# Разпознаване регистрационният номер на МПС по дадена снимка

# Декларация за липса на плагиатство:

- Тази курсова работа е моя работа, като всички изречения, илюстрации и програми от други хора са изрично цитирани.
- Тази курсова работа или нейна версия не са представени в друг университет или друга учебна институция.
- Разбирам, че ако се установи плагиатство в работата ми ще получа оценка “Слаб”.

## 1. Мотивация

Към днешна дата изкуственият интелект навлиза все повече в нашето ежедневие както и в много сфери включително и при извършване на контрол за спазване закона за движение по пътищата.

## 2. Задача

Представям задача за разпознаване на регистрационен номер на МПС по дадена снимка. Задачата демонстрира принципите на компютърното зрение, чието приложение в изкуственият интелект става се по-популярно. Подобна технология се използва не само при OCR (Optical character recognition), но и при магнитно-резонансовите снимки в медицината (при ЯМР).

Към момента са разработени много софтуери и готови библиотеки, които извършват character recognition, image segmentation, object detection. Компютърното зрение и изкуственият интелект имат широк спектър на приложения.

## 2.1. Основни проблеми

Основните проблеми, при дефинираната по-горе задача, са именно как компютърът (или някаква друга машина) като агент, разпознава обектите. Точно този проблем е в областта на компютърното зрение. Допускайки, че имам всички необходими възприятия (съответните технически спецификации и сензори), трябва да намерим начин по който дадено изображение да се представи по „разбираем“ за компютъра начин. Сегментацията на картината и отделянето на определени обекти не е лесна задача за машина, макар и да е тривиална за човек. Също така извършването на необходимата обработка на дадена картина изисква и съответния набор от софтуерни инструменти.

## 2.2. Методи за решаване на проблемите

Има различни методи които се използват при решаването на тази задача. Те са в три направления:

- Image Segmentation – разделянето на картина на множество сегменти.
- Object detection - след необходимата обработка и сегментация, по дадени спецификации намираме обекта.
- Optical character recognition – в конкретната задача, след извличане на обекта е време да разпознаем символите на него (буквите и цифрите на съответния предполагаем рег. номер на МПС).

## 2.3. Използвани средства

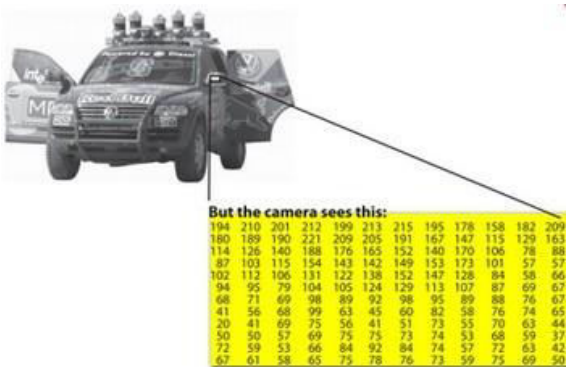
Откъм хардуерни спецификации не се изисква нищо особено, освен камера с прилично качество или вече направена снимка на дадено превозно средство.

Откъм софтуер съм използвал **Open source билбиотека за компютърно зрение и машинно самообучение – OpenCV 3.2**. Официално се поддържа в три програмни езика, аз използвам версията на C++.

# 3. Решение

Като за начало е всяка картина която човека вижда, се представя на едно дигитално устройство като множество от точки с цифрови стойности. В OpenCV картините представяме като матрици.

В началото картината се преобразува в Grayscale ( по подразбиране тя е в BGR, сиреч цветна). Тоест се представя като „черно-бяла“ картина в която всяка точка или пиксел е отенък на сивото. Отенъкът зависи от интензитета на точката. Интензитета на всяка точка се представя като цифрова стойност, обикновено в интервала [0; 255]. Така една картина е множество от такива цифрови стойности, и тези стойности се държат в матрица:




(илюстрация от документацията на OpenCV 3)

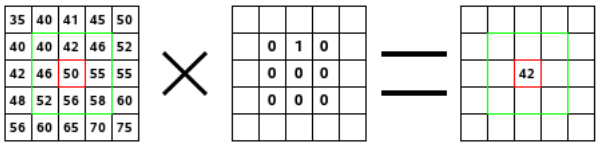
На първата стъпка взимаме най-осветената точка от оригиналната картината (от BGR формат). За целта я превръщаме в HSV формат (Hue Saturation Value), където Value ни дава търсеният интензитет. Тоест всяка точка е във вида HSV. Разделяме картината на 3 канала като взимаме само последния канал (този с интензитета). Така получаваме черно-бяла снимка във формата на GrayScale.

Следва подсилване на констраста (constrast enhancement). Това се извършва чрез последователни морфологични трансформации:

- Tophat – накратко, това ще открои всички елементи, които са по-ярки от заобикалящите ги обекти. Извършва се като се изпълняват последователно „ерозия“ и „разширяване“ (др. Морфологични операции) чрез елемент наречен „ структуриращ елемент“ и резултата се вади от входната картина.
- Blackhat – аналогично на горното само че това е разликата между резултата от последователно разширяване и ерозия, и входната картина.
- За получаване на картина с увеличен контраст взимаме входната черно-бяла картина и към нейната матрица добавяме тази на картината след tophat операцията. От матрицата на получената картина вадим тази на картината след blackhat операцията.

Следва да приложим Гаусово замъгляване (Gaussian blur), чрез прилагане на филтър. За тези цел ни трябва така нареченото ядро – матрица обикновено с размери 5x5. Извършваме така наречената „конволюция“ на матрицата на дадена точка от картината и тази на ядрото и после сумираме всички резултати за да получим матрицата на новата картина. Конволюцията на матриците всъшност става така :

- Взимаме пиксел от картината
- Взимаме околните пиксели ( в зависимост от това какъв размер сме избрали за ядрото). В случая използваме 5x5 тоест 24 съседни пкисела. Тази матрица я умножаваме с ядрото 5x5 на гаусовият филтър.
- Получаваме нова матрица
- Събираме всички такива матрици и получаваме матрицата на резултантната картина

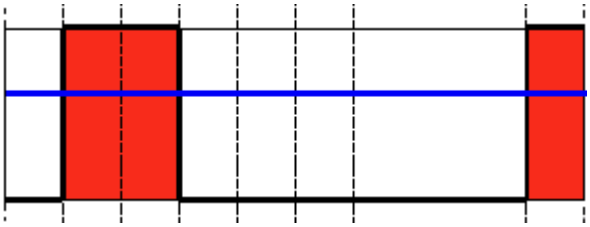



(пример за конволюция при ядро 3x3 от документацията на Gimp. Това не е гаусовото ядро, а само пример!)

Гаусовото замъгляване ще намали леко детайлността, но ще премахне излишното шумово замърсяване на картината.

Следващата най-важна стъпка се нарича **Thresholding** (идващо от праг, threshold). Той е основен при отделяне на региони и обекти от дадена картина. Задава се праг (threshold) и всички точки, чийто интезитет е над този праг, се оцветяват в определен цвят (обикновено бяло, за по-добро констрастиране), останалите се изключват. Разгледан като мат. Функция, нека **f(x,y)** е интензитета на точка (x,y), **t(x,y)** е интензитета на същата точка след thresholding операция, тогава:

t(x,y) = 255, ако f(x,y) > threshold ; в противен случай t(x,y) = 0 (за постигане на макс. контраст)



(илюстрация от документацията на OpenCV 3)

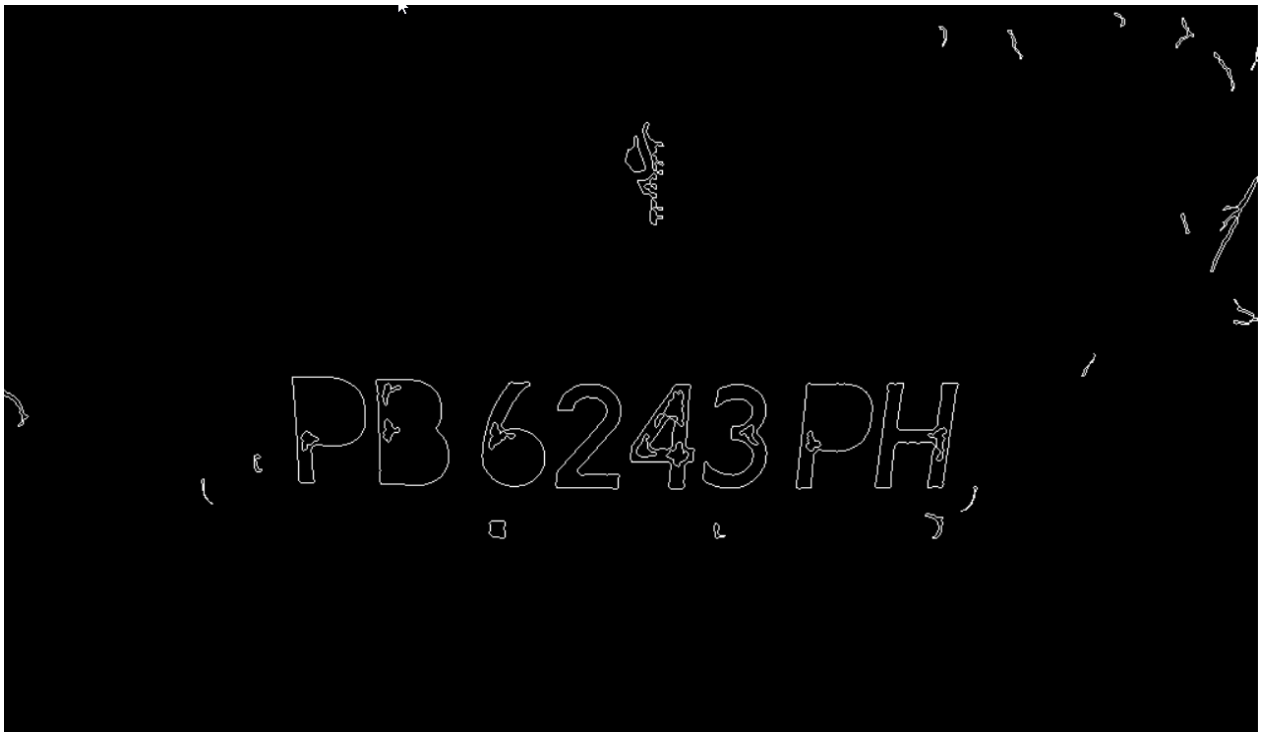
Съществуват няколко метода за thresholding, като аз използвам adaptive thresholding. Това е вид тресхолдинг при който горната граница (прага) се изменя динамично и може да се различава за различните точки. Прага се изчислява по метода **local thresholding**. Принципът при него е че отделните региони на картината имат сходно осветяване и съответно интензитет на точките и затова се изчислява средната стойност на прага за различните региони. Този метод е по-добър от стандартния thresholding!


Това резултата след началната обработка по горните стъпки. Сега вече сме изкарали снимка от на която контрастират и ясно се виждат ръбовете на различните обекти. Разбира се има до известна степен шумово замърсяване. Следващата стъпка е да намерим и нарисуваме контурите. Ето как изглежда намирането на всички контури по зададената обработена снимка:


На тази снимка са илюстрирани 2273 засечени контура. Необходим е подбор – от всички тези контури, нас ни интересуват тези, които за кандидати за символ. Получаваме съвкупност(вектор) от контури, като всеки контур е вектор от точки в пространството. Всеки евентуален символ (кандидат-символ) се задава чрез такъв контур. Рисувам обграждащ правоъгълник за всеки такъв контур. Проверяваме по съответните свойства за всеки такъв контур дали неговият обграждащ и принадлежащ правоъгълник (bounding rectangle) отговаря на изискванията за символ:

- Площ с реф. стойност > 80px
- Височина и ширина в рамките на стандартния за един символ интервал, а именно >2 и >8 px съответно за ширина и височина.
- Коефициента взет от отношението ширината към височината трябва да е в съответния интервал, а именно [0.25, 1.0]
- И други не толкова важни изисквания

Ако този bounding rectangle отговаря на тези реф. Стойности, то съответстващият му контур образува кандидат-символ (преполагаем символ). Целта е да вземе всички такива контури. Съставяме вектор от кандидат-символи (списък).

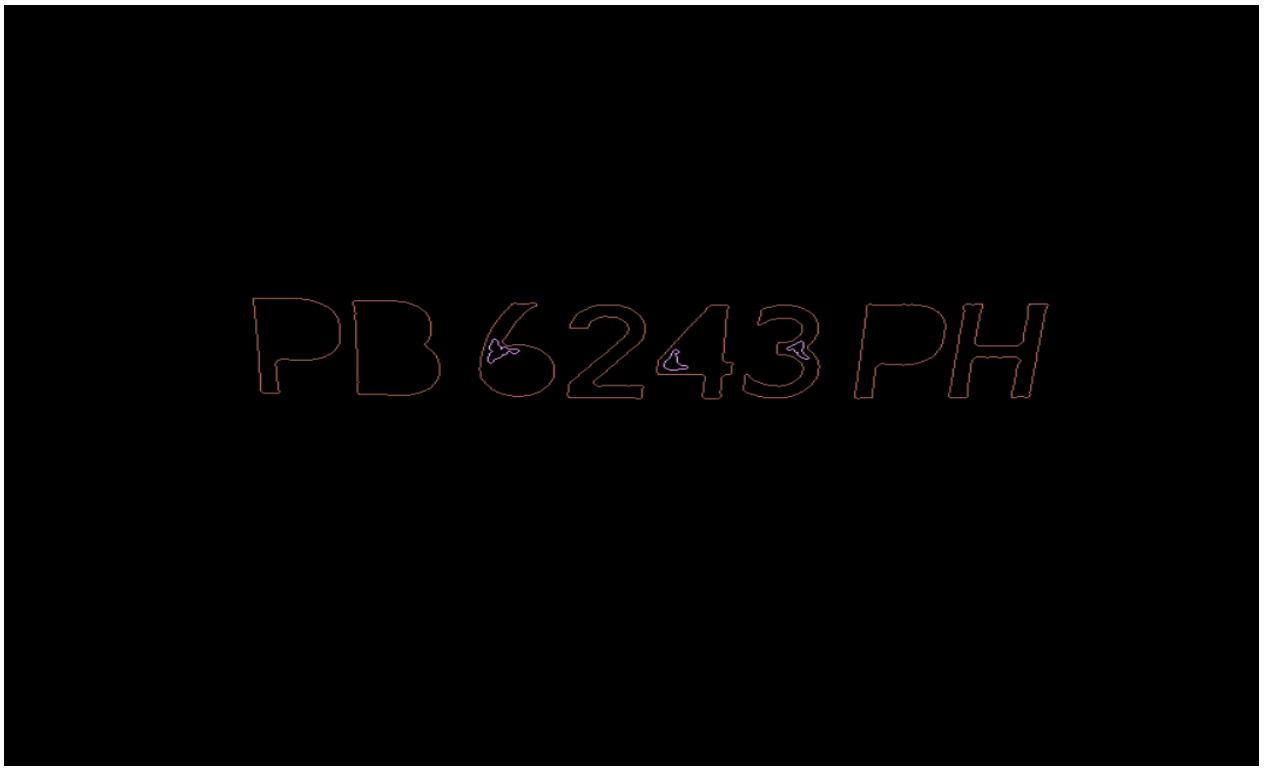



На тази илюстрация са показани кандидат-символите за нашата снимка. Това са контурите, които ни интересуват. Приблизителната бройка е 40 на снимката (малко е изрязана, съжалявам). Тоест от 2273 контура ние взимаме едва 40.

Следващата стъпка е да групираме тези символи. Тоест слагаме ги в групи като една от всички тези групи ще образува надписа на регистрационния номер. За целта ще илюстрираме различните групи кандидат-символи. Това последователно маркираме по един кандидат-символ и го сравняваме с всички останали. Гледаме съвкупност от характеристики, които искаме два кандидат символа да имат:

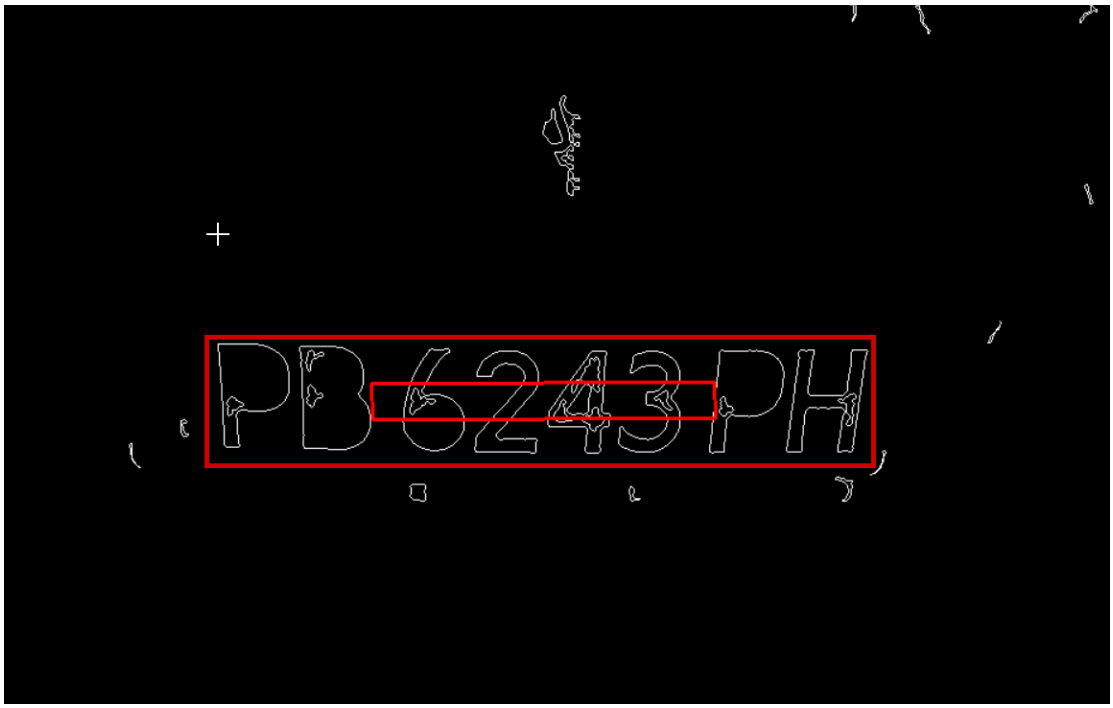
- Разстояние между тях (по обясними причини) – в рег. номер разстоянието между символите е малко
- Ъгъл между тях
- Разлика в площта – ясно е че два символа от рег. номер не могат да имат драстична разлика в площта, тоест тя трябва да е в някакъв интервал и ако е извън него значи нещо не е наред.
- Разлика в ширина
- Разлика във височина

Ако всички тези изисквания са удовлетворени то тези символи влизат в една група – един и същи вектор. Събираме всички такива вектори в един общ вектор от вектори.




В умален и изрязан вариант показвам зоната на интерес. Това е зоната около самият регистрационен номер. Забелязваме че символите от номера са в една група. Освен тях се наблюдава още една група кандидат-символи, които не са азбучни символи.

Следващата стъпка е да вземе всички кандидати за регистрационен номер. Това става с метода findAllPlates(<картина>) който ще анализира големият вектор от вектори от кандидат-символи. Тоест взимаме евентуалние номера които съответстват на тези вектори от кандидат-символи. Съответно от картината горе се вижда че ще имаме само две групи които са кандидати за рег. номер. На долната илюстрация ще видите очертаните кандидат-рег. номера илюстрирани върху черно-бялата снимка на кандидат-символите.




Извършваме проверка и обработка за всеки кандидат-рег. номер (в случая те са само 2). Извиличаме всеки от 2та рег. номера. Двата кандидата за рег. номера се извличат от оригиналната снимка и се обработват както в първата стъпка (вижте по-горе).


Ето как изглежда нашият извлечен кандидат-регистрационен номер от оригиналната снимка ( това е единия от 2та).

Така изглежда след необходимите преобразувания. Представен в този вид, ще го използваме за следващите стъпки на Character Recognition. Взимаме пак всички контури:

Може да се каже, че повтаряме донякъде горните стъпки. Намираме всички кандидат-символи. Отново групираме кандидат-символите  според изискванията и ги записваме в голям вектор от вектори. В случая това ще е вектор от единствен елемент – вектор от символите на рег. номер.

На горната илюстрация виждаме че символите от регистрационният номер са групирани заедно и са единствената такава група.  От другият кандидат-рег. номер също ще получим една или няколко такива групи. Ще премахнем всички излишни контури, които не могат да образуват рег. номер и въобще символ. За целта отново ще проверяваме изискванията.

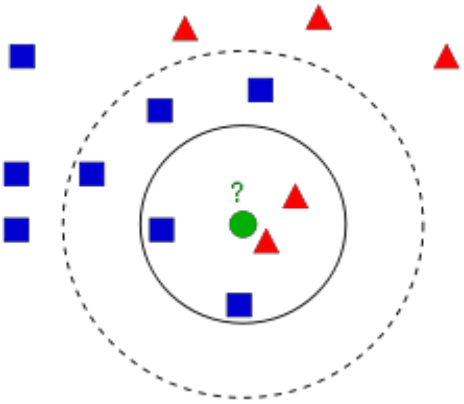
Това ще е финалният резултат. Подреждаме символите във вектора по оста Ox (x координатата). Ограждаме всеки символ с правоъгълник отговарящ на неговият bounding rectangle:

Самото рапознаване на буквите, а именно кой Unicode character описва даден контур, може да се извърши с готова библиотека наречена Tesseract. Тъй като съм разочарован от възможностите на Tesseract, използвах тези на OpenCV.

## 3.1. K – Nearest Neighbours (kNN)

За постигане на моите цели използвах алгоритъма kNN, който е имплементиран в OpenCV. Преди това накратко ще разкажа основните неща за алгоритъма.

Това е класифициращ алгоритъм за обучение. Класифицирането се извършва на базата на най-близките съседи до даден обект.




(илюстрация от документацията на OPENCV 3)

Гледайки картината по-горе виждаме сини квадрати и червени триъгълници. Съответно можем да кажем че елементите се класифицират в два класа – сини квадрати и червени триъгълници. Нека в средата да добавим един нов елемент – зелено кръгче. Как ще класифицираме зеленото кръгче, към кой от двата класа ще го присъединим?

Според kNN класификацията трябва да се извърши базирайки се на най-близките съседи. За k = 1, според алкогиртъма ще вземем най-близкият съсед на кръгчето и ще присъединим кръгчето към неговият клас. Това се нарича Nearest Neighbour algorithm. K може да има и други стойности, но за да избегнем конфликти, допускаме, че трябва да са винаги нечетни. Така за k = 3 ще вземем трите най-близки съседа. Ако два от тях са от черв. Триъгълници, а третият е от сините квадрати, тогава на мажоритарен принцип присъединяваме кръгчето към черв. триъгълници.

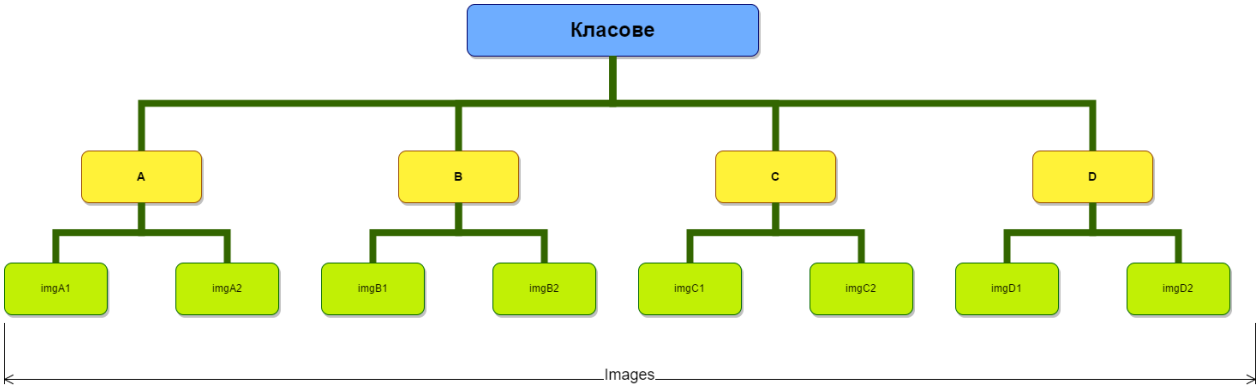
С какво ни помага това? Ами за да ни е от полза алгоритъма, а и въобще за да го изпозлваме са ни необходими данни – datasets. Този алгоритъм е известен при pattern recognition. В тези dataset-ове трябва да имаме класификации и картини, представени в съответната форма. Класификациите съдържат информация за това коя картина от набора картини в кой клас принадлежи. Тези датасетове се генерират съответните методи от библиотеката на OpenCV. По този начин може например да обучим агента да разпознава даден шрифт или дори ръкописен текст. Според БДС, регистрационните номера на моторните превозни средства в България имат точно определени размери, шрифт, образец и набор от позволени символи. Тоест рег. ни номера според БДС са направени от **латински букви** и арабска цифрова азбука – 0,1,2,3,4,5,6,7,8,9 .

Тъй като има много различни проекти за Optical Character Recognition (с различни цели) с OpenCV, съм използвал готов dataset, който включва всички цифри, малки букви и главни букви. Той е по-

голям от необходимото, но ми върши работа. На базата на този dataset извършвам обучението и разпознаването на символите.

### 3.2. Dataset-ове

Намират се в два файла charactergroups.xml и images.xml. Всеки от валидните символи отговарящ на регулярния израз [0-9A-Z] съставя клас. Към този клас са назначени картини представящи съответната буква на която отговаря класа: напр. Класа A представя буквата A написана в шрифтовете Arial, Times New Roman и тн... Така за всяка буква имаме по един клас с поне една картина която е към този клас.




Ето примерна диаграма. На буквата A имаме картини imgA1, imgA2, които и сътоветстват и тн за останалите букви. Когато намерим предполагаем символ, го сравняваме с всички картини по определени критерии. Нека за този пример и за по-голяма определеност да кажем че го сравняваме по пиксели. Ако нашият символ има например 99% съвпадение по пиксели с imgA1 и 95% с imgD1, тогава взимаме най-близкият съсед, а в конкретния случай това е картината,с която имаме най-голямо съвпадение. Това ще е imgA1. imgA1 е от клас A => нашият символ отива в клас A или иначе казано нашият символ е буквата A.  На подобен принцип използваме kNN.

#### 3.2.1. Charactergroups.xml

Такъв вид има файла charactergroups.xml:

```
<?xml version="1.0"?>
<opencv_storage>
<charactergroups type_id="opencv-matrix">
  <rows>120</rows>
  <cols>1</cols>
  <dt>i</dt>
  <data>
    84 80 77 69 68 66 65 89 88 79 78 75 72 67 57 56 55 54 53 51 50 48 52
    49 80 77 68 66 65 89 88 84 79 78 75 72 69 67 52 49 57 56 55 54 53 51
    50 48 89 88 84 80 79 78 77 75 72 69 68 67 66 65 57 56 55 54 53 52 51
    50 49 48 89 88 84 80 78 77 75 72 69 68 66 65 79 67 55 51 50 49 57 56
    54 53 52 48 80 77 68 66 65 89 88 84 79 78 75 72 69 67 55 53 52 57 56
    54 51 50 49 48</data></charactergroups>
</opencv_storage>
```

Тези числа представляват ASCII стойностите на съответните символи, като първото число задава на кой ASCII символ отговаря първата картина в Images.xml , второто – за втората картина и тн....

### 3.2.2. Images.xml

Images.xml съдържа всички картини на съответните символи, като те са представени като съвкупност от числа с плаваща запетая. Такова е изискването от страна на OpenCV за прилагане на kNN.

Взимаме конкретния символ и неговият bounding rectangle. Записваме го в променлива от тип cv::Mat (Представяне на матрица за изображение в openCV). Оразмеряваме картината(на символа) със cv::resize() и взимаме новата матрица и я конвертираме формат с плаваща запетая cv::Mat::convertTo(). Всяка снимка/картина има форма която се състои от редове и колони на матрицата , и канали (ако тя е цветна). Поради изискванията на имплементацията на kNN, картината на нашият символ ще бъде преобразувана ввъ форма с 1 ред и 1 канал чрез

reshape(1, 1). Идва време да оставим kNN алгоритъма да направи своята магия. Извършваме kNearest->findNearest() върху преобразуваната матрица. Получаваме класа, на който отговаря най-близката картина, която наподобява на нашия. Това е търсеният символ. Записваме го в празна (0редове и 0 колони) променлива от тип cv::Mat във формат с плаваща запетая. findNearest() го връща като матрица с размери 1x1. Преобразуваме я във float – формат с плаваща запетая. Последователно извършваме преобразуване в целочислен тип след което взимаме стойността от ASCII таблицата на която отговаря това число. Вече имаме една буква от регистрационният номер. Аналогично намираме останалите.

## 4. Експерименти

Проведе се експеримент с 11 автентични снимки на коли с българска регистрация. Ето и резултатите:

| Грешни | Верни | Грешни с 1 символ | Грешни с > 1 символ |
|---|---|---|---|
| 6 | 5 | 4 | 1 |

| Откриване на рег. номер | Верни | Грешни |
|---|---|---|
| **Брой:** | 9 | 2 |

## 4.2. Резултати

На проведеният експеримент се установиха възможностите софтуерът да разпознава регистрационните номера и техните символи. Резултатите от разпознаването на рег. номер се разделят на 4 категории : Грешни, Верни, Грешни с 1 символ ( разликата е само 1 символ с оригиналният номер), Грешни с повече от 1 символ.

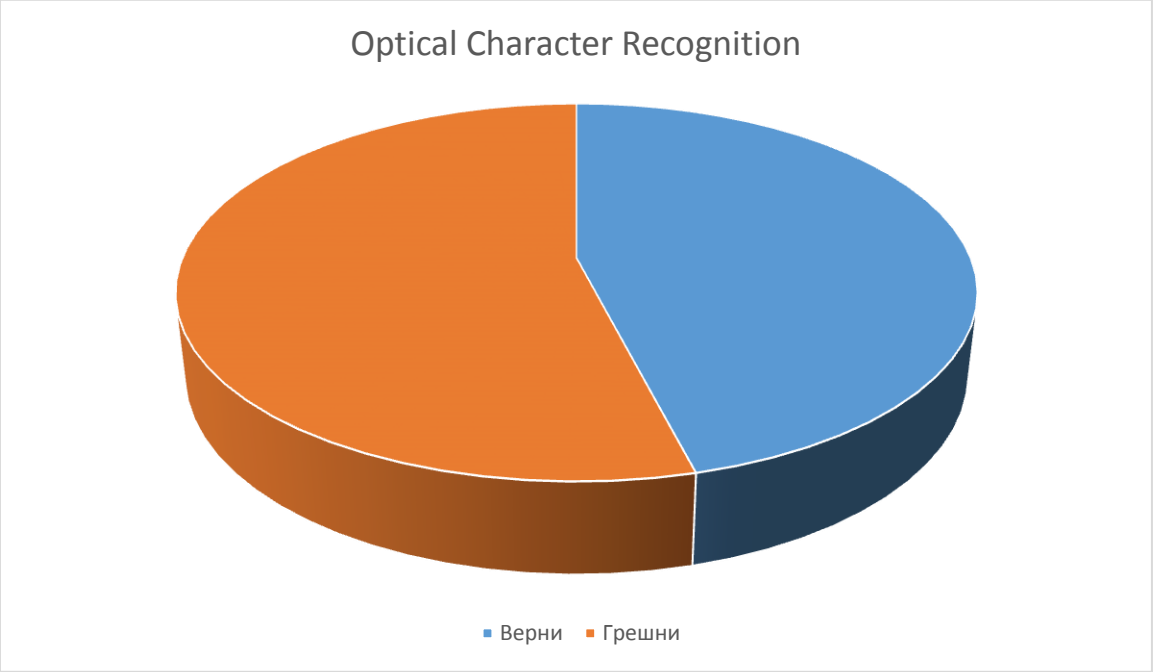
Резултати от Optical character recognition:

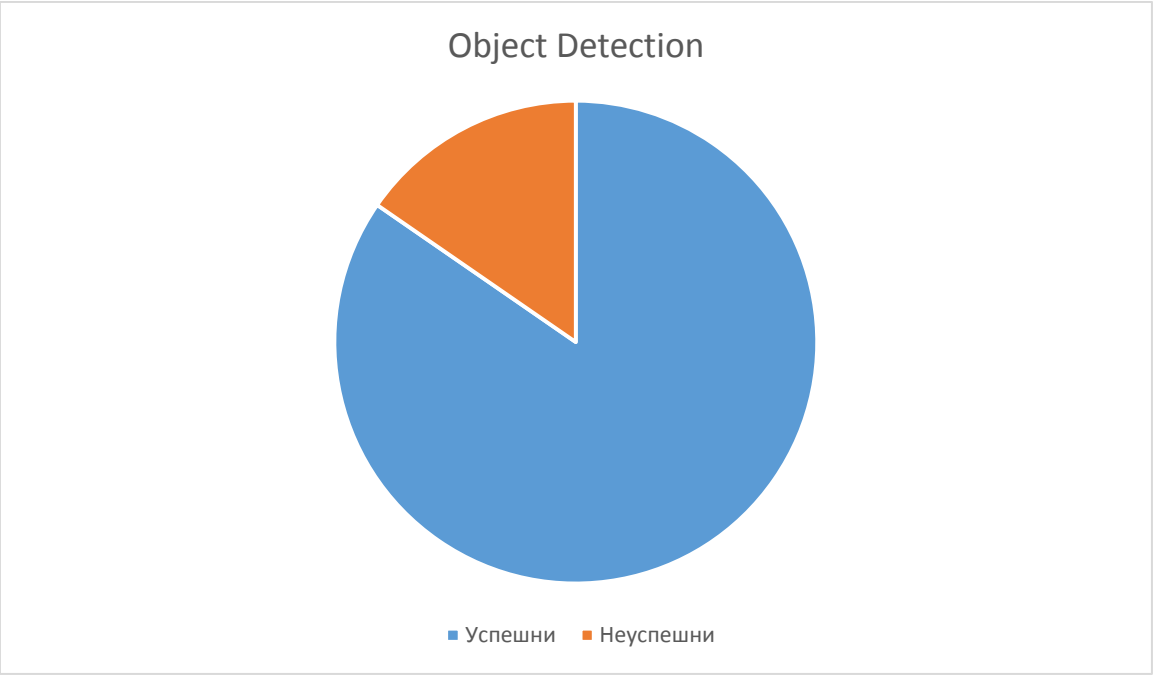
- 54.5% от регистрационните номера са разпознати **неуспешно**.
- 80% от грешните номера се разминават едва с 1 символ от оригинала.

- 20% от грешните номера ( 1 номер) се разминават с повече от 1 символ, т.е. имат сигнификантно отклонение.

Резултати от Object detection:

- 81.8% от всички номера биват открити **успешно и коректно** от програмата.
- 33.3% от грешните номера са сгрешени поради неуспех в стъпката за Object Detection.






### 4.3. Заключение

От проведенето изследване се виждат данни за потенциална полза от компютърното зрение и изкуственият интелект в засичането на обекти. Въпреки това обаче се наблюдава сигнификантна неточност при символно разпознаване, която не позволява изпозлването на текущият софтуер върху агенти чиито данни ще се взимат в предвид в правни институции. Не е подходящ и за засичане регистрационните номера на МПС. Въпреки това би могъл да се използва за тяхното заснемане. Трябва да се вземат в предвид и редица фактори като например качеството на снимките , ъгълът на заснемане и разбира dataset-овете.

### 4.4. Предложения

Това което би могло да се подобри е dataset-а. Би било от полза създаването на софтуер за генериране на dataset-Ове и последващо обучение с по-добър и по-пълен набор от данни. Това би подобрило успеваемостта и евентуално бихме могли да достигнем задоволителни резултати. Би могло и да се променят съответните критерии по които се откриват обектите. Така евентуално можем да достигнем и 100% успеваемост при Object detection.

## 5. Литература и ресурси

### 5.1. Използван софтуер

- Microsoft Visual Studio 2015 (licensed)

- OpenCV 3.2 (Open Source)
- Predefined datasets – датасетовете не са генерирани от мен и не са мои! Взети са от софтуер за четене на текстове от снимка чрез средствата на OpenCV (Потребител с име Ricardo от Stackoverflow).

## 5.2. Използвана литература

Цялата информация по-горе се базира на моите заключения от направеното проучване в документацията на OpenCV 3, Wikipedia.

- Wikipedia – image processing, computer vision, Gaussian Blur
- OpenCV 3 Documentation – methods & classes
- OpenCV 3 Community Support

Текстовете са писани лично от мен въз основа на това което съм научил от съответните източници. Софтуерът също е писан от мен с помощта на съответната документация. Всички останали източници са цитирани изрично!