1. Automatic Number Plate Detection using Traditional Methods

Overview

This project aims to implement an Automatic Number Plate Recognition (ANPR) system using Python. The system processes images, detects license plates, and performs optical character recognition (OCR) to extract plate numbers.

Modules Used

1. `cv2` (OpenCV)

Purpose: OpenCV is an open-source computer vision library used for real-time image processing tasks.

- **Why used**: It is essential for tasks such as image reading, resizing, thresholding, contour detection, and drawing bounding boxes. It helps in preprocessing the input image to make it suitable for OCR and license plate detection.

2. `numpy`

Purpose: NumPy is a powerful library for numerical operations in Python, particularly for working with arrays.

- **Why used**: It is used here to handle image data (images are essentially large matrices of pixel values), and for performing mathematical operations like matrix manipulation (used for image transformations, thresholding, etc.).

3. `pytesseract`

Purpose: Pytesseract is a Python wrapper for Tesseract, an open-source OCR engine.

- **Why used**: Pytesseract is used for extracting text from images. It plays a crucial role in recognizing characters on the detected license plates and converting them into readable text.

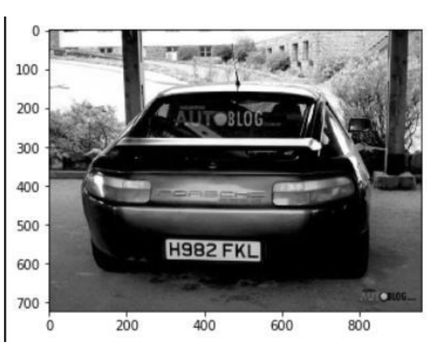
4. `pytesseract.image_to_string()`

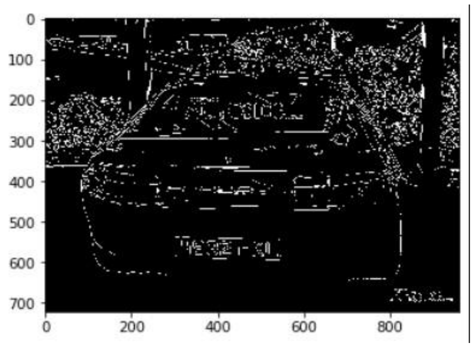
Purpose: This function from pytesseract is used to convert an image of text (in this case, a license plate) into actual text.

- **Why used**: It is the core function for performing OCR on the license plate images. The extracted text (license plate number) can be processed further for use in applications like automatic parking systems or toll collection.

Image Preprocessing Steps

1. Grayscale Conversion

- Converts the image into a single channel (grayscale) format, simplifying the data and enhancing the performance of image analysis.
- **Why used**: Grayscale images are often easier to process and less computationally expensive than color images.
- 



2. Thresholding

- Applies a binary threshold to the image, separating the plate from the background.
- **Why used**: Thresholding enhances the contrast and makes it easier for the algorithm to detect edges and contours, which helps in detecting the license plate.

3. Contour Detection

- Detects the contours in the image, helping identify potential license plate regions.
- **Why used**: Contours are useful for identifying objects in an image. The detected contours are filtered to find the rectangular shape of the license plate.

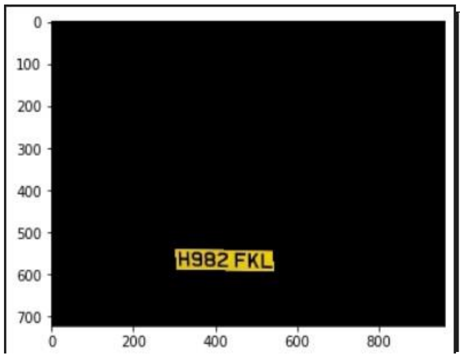
Optical Character Recognition (OCR)

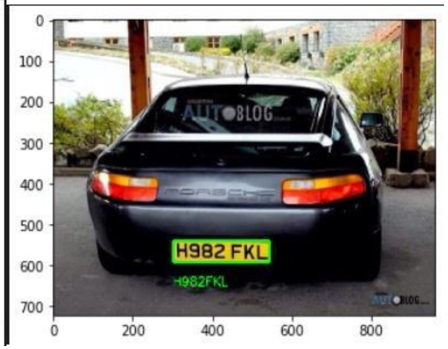
Once the license plate is isolated, the image of the plate is passed to pytesseract for text extraction. The extracted text is the license plate number.

Conclusion

This ANPR project utilizes OpenCV for image processing and Tesseract for optical character recognition. The combination of these libraries makes it possible to implement a robust and efficient number plate recognition system.

Final Outputs:





2. Number Plate Detection and Recognition Using CNN

This project implements a pipeline for detecting and recognizing vehicle number plates using YOLOv3 and EasyOCR. The program processes images from a specified input directory, detects license plates, and extracts text using OCR.

Prerequisites

Ensure the following dependencies are installed:

- Python 3.x
- OpenCV
- NumPy
- Matplotlib
- EasyOCR

File Structure

Ensure the following directory structure is in place:

```
project_directory/
├── model/
│   ├── cfg/
│   │   └── darknet-yolov3.cfg
│   ├── weights/
│   │   └── model.weights
│   └── class.names
├── data/
│   └── [input images]
├── util.py
└── main.py
```

Code Description

Constants and File Paths

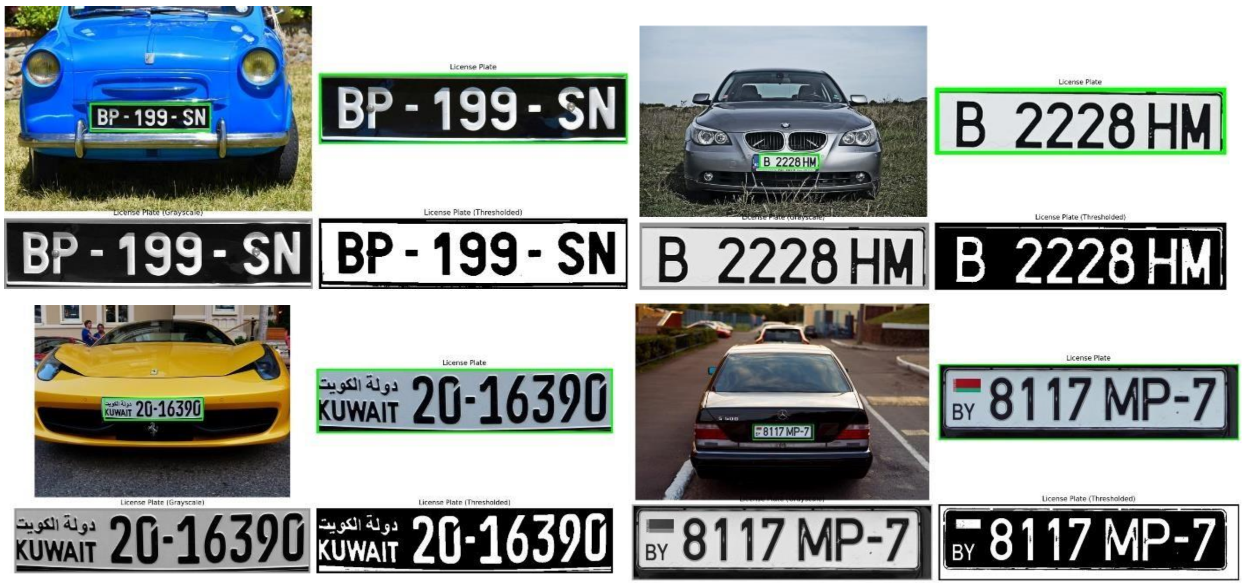
- **model_cfg_path**: Path to YOLOv3 configuration file.
- **model_weights_path**: Path to YOLOv3 weights file.
- **class_names_path**: Path to file containing class names.
- **input_dir**: Directory containing input images.

Loading YOLOv3 Model

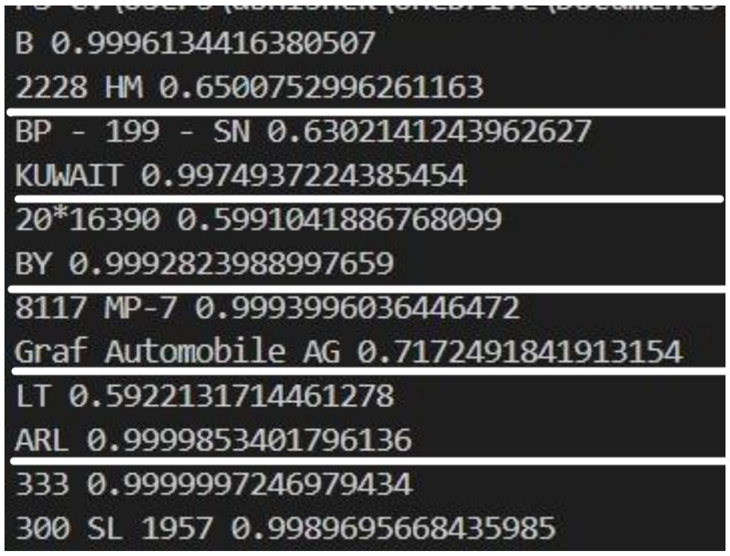
- Loads the YOLOv3 model using OpenCV's `cv2.dnn.readNetFromDarknet(`

Notes

- Ensure the `util.py` file contains the required functions (`get_outputs` and `NMS`).
- Adjust the confidence threshold in the OCR step (`if text_score > 0.4`) as needed.



Final Output Results of Our Model:



```
B 0.9996134416380507
2228 HM 0.6500752996261163
BP - 199 - SN 0.6302141243962627
KUWAIT 0.9974937224385454
20*16390 0.5991041886768099
BY 0.9992823988997659
8117 MP-7 0.9993996036446472
Graf Automobile AG 0.7172491841913154
LT 0.5922131714461278
ARL 0.9999853401796136
333 0.9999997246979434
300 SL 1957 0.9989695668435985
```

Model Configuration Documentation

This document provides an in-depth explanation of the key parameters and layers in the `darknet-yolov3.cfg` file used for the YOLOv3-based number plate detection system.

General Network Configuration

Net Section

The `net` section specifies the overall network and training parameters: -

Input Image Dimensions: - `width=608`, `height=608` (Input images are resized to 608x608). - `channels=3` (RGB color images).

Training Parameters: - Batch size: `batch=64`. - Subdivisions: `subdivisions=16` (Each batch is divided into smaller mini-batches for memory efficiency).

Data Augmentation: - `angle=0`: No rotation of images. - `saturation=1.5`, `exposure=1.5`, `hue=0.1`: Adjustments for brightness, color intensity, and hue for augmentation.

Optimizer: - Momentum: `momentum=0.9`. - Weight decay: `decay=0.0005`.

Learning Rate: - Initial learning rate: `learning_rate=0.001`. - Burn-in: `burn_in=1000` (Gradual increase in learning rate during the first 1000 iterations).

Policy: `steps=400000,450000` with scales `scales=0.1, 0.1` (Learning rate decreases by 90% at specified steps).

Layer Types and Descriptions

The network architecture consists of the following layers:

1. `convolutional` Layers

- **Total Count:** 75 layers.
- **Purpose:** Feature extraction and transformation.
- **Parameters:**
  - `batch_normalize=1`: Enables batch normalization for faster convergence.
  - `filters`: Number of output feature maps (e.g., 32, 64, etc.).
  - `size`: Kernel size (e.g., `size=3` for a 3x3 filter).
  - `stride`: Determines the step size (e.g., `stride=1` for standard convolution, `stride=2` for downsampling).
  - `pad=1`: Adds padding to maintain spatial dimensions.
  - `activation=leaky`: Uses Leaky ReLU activation.

2. `shortcut` Layers

- **Total Count:** 23 layers.
- **Purpose:** Implements residual connections to enhance gradient flow and prevent vanishing gradients.
- **Functionality:** Adds outputs from previous layers to the current layer for improved training stability.

3. `yolo` Layers

- **Total Count:** 3 layers.
- **Purpose:** Object detection by predicting bounding boxes and class probabilities.
- **Anchor Boxes:**
  - Predefined bounding box sizes optimized for different object scales.
- **Detection Scales:**
  - Each YOLO layer handles a specific scale of objects (small, medium, large).

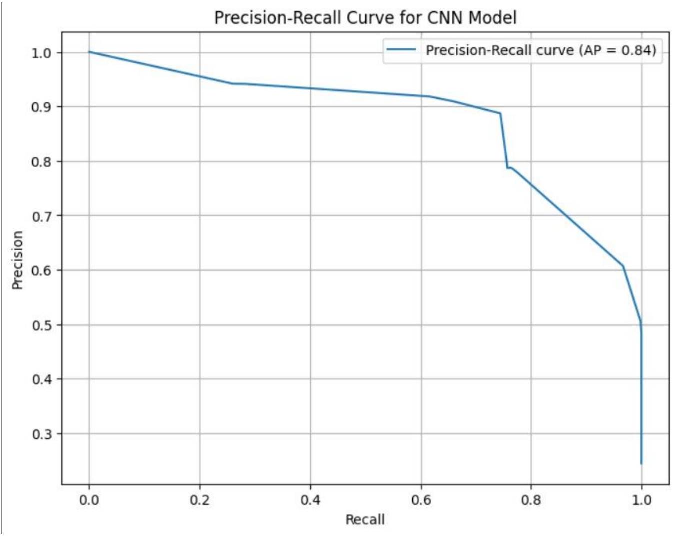
4. `route` Layers

- **Total Count:** 4 layers.
- **Purpose:** Combines feature maps from different layers to integrate multi-scale features.

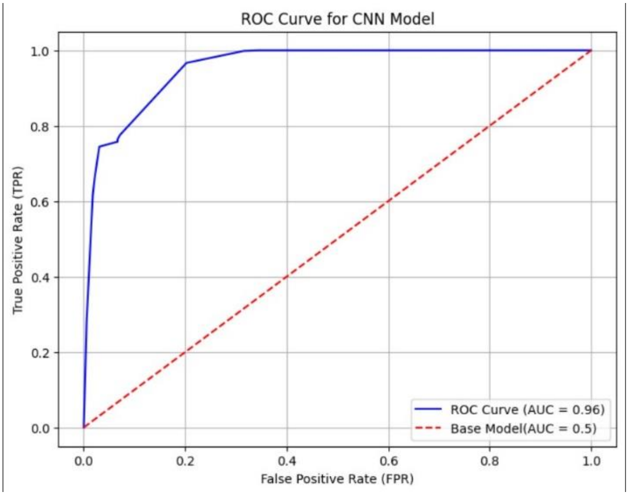
5. `upsample` Layers

- **Total Count:** 2 layers.
- **Purpose:** Increases spatial resolution of feature maps for multi-scale object detection.

Final Results of Our Model:

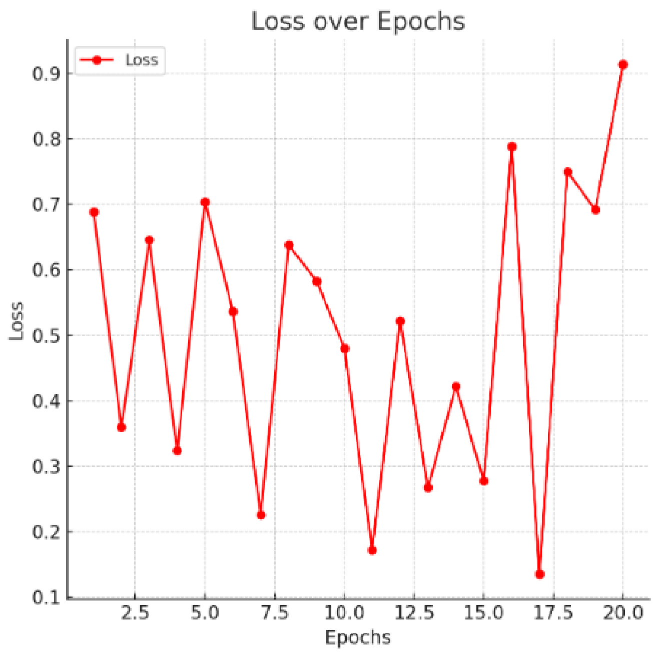


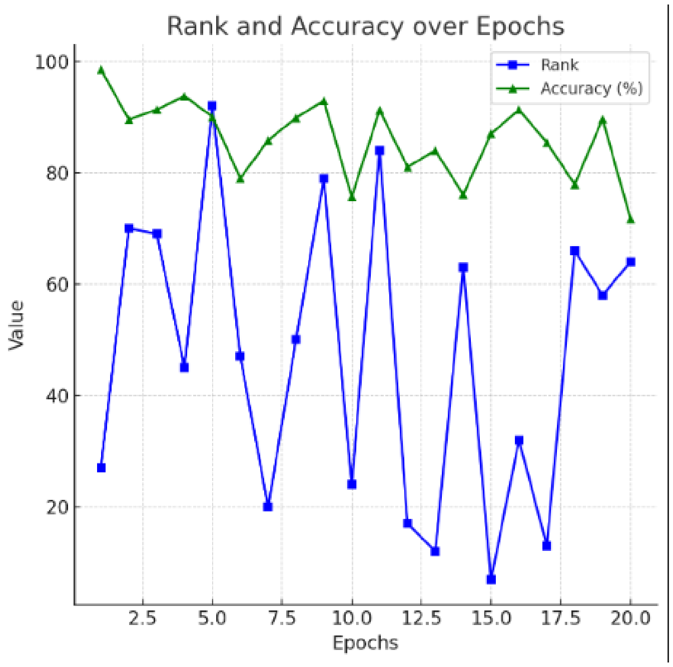
Precision-Recall Curve



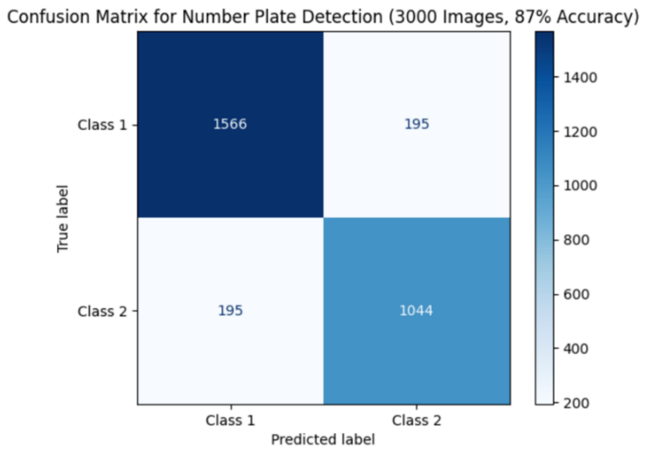
ROC - Curve

Training Set results:





Test Set Results:



	Class 1	Class 2
Class 1	1566	195
Class 2	195	1044