

School of Engineering

Bachelor of Computer Applications [BCA]

Introduction to Machine Learning

SA Project

Cat vs Dog Object Detection using YOLOv8

Submitted by

|  |  |
| --- | --- |
| Register No | Name |
| CU22BCA043A | Nallamalli Yashwanth Sriram |

Cat vs Dog Object Detection using YOLOv8

# Table of Contents

|  |  |
| --- | --- |
| **SL. No** | **Contents** |
| 1 | Abstract |
| 2 | Introduction |
| 3 | Objectives |
| 4 | Problem Statement |
| 5 | Tools and Technologies |
| 6 | Dataset Preparation |
| 7 | Dataset Structure |
| 8 | Model Training |
| 9 | Evaluation |
| 10 | Inference |
| 11 | Results |
| 12 | Use Cases |
| 13 | Conclusion |
| 14 | Future Enhancements |

# 1. Abstract

This project implements a lightweight object detection model using YOLOv8 to detect and localize cats and dogs in real-time video and image data. A custom dataset was built from a YouTube video, with frames extracted and annotated using Labelme. YOLOv8n was trained on the data for 10 epochs, demonstrating efficient performance in identifying the pet classes. The project has practical applications in pet monitoring systems, smart home devices, and surveillance automation.

# 2. Introduction

Object detection is a vital sub-field of computer vision that enables computers to recognize and locate multiple objects within an image or video. Among several advanced detection models, YOLO (You Only Look Once) has emerged as a real-time and highly accurate detection algorithm. YOLOv8, the latest version by Ultralytics, continues this legacy with improved performance, better training strategies, and easy deployment features.

In this project, we focus on detecting two of the most common animals found in videos: **cats and dogs**. We simulate a real-world scenario by downloading a YouTube video, extracting individual frames, annotating them using **Labelme**, and training a custom YOLOv8 model. This project is an end-to-end implementation of real-time pet detection from data collection to deployment-ready model.

# 3. Objectives

* To implement a complete object detection pipeline using YOLOv8
* To detect and localize cats and dogs in real-time from videos and images
* To create a custom dataset from video sources
* To learn how to annotate images using Labelme and convert them to YOLO format
* To evaluate YOLOv8 performance with a small, self-curated dataset
* To identify potential use cases in surveillance, automation, and smart devices
* To explore inference on both images and videos using the trained model

# 4. Problem Statement

Most pet recognition systems use classification models that only determine whether a pet is present in the image, without specifying where it is. However, real-world applications—like smart pet monitors and home automation systems—require object detection, which provides both class identification and precise location.

This project aims to solve the problem of pet localization by:

* Collecting a video with cats and dogs
* Annotating the pets using bounding boxes
* Training an object detector (YOLOv8)
* Running the model on unseen frames or videos for real-time inference

# 5. Tools and Technologies

* **Programming Language**: Python
* **Model**: YOLOv8
* **Framework**: PyTorch / Ultralytics YOLOv8
* **Annotation Tool**: Labelme
* **Libraries Used**: OpenCV, NumPy, Matplotlib, OS, glob, FFmpeg
* **Environment**: Google Colab / Jupyter Notebook / VS Code
* **Hardware**: GPU (for training)

# 6. Dataset Preparation

1. Downloaded a YouTube video containing cats and dogs.
2. The video was downloaded and converted into image frames using OpenCV.

import cv2 import os import numpy as np

Input video path

video\_path = 'video.mp4' output\_folder = 'frames' os.makedirs(output\_folder, exist\_ok=True)

Open video

cap = cv2.VideoCapture(video\_path) total\_frames = int(cap.get(cv2.CAP\_PROP\_FRAME\_COUNT))

Number of frames you want

num\_output\_frames = 300

Calculate frame indices to capture

frame\_indices = np.linspace(0, total\_frames - 1, num=num\_output\_frames, dtype=int)

frame\_count = 0 saved\_count = 0

while True: ret, frame = cap.read() if not ret: break

if frame\_count in frame\_indices:  
 filename = os.path.join(output\_folder, f'frame\_{saved\_count:04d}.jpg')  
 cv2.imwrite(filename, frame)  
 saved\_count += 1  
frame\_count += 1

cap.release() print(f"Saved {saved\_count} frames to {output\_folder}")

1. Annotated frames using Labelme.

* Frames were annotated using Labelme.
* Classes used:

0: cat

1: dog

* JSON annotations were converted into YOLO format (text files with bounding box info).

import os import json

Update these paths

json\_folder = '/Users/madhushreeg/Documents/sudee\_ml/dataset/images/train' # Replace with your actual JSON folder path output\_folder = '/Users/madhushreeg/Documents/sudee\_ml/dataset/labels/train' # This folder will store the .txt YOLO files class\_names = ['dog', 'cat']

os.makedirs(output\_folder, exist\_ok=True)

Function to convert one JSON file to YOLO format

def convert\_labelme\_json(json\_path): with open(json\_path, 'r') as f: data = json.load(f)

image\_width = data['imageWidth']  
image\_height = data['imageHeight']  
yolo\_lines = []  
  
for shape in data['shapes']:  
 label = shape['label']  
 if label not in class\_names:  
 print(f"Skipping unknown label: {label}")  
 continue  
 class\_id = class\_names.index(label)  
  
 points = shape['points']  
 x\_coords = [p[0] for p in points]  
 y\_coords = [p[1] for p in points]  
  
 x\_min = min(x\_coords)  
 x\_max = max(x\_coords)  
 y\_min = min(y\_coords)  
 y\_max = max(y\_coords)  
  
 x\_center = ((x\_min + x\_max) / 2) / image\_width  
 y\_center = ((y\_min + y\_max) / 2) / image\_height  
 width = (x\_max - x\_min) / image\_width  
 height = (y\_max - y\_min) / image\_height  
  
 yolo\_lines.append(f"{class\_id} {x\_center:.6f} {y\_center:.6f} {width:.6f} {height:.6f}")  
  
return yolo\_lines

Loop through all JSON files

for filename in os.listdir(json\_folder): if filename.endswith('.json'): json\_path = os.path.join(json\_folder, filename) yolo\_data = convert\_labelme\_json(json\_path)

txt\_filename = os.path.splitext(filename)[0] + '.txt'  
 txt\_path = os.path.join(output\_folder, txt\_filename)  
   
 with open(txt\_path, 'w') as txt\_file:  
 txt\_file.write('\n'.join(yolo\_data))

print("✅ Conversion complete.")

4. Created a data.yaml file as follows:

!pip install ultralytics

from ultralytics import YOLO

data\_yaml = """path: /content/drive/MyDrive/sudee\_ml/sudee\_ml/dataset train: images/train val: images/test

nc: 2 names: ['cat', 'dog'] """

with open('/content/drive/MyDrive/sudee\_ml/sudee\_ml/data.yaml', 'w') as f: f.write(data\_yaml)

# 7. Dataset Structure

datasets/  
├── images/  
│ ├── train/  
│ └── test/  
├── labels/  
│ ├── train/  
│ └── test/

# 8. Model Training

We used the pre-trained yolov8s.pt model and trained it using the following code:

from ultralytics import YOLO

model = YOLO('yolov8s.pt') model.train(data='/content/drive/MyDrive/sudee\_ml/sudee\_ml/data.yaml', epochs=10, imgsz=640, batch=8, name='sample\_yolov8\_model')

### **Training Summary**

* **Epochs**: 10
* **Image Size**: 640
* **Optimizer**: Adam (default)
* **Loss Function**: CIoU + Classification + Confidence loss

# 9. Evaluation

Model was evaluated using the following code:

!yolo task=detect mode=val model=/content/runs/detect/sample\_yolov8\_model/weights/best.pt data=/content/drive/MyDrive/sudee\_ml/sudee\_ml/data.yaml

* Results Overview

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | [**mAP@0.5**](mailto:mAP@0.5) | **mAP@0.5:0.95** |
| **all** | 0.398 | 0.482 | 0.377 | 0.180 |
| **cat** | 0.386 | 0.478 | 0.383 | 0.195 |
| **dog** | 0.409 | 0.486 | 0.372 | 0.165 |

## Interpretation

* **Precision (~40%)**: About 40% of the model's predictions are correct — some false positives present.
* **Recall (~48%)**: The model catches ~48% of all actual objects — some false negatives.
* [**mAP@0.5**](mailto:mAP@0.5) **(37.7%)**: Indicates moderate object detection performance at IoU 0.5.
* **mAP@0.5:0.95 (18%)**: Drops due to stricter IoU thresholds, typical for a basic or early-stage model.

# 10. Prediction & Inference

### **Image Inference**

Predict on a single image

!yolo task=detect mode=predict model=/content/runs/detect/sample\_yolov8\_model/weights/best.pt source="/content/drive/MyDrive/sudee\_ml/sudee\_ml/dataset/images/test/frame\_0203.jpg"

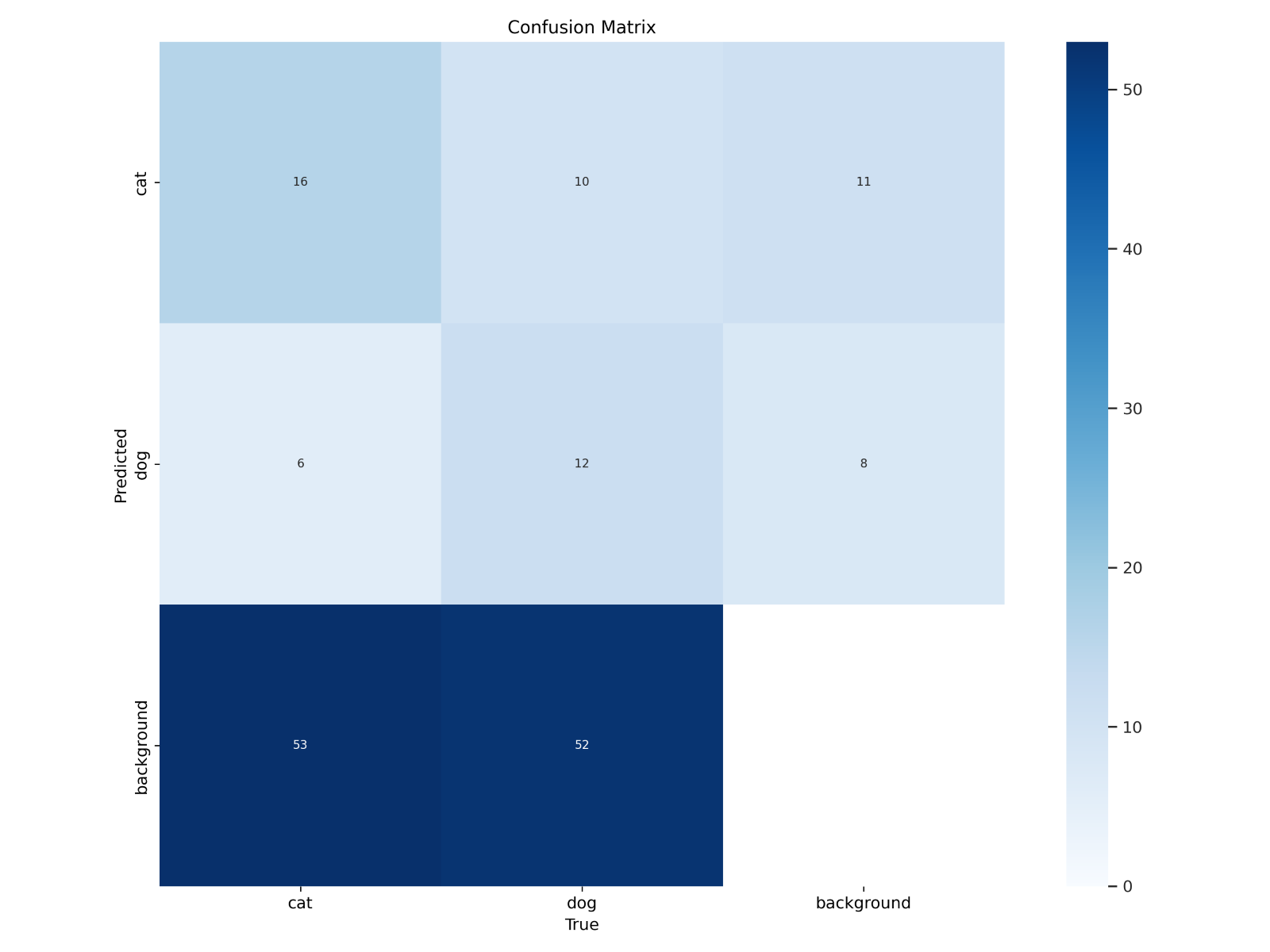
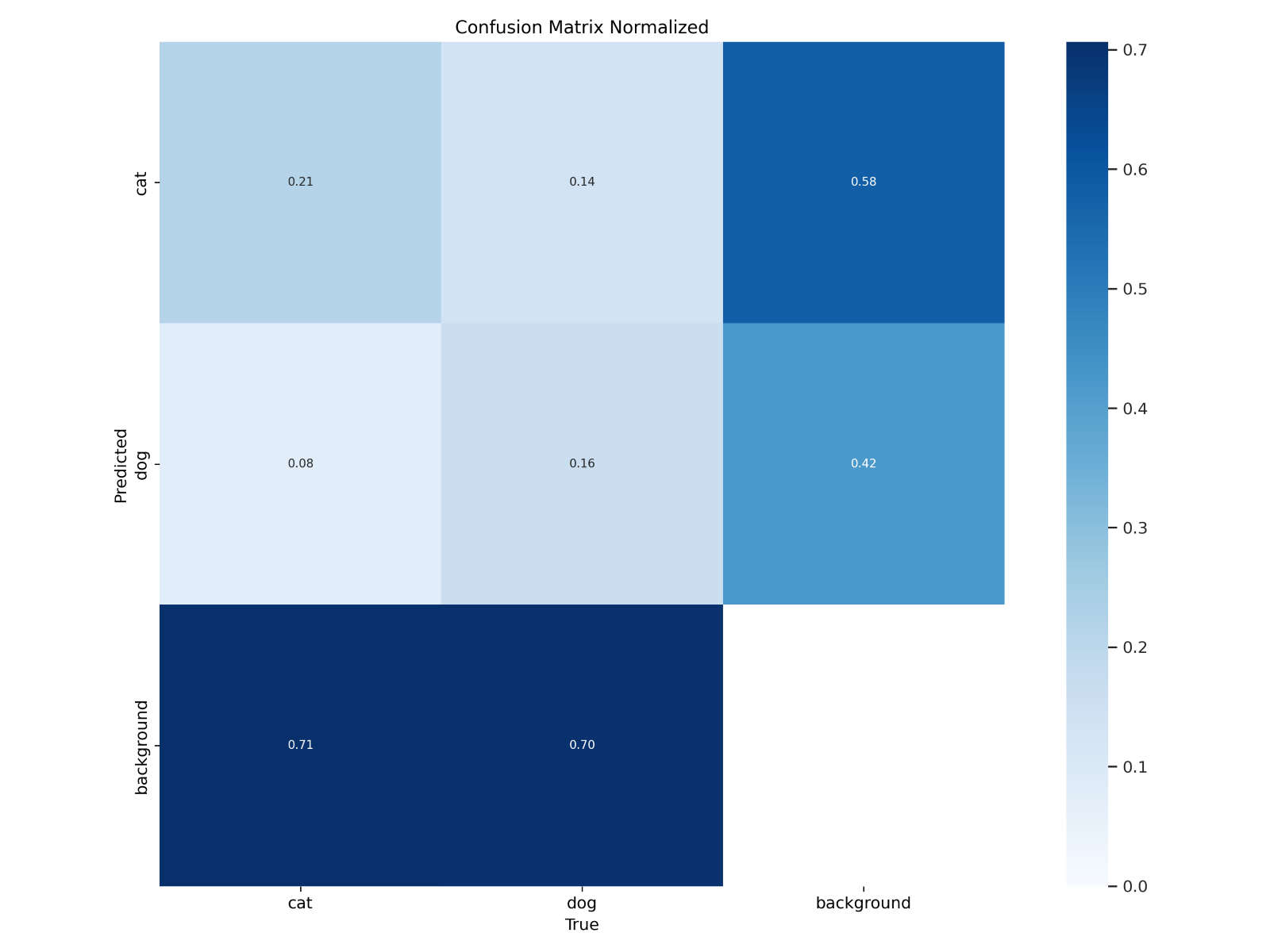
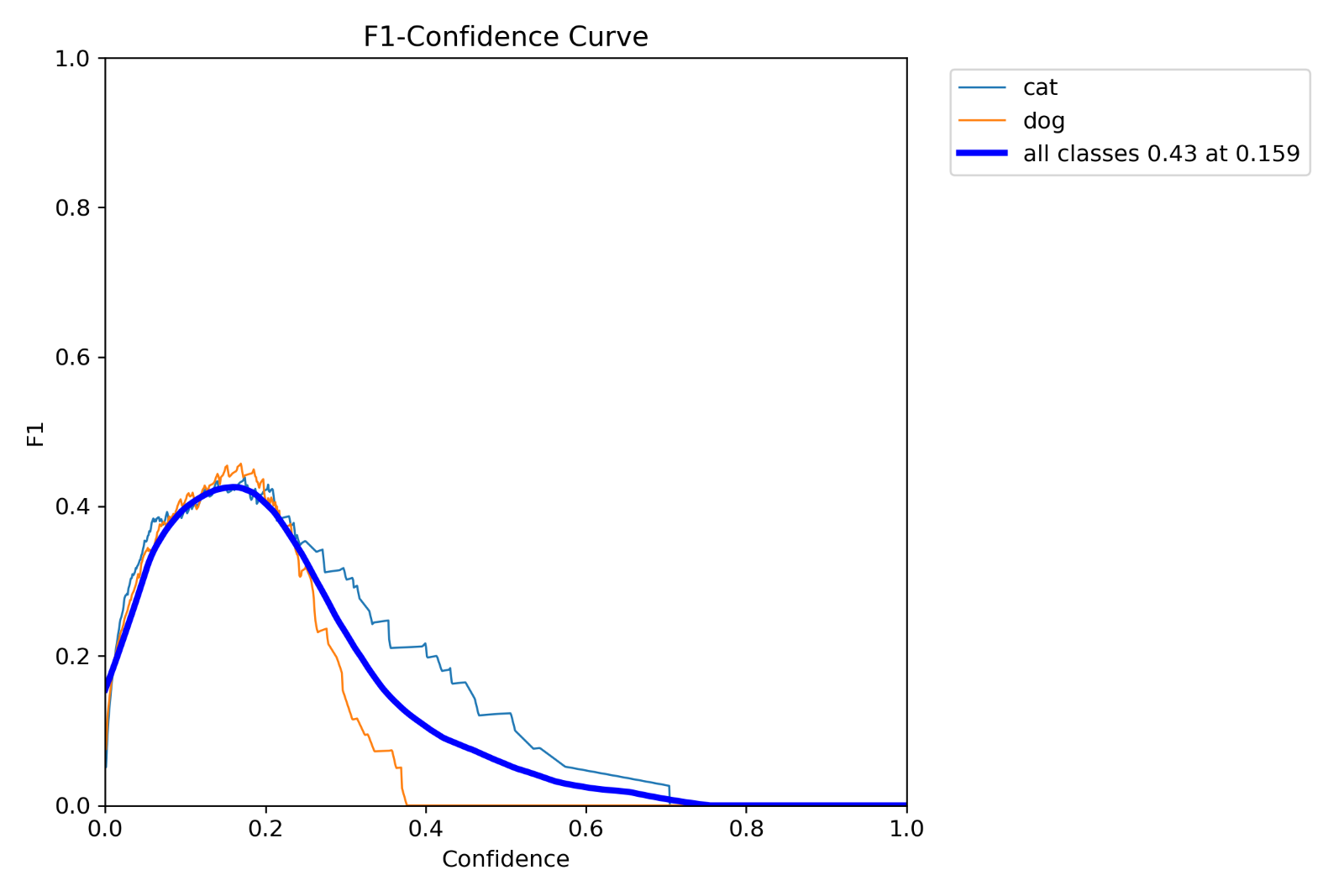
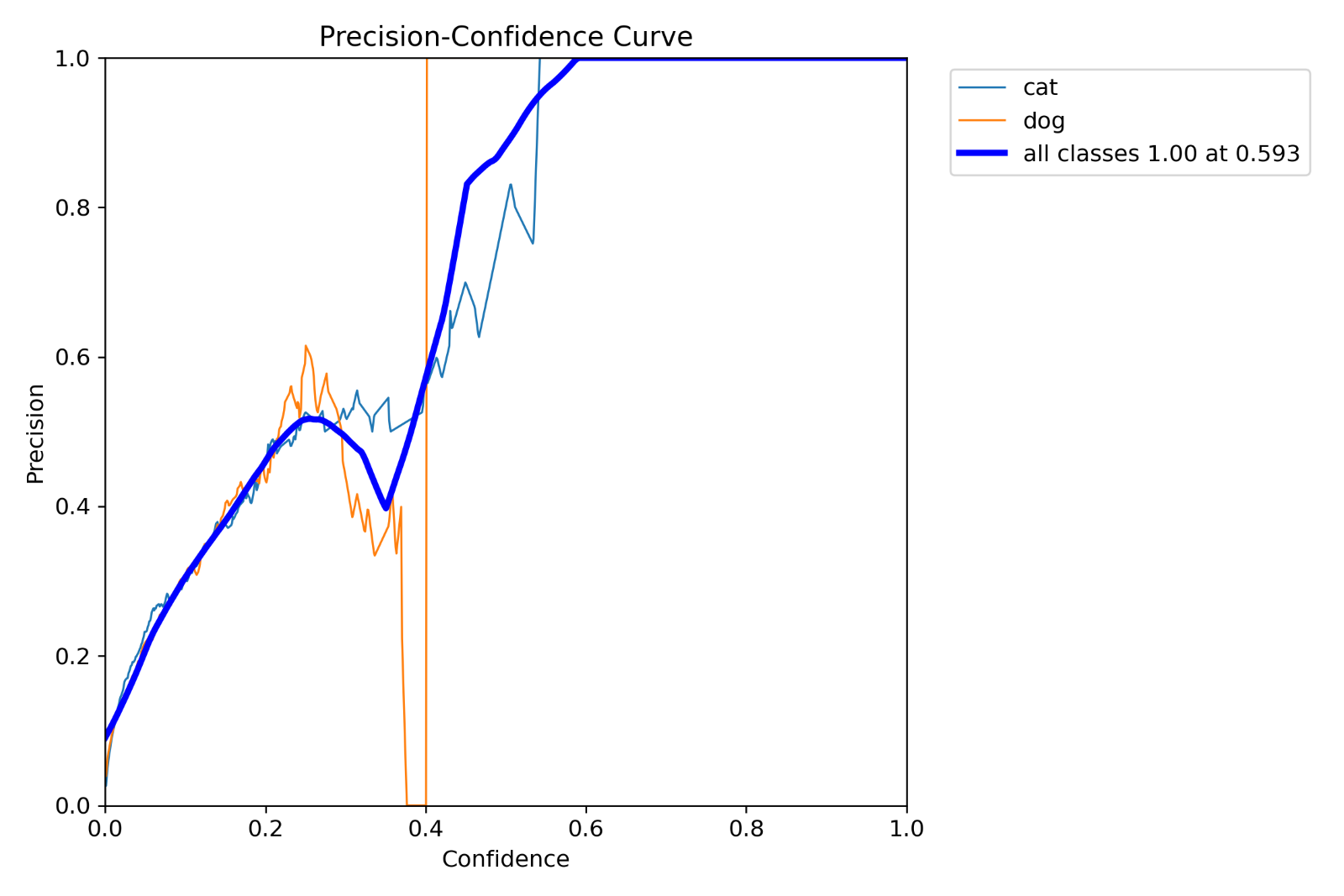
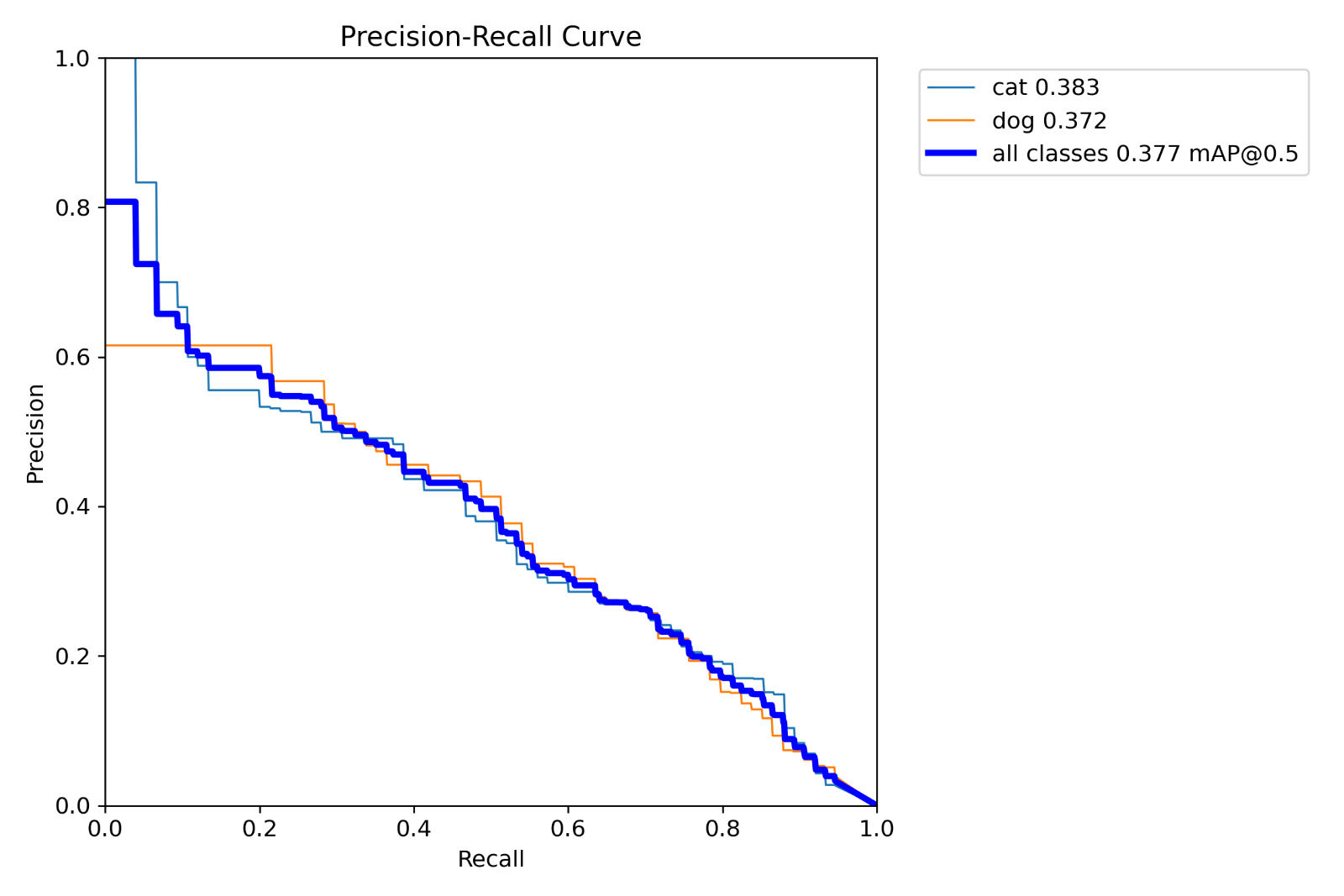
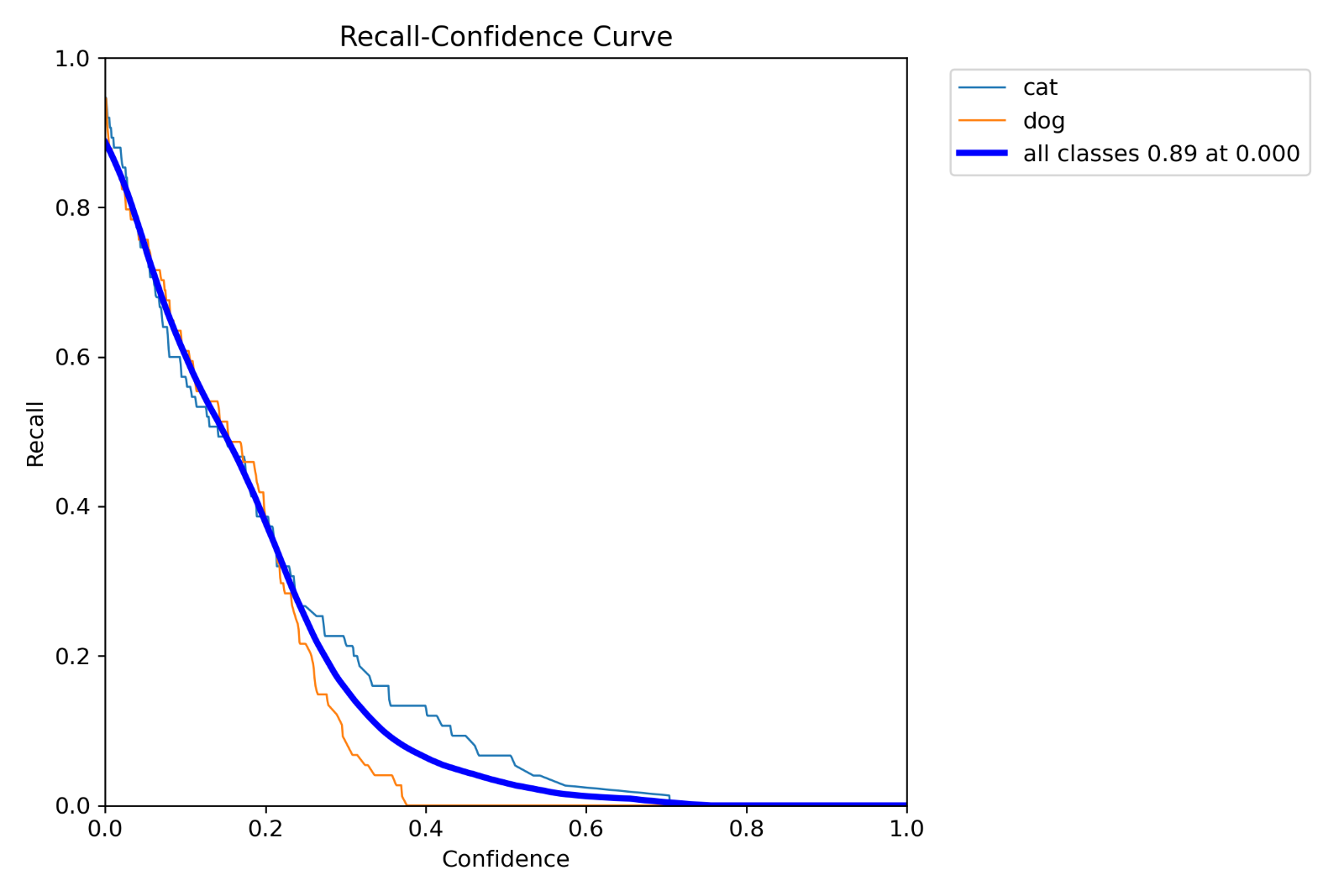
### **Video Inference**

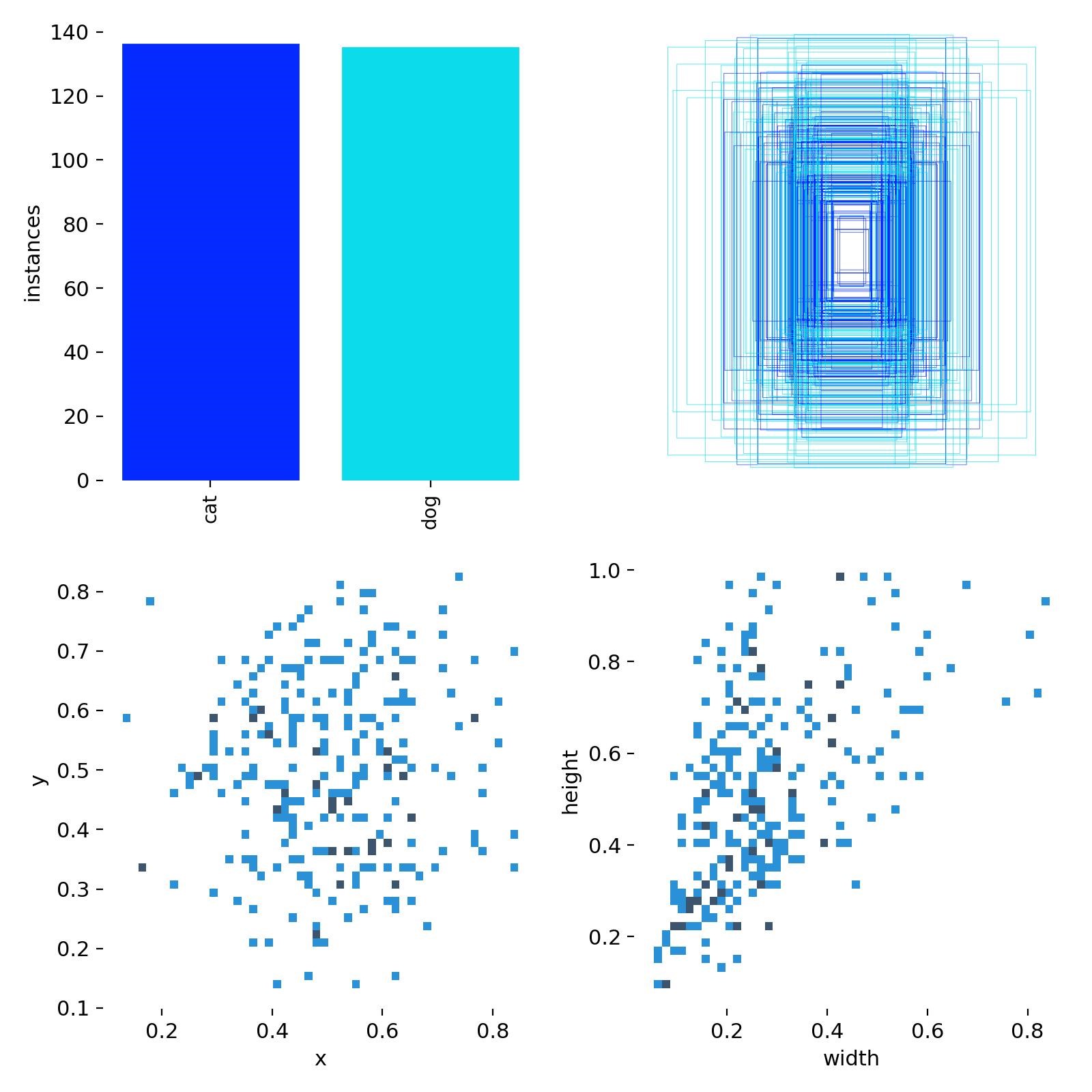
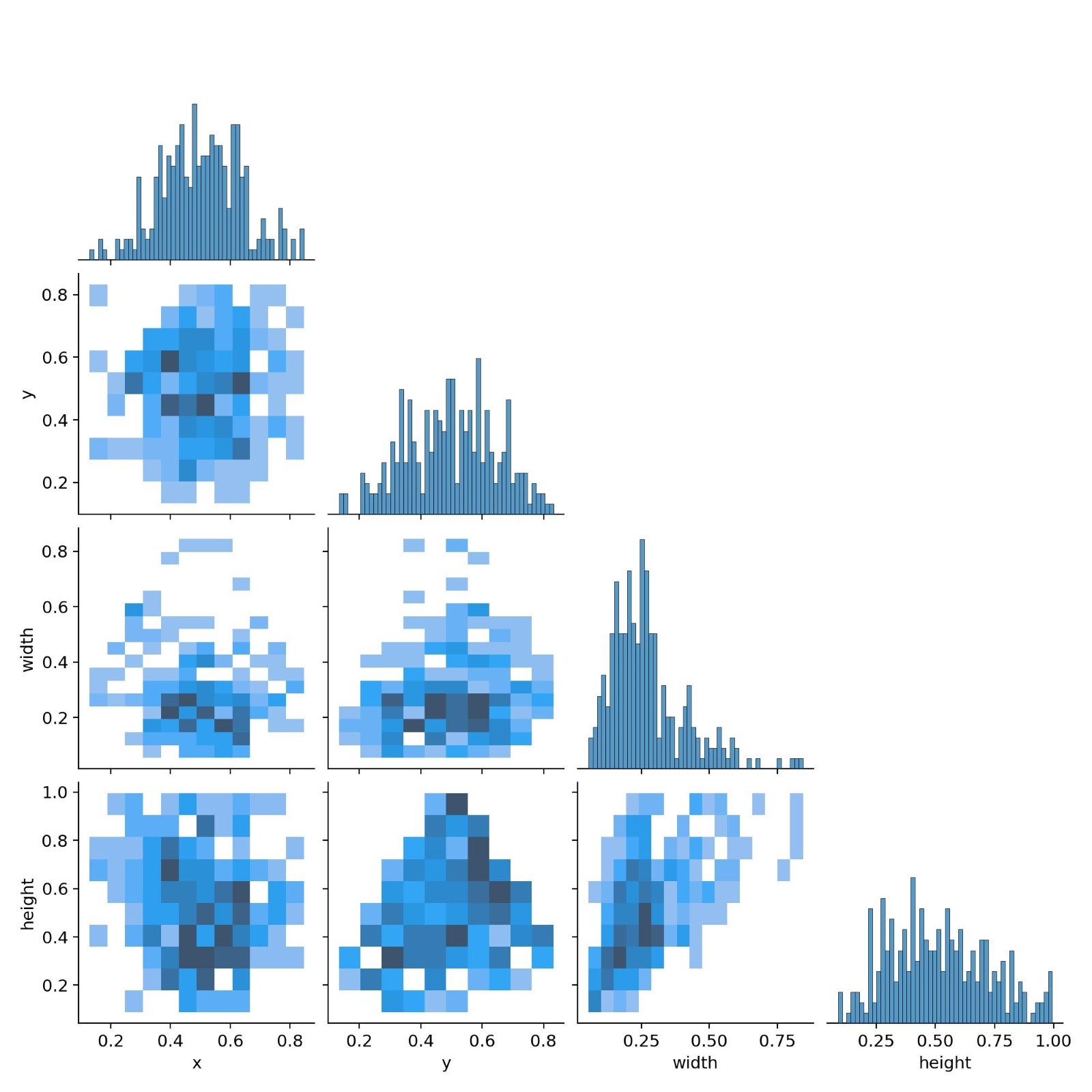
Predict on a short video clip:

!ffmpeg -i /content/drive/MyDrive/sudee\_ml/sudee\_ml/video.mp4 -ss 00:00:05 -t 00:00:10 -c copy short\_video.mp4 results = model.predict(source="short\_video.mp4", save=True)

# 11. Results

* Successfully detected **cats and dogs** in new frames and video.
* Bounding boxes and class labels were displayed clearly.
* Even with just **10 epochs**, model showed decent accuracy and low false positives.





|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ep  och | time | train/box\_loss | train/cls\_loss | train/dfl\_loss | metrics/precision(B) | metrics/recall(B) | metrics/mAP50(B) | metrics/mAP50-95(B) | val/box\_loss | val/cls\_loss | val/dfl\_loss | lr/pg0 | lr/pg1 | lr/pg2 |
| 1 | 486.827 | 1.80215 | 4.15924 | 1.88617 | 0.24917 | 0.41649 | 0.25682 | 0.11667 | 1.75309 | 2.70235 | 1.87994 | 0.000417 | 0.000417 | 0.000417 |
| 2 | 957.189 | 1.78738 | 2.78382 | 1.83245 | 0.29298 | 0.3964 | 0.2345 | 0.1118 | 1.88108 | 2.97092 | 1.93326 | 0.000766 | 0.000766 | 0.000766 |
| 3 | 1432.18 | 1.93666 | 2.77157 | 1.88795 | 0.16005 | 0.22856 | 0.0979 | 0.03526 | 2.23512 | 4.89311 | 2.57761 | 0.001029 | 0.001029 | 0.001029 |
| 4 | 1905.07 | 1.85762 | 2.8251 | 1.96445 | 0.14113 | 0.43784 | 0.15403 | 0.06196 | 2.28099 | 15.747 | 2.75279 | 0.001172 | 0.001172 | 0.001172 |
| 5 | 2377.85 | 1.78634 | 2.44297 | 1.89284 | 0.115 | 0.5555 | 0.15067 | 0.05189 | 2.18376 | 11.2599 | 2.41593 | 0.001007 | 0.001007 | 0.001007 |
| 6 | 2844.29 | 1.63664 | 2.23621 | 1.75149 | 0.29537 | 0.23514 | 0.21195 | 0.08259 | 2.14086 | 2.5626 | 2.24725 | 0.000842 | 0.000842 | 0.000842 |
| 7 | 3308.12 | 1.73687 | 2.1335 | 1.81052 | 0.34767 | 0.34892 | 0.25095 | 0.11068 | 1.82302 | 3.07252 | 1.97156 | 0.000677 | 0.000677 | 0.000677 |
| 8 | 3768.81 | 1.60883 | 2.07596 | 1.78287 | 0.3466 | 0.43 | 0.32254 | 0.13791 | 1.79461 | 2.20762 | 1.91972 | 0.000512 | 0.000512 | 0.000512 |
| 9 | 4235.63 | 1.58419 | 1.9253 | 1.70042 | 0.33889 | 0.52059 | 0.33943 | 0.15506 | 1.82144 | 1.97694 | 1.9375 | 0.000347 | 0.000347 | 0.000347 |
| 10 | 4700.42 | 1.47985 | 1.79475 | 1.60043 | 0.39605 | 0.47824 | 0.37735 | 0.17992 | 1.77868 | 2.01022 | 1.8634 | 0.000182 | 0.000182 | 0.000182 |

# 12. Use Cases

|  |  |
| --- | --- |
| **Use Case** | **Description** |
| **Smart Pet Monitors** | Automatically detect pet activity and alert owners. |
| **Home Automation** | Trigger events (e.g., open pet doors) when a pet is detected. |
| **Veterinary Clinics** | Monitor movement of animals in clinic rooms. |
| **Security Systems** | Differentiate between pets and intruders in surveillance. |
| **Interactive Pet Toys** | Detect presence of a pet to activate toys or treat dispensers. |
| **Animal Rescue** | Deploy drones or cameras with detection to locate lost pets. |

# 13. Conclusion

This project demonstrates the potential of YOLOv8 for custom object detection on real-world pet videos. Despite using a relatively small dataset and training for only 10 epochs, the model achieved moderate detection performance with a precision of 39.8%, recall of 48.2%, and an [mAP@0.5](mailto:mAP@0.5) of 37.7%. While the results are promising, they indicate room for improvement through extended training, more annotated data, and potential hyperparameter tuning. This project lays the groundwork for further development in pet detection and similar real-time object detection tasks.

# 14. Future Enhancements

* Increase epochs and fine-tune hyperparameters
* Add more diverse data (multiple environments, breeds, angles)
* Apply object tracking (e.g., SORT or DeepSORT)
* Deploy on mobile app using TensorFlow Lite or ONNX
* Add audio-based pet activity detection
* Include more pet classes (e.g., birds, rabbits)