**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ**

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

# Пояснювальна записка

до магістерської роботи

на ступінь вищої освіти магістр

на тему: «**Розробка інформаційної системи розпізнавання дорожніх знаків на основі методів машинного навчання»**

Виконав: студент 6 курсу, групи ПДМ–61

спеціальності

121 Інженерія програмного забезпечення

(шифр і назва спеціальності)

Пальчук М.Е.

(прізвище та ініціали)

Керівник Жебка В.В.

(прізвище та ініціали)

Рецензент

(прізвище та ініціали)

Київ – 2021

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ**

**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Магістр»
Спеціальність - 121 «Інженерія програмного забезпечення»

**ЗАТВЕРДЖУЮ**
Завідувач кафедри
Інженерії програмного забезпечення
_____О.В. Негоденко

“____”____20___року

# З А В Д А Н Н Я
## НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

**Пальчуку Михайлу Едуардовичу**
(прізвище, ім‘я, по батькові)

1. Тема роботи: «Розробка інформаційної системи розпізнавання дорожніх знаків на основі методів машинного навчання»

Керівник роботи к.т.н., доцент Жебка Вікторія Вікторівна ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “13” жовтня 2020 року №230.

2. Строк подання студентом роботи 24.12.2020
3. Вихідні дані до роботи:
   3.1. Вимоги до кваліфікаційної роботи магістра з актуальних завдань спеціальності;
   3.2. Нормативні матеріали (стандарти, Гости);
   3.3. Технічні вимоги;
   3.4. Науково-технічна література з питань, пов‘язаних з темою роботи.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
   4.1. Порівняльний аналіз результатів, отриманих іншими авторами;
   4.2. Методика дослідження;
   4.3. Результати дослідження;
   4.4. Висновки
5. Перелік графічного матеріалу.
6. Дата видачі завдання 02.11.2020

# КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів магістерської роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Підбір науково-технічної літератури | 02.11.20 | Виконано |
| 2 | Огляд існуючих рішень та літератури | 03.11.20 | Виконано |
| 3 | Огляд систем розпізнавання знаків | 12.11.20 | Виконано |
| 4 | Розробка системи розпізнавання знаків | 16.11.20 | Виконано |
| 5 | Тестування та покращення розробленої системи | 24.11.20 | Виконано |
| 6 | Вступ, висновки, реферат | 08.12.20 | Виконано |
| 7 | Розробка обов'язкових демонстраційних матеріалів | 11.12.20 | Виконано |
| 8 | Здача роботи | 24.12.20 | |

Студент _____ _____
( підпис ) (прізвище та ініціали)

Керівник роботи _____
підпис ) (прізвище та ініціали)

# РЕФЕРАТ

Текстова частина магістерської роботи : 77 с., 6 табл., 35 рис., 22 джерела.

*Об`єкт дослідження* – розпізнавання дорожніх знаків для IOS.

*Предмет дослідження* – інформаційна система розпізнавання дорожніх знаків.

*Мета роботи* – покращення процесу розпізнавання дорожніх знаків для IOS за допомогою розробленої інформаційної системи.

Методи дослідження — система перевірки ефективності алгоритмів розпізнавання дорожніх знаків, тренування різних моделей нейронних мереж та їх тестування.

У даній статті мова йде про розробку інформаційної системи розпізнавання дорожніх знаків для iOS.

Розпізнавання дорожніх знаків – актуальна проблема у сучасному світі. Технології у даній галузі розвиваються досить швидко, але не стають доступнішими так швидко, як хотів би кінцевий користувач – водій. Крім того, більшість доступних систем мають недоліки – вони розпізнають дуже малу кількість знаків, вразливі до погодніх умов, не переносяться з одного автомобіля на інший. Такі системи занадто тісно інтегровані з автомобілем, присутні лише в нових автомобілях і є дорогими.

У даній роботі досліджується можливість розробити універсальну систему розпізнавання дорожніх знаків для телефону. Сучасні смартфони мають досить велику потужність, їхні процесори оптимізовані для операцій нейронних мереж, і доступні кожному.

У даній роботі досліджується різниця між класичним комп’ютерним зором та нейронними мережами для розпізнавання дорожніх знаків. Також вона описує демо версію системи для розпізнавання знаків з телефону.

Ключові слова: IOS, MACHINE LEARNING, COMPUTER VISION, NEURAL NETWORK, CONVOLUTIONAL NETWORK, ДОРОЖНІ ЗНАКИ.

ЗМІСТ

# СПИСОК СКОРОЧЕНЬ

| | |
|---|---|
| CV | Computer Vision |
| RGB | Red, Green, Blue – кольорова фільтрація |
| HSL | Hue, Saturation, Lightness – альтернативна кольорова фільтрація |
| SIFT | Scale Invariant Feature Transformation |
| NMS | Non Maximum Supression |
| GTSDB | German Traffic Sign Detection Benchmark |
| IOU | Intersection Over Union |
| TP | True Positive |
| FP | True Negative |
| FN | False Negative |
| TN | True Negative |
| CN | Convolutional Network |
| ML | Machine Learning |
| CLAHE | Contrast Limited Adaptive Histogram Equalization |
| iOS | iPhone Operation System |
| SDK | Software Development Kit |
| API | Application Programming Interface |
| NN | Neural Network |
| MVC | Model View Controller |
| MVVM | Model View ViewModel |
| VIPER | View, Interactor, Presenter, Entity, Router |
| UI | User Interface |

# ВСТУП

Щорічно кількість автомобілів на дорогах невпинно зростає. У свою чергу це призводить до збільшення кількості дорожньо-транспортних пригод. Тому гостро стоїть питання безпеки дорожнього руху. Як відомо, у переважній більшості випадків, вирішальну роль відіграє так званий “людський фактор”. Часто трапляться так, що водій не може зреагувати на пішохода, який раптово з’явився на дорозі, через те що перед цим він пропустив попереджувальний знак. Тому для вирішення цієї проблеми виробники автомобілів намагаються якомога більше комп’ютеризувати процес керування автомобілем. Впроваджуються різноманітні спеціалізовані комп’ютерні системи, завданням яких є допомога водієві. Однією з таких систем є система розпізнавання дорожніх знаків. Дану система мають в своєму активі багато відомих автовиробників - Audi, BMW, Ford, Mercedes- Benz, Opel, Volkswagen. Проте більшість із цих систем мають ряд недоліків. До таких недоліків варто віднести те, що деякі з цих систем можуть розпізнавати лише знаки обмеження швидкісного режиму. Також вони не є універсальними, тобто встановлюються компаніями лише у автомобілі власного виробництва і не сумісні з автомобілями інших марок. Висока вартість і закритість цих систем також не можна занести їм в актив. Тому актуальним залишається питання доступних і універсальних систем розпізнавання дорожніх знаків.

У перспективі подібні системи можуть бути використані в повністю автоматизованих безпілотних автомобілях. В даний час тестування таких автомобілів йде повним ходом. Вже у найближчому майбутньому слід очікувати появу безпілотних автомобілів на дорогах. Тому питання про якість систем, що будуть слідкувати за дотриманням правил дорожнього руху є вкрай актуальним.

Крім того, існуючі системи є досить дорогими і малопоширеними, тому актуальним є питання розробки альтернативної системи, яка буде доступна для всіх водіїв без додаткових витрат на обладнання.

# 1 ПОСТАНОВКА ЗАДАЧІ ТА ВИЗНАЧЕННЯ СПОСОБІВ ЇЇ ВИРІШЕННЯ

## 1.1 Постановка задачі

Може скластися враження, що системи розпізнавання дорожніх знаків використовують досить простий алгоритм. Камера знімає ту частину поля зору водія,, де можуть знаходитись дорожні знаки; зображення передаються інформаційній системі, яка здійснює пошук і ідентифікацію дорожнього знаку. Існує декілька методів для вирішення задачі розпізнавання дорожніх знаків, що використовують різні підходи, наприклад, порівняння з шаблоном, еластичні еталони порівняння, класифікація за допомогою нейронних мереж [1] та інші. Всі дорожні знаки створені таким чином, щоб бути помітними і зрозумілими. Це робить їх хорошим об’єктом для розпізнавання. Проте існує ряд проблем, які ускладнюю виявлення і розпізнавання дорожнього знаку. Наприклад, несприятлива погодна, як дощ, сніг, туман, і недастатня освітленість дороги. Крім того, знак може бути пошкоджений або закритиий іншим об'єктом. Саме тому необхідно обрати найбільш ефективний метод, який працює коректно у різноманітних екстремальних умовах.

Усі існуючі сучасні системи ідентифікації дорожніх знаків практично однакові і складаються з відеокамери, операційного блоку і екрану. Операційний блок це процесор, мікроконтролер, на якому виконуються всі обчислювальні операції, пов’язані з розпізнаванням дорожніх знаків на фото, отриманому з камери. При ідентифікації знаку інформація виводиться на екран і може супроводжуватись звуковим сигналом.

Описаний вище тип системи як правило є в наявності лише в автомобілях преміум- класу (BMW 5-ї, 6-ї, 7-ї серії, Mercedes E класу і вище, Opel Insignia). Навіть для таких авто дана система представлена в якості дорогої опції в комплекті з навігаційною системою. Це вбудовані системи, встановлення яких може коштувати від 1000 до 4000 євро. У продажу також можна знайти портативний прилад, який має функцію розпізнавання дорожніх знаків, є комп'ютер-навігатор Blaupunkt TravelPilot 700, який коштує від 600 до 700 євро. Можна зробити

висновок, що інтелектуальні системи допомоги водієві є досить дорогими. Це одна із проблем, які сповтльнюють масове впровадження подібних систем. Виникає проблема: як зробити системи розпізнавання дорожніх знаків доступнішими? Оскільки сучасні мобільні пристрої стрімко розвиваються, мають високу продуктивніть у складних обчисленнях, можна зробити висновок, що найоптимальнішим на сьогоднішній день варіантом є реалізація системи на базі мобільних пристроїв. Відсоток користувачів смартфонів серед водіїв досить великий. Їх часто використовують в якості навігаторів. Технічні характеристики сучасних мобільних пристроїв дозволяють вирішувати ті задачі, з якими стикаються системи забезпечення безпечного дорожнього руху.

Даний підхід можна реалізувати наступним чином: мобільний пристрій за допомогою спеціального тримача, який доступний у вільному продажу вже довгий час, закріплюється на лобовому склі автомобіля. Використовуючи вбудовану в мобільний пристрій камеру, програма в реальному часі робить фото дороги. На кожному кадрі програма намагається виявити дорожні знаки. В разі виявлення знака, на екран мобільного пристрою виводиться повідомлення, що може супроводжуватись звуковим сигналом. Крім того така програма може працювати в фоновому режимі. Тобто на мобільному пристрої можуть бути відкриті Google карти або ж будь який інший додаток, а повідомлення про дорожні знаки будуть з’являтись поверх відкритих додатків. Прикладом такої програм є RoadAR. Проте даний програмний продукт здатен розпізнати лише деякі дорожні знаки.

Для реалізації такої програмної системи на мобільних пристроях існують оптимізовані бібліотеки технічного зору, такі як OpenCV і FastCV.

Головною перевагою системи розпізнавання дорожніх знаків на базі мобільних пристроїв перед аналогами є відсутність для користувачів необхідності в придбанні спеціального дорогого технічного пристрою для виконання необхідних функцій, а також можливість використання у будь-якому автомобілі. Тобто це допоможе вирішити проблему доступності відповідних систем.

Не менш важливим залишається вибір метода розпізнавання дорожніх знаків. Під час дорожнього руху швидкість детектування знака і якість розпізнавання

відіграватимуть вирішальну роль. І навіть найменша неточність може призвести до фатальних наслідків.

Не зважаючи на значне просування світових виробників в області створення інтелектуальних систем забезпечення безпеки дорожнього руху, в плані функцій розпізнавання дорожніх знаків результати залишаються незначними. Як вже було згадано вище, найбільш відомі системи здатні розрізняти лише невеликий набір знаків обмеження швидкості. Пов’язане це, мабуть, з тим, що збільшення кількості знаків, що розпізнаються, призводить до значних обчислювальних затрат, що, в свою чергу, збільшує час реакції системи. А це є не прийнятним з точки зору оперативності.

Численні експериментальні перевірки показують, що переважна більшість помилок пов’язані з невірним детектуванням об’єктів. Наприклад, замість дорожнього знака система може виявити рекламний щит або інші схожі об’єкти. Тому проблема детектування об’єктів на зображеннях зі складним фоном є головною при розробці систем технічного зору.

### 1.2 Визначення підходів до вирішення проблеми розпізнавання знаків

На даний час відомі щонайменше три підходи до виявлення і локалізації дорожніх знаків: виявлення з використанням інформації про колір, виявлення на основі інформації про геометричну форму і методи, які використовують і колір, і форму.

Колір знаків дозволяє відрізнити їх на фоні навколишньої дорожньої обстановки. Зокрема, для виявлення знаків використовують кольорову фільтрацію в просторі RGB [2]. В інших випадках застосовується простір HSI чи HSV.

Ще одним варіантом є метод детектування дорожніх знаків з використанням кольорового представлення зображення в просторі HSI. Відтінок (H - складова) кожного пікселя порівнюється з еталонним відтінком. За допомогою граничних значень на зображені виділяються контури дорожніх знаків.

Також застосовуються методи з використанням граничної функції для детектування об’єктів. Основою метода є розрахунок відстані в просторі RGB між

кольором пікселя та еталонного кольору. Піксель вважається частиною об’єкта, якщо ця відстань достатньо мала.

Підходи до виявлення знаків на основі кольорової фільтрації не вимогливі до обчислювальних ресурсів. Проте точність таких підходів не достатньо висока.

Характерні недоліки даного методу:

- виникнення проблем при поганих погодних умовах (дощ, сніг, туман);

- більшість алгоритмів не справляються з виявленням дорожніх знаків при недостатньому освітлені;

- методи виявляються не ефективними в ситуаціях, коли частина знаку знаходиться в тіні, а інша частина яскраво освітлена.

Обчислювальні потужності сучасних систем дозволяють використовувати більш складні алгоритми детектування об’єктів в реальному часі. Вони використовують інформацію про форму об’єкта [3].

Для аналізу форми на етапі детектування використовуються методи виділення контурів, такі як метод Канні чи перетворення Хафа [4]. Аналізується площа і положення об’єкта, який імовірно є знаком.

Один із алгоритмів базується на обчислені тангенса нахилу дотичної до об’єкта. Оскільки форма знаку може бути одного з трьох варіантів - круг, прямокутник чи трикутник, то такий метод дозволить виявити форму знака навіть при великому шумі.

В деяких працях автори запропонували метод детектування на основі тільки яскравості пікселів. Об’єкт виділяється на фоні за допомогою лапласіана після використання фільтра для розмиття. Для отримання бінарного зображення використовується визначений поріг, а безпосередньо детектування відбувається за допомогою генетичних алгоритмів з використанням завчасно відомих шаблонів округлостей.

Інформація про форму знака в багатьох алгоритмах детектування враховується у вигляді деякого набору ознак, що розраховуються по зображенню. Існують різні способи формування набору ознак. Одним з найперспективніших можна вважати алгоритм масштабно-інваріантних перетворень ознак (Scale

Invariant Feature Transformation - SIFT), запропонований в 2004 році Лоувом. Часто на практиці використовуються різні модифікації даного алгоритму. Алгоритм володіє рядом характеристик, які дуже важливі для процедур автоматичного відбору ознак. Отримані ознаки є інваріантними до змін в освітлені, шуму на зображеннях, геометричних змін зображення, його розміру і розміру об'єкту відносно зображення. Ці характеристики цілком задовольняють вимоги до задач розпізнавання дорожніх знаків. Алгоритм включає наступні послідовності: виявлення екстремуму в просторі масштабних перетворень, локалізація ключових точок, встановлення правильної орієнтації зображення і генерація ознак.

Для базового алгоритму SIFT при визначені зразка, який відповідає зображенню знака, що розглядається, використовується евклідова відстань в просторі векторів ознак. Таким чином, вибирається той зразок, до якого поточне зображення знака виявилось ближче Але такий метод має ряд недоліків при великій кількості зразків і високій розмірності простору ознак.

Окрім алгоритму SIFT існують інші підходи до автоматизованої генерації наборів ознак. Один із них ґрунтується на використанні коефіцієнтів вейвлет-перетворень Хаара в якості ознак. Для класифікації використовується алгоритм AdaBoost. Основний принцип даного алгоритму навчання полягає в тому, що всі наступні класифікатори використовують для навчання ті вектори, по яким були кожен наступний класифікатор навчається по тим векторам, які були невірно класифіковані попереднім класифікатором.

Методи детектування знаків на основі інформації про форму вимагають високих обчислювальних витрат і в деяких випадках можуть не справлятись з поставленим завданням в режимі реального часу.

Основні недоліки даного підходу:

- об’єкти, що мають геометричну форму, подібну до форми дорожнього знаку (рекламні щити, вікна і т. д. ), теж будуть детектовані;

- дорожні знаки можуть буди пропущені при їх незначних пошкодженнях, перекритті іншими об’єктами;

- детектування і розпізнавання значно ускладнюється через те, що

транспортний засіб рухається відносно дорожнього знака, з чого випливає що масштаби поточного зображення і еталона можуть відрізнятись.

Також застосовуються методи, що використовують інформацію про колір і форму одночасно. Прикладом є методи, коли використовується відтінок і насиченість з кольорового простору HSL. Динамічні пороги використовуються для гістограм відтінку і насиченості. Кінцеві бінарні зображення отримуються за допомогою логічного складання значень відтінку і насиченості. Кінцеве бінарне зображення отримується за допомогою логічного складання значення відтінку і насиченості. Пікселі бінарного зображення розподіляються в 7 типів в залежності від цільового пікселя і його сусіда. Ці типи-шаблони використовуються для детектування форми дорожнього знака.

Існує метод в якому детектування будується по декільком кадрам. Система виявляє зображення, що може бути знаком, і слідкує за ним на кількох наступних кадрах, використовуючи алгоритми, подібні до аналізу оптичного потоку. Якщо на декількох послідовних зображеннях межі дорожнього знаку не змінюються, система відбирає цей знак для класифікації. Така процедура відслідковування знака-кандидата може бути реалізована використовуючи фільтр Калмана. Якщо вважати, що на короткому інтервалі часу знак наближається до камери з постійною швидкістю, то можна передбачити положення знака на кадрі. Таким чином можна скоротити витрати на аналіз областей на кадрі які не містять знака.

Ще одним варіантом для розпізнавання форми і кольору знака є використання простої нейронної мережі. Початкові кольорові зображення спочатку обробляються лапласіаном фільтра Гауса. Потім для сегментації по кольору, а тоді по формі об’єктів використовуються спеціально навчені нейронні мережі. При вдалому детектуванні знака для розпізнавання застосовується співставлення шаблонів.

Виходячи з вище сказаного, можна очікувати, що використання комбінованої інформації про колір і форму може дати суттєве покращення якості і надійності детектування дорожніх знаків.

Із наведеного короткого огляду випливає, що в існуючих підходах задачі

детектування і розпізнавання нерозривно пов’язані. Тому ставиться питання про метод, який би забезпечував розпізнавання дорожнього знака без явного попереднього детектування.

В системах автомобільної безпеки для розпізнавання дорожніх знаків часто використовуються нейронні мережі у вигляді багатошарового персептрона. Перевагами такого підходу є те, що не потрібно попереднього формування векторів ознак. В якості навчальної множини для нейронної мережі використовується безпосередньо зображення знаків.

Так як основні затрати часу потрібні для навчання нейронної мережі, що відбувається на попередньому етапі, то на етапі розпізнавання витрачається мінімум часу.

Головним недоліком нейромережевого підходу є непередбачуваність результатів і велика варіативність помилки розпізнавання.

Серед інших методів розпізнавання дорожніх знаків слід відмітити метод співставлення шаблонів. Даний метод ефективно зарекомендував себе за рахунок своєї простоти і швидкості роботи. При використанні цього методу неодмінною вимогою є необхідність точного виділення внутрішньої області знайденого дорожнього знака для співставлення з еталонним зображенням.

Таким чином, етап детектування знака, в явному чи неявному вигляді, присутній у всіх існуючих підходах. Тому завданням є не уникнення цього етапу, а максимальне покращення точності, швидкості і надійності. Проаналізувавши існуючі методи розпізнавання дорожніх знаків, можна сказати, що доцільним є використання штучних нейронних мереж. Однак класичні ШНМ чутливі до різного роду спотворень зображення. До того ж зображення складаються з великої кількості пікселів, у зв’язку з чим збільшується розмір ШНМ, кількість шарів, нейронів і все це призводить до громіздкої структури, збільшення часу роботи. Тому кращим варіантом буде використання згортаючих нейронних мереж [5]-[6]. Вони достатньо інваріантні до різного роду спотворення вхідного сигналу.

Структура згортаючих нейронних мереж включає в себе чергування згортаючих і субдискретизуючих шарів, і наявність повнозв’язних шарів на виході.

Для виділення дорожнього знака на зображені можна використати згортаючу нейронну мережу з одним нейроном у вхідному шарі, який буде приймати значення в інтервалі [-1:+1], що відповідно означає наявність чи відсутність дорожнього знака на зображені.

На рисунку 1.1 представлена структура згортаючої нейронної мережі для виявлення дорожніх знаків на зображенні.

Згортаючі шари такої нейронної мережі функціонують за формулою:

$$y_k^{(i,j)} = b_k + \sum_{g=1}^{k} \sum_{t=1}^{k} w_{k,s,t} x^{((i-1)+s,(j+t))} \quad (1.1)$$

Субдискредитуючі шари функціонують за формулою:

$$y_k^{(i,j)} = b_k + 1/4 w_k \sum_{s=1}^{2} \sum_{t=1}^{2} w_{k,s,t} x^{((i-1)+s,(j+t))} \quad (1.2)$$

Для навчання мережі використовується алгоритм зворотнього поширення помилки:

$$w_{i,j}(t+1) = w_{i,j}(t) + \eta \sigma_{pi} o_{pi} \quad (1.3)$$

Таку нейронну мережу можна використовувати не тільки для виявлення дорожнього знака на зображені, але і для його ідентифікації. З цими двома задачами згортаюча нейронна мережа здатна досить ефективно впоратись. На рисунку 2 зображена структура згортаючої нейронної мережі для розпізнавання дорожніх знаків.

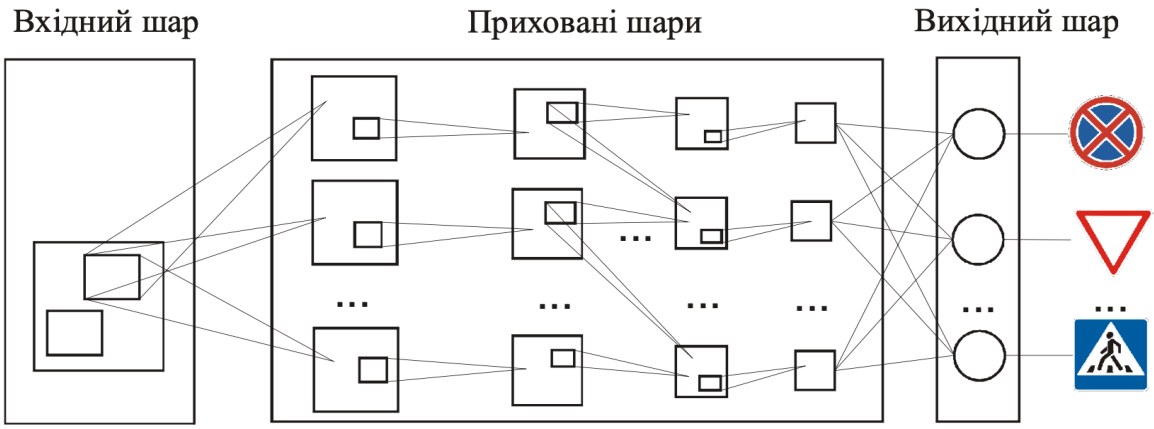



Рисунок 1.1 - Структура згортаючої нейронної мережі для розпізнавання дорожніх знаків

Проте нейронні мережі вимагають тренування. Це ресурсозатратний процес, який, крім того, ще й вимагає велику кількість класифікованих зображень з позначеними знаками для перевірки. Для того, щоб визначити, наскільки ці витрати необхідні, було вирішено дослідити ефективність класичного підходу з розпізнавання кольорів і форм за допомогою комп'ютерного зору, і порівняти з нейронними мережами. Для тренування моделі згорткової нейронної мережі було використано сервіс TensorFlow, розроблений компанією Google, який доволяє дешево тренувати невеликі моделі. Вона тренувалась на 4000 зображень із сету GTSRB (German Traffic Signs Recognition Benchmark), який був зібраний і класифікований вручну для того, щоб можна було тренувати і перевіряти нейронні моделі, що розпізнають дорожні знаки. Він містить знаки всіх класів, які є у Європі.

# 2 РОЗПІЗНАВАННЯ ДОРОЖНІХ ЗНАКІВ ЗА ДОПОМОГОЮ КОМП'ЮТЕРНОГО ЗОРУ

## 2.1 Алгоритм виявлення дорожніх знаків за допомогою комп'ютерного зору

Алгоритм виявлення дорожніх знаків за допомогою комп'ютерного зору складається з декількох етапів, таких як:

- попередня обробка зображень;
- кольорова сегментація;
- фільтрація шуму;
- відповідність фігури;
- немаксимальне придушення.

Мета цього розділу – описати розроблений алгоритм виявлення дорожніх знаків та детально пояснити кожен етап. На кожному кроці відображатиметься стан зображення.

Рисунок 2.1 - Початкове зображення до обробки

Вище приведено приклад початкового зображення до обробки.

## 2.2 Попередня обробка зображень

Перший етап алгоритмів – це обробка зображень. У цьому випадку робиться дуже проста обробка зображень. Зображення в RGB перетворюється на кольорову схему HSV. Кольоровий простір HSV представляє кольори, використовуючи три значення:

Відтінок: Цей канал кодує інформацію про колір. Відтінок можна уявити під кутом, де 0 градусів відповідає червоному кольору, 120 градусів відповідає зеленому кольору, а 240 градусів відповідає синьому кольору.

Насиченість: Цей канал кодує інтенсивність / чистоту кольору. Наприклад, рожевий менш насичений, ніж червоний.

Значення: Цей канал кодує яскравість кольору. У цьому каналі з'являються компоненти тіні та блиску зображення.

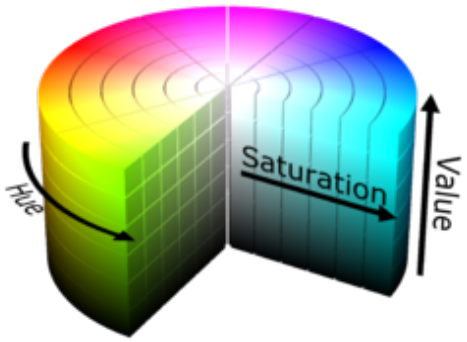



Рисунок 2.2 - Схема кольору HSV

На відміну від RGB, який визначається по відношенню до основних кольорів, HSV визначається способом, подібним до того, як люди сприймають колір. Для нашої проблеми основною перевагою використання кольорового простору HSV є те, що колір, відтінок і довжина хвилі представлені лише компонентом відтінку.

## 2.3 Кольорова сегментація

Існує багато різних методів сегментації кольорів, але в цій роботі ми будемо використовувати метод, розроблений Хасаном Флейхом [4]. Цей метод заснований

на сегментації за регіонами зростання. Зображення відтінків сегментоване відповідно до кольору розглянутого знака. Зазвичай вказується діапазон кутів відтінку, де можна знайти колір знака. Результатом цього кроку є двійкове зображення для ймовірних кандидатів. Це двійкове зображення також використовується для обчислення насіння для зображення насичення. Він розділений на підрегіони 16x16 пікселів, а насіння встановлюється в центрі кожної підрегіону, якщо в бінарному зображенні, генерованому сегментацією відтінків, знайдено достатньо пікселів відтінку. Кількість пікселів достатнього відтінку визначається третиною площі кожного субрегіону. Насіння, генеровані на попередньому етапі, разом із зображенням насичення використовуються як вхідні дані до алгоритму зростання регіону. Зображення насиченості сегментується за допомогою цих насінин, щоб сформувати інше двійкове зображення, що представляє кандидатні об’єкти дорожніх знаків. Останній крок - застосувати логічне І з двох двійкових зображень; тобто бінарне зображення відтінку та бінарне зображення насиченості.

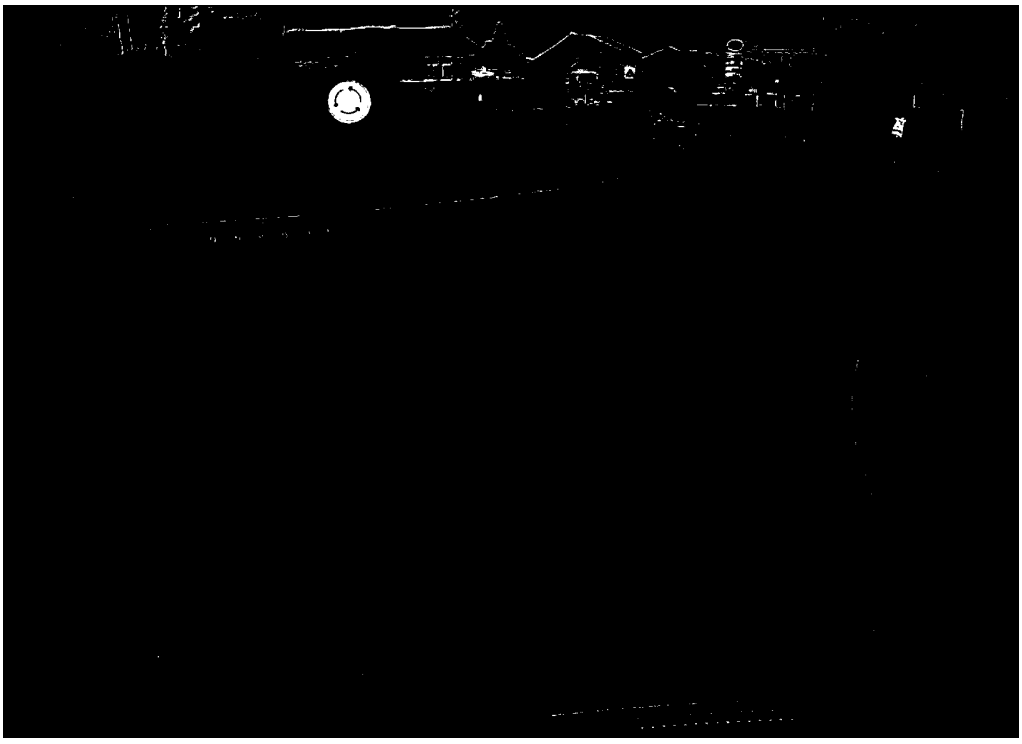


Рисунок 2.3 - Зображення після сегментації кольору

Результат - сегментоване зображення, що містить дорожній знак із зазначеним кольором.

### 2.4 Фільтрація шуму

На зображенні, отриманому з попереднього кроку, є ідеально виділений дорожній знак та деякий шум. Основною метою кроку є видалення шуму для поліпшення якості алгоритму виявлення. Фільтрація шуму складається з 2 етапів: функція “floodFill” та морфологічна трансформація. [16]

#### 2.4.1 Заповнення кольором

Функції floodFill заповнюють підключений компонент, починаючи з точки засівання, заданим кольором. Зв'язок визначається близькістю кольору / яскравості сусідніх пікселів. Для додавання до підключеного компонента колір / яскравість пікселя має бути досить близьким, щоб:

- колір / яскравість одного з сусідів, які вже належать до підключеного компонента у випадку плаваючого діапазону;
- колір / яскравість точки насіння у випадку встановленого діапазону.

За допомогою цих функцій або позначте підключений компонент вказаним кольором на місці, або створіть маску, а потім витягніть контур, або скопіюйте область на інше зображення тощо.

#### 2.4.2 Морфологічне перетворення

Другий крок – морфологічна трансформація. Морфологічні перетворення – це кілька простих операцій, заснованих на формі зображення. Зазвичай це виконується на двійкових зображеннях. Для цього потрібні два входи, один – це наше оригінальне зображення, другий називається структуруючим елементом або ядром, який вирішує характер роботи. Двома основними морфологічними операторами є Ерозія та Розширення. Тоді в гру вступають також його варіанти,

такі як Відкриття, Закриття, Градієнт тощо.

Ерозія - подібна до ерозії грунту, вона розмиває межі об’єкта переднього плану. Ядро ковзає по зображенню (як у 2D згортці). Піксель у вихідному зображенні (або 1, або 0) буде вважатися 1, лише якщо всі пікселі під ядром дорівнюють 1, інакше він стирається (зводиться до нуля). Отже, трапляється так, що всі пікселі біля межі будуть відкинуті залежно від розміру ядра. Тож товщина або розмір об’єкта переднього плану зменшується або просто біла область зменшується на зображенні. Це корисно для усунення невеликих білих шумів від'єднання двох пов'язаних об'єктів тощо.

Розведення – якраз протилежне ерозії. Тут піксельний елемент дорівнює «1», якщо хоча б один піксель під ядром дорівнює «1». Таким чином, це збільшує білу область зображення або збільшує розмір об’єкта переднього плану. Зазвичай у таких випадках, як видалення шуму, ерозія супроводжується розширенням. Тому що ерозія усуває білі шуми, але це також зменшує наш об’єкт. Так він розбавляється. Оскільки шум зник, вони не повернуться, але площа нашого об’єкта збільшується. Це також корисно при з’єднанні зламаних частин об’єкта.

Відкриття – це лише інша назва ерозії, за якою слідує розширення. Це корисно для усунення шуму, як пояснювалося вище.

Закриття – це зворотній спосіб відкриття, розширення з наступною ерозією. Це корисно при закритті невеликих отворів усередині об’єктів переднього плану або невеликих чорних точок на об’єкті.

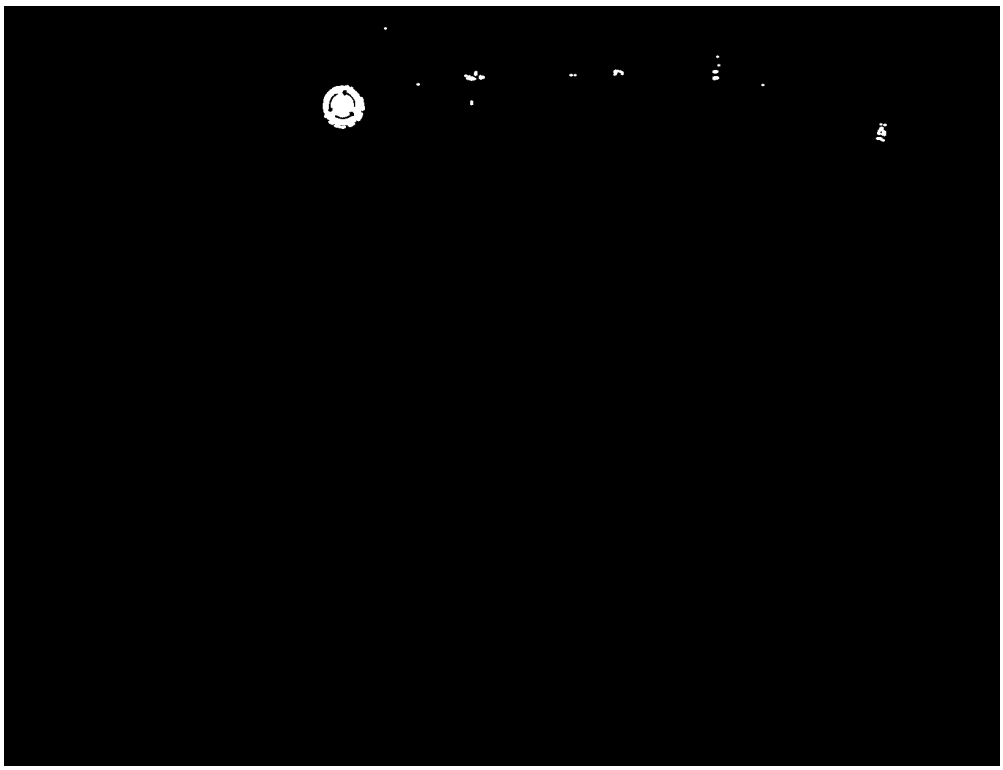


Рисунок 2.4 - Зображення після фільтрації шуму

Для обраної проблеми було використано морфологічне розкриття, оскільки воно добре працює для усунення шуму, який маємо після попереднього кроку.

### 2.5 Відповідність фігури за допомогою розсувного вікна

Оскільки майже всі дорожні знаки мають круглий або квадратний характер, запропонований метод застосовується з цього факту для виявлення гіпотетичних фігур, які досить близькі до дорожніх знаків. Отже, для виявлення будуть використовуватися знаки, що відповідають шаблону зображення.

Відповідність шаблону – це техніка пошуку областей зображення, які збігаються (подібні) із зображенням шаблону (патчем). Нам потрібні два основні компоненти: вихідне зображення (I): зображення, в якому ми очікуємо знайти відповідність зображенню шаблону, та зображення шаблону (T): зображення патча, яке буде порівняно із зображенням шаблону. Наша мета – виявити область, що найбільш відповідає. Щоб визначити область відповідності, ми повинні порівняти зображення шаблону із вихідним зображенням, зсунувши його. Під ковзанням ми

маємо на увазі переміщення патча по одному пікселю за раз (зліва направо, вгору вниз). У кожному розташуванні обчислюється показник, таким чином, він відображає, наскільки "хорошим" чи "поганим" є збіг у цьому місці (або наскільки подібний патч до цієї конкретної області вихідного зображення). Для кожного розташування T над I ви зберігаєте метрику в матриці результатів (R). Кожне розташування (x, y) у R містить метрику відповідності. Показник відповідності, що використовується в даному проекті, обчислюється таким чином:

$$R(x,y) = \frac{\sum_{x`,y§}(T(x`,y`)\cdot I(x+x`,y+y`))}{\sqrt{\sum_{x`,y`} T(x`,y`)^2 \cdot \sum_{x`,y`} I(x+x`,y+y`)^2}} \quad (2.1)$$

У бібліотеці OpenCV ця метрика називається CV_TM_CCORR_NORMED. На рисунку можна побачити шаблони, використані для пошуку збігів.

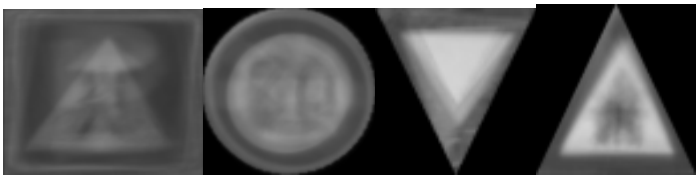


Рисунок 2.5 - Шаблони дорожніх знаків

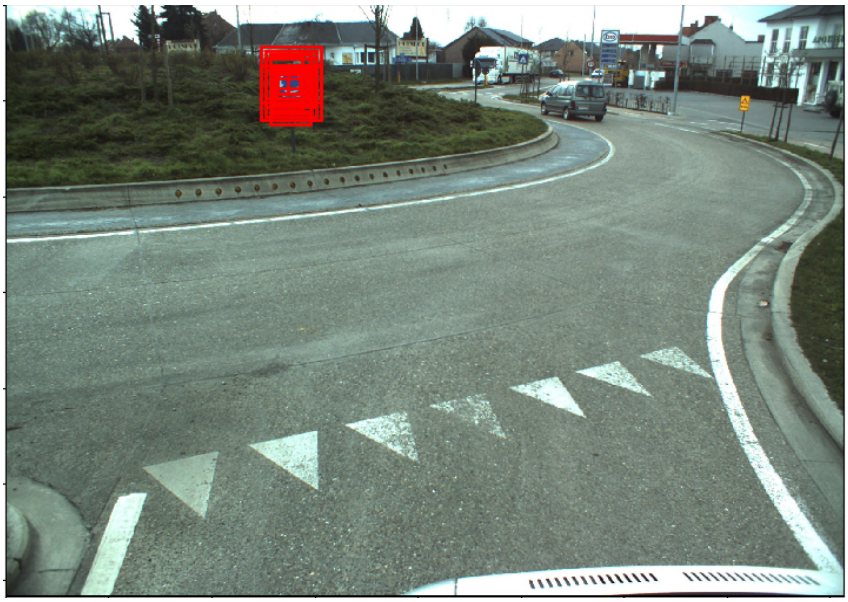


Рисунок 2.6 - Зображення після пошуку за шаблонами

На рисунку 2.6 навколо дорожнього знаку знайдено обмежувальні рамки. Як ми бачимо, вони знаходяться в потрібному місці зображення, але їх дуже багато. На даному етапі потрібен лише обмежувальний квадрат, який представляє знак. Правильний вибір обмежувального вікна – це остання частина алгоритму виявлення дорожніх знаків.

### 2.6 Немаксимальне придушення

Для вибору одинарного обмежувального блоку використовується алгоритм не максимального придушення (NMS) [12]. Взагалі, NMS використовується, щоб переконатися, щоб при виявленні об'єкта конкретний об'єкт ідентифікувався лише один раз. Розглянемо зображення 100×100 із сіткою 9×9, на якому потрібно виявити виявити автомобіль. Якщо цей автомобіль лежить у декількох осередках сітки, NMS гарантує, що буде визначено оптимальну клітинку серед усіх кандидатів, де знаходиться цей автомобіль.

Принцип роботи NMS:

1. Спочатку відкидаються всі ті клітини, де ймовірність присутності об’єкта <= 0,6
2. Тоді визначається клітинка з найбільшою ймовірністю серед кандидатів на об’єкт
3. Нарешті, відкидаються всі комірки з перетином над значенням об'єднання більше 0,5 із клітинкою передбачення.

Нижче наведено псевдокод для алгоритму NMS.

$$
\begin{aligned}
&\textbf{procedure } NMS(b, c) \\
&\quad B_{nms} \leftarrow \emptyset \\
&\quad \textbf{for } b_i \in B \textbf{ do} \\
&\qquad discard \leftarrow False \\
&\qquad \textbf{for } b_i \in B \textbf{ do} \\
&\qquad\quad \textbf{if } same(b_i, b_i) > \lambda_{nms} \textbf{ then} \\
&\qquad\qquad \textbf{if } score(c, b_i) > score(c, b_i) \textbf{ then} \\
&\qquad\qquad\quad discard \leftarrow True \\
&\qquad \textbf{if not } discard \textbf{ then} \\
&\qquad\quad B_{nms} \leftarrow B_{nms} \cup b_i \\
&\textbf{return } B_{nms}
\end{aligned}
\tag{2.2}
$$

NMS була останньою стадією алгоритму. На рисунку 2.8 можна побачити результат роботи алгоритму. Дорожній знак правильно виявлено та підсвічено якісно.

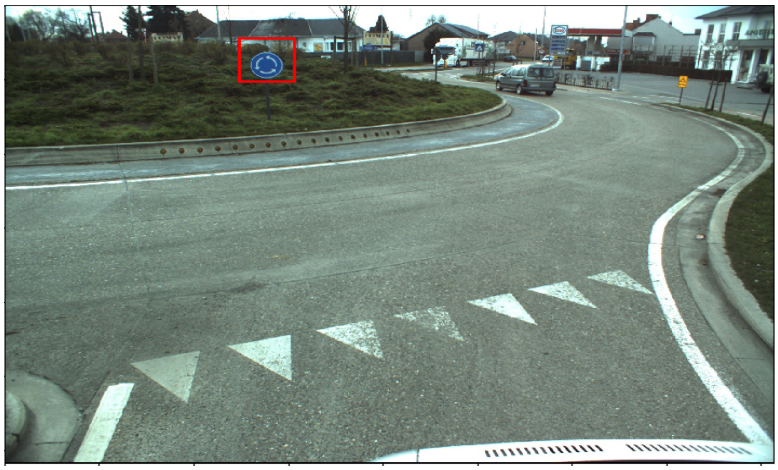


Рисунок 2.7 - Зображення після NMS. Фінальний результат роботи алгоритму

### 2.7 Тестування та оцінка результатів

У даному розділі оцінено роботу алгоритму, використовуючи відомий набір даних дорожніх знаків.

Для оцінки алгоритму було використано набір даних німецького тесту виявлення дорожніх знаків (GTSDB). GTSDB – це оцінка виявлення одного зображення для дослідників, які цікавляться областями комп'ютерного зору, розпізнавання образів та допомоги водію на основі зображень. Передбачається, що він буде представлений на Міжнародній спільній конференції IEEE з нейронних мереж 2013 року. Він має проблему виявлення одного зображення, 900 зображень (розділених на 600 навчальних зображень та 300 оціночних зображень), поділений на систему трьох категорій, яка відповідає властивостям різні підходи до виявлення з різними властивостями, онлайн-система оцінки з негайним аналізом та ранжуванням представлених результатів. Це стандартний набір даних для проблеми виявлення дорожніх знаків.

Проблема виявлення об'єкта не є подібною проблемою з проблемою класифікації, тому загальну точність, точність та відкликання неможливо обчислити безпосередньо. Тож потрібен інший метод оцінки. Для визначення точності та виклику для завдання виявлення об'єкта може використовуватися Intersection Over Union (IOU).

Intersection Over Union (IOU) вимірюється на основі індексу Жакарда, який оцінює перекриття між двома обмежувальними полями. Для цього потрібні обмежувальний ящик Bgt і прогнозований обмежувальний ящик Bp. Застосовуючи IOU, ми можемо визначити, чи є виявлення дійсним (справжнє позитивне) чи ні (хибнопозитивне)

IOU визначається площею перекриття між передбачуваним обмежувальним полем та обмежувальним блоком наземної істини, розділеним площею об’єднання між ними:

$$\mathrm{IOU} = \frac{\mathrm{area}(\mathrm{B_p} \cap \mathrm{B_{gt}})}{\mathrm{area}(\mathrm{B_p} \cup \mathrm{B_{gt}})} \tag{2.3}$$

На зображенні нижче показано IOU між рамкою, що визначає справжні межі об’єкту (зеленим кольором) та виявленою рамкою (червоним кольором).

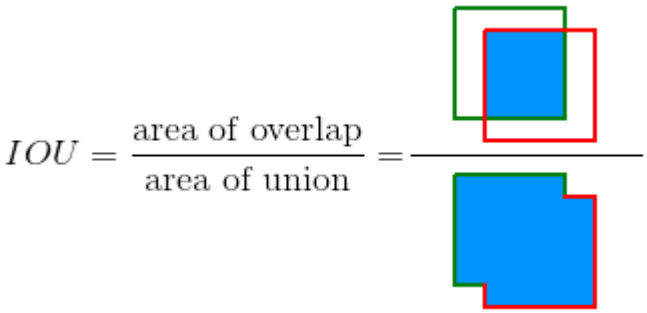



Рисунок 2.8 - Пояснення IOU

Тепер можна визначити точність і відкликання. Деякі основні поняття, що використовуються метриками:

Справжній позитив (TP): правильне виявлення. Виявлення за ВМО ≥ порогове значення

Помилково позитивний (FP): неправильне виявлення. Виявлення за допомогою IOU <поріг

Помилково негативний (FN): Основна істина не виявлена

Справжній негатив (TN): Не застосовується. Це буде виправленим помилковим виявленням. У завданні виявлення об’єкта є багато можливих обмежувальних вікон, які не слід виявляти в зображенні. Таким чином, TN буде всіма можливими обмежувальними коробками, які були неправильно не виявлені (стільки можливих коробок на зображенні). Ось чому він не використовується метриками.

Поріг: залежно від метрики він зазвичай встановлюється на 50%, 75% або 95%.

Точність – це здатність моделі ідентифікувати лише відповідні об'єкти. Це відсоток правильних позитивних прогнозів і визначається як:

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} = \frac{\text{TP}}{\text{all detections}} \tag{2.3}$$

Відкликання – це здатність моделі знаходити всі відповідні випадки (усі обмежувальні рамки істини). Це відсоток справжнього позитиву, виявленого серед усіх відповідних основних істин, і визначається:

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} = \frac{\text{TP}}{\text{all groundc truths}} \quad (2.4)$$

Таблиця 2.1 – Результати оцінки ефективності алгоритму

| Значення | Позитивне виявлення | Негативне виявлення |
|---|---|---|
| Фактична істина | 673 | 0 |
| Фактично помилково | 75 | 152 |

На основі цього можна розрахувати показники:

Відкликання = 81%

Точність = 89%

Правильність = 75%

Результати є цілком непоганими, але далеко не найсучаснішими рішеннями, які використовують підходи глибокого навчання. Але перевага цього підходу полягає в тому, що він не вимагає набору даних для навчання та величезної обчислювальної потужності. Тому він може використовуватися на менш потужній машині, де швидкість виявлення має значення. Саме такий підхід використовується на сучасних автомобілях, бортовий комп'ютер яких не має потужності, достатньої для роботи нейронних мереж.

# 3 ВИКОРИСТАННЯ ГЛИБОКОГО МАШИННОГО НАВЧАННЯ

## 3.1 Алгоритм розпізнавання дорожніх знаків

Алгоритм розпізнавання дорожніх знаків за допомогою машинного навчання складається з таких етапів:

- попередня обробка зображень;
- кольорова сегментація;
- нормалізація;
- фільтрація шуму;

- передача зображення в модель для розпізнавання.

Для тренування моделі було використано той самий набір класифікованих зображень, що і для перевірки першого алгоритму. Під час його аналізу було виявлено, що зображень деяких класів набагато більше, ніж інших, тому модель може бути “упередженою” по відношенню до знаків, що належать до класів, у яких менше прикладів. На рисунку 3.1 показано графік розподілу зображень по класам.

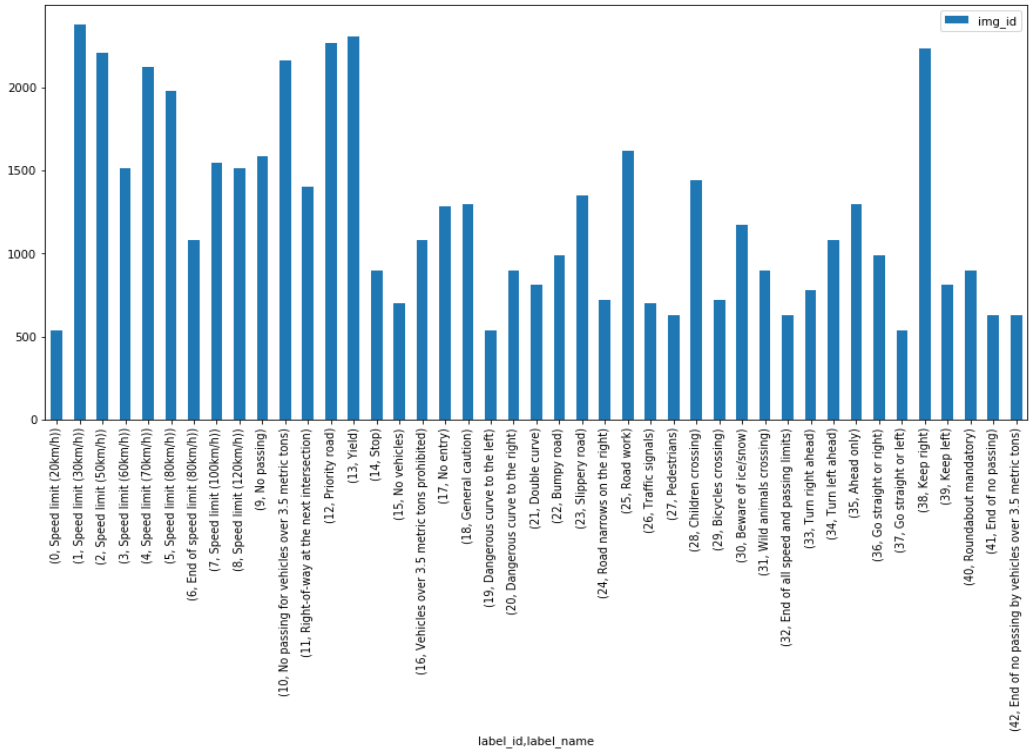



Рисунок 3.1 - Графік розподілу зображень по класам

У наступному розділі описано архітектуру моделі і як вона тренувалася.

### 3.2 Архітектура моделі та її тестування

Підхід базується на згортковій мережі (Convolutional Network) [8, 9], багатошарова архітектура зворотного зв'язку з біологічним натхненням, яка може вивчати кілька етапів інваріантних ознак за допомогою комбінації контрольованих методів та навчання без нагляду. Кожен етап складається із згорткового шару фільтру, шару нелінійного перетворення і шару об’єднання просторових об’єктів.

Шари просторового об’єднання знижують просторову роздільну здатність зображення, тим самим роблячи розпізнавання стійким до невеликих зрушень і геометричних спотворень, подібно до "складних клітин" в стандартній моделі зорової кори. Згорткова мережа зазвичай складається з одного-трьох етапів, обмежених класифікатором з одного або двох додаткових шарів. На основі градієнта процедура нагляду під наглядом оновлює кожен окремий фільтр кожна банка фільтрів у кожному шарі, щоб мінімізувати втрати функція. У традиційних згорткових мережах результати останнього етапу подаються до класифікатора. У цій роботі результати усіх етапів подаються в класифікатор. Це дозволяє класифікатору використовувати не лише функції високого рівня, які, як правило, є глобальними, інваріантними, але з невеликою кількістю точних деталей, але також об'єднані низькорівневі функції, які, як правило, є більш локальними, менш інваріантними, і точніше кодуватимуть місцеві мотиви.

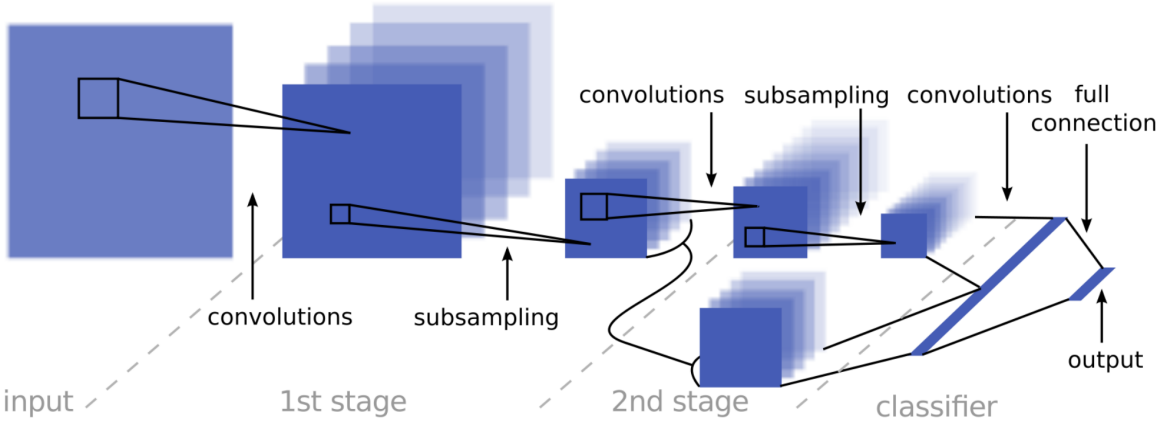



Рисунок 3.2 - Шари згорткової мережі для розпізнавання зображень

Згорткова мережа була реалізована за допомогою EBLearn C ++ пакету з відкритим кодом [10]. Однією з переваг згорткових мереж є те, що їх можна запускати на дуже високій швидкості на недорогих паралельних апаратах малого форм-фактора на базі програмованих мікросхем або графічних процесорів. Вбудовані системи, засновані на мікросхемах, можуть запускати великі згорткові мережі у реальному часі [11], відкриваючи можливість виконання кількох завдань

машинного зору одночасно із загальною інфраструктурою. Згорткова мережа пройшла навчання з повним наглядом за кольором зображення набору даних, використаних у даній роботі, і досягла точності 98,97% на наборі випробувань фази 1. Після закінчення фази 1 додаткові експерименти із зображеннями в сірих тонах встановили новий рекорд точності: 99,17%.

Архітектура мережі, використана в цій роботі, відходить від традиційної згорткової мережі за типом використовуваних нелінійностей, використанням з'єднань, які пропускають шари, та використанням об'єднання шарів з різними коефіцієнтами субдискретизації для з'єднання, які пропускають шари.

Звичайні згорткові мережі організовані в суворих шаруватих архітектурах, в яких подається вихід одного шару лише до шару вище. Натомість вихід першого етапу розгалужується і подається до класифікатора, на додаток до вихід другого етапу (рис.). Вихід першого етапу після об'єднання / субдискретизації виористовується швидше, ніж у попередніх дослідженнях таких мереж. Крім того, застосування другої субдискретизації дало вищу точність, ніж лише з однією. Тому розгалужена 1 виходи на першому етапі більш вибірковий, ніж у традиційних згорткових мереж, але загалом проходить таку саму кількість проб (4x4) ніж 2 виходи другого ступеня. Поєднання представлення з декількох етапів у класифікаторі було зроблено для надання класифікатору різної шкали рецептивних полів. У разі 2 стадій ознак, друга стадія витягує “глобальні” та інваріантні форми та структури, тоді як перша сцена витягує “місцеві” мотиви з більш точними деталями. Було продемонстровано підвищення точності використання такого пропускного шару з'єднання.

У традиційних згорткових мережах нелінійний шар просто складається з точково-сигмовидної функції. Однак останнім часом з'являються більш досконалі нелінійні модулі, що отримують більш високу точність, особливо у ситуаціях з малими навчальними набірами розмірного режиму [9]. Ці нові нелінійні модулі включають точкову виправлену сигмоїдну функцію, за якою відбувається субтрактивна локальна нормалізація, та розділювальна локальна нормалізація. Локальна нормалізація операції була зроблена на основі візуальних моделей

нейронауки [13,14]. Операція віднімання нормалізації для даної ділянки обчислюється за формулою:

$$v_{ijk} = x_{ijk} - \sum_{ipq} w_{pq} \cdot x_{i,j+p,k+q} \quad (3.1)$$

Pwpq – вікно зважування Гауса, нормоване так, що

$$\sum_{ipq} w_{pq} = 1 \quad (3.2)$$

Розділювальна нормалізація обчислюється за формулою

$$y_{ijk} = \frac{v_{ijk}}{\max(c,\sigma_{jk})}, \text{де } v_{ijk} = (\sum_{ipq} w_{pq} * v^2_{i,j+p,k+q})^{1/2} \quad (3.3)$$

Для кожної вибірки константа c встановлюється як середнє значення в експерименті. Знаменник – це зважений стандарт відхилення всіх ознак від просторового сусідства.

Мережа складається з 3-х згорткових шарів – розмір ядра становить 3×3, з подвоєнням глибини на наступному шарі - з використанням функції випрямлення як функції активації, кожен з яких виконується операцією об'єднання максимумів 2×2. Останні 3 шари повністю зв’язані, остаточний шар дає 43 результати (загальна кількість можливих міток), обчислених за допомогою функції активації, яка є функцією пошуку максимум. Мережа навчається з використанням міні-пакетного стохастичного градієнтного спуску за допомогою оптимізатора. Було створено високомодульну інфраструктуру кодування, яка дозволяє динамічно створювати моделі, як у наступних фрагментах коду.

```
mc_3x3 = ModelConfig(EdLeNet,
"EdLeNet_Norm_Grayscale_3x3_Dropout_0.50", [32, 32, 1], [3, 32, 3],
[120, 84], n_classes, [0.75, 0.5])
mc_5x5 = ModelConfig(EdLeNet,
"EdLeNet_Norm_Grayscale_5x5_Dropout_0.50", [32, 32, 1], [5, 32, 2],
[120, 84], n_classes, [0.75, 0.5])

me_g_norm_drpt_0_50_3x3 = ModelExecutor(mc_3x3)
me_g_norm_drpt_0_50_5x5 = ModelExecutor(mc_5x5)
```

Рисунок 3.3 - Фрагмент коду з використанням функції для створення моделі

ModelConfig містить інформацію про модель, таку як:

- функція моделі (наприклад, EdLeNet)
- назва моделі
- формат введення (наприклад, [32, 32, 1] для відтінків сірого),
- конфігурація згорткових шарів (розмір фільтра, глибина початку, кількість шарів),
- повністю зв’язані розміри шарів (наприклад, [120, 84])
- кількість класів
- значення відсотків, що випадають, [p-conv, p-fc]

ModelExeuctor відповідає за навчання, оцінку, прогнозування та створення візуалізації карт активації.

Щоб краще ізолювати моделі та переконатися, що всі вони не існують під одним графіком Tensorflow, використовується така корисна конструкція, як на рисунку 3.4.

```
self.graph = tf.Graph()
with self.graph.as_default() as g:
    with g.name_scope( self.model_config.name ) as scope:

...

with tf.Session(graph = self.graph) as sess:
```

Рисунок 3.4 Фрагмент коду для ізолювання моделей

Таким чином, створюються окремі графіки для кожної моделі, переконуючись у відсутності змішування наших змінних, заповнювачів тощо. Це значно полегшує процес тренування.

Спочатку було обрано величну згорткової глибини 16, але кращі результати було отримано із значенням з 32, тому було вирішено використовувати його. Також було порівняно кольори та відтінки сірого, стандартні та нормалізовані зображення, і визначено, що відтінки сірого мають тенденцію перевершувати колір. На жаль, точність тестового набору на моделях 3x3 або 5x5 становила щонайбільше 93%, при цьому будучи дуже непостійною. Більше того, було виявлено деяку нестабільну поведінку втрат на наборі перевірки після певної кількості епох, що насправді означало, що модель переобладнала навчальний набір, а не узагальнила. Нижче приведено деякі дані з метричних графіків для різних конфігурацій моделі.

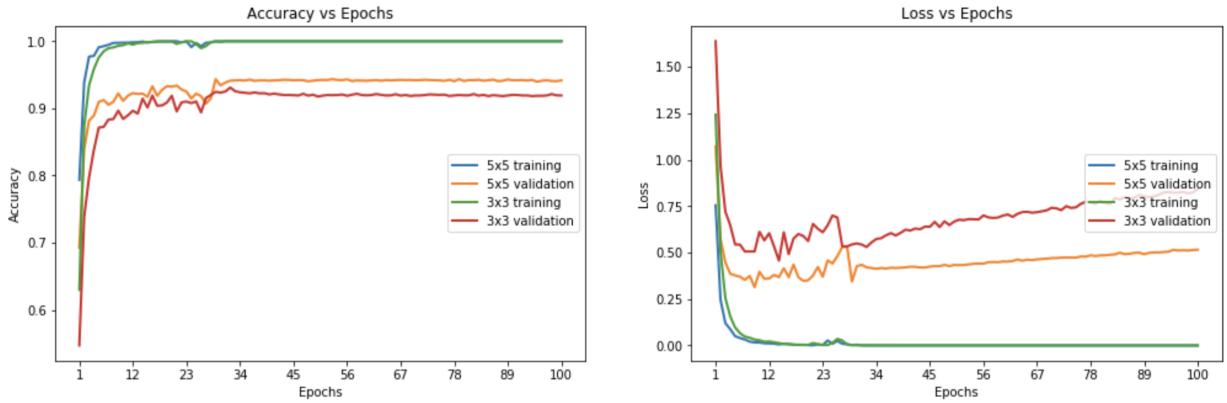



Рисунок 3.5 - Результати, які показала натренована модель на кольорових зображеннях

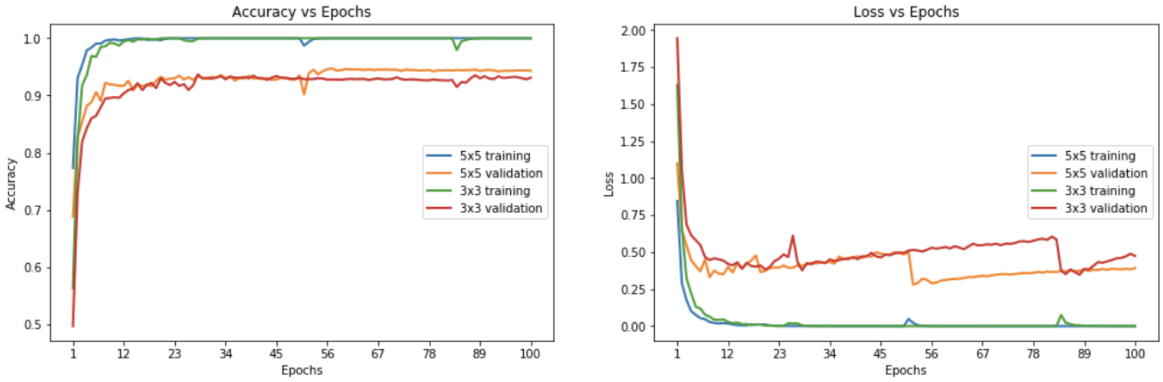



Рисунок 3.6 - Результати моделі на чорно білих зображеннях

З метою покращення надійності моделі було використано методику відсіювання, яка є формою регуляризації, коли ваги зберігаються з імовірністю: таким чином, «неупущені ваги» «падають». Це запобігає перенапруженню моделі. Алгоритм відсіювання було представлено Джеффрі Хінтоном, піонером у просторі глибокого навчання. Даний алгоритм має захоплюючу паралель з біологією та еволюцією.

У поясненні алгоритму автори застосовують різний ступінь відсіву, залежно від типу шару. Тому було вирішено застосувати подібний підхід, визначивши два рівні відсіву, один для згорткових шарів, другий для повністю зв'язаних шарів. Більше того, автори поступово приймають більш агресивні (тобто нижчі) значення відсіву, коли вони заглиблюються в мережу. Тому було вирішено, що

*p-conv >= p-fc*.

Тобто, було вирішено тримати ваги з більшою або рівною ймовірністю у згорткових, ніж повністю зв'язаних шарах. Причини цього полягають в тому, що мережа розглядається як воронка, і тому потрібно поступово затягувати її, просуваючись глибше в шари. Не потрібно відкидати занадто багато інформації на самому початку, оскільки частина цього буде надзвичайно цінна. Крім того, оскільки застосовується MaxPooling у згорткових шарах, втрачається трохи інформації вже на цьому етапі.

Було випробувано різні параметри, але врешті-решт було визначено, що p-conv = 0,75 та p-fc = 0,5, що дозволило досягти точності тестового набору 97,55%

на нормалізованих зображеннях у градаціях сірого за допомогою моделі 3×3. Цікаво, що було досягнуто понад 98,3% точності набору перевірок.

```
Training EdLeNet_Norm_Grayscale_3x3_Dropout_0.50 [epochs=100,
batch_size=512]...

[1]     total=5.222s | train: time=3.139s, loss=3.4993, acc=0.1047 |
val: time=2.083s, loss=3.5613, acc=0.1007
[10]    total=5.190s | train: time=3.122s, loss=0.2589, acc=0.9360 |
val: time=2.067s, loss=0.3260, acc=0.8973
...
[90]    total=5.193s | train: time=3.120s, loss=0.0006, acc=0.9999 |
val: time=2.074s, loss=0.0747, acc=0.9841
[100]   total=5.191s | train: time=3.123s, loss=0.0004, acc=1.0000 |
val: time=2.068s, loss=0.0849, acc=0.9832
Model ./models/EdLeNet_Norm_Grayscale_3x3_Dropout_0.50.chkpt saved
[EdLeNet_Norm_Grayscale_3x3_Dropout_0.50 - Test Set]
time=0.686s, loss=0.1119, acc=0.9755
```

Рисунок 3.8 - Приклад коду з використанням коефіцієнтів, які дали найкращий результат

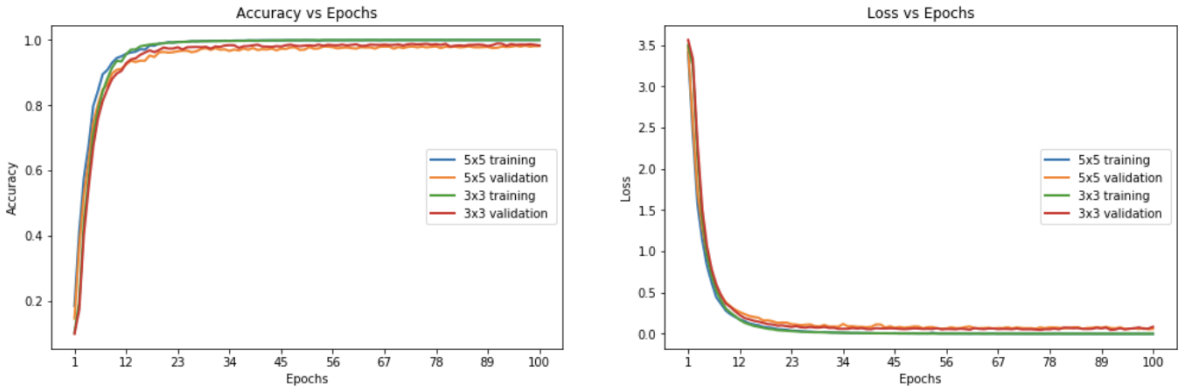



Рисунок 3.9 Графік успішності моделі на чорно білих зображеннях після використання відсіювання

Наведені графіки показують, що модель гладка, на відміну від деяких графіків вище. Було досягнуто мети набрати понад 93% точності на тестовому наборі, але чи можливо зробити це краще? Варто пам’ятати, що деякі зображення були розмитими, а розподіл зображень за класами був дуже нерівномірним. Нижче розглянуто додаткові прийоми, які було використано для вирішення кожного

питання.

Вирівнювання гістограми – це техніка комп'ютерного зору, що використовується для збільшення контрасту на зображеннях. Оскільки деякі зображення страждають від низького контрасту (розмито, темно), було покращено видимість за допомогою функції адаптивного вирівнювання гістограми OpenCV з обмеженням контрастності (вона ж CLAHE).

Було ще раз протестовано різні конфігурації та знайдено найкращі результати з точністю випробувань 97,75% на моделі 3×3, використовуючи такі значення відсіву:

$$\mathrm{p-conv} \;=\; 0{,}6, \mathrm{p-fc} \;=\; 0{,}5 \tag{3.4}$$

```
Training EdLeNet_Grayscale_CLAHE_Norm_Take-2_3x3_Dropout_0.50
[epochs=500, batch_size=512]...


[1]     total=5.194s | train: time=3.137s, loss=3.6254, acc=0.0662 |
val: time=2.058s, loss=3.6405, acc=0.0655
[10]    total=5.155s | train: time=3.115s, loss=0.8645, acc=0.7121 |
val: time=2.040s, loss=0.9159, acc=0.6819
...
[480]   total=5.149s | train: time=3.106s, loss=0.0009, acc=0.9998 |
val: time=2.042s, loss=0.0355, acc=0.9884
[490]   total=5.148s | train: time=3.106s, loss=0.0007, acc=0.9998 |
val: time=2.042s, loss=0.0390, acc=0.9884
[500]   total=5.148s | train: time=3.104s, loss=0.0006, acc=0.9999 |
val: time=2.044s, loss=0.0420, acc=0.9862
Model ./models/EdLeNet_Grayscale_CLAHE_Norm_Take-
2_3x3_Dropout_0.50.chkpt saved
[EdLeNet_Grayscale_CLAHE_Norm_Take-2_3x3_Dropout_0.50 - Test Set]
time=0.675s, loss=0.0890, acc=0.9775
```

Рисунок 3.10 - Приклад коду який видав найвищу точність

Нижче наведено графіки попередніх запусків, де також було протестовано модель 5×5, протягом 220 епох. Тут можна побачити набагато більш плавну криву, що підсилює гіпотезу того, що модель є більш стабільною.

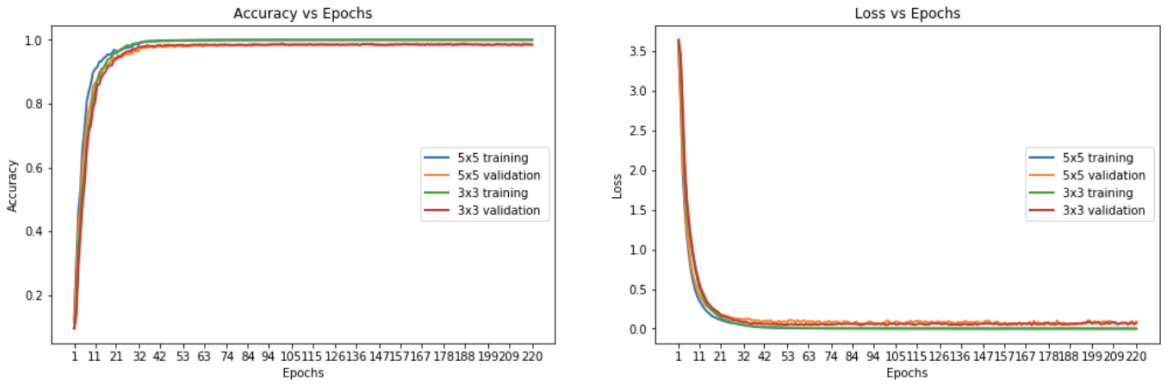



Рисунок 3.11 - Графік успішності моделі на зображеннях з вирівняною гістограмою

Було виявлено 269 зображень, які моделі не можуть ідентифікувати правильно. Нижче приведено 10 із них, вибраних випадковим чином, щоб здогадатися, чому модель помилилася.

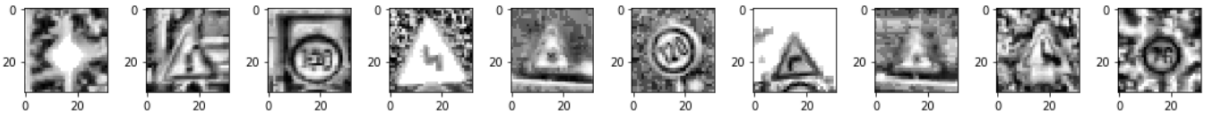


Рисунок 3.12 - Десять випадкових зображень із набору, які модель не змогла розпізнати

Деякі зображення дуже розмиті, незважаючи на вирівнювання гістограми, тоді як інші здаються спотвореними. Напевно, у тестовому наборі недостатньо прикладів таких зображень для покращення прогнозів моделі. До того ж, хоча 97,75% точності тесту є дуже хорошою, існує ще один метод покращення результатів: розширення даних.

Раніше було визначено, що дані представляють яскравий дисбаланс у 43 класах. Попри це було досягнуто дуже високої точність, незважаючи на дисбаланс класів. Також було визначено, що деякі зображення в тестовому наборі спотворені. Тому було використано методи розширення даних наступним чином:

1. Було згенеровано і додано додаткові зображення в різних параметрах освітлення та орієнтації.

2. Було покращено здатність моделі стати більш загальною.
3. Було покращено точність тестування та перевірки, особливо на спотворених зображеннях.

Для створення нових зображень із існуючих було використано бібліотеку imgaug. В основному було застосовано геометричні трансформації для збільшення зображень.

```
def augment_imgs(imgs, p):
    """
    Performs a set of augmentations with with a probability p
    """
    augs =  iaa.SomeOf((2, 4),
          [
              iaa.Crop(px=(0, 4)), # crop images from each side by 0
to 4px (randomly chosen)
              iaa.Affine(scale={"x": (0.8, 1.2), "y": (0.8, 1.2)}),
              iaa.Affine(translate_percent={"x": (-0.2, 0.2), "y":
(-0.2, 0.2)}),
              iaa.Affine(rotate=(-45, 45)), # rotate by -45 to +45
degrees)
              iaa.Affine(shear=(-10, 10)) # shear by -10 to +10
degrees
          ])


    seq = iaa.Sequential([iaa.Sometimes(p, augs)])

    return seq.augment_images(imgs)
```

Рисунок 3.13 - Приклад коду для збільшення кількості зображень за допомогою трансформації

Хоча дисбаланс класів, ймовірно, спричинив деяку упередженість у моделі, було вирішено не вирішувати її на цьому етапі, оскільки це призвело б до того, що набір даних суттєво збільшився б, як і час тренувань. У межах даної роботи не було можливості тренувати модель довше. Натомість було вирішено збільшити кожен клас на 10%. Новий набір даних мав такий вигляд, як показано на рисунку 3.14.


Рисунок 3.14 - Приклад зображень, згенерованих за допомогою трансформації

Звичайно, розподіл зображень суттєво не змінився, але було застосовано зміну відтінків сірого, рівняння гістограми та етапи попередньої обробки для кожного зображення. Було натреновано 2000 епох з відсівом (p-conv = 0,6, p-fc = 0,5) і досягнуто 97,86% точності на тестовому наборі.

```
[EdLeNet] Building neural network [conv layers=3, conv filter
size=3, conv start depth=32, fc layers=2]
Training
EdLeNet_Augs_Grayscale_CLAHE_Norm_Take4_Bis_3x3_Dropout_0.50
[epochs=2000, batch_size=512]...

[1]     total=5.824s | train: time=3.594s, loss=3.6283, acc=0.0797 |
val: time=2.231s, loss=3.6463, acc=0.0687
...
[1970]  total=5.627s | train: time=3.408s, loss=0.0525, acc=0.9870 |
val: time=2.219s, loss=0.0315, acc=0.9914
[1980]  total=5.627s | train: time=3.409s, loss=0.0530, acc=0.9862 |
val: time=2.218s, loss=0.0309, acc=0.9902
[1990]  total=5.628s | train: time=3.412s, loss=0.0521, acc=0.9869 |
val: time=2.216s, loss=0.0302, acc=0.9900
[2000]  total=5.632s | train: time=3.415s, loss=0.0521, acc=0.9869 |
val: time=2.217s, loss=0.0311, acc=0.9902
Model
./models/EdLeNet_Augs_Grayscale_CLAHE_Norm_Take4_Bis_3x3_Dropout_0.5
0.chkpt saved


[EdLeNet_Augs_Grayscale_CLAHE_Norm_Take4_Bis_3x3_Dropout_0.50 - Test
Set]    time=0.678s, loss=0.0842, acc=0.9786
```

Рисунок 3.15 - Приклад коду з використанням усіх описаних методів, з використанням моделі, нетренованої на штучно збільшеному сеті

### 3.3 Тестування на нових зображеннях

Було проведено тестування моделі на нових зображеннях, щоб переконатися, що вона дійсно узагальнена на більше, ніж дорожні знаки у початковому наборі даних. Тому було завантажено п’ять нових зображень і подано їх у натреновану модель для розпізнавання.


Рисунок 3.15 - Нові дорожні знаки, передані у натреновану модель

Дані зображення мають наступне значення:

1. Обмеження швидкості 120 км / год.
2. Головна дорога.
3. Заборонено рух транспортних засобів.
4. Дорожні роботи.
5. Проїзд транспортні засобів вагою понад 3,5 тонн заборонений.

Зображення були обрані з таких причин:

1. Вони представляють різні дорожні знаки, які класифікуються на даному етапі.
2. Вони різняться за формою та кольором.
3. Вони знаходяться в різних умовах освітлення (четвертий має відбиття сонячного світла).
4. Вони знаходяться під різною орієнтацією (3-й похилий).
5. Вони мають різне походження.
6. Деякі з них знаходяться в недостатньо представлених класах.

Першим кроком було застосування алгоритмів обробки зображення до цих нових зображень, що призвело до результатів, зображених на рисунку 3.16.

Рисунок 3.16 - Нові зображення після обробки

Всі нові зображення було класифіковані з точністю 100%. На оригінальному тестовому наборі було досягнуто точності 97,86%. Можна було б дослідити розмиття / спотворення нових зображень або модифікацію контрасту, щоб побачити, як модель обробляє ці зміни в майбутньому.

У додатку 1 показано 5 найкращих ймовірностей SoftMax, розрахованих для кожного зображення, із зеленою смужкою, що відображає істину. На графіках чітко видно, що модель цілком впевнена у своїх прогнозах.

### 3.4 Візуалізація карт активації моделі

Карти активації моделі – це те, як модель бачить зображення на кожному з етапів розпізнавання. Завдяки цим картам можна спробувати зрозуміти, як саме працює модель та на основі чого вона проводить класифікацію. На жаль, зрозуміти модель, створену в рамках даної роботи, дуже важко. Якщо розглянути карти активації для останнього зображення, можна припустити, що на першому етапі модель фокусується на межах кола знаку. Схоже, що вона намагається визначити грані знаку.

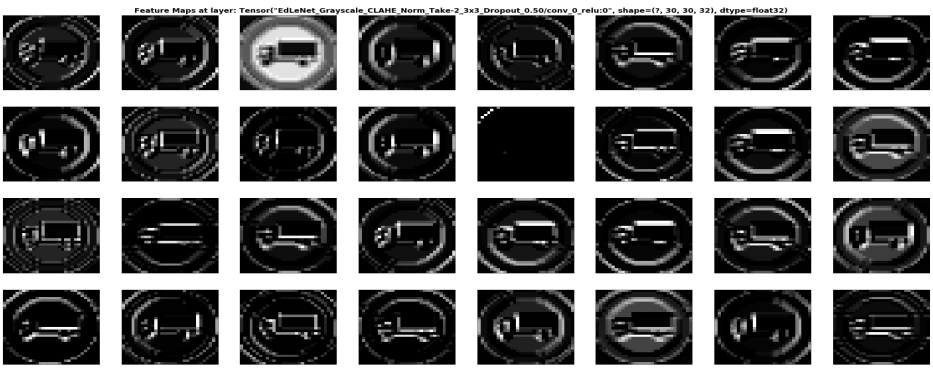



Рисунок 3.17 - Візуалізація карти активації першого шару моделі

Набагато важче визначити, на чому фокусується модель на другому етапі, але можна припустити, що це знову краї кола і частково центр знаку.

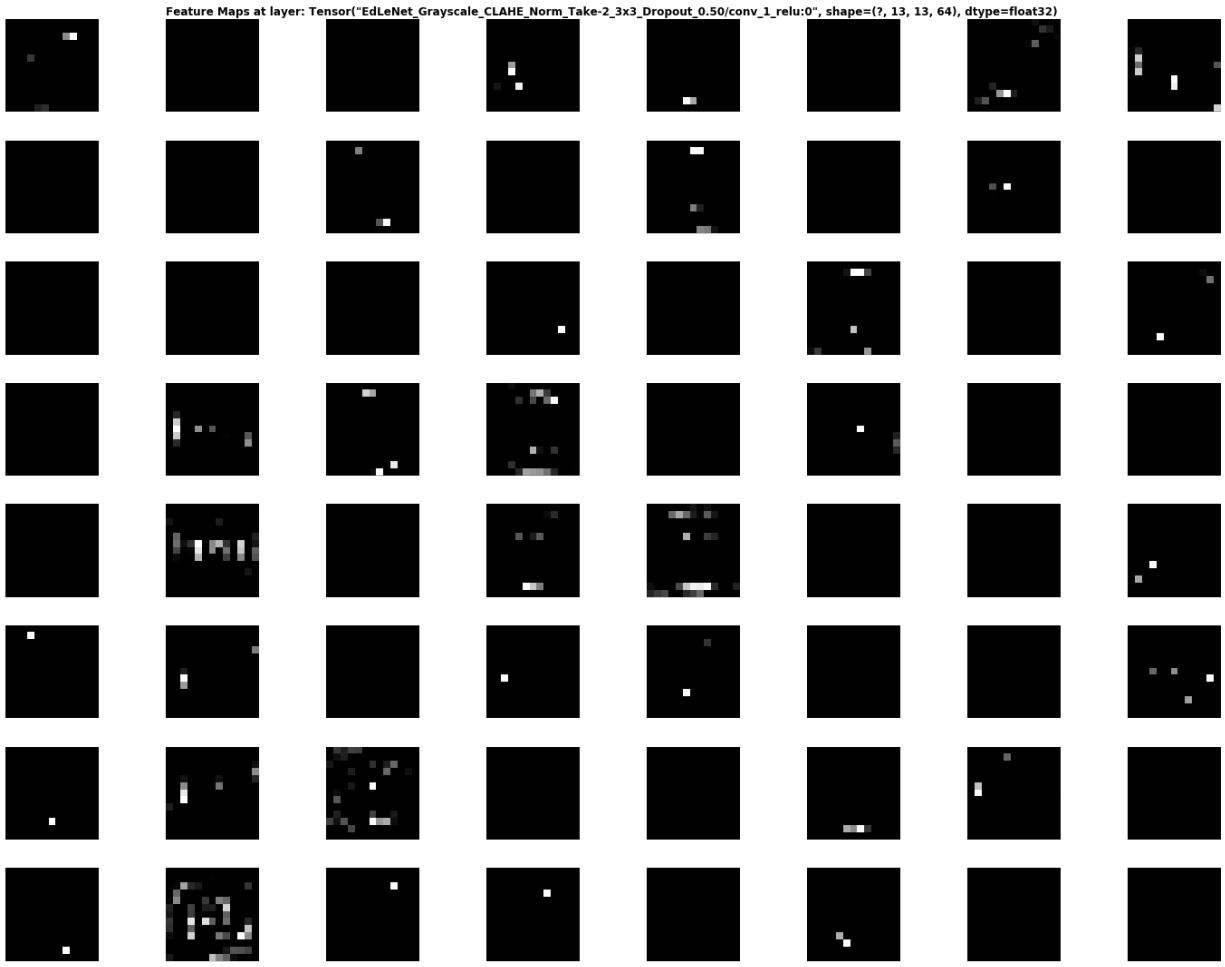



Рисунок 3.18 - Візуалізація карти активації моделі на другому етапі

Визначити, що цікавить модель на третьому етапі, практично неможливо. Можна припустити, що це знову центр і краї знаку, але це лише припущення.

Загалом, зрозуміти, як працює модель і на основі чого вона робить припущення, дуже важко. Вона схожа на чорний ящик, який працює незрозумілим чином, але видає правильні результати. Не дарма візуалізації карт активації чорні - це щось та й означає.

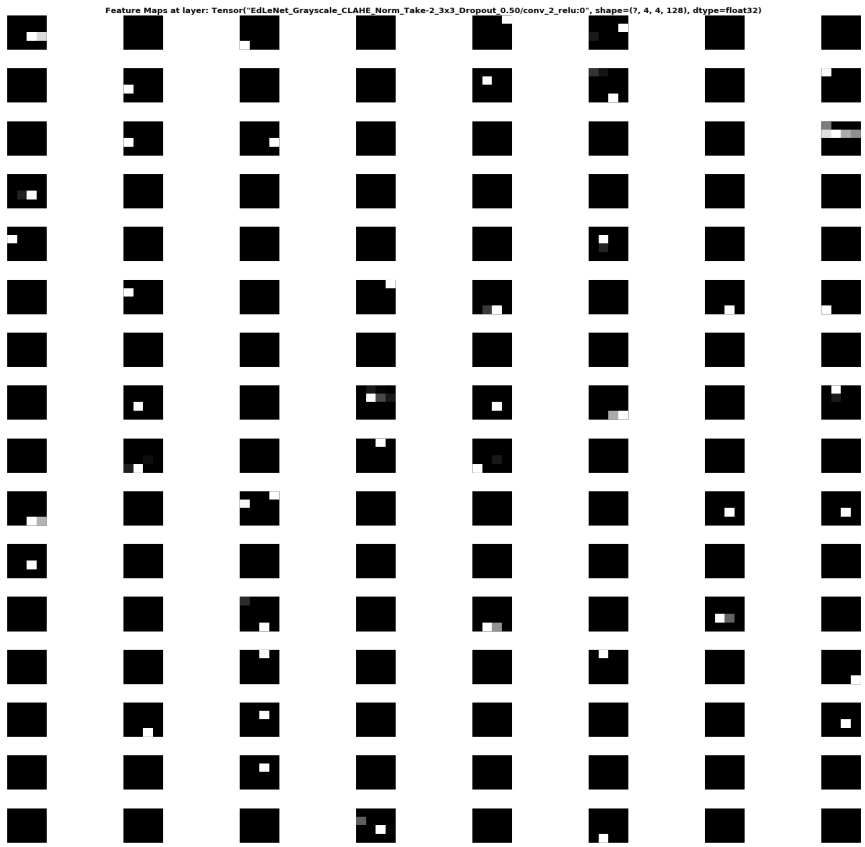



Рисунок 3.19 - Візуалізація третього етапу розпізнавання

### 3.5 Підсумки дослідження способів розпізнавання знаків

Під час прикладного дослідження було підтверджено, що класичний комп’ютерний зір недостатньо ефективний у розпізнаванні дорожніх знаків. Таким способом вдалося досягнути 75% точності. Також було перевірено згорткові нейронні мережі з використанням машинного навчання на колекції з близько 4000 зображень і виявлено, що такий підхід дозволить розпізнавати знаки з точністю близько 98%. Крім того, для коректної роботи першого способу на мобільній платформі IOS, для якої потрібно розробити систему розпізнавання знаків, потрібно реалізувати алгоритм комп’ютерного бачення майже з нуля, так як існуючі алгоритми спроектовані для інших завдань. У той час як для роботи другого способу потрібно лише підключити офіційну бібліотеку від TensorFlow. Таким чином, другий спосіб точніший і легший у реалізації.

# 4 РЕАЛІЗАЦІЯ ДОДАТКУ ДЛЯ IOS

## 4.1 Визначення стеку технологій

Додатки для iOS можуть бути створені за допомогою наступних мов програмування: Objective-C, Swift, C# ( Xamarin framework ), JavaScript ( Reactive Native framework ). Для розробки додатку було обрано мову програмування Swift, розроблену компанією Apple та викладену у вільний доступ. Вона має наступні переваги перед іншими:

- простіший синтаксис, ніж у Obj-C, що значно прискорює розробку;
- компілюється в байт-код напряму, а не через прив'язки до Obj-C, як C# і JS;
- реалізація функціональної та об'єктно-орієнтованої парадигм;
- це сучасна мова програмування, розроблена в Apple для внутрішнього використання під час розробки додатків для iOS, і опублікована у вільний доступ;
- наявність особливих корисних функцій, за замовчуванням відсутніх у інших мовах.

Обрана архітектура – Apple MVC. Вона проста у використанні і надзвичайно ефективна для розробки iOS додатків з невеликою кількістю екранів.

Для перекладу слів було обрано Google Translate API. Він надає переклад слів та словосполучень. Google API показали себе як надійні ресурси, на які можна покласти критично важливий функціонал, із справедливою ціною. Google постійно робить великі кроки у розвитку машинного перекладу, робить його кращим кожного дня. Саме тому було віддано перевагу Google API.

Для отримання визначення, синонімів, антонімів, прикладів використання та іншої інформації про слова було обрано WordsAPI. Це надійний сервіс, що надає доступ до бази із понад 300 000 слів англійської мови. Даний API достатньо простий у використанні і недорогий. Основним конкурентом WordsAPI є Oxford Dictionaries API, який використовується у додатку Amazon Kindle. Було обрано перший через те, що у нього краща інфраструктура, аналітика і більше

безкоштовних запитів кожного місяця.

Для керування залежностями проекту було обрано менеджер залежностей Cocoapods. Swift Package Manager ще не достатньо добре розвинений і непопулярний, а Carthage працює дещо інакше. Cocoapods компілює лише необхідні файли під час кожної збірки. Для пришвидшення процесу він зберігає максимальну кількість даних в кеш. Він дуже легко налаштовується і швидко компілює залежності, на відміну від Carthage, який довго налаштовується і компілює абсолютно всі залежності один раз за командою розробника. Оскільки залежностей даного проекту не повинно бути багато, було обрано Cocoapods.

Для використання нейронних мереж було обрано фреймворк від Apple CoreML, вбудований в операційну систему iOS. Він дозволяє використовувати моделі для різноманітних задач, таких як класифікація, розпізнавання, передбачення дій користувача та інших. Також він дозволяє тренувати існуючу модель на основі рішень, прийнятих користувачем, патернів його поведінки.

Apple також розробила фреймворк для роботи з відео, який називається AVFoundation. Багато ключових функцій та можливостей AVFoundation стосуються відтворення та обробки медіа-ресурсів. Фреймворк моделює активи за допомогою класу AVAsset, який є абстрактним, незмінним типом, що представляє один медіаресурс. Він забезпечує складене уявлення про медіа-актив, що моделює статичні аспекти медіа в цілому. Екземпляр AVAsset може представляти локальні носії на основі файлів, такі як фільм QuickTime або аудіофайл MP3, а також може представляти актив, що поступово завантажується з віддаленого хосту або передається за допомогою HTTP Live Streaming (HLS). Цей фреймворк було використано для отримання фото дороги під час розпізнавання дорожніх знаків.

# 5 ВИВЧЕННЯ ТА АНАЛІЗ РОЗРОБКИ ДОДАТКІВ З НЕЙРОННИМИ МЕРЕЖАМИ ДЛЯ IOS

## 5.1 Розробка додатків для iOS

З релізом iOS 2.0 у 2008 році компанія Apple відкрила доступ до iOS Software Development Kit (SDK). Це набір фреймворків, що дозволяють створювати сторонні додатки для IOS. У першій версії він мав мало можливостей і засобів, але з розростанням і вдосконаленням операційної системи збільшувався і набір фреймворків та бібліотек, створених для вирішення типових проблем. Так, наприклад, через рік після релізу системного менеджера паролів розробникам надали доступ до API даного продукту, за допомогою якого можна зробити інтеграцію з менеджером, підключивши до нього власне сховище паролів.

Загалом сучасна розробка мобільних додатків часто полягає у використанні інструментів, наданих розробниками операційної системи або сервісів, які потрібно використати у додатку, за допомогою API. Для рішення простих типових завдань потрібно зробити лише декілька викликів готових методів, для рішення більше складних – внести необхідні зміни до готової системи або набору інструментів і використати їх модифікований варіант.

У цьому розділі наводяться теоретичні відомості про основні частини iOS додатку та механізми, розроблені Apple для максимально швидкого створення надійних додатків, які повністю відповідають рекомендаціям розробників iOS.

## 5.2 Основний об’єкт додатку

Робота додатку починається із класу AppDelegate. Це клас, який реалізовує протокол (так у мові Swift називають інтерфейси) UIApplicationDelegate. Об’єкт цього класу створюється після того, як додаток завантажується в оперативну пам’ять. Одразу після цього система починає викликати методи делегата, чим сигналізує про змін стану додатку. Життєвий цикл цього об’єкта тісно пов’язаний з об’єктом UIApplication. Це означає, що він існує лише в одному екземплярі і

завжди ініціалізований під час виконання програми. У той час, як UIApplication керує всім, що відбувається «під капотом», куди розробник не має доступу, AppDelegate керує поведінкою додатку через реалізацію методів UIApplicationDelegate.

Таблиця 5.1 – Стани додатку IOS

| Стан | Опис стану |
|---|---|
| Не запущено | Додаток не був запущений або був завершений користувачем чи системою |
| Неактивний | Програма працює на передньому плані, але не отримує подій. (Можливо, виконується інший код.) Програма зазвичай залишається в цьому стані лише коротко, оскільки переходить в інший стан.<br>Після входу в цей стан програма повинна припинити діяльність та очікувати короткого переходу до фону або активного стану. |
| Активний | Програма працює на передньому плані та отримує події. Це нормальний режим для програм переднього плану.<br>Програма в активному стані не містить спеціальних обмежень. Це програма повинна відповідати користувачеві. |
| Задній план (background) | Додаток виконує код, але не відображається на екрані. Коли користувач виходить з програми, система на короткий час переміщує програму до стану заднього плану, перш ніж призупинити її. В інший час система може запустити додаток у фоновому режимі (або розбудити призупинену програму) і дати йому час для обробки конкретних завдань. |
| Призупинений | Програма знаходиться в пам'яті, але не виконує код.<br>Система призупиняє роботи програм, які знаходяться у фоновому режимі, і не мають завершених завдань.<br>Система може чистити призупинені програми в будь-який час, не пробуджуючи їх, щоб звільнити місце для інших програм. |

Першим викликається метод, який позначає завершення запуску додатку. У ньому розробник повинен ініціалізувати такі модулі, як аналітика та авторизація користувача (якщо необхідно), показати початковий інтерфейс або відновити попередній стан. На цьому ж етапі потрібно почати процес увімкнення системи пуш-сповіщень, якщо вона використовується. Крім того, в цей метод передаються аргументи, які містять інформацію про те, яким чином додаток було запущено, що дає можливість налаштувати поведінку і виконати необхідні операції. Наприклад, якщо користувач запустив додаток за допомогою сповіщення, розробник має можливіть в першу чергу показати розгорнуту версію даних, що зацікавили користувача, або виконати дію, описану в сповіщенні.

Об’єкт класу AppDelegate отримує сповіщення про зміну стану додатку через імплементацію відповідних методів. Залежно від стану, додаток має виконувати різні дії. У таблиці 2.1 перелічені всі можливі стану додатку та його очікувана поведінка.

На рисунку 2.1 показано, як змінюються стани програми під час її роботи.

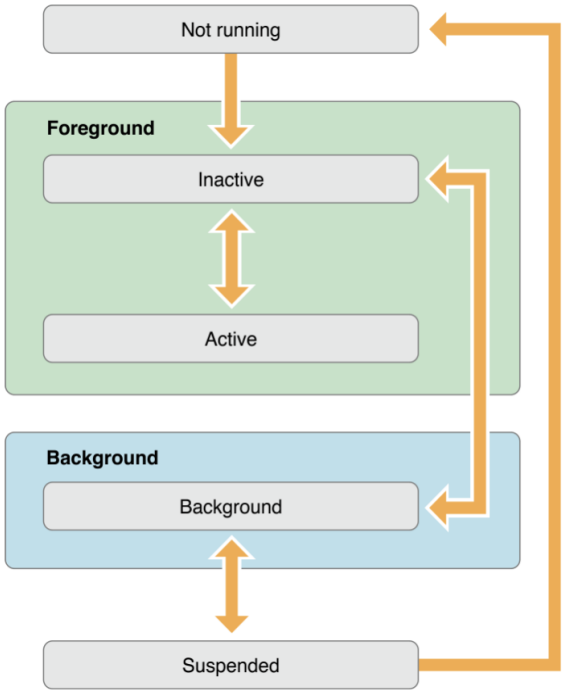



Рисунок 2.1 – Зміна станів додатку

### 5.3 Взаємодія додатку з файловою системою

У системі iOS кожен додаток працює в інкапсульованій частині пам’яті, яка називається «пісочниця». Завдяки цьому жоден інший додаток не має доступу до пам’яті, що знаходиться за межами його «пісочниці». Таким чином підтримується високий рівень захищеності додатків. Жоден додаток від стороннього розробника не має прямого доступу до файлів, що зберігаються на пристрої. Натомість, існують засоби для того, щоб за ініціативою користувача отримати доступ до одного чи декількох файлів, які він обере. Для різних типів файлу, які потрібно імпортувати в додаток, існують різні інструменти. Наприклад, для імпорту документів використовується UIDocumentPickerViewController. Це клас, який містить системний інтерфейс для вибору документу із файлової системи та імпорту його в додаток. У нього є різні режими роботи, які описані в таблиці 2.2.

Таблиця 5.2 – Режими роботи з файлами за межами «пісочниці»

| Режим | Коротка характеристика |
|---|---|
| Імпорт документа зовні | Користувач вибирає зовнішній документ. Система копіює документ, залишаючи оригінал незмінним |
| Експорт документа за межі додатку | Користувач вибирає зовнішній пункт призначення. Система копіює документ, залишаючи оригінал незмінним |
| Відкриття зовнішнього документа | Користувач вибирає зовнішній документ. Система надає доступ до документа, і користувач може редагувати документ на місці |
| Переміщення локального документа | Користувач вибирає зовнішній пункт призначення. Система переміщує документ, однак користувач може отримати доступ до нього як до зовнішнього, і може редагувати документ на місці |

Як операції відкриття, так і переміщення надають доступ до документів поза

межами «пісочниці» програми. Цей доступ надає користувачам безпрецедентну гнучкість при роботі з документами. Тим не менш, він також додає рівень складності для обробки файлів. Зовнішні документи мають такі додаткові вимоги:

1. Операції відкриття та переміщення забезпечують URL-адреси для безпечного доступу до зовнішніх документів. Виклик метода startAccessingSecurityScopedResource() надає доступ до цих документів і метод stopAccessingSecurityScopedResource() закриває його.
2. Потрібно завжди використовувати NSFileCoordinator для читання та запису зовнішніх документів.
3. Для відображення вмісту зовнішнього документа потрібно використовувати NSFilePresenter.
4. Не потрібно зберігати посилання, що надаються операціями відкриття і переміщення документів.

Для того, щоб отримати результати вищеописаних операцій, потрібно імплементувати клас-делегат UIDocumentPickerViewController. Наприклад, для імпорту крижки потрібно реалізувати метод documentPicker(_: didPickDocumentsAt:), параметрами якого є посилання на вибрані користувачем файли, які можна скопіювати у «пісочницю». Створеному контролеру потрібно просто присвоїти делегат, а потім показати цей контролер на екрані.

### 5.4 «Пісочниця» додатку

Розробник має можливість керувати сховищем даних додатку так, як йому зручно. Для отримання посилання на папку документів додатку потрібно викликати метод класу FileManager, який виконує всі операції керування файлами в додатку. У полі FileManager.default знаходиться стандартний об’єкт менеджера, який задовольняє більшість потреб розробника. Для отримання посилання на внутрішню директорію додатку достатньо викликати метод urls(for: in: ), який приймає тип директорії, на яку потрібно отримати посилання. Після цього робота з «пісочницею» відбувається за допогою методів запису та зчитування менеджера.

Розробник також може створювати директорії для організації файлів зручним

способом. Система автоматично створює папку Documents, але це не означає, що розробник зобов’язаний її використовувати.

Під час визначення розташування файлів можна використовувати об'єкти URL або String. Використання класу URL, як правило, є кращим для визначення елементів файлової системи, оскільки URL-адреси можуть конвертувати інформацію про шляхи до більш ефективного внутрішнього представлення. Крім того, об'єкту класу URL дозволяє бути впевненим у тому, що це справді посилання на файл, тоді як String може зберігати будь-який текст, який не обов’язково є правильним посиланням.

Для переміщення, копіювання, зв'язування або видалення файлів та каталогів, можна використовувати делегат спільно з об'єктом менеджера файлів для керування цими операціями. Роль делегатів полягає в тому, щоб підтвердити операцію і вирішити, в якій ситуації потрібно видати помилку. Делегат повинен відповідати протоколу FileManagerDelegate.

Якщо ж стандартного менеджера та його делегата недостатньо і потрібно реалізувати специфічні операції з файлами, можна створити власний об'єкт менеджера і кастомізувати його за допомогою властивостей. Або ж створити власний підклас FileManager та реалізувати конкретні методи по-своєму.

### 5.5 Синхронізація даних між пристроями

У iOS 5.0 і пізніших версіях FileManager включає в себе методи керування елементами, що зберігаються в iCloud. Файли та каталоги з тегами для хмарних сховищ синхронізуються з iCloud, щоб вони могли бути доступними для інших пристроїв iOS користувача. Зміни елемента в одному місці поширюються на всі інші місця, щоб забезпечити синхронізацію елементів. Даний функціонал дозволяє синхронізувати прогрес користувача в грі, бібліотеку книжок чи пісень, між його телефоном і планшетом. Для того, щоб записати дані у хмарне сховище, потрібно лише отримати дозвіл від користувача, отримати URL від FileManager і, власне, записати файл у потрібну директорію.

Важливою частиною синхронізації через хмарне сховище iCloud є те, що всі дані

користувачів знаходяться у повній безпеці, надійно захищені від взломів.

До того ж, використання саме iCloud для синхронізації даних є досить недорогим і не вимагає настройки та менеджменту власних серверів.

### 5.6 Збереження даних

Існує декілька способів зберігати необхідні дані в середині додатку. Найпростіший полягає у використанні вищеописаних засобів для запису даних у файли, наприклад, у форматі json. Це чудове рішення для зберігання кешу, яке дозволяє швидко очистити чи перезаписати його. Однак інколи виникає потреба зберігати складні об’єкти у великій кількості, керувати та модифікувати їх без втрати продуктивності додатку. Для цього можна використати системний фреймоврк Core Data, що дозволяє створити локальну базу даних та виконувати всі необхідні операції з нею.

Середовище розробки Xcode містить редактор моделей, за допомогою якого можна створити моделі об’єктів, типи взаємодії між ними та відповідні класи швидко і легко. Під час виконання додатку фреймворк керує цими об’єктами та надає наступні функції:

1. Постійність – Core Data відображає деталі зіставлення об'єктів із сховищем, що полегшує збереження даних із Swift та Objective-C без безпосереднього керування базою даних.
2. Відміна та повторення операцій індивідуально або групами – менеджер скасування відстежує зміни та може відмінити їх окремо, у групах або всі одночасно, що полегшує додавання відміни та повторення операцій до програми.
3. Робота у фоновому режимі – важкі або блокуючі операції можна виконувати у фоновому режимі.
4. Синхронізація інтерфейсу – Core Data може підтримувати таблиці та колекції у актуальному стані завдяки прив’язці до них.
5. Версії та міграції – фреймворк містить функціонал для керування версіями моделей та міграції з однієї версії на іншу.

Кожна модель є підкласом NSManagedObjectModel. Для моніторингу змін моделі створюється об’єкт NSManagedObjectContext, а для керування моделями потрібен NSPersistantStoreCoordinator, який зберігає та зчитує моделі із сховища. Для керування вищеперерахованими об’єктами використовується NSPersistentContainer. Варто зазначити, що цей клас спроектований для того, щоб розробник створив його підкласи, в які помістив додаткову логіку керування моделями. Проте ніщо не заважає використати його реалізацію за замовчуванням, якщо вона задовольняє всі потреби.

Приємним бонусом є той факт, що всі дані, збережені в Core Data, синхронізуються між пристроями користувача автоматично, без втручання розробника. Кожна зміна даних веде за собою якнайшвидшу синхронізацію із захищеним хмарним сховищем.

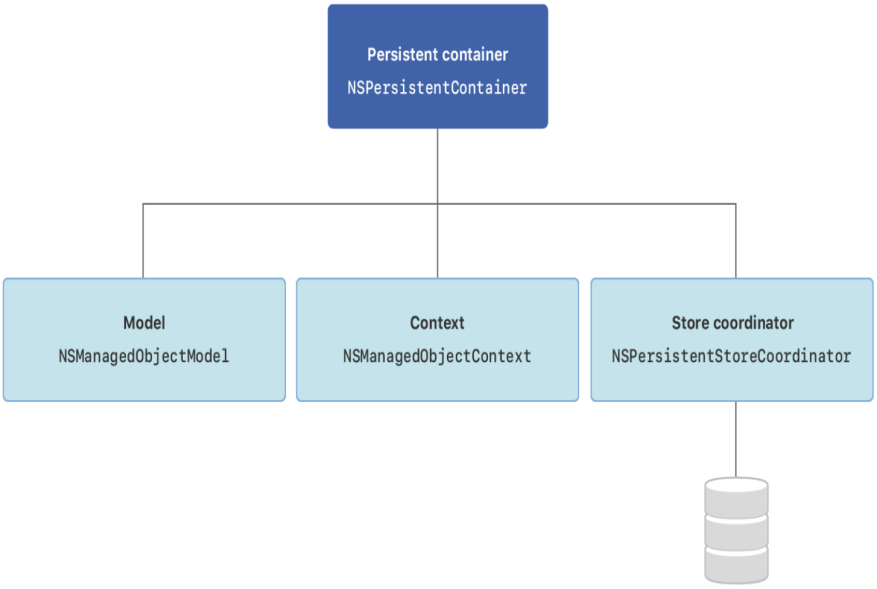



Рисунок 5.2 – Основні об’єкти Core Data

Взаємодію з фреймворком схематично позначено на рисунку 2.2.

### 5.7 Доступ до інтернету з додатку iOS

Для отримання доступу до мережі в iOS використовується клас URLSession.

Він та пов’язані з ним інші класи надають функціонал для завантаження даних з інтернету та їх відправлення. Наприклад, за допомогою об'єкту URLSession створюється один або декількох екземплярів URLSessionTask, які можуть отримувати та повертати дані у програму, завантажувати файли або завантажувати дані та файли у мережу. Щоб налаштувати сеанс, використовується об'єкт URLSessionConfiguration, який керує поведінкою, наприклад, кеш-файлів та файлів cookie.

Для створення завдань можна повторно використовувати один сеанс. Наприклад, веб-переглядач може мати окремі сеанси для звичайного та приватного перегляду, де приватні сеанси не кешують його дані. На рисунку 2.3 показано, як дві сесії з цими конфігураціями можуть створювати кілька завдань.

Для швидкого завантаження даних доступна сесія за замовчуванням, проте у неї немає можливостей для конфігурації.

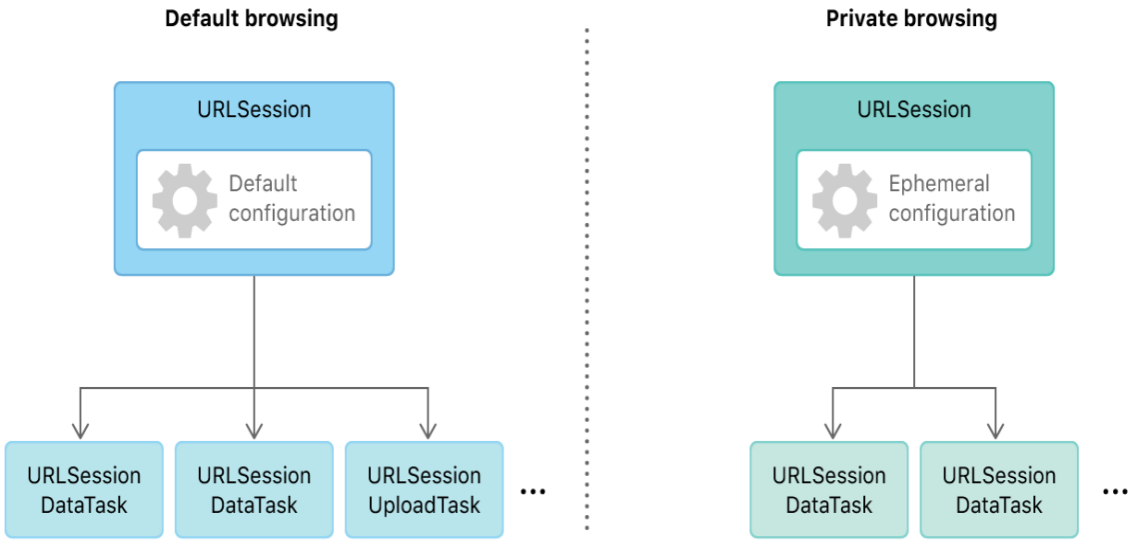



Рисунок 5.3 – Створення сесій для різних типів веб-перегляду

У таблиці 5.3 наведено типи конфігурацій сесії та їх короткий опис.

Таблиця 5.3 – Типи конфігурацій сесії

| Тип | Опис |
|---|---|
| Стандартна сесія | Об’єкт синглтон, що задовольняє мінімальні базові потреби, але не може бути сконфігурований розробником. |
| Звичайна сесія | Загалом поводиться так само, як стандартна, але дозволяє більше конфігурації і дозволяє отримувати дані з делегатом. |
| Ефемерна сесія | Подібна до звичайної, але не записує кеш, файли cookie або облікові дані на диск. |
| Фонова сесія | Фонові сесії дозволяють виконувати відвантаження в мережу та завантаження даних у фоновому режимі, поки програма не запущена. |

Сесії налаштовуються за допомогою об’єкта URLSessionConfiguration. Під час завантаження даних в або з мережі створення об'єкта конфігурації завжди є першим кроком. Цей об'єкт використовується для налаштування значень тайм-ауту, політики кешування, вимог підключення та інших типів інформації, які використовуються з об'єктом URLSession.

Важливо правильно налаштувати ваш об'єкт URLSessionConfiguration, перш ніж використовувати його для ініціалізації об'єкта сеансу. Об'єкти сеансу створюють копію наданих налаштувань конфігурації та використовують ці параметри для налаштування сеансу. Після налаштування об'єкт сесії ігнорує будь-які зміни, внесені до об'єкта URLSessionConfiguration. Якщо потрібно змінити політику перенесення, необхідно оновити об'єкт конфігурації сеансу і використовувати його для створення нового об'єкта URLSession.

Кожен сеанс пов'язаний з делегатом для отримання періодичних оновлень або помилок. Делегат за замовчуванням викликає блок обробника завершення, який надає розробник; якщо ж розробник вирішить надати власний власний делегат, цей блок не викликається.

Можна налаштувати сеанс, щоб він працював у фоновому режимі, таким чином в той час, як програму призупинено, система може завантажувати дані від свого імені та пробуджувати програму, щоб отримати результати.

Як було сказано вище, завантаження відбувається за допомогою об’єктів URLSessionTask. У сесії створюється завдання, які додатково завантажують дані на сервер, а потім отримують дані з сервера або як файл на диску, або як один або більше об'єктів NSData в пам'яті.

API URLSession надає три типи завдань:

1. Завдання для отримання даних типу Data, призначені для швидких запитів до сервера, на які отримують коротку відповідь. Ця відповідь зберігається в оперативну пам’ять.
2. Завдання для завантаження даних на сервер подібні до попередніх, але вони також надсилають дані (часто у вигляді файлу) і підтримують фонові завантаження, поки програма не запущена.
3. Завдання для завантаження даних у вигляді файлу, що отримують великі відповіді від сервера та записують їх у файли, а також підтримують завантаження та вивантаження фонових зображень під час запуску програми.

Як і більшість фреймворків для роботи з мережею, URLSession є асинхронним. Він надає результати викликів методів за допомогою блоків коду, що опрацьовують завершення операції, або через методи делегату. Окрім передачі цієї інформації до делегатів, API URLSession надає властивості статусу та прогресу, які можна перевіряти, якщо потрібно зробити програмні рішення на основі поточного стану завдання. Сесії URL також підтримують скасування, перезавантаження, відновлення та призупинення завдань, а також надають можливість відновити призупинене, скасоване чи невдале завантаження, коли вони зупинилися.

URLSession нативно підтримує data, file, ftp, http, and https URL схеми, з прозорою підтримкою проксі-серверів і шлюзів SOCKS, як налаштовано в системних налаштуваннях користувача. URLSession підтримує протоколи HTTP / 1.1, SPDY і HTTP / 2. Можна також додати підтримку власних мережевих протоколів і схем URL-адрес (для особистого користування програми) за

допомогою підкласу URLProtocol.

Для безпечного користування інтернетом iOS додатки за замовчуванням не мають можливості відправляти незахищені http запити. Можна увімкнути таку можливість вручну, проте Apple не рекомендує робити цього, адже це наражає користувачів на небезпеку і викликає недовіру до додатку, що використовує незахищене з'єднання.

### 5.8 Робота додатку в фоновому режимі

IOS відома своєю оптимізацією і бережливим ставленням до ресурсів. Це частково зумовлено тим, як додатки працюють у фоновому режимі. Розробники Apple все зробили для того, щоб додаток не виконував зайвої роботи і не використовував ресурси без потреби. Коли користувач переходить в інший додаток, той додаток, що був активний, переходить у фоновий режим, щоб завершити операції, які у ньому відбувалися. Час для цього обмежений, і коли він закінчується, додаток переходить у призупинений стан. Існує декілька типів роботи у фоновому режимі. Їх короткий опис наведено у таблиці 5.4.

Кожен з цих режимів вмикається окремо у налаштуваннях дозволів додатку. Всі вони полягають у тому, що система бере під свій контроль певний процес, що відбувається в додатку, та дає йому можливість обробити результати виконання цього процесу.

Таблиця 5.4 – Типи роботи у фоновому режимі

| Назва | Короткий опис |
|---|---|
| Аудіо та AirPlay | Дає можливість відтворювати аудіо та транслювати його за допомогою AirPlay |
| Оновлення місцезнаходження | Призначений для відслідковування оновлень місцезнаходження. У цьому режимі додаток періодично отримує нові координати користувача та має можливість за короткий час виконати необхідні операції |
| Voice over IP | Використовується для інтернет дзвінків |
| Завантаження журналів/газет | Додаток може періодично завантажувати невеликі обсяги даних, наприклад, газети чи журнали, у фоновому режимі та оброблювати їх. |
| Зв’язок із зовнішніми аксесуарами | Використовується для підтримання зв’язку з аксесуарами |
| Використання BLE пристроїв | Режим для підтримки зв’язку з BLE аксесуарами, наприклад, пульсометром, за якого ініціатором комунікації є телефон |
| Виконання ролі BLE пристрою | Режим для підтримки зв’язку з BLE пристроями, за якого телефон не є ініціатором комунікації |
| Періодичне завантаження даних | Система із заданою періодичністю запускає додаток для завантаження нового контенту, якщо ресурсів достатньо. При включеному режимі енергозбереження додаток не буде запущено. |
| Отримання нотифікацій | Вмикається для того, щоб мати можливість обробити нотифікації та виконати необхідні операції |

Варто зазначити, що не завжди потрібно використовувати режим завантаження даних. Якщо завантаження даних у додатку відбуваються рідко і

лише за ініціативою користувача, для продовження завантаження у фоновому режимі можна використати метод AppDelegate, який викликається, коли додаток переходить із одного стану в інший. З цього методу можна викликати системний метод для продовження процесу у фоновому режимі. Таким чином процес отримає обмежений час для завершення свого завдання. Розробник може передати у цей метод блоки, які будуть викликані при успішному завершенні процесу або коли його час буде вичерпано і він завершиться примусово.

Існує спосіб ініціювати завантаження даних за допомогою пуш-сповіщення. Для цього потрібно увімкнути режим отримання нотифікацій у фоновому режимі і відправити із сервера сповіщення з ключем «content-available». Таким чином система визначить, що додатку необхідно завантажити дані з інтернету, і дасть можливість зробити це у фоновому режимі, виділивши для нього більше ресурсів. Проте варто пам’ятати,  що час роботи додатку у фоновому режимі строго обмежений з метою підтримання заряду телефону і якомога кращого досвіду користування системою. Не потрібно завантажувати дані і виконувати операції, без яких можна обійтися без шкоди для користувача.

### 5.9 Використання згорткових нейронних мереж у iOS додатку

Компанія Apple розробила власний фреймворк для нейронних мереж і машинного навчання Core ML. Моделі, які використовуються даним фреймворком, тренуються за допомогою інструментів Xcode (середовища розробки програм для iOS) і оптимізовані під сучасні процесори Apple для максимальної ефективності. У свою чергу, процесори Apple також оптимізовані під операції нейронних мереж, тому продуктивність даної системи набагато вища, ніж у інших смартфонів.

Модель є результатом застосування алгоритму машинного навчання до набору навчальних даних. Вона використовується для прогнозування на основі нових вхідних даних. Моделі можуть виконувати найрізноманітніші завдання, які було б важко або непрактично написати в коді. Наприклад, можна навчити модель класифікувати фотографії або виявляти конкретні об’єкти на фотографії безпосередньо з її пікселів.

Можна побудувати та навчити модель за допомогою програми Create ML у комплекті з Xcode. Моделі, навчені за допомогою Create ML, мають формат Core ML і готові до використання у додатку. Крім того, можна використовувати широкий спектр інших бібліотек машинного навчання, а потім використовувати Core ML Tools для перетворення моделі у формат Core ML. Коли модель знаходиться на пристрої користувача, існує можливість використовувати Core ML для перенавчання або тонкої настройки на пристрої з даними цього користувача.

Core ML оптимізує продуктивність на пристрої, використовуючи центральний процесор, графічний процесор та нейронні системи, мінімізуючи при цьому обсяг пам'яті та енергоспоживання. Запуск моделі строго на пристрої користувача усуває будь-яку потребу в мережному підключенні, що допомагає зберегти конфіденційність даних користувача.

Core ML – це основа для доменних фреймворків та функціональних можливостей. Core ML підтримує Vision для аналізу зображень, Natural Language для обробки тексту, Speech для перетворення звуку в текст і Sound Analysis для ідентифікації звуків в аудіо. Сама Core ML будується на основі низькорівневих примітивів, таких як Accelerate та BNNS, а також шейдерів графічного фреймворку Metal.

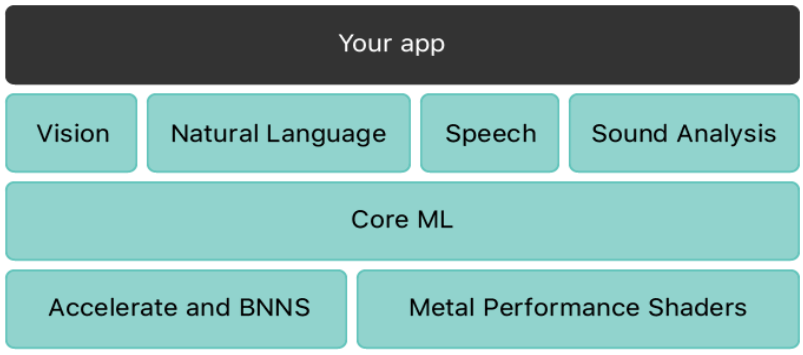



Рисунок 5.4 - Фреймворки та інструменти, доступні для використання з нейронними мережами

Сервіс TensorFlow від Google, який було використано для тренування моделей для розпізнавання дорожніх знаків, надає можливість експортувати

натреновану модель нейронної мережі для використання на інших платформах, у тому числі iOS. Для того, щоб конвертувати модель TensorFlow у модель Core ML, потрібно використати спеціальний фреймворк MLModelInterpreter. Він конвертує модель з одного формату в інший і робить її доступною для використання на iOS пристрої.

# 6 ДОСЛІДЖЕННЯ РЕКОМЕНДОВАНОЇ АРХІТЕКТУРИ ДЛЯ ДОДАТКІВ IOS З НЕЙРОННИМИ МЕРЕЖАМИ

## 6.1 Загальні відомості

Дизайн патерн Model-View-Controller (MVC) – це шаблон високого рівня, що стосується глобальної архітектури програми і класифікує об’єкти на загальні ролі, які вони виконують у додатку. Він є складним, оскільки містить у собі декілька більш елементарних структур. Патерн поділяє об’єкти згідно з їхніми ролями в додатку на три типи: модель, частина інтерфейсу користувача та контролер. Кожен із трьох типів відокремлений від інших абстрактними кордонами і зв’язується з іншими чітко визначеними способами.

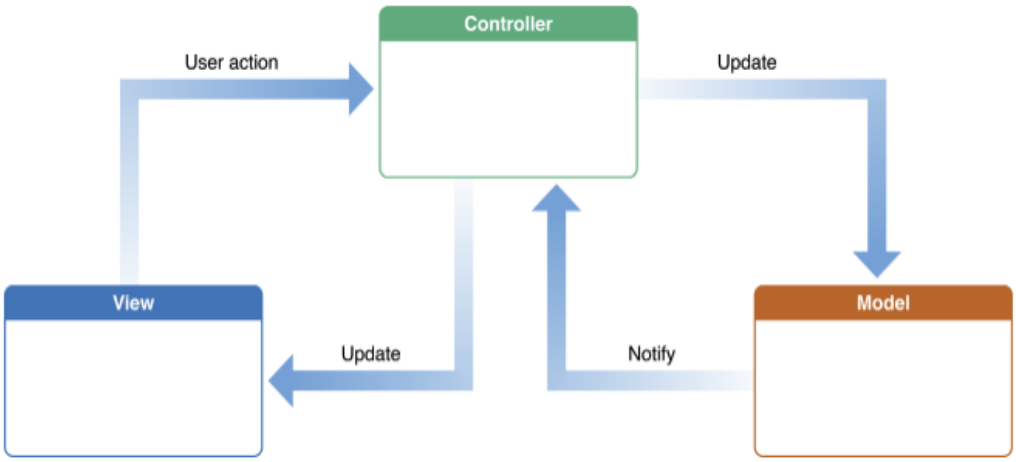



Рисунок 6.1 – Взаємодія між об’єктами патерну MVC

MVC займає центральне місце в хорошому дизайні IOS додатку. Існують численні переваги використання саме цієї архітектури при побудові додатку. Основними перевагами є те, що багато об’єктів у такому додатку використовуються повторно, їхні інтерфейси краще визначені, а самі програми досить легко розширюються. Крім того, більшість фреймворків IOS розроблені із застосуванням MVC і вимагають, щоб об’єкти користувача також відтворювали одну з ролей, представлених у патерні. На рисунку 6.1 наведено діаграму, яка показує типи об’єктів шаблону MVC та зв’язки між ними.

### 6.2 Об'єкти моделі

Об'єкти моделі інкапсулюють дані, специфічні для програми, і визначають логіку та обчислення, що обробляють ці дані та маніпуюють ними. Наприклад, об'єкт моделі може представляти символ у грі або контакт у адресній книзі. Дані зберігаються в полях (властивостях) об'єкту. Методи доступу та оголошені властивості забезпечують способи збереження інкапсуляції, оскільки вони опосередковують доступ до даних об'єкта. Методи доступу зазвичай отримують і встановлюють значення змінних екземплярів (і розмовно називаються методами getter і setter). Оголошеними властивостями є зручність на рівні мови програмування, яка дозволяє середовищі виконання синтезувати методи доступу для класу. Важлива роль методів доступу та оголошених властивостей полягає в управлінні пам'яттю об'єктів. З цієї причини існують рекомендовані форми для реалізації геттер-методів і методів встановлення. На рисунку 6.2 наведено графічний приклад об'єкту моделі, що деякі властивості та методи для отримання і зміни їх значення.

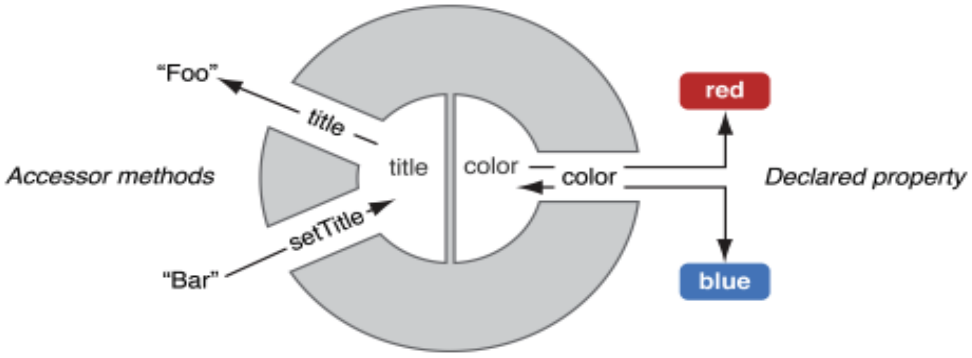



Рисунок 6.2 – Приклад об'єкту моделі

Об'єкт моделі може мати відношення один до одного і з багатьма іншими об'єктами моделі, і тому іноді модельний шар програми є одним або декількома графами об'єктів. Добре спроектований MVC додаток характеризується тим, що всі його важливі дані інкапсульовані в об'єкти моделі. Значна частина даних, які є частиною постійного стану програми (незалежно від того, що постійний стан зберігається у файлах або базах даних), повинна знаходитися в об'єктах моделі після завантаження даних у програму. Оскільки об'єкти моделі являють собою знання та досвід, пов'язані з певною проблематичною областю, вони можуть бути повторно використані в аналогічних доменах проблем. В ідеалі, об'єкт моделі не повинен мати чіткого підключення до об'єктів інтерфейсу, які представляють його дані і дозволяють користувачам редагувати ці дані - це не повинно стосуватися проблем інтерфейсу користувача та відображення даних. Одним з прикладів, коли виключення з цього правила є доцільним, є додаток для малювання, у якому є об'єкти моделі, які представляють графіку, відображену на екрані. Графічні об'єкти можуть знати, яким чином відображати себе на екрані, тому що основною причиною їхнього існування є визачення візуальної речі. Але навіть у цьому випадку графічні об'єкти не повинні покладатися на існування у певному об'єкті інтерфейсу і не повинні відповідати за зображення самих себе на екрані. Ці дії повинні бути ініційовані тим об'єктом інтерфейсу, який відображає ці моделі.

### 6.3 Об'єкти частини інтерфейсу користувача

Об'єктом інтерфейсу користувача є об'єкт, який користувачі можуть бачити. Він керує тим, як відображається на екрані і може реагувати на дії користувача. Основною метою таких об’єктів є відображення даних з об’єктів моделі програми та надання можливості редагувати ці дані. Об’єкт частини інтерфейсу може відображати як один об’єкт моделі, так і його частину, або навіть декілька різних моделей. Незважаючи на це, частини інтерфейсу повинні бути відокремлені від моделей в додатку MVC. Вони також не повинні керувати збереженням даних, які відображають, однак це не означає, що частини інтерфейсу ніколи не зберігають дані. Вони можуть зберігати їх у власний кеш або виконувати інші трюки для підвищення продуктивності та швидкості роботи. Об’єкти інтерфейсу зазвичай використовуються декілька разів у різних місцях та навіть різних додатках. Системний фреймворк UIKit містить багато класів, доступних для використання під час створення інтерфейсу. Таким чином розробник отримує гарантію, що кнопки в його додатку будуть поводити себе так само, як і в інших додатках, забезпечуючи таким чином високий рівень стабільності у зовнішньому вигляді та поведінці різних додатків. Звичайно, зовнішній вигляд таких об’єктів можна кастомізувати під загальний стиль додатку, але принцип їх роботи залишається однаковим у всій системі. На рисунку 3.3 наведено приклади деяких стандартних класів для об’єктів користувацького інтерфейсу.

Важливо зазначити, що такі об’єкти можуть бути вкладеними один в одного. Це дозволяє розділяти складні частини інтерфейсу на менші, які легше контролювати і використовувати в декількох місцях додатку.

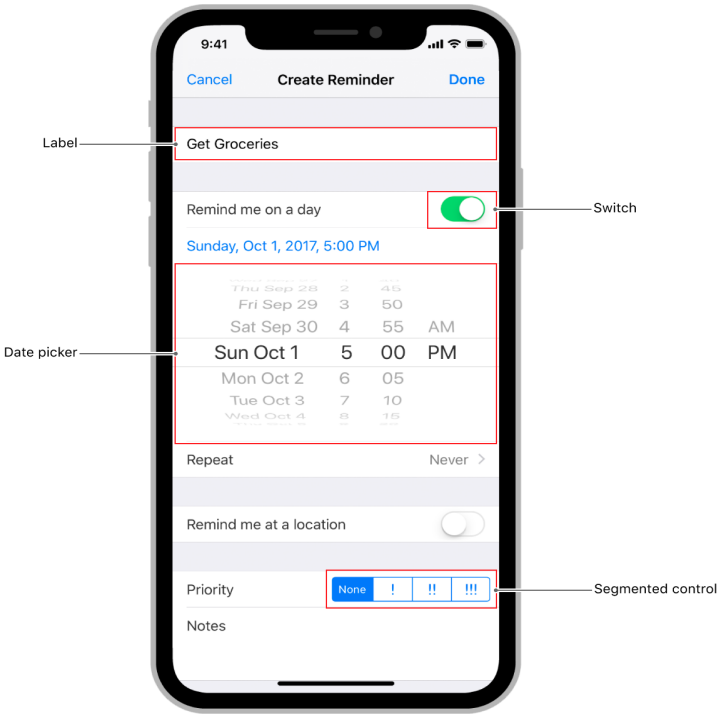



Рисунок 6.3 – Приклади системних об’єктів для побудови інтерфейсу

Об’єкт користувацького інтерфейсу повинен відображати актуальні дані моделі. Оскільки дані можуть динамічно змінюватися, потрібен спосіб для своєчасного оповіщення об’єкту інтерфейсу і оновлення даних на екрані. Оскільки модель не може напряму впливати на інтерфейс, інструментом для цього слугує контролер.

Більшість додатків iOS побудовані з використанням компонентів фреймворку UIKit, основи, що визначає загальні елементи інтерфейсу. Ця структура дозволяє програмам досягти уніфікованого вигляду всієї системи, водночас пропонуючи високий рівень налаштування. Елементи UIKit є гнучкими і знайомими. Вони адаптуються, дозволяючи розробляти одну програму, яка чудово виглядає на будь-якому пристрої iOS, і вони автоматично оновлюються, коли система вносить зміни у зовнішній вигляд.

### 6.4 Об’єкти контролери

Контролер є посередником між моделлю та об’єктом інтерфейсу користувача.

Саме на ньому лежить відповідальність за передачу оновлень між іншими

двома шарами додатку. Один контролер зазвичай передає інформацію про зміни між однією або декількома моделями і одним або декількома об’єктами інтерфейсу. Контролер реагує на дії користувача і оновлює моделі згідно з цими діями. Він також може керувати налаштуванням інтерфейсу та координацією завдань між різними модулями програми.

Попри таке призначення, контролер може бути більше прив’язаний до інтерфейсу, ніж до моделі, і навпаки. У UIKit входить декілька системних типів контролерів, які прив’язані до конкретного типу інтерфейсу. Вони створюють об’єкти інтерфейсу, контролюють їхній життєвий цикл і повністю керують ними. Так, наприклад, UITableViewController створюється для керування таблицею, а UICollectionViewController – для керування колекцією. Вони містять готову реалізацію взаємодії користувача з вказаними типами інтерфейсу, стандартні засоби для їх налаштування і контролю, які можна легко розшири власними. Це помітно прискорює розробку, так як розробнику потрібно писати менше коду для реалізації необхідних функцій. А оскільки кількість коду прямо впливає на кількість багів, це ще й робить додаток більш надійним і якісним. Кастомізувати системні елементи інтерфейсу зазвичай не важко, тому що вони мають всі необхідні властивості. Крім того, можна використати наслідування для додавання нових властивостей і перевизначення методів.

У свою чергу, контролер моделі відноситься більше до шару моделі, ніж інтерфейсу. Він є власником моделі, оновлює її властивості, працює з нею набагато більше, ніж з інтерфейсом. Прикладом такого контролеру в системі є NSDocument, який автоматично виконує всі дії для збереження файлів

Варто зазначити, що контролер може оперувати не лише об’єктами інтерфейсу (view), але й іншими контролерами. Так, UIPageController створено для керування декількома контролерами, де кожен контролер – це окрема сторінка, UINavigationController – для переходів між контролерами і навігації по екранам додатку, UITabBarController – для створення контролерів-вкладок і переходу між ними. Всі ці контролери широко використовуються у всій системі.

### 6.5 Об’єкти-координатори

Можна легко здогадатися, що часто в контролерах зосереджується занадто багато функцій. Чим складніший інтерфейс, тим більше розростаються ці об’єкти. Для вирішення даної проблеми якомога більше функцій контролера перекладаються на інші об’єкти-помічники. Наприклад, для завантаження даних із сервера створюється окремий об’єкт, який викликається з контролера. Для встановлення зв’язку між різними контролерами, кожен з яких представляє собою окремий екран додатку, можна використати об’єкт-маршрутизатор або навігатор. А якщо екранів багато, найкращим рішенням буде зібрати їх у окремі модулі, якими буде керувати об’єкт-координатор модуля, а навігатор буде передавати контроль від одного координатора іншому. Приклад такого модуля зображено на рисунку 3.4.

Для великих додатків можна навіть створювати дочірні координатори - або підкоординатори - які дозволяють відключити частину навігації програми. Наприклад, можна керувати потоком створення облікового запису за допомогою одного підкоординатора та керувати потоком підписки на продукт, використовуючи інший.

Якщо потрібна ще більша гнучкість, зв'язок між контролерами і координаторами може відбуватися за допомогою протоколу, а не конкретного типу. Це дозволяє замінити весь координатор у будь-який момент пізніше, і отримати інший потік програми - можна надати один координатор для iPhone, один для iPad і один для Apple TV, наприклад.

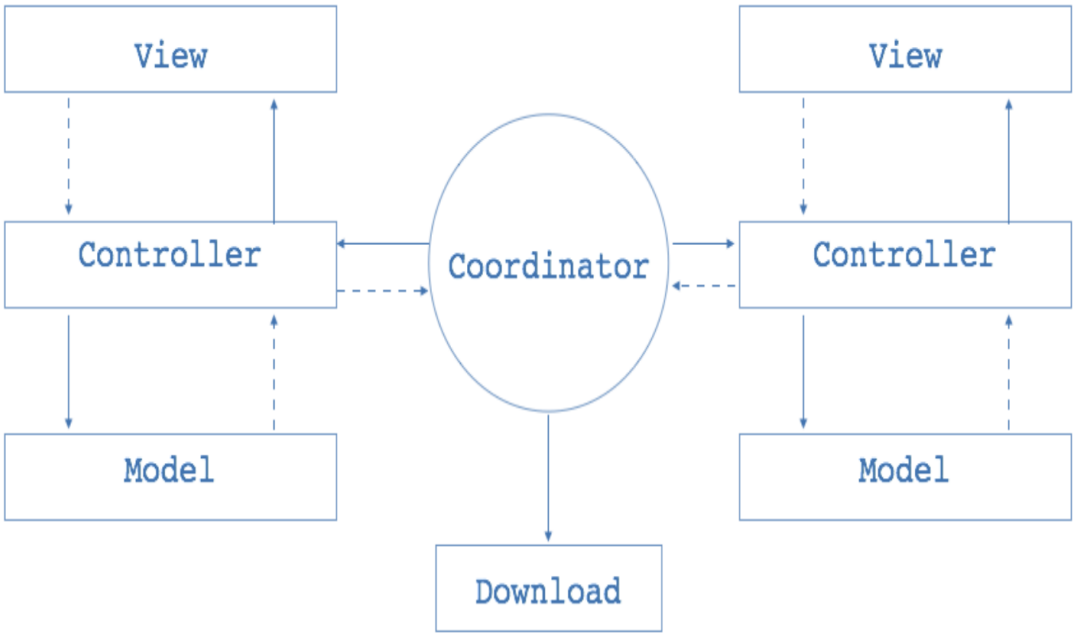



Рисунок 6.4 – Схема модуля з двома екранами

Іншим важливим кроком до впорядкування контролера є збереження стану інтерфейсу в окремій структурі і оновлення кожного елементу інтерфейсу окремими методами. Наприклад, у стані зберігаються така інформація, як спосіб сортування даних або слово пошуку. Завдяки тому, що кожна дія користувача викликає зміни в стані, які одразу ж відображаються на екрані за допомогою окремих методів, код досить легко читати і розширювати. До того ж, всі зміни відбуваються лише з одного місця в коді, що дозволяє легко знайти і виправити помилку.

### 6.6 Модифікація MVC – MVVM

Одним із варіантів рішення проблем MVC є альтернативна архітектура, яка є модифікацією існуючої MVC – MVVM ( Model-View-ViewModel ). Основна її відмінність полягає в тому, що контролер розглядається як частина View, тобто інтерфейсу, а для зв’язку з моделями створюється інший шар – ViewModel. На рисунку 3.5 схематично показані зв’язки між різними шарами.


Рисунок 6.5 – Зв’язки між шарами MVVM

Загалом, для невеликого проекту MVVM не має особливих переваг перед MVC. Для неї потрібні додаткові залежності, а ViewModel мало чим відрізняється від звичайного контролера і має ті самі проблеми.

### 6.7 VIPER

VIPER – (View, Interactor, Presenter, Entity, Router) дизайн патерн, що реалізовує парадигму розмежування приналежності. Він дотримується модульної структури, де для однієї глобальної функції створюється один модуль. Кожен модуль містить декілька об’єктів, що описані у таблиці 3.1

Таблиця 6.1 – Об’єкти архітектури VIPER

| Назва | Значення |
|---|---|
| View | Клас, який має весь код для показу інтерфейсу програми для користувача і отримання від нього відповідей. Після отримання відповіді він передає дані в Presenter |
| Presenter | Ядро модуля. Він отримує відповідь користувача з інтерфейсу та працює відповідно. |
| Interactor | Має бізнес-логіку програми. Перш за все виконує виклики API для отримання даних |
| Router | Виконує переходи між екранами. Отримає команди переходу від об’єкта Presenter |
| Entity | Зберігає класи моделей, що використовуює Ineractor |

Загалом, цей архітектурний патерн найкраще показує себе на великих проектах, над якими працює велика команда із 5 і більше розробників. Як патерн для невеликого проекту з одним розробником він завдає багато клопоту. Для кожного модуля тут потрібно створити як мінімум 5 об'єктів, які залежать один від одного, в той час як у модифікованому MVC достатньо 3 або 4. Через велику кількість залежностей розробка з використанням VIPER для одного програміста займе набагато більше часу.

### 6.7 Оптимальна архітектура

Було визначено, що найкраще підійде архітектура MVC з використанням так званих сервісів, які інкапсулюють бізнес логіку додатку. Так, наприклад, було створено сервіс SignsClasifier для аналізу зображення з камери телефону і розпізнавання знаків. Він використовує ImageProcessor для попередньої обробки зображень та SignsRecognizer для розпізнавання знаків за допомогою нейронної мережі. А для отримання цих зображень було розроблено сервіс CameraService, який при підключенні видає 2 зображення з камери в секунду. Нижче наведено діаграму архітектури додатку.

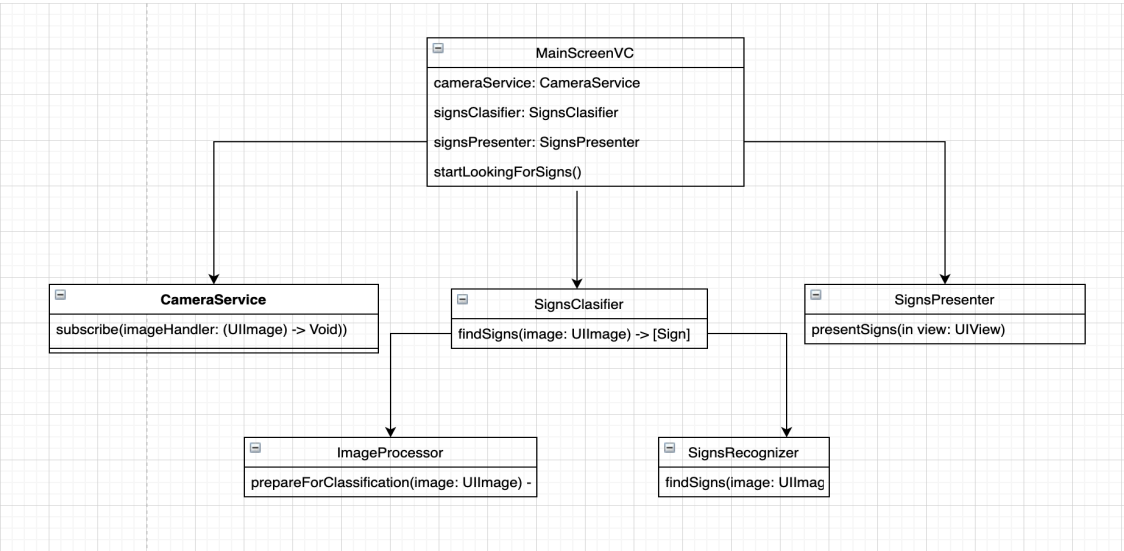



Рисунок 6.6 Архітектура додатку

# 7 РОЗРОБКА ДОДАТКУ ДЛЯ ТЕСТУВАННЯ

## 7.1. Проектування інтерфейсу

Для тестування розробленої моделі було зроблено простий додаток, який підключається до камери телефону, отримує нове зображення кожні 2 секунди і передає його в нейронну мережу для розпізнавання знаків. Було обрано частоту 2 кадри в секунду для спрощення роботи і оптимізації енергоспоживання додатку.

Було спроектовано максимально простий інтерфейс. Головний екран складається із зображення з камери, картинки знаку, яка з’являється, якщо система розпізнає знак, та надпису з назвою знаку, який також з’являється за потреби. Нижче приведено мокап такого екрану та його використання в додатку.


Рисунок 7.1 - Інтерфейс додатку для розпізнавання знаків

Як можна помітити, назва знаку з’являється на білому фоні для покращення її видимості.

### 7.2. Імплементація та тестування

Спочатку було реалізовано сервіс для захоплення зображення з камери телефону на базі стандартних фреймворків від Apple, таких як AVFoundation, Accelerate та Foundation. Розроблений сервіс було інтегровано в основний екран і підключено до інтерфейсу. Потім було розроблено сервіс для розпізнавання знаків на зображеннях з використанням фреймворку MLModelInterpreter від Google, щоб використати нейронну модель, натреновану за допомогою TensorFlow.

Для обробки зображень було спроектовано додатковий модуль, який включає в себе декілька бібліотек для різних задач. Кожен етап обробки було реалізовано окремо за допомогою сторонніх бібліотек, а сам процес обробки зібрано в одному місці

Для ручного тестування було обрано 20 знаків із різних класів. Додаток розпізнав усі знаки правильно. Для повної перевірки функціонування було імплементовано тест для модуля, який розпізнає знаки, з використанням тестового сету з 800 картинок. Було визначено, що точність на даному сеті становить 97.5%, так само, як і на серверах TensorFlow. Це означає, що фреймворк для трансляції моделі з формату TensorFlow у формат CoreML працює відмінно, без втрати результативності моделі.

Крім того, було проведено 5 інтерв’ю з потенційними користувачами, з яких 3 водії зі стажем більше двох років і 2 учнів автошколи. Потенційним користувачам було показано демо версію додатку, повідомлено про перспективи розвитку даних технологій. Один водій вважає додаток корисним, а обидва опитаних учні в захваті від ідеї.

# ВИСНОВКИ

У даній роботі було досліджено інформаційні системи для розпізнавання знаків. Було визначено, що нейронні мережі виконують завдання класифікації зображень краще, ніж класичні системи комп'ютерного зору. Таку мережу було адаптовано для використання на iOS і протестовано. Було експериментально встановлено, що така система може ефективно працювати на сучасних смартфонах, а отже, сприятиме безпеці на дорозі і пришвидшенню розповсюдження подібних систем.

Було визначено, що розроблена система допомагає не лише водіям, але й тим, хто навчається в автошколі для отримання водійського посвідчення, оскільки така система прискорить вивчення дорожніх знаків і може запобігти аваріям з участю тих, хто щойно отримав права.

Розроблена система має такі переваги перед аналогами:

1. Не потребує додаткового обладнання, бо у більшості водіїв є смартфони і кріплення для них.
2. Сумісна з будь-яким автомобілем.
3. Не потребує покупки нового дорогого автомобіля для встановлення.
4. Може розпізнавати більше знаків, ніж деякі існуючі системи.
5. Працює швидше, ніж деякі існуючі системи, бо використовує повну потужність сучасних смартфонів.
6. Під час тестування було виявлено деякі недоліки системи, зокрема:
7. Більшість кріплень телефону всередині автомобіля закривають камеру смартфону, що не дозволяє додатку працювати.
8. Під час користування додатком водії не можуть користуватися навігацією смарфтону.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Хайкин, С. Нейронные сети: полный курс / С. Хайкин. – М.: Вильямс, 2006. – 1104 с.

2. Shneier, M. Road sign detection and recognition // Proc. IEEE Computer Society Int. Conf. on
Computer Visionand Pattern Recognition. – 2005 – P. 215–222.

3. Oh, J.-T. Segmentation and Recognition of Traffic Signs Using Shape, Information / J.-T. Oh, H.-
W. Kwak, Y.-H. Sohn, W.-H. Kim // Segmentation and Recognition of Traffic Signs Using Shape, Information. Lecture Notes in Computer Science, Springer Berlin/Heidelberg. – 2005 – Vol. 3804. – P. 519 – 526.

4. Hardzeyeu, V. On using the Hough transform for driving assistance applications / Hardzeyeu, V., Klefenz, F., // Intelligent Computer Communication and Processing, 2008. ICCP 2008. 4th International Conference on , vol., no., pp.91,98, 28-30 Aug. 2008

5. LeCun Y., Bottou L., Bengio Y., Haffne P. Gradient-Based Learning Applied to Document Recognition // Proc. IEEE. – 1998. – P.59-67.

6. Круглов В.В. Искусственные нейронные сети. Теория и практика. / В.В. Круглов, В.В. Борисов.М.: Телеком, 2001

7. C. Fang, C. Fuh, S. Chen, and P. Yen, "A road sign recognition system based on dynamic visual model," presented at Proc. 2003 IEEE Computer Society Conf. Computer Vision and Pattern Recognition, Madison, Wisconsin, 2003.

8. C. Fang, S. Chen, and C. Fuh, "Road-sign detection and tracking," IEEE Trans. on Vehicular Technology, vol. 52, pp. 1329-1341, 2003

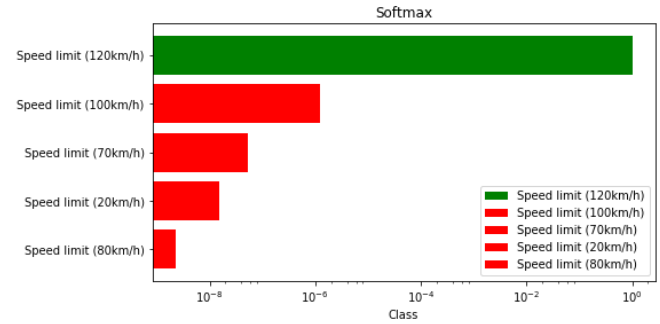
9. Domen Tabernik, Danijel Skocaj, “Deep Learning for Large-Scale Traffic-Sign Detection and Recognition”, Cornell University, 2019

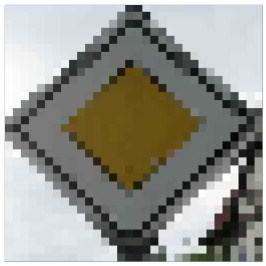
10. Hasan Fleyeh, “Color detection and segmentation for road and traffic signs”, Conference: Cybernetics and Intelligent Systems, 2004

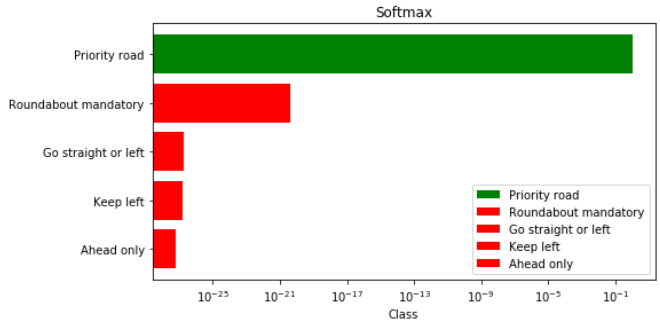
11. J. Miura, T. Kanda, and Y. Shirai, "An active vision system for real-time traffic sign recognition," presented at Proc. 2000 IEEE Intelligent Transportation Systems,

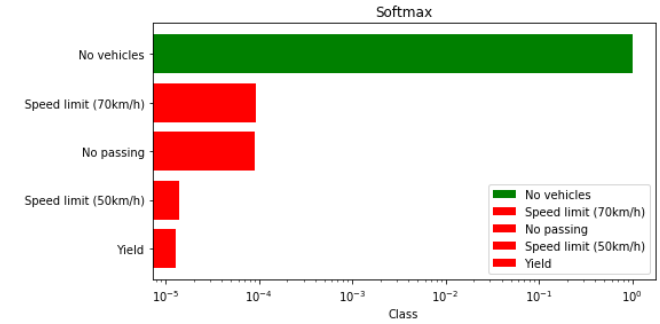
Dearborn, MI, USA, 2000.
12. P. Paclik, J. Novovicova, P. Pudil, and P. Somol, "Road sign classification using Laplace kernel classifier," Pattern Recognition Letters, vol. 21, pp. 1165-1173, 2000.
13. S. Vitabile, G. Pollaccia, G. Pilato, and E. Sorbello, "Road sign Recognition using a dynamic pixel aggregation technique in the HSV color space," presented at Proc. 11th Inter. Conf. Image Analysis and Processing, Palermo, Italy, 2001.
14. S. Vitabile, A. Gentile, and F. Sorbello, "A neural network based automatic road sign recognizer," presented at Proc. The 2002 Inter. Joint Conf. on Neural Networks, Honolulu, HI, USA, 2002.
15. A. de la Escalera, J. Armingol, and M. Mata, "Traffic sign recognition and analysis for intelligent vehicles," Image and Vision Comput., vol. 21, pp. 247-258, 2003.
16. P. Parodi, and G. Piccioli, "A feature-based recognition scheme for traffic scenes," presented at Proc. Intelligent Vehicles '95 Symposium, Detroit, USA, 1995.
17. M. Blancard, "Road Sign Recognition: A study of Vision-based Decision Making for Road Environment Recognition," in Visionbased Vehicle Guidance, I. Masaki, Ed. Berlin, Germany: Springer-Verlag, 1992, pp. 162-172.
18. A. Neubeck, L. Van Gool, “Efficient Non-Maximum Suppression”, 18th International Conference on Pattern Recognition, 2006
19. P. Paclik, and J. Novovicova, "Road sign classification without color information," presented at Proc. Sixth Annual Conf. of the Advanced School for Computing and Imaging, Lommel, Belgium, 2000.
20. A. Hanbury, and J. Serra, "A 3D-polar coordinate color representation suitable for image analysis," Computer Vision and Image Understanding, 2002.
21. J. Angulo, and J. Serra, "Color segmentation by ordered mergings," presented at Proc. Int. Conf. on Image Processing, Barcelona, Spain, 2003.
22. OpenCV Library documentation, available at: https://docs.opencv.org/

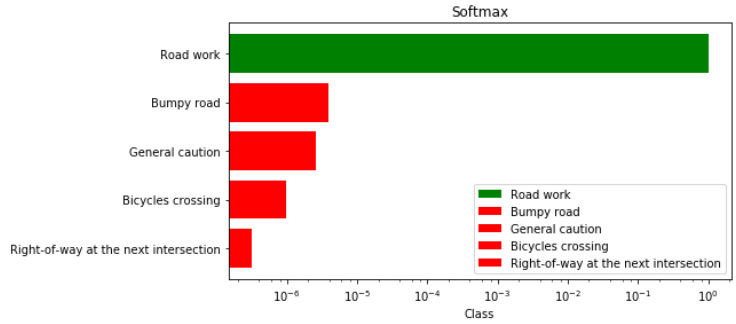
# Додаток 1

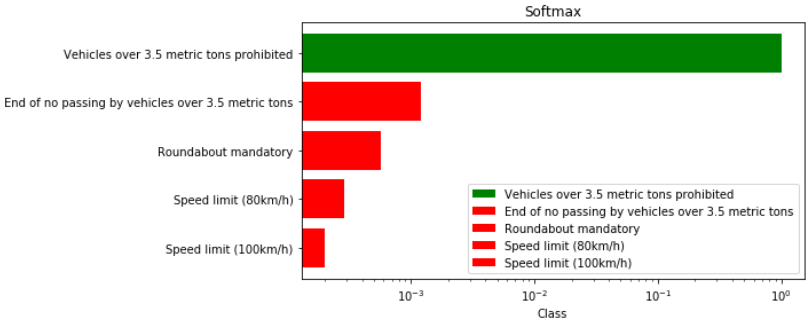

Softmax
Speed limit (120km/h)
Speed limit (100km/h)
Speed limit (70km/h)
Speed limit (20km/h)
Speed limit (80km/h)
Class




Softmax
Priority road
Roundabout mandatory
Go straight or left
Keep left
Ahead only
Class


Softmax
No vehicles
Speed limit (70km/h)
No passing
Speed limit (50km/h)
Yield
Class


Softmax
Road work
Bumpy road
General caution
Bicycles crossing
Right-of-way at the next intersection
Class


Softmax
Vehicles over 3.5 metric tons prohibited
End of no passing by vehicles over 3.5 metric tons
Roundabout mandatory
Speed limit (80km/h)
Speed limit (100km/h)
Class

Додаток 2


**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ**

**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

**КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**


# РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ РОЗПІЗНАВАННЯ ДОРОЖНІХ ЗНАКІВ НА ОСНОВІ МЕТОДІВ МАШИННОГО НАВЧАННЯ

**Студент:** Пальчук Михайло Едуардович, ПДМ-61

**Науковий керівник:** к.т.н., доц., Жебка Вікторія Вікторівна

Київ-2021

2

## АКТУАЛЬНІСТЬ РОБОТИ

- **Проблема:** водії не помічають або не знають деякі дорожні знаки, не дотримуються правил, в результаті спричиняючи ДТП
- **Вирішення:** використання системи, що дозволяє в автоматичному режимі розпізнавати дорожні знаки та звертати на них увагу водія, пояснюючи обмеження, які вони впроваджують.
- **Об'єкт дослідження:** розпізнавання дорожніх знаків за допомогою нейронних мереж.
- **Предмет роботи:** розпізнавання дорожніх знаків за допомогою нейронних мереж для смартфону.
- **Мета:** розробити прототип інформаційної системи розпізнавання дорожніх знаків для iOS.
- **Завдання:** розробка інформаційної системи, що дозволяє розпізнавати дорожні знаки за допомогою телефону.

2

# АКТУАЛЬНІСТЬ РОБОТИ

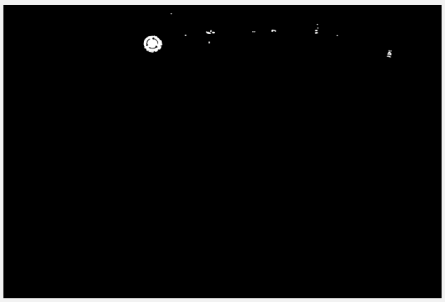
- **Проблема:** водії не помічають або не знають деякі дорожні знаки, не дотримуються правил, в результаті спричиняючи ДТП
- **Вирішення:** використання системи, що дозволяє в автоматичному режимі розпізнавати дорожні знаки та звертати на них увагу водія, пояснюючи обмеження, які вони впроваджують.
- **Об'єкт дослідження:** розпізнавання дорожніх знаків за допомогою нейронних мереж.
- **Предмет роботи:** розпізнавання дорожніх знаків за допомогою нейронних мереж для смартфону.
- **Мета:** розробити прототип інформаційної системи розпізнавання дорожніх знаків для iOS.
- **Завдання:** розробка інформаційної системи, що дозволяє розпізнавати дорожні знаки за допомогою телефону.

4

# ЕТАПИ ОБРОБКИ ЗОБРАЖЕННЯ

1. Початкове зображення

2. Зображення після кольорової сегментації

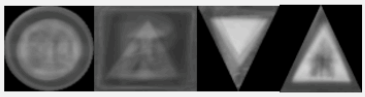


3. Зображення після фільтрації шуму і вирівнювання гістограми

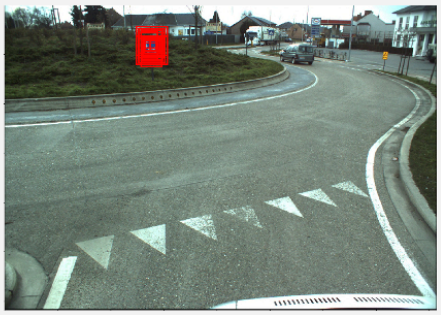
5

# КОМП'ЮТЕРНИЙ ЗІР

## МЕТОД РОЗСУВНОГО ВІКНА ДЛЯ ПОШУКУ ЗОБРАЖЕНЬ ЗА ШАБЛОНОМ



Шаблони знаків



Рамки збігів за шаблоном

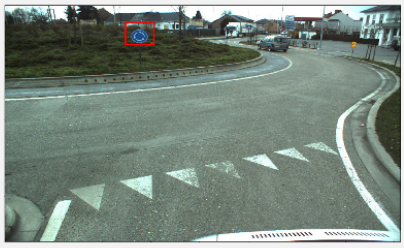
6

# КОМП'ЮТЕРНИЙ ЗІР

## НЕМАКСИМАЛЬНЕ ПРИДУШЕННЯ ДЛЯ ВИЗНАЧЕННЯ НАЙКРАЩОГО ЗБІГУ

$$
\begin{aligned}
&\textbf{procedure } NMS(b, c)\\
&\quad B_{nms} \leftarrow \emptyset\\
&\quad \textbf{for } b_i \in B \textbf{ do}\\
&\quad\quad discard \leftarrow False\\
&\quad\quad \textbf{for } b_i \in B \textbf{ do}\\
&\quad\quad\quad \textbf{if } same(b_i, b_i) > \lambda_{nms} \textbf{ then}\\
&\quad\quad\quad\quad \textbf{if } score(c, b_i) > score(c, b_i) \textbf{ then}\\
&\quad\quad\quad\quad\quad discard \leftarrow True\\
&\quad\quad\quad \textbf{if not } discard \textbf{ then}\\
&\quad\quad\quad\quad B_{nms} \leftarrow B_{nms} \cup b_i\\
&\quad \textbf{return } B_{nms}
\end{aligned}
$$

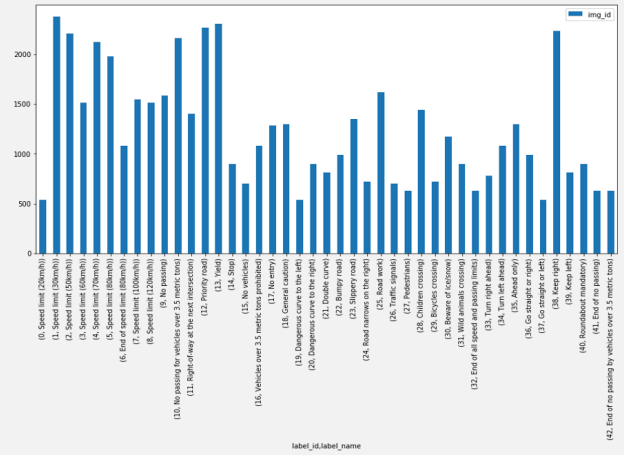
Псевдокод алгоритму



Результат – правильно вибрана рамка

7

# НЕЙРОННА МЕРЕЖА

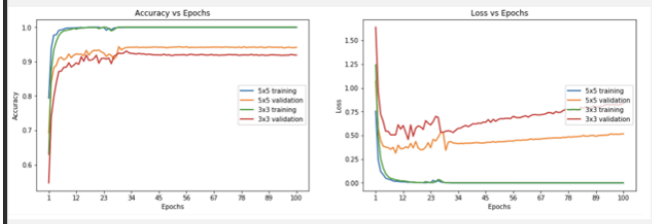




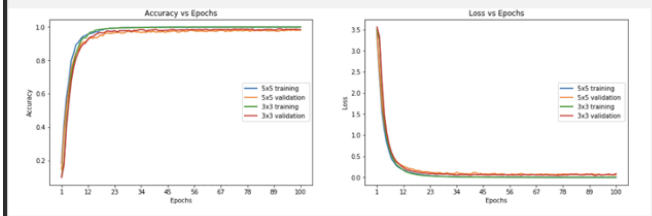
Графік розподілу зображень у датасеті

8

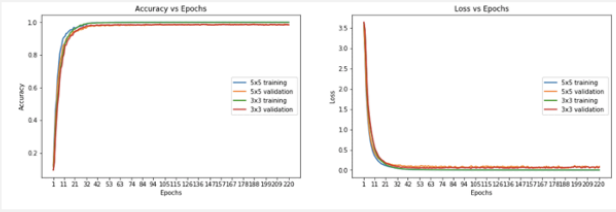
# ТЕСТУВАННЯ НЕЙРОННОЇ МЕРЕЖІ НА РІЗНИХ ЕТАПАХ ОБРОБКИ ЗОБРАЖЕННЯ




1. Результати без обробки: максимальна точність 93%



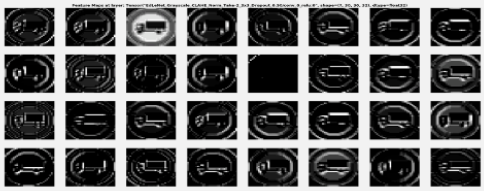

2. Результати після трансформації кольору: максимальна точність 97%, проте результати не дуже стабільні



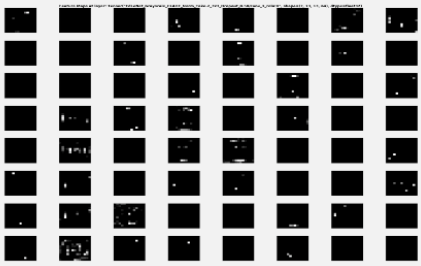

3. Результати після відсіювання шуму і вирівнювання гістограми: максимальна точність 97%, модель стабільна

9

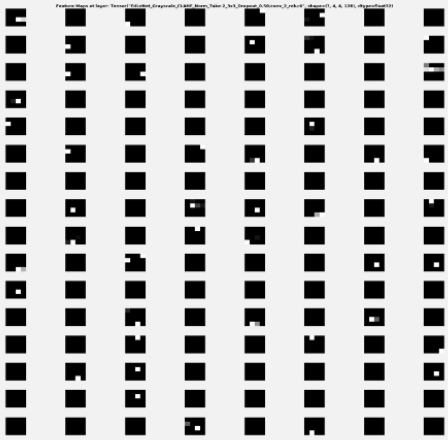
# КАРТИ АКТИВАЦІЇ МОДЕЛІ



Перший шар



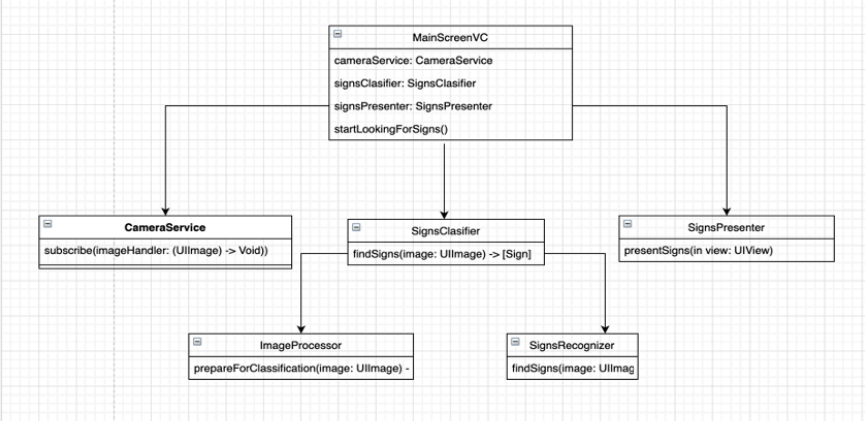
Другий шар



Третій шар

10

# АРХІТЕКТУРА МОБІЛЬНОГО ДОДАТКУ



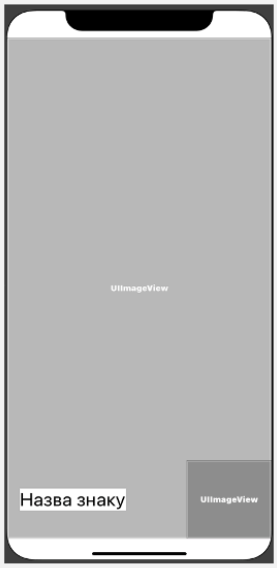
11

## ЗАСОБИ ДЛЯ РОЗРОБКИ ПЗ

| Мови та фреймворки | Платформи |
| --- | --- |
| python™ | Tesseract OCR |
| Swift | Firebase |
| Core ML | OpenCV |
| Firebase | TensorFlow |

12

## ІНТЕРФЕЙС РОЗРОБЛЕНОЇ СИСТЕМИ




13

## ТЕСТУВАННЯ ДЕМО ВЕРСІЇ

- Проведено 3 інтерв'ю з водіями зі стажем більше двох років
  - Один водій вважає додаток корисним
- Проведено 2 інтерв'ю з учнями автошколи
  - Обидва учні хочуть користуватися додатком

- Виявлено основний недолік додатку: відсутність навігації.
- Виявлено додатковий сценарій використання: допомога під час вивчення дорожніх знаків на практичних заняттях.

14

## ВИСНОВКИ

- Проаналізовано існуючі моделі та методи програного моделювання, направлені на вирішення аналогічних проблем
- Вибрані існуючі моделі, які найбільше відповідає завданню роботи. Моделі вдосконалені для розпізнавання дорожніх знаків та адаптовані для мобільної операційної системи.
- Описана архітектура системи модулів, кожен з яких має свої принципи та зону відповідальності, що забезпечує гнучкість та розширюваність програмного продукту. Автономність модулів дає можливість повторного використання коду.
- В основі принципів проектування системи додатку використані широковживані технології та бібліотеки.
- Розроблено прототип мобільного додатку, що дозволяє розпізнавати дорожні знаки.