Contents

Automatic Number Plate Recognition

Overview

Automatic Number Plate Recognition (ANPR), also known as Automatic License Plate Recognition (ALPR), is a technology that uses optical character recognition on images to read vehicle registration plates. The main purposes and goals of ANPR systems include:

Purpose:

1. **Accuracy and Reliability:**
   - ANPR systems aim to achieve high accuracy in reading license plates under various environmental conditions (day/night, different weather conditions, varying angles).
2. **Speed and Efficiency:**
   - The systems are designed to process large volumes of data quickly, enabling real-time identification and response in applications such as law enforcement and toll collection.
3. **Integration and Compatibility:**
   - ANPR systems should integrate seamlessly with existing infrastructure such as CCTV cameras, databases (for vehicle information and law enforcement records), and other technologies used in traffic management and security.
4. **Privacy and Data Security:**
   - Ensuring that ANPR systems comply with privacy laws and regulations, safeguarding collected data, and implementing measures to protect against unauthorized access and misuse.
5. **Scalability and Adaptability:**
   - ANPR systems should be scalable to handle increasing volumes of traffic and adaptable to different geographical locations and operational requirements.

Applications:

1. **Law Enforcement and Security:**
   - **Traffic Monitoring:** ANPR systems are used to monitor and manage traffic flow on roads and highways, detect traffic violations such as speeding or running red lights, and enforce traffic laws.
   - **Crime Prevention:** ANPR helps law enforcement agencies to locate stolen vehicles, identify vehicles associated with criminal activities (e.g., Amber Alerts, wanted vehicles), and track suspects or persons of interest.
2. **Parking Management:**
   - ANPR systems are used in parking facilities to automate entry and exit processes, manage parking space availability, and enforce parking rules and fees.
3. **Toll Collection:**
   - ANPR technology is integrated into toll collection systems to automate toll payment processes by capturing vehicle license plate information and associating it with an account or payment method.
4. **Border Control and Immigration:**

- ANPR systems aid in monitoring and managing vehicle movements at border crossings and immigration checkpoints, enhancing security and immigration control processes.

5. **Access Control:**
   - ANPR systems are used in private and public sectors for access control to gated communities, secure facilities, corporate premises, and restricted areas.

Object Detection

Object detection is a computer vision task that involves identifying and locating objects of interest within an image or video frame. Unlike image classification, which assigns a label to an entire image, object detection techniques not only classify objects but also precisely outline their locations with bounding boxes. Here's an overview of object detection:

Purpose and Applications:

Purpose of Object Detection:

1. **Accurate Localization:** The primary goal of object detection is to accurately localize objects within images or video frames by predicting bounding boxes that tightly enclose each object.
2. **High Classification Accuracy:** Object detection systems aim to correctly classify each detected object into predefined categories or classes, ensuring accurate identification.
3. **Real-time Performance:** Achieving real-time or near-real-time processing speeds is crucial for applications such as autonomous vehicles, surveillance systems, and robotics, where timely detection and response are critical.
4. **Robustness:** Object detection models should be robust to variations in lighting conditions, object scales, orientations, occlusions, and background clutter, ensuring reliable performance in diverse environments.
5. **Efficiency:** Efficient use of computational resources is important, particularly for deployment on resource-constrained devices or for handling large-scale datasets and real-time streaming data.

Applications of Object Detection:

1. **Autonomous Vehicles:**
   - **Purpose:** Detect and classify pedestrians, vehicles, cyclists, and other objects to enable safe navigation and decision-making.
   - **Benefits:** Enhances road safety, improves traffic management, and supports autonomous driving capabilities.
2. **Surveillance and Security:**
   - **Purpose:** Monitor and analyze video feeds to detect suspicious activities, intrusions, or unauthorized objects.
   - **Benefits:** Enhances security monitoring, aids in crime prevention, and facilitates quick response to incidents.

3. **Retail and Inventory Management:**
    - **Purpose:** Track and manage inventory, monitor shelf stocking, and analyze customer behavior through object detection of products and people.
    - **Benefits:** Optimizes inventory levels, improves retail operations, and enhances customer experience through personalized services.
4. **Medical Imaging:**
    - **Purpose:** Assist in medical diagnosis by detecting and localizing anatomical structures, tumors, or abnormalities in medical images such as X-rays, MRI scans, and CT scans.
    - **Benefits:** Supports early detection of diseases, aids in treatment planning, and improves patient care outcomes.
5. **Industrial Automation:**
    - **Purpose:** Inspect products for defects, monitor assembly lines, and ensure quality control in manufacturing processes.
    - **Benefits:** Reduces production errors, enhances efficiency, and lowers operational costs by automating inspection tasks.
6. **Traffic Management and Smart Cities:**
    - **Purpose:** Monitor traffic flow, detect traffic violations (e.g., running red lights), and manage congestion through vehicle and pedestrian detection.
    - **Benefits:** Improves traffic safety, optimizes urban planning, and supports sustainable urban development initiatives.

Text Recognition:

Text recognition, also known as Optical Character Recognition (OCR), is the process of converting different types of text—such as scanned paper documents, PDF files, or images captured by a digital camera—into editable and searchable data. This technology recognizes text within images, transforming it into a format that can be processed by computers.

Purpose:

1. **Automating Data Entry:**
    - **Definition:** Automatically converting printed or handwritten text into digital formats.
    - **Purpose:** Reducing manual data entry tasks, minimizing errors, and improving efficiency in data processing.
2. **Enhancing Accessibility:**
    - **Definition:** Making written content accessible through text-to-speech or other assistive technologies.
    - **Purpose:** Helping visually impaired individuals and enabling the translation of written content into different languages.
3. **Improving Searchability:**
    - **Definition:** Converting text from physical documents into searchable digital text.
    - **Purpose:** Facilitating the easy retrieval of specific information within documents.
4. **Archiving and Preservation:**

- **Definition:** Digitizing old documents, books, and manuscripts.
- **Purpose:** Preserving important documents for the long term and ensuring easy access.

5. **Enabling Advanced Analytics:**
   - **Definition:** Extracting valuable information from large volumes of text.
   - **Purpose:** Supporting data analysis, business intelligence, and informed decision-making.

Applications:

1. **Business and Office Automation:**
   - **Definition:** Automating the scanning and processing of business documents.
   - **Applications:** Handling invoices, receipts, contracts, and ensuring compliance and organization.
2. **Healthcare:**
   - **Definition:** Digitizing patient records and medical documents.
   - **Applications:** Managing patient information, improving data entry in electronic health records (EHR) systems.
3. **Education:**
   - **Definition:** Converting educational materials into digital formats.
   - **Applications:** Distributing textbooks, lecture notes, and exam papers; facilitating e-learning and digital libraries.
4. **Legal Sector:**
   - **Definition:** Digitizing legal documents and case files.
   - **Applications:** Organizing legal texts for research, case analysis, and better retrieval.
5. **Banking and Finance:**
   - **Definition:** Automating the processing of financial documents.
   - **Applications:** Handling checks, forms, financial statements; improving data entry in banking systems.
6. **Retail:**
   - **Definition:** Scanning and processing product-related information.
   - **Applications:** Managing inventory, automating point-of-sale systems, processing barcodes and receipts.
7. **Government and Public Services:**
   - **Definition:** Digitizing public records and administrative documents.
   - **Applications:** Improving the efficiency of administrative processes, enhancing public access to documents.
8. **Publishing:**
   - **Definition:** Converting manuscripts and publications into digital formats.
   - **Applications:** Facilitating online publishing, editing, and proofreading workflows.

Implementation Of Automatic Number Plate Recognition (ANPR)

Models and Libraries Involved

- **OpenCV:**

  - **Definition:** OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It contains over 2500 optimized algorithms for a wide range of computer vision tasks.
  - **Role in ANPR:** OpenCV is used for image processing tasks such as detecting the number plate in an image, preprocessing the image to enhance text recognition, and segmenting characters.

- **Pytesseract:**

  - **Definition:** Pytesseract is a Python wrapper for Google's Tesseract-OCR Engine, which is a popular OCR tool that can recognize text in images.
  - **Role in ANPR:** Pytesseract is used to perform optical character recognition on the preprocessed number plate image to extract the alphanumeric characters.

Haar cascade Model:

Haar Cascade for Russian Number Plate Detection:

The Haar Cascade file “haarcascade_russian_plate_number.xml” is specifically trained to detect Russian number plates. This pre-trained classifier file is included with the OpenCV library and can be used for number plate recognition tasks. It contains the data necessary to detect number plates in images using the Haar feature-based cascade classifier method.

How Haar Cascades Work:

1. **Feature Selection:**
   - Haar Cascades use Haar-like features, which are essentially digital image features used in object recognition.
   - These features are rectangular areas in an image that calculate the difference in pixel intensity between adjacent rectangular groups of pixels.
2. **Integral Image:**
   - To quickly calculate these features, the algorithm uses an integral image, which allows for rapid summation of pixel values in rectangular regions.
3. **Adaboost Training:**
   - The classifier is trained using a machine learning method called AdaBoost, which selects a small number of important features from a large set and combines them to create a strong classifier.

- The training involves feeding the algorithm with a large dataset of positive images (containing the object of interest) and negative images (not containing the object).

4. **Cascade of Classifiers:**
   - The final classifier is a cascade of several stages, where each stage consists of a strong classifier.
   - During detection, the algorithm applies each stage to a region of the image, and only regions that pass all stages are classified as containing the object.

Code for ANPR by image:

```
import cv2# Read the image

image = cv2.imread("11.jpeg")

# Convert to grayscale

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

```
# Apply Gaussian blur

blurred = cv2.GaussianBlur(gray, (5, 5), 0)

# Perform edge detection

edges = cv2.Canny(blurred, 50, 150)

# Find contours

contours, _ = cv2.findContours(edges.copy(), cv2.RETR_EXTERNAL,

cv2.CHAIN_APPROX_SIMPLE)

# Filter contours based on area and aspect ratio

number_plate_contours = []

for contour in contours:

    (x, y, w, h) = cv2.boundingRect(contour)

    aspect_ratio = w / float(h)
```

```
    area = cv2.contourArea(contour)
    if aspect_ratio > 2.0 and aspect_ratio < 5.0 and area > 1000:
        number_plate_contours.append(contour)
# Import the Pytesseract library
import pytesseract
# Configure Pytesseract
pytesseract.pytesseract.tesseract_cmd = r'Tesseract-OCR\tesseract.exe'  #
Specify the path to the Tesseract executable
# Iterate over potential number plate contours
for contour in number_plate_contours:
    # Extract ROI (Region of Interest)
    (x, y, w, h) = cv2.boundingRect(contour)
    roi = blurred[y:y+h, x:x+w]
    # Use Pytesseract to extract text from the ROI
   # Use Pytesseract to extract text from the ROI with adjusted configuration
    number_plate_text = pytesseract.image_to_string(roi, config='--psm 6 -c
tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 -l
eng')

    # Print or store the extracted text
```

```
        print("Number Plate Text:", number_plate_text)


    # Draw bounding boxes around number plates
    for contour in number_plate_contours:
        (x, y, w, h) = cv2.boundingRect(contour)
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)


    # Display the result
    cv2.imshow('Number Plates Detected', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Code for Live Video Stream:

```
import cv2  # Import the OpenCV library for computer vision tasks

# Path to the Haar Cascade file for detecting number plates
harcascade = "model/haarcascade_russian_plate_number.xml"

# Capture video from the IP camera stream
cap = cv2.VideoCapture('http://192.168.10.6:8080/video')

# Set the width of the video frame
```

```
cap.set(3, 640)  # 3 is the property id for width
# Set the height of the video frame
cap.set(4, 480)  # 4 is the property id for height

# Minimum area of the detected number plate to consider it valid
min_area = 500
# Counter for saved number plate images
count = 0

# Infinite loop to continuously capture frames from the video stream
while True:
    # Read a frame from the video capture
    success, img = cap.read()
    # Check if the frame was successfully read
    if not success:
        continue  # If not, skip to the next iteration
    elif img is not None:  # Check if the frame is not empty
        # Convert the frame to grayscale for easier processing
        img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # Load the Haar Cascade for number plate detection
        plate_cascade = cv2.CascadeClassifier(harcascade)

        # Detect number plates in the grayscale image
        plates = plate_cascade.detectMultiScale(img_gray, 1.1, 4)
```

```
        # Loop through all detected number plates
        for (x, y, w, h) in plates:
            # Calculate the area of the detected number plate
            area = w * h

            # Check if the area of the detected number plate is larger than the minimum area
            if area > min_area:
                # Draw a rectangle around the detected number plate
                cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
                # Put text above the detected number plate
                cv2.putText(img, "Number Plate", (x, y - 5),
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1, (255, 0, 255), 2)

                # Extract the region of interest (ROI) containing the number plate
                img_roi = img[y: y + h, x: x + w]
                # Display the ROI
                cv2.imshow("ROI", img_roi)

        # Display the original frame with the detected number plate highlighted
        cv2.imshow("Result", img)

        # Wait for the 's' key to be pressed to save the detected number plate image
        if cv2.waitKey(1) & 0xFF == ord('s'):
            # Save the ROI image to the "plates" folder with a unique name
            cv2.imwrite("plates/scaned_img_" + str(count) + ".jpg", img_roi)
```

```
        # Draw a filled rectangle on the original frame to show the save status

        cv2.rectangle(img, (0, 200), (640, 300), (0, 255, 0), cv2.FILLED)

        # Put text on the frame indicating the plate has been saved

        cv2.putText(img, "Plate Saved", (150, 265),
cv2.FONT_HERSHEY_COMPLEX_SMALL, 2, (0, 0, 255), 2)

        # Display the frame with the save status

        cv2.imshow("Results", img)

        # Wait for 500 milliseconds

        cv2.waitKey(500)

        # Increment the counter for the next saved image

        count += 1
```

Implementation Of Object Detection:

Libraries and Models involved:

Libraries used:

Flask

Flask is a micro web framework written in Python. It is lightweight and easy to use, making it a popular choice for web development, particularly for small to medium-sized applications.

- **Purpose in the Code:**
  - **Flask**: Creates the Flask application instance.
  - **render_template**: Renders HTML templates.
  - **Response**: Generates HTTP responses.
  - **url_for and redirect**: (Not used in the current code, but commonly used for URL handling and redirects.)

OpenCV

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It includes several hundred computer vision algorithms.

- **Purpose in the Code:**
  - **cv2.dnn.readNet**: Loads a pre-trained deep neural network model.

- **cv2.dnn_DetectionModel**: Creates a detection model from the loaded network.
- **cv2.VideoCapture**: Captures video from a camera or video file.
- **cv2.imshow**: Displays an image or video frame.
- **cv2.flip**: Flips an image or video frame.
- **cv2.putText**: Puts text on an image or video frame.
- **cv2.rectangle**: Draws rectangles on an image or video frame.
- **cv2.imencode**: Encodes an image into a specific format (e.g., JPEG).

Random

Random is a module in Python's standard library that implements pseudo-random number generators for various distributions.

- **Purpose in the Code:**
  - **randint**: Generates random integers. In this code, it is used to generate random colors for detected objects.

Model used:

YOLOv4 (You Only Look Once, version 4) is an advanced object detection model that builds upon the previous YOLO models. It's designed to achieve high performance in terms of both speed and accuracy, making it suitable for real-time applications. Here's an overview of YOLOv4:

Key Features

1. **Speed and Accuracy:**
   - **Speed:** YOLOv4 is optimized for real-time object detection. It processes images quickly, making it suitable for applications like video surveillance and autonomous driving.
   - **Accuracy:** YOLOv4 achieves high accuracy in object detection, with improvements in precision and recall over previous versions.
2. **Single-Stage Detector:**
   - YOLOv4 is a single-stage detector, meaning it predicts bounding boxes and class probabilities directly from full images in a single evaluation. This contrasts with two-stage detectors like Faster R-CNN, which first propose regions of interest and then classify them.
3. **Architecture Improvements:**
   - **Backbone:** YOLOv4 uses CSPDarknet53 as its backbone, which improves the learning ability and generalization of the model.
   - **Neck:** It incorporates PANet (Path Aggregation Network) for parameter aggregation from different backbone levels.
   - **Head:** The head of YOLOv4 predicts the bounding boxes and class probabilities.
4. **Bag of Freebies and Specials:**

- **Bag of Freebies:** Techniques that improve accuracy without affecting inference speed, such as data augmentation, label smoothing, and class label assignment.
- **Bag of Specials:** Techniques that slightly increase inference time but significantly improve accuracy, such as Mish activation, SPP (Spatial Pyramid Pooling), and SAM (Spatial Attention Module).

5. **Anchor Boxes:**
   - Uses anchor boxes of different sizes to predict bounding boxes, allowing the model to detect objects of varying scales.
6. **Advanced Loss Functions:**
   - YOLOv4 utilizes advanced loss functions like CIoU (Complete Intersection over Union) loss, which provides better localization accuracy for the predicted bounding boxes.

Code implementation of Object Detection for live video stream:

```
from flask import *  # Import all necessary functions from the Flask framework

import cv2  # Import OpenCV library for computer vision tasks

from random import randint  # Import randint function to generate random integers



app = Flask(__name__)  # Initialize a Flask application



# Load the pre-trained YOLOv4-tiny model weights and configuration file

dnn = cv2.dnn.readNet('yolov4-tiny.weights', 'yolov4-tiny.cfg')

# Create a DNN detection model from the loaded network

model = cv2.dnn_DetectionModel(dnn)

# Set the input parameters for the detection model

model.setInputParams(size=(416, 416), scale=1/255, swapRB=True)

# Load the class labels from a file

with open('classes.txt') as f:

    classes = f.read().strip().splitlines()
```

```
# Open the default camera (webcam)
capture = cv2.VideoCapture(0)
# Set the width of the video frame
capture.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
# Set the height of the video frame
capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)

# Dictionary to store random colors for each class label
color_map = {}

# Function to generate video frames
def gen_frames():
    while True:
        # Read a frame from the video capture
        success, img = capture.read()
        # Make a copy of the original frame
        frame = img.copy()
        # Flip the frame horizontally
        frame = cv2.flip(frame, 1)
        # Check if the frame was successfully read
        if not success:
            break  # If not, exit the loop
        else:
            try:
                # Detect objects in the frame using the detection model
```

```
            class_ids, confidences, boxes = model.detect(frame)
            # Loop through the detected objects
            for id, confidence, box in zip(class_ids, confidences, boxes):
                x, y, w, h = box  # Get the bounding box coordinates
                obj_class = classes[id]  # Get the class label

                # Assign a random color to the class if not already assigned
                if obj_class not in color_map:
                    color = (randint(0, 255), randint(0, 255), randint(0, 255))
                    color_map[obj_class] = color
                else:
                    color = color_map[obj_class]

                # Put the class label and confidence on the frame
                cv2.putText(frame, f'{obj_class.title()} {format(confidence, ".2f")}', (x, y-10),
cv2.FONT_HERSHEY_DUPLEX, 1, color, 2)
                # Draw the bounding box around the detected object
                cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
        except Exception as e:
            print("Error during detection:", e)  # Print any errors that occur during detection

        # Display the processed frame
        cv2.imshow('Video Capture', frame)

        # Encode the frame in JPEG format
        ret, buffer = cv2.imencode('.jpg', frame)
```

```
        frame_bytes = buffer.tobytes()  # Convert the encoded frame to bytes
        # Yield the frame bytes in a format suitable for streaming
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame_bytes + b'\r\n')

@app.route('/')
def index():
    # Render the index.html template
    return render_template('index.html')

@app.route('/video_feed')
def video_feed():
    # Return the video feed as a response with the appropriate MIME type for streaming
    return Response(gen_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')

if __name__ == '__main__':
    # Run the Flask application in debug mode
    app.run(debug=True)
```

Complete ANPR implementation

Libraries used

Flask:

- **Purpose**: Flask is a micro web framework for Python used to build web applications.
- **Usage**: Handles routing, request handling, and response generation for different URLs.

OpenCV (cv2):

- **Purpose**: OpenCV (Open Source Computer Vision Library) is used for image and video processing tasks.

- **Usage**: Performs tasks such as reading video streams, object detection, preprocessing images, and applying text recognition.

pytesseract:

- **Purpose**: pytesseract is a Python wrapper for Google's Tesseract-OCR Engine, used for optical character recognition (OCR).
- **Usage**: Converts images containing text into machine-readable text data.

ultralytics.YOLO:

- **Purpose**: YOLO (You Only Look Once) is a real-time object detection system.
- **Usage**: Performs object detection tasks, identifying and localizing objects within images or video frames.

functools.wraps:

- **Purpose**: functools.wraps is a decorator for updating the attributes of the wrapped function to match those of the original function.
- **Usage**: Used to preserve the metadata of the decorated functions, especially useful in Flask for maintaining function signatures and docstrings.

Other Libraries:

- **spellchecker**: Used for spell checking, though not explicitly utilized in the provided code snippet.
- **re**: Regular expression operations for string matching and manipulation.
- **random.randint**: Generates random integers, used for assigning colors in the object detection visualization.

Code Implementation:

```
from flask import *

import cv2

import pytesseract

import re

from random import randint

from spellchecker import SpellChecker

from ultralytics import YOLO
```

```
from functools import wraps

# Flask app initialization
app = Flask(__name__)

# Function to redirect HTTP requests to HTTPS
def redirect_to_https(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if request.scheme != 'https' and not app.debug:
            return redirect(request.url.replace('http://', 'https://'), code=301)
        return f(*args, **kwargs)
    return decorated_function

# Preprocessing image (e.g., convert to grayscale, apply filters)
def preprocess_image(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)
    return blurred_image

# Postprocessing text (e.g., remove extra whitespace and special characters)
def postprocess_text(text):
    cleaned_text = text.strip()  # Remove leading and trailing whitespace
    cleaned_text = cleaned_text.replace('\n', ' ')  # Replace newline characters with spaces
    return cleaned_text
```

```
# Function to generate video frames for object detection using YOLOv4
def Obj_gen_frames():
    dnn = cv2.dnn.readNet('yolov4-tiny.weights', 'yolov4-tiny.cfg')
    model = cv2.dnn_DetectionModel(dnn)
    model.setInputParams(size=(416, 416), scale=1/255, swapRB=True)
    with open('classes.txt') as f:
        classes = f.read().strip().splitlines()
    capture = cv2.VideoCapture('https://10.184.80.253:8080/video')
    capture.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
    capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
    color_map = {}
    while True:
        success, img = capture.read()  # Read the camera frame
        frame = img.copy()  # Make a copy of the original frame
        frame = cv2.flip(frame, 1)  # Flip the frame horizontally
        if not success:
            break
        else:
            try:
                class_ids, confidences, boxes = model.detect(frame)
                for id, confidence, box in zip(class_ids, confidences, boxes):
                    x, y, w, h = box
                    obj_class = classes[id]
```

```
                if obj_class not in color_map:
                    color = (randint(0, 255), randint(0, 255), randint(0, 255))
                    color_map[obj_class] = color
                else:
                    color = color_map[obj_class]

                cv2.putText(frame, f'{obj_class.title()} {format(confidence, ".2f")}', (x, y-10), cv2.FONT_HERSHEY_DUPLEX, 1, color, 2)
                cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
        except Exception as e:
            print("Error during detection:", e)

        cv2.imshow('Video Capture', frame)

        ret, buffer = cv2.imencode('.jpg', frame)
        frame_bytes = buffer.tobytes()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame_bytes + b'\r\n')

# Function to generate video frames for number plate recognition using Haar Cascade
def NPR_gen_frames():
    cap = cv2.VideoCapture('https://10.184.80.253:8080/video')
    harcascade = "Haarcascades/haarcascade_russian_plate_number.xml"
    cap.set(3, 640)  # Set width
    cap.set(4, 480)  # Set height
    min_area = 500
```

```
    count = 0
    while True:
        success, img = cap.read()  # Read the camera frame
        if not success:
            break
        else:
            plate_cascade = cv2.CascadeClassifier(harcascade)
            img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            plates = plate_cascade.detectMultiScale(img_gray, 1.1, 4)

            for (x, y, w, h) in plates:
                area = w * h

                if area > min_area:
                    cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
                    cv2.putText(img, "Number Plate", (x, y - 5),
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1, (255, 0, 255), 2)

                    img_roi = img[y: y + h, x:x + w]
                    cv2.imshow("ROI", img_roi)

            ret, buffer = cv2.imencode('.jpg', img)
            frame = buffer.tobytes()
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
```

```
# Function to predict number plate text from an image

def prediction1(img):

    image = cv2.imread(img)

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    blurred = cv2.GaussianBlur(gray, (5, 5), 0)

    edges = cv2.Canny(blurred, 50, 150)

    contours, _ = cv2.findContours(edges.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    number_plate_contours = []

    for contour in contours:

        (x, y, w, h) = cv2.boundingRect(contour)

        aspect_ratio = w / float(h)

        area = cv2.contourArea(contour)

        if aspect_ratio > 2.0 and aspect_ratio < 5.0 and area > 1000:

            number_plate_contours.append(contour)

    pytesseract.pytesseract.tesseract_cmd = r'Tesseract-OCR\tesseract.exe'

    for contour in number_plate_contours:

        (x, y, w, h) = cv2.boundingRect(contour)

        roi = blurred[y:y + h, x:x + w]

        number_plate_text = pytesseract.image_to_string(roi, config='--psm 6 -c
tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 -l eng')

        return number_plate_text

    return ''



# Function to perform object detection on an image using YOLOv8

def solve(img):
```

```
    model = YOLO('yolov8x.pt')
    results = model(img, show=False)
    if isinstance(results, list):
        for i, img_result in enumerate(results):
            output_path = f"static\processed_image_{i}.jpg"
            img_result.save(output_path)
    print(output_path)
    return output_path

# Routes for rendering HTML templates
@app.route('/')
def hello():
    return render_template('main.html')

@app.route('/NPR')
def NPR():
    return render_template('number_plate.html')

@app.route('/NPRImage')
def NPRImage():
    return render_template('number_plate_image.html')

@app.route('/TextImage')
def TextImage():
    return render_template('text_extraction.html')
```

```
# Route to predict number plate text from an image
@app.route('/predict1', methods=["GET", "POST"])
def predict1():
    file = request.files['file']
    file_path = r"static/Storage" + file.filename
    file.save(file_path)
    k = prediction1(file_path)
    return render_template('number_plate_image.html', ans=k)

# Route to load video for number plate recognition
@app.route('/NPRVideoLoad')
def NPRVideoLoad():
    return render_template('number_plate_video.html')

# Route to stream video for number plate recognition
@app.route('/NPRVideo')
def NPRVideo():
    return Response(NPR_gen_frames(), mimetype='multipart/x-mixed-replace;
boundary=frame')

# Route to detect text in an uploaded image
@app.route('/textdetect', methods=["GET", "POST"])
def textdetect():
    file = request.files['file']
    file_path = r"static\Storage" + file.filename
```

```
    file.save(file_path)
    image = cv2.imread(file_path)
    pytesseract.pytesseract.tesseract_cmd = r'Tesseract-OCR\tesseract.exe'
    processed_image = preprocess_image(image)
    text = pytesseract.image_to_string(processed_image, lang='eng', config='--psm 6')
    processed_text = postprocess_text(text)
    k = processed_text
    return render_template('TextDetection.html', ans=k)


@app.route('/object')
def object():
    return render_template('object_detection.html')


# Route to stream video for object detection
@app.route('/ObjVideo')
def ObjVideo():
    return Response(Obj_gen_frames(), mimetype='multipart/x-mixed-replace;
boundary=frame')


@app.route('/ObjVideoLoad')
def ObjVideoLoad():
    return render_template('object_detection_video.html')


@app.route('/ObjImage')
def ObjImage():
    return render_template('object_detection_image.html')
```

```
# Route to handle object detection on an uploaded image

@app.route('/predict2', methods=["GET", "POST"])

def predict2():

    file = request.files['file']  # Get uploaded file

    file_path = r"static\Storage" + file.filename  # Define file path for storage

    file.save(file_path)  # Save uploaded file

    k = solve(file_path)  # Perform object detection and get result

    return render_template('object_detection_image.html', file=k[7:])  # Render HTML template with processed image



# Run the Flask application

if __name__ == '__main__':

    app.run(host='0.0.0.0', debug=True, ssl_context='adhoc')  # Run app with debug mode and ad-hoc SSL context
```

Conclusion

The primary objective of this ANPR project was to develop a system capable of accurately detecting and recognizing vehicle license plates from images. The developed ANPR system demonstrated an accuracy of 75% in plate detection and 62% in character recognition, which is competitive with existing solutions in the field.

While the system performs well under controlled conditions, its accuracy drops in low-light environments and with highly reflective plates. Handling variations in plate fonts and damaged plates also remains a challenge. This ANPR system can be utilized in various applications such as traffic management, toll collection, and law enforcement, thereby contributing to more efficient and automated vehicle monitoring systems.

Future work could focus on improving the system's robustness to different lighting conditions and plate styles, as well as incorporating machine learning algorithms to enhance character recognition accuracy.

In conclusion, the ANPR system developed in this project demonstrates significant potential for real-world applications, and further enhancements could lead to even greater performance and versatility.