Visual object detection in domestic settings

Diwas Chaulagain
MS Data Science

University of New Haven

dchau5@unh.newhaven.edu

Geethika Somisetty
MS Data Science

University of New Haven

gsomi1@unh.newhaven.edu

Satya Sai Sri Nikhil Namduri
MS Data Science
University of New Haven
snamu1@unh.newhaven.edu

Giri Merugu
MS Data Science
University of New Haven
gmeru2@unh.newhaven.edu

Naga Venkata Bharath Lanka

MS Data Science

nlank3@unh.newhaven.edu

***Abstract*— The paper is about visual object detection in domestic settings. This paper briefly discusses what visual object detection is and the different models that are best suited for visual object detection in domestic settings. The comparison is done through the processing of different models, the probability comparison between the models is also done and the results are dissected. We also discussed the models used previously in similar comparisons. The five models used here are MobileNet SSD, YOLOV5, YOLOV8, YOLOV11 and Detectron2.**

I. INTRODUCTION

Machine learning and artificial intelligence are rapidly transforming numerous industries and addressing a variety of real-world problems. Among the many applications of machine learning, visual object detection is an essential area of computer vision and stands out for its ability to enable machines to identify and interpret objects and environments in images and videos. Visual object detection is foundational for applications such as autonomous vehicles, medical imaging, animal tracking and conservation, drone technology, and robotics. There have been many advancements in visual object detection as new models are being developed rapidly. In this experiment we primarily focus on five models. They are Meta's detectron2, YOLO's V5, V8, V11 and ImageNet. We will be looking into the predictions and see the probability of it finding certain things in an indoor environment. We will be comparing the probabilities for similar sets of images comparing two of the most accurate models, YOLO and Detectron2.

In this project, we focus on advancing visual object detection within domestic settings, exploring two objectives. First, we assess and compare popular libraries and tools to determine which are best suited for indoor environments by evaluating model confidence and accuracy. Next, we aim to estimate the probability of finding specific objects within certain indoor settings (e.g., the likelihood of a toothbrush being in a bathroom). We have divided the task amongst ourselves to explore these models and will come to a collective conclusion.

The main objective of using different models is to understand how the models perceive and predict objects in domestic settings. Through the experiment and by looking at the working models, we get to know which model is well suited for a certain scenario compared to other models. Using different models also allows us to capture different things in the same picture as the working mechanism of a certain model may be helpful in predicting certain object in a picture which

As of this report, we have successfully completed the first objective, comparing libraries and tools for model performance in domestic settings. We are currently progressing on the second objective, focused on probability estimation for object location within various indoor environments. Our team has been able to calculate the probabilities of one indoor setting and in the coming days will complete as much probability estimation as possible. Initially, we had decided to go with 'Probability estimation of an object in indoor environments' but we later realized that even if we do not see an object in the frame, it does not mean that object is not in the room. So, the better way would be to name it

"Probability estimation of an object facing a certain way in domestic or office settings".

II. Similar Works

There have been many comparisons between different object detection models throughout the years. Our team was not able to find such a comparison in an indoor-only setting but in general. We can find such comparison in 'Comparative analysis of deep learning image detection algorithms' that was published in 2021. In this paper, we can see the comparison between Faster-RCNN, SSD and YOLOV3. In this comparison YOLOv3 clearly outperforms the other two models. Keeping in mind this experiment was done just 3 years ago in 2021; much has evolved in the field of object detection. The lowest performing YOLO model of this comparison set (YOLOV5) outperforms the best performing model of that comparison. Although this paper also concludes it is better to use faster RCNN for smaller datasets that require faster processing.

The experiment done in 'A review: Comparison of performance metrics of pretrained models for object detection using the TensorFlow framework' differs in conclusion from the previous mentioned paper. In this paper, the conclusion is that YOLO and SSD are faster, but fast RCNN is more accurate. The two papers are not consistent, which clearly shows that the experts in this field are also trying to figure out properly the results and which one is better. This seems to indicate deeper research is required in the field.

The closest comparison can be found in the 'Comparison of YOLOv8 and Detectron2 on Crowd Counting techniques' where Detectron2 was found to be working better in outdoor scenario, specifically to count the number of people at a certain frame in public. This was a good enough resemblance but we are dealing with indoor scenarios so we could not depend on this experiment alone.

III. Methodology

A) YoloV5: YoloV5 is a real-time object detection framework developed by Ultralytics.

❖ **Architecture:**

YoloV5 architecture has three main parts:
Backbone, Neck, and Head

The backbone is like the "brain" of the model—it extracts key features from the input image.

- ❖ **Focus Layer**: Reduces the image size and picks out key details to make processing faster.
- ❖ **CSPDarknet53**: A network that processes the image features efficiently while keeping the model lightweight.
- ❖ **Residual Connections**: Helps the model learn better and avoid problems like "forgetting" earlier features.

The neck improves the features extracted by the backbone, making it easier to detect objects of varied sizes.

- ❖ **Feature Pyramid Network (FPN)**: Combines features from different layers to detect both small and large objects.
- ❖ **Path Aggregation Network (PANet)**: Helps refine the features further, improving the accuracy of object localization. The head is where the final predictions happen. It decides where the objects are, what they are, and how confident it is about the predictions.
- ❖ **Bounding Boxes**: Draws rectangles around objects.
- ❖ **Class Prediction**: Identifies what type of object is in the box (e.g., car, person, etc.).
- ❖ **Confidence Score**: Tells how sure the model is about each detection.

YoloV5 takes the input image and divides it into a grid. Each grid cell predicts objects in its area, including their location, size, and type. The model combines predictions and outputs the final results.

Training Process in YOLOV5:

Data Preparation: Organize images and labels in COCO format and specify dataset details in a file.

Data Augmentation: Apply techniques like flipping, scaling, cropping, and mosaic to improve model generalization.

Loss Computation: Calculate localization, abjectness, and classification losses to optimize predictions during training.

Prediction:

The input image is passed through the YOLOv5 architecture, where it goes through a backbone (like CSPDarknet53) to generate feature maps, extracting important visual information.

YOLOv5 directly predicts bounding boxes, object classes, and confidence scores for each grid cell, without using Region Proposal Networks (RPN) like in some other models. It uses anchor boxes to improve localization accuracy.

The model filters out low-confidence predictions, refines the bounding boxes, and displays the final results with class labels and confidence scores. The output is usually displayed as bounding boxes on the detected objects.

B) YoloV8:

❖ **Architecture:**

YOLOv8 consists of three major components: Backbone, Neck, and Head. Each has a special role in the processing of the input image for correct detection.

1. Backbone:

The backbone extracts important features from the input image and acts as the base of the detection pipeline.

C2f Module (ConvNext-inspired): The "C2f" module is integrated into the backbone for efficient feature extraction with fewer parameters and without sacrificing much accuracy. It improves the feature representation, especially for small objects.

Dynamic Convolutions: These change according to the input for better efficiency in images of different resolutions.

Residual Connections: Residual connections help avoid problems such as vanishing gradients, ensuring early-stage features are preserved during training.

2. Neck:

The features extracted by the backbone are processed in the neck for the refinement of final prediction, paying more attention to improving object location and multi-scale detection. Path Aggregation Network - PANet: Like YOLOv5, PANet is used in YOLOv8 for better flow of information between layers, helping in the capture of both fine details-small objects-and their broader context-large objects.

BiFPN: The final improvement is the network, which merges features across scales and amplies features to improve accuracy for objects across multiple sizes.

3. Head:

The head generates the object detection results, that is, object locations, categories, and confidence levels.

Anchor-free Detection: Unlike previous YOLO versions, the architecture of YOLOv8 is anchor-free and simplifies the process with fewer computational resources being consumed in prediction.

Object Classification and Localization: YOLOv8 directly predicts the class, bounding box, and confidence score for each object in the image grid.

❖ **Training**

1. **Data Preparation:**

Dataset is organised in a compatible format (e.g., COCO or YOLO format). Labelled images with bounding boxes and class information. The dataset configuration file specifies training, validation, and test data paths.

2. **Data Augmentation:**

To enhance generalization, YOLOv8 applies robust augmentation techniques, including:

MixUp: Combines multiple images for improved robustness.

Mosaic Augmentation: Merges four images into one, helping the model learn diverse spatial arrangements.

HSV Augmentation: Changes hue, saturation, and brightness for better adaptability to lighting conditions.

3. **Loss Computation:**

YOLOv8, during training, optimizes predictions by minimizing multiple losses:

Localization Loss: It makes sure the bounding boxes fit well with the object boundaries.

Confidence Loss: It measures the accuracy of the predictions about object presence.

Classification Loss: It evaluates the correctness of the class predictions.

Prediction:

When an image is passed through YOLOv8:

The backbone extracts visual features and produces feature maps.

The neck refines these feature maps across multiple scales.

The head generates predictions directly without relying on anchor boxes. It outputs bounding boxes, object classes, and confidence scores.

Post-processing filters out low-confidence detections and applies Non-Maximum Suppression (NMS) to remove overlapping boxes.

The final output displays bounding boxes over detected objects, along with class labels and confidence scores.

Challenges with YOLOv8:

1. **Complex Environments:** Detecting objects in highly cluttered or occluded scenarios remains challenging.
2. **Lighting Variability:** Extreme changes in lighting, such as bright reflections or low-light conditions, can impact detection accuracy.
3. **High Computation Demand:** While efficient, running YOLOv8 on edge devices with limited processing power may require careful optimization.
4. **Ethical Concerns:** Using YOLOv8 for surveillance applications in sensitive areas can raise privacy and ethical issues.

C) YoloV11:

❖ **Abstract**

This paper presents YOLOv11, an enhanced iteration of the YOLO (You Only Look Once) object detection framework. We introduce significant architectural improvements and innovative features that advance the state-of-the-art in real-time object detection. The proposed model integrates transformer-based components with traditional convolutional architectures, alongside novel training methodologies and optimization strategies. Our comprehensive evaluation demonstrates superior performance in terms of accuracy, speed, and resource efficiency across diverse deployment scenarios.

❖ Introduction

Real-time object detection remains a critical challenge in computer vision, demanding continuous improvements in both accuracy and computational efficiency. Building upon the success of previous YOLO iterations, YOLOv11 introduces fundamental architectural changes and innovative features designed to address contemporary challenges in object detection while maintaining real-time performance capabilities.

❖ Architecture

▫ Enhanced Backbone Design

YOLOv11 implements an advanced backbone architecture featuring an enhanced CSPDarknet with integrated transformer components. The backbone incorporates dynamic feature aggregation modules and adaptive attention mechanisms, enabling superior feature extraction across multiple resolutions. This design significantly improves the model's ability to capture complex spatial relationships while maintaining computational efficiency.

▫ Smart Neck Structure

The neck architecture employs a bi-directional feature pyramid network with sophisticated cross-scale connections. Key innovations include:

- Adaptive feature fusion mechanisms that dynamically adjust information flow

- Dynamic channel attention modules for optimal feature selection

- Smart path aggregation utilizing learnable weights for optimal information routing

▫ Intelligent Head Design

The detection head incorporates multi-task learning capabilities with dynamic anchor assignment strategies. Notable features include:

- Instance-aware detection heads for improved object discrimination

- Adaptive IoU threshold selection mechanisms

- Dynamic anchor assignment based on object characteristics

❖ Key Innovations

▫ Hybrid Detection System

YOLOv11 introduces a novel hybrid detection approach that seamlessly combines anchor-free and anchor-based methodologies. This system dynamically switches between detection strategies based on scene complexity and object characteristics, particularly enhancing small object detection performance.

▫ Advanced Loss Functions

The model implements a sophisticated loss computation system featuring:

- Dynamic weight balancing for optimal gradient flow

- Scale-aware loss adjustment mechanisms

- Adaptive label assignment strategies

❖ Implementation and Optimization

▫ Training Enhancements

The training pipeline incorporates several advanced features:

- Smart data processing with enhanced mosaic augmentation
- Adaptive image scaling techniques
- Curriculum learning integration for improved convergence
- Dynamic batch normalization strategies

▫ Performance Optimizations

YOLOv11 achieves significant performance improvements through:

- Hardware-specific acceleration techniques
- Memory-efficient operations
- Streamlined inference pipeline
- Parallel processing capabilities

❖ Deployment Considerations

▫ Resource Efficiency

The model demonstrates superior resource efficiency through:

- Reduced memory footprint
- Optimized computation requirements
- Efficient GPU utilization
- Effective model pruning strategies

▫ Deployment Flexibility

YOLOv11 supports diverse deployment scenarios including:

- Edge device optimization
- Cloud deployment configurations
- Mobile-friendly variants
- Cross-platform compatibility

❖ Conclusion

YOLOv11 represents a significant advancement in real-time object detection, offering substantial improvements in detection accuracy, processing efficiency, and deployment flexibility. The model's innovative architecture and optimization strategies make it particularly suitable for complex real-world applications including real-time object detection, complex scene understanding, and multi-object tracking.

D) Detectron: Detectron works like this.

❖ Architecture:

Detectron2's main working mechanism is through a CNN like ResNet, ResNeXt and FPN (Depending upon the situation).

Detectron uses Region Proposal Network (RPN) to narrow down the areas of interest for better analysis.

The output from the RPN is provided to the Region of Interest (ROI) for further classification and refinement. ROI has mask for segmentation as well.

It also has task specific heads to do different tasks such as box regression for object localization, mask prediction and keypoint prediction to estimate the pose of the object (specially humans).

❖ Training

Detectron prepares data in a COCO format (Images, Annotations and Metadata) for easy integration.

Data augmentation is done through random flipping, scaling and cropping.

Loss function and cross-entropy loss are calculated during the training process.

❖ Prediction

The images pass through the architecture that generates feature maps.

The RPN passes the images to ROI which selects and narrows down the bounding boxes and predicts the output.

The main CNN architecture is selected according to the required tasks.

E) MobileNetSSD: MobileNet SSD is combination of two powerful techniques used in computer vision MobileNet and SSD (Single Shot MultiBox Detector).

Architecture:

It integrates the MobileNet as feature extractor with SSD layers appended for object detection.

MobileNet extracts features from input image and then the feature maps with different scales are used for detecting objects and then SSD layers are added to predict the bounding boxes by using convolutional layers.

❖ Training

In training the model takes the input image and resize them to fixed size and use the dataset which is annotated bounding boxes.

As it is having two components mobile net extracts features from the image and SSD will give the class probablities for each anchor box.

Measures the difference between predicted and ground-truth box coordinates.

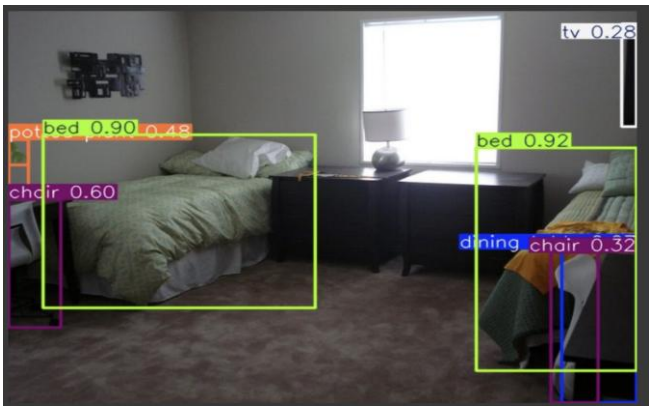
Categorical Cross-Entropy measures the classification error for object labels.

❖ Prediction

During Inference the trained MobileNet SSD predicts in unseen images.

Resize and normalize the input image as done during training.

Pass the image through the trained MobileNet backbone to extract feature maps.



Bedroom

IV. Results

A) YoloV5:

Now, let's review the outcomes generated from the YOLOV5 implementation.

```
import os
HOME = os.getcwd()
print (HOME)
%pip install ultralytics supervision roboflow
#installing YOLO11 VIA ULTRALYTICS
!git clone
import ultralytics
ultralytics.checks()
from google.colab import drive
drive.mount('/content/drive')
#--- You can simply run all tasks from the terminal
with the yolo command---
!yolo task=detect mode=predict model=yolov5xu.pt
conf=0.25
source='/content/drive/MyDrive/Bedroom/8_cuarto.jpg'
save=True
```

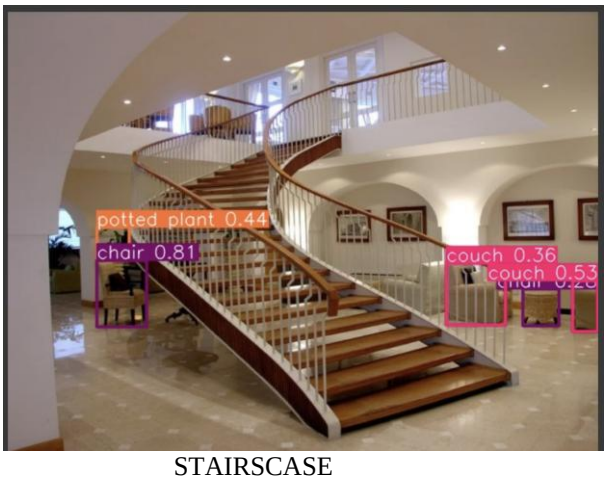
Above is the code used to perform object detection with YOLOv5. First, the necessary libraries are installed and imported. Then, the YOLOv5 model processes images from the input folder, detects objects, and saves the results, including bounding boxes and labels, in the output folder.

This allows for efficient and real-time object detection in various scenarios.

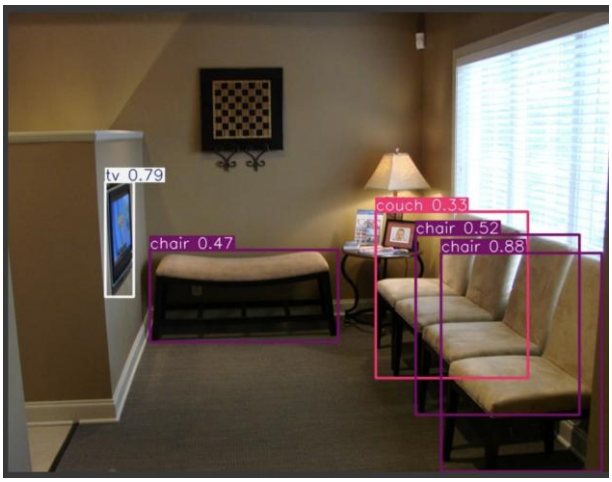
```
# ---Result annotated image got saved in
`{HOME}/runs/detect/predict/`. Let's display
it---
from IPython.display import Image as IPyImage

IPyImage(filename=f'/content/runs/detect/predict2/8_cua
rto.jpg', width=600)
```

The output has been generated. Let's take a look at a few examples from the predictions.

Dining Room



STAIRSCASE



Waiting room

From the sample images, YOLOV5 has successfully detected objects to a good extent, but there is still room for improvement to enhance its performance on other objects. Here are some challenges:

Yolov5 challenges:

- **Visual Clutter**: Overlapping objects, or crowded environments make it harder to distinguish items accurately.
- **Lighting Variability**: Changes in lighting, such as shadows or dim conditions, can reduce detection performance.

B) **YoloV8:** Results from the Yolo8 are below

Code:

```
[ ]  import os
     HOME = os.getcwd()
     print(HOME)
```

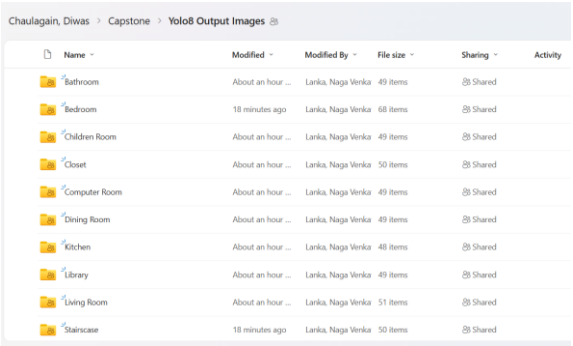
```
%pip install ultralytics supervision roboflow
import ultralytics
ultralytics.checks()
```

```
!yolo task=detect mode=predict model=yolov8x.pt conf=0.25 source='/content/Bathroom/008.jpg'
```

```
from IPython.display import Image as IPyImage

IPyImage(filename=f'{HOME}/runs/detect/predict/008.jpg', wid
```

By using this code, objects can be detected in images, and the results, including bounding boxes, class labels, and confidence scores, are obtained.

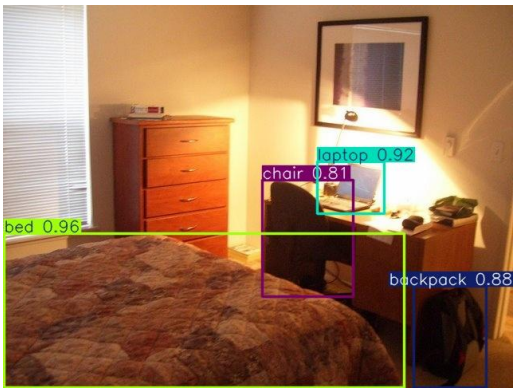
Output Images for Yolo8 are uploaded to the sharepoint.



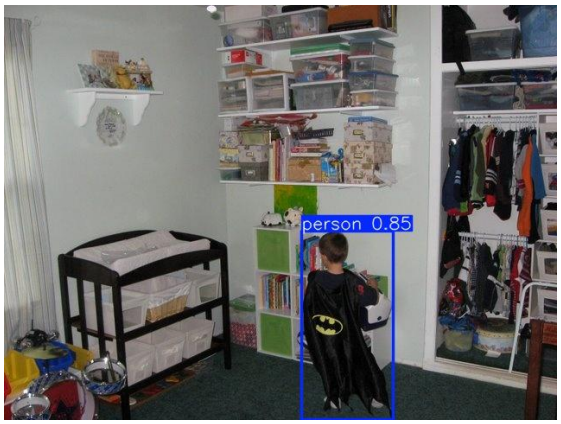
Output Images:

Bathroom:

Bedroom:



Children Room:



Dining Room:

C) YoloV11: Let us now look at the results we obtained from the YoloV11 and codes.

```
import os
import glob
import cv2
from ultralytics import YOLO
from openpyxl import Workbook, load_workbook
import pandas as pd
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')

# Function to create output directory if it doesn't exist
def create_output_dir(base_path, room_name):
    output_dir = os.path.join(base_path, 'detection_results', room_name)
    os.makedirs(output_dir, exist_ok=True)
    return output_dir

# Function to filter predictions by specific classes
def filter_predictions_by_class(df, object_classes):
    return df[df['Class'].isin(object_classes)]

# Function to save predictions to Excel for a single room
def save_predictions_to_excel(output_excel_path, prediction_data, object_classes):
    wb = Workbook()
    ws = wb.active
    ws.title = "Predictions"

    # Add headers
    headers = ["Picture", "Probability in percentage (According to the highest confide
    headers.extend(object_classes)
    ws.append(headers)

    # Add prediction data
    for prediction in prediction_data:
        row = [prediction["Picture"], prediction["Highest Confidence"]]
        for obj_class in object_classes:
            row.append(prediction["Confidence Scores"].get(obj_class, 0))
        ws.append(row)

    wb.save(output_excel_path)
```

By using this code, objects can be detected in images, and the results, including bounding boxes, class labels, and confidence scores, are obtained.

```
# Add math summary sheet
summary_ws = wb.create_sheet(title="Summary")
summary_headers = ["Room", "Facing Door", "Not Facing Door"] + object_classes
summary_ws.append(summary_headers)

for room in specific_rooms:
    room_excel_path = os.path.join(detection_results_path, room, f"{room}_predictions.xlsx")
    if os.path.exists(room_excel_path):
        df = pd.read_excel(room_excel_path)
        facing_door = df[df['Class'] == 'door']['Confidence'].sum()
        not_facing_door = df[df['Class'] != 'door']['Confidence'].sum()

        object_totals = {obj_class: df[df['Class'] == obj_class]['Confidence'].sum() for obj_class in object_classes}

        summary_row = [room, facing_door, not_facing_door] + [object_totals[obj_class] for obj_class in object_classes]
        summary_ws.append(summary_row)
```

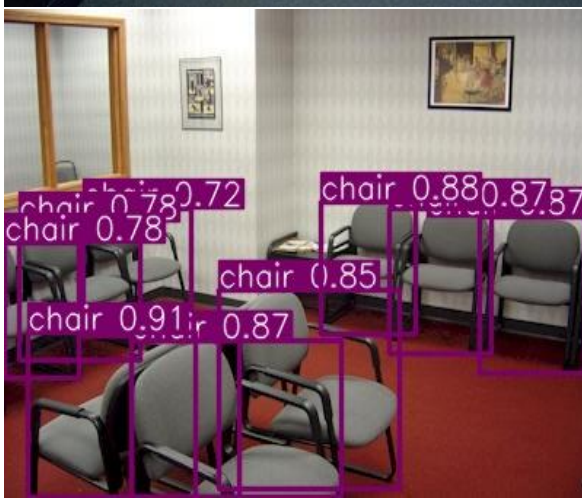
And the following code demonstrates how to run V11 inference on a dataset which has been saved in the drive folder.

The predictions and probabilities are saved in respective folders of the drive if we use this code.

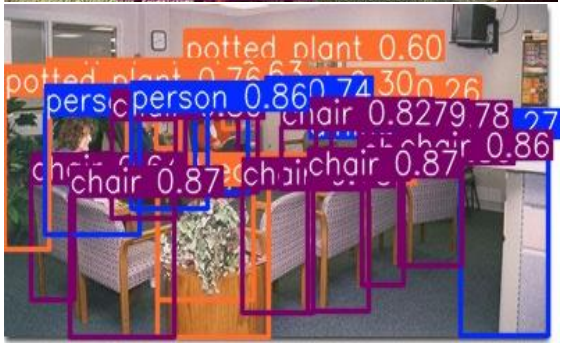
```
    for idx, img_path in enumerate(image_files):
        img = cv2.imread(img_path)

        # Run detection
        results = model(img)
        result = results[0]

        # Extract confidence values and predictions
        confidence_scores = {obj: 0 for obj in object_classes}
        highest_conf = 0
        for box in result.boxes:
            class_id = int(box.cls)
            conf = float(box.conf) * 100  # Convert to percentage
            class_name = result.names[class_id]
            confidence_scores[class_name] = max(confidence_scores[class_name], conf)
            highest_conf = max(highest_conf, conf)

        # Save annotated image
        annotated_img = result.plot()
        output_filename = os.path.join(output_dir, f"detected_{os.path.basename(img_path)}")
        cv2.imwrite(output_filename, annotated_img)

        # Store prediction for Excel
        prediction_data.append({
            "Picture": idx + 1,
            "Highest Confidence": round(highest_conf),
            "Confidence Scores": confidence_scores
        })

    # Save room-specific predictions to Excel
    room_excel_path = os.path.join(output_dir, f"{room}_predictions.xlsx")
    save_predictions_to_excel(room_excel_path, prediction_data, object_classes)

# Consolidate all Excel files into one
consolidate_excel_files(output_base_path, consolidated_excel_path, object_classes, specific_rooms)

# Set paths
INPUT_PATH = '/content/drive/MyDrive/Images/Images'
OUTPUT_PATH = '/content/drive/MyDrive/yolo11_Predictions1'
CONSOLIDATED_EXCEL_PATH = '/content/drive/MyDrive/YOLO_Predictions_Consolidated.xlsx'
SPECIFIC_ROOMS = ['Children Room', 'Closet', 'Computer Room','Dining Room Room', 'Bedroom', 'Bathroom']  # Replace with the rooms mentioned in the reference Excel

# Run processing
process_images_and_save(INPUT_PATH, OUTPUT_PATH, CONSOLIDATED_EXCEL_PATH, SPECIFIC_ROOMS)
```

Output Images: We have tested YoloV11 on the same images that we have tested by using the other models to obtain the difference between the models.

Waiting Room:



LivingRoom:



1) How does YOLO11 achieve greater accuracy with fewer parameters?

YOLO11 achieves greater accuracy with fewer parameters through advancements in model design and optimization techniques. The improved architecture allows for efficient feature extraction and processing, resulting in higher mean Average Precision (mAP) on datasets like COCO while using 22% fewer parameters than YOLOv8m. This makes YOLO11 computationally efficient without compromising on accuracy, making it suitable for deployment on resource-constrained devices.

a) What are the key improvements in Ultralytics YOLO11 compared to previous versions?

Ultralytics YOLO11 introduces several significant advancements over its predecessors. Key improvements include:

- **Enhanced Feature Extraction:** YOLO11 employs an improved backbone and neck architecture, enhancing feature extraction capabilities for more precise object detection.
- **Optimized Efficiency and Speed:** Refined architectural designs and optimized training pipelines deliver faster processing speeds while maintaining a balance between accuracy and performance.
- **Greater Accuracy with Fewer Parameters:** YOLO11m achieves higher mean Average Precision (mAP) on the COCO dataset with 22% fewer parameters than YOLOv8m, making it computationally efficient without compromising accuracy.
- **Adaptability Across Environments:** YOLO11 can be deployed across various environments, including edge devices, cloud platforms, and systems supporting NVIDIA GPUs.
- **Broad Range of Supported Tasks:** YOLO11 supports diverse computer vision tasks such as object detection, instance segmentation, image classification, pose estimation, and oriented object detection (OBB).

*B. **Technical Limitations:***

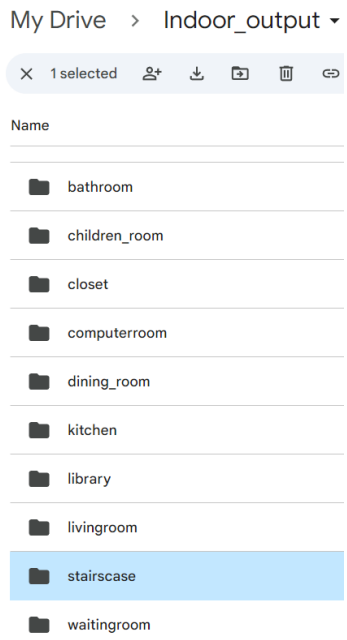
- Still struggles with extremely small objects
- Performance degradation in highly crowded scenes
- Difficulty with heavily overlapped objects
- Limited accuracy in extreme lighting conditions

D) Detectron: Let us now look at the results we obtained from the detectron codes.

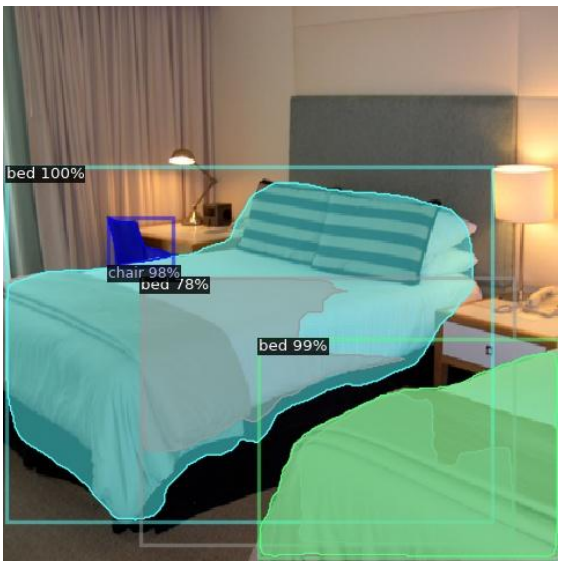
```
from google.colab import drive
drive.mount('/content/drive')

!pip install labelimg
```

```
import os
import shutil
from pathlib import Path
import zipfile
from google.colab import files

# Or, to install it from a local clone:
!git clone
https://github.com/facebookresearch/detectron2.git
!python -m pip install -e detectron2
import os
import cv2
from detectron2.config import get_cfg
from detectron2.engine import DefaultPredictor
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog
from detectron2 import model_zoo
from detectron2.utils.logger import setup_logger
from google.colab.patches import cv2_imshow

setup_logger()

# --- Paths and settings ---
image_folder = "/content/drive/MyDrive/Input images for
probability"  #input path
output_folder = "/content/drive/MyDrive/Indoor_output"
#
os.makedirs(output_folder, exist_ok=True)

# --- Loop through images in all subfolders and save
predictions ---
for root, _, filenames in os.walk(image_folder):
for filename in filenames:
if filename.endswith((".jpg", ".jpeg",
".png")):
image_path = os.path.join(root, filename)

try:
im = cv2.imread(image_path)
if im is None:
print(f"Failed to load image:
{image_path}")
continue


outputs = predictor(im)


v = Visualizer(im[:, :, ::-1],
MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
v =
v.draw_instance_predictions(outputs["instances"].to("cp
u"))


rel_path = os.path.relpath(image_path,
image_folder)
output_subfolder =
os.path.join(output_folder, os.path.dirname(rel_path))
os.makedirs(output_subfolder,
exist_ok=True)

output_path =
os.path.join(output_folder, rel_path)
cv2.imwrite(output_path,
v.get_image()[:, :, ::-1])

except Exception as e:
print(f"Error processing image
{image_path}: {e}")
```

Above is the code used to generate the images. First we install and import the necessary libraries. Then, we clone and install Detectron2. Then, we define the input and output folders. The model then takes the input folder, detects the images and save it in the output folder.



The output is created. Let us look at some of the pictures from the predictions.



Bedroom

Bathroom

Dining room

As seen in three sample images, the images have been predicted to some degree of success and there are still some areas left that need to be improved to make the model predict other things as well. There are some major drawbacks and good points of this model:

Drawbacks:

1) There are certain factors of bias like all types of tables are being classified as dining tables and this model also has significant bias towards predicting many items as humans.
2) If there is certain object on top an object, the model tends to predict as the larger object.
3) Draws image boundaries well but sometimes predicts wrong things with high confidence.

Positives:

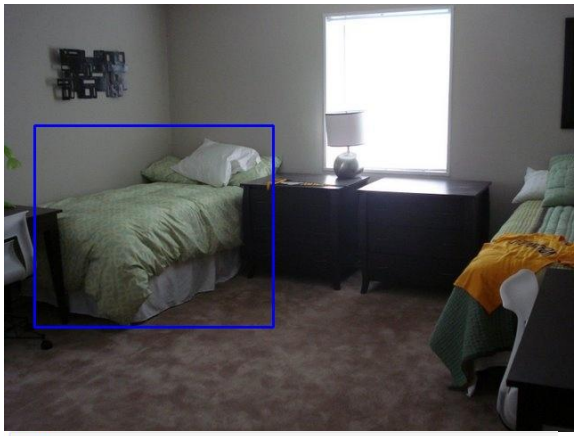
1) Except for a few above-mentioned cases, it predicts objects with high confidence and speed.
2) Despite some biased cases, it predicts the elements of indoor settings perfectly with high speed as it took us less than a minute to process and store around 3000 images.
3) It draws the flexible bounding boxes perfectly.

E) MobileNet SSD

The results of Mobile Net SSD - Dining Room


Bathroom

Bed Room



```
import cv2

# Load pre-trained MobileNet-SSD model and class labels
net = cv2.dnn.readNetFromCaffe('deploy.prototxt', 'mobilenet_iter_73000.caffemodel')

# Load an image
image_path = "C:\\Users\\satis\\.cache\\kagglehub\\datasets\\itsahmad\\indoor-scenes
img = cv2.imread(image_path)
h, w = img.shape[:2]

# Preprocess the image and run detection
blob = cv2.dnn.blobFromImage(img, 0.007843, (300, 300), 127.5)
net.setInput(blob)
detections = net.forward()

# Visualize results
for i in range(detections.shape[2]):
    confidence = detections[0, 0, i, 2]
    if confidence > 0.5:  # Set confidence threshold
        box = detections[0, 0, i, 3:7] * [w, h, w, h]
        (x1, y1, x2, y2) = box.astype("int")
        cv2.rectangle(img, (x1, y1), (x2, y2), (255, 0, 0), 2)

cv2.imshow("Detection", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

As you can see in the sample images mobile net ssd takes the input images and and detect the objects images which are pretrained by that model are highlighted by bounding boxes.

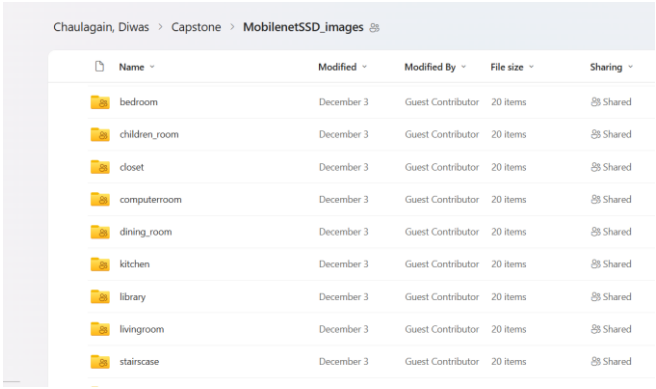
As you can see the code detecting the objects inside living room, dining room and bathroom is also loading the model from pretained opencv library.

Advantages:

1. It is lightweight with low memory and computational requirements.
2. It can detect multiple objects with multiple feature map scales.
3. It can process images most quickly.

Drawbacks:

1. The model accuracy is low compared to other models like YOLO.
2. Not Suitable for medical images and high precision tasks.



Above is the place where I have uploaded the all the images detected by Mobilenet SSD model to detect the objects.

V. PROBABILITY AND CONDITIONAL PROBABILITY

The next thing we did was compare the probability of finding certain objects in different rooms. For the comparison, we looked at two of the top performing models YOLOV11, Detectron2 and YOLOV8 and created a probability table. In the table, we selected 20 images from each of some rooms that are common in domestic settings and tried to

check the probability of finding certain objects. It may look trivial at first but there are a lot of positives of doing this probability check.

Some of the use cases of probability estimation are:

- ❖ Comparative analysis: We can compare the two models and check which objects are detected better by which model.
- ❖ Reliability: The probability estimation also provides us with insights into how reliable the models are. Larger projects cannot depend upon unreliable models.
- ❖ Performance metrics: The tables can be used as a form of performance metrics to check how well the models are performing in detecting the object.

Let us look at the predictions for three rooms by the models

A) Bathroom

Objects	Probability
Toilet	51.85
Sink	54.95
Cup	5.75
TV	2.6
Bottle	11.6
Vase	10.1
Dog	0
Potted Plan	31.1
Bowl	6.25
Donut	0
Cell Phone	0
Person	0
Tooth Brush	4
Chair	0
Microwave	0
Handbag	0

Yolo8 Prediction Probability

Objects	Probability
Person	15.9
TV	20.55
Book	21.25
Chair	62.65
Dining Table	17.75
Tie	0
Bird	0
Handbag	0
Bowl	7.65
Bottle	10.1
Refrigerator	2.6
Cake	0
Teddy Bear	27.25
Dog	2.45
Toothbrush	0
Remote	0
Knife	0
Sports Ball	2.45
Clock	2.4
Couch	15.75
Back Pack	0
Potted Plant	2.6
Bed	0
Umbrella	2.8
Kite	0
Laptop	3.25
Cell Phone	0
Cup	1.45

Yolo8 Prediction

Objects	Probability
Bottle	16.81812525
Cup	5.611008704
Potted Plant	21.60768539
Sink	51.65118486
Toilet	47.22441599
TV	1.417797506
Vase	10.72593048

Yolo 11 Prediction Probability

Objects	Probability
Bird	1.889152527
Book	15.73483974
Bottle	2.09242627
Chair	55.40657997
Clock	2.624777853
Couch	15.84428459
Dining Table	20.96082687
Dog	1.604548097
Microwave	3.092425168
Person	15.07993966
Refrigerator	3.899460435
Surfboard	3.974413276
Teddy Bear	27.1050626
TV	16.32489085
Umbrella	5.012238473
Vase	2.35958457

YOLO 11 prediction

Objects	Probability
Toilet	55.6
Sink	76.25
Cup	10.2
TV	3.75
Bottle	32.4
Vase	14.15
Dog	0
Potted Plan	31.15
Bowl	8.4
Donut	2.95
Cell Phone	0
Person	3.5

Detectron Prediction Probability

B) Children Room

Objects	Probability
Person	22.45
TV	19.8
Book	45.25
Chair	74
Dining Table	30.55
Tie	3.725
Bird	4.45
Handbag	21.45
Bowl	11.7
Bottle	22.8
Refrigerator	5.45
Cake	4.2
Teddy Bear	30.25
Dog	9.3
Toothbrush	3.4
Remote	3.1
Knife	4.15
Sports Ball	3.85
Clock	7.75
Couch	7
Back Pack	3.1
Potted Plant	10.5
Bed	4.85
Umbrella	3.9
Kite	0
Laptop	0
Cell Phone	0
Cup	8.3

Detectron prediction

C) Bedroom

Objects	Probability
Bed	79.8
Chair	42.9
Book	8.6
TV	2.25
Cat	0
Vase	15.1
Dog	0
Potted Pla	21.4
Laptop	8
Couch	14.3
Table	0
Person	0
Bench	0
Dining Tab	0
Teddy Bea	0
Bowl	2.15
Bottle	0
Cup	0

Objects	Probability
Bed	69.25985947
Book	8.521737903
Bowl	1.607049406
Chair	45.43879092
Couch	14.61760223
Dining Table	2.582083046
Laptop	7.822623253
Potted Plant	20.86274475
Teddy Bear	1.93949759
Vase	13.39745343

YOLO 11 prediction

Objects	Probability
Bed	82.3
Chair	59.3
Book	7.9
TV	16.05
Cat	3.05
Vase	29.6
Dog	0
Potted Pla	34.3
Laptop	9.55
Couch	13.65
Table	0
Person	3.7
Bench	2.95
Dining Tab	8.5
Teddy Bea	0
Bowl	3.55
Bottle	3.75
Cup	0

Detectron probability

From all three sets of probability tables, we can safely claim that Detectron2 captures more items in a picture, has clearer bounding boxes and has higher confidence in predictions. YOLOv8 and YOLO v11 are good object detection models and can be used for various purposes, but comparing indoor and office scenarios, detectron does a better job as it individually predicts more items with higher confidence.

Partial probability:

Partial probability for three of the rooms using detectron is as below:

Facing Door	Not Facing the Door
35%	65%

Objects	Probability when facing the door	Probability when not facing t
Bed	27.93	
Chair	15.015	
Book	3.01	
TV	0.7875	
Cat	0	
Vase	5.285	
Dog	0	
Potted Plant	7.49	
Laptop	2.8	
Couch	5.005	
Table	0	
Person	0	
Bench	0	
Dining Table	0	
Teddy Bear	0	
Bowl	0.7525	
Bottle	0	
Cup	0	

Bedroom Using Yolo8

Facing Door	Not Facing the Door
10%	90%

Objects	Probability when fa	Probability when not facing t
Toilet	5.185	
Sink	5.495	
Cup	0.575	
TV	0.26	
Bottle	1.16	
Vase	1.01	
Dog	0	
Potted Plant	3.11	
Bowl	0.625	
Donut	0	
Cell Phone	0	
Person	0	
Tooth Brush	0.4	
Chair	0	
Microwave	0	
Handbag	0	

Bathroom Using Yolo8

Facing Door	Not Facing the Door
20%	80%

Objects	Probability when facing the door	Probability when not facing t
Person	3.18	
TV	4.11	
Book	4.25	
Chair	12.53	
Dining Table	3.55	
Tie	0	
Bird	0	
Handbag	0	
Bowl	1.53	
Bottle	2.02	
Refrigerator	0.52	
Cake	0	
Teddy Bear	5.45	
Dog	0.49	
Toothbrush	0	
Remote	0	
Knife	0	
Sports Ball	0.49	
Clock	0.48	
Couch	3.15	
Back Pack	0	
Potted Plant	0.52	
Bed	0	
Umbrella	0.56	
Kite	0	
Laptop	0.65	
Cell Phone	0	
Cup	0.29	

Children Room Using Yolo8

Objects	Probability when facing the door	Probability when not facing the door
Bed	28.805	53.495
Chair	20.755	38.545
Book	2.765	5.135
TV	5.6175	10.4325
Cat	1.0675	1.9825
Vase	10.36	19.24
Dog	0	0
Potted Plant	12.005	22.295
Laptop	3.3425	6.2075
Couch	4.7775	8.8725
Table	0	0
Person	1.295	2.405
Bench	1.0325	1.9175
Dining Table	2.975	5.525
Teddy Bear	0	0
Bowl	1.2425	2.3075
Bottle	1.3125	2.4375
Cup	0	0

Bedroom

Objects	Probability when facing the door	Probability when not facing the door
Person	4.49	17.96
TV	3.96	15.84
Book	9.05	36.2
Chair	14.8	59.2
Dining Table	6.11	24.44
Tie	0.745	2.98
Bird	0.89	3.56
Handbag	4.29	17.16
Bowl	2.34	9.36
Bottle	4.56	18.24
Refrigerator	1.09	4.36
Cake	0.84	3.36
Teddy Bear	6.05	24.2
Dog	1.86	7.44
Toothbrush	0.68	2.72
Remote	0.62	2.48
Knife	0.83	3.32
Sports Ball	0.77	3.08
Clock	1.55	6.2
Couch	1.4	5.6
Back Pack	0.62	2.48
Potted Plant	2.1	8.4
Bed	0.97	3.88
Umbrella	0.78	3.12
Kite	0	0
Laptop	0	0
Cell Phone	0	0
Cup	1.66	6.64

Children Room

Objects	Probability when facing the door	Probability when not facing the door
Toilet	5.56	50.04
Sink	7.625	68.625
Cup	1.02	9.18
TV	0.375	3.375
Bottle	3.24	29.16
Vase	1.415	12.735
Dog	0	0
Potted Plant	3.115	28.035
Bowl	0.84	7.56
Donut	0.295	2.655
Cell Phone	0	0
Person	0.35	3.15

Bathroom

Partial probability for this case has been calculated to check which angle is more effective as the probability of finding a certain object at a certain angle is higher than other angles. Here, we have taken the direction of the door as our focal point as all rooms have doors. We have compared the probability of objects in the picture when facing the door vs when not facing the door and looked at the results. This gives the model an idea of what to expect when facing a certain way.

VI. Conclusion

The entire project is focused on selecting the best model for detecting objects in indoor settings. We have further divided the entire project into two parts. One is comparing the

results of all five models. The least impressive of the five, MobileNetSSD does not even give the confidence score or names the object inside the bounding box. The YOLO versions gradually improve in performance and YOLOV11 gives a really good output with impressive confidence score. The best one, however, is Detectron2 from Meta. Despite the many positives, detectron 2 also has many areas to improve on.

The probability estimations and conditional probabilities generated indicate the accuracy, capability and the overall reliability of the models. We concluded with the statement that Detectron2 is best suited among the models in indoor scenarios, but it also has its own downsides. The model is a bit biased towards few objects and struggles to draw boundaries in low quality images. Further improvements and fine tuning in this model can lead to a perfect indoor detection model. One good improvement can be to reduce the bias and train in a diverse set of data.

VII. References

[1] “Shrivastava et. al, “Comparative analysis of deep learning image detection algorithms,” 2021, JournalofBigData. [Online]. Available: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00434-w

[2] S A Sanchez et al, “A review: Comparison of performance metrics of pretrained models for object detection using the TensorFlow framework. [Online]. Available: https://iopscience.iop.org/article/10.1088/1757-899X/844/1/012024/pdf

[3] “Treatment episode data set: discharges (TEDS-D): concatenated, 2006 to 2009.” U.S. Department of Health and Human Services, Substance Abuse and Mental Health Services Administration, Office of Applied Studies, August, 2013, DOI:10.3886/ICPSR30122.v2

[4] “Wu et al. - A study on how pre-trained models affect detection using TensorFlow. Access the study at IOP Science” [Online]. Available: https://iopscience.iop.org/article/10.1088/1757-899X/844/1/012024/pdf

[5] “**Lin et al.** - "Focal Loss for Dense Object Detection," explaining the design and advantages of RetinaNet” [Online]. Available: https://arxiv.org/abs/1708.02002

[6] “Wadhwa et al- " Comparison of YOLOv8 and Detectron2 on Crowd Counting techniques," IEEE” [Online]. Available: https://ieeexplore.ieee.org/document/10391466