Project Documentation

Aim: To detect Exudate in fundus image using machine learning

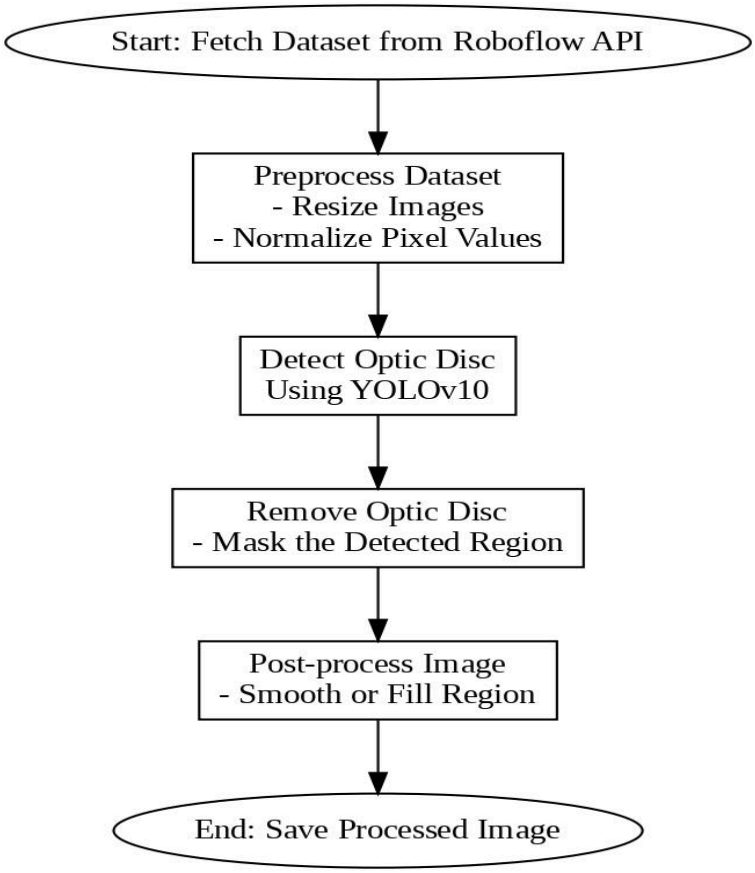
Brief of modules:

Module 1:

Aim : Detect and remove optic disc from fundus image

I. Algorithms can be used YOLOv10 and detectron2 to detect optic disc
Ii. Removing optic disc

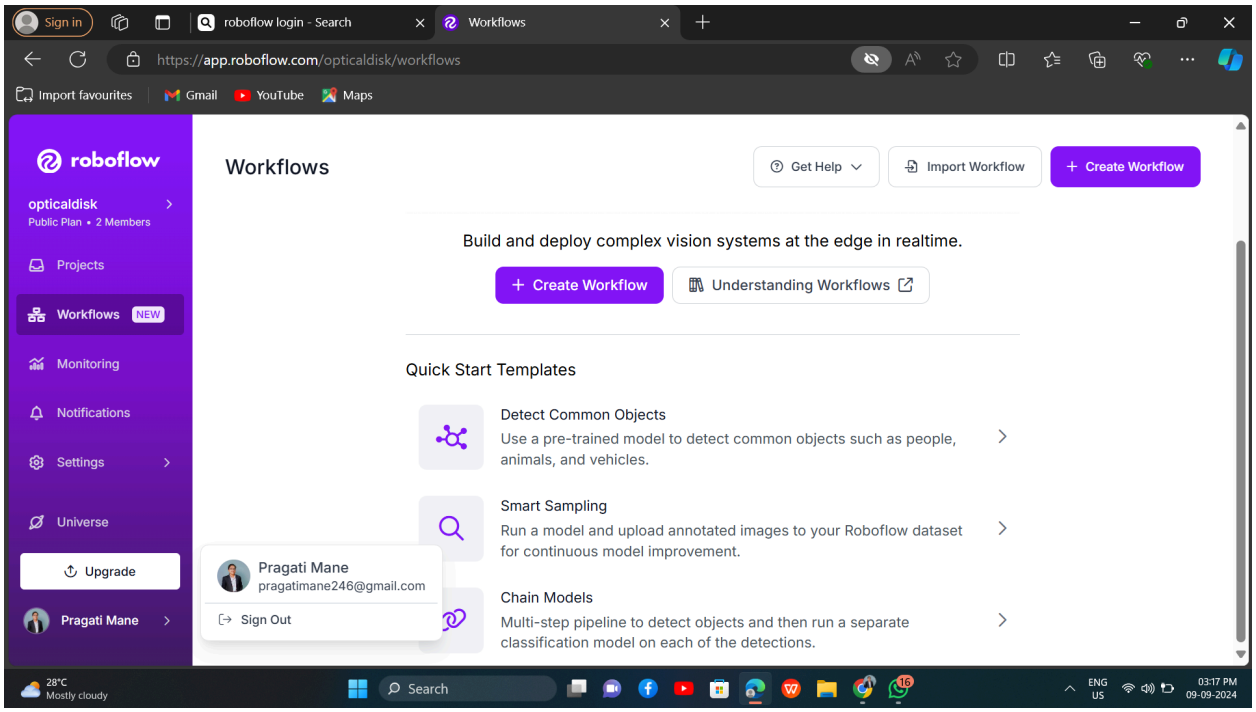
Flowchart:



Roboflow Labeling Dataset

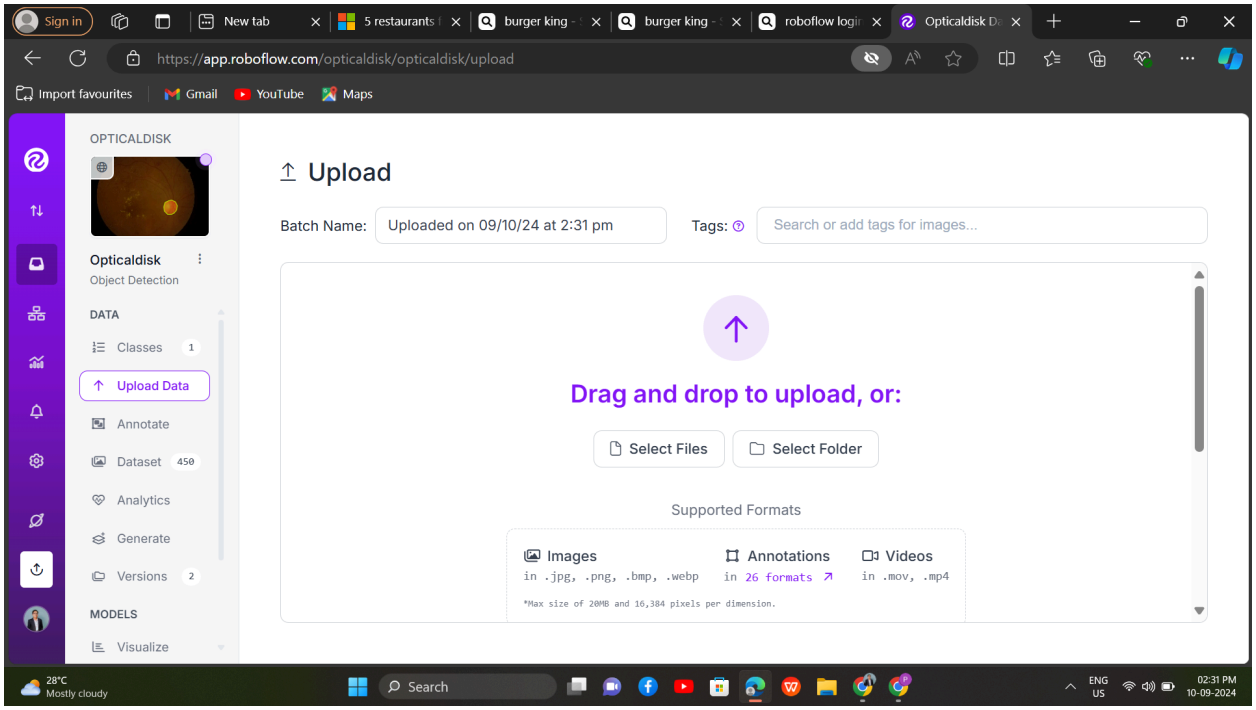
1. Create a Roboflow Account

- Sign up or log in to [Roboflow](https://roboflow.com).
- Once logged in, you will be directed to the Roboflow dashboard.



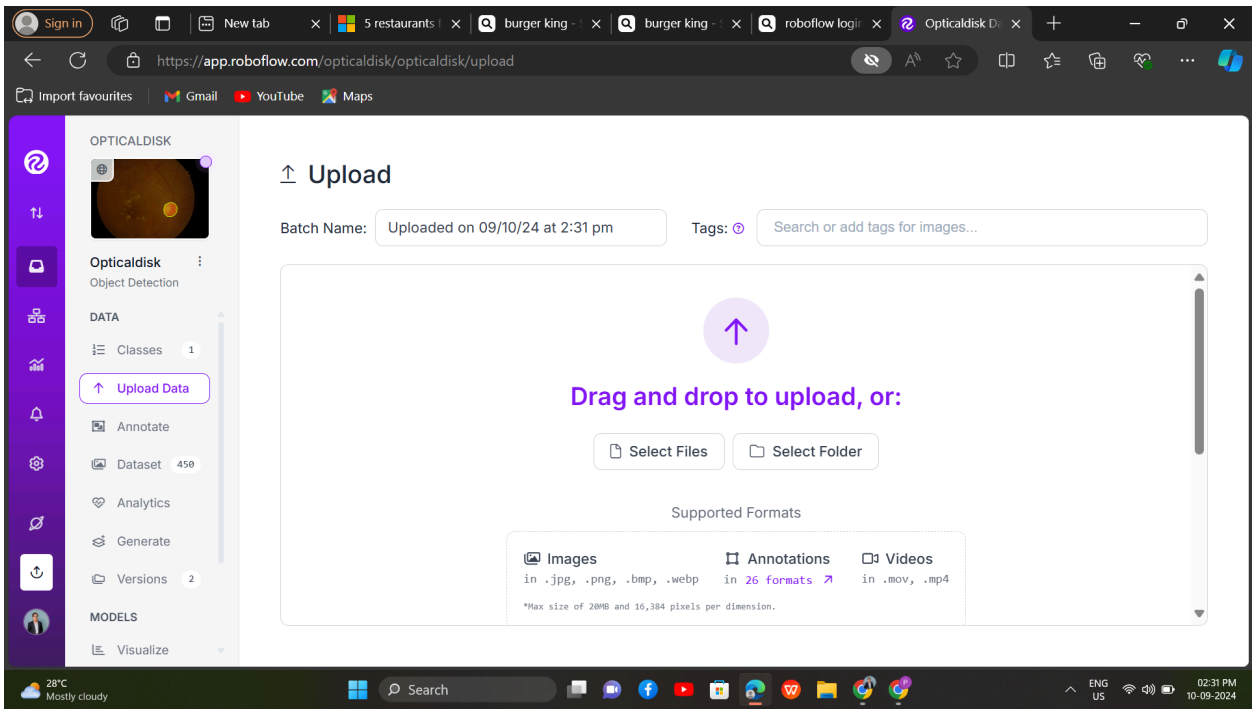
2. Create a New Project

- Click on Create New Project on the dashboard.
- Give your project a name related to diabetic retinopathy (e.g., "Diabetic Retinopathy Detection").
- Choose your task type: Object Detection (since you want to label different types of exudates).
- Select the type of annotation you want: Bounding Box for detecting hard and soft exudates.



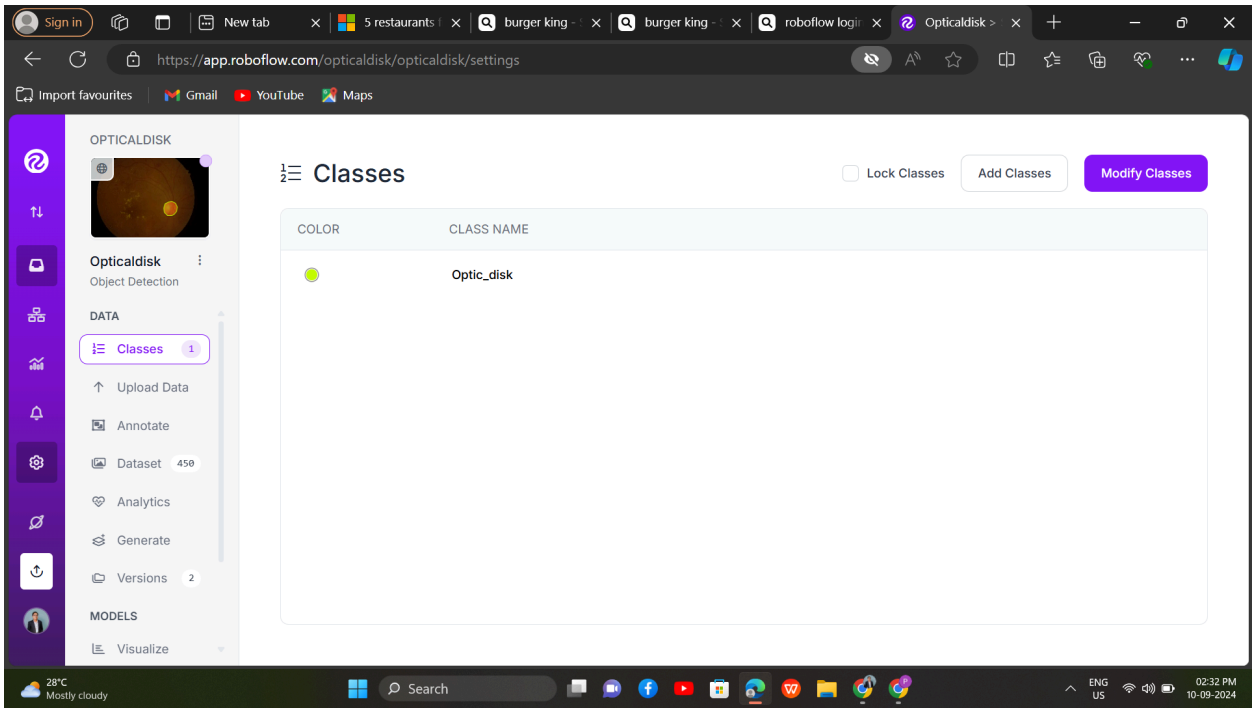
3. Upload Your Images

- After creating the project, you will be prompted to upload images.
- Drag and drop your dataset or select images from your local files. You can upload images for training, testing, and validation here.



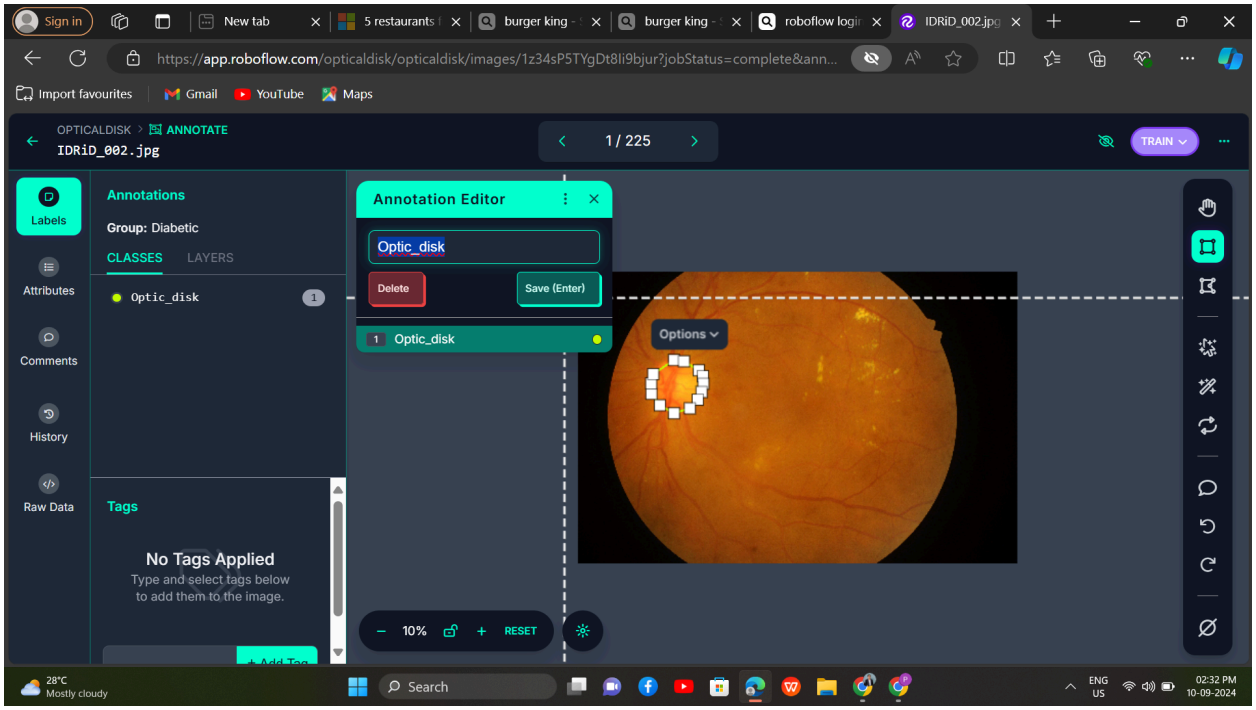
4. Add Classes for Labeling

- After uploading your images, Roboflow will ask you to define your classes.
- Add your classes: Hard Exudates, Soft Exudates, and No Exudates .

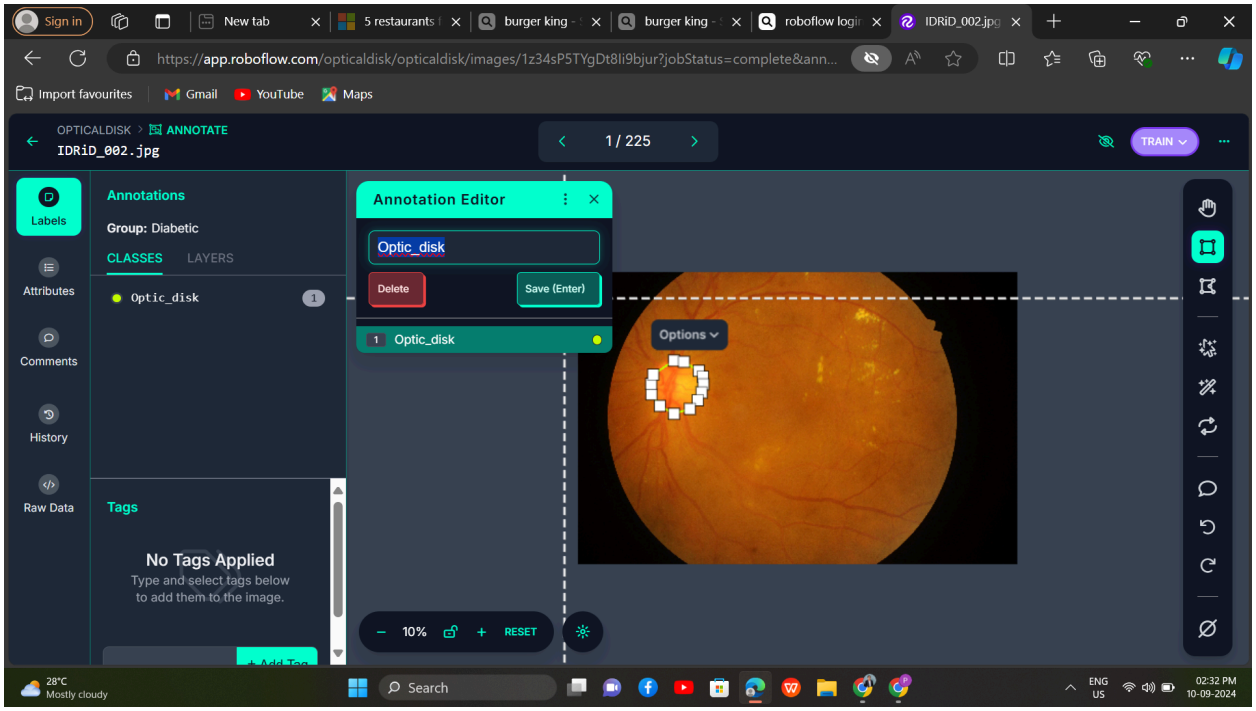


5. Annotate (Label) Your Images

- After your images are uploaded, start annotating them by drawing bounding boxes around the exudates.
- Label each region by selecting the appropriate class: Hard Exudates, Soft Exudates, or No Exudates.
- You can zoom in to accurately label the regions.

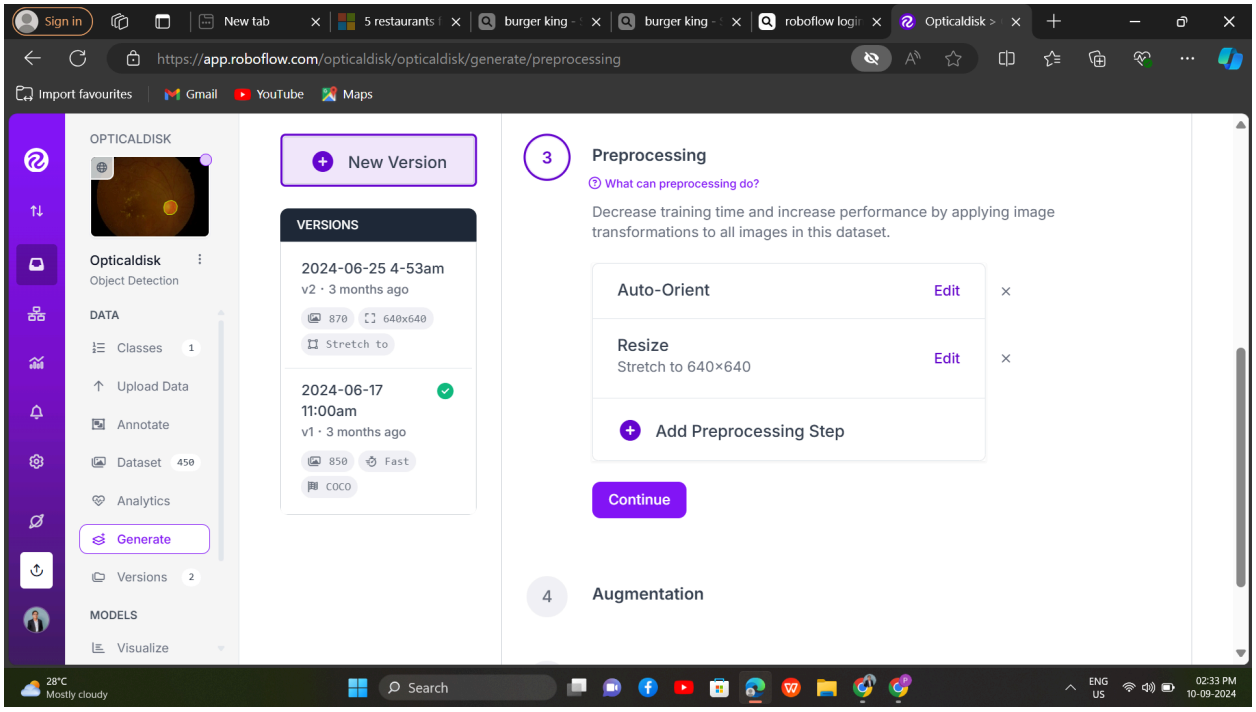


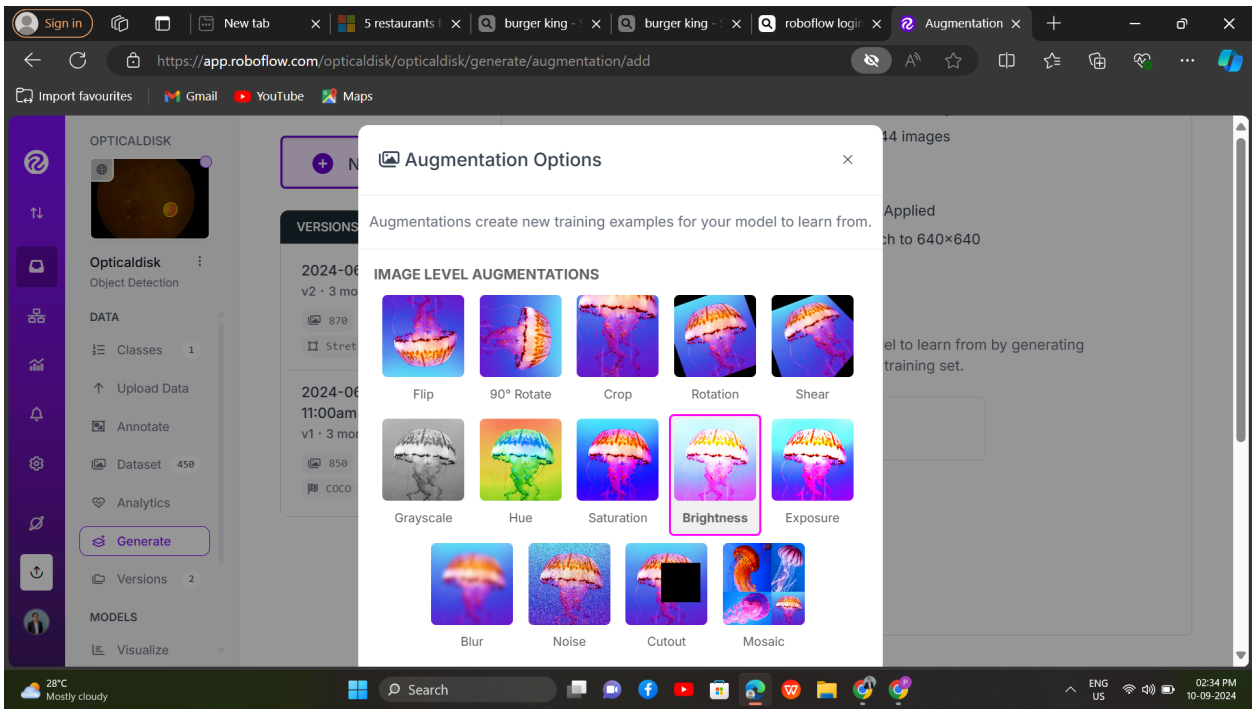
6. Review and Save Annotations
   - Once all your images are annotated, review your work.
   - You can re-adjust bounding boxes if necessary or add additional annotations.
   - After reviewing, save the annotations.



7. Preprocess Data

- Roboflow allows you to preprocess the images. You can resize, augment, or normalize your dataset to improve the performance of your model.
- Choose appropriate preprocessing settings based on your project needs.

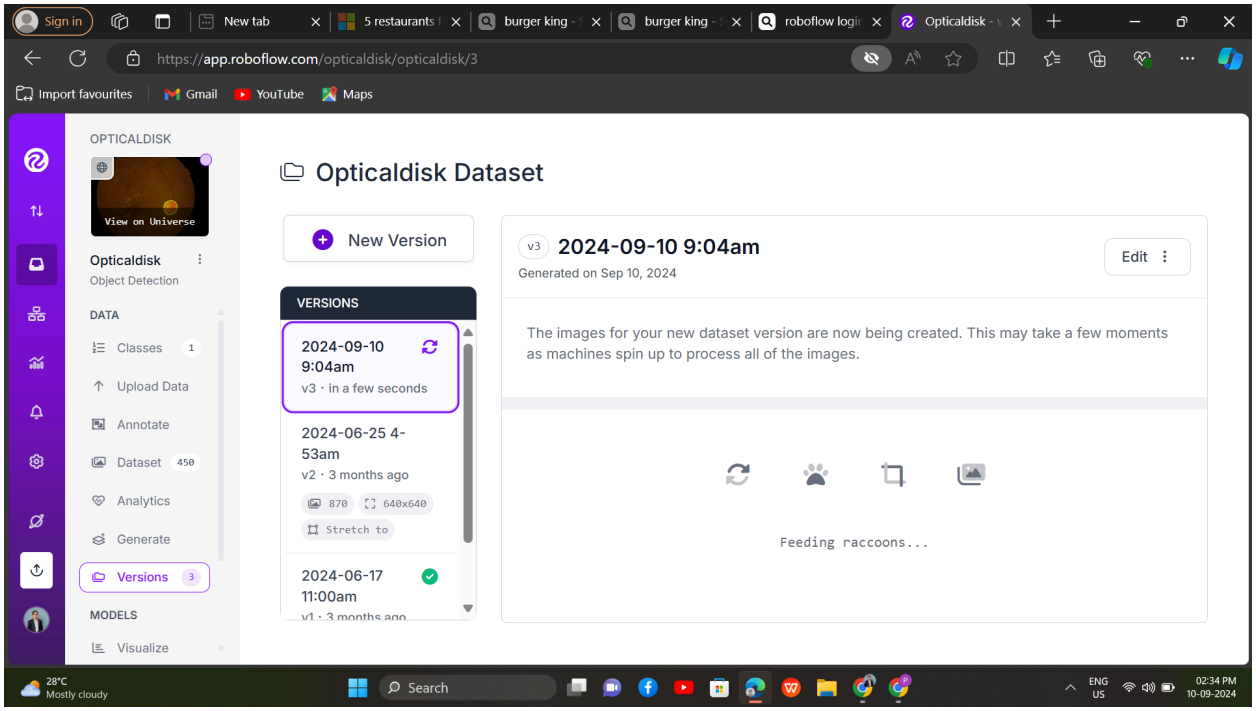


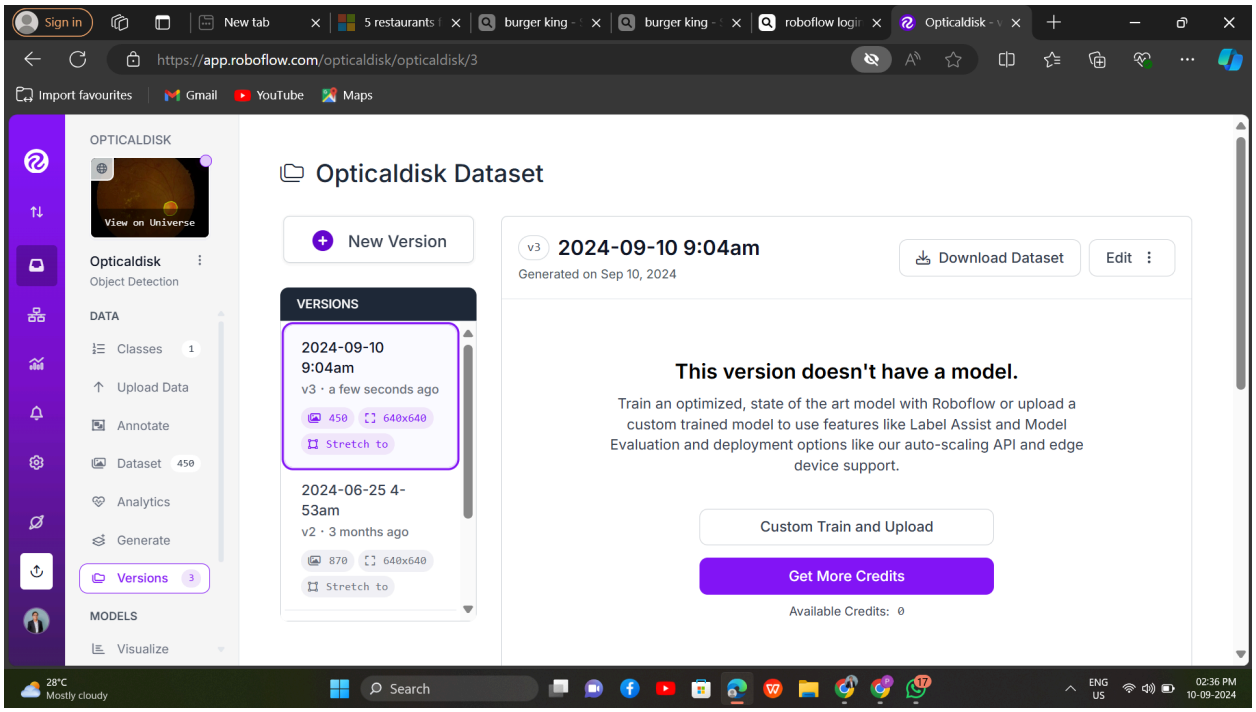


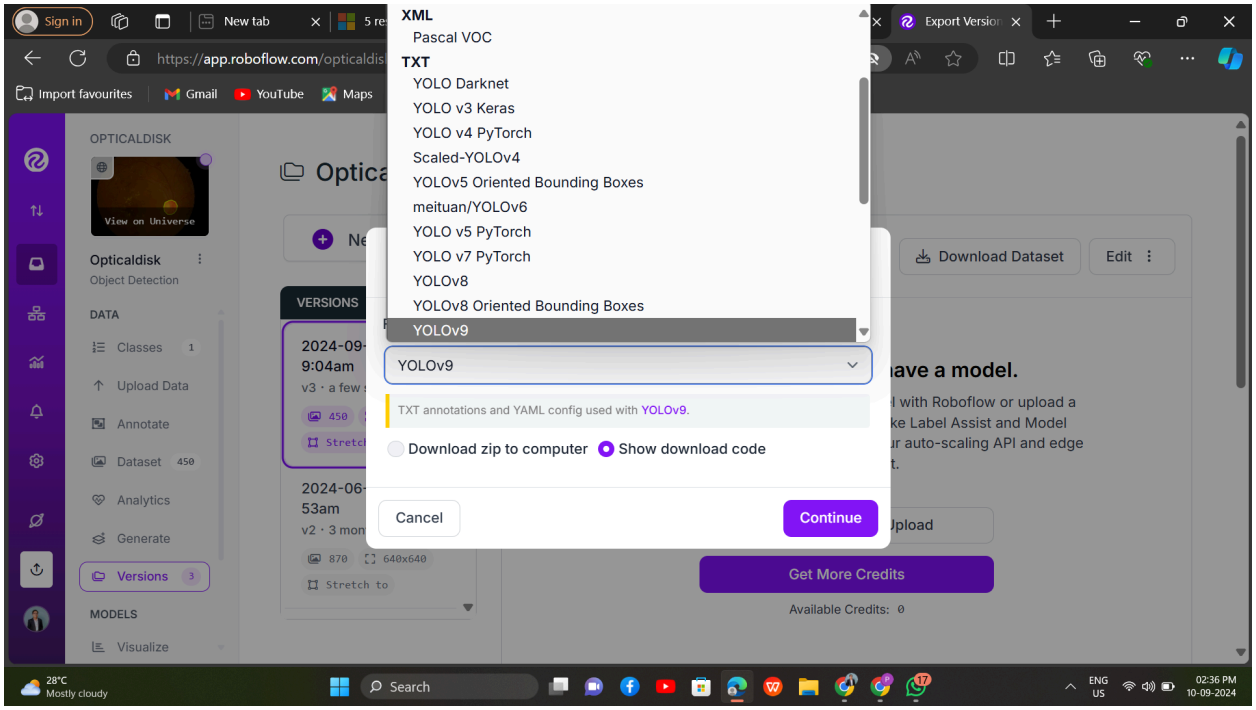
8. Export Dataset

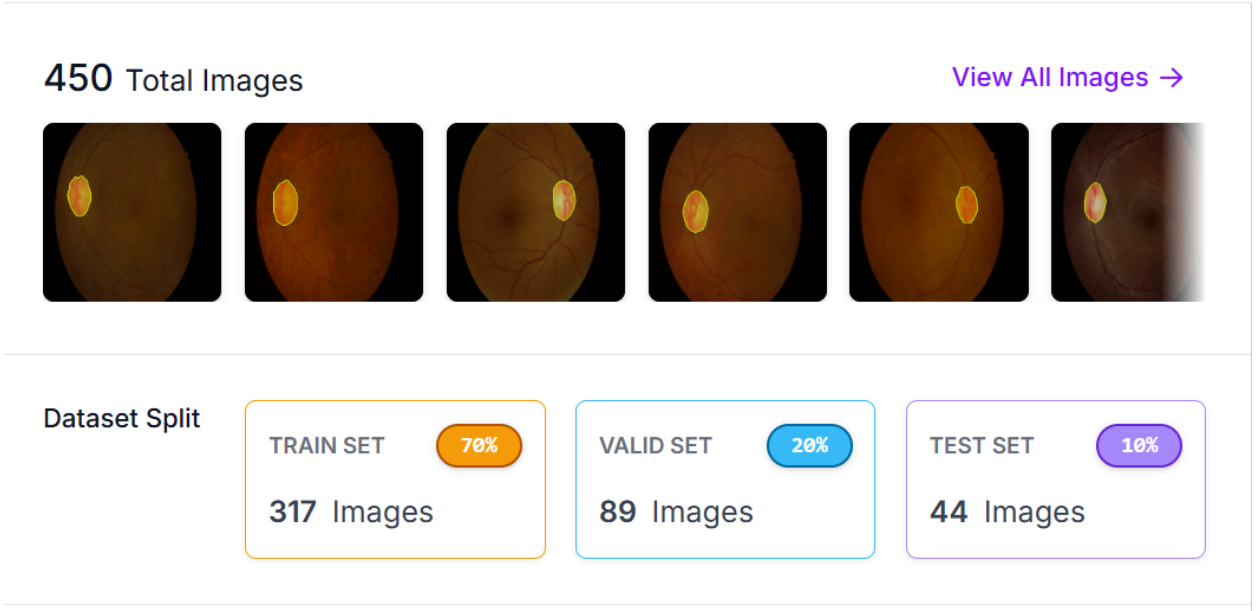
- After completing annotations and preprocessing, you can export the dataset.
- Choose your preferred format, such as TFRecord, COCO JSON, YOLOv5, or others.

- Download the dataset, which includes the images and the respective annotation files.

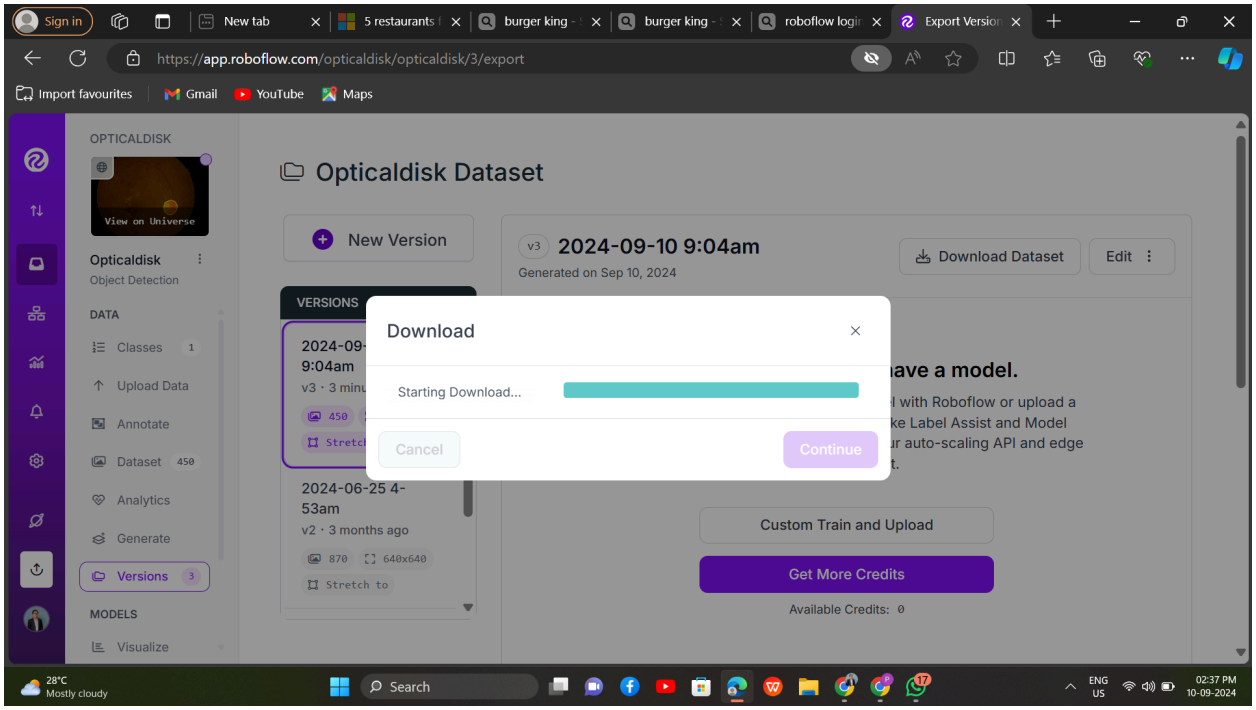








9. Download Labeled Data
   - Once your dataset is exported, download it to your local machine.
   - You will receive a zipped folder containing the images and annotation files in your chosen format.



Reference Link : https://roboflow.com/

Steps To Create Module 1:

!nvidia-smi is a command often used in environments like Jupyter Notebooks or Google Colab to check the status of NVIDIA GPUs.

- **Purpose**: It provides information about NVIDIA GPU devices installed on the system.
- **Output**: Displays a table with details such as GPU utilization, memory usage, temperature, power usage, and processes running on each GPU.
- **Usage**: Helpful for monitoring GPU performance, diagnosing issues related to GPU availability or performance, and verifying if CUDA-compatible operations can be performed.

For example, running !nvidia-smi might show something like this:

```
!nvidia-smi
```

```
Tue Jun 25 06:07:09 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |                    0 |
| N/A   38C    P8              9W /  70W |      0MiB / 15360MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

2)

```
import os
HOME = os.getcwd()
print(HOME)
```

```
/content
```

The code snippet import os and HOME = os.getcwd() demonstrates basic file system handling in Python using the os module. Here's a simple explanation:

1. **Importing os Module**:
    - import os: This statement imports the Python standard library module os, which provides functions for interacting with the operating system.
2. **Getting Current Working Directory**:
    - HOME = os.getcwd(): This line uses the os.getcwd() function to retrieve the current working directory (cwd) of the Python script.
    - **Explanation**:
        - os.getcwd(): This function returns a string representing the current working directory where the Python script is executing.
        - HOME: This variable stores the current working directory path obtained from os.getcwd().
3. **Printing the Current Working Directory**:
    - print(HOME): This prints out the current working directory path stored in the variable HOME.

In Google Colab, when you run `os.getcwd()`, it returns the current working directory path. This path typically refers to the location where your Colab notebook is executing. It's a useful function to confirm the current directory in which your scripts or notebooks are running, especially when you need to manage file paths or organize your project's structure.

3) **YOLOV10 installation steps:**

1)commonds:
Step 1:!pip install -q git+https://github.com/THU-MIG/yolov10.git

```
!pip install -q git+https://github.com/THU-MIG/yolov10.git

  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.3/21.3 MB 55.9 MB/s eta 0:00:00
  Building wheel for ultralytics (pyproject.toml) ... done
```

2)**ROboflow installations:**

#What is Roboflow?

Roboflow is a platform that simplifies the process of building and deploying computer vision models. It offers tools for managing datasets, annotation, model training, and deployment. Here's a brief overview of its key features:

1. **Dataset Management**: Roboflow allows you to upload, organize, and preprocess your datasets. It supports various formats and offers tools for augmenting your data to improve model performance.
2. **Annotation Tools**: It provides a user-friendly interface for annotating images. You can create bounding boxes, polygons, and segmentation masks.
3. **Model Training**: Roboflow integrates with popular machine learning frameworks like TensorFlow, PyTorch, and Detectron2. It provides a streamlined process for training object detection, classification, and segmentation models.
4. **Deployment**: Once your model is trained, Roboflow offers deployment options to various platforms, including cloud services, edge devices, and mobile apps.
5. **API Access**: Roboflow provides APIs for integrating their services into your applications, allowing for automated dataset management, model training, and inference.

2)**Create the directory for weights**

```
!mkdir -p {HOME}/weights
```

 # Download the YOLOv10 weights

```
!wget -P {HOME}/weights -q https://github.com/THU-MIG/yolov10/releases/download/v1.1/yolov10n.pt
!wget -P {HOME}/weights -q https://github.com/THU-MIG/yolov10/releases/download/v1.1/yolov10s.pt
!wget -P {HOME}/weights -q https://github.com/THU-MIG/yolov10/releases/download/v1.1/yolov10m.pt
!wget -P {HOME}/weights -q https://github.com/THU-MIG/yolov10/releases/download/v1.1/yolov10b.pt
```

```
!wget -P {HOME}/weights -q
https://github.com/THU-MIG/yolov10/releases/download/v1.1/yolov10x.pt
!wget -P {HOME}/weights -q
https://github.com/THU-MIG/yolov10/releases/download/v1.1/yolov10l.pt
!ls -lh {HOME}/weights
```

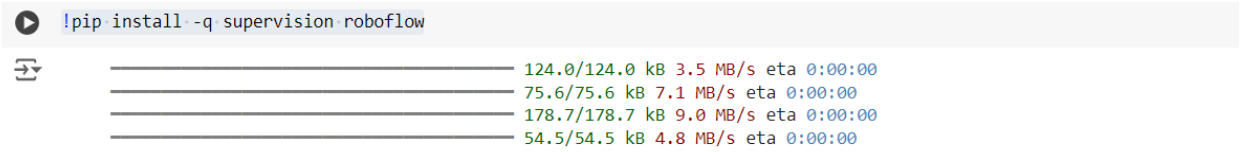
YOLOv10 weights are the parameters of a trained model. They help the model:

1. **Recognize Patterns**: Learn features from images.
2. **Save Time**: Use a model pre-trained on a large dataset.
3. **Make Predictions**: Detect and classify objects in new images.

Step 3: Download dataset from Roboflow Universe

Command:
1)!pip install -q supervision roboflow

Running this command in a Google Colab cell will install the supervision and roboflow packages quietly (without showing the installation progress details):
-q flag stands for "quiet," which reduces the amount of output from the installation process.
supervision and roboflow are the names of the Python packages you want to install.



2)

!pip install roboflow

This command installs the Roboflow Python package, which provides a convenient way to interact with Roboflow's API for dataset management and model integration.

from roboflow import Roboflow
This line imports the Roboflow class from the installed package.

rf = Roboflow(api_key="aj1YNkFeRPD8FAf7DMpN")
This line initializes an instance of the **Roboflow** class using your API key. The API key **(aj1YNkFeRPD8FAf7DMpN)** allows you to authenticate and access your Roboflow account programmatically.

project = rf.workspace("opticaldisk").project("opticaldisk")

This line accesses a specific project within your Roboflow workspace. Here, both the workspace and the project are named "opticaldisk." The **workspace** method returns a **Workspace** object, and the **project** method returns a **Project** object.

version = project.version(2)
This line accesses version 2 of the specified project. The **version** method returns a **Version** object representing the specific version of the project.

dataset = version.download("yolov9")
This line downloads the dataset in a format compatible with YOLOv9. The download method retrieves the dataset and prepares it for use with the specified model format (in this case, YOLOv9). The returned dataset object contains the data and relevant metadata.

3) If using a dataset from Roboflow Universe, run the command below(YOLOV10).
These commands modify the data.yaml file, which is typically used to configure dataset paths for training object detection models. By deleting the last four lines and adding new paths for test, train, and validation datasets, you ensure the file accurately reflects the current dataset locations. This is crucial for the training process to work correctly with the right data

```
!sed -i '$d' {dataset.location}/data.yaml
!sed -i '$d' {dataset.location}/data.yaml
!sed -i '$d' {dataset.location}/data.yaml
!sed -i '$d' {dataset.location}/data.yaml
!echo -e "test: ../test/images\ntrain: ../train/images\nval: ../valid/images" >> {dataset.location}/data.yaml
```

4)Custom Training

```
%cd {HOME}

!yolo task=detect mode=train epochs=30 batch=32 plots=True \
model={HOME}/weights/yolov10s.pt \
data={dataset.location}/data.yaml
```

This command uses the yolo command-line interface to start training a YOLOv10 model. Here are the key components:

- task=detect: Specifies that the task is object detection.
- mode=train: Sets the mode to training.
- epochs=30: Specifies the number of epochs for training (30 epochs in this case).
- batch=32: Sets the batch size for training to 32.
- plots=True: Enables the generation of plots during training.
- model={HOME}/weights/yolov10s.pt: Specifies the pre-trained model weights to be used for training, located in the weights directory within the home directory.
- data={dataset.location}/data.yaml: Points to the data.yaml file, which contains the dataset configuration, including paths to the training, validation, and test datasets.

5)Training Process:

```
[ ] !ls {HOME}/runs/detect/train/
```

```
args.yaml                                       PR_curve.png        train_batch461.jpg
confusion_matrix_normalized.png                 R_curve.png         train_batch462.jpg
confusion_matrix.png                            results.csv         val_batch0_labels.jpg
events.out.tfevents.1719295765.c8d03591b0fe.2974.0  results.png     val_batch0_pred.jpg
F1_curve.png                                    train_batch0.jpg    val_batch1_labels.jpg
labels_correlogram.jpg                          train_batch1.jpg    val_batch1_pred.jpg
labels.jpg                                      train_batch2.jpg    weights
P_curve.png                                     train_batch460.jpg
```

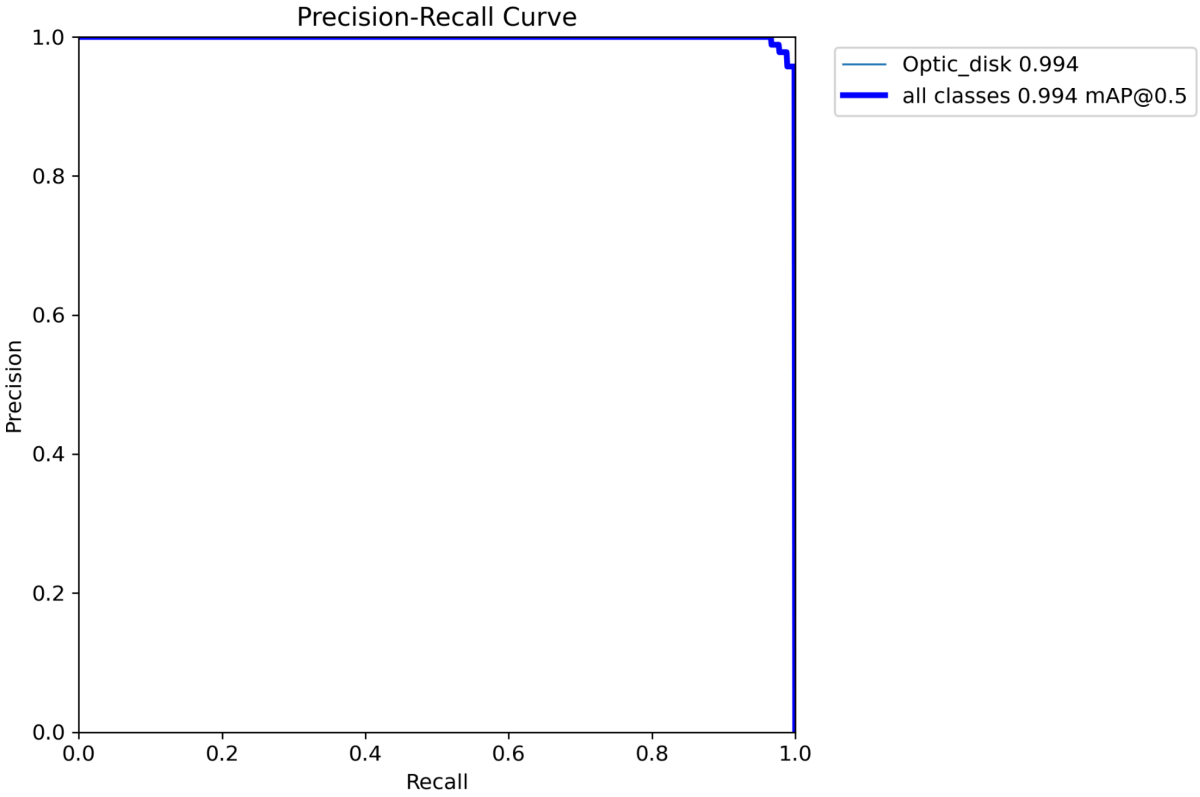
Running this command will list the contents of the directory {HOME}/runs/detect/train/. This directory is typically used to store the outputs and logs from the training process of a YOLO model.

!ls is a shell command to list directory contents.

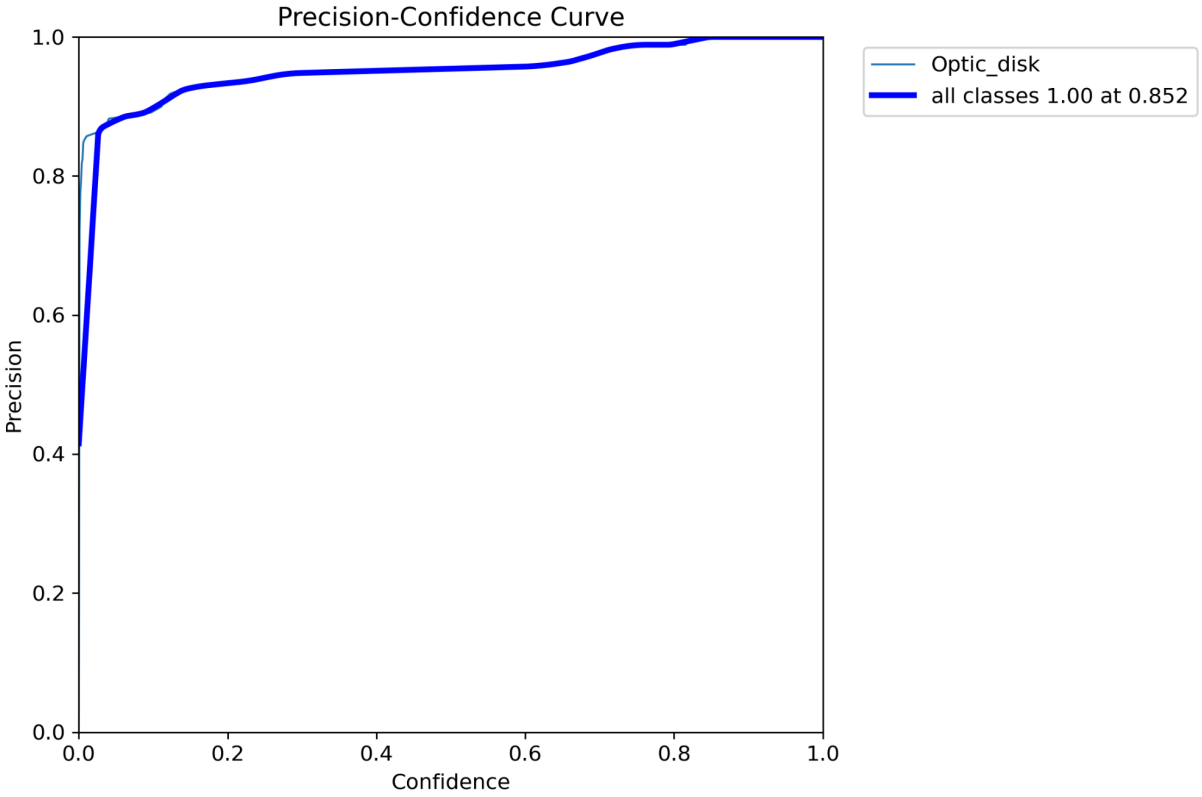
{HOME} refers to the home directory.

/runs/detect/train/ is the subdirectory where training outputs are stored.

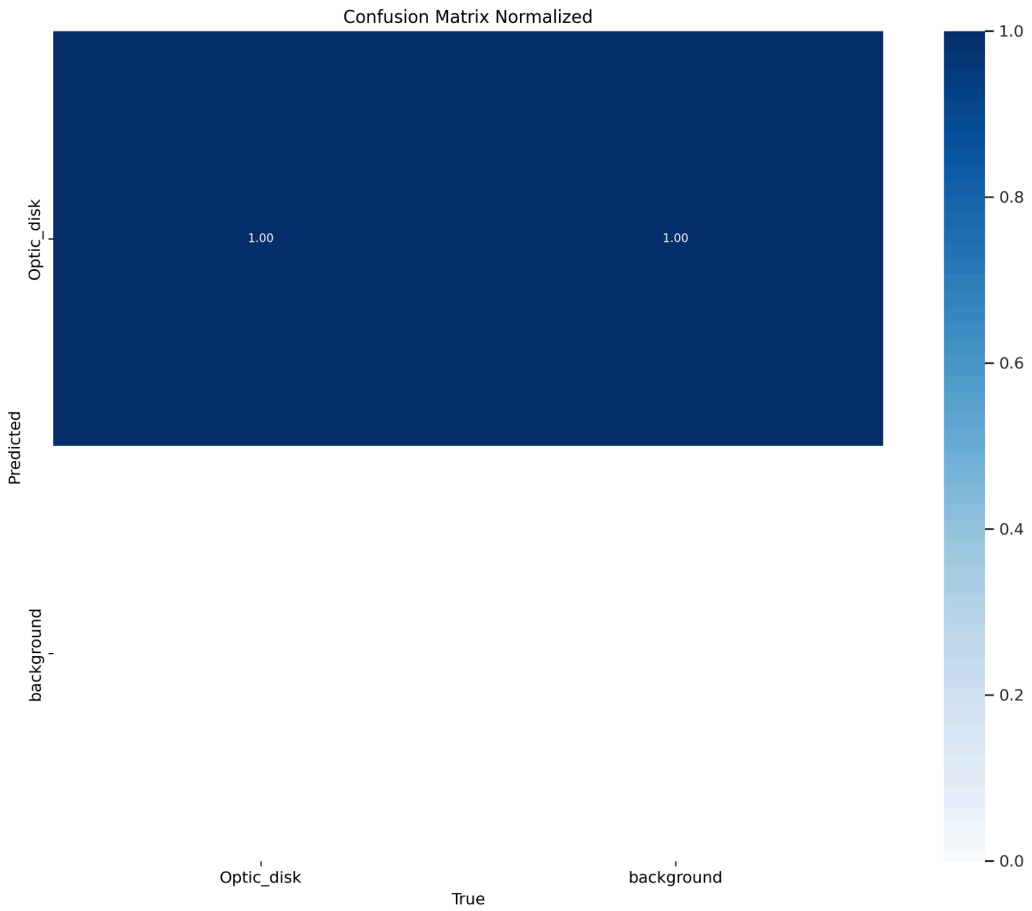
Output images:



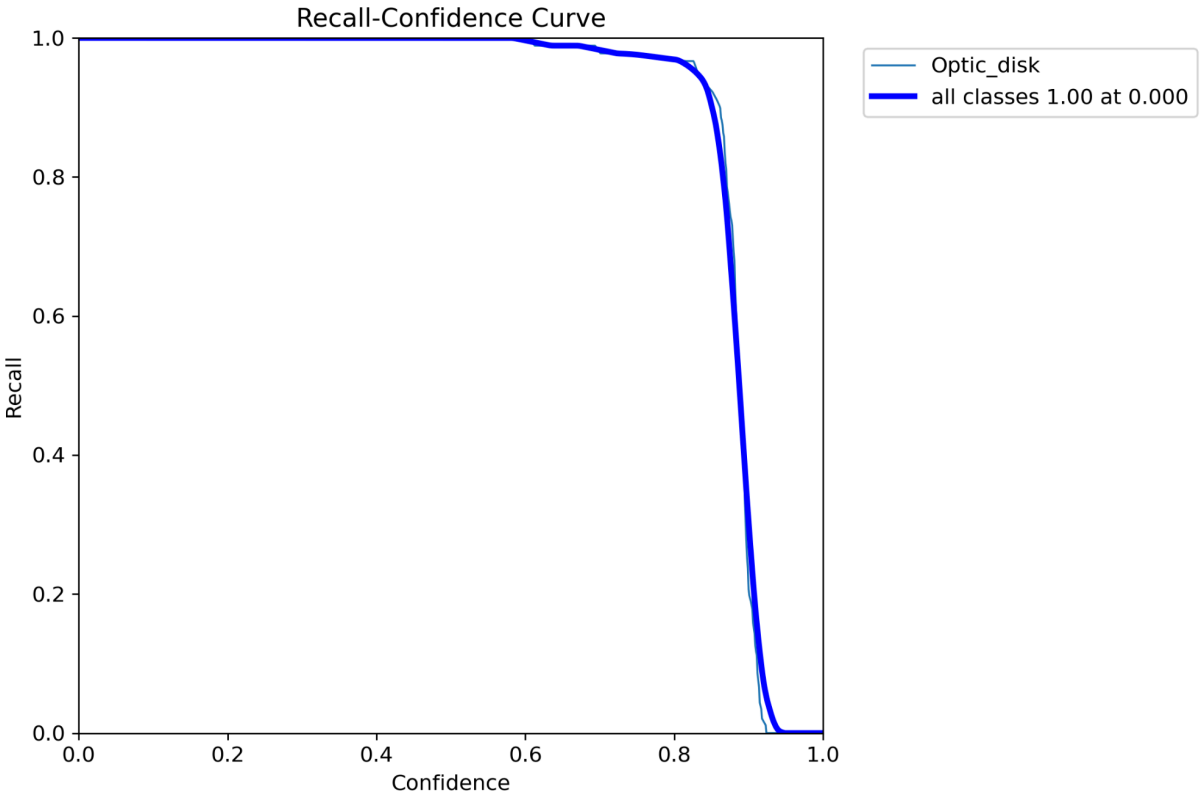
PR_curve.png :train_batch461.jpg



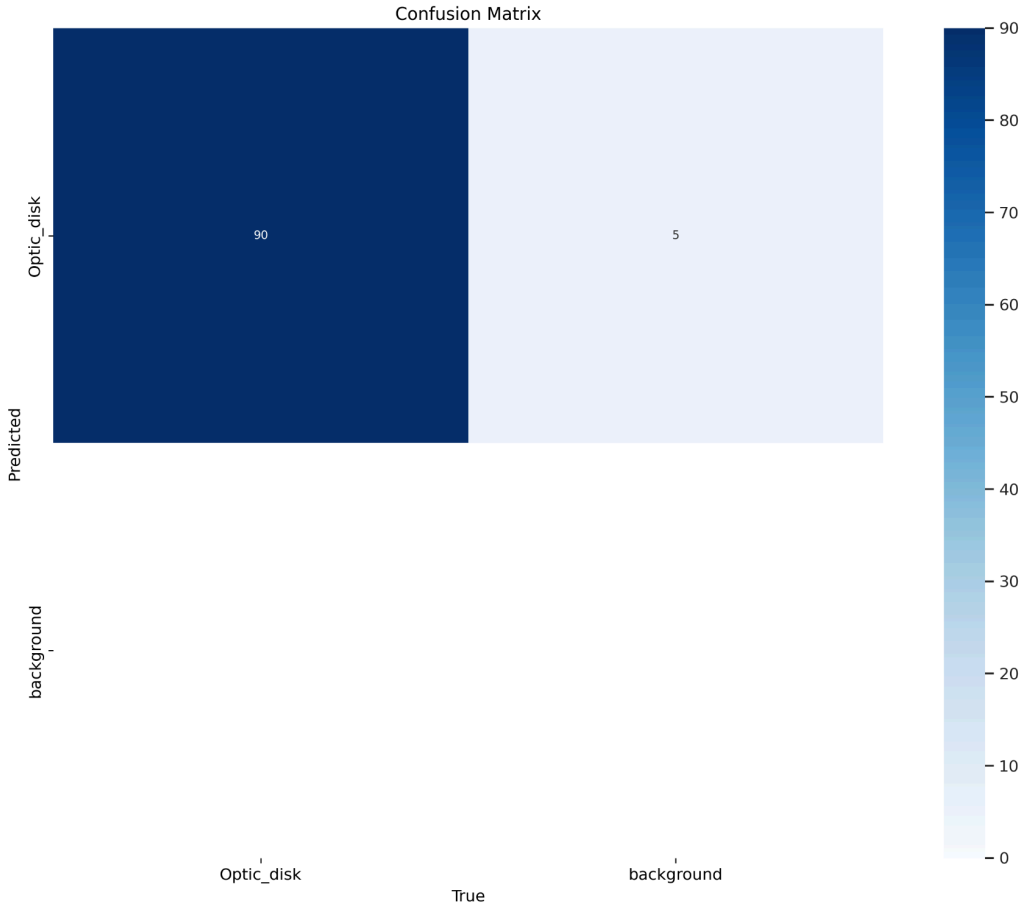
P_curve.png:train_batch460.jpg



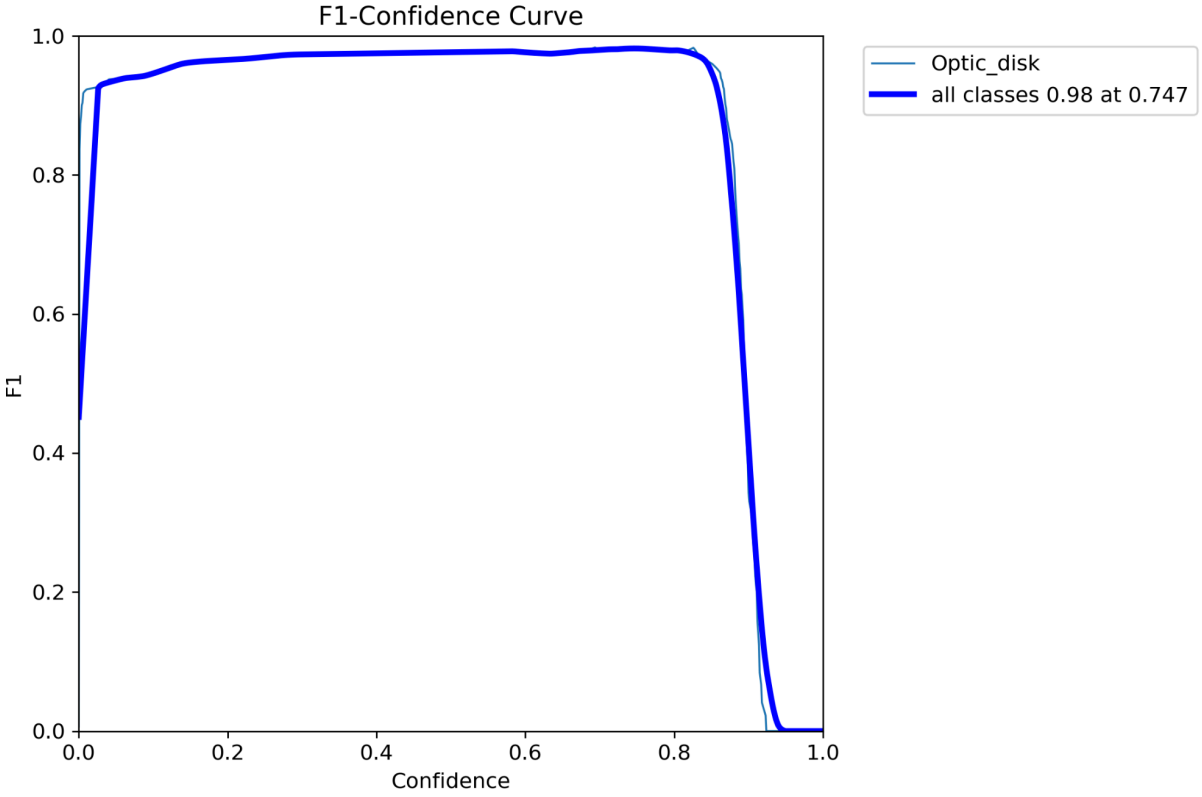
confusion_matrix_normalized.png



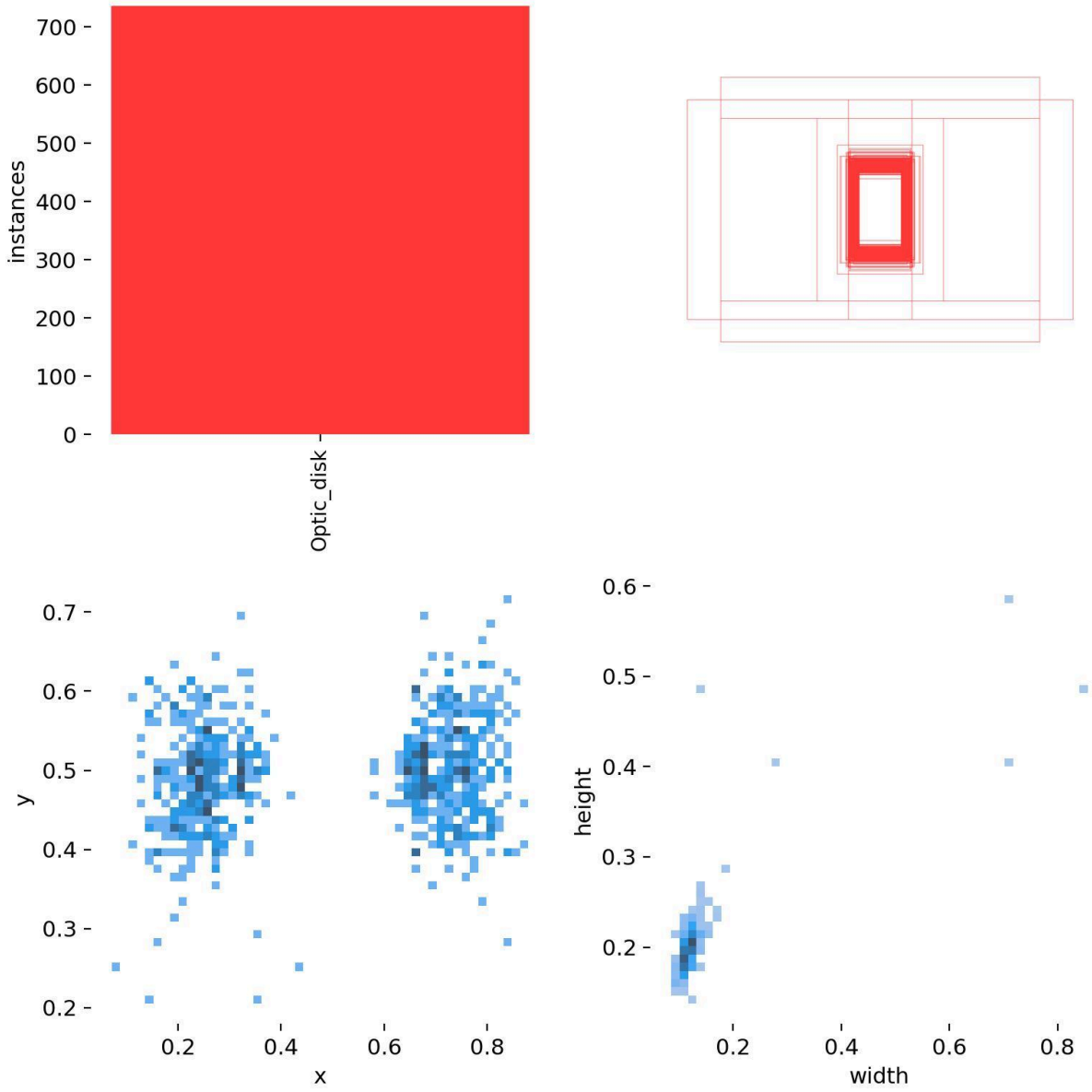
R_curve.png :train_batch462.jpg



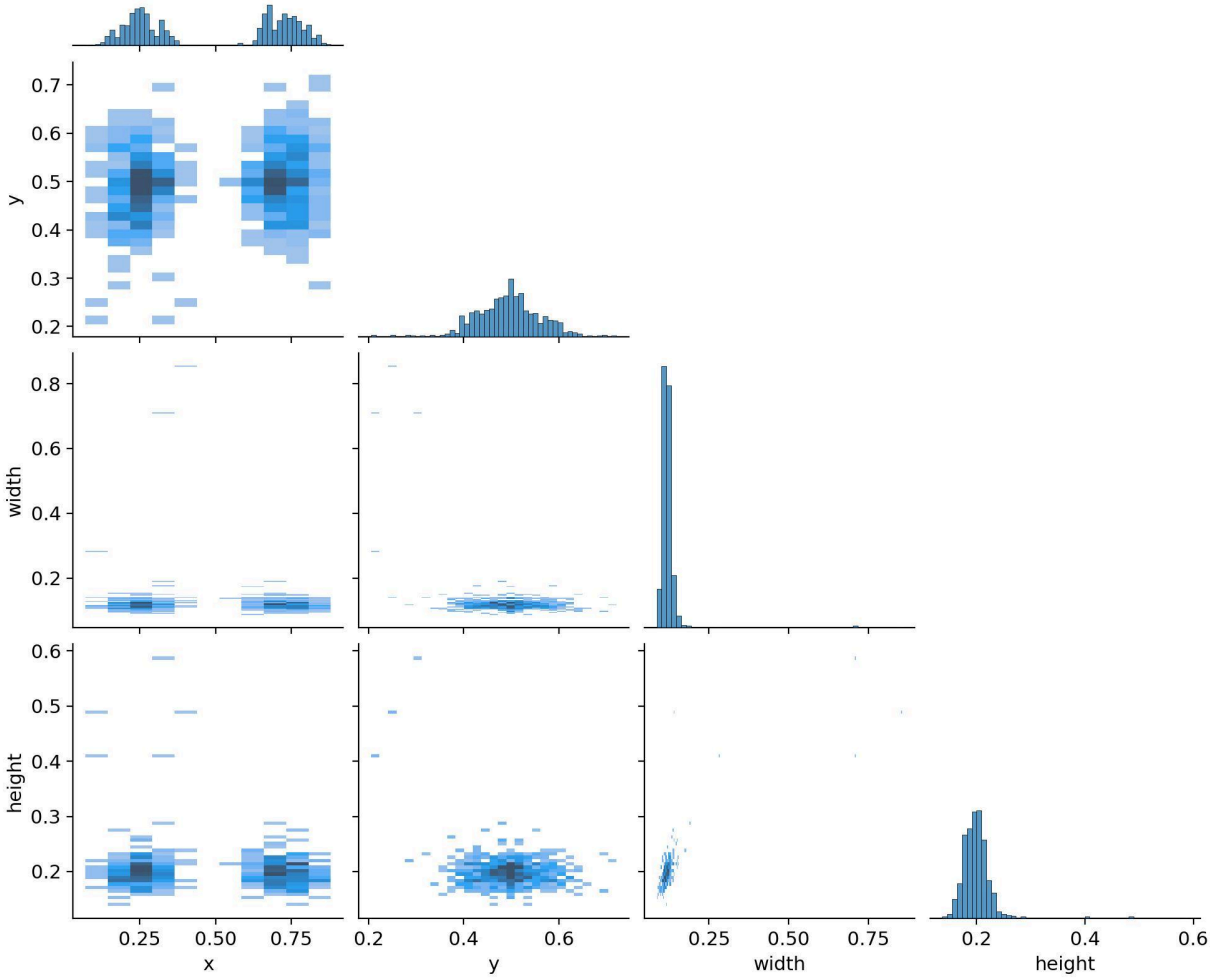
Confusion Matrix


F1-Confidence Curve
Optic_disk
all classes 0.98 at 0.747
F1
Confidence
0.0
0.2
0.4
0.6
0.8
1.0

F1:Curve


instances
700
600
500
400
300
200
100
0
Optic_disk
y
0.7
0.6
0.5
0.4
0.3
0.2
0.2
0.4
0.6
0.8
x
height
0.6
0.5
0.4
0.3
0.2
0.2
0.4
0.6
0.8
width

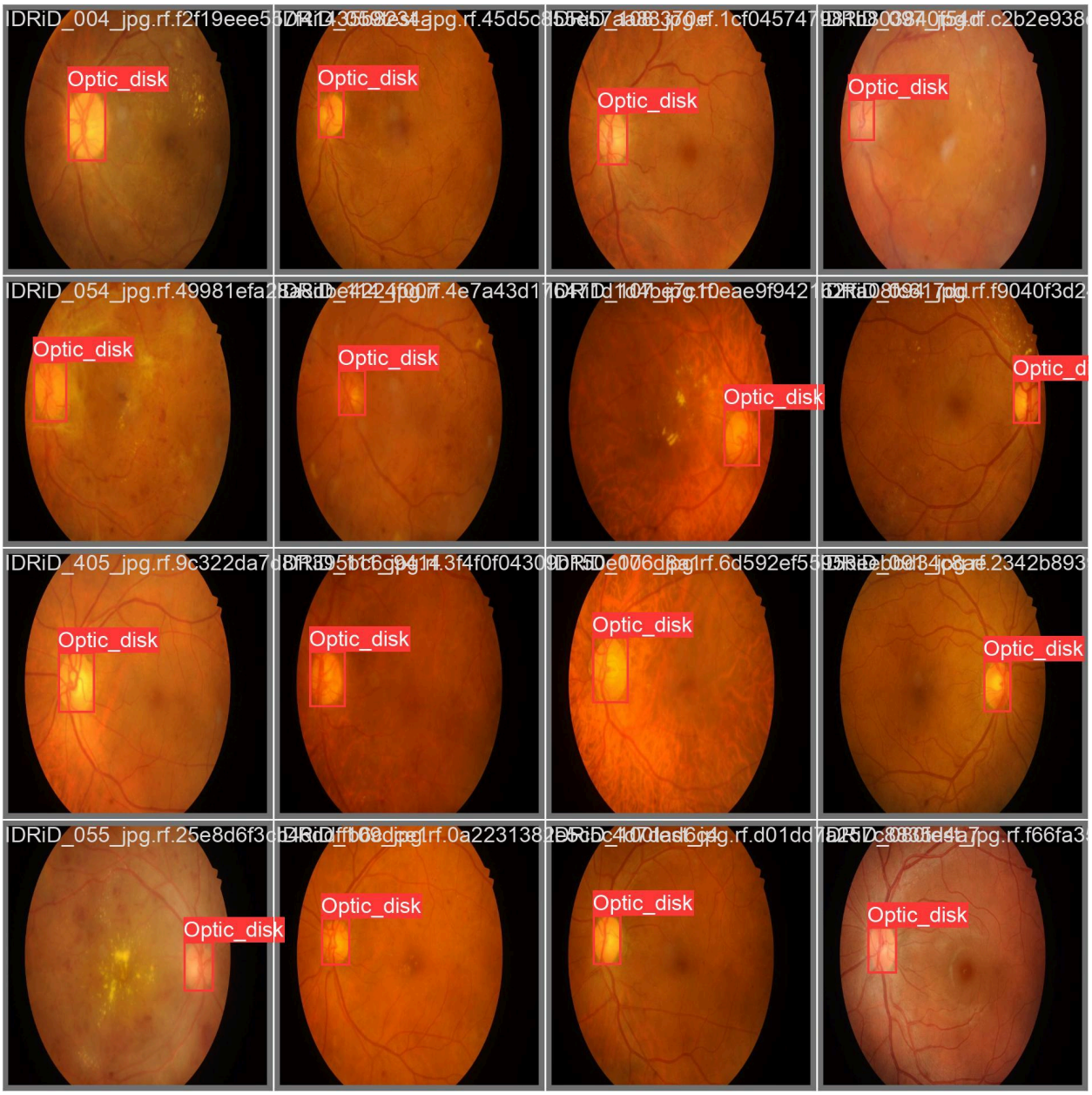
Labels



Labels_correlogram

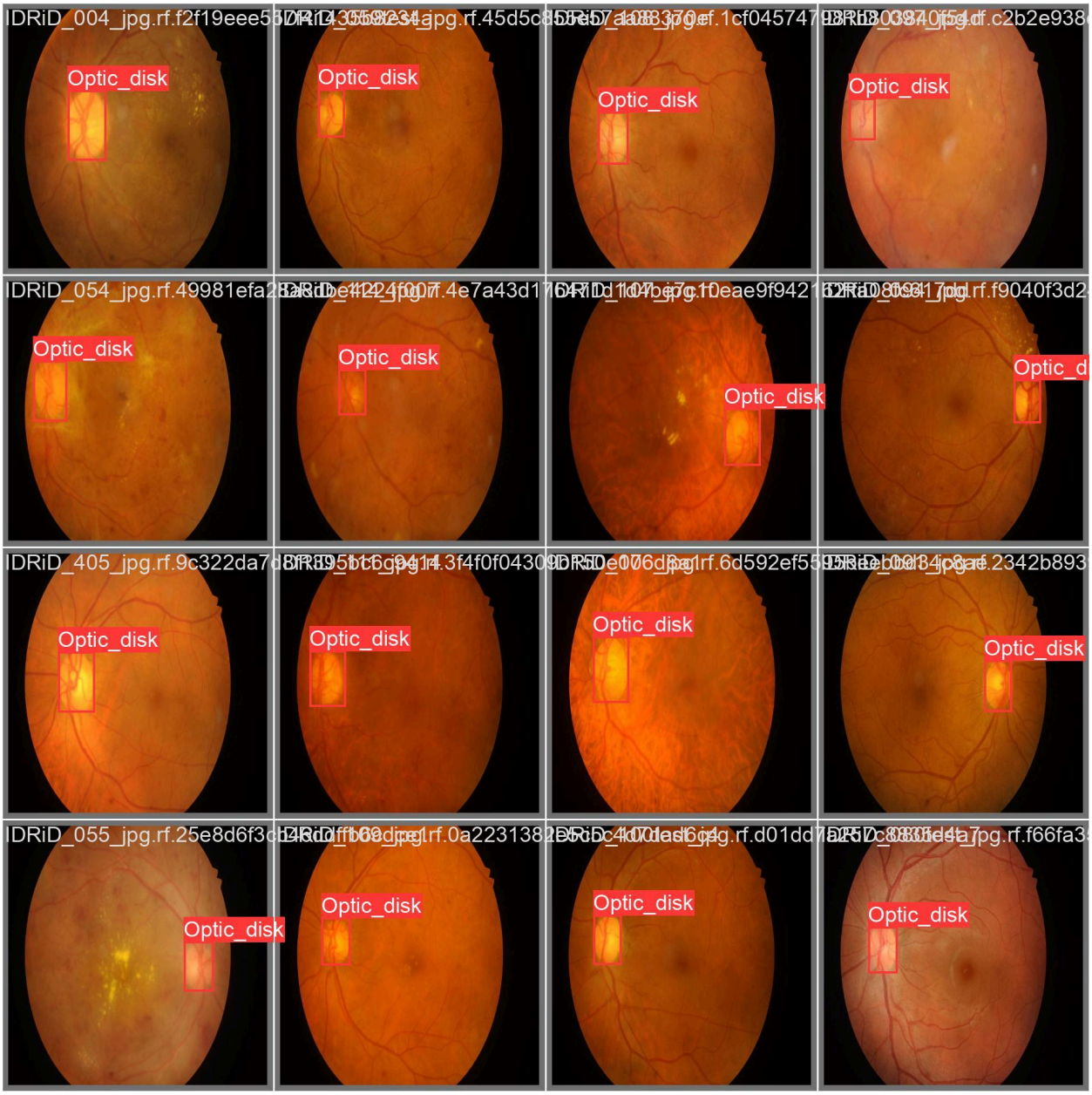
6)From IPython.display import Image

display(Image(filename=f'{HOME}/runs/detect/train/val_batch1_pred.jpg', width=600)).



Val_batch1_pred

```
%cd {HOME}
Image(filename=f'{HOME}/runs/detect/train/val_batch1_pred.jpg',width=600)
```



Val_batch1_labels

7)

```
[ ] %cd {HOME}
    !yolo task=detect mode=val model={HOME}/runs/detect/train/weights/best.pt data={dataset.location}/data.yaml
```

```
/content
Ultralytics YOLOv8.1.34 🚀 Python-3.10.12 torch-2.3.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLOv10s summary (fused): 293 layers, 8035734 parameters, 0 gradients, 24.4 GFLOPs
val: Scanning /content/Opticaldisk-2/valid/labels.cache... 90 images, 0 backgrounds, 0 corrupt: 100% 90/90 [00:00<?, ?it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:06<00:00,  1.16s/it]
                   all         90         90      0.988      0.978      0.994      0.831
Speed: 10.9ms preprocess, 18.4ms inference, 0.0ms loss, 5.0ms postprocess per image
Results saved to runs/detect/val
💡 Learn more at https://docs.ultralytics.com/modes/val
```

This command uses the yolo command-line interface to validate a YOLO model. Here are the key components:

- **task=detect: Specifies that the task is object detection.**
- **mode=val: Sets the mode to validation.**
- **model={HOME}/runs/detect/train/weights/best.pt: Specifies the path to the trained model's best weights, which were saved during training.**
- **data={dataset.location}/data.yaml: Points to the data.yaml file, which contains the dataset configuration, including paths to the training, validation, and test datasets.**

8)

```
[ ] %cd {HOME}
    !yolo task=detect mode=predict model={HOME}/runs/detect/train/weights/best.pt conf=0.25 source={dataset.location}/test/images save=True
```

This command uses the yolo command-line interface to make predictions using the YOLO model. Here are the key components:

- **task=detect: Specifies that the task is object detection.**
- **mode=predict: Sets the mode to prediction.**
- **model={HOME}/runs/detect/train/weights/best.pt: Specifies the path to the trained model's best weights, which were saved during training.**
- **conf=0.25: Sets the confidence threshold to 0.25. Predictions with confidence scores below this value will be ignored.**
- **source={dataset.location}/test/images: Points to the directory containing the test images for which predictions will be made.**
- **save=True: Indicates that the prediction results should be saved.**

9)Green channel Images:

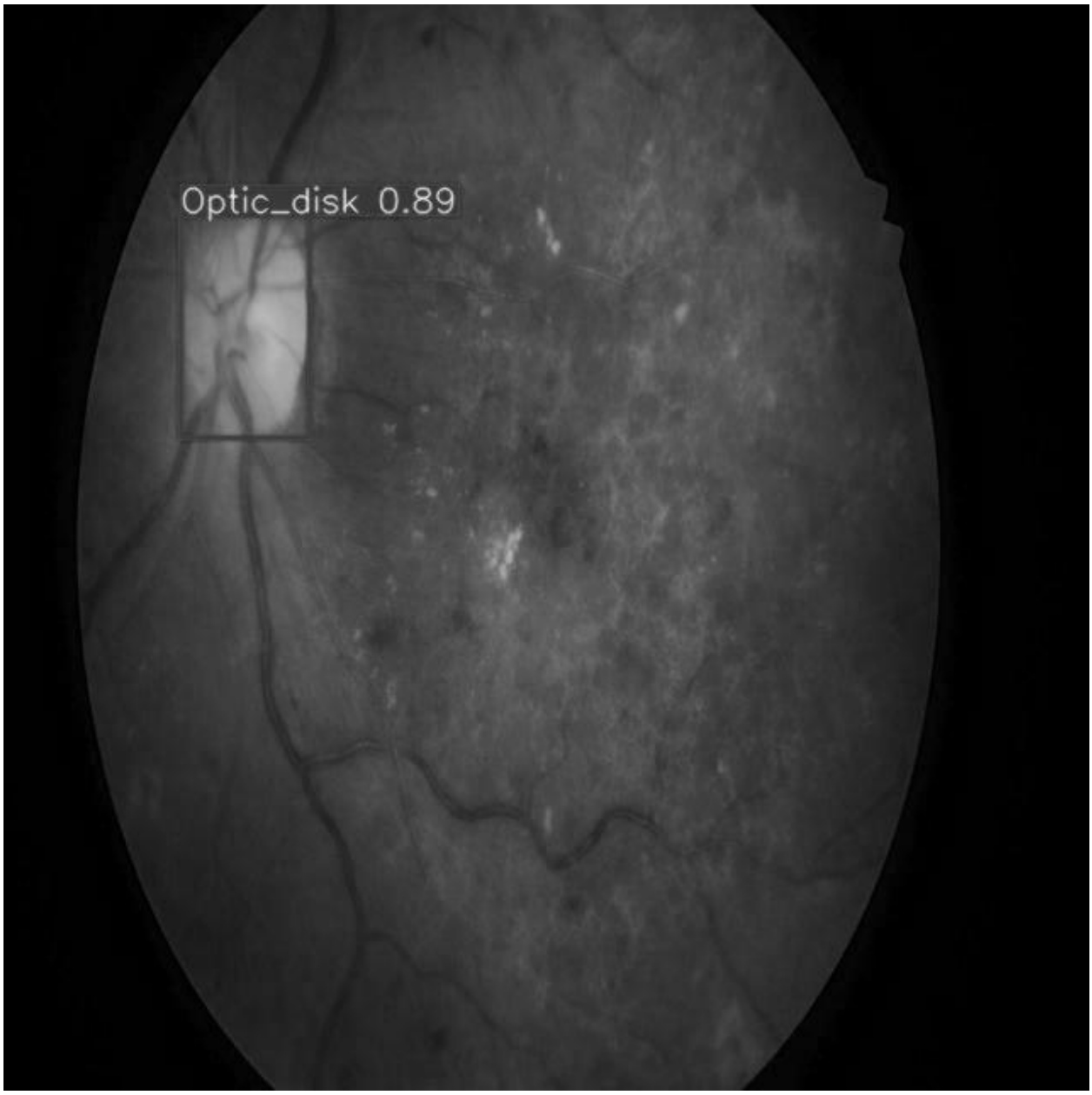
```
import os

# Define the folder path
output_folder = "/content/green_channel_images"

# Create the folder if it doesn't exist
os.makedirs(output_folder, exist_ok=True)
print(f"Folder '{output_folder}' created or already exists.")
```

```
Folder '/content/green_channel_images' created or already exists.
```

Code:

```
import cv2
import matplotlib.pyplot as plt
import os  # Import the os module for path operations

# Load the original right image
image_path = "/content/runs/detect/predict/IDRiD_008_jpg.rf.57b077fe471d548dfceafaed248b0d3a.jpg"
right_image = cv2.imread(image_path)

# Extract the green channel
green_channel = right_image[:, :, 1]

# Specify the output folder where you want to save the green channel image
output_folder = "/content/green_channel_images"  # Example path, change as necessary

# Save the green channel image to the folder
green_image_path = os.path.join(output_folder, "IDRiD_008_green_channel.jpg")
cv2.imwrite(green_image_path, green_channel)

print(f"Green channel image saved at: {green_image_path}")

# Display the green channel image
plt.figure(figsize=(5, 5))
plt.imshow(green_channel, cmap='gray')
plt.axis('off')
plt.show()
```

Output:



IDRiD_008_green_channel

10)Remove optic disk and draw a circle:
Code:

```
import cv2
import numpy as np
import urllib.request
import os

# Global variables for circle parameters
drawing = False  # True if mouse is pressed
ix, iy = -1, -1  # Starting coordinates of the circle
img = None  # Placeholder for the image

def draw_circle(event, x, y, flags, param):
    global ix, iy, drawing, img

    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        ix, iy = x, y

    elif event == cv2.EVENT_MOUSEMOVE:
        if drawing:
            img_copy = img.copy()
            radius = int(((x - ix) ** 2 + (y - iy) ** 2) ** 0.5)
            cv2.circle(img_copy, (ix, iy), radius, (0, 0, 0), -1)
            cv2.imshow('image', img_copy)

    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
        radius = int(((x - ix) ** 2 + (y - iy) ** 2) ** 0.5)
        cv2.circle(img, (ix, iy), radius, (0, 0, 0), -1)
        cv2.imshow('image', img)
```

```
# Function to load image from URL or local path
def load_image(path):
    if os.path.isfile(path):
        # Load image from local file path
        img = cv2.imread(path)
    else:
        # Load image from URL
        resp = urllib.request.urlopen(path)
        img = np.asarray(bytearray(resp.read()), dtype="uint8")
        img = cv2.imdecode(img, cv2.IMREAD_COLOR)
    return img

# Load the image (replace with your image path or URL)
image_path = "/content/green_channel_images/IDRiD_008_green_channel.jpg"
img = load_image(image_path)

# Check if the image was loaded successfully
if img is None:
    print(f"Error: Unable to load image from {image_path}")
    exit()

# Resize the image to fit within the screen dimensions
screen_width, screen_height = 400, 300
img = cv2.resize(img, (screen_width, screen_height))

# Create a window and set the mouse callback for drawing circles
cv2.namedWindow('image')
cv2.setMouseCallback('image', draw_circle)

# Display the image and keep the window open until 'q' key is pressed
while True:
    cv2.imshow('image', img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Clean up
```

```
cv2.destroyAllWindows()
```

Description

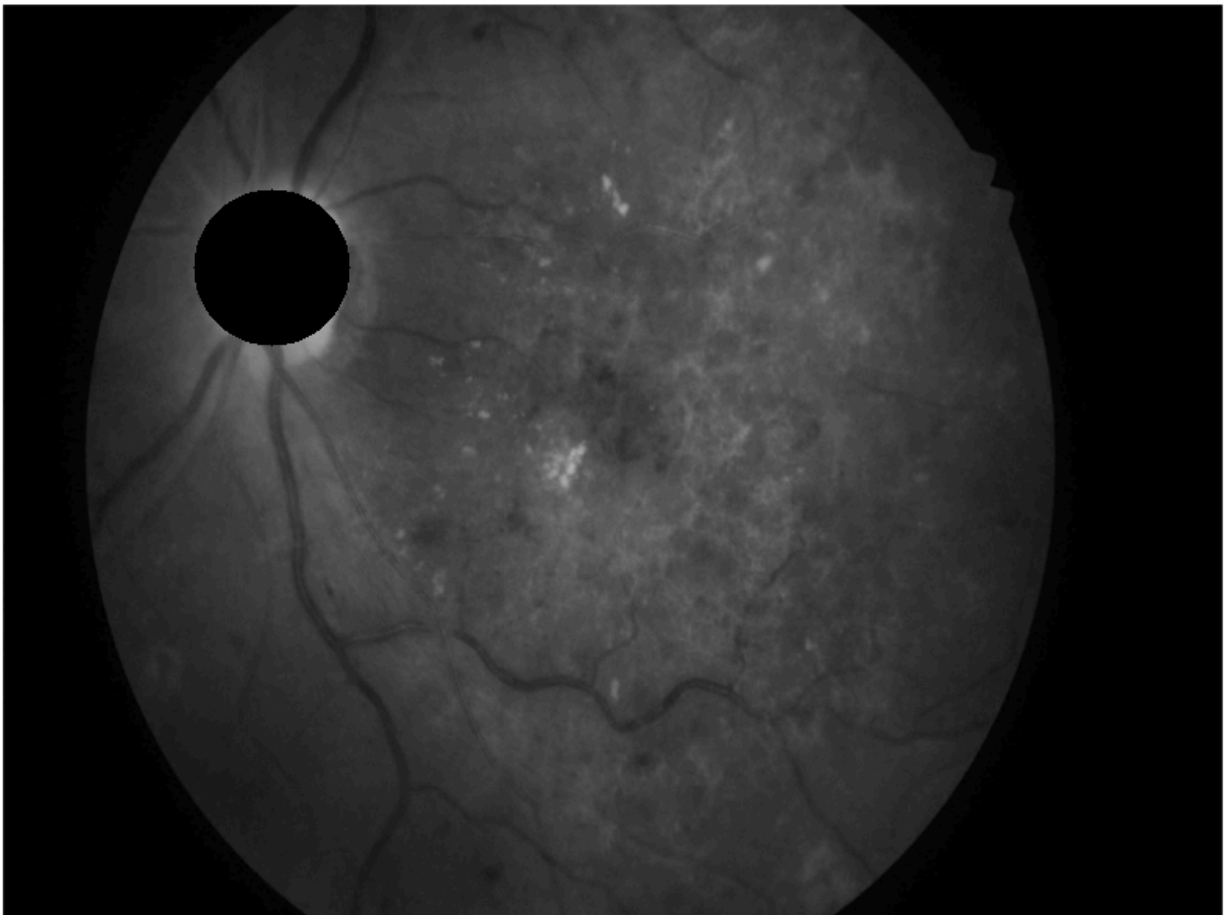
This script allows users to draw black-filled circles on an image by clicking and dragging the mouse. It handles loading images from either a local file path or a URL and supports resizing the image for better visualization within specified screen dimensions.

Key Features

1. **Global Variables:**
    - **drawing:** A flag to indicate whether the mouse is currently pressed and drawing mode is active.
    - ix, iy: Coordinates to store the starting point of the circle when the mouse is pressed.
    - img: The image on which circles will be drawn.
2. **Drawing Function (draw_circle):**
    - Handles mouse events to draw a circle on the image.
    - Mouse Press (cv2.EVENT_LBUTTONDOWN): Starts drawing by capturing the initial coordinates.
    - Mouse Move (cv2.EVENT_MOUSEMOVE): Continuously draws a preview of the circle as the mouse moves, creating a temporary circle that follows the cursor.
    - Mouse Release (cv2.EVENT_LBUTTONUP): Finalizes the circle by drawing it permanently on the image.
3. **Image Loading Function (load_image):**
    - Determines whether to load an image from a local file path or a URL.
    - Uses OpenCV to read the image directly from a local file or via URL using urllib for remote images.
4. **Main Program:**
    - **Load**s the specified image using the load_image function.
    - Checks if the image has been loaded successfully; if not, it displays an error message and exits.
    - Resizes the image to fit within the defined screen dimensions (400x300 pixels).
    - Creates a window to display the image and sets up the mouse callback to enable drawing functionality.
    - Continuously displays the image in a window that remains open until the 'q' key is pressed, allowing the user to interact with the image and draw circles as needed.
5. **Cleanup:**
    - Closes the OpenCV window and releases resources when the user decides to quit by pressing 'q'.

How to Use

- Run the script in an environment where OpenCV is installed and accessible.
- Replace the image_path variable with the desired image file path or URL.
- Click and drag on the image to draw a circle; releasing the mouse button will finalize the circle.
- Press 'q' to exit the image window and terminate the script.



Issues:

YOLOv10 Model Error in Google Colab draw circle window function problem:

While working on a project to remove the optic disc from fundus images and draw a circle around it using the YOLOv10 model in Google Colab, we encountered an error. To address this, we switched to using VS Code IDE.

Reference link:

1.https://roboflow.com/

2..https://www.kaggle.com/datasets

3.https://docs.ultralytics.com/models/yolov10/