МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ІСКУСНИХ ДАР'Я КОСТЯНТИНІВНА

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____ 2024 р.

Інформаційна система для аналізу якості розпізнавання предметів на картинках

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
С. Д. Штовба, професор кафедри
інформаційних технологій,
д-р техн. наук, професор
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____

Вінниця - 2024

АНОТАЦІЯ

Іскусних Д. К. Інформаційна система для аналізу якості розпізнавання предметів на картинках. Спеціальність 122 «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2024.

Метою роботи є дослідження якості розпізнавання об'єктів на зображенні нейронними мережами. Проведено аналіз основних нейронних мереж, які знаходяться у відкритому доступі. Розроблена інформаційна система, що дозволяє взаємодіяти з низкою нейронних мереж у зручній формі та застосовувати фільтри до зображення. За допомогою розробленої програми проведена серія експериментів з визначення впливу трансформацій зображень на якість їх розпізнавання. Результати роботи можуть бути використані для аудиту поточних можливостей нейронних мереж для розпізнавання CAPTHA, а також можуть бути використані як основа для розробки методів графічного захисту, стійких до нейронних мереж.

Ключові слова: нейронні мережі, розпізнавання зображень, інформаційна система, комп'ютерний зір, точність, аналіз якості.

Рис.15. Табл. 2. Бібліограф.: 23 найм.

ABSTRACT

Iskusnykh D. K. Information System for Analyzing the Quality of Object Recognition in Pictures. Specialty 122 “Computer Science”. Vasyl’ Stus Donetsk National University, Vinnytsia, 2024

The work aims to study the quality of image recognition of objects by neural networks. An analysis of the popular neural networks, which are publicly available, was carried out. The developed information system allows you to interact with several neural networks conveniently and apply filters to the image. Using the developed program a series of experiments was conducted to determine the impact of image transformations on the quality of their recognition. The results of the work can be used to audit the current capabilities of neural networks for recognition of the CAPTCHA and can be used as a basis for developing neural network-resistant graphical protection methods.

Keywords: neural networks, image recognition, information system, computer vision, accuracy, quality analysis.

Fig. 15. Tables 2. Bibliograohy.: 23 items.

ЗМІСТ

ВСТУП

Актуальність теми дослідження: У сучасному світі Інтернет стає все більш необхідною технологією, а від його безперебійної роботи залежить багато речей. На даний момент в мережі існує велика кількість сервісів: від поштових і банківських додатків до розважальних і освітніх сайтів. Таким чином, підтримка нормального функціонування всього комплексу пристроїв і технологій, що використовують Інтернет, стає важливим завданням для IT-фахівців.

Заходи, пов'язані із забезпеченням інформаційної безпеки, є вимушеними через наявність кіберзлочинців. Кіберзлочинці регулярно здійснюють атаки з метою отримання фінансової вигоди, шпигунства або кібертероризму. Однією з найпопулярніших з них є атака типу «відмова в обслуговуванні DDoS» [4]. Вона полягає в багаторазовому надсиланні запиту на сервер з метою перевантаження його запитами.

Відповіддю на ці атаки стало широке впровадження CAPTCHA, тесту, призначеного для визначення того, чи є користувач людиною або комп'ютерною програмою (ботом). Цей метод був ефективним до появи нейронних мереж і штучного інтелекту, які навчилися зламувати буквені CAPTCHA [5]. Важливо відзначити, що існують також CAPTCHA, засновані на розпізнаванні об'єктів на картинці. Тому зараз стоїть важливе завдання розуміння потенційних можливостей нейронних мереж для розпізнавання зображень.

Таким чином, дослідження якості розпізнавання нейронними мережами предметів на зображеннях є актуальним завданням, що має велике значення для забезпечення інформаційної безпеки в Інтернеті.

Об'єктом дослідження: графічні інтерфейси людино-комп’ютерної взаємодії.

Предметом дослідження: моделі, алгоритми, програмні модулі для експериментального дослідження якості автоматичного розпізнавання предметів на картинках в інформаційних системах ідентифікації людини.

Мета дослідження: підвищення стійкості графічних фільтрів ідентифікації людини-користувача інформаційної системи від обходу їх програмами-роботами.

Завдання дослідження:

- Аналіз сучасної ситуації
- Моделювання предметної області
- Розробка інформаційної системи
- Проведення експериментальних досліджень

Практичне значення одержаних результатів: полягає в тому, що була розроблена інформаційна система, яка є функціональним та корисним інструментом, за допомогою якого можливо оцінювати якість розпізнавання об'єктів нейронними мережами, спотворювати зображення, а також отримувати корисні статистичні дані.

Структура кваліфікаційної роботи: бакалаврська робота складається із вступу, трьох розділів та висновків до них, списку використаних джерел та двох додатків.

У першому розділі бакалаврської роботи проведено огляд предметної області.

У другому розділі проведено моделювання UML діаграм, а також досліджено наявні нейронні мережі.

У третьому розділі наведено детальні дані про процес розробки, тестування та використання інформаційної системи, ретельно описаний експеримент.

Бакалаврська робота включає 62 сторінок, 15 рисунків, 2 таблиці і список літератури із 23 джерел.

РОЗДІЛ 1. АНАЛІЗ СУЧАСНОЇ СИТУАЦІЇ У СФЕРІ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

У цьому розділі буде детально розглянуто кількість користувачів інтернету. Які загрози для них виникають, як на це вплинуло широке використання штучного інтелекту, а також огляд поточної ситуації щодо сфер впровадження штучного інтелекту.

1.1 Аналіз обсягу IT-сфери

За даними The world bank [9], станом на 2021 рік кількість користувачів Інтернету становила 63,1% населення. За графіком на рисунку 1.1 ми бачимо, що кількість користувачів у світі не просто зростає, а збільшується прискореними темпами. Причому спостерігається не тільки збільшення користувачів, але і обсяг ринку IT-послуг і, за даними Mordor Intelligence, до 2024 року він досяг 1,2 трлн доларів [10].

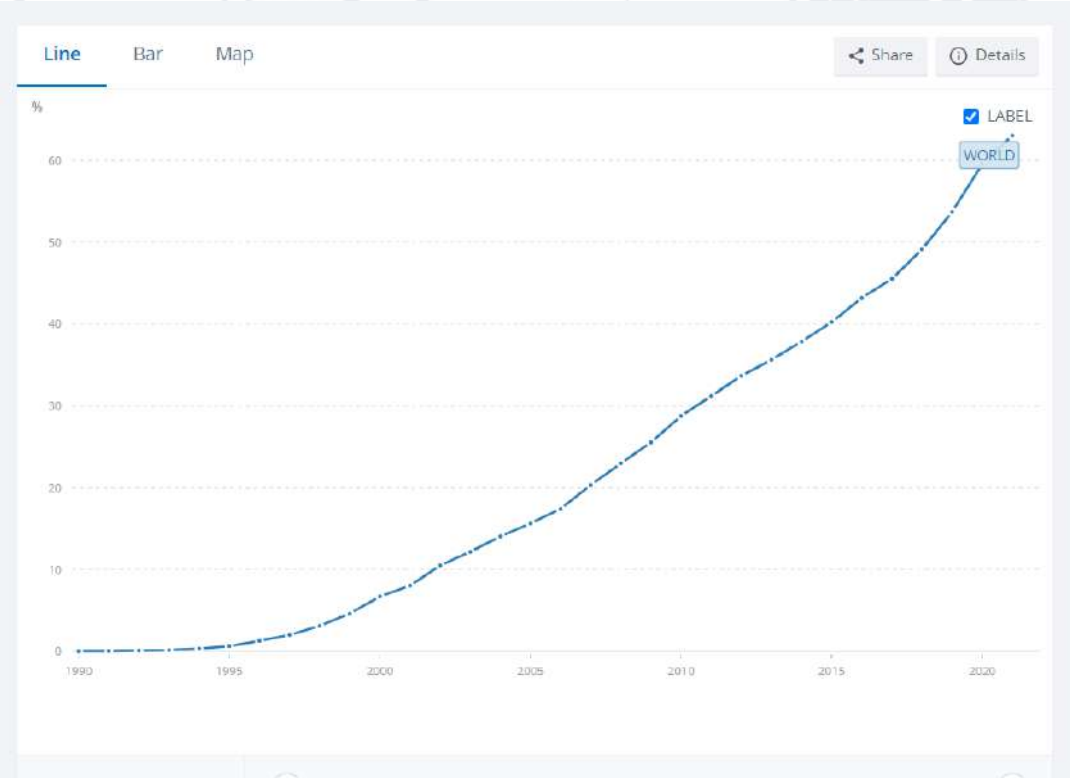

Рисунок 1.1 – Кількість людей, що користуються мережею Інтернет

За представленими даними можна зробити висновок, що кількість користувачів, які мають доступ до інформаційних технологій у всьому світі, буде неухильно зростати[1]. Прискорення темпів зростання свідчить про все більш інтенсивне впровадження інтернет-технологій в різні сфери соціально-економічного життя. Крім того, обсяг IT-ринку свідчить про значне зростання попиту на технологічні рішення та послуги. Таке зростання створює сприятливі умови для інновацій, розробки нових бізнес-моделей і підвищення продуктивності в різних галузях. Відповідно, роль захисту в цій сфері з часом буде тільки зростати.

1.2 Існуючі загрози для IT-сфери

Щомиті у світі відбуваються тисячі кібератак, а антивірусне програмне забезпечення, у свою чергу, блокує їх. Наприклад, за даними однієї з найбільших американських компаній, пов'язаних із безпекою даних, Avast, у 2023 році запобігли 1,05 млрд атак [3]. DDoS-атаки заслуговують на окрему увагу, незважаючи на їх невелику кількість – близько 1% серед усіх[3], вони становлять значну загрозу, оскільки є цілеспрямованими, тобто націлені на конкретну компанію. Ці атаки здебільшого зачіпають критично важливі послуги, такі як фінансові та телекомунікаційні. Більш детальна статистика за напрямками, які вона охоплює, представлена на рисунку 1.2

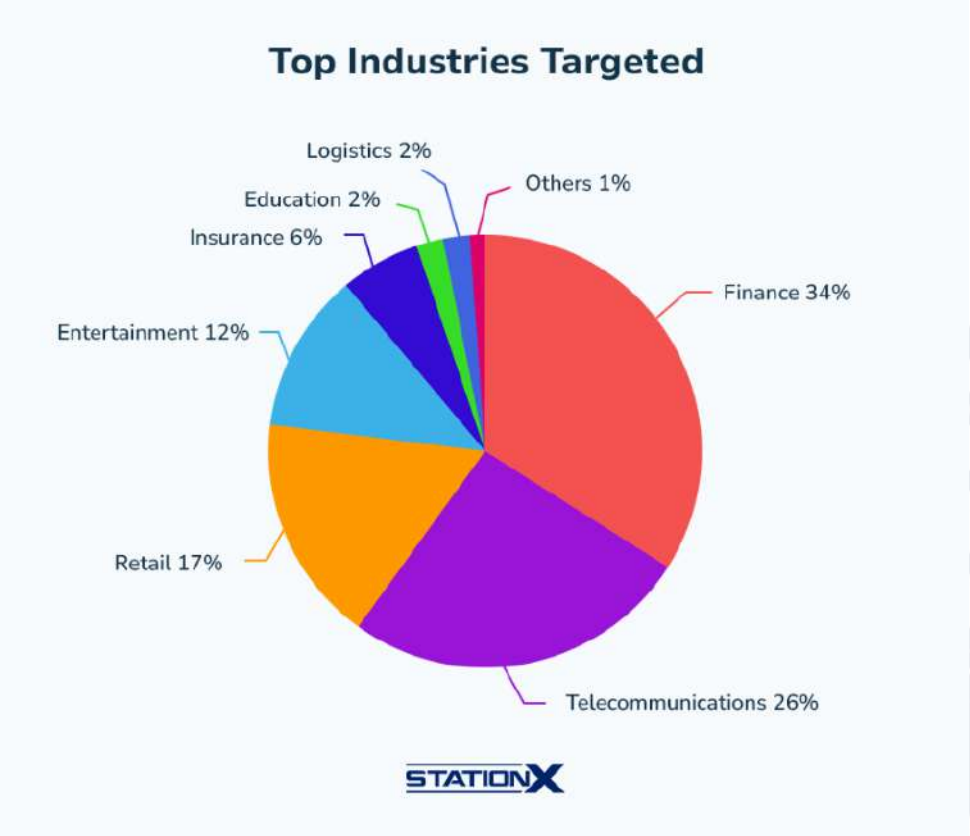

Рисунок 1.2 – Сфери на які було здійснено атаку

Досить часто, крім стандартної фінансової мотивації, ці атаки мають ідеологічний і політичний мотив. Відповідно, підготовка і проведення цих атак носить комплексний характер (використовується потужні і найбільш інноваційні підходи) і найчастіше здійснюються фахівцями з високою компетенцією в області інформаційної безпеки.

1.3 Захист від DDoS-атак

У зв'язку з вищесказаним, найважливішим аспектом для захисту від подібних атак є чітке розуміння поточних можливостей зловмисників. Довгий час CAPTCHA була досить ефективним способом захисту від DDoS-атак, приклади яких наведені на рисунку 1.3.

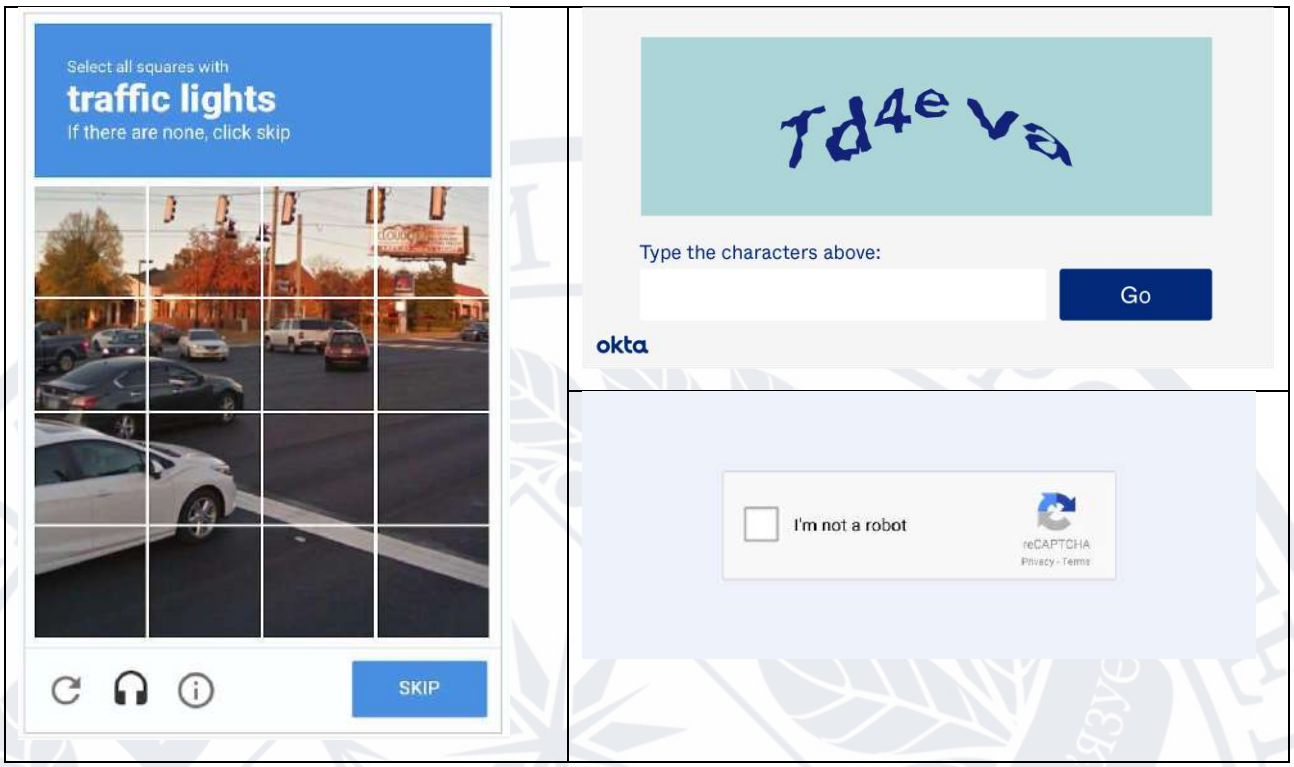

Рисунок 1.3 – Види CAPTCHA

CAPTCHA вводиться для запобігання помилкового доступу до мережевих ресурсів. Він був винайдений в 2000 році Луї фон Аном в Університеті Карнегі-Меллона. Це метод захисту, який дозволяє уникнути електронної реєстрації, спаму та шкідливих додатків-ботів. Він добре відомий тим, що забезпечує безпеку, відрізняючи людей від комп'ютерів; наприклад, захист онлайн-опитувань від ботів для онлайн-голосування або шкідлива реєстрація електронної пошти [4].

Ґрунтуючись на оцінці джерел, ми розглянемо тест CAPTHCA, який легко вирішити людям і важко розв'язати комп'ютерам[11]. Вона вважається успішною, якщо її успішність у прийнятті людських рішень перевищує 90%, а комп'ютери досягають успіху менше 1%.

1.4 Сучасні проблеми захисту за допомогою CAPTCHA

Очевидно, що підтвердження якості CAPTHA вимагає регулярних всебічних аудитів для отримання знань про поточні можливості машин. Безсумнівно, що вирішення таких тестів цілком пов'язане з потужністю і якістю

алгоритму. Що до першого аспекту, згідно із законом Мура, обчислювальна потужність зростає в геометричній прогресії []. У той же час, що стосується другого, штучний інтелект став новим кроком у якісних підходах до розпізнавання CAPTCHA.

Ще в 2010 році БЕН Е. К. Бойтер у своїй роботі ALL About Captha's [8] описав метод розпізнавання текстової CAPTHCA. Він складається з декількох етапів, на першому етапі вирізаємо зображення з текстом, потім перетворюємо його в числовий масив, після чого прибираємо колірні спотворення і відновлюємо текст з виділеними буквами. Останній крок – розпізнавання символів за допомогою заздалегідь навченої саме для цього нейромережі. Схематично цей процес зображений на малюнку 1.4.

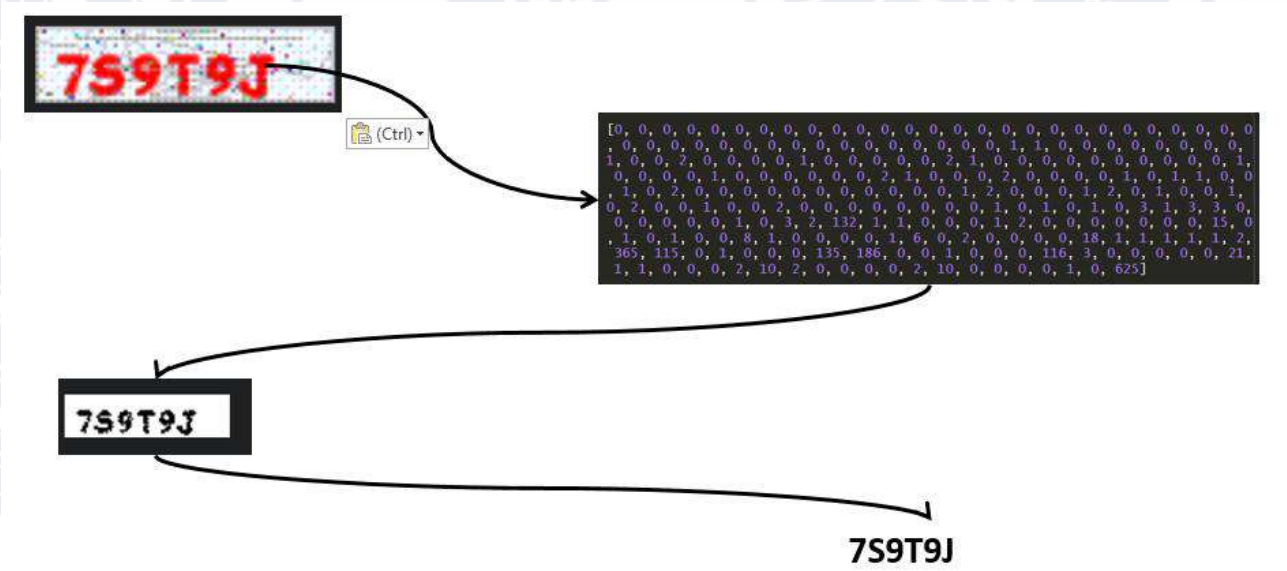

Рисунок 1.4 – Схематичне зображення поетапного розпізнавання CAPTCHA

Таким чином, можна зробити висновок, що текстова CAPTHCA не ефективна.

1.5 Власний експеримент

Щоб оцінити поточні можливості загальнодоступних нейронних мереж, проведемо свій невеликий експеримент. Для нього ми знайдемо одне із зображень, яке часто використовується як CAPTHA, а саме рисунок 1.5.

Рисунок 1.5 – Зображення для розпізнавання в експерименті

Після цього ми скористаємося Google Cloud Vision Api[12]. Сценарій реалізації цього експерименту показаний нижче.

Лістинг коду:

```
from google.cloud import vision
from google.cloud.vision_v1 import types
import io
import os

os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "credentials.json"


image_path = 'hydrant_image.jpg'

client = vision.ImageAnnotatorClient()

with io.open(image_path, 'rb') as image_file:
    content = image_file.read()

image = types.Image(content=content)

response = client.object_localization(image=image)
objects = response.localized_object_annotations

possible_items = []
for object_ in objects:
    possible_items.append(object_.name)

print(possible_items)
```

В результаті гідрант на зображенні був розпізнаний. Результат показаний на малюнку 1.6. Таким чином, ми підтвердили нашу гіпотезу про можливість використання нейронних мереж для злому CAPTHA на основі зображень.

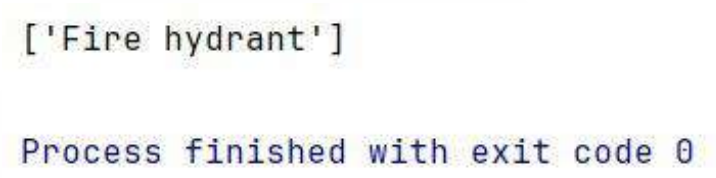

Рисунок 1.6 – Результат розпізнавання зображення 1.5

Слід зазначити, що важливість вивчення можливостей відкритих нейронних мереж полягає в тому, що при їх здатності здійснювати DDOS-атаку вона може переміститися з цільового сегменту в одинокого кіберзлочинця, за рахунок різкого спрощення доступу до цільового ресурсу.

Висновки до розділу 1

У першому розділі в результаті аналізу було визначено охоплення IT-сфери у світі, виявлено існуючі загрози для неї. Розглянуто принципи DDOS-атак, а також способи захисту від неї. Був проведений власний експеримент, що підтверджує актуальність теми. В результаті ми отримали чітко сформульовану актуальну задачу і глибоке розуміння причин необхідності її вирішення.

РОЗДІЛ 2. МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА РОЗРОБКА АЛГОРИТМІВ

Перш ніж приступити до написання програми, важливо провести аналіз предметної області. В рамках чого розглядаються лише ті об'єкти, які необхідні для опису вимог та умов вирішення конкретного завдання. Процес виявлення компонентів предметної області називається її концептуалізацією.

Щоб створити якісну інформаційну систему, необхідно не лише зрозуміти процеси, що відбуваються у системі, але й мати чітке уявлення про інформацію, якою система повинна керувати. Це потребує знання об'єктів, що входять у предметну область інформаційної системи, та логічних зв'язків між ними.

Основною метою побудови моделі предметної області є створення графічного зображення логічної структури досліджуваної предметної області.

Для успішного створення інформаційної системи необхідно глибоко розуміти, якою інформацією вона повинна управляти, а також як ці дані взаємодіють та пов'язані між собою.

2.1 Розробка BPMN діаграми

BPMN (Business Process Model and Notation) діаграми ідеально підходять для моделювання предметної області завдяки своїй здатності наочно і структуровано відображати бізнес-процеси[13]. Вони дозволяють деталізовано описати всі етапи процесів, включаючи дії, події, рішення та взаємодії між різними учасниками. Завдяки зрозумілому графічному відображенню, BPMN діаграми роблять процеси прозорими та легко зрозумілими для всіх зацікавлених сторін, від бізнес-аналітиків до технічних фахівців, що сприяє кращій комунікації і спільному розумінню вимог до системи.

Крім того, BPMN діаграми підтримують стандартизований підхід до моделювання, що полегшує інтеграцію різних частин системи і забезпечує їхню узгодженість. Використання BPMN сприяє точному визначенню та документуванню бізнес-логіки, що є критично важливим для розробки

інформаційних систем. Завдяки своєму універсальному характеру, BPMN дозволяє створювати моделі, які можуть бути легко змінені або доповнені відповідно до нових вимог, що робить процес розробки більш гнучким і адаптивним до змін у бізнес-середовищі.

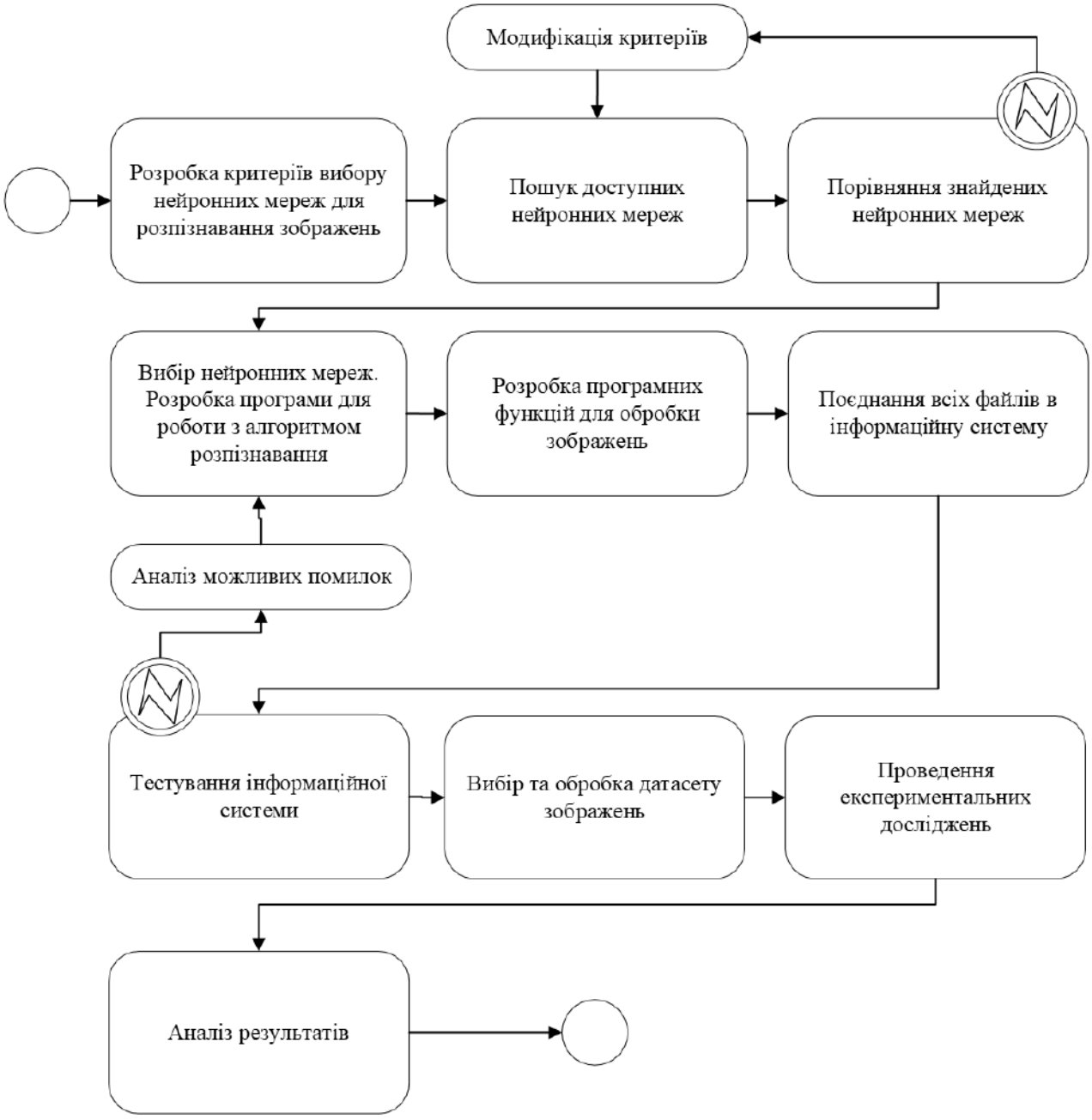

Рисунок 2.1 – BPMN діаграма процесу

2.2 Формалізація критеріїв порівняння нейронних мереж

Для проведення аналізу необхідно розробити чіткі та обґрунтовані критерії, які дозволять систематизувати існуючі нейронні мережі. В рамках цього

дослідження, за озвученими нижче критеріями, ми зосередимося на вивченні загальнодоступних нейронних мереж. Сформуємо перелік критеріїв.

- Загальнодоступність
- Кількість безкоштовних запитів – потрібно багато ітерацій, щоб дослідити та перевірити якість розпізнавання зображень. Якщо кількість безкоштовних запитів обмежена, це може ускладнити проведення повноцінних тестів.
- Швидкість розпізнавання зображень - час, витрачений на розпізнавання зображень, має вирішальне значення для DDOS-атак, тому в рамках цього розпізнавання предметів нейронною мережею мають бути виконані досить швидко.
- Точність розпізнавання зображень є основним показником якості нейронної мережі. Точність розпізнавання безпосередньо впливає на корисність і надійність системи в практичному застосуванні.
- Можливість роботи через API – аспект, який визначає можливість створення зручної автоматизованої системи розпізнавання зображень що не використовує значні технічні ресурси.
- Необхідність реєстрації є одним із потенційно обмежуючих застосувань нейронної мережі, наприклад, необхідність підтверджувати свою особу під час надсилання кожного запиту на визнання була б суттєвим недоліком. Також існує можливість обмежень використання деяких функцій у певних країнах.
- Ресурсозатратність – алгоритм не повинен використовувати значні технічні ресурси.

Крім визначення критеріїв, важливо також встановити порогові значення, перевищення яких призведе до неможливості або недоцільності використання даної системи для подальших досліджень, а також важливо визначити метод оцінки цих критеріїв. Критерій публічної доступності та можливості роботи через API є бінарними, і в необхідності реєстрації визначимо три стани: «без реєстрації», «одноразова реєстрація», «потрібна постійна перереєстрація».

Мінімальний поріг кількості запитів для максимально повного розслідування буде встановлено на рівні 800 запитів, а максимальна швидкість розпізнавання запитів становитиме 1 секунду, інакше людина швидше розв'яже CAPTHA і використання автоматизації втратить будь-який сенс.

Критерій точності розпізнавання зображень вимагає більш детального розгляду, так як нейромережі тренуються на різних наборах зображень, у них різна точність розпізнавання різних типів зображень, наприклад, ШІ, навчений розрізняти дерева, не зможе розпізнавати геометричні фігури і навпаки. Таким чином, для тестування був створений набір з 10 зображень з різною тематикою, які, на суб'єктивну думку, досить поширені в CAPTCHA. Пороговим значенням буде розпізнавання не менше 40% зображень, тобто 4 з 10.

2.3 Порівняльний аналіз наявних алгоритмів розпізнавання

Для аналізу було обрано 8 популярних нейронних мереж, а результат їх порівняння представлений у таблиці 1.

Таблиця 2.1 – порівняльний аналіз наявних загальнодоступних нейронних мереж для розпізнавання зображень

	Google Cloud Vision API	IBM Watson Visual Recognition	Clarifai	Microsoft Azure Computer Vision	Amazon Rekognition	YOLO. YOLOv8	TensorFlow API with MobileNetV2	TensorFlow API with ResNet
Доступність	Часткова	Частково	Частково	Частково	Частково	Повна	Повна	Повна

Кількість безкоштовних запитів	1000 запитів/місяць	1000 запитів/місяць	5000 запитів/місяць	5000 запитів/місяць	5000 запитів/місяць	Необмежено	Необмежено	Необмежено
Ресурсозатратність	низька	низька	низька	низька	низька	висока	висока	висока
Можливість роботи по API	Так	Ні	Так	Так	Так	Ні	Так	Так
Необхідність реєстрації	Так	Так	Так	Так	Так	Ні	Ні	Ні
Точність розпізнавання предметів на зображенні	80%	--	80%	--	--	60%	30%	40%
Швидкість розпізнавання зображень	0.5	--	0.8	--	1	<0.1	<0.1	<0.1

В цій таблиці було наведено порівняльний аналіз 8 алгоритмів. Дані в таблиці базуються як на офіційних джерелах так і на власних експериментах. Наприклад, більшість моделей мають значно більшу точність розпізнавання, однак, так як тут визначалась точність на довільному датасеті, дані спотворені.

Нижче наведені основні переваги або недоліки кожного з алгоритмів переставлених у таблиці.

2.4 Вибір алгоритмів для дослідження

- Google Cloud Vision API:

Перевагами Google Cloud Vision API є потужність для розпізнавання об'єктів у зображеннях, що має велику кількість вбудованих функцій, включаючи розпізнавання обличчя, маркування зображень та визначення тексту. Користувачі можуть легко інтегрувати його з іншими службами Google Cloud[12]. Він забезпечує 1000 безкоштовних запитів на місяць, що робить його доступним для невеликих проектів.

- YOLO:

YOLO (You Only Look Once) є швидким та ефективним алгоритмом з відкритим кодом та вже навченими моделями для розпізнавання об'єктів у реальному часі. На відміну від хмарних API, YOLOv8 працює локально, пропонуючи повну доступність і без обмежень на безкоштовні запити[14]. Однак висока потреба в ресурсах означає, що для нього потрібне потужне обладнання, що може бути недоліком для тих, хто не має доступу до таких ресурсів. Хоча для цього не потрібна реєстрація чи доступ до Інтернету, його впровадження та налаштування можуть бути складними та трудомісткими, особливо для розробників, які не мають досвіду роботи зі структурами глибокого навчання.

- Clarifai:

Clarifai відомий своїми надійними можливостями розпізнавання зображень і відео, пропонуючи такі функції, як виявлення об’єктів, розпізнавання сцен і навчання користувацьких моделей[15]. Він забезпечує 5000 безкоштовних запитів на місяць, що робить його щедрим варіантом для розробників. Однак, як і інші API, він вимагає реєстрації користувача та його користування вимагає регулярної витратити грошей. Незважаючи на високу точність і зручність використання, розробники можуть зіткнутися з проблемами під час інтеграції в

нестандартні системи чи середовища. Залежність API від постійного доступу до Інтернету також може бути обмеженням для автономних програм.

- Microsoft Azure Computer Vision:

Microsoft Azure Computer Vision забезпечує комплексний аналіз зображень, включаючи OCR, виявлення об'єктів тощо[16]. Він пропонує 5000 безкоштовних запитів на місяць, але вимагає облікового запису Microsoft. Незважаючи на перспективні для цього дослідження оглядові параметри, інтеграція з Azure може бути складною, а служба може бути надмірною для простих програм. Користувачі також повинні знати, що залежність платформи від інфраструктури Microsoft може спричинити проблеми сумісності з екосистемами, що не належать Microsoft.

- Amazon Rekognition:

Amazon Rekognition пропонує широкий спектр функцій аналізу зображень і відео, таких як розпізнавання обличчя, виявлення об'єктів і розпізнавання активності[17]. Він забезпечує 5000 безкоштовних запитів на місяць. Однак для налаштування служби потрібен зареєстрований обліковий запис Amazon Web Services (AWS), включаючи підтвердження кредитної картки, що може бути проблематичним, так як не всі картки підтримуються. Ця вимога може бути серйозною перешкодою для деяких користувачів.

- IBM Watson Visual Recognition

IBM Watson Visual Recognition дозволяє аналізувати зображення та відео за допомогою попередньо підготовлених моделей і спеціального навчання моделям[18]. Він пропонує 1000 безкоштовних запитів на місяць, але його часткова доступність і останні проблеми з отриманням ключів API перешкоджають його використанню. Деякі користувачі повідомили про труднощі доступу до API через збої на сайті або проблеми з інтерфейсом. Крім того, для Watson потрібен зареєстрований обліковий запис IBM Cloud, що може стати проблемою для користувачів, які не знайомі з екосистемою IBM. Ці проблеми в поєднанні з обмеженою доступністю роблять не зручним його практичне використання.

- TensorFlow API з MobileNetV2

API TensorFlow із MobileNetV2 забезпечує баланс між продуктивністю та обчислювальною ефективністю, що робить його придатним для мобільних і вбудованих пристроїв. Він працює локально, забезпечуючи повну доступність і без обмежень на безкоштовні запити[19]. Однак це вимагає значних ресурсів для навчання та розгортання. Незважаючи на навчання на наборі даних COCO[21], його точність розпізнавання об'єктів відносно низька, близько 30%. Відсутність надійних попередньо навчених моделей може вимагати додаткового спеціального навчання, що не є частиною цього дослідження.

- TensorFlow API з ResNet

API TensorFlow з ResNet забезпечує більш надійну архітектуру глибокого навчання для завдань розпізнавання зображень із покращеною точністю порівняно з MobileNetV2. Він також працює локально, пропонуючи повну доступність і без обмежень на безкоштовні запити[20]. Однак, як і MobileNetV2, він ресурсномісткий і потребує значної обчислювальної потужності як для навчання, так і для висновків. Незважаючи на те, що він навчався на наборі даних COCO[21], точність його розпізнавання становить близько 40%, що може потребувати додаткового спеціального навчання для покращення. Ця потреба в додаткових ресурсах і досвіді може стати перешкодою для дослідження.

На основі аналізу, проведеного в попередньому розділі, було обрано три нейронні мережі для детального дослідження: Google Cloud Vision API, YOLO та Clarifai. Вибір саме цих алгоритмів зумовлений тим, що у них гарні характеристики, вони відповідають встановленим критеріям та вони не викликають різноманітних проблем з налаштуванням системи.

Висновки до розділу 2

У першій частині другого розділі було проведено моделювання предметної області, в результаті якого була отримана BPMN-діаграма процесу дослідження якості розпізнавання зображень нейронними мережами. Результати цього розділу

допоможуть в майбутньому чітко слідувати зазначеному плану для досягнення поставленої мети.

У другій частині розділу були сформовані критерії порівняння штучного інтелекту, після чого проведено порівняльний аналіз існуючих систем, за результатами якого було обрано 3 нейронні класифікатори. Також в цьому розділі були описані ключові особливості розглянутих нейронних класифікаторів.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

У цьому розділі будуть ретельно описані усі етапи розробки інформаційної системи, а також етапи експериментального дослідження.

3.1 Вибір технічного стеку

Для успішного створення інформаційної системи необхідно звернути увагу на два аспекти, а саме на вибір мови програмування та IDE.

Мовою програмування було обрано Python. Вибір Python як основної мови програмування для розробки інформаційної системи розпізнавання зображень є пріоритетним з кількох ключових причин. По-перше, Python пропонує багату екосистему бібліотек і фреймворків, таких як TensorFlow, Keras, PyTorch, OpenCV і scikit-image, які полегшують розробку та розгортання алгоритмів машинного навчання та комп'ютерного зору. У цих бібліотеках передбачені готові модулі для обробки зображень, побудови та навчання нейронних мереж, що значно скорочує час розробки та дозволяє зосередитися на вирішенні конкретних завдань проекту. По-друге, Python має високу читабельність і простий синтаксис, що полегшує написання та підтримку коду. Це особливо важливо в проектах, пов'язаних з розпізнаванням зображень, де розробка може включати складні алгоритми і великі обсяги даних. Крім того, велика та активна спільнота розробників Python надає доступ до широкої бази знань, документації та зразків коду, що може бути надзвичайно корисним під час вирішення проблем та впровадження нових функцій.

PyCharm було обрано як інтегроване середовище розробки (IDE).

По-перше, PyCharm надає потужні інструменти для розробки на Python, включаючи інтелектуальне автозаповнення, аналіз, рефакторинг і налагодження. Ці функції значно прискорюють процес кодування та допомагають розробникам уникнути помилок, що особливо важливо при роботі зі складними алгоритмами машинного навчання та комп'ютерного зору. Крім того, вбудована підтримка

віртуальних середовищ і систем контролю версій полегшує керування залежностями та спільну роботу над проектом.

По-друге, PyCharm пропонує інтеграцію з популярними бібліотеками та фреймворками для машинного навчання та аналізу даних, такими як TensorFlow, Keras, PyTorch та іншими. Це дозволяє розробникам працювати в єдиному середовищі, ефективно керувати проектами та легко перемикатися між такими завданнями, як розробка моделі, навчання та оцінка. PyCharm також підтримує візуалізацію даних і роботу з Jupyter Notebook, що робить його універсальним інструментом для всіх етапів розробки системи розпізнавання зображень. Таким чином, PyCharm не тільки прискорює процес розробки, але й забезпечує високий рівень зручності та ефективності, що робить його ідеальним вибором для таких проектів.

Також доцільно використовувати Jupyter Notebook з наступних причин. Jupyter Notebook забезпечує інтерактивне середовище, ідеальне для експериментів з алгоритмами машинного навчання та обробки зображень. Розробники можуть писати і виконувати код поетапно, спостерігаючи за результатами чи візуалізуючи дані прямо в документі. Це значно прискорює процес досліджень і розробок, дозволяючи швидко перевіряти гіпотези, корегувати параметри моделі, аналізувати результати.

Крім того, Jupyter Notebook підтримує комбінацію коду, тексту, зображень і графіків в одному документі, що робить його чудовим інструментом для створення репрезентативної документації та представлення результатів. Це особливо корисно для командної співпраці та комунікації із зацікавленими сторонами, оскільки дозволяє візуалізувати прогрес, результати та висновки. Таким чином, використання Jupyter Notebook не тільки полегшує процес розробки, а й сприяє кращій комунікації та обміну знаннями між учасниками проекту.

3.2 Програмна реалізація інформаційної системи

Як показує практика серед великих компаній, перед реалізацією великого і складного проекту потрібно розробити план його виконання. Одним із зручних способів представлення майбутньої програмної реалізації є блок-схема - графічне представлення алгоритму або процесу, яке використовує стандартні символи для показу різних типів дій та їх послідовності.

Візуалізація майбутньої програми дозволить заздалегідь продумати всі логічні та структурні елементи коду, що згодом зменшить кількість помилок. Також блок-схема дозволить виявити можливі логічні невідповідності на ранній стадії, що значно скоротить час розробки. Крім усього іншого, блок-схема - це загальнодоступний і зрозумілий спосіб представлення структури інформаційної системи, тому її створення дозволить зрозуміти принципи роботи майбутньої програми в легкій і доступній формі, не вникаючи в подробиці архітектури та коду.

3.2.1 Створення власної блок-схеми

Виходячи з наведених вище правил, давайте створимо власну блок-схему.

Першим кроком завантажуємо зображення, після чого перетворюємо його на матрицю пікселів у форматі RGB. За допомогою цього виду ви можете застосувати фільтр до зображення (для цілей цієї роботи ми розглянемо відправку вихідного зображення, ідентичного перетворенню). Після цього зображення надсилається за допомогою API. Потім відбувається розгалуження: якщо зображення було розпізнано, ми визначаємо, чи правильно воно було розпізнано і збільшуємо відповідні лічильники, які відображають якість розпізнавання зображення (нерозпізнане, розпізнане, помилкове, розпізнане, правильне), в іншому випадку ми просто збільшуємо лічильник нерозпізнаних зображень. Далі перевіряємо, чи не залишилося зображень для розпізнавання, і якщо є, повторюємо всі описані дії, в іншому випадку припиняємо виконання програми.

Візуалізація описуваної програми представлена на структурній схемі на рисунку 3.1

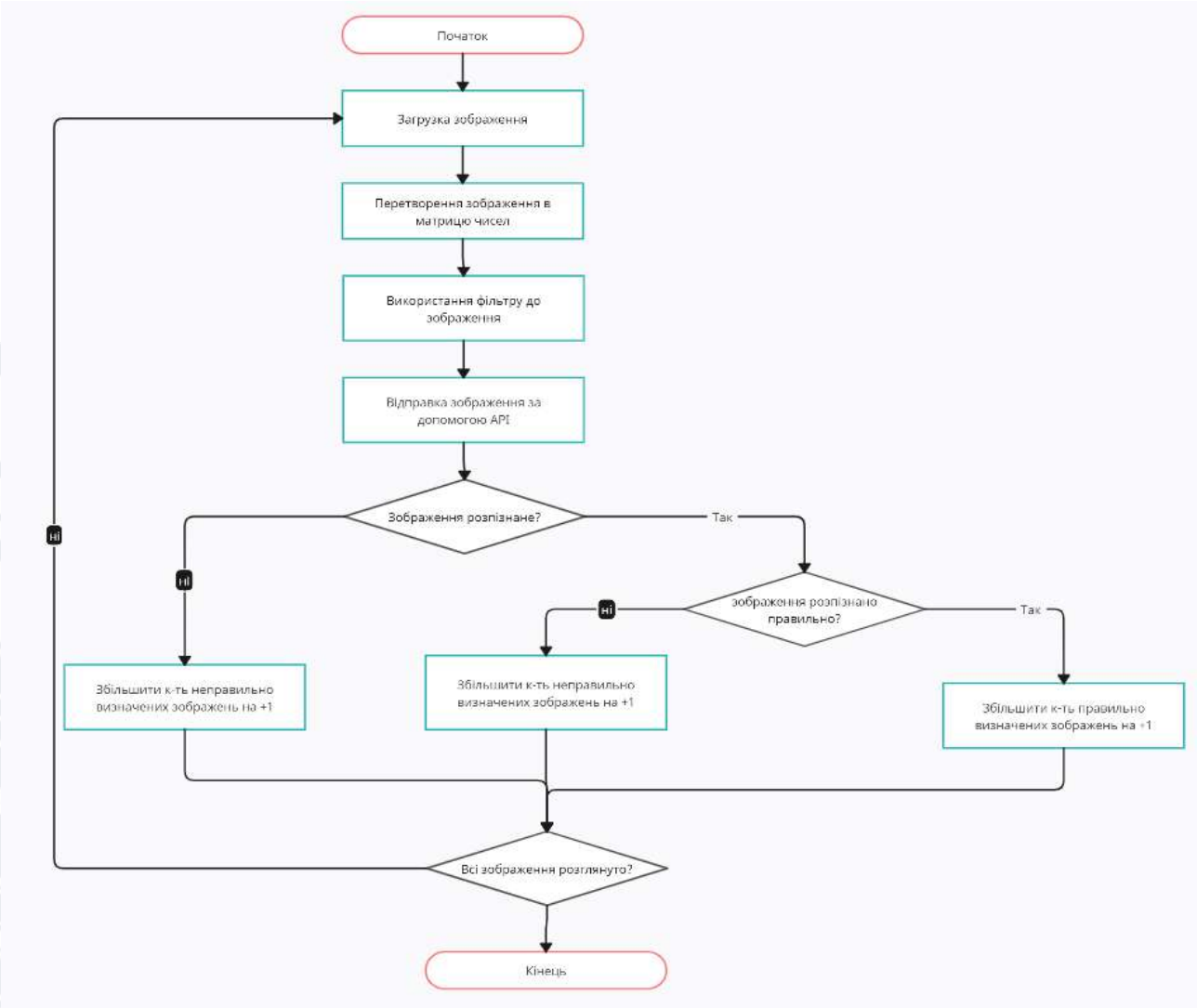

Рисунок 3.1 - Блок-схема інформаційної системи

3.3 Типи фільтрів, що застосовуються до зображень

У цьому розділі ми зосередимося на видах фільтрів, способах їх застосування, програмній реалізації методів їх застосування, а також висунемо гіпотези, чому це може ускладнити розпізнавання зображень для нейронної мережі.

У роботі були вивчені наступні трансформації:

- Ідентична трансформація
- Накладання Ґауссівського шуму
- Видалення каналів кольору зображення
- Інвертування кольорів

- Сегментація зображень
- Видалення частини пікселів

3.3.1 Ідентична трансформація

Ця трансформація була запроваджена з метою стандартизації блок-схеми. За визначенням, ідентичне перетворення зображення - це операція, при якій вихідне зображення залишається незмінним після застосування цього перетворення. Іншими словами, кожен піксель вихідного зображення залишається на своєму місці, а його значення кольору або яскравості залишаються незмінними. Таке перетворення можна представити математично як застосування ідентичної функції до кожного пікселя зображення, яка повертає те саме значення в незмінному вигляді

Цей «фільтр», на відміну від інших, використовується для визначення якості розпізнавання нейронної мережі в цілому.

Рисунок 3.2 – Результат ідентичного трансформування

3.3.2 Ґауссівський шум

Операція застосування Ґауссівського шуму починається зі статистичної теорії, зокрема, в роботі з моделювання випадкових процесів. Ідея полягає в тому, щоб додати випадковий Ґауссівський шум до зображення, щоб імітувати природні артефакти або спотворення, які можуть виникнути в процесі зйомки або передачі даних[22]. Ця методика активно використовується в обробці зображень і в алгоритмах комп'ютерного зору для аналізу продуктивності алгоритмів при різних рівнях шуму, а також для створення синтетичних зображень в дослідженнях.

Лістинг коду:

```
def add_gaussian_noise(image, mean=0, sigma=25):

    img_array = np.array(image)

    row, col, ch = img_array.shape
    gauss = np.random.normal(mean, sigma, (row, col, ch))

    noisy_image = img_array + gauss

    noisy_image = np.clip(noisy_image, 0, 255)

    noisy_image = Image.fromarray(np.uint8(noisy_image))

    return noisy_image
```

Результат застосування Ґауссівського шуму показаний на рисунку 3.3

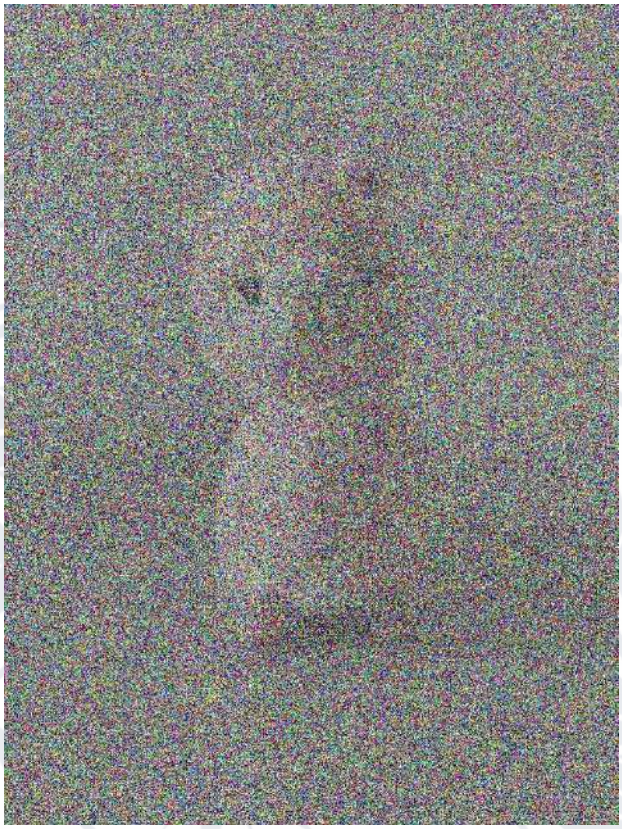

Рисунок 3.3 – Результат Ґауссівського шуму

Гіпотеза що до ефективності цього перетворення основується на тому, що аналізувати як контур зображення, так і його кольори для алгоритму має бути складно. Важливо відзначити, що збільшення параметру δ може значно погіршити ідентифікацію об'єкта на зображенні для людини, тому потрібне більш ретельне налаштування параметрів перетворення.

3.3.3 Видалення каналів зображення

У всіх попередніх випадках ми розкладали зображення на набір пікселів. При цьому у ми отримували матрицю у вигляді RGB. GB (Red, Green, Blue) — це модель представлення кольорів, що описує кольори на основі їх складових червоного, зеленого та синього кольорів. Кожен піксель на зображенні RGB представлений поєднанням цих трьох основних кольорів. Кожен компонент (червоний, зелений і синій) визначається значенням від 0 до 255, де 0 означає відсутність кольору взагалі (чорний), а 255 - максимальна насиченість кольору. Поєднуючи різні рівні червоного, зеленого та синього, можна створити мільйони відтінків і кольорів, що робить RGB одним із найпопулярніших форматів для

представлення кольорових зображень у цифровій графіці. Кожен піксель на зображенні RGB представлений трійкою чисел, де кожне число визначає інтенсивність відповідного кольору. Ці дані можуть бути оброблені та відображені за допомогою програмного забезпечення або пристроїв, здатних інтерпретувати та відображати кольори на основі значень RGB.

З математичної точки зору, видалення одного з каналів зв'язку - це присвоєння нульового значення відповідному компоненту кожного пікселя. Наприклад, один з пікселів має значення (255; 190; 236), якщо прибрати синій канал на зображенні, то отримаємо (255; 190; 0).

Лістинг коду:

```
def remove_channel(image):
    r, g, b = image.split()

    b = b.point(lambda x: 0)
    g = g.point(lambda x: 0)

    result_image = Image.merge('RGB', (r, g, b))

    return result_image
```

Результат застосування такого перетворення, з видаленням зеленого і синього каналів, показаний на рисунку 3.4

Рисунок 3.4 – Результат видалення каналів

Передбачається, що за рахунок видалення одного або декількох каналів зображення значно зменшить обсяг інформації про колір, таким чином робота класифікаторів на основі контрасту, теми, композиції та інших колірних атрибутів зображення, що згодом погіршить розпізнавання зображень ШІ.

3.3.4 Інвертування кольорів

Ідея інвертування кольорів при обробці зображень виникла з необхідності створювати ефекти або візуально покращувати зображення. Інверсія кольору може бути використана для створення ефектів виділення, підкреслення контрасту або просто зміни візуального сприйняття. Ця техніка часто використовується у фотографії, графічному дизайні та цифровому мистецтві для створення цікавих та унікальних ефектів. Крім того, інвертування кольорів також може бути корисним для корекції балансу кольорів на зображеннях або для перетворення зображень на чорно-білі. При цьому важливо враховувати, що крайні випадки інверсії кольору, навпаки, призведуть до труднощів у сприйнятті зображення

З математичної точки зору, інвертування кольорів означає віднімання від максимального значення кольору, тобто 255, поточного значення кольору пікселя. Наприклад, один з пікселів має значення (255; 190; 236), в результаті інвертування кольорів отримуємо (0; 65; 19).

Лістинг коду:

```
def invert_colors(image):
    inverted_image = Image.eval(image, lambda x: 255 - x)

    return inverted_image
```

Результат накладання перетворення інвертування кольору показаний на рисунку 3.5.

Рисунок 3.5 – Результат інвертування кольорів

Так само, як і видалення одного з каналів зображення, наша мета полягає в тому, щоб ускладнити нейронній мережі вилучення інформації про колір. Відповідно, очікування впливу на розпізнавання зображень аналогічні попередній трансформації.

3.3.5 Сегментація зображень

Ідея сегментації зображень полягає в тому, що ми повинні розділити наше зображення на кілька частин, після чого ми випадковим чином перемішуємо його компоненти.

Лістинг коду:

```
    def split_reverse_merge(image, num_parts=6):
        width, height = image.size
        part_width = width // num_parts
        parts = []
        for i in range(num_parts):
            left = i * part_width
            upper = 0
            right = left + part_width
            lower = height
            part = image.crop((left, upper, right, lower))
            parts.append(part)

        reversed_parts  =  [part.transpose(Image.FLIP_LEFT_RIGHT)
for part in parts]

        merged_image = Image.new('RGB', (width, height))
        offset = 0
        for part in reversed_parts:
            merged_image.paste(part, (offset, 0))
            offset += part.width

    return merged_image
```

Результат накладення трансформації сегментації шляхом поділу зображень на 6 горизонтальних ділянок показаний на рисунку 3.6

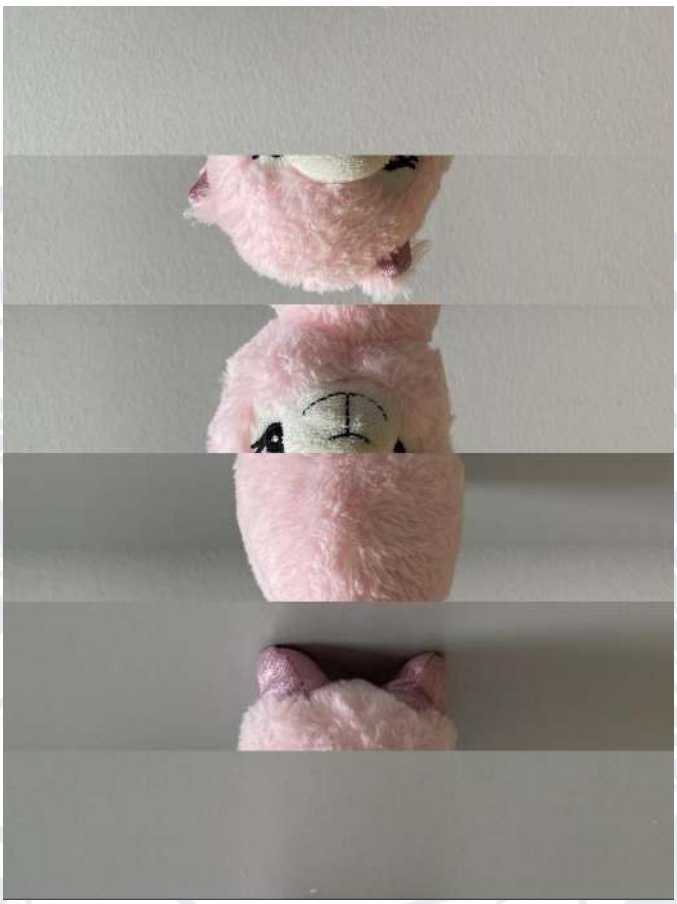

Рисунок 3.6 – Результат сегментації

Припущення про ефективність цього перетворення ґрунтується на тому, що ми змінюємо контур зображення, що згодом має спричинити погіршення розпізнавання зображень нейронними мережами, оскільки виявити чіткі межі зображення звичними методами не вийде.

Важливо відзначити, що збільшення кількості сегментів на зображенні може значно погіршити ідентифікацію об'єкта на зображенні для людини, тому потрібне більш ретельне налаштування параметрів перетворення.

3.3.6 Видалення частини пікселів

Ідея видалення частини пікселів полягає в тому, щоб зменшити кількість інформації в ньому в цілому, це перетворення відрізняється від класичних алгоритмів стиснення тим, що ми не заповнюємо вирізані пікселі якимись альтернативними значеннями, а від класичної обрізки зображення тим, що ми не зменшуємо розмір зображення.

Лістинг коду:

```
def remove_every_second_pixel(img):
    width, height = img.size

    new_img = Image.new("RGB", (width, height), (0, 0, 0))

    for y in range(height):
        for x in range(width):
            if (x + y) % 2 != 0:
                pixel = img.getpixel((x, y))
                new_img.putpixel((x, y), pixel)

    return new_img
```

Результат накладного перетворення видалення частини пікселів, в нашому випадку кожного другого пікселя, показаний на рисунку 3.7.

Рисунок 3.7 – Результат видалення частини пікселів

Припущення про ефективність цього перетворення ґрунтується на тому, що ми в цілому зменшуємо обсяг інформації, що підлягає аналізу нейронною мережею, і аналізувати як контур зображення, так і його кольори має бути складно.

Також дане перетворення має значний діапазон налаштувань, як від вибору пікселів, які ми будемо видаляти, так і від їх кількості та розташування.

3.4 Програмна реалізація

Після проведення порівняльного аналізу, підбору відповідних нейронних мереж і після того, як методи перетворення зображень вивчені, описані і запрограмовані, приступимо до програмної реалізації інформаційної системи за блок-схемою. Код програмної реалізації представлений у додатку А, а структура каталогу з інформаційною системою наведена на рисунку 3.8.

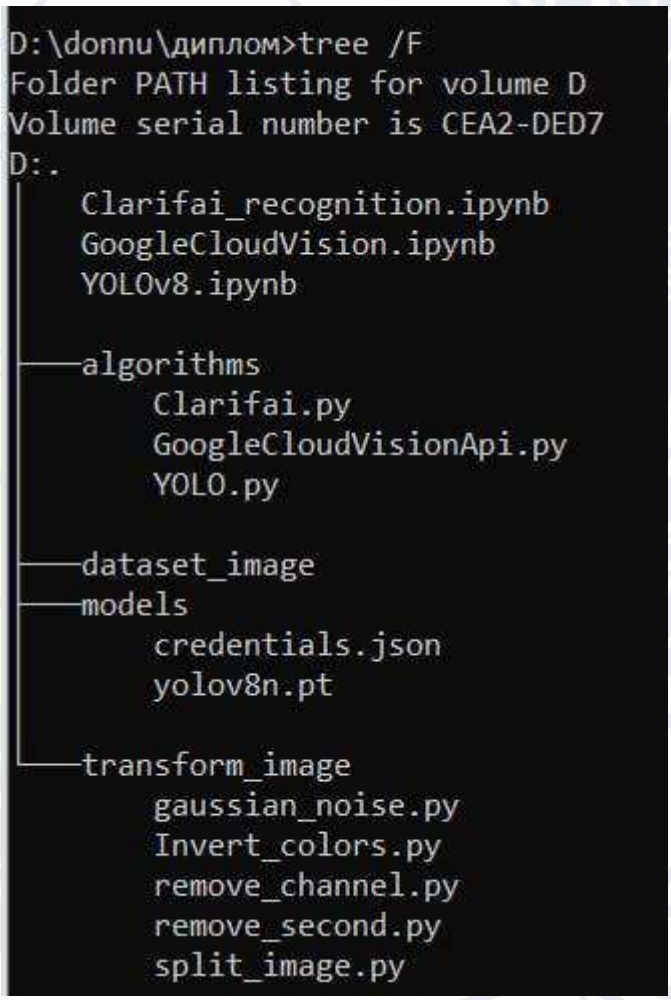

Рисунок 3.8 – Структура каталогу інформаційної системи

У цьому каталозі зібрані чотири субкоталоги та три зошити Юпітера:

- Clarifai_recognition.ipynb: блокнот Jupyter з основним блоком коду.
- GoogleCloudVision.ipynb: блокнот Jupyter з основним блоком коду.

- YOLOv8.ipunb: блокнот Jupyter з основним блоком коду.
- algorithms каталог: містить скриптами Python для реалізації налаштування алгоритмів для розпізнавання. Функції що повертають назву предмета на зображенні.
- dataset_image каталог: зберігаються зображення для розпізнавання.
- models каталог: зберігає навчені моделі або посилання на них.
- transform_image каталог: містить скрипти Python, для трансформації зображень

Ця програма буде використовуватися для подальших досліджень.

3.5 Результати експериментальних досліджень

Для проведення експериментального дослідження ми сформуємо власний набір даних, зазначимо, що дані в ньому зібрані виключно з відкритих джерел та відповідають усім законодавчим та етичним нормам.

Чудовим інструментом для пошуку доступних зображень є Kaggle[23]. Платформа Kaggle надає доступ до широкого спектру даних, включаючи набори даних з різних галузей, таких як медицина, фінанси, соціальні науки, а також зображення. Однак збір даних був ускладнений тим, що зображення найчастіше класифікуються в тематичні колекції на цій платформі, а наше дослідження вимагало різноманітних зображень для оцінки різних аспектів нейронних мереж. У зв'язку з цим ми зібрали власний набір даних зі 133 елементів з кількох готових наборів даних.

Зазначимо, що з урахуванням того, що кожне зображення піддавалося фільтру, всього було проаналізовано не менше 798 зображень, що є практично максимально доступним вільним значенням з обраних нейронних мереж.

Всі вивчені картинки можна розділити на наступні категорії:

- Об'єкти міської інфраструктури
- Транспорт
- Тварини

- Рослини
- Фрукти
- Предмети побуту

Дані категорії охоплюють основний спектр найпопулярніших CAPTHCA. Після складання набору зображень з урахуванням нашої інформаційної системи було проведено дослідження з якості розпізнавання зображень. Зведені результати представлені в таблиці 6.1, вона відображає для кожного виду перетворень та для кожної використаної нейронної мережі відсоток правильно розпізнаних зображень.

Таблиця 3.1 – статистичні оцінки результів експериментів

	YOLO	Clarifai	Google Cloud Vision API	Середній показник
Ідентична трансформація	97%	94%	96%	96%
Ґауссівський шум	51%	63%	85%	66%
Інвертування кольорів	51%	49%	68%	59%
Видалення каналу	45%	40%	63%	49%
Видалення частини пікселів	33%	62%	49%	48%
Сегментація зображення	61%	64%	68%	64%

Виходячи з отриманих даних, можна зробити висновок, що початкова якість розпізнавання зображень не впливає на якість розпізнавання перетворених зображень.

За результатами видно, що в середньому метод з видаленням частини пікселів виявився найбільш ефективним, у випадку з YOLO вдалося знизити

якість розподілу до 33%, що є значним результатом, але недостатнім для використання в CAPTHA за вимогами, наведеними в першому розділі,

Зазначимо, що Ґауссівський шум та сегментація зображення не мали значного впливу на якість нейромережі. Однак ці методи є залежними від параметрів, тому потрібно провести більше експериментів.

В цілому в рамках даної роботи слід зазначити, що жоден з методів захисту від нейронних мереж не виявився достатньо ефективним для використання в CAPTHA. Це вказує на те, що нейронні мережі стали дуже ефективним інструментом для розпізнавання зображень і потребують перегляду для формування тестів для штучного інтелекту та перевірок на основі зображень. При цьому можна виділити перспективний напрямок у вивченні робіт CAPTCHA, а саме видалення частини пікселів, також важливо відзначити можливість комбінування запропонованих методів, але вони вимагають подальшого більш детального вивчення.

Висновки до розділу 3

В рамках даного розділу був детально описаний і обґрунтований вибір технологічного стека для програмної реалізації майбутньої інформаційної системи. Мовою програмування обрано Python, а IDE – PyCharm. Також були чітко описані причини, за якими варто використовувати Jupyter Notebook.

Далі була представлена інформація про всі аспекти робіт з програмної реалізації інформаційної системи від етапу створення блок-схеми до її фактичного створення. В якості артефактів були отримані блок-схеми, а також програма. Важливо відзначити, що інформація в цьому розділі охоплює всі аспекти розробки програмного продукту, включаючи опис математичних основ перетворень зображень, приклади зображень після застосування фільтра, скрипти коду ключів, а також структуру каталогу і теоретичні посилання.

Також у цьому розділі було розглянуто практичне застосування даної інформаційної системи. Був проведений експеримент, який дозволив визначити поточні можливості нейронних мереж розпізнавати як звичайні зображення, так

і зображення з нанесеними фільтрами. Цей експеримент є лише одним з можливих, адже функціонал розробленого програмного комплексу дозволяє гнучко налаштовувати нашу інформаційну систему під різні типи завдань. Слід зазначити, що отримані результати можуть послужити основою для розробки тестів, стійких до нейронних мереж, що показує ефективність як інструменту розробленої нами системи для вирішення великих і складних завдань сучасної науки.

ВИСНОВКИ

У результаті проведеної роботи було досягнуто основну мету дослідження – вивчення якості розпізнавання об'єктів на зображеннях за допомогою нейронних мереж. В рамках роботи було проведено аналіз існуючих нейронних мереж, доступних у відкритих джерелах, і розроблено інформаційну систему, що дозволяє зручно взаємодіяти з різними нейронними мережами та застосовувати фільтри до зображень. Експериментально було визначено вплив перетворень зображень на якість їх розпізнавання.

Результати цієї роботи можуть бути використані для аудиту поточних можливостей нейронних мереж щодо розпізнавання зображень, зокрема CAPTCHA, а також для розробки методів тестування, стійких до нейронних мереж. Отримані дані та розроблена інформаційна система мають великий потенціал для подальших досліджень і практичних застосувань у галузі комп'ютерного зору та штучного інтелекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Number of internet and social media users worldwide as of April 2024. URL: https://www.statista.com/statistics/617136/digital-population-worldwide/ (дата звернення: 29.05.2024).
2. Смолин О. І., Олексюк В. П. Інтернет речей як технологічний феномен XXI століття.
3. Avast blocks all-time high of 1 billion unique cyber attacks per month. URL: https://press.avast.com/avast-blocks-all-time-high-of-1-billion-unique-cyber-attacks-per-month (дата звернення: 29.05.2024).
4. J. Mirkovic, G. Prier, P. Reiher. Attacking DDoS at the source.
5. Ben E. C. Boyter. Decoding CAPTCHA’s.
6. Jonathan Lazar, Jinjuan Feng, Tim Brooks, Genna Melamed, Brian Wentz, Jon Holman, Abiodun Olalere, and Nnanna Ekedebe. The soundsright captcha: an improved approach to audio human interaction proofs for blind users. In Proceedings of the SIGCHI conference on human factors in computing systems, pages 2267–2276. ACM, 2012.
7. R.R. Schaller, George Mason University. Moore's law: past, present and future.
8. Ben E. C. Boyter. ALL About Captha's.
9. Individuals using the Internet (% of population). URL: https://data.worldbank.org/indicator/IT.NET.USER.ZS (дата звернення: 29.05.2024).
10. Mordor Intelligence. URL: https://www.mordorintelligence.com/ (дата звернення: 29.05.2024).
11. Mohammad Moradi, MohammadReza Keyvanpour. CAPTCHA and its Alternatives: A Review.
12. Google Cloud Vision. URL: https://cloud.google.com/vision (дата звернення: 29.05.2024).

13. Michele Chinosi, Alberto Trombetta. BPMN: An introduction to the standard.
14. Ultralytics Documentation. URL: https://docs.ultralytics.com/ (дата звернення: 29.05.2024).
15. Clarifai. URL: https://www.clarifai.com/ (дата звернення: 29.05.2024).
16. Microsoft AI Vision. URL: https://azure.microsoft.com/en-us/products/ai-services/ai-vision (дата звернення: 29.05.2024).
17. AWS Rekognition. URL: https://aws.amazon.com/de/rekognition/ (дата звернення: 29.05.2024).
18. IBM Watson Visual Recognition. URL: https://mediacenter.ibm.com/media/IBM+Watson+Visual+Recognition/0_jbsmp6lq (дата звернення: 29.05.2024).
19. TensorFlow MobileNetV2. URL: https://www.tensorflow.org/api_docs/python/tf/keras/applications/MobileNetV2 (дата звернення: 29.05.2024).
20. TensorFlow ResNet. URL: https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet (дата звернення: 29.05.2024).
21. COCO Dataset. URL: https://cocodataset.org/#home (дата звернення: 29.05.2024).
22. Ameen Mohammed Abd-Alsalam Selami, Ahmed Freidoon Fadhil. A Study of the Effects of Gaussian Noise on Image Features.
23. Kaggle. URL: https://www.kaggle.com/ (дата звернення: 29.05.2024).

ДОДАТОК А

Лістинг файлів

gaussian_noise.py

```
import numpy as np
from PIL import Image


def add_gaussian_noise(image, mean=0, sigma=25):

    img_array = np.array(image)

    row, col, ch = img_array.shape
    gauss = np.random.normal(mean, sigma, (row, col, ch))

    noisy_image = img_array + gauss

    noisy_image = np.clip(noisy_image, 0, 255)

    noisy_image = Image.fromarray(np.uint8(noisy_image))

    return noisy_image
```

remove_channel.py

```
from PIL import Image


def remove_channel(image):
    r, g, b = image.split()

    b = b.point(lambda x: 0)
    g = g.point(lambda x: 0)

    result_image = Image.merge('RGB', (r, g, b))

    return result_image
```

remove_second.py

```
from PIL import Image
```

```
def remove_every_second_pixel(img):
    width, height = img.size

    new_img = Image.new("RGB", (width, height), (0, 0, 0))

    for y in range(height):
        for x in range(width):
            if (x + y) % 2 != 0:
                pixel = img.getpixel((x, y))
                new_img.putpixel((x, y), pixel)

    return new_img
```

invert_colors.py

```
from PIL import Image

def invert_colors(image):
    inverted_image = Image.eval(image, lambda x: 255 - x)

    return inverted_image
```

split_image.py

```
from PIL import Image

def split_reverse_merge(image, num_parts=6):
    width, height = image.size
    part_width = width // num_parts
    parts = []
    for i in range(num_parts):
        left = i * part_width
        upper = 0
        right = left + part_width
            lower = height
            part = image.crop((left, upper, right, lower))
            parts.append(part)
```

```
        reversed_parts = [part.transpose(Image.FLIP_LEFT_RIGHT)
for part in parts]

        merged_image = Image.new('RGB', (width, height))
        offset = 0
        for part in reversed_parts:
            merged_image.paste(part, (offset, 0))
            offset += part.width

    return merged_image
```

Clarifai.py

```
import requests
import base64
from io import BytesIO

def detect_objects_clarifai(image):
    api_key = '**********' # конфіденційно

    url    =    "https://api.clarifai.com/v2/models/general-image-
recognition/outputs"

    buffered = BytesIO()
    image.save(buffered, format="JPEG")
    encoded_image                                                    =
base64.b64encode(buffered.getvalue()).decode('utf-8')

    headers = {
        'Authorization': f'Key {api_key}',
        'Content-Type': 'application/json',
    }

    data = {
        'inputs': [
            {
                'data': {
                    'image': {
                        'base64': encoded_image
                    }
                }
            }
        ]
    }

    response = requests.post(url, headers=headers, json=data)

    if response.status_code == 200:
        result = response.json()
```

```
        concepts = result['outputs'][0]['data']['concepts']
        items = [(concept['name']) for concept in concepts]
        return items
    else:
        print("Помилка при відправці запита:",
response.status_code, response.json())
        return []
```

GoogleCloudVisionApi.py

```
from PIL import Image
from google.cloud import vision
from google.cloud.vision_v1 import types
import io
import os

os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "credentials.json"

def detect_objects_google(image):
    client = vision.ImageAnnotatorClient()

    with io.BytesIO() as output:
        image.save(output, format="JPEG")
        content = output.getvalue()

    image = types.Image(content=content)

    response = client.object_localization(image=image)
    objects = response.localized_object_annotations

    possible_items = []
    for object_ in objects:
        possible_items.append(object_.name)

    return possible_items
```

YOLO.py

```
from ultralytics import YOLO

model = YOLO('yolov8n.pt')


def detect_objects_yolo(image):
    results = model(image)
```

```
    if isinstance(results, list):
        results = results[0]

    detected_objects = results.names
    class_ids = results.boxes.cls

    objects = [detected_objects[int(class_id)] for class_id in
class_ids]

    return objects
```

ДОДАТОК Б

Лістинг з Jupyter Notebook

GoogleCloudVisionApi.ipynb

```
import os
import pandas as pd
from PIL import Image
#%%
from GoogleCloudVisionApi import detect_objects_google
from gaussian_noise import add_gaussian_noise
from Invert_colors import invert_colors
from remove_second import remove_every_second_pixel
from remove_channel import remove_channel
from split_image import split_reverse_merge
#%%
folder_path = 'dataset_image'
files = os.listdir(folder_path)
#%%
original_data = []
gauss_data = []
invert_data = []
pixel_data = []
channel_data = []
split_data = []
#%%
for file in files:
```

```
    if file.lower().endswith(('.jpg', '.png', '.jpeg')):  # 
Unified condition

        image_path = os.path.join(folder_path, file)

        image = Image.open(image_path)


        # Detect objects in the original image

        original_detection = detect_objects_google(image)

        original_data.append([file, original_detection])


        # Detect objects in the image with Gaussian noise

        noisy_image = add_gaussian_noise(image)

        gauss_detection = detect_objects_google(noisy_image)

        gauss_data.append([file, gauss_detection])


        # Detect objects in the inverted color image

        inverted_image = invert_colors(image)

        invert_detection = detect_objects_google(inverted_image)

        invert_data.append([file, invert_detection])


        # Detect objects in the image with every second pixel 
deleted

        pixel_image = remove_every_second_pixel(image)

        pixel_data.append([file, 
detect_objects_google(pixel_image)])


        # Detect objects in the removed blue and green channels 
image

        channel_image = remove_channel(image)
```

```
        channel_data.append([file,
detect_objects_google(channel_image)])


        # Detect objects in the split image

        split_image = split_reverse_merge(image)

        split_data.append([file,
detect_objects_google(split_image)])

#%%

df = pd.DataFrame(original_data, columns=['Image',
'original_detection'])


# Create dictionaries for mapping

gauss_dict = {image: objects for image, objects in gauss_data}

invert_dict = {image: objects for image, objects in invert_data}

channel_dict = {image: objects for image, objects in channel_data}

pixel_dict = {image: objects for image, objects in pixel_data}

split_dict = {image: objects for image, objects in split_data}


# Add new columns to DataFrame

df['gauss_data'] = df['Image'].map(gauss_dict)

df['invert_data'] = df['Image'].map(invert_dict)

df['channel_data'] = df['Image'].map(channel_dict)

df['pixel_data'] = df['Image'].map(pixel_dict)

df['split_data'] = df['Image'].map(split_dict)


# Remove rows with empty detections

df = df[df['original_detection'].apply(lambda x: len(x) > 0)]

df.reset_index(drop=True, inplace=True)
```

```
def calculate_accuracy(original, detected):
    if not original and not detected:
        return 1.0
    elif not original or not detected:
        return 0.0

    original_set = set(original)
    detected_set = set(detected)
    intersection = original_set.intersection(detected_set)

    return len(intersection) / len(original_set)

accuracy_scores = {
    'gauss_data': [],
    'invert_data': [],
    'channel_data': [],
    'pixel_data': [],
    'split_data': []
}

for index, row in df.iterrows():
    original = row['original_detection']
    for col in accuracy_scores.keys():
        detected = row[col]
        accuracy = calculate_accuracy(original, detected)
        accuracy_scores[col].append(accuracy)
```

```
average_accuracies = {col: sum(scores) / len(scores) for col,
scores in accuracy_scores.items()}


print("Середня точність для кожного стовпця:")

for col, avg_accuracy in average_accuracies.items():

    print(f"{col}: {avg_accuracy:.2f}")
```

Clarifai_recognition.ipynb

```
import os

import pandas as pd

from PIL import Image

#%%

from Clarifai import detect_objects_clarifai

from gaussian_noise import add_gaussian_noise

from Invert_colors import invert_colors

from remove_second import remove_every_second_pixel

from remove_channel import remove_channel

from split_image import split_reverse_merge

#%%

folder_path = 'dataset_image'

files = os.listdir(folder_path)

#%%

original_data = []

gauss_data = []

invert_data = []
```

```
pixel_data = []

channel_data = []

split_data = []

#%%

for file in files:

    if file.lower().endswith(('.jpg', '.png', '.jpeg')):  # 
Unified condition

        image_path = os.path.join(folder_path, file)

        image = Image.open(image_path)


        # Detect objects in the original image

        original_detection = detect_objects_clarifai(image)

        original_data.append([file, original_detection])


        # Detect objects in the image with Gaussian noise

        noisy_image = add_gaussian_noise(image)

        gauss_detection = detect_objects_clarifai(noisy_image)

        gauss_data.append([file, gauss_detection])


        # Detect objects in the inverted color image

        inverted_image = invert_colors(image)

        invert_detection = detect_objects_clarifai(inverted_image)

        invert_data.append([file, invert_detection])


        # Detect objects in the image with every second pixel 
deleted

        pixel_image = remove_every_second_pixel(image)
```

```
        pixel_data.append([file,
detect_objects_clarifai(pixel_image)])


        # Detect objects in the removed blue and green channels
image

        channel_image = remove_channel(image)

        channel_data.append([file,
detect_objects_clarifai(channel_image)])


        # Detect objects in the split image

        split_image = split_reverse_merge(image)

        split_data.append([file,
detect_objects_clarifai(split_image)])

#%%

df = pd.DataFrame(original_data, columns=['Image',
'original_detection'])


# Create dictionaries for mapping

gauss_dict = {image: objects for image, objects in gauss_data}

invert_dict = {image: objects for image, objects in invert_data}

channel_dict = {image: objects for image, objects in channel_data}

pixel_dict = {image: objects for image, objects in pixel_data}

split_dict = {image: objects for image, objects in split_data}


# Add new columns to DataFrame

df['gauss_data'] = df['Image'].map(gauss_dict)

df['invert_data'] = df['Image'].map(invert_dict)

df['channel_data'] = df['Image'].map(channel_dict)

df['pixel_data'] = df['Image'].map(pixel_dict)
```

```
df['split_data'] = df['Image'].map(split_dict)
# Remove rows with empty detections
df = df[df['original_detection'].apply(lambda x: len(x) > 0)]
df.reset_index(drop=True, inplace=True)

def calculate_accuracy(original, detected):
    if not original and not detected:
        return 1.0
    elif not original or not detected:
        return 0.0

    original_set = set(original)
    detected_set = set(detected)
    intersection = original_set.intersection(detected_set)

    return len(intersection) / len(original_set)

accuracy_scores = {
    'gauss_data': [],
    'invert_data': [],
    'channel_data': [],
    'pixel_data': [],
    'split_data': []
}

for index, row in df.iterrows():
    original = row['original_detection']
```

```
    for col in accuracy_scores.keys():
        detected = row[col]
        accuracy = calculate_accuracy(original, detected)
        accuracy_scores[col].append(accuracy)

average_accuracies = {col: sum(scores) / len(scores) for col,
scores in accuracy_scores.items()}


print("Середня точність для кожного стовпця:")
for col, avg_accuracy in average_accuracies.items():
    print(f"{col}: {avg_accuracy:.2f}")
```

YOLOv8.ipynb

```
import os
import pandas as pd
from PIL import Image
#%%
from YOLO import detect_objects_yolo
from gaussian_noise import add_gaussian_noise
from Invert_colors import invert_colors
from remove_second import remove_every_second_pixel
from remove_channel import remove_channel
from split_image import split_reverse_merge
#%%
folder_path = 'dataset_image'
files = os.listdir(folder_path)
```

```
#%%

original_data = []

gauss_data = []

invert_data = []

pixel_data = []

channel_data = []

split_data = []

#%%

for file in files:

    if file.lower().endswith(('.jpg', '.png', '.jpeg')):  # Unified condition

        image_path = os.path.join(folder_path, file)

        image = Image.open(image_path)


        # Detect objects in the original image

        original_detection = detect_objects_yolo(image)

        original_data.append([file, original_detection])


        # Detect objects in the image with Gaussian noise

        noisy_image = add_gaussian_noise(image)

        gauss_detection = detect_objects_yolo(noisy_image)

        gauss_data.append([file, gauss_detection])


        # Detect objects in the inverted color image

        inverted_image = invert_colors(image)

        invert_detection = detect_objects_yolo(inverted_image)

        invert_data.append([file, invert_detection])
```

```
        # Detect objects in the image with every second pixel
deleted

        pixel_image = remove_every_second_pixel(image)

        pixel_data.append([file,
detect_objects_yolo(pixel_image)])



        # Detect objects in the removed blue and green channels
image

        channel_image = remove_channel(image)

        channel_data.append([file,
detect_objects_yolo(channel_image)])



        # Detect objects in the split image

        split_image = split_reverse_merge(image)

        split_data.append([file,
detect_objects_yolo(split_image)])

#%%

df = pd.DataFrame(original_data, columns=['Image',
'original_detection'])



# Create dictionaries for mapping

gauss_dict = {image: objects for image, objects in gauss_data}

invert_dict = {image: objects for image, objects in invert_data}

channel_dict = {image: objects for image, objects in channel_data}

pixel_dict = {image: objects for image, objects in pixel_data}

split_dict = {image: objects for image, objects in split_data}



# Add new columns to DataFrame
```

```
df['gauss_data'] = df['Image'].map(gauss_dict)
df['invert_data'] = df['Image'].map(invert_dict)
df['channel_data'] = df['Image'].map(channel_dict)
df['pixel_data'] = df['Image'].map(pixel_dict)
df['split_data'] = df['Image'].map(split_dict)

# Remove rows with empty detections
df = df[df['original_detection'].apply(lambda x: len(x) > 0)]
df.reset_index(drop=True, inplace=True)


def calculate_accuracy(original, detected):
    if not original and not detected:
        return 1.0  # Если оба пустые, считаем точность 100%
    elif not original or not detected:
        return 0.0  # Если одна из них пустая, а другая нет, точность 0%

    original_set = set(original)
    detected_set = set(detected)
    intersection = original_set.intersection(detected_set)

    return len(intersection) / len(original_set)

accuracy_scores = {
    'gauss_data': [],
    'invert_data': [],
```

```
    'channel_data': [],
    'pixel_data': [],
    'split_data': []
}

for index, row in df.iterrows():
    original = row['original_detection']
    for col in accuracy_scores.keys():
        detected = row[col]
        accuracy = calculate_accuracy(original, detected)
        accuracy_scores[col].append(accuracy)

average_accuracies = {col: sum(scores) / len(scores) for col,
scores in accuracy_scores.items()}

print("Середня точність для кожного стовпця:")
for col, avg_accuracy in average_accuracies.items():
    print(f"{col}: {avg_accuracy:.2f}")
```

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, Іскусних Дар'я Костянтинівна, студентка бакалаврської освітньої програми «Комп'ютерні науки», що нижче підписалась, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла чиїсь роботи повністю або частково як свої власні. Там, де я скористалася працею інших, я зробила відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута до академічної відповідальності.

_____ _____

(дата) (підпис)