A

Major Project Report

On

IDENTIFICATION OF WEEDS FROM CROPS USING CONVOLUTIONAL NEURAL NETWORKS

(Submitted in partial fulfillment of the requirements for the award of Degree)

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

M.BALA SRAVANI (187R1A05M1)

L. SHIVANI GOUD (187R1A05J2)

B. SHIVA (197R5A0521)

Under the Guidance
Of

K. RANJITH REDDY

(Assistant Professor)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CMR TECHNICAL CAMPUS

UGC AUTONOMOUS

(Accredited by NAAC, NBA, Permanently Affiliated to JNTUH, Approved by AICTE, New Delhi)

Recognized Under Section 2(f) & 12(B) of the UGCAct.1956,

Kandlakoya (V), Medchal Road, Hyderabad-501401.

2018-22

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitled "**IDENTIFICATION OF WEEDS FROM CROPS USING CONVOLUTIONAL NEURAL NETWORKS"** being submitted by **M.BALA SRAVANI (187R1A05M1), L.SHIVANI GOUD (187R1A05J2) & B.SHIVA (197R5A0521)** in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science and Engineering to the Jawaharlal Nehru Technological University Hyderabad, is a record of bonafide work carried out by him/her under our guidance and supervision during the year 2021-22.

The results embodied in this have not been submitted to any other University or Institute for the award of any degree or diploma.

K. RANJITH REDDY
Assistant Professor
INTERNAL GUIDE

Dr. A. Raji Reddy
DIRECTOR

Dr. K. Srujan Raju
HOD

EXTERNAL EXAMINER

Submitted for viva voice Examination held on________________________

ACKNOWLEDGEMENT

Apart from the efforts of us, the success of any project depends largely on the encouragement andguidelines of many others. We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project. We take this opportunity to express my profound gratitude and deep regard to my guide.

K. RANJITH REDDY, Assistant Professor, for his exemplary guidance, monitoring, and constantencouragement throughout the project work. The blessing, help and guidance given by him shall carry us a long way in the journey of life on which we are about to embark. We also take this opportunity to express a deep sense of gratitude to the Project Review Committee (PRC) **Dr. M. Varaprasad Rao, Mr. J. Narasimha Rao, Dr. T. S. Mastan Rao, Dr. Suwarna Gothane, Mr. A.Uday Kiran, Mr. A. Kiran Kumar, Mrs. G. Latha** for their cordial support, valuable informationand guidance, which helped us in completing this task through various stages.

We are also thankful to the head of the Department **Dr. K. Srujan Raju** for providing excellent infrastructure and a nice atmosphere for completing the project successfully. We are obliged to **Dr. A. Raji Reddy,** Director for being cooperative throughout the course of this project. We also express our sincere gratitude to Sri. **Ch. Gopal Reddy,** Chairman for providing excellent infrastructure and a nice atmosphere throughout the course of this project.

The guidance and support received from all the members of **CMR Technical Campus** who contributed to the completion of the project. We are grateful for their constant support and help. Finally, we would like to take this opportunity to thank our family for their constant encouragement, without which this assignment would not be completed. We sincerely acknowledge and thank all those who gave support directly and indirectly in the completion of thisproject.

M.BALA SRAVANI (187R1A05M1)

L. SHIVANI GOUD (187R1A05J2)

B. SHIVA (197R5A0521)

ABSTRACT

Weeds are very annoying for farmers and also not very good for the crops. Weeds are competitive, fighting agriculture crops or lawn grass for water, light, nutrients and space. Most are quick growers and will take over many of the areas in which you find them Its existence might damage the growth of the crops. Therefore, weed control is very important for farmers. Farmers need to ensure their agricultural fields are free from weeds for at least once a week, whether they need to spray weeds herbicides to their plantation or remove it using tools or manually. The aim of this project is to build an application that can identify the weeds in a given video. An automated image classification application has been designed to differentiate between weeds and crops. For the image classification method, we employ the convolutional neural network algorithm to process the image of the object.For the weed detection algorithm, the convolutional neural network is suitable for this project because CNN are most applied to analyzing visual image. CNN use variation of multilayer perceptron's designed to use minimal preprocessing.

We will be implementing a web-based application where the user will upload the video of the crops and the system will detect the weeds in the given video. This can also made into an attachment to robots where it can dig out the weeds for us. But our primary objective is to make a system that can detect the weeds.

LIST OF FIGURES

LIST OF SCREENSHOTS

TABLE OF CONTENTS

1. INTRODUCTION

1. INTRODUCTION

1.1 PROJECT SCOPE

Weed plants can be seen more in a plantation because the plantation have all nutrition and water for the weeds to grow. This is a very big problem to the farmer because the weeds consume a large amount of nutrition and water and the other plants cannot grow in a good shape. Therefore, weed detection system is important in agriculture. This study presents a method for detecting weeds in crops using convolutional neural networks.

1.2 PROJECT PURPOSE

The purpose of this study is to effectively detect the weeds in crops using convolution neural networks. As the weed plant consumes a large amount of nutrition and water, the other crop plants cannot grow in a good shape. Weeds compete with crops for water, light, nutrients, and space, therefore it is very important to design a system that can detect the weeds in the crops. Our primary objective is to provide a system that can detect weeds.

1.3 PROJECT FEATURES

The main feature of the system is to propose a general and effective approach to detect the weeds in the crop. An automated image classification system has been designed to differentiate between weeds and crops. For the weed detection algorithm, the convolutional neural network is suitable because CNN are most applied for analyzing visual image. The user will upload the video containing the crop field in the application and the system gives the output video containing identified weeds in the given video.

2. SYSTEM ANALYSIS

2.SYSTEM ANALYSIS

SYSTEM ANALYSIS:

System Analysis is an important phase in the system development process. The system is studied to the minute details and analyzed. The system analyst plays an important role as an interrogator and dwells deep into the working of the present system. In analysis, a detailed study of these operations performed by the system and their relationships within and outside the system is done. A key question considered here is, "what must be done to solve the problem?" The system is viewed as a whole and the inputs to the system are identified. Once analysis is completed the analyst has a firm understanding of what is to be done.

2.1 PROBLEM DEFINITION

Building This project is primarily concerned with detecting the weeds in the crops. The suggested approach attempts to use convolution neural network to obtain the optimal function in the most effective manner.

- The main objective is to detect weeds in the crops.
- To create a system which uses the given dataset to train and later can detect the weeds in new data given.

2.2 EXISTING SYSTEM

Existing Systems are available which are based on manual weed detection, in contrast with the weed detection using machine learning, which has been traditional way of visual inspection. There are also systems where weeds are identified in the pictures uploaded by the user. But uploading a lot of pictures is tedious.

2.2.1 LIMITATIONS OF EXISTING SYSTEM

- Entering the large number of pictures to the system is a tedious work
- Clicking many pictures and uploading them can be tedious
- It is very time consuming.

2.3 PROPOSED SYSTEM

We will be building a system that uses deep learning techniques to identify weeds. The main feature of the system is to propose a general and effective approach to detect the weeds in the crop. An automated image classification system has been designed to differentiate between weeds and crops. For the weeds detection algorithm, the convolutional neural network is suitable because CNN are most commonly applied to analyzing visual image. The User will provide the video of his crop plants including weed crops as an input. Our system will detect the weeds in the input and gives the output video containing identified weeds in the given crop video. One of the important steps in increasing the yield is to treat the weeds as it is directly associated with crop yield. We will be using advanced algorithms like Mask R-CNN built on ResNet-101 and FPN for our system and we will also compare with other older algorithms to prove higher accuracy of our system.

2.3.1 ADVANTAGES OF PROPOSED SYSTEM

The system is very simple to design and implement. The system requires GTX 970 graphics card and the system will work in almost all configurations. It has the following features :-

- Better services
- Ensure data accuracies.
- Greater efficiency
- Minimum time needed for the various processing

2.4 FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project with some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. Three key considerations involved in the feasibility analysis are

- Economical feasibility
- Technical feasibility
- Social feasibility

2.4.1 ECONOMIC FEASIBILITY

The developing system must be justified by cost and benefit. Criteria to ensure that effort is concentrated on a project, which will give the best return at the earliest. One of the factors, which affect the development of a new system, is the cost it would require. The following are some of the important financial questions asked during preliminary investigation:

- They conduct a full system investigation.
- The cost of the hardware and software.
- The benefits in the form of reduced costs or fewer costly errors.

Since the system is developed as part of the project work, there is no manual cost to spend for the proposed system. Also, all the resources are already available, which indicates that the system is economically possible for the development.

2.4.2 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

2.4.3 BEHAVIORAL FEASIBILITY

This includes the following questions:

- Is there sufficient support for the users?
- Will the proposed system cause harm?

The project would be beneficial because it satisfies the objectives of detecting the weeds from the data given by the user. All behavioral aspects are considered carefully and conclude that the project is behaviorally feasible.

2.5 HARDWARE AND SOFTWARE REQUIREMENTS

2.5.1 HARDWARE REQUIREMENTS

Hardware interfaces specify the logical characteristics of each interface between the software product and the hardware components of the system. The following are some hardware requirements.

• Graphic Card	:	NVIDIA GeForce MX350
• Processor	:	11th Gen Intel(R) Core(TM) i5-1135G7
• RAM	:	Min 4GB or Above
• Hard disk	:	Min 100 GB

2.5.2 SOFTWARE REQUIREMENTS

Software Requirements specify the logical characteristics of each interface and software components of the system. The following are some software requirements:

• Operating System	:	Windows 10
• Technology	:	Python 3.6
• IDE	:	Google colabs

3.ARCHITECTURE

3. ARCHITECTURE

3.1 PROJECT ARCHITECTURE

This project architecture shows the procedure followed to detect the weeds in the crop using convolution neural network.

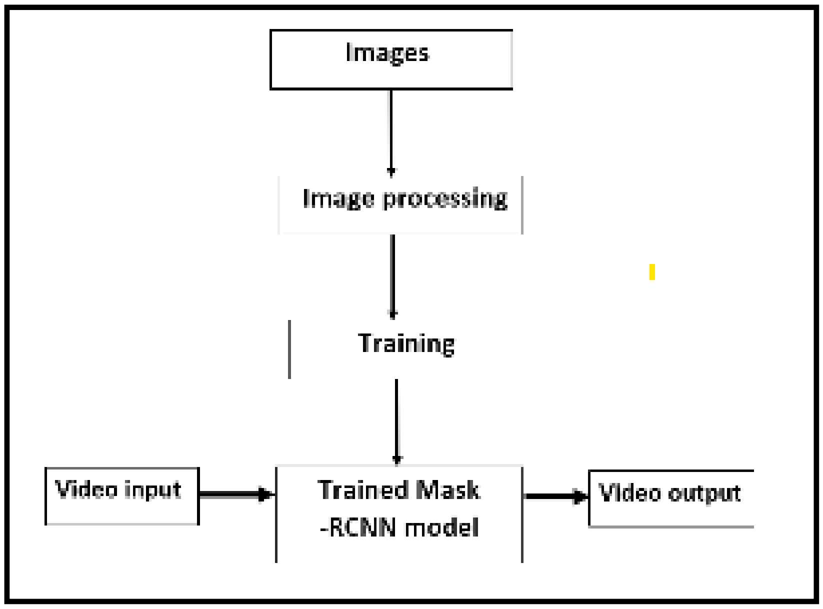


Figure 3.1 : Project architecture of Identification of weeds from Crops

3.2 MODULE DESCRIPTION

In the proposed work we used Mask R-CNN built on FPN(Feature Pyramid Networks) and ResNet101. Feature Pyramid Network (FPN) is a feature extractor designed for accuracy and speed. It generates multiple feature map layers (multi-scale feature maps) with better quality information than the regular feature pyramid for object detection. ResNet-101 is a convolutional neural network that is 101 layers deep. The network can learn rich feature representations for a wide range of images.

The key element of Mask R-CNN is the pixel-to-pixel alignment, which is the main missing piece of Fast/Faster R-CNN. Mask R-CNN adopts the same two-stage procedure with an identical first stage (which is RPN). In the second stage, in parallel to predicting the class and box offset, Mask R-CNN also outputs a binary mask for each RoI. This is in contrast to most recent systems, where classification depends on mask predictions.

Furthermore, Mask R-CNN is simple to implement and train given the Faster R-CNN framework, which facilitates a wide range of flexible architecture designs. Additionally, the mask branch only adds a small computational overhead, enabling a fast system and rapid experimentation.

Mask R-CNN is basically an extension of Faster R-CNN. Faster R-CNN is widely used for object detection tasks. For a given image, it returns the class label and bounding box coordinates for each object in the image. The Mask R-CNN framework is built on top of Faster R-CNN. So, for a given image, Mask R-CNN, in addition to the class label and bounding box coordinates for each object, will also return the object mask.

Backbone Model

Similar to the ConvNet that we use in Faster R-CNN to extract feature maps from the image, we use the ResNet 101 architecture to extract features from the images in Mask R-CNN. So, the first step is to take an image and extract features using the ResNet 101 architecture. These features act as an input for the next layer.

Region Propose Network(RPM)

Now, we take the feature maps obtained in the previous step and apply a region proposal network (RPM). This basically predicts if an object is present in that region (or not). In this step, we get those regions or feature maps which the model predicts contain some object.

Region Of Interest (ROI)

The regions obtained from the RPN might be of different shapes, right? Hence, we apply a pooling layer and convert all the regions to the same shape. Next, these regions are passed through a fully connected network so that the class label and bounding boxes are predicted.

Till this point, the steps are almost similar to how Faster R-CNN works. Now comes the difference between the two frameworks. In addition to this, Mask R-CNN also generates the segmentation mask.

For that, we first compute the region of interest so that the computation time can be reduced. For all the predicted regions, we compute the Intersection over Union (IoU) with the ground truth boxes. We can computer IoU like this:

IoU = Area of the intersection / Area of the union

Now, only if the IoU is greater than or equal to 0.5, we consider that as a region of interest. Otherwise, we neglect that particular region. We do this for all the regions and then select only a set of regions for which the IoU is greater than 0.5.

Segmentation Mask

Once we have the RoIs based on the IoU values, we can add a mask branch to the existing architecture. This returns the segmentation mask for each region that contains an object. It returns a mask of size 28 X 28 for each region which is then scaled up for inference.

This is the final step in Mask R-CNN where we predict the masks for all the objects in the image.

Images: In our work we have taken 80 foreground images of various types of weeds along with 37background images of various fields.

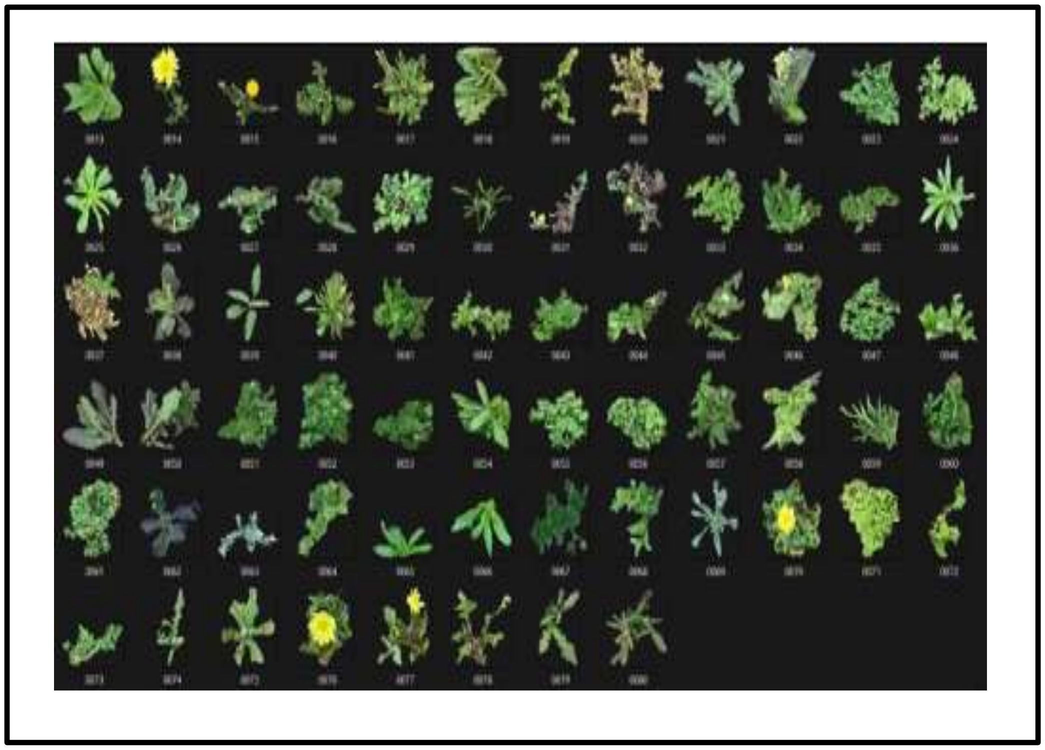


Figure 3.2.1: Foreground Images

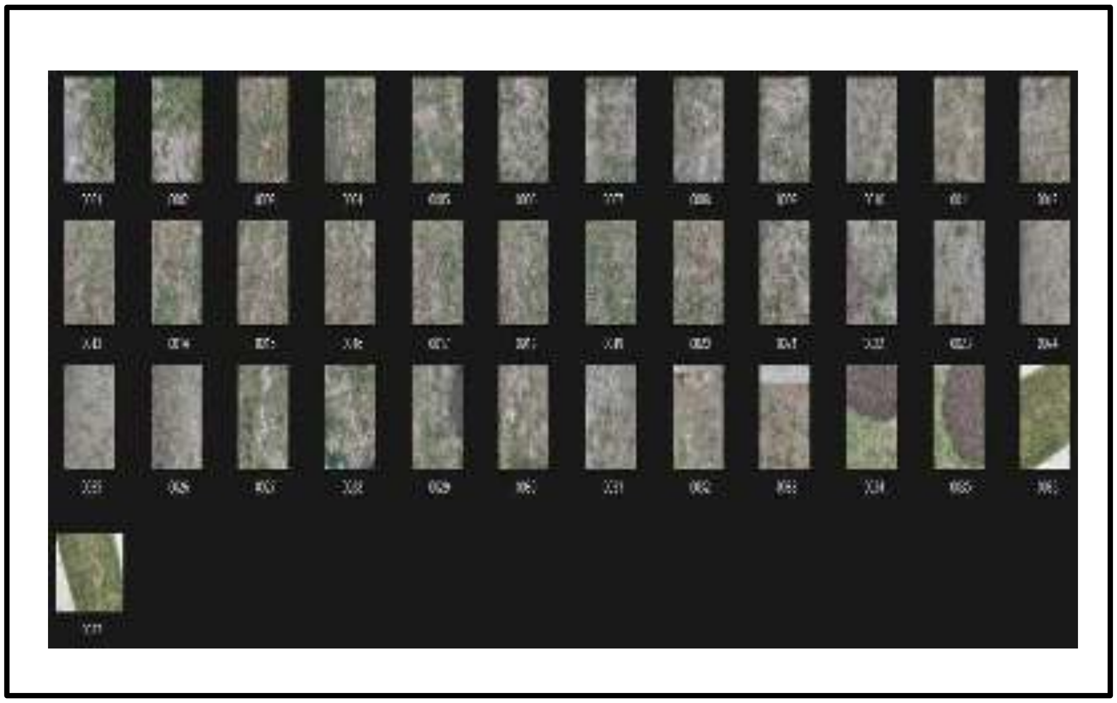


Figure 3.2.2 : Background Images

Image Preprocessing: Here, composed images are created i.e., foreground images are placed on given different backgrounds and mask images are created accordingly along with mask definition file (.json file). Mask definition file contains all the details about mask like categories, subcategories, name, id, etc.

Preprocessing an image is a must so that programs work properly to give the expected output. The aim of pre-processing is to improve the quality of the image so that we can analyze it in a better way. By preprocessing we can suppress undesired distortions and enhance some features which are necessary for the application we are working for.

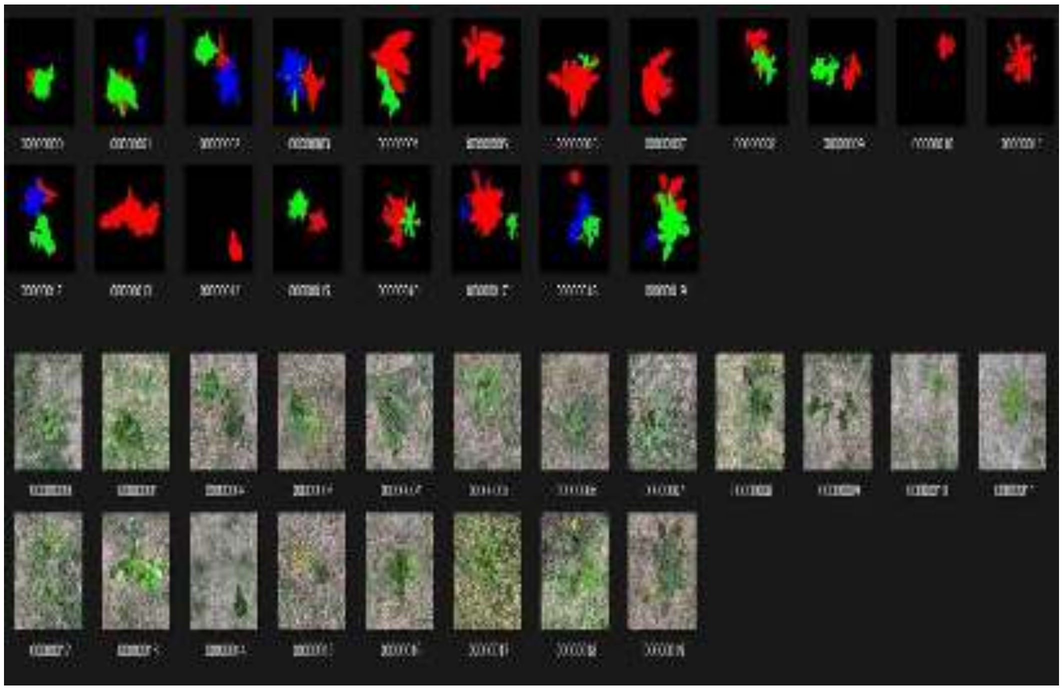


Figure 3.2.3 : Example of Training Data along with Mask

Trained Mask R-CNN model:

Mask R-CNN is an extension of Faster R-CNN. Mask R-CNN has an additional branch for predicting segmentation masks on each Region of Interest (RoI) in a pixel-to pixel manner.

Mask R-CNN model is divided into two parts.

- Region proposal network (RPN) to proposes candidate object bounding boxes.
- Binary mask classifier to generate mask for every class

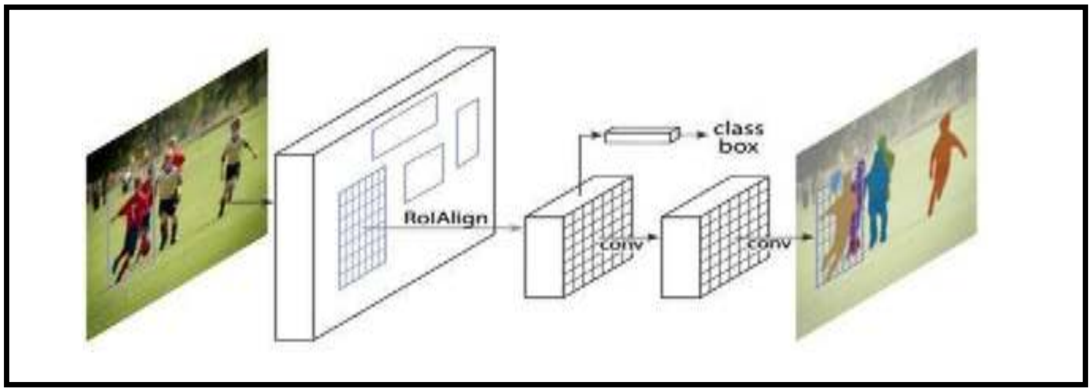


Figure 3.2.4 : Mask R-CNN

- ❖ Image is run through the CNN to generate the feature maps.
- ❖ Region Proposal Network (RPN) uses a CNN to generate the multiple Region of Interest (RoI) using a lightweight binary classifier. It does this using 9 anchors boxes over the image. The classifier returns object/no-object scores. Non-Max suppression is applied to Anchors with high objectness score

- The RoI Align network outputs multiple bounding boxes rather than a single definite one and warp them into a fixed dimension.
- Warped features are then fed into fully connected layers to make classification using softmax and boundary box prediction is further refined using the regression model
- Warped features are also fed into Mask classifier, which consists of two CNN's to output a binary mask for each RoI. Mask Classifier allows the network to generate masks for every class without competition among classes.

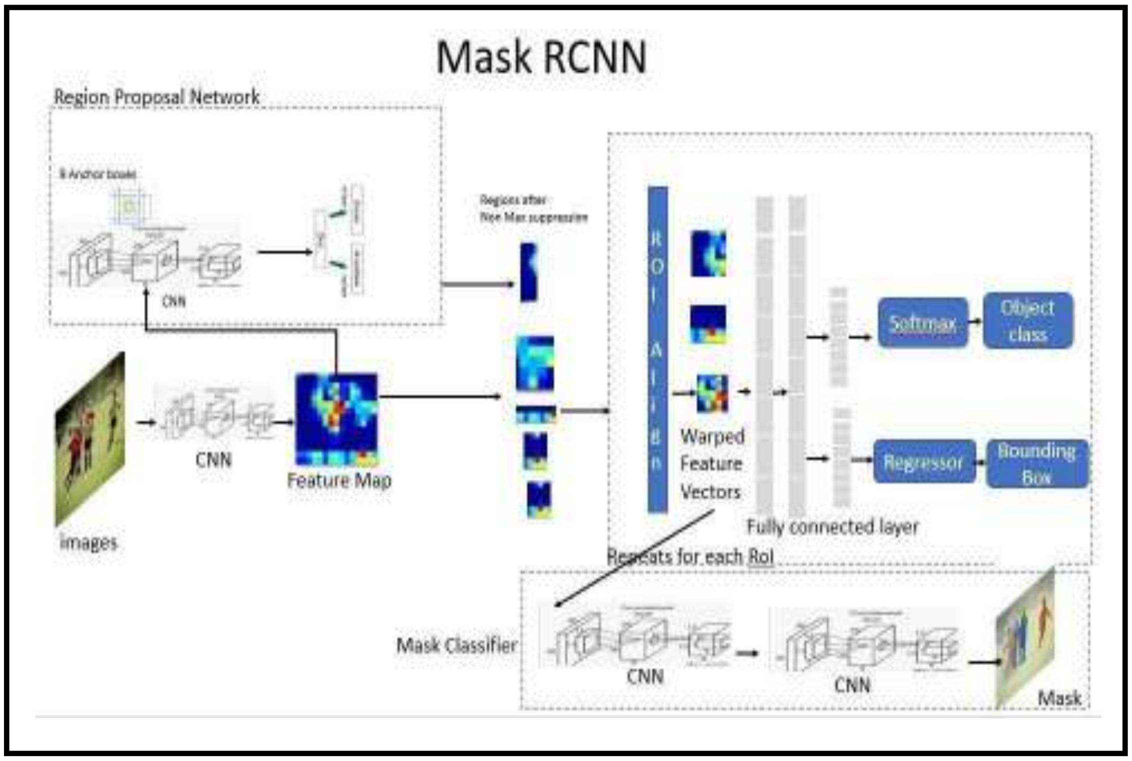


Figure 3.2.5 : Mask R-CNN on RPN

Step by Step Detection

1. Anchor sorting and filtering

Visualizes every step of the first stage Region Proposal Network and displays positive and negative anchors along with anchor box refinement.

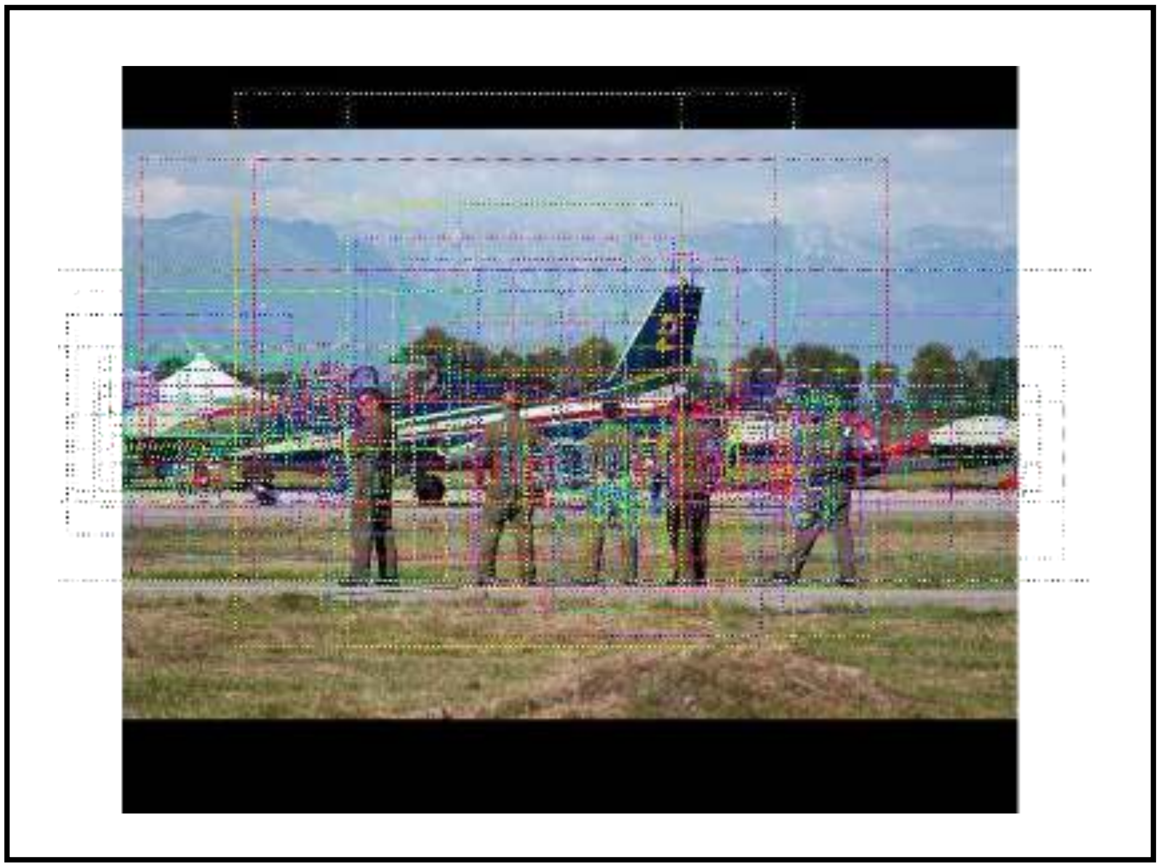


Figure 3.2.6 : Image with multiple anchor boxes

2. Bounding Box Refinement

This is an example of final detection boxes (dotted lines) and the refinement applied to them (solid lines) in the second stage. Non-Max Suppression will remove all bounding boxes where IoU is less than or equal to 0.5. It picks the bounding box with the highest value for IoU and suppress the other bounding boxes for identifying the same object.

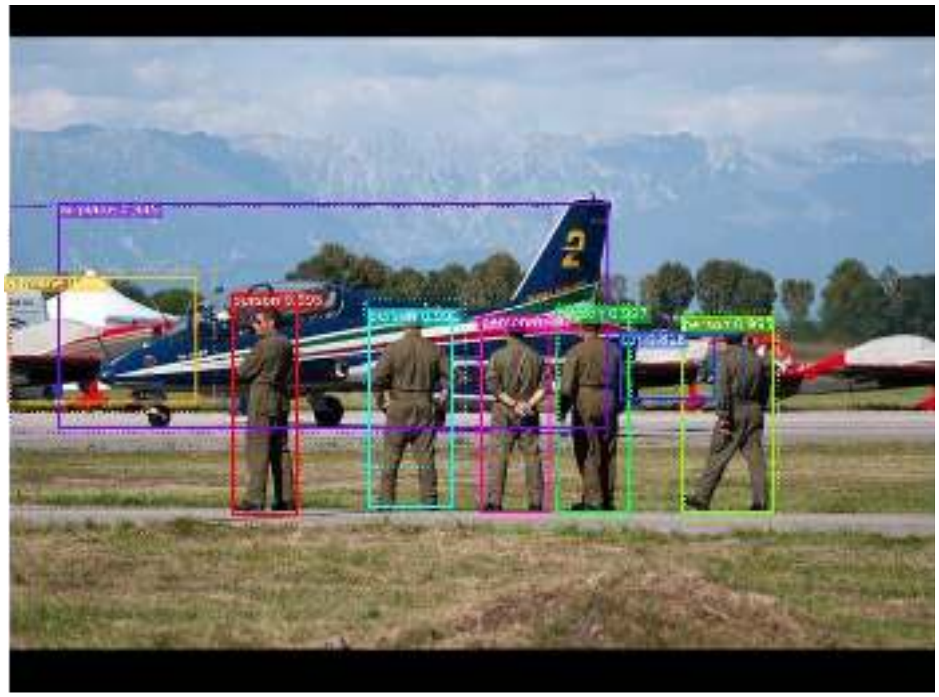


Figure 3.2.7 : Bounding box refinement

3. Mask Generation

Masks are generated. These then get scaled and placed on the image in the right location.

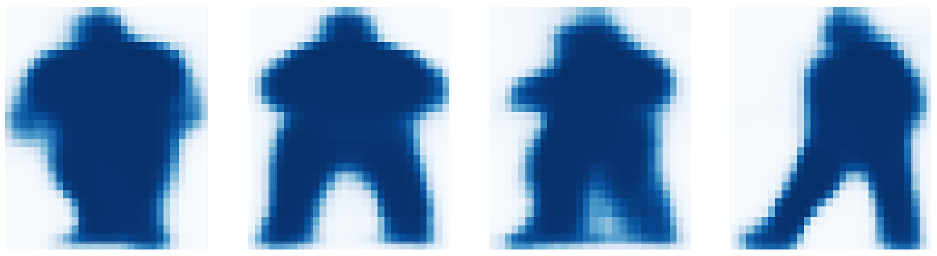
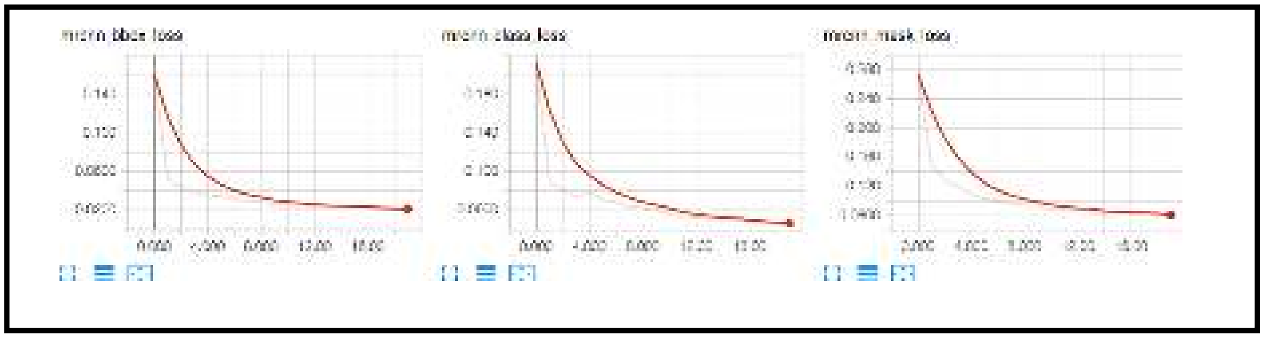


Figure 3.2.8 : Examples of generated masks

4. Logging to TensorBoard

TensorBoard is a great debugging and visualization tool. The model is configured to log losses and save weights at the end of every epoch.



Graph 3.2.9 : Graphs showing loss at every epoch

5. Composing the different pieces into the final result

This is the result we obtain after identifying the objects in the given image. We can have some false positives here and there. We can reduce them by increasing the training hours.

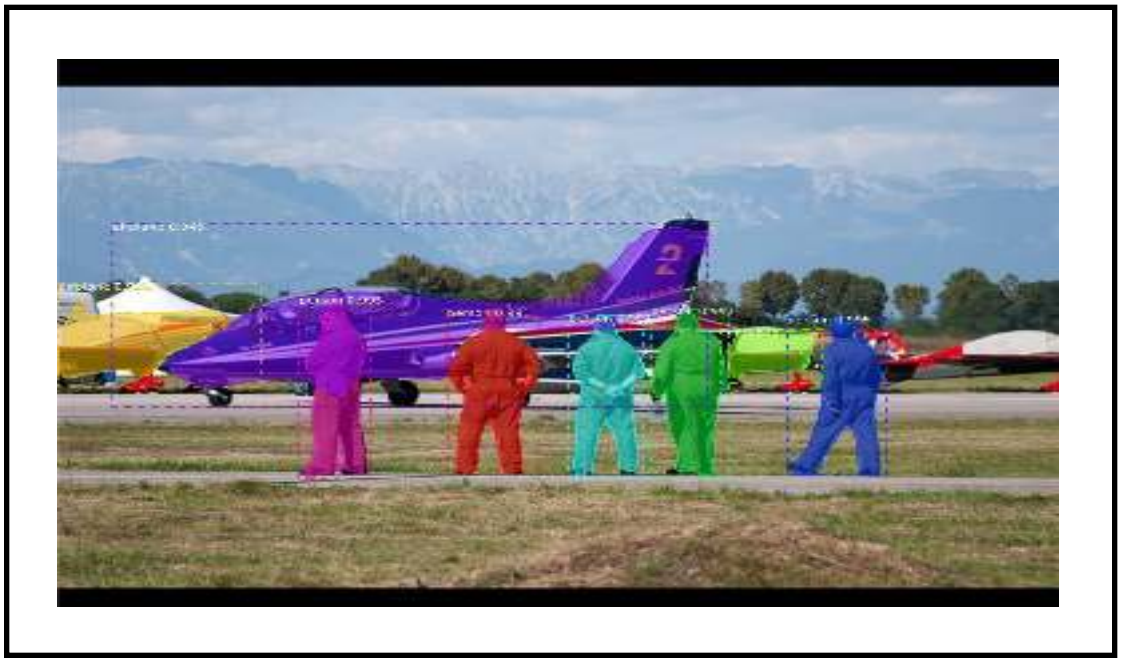


Figure 3.2.10 : Image with identified object

Steps to implement Mask R-CNN:

We will be using the mask R-CNN framework created by the Data scientists and researchers at Facebook AI Research (FAIR).

Let’s have a look at the steps which we will follow to perform image segmentation using Mask R-CNN.

Step 1: Clone the repository

First, we will clone the mask rcnn repository which has the architecture for Mask R-CNN. Use the following command to clone the repository:

git clone https://github.com/matterport/Mask_RCNN.git

Once this is done, we need to install the dependencies required by Mask R-CNN.

Step 2: Install the dependencies

Here is a list of all the dependencies for Mask R-CNN:

- numpy
- scipy
- Pillow
- cython
- matplotlib
- scikit-image
- tensorflow>=1.3.0
- keras>=2.0.8
- opencv-python
- h5py
- imgaug
- IPython

You must install all these dependencies before using the Mask R-CNN framework.

Step 3: Download the pre-trained weights (trained on MS COCO)

Next, we need to download the pretrained weights. These weights are obtained from a model that was trained on the MS COCO dataset. Once you have downloaded the weights, paste this file in the samples folder of the Mask RCNN repository that we cloned in step 1.

Step 4: Predicting for our image

Finally, we will use the Mask R-CNN architecture and the pretrained weights to generate predictions for our own images.

We will implement all these things in Python and then generate the masks along with the classes and bounding boxes for objects in our images.

Advantages of mask R-CNN:

- Simplicity: Mask R-CNN is simple to train.
- Performance: Mask R-CNN outperforms all existing, single-model entries on every task.
- Efficiency: The method is very efficient and adds only a small overhead to Faster R-CNN.
- Flexibility: Mask R-CNN is easy to generalize to other tasks. For example, it is possible to use Mask R-CNN for human pose estimation in the same framework.

Video Input: The path of the video containing weeds is given as the input to thetrained Mask R-CNN model.

Video Output: Weeds are detected in the given video input using the algorithm and theoutput video of identified weeds is sent to the given path.

3.3 USE CASE DIAGRAM

In the use case diagram, we have basically two actors who are the user and the system. The user uploads the video of crops and the system detects the weeds in thegiven video.

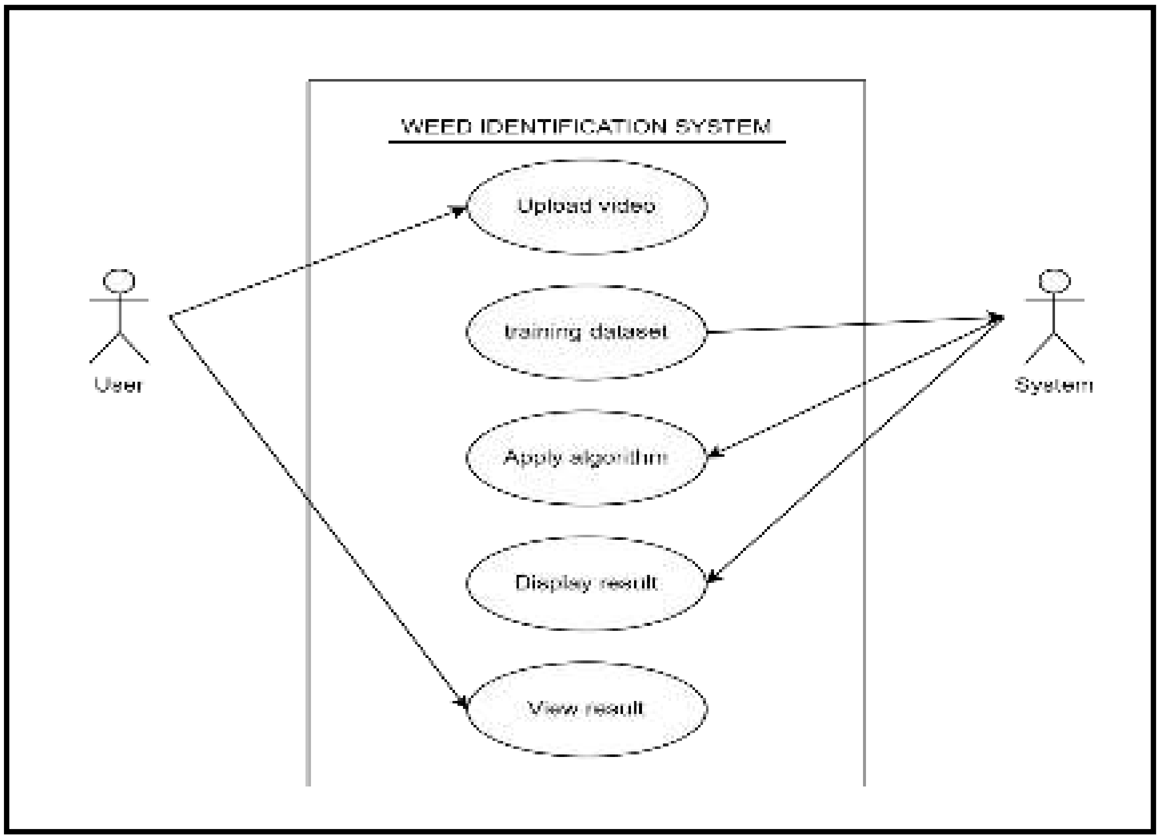


Figure 3.3 : Use case diagram for Identification of weeds from crops

3.4 CLASS DIAGRAM

Class Diagram is a collection of classes and objects.

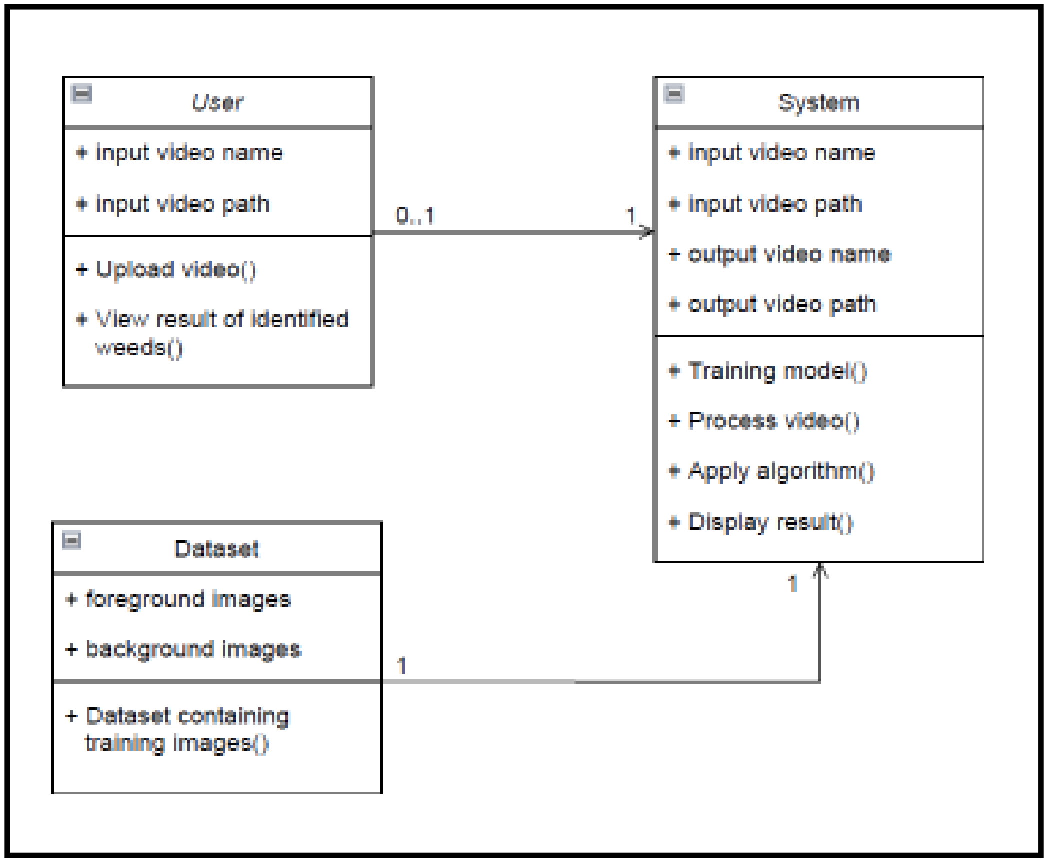


Figure 3.4 : Class Diagram for Identification of weeds from crops

3.5 SEQUENCE DIAGRAM

It describes the object interactions arranged in a time sequence .

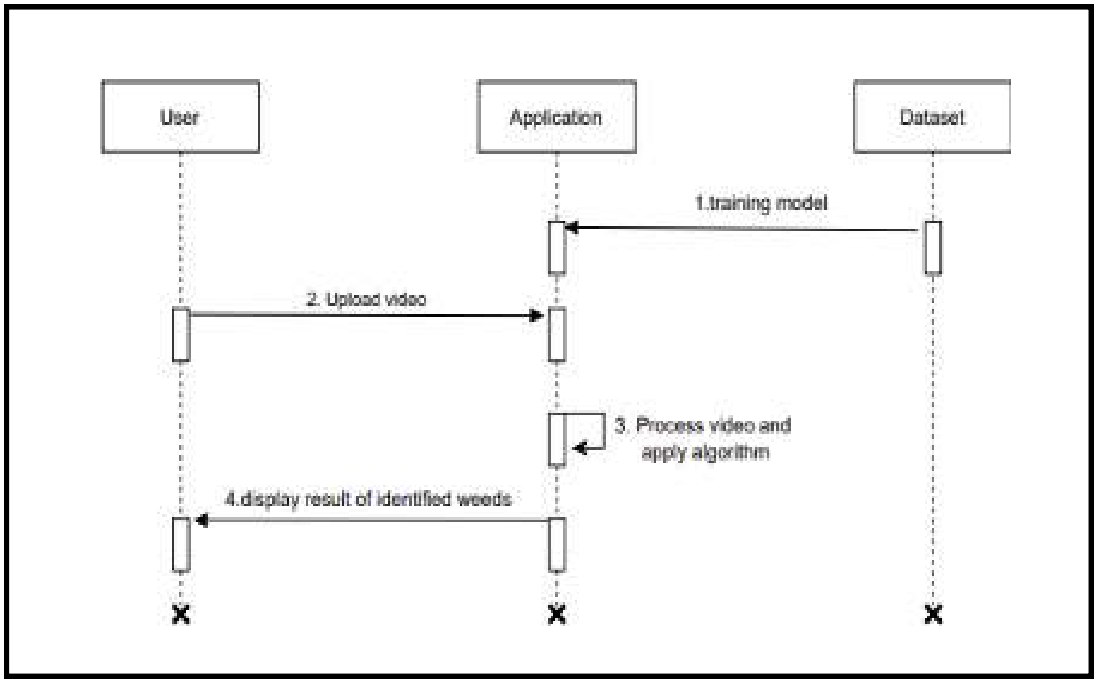

User
Application
Dataset
1.training model
2. Upload video
3. Process video and apply algorithm
4.display result of identified weeds

Figure 3.5 : Sequence diagram for Identification of weeds from crops

3.6 ACTIVITY DIAGRAM

It describes the flow of activity states.

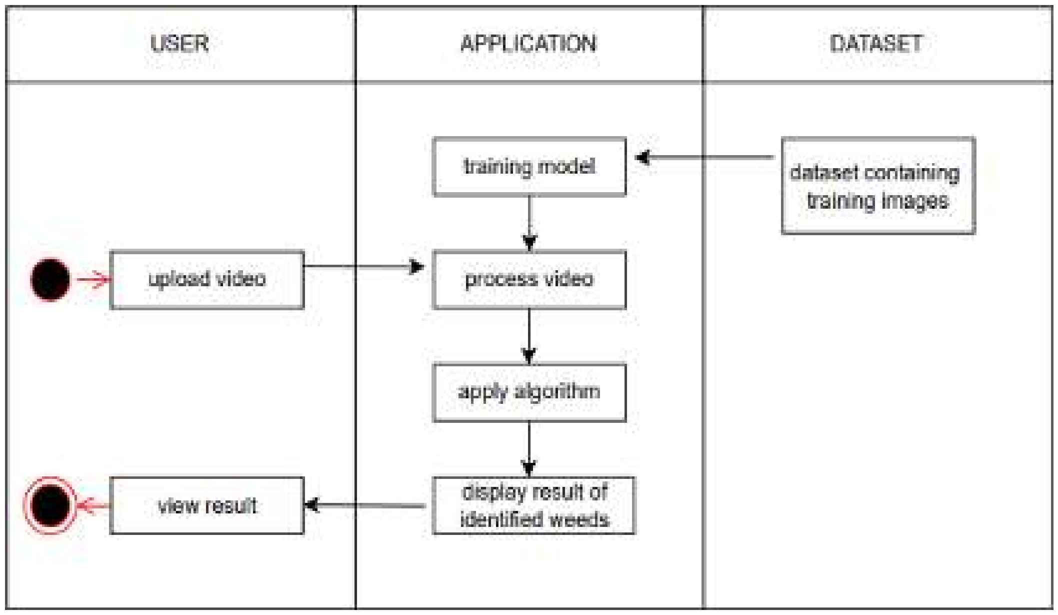


Figure 3.6 : Activity diagram for Identification of weeds from crops

4.IMPLEMENTATION

4. IMPLEMENTATION

4.1 SAMPLE CODE

A)MASK_RCNN.IPYNB

```
%load_ext autoreload
import os
import sys
import json
import numpy as np
import time
from PIL import Image, ImageDraw A)MASK_RCNN.IPYNB
%load_ext autoreload
import os
import sys
import json
import numpy as np
import time
from PIL import Image, ImageDraw
from pathlib import Path

# Set the ROOT_DIR variable to the root directory of the Mask_RCNN git repo

ROOT_DIR = 'C:/Users/HPPP/anaconda3/cocosynth-master/Mask_RCNN-master/'
assert os.path.exists(ROOT_DIR), 'ROOT_DIR does not exist. Did you forget to read
the instructions above? ;)'

# Directory to save logs and trained model

MODEL_DIR = os.path.join(ROOT_DIR, "logs")

# Local path to trained weights file

COCO_MODEL_PATH = os.path.join(ROOT_DIR, "mask_rcnn_coco.h5")

# Download COCO trained weights from Releases if needed

if not os.path.exists(COCO_MODEL_PATH):
 utils.download_trained_weights(COCO_MODEL_PATH)

#CONFIGURATION

class CocoSynthConfig(Config):
 """Configuration for training on the box_synthetic dataset.
 Derives from the base Config class and overrides specific values.
 """
 # Give the configuration a recognizable name
 NAME = "cocosynth_dataset"

# Train on 1 GPU and 1 image per GPU. Batch size is 52 (GPUs * images/GPU).

 GPU_COUNT = 1
 IMAGES_PER_GPU = 1

 # Number of classes (including background)

 NUM_CLASSES = 2 # background + 14 box types

 # All of our training images are 512x512

 IMAGE_MIN_DIM = 512
 IMAGE_MAX_DIM = 512

 # You can experiment with this number to see if it improves training

 STEPS_PER_EPOCH = 1000

# This is how often validation is run. If you are using too much hard drive space
 # on saved models (in the MODEL_DIR), try making this value larger.

 VALIDATION_STEPS = 5

 # Matterport originally used resnet101, but I downsized to fit it on my graphics card

 BACKBONE = 'resnet101'

 # To be honest, I haven't taken the time to figure out what these do

 RPN_ANCHOR_SCALES = (8, 16, 32, 64, 128)
 TRAIN_ROIS_PER_IMAGE = 32
 MAX_GT_INSTANCES = 50
 POST_NMS_ROIS_INFERENCE = 500
 POST_NMS_ROIS_TRAINING = 1000
config = CocoSynthConfig()
config.display()

#DEFINE A DATASET
class CocoLikeDataset(utils.Dataset):
 """ Generates a COCO-like dataset, i.e. an image dataset annotated in the style of the
COCO dataset.
 See http://cocodataset.org/#home for more information.
 """
 def load_data(self, annotation_json, images_dir):
 """ Load the coco-like dataset from json
 Args:
 annotation_json: The path to the coco annotations json file
 images_dir: The directory holding the images referred to by the json file
 """

 # Load json from file

 json_file = open(annotation_json)
 coco_json = json.load(json_file)
 json_file.close()

# Add the class names using the base method from utils.Dataset

 source_name = "coco_like"
 for category in coco_json['categories']:
 class_id = category['id']
 class_name = category['name']
 if class_id < 1:
 print('Error: Class id for "{}" cannot be less than one. (0 is reserved for the
 background)'.format(class_name))
 return
 self.add_class(source_name, class_id, class_name)

 # Get all annotations

 annotations = {}
 for annotation in coco_json['annotations']:
 image_id = annotation['image_id']
 if image_id not in annotations:
 annotations[image_id] = []
 annotations[image_id].append(annotation)

 # Get all images and add them to the dataset

 seen_images = {}
 for image in coco_json['images']:
 image_id = image['id']
 if image_id in seen_images:
 print("Warning: Skipping duplicate image id: {}".format(image))
 else:
 seen_images[image_id] = image
 try:
 image_file_name = image['file_name']
 image_width = image['width']
 image_height = image['height']
 except KeyError as key:
 print("Warning: Skipping image (id: {}) with missing key: {}".format(image_id,
key))
  image_path = os.path.abspath(os.path.join(images_dir, image_file_name))
 image_annotations = annotations[image_id]

# Add the image using the base method from utils.Dataset
```

```
self.add_image(
source=source_name,
image_id=image_id,
path=image_path,
width=image_width,
height=image_height,
annotations=image_annotations
)

def load_mask(self, image_id):
""" Load instance masks for the given image.
MaskRCNN expects masks in the form of a bitmap [height, width, instances].
Args:
image_id: The id of the image to load masks for
Returns:
masks: A bool array of shape [height, width, instance count] with
one mask per instance.
class_ids: a 1D array of class IDs of the instance masks.
"""
image_info = self.image_info[image_id]
annotations = image_info['annotations']
instance_masks = []
class_ids = []
for annotation in annotations:
class_id = annotation['category_id']
mask = Image.new('1', (image_info['width'], image_info['height']))
mask_draw = ImageDraw.ImageDraw(mask, '1')
for segmentation in annotation['segmentation']:
mask_draw.polygon(segmentation, fill=1)
bool_array = np.array(mask) > 0
instance_masks.append(bool_array)
class_ids.append(class_id)
mask = np.dstack(instance_masks)
class_ids = np.array(class_ids, dtype=np.int32)
 return mask, class_ids

#CREATE THE TRAINING AND VALIDATION DATASETS

dataset_train = CocoLikeDataset()
dataset_train.load_data('C:/Users/HPPP/anaconda3/cocosynthmaster/datasets/weeds/
output/training/coco_instances.json',
 'C:/Users/HPPP/anaconda3/cocosynth-master/datasets/weeds/output/training/images')
dataset_train.prepare()
dataset_val = CocoLikeDataset()
dataset_val.load_data('C:/Users/HPPP/anaconda3/cocosynthmaster/datasets/weeds/out
put/val/coco_instances.json',
 'C:/Users/HPPP/anaconda3/cocosynth-master/datasets/weeds/output/val/images')
dataset_val.prepare()
```

```
#DISPLAY FEW IMAGES FROM TRAIN AND VAL DATASETS

for name, dataset in [('training', dataset_train), ('validation', dataset_val)]:
 print(f'Displaying examples from {name} dataset:')
  image_ids = np.random.choice(dataset.image_ids, 3)
 for image_id in image_ids:
 image = dataset.load_image(image_id)
 mask, class_ids = dataset.load_mask(image_id)
 visualize.display_top_masks(image, mask, class_ids, dataset.class_names)

#CREATE THE TRAINING MODEL AND TRAIN
# Create model in training mode

model = modellib.MaskRCNN(mode="training", config=config,
 model_dir=MODEL_DIR)

# Which weights to start with?
init_with = "coco" # imagenet, coco, or last
if init_with == "imagenet":
 model.load_weights(model.get_imagenet_weights(), by_name=True)
elif init_with == "coco":

 # Load weights trained on MS COCO, but skip layers that
 # are different due to the different number of classes
 # See README for instructions to download the COCO weights

 model.load_weights(COCO_MODEL_PATH, by_name=True,
 exclude=["mrcnn_class_logits", "mrcnn_bbox_fc",
 "mrcnn_bbox", "mrcnn_mask"])
elif init_with == "last":

 # Load the last model you trained and continue training

 model.load_weights(model.find_last(), by_name=True)

#TRAINING
# Train the head branches
# Passing layers="heads" freezes all layers except the head
# layers. You can also pass a regular expression to select
# which layers to train by name pattern.

start_train = time.time()
model.train(dataset_train, dataset_val,
 learning_rate=config.LEARNING_RATE,
 epochs=4,
 layers='heads')
end_train = time.time()
minutes = round((end_train -start_train) / 60, 2)
print(f'Training took {minutes} minutes')
```

```
# Fine tune all layers
# Passing layers="all" trains all layers. You can also
# pass a regular expression to select which layers to
# train by name pattern.

start_train = time.time()
model.train(dataset_train, dataset_val,
 learning_rate=config.LEARNING_RATE / 10,
 epochs=8,
 layers="all")
end_train = time.time()
minutes = round((end_train -start_train) / 60, 2)
print(f'Training took {minutes} minutes')

#VIDEO INFERENCE

video_file = Path("../datasets/box_dataset_synthetic/videotest/boxvideo_24fps.mp4")
video_save_dir = Path("../datasets/box_dataset_synthetic/videotest/save")
video_save_dir.mkdir(exist_ok=True)

#ADJUST CONFIG PARAMETERS

class VideoInferenceConfig(CocoSynthConfig):
 GPU_COUNT = 1
 IMAGES_PER_GPU = 1
 IMAGE_MIN_DIM = 1088
 IMAGE_MAX_DIM = 1920
 IMAGE_SHAPE = [1920, 1080, 3]
 DETECTION_MIN_CONFIDENCE = 0.80
inference_config = VideoInferenceConfig()

#SETUP MODEL AND LOAD TRAINED WEIGHTS
# Recreate the model in inference mode

model = modellib.MaskRCNN(mode="inference",
 config=inference_config,
 model_dir=MODEL_DIR)

# Get path to saved weights
# Either set a specific path or find last trained weights
# model_path = str(Path(ROOT_DIR) / "logs" /

"box_synthetic20190328T2255/mask_rcnn_box_synthetic_0016.h5" )
model_path = model.find_last()

# Load trained weights (fill in path to trained weights here)

assert model_path != "", "Provide path to trained weights"
print("Loading weights from ", model_path)
model.load_weights(model_path, by_name=True)
```

```
import cv2
import skimage
import random
import colorsys
from tqdm import tqdm
def random_colors(N, bright=True):
 """ Generate random colors.
 To get visually distinct colors, generate them in HSV space then
 convert to RGB.
 Args:
 N: the number of colors to generate
 bright: whether or not to use bright colors
 Returns:
 a list of RGB colors, e.g [(0.0, 1.0, 0.0), (1.0, 0.0, 0.5), ...]
 """
 brightness = 1.0 if bright else 0.7
 hsv = [(i / N, 1, brightness) for i in range(N)]
 colors = list(map(lambda c: colorsys.hsv_to_rgb(*c), hsv))
 random.shuffle(colors)
 return colors
def apply_mask(image, mask, color, alpha=0.5):
 """ Apply the given mask to the image.
 Args:
 image: a cv2 image
 mask: a mask of which pixels to color
 color: the color to use
 alpha: how visible the mask should be (0 to 1)
 Returns:
 a cv2 image with mask applied
 """
 for c in range(3):
 image[:, :, c] = np.where(mask == 1,
 image[:, :, c] *
 (1 - alpha) + alpha * color[c] * 255,
 image[:, :, c])
 return image
def display_instances(image, boxes, masks, ids, names, scores, colors):
 """ Take the image and results and apply the mask, box, and label
 Args:
 image: a cv2 image
 boxes: a list of bounding boxes to display
 masks: a list of masks to display
 ids: a list of class ids
 names: a list of class names corresponding to the ids
 scores: a list of scores of each instance detected
 colors: a list of colors to use
 Returns:
 a cv2 image with instances displayed
 """
 n_instances = boxes.shape[0]
 if not n_instances:
 return image # no instances
 else:
 assert boxes.shape[0] == masks.shape[-1] == ids.shape[0]
 for i, color in enumerate(colors):

# Check if any boxes to show

 if not np.any(boxes[i ])
 continue

 # Check if any scores to show

 if scores is not None:
 score = scores[i]
 else:
 score = None

 # Add the mask

 image = apply_mask(image, masks[:, :, i], color)

 # Add the bounding box

 y1, x1, y2, x2 = boxes[i]
 image = cv2.rectangle(image, (x1, y1), (x2, y2), color, 2)

 # Add the label

 label = names[ids[i]]
 if score:
 label = f'{label} {score:.2f}'
 label_pos = (x1 + (x2 - x1) // 2, y1 + (y2 - y1) // 2) # center of bounding box
 image = cv2.putText(image, label, label_pos, cv2.FONT_HERSHEY_DUPLEX, 0.7,
color, 2)
 return image

#PREPARE FOR INFERENCE

video_file = Path("../datasets/box_dataset_synthetic/videotest/boxvideo_24fps.mp4")
video_save_dir = Path("../datasets/box_dataset_synthetic/videotest/save")
video_save_dir.mkdir(exist_ok=True)
vid_name = video_save_dir / "output.mp4"
v_format="FMP4"
fourcc = cv2.VideoWriter_fourcc(*v_format)
print('Writing output video to: ' + str(vid_name))

#colors = random_colors(30)

colors = [(1.0, 1.0, 0.0)] * 30
```

```
# Change color representation from RGB to BGR before displaying instances

colors = [(color[2], color[1], color[0]) for color in colors]

#PREPARE INFERENCE ON VIDEO

input_video = cv2.VideoCapture(str(video_file))
frame_count = int(input_video.get(cv2.CAP_PROP_FRAME_COUNT))
fps = int(input_video.get(cv2.CAP_PROP_FPS))
output_video = None
vid_size = None
current_frame = 0
for i in tqdm(range(frame_count)):

 # Read the current frame

 ret, frame = input_video.read()
 if not ret:
 break
 current_frame += 1

 # Change color representation from BGR to RGB before running model.detect()

 detect_frame = frame[:, :, ::-1]

 # Run inference on the color-adjusted frame

 results = model.detect([detect_frame], verbose=0)
 r = results[0]
 n_instances = r['rois'].shape[0]

# Make sure we have enough colors

 if len(colors) < n_instances:

 # not enough colors, generate more

 more_colors = random_colors(n_instances - len(colors))

 # Change color representation from RGB to BGR before displaying instances

 more_colors = [(color[2], color[1], color[0]) for color in more_colors]
 colors += more_colors

 # Display instances on the original frame

 display_frame = display_instances(frame, r['rois'], r['masks'], r['class_ids'],
 dataset_train.class_names, r['scores'], colors[0:n_instances])

 # Make sure we got displayed instances
 if display_frame is not None:
 frame = display_frame

 # Create the output_video if it doesn't yet exist

 if output_video is None:
 if vid_size is None:
 vid_size = frame.shape[1], frame.shape[0]
 output_video = cv2.VideoWriter(str(vid_name), fourcc, float(fps), vid_size, True)

 # Resize frame if necessary

 if vid_size[0] != frame.shape[1] and vid_size[1] != frame.shape[0]:
 frame = cv2.resize(frame, vid_size)

 # Write the frame to the output_video

 output_video.write(frame)
input_video.release()
output_video.release()
from pathlib import Path

# Set the ROOT_DIR variable to the root directory of the Mask_RCNN git repo

ROOT_DIR = 'C:/Users/HPPP/anaconda3/cocosynth-master/Mask_RCNN-master/'
assert os.path.exists(ROOT_DIR), 'ROOT_DIR does not exist. Did you forget to read the
instructions above? ;)'

# Directory to save logs and trained model

MODEL_DIR = os.path.join(ROOT_DIR, "logs")

# Local path to trained weights file

COCO_MODEL_PATH = os.path.join(ROOT_DIR, "mask_rcnn_coco.h5")

# Download COCO trained weights from Releases if needed

if not os.path.exists(COCO_MODEL_PATH):
 utils.download_trained_weights(COCO_MODEL_PATH)

#CONFIGURATION

class CocoSynthConfig(Config):
 """Configuration for training on the box_synthetic dataset.
 Derives from the base Config class and overrides specific values.
 """

# Give the configuration a recognizable name
```

```
 NAME = "cocosynth_dataset"

 # Train on 1 GPU and 1 image per GPU. Batch size is 52 (GPUs * images/GPU).

 GPU_COUNT = 1
 IMAGES_PER_GPU = 1

 # Number of classes (including background)

 NUM_CLASSES = 2 # background + 14 box types

 # All of our training images are 512x512

 IMAGE_MIN_DIM = 512
 IMAGE_MAX_DIM = 512

 # You can experiment with this number to see if it improves training

 STEPS_PER_EPOCH = 1000

 # This is how often validation is run. If you are using too much hard drive space
 # on saved models (in the MODEL_DIR), try making this value larger.

 VALIDATION_STEPS = 5

 # Matterport originally used resnet101, but I downsized to fit it on my graphics card
 BACKBONE = 'resnet101'
 # To be honest, I haven't taken the time to figure out what these do

 RPN_ANCHOR_SCALES = (8, 16, 32, 64, 128)
 TRAIN_ROIS_PER_IMAGE = 32
 MAX_GT_INSTANCES = 50
 POST_NMS_ROIS_INFERENCE = 500
 POST_NMS_ROIS_TRAINING = 1000
config = CocoSynthConfig()
config.display()

#DEFINE A DATASET

class CocoLikeDataset(utils.Dataset):
 """ Generates a COCO-like dataset, i.e. an image dataset annotated in the style of the
COCO dataset.
 See http://cocodataset.org/#home for more information.
 """
 def load_data(self, annotation_json, images_dir):
 """ Load the coco-like dataset from json
 Args:
 annotation_json: The path to the coco annotations json file
 images_dir: The directory holding the images referred to by the json file
 """
 # Load json from file

 json_file = open(annotation_json)
 coco_json = json.load(json_file)
 json_file.close()

 # Add the class names using the base method from utils.Dataset

 source_name = "coco_like"
 for category in coco_json['categories']:
 class_id = category['id']
 class_name = category['name']
 if class_id < 1:
 print('Error: Class id for "{}" cannot be less than one. (0 is reserved for the
background)'.format(class_name))
 return
 self.add_class(source_name, class_id, class_name)

 # Get all annotations

 annotations = {}
 for annotation in coco_json['annotations']:
 image_id = annotation['image_id']
 if image_id not in annotations:
 annotations[image_id] = []
 annotations[image_id].append(annotation)

 # Get all images and add them to the dataset

 seen_images = {}
 for image in coco_json['images']:
 image_id = image['id']
 if image_id in seen_images:
 print("Warning: Skipping duplicate image id: {}".format(image))
 else:
 seen_images[image_id] = image
 try:
 image_file_name = image['file_name']
 image_width = image['width']
 image_height = image['height']
 except KeyError as key:
 print("Warning: Skipping image (id: {}) with missing key: {}".format(image_id,
key))
  image_path = os.path.abspath(os.path.join(images_dir, image_file_name))
 image_annotations = annotations[image_id]

# Add the image using the base method from utils.Dataset

 self.add_image(
 source=source_name,
 image_id=image_id,
 path=image_path,
 width=image_width,
 height=image_height,
 annotations=image_annotations
 )

 def load_mask(self, image_id):
 """ Load instance masks for the given image.
 MaskRCNN expects masks in the form of a bitmap [height, width, instances].
 Args:
 image_id: The id of the image to load masks for
 Returns:
 masks: A bool array of shape [height, width, instance count] with
 one mask per instance.
 class_ids: a 1D array of class IDs of the instance masks.
 """
 image_info = self.image_info[image_id]
 annotations = image_info['annotations']
 instance_masks = []
 class_ids = []
 for annotation in annotations:
 class_id = annotation['category_id']
 mask = Image.new('1', (image_info['width'], image_info['height']))
 mask_draw = ImageDraw.ImageDraw(mask, '1')
 for segmentation in annotation['segmentation']:
 mask_draw.polygon(segmentation, fill=1)
 bool_array = np.array(mask) > 0
 instance_masks.append(bool_array)
 class_ids.append(class_id)
 mask = np.dstack(instance_masks)
 class_ids = np.array(class_ids, dtype=np.int32)
  return mask, class_ids

#CREATE THE TRAINING AND VALIDATION DATASETS

dataset_train = CocoLikeDataset()
dataset_train.load_data('C:/Users/HPPP/anaconda3/cocosynthmaster/datasets/weeds/
output/training/coco_instances.json',
 'C:/Users/HPPP/anaconda3/cocosynth-master/datasets/weeds/output/training/images')
dataset_train.prepare()
dataset_val = CocoLikeDataset()
dataset_val.load_data('C:/Users/HPPP/anaconda3/cocosynthmaster/datasets/weeds/out
put/val/coco_instances.json',
 'C:/Users/HPPP/anaconda3/cocosynth-master/datasets/weeds/output/val/images')
dataset_val.prepare()

#DISPLAY FEW IMAGES FROM TRAIN AND VAL DATASETS

for name, dataset in [('training', dataset_train), ('validation', dataset_val)]:
 print(f'Displaying examples from {name} dataset:')
  image_ids = np.random.choice(dataset.image_ids, 3)
 for image_id in image_ids:
 image = dataset.load_image(image_id)
 mask, class_ids = dataset.load_mask(image_id)
 visualize.display_top_masks(image, mask, class_ids, dataset.class_names)

#CREATE THE TRAINING MODEL AND TRAIN
# Create model in training mode

model = modellib.MaskRCNN(mode="training", config=config,
 model_dir=MODEL_DIR)

# Which weights to start with?

init_with = "coco" # imagenet, coco, or last
if init_with == "imagenet":
 model.load_weights(model.get_imagenet_weights(), by_name=True)
elif init_with == "coco":

 # Load weights trained on MS COCO, but skip layers that
 # are different due to the different number of classes
 # See README for instructions to download the COCO weights

 model.load_weights(COCO_MODEL_PATH, by_name=True,
 exclude=["mrcnn_class_logits", "mrcnn_bbox_fc",
 "mrcnn_bbox", "mrcnn_mask"])
elif init_with == "last":

 # Load the last model you trained and continue training

 model.load_weights(model.find_last(), by_name=True)

#TRAINING
# Train the head branches
# Passing layers="heads" freezes all layers except the head
# layers. You can also pass a regular expression to select
# which layers to train by name pattern.

start_train = time.time()
model.train(dataset_train, dataset_val,
 learning_rate=config.LEARNING_RATE,
 epochs=4,
 layers='heads')
end_train = time.time()
minutes = round((end_train -start_train) / 60, 2)
print(f'Training took {minutes} minutes')
```

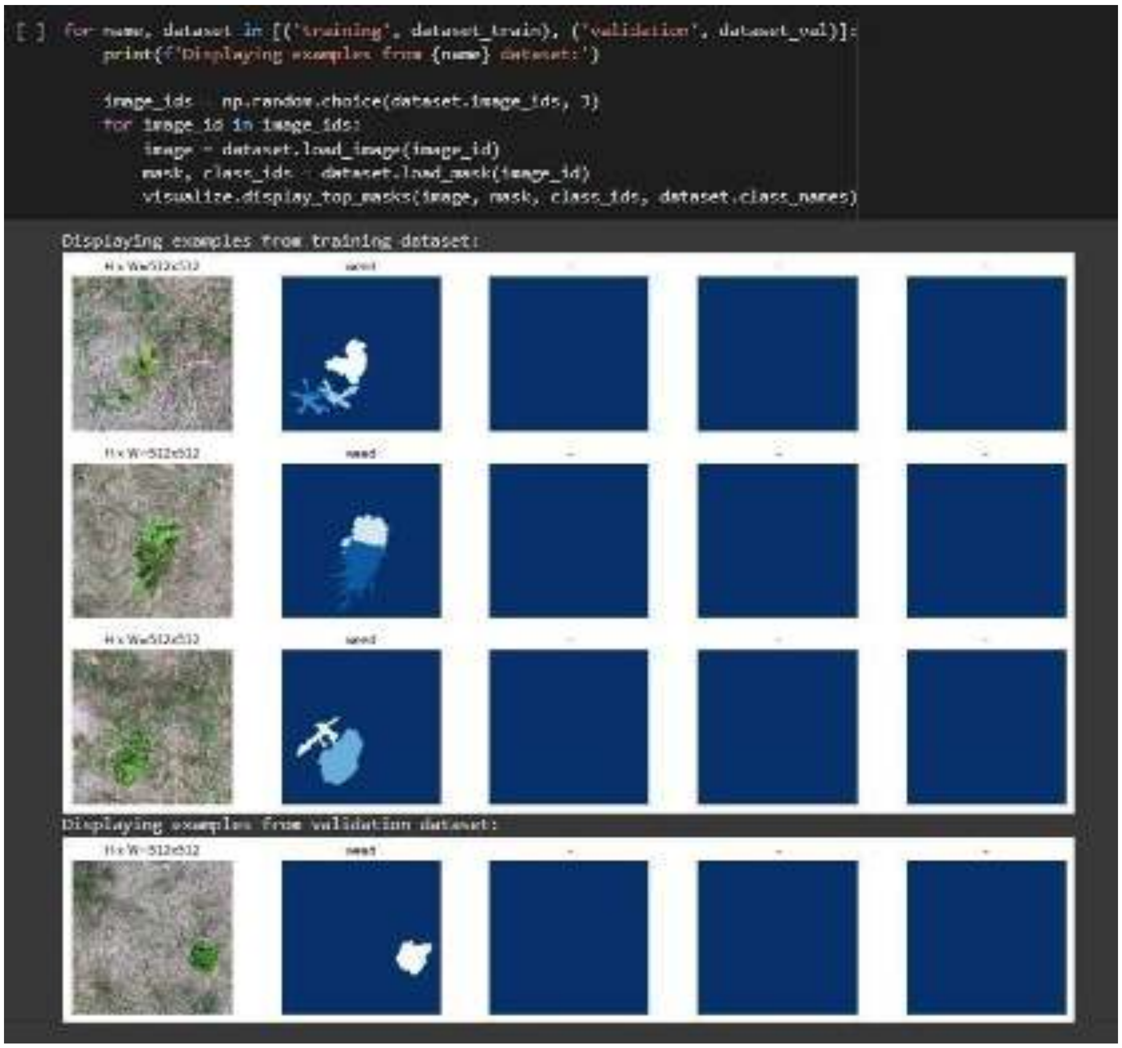
```
# Fine tune all layers
# Passing layers="all" trains all layers. You can also
# pass a regular expression to select which layers to
# train by name pattern.

start_train = time.time()
model.train(dataset_train, dataset_val,
 learning_rate=config.LEARNING_RATE / 10,
 epochs=8,
 layers="all")
end_train = time.time()
minutes = round((end_train -start_train) / 60, 2)
print(f'Training took {minutes} minutes')

#VIDEO INFERENCE

video_file = Path("../datasets/box_dataset_synthetic/videotest/boxvideo_24fps.mp4")
video_save_dir = Path("../datasets/box_dataset_synthetic/videotest/save")
video_save_dir.mkdir(exist_ok=True)

#ADJUST CONFIG PARAMETERS

class VideoInferenceConfig(CocoSynthConfig):
 GPU_COUNT = 1
 IMAGES_PER_GPU = 1
 IMAGE_MIN_DIM = 1088
 IMAGE_MAX_DIM = 1920
 IMAGE_SHAPE = [1920, 1080, 3]
 DETECTION_MIN_CONFIDENCE = 0.80
inference_config = VideoInferenceConfig()

#SETUP MODEL AND LOAD TRAINED WEIGHTS
# Recreate the model in inference mode

model = modellib.MaskRCNN(mode="inference",
 config=inference_config,
 model_dir=MODEL_DIR)

# Get path to saved weights
# Either set a specific path or find last trained weights
# model_path = str(Path(ROOT_DIR) / "logs" /

"box_synthetic20190328T2255/mask_rcnn_box_synthetic_0016.h5" )
model_path = model.find_last()

# Load trained weights (fill in path to trained weights here)

assert model_path != "", "Provide path to trained weights"
print("Loading weights from ", model_path)
model.load_weights(model_path, by_name=True)
```

```
import cv2
import skimage
import random
import colorsys
from tqdm import tqdm
def random_colors(N, bright=True):
 """ Generate random colors.
 To get visually distinct colors, generate them in HSV space then
 convert to RGB.
 Args:
 N: the number of colors to generate
 bright: whether or not to use bright colors
 Returns:
 a list of RGB colors, e.g [(0.0, 1.0, 0.0), (1.0, 0.0, 0.5), ...]
 """
 brightness = 1.0 if bright else 0.7
 hsv = [(i / N, 1, brightness) for i in range(N)]
 colors = list(map(lambda c: colorsys.hsv_to_rgb(*c), hsv))
 random.shuffle(colors)
 return colors
def apply_mask(image, mask, color, alpha=0.5):
 """ Apply the given mask to the image.
 Args:
 image: a cv2 image
 mask: a mask of which pixels to color
 color: the color to use
 alpha: how visible the mask should be (0 to 1)
 Returns:
 a cv2 image with mask applied
 """
 for c in range(3):
 image[:, :, c] = np.where(mask == 1,
 image[:, :, c] *
 (1 - alpha) + alpha * color[c] * 255,
 image[:, :, c])
 return image
def display_instances(image, boxes, masks, ids, names, scores, colors):
 """ Take the image and results and apply the mask, box, and label
 Args:
 image: a cv2 image
 boxes: a list of bounding boxes to display
 masks: a list of masks to display
 ids: a list of class ids
 names: a list of class names corresponding to the ids
 scores: a list of scores of each instance detected
 colors: a list of colors to use
 Returns:
 a cv2 image with instances displayed
 """
 n_instances = boxes.shape[0]
 if not n_instances:
 return image # no instances
 else:
 assert boxes.shape[0] == masks.shape[-1] == ids.shape[0]
 for i, color in enumerate(colors):

 # Check if any boxes to show

 if not np.any(boxes[i ])
 continue

 # Check if any scores to show

 if scores is not None:
 score = scores[i]
 else:
 score = None

 # Add the mask

 image = apply_mask(image, masks[:, :, i], color)

 # Add the bounding box

 y1, x1, y2, x2 = boxes[i]
 image = cv2.rectangle(image, (x1, y1), (x2, y2), color, 2)

 # Add the label

 label = names[ids[i]]
 if score:
 label = f'{label} {score:.2f}'
 label_pos = (x1 + (x2 - x1) // 2, y1 + (y2 - y1) // 2) # center of bounding box
 image = cv2.putText(image, label, label_pos, cv2.FONT_HERSHEY_DUPLEX, 0.7,
color, 2)
 return image

#PREPARE FOR INFERENCE

video_file = Path("../datasets/box_dataset_synthetic/videotest/boxvideo_24fps.mp4")
video_save_dir = Path("../datasets/box_dataset_synthetic/videotest/save")
video_save_dir.mkdir(exist_ok=True)
vid_name = video_save_dir / "output.mp4"
v_format="FMP4"
fourcc = cv2.VideoWriter_fourcc(*v_format)
print('Writing output video to: ' + str(vid_name))

#colors = random_colors(30)

colors = [(1.0, 1.0, 0.0)] * 30
```

```
# Change color representation from RGB to BGR before displaying instances

colors = [(color[2], color[1], color[0]) for color in colors]

#PREPARE INFERENCE ON VIDEO

input_video = cv2.VideoCapture(str(video_file))
frame_count = int(input_video.get(cv2.CAP_PROP_FRAME_COUNT))
fps = int(input_video.get(cv2.CAP_PROP_FPS))
output_video = None
vid_size = None
current_frame = 0
for i in tqdm(range(frame_count)):

 # Read the current frame

 ret, frame = input_video.read()
 if not ret:
 break
 current_frame += 1

 # Change color representation from BGR to RGB before running model.detect()

 detect_frame = frame[:, :, ::-1]

 # Run inference on the color-adjusted frame

 results = model.detect([detect_frame], verbose=0)
 r = results[0]
 n_instances = r['rois'].shape[0]

 # Make sure we have enough colors

 if len(colors) < n_instances:

 # not enough colors, generate more

 more_colors = random_colors(n_instances - len(colors))

 # Change color representation from RGB to BGR before displaying instances

 more_colors = [(color[2], color[1], color[0]) for color in more_colors]
 colors += more_colors

 # Display instances on the original frame
 display_frame = display_instances(frame, r['rois'], r['masks'], r['class_ids'],
 dataset_train.class_names, r['scores'], colors[0:n_instances])
 # Make sure we got displayed instances

 if display_frame is not None:
 frame = display_frame

 # Create the output_video if it doesn't yet exist

 if output_video is None:
 if vid_size is None:
 vid_size = frame.shape[1], frame.shape[0]
 output_video = cv2.VideoWriter(str(vid_name), fourcc, float(fps), vid_size, True)

 # Resize frame if necessary

 if vid_size[0] != frame.shape[1] and vid_size[1] != frame.shape[0]:
 frame = cv2.resize(frame, vid_size)

 # Write the frame to the output_video

 output_video.write(frame)
input_video.release()
output_video.release()
```

5.RESULTS

5.1 SCREENSHOTS



```
for name, dataset in [('training', dataset_train), ('validation', dataset_val)]:
    print(f'Displaying examples from {name} dataset:')

    image_ids = np.random.choice(dataset.image_ids, 3)
    for image_id in image_ids:
        image = dataset.load_image(image_id)
        mask, class_ids = dataset.load_mask(image_id)
        visualize.display_top_masks(image, mask, class_ids, dataset.class_names)
```

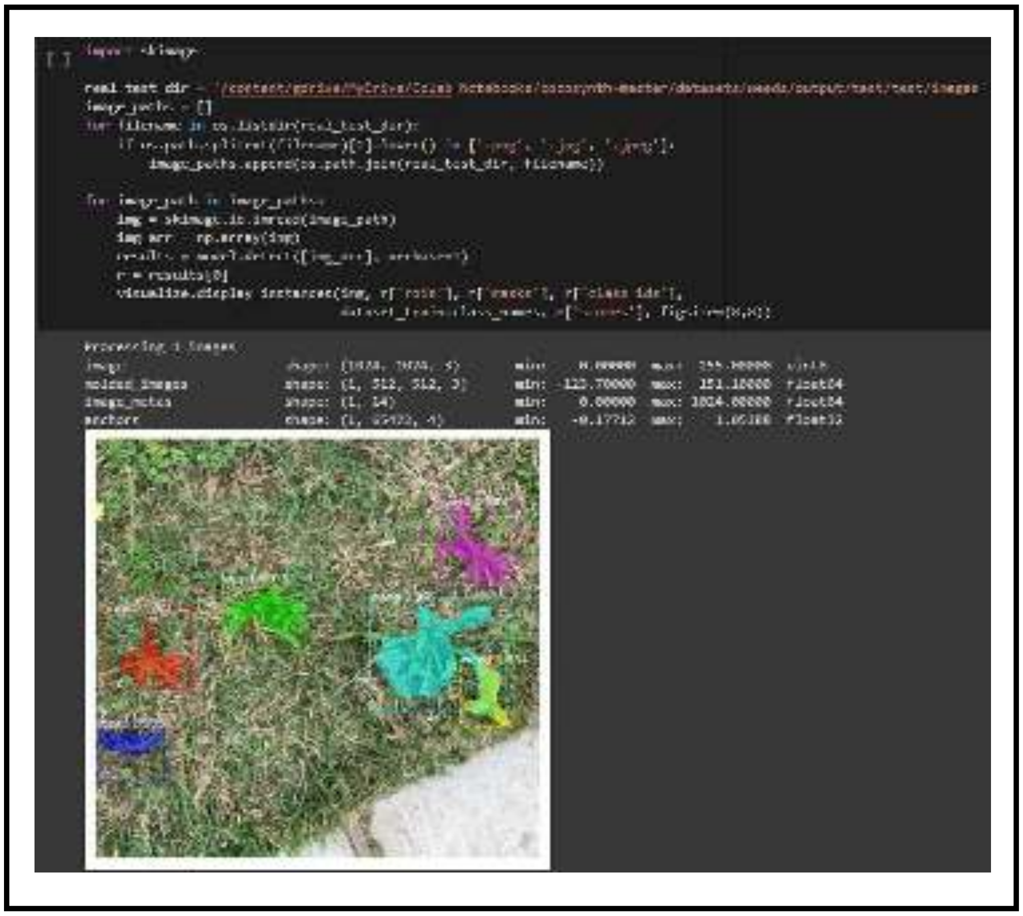
```
Displaying examples from training dataset:
```

```
Displaying examples from validation dataset:
```

Screenshot 5.1.1 : Displaying few images from training and validation dataset

Here we are loading an image to see how our model performs. We can load any of our images to test the model.

We will use the Mask R-CNN model along with the pretrained weights and see how well it segments the objects in the image. We will first take the predictions from the model and then plot the results to visualize them.



```
import skimage

real_test_dir = '/content/gdrive/MyDrive/Colab Notebooks/cocosynth-master/datasets/weeds/output/test/test/images
image_paths = []
for filename in os.listdir(real_test_dir):
    if os.path.splitext(filename)[1].lower() in ['.png', '.jpg', '.jpeg']:
        image_paths.append(os.path.join(real_test_dir, filename))

for image_path in image_paths:
    img = skimage.io.imread(image_path)
    img_arr = np.array(img)
    results = model.detect([img_arr], verbose=1)
    r = results[0]
    visualize.display_instances(img, r['rois'], r['masks'], r['class_ids'],
                                dataset_train.class_names, r['scores'], figsize=(8,8))
```

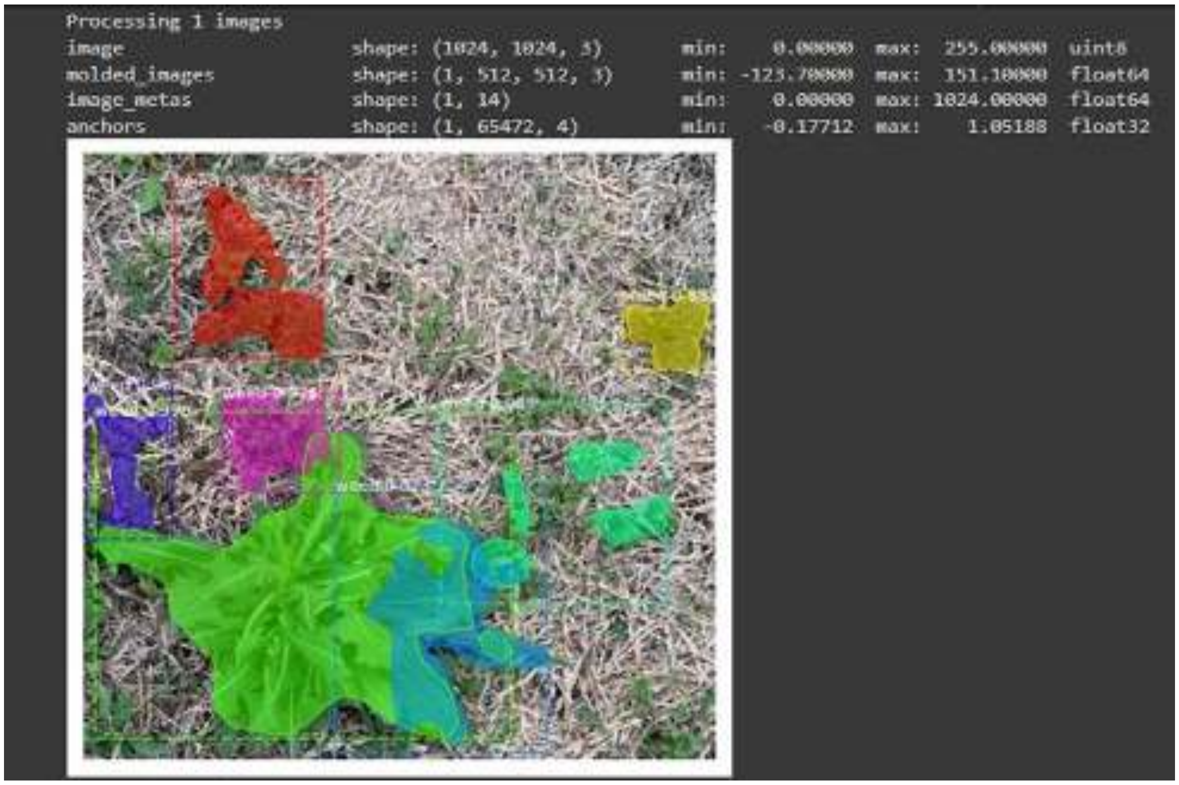
```
Processing 1 images
image                    shape: (1024, 1024, 3)       min:    0.00000  max:  255.00000  uint8
molded_images            shape: (1, 512, 512, 3)      min: -123.70000  max:  151.10000  float64
image_metas              shape: (1, 14)               min:    0.00000  max: 1024.00000  float64
anchors                  shape: (1, 65472, 4)         min:   -0.17712  max:    1.05188  float32
```

Screenshot 5.1.2 : Image inference on real image

We can see that the model has done pretty well to segment the weeds in the image. We can look at each mask or the segmented objects separately as well.

Here we are loading an image to see how our model performs. We can load any of our images to test the model.

We will use the Mask R-CNN model along with the pretrained weights and see how well it segments the objects in the image. We will first take the predictions from the model and then plot the results to visualize them.

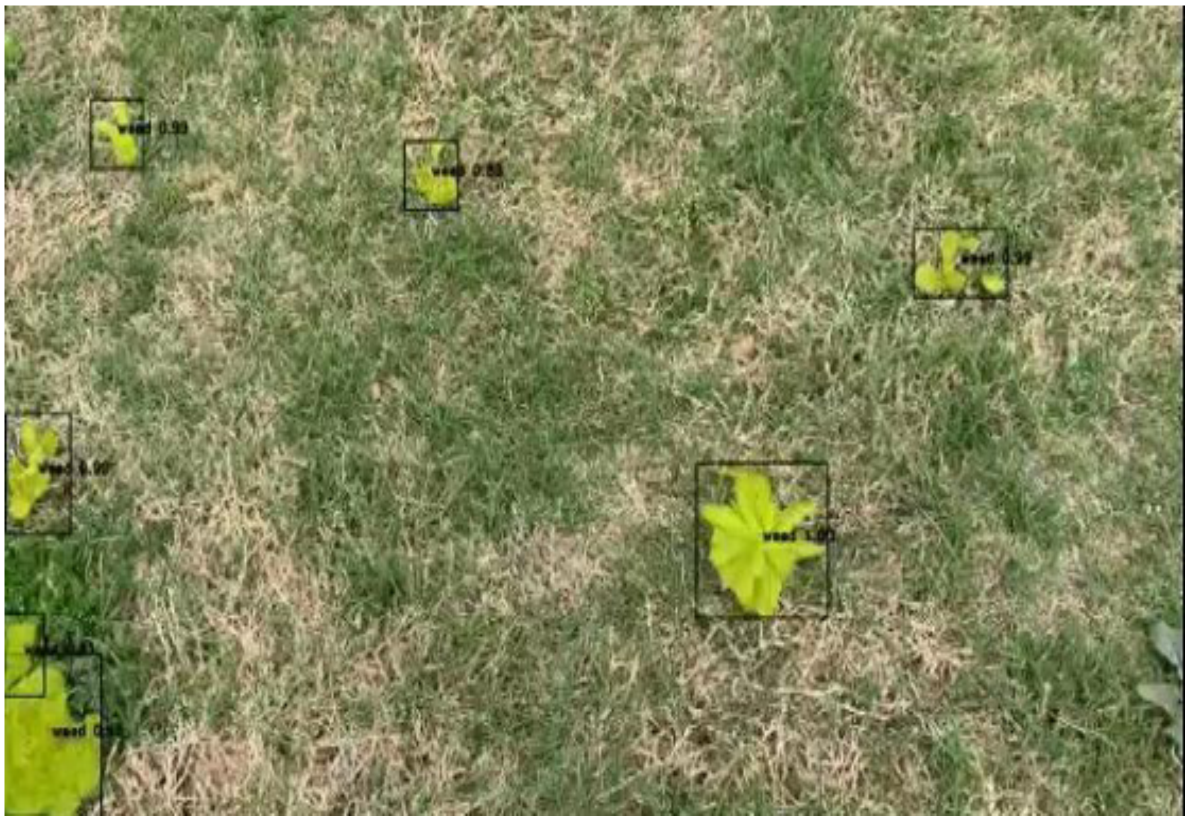


```
Processing 1 images
image                    shape: (1024, 1024, 3)      min:    0.00000  max:  255.00000  uint8
molded_images            shape: (1, 512, 512, 3)     min: -123.70000  max:  151.10000  float64
image_metas              shape: (1, 14)              min:    0.00000  max: 1024.00000  float64
anchors                  shape: (1, 65472, 4)        min:   -0.17712  max:    1.05188  float32
```

Screenshot 5.1.3 : Image inference on real image

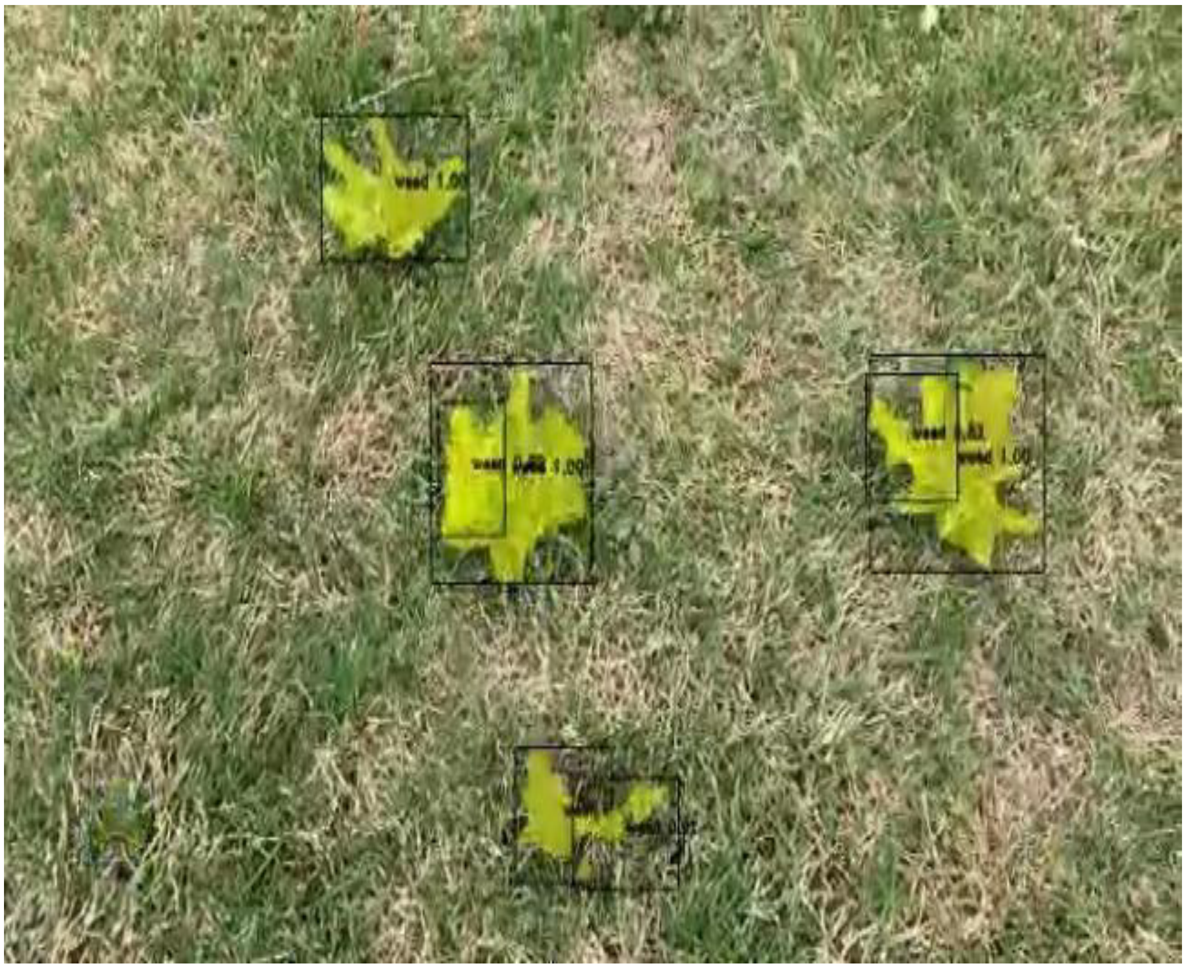
We can see that the model has done pretty well to segment the weeds in the image. We can look at each mask or the segmented objects separately as well.

This is the result we got using our Mask-RCNN model. We can see that it predicted the weeds pretty well.



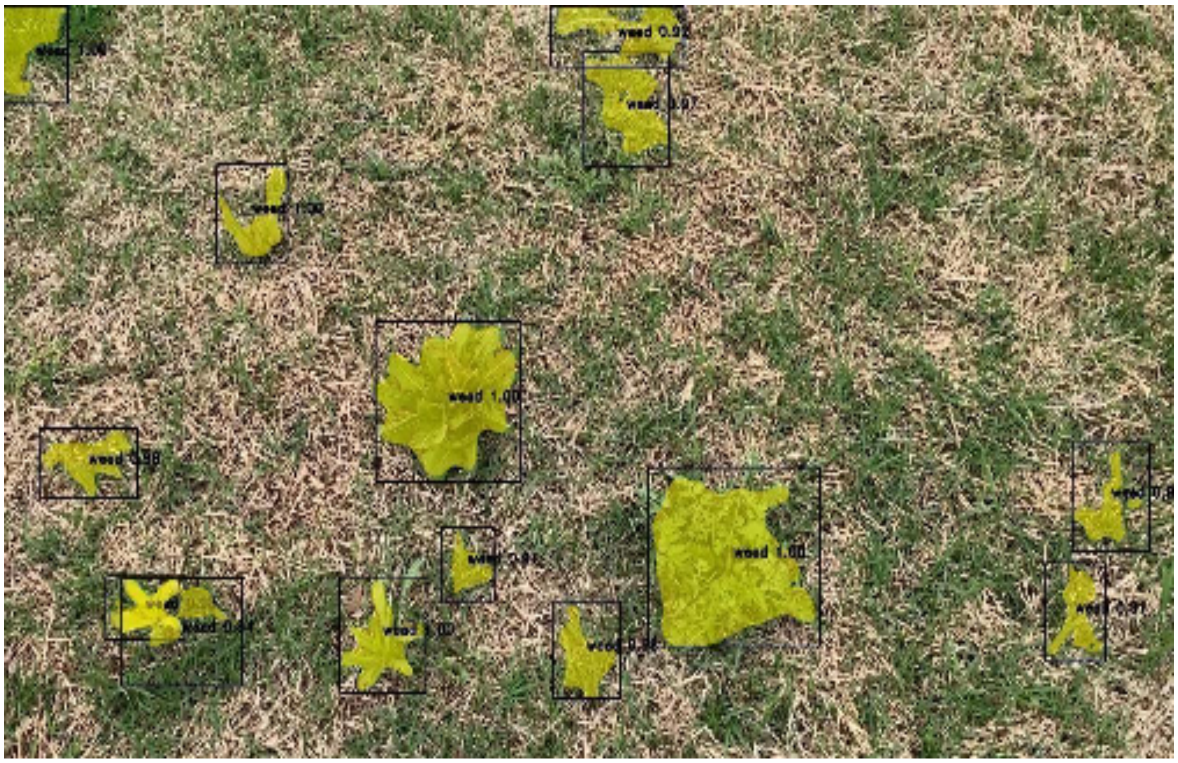
Screenshot 5.1.4 : Screenshot of video output

This is the result we got using our Mask-RCNN model. We can see that it predicted the weeds pretty well.



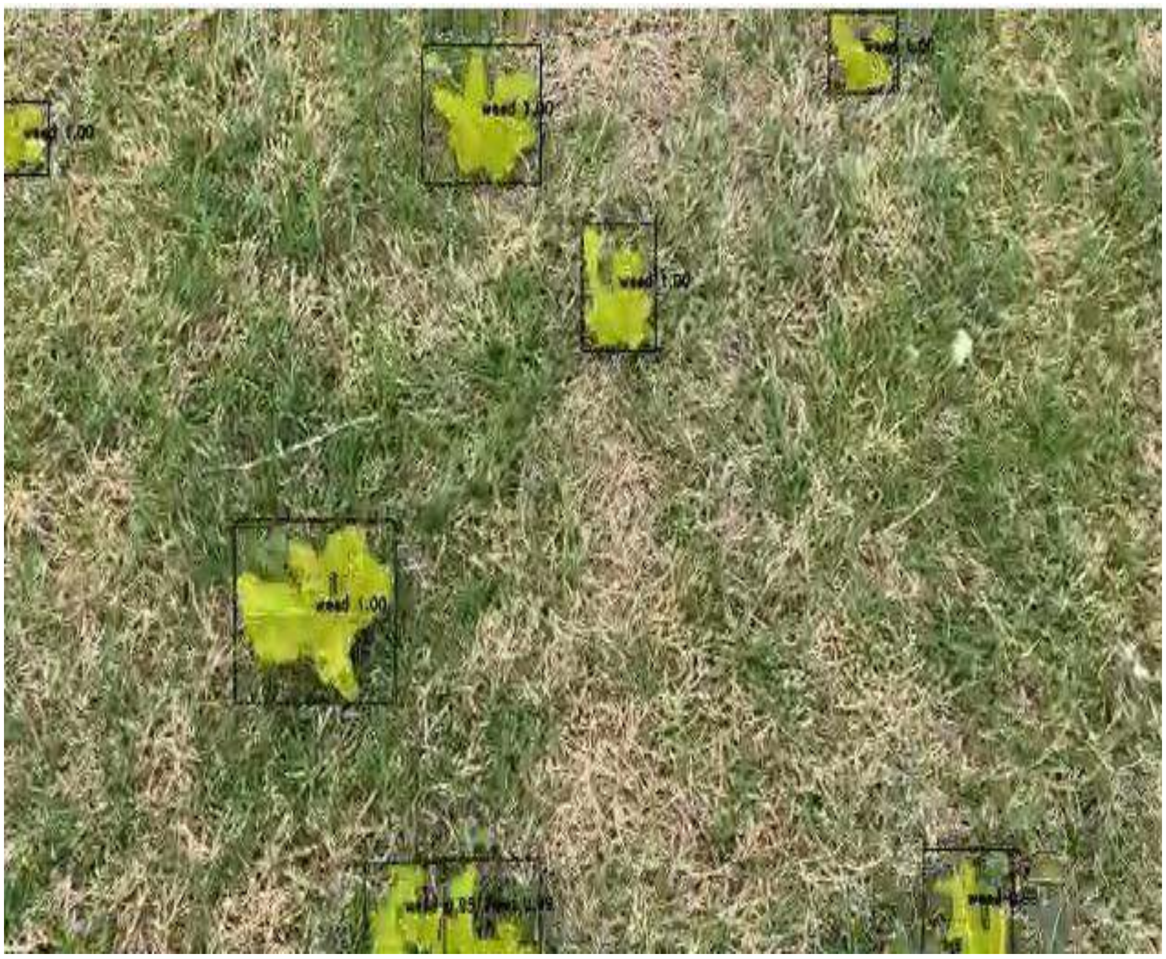
Screenshot 5.1.5 : Screenshot of video output

This is the result we got using our Mask-RCNN model. We can see that it predicted the weeds pretty well.



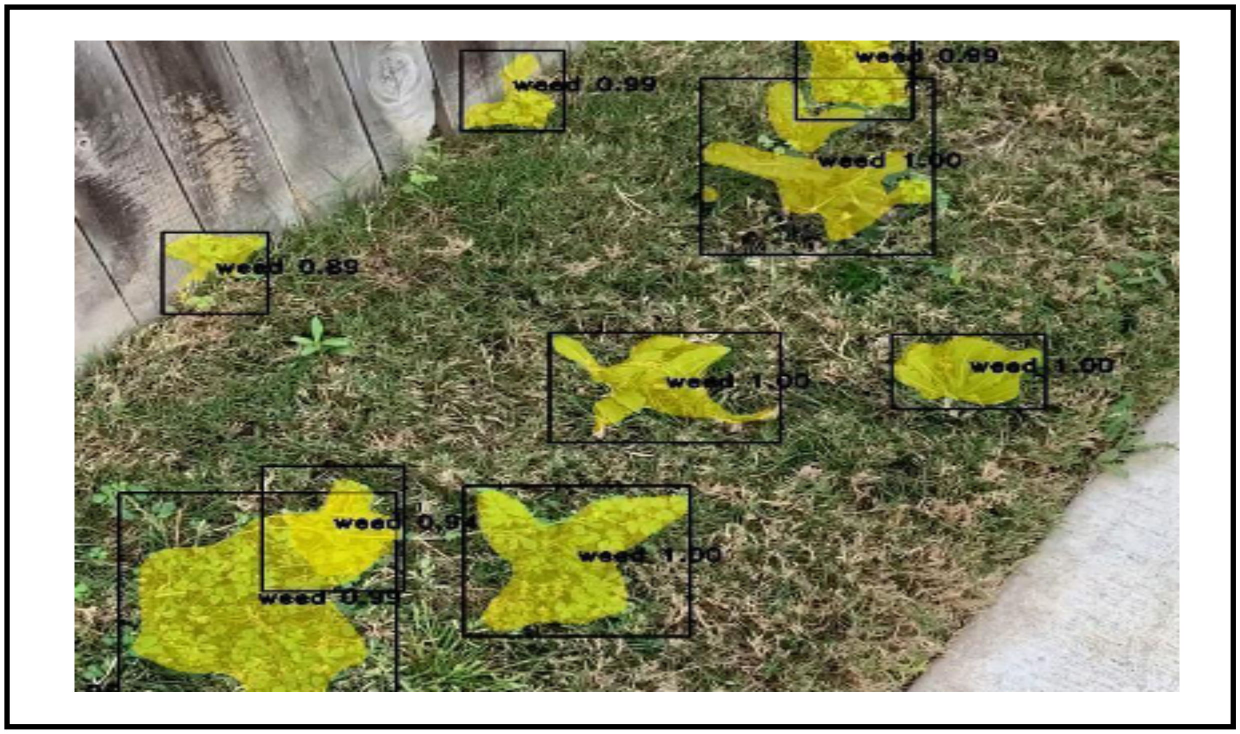
Screenshot 5.1.6 : Screenshot of video output

This is the result we got using our Mask-RCNN model. We can see that it predicted the weeds pretty well.



Screenshot 5.1.7 : Screenshot of video output

This is the result we got using our Mask-RCNN model. We can see that it predicted the weeds pretty well.



Screenshot 5.1.8 : Screenshot of video output

5.2 RESULT ANALYSIS:

The proposed work has been performed on a dataset of around 1000 training images and up to 300 validation images. The purpose of the work is to detect the weeds using mask R-CNN.

Mask R-CNN (Regional Convolutional Neural Network) has been the state-of-the-art model for object instance segmentation since it was proposed. Mask R-CNN utilizes a relatively simple method to achieve its success in tasks of object detection, instance segmentation, and key point detection. We had tested with several backbones and different instance segmentation methods.

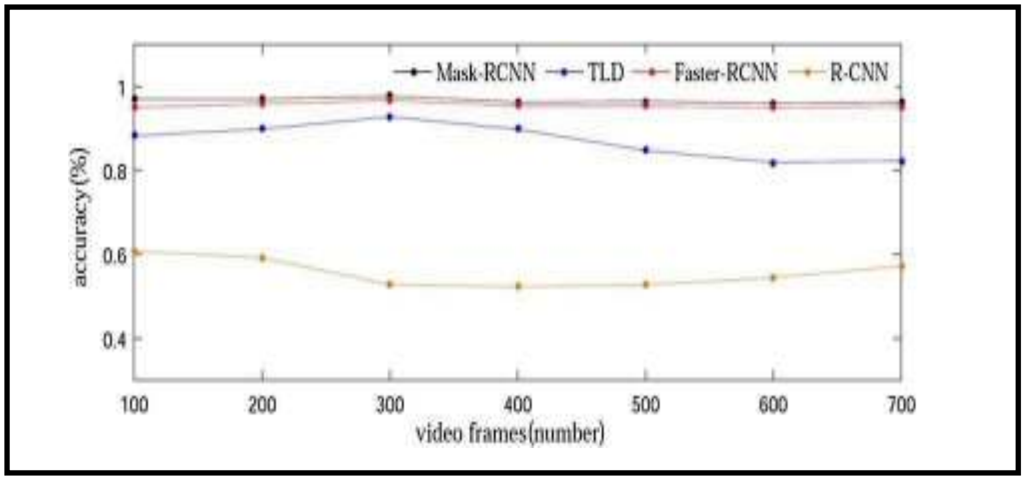
The key element of Mask R-CNN is the pixel-to-pixel alignment, which is the main missing piece of Fast/Faster R-CNN. Mask R-CNN adopts the same two-stage procedure with an identical first stage (which is RPN). In the second stage, in parallel to predicting the class and box offset, Mask R-CNN also outputs a binary mask for each RoI. This is in contrast to most recent systems, where classification depends on mask predictions.

Furthermore, Mask R-CNN is simple to implement and train given the Faster R-CNN framework, which facilitates a wide range of flexible architecture designs. Additionally, the mask branch only adds a small computational overhead, enabling a fast system and rapid experimentation.

Method	Backbone	AP	AP@0.5	AP@0.75
MNC(7)	ResNet-101	24.6	44.3	24.8
FCIS(23)	ResNet-101	29.2	49.5	-
FCIS+++(23)	ResNet-101	33.6	54.5	-
Mask R-CNN [15]	ResNet-101	33.1	54.9	34.8
Mask R-CNN [15]	ResNet-101 FPN	35.7	58.0	37.8
MaskLab [3]	ResNet-101	35.4	57.4	37.4
Mask R-CNN	ResNet-101	34.3	55.0	36.6
Mask R-CNN	ResNet-101 FPN	37.0	59.2	39.5

Table 5.2.1 : Comparing different instance segmentation methods

The results show that no matter what backbone network is used, Mask R-CNN can always outperform. We can also see that using ResNet-101 FPN can give much better results.



Graph 5.2.2 : Graph showing accuracy of Mask R-CNN

Accuracy comparison of Mask R-CNN, TLD, Faster-RCNN, RCNN algorithms for maintenance personnel. In the figure, the X-axis represents the number of images measured, and the y-axis represents the accuracy of the algorithm recognition. We can see that Mask R-CNN outperforms all the other algorithms with outmostaccuracy.

6.TESTING

6 TESTING

6.1 INTRODUCTION TO TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

6.2 TYPES OF TESTING

6.2.1 UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

6.2.2 INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

6.2.3 FUNCTIONAL TESTING

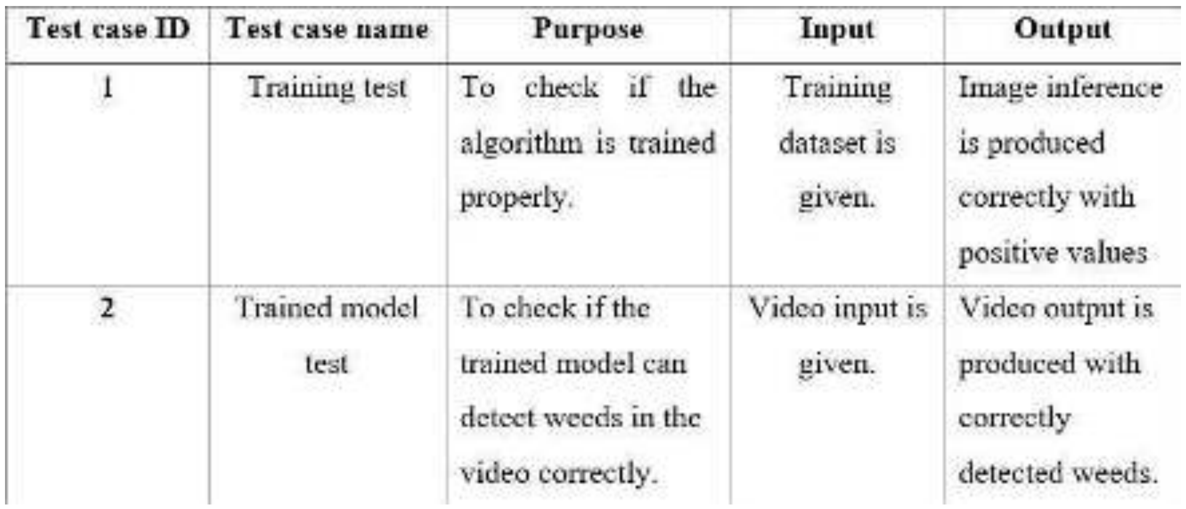
Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input	: Identified classes of valid input must be accepted.
Invalid Input	: Identified classes of invalid input must be rejected.
Functions	: Identified functions must be exercised.
Output	: Identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked. Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows, data fields, predefined processes.

6.3 TEST CASES



Test case ID	Test case name	Purpose	Input	Output
1	Training test	To check if the algorithm is trained properly.	Training dataset is given.	Image inference is produced correctly with positive values
2	Trained model test	To check if the trained model can detect weeds in the video correctly.	Video input is given.	Video output is produced with correctly detected weeds.

7.CONCLUSION

7.1 CONCLUSION

As the weeds are mostly quick growers and compete with the crops for light, water, nutrients, and space, it is very important remove weeds from the crops. But manually removing them is tedious and takes a lot of time. Spraying herbicides can cause pollution. Hence, a deep learning model is developed using convolution neural network to detect weeds with a good accuracy so that the model could be used to detect the weeds in the field in a shorter time. Our proposed work uses Mask R-CNN built on ResNet 101 and FPN thereby reducing the complexity in training as compared to other backbones and instance segmentation methods. Further work can be enhanced using larger datasets for improved results.

7.2 FUTURE SCOPE

We can improve both the training and also the detection time of weeds. We can further enhance it by attaching it to robots or tractors to pluck the weeds from the crops thereby, saving the time and reducing the efforts of the farmer. We can also develop the model further in such a way that it can tell the percentage of weeds and also the future risks.

8.BIBILOGRAPHY

8.BIBILOGRAPHY

8.1 REFERENCES

1. Noxious Weeds Management In: ARTICLE 1.7. California Legislature. 2018.
2. Lanini W, Strange M. Low-input management of weeds in vegetable fields. Calif Agric. 1991;45(1):11–3.
3. Hodgson JM. The nature, ecology, and control of Canada thistle. vol 1386. Agricultural Research service, US Dept. of Agriculture.
4. Monaco T, Grayson A, Sanders D. Influence of four weed species on the growth, yield, and quality of direct-seeded tomatoes (Lycopersicon esculentum). Weed Sci. 1981;29(4):394–7.
5. Nave W, Wax L. Effect of weeds on soybean yield and harvesting efficiency. Weed Sci. 1971;19(5):533–5.
6. Smith DT, Baker RV, Steele GL. Palmer amaranth (Amaranthus palmeri) impacts on yield, harvesting, and ginning in dryland cotton (Gossypium hirsutum). Weed Technol. 2000;14(1):122–6.
7. Weis M, Gerhards R. Detection of weeds using image processing and clustering. Bornimer Agrartechnische Berichte. 2008;69(138):e144.
8. Desai R, Desai K, Desai S, Solanki Z, Patel D, Patel V. Removal of weeds using image processing: a technical review. Int J Adv Comput Technol. 2015;4:27–31.
9. Weis M. An image analysis and classification system for automatic weed species identification in different crops for precision weed management. 2010.
10. Choudhary J, Nayak S. A survey on weed detection using image processing in agriculture. Int J Comput Sci Eng. 2016;4(6).
11. Mustafa MM, Hussain A, Ghazali KH, Riyadi S, editors. Implementation ofimage processing technique in real time vision system for automatic weeding strategy. 2007 IEEE International Symposium on Signal Processing and Information Technology; 2007: IEEE.
12. Robovator. VisionWeeding.
13. Herrera P, Dorado J, Ribeiro Á. A novel approach for weed type classification based on shape descriptors and a fuzzy decision-making method. Sensors. 2014;14(8):15304–24.
14. Aravind R, Daman M, Kariyappa B, editors. Design and development of automatic weed detection and smart herbicide sprayer robot. 2015 IEEE Recent Advances in Intelligent Computational Systems (RAICS); 2015: IEEE.

8.2 WEBSITES

leginfo.legislature.ca.gov/faces/codes_displayText.xhtml?lawCode=FAC&division=4 http://www.visionweeding.com/robovator/.

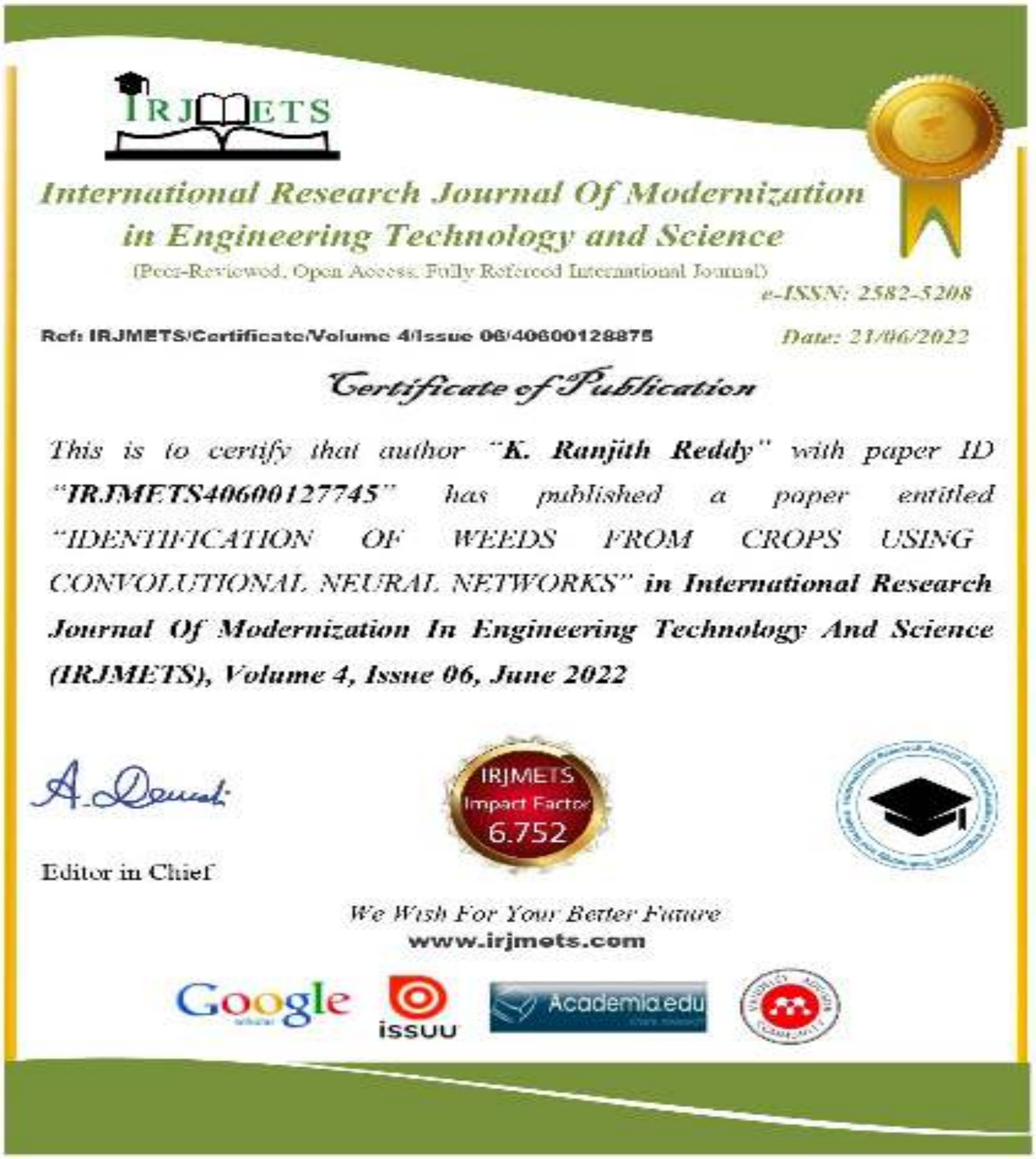
http://www.visionweeding.com/robovator/

8.3 GITHUB LINK

https://github.com/sunilnanigit/weed-detection-

9.PAPER PUBLICATION

10.CERTIFICATES



IRJMETS

International Research Journal Of Modernization in Engineering Technology and Science

(Peer-Reviewed, Open Access, Fully Refereed International Journal)

e-ISSN: 2582-5208

Ref: IRJMETS/Certificate/Volume 4/Issue 06/40600128875 *Date: 21/06/2022*

Certificate of Publication

This is to certify that author ***"K. Ranjith Reddy"*** *with paper ID* ***"IRJMETS40600127745"*** *has published a paper entitled "IDENTIFICATION OF WEEDS FROM CROPS USING CONVOLUTIONAL NEURAL NETWORKS"* ***in International Research Journal Of Modernization In Engineering Technology And Science (IRJMETS), Volume 4, Issue 06, June 2022***

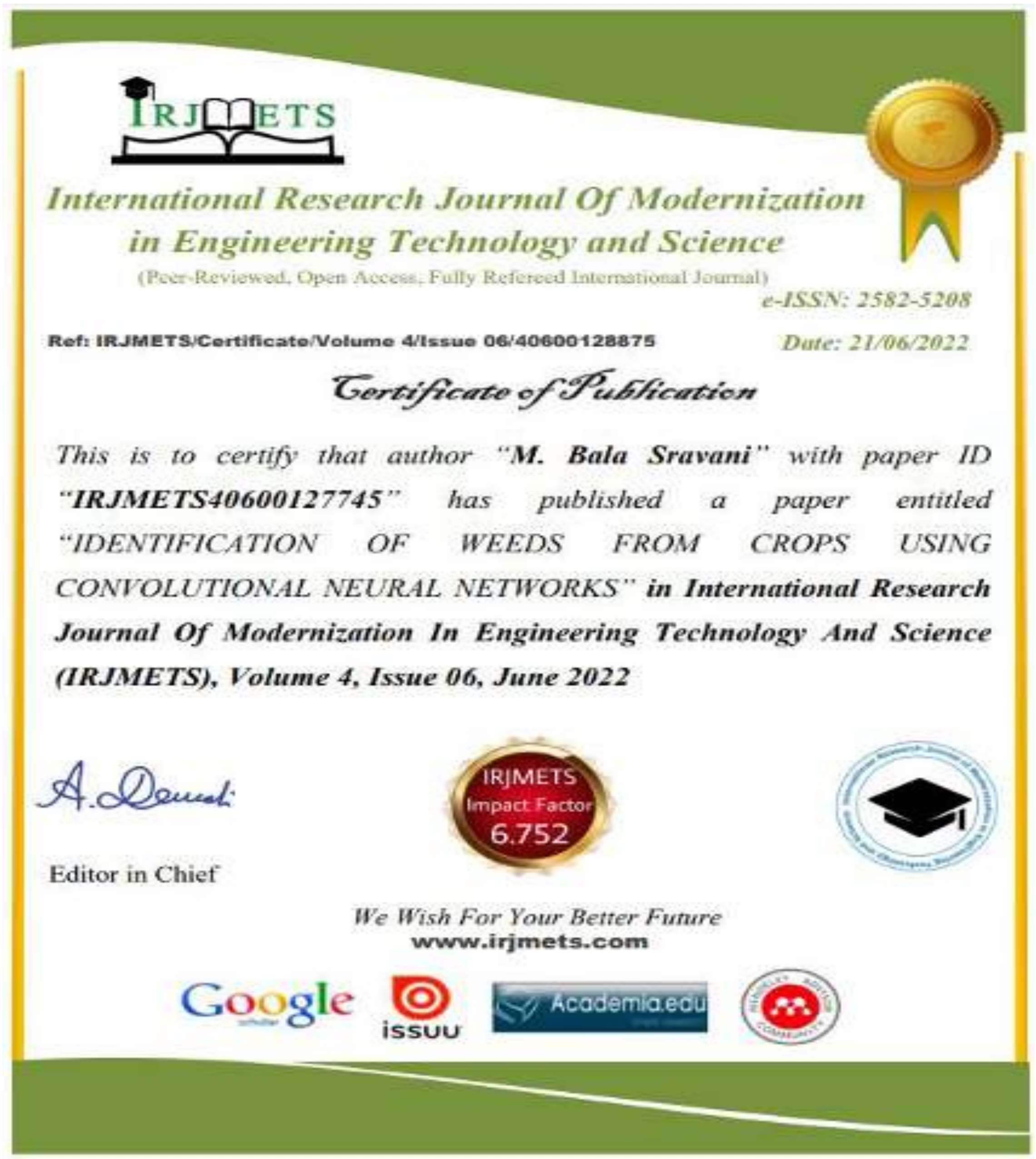
IRJMETS Impact Factor 6.752

[illegible]

Editor in Chief

We Wish For Your Better Future

www.irjmets.com



IRJMETS

International Research Journal Of Modernization in Engineering Technology and Science

(Peer-Reviewed, Open Access, Fully Refereed International Journal)

e-ISSN: 2582-5208

Ref: IRJMETS/Certificate/Volume 4/Issue 06/40600128875

Date: 21/06/2022

Certificate of Publication

This is to certify that author "**M. Bala Sravani**" with paper ID "**IRJMETS40600127745**" has published a paper entitled "IDENTIFICATION OF WEEDS FROM CROPS USING CONVOLUTIONAL NEURAL NETWORKS" **in International Research Journal Of Modernization In Engineering Technology And Science (IRJMETS), Volume 4, Issue 06, June 2022**

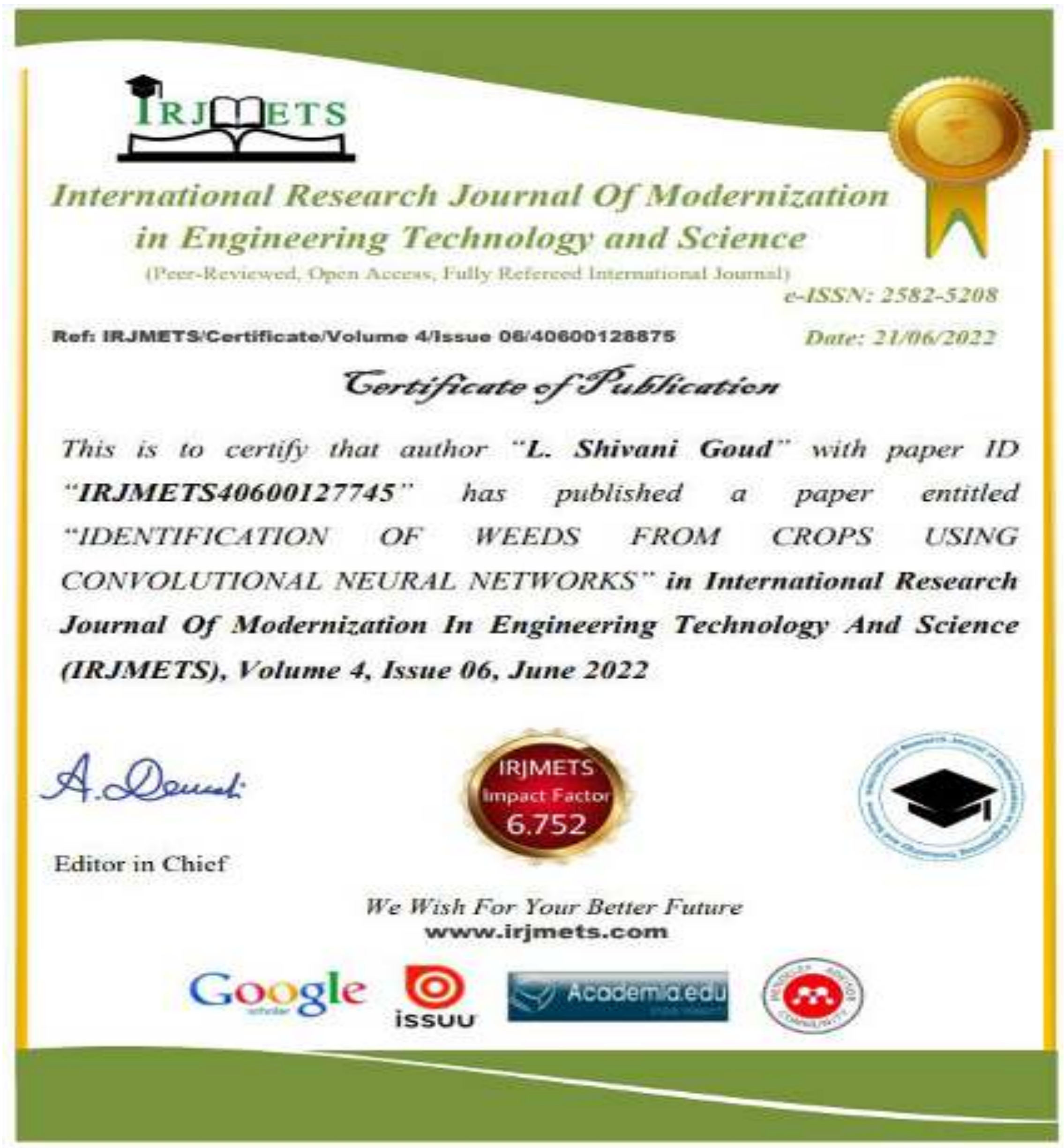
[illegible]

Editor in Chief

IRJMETS Impact Factor 6.752

We Wish For Your Better Future

www.irjmets.com



IRJMETS

International Research Journal Of Modernization in Engineering Technology and Science

(Peer-Reviewed, Open Access, Fully Refereed International Journal)

e-ISSN: 2582-5208

Ref: IRJMETS/Certificate/Volume 4/Issue 06/40600128875 Date: 21/06/2022

Certificate of Publication

This is to certify that author "**L. Shivani Goud**" with paper ID "**IRJMETS40600127745**" has published a paper entitled "IDENTIFICATION OF WEEDS FROM CROPS USING CONVOLUTIONAL NEURAL NETWORKS" **in International Research Journal Of Modernization In Engineering Technology And Science (IRJMETS), Volume 4, Issue 06, June 2022**

[illegible]

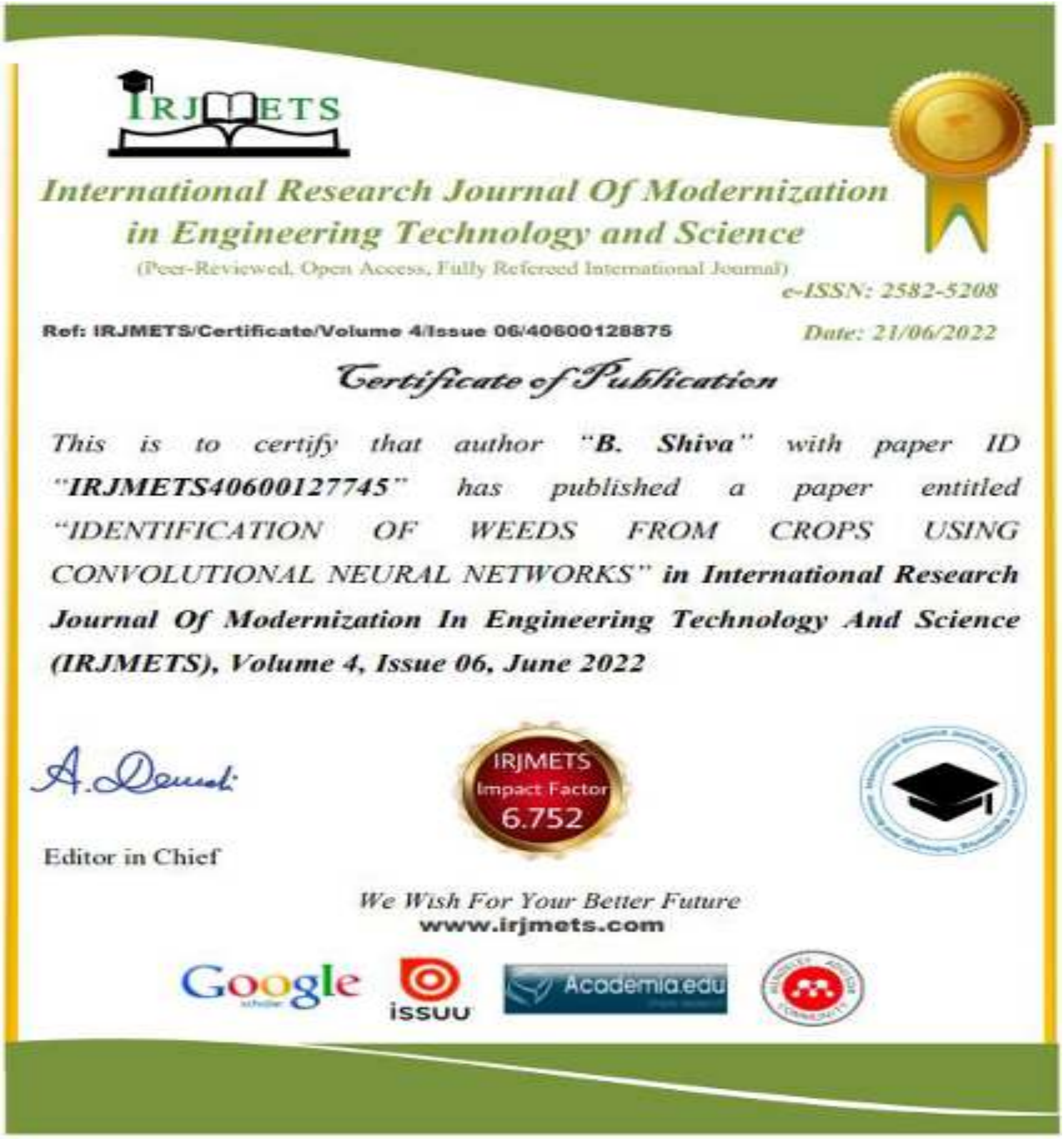
Editor in Chief

IRJMETS Impact Factor 6.752

We Wish For Your Better Future

www.irjmets.com

Google scholar issuu Academia.edu



IRJMETS

International Research Journal Of Modernization in Engineering Technology and Science

(Peer-Reviewed, Open Access, Fully Refereed International Journal)

e-ISSN: 2582-5208

Ref: IRJMETS/Certificate/Volume 4/Issue 06/40600128875

Date: 21/06/2022

Certificate of Publication

*This is to certify that author **"B. Shiva"** with paper ID **"IRJMETS40600127745"** has published a paper entitled "IDENTIFICATION OF WEEDS FROM CROPS USING CONVOLUTIONAL NEURAL NETWORKS" **in International Research Journal Of Modernization In Engineering Technology And Science (IRJMETS), Volume 4, Issue 06, June 2022***

[illegible]

Editor in Chief

IRJMETS Impact Factor 6.752

We Wish For Your Better Future

www.irjmets.com

Google scholar issuu Academia.edu