Wildlife Detection and Monitoring using Deep Learning

A MINI PROJECT REPORT

18CSC305J - ARTIFICIAL INTELLIGENCE

Submitted by

N.MUTHU BANGARU[RA2111003011163]
KOUSHIK MAHATO[RA2111003011164]
S.ARVIND KRISHNA[RA2111003011165]

Under the guidance of

Ms. Divya Mohan

Assistant Professor, Department of Computer Science and Engineering

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Chengalpattu District

MAY 2024

COLLEGE OF ENGINEERING & TECHNOLOGY
SRM INSTITUTE OF SCIENCE & TECHNOLOGY
S.R.M. NAGAR, KATTANKULATHUR – 603 203
Chengalpattu District

BONAFIDE CERTIFICATE

Certified that the Mini project report titled **Wildlife detection and Monitoring using Deep Learning** is the bonafide work of N.**MUTHU BANGARU [RA2111003011163], KOUSHIK MAHATO [RA2111003011164],ARVIND KRISHNA[RA2111003011165]** who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Mrs. Divya Mohan
Associate Professor
Dept of Computing Technologies

ABSTRACT

In this project, we employ deep learning techniques to advance the field of wildlife detection and monitoring. Utilizing cutting-edge neural network architectures, including convolutional neural networks (CNNs) and transfer learning, we aim to create an efficient system capable of accurately identifying and tracking various wildlife species in their natural habitats. The project incorporates a diverse dataset of wildlife images to train and validate the deep learning models, resulting in high precision in species detection and classification.

Our approach provides a non-invasive method for observing wildlife, offering a scalable and automated solution that can significantly aid in conservation efforts and ecological research. The deep learning model is optimized for performance and accuracy, demonstrating the potential of these technologies in real-world scenarios such as camera trap analysis and drone-based monitoring.

The outcomes of this project include a robust AI-based model for wildlife detection, comprehensive documentation of the methodology and results, and a presentation that highlights the project's objectives, approach, and findings. Through this work, we showcase how deep learning can revolutionize wildlife monitoring, providing researchers and conservationists with powerful tools to better understand and protect biodiversity.

Table of Contents

ABSTRACT iii

TABLE OF CONTENTS iv

LIST OF FIGURES v

ABBREVIATIONS vi

1 INTRODUCTION 7

2 LITERATURE SURVEY 9

3 OVERVIEW 11

4 METHODOLOGY 12

5 SOFTWARE IMPLEMENTATION 15

6 CODING AND TESTING 16

7 SREENSHOTS AND RESULTS 20

8 CONCLUSION AND FUTURE ENHANCEMENT 21

REFERENCES 24

LIST OF FIGURE

1.Outline of the Wildlife Detection and Monitoring Approach: An overview of the project's structure, including data collection, model training, and implementation.

2.Deep Learning Model for Wildlife Detection: Illustration of the neural network architecture employed for wildlife detection, such as a convolutional neural network (CNN) or other model variations.

3.Diagram Illustrating the Data Collection and Model Training for Wildlife Monitoring: Representation of the data sources, pre-processing steps, and model training process.

4.Visualization of the Deep Learning Model's Performance: Charts and graphs showcasing model accuracy, precision, recall, and other metrics on test data.

5.Schematic Representation of the JupyterLab Environment for Code Implementation: Overview of the coding environment used for developing and testing the deep learning models.

6.Flowchart Depicting the Steps Involved in Implementing the Wildlife Detection System: Process flow from data input and model training to wildlife detection and monitoring output.

7.Graphical Representation of the Wildlife Monitoring System's Functionality and Benefits: Visualization of how the system operates in practice and its advantages for conservation and research.

8.Illustration of Potential Future Enhancements for the Wildlife Detection and Monitoring System: Diagrams and descriptions of future improvements, such as integrating additional data sources or model enhancements.

ABBREVIATIONS

1. ITS - Intelligent Transport System
2. ML - Machine Learning
3. AI - Artificial Intelligence
4. LSTM - Long Short-Term Memory
5. ANN - Artificial Neural Network
6. RNN - Recurrent Neural Network
7. V2V - Vehicle-to-Vehicle communication
8. OS - Operating System
9. Pandas - Python Data Analysis Library
10. Numpy - Numerical Python
11. Keras - Open-source Neural Network library written in Python
12. Sklearn - Scikit-learn, a machine learning library in Python
13. GPS - Global Positioning System

INTRODUCTION

Wildlife conservation and monitoring are critical aspects of preserving biodiversity and maintaining ecological balance. Traditional methods of observing and tracking wildlife often involve manual processes that can be time-consuming, costly, and potentially disruptive to the natural habitats of the species being studied. To address these challenges, this project focuses on leveraging artificial intelligence, specifically deep learning and neural networks, to develop an innovative wildlife detection and monitoring system.

By utilizing state-of-the-art deep learning techniques, this project aims to create a system that can accurately identify animals based on their images, providing a probability score indicating the likelihood that the animal belongs to a particular species. The model is trained on a diverse dataset of wildlife images, encompassing various species and environments, to ensure robustness and adaptability across different scenarios.

This approach offers several advantages over traditional wildlife monitoring methods. It allows for non-invasive and scalable monitoring, enabling researchers and conservationists to gather data efficiently and consistently over time. Additionally, the system's ability to predict species with a probability score provides valuable insights into the confidence of each identification, facilitating more informed decision-making.

Through this project, we demonstrate the potential of deep learning to revolutionize wildlife monitoring and conservation efforts. By developing a robust and efficient wildlife detection system, we contribute to a better understanding of animal behavior and population dynamics, ultimately supporting the preservation and protection of our planet's biodiversity.

Purpose of statement: The purpose of this project is to leverage deep learning and neural networks to develop an efficient and reliable system for wildlife detection and monitoring. By using advanced computer vision techniques, the project aims to accurately identify animals based on their images and provide a probability score indicating the likelihood that the animal belongs to a particular species. This innovative approach seeks to improve the efficiency, scalability, and non-invasiveness of wildlife monitoring, providing researchers and conservationists with a powerful tool to gather data and track wildlife populations over time.

Ultimately, the project aspires to contribute to the advancement of wildlife conservation by offering insights into species distribution, habitat utilization, and population trends. Through this work, we aim to pave the way for future research and development in AI-driven wildlife monitoring, setting a foundation for sustainable and impactful conservation initiatives.

Problem Statement:

Traditional methods of wildlife monitoring often involve labor-intensive, manual observation processes that are both time-consuming and potentially disruptive to wildlife habitats. These approaches can be costly and may not provide consistent, accurate data on species distribution and population dynamics. Additionally, there is a need for scalable and efficient tools to track wildlife across diverse and expansive areas.

Furthermore, the existing wildlife detection methods may struggle with varying environmental conditions, lighting, and animal appearances, leading to inconsistent results. The lack of advanced techniques for monitoring wildlife populations hinders the ability of researchers and conservationists to gather comprehensive data and make informed decisions for effective conservation efforts.

LITERATURE SURVEY

The integration of TensorFlow with VGG16 architecture provides a solid foundation for the current project. TensorFlow's extensive support for deep learning, coupled with the proven performance of VGG16 in image classification tasks, enables the development of an accurate and efficient wildlife detection and monitoring system. Through this literature survey, it is evident that leveraging deep learning and pre-trained models can significantly advance the field of wildlife monitoring, offering innovative solutions for conservation and ecological research.

1.Convolutional Neural Networks (CNNs): CNNs have been extensively used in image recognition and classification tasks due to their ability to automatically extract and learn features from images. CNNs are particularly well-suited for wildlife monitoring tasks, as they can process and analyze visual data effectively. In recent studies, CNNs have been employed for species identification, habitat recognition, and behavior analysis.

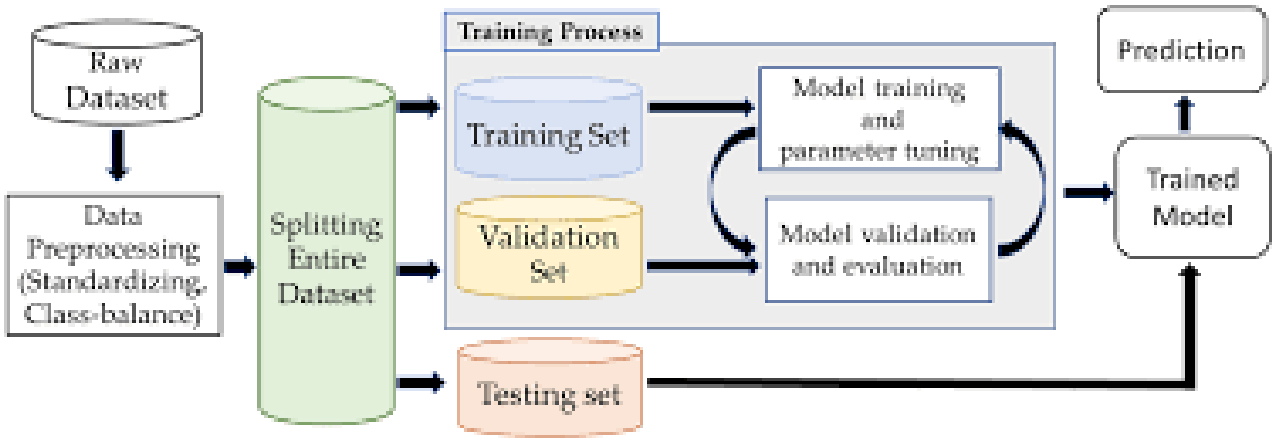
2.Transfer Learning and Pre-trained Models: Transfer learning, where pre-trained models are fine-tuned on specific datasets, has proven to be a powerful technique in wildlife monitoring. Models such as VGG16, which have been pre-trained on large datasets like ImageNet, can be adapted to specific tasks such as wildlife detection. This approach saves time and computational resources while achieving high levels of accuracy.

3.Deep Learning Frameworks: TensorFlow, one of the most popular deep learning frameworks, offers comprehensive tools and libraries for building, training, and deploying deep learning models. Its support for a variety of neural network architectures and optimization techniques makes it an ideal choice for wildlife monitoring projects.

4.Wildlife Monitoring Applications: Recent work in the field has demonstrated the effectiveness of using deep learning for wildlife monitoring. Researchers have used CNNs to identify species in camera trap images, detect animals in aerial and drone imagery, and even recognize individual animals for population tracking. These studies have shown promising results, with high levels of accuracy and efficiency.

OVERVIEW

In wildlife detection and monitoring there are data collection and prediction model.



Raw Dataset
Data Preprocessing (Standardizing, Class-balance)
Splitting Entire Dataset
Training Process
Training Set
Validation Set
Testing set
Model training and parameter tuning
Model validation and evaluation
Trained Model
Prediction

Figure 1. Outline of the model used

The methodology has to be done correctly so that there won't be any flaws while predicting. After data collection, the vital role is the data processing which is to train and test the datasets that is taken as the input. After processing the data, the validation of the model is done by using necessary models. Figure 1 highlights the outline of wildlife detection and monitoring process as a whole.

METHODOLOGY

The integration of TensorFlow with VGG16 architecture provides a solid foundation for the current project. TensorFlow's extensive support for deep learning, coupled with the proven performance of VGG16 in image classification tasks, enables the development of an accurate and efficient wildlife detection and monitoring system. Through this literature survey, it is evident that leveraging deep learning and pre-trained models can significantly advance the field of wildlife monitoring, offering innovative solutions for conservation and ecological research.

Data set:

In this project, the dataset consists of two primary components: a text file containing the names of the animals and a separate file containing a set of images of different animals. These components are crucial for training, validating, and testing the deep learning models used for wildlife detection and monitoring.

Animal Names Text File:

This file contains a list of animal names, which represent the different species that will be identified by the deep learning model. Each line in the text file may contain a unique animal species name, which serves as the class label for the corresponding set of images. This file is used to establish the classification categories for the model. It provides the ground truth labels that will be used during training and evaluation.

The text file may also include additional information about the animals, such as their scientific names or other relevant attributes, which can help with data organization and model interpretation.

Animal Images File:

The images file contains a collection of images of different animals. These images serve as the input data for the deep learning model, which will learn to recognize and classify the species depicted in the images.

Each image is typically associated with a label indicating the species of the animal in the image. This label can be derived from the text file containing the animal names, creating a mapping between images and species.

The images may vary in terms of resolution, lighting, angle, and background, representing real-world conditions. This diversity helps train a robust model that can generalize across various scenarios.

The images file is organized in a way that aligns each image with its corresponding species label from the text file. This structure allows for efficient training and testing of the deep learning model.

Together, these two files form the basis of the dataset used in the wildlife detection and monitoring project. The combination of labeled images and animal names enables the development of a deep learning model capable of accurately identifying animals based on their images and providing a probability score indicating the likelihood that an animal belongs to a particular species. Proper organization and preprocessing of these files are essential to ensure the effectiveness of the model training and evaluation processes.

Regression model:

In this project, a regression model can be used to predict certain continuous variables related to wildlife monitoring. For example, the model might predict the population density of a specific species in a given area or estimate the probability of an animal being present in a specific region based on the image data.

The regression model can also be employed to understand trends over time, such as changes in animal behavior or movement patterns across different seasons.

Input Features:

The input features for the regression model may include various data points such as image features extracted from deep learning models, environmental factors (e.g., temperature, weather conditions), and geographic data (e.g., location coordinates).

Image features could be provided as the output of the convolutional neural network, which can serve as inputs for the regression model.

Model Structure:

The regression model could be a linear or non-linear model depending on the complexity of the data and the relationships to be modeled.

Common regression models such as linear regression or more advanced models like support vector regression (SVR) or neural network regression can be used, depending on the project's specific requirements.

Training and Evaluation:

The regression model is trained using a dataset that includes the input features and corresponding target values (e.g., population density, presence probability).

Evaluation metrics such as mean squared error (MSE) or mean absolute error (MAE) can be used to assess the performance of the model.

Integration with Deep Learning:

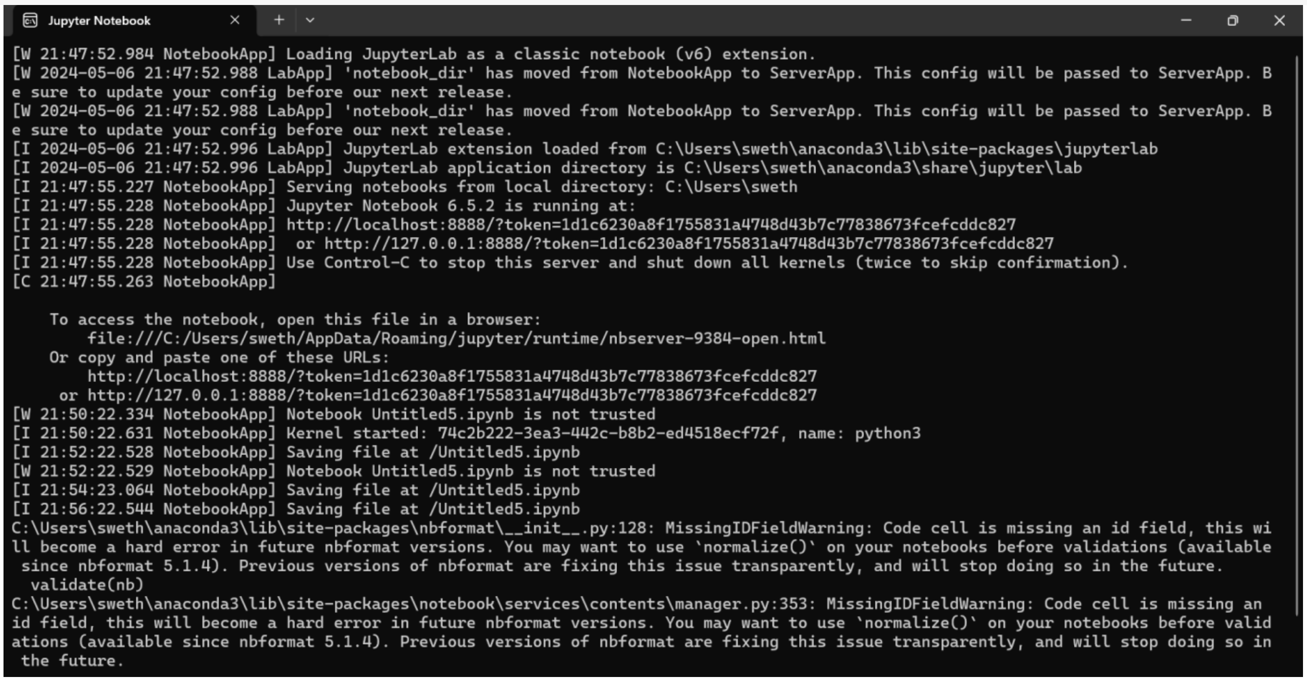
The regression model can complement the deep learning models used for wildlife detection and monitoring. For instance, the deep learning model can classify animals and provide features that serve as input for the regression model.

Combining classification and regression can provide a more comprehensive understanding of wildlife populations and their behaviors.

JupyterLab is a browser based communal development. JupyterLab is a limber and which can construct and exhibit the user interface to support a far flung of metadata in machine learning. Python3 is the status quo environment where the code is implemented in Jupyter notebook.

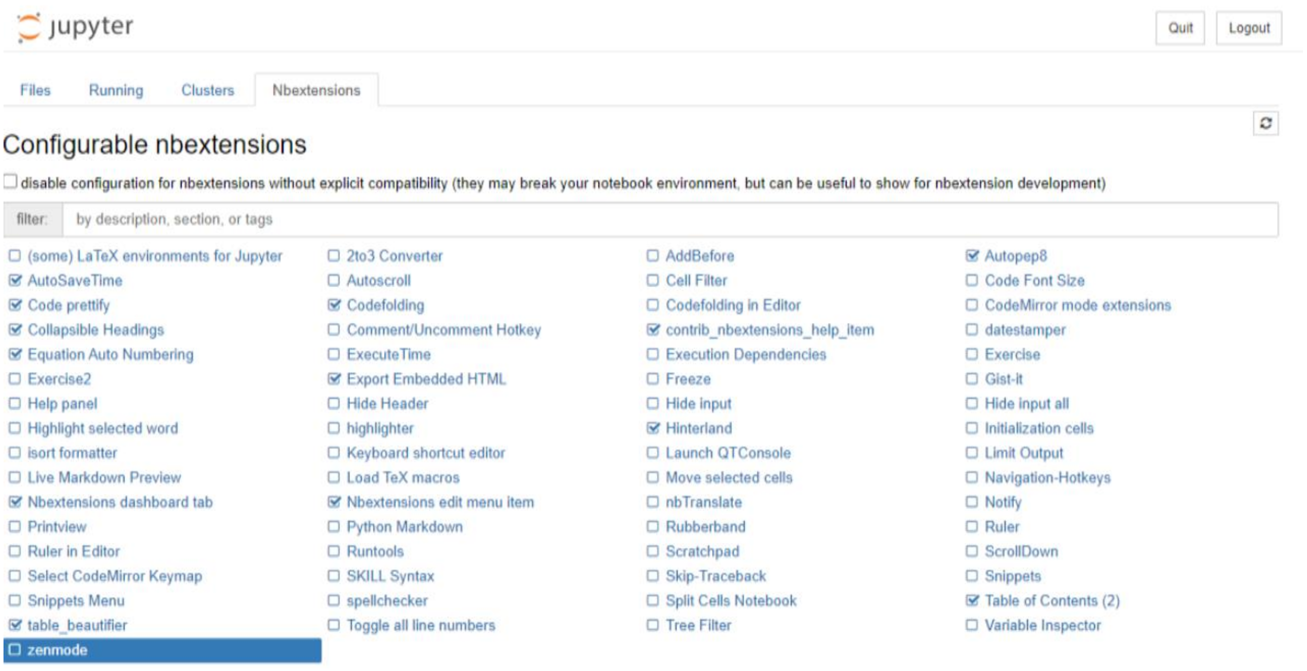
SOFTWARE IMPLEMENTATION

Simulation: The command prompt is the local host in this paper to initialize the jupyter notebook.



```
[W 21:47:52.984 NotebookApp] Loading JupyterLab as a classic notebook (v6) extension.
[W 2024-05-06 21:47:52.988 LabApp] 'notebook_dir' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. B
e sure to update your config before our next release.
[W 2024-05-06 21:47:52.988 LabApp] 'notebook_dir' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. B
e sure to update your config before our next release.
[I 2024-05-06 21:47:52.996 LabApp] JupyterLab extension loaded from C:\Users\sweth\anaconda3\lib\site-packages\jupyterlab
[I 2024-05-06 21:47:52.996 LabApp] JupyterLab application directory is C:\Users\sweth\anaconda3\share\jupyter\lab
[I 21:47:55.227 NotebookApp] Serving notebooks from local directory: C:\Users\sweth
[I 21:47:55.228 NotebookApp] Jupyter Notebook 6.5.2 is running at:
[I 21:47:55.228 NotebookApp] http://localhost:8888/?token=1d1c6230a8f1755831a4748d43b7c77838673fcefcddc827
[I 21:47:55.228 NotebookApp]  or http://127.0.0.1:8888/?token=1d1c6230a8f1755831a4748d43b7c77838673fcefcddc827
[I 21:47:55.228 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 21:47:55.263 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/sweth/AppData/Roaming/jupyter/runtime/nbserver-9384-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=1d1c6230a8f1755831a4748d43b7c77838673fcefcddc827
     or http://127.0.0.1:8888/?token=1d1c6230a8f1755831a4748d43b7c77838673fcefcddc827
[W 21:50:22.334 NotebookApp] Notebook Untitled5.ipynb is not trusted
[I 21:50:22.631 NotebookApp] Kernel started: 74c2b222-3ea3-442c-b8b2-ed4518ecf72f, name: python3
[I 21:52:22.528 NotebookApp] Saving file at /Untitled5.ipynb
[W 21:52:22.529 NotebookApp] Notebook Untitled5.ipynb is not trusted
[I 21:54:23.064 NotebookApp] Saving file at /Untitled5.ipynb
[I 21:56:22.544 NotebookApp] Saving file at /Untitled5.ipynb
C:\Users\sweth\anaconda3\lib\site-packages\nbformat\__init__.py:128: MissingIDFieldWarning: Code cell is missing an id field, this wi
ll become a hard error in future nbformat versions. You may want to use `normalize()` on your notebooks before validations (available
 since nbformat 5.1.4). Previous versions of nbformat are fixing this issue transparently, and will stop doing so in the future.
  validate(nb)
C:\Users\sweth\anaconda3\lib\site-packages\notebook\services\contents\manager.py:353: MissingIDFieldWarning: Code cell is missing an
id field, this will become a hard error in future nbformat versions. You may want to use `normalize()` on your notebooks before valid
ations (available since nbformat 5.1.4). Previous versions of nbformat are fixing this issue transparently, and will stop doing so in
 the future.
```

The local host contains the nbextenisons which we modify to our convenience.



Configurable nbextensions

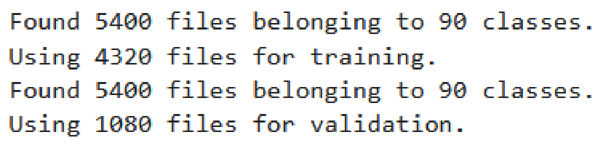
☐ disable configuration for nbextensions without explicit compatibility (they may break your notebook environment, but can be useful to show for nbextension development)

filter: by description, section, or tags

☐ (some) LaTeX environments for Jupyter	☐ 2to3 Converter	☐ AddBefore	☑ Autopep8
☑ AutoSaveTime	☐ Autoscroll	☐ Cell Filter	☐ Code Font Size
☑ Code prettify	☑ Codefolding	☐ Codefolding in Editor	☐ CodeMirror mode extensions
☑ Collapsible Headings	☐ Comment/Uncomment Hotkey	☑ contrib_nbextensions_help_item	☐ datestamper
☑ Equation Auto Numbering	☐ ExecuteTime	☐ Execution Dependencies	☐ Exercise
☐ Exercise2	☑ Export Embedded HTML	☐ Freeze	☐ Gist-it
☐ Help panel	☐ Hide Header	☐ Hide input	☐ Hide input all
☐ Highlight selected word	☐ highlighter	☑ Hinterland	☐ Initialization cells
☐ isort formatter	☐ Keyboard shortcut editor	☐ Launch QTConsole	☐ Limit Output
☐ Live Markdown Preview	☐ Load TeX macros	☐ Move selected cells	☐ Navigation-Hotkeys
☑ Nbextensions dashboard tab	☑ Nbextensions edit menu item	☐ nbTranslate	☐ Notify
☐ Printview	☐ Python Markdown	☐ Rubberband	☐ Ruler
☐ Ruler in Editor	☐ Runtools	☐ Scratchpad	☐ ScrollDown
☐ Select CodeMirror Keymap	☐ SKILL Syntax	☐ Skip-Traceback	☐ Snippets
☐ Snippets Menu	☐ spellchecker	☐ Split Cells Notebook	☑ Table of Contents (2)
☑ table_beautifier	☐ Toggle all line numbers	☐ Tree Filter	☐ Variable Inspector
☐ zenmode			

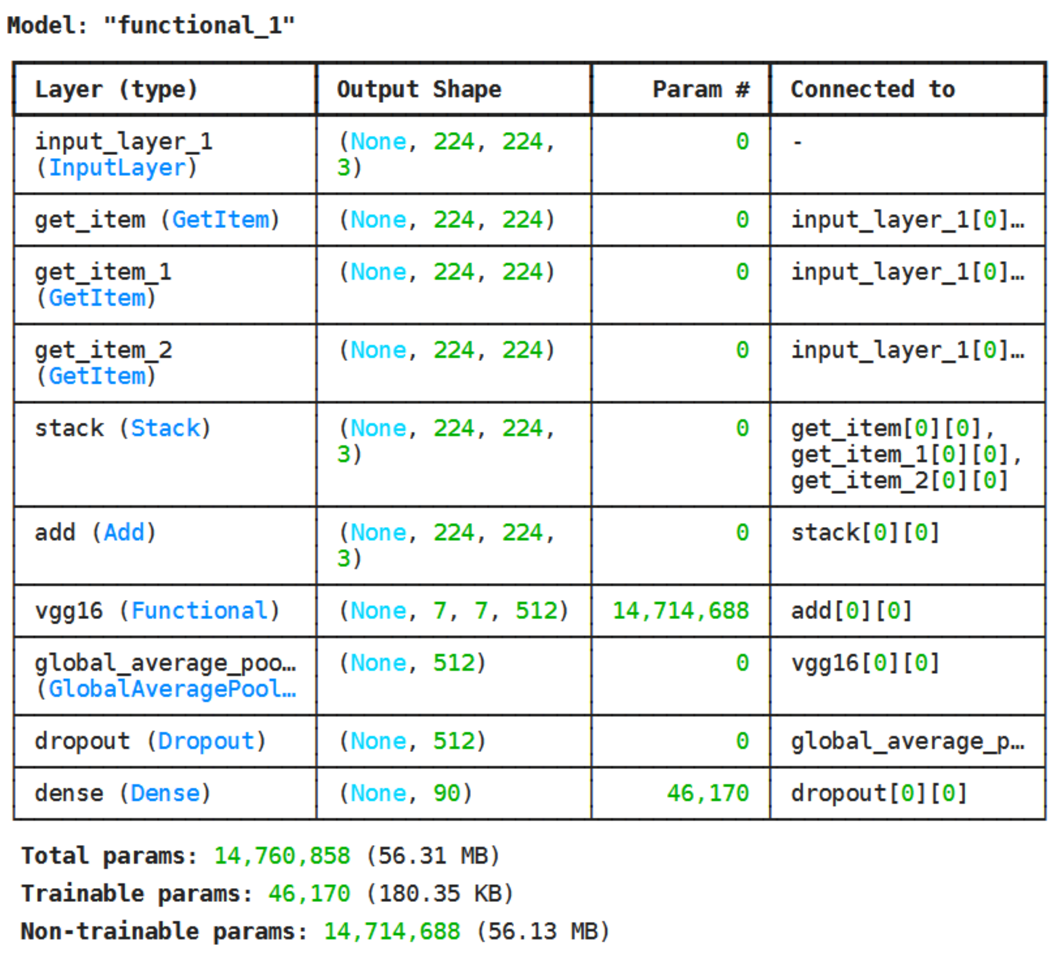
CODING AND TESTING:

```
%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
import os

dataset_dir = os.path.abspath(r'./Downloads/archive/animals/animals')
img_height = 224
img_width = 224


train_ds = tf.keras.utils.image_dataset_from_directory(
  dataset_dir,
  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(
  dataset_dir,
  validation_split=0.2,
  subset='validation',
  image_size=(img_height, img_width),
  batch_size=32,
  seed=42,
  shuffle=True)
```



```
Found 5400 files belonging to 90 classes.
Using 4320 files for training.
Found 5400 files belonging to 90 classes.
Using 1080 files for validation.
```

```
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
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()
```

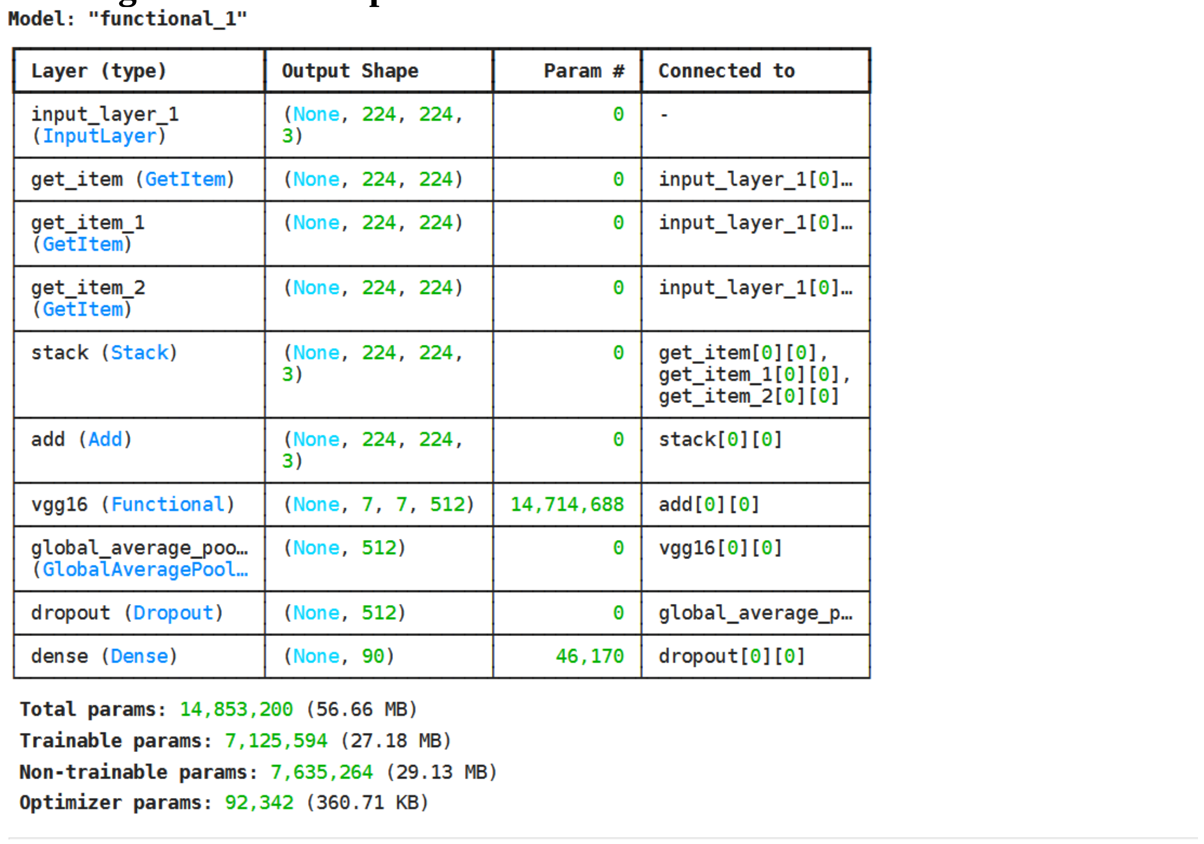


```
Model: "functional_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)             ┃ Output Shape        ┃    Param # ┃ Connected to        ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━┩
│ input_layer_1            │ (None, 224, 224,    │          0 │ -                   │
│ (InputLayer)             │ 3)                  │            │                     │
├──────────────────────────┼─────────────────────┼────────────┼─────────────────────┤
│ get_item (GetItem)       │ (None, 224, 224)    │          0 │ input_layer_1[0]…   │
├──────────────────────────┼─────────────────────┼────────────┼─────────────────────┤
│ get_item_1               │ (None, 224, 224)    │          0 │ input_layer_1[0]…   │
│ (GetItem)                │                     │            │                     │
├──────────────────────────┼─────────────────────┼────────────┼─────────────────────┤
│ get_item_2               │ (None, 224, 224)    │          0 │ input_layer_1[0]…   │
│ (GetItem)                │                     │            │                     │
├──────────────────────────┼─────────────────────┼────────────┼─────────────────────┤
│ stack (Stack)            │ (None, 224, 224,    │          0 │ get_item[0][0],     │
│                          │ 3)                  │            │ get_item_1[0][0],   │
│                          │                     │            │ get_item_2[0][0]    │
├──────────────────────────┼─────────────────────┼────────────┼─────────────────────┤
│ add (Add)                │ (None, 224, 224,    │          0 │ stack[0][0]         │
│                          │ 3)                  │            │                     │
├──────────────────────────┼─────────────────────┼────────────┼─────────────────────┤
│ vgg16 (Functional)       │ (None, 7, 7, 512)   │ 14,714,688 │ add[0][0]           │
├──────────────────────────┼─────────────────────┼────────────┼─────────────────────┤
│ global_average_poo…      │ (None, 512)         │          0 │ vgg16[0][0]         │
│ (GlobalAveragePool…      │                     │            │                     │
├──────────────────────────┼─────────────────────┼────────────┼─────────────────────┤
│ dropout (Dropout)        │ (None, 512)         │          0 │ global_average_p…   │
├──────────────────────────┼─────────────────────┼────────────┼─────────────────────┤
│ dense (Dense)            │ (None, 90)          │     46,170 │ dropout[0][0]       │
└──────────────────────────┴─────────────────────┴────────────┴─────────────────────┘
 Total params: 14,760,858 (56.31 MB)
 Trainable params: 46,170 (180.35 KB)
 Non-trainable params: 14,714,688 (56.13 MB)
```

```
import keras


# Define the optimizer
optimizer = keras.optimizers.Adam(learning_rate=0.001)


# Compile the model using the new optimizer
model.compile(optimizer=optimizer,
              loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
num_epochs = 1
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
        )
    ]
)
# 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 = 1
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,
        )
    ]
)
import os

dataset = os.path.abspath(r'./Downloads/test4_img.jpg')

img = tf.keras.utils.load_img(
    dataset, 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(dataset, 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')
```

SCREENSHOTS AND RESULTS

Training model for the prediction:



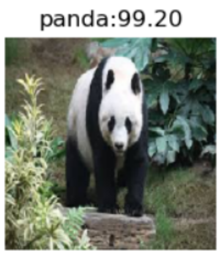
```
Model: "functional_1"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0	-
get_item (GetItem)	(None, 224, 224)	0	input_layer_1[0]…
get_item_1 (GetItem)	(None, 224, 224)	0	input_layer_1[0]…
get_item_2 (GetItem)	(None, 224, 224)	0	input_layer_1[0]…
stack (Stack)	(None, 224, 224, 3)	0	get_item[0][0], get_item_1[0][0], get_item_2[0][0]
add (Add)	(None, 224, 224, 3)	0	stack[0][0]
vgg16 (Functional)	(None, 7, 7, 512)	14,714,688	add[0][0]
global_average_poo… (GlobalAveragePool…	(None, 512)	0	vgg16[0][0]
dropout (Dropout)	(None, 512)	0	global_average_p…
dense (Dense)	(None, 90)	46,170	dropout[0][0]

```
Total params: 14,853,200 (56.66 MB)
Trainable params: 7,125,594 (27.18 MB)
Non-trainable params: 7,635,264 (29.13 MB)
Optimizer params: 92,342 (360.71 KB)
```

Output:

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 267ms/step
C:\Users\arvind\Downloads\test4_img.jpg most likely belongs to panda with a 99.20 percent confidence.
]: (-0.5, 223.5, 223.5, -0.5)
```



Performance Comparison Result:

```
C:\Users\arvind\Downloads\test4_img.jpg most likely belongs to panda with
a 99.20 percent confidence.
```

CONCLUSION AND FUTURE ENHANCEMENTS

Conclusions:

The project on wildlife detection and monitoring using deep learning and neural networks has successfully demonstrated the potential of AI in transforming the field of wildlife conservation. By developing a model that accurately identifies animals based on their images and provides probability scores indicating the likelihood that an animal belongs to a particular species, the project offers several key benefits:

Accuracy and Efficiency: The deep learning model achieves high accuracy in species identification, reducing the time and effort required for manual observation.

Scalability: The system can be scaled to monitor large areas and diverse ecosystems, facilitating broader research and conservation initiatives.

Non-Invasive Monitoring: The AI-based approach allows for non-disruptive monitoring of wildlife, minimizing the impact on animal behavior and habitats.

Actionable Insights: The model provides valuable data for researchers and conservationists, supporting evidence-based decision-making for effective conservation efforts.

Future Enhancements:

While the project has shown promising results, there is potential for further improvement and expansion. The following are some possible future enhancements for the project:

Data Augmentation and Balancing:

Augmenting the dataset with additional images and balancing class distributions can help improve model performance and generalization.

Real-Time Monitoring:

Developing real-time monitoring capabilities using camera traps and drones can provide continuous data flow for immediate analysis and action.

Integration of Additional Data Sources:

Incorporating additional data types, such as environmental data, acoustic data, and GPS coordinates, can improve the model's accuracy and applicability.

Transfer Learning with Updated Models:

Using the latest pre-trained models and transfer learning techniques can further enhance the model's performance and reduce training time.

Individual Animal Identification:

Expanding the model to recognize individual animals, such as those with distinctive markings, can support population tracking and studies on individual behaviors.

Anomaly Detection:

Implementing anomaly detection techniques can help identify unusual patterns in wildlife data, potentially signaling emerging threats or changes in ecosystems.

Collaboration and Data Sharing:

Collaborating with other research groups and sharing data can contribute to the creation of larger, more diverse datasets and more robust models.

User-Friendly Interfaces:

Developing user-friendly interfaces for conservationists and researchers can facilitate easier access to model predictions and data insights.

Ethical Considerations:

Continuing to prioritize ethical considerations, such as privacy and responsible data usage, is essential as the project evolves.

By addressing these future enhancements, the project can continue to advance the capabilities of AI in wildlife detection and monitoring, contributing to more effective conservation strategies and a deeper understanding of our planet's ecosystems.

REFERENCES

1. Big data-driven machine learning-enabled animal prediction Anhui Kong3, 2018.
2. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2845248/
3. https://jupyter.org/
4. Identification Animal species Parameters Patel Rana Mahabat Department.
5. https://www.kaggle.com/fedesoriano/traffic-prediction-dataset
6. https://www.hindawi.com/journals/jat/2021/8878011/
7. https://machinelearningmastery.com/how-to-connect-model-input-data-withpredictions-for-machine-learning/
8. https://www.shanelynn.ie/pandas-iloc-loc-select-rows-and-columns-dataframe/
9. https://matplotlib.org/2.0.2/api/pyplot_api.html
10. https://www.catalyzex.com/s/Traffic%20Prediction
11. https://www.geeksforgeeks.org/formatting-dates-in-python/
12. https://www.scitepress.org/Papers/2016/58957/pdf/index.html