

VEGETATION ENCROACHMENT RISK ASSESSMENT NEAR ELECTRIC POLES USING IMAGE PROCESSING

Project Report submitted by

SHELDON MATHIAS

(4NM20CS166)

YASHASWI P

(4NM20CS216)

SHRI HARI S

(4NM20CS177)

RAJATH

(4NM21CS413)

Under the Guidance of
Mrs. SHILPA KAREGOUDAR

Asst. Professor Gd-II,

Dept. of CSE,
NMAMIT

*In partial fulfillment of the requirements for the award
of the Degree of*

**Bachelor of Engineering in Computer Science and
Engineering**

from

Visvesvaraya Technological University, Belagavi

Department of Computer Science and Engineering
NMAM Institute of Technology, Nitte - 574110
(An Autonomous Institution affiliated to VTU, Belagavi)

APRIL 2024

CERTIFICATE




DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

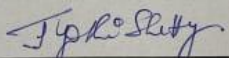
Certified that the project work entitled
"Vegetation Encroachment Risk Assessment near Electric Poles
using Image Processing "
is a bonafide work carried out by
Sheldon Mathias (4NM20CS166) Shri Hari S (4NM20CS177)
Yashaswi P (4NM20CS216) Rajath (4NM21CS413)
in partial fulfilment of the requirements for the award of
Bachelor of Engineering Degree in Computer Science and Engineering
prescribed by Visvesvaraya Technological University, Belagavi
during the year 2023-2024.

It is certified that all corrections/suggestions indicated for Internal Assessment have been
incorporated in the report deposited in the departmental library.

The project report has been approved as it satisfies the academic requirements in respect of the
project work prescribed for the Bachelor of Engineering Degree.



Signature of the Guide



Signature of the HOD


Signature of the Principal

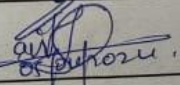
Semester End Viva Voce Examination

Name of the Examiners

Signature with Date

1. Anuritha A Nayak
2. Sareitha Shetty





ACKNOWLEDGEMENT

The satisfaction that accompanies the completion of any task would be incomplete without the mention of all the people, without whom this endeavor would have been a difficult one to achieve. Their constant blessings, encouragement, guidance and suggestions have been a constant source of inspiration.

First and foremost, my gratitude to my project guide, **Mrs. Shilpa Karegoudar** for her constant guidance throughout the course of this project Phase-2 and for the valuable suggestions.

I also take this opportunity to express a deep sense of gratitude to the project coordinators for their valuable guidance and support.

I acknowledge the support and valuable inputs given by, **Dr. Jyothi Shetty** Head of the Department, Computer Science and Engineering, NMAMIT, Nitte.

My sincere thanks to our beloved principal, **Dr. Niranjan N Chiplunkar** for permitting us to carry out this project at our college and providing us with all needed facilities.

Finally, thanks to staff members of the Department of Computer Science and Engineering and our friends for their honest opinions and suggestions throughout the course of our project.

Sheldon Mathias(4NM20CS166)
Shri Hari S(4NM20CS177)
Yashaswi P (4NM20CS216)
Rajath (4NM21CS413)

ABSTRACT

In contemporary urban landscapes, the coexistence of urban trees and power lines poses a multifaceted challenge characterized by the escalating complexity of managing entanglements. Rapid urbanization, coupled with the imperative for climate adaptation and environmental services, necessitates a comprehensive understanding of the intricate interplay between these vital elements. The overarching problem lies in the potential safety hazards, service disruptions, and ecological consequences arising from tree-power line entanglements, demanding innovative and automated solutions for effective detection and management. By using modern day techniques such as Image Processing and Deep learning, we plan to identify the regions that have vegetation-wire entanglement so that necessary action can be taken.

Keywords: Image Processing, Deep Learning, Vegetation Encroachment, Electric power lines

CONTENT

CHAPTER NO.	CHAPTER NAME	PAGE
1	INTRODUCTION	8 - 9
2	LITERATURE SURVEY	10 - 12
2.2	LITERATURE SURVEY SUMARRY	13 - 15
3	OBJECTIVES	16
4	PROBLEM DEFINITION	17
5	SYSTEM AND SOFTWARE REQUIREMENTS	18
6	SYSTEM DESIGN	19-22
7	IMPLEMENTATION	23-35
8	RESULTS	36-39
9	CONCLUSION	40
10	REFERENCES	41

LIST OF FIGURES

Figure no.	Description	Page No.
1	system design	19
2	feature extraction	23
3	libraries	24
4	defining model	24
5	model testing	25
6	saving model	25
7	model loading	25
8	line and feature processing	26
9	image downloader libraries	27
10	download tile function	28
11	download preferences	29
12	the merging function	30
13	database function	31
14	export csv function	31
15	havesine distance function	32
16	distance calculation function	33
17	get nearest line segment function	34
18	pixel mapping function	34
19	draw square function	35
20	training and accuracy loss and validation graph	37

21	accuracy and loss results	37
22	confusion matrix on test result	37
23	Predicted images	38
24	figure showing encroachment	38
25	figure without encroachment	39

LIST OF TABLES

Table no.	Description	Page No.
1	Literature Survey	9-10

CHAPTER 1

INTRODUCTION

Electric power distribution systems play a vital role in supplying energy to communities, industries, and households. However, the uninterrupted flow of electricity is contingent upon the efficient operation and maintenance of the infrastructure, particularly the electric poles that form the backbone of the distribution network. One critical challenge faced in this regard is the encroachment of vegetation in the vicinity of these poles. Vegetation encroachment poses a significant threat to the reliability and safety of the power distribution system. Overgrown vegetation can lead to power outages, equipment damage, and even pose fire hazards. Traditional methods of manual inspection are not only labor-intensive but also time-consuming. To address these challenges, this report presents a novel approach leveraging image processing techniques for the automated assessment of vegetation encroachment risks near electric poles.

The objective of this project is to develop an efficient and accurate system that can analyze images captured in the field, identify the presence of vegetation, and assess the level of risk associated with its proximity to electric poles. Through the integration of advanced image processing algorithms, this approach aims to enhance the overall reliability of power distribution systems while reducing the need for extensive manual inspections. This report outlines the methodology employed, including data collection processes, image processing techniques, and risk assessment methodologies. The findings of this study provide valuable insights into the feasibility and effectiveness of utilizing image processing for vegetation management in the context of power distribution infrastructure.

As we delve into the details of our methodology and results, it becomes apparent that the integration of image processing technologies offers a promising avenue for proactive vegetation management, ensuring the resilience and sustainability of electric power distribution systems.

Electric power distribution systems are susceptible to disruptions caused by various external factors, and vegetation encroachment emerges as a recurring challenge with the potential for severe consequences. The increasing demand for electricity, coupled with changing climate patterns, has heightened the importance of robust infrastructure management. Traditional methods of vegetation monitoring often fall short in providing timely and comprehensive assessments. This project responds to the pressing need for a more proactive and technologically advanced approach to vegetation management near electric poles. By harnessing the capabilities of image processing, we aim to create a system that not only identifies the presence of vegetation but also evaluates the associated risk levels. This transition towards automation not only enhances the efficiency of the inspection process but also allows for more frequent and consistent monitoring.

The integration of image processing techniques promises a paradigm shift in how we perceive and address vegetation encroachment risks. The ability to analyze large volumes of image data rapidly empowers utility companies to make informed decisions in real-time, mitigating potential risks before they escalate. As we embark on this exploration of image processing applications, the intention is to contribute to the ongoing dialogue on innovative solutions for enhancing the resilience and reliability of power distribution networks. Through a detailed examination of our methodology, results, and subsequent discussions, this report endeavors to showcase the potential impact of image processing in the realm of vegetation encroachment risk assessment. By doing so, we not only advance the field of infrastructure management but also pave the way for sustainable and adaptive approaches to challenges that stand at the intersection of technology and utility services.

CHAPTER 2

LITERATURE SURVEY

Author	Objective	Flaws
Yangu Chen , Jiayuan Lin, Xiaohan Liao [1]	Early detection of tree encroachment in high voltage powerline corridor using growth model and UAV-borne LiDAR	Limited generalizability stemming from a focus on specific case studies and potential methodological constraints, such as data quality and algorithmic complexity, which could impact the validity and reliability of the findings.
Fathi mahdi elsiddig haroun , Siti noratqah mohamad deros and Norashidah md din [2]	Detection and Monitoring of Power Line Corridor From Satellite Imagery Using RetinaNet and K-Mean Clustering	Scalability and robustness of the proposed method when applied to diverse geographical regions or varying environmental conditions.
Levente J Klein ,Wang Zhou ,Hendrik F Hamann,Conrad M Albrecht,Carlo Siebenschuh,Siyuan Lu,Sharathchandra Pankanti [3]	N-dimensional geospatial data and analytics for critical infrastructure risk assessment	The paper lacks validation studies, unclear methodology, incomplete literature review, and ignores ethical and environmental considerations.
Abdul Qayyum, Aamir Saeed Malik, Mohamad Naufal ,Mohamad Saad, Mahboob Iqbal, Rana Fayyaz Ahmad, Tuan Ab Rashid Bin Tuan	Monitoring of Vegetation Near Power Lines Based on Dynamic Programming using Satellite Stereo Images	The paper's proposed technique for monitoring vegetation near power lines lacks thorough comparison, validation, error analysis, implementation discussion, and future research direction, weakening its credibility.

Abdullah,Ahmad Quisti Ramli [4]		
Rabab Abdelfattah, Xiaofeng Wang, Song Wang [5]	An Aerial-Image Dataset for Detection and Segmentation of Transmission Towers and Power Lines	The paper lacks thorough comparative analysis, provides limited evaluation metrics, and insufficiently describes the annotation process, weakening its contribution to the field.
Zhengrong Li Yuee Liu Ross Hayward Jinglan Zhang Jinhai Cai [6]	Knowledge-based Power Line Detection for UAV Surveillance and Inspection Systems	The paper lacks comparisons, thorough metrics, parameter details, real-world considerations, generalizability, alternative methods, dataset access, and computational complexity discussion.
Shuaiang Rong,Liang Du,Zuyi li,Shiwen yu [7]	Intelligent Detection of Vegetation Encroachment of Power Lines with Advanced Stereovision	The effectiveness of the proposed framework lacks validation with real-world data, which may undermine its reliability in practical deployment.
Nina m. singer, Vijayan k. asari [8]	DALES Objects: A Large Scale Benchmark Dataset for Instance Segmentation in Aerial Lidar	The DALES Objects paper lacks comparison, quality assurance, generalization solutions, diversity exploration, evaluation, accessibility, and documentation.

Seulbi Lee and Youngjib Ham [9]	Measuring the Distance between Trees and Power Lines under Wind Loads to Assess the Heightened Potential Risk of Wildfire	The study may overlook important variations in tree behavior and ignition risk due to factors such as tree species and moisture content, potentially limiting the generalizability of its findings.
Yunping Chen,n , Yang Li , Huixiong Zhang , Ling Tong, Yongxing Cao, Zhihang Xue [10]	Automatic power line extraction from high resolution remote sensing imagery based on an improved Radon transform	The paper fails to provide a thorough review of prior research on power line extraction from satellite imagery, potentially neglecting valuable insights and comparisons.

2.2 LITERATURE SURVEY SUMMARY

Paper 1: Yangyu Chen , Jiayuan Lin, Xiaohan Liao

This study used UAV-borne LiDAR and the Richards growth model to detect tree encroachments in high voltage powerline corridors. UAV-borne LiDAR was effective in accurately measuring tree heights. The Richards model proved to be the best fit for modeling tree growth. A bounding box-based algorithm greatly improved detection efficiency, and a 50-meter unit length was recommended for segmenting powerline buffers. The study suggested targeted inspections to enhance efficiency but noted challenges in species recognition and environmental factors affecting powerline positions.

Paper 2: Fathi mahdi elsiddig haroun , Siti noratiqah mohamad deros and Norashidah md din

The paper introduces a new method for monitoring power line corridors using satellite imagery, addressing the challenge of detecting transmission towers (TTs) and vegetation encroachment (VE) efficiently and cost-effectively. Traditional methods like UAVs and airborne photography are expensive and impractical for wide-area monitoring.

Paper 3: Paper 3: Levente J Klein ,Wang Zhou ,Hendrik F Hamann,Conrad M Albrecht,Carlo Siebensschuh,Siyuan Lu,Sharathchandra Pankanti

A combination of LiDAR data with aerial and radar imagery allows to track dynamic seasonal growth of vegetation around critical infrastructure such as power lines. It presents a general framework that integrates tree identification and growth assessment around power lines with the goal to identify locations of high risk where trees potentially cause power outages

Paper 4: Abdul Qayyum, Aamir Saeed Malik, Mohamad Naufal ,Mohamad Saad, Mahboob Iqbal, Rana Fayyaz Ahmad, Tuan Ab Rashid Bin Tuan Abdullah,Ahmad Quisti Ramli

In this paper, a novel technique for depth estimation of vegetation/trees is proposed. In the study, Dynamic Programming is employed on stereo satellite images to determine depth of

vegetation and trees. The experimental results on QuickBird imagery exhibit that the proposed technique performs better compared to block matching technique in terms of accuracy.

Paper 5: Rabab Abdelfattah, Xiaofeng Wang, Song Wang

The TTPLA dataset is introduced for aerial object detection and segmentation, with a focus on transmission towers (TTs) and power lines (PLs). It addresses challenges like diverse views, background complexity, and crowded objects. The dataset is created from aerial videos captured by UAVs, and it follows an instance segmentation annotation policy for precise object recognition.

Paper 6: Zhengrong Li Yuee Liu Ross Hayward Jinglan Zhang Jinhai Cai

A knowledge-based power line detection method for a vision based UAV surveillance and inspection system. A PCNN filter is developed to remove background noise from the images prior to the Hough transform being employed to detect straight lines. Finally knowledge based line clustering is applied to refine the detection results.

Paper 7: Shuaiang Rong,Liang Du,Zuyi li,Shiwen yu

The paper proposes an intelligent detection framework for monitoring vegetation encroachment on power lines, combining deep learning and advanced stereovision techniques. It aims to enhance grid stability by detecting potential circuit failures caused by vegetation growth. The framework consists of three modules: vegetation region detection using Faster R-CNN, power line detection via the Hough transform, and vegetation encroachment detection with an advanced stereovision algorithm. While promising, the framework lacks comprehensive evaluation, scalability considerations, and validation with real-world data, raising questions about its practical reliability.

Paper 8: Nina m. singer, vijayan k. asari

The excerpt discusses the significance of the "DALES Objects" dataset, a large-scale dataset for instance segmentation in aerial LiDAR data. It emphasizes the limited availability of instance segmentation datasets and highlights the dataset's scope, which includes semantic and instance segmentation labels, original point data in UTM projection, and intensity information. The dataset aims to support research in 3D deep learning for both LiDAR and outdoor scenes.

Paper 9: Seulbi Lee and Youngjib Ham

In this paper, investigated the critical wind speed that heightens the risk of wildfires by calculating the distance between trees and wires. To conduct this study, we used airborne LiDAR data collected from Sonoma County in northern California and analyzed the behavior of a sample tree having a height of 19.2 m under wind loads. Our analysis showed that the main factor determining tree deflection is the ratio of the tree height to the trunk diameter.

Paper 10: Yunping Chen,n , Yang Li ,Huixiong Zhang ,Ling Tong,Yongxing Cao, Zhihang Xue

This paper discusses the need for research in extracting high voltage power lines from high-resolution satellite images. It introduces a new algorithm called Cluster Radon Transform (CRT) for this purpose, emphasizing its anti-noise capabilities and efficiency. The passage concludes by noting the potential for further research on extracting curved and broken power lines.

CHAPTER 3

OBJECTIVES

Key objectives of the project include:

1. Collect diverse satellite images encompassing both dense and non-dense vegetation to ensure comprehensive data acquisition for analysis
2. Develop a Machine Learning Model
3. Analyze and validate the model for existing data
4. Integrate this model with Google Earth API such that the image of the region of interest can be directly extracted from Google Earth
5. Map a power line in a region and identify locations where there is vegetation encroachment present near power lines model to classify each region of interest, and visualize the results by overlaying them on the satellite imagery.

CHAPTER 4

PROBLEM DEFINITION

In the dynamic urban landscape, the intricate interaction between urban trees and power lines gives rise to a significant problem. As cities expand, the challenge is to manage the entanglements between these elements, leading to safety concerns, service interruptions, and ecological repercussions. Recognizing the need for a nuanced understanding of these intersections, this research focuses on developing innovative solutions leveraging image processing and machine learning techniques. By exploring advanced technologies, the project aims to detect tree-power line entanglements efficiently.

To address this pressing issue, this research initiative is dedicated to pioneering an innovative solution centered around the application of image processing techniques for vegetation encroachment risk assessment near electric poles. By harnessing the capabilities of digital imagery, the primary objective is to automate the detection and analysis of vegetation growth in close proximity to electric infrastructure. This approach promises to empower utility companies and municipal authorities to efficiently identify areas of concern and prioritize maintenance efforts accordingly.

CHAPTER 5

SYSTEM AND SOFTWARE REQUIREMENTS

1. Operating System: Windows 11 (64-bit)
2. Processor: Intel Core i5 or equivalent
3. RAM: Minimum 8 GB
4. Storage: Minimum 256 GB
5. Graphics Card: Dedicated GPU with CUDA support
6. Python: Version 3.7 or later.
7. External libraries: Libraries such as TensorFlow, Scikit-learn, Open cv for the purpose of feature extraction for machine learning and deep learning.

CHAPTER 6

SYSTEM DESIGN

Our system design orchestrates machine learning algorithms, image processing techniques, and feature extraction methods to analyze vegetation density near power lines using satellite imagery. By amalgamating these components, we create a robust framework capable of accurately classifying dense vegetation encroachment. This holistic approach ensures efficient preemptive maintenance strategies, thereby bolstering the resilience and security of power line infrastructure.

Flowchart:

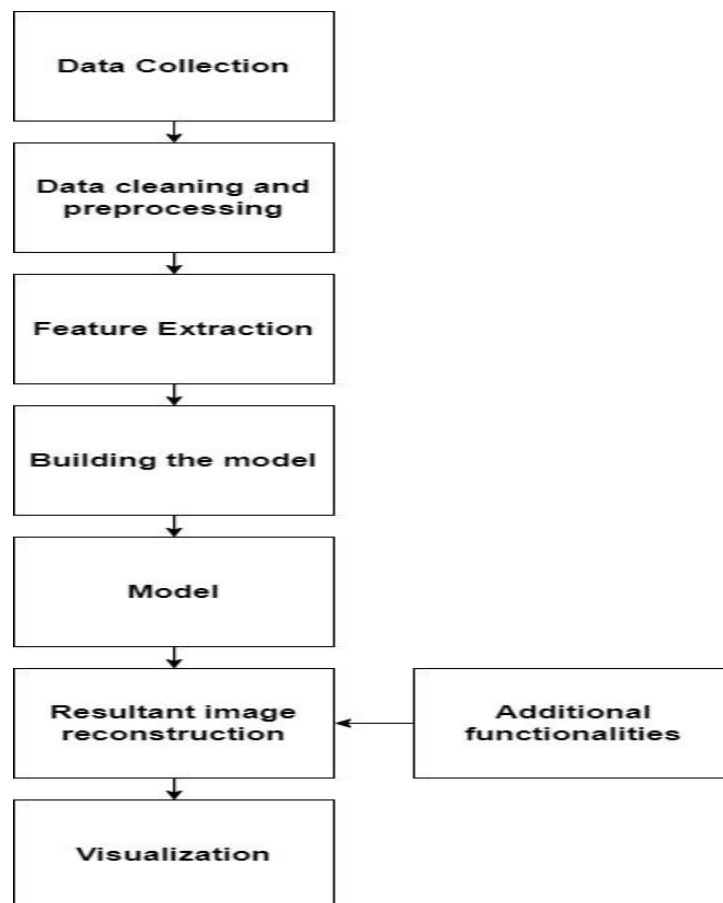


Figure 1: System design

Data collection and preprocessing

The initial phase of the project involves collecting images from Google Earth and processing them into smaller chunks of size 32x32. These chunks are manually classified as either Dense or Non-Dense vegetation, providing labeled data for subsequent analysis. This manual classification process ensures that the dataset accurately represents the diversity of vegetation types and densities present in the images. Once classified, the chunks are stored in the system, forming the basis of the dataset for further analysis and model development.

Feature Extraction

After dataset preparation, the next step is feature extraction, where various features of the image chunks are computed and stored in a structured format. These features include statistical measures such as Mean, Variance, and Standard Deviation, which provide insights into the characteristics of the vegetation present in each chunk. By extracting these features, the dataset is enriched with quantitative information that can be used to train machine learning models effectively. Texture features such as energy, ASM (Angular Second Moment), contrast, and homogeneity, which provide insights into the spatial arrangement of pixel intensities. By operating in different color spaces (BGR, HSV, LAB), the function ensures that a comprehensive range of features is captured, accounting for variations in color representation and perception.

Data Splitting

With the dataset prepared and features extracted, the data is then split into separate training and testing sets in the ratio of 80% for training and 20% for testing. This data splitting ensures that the models are trained on a sufficient amount of data while also allowing for independent evaluation to assess their performance accurately. By partitioning the data in this manner, the models can be trained on one subset and evaluated on another, enabling robust validation of their predictive capabilities.

Machine Learning model

In this phase, various machine learning models are developed using the training data to classify vegetation chunks as Dense or Non-Dense. These models encompass a range of algorithms, including but not limited to Decision Trees, Random Forests, and Neural

Networks. Each model is trained on the training dataset using the extracted features, aiming to learn patterns and relationships that distinguish between Dense and Non-Dense vegetation.

Model Evaluation

Once the models are trained, they are evaluated using the testing dataset to assess their performance and identify the most effective model for the task. Evaluation metrics such as accuracy, precision, recall, and F1-score are calculated to measure the models' ability to correctly classify vegetation chunks. Additionally, techniques like cross-validation may be employed to ensure the robustness of the evaluation process. The model that demonstrates the highest performance metrics on the testing dataset is selected for use in subsequent phases of the project, where it will be deployed for prediction and further analysis.

Image reconstruction and additional functionalities

Following the prediction on new data using the trained DNN model, the system proceeds with the reconstruction of the resultant image, leveraging the predicted classifications for individual image chunks. Concurrently, it integrates additional functionalities, including a Satellite Imagery Downloader and utilities for distance calculation and pixel mapping. This integration enhances both data acquisition efficiency and analysis accuracy. Through the incorporation of these functionalities, the system achieves a seamless workflow, from data collection and preprocessing to model deployment and result visualization. The reconstructed image, highlighting areas of dense vegetation encroachment near power lines, is pivotal for providing actionable insights to decision-makers. By combining image reconstruction with these supplementary features, the system offers a comprehensive solution for vegetation density analysis and preemptive maintenance efforts in power line infrastructure management.

Image visualization

To visualize the predicted areas of dense vegetation encroachment near power lines, we integrate the output of the trained DNN model with a visual representation of the satellite image. Leveraging the `draw_square` function, we mark areas identified as dense vegetation with squares overlaid onto the image. These squares serve as indicators of potential risk zones, aiding in preemptive maintenance planning. Additionally, we employ color-coding or other visual cues to distinguish between areas of dense vegetation and other regions, enhancing the

interpretability of the results. By combining the model's predictions with visual indicators, decision-makers gain actionable insights into vegetation density patterns. This integrated approach facilitates efficient identification and prioritization of areas requiring attention, streamlining vegetation management efforts and ensuring the reliability and safety of power line infrastructure.

CHAPTER 7

IMPLEMENTATION

Model

Feature Extraction

Feature extraction is a fundamental step in image processing and computer vision tasks. In this section of the code, the `extract_features` function is defined to extract a diverse set of features from input images. These features encompass statistical properties such as mean, standard deviation, and variance, which capture essential aspects of the image's color distribution.

Moreover, the function computes texture features such as energy, ASM (Angular Second Moment), contrast, and homogeneity, which provide insights into the spatial arrangement of pixel intensities. By operating in different color spaces (BGR, HSV, LAB), the function ensures that a comprehensive range of features is captured, accounting for variations in color representation and perception.

```
def extract_features(img):
    # Convert the image to different color spaces
    hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    lab_img = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)

    # Split the channels
    b, g, r = cv2.split(img)
    h, s, v = cv2.split(hsv_img)
    l, a, b_human = cv2.split(lab_img)

    # Calculate mean values
    mean_r, mean_g, mean_b = np.mean(r), np.mean(g), np.mean(b)
    mean_h, mean_s, mean_v = np.mean(h), np.mean(s), np.mean(v)
    mean_l, mean_a, mean_bhuman = np.mean(l), np.mean(a), np.mean(b_human)

    # Calculate standard deviations
    std_r, std_g, std_b = np.std(r), np.std(g), np.std(b)
    std_h, std_s, std_v = np.std(h), np.std(s), np.std(v)
    std_l, std_a, std_bhuman = np.std(l), np.std(a), np.std(b_human)

    # Calculate variances
    var_r, var_g, var_b = np.var(r), np.var(g), np.var(b)
    var_h, var_s, var_v = np.var(h), np.var(s), np.var(v)
    var_l, var_a, var_bhuman = np.var(l), np.var(a), np.var(b_human)
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    kurtosis_r, kurtosis_g, kurtosis_b = kurtosis(r), kurtosis(g), kurtosis(b)
    kurtosis_r_single = np.mean(kurtosis_r)
    kurtosis_g_single = np.mean(kurtosis_g)
    kurtosis_b_single = np.mean(kurtosis_b)

    # Other texture features
    glcm = feature.graycomatrix(gray_img, [1], [0], symmetric=True, normed=True)
    energy = feature.graycoprops(glcm, prop='energy')[0, 0]
    ASM = feature.graycoprops(glcm, prop='ASM')[0, 0]
    contrast = feature.graycoprops(glcm, prop='contrast')[0, 0]
    homogeneity = feature.graycoprops(glcm, prop='homogeneity')[0, 0]

    features_dict = {
        "mean_r": mean_r, "mean_g": mean_g, "mean_b": mean_b,
        "mean_h": mean_h, "mean_s": mean_s, "mean_v": mean_v,
        "mean_l": mean_l, "mean_a": mean_a, "mean_bhuman": mean_bhuman,
        "std_r": std_r, "std_g": std_g, "std_b": std_b,
        "std_h": std_h, "std_s": std_s, "std_v": std_v,
        "std_l": std_l, "std_a": std_a, "std_bhuman": std_bhuman,
        "var_r": var_r, "var_g": var_g, "var_b": var_b,
        "var_h": var_h, "var_s": var_s, "var_v": var_v,
        "var_l": var_l, "var_a": var_a, "var_bhuman": var_bhuman,
        "energy": energy, "ASM": ASM, "contrast": contrast, "homogeneity": homogeneity,
        "kurtosis_r_single": kurtosis_r_single, "kurtosis_g_single": kurtosis_g_single, "kurtosis_b_single": kurtosis_b_single
    }

    return features_dict
```

Figure 2: Feature Extraction

Libraries

```
import cv2
import numpy as np
from skimage import color, feature
from scipy.stats import kurtosis
from PIL import Image, ImageDraw
import requests
import re
import os
import json
from datetime import datetime
import threading
from tensorflow.keras.models import load_model, Sequential
from tensorflow.keras.layers import Dense, Dropout
```

Figure 3: Libraries

Model Defining

In deep learning, model defining involves specifying the architecture and parameters of the neural network. This includes choosing the number of layers, types of activation functions, and connections between neurons. It's a crucial step as it determines the model's capacity to learn and represent complex patterns in data. As we are using Keras for the purpose of designing and implementing the network, the model is defined by adding various layers used and we compile the model before training as shown in the figure 4.

```
# Create a DNN model
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_val, y_val))
```

Figure 4: Defining model

Model Training

Model training is the process of feeding data into the defined neural network to adjust its parameters through backpropagation and optimization algorithms like gradient descent. During training, the model learns to map inputs to outputs by minimizing the difference between predicted and actual values, a process often iterated over multiple epochs to improve performance. The way it is done by using the inbuilt fit function defined in the Keras library by passing the training data to train the model.

Model Testing

Model testing involves evaluating the trained model's performance on unseen data to assess its generalization ability. This step helps determine if the model can effectively make accurate predictions on real-world data beyond the training set. The metrics used for evaluation are accuracy score and loss in this model.

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test loss: {loss}')
print(f'Test accuracy: {accuracy}')
loss, accuracy = model.evaluate(X_train, y_train)
print(f'train loss: {loss}')
print(f'train accuracy: {accuracy}')
loss, accuracy = model.evaluate(X_val, y_val)
print(f'val loss: {loss}')
print(f'val accuracy: {accuracy}')
```

Figure 5: Testing model performance

Model Saving

Model saving entails persisting the trained neural network's parameters and architecture to disk for later use. This is crucial for deploying the model in production environments or sharing it with others for further analysis. This is done by using the save function in the keras library that takes in the filename and saves the model with that name.

```
model.save('MyModel.h5')
```

Figure 6: Saving Model

Model Loading

```
model = load_model('./MyModel.h5')
```

Figure 7: Model Loading

In the context of machine learning applications, the process of model loading is essential as it initializes the neural network architecture and its learned parameters from a saved file. Here, the code utilizes the Keras library's load_model function to load a pre-trained neural network model stored in the file "MyModel.h5". This model is likely trained on a dataset relevant to the task at hand, such as image classification or segmentation.

The significance of this step lies in the fact that a pre-trained model encapsulates knowledge learned from vast amounts of data during the training phase. By loading such a model, we can leverage its ability to make predictions or extract features from new, unseen data. This not only saves time and computational resources but also allows us to benefit from state-of-the-art performance achieved through extensive training.

Line and Feature Processing

The `extract_features_and_color_NLines` function extends the feature extraction process to handle specific regions of interest defined by lines within the input image. This function iterates through the provided lines, drawing them onto a copy of the original image. Subsequently, it divides the image into smaller tiles to efficiently process regions surrounding these lines.

For each tile, the function calculates features using the `extract_features` function and intersects it with the provided lines to determine the regions of interest. By leveraging geometric concepts such as polygons and lines, the function accurately identifies areas for feature extraction and color prediction. The predictions generated by the pre-trained neural network model indicate the color of these regions, enabling visual differentiation between different classes or categories. This functionality is crucial for tasks such as object detection, where identifying and delineating specific objects within an image is essential.

```
def extract_features_and_color_NLines(large_image, model, lines):
    # Get the size of the large image
    width, height = large_image.size
    # Define the size of the smaller images
    tile_size = 16
    arrr = []
    # Create a copy of the original image to draw on
    img_draw = large_image.copy()
    draw = ImageDraw.Draw(img_draw)

    # Draw blue lines on the image for each line in the list
    for line_points in lines:
        draw.line(line_points, fill=(207,86,255), width=4)

    for y in range(0, height, tile_size):
        for x in range(0, width, tile_size):
            tile = large_image.crop((x, y, x + tile_size, y + tile_size))
            tile_np = np.array(tile)
            thickness = 2
            tile_polygon = Polygon([(x, y), (x + tile_size, y), (x + tile_size, y + tile_size), (x, y + tile_size)])
            for line_points in lines:
                line = LineString(line_points)
                if line.intersects(tile_polygon):
                    print(x,y)
                    features = extract_features(tile_np)
                    print(feature)
                    prediction = model.predict(np.array([list(features.values())]))
                    print(prediction)
                    arrr.append(prediction)
                    color = (255, 0, 0) if prediction > 0.5 else [0, 255, 0]
                    print(color)
                    for i in range(thickness):
                        draw.rectangle([x - i, y - i, x + tile_size + i, y + tile_size + i], outline=color)
                    break
            else:
                arrr.append(0) # If no intersection, append 0 to arrr

    img_draw.save('finalout.png')
    return arrr
```

Figure 8: Line and Feature Processing

Usage

The usage section provides guidance on employing the functionalities offered by the code. Users are advised to follow a structured approach:

Image Loading: Load the input image using a suitable library such as PIL (Python Imaging Library).

Line Definition: Define the lines of interest within the image, specifying their coordinates based on the application requirements.

Function Invocation: Call the `extract_features_and_color_NLines` function with the loaded image, the pre-trained model, and the defined lines as input parameters.

Result Retrieval: Retrieve the predictions generated by the function, which indicate the predicted color of regions surrounding the provided lines.

Google maps Image downloader

Libraries

```
from PIL import Image, ImageDraw
import numpy as np
from shapely.geometry import LineString, Polygon
```

Figure 9: Image downloader libraries

The libraries PIL (Python Imaging Library), numpy, and Shapely are essential for downloading and manipulating Google Maps images in Python. PIL facilitates the opening, editing, and saving of various image formats. Numpy empowers efficient numerical operations on image data, converting them into arrays for analysis. Shapely adds geometric manipulation capabilities, crucial for defining regions of interest on maps. Together, they form a powerful toolkit for image processing and geometric analysis tasks. With these libraries, users can download, process, and analyze Google Maps images seamlessly, enabling diverse applications from spatial analysis to computer vision tasks.

Download_tile function

```
def download_tile(url, headers, channels):
    response = requests.get(url, headers=headers)
    arr = np.asarray(bytearray(response.content), dtype=np.uint8)
    if channels == 3:
        return cv2.imdecode(arr, 1)
    return cv2.imdecode(arr, -1)

def project_with_scale(lat, lon, scale):
    siny = np.sin(lat * np.pi / 180)
    siny = min(max(siny, -0.9999), 0.9999)
    x = scale * (0.5 + lon / 360)
    y = scale * (0.5 - np.log((1 + siny) / (1 - siny))) / (4 * np.pi)
    return x, y
```

Figure 10: Download tile function

The `download_tile` function retrieves image tiles from a specified URL with given headers and channels. It utilizes the `requests` library to fetch the image data, converting it into a NumPy array of unsigned 8-bit integers (`dtype=np.uint8`). Depending on the specified number of channels, the function decodes the array using OpenCV's `imdecode` function and returns the resulting image array. This function is vital for downloading and decoding image tiles, often employed in mapping or geospatial applications. On the other hand, `project_with_scale` is essential for projecting latitude and longitude coordinates onto a 2D plane, typically used in cartography or geographical visualization. Given latitude, longitude, and a scale factor, it transforms the latitude into a value between $-\pi/2$ and $\pi/2$, then calculates the projected x and y coordinates on the 2D plane using a Mercator projection formula. These projected coordinates are then scaled according to the provided scale factor and returned, facilitating the visualization and analysis of geographical data.

Downloading google map tiles

```
default_prefs = {
    'url': 'https://mt.google.com/vt/lyrs=s&x={x}&y={y}&z={z}',
    'tile_size': 1024,
    'channels': 3,
    'dir': os.path.join(file_dir, 'images'),
    'headers': {
        'cache-control': 'max-age=0',
        'sec-ch-ua': '" Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"',
        'sec-ch-ua-mobile': '?0',
        'sec-ch-ua-platform': "Windows",
        'sec-fetch-dest': 'document',
        'sec-fetch-mode': 'navigate',
        'sec-fetch-site': 'none',
        'sec-fetch-user': '?1',
        'upgrade-insecure-requests': '1',
        'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.82 Safari/537.36',
    },
    'tl': '',
    'br': '',
    'zoom': ''
}
```

Figure 11: Download Preferences

The `default_prefs` dictionary sets default preferences for downloading Google Maps tiles. It includes parameters such as the URL template for fetching tiles, tile size, number of channels, directory for saving images, and default headers for HTTP requests. The URL template contains placeholders for tile coordinates (x, y, z) to dynamically fetch tiles based on zoom level and location. The headers ensure proper communication with the server, mimicking a web browser's behavior. Additionally, the dictionary contains placeholders for top-left (tl) and bottom-right (br) coordinates, as well as zoom level (zoom), which can be filled in dynamically depending on the specific use case. This setup provides a convenient starting point for configuring and customizing tile downloading processes in Google Maps applications.

Building the image

```
def build_row(tile_y):
    for tile_x in range(tl_tile_x, br_tile_x + 1):
        tile = download_tile(url.format(x=tile_x, y=tile_y, z=zoom), headers, channels)
        if tile is not None:
            # Find the pixel coordinates of the new tile relative to the image
            tl_rel_x = tile_x * tile_size - tl_pixel_x
            tl_rel_y = tile_y * tile_size - tl_pixel_y
            br_rel_x = tl_rel_x + tile_size
            br_rel_y = tl_rel_y + tile_size
            # Define where the tile will be placed on the image
            img_x_l = max(0, tl_rel_x)
            img_x_r = min(img_w + 1, br_rel_x)
            img_y_l = max(0, tl_rel_y)
            img_y_r = min(img_h + 1, br_rel_y)
            # Define how border tiles will be cropped
            cr_x_l = max(0, -tl_rel_x)
            cr_x_r = tile_size + min(0, img_w - br_rel_x)
            cr_y_l = max(0, -tl_rel_y)
            cr_y_r = tile_size + min(0, img_h - br_rel_y)
            img[img_y_l:img_y_r, img_x_l:img_x_r] = tile[cr_y_l:cr_y_r, cr_x_l:cr_x_r]
```

Figure 12: Tile merging function

The `download_image` function is designed to download and stitch together Google Maps tiles to create a single image. It takes latitude and longitude coordinates representing the top-left (`lat1`, `lon1`) and bottom-right (`lat2`, `lon2`) corners of the desired map area, along with the zoom level (`zoom`), URL for tile fetching (`url`), headers for HTTP requests (`headers`), and optional parameters such as tile size (`tile_size`) and number of channels (`channels`). It first calculates the scale factor based on the zoom level and projects the coordinates onto a 2D plane using the `project_with_scale` function. Then, it determines the pixel coordinates and tile coordinates for the top-left and bottom-right corners of the image. Subsequently, it calculates the dimensions of the image and initializes an empty NumPy array (`img`) to store the image data. The function employs multithreading to improve performance. It iterates through each row of tiles within the specified tile range, creating a thread for each row using the `build_row` function. Inside `build_row`, it iterates through each tile along the x-axis, downloading each tile using the `download_tile` function and stitching it into the image array based on its position relative to the image boundaries. Once all threads have completed their tasks, the function returns the stitched image array. This function is crucial for downloading and assembling Google Maps images efficiently and is suitable for various mapping and geospatial applications.

Database

Adding values and sorting

```
def add_value(self, place, top_left, bottom_right, point_a, point_b):
    unique_id = self.id_generator.generate_id()
    self.data[unique_id] = {
        "place": place,
        "top_left": top_left,
        "bottom_right": bottom_right,
        "point_a": point_a,
        "point_b": point_b
    }
    return unique_id

def sort_data(self):
    sorted_data = sorted(self.data.values(), key=lambda x: (x["place"], x["point_a"], x["point_b"]))
    return sorted_data
```

Figure 13: Database function

The `add_value` method of the class takes various parameters such as `place`, `top_left`, `bottom_right`, `point_a`, and `point_b` and adds a new entry to the class's data dictionary. It generates a unique ID for the entry using the `id_generator` object and assigns it as the key in the dictionary. The method then stores the provided parameters as values corresponding to this unique ID. Finally, it returns the generated unique ID. The `sort_data` method of the class sorts the data stored in the class's data dictionary based on three criteria: `place`, `point_a`, and `point_b`. It retrieves the values from the dictionary using `self.data.values()`, then sorts them using Python's built-in `sorted` function. The key parameter is utilized to specify a custom sorting criterion, defined as a lambda function that extracts the relevant fields from each entry. The method returns the sorted list of data entries.

Searching values and exporting to csv

```
def get_value(self, unique_id):
    return self.data.get(unique_id, None)

def export_to_csv(self, filename):
    with open(filename, mode='w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(["ID", "Place", "Top Left", "Bottom Right", "Point A", "Point B"])
        for unique_id, entry in self.data.items():
            writer.writerow([unique_id, entry["place"], entry["top_left"], entry["bottom_right"], entry["point_a"], entry["point_b"]])

store = DataStore()
store.add_value("Nitte", (13.1836393, 74.9377762), (13.1823240, 74.9391743), (13.1834781, 74.9382653), (13.1824829, 74.9382841))
store.add_value("Nitte", (13.1804760, 74.9322961), (13.1792047, 74.9327511), (13.1801400, 74.9328341), (13.1792047, 74.9327511))
store.add_value("Nitte", (13.1762453, 74.9301518), (13.1759166, 74.9309925), (13.1760525, 74.9302892), (13.1762360, 74.9308400))
store.add_value("Nitte", (13.2090413, 74.9098029), (13.2083968, 74.9107404), (13.2085870, 74.9105566), (13.2087791, 74.9100519))
```

Figure 14: Export csv function

The `get_value` method retrieves the data associated with a given unique ID from the class's data dictionary. It takes the unique ID as input and uses the `get` method of the dictionary to retrieve the corresponding entry. If the unique ID does not exist in the dictionary, it returns `None`. The `export_to_csv` method exports the data stored in the class's data dictionary to a CSV file. It takes a filename as input and opens a file in write mode. It initializes a CSV writer object with the opened file and writes the header row containing the column names. Then, it iterates through each entry in the data dictionary, writing the unique ID and the corresponding values for "Place", "Top Left", "Bottom Right", "Point A", and "Point B" into the CSV file. Finally, it closes the file after writing all the data.

Getting nearest powerline

Haversine distance

```
def haversine_distance(self, lat1, lon1, lat2, lon2):
    # Calculate the great-circle distance between two points on the Earth's surface
    R = 6371 # Radius of the Earth in kilometers
    dlat = math.radians(lat2 - lat1)
    dlon = math.radians(lon2 - lon1)
    a = math.sin(dlat / 2) * math.sin(dlat / 2) + math.cos(math.radians(lat1)) * math.cos(math.radians(lat2)) * math.sin(dlon / 2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    distance = R * c
    return distance
```

Figure 15: Haversine distance function

The `haversine_distance` method calculates the great-circle distance between two points on the Earth's surface using the Haversine formula. It takes latitude and longitude coordinates of two points (`lat1`, `lon1`, `lat2`, `lon2`) as input parameters. The method begins by defining the radius of the Earth (`R`) in kilometers. Then, it computes the differences in latitude (`dlat`) and longitude (`dlon`) between the two points, converting them into radians. Next, it applies the Haversine formula, which involves trigonometric calculations using sine and cosine functions, to determine the intermediate values `a` and `c`. Finally, it calculates the distance using the formula $\text{distance} = R * c$, where `c` represents the central angle between the two points on the Earth's surface. The computed distance is returned as the output of the method, representing the distance between the two points in kilometers. This method is useful for calculating distances between geographical locations, which is essential for various applications such as navigation, logistics, and geospatial analysis.

Calculating distance to powerline segments

```
def distance_to_line_segment(self, point_a, point_b, target_point):
    # Calculate distance between a point and a line segment
    lat1, lon1 = point_a
    lat2, lon2 = point_b
    lat3, lon3 = target_point

    d_lat_ab = lat2 - lat1
    d_lon_ab = lon2 - lon1
    d_lat_ac = lat3 - lat1
    d_lon_ac = lon3 - lon1
    d_lat_bc = lat3 - lat2
    d_lon_bc = lon3 - lon2

    dot_ab_ac = d_lat_ab * d_lat_ac + d_lon_ab * d_lon_ac
    dot_ba_bc = -d_lat_ab * d_lat_bc - d_lon_ab * d_lon_bc

    if dot_ab_ac <= 0:
        return self.haversine_distance(lat1, lon1, lat3, lon3)
    elif dot_ba_bc <= 0:
        return self.haversine_distance(lat2, lon2, lat3, lon3)
    else:
        distance = abs(d_lat_ab * d_lon_ac - d_lon_ab * d_lat_ac) / math.sqrt(d_lat_ab ** 2 + d_lon_ab ** 2)
        return distance
```

Figure 16: Distance calculation function

The `distance_to_line_segment` method calculates the distance between a point and a line segment defined by two other points using vector operations. It takes three sets of latitude and longitude coordinates (`point_a`, `point_b`, `target_point`) as input parameters. The method starts by extracting the latitude and longitude coordinates for each point. Then, it computes the differences in latitude and longitude between points to define vectors representing segments AB, AC, and BC. Next, it calculates the dot products between vectors AB and AC (`dot_ab_ac`) and between vectors BA and BC (`dot_ba_bc`). These dot products are used to determine whether the target point falls outside the line segment AB. If `dot_ab_ac` is less than or equal to 0, the target point is closer to point A, so the distance between point A and the target point is returned. If `dot_ba_bc` is less than or equal to 0, the target point is closer to point B, so the distance between point B and the target point is returned. If the target point lies within the line segment AB, the method proceeds to calculate the perpendicular distance between the target point and the line segment AB using the formula $\text{distance} = \text{abs}(d_{\text{lat_ab}} * d_{\text{lon_ac}} - d_{\text{lon_ab}} * d_{\text{lat_ac}}) / \text{math.sqrt}(d_{\text{lat_ab}}^2 + d_{\text{lon_ab}}^2)$. This distance is then returned as the result.

Getting nearest powerline

```
def get_nearest_line_segment_id(self, place, target_point):
    min_distance = float('inf')
    nearest_segment_id = None

    for unique_id, entry in self.data.items():
        if entry["place"] == place:
            # Calculate distance between target point and line segment AB
            distance = self.distance_to_line_segment(entry["point_a"], entry["point_b"], target_point)
            if distance < min_distance:
                min_distance = distance
                nearest_segment_id = unique_id
    return nearest_segment_id, min_distance * 1000 # Convert distance from kilometers to meters
```

Figure 17: get_nearest line segment function

The `get_nearest_line_segment_id` method identifies the nearest line segment to a given target point within a specified place. It iterates through data entries associated with the specified place, computing the distance between the target point and each line segment. It updates the minimum distance and nearest segment ID if a closer segment is found. Finally, it returns the ID of the nearest segment and the distance in meters from the target point to that segment. This method is essential for tasks like locating nearby features or determining optimal routes in geospatial applications.

Pixel mapping and encroachment detection

Pixel mapping

```
def lat_lon_to_pixel(lat, lon, image_width, image_height, lat1, lon1, lat2, lon2):
    x = (lon - lon1) / (lon2 - lon1) * image_width
    y = (lat - lat1) / (lat2 - lat1) * image_height
    return int(x), int(y)
```

Figure 18: Pixel mapping function

The function `lat_lon_to_pixel` converts latitude and longitude coordinates to pixel coordinates within an image. It takes latitude (lat) and longitude (lon) coordinates, along with the dimensions of the image (image_width and image_height). Additionally, it requires the latitude and longitude coordinates of two reference points (lat1, lon1, lat2, lon2) that define the corners of the image. Using linear interpolation, it calculates the relative position of the input coordinates within the image bounds, considering the relationship between the input coordinates and the reference points. The calculated pixel coordinates (x, y) are then returned as integers, representing the position of the input coordinates within the image. This function is crucial for mapping geographic coordinates onto an image for visualization or analysis.

Encroachment detection

```
def draw_square(image, center_pixel, size=100, color=(71, 233, 255), thickness=2):
    # Calculate the coordinates of the top-left corner of the square
    top_left_x = center_pixel[0] - size // 2
    top_left_y = center_pixel[1] - size // 2

    # Calculate the coordinates of the bottom-right corner of the square
    bottom_right_x = center_pixel[0] + size // 2
    bottom_right_y = center_pixel[1] + size // 2

    # Draw the square boundary box
    cv2.rectangle(image, (top_left_x, top_left_y), (bottom_right_x, bottom_right_y), color, thickness)

    # Count the number of red pixels inside the square
    red_count = 0
    total_pixels = 0
    for x in range(top_left_x, bottom_right_x):
        for y in range(top_left_y, bottom_right_y):
            if image[y, x][2] == 255 and image[y, x][0] == 0 and image[y, x][1] == 0: # Checking if R=255, G=0, B=0
                red_count += 1
                total_pixels += 1

    # Calculate the percentage of red pixels
    red_percent = (red_count / total_pixels) * 100
```

Figure 19: Draw square function

The draw_square function is used to draw a square on an image given its center pixel, size, color, and thickness. It first calculates the coordinates of the top-left and bottom-right corners of the square based on the provided center pixel and size. Then, it draws a rectangle representing the square boundary box on the image using OpenCV's cv2.rectangle function and its center is the target point. After drawing the square, the function counts the number of red pixels inside the square by iterating through each pixel within the square's boundaries. It checks if a pixel is red (R=255, G=0, B=0) and increments the red pixel count accordingly. Simultaneously, it calculates the total number of pixels inside the square. Using the counts of red pixels and total pixels, the function calculates the percentage of red pixels within the square. If the percentage of red pixels is greater than 0%, it prints a message indicating a risk area is detected. Otherwise, it prints a message indicating no risk is detected. Finally, the function saves the modified image with the drawn square as "final_image.png" using cv2.imwrite. This function is suitable for applications such as image analysis and risk detection, where identifying specific regions within an image based on color criteria is required.

CHAPTER 8

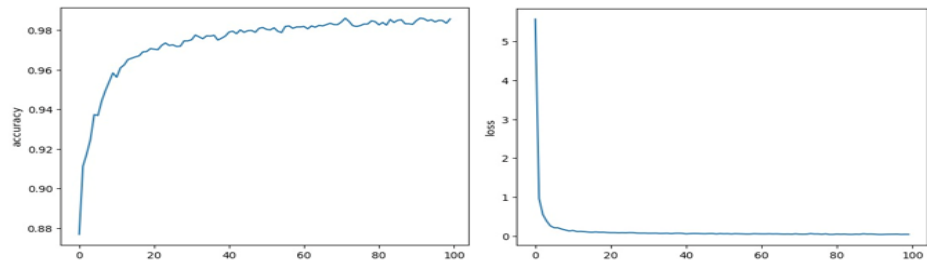
RESULTS

The data split was done in the ratio 60 – 20 – 20 to training validation and testing purpose. The training dataset contains 7410 samples of data stored in 232 batches. The testing and validation dataset contains 2471 samples of data stored in 78 batches. The model is trained for 100 epochs. The train time for each epoch was about 1s with 3ms for each step.

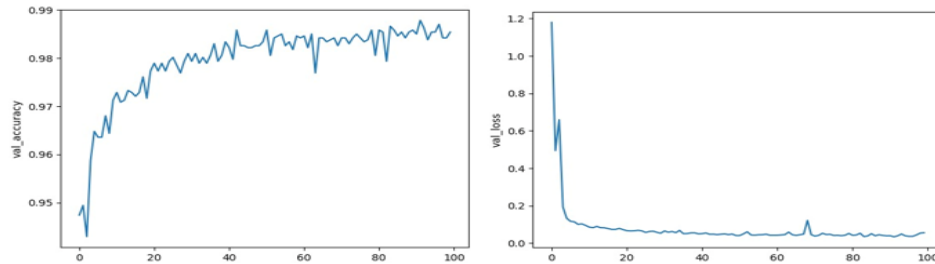
During the initial stages of training, both the training and validation losses are observed to be high, while the accuracy remains relatively low. However, as training progresses there is a noticeable decrease in the loss metrics alongside an increase in accuracy for both the training and validation sets. This trend signifies that the model is effectively learning from the provided data, enhancing its performance over successive epochs. As the training nears completion, the loss values stabilize, and the accuracy metrics reach a plateau. This stabilization indicates that the model may have converged to a certain extent, suggesting that further training iterations might not significantly improve its performance. Overall, the observed patterns in loss reduction and accuracy improvement throughout the training process reflect the model's capability to learn from the data and optimize its predictive capabilities, ultimately contributing to its overall effectiveness in making accurate predictions.

The plotted figures depict the performance metrics of a neural network model during training and validation across multiple epochs. The upper graph illustrates the progression of training and validation accuracy over epochs. As training advances, both accuracies demonstrate an upward trend, indicating the model's improvement in learning and ability to generalize to unseen data. Conversely, the lower graph presents the training and validation loss over epochs. Here, the declining trend in loss values suggests the model's enhanced predictive capabilities, as lower losses correspond to improved performance. The convergence of training and validation metrics signifies the model's effective learning of data patterns and its ability to generalize predictions accurately.

The model achieved accuracy of 98.75% on the test dataset with a loss of 0.0510. The accuracy and precision of the model on the training and validation data is 98.69% with 0.03484 loss and 98.46% and 0.0475.



Training accuracy and loss graph



Validation accuracy and loss graph

Figure 20- Training and Validation Accuracy and Loss Graph

```
78/78 [=====] - 0s 2ms/step - loss: 0.0510 - accuracy: 0.9875
Test loss: 0.05104377120733261
Test accuracy: 0.9874544739723206
232/232 [=====] - 0s 2ms/step - loss: 0.0348 - accuracy: 0.9869
train loss: 0.034844476729631424
train accuracy: 0.9869095683097839
78/78 [=====] - 0s 2ms/step - loss: 0.0475 - accuracy: 0.9846
val date: 0.047500018030405045
val accuracy: 0.9846153855323792
```

Figure 21- Accuracy and loss results

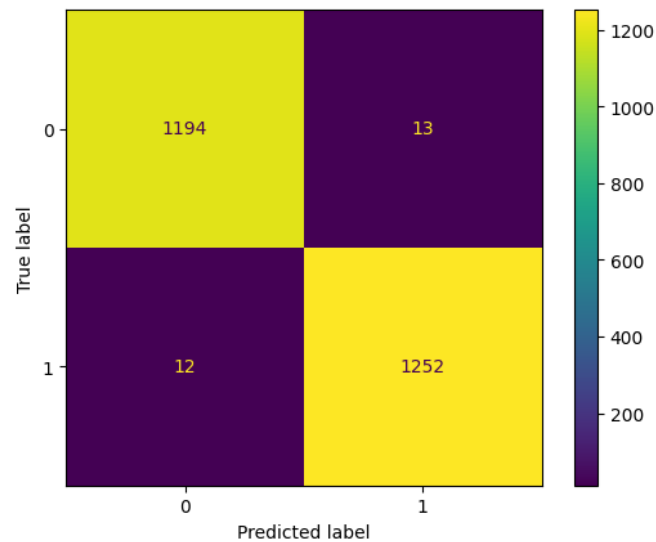


Figure 22- Confusion matrix on test data

We ran our model on some of the images in the test dataset. The results show that the model is able to correctly identify the encroachment in the given area

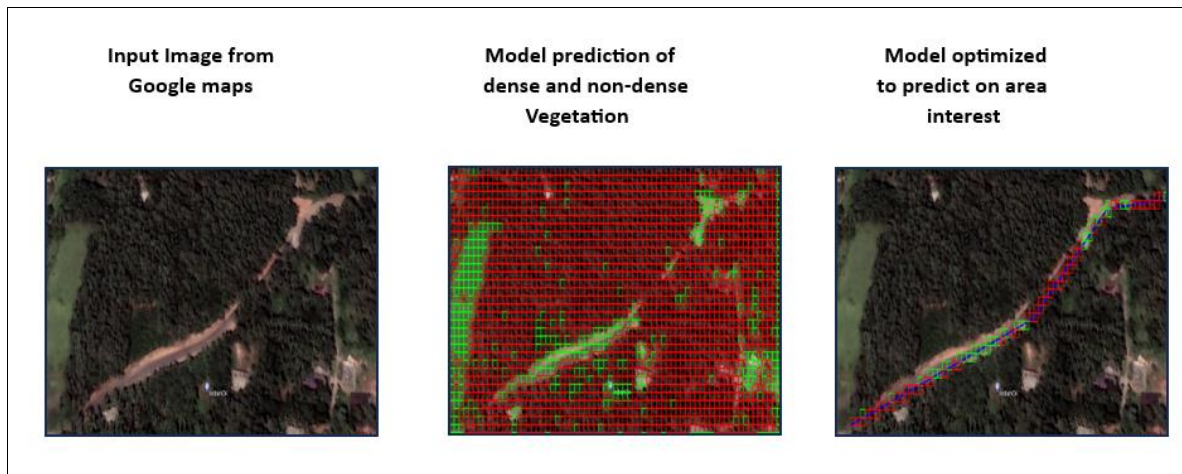


Figure 23: Predicted images

The validation of the model was done near Nitte Gents hostel the model predictions accurately match with the actual output



Figure 24: Figure showing encroachment

Area near Nitte Gents hostel
Ground where there is no encroachment
in area of interest



Ground image of the area
where there is no encroachment



Figure 25: Figure without encroachment

CHAPTER 9

CONCLUSION

Our research endeavors have been multifaceted, involving a meticulous examination of diverse strategies, the acquisition of a dataset of paramount importance, and the exploration of future possibilities. Through a rigorous process, we have delved into various approaches, aiming to deepen our understanding of the subject matter at hand. The dataset we meticulously gathered stands as a cornerstone of our research, providing relevant and crucial insights essential for informed decision-making. Furthermore, our exploration of future options has equipped us with invaluable insights, setting the stage for strategic actions in the ever-changing landscape. This comprehensive analysis not only enriches our understanding but also serves as a guiding force for future decisions and strategies. By amalgamating meticulous research methodologies with forward-thinking exploration, we are well-positioned to navigate the dynamic terrain effectively.

CHAPTER 10

REFERENCES

- [1] . **Yangyu Chen , Jiayuan Lin, Xiaohan Liao(2022)** “Early detection of tree encroachment in high voltage powerline corridor using growth model and UAV-borne LiDAR”.
- [2] . **Fathi mahdi elsiddig haroun , Siti noratiqah mohamad deros and Norashidah md din (2021)** “Detection and Monitoring of Power Line Corridor From Satellite Imagery Using RetinaNet and K-Mean Clustering”
- [3] . **Levente J Klein ,Wang Zhou ,Hendrik F ,Hamann,Conrad M Albrecht,Carlo ,Siebenschuh,Siyuan Lu,Sharathchandra Pankanti (2019)** “N-dimensional geospatial data and analytics for critical infrastructure risk assessment”
- [4]. **Abdul Qayyum, Aamir Saeed Malik, Mohamad Naufal ,Mohamad Saad, Mahboob Iqbal, Rana Fayyaz Ahmad, Tuan Ab Rashid Bin Tuan Abdullah,A hmad Quisti Ramli (2014)** “Monitoring of Vegetation Near Power Lines Based on Dynamic Programming using Satellite Stereo Images”
- [5] . **Rabab Abdelfattah, Xiaofeng Wang, Song Wang (2020)** “An Aerial-Image Dataset for Detection and Segmentation of Transmission Towers and Power Lines”
- [6] . **Zhengrong Li Yuee Liu Ross Hayward Jinglan Zhang Jinhai Cai (2008)** “Knowledge-based Power Line Detection for UAV Surveillance and Inspection Systems”
- [7] . **Shuaiang Rong,Liang Du,Zuyi li,Shiwen yu (2020)** “Intelligent Detection of Vegetation Encroachment of Power Lines with Advanced Stereovision”
- [8] . **Nina m. singer, Vijayan k. asari (2021)** “A large Scale Benchmark Dataset for Instance Segmentation in Aerial Lidar”
- [9] . **Seulbi Lee and Youngjib Ham (2023)** “Measuring the Distance between Trees and Power Lines under Wind Loads to Assess the Heightened Potential Risk of Wildfire”
- [10] . **Yunping Chen,n , Yang Li , Huixiong Zhang , Ling Tong, Yongxing Cao, Zhihang Xue (2016)** “Automatic power line extraction from high resolution remote sensing imagery based on an improved Radon transform”