Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ ПО ОНЛАЙН ЗЧИТУВАННЮ ТА РОЗПІЗНАВАННЮ QR-КОДУ НА МОБІЛЬНОМУ ПРИСТРОЇ
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-18-2

Скударнов М.Д.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Машталір С.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Скударнову Михайлу Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка програмного засобу по онлайн зчитуванню та розпізнаванню QR-коду на мобільному пристрої

затверджена наказом по університету від 16 травня 2022 року № 541Ст

2. Термін подання студентом роботи до екзаменаційної комісії 27 травня 2022 р.

3. Вихідні дані до роботи: матеріали про розробку мобільних застосунків,інтернет-ресурси, результати досліджень розробки алгоритмів для розпізнавання об'єктів, результати обробки візуальної інформації, фреймворк KotlinDL, мова програмування Kotlin

4. Перелік питань, що потрібно опрацювати в роботі

1. Огляд методів розпізнавання QR-коду.
2. Основні методи виділення QR-коду на зображенні.
3. Вирішення задачі класифікації за допомогою нейронних мереж.
4. Аналіз структури нейронних мереж.
5 Огляд підходів розробки мобільних застосунків
6. Програмна реалізація застосунку по розпізнаванню QR-коду у реальному часі із використанням KotlinDL та мови програмування Kotlin.
7. Тестування розробленої моделі.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми розпізнавання QR-коду, постановка задачі, навчання нейронної мережі із використанням KotlinDL, використання нейронної мережі у мобільному застосунку, тестові зображення, аналіз результатів моделювання, висновки

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Консультант з дотримання діючих стандартів та норм	Доцент Бєлова Н.В.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	18.04.2022	
2	Аналіз завдання, аналіз літератури	19.04.22-23.04.22	
3	Дослідження існуючих методів вирішення задачі розпізнавання QR-коду	23.04.22-26.04.22	
4	Огляд існуючих алгоритмів розпізнавання QR-коду	26.04.22-30.04.22	
5	Програмна реалізація	01.05.22-15.05.22	
6	Проведення тестування розробленої моделі	16.05.22-23.05.22	
7	Оформлення пояснювальної записки	25.05.22-03.06.22	
8	Перевірка на плагіат	04.06.2022	
9	Рецензування	05.06.2022	
10	Підготовка презентації та доповіді	06.06.22-11.06.22	
11	Занесення роботи в електронний архів	12.06.22	
12	Попередній захист кваліфікаційної роботи	13.06.22	

Дата видачі завдання 18 04 2022 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Машталір С.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 50 с., 6 табл., 31 рис., 1 дод., 30 джерел.

РОЗПІЗНАВАННЯ QR-КОДІВ. QR-КОД. МЕТОДИ РОЗПІЗНАВАННЯ QR-КОДУ. НЕЙРОННІ МЕРЕЖІ. ОНЛАЙН РОЗПІЗНАВАННЯ QR-КОДУ. QR-СКАНЕР НА МОБІЛЬНОМУ ПРИСТРОЇ.

Об’єктом роботи є набір зображень з QR-кодами.

Метою роботи є розробка мобільного застосунку, який дозволить у реальному часі зчитувати QR-код.

Для виконання цієї задачі було навчено та використано нейронну мережу. Сама нейрона мережа була навчена із використанням KotlinDL. Сам мобільний застосунок написано мовою програмування Kotlin. Також були наведені основні методи розпізнавання QR-коду. Були здобуті навички з мови програмування Kotlin.

Результатом данної роботи є мобільний застосунок та файл для його встановлення на мобільний пристрій.

RECOGNITION OF QR-CODES. QR CODE. METHODS OF RECOGNITION OF QR-CODE. NEURAL NETWORKS. ONLINE QR CODE RECOGNITION. QR-SCANNER ON MOBILE DEVICE.

The object of the work is a set of images with QR-codes.

The aim of the work is to develop a mobile application that will allow real-time reading of QR-code.

To perform this task, a neural network was trained and used. The neural network itself was trained using KotlinDL. The mobile application itself is written in the Kotlin programming language. The main methods of QR code recognition were also presented. Kotlin programming language skills were acquired.

The result of this work is a mobile application and a file for installation on a mobile device.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AIM – Alternative Investments Market

JIS – Japan Industrial Standards

EDI – Electronic Data Interchange

ISO – International Organization for Standardization

ANN – Artificial Neural Networks

DNN – Deep Neural Networks

DRN – Deep Rectifier Neural Networks

DCT – Discrete Cosine Transform

DFT – Discrete Fourier Transform

MSE – Mean Square Error

AUC – Area Under the ROC Curve

NN – Neural Network

ВСТУП

Використання візуальних кодів набуло широкого поширення в нашому житті, як у промислових застосунках, так і в приватних проектах. У порівнянні з іншими технологіями, такими як RFID, вони простіші у використанні та менш дорогі. З удосконаленням технологій, що стосуються як апаратних, так і програмних систем, зріс інтерес до автоматичного розпізнавання не тільки добре розташованих та орієнтованих кодів, але й усіх кодів у зоні датчика, різного розміру та орієнтації. . Це завдання тепер доступне для виконання в режимі реального часу завдяки еволюції комп'ютерного обладнання. Для більш простих вбудованих систем потрібні алгоритми, які вимагають невеликої обчислювальної потужності.

Для налаштувань, які дозволяють більш складне обладнання, тепер доступні алгоритми, що використовують методи машинного навчання та різні методи класифікації. Однак усі ці алгоритми балансують між швидкістю обробки та точністю. Кожна програма кінцевого користувача може визначити, який із цих двох є більш важливим. Застосунок для смартфона для зчитування штрих-кодів може бути більш неточним і може вимагати зробити більше знімків або краще позиціонувати, тоді як промислова програма зчитування може вимагати максимальної точності та автономної роботи.

1 ОГЛЯД ОСНОВНИХ ТИПІВ QR-КОДІВ ТА МЕТОДІВ ЇХ ВИЯВЛЕННЯ

1.1 Зростаюча актуальність

У 1960-х роках, коли Японія вступила в період високого економічного зростання, у багатьох районах почали з’являтися супермаркети, що продають широкий асортимент товарів – від продуктів харчування до одягу. Касові апарати, які потім використовувалися на касах у цих магазинах, вимагали введення ціни вручну. Через це багато касирів страждали від оніміння зап’ястя та синдрому зап’ястного каналу. «Касири відчайдушно прагнули якимось способом полегшити свій тягар»[1]. Винахід штрих-кодів дозволив вирішити цю проблему.

Згодом була розроблена POS-система, в якій ціна товару автоматично відображалася на касовому апараті при скануванні оптичним датчиком штрих-коду товару, а інформація про товар відправлялася на комп’ютер. Однак у міру поширення використання штрих-кодів стали очевидними й їхні обмеження. Найбільш помітним був той факт, що штрих-код може містити лише 20 буквено-цифрових символів або близько того.

Оскільки QR-код є відкритим кодом, який може використовувати будь-хто, він використовується не тільки в Японії, а й у країнах по всьому світу. Оскільки правила його використання були передбачені, а кодекс стандартизовано, його використання поширилося далі. У 1997 році він був затверджений як стандарт AIM для використання в галузі автоматичної ідентифікації. У 1999 році він був затверджений як стандартний 2D-код Японськими промисловими стандартами і внесено стандартний 2D-символ у стандартні форми транзакцій EDI Японської асоціації виробників автомобілів. Більше того, у 2000 році він був затверджений ISO як один із міжнародних стандартів. В даний час використання QR-коду настільки поширене, що без перебільшення можна сказати, що він використовується скрізь у світі.

У той час як використання QR-коду поширилося по всьому світу, нові типи QR-коду для задоволення більш складних потреб створювалися один за одним. Мікро QR-код був створений для задоволення потреби в менших кодах. Він настільки малий, що його можна надрукувати на невеликій площі, і він став стандартом JIS у 2004 році. У 2008 році код iQR, який має невелику площу, незважаючи на велику здатність кодування, дозволяє використовувати прямокутні модулі коду, був випущений[2]. Крім того, тип QR-коду, який реалізує обмеження читання, був розроблений, щоб задовольнити вимоги користувачів щодо підвищеного рівня конфіденційності тощо. «FrameQR» був представлений у 2014 році. FrameQR може покращити дизайн вашого коду, вільно поєднуючи ілюстрації та фотографії.

1.2 Різновиди QR-кодів

Мікро QR-код (рис. 1.1): цей QR-код зазвичай можна знайти на упаковці продукту. Він має лише одну орієнтацію, що полегшує друк на менших поверхнях. Цей код життєздатний навіть як 2 модуля, тоді як QR-код вимагає щонайменше 4 модулів[3]. Найбільша версія цього QR-коду M4 (17×17 модулів) і може зберігати 35 модулів.

Рисунок 1.1 – Мікро QR-код

iQR-код (рис. 1.2): його можна надрукувати у вигляді квадратного або прямокутного QR-коду. Його можна надрукувати у вигляді точкового шаблону, коду інверсії або перевернутого коду. Максимальна версія – 61 (модулі 422×422), яка може зберігати близько 40 000 цифр.

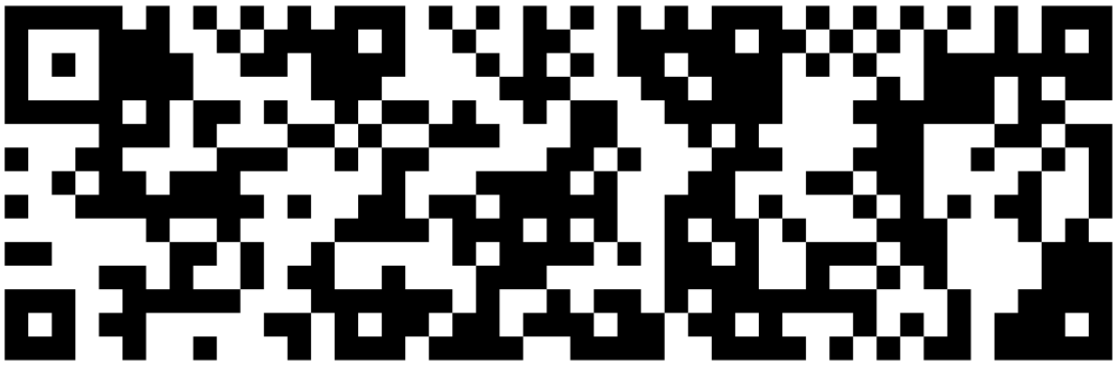


Рисунок 1.2 – IQR-код

SQRC-код (рис. 1.3): виглядає як звичайний QR-код, за винятком того, що він обмежений і використовується для зберігання конфіденційної інформації. SQRC можна прочитати за допомогою певного сканера. Дані в SQRC складаються з публічних і приватних розділів. Один код може нести два типи інформації з різними рівнями керування.

Рисунок 1.3 – SQRC-код

FrameQR (рис. 1.4): цей тип QR-коду має область рамки, в якій користувач може розміщувати літери та зображення та використовуватися для рекламних заходів.

Рисунок 1.4 – Frame QR-код

Код HCC2D (рис. 1.5): кольоровий двовимірний код високої ємності (HCC2D) все ще знаходиться на стадії створення прототипу і був запропонований дослідниками для збереження QR-стійкості до спотворення. Він використовує кольори, щоб збільшити щільність даних і впоратися з хроматичними спотвореннями. Коди HCC2D використовують додаткове поле під назвою Шаблон колірної палітри.

Рисунок 1.5 – HCC2D-код

1.3 Структура QR-коду

QR-код кодує рядок тексту. Стандарт QR має чотири режими кодування тексту: числовий, буквено-цифровий, байтовий і кандзі[4]. Кожен режим кодує текст у вигляді рядка бітів, але кожен режим використовує інший метод для перетворення тексту в біти, і кожен метод кодування оптимізовано для кодування даних з найкоротшим можливим рядком бітів. Але спочатку розглянемо кожен елемент:

Помітки позиціонування (рис. 1.6): вони вказують, яким способом друкується код.

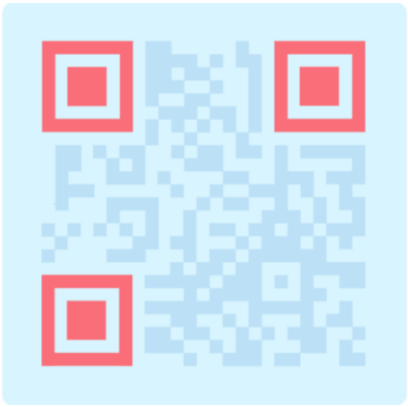


Рисунок 1.6 – Область помітки позиціонування

Розмітки вирівнювання (рис. 1.7): при використанні більших кодів ці позначки можуть допомогти з орієнтацією.

Рисунок 1.7 – Область помітки вирівнювання

Координуючі помітки (рис. 1.8): це рядки, які повідомляють сканеру розмір матриці даних.

Рисунок 1.8 – Область координуючі поміток

Інформація про версію (рис. 1.9): за допомогою цього розділу ви можете вказати, яка версія всіх QR-кодів використовується. Їх існує більше 40 версій, але зазвичай часто використовуються версії від 1 до 7.

Рисунок 1.9 – Область версії

Інформація про формат (рис. 1.10): ці шаблони містять інформацію про толерантність до помилок і шаблон маски даних, щоб спростити сканування коду.

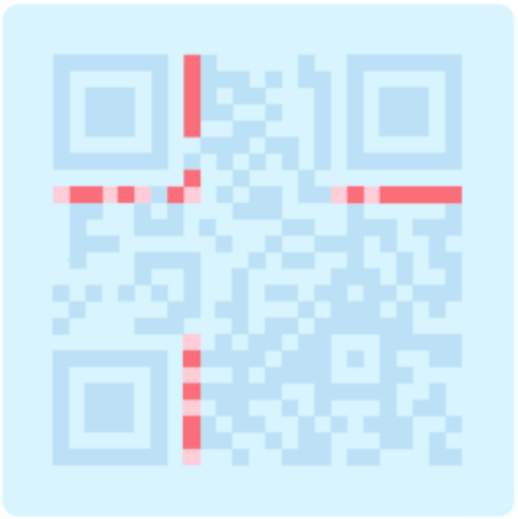


Рисунок 1.10 – Область формату

Дані та ключі виправлення помилок (рис. 1.11): тут відображаються дані.

Рисунок 1.11 – Область даних

1.4 Методи розпізнавання

Розпізнавання QR-коду на фотографії – це добре поставлене завдання машинного зору. По-перше, у задачі досліджується об’єкт, спочатку спеціально розроблений для «зручного» розпізнавання. По-друге, саме завдання розбивається на кілька незалежних зрозумілих підзадач: локалізація

QR-коду, орієнтація QR-коду та безпосередньо декодування QR-коду. Так виявилося, що суспільне надбання вже досить давно має гарні бібліотеки, здатні вирішити останні два завдання: орієнтацію та декодування QR-коду. Одна проблема: для якісного декодування такі бібліотеки очікують на вхід гарне бінарне зображення безпосередньо штрих-коду[5]. І навпаки, завдання локалізації штрих-коду на зображенні приділяється мало уваги.

1.4.1 Метод MIN-MAX

Назва методу MIN-MAX походить від проміжні кроки самого алгоритму, точніше, від розмивання відтінків сірого і розширення, які створюють мінімальні та максимальні зображення через локальні мінімуми та максимуми. Оскільки цей метод добре справляється із зашумленими, розмитими або спотвореними зображеннями, крім необов'язкова нормалізація, операції попередньої обробки не потрібні[6]. Крім того, MIN-MAX також не вимагає спрямованих отворів, що робить цей підхід швидшим порівняно з вищезгаданим (рис. 1.12).

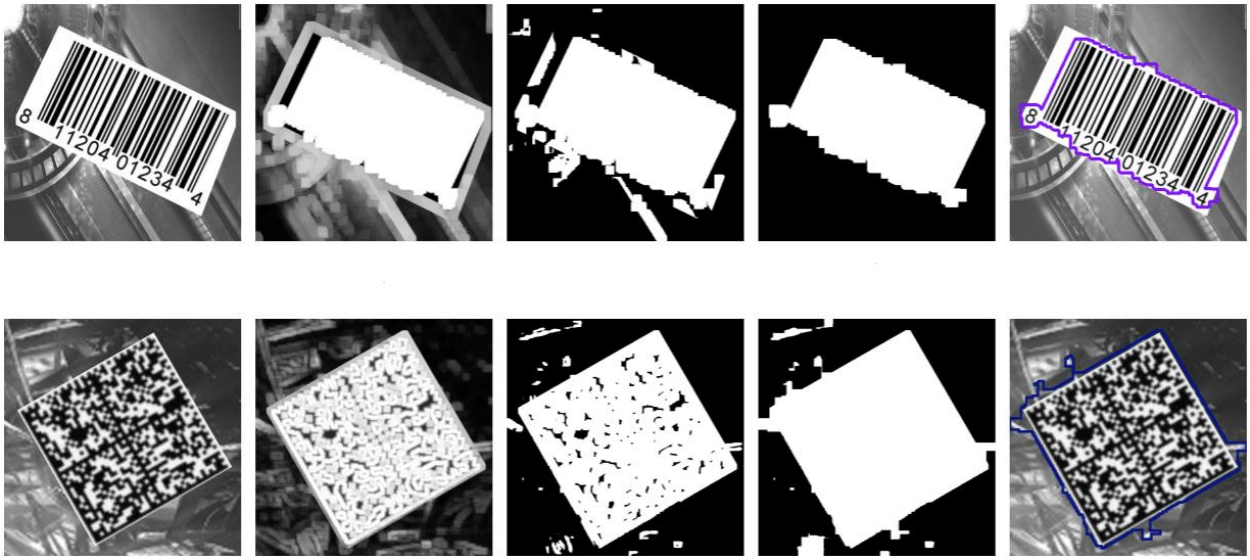


Рисунок 1.12 – Приклад роботи MIN-MAX методу

На цьому прикладі, оператор морфологічного градієнта був застосований до зображення з ядром коробки з шириною очікуваної найширшої смуги плюс одна одиниця. Наступним кроком було видалення слабкої фігури із зображення об’єкта з бінарним порогом. Хорошим практичним правилом щодо порогового значення може бути 75% повної шкали інтенсивності, оскільки штрих-коди створюють ділянки, близькі до максимальної інтенсивності. Після застосування оператора морфологічного градієнта та встановлення порога результату, на ньому виконується операція морфологічного відкриття, з попередньо визначеним ядром для закриття невеликих проміжків, спричинених подряпинами, відбитками або інші зображення оригінального зображення.

На цьому етапі зображення ознаки вже позначає білим кольором ділянки, схожі на штрих-код, таким чином, остання операція, яку потрібно виконати, – це вимірювання точної площі цих ділянок та їх введення[7].

1.4.2 Кластеризація локальної інтенсивності

Локальна кластеризація пікселів є найбільш спрощеним алгоритмом, заснованим на розбивці зображень. Він ділить клітини на чорні та білі сегменти (рис. 1.13).

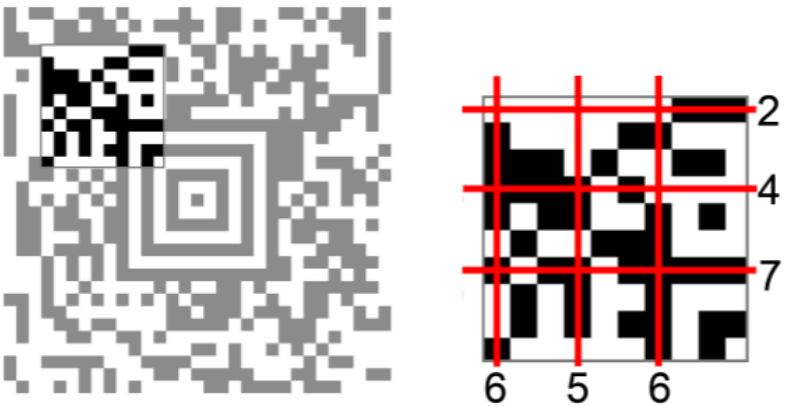


Рисунок 1.13 – Розбиття картинки на сегменти

Область зображення, що містить частину штрих-коду, має багато подібних розтягнутих кластерів. Мінімальна кількість очікуваних сегментів може бути отримана з найширшого штрих-коду та розміру плитки. Подібно до більшості підходів цього сімейства, параметри вимагають припущень щодо очікуваних штрих-кодів. Ступінь розтягування можна визначити як подвійну відстань до найдальшої точки відрізка від його центру[8]. Для точно горизонтальних або вертикальних ліній найбільше значення цього показника це розмір плитки, однак у похилих ситуаціях очікується, що він буде довшим.

Крім того, розтягнуті окремі сегменти потрібно вирівняти приблизно однаково. Інакше один сегмент зливатиметься з іншим, зменшуючи кількість окремих компонентів у плитці і таким чином опускаючись нижче нашого порогу. Коли справа доходить до попередньої обробки, спочатку використовується медіанний фільтр. На реальних зображеннях із низьким контрастом у ділянках штрих-коду необхідне адаптивне порогове значення. Мінімальний розмір сегмента можна легко обчислити з максимальної ширини смуги, помноженої на розмір плитки.

Після локальної оцінки всіх плиток здійснюється пошук компактних областей у матриці ознак. Порогове значення застосовується до значень, щоб класифікувати, чи містить область частину штрих-коду. Визначення порогу вище $T = 0{,}5$ знижує точність виявлення, а встановлення нижче збільшує частоту хибнопозитивних результатів. Після маркування підключеного компонента невеликі компоненти відкидаються, а центри решти кластерів повертаються[9]. Кластери вважаються малими, якщо вони містять менше N плиток:

$$N = \max\left(2, \frac{|h-s| \times |w-s|}{s^2}\right), \tag{1.1}$$

де s – розмір плитки (який у наших прикладах становить 1/3 висоти коду.

Обмежувальні рамки у вихідних зразках, у цьому прикладі, не містять весь штрих-код у кожному разі. Це пов’язано з тим, що були розраховані лише

нижня та верхня межі для кластерів у матриці ознак, а кутові фрагменти штрих-коду є занадто слабкими для цієї функції[9]. Обмежувальні прямокутники можна просто покращити, знайшовши замість них вирівняні прямокутники. Найкращий розмір плитки становить приблизно 1/3 висоти штрих-коду, як при вимірюванні довжини. Виконання методу на одній сцені з різними наборами дає різну точність виявлення, що показує, що цей підхід чутливий до вибору мозаїки. Оцінку з плитками внахлест можна ввести як у два етапи. На першому етапі локальна кластеризація виконується з нульовим набором плиток, а на другому етапі така ж процедура виконується шляхом застосування набору о половини розміру плитки в обох напрямках. Кодові центри, виявлені на вищезгаданих фазах, об’єднуються разом із додатковим фільтруванням, при якому ті кодові центри, які виявлені на обох фазах і знаходяться близько один до одного, об’єднуються в один (більший кластер зберігається), оскільки вони, ймовірно, відповідають до того самого коду (рис. 1.14).

Рисунок 1.14 – Результат методу кластеризації локальної інтенсивності

1.4.3 Звичайні та глибокі нейронні мережі

Протягом останніх кількох років відновився інтерес до застосування нейронних мереж, особливо глибоких нейронних мереж, для різних завдань. Як випливає з їх назви, Глибокі нейронні мережі (DNN) відрізняються від

звичайних (ANN) тим, що складаються з кількох прихованих шарів[10]. Однак, якщо ми хочемо навчити ці глибокі мережі належним чином, ми повинні усвідомлювати той факт, що метод навчання вимагає модифікації, оскільки звичайний алгоритм зворотного поширення стикається з труднощами, наприклад так званий зникаючий градієнт і пояснювальні ефекти. У цьому самому контексті ефект зникнення градієнта означає, що помилка може зникнути, коли вона поширюється назад через приховані шари. Таким чином, деякі приховані шари, зокрема ті, що знаходяться поблизу вхідного шару, можуть не засвоїтися під час навчання. У той же час у повністю пов’язаних глибоких мережах пояснювальні ефекти роблять висновки надзвичайно складними на практиці. Для боротьби з цими проблемами було запропоновано кілька рішень. Рішення змінюють або алгоритм навчання, розширюючи його фазою попереднього навчання, або архітектуру нейронних мереж.

1.5 Постановка задачі

Отже, зчитування QR-коду є актуальним завданням на сьогоднішній день, особливо на мобільному пристрої, де ми маємо ситуацію обмежених ресурсів. У зв’язку з цим ставиться завдання розробки та налаштування нейронної мережі для знаходження QR-коду у реальному часі. Метою роботи є розробка мобільного застосунку на основі використання нейронної мережі, яка дозволяє розпізнавати та декодувати QR-код.

Для досягнення мети необхідно вирішити такі завдання:

– провести аналіз існуючих підходів розпізнавання QR-коду;
– розробити алгоритм нейронну мережу для розпізнавання QR-коду;
– реалізувати мобільний застосунку сканера.

Результатом роботи має бути файл для установки застосунку на мобільний пристрій.

2 ЛОКАЛІЗАЦІЯ QR КОДУ ЗА ДОПОМОГОЮ НЕЙРОННИХ МЕРЕЖ

Подібно методу кластеризація локальної інтенсивності для обробки зображення буде поділене на сегменти або блоки. Для кожного блоку нейронна мережа призначає міру, яка відображає ймовірність присутності частини QR-коду в цьому блоці, в результаті чого утворюється матриця ознак (зображення функції), яка представляє регіони, що представляють інтерес. Наступним кроком процесу є пошук у цій матриці кластерів, які мають достатній розмір, компактність і високі значення ймовірності для формування QR-коду[11]. Кінцевий кроком буде повернення центрів кластерів, які задовольняють вищезазначеним умовам, і надаються обмежувальні рамки для кандидатів на область QR-коду.

2.1 Навчання нейронної мережі

Для кожного блоку, на який розбивається зображення, формується вектор на основі інформації всередині блоку. Вилучені вектори передаються в ядро цього підходу, яке є нейронною мережею. Тут були оцінені як звичайні, так і глибокі нейронні мережі (DRN). DRN змінюють приховані нейрони в мережі, а не алгоритм навчання, використовуючи випрямлені лінійні одиниці. Ці випрямлені одиниці відрізняються від стандартних нейронів лише своєю функцією активації, оскільки вони застосовують функцію випрямляча ($\max(0, x)$) замість активації сигмовидної або гіперболічної дотичної[12]. Завдяки своїм властивостям DRN не вимагають попереднього навчання для досягнення гарних результатів.

Функція випрямляча має дві важливі властивості, а саме жорстке насичення при 0 і лінійну поведінку для позитивного входу. Перша

властивість означає, що лише підмножина нейронів буде активною в кожному прихованому шарі. Наприклад, коли ми рівномірно ініціалізуємо ваги, приблизно половина вихідних прихованих одиниць буде нулями. Теоретично це жорстке насичення при 0 може зашкодити оптимізації, блокуючи зворотне поширення градієнта. Слід зазначити, що експериментальні результати не підтверджують цю гіпотезу, показуючи, що жорсткі нелінійності не завдають шкоди, доки градієнт може поширюватися вздовж деякого шляху. Враховуючи іншу властивість випрямлених блоків, а саме лінійну поведінку активних одиниць, немає ефекту «зникаючого градієнта». Ця лінійна поведінка також означає, що обчислювальні витрати будуть меншими, оскільки немає необхідності обчислювати експоненціальну функцію під час розрахунку активації, а також можна використовувати розрідженість. Ця лінійність також має недолік, який називається ефектом «вибухаючий градієнт», що означає, що градієнти можуть зростати без обмежень. Щоб запобігти цьому, нормалізація L_1 застосовується шляхом масштабування ваг таким чином, щоб норма L_1 ваг кожного шару залишалася такою ж, як і після ініціалізації. Це робить можливим те, що підмножина активних нейронів веде себе лінійно для даного входу, тому масштабування ваг еквівалентно масштабуванню активації. Глибокі мережі, використані в цій дипломній роботі, складалися з трьох прихованих шарів, і кожен прихований шар мав 1000 випрямлених нейронів, оскільки DRN з цією структурою давав найкращі результати на наборах для розробки. Неглибока нейронна мережа була сигмовидною сіткою лише з одним прихованим шаром, з такою ж кількістю прихованих нейронів (3000), що й у глибокої. Вихідний шар нейронних мереж складався з двох нейронів softmax (один для позитивної мітки і один для негативної мітки), що дозволило мережам виводити не тільки рішення щодо класифікації, але й апостеріорні значення ймовірності[13]. Що стосується функції помилки, то була застосована функція крос-ентропії.

Для запобігання надмірному тінгу було використано два методи регуляризації, а саме раннє припинення і зниження ваги. Рання зупинка була

досягнута зупинкою навчання, коли не було покращення в двох наступних ітераціях набору перевірки. Що стосується регуляризації спаду ваги, ваги зменшувалися після кожної ітерації, змушуючи їх сходитися до менших абсолютних значень, ніж це було б інакше.

Нейронні мережі навчалися за допомогою напівпакетного зворотного поширення, розмір пакету становив 100. Початкова швидкість навчання була встановлена на 0,001 і залишалася фіксованою, в той час як помилка на наборі розробки продовжувала зменшуватися. Згодом, якщо частота помилок не зменшувалася для даної ітерації, швидкість навчання згодом зменшувалася вдвічі[14].

2.1.1 Вибір вхідного вектора

Для вхідних векторів доступно кілька варіантів. Нейронні мережі можуть працювати з частковими або повними даними, і вони можуть навчатися на зразках як зображення, так і частотної області. Крім того, у частотній області також можна вибрати квантовані дані, якщо вихідні вектори недоступні. Для кожного блоку, на який розбивається зображення, формується одновимірний вектор шляхом зчитування пікселів у вигляді кругової схеми, який можна адаптувати до QR-кодів (рис. 2.1). Цей шаблон забезпечує помітні функції, зберігаючи низьку кількість необхідних пікселів, зазвичай лише 6-10 % пікселів зображення[15].

Рисунок 2.1 – Приклад зчитування блоку QR-коду за допомогою кругового шаблону

Крім того, круговий візерунок може вказувати на наявність QR-коду в будь-якій орієнтації. Навчальні вектори позначаються позитивними, якщо коефіцієнт покриття QR-кодом був вищим, ніж вибраний поріг T_c для цього блоку. Як правило, піки F-міри знаходяться при $T_c \approx 0{,}5$, тоді як $T_c \approx 0{,}1$ призводить до кращого запам’ятовування (швидкості попадання). Однак кількість цих частково покритих блоків на порядок менша, ніж кількість порожніх і повністю покритих блоків, отже, T_c неможливо визначити під час фази навчання. Більше того, навіть якщо DRN пропускає частково закриті блоки, це лише означає, що він пропускає периметр об’єкта коду. Розширення позитивно класифікованих груп клітинок матриці ознак долає цю проблему. Розмір блоку повинен бути достатньо великим, щоб забезпечити помітні характеристики у вхідних векторах нейронної мережі. Необхідно також визначити величину перекриття. Різні зміщення для блоків, що мають оптимальний розмір, також оцінюються емпірично, оскільки в статтях на цю тему не вказується категорично, чи покращує перекриття продуктивність нейронної мережі[16]. Проте широко використовуються згорткові нейронні мережі, що мотивує оцінку з різними зміщеннями блоків.

2.1.2 Навчання DRN з блоками JPEG DCT

JPEG є одним із найпоширеніших форматів нерухомих зображень і забезпечує ефективні дані зберігання та перенесення. Більшість камер можуть отримувати зображення безпосередньо у форматі JPEG, і деякі пристрої можуть навіть виводити потік зображень JPEG, що мотивує дослідження в методи обробки зображень за допомогою цього формату[17].

Нейронні мережі також здатні навчатися в частотній області та JPEG формат можна обробляти як підмножину цього домену, також маючи блок 8×8 пікселів розмір для введення. Використовуючи цей підхід, для локалізації коду

необхідно виконати лише перші кроки декомпресії, у той час як найскладніший крок, зворотний DCT можна пропустити.

Для налаштувань, що використовують зображення або потоки JPEG, процес декодування JPEG можна зупинити в точці, де квантовані коефіцієнти DCT відновлюються з файлу, відразу після виконання зворотного RLE і кодування Хаффмана. Матриця коефіцієнтів, які представляють блок 8×8 пікселів у зображенні, служить вхідним вектором DNN. Для порядку коефіцієнтів підходить «зигзагоподібний» шаблон, який визначається стандартом JPEG, оскільки немає різниці в ефективності навчання при використанні різновпорядкованих векторів одного навчального набору[18]. Цей порядок також корелює з візуальним рівнем важливості коефіцієнтів, і тому рекомендується для оцінки продуктивності DNN, використовуючи лише префікс вектора.

Вхідні вектори для цього конкретного DRN є одновимірними векторами, утвореними квантованими коефіцієнтами DCT блоку 8×8 пікселів. Під час декодування множення з таблицею квантування можна пропустити з двох причин. По-перше, через нелінійну природу нейронних мереж вони здатні навчатися на векторному наборі та на тій самій множині, поелементно помноженій на інший фіксований вектор, із подібною ефективністю. По-друге, компоненти вхідних векторів були нормовані, щоб мати нульове середнє значення та одиничну дисперсію. Ця нормалізація покращила чисельну умову задачі оптимізації під час навчання, забезпечивши більш швидку збіжність.

При цьому налаштуванні DRN потрібно навчати, використовуючи зображення того ж рівня стиснення, що й зображення програми кінцевого користувача, оскільки оригінальна матриця DCT не була застосована, що вимагало б множення на таблицю квантування. Без деквантування різні рівні стиснення, застосовані до одного і того ж вмісту зображення, призводять до різних векторів. Ці вектори можуть бути далекі від навчальних вибірок певного рівня стиснення. Щоб подолати це, DRN можна навчати за допомогою

деквантованих векторів коефіцієнтів, які приблизно однакові на схожих рівнях стиснення[19].

2.2 Оцінка та результати

Для вхідних даних були доступні різні варіанти, і для кожного типу вхідних даних проводилося окреме навчання (табл. 2.1). Першим вибором було навчити нейронну мережу на необроблених піксельних даних і прочитати вибраний шаблон для кожного блоку. Двійкова версія векторів також була оцінена для порівняння ефективності нейронних мереж на сірих і бінарних зображеннях. Крім того, оскільки QR-код має чітко визначену, сувору структуру, це означає, що блоки частин QR-коду, ймовірно, мають дуже специфічні компоненти в частотній області. Це припущення послужило мотивацією для проведення експериментів з векторами в цій області.

Навчальні вектори були перетворені в домени DCT і DFT, а нейронні мережі були оцінені в обох наборах. У цьому випадку наш інтерес був спеціально зосереджений на продуктивності DRN в домені DCT, оскільки багато налаштувань передбачають стиснення JPEG за допомогою апаратного забезпечення, яке також можна використовувати.

Також нейронна мережа була навчена на карті країв, оскільки структура QR-кодів також передбачає дуже специфічні макети країв. Навчальні вектори в цьому випадку складалися з пікселів уніфікованої карти величин градієнтів Собел-X і Собеля-Y (рис. 2.2), зчитованих по колу[20]. Результати показують, що 8-розрядні вхідні вектори лише трохи кращі, ніж двійкові версії. І частотна область, і карта країв підходять для введення, однак вони не варті обчислювальних витрат перетворення, якщо вхідні дані не в цьому форматі за замовчуванням.

Таблиця 2.1 – Результати навчання для різних типів і діапазонів вхідних даних

Input data	Range	Precision	Recall	*F*-measure
ANN				
Raw pixels	8-bit	0,9892	0,9953	0,9922
Raw pixels	binary	0,9705	0,9846	0,9774
DCT	8-bit	0,9889	0,9951	0,9920
DCT	binary	0,9693	0,9860	0,9776
DFT	8-bit	0,9904	0,9981	0,9943
DFT	binary	0,9711	0,9808	0,9760
Sobel-XY	8-bit	0,9979	0,9990	0,9984
DRN				
Raw pixels	8-bit	0,9947	0,9972	0,9959
Raw pixels	binary	0,9704	0,9862	0,9782
DCT	8-bit	0,9941	0,9967	0,9954
DCT	binary	0,9686	0,9873	0,9778
DFT	8-bit	0,9933	0,9958	0,9945
DFT	binary	0,9621	0,9850	0,9734
Sobel-XY	8-bit	0,9978	0,9991	0,9984

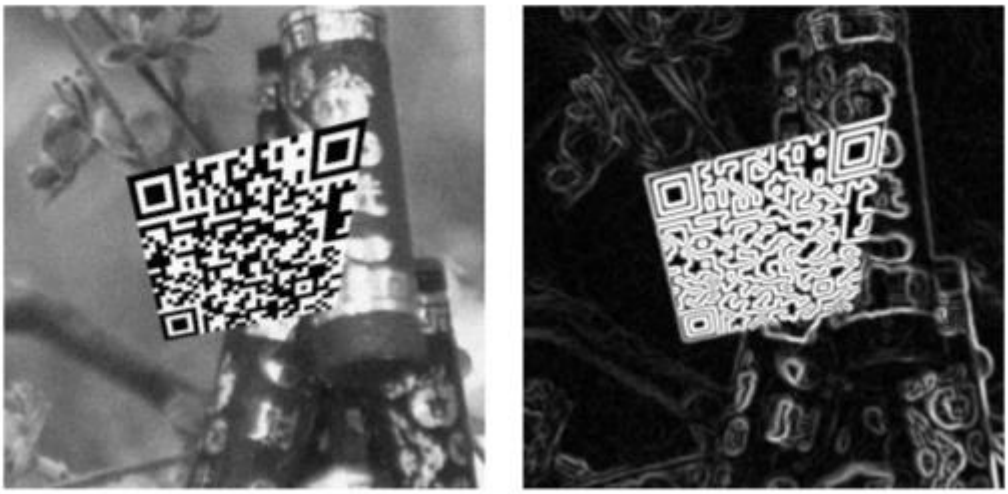


Рисунок 2.2 – Зразок синтетичного зображення та зображення величини Собеля

2.2.1 Якість JPEG та кількість коефіцієнтів

Вплив стиснення JPEG на продуктивність DRN також розглядався в цій дипломній роботі. Як і очікувалося, краща якість зображення призвела до кращих результатів навчання, як показано на рисунку 2.3. На рисунку 2.4 показані показники продуктивності DRN, навчених на перших n елементах векторів[21]. Можна помітити, що приблизно перших 10 елементів вектора коефіцієнтів достатньо для навчання DRN, що має *F*-міру вище 0,9, і продуктивність лише трохи покращується при використанні більше половини векторних коефіцієнтів.

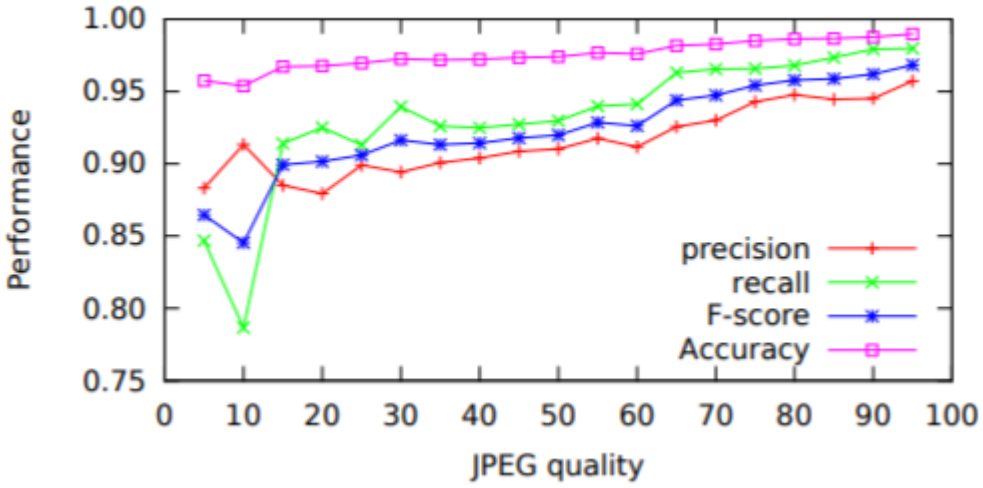


Рисунок 2.3 – Продуктивність DRN для якості набору даних

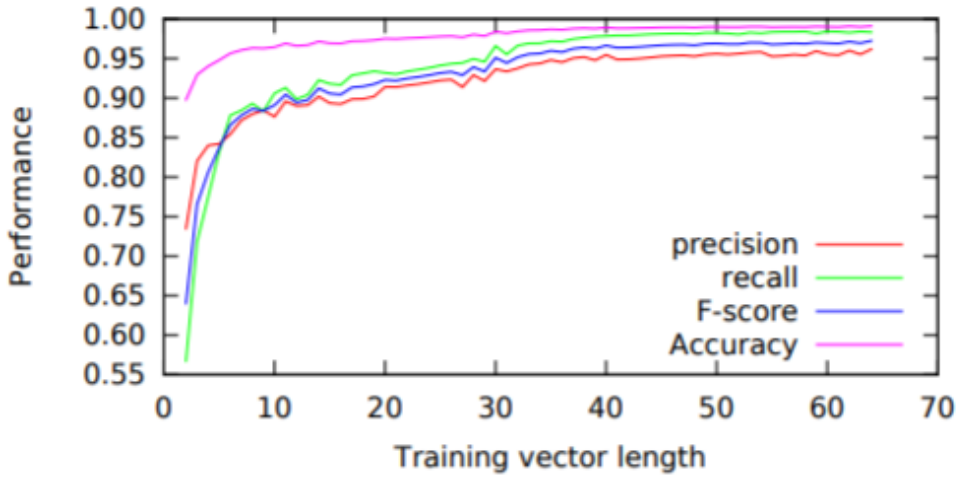


Рисунок 2.4 – Продуктивність DRN для довжини навчальних векторів

Усі мережі, натреновані за допомогою тестового набору певної якості, також були оцінені на всіх наборах, таким чином показавши надійність мереж щодо різниці в рівнях стиснення між навчальним набором і наборами тестів (рис. 2.5). Результати показують, що DRN є досить специфічними для якості, на якій вони пройшли навчання, але все ж працюють добре до 10-15% допуску. Виходячи з цих результатів, хоча навчання DRN до певної якості зображення не вимагається, вони працюють найкраще, коли навчальні та тестові зображення мають однакову якість. Аналогічне правило застосовується при навчанні DRN з використанням усіх векторів, витягнутих із зображень різної якості JPEG. DRN, натренований з використанням цих векторів, буде працювати гірше, ніж той, який навчався з такою ж якістю JPEG, що й налаштування кінцевого користувача[22].

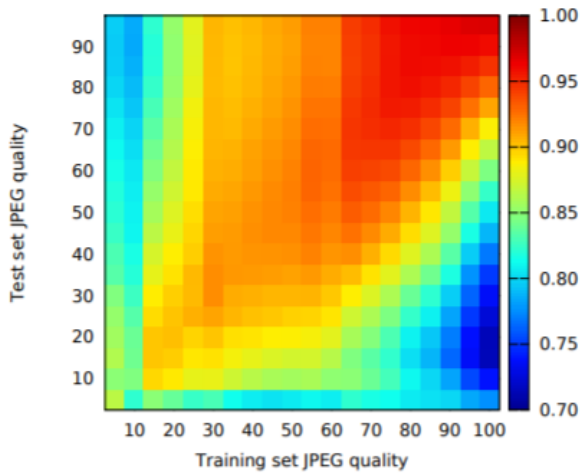


Рисунок 2.5 – Карта продуктивності навчених DRN, що використовуються для зображень з однаковим вмістом, але різною якістю JPEG

2.2.2 Частково перекриті блоки

У першому випадку навчання проводилося лише з використанням векторів для фону та блоків, повністю покритих частинами QR-коду. Як наступний крок цього експерименту, вхідні дані були розширені для частково

покритих блоків і встановлені порогові значення 0,1 і 0,5 коефіцієнтів покриття для позитивного маркування[23]. До кількості негативно мічених векторів фільтрування не застосовувалося. І ANN, і DRN навчалися окремо на векторах як синтетичних, так і реальних зображень.

Результати (табл. 2.2) показують, що використання частково охоплених вибірок для навчання суттєво знижує точність NN. Крім того, поріг 0,1 для позитивних міток підтримує відкликання на задовільному рівні, тоді як 0,5 різко знижує його (рис. 2.6). Проте низький рівень відкликання не є проблемою, оскільки в матриці ймовірностей залишається достатньо позитивно передбачених блоків, щоб навіть у цих випадках можна було вказати кандидата QR-коду (рис. 2.7).

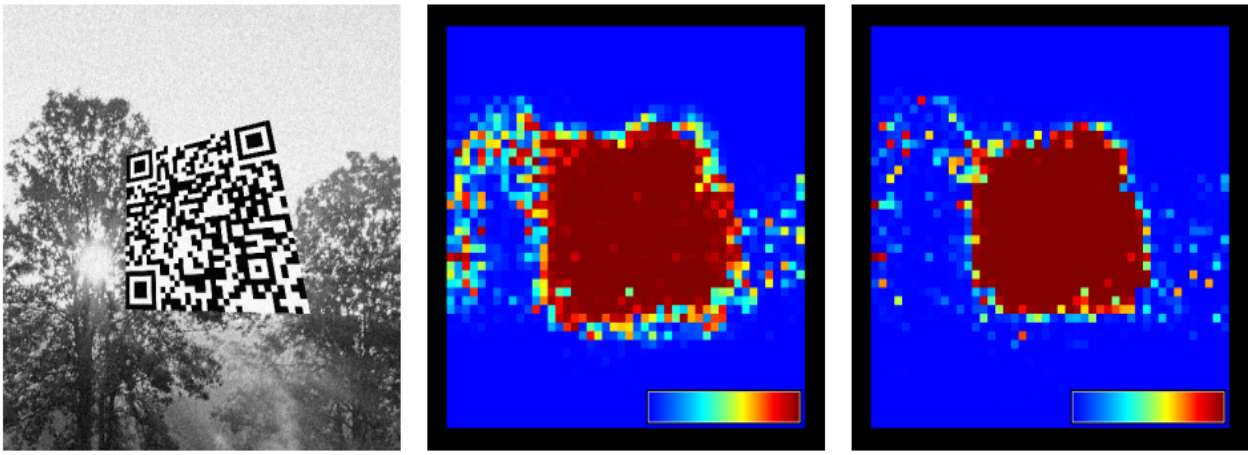


Рисунок 2.6 – Синтетичне зображення, створене з виходу нейронних мереж

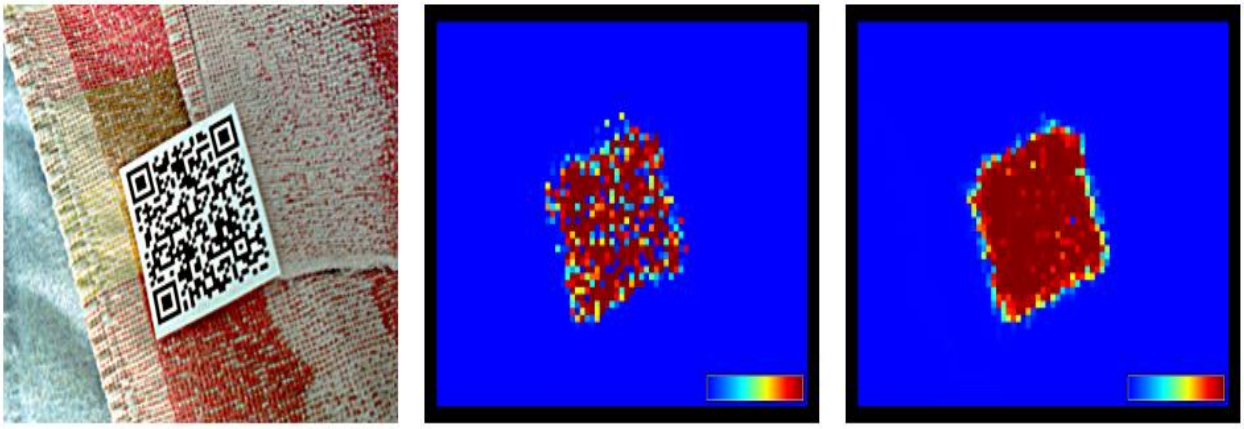


Рисунок 2.7 – Реальне зображення

Отже, можна зробити висновок, що нейронні мережі, навчені на повних зображеннях і на зменшеній підмножині векторів (де позитивно та негативно мічені зразки приблизно однакові) не відрізняються суттєво, тоді як виключення векторів, які надходять із частково покритих блоків, різко зменшує виконання. У таблиці 2.3 узагальнено результати, отримані при використанні вищезгаданих навчальних наборів. N^+ і N^- позначають позитивні та негативні вибірки, тоді як N_b^+ і N_b^- є підмножинами N^+ і N^- з виключеними частково покритими блоками.

Таблиця 2.2 – Оцінка нейронних мереж

NN type	Data type	Precision	Recall	*F*-measure
T+ = 0.1				
ANN	Real	0,6454	0,9518	0,7692
DRN	Real	0,6699	0,9419	0,7829
ANN	Synthetic	0,5654	0,9901	0,7198
DRN	Synthetic	0,5630	0,9865	0,7169
T+ = 0.5				
ANN	Real	0,9175	0,5994	0,7251
DRN	Real	0,8962	0,8414	0,8679
ANN	Synthetic	0,8703	0,8947	0,8823
DRN	Synthetic	0,9059	0,9347	0,9201

Таблиця 2.3 – Результати DRN для різних підмножин вхідних векторів синтетичних зображень

Subset	Opt. Thresh.	max(*F*1)	AUC
All vectors	0,62	0,9343	0,9957
$N^+ = N^-$	0,63	0,9270	0,9949
$N_b^+ = N_b^-$	0,82	0,8312	0,9608

2.3 Тренування регресії

Замість навчання нейронної мережі класифікатора для частково покритих блоків, мережі можна також навчити регресії за допомогою функції помилки MSE та одного вихідного нейрона сигмовидної форми. У таблиці 2.4 наведено результати навчання. І ANN, і DRN були оцінені за синтетичними та реальними тестовими даними з використанням усіх блоків зображень, що означає приблизно в 4 рази більше таких вхідних векторів, які не містять частин QR-коду, порівняно з кількістю тих, які містять. Загальні показники якості регресій, такі як MSE, AUC та R2 , використовувалися для визначення відповідності[24]. Характерні зображення регресії можна побачити на рисунках 2.8 та 2.9.

Добре видно, що кількість векторів, витягнутих із реальних зображень, була недостатньою для навчання. Хороші значення AUC свідчать про те, що, використовуючи правильний поріг, ми можемо досягти хороших результатів із DRN. На жаль, регресійне навчання не покращує результати в порівнянні з результатами, отриманими за допомогою стандартного навчання класифікації.

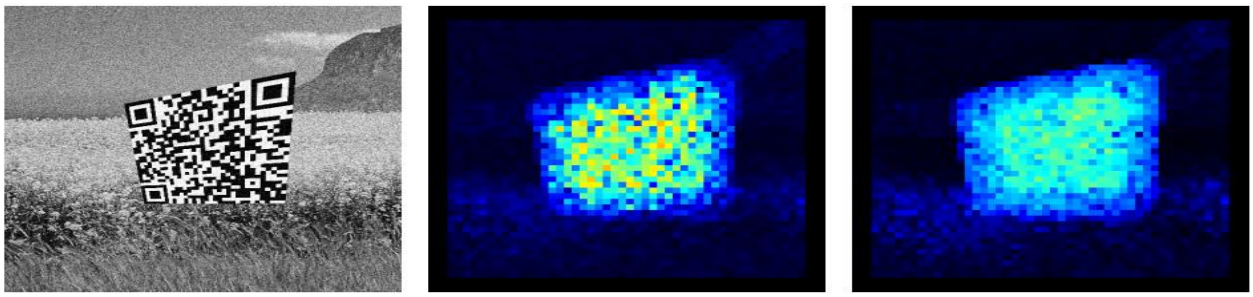


Рисунок 2.8 – Синтетичне зображення для регресійного навчання

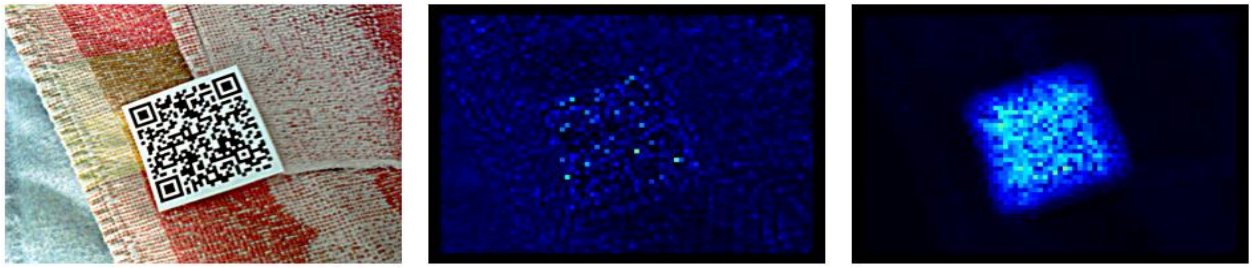


Рисунок 2.9 – Реальне зображення для регресійного навчання

Таблиця 2.4 – Ефективність нейронних мереж, навчених для регресії

NN type	Data type	MSE	AUC	R2
ANN	Real	0,0953	0,5416	0,0794
DRN	Real	0,0490	0,9837	0,1960
ANN	Synthetic	0,537	0,9682	0,2874
DRN	Synthetic	0,0465	0,9782	0,2918

2.3.1 Адаптація домену

Нейронним мережам потрібно багато навчальних векторів, які можуть бути недоступні для кожної програми кінцевого користувача[25]. Штучно згенеровані навчальні вектори, доповнені одними з кінцевої установки, можуть підвищити точність, як показано на рисунку 2.10. Мережі навчалися з використанням 1,5 мільйонів векторів синтетичної бази даних, які мали низьку точність щодо реального набору даних, яку можна було покращити, додавши вектори останнього набору. На останньому етапі адаптації, коли ми використали всі 1,75 мільйона навчальних векторів, наш DRN досяг *F*-міри 86,81 на реальних даних. Ця продуктивність трохи краща, ніж та, яка досягається шляхом навчання DRN лише на реальних даних, але вимагає більше часу на навчання.

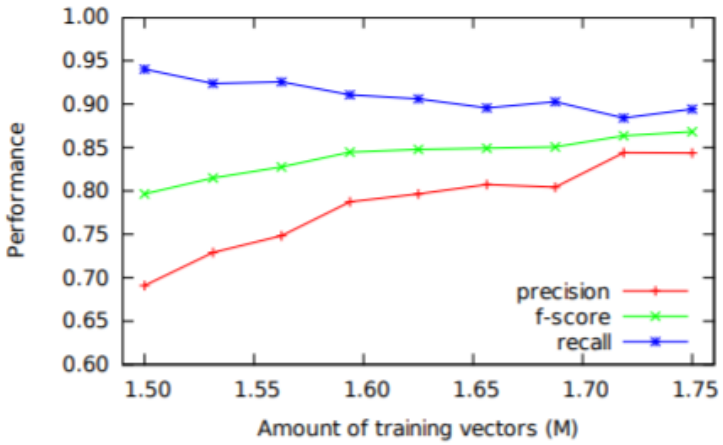


Рисунок 2.10 – Можливості адаптації домену

2.3.2 Інші шаблони та типи коду

Хоча круговий шаблон, як показано, підходить для оцінки блоку, були також оцінені різні інші шаблони (табл. 2.5). Коли тільки кожен *n*-й піксель зчитується вздовж осей, *F*-міра знижується з 0,95 (n = 1) до 0,89 (n = 10). Зчитування пікселів уздовж центральних ліній або діагоналей блоку також є ефективним і включає лише кілька пікселів. У цьому конкретному випадку вивчення гістограми виявляється найменш надійним методом. Що стосується різних типів коду (табл. 2.6), то ефективність навчання істотно не змінюється, за винятком кодів PDF417, який не є справжнім типом 2D-коду, а є складеним одновимірним штрих-кодом, який має найменшу інваріантність до обертання.

Таблиця 2.5 – Оцінювання за різними шаблонами

Input	Precision	Recall	*F*-measure
Pattern-1	0,9465	0,9688	0,9575
Pattern-2	0,9404	0,9630	0,9516
Pattern-3	0,9359	0,9654	0,9504
Pattern-4	0,9322	0,9567	0,9443
Pattern-5	0,9162	0,9539	0,9346
Pattern-6	0,9263	0,9157	0,9210
Pattern-7	0,9203	0,8884	0,9041
Pattern-8	0,9254	0,8825	0,9034
Pattern-9	0,9015	0,8975	0,8995
Pattern-10	0,8925	0,8898	0,8911
Pattern-Plus	0,9320	0,9682	0,9497
Pattern-X	0,9200	0,9334	0,9267
Pattern-Hist	0,8644	0,8905	0,8773

Таблиця 2.6 – Оцінка популярних двовимірних типів коду

Type	Precision	Recall	*F*-measure
QR	0,9879	1,000	0,9939
PDF417	0,8888	0,9032	0,896
Data matrix	0,9780	0,9888	0,9834
Codablock	0,9764	0,9431	0,9595
Aztec code	0,9906	0,9883	0,9894

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ МОДЕЛІ НЕЙРОНОЇ МЕРЕЖІ ТА СТВОРЕННЯ ДИЗАЙНУ МОБІЛЬНОГО ЗАСТОСУНКУ

3.1 Середовище розробки

Так як робота передбачає мобільний застосунок, середовище було обране відповідне – Android Studio. Завдяки IntelliJ IDEA ця IDE забезпечує швидке завершення коду та миттєву оцінку робочого процесу. Існують певні функції Android Studio, такі як підтримка коду для змін і чудовий редактор коду для оптимізації виведення коду. Android Studio дозволяє розробникам швидко вносити зміни, відправляючи код під час виконання роботи і полегшуючи швидкі зміни без повного перезапуску програми. Це забезпечує чудову гнучкість для внесення невеликих змін до програми, поки програма все ще працює. Інтуїтивно зрозумілий редактор коду Android Studio є ключовою функцією, яка забезпечує одну з ключових переваг Android Studio, як-от більш швидке програмування. У той же час він забезпечує найсучасніший рефакторинг, завершення коду та аналіз коду[26].

Також Android Studio підтримує декілька мов програмування:

– Java – Java є офіційною мовою розробки Android і підтримується Android Studio;

– Kotlin – Kotlin був нещодавно представлений як вторинна «офіційна» мова Java;

– C/C++ — Android Studio також підтримує C++ з використанням Java NDK.

Середовище розробки адаптоване для виконання типових завдань, що вирішуються в процесі розробки застосунків для платформи Android. У тому числі у середовище включені засоби для спрощення тестування програм на сумісність з різними версіями платформи та інструменти для проектування застосунків, що працюють на пристроях з екранами різної роздільності (планшети, смартфони, ноутбуки, годинники, окуляри тощо). Крім

можливостей, присутніх в IntelliJ IDEA, в Android Studio реалізовано кілька додаткових функцій, таких як нова уніфікована підсистема складання, тестування і розгортання застосунків, заснована на складальному інструментарії Gradle і підтримуюча використання засобів безперервної інтеграції.

Для прискорення розробки застосунків представлена колекція типових елементів інтерфейсу і візуальний редактор для їхнього компонування, що надає зручний попередній перегляд різних станів інтерфейсу застосунку (наприклад, можна подивитися як інтерфейс буде виглядати для різних версій Android і для різних розмірів екрану). Для створення нестандартних інтерфейсів присутній майстер створення власних елементів оформлення, що підтримує використання шаблонів. У середовище вбудовані функції завантаження типових прикладів коду з GitHub.

До складу також включені пристосовані під особливості платформи Android розширені інструменти рефакторингу, перевірки сумісності з минулими випусками, виявлення проблем з продуктивністю, моніторингу споживання пам’яті та оцінки зручності використання. У редактор доданий режим швидкого внесення правок. Система підсвічування, статичного аналізу та виявлення помилок розширена підтримкою Android API. Інтегрована підтримка оптимізатора коду ProGuard. Вбудовані засоби генерації цифрових підписів. Надано інтерфейс для управління перекладами на інші мови.

Особливості та функції передбачені середовищем:

– живі макети (layout): редагувальник WYSIWYG – живе кодування – подання (rendering) програми в реальному часі;

– консоль розробника: підказки по оптимізації, допомога по перекладу, стеження за напрямком, агітації та акції — метрики Google аналітики;

– резерви бета релізів та покрокові релізи;

– базування на Gradle;

– Android-орієнтований рефакторинг та швидкі виправлення;

– Lint утиліти для охоплення продуктивності, юзабіліті, сумісності версій та інших проблем;

– використання можливостей ProGuard та підписів до програм;

– шаблони для створення поширених Android дизайнів та компонентів;

– багатий редактор макетів (layouts) що дозволяє користувачам перетягнути і покласти (drag-and-drop) компоненти користувацького інтерфейсу, як варіант, переглянути одночасно макети (layouts) на різних конфігураціях екранів[27].

3.1.1 Огляд мови програмування Kotlin

Kotlin – статично типізована, об’єктно-орієнтована мова програмування, що працює поверх Java Virtual Machine і розробляється компанією JetBrains. Також компілюється в JavaScript і в код ряду платформ через інфраструктуру LLVM. Мова названа на честь острова Котлін у Фінській затоці, на якій розташоване місто Кронштадт.

Автори ставили за мету створити мову більш лаконічну і типобезпечну, ніж Java, і простішу, ніж Scala. Наслідком спрощення порівняно зі Scala стали також швидша компіляція та краща підтримка мови у IDE. Мова повністю сумісна з Java, що дозволяє Java-розробникам поступово перейти до її використання; зокрема, мова також вбудовується Android, що дозволяє для існуючого android-програми впроваджувати нові функції на Kotlin без переписування програми повністю.

Синтаксис мови використовує елементи JavaScript, Паскаля, TypeScript, Haxe, PL/SQL, F#, Go і Scala, C++, Java, C#, Rust і D. При оголошенні змінних параметрів типи даних вказуються після назви (розділювач – двокрапка). Крапка з комою, як роздільник операторів, також необов’язкова (як у Scala, Groovy та JavaScript); у більшості випадків переведення рядка достатньо, щоб

компілятор зрозумів, що вираз закінчився. Крім об’єктно-орієнтованого підходу Kotlin також підтримує процедурний стиль з використанням функцій. Як і C, C++ і D, точка входу в програму – функція main, що приймає масив параметрів командного рядка. Програми Kotlin також підтримують perl- і shell-стиль інтерполяції рядків (змінні, включені в рядок, замінюються на свій вміст).

Kotlin дозволяє писати менше коду. Менше коду важливо, але є читабельність, яку також слід розглянути та бажано покращити. З Kotlin ви отримуєте їх обидва. JetBrains зробили все можливе, щоб зробити мову якомога лаконічнішою, і їм це вдалося. Менше коду, якщо все зроблено правильно, призводить до меншої кількості помилок. Якщо ви дозволите фреймворку подбати про певні звичайні аспекти кодування, ви зможете зосередитися на більш важливих речах. Kotlin – це висока читабельність, простота та полегшення процесу розробки програми[28].

3.1.2 Огляд можливостей фреймворка KotlinDL

KotlinDL – це високорівневий API глибокого навчання, написаний на Kotlin і натхненний Keras. Keras – високорівневий API глибокого навчання, розроблений Google для впровадження нейронних мереж. Він написаний на Python і використовується для полегшення реалізації нейронних мереж. Він також підтримує обчислення кількох серверних нейронних мереж.

KotlinDL, як і Keras, розроблений для людей, а не для машин, дотримується найкращих практик щодо зниження когнітивного навантаження: він пропонує послідовні та прості API, мінімізує кількість дій користувача, необхідних для поширених випадків використання, а також надає чіткі й ефективні повідомлення про помилки. Він також містить велику документацію та посібники для розробників. Під капотом він використовує TensorFlow Java API і ONNX Runtime API для Java. KotlinDL пропонує прості

API для навчання моделей глибокого навчання з нуля, імпортування існуючих моделей Keras та ONNX для висновку та використання передачі навчання для адаптації наявних попередньо навчених моделей до ваших завдань[29].

KotlinDL дозволяє дуже легко конструювати нейронні мережі, як це показано на рисунку 3.1.

```
val model = Sequential.of(
    Input(
        IMAGE_SIZE,
        IMAGE_SIZE,
        NUM_CHANNELS
    ),
    Conv2D(
        filters = 32,
        kernelSize = longArrayOf(5, 5),
        strides = longArrayOf(1, 1, 1, 1),
        activation = Activations.Relu,
        kernelInitializer = GlorotNormal(SEED),
        biasInitializer = Zeros(),
        padding = ConvPadding.SAME
    ),
    MaxPool2D(
        poolSize = intArrayOf(1, 2, 2, 1),
        strides = intArrayOf(1, 2, 2, 1)
    ),
    Conv2D(
        filters = 64,
        kernelSize = longArrayOf(5, 5),
        strides = longArrayOf(1, 1, 1, 1),
        activation = Activations.Relu,
        kernelInitializer = GlorotNormal(SEED),
        biasInitializer = Zeros(),
        padding = ConvPadding.SAME
    ),
```

Рисунок 3.1 – Код створення шарів нейронної мережі

Для навчання було використано набір даних QR-DN1.0. Він включає 5 категорій QR-кодів, які охоплюють рівні від низької до високої щільності. Кожна група має 15 QR-кодів: 5 зображень для тестування та 10 зображень для навчання. Після вбудовування QR-кодів у 30 кольорових зображень за

допомогою методів сліпого водяного знака, а потім вилучення QR-кодів із зображень, зроблених камерою мобільного телефону трьома різними методами, ми матимемо три групи по 2250 витягнутих QR-зображень, що забезпечить загалом 6750 спотворених та шумні QR-зображення. У кожній із згаданих трьох категорій дані поділяються на дві частини: тестування з 750 зображеннями та навчання з 2250 зображеннями. Для кожного спотвореного QR у наборі даних його неспотворений екземпляр розміщується як основна істина. Однією з переваг цього набору даних є його реальність. Оскільки до зображень не додано жодного змодельованого шуму, і цей набір даних повністю походить від справжнього завдання вилучення вбудованих QR-кодів у кольорові зображення, зроблені із зображення з водяним знаком на екрані. Він також включає різні типи QR-кодів, такі як один символ, коротке речення, довге речення, URL-адреса та місцезнаходження[30].

Сам процес навчання теж не викликає ніяких складнощів, все що потрібно це викликати метод fit(), як це показано на рисунку 3.2.

```
it.fit(
    dataset = train,
    epochs = 10,
    batchSize = 100
)
```

Рисунок 3.2 – Код навчання нейронної мережі

Вхідні параметри можно трактувати як: *epochs* – кількість ітерацій над даними, які потрібно виконати в процесі навчання; *batchSize* – скільки прикладів буде використано для оновлення параметрів моделі (він же ваг і зміщення) за раз.

3.2 Програмна реалізація для мобільного пристрою

Перш за все нашому застосунку потрібен доступ до камери смартфона. Щоб отримати його нам необхідно у файлі AndroidManifest безпосередньо вказати його, як це показано на рисунку 3.3.

```
<uses-permission android:name="android.permission.CAMERA"/>
```

Рисунок 3.3 – Дозвіл використання камери пристрою

Також нам необхідно створити дизайн. Для цього ми скористалися вбудованим редактором Android Studio. Результат на рисунку 3.4.

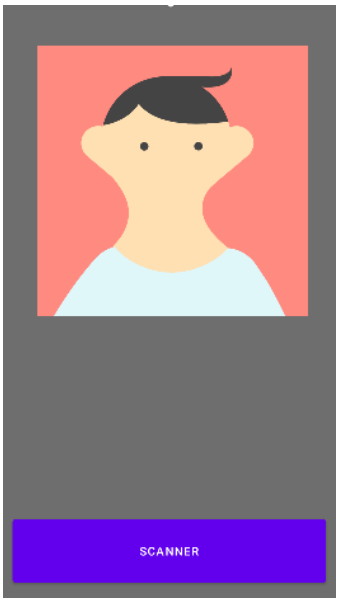


Рисунок 3.4 – Дизайн мобільного застосунку

Наступними кроками буде виклик навченої нейронної мережі та декодування QR-коду. Декодуванням відбувається за допомогою бібліотеки ZBar. ZBar – це пакет програмного забезпечення з відкритим вихідним кодом для зчитування штрих-кодів із різних джерел, таких як відеопотоки, файли

зображень та необроблені датчики інтенсивності. Гнучка багатошарова реалізація полегшує декодування штрих-коду для будь-якої програми.

Отже, даємо на вхід нейронній мережі зображення с камери та відкриваємо дешифроване повідомлення одразу у браузері (рис. 3.5)

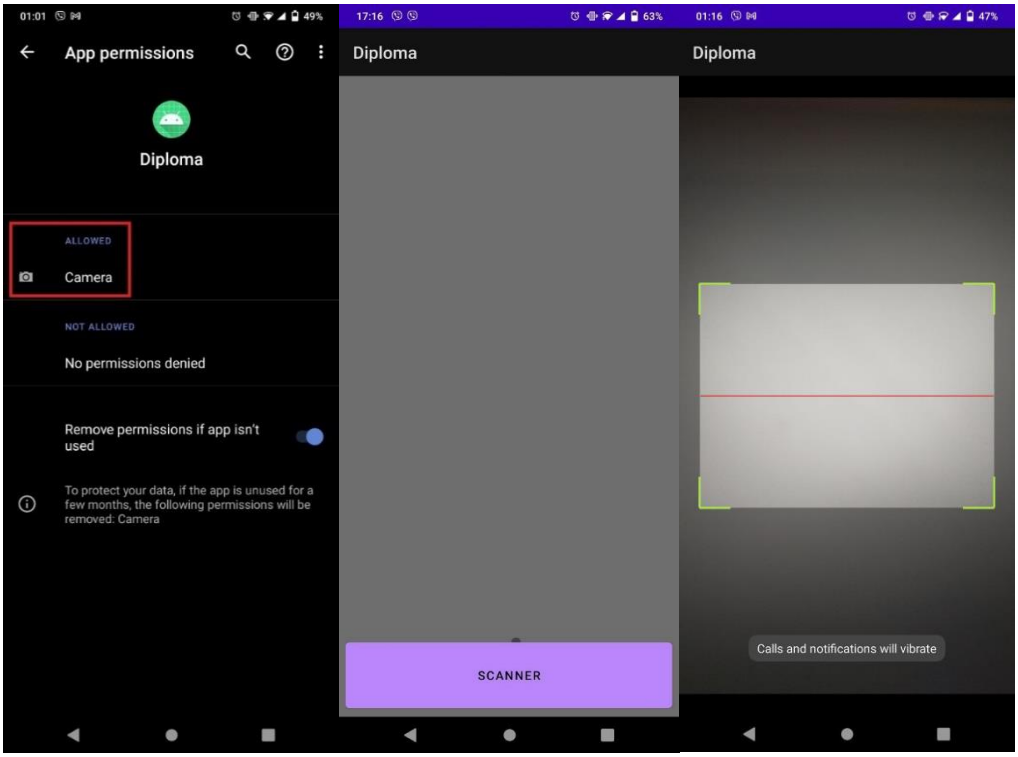
```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    zbView = ZBarScannerView( context: this)
    setContentView(zbView)
}

override fun onResume() {...}

override fun onPause() {...}

override fun handleResult(result: Result?) {
    Log.d( tag: "MyLog", msg: "Result:${result?.contents}")
    val resultUrl = result?.contents
    val openURL = Intent(Intent.ACTION_VIEW)
    openURL.data = Uri.parse(resultUrl.toString())
    startActivity(openURL)
}
```

Рисунок 3.5 – Код обробки дешифрованого повідомлення

3.3 Запуск та використання застосунку

Для того щоб встановити розроблений QR-сканер необхідно завантажити та запустити APK файл до вашого смартфону. Android Package (APK) – формат архівних виконуваних файлів-програм для Android та ряду інших операційних систем, заснованих на Android. Кожна програма Android скомпільована і упакована в один файл, який включає весь код програми (.DEX-файли), ресурси, активи (assets), файл маніфесту AndroidManifest.xml і нативні бібліотеки (jniLibs). Файл програми може мати будь-яке ім’я, але розширення має бути .APK.

Після цього треба переконатися, що застосунок отримав доступ до камери. Зробити це можна у налаштуванні вашого девайсу (рис. 3.6(а)). Тепер залишилося натиснути кнопку «Сканувати» (рис. 3.6(б)) та навести камеру вашого пристрою на QR-код (рис. 3.6(в))



а) б) в)

Рисунок 3.6 – Знімки екрану під час роботи

а) необхідний дозвіл використання камери;

б) інтерфейс мобільного застосунку;

в) знімок екрана під час сканування

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений і реалізований метод розпізнавання QR-коду у реальному часі з допомогою KotlinDL. На базі цього методу було створено мобільний застосунок для пристроїв на базі Android.

У якості об’єкту дослідження було обрано послідовність синтетично створених та реальних фото QR-коду.

Результатом проведеного дослідження є виконання всіх поставлених цілей:

− проведений аналіз існуючих методів розпізнавання QR-коду;

− створений алгоритм розпізнавання QR-коду у реальному часі;

− реалізація мобільного застосунку із використанням вище вказаного методу;

− проведене тестування на експериментальних даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Bilonoh, B., Bodyanskiy, Y., Kolchygin, B., & Mashtalir, S. (2021, May). Tunable Activation Functions for Deep Neural Networks. In *International Scientific Conference "Intellectual Systems of Decision Making and Problem of Computational Intelligence"* (pp. 624-633). Springer, Cham.
2. Bilonoh, B., & Mashtalir, S. (2020, August). Parallel multi-head dot product attention for video summarization. In *2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP)* (pp. 158-162). IEEE.
3. Mashtalir, S., & Mashtalir, V. (2020). Spatio-temporal video segmentation. In *Advances in Spatio-Temporal Segmentation of Visual Data* (pp. 161-210). Springer, Cham.
4. Mashtalir, S. V., Stolbovyi, M. I., & Yakovlev, S. V. (2019). Clustering video sequences by the method of harmonic k-means. *Cybernetics and Systems Analysis*, *55*(2), 200-206.
5. Roberts, D. A., Yaida, S., & Hanin, B. (2022). *The Principles of Deep Learning Theory: An Effective Theory Approach to Understanding Neural Networks*. Cambridge University Press.
6. Nielsen, M. A. (2015). *Neural networks and deep learning* (Vol. 25). San Francisco, CA, USA: Determination press.
7. Koul, A., Ganju, S., & Kasam, M. (2019). *Practical Deep Learning for Cloud, Mobile, and Edge: Real-World AI & Computer-Vision Projects Using Python, Keras & TensorFlow*. O'Reilly Media.
8. Путятін, Є. П., Гороховатський, В. О., & Матат, О. О. (2006). Методи та алгоритми комп'ютерного зору: навч. посібник.
9. Творошенко, І. С. (2021). Технології прийняття рішень в інформаційних системах: навч. посібник. *Харків: ХНУРЕ*.
10. Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., ... & Farhan, L. (2021). Review of deep learning: Concepts, CNN

architectures, challenges, applications, future directions. *Journal of big Data*, *8*(1), 1-74.

11. Tiwari, S. (2016, December). An introduction to QR code technology. In *2016 international conference on information technology (ICIT)* (pp. 39-44). IEEE.

12. Liu, Y., Yang, J., & Liu, M. (2008, July). Recognition of QR Code with mobile phones. In *2008 Chinese control and decision conference* (pp. 203-206). IEEE.

13. Gu, Y., & Zhang, W. (2011, March). QR code recognition based on image processing. In *international conference on information science and technology* (pp. 733-736). IEEE.

14. Гороховатський, В. О., & Творошенко, І. С. (2021). Методи інтелектуального аналізу та оброблення даних: навч. посібник.

15. Кобилін, О. А., & Творошенко, І. С. (2021). Методи цифрової обробки зображень: навч. посібник. *Харків: ХНУРЕ*.

16. Kobylin O., Gorokhovatskyi V., Tvoroshenko I., and Peredrii O. (2020) The application of non-parametric statistics methods in image classifiers based on structural description components, *Telecommunications and Radio Engineering*, 79(10), pp. 855-863.

17. Daradkeh, Y.I., Tvoroshenko, I., Gorokhovatskyi, V., Latiff, L.A., and Ahmad, N. (2021) Development of Effective Methods for Structural Image Recognition Using the Principles of Data Granulation and Apparatus of Fuzzy Logic, *IEEE Access*, 9, pp. 13417-13428.

18. Daradkeh, Y.I., Gorokhovatskyi, V., Tvoroshenko, I., Gadetska, S., and Al-Dhaifallah, M. (2021) Methods of Classification of Images on the Basis of the Values of Statistical Distributions for the Composition of Structural Description Components, *IEEE Access*, 9, pp. 92964-92973.

19. Gorokhovatskyi, V.O., Tvoroshenko, I.S., and Peredrii O.O. (2020) Image classification method modification based on model of logic processing of bit

description weights vector, *Telecommunications and Radio Engineering*, 79(1), pp. 59-69.

20. Gorokhovatskyi, V., Gorokhovatskyi, O., Yevgenyi, P., & Olena, P. (2018). Quantization of the Space of Structural Image Features as a Way to.

21. Tvoroshenko, I., & Kukharchuk, V. (2021). Current state of development of applications for recognition of faces in the image and frames of video captures.

22. Hkdh, B. (1999). Neural networks in materials science. *ISIJ international*, *39*(10), 966-979.

23. Gurney, K. (2018). *An introduction to neural networks*. CRC press.

24. The flow of improved BRISK algorithm. URL: https://www.researchgate.net/figure/the-flow-of-improved-BRISK-algorithm_fig1_328946366 (дата звернення 24.04.2022).

25. ZBar Android SDK. URL: https://github.com/ZBar/ZBar/tree/master/android/ (дата звернення 22.04.2022).

26. KotlinDL: High-level Deep Learning API in Kotlin. URL: https://github.com/Kotlin/kotlindl (дата звернення 11.05.2022).

27. The Kotlin Blog. URL: https://blog.jetbrains.com/ru/kotlin/2020/12/deep-learning-with-kotlin-introducing-kotlindl-alpha/ (дата звернення 06.05.2022).

28. QR Code Tutorial: Introduction. URL: https://www.thonky.com/qr-code-tutorial/introduction (дата звернення 28.04.2022).

29. Structure of The QR Code. URL: https://myqrbc.com/structure-of-the-qr-code-how-is-the-data-coded/ (дата звернення 27.04.2022).

30. Keras: Deep Learning for humans. URL: https://github.com/keras-team/keras (дата звернення 02.05.2022).