Center for Global Cyber Strategy

–

Object Of Search

Data Science Capstone (COMP4449)

Aakarsh Sagar

Introduction

Object Detection is an essential Computer Vision task used to detect the instances of objects of certain classes in an image or a set of images, or video frames. The goal of this task is to develop a deep learning model that answers the question: “What objects are where?”

Object detection forms the basis of many other computer vision tasks such as image segmentation, image captioning, object tracking, among others.

Objectives

Numerous “White hat” hacker organizations have joined forces to fight back against malicious cyber-attacks. One such anonymous White Hat group accidentally launched a cyber event that took down the global internet. The Center for Global Cyber Strategy(CGCS) narrowed its vast graph of offline records down to 40 individuals of interest in which a sub-group of eight individuals might have triggered the event. This group rarely meets in-person but uses a special item – a totem – as a secret signal of their affiliation.

The CGCS gathered all records associated with those individuals posted on the social media platform, “Y* INT”, including photos and captions.

The objective of this project is to identify the “totem” and the group of people in possession of it.

Dataset Description

The data contains Images captured by each of the individuals of interest. Some Images also contain captions characterized by text files. These images have been run through a machine learning model that identify objects in the image and classify them according to one of the 43 classes. The output has these features: “x”, "y", "Width", "Height"," Score", "Label". An example of such an output is shown below:

X	Y	Width	Height	Score	Label
830	860	1140	1470	0.50916	birdCall
1070	1120	650	930	0.34633	cloudSign
1260	1370	660	480	0.40703	eyeball
750	470	1300	1820	0.53835	pumpkinNotes
960	890	860	1050	0.31679	hairClip
2090	1380	320	460	0.59546	lavenderDie
960	840	860	1560	0.3109	stickerBox

Figure 1Image file associated with the table above

These model outputs look like YOLO model outputs where: “x”, "y", "Width", "Height" are the bounding box coordinates. x and y are coordinates of the center of the box along with width and height giving us the perimeter of the bounding boxes. The “score” column denotes how confident the model was in detecting and classifying the object it has detected.

There is also a TrainingImages dataset in case I want to run my own object detection model instead of choosing the provided model outputs. This TrainingImages dataset contains 12 images for each of the 43 classes, giving me a total of 516 un-annotated images to train and validate the custom model on.

Data Preparation and Analysis

The TrainingImages dataset is un-annotated. This means each image needs to be annotated manually by drawing bounding boxes around the objects in the image and labelling them. If the object is something that can be disassembled, each segment of that object needs to have a bounding box drawn around it and be labelled.

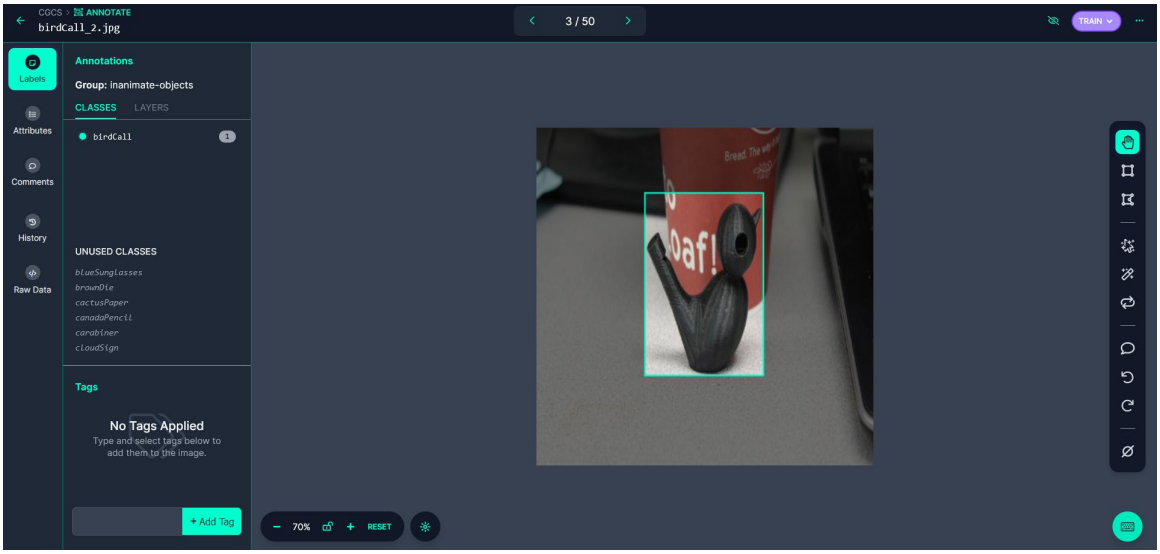


Figure 2: example of manullay annotating an image on roboflow.ai

This process was repeated on 516 images of the TrainingImages dataset. The annotation text files for each image contain the class_id, x, y, width, height coordinates of the object in the image. After annotation, the images were pre-processed by applying auto-orientation, and resized to 640x640 to be compatible with the YOLOv8 models. After the pre-processing step, Image augmentation was applied to these images. Techniques such as:

1. **Flip**: Horizontal & Vertical
2. **Shear**: ±10° Horizontal, ±10° Vertical
3. **Brightness**: Between -20% and +20%
4. **Exposure**: Between -15% and +15%

The Pre-Processing and augmented images were split into a 84%/16% Train-Validation split (2704 / 505 images) which resulted in a total 3209 images.

1. ## Visualization:

A histogram was plot on all three datasets to show the distribution of the images across the 43 labels. It is observed that there are certain labels that have close to 2100 samples while a majority of the labels barely have 500 samples. This non-uniform distribution would affect a model's performance due to lack of enough training data.

Model Implementation

1. Interpreting the results of the given machine learning model outputs

The model outputs csv files were merged for each person to check what kind of objects were identified and if the confidence scores were acceptable or not. For example: for person 1, there were about 62 classifications (many duplicate) along with different confidence scores.

	x	y	Width	Height	Score	Label	Person
0	2043	1109	1358	615	0.25584	eyeball	Person1
1	1600	384	1707	1986	0.50089	pumpkinNotes	Person1
2	310	11	820	1795	0.53331	sign	Person1
3	14	202	2043	1916	0.41412	silverStraw	Person1
4	1775	414	1385	1986	0.41576	yellowBalloon	Person1
...	...	...	...	...	...	...	...
57	1076	726	1600	1351	0.40620	metalKey	Person1
58	1224	1170	1304	484	0.33216	blueSunglasses	Person1
59	2460	11	1129	817	0.36974	hairClip	Person1
60	3065	91	457	444	0.30791	lavenderDie	Person1
61	1896	11	2151	2067	0.29958	yellowBag	Person1

62 rows × 7 columns

The model mis-classified many objects in the model. For example, the below image shows an image that contains an object called noisemaker. The machine learning model mis-classified the object while also identifying objects that were not even in the image.

x	y	Width	Height	Score	Label
1479	11	1922	1583	0.33861	birdCall
2554	313	955	888	0.28139	eyeball
1855	1724	578	1079	0.34569	pinkCandle
14	666	1465	2309	0.42683	pumpkinNotes
1560	1301	1197	615	0.32564	blueSunglasses
14	464	1775	2268	0.38394	yellowBag

x	y	Width	Height	Score	Label
3199	11	793	928	0.25478	cloudSign
2930	1684	1116	716	0.35547	eyeball
14	1644	1600	1392	0.32169	partyFavor
14	11	1855	1654	0.26094	pumpkinNotes
1479	1160	1331	444	0.35003	blueSunglasses
3119	1563	928	1039	0.5674	hairClip
2487	293	484	454	0.58108	lavenderDie

The above image only contains one object of Interest, .i.e., lavenderDie. But the model hallucinates many objects that were not present in the image.

Overall this model seems to lack the ability to detect objects in the image. A new model would need to be trained on the annotated images from the previous section and tested on the images provided for each Person.

2. You Only Look Once ver. 8 (YOLOv8) model

The YOLOv8 is a pytorch pre-trained model provided by Ultralytics. I chose the YOLOv8m model for this project because of its lower computational requirements, latency, and the right amount of parameters. This model has 295 layers, 25.88 million parameters, and 25.881 million gradients. This YOLOv8 model is designed for object detection tasks. The model was run for 50 epochs with the AdamW optimizer, with default values for the box loss, classification loss, distributed focal loss parameters.

```
Ultralytics YOLOv8.2.28 🚀 Python-3.10.13 torch-2.1.2 CUDA:0 (Tesla P100-PCIE-16GB, 16276
Model summary (fused): 218 layers, 25864657 parameters, 0 gradients, 78.8 GFLOPs
```

Class	Images	Instances	Box(P	R	mAP50	mAP50-95):
all	505	516	0.983	0.99	0.994	0.916
birdCall	10	10	1	1	0.995	0.952
blueSunglasses	6	6	0.959	1	0.995	0.904
brownDie	11	11	0.98	1	0.995	0.875
cactusPaper	8	8	0.968	1	0.995	0.983
canadaPencil	8	8	0.979	1	0.995	0.925
carabiner	15	15	0.983	1	0.995	0.976
cloudSign	11	11	1	1	0.995	0.977
cowbell	12	12	0.942	1	0.995	0.969
cupcakePaper	10	10	1	1	0.995	0.955
eyeball	9	9	0.971	1	0.995	0.958
foamDart	14	14	0.981	1	0.995	0.956
gClamp	12	12	0.98	1	0.995	0.961
giftBag	12	12	0.979	1	0.995	0.97
glassBead	7	18	0.985	0.944	0.966	0.856
gyroscope	8	8	0.968	1	0.995	0.984
hairClip	10	10	0.976	1	0.995	0.949
hairRoller	16	16	0.985	1	0.995	0.918
lavenderDie	11	11	0.984	1	0.995	0.826
legoBracelet	11	11	0.982	1	0.995	0.907
metalKey	13	13	1	1	0.995	0.918
miniCards	13	13	0.974	1	0.995	0.947
noiseMaker	11	11	0.979	1	0.995	0.916
paperPlate	10	10	1	1	0.995	0.989
partyFavor	14	14	0.985	1	0.995	0.869
pinkCandle	13	13	0.985	1	0.995	0.762
pinkEraser	9	9	1	1	0.995	0.929
plaidPencil	9	9	1	0.987	0.995	0.934
pumpkinNotes	13	13	0.979	1	0.995	0.963
rainbowPens	14	14	0.984	1	0.995	0.886
redBow	6	6	0.96	1	0.995	0.931
redDart	9	9	1	1	0.995	0.889
redWhistle	12	12	0.978	1	0.995	0.938
rubiksCube	15	15	0.984	1	0.995	0.831
sign	8	8	0.973	1	0.995	0.936
silverStraw	15	15	1	0.835	0.988	0.73
spiderRing	15	15	0.982	1	0.995	0.847
stickerBox	16	16	0.983	1	0.995	0.932
trophy	14	14	0.982	1	0.995	0.933
turtle	12	12	1	1	0.995	0.877
vancouverCards	16	16	0.983	1	0.995	0.97
voiceRecorder	12	12	0.98	1	0.995	0.946
yellowBag	18	18	1	0.799	0.995	0.85
yellowBalloon	17	17	0.981	1	0.995	0.883

```
Speed: 0.2ms preprocess, 10.0ms inference, 0.0ms loss, 2.8ms postprocess per image
```

Figure 3: Model Summary

The model's training results were satisfactory. The mean of the average precision(mAP(50-95)) metric that tells us how precise the model was when detecting the tougher objects, such as when there are multiple objects in the image, or if the model is able to detect objects in a crowded background. The mAP(50-95) metric in my model had a score of 0.916 for all classes with the Recall and Precision score at 0.99 and 0.983.

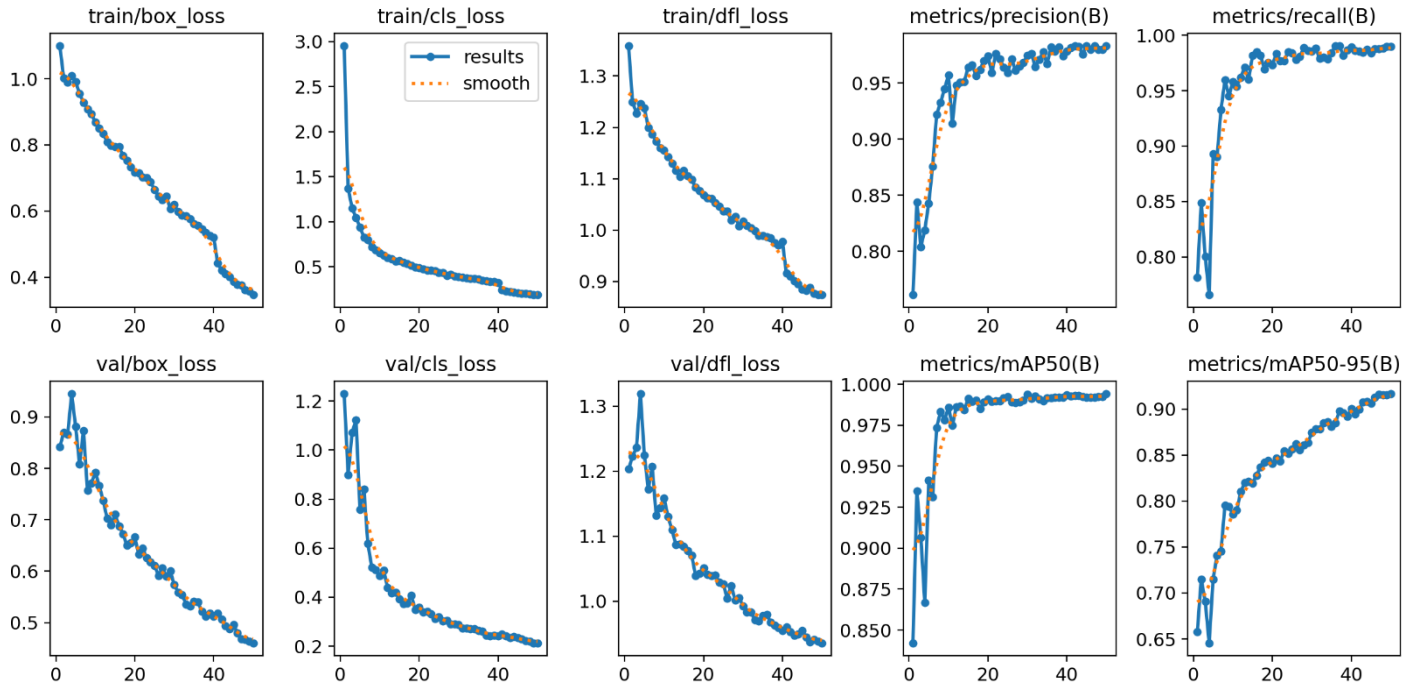


Figure 4: Results of the training model.

Training and Validation metrics Inference:

1. **train/box_loss**: Measures loss related to bounding box predictions. Curve shows a steady decrease, indicating the model's ability to predict bounding boxes as training progresses.
2. **train/cls_loss**: represents the classification loss, related to how well the model is distinguishing between different object classes. The curve also steadily decreases, showing that the model's classification accuracy is improving over time
3. **train/dfl_loss**: is a variant of focal loss that helps improve model performance when training data is imbalanced. The consistent downward trend suggests that the model's overall error in predictions is reducing.
4. **val/box_loss**: It follows a decreasing trend, indicating good generalization of bounding box predictions to unseen data.
5. **val/cls_loss**: The decrease, although with some fluctuations, shows improving classification performance on validation data.
6. **val/dfl_loss**: The downward trend indicates reducing errors in predictions on validation data, with some minor fluctuations.
7. **precision(B)**: The precision is quite high, approaching 0.95, and shows improvement as training progresses.
8. **recall(B) :** This metric is also high, approaching 0.9, indicating that the model is successfully identifying most of the actual positives.
9. **metrics/mAP50(B) :** Mean Average Precision at IoU threshold 0.50. This is a standard metric for object detection, combining both precision and recall. The mAP50 metric shows significant improvement, stabilizing around 0.95, which indicates high accuracy in object detection.
10. **metrics/mAP50-95(B):** Mean Average Precision across different IoU thresholds from 0.50 to 0.95. This metric provides a more stringent evaluation of the model's performance. The steady

increase and eventual stabilization around 0.9 suggest that the model performs well across various IoU thresholds, indicating robust detection capabilities.

The model's performance, as depicted by the provided graphs, shows strong learning and generalization capabilities. The steady reduction in loss metrics and high values in precision, recall, and mAP metrics suggest that the model is accurate and reliable in detecting and classifying objects. The minor fluctuations in validation losses indicate that there might be some overfitting, but overall, the performance metrics are quite strong.

Validation Performance:

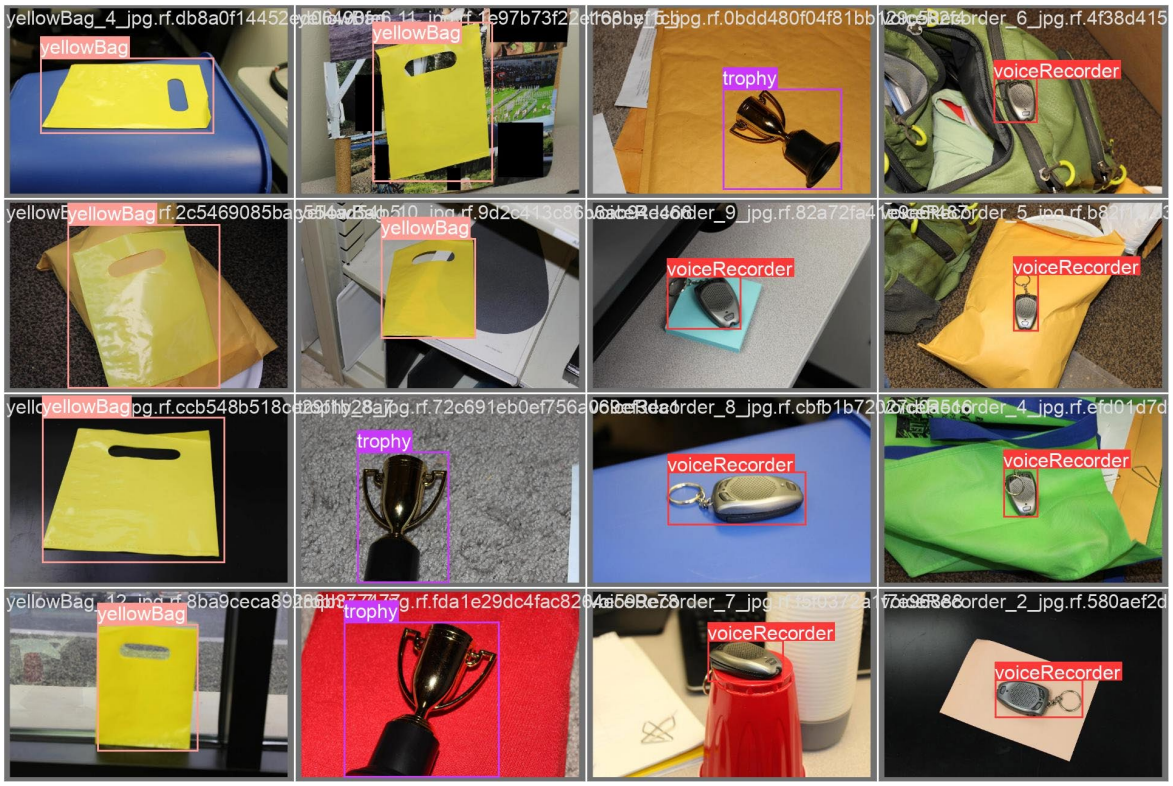


Figure 5: Validation batch 1 labels

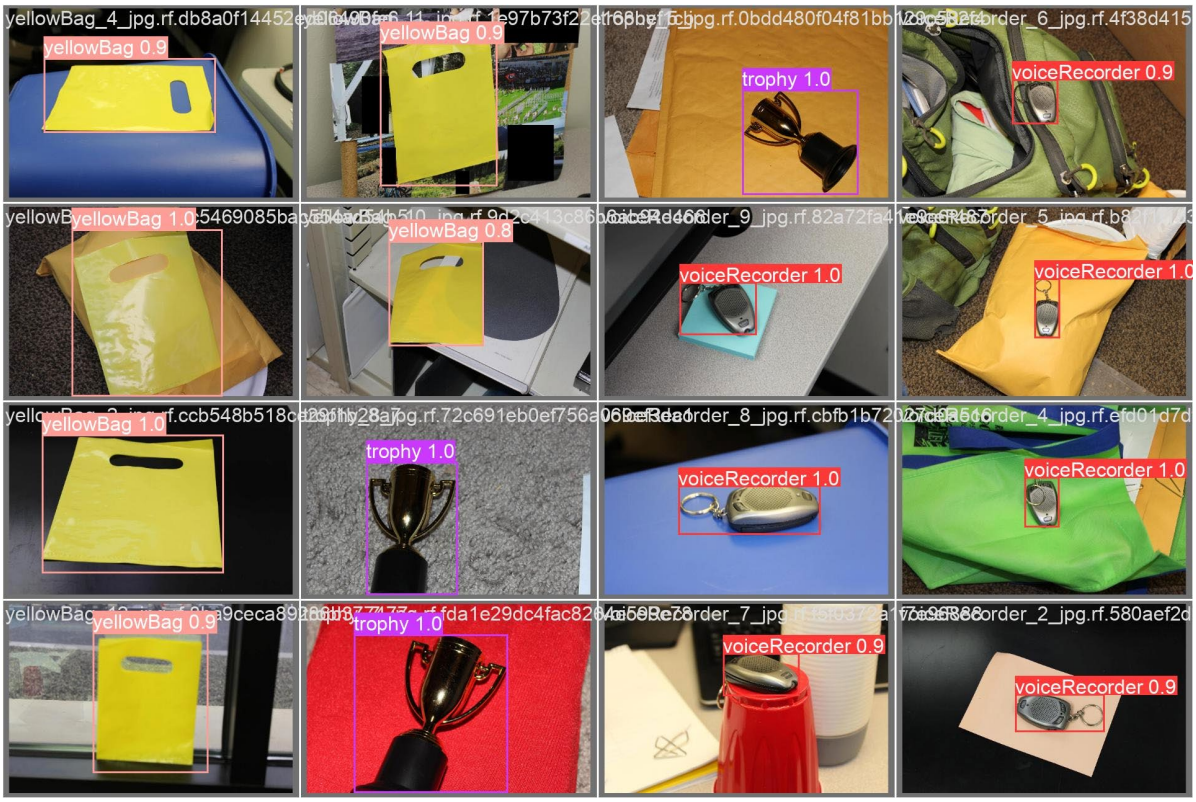
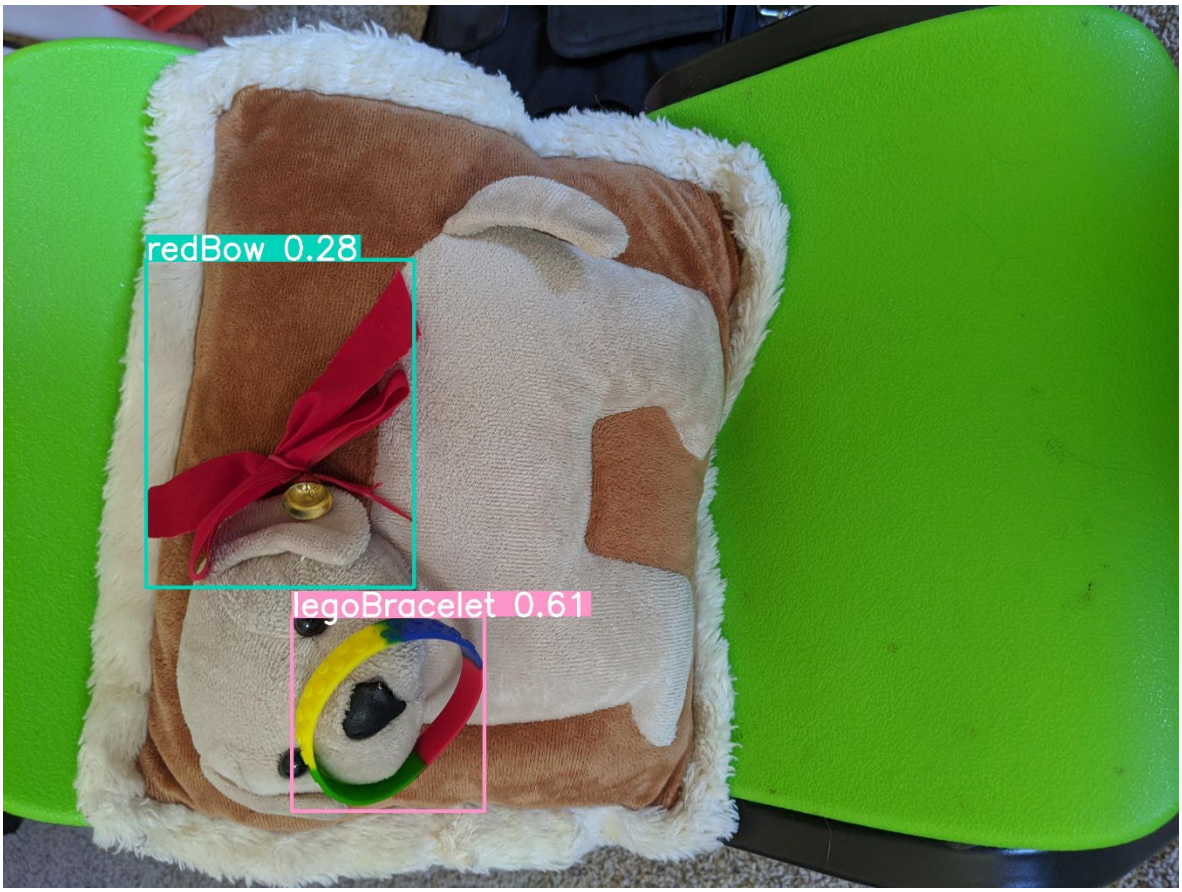


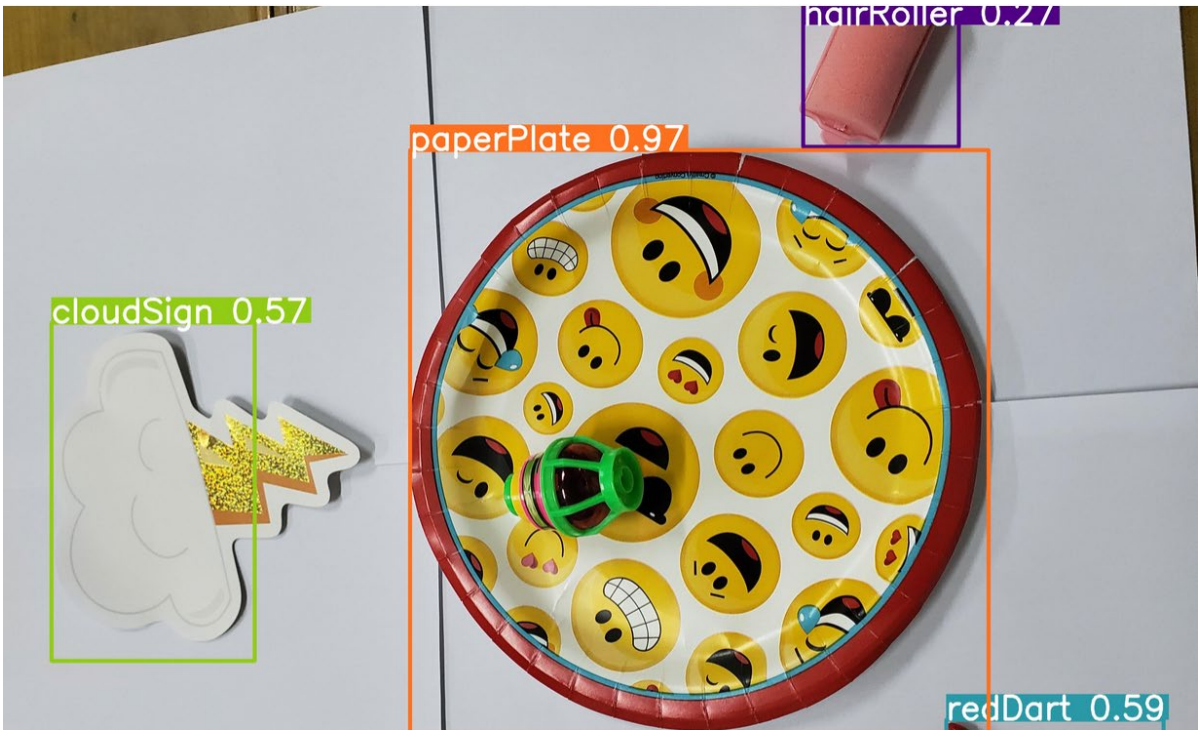
Figure 6: Validation batch 1 predictions

The model performed well on validation set, but with the validation metrics saying it might have overfit the data, we might see some mis-classifications and detection errors on the un-annotated test dataset.

Model Performance on Test Dataset:



The model was able to perform well at identifying objects and classifying them in the correct classes. Some objects were not identified in images due to the background being crowded, or too many object classes in the image. For example, in the below image, the gyroscope was not identified on the paper plate, while all other objects were identified.



Questions to be answered:

1. **Examine the outputs from the model – either from the detection results provided or the results from a model you chose. Which objects were identified well by the model and which were not?**

   Identified Objects

   - **Well Identified Objects**: The new YOLOv8m model seems to have good performance metrics on the training and validation datasets, which implies that objects such as lavenderDie, redDart, and rubiksCube might be well-identified given their frequent and correct detections across multiple images.
   - **Poorly Identified Objects**: The old YOLOv2 model misclassified several objects, such as the noiseMaker and lavenderDie, and hallucinated objects not present in the images, indicating poor performance in complex scenes or with objects having similar features.

2. **Demonstrate your process for using visual analytics to correct for classification errors in the results. How do you represent confidence and uncertainty? How could the correction process be made more efficient?**

   Due to lack of knowledge in labelling tools like LabelImg and CVAT, I was unable to correct the mis-classifications on the Test Dataset. Ideally the correction process can be made more efficient

by using semi-supervised learning: Leverage predictions with high confidence to auto-label a portion of the dataset. Second suggestion would be Active learning: Prioritize human review of images where the model has low confidence or high disagreement with ground truth.

3. **Characterize the distribution of objects across the forty people.**
    a. **Which people have which objects?**
    b. **Identify groups of people that have object(s) in common.**

People and Objects:

Person1: [yellowBalloon, miniCards, hairRoller]

Person10: [plaidPencil x3, partyFavor, voiceRecorder]

Person11: [carabiner, giftBag, cupcakePaper x2, rainbowPens]

Person12: [pumpkinNotes x2, noiseMaker, legoBracelet, cupcakePaper]

Person13: [partyFavor, miniCards, blueSunglasses, paperPlate x2, cupcakePaper]

(and so on for each person...)

Groups with Common Objects:

blueSunglasses: ['Person2', 'Person7', 'Person27', 'Person22', 'Person39', 'Person9', 'Person13']

redWhistle: ['Person7', 'Person37', 'Person27', 'Person5', 'Person33', 'Person6', 'Person23', 'Person18', 'Person38', 'Person29', 'Person3']

cowbell: ['Person17', 'Person21', 'Person38', 'Person35', 'Person4', 'Person3', 'Person16']

(and so on for each object...)

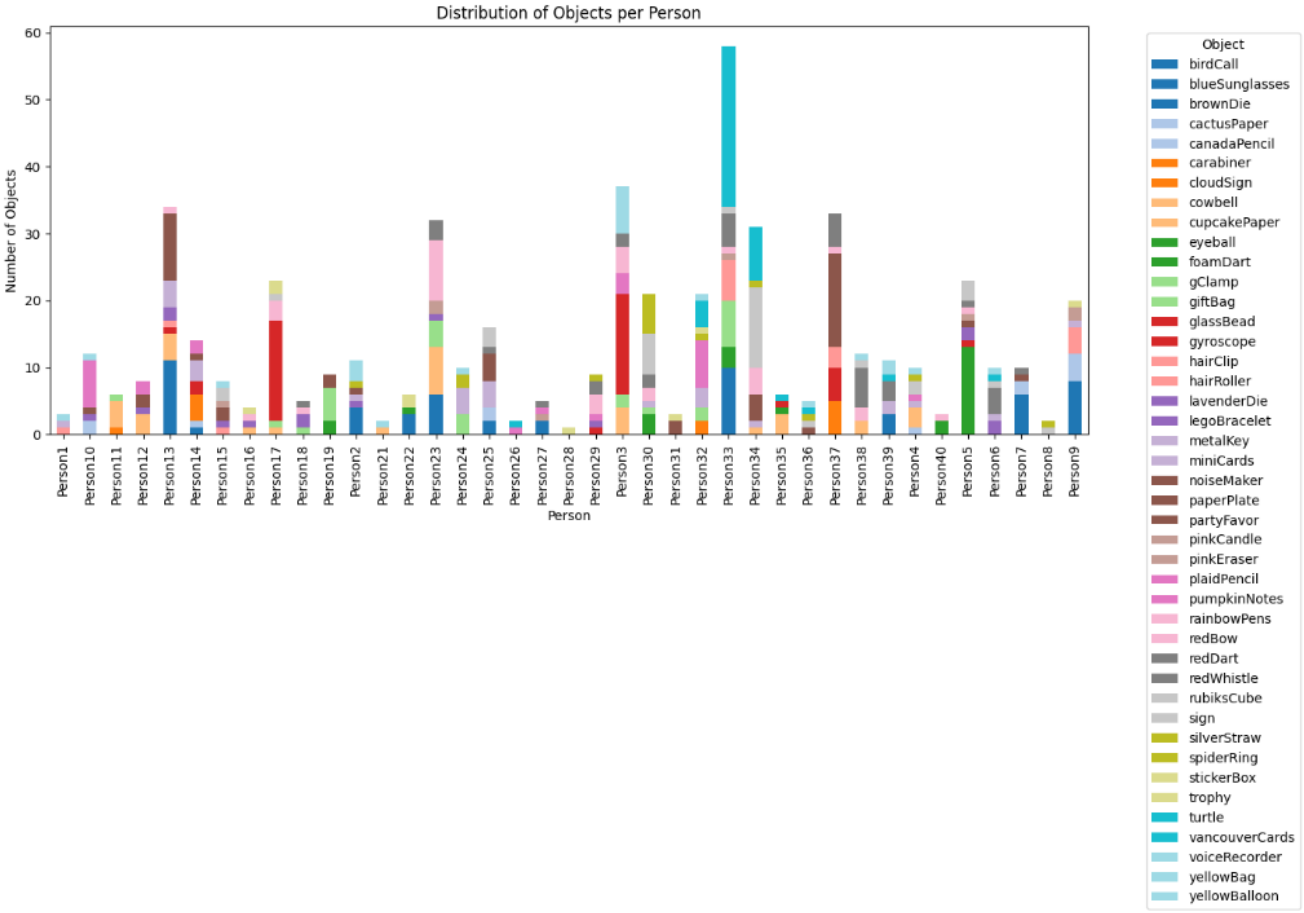


Figure 7: Distribution of objects among each person

4. **Which group do you think is the most likely group with the "totem"? What is your rationale for that assessment?**

```
Potential Totem Objects:
          class  unique_persons_count
17  lavenderDie                     8
30      redDart                     8
32   rubiksCube                     8
People holding potential totem objects:
         class                                                                person
0  lavenderDie   [Person2, Person15, Person10, Person6, Person23, Person18, Person29, Person16]
1      redDart  [Person37, Person30, Person33, Person39, Person25, Person38, Person23, Person3]
2   rubiksCube     [Person17, Person30, Person5, Person6, Person38, Person8, Person4, Person34]
```

It seems there might be 3 potential totems in the possession of exactly 8 different people. I am going to cross-verify with the images since the model might have mis-classified some images.

The cross-verification of the images is successful in this case, because there is evidence of many False positive detection and identification of the objects: 'redDart' and 'rubiksCube'.

The Totem is likely the 'lavenderDie' with the group of people being:

1. Person2
2. Person15
3. Person10
4. Person6
5. Person23
6. Person18
7. Person29
8. Person16

5. **Process question: Did you choose to use the object recognition model results provided or use your own machine learning algorithm? Why did you make that choice? What was the biggest challenge you faced?**

I initially chose to use the object recognition model results (yolov2) provided to set a baseline of what data analysis I could perform on the detected results, and how effective the model was on the unseen Testing dataset. The object recognition model did not give me good results. The reason is that the data only provides one score for each bounding box, which hinders us from reasoning the decision's uncertainty. As the score only represents the detector's confidence that the bounding box contains the specific object, it does not tell the uncertainty of this decision. Considering that most object detectors also provide the score distribution for each bounding box, I switched over to training my own YOLOv8m model using the TrainingImages dataset that contains about 516 images(12 images per class).

The biggest challenge I faced was the manual annotation of the Training Images. It was the first time I had ever performed annotations on a image dataset, so there was a lot of room for imporvement. Sometimes, the model had trouble detecting an object among a crowded background, or if there were too many objects of interest, the model would not detect any object.

This model, while having good performance metrics on the training and validation datasets, did not perform as well on the Testing Image.