CAP6415_F25_project-Training-a-large-model-using-Unity-Perception-images

1a. INTRODUCTION & PROBLEM STATEMENT

Modern object detection models such as YOLOv8, DETR, and Mask R-CNN achieve high accuracy when trained on sufficiently diverse and well-balanced datasets. However, real-world datasets frequently contain class imbalance, where certain categories—often those that are rare in daily environments—are significantly underrepresented. This imbalance leads to poor generalization, decreased recall on minority classes, and unstable learning behavior during training.

In many practical domains, collecting additional real images for such rare classes is expensive, time-consuming, or infeasible. Consequently, there is growing interest in synthetic data generation, particularly 3D simulation environments that can automate the creation of diverse, labeled images. Unity's Perception package provides an efficient pipeline for generating images with photorealistic lighting, camera variation, and automatic annotations. It also allows scripting of complex scenes and procedural randomness, enabling the creation of large numbers of diverse training samples with minimal manual labeling cost. This project leverages that capability to supplement missing or underrepresented object categories.

1b. PROBLEM STATEMENT

Many object detection pipelines fail to detect certain classes not because the model architecture is insufficient, but because the dataset does not adequately represent those classes. Underrepresentation leads to biased training, weak feature formation, and a significant drop in detection performance on those categories.

This project investigates whether **synthetic 3D data**, generated programmatically using Unity Perception, can effectively compensate for missing or scarce real-world data. The objective is to build a controlled synthetic dataset targeting underrepresented classes and fuse it with a smaller real-world dataset to evaluate whether performance improves across all categories.

The primary goals are:

1. **Synthetic Scene Generation**
   Use Unity Perception to procedurally generate 3D scenes containing rare or missing object classes and automatically export labeled images in a consistent format.

2. **Model Training With Synthetic and Mixed Data**
   Train or fine-tune a state-of-the-art object detection model (YOLOv8 in this study), comparing:

   - synthetic-only training,
   - real-only training,
   - and a mixed dataset combining real and synthetic images.

3. **Evaluation of Synthetic Data Effectiveness**
   Measure performance across accuracy, precision, recall, and class-wise F1 scores to determine whether synthetic data improves recognition of underrepresented objects.

1c. SCOPE OF THE PROJECT

The study focuses on five everyday objects—bottle, cup, chair, laptop, and book—chosen because they appear frequently in real environments yet show substantial variation across datasets. Synthetic data is generated using controlled Unity scenes, while real images are drawn from curated subsets of COCO and small manually collected datasets.

The work does not explore domain randomization beyond standard lighting and pose variation, nor does it attempt extreme photorealism or large-scale 3D rendering. Instead, the objective is to evaluate practical, lightweight synthetic augmentation that can be reproduced easily.

---

2. Related Work

2a. Dataset Imbalance in Object Detection

The challenge of class imbalance is a persistent issue in object detection, directly impacting both the recall and precision of underrepresented object classes. Foreground-foreground and foreground-background imbalance, where certain classes possess significantly fewer labeled instances than others, are particularly problematic, often causing detectors to favor majority classes and neglect rare categories. This leads to diminished recall for uncommon classes and increases the likelihood of misclassification, ultimately degrading the overall performance of the detection model. Various mitigation strategies have emerged, including data-level methods such as resampling, algorithmic techniques like loss reweighting and focal loss, and augmentation-based approaches. While sampling and reweighting have demonstrated effectiveness for two-stage detectors, recent evidence indicates their limited benefits—and occasional drawbacks—when applied to modern one-stage architectures like YOLO, where augmentation dominates as the preferred solution for boosting detection rates among minority classes.arxiv+4

2b. Synthetic Data and Domain Randomization

Synthetic data has become an essential resource for training object detectors, especially where data acquisition and labeling are expensive or infeasible. Domain randomization, involving the procedural variation of scene content, lighting, material properties, and object orientations, has proven to be a powerful technique for bridging the domain gap between synthetic and real-world images. By introducing rich diversity in synthetic scenes, models can better generalize to real data, as highlighted in industrial inspection, robotics, and autonomous driving applications. Unity Perception offers a robust framework for generating vast quantities of randomized, pixel-perfectly labeled data, reducing the annotation burden while enabling comprehensive coverage of rare scenarios and edge cases.arxiv+5

2c. Synthetic Data for Minority-Class Augmentation

Research demonstrates that targeted augmentation with synthetic data can measurably improve the predictive performance of object detection models for underrepresented classes. Tools such as SMOTE and its advanced variants generate synthetic samples in feature space, boosting the presence and diversity of minority-class instances and enabling models to recognize rare patterns. In computer vision, the strategic inclusion of synthetic data has repeatedly yielded performance gains on imbalanced datasets compared to pure real-data training, with some studies showing up to 5% improvement in mAP for minority classes when synthetic images are mixed with real samples. Importantly, mixed datasets frequently outperform real-only datasets, especially in low-resource or long-tailed scenarios, reinforcing the value of synthetic augmentation in supporting minority-class detection.arxiv+6

2d. Applications in Robotics, Autonomous Driving, and Industry

Synthetic datasets play a critical role in fields where exhaustive real-world annotation is impractical, including robotics, industrial inspection, and autonomous vehicle perception. Synthetic data can be efficiently generated for these domains, facilitating the exploration of a more extensive range of conditions—such as rare object types, lighting, and occlusion—than would otherwise be possible with manual labeling. This approach has rapidly gained traction, as seen in Unity Perception workflows, which integrate procedural content generation with automated labeling for scalable, domain-adaptive object detection pipelines.roboflow+5

2e. Research Gap and Motivation

Despite extensive progress, knowledge gaps remain regarding the systematic use of synthetic 3D datasets, particularly those generated via Unity Perception, to address class imbalance in detection tasks for specific object categories. Previous work has validated the positive impact of synthetic augmentation for minority classes; however, the effectiveness of diverse synthetic-only versus real-only and mixed training schemes—specifically for modern models like YOLOv8, DETR, and Mask R-CNN—has not been comprehensively evaluated. This project is therefore motivated to address this gap by investigating whether procedurally synthesized, automatically labeled 3D data can reliably enhance detection performance for underrepresented categories, and under what cnditions synthetic data is most beneficial.

---

3. METHODOLOGY OVERVIEW

3a. Methodology Overview

This project establishes a complete, reproducible pipeline for generating synthetic images using Unity Perception, converting them into YOLO-compatible datasets, and training modern object detection models using synthetic-only, real-only, and mixed datasets. The pipeline integrates Unity-based 3D rendering, Python preprocessing, dataset management, training scripts, and evaluation tools. This section provides a high-level overview of the entire system.

The methodology consists of five core stages:

1. Synthetic data generation with Unity Perception
2. Annotation extraction and conversion into YOLO format
3. Dataset assembly (synthetic, real, mixed)
4. Training modern object detection models

5. Evaluation and analysis

Detailed documentation for Unity setup, randomizers, scene configuration, and dataset conversion is provided separately in the Unity and Python pipeline documents and was iteratively refined across Week 1–4 development logs .

3.b Synthetic Dataset Generation (Unity Perception)

Unity Perception is used to generate labeled images of five object classes: bottle, cup, chair, laptop, and book. A dedicated capture scene is built using:

- FixedLengthScenario to control the number of frames
- BoundingBox2DLabeler for automatic annotation
- Camera, Light, and PrefabPlacement randomizers to introduce variation
- Deterministic seeds for exact reproducibility

These randomizers produce diverse camera angles, lighting conditions, object rotations, and positions, ensuring a wide visual distribution of each class. Unity outputs pairs of:

- step0.camera.png (RGB image)
- step0.frame_data.json (annotations containing instance IDs and bounding boxes)

The complete scene configuration is documented in the Unity pipeline file .

3.c Unity → YOLO Format Conversion

Unity's JSON annotations must be translated into the YOLO (class_id, x_center, y_center, width, height) format. This is performed by the UnityToYOLO.py converter, which:

- Parses each JSON file
- Extracts 2D bounding boxes
- Normalizes coordinates using image resolution
- Writes a .txt label file for each image
- Splits data into train/ and val/ sets
- Generates dataset.yaml and classes.txt

Folder structure and conversion logic follow the Python pipeline documentation .
 Dataset validation was performed using sanity-check scripts that overlay bounding boxes onto images to verify label correctness.

3d Dataset Assembly: Real, Synthetic, and Mixed

Three datasets are prepared to enable controlled experiments:

1. Synthetic-only dataset
 Generated entirely using Unity and converted into YOLO format.

2. Real-only dataset
 Built from a curated COCO 2017 subset containing the five target classes, following extraction procedures described earlier in the weekly logs .

3. Mixed dataset
 A combination of both, ensuring class counts are balanced across synthetic and real samples. Mixed data provides both domain diversity (real textures and noise) and synthetic variation (poses, lighting, randomized camera viewpoints).

All datasets are structured into the standard YOLO directory format:

```None
images/
   train/
   val/
labels/
   train/
   val/
dataset.yaml
```

3e Model Training Pipeline

Model training is performed using YOLOv8 (Ultralytics). The pipeline supports:

- Synthetic-only training

- Real-only training
- Mixed synthetic + real training

Training steps include:

- Initializing YOLOv8n (nano) for compute feasibility
- Specifying dataset YAML files
- Running training loops through Jupyter notebooks (`SyntheticTraining.ipynb` and `MixedTraining.ipynb`)
- Saving logs, weights, predictions, and metrics under `runs/detect/`

Hyperparameters such as batch size, image size, number of epochs, and device settings align with the reproducibility instructions in the Python documentation .

3f Evaluation Methodology

The final evaluation compares all three models on the same real-image test set using metrics:

- Overall accuracy
- Precision
- Recall
- F1-score
- Per-class performance
- Confusion matrices

Evaluation scripts automatically:

- Run inference on each image
- Match predictions to ground truth
- Generate per-class CSVs
- Produce confusion-matrix visualizations

These procedures derive from the test script documentation provided in the repository .

3g Summary of Methodology

The proposed pipeline creates a controlled, fully reproducible workflow spanning:

1. Unity-driven synthetic generation
2. Python-based annotation conversion
3. Dataset structuring
4. YOLOv8 training (synthetic-only, real-only, mixed)
5. Comprehensive evaluation on real data

This layered methodology enables a fair investigation into the impact of synthetic data on underrepresented class performance and forms the basis for the experimental results described in later sections.

Below is a **clean, formal, IEEE-style PART 4 — Unity Perception Dataset Generation**, rewritten **from scratch**, focusing only on the dataset-generation component, and formatted specifically as a polished report section.

(Times New Roman 12, single spacing, IEEE citation markers included; full references can be compiled later.)

---

4. UNITY PERCEPTION DATASET GENERATION

Unity Perception is used in this project to generate a controlled, diverse, and fully reproducible synthetic dataset for five target object categories: **bottle, cup, chair, laptop, and book**. The package provides automated annotation, deterministic randomization, and structured export of metadata, enabling the creation of large volumes of high-quality training samples without manual labeling. All Unity configuration details and custom scripts applied in this project are documented in the Unity pipeline file , and the generation process is recorded across Weeks 1–4 of development .

4a Unity Environment and Scene Structure

The dataset was produced using Unity 2022.3 LTS with the official com.unity.perception package. A dedicated synthetic-capture scene (Perception.unity) was created inside the project's Assets/Scenes/ directory, containing:

1. **A FixedLengthScenario** – controls the number of images generated.
2. **A Perception Camera** – captures RGB images and emits 2D bounding-box annotations.
3. **Randomizers** – introduce procedural variation in lighting, camera configuration, and object placement.
4. **Class-specific prefabs** – high-quality 3D assets stored in Assets/Prefabs/ for each target category.

The Perception scene serves as a deterministic, isolated generation environment where each frame corresponds to a single annotated training sample.

4b Labelers and Automated Annotation

Unity Perception provides built-in labelers that automatically generate training annotations. Two were used:

- **BoundingBox2DLabeler:**
  Generates 2D bounding boxes around each rendered object, normalized to pixel coordinates.
- **RenderedObjectInfoLabeler:**
  Records object instance IDs, visibility information, and metadata required for debugging and conversion.

The camera resolution was fixed at **1024 × 1024** pixels to maintain uniformity across all generated images and ensure compatibility with the YOLO preprocessing pipeline.

4c Randomization for Dataset Diversity

Synthetic images must contain sufficient variation for models to generalize to real-world conditions. To support this, three procedural randomizers were implemented:

4c.1 Camera Randomizer

A custom C# script (documented in the Unity file) adjusts:

- camera distance from the object,
- elevation angle,
- final orientation (always forced to "look at" the object).

This ensures each frame presents a unique viewpoint while maintaining the object fully inside the camera's field of view.

4c.2 Lighting Randomizer

Controls:

- light intensity,
- rotation along three axes.

Varying illumination produces shadows, specular highlights, and scene contrast changes, all of which improve robustness to real-world lighting variability.

4c.3 Prefab Placement Randomizer

Selects and spawns exactly **one object instance** per frame. It randomizes:

- which prefab is used,
- its spatial location,
- its rotation,
- and removes it after each iteration.

It also supports fixed seeding:

```CSharp
useSeed = true;

seed = 1234;
```

This ensures deterministic reproduction of object placement across runs.

All randomizer implementations are detailed in the Unity Perception documentation file .

4d Per-Class Capture Procedure

To avoid occlusion artifacts and ensure uniform distribution, the dataset was generated **one class at a time**, following this procedure:

1. Load the Perception scene.
2. Insert only the prefabs belonging to a single object category (e.g., all cup prefabs) into the PrefabPlacementRandomizer.
3. Set the scenario frame count (e.g., 100 frames).
4. Run the editor in **Play Mode** to generate the captures.
5. Unity outputs a folder structure containing sequential frames:

```None
SyntheticOutput/

   solo/

      sequence.0/

         step0.camera.png

         step0.frame_data.json

      sequence.1/

      ...
```

The procedure is repeated for all five classes, ensuring consistent diversity and identical randomization settings.

4e Output Format and Metadata

Each frame is saved as:

- `step0.camera.png` — the RGB image.
- `step0.frame_data.json` — metadata containing:
    - bounding boxes,
    - object category,
    - instance identifiers,
    - pose and position data,
    - randomizer parameter values,
    - and camera/light configurations.

This structured JSON format is designed for downstream consumption by the Python conversion script (`UnityToYOLO.py`) and allows complete traceability of each rendered sample.

4f Reproducibility Measures

Unity's synthetic generation is fully deterministic when randomizer seeds are fixed. The following measures ensure reproducibility:

- **Fixed scenario seed** in the Perception loop.
- **Fixed random seeds** in the PrefabPlacementRandomizer.
- **Consistent list order** for prefabs in each object category.
- **Identical parameter ranges** for camera and lighting randomizers.
- **Metadata logging** in JSON files for every iteration.

These steps ensure that the TA or any external user can re-generate the exact same synthetic dataset following the reproducibility instructions provided in Week 4 .

4g Summary

Unity Perception enables a high-diversity, annotation-dense synthetic dataset tailored to underrepresented object classes. Through structured scene design, systematic per-class rendering, procedural randomizers, and deterministic seeds, this pipeline provides:

- Reliable, automatically labeled training images
- Broad variation in pose, lighting, and camera viewpoints
- Zero manual annotation effort
- Guaranteed reproducibility across environments

These Unity-generated images form the foundation for the YOLO dataset conversion, model training, and evaluation workflows discussed in the next sections.

.

5. PYTHON PIPELINE, DATASET CONVERSION, AND TRAINING PROCEDURE

5. Overview

This section describes the full Python-side workflow responsible for:

1. **Converting Unity Perception output into YOLO-compatible datasets**,
2. **Preparing synthetic, real, and mixed datasets**, and
3. **Training YOLOv8 models using these datasets**.

5a Python Environment Setup

A reproducible Python environment is required to ensure that dataset conversion, YOLO training, and evaluation behave identically across machines. The environment setup follows the reproducibility documentation .

5a.1 Virtual Environment

```Shell
python -m venv .venv
```

Activate on Windows:

```Shell
.venv\Scripts\activate
```

Activate on macOS/Linux:

```Shell
source .venv/bin/activate
```

5a.2 Install Required Packages

The project uses a pinned set of dependencies:

```Shell
pip install -r requirements.txt
```

Core packages include:

- **Ultralytics YOLOv8**
- **PyTorch**, **Torchvision**
- **OpenCV**, **Pillow**
- **pandas**, **numpy**, **plotly**
- **tqdm**, **PyYAML**

These versions provide stable operation across training, label conversion, evaluation, and plotting.

5b Unity → YOLO Dataset Conversion Pipeline

Unity Perception exports images and JSON annotation files for each synthetic frame. However, YOLOv8 requires a specific directory structure and TXT-based bounding box format. The conversion process is handled by the script `UnityToYOLO.py`, documented fully in the Python pipeline files .

5b.1 Input Format from Unity

Unity produces:

```None
sequence.X/

    step0.camera.png

    step0.frame_data.json
```

Each JSON file contains:

- bounding box coordinates,
- object class index,
- object instance ID,
- camera metadata,
- and randomizer state.

5b.2 Conversion Steps

The conversion script performs the following:

(1) Parse Unity JSON

Extract bounding boxes from each frame's frame_data.json.

(2) Normalize Bounding Boxes

Convert from pixel coordinates to YOLO format:

```None
class_id  x_center  y_center  width  height
```

(3) Write YOLO Label Files

One `.txt` file per image:

```None
0 0.512 0.423 0.200 0.310
```

(4) Organize Dataset Structure

Automatically creates:

```None
yolo_synthetic_dataset/

   images/train

   images/val

   labels/train

   labels/val

   dataset.yaml
```

(5) Train/Validation Split

Script supports --val-ratio, e.g.:

```Shell
python UnityToYOLO.py --val-ratio 0.1 -v
```

(6) Class Name Canonicalization

During Week 3, inconsistencies (e.g., *books* vs *book*) were resolved by enforcing a standardized class list inside the converter .

5b.3 Dataset Sanity Checks

A quick Python script overlays bounding boxes on images to verify correctness. This script is described in both the reproducibility and Python pipeline docs .

5c Real and Mixed Dataset Preparation

The pipeline uses **three datasets**:

1. Synthetic Dataset

Fully generated via Unity and converted to YOLO format.

2. Real Dataset

Extracted from curated COCO 2017 subsets (bottle, cup, chair, laptop, book) and re-structured into YOLO format during Week 2 and Week 3 development .

3. Mixed Dataset

A combination of:

- Real images (COCO 2017 subset)
- Unity-generated synthetic images

The mixed dataset required careful class balancing, documented in Week 4 logs, where re-rendering was performed to address class count disparities .

Both synthetic and mixed datasets follow this structure:

```
None
datasets/

   yolo_synthetic_dataset/

   yolo_mixed_dataset/
```

Each contains:

```None
images/train

images/val

labels/train

labels/val

dataset.yaml
```

5d YOLOv8 Training Procedure

Model training is performed using the **Ultralytics YOLOv8** framework via Jupyter notebooks:

- SyntheticTraining.ipynb
- MixedTraining.ipynb

These notebooks implement reproducible training procedures described in the Python reproducibility document .

5d.1 Model Variants Trained

Three core models were trained:

(A) Synthetic-Only Model

Trained exclusively on Unity-generated images.
Used to evaluate the effect of synthetic data alone.

(B) Real-Only Model

Trained on the COCO subset only.
Acts as a baseline for natural images.

(C) Mixed Dataset Model

Trained on both synthetic and real images.
Serves as the **primary experimental model** and achieves the best performance.

5d.2 Training Configuration

Common hyperparameters:

- **Model:** yolov8n.pt (YOLO Nano, chosen for CPU feasibility)
- **Image size:** 640 × 640
- **Epochs:** typically 50
- **Batch size:** 8
- **Device:** CPU
- **Optimizer:** SGD (default YOLO settings)
- **Fixed random seed:** 42

Example training command (automatically executed in notebooks):

```python
model.train(

    data="datasets/yolo_synthetic_dataset/dataset.yaml",

    epochs=50,

    imgsz=640,

    batch=8,

    device="cpu",

    name="synth_only_run"

)
```

5d.3 Output Artifacts

Each training run produces:

```
runs/detect/<run_name>/

   weights/ (best.pt, last.pt)
```

```
results.csv

results.png

confusion_matrix.png

predictions/
```

These outputs are later consumed by evaluation scripts discussed in the next section.

5e Evaluation Pipeline

Evaluation uses dedicated Python scripts documented in the test-script file .

Scripts include:

- mixed_model_test.py
- pure_yolo_test.py
- synth_model_test.py

Each script:Runs inference on a folder-structured test dataset

1. Records per-image predictions
2. Computes per-class precision, recall, and F1
3. Generates a confusion matrix (Plotly HTML)
4. Saves a summary JSON and CSV metrics

This pipeline allowed consistent comparison between the synthetic-only, real-only, and mixed dataset models.

5f Summary

The Python-side pipeline forms the backbone of this project by converting Unity's structured synthetic output into a YOLO-ready dataset, organizing real and mixed datasets, and providing a complete training and evaluation workflow. Together with deterministic Unity generation and controlled training configurations, this pipeline ensures:

- Strict reproducibility of experiments
- Fair comparison among dataset configurations
- Seamless integration of synthetic and real imagery
- Reliable assessment of synthetic-data contributions

The next section presents results obtained from these models and discusses the impact of synthetic data on underrepresented class performance.

---

6. PYTHON PIPELINE, DATASET CONVERSION, AND TRAINING PROCEDURE

6. Overview

This section explains the complete Python-side workflow used after Unity dataset generation. It covers:

1. Conversion of Unity Perception output into YOLO-format datasets,
2. Preparation of synthetic, real, and mixed datasets, and
3. Training YOLOv8 models using these datasets.

6a Python Environment Setup

A controlled virtual environment ensures that dataset conversion, training, and evaluation remain consistent across machines.

6a.1 Create Virtual Environment

```shell
python -m venv .venv
```

6a.2 Activate Environment

Windows:

```shell
.venv\Scripts\activate
```

macOS/Linux:

```shell
source .venv/bin/activate
```

6a.3 Install Dependencies

```shell
pip install -r requirements.txt
```

Core packages include:

- Ultralytics YOLOv8
- PyTorch and Torchvision
- OpenCV, Pillow, numpy, pandas, plotly
- tqdm, PyYAML

These ensure stable operation for training, label processing, and evaluation.

6b Unity → YOLO Dataset Conversion Pipeline

Unity exports per-frame `.png` images and `frame_data.json` label files. YOLO, however, requires a very specific text-based annotation format and directory structure.
This conversion is handled by the script `UnityToYOLO.py`, fully documented in the Python pipeline file .

6b.1 Unity Output Format

Unity Perception produces folders like:

```None
sequence.0/
   step0.camera.png
   step0.frame_data.json
sequence.1/
   ...
```

Each JSON includes:

- bounding boxes
- object class name
- instance-level metadata
- camera & randomizer states

6b.2 Conversion Process

(1) Parse JSON Annotations

Extract bounding boxes defined in pixel coordinates.

(2) Normalize for YOLO Format

YOLO requires:

```None
class_id  x_center  y_center  width  height
```

All normalized to [0, 1].

(3) Generate YOLO Label Files

For each `image.png`, a corresponding `image.txt` is written.

(4) Automatic Folder Structuring

The script produces:

```None
yolo_synthetic_dataset/
   images/{train,val}
   labels/{train,val}
   dataset.yaml
   classes.txt
```

(5) Validation Split

Configured using:

```Shell
--val-ratio 0.1
```

(6) Class Name Normalization

During Week 3, inconsistencies (e.g., “books” vs “book”) were fixed inside the converter, ensuring dataset uniformity .

6b.3 Dataset Sanity Verification

Bounding box overlay scripts from the reproducibility and Python pipeline docs draw red rectangles on images to visually confirm annotation correctness before training.

6c Real and Mixed Dataset Preparation

Three datasets were prepared:

(1) Synthetic Dataset

Generated entirely in Unity and converted to YOLO format.

(2) Real Dataset

Sourced from COCO 2017, containing the five target classes.
Extraction and preparation occurred primarily during Week 2 and Week 3 work .

(3) Mixed Dataset

Contains both:

- real COCO images
- synthetic Unity images

Week 4 development logs document additional synthetic captures made to rebalance class counts in the mixed dataset .

Final structure:

```None
datasets/
   yolo_synthetic_dataset/
   yolo_mixed_dataset/
```

Each includes:

```None
images/train, images/val
labels/train, labels/val
dataset.yaml
```

6d YOLOv8 Training Procedure

Two Jupyter notebooks orchestrate the training:

- SyntheticTraining.ipynb
- MixedTraining.ipynb

Training steps adhere to the reproducibility guidelines in the Python instructions .

6d.1 Model Variants Trained

A. Synthetic-Only Model

Evaluates the effect of Unity-generated data alone.

B. Real-Only Model

Acts as a baseline using natural images.

C. Mixed Model (Primary Experimental Model)

Uses both real and synthetic images.
This model achieved the best results.

6d.2 Training Parameters

Consistent across all runs:

- Model: YOLOv8n (nano variant)
- Image Size: 640×640
- Epochs: 50
- Batch Size: 8
- Optimizer: YOLO default (SGD)
- Device: CPU
- Seed: 42 (fixed for reproducibility)

Example:

```python
model.train(
    data="datasets/yolo_synthetic_dataset/dataset.yaml",
    epochs=50,
    imgsz=640,
    batch=8,
    device="cpu",
    name="synth_only_run"
)
```

6d.3 Output Artifacts

Each run produces:

```None
runs/detect/<run_name>/
   weights/best.pt
   weights/last.pt
   results.csv
   results.png
   confusion_matrix.png
   predictions/
```

These feed into the evaluation stage.

6e Evaluation Pipeline

Evaluation scripts (documented in the test-script file) include:

- `mixed_model_test.py`
- `pure_yolo_test.py`
- `synth_model_test.py`

Each script:

1. Runs inference on a folder-structured real test dataset
2. Generates per-image results
3. Computes precision, recall, and F1 scores
4. Creates a confusion matrix (Plotly HTML)
5. Saves metrics in CSV and JSON formats

This ensures strict comparability across all trained models.

6f Summary

The Python pipeline converts Unity's structured synthetic data into a ready-to-train YOLO dataset, organizes real and mixed datasets, and executes all training and evaluation steps. Together with deterministic Unity generation, fixed seeds, and consistent hyperparameters, this process ensures:

- Fully reproducible experiments
- Fair comparisons between dataset configurations
- Reliable conclusions about the impact of synthetic data on underrepresented classes

The next section presents the experimental results and evaluates how synthetic data influenced detection performance.

---

7. RESULTS AND ANALYSIS

7. Results and Analysis

Three YOLOv8 models were evaluated on a real-image test set consisting of the five target classes (bottle, cup, chair, laptop, book). The evaluation pipeline and metrics follow the scripts documented in the project test-suite , and the numeric results originate from the consolidated project README .

7a Overall Accuracy

Model	Description	Accuracy
Mixed Model (A)	Real + Synthetic	0.9900
Real-Only Model (B)	COCO subset only	0.6915
Synthetic-Only Model (C)	Unity Perception only	0.1194

Mixed training achieved the highest accuracy, confirming that synthetic data effectively supplements missing examples and reduces class imbalance.

7b Class-wise Performance

The mixed dataset model achieved near-perfect precision and recall across all five classes.
In comparison:

- The real-only model performed well on structurally simple, high-contrast classes (chair, laptop) but struggled with low-contrast categories (book, cup).
- The synthetic-only model showed unstable results due to domain gap effects but demonstrated early formation of class-specific representations, especially for shape-dominant objects like chairs.

These patterns match observations documented in the training logs and evaluation summaries .

7c Key Observations

1. Synthetic data meaningfully boosts minority-class performance when combined with real images.
2. Mixed datasets consistently outperform both pure real and pure synthetic datasets, proving that Unity-generated variation fills gaps in real-world image diversity.
3. The domain gap prevents synthetic-only training from achieving strong real-world accuracy, but synthetic samples still contribute valuable structural variation.
4. The experiment validates the project's hypothesis: *synthetic 3D data is an effective tool for improving recognition of underrepresented objects*

7d Summary

The mixed model demonstrated unmistakable performance gains across all metrics, particularly recall, confirming that synthetic augmentation is a practical solution to dataset imbalance. Real-only training lacked sufficient variation for certain classes, while synthetic-only training lacked realism—but together they provided the strongest results.

---

8. CONCLUSION AND FUTURE WORK

8a Conclusion

This project investigated whether Unity Perception–generated synthetic data can improve detection of underrepresented object classes when training modern object detection models. The full pipeline—from Unity scene generation to YOLO training and evaluation—was made fully reproducible using deterministic seeds, controlled randomizers, and structured Python scripts, as documented throughout the project logs .

Experimental results clearly show that:

- The mixed dataset model (synthetic + real) achieved the highest overall accuracy (0.99) and near-perfect class-wise precision and recall.
- The real-only model performed inconsistently due to class imbalance and insufficient variation.
- The synthetic-only model showed limitations due to the domain gap but contributed valuable diversity when combined with real data.

Overall, these findings confirm that synthetic 3D data is an effective and practical strategy to supplement real-world datasets, particularly when certain classes are scarce or costly to collect.

8b Future Work

Several extensions can further strengthen the pipeline:

(1) Improve Photorealism

Using HDRP, PBR materials, or texture-randomization could reduce the domain gap and increase synthetic-only performance.

(2) Expand Object Categories

More classes and additional object variations (shape, color, texture) would help evaluate generalization across diverse categories.

(3) Domain Adaptation Techniques

Methods such as CycleGAN, style transfer, or feature-level alignment could help bridge the gap between synthetic and real images.

(4) Larger-Scale Experiments

Training higher-capacity YOLO models (YOLOv8l/x) or DETR variants may reveal further benefits from synthetic augmentation.

(5) Temporal or Multi-Object Scenes

Extending the pipeline to segmentations, multi-object scenes, or video sequences could further broaden applicability.

8c Final Remarks

This work demonstrates that synthetic 3D datasets are not a replacement for real data, but they are a highly effective complement—particularly for rare, underrepresented, or difficult-to-collect object classes. Unity Perception and YOLOv8 form a powerful combination for building scalable, controllable, and reproducible training pipelines in modern computer vision.

References

K. Oksuz, et al., "Imbalance Problems in Object Detection: A Review," arXiv:1909.00169, 2019.arxiv
"Class Imbalance - an overview," ScienceDirect, 2023.sciencedirect
K. Oksuz, et al., "Imbalance Problems in Object Detection: A Review," IEEE Xplore, 2020.ieeexplore.ieee
M. Tomaszewski, et al., "Effectiveness of Data Resampling in Mitigating Class Imbalance for Object Detection," CEUR-WS, 2020.arxiv
L. T. Niño, H. A. A. Gardi, "Synthetic-to-Real Object Detection using YOLOv11 and Domain Randomization Strategies," arXiv:2509.15045, 2025.arxiv
"Improving Predictions on Highly Unbalanced Data Using Synthetic Data," arXiv:2507.16419, 2025.arxiv
"Evaluating the Impact of Synthetic Data on Object Detection Tasks in Real-World Applications," arXiv:2503.09803, 2024.arxiv
"Using Unity Perception to train an object detection model with synthetic data," roboflow.com, Nov. 2025.roboflow
"Effectiveness of Data Resampling in Mitigating Class Imbalance for Object Detection," CEUR-WS, 2020.ceur-ws
X. Zhu et al., "Domain Randomization for Object Detection in Industrial Manufacturing," arXiv:2506.07539, 2025.arxiv
"Synthetic Data Counters Lack of Diversity," National Cancer Institute, 2024.cancer
"The Impact of Synthetic Data on Object Detection Model Performance," arXiv:2510.12208, 2025.arxiv
"Perception Synthetic Data Tutorial," Unity Documentation, 2019.unity3d

R. Adam, “Synthetic Training Data Generation and Domain Randomization for Object Detection,” IEEE Xplore, 2022.[ieeexplore.ieee]

G. Wei et al., "An improved and random synthetic minority oversampling technique," Knowledge-Based Systems, 2022.[sciencedirect]

“PCGOD: Enhancing Object Detection With Synthetic Data for Scarce Scenarios,” IEEE Xplore, 2023.[ieeexplore.ieee]

“Unity-Technologies/SynthDet,” GitHub, 2020.[github]

A. Westerski, “Synthetic Data for Object Detection with Neural Networks,” ACM, 2024.[acm]

“Improving massively imbalanced datasets in machine learning with synthetic data,” Gretel, 2022.[gretel]

1. **https://arxiv.org/pdf/1909.00169.pdf**
2. **https://www.sciencedirect.com/topics/computer-science/class-imbalance**
3. **https://ieeexplore.ieee.org/iel7/34/9527405/09042296.pdf**
4. **https://arxiv.org/html/2403.07113v1**
5. **https://ceur-ws.org/Vol-3628/paper14.pdf**
6. **https://arxiv.org/abs/2509.15045**
7. **https://blog.roboflow.com/unity-perception-synthetic-dataset-tutorial/**
8. **https://arxiv.org/abs/2506.07539**
9. **https://docs.unity3d.com/Packages/com.unity.perception@1.0/manual/Tutorial/TUTORIAL.html**
10. **https://ieeexplore.ieee.org/document/9987772/**
11. **https://github.com/Unity-Technologies/SynthDet**
12. **https://arxiv.org/abs/2507.16419**
13. **https://arxiv.org/html/2503.09803v1**
14. **https://www.cancer.gov/about-nci/organization/cbiit/news-events/blog/2024/synthetic-data-helps-counter-lack-diversity-data**
15. **https://arxiv.org/html/2510.12208v1**
16. **https://www.sciencedirect.com/science/article/abs/pii/S0950705122004002**
17. **https://ieeexplore.ieee.org/document/11009168/**
18. **https://gretel.ai/blog/improving-massively-imbalanced-datasets-in-machine-learning-with-synthetic-data**
19. **https://dl.acm.org/doi/full/10.1145/3637064**
20. **https://github.com/kemaloksuz/ObjectDetectionImbalance**