pipeline.

NMS is a common post-processing step in object detection pipelines, and it can be applied to other object detection models as well, not just Faster R-CNN.

Region Of Interest Pooling (RoIs Pooling)

The ROI (Region of Interest) pooling layer is a type of pooling layer commonly used in object detection models, such as Faster R-CNN and Mask R-CNN. The purpose of the ROI pooling layer is to extract fixed-size feature maps from variable-sized regions of interest in a convolutional feature map. This is necessary in object detection because the regions of interest can be of different sizes and aspect ratios, and they need to be transformed into a fixed size in order to be fed into fully connected layers for classification or regression.

Here's an example of how the ROI pooling layer works. Let's say we have an input image of size 600x400 and a convolutional feature map of size 75x50x256. We also have two region proposals, which are represented as bounding boxes in the input image: one bounding box corresponds to a car, and the other corresponds to a pedestrian.

To extract features from these region proposals, we need to apply the ROI pooling layer. The ROI pooling layer takes as input the convolutional feature map and the region proposals, and outputs fixed-size feature maps for each proposal. Here's how the ROI pooling layer works in more detail:

- Divide each region proposal into a fixed number of sub-windows. For example, we might divide each proposal into 7x7 sub-windows.
- For each sub-window, find the maximum value in the corresponding region of the convolutional feature map. This is similar to max pooling, but the region of interest is variable-sized and non-overlapping.
- Collect the maximum values from all sub-windows into a fixed-size feature map for each region proposal. For example, we might output a feature map of size 7x7x256 for each proposal.

The output of the **ROI pooling layer** is a set of fixed-size feature maps, one for each region proposal. These feature maps can then be fed into fully connected layers for classification or regression.

Steps in Faster R-CNN Object Detection Algorithm

The Faster R-CNN algorithm involves several steps, which can be summarized as follows:

- **Input Image --** The input image is the first step of the pipeline. The image is fed into the neural network to detect objects in the image.

- **Convolutional Layers --** The convolutional layers are the first set of layers in the network that extract features from the input image. These layers use a set of learnable filters to scan the input image and generate a feature map that encodes the presence of different visual patterns in the image.
  For example, the first few convolutional layers may detect edges and lines in the image, while deeper layers may detect more complex features like shapes, textures, and object parts.

- **Region Proposal Network (RPN) --** The RPN is a separate neural network that takes the feature map generated by the convolutional layers as input and generates a set of candidate regions for objects.
  The RPN is a small convolutional network that slides over the feature map and generates a set of object proposals for each location in the feature map. These object proposals are generated by predicting the probability of each region containing an object and the offset between the region and its corresponding ground-truth bounding box.
  The RPN generates a large number of object proposals, typically several thousand for each input image.

- **Non-Maximum Suppression (NMS) --** The candidate regions generated by the RPN are filtered using non-maximum suppression. NMS is a technique that removes redundant and overlapping regions and retains only the most confident regions.
  The NMS algorithm selects the region with the highest objectness score and removes all regions that have an overlap with the selected region above a certain threshold (e.g., IoU > 0.5). The process is repeated until all regions have been processed, resulting in a set of non-overlapping candidate regions.

- **RoI Pooling Layer --** The RoI pooling layer takes the filtered candidate regions generated by the RPN as input and extracts a fixed-size feature map for each region.
  The RoI pooling layer divides each region into a fixed-size grid of sub-regions and max-pools the activations of each sub-region to produce a fixed-size feature map. The output of the RoI pooling layer is a set of fixed-size feature maps, one for each candidate region.

- **Fully Connected Layers --** The fixed-size feature maps generated by the RoI pooling layer are passed through a set of fully connected layers that perform object classification and bounding box regression.
  The object classification layer predicts the probability of each candidate region containing an object of a particular class (e.g., car, person, dog). The bounding box regression layer predicts the offset between each candidate region and its corresponding ground-truth bounding box.

- **NMS for Final Detection --** The final set of object detections is generated by applying non-maximum suppression to the output of the object classification and bounding box regression layers.
  The NMS algorithm selects the detection with the highest objectness score and removes all detections that have an overlap with the selected detection above a certain threshold (e.g., IoU > 0.5). The process is repeated until all detections have been processed, resulting in a set of non-overlapping and accurate object detections.

Diagrammatic Working of Faster R-CNN

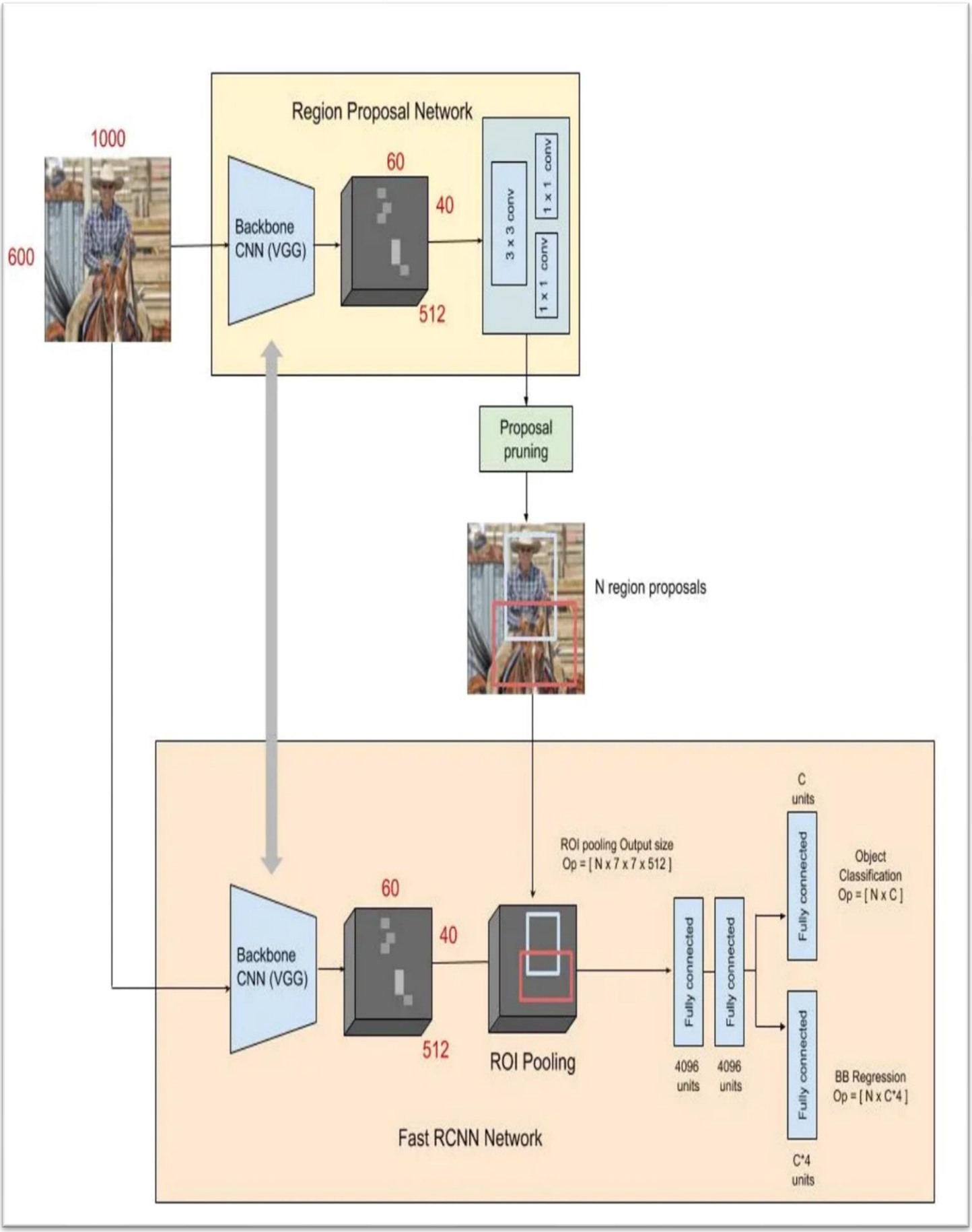


Fig 2.6 The RPN for Regional Proposal and Fast R-CNN as a detector in the Faster R-CNN Detection Pipeline

Code Explanation

During the training process of a dataset, various functions are used to optimize the model's performance. One such function is the loss function, which measures the difference between the predicted output and the actual output. The optimizer function is used to minimize the loss function and adjust the model's parameters.

Other functions used during training include the activation function, which introduces non-linearity into the model and helps it learn complex patterns. The regularization function helps prevent overfitting by adding a penalty to the loss function for complex models. Finally, the data pre-processing function transforms the raw data into a format that the model can understand and learn from. Together, these functions play a crucial role in training a deep learning model.

Get_data()

The function get_data takes an annotation file path as input and returns a list of dictionaries containing image filepaths, dimensions, and bounding boxes for each object instance in the image. The function also returns a dictionary of class counts and a dictionary mapping each class name to a unique index. The function reads each line of the annotation file, parses the information, and stores it in the appropriate data structure. If the class name is new, it is added to the class count dictionary and the class mapping dictionary. If the class name is "bg", it is treated as a special case for background regions. Finally, the function returns the aggregated data and mappings.

- **Input --** This is a Python function called get_data that takes an input path to an annotation file as a parameter.

  Example -- ./annotations/train_annotations.txt

- **Output --** returns a tuple of three elements: a list of dictionaries containing information about images and their bounding boxes, a dictionary containing the count of each class in the dataset, and a dictionary that maps class names to indices.
  Example –

  1. **all_data --** {'filepath': 'path/to/image1.jpg', 'width': 640, 'height': 480, 'bboxes': [{'class': 'car', 'x1': 120, 'y1': 100, 'x2': 300, 'y2': 200}.
  2. **class_mapping --** e.g. {'Car': 0, 'Mobile phone': 1, 'Person': 2}.
  3. **classes_count --** e.g. {'Car': 2383, 'Mobile phone': 1108, 'Person': 3745}.

Get_anchor_gt()

This code defines a generator function that yields preprocessed image data and their corresponding ground-truth anchors for training or testing object detection models. It performs image augmentation, resizing, zero-centering, and standard scaling. The generator yields the processed data in batches.

- **Input --** This code defines a generator function called get_anchor_gt () that yields batches of data to be used in training a model for object detection using the Region Proposal Network (RPN) approach. The function takes in the following arguments --
  1. **all_img_data --** a list of dictionaries, where each dictionary contains information about an image and its annotations.
  2. **C --** a configuration object that contains various parameters for the model, such as the input

image size, anchor box sizes and ratios, and mean pixel values.
3. **img_length_calc_function --** a function that calculates the length of an image along its shorter side after resizing.
4. **mode --** a string indicating whether the function is being used in training mode ('train') or inference mode ('test').

Output -- get_anchor_gt () is a generator function, so it does not return a value directly. Instead, it yields a batch of data for each iteration of the loop. The batch of data that is yielded on each iteration consists of the following items --

1. **x_img --** a NumPy array of shape (1, resized_height, resized_width, 3) containing the preprocessed image data.
2. **[y_rpn_cls, y_rpn_regr] --** a list of two NumPy arrays representing the RPN targets. y_rpn_cls is a binary classification label map of shape (1, feature_map_height, feature_map_width, num_anchors) and y_rpn_regr is a regression target map of shape (1, feature_map_height, feature_map_width, num_anchors * 4).
3. **img_data_aug --** a dictionary containing various information about the original image, such as its filename, annotations, and original width and height.
4. **debug_img --** a copy of the resized image for debugging purposes.
5. **num_pos --** an integer representing the number of positive anchors in the RPN targets.

def augment ()

- **Input --** This is a function that applies data augmentation techniques to an image and its corresponding bounding boxes. It takes in three parameters --
  1. **img_data --** A dictionary containing information about the image, such as the file path, width, height, and bounding boxes of objects in the image.
  2. **config --** An object that contains various configuration settings for the model, such as whether to use horizontal or vertical flips, whether to rotate the image by 90 degrees 0r 180 degrees.
  3. **augment --** A boolean value that determines whether or not to apply data augmentation.
- **Output --** The augment function returns a tuple consisting of the augmented img_data_aug dictionary and the augmented image img. img_data_aug is a dictionary containing the following keys: The img variable contains the augmented image in NumPy array format.
  1. **Width --** the width of the image.
  2. **Height --** the height of the image.

def get_new_img_size ()

The get_new_img_size function resizes an image such that the smaller side is equal to or greater than a specified minimum side length. The function calculates the new dimensions based on the original width and height, and returns the new size as a tuple.

- **Input --** This is a Python function called get_new_img_size that takes in three parameters: width, height, and img_min_side. The purpose of this function is to resize an image with the given width and height while maintaining its aspect ratio, such that the length of its smaller dimension is at least img_min_side pixels. The function returns a tuple of the new width and height of the resized image.

- **Output --** The output of this function is a tuple of two integers that represent the new width and height of the resized image.

  Example –
  If we call the function with width = 400, height = 200, and img_min_side = 300, the function would first check that width <= height is false, so it would calculate the scaling factor based on height.

def calc_rpn()

This code implements the region proposal network (RPN) algorithm for object detection. It calculates anchor boxes and their corresponding regression targets for each location in an image. It also determines whether each anchor box should be labelled as a target or background based on their intersection-over-union (IOU) with ground-truth boxes. The resulting outputs are used as training data for the RPN in a Faster R-CNN object detection model.

- **Input --** The calc_rpn function takes the following arguments --
  1. **C** -- The configuration object that contains various parameters required for the RPN (Region Proposal Network).
  2. **img_data** -- The augmented image data.
  3. **width** -- The original width of the image.
  4. **height** -- The original height of the image.
  5. **resized_width** -- The width of the resized image.
  6. **resized_height** -- The height of the resized image.
  7. **img_length_calc_function** -- A function to calculate the final layer's feature map size according to the input image size.
- **Output --** The calc_rpn function returns two lists -- **y_rpn_cls** and **y_rpn_regr**.
  1. **y_rpn_cls** is a list with shape (num_bboxes, y_is_box_valid + y_rpn_overlap), where num_bboxes is the number of bounding boxes in the input image. For each of the potential anchor boxes, y_is_box_valid indicates whether the anchor is valid (i.e., it for each of the potential anchor boxes, y_is_box_valid indicates whether the anchor is valid (i.e., it has an intersection over union (IoU) overlap with a ground truth bounding box greater than or equal to the threshold defined in the configuration C). y_rpn_overlap indicates whether the anchor is an object or not (i.e., whether it has an IoU overlap with a ground truth bounding box greater than or equal to 0.7).
  2. **y_rpn_regr** is a list with shape (num_bboxes, 4*y_rpn_overlap + y_rpn_regr), where y_rpn_regr contains the bounding box coordinates for each anchor box (i.e., x1, y1, x2, y2), and y_rpn_overlap is the same as in y_rpn_cls. If an anchor is not an object (y_rpn_overlap = 0), then the corresponding bounding box coordinates are set to zero.

def get_img_output_length ()

The function get_img_output_length () contains a nested function get_output_length (). The get_output_length () function takes in a single argument input_length and returns the value of input_length divided by 16, with the quotient rounded down to the nearest integer using the floor division operator //.

The get_img_output_length () function itself returns a tuple containing the result of calling get_output_length () on width and height, respectively. This means that the function is intended to be used to calculate the output size of a convolutional neural network layer based on the size of the input image.

- **Input --** This is a function that takes in the width and height of an image and returns a tuple of the output length for each dimension.
- **Output --** It returns the output length for a given input length (which is width or height in this case).

def union ()

The union function takes in two bounding boxes, au and bu, along with the area of their intersection. It computes the area of the union of the two boxes by adding the areas of both boxes and then subtracting the area of their intersection. The resulting value represents the total area covered by both boxes without double-counting the overlapping region. This is useful in object detection and tracking tasks, where we may want to combine the bounding boxes of multiple objects to get a single bounding box that encompasses all of them.

- **Input --** This is a function takes in two bounding boxes (au and bu) and the area of their intersection (area intersection).
- **Output --** It returns the returns the area of their union.

def intersection ()

The given function, "intersection", calculates the overlapping area between two rectangles (ai and bi). It first finds the x-coordinate and y-coordinate of the top left corner of the overlapping rectangle by taking the maximum value of the respective coordinates of the two rectangles. Then it calculates the width (w) and height (h) of the overlapping rectangle by subtracting the minimum x-coordinate of one rectangle from the maximum x-coordinate of the other and the minimum y-coordinate of one rectangle from the maximum y-coordinate of the other. If either the width or height is negative, the rectangles do not overlap and the function returns 0. Otherwise, it returns the area of the overlapping rectangle, which is simply the product of its width and height.

- **Input --** This is a function takes in two bounding boxes (ai and bi).
- **Output --** It returns the output length for a given input length (which is width or height in this case).

def iou ()

It seems like the function iou (a, b) calculates the intersection over union (IoU) score between two bounding boxes a and b. The IoU score is a metric used in object detection and image segmentation tasks to measure the overlap between two regions. The score ranges from 0 (no overlap) to 1 (perfect overlap). Finally, it returns the IoU score, which is the ratio of intersection area to union area (plus a small value to avoid division by zero errors). The returned value is a floating-point number between 0 and 1, where a higher score indicates better overlap between the two bounding boxes.

- **Input --** This is a function takes two list that is [gta [bbox_num, 0], gta[bbox_num, 2], gta [bbox_num, 1], gta [bbox_num, 3]] and [x1, y1, x2, y2].
- **Output --** It returns the area of their intersection.

VGG Network Model

It is also known as Backbone Convolutional Network.

- **Input –** Image.
- **Output –** Feature Map.

RPN Model

- **Input –**
    1. **base_layers --** VGG in here.
    2. **num_anchors --** 9 in here.
- **Output –**
    1. **[x_class, x_regr, base_layers]**
    2. **x_class --** classification for whether it's an object.
    3. **x_regr --** bboxes regression.
    4. **base_layers –** VGG in here.

Classifier Model

- **Input –**
    1. **base_layers –** VGG.
    2. **input_rois --** `(1,num_rois,4)` list of rois, with ordering (x,y,w,h).
    3. **num_rois --** number of rois to be processed in one time (4 in here).
- **Output –**
    1. **out_class --** classifier layer output.
    2. **out_regr --** regression layer output.

rpn_to_roi()

The purpose of the function is to convert the output of the region proposal network (RPN) to region of interest (ROI) bounding boxes.

- **Input –**
    1. **rpn_layer --** output layer for rpn classification shape (1, feature_map.height, feature_map.width, num_anchors) Might be (1, 18, 25, 18) if resized image is 400 width and 300.
    2. **regr_layer --** output layer for rpn regression shape (1, feature_map.height, feature_map.width, num_anchors) Might be (1, 18, 25, 72) if resized image is 400 width and 300.
    3. **C –** config.
    4. **use_regr --** Wether to use bboxes regression in rpn.

5. **max_boxes --** max bboxes number for non-max-suppression (NMS).
6. **overlap_thresh --** If iou in NMS is larger than this threshold, drop the box.

- **Output –**

  Boxes from non-max-suppression (shape = (300, 4)).
  Boxes -- coordinates for bboxes (on the feature map).

The Faster R-CNN object detection algorithm involves several key functions that enable its successful operation. These functions form the core of the algorithm and are critical for achieving accurate object detection in images.

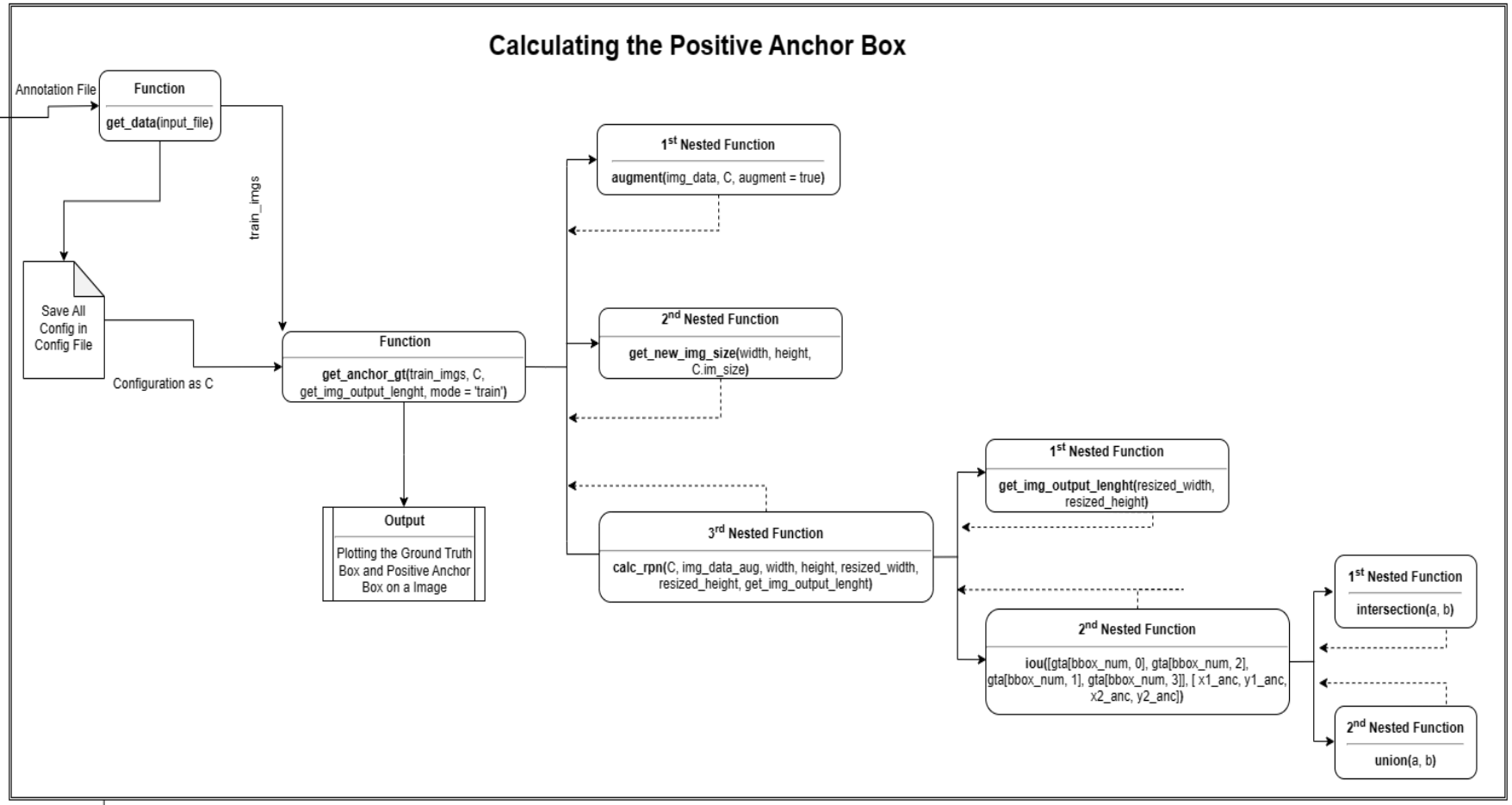
Adding Directory

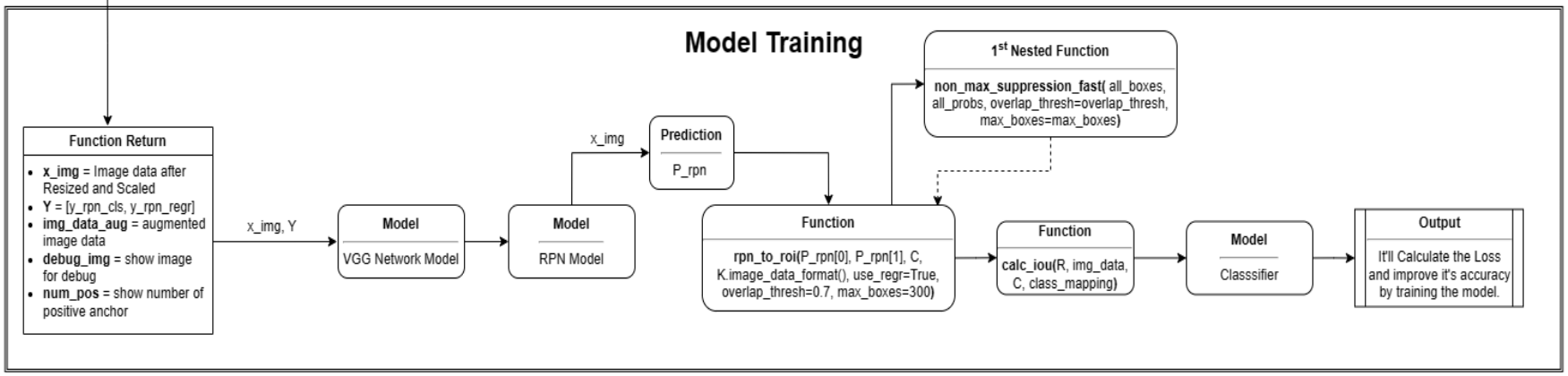
Input Files

- Base Path
- Annotation File
- Custom VGG model

Output Files

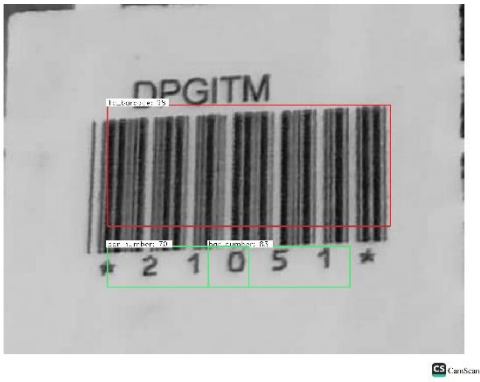
- Records File
- Weight File
- Config File

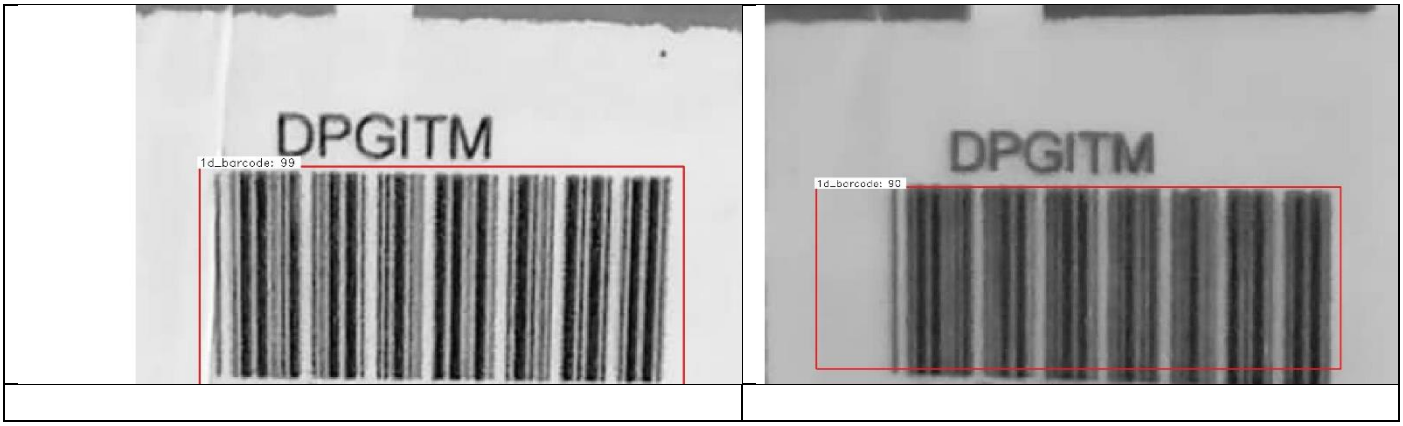




Result

The results of the Faster R-CNN model include images with bounding boxes and corresponding classification labels. These results are generated through the application of the trained model to new input images, allowing for accurate and efficient detection and classification of objects within the images.





Performance of the Model

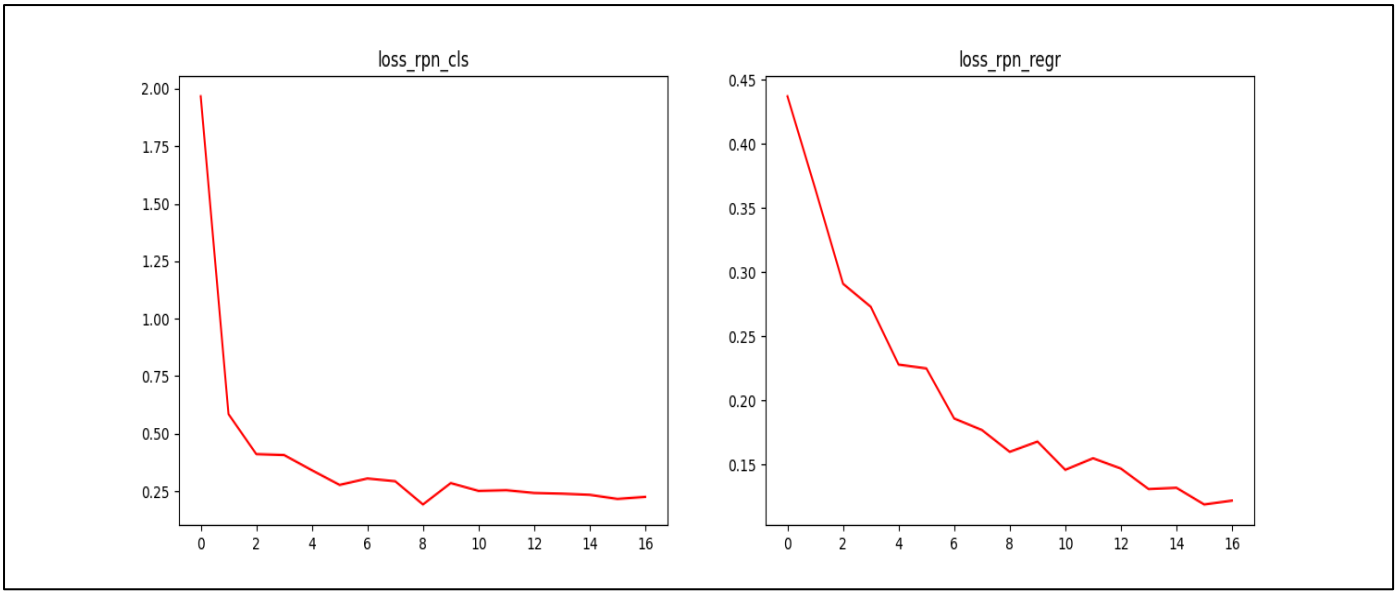


Fig 6.1 Loss of RPN Classification and Regression Layer

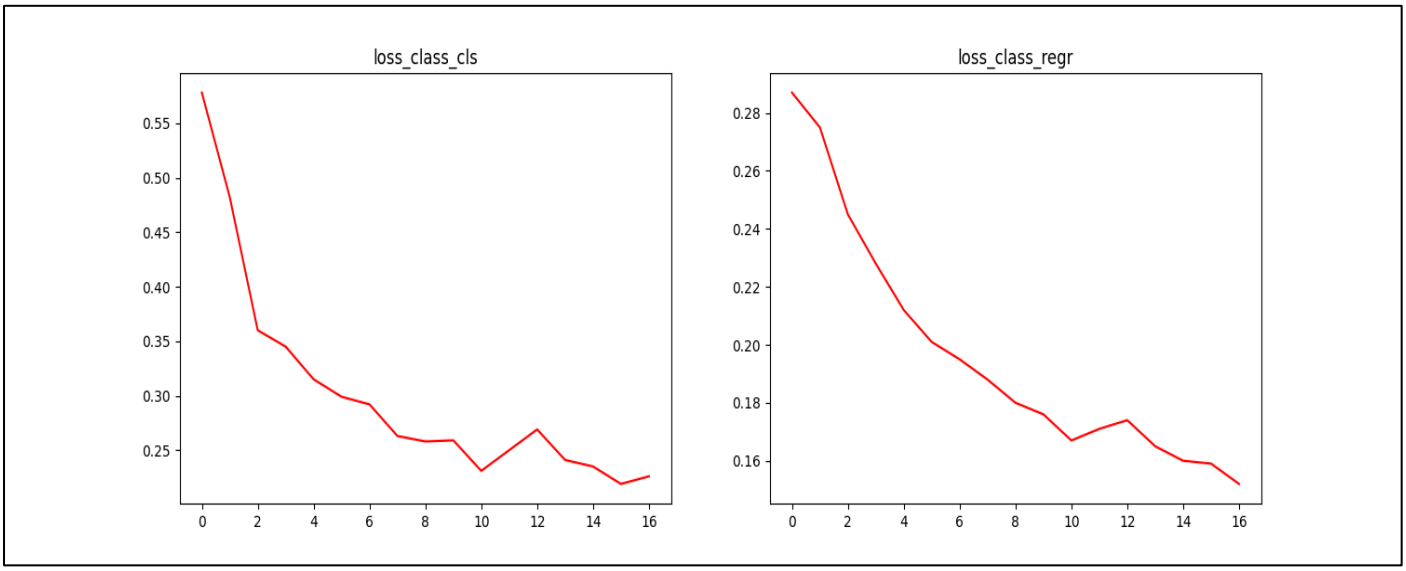


Fig 6.2 Loss of Fast R CNN Classification and Regression Layer

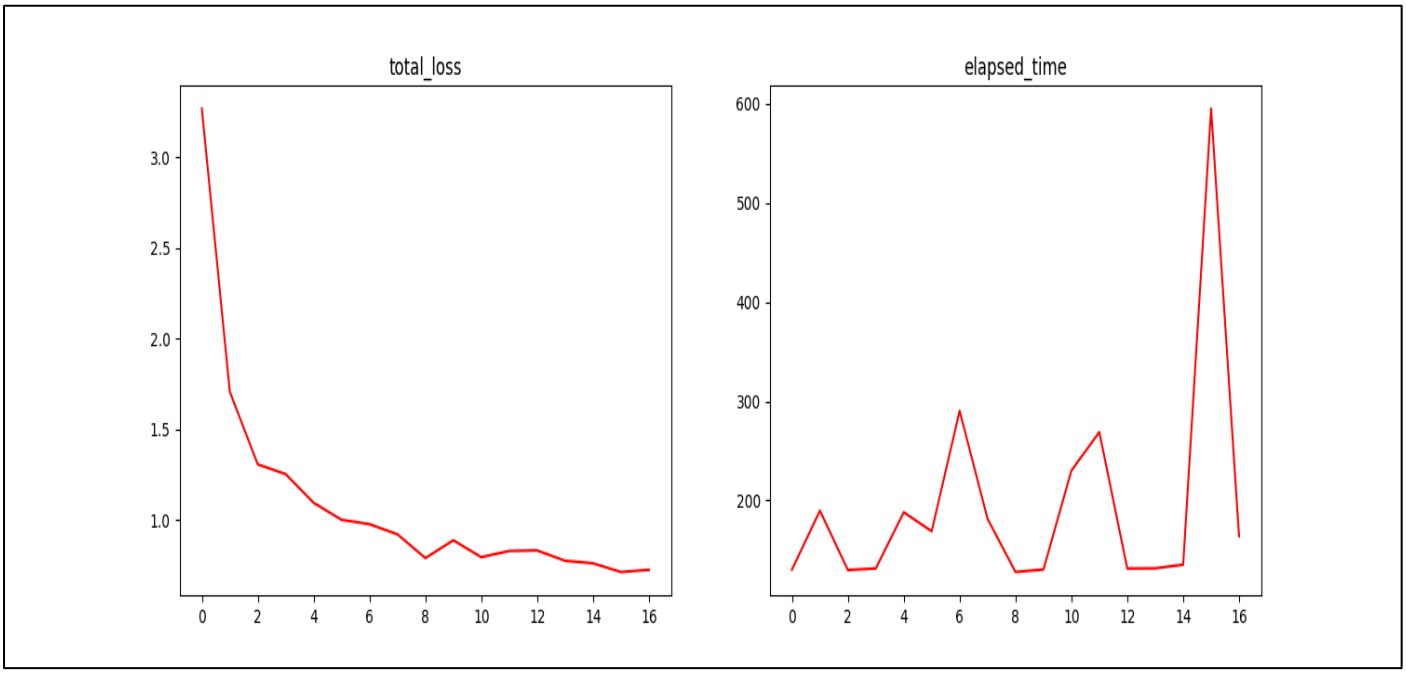


Fig 6.3 Total Loss of Faster R CNN (left) and Total Time taken for Process (right)

Conclusion

In conclusion, the implementation of Faster R-CNN for barcode detection has proven to be an accurate and efficient approach. The model can detect and classify barcodes with high precision and recall, and it is capable of processing large quantities of data in real-time. However, the performance of the model is heavily influenced by the quality of the input images, which can affect the detection accuracy, as well as factors such as lighting conditions and barcode orientation.

The benefits of using Faster R-CNN for barcode detection are numerous, including improved productivity, reduced error rates, and increased efficiency in inventory management. The ability to quickly and accurately scan barcodes can save businesses time and money, while also reducing the risk of errors that can lead to costly mistakes.

Despite the successes of the project, there are still some limitations to consider. One of the main challenges is the need for a large and diverse dataset to train the model effectively. Additionally, the model requires significant computing power to run, which may limit its accessibility for some users. Furthermore, the accuracy of the model may be limited when detecting barcodes in complex or cluttered environments.

Moving forward, future work could involve exploring the use of alternative deep learning models, such as YOLO or SSD, which may offer improved performance or efficiency. Additionally, optimizing the model parameters, refining the training data, and integrating the model with other technologies could enhance its capabilities and expand its potential applications. Overall, the implementation of Faster R-CNN for barcode detection has demonstrated the potential for deep learning to transform the way businesses handle inventory management and related tasks.