19UK1A05J9

RICE CROP DISEASE DETECTION USING YOLO

UG Phase-1 Project

Md Aliuddin Hyder

2022-23

VAAGDEVI ENGINEERING COLLEGE

TABLE OF CONTENTS:

1. PROJECT INTRO

Increasing grain production is essential to those areas where food is scarce. Increasing grain productionby controlling crop diseases in time should be effective. To construct a prediction model for plant diseases and to build a real-time application for crop diseases in the future, deep learning-based image detection architecture with a custom backbone was proposed for detecting plant diseases.

In order to get a good amount of crop, we need to detect the disease at the earliest.

Basically, crop disease diagnosis depends on different characteristics like color, shape, texture, etc. Here the person can capture the images of the crop and then the image will be sent to the trained YOLO model. The model analyzes the image and detects crop diseases like Bacterial Blight, Leaf Smut, and WhiteTip.

2. ARCHITECTURE

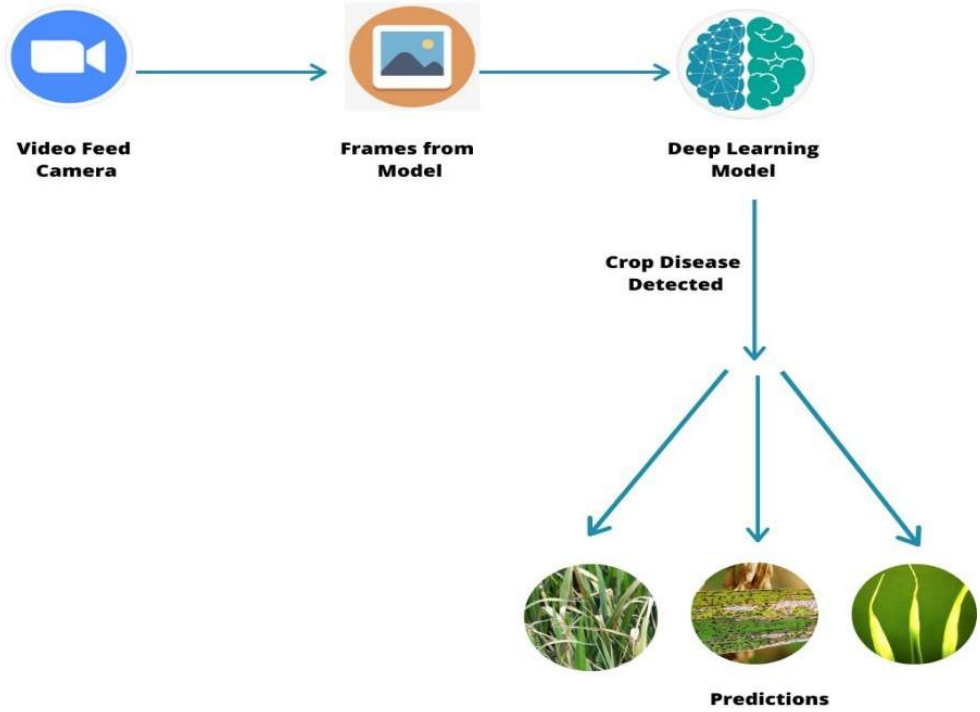


Fig. Project Architectural Diagram

3. PROJECT OBJECTIVES

By the end of this project you'll understand:

- YOLO-based Convolution Neural Network family of models for object detection and the mostrecent variation called YOLOv3.
- How to train a YOLO model in a windows environment.
- How to annotate images using Microsoft's Visual Object Tagging Tool (VoTT).

4. PRE-REQUISITES

To complete this project, you must require the following software's, concepts, and packages

- Python IDE (IDLE / Spyder / PyCharm)(Python 3.7)
- Microsoft's Visual Object Tagging Tool (VoTT)
- Python Packages need to be installed

Task 1: Anaconda Installation

To complete this project you should have the following software and packages

Anaconda Navigator:

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows,Linux, and macOS. Conda is an open-source, cross-platform, package management system. Anacondacomes with so very nice tools like JupyterLab, Jupyter Notebook,

QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupyter notebook and spyder

To install Anaconda navigator and to know how to use Jupyter Notebook a Spyder using Anaconda watch the video:

Link: https://youtu.be/5mDYijMfSzs

To build Deep learning models you must require the following packages:

Tensor flow: Tensor Flow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers can easily build and deploy ML-powered applications.

Keras: Keras leverages various optimization techniques to make high-level neural network API easierand more performant. It supports the following features:

- Consistent, simple, and extensible API.
- Minimal structure - easy to achieve the result without any frills.
- It supports multiple platforms and backends.
- It is a user-friendly framework that runs on both CPU and GPU.
- Highly scalability of computation.

To implement this project we will be usingPython 3.7 strictly.

Link: https://repo.anaconda.com/archive/Anaconda3-2019.07-Windows-x86_64.exe

Task 2: Install python packages

Follow the below steps to install packages :

- Open anaconda prompt as administrator.
- Type "pip install tensorflow==1.15.1" and click enter.
- Type "pip install keras=2.2.4" and click enter.
- Type "pip install opencv-python==4.1.0.25" and click enter.
- Type "pip install Pillow==6.2.2 and click enter.

the above steps allow you to install the packages in the anaconda environment

Task 3: Install Microsoft's Visual Object Tagging Tool (VoTT)

Head to VoTT [releases](https://github.com/Microsoft/VoTT/releases) and download and install theversion for your operating system. Under `Assets` select the package for your operating system:

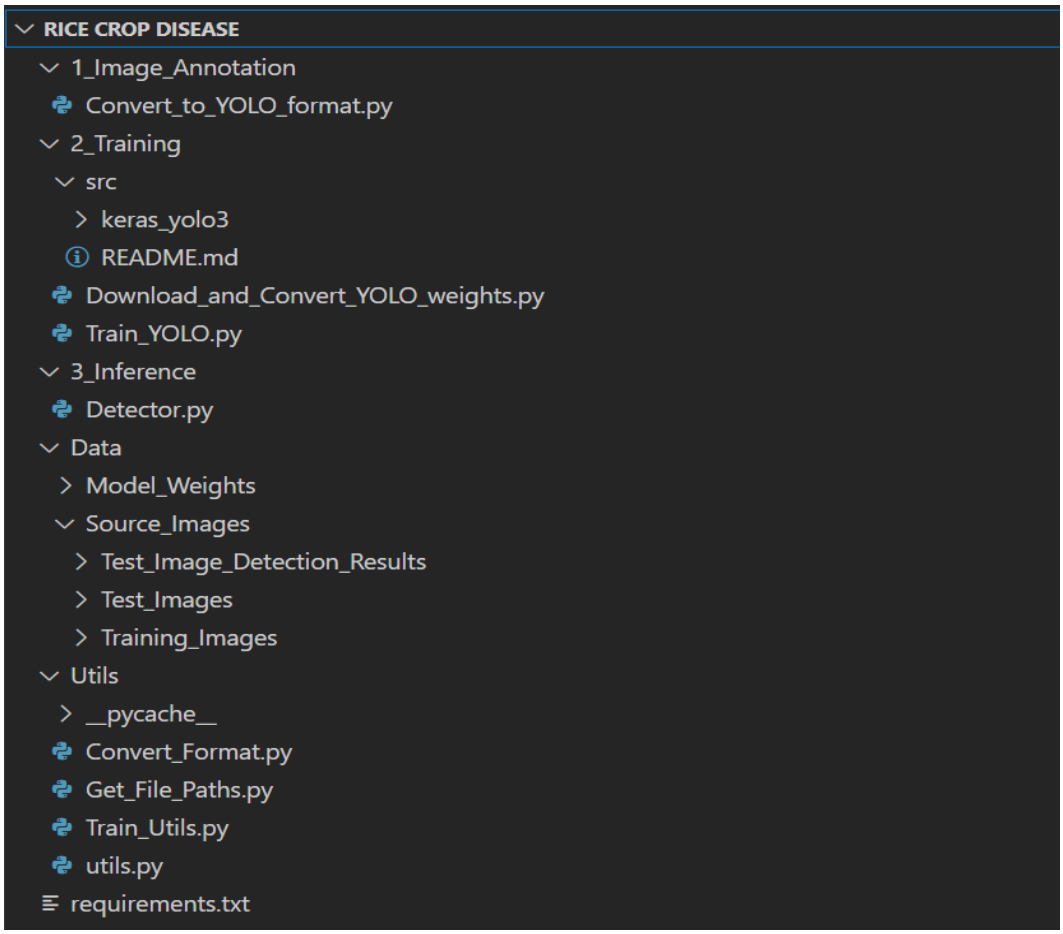
- 'vott-2.x.x-darwin.dmg' for Mac users,

- 'vott-2.x.x-win32.exe' for Windows users and

- 'vott-2.x.x-linux.snap' for Linux users.

Link: https://www.youtube.com/watch?v=h8NUIiOt1Hk

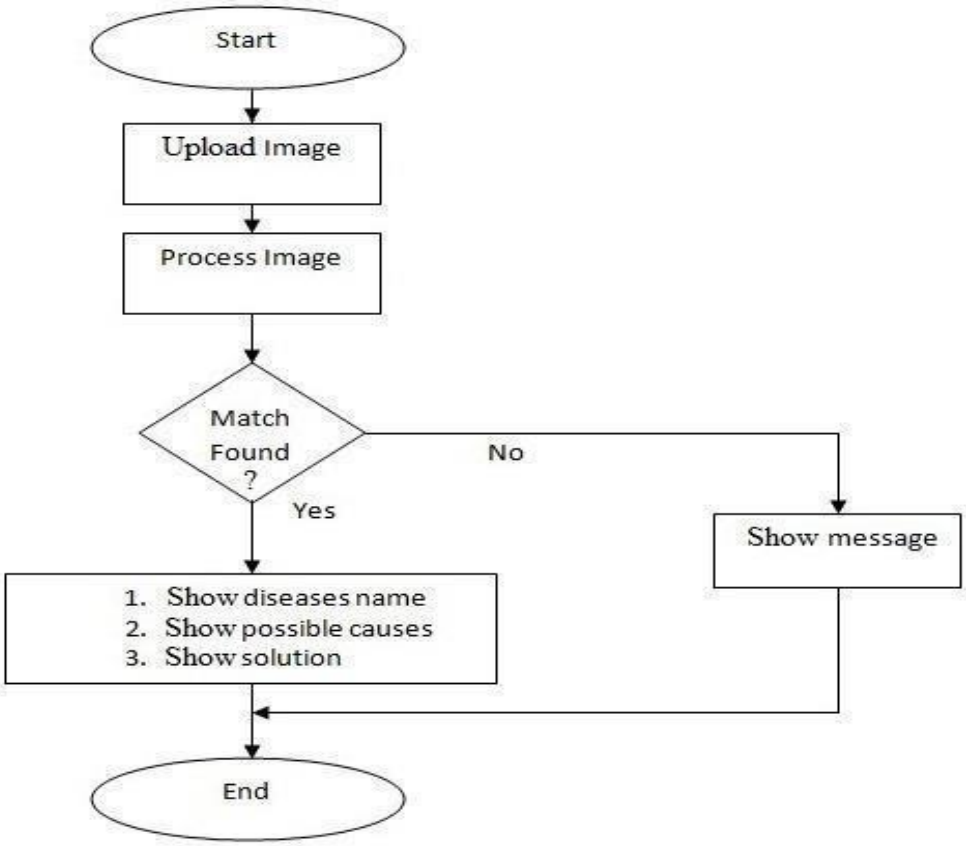
5. PROJECT STRUCTURE



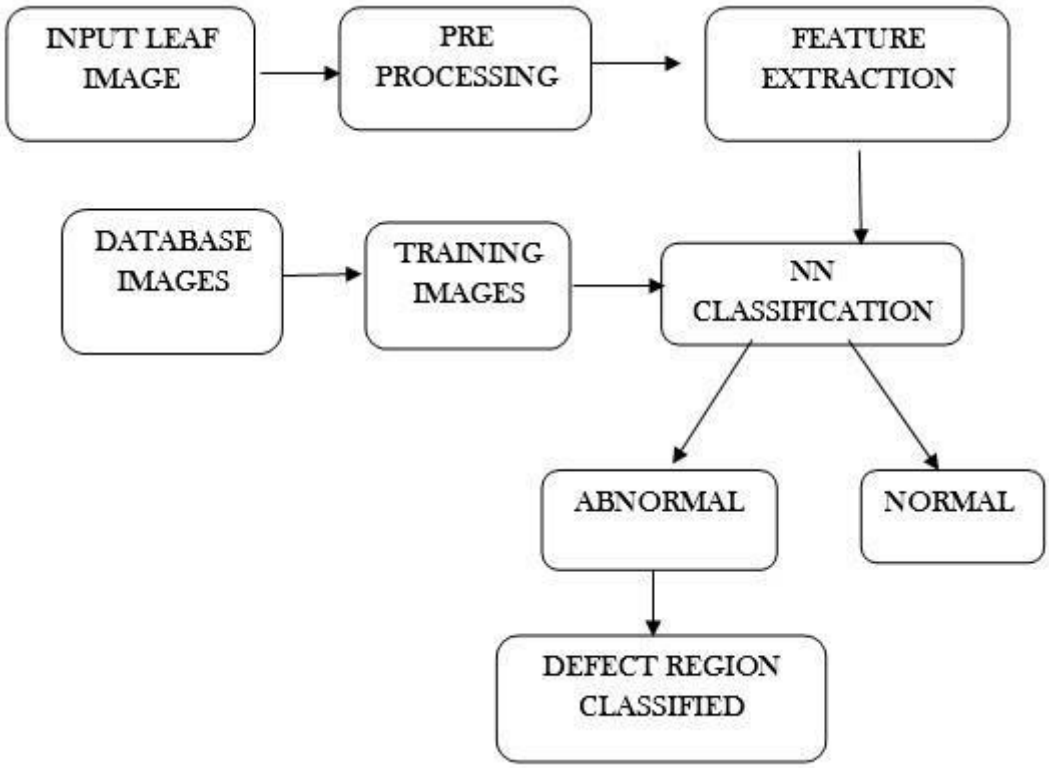
Note: These are generic YoLo code files that are not confined to particular project

6. DESIGN

i. FLOWCHART



ii. Block Diagram



7. CREATE DATASET

Now collect the images of Diseased Plants from Google or click here to download the dataset

Link: https://www.kaggle.com/minhhuy2810/rice-diseases-image-dataset

- ## Create Dataset from Scratch

To train the YOLO object detector on your own dataset, copy your training images to[`Rice crop disease/Data/Source_Images/Training_Images`]
Collect at least 50 images of Bacterial Blight, 50 images of Leaf Smut, 50 images of White Tip diseaseimages, and place them in this folder.

Tip: If you do not already have an image dataset, consider using a Chrome extension such as [FatkunBatchDownloader]

(https://chrome.google.com/webstore/detail/fatkun-batch-download-ima/nnjjahlikiabnchcpehcpkdeckfgnohf?hl=en) which lets you search and download images from Google Images.

8. ANNOTATE IMAGES

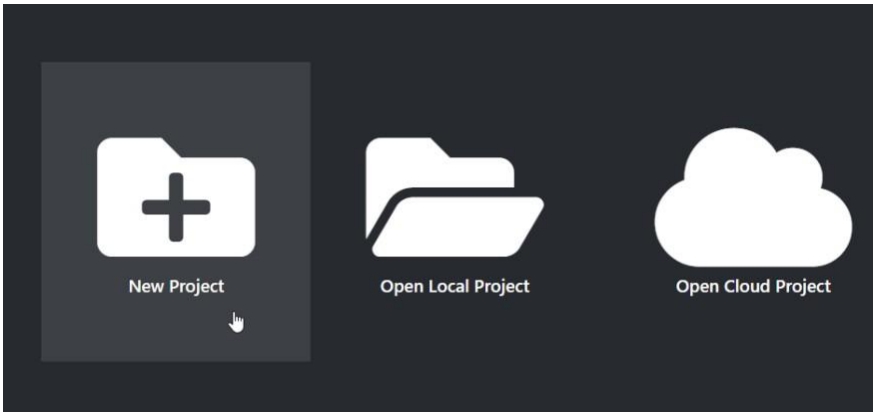
To make our detector learn, we first need to feed it some good training examples. We use Microsoft'sVisual Object Tagging Tool (VoTT) to manually label images in our training folder [`Rice crop disease/Data/Source_Images/Training_Images].

To achieve decent results annotate at least 100 images for each category. For good results label at least300 images and for great results label 1000+ images for each category.

Tip: If you increase no of images processing time increases

Annotating Images using VOTT:

TASK -1: CREATE NEW PROJECT



Step1:

Create a '**New Project**' and call it `**Annotations**`. It is highly recommended to use `**Annotations**` as your project name. If you like to use a different name for your project, you will have to modify the command line arguments of subsequent scripts accordingly.

Step2:

Under '**Source Connection**' choose '**Add Connection**' and put "**Images**" as '**Display Name**'. Under '**Provider**' choose '**Local File System**' and select
[`Rice crop disease/Data/Source_Images/Training_Images`]) And then '**Save Connection**'.
For 'Target Connection' choose the same folder as for 'Source Connection'. And Tags of diseases. Hit '**Save Project**' to finish project creation.

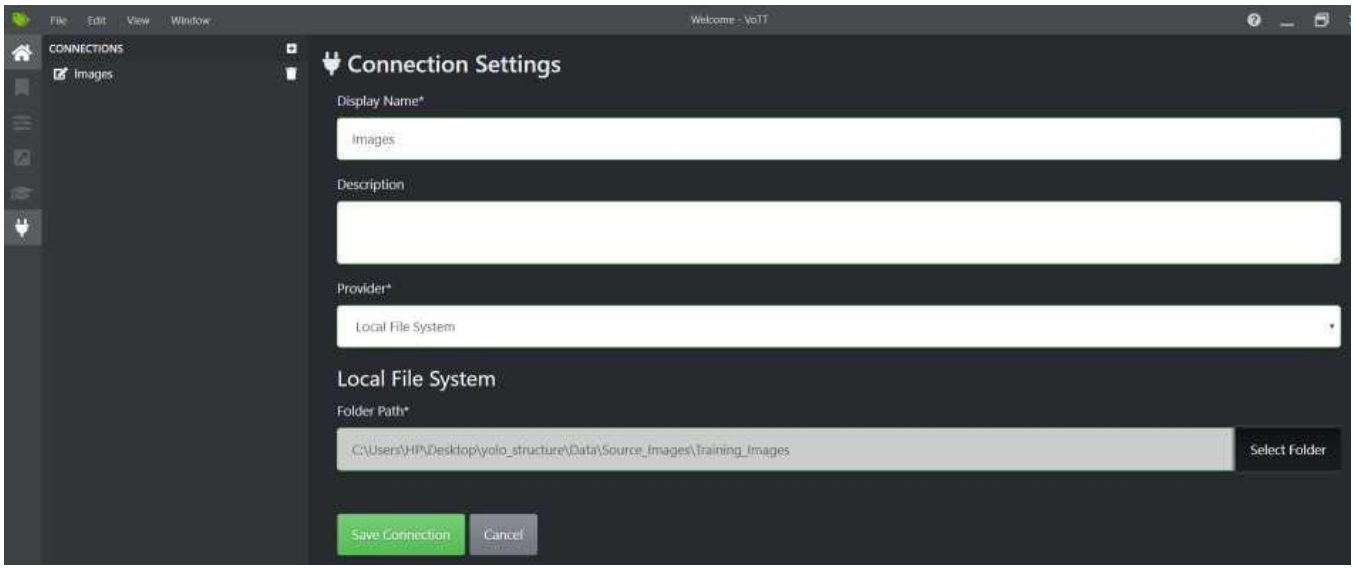


Fig. Connection Settings in VOTT

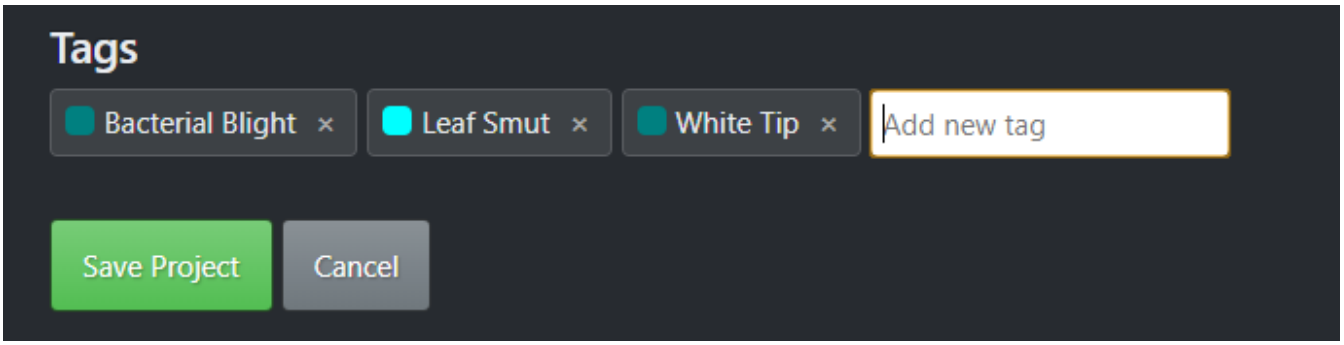


Fig. Add Tags under in VOTT

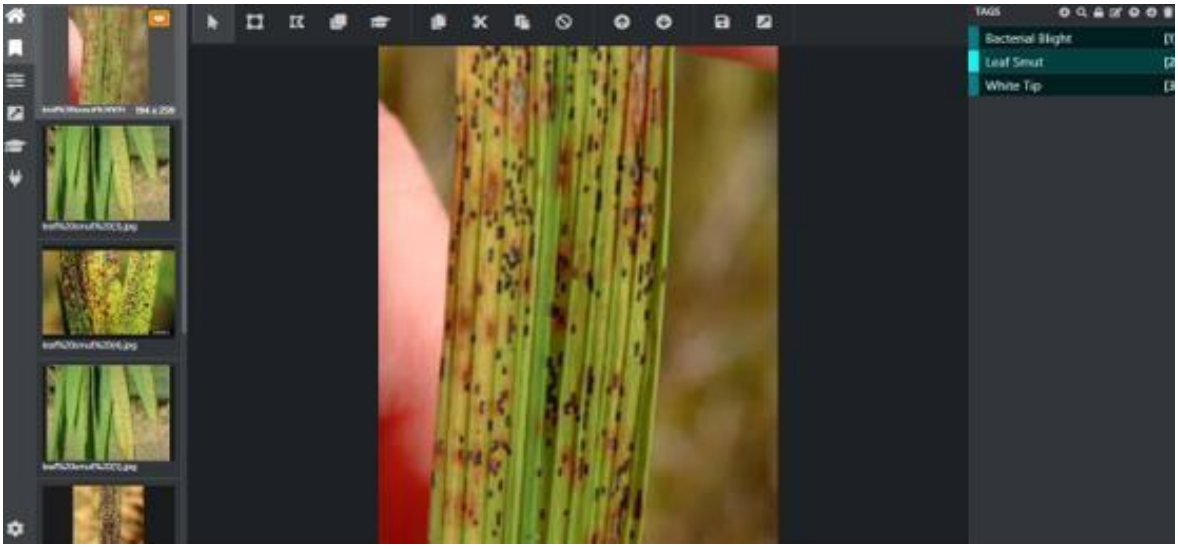


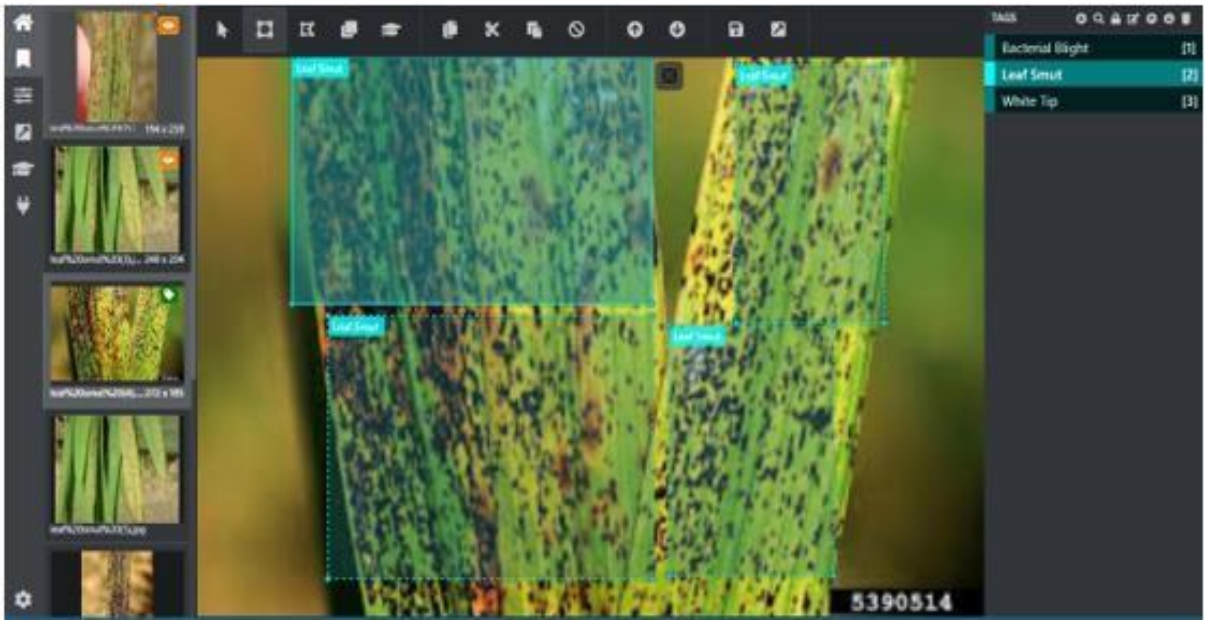
Fig. Annotating Images

Step3:

Navigate to '**Export Settings**' in the sidebar and then change the '**Provider**' to '**CommaSeparated Values (CSV)',** then hit '**Save Export Settings**'.

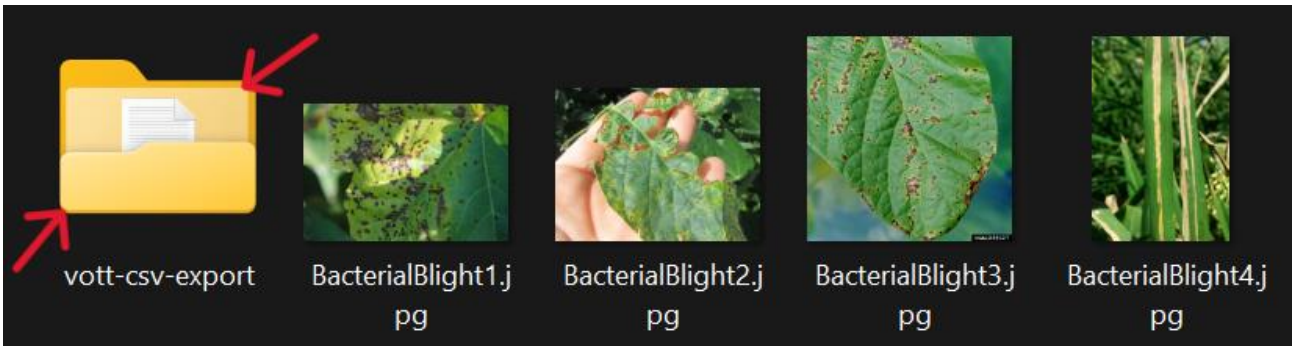
Step4:

First, create a new tag on the right and give it a relevant tag name. In our example, we choose `**BacterialBlight, Leaf Smut, White Tip**`. Then draw bounding boxes around your objects for respective diseases.



Step5:

Once you have labeled enough images press '**CRTL+E**' to export the project. You should now see a folder called [`vott-csv-export`](/Data/Source_Images/Training_Images/vott-csv-export) directory. Within that folder, you should see a `.csv` file called [`Annotations-export.csv`](/Data/Source_Images/Training_Images/vott-csv-export/Annotations export.csv) which contains file names and bounding box coordinates.



Step 6:

As a final step, convert the VoTT CSV format to the YOLOv3 format. To do so, run the conversion scriptfrom within the [`Rice crop disease/1_Image_Annotation`] folder.

TASK-2: CONVERT TO YOLO

(Here '**Rice1**'is the Environment)
To run the file open the anaconda prompt navigate to Rice crop disease /1_Image_Annotation and run

Convert_to_YOLO_format.py

The script generates two output files: [`data_train.txt`](/Data/Source_Images/Training_Images/vott-csv-export/data_train.txt) located in the

[`Rice crop disease \Data\Source_Images\Training_Images\vott-csv-export`]

(/Data/Source_Images/Training_Images/vott-csv-export) folder and[`data_classes.txt`]

(\Data\Model_Weights**data_classes.txt**) located in the

[`Rice crop disease\Data\Model_Weights`](\Data\Model_Weights\) folder.

To list available command line options run `python Convert_to_YOLO_format.py -h`.

Link : https://youtu.be/uDWgWJ5Gpwc

TASK-3: TRAINING YOLO

In this milestone we will train our model using YOLO weights.

Task-1

Download and Convert Pre-Trained Weights

Step1:

Using the training images located in[`Rice crop disease \Data\Source_Images\Training_Images`] (\Data\Source_Images\Training_Images)and the annotation file [`data_train.txt`](\Data\Source_Images\Training_Images\vott-csv-export) which wehave created in the [previous step](/1_Image_Annotation/) we are now ready to train our YOLOv3detector.

Step2:

Before getting started download the pre-trained YOLOv3 weights and convert them to the Keras format.To run both steps run the download and conversion script from within the

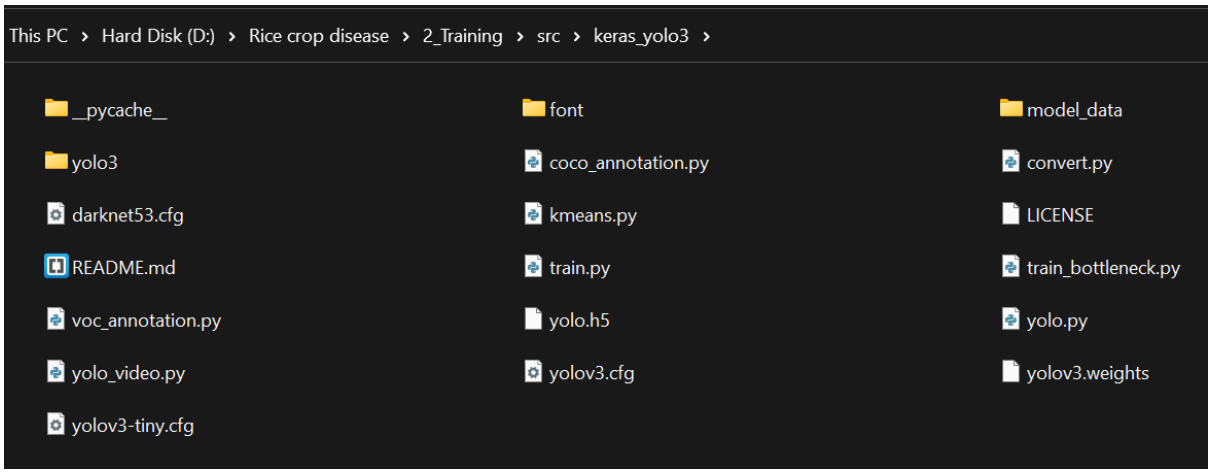
[`Rice crop disease \2_Training`](\2_Training/) directory

Run python Download_and_Convert_YOLO_weights.py

```
(rice1) D:\Rice crop disease>cd 2_Training

(rice1) D:\Rice crop disease\2_Training>python Download_and_Convert_YOLO_weights.py
```

- This will download yolov3 weights and convert them into keras format. These files will bedownloaded in[Rice crop disease \2_Training\src\keras_yolo3].
- Tip: This step would take some time as it needs to download the weights and convert intokeras format.



- The weights are pre-trained on the [ImageNet 1000 dataset](http://image-net.org/challenges/LSVRC/2015/index) and thus work well for object detection tasks that arevery similar to the types of images and objects in the ImageNet 1000 dataset.

>>>> **RUNNING TRAIN FILE**

To start the training, run the training script from within the [`Rice crop disease \2_Training`](\2_Training\)directory

Run python Train_YOLO.py

```
(rice1) D:\Rice crop disease>cd 2_Training

(rice1) D:\Rice crop disease\2_Training>python Train_YOLO.py
```

Depending on your set-up, this process can take a few minutes to a few hours. The final weights are saved in [`Rice crop disease/Data/Model_weights`](/Data/Model_weights). To list available command lineoptions run `python Train_YOLO.py -h`.

TASK-4: TESTING THE MODEL

In this step, we test our detector on infected crop images located in [`yolostructure/Data/Source_Images/Test_Images`](/Data/Source_Images/Test_Images). If you like to test the detector on your own images or videos, place them in the [`Test_Images`](/Data/Source_Images/Test_Images) folder.

Run python Detector.py

```
(rice1) D:\Rice crop disease>cd 3_Inference

(rice1) D:\Rice crop disease\3_Inference>python Detector.py
```

9. OUTPUT

The result of test images is stored in the test images folder.

D:\Rice crop disease\Data\Source_Images\Test_Image_Detection_Result

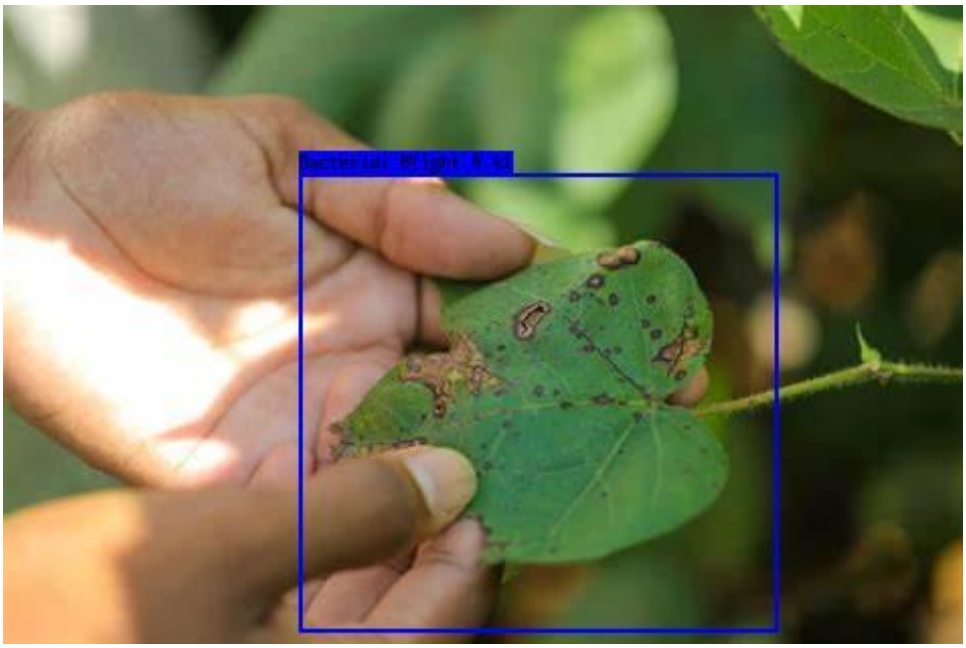


Fig. Bacterial Blight

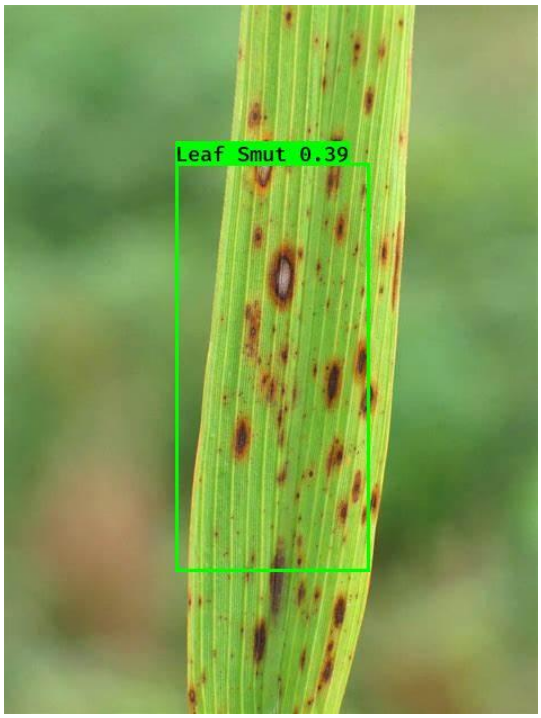


Fig. Leaf Smut

Fig. White Tip

10. CONCLUSION

YOLOv3:

- One-shot object detection
- Move the prediction from cell to box to enable the detection of several objects in one cells
- Use logistic activation for box prediction to constrain the range
- Use logistic classification to enable multi-labeling
- Use feature pyramid to reinforce the small object detection

11. APPENDIX

Convert_to_YOLO_format.py

```
from PIL import Image
from os import path, makedirs
import os
import re
import pandas as pd
import sys
import argparse


def get_parent_dir(n=1):
    """ returns the n-th parent dicrectory of the current
    working directory """
    current_path = os.path.dirname(os.path.abspath(__file__))
    for k in range(n):
        current_path = os.path.dirname(current_path)
    return current_path


sys.path.append(os.path.join(get_parent_dir(1), "Utils"))
from Convert_Format import convert_vott_csv_to_yolo

Data_Folder = os.path.join(get_parent_dir(1), "Data")
VoTT_Folder = os.path.join(
    Data_Folder, "Source_Images", "Training_Images", "vott-csv-export"
)
VoTT_csv = os.path.join(VoTT_Folder, "Annotations-export.csv")
YOLO_filename = os.path.join(VoTT_Folder, "data_train.txt")

model_folder = os.path.join(Data_Folder, "Model_Weights")
classes_filename = os.path.join(model_folder, "data_classes.txt")

if __name__ == "__main__":
    # surpress any inhereted default values
    parser = argparse.ArgumentParser(argument_default=argparse.SUPPRESS)
    """
    Command line options
    """
    parser.add_argument(
        "--VoTT_Folder",
        type=str,
        default=VoTT_Folder,
        help="Absolute path to the exported files from the image tagging step with VoTT. Default is
"
        + VoTT_Folder,
    )

    parser.add_argument(
        "--VoTT_csv",
        type=str,
        default=VoTT_csv,
        help="Absolute path to the *.csv file exported from VoTT. Default is "
        + VoTT_csv,
    )
```

```
    parser.add_argument(
        "--YOLO_filename",
        type=str,
        default=YOLO_filename,
        help="Absolute path to the file where the annotations in YOLO format should be saved.
Default is "
        + YOLO_filename,
    )

    FLAGS = parser.parse_args()

    # Prepare the dataset for YOLO
    multi_df = pd.read_csv(FLAGS.VoTT_csv)
    labels = multi_df["label"].unique()
    labeldict = dict(zip(labels, range(len(labels))))
    multi_df.drop_duplicates(subset=None, keep="first", inplace=True)
    train_path = FLAGS.VoTT_Folder
    convert_vott_csv_to_yolo(
        multi_df, labeldict, path=train_path, target_name=FLAGS.YOLO_filename
    )

    # Make classes file
    file = open(classes_filename, "w")

    # Sort Dict by Values
    SortedLabelDict = sorted(labeldict.items(), key=lambda x: x[1])
    for elem in SortedLabelDict:
        file.write(elem[0] + "\n")
    file.close()
```

Download_and_Convert_YOLO_weights.py

```
import os
import subprocess
import time
import sys
import argparse
import requests
import progressbar

FLAGS = None

root_folder = os.path.dirname(os.path.abspath(__file__))
download_folder = os.path.join(root_folder, "src", "keras_yolo3")

if __name__ == "__main__":
    # Delete all default flags
    parser = argparse.ArgumentParser(argument_default=argparse.SUPPRESS)
    """
    Command line options
    """
    parser.add_argument(
        "--download_folder",
        type=str,
        default=download_folder,
        help="Folder to download weights to. Default is " + download_folder,
    )
```

```
    FLAGS = parser.parse_args()

    url = "https://pjreddie.com/media/files/yolov3.weights"
    r = requests.get(url, stream=True)

    f = open(os.path.join(download_folder, "yolov3.weights"), "wb")
    file_size = int(r.headers.get("content-length"))
    chunk = 100
    num_bars = file_size // chunk
    bar = progressbar.ProgressBar(maxval=num_bars).start()
    i = 0
    for chunk in r.iter_content(chunk):
        f.write(chunk)
        bar.update(i)
        i += 1
    f.close()

    call_string = "python convert.py yolov3.cfg yolov3.weights yolo.h5"

    subprocess.call(call_string, shell=True, cwd=download_folder)
```

Train_YOLO.py

```
"""
MODIFIED FROM keras-yolo3 PACKAGE, https://github.com/qqwweee/keras-yolo3
Retrain the YOLO model for your own dataset.
"""

import os
import sys
import argparse
import warnings

def get_parent_dir(n=1):
    """ returns the n-th parent dicrectory of the current
    working directory """
    current_path = os.path.dirname(os.path.abspath(__file__))
    for k in range(n):
        current_path = os.path.dirname(current_path)
    return current_path


src_path = os.path.join(get_parent_dir(0), "src")
sys.path.append(src_path)

utils_path = os.path.join(get_parent_dir(1), "Utils")
sys.path.append(utils_path)

import numpy as np
import keras.backend as K
from keras.layers import Input, Lambda
from keras.models import Model
from keras.optimizers import Adam
from keras.callbacks import (
    TensorBoard,
    ModelCheckpoint,
```

```
    ReduceLROnPlateau,
    EarlyStopping,
)
from keras_yolo3.yolo3.model import (
    preprocess_true_boxes,
    yolo_body,
    tiny_yolo_body,
    yolo_loss,
)
from keras_yolo3.yolo3.utils import get_random_data
from PIL import Image
from time import time
import tensorflow.compat.v1 as tf
import pickle

from Train_Utils import (
    get_classes,
    get_anchors,
    create_model,
    create_tiny_model,
    data_generator,
    data_generator_wrapper,
    ChangeToOtherMachine,
)


keras_path = os.path.join(src_path, "keras_yolo3")
Data_Folder = os.path.join(get_parent_dir(1), "Data")
Image_Folder = os.path.join(Data_Folder, "Source_Images", "Training_Images")
VoTT_Folder = os.path.join(Image_Folder, "vott-csv-export")
YOLO_filename = os.path.join(VoTT_Folder, "data_train.txt")

Model_Folder = os.path.join(Data_Folder, "Model_Weights")
YOLO_classname = os.path.join(Model_Folder, "data_classes.txt")

log_dir = Model_Folder
anchors_path = os.path.join(keras_path, "model_data", "yolo_anchors.txt")
weights_path = os.path.join(keras_path, "yolo.h5")

FLAGS = None

if __name__ == "__main__":
    # Delete all default flags
    parser = argparse.ArgumentParser(argument_default=argparse.SUPPRESS)
    """
    Command line options
    """

    parser.add_argument(
        "--annotation_file",
        type=str,
        default=YOLO_filename,
        help="Path to annotation file for Yolo. Default is " + YOLO_filename,
    )
    parser.add_argument(
        "--classes_file",
        type=str,
        default=YOLO_classname,
        help="Path to YOLO classnames. Default is " + YOLO_classname,
```

```
    )

    parser.add_argument(
        "--log_dir",
        type=str,
        default=log_dir,
        help="Folder to save training logs and trained weights to. Default is "
        + log_dir,
    )

    parser.add_argument(
        "--anchors_path",
        type=str,
        default=anchors_path,
        help="Path to YOLO anchors. Default is " + anchors_path,
    )

    parser.add_argument(
        "--weights_path",
        type=str,
        default=weights_path,
        help="Path to pre-trained YOLO weights. Default is " + weights_path,
    )
    parser.add_argument(
        "--val_split",
        type=float,
        default=0.1,
        help="Percentage of training set to be used for validation. Default is 10%.",
    )
    parser.add_argument(
        "--is_tiny",
        default=False,
        action="store_true",
        help="Use the tiny Yolo version for better performance and less accuracy. Default is
False.",
    )
    parser.add_argument(
        "--random_seed",
        type=float,
        default=None,
        help="Random seed value to make script deterministic. Default is 'None', i.e. non-
deterministic.",
    )
    parser.add_argument(
        "--epochs",
        type=float,
        default=51,
        help="Number of epochs for training last layers and number of epochs for fine-tuning
layers. Default is 51.",
    )
    parser.add_argument(
        "--warnings",
        default=False,
        action="store_true",
        help="Display warning messages. Default is False.",
    )

    FLAGS = parser.parse_args()
```

```
    if not FLAGS.warnings:
        tf.logging.set_verbosity(tf.logging.ERROR)
        os.environ['TF_CPP_MIN_LOG_LEVEL']='3'
        warnings.filterwarnings("ignore")

    np.random.seed(FLAGS.random_seed)

    log_dir = FLAGS.log_dir

    class_names = get_classes(FLAGS.classes_file)
    num_classes = len(class_names)
    anchors = get_anchors(FLAGS.anchors_path)
    weights_path = FLAGS.weights_path

    input_shape = (416, 416)  # multiple of 32, height, width
    epoch1, epoch2 = FLAGS.epochs, FLAGS.epochs

    is_tiny_version = len(anchors) == 6  # default setting
    if FLAGS.is_tiny:
        model = create_tiny_model(
            input_shape, anchors, num_classes, freeze_body=2, weights_path=weights_path
        )
    else:
        model = create_model(
            input_shape, anchors, num_classes, freeze_body=2, weights_path=weights_path
        )  # make sure you know what you freeze

    log_dir_time = os.path.join(log_dir, "{}".format(int(time())))
    logging = TensorBoard(log_dir=log_dir_time)
    checkpoint = ModelCheckpoint(
        os.path.join(log_dir, "checkpoint.h5"),
        monitor="val_loss",
        save_weights_only=True,
        save_best_only=True,
        period=5,
    )
    reduce_lr = ReduceLROnPlateau(monitor="val_loss", factor=0.1, patience=3, verbose=1)
    early_stopping = EarlyStopping(
        monitor="val_loss", min_delta=0, patience=10, verbose=1
    )

    val_split = FLAGS.val_split
    with open(FLAGS.annotation_file) as f:
        lines = f.readlines()

    # This step makes sure that the path names correspond to the local machine
    # This is important if annotation and training are done on different machines (e.g. training on
AWS)
    lines = ChangeToOtherMachine(lines, remote_machine="")
    np.random.shuffle(lines)
    num_val = int(len(lines) * val_split)
    num_train = len(lines) - num_val

    # Train with frozen layers first, to get a stable loss.
    # Adjust num epochs to your dataset. This step is enough to obtain a decent model.
    if True:
        model.compile(
            optimizer=Adam(lr=1e-3),
            loss={
```

```
            # use custom yolo_loss Lambda layer.
            "yolo_loss": lambda y_true, y_pred: y_pred
        },
    )

    batch_size = 32
    print(
        "Train on {} samples, val on {} samples, with batch size {}.".format(
            num_train, num_val, batch_size
        )
    )
    history = model.fit_generator(
        data_generator_wrapper(
            lines[:num_train], batch_size, input_shape, anchors, num_classes
        ),
        steps_per_epoch=max(1, num_train // batch_size),
        validation_data=data_generator_wrapper(
            lines[num_train:], batch_size, input_shape, anchors, num_classes
        ),
        validation_steps=max(1, num_val // batch_size),
        epochs=epoch1,
        initial_epoch=0,
        callbacks=[logging, checkpoint],
    )
    model.save_weights(os.path.join(log_dir, "trained_weights_stage_1.h5"))

    step1_train_loss = history.history["loss"]

    file = open(os.path.join(log_dir_time, "step1_loss.npy"), "w")
    with open(os.path.join(log_dir_time, "step1_loss.npy"), "w") as f:
        for item in step1_train_loss:
            f.write("%s\n" % item)
    file.close()

    step1_val_loss = np.array(history.history["val_loss"])

    file = open(os.path.join(log_dir_time, "step1_val_loss.npy"), "w")
    with open(os.path.join(log_dir_time, "step1_val_loss.npy"), "w") as f:
        for item in step1_val_loss:
            f.write("%s\n" % item)
    file.close()

# Unfreeze and continue training, to fine-tune.
# Train longer if the result is unsatisfactory.
if True:
    for i in range(len(model.layers)):
        model.layers[i].trainable = True
    model.compile(
        optimizer=Adam(lr=1e-4), loss={"yolo_loss": lambda y_true, y_pred: y_pred}
    )  # recompile to apply the change
    print("Unfreeze all layers.")

    batch_size = (
        4  # note that more GPU memory is required after unfreezing the body
    )
    print(
        "Train on {} samples, val on {} samples, with batch size {}.".format(
            num_train, num_val, batch_size
        )
```

```
        )
        history = model.fit_generator(
            data_generator_wrapper(
                lines[:num_train], batch_size, input_shape, anchors, num_classes
            ),
            steps_per_epoch=max(1, num_train // batch_size),
            validation_data=data_generator_wrapper(
                lines[num_train:], batch_size, input_shape, anchors, num_classes
            ),
            validation_steps=max(1, num_val // batch_size),
            epochs=epoch1 + epoch2,
            initial_epoch=epoch1,
            callbacks=[logging, checkpoint, reduce_lr, early_stopping],
        )
        model.save_weights(os.path.join(log_dir, "trained_weights_final.h5"))
        step2_train_loss = history.history["loss"]

        file = open(os.path.join(log_dir_time, "step2_loss.npy"), "w")
        with open(os.path.join(log_dir_time, "step2_loss.npy"), "w") as f:
            for item in step2_train_loss:
                f.write("%s\n" % item)
        file.close()

        step2_val_loss = np.array(history.history["val_loss"])

        file = open(os.path.join(log_dir_time, "step2_val_loss.npy"), "w")
        with open(os.path.join(log_dir_time, "step2_val_loss.npy"), "w") as f:
            for item in step2_val_loss:
                f.write("%s\n" % item)
        file.close()
```

Detector.py

```
import os
import sys


def get_parent_dir(n=1):
    """ returns the n-th parent dicrectory of the current
    working directory """
    current_path = os.path.dirname(os.path.abspath(__file__))
    for k in range(n):
        current_path = os.path.dirname(current_path)
    return current_path


src_path = os.path.join(get_parent_dir(1), "2_Training", "src")
utils_path = os.path.join(get_parent_dir(1), "Utils")

sys.path.append(src_path)
sys.path.append(utils_path)

import argparse
```

```
from keras_yolo3.yolo import YOLO, detect_video
from PIL import Image
from timeit import default_timer as timer
from utils import load_extractor_model, load_features, parse_input, detect_object
import test
import utils
import pandas as pd
import numpy as np
from Get_File_Paths import GetFileList
import random

os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

# Set up folder names for default values
data_folder = os.path.join(get_parent_dir(n=1), "Data")

image_folder = os.path.join(data_folder, "Source_Images")

image_test_folder = os.path.join(image_folder, "Test_Images")

detection_results_folder = os.path.join(image_folder, "Test_Image_Detection_Results")
detection_results_file = os.path.join(detection_results_folder, "Detection_Results.csv")

model_folder = os.path.join(data_folder, "Model_Weights")

model_weights = os.path.join(model_folder, "trained_weights_final.h5")
model_classes = os.path.join(model_folder, "data_classes.txt")

anchors_path = os.path.join(src_path, "keras_yolo3", "model_data", "yolo_anchors.txt")

FLAGS = None

if __name__ == "__main__":
    # Delete all default flags
    parser = argparse.ArgumentParser(argument_default=argparse.SUPPRESS)
    """
    Command line options
    """

    parser.add_argument(
        "--input_path",
        type=str,
        default=image_test_folder,
        help="Path to image/video directory. All subdirectories will be included. Default is "
        + image_test_folder,
    )

    parser.add_argument(
        "--output",
        type=str,
        default=detection_results_folder,
        help="Output path for detection results. Default is "
        + detection_results_folder,
    )

    parser.add_argument(
        "--no_save_img",
        default=False,
        action="store_true",
```

```
        help="Only save bounding box coordinates but do not save output images with annotated
boxes. Default is False.",
    )

    parser.add_argument(
        "--file_types",
        "--names-list",
        nargs="*",
        default=[],
        help="Specify list of file types to include. Default is --file_types .jpg .jpeg .png .mp4",
    )

    parser.add_argument(
        "--yolo_model",
        type=str,
        dest="model_path",
        default=model_weights,
        help="Path to pre-trained weight files. Default is " + model_weights,
    )

    parser.add_argument(
        "--anchors",
        type=str,
        dest="anchors_path",
        default=anchors_path,
        help="Path to YOLO anchors. Default is " + anchors_path,
    )

    parser.add_argument(
        "--classes",
        type=str,
        dest="classes_path",
        default=model_classes,
        help="Path to YOLO class specifications. Default is " + model_classes,
    )

    parser.add_argument(
        "--gpu_num", type=int, default=1, help="Number of GPU to use. Default is 1"
    )

    parser.add_argument(
        "--confidence",
        type=float,
        dest="score",
        default=0.25,
        help="Threshold for YOLO object confidence score to show predictions. Default is 0.25.",
    )

    parser.add_argument(
        "--box_file",
        type=str,
        dest="box",
        default=detection_results_file,
        help="File to save bounding box results to. Default is "
        + detection_results_file,
    )

    parser.add_argument(
        "--postfix",
```

```
        type=str,
        dest="postfix",
        default="_disease",
        help='Specify the postfix for images with bounding boxes. Default is "_disease"',
    )

    FLAGS = parser.parse_args()

    save_img = not FLAGS.no_save_img

    file_types = FLAGS.file_types

    if file_types:
        input_paths = GetFileList(FLAGS.input_path, endings=file_types)
    else:
        input_paths = GetFileList(FLAGS.input_path)

    # Split images and videos
    img_endings = (".jpg", ".jpeg", ".png")
    vid_endings = (".mp4", ".mpeg", ".mpg", ".avi")

    input_image_paths = []
    input_video_paths = []
    for item in input_paths:
        if item.endswith(img_endings):
            input_image_paths.append(item)
        elif item.endswith(vid_endings):
            input_video_paths.append(item)

    output_path = FLAGS.output
    if not os.path.exists(output_path):
        os.makedirs(output_path)

    # define YOLO detector
    yolo = YOLO(
        **{
            "model_path": FLAGS.model_path,
            "anchors_path": FLAGS.anchors_path,
            "classes_path": FLAGS.classes_path,
            "score": FLAGS.score,
            "gpu_num": FLAGS.gpu_num,
            "model_image_size": (416, 416),
        }
    )

    # Make a dataframe for the prediction outputs
    out_df = pd.DataFrame(
        columns=[
            "image",
            "image_path",
            "xmin",
            "ymin",
            "xmax",
            "ymax",
            "label",
            "confidence",
            "x_size",
            "y_size",
        ]
```

```
    )

    # labels to draw on images
    class_file = open(FLAGS.classes_path, "r")
    input_labels = [line.rstrip("\n") for line in class_file.readlines()]
    print("Found {} input labels: {} ...".format(len(input_labels), input_labels))

    if input_image_paths:
        print(
            "Found {} input images: {} ...".format(
                len(input_image_paths),
                [os.path.basename(f) for f in input_image_paths[:5]],
            )
        )
        start = timer()
        text_out = ""

        # This is for images
        for i, img_path in enumerate(input_image_paths):
            print(img_path)
            prediction, image,lat,lon= detect_object(
                yolo,
                img_path,
                save_img=save_img,
                save_img_path=FLAGS.output,
                postfix=FLAGS.postfix,
            )
            print(lat,lon)
            y_size, x_size, _ = np.array(image).shape
            for single_prediction in prediction:
                out_df = out_df.append(
                    pd.DataFrame(
                        [
                            [
                                os.path.basename(img_path.rstrip("\n")),
                                img_path.rstrip("\n"),
                            ]
                            + single_prediction
                            + [x_size, y_size]
                        ],
                        columns=[
                            "image",
                            "image_path",
                            "xmin",
                            "ymin",
                            "xmax",
                            "ymax",
                            "label",
                            "confidence",
                            "x_size",
                            "y_size",
                        ],
                    )
                )
        end = timer()
        print(
            "Processed {} images in {:.1f}sec - {:.1f}FPS".format(
                len(input_image_paths),
                end - start,
```

```
                len(input_image_paths) / (end - start),
            )
        )
        out_df.to_csv(FLAGS.box, index=False)

    # This is for videos
    if input_video_paths:
        print(
            "Found {} input videos: {} ...".format(
                len(input_video_paths),
                [os.path.basename(f) for f in input_video_paths[:5]],
            )
        )
        start = timer()
        for i, vid_path in enumerate(input_video_paths):
            output_path = os.path.join(
                FLAGS.output,
                os.path.basename(vid_path).replace(".", FLAGS.postfix + "."),
            )
            detect_video(yolo, vid_path, output_path=output_path)

        end = timer()
        print(
            "Processed {} videos in {:.1f}sec".format(
                len(input_video_paths), end - start
            )
        )
    # Close the current yolo session
    yolo.close_session()
```