Project Documentation

Aim: To detect Exudate in fundus image using machine learning

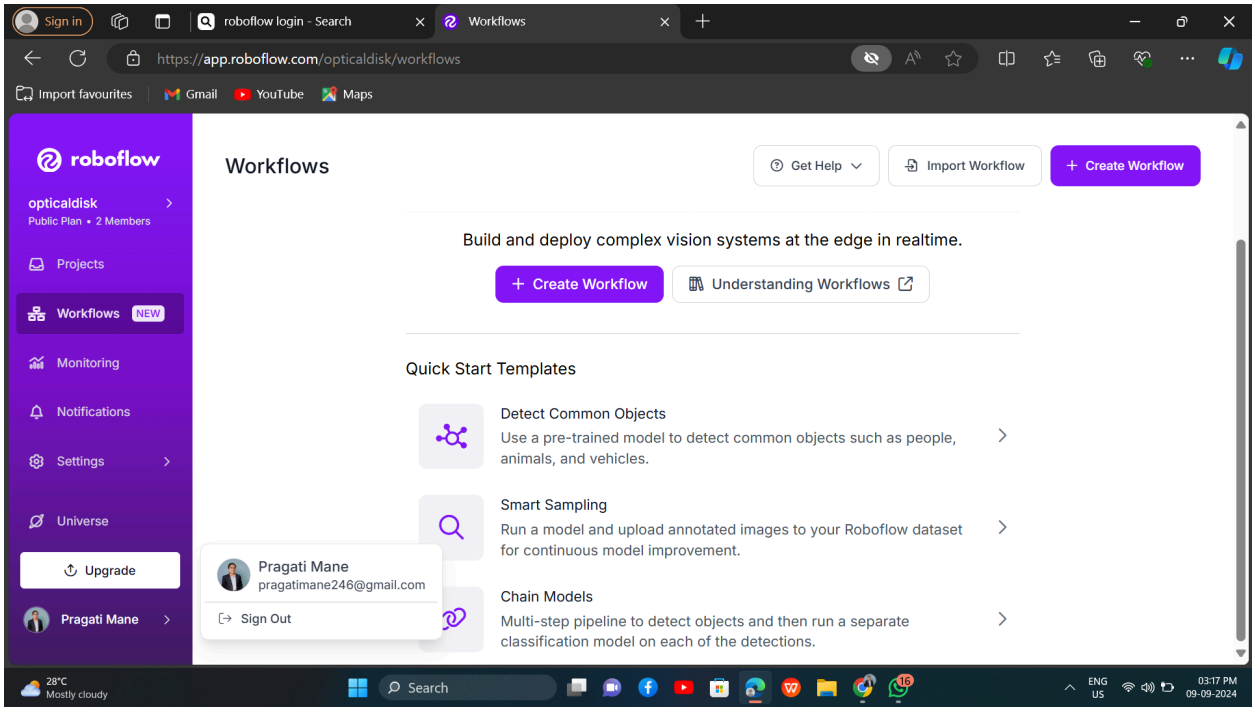
Brief of modules:

Module 1:
Aim : Detect and remove optic disc from fundus image

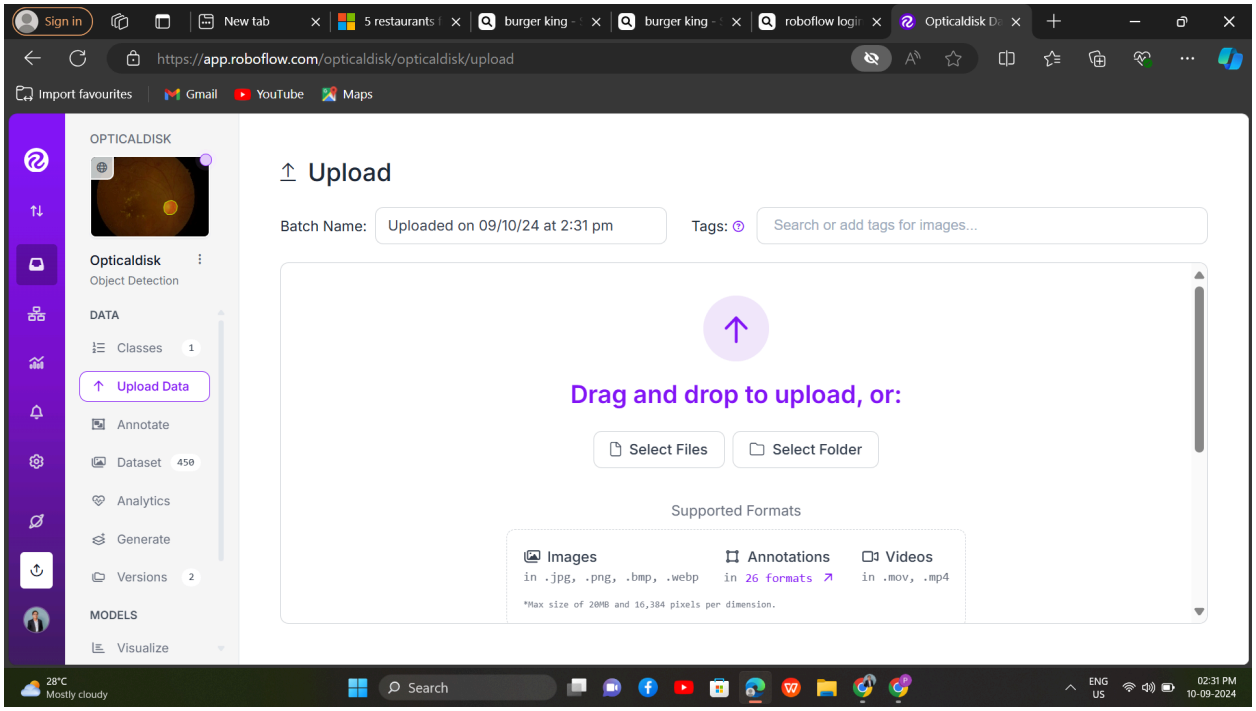
I. Algorithms can be used YOLOv10 and detectron2 to detect optic disc
Ii. Removing optic disc

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.

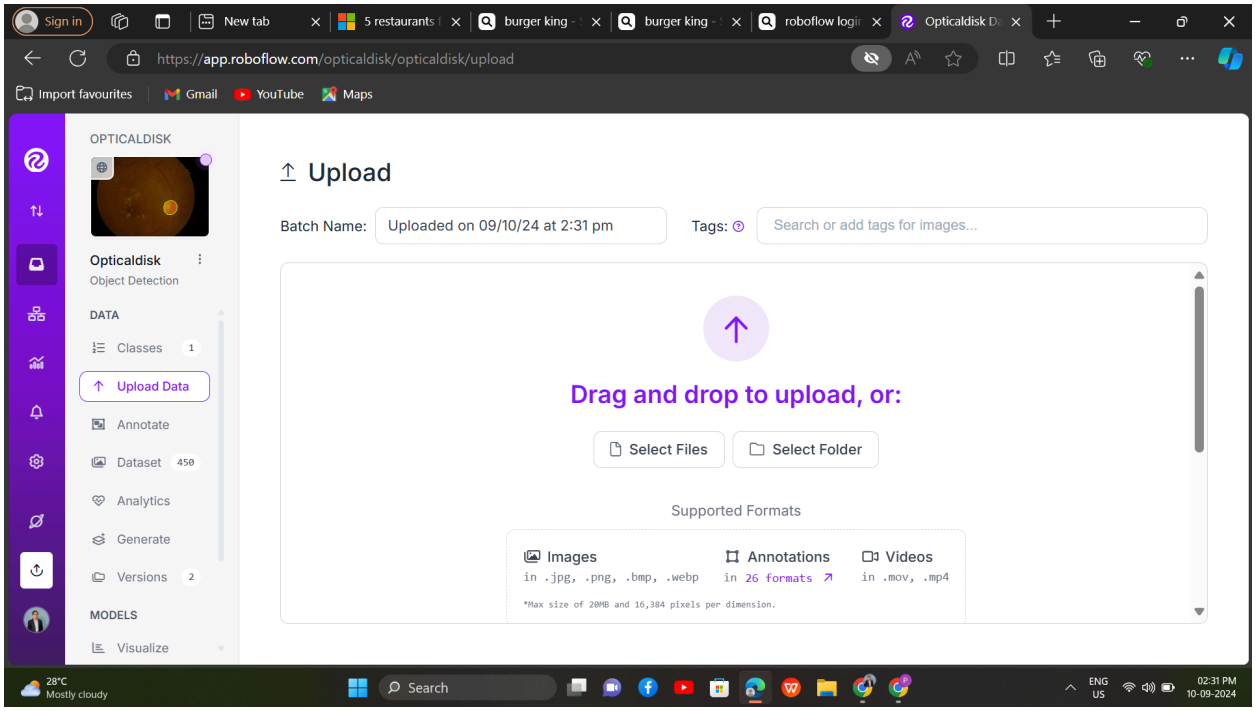


2. Create a New Project
- Click on Create New Project on the dashboard.
- Give your project a name related to diabetic retinopathy (e.g., "Diabetic Retinopathy Detection").
- Choose your task type: Object Detection (since you want to label different types of exudates).
- Select the type of annotation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



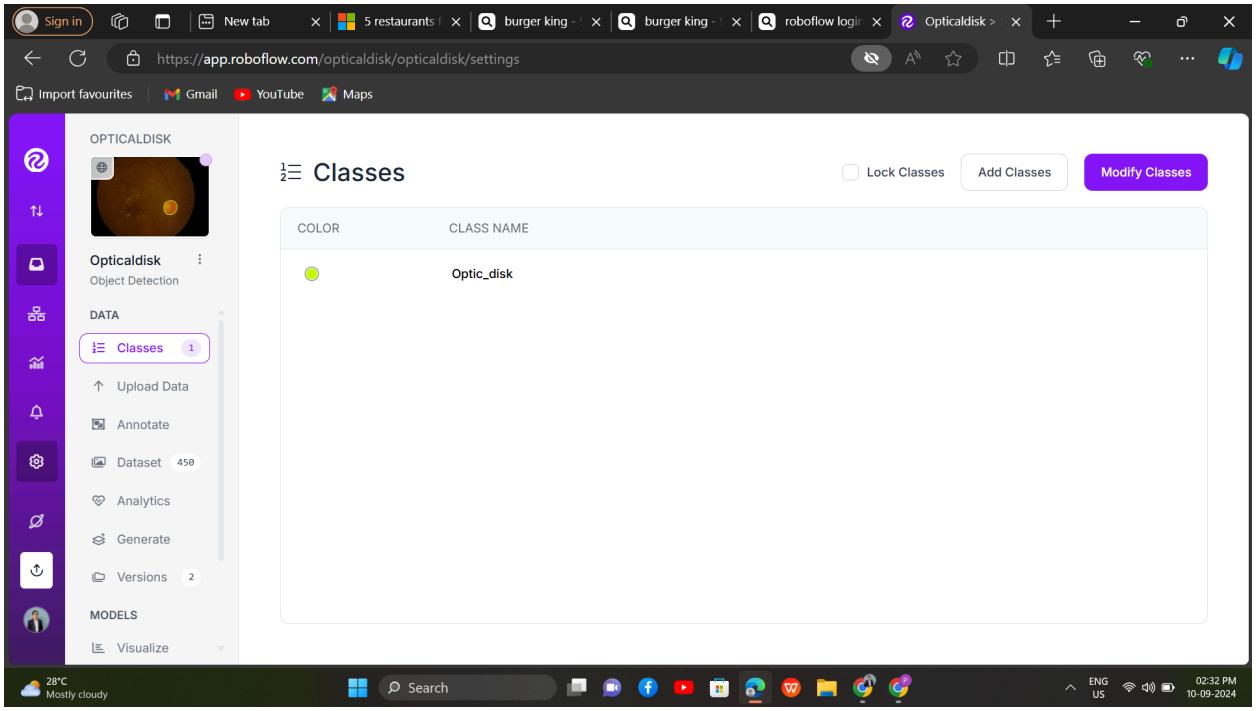
3. Upload Your Images

- 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.



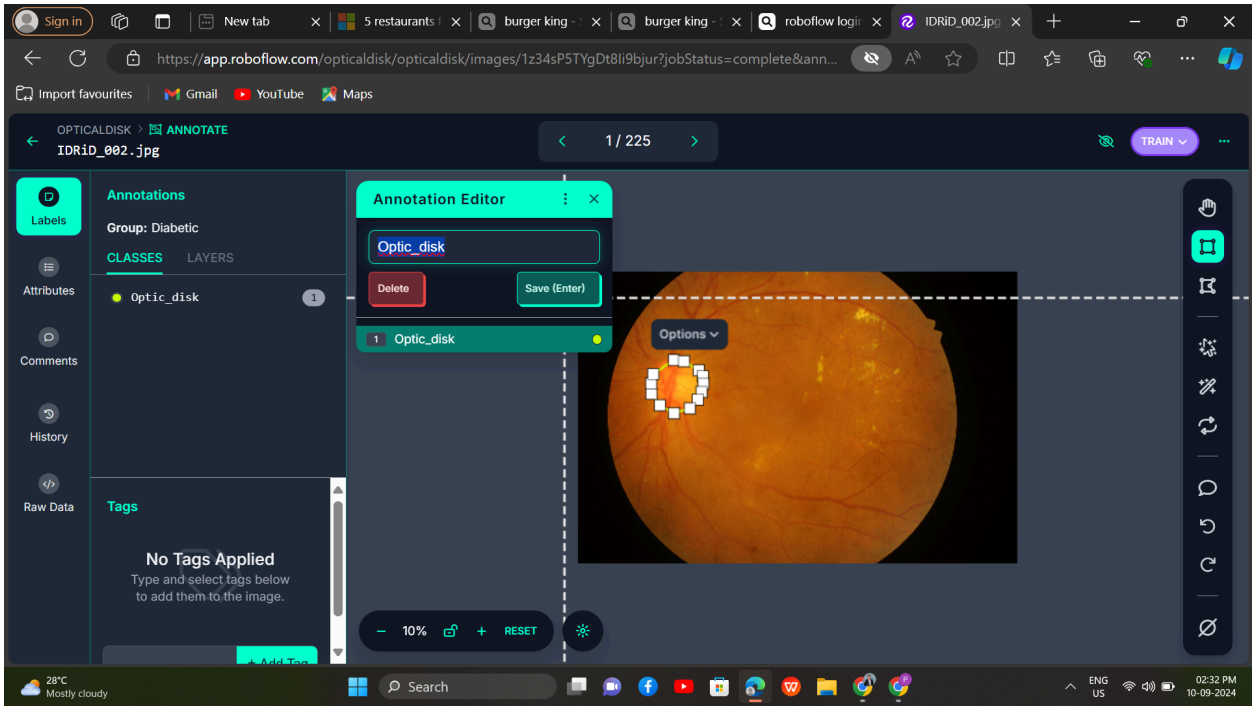
4. Add Classes for Labeling

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

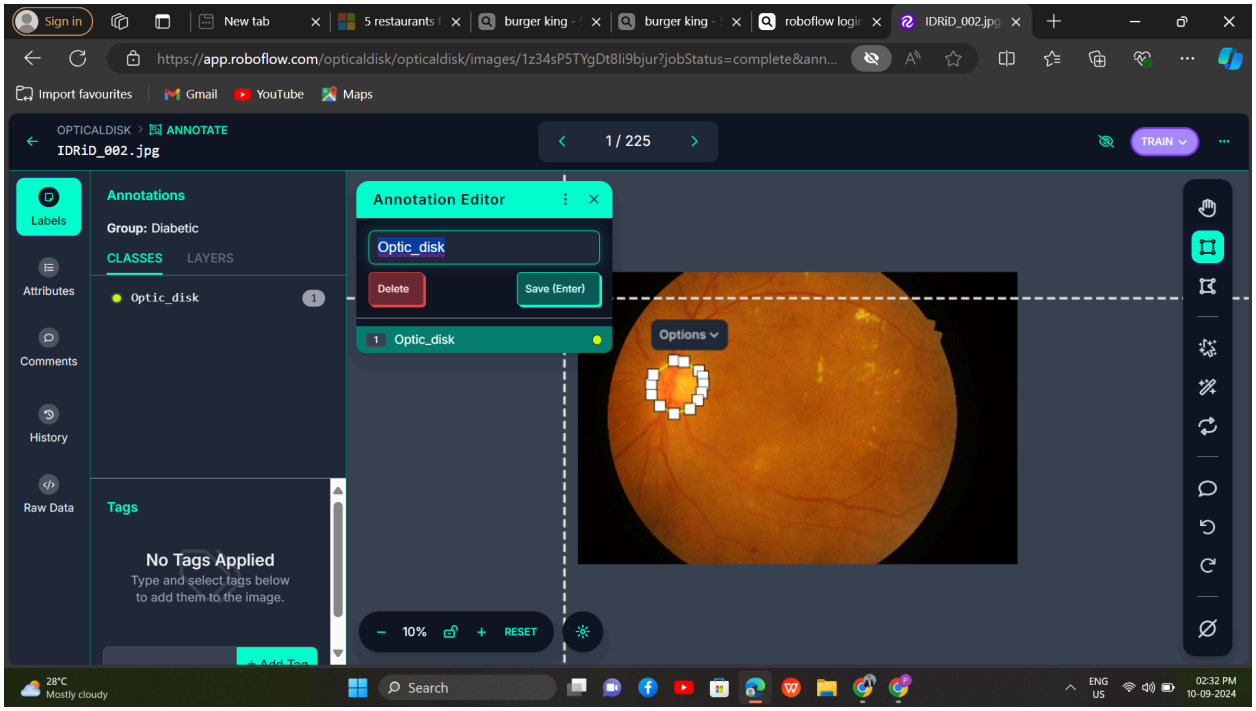


5. Annotate (Label) Your Images

- 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.

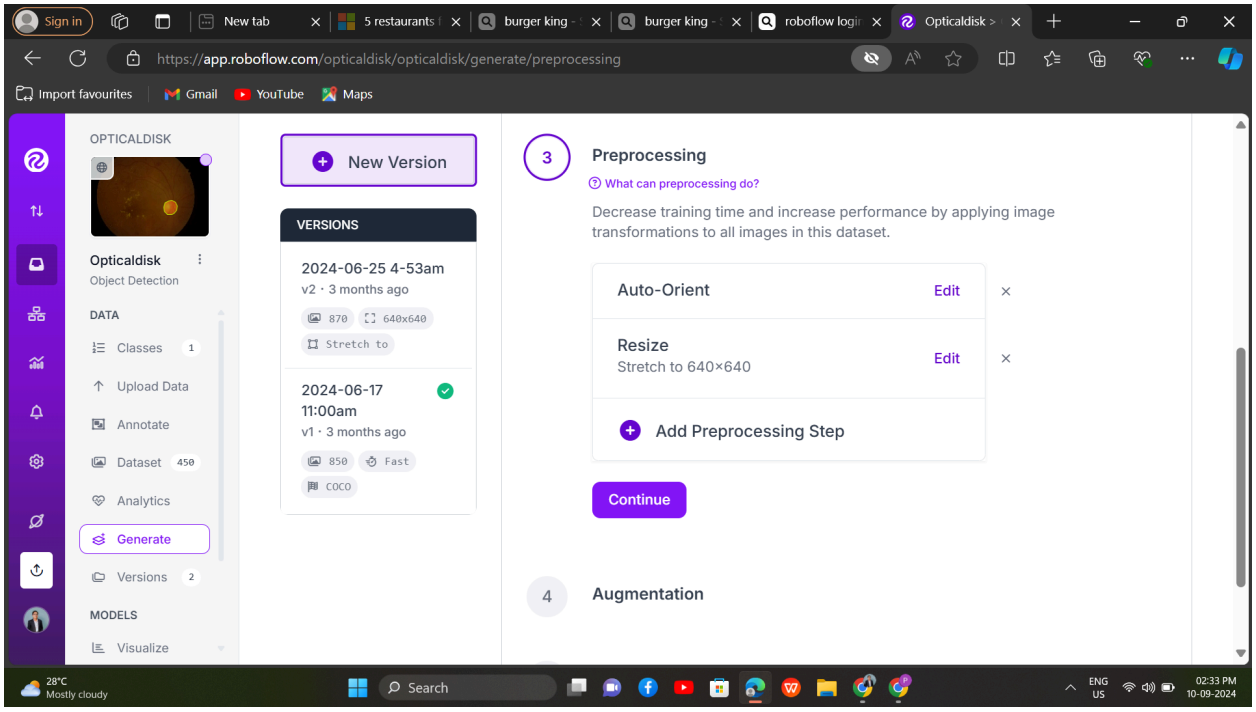


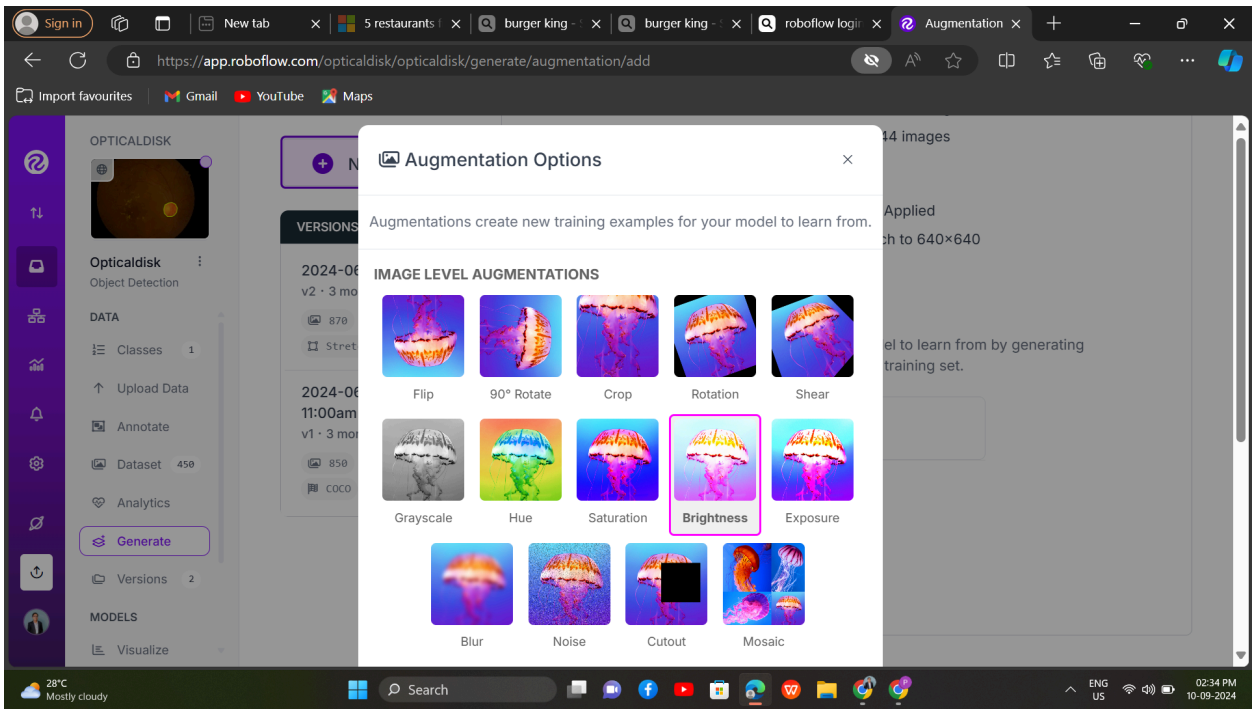
6. Review and Save Annotations
   - Once all your images are annotated, review your work.
   - You can re-adjust bounding boxes if necessary or add additional annotations.
   - After reviewing, save the annotations.



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.

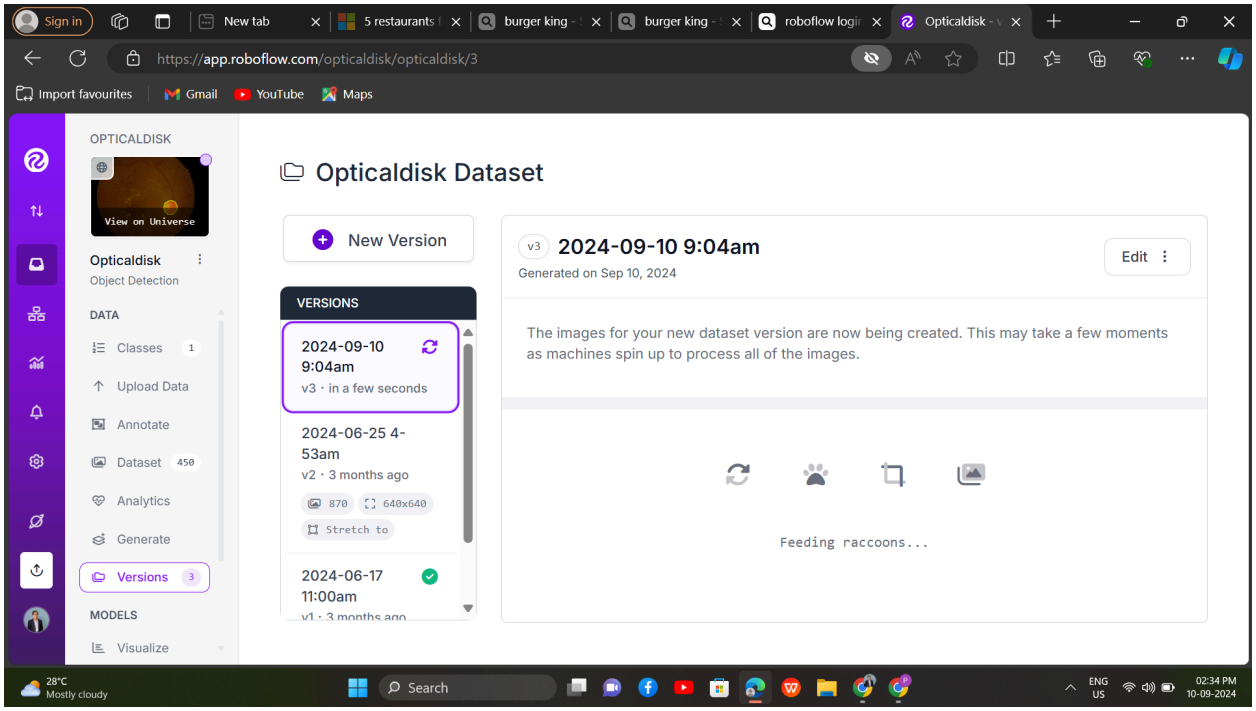


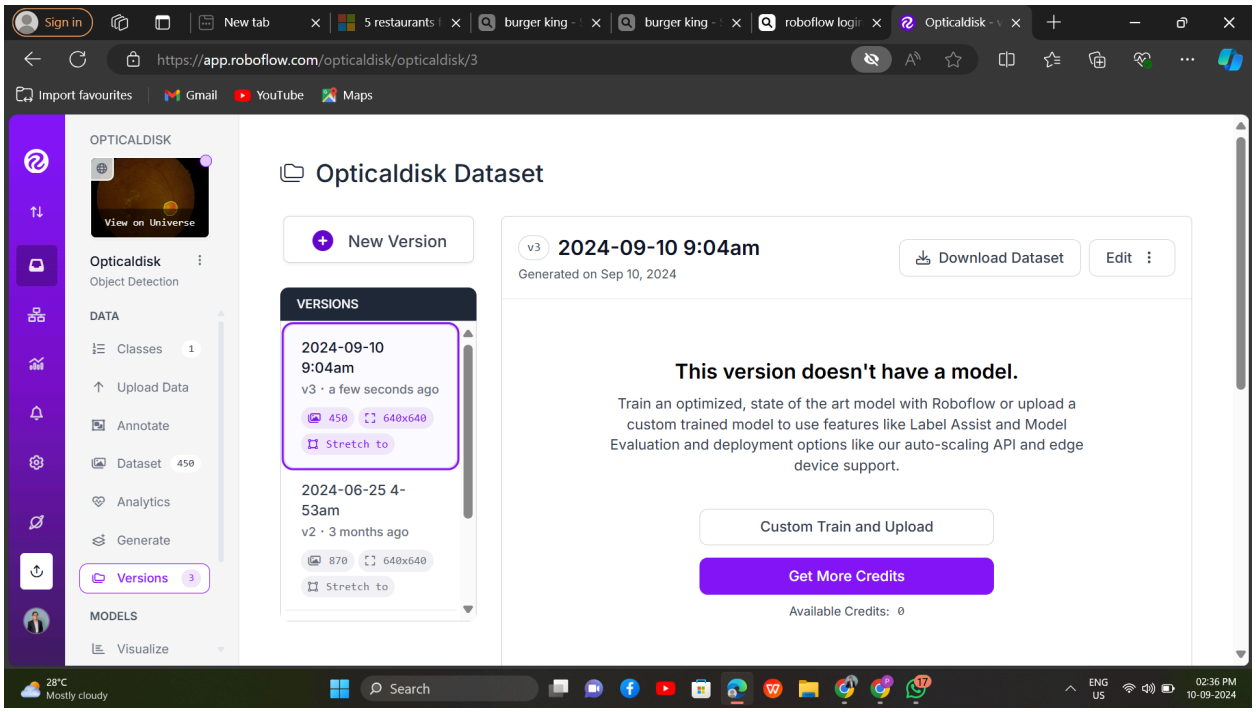


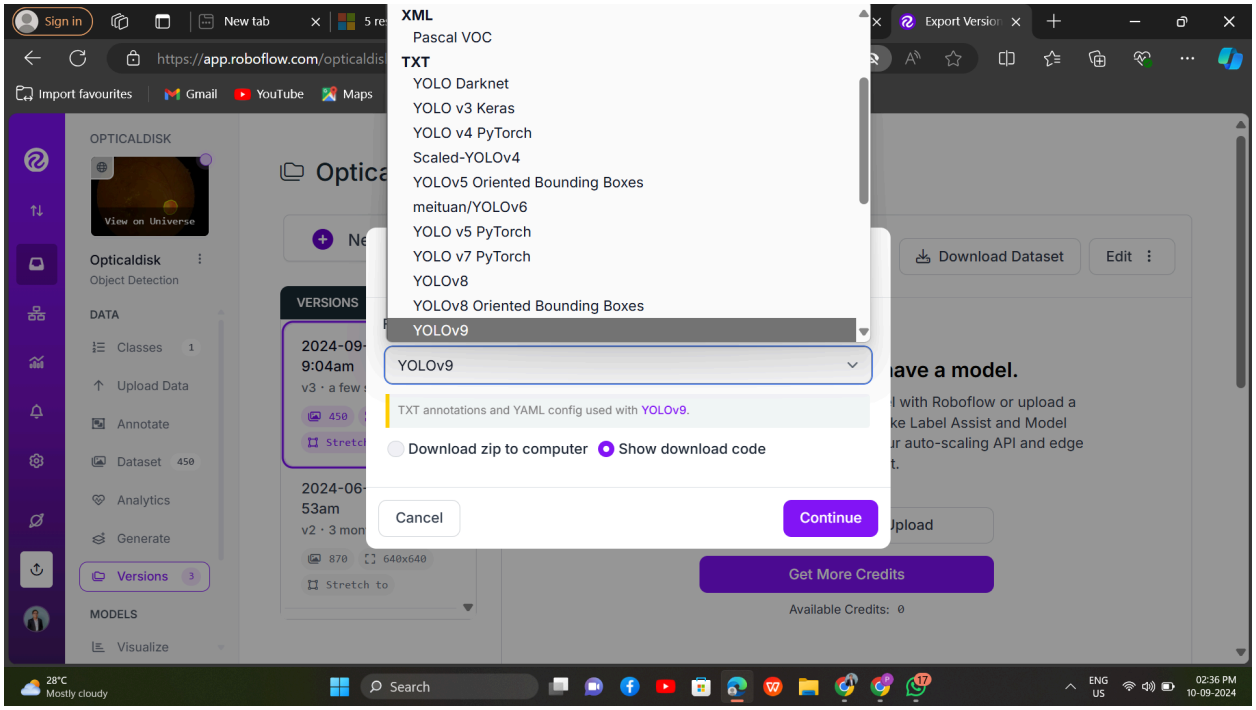
8. Export Dataset

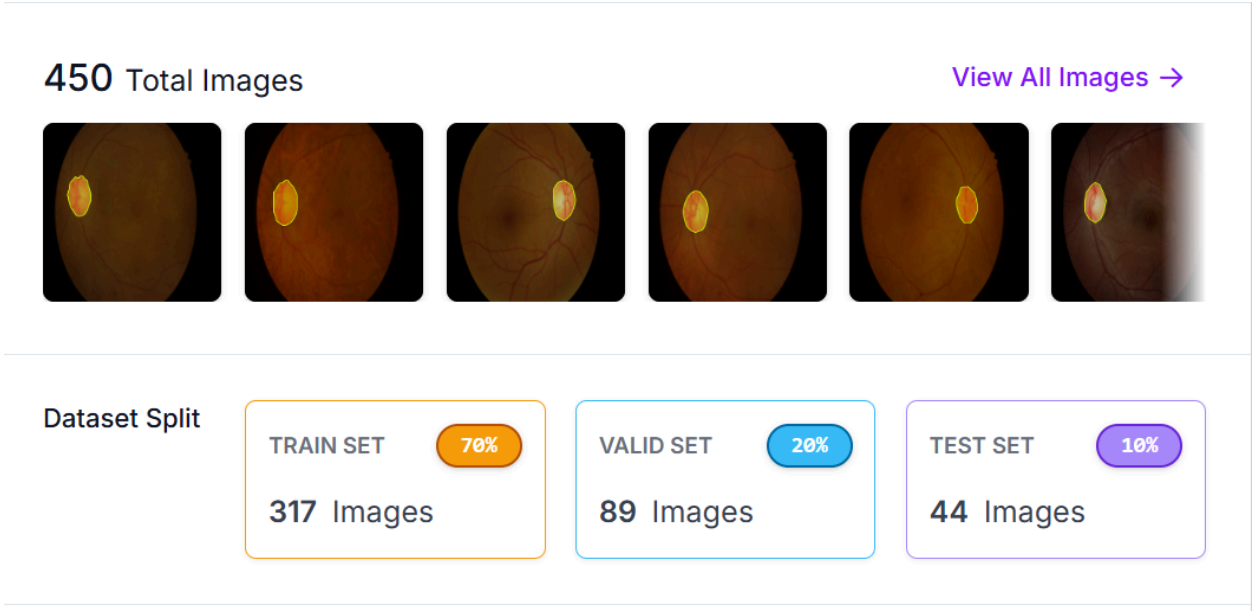
- After completing annotations and preprocessing, you can export the dataset.
- Choose your preferred format, such as TFRecord, COCO JSON, YOLOv5, or others.

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

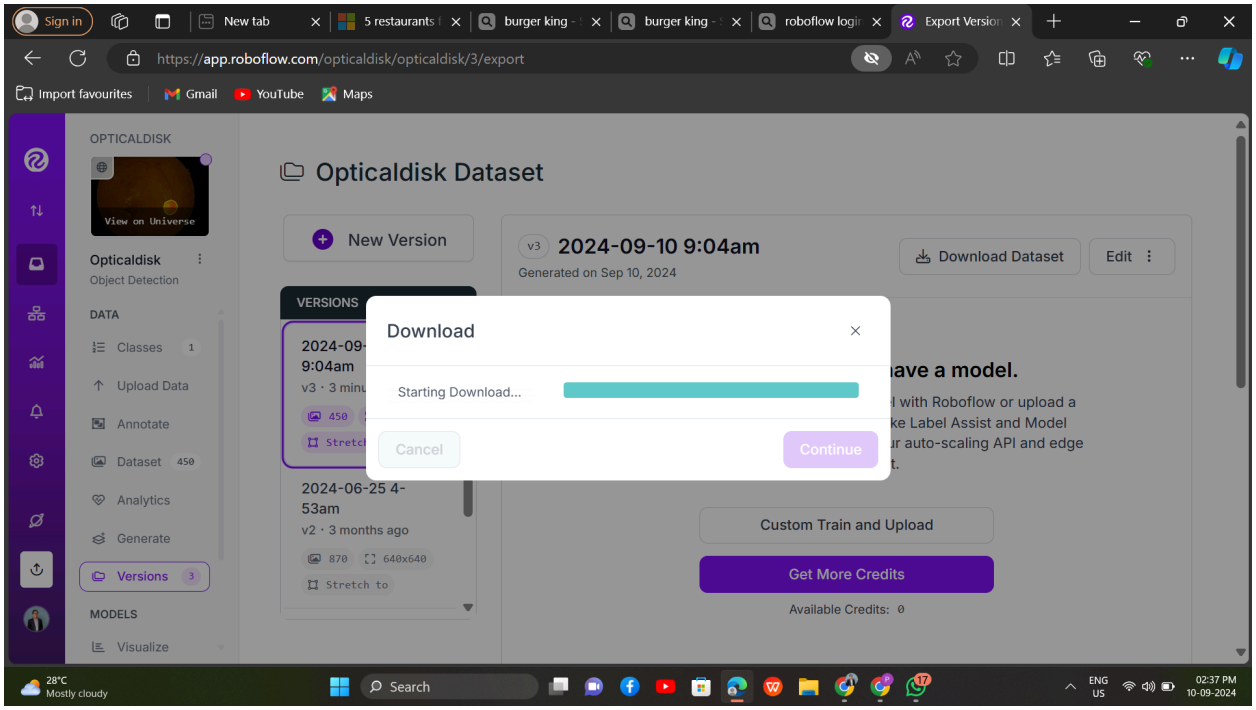








9. Download Labeled Data
   - Once your dataset is exported, download it to your local machine.
   - You will receive a zipped folder containing the images and annotation files in your chosen format.



Reference Link : https://roboflow.com/

Detron 2

Code :

```
# Install PyTorch and torchvision
!pip install torch torchvision

# Install Detectron2
!pip install -U 'git+https://github.com/facebookresearch/detectron2.git'

# Install roboflow
!pip install roboflow
```

Output:

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.4.0+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (0.19.0+cu121)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.15.4)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch) (4.12.2)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.13.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2024.6.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchvision) (1.26.4)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision) (9.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch) (2.1.5)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
Collecting git+https://github.com/facebookresearch/detectron2.git
  Cloning https://github.com/facebookresearch/detectron2.git to /tmp/pip-req-build-vqq4zdhv
  Running command git clone --filter=blob:none --quiet https://github.com/facebookresearch/detectron2.git /tmp/pip-req-build-vqq4zdhv
  Resolved https://github.com/facebookresearch/detectron2.git to commit ebe8b45437f86395352ab13402ba45b75b4d1ddb
  Preparing metadata (setup.py) ... done
Requirement already satisfied: Pillow>=7.1 in /usr/local/lib/python3.10/dist-packages (from detectron2==0.6) (9.4.0)
```

Line 1: Install PyTorch and torchvision

python
Copy code
!pip install torch torchvision

- **Explanation**:
  - torch is the main package for PyTorch, a deep learning framework developed by Facebook's AI Research lab. PyTorch is widely used for its dynamic computation graph and ease of use.
  - torchvision is an extension of PyTorch that includes popular datasets, model architectures, and image transformations for computer vision tasks.
  - PyTorch provides robust support for GPU acceleration, making it possible to perform complex computations efficiently. This is crucial for training large-scale neural networks on massive datasets.

Line 2: Install Detectron2

python
Copy code
!pip install -U 'git+https://github.com/facebookresearch/detectron2.git'

- **Explanation**:
  - -U stands for "upgrade" and ensures that the latest version of the package is installed.
  - git+https://github.com/facebookresearch/detectron2.git installs Detectron2 directly from its GitHub repository. Detectron2 is a popular object detection library developed by Facebook AI Research (FAIR) that includes implementations of various object detection and segmentation algorithms, such as Faster RCNN, Mask RCNN, and more.

Line 3: Install Roboflow

python
Copy code
!pip install roboflow

- **Explanation**:
  - roboflow is a Python package provided by Roboflow, a platform for managing, preprocessing, and augmenting image datasets for computer vision projects. The Roboflow package allows users to easily integrate with their online platform, enabling seamless dataset management and integration into machine learning workflows.

Code :

```
import detectron2
from detectron2.engine import DefaultTrainer
from detectron2.config import get_cfg
from detectron2.data import DatasetCatalog, MetadataCatalog
from detectron2.data.datasets import register_coco_instances
import detectron2.model_zoo as model_zoo
```

Output :

```
Requirement already satisfied: roboflow in /usr/local/lib/python3.10/dist-packages (1.1.44)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from roboflow) (2024.8.30)
Requirement already satisfied: idna==3.7 in /usr/local/lib/python3.10/dist-packages (from roboflow) (3.7)
Requirement already satisfied: cycler in /usr/local/lib/python3.10/dist-packages (from roboflow) (0.12.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.10/dist-packages (from roboflow) (1.4.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from roboflow) (3.7.1)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.10/dist-packages (from roboflow) (1.26.4)
Requirement already satisfied: opencv-python-headless==4.10.0.84 in /usr/local/lib/python3.10/dist-packages (from roboflow) (4.10.0.84)
Requirement already satisfied: Pillow>=7.1.2 in /usr/local/lib/python3.10/dist-packages (from roboflow) (9.4.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from roboflow) (2.8.2)
Requirement already satisfied: python-dotenv in /usr/local/lib/python3.10/dist-packages (from roboflow) (1.0.1)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from roboflow) (2.32.3)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from roboflow) (1.16.0)
Requirement already satisfied: urllib3>=1.26.6 in /usr/local/lib/python3.10/dist-packages (from roboflow) (2.0.7)
Requirement already satisfied: tqdm>=4.41.0 in /usr/local/lib/python3.10/dist-packages (from roboflow) (4.66.5)
Requirement already satisfied: PyYAML>=5.3.1 in /usr/local/lib/python3.10/dist-packages (from roboflow) (6.0.2)
Requirement already satisfied: requests-toolbelt in /usr/local/lib/python3.10/dist-packages (from roboflow) (1.0.0)
Requirement already satisfied: filetype in /usr/local/lib/python3.10/dist-packages (from roboflow) (1.2.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->roboflow) (1.3.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->roboflow) (4.53.1)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->roboflow) (24.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->roboflow) (3.1.4)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->roboflow) (3.3.2)
loading Roboflow workspace...
loading Roboflow project...
Downloading Dataset Version Zip in Opticaldisk-2 to coco:: 100%|██████████| 21658/21658 [00:00<00:00, 31913.45it/s]

Extracting Dataset Version Zip to Opticaldisk-2 in coco:: 100%|██████████| 878/878 [00:00<00:00, 5426.00it/s]/content/Opticaldisk-2
Datasets registered successfully!
```

Line 1: Import Detectron2

python
Copy code

```
import detectron2
```

- **Explanation**: This imports the main Detectron2 library, which is an open-source object detection library developed by Facebook AI Research. Detectron2 provides a modular design that allows for easy experimentation with different models and configurations.

Line 2: Import DefaultTrainer from Detectron2

python
Copy code

```
from detectron2.engine import DefaultTrainer
```

- **Explanation**: This imports the DefaultTrainer class from the detectron2.engine module. DefaultTrainer is a standard training loop provided by Detectron2, which includes functionalities like loading data, setting up the model, and running the training process.

Line 3: Import get_cfg from Detectron2

python

Copy code

```
from detectron2.config import get_cfg
```

- **Explanation**: This imports the get_cfg function from the detectron2.config module. get_cfg is used to create a configuration object that holds all the settings and hyperparameters for training and evaluation. This object can be customized to change various aspects of the training process.

Line 4: Import DatasetCatalog and MetadataCatalog from Detectron2

Python code

```
from detectron2.data import DatasetCatalog, MetadataCatalog
```

- **Explanation**: This imports DatasetCatalog and MetadataCatalog from the detectron2.data module. These are used to register and manage datasets in Detectron2. DatasetCatalog keeps track of the datasets and their information, while MetadataCatalog stores metadata associated with the datasets (such as class names and colors).

Line 5: Import register_coco_instances from Detectron2

python

Copy code

```
from detectron2.data.datasets import register_coco_instances
```

- **Explanation**: This imports the register_coco_instances function from the detectron2.data.datasets module. register_coco_instances is a utility function to register a dataset in COCO format. It simplifies the process of making Detectron2 aware of a dataset by providing paths to the dataset's JSON annotations and image files.

Line 6: Import Detectron2's Model Zoo

python

Copy code

```
import detectron2.model_zoo as model_zoo
```

- **Explanation**: This imports the model_zoo module from Detectron2. The model zoo contains pre-trained models and their configurations that can be used for transfer learning or as a starting

point for training new models. It provides a convenient way to access high-performing models without needing to train them from scratch.

Code & Output :

```
from roboflow import Roboflow
import os

rf = Roboflow(api_key="aj1YNkFeRPD8FAf7DMpN")
project = rf.workspace("opticaldisk").project("opticaldisk")
version = project.version(2)
dataset = version.download("coco")

# Print dataset location
print(dataset.location)

# Register your dataset (adjust paths accordingly)
data_path = dataset.location

register_coco_instances("my_dataset_train", {}, os.path.join(data_path, "train/_annotations.coco.json"), os.path.join(data_path, "train"))
register_coco_instances("my_dataset_val", {}, os.path.join(data_path, "valid/_annotations.coco.json"), os.path.join(data_path, "valid"))
```

```
loading Roboflow workspace...
loading Roboflow project...
/content/Opticaldisk-2
```

Line 1: Import Roboflow

python

Copy code

from roboflow import Roboflow

- **Explanation**: This line imports the Roboflow class from the roboflow package. Roboflow provides tools for managing and preprocessing image datasets.

Line 2: Import the os Module

python

Copy code

import os

- **Explanation**: This line imports the os module, which provides functionalities for interacting with the operating system, such as handling file and directory paths.

Line 3: Initialize Roboflow with API Key

python

Copy code

```
rf = Roboflow(api_key="aj1YNkFeRPD8FAf7DMpN")
```

Explanation: This initializes an instance of the Roboflow class using the provided API key (aj1YNkFeRPD8FAf7DMpN). The API key is necessary to authenticate and access

Line 4: Access a Specific Project

python

Copy code

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

- **Explanation**: This accesses a specific project within a workspace on Roboflow. The workspace method specifies the workspace (named "opticaldisk" in this case), and the project method specifies the project within that workspace, also named "opticaldisk".

  your projects on Roboflow.

Line 5: Get a Specific Version of the Project

python

Copy code

```
version = project.version(2)
```

- **Explanation**: This accesses version 2 of the "opticaldisk" project. Versions allow you to manage different iterations or states of your dataset.

Line 6: Download the Dataset in COCO Format

python

Copy code

```
dataset = version.download("coco")
```

- **Explanation**: This downloads the dataset in the COCO format. The download method takes the desired format ("coco" in this case) as an argument and retrieves the dataset files accordingly.

Line 7: Print Dataset Location

python

Copy code

```
# Print dataset location

print(dataset.location)
```

- **Explanation**: This prints the location where the dataset has been downloaded. The location attribute of the dataset object contains the path to the downloaded dataset files on your local machine or Colab environment.

Line 8: Define the Path to the Dataset

python

Copy code

```
# Register your dataset (adjust paths accordingly)

data_path = dataset.location
```

- **Explanation**: This assigns the location of the downloaded dataset to the variable data_path. This path will be used to register the dataset with Detectron2.

Line 9: Register Training Dataset with Detectron2

python

Copy code

```
register_coco_instances("my_dataset_train", {}, os.path.join(data_path, "train/_annotations.coco.json"), os.path.join(data_path, "train"))
```

- **Explanation**: This registers the training dataset with Detectron2.
    - "my_dataset_train": The name given to the training dataset.
    - {}: An empty dictionary for metadata (which can be populated with additional information if needed).
    - os.path.join(data_path, "train/_annotations.coco.json"): The path to the COCO annotations file for the training data.
    - os.path.join(data_path, "train"): The path to the directory containing the training images.

Line 10: Register Validation Dataset with Detectron2

python

Copy code

```
register_coco_instances("my_dataset_val", {}, os.path.join(data_path, "valid/_annotations.coco.json"), os.path.join(data_path, "valid"))
```

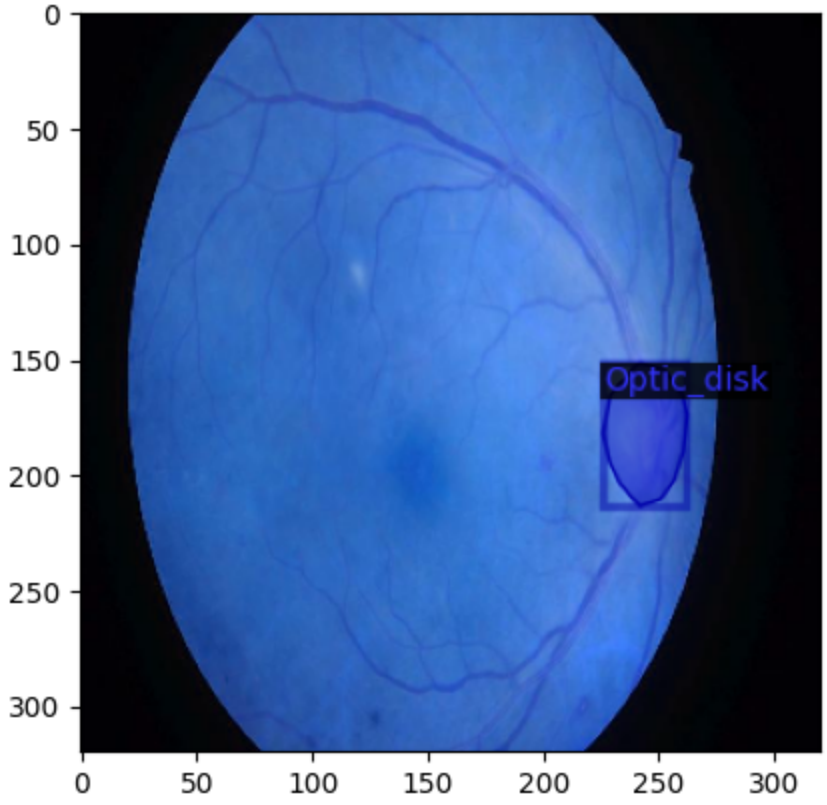
- **Explanation**: This registers the validation dataset with Detectron2.
    - "my_dataset_val": The name given to the validation dataset.
    - {}: An empty dictionary for metadata (which can be populated with additional information if needed).
    - os.path.join(data_path, "valid/_annotations.coco.json"): The path to the COCO annotations file for the validation data.
    - os.path.join(data_path, "valid"): The path to the directory containing the validation images.

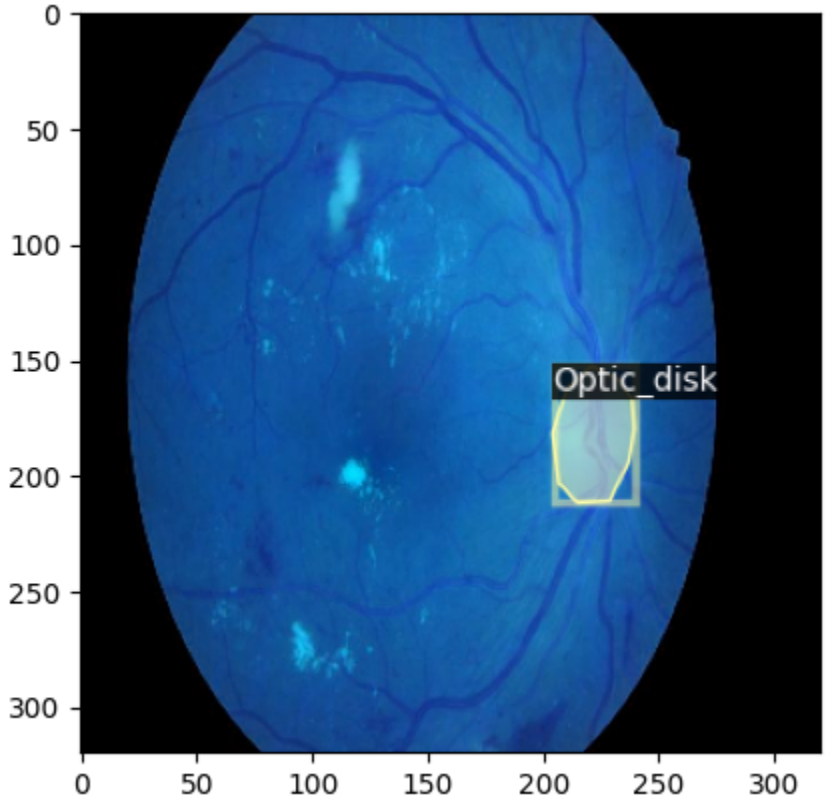
Code :

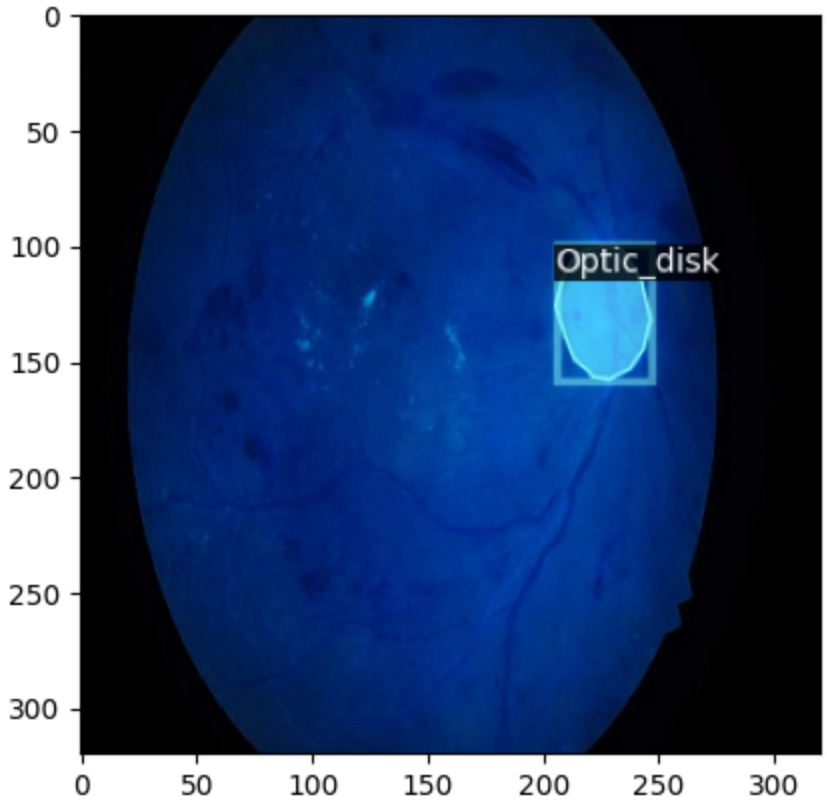
```python
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog, DatasetCatalog

metadata = MetadataCatalog.get("my_dataset_train")
dataset_dicts = DatasetCatalog.get("my_dataset_train")
import random
import cv2
import matplotlib.pyplot as plt

for d in random.sample(dataset_dicts, 3):
    img = cv2.imread(d["file_name"])
    visualizer = Visualizer(img[:, :, ::1], metadata=metadata, scale=0.5)
    vis = visualizer.draw_dataset_dict(d)
    plt.imshow(vis.get_image()[:, :, ::1])
    plt.show()
```

Output :

```
WARNING:detectron2.data.datasets.coco:
Category ids in annotations are not in [1, #categories]! We'll apply a mapping for you.
```


Optic_disk
0
50
100
150
200
250
300


Optic_disk
0
50
100
150
200
250
300



Line 1: Import Visualizer from Detectron2

python

Copy code

```
from detectron2.utils.visualizer import Visualizer
```

- **Explanation**: This line imports the Visualizer class from the detectron2.utils.visualizer module. The Visualizer class is used to visualize annotations and predictions on images.

Line 2: Import MetadataCatalog and DatasetCatalog from Detectron2

python

Copy code

```
from detectron2.data import MetadataCatalog, DatasetCatalog
```

- **Explanation**: This line imports MetadataCatalog and DatasetCatalog from the detectron2.data module. These catalogs are used to access metadata (such as class names) and registered datasets, respectively.

Line 3: Get Metadata for the Training Dataset

python

Copy code

```
metadata = MetadataCatalog.get("my_dataset_train")
```

- **Explanation**: This line retrieves metadata associated with the training dataset named "my_dataset_train" from the MetadataCatalog. Metadata typically includes information like class names and colors.

Line 4: Get Dataset Dictionary for the Training Dataset

python

Copy code

```
dataset_dicts = DatasetCatalog.get("my_dataset_train")
```

- **Explanation**: This line retrieves the dataset dictionary (dataset_dicts) for the training dataset named "my_dataset_train" from the DatasetCatalog. Each item in dataset_dicts contains information about one image and its annotations.

Line 5: Import Necessary Libraries (random, cv2, matplotlib.pyplot)

python

Copy code

```
import random

import cv2

import matplotlib.pyplot as plt
```

- **Explanation**: This imports additional Python libraries:
  - random: Used here to randomly sample 3 images from the dataset for visualization.
  - cv2: OpenCV library for image processing tasks, such as reading images (cv2.imread).
  - matplotlib.pyplot as plt: Matplotlib library for plotting and visualization.

Line 6-12: Iterate Over Randomly Sampled Dataset Items and Visualize

python

Copy code

```
for d in random.sample(dataset_dicts, 3):

    img = cv2.imread(d["file_name"])

    visualizer = Visualizer(img[:, :, ::1], metadata=metadata, scale=0.5)

    vis = visualizer.draw_dataset_dict(d)

    plt.imshow(vis.get_image()[:, :, ::1])

    plt.show()
```

- **Explanation**:
  - for d in random.sample(dataset_dicts, 3)::
    - Iterates over a random sample of 3 items (d) from dataset_dicts.
  - img = cv2.imread(d["file_name"]):
    - Reads the image corresponding to the current dataset item (d) using OpenCV (cv2.imread). d["file_name"] contains the file path of the image.
  - visualizer = Visualizer(img[:, :, ::1], metadata=metadata, scale=0.5):

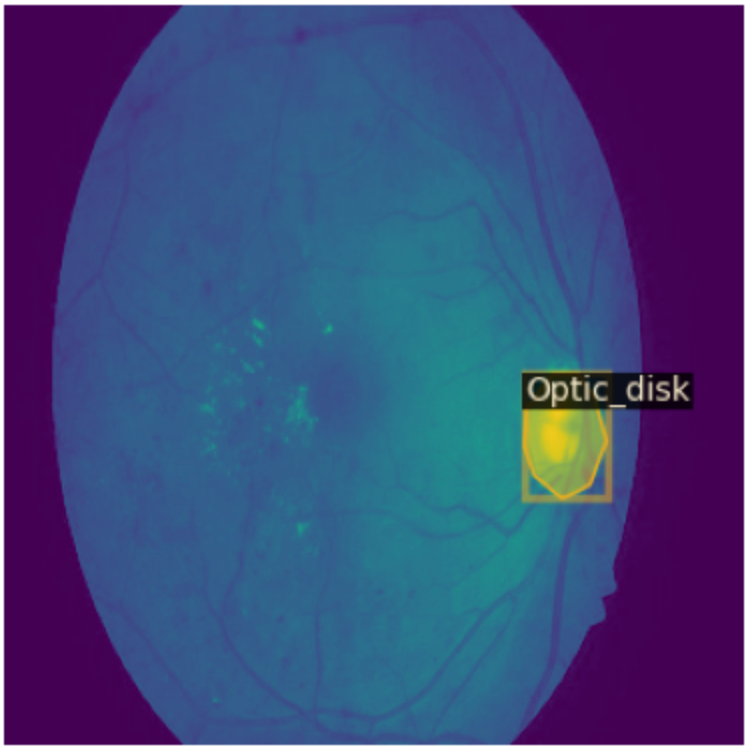
        - Initializes a Visualizer object (visualizer) with the image (img[:, :, ::1] to convert BGR to RGB), metadata (metadata), and scale (0.5 scales the image for visualization).
    - vis = visualizer.draw_dataset_dict(d):
        - Draws annotations (d) on the image using the draw_dataset_dict method of the Visualizer object.
    - plt.imshow(vis.get_image()[:, :, ::1]):
        - Displays the annotated image using Matplotlib (plt.imshow). vis.get_image()[:, :, ::1] converts the image back to RGB format for display.
    - plt.show():
        - Shows the annotated image plot.

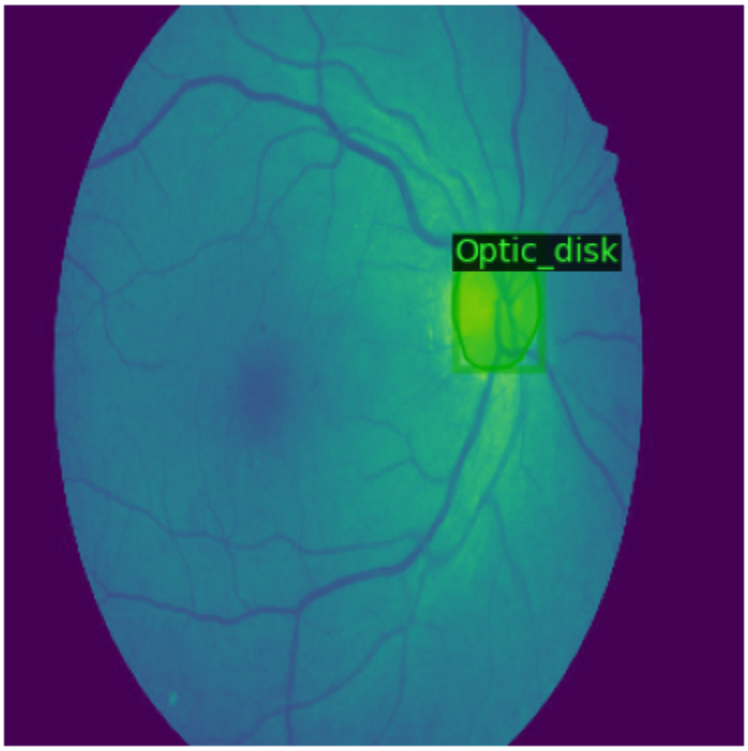
Code :

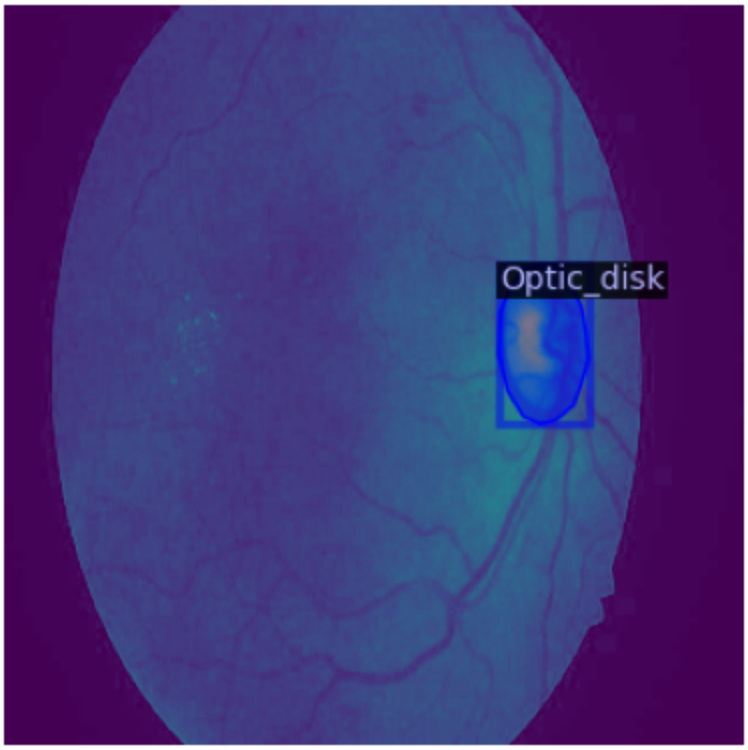
```
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog, DatasetCatalog

metadata = MetadataCatalog.get("my_dataset_train")
dataset_dicts = DatasetCatalog.get("my_dataset_train")

import random
import cv2
import matplotlib.pyplot as plt
import numpy as np

for d in random.sample(dataset_dicts, 3):
    img = cv2.imread(d["file_name"])

    # Extract green channel
    green_channel = img[:, :, 1]  # Index 1 corresponds to green channel in BGR

    # Create a visualizer with the green channel image
    visualizer = Visualizer(green_channel, metadata=metadata, scale=0.5)

    # Draw annotations on the green channel image
    vis = visualizer.draw_dataset_dict(d)

    # Display the green channel image with annotations
    plt.imshow(vis.get_image(), cmap='gray')  # Use gray colormap to display green channel
    plt.axis('off')
    plt.show()
```

Output :

```
WARNING:detectron2.data.datasets.coco:
Category ids in annotations are not in [1, #categories]! We'll apply a mapping for you.
```


Optic_disk


Optic_disk



Line 1: Import Visualizer from Detectron2

python

Copy code

```
from detectron2.utils.visualizer import Visualizer
```

- **Explanation**: This line imports the Visualizer class from the detectron2.utils.visualizer module. The Visualizer class is used to visualize annotations and predictions on images.

Line 2: Import MetadataCatalog and DatasetCatalog from Detectron2

python

Copy code

```
from detectron2.data import MetadataCatalog, DatasetCatalog
```

- **Explanation**: This line imports MetadataCatalog and DatasetCatalog from the detectron2.data module. These catalogs are used to access metadata (such as class names) and registered datasets, respectively.

Line 4: Get Metadata for the Training Dataset

python

Copy code

```
metadata = MetadataCatalog.get("my_dataset_train")
```

- **Explanation**: This line retrieves metadata associated with the training dataset named "my_dataset_train" from the MetadataCatalog. Metadata typically includes information like class names and colors.

Line 5: Get Dataset Dictionary for the Training Dataset

python

Copy code

```
dataset_dicts = DatasetCatalog.get("my_dataset_train")
```

- **Explanation**: This line retrieves the dataset dictionary (dataset_dicts) for the training dataset named "my_dataset_train" from the DatasetCatalog. Each item in dataset_dicts contains information about one image and its annotations.

Line 7-12: Iterate Over Randomly Sampled Dataset Items and Visualize

python

Copy code

```
import random

import cv2

import matplotlib.pyplot as plt

import numpy as np

for d in random.sample(dataset_dicts, 3):

    img = cv2.imread(d["file_name"])

    # Extract green channel

    green_channel = img[:, :, 1]  # Index 1 corresponds to green channel in BGR

    # Create a visualizer with the green channel image

    visualizer = Visualizer(green_channel, metadata=metadata, scale=0.5)

    # Draw annotations on the green channel image

    vis = visualizer.draw_dataset_dict(d)

    # Display the green channel image with annotations

    plt.imshow(vis.get_image(), cmap='gray')  # Use gray colormap to display green channel

    plt.axis('off')

    plt.show()
```

- **Explanation**:
    - import random, cv2, matplotlib.pyplot as plt, numpy as np:
        - Imports necessary libraries:
            - random: For random sampling of dataset items.
            - cv2: OpenCV library for image processing tasks, such as reading images (cv2.imread).
            - matplotlib.pyplot as plt: Matplotlib library for plotting and visualization.
            - numpy as np: NumPy library for numerical operations.
    - for d in random.sample(dataset_dicts, 3)::

        - Iterates over a random sample of 3 items (d) from dataset_dicts.
    - img = cv2.imread(d["file_name"]):
        - Reads the image corresponding to the current dataset item (d) using OpenCV (cv2.imread). d["file_name"] contains the file path of the image.
    - green_channel = img[:, :, 1]:
        - Extracts the green channel from the BGR image (img). In OpenCV, color channels are ordered as BGR (Blue, Green, Red), so img[:, :, 1] selects the green channel.
    - visualizer = Visualizer(green_channel, metadata=metadata, scale=0.5):
        - Initializes a Visualizer object (visualizer) with the green channel image (green_channel), metadata (metadata), and a scale of 0.5 for visualization.
    - vis = visualizer.draw_dataset_dict(d):
        - Draws annotations (d) on the green channel image using the draw_dataset_dict method of the Visualizer object.
    - plt.imshow(vis.get_image(), cmap='gray'):
        - Displays the annotated green channel image using Matplotlib (plt.imshow). vis.get_image() retrieves the annotated image, and cmap='gray' specifies the grayscale colormap for display.
    - plt.axis('off'):
        - Turns off the axis (i.e., hides the axis) for the plot.
    - plt.show():
        - Displays the plot of the annotated green channel image.

Code :

```
import os

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

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

Output:

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

Line 1 : Import the `os` Module

import os

```
# Define the folder path
output_folder = "/content/green_channel_image"

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

- `import os` allows you to use functions from the `os` module, which provides a way to interact with the operating system, including file and directory operations.

Line 2 : Define the Folder Path

- `output_folder = "/content/green_channel_image"`: This sets a variable `output_folder` to the path where you want to create or check for the folder. In this case, the path is `"/content/green_channel_image"`.

Line 3 : Create the Folder if It Doesn't Exist:

- `os.makedirs(output_folder, exist_ok=True)`: This line creates the directory specified by `output_folder`. The `exist_ok=True` argument tells the function not to raise an error if the directory already exists. If `exist_ok` were set to `False` (which is the default), it would raise a `FileExistsError` if the directory already exists.

Line 4 : Print Confirmation:

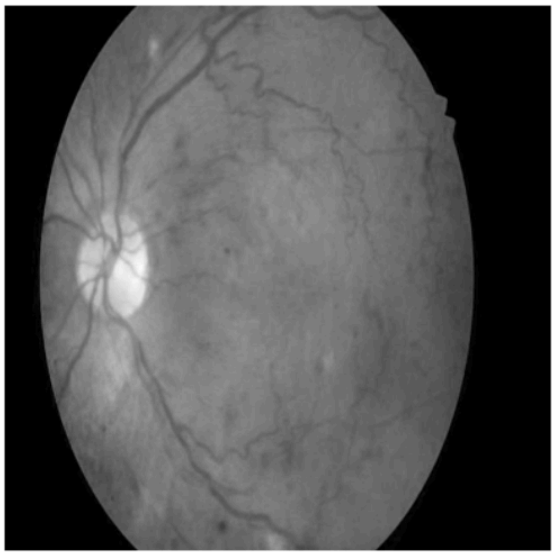
- `print(f"Folder '{output_folder}' created or already exists.")`: This prints a message to the console confirming that the folder has been created or that it already existed.

Code:

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

# Load the original right image
image_path = "/content/Opticaldisk-2/test/IDRiD_007test_jpg.rf.f818958ddd0d4f243d94e074a1fe7deb.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_image"  # 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:

```
Green channel image saved at: /content/green_channel_image/IDRiD_008_green_channel.jpg
```



Line 1 : Import Libraries

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

- `import cv2`: Imports OpenCV, a library used for image processing tasks.
- `import matplotlib.pyplot as plt`: Imports Matplotlib, a plotting library used to display images.
- `import os`: Imports the OS module for file and directory operations.

Line 2 : Load the Original Image

```
# Load the original right image
image_path =
"/content/Opticaldisk-2/test/IDRiD_007test_jpg.rf.f818958ddd0d4f243d94e074a1fe7deb.jpg"
right_image = cv2.imread(image_path)
```

- `image_path = "/content/Opticaldisk-2/test/IDRiD_007test_jpg.rf.f818958ddd0d4f243d94e074a1fe7deb.jpg"`: Defines the path to the image file you want to process.
- `right_image = cv2.imread(image_path)`: Uses OpenCV's `imread` function to load the image from the specified path into the variable `right_image`. This image is loaded as a NumPy array.

Line 3 : Extract the Green Channel

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

- `green_channel = right_image[:, :, 1]`: Extracts the green channel from the image. Images in OpenCV are represented in BGR (Blue, Green, Red) format by default. The `[:, :, 1]` notation selects the green channel.

Line 4 : Specify the Output Folder

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

- `output_folder = "/content/green_channel_image"`: Sets the path where the green channel image will be saved.

Line 5 : Save the Green Channel Image

```
# 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)
```

- `green_image_path = os.path.join(output_folder, "IDRiD_008_green_channel.jpg")`: Creates the full path for the output image by joining the `output_folder` path with the desired filename.
- `cv2.imwrite(green_image_path, green_channel)`: Saves the green channel image to the specified path using OpenCV's `imwrite` function.

Line 6 : Print Confirmation

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

- `print(f"Green channel image saved at: {green_image_path}")`: Prints a message to the console indicating where the green channel image has been saved.

Line 7 : Display the Green Channel Image

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

- `plt.figure(figsize=(5, 5))`: Creates a new figure with a size of 5x5 inches.
- `plt.imshow(green_channel, cmap='gray')`: Displays the green channel image using Matplotlib with a grayscale colormap.
- `plt.axis('off')`: Hides the axis labels and ticks from the plot.
- `plt.show()`: Renders the plot and displays the green channel image.

List of issues

List of issues faced during the development process how they are tackled whether they are resolved or not .

1.Duplicate Installation Commands:

- Problem: Multiple installation commands for roboflow were present.
- Resolution: Removed the redundant installation commands to streamline the setup process.

2.Dataset Download Path:

- Problem: Incorrect dataset paths or unexpected dataset structure might cause issues.
- Resolution: Added checks to print and verify the dataset location and file paths. Ensured that paths matched the dataset's structure by inspecting the directory contents.

3.Dataset Registration:

- Problem: Potential conflicts if the dataset was already registered.
- Resolution: Checked if the dataset was already registered and removed it if necessary before re-registering with unique names.

4.Annotation File Paths:

- Problem: The script might fail if the annotation files were not found at the expected paths.
- Resolution: Added checks to verify the existence of annotation files and raised informative errors if they were missing.

5.Visualization of Green Channel Images:

- Problem: Green channel images might not display correctly with annotations due to their single-channel nature.
- Resolution: Used a grayscale colormap (cmap='gray') for displaying green channel images to handle the single-channel nature appropriately.

6.Mouse Event Handling for Drawing Circles:

- Problem: Mouse events for drawing circles on images might not function as expected in different environments.
- Resolution: Provided a mechanism to load images from either local paths or URLs. Included resizing of the image to fit screen dimensions for better visualization.

7.Matplotlib and OpenCV Integration:

- Problem: Combining Matplotlib with OpenCV for interactive features might have compatibility issues.
- Resolution: Used Matplotlib to display images and annotations in a way compatible with OpenCV's image manipulation.

8.File Path and Directory Creation:

- Problem: Folder creation for saving images might fail due to permission issues or incorrect paths.
- Resolution: Added checks to create directories if they did not exist and printed confirmation messages.

9.Library Version Conflicts:

- Problem: Potential conflicts between different library versions (e.g., detectron2, torch, torchvision).
- Resolution: Ensured installation of compatible versions by specifying the necessary library versions or using the latest versions.

10.Error Handling and Debugging:

- Problem: Potential runtime errors due to incorrect assumptions about file paths or data formats.
- Resolution: Included error handling and debugging statements to check file existence and correct data formats before proceeding with further processing.