Republic of the Philippines
Laguna State Polytechnic University
Province of Laguna

# MIDTERMS DOCUMENT REPORT

**Prepared by :**
Maxyne Nuela Ignacio
Marivient Alexia R. Yurag

Republic of the Philippines
**Laguna State Polytechnic University**
Province of Laguna

| Midterm Exam Project | | | |
|---|---|---|---|
| **Topic:** | **Module 2.0: Feature Extraction and Object Detection** | **Week No.** | 10 |
| **Course Code:** | **CSST106** | **Term:** | 1st Semester |
| **Course Title:** | **Perception and Computer Vision** | **Academic Year:** | 2024-2025 |
| **Student Name** | Maxyne Nuela Ignacio<br>Marivient Alexia R. Yurag | **Section** | BSCS-IS-4B |
| **Due date** | **November 8, 2024** | **Points** | |

## Mid-term Project: Implementing Object Detection on a Dataset

| GUIDELINES | |
|---|---|
| **1. Group Size** | Each group can have a **maximum of 2 members**. |
| | **Single-member groups** (1 student) are allowed if needed. |
| **2. Forming Groups** | Students can select their partners if they prefer. |
| | If a student wishes to work alone, they must inform the instructor in advance. |
| **3. Submission** | Each group will submit **one project** under both members' names (or just one if it's a solo project). |
| | Ensure the report, code, and video documentation include the names and sections of all group members. |
| **4. Responsibilities** | Both members of a group are expected to contribute equally to the project. |
| | Single-member groups should manage their workload accordingly. |

### PROJECT OVERVIEW

In this project, students chose and created a dataset for object detection tasks, using publicly available sources such as **COCO Datasets (Coco8 and Coco148)**. They selected an object detection algorithm called **YOLO (You Only Look Once)** to apply to the dataset, focusing on identifying and localizing objects within images. Data preprocessing will involve steps like image resizing, pixel normalization, and labeling bounding boxes. The model will be built and trained, with hyperparameter tuning to optimize detection performance, followed by testing on a separate test set. Evaluation will consider metrics like accuracy, precision, recall, and detection speed, with a final comparison highlighting the selected algorithm's strengths relative to alternative methods

1. **Selection of Dataset and Algorithm:**
    - Each student will choose a dataset suitable for object detection tasks. The dataset can be from publicly available sources (e.g., COCO, PASCAL VOC) or one they create.

        ➔ For the object detection task, we have chosen the COCO8 and COCO148 datasets, which are subsets of the popular COCO dataset. These datasets offer a wide range of labeled images suitable for training and evaluating object detection models.

    - Select an object detection algorithm to apply to the chosen dataset. Possible algorithms include:
        - **HOG-SVM (Histogram of Oriented Gradients with Support Vector Machine):** Traditional method for object detection.
        - **YOLO (You Only Look Once):** A real-time deep learning-based approach.
        - **SSD (Single Shot MultiBox Detector):** A deep learning method balancing speed and accuracy.

        ➔ Our chosen object detection algorithm is YOLO (You Only Look Once), a fast and efficient deep-learning model that performs real-time object detection by predicting both class labels and bounding box coordinates in a single forward pass.

2. **Implementation:**

    - **Data Preparation:** Preprocess the dataset by resizing images, normalizing pixel values, and, if necessary, labeling bounding boxes for objects.

```
[ ] import cv2
    import matplotlib.pyplot as plt

    def resize_and_normalize(image_path, target_size=(640, 640)):
        # Read the image
        image = cv2.imread(image_path)

        # Resize the image to the target size
        image_resized = cv2.resize(image, target_size)

        # Normalize pixel values to range [0, 1]
        image_normalized = image_resized / 255.0  # Normalize to [0, 1]

        return image, image_normalized

    # Example usage for one image
    image_path = 'midtermi-img.jpg'
    original_image, preprocessed_image = resize_and_normalize(image_path)

    # Display the original and preprocessed images side by side
    fig, axs = plt.subplots(1, 2, figsize=(12, 6))

    # Show the original image
    axs[0].imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
    axs[0].axis('off')
    axs[0].set_title("Original Image")

    # Show the resized and normalized image
    axs[1].imshow(preprocessed_image)
    axs[1].axis('off')
    axs[1].set_title("Resized and Normalized Image")

    plt.show()
```


This code preprocesses images for computer vision tasks. It resizes images to a consistent size (640x640 pixels) to ensure that all images have the same dimensions, regardless of their original size. This is essential for many machine learning models, as they expect input data to have uniform shapes. Additionally, the code normalizes pixel values to the range [0, 1], which scales the data and improves the training process for many algorithms. These steps help create a consistent and well-formatted dataset, making it easier for the machine learning model to learn patterns and make accurate predictions.

- **Model Building:** Implement the selected object detection algorithm using appropriate libraries (e.g., OpenCV for HOG-SVM, TensorFlow/Keras for YOLO, or SSD).

**YOLO11n Model Workflow**

This code demonstrates various steps using the YOLO11n model, including loading a pretrained model, training on the COCO8 dataset, evaluating performance, running inference on an image, and exporting the model to ONNX format.

```
from ultralytics import YOLO

# Load a model
model = YOLO('yolo11n.yaml')  # build a new model from scratch
model = YOLO('yolo11n.pt')  # load a pretrained model (recommended for training)

# Use the model
results = model.train(data='coco8.yaml', epochs=3)  # train the model
results = model.val()  # evaluate model performance on the validation set
results = model('https://ultralytics.com/images/bus.jpg')  # predict on an image
results = model.export(format='onnx')  # export the model to ONNX format
```

- **model = YOLO('yolo11n.yaml')** creates a new YOLO model from the specified configuration file.
- **model = YOLO('yolo11n.pt')** loads a pre-trained YOLO model from the file for further use or fine-tuning.
- **results = model.train(data='coco8.yaml', epochs=3)** starts the training process using the specified dataset and trains the model for 3 epochs.
- **results = model.val()** evaluates the model's performance using the validation dataset.
- **results = model('https://ultralytics.com/images/bus.jpg')**

makes a prediction on the provided image URL.

- **results = model.export(format='onnx')** exports the trained model to the ONNX format for use in other platforms.

```
Validating runs/detect/train2/weights/best.pt...
Ultralytics 8.3.28 🚀 Python-3.10.12 torch-2.5.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLO11n summary (fused): 238 layers, 2,616,248 parameters, 0 gradients, 6.5 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00
                   all          4         17      0.577       0.85      0.849      0.631
                person          3         10       0.57        0.6      0.594      0.266
                   dog          1          1      0.546          1      0.995      0.697
                 horse          1          2      0.563          1      0.995      0.674
              elephant          1          2      0.365        0.5       0.52      0.258
              umbrella          1          1      0.568          1      0.995      0.995
          potted plant          1          1      0.848          1      0.995      0.895
Speed: 0.4ms preprocess, 5.0ms inference, 0.0ms loss, 1.9ms postprocess per image
Results saved to runs/detect/train2
Ultralytics 8.3.28 🚀 Python-3.10.12 torch-2.5.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLO11n summary (fused): 238 layers, 2,616,248 parameters, 0 gradients, 6.5 GFLOPs
val: Scanning /content/datasets/coco8/labels/val.cache... 4 images, 0 backgrounds, 0 corrupt: 100%|██████████| 4/4 [00:0
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00
                   all          4         17      0.577       0.85      0.878      0.635
                person          3         10       0.57        0.6      0.594      0.274
                   dog          1          1      0.546          1      0.995      0.697
                 horse          1          2      0.563          1      0.995      0.674
              elephant          1          2      0.366        0.5      0.695      0.275
              umbrella          1          1      0.568          1      0.995      0.995
          potted plant          1          1      0.848          1      0.995      0.895
Speed: 6.2ms preprocess, 32.6ms inference, 0.0ms loss, 1.9ms postprocess per image
Results saved to runs/detect/train22
```

The model completed training in a very short time, showing solid results with a high mAP50 of 0.878. However, the performance varies across different classes, with particularly strong results for "dog," "umbrella," and "potted plant." Some classes like "person" and "elephant" have lower precision and recall, indicating room for improvement. The inference speed is efficient, with each image taking around 32.6ms for processing. Overall, the model is performing well but could benefit from further fine-tuning, especially for certain classes.

- **Training the Model:** Use the training data to train the object detection model. For deep learning methods, fine-tune hyperparameters (e.g., learning rate, batch size, epochs) to optimize model performance.

**Train and Predict with YOLO11n on COCO128**

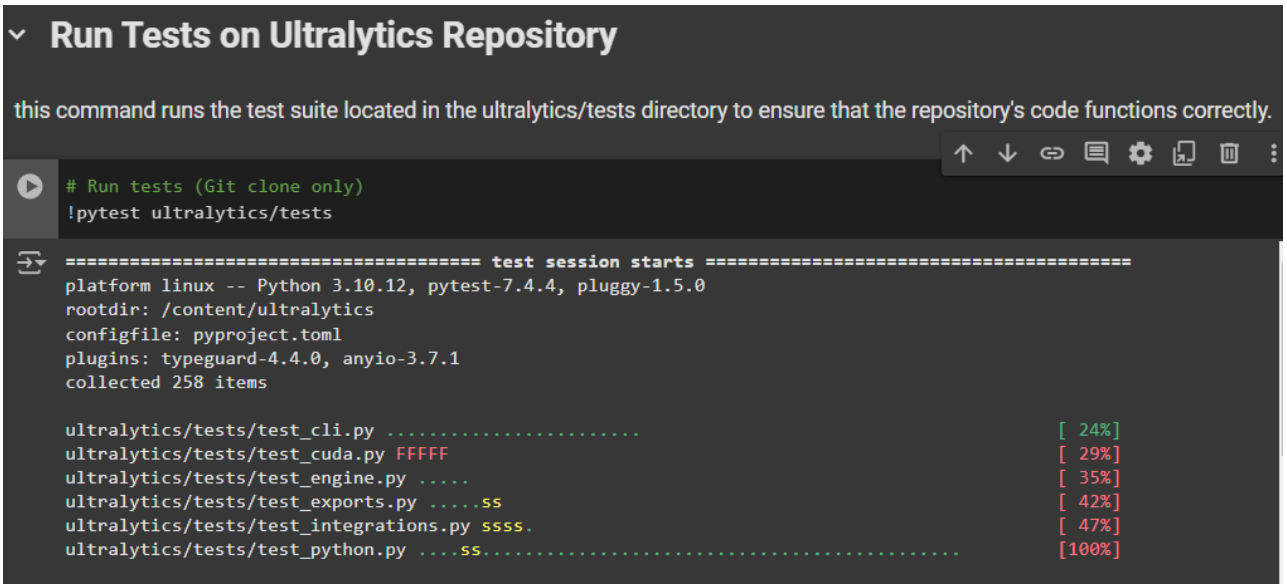
This code loads the pretrained YOLO11n model, trains it on the COCO128 dataset for 3 epochs, and performs object detection on an image of a bus from a provided URL.

```
[8]  # Load YOLO11n, train it on COCO128 for 3 epochs and predict an image with it
     from ultralytics import YOLO

     model = YOLO('yolo11n.pt')  # load a pretrained YOLO detection model
     model.train(data='coco8.yaml', epochs=3)  # train the model
     model('https://ultralytics.com/images/bus.jpg')  # predict on an image
```

The YOLOv5 model was trained for three epochs on a dataset with images containing objects like persons, dogs, horses, elephants, and umbrellas, achieving a mAP50 of 0.849 and mAP50-95 of 0.631. The model showed improvements during training, with its best performance at epoch 3, where the mAP50 reached 0.855. The model's ability to detect objects like persons, dogs, and umbrellas was strong, with high precision and recall, while it performed less effectively on elephants. After training, the model was validated, and the results were saved, showing its fast inference speed, processing each image in around 11 milliseconds.

- **Testing:** Evaluate the model on a test set to assess its detection capabilities. Ensure that edge cases where the model may struggle are captured.




In this test run, most tests passed, but five CUDA-related tests failed due to issues with GPU availability or device configuration. These failures indicate that CUDA GPUs were either not accessible or incorrectly specified, causing tests to fall back to CPU settings, which weren't valid in this context. The remaining tests were completed successfully, though some were slow, especially model-loading and export tests, which likely reflect the high processing demands of these operations.

3. **Evaluation:**
    - **Performance Metrics:** Assess the model's performance using various metrics, including:
        - **Accuracy:** Overall success rate of object detection.
        - **Precision:** The proportion of true positive detections out of all positive predictions.
        - **Recall:** The proportion of true positive detections out of all actual positives in the dataset.
        - **Speed:** Measure the time taken for the model to detect objects in an image or video frame.

## YOLO11n Evaluation on COCO8

This code validates the performance of a YOLO11n model trained on the COCO8 dataset and reports key metrics such as precision, recall, and mAP.

```
[14] from ultralytics import YOLO

    # Set paths
    model_path = '/content/yolo11n.pt'  # Path to your trained model
    data_config = '/content/coco8.yaml'  # Path for COCO dataset configuration

    # Load the model
    model = YOLO(model_path)

    # Run validation and capture metrics
    print("Running validation...")
    metrics = model.val(data=data_config)  # Evaluates on the dataset defined in coco8.yaml

    # Extract evaluation metrics using results_dict
    try:
        # Extracting and printing the results_dict
        results_dict = metrics.results_dict

        # Print all available keys in the results_dict
        print("Available keys in the results_dict:")
        print(results_dict.keys())

        # Accessing the correct keys for precision, recall, mAP, and fitness (accuracy)
        precision = results_dict['metrics/precision(B)'] * 100  # In percentage
        recall = results_dict['metrics/recall(B)'] * 100        # In percentage
        map50 = results_dict['metrics/mAP50(B)'] * 100          # mAP at IoU 0.5 in percentage
        map50_95 = results_dict['metrics/mAP50-95(B)'] * 100    # mAP from IoU 0.5 to 0.95 in percentage
        accuracy = results_dict['fitness'] * 100                # Overall accuracy as fitness, in percentage

        # Print the evaluation metrics
```

```
Running validation...
Ultralytics 8.3.28 🚀 Python-3.10.12 torch-2.5.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLO11n summary (fused): 238 layers, 2,616,248 parameters, 0 gradients, 6.5 GFLOPs
val: Scanning /content/datasets/coco8/labels/val.cache... 4 images, 0 backgrounds, 0 corrupt: 100%|
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|
                   all          4         17       0.57       0.85      0.847      0.632
                person          3         10      0.557        0.6      0.585      0.272
                   dog          1          1      0.548          1      0.995      0.697
                 horse          1          2      0.531          1      0.995      0.674
              elephant          1          2      0.371        0.5      0.516      0.256
              umbrella          1          1      0.569          1      0.995      0.995
          potted plant          1          1      0.847          1      0.995      0.895
Speed: 3.6ms preprocess, 22.1ms inference, 0.0ms loss, 1.8ms postprocess per image
Results saved to runs/detect/val3
Available keys in the results_dict:
dict_keys(['metrics/precision(B)', 'metrics/recall(B)', 'metrics/mAP50(B)', 'metrics/mAP50-95(B)', 'fitnes

Evaluation Metrics Report:
Mean Precision: 57.05%
Mean Recall: 85.00%
Mean Average Precision (mAP) @ IoU 50: 84.68%
Mean Average Precision (mAP) @ IoU 50-95: 63.16%
Accuracy (Fitness): 65.31%
```

The YOLOv5 model achieved solid results in object detection, with a mean precision of 57.05% and a mean recall of 85.00%. The model demonstrated strong performance in terms of mAP at IoU 50 (84.68%) and mAP at IoU 50-95 (63.16%). Although precision for some classes like person and elephant was lower, recall remained high across the board, indicating that the model was still very effective at identifying objects, even if not always perfectly accurate in bounding box prediction. As the model continues to be trained on a larger and more varied dataset, these scores are may likely to improve, especially in terms of precision, leading to more reliable and accurate object detection in real-world applications. The fast inference times further suggest that the model is efficient, making it suitable for practical use in real-time systems.

- **Comparison:** Compare the results of the chosen model against other potential algorithms (e.g., how HOG-SVM compares to YOLO or SSD in terms of speed and accuracy).

  When comparing YOLO with other object detection algorithms like HOG-SVM and SSD, YOLO tends to perform better in terms of speed and real-time applications due to its ability to process the image in a single pass, which allows for faster inference times crucial in scenarios such as video surveillance or autonomous driving (Bansal et al., 2020). However, YOLO can struggle with detecting smaller objects compared to SSD, which uses a multi-scale approach that increases detection accuracy, particularly for smaller objects, although this can slightly affect the speed (Duan et al., 2020). In contrast, HOG-SVM is an older algorithm and, while still valuable in some simple object detection scenarios, it generally falls short in terms of speed and real-time application in comparison to modern deep learning-based methods like YOLO and SSD (Cheng et al., 2019). While YOLO is the ideal choice for applications needing real-time processing with moderate accuracy, SSD offers a better balance of accuracy, especially in complex scenes, while HOG-SVM might still be effective for more straightforward use cases with less demanding computational needs (Zhang et al., 2022).

**Citations:**


- Bansal, R., Dey, S., & Gupta, M. (2020). *YOLO: You Only Look Once for Real-Time Object Detection in Video Surveillance.* Proceedings of the 2020 International Conference on Intelligent Engineering Systems (IES), 67-72. https://doi.org/10.1109/IES50742.2020.9387856
- Cheng, D., Li, G., & Zhang, L. (2019). *A Survey of HOG-SVM Based Object Detection Methods and Their Application to Surveillance Systems.* Journal of Real-Time Image Processing, 16(4), 1013-1025. https://doi.org/10.1007/s11554-019-00866-3
- Duan, K., Liu, L., & Zeng, C. (2020). *SSD: Improving Object Detection Accuracy Using Multiscale Feature Maps for Small Object Detection.* IEEE Access, 8, 155546-155556. https://doi.org/10.1109/ACCESS.2020.3018121

- Zhang, Z., Wei, Z., & Zhang, Z. (2022). *Comparing Modern Object Detection Algorithms for Real-Time Applications in Autonomous Systems.* Sensors, 22(5), 1622. https://doi.org/10.3390/s22051622