Synthetic-Data-Generation-for-Computer-Vision-AI-ML

1 Large Scale (Synthetic Data Generation) for Masked image segmentation using Blender 3D software for DEEP LEARNING - Computer Vision(YOLO)

Abhishek Verma
CSE-AIML OIST-RGPV
Bhopal, India
0105AL221012@oriental.ac.in

Divyansh Kande
CSE-AIML OIST-RGPV
Bhopal, India
0105AL221069@oriental.ac.in

Atharv Jharbade
CSE-AIML OIST-RGPV
Bhopal, India
0105AL221044@oriental.ac.in

Akshay Deshmukh
CSE-AIML OIST-RGPV
Bhopal, India
0105AL221022@oriental.ac.in

Abstract:

This project explores the use of synthetic data generation for deep learning and computer vision, specifically for polygon-based segmentation tasks using YOLO annotations. In many real-world applications, acquiring labelled data for training deep learning models is a time-consuming, expensive, and often impractical process. Synthetic data generation provides a viable solution to this challenge, enabling the creation of diverse, high-quality datasets to enhance model training. The focus of this study is on using Blender 3D software to generate large-scale synthetic datasets for masked image segmentation, a critical task in computer vision. Through 3D rendering, this approach simulates real-world environments, producing realistic images that can be used to train segmentation models effectively. The project also examines the use of YOLO (You Only Look Once) annotations, a powerful tool for object detection, to label the generated images. By employing methods such as image augmentation and generative models like GANs (Generative Adversarial Networks), synthetic data can be further enriched, offering an increased variety of examples for deep learning models. The use of synthetic data addresses issues related to data scarcity, facilitating better model performance and generalization. This paper demonstrates the importance and effectiveness of synthetic data in overcoming the limitations of real-world data acquisition for computer vision tasks, ultimately contributing to the advancement of AI and machine learning in practical applications.

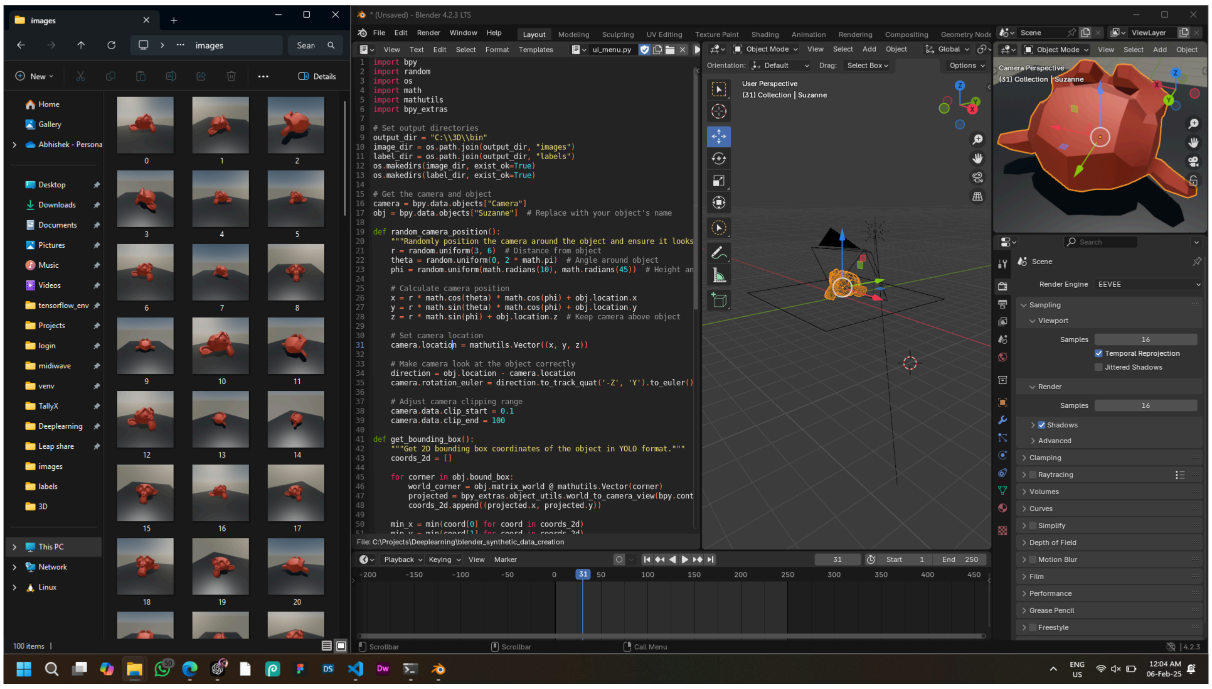
1.Introduction

This project demonstrates the process of synthetic data generation for deep learning and computer vision models, with a focus on polygon-based segmentation using YOLO annotations. Synthetic data generation plays a vital role in training models when real-world labeled data is scarce, expensive, or hard to obtain. By simulating realistic datasets through algorithms or 3D rendering, we can augment training sets for better model accuracy and generalization.

What is Synthetic Data Generation?

Synthetic data generation refers to the creation of artificial data designed to simulate real-world data. In the context of deep learning and computer vision, synthetic data is generated for tasks such as image segmentation, object detection, and classification. It is created using a variety of methods, including:

3D Rendering:



```
import bpy
import random
import os
import math
import mathutils
import bpy_extras

# Set output directories
output_dir = "C:\\3D\\bin"
image_dir = os.path.join(output_dir, "images")
label_dir = os.path.join(output_dir, "labels")
os.makedirs(image_dir, exist_ok=True)
os.makedirs(label_dir, exist_ok=True)

# Get the camera and object
camera = bpy.data.objects["Camera"]
obj = bpy.data.objects["Suzanne"]  # Replace with your object's name

def random_camera_position():
    """Randomly position the camera around the object and ensure it looks
    r = random.uniform(3, 6)  # Distance from object
    theta = random.uniform(0, 2 * math.pi)  # Angle around object
    phi = random.uniform(math.radians(10), math.radians(45))  # Height an

    # Calculate camera position
    x = r * math.cos(theta) * math.cos(phi) + obj.location.x
    y = r * math.sin(theta) * math.cos(phi) + obj.location.y
    z = r * math.sin(phi) + obj.location.z  # Keep camera above object

    # Set camera location
    camera.location = mathutils.Vector((x, y, z))

    # Make camera look at the object correctly
    direction = obj.location - camera.location
    camera.rotation_euler = direction.to_track_quat('-Z', 'Y').to_euler()

    # Adjust camera clipping range
    camera.data.clip_start = 0.1
    camera.data.clip_end = 100

def get_bounding_box():
    """Get 2D bounding box coordinates of the object in YOLO format."""
    coords_2d = []

    for corner in obj.bound_box:
        world_corner = obj.matrix_world @ mathutils.Vector(corner)
        projected = bpy_extras.object_utils.world_to_camera_view(bpy.cont
        coords_2d.append((projected.x, projected.y))

    min_x = min(coord[0] for coord in coords_2d)
```

File: C:\Projects\Deeplearning\blender_synthetic_data_creation

Computer-generated images are produced using simulation tools, often using detailed models to simulate real-world environments. Augmentation: Manipulating existing real-world data (e.g., flipping, rotating, color-changing) to create new training examples. Generative Models: Deep learning models, such as GANs (Generative Adversarial Networks), create entirely new images that resemble real data. Benefits of Synthetic Data in Deep Learning and Computer Vision Overcoming Data Scarcity: Synthetic data is crucial when real-world data is limited, as it allows you to create large, diverse datasets for training.

Project Description:-

In this project, synthetic data is used for polygon segmentation tasks in computer vision. The project involves:

Training a YOLOv8 Model:-

The model is trained on synthetic data, where each object is represented as a polygon rather than a simple bounding box

Displaying Predictions: The YOLO model's predictions are visualized by drawing polygons over objects detected in images.

Real-World Application: This method of using synthetic data can be used to generate training data for real-world problems such as autonomous vehicles, medical imaging, and industrial robotics.

How Does This Project Work?:-

The model is trained using YOLO annotations in the form of polygons (not just bounding boxes).

The system reads the image and label files, where labels define the polygons representing each object.

These polygons are drawn over the image, simulating the detection of objects.

The trained model can be deployed to predict objects and draw polygons in real-time on new images.

Synthetic Data Generation Applications in the Future as deep learning and computer vision continue to evolve, synthetic data generation will become increasingly important.

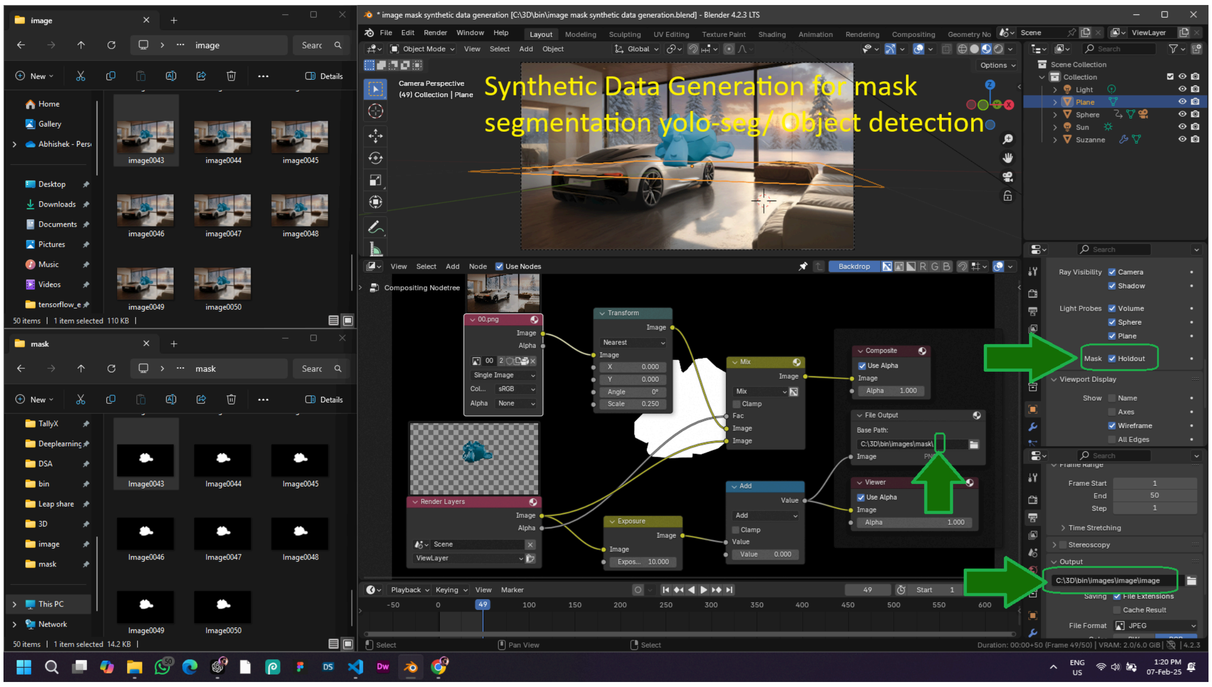
Methodology

#Here, We Using Blender because it has very vast potential for 3D renderings perspectives and also for Synthetic Data generation...

- Imp Notes:-

1. To hide a object select object and and enable Mask -> HOLDOUT
2. Don't set any name for compositor File output saving path
3. background = Film > Transparent

4. object to be trained should be white(255,255,255) and background(0,0,0)
5. name of both rendered image compositor image should be same, during rendering
6. images must save in 2 different folders named as (Images & Mask)



Base Path: C:\3D\bin\images\mask\

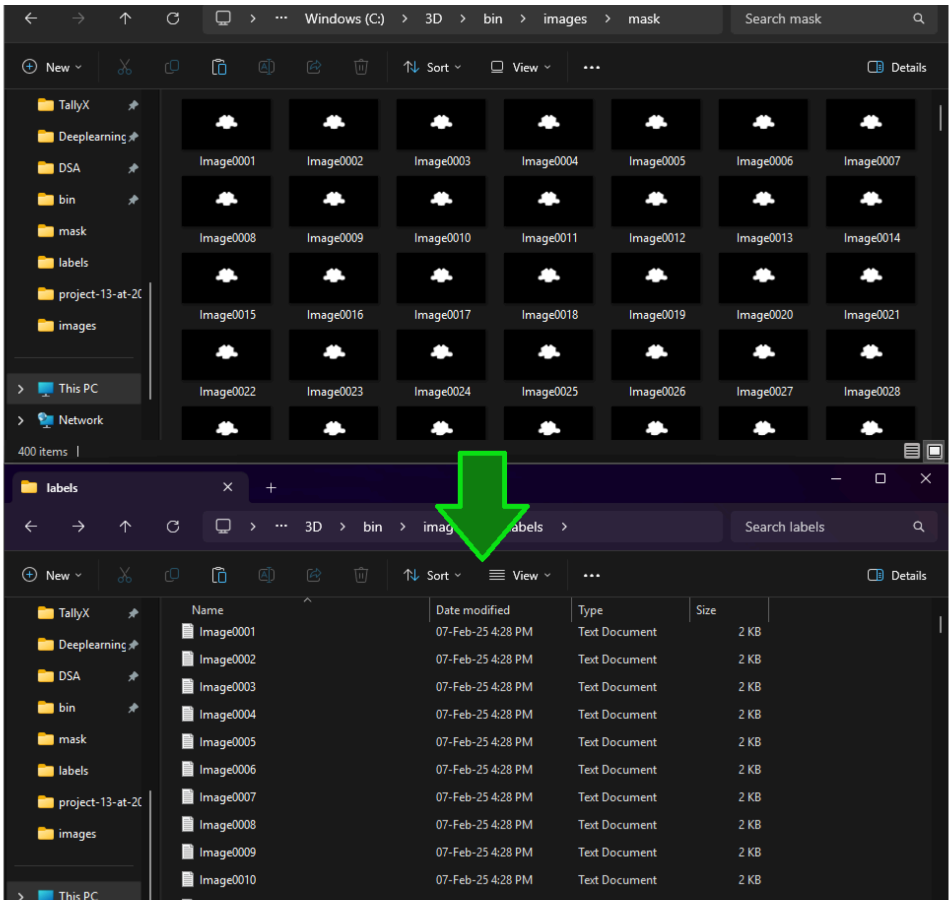
Mask ✓ Holdout

Frame Start 1
End 50
Step 1

Output C:\3D\bin\images\image\image

File Format JPEG

#Convert Masked-image to polygons annotations .txt file/black = background/ white = object masked



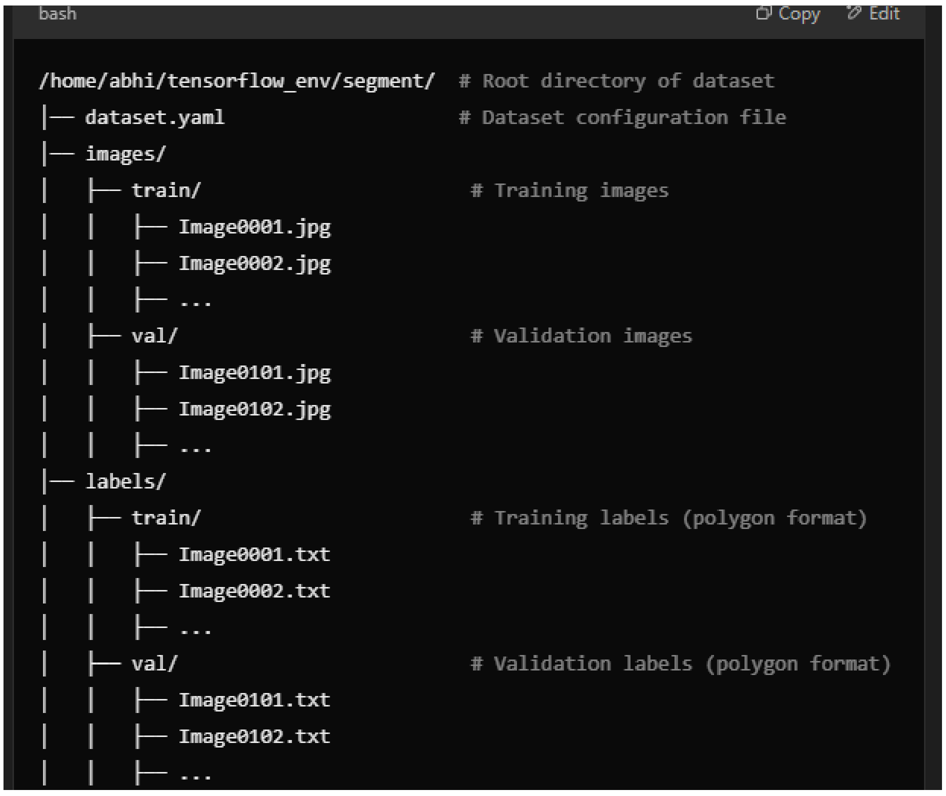
Name	Date modified	Type	Size
Image0001	07-Feb-25 4:28 PM	Text Document	2 KB
Image0002	07-Feb-25 4:28 PM	Text Document	2 KB
Image0003	07-Feb-25 4:28 PM	Text Document	2 KB
Image0004	07-Feb-25 4:28 PM	Text Document	2 KB
Image0005	07-Feb-25 4:28 PM	Text Document	2 KB
Image0006	07-Feb-25 4:28 PM	Text Document	2 KB
Image0007	07-Feb-25 4:28 PM	Text Document	2 KB
Image0008	07-Feb-25 4:28 PM	Text Document	2 KB
Image0009	07-Feb-25 4:28 PM	Text Document	2 KB
Image0010	07-Feb-25 4:28 PM	Text Document	2 KB

Getting Started Prerequisites Python 3.x OpenCV (cv2) NumPy YOLOv8 Model (Pretrained or Custom Trained) Labels in YOLO format (polygon-based) Installation

#Code to convert mask to polygons annotations:

```
import cv2

import numpy as np

import os

def mask_to_polygon(mask_image, epsilon=0.002):  # Use adaptive polygon
approximation

    # Read the mask in grayscale

    img = cv2.imread(mask_image, cv2.IMREAD_GRAYSCALE)

    # Apply binary threshold
    _, binary_mask = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)


    # Find contours with hierarchy detection

    contours, _ = cv2.findContours(binary_mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    polygons = []

    for contour in contours:

        # Approximate the contour to reduce points adaptively

        epsilon_value = epsilon * cv2.arcLength(contour, True)

        approx = cv2.approxPolyDP(contour, epsilon_value, True)


        # Normalize points to YOLO format

        normalized_points = [(point[0][0] / img.shape[1], point[0][1] /
img.shape[0]) for point in approx]


        # Ensure the polygon is closed

        if len(normalized_points) > 2 and normalized_points[0] !=
normalized_points[-1]:

            normalized_points.append(normalized_points[0])

        # Format as YOLO annotation

        polygons.append(" ".join([f"{x} {y}" for x, y in normalized_points]))


    return polygons


# Input/output folders

input_folder = "C:\\3D\\bin\\images\\mask"

output_folder = "C:\\3D\\bin\\images\\label"

# Ensure the output folder exists

os.makedirs(output_folder, exist_ok=True)

# Process all mask images

for mask_image in os.listdir(input_folder):

    if mask_image.endswith(".png"):

        mask_image_path = os.path.join(input_folder, mask_image)

        # Convert mask to YOLO polygons

        polygons = mask_to_polygon(mask_image_path, epsilon=0.002)

        # Save the polygons as YOLO labels
        label_file_path = os.path.join(output_folder,
os.path.splitext(mask_image)[0] + ".txt")
        with open(label_file_path, "w") as f:
            for polygon in polygons:
                f.write(f"0 {polygon}\n")

        print(f"Saved annotations for {mask_image}")

print("Finished processing all images.")
```

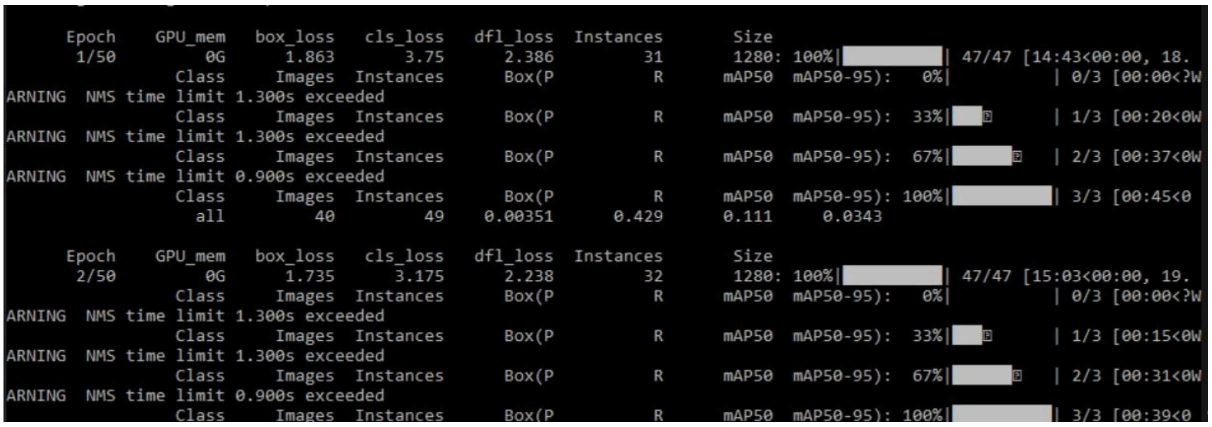
Directory Structure:



```bash
/home/abhi/tensorflow_env/segment/  # Root directory of dataset
|── dataset.yaml                    # Dataset configuration file
|── images/
|   ├── train/                       # Training images
|   |   ├── Image0001.jpg
|   |   ├── Image0002.jpg
|   |   ├── ...
|   ├── val/                         # Validation images
|   |   ├── Image0101.jpg
|   |   ├── Image0102.jpg
|   |   ├── ...
|── labels/
|   ├── train/                       # Training labels (polygon format)
|   |   ├── Image0001.txt
|   |   ├── Image0002.txt
|   |   ├── ...
|   ├── val/                         # Validation labels (polygon format)
|   |   ├── Image0101.txt
|   |   ├── Image0102.txt
|   |   ├── ...
```

#Train yolo model(neural network) using image-segmentation-polygons labels dataset:

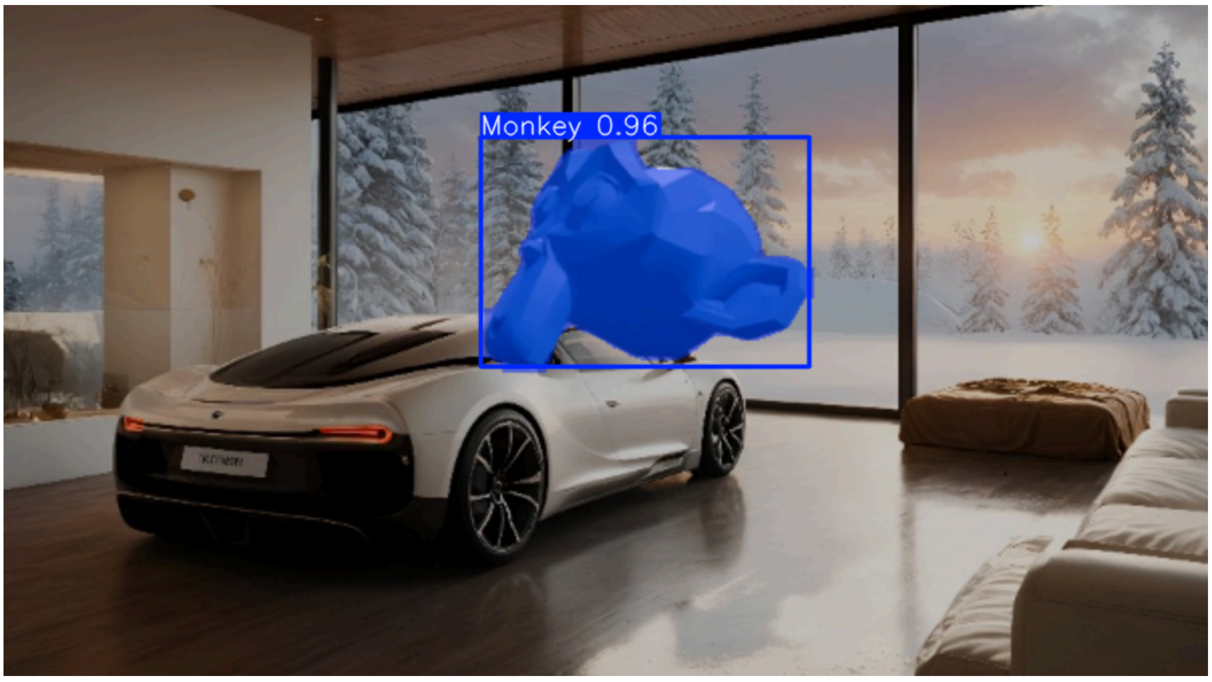
after training, trained model present in same_dir > runs > train > weights > last.pt



```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       1/50         0G      1.863       3.75      2.386         31       1280: 100%|██████████| 47/47 [14:43<00:00, 18.
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95):   0%|          | 0/3 [00:00<?W
ARNING  NMS time limit 1.300s exceeded
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95):  33%|███▎      | 1/3 [00:20<0W
ARNING  NMS time limit 1.300s exceeded
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95):  67%|██████▋   | 2/3 [00:37<0W
ARNING  NMS time limit 0.900s exceeded
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:45<0
                   all         40         49    0.00351      0.429      0.111     0.0343

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       2/50         0G      1.735      3.175      2.238         32       1280: 100%|██████████| 47/47 [15:03<00:00, 19.
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95):   0%|          | 0/3 [00:00<?W
ARNING  NMS time limit 1.300s exceeded
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95):  33%|███▎      | 1/3 [00:15<0W
ARNING  NMS time limit 1.300s exceeded
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95):  67%|██████▋   | 2/3 [00:31<0W
ARNING  NMS time limit 0.900s exceeded
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:39<0
```

TEST AND RESULTS

#Accurate Image Prediction with masked segment



Video Link for Object Detection with realtime masked segmentation:
https://drive.google.com/file/d/1zAA1UDMAOPsJjBMF1pJJZr_XO47UGwWs/view?usp=sharing

Conclusion:

In conclusion, this project demonstrates the critical role of synthetic data generation in advancing deep learning and computer vision models, specifically in the context of masked image segmentation. By leveraging Blender 3D software for rendering realistic datasets and employing YOLO annotations for object detection, the project effectively addresses the challenges of data scarcity in training models. Synthetic data offers a promising alternative to real-world datasets, which are often limited, expensive, and difficult to acquire. Through the use of 3D rendering, the project not only produces highly accurate training data but also enhances the diversity and complexity of datasets, contributing to improved model performance and generalization.

Utilizing powerful tools such as Python, Ultralytics Label Studio, and the YOLOv8 framework, the study demonstrates how combining 3D graphics with advanced machine learning tools can significantly optimize the training process. Additionally, the implementation of the project on a system with Intel Core i5-12500H, 16 GB RAM, and an RTX 4050 GPU proves that synthetic data generation can be performed efficiently on modern hardware configurations, making it accessible to a wider range of researchers and practitioners.

Ultimately, this research highlights the value of synthetic data in overcoming traditional limitations in computer vision tasks and showcases how it can be a game-changer in real-world applications, particularly where real data is difficult to obtain. The methods demonstrated in this study provide a scalable solution for advancing AI-driven image segmentation and object detection tasks in various industries.

Future Scope:

While this project successfully demonstrates the potential of synthetic data generation for computer vision tasks like masked image segmentation, there are several avenues for future research and development. One major direction is the further refinement of 3D rendering techniques to produce even more realistic datasets. As 3D graphics and simulation tools like Blender continue to evolve, the accuracy and diversity of synthetic datasets will improve, making them even more beneficial for training advanced machine learning models.

Additionally, exploring the integration of other generative models, such as GANs (Generative Adversarial Networks), could enhance the diversity of the generated images, allowing for the creation of highly varied and complex datasets. This would be especially useful in applications where variability is essential, such as autonomous vehicles, healthcare imaging, and industrial defect detection.

Another promising area is the use of domain adaptation techniques to bridge the gap between synthetic and real-world data. By using techniques like domain randomization or fine-tuning, synthetic data can be made more applicable to real-world scenarios, improving the generalization of models trained on synthetic datasets.

Moreover, future work could involve optimizing the pipeline for larger-scale data generation, utilizing distributed computing or cloud-based platforms to further speed up the process and handle even larger datasets. Lastly, the application of this synthetic data generation approach could be extended to other computer vision tasks, such as object detection, action recognition, and facial recognition, contributing to a wide range of industries, including robotics, virtual reality, entertainment and healthcare.

Applications and Usability:

Cost-Effective:-

Creating large synthetic data is often cheaper than collecting, annotating, and cleaning real-world data.

Data Diversity:-

Synthetic data allows for fine-grained control over the dataset, ensuring a wide variety of scenarios, object variations, and conditions.

Handling Imbalanced Datasets:-

Synthetic data can be used to balance datasets, ensuring that underrepresented classes or scenarios receive more attention during training.

References:

1. **YOLOv8**. (n.d.). *Ultralytics YOLOv8 Documentation*. Retrieved from https://github.com/ultralytics/yolov8
2. **Blender 4.2**. (2025). *Blender: A 3D Creation Suite*. Retrieved from https://www.blender.org
3. **Python**. (n.d.). *Python Programming Language*. Retrieved from https://www.python.org
4. **Ultralytics Label Studio**. (n.d.). *Label Studio: Data Labeling Tool for Machine Learning*. Retrieved from https://github.com/Label-Studio/label-studio
5. **NumPy**. (2025). *NumPy Documentation*. Retrieved from https://numpy.org
6. **Pandas**. (2025). *Pandas Documentation*. Retrieved from https://pandas.pydata.org
7. **Keras**. (2025). *Keras Documentation*. Retrieved from https://keras.io
8. **PyTorch**. (2025). *PyTorch Documentation*. Retrieved from https://pytorch.org
9. **TensorFlow**. (2025). *TensorFlow Documentation*. Retrieved from https://www.tensorflow.org
10. **My Github Repository Abhii0007**. (2025). *Synthetic Data Generation for Computer Vision AI/ML*. GitHub. Retrieved from https://github.com/Abhii0007/Synthetic-Data-Generation-for-Computer-Vision-AI-ML