Name – Priyansu Saha **Email – saha_priyansu@hotmail.com**

Computer Vision Internship - Coding Round

Question 1: Dog Face & Pose Estimation Detection

Approach

Step 1: Dataset Preparation

For this task, the provided dataset was first annotated and then augmented to improve the model's robustness and generalization capability. The annotation process involved labeling the key features in the images, which were crucial for detecting vanishing points and the lines that lead to them.

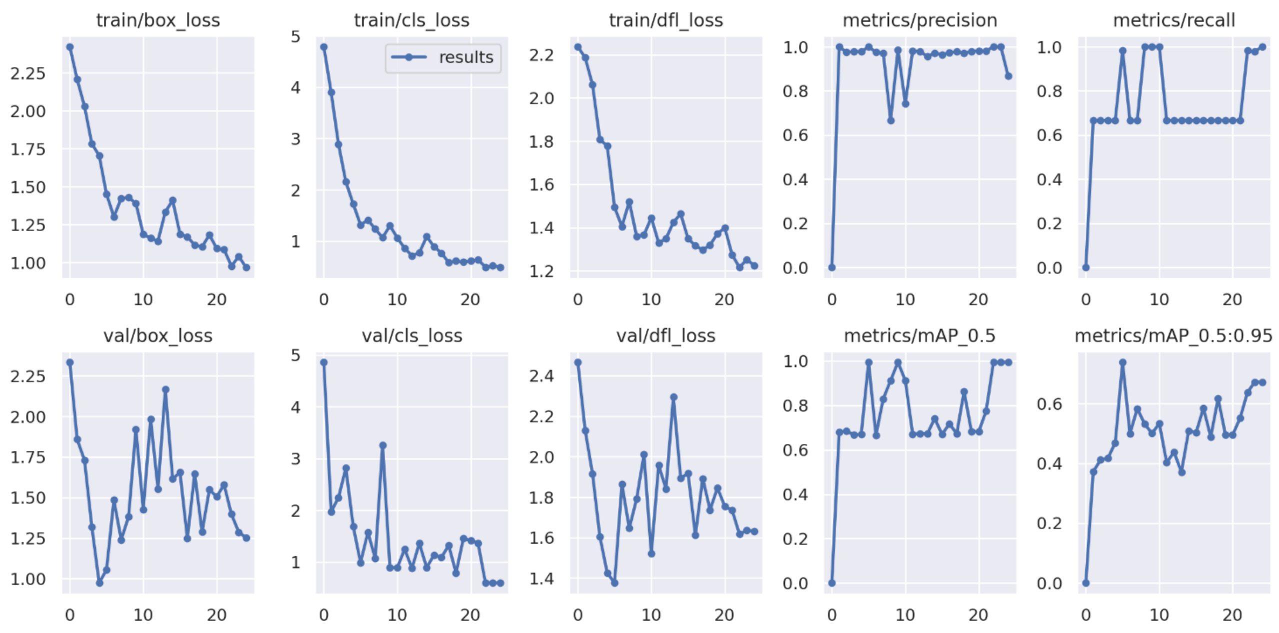
To enhance the dataset and create a more diverse set of images for training, the following augmentations were applied using Roboflow:

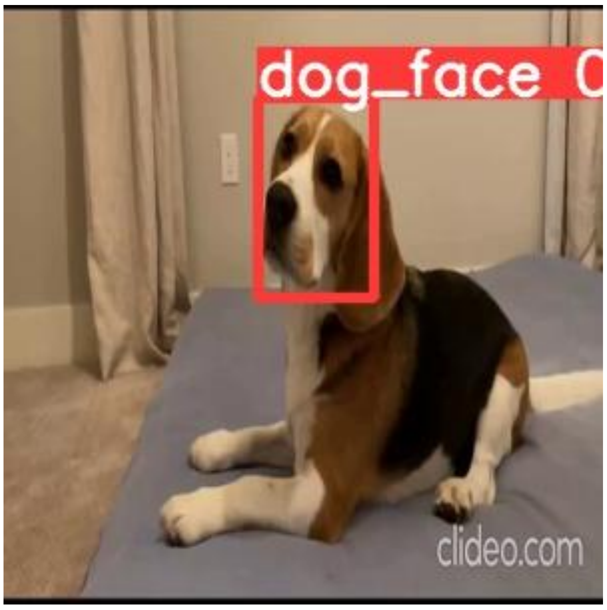
- 50% Probability of Horizontal Flip: Randomly flipped the images horizontally to introduce variation in the orientation of objects and lines in the images.
- 50% Probability of Vertical Flip: Similar to the horizontal flip, this augmentation flipped the images vertically, providing additional perspectives for the model to learn.
- Random Crop (0-20%): Applied a random crop on each image, removing between 0% and 20% of the image area. This helped the model focus on smaller parts of the image and learn to generalize better.
- Random Rotation (-15 to +15 degrees): The images were rotated randomly between -15 and +15 degrees to simulate different camera angles and perspectives.
- Random Brightness Adjustment (-15% to +15%): The brightness of the images was randomly adjusted within a range of -15% to +15%, simulating different lighting conditions and improving the model's ability to handle various illumination scenarios.

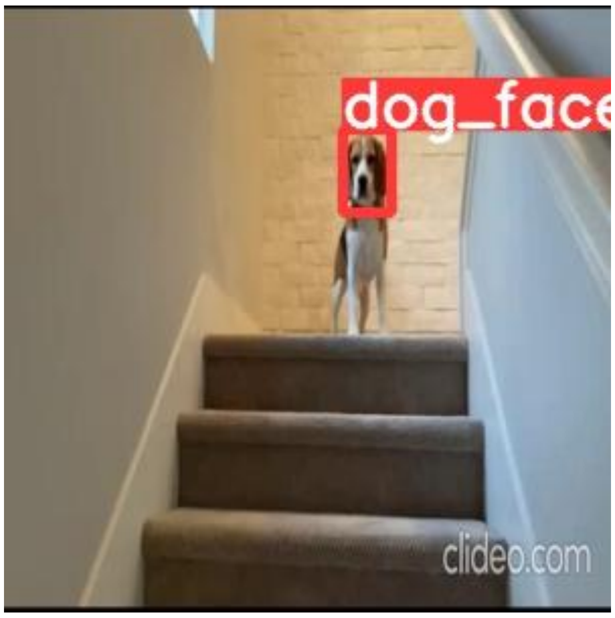
Step 2: Training the deep learning model

For training the dataset to detect dog faces, I used YOLOv9 due to its efficiency and high accuracy in object detection tasks. The model was trained on the annotated and augmented dataset, with a focus on detecting dog faces specifically. After training the model for 25 epochs, I achieved an impressive mean Average Precision (mAP) of 99.5%**,** indicating a high level of accuracy in detecting and localizing dog faces in the images. The YOLOv9 model was initialized with the Gelen-C pre-trained weights, enabling faster convergence and leveraging previously learned features.

Results







Challenges Faced

During the course of this project, one of the primary challenges was the pose detection component of the assignment. Specifically, detecting pose landmarks on the dog's face, such as the nose, eyes, and ears, proved to be more complex than initially anticipated. Although the dog face detection task using YOLOv9 was successfully completed.

Question 2: Vanishing Point Estimation in Computer Vision.

Approach

To solve this problem, I followed these steps:

I. **Detect Lines**: First, I identified all the lines present in the image. These lines should be at least a few pixels long.

   To achieve this, I first converted the colored input image to grayscale using the cvtColor function from the OpenCV library. Then, I applied a Gaussian blur to the image using the GaussianBlur function, which helped remove minor noise that could cause irregular edges. This step was crucial to avoid false positives, as such noise could result in detecting unnecessary lines.

   Next, I generated the edge image using the Canny function from the OpenCV library. This edge image served as the basis for detecting lines. Finally, I used the HoughLinesP function from OpenCV to find the lines in the edge image. If no lines were detected, I exited the process at this step.

II. **Filter Lines**: Next, I filtered the detected lines based on their angles with respect to the horizontal and their lengths.

   To filter out horizontal and vertical lines, I calculated the equation of each line in the format $y = m x + c$ y=mx+c, where m m is the slope. Using the slope, I found the angle of each line relative to the horizontal using the atan function from the math library, converting the result to degrees. I then rejected lines with angles close to 0° (horizontal) or ±90° (vertical) by introducing a threshold, REJECT_DEGREE_TH = 4.0. This ensured only lines with angles in the ranges $[-86°, -4°]$ [−86°,−4°] and $[4°, 86°]$ [4°,86°] were kept. I also calculated the length of each line using the Euclidean distance formula and stored the filtered lines in the format $[x1, y1, x2, y2, m, c, l]$ [x1,y1,x2,y2,m,c,l], where l l is the line's length. These filtered lines will be used in later stages of the project.

   To improve accuracy and efficiency, I avoided using very short lines in the processing, as extending them might lead to errors in finding the vanishing point. Instead of setting a fixed minimum length, I took a different approach for two reasons:

   1. Eliminating short lines might result in losing all detected lines, especially if the input image is small.
   2. Having too many lines after filtering could slow down the computation unnecessarily, as only a few lines are needed to find the vanishing point accurately.

   To address this, I selected only the 8 longest lines from the filtered lines (or all lines if fewer than 8 were available). I sorted the lines by their length using the sorted function and sliced out the top 8. This approach ensured efficiency while retaining enough lines for accurate computation of the vanishing point.

III. **Find the Vanishing Point**: Finally, I determined the vanishing point of the image using the lines identified in the previous steps. The vanishing point is approximately the intersection point of these lines.

To calculate the vanishing point, I introduced an "error" factor, which is the square root of the sum of squared distances from a point to each of the lines. The point with the minimum error is likely the closest to the vanishing point. For example, I considered the intersection of two lines, L1 and L2, with equations $y = m_1 x + c_1$ y=m 1 x+c 1 and $y = m_2 x + c_2$ y=m 2 x+c 2 , respectively. To find the distance of the intersection point (x_0, y_0) (x 0 ,y 0) from another line L, I created a perpendicular line L_ through (x_0, y_0) (x 0 ,y 0). The distance between (x_0, y_0) (x 0 ,y 0) and its intersection with L gives the error for that point with respect to the line. By calculating the error for each line, I could determine the point with the smallest error as the vanishing point.

Results

Image 1

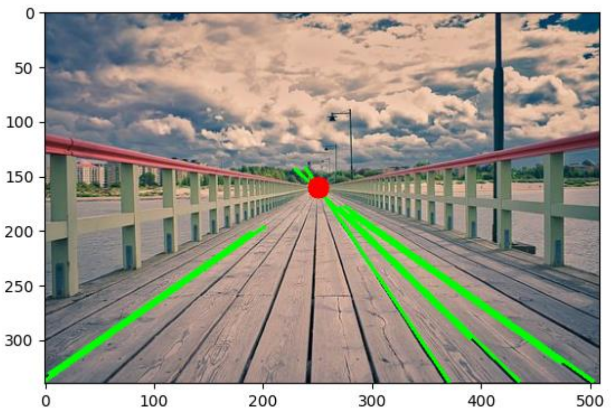


Image 2

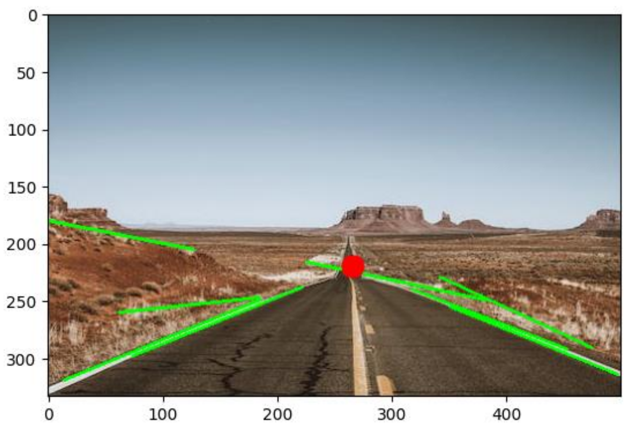


Image 3

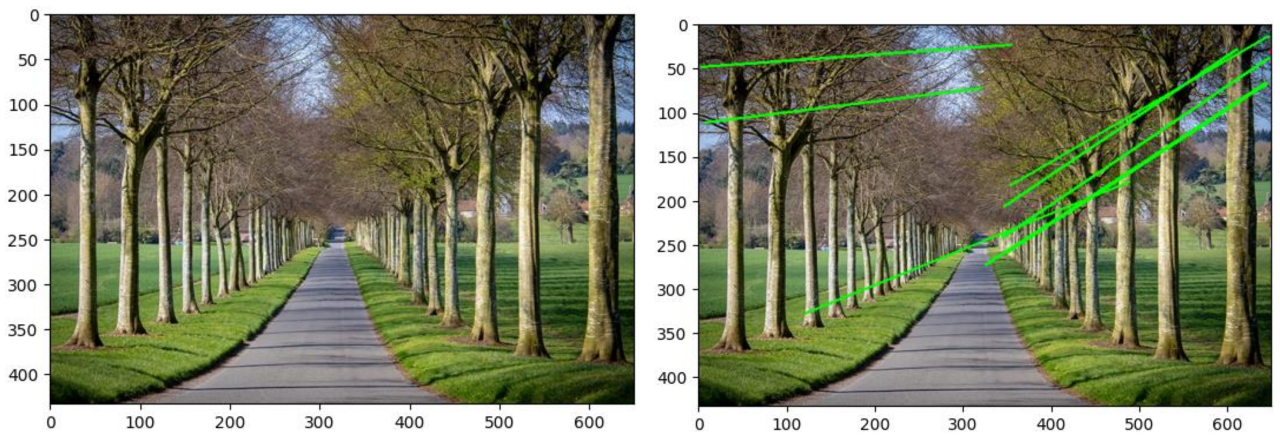
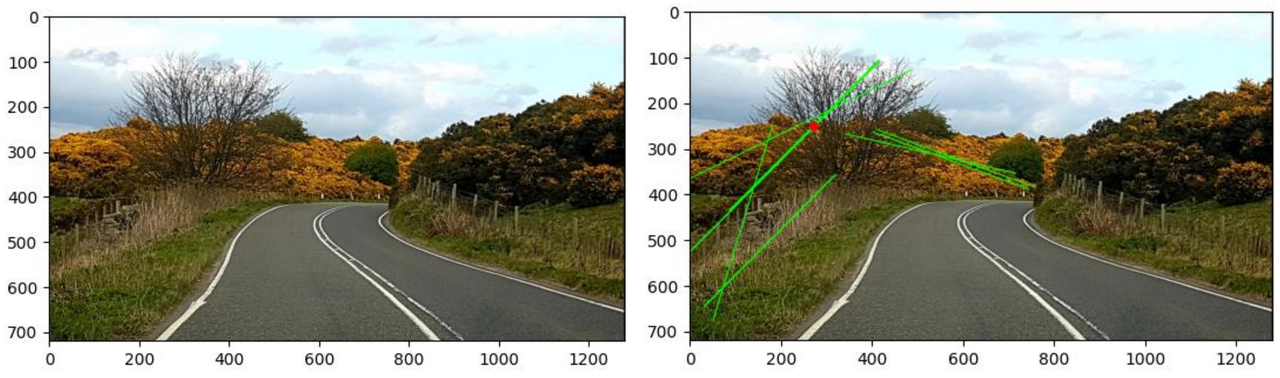


Image 4



Challenges Faced

One of the major challenges I faced during this assessment was that, in Image 3 and Image 4, the lines leading to the vanishing points were not straight, but rather curved. This posed a significant issue because the approach I used for detecting vanishing points relied on straight lines and their intersections. The process of identifying the vanishing point involves finding lines and calculating their intersection points, assuming that these lines are straight.