**Object Detection using MLP Classifier - Interview Task Report**

Introduction:

In this report, I will provide an overview of the Python code that implements object detection using a Multi-Layer Perceptron (MLP) Classifier. The code reads and preprocesses grayscale images, trains an MLP Classifier to detect objects, and evaluates its accuracy. Below, I'll break down the code and provide a summary of the work, including the instructions used.

Code Overview:

The code can be divided into several sections, each serving a specific purpose:

1. Importing Libraries:

- The code begins by importing necessary libraries, including scikit-learn (sklearn) for the MLP Classifier, os for file handling, OpenCV (cv2) for image processing, and NumPy for numerical operations.

2. Preprocessing Function:

- The `preprocess\_image` function reads and processes images. It loads images in grayscale, resizes them to 100x100 pixels, and flattens them into a 1D array.

3. Data Preparation:

- The code initializes empty lists for `images`, `labels`, and `image\_files` to store image data, labels (always set to 1), and the names of image files, respectively.

- It then iterates through files in the "ml\_task" directory, identifies image files (those ending with ".jpg"), preprocesses them using the `preprocess\_image` function, and stores them along with their labels in the respective lists.

4. Train-Test Split:

- The `train\_test\_split` function from scikit-learn is used to split the data into training and testing sets (70% for training and 30% for testing) with a fixed random state for reproducibility.

5. Model Building and Training:

- An MLP Classifier is created with a single hidden layer containing 100 neurons and a maximum of 500 iterations for training.

- The model is trained on the training data using `model.fit`.

6. Model Evaluation:

- The code calculates the accuracy of the trained model on the test data using `model.score`.

- It also calculates the total number of objects detected in the test dataset.

7. Object Detection per Image:

- For each test image, the code predicts the number of objects using `model.predict` and stores the result along with the image file name.

8. Displaying Results:

- The code prints the overall accuracy and the number of objects detected in each test image.

Instructions Used:

The following instructions were used in the code:

- Import the necessary libraries, including scikit-learn, OpenCV, os, and NumPy.

- Define a function (`preprocess\_image`) to read and preprocess images (grayscale conversion, resizing, and flattening).

- Prepare the dataset by iterating through image files in the "ml\_task" directory, preprocessing images, and storing them with labels.

- Split the dataset into training and testing sets (70% training, 30% testing) using `train\_test\_split`.

- Create an MLP Classifier with one hidden layer containing 100 neurons and a maximum of 500 training iterations.

- Train the model on the training data.

- Evaluate the model's accuracy on the test data.

- Predict and record the number of objects detected in each test image.

- Display the overall accuracy and the number of objects detected in each test image.

Conclusion:

The code successfully implements object detection using an MLP Classifier. It processes a set of grayscale images, trains the classifier, and evaluates its accuracy. Additionally, it provides a per-image breakdown of object detection results. The code can be further improved by expanding the dataset, fine-tuning model parameters, and optimizing object detection accuracy.