COMPUTER VISION

Automatic License Plate Reading

Nicola Dainese

SCHEDULE

1. Project overview
2. Technical aspects
3. Common issues

Supplementary material

Project Overview

1. Preprocessing
2. Plate detection
3. Plate recognition

OVERVIEW - PREPROCESSING - 1/3

Original image

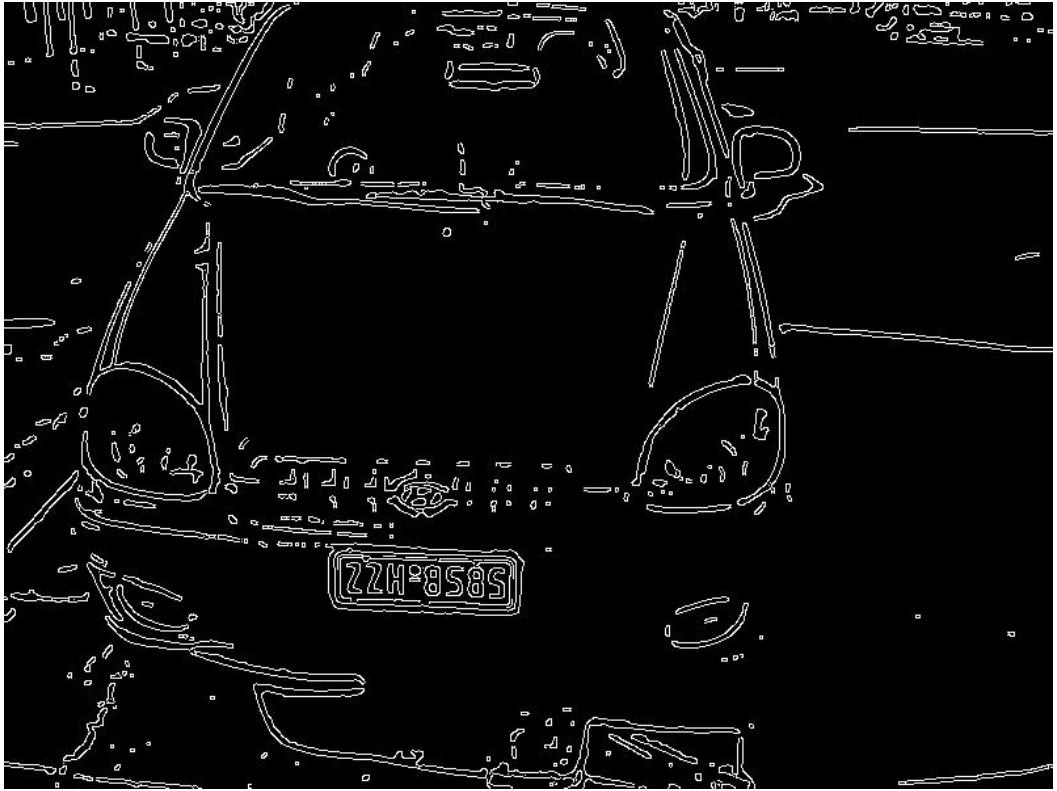
OVERVIEW - PREPROCESSING - 2/3

Grayscale +
Smoothing with
bilateral filter

OVERVIEW - PREPROCESSING - 3/3

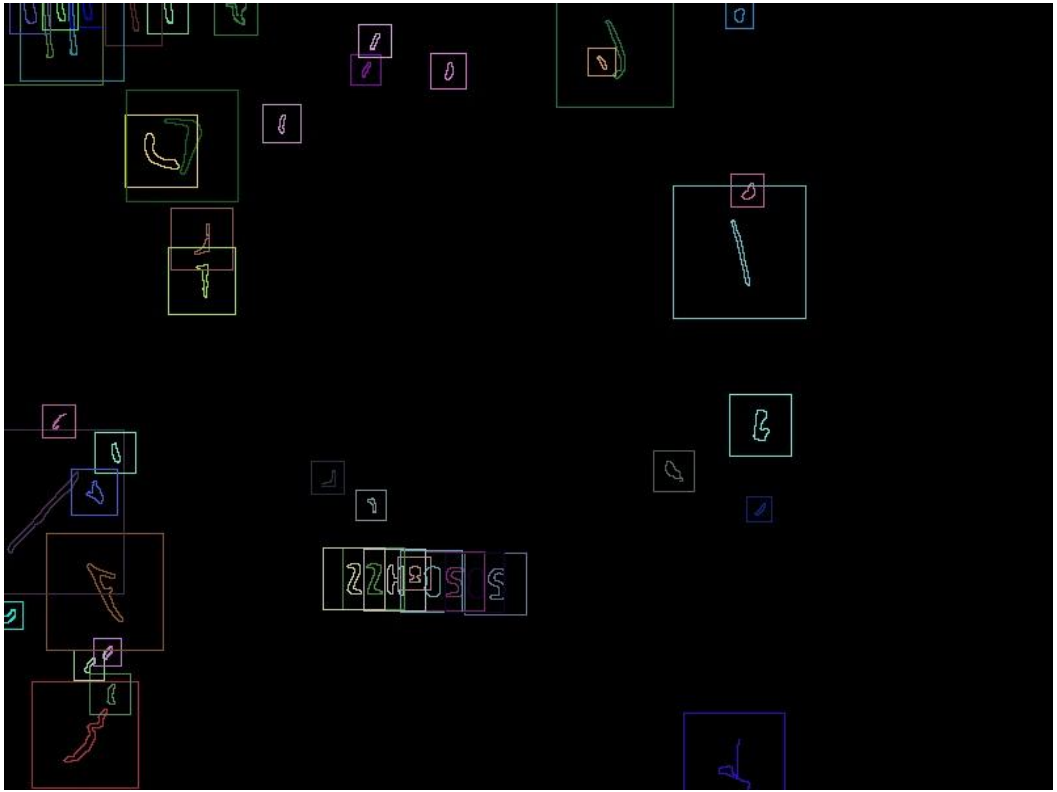
Binary image with adaptive thresholding

OVERVIEW - PLATE DETECTION - 1/5



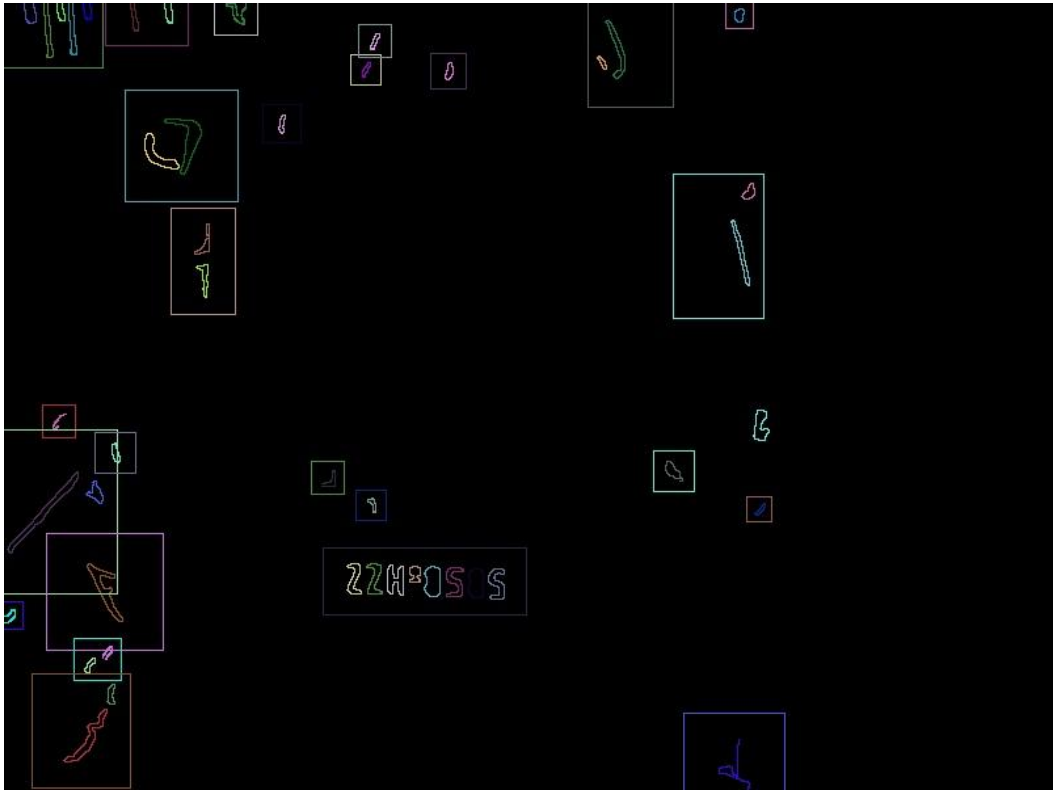
Canny edge detector

OVERVIEW - PLATE DETECTION - 2/5



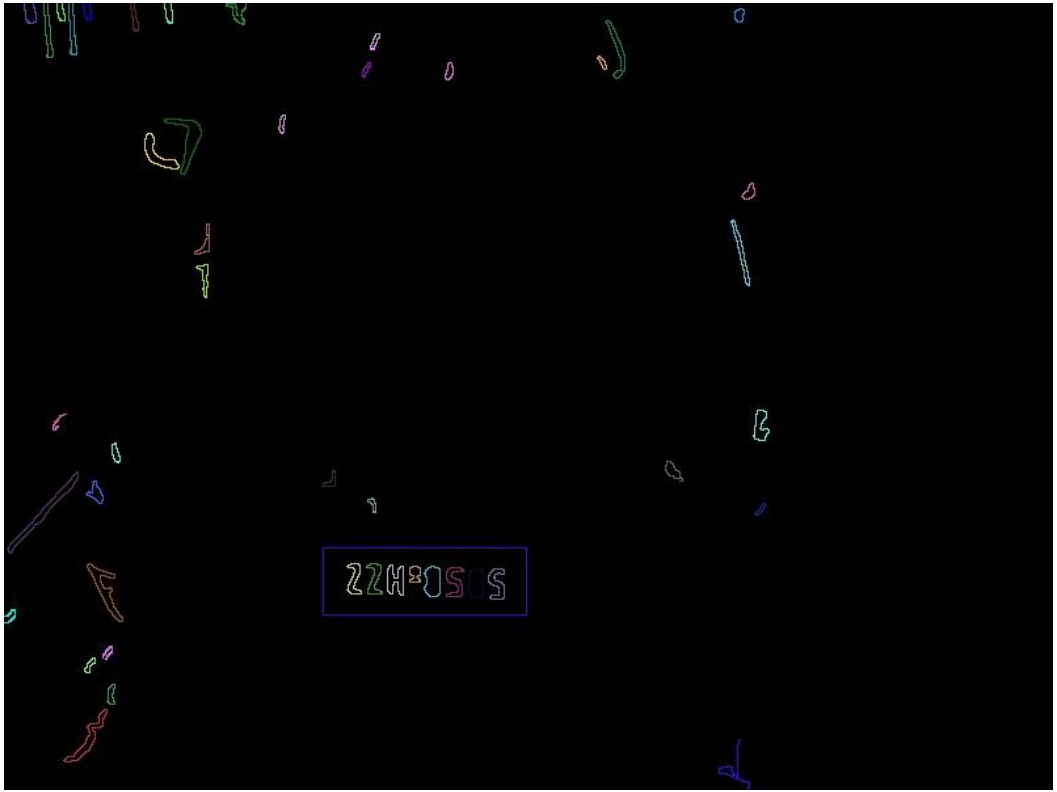
Contours detection +
PossibleChar filtering +
RectCluster initialization

OVERVIEW - PLATE DETECTION - 3/5



Merging adjacent RectClusters

OVERVIEW - PLATE DETECTION - 4/5



Require

1. Height/width ratio within 0.18 - 0.40
2. At least 6 characters inside the RectCluster

OVERVIEW - PLATE DETECTION - 5/5

Plate detection

OVERVIEW - PLATE RECOGNITION - 1/5

Canny edge detection

Contours detection

OVERVIEW - PLATE RECOGNITION - 2/5

Filter by

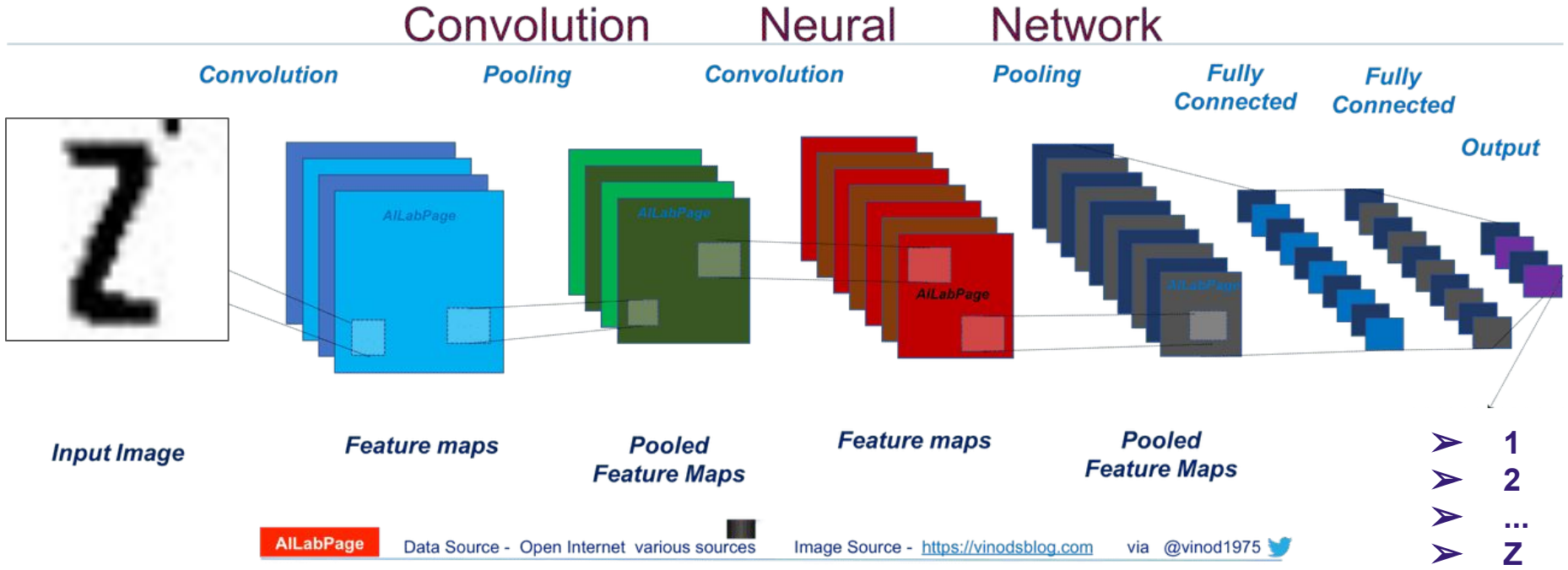
1. Width/height ratio in 0.15 - 1.00
2. Height variation from the average < 5px
3. Horizontal alignment within 2 standard deviations

OVERVIEW - PLATE RECOGNITION - 3/5

Characters preprocessing

1. Characters extraction
2. Adaptive threshold
3. Padding
4. Resize to 32 × 32

OVERVIEW - PLATE RECOGNITION - 4/5



PyTorch

OVERVIEW - PLATE RECOGNITION - 5/5

Plate reading

Technical Aspects

1. C++ code
2. Python code

TECHNICAL ASPECTS - C++ CODE - 1/9

```
#include "main.h"

// take as argument the filename with its path, e.g. /home/workdir/final_project/data/DSCN0408.jpg
int main(int argc, char* argv[]) {

	//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
	// Import image and preprocessing

	String filename = argv[1];
	cout << "Reading image " << filename << endl;
	Mat originalImg = imread(filename);

	cout << endl << "Preprocessing image... " << endl;
	Mat binaryImg = preprocessing(originalImg);

	//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
	// Plate detection - Canny edge detector + filter possible characters proportions

	cout << endl << "Plate detection... " << endl;
	Mat drawing = Mat::zeros(binaryImg.size(), CV_8UC3 );
	vector<PossibleChar> vecPossibleChar;
	vecPossibleChar = detectChar(binaryImg, drawing);

	//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
	// Plate detection - clustering characters + possible plates selection

	vector<Rect> selectedPlates;
	selectedPlates = selectPlates(vecPossibleChar, drawing);

	String title0 = "Original image with contoured plate";
	drawRects(originalImg, selectedPlates, title0, 3); // last variable is the thickness of the rectangle borders
```

preprocessing.h

PossibleChar.h

CharDetection.h

RectCluster.h + Clustering.h

graphics.h

TECHNICAL ASPECTS - C++ CODE - 2/9

```cpp
#ifndef PREPROCESSING_H
#define PREPROCESSING_H


#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>

#include "filter.h"
#include "graphics.h"

using namespace cv;
using namespace std;

const int ADAPTIVE_THRESH_BLOCK_SIZE = 19;
const int ADAPTIVE_THRESH_WEIGHT = 9;
const int S_SPACE = 80;
const  int S_RANGE = 80;
const int FILTER_SIZE = 7;


Mat preprocessing(Mat & originalImg);

#endif
```

```cpp
#include "preprocessing.h"

Mat preprocessing(Mat & originalImg)
{
        showIm(originalImg, "Original image");

        Mat greyImg;
        cvtColor(originalImg, greyImg, CV_BGR2GRAY);    //convert to greyscale
        //showIm(greyImg, "Greyscale image");

        BilateralFilter bf(greyImg, FILTER_SIZE, S_SPACE, S_RANGE);
        bf.doFilter();  //filter with bilateral filter to smooth without blurring contours
        Mat blurImg = bf.getResult();
        //showIm(blurImg, "Blurred image");
        imwrite("blurImg.jpg", blurImg);

        Mat binaryImg; //apply threshold to get binary image
        adaptiveThreshold(blurImg, binaryImg, 255.0, CV_ADAPTIVE_THRESH_GAUSSIAN_C, CV_THRESH_BINARY_INV,
                                                     ADAPTIVE_THRESH_BLOCK_SIZE, ADAPTIVE_THRESH_WEIGHT);
        //showIm(binaryImg, "Thresholded image");
        imwrite("binaryImg.jpg", binaryImg);

        return binaryImg;
}
```

graphics.h

filter.h

TECHNICAL ASPECTS - C++ CODE - 3/9

```
#ifndef POSSIBLE_CHAR_H
#define POSSIBLE_CHAR_H

#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>

class PossibleChar {
public:
        // member variables
        std::vector<cv::Point> contour;

        cv::Rect boundingRect;

        int intCenterX;
        int intCenterY;

        int area;

        double dblAspectRatio;

        PossibleChar(std::vector<cv::Point> _contour);

};

#endif // POSSIBLE_CHAR_H
```

```
#include "PossibleChar.h"

PossibleChar::PossibleChar(std::vector<cv::Point> _contour)
{
contour = _contour;

boundingRect = cv::boundingRect(contour);

intCenterX = (boundingRect.x + boundingRect.x + boundingRect.width) / 2;
intCenterY = (boundingRect.y + boundingRect.y + boundingRect.height) / 2;

area = boundingRect.width * boundingRect.height;

dblAspectRatio = (float)boundingRect.width / (float)boundingRect.height;
};
```

TECHNICAL ASPECTS - C++ CODE - 4/9

```cpp
#ifndef RECTCLUSTER_H
#define RECTCLUSTER_H

#include <opencv2/core.hpp>

const double MIN_RATIO = 0.18;
const double MAX_RATIO = 0.4;

class RectCluster {
public:
        int xHead;
        int yHead;
        double dHead;

        int xTail;
        int yTail;
        double dTail;

        RectCluster(cv::Rect headRect, cv::Rect tailRect);

        void setTail(int x, int y, double d);
        void setHead(int x, int y, double d);

        cv::Rect getBoundingRect();

        bool isPlate();
};

#endif
```

```cpp
#include "RectCluster.h"

RectCluster::RectCluster(cv::Rect headRect, cv::Rect tailRect)
{
        xHead = (int)( headRect.x + headRect.width/2);
        yHead = (int)(headRect.y + headRect.height/2);
        dHead = headRect.height;

        xTail = (int)(tailRect.x + tailRect.width/2);
        yTail = (int)(tailRect.y + tailRect.height/2);
        dTail = tailRect.height;
};
```

TECHNICAL ASPECTS - C++ CODE - 5/9

```
cv::Rect RectCluster::getBoundingRect()
{
        int width = (int)(xTail + dTail - xHead + dHead);
        int height = (int)(std::max(yHead+dHead,yTail+dTail) - std::min(yHead - dHead, yTail - dTail));
        int xTopLeft = (int)(xHead-dHead);
        int yTopLeft = (int)(std::min(yHead-dHead,yTail-dTail));
        cv::Rect contourRect(xTopLeft, yTopLeft, width, height);
        return contourRect;
};


bool RectCluster::isPlate()
{
        cv::Rect contourRect = getBoundingRect();
        double ratio = (double)contourRect.height / contourRect.width;
        if (MIN_RATIO < ratio && ratio < MAX_RATIO)
                {return true;}
        else
                {return false;};

};
```

TECHNICAL ASPECTS - C++ CODE - 6/9

```cpp
#ifndef CHAR_DETECTION_H
#define CHAR_DETECTION_H

#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>

#include "PossibleChar.h"
#include "graphics.h"

using namespace cv;
using namespace std;

// global constants ////////////////////////////////////////////////////////////////////////////
// constants for checkIfPossibleChar, this checks one possible char only (does not compare to another char)
const int MIN_PIXEL_WIDTH = 2;
const int MIN_PIXEL_HEIGHT = 8;
const int MAX_PIXEL_HEIGHT = 75; //all these measures are not invariant under changes of resolution/pixel size
const double MIN_ASPECT_RATIO = 0.15;
const double MAX_ASPECT_RATIO = 1.0;
const int MIN_PIXEL_AREA = 80;
const int HEIGHT_THRESH = 5;

// canny edge detector variables
const int THRESH1 = 100;
const int THRESH2 = 200;
const int CANNY_FILTER_SIZE = 5;

vector<PossibleChar> detectChar(Mat & binaryImg, Mat drawing);

vector<PossibleChar> filterByHeightVar( vector<PossibleChar> & originalVec);

vector<PossibleChar> filterByAllignment( vector<PossibleChar> & originalVec);

bool checkIfPossibleChar(PossibleChar &possibleChar);

bool doIntersect(PossibleChar & possChar1, PossibleChar & possChar2);

#endif
```

```cpp
#ifndef CLUSTERING_H
#define CLUSTERING_H


#include<opencv2/core/core.hpp>

#include "PossibleChar.h"
#include "RectCluster.h"
#include "graphics.h"

using namespace std;
using namespace cv;

const double ALPHA = 0.85; // sum of radii * alpha > distance of centers => merge clusters
const int MIN_COMPONENTS = 6; // min number of characters in a licence plate

std::vector<Rect> selectPlates(vector<PossibleChar> & vecPossibleChar, cv::Mat drawing);

std::vector<RectCluster> sortVecOfCluster(std::vector<RectCluster>& originalVec);

std::vector<PossibleChar> sortVecOfPossibleChar(std::vector<PossibleChar>& originalVec);

std::vector<RectCluster> mergeCluster(std::vector<RectCluster> VecOfCluster);

bool isInFrontOf(RectCluster& firstCluster, RectCluster& secondCluster);

double computeDistance(RectCluster& tailCluster, RectCluster& headCluster);

bool isCharInCluster(PossibleChar& possChar, RectCluster& cluster);

#endif
```

TECHNICAL ASPECTS - C++ CODE - 7/9

```
//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
// Plate recognition (if only 1 plate is recognized) - preprocessing + CNN prediction
if (selectedPlates.size()==1)
{
        cout << endl << "Plate recognition... " << endl;
        Rect roi(selectedPlates[0]);
        recognizePlate(originalImg, roi);
        string plateID = readResult(); // reading in the results of CNN prediction
        cout << "License Plate ID: " << plateID << endl;
        putText(originalImg, plateID, Point(roi.x ,roi.y ), FONT_HERSHEY_COMPLEX_SMALL, 1.5,Scalar(0,0,255), 1, CV_AA);
        showIm(originalImg, "Plate recognized");
        }
else
{
        return 0;
}

return 0;
}
```

plateRecognition.h (annotation pointing to `recognizePlate(originalImg, roi);`)

TECHNICAL ASPECTS - C++ CODE - 8/9

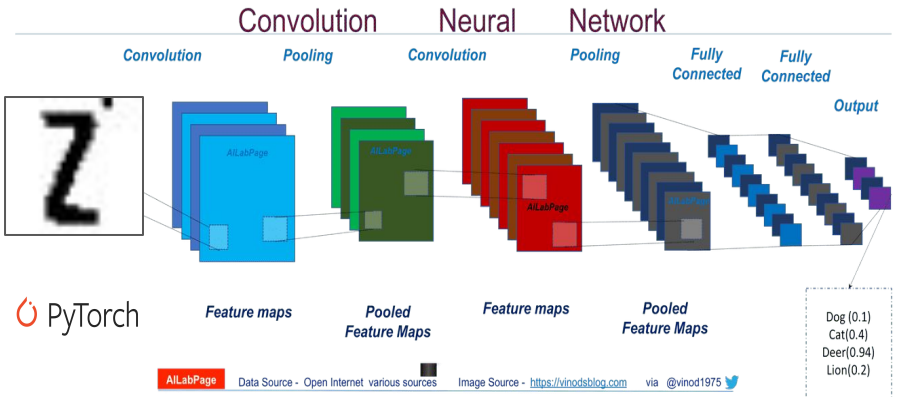
```
#ifndef PLATE_SEGMENTATION_H
#define PLATE_SEGMENTATION_H

#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>

#include "graphics.h"
#include "PossibleChar.h"
#include "CharDetection.h"
#include "Clustering.h"


using namespace std;
using namespace cv;

const int ADAPTIVE_THRESH_BLOCK_SIZE1 = 19;
const int ADAPTIVE_THRESH_WEIGHT1 = 10;

Mat preprocessPlate(Mat & plateImg);

vector<Mat> cropROIs(Mat & plateImg, vector<PossibleChar> & charsAlligned);

void recognizePlate(Mat & originalImg, Rect & roi);

Mat GetSquareImage( Mat& img);

Mat padImage(Mat & src);

#endif
```

TECHNICAL ASPECTS - C++ CODE - 9/9

```
for (size_t i = 0; i < charImgs.size(); i++)
{
        //showIm(charImgs[i],"Character resized");
        ostringstream name;
        name << "ImChar" << i << ".png";
        imwrite(name.str(), charImgs[i]);

        string command;
        command = "python -c 'from CNN_PredictChar import predict; predict(\""+ name.str()+ "\")' >> result.txt";
        cout << "Command executed: " << command << endl;
        const char *command2 = command.c_str();
        system(command2);
}
```

function predict in CNN_PredictChar.py

TECHNICAL ASPECTS - PYTHON CODE - 1/3




```python
def createLossAndOptimizer(net, learning_rate=0.001):

    #Loss function
    loss = torch.nn.CrossEntropyLoss()

    #Optimizer
    optimizer = optim.Adam(net.parameters(), lr=learning_rate)

    return(loss, optimizer)
```

```python
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 3x3 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, kernel_size = 3, stride = 1, padding = 1 )
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.conv2 = nn.Conv2d(6, 12, kernel_size = 3, stride = 1, padding = 1 )
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(12 * 8 * 8, 256)  # 8*8 from image dimension
        self.fc2 = nn.Linear(256, 64)
        self.fc3 = nn.Linear(64, 36)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)

        x = F.relu(self.conv2(x))
        x = self.pool2(x)

        x = x.view(-1, self.num_flat_features(x))

        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features
```

TECHNICAL ASPECTS - PYTHON CODE - 2/3

Training dataset: chars74K
http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/

Preprocessing and training in Jupyter Notebook

Weights of the trained net saved and ready to be reloaded

```
# Save model
torch.save(CNN.state_dict(), 'saved_model.pth')
```

```
import skimage
from os import walk

common_path = '0-9_and_A-Z/Sample'
images = []
labels = []

for n in range(1,37):
    f = []
    x = "%03.0f"%n
    mypath = '0-9_and_A-Z/Sample'+x
    for (dirpath, dirnames, filenames) in walk(mypath):
        f.extend(filenames)
        break
    print("Number of files in "+mypath+" : ", len(f), '\n')

    for i in range(len(f)):
        f[i] = mypath+'/'+f[i]

    template_imgs = skimage.io.imread_collection(f) # loads all the images

    rescaled_template_imgs = []
    #skimage.io.imshow(template_imgs[0])

    for i in range(len(template_imgs)):
        image_rescaled = skimage.transform.rescale(template_imgs[i], 1.0 / 4.0, anti_aliasing=False)
        rescaled_template_imgs.append(image_rescaled)

    rescaled_template_imgs = np.array(rescaled_template_imgs)
    filtered_imgs = rescaled_template_imgs[mask]
    hotEncodedLabel = np.array([i==n-1 for i in range(36)], dtype=int) # one hot encoding
    class_labels = np.full((len(filtered_imgs),len(hotEncodedLabel)), hotEncodedLabel)

    #skimage.io.imshow(rescaled_template_imgs[0])
    images.append(filtered_imgs)
    labels.append(class_labels)
    #print(template_imgs.shape)
```

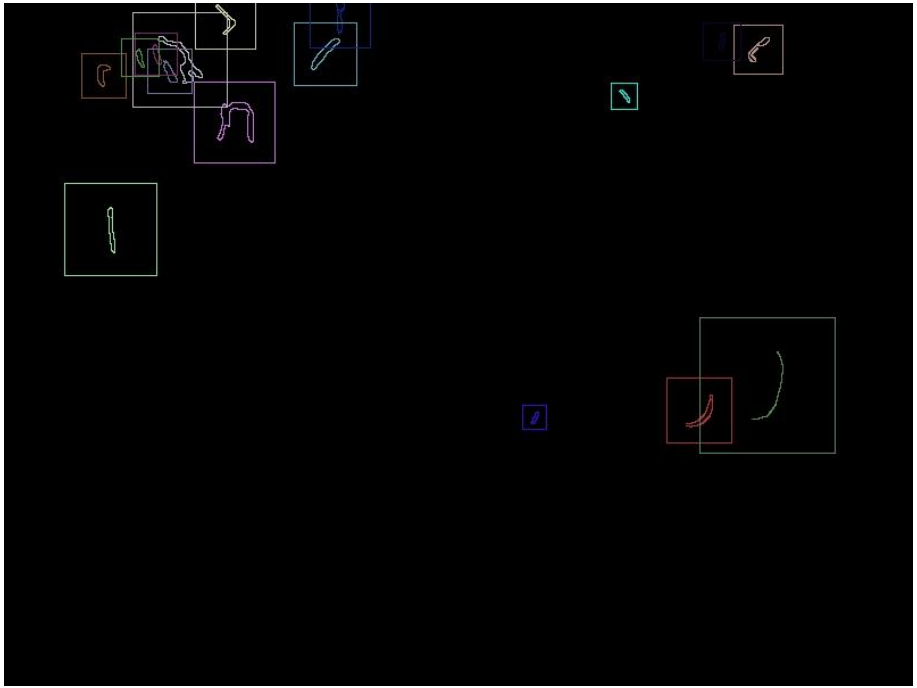
TECHNICAL ASPECTS - PYTHON CODE - 3/3

```python
import ConvolutionalNN as myCNN
import torch
import string
import skimage
import numpy as np

def predict(image_path):
    """
    Image_path can be local (if found in the same folder of the script) or complete
    Image must be of a number 0-9 or a capital letter A-Z.
    Only resolution accepted: 32x32.
    """

    CNN = myCNN.Net().double()
    state_dict = torch.load('saved_model.pth')
    CNN.load_state_dict(state_dict)

    #IncompatibleKeys(missing_keys=[], unexpected_keys=[]) means "no error"

    img = skimage.io.imread(image_path, as_gray=True)
    #skimage.io.imshow(img)
    img = np.array(img).reshape((1,1,32,32)) #vector of a single element with 1 channel of 32x32 pixels
    img = img/img.max() # in case it wasn't normalized
    img = torch.from_numpy(img)

    my_dict = define_dict()

    outputs = CNN(img)
    _, predicted = torch.max(outputs.data, 1)
    predicted = predicted.numpy()
    char = my_dict[predicted[0]]
    print(char)
```

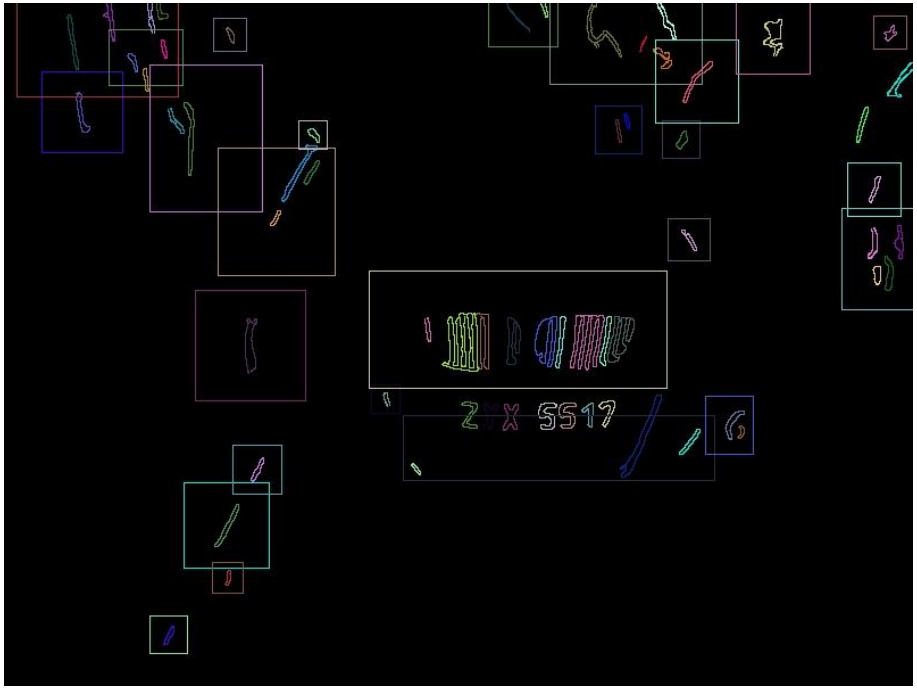
Common Issues

1. Plate detection
2. Characters detection
3. Characters recognition

COMMON ISSUES - PLATE DETECTION - 1/2



COMMON ISSUES - PLATE DETECTION - 2/2



COMMON ISSUES - CHAR. DETECTION

COMMON ISSUES - CHAR. RECOGNITION