НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет Інформатики і Обчислювальної Техніки

Кафедра Обчислювальної Техніки

До захисту допущено:

Завідувач кафедри

_____ Сергій СТІРЕНКО

«___»_____2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Система пошуку зображень транспортних засобів за номерними знаками»

Виконав:
студент (-ка) IV курсу, групи ІП-62
КОСТЯЄВ Анатолій Олегович _____
Керівник:
Доцент, кандидат технічних наук
ПОРЄВ Віктор Миколайович _____
Консультант з нормоконтролю:
Професор, доктор технічних наук
СІМОНЕНКО Валерій Павлович _____
Рецензент:
_____
_____ _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМ. ІГОРЯ СІКОРСЬКОГО»
Факультет Інформатики та обчислювальної техніки
Кафедра Обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних та комп'ютеризованих систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____2020 р.

ЗАВДАННЯ

на бакалаврський дипломний проєкт студента

Костяєва Анатолія Олеговича
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Система пошуку зображень транспортних засобів за номерними знаками*
керівник проекту (роботи) доцент, к.т.н. Порєв Віктор Миколайович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
2. Термін здачі студентом закінченого проекту (роботи) *5 червня 2020* р.
3. Вихідні дані до проекту (роботи) *технічна документація. теоретичні та статистичні дані*
4. Зміст розрахунково-пояснювальної записки
Огляд та аналіз методів і алгоритмів класифікації та пошуку зображень, зразки та можливості сучасних систем розпізнавання зображень, архітектура системи пошуку
5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, узагальнена схема роботи системи, блок-схема алгоритму системи.
6. Консультанта проекту (робота), з вказівкою розділів роботи, які до них вносяться

Розділ	Консульт	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проекту (роботи)	Строк виконання етапів проекту(роботи)	Примітки
/	*Затвердження теми роботи*	*10.12.2019-15.12.2019*	
2	*Вивчення та аналіз завдання*	*15.12.2019-15.03.2020*	
3.	*Розробка архітектури та загальної структури систем*	*15.03. 2020-25.03. 2020*	
4.	*Розробка структур окремих підсистем*	*25.03. 2020-5.04. 2020*	
5	*Програмна реалізація системи*	*5.04. 2020-15.04. 2020*	
6	*Оформлення пояснювальної*	*15.04. 2020-20.05. 2020*	
7	*Захист програмного продукту*	*05.06. 2020*	
8	*Передзахист*	*15.05. 2020*	
9	*Захист*	*20.06. 2020*	

Студент-дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Дипломну роботу виконано на 84 аркушах, вона містить 4 додатки та перелік посилань на використані джерела з 11 найменувань. У роботі наведено 37 зображень та 30 формул.

Метою даної дипломної роботи є створення системи пошуку зображень транспортних засобів за номерником знаками.

У роботі виконано огляд та аналіз методів і алгоритмів класифікації та пошуку зображень, оглянуто зразки та можливості сучасних систем розпізнавання зображень, розроблено архітектуру системи пошуку. Для розв’язання задачі в роботі тришарову архітектуру побудовану за принципом REST, основану на структурному шаблоні Модель-Вид-Контролер.

Ключові слова: пошук зображень, класифікація зображень, розпізнавання зображень, алгоритм.

ABSTRACT

The thesis is performed on 84 sheets, it contains 4 appendices and a list of references to the sources used with 10 titles. The paper contains 37 figures and 30 formulas.

The purpose of this thesis is to create a Vehicle Image Search Engine.

The analysis of the existing types of architectural solutions of the set task - multilayer types of architectures, basic information on architecture design is carried out in the work. Their comparison is performed in terms of development efficiency, load resistance, fault tolerance, reliability of data storage. To solve the problem, the three-layer architecture is built on the principle of REST, based on the structural template Model-View-Controller.

Keywords: image search, image classification, image recognition, algorithm.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ІАЛЦ 467400.001 ВП	Відомість проєкту	1	
3	А4	ІАЛЦ 467400.002 ПЗ	Пояснювальна записка	56	
4	А3	ІАЛЦ 467400.003 Д1	Принципова схема	1	
5	А3	ІАЛЦ 467400.004 Д2	Структурна схема	1	
6	А3	ІАЛЦ 467400.005 Д3	Функціональна схема	1	
7	А4	ІАЛЦ 467400.006 Д4	Лістинг частин програми	16	

Пояснювальна записка
до дипломного проєкту
на тему: «Система пошуку зображень транспортних засобів за номерними знаками»

Київ – 2020 року

ЗМІСТ

ВСТУП

Актуальність теми. В даний за допомогою комп'ютерних технологій автоматизується широке коло справ і процесів, які в недалекому минулому покладалися тільки на людину. Інформаційні технології використовуються всюди: в освіті, в промисловості, в транспорті і т.п. Програмісти всього світу розробляють абсолютно нові і удосконалюють вже існуючі рішення для автоматизації.

Рішення складності фіксування транспортного засобу за реєстраційним номерним знаком на даний момент є важливим аспектом безпеки, обліку і контролю. Використовувати такий продукт можна в різних сферах, які стосуються автотранспорту, наприклад, необхідність ідентифікації та обліку автомобілів. Так само прикладом можуть служити автотранспортна промисловість, контролювання швидкості руху, гаражі, автомобільні стоянки, контроль в'їзду на територію підприємства і т.п.

В даний час існує не так багато систем визначення номерних знаків, і багато з них не є по справжньому якісною продукцією. Найкращі системи розпізнавання показують результат - 90 відсотків розпізнавання номерів. Однак, паралельно зі створенням алгоритмів, розробляються апаратні засоби, які будуть займатися розпізнаванням.

Системи і комплекси, що володіють високою точністю і швидким розпізнаванням, як правило, дуже дорогі, або вимагають ретельного вивчення інструкції, на яку іноді часу немає. Висока ціна і складність існуючих систем не дозволяють здійснити їх масове впровадження в нашій країні.

Завдання розпізнавання і фіксації автомобіля можна розділити на три підзадачі: знаходження номерного знака на машині, локалізація номерної пластини, тобто самого номера і розпізнавання символів на ньому.

Метою написання роботи є розробка системи пошуку зображень транспортних засобів за номерними знаками. У загальному випадку розпізнавання реалізується в чотири етапи: попередня обробка зображення, сегментація, розпізнавання символів, запис номера в сховище даних.

Попередня обробка зображення полягає в обробці з виділенням номерного знака на отриманому зображенні різними фільтрами з метою поліпшення його якості. На етапі сегментації виділяються букви і цифри знака, які потім розпізнаються будь-яким методом, після чого розпізнаний об'єкт заноситься в обране джерело даних.

При написанні роботи були поставлені наступні задачі:

1. Виконати огляд та аналіз методів і алгоритмів класифікації та пошуку зображень номерних знаків;

2. Оглянути зразки та можливості сучасних систем розпізнавання зображень;

3. Виконати розробку архітектури системи пошуку.

Предметом дослідження є система розпізнавання номерних знаків.

Об’єкт дослідження – процес розпізнавання транспортних засобів за номерними знаками.

При написанні роботи були використані методи аналізу, дослідження і порівняння.

РОЗДІЛ 1. ОГЛЯД ТА АНАЛІЗ МЕТОДІВ КЛАСИФІКАЦІЇ ТА ПОШУКУ ЗОБРАЖЕНЬ

1.1 Призначення систем розпізнавання транспортних засобів за номерними знаками

Автоматизація розпізнавання номерів автомобілів – одне з завдань для корпоративних систем безпеки. Дані системи дозволяють істотно підвищити розпізнавання машин і сформувати звітність за багатьма аспектами. Є проблеми, з якими можна зіткнутися при розробці даної системи, такі як [1]:

- Знаходження розташування номера на машині;
- Локалізація номера автомобіля;
- Сегментація символів номерного знака автомобіля;
- Розпізнавання кожного символу в номері автомобіля.

Серед завдань розпізнавання можна виділити наступні:

- Знаходження номерного знака на машині;
- Локалізація знака доступними фільтрами для розпізнавання;
- Сегментація і обробка символів;
- Запис даних в БД для подальшого аналізу.

Проаналізувавши перераховані вище проблеми і завдання, можна зробити висновок, що для їх вирішення рекомендується застосування сучасних інформаційних технологій.

1.2 Загальна архітектура апаратного комплексу розпізнавання

Система зчитування автомобільних номерів в загальному випадку складається декількох апаратних і програмних модулів (рисунок 1.1) [3]:

- відеокамера;
- плата відеозахоплення;
- модуль знаходження номера;
- модуль обробки зображення номера

- модуль розпізнавання;
- зовнішня чи внутрішня БД;

модуль обробки інформації.

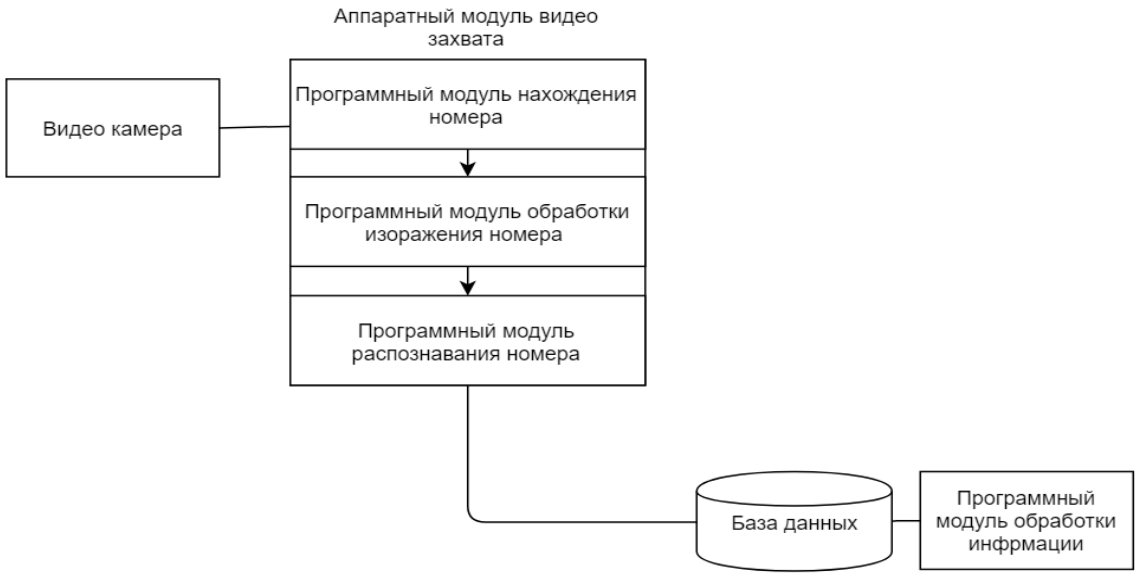


Рисунок 1.1 – Загальна архітектура апаратного комплексу розпізнавання. Джерело [3]

Зображення з відеокамери надходить на вхід модуля знаходження. Програмний детектор знаходить на зображенні автомобіль, що рухається. Потім алгоритмом визначається положення або знаходження номерного знака, тобто область, де перебуває номерний знак.

Після цього виділений номер обробляється програмним забезпеченням для оптичного розпізнавання. Далі йде база даних, яка в залежності від поставлених перед системою завдань, може мати різну схему, яка забезпечує зберігання даних, потім вже обробляється інформація з БД для:

- Перевірки номера на факт викрадення

- Аналіз проїжджих машин і т.п.

Основними завданнями є: захоплення номера, дати і часу появи автомобіля в куті огляду камери; запис стоп-кадру автомобіля в потрібний час або відрізка відео з машиною; перевірка номера на реєстрації в БД і т.д [4].

1.3 Огляд алгоритмів класифікації і пошуку зображень

Розглянемо алгоритми класифікації та пошуку зображень, які можна використовувати для систем розпізнавання номерних знаків.

Бінаризація за порогом

Найбільш просте перетворення - це бінаризація зображення за порогом (image thresholding). У RGB зображеннях і зображеннях в градаціях сірого порогом є значення кольору [5].

Досить часто зустрічаються ідеальні завдання, в яких такого перетворення досить. Припустимо, потрібно автоматично виділити фігури на зображенні з чорним фоном (рисунок 1.2):

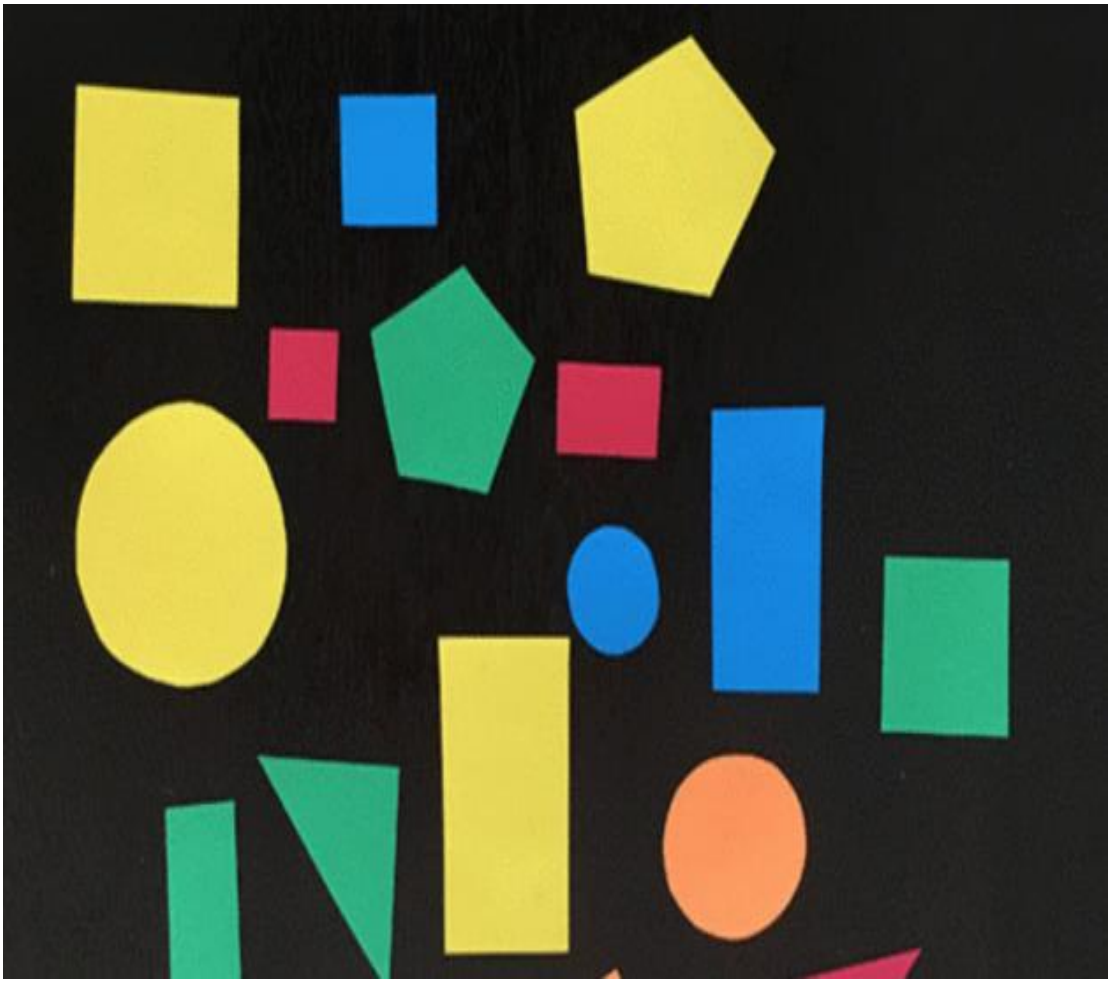


Рисунок 1.2 – Фігури на чорному тлі

Вибір порога, за яким відбувається бінаризація, багато в чому визначає процес самої бінаризації.

В даному випадку, зображення було бінаризованими за середнім кольором (рисунок 1.3):

Рисунок 1.3 - Бінаризований малюнок за середнім кольором

Зазвичай бінаризація здійснюється за допомогою алгоритму, який адаптивно вибирає поріг. Таким алгоритмом може бути вибір моди.

Фільтрація контурів.

Окремий клас фільтрів - фільтрація меж і контурів. Контури досить корисні, коли ми хочемо перейти від роботи з зображенням до розпізнавання і роботі об'єктів на ньому. Коли об'єкт складно розпізнати, але він добре виділяється, то часто єдиним способом роботи з ним - виділення його контурів, для визначення що це за об'єкт.

В даний час існує більше 10 алгоритмів, що вирішують завдання фільтрації контурів, ми перерахуємо основні 5:

- оператор Кенні

- оператор Собеля

- оператор Лапласа

- оператор Прюітт

- оператор Робертса

Найчастіше використовується саме Кенні, який виробить контури при правильних налаштуваннях (рисунок 1.4):

Рисунок 1.4 - використання оператора Кенні

Каскад Хаара використовується в методі Віоли-Джонса. Цей алгоритм дозволяє виявляти об'єкти на зображеннях.

Алгоритм добре справляється з розпізнаванням різних класыв зображень, хоча основним завданням при його створенні було виявлення осіб людини. Основна перевага детектора Хаара - це швидкість. Завдяки високій швидкості обробки зображення можна з легкістю обробляти відео в потоці.

Детектор Хаара використовується для розпізнавання для великої кількості класів об'єктів. До них відносяться руки, ноги і т.п., машини, пішоходи, назва адрес, домашні тварини тощо [5].

Принцип даного методу полягає в тому, що класифікатор збирається на примітивах Хаара шляхом розрахунку значень ознак. Для навчання на вхід класифікатора спочатку подається вибірка, що складається з «правильних» зображень з попередньо виділеною областю на зображенні, далі починається перебір примітивів і вираховування значення ознаки. Попередньо обчислені значення зберігаються в файлі з потрібною назвою, формат якого xml.

Метод k-Means**.** Досить простий, але і в той же час досить неточний метод кластеризації в класичній реалізації. У ньому розбивається множина елементів векторного простору на відоме нам число кластерів k [6].

Етапи та дії алгоритмів такі, що вони прагнуть мінімізувати середньоквадратичне відхилення на точках кожного кластера. Ідея полягає в тому, що на кожній ітерації циклу переобчислюють центр мас для кожного кластера, отриманого на попередньому етапі, після цього вектори розбиваються на сегменти знову відповідно до того, який з нових центрів виявився найближче за обраною метрикою. Потім алгоритм завершується, коли на якийсь ітерації не відбулося зміни кластерів.

Найбільш поширені проблеми алгоритму k-means:

- необхідно заздалегідь знати скільки кластерів нам потрібно взяти. Можна використовувати такий метод, який ґрунтувався на знаходженні кластерів, розподілених по якомусь закону, який можна самостійно обирати.

Після цього можна виконати класичний алгоритм k-means, який давав більш точні результати.

- даний алгоритм найбільш чутливий до вибірки початкових центрів кластерів. Зазвичай мається на увазі випадковий вибір кластерів - це класичний варіант, через нього з'являлися похибки. Щоб уникнути такого, можна проводити дослідження об'єкта для більш точного визначення центрів початкових кластерів.

- не виходить впорається із завданням, коли об'єкт належить до різних кластерів в однаковій мірі або не належить жодному з них.

Алгоритм розпізнавання номерного знака складається з наступних етапів (рисунок 1.5):

1. Нормалізація
2. Сегментація.
3. Розпізнавання.

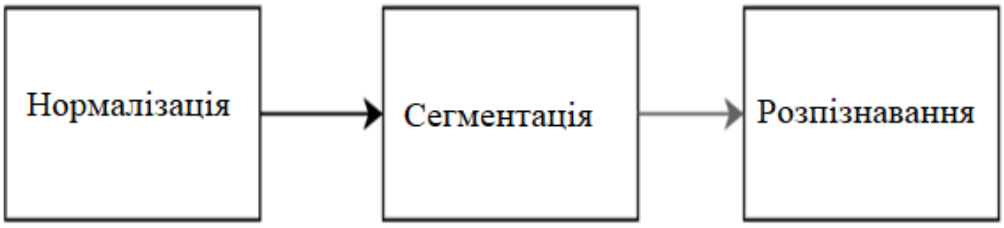


Рисунок 1.5 - Етапи розпізнавання

Процедура нормалізації використовується практично завжди після отримання інформації і являє собою застосування операцій усереднення і вирівнювання гістограм, різного типу фільтрів для виключення перешкод, а також придушення зовнішніх шумів, так само і перенесення зображення в горизонтальний вид.

Під сегментацією розуміється процес поділу зображення на окремі символи. Кінцевий етап обробки - розпізнавання. Для цього етапу вхідними даними є зображення, отримані в результаті шумозаглушення і процесу сегментації. Розпізнавання символів можливе за допомогою готових програм, або за допомогою алгоритмів [7].

Розглянемо алгоритм нормалізації. Виявлення номера відбувається не зовсім точно і вимагає подальшого уточнення його положення, а також поліпшення якості знімка.

По-перше, нам потрібно нормалізувати, а тобто стабілізувати наш номерний знак по горизонталі. Це виділяється за рахунок алгоритму перетворення Хафа, який виділяє основні лінії, за якими потім можна повертати зображення.

Класичний алгоритм перетворення Хафа пов'язаний з ідентифікацією прямих в зображенні, що дозволить використовувати його в роботі.

Наступним кроком по обробці номера буде збільшення різкості і контрасту.

Крім описаного способу, можна виділяти границі за допомогою градієнтних фільтрів першого і другого порядку. Можна застосувати фільтр другого порядку - LoG-фільтр (Marr-Hildreth), який працює таким шляхом, як звірка зображення з лапласіаном функції Гаусса.

Він поєднує в собі виявлення границь зі згладжуванням.

При обробці зображення є певні труднощі, такі як:

- Низька ефективність розпізнавання при забрудненні номера.

- Горизонтальні границі номера не завжди є хорошим орієнтиром для розбиття нього за сегментами.

- Автомобільні номери різних країн мають різні шрифти і різний формат.

- Автомобільні номери можуть бути встановлені в місця, не передбачені конструкцією транспортного засобу, або відсутні.

Розглянемо алгоритм сегментації.

Якщо пов'язана область (контур) має висоту в пікселях від H1 до H2 а ширина і висота пов'язана відношенням від K1 до K2, то залишаємо в кадрі і відзначаємо, що в цій області може бути знак.

На цьому етапі залишаться лише цифри і букви, решта «сміття» з кадру піде. Для сегментації можна обрати контури – прямокутники, навести їх до одного масштабу і працювати з кожною буквою / цифрою окремо (рисунок 1.6).

Рисунок 1.6 – Приклад сегментації

Так само можна провести бінаризацію, тобто розділити пікселі на чорне і біле, простіше кажучи на два класи.

При поділі номера на символи ця операція важлива, так як ми наближаємося до чорно-білого зображенню.

Після цього досить знайти максимум або максимуми горизонтальних діаграм, вони і будуть проміжками між зображеннями символів. Особливо якщо ми знаємо відстань між символами, то розбиття на цифри і букви по гістограмі спрацює відмінно.

Це приклад одного з допустимих алгоритмів, щоб розділити кожен символ один від одного (рисунок 1.7).

Рисунок 1.7 - Сегментація за допомогою максимумів

При значному забрудненні номера періодичні максимуми при розбитті на символи можуть не проявитися, і можливо вони не будуть видні, але самі символи можуть бути візуально цілком читані.

Варто відзначити, що верхня і нижня горизонтальні лінії не зовсім хороший орієнтир. Номери можуть знаходиться в поглибленні, і так само лінії може і не бути, тому що вона в ущільненні, але це зустрічається в більшості випадків на американських машинах. А верхня межа заднього номера просто часто прикрита елементами кузова, і тоді номер в принципі складно знайти. Так само іноді через неправильну обробку можуть з'явиться зайві елементи, які не є символами.

Для вирішення задачі розпізнавання символів використовуються нейронні мережі або алгоритми класифікації, які входять в розділ машинного навчання. Наприклад, нейронні мережі Кохонена, що забезпечують топологічне упорядкування вхідного простору образів. Вони відображають вхідні n-мірні і вихідні m-мірні сегменти.

Ключовим аспектом нейронних мереж є навчання за вибіркою.

Алгоритми навчання нейронної мережі спрощено зводяться до визначення коефіцієнта між двома нейронами, що підтверджують цю залежність.

Найбільш поширеним алгоритмом навчання нейронної мережі є алгоритм зворотного поширення помилки.

Відповідний граф передачі сигналу в процесі навчання за методом зворотного поширення помилки, який ілюструє як пряму, так і зворотну фазу обчислень, представлений на рисунку 1.8 для випадку $L = 2$ і $m_0 = m_1 = m_2 = 3$

У верхній частині графа передачі сигналу показаний прямий прохід, в нижній - зворотний. Останній ще носить назву графа чутливості для обчислення локальних градієнтів в алгоритмі зворотного поширення.

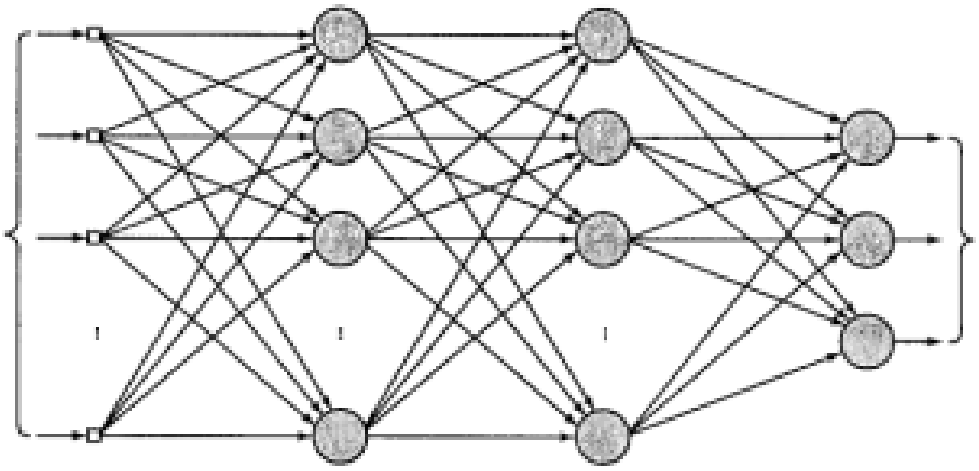


Рисунок 1.8 - Граф багатошарового персептрона з двома прихованими шарами

Послідовне коригування ваги є кращим режимом алгоритму зворотного поширення для реалізації в реальному часі. В цьому режимі алгоритм циклічно обробляє приклади з навчальної множини $\{(x(n), d(n))\}_{n=1}^{N}$ наступним чином.

Граф передачі сигналу представлений на рисунку 1.9.

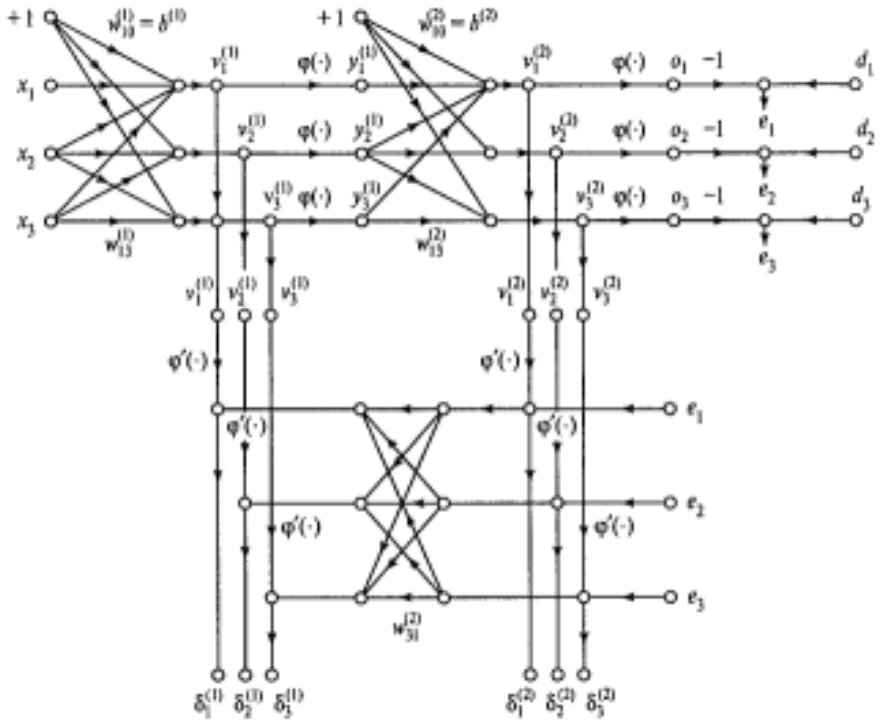


Рисунок 1.9 - Граф передачі сигналу для процесу навчання за методом зворотного поширення

Верхня частина графа - прямий прохід, нижня - зворотний прохід.

Ініціалізація. Припускаючи відсутність апріорної інформації, генеруємо синаптичні ваги і порогові значення за допомогою рівномірно розподілених чисел із середнім значенням 0. Дисперсія вибирається таким чином, щоб стандартне відхилення індукованого локального поля нейронів доводилася на лінійну частину сигмоїдальної функції активації (і не досягала області насичення) [8].

Пред'явлення прикладів навчання. У мережу подаються образи з навчальної множини (епохи).

Прямий прохід (forward computation). Нехай приклад навчання представлений парою $(x(n), d(n))$, де $x(n)$- вхідний вектор, який пред'являється вхідному шару сенсорних вузлів; $d(n)$- бажаний відгук, що надається вихідному прошарку нейронів для формування сигналу помилки.

Обчислюємо індуковані локальні поля і функціональні сигнали мережі, проходячи по ній пошарово в прямому напрямку [9].

Індуковане локальне поле нейрона j шару l обчислюється за формулою:

$$y^l(n) = \sum \omega_{ij}^l(n)\, y_i^{l-1}(n),$$

де $y_i^{(l-1)}(n)$ - вихідний (функціональний) сигнал нейрона i, розташованого в попередньому шарі $l-1$, на ітерації n;

$w_{ji}^{(l)}(n)$ - синаптична вага зв'язку нейрона j шару l з нейроном i шару $l-1$.

для $i=0$ $y_0^{(l-1)}(n)=+1$, а $w_{j0}^{(l)}(n)=b_j^l(n)$ - поріг, застосовуваний до нейрона j шару l.

Якщо використовується сигмоїдальна функція, то вихідний сигнал нейрона j шару l виражається наступним чином:

$$y_j^{(l)}(n) = \varphi_j(v_j(n)).$$

Якщо нейрон j знаходиться в першому прихованому шарі (тобто $l=1$), то

$$y_j^{(0)}(n) = x_j(n)$$

де $x_j(n)$ - j-й елемент вхідного вектора x (n).

Якщо нейрон j знаходиться в вихідному шарі (тобто $l=L$, де L - глибина мережі), то

$$y_j^{(L)}(n) = 0_j(n)$$

Обчислюємо сигнал помилки

$$e_j(n) = d_j(n) - 0_j(n)$$

де $d_j(n)$ – j-й елемент вектора бажаного відгуку $d(n)$.

Зворотний прохід. Обчислюємо локальні градієнти вузлів мережі за такою формулою:

$$\delta_j^{(l)}(n) = \begin{cases} e_j^{(L)}(n)\varphi_j'(v_j^{(L)}(n)), \\ \varphi_j'(v_j^{(l)}(n))\sum_k \delta_k^{(l+1)}(n)\delta_{kj}^{(l+1)}(n), \end{cases}$$

для нейрона j вихідного шару L, для нейрона j прихованого шару l, де штрих до функцій $\varphi_j'(\cdot)$ позначає диференціювання по аргументу.

Зміна синаптичних ваг шару l мережі виконується відповідно до узагальненого дельта-правила:

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha\left[w_{ji}^{(l)}(n-1)\right] + \eta\delta_j^{(l)}(n)y_j^{(l-1)}(n)$$

де η-параметр швидкості навчання;

α - постійна моменту.

Ітерації. Послідовно виконуємо прямий і зворотний проходи, пред'являючи мережі всі приклади навчання з епохи, поки не буде досягнуто критерію зупину.

Порядок подання прикладів навчання може випадковим чином змінюватися від епохи до епохи. Параметри моменту і швидкості навчання налаштовуються (і зазвичай зменшуються) в міру зростання кількості ітерацій.

Метою навчання є знаходження такого набору вагових коефіцієнтів мережі, який забезпечує рішення даного конкретного завдання.

Розглянемо навчання мережі даними алгоритмом на прикладі тришарової нейронної мережі, представленої на рисунку 1.10.

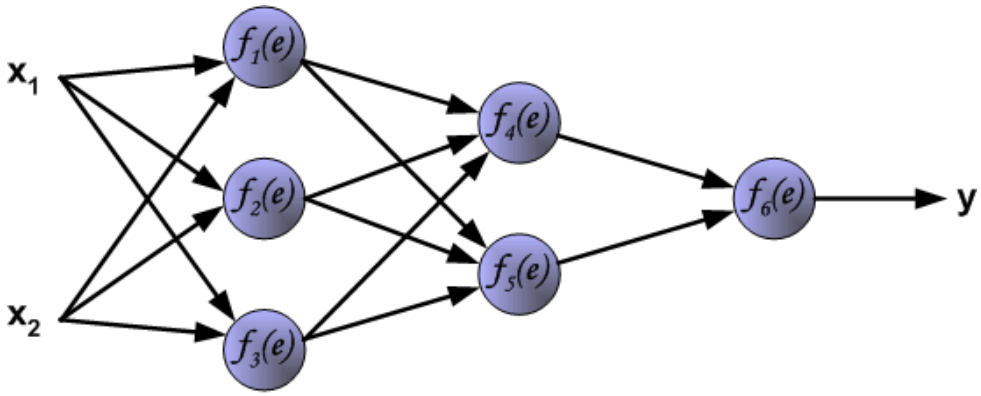


Рисунок 1.10 - Схема багатошарової нейронної мережі

Для навчання нейронної мережі ми повинні підготувати навчальні приклади.

Є приклади, що складаються з вхідних сигналів (X_1 і X_2) і бажаного результату Z.

У кожній ітерації вагові коефіцієнти нейронів підганяються з використанням нових даних з навчальних прикладів. Зміна вагових коефіцієнтів синаптичного зв'язку і становлять суть алгоритму.

Кожен крок навчання починається з впливу вхідних сигналів з навчальних прикладів. Після цього можна визначити значення вихідних сигналів для всіх нейронів в кожному шарі мережі.

На рисунку 1.11 показано напрямок сигналу в мережі.

$W(X_{mn})$- вага зв'язку між входом X_m і нейроном n у вхідному шарі.

$Y(n)$- вихід нейрона n.

$$y_1 = f_1(w_{(x_1)1}x_1 + w_{(x_2)1}x_2),$$
$$y_2 = f_2(w_{(x_1)2}x_1 + w_{(x2)2}x_2),$$
$$y_3 = f_3(w_{(x_1)3}x_1 + w_{(x_2)3}x_2).$$

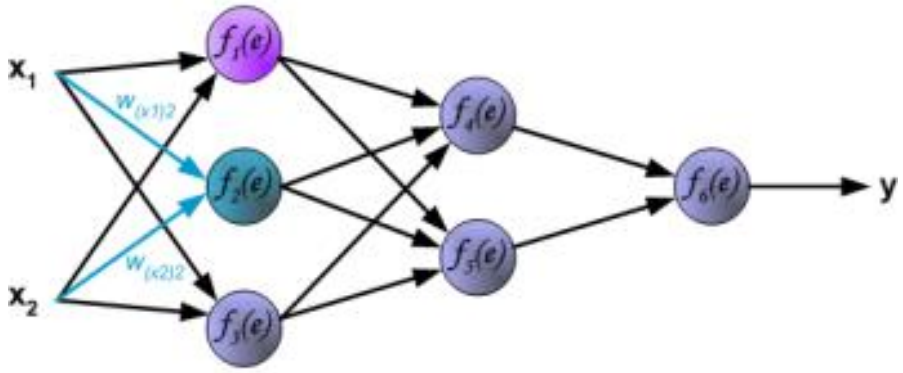


Рисунок 1.11 - Поширення сигналу від входу до 1-го прихованого шару

W_{mn} - ваги зв'язків між виходом нейрона m і входом нейрона n в наступному шарі.

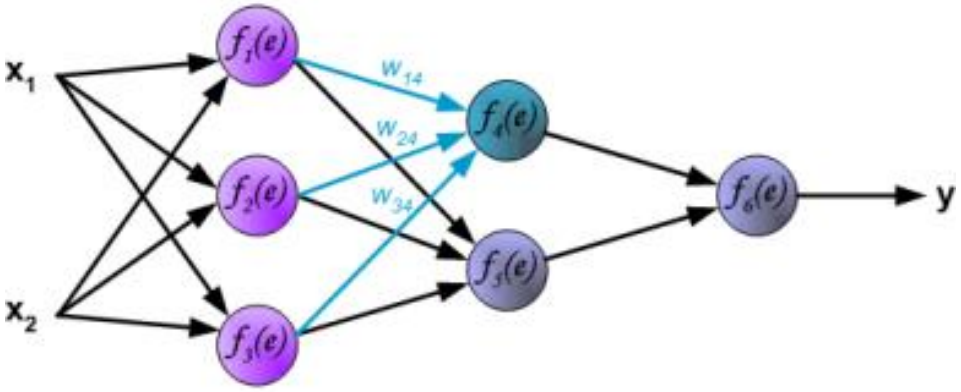


Рисунок 1.12 - Поширення сигналу до 2-го прихованого шару

На рисунку 1.12 показано як поширюється сигнал від першого шару до другого. У такому напрямку обчислюються виходи всіх нейронів, в тому числі і вихідного:

$$y_4 = f_4(w_{14}y_1 + w_{24}y_2 + w_{34}y_3),$$
$$y_5 = f_5(w_{15}y_1 + w_{25}y_2 + w_{35}y_3).$$

Далі йде прохід сигналу через вихідний шар:

$$y = f_6(w_{46}y_4 + w_{56}y_5).$$

Наступний етап алгоритму полягає в наступному: вихідний сигнал y порівнюється з бажаним виходом мережі z.

δ - помилка (різниця) між сигналами z і y: $\delta = z - y$.

На рисунку 1.13 показано розподіл помилки по мережі.

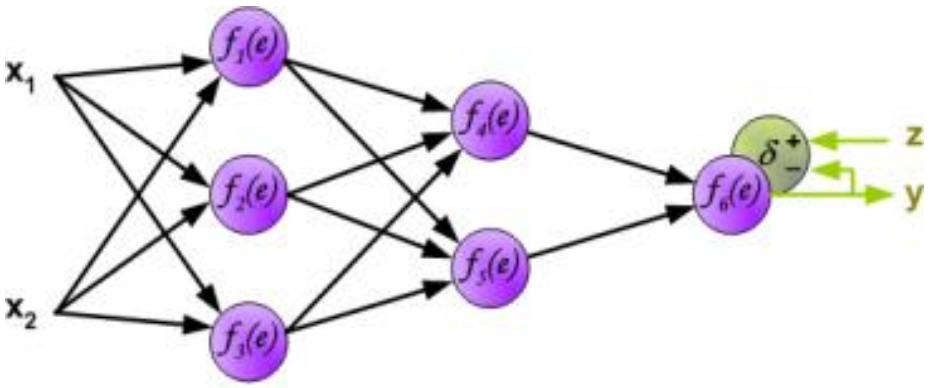


Рисунок 1.13 - Поширення помилки

Далі сигнал δ поширюється в зворотному напрямку на всі нейрони, чиї вихідні сигнали були вхідними для останнього нейрона.

$$\delta_4 = w_{46}\delta,$$
$$\delta_5 = w_{56}\delta.$$

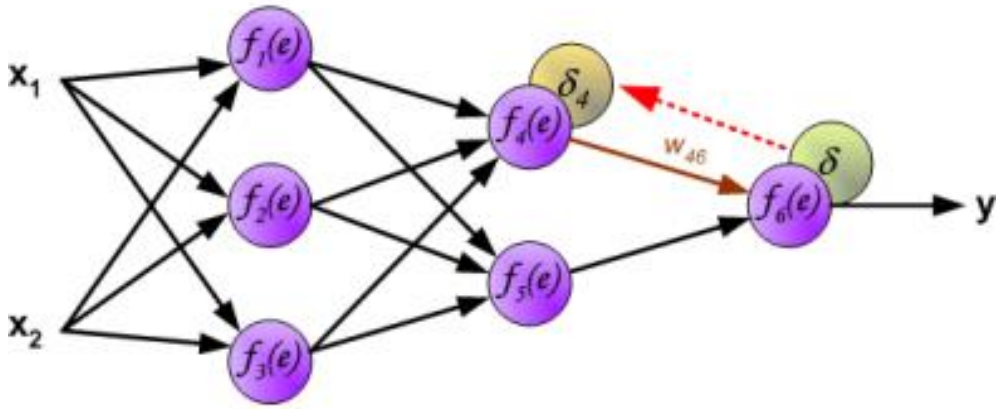


Рисунок 1.14 - Поширення помилки від вихідного шару до 2-го прихованого шару

Вагові коефіцієнти W_{mn} рівні тим же коефіцієнтам, що використовувалися під час обчислення вихідного сигналу y.

Змінюється напрямок потоку даних (сигнали передаються від виходу до входу). Процес повторюється для всіх верств мережі.

Якщо помилка прийшла від декількох нейронів - вона підсумовуються (рисунок 1.15):

$$\delta_1 = w_{14}\delta_4 + w_{15}\delta_5,$$
$$\delta_2 = w_{24}\delta_4 + w_{25}\delta_5,$$
$$\delta_3 = w_{34}\delta_4 + w_{35}\delta_5.$$

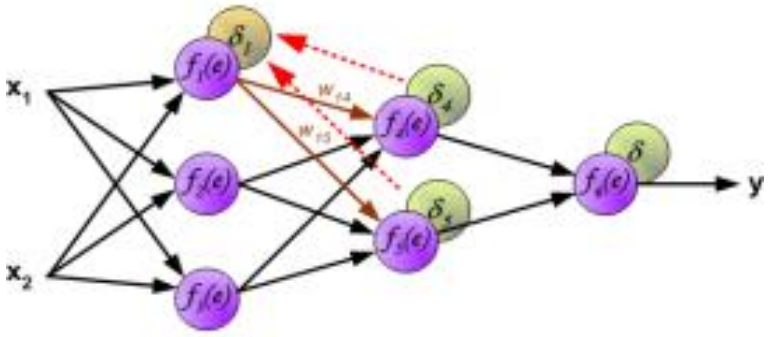


Рисунок 1.15 - Поширення помилки δ в зворотному напрямку в 1-ому прихованому шарі

Існує можливість перенавчання мережі. Весь набір образів, наданих до навчання, буде вивчено мережею, але будь-які інші образи, навіть дуже схожі, можуть бути класифіковані невірно.

Коли обчислюється помилка δ для кожного нейрона - можна скорегувати вагові коефіцієнти кожного нейрона (рисунок 1.16).

$$w'_{(x_1)1} = w_{(x_1)1} + \eta\delta_1 \frac{df_1(e)}{de} x_1,$$

$$w'_{(x_2)1} = w_{(x_2)1} + \eta\delta_1 \frac{df_1(e)}{de} x_2.$$

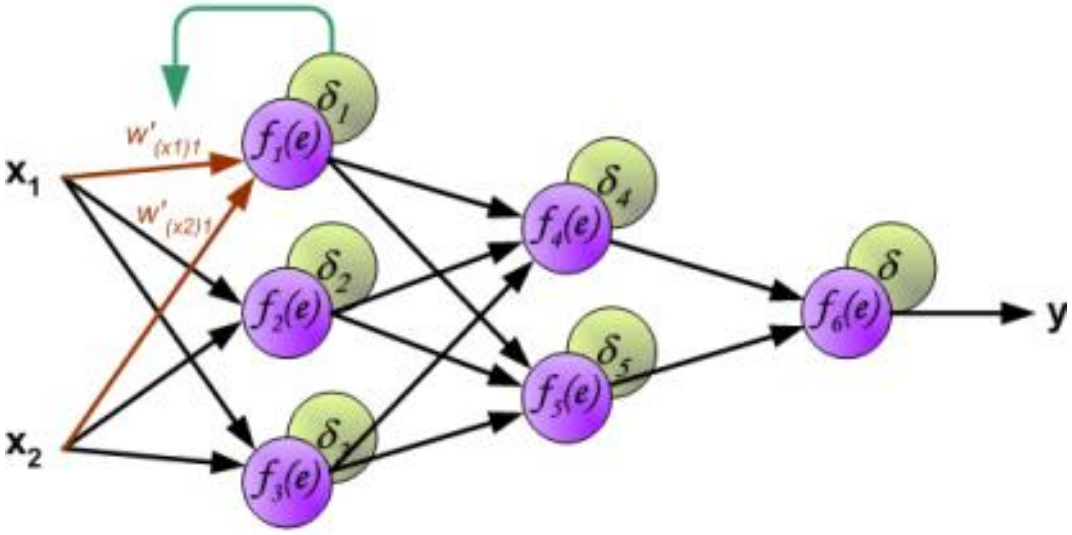


Рисунок 1.16 - Коригування ваг в 1-му шарі

де, $\frac{df(e)}{de}$ - похідна від функції активації нейрона (того, чиї ваги налаштовуються),

η - коефіцієнт, що впливає на швидкість навчання.

$$w'_{(x_1)2} = w_{(x_1)2} + \eta\delta_2 \frac{df_2(e)}{de} x_1,$$

$$w'_{(x_2)2} = w_{(x_2)2} + \eta\delta_2 \frac{df_2(e)}{de} x_2.$$

так само відповідно обчислюємо $w'_{(x_1)3}$ і $w'_{(x_2)3}$

Отже, сутність алгоритму зворотного поширення помилки зводиться до завдання найбільш довільних невеликих початкових значень ваг зв'язків нейронів. Потім за допомогою алгоритму рекурсії, починаючи з вихідних вузлів у напрямку до першого прихованого шару для всіх навчальних пар «значення вхідних ознак - значення вихідної ознаки» (прикладів з навчальної вибірки) обчислюється вихід

мережі (Y). Даний процес триває до тих пір, поки не буде досягнутий мінімальний рівень помилки.

Так само можна використовувати відкрите програмне забезпечення, яке виконує автоматичне розпізнавання як одиничної літери, так і відразу тексту, як Tessaract ORC. Ця програма зручна тим, що є для будь-яких ОС, стабільно працює, але по використанню не завжди ясно що потрібно робити з навчанням. Так само програма дуже погано працює з замиленим, битим, брудним і деформованим текстом.

ВИСНОВОК ДО РОЗДІЛУ 1

Таким чином, існує багато методів і алгоритмів пошуку зображень для систем розпізнавання номерних знаків. Від правильного вибору методу залежить відсоток правильного розпізнавання номерних знаків. Перспективним є використання нейронних мереж.

РОЗДІЛ 2. ОГЛЯД ЗРАЗКІВ ТА МОЖЛИВОСТЕЙ СУЧАСНИХ СИСТЕМ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ

2.1 Математична модель розпізнавання зображень

В деяких випадках уникнути відмінностей з еталонним зображенням не вдається.

Такими відзнаками можуть бути: умови отримання початкового зображення (відстань від камери до об'єкта, ракурс, умови освітлення і т.д.), параметри вихідного зображення (гамма, контраст, ступінь зашумленості і т.д.).

З огляду на перелічені відмінності, автоматичне розпізнавання номерів може бути ускладнене, якщо не сформульовані обмеження, що накладаються на умови підготовки зображень.

Якщо на вхід системи розпізнавання надходять зображення, що не відповідають перерахованим нижче обмеженням, результат роботи системи може бути некоректним.

Розмір області, що містить номерний знак на зображенні. В загальному випадку потрібно, щоб довжина і ширина області, що містить номерний знак, була не меншою 100 пікселів, а відстань між координатами центрів має бути більше 50 пікселів.

В області номерного знаку не повинні бути присутніми затемнення.

Освітленість. При формуванні зображення, об'єкт необхідно рівномірно висвітлювати. Наявність будь—якого спрямованого освітлення на зображенні є критичним для розпізнавання. Необхідно уникати присутності на зображенні відблисків, що виникають при використанні одного високоінтенсивного джерела освітлення.

Система розпізнавання зображень може включати наступні модулі:

— модуль побудови моделі об'єкта, що включає пошук координат на зображенні, локалізацію інформативної області, попередню обробку і нормалізацію;

— модуль підтвердження об'єкта (алгоритми підтвердження об'єкта шляхом відстеження можливої подачі на вхід системи розпізнавання фотографій або відеозаписів раніше зареєстрованого номерного знаку);

— модуль обчислення інформативних ознак;

— модуль зіставлення інформативних ознак вихідного зображення з еталонним (приховані марковські моделі, ІНС, різні метрики, такі, як хіквадрат, коефіцієнти кореляції).

Нижче представлена формальна постановка задачі розпізнавання зображення [10].

Нехай вся множина зображень Ω0, Ω1, ..., ΩL—1 ділиться на L класів. Для кожного зображення обчислюється вектор ознак, що складається з K значень:

$$\rho = (\rho_0, \rho_1 \dots, \rho_{k-1})^T,$$

де T — оператор транспонування.

Ідентифікація — прийняття рішення про співвіднесення деякого вхідного вектора ознак $\bar{\rho}$ (зображення особи) певного класу ΩL. Вхідний об'єкт вважається, що належить класу ΩL, якщо значення функції відстані (або функції схожості) d (ρl, $\bar{\rho}$) для вектора ознак $\bar{\rho}$ є найбільшою:

$$d(\rho_l, \rho) > d\big(\rho_j, \rho\big), l \neq j, j = 0{,}1 \dots L - 1,$$

Додатково слід обчислювати випадки помилкового співвіднесення вхідних векторів ознак $\bar{\rho}$, відсутніх в БД.

Для цього значення функції відстані d (ρl, $\bar{\rho}$) має перевищувати заздалегідь обчислене порогове значення τROC:

$$d(\rho_l, \rho) \geq \tau_{ROC}.$$

Для початку процесу розпізнавання необхідно визначити координати на зображенні. Процес виявлення є першим кроком системи розпізнавання. Цей процес визначає продуктивність системи, так як вимагає більшої кількості процесорного часу. Пошук координат об'єкта на зображенні є трудомістким завданням через низку факторів:

— детектор використовується з різним освітленням і розташування об'єкта;

— складний фон створює додаткові труднощі для виявлення знаку;

— часткове перекриття створює складності для алгоритмів пошуку, тому що тільки частину об'єкта видно для обробки.

Для вирішення завдання пошуку координат номерного знаку на зображенні багато дослідників запропонували різні підходи. Існує чотири групи методів виявлення особи.

Математичний вираз для визначення інтегрального зображення представлено нижче:

$$ii(x, y) = \sum_{i=0}^{xy} i(x', y'),$$

де ii (x, y) — значення і—го елемента інтегрального зображення з наступними координатами (x, y);

i $(x\,', y')$ — інтенсивність пікселя розглянутого зображення з координатами $(x\,', y')$.

Інтегральне зображення обчислюється незалежно від розміру та місцезнаходження.

Ознаки fi поміщають на зображення, сума інтенсивності пікселів, що лежать в білих областях, віднімається від суми інтенсивностей пікселів, що лежать в чорних областях:

$$f_i = \sum_{i=0}^{yx} r_i, white - \sum_{i=0}^{yx} r_i, black,$$

Алгоритм AdaBoost сприяє збільшенню ефективності класифікації, шляхом об'єднання слабких класифікаторів та формування сильного класифікатора. Нижче представлений алгоритм AdaBoost:

1. Проводиться пошук «найкращого» слабкого класифікатора. Вираз для цього класифікатора з граничним значенням τc має вигляд:

$$\omega = \begin{cases} 1, f_i \geq \tau_c \\ -1, f_i < \tau_c \end{cases}.$$

2. Обчислюється «найкращий» сильний класифікатор з фіксованим числом слабких класифікаторів. Вираз для сильного класифікатора виглядає наступним чином:

$$\omega = \begin{cases} 1, \sum_{c=1}^{C} \alpha_c \omega_c \geq \frac{1}{2} \sum_{c=1}^{C} \alpha_c \\ 0, \sum_{c=1}^{C} \alpha_c \omega_c < \frac{1}{2} \sum_{c=1}^{C} \alpha_c \end{cases},$$

де wc — слабкий класифікатор;

ac, βc — вагові коефіцієнти слабкого класифікатора;

c — номер поточного слабкого класифікатора,

C — кількість слабких класифікаторів.

Ідея каскадів класифікаторів полягає в побудові менших і більш ефективних класифікаторів для усунення негативних прикладів і зменшення часу обчислення.

Каскади класифікаторів розраховуються для всіх фрагментів зображення. У разі зупинки фрагмента будь—яким класифікатором фрагмент відкидають і наступні обчислення не виробляються. Якщо фрагмент проходить перший етап, який не потребує трудомістких обчислень, фрагмент потрапляє на наступний етап, в якому виконуються більш трудомісткі обчислення.

Для пошуку координат на зображенні потрібно пройти всі класифікатори в каскаді [10].

Метод локалізації інформативної області призначений для швидкої і точної локалізації інформативної області на зображеннях.

Наступна дія після знаходження координат об'єкта — уточнення координат і точна локалізація границь номерного знаку. Вирішення цього завдання запропонували багато дослідників.

Слабка контрастність, різні умови освітлення і сильна деформація не дозволяють використовувати класичні детектори границь. Моделі форми дозволяють отримати точний опис меж об'єкта при наявності приблизних координат його положення.

Спочатку моделі форми застосовувалися для роботи з об'єктами, що мають чітко виражені границі (наприклад, зображення друкованих плат). З розвитком методів побудови моделей стало можливим застосовувати моделі для наступних завдань: розпізнавання образів, аналізу форми різних об'єктів на викладах і т.д.

У 1992 році група під керівництвом Тіма Кутса запропонувала метод побудови статистичних моделей форми (Statistical shape models). Статистична модель форми містить наступну інформацію: взаємне розташування границь (контурів), розташування точок і інформацію про форму.

Статистична модель форми несприйнятлива до відмінностей в масштабі, зміни ракурсу і зсувів. Статистична модель форми будується по ряду розмічених зображень. Мітки на зображенні слід вибирати за такими відповідностям характерних точок:

— кути;

— «Т» — образні перетини ліній;

— точки вздовж кордонів об'єкта не настільки виражені, як попередні, але знаходяться на великій відстані між знайденими точками.

У зв'язку зі складністю пошуку потрібної координати, мітки на зображенні наносяться в ручному (самим експертом) або в напівавтоматичному режимі (експерту необхідно уточнювати координати міток після використання будь—якого алгоритму пошуку міток).

Перш ніж зібрати статистичну модель, слід все дані привести в одну систему координат. Для компенсації невідповідності в масштабі, ракурсі і зміщення між

мітками необхідно якомога краще зіставити між собою і нормалізувати положення всіх форм. Класичним рішенням вирівнювання є «Прокрустов аналіз» [11].

Безліч міток формується у вигляді матриці, де положення точок без урахування їх взаємного розташування. Щоб обчислити взаємне розташування, слід застосувати метод головних компонент. Для статистичної моделі форми скористаємося наступним виразом:

$$s = s_0 + P_s b_s,$$

де s_0 — усереднена по всіх наборах навчальної вибірки (усереднена форма) форма;

Ps — матриця головних компонент;

bs — параметри моделі форми.

Вираз для статистичної моделі форми демонструє, що модель форми можна виразити як суму усередненої форми s_0 з сукупністю власних форм, що знаходяться в матриці P_s.

При зміні вектора параметрів b_s виробляються різного роду деформації для приведення форми до вихідного зображення.

2.2 Огляд систем розпізнавання зображень номерних знаків

Розглянемо розповсюджені системи розпізнавання номерних знаків.

1. HikVision

Рішення від найбільшої світової компанії лідера ринку систем відеоспостереження і систем безпеки. Розпізнавання автомобільних номерів підтримують всі камери 4-ої серії DS-2CD4xxx Smart-IP.

Розробник: Hikvision Digital Technology.

Розпізнавання автономерів від компанії HikVison можна використовувати у трьох різних комплектаціях.

Перший варіант – використання тільки камери за допомогою браузера проходить підключення до камери і створюється база фіксованого набору, при проїзді автомобіля камера буде самостійно управляти шлагбаумом, якщо

автомобіль в білому списку - то відкривати, якщо такого номера немає тоді залишить закритим.

Особливість даного варіанту в тому, що дані при проїзді, наприклад час або напрямок проїзду не зберігаються, а значить немає можливості наприклад встановити хто і коли проїжджав через шлагбаум або побудувати звіти.

До недоліків даної системи розпізнавання можна віднести те, що для формування "чорних" і "білих" списків автомобільних номерів доведеться виконувати всі дії на кожній камері. Якщо камер багато, то це може бути досить тривалий процес.

Смарт камери HikVision підтримують розпізнавання автомобільних номерів

- 2Мп Smart IP-камера DS-2CD4025FWD-AP;
- 2Мп купольна Smart IP-камера DS-2CD4125FWD-IZ;
- 3Мп купольна Smart IP-камера DS-2CD4135FWD-IZ.

Другий варіант. Розпізнавання номера все також відбувається на борту камери, але розпізнані дані камера відправляє на смарт відеореєстратор, де ведеться база даних зі статистикою всіх проїздів. База даних зі списками "чорних" і "білих" автомобільних номерів формується один раз в програмному інтерфейсі смарт відеореєстратора.

2. Axis Завдяки відкритій платформі Axis Communications - ACAP, сторонні розробники можуть розробляти програми для установки їх безпосередньо на IP камеру. Саме таким чином реалізована можливість розпізнавання автомобільних номерів в камерах Axis.

Розробник програмного забезпечення для розпізнавання автомобільних номерів компанія FF Group, розробила додаток, який може бути встановлено на камеру Axis. На даний момент адаптовано для країн Євросоюзу, СНД, Ізраїлю та Туреччини.

3. Трал-Паркінг Система розпізнавання автомобільних номерів побудована на базі малогабаритних готових модулів "Трал-Паркінг 2", що складаються з аналогової відеокамери і контролера, який виробляє обробку зображення,

перевірку автомобільних номерів і відкриття виконавчих пристроїв, підключених до його релейних виходів.

Існує два типи реалізації модулів - можна придбати готовий виріб в герметичному корпусі з рівнем захисту IP66, або просто купити контролер з камерою для установки.

Програма розпізнавання записана в пам'ять контролера, саме ж пристрій підключається до комп'ютера по TCP \ IP протоколу через інтерфейс NetCore Паркінг, завдяки чому можна здійснювати перегляд подій і налаштування модулів розпізнавання в реальному часі.

Працювати контролери можуть самостійно, для цього в ньому є USB-порт для підключення зовнішнього пристрою, що запам'ятовує, на якому зберігається база номерів і записуються події проїзду. Їх кількість в загальній системі може бути будь-якою, але варто пам'ятати, що для реалізації онлайн перегляду і налаштування всі вони повинні бути підключені до однієї локальної мережі.

Події проїзду містять наступну інформацію: фотографію автомобіля з його державним реєстраційним знаком, час проїзду, напрямок проїзду, факт розпізнавання номера і його приналежність до певної групи.

Робоча станція використовується для перегляду відео з точок проїзду, редагування списку номерів автомобілів, передачі їх в пам'ять контролерів і для зберігання архіву подій, і ніякого іншого способу роботи з модулями розпізнавання немає. Система розпізнає з ймовірністю до 92%.

4. НомерОк. Програмно-апаратний комплекс «НомерОк» призначений для розпізнавання автомобільних номерів і управління виконавчими пристроями.

Модуль дозволяє з імовірністю до 97% розпізнавати автомобільні державні реєстраційні знаки Російської Федерації, Республіки Білорусь, України, Ізраїлю та більшості європейських країн.

Модуль працює як з IP, так і з аналоговими відеокамерами, управління зовнішніми пристроями здійснюється за допомогою релейних блоків-контролерів «Барбос» і ICP CON PET-7060, причому контролери можуть не тільки видавати

керуючі сигнали, але і приймати їх від інших пристроїв (фотоелементи , індукційні петлі та інші пристрої, тип вихідного сигналу яких буде зрозумілий контролерам).

Інтерфейс роботи з контролерами вбудований в основне програмне забезпечення. Кількість одночасно підключених до модулю розпізнавання відеокамер обмежена програмно до 8.

У модулі реалізовано два варіанти роботи з базою даних:

- сервер бази даних SQLite - база даних і модуль розпізнавання встановлені на одному локальному комп'ютері, що має на увазі автономну роботу;

- сервер бази даних Firebird - кілька терміналів с модулем розпізнавання працюють з однією базою, яка може зберігатися на будь-якому з них, при цьому окремої клієнтської частини немає, тобто для віддаленого адміністрування потрібна установка додаткового ПЗ " НомерОк ". У такій версії зв'язок з сервером бази даних здійснюється постійно. Кількість терміналів з модулем розпізнавання, що працюють з однією базою даних, необмежена.

У модулі існує налаштування тривожної події для окремого номера або групи номерів. Тривожною подією може бути:

• текстова інформація будь-якого змісту (затримати, додивитися, пропустити і т.д.);

• звукові сигнали (на кожну подію можна налаштувати свій звуковий файл);

• світлова індикація (до контролера підключаються сигнальні лампи або світлодіоди);

• включення будь-якого зовнішнього пристрою за допомогою релейних виходів контролерів.

Інтеграція з СКУД. СКУД Gate - сервер розпізнавання номерів з ПО номерок v.2 самостійно не приймає ніяких рішень, а тільки розпізнає номер автомобіля і передає розпізнаний номер, як ідентифікатор, безпосередньо в контролер Gate-8000 Авто.

Таким чином, сервер розпізнавання стає для контролера Gate зчитувачем ідентифікаторів. При цьому вся робота системи доступу проводиться в штатному режимі за типовими сценаріями і принципам класичної СКУД.

Автомобільний номер використовується в системі доступу як самостійний ідентифікатор, причому не в кодованому вигляді, а безпосередньо в явному вигляді автомобільного номера. Це забезпечує значну зручність і користувачам, і службі експлуатації при занесенні ідентифікаторів в базу, при аналізі подій доступу і при формуванні необхідних звітів.

У модулі передбачено формування звітів:

Загальні звіти - всі події розпізнавання, згенеровані за обраними фільтрам:

- За періодом часу

- По групі або окремому номеру;

- По каналах і по зонам розпізнавання;

- За номером або частиною номера;

- У напрямку руху;

- За описом;

2. Консолідовані звіти:

- Режим «Розпізнавання». Сумарна кількість машин, згрупованих по каналах / зонам і по напрямку руху

- Режим «КПП». Сумарна кількість машин, згрупованих у напрямку руху і по доступу.

Для зручності дані звітів можуть бути представлені в Excel форматі.

5. iPera EX-LPR. Модуль розпізнавання автомобільних номерів EX-LPR є спільною розробкою фахівців компаній iPera і Exacq Technologies Inc., є клієнтським додатком системи відеоспостереження ExacqVision і призначений для автоматичного розпізнавання всіх автомобільних номерів, які потрапили в поле зору відеокамери, і їх реєстрації.

Для роботи модуля необхідно встановлене ПЗ відеоспостереження exacqVision, для невеликої системи все програмне забезпечення може встановлюватися на один комп'ютер.

Особливістю цього модуля на даний час є те, що пошук автомобільних номерів відбувається по всьому об'єму кадру, а в конкретній його області, що задається при налаштуванні модулів розпізнавання інших виробників - це вносить

свої корективи в вимоги до обчислювальної потужності використовуваного комп'ютерного обладнання.

Кількість камер, одночасно підключених до одного модуля розпізнавання, обмежена тільки апаратними можливостями використовуваного обладнання, для зручності підбору обладнання з урахуванням збереження достатньої швидкості роботи модуля розробники рекомендують використовувати на один канал розпізнавання одне ядро процесора (робоча частота буде залежати від вирішення використовуваної камери).

Для управління виконавчими пристроями в разі автономної роботи використовуються поки тільки тривожні виходи відеокамер, проте модуль має більш широкі інтеграційні можливості.

Модуль розпізнавання EX-LPR є повноцінним сервер-клієнт додатком на базі сервера бази даних MySQL. Зберігання бази даних може здійснюватися на будь-якому комп'ютері або сервері, на якому встановлений сервер MySQL і до якого є доступ по мережі.

Всі клієнтські підключення безкоштовні і не мають обмежень за їх кількістю. За допомогою такого підключення можна переглядати події, налаштовувати систему, редагувати списки номерів, створювати звіти.

Другий тип клієнтського підключення, реалізований розробниками - веб-інтерфейс.

6. Система CVS Авто розроблена фахівцями компанії «Нові Технології», є клієнтським додатком для основної програми системи відеоспостереження CVSCenter і призначена для автоматичного розпізнавання і фіксації автомобільних номерів, які потрапили в поле зору відеокамери. Для роботи модуля потрібно встановлене ПЗ сервера відеоспостереження, в разі необхідності весь комплект програм може встановлюватися на один комп'ютер.

Кількість камер розпізнавання до робочої станції обмежена в разі встановленого ПО CVS Авто до 4, якщо використовується CVS Авто + - їх кількість може досягати 8, але на одному комп'ютері можна запустити кілька копій CVS Авто+.

Програмою підтримується робота як з IP, так і з аналоговими відеокамерами.

Для управління зовнішніми пристроями компанією розробником представлений контролер CVS-DIO, з його допомогою можна також отримувати сигнали з датчиків або інших зовнішніх пристроїв для реалізації складних алгоритмів роботи.

У модулі передбачена можливість ручного коригування номера автомобіля, факт якої відобразиться в журналі подій, як коригування.

Реакцію системи можна налаштувати на окремий номер, групу номерів і тип номерів (шаблон).

Реакцією системи можуть бути:

- текстова інформація будь-якого змісту (затримати, додивитися, пропустити і т.д.);

- звукові сигнали (кожній події можна налаштувати окремий звуковий файл);

- світлова індикація (до контролера підключаються сигнальні лампи або світлодіоди);

- включення будь-якого зовнішнього пристрою.

У модулі реалізована можливість ідентифікації транспортного засобу - при внесенні номера автомобіля в базу даних додається його фото (можна використовувати один знімок). Додаткова текстова інформація про автомобіль може містити модель, колір автомобіля, ім'я власника, контактну інформацію.

У процесі розпізнавання ця інформація разом зі знімком може відображатися на моніторі оператора.

У модулі реалізовано два варіанти роботи з базою даних:

- варіант 1 - (версія CVS-Auto) база даних і модуль розпізнавання встановлені на одному локальному комп'ютері, що має на увазі автономну роботу без можливості клієнтських підключень;

- варіант 2 - кілька терміналів з модулем розпізнавання працюють з однією базою, яка може зберігатися на будь-якому з них або на іншому зовнішньому сервері (версія CVS-Auto +).

У такій версії зв'язок з сервером бази даних здійснюється постійно. Кількість терміналів з модулем розпізнавання в мережі так само як і клієнтських підключень для оператора бази даних необмежена.

Кількість клієнтських підключень не обмежена і всі вони безкоштовні.

7. ATAPY ANPR SDK – спеціалізований програмний продукт з технологією автоматичного розпізнавання номера автомобіля.

Призначений для системних інтеграторів і розробників прикладних програм, які бажають включити технологію автоматичного розпізнавання номера транспортного засобу в своїх додатках. Система ATAPY ANPR SDK має високу точність розпізнавання, бо працює на одному з найбільш точних продуктів в галузі OCR – движку OCR, розробленому ABBYY Software House. Реалізована функція автоматичної геометричної корекції перекосу/нахилу (до 30°).

8. АВТОМАРШАЛ – система розпізнавання автомобільних номерів, що має 12 конфігурацій (автомийка, парковка, склад, КПП та ін.), які дозволяють легко налаштувати систему під поставлену задачу. Має зручний програмний помічник для швидкого налаштування, також підтримується інтеграція з різноманітним обладнанням (шлагбауми, світлофори, ваги та ін).

ВИСНОВОК ДО РОЗДІЛУ 2

В розділі було описано декілька зразків сучасних систем для розпізнавання мереж. Всі вони в певній мірі мають переваги один над іншим в залежності від мети з якою їх застосовують. Алгоритми на основі нейронних мереж можуть класифікувати зображення за темами, а тому придатні розрізняти саме про що зображено в тій чи іншій мірі. Алгоритми без застосування нейронних мереж оперують більш примітивними поняттями як розташування та набори слів, а отже можуть використовуватись для визначення змін у тексті і оцінювати на скільки текст відрізняється від своєї попередньої версії.

РОЗДІЛ 3. РОЗРОБКА АРХІТЕКТУРИ СИСТЕМИ ПОШУКУ

3.1 Застосування бібліотеки комп’ютерного зору

Для обробки зображень одним із оптимальних рішень є застосування системи комп’юетрного зору.

OpenCV - бібліотека алгоритмів комп'ютерного зору, яка створена для обробки зображень і чисельних алгоритмів загального призначення з відкритим кодом. Реалізована на C / C ++, також розробляється для Python, Java, Ruby, Matlab, Lua і т.д.

Може вільно використовуватися в академічних і комерційних цілях - поширюється в умовах ліцензії BSD. Дана бібліотека може надати реалізацію алгоритмів, які були перераховані в попередніх розділах. На подив, OpenCV - це набір типів даних, функцій і класів для обробки зображень алгоритмами розділу «комп'ютерний зір».

Основні модулі бібліотеки представлені на рисунку 3.1.

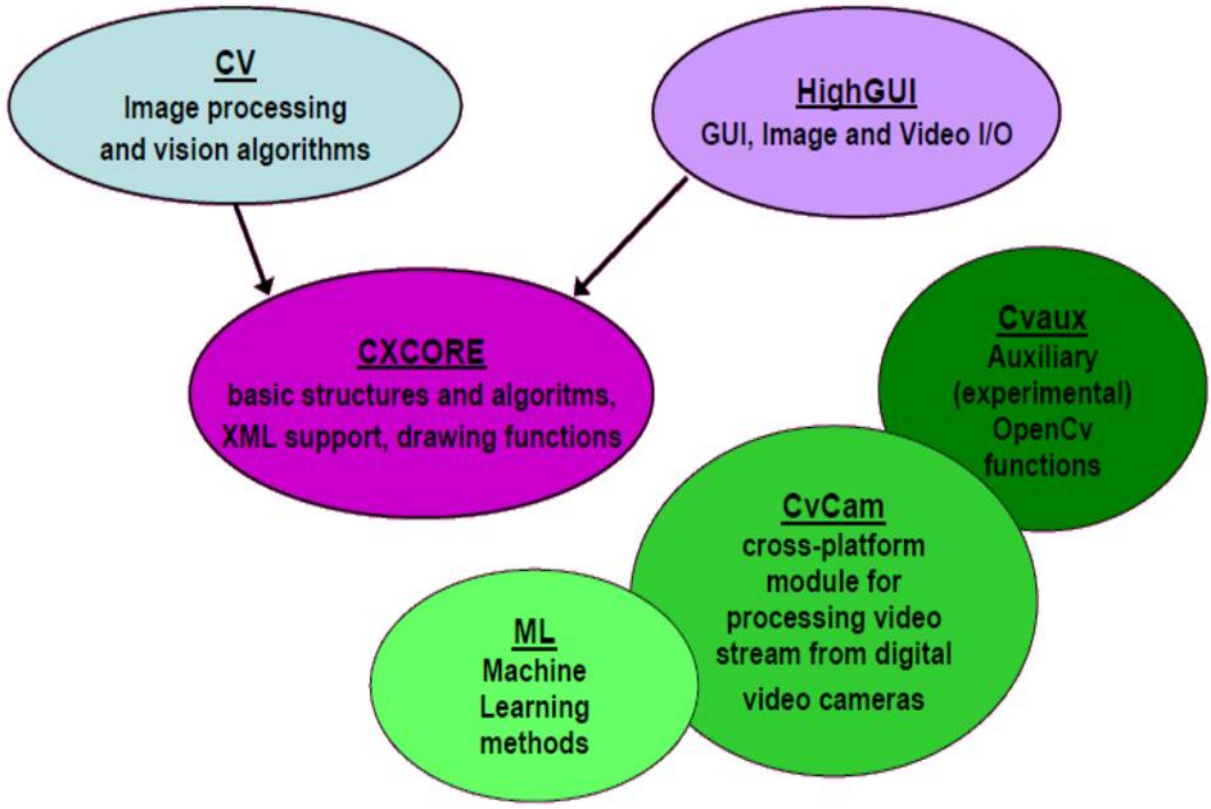


Рисунок 3.1 - Схема основних модулів

Модулі системи:

1. cxcore - ядро містить основну базу, як, базові структури даних і алгоритми:

- Базові операції над матрицями

- Матрична алгебра, математичні функції, генератори випадкових чисел

- Запис / відновлення структур даних в / з XML

- Базові функції векторної графіки, тобто 2D

2. CV - модуль обробки зображень і комп'ютерного зору:

- Базові операції над зображеннями (фільтрація зображення алгоритмами контурів, перетворення зображення в геометричному сенсі, перетворення кольорів і т.д.)

- Аналіз зображень (вибір відмінних ознак, морфологія, пошук контурів, гістограми)

- Аналіз руху, стеження за об'єктами

- Виявлення об'єктів, такі як: особи, руки, ноги, номери машин і т.п.

- Калібрування камер, елементи відновлення просторової структури

3. Highgui - модуль для введення / виведення зображень і відео, створення призначеного для користувача інтерфейсу:

- захоплення відео потоку з камер і з відео файлів, читання / запис статичних зображень

- функції для організації простого UI (всі демо додатки використовують HighGUI)

4. CvCam - захоплення відео:

- дозволяє здійснювати захоплення з відео з цифрових відеокамер (підтримка припинена і в останніх версіях цей модуль відсутній).

3.2 Архітектура системи пошуку

Приклад архітектури системи пошуку представлений на рисунку 3.2.

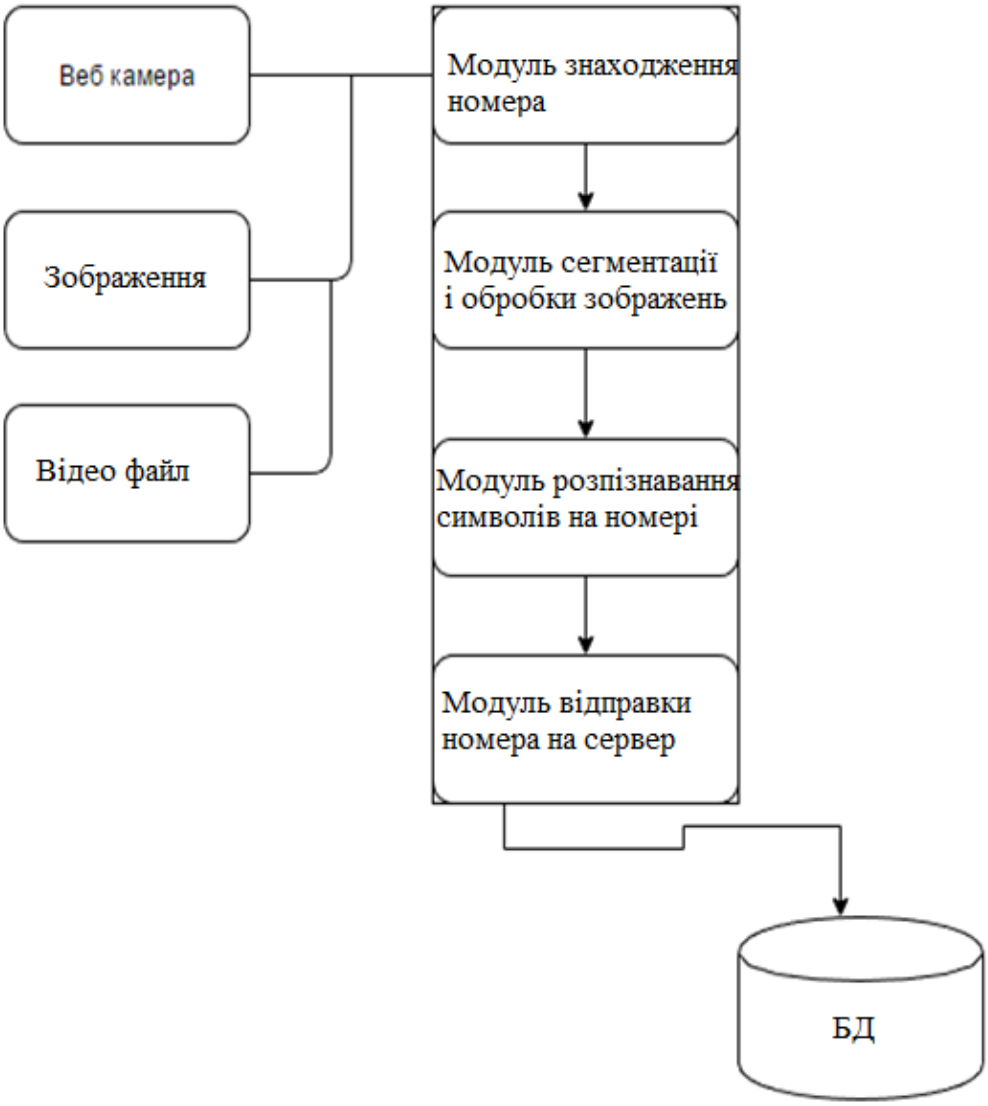


Рисунок 3.2 – Архітектура системи пошуку

Розглянемо детальніше деякі з модулів системи та можливості їх реалізації.

Модуль знаходження номера. Раніше вже було розглянуто алгоритм знаходження номера за допомогою каскаду Хаара, в даному розділі будуть описані етапи навчання класифікатора для модуля знаходження.

Створення класифікатора складаються з декількох етапів:

- Створення папки Good з вибіркою з 500 картинок, на яких є машина і номер, так само в ній повинен бути створений файл, названий так само як директорія, тобто «Good.bat», в ньому дані повинні бути записані як шляхи до зображень і так само потрібно вказати положення даного об'єкту і його розмір, рисунок 3.3.

```
Good \0.bmp  1  0 0 414 148
Good \1.bmp  1  0 0 568 164
Good \....bmp  1  0 0 440 144
Good \N.bmp  1  0 0 590 182
```

Рисунок 3.3 - Структура файлу Good.dat

Створення папки Bad з вибіркою з 600 картинок, на який є всі що завгодно, але немає машин, так само в ній повинен лежати файл, названий так само як директорія, тобто «Bad.dat», в ньому дані повинні бути записані як відносні шляхи до зображень, рисунок 3.4.

```
Bad\1. bmp
Bad\2. bmp
Bad\.... bmp
Bad\N. bmp
```

Рисунок 3.4 - структура файлу Bad.dat

- Перетворення картинок до загального розміру за допомогою програми opencv_createsamples.exe.

- Підрахунок підсумкового каскаду за допомогою програми opencv_traincascade.exe.

Після виконання даних етапів створюється файл у форматі xml, під назвою haarcascade_russian_plate_number.xml структура, якого представлена на рисунку 3.5.

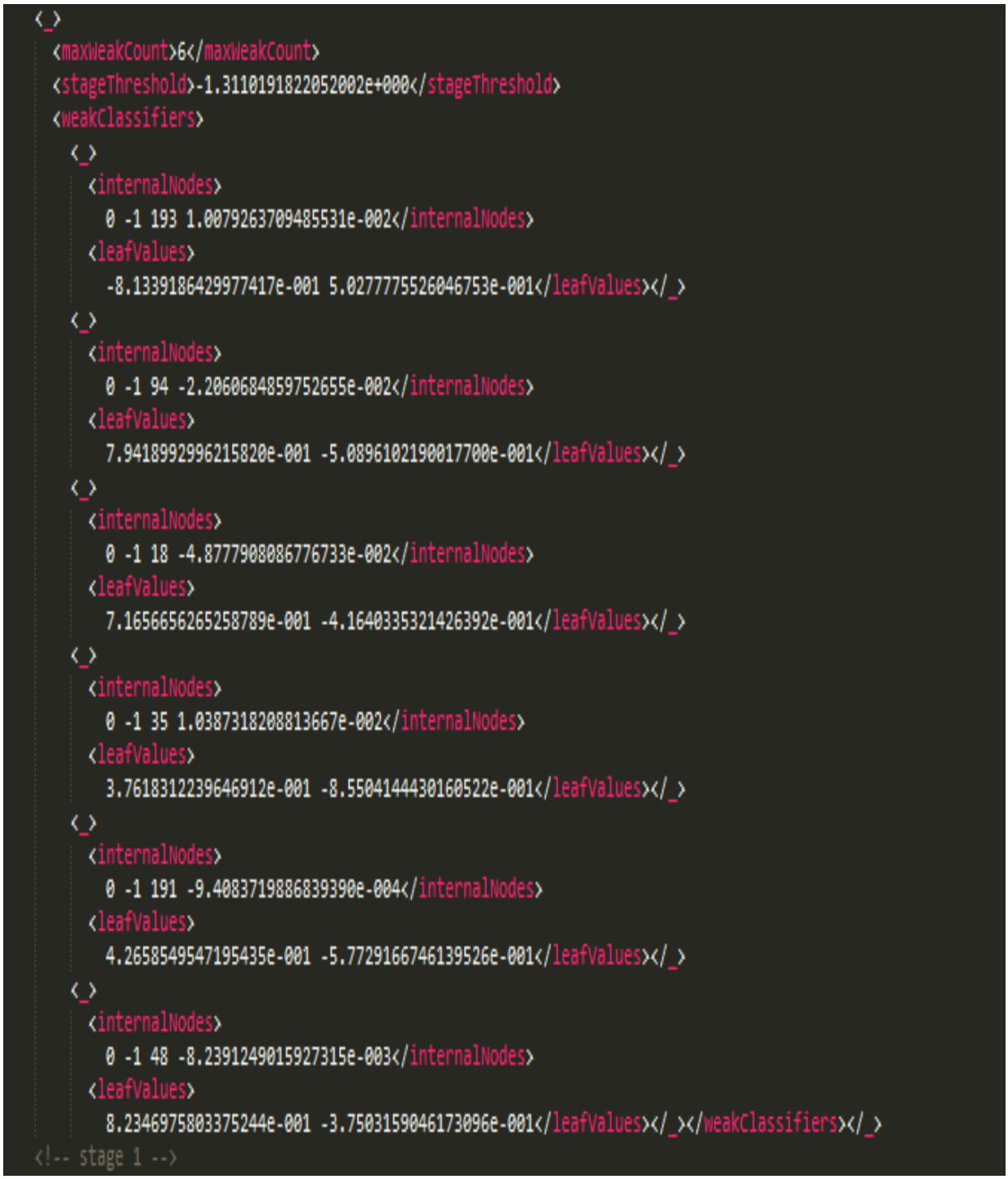


Рисунок 3.5 - Структура файлу haarcascade_russian_plate_number.xml

Модуль сегментації і обробки зображення. Після отримання локалізованого зображення номера, потрібно його нормалізувати для подальшої обробки. Для цього спочатку виконується бінаризація за алгоритмом Оцу, який дозволяє розділити пікселі двох класів («корисні» і «фонові»), розраховуючи такий поріг, щоб дисперсія була мінімальною.

Після бінаризації йде етап обробки. Спочатку потрібно повернути номер, щоб той був розташований горизонтально. Для цього на номері повинні бути знайдені всі горизонтальні лінії і відносно цих ліній необхідно розрахувати кути відхилення.

Після цього потрібно розрахувати середнє арифметичне кутів, і повертати вже даний кут.

Після стабілізації номера в горизонтальне положення, обрізається картинка за алгоритмом контурів. На зображенні перебувають всі елементи зображення, які можна помістити в контур, далі необхідно їх вирізати по найпершому контуру, який знайдеться на зображенні, зазвичай це контур номерного знака, а не його символ,

На цьому етапі обробка зображення закінчується, і можна почати сегментацію символів на ньому.

Алгоритм сегментації був описаний в попередніх розділах. Але необхідно враховувати, щоб в буквах Н,В, цифрах 9,0,8 і т.п. не виділялося нічого зайвого, потрібно перевіряти, не перетинається чи дані цифри і букви з якою-небудь областю, яка всередині їх. Якщо даної операції не робити, то сегментація виділяє зайві елементи (рисунок 3.6).

Рисунок 3.6 - Результат алгоритму без видалення непотрібних областей

При використанні алгоритму видалення областей виходить такий результат, рисунок 3.7.

Рисунок 3.7 - Результат алгоритму с видаленням непотрібних областей

В результаті роботи даного модуля отримується список символів, які потім будуть розпізнаватися.

Рисунок 3.8 - Результат алгоритму сегментації

Або ж при поганому освітленні і бруді, алгоритм бінаризації перетворює картинку так, що символи будуть трохи один з одним перетинатися і окремо символ не можна буде виділити.

Модуль розпізнавання символів потрібен для того, щоб отримати кінцевий результат, тобто розпізнаний номер в текстовому форматі.

В цьому випадку потрібно навчати нейронну мережу, але ця процедура може тривати достатньо довго. Простіше вирізати кожен символ в номерах з усіх зображень, щоб отримати якусь вибірку, так само потрібно кожен символ зображення класифікувати.

Можна використовувати готові шрифти, які використовуються в українських автомобільних номерах такі, як на рисунку 3.9.

Рисунок 3.9 - Шрифт цифр

Щоб швидше навчити дані шрифти на розпізнавання було вирішено використовувати алгоритм класифікації kNN.

kNN або k Nearest Neighbor або k Найближчих Сусідів - це, напевно, один з найпростіших алгоритмів класифікації. Завдяки своїй простоті, він є хорошим початком для знайомства з областю Machine Learning.

Завдання класифікації елементів в машинному навчанні - це завдання віднесення об'єкта до певного класу, який ми заздалегідь визначаємо. Кожен з об'єктів в даній задачі відноситься до N-мірного векторного простору, вимір кожного визначається за своїми критеріями. Припустимо нам потрібно класифікувати човни: вимірами в нашому просторі параметрів будуть величина, співвідношення сторін, колір, довжина, вартість і ін. Випадок класифікації текстів дещо складніше, в основному для них використовуються матриці.

Для навчання нам потрібні об'єкти і певні для них класи. Таке безліч називається навчальною вибіркою, її складання відбувається вручну за власним завданням. У класифікації може бути більше двох класів, кожен з об'єктів мати кілька класів.

Для класифікації кожного з об'єктів з тієї вибірки, яка у нас є, потрібно пройти по порядку кілька етапів:

- Обчислити відстань до кожного з об'єктів навчальної вибірки;

- Відібрати k об'єктів навчальної вибірки, відстань до яких мінімальна;

- Клас об'єкта - це клас, який найчастіше трапляється серед k найближчих сусідів.

Тобто спочатку вирізаємо кожну букву і цифру із зображень шрифтів, наводимо кожен елемент до одного розміру і записуємо масиви в файли: літери окремо, цифри окремо.

Припустимо, це файли char_samples.data, де зберігаються букви і digits_samples.data, який зберігає цифри та має структуру, яка зображена на малюнках нижче.

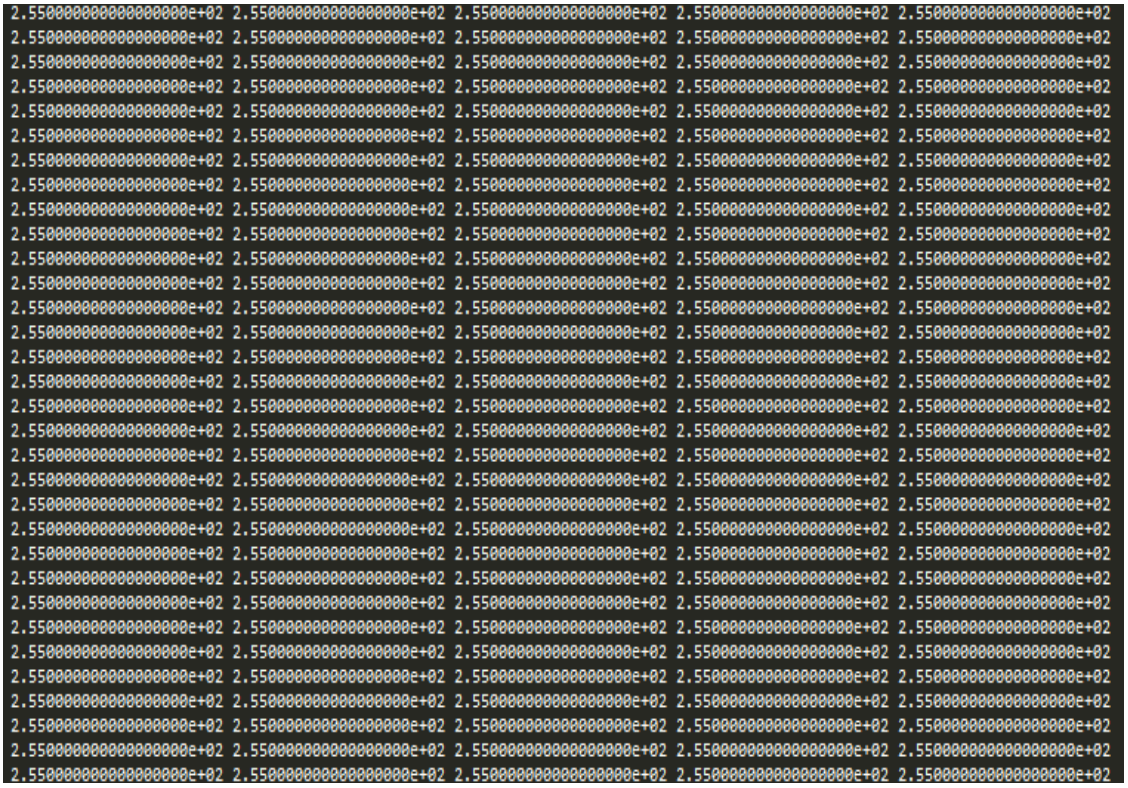


Рисунок 3.10 - Структура файлу char_samples.data

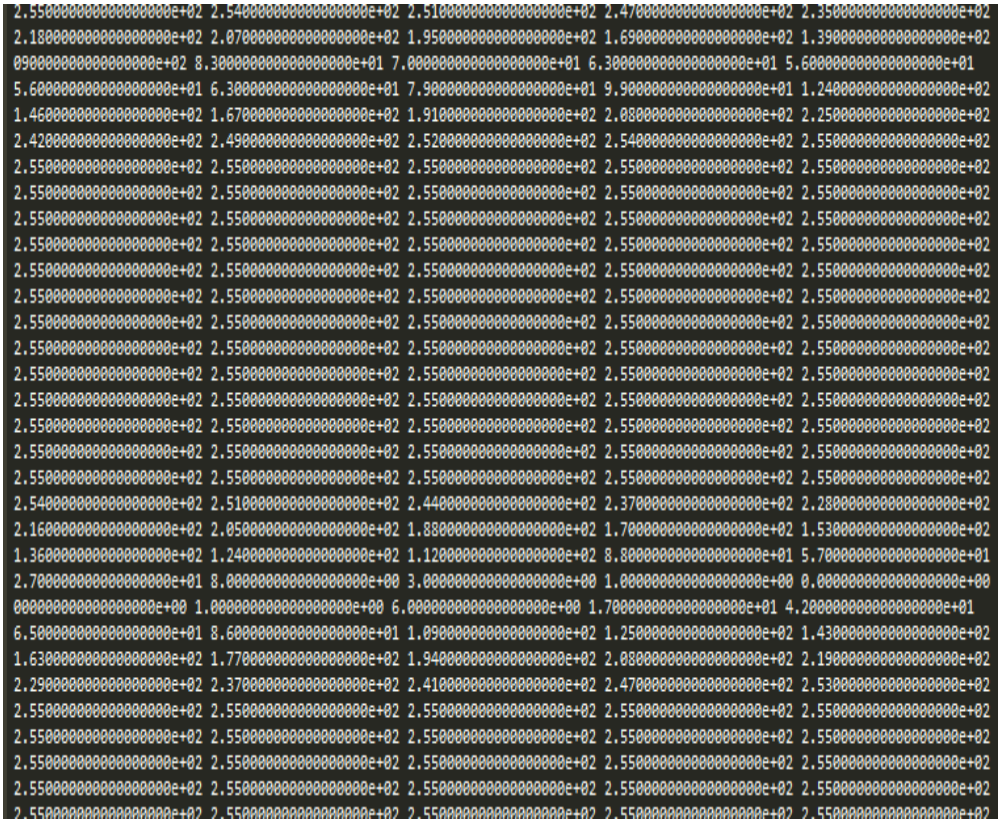


Рисунок 3.11 - Структура файлу digits_samples.data

Далі записуємо результати кожної букви в числовому поданні в файл chars_responses.data, рисунок 3.12.

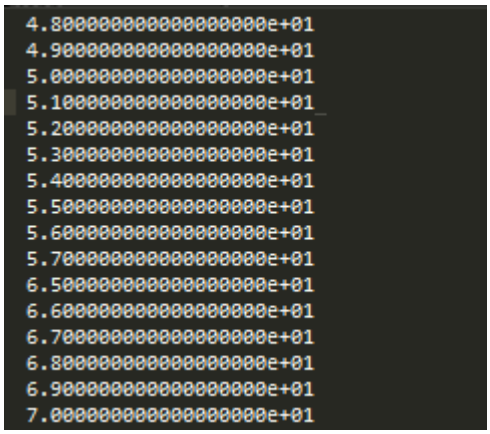


Рисунок 3.12 - Структура файлу chars_responses.data

Теж саме робимо і з цифрами, рисунок 3.13.

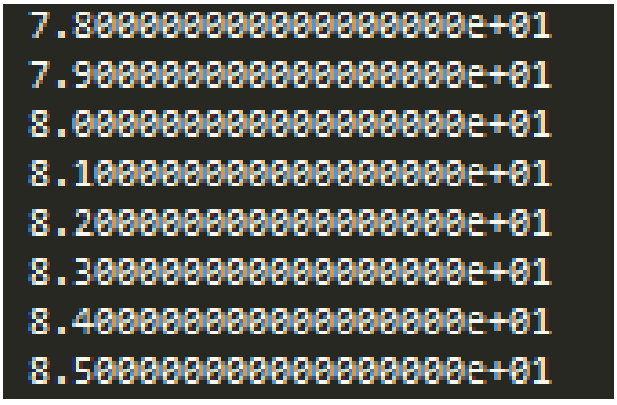


Рисунок 3.13 - Структура файлу digits_responses.data

Далі тренуємо нашу модель і після цього починаємо виконувати алгоритм по знаходженню більш схожого символу на нашому номері.

Алгоритм тренування наступний. Розглянемо навчання мережі даними алгоритмом на прикладі тришарової нейронної мережі, представленої на рисунку 3.14.

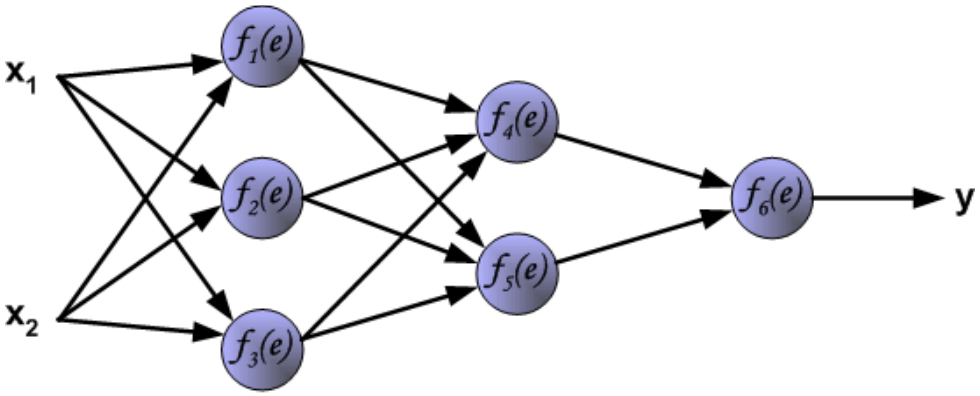


Рисунок 3.14 - Схема багатошарової нейронної мережі

Для навчання нейронної мережі ми повинні підготувати навчальні приклади.

Кожен крок навчання починається з впливу вхідних сигналів з навчальних прикладів. Після цього можна визначити значення вихідних сигналів для всіх нейронів в кожному шарі мережі.

На рисунку 3.15 показано напрямок сигналу в мережі.

$W(X_{mn})$- вага зв'язку між входом X_m і нейроном n у вхідному шарі.

$Y(n)$- вихід нейрона n.

$$y_1 = f_1(w_{(x_1)1}x_1 + w_{(x_2)1}x_2),$$
$$y_2 = f_2(w_{(x_1)2}x_1 + w_{(x2)2}x_2),$$
$$y_3 = f_3(w_{(x_1)3}x_1 + w_{(x_2)3}x_2).$$

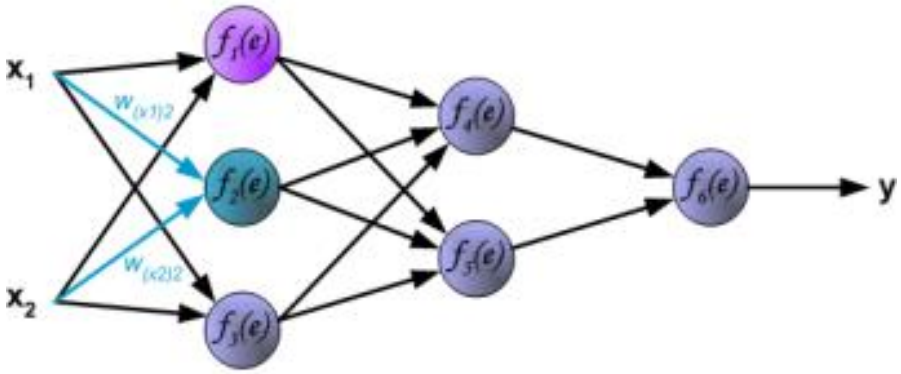


Рисунок 3.15 - Поширення сигналу від входу до 1-го прихованого шару

W_{mn} - ваги зв'язків між виходом нейрона m і входом нейрона n в наступному шарі.

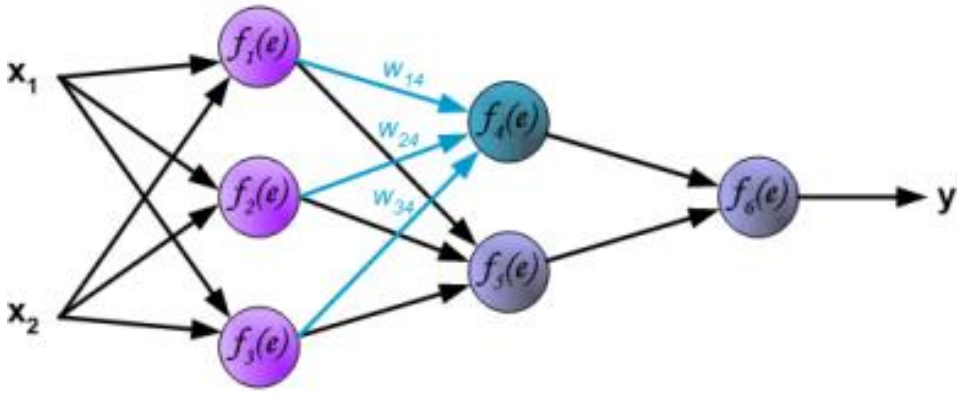


Рисунок 3.16 - Поширення сигналу до 2-го прихованого шару

На рисунку 3.16 показано як поширюється сигнал від першого шару до другого. У такому напрямку обчислюються виходи всіх нейронів, в тому числі і вихідного:

$$y_4 = f_4(w_{14}y_1 + w_{24}y_2 + w_{34}y_3),$$
$$y_5 = f_5(w_{15}y_1 + w_{25}y_2 + w_{35}y_3).$$

Далі йде прохід сигналу через вихідний шар:

$$y = f_6(w_{46}y_4 + w_{56}y_5).$$

Наступний етап алгоритму полягає в наступному: вихідний сигнал y порівнюється з бажаним виходом мережі z.

δ - помилка (різниця) між сигналами z і y: $\delta = z - y$.

На рисунку 3.17 показано розподіл помилки по мережі

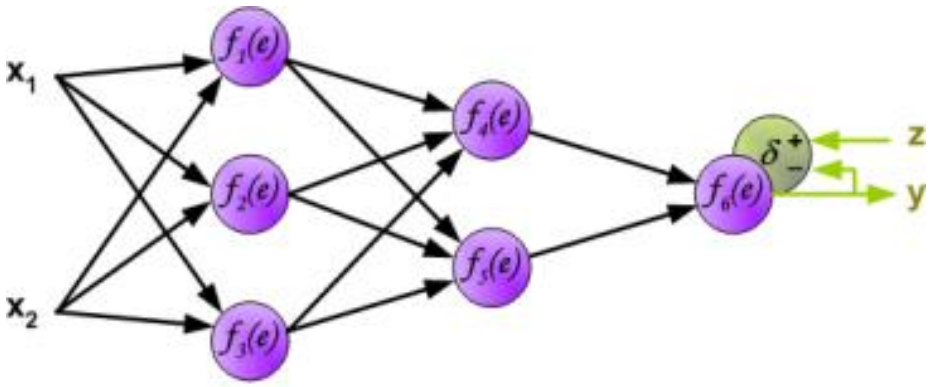


Рисунок 3.17 - Поширення помилки

Далі сигнал δ поширюється в зворотному напрямку на всі нейрони, чиї вихідні сигнали були вхідними для останнього нейрона.

$$\delta_4 = w_{46}\delta,$$
$$\delta_5 = w_{56}\delta.$$

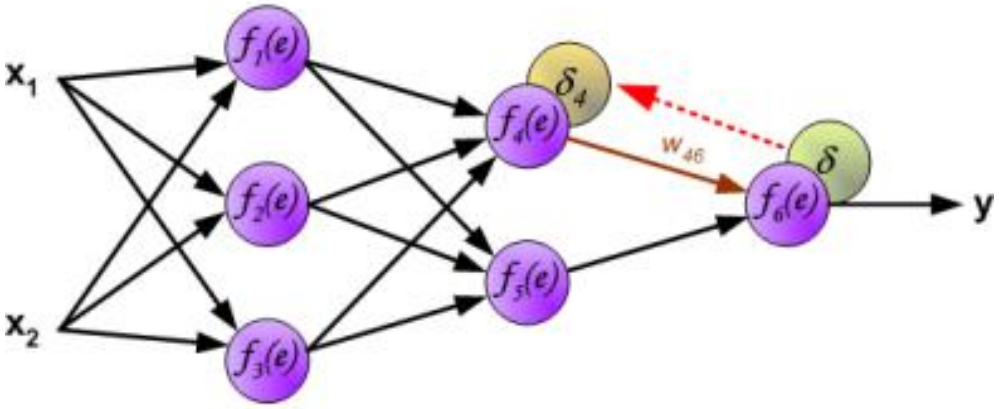


Рисунок 3.18 - Поширення помилки від вихідного шару до 2-го прихованого шару

Вагові коефіцієнти W_{mn} рівні тим же коефіцієнтам, що використовувалися під час обчислення вихідного сигналу y.

Змінюється напрямок потоку даних (сигнали передаються від виходу до входу). Процес повторюється для всіх верств мережі.

Якщо помилка прийшла від декількох нейронів - вона підсумовуються (рисунок 3.19):

$$\delta_1 = w_{14}\delta_4 + w_{15}\delta_5,$$
$$\delta_2 = w_{24}\delta_4 + w_{25}\delta_5,$$
$$\delta_3 = w_{34}\delta_4 + w_{35}\delta_5.$$

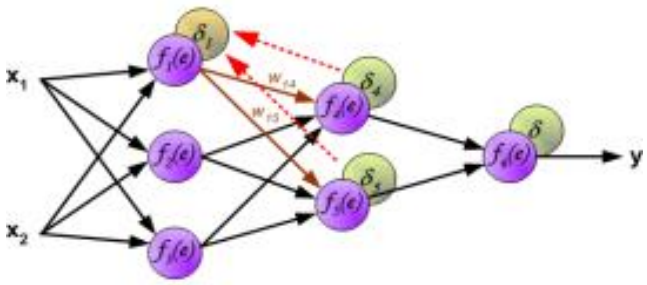


Рисунок 3.19 - Поширення помилки δ в зворотному напрямку в 1-ій прихованому шарі

Існує можливість перенавчання мережі. Весь набір образів, наданих до навчання, буде вивчено мережею, але будь-які інші образи, навіть дуже схожі, можуть бути класифіковані невірно.

Коли обчислюється помилка δ для кожного нейрона - можна скорегувати вагові коефіцієнти кожного нейрона (рисунок 3.20).

$$w'_{(x_1)1} = w_{(x_1)1} + \eta\delta_1 \frac{df_1(e)}{de} x_1,$$

$$w'_{(x_2)1} = w_{(x_2)1} + \eta\delta_1 \frac{df_1(e)}{de} x_2.$$

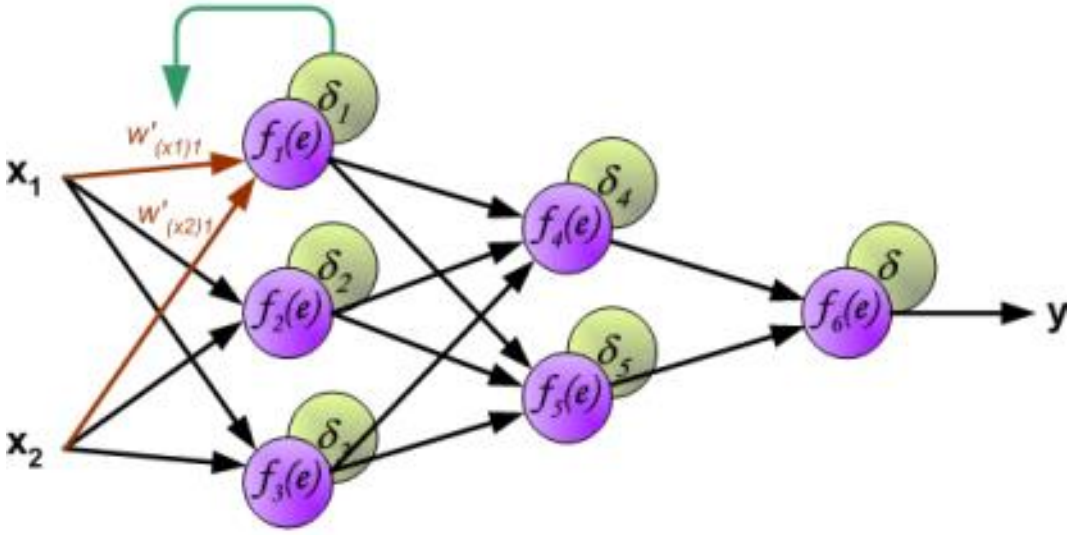


Рисунок 3.20 - Коригування ваг в 1-му шарі

де, $\frac{df(e)}{de}$ - похідна від функції активації нейрона (того, чиї ваги налаштовуються),

η - коефіцієнт, що впливає на швидкість навчання.

$$w'_{(x_1)2} = w_{(x_1)2} + \eta\delta_2 \frac{df_2(e)}{de} x_1,$$

$$w'_{(x_2)2} = w_{(x_2)2} + \eta\delta_2 \frac{df_2(e)}{de} x_2.$$

так само відповідно обчислюємо $w'_{(x_1)3}$ і $w'_{(x_2)3}$

На рисунку 3.21 зображено, в якому напрямку проходить корекція ваг в вихідному шарі.

$$w'_{14} = w_{14} + \eta\delta_4 \frac{df_4(e)}{de} y_1,$$

$$w'_{24} = w_{24} + \eta\delta_4 \frac{df_4(e)}{de} y_2,$$

$$w'_{34} = w_{34} + \eta\delta_4 \frac{df_4(e)}{de} y_3.$$

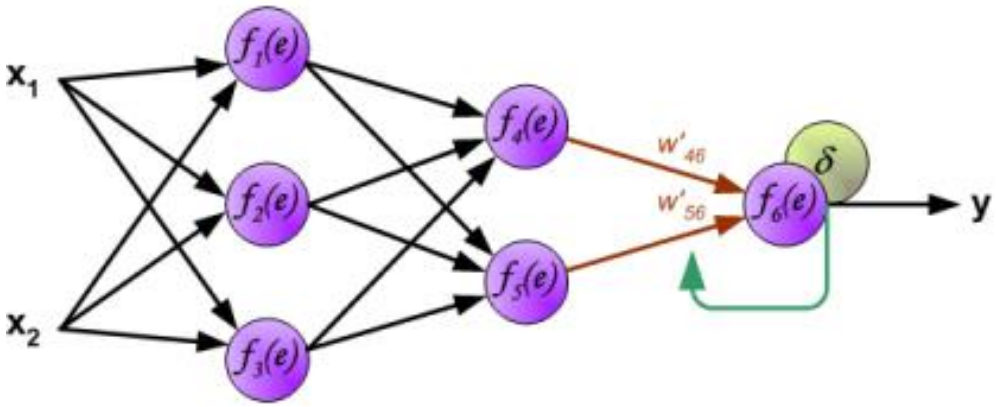


Рисунок 3.21 - Коригування ваг в вихідному шарі

Однак даний алгоритм має ряд недоліків. Серед таких недоліків можна виділити те, що може бути невизначено довгий процес навчання нейронної мережі.

ВИСНОВОК ДО РОЗДІЛУ 3

В даному розділі було описано архітектуру системи пошуку. Для розробки ПЗ було обрано бібліотеку OpenCV, оскільки вона є найзручнішою бібліотекою комп’ютерного зору та ідеально підходить для наших цілей. Мова програмування Python, для якої реалізована дана бібліотека, дозволяє розширювати вже існуючі системи, навіть написані на іншій мові, а тому підходить для розробки додатків під всі системи.

ВИСНОВКИ

В результаті написання дипломної роботи розглянуто систему пошуку зображень транспортних засобів за номерними знаками.

В першому розділі розглянуто методи класифікації зображень. Встановлено, що автоматизація розпізнавання номерів автомобілів – одне з завдань для корпоративних систем безпеки. Дані системи дозволяють істотно підвищити розпізнавання машин і сформувати звітність за багатьма аспектами.

В другом розділі розглянуто зразки та можливості сучасних систем розпізнавання зображень.

В третьому розділі представлена розробка архітектури системи розпізнавання номерних знаків.

Основні модулі розробленої архітектури:

- Модуль локалізації номера і бінаризації;
- Модуль обробки зображення;
- Модуль сегментації;
- Модуль розпізнавання символів.

В результаті написання роботи були виконані наступні задачі:

1. Виконано огляд та аналіз методів і алгоритмів класифікації та пошуку зображень номерних знаків;

2. Розглянуто зразки та можливості сучасних систем розпізнавання зображень;

3. Виконано розробку архітектури системи пошуку.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Автоматическое управление и вычислительная техника. Выпуск 10. Распознавание образов: моногр. . - М.: Машиностроение, 2016**.** - 256 с.

2. Гренандер, У. Лекции по теории образов (Том 1. Синтез образов) / У. Гренандер. - М.: Машиностроение, 2014**.** - 571 с.

3. Фисенко В.Т., Фисенко Т.Ю. — Компьютерная обработка и распознавание изображений. — СПб.:, 2008.

4. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel: Backpropagation Applied to Handwritten Zip Code Recognition, Neural Computation, 1(4):541-551, Winter 1989.

5. T. Xiao, J. Zhang et al., “Error-driven incremental learning in deep convolutional neural network for large-scale image classification,” in International Conference on Multimedia, no. 22. ACM, 2014, pp. 177– 186.

6. Елисеева, И. И. Группировка, корреляция, распознавание образов (статистические методы классификации и измерения связей) / И.И. Елисеева, В.О. Рукавишников. - Москва: РГГУ**,** 2014. - 144 с.

7. Капитонова, Т. А. Нейросетевое моделирование в распознавании образов. Философско-методические аспекты / Т.А. Капитонова. - Москва: РГГУ**,** 2015**.** - 684 с.

8. Уздин, Д. З. Новые меры близости, функции состояний и решающие правила в теории распознавания образов (состояний) / Д.З. Уздин. - М.: МАКС Пресс, 2015. - 110 с.

9. Фукунага, К. Введение в статистическую теорию распознавания образов / К. Фукунага. - М.: Главная редакция физико-математической литературы издательства "Наука", 2013. - 368 с.

10. J. Zhao, M. Mathieu et al., “Stacked what-where auto-encoders,” arXiv preprint arXiv:1506.02351, Jun. 2015. [Online]. Available: https: //arxiv.org/abs/1506.02351v1.

11. Тарасенко-Клятченко О.В., Буц В.В. стаття «Організація багатоетапного методу навчання згорткової нейронної мережі». Журнал «Інтернаука» випуск 6

(березень 2018). [Інтернет-ресурс] Посилання: https://www.inter-nauka.com/issues/2018/6/3624/.

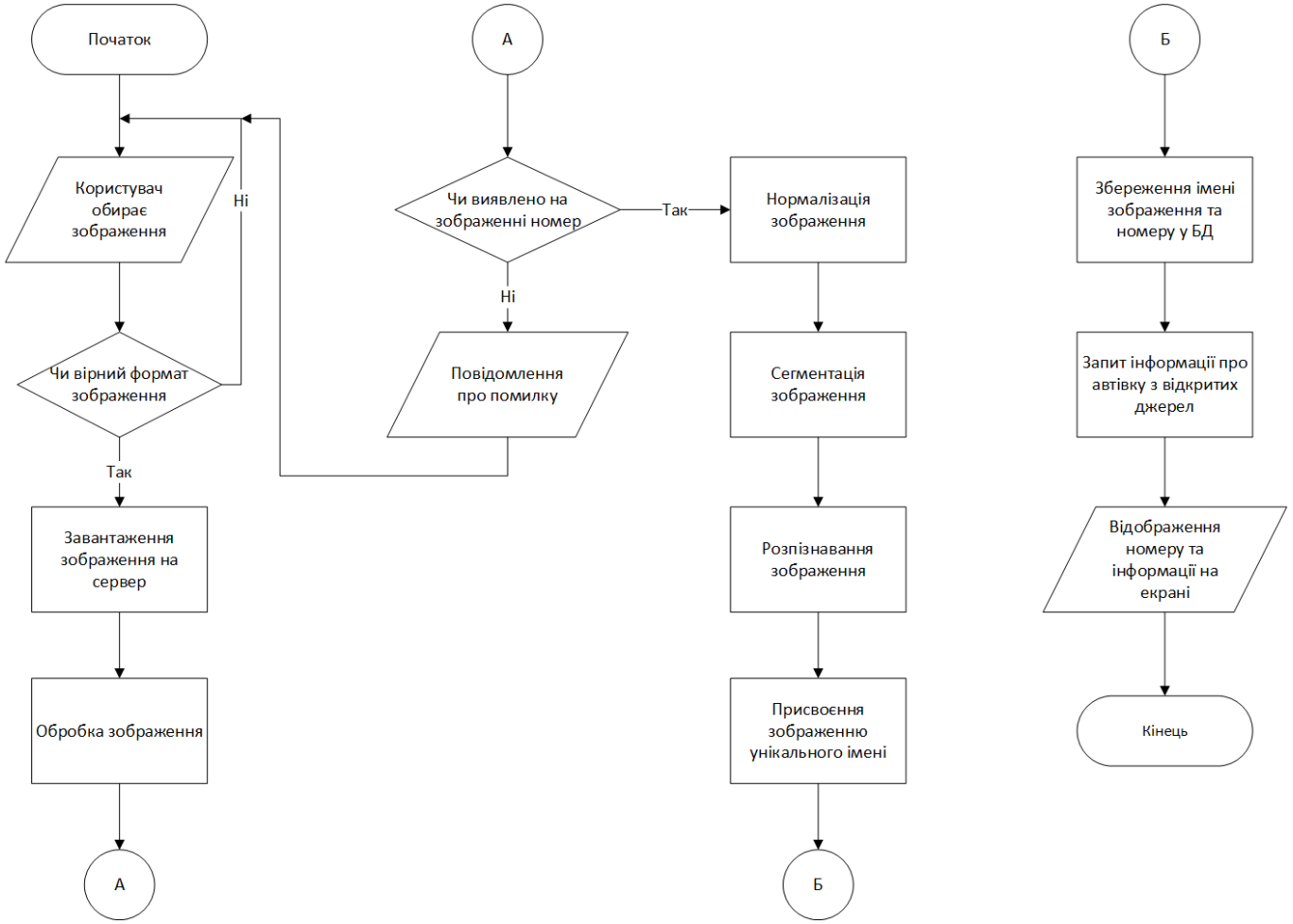
ДОДАТОК А

Система пошуку зображень транспортних засобів за номерними знаками

Принципова схема алгоритму дій користувача

ІАЛЦ.467400.003 Д1

Аркушів 1



Зм.	Арк.	№ документу	Підпис	Дата
Розробив		Костяєв А.О.		
Перевірив		Порєв В.М.		
Н. Контр.		Сімоненко В.П.		
Затверд.				

ІАЛЦ.467400.003 Д1

Система пошуку зображень транспортних засобів за номерними знаками

Літ.	Аркуш	Аркушів
	1	1

НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ, гр. ІП-62

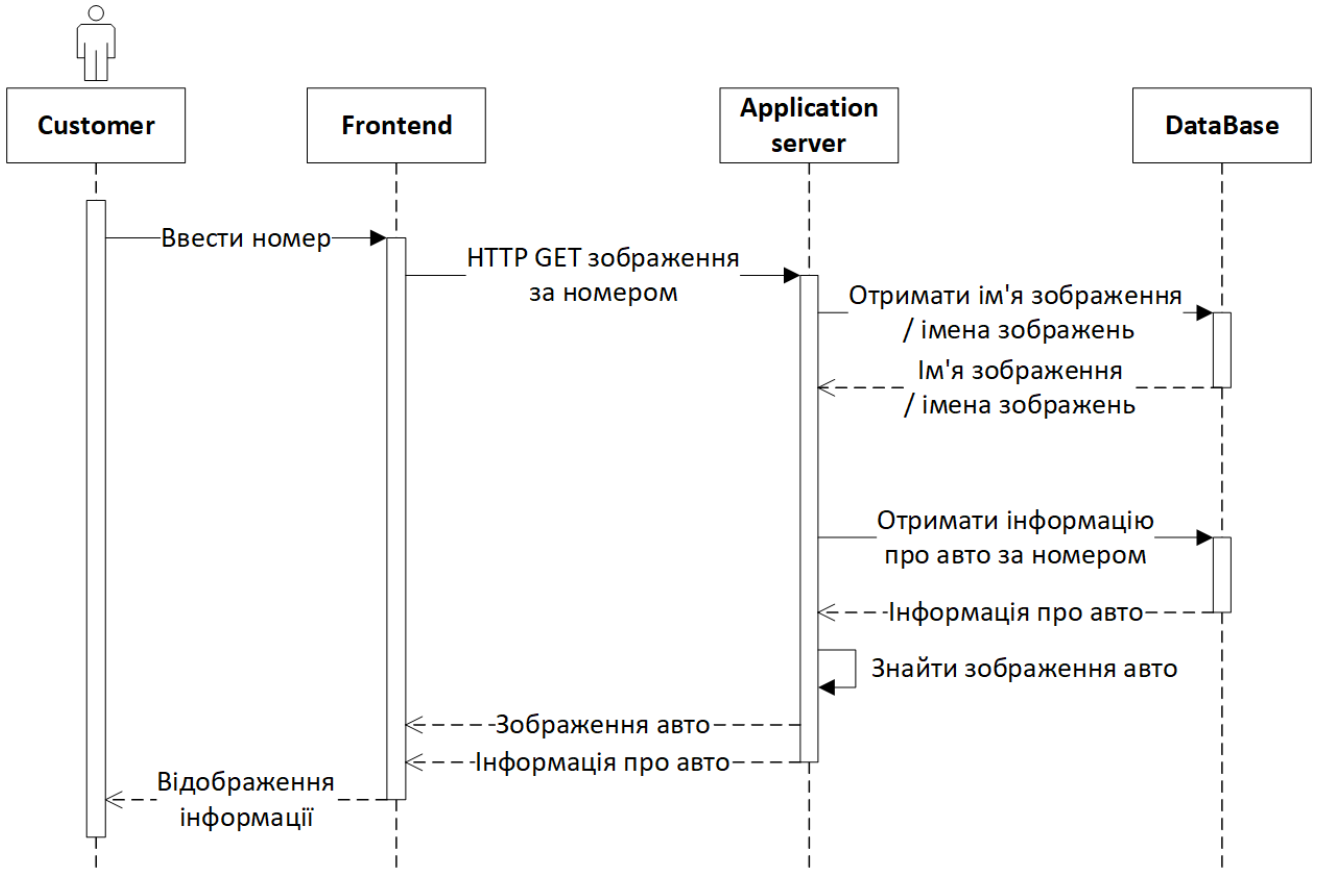
ДОДАТОК Б

Система пошуку зображень транспортних засобів за номерними знаками

Структурна схема

ІАЛЦ.467400.004 Д2

Аркушів 1



Зм.	Арк.	№ документу	Підпис	Дата
Розробив		Костяєв А.О.		
Перевірив		Порєв В.М.		
Н. Контр.		Сімоненко В.П.		
Затверд.				

ІАЛЦ.467400.004 Д2

Система пошуку зображень транспортних засобів за номерними знаками

Лит.	Аркуш	Аркушів
	1	1

НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ, гр. ІП-62

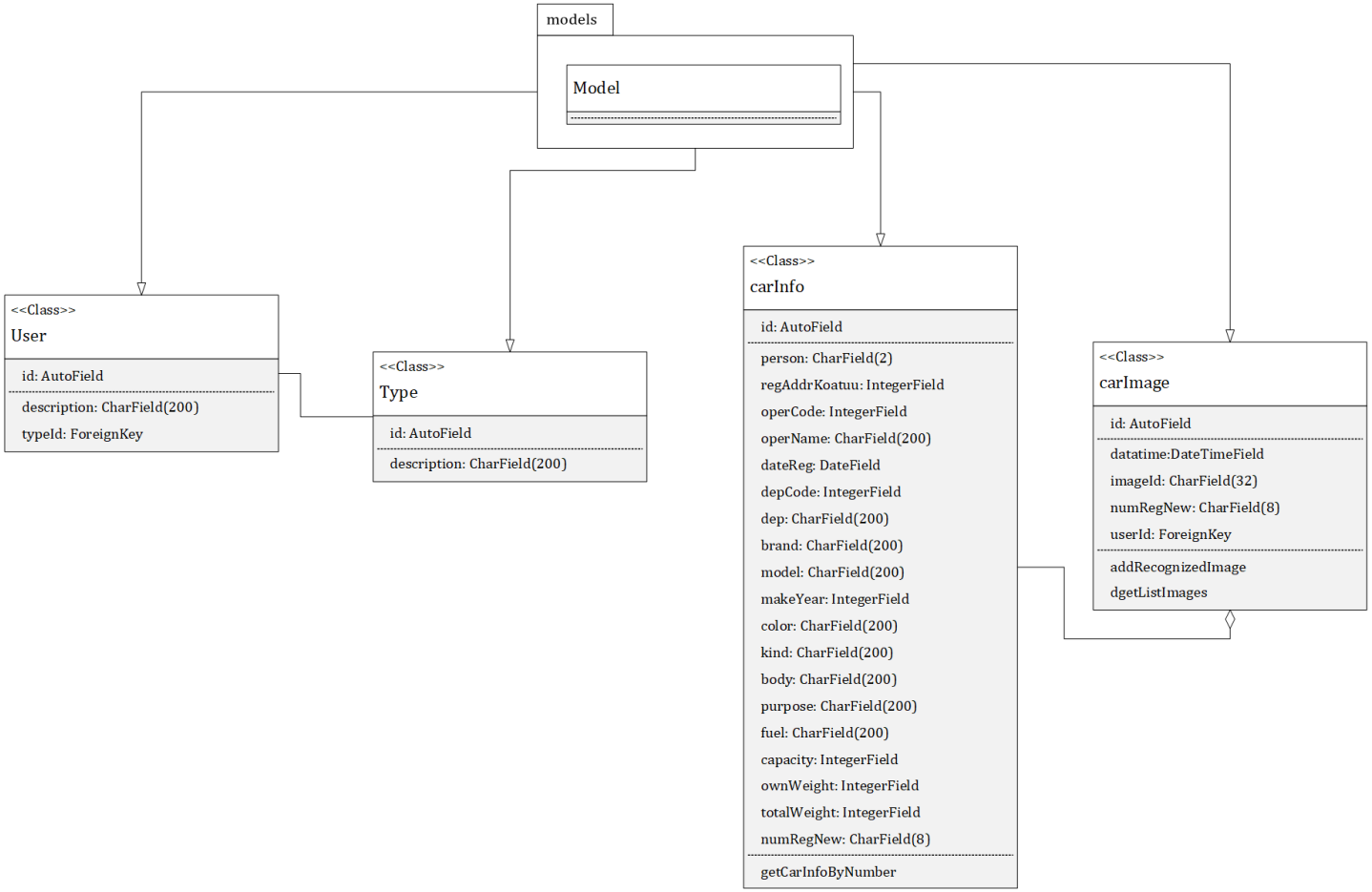
ДОДАТОК В

Система пошуку зображень транспортних засобів за номерними знаками

Функціональная схема

ІАЛЦ.467400.005 ДЗ

Аркушів 1



Зм.	Арк.	№ документу	Підпис	Дата
Розробив		Костяєв А.О.		
Перевірив		Порєв В.М.		
Н. Контр.		Сімоненко В.П.		
Затвердив				

ІАЛЦ.467400.005 ДЗ

Система пошуку зображень транспортних засобів за номерними знаками

Літ.	Аркуш	Аркушів
	1	1

НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ, гр. ІП-62

ДОДАТОК Г

Система пошуку зображень транспортних засобів за номерними знаками

Лістинг частин програми

ІАЛЦ.467400.006 Д4

Аркушів 16

```
import os
import sys
import json
import matplotlib.image as mpimg

# change this property
NOMEROFF_NET_DIR = "/var/www/nomeroff-net/"
MASK_RCNN_DIR = "/var/www/Mask_RCNN/"

MASK_RCNN_LOG_DIR = os.path.join(NOMEROFF_NET_DIR, "logs/")
MASK_RCNN_MODEL_PATH          =          os.path.join(NOMEROFF_NET_DIR,
"models/mask_rcnn_numberplate_0700.h5")
REGION_MODEL_PATH           =           os.path.join(NOMEROFF_NET_DIR,
"models/imagenet_vgg16_np_region_2019_1_18.h5")

sys.path.append(NOMEROFF_NET_DIR)

# Import license plate recognition tools.
from  NomeroffNet  import    filters,  RectDetector,  TextDetector,
RegionDetector, Detector, textPostprocessing

# Initialize npdetector with default configuration file.
nnet = Detector(MASK_RCNN_DIR, MASK_RCNN_LOG_DIR)
# Load weights in keras format.
nnet.loadModel(MASK_RCNN_MODEL_PATH)

# Initialize rect detector with default configuration file.
rectDetector = RectDetector()

# Initialize text detector.
textDetector = TextDetector()

# Initialize numberplate region detector.
regionDetector = RegionDetector()
regionDetector.load(REGION_MODEL_PATH)

img_path = './examples/images/example1.jpeg'
img = mpimg.imread(img_path)
NP = nnet.detect([img])

# Generate image mask.
cv_img_masks = filters.cv_img_mask(NP)

for img_mask in cv_img_masks:
    # Detect points.
    points   =   rectDetector.detect(img_mask,   fixRectangleAngle=1,
outboundWidthOffset=3)

    # Split on zones
    zone = rectDetector.get_cv_zones(img, points)

    # find standart
    regionId = regionDetector.predict(zone)
```

```
    regionName = regionDetector.getLabels(regionId)

    # find text with postprocessing by numberplate region detector
    text = textDetector.detect(zone)
    text = textPostprocessing(text, regionName)
    print('Detected numberplate: "%s" in region
[%s]'%(text,regionName))
    # Detected numberplate: "AC4921CB" in region [eu-ua-2015]
```

Detector.py

```
import sys
import os
import matplotlib.image as mpimg
import tensorflow as tf
import numpy as np
from tensorflow.python.framework import graph_io
from tensorflow.python.tools import freeze_graph
from tensorflow.core.protobuf import saver_pb2
from tensorflow.python.training import saver as saver_lib
from tensorflow.python.framework.graph_util import
convert_variables_to_constants
import keras
import git

config = tf.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.Session(config=config)

from .mcm.mcm import download_latest_model

class Detector:
    def download_mrcnn():
        import git

git.Git("/your/directory/to/clone").clone("git://gitorious.org/git-
python/mainline.git")

    def __init__(self, mask_rcnn_dir=None, log_dir="./logs/",
mask_rcnn_config = None):
        self.MASK_RCNN_DIR = mask_rcnn_dir
        self.LOG_DIR = log_dir

        DEFAULT_MASK_RCNN_CONFIG = {
          "NAME": "numberplate",
          "GPU_COUNT": 1,
          "IMAGES_PER_GPU": 1,
          "NUM_CLASSES": 2,
          "DETECTION_MIN_CONFIDENCE": 0.7,
          "IMAGE_MAX_DIM": 1024, # work ?
          "IMAGE_RESIZE_MODE": "square" # work ?
        }
```

```
        self.NN_MASK_RCNN_CONFIG      =      mask_rcnn_config      or
DEFAULT_MASK_RCNN_CONFIG
        if not mask_rcnn_dir is None:
            sys.path.append(mask_rcnn_dir)

        from .nnmrcnn import InferenceConfig
        self.CONFIG = InferenceConfig(self.NN_MASK_RCNN_CONFIG)

        # for frozen graph
        self.INPUT_NODES  =  ("input_image:0",  "input_image_meta:0",
"input_anchors:0") # 3 elem

        self.OUTPUT_NODES      =      ("mrcnn_detection/Reshape_1:0",
"mrcnn_class/Reshape_1:0",                    "mrcnn_bbox/Reshape:0",
"mrcnn_mask/Reshape_1:0",
                                "ROI/packed_2:0",  "rpn_class/concat:0",
"rpn_bbox/concat:0") # 7 elem

    def loadFrozenModel(self, FROZEN_MODEL_PATH):
        graph_def = tf.GraphDef()
        with tf.gfile.GFile(FROZEN_MODEL_PATH, "rb") as f:
            graph_def.ParseFromString(f.read())

        graph = tf.Graph()
        with graph.as_default():
            self.input_image,                   self.input_image_meta,
self.input_anchors,      self.mrcnn_detection,      self.mrcnn_class,
self.mrcnn_bbox,    self.mrcnn_mask,    self.ROI,    self.rpn_class,
self.rpn_bbox = tf.import_graph_def(
                graph_def,  return_elements  =  self.INPUT_NODES  +
self.OUTPUT_NODES
            )

        sess_config = tf.ConfigProto()
        sess_config.gpu_options.allow_growth = True
        self.sess = tf.Session(graph=graph, config=sess_config)

    @classmethod
    def get_classname(cls):
        return cls.__name__

    def   loadModel(self,   model_path="latest",   verbose   =   0,
mode="inference"):
        if model_path == "latest":
            model_info  =  download_latest_model(self.get_classname(),
"mrcnn")
            model_path = model_info["path"]

        import mrcnn.model as modellib
        self.MODEL       =       modellib.MaskRCNN(mode="inference",
model_dir=self.LOG_DIR, config=self.CONFIG)
        if model_path.split(".")[-1] == "pb":
            self.loadFrozenModel(model_path)
```

```
            self.detect = self.frozen_detect
        else:
            self.MODEL.load_weights(model_path, by_name=True)

    def getKerasModel(self):
        return self.MODEL.keras_model

    def normalize(self, images):
        res = []
        for image in images:
            # delete 4 chanel
            res.append(image[..., :3])
        return res

    def detectFromFile(self, image_paths, verbose = 0):
        images = [mpimg.imread(image_path) for image_path in
image_paths]
        return self.detect(images, verbose=verbose)

    def detect(self, images, verbose = 0):
        return self.MODEL.detect(self.normalize(images),
verbose=verbose)

#    def detect_masks(self, images, verbose = 0):
#       r = self.MODEL.detect(self.normalize(images), verbose=verbose)
#        # loop over of the detected object's bounding boxes and masks
#        for i in range(0, r["rois"].shape[0]):
#           # extract the class ID and mask for the current detection,
then
#            # grab the color to visualize the mask (in BGR format)
#            classID = r["class_ids"][i]
#            mask = r["masks"][:, :, i]
#            color = COLORS[classID][::-1]
#
#            # visualize the pixel-wise mask of the object
#          image = visualize.apply_mask(image, mask, color, alpha=0.5)



    def frozen_detect(self, images, verbose = 0):
        if verbose:
            print(self.CONFIG.BATCH_SIZE)
        assert len(images) == self.CONFIG.BATCH_SIZE, "len(images)
must be equal to BATCH_SIZE"

        # Mold inputs to format expected by the neural network
        molded_images, image_metas, windows =
self.MODEL.mold_inputs(images)

        # Validate image sizes
        # All images in a batch MUST be of the same size
        image_shape = molded_images[0].shape
        for g in molded_images[1:]:
```

```
            assert g.shape == image_shape,\
                "After resizing, all images must have the same size.
Check IMAGE_RESIZE_MODE and image sizes."

        # Anchors
        anchors = self.MODEL.get_anchors(image_shape)
        anchors = np.broadcast_to(anchors, (self.CONFIG.BATCH_SIZE,)
+ anchors.shape)

        if verbose:
            print("molded_images", molded_images)
            print("image_metas", image_metas)
            print("anchors", anchors)

        #print("Run detection")
        # Run object detection
        detections,    _,    _,    mrcnn_mask,    _,    _,    _    =
self.sess.run([self.mrcnn_detection,                self.mrcnn_class,
self.mrcnn_bbox,    self.mrcnn_mask,    self.ROI,    self.rpn_class,
self.rpn_bbox], feed_dict={
                    self.input_image:      molded_images,
                    self.input_image_meta: image_metas,
                    self.input_anchors:    anchors
                })

        # Process detections
        results = []
        for i, image in enumerate(images):
            final_rois, final_class_ids, final_scores, final_masks =\
                self.MODEL.unmold_detections(detections[i],
mrcnn_mask[i],
                                             image.shape,
molded_images[i].shape,
                                             windows[i])
            results.append({
                "rois": final_rois,
                "class_ids": final_class_ids,
                "scores": final_scores,
                "masks": final_masks,
            })
        return results

    ############################################################
    #  Training
    ############################################################
    def train(self, augmentation=None, verbose=1):
        keras.backend.clear_session()
        from .nnmrcnn import Dataset
        import mrcnn.model as modellib
        from mrcnn import utils
        from imgaug import augmenters as iaa
        if verbose:
            self.CONFIG.display()
```

```
        model           =           modellib.MaskRCNN(mode="training",
config=self.CONFIG, model_dir=self.LOG_DIR)

        # Select weights file to load
        assert  self.CONFIG.WEIGHTS  and  type(self.CONFIG.WEIGHTS)  is
str

        if self.CONFIG.WEIGHTS.lower() == "coco":
            weights_path         =         os.path.join(self.LOG_DIR,
"mask_rcnn_coco.h5")
            # Download weights file
            if not os.path.exists(weights_path):
                utils.download_trained_weights(weights_path)
        elif self.CONFIG.WEIGHTS.lower() == "last":
            # Find last trained weights
            weights_path = model.find_last()
        elif self.CONFIG.WEIGHTS.lower() == "imagenet":
            # Start from ImageNet trained weights
            weights_path = model.get_imagenet_weights()
        else:
            weights_path = self.CONFIG.WEIGHTS

        # Load weights
        if verbose:
            print("Loading weights ", weights_path)
        if self.CONFIG.WEIGHTS.lower() == "coco":
            # Exclude the last layers because they require a matching
            # number of classes
            model.load_weights(weights_path, by_name=True, exclude=[
                "mrcnn_class_logits",  "mrcnn_bbox_fc",  "mrcnn_bbox",
"mrcnn_mask"])
        else:
            model.load_weights(weights_path, by_name=True)

        # Training dataset.
        if verbose:
            print("Prepare train data")
        dataset_train = Dataset()
        dataset_train.load_numberplate("train", self.CONFIG)
        dataset_train.prepare()

        # Validation dataset
        if verbose:
            print("Prepare validation data")
        dataset_val = Dataset()
        dataset_val.load_numberplate("val", self.CONFIG)
        dataset_val.prepare()

        # Image augmentation
        #
http://imgaug.readthedocs.io/en/latest/source/augmenters.html
        augmentation_default = iaa.SomeOf((0, 2), [
            iaa.Fliplr(0.5),
```

```
            iaa.Flipud(0.5),
            iaa.OneOf([iaa.Affine(rotate=90),
                       iaa.Affine(rotate=180),
                       iaa.Affine(rotate=270)]),
            iaa.Multiply((0.8, 1.5)),
            iaa.GaussianBlur(sigma=(0.0, 5.0))
        ])
        augmentation = augmentation or augmentation_default

        # *** This training schedule is an example. Update to your
needs ***
        # Since we're using a very small dataset, and starting from
        # COCO trained weights, we don't need to train too long. Also,
        # no need to train all layers, just the heads should do it.
        if verbose:
            print("Training network")
        model.train(dataset_train, dataset_val,
                    learning_rate=self.CONFIG.LEARNING_RATE,
                    epochs=self.CONFIG.EPOCHS,
                    augmentation=augmentation,
                    layers=self.CONFIG.LAYERS )
```

OptionsDetector.py

```
import cv2
import keras
import os
import numpy as np
from keras.layers import merge
from keras.optimizers import Adam
from keras.regularizers import l2
from keras.layers.normalization import BatchNormalization
from keras import models
from keras import layers
from keras import backend as K
from keras.models import Model, Input
from keras.preprocessing import image
from keras.applications import VGG16
from keras import callbacks
from keras.models import load_model
import tensorflow as tf
from tensorflow.python.framework import graph_io
from tensorflow.python.tools import freeze_graph
from tensorflow.core.protobuf import saver_pb2
from tensorflow.python.training import saver as saver_lib
from          tensorflow.python.framework.graph_util          import
convert_variables_to_constants

from .mcm.mcm import download_latest_model
from .Base.ImgGenerator import ImgGenerator

class OptionsDetector(ImgGenerator):
    def __init__(self, options = {}):
```

```
        # input
        self.DEPTH          = 1
        self.HEIGHT         = 64
        self.WEIGHT         = 295
        self.COLOR_CHANNELS = 3

        # outputs 1
        self.CLASS_REGION        =        options.get("class_region",
["xx_unknown",   "eu_ua_2015",   "eu_ua_2004",   "eu_ua_1995",   "eu",
"xx_transit", "ru", "kz", "eu-ua-fake-dnr", "eu-ua-fake-lnr"])

        # outputs 2
        self.CLASS_STATE  =  options.get("class_state",  ["garbage",
"filled", "not filled", "empty"])

        # outputs 3
        self.CLASS_COUNT_LINE = options.get("class_count_line", ["0",
"1", "2", "3"])

        # model
        self.MODEL = None

        # model hyperparameters
        self.OUT_DENSE_INIT                = 'uniform'
        self.OUT_DENSE_ACTIVATION          = 'softmax'
        self.DENSE_ACTIVATION              = 'softmax'
        self.DROPOUT_1                     = 0.2
        self.DROPOUT_2                     = 0.5
        self.DENSE_LAYERS                  = 512
        self.ENSEMBLES                     = 1
        self.BATCH_NORMALIZATION_AXIS      = -1
        self.L2_LAMBDA                     = 0.001
        self.W_REGULARIZER                 = l2(self.L2_LAMBDA)

        # callbacks hyperparameters
        self.REDUCE_LRO_N_PLATEAU_PATIENCE = 10
        self.REDUCE_LRO_N_PLATEAU_FACTOR   = 0.1

        # train hyperparameters
        self.BATCH_SIZE       = 32
        self.STEPS_PER_EPOCH  = 0 # defain auto
        self.VALIDATION_STEPS = 0 # defain auto
        self.EPOCHS           = 150

        # compile model hyperparameters
        self.LOSSES = {
            "REGION":              "categorical_crossentropy",      #
tf.losses.softmax_cross_entropy
            "STATE":               "categorical_crossentropy",      #
tf.losses.softmax_cross_entropy
            "COUNT_LINE":     "categorical_crossentropy",          #
tf.losses.softmax_cross_entropy
        }
```

```
        self.LOSS_WEIGHTS    =    {"REGION":    1.0,    "STATE":    1.0,
"COUNT_LINE": 1.0}
        self.OPT = "adamax" #  tf.keras.optimizers.Adamax
        self.METRICS = ["accuracy"]

        # for frozen graph
        self.INPUT_NODE = "input_2:0"
        self.OUTPUT_NODES  =  ("REGION/Softmax:0",  "STATE/Softmax:0",
"COUNT_LINE/Softmax:0")

    def change_dimension(self, w, h):
        if w != self.WEIGHT and h != self.HEIGHT:
            self.HEIGHT         = h
            self.WEIGHT         = w
            if self.MODEL != None:
                self.MODEL.layers.pop(0)
                newInput   =   Input(shape=(self.HEIGHT,   self.WEIGHT,
self.COLOR_CHANNELS))
                newOutputs = self.MODEL(newInput)
                self.MODEL = Model(newInput, newOutputs)

    def   create_model(self,   input_model,   conv_base,   dropout_1,
dropout_2, dense_layers, output_labels1, \
                       output_labels2, output_labels3, out_dense_init,
W_regularizer, \
                       out_dense_activation,          dense_activation,
BatchNormalization_axis):
        # cnn
        x = conv_base

        # classificator 1
        x1 = layers.Flatten()(x)
        x1 = layers.Dropout(dropout_2)(x1)
        x1                  =                layers.Dense(dense_layers,
activation=dense_activation)(x1)
        x1                                                              =
layers.BatchNormalization(axis=BatchNormalization_axis)(x1)
        x1    =    layers.Dense(output_labels1,    init=out_dense_init,
W_regularizer=W_regularizer)(x1)
        x1          =          layers.Activation(out_dense_activation,
name="REGION")(x1)

        # classificator 2
        x2 = layers.Flatten()(x)
        x2 = layers.Dropout(dropout_2)(x2)
        x2                  =                layers.Dense(dense_layers,
activation=dense_activation)(x2)
        x2                                                              =
layers.BatchNormalization(axis=BatchNormalization_axis)(x2)
        x2    =    layers.Dense(output_labels2,    init=out_dense_init,
W_regularizer=W_regularizer)(x2)
        x2          =          layers.Activation(out_dense_activation,
name="STATE")(x2)
```

```
        # classificator 3
        x3 = layers.Flatten()(x)
        x3 = layers.Dropout(dropout_2)(x3)
        x3                =                layers.Dense(dense_layers,
activation=dense_activation)(x3)
        x3                                                          =
layers.BatchNormalization(axis=BatchNormalization_axis)(x3)
        x3   =   layers.Dense(output_labels3,   init=out_dense_init,
W_regularizer=W_regularizer)(x3)
        x3         =         layers.Activation(out_dense_activation,
name="COUNT_LINE")(x3)

        #x = keras.layers.concatenate([x1, x2], axis=1)
        model = Model(input=input_model, outputs=[x1, x2, x3])

        # compile model
        model.compile(
            optimizer=self.OPT,
            loss=self.LOSSES,
            loss_weights=self.LOSS_WEIGHTS,
            metrics=self.METRICS
        )
        return model

    def ensemble(self, models, model_input):
        # collect outputs of models in a list
        outputs = [model(model_input) for model in models]

        # averaging outputs
        y = layers.Average()(outputs)

        # build model from same input and avg output
        model = Model(inputs=model_input, outputs=y, name='ensemble')

        return model

    def prepare(self, base_dir, verbose=1):
        if verbose:
            print("START PREPARING")
        # you mast split your data on 3 directory
        train_dir = os.path.join(base_dir, 'train')
        validation_dir = os.path.join(base_dir, 'val')
        test_dir = os.path.join(base_dir, 'test')

        # compile generators
        self.train_generator                                        =
self.compile_train_generator(train_dir,  (self.HEIGHT,  self.WEIGHT),
self.BATCH_SIZE)
        self.validation_generator                                   =
self.compile_test_generator(validation_dir,             (self.HEIGHT,
self.WEIGHT), self.BATCH_SIZE)
```

```
        self.test_generator  =  self.compile_test_generator(test_dir,
(self.HEIGHT, self.WEIGHT), self.BATCH_SIZE)
        if verbose:
            print("DATA PREPARED")

    def create_conv(self, inp):
         # trainable cnn model
        conv_base = VGG16(weights='imagenet',
            include_top=False)
        # block trainable cnn parameters
        conv_base.trainable = False
        return conv_base(inp)

    def create_simple_conv(self, inp):
        conv_base = layers.Conv2D(32, (3, 3), activation='relu')(inp)
        conv_base = layers.MaxPooling2D((2, 2))(conv_base)

        conv_base       =       layers.Conv2D(64,       (3,       3),
activation='relu')(conv_base)
        conv_base = layers.MaxPooling2D((2, 2))(conv_base)

        conv_base       =       layers.Conv2D(128,       (3,       3),
activation='relu')(conv_base)
        conv_base = layers.MaxPooling2D((2, 2))(conv_base)

        conv_base       =       layers.Conv2D(128,       (3,       3),
activation='relu')(conv_base)
        conv_base = layers.MaxPooling2D((2, 2))(conv_base)

        return conv_base

    def train(self, log_dir="./", verbose=1, cnn="simple"):
        # init count outputs
        self.OTPUT_LABELS_1 = len(self.CLASS_REGION)
        self.OTPUT_LABELS_2 = len(self.CLASS_STATE)
        self.OTPUT_LABELS_3 = len(self.CLASS_COUNT_LINE)

        # create input
        input_model    =    Input(shape=(self.HEIGHT,    self.WEIGHT,
self.COLOR_CHANNELS))

        if (cnn == "simple"):
            conv_base = self.create_simple_conv(input_model)
        else:
            conv_base = self.create_conv(input_model)

        # traine callbacks
        self.CALLBACKS_LIST = [
            callbacks.ModelCheckpoint(
                filepath=os.path.join(log_dir, 'buff_weights.h5'),
                monitor='val_loss',
                save_best_only=True,
            ),
```

```
            callbacks.ReduceLROnPlateau(
                monitor='val_loss',
                factor=self.REDUCE_LRO_N_PLATEAU_FACTOR,
                patience=self.REDUCE_LRO_N_PLATEAU_PATIENCE,
            )
        ]

        # train all
        modelsArr = []
        for i in np.arange(self.ENSEMBLES):
            # create model
            model  =  self.create_model(input_model  =  input_model,
conv_base = conv_base, \
                                        dropout_1    =    self.DROPOUT_1    ,
dropout_2 = self.DROPOUT_2, dense_layers = self.DENSE_LAYERS, \
                                        output_labels1                        =
self.OTPUT_LABELS_1, output_labels2 = self.OTPUT_LABELS_2, \
                                        output_labels3                        =
self.OTPUT_LABELS_3,
                                        out_dense_init                        =
self.OUT_DENSE_INIT, W_regularizer = self.W_REGULARIZER, \
                                        out_dense_activation                  =
self.OUT_DENSE_ACTIVATION , dense_activation = self.DENSE_ACTIVATION,
\
                                        BatchNormalization_axis               =
self.BATCH_NORMALIZATION_AXIS)

            # train
            history = model.fit_generator(
                self.train_generator,
                steps_per_epoch=self.STEPS_PER_EPOCH,
                epochs=self.EPOCHS,
                callbacks=self.CALLBACKS_LIST,
                validation_data=self.validation_generator,
                validation_steps=self.VALIDATION_STEPS,
                verbose=verbose
            )

            # load best model
            model.load_weights(os.path.join(log_dir,
'buff_weights.h5'))

            # append to models
            modelsArr.append(model)

        # merge ensembles
        if len(modelsArr) > 1:
            self.MODEL = self.ensemble(modelsArr, input_model)
        elif len(modelsArr) == 1:
            self.MODEL = modelsArr[0]

        return  self.MODEL
```

```
    def test(self):
        test_loss, test_loss1, test_loss2, test_loss3, test_acc1,
test_acc2,                          test_acc3                          =
self.MODEL.evaluate_generator(self.test_generator,
steps=self.VALIDATION_STEPS)
        print("test loss: {}".format(test_loss))
        print("test loss: {}      test loss: {}          test loss:
{}".format(test_loss1, test_loss2, test_loss3))
        print("test acc: {}       test acc {}           test acc
{}".format(test_acc1, test_acc2, test_acc3))
        return  test_loss,  test_loss1,  test_loss2,  test_loss3,
test_acc1, test_acc2, test_acc3

    def save(self, path, verbose=1):
        if self.MODEL != None:
            if bool(verbose):
                print("model save to {}".format(path))
            self.MODEL.save(path)

    def isLoaded(self):
        if self.MODEL == None:
            return False
        return True

    @classmethod
    def get_classname(cls):
        return cls.__name__

    def load(self, path_to_model, options={}, verbose = 0):
        if path_to_model == "latest":
            model_info                                              =
download_latest_model(self.get_classname(), "simple")
            path_to_model   = model_info["path"]
            options["class_region"] = model_info["class_region"]

        self.CLASS_REGION   =   options.get("class_region",   ["xx-
unknown",  "eu-ua-2015",  "eu-ua-2004",  "eu-ua-1995",  "eu",  "xx-
transit", "ru", "kz", "eu-ua-ordlo-dnr", "eu-ua-ordlo-lnr", "ge"])
        if path_to_model.split(".")[-1] != "pb":
            self.MODEL = load_model(path_to_model)
            if verbose:
                self.MODEL.summary()
        else:
            self.load_frozen(path_to_model)
            self.predict = self.frozen_predict

    def getRegionLabel(self, index):
        return self.CLASS_REGION[index]

    def getStateLabel(self, index):
        return self.CLASS_STATE[index]

    def predict(self, imgs, return_acc=False):
```

```
        Xs = []
        for img in imgs:
            Xs.append(self.normalize(img))

        predicted = [[], [], []]
        if bool(Xs):
            predicted = self.MODEL.predict(np.array(Xs))

        regionIds = []
        for region in predicted[0]:
            regionIds.append(int(np.argmax(region)))

        stateIds = []
        for state in predicted[1]:
            print(state)
            stateIds.append(int(np.argmax(state)))

        countLines = []
        for countL in predicted[2]:
            countLines.append(int(np.argmax(countL)))

        if return_acc:
            return regionIds, stateIds, countLines, predicted
        return regionIds, stateIds, countLines

    def getRegionLabels(self, indexes):
        return [self.CLASS_REGION[index] for index in indexes]

    def compile_train_generator(self, train_dir, target_size,
batch_size=32):
        # with data augumentation
        imageGenerator = ImgGenerator(train_dir, self.WEIGHT,
self.HEIGHT, self.BATCH_SIZE, [len(self.CLASS_STATE),
len(self.CLASS_REGION), len(self.CLASS_COUNT_LINE)])
        print("start train build")
        imageGenerator.build_data()
        self.STEPS_PER_EPOCH = self.STEPS_PER_EPOCH or
imageGenerator.n / imageGenerator.batch_size or imageGenerator.n /
imageGenerator.batch_size + 1
        print("end train build")
        return  imageGenerator.generator()

    def compile_test_generator(self, test_dir, target_size,
batch_size=32):
        imageGenerator = ImgGenerator(test_dir, self.WEIGHT,
self.HEIGHT, self.BATCH_SIZE, [len(self.CLASS_STATE),
len(self.CLASS_REGION), len(self.CLASS_COUNT_LINE)])
        print("start test build")
        imageGenerator.build_data()
        self.VALIDATION_STEPS = self.VALIDATION_STEPS or
imageGenerator.n / imageGenerator.batch_size or imageGenerator.n /
imageGenerator.batch_size + 1
        print("end test build")
```

```
        return  imageGenerator.generator()

    def load_frozen(self, FROZEN_MODEL_PATH):
        graph_def = tf.GraphDef()
        with tf.gfile.GFile(FROZEN_MODEL_PATH, "rb") as f:
            graph_def.ParseFromString(f.read())

        graph = tf.Graph()
        with graph.as_default():
            self.net_inp,     self.net_out1,     self.net_out2     =
tf.import_graph_def(
                graph_def,   return_elements   =   [self.INPUT_NODE,
self.OUTPUT_NODES[0], self.OUTPUT_NODES[1]]
            )
            #print([x.name for x in graph_def.node])

        sess_config = tf.ConfigProto()
        sess_config.gpu_options.allow_growth = True
        self.sess = tf.Session(graph=graph, config=sess_config)

    def frozen_predict(self, imgs):
        Xs = []
        for img in imgs:
            img = self.normalize(img)
            Xs.append(img)

        predicted = [[], []]
        if bool(Xs):
            predicted = self.sess.run([self.net_out1, self.net_out2],
feed_dict={self.net_inp:Xs})

        regionIds = []
        for region in predicted[0]:
            regionIds.append(int(np.argmax(region)))

        stateIds = []
        for state in predicted[1]:
            stateIds.append(int(np.argmax(state)))

        return regionIds, stateIds
```