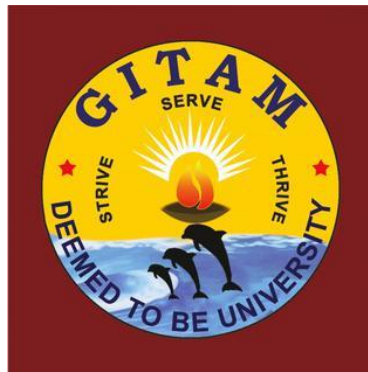


TREES IN STREET DETECTION USING DEEP LEARNING

**Mini Project submitted in partial fulfilment of the requirements for the award of
the degree of
BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING**

**Submitted by
Chilakamarri Sai Srinivas(221710302017)
Sathunuri Dinesh(221710302059)
Saimpu Sai Mohit(221710302056)
Potlapally Shantan(221710302049)**

**Under the guidance of
Dr. G.Yugandhar
Assistant Professor**



**DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING**

GITAM

(Deemed to be University)

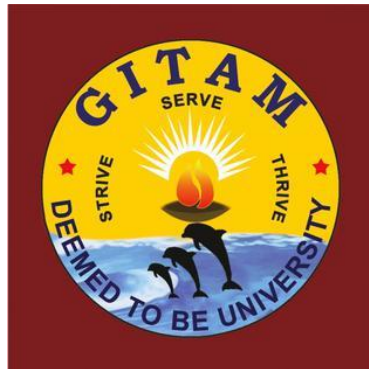
HYDERABAD

DECEMBER 2020

**DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING**

GITAM

(Deemed to be University)



DECLARATION

We hereby declare that the mini project entitled “Trees in Street Detection using Deep Learning” is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfilment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date:

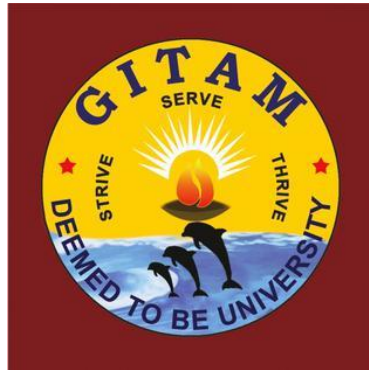
Chilakamarri Sai Srinivas (221710302017)

Sathunuri Dinesh (221710302059)

Saimpu Sai Mohit (221710302056)

Potlapally Shantan (221710302059)

**DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING
GITAM
(Deemed to be University)**



CERTIFICATE

This is to certify that Mini Project entitled “Trees in Street Detection using Deep Learning” is submitted by Chilakamarri Sai Srinivas (221710302017) , Sathunuri Dinesh (221710302059), Saimpu Sai Mohit (221710302056), Potlapally Shantanu (221710302059) in partial fulfilment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering. The Mini Project has been approved as it satisfies the academic requirements.

Dr. G.Yugandhar
Assistant Professor
Dept. of Computer
Science and Engineering

ACKNOWLEDGEMENT

Our Mini Project would not have been successful without the help of several people. we would like to thank the personalities who were part of our seminar in numerous ways, those who gave us outstanding support from the birth of the seminar.

We are extremely thankful to our honourable Pro-Vice Chancellor, **Prof. N. Siva Prasad** for providing necessary infrastructure and resources for the accomplishment of our seminar.

We are highly indebted to **Prof. N. Seetharamaiah**, Principal, School of Technology, for his support during the tenure of the seminar.

We are very much obliged to our beloved **Prof. S. Phani Kumar**, Head of the Department of Computer Science & Engineering for providing the opportunity to undertake this seminar and encouragement in completion of this seminar.

We hereby wish to express our deep sense of gratitude to **Dr. S Aparna**, Assistant Professor, Department of Computer Science and Engineering, School of Technology and to **Dr. G.Yugandhar**, Assistant Professor, Department of Computer Science and Engineering, School of Technology for the esteemed guidance, moral support and invaluable advice provided by them for the success of the summer Internship.

We are also thankful to all the staff members of Computer Science and Engineering department who have cooperated in making our seminar a success. We would like to thank all our parents and friends who extended their help, encouragement and moral support either directly or indirectly in our seminar work.

Sincerely,

Chilakamarri Sai Srinivas(221710302017)

Sathunuri Dinesh(221710302059)

Saimpu Sai Mohit(221710302056)

Potlapally Shantan(221710302049)

CONTENTS

ABSTRACT	I
LIST OF FIGURES	II
SYMBOLS AND ABBREVIATIONS	III
1.INTRODUCTION	1
1.1 ARTIFICIAL INTELLIGENCE (AI)	2
1.2 MACHINE LEARNING (ML)	2
1.3 DEEP LEARNING	3
1.4 TYPES OF MACHINE LEARNING	4
1.4.1 SUPERVISED LEARNING	4
1.4.2 UNSUPERVISED LEARNING	5
1.4.3 REINFORCEMENT LEARNING	6
1.5 PROBLEM DEFINITION	7
1.5.1 OBJECTIVE	7
1.5.2 LIMITATIONS	7
1.5.3 MOTIVATION FOR THE PROJECT	8
1.6 APPLICATIONS OF TREE DETECTION	8
2.LITERATURE SURVEY	9
2.1 PREVIOUSLY DONE RESEARCH TO DETECT TREES	10
2.2 OUR RESEARCH	13
2.3 RELIABILITY	13
3.ANALYSIS	14
3.1 PROBLEM DEFINITION	14
3.2 REQUIREMENT ANALYSIS	14
3.2.1 HARDWARE REQUIREMENTS	14
3.2.2 SOFTWARE REQUIREMENTS	15
3.3 FEASIBILITY STUDY	15
3.3.1 TECHNICAL FEASIBILITY	15
3.3.2 FINANCIAL FEASIBILITY	16
3.3.3 OPERATIONAL FEASIBILITY	16
3.3.4 RESOURCE FEASIBILITY	17

3.3.5 TIME FEASIBILITY	17
3.4 BOUNDARIES OF THE PROJECT	17
4.DESIGN	19
4.1 STEPS TO IMPLEMENT THE MODEL	19
4.2 RESNET MODEL	19
4.2.1 INTRODUCTION	19
4.2.2 ARCHITECTURE	20
4.2.3 USES	21
4.3 PART ATTENTION NETWORK FOR TREE DETECTION(PANTD)	21
4.3.1 WHAT IS A CONVOLUTION	22
4.3.2 WHAT IS A POOLING	23
4.3.3 STRIDING	25
4.3.4 USER INTERFACE USING TKINTER	26
5.IMPLEMENTATION	28
5.1 INTRODUCTION	28
5.2 PROGRAM IMPLEMENTATION	28
5.3 SOURCE CODE	30
6.TESTING AND VALIDATION	37
6.1 INTRODUCTION TO TESTING	37
6.2 TESTING STRATEGIES	37
6.2.1 WHAT DO YOU VERIFY IN SYSTEM TESTING?	37
6.2.2 DIFFERENT TYPES OF SYSTEM TESTING	39
6.2.3 WHAT TYPES OF SYSTEM TESTING SHOULD TESTERS USE?	39
6.3 OUR TESTING AND VALIDATION RESULTS	40
7.RESULTS ANALYSIS	43
8.CONCLUSION	46
9.REFERENCES	47
WEBSITES REFERRED	47

ABSTRACT

Trees play a major part in maintaining the balance in nature by reducing noise and air pollution. Therefore, monitoring their number in a place is important part but to identify the trees and their categories is difficult for humans as we need large number of resources and time. With the help of deep learning techniques for object detection we can identify and categorize trees easily.

In this paper, we propose trainable network for automatic street tree detection, based on a state-of-the-art deep learning-based object detector. This model is based on R-CNN and we used street view images as input and identify the trees in the image.

Index Terms: Tree detection, convolutional neural network, deep learning, part attention, occlusion.

S.pin Number	Name of the student.	Name and Signature
221710302017	Chilakamarri Sai Srinivas	of project guide
221710302059	Sathunuri Dinesh	Dr. G. Yugandhar
221710302056	Saimpu Sai Mohit	
221710302049	Potlapally Shantan	

Signature of Project coordinator

Dr. S.Aparna

LIST OF FIGURES

Fig 1.1 Relation between AI, ML, Deep Learning	3
Fig 1.2 Supervised Learning Example	4
Fig 1.3 Unsupervised Learning	5
Fig 1.4 Reinforcement Learning	6
Fig 4.1 Flow diagram for model	19
Fig 4.2 ResNet-50 model	20
Fig 4.3 Convolutional Neural Network	22
Fig 4.4 Convolution operation	23
Fig 4.5 Max Pooling	24
Fig 4.6 No Striding Length Mentioned	25
Fig 4.7 Stride length is 1	25
Fig 4.8 Stride length is 2	25
Fig 6.1 Flow Chart for Testing	38
Fig 6.2 Result-1	41
Fig 6.3 Result-2	41
Fig 6.4 Result-3	42
Fig 7.1 Output-1	43
Fig 7.2 Output-2	44
Fig 7.3 Output-3	44
Fig 7.4 Output-4	45

SYMBOLS AND ABBREVIATIONS

AI	Artificial Intelligence
ML	Machine Learning
CNN	Convolutional Neural Network
PANTD	Part Attention Network for Tree Detection
ResNet	Residual Neural Network

1.INTRODUCTION

Over the years, technology has revolutionized our world and daily lives. Technology has created amazing tools and resources, putting useful information at our fingertips. Modern technology has paved the way for multi-functional devices like the smartwatch and the smartphone.

Computers are increasingly faster, more portable, and higher-powered than ever before. With all of these revolutions, technology has also made our lives easier, faster, better, and more fun. These days many tasks performed by humans are being carried out by machines and robots. Artificial intelligence and machine learning, deep learning have become buzz words recently in the technology sector.

There are many companies and organisations trying to create very good applications based on Artificial intelligence and Machine learning, Deep learning. Organisations namely Google, Microsoft and other IT companies working on these projects.

We also have good number of resources to work on these AI and ML projects, there are huge number of libraries developed by many organisations. Python is most used language these days and its usage has been growing since last 5 years. This programming language is known for its simplicity and the number of libraries it offers to user are simply amazing.

Some of the libraries are named below:

- **TensorFlow library:** This library is used to work with Ai and ML models, it offers a bunch of tools and utilities. It is developed by Google.
- **Pytorch library:** This library is also most popular one and offers huge number of methods to work on AI and ML projects. It is developed by Facebook's AI Research Lab.

1.1 ARTIFICIAL INTELLIGENCE (AI)

Artificial intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions. The term may also be applied to any machine that exhibits traits associated with a human mind such as learning and problem-solving. The ideal characteristic of artificial intelligence is its ability to rationalize and take actions that have the best chance of achieving a specific goal.

The intelligence is intangible. It is composed of

- Reasoning
- Learning
- Problem Solving
- Perception
- Linguistic Intelligence

The objectives of AI research are reasoning, knowledge representation, planning, learning, natural language processing, realization, and ability to move and manipulate objects. There are long-term goals in the general intelligence sector. Approaches include statistical methods, computational intelligence, and traditional coding AI. During the AI research related to search and mathematical optimization, artificial neural networks and methods based on statistics, probability, and economics, we use many tools. Computer science attracts AI in the field of science, mathematics, psychology, linguistics, philosophy and so on

1.2 MACHINE LEARNING (ML)

Machine learning (ML) is the study of computer algorithms that improve automatically through experience. It is seen as a subset of artificial intelligence. Machine learning algorithms build a model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or unfeasible to develop

conventional algorithms to perform the needed tasks.

A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers; but not all machine learning is statistical learning. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning.

1.3 DEEP LEARNING

Deep learning (also known as deep structured learning) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised.

Deep-learning architectures such as deep neural networks, deep belief networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, machine vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance. The adjective "deep" in deep learning comes from the use of multiple layers in the network.

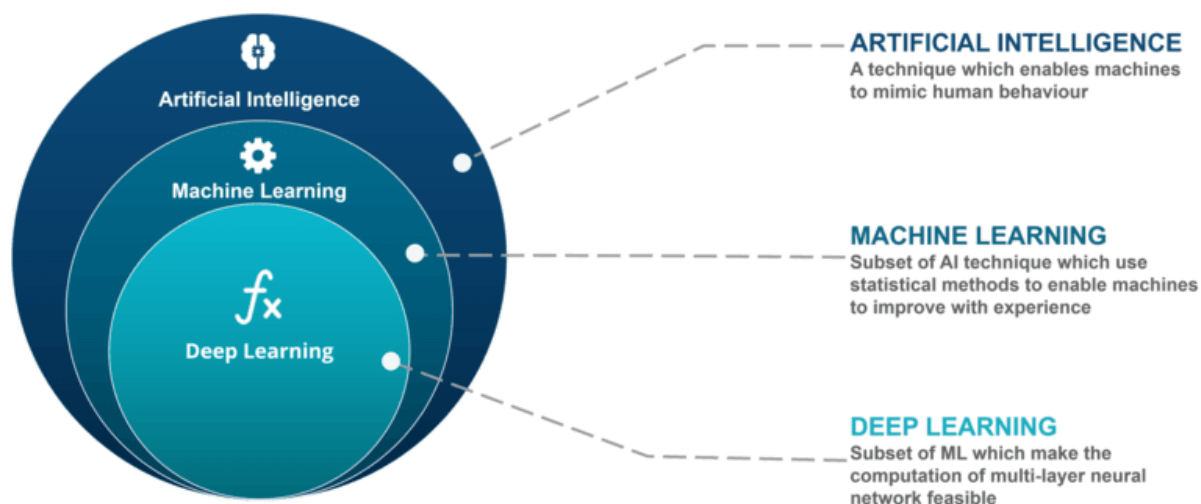


Fig 1.1 Relation between AI, ML, Deep Learning

1.4 TYPES OF MACHINE LEARNING

1.4.1 Supervised Learning

Supervised learning is the most popular paradigm for machine learning. It is the easiest to understand and the simplest to implement. It is very similar to teaching a child with the use of flash cards.

Given data in the form of examples with labels, we can feed a learning algorithm these example-label pairs one by one, allowing the algorithm to predict the label for each example, and giving it feedback as to whether it predicted the right answer or not. Over time, the algorithm will learn to approximate the exact nature of the relationship between examples and their labels. When fully-trained, the supervised learning algorithm will be able to observe a new, never-before-seen example and predict a good label for it.

Supervised learning is often described as task-oriented because of this. It is highly focused on a singular task, feeding more and more examples to the algorithm until it can accurately perform on that task. This is the learning type that you will most likely encounter, as it is exhibited in many of the common applications such as Advertisement Popularity, Spam Classification etc.

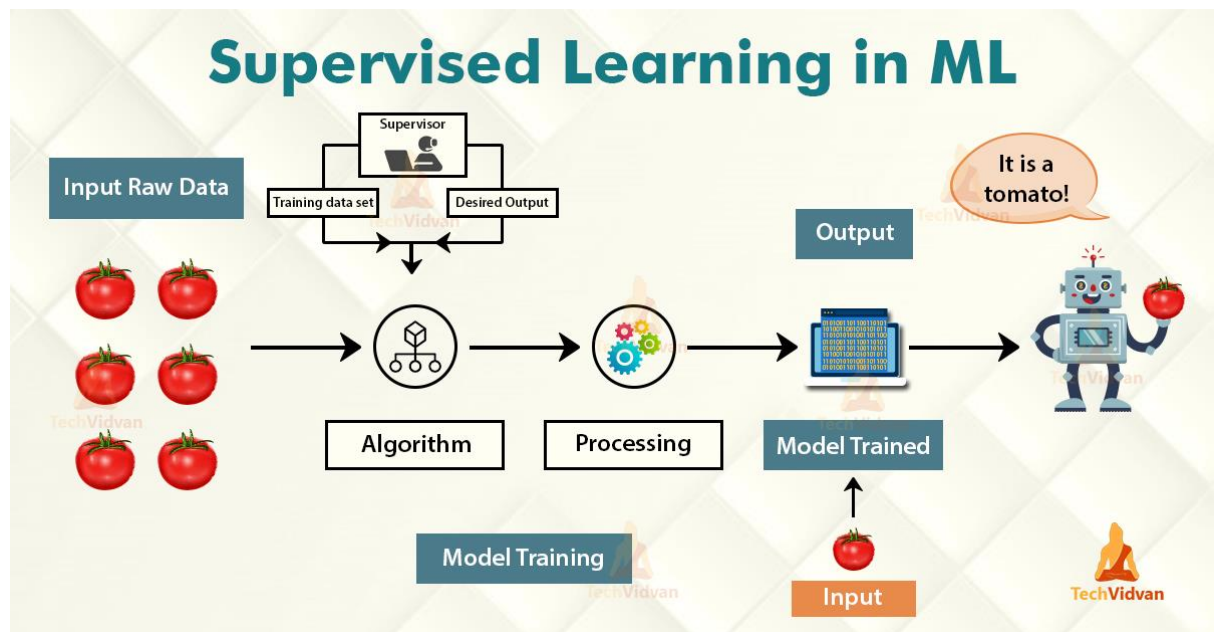


Fig 2.2 Supervised Learning Example

1.4.2 Unsupervised Learning

Unsupervised learning is very much the opposite of supervised learning. It features no labels. Instead, our algorithm would be fed a lot of data and given the tools to understand the properties of the data. From there, it can learn to group, cluster, and/or organize the data in a way such that a human (or other intelligent algorithm) can come in and make sense of the newly organized data.

For example, what if we had a large database of every research paper ever published and we had an unsupervised learning algorithms that knew how to group these in such a way so that you were always aware of the current progression within a particular domain of research. Now, you begin to start a research project yourself, hooking your work into this network that the algorithm can see. As you write your work up and take notes, the algorithm makes suggestions to you about related works, works you may wish to cite, and works that may even help you push that domain of research forward. With such a tool, your productivity can be extremely boosted.

Because unsupervised learning is based upon the data and its properties, we can say that unsupervised learning is data-driven. The outcomes from an unsupervised learning task are controlled by the data and the way it's formatted. Some areas you might see unsupervised learning crop up are Recommender systems, buying habits, grouping user log.

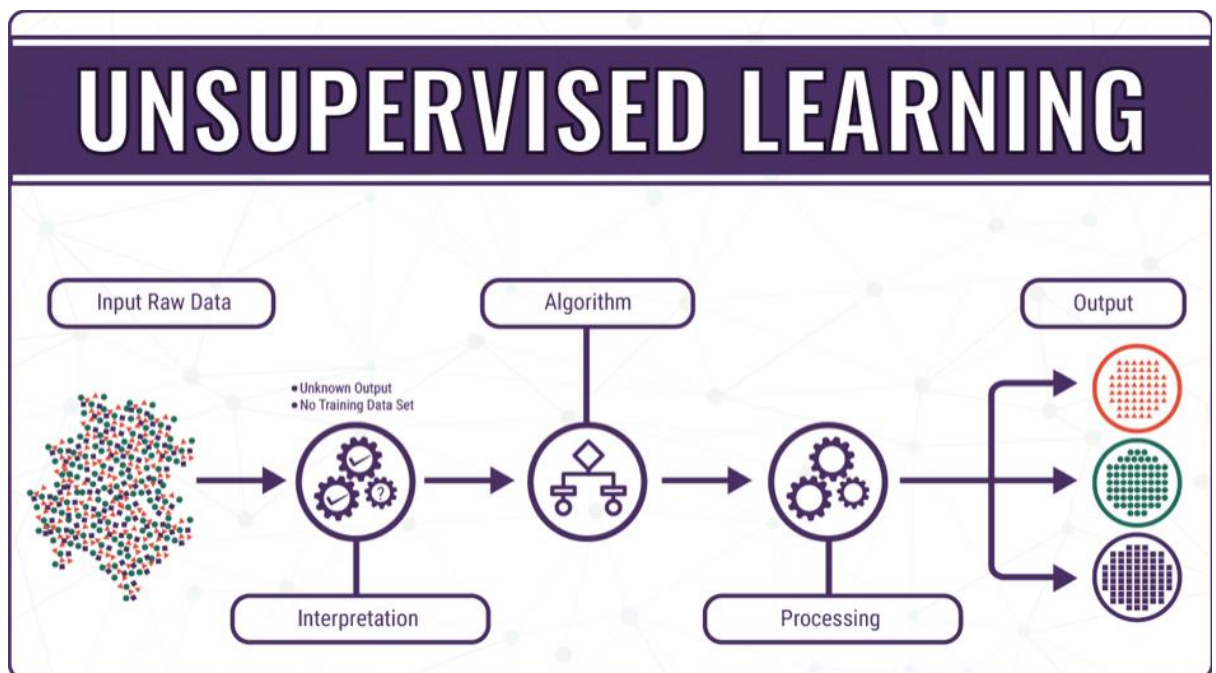


Fig 1.3 Unsupervised Learning

1.4.3 Reinforcement Learning

Reinforcement learning is fairly different when compared to supervised and unsupervised learning. Where we can easily see the relationship between supervised and unsupervised (the presence or absence of labels), the relationship to reinforcement learning is a bit murkier. Some people try to tie reinforcement learning closer to the two by describing it as a type of learning that relies on a time-dependent sequence of labels.

Reinforcement learning is like learning from mistakes. Place a reinforcement learning algorithm into any environment and it will make a lot of mistakes in the beginning. So long as we provide some sort of signal to the algorithm that associates good behaviors with a positive signal and bad behaviors with a negative one, we can reinforce our algorithm to prefer good behaviors over bad ones. Over time, our learning algorithm learns to make less mistakes than it used to.

In real world reinforcement learning is used for Video games, Industrial simulation and Resource management.

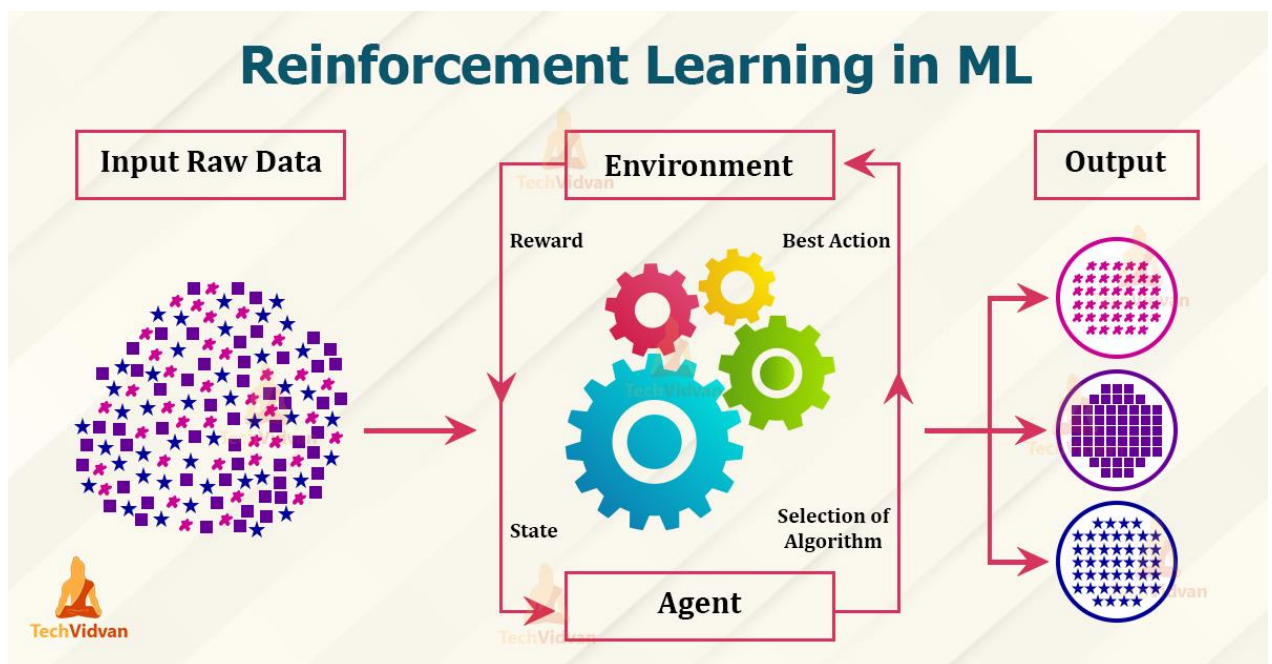


Fig 1.4 Reinforcement Learning

1.5 PROBLEM DEFINITION

Trees play a major part in maintaining the balance in nature by reducing noise and air pollution. In the past, this problem can only be solved manually. Experts were asked to go into the street, counting and classifying trees along the road in sequence, which obviously required considerable human resources and time.

Therefore, monitoring their number in a place is important part but to identify the trees and their categories is difficult for humans as we need large number of resources and time. With the help of deep learning techniques for object detection we can identify and categorize trees easily.

1.5.1 Objective

The main objective of this project is to identify the type of tree that is shown in the image and display the name on the image with bounding box around it.

1.5.2 Limitations

- **Image/sensor noise:** Sensor noise from a handheld camera is typically higher than that of a traditional scanner. Additionally, low-priced cameras will typically interpolate the pixels of raw sensors to produce real colours.
- **Blurring:** Uncontrolled environments tend to have blur, especially if the end user is utilizing a smartphone that does not have some form of stabilization.
- **Lighting conditions:** We cannot make any assumptions regarding our lighting conditions in natural scene images. It may be near dark, the flash on the camera may be on, or the sun may be shining brightly, saturating the entire image.
- **Resolution:** Not all cameras are created equal — we may be dealing with cameras with sub-par resolution.

These all factors mentioned above affect the performance or the accuracy of the machine learning model and sometimes the model even though trained on large set of samples may not be able to detect text exactly as expected.

1.5.3 Motivation for the Project

The tree has become an indispensable part for densely built cities. The majority of the trees are planted along the roads and play an important role in the city system. They act as multifunction systems of dust and noise reduction, sunshade for pedestrians, etc. Therefore, monitoring their health and growth is necessary. The first and most important task is to figure out their quantity. The government would like to know the specific quantity of street trees in a given area and the category of tree they belong to. In the past, this problem can only be solved manually. Experts were asked to go into the street, counting and classifying trees along the road in sequence, which obviously required considerable human resources and time. Recently, leveraging the usage of street view vehicles, municipal administration companies can capture a series of street view images in a short time. These images are then sent to experts to detect trees manually via labelling tools in the computer.

1.6 Applications of Tree Detection

Trees provide shade and shelter, timber for construction, fuel for cooking and heating, and fruit for food as well as having many other uses. In parts of the world, forests are shrinking as trees are cleared to increase the amount of land available for agriculture.

They act as multifunction systems of dust and noise reduction, sunshade for pedestrians, etc. Therefore, monitoring their health and growth is necessary [1]. The first and most important task is to figure out their quantity. The government would like to know the specific quantity of street trees in a given area and the category of tree they belong to.

2.LITERATURE SURVEY

Tree detection is a long-standing research problem in both remote sensing and computer vision. There have been numerous attempts to extract trees, in both the wild scene and urban scene, employing the state-of-the-art detection and pattern classification algorithms in the remote sensing field for forest inventory. Airborne laser and lidar are usually used to capture the 3D point cloud of the trees. The trees are then detected from the point cloud by analyzing their 3D structure information. Aval et al. proposed to detect individual trees along the street from airborne hyperspectral data and digital surface models. Nevertheless, 3D sensors, like laser scanner, are much more expensive.

Using high-resolution satellite images, Li et al. proposed a two-stage convolutional neural network for oil palm tree detection, obtaining the F1-score of 94.99% in a study area of about 55km². In contrast, normal 2D images are a cheaper choice. In the computer vision field, a street tree is important information for the automatic driving task. Videos captured by the vision system would be preprocessed via analyzing the structure information before it is used to assist the decision making. Segmentation is always the first step. Most of the vision based automatic driving techniques would first segment the scene, including trees, captured by the cameras in the car. However, this kind of segmentation is a coarse analysis of street tree information which is unsatisfactory for tasks, like individual tree counting.

There are also some attempts on applying deep learning techniques into the detection of trees. For instance, Shah et al. developed a automatic framework based on a tree detection algorithm and the quadcopter device to recognize and localize trees for preventing deforestation. Their algorithm consists of the state-of-the-art one-stage CNN based detector YOLO to detect trees, and monocular quadcopter flies to take image pairs and locate detected trees via assigning GPS coordinates of the quadcopter to them. Nevertheless, they directly employ the generic object detector in via finetuning in their own dataset of trees. Thus, their tree detection module has a very limited capability of handling the occlusion problem in tree detection scenarios, due to the inherent characteristics of the YOLO architecture.

2.1 PREVIOUSLY DONE RESEARCH TO DETECT TREES

In 2007, J. Secord and A. Zakhor took interest in the construction of 3-D models of urban and suburban environments. Traditionally, stereo-imaging methods have been used since aerial imagery is readily available and relatively inexpensive to obtain. However, interest in aerial lidar data is emerging due to the higher achievable accuracy than in the past and the increased number of algorithms to process the data. One such approach has been developed in the Video and Image Processing Lab at the University of California, Berkeley, over the past five years.

This approach involves segmenting aerial lidar data and applying Random Sample Consensus (RANSAC) polygonization algorithm to delineate roofs of individual buildings. While this approach works well on urban regions with few trees, there is a substantial performance degradation in suburban regions with a large number of trees. Therefore, it is conceivable to improve the accuracy and appearance of the overall models by removing all data points corresponding to trees from the aerial imagery and lidar data prior to applying the RANSAC-based polygonization algorithm.

In 2013, W. Ouyang and X. Wang worked on Pedestrian detection is a key technology in automotive safety, robotics, and intelligent video surveillance to understand the use of deep leaning techniques to detect trees. It has attracted a great deal of research interest. The main challenges of this task are caused by the intra-class variation of pedestrians in clothing, lighting, backgrounds, articulation, and occlusion.

In order to handle these challenges, a group of interdependent components are important. First, features should capture the most discriminative information of pedestrians. Well-known features such as Haar-like features, SIFT, and HOG are designed to be robust to intraclass variation while remain sensitive to inter-class variation. Second, deformation models should handle the articulation of human parts such as torso, head, and legs. The state-of-the-art deformable part-based model in allows human parts to articulate with constraint. Third, occlusion handling approaches seek to identify the occluded regions and avoid their use when determining the existence of a pedestrian in a window.

Finally, a classifier decides whether a candidate window shall be detected as enclosing a pedestrian. SVM, boosted classifiers, random forests, and their variations are often used. Although these components are interdependent, their interactions have not been well explored. Currently, they are first learned or designed individually or sequentially, and then put together in a pipeline. The interaction among these components is usually achieved using manual parameter configuration. Consider the following three examples. The HOG feature is individually designed with its parameters manually tuned given the linear SVM classifier being used in then HOG feature become fixed when people design new classifiers. A few HOG feature parameters are tuned in and fixed, and then different part models are learned in by fixing HOG features and deformable models, occlusion handling models are learned in, using the part-detection scores as input.

In 2014, S. Malek, Y. Bazi, N. Alajlan, H. AlHichri, and F. Melgani have implemented in the latest developments in unmanned aerial vehicles (UAVs) and associated sensing systems make these platforms increasingly attractive to the remote sensing community. The large amount of spatial details contained in these images opens the door for advanced monitoring applications. In this paper, we use this cost-effective and attractive technology for the automatic detection of palm trees. Given a UAV image acquired over a palm farm, first we extract a set of keypoints using the Scale-invariant Feature Transform (SIFT). Then, we analyze these keypoints with an extreme learning machine (ELM) classifier a priori trained on a set of palm and non-palm keypoints.

As output, the ELM classifier will mark each detected palm tree by several keypoints. Then, in order to capture the shape of each tree, we propose to merge these keypoints with an active contour method based on level sets (LSs). Finally, we further analyze the texture of the regions obtained by LS with local binary patterns (LBPs) to distinguish palm trees from other vegetations. Experimental results obtained on UAV images with 3.5 cm of spatial resolution and acquired over two different farms confirm the promising capabilities of the proposed framework.

In 2015, S. Ren, K. He, R. Girshick, and J. Sun worked on State-of-the-art object detection networks depend on region proposal algorithms to hypothesize object locations. Advances like SPPnet and Fast R-CNN have reduced the running time of

these detection networks, exposing region proposal computation as a bottleneck. In this work, we introduce a Region Proposal Network(RPN) that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals.

An RPN is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position. The RPN is trained end-to-end to generate high-quality region proposals, which are used by Fast R-CNN for detection. We further merge RPN and Fast R-CNN into a single network by sharing their convolutional features-using the recently popular terminology of neural networks with 'attention' mechanisms, the RPN component tells the unified network where to look. For the very deep VGG-16 model, our detection system has a frame rate of 5 fps (including all steps) on a GPU, while achieving state-of-the-art object detection accuracy on PASCAL VOC 2007, 2012, and MS COCO datasets with only 300 proposals per image. In ILSVRC and COCO 2015 competitions, Faster R-CNN and RPN are the foundations of the 1st-place winning entries in several tracks. Code has been made publicly available.

In 2017, U. Shah, R. Khawad, and K. M. Krishna worked in detecting, localizing, and recognizing trees with a quadcopter equipped with monocular camera. The quadcopter flies in an area of semidense plantation filled with many trees of more than 5 meter in height. Trees are detected on a per frame basis using state of the art Convolutional Neural Networks inspired by recent rapid advancements showcased in Deep Learning literature. Once detected, the trees are tagged with a GPS coordinate through our global localizing and positioning framework. Further the localized trees are segmented, characterized by feature descriptors, and stored in a database by their GPS coordinates.

In a subsequent run in the same area, the trees that get detected are queried to the database and get associated with the trees in the database. The association problem is posed as a dynamic programming problem and the optimal association is inferred. The algorithm has been verified in various zones in our campus infested with trees with varying density on the Bebop 2 drone equipped with omnidirectional vision. High percentage of successful recognition and association of the trees between two or more runs is the cornerstone of this effort. The proposed method is also able to

identify if trees are missing from their expected GPS tagged locations thereby making it possible to immediately alert concerned authorities about possible unlawful felling of trees.

2.2 OUR RESEARCH

Our project presents a novel framework for the detection of street trees in low illuminance situations and crowded scenarios. The method is comprised of two primary stages: (1) brightness adjustment for input images and (2) street tree detection. The first preprocessing stage is responsible for the adaptive restoration for images with low illuminance, which would increase the detection accuracy, as will be shown in the experiment section. Once the images are brightened, the second stage focuses on the tree detection with the proposed network. The detection network, i.e. PANTD, is based on the stateof-the-art two-stage Faster R-CNN detector, composed of the ResNet backbone, region proposal network and the Fast RCNN module. To address the occlusion problem in street tree detection, we design a part attention module and add it into the Faster R-CNN network. We also propose a new loss function, TLoss, to further alleviate occlusion via punishing proposals from shifting to other ground-truth objects

2.3 RELIABILITY

The improvements of the proposed algorithm over previous methods prove that a tree detection, which directly targets the final goal and eliminating redundant processes, can beat elaborated models, even those integrated with large neural network models.

3.ANALYSIS

Before we get into how this project was implemented and designed, we need to evaluate or analyse the problem on certain criteria. The criteria include:

- Problem definition
- Requirements analysis
- Feasibility study
- Limitations/Boundaries

3.1 PROBLEM DEFINITION

The tree has become an indispensable part for densely built cities. The majority of the trees are planted along the roads and play an important role in the city system. They act as multifunction systems of dust and noise reduction, sunshade for pedestrians, etc. Therefore, monitoring their health and growth is necessary. The first and most important task is to figure out their quantity. The government would like to know the specific quantity of street trees in a given area and the category of tree they belong to. In the past, this problem can only be solved manually. Experts were asked to go into the street, counting and classifying trees along the road in sequence, which obviously required considerable human resources and time.

3.2 REQUIREMENT ANALYSIS

Now coming on to the requirements for this project it depends on which method you will choose while implementation as there are many ways in which this project can be implemented. But the generalized requirements both hardware and software are mentioned below:

3.2.1 Hardware requirements

- **Processor:** Intel-i5/Intel-i7 or any AMD equivalent
- **RAM:** 8GB minimum or higher
- **GPU:** As it's a regression model the GPU isn't the most important component. More RAM matters here
- **Secondary storage:** More storage is recommended for datasets.

3.2.2 Software requirements

- Anaconda
- Visual studio code or PyCharm
- OpenCV

3.3 FEASIBILITY STUDY

This feasibility study provides clear understanding on how resources such as time, money etc are spent on this project and also tells what's the future of this project going to be like and how previous attempts have been performing. There are many factors to be considered such as cost, time, technical feasibility, operational feasibility.

3.3.1 Technical Feasibility

The technical feasibility deals with all possible ways to implement this project. As of now there are three ways in which this text detection project can be implemented. Those three are mentioned below:

1. **Everything from scratch:** In this method everything is developed from scratch on our own. The machine learning model is also developed from scratch and trained on image data sets with sufficient samples until good accuracy is achieved by the model. Now after we are done with the model, we are going to write program on how to use this model and get the predictions based on training experience for the given input images. Here the user will have the ability to tweak the model if he or she finds anything that has to be changed in the later stages.
2. **Using pre-trained model:** In this method to give the user a head starts the pre-trained model code or file will be available already. Here the user has to work upon how to use this model efficiently and design a

program that could use this model data to predict the text in new images or user inputted images and display the result as expected. We have libraries such as scikit learn and statsmodel which have very powerful and accurate models. These models can be used for most academic projects and obtain very high accuracies.

3.3.2 Financial Feasibility

The cost indulged in this project is zero as this is done on academic purpose. But if this project was to be released into the real-world market even then it would not incur much cost. We can call it as low budget project. The cost incurred on first two methods mentioned above are almost negligible. If the project is implemented using the google cloud vision API it may cost us a few bucks if the minimum usage of API exceeds the limit. The cost charged for using API after exceeding limit is shown below.

3.3.3 Operational Feasibility

The operational feasibility deals with how well the proposed system solves the problem. Any one of the above methods satisfies the problem partially but need to be investigated further as the machine learning models accuracy may improve overtime or may be in future due to technological advancements. Method one and method 2 mentioned above in technical feasibility, uses the systems resources as mentioned in the requirements section.

This project also fits into the real-world business as machine learning is one of the recent advancements in technology sector, there are various organisations working on this type of projects and have demand for this type of projects. This could be a very good real-world application if it could be enhanced more. It is also affordable in terms of expenditure and moreover it should be continuously refined where ever there is scope for improvement if deployed as a real-world application. System design and development requires appropriate and timely application of engineering and management efforts to meet the previously mentioned parameters. A system may serve its intended purpose most effectively when its technical and operating characteristics are engineered into the design. Therefore, operational feasibility is a

critical aspect of systems engineering that needs to be an integral part of the early design phases.

3.3.4 Resource Feasibility

The number of resources used by the program depends on what method is being to train and develop the machine learning model, from the methods mentioned above in technical feasibility section method 1 (Everything from scratch) uses most of the system resources while running because when the model trains on such huge datasets its going to require lot of memory for processing that image data and store the results. The CPU and GPU also are continuously working in the back to provide speed access to the data for the machine learning model to train.

The other 2 methods use limited resources probably less than the first method as they do not have training process in those methods, but while they are running, they may use webcam if the user want to feed video live to the program to predict output.

3.3.5 Time Feasibility

A time feasibility study will take into account the period in which the project is going to take up to its completion. A project will fail if it takes too long to be completed before it is useful. Typically, this means estimating how long the system will take to develop. The time taken for the project completion is completely based on the technical expertise, that is the programmer's knowledge about the project and his or her capacity to do programs.

But one certain thing that is know is that it will take lot of time to train the model on image samples because sometimes the datasets may be huge, for example a dataset may have 10,000 to 20,000 images. So, while working on such large scale it takes time.

3.4 BOUNDARIES OF THE PROJECT

This particular project is only specific to tree detection in images, other objects and features of the image are not being detected here. The detected tree is

show in bounding box on the image on the output but we are not able to edit or make any changes to the detected tree.

Even though it is only limited to tree-detection this project acts as a base for identifying the type trees which is the next level of this project. In order to identify tree, we have to first detect it, so this project helps to do that.

4.DESIGN

4.1 STEPS TO IMPLEMENT THE MODEL

This chapter is going to deal with how project was designed and how a structure was formed. Here we are going to discuss about what is the general flow of the program, what is the structure of the machine learning models, and what all are included.

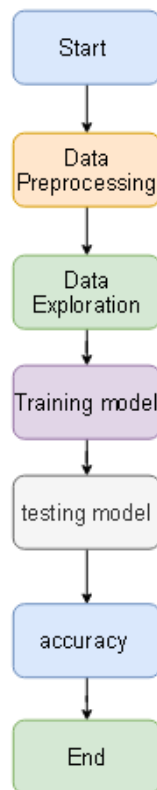


Fig 4.1 Flow diagram for model

4.2 RESNET MODEL

4.2.1 Introduction

In 2012 at the LSVRC2012 classification contest AlexNet won the the first price, After that ResNet was the most interesting thing that happened to the computer vision and the deep learning world.

Because of the framework that ResNets presented it was made possible to train ultra deep neural networks and by that i mean that i network can contain hundreds or thousands of layers and still achieve great performance.

The ResNets were initially applied to the image recognition task but as it is mentioned in the paper that the framework can also be used for non computer vision tasks also to achieve better accuracy.

Many of you may argue that simply stacking more layers also gives us better accuracy why was there a need of Residual learning for training ultra deep neural networks.

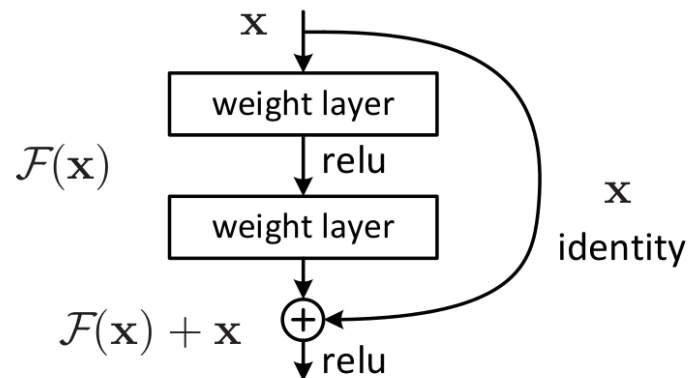


Fig 4.2 ResNet-50 model

4.2.2 Architecture

The ResNet 50 architecture contains the following element:

- A convolution with a kernel size of $7 * 7$ and 64 different kernels all with a stride of size 2 giving us 1 layer.
- Next, we see max pooling with also a stride size of 2.
- In the next convolution there is a $1 * 1,64$ kernel following this a $3 * 3,64$ kernel and at last a $1 * 1,256$ kernel, these three layers are repeated in total 3 time so giving us 9 layers in this step.
- Next, we see kernel of $1 * 1,128$ after that a kernel of $3 * 3,128$ and at last a kernel of $1 * 1,512$ this step was repeated 4 time so giving us 12 layers in this step.
- After that there is a kernel of $1 * 1,256$ and two more kernels with $3 * 3,256$ and $1 * 1,1024$ and this is repeated 6 time giving us a total of 18 layers.

- And then again, a $1 * 1,512$ kernel with two more of $3 * 3,512$ and $1 * 1,2048$ and this was repeated 3 times giving us a total of 9 layers.
- After that we do a average pool and end it with a fully connected layer containing 1000 nodes and at the end a SoftMax function so this gives us 1 layer.

We don't actually count the activation functions and the max/ average pooling layers.

So, totalling this it gives us a $1 + 9 + 12 + 18 + 9 + 1 = 50$ layers Deep Convolutional network.

4.2.3 Uses

- This architecture can be used on computer vision tasks such as image classification, object localization, object detection.
- and this framework can also be applied to non-computer vision tasks to give them the benefit of depth and to reduce the computational expense also.

4.3 PART ATTENTION NETWORK FOR TREE

DETECTION(PANTD)

In deep learning-based object detection architectures, the first component is normally a pretrained CNN with the output of an intermediate layer. As the feature maps extracted by the pre-trained network could lay a good foundation for the subsequent task, i.e. region proposal and classification, the choice of the backbone network is crucial. This is because the types of layers and the number of parameters would directly affect the memory, speed and performance of the detector. As verified in several works ResNet achieves a significant performance improvement in detection accuracy compared to the original VGG16 and ZF-net in Faster R-CNN. With residual connections and batch normalization operations, ResNet is easier to train deep models. Thus, we adapt the ResNet-50 pre-trained on the ImageNet dataset as our backbone network. ResNet-50 consists of 5 main residual blocks (i.e. conv1, conv2 x, conv3 x, conv4 x, conv5 x), each of which contains a set of repeated residual layers. In all, ResNet-50 architecture is comprised of 50 layers, which are repeated

convolution and pooling layers along with fully connected layers. In our experiments, we choose the last layer of conv4 x layers in ResNet-50 to use for predicting region proposals.

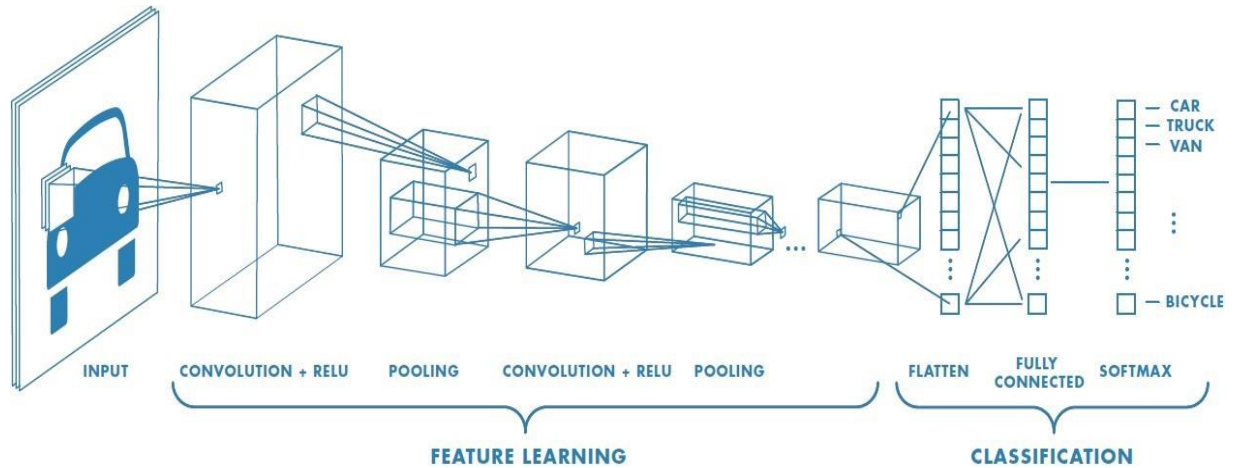


Fig 4.3 Convolutional Neural Network

4.3.1 What is a Convolution

Convolutional layers are the major building blocks used in convolutional neural networks. A convolution is the simple application of a filter to an input that results in an activation. Repeated application of the same filter to an input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input, such as an image.

The innovation of convolutional neural networks is the ability to automatically learn a large number of filters in parallel specific to a training dataset under the constraints of a specific predictive modelling problem, such as image classification. The result is highly specific features that can be detected anywhere on input images.

In the context of a convolutional neural network, a convolution is a linear operation that involves the multiplication of a set of weights with the input, much like a traditional neural network. Given that the technique was designed for two-dimensional input, the multiplication is performed between an array of input data and a two-dimensional array of weights, called a filter or a kernel. The idea of applying the convolutional operation to image data is not new or unique to convolutional neural networks; it is a common technique used in computer vision.

The 2D convolution is a fairly simple operation at heart: you start with a kernel, which is simply a small matrix of weights. This kernel “slides” over the 2D input data, performing an elementwise multiplication with the part of the input it is currently on, and then summing up the results into a single output pixel.

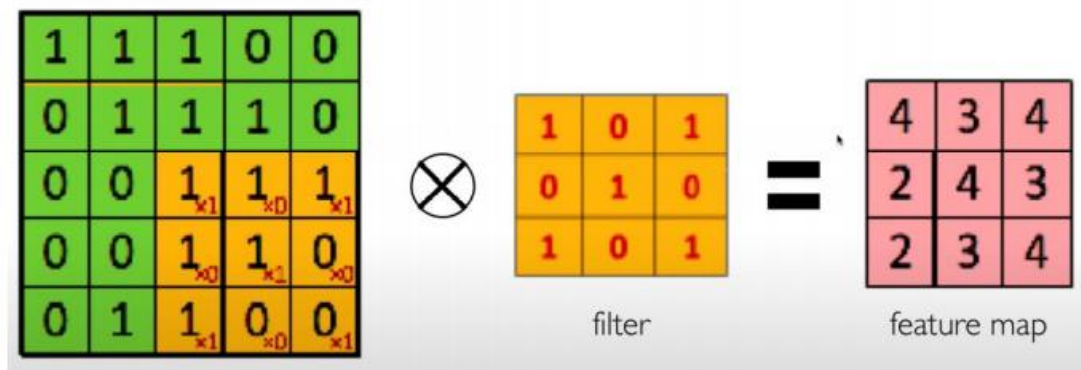


Fig 4.4 Convolution operation

4.3.2 What is a Pooling

Pooling layers provide an approach to down sampling feature maps by summarizing the presence of features in patches of the feature map. Two common pooling methods are average pooling and max pooling that summarize the average presence of a feature and the most activated presence of a feature respectively.

Convolutional layers in a convolutional neural network systematically apply learned filters to input images in order to create feature maps that summarize the presence of those features in the input. Convolutional layers prove very effective, and stacking convolutional layers in deep models allows layers close to the input to learn low-level features (e.g., lines) and layers deeper in the model to learn high-order or more abstract features, like shapes or specific objects.

A limitation of the feature map output of convolutional layers is that they record the precise position of features in the input. This means that small movements in the position of the feature in the input image will result in a different feature map. This can happen with re-cropping, rotation, shifting, and other minor changes to the input image. A common approach to addressing this problem from signal processing is called down sampling. This is where a lower resolution version of an input signal is

created that still contains the large or important structural elements, without the fine detail that may not be as useful to the task. Down sampling can be achieved with convolutional layers by changing the stride of the convolution across the image.

A more robust and common approach is to use a pooling layer. A pooling layer is a new layer added after the convolutional layer. Specifically, after a nonlinearity (e.g., ReLU) has been applied to the feature maps output by a convolutional layer. The pooling layer operates upon each feature map separately to create a new set of the same number of pooled feature maps.

Pooling involves selecting a pooling operation, much like a filter to be applied to feature maps. The size of the pooling operation or filter is smaller than the size of the feature map; specifically, it is almost always 2×2 pixels applied with a stride of 2 pixels. This means that the pooling layer will always reduce the size of each feature map by a factor of 2, e.g., each dimension is halved, reducing the number of pixels or values in each feature map to one quarter the size. For example, a pooling layer applied to a feature map of 6×6 (36 pixels) will result in an output pooled feature map of 3×3 (9 pixels).

The pooling operation is specified, rather than learned. Two common functions used in the pooling operation are:

- **Average Pooling:** Calculate the average value for each patch on the feature map.
- **Maximum Pooling (or Max Pooling):** Calculate the maximum value for each patch of the feature map.

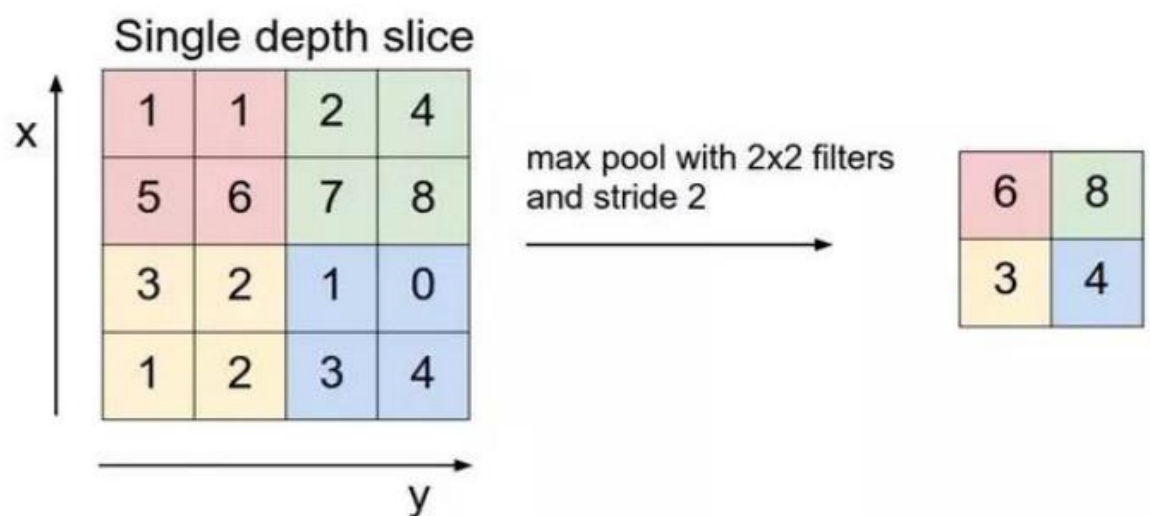


Fig 4.5 Max Pooling

4.3.3 Striding

Stride is a component of convolutional neural networks, or neural networks tuned for the compression of images and video data. Stride is a parameter of the neural network's filter that modifies the amount of movement over the image or video. For example, if a neural network's stride is set to 1, the filter will move one pixel, or unit, at a time. The size of the filter affects the encoded output volume, so stride is often set to a whole integer, rather than a fraction or decimal.

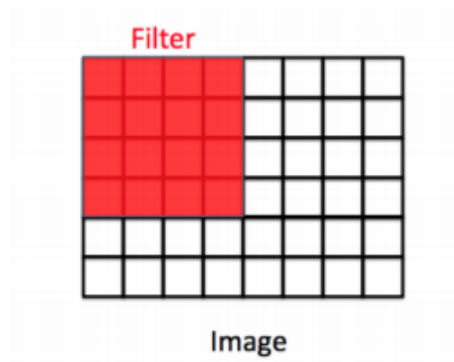


Fig 4.6 No Striding Length Mentioned

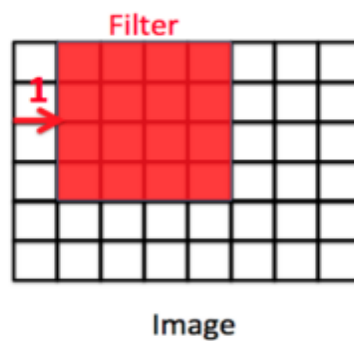


Fig 4.7 Stride length is 1

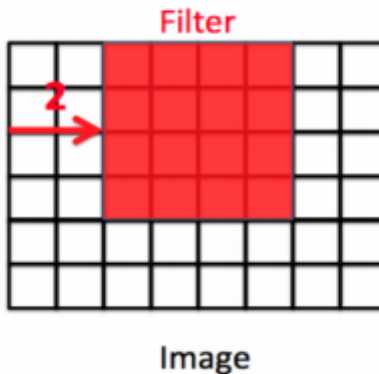


Fig 4.8 Stride length is 2

4.3.4 User Interface using Tkinter

The tkinter package (“Tk interface”) is the standard Python interface to the Tk GUI toolkit. Both Tk and tkinter are available on most Unix platforms, as well as on Windows systems. (Tk itself is not part of Python; it is maintained at ActiveState.)

Running `python -m tkinter` from the command line should open a window demonstrating a simple Tk interface, letting you know that tkinter is properly installed on your system, and also showing what version of Tcl/Tk is installed, so you can read the Tcl/Tk documentation specific to that version.

Most of the time, tkinter is all you really need, but a number of additional modules are available as well. The Tk interface is located in a binary module named `_tkinter`. This module contains the low-level interface to Tk, and should never be used directly by application programmers. It is usually a shared library (or DLL), but might in some cases be statically linked with the Python interpreter.

In addition to the Tk interface module, tkinter includes a number of Python modules, `tkinter.constants` being one of the most important. Importing tkinter will automatically import `tkinter.constants`, so, usually, to use Tkinter all you need is a simple import statement:

```
import tkinter
```

Or, more often:

```
from tkinter import *
```

- Tk was written by John Ousterhout while at Berkeley.
- Tkinter was written by Steen Lumholt and Guido van Rossum.
- This Life Prese
- rver was written by Matt Conway at the University of Virginia.

- The HTML rendering, and some liberal editing, was produced from a FrameMaker version by Ken Manheimer.
- Fredrik Lundh elaborated and revised the class interface descriptions, to get them current with Tk 4.2.
- Mike Clarkson converted the documentation to LaTeX, and compiled the User Interface chapter of the reference manual.

5.IMPLEMENTATION

5.1 INTRODUCTION

In this chapter we are going to look at implementation part in this project. In this we have used Visual Studio Code as our IDE. The first thing is to load all modules required for the image handling and some other libraries for other utilities. When the user runs the program and gives any image the dimensions of the image should be captured and resized according to the model input dimensions. Then after we load our model into the program by using function in open cv. Then we take the images given and make a forward pass-through layers of model. After doing this we get the output as prediction values, so now we decode these predictions and then display the image with a bounding box around the trees.

Now we will look into a detailed code and how we implemented this project. So first let's start with what are all the modules required to run this program. These are called the pre-requisite modules which are required for running the program as expected.

5.2 PROGRAM IMPLEMENTATION

The following are the main modules used in the source code of project

NumPy: The second package we imported here is NumPy. NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open-source project and you can use it freely. NumPy stands for Numerical Python. In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.

cv2: This package we used here was cv2. 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. When it is integrated with various libraries, such as NumPy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e. whatever operations one can do in NumPy can be combined with OpenCV.

tkinter: The tkinter package (“Tk interface”) is the standard Python interface to the Tk GUI toolkit. Both Tk and tkinter are available on most Unix platforms, as well as on Windows systems. (Tk itself is not part of Python; it is maintained at ActiveState.) Running `python -m tkinter` from the command line should open a window demonstrating a simple Tk interface, letting you know that tkinter is properly installed on your system, and also showing what version of Tcl/Tk is installed, so you can read the Tcl/Tk documentation specific to that version.

OS: The OS module in Python provides a way of using operating system dependent functionality. The functions that the OS module provides allows you to interface with the underlying operating system that Python is running on – be that Windows, Mac or Linux.

Pickle: Python pickle module is used for serializing and de-serializing a Python object structure. Any object in Python can be pickled so that it can be saved on disk. What pickle does is that it “serializes” the object first before writing it to file. Pickling is a way to convert a python object (list, dict, etc.)

Keras: Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Up until version 2.3 Keras supported multiple backends, including TensorFlow, Microsoft Cognitive Toolkit, R, Theano, and PlaidML. As of version 2.4, only TensorFlow is supported. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

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. The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel.

5.3 SOURCE CODE

```
#importing all the required packages
from tkinter import messagebox
from tkinter import *
from tkinter import simpledialog
import tkinter
from tkinter import filedialog
from tkinter.filedialog import askopenfilename
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.resnet50 import ResNet50
from keras.models import Model
import keras
import pickle

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
    InputLayer
from keras.models import Sequential
from keras import optimizers
from keras.models import model_from_json
from keras.preprocessing import image
import numpy as np
import cv2
import os

#initialising the graphical user interface
main = tkinter.Tk()
main.title("Trees in Street Detection via Deep Learning")
main.geometry("1000x800")

#initialising the list of classes to be predicted
tree = ['Mango','Neem']

# function to upload the dataset
def uploadDataset():
    filename = filedialog.askdirectory(initialdir=".")
    text.delete('1.0', END)
    text.insert(END,filename+" loaded\n\n")
```

```

# function to create and train the model with dataset
def trainLearningModel():
    text.delete('1.0', END)
    if os.path.exists('model/weight.json'):
        with open('model/weight.json', "r") as json_file:
            type_model_json = json_file.read()
            detector = model_from_json(type_model_json)
            detector.load_weights("model/weights.h5")
            detector._make_predict_function()
            print(detector.summary())
            f = open('model/history.pckl', 'rb')
            data = pickle.load(f)
            f.close()
            acc = data['accuracy']
            loss = data['loss']
            loss = loss[0]
            accuracy = acc[0] * 100
            text.insert(END, "PANTD Model Tree Detection = "+str(accuracy)+"
\n")
            text.insert(END, "PANTD Miss Rate(MR) or Loss = "+str(loss))
        else:
            #creating object of resnet
            restnet = ResNet50(include_top=False, weights='imagenet', input
_shape=(300,300,3))
            output = restnet.layers[-1].output
            output = keras.layers.Flatten()(output)
            restnet = Model(restnet.input, output=output)

            #making last resnet layer to be false to include our PANTD mode
l
            for layer in restnet.layers:
                layer.trainable = False
            restnet.summary()

            #adding restnest model to our PANTD model
            model = Sequential()
            model.add(restnet)

```



```

model.add(Dense(512, activation='relu', input_dim=(300,300,3)))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(2, activation='sigmoid'))

#initialising the methods to calculate accuracy and loss value
of the model
model.compile(loss='binary_crossentropy', optimizer=optimizers.
RMSprop(lr=2e-5), metrics=['accuracy'])
model.summary()

#initialising the variables to store the train and test dataset
s
train_datagen = ImageDataGenerator()
test_datagen = ImageDataGenerator()

#reading train and test images
training_set = train_datagen.flow_from_directory('dataset/train
',target_size = (300, 300), batch_size = 2, class_mode = 'categorical',
shuffle=True)
test_set = test_datagen.flow_from_directory('dataset/train',tar
get_size = (300, 300), batch_size = 2, class_mode = 'categorical', shuf
fle=False)

#building PANTD model
hist = model.fit_generator(training_set, steps_per_epoch=100, e
pochs=1, validation_data=test_set, validation_steps=50, verbose=1)

#storing the data about weights
model.save_weights('model/weights.h5')

#storing the data about model in json file
model_json = model.to_json()
with open("model/weight.json", "w") as json_file:
    json_file.write(model_json)
print(training_set.class_indices)
print(model.summary)

```

```

        #storing the history of data in pickle file
        f = open('model/history.pkl', 'wb')
        pickle.dump(hist.history, f)
        f.close()

        f = open('model/history.pkl', 'rb')
        data = pickle.load(f)
        f.close()

        #getting data about loss and accuracy and print it
        acc = data['accuracy']
        accuracy = acc[0] * 100
        loss = data['loss']
        loss = loss[0]
        text.insert(END, "PANTD Model Tree Detection = "+str(accuracy)+"
\n")

        text.insert(END, "PANTD Miss Rate(MR) or Loss = "+str(loss))
        detector = model

# function to give the output for given images
def detectTree():
    #read the image from directory
    filename = filedialog.askopenfilename(initialdir="testImages")

    #bounds
    lower_green = np.array([36,25,25])
    upper_green = np.array([86, 255,255])

    #resizing the image and converting to array
    imagetest = image.load_img(filename, target_size = (300,300))
    imagetest = image.img_to_array(imagetest)
    imagetest = np.expand_dims(imagetest, axis = 0)

    if os.path.exists('model/weight.json'):
        #read the trained model data and load it
        with open('model/weight.json', "r") as json_file:

```

```

        type_model_json = json_file.read()
        detector = model_from_json(type_model_json)
        detector.load_weights("model/weights.h5")

        #predict the output
        detector._make_predict_function()
        print(detector.summary())

        #save the data
        f = open('model/history.pkl', 'rb')
        data = pickle.load(f)
        f.close()
    else:
        trainLearningModel()

    predict = detector.predict(imagetest)
    print(np.argmax(predict))
    tree_type = tree[np.argmax(predict)]

    #initialise output frame
    frame = cv2.imread(filename)
    frame = cv2.resize(frame,(500,500))
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange (hsv, lower_green, upper_green)

    #plot contours and show the output frame
    contours,temp = cv2.findContours(mask.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    cv2.putText(frame, 'Tree Detected As : '+tree_type, (30, 80), cv2.FONT_HERSHEY_SIMPLEX,0.6, (0, 255, 255), 2)
    for i in range(len(contours)):
        if len(contours[i]) > 10:
            tree_area = contours[i]
            x, y, w, h = cv2.boundingRect(tree_area)
            cv2.rectangle(frame,(x, y),(x+w, y+h),(0, 0, 255), 2)
    cv2.imshow('Tree Detected As : '+tree_type,frame)
    cv2.waitKey(0)

```

```

#function that exits the user interface
def close():
    main.destroy()

#title of the interface
font = ('Calibri', 16, 'bold')
title = Label(main, text='Trees in Street Detection via Deep Learning',
anchor=W, justify=CENTER)
title.config(bg='slateblue', fg='white')
title.config(font=font)
title.config(height=2, width=50)
title.place(x=10,y=10)

#training label
label = Label(main, text="Training the model:")
label.config(font=font)
label.place(x=10,y=100)

#initialise text field
font1 = ('Calibri', 14)
text=Text(main,height=2,width=50)
scroll=Scrollbar(text)
text.configure(yscrollcommand=scroll.set)
text.place(x=100,y=150)
text.config(font=font1)

#button to upload dataset
upload = Button(main, text="Upload Trees Dataset",bg='firebrick1', fg='white', command=uploadDataset)
upload.place(x=650,y=150)
upload.config(font=font1)

#button to train uploaded dataset
trainButton = Button(main, text="Train PANTD Model using Street Tree Dataset", bg='firebrick1', fg='white',command=trainLearningModel)
trainButton.place(x=180,y=210)
trainButton.config(font=font1)

```

```

#testing label
label = Label(main, text="Testing the model:")
label.config(font=font)
label.place(x=10,y=320)

#button to test a new input
testButton = Button(main, text="Upload Test Image & Detect Trees",bg='firebrick1', fg='white', command=detectTree)
testButton.place(x=220,y=370)
testButton.config(font=font1)

#exit the user interface
exitButton = Button(main, text="Exit",bg='firebrick1', fg='white', command=close)
exitButton.place(x=330,y=500)
exitButton.config(font=font1)

main.config(bg='lightblue')
main.mainloop()

```

6.TESTING AND VALIDATION

6.1 INTRODUCTION TO TESTING

System Testing is the testing of a complete and fully integrated software product. Usually, software is only one element of a larger computer-based system. Ultimately, software is interfaced with other software/hardware systems. System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system.

6.2 TESTING STRATEGIES

- Black Box Testing
- White Box Testing

System test falls under the **black box testing** category of software testing.

White box testing is the testing of the internal workings or code of a software application. In contrast, black box or System Testing is the opposite. System test involves the external workings of the software from the user's perspective.

6.2.1 What do you verify in System Testing?

System Testing involves testing the software code for following

- Testing the fully integrated applications including external peripherals in order to check how components interact with one another and with the system as a whole. This is also called End to End testing scenario.
- Verify thorough testing of every input in the application to check for desired outputs.
- Testing of the user's experience with the application.

That is a very basic description of what is involved in system testing. You need to build detailed test cases and test suites that test each aspect of the application as seen from the outside without looking at the actual source code.

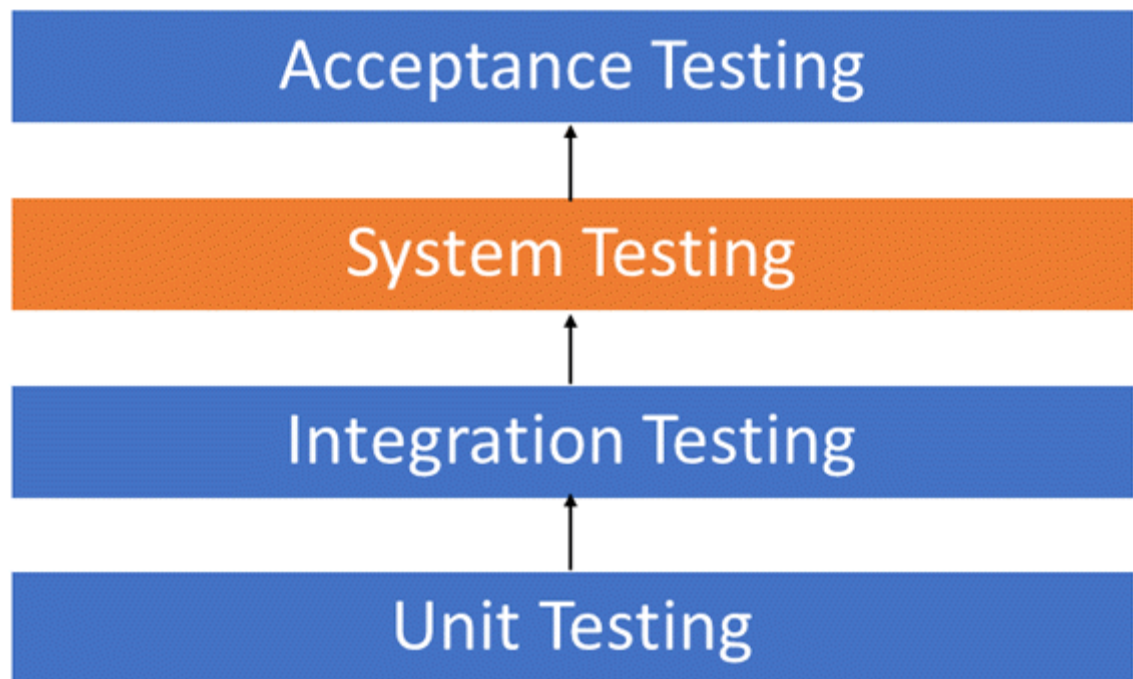


Fig 6.1 Flow Chart for Testing

As with almost any software engineering process, software testing has a prescribed order in which things should be done. The following is a list of software testing categories arranged in chronological order. These are the steps taken to fully test new software in preparation for marketing it:

- **Unit testing** - testing performed on each module or block of code during development. Unit Testing is normally done by the programmer who writes the code.
- **Integration testing** - testing done before, during and after integration of a new module into the main software package. This involves testing of each individual code module. One piece of software can contain several modules which are often created by several different programmers. It is crucial to test each module's effect on the entire program model.
- **System testing** - testing done by a professional testing agent on the completed software product before it is introduced to the market.
- **Acceptance testing** - beta testing of the product done by the actual end users.

6.2.2 Different Types of System Testing

There are more than 50 types of System Testing. Below we have listed types of system testing a large software development company would typically use

1. **Usability Testing** - Usability Testing mainly focuses on the user's ease to use the application, flexibility in handling controls and ability of the system to meet its objectives
2. **Load Testing** - Load Testing is necessary to know that a software solution will perform under real-life loads.
3. **Regression Testing**- - Regression Testing involves testing done to make sure none of the changes made over the course of the development process have caused new bugs. It also makes sure no old bugs appear from the addition of new software modules over time.
4. **Recovery Testing** - Recovery testing is done to demonstrate a software solution is reliable, trustworthy and can successfully recoup from possible crashes
5. **Migration Testing** - Migration testing is done to ensure that the software can be moved from older system infrastructures to current system infrastructures without any issues.
6. **Functional Testing** - Also known as functional completeness testing, Functional Testing involves trying to think of any possible missing functions. Testers might make a list of additional functionalities that a product could have to improve it during functional testing.
7. **Hardware/Software Testing** - IBM refers to Hardware/Software testing as "HW/SW Testing". This is when the tester focuses his/her attention on the interactions between the hardware and software during system testing.

6.2.3 What Types of System Testing Should Testers Use?

There are over 50 different types of system testing. The specific types used by a tester depend on several variables. Those variables include:

- **Who the tester works for** - This is a major factor in determining the types of system testing a tester will use. Methods used by large companies are different than that used by medium and small companies.
- **Time available for testing** - Ultimately, all 50 testing types could be used. Time is often what limits us to using only the types that are most relevant for the software project.
- **Resources available to the tester** - Of course some testers will not have the necessary resources to conduct a testing type. For example, if you are a tester working for a large software development firm, you are likely to have expensive automated testing software not available to others.
- **Software Tester's Education** - There is a certain learning curve for each type of software testing available. To use some of the software involved, a tester has to learn how to use it.

Testing Budget - Money becomes a factor not just for smaller companies and individual software developers but large companies as well.

6.3 OUR TESTING AND VALIDATION RESULTS

To quantitatively evaluate the performance of tree detectors, we plot miss rate (MR) against false positives per image (FPPI) on a log scale, by varying the threshold on detection confidence, following [49]. For certain tasks with only one class of target, e.g. pedestrian detection and defect region detection, miss rate vs false positives per image curve is preferred to precision recall curves in generic object detection. We then use the log-average miss rate MR^{-2} as a single reference value to generally summarize detector performance, which is similar to the average precision. This value is computed in MATLAB, by averaging miss rate at nine FPPI rates uniformly spaced in log-space in the range.

$$MR^{-2} = \exp \left[\frac{1}{n} \sum_{i=1}^n \ln a_i \right]$$

where n is set to be 9, and a_i is the positive value corresponding to the miss rate at 9 evenly spaced FPPI point.

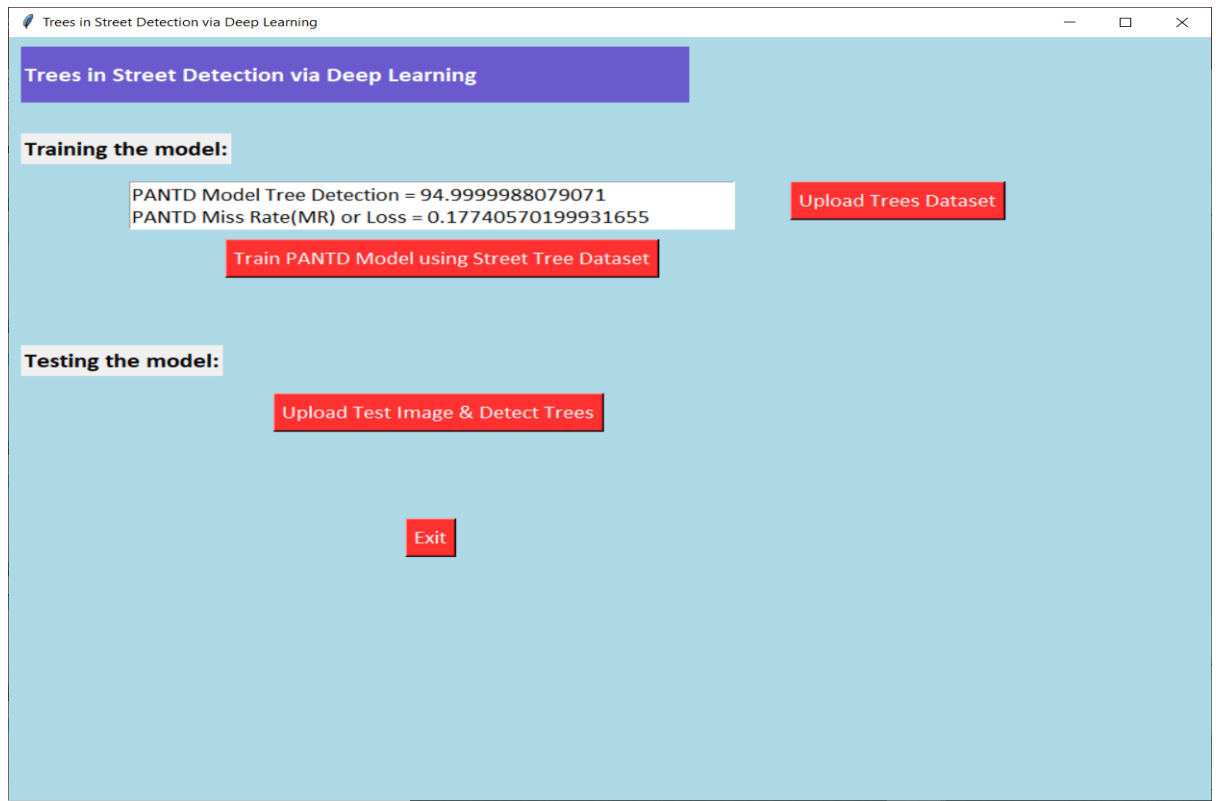


Fig 6.2 Result-1



Fig 6.3 Result-2

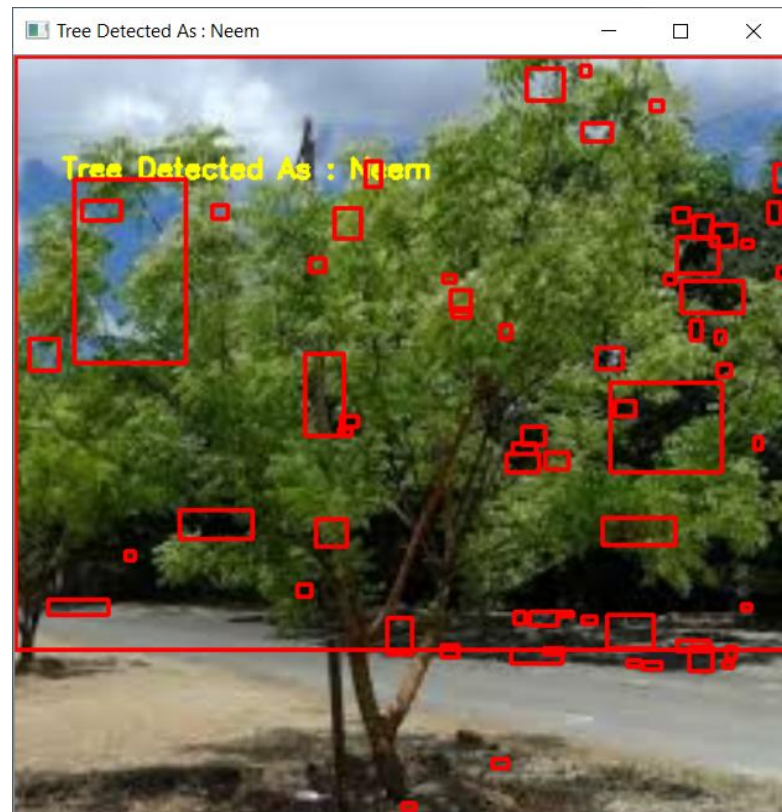
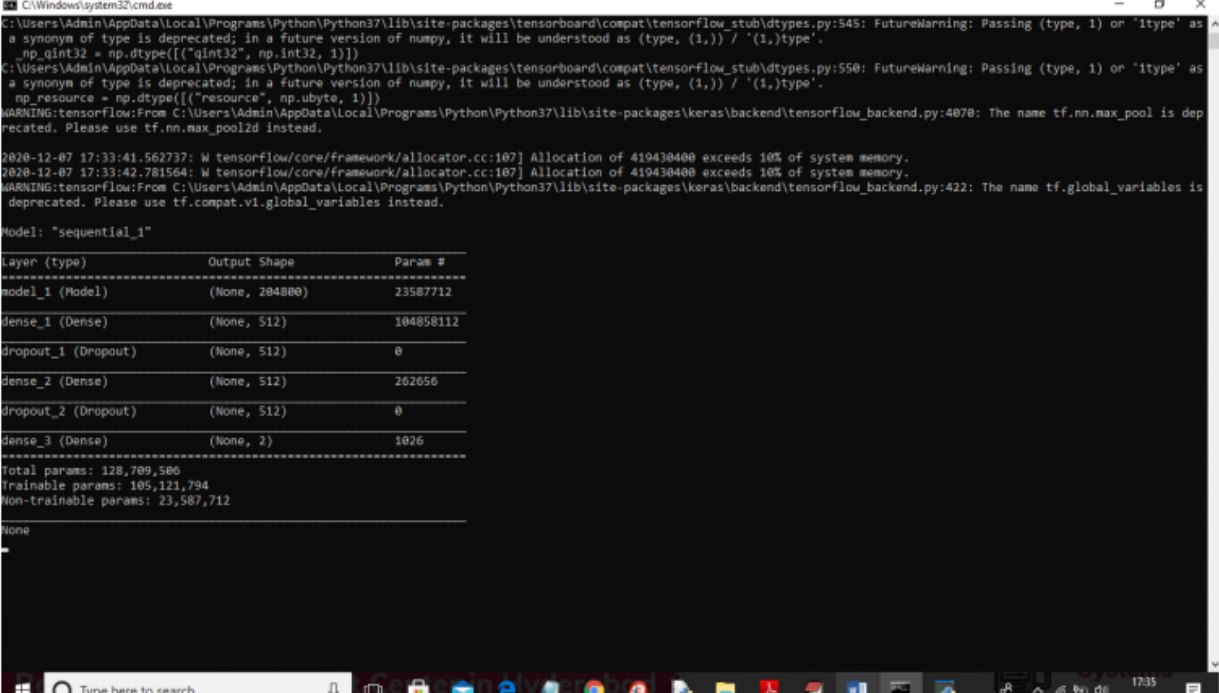


Fig 6.4 Result-3

7.RESULTS ANALYSIS

We have found that our deep learning model based on ResNet architecture to predict different types of trees at roadside. All existing RCNN or FR-CNN models are not good at detecting trees in low resolution or dark images or the trees hidden (or occlusion) behind any other object. In proposed model called Part Attention Network for Tree Detection (PANTD) author is overcoming from hidden or low-resolution images by adding adaptive brightness to the images whose darkness level is below 60. This model is trained with all bright images and then new test images also will go through adaptive brightness procedure to make clear image and then this image will be given to PANTD model and this model will predict type of tree. After building model we have calculated PANTD tree prediction accuracy and loss or miss rate. This model is able to achieve accuracy more than 90 and miss rate is less than 0.17%. In this project we have built PANTD model on top of ResNet model to get better prediction accuracy and less miss rate.

The following are the outputs of the project



```
C:\Windows\system32\cmd.exe
C:\Users\Admin\AppData\Local\Programs\Python\Python37\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:545: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype(("qint32", np.int32, 1))
C:\Users\Admin\AppData\Local\Programs\Python\Python37\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:550: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_resource = np.dtype(("resource", np.ubyte, 1))
WARNING:tensorflow:From C:\Users\Admin\AppData\Local\Programs\Python\Python37\lib\site-packages\keras\backend\tensorflow_backend.py:4070: The name tf.nn.max_pool is dep
recated. Please use tf.nn.max_pool2d instead.

2020-12-07 17:33:41.562737: W tensorflow/core/framework/allocator.cc:107] Allocation of 419430400 exceeds 10% of system memory.
2020-12-07 17:33:42.781564: W tensorflow/core/framework/allocator.cc:107] Allocation of 419430400 exceeds 10% of system memory.
WARNING:tensorflow:From C:\Users\Admin\AppData\Local\Programs\Python\Python37\lib\site-packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables is
deprecated. Please use tf.compat.v1.global_variables instead.

Model: "sequential_1"
Layer (type)                Output Shape                Param #
-----
model_1 (Model)              (None, 204800)              23587712
dense_1 (Dense)              (None, 512)                 104858112
dropout_1 (Dropout)          (None, 512)                 0
dense_2 (Dense)              (None, 512)                 262656
dropout_2 (Dropout)          (None, 512)                 0
dense_3 (Dense)              (None, 2)                   1026
-----
Total params: 128,709,506
Trainable params: 105,121,794
Non-trainable params: 23,587,712
None
```

Fig 7.1 Output-1

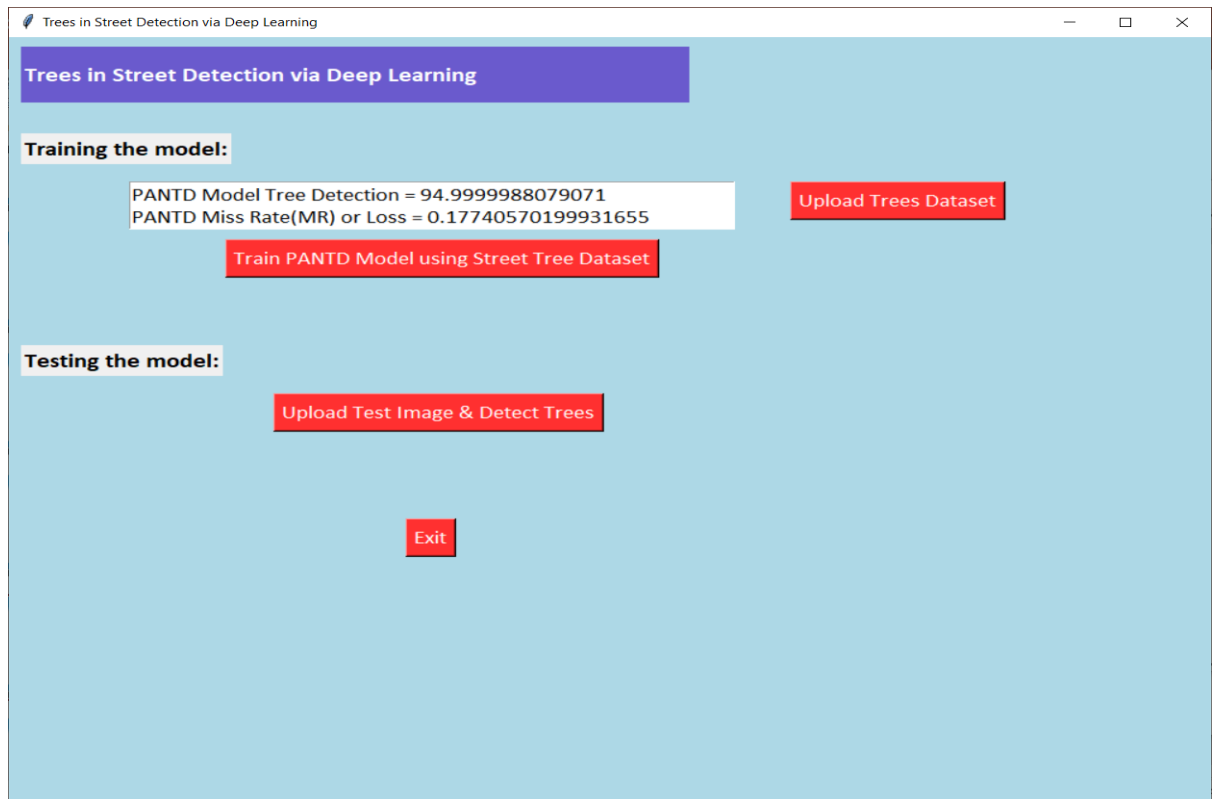


Fig 7.2 Output-2

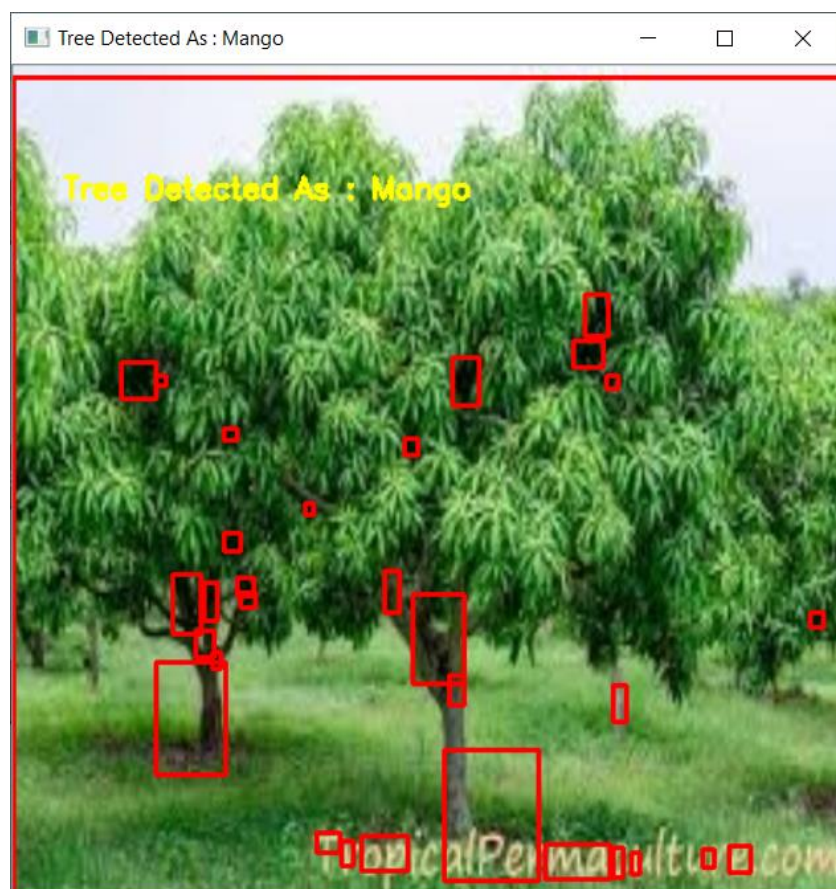


Fig 7.3 Output-3

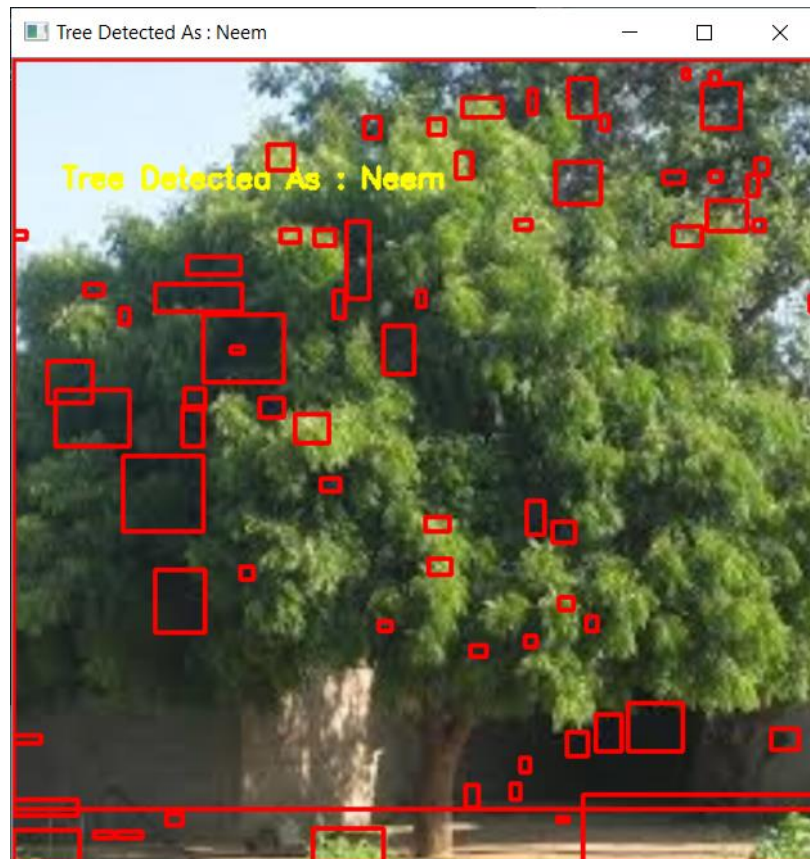


Fig 7.4 Output-4

8.CONCLUSION

This paper presents a novel framework for tree detection, leveraging state-of-the-art deep learning-based detection methods, along with two innovations in training loss definition and network module designation to handle the occlusion problem. In particular, we introduce a new training loss to suppress false detections caused by crowds, via simultaneously enforcing the proposals, associated with the same ground truth trees, to locate compactly with each other and repel those ground truth trees which are not their targets. Additionally, to effectively cope with a tree part missing, we apply the part-attention module to integrate the prior structure information of trees with occlusion prediction into the Faster R-CNN detector. The occluded parts predicted by the proposed unit would be accordingly assigned to smaller weights to release their negative influence in the subsequent classification task. We apply our tree part-attention network into the street tree detection task. The proposed method reveals high fidelity detection over the presented street tree dataset captured by street view collection vehicles.

Detailed experimental comparisons also demonstrate that our proposed framework can improve detection accuracy by a larger margin than baseline, particularly in crowded scenarios, where occlusion problems occur. However, owing to the difference between the same tree species and the similarity within the diverse tree species, street tree classification is still a challenging problem. A possible future direction would be designing efficient classification network to tell these trees apart according to their species after they are detected. Furthermore, the proposed preprocess method is highly related with the set of images used in this paper. It may lack the ability of generalization to other image dataset. Thus, a more general method for the low-illumination problem would be another research direction of future work.

9.REFERENCES

- [1] J. Secord and A. Zakhori, "Tree detection in urban regions using aerial lidar and image data," IEEE Geoscience and Remote Sensing Letters, vol. 4, no. 2, pp. 196–200, 2007.
- [2] W. Ouyang and X. Wang, "Joint deep learning for pedestrian detection," in Proceedings of the IEEE International Conference on Computer Vision, 2013, pp. 2056–2063.
- [3] S. Malek, Y. Bazi, N. Alajlan, H. AlHichri, and F. Melgani, "Efficient framework for palm tree detection in uav images," IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 7, no. 12, pp. 4692–4703, 2014.
- [4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in Advances in neural information processing systems, 2015, pp. 91–99.
- [5] U. Shah, R. Khawad, and K. M. Krishna, "Detecting, localizing, and recognizing trees with a monocular mav: Towards preventing deforestation," in Robotics and Automation (ICRA), 2017 IEEE International Conference on. IEEE, 2017, pp. 1982–1987.

Websites referred

- [1] <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neuralnetworks-the-eli5-way-3bd2b1164a53>
- [2] <https://deeptai.org/machine-learning-glossary-and-terms/stride>
- [3] https://en.wikipedia.org/wiki/Convolutional_neural_network
- [4] <https://cloud.google.com/vision/docs/ocr>
- [5] <https://www.tensorflow.org/>
- [6] <https://stackoverflow.com/questions/24385714/detect-text-region-in-image-using-opencv>