Project Report: Animal Herd Detection System

1. Introduction

The Animal Herd Detection System is a computer vision-based application designed to detect and localize animals in an image or video. It leverages YOLOv4 (You Only Look Once) deep learning model for real-time object detection. The system can be useful for wildlife conservation, preventing animal-human conflicts, and tracking animal movements in forests or agricultural lands.

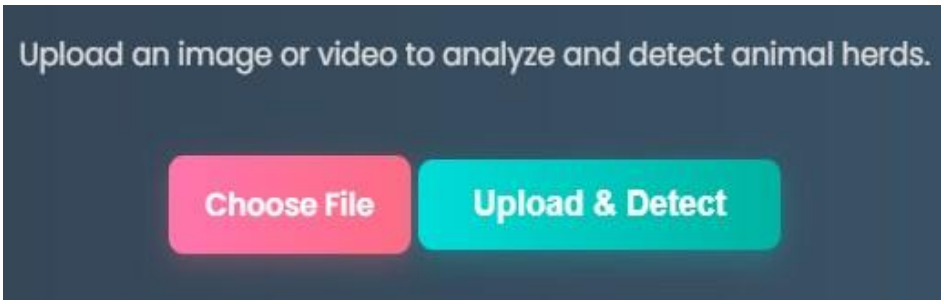
2. Objectives

- Detect and classify animals in an uploaded image.
- Display detected animals with bounding boxes.
- Provide GPS coordinates for detected animal locations.
- Create an alert system to notify concerned authorities.

3. System Architecture

The Animal Herd Detection System follows a modular design:

- **User Interface (UI**): Frontend with an upload option.



- **Backend Processing:** Flask-based API to process uploaded images.

```
app = Flask(__name__)

UPLOAD_FOLDER = 'uploads'
RESULT_FOLDER = 'results'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
os.makedirs(RESULT_FOLDER, exist_ok=True)

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['RESULT_FOLDER'] = RESULT_FOLDER

# Load YOLO Model
try:
    net = cv2.dnn.readNet("C:\\Users\\MY PC\\Desktop\\Animal detection\\project\\yolov4.weights",
                          "C:\\Users\\MY PC\\Desktop\\Animal detection\\project\\yolov4.cfg")

    net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)  # Use OpenCV backend
    net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)  # Use CPU (or CUDA if available)
except cv2.error as e:
    print("Error loading YOLO model:", e)
    exit(1)
```

- **Object Detection Model:** YOLOv4 for animal detection.

```
try:
    net = cv2.dnn.readNet("C:\\Users\\MY PC\\Desktop\\Animal detection\\project\\yolov4.weights",
                          "C:\\Users\\MY PC\\Desktop\\Animal detection\\project\\yolov4.cfg")

    net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)  # Use OpenCV backend
    net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)  # Use CPU (or CUDA if available)
except cv2.error as e:
    print("Error loading YOLO model:", e)
    exit(1)
```

- **Result Visualization**: Annotated images and mapping on Google Maps.

4. Technology Stack

Software & Tools Used:

- Python (Flask, OpenCV, NumPy)
- Deep Learning Framework: OpenCV DNN Module
- Frontend: HTML, CSS, JavaScript
- Mapping Tool: Google Maps API
- Deployment: Flask-based Web Application

5. Working Mechanism (Step-by-Step)

Step 1: Image Upload

- The user selects and uploads an image file via the web interface.
- The system ensures that only valid file formats (JPG,JPEG, PNG) are accepted.

```
ALLOWED_EXTENSIONS = {'jpg', 'jpeg', 'png'}

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
```

Step 2: Preprocessing the Image

- The uploaded image is read using OpenCV (cv2.imread()).
- It is resized to a standard dimension (416x416) for YOLOv4 model processing.
- The image is converted into a blob for better object detection accuracy.

Step 3: Object Detection using YOLOv4

- The YOLOv4 model (yolov4.weights, yolov4.cfg) is loaded.
- The image is passed through the network using net.forward(out_layers).
- The model outputs bounding boxes, confidence scores, and class IDs.
- Animals detected with confidence > 50% are retained.

```
try:
    net = cv2.dnn.readNet("C:\\Users\\MY PC\\Desktop\\Animal detection\\project\\yolov4.weights",
                          "C:\\Users\\MY PC\\Desktop\\Animal detection\\project\\yolov4.cfg")

    net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)  # Use OpenCV backend
    net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)  # Use CPU (or CUDA if available)
except cv2.error as e:
    print("Error loading YOLO model:", e)
    exit(1)
```

Step 4: Post-Processing & Labeling

- Bounding boxes are drawn around detected animals.
- The detected animals confidence scores are displayed.
- The processed image is saved in the results directory.

```
classes = []
coco_path = r"C:\Users\MY PC\Desktop\Animal detection\project\coco.names"
if os.path.exists(coco_path):
    with open(coco_path, "r") as f:
        classes = [line.strip() for line in f.readlines()]
else:
    print("Error: coco.names file not found.")
    exit(1)
```

Step 5: Mapping Animal Locations

- If the image contains GPS metadata, coordinates are extracted.
- Otherwise, approximate locations are provided.
- The detected animals are marked on Google Maps.

Step 6: Displaying Results

- The processed image is displayed in the result.html page.
- If an alert system is integrated, notifications are sent to authorities.

6. Results & Evaluation

- Accuracy: The YOLOv4 model has high accuracy for detecting animals in clear images.
- Processing Speed: Real-time detection capability with minimal delay.
- Limitations: May struggle in poor lighting or occluded scenes.

7. Future Enhancements

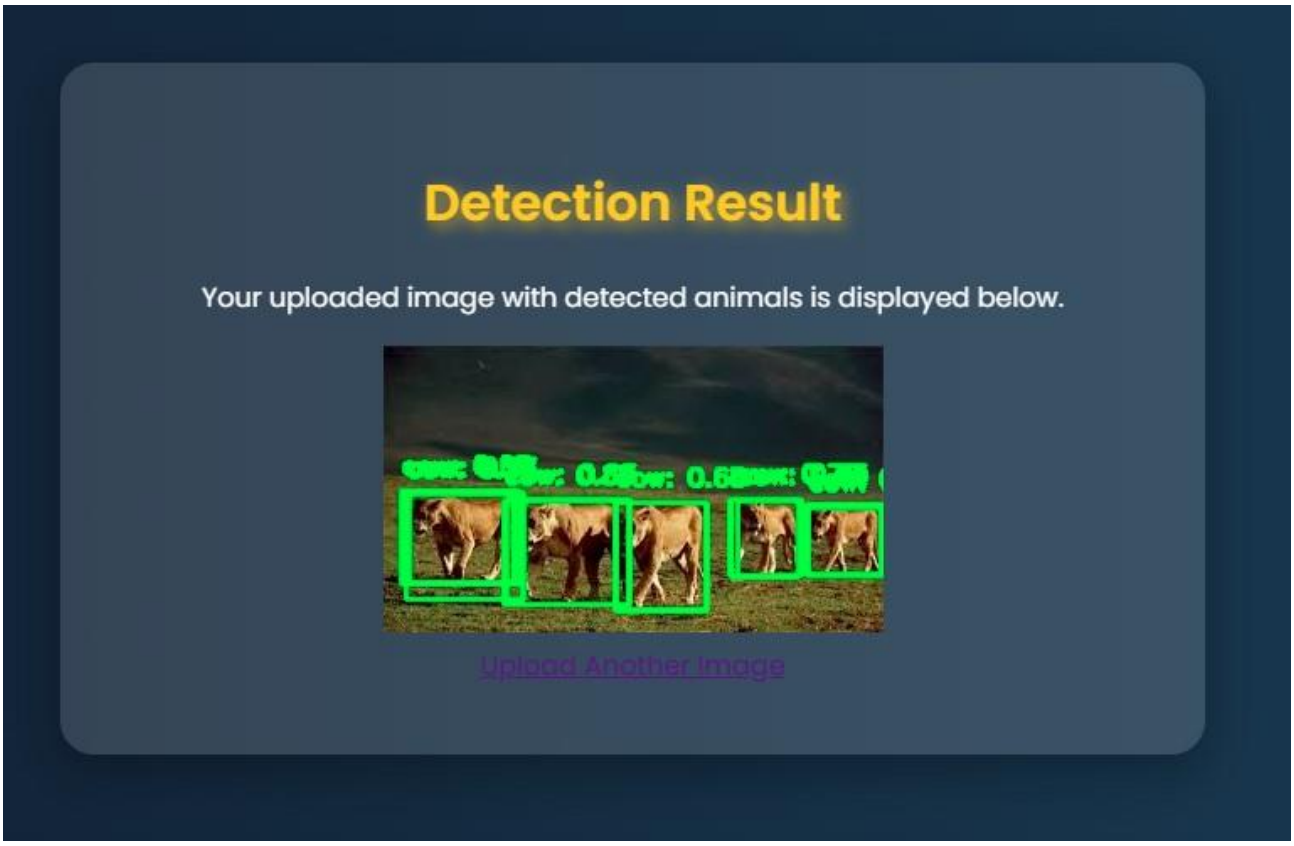
- Enhance model accuracy using YOLOv8 or custom-trained datasets.
- Add live video streaming support for real-time monitoring.
- Integrate drone-based tracking for larger areas.
- Develop a mobile application for on-the-go detection.

8. Conclusion

The Animal Herd Detection System effectively identifies and tracks animals using deep learning and computer vision. With further improvements, it can play a significant role in wildlife conservation, agriculture, and human-wildlife conflict mitigation.

ORIGNAL IMAGE

OUTPUT


Detection Result
Your uploaded image with detected animals is displayed below.
Upload Another Image