

# **IDENTIFICATION OF INSECTS AND PREDICTION OF PESTICIDES USING DEEP LEARNING**



**Department of Computer Science  
And Engineering**

**NATIONAL INSTITUTE OF TECHNOLOGY, MANIPUR  
MAY, 2021**

**SUBMITTED BY:**

***Gunakar Challa***

Enrollment Number: 17103032

***Ketavath Prabhakar***

Enrollment Number: 17103048

**GUIDED BY:**

***Mrs. Maibam Mangalleibi Chanu***

*Lecturer*

*Department of Computer Science  
and Engineering*

*National Institute of Technology,  
Manipur*

# **IDENTIFICATION OF INSECTS AND PREDICTION OF PESTICIDES USING DEEP LEARNING**

*Report submitted to  
National Institute of Technology Manipur*

*for the award of the degree*

*of*

**Bachelor of Technology  
in Computer Science & Engineering**

*by*

**Gunakar Challa (17103032)**

**Ketavath Prabhakar (17103048)**

*Under the supervision of*

**Maibam Mangalleibi Chanu**

**Assistant Professor**

**CSE Department, NIT Manipur**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING  
NATIONAL INSTITUTE OF TECHNOLOGY MANIPUR**

**May 2021**

## DECLARATION

I certify that

- a. the work contained in this report is original and has been done by me under the guidance of my supervisor(s).
- b. the work has not been submitted to any other Institute for any degree or diploma.
- c. I have followed the guidelines provided by the Institute in preparing the report.
- d. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
- e. whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the report and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.



Ketavath Prabhakar



Gunakar Challa



राष्ट्रीय प्रौद्योगिकी संस्थान मणिपुर  
**NATIONAL INSTITUTE OF TECHNOLOGY MANIPUR**

Imphal, Manipur, Ph.(0385) 2058566 / 2445812

E-mail : [director@nitmanipur.ac.in](mailto:director@nitmanipur.ac.in) , Website : [www.nitmanipur.ac.in](http://www.nitmanipur.ac.in)

An Autonomous Institute under Ministry of Education, Govt. of India.

No. NITMN.3/(89-Acad)/CSE/B.Tech/Project/2021/-16

Date :- 28<sup>th</sup> May, 2021

**CERTIFICATE**

This is to certify that the Dissertation **Report** entitled, “**Identification of Insects and Prediction of Pesticides using Deep Learning**” submitted by **Mr./Ms. “Gunakar Challa (17103032)”** and **Mr./Ms. “Ketavath Prabhakar (17103048)”** to National Institute of Technology Manipur, India, is a record of bonafide Project work carried out by him/her under my/our supervision and guidance and is worthy of consideration for the award of the degree of Bachelor of Technology in Computer Science & Engineering of the Institute.

\_\_\_\_\_  
**Dr. Khundrakpam Johnson Singh**

Assistant Professor  
(Head of Department, CSE)

\_\_\_\_\_  
**Mrs. Maibam Mangalleibi Chanu**

Lecturer, Dept. of CSE  
(Project Guide)

# ACKNOWLEDGEMENT

On the submission of the thesis report of “**Identification of Insects and Prediction of Pesticides using Deep Learning**”, We would like to articulate our profound gratitude and indebtedness to our project guide **Mrs. Maibam Mangalleibi Chanu**, Department of Computer Science and Engineering, who has always been a constant motivator and guiding factor throughout the project time in and out as well. It has been a great pleasure for us to get an opportunity to work under her and complete the project successfully. We are indebted to her for having helped us shape the problem and providing insights towards the solution.

We wish to extend our sincere thanks to **Dr. Khundrakpam Johnson Singh**, Head of Department, Computer Science and Engineering, for our project with great interest.

We will be failing in our duty if we do not mention the administrative staff and laboratory staff of this department for their timely help.

We would like to thank all whose direct and indirect support helped us in completing the thesis in time.

This thesis would have been impossible if not for the perpetual moral support from our family members and friends. We would like to thank them all.

Gunakar Challa

Ketavath Prabhakar

# ABSTRACT

The project aims to identify insects and also suggest list of right pesticides which doesn't harm the crops and kills the insect. The identification of insects is done using YOLO V4 model trained on custom insects dataset. The images of insects are downloaded from google using automated scripts and are labelled manually to produce annotations in required in YOLO format.

We then mimic the darknet architecture required to train the model. We fine tune the training parameters in accordance with the number of classes of insects, size, learning rate, etc. We then train the model for sufficient number of iterations and choose the best weights of model which neither overfit nor underfit the dataset with low loss.

We then use these weights to generate checkpoints required to upload the model on to UbiOps server. The checkpoint is then optimized further and changed in accordance with UbiOps requirement. We then deploy the model onto server. We now create API tokens required to communicate with the deployed model via APIs provided by UbiOps and also assign suitable roles to it.

Next, we create the database required for pesticide suggestion on MariaDB instance on 000webhost. We also create PHP-APIs required to access the database through front-end.

Finally, we create a Flutter App which can communicate to both UbiOps and 000webhost servers through APIs to accomplish our goals.

# Table of Contents

Title Page .....	i
Cover Page .....	ii
Declaration.....	iii
Certificate.....	iv
Acknowledgement .....	v
Abstract .....	vi
Table of Contents.....	vii
<b>1 INTRODUCTION.....</b>	<b>1</b>
1.1 Introduction .....	1
1.2 Image Recognition.....	2
<b>2 LITERATURE REVIEW .....</b>	<b>3</b>
2.1 Paper 1 .....	3
2.2 Paper 2 .....	3
2.3 Paper 3 .....	4
2.4 Paper 4 .....	4
2.5 Paper 5 .....	4
2.6 Paper 6 .....	5
<b>3 PROPOSED ALGORITHM EXPLANATION .....</b>	<b>6</b>
3.1 YOLO Algorithm .....	6
3.2 Algorithm Explanation .....	6
3.3 Example.....	7
3.4 Non-Max Suppression .....	8
3.5 Darknet Architecture .....	8
<b>4 IMPLEMENTATION 1 – INSECT IDENTIFICATION .....</b>	<b>9</b>
4.1 Downloading Images.....	9
4.2 Labelling the images.....	9
4.3 Training Custom YOLO V4.....	11
4.4 Optimising the model for UbiOps .....	12
4.5 Deploying on to UbiOps.....	14

4.6	Accessing the Deployment Requests Through Builtin API .....	15
4.6.1	Upload Blob Request .....	15
4.6.2	Create Deployment Request .....	16
4.6.3	Download Blob Request .....	16
4.7	Flutter Connection with UbiOps .....	17
4.7.1	Uploading Blob .....	17
4.7.2	Deployment Request .....	17
4.7.3	Download Blob .....	18
5	IMPLEMENTATION 2 – PESTICIDE SUGGESTION .....	19
5.1	Constructing ER Diagram .....	19
5.2	Database Creation .....	20
5.3	Creation of APIs Using PHP .....	22
5.3.1	Get all Crops API .....	22
5.3.2	Get all Insects API .....	22
5.3.3	Get required Pesticides API .....	22
5.4	Flutter Connection to 000webhost and API management .....	23
6	FLOW DIAGRAMS .....	24
7	HARDWARE AND SOFTWARE REQUIREMENTS .....	26
8	INPUT AND OUTPUT .....	27
8.1	Insect Identification .....	27
8.2	Pesticide Suggestion .....	28
9	ADVANTAGES .....	29
10	CONCLUSION AND FUTURE PROSPECTS .....	30
	REFERENCES .....	31



# CHAPTER 1

## INTRODUCTION

### 1.1 Introduction



FIG 1.1 Spraying pesticide

- During farming, a number of agricultural pests attack the crop damaging its health ultimately leading to low and poor-quality yield.
- This situation is mainly controlled by spraying correct pesticide which helps control the spread of these pests.
- Farmers in general aren't much aware about the technicalities of the pesticides.
- They have to choose pesticide which not only kills that specific insect but also should be aware whether the crop is compatible with it or not.
- They usually ask their counterparts or the pesticide vendor for advice.
- They have to rely on others knowledge who might not be experts in that field.
- Our project aims to solve this problem by showing them right pesticides suggested by the experts.
- Our model uses object detection technique to get the name of the pest and uses relational database to get the list of right pesticides.

## 1.2 Image Recognition



Fig 1.2 Image Recognition - Caterpillar

Object detection is a computer vision approach for identifying and locating things in images and videos. Object detection may be used to count items in a scene, determine and monitor their precise positions, and precisely label them using this type of identification and localization.

## **Chapter 2**

### **Literature Review**

In this chapter, we will take a brief look into some papers based on YOLO. We will analyse the projects, take note of their purpose, implementation, advantages and limitations. Finally, we will discuss how we have understood and overcome those limitations in our project.

#### **2.1 Paper 1: YOLO based Human Action Recognition and Localization <sup>[1]</sup>**

In this paper, the authors have used LIRIS dataset to train the original YOLO (V1) model to recognise and localize the human action in a video. The project takes in 30 frames of a video and try to detect action in each frame and if the model is able to detect the same action in 5 different frames with a confidence level above 0.5 then the action is concluded.

The project is good at the time of build, but today we have more advanced YOLO V4 algorithm which can infer data faster and more accurate.

#### **2.2 Paper 2: Visual Object Detection and Tracking using YOLO and SORT <sup>[2]</sup>**

In this paper, the author has used YOLO and SORT algorithms to identify and tack objects in a video. The author has just used 800 images for training and 200 images for validation. This has restricted them to train their model for 320 epochs only. Though they have trained their model on previously trained weights on COCO dataset, this low number of images tends to make the model less confident on new data. Even if they train for more epochs, this will only lead to overfitting of the model on training data.

## **2.3 Paper 3: Vehicle Detection and License Plate**

### **Recognition using Deep Learning <sup>[3]</sup>**

In this paper the author has trained the YOLO model on 200 images and validated the same using 8000 images, which is a blunt mistake. The author has later corrected himself by training on 400 images and validate on 100 images. But the author has used the first weights itself for later in the project due to performance issues and time constraints. The ideal mixture of images is about training on 2000 images and validate on 1000 images according to official darknet documentation.

## **2.4 Paper 4: Object Detection with YOLO on Artwork**

### **Dataset<sup>[4]</sup>**

The author has used the original YOLO model to detect the artwork dataset. But the author's model was not up to the mark when inferencing because of the non-fine tuning of training parameters. The precision of their model is just 40% and is the result of aforementioned reason. The author could have spent more time on maintaining standards for tuning of parameters.

## **2.5 Paper 5: Tinier-YOLO: A Real-Time Object Detection**

### **Method for Constrained Environments <sup>[5]</sup>**

In this paper, the authors cum inventors have created a new lightweight model called Tinier-YOLO which is derived from Tiny-YOLO V3. They have tweaked the connection style in the dense layers of the model to shrink the model at the same time improving the model detection performance. They have abled to achieve a mean average precision of 65.7% on PASCAL VOC.

The problem in lightweight models is that they are shrunk down in size to be able to be deployed onto small devices like smartphones or embedded devices at the cost of

performance. But today we have servers like UbiOps wherein we can directly deploy our models onto them without major compromises on performance.

## **2.6 Paper 6: Final Project Report - YOLO on iOS [6]**

In this paper cum project, the author has replaced the conventional convolutional layer of the Tiny YOLO model with custom convolutional layer using metal shading language also handling the low-level metal interface to run on an iPhone, thereby increasing the performance of convolutional layer by 30%. But the problem with such an approach is that the heavy computation is done on client side, and many people cannot afford to buy high end smartphones. Instead one can deploy onto ML Ops server for inferencing.

We have gained the knowledge from all these papers & their project implementations. We have also learnt their limitations and avoided them in our implementation. We have carefully designed, developed and optimised our object detection model as follows:

1. Firstly, we have used the latest cutting-edge technology and algorithm like YOLO V4 for our project.
2. We have created large dataset, almost 2000 images per class, which are enough to predict the object accurately with considerable amount of confidence.
3. Then have taken care of training & validation images mixture ratio and have split the dataset into approximately 1500 training images & around 500 validation images for each class.
4. Later we decided final training parameters after several revisions of trial training.
5. After training with finalised parameters, instead converting to Tiny YOLO weights we have generated checkpoints which are far more performant than aforementioned weights.
6. Finally, we have deployed it onto server instead of running it on the client side, thus delivering fast computation regardless of client-side machine performance capability.

## Chapter 3

# PROPOSED ALGORITHM EXPLANATION

### 3.1 YOLO Algorithm

- YOLO Algorithm originally developed for object detection holds the capability to distinguish and detect several different objects in image.
- It is extremely fast and accurate.
- YOLO sees the entire image during training and test time so it so it implicitly encodes contextual information about classes as well as their appearance.
- YOLO develops generalizable representations of objects, outperforming existing top detection approaches when trained on natural photos and evaluated on artwork.
- Updated YOLO versions can detect more than 9000 classes.

### 3.2 Algorithm explanation <sup>[7]</sup>

1. Yolo algorithm takes an input image of shape (608, 608, 3).
2. This picture is fed into a convolutional neural network (CNN), which produces a 19-dimensional output (19, 19, 5, 85).
3. The output volume of (19, 19, 425) is obtained by flattening the final two dimensions of the above output:
  1. A 19 X 19 grid delivers 425 numbers in each cell.
  2. The number of anchor boxes each grid is 5, hence  $425 = 5 * 85$ .
  3.  $85 = 5 + 80$ , where 5 is (pc, bx, by, bh, bw) and 80 is the number of classes we want to detect
4. Finally, we do the IoU and Non-Max Suppression to avoid selecting overlapping boxes.

### 3.3 Example

We'll start by dividing a picture into grids, say 3x3.



Fig 3.1 3x3 Grid

We estimate one or more classes for each grid based on the number of anchor boxes per grid, say 2.

y =	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3
	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3

Fig 3.2 Output Details

Bounding box coordinates are determined in relation to the grid they contain.

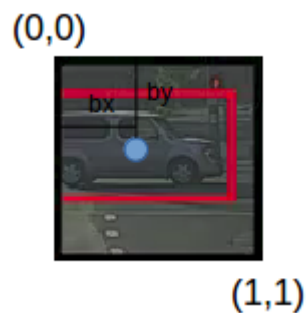


Fig 3.3 Bx By Coordinate System

While it is possible that it will forecast the same item several times, we use non-max suppression to avoid this possibility.

### **3.4 Non-Max Suppression**

1. All boxes with probability less than or equal to a pre-determined threshold are discarded (say, 0.5).
2. For the remaining boxes:
  1. As the output forecast, choose the box with the highest likelihood.
  2. Any additional box with an IoU larger than the threshold should be discarded together with the output box from the previous phase.
3. Step 2 should be repeated until all of the boxes have been either selected as the output prediction or rejected.

This deep learning model is built and trained using Darknet architecture.

### **3.5 Darknet Architecture**

Darknet is primarily used for object detection, and its design and features differ from those of other deep learning frameworks. It outperforms several other NN designs and techniques, such as Faster RCNN and others. If we require speed, then must use C, and most deep neural network frameworks are written in C. TensorFlow, in contrast, has a larger reach, whereas Darknet architecture and YOLO are specialised frameworks that excel in speed and precision. YOLO can be run on a CPU; however, it runs 500 times faster on a GPU thanks to the use of CUDA and cuDNN.



## Chapter 4

# IMPLEMENTATION 1 – INSECT IDENTIFICATION

### 4.1 Downloading images <sup>[8]</sup>

We download simple image download package through anaconda prompt using the following command:

```
Pip install simple_image_download
```

We then write a script, say `download_from_google.py` and include simple image download as header file. This script then downloads the images from google up to the number specified. Then run the following command in the anaconda prompt:

```
Python download_from_google.py
```

The images are then downloaded to the simple images folder into sub folders of each category.

### 4.2 Labelling the images

1. We then assemble all images in one folder to draw a bounding box and label them.
2. We download a tool called Binary from Darrenl Tzutalin's labelImg repository.

Link for binary: <https://github.com/tzutalin/labelImg/releases/tag/v1.8.1>

3. We now run `labelImg.exe` and label the images as shown in the following screenshot.

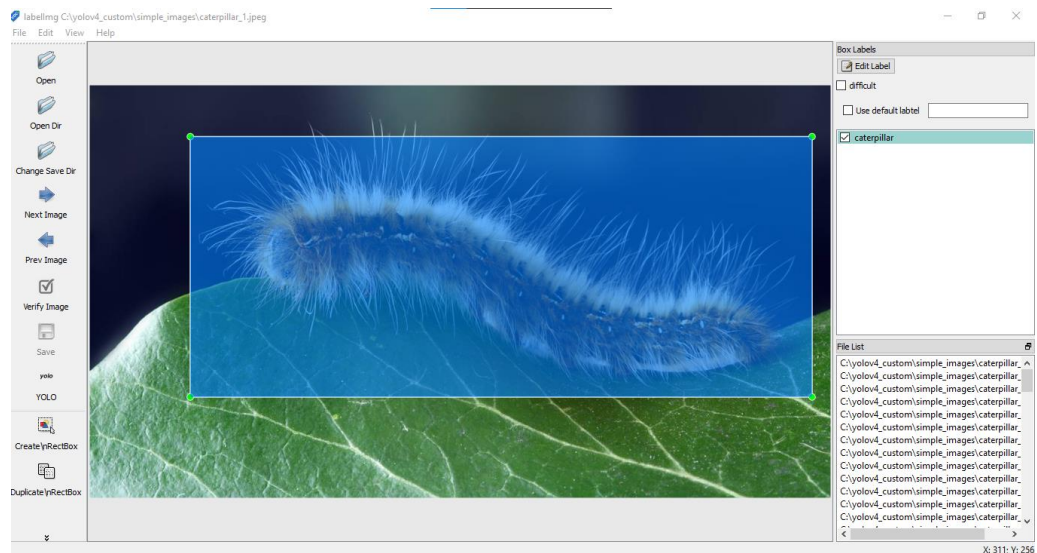


Fig 4.1 LabelImg Tool

4. We set the save format to Yolo and draw a bounding box around insects as precise as possible.
5. After saving, we will find a .txt file against each image containing annotations of class and bounding box.

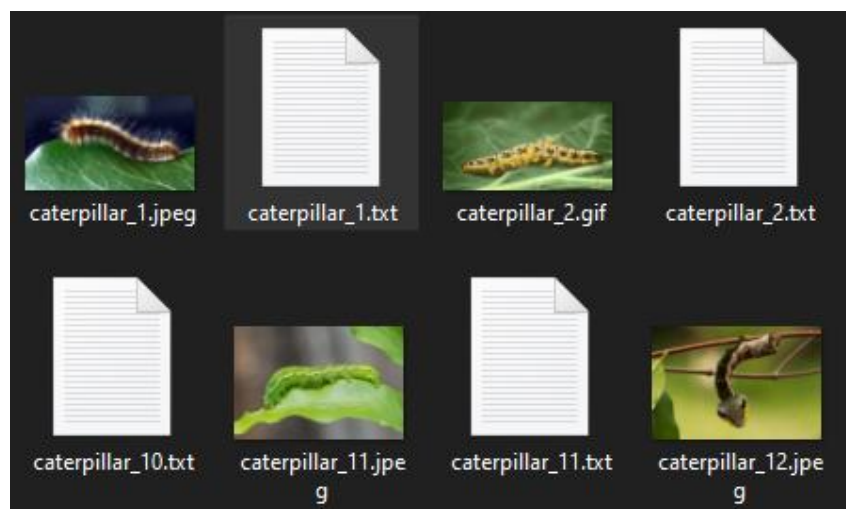


Fig 4.2 Annotations of Labelled Dataset

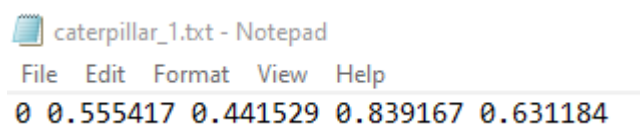


Fig 4.3 Annotation details

## 4.3 Training Custom YOLO V4 <sup>[9]</sup>

1. We first build darknet yolo v4 with GPU support.
2. In darknet folder we would find a x64 folder, in that, open data folder and create a folder (name as obj) for labelled images.
3. We would then create a copy of two files obj.data and obj.names and rename with obj. Provide class names and data then hit save.
4. Next, open cfg folder and make a copy of yolo4-custom.cfg then open with any text editor and modify as follows.

Training Yolo v4 (and v3):

0. For training `cfg/yolov4-custom.cfg` download the pre-trained weights-file (162 MB): [yolov4.conv.137](#) (Google drive mirror [yolov4.conv.137](#) )
1. Create file `yolo-obj.cfg` with the same content as in `yolov4-custom.cfg` (or copy `yolov4-custom.cfg` to `yolo-obj.cfg`) and:
  - change line `batch` to `[ batch=64 ]`
  - change line `subdivisions` to `[ subdivisions=16 ]`
  - change line `max_batches` to `( classes*2000`  but not less than number of training images, but not less than number of training images and not less than `6000` ), f.e. `[ max_batches=6000 ]` if you train for 3 classes
  - change line `steps` to 80% and 90% of `max_batches`, f.e. `[ steps=4800,5400 ]`
  - set network size `width=416 height=416` or any value multiple of 32
  - change line `classes=80` to your number of objects in each of 3 `[yolo]` -layers:
  - change `[ filters=255 ]` to `filters=(classes + 5)x3` in the 3 `[convolutional]` before each `[yolo]` layer, keep in mind that it only has to be the last `[convolutional]` before each of the `[yolo]` layers.

Fig 4.4 Training Parameters

5. We then download `yolov4.conv.137` file that is basically a pre-trained weights file that utilize in training and place inside x64 folder.
6. Next we create list of images by using following python command:

```
python create_list_of_images.py
```
7. We start training by using the following command:

```
darknet.exe detector train data/obj.data yolo-obj.cfg yolov4.conv.137
```
8. After completion of training the following data graphs are created.

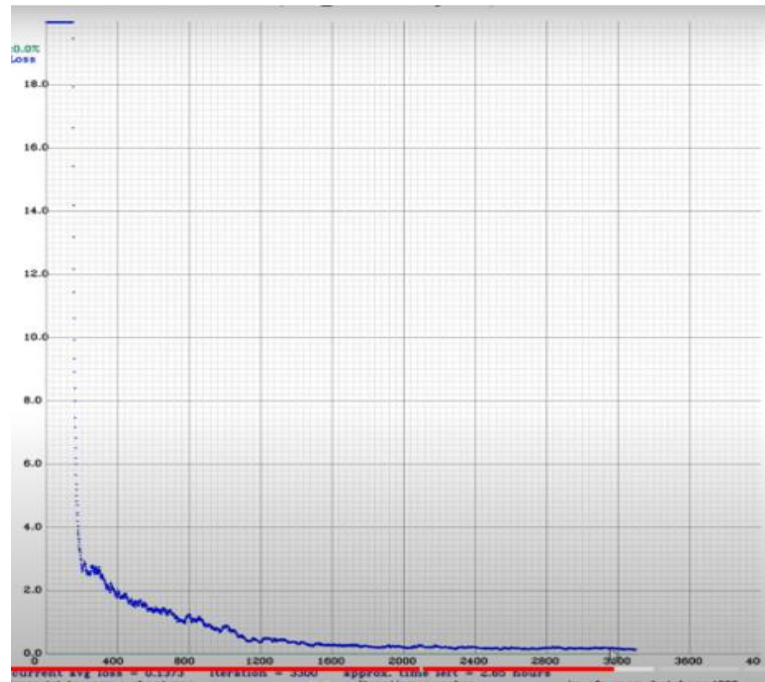


Fig 4.5 Training Chart

9. If we go to backup folder, there will be weight files after every 1000 iterations.
10. If we are unsatisfied with accuracy of the model then we can resume the training of model for better performance.

11. For resume the training use the following command:

```
darknet.exe detector data/obj.data cfg/yolov4-obj.cfg backup/yolov4-obj_last.weights -c 0
```

12. To start inferencing on custom trained model we issue the following command:

```
darknet.exe demo data/obj.data cfg/yolov4-obj.cfg backup/yolov4-obj_last.weights wick.mp4 -thresh 0.6
```

13. To run object detection on images execute the following command:

```
darknet.exe detector test data/obj.data cfg/yolov4-obj.cfg backup/yolov4-obj_last.weights
```

## 4.4 Optimising the model for UbiOps <sup>[10]</sup>

- 1 Download the following the zip file *tensorflow-yolov4-tflite* from github and extract it.

Link: <https://github.com/hunglc007/tensorflow-yolov4-tflite>

- Copy the yolov4-obj\_last.weights file of darknet into the data folder of tensorflow-yolov4-flite and also copy obj.names file into the classes folder of tensorflow-yolov4-flite.insode core folder open config.py and change classes to obj.names as in following figure.

```
1 #!/usr/bin/env python
2 # coding=utf-8
3 from easydict import EasyDict as edict
4
5
6 __C = edict()
7 # Consumers can get config by: from config import cfg
8
9 cfg = __C
10
11 # YOLO options
12 __C.YOLO = edict()
13
14 __C.YOLO.CLASSES = "./data/classes/obj.names"
15 __C.YOLO.ANCHORS = [12,16, 19,36, 40,28, 36,75, 76,55, 72,146, 142,110, 192,243, 459,481]
16 __C.YOLO.ANCHORS_V3 = [10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326]
17 __C.YOLO.ANCHORS_TINY = [23,27, 37,58, 81,82, 81,82, 135,169, 344,319]
18 __C.YOLO.STRIDES = [8, 16, 32]
19 __C.YOLO.STRIDES_TINY = [16, 32]
20 __C.YOLO.XYSCALE = [1.2, 1.1, 1.05]
21 __C.YOLO.XYSCALE_TINY = [1.05, 1.05]
22 __C.YOLO.ANCHOR_PER_SCALE = 3
23 __C.YOLO.IOU_LOSS_THRESH = 0.5
24
25
26 # Train options
27 __C.TRAIN = edict()
28
29 __C.TRAIN.ANNOT_PATH = "./data/dataset/val2017.txt"
30 __C.TRAIN.BATCH_SIZE = 2
31 # __C.TRAIN.INPUT_SIZE = [320, 352, 384, 416, 448, 480, 512, 544, 576, 608]
32 __C.TRAIN.INPUT_SIZE = 416
33 __C.TRAIN.DATA_AUG = True
34 __C.TRAIN.LR_INIT = 1e-3
```

Fig 4.6 Config.py File

- Then we issue the following command to generate checkpoints for the model:

```
python save_model.py --weights ./data/yolov4.weights
--output ./checkpoints/yolov4-416 --input_size 416 -
-model yolov4
```

- To run object detection on generated checkpoints we use the following command:

```
python detect.py --weights ./checkpoints/yolov4-416 -
-size 416 --model yolov4 --image ./data/ant.jpg
```

- Now that we have a working model we need to adapt it a bit so that UbiOps can use it. Now we have the "detect.py" with

the functionality in it to classify images like we just did. UbiOps however expects a model.py with a specific structure.

- 6 The model.py needs to have a “model” class which contains 2 methods. The methods are init and request. UbiOps can also use a requirement.txt to automatically install dependencies.
- 7 fed the image into the classifier using this line:

```
original_image = cv2.imread(data["image_input"])
```

- 8 And the output image using these lines:

```
cv2.imwrite('output.jpg', image)
return {'image_output': "output.jpg"}
```

- 9 Because UbiOps does not provide a desktop environment and the opencv library, change opencv-python to opencv-python-headless and also add as a requirement in requirements.txt
- 10 Remove cv.show() call, which requires a desktop environment.
- 11 We also add ubiops.yaml file to install some missing OS specific files. In our case we add libglb2.0-0 package as follows:

```
apt:
  packages:
  - libglb2.0-0
```

## 4.5 Deploying onto Ubiops

- 1 Go to UbiOps and login. In the sidebar on the left go to **Models** -> **Create**. Set the **Name**. Set **Input type** to *structured*. Add an input field image\_input with data-type *file*. Set **Output type** to *structured*. Add an output image\_output with data-type *file*.
- 2 Input and Output in UbiOps: Click **Next Step** and then **Confirm**. Set the **Language** to Python 3.7. Upload the zipped model package and click on the Create button. Model can be seen in the **Models overview** page.
- 3 Click on **Models** in the sidebar on the left and then click on your model name(yolov4) and then on one of the versions (v1). Now click **CREATE DIRECT REQUEST** and upload an image to UbiOps. Click **Create** to create the request.

- 4 When the model has finished processing we can click the notification in the right bottom named **Results** and see the results of our request.

## 4.6 Accessing the Deployment Requests Through Built-in API

We first create API authentication token in Users & Permissions. Next, we assign deployment-admin & blob-admin roles to our generated token.

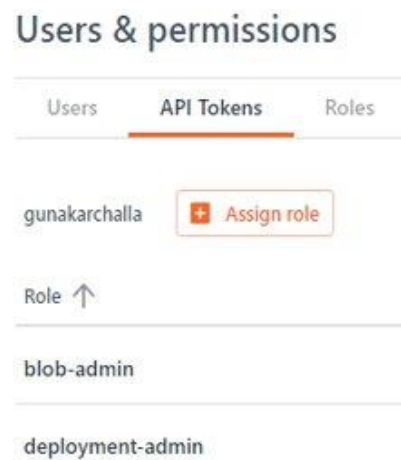


Fig 4.7 Users & Permissions

We add Authorisation and Token as key value pair as header in every HTTP request.

### 4.6.1 Upload blob request

In order to upload blob/file onto UbiOps server we have to send HTTP POST request from client side as depicted in the following picture.



Fig 4.8 Upload Blob URL

If we can successfully upload the blob we get a response as shown in picture:



```
{
  "id": "b58fb853-9311-4583-9688-abad61830abc",
  "creation_date": "2020-05-18T11:26:57.904+00:00",
  "last_updated": "2020-05-18T11:26:57.904+00:00",
  "filename": "original-filename.jpg",
  "size": 3439,
  "ttl": 259200
}
```

Fig 4.9 Upload Blob Response

## 4.6.2 Create Deployment Request

In order to call deployment request viz run YOLO V4 detection algorithm on uploaded picture we send a HTTP POST request as shown in picture:

**POST** `/projects/{project_name}/deployments/{deployment_name}/request` Create deployment requests

Fig 4.10 Deployment Request URL

We also send blob id as key value pair in json format as shown below:

```
{
  "input-field-1": 5.0,
  "blob-input-field": "f52ff875-4980-4d71-9798-a469ef8cece2"
}
```

Fig 4.11 Deployment Request Body

If we are able to call deployment request successfully then we would get a response as follows:

```
{
  "version": "v2",
  "success": true,
  "result": {
    "output-field-1": "2.1369",
    "output-field-2": "5.5832",
  },
  "error_message": None
}
```

Fig 4.11 Deployment Request Response

## 4.6.3 Download Blob Request

In order to download blob/file from UbiOps server we have to send HTTP GET request from client side as depicted in the following picture:



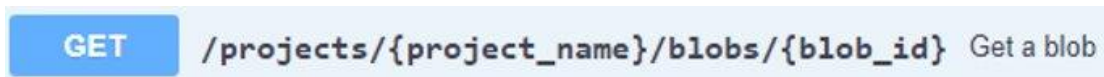


Fig 4.12 Download Blob URL

## 4.7 Flutter Connection with UbiOps

We have used Image\_Picker package provided by flutter to create a widget wherein one can choose the picture of insect from gallery of their phones and save it in a file variable.

### 4.7.1 Uploading Blob

We have used Flutter's multi part request function to send the image file in HTTP request to UbiOps deployment endpoint. We have added Authorisation header and assigned the value of Token generated by us. Also added blob-ttl header which decides how much time(in seconds) the blob should stay in server before automatic deletion and assigned a value of 900(15 minutes). This reduces unnecessary storage of uploaded files.

The response specifies various parameters like id, ttl, size, creation\_date,etc Out of which the main parameters to be used is id of the blob uploaded.

```
I/flutter ( 7619): Status code of uploading blob request: 201
I/flutter ( 7619): Response for uploading blob request: {"id":"3708c956-33a5-4f25-9704-87a9a38b28d5","created_by":"fdd3d153-04ac-448e-939b-723a03d0ede4.bryclay@serviceuser.ubiops.com","creation_date":"2021-05-26T01:09:57.515+00:00","last_updated":"2021-05-26T01:09:57.515+00:00","filename":"phone_input.jpg","size":21955,"ttl":900}
```

Fig 4.13 Status code & Response of Uploading Blob

### 4.7.2 Deployment Request

- We have used multi part request function to call deployment request. We have to add Authorization token in header. Also we have to specify the blob id, in

the body of request in json format, on which the deployment request has to run.

- The request takes 30 to 40 seconds depending upon the image size and internet speed of mobile phone.
- The response specifies various parameters like request\_id, version, success, result, etc. Out of which the main parameters to be used in next step is result parameter which specifies the output image blob id of the deployment request.

```
I/flutter ( 7619): Status code of deployment request: 200  
I/flutter ( 7619): Response for deployment request: {"request_id":"e25701e0-c5a2-45d7-acb3-98e8ab122e6a","version":"v1","success":true,"error_message":"","result":{"image_output":"e184fc01-cef1-446f-9884-57f1e09af498"}}
```

Fig 4.14 Status code & Response of Deployment Request

### 4.7.3 Downloading Blob

- We have used NetworkImage() function to download and show the output blob. We have to add Authorization token in header. We have used an asset image (downloading picture) as a place holder until the downloading finishes and is displayed. Also we have to specify the blob id, in the URL of the request.
- The will be getting file in response body.

## CHAPTER 5

# IMPLEMENTATION 2 – PESTICIDE SUGGESTION

### 5.1 Constructing ER Diagram

The relation between them is very complex in nature, therefore Pesticide selection procedure is implemented using relational databases like MySQL.

- A pesticide can kill multiple insects and an insect can be killed by many pesticides.
- Also, each insect can be killed by at least one pesticide and each pesticide can kill at least one insect.
- Similarly, a pesticide is compatible with multiple different crops and a crop is compatible with many pesticides
- Also, each pesticide is compatible with at least one crop and each crop is compatible with at least one pesticide.

To build a suggestion system we have to model a relational database as per the following ER diagram.

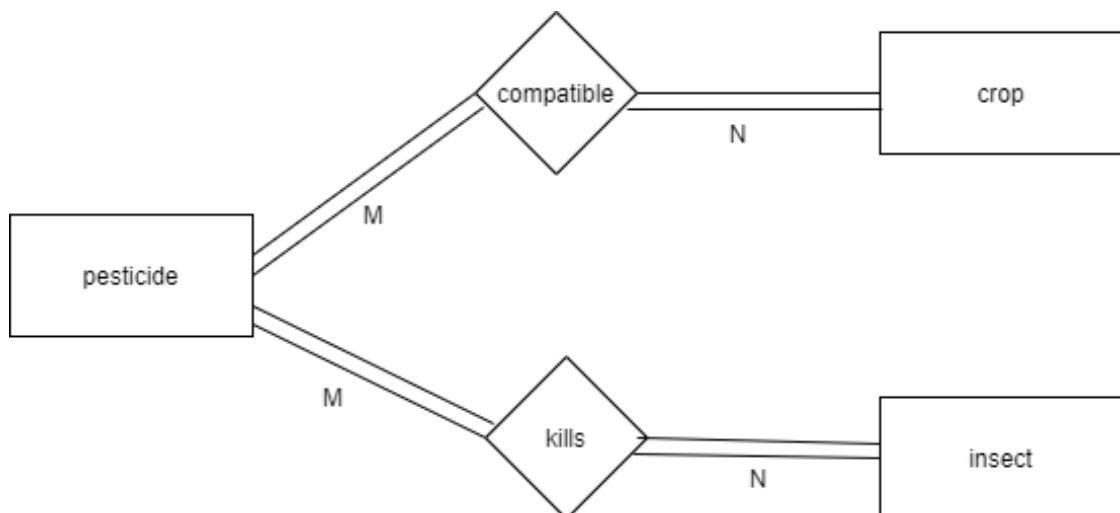


Fig 5.1 ER Diagram of Insects, Crops & Pesticides

## 5.2 Database Creation

We construct a relational database to suit the ER diagram as follows:

### PesticideSuggestionDB

```
{  
Pesticides(Pesticide id, Pesticide name, Cost, Volume, Waiting period, Footnotes)  
Insects(Insect id, Insect name, Foot notes)  
Crops(Crop id, Crop name, Foot notes)  
Compatible( Pesticide id, Crop id )  
Kills(Pesticide id, Insect id)  
}
```

One can create the database in any server, we have used Maria DB instance in 000webhost.



Fig 5.2 Database

Fill in the Database with appropriate data with the help of the expert.

In foot notes of pesticides table, we can store URLs of e-commerce sites to facilitate the purchase of pesticide for the user.

Pesticide id	Pesticide name	Cost	Volume	Waiting period	Foot notes
1	Acephate	450	0.50	15	<a href="https://www.bighaat.com/products/acemain-acephate-...">https://www.bighaat.com/products/acemain-acephate-...</a>
2	Aluminum Phosphide	180	0.80	2	<a href="https://www.indiamart.com/proddetail/aluminium-pho...">https://www.indiamart.com/proddetail/aluminium-pho...</a>
3	Bromadiolone	220	1.00	14	<a href="https://www.indiamart.com/proddetail/bromadiolone-...">https://www.indiamart.com/proddetail/bromadiolone-...</a>
4	Buprofezin	180	1.00	20	<a href="https://www.bighaat.com/products/apple-insecticide">https://www.bighaat.com/products/apple-insecticide</a>

Fig 5.3 Pesticides Table

In footnotes of insect table, we can store the text like details about how it damages the crop or we can store URL like in pesticides table which can point to an article about it.

Insect id	Insect name	Foot notes
0	Ant	Damage by disrupting soil under roots.
1	Beetle	Damage by consumption and boring of stem.
2	Caterpillar	damage by consumption.
3	Moth	damage by consumption.

Fig 5.4 Insects Table

In footnotes of crop table, we can store the text like details about crop management or we can store URL like in pesticides table which can point to an article about it.

Crop id	Crop name	Foot notes
1	Cotton	<i>NULL</i>
2	Rice	<i>NULL</i>
3	Wheat	<i>NULL</i>
4	Millet	<i>NULL</i>

Fig 5.5 Crops Table

In compatible table, we have to store in data about which pesticides are compatible with what all crops.

Pesticide id	Crop id
1	1
1	2
2	4
3	2
3	3
4	1
4	2

Fig 5.6 Compatible Table

In kills table, we have to store in data about which pesticides kill what all insects.

Pesticide id	Insect id
1	3
2	1
2	2
3	0
4	1
4	3

Fig 5.7 Kills Table

## 5.3 Creation of APIs Using PHP

### 5.3.1 Get all Crops API

- We should create a PHP script which acts as API in file manager of 000webhost.com which sends the crops available for selection to client.
- There we have to create variables which store the credentials of database. Then we create a connection variable to database using `mysqli_connect` function.
- We then execute "SELECT \* FROM Crops" SQL query and store all results in JSON format.
- We then send this json tree to client calling the API.

```
I/flutter ( 7619): Status code for get all crops request: 200
I/flutter ( 7619): Response for get all crops request: [{"Crop id":"1","Crop name":"Cotton","Foot notes":null}, {"Crop id":"2","Crop name":"Rice","Foot notes":null}, {"Crop id":"3","Crop name":"Wheat","Foot notes":null}, {"Crop id":"4","Crop name":"Millet","Foot notes":null}]
```

Fig 5.8 Output of getallcrops.php

### 5.3.2 Get all Insects API

- Script is almost same as get all crops but small change in SQL query.
- SQL query: "SELECT \* FROM Insects"

```
I/flutter ( 7619): Status code for get all insects request: 200
I/flutter ( 7619): Response for get all insects request: [{"Insect id":"0","Insect name":"Ant","Foot notes":"Damage by disrupting soil under roots."}, {"Insect id":"1","Insect name":"Beetle","Foot notes":"Damage by consumption and boring of stem."}, {"Insect id":"2","Insect name":"Caterpillar","Foot notes":"damage by consumption."}, {"Insect id":"3","Insect name":"Moth","Foot notes":"damage by consumption."}]
```

Fig 5.9 Output of getallinsects.php

### 5.3.3 Get required pesticides API

- Script is almost same as get all crops but change in SQL query and body parameters has to be added so as to select combination of insect and crop.
- SQL query: "SELECT \* FROM Pesticides WHERE `Pesticide id` IN (SELECT C.`Pesticide id` FROM Compatible AS C INNER JOIN Kills AS K ON C.`Pesticide id`=K.`Pesticide id` WHERE C.`Crop id`=(SELECT `Crop id` FROM Crops WHERE `Crop name`='\$crop') AND K.`Insect id`=(SELECT `Insect id` FROM Insects WHERE `Insect name`='\$insect'))"

```
I/flutter ( 7619): Status code for get required pesticides request: 200
I/flutter ( 7619): Response for get required pesticides request: [{"Pesticide id":"4","Pesticide name":"Buprofezin","Cost":"180","Volume":"1.00","Waiting period":"20","Foot notes":"https://www.bighaat.com/products/apple-insecticide"}]
```

Fig 5.10 Output of getrequiredpesticides.php

## 5.4 Flutter Connection to 000webhost and API management

- We have used http package provided by flutter to send http get request to fetch crops & insects and http post request to fetch pesticides.
- We have added body parameters of type form-data to get required pesticides query.
- We have saved the response json tree in an Map, which is equivalent to dictionary in python, which is a data structure used to store data in key-value pairs.
- Then we iterated through map to save each name of insect/crop in a list of strings.
- We have used builder functions to create a list item widget for each of pesticide received in json tree.

## CHAPTER 6

### FLOW DIAGRAMS

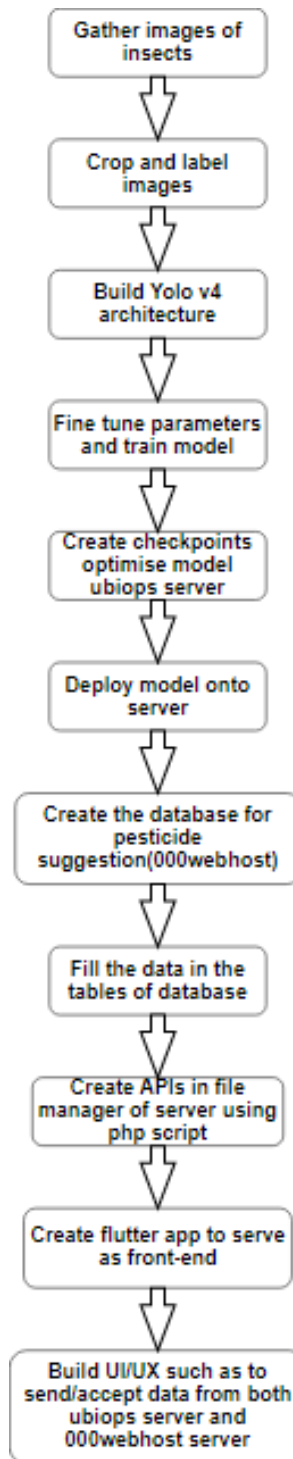


Fig 6.1 Flow Chart for Building the Project



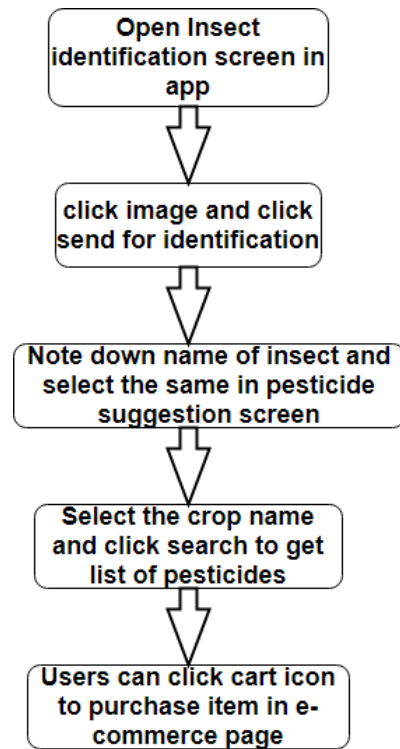


Fig 6.2 Flow Chart for Using the App

## **CHAPTER 7**

# **HARDWARE AND SOFTWARE REQUIREMENTS**

- Laptop/Desktop with at least following specifications:

Hardware: 1.Free storage 5GB

2. 8 GB RAM

3. Intel core i3 8th generation.

Software: 1. Windows OS with windows 7 SP1 (or) macOS with version 10.10

2. Git 2.0

- Smartphone/Mobile with at least following specifications:

Hardware: 1.Free storage 200MB

2. Processor with at least one core more than 1GHz

3. 1GB RAM and Camera 5MP

Software: Android OS 4.1 (or) iOS 8

- Internet connection with speed at least 0.5 Mbps.

## CHAPTER 8

# INPUT AND OUTPUT

### 8.1 Insect identification

#### Input:

We select image of insect to be identified and send it for identification.

Selection



Fig 8.1 Undetected Image

#### Output:

We get an identified image

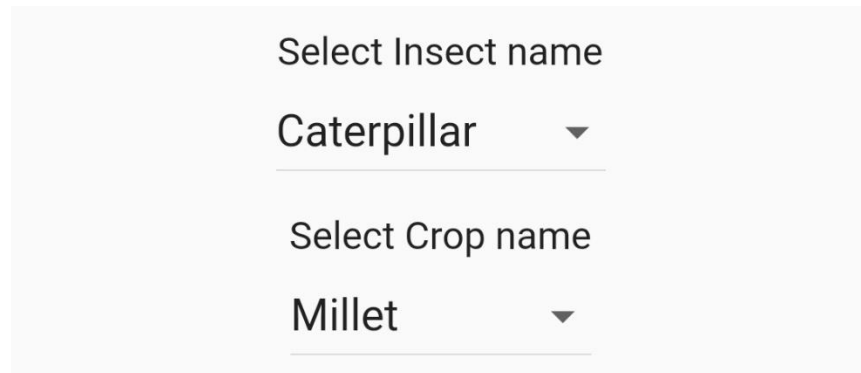


Fig 8.2 Detected Image

## 8.2 Pesticide suggestion

### Input:

Selection of insect and crop combination

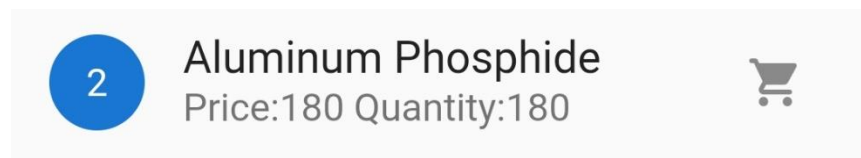


The screenshot shows a light gray rectangular box containing two dropdown menus. The first dropdown is labeled 'Select Insect name' and has 'Caterpillar' selected, with a downward arrow to its right. The second dropdown is labeled 'Select Crop name' and has 'Millet' selected, also with a downward arrow to its right.

Fig 8.3 Insect & Crop Selection

### Output:

List of pesticides



The screenshot shows a single item in a list, represented by a light gray horizontal bar. On the left is a blue circle with the number '2'. To its right is the text 'Aluminum Phosphide' in a large font, and 'Price:180 Quantity:180' in a smaller font below it. On the far right is a gray shopping cart icon.

Fig 8.4 Pesticides List

## CHAPTER 9

### ADVANTAGES

- Whole process uses minimal computation power on smartphone, hence is very efficient and doesn't require high end smartphones.
- Farmer is able to get details of compatible pesticides using his smartphone within minutes, hence process is very convenient.
- Database is filled with the help of experts, therefore the information retrieved is very correct and reliable.
- The whole recognition model is deployed in UbiOps server hence it is very convenient to deploy improvised model.
- The suggestion system is deployed in 000webhost server hence it is easy to increase the classes of crops and pesticides.
- This is a viable business model wherein admin can use affiliate links to generate revenue instead of annoying ads.

## CHAPTER 10

# CONCLUSION AND FUTURE PROSPECTS

In summary, we have downloaded insect images using python script and binaries. We have labelled dataset using \_\_\_\_ tool. Then we have built the darknet architecture with GPU support. We have fine tuned the yolo v4 training parameters and trained it for sufficient iterations. We then create the checkpoints and modify the model so as to make it compatible to deploy onto UbiOps server.

We have created a deployment of yolo v4 onto UbiOps and have processed deployment requests through built-in APIs. We then created database consisting of 5 tables to store the data regarding pesticides, insects and crops. We have also created PHP-APIs to send the data to front-end.

Finally, we have built the front-end mobile app using Flutter where we have built the UI/UX so as to accommodate insect recognition system and pesticide suggestion system using HTTP requests and responses. We have also included URL launcher package so as to facilitate users to purchase right pesticides from e-commerce websites through affiliate links. This can generate enough revenue from app to sustain the whole project.

We have trained the model to recognize four classes of insects for the purpose of presentation but the project can be scaled to recognize almost 9000 different insects. Also, present database structure allows suggest pesticides for crops affected by single insect, this can be changed by changing the database structure. We can further provide/publish articles related to insects, crops & pesticides, which enhances users knowledge regarding this domain.

## REFERENCES

- [1] Shubham Shinde, Ashwin Kothari, Vikram Gupta, “YOLO based Human Action Recognition and Localization”, *International Conference on Robotics and Smart Manufacturing*,  
<https://www.sciencedirect.com/science/article/pii/S1877050918310652/pdf?md5=a8f0aea2e42cf5b00fbc2a636d0ea147&pid=1-s2.0-S1877050918310652-main.pdf>, July 2018.
- [2] Prof. Grishma Sharma, Akansha Bathija, “Visual Object Detection and Tracking using YOLO and SORT”, *International Journal of Engineering Research & Technology*,  
<https://www.ijert.org/research/visual-object-detection-and-tracking-using-yolo-and-sort-IJERTV8IS110343.pdf>, November 2019.
- [3] Prof. Jie Liang, Arlene Fu, Ricky Chen, Toky Saleh, Karamveer Dhillon, “Vehicle Detection and License Plate Recognition using Deep Learning”, *Simon Fraser University*, <http://www.sfu.ca/~jfa49/Files/Vehicle424.pdf>, 2019.
- [4] Yihui He , “Object Detection with YOLO on Artwork Dataset”, *Xi'an Jiaotong University* [https://nbviewer.jupyter.org/github/yihui-he/Objects-Detection-with-YOLO-on-Artwork-Dataset/blob/master/Report\\_Yihui.pdf](https://nbviewer.jupyter.org/github/yihui-he/Objects-Detection-with-YOLO-on-Artwork-Dataset/blob/master/Report_Yihui.pdf), June 2016.
- [5] Wei Fang, Lin Wang, Peiming Ren, “Tinier-YOLO: A Real-Time Object Detection Method for Constrained Environments”, *IEEE*,  
<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8941141>, December 2019.
- [6] Yang You, Tianyi Jin, Jingxuan Zhang, “Final Project Report - YOLO on iOS”, *Connelly Barnes*,  
[http://www.connellybarnes.com/work/class/2017/intro\\_vision/final\\_writeup\\_yang\\_yo\\_u.pdf](http://www.connellybarnes.com/work/class/2017/intro_vision/final_writeup_yang_yo_u.pdf), June 2017.
- [7] Pulkit Sharma, ”A Practical Guide to Object Detection using the Popular YOLO Framework – Part III (with Python codes)”, *Analytics Vidhya*,  
<https://www.analyticsvidhya.com/blog/2018/12/practical-guide-object-detection-yolo-framework-python/>, December 2018.

- [8] TheCodingBug, “YOLOv4 Custom Object Detection Tutorial: Part 1 (Preparing Darknet YOLOv4 Custom Dataset)”, *YouTube*, <https://www.youtube.com/watch?v=sKDysNtnhJ4>, December 2020.
- [9] TheCodingBug, “YOLOv4 Custom Object Detection Tutorial: Part 2 (Training YOLOv4 Darknet on Custom Dataset)”, *YouTube*, <https://www.youtube.com/watch?v=-NEB5P-SLi0>, December 2020.
- [10] Raoul Fasel, “How to deploy YOLOv4 on UbiOps”, *UbiOps*, <https://ubiops.com/how-to-deploy-yolov4-on-ubiops/>, September 2020.