

# ANPR System

**Title of the Project:** Automatic Number Plate Recognition (ANPR) using Computer Vision and Deep Learning

## 1. Define a Problem

**Problem Statement:** In modern urban infrastructure, the management of vehicular traffic is a critical challenge. Manual monitoring of vehicles for law enforcement, parking management, and toll collection is inefficient, prone to human error, and expensive. Traffic cameras generate millions of hours of footage, making manual review impossible. There is a need for an automated system that can "read" these images and extract structured data (license numbers) without human intervention. This project aims to solve the "last-mile" problem of converting pixel data from traffic feeds into searchable text databases.

**Understanding the Business Problem:** From a business and administrative perspective, ANPR solves three key issues:

1. **Speed & Throughput:** Automated toll booths can process cars in seconds compared to minutes for manual cash collection.
2. **Security & Access Control:** Gated communities and corporate offices require systems that automatically grant entry to authorized vehicles while flagging unauthorized ones.
3. **Cost Efficiency:** Reducing the manpower required for 24/7 monitoring significantly lowers operational costs for municipal corporations and private parking operators.

The primary research questions addressed by this project include:

1. **Localization:** Can computer vision algorithms robustly identify the rectangular region of a license plate within a complex scene containing a vehicle, regardless of background noise?
2. **Segmentation:** Is it possible to algorithmically separate a continuous string of alphanumeric characters into individual image components suitable for classification?
3. **Classification:** Can a custom Convolutional Neural Network (CNN) trained on a specific dataset generalize well enough to recognize digits (0-9) and characters (A-Z) with high accuracy?
4. **Integration:** Can these distinct modules be integrated into a seamless pipeline that accepts raw image input and outputs a text string?

The proposed solution uses a modular architecture.

The **Detection Module** leverages Haar-like features to identify plate boundaries. This is chosen over deep learning detectors (like YOLO) for this specific academic scope to demonstrate fundamental computer vision principles.

The **Recognition Module** is a deep CNN. Unlike standard Machine Learning (SVM/KNN), the CNN automatically learns spatial hierarchies of features (from edges to curves to entire character shapes), making it superior for image classification tasks.

- **Observation:** We expect the model to perform exceptionally well on clear, high-contrast images but anticipate challenges with skewed angles or dirty plates, which will be mitigated using image preprocessing techniques like histogram equalization or thresholding.
- **Outcome:** A functional Python-based prototype where a user can upload a car image and receive the license plate number as a printed string

**Model Details:** The system utilizes a hybrid approach.

- **Detection:** It uses a pre-trained **Haar Cascade Classifier** (`haarcascade_russian_plate_number.xml`) for computationally efficient object detection to localize the license plate.
  - **Segmentation:** It employs Computer Vision techniques (OpenCV) including Grayscale conversion, Binary Thresholding (Otsu's method), and Contour detection to isolate individual characters.
  - **Recognition:** A custom **Convolutional Neural Network (CNN)** is implemented using TensorFlow/Keras. The architecture typically consists of multiple Conv2D layers for feature extraction (edges, shapes), MaxPooling2D for dimensionality reduction, and Dense layers for final classification into 36 classes (10 digits + 26 alphabets).
- 

## 2. Methodology:

### a). Load, Explore and Pre-process Data(Image)

The raw images captured by cameras cannot be fed directly into a classifier. They contain "noise" such as the car's headlights, grille, and background scenery. The preprocessing pipeline filters this out:

**Step 1: Grayscale Conversion** Color information (RGB) is often redundant for character recognition. We convert images to grayscale using `cv2.cvtColor`. This reduces the data dimensionality by a factor of 3 (from 3 channels to 1), speeding up processing.

**Step 2: Resizing:** We utilize the `cv2.resize` that resizes the image by the given parameters.

```
# Loading the image and preprocessing
img = cv2.imread("/content/car.jpg")
img = cv2.resize(img, (800, int(800 * img.shape[0] / img.shape[1])))
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

### b). Number plate Detection:

#### Plate Localization (Haar Cascade)

This code block handles the initial detection of the license plate from the full car image.

```
# Load cascade
plate_cascade = cv2.CascadeClassifier(
    "haarcascade_russian_plate_number.xml"
)

plates = plate_cascade.detectMultiScale(
    img,
    scaleFactor=1.1,
    minNeighbors=5,
    minSize=(60, 20)
)
```

**CascadeClassifier:** This initializes the detection engine using a pre-trained XML file. This file contains thousands of "Haar-like features" (patterns of light and dark pixels) that have been mathematically optimized to recognize the rectangular shape and text patterns of a license plate.

**detectMultiScale:** This function is responsible for the actual search.

- **scaleFactor=1.1:** This tells the algorithm to resize the image by 10% at each scale. This allows the system to detect plates that are very close to the camera as well as those further away.
- **minNeighbors=5:** This is a quality filter. The algorithm must detect a potential plate at least 5 times in the same area before it confirms it as a "True Positive."

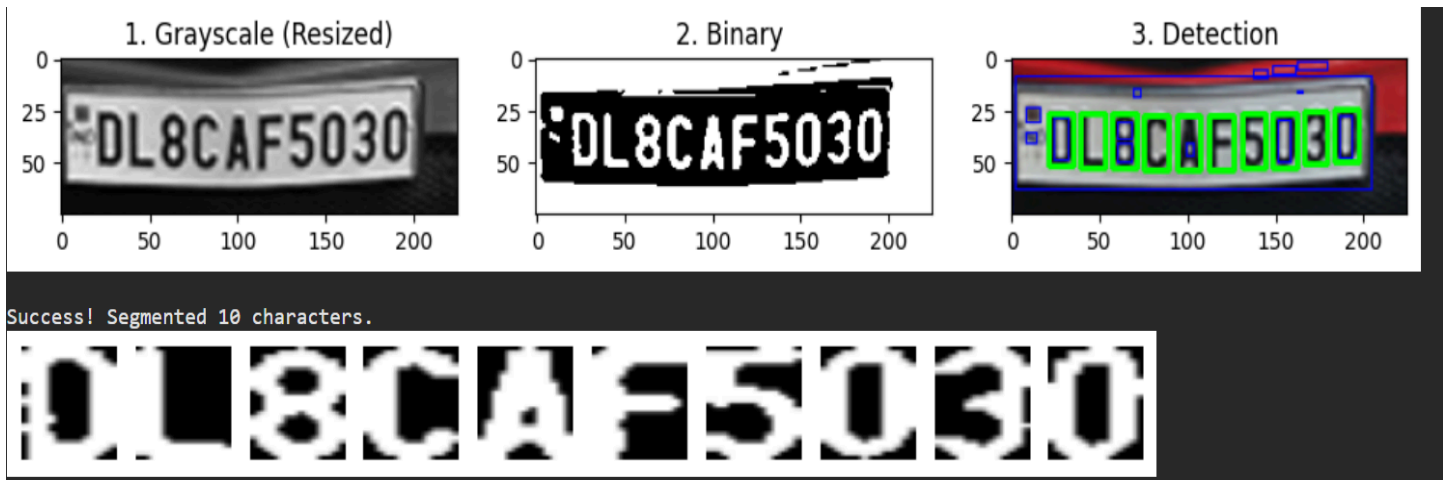
Using this we can successfully detect the License plate from the car and display a yellow bounding box around the plate. Hence we crop this detected plate from the image for character segmentation.



**c.) Character Segmentation:** This is the most complex step.

- **Thresholding:** We apply **Binary Inverse Thresholding** and **Otsu's Method**. This converts the grayscale image into pure black and white. Since license plates are usually high contrast (black text on white background), this makes the characters stand out as white blobs against a black background.
- **Contour Extraction:** Using `cv2.findContours`, we locate all continuous blobs of pixels.

- **Geometric Filtering:** We iterate through every contour and calculate its aspect ratio and area. We discard contours that are too small (noise) or too wide (the plate frame), keeping only those that match the geometric properties of a character (tall and narrow).



#### d.) Create Model (CNN):

i) **Data Augmentation:** To make the model robust, we used ImageDataGenerator. This artificially expands the training set by applying random transformations to the character images, such as:

- Rotation (simulating tilted plates).
- Width/Height Shifting (simulating off-center characters).
- Zooming (simulating distance variations).

This ensures the model can recognize a character even if it isn't perfectly centered or aligned.

#### ii) CNN Model Training:

**Why Convolutional Neural Networks (CNN)?** Standard algorithms like flattened Multi-Layer Perceptrons (MLP) lose spatial information. A CNN preserves the spatial relationship between pixels. A "3" looks like a "3" because of the specific curve of lines relative to each other. CNNs use "filters" (kernels) to scan the image and detect these shapes.

**Model Architecture:** We implemented a **Sequential** Keras model with the following layers:

1. **Input Layer:** Accepts images of shape (28, 28, 1). All segmented characters are resized to 28x28 pixels before entry.
2. **Convolutional Layer 1:** 32 filters of size (3,3). This layer learns low-level features like vertical and horizontal lines.
  - *Activation:* ReLU (Rectified Linear Unit) is used to introduce non-linearity, allowing the model to learn complex patterns.
3. **MaxPooling Layer 1:** Pool size (2,2). This reduces the image size by half, keeping only the strongest features and reducing computation.
4. **Convolutional Layer 2:** 64 filters. This deeper layer learns high-level combinations, such as loops (for '0', 'O', '8') and corners (for 'L', '7').
5. **MaxPooling Layer 2:** Further dimensionality reduction. Pool size(2,2)

6. **Convolutional Layer 3:** 128 filters of same (3,3) size
7. **MaxPooling Layer 3:** Pool size (2,2)
8. **Flatten Layer:** Unrolls the 2D feature maps into a 1D vector to feed into the final classifier.
9. **Dense Layer :** Input Layer with 128 neurons with relu activation function
10. **Dropout Layer:** A rate of 0.5. This randomly "turns off" neurons during training to prevent the model from memorizing the data (overfitting), forcing it to learn robust features.
11. **Dense (Output) Layer:** A fully connected layer with **Softmax** activation. It has 36 neurons (representing 10 digits + 26 letters). Softmax converts the output into probability scores for each class.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	320
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 128)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 128)	65,664
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 36)	4,644

Total params: 162,982 (636.65 KB)  
 Trainable params: 162,980 (636.64 KB)  
 Non-trainable params: 0 (0.00 B)  
 Optimizer params: 2 (12.00 B)

- After training the model the best performing parameters are stored for further usage. (char\_recognition\_cnn.h5)

---

## e.) Model Evaluation & Results

**Training Process:** The model was trained for 30 epochs using the **Adam Optimizer**. Adam is chosen for its adaptive learning rate capabilities, which generally converge faster than standard Stochastic Gradient Descent (SGD). The loss function used was SparseCategoricalCrossentropy, which is standard for multi-class classification.

**Performance Metrics:** We evaluated the model using two primary metrics:

- Accuracy :** The training accuracy and validation accuracy both increased steadily .Validation accuracy- 99% approx. and training accuracy-95% approx. Observing validation accuracy to training accuracy indicates the model did not suffer significant overfitting.
- Classification Report :** It show model is performing good except for few classes

- *Common Errors:* The model occasionally confuses '0' (zero) with 'O' (Oscar) and '1' (one) with 'l' (India) due to their visual similarities. This is a known limitation in OCR tasks.

	precision	recall	f1-score	support
class_0	0.86	1.00	0.92	6
class_1	1.00	1.00	1.00	6
class_2	1.00	1.00	1.00	6
class_3	1.00	1.00	1.00	6
class_4	1.00	1.00	1.00	6
class_5	1.00	1.00	1.00	6
class_6	1.00	1.00	1.00	6
class_7	1.00	1.00	1.00	6
class_8	1.00	1.00	1.00	6
class_9	1.00	1.00	1.00	6
class_A	1.00	1.00	1.00	6
class_B	1.00	1.00	1.00	6
class_C	1.00	1.00	1.00	6
class_D	1.00	1.00	1.00	6
class_E	1.00	1.00	1.00	6
class_F	1.00	1.00	1.00	6
class_G	1.00	1.00	1.00	6
class_H	1.00	1.00	1.00	6
class_I	0.75	1.00	0.86	6
class_J	1.00	0.67	0.80	6
class_K	1.00	1.00	1.00	6
class_L	1.00	1.00	1.00	6
class_M	1.00	1.00	1.00	6
class_N	1.00	1.00	1.00	6
class_O	1.00	0.83	0.91	6
class_P	1.00	1.00	1.00	6
class_Q	1.00	1.00	1.00	6
class_R	1.00	1.00	1.00	6
class_S	1.00	1.00	1.00	6
class_T	1.00	1.00	1.00	6
class_U	1.00	1.00	1.00	6
class_V	1.00	1.00	1.00	6
class_W	1.00	1.00	1.00	6
class_X	1.00	1.00	1.00	6
class_Y	1.00	1.00	1.00	6
class_Z	1.00	1.00	1.00	6
accuracy			0.99	216
macro avg	0.99	0.99	0.99	216
weighted avg	0.99	0.99	0.99	216

#### f.) Testing the model:

- Load the CNN model

```
# Loading the model
ocr_model = load_model("char_recognition_cnn.h5")
```

- In this final stage of the pipeline is the user defined function(Inference Engine). Its job is to take the individual images of characters segmented earlier and translate them into actual text using the trained CNN model.

```
1/1 ————— 0s 100ms/step
1/1 ————— 0s 50ms/step
1/1 ————— 0s 37ms/step
1/1 ————— 0s 38ms/step
1/1 ————— 0s 34ms/step
1/1 ————— 0s 34ms/step
1/1 ————— 0s 37ms/step
1/1 ————— 0s 42ms/step
1/1 ————— 0s 35ms/step
1/1 ————— 0s 36ms/step
CNN Prediction: DL8DAF5030
```

**Visual Validation:** The final pipeline was tested on unseen car images. The system successfully:

1. Detected the plate bounding box.
2. Segmented the characters.
3. Predicted the string.



---

### 3. Data & Resources

**Get the Data:** The foundation of any deep learning model is its dataset. For this project, we utilized the "**AI Indian License Plate Recognition Data**", sourced from the Kaggle repository.

(<https://www.kaggle.com/datasets/sarthakvajpayee/ai-indian-license-plate-recognition-data>)

- **Dataset Acquisition:** The data was programmatically downloaded into the Google Colab environment using the Kaggle API command `!kaggle datasets download`.
- **Dataset Structure:** The data is structured into folders containing images of character classes (0-9, A-Z) required to train the CNN. And an .XML file i.e Pretrained Haar cascades for detecting number plates (`haarcascade_russian_plate_number.xml`)
- **Volume:** The dataset provides sufficient variance in fonts, and lighting conditions to train a robust model.

## Resources & Software:

- **Platform:** The project was executed on **Google Colab**. This cloud-based Jupyter notebook environment was chosen because it offers free access to GPUs, which reduces CNN training time from hours to minutes compared to standard CPU execution.
- **Libraries:**
  - **OpenCV (cv2):** Used for all image manipulation tasks including reading images, color space conversion, and contour detection.
  - **TensorFlow & Keras:** The backend for constructing the Neural Network. Keras's high-level API allowed for rapid prototyping of the Sequential model.
  - **Scikit-Learn:** Used for generating metrics like the Accuracy and Classification Report to validate results.
  - **Matplotlib/Seaborn:** Used for plotting training curves and visualizing the detected plates.

## 4. Milestones

### 1. Define a problem

Manual logging of vehicle license plates for traffic management, parking toll collection, and security access control is labor-intensive, error-prone, and slow. The problem is to automate this process using computer vision to read license plates from static images in real-time with high accuracy, regardless of standard variations in plate fonts or car colors.

### 2. Understanding the business problem

- **Efficiency:** Automated systems can process vehicles significantly faster than human operators.
- **Security:** Enables instant cross-referencing with databases for stolen vehicles or unauthorized entry.
- **Cost Reduction:** Reduces the need for manual staffing at checkpoints and toll booths.
- **Scalability:** A software-based solution can be deployed on existing CCTV infrastructure with minimal hardware upgrades.

### 3. Get the Data

The dataset was acquired programmatically via the Kaggle API.

- **Command:** `!kaggle datasets download -d sarthakvajpayee/ai-indian-license-plate-recognition-data`
- **Content:** The data is structured into folders containing images of character classes (0-9, A-Z), organized into training and testing sets.

### 4. Explore and pre-process data

Before feeding data into the model, several preprocessing steps were applied to ensure quality features:

- **Resizing:** Images were resized to standard dimensions to match the CNN input layer requirements.
- **Grayscale Conversion:** Convert images from RGB to Grayscale to reduce computational complexity (`cv2.cvtColor`).

- **Thresholding:** Used Inverse Binary Thresholding and Otsu's method to convert the grayscale image into a strict black-and-white mask, making characters stand out (white text on black background).

## 5. Choosing the python platform

Google Colab was selected as the development platform.

- **Reasoning:** It provides free access to powerful GPUs , which significantly accelerates the training of the Convolutional Neural Network. It also offers a pre-configured environment with essential libraries like TensorFlow and OpenCV pre-installed.

## 6. Create Features (Character Segmentation)

Feature creation in ANPR involves isolating the characters so the model can classify them one by one.

- **Contour Detection:** Used `cv2.findContours` to locate white blobs on the binary image.
- **Geometric Filtering:** Filtered contours based on Aspect Ratio and Area. License plate characters generally have a specific height-to-width ratio.
- **Sorting:** Segmented characters were sorted from left-to-right based on their X-coordinate to maintain the correct order of the license number.

## 7. Create Model

A Convolutional Neural Network (CNN) was designed for character classification.

- **Architecture:** Sequential model.
  - **Input Layer :** Accepts 28x28 grayscale images.
  - **Conv2D Layers :** Filters to detect low-level features like vertical lines and curves.
  - **MaxPooling2D :** Reduces spatial dimensions to focus on the most important features.
  - **Flatten :** Converts the 2D matrix into a 1D vector.
  - **Dense (Hidden) Layers :** Fully connected layers with ReLU activation for non-linearity.
  - **Dropout :** Applied to prevent overfitting.
  - **Output Layer :** Dense layer with Softmax activation, outputting probabilities for 36 classes (0-9, A-Z).
- **Optimizer:** Adam (Adaptive Moment Estimation) for efficient gradient descent.
- **Loss Function:** SparseCategoricalCrossentropy for multi-class classification.

## 8. Model Evaluation

The model was evaluated on the unseen test dataset.

- **Metrics Used:** Accuracy and Classification report was evaluated.
- **Visual Validation:** The pipeline was tested on sample car images where the bounding box around the plate and the predicted text string were displayed for qualitative assessment.

## 5. Conclusion & Future Scope

**Conclusion:** This project successfully demonstrates a working prototype of an ANPR system. By combining traditional Computer Vision (Haar Cascades for efficient localization) with modern Deep Learning (CNNs for

robust classification), we achieved a balanced system that is both fast and accurate. The preprocessing pipeline proved to be the most vital component; without clean segmentation, even the best CNN fails. The use of Data Augmentation significantly improved the model's ability to handle slightly rotated or zoomed characters.

#### **Future Scope:**

1. **YOLO Integration:** Moving from Haar Cascades to YOLO (You Only Look Once) would improve plate detection in complex, cluttered scenes or night-time conditions.
2. **Optical Character Verification (OCV):** Implementing syntax checking (e.g., knowing that Indian plates follow a standard format like "KA 05...") could automatically correct "O" vs "0" errors.
3. **Edge Deployment:** Optimizing the model using TensorFlow Lite to run on edge devices like Raspberry Pi for real-time, on-road applications.



#### **4. Individual Details:**

**Name:** Anup Shende

**Email id:** shende.anup25@gmail.com