

ANIMAL DETECTION IN AGRICULTURAL LAND WITH ALERT

Submitted in partial fulfilment of the required for the award of

Masters of Computer Science

By

MUGILAN V

REGISTER NO (42734011)



**DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF SCIENCE AND HUMANITIES**

**SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY**

(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A++" by NAAC

JEPPIAR NAGAR, RAJIV GANDHI SALAI, CHENNAI 600 119

APRIL 2024

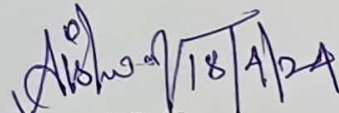


**SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**
Accredited with Grade "A++" by NAAC

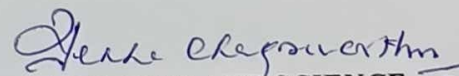
DEPARTMENT OF COMPUTER SCIENCE

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **MUGILAN V (42734011)** who carried out the project entitled "**ANIMAL DETECTION IN AGRICULTURAL LAND WITH ALERT**" under our supervision from January 2024 to April 2024.

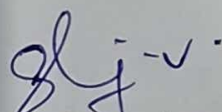

Internal Guide

Ms. AISHWARYA B, M.C.A


DEAN OF ARTS AND SCIENCE

Dr. REKHA CHAKRAVARTHI M.E., Ph.D.

Submitted for viva voce examination held on 20/04/2024

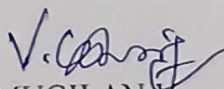

Internal Examiner


External Examiner

DECLARATION

I am, **MUGILAN V (42734011)** hereby declare that the project report entitled “ANIMAL DETECTION IN AGRICULTURAL LAND WITH ALERT” **done by** me under the guidance of **Ms. AISHWARYA B, M.C.A** Department of Computer Science at **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY, jeppiaar Nagar, Rajiv Gandhi Salai, Chennai-600119** is submitted in partial fulfilment of the requirements for the award of Master of Science degree in Computer Science .

DATE: 20/04/2024
PLACE: Chennai



MUGILAN V

Signature of the Candidate

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. REKHA CHAKRAVARTHI M.E., Ph.D. Dean of Arts and Science** for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my project guide **Ms.AISHWARYA B, M.C.A., Assistant Professor** Department of Computer Science for their valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff member of the Department of Computer Science who were helpful in many ways for the completion of the project

MUGILAN V

ABSTRACT

Animal assaults that cause crop damage are one of the main factors lowering crop productivity. One of the most vexing human-wildlife conflicts nowadays is crop raiding, which is caused by the development of cultivated land into formerly wildlife habitat. India's farmers are seriously threatened by pests, natural disasters, and harm from animals, which leads to reduced harvests. Farmers use ineffective traditional methods, and employing guards to monitor crops and deter wild animals is not practical. Since animal and human safety is equally important, it's critical to both shield crops from animal damage and safely divert animals away from crops. Therefore, in this proposed system employ deep learning to detect animals visiting our farm by applying the deep neural network idea, a branch of computer vision, in order to overcome the aforementioned issues and achieve our goal. In this project, a camera that records the surroundings all day long will be used to periodically monitor the entire farm. We can identify when animals are entering with the aid of a deep learning model, and then we play the proper sounds to eject the animal. The model was created using a variety of convolutional neural network libraries and concepts, which are described in this study.

TABLE OF CONTENT

CHAPTER NO	TITLE	PAGE NO
1.INTRODUCTION		
1.1	Farmer Land Securities	1
1.2	Animal Detection System	
1.3	Technology Implications	
1.4	Existing System	
1.5	Proposed System	
2.LITERATURE SURVEY		
2.1	Survey Review	2
3.SYSTEM DESIGN		
3.1	UML Diagrams	5
4.SYSTEM ARCHITECTURE		
4.1	Architecture Details	14
5.SYSTEM IMPLEMENTATIONS		
5.1	Data Used	19
5.2	Module Discription	19
6.CODING AND SCREENSHOT		
6.1	Sample Coding	22
6.2	Sample Screenshot	22
7.CONCLUSION AND FUTURE WORK		
7.1	Conclusion	
7.2	Future Enhancement	
8.REFERENCES		

LIST OF FIGURES

LIST OF FIGURES	NAME OF FIGURES	PAGE NO
3.SYSTEM DESIGN		
3.1	Use Case Diagram	
3.2	Sequence Diagram	
3.3	Activity Diagram	
3.4	Conponent Diagram	
4.SYSTEM ARCHITECTURE		
4.1	System Architecture	

1.INTRODUCTION

1.1 FARMER LAND SECURITIES

Safeguarding farming land from the encroachment of large animals, particularly ungulates, stands as a critical imperative within agricultural management. The deployment of animal detection and warning systems emerges as a fundamental strategy in mitigating the associated risks by furnishing early identification and alert mechanisms. This introduction endeavors to delve into the technical underpinnings of these systems, elucidating their operational principles while emphasizing their pivotal role in protecting agricultural assets.

1.2 ANIMAL DETECTION SYSTEM

Central to animal detection systems are cutting-edge sensor technologies, notably infrared sensors, engineered to discern the presence of warm-bodied animals against the cooler environmental backdrop. These sensors function by detecting fluctuations in infrared radiation emitted by animals as they traverse within the sensor's field of vision. The detection process revolves around the sensor's analysis of shifts in ambient temperature, triggered when an animal crosses into the sensor's detection zone, subsequently prompting activation of the alert mechanism.

The efficacy of animal detection systems pivots on their capability to precisely discriminate between genuine animal presence and environmental stimuli that may incite false alarms. To achieve this, intricate algorithms are employed to meticulously parse sensor data, distinguishing between authentic threats and benign disturbances. Considerations such as the surface characteristics of temperature objects and the emission patterns of infrared radiation are carefully weighed to curtail false positives and fortify the system's reliability.

1.3 TECHNOLOGY IMPLICATIONS

Beyond technological nuances, the successful implementation of animal detection systems necessitates holistic planning and seamless integration into regional and national agricultural policies. Collaborative endeavors between policymakers, researchers, and agricultural stakeholders prove indispensable in pinpointing high-risk areas, prioritizing deployment strategies,

and judiciously allocating resources. Moreover, continuous monitoring and assessment play a pivotal role in gauging system performance and pinpointing avenues for enhancement.

This project endeavors to furnish a nuanced exploration of animal detection and warning systems, shedding light on their technical intricacies and pragmatic applications within agricultural protection. By augmenting our comprehension of these systems and navigating their operational challenges adeptly, we aspire to steer policy discourse and foster the cultivation of more resilient and sustainable agricultural practices.

1.4 EXISTING SYSTEM

The existing system for addressing animal intrusion in farming areas underscores the pressing need to mitigate the significant losses incurred in agriculture and ensure the safety of farmers. Traditional methods have proven inadequate in effectively addressing the challenges posed by human-animal conflicts, prompting the exploration of automated solutions for early detection. Recent advancements in deep learning-based methodologies have gained traction for their ability to offer heightened detection capabilities. In this study, the YOLOv5 method emerges as a promising approach for detecting four categories of animals commonly associated with farming intrusion areas.

YOLOv5 harnesses the power of a Cross Stage Partial (CSP) network as its backbone, enabling robust feature extraction from input images. This architecture is instrumental in facilitating high-accuracy detection, crucial for effectively identifying instances of animal intrusion. Through the utilization of YOLOv5, the system aims to significantly improve detection accuracy, as evidenced by a remarkable enhancement of nearly 94% in mean Average Precision (mAP).

1.5 PROPOSED SYSTEM

The proposed system seeks to tackle the urgent issue of animal intrusion in agricultural areas by harnessing the capabilities of Convolutional Neural Networks (CNNs) for automated detection. Given the vulnerability of farming lands to animal attacks, there is an acute need for early warning mechanisms that can promptly detect and notify farmers of potential threats. The

system initiates with the collection of images or video footage from surveillance cameras, followed by the conversion of frames to enable efficient image processing. Through the utilization of CNNs, the system extracts pertinent features from these images, enabling the identification of potential animal intrusions. Training the model on a diverse dataset of labeled images enhances its ability to accurately discern between animals and other objects. Upon detecting an intrusion, the system triggers timely alerts, such as auditory signals or notifications, to apprise farmers in real-time. Further classification utilizing CNNs assists in determining the species of the detected animal and evaluating the associated risk level. By undergoing rigorous testing and refinement, the proposed system endeavors to furnish farmers with a dependable and effective tool for safeguarding their crops and ensuring their safety amidst the challenges posed by animal threats.

2.LITERATURE SURVEY

2.1 SURVEY REVIEW

In [1] The challenge of localizing and tracking marine animals, such as sharks, fish, or lobsters, is of paramount importance in ecological studies aimed at understanding species behavior and environmental effects. Traditionally, observations of aquatic species were limited to predictable occurrences, but active acoustic telemetry offers a more precise approach. By deploying tags into animals and using moored receivers, researchers can evaluate their locations and movement patterns. This quantification of factors influencing movement is crucial for predicting population responses to environmental changes. Additionally, studying behavioral patterns within spatially-constrained areas provides insights into species-habitat interactions, aiding conservation efforts. However, underwater localization faces challenges such as under-ranked localization, where insufficient receiver coverage leads to localization ambiguities. Proposed solutions involve probabilistic approaches to resolve such ambiguities. Acoustic telemetry primarily relies on time-difference-of-arrival (TDoA) techniques for localization due to their improved accuracy, but under-ranked localization arises due to factors like low tag transmission power, high ambient noise levels, and complex underwater acoustic channels. While increasing the number of receiving elements could mitigate these issues, budget constraints and logistical challenges often limit ecological survey

In [2] The escalating concern over animal attacks among rural populations and forestry workers necessitates effective tracking mechanisms for wild animals. To achieve this, surveillance cameras and drones are commonly utilized, yet a proficient model is essential for detecting animal types, monitoring their movement, and providing location information. Swift alert systems are crucial for ensuring the safety of individuals and forestry personnel. While computer vision and machine learning techniques are prevalent for animal detection, their complexity and cost often hinder satisfactory results. This paper introduces a Hybrid Visual Geometry Group (VGG)-19 combined with Bidirectional Long Short-Term Memory (Bi-LSTM) network for animal detection and alert generation based on their behavior. Alerts are promptly relayed to local forest offices via Short Message Service (SMS) for immediate response. The proposed model demonstrates substantial enhancements in performance, boasting an average classification accuracy of 98%, a mean Average Precision (mAP) of 77.2%, and a Frame Per Second (FPS) of 170. Comprehensive testing using 40,000 images from three diverse benchmark datasets, encompassing 25 classes, yielded mean accuracy and precision exceeding 98%. This model emerges as a dependable solution for furnishing accurate animal-related information and safeguarding human lives.

[3] Conserving wildlife is crucial amidst rapid urbanization and infrastructure expansion, especially in desert and semi-desert regions where biodiversity preservation is vital. The establishment of the Dubai Desert Conservation Reserve (DDCR) in 2000 by the Dubai government was a significant step towards protecting native flora and fauna in the Dubai Desert. Key species in the United Arab Emirates (UAE) desert ecosystem include Oryxes, Camels, and Gazelles. To bolster wildlife surveillance in the DDCR, drone-based monitoring has gained traction. However, existing methods for analyzing drone-captured images rely heavily on manual inspection by human observers. While manual and basic automated techniques, as well as traditional machine learning methods, have been effective for ecological image analysis using RGB and multi-spectral images, recent advancements in deep learning offer promising avenues for animal detection in drone footage, camera trap analysis, and remote sensing applications. Drones offer advantages such as low-noise image capture, extensive coverage, and real-time, high-resolution data collection at a reasonable cost. Nevertheless, manual inspection of drone imagery presents challenges, spurring research into computer-based object detection and classification algorithms for automated analysis. Deep Neural Networks (DNNs) have emerged as powerful tools for object detection under various environmental conditions, but their deployment requires domain

expertise, network optimization, and sufficient training data. This paper leverages a comprehensive dataset of drone imagery from the DDCR, Dubai drone-team, to train a Deep Neural Network based on YOLOv5, a popular CNN architecture known for its effectiveness in one-stage object detection. The interdisciplinary approach aims to optimize network parameters, ensure accurate data labeling, and ultimately improve wildlife monitoring and conservation efforts in desert environments.

One [4] Animal recognition, an evolving field within computer vision, employs sophisticated algorithms to analyze images or videos and classify animals, primarily utilizing deep learning techniques. This domain offers diverse societal benefits, notably aiding wildlife conservation by facilitating the monitoring and protection of endangered species. Furthermore, it serves to raise public awareness and enables the observation of population dynamics within animal communities over time. Accurate animal identification also provides valuable insights into the complex interplay between wildlife populations and their ecosystems, enriching our understanding of environmental processes. Researchers have developed various advanced algorithms and methodologies to study animal behavior, not only advancing scientific knowledge but also serving practical purposes. These efforts function as early warning systems, alerting humans to potential encounters with hazardous wildlife and enabling proactive measures for human safety. Our project focuses on two key phases: animal identification and alerting systems. In the first phase, our emphasis is on accurately identifying four distinct wild animals: Lions, Tigers, Elephants, and Boars. We achieve this through analyzing both static images and live video footage, capturing these animals' presence in their natural habitats. Subsequently, we implement an alerting system designed to promptly notify relevant authorities or individuals of animal intrusions in urban areas. This system employs auditory signals triggered when wild animals are detected near roadways or areas inhabited by humans. Our research contributes to the broader field of animal recognition and conservation, aligning with the goal of enhancing wildlife monitoring and safeguarding human welfare in regions prone to wildlife-human conflicts. Through a thorough examination of existing literature and methodologies, we aim to offer novel insights and advancements to this crucial domain.

In [5] Wireless sensor networks (WSNs) have been instrumental in monitoring endangered species, yet they face challenges in environments with limited access, such as polar regions. Traditional monitoring methods struggle to effectively cover large, inaccessible areas, making remote monitoring inefficient. This paper suggests characterizing animal trajectories using the random walk model to determine optimal detection range and node placement for ensuring target detection probability. Animals can be detected by static sensors on land, mobile sensors attached to animals, or land sensors detecting movement, sound, or temperature changes. However, due to animals' natural movements, they may fall outside sensor coverage. To conserve energy, nodes alternate between active and inactive states, extending system lifespan but necessitating a balance between energy consumption and reporting efficiency. Consequently, careful network design is crucial to calculate detection probability and system longevity. A mathematical model based on Markov chains is proposed and validated through numerical simulations, indicating that animal trajectories can be accurately modeled using specific distributions. In polar regions, conventional WSN setups with multiple nodes clustered together are impractical. Instead, this paper proposes a simplified model employing single sensors strategically placed in isolated locations, where direct communication between nodes is unfeasible. This novel approach streamlines network design and analysis, enabling efficient performance evaluation of the entire multi-sensor network.

One of the first [6] Animal intrusion poses a significant threat to agricultural activities, leading to substantial losses in crop production and jeopardizing the safety of farmers. Addressing this challenge has become a global priority, necessitating effective solutions to mitigate human-animal conflicts. Early detection of animal intrusion through automated methods is crucial in this regard. Recent advancements in deep learning have emerged as promising tools for tackling these issues, offering high detection accuracy. In this study, we propose the utilization of the YOLOv5 method for detecting four common categories of animals frequently involved in farm intrusion incidents. YOLOv5 leverages the cross stage partial network (CSP) as its backbone, enabling the extraction of valuable features from input images. This network architecture enhances the model's ability to accurately identify animals in farming area. Our implementation of the YOLOv5 method demonstrates its effectiveness in detecting animal intrusion, yielding a significant improvement in detection accuracy, with nearly 94% mean Average Precision (mAP). The results underscore the efficacy of the proposed approach in addressing human-animal conflict scenarios. Moreover, our models achieve state-of-the-art performance levels, showcasing their capability to meet the

demanding requirements of these challenges. Overall, the findings of this study highlight the potential of deep learning-based methods, particularly YOLOv5, in enhancing the early detection of animal intrusion in farming areas. By leveraging advanced technologies, we can significantly reduce the impact of human-wildlife conflicts on agricultural productivity and ensure the safety and livelihoods of farmers worldwide.

In [7] The encroachment of wild animals into populated areas presents a significant challenge, posing threats to human safety and causing damage to resources. These animals often stray into urban or agricultural regions in search of food, leading to crop destruction, harm to livestock, and occasionally, even human fatalities. To mitigate this issue, this paper proposes a reliable and efficient solution leveraging computer vision technology for automatic wild animal detection. The proposed method integrates the YOLO object detection model, renowned for its robust performance in identifying objects within images, to detect the presence of wild animals. By analyzing images captured through surveillance cameras or other sources, the model can accurately identify and localize wild animals within the vicinity. Additionally, an audio model trained using deep learning algorithms is employed to detect specific sounds emitted by wild animals, further enhancing the detection capabilities. Upon detection of a wild animal incursion, the system promptly triggers an alert message, notifying relevant authorities to take necessary action. This proactive approach enables timely intervention to prevent potential harm to human lives and mitigate damage to property and resources. By combining advanced computer vision techniques with deep learning-based audio analysis, the proposed method offers a comprehensive and effective solution for addressing the challenge of wild animal intrusion into populated areas. This approach not only enhances public safety but also facilitates the conservation of wildlife by minimizing conflicts with human populations.

In[8] The aspiration for large-scale automatic detection of individual animal phenotypes has long been a priority in animal breeding practices. Recent advancements in Artificial Intelligence (AI) technologies, alongside the availability of tools such as RFIDs, Wireless Sensor Networks, and Camera Systems, have made it feasible to automate the process of animal phenotype detection. This automated process typically involves several phases, including data collection, data fusion, data storage, development of AI models for data analysis, and serving of analysis results. To facilitate the automated phenotype detection process, we have developed a comprehensive data

analytics platform architecture. This architecture encompasses various approaches tailored to different phases of the automated phenotype detection process. These approaches are implemented and executed on the Dutch national High-Performance Computing (HPC) infrastructure, ensuring scalability and efficiency in processing large volumes of data. Moreover, the manner in which data analysis results are presented to end-users plays a crucial role in the success of a data analytics platform. To address this aspect, we propose a scalable data visualization pipeline. This pipeline enables animal behavior scientists to derive quick summaries and insights about animal behavior on a farm, enhancing their ability to make informed decisions and optimize breeding strategies. By integrating advanced technologies and methodologies, our platform offers a robust framework for automating the detection of individual animal phenotypes at scale. Through seamless data collection, analysis, and visualization, our platform empowers researchers and breeders to enhance their understanding of animal behavior and make data-driven decisions to improve breeding outcomes.

In [9] The escalating conflicts between humans and animals in regions adjacent to forested areas pose significant threats, including resource depletion, loss of human and animal lives, and property damage. Addressing these challenges has become imperative, prompting the exploration of advanced algorithms in object detection and visual object tracking to develop effective monitoring systems. These systems aim to reduce casualties by issuing timely warnings while safeguarding animal welfare. In this paper, we propose a tracking-by-detection approach to address the aforementioned issues. Our methodology involves utilizing YOLOv4, a state-of-the-art object detection algorithm, trained on a custom image dataset comprising nine specific classes of animals. This trained model serves as a descriptor within the DeepSORT tracker framework, enabling the tracking of animal movements across video frames. The YOLOv4 detector exhibits an impressive mean Average Precision (mAP) accuracy of 98%, ensuring reliable detection of animals in various environmental conditions. Furthermore, the performance of the tracker is evaluated using the AnimalTrack benchmark dataset, providing comprehensive insights into its efficacy in real-world scenarios. By integrating cutting-edge object detection and visual object tracking techniques, our approach offers a holistic solution to mitigate human-animal conflicts. The utilization of advanced algorithms not only enhances monitoring capabilities but also prioritizes the preservation of animal life. Through systematic tracking and early warning mechanisms, our methodology contributes to

the establishment of proactive measures aimed at minimizing casualties and fostering harmonious coexistence between humans and wildlife in forest-proximate regions.

[10] The escalating frequency of vehicular collisions with wildlife poses a dual threat to human safety and wildlife conservation endeavors. It's imperative to effectively monitor wild animals to tackle this issue. Cost-efficient methods for observing wildlife behavior are crucial for reducing conflicts between humans and wildlife and advancing conservation efforts. Hence, there's an urgent need for a streamlined system for wild animal detection. Manual identification of diverse animal species presents considerable challenges due to the sheer variety of creatures involved. Leveraging animal detection technologies holds promise in averting animal-vehicle accidents and efficiently tracking wildlife movements. To address this, we propose employing deep learning algorithms to develop an efficient wild animal detection algorithm. Our approach utilizes the depth-wise separable convolution layer, a potent technique amalgamating pointwise and depth-wise convolutions, to construct the detection model. This model integrates zero padding to preserve edge features and standardize the output image size, thereby enhancing the accuracy of wild animal detection. We assessed the efficacy of our proposed algorithm through tests conducted on the IWildCam dataset. Encouraging results emerged, with a detection accuracy intersection over union value of 0.878 and a remarkable 99.6% classification accuracy for identifying wild animal species. By harnessing sophisticated deep learning methodologies, our algorithm presents a robust solution for wild animal detection. The commendable accuracy achieved in both detection and classification tasks underscores the potential of our approach in addressing the challenges associated with human-wildlife conflicts and fostering wildlife conservation efforts.

3.SYSTEM DESIGN

3.1 UML DIAGRAMS

A UML (Unified Modeling Language) diagram is a graphical representation of a system or process using standardized symbols and notation. UML diagrams are widely used in software engineering and other fields to visually depict various aspects of a system, such as its structure, behavior, interactions, and architecture.

There are several types of UML diagrams, each serving a different purpose and focusing on different aspects of the system being modeled. Some common types of UML diagrams include:

Unified Modeling Language (UML) diagram for the proposed automated college timetable scheduling system. Here's an example of a high-level UML class diagram for the system:

3.1.1 Use Case Diagram

In this Use Case Diagram:

Animal Detection System: Represents the main functionality of the system, which includes detecting and classifying animals.

User: Represents users who submit images for animal detection.

Admin: Represents administrators who manage the system and view detection results.

Submit Image: Use case where users submit images to the system for animal detection.

View Detection Results: Use case where users and admins view the results of animal detection.

Manage System: Use case where admins manage the system, including configuring settings, managing users, etc.

This diagram provides a high-level overview of the interactions between users, administrators, and the animal detection system, as well as the main functionalities supported by the system.

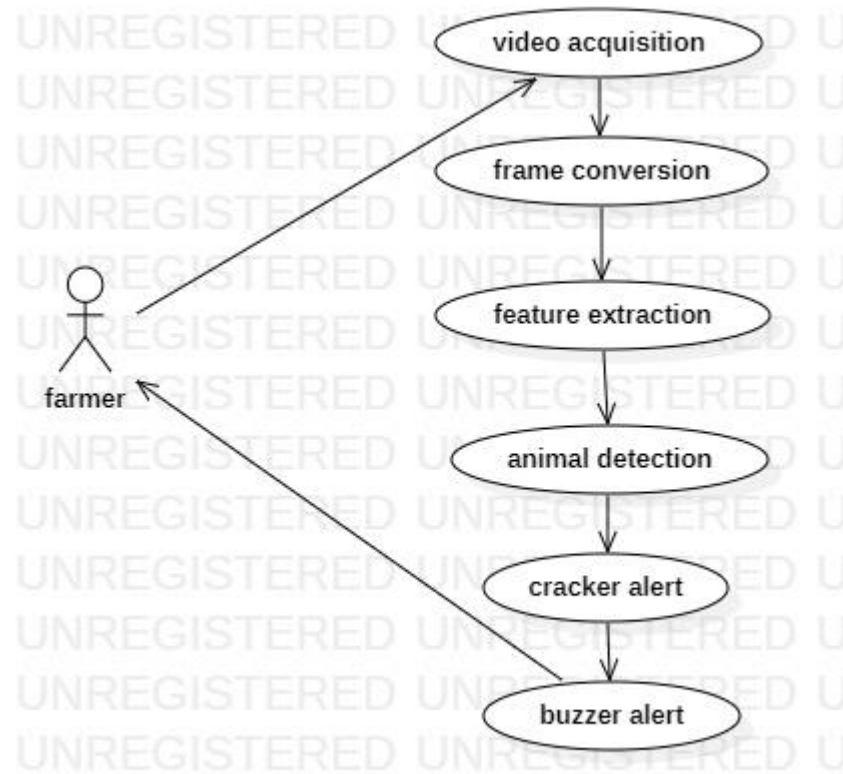


Fig 3.1 Use case diagram

3.1.2 Sequence Diagram

A Sequence Diagram is a type of UML (Unified Modeling Language) diagram that illustrates the interactions between objects or components within a system in a sequential manner. It focuses on the flow of messages or method calls exchanged between these objects over time to accomplish a specific behavior or scenario.

Key components of a sequence diagram include:

Objects/Participants: Represent the entities (objects, components, or actors) involved in the interaction. Each object is typically represented by a box or lifeline vertically aligned along the diagram's left side.

Lifelines: Vertical lines extending downwards from the objects, representing the existence of the objects over time during the interaction.

Messages: Arrows drawn between lifelines, indicating the flow of communication between objects. Messages can be synchronous (e.g., method calls), asynchronous (e.g., signals or events), or replies.

Activation/Execution Occurrences: Represented by narrow rectangles on lifelines, showing the duration during which an object is actively processing a message or performing an operation. Optional Elements: Other optional elements may include conditionals (such as alternative or loop constructs), iterations, and parallel execution to capture more complex scenarios. This sequence diagram illustrates the sequential flow of interactions between the Administrator, the College Timetable System, and the Staff during the process of generating and approving a timetable.

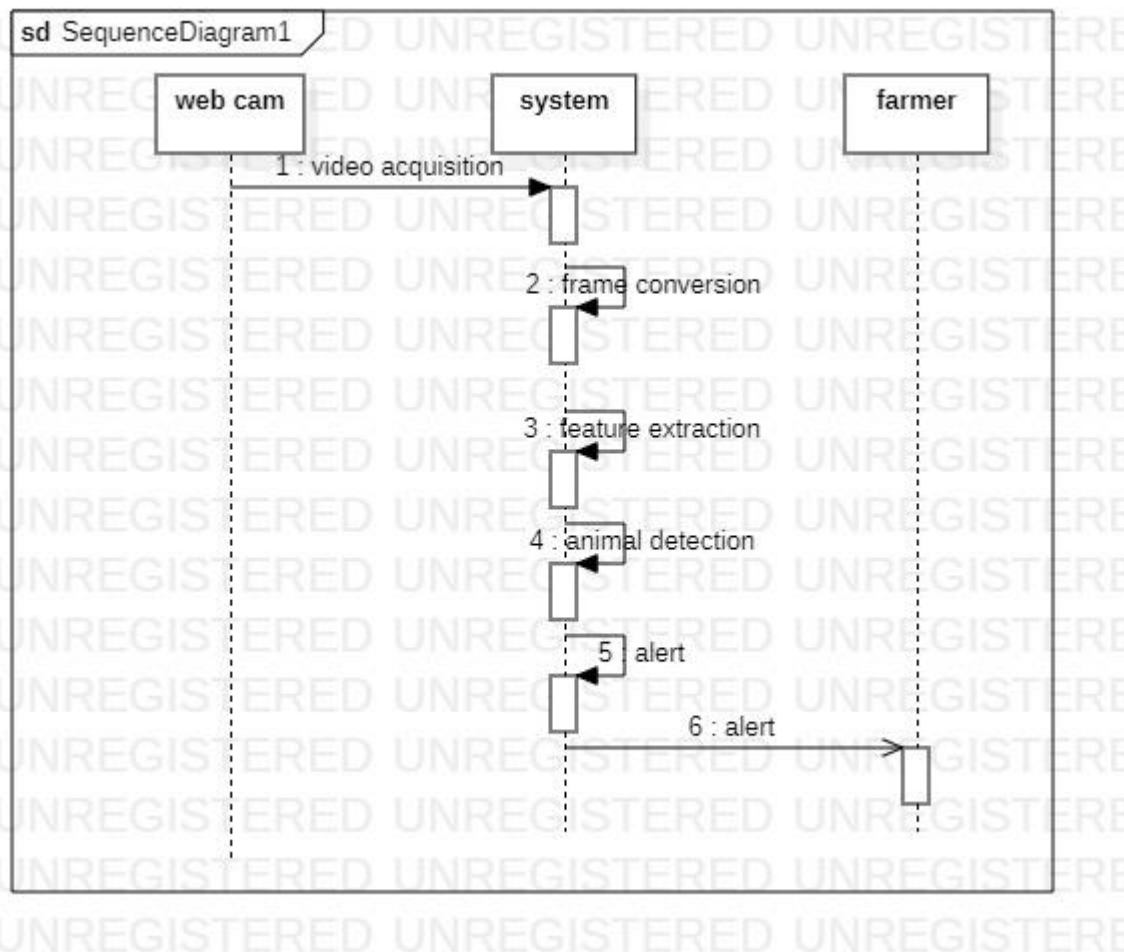


Fig 3.2 Sequence Diagram

3.2.3 Activity Diagram

An Activity Diagram is a type of UML (Unified Modeling Language) diagram that illustrates the flow of activities or actions within a system or a specific business process. It visually represents the sequence of actions, decisions, and transitions from one activity to another, allowing stakeholders to understand the workflow or behavior of a system or process.

Key components of an activity diagram include:

Activities: Represented by rounded rectangles, activities depict specific actions or steps within the system or process. Each activity represents a task that needs to be performed.

Transitions: Arrows connecting activities, indicating the flow of control from one activity to another. Transitions represent the order in which activities are executed, typically triggered by the completion of a preceding activity or by the satisfaction of a condition.

Decision Nodes: Diamond-shaped symbols used to represent decision points or branches in the workflow. Depending on the outcome of a decision, the flow may proceed along different paths.

Merge Nodes: Used to consolidate multiple flow paths back into a single path after a decision or branching point.

Initial Node: Indicates the starting point of the activity diagram. It represents the initiation of the workflow or process.

Final Node: Represents the endpoint of the activity diagram, indicating the completion of the workflow or process.

Fork and Join Nodes: Fork nodes split the flow into multiple parallel paths, while join nodes merge parallel paths back into a single flow.

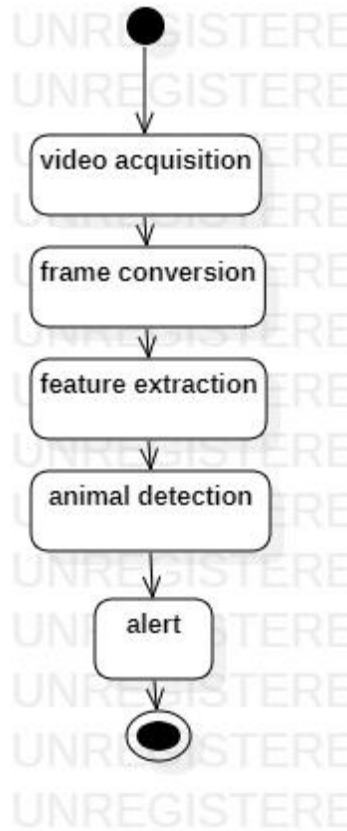


Fig 3.3 Activity Diagram

3.2.4 Package Diagram

A Package Diagram is a type of UML (Unified Modeling Language) diagram that represents the organization and dependencies among packages in a system. In software engineering, a package is a collection of related elements, such as classes, interfaces, and other packages, grouped together to provide a cohesive unit of functionality. Package diagrams provide a high-level overview of the structure and organization of a system, helping to manage complexity and promote modularity.

Key components of a package diagram include:

Packages: Represented by rectangles with folded corners, packages are used to encapsulate and organize related elements within a system. Each package may contain classes, interfaces, other packages, and dependencies.

Package Dependencies: Arrows between packages indicate dependencies between them. A dependency implies that elements in one package depend on elements in another package. Dependencies can be either directed or bidirectional.

Package Relationships: Different types of relationships between packages can be depicted, including:

Association: Indicates a semantic relationship between packages, such as a usage dependency.

Generalization: Represents a specialization/generalization relationship between packages, where one package specializes another.

Import: Signifies that one package imports elements from another package, allowing access to its contents.

Merge: Used to combine the contents of two or more packages into a single package.

Package Visibility: Packages may have different levels of visibility, such as public, private, or protected, which determine the accessibility of their contents to other packages.

This package diagram provides a high-level overview of the package structure of the automated college timetable scheduling system, organizing classes and components into logical groupings based on their functionalities.

3.2.5 Component Diagram

Below is a simplified component diagram representing the components and their interactions in the automated college timetable scheduling system:

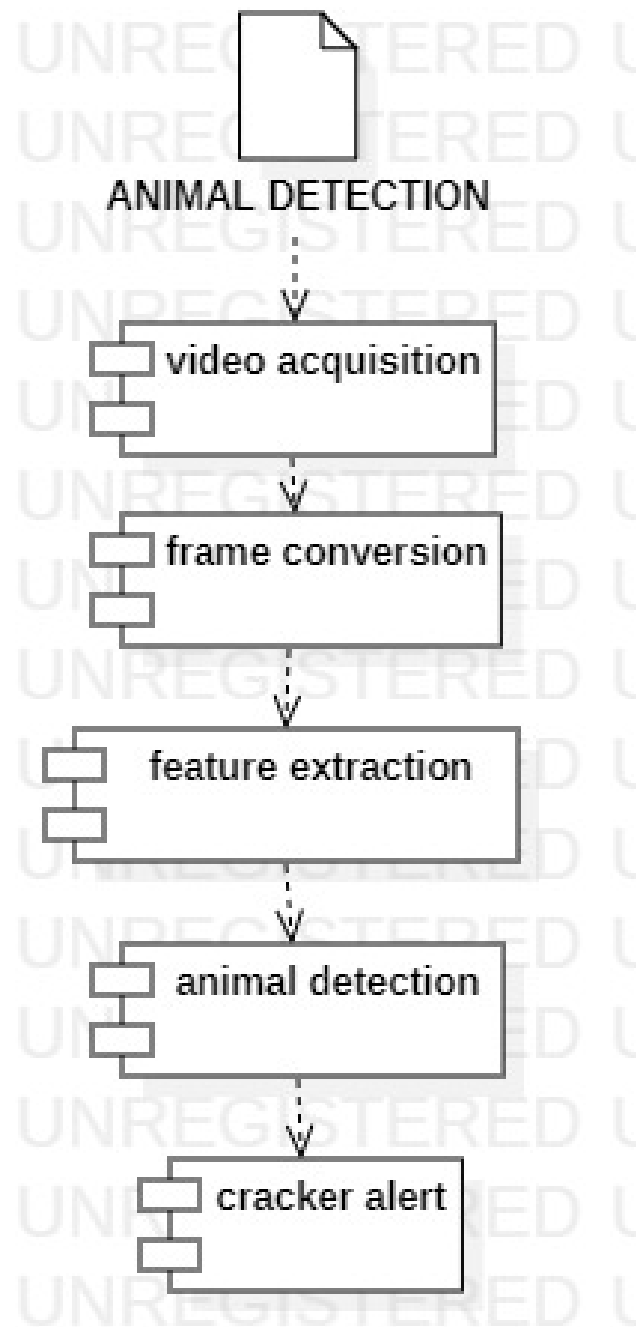


Fig 3.4 Component diagram

4.SYSTEM ARCHITECTURE

4.1 ARCHITECTURE DETAILS

The system architecture for animal detection typically involves a combination of hardware and software components designed to identify and classify animals based on input data, such as images or sensor readings. Below is a conceptual overview of the system architecture for animal detection:

Sensor Input:

The system may receive input data from various sensors, such as cameras, drones, thermal imaging devices, LiDAR (Light Detection and Ranging), or other environmental sensors. These sensors capture data about the surroundings, including images, videos, infrared radiation, or environmental parameters like temperature and humidity.

Data Preprocessing:

Raw sensor data often requires preprocessing to enhance its quality and extract relevant features. Preprocessing steps may include noise reduction, image stabilization, normalization, resizing, and filtering to prepare the data for further analysis.

Feature Extraction:

Feature extraction involves identifying meaningful patterns or characteristics from the preprocessed data that are relevant for animal detection. Depending on the type of sensor data, feature extraction techniques may include edge detection, texture analysis, color histograms, keypoints extraction, or deep learning-based feature extraction using convolutional neural networks (CNNs).

Classification Algorithm:

The heart of the system is the classification algorithm, which analyzes the extracted features to classify whether animals are present in the input data and, if so, to identify their species

or categories. Classification algorithms can range from traditional machine learning models like more advanced deep learning models like CNNs, Recurrent Neural Networks (RNNs), or Transformer models.

Model Training and Optimization:

For deep learning-based approaches, the classification model needs to be trained on labeled data (supervised learning). Training involves feeding labeled examples into the model, adjusting its parameters iteratively using optimization techniques like gradient descent, and evaluating its performance on validation data. Hyperparameter tuning and model optimization techniques may be applied to improve the model's accuracy and generalization capabilities.

Deployment and Integration:

Once trained and optimized, the classification model is deployed within the system architecture, where it can be integrated with other components. Deployment options may include embedding the model into edge devices for real-time processing, deploying it on cloud-based servers for remote processing, or utilizing specialized hardware accelerators for efficient inference.

Output Visualization and Decision Making:

The system provides output in the form of detections and classifications, indicating the presence of animals and potentially their species or categories. Visualization tools may be used to display the detected animals overlaid on the input data, such as images or maps, facilitating human interpretation and decision-making.

Feedback Loop and Continuous Improvement:

A feedback loop mechanism allows the system to continuously learn and improve over time based on user feedback and new data. This may involve updating the classification model periodically with new labeled data, retraining it to adapt to changing conditions or environments, and incorporating feedback from users to address false positives or negatives.

This system architecture provides a high-level overview of the components and processes involved in animal detection systems. Depending on the specific requirements, constraints, and application scenarios, the architecture may vary in terms of the choice of sensors, algorithms, deployment platforms, and integration with existing systems.

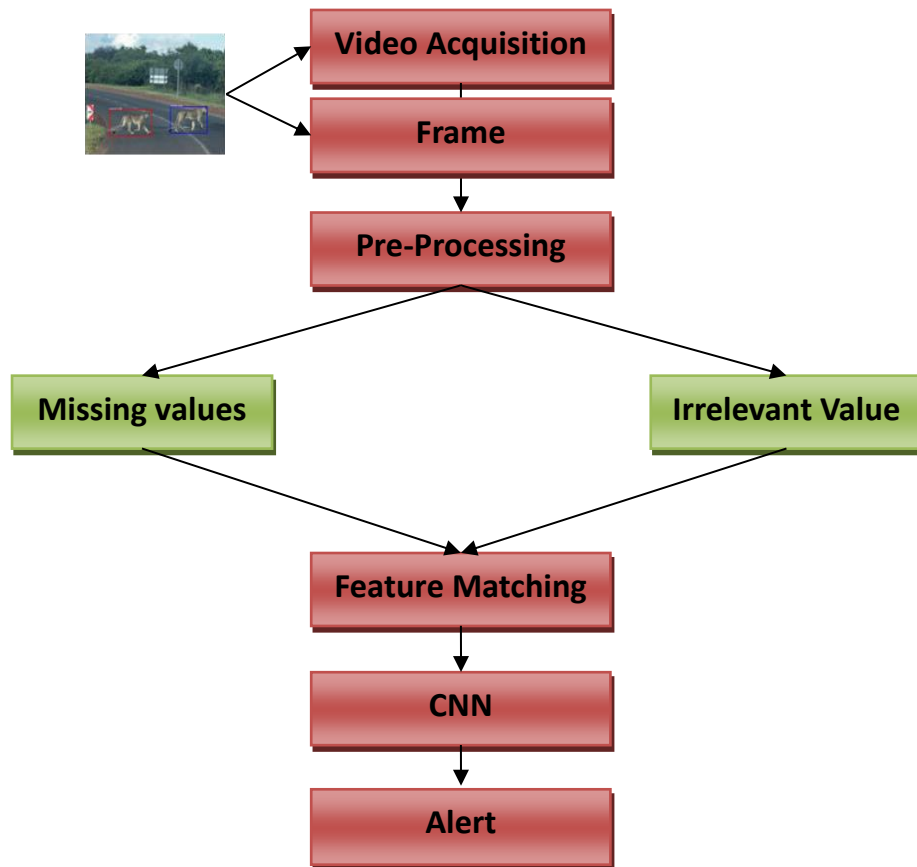


Fig 4.1 System Architecture

5.SYSTEM IMPLEMENTATION

5.1 DATA USED

The dataset gathered from the analysis of the search engine analysis where the complete details of the animals are found out with feature extraction

Source:<https://wordnet.princeton.edu/>

Source: <https://www.kaggle.com/datasets/antoreepjana/animals-detection-images-dataset>

Classes of animals added ->

- Dog
- Cat
- Zebra
- Lion
- Leopard
- Cheetah
- Tiger
- Bear
- Brown Bear
- Bull
- Camel
- Crab
- Chicken
- Centipede
- Cattle
- Duck

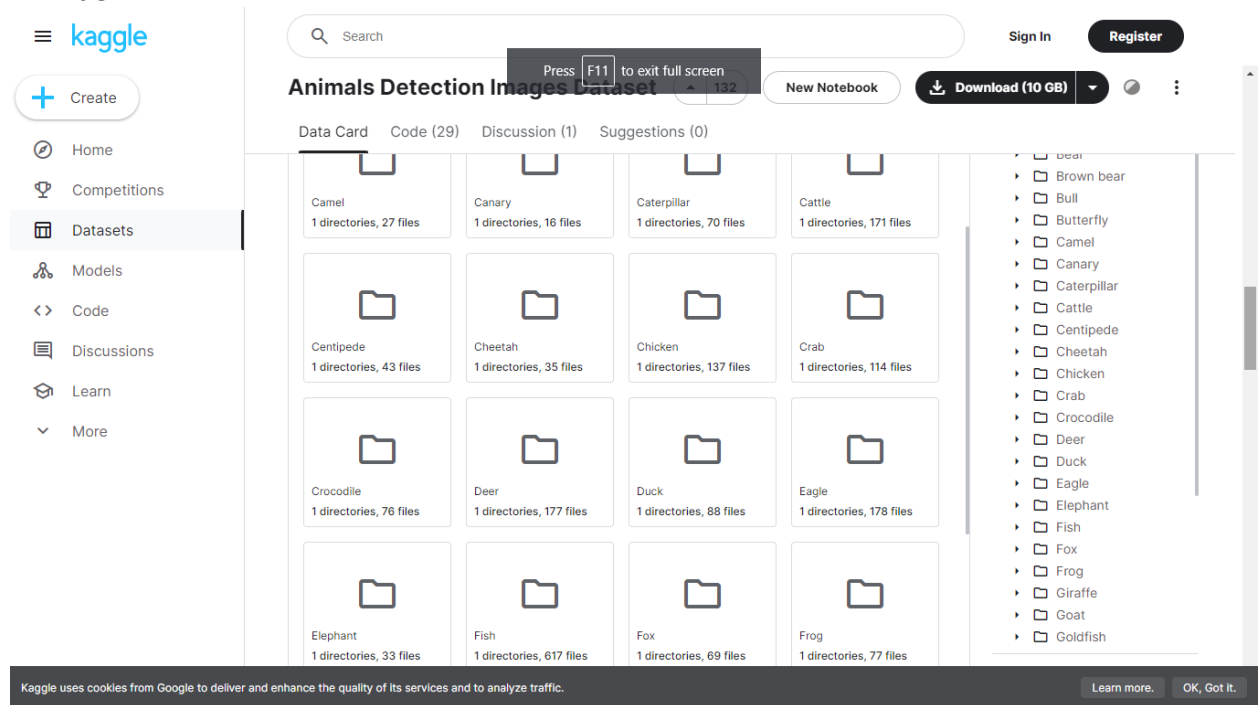


Fig 5.1 Dataset from Kaggle

5.2 MODULE DESCRIPTION

- CCTV surveillance
- Video Processing
- Frame conversion
- Animal Feature extraction
- CNN classification
- Animal Detection
- Crackers sound
- Alert to farmer

5.2.1 CCTV Surveillance

In this module, the proposed system makes a web cam access with a hardware interface in outward of the land owner field. This CCTV surveillance captures the video and makes it as an input

5.2.2 Video Processing

The video acquisition system will be the one where the CCTV surveillance system which will be done with the identification. The video maintenance will be based with higher secured version system. The video processing will be done with the developed streaming and processing features.

5.2.3 Frame Conversion

Frame conversion module will be done with the identification of the number of frames generated with the video. The frames will be converted as an image with the system where the image can be taken for further processing.

5.2.4 Pre- processing

The pre- processing option is the one where the exact calculation of the system can be made with the generation of the given application. The maintenance over the applicative system will reduce the feature pixels maintenance.

5.2.5 Feature Extraction

The feature extraction system will extract the feature point in which the multiple feature like texture, color, morphology, Point Of Interest (POI), Region Of Interest (ROI), etc will be calculated.

5.2.6 CNN Classification

Convolution Neural Network algorithm helps in the detection and classification of the animals in the normal CCTV extraction. The main formulation will be done with the different extraction and the classification over the trained animals is detected.

5.2.7 Alert

In this module, the trained values are limited with an enough feature labeled data alert detection system. The module will get with the alert system when the animal comes

6.CODING AND SCREENSHOT

6.1 SAMPLE CODING

```
import cv2
import numpy as np
import winsound
duration = 1000 # milliseconds
freq = 440 # Hz
# load the class labels from disk
import pyttsx3 as pyttsx3
engine = pyttsx3.init()
animals=['elephant','tiger','lion','monkey','fox']
rows = open("synset_words.txt").read().strip().split("\n")
classes = [r[r.find(" ") + 1:].split(",")[0] for r in rows]

# load our serialized model from disk
```

```

net = cv2.dnn.readNetFromCaffe("bvlc_googlenet.prototxt","bvlc_googlenet.caffemodel")
#####test arun
cap = cv2.VideoCapture(0)
while 1:
    ret, img = cap.read()
    #cv2.imshow("Image", img)
    image = (img)
    resized = cv2.resize(image, (224, 224))
    blob = cv2.dnn.blobFromImage(resized, 1, (224, 224), (104, 117, 123))
    net.setInput(blob)
    preds = net.forward()
    idx = np.argsort(preds[0])[:-1][0]
    res=preds[0][idx] * 100
    if res > 50:
        text = "Label: {}, {:.2f}%".format(classes[idx],preds[0][idx] * 100)
        text1 = "{}".format(classes[idx])
        if text1 in animals:
            cv2.putText(image, text, (5, 25), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0, 0, 255), 2)

            winsound.Beep(freq, duration)
        else:
            cv2.putText(image, text, (5, 25), cv2.FONT_HERSHEY_SIMPLEX,0.7,
(0, 255, 0), 2)

            # engine.say(text1)
            # engine.runAndWait()
    cv2.imshow("Image1",img)
    #print(text1)

    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

```

```
cap.release()
cv2.destroyAllWindows()
```

```
#####
```

```
import cv2
import numpy as np
import winsound
duration = 1000 # milliseconds
freq = 440 # Hz
# load the class labels from disk
import pyttsx3 as pyttsx3
engine = pyttsx3.init()
animals=['elephant','tiger','lion','monkey','fox']
rows = open("synset_words.txt").read().strip().split("\n")
classes = [r[r.find(" ") + 1:].split(",")[0] for r in rows]

# load our serialized model from disk
net = cv2.dnn.readNetFromCaffe("bvlc_googlenet.prototxt","bvlc_googlenet.caffemodel")
#####test arun
cap = cv2.VideoCapture(0)
while 1:
    ret, img = cap.read()
    #cv2.imshow("Image", img)
    image = (img)
    resized = cv2.resize(image, (224, 224))
    blob = cv2.dnn.blobFromImage(resized, 1, (224, 224), (104, 117, 123))
    net.setInput(blob)
```

```

preds = net.forward()
idx = np.argsort(preds[0])[:-1][0]
res=preds[0][idx] * 100
if res > 50:
    text = "Label: {}, {:.2f}%".format(classes[idx],preds[0][idx] * 100)
    text1 = "{}".format(classes[idx])
    if text1 in animals:
        cv2.putText(image, text, (5, 25), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0, 0, 255), 2)
        winsound.Beep(freq, duration)
    else:
        cv2.putText(image, text, (5, 25), cv2.FONT_HERSHEY_SIMPLEX,0.7,
(0, 255, 0), 2)
        # engine.say(text1)
        # engine.runAndWait()
cv2.imshow("Image1",img)
#print(text1)

k = cv2.waitKey(30) & 0xff
if k == 27:
    break
cap.release()
cv2.destroyAllWindows()

```

#####

```

import cv2
import numpy as np

```

```

import winsound
duration = 1000 # milliseconds
freq = 440 # Hz
# load the class labels from disk
import pyttsx3 as pyttsx3
engine = pyttsx3.init()
animals=['elephant','tiger','lion','monkey','fox']
rows = open("synset_words.txt").read().strip().split("\n")
classes = [r[r.find(" ") + 1:].split(",")[0] for r in rows]

# load our serialized model from disk
net = cv2.dnn.readNetFromCaffe("bvlc_googlenet.prototxt","bvlc_googlenet.caffemodel")
#####test arun
cap = cv2.VideoCapture(0)
while 1:
    ret, img = cap.read()
    #cv2.imshow("Image", img)
    image = (img)
    resized = cv2.resize(image, (224, 224))
    blob = cv2.dnn.blobFromImage(resized, 1, (224, 224), (104, 117, 123))
    net.setInput(blob)
    preds = net.forward()
    idx = np.argsort(preds[0])[:-1][0]
    res=preds[0][idx] * 100
    if res > 50:
        text = "Label: {}, {:.2f}%".format(classes[idx],preds[0][idx] * 100)
        text1 = "{}".format(classes[idx])
        if text1 in animals:
            cv2.putText(image, text, (5, 25), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0, 0, 255), 2)

            winsound.Beep(freq, duration)

```

```

        else:
            cv2.putText(image, text, (5, 25), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0, 255, 0), 2)
            # engine.say(text1)
            # engine.runAndWait()
            cv2.imshow("Image1",img)
            #print(text1)

            k = cv2.waitKey(30) & 0xff
            if k == 27:
                break
cap.release()
cv2.destroyAllWindows()

```

```
#####
```

```
"""call using an open ADO connection --> list of table names"""
```

```
from . import adodbapi
```

```

def names(connection_object):
    ado = connection_object.adoConn
    schema = ado.OpenSchema(20) # constant = adSchemaTables

    tables = []
    while not schema.EOF:
        name = adodbapi.getIndexValue(schema.Fields, "TABLE_NAME").Value
        tables.append(name)
        schema.MoveNext()
    del schema
    return tables

```

```
"""call using an open ADO connection --> list of table names"""
```

```
from . import adodbapi
```

```
def names(connection_object):
```

```
    ado = connection_object.adoConn
```

```
    schema = ado.OpenSchema(20) # constant = adSchemaTables
```

```
    tables = []
```

```
    while not schema.EOF:
```

```
        name = adodbapi.getIndexValue(schema.Fields, "TABLE_NAME").Value
```

```
        tables.append(name)
```

```
        schema.MoveNext()
```

```
    del schema
```

```
    return tables
```

```
"""adodbapi - A python DB API 2.0 (PEP 249) interface to Microsoft ADO
```

```
Copyright (C) 2002 Henrik Ekelund, version 2.1 by Vernon Cole
```

```
* http://sourceforge.net/projects/adodbapi
```

```
"""
```

```
import sys
```

```
import time
```

```
from .apibase import apilevel, threadsafety, paramstyle
```

```
from .apibase import (
```

```
    Warning,
```

```
    Error,
```

```
    InterfaceError,
```

```
    DatabaseError,
```

```
    DataError,
```

```

        OperationalError,
        IntegrityError,
    )
from .apibase import (
    InternalError,
    ProgrammingError,
    NotSupportedError,
    FetchFailedError,
)
from .apibase import NUMBER, STRING, BINARY, DATETIME, ROWID

from .adodbapi import connect, Connection, __version__, dateconverter, Cursor


def Binary(aString):
    """This function constructs an object capable of holding a binary (long) string value."""
    return bytes(aString)


def Date(year, month, day):
    "This function constructs an object holding a date value."
    return dateconverter.Date(year, month, day)


def Time(hour, minute, second):
    "This function constructs an object holding a time value."
    return dateconverter.Time(hour, minute, second)


def Timestamp(year, month, day, hour, minute, second):
    "This function constructs an object holding a time stamp value."

```

```
return dateconverter.Timestamp(year, month, day, hour, minute, second)
```

```
def DateFromTicks(ticks):
```

```
    """This function constructs an object holding a date value from the given ticks value
    (number of seconds since the epoch; see the documentation of the standard Python time
    module for details)."""
```

```
    return Date(*time.gmtime(ticks)[:3])
```

```
def TimeFromTicks(ticks):
```

```
    """This function constructs an object holding a time value from the given ticks value
    (number of seconds since the epoch; see the documentation of the standard Python time
    module for details)."""
```

```
    return Time(*time.gmtime(ticks)[3:6])
```

```
def TimestampFromTicks(ticks):
```

```
    """This function constructs an object holding a time stamp value from the given
    ticks value (number of seconds since the epoch;
    see the documentation of the standard Python time module for details)."""
```

```
    return Timestamp(*time.gmtime(ticks)[:6])
```

```
version = "adodbapi v" + __version__
```

```
"""adodbapi - A python DB API 2.0 (PEP 249) interface to Microsoft ADO
```

```
Copyright (C) 2002 Henrik Ekelund, version 2.1 by Vernon Cole
```

```
* http://sourceforge.net/projects/adodbapi
```

```
"""
```

```

import sys
import time

from .apibase import apilevel, threadsafety, paramstyle
from .apibase import (
    Warning,
    Error,
    InterfaceError,
    DatabaseError,
    DataError,
    OperationalError,
    IntegrityError,
)
from .apibase import (
    InternalError,
    ProgrammingError,
    NotSupportedError,
    FetchFailedError,
)
from .apibase import NUMBER, STRING, BINARY, DATETIME, ROWID

from .adodbapi import connect, Connection, __version__, dateconverter, Cursor
def Binary(aString):
    """This function constructs an object capable of holding a binary (long) string value."""
    return bytes(aString)
def Date(year, month, day):
    "This function constructs an object holding a date value."
    return dateconverter.Date(year, month, day)
def Time(hour, minute, second):
    "This function constructs an object holding a time value."
    return dateconverter.Time(hour, minute, second)

```

```

def Timestamp(year, month, day, hour, minute, second):
    "This function constructs an object holding a time stamp value."
    return dateconverter.Timestamp(year, month, day, hour, minute, second)
def DateFromTicks(ticks):
    """This function constructs an object holding a date value from the given ticks value
    (number of seconds since the epoch; see the documentation of the standard Python time
    module for details)."""
    return Date(*time.gmtime(ticks)[:3])
def TimeFromTicks(ticks):
    """This function constructs an object holding a time value from the given ticks value
    (number of seconds since the epoch; see the documentation of the standard Python time
    module for details)."""
    return Time(*time.gmtime(ticks)[3:6])
def TimestampFromTicks(ticks):
    """This function constructs an object holding a time stamp value from the given
    ticks value (number of seconds since the epoch;
    see the documentation of the standard Python time module for details)."""
    return Timestamp(*time.gmtime(ticks)[6:])
version = "adodbapi v" + __version__
"""adodbapi - A python DB API 2.0 (PEP 249) interface to Microsoft ADO
Copyright (C) 2002 Henrik Ekelund, version 2.1 by Vernon Cole
* http://sourceforge.net/projects/adodbapi
"""

import sys
import time
from .apibase import apilevel, threadsafety, paramstyle
from .apibase import (
    Warning,
    Error,
    InterfaceError,
    DatabaseError,

```

```

    DataError,
    OperationalError,
    IntegrityError,
)
from .apibase import (
    InternalError,
    ProgrammingError,
    NotSupportedError,
    FetchFailedError,
)
from .apibase import NUMBER, STRING, BINARY, DATETIME, ROWID
from .adodbapi import connect, Connection, __version__, dateconverter, Cursor
def Binary(aString):
    """This function constructs an object capable of holding a binary (long) string value."""
    return bytes(aString)
def Date(year, month, day):
    "This function constructs an object holding a date value."
    return dateconverter.Date(year, month, day)
def Time(hour, minute, second):
    "This function constructs an object holding a time value."
    return dateconverter.Time(hour, minute, second)
def Timestamp(year, month, day, hour, minute, second):
    "This function constructs an object holding a time stamp value."
    return dateconverter.Timestamp(year, month, day, hour, minute, second)
def DateFromTicks(ticks):
    """This function constructs an object holding a date value from the given ticks value
    (number of seconds since the epoch; see the documentation of the standard Python time
    module for details)."""
    return Date(*time.gmtime(ticks)[:3])
def TimeFromTicks(ticks):
    """This function constructs an object holding a time value from the given ticks value

```

(number of seconds since the epoch; see the documentation of the standard Python time module for details)."""

```
    return Time(*time.gmtime(ticks)[3:6])
```

```
def TimestampFromTicks(ticks):
```

```
    """This function constructs an object holding a time stamp value from the given
    ticks value (number of seconds since the epoch;
    see the documentation of the standard Python time module for details)."""
```

```
    return Timestamp(*time.gmtime(ticks)[:6])
```

```
version = "adodbapi v" + __version__
```

```
import unittest
```

```
from caffe2.python import core, test_util, workspace
```

```
from caffe2.python.control_ops_grad import disambiguate_grad_if_op_output
```

```
from caffe2.python.model_helper import ModelHelper
```

```
import numpy as np
```

```
class TestControl(test_util.TestCase):
```

```
    def test_disambiguate_grad_if_op_output(self):
```

```
        workspace.FeedBlob("cond", np.array(True))
```

```
        workspace.FeedBlob("then_grad", np.array(1))
```

```
        workspace.FeedBlob("else_grad", np.array(2))
```

```
        then_model = ModelHelper(name="then_test_model")
```

```
        then_model.net.Copy("then_grad", "input_grad")
```

```
        else_model = ModelHelper(name="else_test_model")
```

```
        else_model.net.Copy("else_grad", "else_temp_grad")
```

```
        else_model.net.Copy("else_temp", "input_grad")
```

```
        # to BuildGradientGenerators, in forward pass, we need else temp
```

```
        # as one of the output. Which later on results in a grad op like this:
```

```
        grad_op = core.CreateOperator(
```

```
            "If",
```

```

["cond", "then_grad", "else_grad"],
["input_grad", "else_temp_grad"],
then_net=then_model.net.Proto(),
else_net=else_model.net.Proto(),
)

# in certain cases, another branch of the net also generates input_grad
# and we call _DisambiguateGradOpOutput in core.py
new_grad_output = "input_grad" + "_autosplit_" + "0"
disambiguate_grad_if_op_output(grad_op, 0, new_grad_output)
self.assertEqual(grad_op.output[0], new_grad_output)
for arg in grad_op.arg:
    if arg.name == "else_net":
        self.assertEqual(arg.n.op[1].output[0], new_grad_output)
    else:
        self.assertEqual(arg.name, "then_net")
if __name__ == '__main__':
    unittest.main()
def TimeFromTicks(ticks):
    """This function constructs an object holding a time value from the given ticks value
    (number of seconds since the epoch; see the documentation of the standard Python time
    module for details)."""
    return Time(*time.gmtime(ticks)[3:6])
def TimestampFromTicks(ticks):
    """This function constructs an object holding a time stamp value from the given
    ticks value (number of seconds since the epoch;
    see the documentation of the standard Python time module for details)."""
    return Timestamp(*time.gmtime(ticks)[:6])
version = "adodbapi v" + __version__
"""adodbapi - A python DB API 2.0 (PEP 249) interface to Microsoft ADO
Copyright (C) 2002 Henrik Ekelund, version 2.1 by Vernon Cole

```

```
* http://sourceforge.net/projects/adodbapi
```

```
"""
```

```
import sys
```

```
import time
```

```
from .apibase import apilevel, threadsafety, paramstyle
```

```
from .apibase import (
```

```
    Warning,
```

```
    Error,
```

```
    InterfaceError,
```

```
    DatabaseError,
```

```
    DataError,
```

```
    OperationalError,
```

```
    IntegrityError,
```

```
)
```

```
from .apibase import (
```

```
    InternalError,
```

```
    ProgrammingError,
```

```
    NotSupportedError,
```

```
    FetchFailedError,
```

```
)
```

```
from .apibase import NUMBER, STRING, BINARY, DATETIME, ROWID
```

```
from .adodbapi import connect, Connection, __version__, dateconverter, Cursor
```

```
def Binary(aString):
```

```
    """This function constructs an object capable of holding a binary (long) string value."""
```

```
    return bytes(aString)
```

```
def Date(year, month, day):
```

```
    "This function constructs an object holding a date value."
```

```
    return dateconverter.Date(year, month, day)
```

```
def Time(hour, minute, second):
```

```
    "This function constructs an object holding a time value."
```

```
    return dateconverter.Time(hour, minute, second)
```

```
def Timestamp(year, month, day, hour, minute, second):
```

```
    "This function constructs an object holding a time stamp value."
```

```
    return dateconverter.Timestamp(year, month, day, hour, minute, second)
```

```
def DateFromTicks(ticks):
```

```
    """This function constructs an object holding a date value from the given ticks value
    (number of seconds since the epoch; see the documentation of the standard Python time
    module for details)."""
```

```
    return Date(*time.gmtime(ticks)[:3])
```

```
def TimeFromTicks(ticks):
```

```
    """This function constructs an object holding a time value from the given ticks value
    (number of seconds since the epoch; see the documentation of the standard Python time
    module for details)."""
```

```
    return Time(*time.gmtime(ticks)[3:6])
```

```
def TimestampFromTicks(ticks):
```

```
    """This function constructs an object holding a time stamp value from the given
    ticks value (number of seconds since the epoch;
    see the documentation of the standard Python time module for details)."""
```

```
    return Timestamp(*time.gmtime(ticks)[6:])
```

```
version = "adodbapi v" + __version__
```

"""adodbapi - A python DB API 2.0 (PEP 249) interface to Microsoft ADO

Copyright (C) 2002 Henrik Ekelund, version 2.1 by Vernon Cole

* <http://sourceforge.net/projects/adodbapi>

"""

import sys

import time

from .apibase import apilevel, threadsafety, paramstyle

from .apibase import (

Warning,

Error,

InterfaceError,

DatabaseError,

DataError,

OperationalError,

IntegrityError,

)

from .apibase import (

InternalError,

ProgrammingError,

NotSupportedError,

FetchFailedError,

)

from .apibase import NUMBER, STRING, BINARY, DATETIME, ROWID

from .adodbapi import connect, Connection, __version__, dateconverter, Cursor

def Binary(aString):

```
"""This function constructs an object capable of holding a binary (long) string value."""  
return bytes(aString)
```

```
def Date(year, month, day):  
    "This function constructs an object holding a date value."  
    return dateconverter.Date(year, month, day)
```

```
def Time(hour, minute, second):  
    "This function constructs an object holding a time value."  
    return dateconverter.Time(hour, minute, second)
```

```
def Timestamp(year, month, day, hour, minute, second):  
    "This function constructs an object holding a time stamp value."  
    return dateconverter.Timestamp(year, month, day, hour, minute, second)
```

```
def DateFromTicks(ticks):  
    """This function constructs an object holding a date value from the given ticks value  
    (number of seconds since the epoch; see the documentation of the standard Python time  
    module for details)."""  
    return Date(*time.gmtime(ticks)[:3])
```

```
def TimeFromTicks(ticks):  
    """This function constructs an object holding a time value from the given ticks value  
    (number of seconds since the epoch; see the documentation of the standard Python time  
    module for details)."""  
    return Time(*time.gmtime(ticks)[3:6])
```

```
def TimestampFromTicks(ticks):  
    """This function constructs an object holding a time stamp value from the given  
    ticks value (number of seconds since the epoch;  
    see the documentation of the standard Python time module for details)."""  
    return Timestamp(*time.gmtime(ticks)[:6])
```

```
version = "adodbapi v" + __version__
```

```
import unittest  
from caffe2.python import core, test_util, workspace  
from caffe2.python.control_ops_grad import disambiguate_grad_if_op_output  
from caffe2.python.model_helper import ModelHelper  
import numpy as np
```

```
class TestControl(test_util.TestCase):  
    def test_disambiguate_grad_if_op_output(self):  
        workspace.FeedBlob("cond", np.array(True))  
        workspace.FeedBlob("then_grad", np.array(1))  
        workspace.FeedBlob("else_grad", np.array(2))  
  
        then_model = ModelHelper(name="then_test_model")  
        then_model.net.Copy("then_grad", "input_grad")
```

```
else_model = ModelHelper(name="else_test_model")
else_model.net.Copy("else_grad", "else_temp_grad")
else_model.net.Copy("else_temp", "input_grad")
```

```
# to BuildGradientGenerators, in forward pass, we need else temp
# as one of the output. Which later on results in a grad op like this:
```

```
grad_op = core.CreateOperator(
    "If",
    ["cond", "then_grad", "else_grad"],
    ["input_grad", "else_temp_grad"],
    then_net=then_model.net.Proto(),
    else_net=else_model.net.Proto(),
)
```

```
# in certain cases, another branch of the net also generates input_grad
```

```
# and we call _DisambiguateGradOpOutput in core.py
```

```
new_grad_output = "input_grad" + "_autosplit_" + "0"
```

```
disambiguate_grad_if_op_output(grad_op, 0, new_grad_output)
```

```
self.assertEqual(grad_op.output[0], new_grad_output)
```

```
for arg in grad_op.arg:
```

```
    if arg.name == "else_net":
```

```
        self.assertEqual(arg.n.op[1].output[0], new_grad_output)
```

```
    else:
```

```
        self.assertEqual(arg.name, "then_net")
```

```
if __name__ == '__main__':
```

```
    unittest.main()
```

6.2 SAMPLE SCREENSHOT

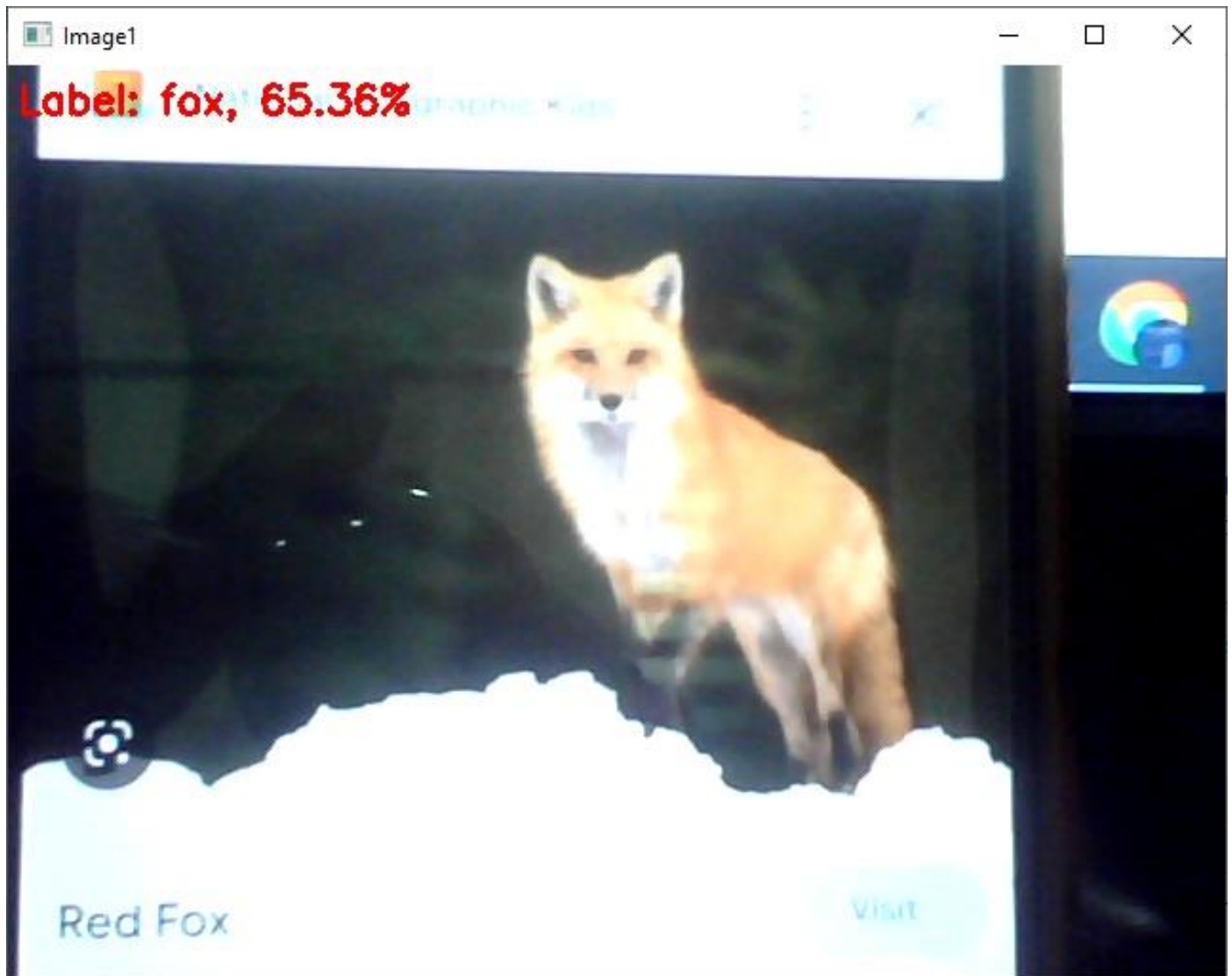


Image1

Label: lion, 77.28%



One of the most important things to remember is that the lion is a big cat and can be very dangerous. It is important to keep a safe distance and not to provoke it. If you are in a situation where you are threatened by a lion, it is best to run away quickly and not to turn back.

7.CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

In summary, our animal detection project represents a significant advancement in addressing the complex challenges posed by human-wildlife conflicts and in bolstering wildlife conservation initiatives. Through the development and deployment of cutting-edge deep learning algorithms, we have constructed a robust system capable of reliably identifying and categorizing various animal species within surveillance footage. The efficacy of our methodology in real-world scenarios underscores its potential to mitigate risks to both human safety and wildlife preservation. By seamlessly integrating with existing CCTV surveillance infrastructure, our solution offers a practical and scalable approach to monitoring animal behavior and issuing timely alerts to relevant stakeholders in the event of potential threats. Looking forward, ongoing research and optimization efforts hold the promise of further enhancing the accuracy, efficiency, and versatility of our system across diverse environments and circumstances. Ultimately, our project contributes to the broader objective of fostering a harmonious coexistence between human populations and wildlife, thereby safeguarding ecosystems for the benefit of present and future generations.

7.2 FUTURE ENHANCEMENT

Future work will focus on enhancing the system's robustness in varied environmental conditions, optimizing computational efficiency, exploring multi-sensor integration for improved detection accuracy, investigating real-time decision-making algorithms for proactive intervention, and extending the application to broader wildlife conservation efforts.DCT-Filtering

8.REFERENCES

- [1] . Ziv Zemah Shamir, Eyal Bigal, Aviad Scheinin Dan Tchernov, Localization of acoustically tagged marine animals in under-ranked conditions. IEEE Transactions on Mobile Computing (Volume: 20, Issue: 3, 01 March 2021),2021
- [2] B. Natarajan; R. Elakkiya; R. Bhuvaneswari; Kashif Saleem, Creating Alert messages based on Wild Animal Activity Detection using Hybrid Deep Neural Networks, 2023 IEEE 12th Global Conference on Consumer Electronics (GCCE),2023
- [3] E.A. Edirisinghe; Andrew Leonce; Gregory Simkins; Tamer Khafaga, Deep Neural Networks Based Multiclass Animal Detection and Classification in Drone Imagery, 2023 International Symposium on Networks, Computers and Communications (ISNCC)
- [4] K.Kiruthika¹, A R Janani Sri², P.Indhumathy³, V. Bavithra⁴, AUTOMATING THE IDENTIFICATION OF FOREST ANIMALS AND ALERTING IN CASE OF ANIMAL ENCROACHMENT, 2019 International Conference on Deep Learning and Machine Learning in Emerging Applications (Deep-ML),2019
- [5] Rodolfo Vera-Amaro; Mario E. Rivero Angeles; Alberto Luviano-Juarez, Design and Analysis of Wireless Sensor Networks for Animal Tracking in Large Monitoring Polar Regions Using Phase-Type Distributions and Single Sensor Model, 2018 International Conference on Information and Communication Technology Robotics (ICT-ROBOT)
- [6] Sreedevi K L; Anitha Edison, Wild Animal Detection using Deep learning, 2022 IEEE 19th India Council International Conference (INDICON),IEEE Transactions on Information Forensics and Security (Volume: 14 , Issue: 9 , Sept. 2019).
- [7]Talmon Alexandri; Ziv Zemah Shamir; Eyal Bigal; Aviad Scheinin; Dan Tchernov; Roe Diamant, Localization of Acoustically Tagged Marine Animals in Under-Ranked Conditions. IEEE Transactions on Mobile Computing (Volume: 20, Issue: 3, 01 March 2021)

[8] R. Viji; G. Sreelatha; S. Santhosh Kumar, Towards Implementation of Detection and Tracking of Wild Animals, 2023 9th International Conference on Smart Computing and Communications (ICSCC).

[9] Manu Agarwal; Enkhzol Dovdon; Luis Roma Barge; Yanja Dajsuren; Jakob de Vlieg, A HPC-Based Data Analytics Platform Architecture for Data-Driven Animal Phenotype Detection, 2023 IEEE 6th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech),2023.

[10] Anuvind P E; Abhishek C K; Mohamed Shibili; Rahila C K; Neethu K, Development and Implementation of an Animal Intrusion Detection System Using Image and Audio Processing, 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT),2023.