AUTOMATED POLE DETECTION SYSTEM

Bonafide record of work done by

AKSHAY PERISON DAVIS J	**(21Z205)**
ASWINKUMAR V	**(21Z212)**
MITHRAN M	**(21Z230)**
BHARATH S	**(21Z246)**
SNEHAN E M	**(21Z257)**
ROOHITH M R	**(22Z432)**

19Z610 – MACHINE LEARNING LABORATORY

Dissertation submitted in the fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

BRANCH: COMPUTER SCIENCE AND ENGINEERING

APRIL 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

PROBLEM STATEMENT:

Given a Google Maps image with a pole, the system has to do one thing :

1. Detects if a pole is present.

This research proposes an automated system designed to detect the presence of poles within Google Maps images and subsequently estimate the height of the identified poles from the ground. The system aims to assist in various applications such as urban planning, infrastructure maintenance, and environmental monitoring. This system optimizes resources by reducing manual efforts and improving data accuracy, enabling data-driven decision-making for urban planners and developers.

DATASET DESCRIPTION:

The Electric Pole Image Dataset is a collection of images depicting various types of electric poles. The dataset consists of two main categories: full pole images and half pole images. These images are primarily intended for use in tasks such as object detection, image classification, and semantic segmentation related to electric pole infrastructure.

Dataset Statistics:

- Total Number of Images: 1400
- Full Pole Images: 800
- Half Pole Images: 600
- Resolution: 640 * 640 pixels.
- Format: Images are provided in commonly used formats such as JPEG, PNG, etc.

Annotations:

The dataset may or may not contain annotations depending on the specific use case or project requirements. Annotations could include bounding boxes outlining the poles, labels indicating whether the pole is full or half, or other relevant information.

METHODOLOGIES / MODELS USED:

The following are the models that were used to train our system for image detection.

YOLO v7:

YOLO is a popular deep learning model for object detection. YOLO v7 is the seventh version of this model, known for its speed and accuracy in detecting objects in images and videos. YOLO v7 uses a single neural network to predict bounding boxes and class probabilities directly from full images in one evaluation. This model is well-suited for real-time applications, including pole detection in urban environments.

TensorFlow Models:

TensorFlow is an open-source machine learning framework developed by Google. It provides a collection of pre-trained models and tools for various machine learning tasks, including object detection. TensorFlow models, such as Faster R-CNN, SSD, and RetinaNet, are commonly used for object detection tasks. These models can be fine-tuned and adapted for specific applications, such as pole detection, by training them on custom datasets.

YOLO v3:

YOLO v3 is another version of the YOLO model, known for its improved performance and accuracy compared to earlier versions. YOLO v3 divides the image into a grid and predicts bounding boxes and probabilities for each grid cell. It is suitable for real-time object detection, including pole detection in urban environments.

R-CNN (REGION BASED CONVOLUTIONAL NEURAL NETWORK):

The Region-based Convolutional Neural Network (RCNN) is a seminal model in the field of computer vision, particularly renowned for its effectiveness in object detection tasks. Introduced by Ross Girshick et al. in 2014, RCNN addresses the challenges of object detection by combining the power of Convolutional Neural Networks (CNNs) with region-based methods.

PROPOSED MODEL INTEGRATION:

To enhance pole detection accuracy, a fusion of YOLO v7 and TensorFlow models can be employed. YOLO v7 can be used for initial object detection, identifying potential poles in urban environments. Subsequently, TensorFlow models can be used for detailed analysis and verification, ensuring accurate pole detection. This integration of models can improve the system's ability to detect poles in various conditions and environments, making it more robust and reliable for real-world application.

Flow of Work

Data Acquisition: Preprocessing: Object Detection: Bounding Box Prediction: Post-processing: Visualization: User Interface: Deployment:

TOOLS USED:

The following are the software requirements for the proposed system.

- **Python Libraries:**
  - matplotlib>=3.2.2: For plotting graphs and visualizations.
  - numpy>=1.18.5,<1.24.0: For numerical computations.
  - opencv-python>=4.1.1: For image and video processing.
  - Pillow>=7.1.2: For image processing.
  - PyYAML>=5.3.1: For YAML file parsing.
  - requests>=2.23.0: For making HTTP requests.
  - scipy>=1.4.1: For scientific computing.
  - tqdm>=4.41.0: For displaying progress bars.
  - protobuf<4.21.3: For protocol buffer serialization.
- **Logging:**
  -tensorboard>=2.4.1: For logging and visualization of training metrics.
- **Plotting:**
  - pandas>=1.1.4: For data manipulation and analysis.
  - seaborn>=0.11.0: For statistical data visualization.
- **Extras:**
  - ipython: For an interactive Python shell.
  - psutil: For system utilization monitoring.
  - thop: For computing the number of FLOPs (Floating Point Operations) in a model.

CHALLENGES FACED:

Detecting poles, such as utility poles or streetlight poles, can pose several challenges depending on the context and environment. Here are some common challenges in pole detection systems:

Variability in Pole Types and Sizes: Utility poles, streetlight poles, and other types of poles come in various shapes and sizes. Some may be wooden, metal, or concrete, and their dimensions can vary widely. This variability makes it challenging to develop a one-size-fits-all detection algorithm.

Background Clutter: Poles are often located in urban environments with complex backgrounds, such as buildings, trees, and other structures. Distinguishing poles from background clutter can be difficult, especially in crowded scenes.

Illumination Variations: Lighting conditions can vary significantly throughout the day and under different weather conditions. Shadows, reflections, and changes in lighting angle can affect the appearance of poles, making them harder to detect consistently.

Occlusion: Poles may be partially or fully occluded by other objects, such as vehicles, vegetation, or infrastructure. Detecting poles in such scenarios requires algorithms capable of inferring the presence of poles even when they are not fully visible.

Training Data Availability: Developing accurate pole detection algorithms often requires large amounts of labeled training data. Obtaining high-quality, diverse datasets that adequately represent the variability of poles and environmental conditions can be challenging.

CONTRIBUTION OF TEAM MEMBERS:

NAME	ROLL.NO	CONTRIBUTION
AKSHAY PERISON DAVIS J SNEHAN E M	21Z205 21Z257	● Trained YOLO v7 ● Backend and Frontend integration ● Dataset Separation ● Documentation
MITHRAN M BHARATH S	21Z230 21Z246	● Trained TensorFlow and YOLO v3 ● Dataset Preparation ● Performance Analysis ● Documentation
ASWINKUMAR V ROOHITH M R	21Z212 22Z432	● Trained R-CNN ● Dataset Collection ● Frontend Design ● Documentation of results

ANNEXURE I:

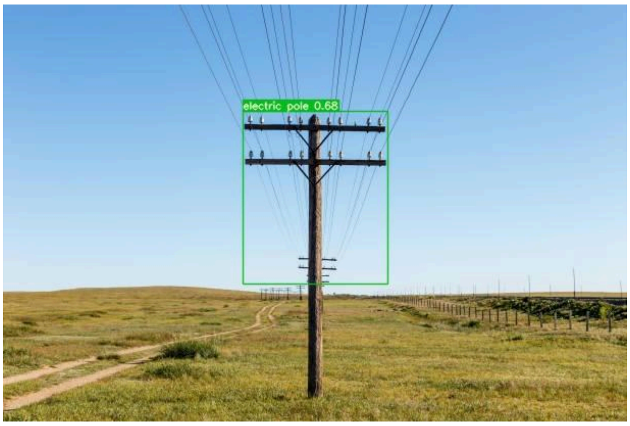
Main.py:

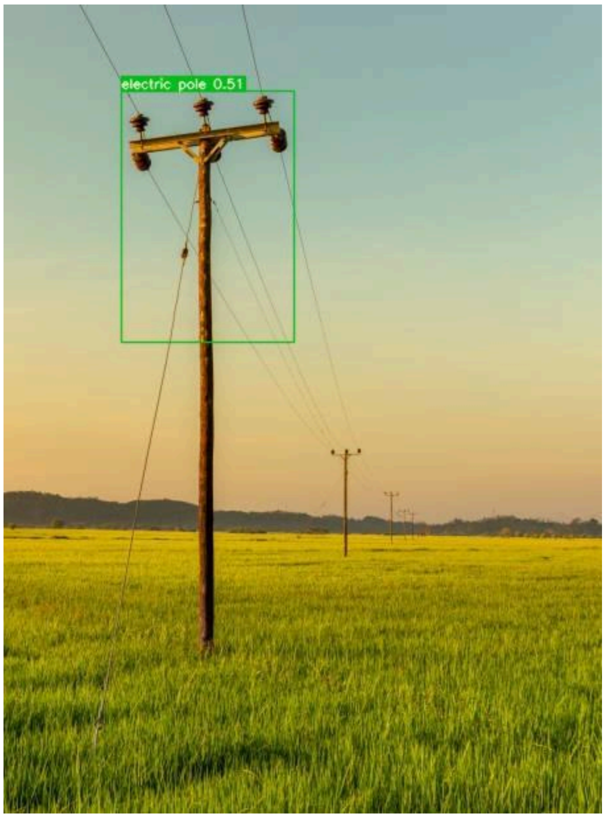
```
from flask import Flask,
redirect, render_template,
url_for
from flask_wtf import
FlaskForm
from wtforms import
FileField, SubmitField
from werkzeug.utils import
secure_filename
import os
from wtforms.validators
import InputRequired
import subprocess
import sys
app = Flask(__name__)
app.config['SECRET_KEY'] =
'supersecretkey'
app.config['UPLOAD_FOLD
ER'] = 'static/files'
def run_script2(args):
   script2_path =
"flaskdetect.py"
 subprocess.call([sys.executable,
script2_path] + args.split())
class UploadFileForm(FlaskForm):
   file = FileField("File",
validators=[InputRequired()])
   submit =
SubmitField("Upload File")
@app.route('/',
methods=['GET',"POST"])
@app.route('/home',
methods=['GET',"POST"])
def home():
   form = UploadFileForm()
   if
form.validate_on_submit():
      file = form.file.data #
First grab the file
      filename =
  imgsz =
check_img_size(imgsz,
s=stride)  # check img_size
   if trace:
      model =
TracedModel(model, device,
opt.img_size)
   if half:
      model.half()  # to FP16
   # Second-stage classifier
   classify = False
   if classify:
      modelc =
load_classifier(name='resnet10
1', n=2)  # initialize
modelc.load_state_dict(torch.l
oad('weights/resnet101.pt',
map_location=device)['model']
).to(device).eval()
   # Set Dataloader
   vid_path, vid_writer =
None, None
   if webcam:
      view_img =
check_imshow()
      cudnn.benchmark = True
# set True to speed up constant
image size inference
      dataset =
secure_filename(file.filename)
file.save(os.path.join(os.path.a
bspath(os.path.dirname(__file_
_)),app.config['UPLOAD_FO
LDER'],secure_filename(file.fi
lename))) # Then save the file
      args = "--weights
before_march24.pt --conf 0.3
--img-size 640 --source
"+app.config['UPLOAD_FOL
DER']+'/'+secure_filename(file
.filename)+" --view-img
--no-trace"
      run_script2(args)
                                    return
```

```
redirect('/display/'+filename)
    return
render_template('index.html',
form=form)
@app.route('/display/<filenam
e>')
def display_image(filename):
    #print('display_image
filename: ' + filename)
    return
redirect(url_for('static',
filename='files/' + filename),
code=301)
if __name__ == '__main__':
    app.run(debug=True)
```

ANNEXURE II:

Snapshots of the output





REFERENCES

[1] Malak Abdullah∗, Mirsad Hadzikadic† and Samira Shaikh‡, “*SEDAT: Sentiment and Emotion Detection in Arabic Text using CNN-LSTM Deep Learning*” 2018 17th IEEE International Conference on Machine Learning and Applications

[2] Nguyen, T., et al. (2020). "Automated Detection of Utility Poles in LiDAR Point Cloud Data Using Deep Learning."

[3] Wang, Z., et al. (2017). "Automatic Detection of Utility Poles in Aerial Images Using Convolutional Neural Networks."

[4] Chen, X., et al. (2019). "An Automated Method for Utility Pole Detection in LiDAR Data Based on Object-Based Image Analysis."

[5] Kang, J., et al. (2020). "Deep Learning-based Pole Detection from Aerial Imagery."

[6] Li, W., et al. (2018). "Automatic Detection of Utility Poles from Mobile LiDAR Point Clouds."

[7] Ren, S., et al. (2015). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks."

[8] Liu, W., et al. (2016). "SSD: Single Shot MultiBox Detector."

[9] He, K., et al. (2017). "Mask R-CNN."

[10] Redmon, J., et al. (2016). "You Only Look Once: Unified, Real-Time Object Detection.