

COMPUTER VISION AND IMAGE ANALYSIS PROJECT REPORT

Title: License Plate Recognition

COMP – 5435 FA

Name: Saarthak Mahajan – 0892203

Group 4

Instructor: Dr. Shan Du



I.

INTRODUCTION

The concept of License plate recognition evolves from the world of machine learning. Machine learning is the ability of machines to take decisions based on the experiences or history of outputs. Formally speaking, Machine learning (ML) is the study of algorithms ^[1] and mathematical models that computer systems use to progressively improve their performance on a specific task. Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task.

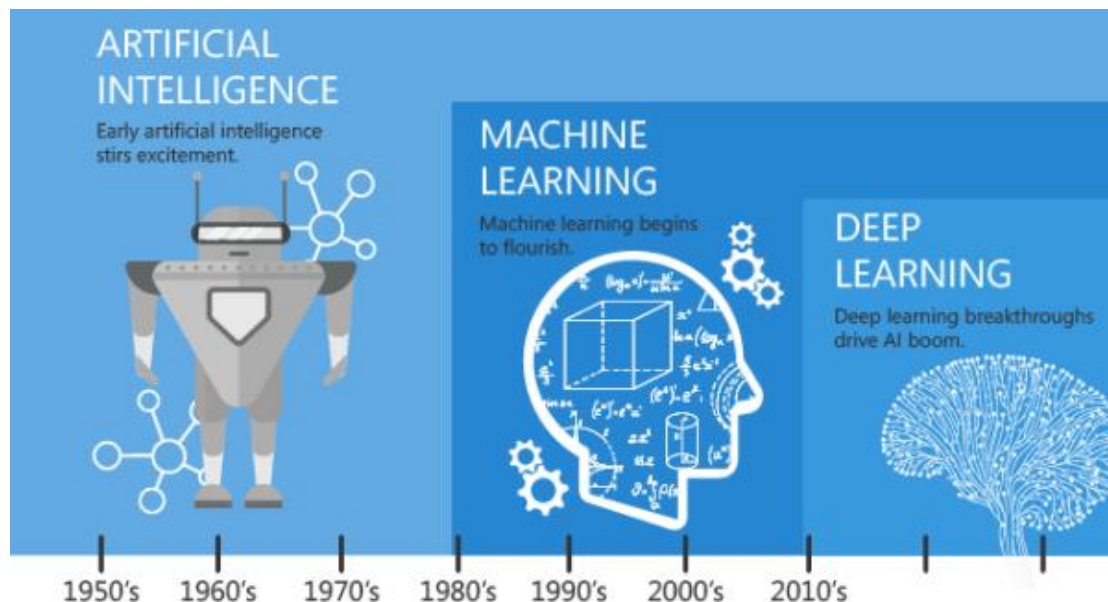


Figure: History of ML, AI, DL

Machine learning is a subset of Artificial Intelligence (AI). AI is the quest of humans to mimic human behaviour in machines. A relatively new subset of ML, Deep Learning further enhances the concept in the form of Neural Networks and enables machines to learn based on existing as well as new data.

Computer Vision is a field that is in constant growth and has the same fundamental goal of Machine Learning, to automate tasks that the human visual system can do. Computer vision tasks^[2] include methods for acquiring, processing, analysing and understanding digital images,

and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions.

When we further go into various other sub-fields of Computer Vision, we encounter applications such as Pattern Recognition, Face Detection, Video Tracking, Image Restoration etc.

Pattern Recognition consists of several applications such as Character Recognition. We will be focusing on License Plate Recognition (LPR). LPR is the ability of machines or cameras to detect the license plate of a vehicle by using the concepts of Optical character recognition, character segmentation. There is a huge need and value for all the research that is taking place in the field of Computer Vision. A decade ago, computer vision was not as hot as today. With each year, the venture capital funding in Computer Vision is tripling. This growth in the field is supported by advances in Computer Hardware, emergence of Deep Learning and increase in number of datasets. Our goal of imitating Human Vision is very far away. Human Vision is a very complex process and we need to understand how it works. Machine Learning and Artificial Intelligence makes sure that the Computer Vision application is implemented with proper optimizations in its frameworks.

ALGORITHMS

The software requires seven primary algorithms for identifying a license plate:

1. **Plate localization** – responsible for finding and isolating the plate on the picture.
2. **Plate orientation and sizing** – compensates for the skew of the plate and adjusts the dimensions to the required size.
3. **Normalization** – adjusts the brightness and contrast of the image.
4. **Character segmentation** – finds the individual characters on the plates.
5. **Optical character recognition.**
6. **Syntactical/Geometrical analysis** – check characters and positions against country-specific rules.
7. The averaging of the recognised value over multiple fields/images to produce a more reliable or confident result. Especially since any single image may contain a reflected light flare, be partially obscured or other temporary effect.



Figure 1.2: Segmented Characters

A 2008 article in Parking Trend International discussed a disparity in claimed vs. experienced license plate recognition read rates, with manufacturers claiming that their recognition engines can correctly report 98% of the time, although customers experience only 90% to 94% success, even with new equipment under perfect conditions. Early systems were reportedly only 60% to 80% reliable.

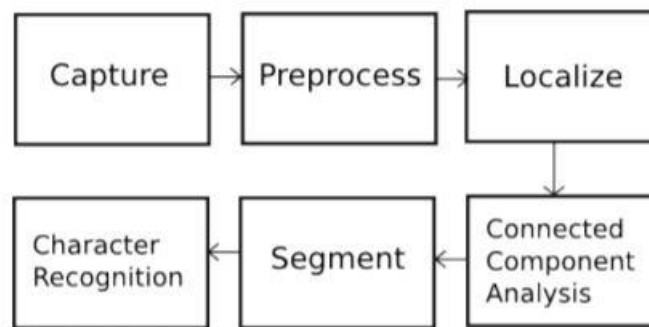


Figure 1.3: Proposed System

An LPR system is based on the image above, which captures an image, pre-processes it to improve the quality of the image, localizes the license plate and then performs character segmentation as well as character recognition on the segmented characters. The implementation can be done in different ways in terms of the language, platform used to run that LPR software. The performance of these systems might also differ depending on various factors. This performance is measured in various ways. One of the ways is to measure its accuracy and see how many characters the system recognized successfully.

We also need to understand how the system can be improved by using better algorithms that are more accurate than existing ones. These algorithms, based on supervised as well as unsupervised learning are under constant improvement which means that new algorithms come up everyday which can be incorporated as well to implement the concept of LPR. We will also learning some insight questions that we need to address to further enhance understanding of our project.

II.

OBJECTIVES

The main objective of our project was to develop an application successfully able to detect and recognize License plate of vehicles. The scope of our test vehicles was supposed to be just for cars and trucks but the system could also be used for bikes, trailers or any other vehicle with valid License plate. Therefore, the objectives were –

- To develop a C++ application capable of performing LPR on North American vehicles.
- To develop a LPR system that trains and tests on a data set and uses that trained data set to perform character recognition on a license plate
- Use an appropriate algorithm that gives back feasible results.
- Measure the system's performance by testing the application on more than 50 images.
- Suggest improvements that could be made to the system in future scope.

To achieve all these objectives, we had to gather all the information and resources required to implement this application.

REQUIREMENTS

We need a platform to run our application on. Specifically, we need to have a computer system with enough RAM to run our application. We also need at least 4GB of storage to accommodate various software.

- OpenCV Library (3.4.3)
- Microsoft Visual Studio 2017 (IDE)
- Working knowledge of C++ programming language
- At least 50 North American vehicle images preferably in high quality
- C++ libraries installed in Ms Visual Studio
- Microsoft Windows 10 Platform OS
- To implement a working Machine Learning algorithm – in our case (KNN)

We need to make sure we have the above resources to make our system work smoothly in our computer. Before we move on to how we achieved our application, we need to discuss about these resources and why we use them.

OPENCV (Open Source Computer Vision Library v3.4.3)



OpenCV (*Open Source Computer Vision*) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage and is now maintained by Itseez. The library is cross-platform and free for use under the open-source BSD license.

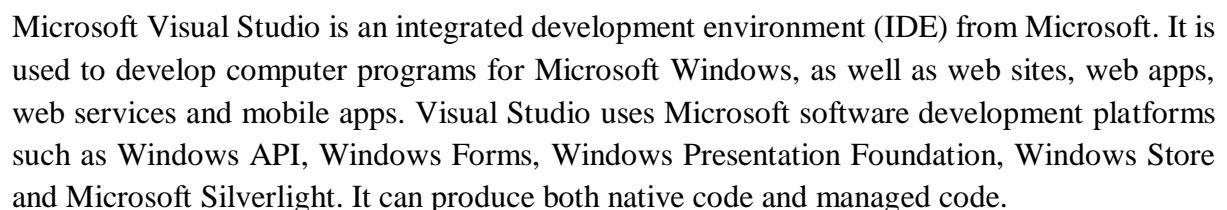
Officially launched in 1999, the OpenCV project was initially an Intel Research initiative to advance CPU-intensive applications, part of a series of projects including real-time ray tracing and 3D display walls. The main contributors to the project included a number of optimization experts in Intel Russia, as well as Intel's Performance Library Team. In the early days of OpenCV, the goals of the project were described as:

To support some of the above areas, OpenCV includes a statistical machine-learning library that contains:

- Boosting
- Decision tree learning
- Gradient boosting trees
- Expectation-maximization algorithm
- k-nearest neighbour algorithm
- Naive Bayes classifier
- Artificial neural networks
- Random forest
- Support vector machine (SVM)
- Deep neural networks (DNN)

Why OpenCV?

- ## Microsoft Visual Studio 2017



Visual Studio includes a code editor supporting IntelliSense (the code completion component) as well as code refactoring. The integrated debugger works as both a source-level debugger and a machine-level debugger. Other built-in tools include a code profiler, forms designer for building GUI applications, web designer, class designer, and database schema designer. It accepts plug-ins that enhance the functionality at almost every level—including adding support for source control systems (like Subversion) and adding new toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle (like the Team Foundation Server client: Team Explorer).

Visual Studio supports 36 different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C, C++ and C++/CLI (via Visual C++), VB.NET (via Visual Basic .NET), C# (via Visual C#), F# (as of Visual Studio 2010) and TypeScript (as of Visual Studio 2013 Update 2). Support for other languages such as Python, Ruby, Node.js, and M among others is available via language services installed separately. It also supports XML/XSLT, HTML/XHTML, JavaScript and CSS. Java (and J#) were supported in the past.

Like any other IDE, it includes a code editor that supports syntax highlighting and code completion using IntelliSense for variables, functions, methods, loops and LINQ queries. IntelliSense is supported for the included languages, as well as for XML and for Cascading Style Sheets and JavaScript when developing web sites and web applications. Autocomplete suggestions appear in a modeless list box over the code editor window, in proximity of the editing cursor. In Visual Studio 2008 onwards, it can be made temporarily semi-transparent to see the code obstructed by it. The code editor is used for all supported languages.

SELECTION OF APPROPRIATE ALGORITHM

The selection of an appropriate algorithm is directly linked to the performance of our LPR application. There are many machine learning algorithms out there -

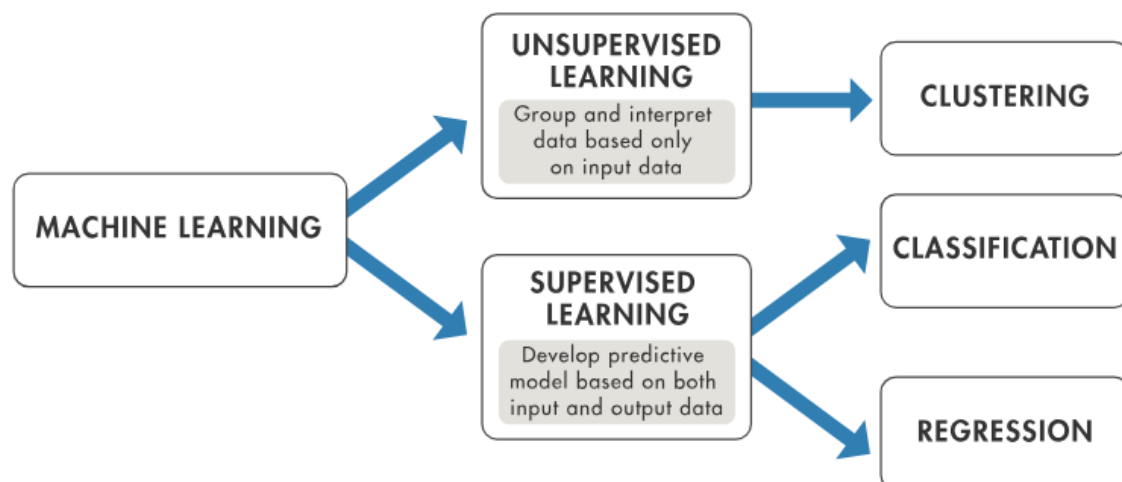


Figure: – Classification of ML algorithms

Supervised learning algorithms are trained using labeled examples, such as an input where the desired output is known. For example, a piece of equipment could have data points labeled either “F” (failed) or “R” (runs). The learning algorithm receives a set of inputs along with the corresponding correct outputs, and the algorithm learns by comparing its actual output with correct outputs to find errors. It then modifies the model accordingly. Through methods like classification, regression, prediction and gradient boosting, supervised learning uses patterns to predict the values of the label on additional unlabeled data. Supervised learning is commonly used in applications where historical data predicts likely future events. For example, it can anticipate when credit card transactions are likely to be fraudulent or which insurance customer is likely to file a claim.

Unsupervised learning is used against data that has no historical labels. The system is not told the "right answer." The algorithm must figure out what is being shown. The goal is to explore the data and find some structure within. Unsupervised learning works well on transactional data. For example, it can identify segments of customers with similar attributes who can then be treated similarly in marketing campaigns. Or it can find the main attributes that separate customer segments from each other. Popular techniques include self-organizing maps, nearest-neighbor mapping, k-means clustering and singular value decomposition. These algorithms are also used to segment text topics, recommend items and identify data outliers.

Semisupervised learning is used for the same applications as supervised learning. But it uses both labeled and unlabeled data for training – typically a small amount of labeled data with a large amount of unlabeled data (because unlabeled data is less expensive and takes less effort to acquire). This type of learning can be used with methods such as classification, regression and prediction. Semisupervised learning is useful when the cost associated with labeling is too high to allow for a fully labeled training process. Early examples of this include identifying a person's face on a web cam.

Reinforcement learning is often used for robotics, gaming and navigation. With reinforcement learning, the algorithm discovers through trial and error which actions yield the greatest rewards. This type of learning has three primary components: the agent (the learner or decision maker), the environment (everything the agent interacts with) and actions (what the agent can do). The objective is for the agent to choose actions that maximize the expected reward over a given amount of time. The agent will reach the goal much faster by following a good policy. So, the goal in reinforcement learning is to learn the best policy.

After looking at all the above type of ML algorithms, we reach to a conclusion that we shall use Supervised Learning for our application –

Now among supervised learning, there are various categories to choose from. The basic idea is train on already existing data sets that enable us to use those trained data sets in our application on testing such as SVM, KNN, and Decision Trees etc. In our project, we have chosen KNN or k-nearest neighbours as the algorithm to train a dataset. The reason we choose KNN will be discussed further.

KNN ALGORITHM (K-nearest neighbours)

In pattern recognition, the k-nearest neighbours algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

In **k-NN classification**, the output is a class membership. An object is classified by a majority vote of its neighbours, with the object being assigned to the class most common among its k nearest neighbours (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbour.

In **k-NN regression**, the output is the property value for the object. This value is the average of the values of its k nearest neighbours.

K-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.

Both for classification and regression, it can be useful to assign weight to the contributions of the neighbours, so that the nearer neighbours contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbour a weight of $1/d$, where d is the distance to the neighbour.

The neighbours are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

A peculiarity of the k-NN algorithm is that it is sensitive to the local structure of the data. The algorithm is not to be confused with k-means, another popular machine learning technique.

Algorithm

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples. In the classification phase, k is a user-defined constant, and an unlabelled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point.

A commonly used distance metric for continuous variables is Euclidean distance. For discrete variables, such as for text classification, another metric can be used, such as the overlap metric (or Hamming distance). In the context of gene expression microarray data, for example, k-NN has also been employed with correlation coefficients such as Pearson and Spearman. Often, the classification accuracy of k-NN can be improved significantly if the distance metric is learned with specialized algorithms such as Large Margin Nearest Neighbour or Neighbourhood components analysis.

A drawback of the basic "majority voting" classification occurs when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the k nearest neighbours due to their large number. One way to overcome this problem is to weight the classification, taking into account the distance from the test point to each of its k nearest neighbours. The class (or value, in regression problems) of each of the k nearest points is multiplied by a weight proportional to the inverse of the distance from that point to the test point. Another way to overcome skew is by abstraction in data representation. For example, in a self-organizing map (SOM), each node is a representative (a centre) of a cluster of similar points, regardless of their density in the original training data. K-NN can then be applied to the SOM.

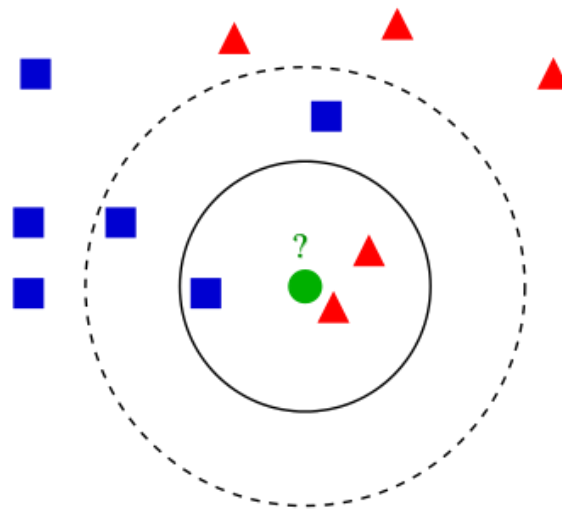


Figure: KNN example

The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If $k = 3$ (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If $k = 5$ (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

Why we used KNN? The answer to this question will be covered in the last section of ***INSIGHT QUESTIONS***

To conclude, our main objective is to use the KNN algorithm to train on an existing dataset. The trained dataset resulted from that training will be used in our application of LPR and we will test many images for recognizing the license plates of North American vehicles. The performance of our system will be directly linked to the successful recognition of license plates. We will be using KNN libraries already available in the OpenCV libraries. We will integrating various components of a LPE in one single repository making them work together perfectly.

III.

METHODOLOGY

As we discussed before, the basic idea between any LPR system is the same as follows –

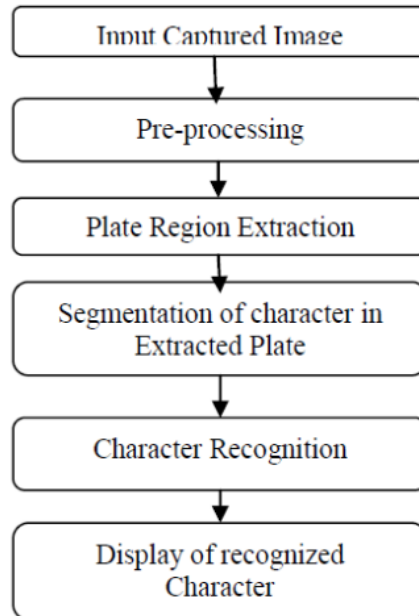


Figure: Flow Chart for LPR

Since we need to go flow wise, the whole program goes in the following order –

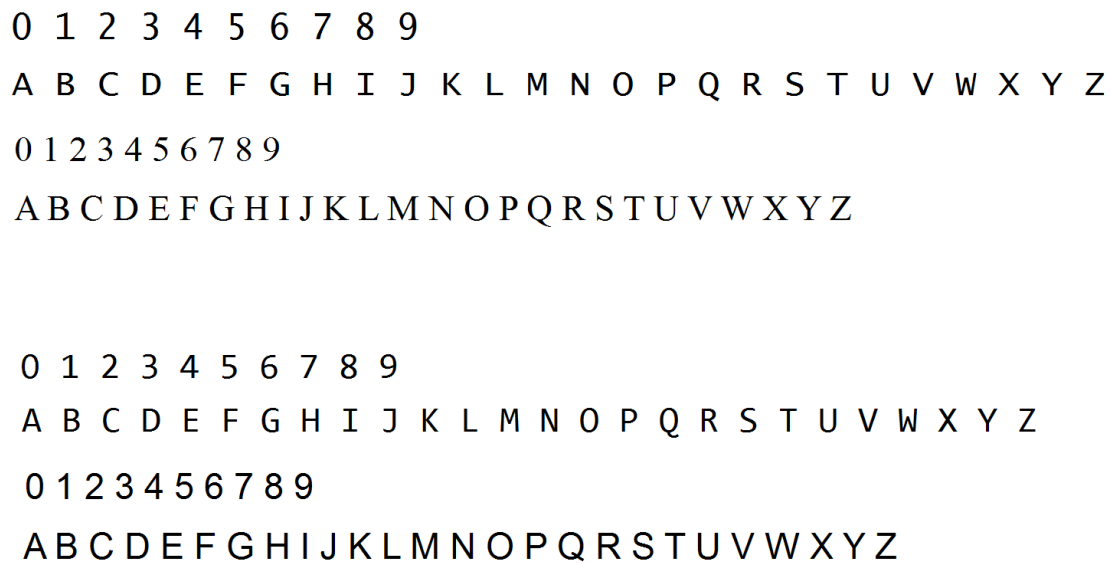
1. We use K-nearest neighbours (KNN algorithm) to train the North American font which is usually a variant of *gothic* font. This training results in two xml files – classifications.xml & images.xml
2. The trained data sets exist in our xml files above which will be used in our main program for LPR.
3. As we discussed above, there are main two jobs that the program has to do with the test image that we input from the computer –

Plate Detection – Contour matching

Character Recognition – KNN algorithm

4. The output of the above program will be the required characters of the license plate.

Step 1 - To start we will be training our KNN classifier on a sample image consisting of various fonts of North American vehicles.



0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Figure: North American Font characters image (test1.png)

We run a program that creates two xml files, which contained our trained data sets – *classifications.xml* & *images.xml*. The main part of the code of this program is included below:

```
cv::Mat imgTrainingNumbers;  
cv::Mat imgGrayscale;  
cv::Mat imgBlurred;  
cv::Mat imgThresh;  
cv::Mat imgThreshCopy;  
  
std::vector<std::vector<cv::Point> > ptContours;  
std::vector<cv::Vec4i> v4iHierarchy;  
  
cv::Mat matClassificationInts;  
  
cv::Mat matTrainingImagesAsFlattenedFloats;  
  
std::vector<int> intValidChars = { '0', '1', '2', '3', '4', '5', '6', '7', '8',  
'9',  
    'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',  
    'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',  
    'U', 'V', 'W', 'X', 'Y', 'Z' };  
  
imgTrainingNumbers = cv::imread("test1.png");  
  
if (imgTrainingNumbers.empty()) {  
    std::cout << "error: image not read from file\n\n";  
    return(0);  
}  
  
cv::cvtColor(imgTrainingNumbers, imgGrayscale, CV_BGR2GRAY);
```

```

cv::GaussianBlur(imgGrayscale,
    imgBlurred,
    cv::Size(5, 5),
    0);

cv::adaptiveThreshold(imgBlurred,
    imgThresh,
    255,
    cv::ADAPTIVE_THRESH_GAUSSIAN_C,
    cv::THRESH_BINARY_INV,
    11,
    2);

cv::imshow("imgThresh", imgThresh);
imgThreshCopy = imgThresh.clone();

cv::findContours(imgThreshCopy,
    ptContours,
    v4iHierarchy,
    cv::RETR_EXTERNAL,
    cv::CHAIN_APPROX_SIMPLE);

for (int i = 0; i < ptContours.size(); i++) {
    if (cv::contourArea(ptContours[i]) > MIN_CONTOUR_AREA) {
        cv::Rect boundingRect = cv::boundingRect(ptContours[i]);

        cv::rectangle(imgTrainingNumbers, boundingRect, cv::Scalar(0, 0,
255), 2);

        cv::Mat matROI = imgThresh(boundingRect);

        cv::Mat matROIResized;
        cv::resize(matROI, matROIResized, cv::Size(RESIZED_IMAGE_WIDTH,
RESIZED_IMAGE_HEIGHT));

        cv::imshow("matROI", matROI);
        cv::imshow("matROIResized", matROIResized);
        cv::imshow("imgTrainingNumbers", imgTrainingNumbers);

        int intChar = cv::waitKey(0);

        if (intChar == 27) {
            return(0);
        }
        else if (std::find(intValidChars.begin(), intValidChars.end(),
intChar) != intValidChars.end()) {

            matClassificationInts.push_back(intChar);

            cv::Mat matImageFloat;
            matROIResized.convertTo(matImageFloat, CV_32FC1);

            cv::Mat matImageFlattenedFloat = matImageFloat.reshape(1,
1);

            matTrainingImagesAsFlattenedFloats.push_back(matImageFlattenedFloat);

        }
    }
}

```

```

    }

    std::cout << "training complete\n\n";

    cv::FileStorage fsClassifications("classifications.xml",
cv::FileStorage::WRITE);

    if (fsClassifications.isOpened() == false) {
        std::cout << "error, unable to open training classifications file,
exiting program\n\n";
        return(0);
    }

    fsClassifications << "classifications" << matClassificationInts;
    fsClassifications.release();

    cv::FileStorage fsTrainingImages("images.xml", cv::FileStorage::WRITE);

    if (fsTrainingImages.isOpened() == false) {
        std::cout << "error, unable to open training images file, exiting
program\n\n";
        return(0);
    }

    fsTrainingImages << "images" << matTrainingImagesAsFlattenedFloats;
    fsTrainingImages.release();
    return(0);
}

```

Although the program is not the main one, this is important because we need a set of trained data that we will use in our main program. We get the trained set from the code above.

Step 2 — We will use these xml files in our main program that performs contour matching for finding the license plate of the vehicle and then it performs Character recognition using the xml files we generated above.

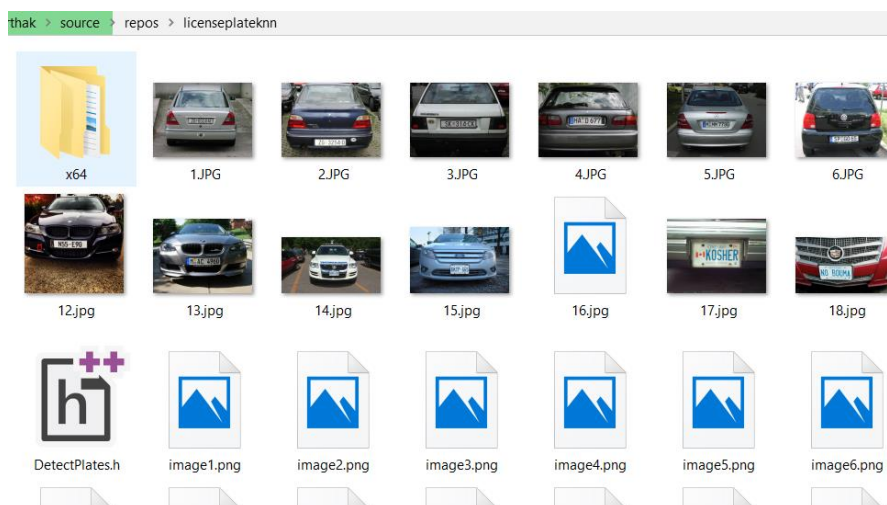


Figure: Input Images dataset that we can choose from

So now, we take the following input image to perform LPR on –

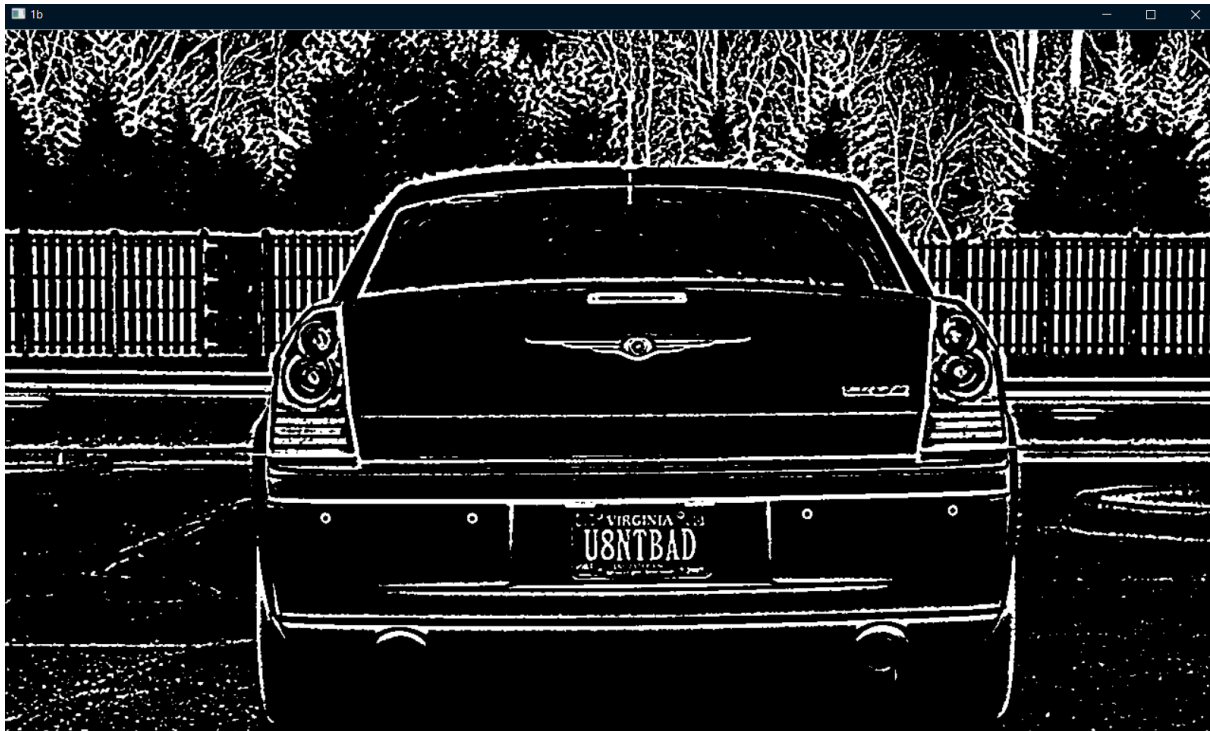


Figure – (image15.png) on which the LPR is run

The first step is to detect the rectangular plate from the image above and that is done using contour matching. To explain, contour is the shape of an enclosed figure. The program pre-processes the image from normal to grayscale to an image capable of recognizing various contours as follows –



Step 1: Convert to Grayscale



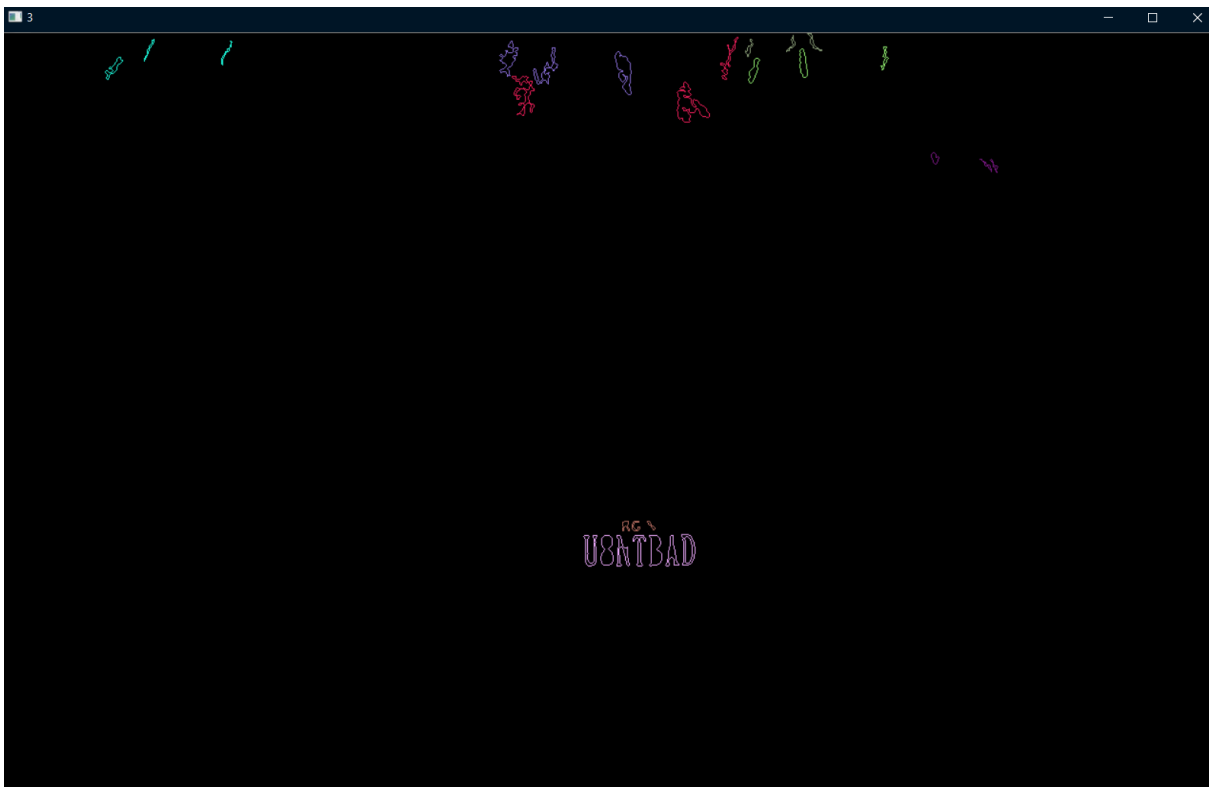
Step 2: Threshold an Image



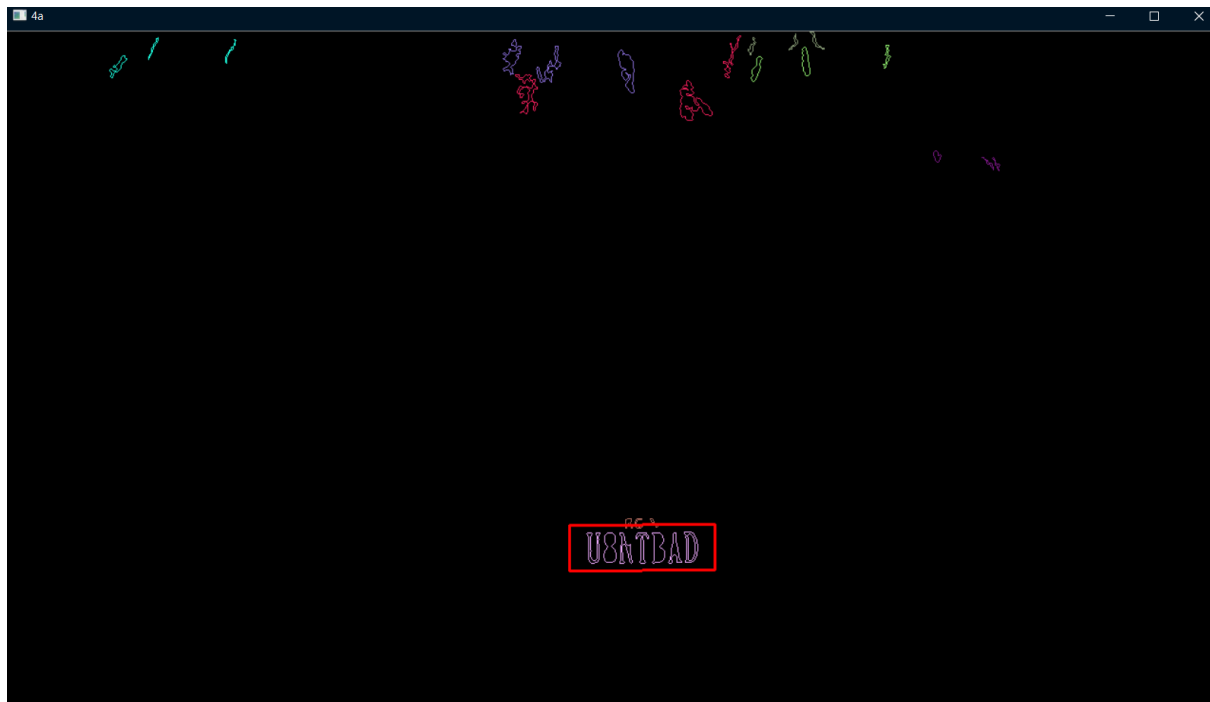
Step 3: Pre-process to find all contours from the image – 4173 contours found



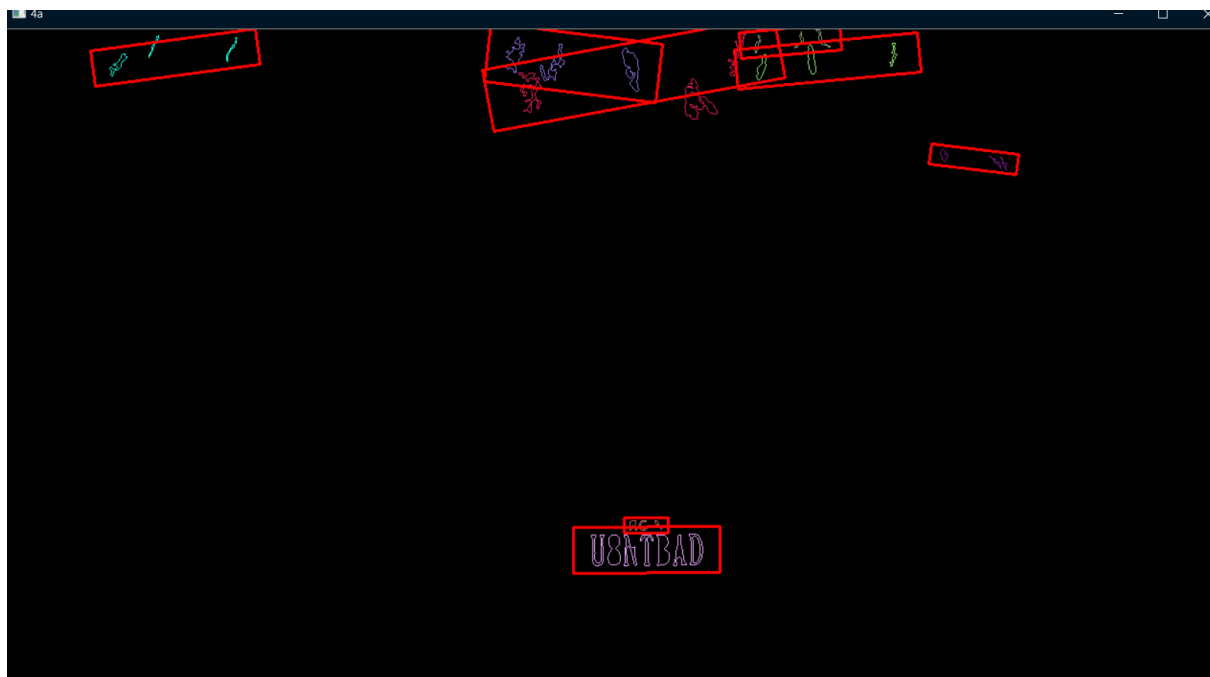
Step 4: All contours that can be characters (of specific size in height and width)



Step 5: Isolate groups of matching contours, here eight groups



Step 6: Draw a rectangle around each group of matching contours and perform character recognition on each one of them.



As we can see, we bound each group with a rectangle and perform character recognition one ach one of them.

```
C:\Users\Saarthak Mahajan\source\repos\licenseplateknn\x64\Debug\licenseplateknn.exe
possible plate 1, click on any image and press a key to continue . . .
possible plate 2, click on any image and press a key to continue . . .
possible plate 3, click on any image and press a key to continue . . .
possible plate 4, click on any image and press a key to continue . . .
possible plate 5, click on any image and press a key to continue . . .
possible plate 6, click on any image and press a key to continue . . .
possible plate 7, click on any image and press a key to continue . . .

late detection complete, click on any image and press a key to begin char recognition . . .

chars found in plate number 0 = U8NTBAD, click on any image and press a key to continue . . .
chars found in plate number 1 = RGN, click on any image and press a key to continue . . .
chars found in plate number 2 = (none), click on any image and press a key to continue . . .
chars found in plate number 3 = 7DZ, click on any image and press a key to continue . . .
chars found in plate number 4 = ZJ7, click on any image and press a key to continue . . .
chars found in plate number 5 = I7D7, click on any image and press a key to continue . . .
chars found in plate number 6 = 4ZY, click on any image and press a key to continue . . .
chars found in plate number 7 = 2QS, click on any image and press a key to continue . . .

char detection complete, click on any image and press a key to continue . . .

license plate read from image = U8NTBAD
-----
```

Figure: Output Console

As we can see from the above console, after character recognition using KNN, we get the various results for each group of our contours. We already know that one of those groups is our number plate with most number of characters. Therefore, we make an assumption that the required result will be the one with the most number of characters.

In the above case, our license plate has most number of characters- **8** out of all the eight groups of matching contours that might be characters.

So the final output will be as follows -



In the above output,

Required result – U8NTBAD

Obtained Result – U8NTBAD

Therefore, to summarize, in our main program we reduced the number of contours from 4000 to 123 to a group of eight matching contours or shape. Since the characters on a number plate are equally spaced and sized, there would eventually be a group with our license plate.

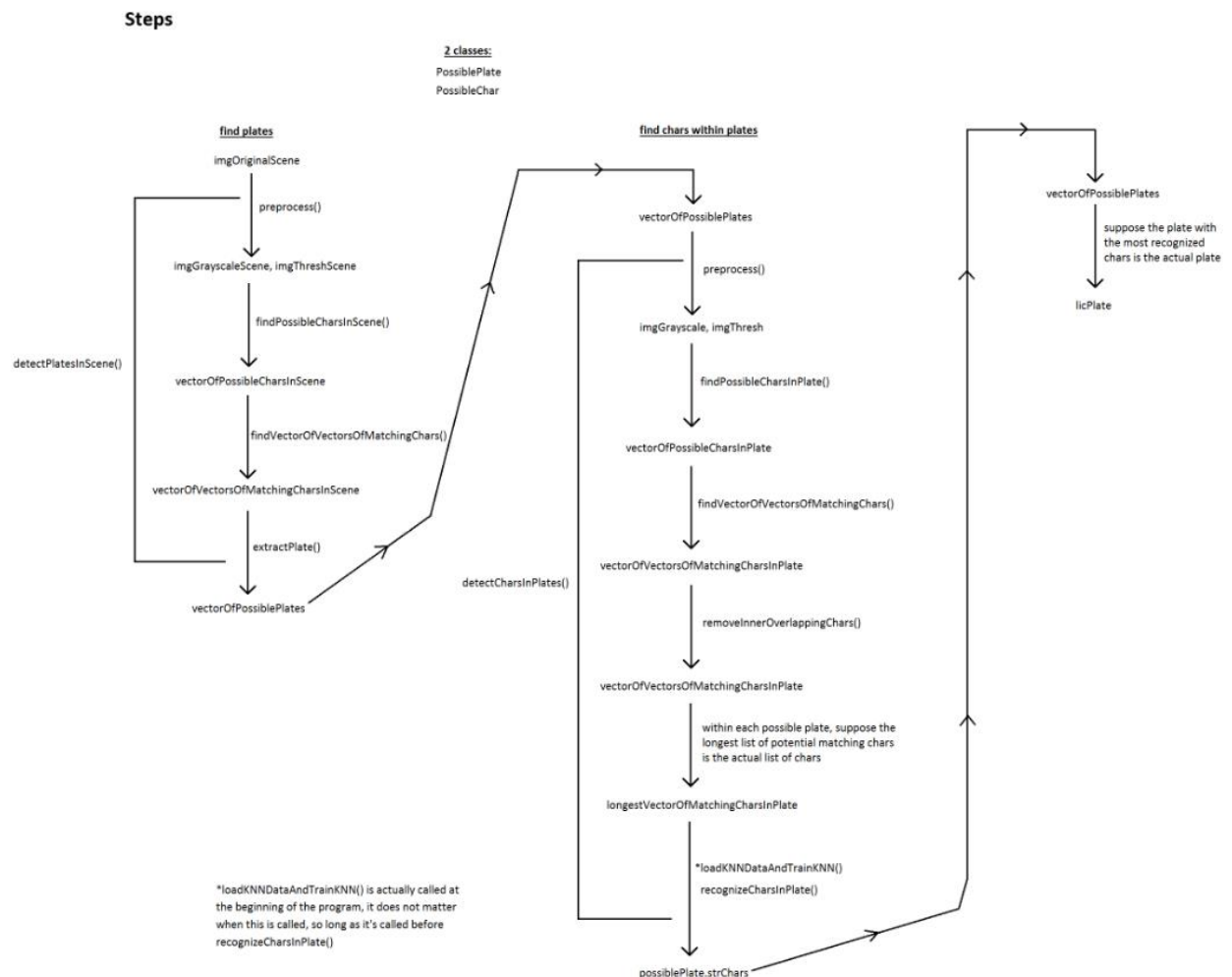


Figure: Stepwise Function calls in our main LPR program

More outputs-



imgOriginalScene



preprocess()



imgGrayscaleScene, imgThreshScene



findPossibleCharsInScene()

all contours

(2362 w/MCLRN F1 image)



vectorOfPossibleCharsInScene

(131 w/MCLRN F1 image)



findVectorOfVectorsOfMatchingChars()

vectorOfVectorsOfMatchingCharsInScene (13 w/MCLRNF1 image)



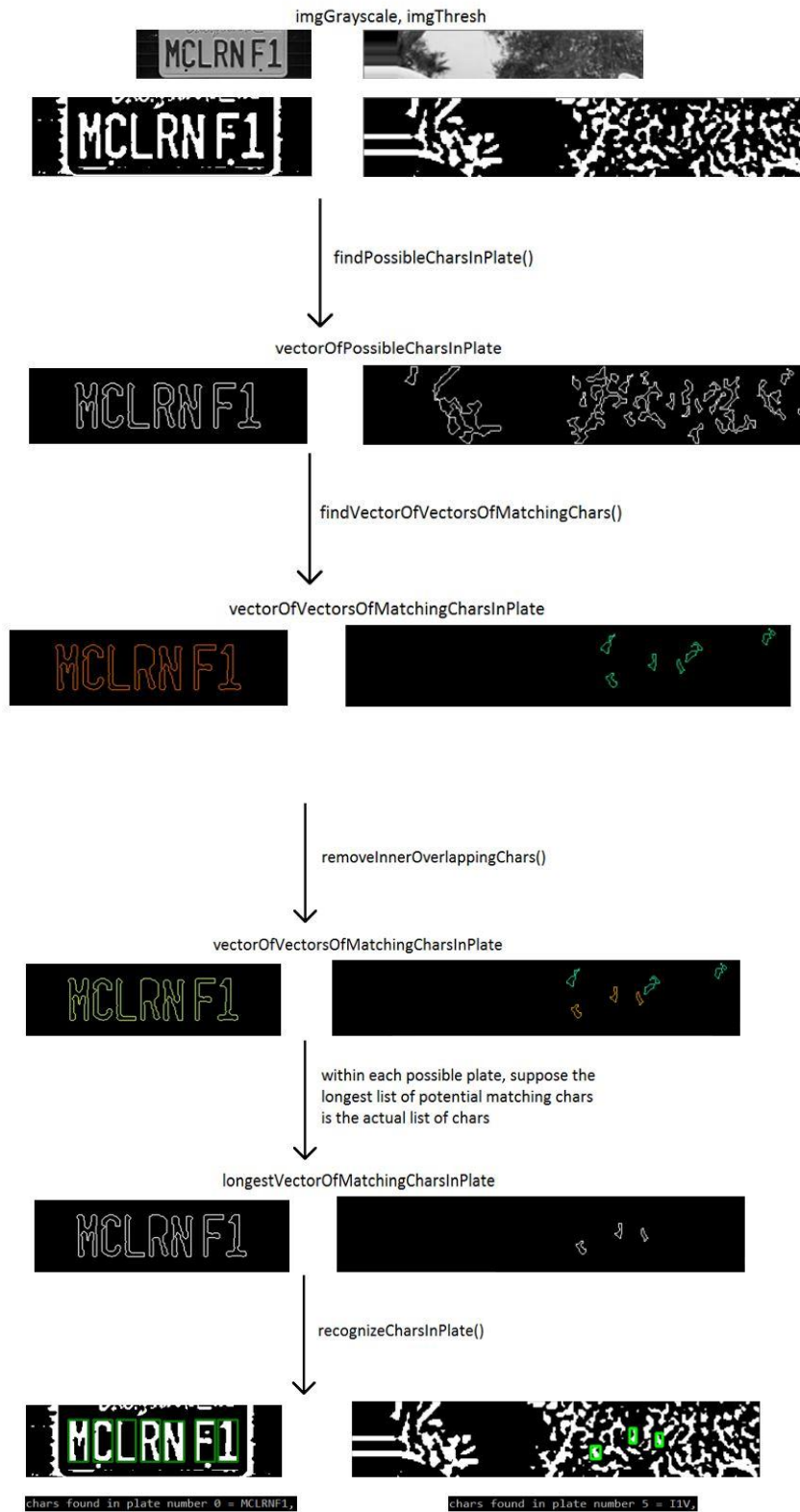
extractPlate()

vectorOfPossiblePlates (13 w/MCLRNF1 image)



preprocess()





possiblePlate.strChars

suppose the plate with
the most recognized
chars is the actual plate



C:\Users\Saarthak Mahajan\source\repos\LPR\x64\Debug\LPR.exe

13 possible plates found

license plate read from image = MCLRN F1



The code for the main program is below –

```
// Main.cpp

#include "Main.h"

int main(void) {

    bool blnKNNTrainingSuccessful = loadKNNDataAndTrainKNN();

    if (blnKNNTrainingSuccessful == false) {

        std::cout << std::endl << std::endl << "error: error: KNN traning was not
successful" << std::endl << std::endl;
        return(0);
    }

    cv::Mat imgOriginalScene;

    imgOriginalScene = cv::imread("image8.png");

    if (imgOriginalScene.empty()) {
        std::cout << "error: image not read from file\n\n";
        _getch();
        return(0);
    }

    std::vector<PossiblePlate> vectorOfPossiblePlates =
detectPlatesInScene(imgOriginalScene);
```

```

vectorOfPossiblePlates = detectCharsInPlates(vectorOfPossiblePlates);

cv::imshow("imgOriginalScene", imgOriginalScene);

if (vectorOfPossiblePlates.empty()) {
    std::cout << std::endl << "no license plates were detected" << std::endl;
}
else {

    std::sort(vectorOfPossiblePlates.begin(), vectorOfPossiblePlates.end(),
PossiblePlate::sortDescendingByNumberOfChars);

    PossiblePlate licPlate = vectorOfPossiblePlates.front();

    cv::imshow("imgPlate", licPlate.imgPlate);
    cv::imshow("imgThresh", licPlate.imgThresh);

    if (licPlate.strChars.length() == 0) {
        std::cout << std::endl << "no characters were detected" << std::endl <<
std::endl;
        return(0);
    }

    drawRedRectangleAroundPlate(imgOriginalScene, licPlate);

    std::cout << std::endl << "license plate read from image = " <<
licPlate.strChars << std::endl;
    std::cout << std::endl << "-----" <<
std::endl;

    writeLicensePlateCharsOnImage(imgOriginalScene, licPlate);

    cv::imshow("imgOriginalScene", imgOriginalScene);

    cv::imwrite("imgOriginalScene.png", imgOriginalScene);
}

cv::waitKey(0);

return(0);
}

void drawRedRectangleAroundPlate(cv::Mat &imgOriginalScene, PossiblePlate &licPlate) {
    cv::Point2f p2fRectPoints[4];

    licPlate.rrLocationOfPlateInScene.points(p2fRectPoints);

    for (int i = 0; i < 4; i++) {
        cv::line(imgOriginalScene, p2fRectPoints[i], p2fRectPoints[(i + 1) % 4],
SCALAR_RED, 2);
    }
}

void writeLicensePlateCharsOnImage(cv::Mat &imgOriginalScene, PossiblePlate &licPlate)
{
    cv::Point ptCenterOfTextArea;
    cv::Point ptLowerLeftTextOrigin;

    int intFontFace = CV_FONT_HERSHEY_SIMPLEX;

```

```

double dblFontScale = (double)licPlate.imgPlate.rows / 30.0;
int intFontThickness = (int)std::round(dblFontScale * 1.5);
int intBaseline = 0;

cv::Size textSize = cv::getTextSize(licPlate.strChars, intFontFace, dblFontScale,
intFontThickness, &intBaseline);

ptCenterOfTextArea.x = (int)licPlate.rrLocationOfPlateInScene.center.x;
if (licPlate.rrLocationOfPlateInScene.center.y < (imgOriginalScene.rows * 0.75)) {

    ptCenterOfTextArea.y =
(int)std::round(licPlate.rrLocationOfPlateInScene.center.y) +
(int)std::round((double)licPlate.imgPlate.rows * 1.6);
}
else {

    ptCenterOfTextArea.y =
(int)std::round(licPlate.rrLocationOfPlateInScene.center.y) -
(int)std::round((double)licPlate.imgPlate.rows * 1.6);
}

ptLowerLeftTextOrigin.x = (int)(ptCenterOfTextArea.x - (textSize.width / 2));
ptLowerLeftTextOrigin.y = (int)(ptCenterOfTextArea.y + (textSize.height / 2));

cv::putText(imgOriginalScene, licPlate.strChars, ptLowerLeftTextOrigin,
intFontFace, dblFontScale, SCALAR_RED, intFontThickness);
}

```

The above methodology is simple and it works on the basic principle of plate detection, character segmentation and character recognition.



IV.

EXPERIMENTAL RESULTS

We tested this system on 50 sample images. These sample images were all North American cars and some bikes with an average of 7-8 characters.

To measure the performance of our LPR system, we will define a variable Accuracy that is

$$\text{Accuracy} = (\text{Characters Successfully recognized} / \text{Total Characters to recognize}) * 100$$

As we saw in the above outputs, the LPR system did not successfully detect all the characters of a vehicle. For eg.



EXPECTED OUTPUT – NITESKY

ORIGINAL OUTPUT – NITESIY

$$\text{Accuracy} = (\text{Characters successfully recognized} / \text{Total Characters}) * 100$$

$$= 7 / 8 = 87.5\% \text{ for the above image}$$

Therefore, after testing 50 more images and noting down the successful recognition of each License plate we calculated the overall accuracy of our LPR system –

$$\text{Final Accuracy} = \frac{\text{Characters successfully recognized}}{\text{Total characters to recognize}} * 100$$

$$= (384 / 409) * 100$$

$$= 93.88\%$$

Therefore, the accuracy of the system obtained after testing 50 sample images is **93.88%**

V.

CONCLUSIONS

- After running the program and system on 50 sample images, we can conclude that our LPR system is a highly accurate system. It is able to successfully recognize 384 characters out of 409 characters.
- The system runs with very low latency in terms of computational speed and response.
- The system although highly accurate cannot match its performance to a production grade LPR system out there with almost 100% accuracy.
- There are limitations when it comes to the quality of input image being used, it has to be of good pixel quality in order to get very good results.
- The choice of algorithm KNN is efficient in our case, however more algorithms can be looked into to compare accuracy of other algorithm based systems.
- This LPR system can only be used for North American vehicles since we trained only on North American font characters.
- We can use this system in various applications such as to detect stolen vehicles and it can be incorporated in various Machine learning systems.

VI.

INSIGHT QUESTIONS

Question1: If you used some existing algorithms from open source, explain why these algorithms? Why not others?

Answer 1: We used K-nearest neighbours because it is a very effective supervised learning algorithm. Some more features are as follows –

- Robust to trained data filled with noise.
- Effective when training data size is large.
- Easy to implement
- Low latency and high computational speed
- Lowest time complexity
- Extremely flexible classification scheme.

Question 2: Describe the procedure of your method in detail and show some results STEP by STEP.

Answer 2: We already explained how the system of LPR works in detail with screenshots of outputs in the **methodology** section

Question 3: How can the accuracy of the existing system be improved?

Answer 3: We can improve the system's accuracy by clicking pictures in high quality, perpendicular to the license plate. In addition, we can click pictures of the car by changing its background to get specific contour matching.

VII.

REFERENCES

[1] - https://en.wikipedia.org/wiki/Machine_learning

[2] - https://en.wikipedia.org/wiki/Computer_vision

[3] - H. Rajput, T. Som and S. Kar, "An Automated Vehicle License Plate Recognition System," in Computer, vol. 48, no. 8, pp. 56-61, Aug. 2015.

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7185290&isnumber=7185273>

[4] H. Seibel, S. Goldenstein and A. Rocha, "Eyes on the Target: Super-Resolution and License-Plate Recognition in Low-Quality Surveillance Videos," in IEEE Access, vol. 5, pp. 20020-20035, 2017.

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8016340&isnumber=7859429>

[5] Amr Badr, Mohamed M. Abdelwahab, Ahmed M. Thabet, and Ahmed M. Abdelsadek, "Automatic Number Plate Recognition System", Annals of the University of Craiova, Mathematics and Computer Science Series, Volume 38(1), 2011
