Submitted by – Mohammad Farhan

Challenge 1: The challenge requires to write a solution that separates the circular objects in the foreground. Displaying the mask and the image in the same display window.

- Create endpoint to upload image and store into persistent storage.
- Assign unique reference identifier to each circular object.
- Create endpoint to retrieve list of all circular objects (id and bounding box) for queried image.
- Create endpoint to find bounding box, centroid and radius for queried circular object.
- Create model(algo) evaluation strategy.
- The solution should be based in Python.
- Containerize the solution.

Sample Image:

Solution:

1> Developed a flask-based application. It consists of 3 requests.

2> First request is just to check if the application is up and running. Second is the post request to upload the image. Third is GET request to perform all the required steps.

3> Containerized the application and hosted on localhost:5000 port.

4> Used Python for development and Postman for testing the application.

Details explanation of steps.

→ **Upload of image:**

Created a variable containing folder name i.e ‘uploads ’. The uploaded images are getting saved into this folder.

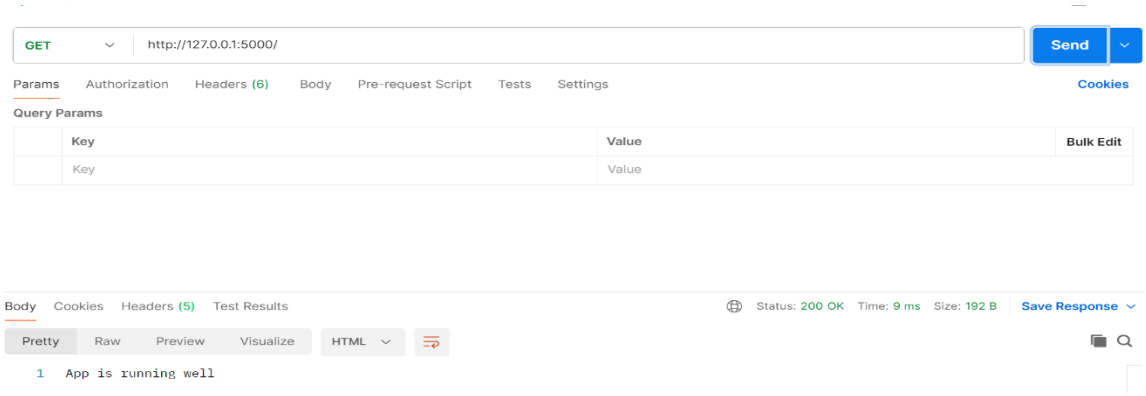
Handled as many exceptions in limited time frame.

Any image getting uploaded in the ‘upload’ folder is getting renamed as ' circle_coin.jpg'.

When the application is Containerized persistent storage is maintained for the images.

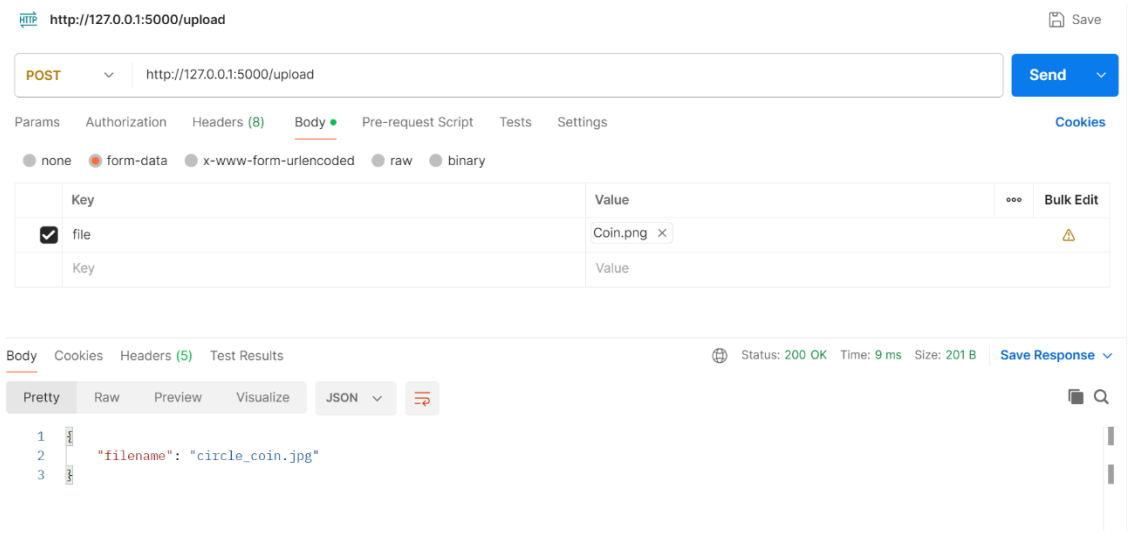
Testing of the above with screen shots:

- Application is up and running on localhost:5000 port.



- POST- http://127.0.0.1:5000/upload to upload the file.

In the body select key:file and select the image in the value. On successful upload it will give the filename.



→ **Performing other steps. (Covered all the points given the challenge)**

The file uploaded will be only processed.

Kept the solution simple by using cv2 package.

When an extra step by saving the image with all details like ID , radius , centroid details etc.

Testing with screen shots:

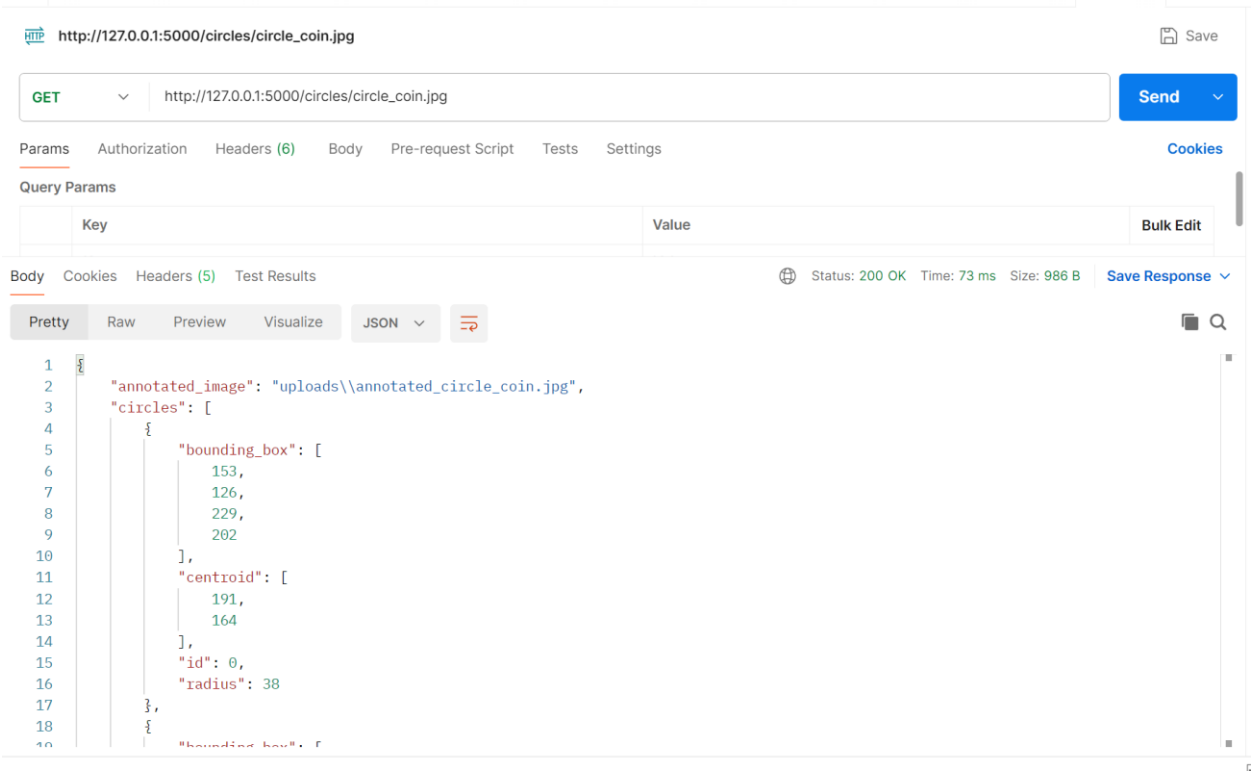
- GET method http://127.0.0.1:5000/circles/circle_coin.jpg

  The url should always be the same as we are loading the image with circle_coin.jpg name.

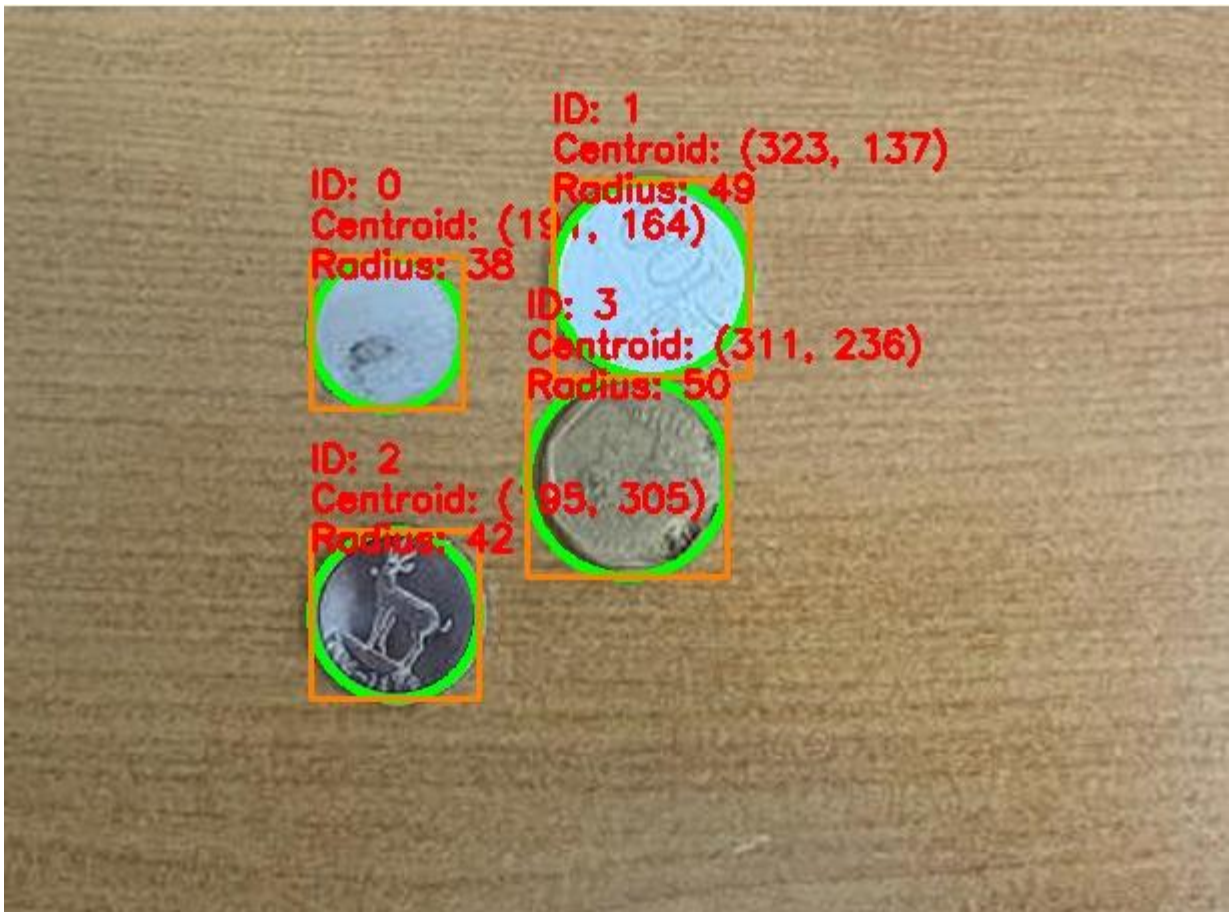
  An image with all the details is getting saved in the same folder with annoated_circle_coin.jpg file.

  All the values can also be seen in the output in response.json format in Postman.

response.json



- All the output is feeded into the annoatated_circle_coin.jpg file.
    - Can observe from the image of the circle is masked. Detected the object with bounded box as well.
    - Unique ID number to each circle.
    - Coordinates of the centroid
    - Radius of the circle.



- Containerized the application

Steps to run the same.

1. Unzip the file.
2. Open the CMD prompt and go into challnge_1 folder and then into app folder.(Make sure docker is installed and running)
3. Run --> docker build . -t flask-app:v1
4. Run --> docker container run -d -p 5000:5000 flask-app:v1
5. Implemented peristant storage.
6. The application can now be accessed with Postman with the above url given in the screenshots.

Screen shot of running container.

Testing Strategy:

- Process all the image and fetch the metric.
- Use IOU (Intersection of Union) and by providing a threshold value evaluate accuracy score.
- As an alternative mAP score can also be used.

Due to constrain of time could not deep dive into the testing along with code.

Note --> The solution could have been more dynamic. Considering the timeline and the intent of solution kept the solution the same way.

Challenge 2: This challenge has an attached csv file which contains image data referenced by the column depth. The rest of the columns (200) represents image pixel values from 0 to 255 at each depth.

Below is the expectations from the solution.

- Need to resize the image width to 150 instead of 200.
- The resized image has to be stored in a database.
- An API is required to request image frames based on depth_min and depth_max.
- Apply a custom color map to the generated frames.
- The solution should be based in Python.
- Containerize the solution.

Solution:

1> Developed a flask-based application. It consists of 2 requests.

2> First request is just to check if the application is up and running. Second is GET request to perform all the required steps by taking minimum and maximum depth range from the user and proving the image.

3> Containerized the application and hosted on localhost:5000 port.

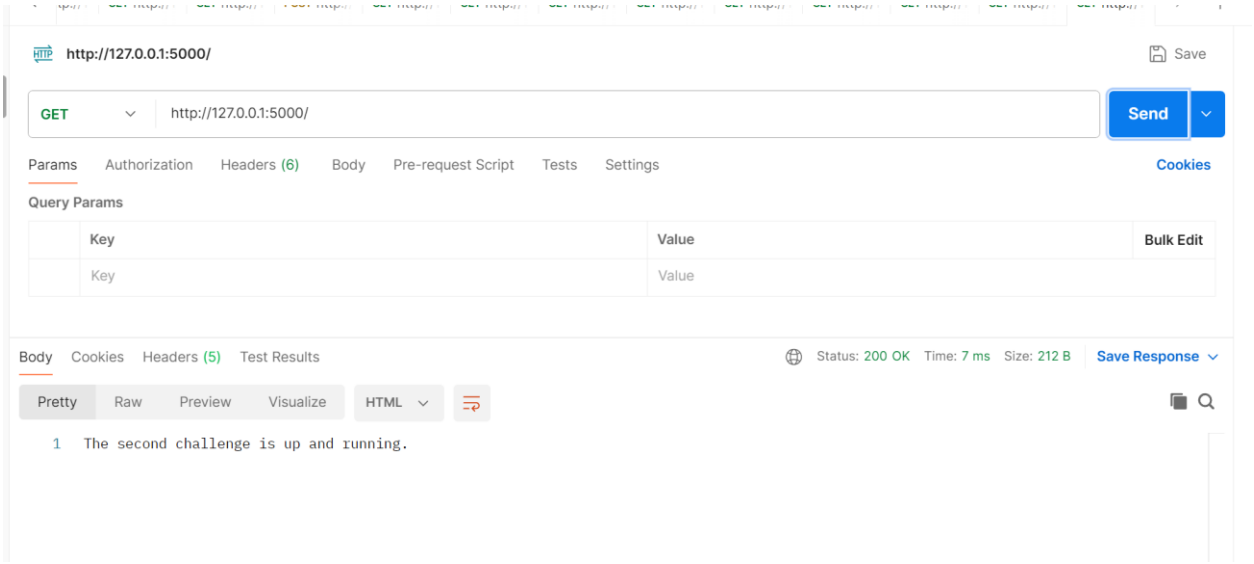
4> Used Python for development and Postman for testing the application.

Details explanation of steps.

1. Read the data from the provided csv file. Used pandas to fetch the data and load it into data frame.
2. Resized the image from 200 to 150. For reference saving the resized image as well.
3. Used sqlite3 as the database to store the data and created a table named as images. The database can be seen in the

executing folder.

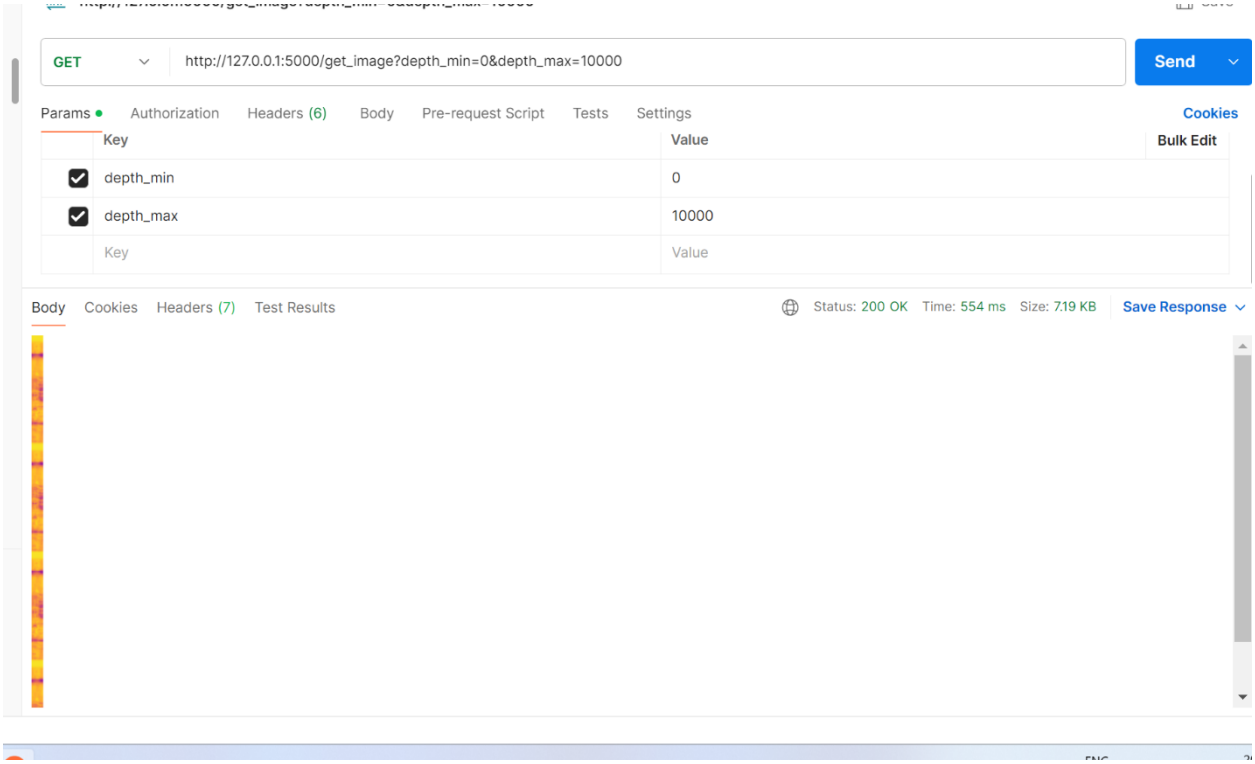
Testing the application:
Application is up and running on localhost:5000 port



- GET method http://127.0.0.1:5000/get_image?depth_min=0&depth_max=10000

  We need to ensure that min and max depth is provided.

  In the output the custom color map is applied to the generated images.



- In the working folder we can see a Database file with name “images”. And resized image as well.

- Containerized the application

Steps to run the same.

1. Unzip the file.
2. Open the CMD prompt and go into challnge_2 folder and then into app folder.(Make sure docker is installed and running)
3. Run --> docker build . -t flask-app-ch2:v1
4. Run --> docker container run -d -p 5000:5000 flask-app-ch2:v1
5. The application can now be accessed with Postman with the above url given in the screenshots.

Screen shot from dcoker

Ending Keynotes:

1. The solution has been tested both with and without containerization.
2. All the important requirements have been addressed.
3. The solution provided is done keeping the time line and purpose into consideration.
4. All the steps to run are provided in the solution with screen shots.
5. There were a few queries but due to limited time have proceeded with the solution with some assumptions. This was also suggested.
6. Both the application is hosted on the same port. Make sure to run one at a time.