WILDLIFE DETECTION AND MONITORING

A PROJECT REPORT

Submitted by

Dharamveer Prakash(RA2111003010421)

Rahul Narayanan (RA2111003010440)

Vignesh Ram (RA2111003010445)

Under the Guidance of

DR. S. PADMINI

Associate Professor, Department of Computing Technologies

In partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR – 603 203

APRIL 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this B.Tech project report titled " **WILDLIFE DETECTION AND MONITORING**" is the bonafide work of Mr. Dharamveer Prakash[RA2111003010421] , Mr. Rahul Narayanan [RA2111003010440] and Mr.Vignesh Ram [RA2111003010445]who carried out the project work under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

DR. S. PADMINI Ph.D.

Associate Professor

Department of Computing Technologies

TABLE OF CONTENTS

ABSTRACT

The rapid decline in global biodiversity underscores the urgency of developing effective wildlife monitoring systems. Traditional methods of wildlife detection, such as camera traps and manual surveys, are labor-intensive, time-consuming, and often inefficient. This project proposes an automated wildlife detection system leveraging machine learning techniques to streamline the process of wildlife monitoring. The system utilizes convolutional neural networks (CNNs) for image classification, trained on a diverse dataset of wildlife images collected from various habitats. Transfer learning techniques are employed to fine-tune pre-trained models, enhancing their ability to recognize specific species and environmental conditions. Key features of the proposed system include real-time image processing, allowing for immediate detection and classification of wildlife from live camera feeds or static images. The system is designed to be scalable and adaptable, capable of integrating with existing camera trap networks or deployed as standalone units in remote locations. Evaluation of the system's performance includes accuracy metrics, such as precision, recall, and F1 score, as well as computational efficiency measures to assess real-world applicability. Field tests in different ecological settings validate the system's effectiveness in detecting target species while minimizing false positives. The proposed automated wildlife detection system offers a cost-effective and scalable solution for wildlife monitoring, enabling conservation efforts to be more targeted and efficient. By harnessing the power of machine learning, this technology represents a significant advancement in wildlife conservation and habitat management.

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. General

Wildlife detection and monitoring play pivotal roles in understanding ecosystems, conserving biodiversity, and mitigating human-wildlife conflicts. Over the past decades, advancements in technology have revolutionized the methodologies and capabilities within this field, offering unprecedented opportunities for researchers, conservationists, and policymakers alike. In this rapidly evolving landscape, the symbiotic relationship between technology and conservation science is becoming increasingly evident. From the deployment
of camera traps and radio telemetry to the utilization of satellite imagery and artificial intelligence, the toolbox available for wildlife researchers is expanding at an unprecedented pace.

Moreover, the democratization of technology through citizen science initiatives and community engagement has empowered individuals worldwide to contribute meaningfully to wildlife monitoring efforts. By fostering collaboration and knowledge sharing, these grassroots initiatives are transforming the way we perceive and address conservation challenges.

1.2.Scope

Scope for the Automated Wildlife Detection System Using Machine Learning Techniques:

Species Diversity: The system can be designed to detect a wide range of wildlife species, including mammals, birds, reptiles, and insects, depending on the targeted ecosystem. The scope can encompass both common and endangered species.

Habitat Variation: The system should be adaptable to different habitats, including forests, grasslands, wetlands, and marine environments. It should account for variations in lighting conditions, terrain, and vegetation density.

Camera Platforms: The scope includes compatibility with various camera platforms, including stationary camera traps, drones, and satellites. Integration with existing camera trap networks or the development of proprietary hardware for deployment can be considered.

Real-time Processing: Implementation of real-time image processing capabilities enables immediate detection and response to wildlife presence, facilitating timely conservation interventions or data collection.

Data Management: The system should include mechanisms for efficient data storage, management, and retrieval. This involves organizing annotated images, metadata, and detection logs for analysis and reporting purposes.

User Interface: Development of user-friendly interfaces for system configuration, monitoring, and data visualization enhances usability for conservationists, researchers, and wildlife managers.

Scalability: The system's architecture should be scalable to accommodate large-scale deployments across multiple locations, allowing for comprehensive wildlife monitoring on a regional or global scale.

Robustness: Robustness against environmental factors such as weather conditions, occlusions, and image noise is essential for reliable performance in challenging field conditions.

Privacy and Ethics: Consideration of privacy concerns and ethical implications related to wildlife monitoring, including protocols for data anonymization and informed consent where necessary.

Integration with Conservation Efforts: Collaboration with conservation organizations, government agencies, and research institutions facilitates the integration of the system into existing conservation initiatives, maximizing its impact on wildlife protection and management.

Long-term Monitoring: The system's scope may extend to long-term monitoring projects aimed at tracking changes in wildlife populations, habitat dynamics, and human-wildlife interactions over time.

Research and Development: Continued research and development efforts are necessary to improve the system's accuracy, efficiency, and adaptability through advancements in machine learning algorithms, sensor technology, and data processing techniques.

By addressing these aspects within the scope of the project, the Automated Wildlife Detection System can contribute significantly to wildlife conservation efforts worldwide.

1.3. Motivation

- **Conservation Imperative:** The alarming rates of species decline and habitat loss underscore the urgent need for effective conservation strategies. Wildlife detection and monitoring provide essential data for assessing population trends, identifying conservation priorities, and implementing targeted interventions to protect vulnerable species and ecosystems.
- **Ecosystem Health:** Wildlife plays integral roles in ecosystem functioning, from regulating populations of prey species to influencing nutrient cycling and seed dispersal. Monitoring wildlife populations and behaviors allows us to gauge the health and resilience of ecosystems, thereby informing management practices aimed at preserving ecosystem services and biodiversity.
- **Human-Wildlife Conflict Mitigation:** As human populations expand and encroach upon natural habitats, conflicts between humans and wildlife escalate. Monitoring wildlife movements and behaviors helps identify areas of conflict, develop mitigation strategies, and foster coexistence between humans and wildlife, ultimately reducing negative interactions and promoting community livelihoods.
- **Public Engagement and Education:** Wildlife detection and monitoring initiatives offer opportunities for public engagement and education. Citizen science projects, in particular, empower individuals to contribute to scientific research, raise awareness about conservation issues, and foster a sense of stewardship towards the environment.
- **Global Challenges:** Climate change, habitat fragmentation, invasive species, and

other anthropogenic pressures pose significant threats to wildlife worldwide. Effective detection and monitoring programs provide essential data for understanding and responding to these challenges, informing adaptive management strategies and policy decisions at local, regional, and global scales.

- **Scientific Discovery:** Studying wildlife provides invaluable insights into animal behavior, evolutionary processes, and ecological dynamics. Advanced monitoring technologies enable researchers to uncover new discoveries, refine ecological theories, and expand our understanding of the natural world.

1.4 Computer Vision

In recent years, computer vision has emerged as a powerful tool in the field of wildlife detection and monitoring, revolutionizing the way we observe, analyze, and conserve natural ecosystems. By leveraging advances in machine learning and image processing techniques, computer vision systems can automatically detect, classify, and track wildlife species from visual data captured by cameras, drones, or satellite imagery.

Traditional methods of wildlife monitoring, such as manual surveys and camera traps, are often labor-intensive, time-consuming, and prone to human error. In contrast, computer vision offers a scalable and efficient solution that enables real-time or near-real-time detection of wildlife, providing invaluable insights into species distributions, behaviors, and population dynamics.

At the heart of computer vision for wildlife detection and monitoring are sophisticated algorithms capable of extracting meaningful information from digital images or video streams. These algorithms utilize deep learning architectures, such as convolutional neural networks (CNNs), to automatically learn relevant features and patterns from vast amounts of labeled training data.

CHAPTER 2

LITERATURE REVIEW

In this section, we delve into the prior studies and research conducted in an attempt to highlight the significance of such technology in the field of wildlife conservation. The information presented in this section focuses on the application of computer vision techniques in wildlife monitoring. It reviews studies that have used machine learning algorithms, such as convolutional neural networks (CNNs), for species identification and object detection. It also discusses the ethics and legality of the project and its importance in the conservation effort.

2.1 Wildlife Ecology and Behavior:

Understanding wildlife ecology and behavior is fundamental for designing effective wildlife monitoring projects. Wildlife populations interact with their environments in complex ways, influenced by factors such as habitat availability, resource distribution, predation, and competition. Ecological principles, including niche theory, population dynamics, and community ecology, provide insights into the relationships between species and their habitats. Additionally, knowledge of animal behavior, such as foraging strategies, mating behaviors, and seasonal movements, helps researchers anticipate where and when to observe wildlife. For instance, understanding the migratory patterns of birds or the ranging behavior of large mammals informs the placement of camera traps or survey efforts in strategic locations. Conservation biologists and ecologists rely on this understanding to develop conservation strategies that protect critical habitats, mitigate human-wildlife conflicts, and conserve biodiversity.

2.2 Image Processing and Computer Vision:

Image processing and computer vision play crucial roles in analyzing visual data collected during wildlife monitoring projects. Image processing techniques, such as filtering,

enhancement, and noise reduction, improve the quality and clarity of images, making them more suitable for automated analysis. Computer vision algorithms enable the extraction of meaningful information from images, such as identifying wildlife species, detecting animal movements, and quantifying habitat characteristics. Advanced machine learning methods, including deep learning architectures like convolutional neural networks (CNNs), have revolutionized computer vision by enabling systems to learn complex patterns and features directly from raw image data. These techniques are essential for automating tasks such as species identification, object detection, and behavioral analysis, facilitating more efficient and accurate wildlife monitoring efforts.

2.3 Machine Learning and Deep Learning:

Machine learning and deep learning are indispensable tools for developing automated wildlife detection and monitoring systems. Machine learning algorithms enable computer systems to learn patterns and make predictions from data without explicit programming. Deep learning, a subset of machine learning, utilizes neural networks with multiple layers to automatically discover hierarchical representations of data. In the context of wildlife monitoring, machine learning models can be trained on labeled datasets of wildlife images to recognize species, detect animals in images, and classify behaviors. Deep learning approaches, such as CNNs, have shown remarkable success in tasks like species identification, even with large-scale and diverse datasets. By leveraging machine learning and deep learning techniques, researchers can build robust and scalable systems for wildlife detection and monitoring across different environments and species.

2.4 Ethical and Legal Considerations:

Ethical considerations in wildlife monitoring encompass various aspects, including animal welfare, privacy, and the responsible use of technology. Researchers must ensure that monitoring activities minimize harm to wildlife, avoid unnecessary disturbance, and adhere to ethical guidelines for animal research and observation. This may involve obtaining permits from relevant authorities, implementing protocols for humane trapping and handling of animals, and minimizing the impact of monitoring equipment on wildlife

behavior. Additionally, researchers must consider the ethical implications of data collection and sharing, including obtaining informed consent for sharing images of individuals captured during monitoring activities. From a legal perspective, wildlife monitoring projects are subject to regulations governing wildlife research, data collection, and privacy rights. Researchers must comply with national and international laws, such as the Endangered Species Act, the Convention on Biological Diversity, and data protection regulations, to ensure ethical and legal integrity in their work.

2.5 Conservation Biology and Management:

Conservation biology integrates principles from ecology, genetics, and other disciplines to study and preserve biodiversity. Wildlife monitoring plays a crucial role in conservation biology by providing data on species distributions, population trends, and habitat conditions. Conservation management strategies, such as protected area design, habitat restoration, and species reintroduction, rely on accurate and up-to-date information from monitoring programs. By monitoring changes in wildlife populations and ecosystems over time, conservation biologists can assess the effectiveness of conservation interventions and adapt management strategies as needed. Collaboration with stakeholders, policymakers, and local communities is essential for implementing evidence-based conservation actions that address threats to biodiversity and promote sustainable land use practices.

2.6 Community Engagement and Stakeholder Collaboration:

Community engagement and stakeholder collaboration are essential for building support, fostering cooperation, and achieving long-term success in wildlife monitoring and conservation projects. Engaging with local communities, indigenous peoples, and stakeholders from the outset of a project helps ensure that monitoring activities respect local knowledge, cultural values, and traditional practices. By involving communities in data collection, analysis, and decision-making processes, researchers can empower local stakeholders to become active participants in conservation efforts. Collaborative approaches that emphasize co-management, knowledge exchange, and capacity building enhance project sustainability and promote inclusive governance of natural resources.

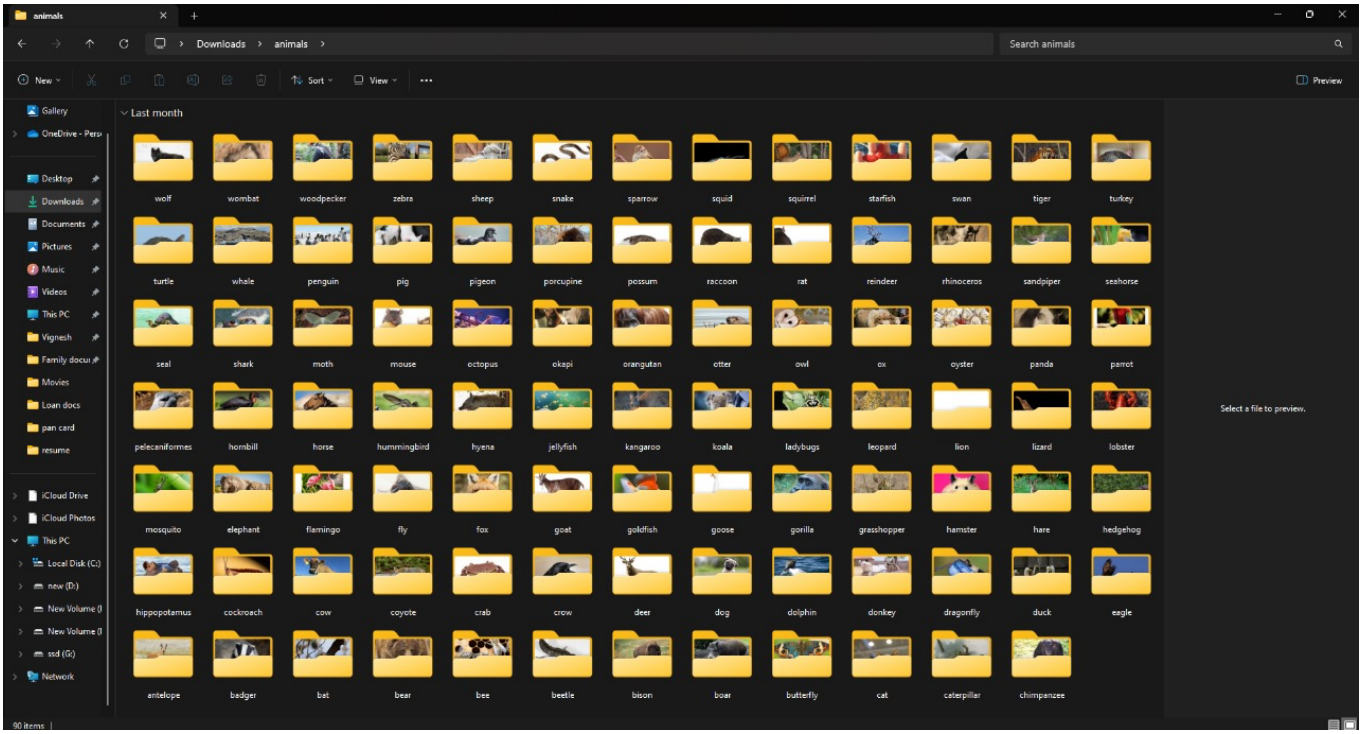
Effective communication and outreach strategies, including workshops, citizen science initiatives, and participatory mapping exercises, facilitate dialogue, build trust, and strengthen partnerships between researchers and stakeholders.

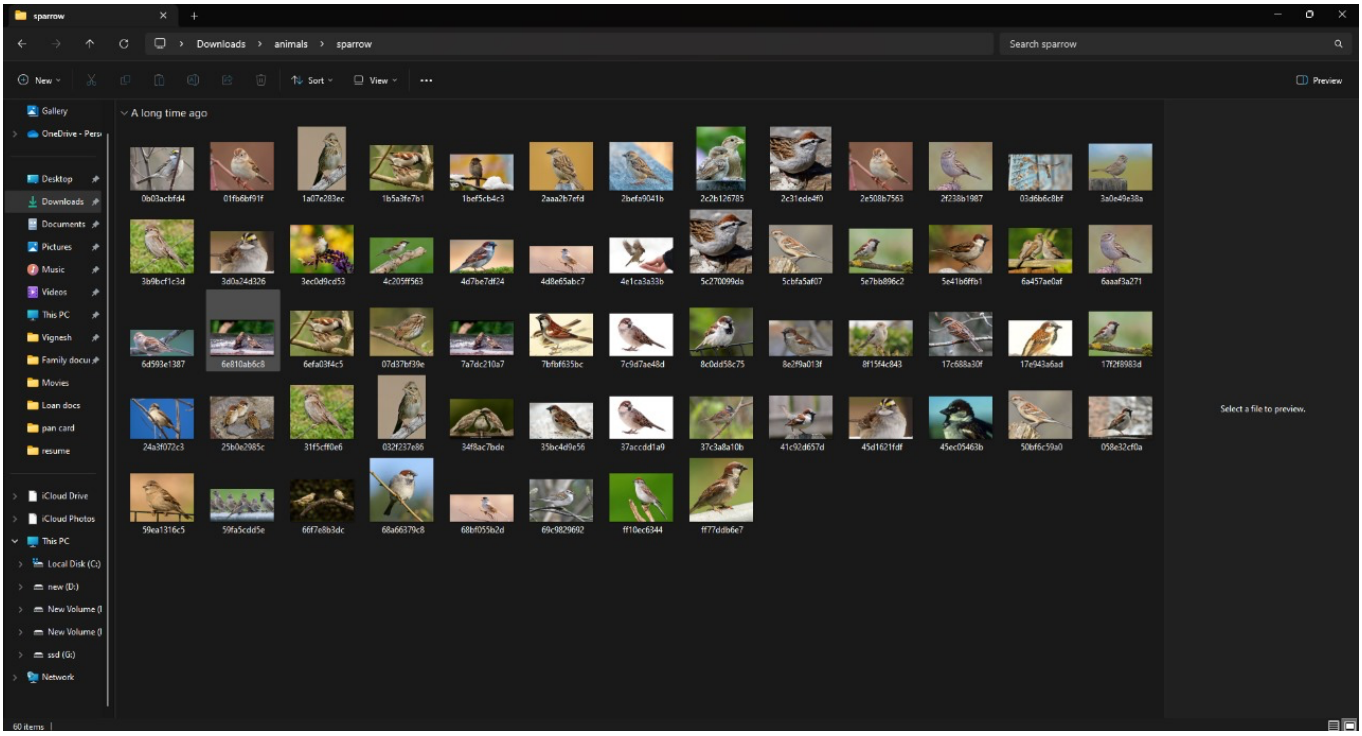
CHAPTER 3

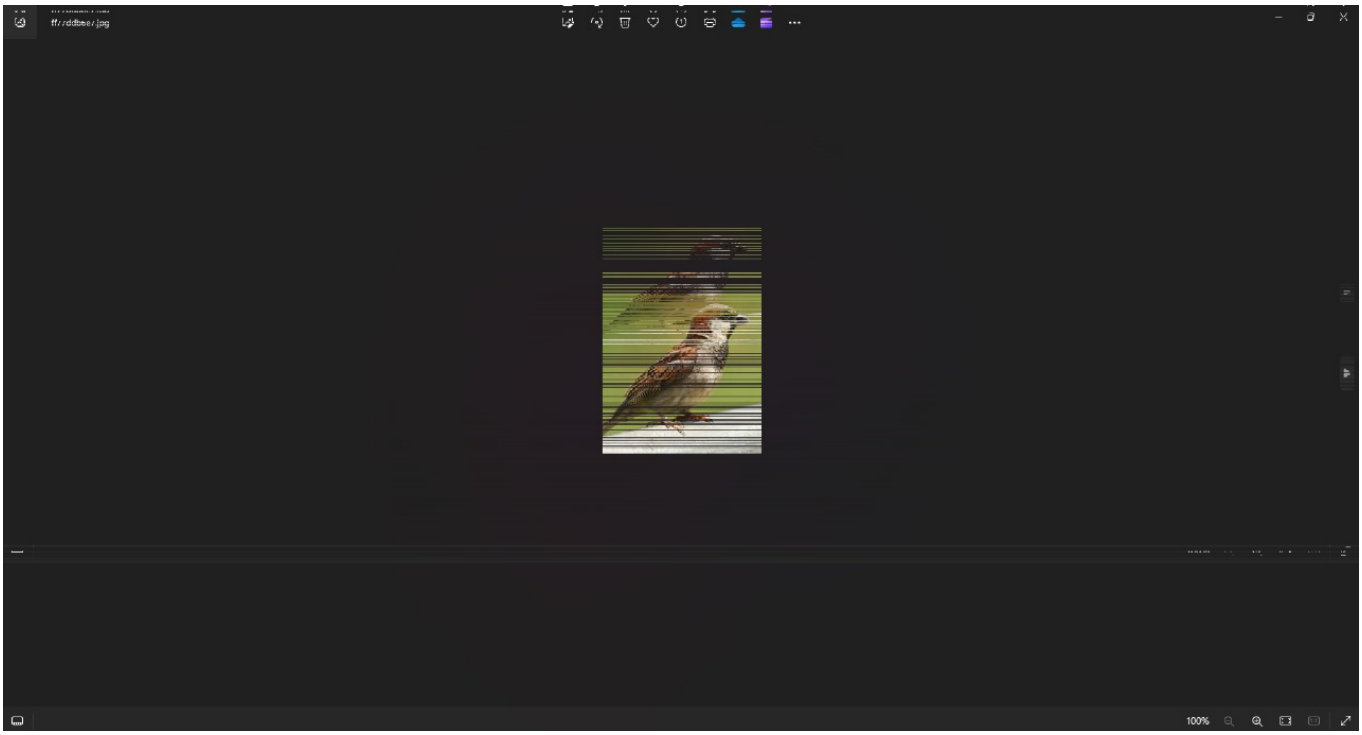
PROPOSED METHODOLOGY

In this section, we offer information regarding the dataset employed and outline the proposed architecture for wildlife detection. First of all, we created a dataset containing images of various animals. Secondly, using computer vision and the dataset, we created a CNN Model with the help of VGG16 algorithm to identify the animal present in the image.

3.1 Dataset:




sparrow
Downloads › animals › sparrow
Search sparrow
New
Sort
View
Preview
Gallery
OneDrive - Perso
Desktop
Downloads
Documents
Pictures
Music
Videos
This PC
Vignesh
Family docu
Movies
Loan docs
pan card
resume
iCloud Drive
iCloud Photos
This PC
Local Disk (C:)
new (D:)
New Volume (I
New Volume (I
ssd (G:)
Network
A long time ago
Select a file to preview.
60 items


100%

3.2 Learning Model: The model employs deep learning and convolutional neural networks (CNNs), trained on labeled datasets to recognize species-specific features and patterns.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.__operators__.getitem (  (None, 224, 224, 3)       0
 SlicingOpLambda)

 tf.nn.bias_add (TFOpLambda  (None, 224, 224, 3)       0
 )

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 global_average_pooling2d (  (None, 512)               0
 GlobalAveragePooling2D)

 dropout (Dropout)           (None, 512)               0

 dense (Dense)               (None, 90)                46170

=================================================================
Total params: 14760858 (56.31 MB)
Trainable params: 46170 (180.35 KB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

3.3 Algorithm

- Data Collection:
    - Collect visual data using camera traps, drones, or satellite imagery in the target area.
    - Ensure that the data collection process adheres to ethical guidelines and legal regulations.
- Data Preprocessing:
    - Clean and preprocess the raw images to enhance quality, remove noise, and standardize formatting.
    - Apply techniques such as image resizing, color correction, and normalization to prepare the data for analysis.

- Feature Extraction:
    - Extract relevant features from the preprocessed images, such as shapes, textures, and colors.
    - Use techniques like histogram of oriented gradients (HOG), scale-invariant feature transform (SIFT), or deep learning-based feature extraction to capture discriminative information from the images.

- Training Data Preparation:
    - Create a labeled dataset by annotating the images with metadata, including species labels, location coordinates, and timestamps.
    - Split the dataset into training, validation, and test sets for model training and evaluation.

- Model Training:
    - Select an appropriate machine learning model or deep learning architecture for the task, such as a convolutional neural network (CNN).
    - Train the model using the labeled training data to learn to classify wildlife species or detect animals in the images.
    - Fine-tune the model using techniques like transfer learning to leverage pre-trained models and improve performance.

- Model Evaluation:
    - Evaluate the trained model using the validation set to assess its accuracy, precision, recall, and F1 score.
    - Fine-tune model hyperparameters and architecture based on evaluation metrics to optimize performance.

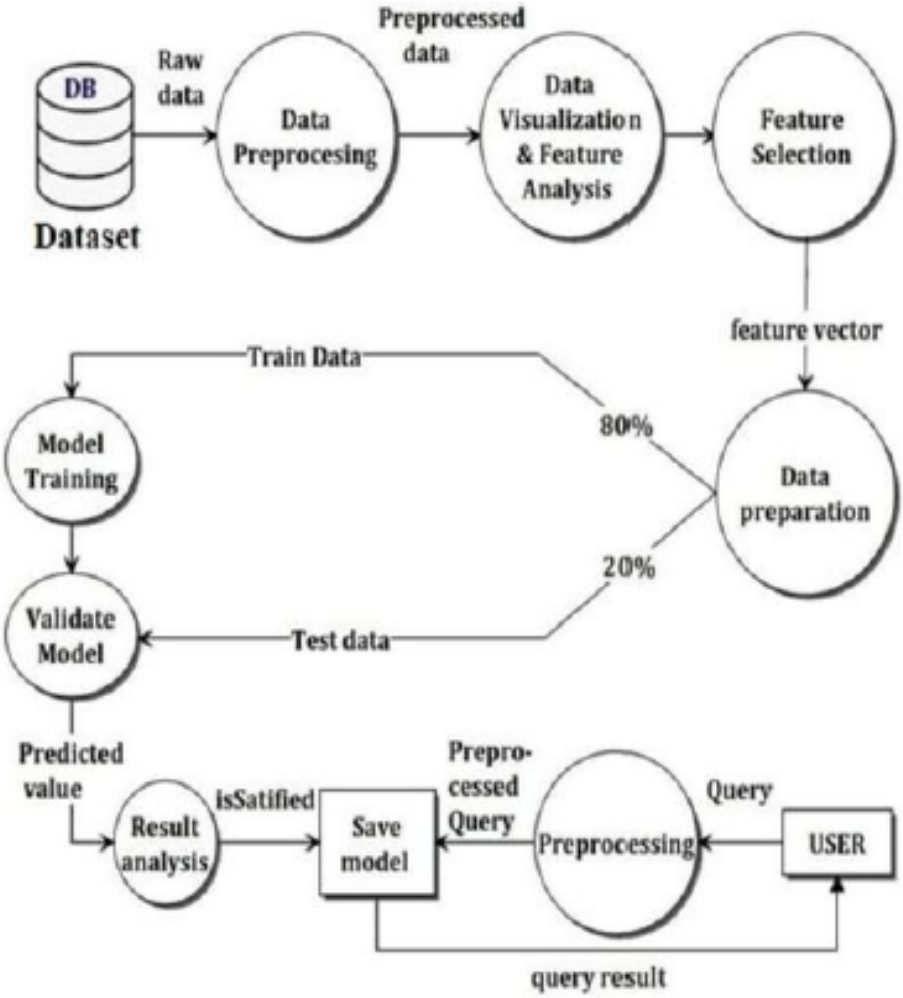
- Testing and Deployment:
    - Test the trained model on unseen data from the test set to evaluate its generalization performance.
    - Deploy the model to analyze new images in real-time or batch processing mode for wildlife detection and monitoring.

- Post-processing and Analysis:
    - Post-process the model predictions to filter out false positives and refine the detection results.
    - Analyze the detected wildlife data to extract insights into species distributions, population trends, and habitat usage patterns.

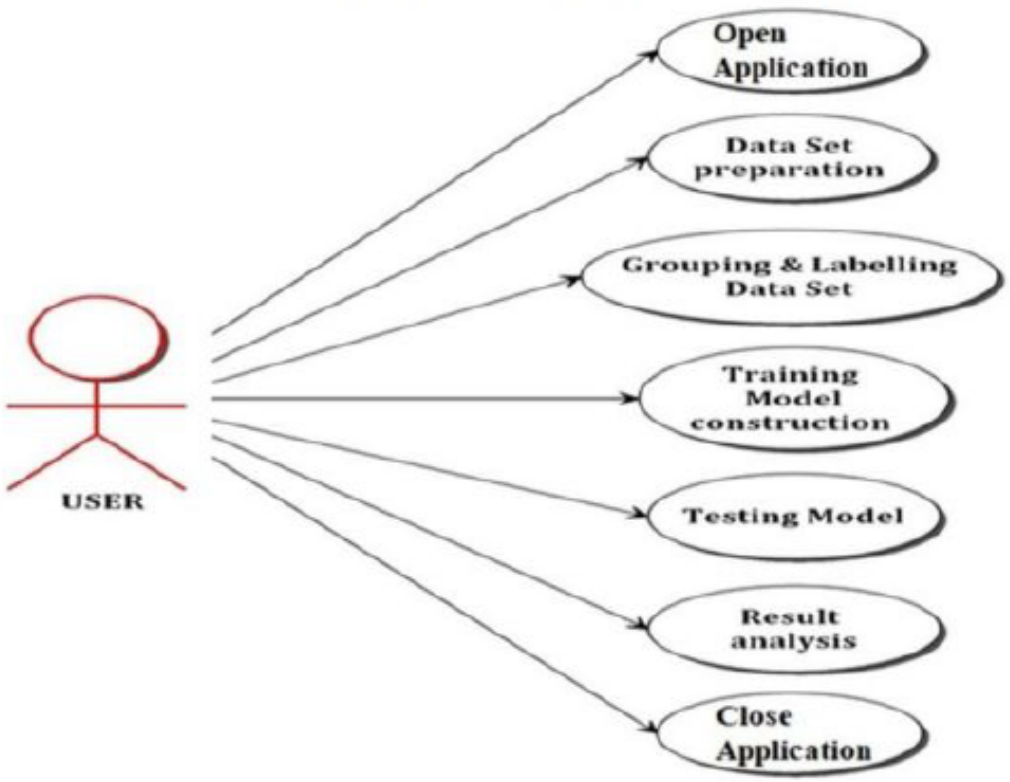
- Iterative Improvement:
    - Continuously monitor and evaluate the performance of the deployed model in real-world scenarios.
    - Collect feedback from stakeholders and users to identify areas for improvement and incorporate updates to the algorithm as needed.

- Documentation and Reporting:
    - Document the entire process, including data collection procedures, model training parameters, and evaluation results.
    - Prepare reports or presentations summarizing the findings and conclusions of the wildlife detection and monitoring project.

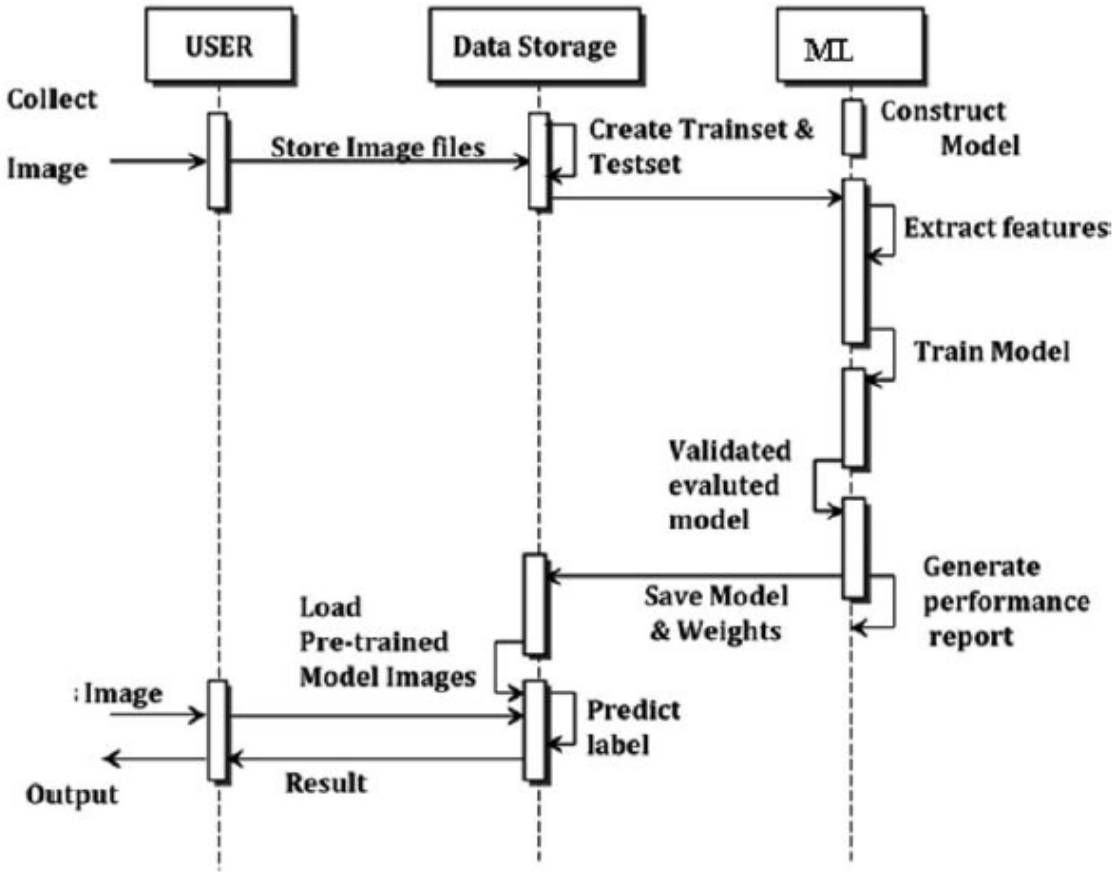
3.4 UML DIAGRAM:




Images
Feature Extractor (Backbone)
Multi-scale features
Detector
Bounding Boxes, Classes


USER
Open Application
Data Set preparation
Grouping & Labelling Data Set
Training Model construction
Testing Model
Result analysis
Close Application

Sequence Diagram



CHAPTER 4

CODING AND TESTING

Code:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import random
from cv2 import resize
from glob import glob

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

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

!ls /content/drive/

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import random
from cv2 import resize
from glob import glob
import tensorflow as tf

img_height = 224
img_width = 224


train_ds = tf.keras.utils.image_dataset_from_directory(
  r'/content/drive/MyDrive/animals',
  validation_split=0.2,
  subset='training',
  image_size=(img_height, img_width),
  batch_size=32,
  seed=42,
  shuffle=True)
```

```
val_ds = tf.keras.utils.image_dataset_from_directory(
  r'/content/drive/MyDrive/animals',
  validation_split=0.2,
  subset='validation',
  image_size=(img_height, img_width),
  batch_size=32,
  seed=42,
  shuffle=True)

#Let's load VGG-16 pretrained on imagenet as the base model.

base_model = tf.keras.applications.VGG16(
    include_top=False,
    weights='imagenet',
    input_shape=(img_height, img_width, 3)
)
base_model.trainable = False # Freeze VGG-16 for now

import torch

# Check if GPU is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Assuming you have a tensor 'input_tensor' and a model 'model'
input_tensor = torch.tensor([1, 2, 3])  # Example tensor
model = torch.nn.Linear(10, 5)  # Example model

input_tensor = input_tensor.to(device)
model = model.to(device)

#Add own output layers to form a complete model.

inputs = tf.keras.Input(shape=(img_height, img_width, 3))
x = tf.keras.applications.vgg16.preprocess_input(inputs)
x = base_model(x, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dropout(0.3)(x)
outputs = tf.keras.layers.Dense(90)(x)
model = tf.keras.Model(inputs, outputs)

model.summary()

model.compile(optimizer=tf.keras.optimizers.legacy.Adam(0.001),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

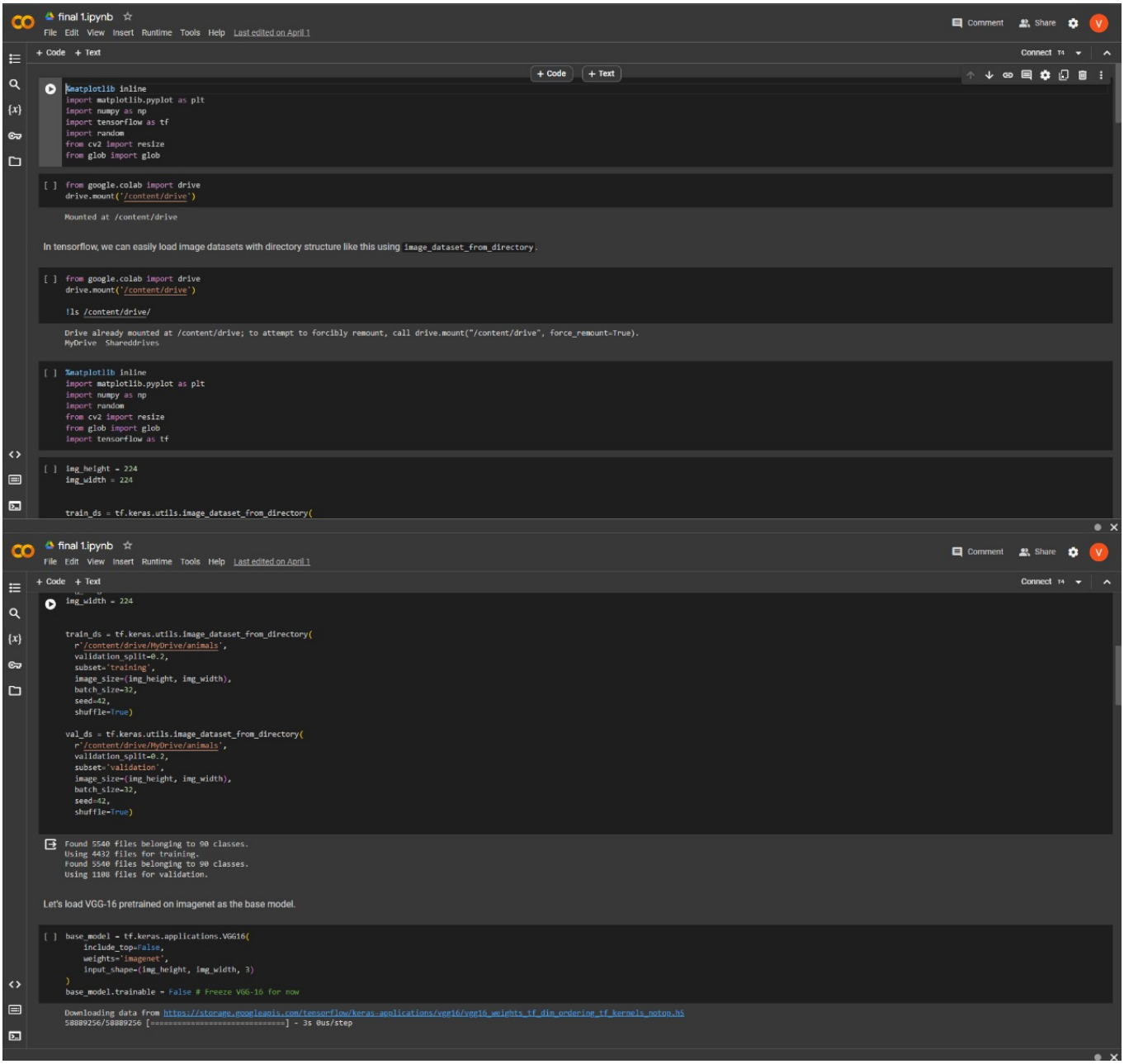
```
#Train the fully-connected layer.

num_epochs = 10
model.fit(train_ds, validation_data=val_ds, epochs=num_epochs,
   callbacks = [
      tf.keras.callbacks.EarlyStopping(
         # Stop training when `val_loss` is no longer improving
         monitor="val_loss",
         # "no longer improving" being defined as "no better than 1e-2 less"
         min_delta=1e-2,
         # "no longer improving" being further defined as "for at least 2 epochs"
         patience=2,
         verbose=1,
         restore_best_weights=True
      )
   ]
)

#If output layer is showing good result, now we unfreeze part of the VGG-16 to do fine-
tuning.

# fine tuning
base_model.trainable = True
for layer in base_model.layers[:14]:
   layer.trainable = False
model.summary()

model.compile(optimizer=tf.keras.optimizers.Adam(0.0001), # use a lower learning rate
when fine-tuning
            loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
            metrics=['accuracy'])

num_epochs = 5
history = model.fit(train_ds, validation_data=val_ds, epochs=num_epochs,
   callbacks = [
      tf.keras.callbacks.EarlyStopping(
         # Stop training when `val_loss` is no longer improving
         monitor="val_loss",
         # "no longer improving" being defined as "no better than 1e-2 less"
         min_delta=1e-2,
         # "no longer improving" being further defined as "for at least 2 epochs"
         patience=2,
         verbose=1,
      )
   ]
```

```
#After training, choose a random picture and see its prediction.

img_path_list = glob(r'/content/drive/MyDrive/animals/tiger/*.jpg')
print(img_path_list)

import random
from glob import glob

# Assuming multiple images are in '/content/drive/MyDrive/'
img_path_list = glob('/content/drive/MyDrive/animals/sparrow/*.jpg')  # Modify the
pattern if necessary

if img_path_list:
    img_name = random.choice(img_path_list)
    img = tf.keras.utils.load_img(
        img_name, target_size=(img_height, img_width)
    )
    img_array = tf.keras.utils.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0) # Create a batch

    predictions = model.predict(img_array)
    score = tf.nn.softmax(predictions[0])

    print(
        "{} most likely belongs to {} with a {:.2f} percent confidence."
        .format(img_name, train_ds.class_names[np.argmax(score)], 100 * np.max(score))
    )

    plt.figure(figsize=(2, 2))
    plt.imshow((img_array[0].numpy()).astype('uint8'))
    plt.title("{}:{:.2f}".format(train_ds.class_names[np.argmax(score)], 100 *
np.max(score)))
    plt.axis('off')
    plt.show()
```

CHAPTER 5

RESULTS

Screenshots:



final 1.ipynb

File Edit View Insert Runtime Tools Help Last edited on April 1

Comment Share

+ Code + Text

Connect T4

```
[ ] import torch

    # Check if GPU is available
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # Assuming you have a tensor 'input_tensor' and a model 'model'
    input_tensor = torch.tensor([1, 2, 3])  # Example tensor
    model = torch.nn.Linear(10, 5)  # Example model

    input_tensor = input_tensor.to(device)
    model = model.to(device)
```

```
inputs = tf.keras.Input(shape=(img_height, img_width, 3))
x = tf.keras.applications.vgg16.preprocess_input(inputs)
x = base_model(x, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dropout(0.3)(x)
outputs = tf.keras.layers.Dense(90)(x)
model = tf.keras.Model(inputs, outputs)

model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.__operators__.getitem (  (None, 224, 224, 3)       0
 SlicingOpLambda)

 tf.nn.bias_add (TFOpLambda  (None, 224, 224, 3)       0
 )

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 global_average_pooling2d (  (None, 512)               0
 GlobalAveragePooling2D)

 dropout (Dropout)           (None, 512)               0
```

final 1.ipynb

File Edit View Insert Runtime Tools Help Last edited on April 1

Comment Share

+ Code + Text

Connect T4

```
[ ] dense (Dense)           (None, 90)                46170
    =================================================================
    Total params: 14760858 (56.31 MB)
    Trainable params: 46170 (180.35 KB)
    Non-trainable params: 14714688 (56.13 MB)
    _________________________________________________________________
```

```
[ ] model.compile(optimizer=tf.keras.optimizers.legacy.Adam(0.001),
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics=['accuracy'])
```

Train the fully-connected layer.

```
num_epochs = 10
model.fit(train_ds, validation_data=val_ds, epochs=num_epochs,
    callbacks = [
        tf.keras.callbacks.EarlyStopping(
            # Stop training when `val_loss` is no longer improving
            monitor="val_loss",
            # "no longer improving" being defined as "no better than 1e-2 less"
            min_delta=1e-2,
            # "no longer improving" being further defined as "for at least 2 epochs"
            patience=2,
            verbose=1,
            restore_best_weights=True
        )
    ]
)
```

```
Epoch 1/10
139/139 [==============================] - 2050s 14s/step - loss: 12.0196 - accuracy: 0.1103 - val_loss: 3.6143 - val_accuracy: 0.4215
Epoch 2/10
139/139 [==============================] - 39s 271ms/step - loss: 4.4257 - accuracy: 0.4095 - val_loss: 1.8529 - val_accuracy: 0.6462
Epoch 3/10
139/139 [==============================] - 40s 277ms/step - loss: 2.4863 - accuracy: 0.5848 - val_loss: 1.4264 - val_accuracy: 0.7238
Epoch 4/10
139/139 [==============================] - 39s 272ms/step - loss: 1.7417 - accuracy: 0.6787 - val_loss: 1.1470 - val_accuracy: 0.7771
Epoch 5/10
139/139 [==============================] - 40s 276ms/step - loss: 1.2029 - accuracy: 0.7545 - val_loss: 1.0872 - val_accuracy: 0.7879
Epoch 6/10
```

final 1.ipynb

File Edit View Insert Runtime Tools Help Last edited on April 1

Comment Share

+ Code + Text

Connect T4

```
Epoch 4/10
139/139 [==============================] - 39s 272ms/step - loss: 1.7417 - accuracy: 0.6787 - val_loss: 1.1470 - val_accuracy: 0.7771
Epoch 5/10
139/139 [==============================] - 40s 276ms/step - loss: 1.2029 - accuracy: 0.7545 - val_loss: 1.0872 - val_accuracy: 0.7879
Epoch 6/10
139/139 [==============================] - 39s 268ms/step - loss: 0.9501 - accuracy: 0.7888 - val_loss: 1.0172 - val_accuracy: 0.8060
Epoch 7/10
139/139 [==============================] - 38s 266ms/step - loss: 0.7603 - accuracy: 0.8154 - val_loss: 0.9441 - val_accuracy: 0.8177
Epoch 8/10
139/139 [==============================] - 39s 271ms/step - loss: 0.6951 - accuracy: 0.8278 - val_loss: 0.9469 - val_accuracy: 0.8294
Epoch 9/10
139/139 [==============================] - ETA: 0s - loss: 0.5539 - accuracy: 0.8646Restoring model weights from the end of the best epoch: 7.
139/139 [==============================] - 36s 253ms/step - loss: 0.5539 - accuracy: 0.8646 - val_loss: 0.9473 - val_accuracy: 0.8375
Epoch 9: early stopping
<keras.src.callbacks.History at 0x7a9badcdfb20>
```

If output layer is showing good result, now we unfreeze part of the VGG-16 to do fine-tuning.

```
[ ] # fine tuning
    base_model.trainable = True
    for layer in base_model.layers[:14]:
        layer.trainable = False
    model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.__operators__.getitem (  (None, 224, 224, 3)       0
 SlicingOpLambda)

 tf.nn.bias_add (TFOpLambda  (None, 224, 224, 3)       0
 )

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 global_average_pooling2d (  (None, 512)               0
 GlobalAveragePooling2D)

 dropout (Dropout)           (None, 512)               0
```

final 1.ipynb

File Edit View Insert Runtime Tools Help Last edited on April 1

Comment Share

+ Code + Text

Connect T4

```
[ ]
Total params: 14760858 (56.31 MB)
Trainable params: 7125594 (27.18 MB)
Non-trainable params: 7635264 (29.13 MB)
```

```
[ ] model.compile(optimizer=tf.keras.optimizers.Adam(0.0001), # use a lower learning rate when fine-tuning
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics=['accuracy'])
```

```
num_epochs = 5
history = model.fit(train_ds, validation_data=val_ds, epochs=num_epochs,
    callbacks = [
        tf.keras.callbacks.EarlyStopping(
            # Stop training when `val_loss` is no longer improving
            monitor="val_loss",
            # "no longer improving" being defined as "no better than 1e-2 less"
            min_delta=1e-2,
            # "no longer improving" being further defined as "for at least 2 epochs"
            patience=2,
            verbose=1,
        )
    ]
)
```

```
Epoch 1/5
139/139 [==============================] - 42s 275ms/step - loss: 0.9131 - accuracy: 0.7459 - val_loss: 0.7996 - val_accuracy: 0.8141
Epoch 2/5
139/139 [==============================] - 38s 264ms/step - loss: 0.4007 - accuracy: 0.8786 - val_loss: 0.7062 - val_accuracy: 0.8375
Epoch 3/5
139/139 [==============================] - 38s 266ms/step - loss: 0.2277 - accuracy: 0.9269 - val_loss: 0.5880 - val_accuracy: 0.8439
Epoch 4/5
139/139 [==============================] - 40s 280ms/step - loss: 0.1991 - accuracy: 0.9407 - val_loss: 0.6716 - val_accuracy: 0.8457
Epoch 5/5
139/139 [==============================] - 39s 268ms/step - loss: 0.1465 - accuracy: 0.9605 - val_loss: 0.7831 - val_accuracy: 0.8249
Epoch 5: early stopping
```

After training, choose a random picture and see its prediction.

final 1.ipynb

File Edit View Insert Runtime Tools Help Last edited on April 1

Comment Share

+ Code + Text

Connect T4

After training, choose a random picture and see its prediction.

```
[ ] img_path_list = glob(r'/content/drive/MyDrive/animals/tiger/*.jpg')
    print(img_path_list)
```

```
[ ] import random
    from glob import glob

    # Assuming multiple images are in '/content/drive/MyDrive/'
    img_path_list = glob('/content/drive/MyDrive/animals/sparrow/*.jpg')  # Modify the pattern if necessary

    if img_path_list:
        img_name = random.choice(img_path_list)
        img = tf.keras.utils.load_img(
            img_name, target_size=(img_height, img_width)
        )
        img_array = tf.keras.utils.img_to_array(img)
        img_array = tf.expand_dims(img_array, 0) # Create a batch

        predictions = model.predict(img_array)
        score = tf.nn.softmax(predictions[0])

        print(
            "{} most likely belongs to {} with a {:.2f} percent confidence."
            .format(img_name, train_ds.class_names[np.argmax(score)], 100 * np.max(score))
        )

        plt.figure(figsize=(2, 2))
        plt.imshow((img_array[0].numpy()).astype('uint8'))
        plt.title("{}:{:.2f}".format(train_ds.class_names[np.argmax(score)], 100 * np.max(score)))
        plt.axis('off')
        plt.show()
```

```
1/1 [==============================] - 0s 18ms/step
/content/drive/MyDrive/animals/sparrow/5e41b6ffb1.jpg most likely belongs to sparrow with a 100.00 percent confidence.
```

final 1.ipynb

File Edit View Insert Runtime Tools Help Last edited on April 1

Comment Share

+ Code + Text

Connect T4

```
[ ]
    if img_path_list:
        img_name = random.choice(img_path_list)
        img = tf.keras.utils.load_img(
            img_name, target_size=(img_height, img_width)
        )
        img_array = tf.keras.utils.img_to_array(img)
        img_array = tf.expand_dims(img_array, 0) # Create a batch

        predictions = model.predict(img_array)
        score = tf.nn.softmax(predictions[0])

        print(
            "{} most likely belongs to {} with a {:.2f} percent confidence."
            .format(img_name, train_ds.class_names[np.argmax(score)], 100 * np.max(score))
        )

        plt.figure(figsize=(2, 2))
        plt.imshow((img_array[0].numpy()).astype('uint8'))
        plt.title("{}:{:.2f}".format(train_ds.class_names[np.argmax(score)], 100 * np.max(score)))
        plt.axis('off')
        plt.show()
```

```
1/1 [==============================] - 0s 18ms/step
/content/drive/MyDrive/animals/sparrow/5e41b6ffb1.jpg most likely belongs to sparrow with a 100.00 percent confidence.
```

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENTS

In conclusion, the development of a robust system for wildlife detection and monitoring is crucial for conservation efforts, habitat management, and biodiversity research. By integrating advanced technologies such as machine learning, sensor networks, and data analytics, we can enhance our ability to monitor wildlife populations effectively and make informed decisions to protect and preserve natural ecosystems.

The implementation of an automated species detection module within a well-defined system architecture provides several key benefits:

Efficiency: Automation reduces the need for manual labor in data analysis and enables real-time or near-real-time monitoring, allowing for timely interventions when necessary.

Accuracy: Machine learning algorithms can achieve high levels of accuracy in species detection, even in complex and diverse environments, improving the reliability of monitoring data.

Scalability: The modular design of the system architecture facilitates scalability, enabling the integration of additional sensors, data sources, and analysis tools as monitoring needs evolve.

Insights: Data analytics and visualization tools provide valuable insights into wildlife populations, habitat dynamics, and ecosystem health, supporting evidence-based decision-making and adaptive management strategies.

Collaboration: By providing APIs and interfaces for integration with external systems, the wildlife detection and monitoring system can facilitate collaboration among researchers, conservation organizations, and government agencies. Looking to the future, there are several avenues for further development and enhancement of wildlife detection and monitoring systems:

Improving Accuracy: Continued research and development in machine learning and computer vision can lead to further improvements in the accuracy and efficiency of species detection algorithms.

Integration of Emerging Technologies: Incorporating emerging technologies such as drones, remote sensing, and IoT devices can expand the capabilities of wildlife monitoring

systems and provide additional insights into animal behavior and habitat dynamics.

Data Sharing and Collaboration: Promoting data sharing and collaboration among different stakeholders can enhance the effectiveness of wildlife monitoring efforts and facilitate the exchange of knowledge and best practices.

Incorporating Citizen Science: Engaging citizen scientists in data collection and analysis efforts can augment existing monitoring programs and provide valuable data at larger spatial and temporal scales.

Addressing Ethical and Privacy Concerns: As wildlife monitoring systems become more widespread, it is important to address ethical and privacy concerns related to data collection, storage, and use, ensuring that the rights and welfare of animals and local communities are respected.

Overall, the ongoing advancement and refinement of wildlife detection and monitoring technologies hold great promise for improving our understanding of ecosystems and biodiversity and supporting conservation efforts in the face of environmental challenges.

CHAPTER 7

REFERENCES

1. Krebs, C. J. (2009). Ecology: The Experimental Analysis of Distribution and Abundance. Benjamin Cummings.

2. Sinclair, A. R. E., Mduma, S., & Brashares, J. S. (2003). Patterns of predation in a diverse predator-prey system. Nature, 425(6955), 288–290. https://doi.org/10.1038/nature01934

3. Pulliam, H. R. (2000). On the relationship between niche and distribution. Ecology Letters, 3(4), 349–361. https://doi.org/10.1046/j.1461-0248.2000.00143.x

4. Gonzalez, R. C., & Woods, R. E. (2008). Digital Image Processing (3rd ed.). Prentice Hall.

5. Szeliski, R. (2010). Computer Vision: Algorithms and Applications. Springer. https://doi.org/10.1007/

6. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. http://www.deeplearningbook.org/

7. Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer. https://doi.org/10.1007/978-0-387-31073-2

8. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436–444. https://doi.org/10.1038/nature14539

9. Fraser, D., & MacRae, A. M. (2011). Pain and suffering in invertebrates? ILAR Journal, 52(2), 175–184. https://doi.org/10.1093/ilar.52.2.175

10. Redford, K. H., & Taber, A. (2000). Writing the wrongs: Developing a safe-fail culture in conservation. Conservation Biology, 14(6), 1567–1568. https://doi.org/10.1111/j.1523-1739.2000.99399.x

11. The IUCN Red List of Threatened Species. (n.d.). Retrieved from https://www.iucnredlist.org/

12. Primack, R. B. (2010). Essentials of Conservation Biology (5th ed.). Sinauer Associates.

13. Sutherland, W. J., Pullin, A. S., Dolman, P. M., & Knight, T. M. (2004). The need for evidence-based conservation. Trends in Ecology & Evolution, 19(6), 305–308. https://doi.org/10.1016/j.tree.2004.03.018

14. Millennium Ecosystem Assessment. (2005). Ecosystems and Human Well-being: Biodiversity Synthesis. World Resources Institute.

15. Berkes, F. (2009). Evolution of co-management: Role of knowledge generation, bridging organizations and social learning. Journal of Environmental Management, 90(5), 1692–1702. https://doi.org/10.1016/j.jenvman.2008.12.001

16. Reed, M. S., Stringer, L. C., Fazey, I., Evely, A. C., & Kruijsen, J. H. J. (2014). Five principles for the practice of knowledge exchange in environmental management. Journal of Environmental Management, 146, 337–345. https://doi.org/10.1016/j.jenvman.2014.07.021

17. Ostrom, E. (1990). Governing the Commons: The Evolution of Institutions for Collective Action. Cambridge University Press.

18. H. Nguyen et al., "Animal Recognition and Identification with Deep Convolutional Neural Networks for Automated Wildlife Monitoring," 2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA), Tokyo, Japan, 2017, pp. 40-49, doi: 10.1109/DSAA.2017.31. keywords: {Wildlife;Cameras;Monitoring;Automation;Australia;deep learning;convolutional neural networks;large scale image classification;animal recognition;wildlife monitoring;citizen science},