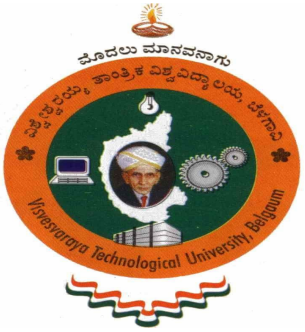
Visvesvaraya Technological University
Belgaum, Karnataka- 590014



A Project Report On

"Identification Of Scene Images"

Submitted in the partial fulfilment of the requirements for the award of the Degree of

BACHELOR OF ENGINEERING

In

INFORMATION SCIENCE AND ENGINEERING

ACCREDITED BY NBA

Submitted by

Rachita S (1DS16IS073) **Thejaswini Ram (1DS16IS115)**
Yuktha Lakshmi A B (1DS16IS122) **Zehra Ali (1DS16IS123)**

Under the Guidance of

Ms. Shilpashree S

Asst. Professor, Dept. of ISE

2019-2020

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

DAYANANDA SAGAR COLLEGE OF ENGINEERING

SHAVIGE MALLESHWARA HILLS, KUMARASWAMY LAYOUT, BANGALORE-78

DAYANANDA SAGAR COLLEGE OF ENGINEERING

Shavige Malleshwara Hills, Kumaraswamy Layout
Bangalore-560078

Department of Information Science and Engineering

ACCREDITED BY NBA

2019-2020

Certificate

This is to certify that the Project Work entitled —**"Identification of Scene Images"** is a bonafide work carried out by **Rachita S** (1DS16IS073), **Thejaswini Ram** (1DS16IS115), **Yuktha Lakshmi A B** (1DS16IS122), and **Zehra Ali** (1DS16IS123) in partial fulfilment for the 8th semester of Bachelor of Engineering in Information Science & Engineering of the Visvesvaraya Technological University, Belgaum during the year 2019-2020. The Project Report has been approved as it satisfies the academics prescribed for the Bachelor of Engineering degree.

_____	_____	_____
Signature of Guide [Ms. Shilpashree S]	Signature of HOD [Dr. K.N. Rama Mohan Babu]	Signature of Principal [Dr. C.P.S Prakash]

Name of the Examiners	Signature with Date
1._____	_____
2._____	_____

ACKNOWLEDGEMENT

It is great pleasure for us to acknowledge the assistance and support of a large number of individuals who have been responsible for the successful completion of this project.

We take this opportunity to express our sincere gratitude to **Dayananda Sagar College of Engineering** for having provided us with a great opportunity to pursue our Bachelor Degree in this institution.

In particular we would like to thank **Dr. C. P. S Prakash,** Principal, Dayananda Sagar College of Engineering for his constant encouragement and advice.

Special thanks to **Dr. K.N. Rama Mohan Babu,** HOD, Department of Information Science & Engineering, Dayananda Sagar College of Engineering for his motivation and invaluable support well through the development of this project.

We are highly indebted to our internal guide **Ms. Shilpashree S,** Asst. Professor, Department of Information Science & Engineering, Dayananda Sagar College of Engineering for his constant support and guidance. He has been a great source of inspiration throughout the course of this project. Finally, we gratefully acknowledge the support of our families during the completion of the project.

Rachita S (1DS16IS073)

Thejaswini Ram (1DS16IS115)

Yuktha Lakshmi A B (1DS16IS122)

Zehra Ali (1DS16IS123)

ABSTRACT

The purpose of this project is to develop a computer vision model using deep learning techniques that can detect objects in a live video and when given image or video as input . Recognizing objects in a video is a difficult task and the difficulty increases if the inference is expected in the real-time. The project begins with analyzing advantages and disadvantages of existing object detection architectures on the self-prepared dataset. The goal is to come up with an optimized object detection architecture using advantages of existing architectures. The developed architecture may be used for various applications in real-time with better accuracy and lesser time taken for generating the inference.

The superiority of the architecture will be demonstrated by developing a computer vision model that recognizes objects belonging to different classes(College buildings in our case). The model will be deployed to detect objects in a live streaming video or with an image or video as the input .

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Overview:

Our brain instantly recognizes the objects contained when we're shown an image. A lot of time is taken and a huge amount of training data is required for a machine to identify these objects. But with the recent advances in deep learning and hardware aspects, the computer vision field has become more intuitive.

A Convolutional Neural Network is a deep learning algorithm which can take in an input image, assign importance (biases and learnable weights) to various objects in the image and be able to differentiate one from the other. The pre-processing required in a convolutional network is much lower as compared to other classification algorithms. While in primitive methods filters are manually designed, with enough training, convolutional networks have the ability to learn the filters and characteristics.

Given that there are many techniques for object detection which have been developed, this project aims at reviewing and researching the efficiency of the major techniques used for live object detection in videos using Convolutional Neural Networks.

1.2 Problem Statement

Object Identification and classification are classical problems where we need to identify the objects in images. In our project, we aim to recognize the important buildings of our college in images. We plan on using YOLO, which is one of the most effective object detection algorithms, to accomplish this task.

1.3 Objective

To identify important buildings of our college in images and to build a Deep Learning model to help automate this process.

1.4 Motivation

The ubiquitous and wide applications like scene understanding, video surveillance, robotics, and self-driving systems triggered vast research in the domain of computer vision in the most recent decade. Being the core of all these applications, visual recognition systems which encompasses image classification, localization and detection have achieved great research momentum. Due to significant development in neural networks especially deep learning, these visual recognition systems have attained remarkable performance.

Object detection is one of these domains witnessing great success in computer vision. Deep learning technology has become a buzzword nowadays due to the state-of-the-art results obtained in the domain of image classification, object detection, natural language processing. The reasons behind the popularity of deep learning are two folded, viz. large availability of datasets and powerful Graphics Processing Units.

1.5 Existing Systems :

- R-CNN and their variants, including the original R-CNN, Fast R- CNN, and Faster R-CNN
- Single shot detector(SSD)

R-CNN , Fast R-CNN, Faster R-CNN

R-CNNs are one of the first deep learning-based object detectors and are an example of a two-stage detector. In the first R-CNN publication, Rich feature hierarchies for accurate object detection and semantic segmentation, (2013) Girshick et al. proposed an object detector that required an algorithm such as Selective Search (or equivalent) to propose candidate bounding boxes that could contain objects. These regions were then passed into a CNN for classification, ultimately leading to one of the first deep learning-based object detectors.Girshick et al. published a second paper in 2015, entitled Fast R- CNN. The Fast R- CNN algorithm made considerable improvements to the original R-CNN, namely increasing accuracy and reducing the time it took to perform a forward pass; however, the model still relied on an external region proposal algorithm.

It wasn't until Girshick et al.'s follow-up 2015 paper, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, that R-CNNs became a true end-to-end deep learning object detector by removing the Selective Search requirement and instead relying on a Region Proposal Network (RPN) that is (1) fully convolutional and (2) can predict the object bounding boxes and "objectness" scores (i.e., a score quantifying how likely it is a region of an image may contain an image). The outputs of the RPNs are then passed into the R-CNN component for final classification and labeling.

Disadvantages of R-CNN, Fast R-CNN, Faster R-CNN

- The problem with the standard R-CNN method was that it was *painfully slow* and not a complete end-to-end object detector.
- While R-CNNs tend to very accurate, the biggest problem with the R-CNN family of networks is their speed — they were incredibly slow, obtaining only 5 FPS on a GPU.

Single Shot Detector (SSD)

SSD attains a better balance between swiftness and precision. SSD runs a convolutional network on input image only one time and computes a feature map. We run a small 3×3 sized convolutional kernel on this feature map to foresee the bounding boxes and categorization probability.

SSD also uses anchor boxes at a variety of aspect ratio comparable to Faster-RCNN and learns the off-set to a certain extent than learning the box. In order to hold the scale, SSD predicts bounding boxes after multiple convolutional layers. Since every convolutional layer functions at a diverse scale, it is able to detect objects of a mixture of scales.

Disadvantages of SSD

In general, single-stage detectors tend to be less accurate than two-stage detectors but are significantly faster.

1.6 Proposed System:

You only look once (YOLO)

For YOLO, detection is a straightforward regression dilemma which takes an input image and learns the class possibilities with bounding box coordinates. YOLO divides every image into a grid of S x S and every grid predicts N bounding boxes and confidence. The confidence reflects the precision of the bounding box and whether the bounding box in point of fact contains an object in spite of the defined class. YOLO even forecasts the classification score for every box for each class. You can

merge both the classes to work out the chance of every class being in attendance in a predicted box.

So, total SxSxN boxes are forecasted. On the other hand, most of these boxes have lower confidence scores and if we set a doorstep say 30% confidence, we can get rid of most of them.

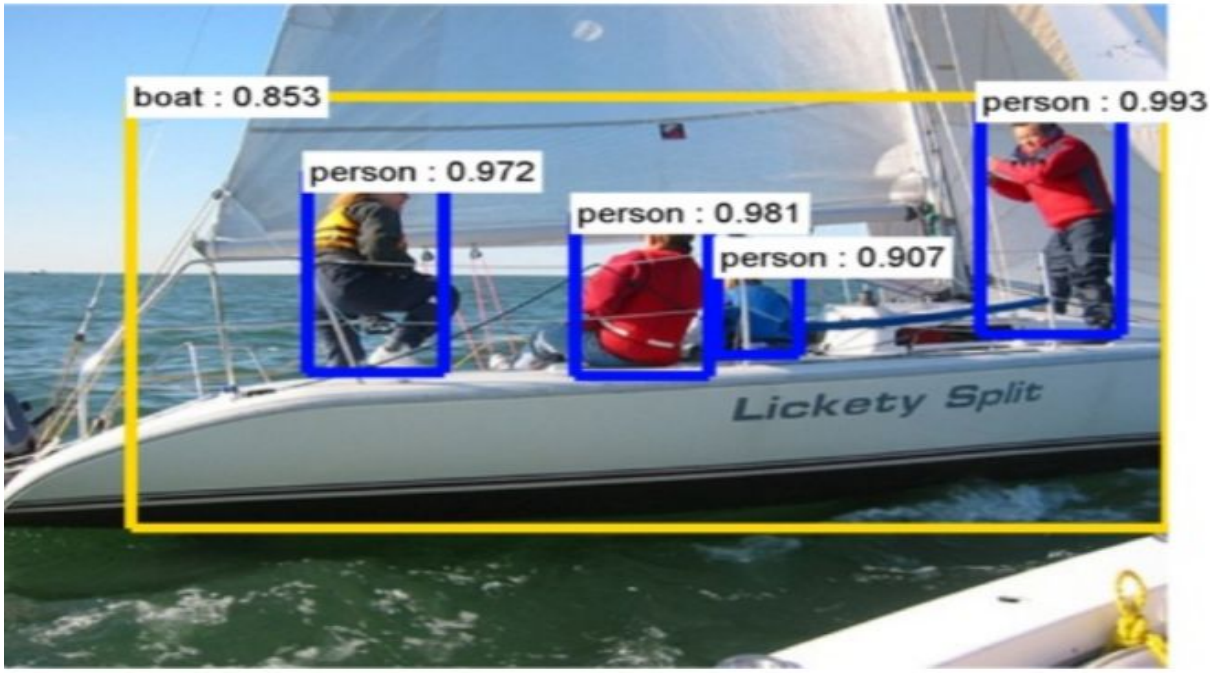


Fig. 1.5.1 Object Detection with YOLO

Features of YOLO

* YOLO, an object detection algorithm finds all objects in an image grid simultaneously

* Uses a single convolutional network for full image

* YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearances, unlike the sliding window or region-based techniques. Thus making less than half the number of background errors compared to Fast R-CNN.

* YOLO uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously. Predicts the bounding boxes and the class probabilities for these boxes.

* Treats detection as a regression problem

* Extremely fast and accurate

Advantages of YOLO

- Speed (45 frames per second — better than realtime)
- Network understands generalized object representation (This allowed them to train the network on real world images and predictions on artwork was still fairly accurate).
- faster version (with smaller architecture) — 155 frames per sec but is less accurate.
- open source: https://pjreddie.com/darknet/yolo/

CHAPTER 2

LITERATURE SURVEY

Various Books and information materials from the web regarding Deep learning and Convolutional Neural Networks have been studied through in order to achieve the required information concern to this project. Among them, following are the key points extracted through:

1. The Research paper **"Object Detection with Deep Learning: A Review"** by **Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, Xindong Wu** gave us a brief knowledge of how with the rapid development in deep learning, more powerful tools, which are able to learn semantic, high-level, deeper features, are introduced to address the problems existing in traditional architectures.

   In this paper, a review on deep learning based object detection frameworks is provided. The review begins with a brief introduction on the history of deep learning and its representative tool, namely Convolutional Neural Network (CNN). Then focus shifts to typical generic object detection architectures along with some modifications and useful tricks to improve detection performance further.

2. The Research paper "**Application of deep learning in object detection"** by **Xinyi Zhou , Wei Gong , WenLong Fu , Fengtong Du** gave a simple summary of the datasets and deep learning algorithms commonly used in computer vision.

   This paper deals with the field of computer vision, mainly for the application of deep learning in object detection tasks.

3. The Research paper **"You Only Look Once: Unified, Real-Time Object Detection"** by **Joseph Redmon, Santosh Divvala and Ali Farhadi,** we learnt about a new approach to object detection.

In Yolo, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

4. The Research paper **"YOLOv3: An Incremental Improvement"** by **Joseph Redmon and Ali Farhadi** , includes some updates to YOLO which are presented with a bunch of little design changes to make it better.

   In this paper, they speak about how speed has been traded off for boosts in accuracy in YOLO v3. It's a little bigger than YOLO v2 but more accurate.

5. The Research paper, **"Research on Computer Vision-Based Object Detection and Classification"** by **Juan Wu, Bo Peng, Zhenxiang Huang and Jietao Xie,** we learnt how two of the most demanding and widely studied applications related to Computer vision are object detection and classification. Research in these fields has resulted in a wealth of processing and analysis methods. This paper, explicitly explored current advances in the field of object detecting and categorizing based on computer vision, and a comparison of these methods is given.

6. The Research paper, **"Real-Time Object Detection with Yolo"** by **Geethapriya. S, N. Duraimurugan, S.P. Chokkalingam ,** shows how YOLO has several advantages as compared to other object detection algorithms. In other algorithms like Convolutional Neural Network, Fast Convolutional Neural Network the algorithm will not look at the image completely but in YOLO the algorithm looks the image completely by predicting the bounding boxes using convolutional network and the class probabilities for these boxes and detects the image faster as compared to other algorithms.

7. The https://medium.com/ website provided us the required real time implementation knowledge in order to code the model.

CHAPTER 3

REQUIREMENTS

3.1 Functional Requirements

The model should be able to recognise objects which it is trained on.

3.2 Non Functional Requirements

- The accuracy of the predicted value must be precise.
- The model should never fail in the middle of operation.
- The model should work consistently across various platforms.

3.2.1.1 Software Requirements

Coding Language: Python 3.6

Realtime Computer Vision with openCV

Tensorflow

Keras

NumPy

3.2.1.2 Hardware Requirements

Processor: i5 or i7 Intel Processor

Primary Storage: 8 GB RAM or above (Recommended 16 GB)

Secondary Storage: Any standard HDD or SDD

Web-Cam

CHAPTER 4

SYSTEM ANALYSIS & DESIGN

4.1 Analysis

The procedure of breaking a difficult topic or substance into small parts to gain a better knowledge of the problem is known as analysis. Analysts in the field of engineering look at the structures and requirements, mechanisms, and systems dimensions. Analysis is an activity of exploration. The project life cycle begins in the analysis .

4.2 System Design

The definition of the architecture of a system, components, modules, interfaces, and data for a system to fulfil specified requirements is system design. Systems design could be seen as the application of systems theory to product development.

The design phase produces the overall design of the software. The goal of the design phase is to figure out the modules that should be in the system to fulfil all the system requirements in an efficient manner. It will contain the details of all these modules, their working with other modules and the desired output from each module. The output of the design process is a description of the software architecture.

4.2.1 System Architecture Diagram

The definition of the structure and operation and more views of a system is known as system architecture. A formal description and rendition of a system, organized in a way so that it supports reasoning about the working and behaviors of the system is called architecture description.System architecture comprises system components that work together and implement the overall system.

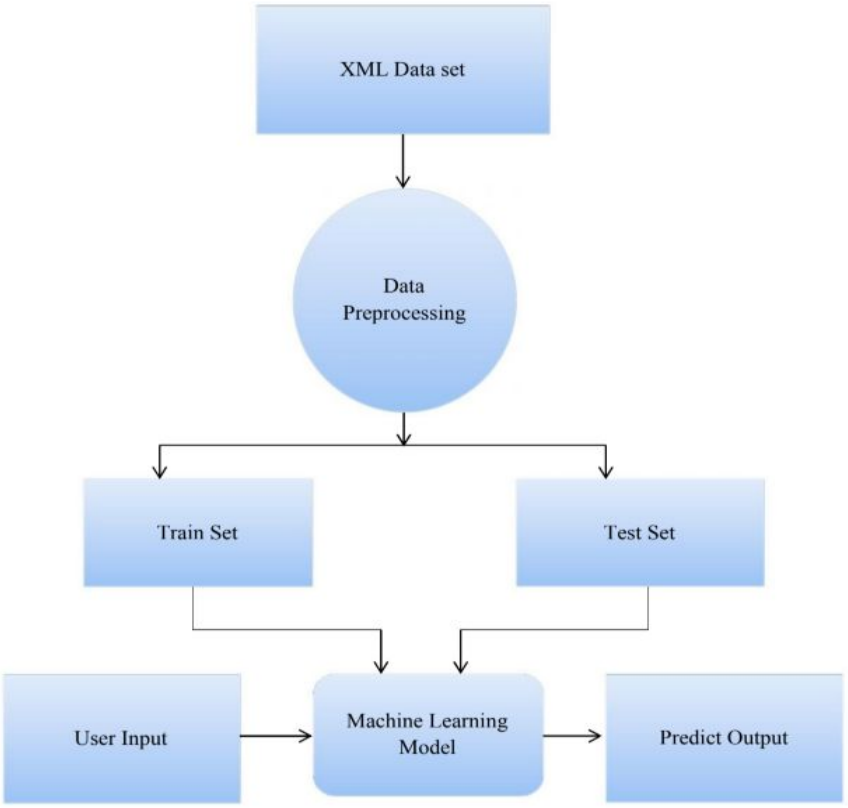


Fig. 4.2.1: System Architecture Diagram

4.2.1.1 Data Flow Diagram

A data flow diagram is the graphical representation of the flow of data through an information system. DFD is very useful in understanding a system and can be efficiently used during analysis. A DFD shows the flow of data through a system. It views a system as a function that transforms the inputs into desired outputs. Any complex systems will not perform this transformation in a single step and a data will typically undergo a series of transformations before it becomes the output.

With a data flow diagram, users are able to visualize how the system will operate that the system will accomplish and how the system will be implemented, old system data flow diagrams can be drawn up and compared with a new systems data flow diagram to draw comparisons to implement a more efficient system.

Data flow diagrams can be used to provide the end user with a physical idea of where they input, ultimately as an effect upon the structure of the whole system.

In the perspective of application development Data Flow Diagram (DFD) is a special chart type which lets graphically illustrate the "flow" of data through various application component. So the Data Flow Diagrams can be successfully used for visualization of data processing or structured design.

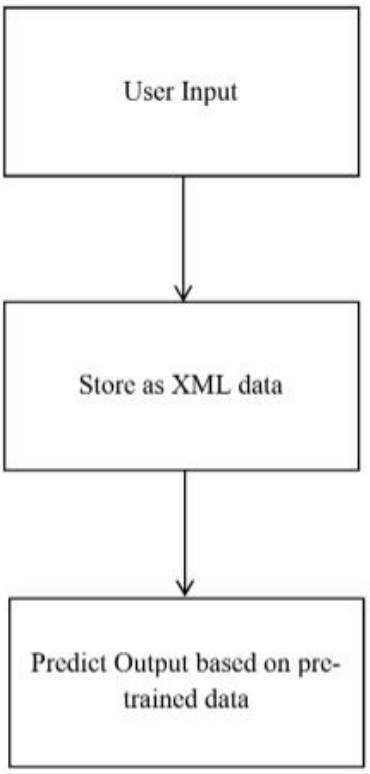


Fig. 4.2.1.1: Data Flow Diagram

4.2.1.2 Flow Chart

Flowcharts are used in designing and documenting simple processes or programs.Like other types of diagrams, they help visualize what is going on and thereby help understand a process, and perhaps also find flaws, bottlenecks, and other less-obvious features within it. There are many different types of flowcharts, and each type has its own repertoire of boxes and notational conventions.

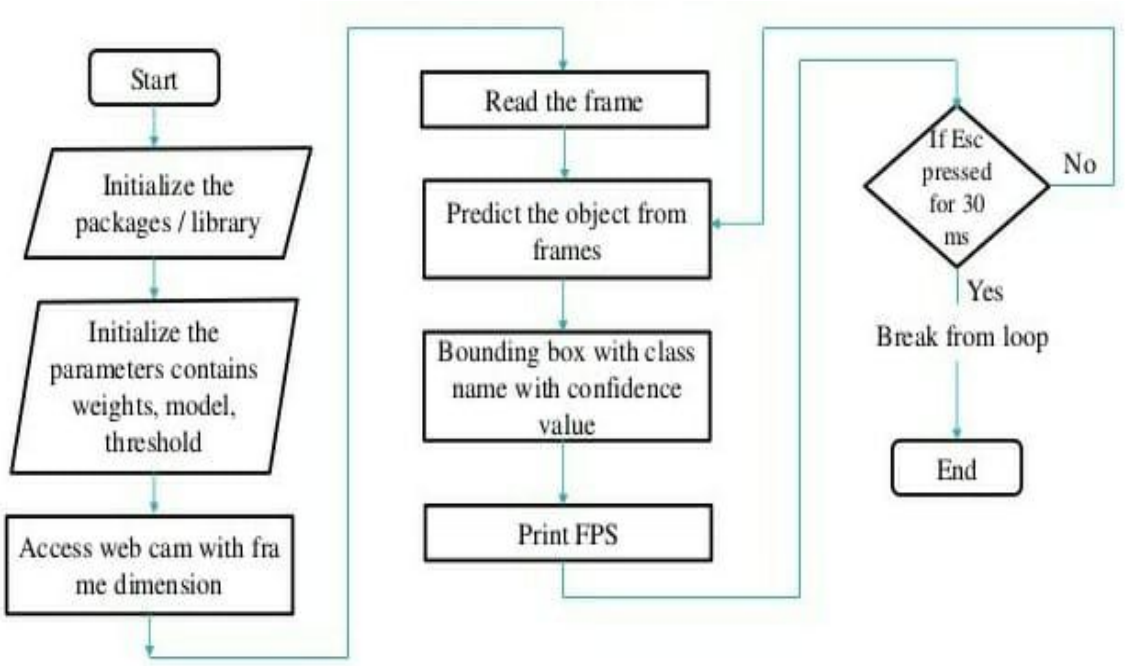
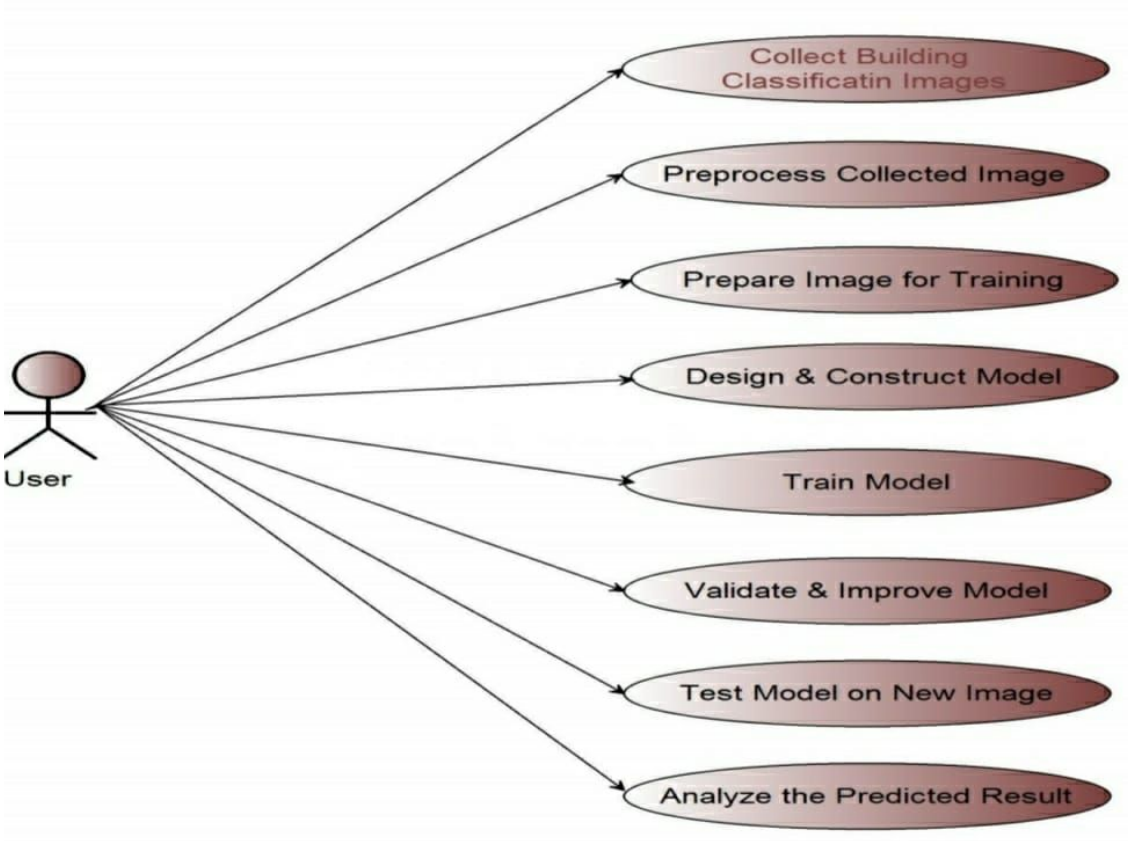


Fig.4.2.1.2 Flowchart

4.2.1.3 Use Case Diagram

Use case represents the functionality or services provided by the system to users.Use case diagram depicts different users and functions of the system. It uncovers the requirement of the system.In this project actors are the users of the system. Association between the actors and the use case are indicated by solid lines,an association exists whenever an actor is involved with an interaction described by the use case.



Fig`.4.2.1.3 Use case Diagram

4.2.1.4 Sequence Diagram

A sequence diagram in a Unified Modelling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It shows the participants in an interaction and the sequence of messages among them; each participant is assigned a column in the table.

Graphically a sequence diagram is a table that shows objects arranged along the X axis and messages, ordered in increasing time along the Y axis.

- Order- Instance of the class participating in interactions.
- Lifeline-The lifeline represents the object's life during interactions.
- Activation- Activation box represents the time an object needs to complete the task.

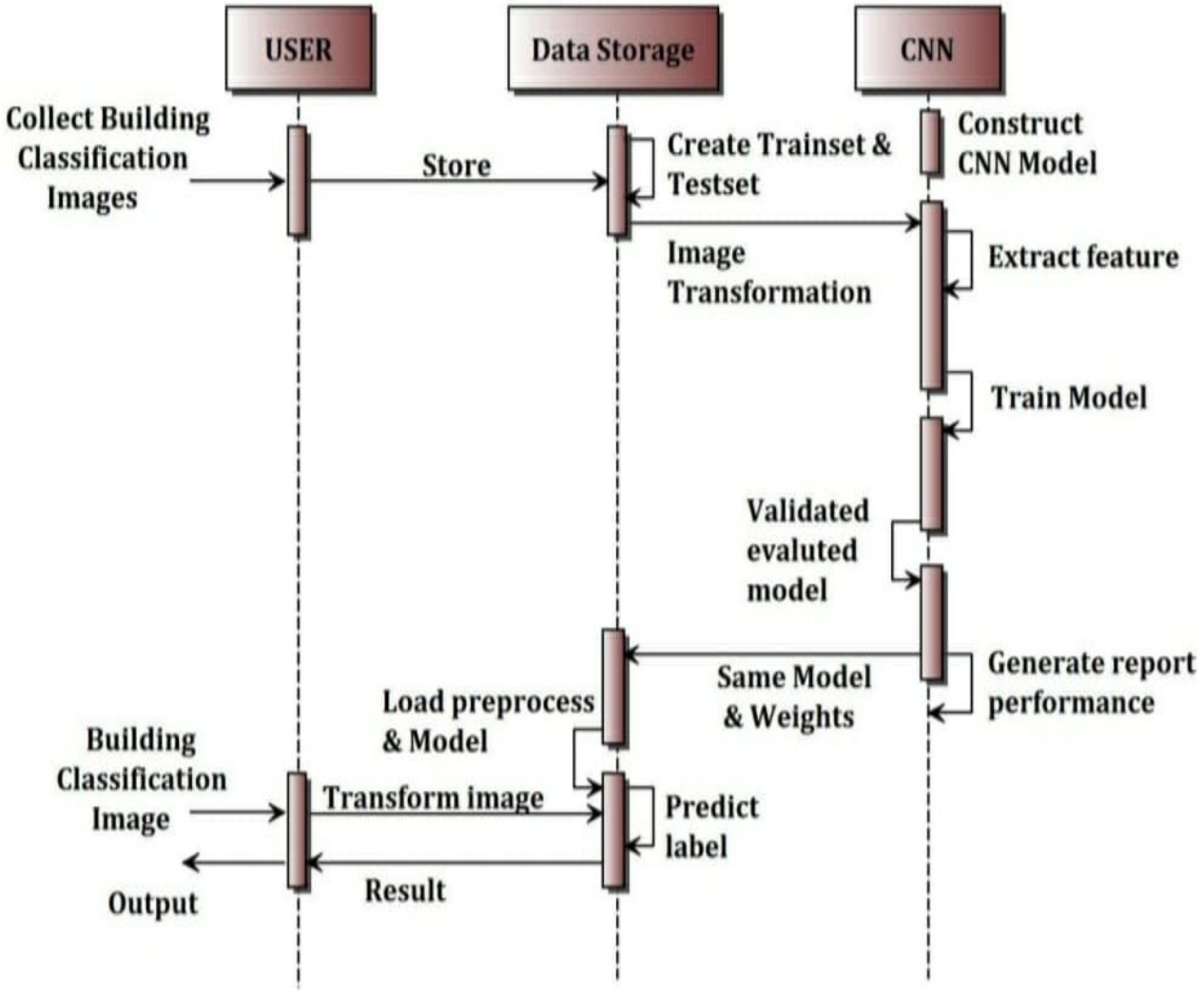


Fig.4.2.1.4 Sequence Diagram

CHAPTER 5

IMPLEMENTATION

5.1 Introduction

The realizing of an application or execution of a plan, idea, model, design, specification, standard, algorithm, or policy is known as implementation. In other words, an implementation is a realization of a technical specification or algorithm as a program, software component, or other computer system through programming and deployment. Many implementations may exist for a given specification or standard.

Implementation is one of the most important phases of the Software Development Life Cycle (SDLC). It encompasses all the processes involved in getting new software or hardware operating properly in its environment, including installation, configuration, running, testing, and making necessary changes. Specifically, it involves coding the system using a particular programming language and transferring the design into an actual working system.

5.2 Overview of System Implementation

This project is implemented considering the following aspects:

Usability Aspect.

Technical Aspect.

5.2.1 Usability Aspect

The usability aspect of implementation of the project is realized using two principles:

5.2.1.1 The project is implemented using PYTHON

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Python features a dynamic tape system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open-source software and has a community-based development model. Python and CPython are managed by the non-profit Python Software Foundation.

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

5.2.2 Technical Aspect

The technical aspect of implementation of the project is realized using following principle:

5.2.2.1 Keras

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML.Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a

Google engineer. Chollet also is the author of the XCeption deep neural network model.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.

In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine.[3] It also allows use of distributed training of deep-learning models on clusters of Graphics processing units (GPU) and tensor processing units (TPU) principally in conjunction with CUDA.

5.2.2.2 NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

a powerful N-dimensional array object

sophisticated (broadcasting) functions

tools for integrating C/C++ and Fortran code

useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

5.2.2.3 Tensorflow

TensorFlow can train and run deep neural networks for handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing,

and PDE (partial differential equation) based simulations. Best of all, TensorFlow supports production prediction at scale, with the same models used for training.

TensorFlow allows developers to create *dataflow graphs*—structures that describe how data moves through a graph, or a series of processing nodes. Each node in the graph represents a mathematical operation, and each connection or edge between nodes is a multidimensional data array, or *tensor*.

TensorFlow provides all of this for the programmer by way of the Python language. Python is easy to learn and work with, and provides convenient ways to express how high-level abstractions can be coupled together. Nodes and tensors in TensorFlow are Python objects, and TensorFlow applications are themselves Python applications.

The actual math operations, however, are not performed in Python. The libraries of transformations that are available through TensorFlow are written as high-performance C++ binaries. Python just directs traffic between the pieces, and provides high-level programming abstractions to hook them together.

5.2.2.4 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken

using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

5. 2. 2. 5 YOLO

"You Only Look Once" is an algorithm that uses convolutional neural networks for object detection. You only look once, or YOLO, is one of the faster object detection algorithms out there. Though it is not the most accurate object detection algorithm, but it is a very good choice when we need real-time detection, without loss of too much accuracy.

In comparison to recognition algorithms, a detection algorithm does not only predict class labels but detects locations of objects as well. So, It not only classifies the image into a category, but it can also detect multiple Objects within an Image. This Algorithm applies a single Neural network to the Full Image. It means that this network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

Architecture

YOLO makes use of only convolutional layers, making it a fully convolutional network (FCN). In YOLO v3 paper, the authors present new, deeper architecture of feature extractor called Darknet-53. As it's name suggests, it contains of 53 convolutional layers, each followed by batch normalization layer and Leaky ReLU activation. No form of pooling is used, and a convolutional layer with stride 2 is used to downsample the feature maps. This helps in preventing loss of low-level features often attributed to pooling.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Fig.5.2.2.5.1 Darknet-53 model

Interpreting the output

The input is a batch of images of shape (m, 416, 416, 3).

The output is a list of bounding boxes along with the recognized classes. Each bounding box is represented by 6 numbers (pc,bx,by,bh,bw,c). If we expand c into an 80-dimensional vector, each bounding box is then represented by 85 numbers.

For example if we have (B x (5 + C)) entries in the feature map. B represents the number of bounding boxes each cell can predict. According to the paper, each of these B bounding boxes may specialize in detecting a certain kind of object. Each of the bounding boxes have 5 + C attributes, which describe the center coordinates, the

dimensions, the objectiveness score and C class confidences for each bounding box. YOLO v3 predicts 3 bounding boxes for every cell.

We expect each cell of the feature map to predict an object through one of it's bounding boxes if the center of the object falls in the receptive field of that cell.

This has to do with how YOLO is trained, where only one bounding box is responsible for detecting any given object. First, we must ascertain which of the cells this bounding box belongs to.

To do that, we divide the input image into a grid of dimensions equal to that of the final feature map. Take a look at example below, where the input image is 416 x 416, and stride of the network is 32. As pointed earlier, the dimensions of the feature map will be 13 x 13. We then divide the input image into 13 x 13 cells.

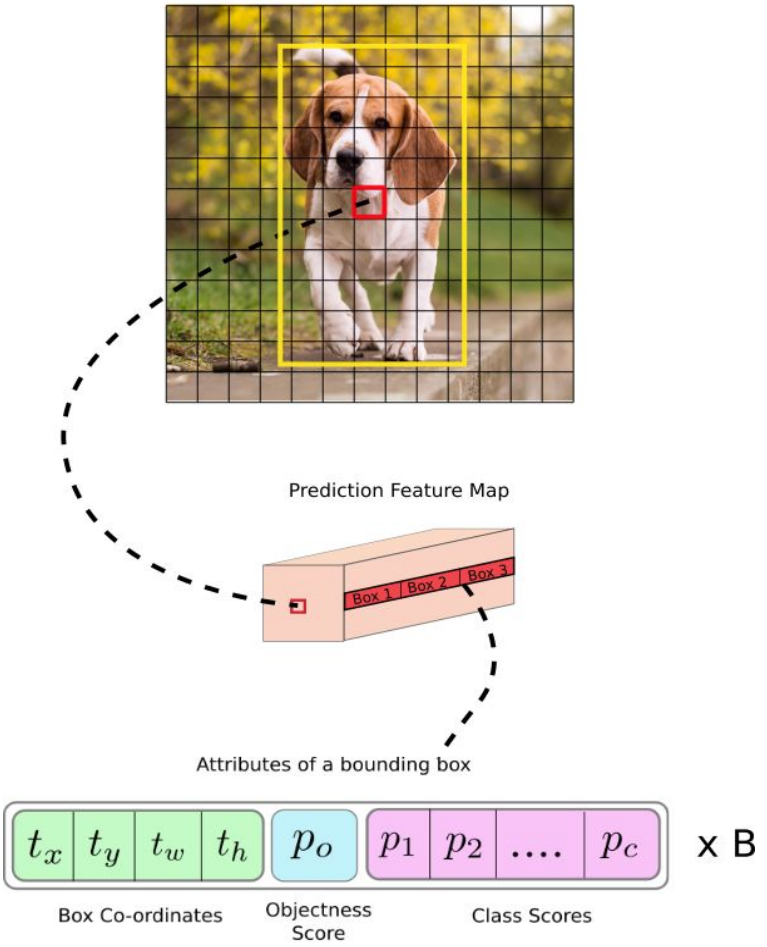


Fig. 5.2.2.5.2 determining which cell of the prediction feature map is responsible for prediction

Now, the red cell is the 7th cell in the 7th row on the grid. We now assign the 7th cell in the 7th row on the feature map (corresponding cell on the feature map) as the one

responsible for detecting the dog.Now, this cell can predict three bounding boxes.

Which one will be assigned to the dog's ground truth label? In order to understand that, we must wrap out head around the concept of anchors. (Note that the cell we're talking about here is a cell on the prediction feature map. We divide the input image

into a grid just to determine which cell of the prediction feature map is responsible for prediction).

For better understanding we can analyse another example with more images:

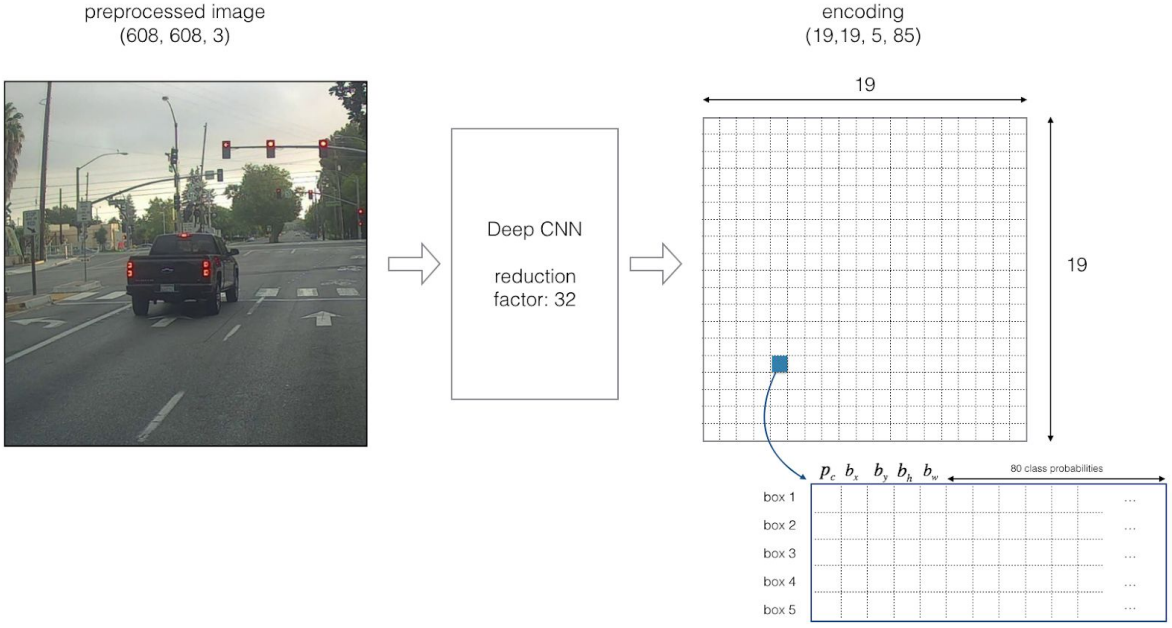


Fig. 5.2.2.5.3 YOLO v3 architecture for given image

If the center/midpoint of an object falls into a grid cell, that grid cell is responsible for detecting that object.

Since we are using 5 anchor boxes, each of the 19x19 cells thus encodes information about 5 boxes. For simplicity, we will flatten the last two dimensions of the shape (19, 19, 5, 85) encoding. So the output of the Deep CNN is (19, 19, 425):s. Anchor boxes are defined only by their width and height.

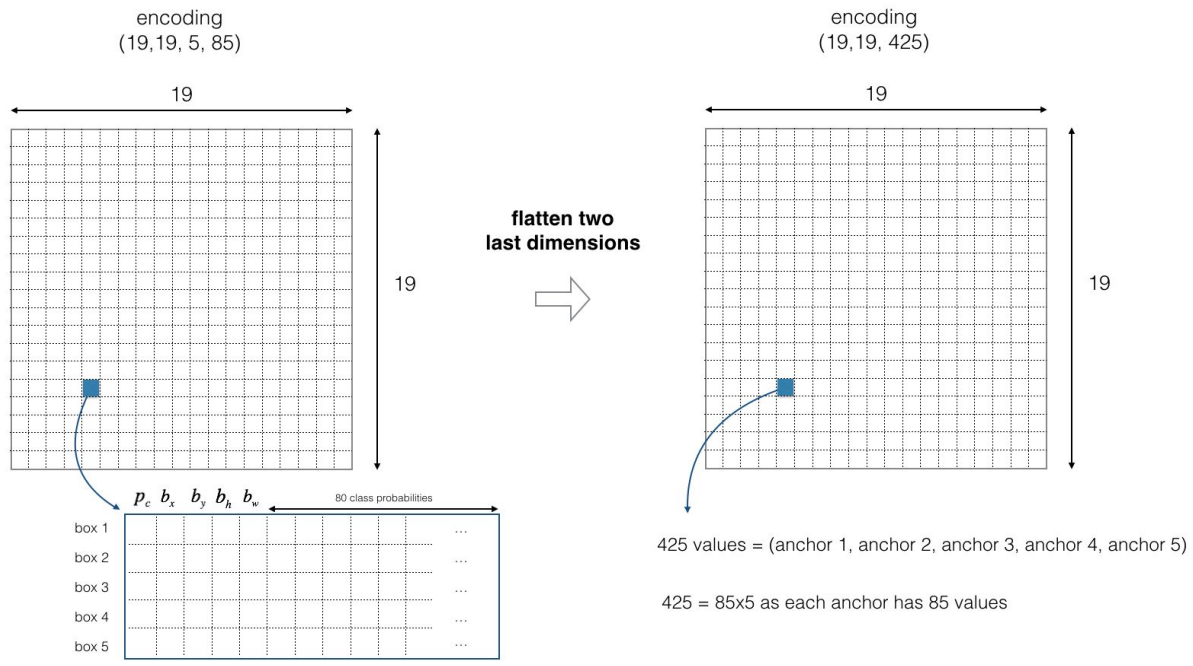


Fig. 5.2.2.5.4 Flattening the last two dimensions

Now, for each box (of each cell) we will compute the following elementwise product and extract a probability that the box contains a certain class.

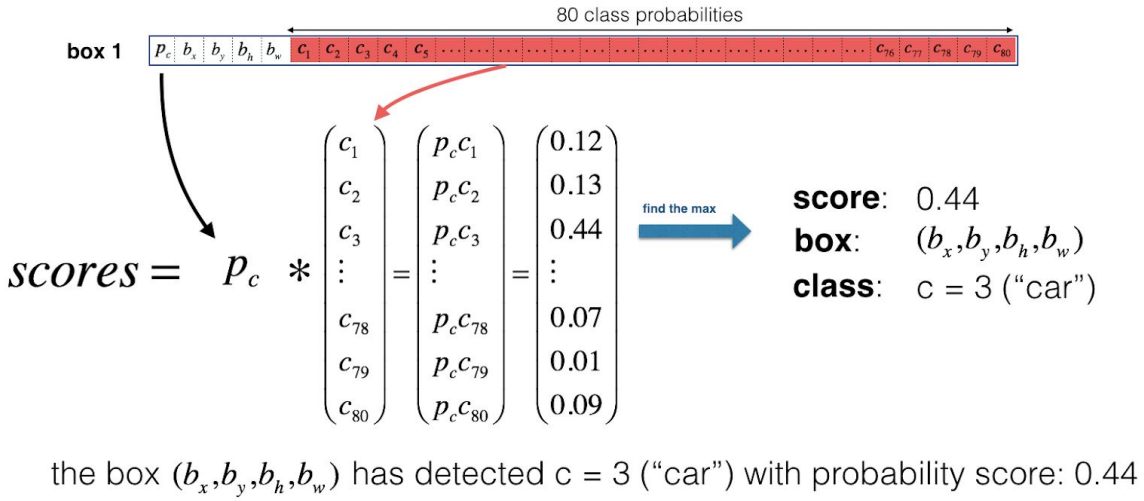


Fig. 5.2.2.5.5 Extract a probability that the box contains a certain class.

Anchor Boxes and Predictions

It might make sense to predict the width and the height of the bounding box, but in practice, that leads to unstable gradients during training. Instead, most of the modern object detectors predict log-space transforms, or simply offsets to pre-defined default bounding boxes called anchors.

Then, these transforms are applied to the anchor boxes to obtain the prediction. YOLO v3 has three anchors, which result in prediction of three bounding boxes per cell.

Anchors are sort of bounding box priors, that were calculated on the COCO dataset using k-means clustering. We are going to predict the width and height of the box as offsets from cluster centroids. The center coordinates of the box relative to the location of filter application are predicted using a sigmoid function.

The following formulae describes how the network output is transformed to obtain bounding box predictions:

$bx=\sigma(tx)+cx$

$by=\sigma(ty)+cy$

$bw=pw*etw$

$bh=ph*eth$

Here bx, by, bw, bh are the x,y center coordinates, width and height of our prediction. tx, ty, tw, th is what the network outputs. cx and cy are the top-left coordinates of the grid. pw and ph are anchors dimensions for the box.

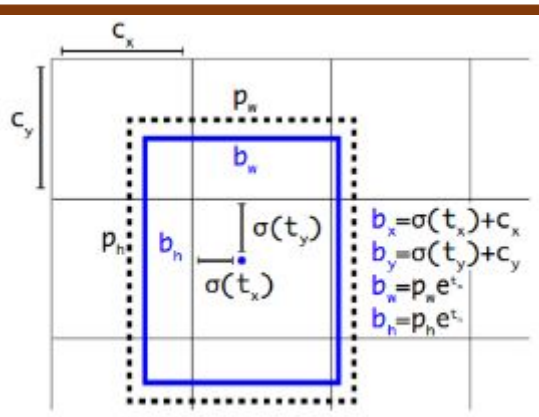


Fig. 5.2.2.5.6 Bounding boxes with dimension priors and location prediction.

Objectness Score and Class Confidences

First of all, object score represents the probability that an object is contained inside a bounding box. It should be nearly 1 for the red and the neighboring grids, whereas almost 0 for, say, the grid at the corners.

The objectness score is also passed through a sigmoid, as it is to be interpreted as a probability.

Talking about Class confidences, they represent the probabilities of the detected object belonging to a particular class (Dog, cat, person, car, bicycle etc). In older YOLO versions it was used to softmax the class scores.

In YOLO authors have decided using sigmoid instead. The reason is that Softmaxing class scores assume that the classes are mutually exclusive. In simple words, if an object belongs to one class, then it cannot belong to another class. This is true for COCO database, which we will implement first.

Output Processing (Filtering with a threshold on class scores)

For an image of size 416 x 416, YOLO predicts ((52 x 52) + (26 x 26) + 13 x 13)) x 3 = 10647 bounding boxes. However, in case of our image, there's only one object, a dog. So, you may ask how do we reduce the detections from 10647 to 1?

First, we filter boxes based on their objectness score. Generally, boxes having scores below a threshold (for example below 0.5) are ignored. Next, Non-maximum Suppression (NMS) intends to cure the problem of multiple detections of the same image. For example, all the 3 bounding boxes of the red grid cell may detect a box or the adjacent cells may detect the same object, so NMS is used to remove multiple detections.

Specifically, we'll carry out these steps:

Get rid of boxes with a low score (meaning, the box is not very confident about detecting a class)

Select only one box when several boxes overlap with each other and detect the same object (Non-max suppression).

For better understanding we will use same example with car we mentioned above. At first we are going to apply a first filter by thresholding. We would like to get rid of any box for which the class "score" is less than a chosen threshold. The model gives us a total of 19x19x5x85 numbers, with each box described by 85 numbers. It'll be convenient to rearrange the (19,19,5,85) (or (19,19,425)) dimensional tensor into the following variables:

box_confidence: tensor of shape (19×19,5,1) containing pc (confidence probability that there's some object) for each of the 5 boxes predicted in each of the 19x19 cells.

boxes: tensor of shape (19×19,5,4) containing (bx,by,bh,bw) for each of the 5 boxes per cell.

box_class_probs: tensor of shape (19×19,5,80) containing the detection probabilities (c1,c2,...c80) for each of the 80 classes for each of the 5 boxes per cell.

Even after filtering by thresholding over the classes scores, we still end up a lot of overlapping boxes. A second filter for selecting the right boxes is called NMS. NMS uses the very important function called "Intersection over Union", or IoU.

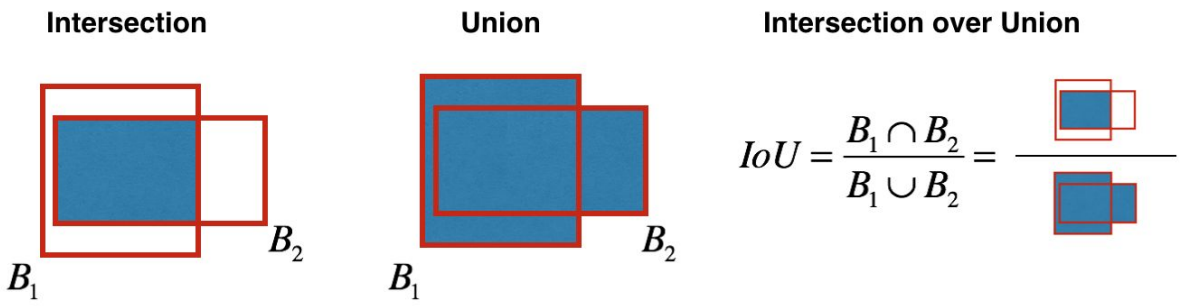


Fig. 5.2.2.5.7 IoU

Implementing IoU:

So we define a box using its two corners (upper left and lower right): (x1, y1, x2, y2) rather than the midpoint and height/width.

To calculate the area of a rectangle we need to multiply its height (y2 - y1) by its width (x2 - x1)

We also need to find the coordinates (xi1, yi1, xi2, yi2) of the intersection of two boxes:

xi1 = maximum of the x1 coordinates of the two boxes

yi1 = maximum of the y1 coordinates of the two boxes

xi2 = minimum of the x2 coordinates of the two boxes

yi2 = minimum of the y2 coordinates of the two boxes

Arguments:
box1 - first box, list object with coordinates (x1, y1, x2, y2)
box2 - second box, list object with coordinates (x1, y1, x2, y2)

Code:

```
xi1 = max(box1[0], box2[0])
yi1 = max(box1[1], box2[1])
xi2 = min(box1[2], box2[2])
yi2 = min(box1[3], box2[3])
```

```
inter_area = (xi2 - xi1)*(yi2 - yi1)

# Formula: Union(A,B) = A + B - Inter(A,B)
box1_area = (box1[3] - box1[1])*(box1[2]- box1[0])
box2_area = (box2[3] - box2[1])*(box2[2]- box2[0])
union_area = (box1_area + box2_area) - inter_area

# compute the IoU
IoU = inter_area / union_area
```

Now, to implement non-max suppression, the key steps are:

Select the box that has the highest score.

Compute its overlap with all other boxes, and remove boxes that overlap it more than iou_threshold.

Go back to step 1 and iterate until there's no more boxes with a lower score than the current selected box.

These steps will remove all boxes that have a large overlap with the selected boxes. Only the "best" boxes remain:

Fig. 5.2.2.5.8 Non-Max Suppression

5.3 Implementation Support

5.3.1 Installation of LabelImg for annotating

In Computer Vision Problems Image Annotation is much important to detect and mark faces in images. labelImg is an excellent tool which makes the face annotation boxes to XML files easily.

Python3 and above

(i) Open Ubuntu Terminal (Ctrl+Alt+T)

(ii) *sudo apt-get install pyqt5-dev-tools*

(iii) *sudo pip install lxml*

(iv) Now clone the **labelImg GitHub** repository on your machine or download the zip file and extract.

(v) cd into the labelImg directory *cd labelImg-master*

(vi) In your terminal type *make qt5py3*

(vii) That's it. Now you can Run the *python labelImg.py* to run the labelImg.

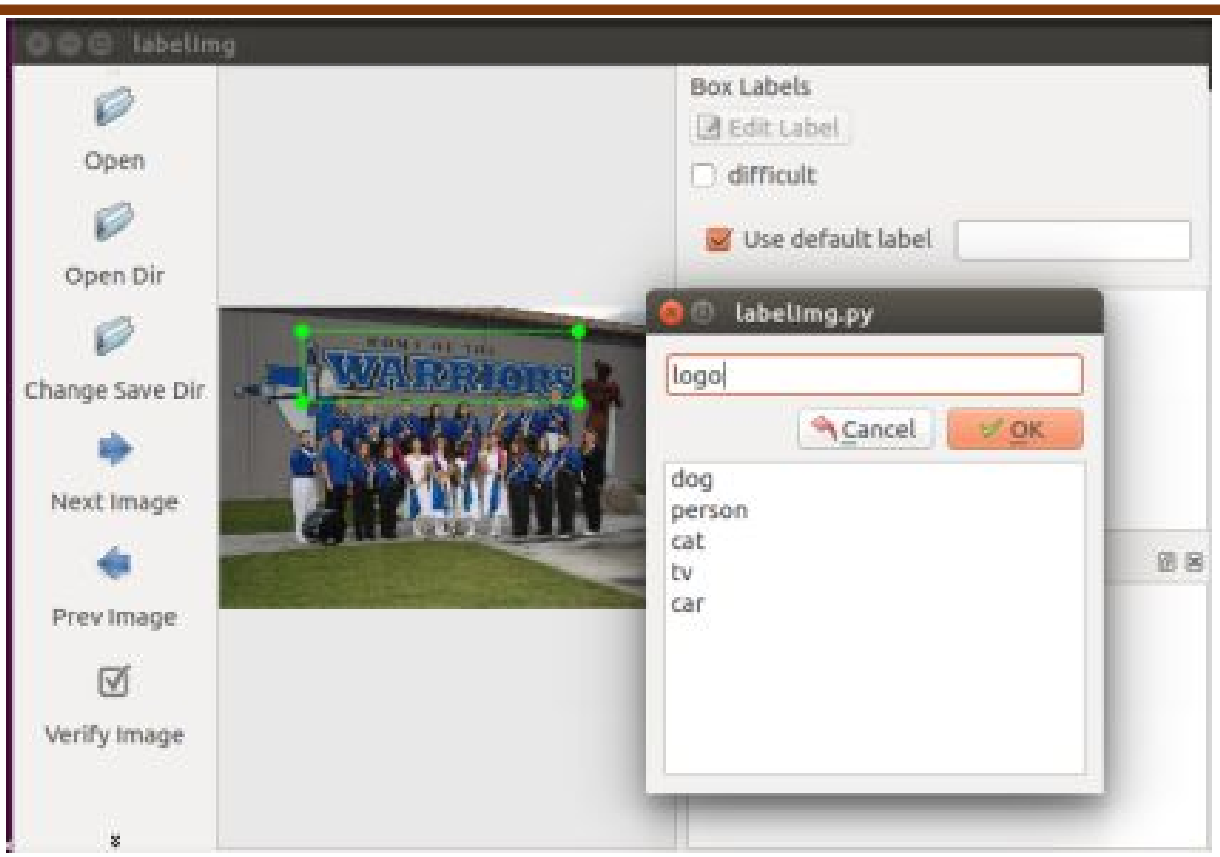


Fig. 5.2.2.5.9 LabelImg

CHAPTER 6

PSEUDO CODE

Pseudo Code uses the structural conventions of a normal programming language but the intention is for human reading rather than machine reading. Omission of details that are essential for machine understanding of the algorithm such as variable declarations, specific system code and some sub-routines is pseudocode. Augmentation of programming language is done with natural language description details where convenient or with compact mathematical notation. The purpose of using pseudocode is that it is easier for people to understand than conventional programming language code and that the key principles of an algorithm are efficiently and environment independently described. Pseudocode is commonly used in textbooks and scientific publications that are documenting various algorithms and also in plan of computer program development, for sketching the structure of the program before the actual coding takes place.

Steps:

Installing

To install the dependencies, run
bash
pip install -r requirements.txt
And for the GPU to work, make sure you've got the drivers installed beforehand (CUDA).
It has been tested to work with Python 2.7.13 and 3.5.3.

Detection

Grab the pretrained weights of yolo3 from https://pjreddie.com/media/files/yolov3.weights.
python yolo3_one_file_to_detect_them_all.py -w yolo3.weights -i testing_image1.jpg

Training

1.Data preparation : We collected images of 5 buildings of interest and annotated using labelImg which helps us to annotate each image and creates a corresponding XML file that contains the label for the building that we are annotating and the coordinates of the bounding box around the building which we are annotating.

Organize the dataset into 4 folders:

train_image_folder <= the folder that contains the train images.

train_annot_folder <= the folder that contains the train annotations in VOC format.

valid_image_folder <= the folder that contains the validation images.

valid_annot_folder <= the folder that contains the validation annotations in VOC format.

2. Edit the configuration file

The configuration file **(config.json)** is a json file, which looks like this:

```
{
    "model" : {
        "min_input_size":       288,
        "max_input_size":       448,
        "anchors":              [55,69, 75,234, 133,240, 136,129, 142,363, 203,290, 228,184, 285,359, 341,260],
        "labels":               ["automobile", "business", "biotech", "heritage", "is", "mechanical"]
    },

    "train": {
  "train_image_folder": "/home/userd522/Desktop/keras-yolo3/keras-yolo3/images/",
"train_annot_folder":"/home/userd522/Desktop/keras-yolo3/keras-yolo3/annotations/",
        "cache_name":           "departments.pkl",
```

```
        "train_times":      8,
        "batch_size":       4,
        "learning_rate":    1e-4,
        "nb_epochs":        100,
        "warmup_epochs":    3,
        "ignore_thresh":    0.5,
        "gpus":             "0",
        "grid_scales":      [1,1,1],
        "obj_scale":        5,
        "noobj_scale":      1,
        "xywh_scale":       1,
        "class_scale":      1,
        "tensorboard_dir":  "logs",
        "saved_weights_name": "departments.h5",
        "debug":            true
    },
    "valid": {
        "valid_image_folder":   "",
        "valid_annot_folder":   "",
        "cache_name":       "",
        "valid_times":      1
    }
}
```

3. Generate anchors for your dataset (optional)

python gen_anchors.py -c config.json

Copy the generated anchors printed on the terminal to the anchors setting in config.json

4. Start the training process

python train.py -c config.json

By the end of this process, the code will write the weights of the best model to file departments.h5

5. Perform detection using trained weights on image, set of images, video, or webcam

python predict.py -c config.json -i /path/to/image/or/video

Evaluation

python evaluate.py -c config.json

6.1 yolo3_one_file_to_detect_them_all.py

```
import argparse
import os
import numpy as np
from keras.layers import Conv2D, Input, BatchNormalization, LeakyReLU,
ZeroPadding2D, UpSampling2D
from keras.layers.merge import add, concatenate
from keras.models import Model
import struct
import cv2
import sys
sys.path.append("numpy_path")

np.set_printoptions(threshold=sys.maxsize)
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"]="0"
argparser = argparse.ArgumentParser(
    description='test yolov3 network with coco weights')
argparser.add_argument(
    '-w',
    '--weights',
    help='path to weights file')

argparser.add_argument(
    '-i',
    '--image',
    help='path to image file')
```

```
class WeightReader:

    def __init__(self, weight_file):
        with open(weight_file, 'rb') as w_f:
            major,    = struct.unpack('i', w_f.read(4))
            minor,    = struct.unpack('i', w_f.read(4))
            revision, = struct.unpack('i', w_f.read(4))

            if (major*10 + minor) >= 2 and major < 1000 and minor < 1000:
                w_f.read(8)
            else:
                w_f.read(4)

            transpose = (major > 1000) or (minor > 1000)

            binary = w_f.read()

        self.offset = 0
      self.all_weights = np.frombuffer(binary, dtype='float32')

    def read_bytes(self, size):
      self.offset = self.offset + size
      return self.all_weights[self.offset-size:self.offset]

    def load_weights(self, model):
        for i in range(106):
            try:
                conv_layer = model.get_layer('conv_' + str(i))
                print("loading weights of convolution #" + str(i))

                if i not in [81, 93, 105]:
                    norm_layer = model.get_layer('bnorm_' + str(i))

                    size = np.prod(norm_layer.get_weights()[0].shape)

                    beta  = self.read_bytes(size) # bias
                    gamma = self.read_bytes(size) # scale
                    mean  = self.read_bytes(size) # mean
                    var   = self.read_bytes(size) # variance

                    weights = norm_layer.set_weights([gamma, beta, mean, var])
                if len(conv_layer.get_weights()) > 1:
                    bias   = self.read_bytes(np.prod(conv_layer.get_weights()[1].shape))
                    kernel = self.read_bytes(np.prod(conv_layer.get_weights()[0].shape))

                    kernel kernel.reshape(list(reversed(conv_layer.get_weights()[0].shape)))
```

```
                kernel = kernel.transpose([2,3,1,0])
                conv_layer.set_weights([kernel, bias])
            else:
                kernel = self.read_bytes(np.prod(conv_layer.get_weights()[0].shape))
                kernel =kernel.reshape(list(reversed(conv_layer.get_weights()[0].shape)))

        kernel = kernel.transpose([2,3,1,0])
                conv_layer.set_weights([kernel])
         except ValueError:
            print("no convolution #" + str(i))

    def reset(self):
        self.offset = 0
class BoundBox:
    def __init__(self, xmin, ymin, xmax, ymax, objness = None, classes = None):
        self.xmin = xmin
        self.ymin = ymin
        self.xmax = xmax
        self.ymax = ymax
        self.objness = objness
        self.classes = classes
        self.label = -1
        self.score = -1

    def get_label(self):
        if self.label == -1:
            self.label = np.argmax(self.classes)

        return self.label
    def get_score(self):
        if self.score == -1:
            self.score = self.classes[self.get_label()]

         return self.score
    def _conv_block(inp, convs, skip=True):
      x = inp
      count = 0

      for conv in convs:
          if count == (len(convs) - 2) and skip:
              skip_connection = x
          count += 1
           if conv['stride'] > 1: x = ZeroPadding2D(((1,0),(1,0)))(x) # peculiar padding as
                                                                    darknet prefer left and top
        x = Conv2D(conv['filter'],
                   conv['kernel'],
                   strides=conv['stride'],
                   padding='valid' if conv['stride'] > 1 else 'same', #peculiar padding as
darknet prefer left and top
```

```
                name='conv_' + str(conv['layer_idx']),
                use_bias=False if conv['bnorm'] else True)(x)
        if conv['bnorm']: x = BatchNormalization(epsilon=0.001, name='bnorm_' +
str(conv['layer_idx']))(x)
        if conv['leaky']: x = LeakyReLU(alpha=0.1, name='leaky_' +
str(conv['layer_idx']))(x)

    return add([skip_connection, x]) if skip else x
def _interval_overlap(interval_a, interval_b):
    x1, x2 = interval_a
    x3, x4 = interval_b

    if x3 < x1:
        if x4 < x1:
            return 0
        else:
            return min(x2,x4) - x1
    else:
        if x2 < x3:
             return 0
        else:
            return min(x2,x4) - x3
def _sigmoid(x):
    return 1. / (1. + np.exp(-x))

def bbox_iou(box1, box2):
    intersect_w = _interval_overlap([box1.xmin, box1.xmax], [box2.xmin,
box2.xmax])
    intersect_h = _interval_overlap([box1.ymin, box1.ymax], [box2.ymin, box2.ymax])

    intersect = intersect_w * intersect_h

    w1, h1 = box1.xmax-box1.xmin, box1.ymax-box1.ymin
    w2, h2 = box2.xmax-box2.xmin, box2.ymax-box2.ymin

    union = w1*h1 + w2*h2 - intersect

    return float(intersect) / union
def make_yolov3_model():
    input_image = Input(shape=(None, None, 3))

    # Layer  0 => 4
    x = _conv_block(input_image, [{'filter': 32, 'kernel': 3, 'stride': 1, 'bnorm': True,
'leaky': True, 'layer_idx': 0},
                                  {'filter': 64, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True,
'layer_idx': 1},
                                  {'filter': 32, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 2},
```

```
                                {'filter': 64, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 3}])

    # Layer  5 => 8
   x = _conv_block(x, [{'filter': 128, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True,
'layer_idx': 5},
                        {'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 6},

    {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 7}])

    # Layer  9 => 11
    x = _conv_block(x, [{'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 9},
                        {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 10}])

    # Layer 12 => 15
    x = _conv_block(x, [{'filter': 256, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True,
'layer_idx': 12},
                        {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 13},
                        {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 14}])

    # Layer 16 => 36
 for i in range(7):
        x = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
True, 'layer_idx': 16+i*3},
                            {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 17+i*3}])

    skip_36 = x

    # Layer 37 => 40
  x = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True,
'layer_idx': 37},
                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 38},
                        {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 39}])

    # Layer 41 => 61
 for i in range(7):
        x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
True, 'layer_idx': 41+i*3},
                            {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 42+i*3}])
```

```
    skip_61 = x

    # Layer 62 => 65
  x = _conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True,
'layer_idx': 62},
                      {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 63},
                      {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 64}])

    # Layer 66 => 74
  for i in range(3):
        x = _conv_block(x, [{'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
True, 'layer_idx': 66+i*3},
                            {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 67+i*3}])

    # Layer 75 => 79
    x = _conv_block(x, [{'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 75},
                      {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 76},
                      {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 77},
                      {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 78},
                      {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 79}], skip=False)

    # Layer 80 => 82
    yolo_82 = _conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True,
'leaky': True,  'layer_idx': 80},
                            {'filter':  255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False,
'layer_idx': 81}], skip=False)

    # Layer 83 => 86
    x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 84}], skip=False)
    x = UpSampling2D(2)(x)
    x = concatenate([x, skip_61])

  # Layer 87 => 91
    x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 87},
                      {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 88},
                      {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 89},
```

```
                    {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 90},
                    {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 91}], skip=False)

   # Layer 92 => 94
   yolo_94 = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True,
'leaky': True,  'layer_idx': 92},
                         {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False,
'layer_idx': 93}], skip=False)


  # Layer 95 => 98
   x = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 96}], skip=False)
   x = UpSampling2D(2)(x)
   x = concatenate([x, skip_36])

   # Layer 99 => 106
   yolo_106 = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,
'leaky': True,  'layer_idx': 99},
                         {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,
'layer_idx': 100},
                         {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,  'leaky': True,
'layer_idx': 101},
                         {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,
'layer_idx': 102},
                         {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,  'leaky': True,
'layer_idx': 103},
                         {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,
'layer_idx': 104},
                         {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False,
'layer_idx': 105}], skip=False)

   model = Model(input_image, [yolo_82, yolo_94, yolo_106])
   return model

def preprocess_input(image, net_h, net_w):
   new_h, new_w, _ = image.shape

   # determine the new size of the image
   if (float(net_w)/new_w) < (float(net_h)/new_h):
      new_h = (new_h * net_w)/new_w
      new_w = net_w
   else:
      new_w = (new_w * net_h)/new_h
      new_h = net_h

   # resize the image to the new size
```

```
    resized = cv2.resize(image[:,:,::-1]/255., (int(new_w), int(new_h)))

    # embed the image into the standard letter box
    new_image = np.ones((net_h, net_w, 3)) * 0.5
    new_image[int((net_h-new_h)//2):int((net_h+new_h)//2),
int((net_w-new_w)//2):int((net_w+new_w)//2), :] = resized
    new_image = np.expand_dims(new_image, 0)

    return new_image

def decode_netout(netout, anchors, obj_thresh, nms_thresh, net_h, net_w):
    grid_h, grid_w = netout.shape[:2]

    nb_box = 3
    netout = netout.reshape((grid_h, grid_w, nb_box, -1))
    nb_class = netout.shape[-1] - 5

    boxes = []

    netout[..., :2]  = _sigmoid(netout[..., :2])
    netout[..., 4:]  = _sigmoid(netout[..., 4:])
    netout[..., 5:]  = netout[..., 4][..., np.newaxis] * netout[..., 5:]
    netout[..., 5:] *= netout[..., 5:] > obj_thresh

    for i in range(grid_h*grid_w):
        row = i / grid_w
        col = i % grid_w

        for b in range(nb_box):
            # 4th element is objectness score
            objectness = netout[int(row)][int(col)][b][4]
            #objectness = netout[..., :4]

            if(objectness.all() <= obj_thresh): continue

            # first 4 elements are x, y, w, and h
            x, y, w, h = netout[int(row)][int(col)][b][:4]

            x = (col + x) / grid_w # center position, unit: image width
            y = (row + y) / grid_h # center position, unit: image height
            w = anchors[2 * b + 0] * np.exp(w) / net_w # unit: image width
            h = anchors[2 * b + 1] * np.exp(h) / net_h # unit: image height

            # last elements are class probabilities
            classes = netout[int(row)][col][b][5:]

            box = BoundBox(x-w/2, y-h/2, x+w/2, y+h/2, objectness, classes)
            #box = BoundBox(x-w/2, y-h/2, x+w/2, y+h/2, None, classes)
```

```
            boxes.append(box)

    return boxes
def correct_yolo_boxes(boxes, image_h, image_w, net_h, net_w):
    if (float(net_w)/image_w) < (float(net_h)/image_h):
        new_w = net_w
        new_h = (image_h*net_w)/image_w
    else:
        new_h = net_w
        new_w = (image_w*net_h)/image_h

    for i in range(len(boxes)):
        x_offset, x_scale = (net_w - new_w)/2./net_w, float(new_w)/net_w

        y_offset, y_scale = (net_h - new_h)/2./net_h, float(new_h)/net_h

        boxes[i].xmin = int((boxes[i].xmin - x_offset) / x_scale * image_w)
        boxes[i].xmax = int((boxes[i].xmax - x_offset) / x_scale * image_w)
        boxes[i].ymin = int((boxes[i].ymin - y_offset) / y_scale * image_h)
        boxes[i].ymax = int((boxes[i].ymax - y_offset) / y_scale * image_h)

def do_nms(boxes, nms_thresh):
    if len(boxes) > 0:
        nb_class = len(boxes[0].classes)
    else:
        return

    for c in range(nb_class):
        sorted_indices = np.argsort([-box.classes[c] for box in boxes])

        for i in range(len(sorted_indices)):
            index_i = sorted_indices[i]

            if boxes[index_i].classes[c] == 0: continue

            for j in range(i+1, len(sorted_indices)):
                index_j = sorted_indices[j]

                if bbox_iou(boxes[index_i], boxes[index_j]) >= nms_thresh:
                    boxes[index_j].classes[c] = 0

def draw_boxes(image, boxes, labels, obj_thresh):
    for box in boxes:
        label_str = ''
        label = -1

        for i in range(len(labels)):
            if box.classes[i] > obj_thresh:
                label_str += labels[i]
```

```
                label = i
                print(labels[i] + ': ' + str(box.classes[i]*100) + '%')

        if label >= 0:
            cv2.rectangle(image, (box.xmin,box.ymin), (box.xmax,box.ymax),
(0,255,0),3)
            cv2.putText(image,
                        label_str + ' ' + str(box.get_score()),
                        (box.xmin, box.ymin - 13),
                        cv2.FONT_HERSHEY_SIMPLEX,
                        1e-3 * image.shape[0],
                        (0,255,0), 2)

    return image

def _main_(args):
    weights_path = args.weights
    image_path   = args.image

    # set some parameters
    net_h, net_w = 416, 416
    obj_thresh, nms_thresh = 0.5, 0.45
    anchors = [[116,90,  156,198,  373,326],  [30,61, 62,45,  59,119], [10,13,  16,30,
33,23]]
    labels = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus", "train",
"truck", \
              "boat", "traffic light", "fire hydrant", "stop sign", "parking meter", "bench", \
              "bird", "cat", "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra",
"giraffe", \
              "backpack", "umbrella", "handbag", "tie", "suitcase", "frisbee", "skis",
"snowboard", \
              "sports ball", "kite", "baseball bat", "baseball glove", "skateboard",
"surfboard", \
              "tennis racket", "bottle", "wine glass", "cup", "fork", "knife", "spoon",
"bowl", "banana", \
              "apple", "sandwich", "orange", "broccoli", "carrot", "hot dog", "pizza",
"donut", "cake", \
              "chair", "sofa", "pottedplant", "bed", "diningtable", "toilet", "tvmonitor",
"laptop", "mouse", \
              "remote", "keyboard", "cell phone", "microwave", "oven", "toaster", "sink",
"refrigerator", \
              "book", "clock", "vase", "scissors", "teddy bear", "hair drier", "toothbrush"]

    # make the yolov3 model to predict 80 classes on COCO
    yolov3 = make_yolov3_model()

    # load the weights trained on COCO into the model
    weight_reader = WeightReader(weights_path)
    weight_reader.load_weights(yolov3)
```

```
    # preprocess the image
    image = cv2.imread(image_path)
    image_h, image_w, _ = image.shape
    new_image = preprocess_input(image, net_h, net_w)

    # run the prediction
    yolos = yolov3.predict(new_image)
    boxes = []

    for i in range(len(yolos)):
        # decode the output of the network
        boxes += decode_netout(yolos[i][0], anchors[i], obj_thresh, nms_thresh, net_h,
net_w)


  # correct the sizes of the bounding boxes
    correct_yolo_boxes(boxes, image_h, image_w, net_h, net_w)

    # suppress non-maximal boxes
    do_nms(boxes, nms_thresh)

    # draw bounding boxes on the image using labels
    draw_boxes(image, boxes, labels, obj_thresh)

    # write the image with bounding boxes to file
    cv2.imwrite(image_path[:-4] + '_detected' + image_path[-4:],
(image).astype('uint8'))

if __name__ == '__main__':
    args = argparser.parse_args()
     _main_(args)
```

6.2 predict.py

```
import os
import argparse
import json
import cv2
from utils.utils import get_yolo_boxes, makedirs
from utils.bbox import draw_boxes
from keras.models import load_model
from tqdm import tqdm
import numpy as np
```

```
def _main_(args):
    config_path  = args.conf
    input_path   = args.input
    output_path  = args.output

    with open(config_path) as config_buffer:
        config = json.load(config_buffer)

    makedirs(output_path)

 #   Set some parameter

 net_h, net_w = 416, 416 # a multiple of 32, the smaller the faster
   obj_thresh, nms_thresh = 0.5, 0.45

 #   Load the model

 os.environ['CUDA_VISIBLE_DEVICES'] = config['train']['gpus']
   infer_model = load_model(config['train']['saved_weights_name'])

 #   Predict bounding boxes

 if 'webcam' in input_path: # do detection on the first webcam
       video_reader = cv2.VideoCapture(0)

       # the main loop
       batch_size  = 1
       images      = []
       while True:
          ret_val, image = video_reader.read()
          if ret_val == True: images += [image]

          if (len(images)==batch_size) or (ret_val==False and len(images)>0):
             batch_boxes = get_yolo_boxes(infer_model, images, net_h, net_w,
config['model']['anchors'], obj_thresh, nms_thresh)

             for i in range(len(images)):
                draw_boxes(images[i], batch_boxes[i], config['model']['labels'],
obj_thresh)
                cv2.imshow('video with bboxes', images[i])
             images = []
          if cv2.waitKey(1) == 27:
             break  # esc to quit
       cv2.destroyAllWindows()
    elif input_path[-4:] == '.mp4': # do detection on a video
       video_out = output_path + input_path.split('/')[-1]
       video_reader = cv2.VideoCapture(input_path)

       nb_frames = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))
```

```
            frame_h = int(video_reader.get(cv2.CAP_PROP_FRAME_HEIGHT))
            frame_w = int(video_reader.get(cv2.CAP_PROP_FRAME_WIDTH))

        video_writer = cv2.VideoWriter(video_out,
                                       cv2.VideoWriter_fourcc(*'MPEG'),
                                       50.0,
                                       (frame_w, frame_h))
            # the main loop
            batch_size  = 1
            images      = []
            start_point = 0 #%
            show_window = False
            for i in tqdm(range(nb_frames)):
                _, image = video_reader.read()

                if (float(i+1)/nb_frames) > start_point/100.:
                    images += [image]

                    if (i%batch_size == 0) or (i == (nb_frames-1) and len(images) > 0):
                        # predict the bounding boxes
                        batch_boxes = get_yolo_boxes(infer_model, images, net_h, net_w,
config['model']['anchors'], obj_thresh, nms_thresh)

                        for i in range(len(images)):
                            # draw bounding boxes on the image using labels
                            draw_boxes(images[i], batch_boxes[i], config['model']['labels'],
obj_thresh)

                            # show the video with detection bounding boxes
                            if show_window: cv2.imshow('video with bboxes', images[i])

                            # write result to the output video
                            video_writer.write(images[i])
                        images = []
                    if show_window and cv2.waitKey(1) == 27: break  # esc to quit

            if show_window: cv2.destroyAllWindows()
            video_reader.release()
            video_writer.release()
        else: # do detection on an image or a set of images
            image_paths = []

            if os.path.isdir(input_path):
                for inp_file in os.listdir(input_path):
                    image_paths += [input_path + inp_file]
            else:
                image_paths += [input_path]
```

```
            image_paths = [inp_file for inp_file in image_paths if (inp_file[-4:] in ['.jpg',
'.png', 'JPEG'])]

            # the main loop
            for image_path in image_paths:
                image = cv2.imread(image_path)
                print(image_path)

                # predict the bounding boxes
                boxes = get_yolo_boxes(infer_model, [image], net_h, net_w,
config['model']['anchors'], obj_thresh, nms_thresh)[0]

                # draw bounding boxes on the image using labels
                draw_boxes(image, boxes, config['model']['labels'], obj_thresh)

                # write the image with bounding boxes to file
                cv2.imwrite(output_path + image_path.split('/')[-1], np.uint8(image))

    if __name__ == '__main__':

        argparser = argparse.ArgumentParser(description='Predict with a trained yolo
model')
        argparser.add_argument('-c', '--conf', help='path to configuration file')
        argparser.add_argument('-i', '--input', help='path to an image, a directory of
images, a video, or webcam')
        argparser.add_argument('-o', '--output', default='output/', help='path to output
directory')

        args = argparser.parse_args()
      _main_(args)
```

6.3 train.py

```
import argparse
import os
import numpy as np
import json
from voc import parse_voc_annotation
from yolo import create_yolov3_model, dummy_loss
from generator import BatchGenerator
from utils.utils import normalize, evaluate, makedirs
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras.optimizers import Adam
from callbacks import CustomModelCheckpoint, CustomTensorBoard
from utils.multi_gpu_model import multi_gpu_model
import tensorflow as tf
```

```
import keras
from keras.models import load_model


config = tf.compat.v1.ConfigProto(
    gpu_options = tf.compat.v1.GPUOptions(per_process_gpu_memory_fraction=0.9)
    # device_count = {'GPU': 1}
)
config.gpu_options.allow_growth = True
session = tf.compat.v1.Session(config=config)
tf.compat.v1.keras.backend.set_session(session)

def create_training_instances(
    train_annot_folder,
    train_image_folder,
    train_cache,
    valid_annot_folder,

    valid_image_folder,
    valid_cache,
    labels,
):
    # parse annotations of the training set
    train_ints, train_labels = parse_voc_annotation(train_annot_folder,
train_image_folder, train_cache, labels)

    # parse annotations of the validation set, if any, otherwise split the training set
    if os.path.exists(valid_annot_folder):
        valid_ints, valid_labels = parse_voc_annotation(valid_annot_folder,
valid_image_folder, valid_cache, labels)
    else:
        print("valid_annot_folder not exists. Spliting the trainining set.")

        train_valid_split = int(0.8*len(train_ints))
        np.random.seed(0)
        np.random.shuffle(train_ints)
        np.random.seed()

        valid_ints = train_ints[train_valid_split:]
        train_ints = train_ints[:train_valid_split]

    # compare the seen labels with the given labels in config.json
```

```
    if len(labels) > 0:
        overlap_labels = set(labels).intersection(set(train_labels.keys()))

        print('Seen labels: \t'  + str(train_labels) + '\n')
        print('Given labels: \t' + str(labels))

        # return None, None, None if some given label is not in the dataset
        if len(overlap_labels) < len(labels):
            print('Some labels have no annotations! Please revise the list of labels in the
config.json.')
            return None, None, None
    else:
        print('No labels are provided. Train on all seen labels.')
        print(train_labels)
        labels = train_labels.keys()

    max_box_per_image = max([len(inst['object']) for inst in (train_ints + valid_ints)])


  return train_ints, valid_ints, sorted(labels), max_box_per_image

def create_callbacks(saved_weights_name, tensorboard_logs, model_to_save):
    makedirs(tensorboard_logs)

    early_stop = EarlyStopping(
        monitor     = 'loss',
        min_delta   = 0.01,
        patience    = 7,
        mode        = 'min',
        verbose     = 1
    )
    checkpoint = CustomModelCheckpoint(
        model_to_save   = model_to_save,
        filepath        = saved_weights_name,# + '{epoch:02d}.h5',
        monitor         = 'loss',
        verbose         = 1,
        save_best_only  = True,
        mode            = 'min',
        period          = 1
    )
    reduce_on_plateau = ReduceLROnPlateau(
        monitor  = 'loss',
```

```
            factor   = 0.1,
            patience = 2,
            verbose  = 1,
            mode     = 'min',
            epsilon  = 0.01,
            cooldown = 0,
            min_lr   = 0
        )
        tensorboard = CustomTensorBoard(
            log_dir              = tensorboard_logs,
            write_graph          = True,
            write_images         = True,
        )
        return [early_stop, checkpoint, reduce_on_plateau, tensorboard]

    def create_model(
        nb_class,
        anchors,
        max_box_per_image,

        max_grid, batch_size,
        warmup_batches,
        ignore_thresh,
        multi_gpu,
        saved_weights_name,
        lr,
        grid_scales,
        obj_scale,
        noobj_scale,
        xywh_scale,
        class_scale
    ):
        if multi_gpu > 1:
            with tf.device('/cpu:0'):
                template_model, infer_model = create_yolov3_model(
                    nb_class          = nb_class,
                    anchors           = anchors,
                    max_box_per_image   = max_box_per_image,
                    max_grid          = max_grid,
                    batch_size        = batch_size//multi_gpu,
                    warmup_batches      = warmup_batches,
                    ignore_thresh       = ignore_thresh,
```

```
                grid_scales        = grid_scales,
                obj_scale          = obj_scale,
                noobj_scale        = noobj_scale,
                xywh_scale         = xywh_scale,
                class_scale        = class_scale
            )
        else:
            template_model, infer_model = create_yolov3_model(
                nb_class           = nb_class,
                anchors            = anchors,
                max_box_per_image  = max_box_per_image,
                max_grid           = max_grid,
                batch_size         = batch_size,
                warmup_batches     = warmup_batches,
                ignore_thresh      = ignore_thresh,
                grid_scales        = grid_scales,
                obj_scale          = obj_scale,
                noobj_scale        = noobj_scale,
                xywh_scale         = xywh_scale,
                class_scale        = class_scale

    )

        # load the pretrained weight if exists, otherwise load the backend weight only
        if os.path.exists(saved_weights_name):
            print("\nLoading pretrained weights.\n")
            template_model.load_weights(saved_weights_name)
        else:
            template_model.load_weights("backend.h5", by_name=True)

        if multi_gpu > 1:
            train_model = multi_gpu_model(template_model, gpus=multi_gpu)
        else:
            train_model = template_model

        optimizer = Adam(lr=lr, clipnorm=0.001)
        train_model.compile(loss=dummy_loss, optimizer=optimizer)

        return train_model, infer_model

    def _main_(args):
        config_path = args.conf
```

```
    with open(config_path) as config_buffer:
        config = json.loads(config_buffer.read())

    #   Parse the annotations

    train_ints, valid_ints, labels, max_box_per_image = create_training_instances(
        config['train']['train_annot_folder'],
        config['train']['train_image_folder'],
        config['train']['cache_name'],
        config['valid']['valid_annot_folder'],
        config['valid']['valid_image_folder'],
        config['valid']['cache_name'],
        config['model']['labels']
    )
    print('\nTraining on: \t' + str(labels) + '\n')


    #   Create the generators

    train_generator = BatchGenerator(

        instances           = train_ints,
        anchors             = config['model']['anchors'],
        labels              = labels,
        downsample          = 32, # ratio between network input's size and network
output's size, 32 for YOLOv3
        max_box_per_image   = max_box_per_image,
        batch_size          = config['train']['batch_size'],
        min_net_size        = config['model']['min_input_size'],
        max_net_size        = config['model']['max_input_size'],
        shuffle             = True,
        jitter              = 0.3,
        norm                = normalize
    )

    valid_generator = BatchGenerator(
        instances           = valid_ints,
        anchors             = config['model']['anchors'],
        labels              = labels,
        downsample          = 32, # ratio between network input's size and network
output's size, 32 for YOLOv3
```

```
        max_box_per_image   = max_box_per_image,
        batch_size          = config['train']['batch_size'],
        min_net_size        = config['model']['min_input_size'],
        max_net_size        = config['model']['max_input_size'],
        shuffle             = True,
        jitter              = 0.0,
        norm                = normalize
    )


    #   Create the model

    if os.path.exists(config['train']['saved_weights_name']):
        config['train']['warmup_epochs'] = 0
    warmup_batches = config['train']['warmup_epochs'] *
(config['train']['train_times']*len(train_generator))

    os.environ['CUDA_VISIBLE_DEVICES'] = config['train']['gpus']
    multi_gpu = len(config['train']['gpus'].split(','))

    train_model, infer_model = create_model(
        nb_class            = len(labels),

        anchors             = config['model']['anchors'],
        max_box_per_image   = max_box_per_image,
        max_grid            = [config['model']['max_input_size'],
config['model']['max_input_size']],
        batch_size          = config['train']['batch_size'],
        warmup_batches      = warmup_batches,
        ignore_thresh       = config['train']['ignore_thresh'],
        multi_gpu           = multi_gpu,
        saved_weights_name  = config['train']['saved_weights_name'],
        lr                  = config['train']['learning_rate'],
        grid_scales         = config['train']['grid_scales'],
        obj_scale           = config['train']['obj_scale'],
        noobj_scale         = config['train']['noobj_scale'],
        xywh_scale          = config['train']['xywh_scale'],
        class_scale         = config['train']['class_scale'],
    )

    #   Kick off the training
```

```
    callbacks = create_callbacks(config['train']['saved_weights_name'],
config['train']['tensorboard_dir'], infer_model)

    train_model.fit_generator(
        generator        = train_generator,
        steps_per_epoch  = len(train_generator) * config['train']['train_times'],
        epochs           = config['train']['nb_epochs'] + config['train']['warmup_epochs'],
        verbose          = 2 if config['train']['debug'] else 1,
        callbacks        = callbacks,
        workers          = 4,
        max_queue_size   = 8
    )
    # make a GPU version of infer_model for evaluation
    if multi_gpu > 1:
        infer_model = load_model(config['train']['saved_weights_name'])

    #   Run the evaluation

    # compute mAP for all the classes
    average_precisions = evaluate(infer_model, valid_generator)

    # print the score
    for label, average_precision in average_precisions.items():

    print(labels[label] + ': {:.4f}'.format(average_precision))
    print('mAP: {:.4f}'.format(sum(average_precisions.values()) /
len(average_precisions)))

if __name__ == '__main__':
    argparser = argparse.ArgumentParser(description='train and evaluate YOLO_v3
model on any dataset')
    argparser.add_argument('-c', '--conf', help='path to configuration file')

    args = argparser.parse_args()
    _main_(args)
```

6.4 evaluate.py

```
import argparse
import os
import numpy as np
```

```
import json
from voc import parse_voc_annotation
from yolo import create_yolov3_model
from generator import BatchGenerator
from utils.utils import normalize, evaluate
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.optimizers import Adam
from keras.models import load_model

def _main_(args):
    config_path = args.conf

    with open(config_path) as config_buffer:
        config = json.loads(config_buffer.read())

    #   Create the validation generator
        valid_ints, labels = parse_voc_annotation(
        config['valid']['valid_annot_folder'],
        config['valid']['valid_image_folder'],
        config['valid']['cache_name'],
        config['model']['labels']
    )

    labels = labels.keys() if len(config['model']['labels']) == 0 else
config['model']['labels']
    labels = sorted(labels)

    valid_generator = BatchGenerator(
        instances           = valid_ints,
        anchors             = config['model']['anchors'],
        labels              = labels,
        downsample          = 32, # ratio between network input's size and network
output's size, 32 for YOLOv3
        max_box_per_image   = 0,
        batch_size          = config['train']['batch_size'],
        min_net_size        = config['model']['min_input_size'],
        max_net_size        = config['model']['max_input_size'],
        shuffle             = True,
        jitter              = 0.0,
        norm                = normalize
    )
```

```
        #   Load the model and do evaluation
        os.environ['CUDA_VISIBLE_DEVICES'] = config['train']['gpus']

        infer_model = load_model(config['train']['saved_weights_name'])

        # compute mAP for all the classes
        average_precisions = evaluate(infer_model, valid_generator)

        # print the score
        for label, average_precision in average_precisions.items():

 print(labels[label] + ': {:.4f}'.format(average_precision))
    print('mAP: {:.4f}'.format(sum(average_precisions.values()) /
len(average_precisions)))

if __name__ == '__main__':
    argparser = argparse.ArgumentParser(description='Evaluate YOLO_v3 model on any
dataset')
    argparser.add_argument('-c', '--conf', help='path to configuration file')

    args = argparser.parse_args()
      _main_(args)
```

CHAPTER 7

TESTING

Software testing is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified requirements or not and to identify the defects to ensure that the product is defect-free in order to produce the quality product.

The discovery of error is the purpose of testing. Testing provides a way to check the functionality of the components, sub assemblies, assemblies and a finished project. Exercise of software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner is the process of testing. The system has been verified and validated by running the test data and live data.

7.1 Test Cases

Table 7.1 Test cases for model

Test Case ID	Description	Input	Expected Output	Actual Output	Result
1	Building with High Exposure to light	Automobile Building	Building label “automobile”	Building label “automobile”	Pass
2	Building with low exposure to light	Business	Building label “business”	Building label “business”	Pass
3	Building with low exposure of light	Mechanical	Building label “mechanical”	Building Label “mechanical”	Pass

4	Building with high exposure of light	Biotech	Building label "Biotech"	Building label "Biotech"	Pass
5	Building with high exposure of light	ISE	Building label "ISE"	Building label "ISE"	Pass
6	Building with low exposure of light	Heritage	Building label "Heritage"	Building label "heritage"	Pass
7	Building focused with less crowd people	Heritage	Building label "Heritage"	Building label "Heritage"	Pass
8	Building focused with more crowded people	Mechanical	Building label "Mechanical"	Building label "Mechanical"	Pass
9	Building focused with different angle	Automobile	Building Label "Automobile"	Building label "automobile"	Pass
10	Building focused with different angle	ISE	Building label "ISE"	Building label "ISE"	Pass

CHAPTER 8

RESULTS

The model detects the 5 college buildings of Dayananda Sagar College of Engineering that we took under consideration accurately and the output for any image as input or any video as input is stored in the output folder that we specify and the model can also detect the 5 college buildings of Dayananda Sagar College of Engineering in real-time and that's the main reason we went for YOLO algorithm for object detection as the FPS(Frames per Second) is 45 which enables object detection at real-time.

The package.json in here, is a json file containing all the necessary parameters such as the folder where the images are found, folder where the annotations are found, no of epochs, batch size etc, this file is passed as an argument while training which is then parsed and used to initialise the necessary parameters while training.

8.1 Results using YOLO (Detection of Buildings in images)

Fig. 8.1.1 Identification of ISE Department

Fig. 8.1.2 Identification of Business Block

Fig 8.1.3 Identification of Biotech Department

Fig 8.1.4 Identification of Automobile Department

Fig. 8.1.5 Identification of Automobile Department (from a different angle)

Fig. 8.1.6 Identification of Mechanical Department

CHAPTER 9

CONCLUSION AND FUTURE SCOPE

The difficulty for a beginner to the field of deep learning to know what type of network to use is high. There are so many types of networks to choose from and new methods being published and discussions are happening every day. Making things worse, most neural networks are flexible enough that they work and make a prediction even when used with the wrong type of data or prediction problem.

Convolutional Neural Networks were designed to map image data to an output variable. They have proven so effective that they are the go to method for any type of prediction problem which involves image data as an input.

The R-CNN family of techniques primarily use regions to localize the objects within the image. The network does not look at the entire image, only at the parts of the images which have a higher chance of containing an object.

The YOLO framework (You Only Look Once) on the other hand, deals with object detection in a different way. It takes the entire image in a single instance and predicts the bounding box coordinates and class probabilities for these boxes. The biggest advantage of using YOLO is its superb speed – it's incredibly fast and can process 45 frames per second. YOLO also understands generalized object representation. Hence, we identified that this would be the best algorithm for us as per our requirement.

Chapter 10

REFERENCES

Reference Papers:

[1] Zhong-Qiu Zhao, Member, IEEE, Peng Zheng, Shou-tao Xu, and Xindong Wu, Fellow, IEEE:Object Detection with Deep Learning: A Review, April 2019

[2] Xinyi Zhou, Wei Gong, WenLong Fu, Fengtong Du: Application of deep learning in object detection, 2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)

[3] Reagan L.Galvez, Elmer P. Dadios, Argel A. Bandala,Ryan Rhay P. Vicerra, Jose Martin Z.Maningo:Object Detection Using Convolutional Neural Networks ,Proceedings of TENCON 2018 - 2018 IEEE Region 10 Conference (Jeju, Korea, 28-31 October 2018)

[4] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, "You Only Look Once: Unified, Real-Time Object Detection" arXiv: 1506.02640v5 [cs.CV] 9 May 2016

[5] Joseph Redmon, Ali Farhadi , "YOLOv3: An Incremental Improvement" arXiv: 1804.02767v1 [cs.CV] 8 April 2018

[6] Geethapriya. S, N. Duraimurugan, S.P. Chokkalingam, "Real-Time Object Detection with Yolo" International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-8, Issue-3S, February 2019

[7] Anshika Sharma:A survey on object recognition and segmentation techniques, IEEE Conference, At New Delhi, March,2016

[8] Zheng Yin, Chen Quanqi, Zhang Yujin:Deep Learning and its new progress in object and behavior recognition,2014

[9] Hakan Bilen, Marco Pedersoli, Vinay P. Namboodiri, Tinne Tuytelaars, Luc Van Gool:Object Classification with Adaptable Regions,IEEE Conference on Computer Vision and Pattern Recognition 2014

[10] Juan Wu, Bo Peng, Zhenxiang Huang, Jietao Xie:Research on Computer Vision-Based Object Detection and Classification, CCTA 2012: Computer and Computing Technologies in Agriculture VI pp 183-188

[11] Yimeng Zhang, Tsuhan Chen:Weakly Supervised Object Recognition and Localization with Invariant High Order Features,Conference: British Machine Vision Conference, BMVC 2010, Aberystwyth, UK, August 31 - September 3, 2010.

[12] Hedi Harzallah, Frederic Jurie, Cordelia Schmid:Combining efficient object localization and image classification, 2009 IEEE 12th International Conference on Computer Vision.

[13] A. Vidyavani , K. Dheeraj, M. Rama Mohan Reddy, KH. Naveen Kumar , "Object Detection Method Based on YOLOv3 using Deep Learning Networks" International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-9 Issue-1, November 2019

[14] Qiang Chen , Zheng Song , Jian Dong , Zhongyang Huang , Yang Hua , Shuicheng Yan,
"Contextualizing Object Detection and Classification" IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume: 37 , Issue: 1 , Jan. 1 2015)

[15] Licheng Jiao, Fellow, IEEE, Fan Zhang, Fang Liu, Senior Member, IEEE, Shuyuan Yang, Senior Member, IEEE, Lingling Li, Member, IEEE, Zhixi Feng, Member, IEEE, and Rong Qu, Senior Member, IEEE ,"A Survey of Deep Learning-based Object Detection"
arXiv:1907.09408v2 [cs.CV] 10 Oct 2019

Reference Websites

https://towardsdatascience.com/

https://medium.com/

https://en.wikipedia.org/wiki/Deep_learning

https://en.wikipedia.org/wiki/Object_detection