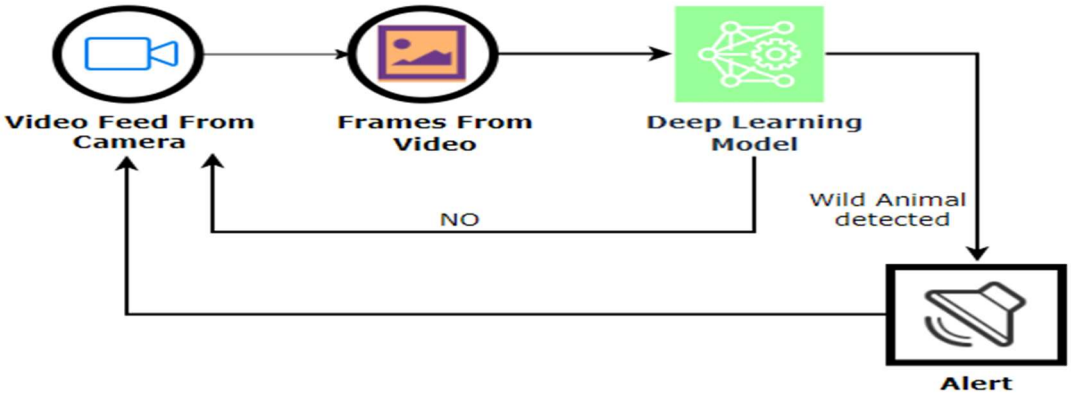
Intelligent Alert System For Forest Tribal People

1. INTRODUCTION

1.1 Overview

Tribal people constitute 8.6% of the nation's total population, and most of them spend the greater part of their lives in the proximity of trees and villages or clans near to the forest. There are so many challenges faced by these people out of which Human-wildlife conflict (Human-wildlife conflict refers to the interaction between wild animals and humans, and the resultant negative impact on people, animals, resources, and habitats) is a serious challenge undermining the protection of tribal regions. The major types of human-wildlife conflict in the area include crop-raiding, livestock predation, increased risk of livestock diseases, and direct threats to human life. So active measures are to be implemented to mitigate these problems and safeguard the future of the wildlife.



1.2 Purpose

- Get to know about image processing.
- Understand Deep learning Architecture.
- Understand the concepts like Artificial neural network ana Convolution neural network.
- Classify images using a Convolutional Neural Network.
- Understand the concepts of image and video analysis using OpenCV.
- To build develop an Intelligent Alert System using Deep Learning.
- This project aims to save the lives of tribal by notifying about wildlife predation with the help of Artificial Intelligence.

2. LITERATURE SURVEY

2.1 Existing problem

Human-wildlife conflict is when encounters between humans and wildlife lead to negative results, such as loss of property, livelihoods, and even life. Defensive and retaliatory killing may eventually drive these species to extinction. These encounters not only result in suffering for both people and wildlife immediately impacted by the conflict; they can also have a global reach, with groups such as sustainable development agencies and businesses feeling its residual effects. The scope of the issue is significant and truly global, but we are nowhere near being able to address it at the scale needed. As human populations and demand for space continue to grow, people and wildlife are increasingly interacting and competing for resources, which can lead to increased human-wildlife conflict.

Along with other threats, human-wildlife conflict has driven the decline of once-abundant species and is pushing others to the brink of extinction. But the human-wildlife conflict issue has far-reaching impacts beyond the wildlife and communities immediately affected by it. With human-wildlife conflict centered around the interaction between wildlife and humans, human-wildlife coexistence is strongly linked and important to sustainable development activities. If not effectively managed, human-wildlife conflict has the potential to negatively affect these activities and conservation much more broadly.

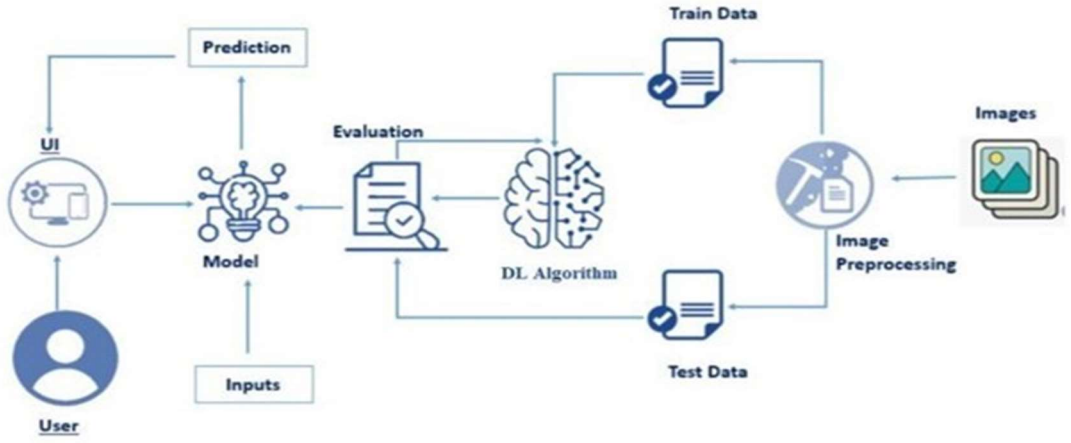
2.2 Proposed solution

An Intelligent Alert System that alerts tribal people by notifying about wildlife predation with the help of Artificial Intelligence (Deep Learning). The proposed system does the following

- Take the Video feed from the pre-installed camera.
- Read the Frame.
- Give the image as input to a model built for analysis.
- Alert the people if any wild animal is detected.

3 THEORITICAL ANALYSIS

3.1 Block diagram



3.2 Hardware/ Software designing

- Hardware requirements
  - Processor – i5
  - RAM – 8GB
- Software required – Anaconda, Jupyter IDE.
- Packages required
  - Tensorflow - This package is used as backend support to Keras
  - Keras - This package is used for building Neural Network layers
  - opencv - This package is used for image processing
  - Twilio – This package used to use Twilio for sending messages.

4. EXPERIMENTAL INVESTIGATIONS

Milestone 1: Dataset Collection

Artificial Intelligence is a data hunger technology, it depends heavily on data, without data, it is impossible for a machine to learn. It is the most crucial aspect that makes algorithm training possible. In Convolutional Neural Networks, as it deals with images, we need training and testing data set. It is the actual data set used to train the model for performing various actions.

Activity 1: Download the dataset

You can collect datasets from different open sources like kaggle.com, data.gov, UCI machine learning repository, etc.

You can also create data set with a different set of classes like Domestic Animals, Humans, and Wild Animals. Moreover, to evaluate the model we divided the dataset to 70% for training data, 30% for testing the data.

The dataset used for this project was obtained from Kaggle. Please refer to the link given below to download the data set and project files

Milestone 2: Image Preprocessing

Image Pre-processing includes the following main tasks

- Import ImageDataGenerator Library.
- Configure ImageDataGenerator Class.
- Applying ImageDataGenerator functionality to train set and test set.

Note: The ImageDataGenerator accepts the original data, randomly transforms it, and returns only the new, transformed data.

Activity 1: Importing The ImageDataGenerator Library

The first step is usually importing the libraries that will be needed in the program.

Import Keras library from that library import the ImageDataGenerator Library to your Python script:

```
#Import ImageDataGenerator Library
import keras
from keras.preprocessing.image import ImageDataGenerator
```

Define the parameters /arguments for ImageDataGenerator class

```
#Define the parameters /arguments for ImageDataGenerator class
train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,
        rotation_range=180,zoom_range=0.2,horizontal_flip=True)

test_datagen=ImageDataGenerator(rescale=1./255)
```

Here the arguments which we are given inside the image data generator class is like, rescale, shear_range, rotation range of the image, and zoom range that we can consider for image, etc.

- Image shifts via the width_shift_range and height_shift_range arguments.
- The image flips via the horizontal_flip and vertical_flip arguments.
- Image rotations via the rotation_range argument

Activity 2: Applying ImageDataGenerator Functionality To Trainset And Testset

The ImageDataGenerator class has three methods flow ( ), flow_from_directory ( ) and flow_from_dataframe ( ) to read the images from a big numpy array and folders containing images.

flow_from_directory ( ) expects at least one directory under the given directory path.

Apply flow_from_directory ( ) method for Train folder

```
#Applying ImageDataGenerator functionality to trainset
x_train=train_datagen.flow_from_directory(
    directory= r"E:\dataset\train_set",
    target_size=(64,64),
    batch_size=32,
    class_mode='categorical')
```

Now will apply flow_from_directory ( ) method for test folder.

```
#Applying ImageDataGenerator functionality to test set
x_test=test_datagen.flow_from_directory(
    directory= r"E:\dataset\test_set",
    target_size=(64,64),
    batch_size=32,
    class_mode='categorical')
```

- The directory must be set to the path where your training folders are present.
- The target_size is the size of your input images, every image will be resized to this size.
- batch_size: No. of images to be yielded from the generator per batch.
- "batch_size" in both train and test generators is to some number that divides your total number of images in your train set and train set respectively.
- class_mode: Set "binary" if you have only two classes to predict, if not set to "categorical".

Milestone 3: Model Building

This activity includes the following steps

- Import the model building Libraries
- Initializing the model
- Adding CNN Layers
- Adding Hidden Layer
- Adding Output Layer
- Configure the Learning Process
- Training and testing the model
- Saving the model

Activity 1: Import Libraries

This is a very crucial step in our deep learning model building process. We have to define how our model will look and that requires.

```
'''Importing the model building libraries'''
#to define linear initializations import Sequential
from keras.models import Sequential
#To add layers import Dense
from keras.layers import Dense
# to create a convolution kernel import Convolution2D
from keras.layers import Convolution2D
# Adding Max pooling Layer
from keras.layers import MaxPooling2D
# Adding Flatten Layer
from keras.layers import Flatten
```

Activity 2: Initializing the Model

Keras has 2 ways to define a neural network:

- Sequential
- Function API

The Sequential class is used to define linear initializations of network layers which then, collectively, constitute a model. In our example below, we will use the Sequential constructor to create a model, which will then have layers added to it using the add () method.
Now, will initialize our model.

```
# Initializing the model
model=Sequential()
```

Activity 2: Adding Cnn Layers

We will be adding three layers for CNN

- Convolution layer
- Pooling layer
- Flattening layer

Adding Convolutional Layer

The convolutional layer is the first and core layer of CNN. It is one of the building blocks of a CNN and is used for extracting important features from the image.

In the Convolution operation, the input image will be convolved with the feature detector/filters to get a feature map. The important role of the feature detector is to extract the features from the image. The group of feature maps is called a feature layer.

```
# Adding CNN layers
model.add(Convolution2D(32,(3,3),input_shape=(64,64,3),
                        activation='relu'))
```

Activity 3: Adding Dense Layers

The name suggests that layers are fully connected (dense) by the neurons in a network layer. Each neuron in a layer receives input from all the neurons present in the previous layer. Dense is used to add the layers.

Adding Hidden layers

This step is to add a dense layer (hidden layer). We flatten the feature map and convert it into a vector or single dimensional array in the Flatten layer. This vector array is fed it as an input to the neural network and applies an activation function, such as sigmoid or other, and returns the output.

```
# Adding Hidden Layers
model.add(Dense(init='uniform',activation='relu',units=300))
```

- init is the weight initialization; initialization function is the network initialization function which sets all the weights and biases of a network to values suitable as a starting point for training.
- units, which denotes is the number of neurons in the hidden layer.
- Activation function defines the output of input or set of inputs or in other terms defines node of the output of node that is given in inputs. They basically decide to deactivate neurons or activate them to get the desired output. It also performs a nonlinear transformation on the input to get better results on a complex neural network. Its nodes here just pass on the information (features) to the hidden layer.
- You can add many hidden layers, in our project we are added more two hidden layers. The 2nd hidden layer with 100 neurons and 3rd hidden layer with 60 neurons.

```
# Adding 2nd hidden layer
model.add(Dense(init='uniform',activation='relu',units=100))
```

```
# Adding 3rd hidden layer
model.add(Dense(init='uniform',activation='relu',units=60))
```

Adding the output layer

This step is to add a dense layer (output layer) where you will be specifying the number of classes your dependent variable has, activation function, and weight initializer as the arguments. We use the add () method to add dense layers. In this layer, no need of
mentioning input dimensions as we have mentions them in the above layer itself.

```
# Adding output layer
model.add(Dense(init='uniform',activation='softmax',units=3))
```

Note: if you have only one or two class in output put “units= 1” and “activation = sigmoid”. If you have more classes for supposing there are 3 classes then put “units= 3” and “activation = softmax”.

Activity 3: Configuring The Learning Process

With both the training data defined and model defined, it's time configure the learning process. This is accomplished with a call to the compile () method of the Sequential model class. Compilation requires 3 arguments: an optimizer, a loss function, and a list of metrics.

```
# Configure the learning process
model.compile(loss='categorical_crossentropy',
              optimizer='adam',metrics=["accuracy"])
```

Note: In our project we have 3 classes in the output, so the loss is categorical_crossentropy.
If you have only one or two classes in output put “loss = binary_crossentropy”.

Activity 4: Training The Model

At this point, we have training data and a fully configured neural network to train with loaded data. All that is left is to pass the data to the model for the

training process to commence, a process that is completed by iterating on the training data. Training begins by calling the fit ( ) method.

```
# Training the model
model.fit_generator(x_train,steps_per_epoch=31,
                    epochs=15,
                    validation_data=x_test,
                    validation_steps=11)
```

steps_perepoch = number of training images/batchsize
validation steps = no:of test images/32

Activity 5: Save The Model

Your model is to be saved for future purposes. This saved model can also be integrated with an android application or web application in order to predict something.

```
#save model
model.save('alert.h5')
```

The last and final step is to make use of our saved model to do predictions. For that we have a class in keras called load_model. Load_model is used to load our saved model h5 file (alert.h5).

```
#import numpy library
import numpy as np
#import load_model method to load our saved model
from keras.models import load_model
#import image from keras.preprocessing
from keras.preprocessing import image
#loading our saved model file
model = load_model("alert.h5")
img = image.load_img(r"E:\Guided Projects\domestic\cat.4003.jpg",
                     target_size=(64,64))

x = image.img_to_array(img)
#expanding the shape of image to 4 dimensions
x = np.expand_dims(x,axis=0)
pred = model.predict_classes(x)
pred
```

The above code implements the following steps:

- Load the model and import the libraries necessary for predictions
- load the image that you would like to predict
- convert the loaded image to an array
- The loaded image will be of three dimensions and our CNN model architecture cannot accept a Three-dimensional array
- so expand the dimensions so that the array will be of 4 dimensions
- Now give the array to predict the class

Milestone 4: Video Analysis

OpenCV is an open source library which provides us with the tools to perform almost any kind of image and video processing.

Activity 1: Capture Video From Camera

Often, we have to capture live stream with a camera. OpenCV provides a very simple interface to this. Let's capture a video from the camera (I am using the in-built webcam of my laptop), convert it into grayscale video, and display it.

To capture a video, you need to create a VideoCapture object. Its argument can be either the device index or the name of a video file. The device index is just the number to specify which camera. Normally one camera will be connected (as in my case). So I simply pass 0 (or -1). You can select the second camera by passing 1 and so on. After that, you can capture the frame-by-frame. But at the end, don't forget to release the capture. To read web cam will see the code.

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

while(True):
    ''' Capture frame-by-frame'''
    ret, frame = cap.read()

    '''Our operations on the frame come here'''
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    '''Display the resulting frame'''
    cv2.imshow('frame',gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

''' When everything done, release the capture'''
cap.release()
cv2.destroyAllWindows()
```

Activity 2: Importing The Required Libraries.

```
#import opencv
import cv2
#import numpy
import numpy as np
from keras.preprocessing import image
from keras.models  import load_model
#import Client from twilio API
from twilio.rest import Client
#import playsound package
from playsound import playsound
```

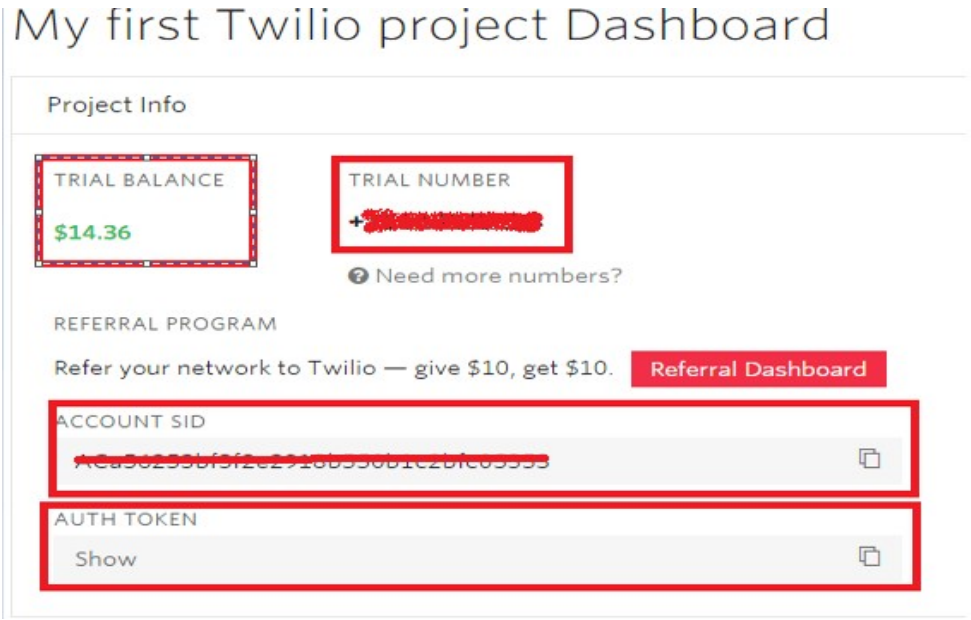
Loading our saved model file using load_model from Keras library

```
#Load saved model file using load_model method
model = load_model('alert.h5')
```

Activity 3: Creating An Account In Twilio Service

To proceed with coding you have to first create a service in twilio

- Go to this link and create a Twilio account. It is free and no credit card required.
- After creating account login to Twilio. Using this link,
- After login, you will be redirected to the home page or dashboard from there buy a free number to buy follow this link
- Once after you are done with buy a number, you can able to see your free trail number and the total balance in the dashboard.
- Along with that you can able to see the Account SID and Authentication token.
- By using that free trial number and the Account SID and Authentication token we alerting the people.
- We are going to integrating this Twilio credentials with our OpenCV. When the wild animal is detected it will give an alert through a message.



Activity 4: Use API To Send Message.

We got the Account SID and Authentication token from Twilio. Now we have to integrate all the OpenCV and Twilio to send an alert message.

To integrate it with python, the code looks like.

```
from twilio.rest import Client


# Your Account Sid and Auth Token from twilio.com/console
# DANGER! This is insecure. See http://twil.io/secure
account_sid = 'ACa56253bf3f2e2918b550b1c2bfc05353'
auth_token = 'your_auth_token'
client = Client(account_sid, auth_token)

message = client.messages \
                .create(
                     body="Join Earth's mightiest heroes. 
                     from_='+15017122661',
                     to='+15558675310'
                 )

print(message.sid)
```

Sending alert message by combining all the code snippets

- To play an alerting sound we required to install play sound library.

- To install this library, open anaconda prompt and execute the below command.
- Type “pip install play sound” click enter.

```
#import opencv
import cv2
#import numpy
import numpy as np
from keras.preprocessing import image
from keras.models  import load_model
#import Client from twilio API
from twilio.rest import Client
#import playsound package
from playsound import playsound
```


```
#import opencv
import cv2
#import numpy
import numpy as np
from keras.preprocessing import image
from keras.models  import load_model
#import Client from twilio API
from twilio.rest import Client
#import playsound package
from playsound import playsound
```


Activity 5: Use API To Send Message-2

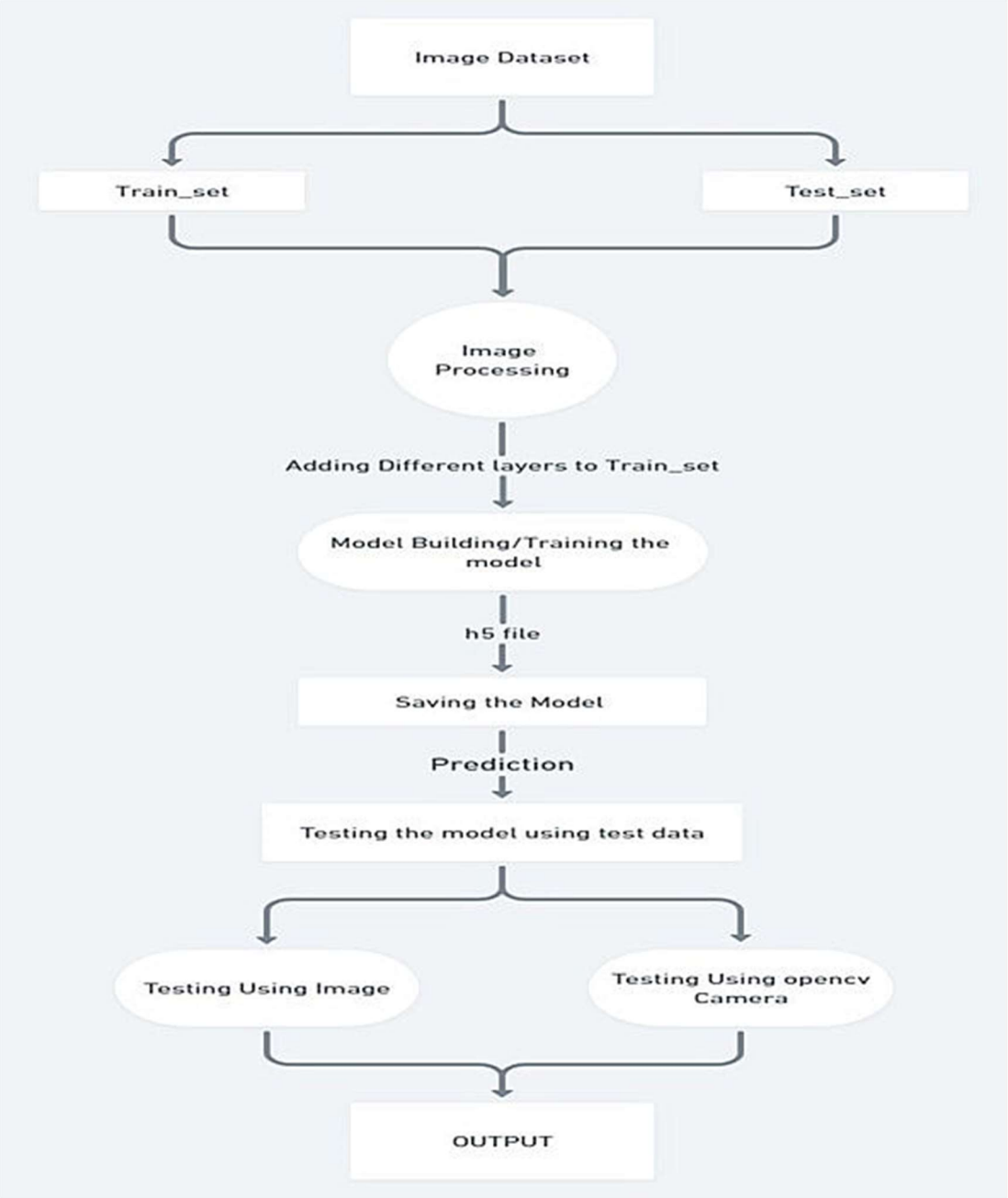
```
pred = model.predict_classes(x)
if pred[0]==2:
    #twilio account ssid
    account_sid = 'ACa56253bf3f2e2918b550b1c2bfc05353'
    #twilo account authentication toke
    auth_token = 'a10cb957a1b8bc17abba1e1de952d4b4'
    client = Client(account_sid, auth_token)

    message = client.messages \
    .create(
     body='Danger!. Wild animal is detected, stay alert',
     from_=' +15035125124', #the free number of twilio
     to='+91xxxxxxxxx')
    print(message.sid)
    print('Danger!!')
    print('Animal Detected')
    print ('SMS sent!')
    playsound(r'C:\Users\DELL\Downloads\Tornado_Siren_II-Delilah-0.mp3')
    #break
```

```
    else:
        print("No Danger")
       #break
    cv2.imshow("image",frame)
    if cv2.waitKey(1) & 0xFF == ord('a'):
        break

video.release()
cv2.destroyAllWindows()
```

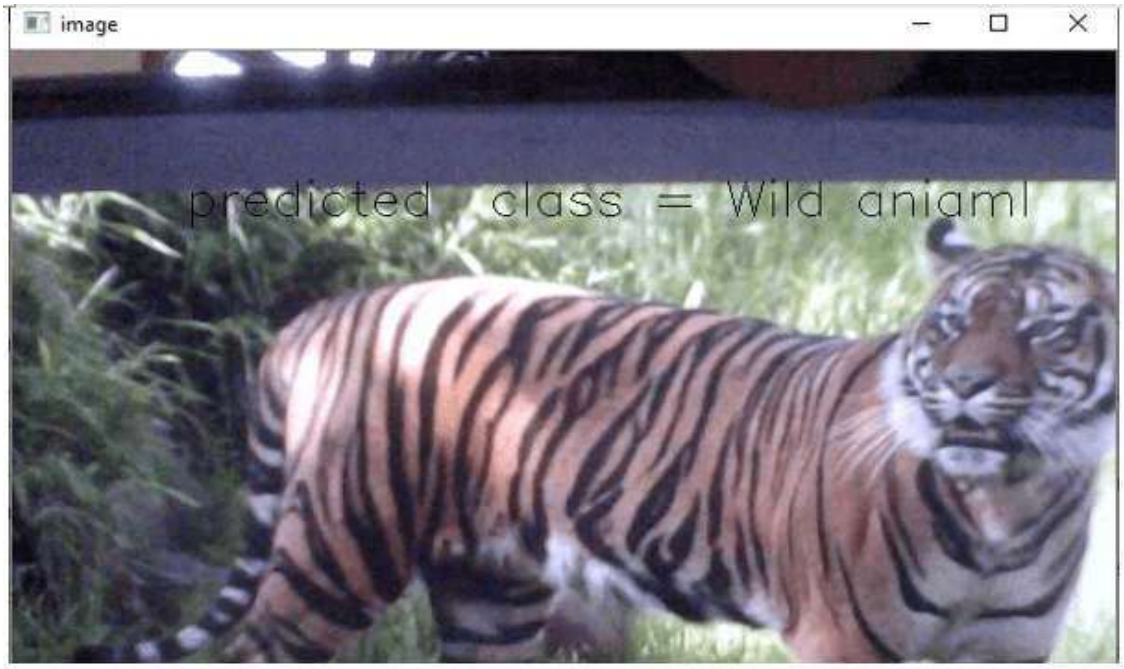
5. FLOWCHART



6. RESULT

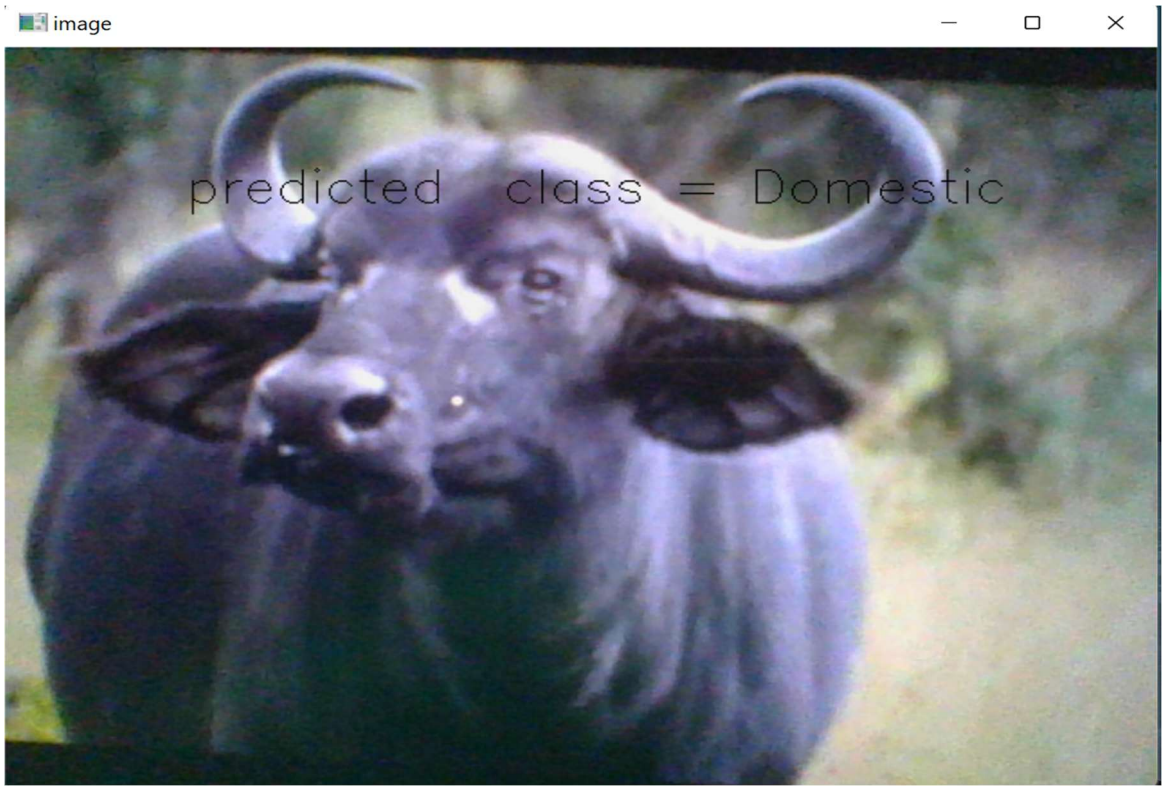
Once you run the above code, another window opens which displays the live stream of the video.

- Place an Wild Animal Image in front of the laptop camera
- You can see the label of the predicted animal on the window as shown below



- A buzzer sound will be played as soon as it detects an animals
- At the same time an SMS is sent to the mobile number

- Place a Domestic Animal Image in front of the laptop camera
- You can see the label of the predicted animal(i.e Domestic) on the window as shown below



7. ADVANTAGES & DISADVANTAGES

Advantages

- More datasets and images can be used to train for a more accurate outcome
- Predicts the wild animals easily.

Disadvantages

- The accuracy for detecting some images is low due to the limited quantity/quality of photos in the dataset, but this may easily be increased by changing the dataset.
- If dataset is not large, this model do not predict correctly in some cases.

9. CONCLUSION

The model classifies animals efficiently with a good number of accuracy. It detects the wild animals and notifies the tribal people to save their lives.

10. FUTURE SCOPE

This model can be used to detect animals in backyard, agriculture fields and in different areas by changing the dataset.

11.BIBILOGRAPHY

Anaconda Installation : https://www.anaconda.com/products/distribution
https://www.youtube.com/watch?v=5mDYijMfSzs

CNN Reference: https://www.youtube.com/watch?v=umGJ30-15_A&t=12s

ANN Reference: https://www.youtube.com/watch?v=fv6Qll3laUU

OpenCV Reference: https://learnopencv.com/read-write-and-display-a-video-using-opencv-cpp-python/

Image Data Generator Refernce: https://learnopencv.com/read-write-and-display-a-video-using-opencv-cpp-python/

Load Data From Directory Reference:
https://keras.io/api/preprocessing/image/#imagedatasetfromdirectory-function

Layers link : https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add

Read Frames Using Opencv:
https://www.youtube.com/watch?v=YNKo11c3EX0

Twilio Refernce: https://www.youtube.com/watch?v=knxlmCVFAZI