A close up of a sign

Description automatically generated

DISTRIBUTED SYSTEMS (CSE -3261) MINI PROJECT REPORT ON

**Wildlife Monitoring System: A Distributed Approach**

SUBMITTED TO

**Department of Computer Science & Engineering**

By

Moksha Deepak Kothari

210905017

Roll Number 4

VI Semester  
Section A

Vaibhav Sandhir

210905152

Roll Number 24

VI Semester  
Section A

Sujeet Amberkar

200905092

Roll Number 1

VI Semester  
Section A

Name & Signature of Evaluator 1 Name & Signature of Evaluator 2

Dr. Venkatesh A Bhandage Mr. Vinayak Pai

(Jan 2024 – May 2024)

|  |  |  |
| --- | --- | --- |
| **Table of Contents** | | |
|  | | Page No |
|  | | |
| **Chapter 1** | INTRODUCTION | 3 |
| **Chapter 2** | BACKGROUND THEORY | 3 |
| **Chapter 3** | METHODOLOGY | 4 |
| **Chapter 4** | RESULTS AND DISCUSSION | 21 |
| **Chapter 5** | CONCLUSIONS AND FUTURE ENHANCEMENTS | 22 |
| **REFERENCES** | | 23 |

# **1. INTRODUCTION**

Wildlife conservation and monitoring presents numerous challenges, especially with increasing environmental concerns such as climate change, illegal hunting, and poaching. Despite the establishment of conservation areas such as national parks, these challenges persist, demanding an innovative solution.

A major challenge in preserving wildlife is being able to track and monitor animal populations. Traditional methods such as placing trackers are often impractical. As a result, there is a pressing need for an innovative wildlife monitoring system that can overcome these limitations.

One approach involves capturing images at different sites in the national park for a census. By placing cameras strategically around the park, conservationists can assess the population efficiently. Though not perfect, this is a practical approach to gathering data. The challenge with this is that it requires a ton of computational power to process all these images, making it both time-consuming and resource intensive.

This project aims to address this problem by developing a distributed wildlife monitoring system. By leveraging the computational power of several distributed servers across the network, the system distributed the workload, making computation faster, scalable, and more efficient. It not only accelerates the analysis process but also makes wildlife monitoring more accessible, efficient and affordable

**2. BACKGROUND THEORY**

**Single Coherent System View:** The main principle in distributed computing is the maintenance of a single-system view. The user interaction with the system, regardless of the complexity or the number of nodes involved, should remain consistent and straightforward. This philosophy extends to the realm of distributed applications, where the need for components of an application to communicate internally and externally should not translate to added complexity for the user.

A diagram of a computer

Description automatically generated  
Figure 1: A distributed system organized as middleware. The middleware layer extends over multiple machines and offers each application the same interface.

# **3. METHODOLOGY**

This section outlines the architecture and workings of the distributed wildlife monitoring system.

**Architecture Overview**

The architecture of the distributed system consists of a client machine, middleware, and distributed servers. Each component plays a crucial role in the data processing workflow.

**Client Machine**

The client machine initiates the data processing task. It is typically a machine at the conservation center or research facility that contains the images taken from the various locations in the national park. It receives these images from the camera traps and requests data processing.

**Middleware**

Middleware serves as the intermediary between the client and the distributed servers. It makes the distribution of data and processing transparent to the client. When the client contacts it for data processing, it distributes the images to the servers, collects and aggregates results from the servers and sends them back to the client, effectively acting as the data processing server from the perspective of the client.

**Distributed Servers**

These machines are responsible for the actual data processing task. They are usually high-performance computers equipped with resources to carry out computationally intensive tasks efficiently. Each machine is assigned a certain animal it is supposed to identify and keep a count of from the images. Upon receiving the images, the servers process the images concurrently and return the results to the middleware.

**Working Of** **The System**

A diagram of a software server

Description automatically generated with medium confidence

Figure 2: Overview

1. When the client is ready with the images from the camera traps, it initiates a TCP connection request to the middleware process that is always running. Upon acceptance of the request, it sends all the images to the server to be processed.
2. The middleware then contacts all the processing servers that are always running and sends all the images to all the servers for processing. Each server is assigned a certain animal.
3. In this case, server1 is assigned cats, server2 is assigned elephants, and server3 is assigned birds.
4. Upon reception, each server detects its assigned animal in the images, takes a total count and sends the results to the middleware as a CSV file.
5. The middleware then receives these CSV files, aggregates them, and sends the aggregated file to the client for further analysis and decision-making

**3.1 Data Flow Diagram**

### **Level 0**

A diagram of a blue circle with white text

Description automatically generated

#### **External Entities:**

Client: The user or system that interacts with the distributed system by sending images for processing and receiving the aggregated results.

#### **Process:**

Wildlife Monitoring System (Image Processing System): Represents the entire system as a single process. This encompasses receiving images from the client, processing the images through the distributed components (x.py, y.py, z.py, and ds.py), and sending back the aggregated results.

#### **Data Flows:**

1. Images to Process: Flow from the Client to the Image Processing System.
2. Aggregated Results (CSV): Flow from the Image Processing System back to the Client.

### **Level 1**

A diagram of a computer

Description automatically generated

#### **External Entity:**

1. **Client**: Initiates the image processing by sending images and receiving the final processed results.

#### **Processes:**

1. **Receive and Distribute Images (Ds.py)**: Acts as a layer of abstraction to hide the distributed processing from the client. Receives images from the client and distributes them to the servers (x.py, y.py, z.py).
2. **Process Cats (x.py)**: Receives images from Ds.py. Calls the file predict\_cat.py to identify and count cats, generates a CSV file with the counts and sends the CSV file to the server.
3. **Process Elephants (y.py)**: Receives images from Ds.py. Calls the file predict\_elephant.py to identify and count elephants, generates a CSV file with the counts and sends the CSV file to the server.
4. **Birds (z.py)**: Receives images from Ds.py. Calls the file predict\_bird.py to identify and count birds, generates a CSV file with the counts and sends the CSV file to the server.
5. **Aggregate and Send Results (Ds.py)**: Collects the CSV files from x.py, y.py, z.py, aggregates the data into a final CSV file, and sends it back to the client.

#### **Data Stores:**

1. **Unprocessed Images**: Temporarily stores the images received from the client before processing.
2. **Cat Count, Bird Count, Elephant Count CSV**: Temporarily stores CSV files from each processing node before aggregation.
3. **Aggregated Count CSV**: Holds the final aggregated results.

**x.py (server)**

import socket

import os

import zipfile

import shutil

import predict\_bird

def zip\_directory(directory\_name, zip\_name):

with zipfile.ZipFile(zip\_name, 'w', zipfile.ZIP\_DEFLATED) as zipf:

for root, dirs, files in os.walk(directory\_name):

for file in files:

file\_path = os.path.join(root, file)

zipf.write(file\_path, arcname=os.path.relpath(file\_path, start=directory\_name))

def process\_files(input\_dir, output\_dir):

predict\_bird.predict(input\_dir)

def unzip\_file(zip\_path, extract\_to):

"""

Unzips the file at zip\_path to the directory specified by extract\_to.

"""

if not os.path.exists(extract\_to):

os.makedirs(extract\_to)

with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

zip\_ref.extractall(extract\_to)

print(f"Files have been extracted to {extract\_to}")

def receive\_file(file\_path, host, port):

"""

Listens for an incoming file transfer and saves the file to file\_path.

"""

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as sock:

sock.bind((host, port))

sock.listen(1)

print(f"Listening on {host}:{port} for incoming files...")

conn, addr = sock.accept()

with conn:

print(f"Connection established with {addr}")

with open(file\_path, 'wb') as file:

while True:

data = conn.recv(1024)

if not data:

break

file.write(data)

print(f"File received and saved as {file\_path}")

def send\_file\_back(file\_path, host, port):

"""

Sends the specified file to the given host and port using a TCP socket.

"""

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as sock:

try:

sock.connect((host, port))

with open(file\_path, 'rb') as file:

sock.sendfile(file)

print(f"File {file\_path} has been sent back to {host}:{port}")

except Exception as e:

print(f"Failed to send {file\_path} back to {host}:{port}. Error: {str(e)}")

def delete\_file(file\_path):

# Check if file exists

if os.path.exists(file\_path):

# Delete the file

os.remove(file\_path)

print(f"File {file\_path} has been deleted.")

else:

print(f"File {file\_path} does not exist.")

if \_\_name\_\_ == "\_\_main\_\_":

file\_path = 'received\_user\_input\_x.zip' # The file path to save the received zip file

host = 'localhost' # The host IP address to bind to

port = 12346 # The port to listen on

input\_dir = "X\_INPUT" # The directory to extract the zip contents to

output\_dir = "X\_OUTPUT" # The directory to save processed files to

# Receive a file transfer

receive\_file(file\_path, host, port)

# Unzip the received file

unzip\_file(file\_path, input\_dir)

# Process the unzipped files

process\_files(input\_dir, output\_dir)

zip\_directory(output\_dir,"x.zip")

send\_file\_back("x.zip", 'localhost', 12349)

delete\_file('received\_user\_input\_x.zip')

delete\_file('x.zip')

**y.py (server)**

import socket

import os

import zipfile

import shutil

import predict\_cat

def zip\_directory(directory\_name, zip\_name):

with zipfile.ZipFile(zip\_name, 'w', zipfile.ZIP\_DEFLATED) as zipf:

for root, dirs, files in os.walk(directory\_name):

for file in files:

file\_path = os.path.join(root, file)

zipf.write(file\_path, arcname=os.path.relpath(file\_path, start=directory\_name))

def process\_files(input\_dir, output\_dir):

predict\_cat.predict(input\_dir)

def send\_file\_back(file\_path, host, port):

"""

Sends the specified file to the given host and port using a TCP socket.

"""

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as sock:

try:

sock.connect((host, port))

with open(file\_path, 'rb') as file:

sock.sendfile(file)

print(f"File {file\_path} has been sent back to {host}:{port}")

except Exception as e:

print(f"Failed to send {file\_path} back to {host}:{port}. Error: {str(e)}")

def unzip\_file(zip\_path, extract\_to):

"""

Unzips the file at zip\_path to the directory specified by extract\_to.

"""

if not os.path.exists(extract\_to):

os.makedirs(extract\_to)

with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

zip\_ref.extractall(extract\_to)

print(f"Files have been extracted to {extract\_to}")

def receive\_file(file\_path, host, port):

"""

Listens for an incoming file transfer and saves the file to file\_path.

"""

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as sock:

sock.bind((host, port))

sock.listen(1)

print(f"Listening on {host}:{port} for incoming files...")

conn, addr = sock.accept()

with conn:

print(f"Connection established with {addr}")

with open(file\_path, 'wb') as file:

while True:

data = conn.recv(1024)

if not data:

break

file.write(data)

print(f"File received and saved as {file\_path}")

def delete\_file(file\_path):

# Check if file exists

if os.path.exists(file\_path):

# Delete the file

os.remove(file\_path)

print(f"File {file\_path} has been deleted.")

else:

print(f"File {file\_path} does not exist.")

if \_\_name\_\_ == "\_\_main\_\_":

file\_path = 'received\_user\_input\_y.zip' # The file path to save the received zip file

host = 'localhost' # The host IP address to bind to

port = 12347 # The port to listen on

input\_dir = "Y\_INPUT" # The directory to extract the zip contents to

output\_dir = "Y\_OUTPUT" # The directory to save processed files to

# Receive a file transfer

receive\_file(file\_path, host, port)

# Unzip the received file

unzip\_file(file\_path, input\_dir)

# Process the unzipped files

process\_files(input\_dir, output\_dir)

zip\_directory(output\_dir,"y.zip")

send\_file\_back("y.zip", 'localhost', 12349)

delete\_file('received\_user\_input\_y.zip')

delete\_file('y.zip')

**z.py (server)**

import socket

import os

import zipfile

import shutil

import predict\_elephant

def zip\_directory(directory\_name, zip\_name):

with zipfile.ZipFile(zip\_name, 'w', zipfile.ZIP\_DEFLATED) as zipf:

for root, dirs, files in os.walk(directory\_name):

for file in files:

file\_path = os.path.join(root, file)

zipf.write(file\_path, arcname=os.path.relpath(file\_path, start=directory\_name))

def process\_files(input\_dir, output\_dir):

predict\_elephant.predict(input\_dir)

def unzip\_file(zip\_path, extract\_to):

"""

Unzips the file at zip\_path to the directory specified by extract\_to.

"""

if not os.path.exists(extract\_to):

os.makedirs(extract\_to)

with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

zip\_ref.extractall(extract\_to)

print(f"Files have been extracted to {extract\_to}")

def receive\_file(file\_path, host, port):

"""

Listens for an incoming file transfer and saves the file to file\_path.

"""

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as sock:

sock.bind((host, port))

sock.listen(1)

print(f"Listening on {host}:{port} for incoming files...")

conn, addr = sock.accept()

with conn:

print(f"Connection established with {addr}")

with open(file\_path, 'wb') as file:

while True:

data = conn.recv(1024)

if not data:

break

file.write(data)

print(f"File received and saved as {file\_path}")

def send\_file\_back(file\_path, host, port):

"""

Sends the specified file to the given host and port using a TCP socket.

"""

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as sock:

try:

sock.connect((host, port))

with open(file\_path, 'rb') as file:

sock.sendfile(file)

print(f"File {file\_path} has been sent back to {host}:{port}")

except Exception as e:

print(f"Failed to send {file\_path} back to {host}:{port}. Error: {str(e)}")

def delete\_file(file\_path):

# Check if file exists

if os.path.exists(file\_path):

# Delete the file

os.remove(file\_path)

print(f"File {file\_path} has been deleted.")

else:

print(f"File {file\_path} does not exist.")

if \_\_name\_\_ == "\_\_main\_\_":

file\_path = 'received\_user\_input\_z.zip' # The file path to save the received zip file

host = 'localhost' # The host IP address to bind to

port = 12348 # The port to listen on

input\_dir = "Z\_INPUT" # The directory to extract the zip contents to

output\_dir = "Z\_OUTPUT" # The directory to save processed files to

# Receive a file transfer

receive\_file(file\_path, host, port)

# Unzip the received file

unzip\_file(file\_path, input\_dir)

# Process the unzipped files

process\_files(input\_dir, output\_dir)

zip\_directory(output\_dir,"z.zip")

# Assuming ds.py is listening on localhost:12349 for the files from x.py, y.py, and z.py

send\_file\_back("z.zip", 'localhost', 12349)

delete\_file('received\_user\_input\_z.zip')

delete\_file('z.zip')

**Ds.py (middleware)**

import socket

import time

import os

import zipfile

import pandas as pd

def receive\_file(file\_path, host, port):

"""

Receives a file over TCP and saves it to the specified path.

"""

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as sock:

sock.bind((host, port))

sock.listen(1)

print(f"Waiting for file on {host}:{port}...")

conn, \_ = sock.accept()

with conn:

with open(file\_path, 'wb') as file:

while True:

data = conn.recv(1024)

if not data:

break

file.write(data)

print(f"Received file saved as {file\_path}.")

def send\_file(file\_path, target\_servers, max\_attempts=5, retry\_delay=2):

"""

Attempts to send the specified file to each target server in the target\_servers list.

Retries a maximum of max\_attempts times with a delay of retry\_delay seconds between attempts.

"""

for host, port in target\_servers:

for attempt in range(1, max\_attempts + 1):

try:

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as sock:

sock.connect((host, port))

with open(file\_path, 'rb') as file:

sock.sendfile(file)

print(f"Successfully sent {file\_path} to {host}:{port}")

break # Success, no need to retry

except ConnectionRefusedError:

print(f"Attempt {attempt} failed: Connection to {host}:{port} refused.")

if attempt < max\_attempts:

print(f"Retrying in {retry\_delay} seconds...")

time.sleep(retry\_delay)

else:

print(f"Failed to send {file\_path} to {host}:{port} after {max\_attempts} attempts.")

def receive\_and\_unzip\_files(host, port, output\_dir):

"""

Listens for incoming zip files from x.py, y.py, and z.py, saves them, and unzips them into the specified output directory.

"""

if not os.path.exists(output\_dir):

os.makedirs(output\_dir)

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((host, port))

server\_socket.listen(3) # Expecting 3 connections

print(f"Waiting for zip files on {host}:{port}...")

for \_ in range(3): # Expecting 3 files in total

conn, addr = server\_socket.accept()

with conn:

print(f"Connection established with {addr}")

received\_zip = os.path.join(output\_dir, f"received\_{addr[1]}.zip")

with open(received\_zip, 'wb') as file:

while True:

data = conn.recv(1024)

if not data:

break

file.write(data)

print(f"File received and saved as {received\_zip}")

unzip\_file(received\_zip, output\_dir)

os.remove(received\_zip) # Delete the zip file after extraction

def unzip\_file(zip\_path, extract\_to):

"""

Unzips the file at zip\_path to the directory specified by extract\_to.

"""

with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

zip\_ref.extractall(extract\_to)

print(f"Files from {zip\_path} have been extracted to {extract\_to}")

def zip\_directory(directory\_name, zip\_name):

with zipfile.ZipFile(zip\_name, 'w', zipfile.ZIP\_DEFLATED) as zipf:

for root, dirs, files in os.walk(directory\_name):

for file in files:

file\_path = os.path.join(root, file)

zipf.write(file\_path, arcname=os.path.relpath(file\_path, start=directory\_name))

print(f"Directory {directory\_name} zipped into {zip\_name}")

def combine\_csv\_files(directory\_path):

# List all CSV files in the directory

csv\_files = [file for file in os.listdir(directory\_path) if file.endswith('.csv')]

# Combine all CSV files into one dataframe

combined\_csv\_data = pd.DataFrame()

for file in csv\_files:

file\_path = os.path.join(directory\_path, file)

csv\_data = pd.read\_csv(file\_path)

combined\_csv\_data = pd.concat([combined\_csv\_data, csv\_data], ignore\_index=True)

# Save the combined dataframe to 'final.csv' in the same directory

final\_csv\_path = os.path.join(directory\_path, 'final.csv')

combined\_csv\_data.to\_csv(final\_csv\_path, index=False)

# Delete the old CSV files

#for file in csv\_files:

#os.remove(os.path.join(directory\_path, file))

def delete\_file(file\_path):

# Check if file exists

if os.path.exists(file\_path):

# Delete the file

os.remove(file\_path)

print(f"File {file\_path} has been deleted.")

else:

print(f"File {file\_path} does not exist.")

if \_\_name\_\_ == "\_\_main\_\_":

file\_path = 'received\_data.zip'

receive\_host = 'localhost'

receive\_port = 12345

# Target servers where the file should be sent after receipt

target\_servers = [

('localhost', 12346), # Server for x.py

('localhost', 12347), # Server for y.py

('localhost', 12348) # Server for z.py

]

# Step 1: Receive a file from a client

receive\_file(file\_path, receive\_host, receive\_port)

# Step 2: Forward the file to the target servers

send\_file(file\_path, target\_servers)

common\_output\_dir = 'Common\_Output' # Directory where files from x, y, z are extracted

receive\_and\_unzip\_files('localhost', 12349, common\_output\_dir)

combine\_csv\_files("Common\_Output")

final\_zip\_name = 'final\_output.zip'

zip\_directory('Common\_Output', final\_zip\_name)

send\_file(final\_zip\_name, [('localhost', 12350)])

delete\_file('received\_data.zip')

delete\_file('final\_output.zip')

**client.py**

import os

import socket

import zipfile

def zip\_files(input\_dir, output\_zip):

"""

Zips all files in the specified input directory and saves the zip file with the given name.

"""

with zipfile.ZipFile(output\_zip, 'w', zipfile.ZIP\_DEFLATED) as zipf:

for root, \_, files in os.walk(input\_dir):

for file in files:

file\_path = os.path.join(root, file)

zipf.write(file\_path, os.path.relpath(file\_path, input\_dir))

def send\_file(file\_path, host, port):

"""

Sends the specified file to the given host and port using a TCP socket.

"""

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as sock:

sock.connect((host, port))

with open(file\_path, 'rb') as file:

sock.sendfile(file)

print(f"File {file\_path} has been sent to {host}:{port}")

def receive\_and\_unzip\_file(file\_path, host, port, extract\_to):

"""

Receives a file over TCP and saves it to the specified path, then unzips it.

"""

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as sock:

sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

sock.bind((host, port))

sock.listen(1)

print(f"Waiting for file on {host}:{port}...")

conn, \_ = sock.accept()

with conn:

with open(file\_path, 'wb') as file:

while True:

data = conn.recv(1024)

if not data:

break

file.write(data)

print(f"Received file saved as {file\_path}")

# Unzip the received file

with zipfile.ZipFile(file\_path, 'r') as zip\_ref:

zip\_ref.extractall(extract\_to)

print(f"File {file\_path} has been unzipped to {extract\_to}")

def delete\_file(file\_path):

# Check if file exists

if os.path.exists(file\_path):

# Delete the file

os.remove(file\_path)

print(f"File {file\_path} has been deleted.")

else:

print(f"File {file\_path} does not exist.")

if \_\_name\_\_ == "\_\_main\_\_":

input\_dir = 'User\_Input' # Directory to zip

output\_zip = 'user\_input.zip' # Output zip file name

host = 'localhost' # Server's IP address (change if needed)

port = 12345 # Server's port number

# Ensure the 'User\_Input' directory exists

if not os.path.exists(input\_dir):

print(f"The directory {input\_dir} does not exist. Please create it and try again.")

else:

zip\_files(input\_dir, output\_zip) # Zip the files

send\_file(output\_zip, host, port) # Send the zip file

file\_path = 'received\_final\_output.zip'

host = 'localhost' # Listening host

port = 12350 # Listening port

extract\_to = 'User\_Output' # Extraction directory

# Ensure the extraction directory exists

if not os.path.exists(extract\_to):

os.makedirs(extract\_to)

receive\_and\_unzip\_file(file\_path, host, port, extract\_to)

delete\_file('user\_input.zip')

delete\_file('received\_final\_output.zip')

**predict\_elephant.py**

from ultralytics import YOLO

from PIL import Image as PILImage

from IPython.display import display, Image

import os

import csv

def predict(directory\_path):

model = YOLO('/home/vaibhav/Desktop/temp/elephant/weights/best.pt')

num\_annotated\_boxes\_total = 0

for filename in os.listdir(directory\_path):

if filename.endswith(".jpg") or filename.endswith(".jpeg") or filename.endswith(".png"):

image\_path = os.path.join(directory\_path, filename)

results = model.predict(source=image\_path, conf=0.7)

num\_annotated\_boxes = 0

for result in results:

img\_bytes = result.save()

display(Image(filename=img\_bytes))

num\_annotated\_boxes += len(result.boxes.xyxy)

num\_annotated\_boxes\_total += num\_annotated\_boxes

print("Total number of annotated boxes for", filename, ":", num\_annotated\_boxes)

data =[{'Elephant' : num\_annotated\_boxes\_total}]

path = '/home/vaibhav/Desktop/temp/Z\_OUTPUT/outz.csv'

with open(path, 'w', newline='') as csvfile:

fieldnames = ['Elephant']

writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

# Write the header

writer.writeheader()

# Write the data

for row in data:

writer.writerow(row)

print("Total number of annotated boxes for all images:", num\_annotated\_boxes\_total)

if \_\_name\_\_ == '\_\_main\_\_':

predict('/home/vaibhav/Desktop/temp/User\_Input')

**predict\_cat.py**

from ultralytics import YOLO

from PIL import Image as PILImage

from IPython.display import display, Image

import os

import csv

def predict(directory\_path):

model = YOLO('/home/vaibhav/Desktop/temp/cat/weights/best.pt')

num\_annotated\_boxes\_total = 0

for filename in os.listdir(directory\_path):

if filename.endswith(".jpg") or filename.endswith(".jpeg") or filename.endswith(".png"):

image\_path = os.path.join(directory\_path, filename)

results = model.predict(source=image\_path, conf=0.7)

num\_annotated\_boxes = 0

for result in results:

img\_bytes = result.save()

display(Image(filename=img\_bytes))

num\_annotated\_boxes += len(result.boxes.xyxy)

num\_annotated\_boxes\_total += num\_annotated\_boxes

print("Total number of annotated boxes for", filename, ":", num\_annotated\_boxes)

data =[{'Cat' : num\_annotated\_boxes\_total}]

path = '/home/vaibhav/Desktop/temp/Y\_OUTPUT/outy.csv'

with open(path, 'w', newline='') as csvfile:

fieldnames = ['Cat']

writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

# Write the header

writer.writeheader()

# Write the data

for row in data:

writer.writerow(row)

print("Total number of annotated boxes for all images:", num\_annotated\_boxes\_total)

if \_\_name\_\_ == '\_\_main\_\_':

predict('/home/vaibhav/Desktop/temp/User\_Input')

**predict\_bird.py**

from ultralytics import YOLO

from PIL import Image as PILImage

from IPython.display import display, Image

import os

import csv

def predict(directory\_path):

model = YOLO('/home/vaibhav/Desktop/temp/bird/weights/best.pt')

num\_annotated\_boxes\_total = 0

for filename in os.listdir(directory\_path):

if filename.endswith(".jpg") or filename.endswith(".jpeg") or filename.endswith(".png"):

image\_path = os.path.join(directory\_path, filename)

results = model.predict(source=image\_path, conf=0.7)

num\_annotated\_boxes = 0

for result in results:

img\_bytes = result.save()

display(Image(filename=img\_bytes))

num\_annotated\_boxes += len(result.boxes.xyxy)

num\_annotated\_boxes\_total += num\_annotated\_boxes

print("Total number of annotated boxes for", filename, ":", num\_annotated\_boxes)

data =[{'Bird' : num\_annotated\_boxes\_total}]

path = '/home/vaibhav/Desktop/temp/X\_OUTPUT/outx.csv'

with open(path, 'w', newline='') as csvfile:

fieldnames = ['Bird']

writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

# Write the header

writer.writeheader()

# Write the data

for row in data:

writer.writerow(row)

print("Total number of annotated boxes for all images:", num\_annotated\_boxes\_total)

if \_\_name\_\_ == '\_\_main\_\_':

predict('/home/vaibhav/Desktop/temp/User\_Input')

**4. RESULTS AND DISCUSSION**

Model Training:

* Three separate YOLO models were trained using the collected dataset, each focusing on detecting a specific animal (dogs, cats, and birds).
* The training process involved labelling the images, configuring the YOLO model architecture, and optimizing the model parameters.

Inference and Counting:

* After training, the YOLO models were used to perform inference on a set of test images.
* During inference, the models identified the presence of animals in each image and counted the occurrences of the specified animal classes.

CSV File Generation:

* For each animal class (dogs, cats, birds), the YOLO models generated separate CSV files containing image filenames and corresponding counts of animal occurrences.

Middleware Integration:

* A middleware application was developed to combine the three separate CSV files into a single consolidated CSV file.
* The middleware ensured proper formatting and alignment of data from all three sources before generating the final CSV output.

User:

* The combined CSV file containing the counts of animal occurrences was made available to users.
* Users could easily download and analyse the data for further insights and decision-making.

**5. CONCLUSION AND FUTURE ENHANCEMENTS**

**Conclusion**

The project's main objective is to represent a domain of distributed computing, specifically in maintaining the transparency of complex processes from the client's perspective. The client interacts with the system as though all operations occur on a singular server, an illusion meticulously crafted and maintained through the system's design. This level of transparency is emblematic of classical distributed systems, where the intricacies of data processing across multiple servers are abstracted away from the end-user

**Future Enhancements**

1. **User Interface**: As of now we are using a terminal to interface with the user, in future we can work on a more user-friendly Interface
2. **Dynamic Load Balancing:** Implementing a more dynamic load balancing system could ensure optimal utilization of resources, reducing bottlenecks and improving response times.

**REFERENCES**

1. Zhang, Junguo & Luo, Xin & Chen, Chen & Liu, Zhen & Cao, Shuai. (2014). A Wildlife Monitoring System Based on Wireless Image Sensor Networks. Sensors & Transducers Journal. 180. 104-109.
2. Bandari, G., Nirmala Devi, P. L., & Srividya, P. (2022). Wild Animal Detection using a Machine Learning Approach and Alerting using LoRa Communication. In 2022 International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON) (pp. 1-5). Bangalore, India. doi: 10.1109/SMARTGENCON56628.2022.10083577. Keywords: GSM, Animals, Image processing, Wildlife, Forestry, Streaming media, Monitoring, Convolutional Neural Network, Prediction, Training and Validation, Machine Learning, Wild Animal Detection, Deep Learning, LoRa Communication, Automatic Irrigation, Sensors, Internet of Things.