Low Level Design (LLD)
Property Image Classification

Revision Number: 1.0
Last date of revision: 07/12/2022

Shubham Kumar Chaturvedi

Document Version Control

Date Issued	Version	Description	Author
01/12/2022	1.0	First Draft for image classification	Shubham Kumar Chaturvedi

Contents

Abstract

The classification of image dataset at the time of uploading in “pataa” account creation (image/property). Dataset created by user, targeted areas are valid or not.Therefore continuous monitoring of regions/Image is mandatory. An Automatic/Api based data control System can prove to be a solution to above mentioned problems.

Automatic Image Counter Control can also help in drawing inferences from the recorded data.

1 Introduction

1.1 Why this Low-Level Design Document?

The purpose of this document is to present a detailed description of the Deep classification(Machine learning model) of images created by users during “pataa” creation. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli. This document is intended for both the stakeholders and the developers of the system and will be proposed to the higher management for its approval.

- Validate Image/Property of “pataa” account history, make future plans , image collection and test results .
- Allow access to evidence-based tools that providers can use to make decisions about a “pataa” account image updation/creation.

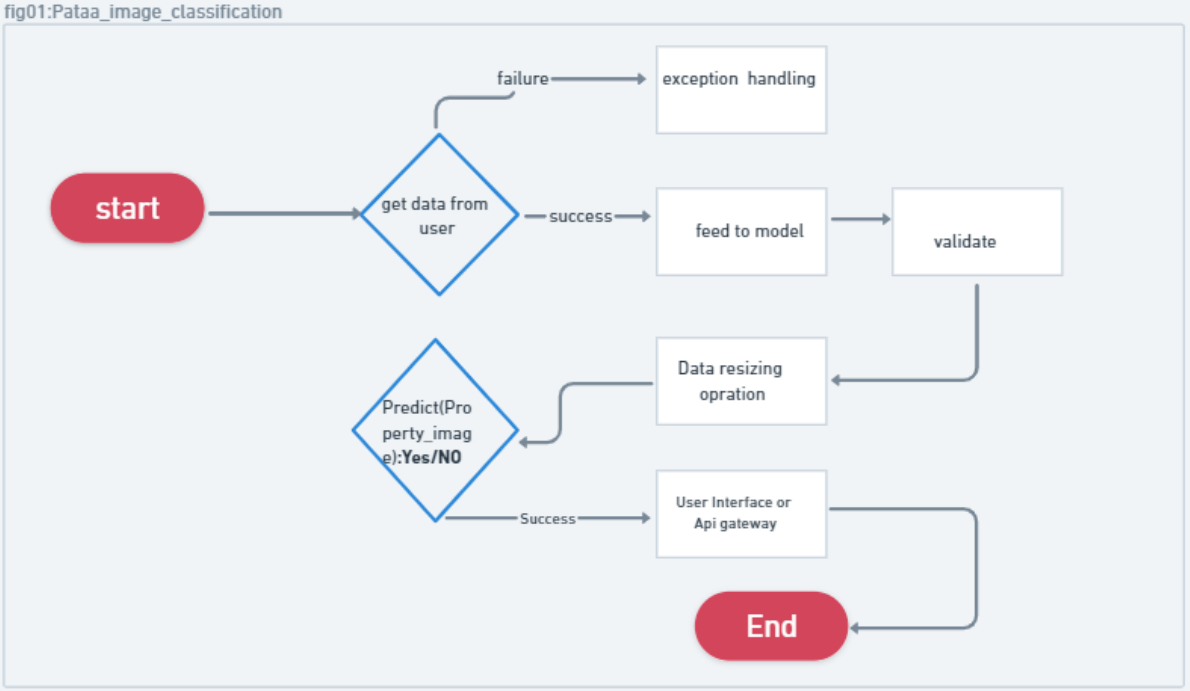
- Automate and streamline image provider workflow.

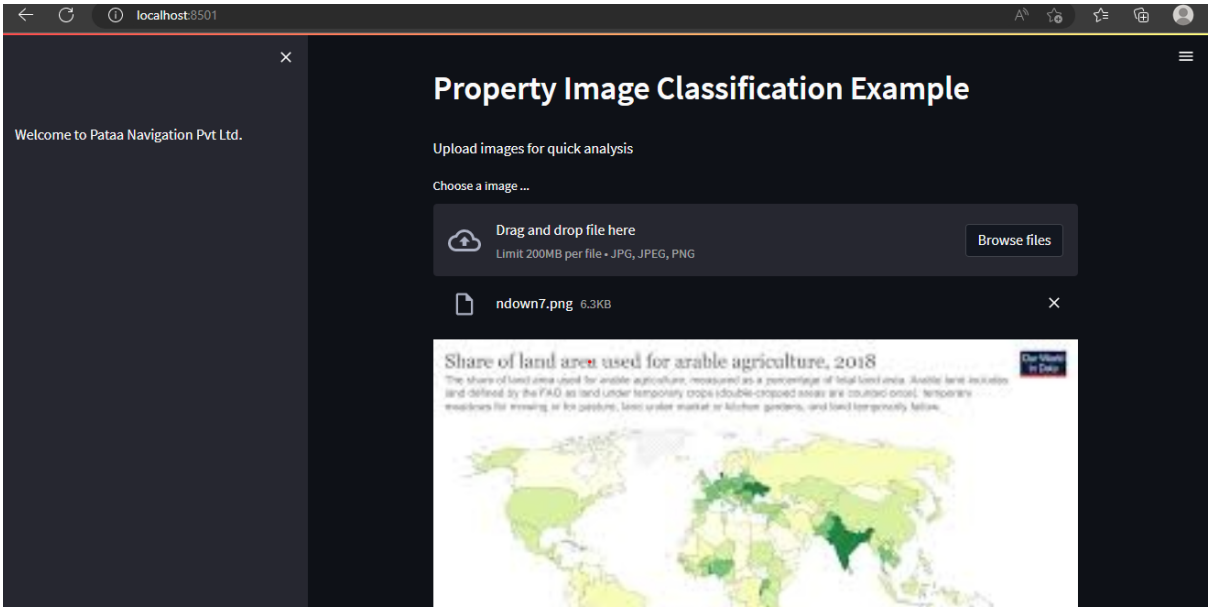
The Property Image Classification model contains information, such as:

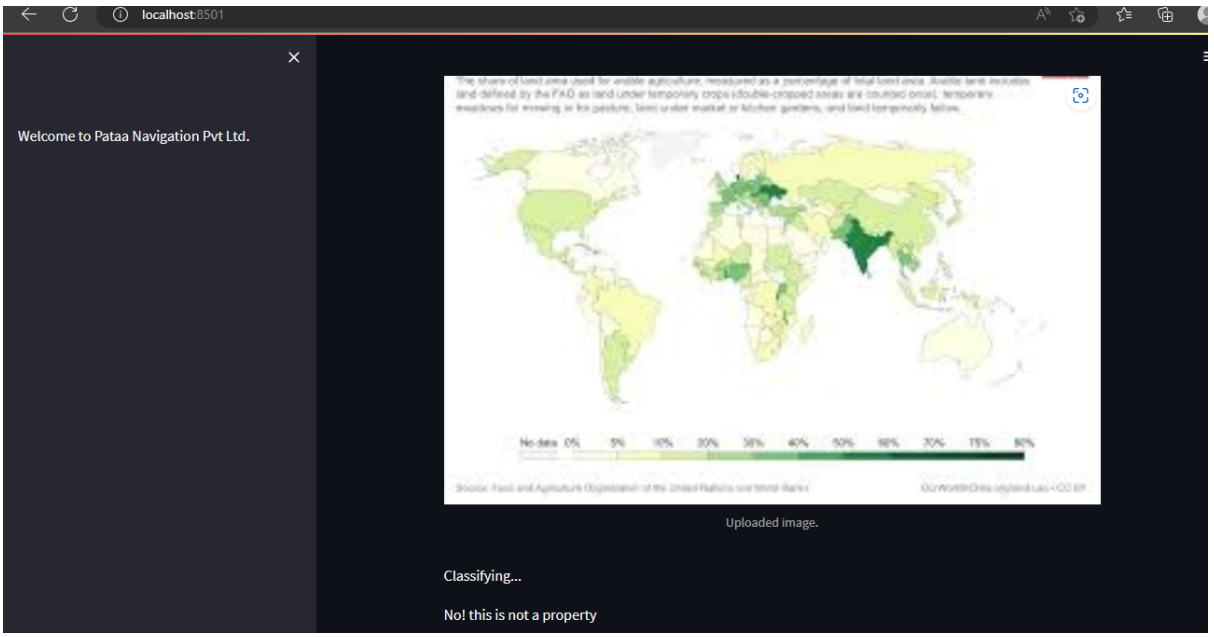
- Image Validation and classification.

  Trained model to predict Image class: Model Training >

fig01:Pataa_image_classification







This project shall be delivered in two phases:

Phase 1: All the functionalities with API and Integration with “pataa” app and packages.

Phase2: Integration of UI or API to all the functionalities with front end technology and “Pataa App” for testing and demo.

1.2 Scope

This software system will be a part of application development for mobile as well as web. This system will be designed to detect the Users uploaded image at earliest for better management, improved interventions, and more efficient resource allocation using previous “pataa” records available. More specifically, Early detection of any preventable causes is important for better decision/ management. This system is designed to predict the use case from account information such as account history,Uploaded Images, research results, procedures and advanced analytics.

1.3 Constraints

We will only be selecting a few of the important areas for the time being.

1.4 Risks

Document specific risks that have been identified or that should be considered. Sometimes it might happen when prediction would not be 100 percent accurate at that time model performance issue may occur but it can be solved by retaining the model from time to time.

1.5 Out of Scope

Delineate specific activities, capabilities, and items that are out of scope for the project.

2 Technical specifications

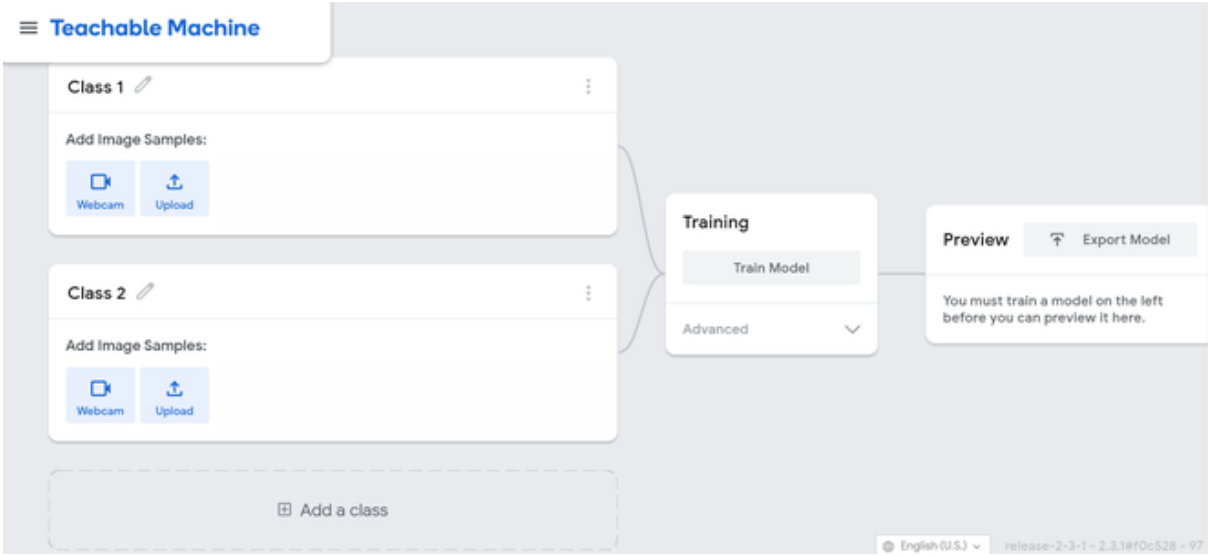
2.1 Dataset

Source	Finalized	Source
Image data set collected and created by some open source data library.	class1: 1500 dataset Property. class2: 1500 dataset Non-Property.	NA

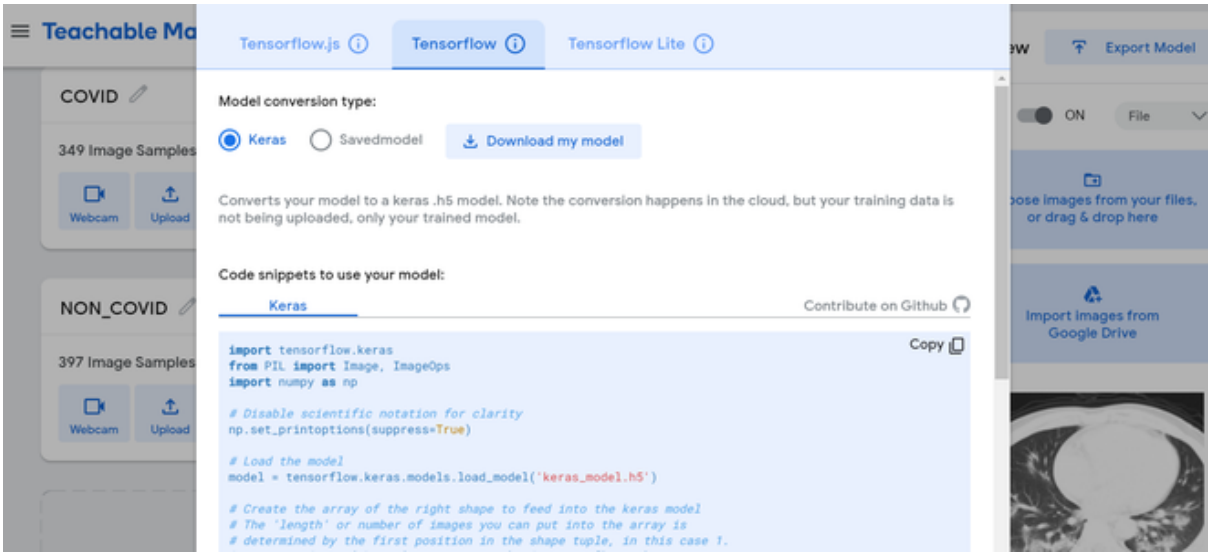
2.1.1 Pataa count dataset overview

Consists of 2 different Labels, the Property/Non-Property class consists of the Images information and most importantly we have the historic data of a Users in the form of image dataset.

- Training using teachable machine:



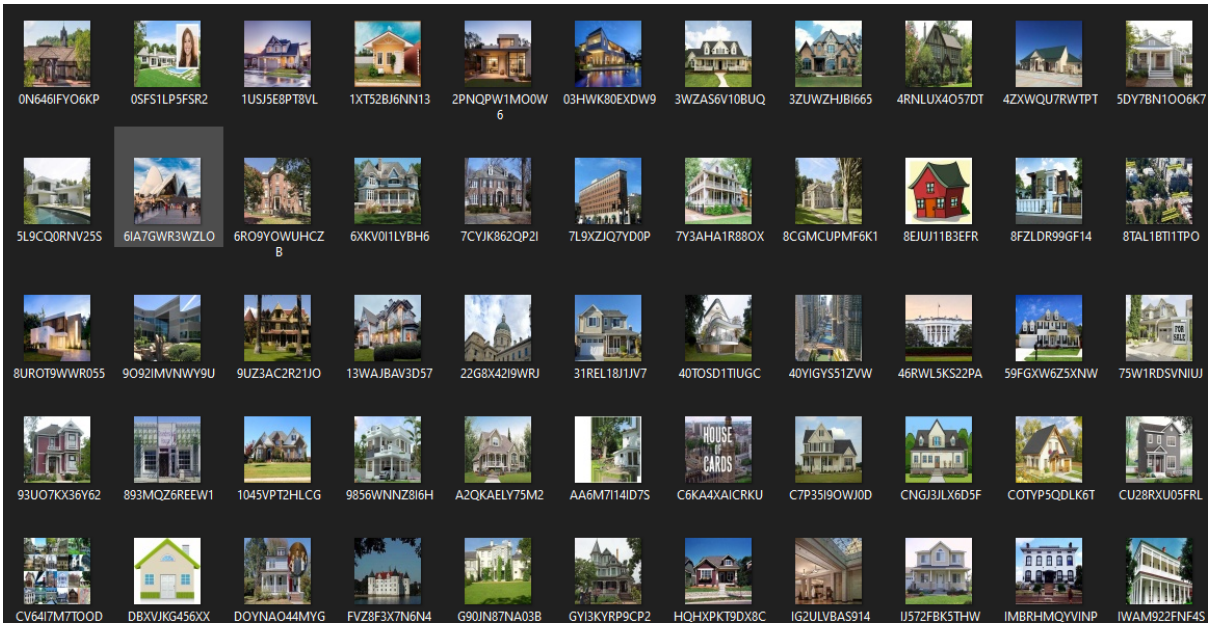
- Trained Model:



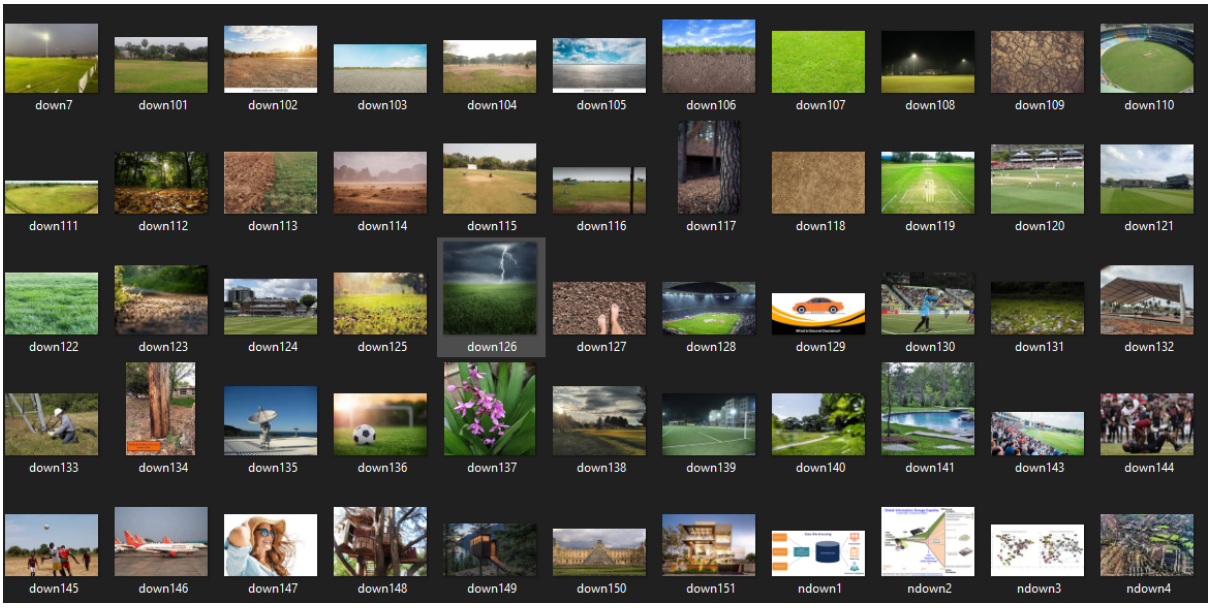
- User interface and stored procedure:

2.1.2 Input schema

1. Property Dataset:



2. Non-Property Dataset:



2.2 Predicting/get information

- The system displays to choose the image to load.
- The User chooses the target area/image by clicking one of the available areas.
- The User selects the upload button.
- The system presents the set of information required from the user.
- The system processes the internal module and predicts the binary result in the form of Property and Non-Property.
- After successful validation data can be stored in an internal database to image property against the user account.

2.3 Logging

- We are not going to log every activity done by the user in the model prediction phase although all previous logs can be stored as user signs . It is a simple prediction model to show required information.

2.4 Database

System needs to store every request into the database and we need to store and process it in such a way that it is easy to retrain the model as well.

1. The system stores each and every data given by the user or received on request to the database/API. Database is in any Unstructured database.

2.5 Deployment

1. AWS 2.Android

3 Technology stack

Front End	Api call or web for demo used streamlit and React js.
Backend	Trained Model using google teachable Machine as well as keras library.
Database	User Input
Deployment	AWS,android.

4 Proposed Solution

refer: dashboard · Streamlit http://localhost/8501

Based on the actual research paper, if we are using history of the users to predict the future then we might want to consider using custom model training for better prediction and modification. However, drawing a baseline in the form of some Machine Learning algorithm would be helpful. Why make a baseline model important? Well, to compare the performance of our actual model, let say ResNet-50 in this case, is very important to ascertain that we are in the right direction as if performance of ResNet-50 is not better than the baseline model then there is no point in using ResNe-50.

1. Baseline Model: MobileNet, since this is a classification problem.
2. Actual model: Google teachable machine model.

5 User I/O workflow and stored procedure :

1. Code Snapshot:

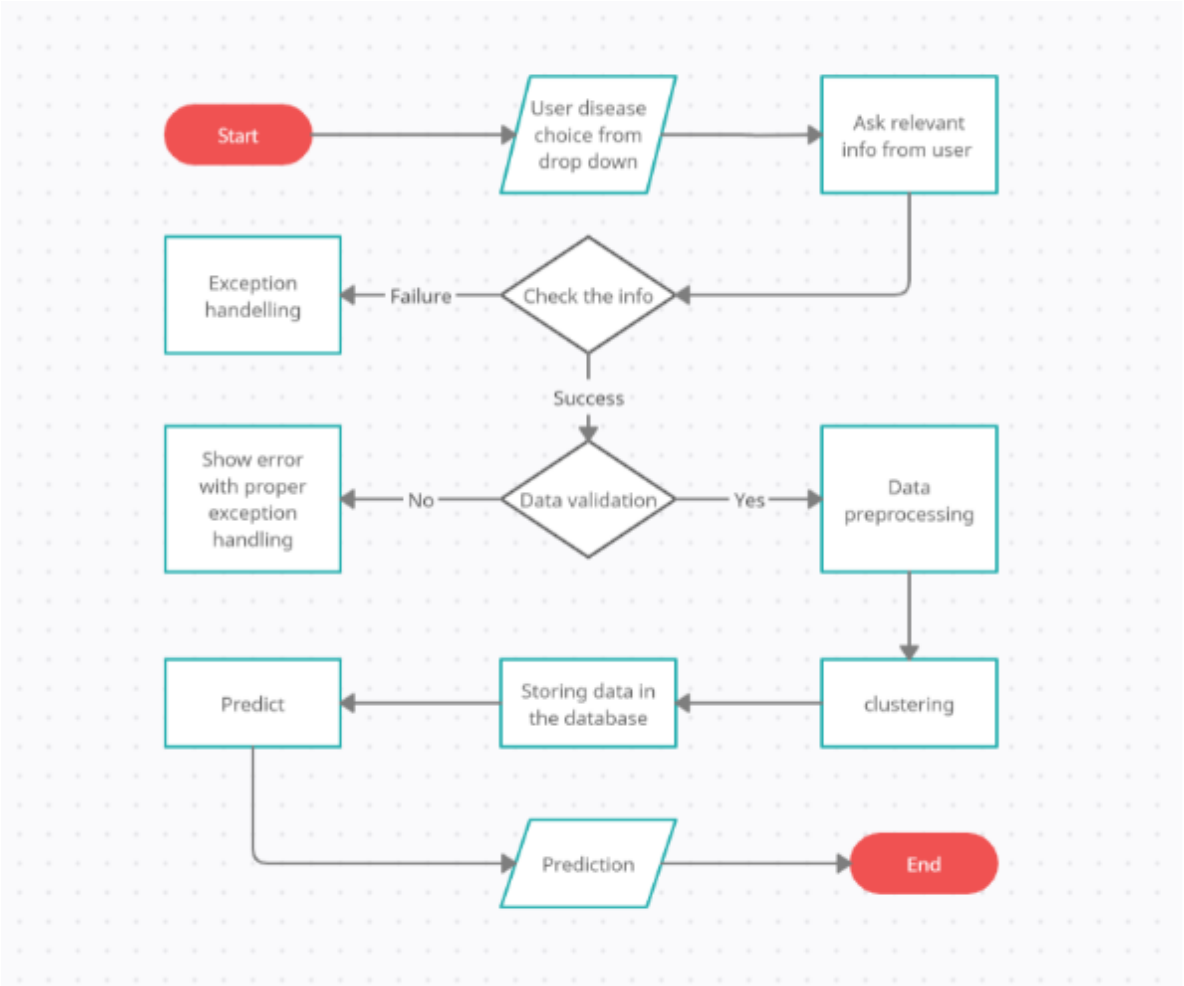
```python
import numpy as np
from flask import Flask, request
from flask_cors import CORS
from keras.models import load_model
from PIL import Image, ImageOps


app = Flask(__name__)
CORS(app)


@app.route('/upload', methods=['POST'])
def upload():
    # Load the model
    model = load_model('keras_model.h5', compile=False)

    # Create the array of the right shape to feed into the keras model
    # The 'length' or number of images you can put into the array is
    # determined by the first position in the shape tuple, in this case 1.
    data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)
    # Replace this with the path to your image
    image = Image.open(request.files['file'])
    # resize the image to a 224x224 with the same strategy as in TM2:
    # resizing the image to be at least 224x224 and then cropping from the center
    size = (224, 224)
    image = ImageOps.fit(image, size, Image.ANTIALIAS)

    # turn the image into a numpy array
    image_array = np.asarray(image)
    # Normalize the image
    normalized_image_array = (image_array.astype(np.float32) / 127.0) - 1
    # Load the image into the array
```



6 Exceptional scenarios

Step	Exception	Mitigation	Module
19th Sep 2022	1.0	First Draft	Shubham Kumar Chaturvedi

7 Test cases

Test case	Steps to perform test case	Module	Pass/Fail
1.		Second Draft	

8 Key performance indicators (KPI)

- Comparison of accuracy of model prediction and actual events.
- classify the image as property or not .
- Target User base in particular regions.
- Reduce redundancy and raw data localisation.