МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Аналіз і синтез інформаційної системи видалення шуму з фотографій»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Петров С.О.**

Студента групи ІН.мз-91с — **Хапалова Я.М.**

СУМИ 2020

Сумський державний університет
(назва вузу)

Факультет *ЦЗДВФН* Кафедра *Комп'ютерних наук.*
Спеціальність *«Інформатика»*

Затверджую:
зав.кафедрою _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Хапалову Ярославу Михайловичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Аналіз і синтез інформаційної системи видалення шуму з фотографій.*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи)
_____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____
___

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз наукового доробку з проблеми обробки фотографій. 2) Технічне завдання по розробці інформаційної системи. 3) Проектування системи видалення шуму з фотографій. 4) Програмна реалізація.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) *Блок-схема роботи системи по видаленню шуму з фотографій.*

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	*Аналіз наукового доробку з проблеми обробки фотографій*	*01.09.2020 – 14.09.2020*	
2.	*Технічне завдання по розробці інформаційної системи*	*15.09.2020 – 30.09.2020*	
3.	*Проектування системи видалення шуму з фотографій*	*01.10.2020 – 14.10.2020*	
4.	*Програмна реалізації*	*15.10.2020 – 09.11.2020*	
5.	*Оформлення дипломної документації*	*10.11.2020 – 30.11.2020*	

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 82 стор., 32 рис., 1 табл., 2 додатки, 60 джерел.

Об’єкт дослідження — система видалення шуму з фотографій.

Мета роботи — розробка системи видалення шуму з фотографій з використанням нейронних мереж.

Методи дослідження — аналіз та синтез; експериментально-теоретичні та емпіричні методи.

Результати — було розроблено програмне забезпечення у вигляді Telegram-бота, що використовує наявну нейронну мережу для видалення решітчастої загорожі з фотографій. При цьому були використані такі інструменти як Python, Docker та Google Cloud Platform. Запропоноване рішення може бути у майбутньому модифіковане для видалення великої низки шумів (артефактів) з фотографій.

НЕЙРОННІ МЕРЕЖІ, КОМП’ЮТЕРНИЙ ЗІР, СИСТЕМА ВИДАЛЕННЯ ШУМУ З ФОТОГРАФІЙ, PYTHON, TELEGRAM-БОТ, ХМАРНІ СЕРВІСИ, DOCKER, GOOGLE CLOUD PLATFORM, GCP

ЗМІСТ

ВСТУП

З кожним роком збільшується кількість мобільних пристроїв у світі, функціонал яких передбачає фотозйомку. Це, в свою чергу, впливає на попит на різноманітні системи по обробці фотознімків. Не менш зростає і рівень застосування різноманітних комп'ютерних систем, що використовують фото та відео для своєї роботи (автоматичні радари перевищення швидкості тощо). Таким чином перед фахівцями постає необхідність навчання комп'ютерів працювати з зображенням та відео на більш високому рівні.

Зрештою, цікавою проблемою у цьому напрямку є проблема видалення шуму (артефактів тощо) з фотознімків. Вирішення цієї проблеми не тільки допоможе звичайному користувачеві у швидкій, простій та якісній обробці зображень, наприклад, виявленню та видаленню зайвих об'єктів на них, але й стане важливим доробком на шляху до покращення машинного зору та його застосуванню у різноманітних комп'ютерних системах та роботах.

Метою даної роботи є створення системи видалення шуму з фотографій. Для рішення цього завдання планується провести аналіз існуючих подібних рішень, а саме систем та алгоритмів, що в автоматичному режимі можуть виявити об'єкт (шум, артефакт тощо) на фотознімку та прибрати його.

Здебільшого, наразі існуючі системи не задовольняють важливу вимогу сучасного світу – зручність та доступність для кінцевого користувача. Таким чином, планується розробка такої системи, що буде надавати можливість використати найсучасніші алгоритми по обробці зображень у найзручніший спосіб.

Розробка такої системи вимагатиме аналізу наявних рішень та використанню найсучасніших інструментів для рішення цієї проблеми. Розроблена система стане в нагоді майбутнім дослідником та буде прикладом використання наукових здобутків у практичних цілях.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Проблема видалення шуму на цифрових фотографіях

У наш час людина навряд чи буде у ситуації, коли не може швидко зробити фотознімок, адже поширеність телефонів з вбудованими камера щороку збільшується [1, 2]. Цифрові фотознімки дозволяють зберегти важливі спогади або інформацію у дуже зручний та швидкий спосіб. Проте наявність шумів – перешкод, артефактів тощо – може зіпсувати важливі фотознімки. Візьмемо, наприклад, випадкових туристів, які потрапляють на фотографії під час відпусток. Кожний погодиться, що вони б виглядали набагато краще без них. Звичайно, можна спробувати зняти фотографію за ідеальних умов – виставити камеру, підібрати час зйомки тощо. Проте люди полюбляють мобільну фотографію саме завдяки тому, що вона дає змогу зафіксувати момент «тут і зараз», тобто у людини зазвичай не має часу на те, щоб налаштувати все ідеально і їй треба бути задоволеним тим, що вийшло. І саме тут на допомогу приходить пост-обробка зображень.

Для пост-обробки в основному використовують редактори зображень. Сучасні редактори зображень можна використовувати для видалення небажаних шумів (артефактів) із зображення. Ручна реконструкція зображення або алгоритми малювання в основному працюють у 3 етапи – ідентифікація небажаного об'єкта, видалення небажаного об'єкта, заповнення створеного таким чином порожнього простору новими даними, які відповідають решті зображення [3]. Перший крок виявлення небажаного об'єкта або розбіжності на зображенні часто проводиться вручну. Область, яку слід розглянути для виправлення, виділяється перед виконанням будь-яких робіт із зображенням. Другий крок передбачає очищення пікселів, які лежать у цій області. Третім і найважливішим кроком є заповнення порожнечі, що було створено на другому кроці, новими даними [3].

Можна зробити висновок, що зручна система для видалення шумів з фотознімків повинна зводити до абсолютного мінімуму ручну роботу з боку користувача та надавати передові (state-of-the-art) результати. В свою чергу, це робить цю систему доволі обмеженою стосовно засобів її практичного використання, адже вимагає певної спеціалізації цієї системи щодо шумів, для яких вона призначена. Таким чином, об’єктом даної роботи було обрану таку систему, яка могла би прибрати шум з фотографії, а програмного реалізовано її на прикладі прибирання сітчастої огорожі з фотознімку як зображено на рис. 1.1.

Рисунок 1.1 — Приклад цільового зображення

Проблемою такого характеру вже переймалися численні дослідники в останні роки, адже з поширенням цифрових камер (у тому числі вбудованих в мобільні телефони), кожного року невпинно зростає необхідність зручних інструментів по видаленню небажаних об’єктів на фотографіях. Багато хто з дослідників вже запропонував свої методи по видаленню шумів, таких як відблиски у склі (вітрин, машин тощо) [4–13], краплі на склі [4, 6, 14], огорожа з сітки [4, 6, 15–17]. У наступних підрозділах були розглянуті ці методи відповідно до двох існуючих принципово різних підходів: обробки зображення

за допомогою одного зображення (single image) та декількох зображень (кадрів) (multi-frame approach) [4].

1.2 Видалення шуму за допомогою однокадрового підходу

Зробити фотографії через скляні поверхні (наприклад, вікна, вітрини тощо) або оклюзійні елементи (наприклад, огорожі) є складним завданням, оскільки захоплені зображення неминуче містять як сцени, що мають значення для користувача, так і шуми (перешкоди), спричинені відображеннями або оклюзіями [4]. Вирішенням цієї проблеми вже не один рік зацікавлені дослідники у всьому світі, адже воно є важливим не тільки для побутових потреб, але й, що найважливіше, для поліпшення машинного зору, що активно використовується в розробках роботів.

Останні зусилля дослідників були спрямовані на автоматичне видалення небажаних відображень або оклюзій, використовуючи лише один кадр зображення, тобто одне фото [10–13, 15, 16, 18, 19]. Деякі з цих методів використовують «примарні» відображення (ghosting cues) для відокремлення небажаного відображення від корисного фото [19]. Інші застосовують підходи, засновані на навчанні, щоб створити близькі до натурального зображення (natural image prior) та за допомогою них знаходять відображення або оклюзію [10–13]. Розглянемо більш детально роботи дослідників.

Наприклад, дослідники з Китаю застосували архітектуру глибоких нейронних мереж для видалення відображення на одному зображенні [10]. У їх роботі пропонується використання глибокої структури нейронних мереж, яка використовує крайову інформацію (edge information) для вирішення репрезентативних завдань машинного зору низького рівня, таких як розділення шарів та фільтрація зображень. На відміну від більшості інших стратегій глибокого навчання, що застосовуються в цьому для вирішення подібних завдань, цей підхід вирішує ці складні проблеми, оцінюючи краї та реконструюючи зображення, використовуючи лише каскадні згорткові шари,

розташовані таким чином, що не потрібні ніякі ручні та специфічні для обробки зображень інструменти. Використовуючи припущення про м’якість відбиття та новий синтетичний метод генерації даних, який діє як тип слабкого нагляду (weak supervision), їх мережа здатна вирішити набагато складніші випадки відображення, яке потрібно прибрати [10]. Нажаль, запропонований метод важко застосувати до вирішення більш широких проблем оклюзій.

Ще одними дослідниками, які застосували нейронні мережі до вирішення проблеми відображення, є дослідники з університету Берну (Швейцарія) [12]. Вони представили новий підхід глибокого навчання для автоматичного відокремлення двох накладених один на одного шарів – фону (цільового зображення) та шару відображення. Була проаналізована проблема розділення шарів та використані отримані дані для розробки синтетичної моделі генерації даних та для проектування нейронної мережі з широким полем сприйняття. Дослідники навчали нейронну мережу за допомогою синтетичних даних, отриманих від цієї моделі, і показали, що вона добре опрацьовує зображення і перевершує попередні роботи. Їх підхід поєднує здатність нейронних мереж будувати попередні зображення на великих областях зображення за допомогою моделі зображення, яка враховує невизначеність яскравості та насиченість зображення [12]. Отже, хоча результати дослідників вражають, проте їх підхід не має застосування за межами проблеми видалення відображень.

Цікавою є робота дослідників з Індії [16]. У своїй роботі вони запропонували алгоритм прибирання огорожі для зображень динамічних сцен із використанням методу оптичного потоку, орієнтованого на оклюзію. Вони розділили проблему прибирання огорожі (de-fencing) на зображенні на завдання автоматизованої сегментації огорожі з одного зображення, оцінки руху при відомих оклюзіях та злиття даних з декількох кадрів знятого відео. Зокрема, науковці використали попередньо навчену згорткову нейронну мережу, щоб сегментувати пікселі огорожі з одного зображення. Знання просторового розташування огорожі використовується для подальшої оцінки оптичного

потоку в закритих кадрах відео для кінцевого етапу синтезу даних. Для своєї роботи вони використали алгоритм FISTA (fast iterative shrinkage-thresholding algorithm) [16]. Хоча експериментальні результати показують ефективність запропонованого алгоритму, його важко застосувати до фото.

Більш цікавою є робота дослідників із США [18]. Вони представили новий метод прибирання огорожі на зображеннях, що був придатний для звичайної фотозйомки, тобто користувач міг досягти гарних результатів, використовуючи будь-яку камеру та її налаштування. По-перше, виявлення решіток зі світлопрозорими текстурами виконується в ітеративному процесі з використанням онлайн-навчання і класифікації за проміжними результатами, що сприяє подальшому виявленню оклюзій. Потім здійснюється сегментація переднього плану з використанням накопиченої статистики з усіх точок решітки. Далі виконується багатоканальне фарбування щоб заповнити закриті області інформацією із видимих частин зображення. Для областей, закритих у всіх точках зображення, використовується нове забарвлення з симетрією, яка поєднує в собі традиційний текстурний синтез зі збільшеним пулом патчів-кандидатів, знайдених шляхом моделювання двосторонніх патернів симетрії на початковому зображенні [18]. Результати показують ефективність запропонованого методу. Хоча розробка дослідників справляє враження, на вихідних фотографіях все ще присутня значна кількість артефактів (шумів).

Робота фахівців з Пекінського інституту технологій пропонує інший підхід [11]. У своїй статті вони подолали проблеми видалення оклюзії, використовуючи цілеспрямоване вдосконалення нейронної мережі та нове використання зміщених (misaligned) даних. Вони розширили базову мережеву архітектуру, інтегруючи модулі контекстного кодування, здатні використовувати контекстні підказки високого рівня для зниження детермінованості в областях, що містять сильні відображення. Також була введена функція втрат з урахуванням змін, яка полегшує використання зміщених реальних даних навчання [11]. Результати експериментів в сукупності

показують, що метод перевершує сучасний рівень зі зміщеними даними і що при використанні додаткових даних можливі значні поліпшення.

Подібних результатів досягли науковці з університету Аделаїди в Австралії [13]. Ними був запропонований підхід, який передбачає навчання глибокої нейронної мережі, яка безпосередньо зіставляє зображення, що містить відображення з фоновим (цільовим) зображенням (тобто із зображенням без відображення). Стверджується, що для того, щоб видалити відображення на високому рівні, потрібно визначити ступінь відображення і використати його для визначення фонового зображення. Була запропонована каскадна глибока нейронна мережа, яка визначає як фонове зображення, так і відображення [13]. Це значно покращує видалення відображень. У каскадної глибокої нейронної мережі використовується визначене фонове зображення для визначення відображення, а потім визначене відображення для визначення фонового зображення. Хоча результати роботи зацікавлюють, проте їх важко застосувати до інших типів оклюзій.

Останньою розглянутою роботою у цьому підрозділі стала стаття фахівців з Массачусетського технологічного інституту (США) [19]. У цій роботі науковці використовують примарні сліди (ghosting cues), які виявляють асиметрію між шарами зображення (решітка та фонове (цільове) зображення). Ці сліди виникають через зміщення подвійних відображень відображеної сцени на поверхні скла. У вікнах з подвійним склом кожна панель відображає зміщені і слабкіші версії об'єктів на стороні скла з камерою. У вікнах з одним вікном примарні сліди виникають в результаті зсунутих відображень на двох поверхнях скла. Незважаючи на те, що іноді примарні сліди ледь вловимі людиною, науковці наполягають, що все ж можуть використовувати цей сигнал для відокремлення шарів. У своїй роботі вони моделюють примарне відображення за допомогою ядра подвійної імпульсної згортки й автоматично оцінюють просторове розділення і відносне загасання компонентів примарного відображення [19]. Для поділу шарів пропонується алгоритм, який

використовує Гаусову модель змішування для регуляризації зображень. Пропонований метод є автоматичним і вимагає тільки одного вхідного зображення. Він показав свою дієвість як на синтетичних, так і реальних зображеннях.

Хоча були розглянуті методи з гарними результатами, відділення чистого фону від відображення / оклюзії є принципово некоректно поставленим завданням і часто вимагає семантичного розуміння сцени на високому рівні для гарного виконання. Зокрема, ефективність методів, заснованих на навчанні нейронних мереж і використанні лише одного зображення, значно знижується при застосуванні алгоритму до реальних зображень [4]. Для вирішення цієї проблем були запропоновані багатокадрові підходи до видалення відображення / оклюзії, які були розглянуті в наступному підрозділі 1.3.

1.3 Видалення шуму за допомогою багатокадрового підходу

Основна ідея багатокадрового підходу полягає в тому, щоб використовувати той факт, що фонова сцена й елементи, які її приховують (відображення, або решітка огорожі), розташовані на різних глибинах стосовно камери (наприклад, віртуальна глибина відображення у вікні). Отже, зйомка декількох зображень за допомогою камери, що ледве рухається, виявляє відмінності в русі між цими двома шарами [4].

Наприклад, у роботі науковців з Пекінського університету (Китай) розглядається сліпе відділення шарів-джерел від накладених на них змішувань з невідомими рухами і невідомими коефіцієнтами змішування шарів у кожній композиції [5]. Попередні підходи сліпого поділу для таких завдань припускають, що рухи є однаковими, а отже, обмежені для застосування в реальному світі [5]. Дослідники презентують алгоритм розрідженого сліпого поділу для оцінки як параметризованих рухів, так і коефіцієнтів змішування. Також представлений новий підхід до реконструкції для відновлення всіх шарів з використанням не тільки моделі змішування, а й статистичних властивостей

натуральних зображень. Весь метод може обробляти більш загальні рухи, включаючи масштабування, обертання й інші перетворення. Окрім цього, кількість шарів визначається автоматично, і всі шари можуть бути відновлені навіть в недостатньо конкретному випадку, коли кількість змішувань менше, ніж кількість шарів. Ефективність цієї технології показана в експериментах на двох змодельованих сумішах з чотирьох шарів та на реальних фотографіях.

Схожого результату досягають інші дослідники з Китаю, які запропонували надійний метод відділення двох шарів від множинних зображень, який використовує кореляцію переданого шару по множинним зображенням, а також розрідженість і незалежність градієнтних полів двох шарів [9]. Новий алгоритм, заснований на розширеному мультиплікаторі Лагранжу, призначений для ефективного і дієвого вирішення проблеми декомпозиції. Результати експериментів як на змодельованих, так і на реальних даних демонструють перевагу запропонованого методу над сучасними прикладами з точки зору точності і простоти.

Проте ще кращого результату у своїй роботі досягли дослідники із Сінгапуру [7]. Їх підхід використовує тонкі зміни відображення відносно фону в невеликому наборі зображень, зроблених з трохи різними точками огляду. Ключовим моментом в цій ідеї є використання SIFT-потоку для вирівнювання зображень таким чином, щоб можна було проводити порівняння по пікселям по всьому вхідному набору. Масштабонезалежне перетворення ознак (SIFT - scale-invariant feature transform) є алгоритмом в комп'ютерному зорі для виявлення і опису локальних ознак на зображеннях [20]. Передбачається, що градієнти з варіаціями по всьому набору зображень належать відображенню у склі, в той час як постійні градієнти належать цільовій фонової сцені. За правильного маркування градієнтів, що належать відображенню або фону, фонова сцена може бути відділена від інтерференції відображення. На відміну від попередніх підходів, що використовують рух, цей підхід не робить ніяких припущень щодо геометрії фону або відображених сцен і не вимагає, щоб відображення було

статичним. Це робить підхід практичним для використання в сценаріях випадкової візуалізації [7].

Масштабонезалежне перетворення ознак (SIFT) є популярним серед дослідників цієї проблеми. Так, науковці з Массачусетського технологічного інституту (США) та компанії Adobe запропонували використати SIFT потік за аналогією з оптичним потоком, коли зображення вирівнюється по тимчасовому сусіднього кадру [21]. Для вхідного знімка використовується перетин гістограми на репрезентації «мішку візуальних слів» (bag-of-visual-words) для пошуку набору найближчих сусідів в базі даних. У цьому випадку алгоритм SIFT-потоку полягає у збігу щільно дискретних особливостей SIFT між двома зображеннями зі збереженням просторових розривів. Використання SIFT-функцій дозволяє здійснювати точний збіг по різним сценам / відображенню об'єкта, а просторова модель зі збереженням розривів дозволяє зіставляти об'єкти, розташовані в різних частинах сцени [21].

Дослідники з компанії Microsoft, на противагу згаданим методам, використали інший алгоритм, з допомогою якого вони відокремили різні шари зображень [22]. У своїй роботі вони розробили оптимальний підхід до відновлення шарів зображень і пов'язаних з ним рухів з довільної кількості композитних зображень. Дослідники розробили дві різні методики оцінки зображень компонентного шару з урахуванням відомих оцінок руху. Перший підхід використовує обмежені найменші квадрати для відновлення шарів зображення. Другий підхід ітеративно уточнює нижню і верхню межу шарів зображення, використовуючи дві нові композиційні операції, а саме мінімальний і максимальний склад вирівняних зображень. Вчені комбінують ці методи вилучення шарів з домінантним оцінювачем руху і наступним етапом уточнюють цей рух. Таким чином, їм вдалося створити систему, що автоматично визначає шари зображення (шар оклюзії та шар цільового зображення) та відокремлює їх один від одного.

Особливої уваги заслуговує алгоритм розроблений фахівцями з Массачусетського технологічного інституту та компанії Google (США) [6]. Вони представили уніфікований обчислювальний підхід для зйомки фотографій через оклюзійні елементи, такі як вікна й огорожі. Замість того, щоб робити один знімок, вони пропонують користувачу робити коротку послідовність зображень, злегка рухаючи камеру. Відмінності, які існують у відносному положенні фону і оклюзійних елементів дозволяють відокремити їх на основі руху та відновити бажану цільову сцену фото. Вони показали блискучі результати як для експериментальних, так і живих фото.

Проте, даний підхід все ж мав свої вади, які були подолані у роботі групи дослідників з Тайванського університету, Google та MediaTek [4]. Вони з'ясували, що раніше згаданий алгоритм вимагає вимогливого до ресурсів комп'ютера процесу оптимізації, а також ґрунтується на строгих припущеннях про сталість яскравості або точній оцінці руху під час зйомки [4].

Враховуючи, що дослідники отримали найкращі результати у проблемі видалення оклюзій на фотографіях, а також надали детальний опис свого підходу та поділилися реалізацією на порталі GitHub [23], саме їх реалізацію було взято за основу розробленої системи видалення шуму на фотографіях, а конкретно видалення решітчастих огорож. Детальний огляд використаного підходу наданий у розділі 2.

2 АНАЛІЗ ЗАСТОСОВАНОГО РІШЕННЯ

2.1 Загальний огляд методу

У роботі науковців з Тайванського університету, Google та MediaTek було запропоновано алгоритм видалення обструкцій за допомогою багато-кадрового (multi-frame) підходу, який об’єднує в собі переваги як оптимізаційних, так і навчальних методів [6], запропонований алгоритм чергує кроки оцінки щільності руху (dense motion) з кроками реконструкції шару з обструкцією та цільовим зображенням за допомогою підходу «від не точного до точного». Під підходом «від не точного до точного» (coarse-to-fine) в контексті візуального розпізнавання зазвичай розуміється застосування методу (або моделі машинного навчання) до версії зображення з низькою роздільною здатністю, та поступове збільшення здатності і застосування попередніх результатів до «більш точного» зображення [4].

Моделювання щільності руху дозволяє поступово відновлювати деталізований вміст у відповідних шарах зображення. Замість того, щоб покладатися на написані методи по відокремленню шарів зображення, використовується метод машинного навчання для злиття викривлених потоком зображень, щоби рахувати потенційні порушення постійної яскравості і помилки в оцінці потоку. Навчання мережі злиття відбувається з використанням синтетично згенерованого набору даних, який добре себе показує на реальних послідовностях зображень, які раніше не використовувалися. Окрім того, був запропонований процес онлайн-оптимізації для подальшого поліпшення візуальної якості реальних зображень. На прикладі широкого спектру складних послідовностей дослідники доводять, що запропонований метод вигідно відрізняється від існуючих алгоритмів. Цей фреймворк побудований на заснованому на оптимізації формулюванні [6], але відрізняється тим, що ця модель орієнтована виключно на дані і не спирається на класичні допущення, такі як сталість яскравості [6], точні поля потоку [7] або пласка поверхня [9] сцени зображення. Коли ці допущення порушуються

(наприклад, розмиття руху, неточний потік), класичні підходи можуть не відтворити достатньо точно передній (оклюзійний) та фоновий (цільовий) шари. З іншого боку, цей метод уникає описаних вище помилок, адже повністю спирається лише на дані, а також вчиться на різноманітних тренінгових даних. Використовуючи глибокі згорткові нейронні мережі (CNN) для оцінки оптичного потоку і реконструкції фонового зображення, цей метод є прийнятним для видалення не лише якось певного типу оклюзій (відображення у склі, краплі води на склі, решітчаста огорожа тощо), а загалом може бути використаний для усунення з зображень будь-яких перешкод [4].

2.2 Теоретичне обґрунтування алгоритму

З огляду на послідовність за формулою (2.1) з кадрів T, мета алгоритму полягає в тому, щоб розкласти кожен кадр I_k на два шари, один для цільового зображення (фону), а інший для перешкоди на зображені, яку потрібно прибрати.

$$\{I_t\}_t^T = 1 \qquad (2.1)$$

Декомпозиція послідовності зображень на фонові та обструкційні шари складна тим, що включає в себе рішення двох тісно пов'язаних проблем: декомпозицію оптичного потоку і саме реконструкцію шарів [4]. Без точної декомпозиції потоку шари не можуть бути достовірно реконструйовані через неузгодженість, викликану не достатньо точним визначенням руху. З іншого боку, без достовірно реконструйованих фонового та оклюзійного шару неможливо точно оцінити оптичний потік через змішаний зміст зображення. У зв'язку з природою цієї проблеми «курки та яйця» - незрозуміло з чого почати, так як у нас відсутня інформація як про потоки, так і про шари [4].

Задля вирішення цієї проблеми, пропонується використати глибокі згорткові нейронні мережі (CNN). У своїй основі метод має три модулі, які також зображенні на рис. 2.1 [4]:

а) початкова декомпозиція потоку;

б) реконструкція фонового і обструкційного шару;

в) коректування оптичного потоку.

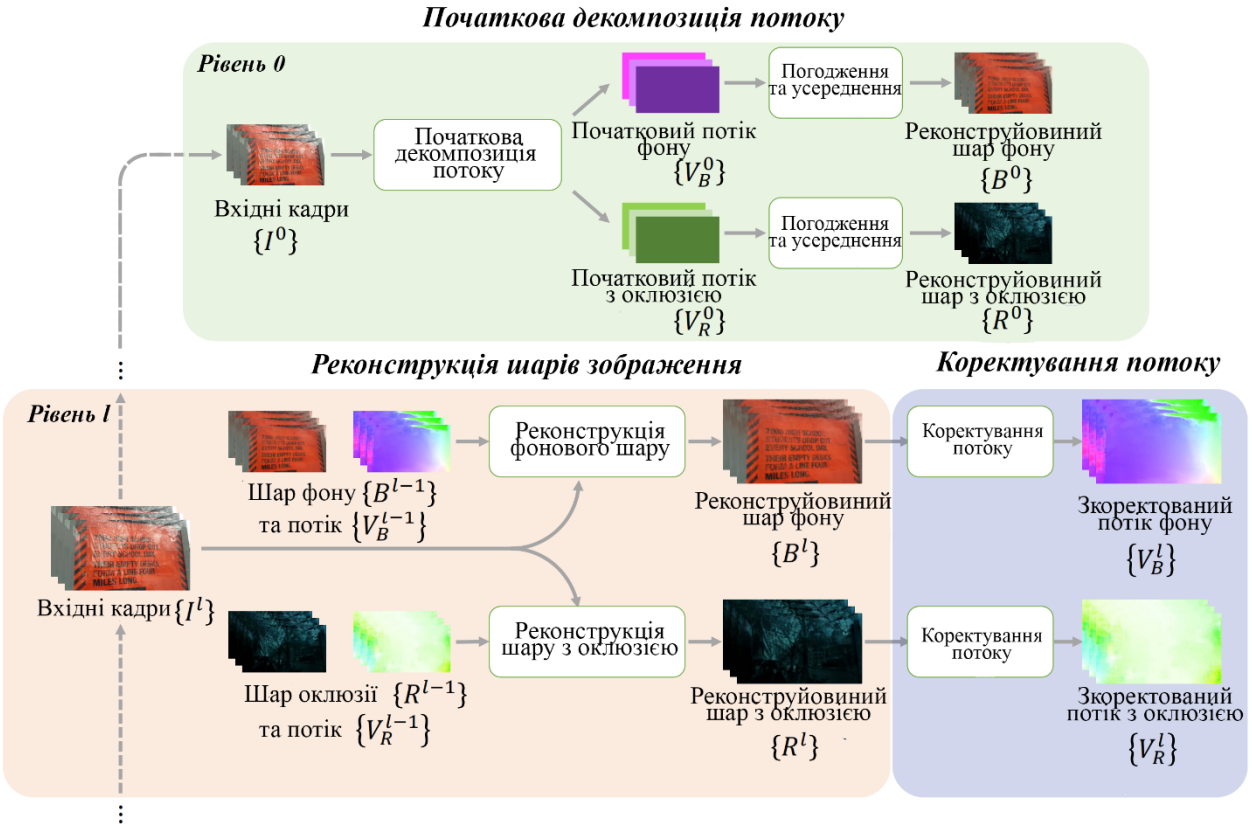

Рисунок 2.1 — Огляд алгоритму

Як показано на рис. 2.1, відбувається реконструкція шарів зображення підходом «від не точного до точного». На найбільш не точному рівні 0 оцінюється однорідність полів потоку як для фонового шару, так і для шару з оклюзією, а потім відбувається відновлення шарів завдяки усередненню узгоджених кадрів. На рівні l застосовується модуль фонового і обструкційного шару для реконструкції обох шарів, а також використовується PWC-нейронна мережа для прогнозування скоректованих полів потоку для обох шарів [4]. PWC (Pyramid, Warping, Cost Network) це згорткова нейронна мережа розроблена фахівцями компанії NVIDIA відповідно до таких принципів: пірамідальна обробка (pyramidal processing), деформація (warping) та використання обсягу витрат (use of a cost volume) [24]. Таким чином запропонований фреймворк поступово відновлює шари і поля потоку зображення до найточнішого рівня.

Метод бере кадри T в якості вхідних і розкладає ключовий кадр I_k одночасно на фоновий шар B_k і шар з оклюзією R_k. Далі відбувається реконструкція вихідного зображення за допомогою підходу «від не точного до точного» в ієрархії l-рівня. Спочатку оцінюються потоки, починаючі від не точного рівні у модулі початкової декомпозиції потоку. Потім послідовно відновлюється фоновий (цільовий) та оклюзійні шари і уточнюються оптичні потоки до останнього рівня як показано на малюнку 2.1. Цей уніфікований фреймворк може бути застосований до багатьох проблем декомпозиції шарів, таких як прибирання відображення у склі, решітчастої огорожі, крапель на склі тощо [4]. Без втрати спільності, приклад застосування алгоритму описується надалі за допомогою завдання прибирання відображення у склі.

Говорячи про декомпозицію потоку, передбачається, що потік визначається як для фонового шару, так і для шару відображення на не точному рівні (l = 0), що є суттєвою відправною точкою застосованого алгоритму [4]. Замість того, щоб оцінювати щільні поля потоку, пропонується вивчити уніфікований вектор для кожного шару. Застосована мережа декомпозиції потоків складається з двох підмодулів: 1) екстрактор деталей кадру; 2) оцінювач потоку шару. Екстрактор спочатку генерує карти деталей для всіх вхідних кадрів, а потім будується обсяг витрат (CV) між двома кадрами за допомогою кореляційного шару. Це зображено на формулі (2.2).

$$CV_{jk}(x_1, x_2) = c_j(x_1)\, c_k(x_2) \qquad (2.2)$$

У ній c_j і c_k витягнуті особливості кадру j і k відповідно, а x індекс пікселів. Так як просторовий дозвіл на цьому рівні досить малий, то діапазон пошуку кореляційного шару встановлюється лише на 4 пікселя [4]. Обсяг витрати CV потім узгоджується з ознакою c_j і подається в оцінювач потоку шару.

Для створення двох глобальних векторів руху в оцінювача потоку шару використовується глобальний середній пул (pooling) і повністю пов'язані шари.

Нарешті, глобальні вектори руху об’єднуються в два рівномірних поля потоку для фонового шару і для шару з оклюзією.

Наступним кроком є реконструкція цільового шару та шару з оклюзією за допомогою модуля реконструкції шарів. Він призначений для реконструкції чистого фонового зображення (цільового шару) B_k і зображення з оклюзією B_k. Незважаючи на те, що два завдання реконструкції фону і оклюзії схожі за своїми цілями, характеристики фонового і шарів з оклюзією абсолютно різні [4]. Наприклад, фонові шари часто є більш домінуючими за зовнішнім виглядом, але в деяких кадрах вони можуть бути закриті. З іншого боку, шари з оклюзіями часто розмиті і темніші. Отже, відбувається навчання двох незалежних мереж для реконструкції обох шарів. Ці дві мережі мають однакову архітектуру, але не мають спільних мережевих параметрів. Далі описується мережа для реконструкції фонового шару, тому що інша мережа реконструює шар з оклюзією точно таким способом.

Реконструкція фонового шару відбувається уже раніше згаданим підходом «від не точного до точного» (coarse-to-fine). На самому не точному рівні ($l = 0$) для вирівнювання сусідніх кадрів використовується поле потоку, обчислене з модуля початкової декомпозиції потоку. Потім обчислюється середнє значення всіх зареєстрованих фонових кадрів як прогнозоване фонове зображення за допомогою формули (2.3).

$$B_k^0 = \frac{1}{T}\sum_{j=1}^{T} W\left(I_j^0, V_{B,j\to k}^0\right) \qquad (2.3)$$

У ній I_j^0 - кадр j занижений до рівня 0, а W () - операція білінійного семплірування. Мережа приймає на вхід від попереднього рівня реконструйоване фонове зображення, зображення з оклюзією, оптичні потоки фону, а також вхідні кадри від поточного рівня.

На поточному рівні модель намагається реконструювати фонове зображення ключового кадру. Спочатку підвищується значення полів фонового, а всі вхідні кадри вирівнюються по ключовому кадру [4]. Так як деякі пікселі можуть стати недійсними через оклюзії або деформації зображення,

обчислюється карта відмінностей, яка додається до мережі для зменшення кількості можливих артефактів.

Далі відбувається конкатенація кадрів, карт відмінностей, а також шару з фоном збільшеної роздільної здатності та шару з оклюзією з попереднього рівня. Це все використовується як вхідні дані для роботи мережі по реконструкції фонового зображення. Сама реконструкція відбувається за допомогою залишкового навчання [4]. Треба зазначити, що шар з оклюзією також використовується в реконструкції фонового шару, який з'єднує мережі реконструкції фонового шару та шару з оклюзією разом для спільного навчання.

Після реконструкції всіх фонових зображень відбувається коректування оптичних потоків фонового зображення. Для цього використовується попередньо підготовлена PWC (Pyramid, Warping, Cost Network) мережа для оцінки полів потоку між парними фоновими зображеннями [24].

Важливим етапом роботи алгоритму є навчання мережі. Для підвищення стабільності навчання застосовується двоступенева процедура навчання. На першому етапі відбувається навчання мережі початкової декомпозиції потоку [4]. Потім мережа початкової декомпозиції потоку заморожується і відбувається навчання мережі реконструкції шарів з втратами при реконструкції зображень і градієнтними втратами. Градієнтні втрати спонукають мережу відновлювати більше деталей для подальшого поліпшення візуальної якості [4].

За результатами тестування, описаний метод перевершив досягнення попередників та показав найкращі результати [4].

Програмна реалізація цього методу була підготовлена науковцями та розміщена на офіційній сторінці в GitHub [23]. У даній роботі був використаний розроблений алгоритм видалення шуму за допомогою нейронної мережі та наявна програмна реалізація (надалі модуль нейронної мережі) для створення системи видалення шуму (артефактів) на фотографіях. Було обрано решітчасту

загорожу як об’єкт оклюзії, з яким буде працювати дана система. Дана система повинна стати зручним способом використання наведеного у цьому підрозділі методу.

2.3 Опис розробленої системи та застосування алгоритму в ній

Без сумніву, найважливішим критерієм успіху будь-якого програмного рішення в наш час є вдала реалізація з точки зору кінцевого користувача. Незважаючи на сучасність розробленого алгоритму по видаленню оклюзій з фотографій, його програмна реалізація на GitHub не є повноцінною системою. Багато хто з користувачів навіть не зміг скористуватися розміщеним першокодом для запуску цієї нейронної мережі [23]. Завданням даної роботи є налаштування нейронної мережі для роботи та створення на її основі повноцінної системи, яка б дозволила якомога більшій кількості користувачів випробувати вражаючі можливості алгоритму у найпростіший спосіб. Отже, архітектурно ця система буде складатися з двох модулів, що зображені на рис. 2.2.

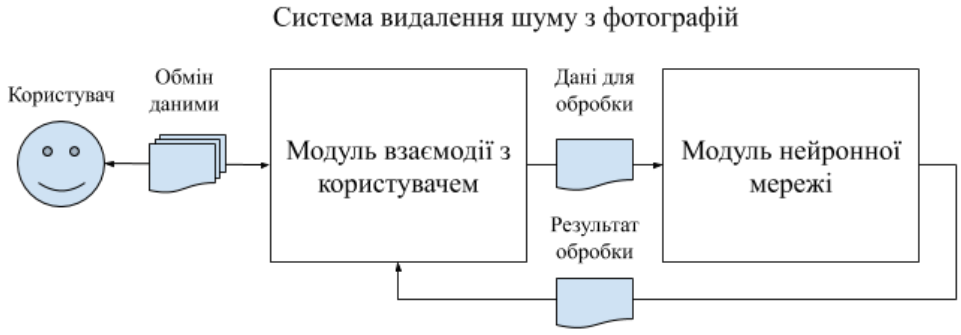

Рисунок 2.2 — Блок-схема системи видалення шуму з фотографій

Відповідно до вище наведеної схеми, було визначено критерії вибору підходу до реалізації системи загалом та модуля взаємодії з користувачем зокрема.

Цими критеріями є:

а) можливість зручної інтеграцій модуля нейронної мережі з модулем взаємодії з користувачем;

б) проста та зручність користування модулем взаємодії з користувачем (модель сервер-клієнт);

в) наявні потужні інструменти розробки (API тощо);

г) доступність для широкого загалу.

За результатами дослідження, було визначено, що Telegram-бот є найбільш вдалим способом реалізації модуля взаємодії з користувачем (програмного інтерфейсу).

Telegram-бот це аккаунт у популярному додатку для обміну повідомленнями Telegram, яким керує програмне забезпечення, а не людина [25]. Завдяки відкритому API, кожен може створити своє програмне забезпечення та використати Telegram-бота майже для будь-чого, що може роботи будь-яке інше програмне забезпечення. Таким чином, Telegram-бот є дуже зручним інтерфейсом користувача для програмного забезпечення, яке керує цим ботом. Отже, переваги такого способу реалізації модуля взаємодії з користувачем є суттєвими:

а) величезна база користувачів: за даними Data Magic в Україні не менше ніж 4,5 млн користувачів, що робить бота доступним численній аудиторії [26];

б) користувачам не треба встановлювати стороннє програмне забезпечення, або використовувати браузер – достатньо почати час з ботом;

в) зручність та швидкість: бот працює на сервері, що впливає на швидкість, й до того ж не приділяє ніяких вимог до клієнтського обладнання;

г) Telegram надає зручний API для розробників, що постійно оновлюється й розширює функціонал ботів;

д) є велика кількість готових бібліотек по роботі з Telegram API для всіх популярних мов програмування.

Враховуючи, що нейронна мережа написана мовою Python, для написання Telegram-боту теж було обрано саме її. Це дасть можливість нативно інтегрувати модуль нейронної мережі з модулем Telegram-боту.

Програмна реалізація системи видалення шуму з фотографій (далі продукт) відбувалася за ітеративно-інкрементною моделлю. З одного боку, ця модель передбачає, що робота розбивається на відносно малі частини, планується розробка кожної з них та їх інтеграція по завершенню у готовий продукт [27]. З іншого – розробка відбувається циклами, що повторюються. Кожний цикл включає в себе аналіз, дизайн, написання та/або правки коду, тестування. Схематичне зображення використаної моделі зображено на рис. 2.3 [28]. Застосовуючи цей підхід, ціллю було зменшити кількість часу на переробку вже готової системи, що неодмінно мало б місце за каскадної моделі розробки. Тим паче, концепція ітеративно-інкрементної розробки пройшла перевірку часом та є рекомендованою відомими науковцями у галузі комп'ютерних наук [29].

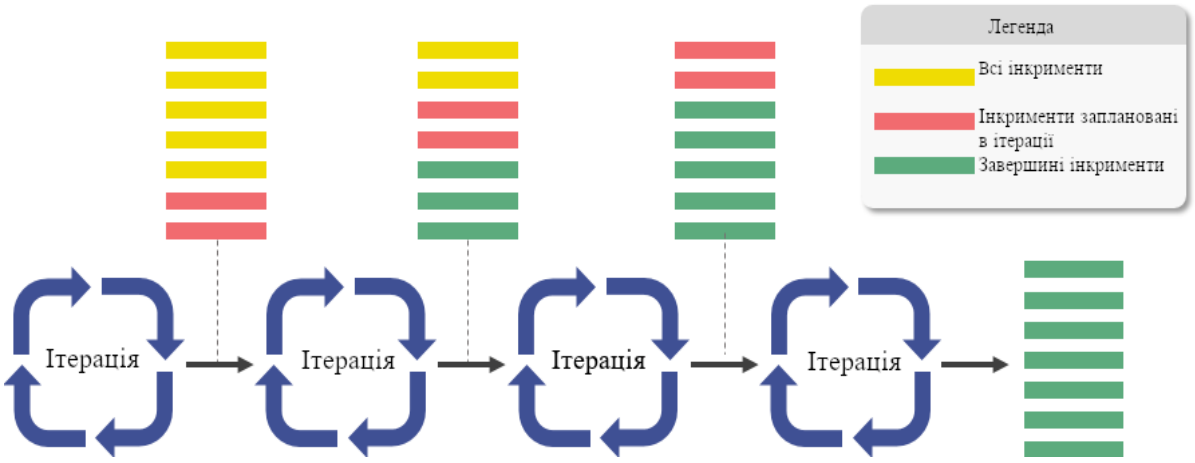

Рисунок 2.3 — Схема ітеративно-інкрементної моделі

Першим кроком для початку роботи (ітерації №1) було визначення високорівневих вимог до бажаної системи. Були окреслені наступні вимоги:

а) Telegram-бот повинен отримувати фотографії від користувача;

б) Telegram-бот повинен конвертувати та перейменовувати фотографії;

в) Telegram-бот повинен запускати нейронну мережу та опрацьовувати ці фотографії;
г) Telegram-бот повинен надсилати готовий результат користувачеві;
д) Telegram-бот повинен бути запущеним на сервері для постійної роботи.

Відповідно до вимог вище, були визначені кроки реалізації (ітерації) та інкременти (маленькі працюючі частини задуманого продукту), що мають згодом стати повноцінним продуктом. Представлені інкременти у таблиці 2.1 є фінальними, тобто враховуючи ітеративну модель розробки, вони змінювалися залежно від нової інформації або досвіду, що були отримані по завершенню попередньої ітерації. Розробку системи було розглянуто у розділі 3 максимально близько до ітерацій нижче, адже це дало змогу у повному обсязі описати програмну реалізацію.

Таблиця 2.1 Опис інкрементно-ітеративного підходу

Ітерація №	Назва інкременту	Опис
1	Працюючий Google Colab	Запуск та тест нейронної мережі у сервісі Google Colab
	Локально працююча нейронна мережа	Налаштування та тест нейронної мережі на локальному комп’ютері
2	Docker контейнер з мережею	Знайомство з Docker. Створення Docker контейнера з мережею.
3	Простий Telegram бот	Створення, запуск та тест простого Telegram бота
	Docker контейнер з ботом запущено у хмарному сервісі	Знайомство з хмарними сервісами. Запуск бота у Docker контейнері на хмарному сервісі Google Cloud

4	Telegram bot по обробці фотографій	Створення повноцінно функціонуючого бота для отримання, обробки та надсиланню фотографій користувачеві.
	MVP	Об’єднання бота з нейронною мережею та створення MVP (мінімально життєздатного продукту). Тестування.
	Фінальна версія	Рефакторінг коду та розгортання бота у хмарному сервісі Google Cloud.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Огляд програмної реалізації застосованої нейронної мережі

Програмна реалізація модуля нейронної мережі була розроблена фахівцями з Тайванського університету, компаній Google та MediaTek та розміщена на порталі спільної розробки програмного забезпечення GitHub [23]. Це дає змогу будь-кому використати надбання науковців для своїх систем або досліджень та дає змогу взяти участь у поліпшенні їх алгоритму тощо. Лістинг коду нейронної мережі можна переглянути безпосередньо на сторінці GitHub https://github.com/alex04072000/ObstructionRemoval .

Для реалізації своїх ідей дослідники обрали мову програмування Python 3.7. Python є найпопулярнішою мовою за останні декілька років загалом, та 100% найпопулярнішою мовою для машинного навчання [31]. Python став улюбленим вибором фахівців для аналізу даних, машинного навчання завдяки своїй великій бібліотечної системі, що дає змогу ентузіастам машинного навчання з легкістю отримувати обробляти та використовувати великі об’єми даних. Також Python завоював серце інженерів машинного навчання, завдяки своїй платформній незалежності, простоті і гарній читабельності коду [31].

Всі ці переваги були застосовані для реалізації нейронної мережі. А саме, були використані бібліотеки pandas, tqdm, matplotlib, scikit-image, pylint, pillow, opencv-python та tensorflow. Ці бібліотеки будуть детальніше описані у підрозділі 3.2.3 під час опису запуску нейронної мережі.

Без сумніву, найважливішим елементом програмної реалізації є саме бібліотека TensorFlow. Ця програмна бібліотека була розроблена компанією Google для своїх потребу у створенні нейронних мереж та задач машинного навчання. Згодом, бібліотека була відкрита для широкого загалу та будь-хто міг використати її для своїх проектів або дослідів [46]. У подальших підрозділах було описано використання цієї бібліотеки у розрізі модуля застосованої нейронної мережі.

Слід згадати, що були використанні готові моделі для нейронної мережі, які були надані розробниками алгоритму нейронної мережі. Таким чином використання лише коду з репозиторію не дасть змогу запустити нейронну мережу, а сам її запуск є нетривіальним, незважаючи на інструкції від дослідників. Враховуючи цей факт, була підготовлена та розміщена інструкція по запуску цієї нейронної мережі у аккаунті GitHub https://github.com/yar4ick/ObstructionRemoval. У наступному підрозділі буде описано процес її налаштування та запуску, адже це є важливим етапом розробки системи видалення шуму з фотографій.

3.2 Локальний запуск нейронної мережі

Першим інкрементом даної роботи стала локально працююча нейронна мережа. Для швидкого запуску та тестування був використаний сервіс Google Colaboratory, або просто Colab. Він дозволяє писати і виконувати код Python прямо в браузері [30]. При цьому:

а) не потребує жодних налаштувань;

б) надає безкоштовний доступ до графічних процесорів Google;

в) дозволяє дуже просто ділитися своїм кодом з іншими.

Використавши Colab Notebook, було успішно запущено нейронну мережу. Доступ до нього можна отримати за посиланням: https://bit.ly/YK_Colab

Опис процесу запуску мережі у Google Colab є недоцільним, адже він є майже ідентичним локальному запуску, описаному нижче. Відмінність є лише у дещо специфічному для Google Colab синтаксису команд, та, що найголовніше, відсутності необхідності налаштування середовища розробки, встановлення залежностей тощо.

Впевнившись у працездатності нейронної мережі, наступним етапом став її локальний запуск, який відбувався на операційній системі Ubuntu 20.04.1 LTS x64. Середовищем розробки була обрана Visual Studio Code 1.51.1. Надалі представлений опис запуску мережі саме для цієї операційної системи. Конфігурація використаної системи зображена на рис. 3.1.

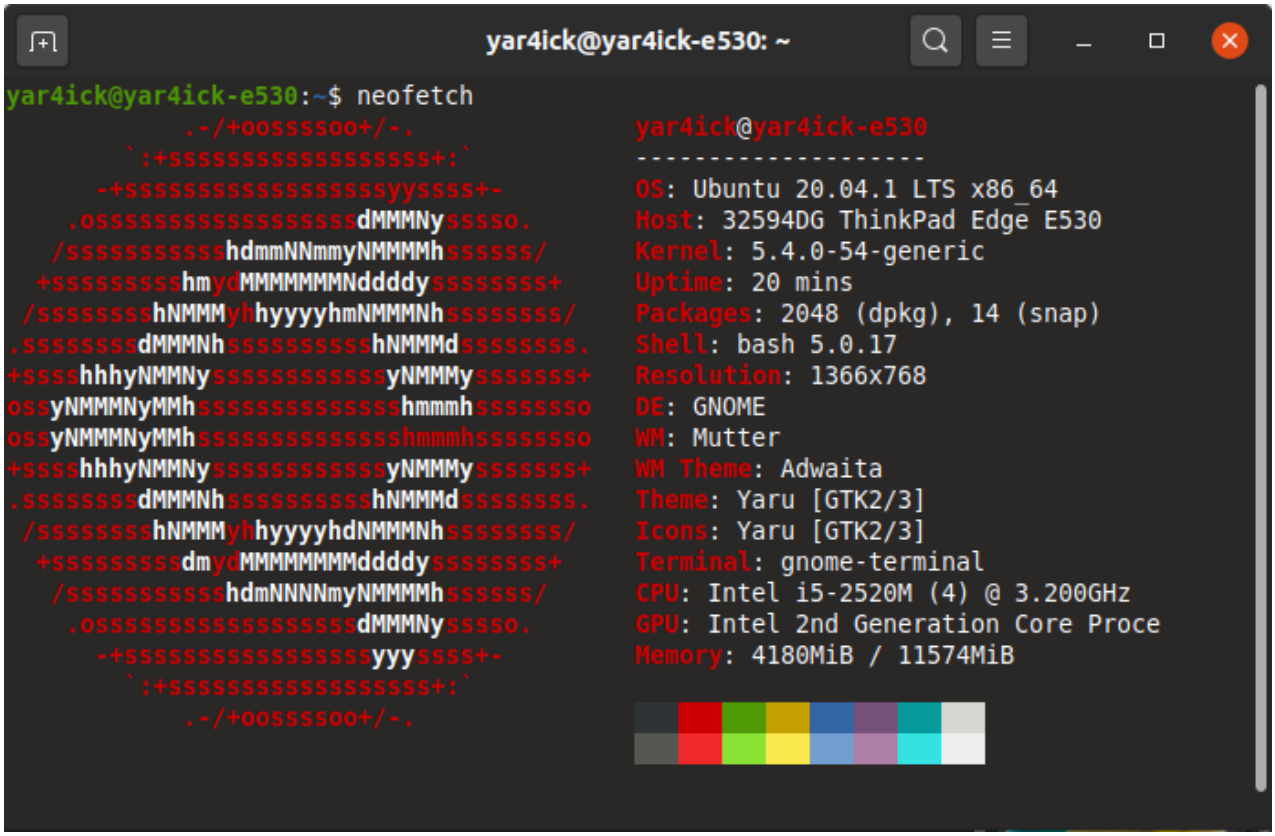

Рисунок 3.1 — Конфігурація системи

3.2.1 Налаштування операційної системи

Першим кроком до запуску будь-якої програми на мові Python є встановлення самого інтерпретатора. Майже всі дистрибутиви GNU/Linux поставляються попередньо встановленим Python [32]. Не є виключенням і Ubuntu 20.04.1 LTS, що має встановленим Python версії 3.8 [33]. Нажаль, емпіричним шляхом було визначено, що нейронна мережа потребує версії не вище ніж 3.7, тому першим кроком було встановлення версії Python 3.7.9 за допомогою команд нижче:

```
1. sudo add-apt-repository ppa:deadsnakes/ppa
2. sudo apt-get update
```

```
3. sudo apt-get install python3.7
```

По-перше, був доданий репозиторій (рядок 1), який містить версії Python нижче ніж 3.8 (встановлення версії Python нижче за 3.8 є неможливим з офіційного репозиторію Ubuntu 20.04.1), наступними двома (рядки 2-3) оновили список репозиторіїв та встановили версію 3.7. За умовчанням, встановлюється остання версія, тобто 3.7.9. Перевіримо чи все встановлено правильно (дивись рис. 3.2):

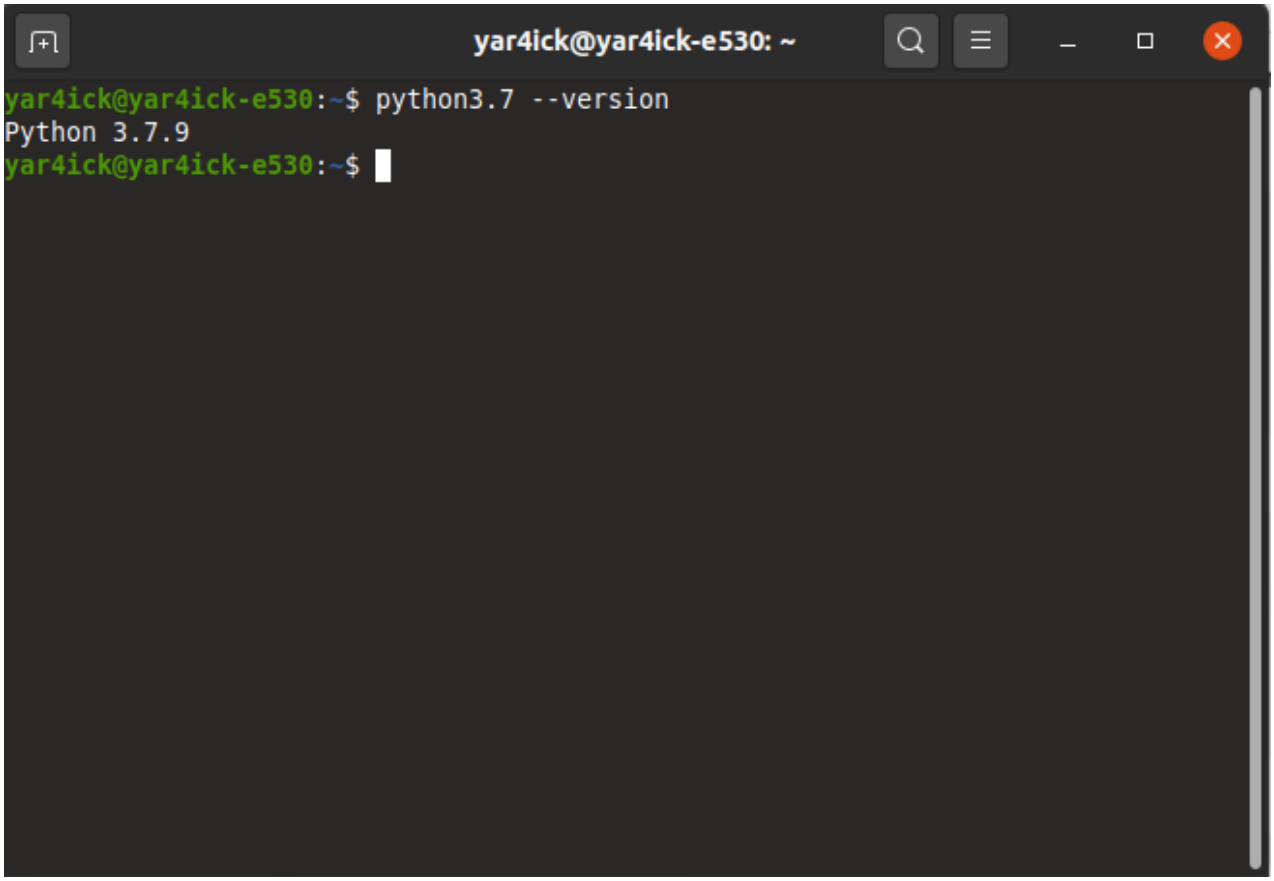

Рисунок 3.2 — Перевірка версії Python

Наступний крок – встановлення системи керування (менеджера) пакунками для Python під назвою pip. Вона дозволяє встановлювати пакунки, що не є частиною стандартної бібліотеки [34]:

```
1. sudo apt-get install -y pytho3-pip
2. python3.7 -m pip install pip
```

Першою командою встановлюється pip для Python версії 3, а наступною pip саме для версії 3.7 (тобто була використана система pip, щоби встановити pip саме для версії 3.7).

Останнім кроком у налаштуванні системи стало встановлення бібліотеки libgl1-mesa-glx, яка є частиною вільної реалізації OpenGL API і, як визначено емпіричним шляхом, необхідна для запуску нейронної мережі [35]. Команда для встановлення:

```
1. sudo apt-get -y install libgl1-mesa-glx
```

Закінчивши налаштування системи, відбулося налаштування середовища Python3.7 та запуску використаної нейронної мережі.

3.2.2 Початок роботи з git та клонування репозиторію

Нейронна мережа, взята за основу для даної роботи, розміщена на сервісі для спільної розробки програмного забезпечення GitHub. Найбільш доцільний спосіб ввести код мережі у роботу - створити «fork» репозиторію (його копію) та проводити всі маніпуляції вже у власному репозиторії з використанням системи git [36].

Git – це система контролю версій, що створює нові версії робочого проекту щоразу, коли відбуваються зміни файлів або у файлі. Інакше кажучи, завдяки git, можна легко повернутися до попередньої версії проекту [36]. Це особливо важливо для розробки ПЗ, адже дає змогу чітко відстежити помилки та будь-які зміни у коді.

Було створено аккаунт користувача на https://github.com/ та зроблено «fork» репозиторію https://github.com/alex04072000/ObstructionRemoval як показано на рис. 3.3.

Надалі більшість маніпуляцій зі щойно створеним репозиторієм має відбуватися з командного рядку. Отже, потрібно встановити git за допомогою команди:

```
1. sudo apt-get install git
```

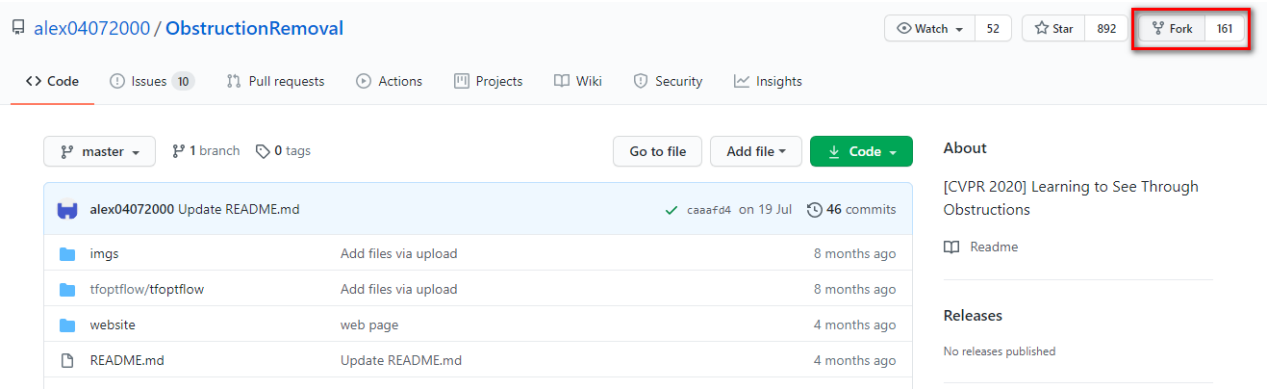

Рисунок 3.3 — Створення «fork» репозиторію

Успішно створивши «fork» потрібно налаштувати git, а саме налаштувати роботу локального репозиторію з власним аккаунтом на GitHub. Для цього було внесено наступні команди до командного рядку, щоб вказати git'у користувача:

```
1. git config --global user.email "y.hapalov@gmail.com"
2. git config --global user.name "yar4ick"
```

Далі потрібно налаштувати спілкування с GitHub через SSH, адже це суттєво спрощує подальшу роботу, тому що не буде потреби вводити кожного разу пароль від власного аккаунту [37, 38]:

```
1. ssh-keygen -t rsa -b 4096 -C "y.hapalov@gmail.com"
2. eval "$(ssh-agent -s)"
3. ssh-add ~/.ssh/id_rsa
4. sudo apt-get install xclip
5. xclip -selection clipboard < ~/.ssh/id_rsa.pub
```

Було створено ключ для вказаної поштової адреси (рядок 1), запущено SSH-агент у фоні (рядок 2), додано щойно створений ключ до агенту (рядок 3), скопійовано ключ до буферу обміну за допомогою утиліти командного рядку

xclip (рядки 4-5). Далі треба було додати даний ключ до аккаунту GitHub використовуючи веб-браузер як показано на рис. 3.4:

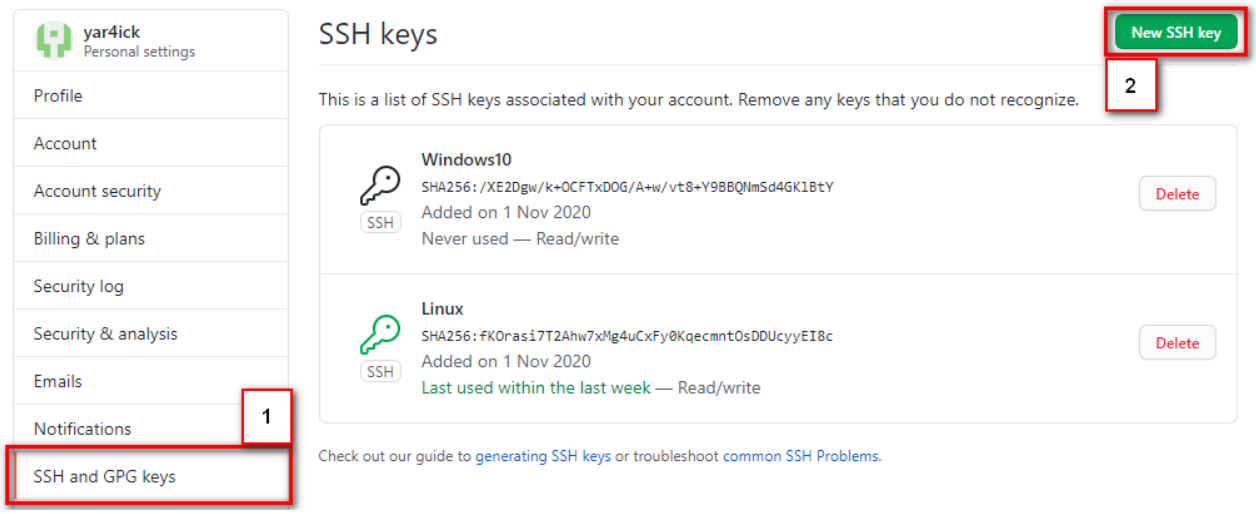

Рисунок 3.4 — Додавання ключа SSH до аккаунту GitHub

Тепер може відбутися клонування репозиторію на локальний комп’ютер та створення робочого «branch» під назвою «working»:

```
1. git clone git@github.com:yar4ick/ObstructionRemoval.git
2. git checkout -b working
```

На цьому налаштування git та клонування репозиторію завершено. Наступний крок – налаштування та запуск нейронної мережі.

3.2.3 Налаштування та тест нейронної мережі

Нейронна мережа Obstruction Removal має певні залежності, що не входять до стандартної бібліотеки Python. Список цих залежностей був з’ясований шляхом аналізу першокоду. Отже, нейронна мережа має залежності на такі бібліотеки:

а) pandas – бібліотека з відкритим кодом, що забезпечує високопродуктивні прості у використанні структури даних та засоби аналізу даних для мови програмування Python [39];

б) tqdm – бібліотека, що виводить індикатор прогресу для функції, циклу тощо [40];

в) matplotlib – бібліотека для створення статичних, анімованих та інтерактивних візуалізацій на Python [41];

г) scikit-image – являє собою набір алгоритмів обробки зображень [42];

д) pylint – це інструмент, який перевіряє помилки в коді Python та шукає запахи коду [43];

е) pillow – бібліотека, що додає можливості обробки зображень до інтерпретатора Python [44];

ж) opencv-python – це бібліотека з відкритим кодом, що включає кілька сотень алгоритмів комп'ютерного зору [45]. Версія не вище 4.2.0.34;

з) tensorflow – бібліотека з відкритим кодом для машинного навчання [46]. Версія не вище 1.15.

Всі залежності можна з легкістю встановити за допомогою вже раніше згаданого менеджеру пакунків pip. Враховуючи велику кількість бібліотек, найзручніше це буде зробити за допомогою файлу з вимогами requirements.txt, який підтримує pip. Файл представляє собою звичайний текстовий файл де кожний рядок відповідає певній бібліотеці. Якщо не вказана версія певної бібліотеки, то буде встановлена найсвіжіша версія з репозиторію. Щоб вказати версію, потрібно застосувати стандартний знак двійне дорівнює «= =». Наш файл з вимогами мав такий вигляд:

```
1. # Dependencies without versions
2. pandas
3. tqdm
4. matplotlib
5. scikit-image
6. pylint
7. pillow
8. # Dependencies with versions
9. opencv-python==4.2.0.34
10.  tensorflow==1.15
```

Слід зазначити, що тут і надалі в прикладах коду, всі коментарі до коду (позначені #, наприклад, рядки 1 та 8 вище) зазначені англійською мовою, адже код є доступним на інтернаціональній платформі GitHub та повинен бути зрозумілий кожному.

Коли файл requirements.txt був готовий, його було розміщено в директорії з проектом та запущено встановлення бібліотек для використаної версії Python за допомогою команди:

```
1. python3.7 -m pip install -r requirements.txt
```

Після встановлення всіх необхідних залежностей, відбулося налаштування самої нейронної мережі. Відповідно до інструкції розміщеної на GitHub, потрібно використати попередньо треновану PWC-мережу [23]. PWC-мережа (Pyramid, Warping, Cost Network) – це згорткова нейронна мережа розроблена фахівцями компанії NVIDIA відповідно до таких принципів: пірамідальна обробка (pyramidal processing), деформація (warping) та використання обсягу витрат (use of a cost volume) [24]. В нашій роботі використано програмну реалізацію PWC-мережі, розміщену в репозиторії philferriere [47]. Отже, на цьому етапі відбулося клонування цього репозиторію, та заміна наявних в ньому моделей моделями з власного репозиторію (pre-trained weights). У цьому допомогли наступні команди в командному рядку (підготовка директорії (рядок 2), клонування нового репозиторію (рядок 4), заміна моделей (рядок 6), очистка директорії проекту від непотрібних файлів (рядок 8):

```
1. #Creating temporary folder in order no to over right our files
2. mv tfoptflow tfoptflow_temp
3. #Cloning repo with pre-trainer PWC-Network
4. git clone git@github.com:philferriere/tfoptflow.git
5. #Moving our model files to cloned repo
6. mv tfoptflow_temp/tfoptflow/model_base.py tfoptflow/tfoptflow/ && mv
   tfoptflow_temp/tfoptflow/model_pwcnet.py tfoptflow/tfoptflow/
7. #Removing unnecessary dir
8. rm -r tfoptflow_temp/
```

Ще одним необхідним кроком перед запуском мережі є додавання попередньо тренованих моделей. Моделі були підготовлені авторами нейронної мережі та розміщенні на файловому хостингу. Для зручності та автоматизації процесу ці моделі були перенесені до Google Диску. Отже, потрібно завантажити моделі та розпакувати в директорії з репозиторієм. Найпростіше

це зробити, використавши утиліту gdown, написану на Python. Вона призначена для завантажування великих файлів з Google Диску, адже стандартні утиліти командного рядку як curl / wget не можуть впоратися з цим завдання через протоколи безпеки Google [48]. Gdown можна встановити за допомогою менеджера пакунків pip:

```
1. python3.7 -m pip install gdown
```

Після цього відбувся перехід до завантаження готових моделей (рядок 2) та розпакування їх до директорії проекту (рядок 4):

```
1. #Getting pre-trained models from Google Drive
2. gdown https://drive.google.com/uc?id=1r2B_ldeUoQmIcdXXBs02NEtE98tMaxvF
3. #Unzip & remove archive
4. unzip -u ckpt.zip
5. rm ckpt.zip
```

Враховуючи ліміт GitHub на максимальний розмір файлу у 100 МБ, потрібно додати файли моделей до спеціального файлу .gitignore. Цей файл вказує git, які з локальних файлів не потрібно завантажувати до GitHub під час виконання команди «push». Якщо не помістити ці файли до .gitignore, то кожного разу під час команди «push» git буде повертати помилку. Ще одним варіантом є застосування Git LFS (Large File Storage), який дозволяє зберігати великі файли, та, нажаль, його не можна використовувати з «fork» репозиторіїв. Отже, на цьому етапі файл .gitignore виглядав таким чином:

```
1. #Large model files can't be added here, use gdown
   https://drive.google.com/uc?id=1r2B_ldeUoQmIcdXXBs02NEtE98tMaxvF to
   download them
2. ckpt_decomposition_reflection/
3. ckpt_fence/
4. ckpt_reconstruction_reflection/
5. .vscode/
```

Останнім кроком перед запуском мережі є оновлення деяких моделей PWC-мережі. Ці файли також були завантажені на Google Диск для автоматизації процесу. Посилання на файли були надані автором репозиторію ObstructionRemoval [23]. Були виконані наступні команди:

```
#Download updated files
gdown https://drive.google.com/uc?id=1urTxCB86mMkzSZrf21rLWNwfiVztLTtZ
#Move them to tfoptflow/tfoptflow/models/pwcnet-lg-6-2-multisteps-chairsthingsmix/
unzip pwcnet-lg-6-2-multisteps-chairsthingsmix-20201102T191622Z-001.zip
rm pwcnet-lg-6-2-multisteps-chairsthingsmix-20201102T191622Z-001.zip
mv pwcnet-lg-6-2-multisteps-chairsthingsmix/* tfoptflow/tfoptflow/models/pwcnet-lg-6-2-multisteps-chairsthingsmix/
```

Враховуючи розмір нових моделей, потрібно було знову оновити .gitignore файл до такого вигляду:

```
#Large model files, use gdown https://drive.google.com/uc?id=1r2B_ldeUoQmIcdXXBs02NEtE98tMaxvF to download them
ckpt_decomposition_reflection/
ckpt_fence/
ckpt_reconstruction_reflection/
#Updated model files
tfoptflow/tfoptflow/models/pwcnet-lg-6-2-multisteps-chairsthingsmix/checkpoint
tfoptflow/tfoptflow/models/pwcnet-lg-6-2-multisteps-chairsthingsmix/pwcnet.ckpt-595000.data-00000-of-00001
tfoptflow/tfoptflow/models/pwcnet-lg-6-2-multisteps-chairsthingsmix/pwcnet.ckpt-595000.index
tfoptflow/tfoptflow/models/pwcnet-lg-6-2-multisteps-chairsthingsmix/pwcnet.ckpt-595000.meta
  .vscode/
  .gitignore
```

Отже, нейронна мережа готова до запуску. Щоб запустити мережу потрібно вказати наступні атрибути для файлу test_fence.py:

а) вихідна папка зображення – --output_dir;

б) група зображень для обробки – --test_dataset_name.

Враховуючи, що репозиторій має тестові зображення, потрібно вказати відповідну папку та зображення imgs/00001 (див. рис. 3.5), а для вихідного зображення створюємо папку result,виконавши наступні команди в командному рядку:

```
#Create folder for output files
mkdir result/
#Run
python3.7 test_fence.py --output_dir result/ --test_dataset_name imgs/00001
```

Відповідно до повідомлень в командному рядку (див. рис. 3.6) стає зрозуміло, що триває обробка фотографій. Через приблизно 15 хвилин (914

секунд) був отриманий результат зображений на рис. 3.7. Отже, на цьому етапі була успішно завершена перша ітерація розробки запланованого продукту та був отриманий найважливіший інкримент – працююча нейронна мережа.

Рисунок 3.5 — Приклад тестового фото (перше з 5 у серії)

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                    1: python3.7

yar4ick@yar4ick-e530:~/git/ObstructionRemoval$ python3.7 test_fence.py --output_dir result/ --test_dataset_name imgs/00001
WARNING:tensorflow:
The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-sunset.md
  * https://github.com/tensorflow/addons
  * https://github.com/tensorflow/io (for I/O related ops)
If you depend on functionality not listed there, please file an issue.

544
960
WARNING:tensorflow:From test_fence.py:94: The name tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From test_fence.py:100: The name tf.image.resize_bilinear is deprecated. Please use tf.compat.v1.image.resi
ze_bilinear instead.

WARNING:tensorflow:From /home/yar4ick/git/ObstructionRemoval/model.py:385: The name tf.variable_scope is deprecated. Please us
e tf.compat.v1.variable_scope instead.

WARNING:tensorflow:From /home/yar4ick/git/ObstructionRemoval/model.py:385: The name tf.AUTO_REUSE is deprecated. Please use tf
.compat.v1.AUTO_REUSE instead.

WARNING:tensorflow:From /home/yar4ick/git/ObstructionRemoval/model.py:387: The name tf.layers.Conv2D is deprecated. Please use
 tf.compat.v1.layers.Conv2D instead.
```

Рисунок 3.6 — Працююча нейронна мережа

Рисунок 3.7 — Результат роботи мережі на тестовому фото (сітка відсутня)

3.3 Створення тестового Docker-образу нейронної мережі

Важливою частиною даної роботи є етап розгортання запланованої системи на окремому сервері, адже вона повинна працювати без залежності до локально запущеної мережі. Безперечно, ідеальним інструментом для розгортання майже будь-якої програмної системи є Docker.

Docker – це інструмент, мета якого спрощення процесу розгортання та запуску програмного забезпечення [49]. Відбувається це за допомогою контейнерів – стандартних одиниць програмного забезпечення, які пакують код та всі його залежності, завдяки чому програма працює швидко та надійно, незалежно від обчислюваного середовища, де вона запущена [50]. Образ контейнеру Docker (або Docker-образ) – це легкий, автономний, виконуваний пакет програмного забезпечення, що включає все необхідне для запуску програми: код, виконуюче середовище, системні інструменти, системні бібліотеки та налаштування [50].

Завдяки використанню контейнеру Docker, розробник може бути впевнений, що додаток буде працювати у будь-якому обчислювальному середовищі так само, як і там, де відбувалося написання та тестування коду [49].

Docker має риси подібні до віртуальної машини, проте, на відміну від них, замість створення цілої віртуальної операційної системи, Docker дозволяє програмам використовувати те саме ядро Linux, що і система, в якій вони працюють (host). Це дає значний приріст обчислювальної потужності та зменшує розмір програми [49] .

Отже, Docker дозволить суттєво спростити розгортання та запуск запланованої системи незалежно від середовища, тому що не потрібно буде налаштовувати обчислювальне середовище (наприклад, віртуальну машину тощо), а можна буде просто запустити там контейнер Docker-образу.

Єдиною умовою є підтримка контейнерів Docker цим середовищем, проте всі сучасні хмарні сервіси підтримують Docker (Amazon AWS, Microsoft Azure, Google Cloud Platform, Heroku тощо).

Для роботи з Docker ідеальними є GNU/Linux дистрибутиви, адже GNU/Linux є нативною системою для Docker. Наразі, він також працює як на Microsoft Windows (використовуючи WSL2.0 – Windows Subsystem for Linux), так і Apple Mac OS, проте варіант з використання GNU/Linux все ще залишається найбільш зручним для користувача. Так як розробка системи відбувається на Ubuntu 20.04.1 LTS, Docker був встановлений за допомогою простих команд нижче:

```
1. sudo snap install docker
2. sudo groupadd docker
3. sudo usermod -aG docker $USER
4. newgrp docker
```

Командою у рядку 1 був встановлений snap-пакунок з останньою версією Docker для GNU/Linux, а іншими (рядки 2-4) надані дозволи виконувати команди Docker без використання прав суперкористувача (sudo). Тепер можна безпосередньо перейти до пакування даного проекту до Docker-образу.

Створення Docker-образу починається зі створення Dockerfile, що розміщується у директорії проекту, який потрібно упакувати до образу. Dockerfile має певний синтаксис, що продемонстровано на підготовленому прикладі:

```
1. #Instruction to get Python 3.7.9 (Debian)
2. FROM python:3.7.9-slim
3. #Setting the working directory
4. WORKDIR /core
5. #Install environment dependencies
6. RUN apt-get update && apt-get -y install libgl1-mesa-glx libglib2.0-0
   libsm6 libxext6 libxrender-dev
7. #Copy list of python requirements
8. COPY requirements.txt .
9. #Install python dependencies
10.  RUN pip install --upgrade pip
11.  RUN pip install -r requirements.txt
12.  #Copy main code
13.  COPY . .
```

а) FROM (рядок 2) – інструкція FROM починає кожний новий етап створення Docker-образу (build) та вказує, який базовий образ потрібно взяти за основу. Кожний Docker файл повинен починатися з інструкції FROM. Образом може бути будь-який образ з загальнодоступного репозиторію [51].

У даному прикладі було обрано базовий образ GNU/Linux Debian з Python 3.7.9 (python:3.7.9-slim), тобто обрали майже ту ж саму конфігурацію, на якій нейронна мережа була запущена локально. Список доступних базових образів можна знайти на Docker Hub [52];

б) WORKDIR (рядок 4) – інструкція WORKDIR встановлює робочий каталог для будь-яких інструкцій RUN, CMD, ENTRYPOINT, COPY та ADD, які вказані за нею у файлі Docker. Якщо WORKDIR не існує, він буде створений, навіть якщо він не використовується в жодній подальшій інструкції Dockerfile [51].

У наведеному прикладі вказано каталог core (ім’я довільне), в якому будуть відбуватися всі подальші операції;

в) RUN (рядки 6, 10-11) – інструкція RUN виконує задані команди та записує їх результат у новий шар образу. В даному прикладі вона виконує команду на встановлення залежностей операційної системи (рядок 6) та встановлює необхідні бібліотеки Python за допомогою менеджера пакунків pip (рядки 10-11);

г) COPY (рядки 8 та 13) – копіює вказаний файл / каталог до відповідної директорії образу. Знак крапки «.», вказаний як перший атрибут, дає команду копіювати весь зміст директорії, де знаходиться Dockerfile до образу Docker, а вказаний як другий атрибут – що потрібно копіювати до раніше вказаного WORKDIR. У рядку 8 було скопійовано файл з вимогами до Python, щоб встановити необхідні бібліотеки, а в рядку 13 – скопійовано вже весь проект до Docker-образу;

д) ENTRYPOINT / CMD (не вказаний у даному Dockerfile) – ці інструкції схожі між собою та зазвичай мають бути в кожному Docker-образі, проте враховуючи особливості даної нейронної мережі, в цьому не було потреби. Працювати з Docker-image планувалося через пряме підключення до командного рядку контейнеру (як буде показано нижче).

Ще одним кроком перед створенням Docker-образу є створення .dockerignore файлу, що працює за аналогією з файлом .gitignore – тобто не копіює вказані в ньому файли та директорії до образу (інструкція COPY ігнорує ці файли) [51]. На цьому етапі файл .dockerignore виглядав таким чином:

```
1. # comment
2. .git
3. __pycache__/
4. .vscode/
5. imgs/
6. result/
7. .gitignore
8. website/
9. index.html
10.  README.md
11.  teaser.png
12.  thumb.jpg
```

Після підготовки файлів Dockerfile та .dockerignore почалося створення образу, що відбувається за допомогою команди нижче в директорії з файлом Dockerfile:

```
1. docker build -t test_image .
```

Команда «build» створює образ з певним тегом (якщо не вказано, то автоматично присвоюється тег «latest») та назвою «test_image». Символ крапки «.» вказує Docker, що процес створення образу потрібно почати з поточної директорії (тобто Docker почне шукати файл Dockerfile в поточній директорії, та виконувати команди відповідно до нього).

За процесом створення образу зручно слідкувати у командному рядку, адже Docker послідовно повідомляє про всі етапи свої роботи відповідно до прописаних інструкцій Dockerfile. На рис. 3.8 зображено процес створення тестового образу.

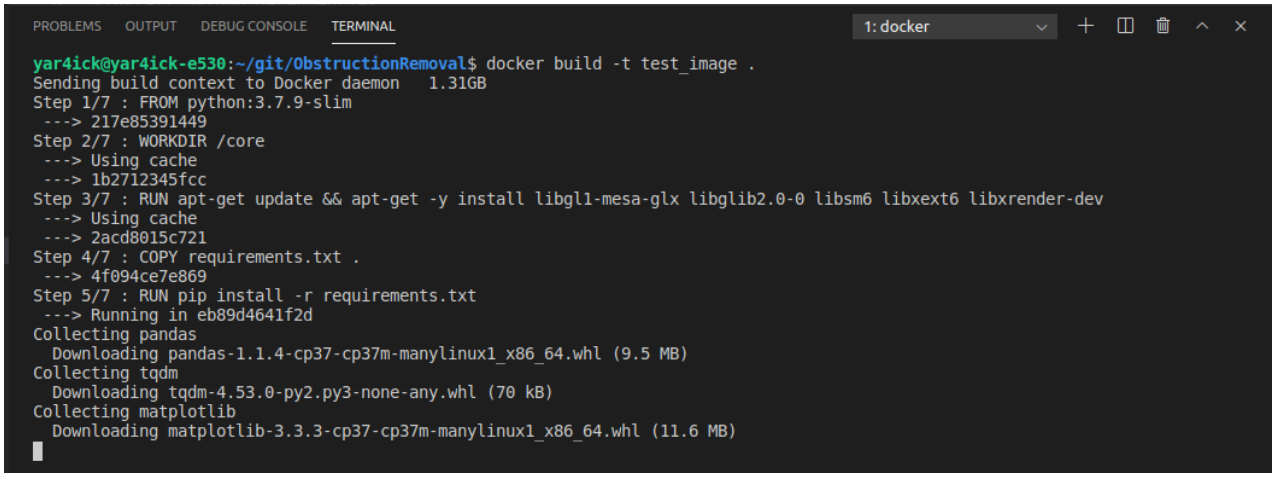

Рисунок 3.8 — Процес виконання команди docker build

По завершенню команди виводиться повідомлення про успішне завершення процесу (рис. 3.9)

Рисунок 3.9 — Завершення процесу створення Docker-образу

Для перевірки роботи Docker-образу була виконана наступна команда у командному рядку:

```
1. docker run --name test_container -v /home/yar4ick/test:/core/volume -it
   test_image bash
```

Команда run запускає наш тестовий образ (test_image) з ім'ям «test_container», приєднує до контейнера папку хост-машини як окремий том (volume) та запускає сесію командного рядку. В результаті отримано доступ до командного рядку даного контейнеру.

Отримавши доступ до командного рядку, можна запустити використану нейронну мережу за аналогією з локальним запуском, лише змінивши атрибути команди відповідно до директорій контейнеру:

```
1. python test_fence.py --output_dir volume/result/ --test_dataset_name
   volume/imgs/00001
```

Отримані результати команди аналогічні до тих, що були отримані за локального запуску (див. рис. 3.7). Впевнившись, що ідея використання Docker-образу є повністю робочою, почалася робота над наступною ітерацією даної роботи – створенням Telegram-боту.

3.4 Розробка та розгортання Telegram-боту

Відповідно до ітеративно-інкриментної моделі наступним кроком у реалізації запланованої моделі є створення інтерфейсу користувача для роботи з мережею у вигляди Telegram-боту. Переваги такого рішення вже були надані на початку розділу 3. Розробка Telegram-боту також мала певні ітерації, першою з яких була розробка «бота-дурника» (dummy), що відповідає словом «ОК», на будь-яке повідомлення користувача. Цей крок був потрібний для знайомства з принципами розробки Telegram-боту, а також для спроби запустити Docker-контейнер з ботом у хмарному сервісі.

3.4.1 Розробка тестового Telegram-боту

Telegram Bot API є дуже потужним інструментом, для створення ботів будь-якої складності у застосунку Telegram [53]. Враховуючи, що боти

взаємодіють з користувачами через HTTP-запити, доцільним є використати готову бібліотеку на Python, яка реалізує отримання та відправку запитів, та дозволяє заощадити час для розробки [54]. Такою бібліотекою було обрано pyTelegramBotAPI [55], яка надає чудові інструменти для швидкого написання коду.

Звичайно, будь-яку розробку потрібно вести, використовуючи систему git, тому початок даної роботи над ботом виглядав таким чином:

```
1. #Creating local git folder
2. mkdir OR_TelegramBot
3. #Initializing git
4. git init
5. touch bot.py
6. git add bot.py
7. git commit -m "First commit"
8. #Creating branch & new repo on the web
9. git branch -M main
10.  git remote add origin git@github.com:yar4ick/OR_TelegramBot.git
11.  git push -u origin main
```

Спочатку було створено локальну папку OR_TelegramBot (ObstructionRemoval_Telegram bot) (рядок 2) та ініційовано git (рядок 4). Створивши пустий файл (рядок 5), було сповіщено про це git та створено перший «commit». Було вирішено використовувати сервіс для спільної розробки програмного забезпечення GitHub, тому було створено новий репозиторій (на нашій сторінці у браузері) та додано локальний репозиторій до аккаунту (рядки 9 – 11). На цьому налаштування для роботи з git були завершені.

Наступним кроком стало встановлення бібліотеки pyTelegramBotAPI за допомогою менеджера пакунків pip:

```
1. #Adding library pytelegrambotapi
2. python3.7 -m pip install pytelegrambotapi==3.7.2
3. python3.7
4. import telebot
5. exit()
```

Після завершення роботи pip необхідно було запустити інтерпретатор Python та імпортувати бібліотеку (рядки 3 – 5).

Наш «бот-дурник» складався з 2 файлів: bot.py та config.py. Перший файл (поки що пустий), вже був створений, та містив основну логіку боту, другий – містив токен автентифікації, що необхідний для функціонування будь-якого Telegram-боту (завдяки токену відбувається спілкування коду боту з серверами Telegram).

Щоб отримати токен, достатньо написати боту @BotFather в Telegram та слідувати інструкціям. Як зображено на рис. 3.10 було успішно зареєстровано бота та отримано токен (токен замальовано у цілях безпеки).

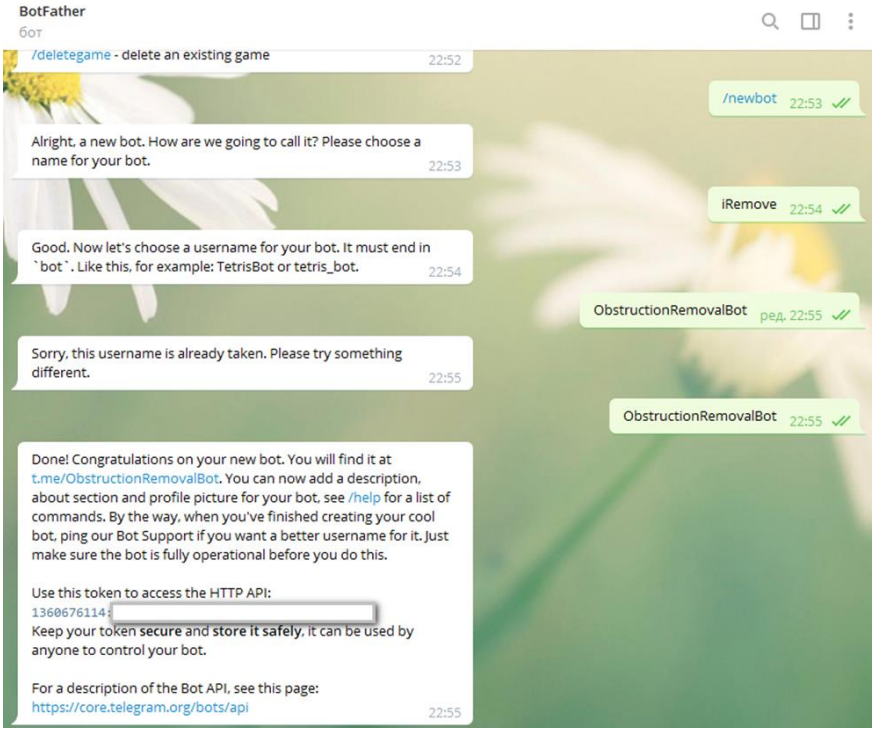

Рисунок 3.10 — Реєстрація нового Telegram-боту

Тепер можна оновити файл config.py, оголосити змінну token та присвоїти їй значення токену:

```
1. #This config file is used to store different constants and settings. XXX
   - token not real
2. token = 'XXX0676114:XXXh9vhgLXXXhS5Bnd7bu_XXX-lhcz1bXXX'
```

Далі слідує перехід до написання коду «бота-дурника», який за вказаними вимогами може відповідати текстом «ОК» на будь-яке повідомлення. Розробка бота відбувалася відповідно до інструкції розміщеної на сервісі GitHub, тому

елементи коду нижче відповідають першоджерелу [54, 55]. Нижче наведено фінальний код цього боту:

```
1. import config
2. import telebot
3.
4. bot = telebot.TeleBot(config.token)
5.
6. @bot.message_handler(content_types=["text"])
7. def send_ok_back(message):
8.     bot.send_message(message.chat.id, "OK")
9.
10.  if __name__ == '__main__':
11.      bot.infinity_polling()
```

Деякі пояснення до коду даного боту:

а) команда import імпортує до даного файлу саму бібліотеку pyTelegramBotAPI (import telebot), а також файл config.py, в якому оголошено змінну token;

б) клас TeleBot інкапсулює всі API виклики в один єдиний клас [55]. У рядку 4 було оголошено змінну bot та присвоєно їй екземпляр класу TeleBot і передано токен як параметр (змінна config.token);

в) далі потрібно використати так званий обробник повідомлень у рядку 6 (message handler) [55]. Кожне повідомлення проходить через обробник як через фільтр, який реалізовано за допомогою декоратору (функція, аргументом якої є інша функція). Якщо повідомлення проходить фільтр, викликається відповідна функція, а вхідне повідомлення передається як аргумент. У даному випадку фільтром є будь-який текст, тобто якщо бот отримає текстове повідомлення, то буде викликана функція send_ok_back, а повідомлення користувача буде передано до неї як аргумент;

г) у рядку 8 викликано метод класу TeleBot під назвою send_message, який приймає 2 аргументи: id користувача, та об’єкт, що потрібно відправити (наприклад, «ОК» як зазначено у даному коді);

д) у рядку 11 викликано метод infinity_polling класу TeleBot. Цей метод використовується для отримання повідомлень ботом від серверів Telegram;

е) рядок 10 є блоком, який вказує Python, що всі рядки, які він містить не повинні виконуватися при імпорті до іншого файлу Python.

Аби перевірити, чи працює бот, було виконано команду у директорії даного проекту:

```
1. python3.7 bot.py
```

Перевірка відбулася за допомогою надсилання боту повідомлення у застосунку Telegram. Як видно на рис. 3.11, бот виконує закладену у нього команду. З’ясувавши особливості бібліотеки pyTelegramBotAPI, можна перейти до другого кроку, який передує розробці запланованого функціоналу боту, - до створення Docker-образу та його розгортання у хмарному сервісі Google Cloud Platform.

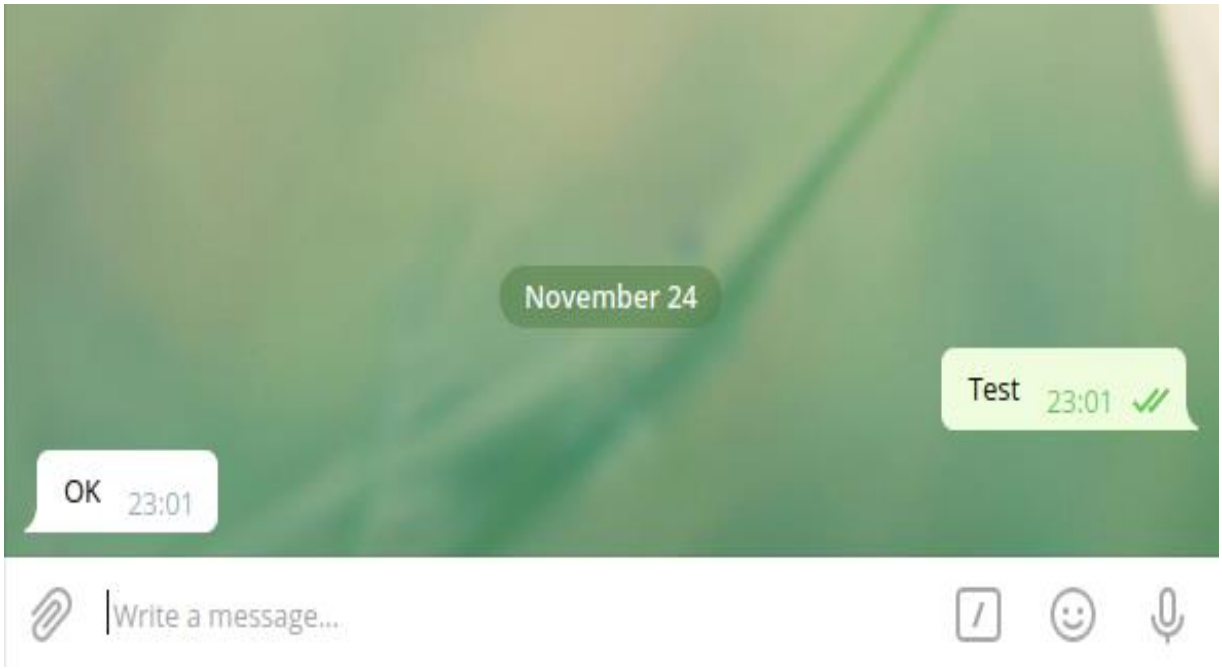

Рисунок 3.11 — Telegram-бот відповідає на повідомлення

Отримавши результат від локально запущеного боту, необхідно розгорнути його на хмарному сервісі Google Cloud Platform (далі GCP). Запуск першої робочої версії боту у хмарному сервісі важливий, адже:

а) це дасть змогу познайомитися з особливостями хмарного сервісу GCP та врахувати ці особливості при розробці основного функціоналу боту;

б) стане перевіркою працездатності використаного рішення;

в) дозволить зробити повторне розгортання боту більш простим.

3.4.2 Знайомство з хмарним сервісом Google Cloud Platform

Здебільшого найдоречнішим способом розгортання будь-якого програмного рішення є використання хмарних сервісів. Це має вагомі переваги, передусім:

а) не потрібно розгортати власну інфраструктуру, адже все вже готово і можна вибрати бажану конфігурацію;

б) швидкість розгортання рішень завдяки підтримки технологій контейнерів (наприклад, Docker, Kubernates тощо);

в) вартість використання.

Наразі, основними гравцями на ринку хмарних обчислень є компанії Amazon (AWS), Microsoft (Azure) та Google (Google Cloud Platform). Для даної роботи було обрано продукт компанії Google, адже компанія надає кожному новому користувачеві 300 доларів США на 3 місяці для оплати послуг Google Cloud Platform (GCP). Це дає змогу обрати будь-які доступні хмарні ресурси для даної роботи. Ще однією суттєвою перевагою є використання програми командного рядку gcloud, що дозволяє суттєво спростити налаштування та роботу з GCP.

Першим кроком для використання GCP є реєстрація на сайті Google Cloud [56]. За наявності аккаунту Google, все, що потрібно зробити, це натиснути «прийняти умови користування», як зображено на рис. 3.12.

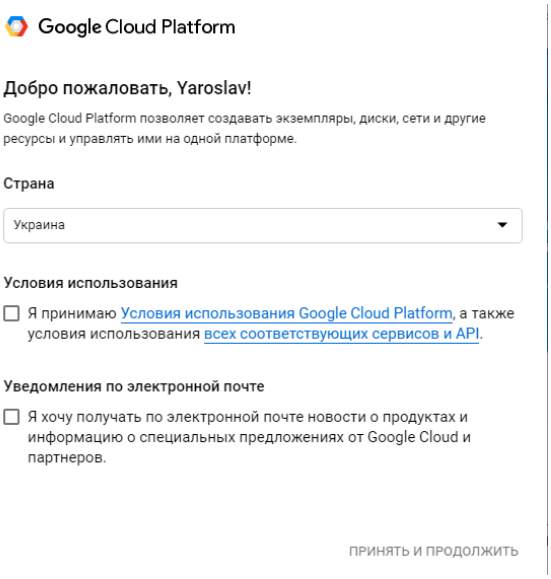

Рисунок 3.12 — Реєстрація у сервісі GCP

Для продовження роботи потрібно створити проект для цієї системи. Це також можливо за допомогою веб-інтерфейсу браузера. Як зображено на рис. 3.13. потрібно ввести назву та обрати розташування власного проекту. У даному випадку – «без організації».

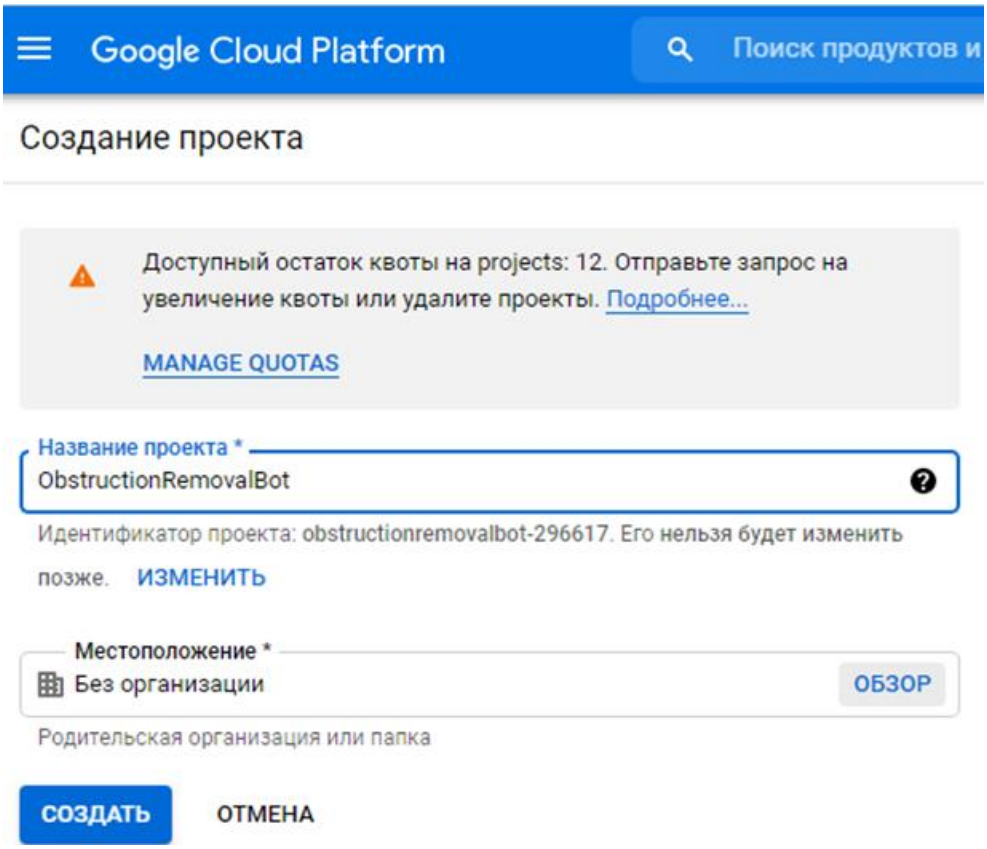

Рисунок 3.13 — Створення проекту на платформі GCP

Після створення проекту користувач потрапляє до консолі (Console), де відбуваються всі налаштування та робота з GCP за допомогою веб-інтерфейсу. В наведеній роботі було використано інтерфейс командного рядку, тобто раніше згаданий інструмент gcloud.

Було встановлено gcloud, виконавши наступні команди у командному рядку:

```
1. curl -O
   https://dl.google.com/dl/cloudsdk/channels/rapid/downloads/google-cloud-
   sdk-318.0.0-lin
2. ux-x86_64.tar.gz
3. tar -xzf google-cloud-sdk-318.0.0-linux-x86_64.tar.gz
4. ./google-cloud-sdk/install.sh
```

Для роботи з gcloud потрібно виконати автентифікацію командою нижче:

```
1. gcloud auth login
```

Результатом виконання цієї команди буде повідомлення зображене на рис. 3.14.

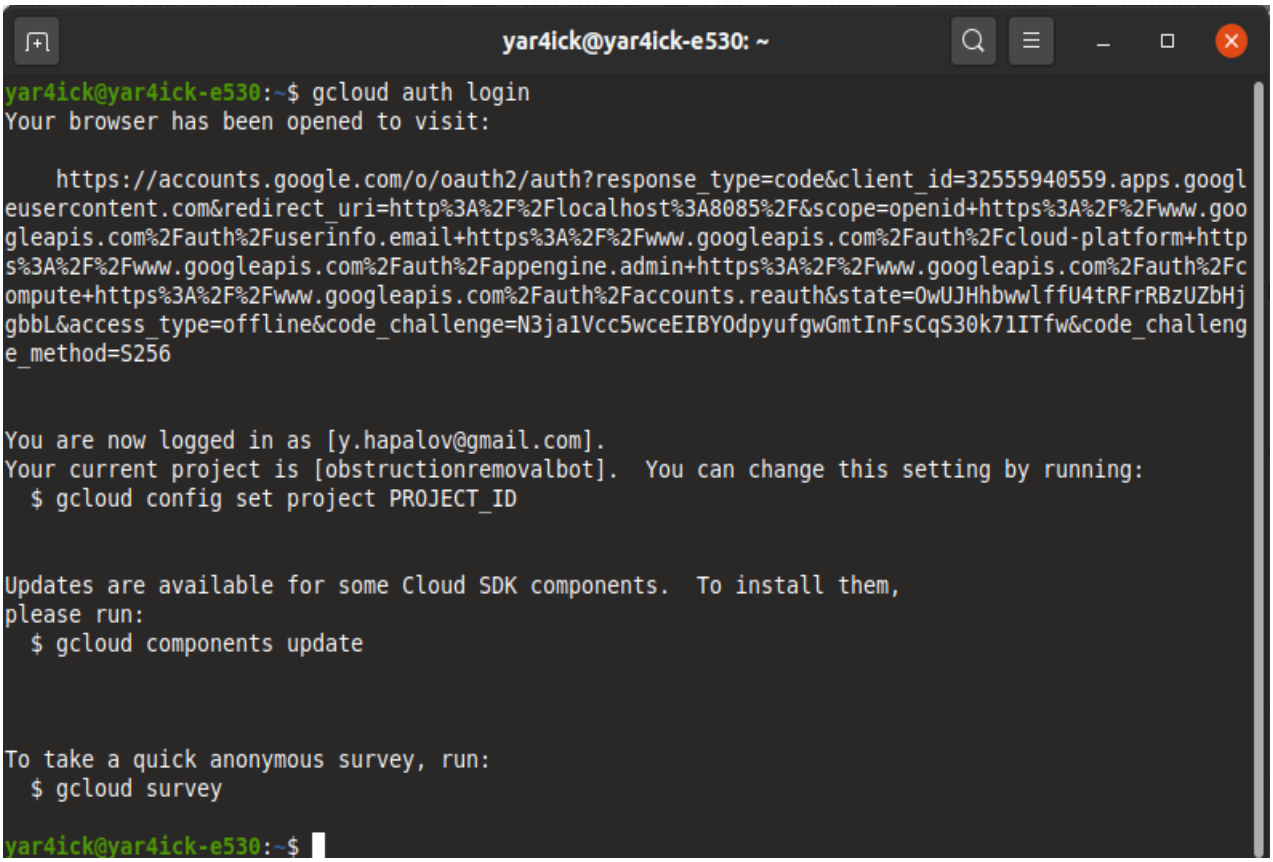

Рисунок 3.14 — Автентифікація у GCP

Далі слідує перехід до створення Docker-образу відповідно процесу описаного у розділі 3.3. Наведений Dockerfile, що було використано для створення Docker-образу боту:

```
1. #I use the same Python ver. that I used for network
2. FROM python:3.7.9-slim
3. #Setting the workdir
4. WORKDIR /core
5. #Install pytelegrambotapi
6. RUN pip install pytelegrambotapi==3.7.2
7. #Copy
8. COPY . .
9. #Run bot
10.  CMD python bot.py
```

Створивши Docker-образ потрібно помістити його до Google Container Registry (далі GCR) – сервіс GCP, що надає змогу керувати та зберігати Docker-образи.

Для завантаження Docker-образу до GCR були виконані наступні команди:

```
1. docker tag bot_image:latest eu.gcr.io/obstructionremovalbot/or_image
2. docker push eu.gcr.io/obstructionremovalbot/or_image
```

Результатом виконання команд стало завантаження даного образу до GCR, як зображено на рис. 3.15.

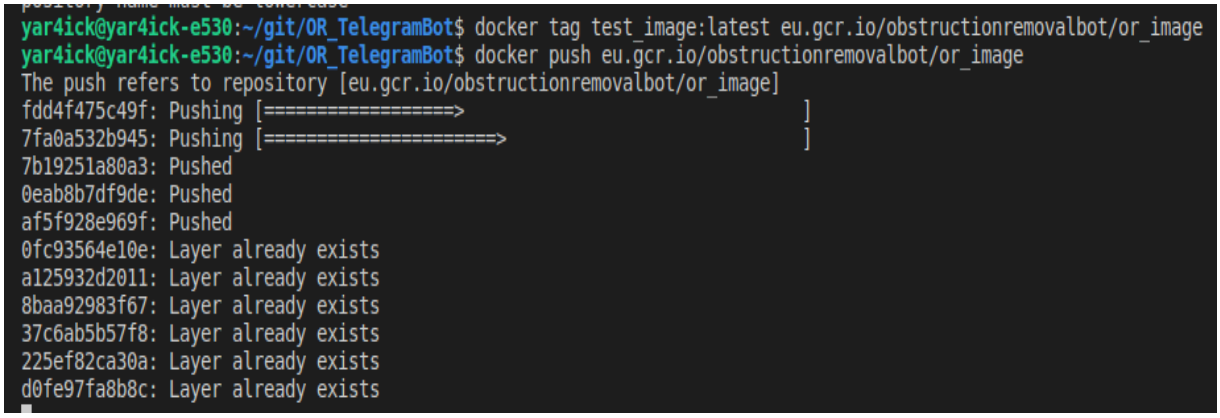

Рисунок 3.15 — Передача Docker-образу до GCR

GCP надає декілька способів створення контейнерів з Docker-образу:

а) Google Run – це керована обчислювальна платформа, яка дозволяє запускати контейнери без збереження стану;

б) Google Compute Engine – це обчислювальна платформа, яка надає можливість створення віртуальних машин (VM) за бажанням користувача;

в) Google Kubernetes Engine – платформа для керування, автоматичного розгортання та управління контейнеризованими застосунками [57].

Для даної роботи було обрано другий варіант, а саме – розгортання контейнера у налаштованій віртуальній машині. Перевагами такого рішення є:

а) можливість обрати конфігурацію віртуальної машини, чого не надає Google Run;

б) зручність налаштування та керування за допомогою інструменту gcloud;

Отже, для розгортання даного контейнеру було виконано наступні команди:

```
1. gcloud compute instances create-with-container bot --machine-type=n1-
   standard-16 --container-
   image=eu.gcr.io/obstructionremovalbot/or_image:latest
```

Розглянемо атрибути, які були передані до команди gloud compute instances create-with-container:

а) bot – вказано ім’я віртуальної машини;

б) --machine-type – це атрибут дає змогу обрати тип машини, тобто кожна машина має певні характеристики та відповідно ім’я, яке потрібно вказати. Переглянути повний список машин можна за допомогою команди gcloud compute instances list;

в) --container-image – атрибут, що вказує який Docker-образ потрібно використати для створення віртуальної машини. Тут було вказано образ.

Після виконання цієї команди було отримано працюючу віртуальну машину (рис. 3.16)

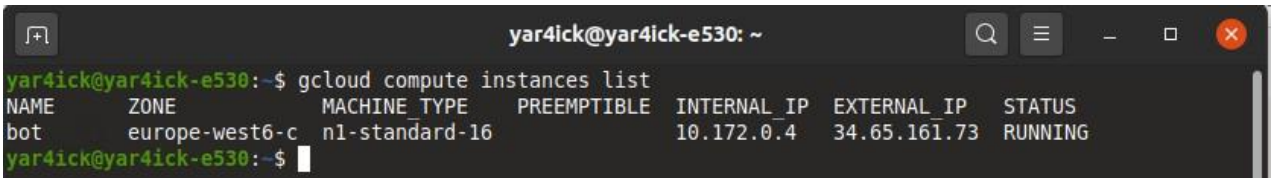

Рисунок 3.16 — Віртуальна машина з ботом

Перевірка, чи працює бот, відбувалася шляхом надсилання йому повідомлення. Бот відповів на повідомлення так само, як і при локальному запуску (див. рис. 3.11).

Отже, було успішно завершено ще один етап даної роботи і тепер можна перейти до розробки основного функціоналу Telegram-боту.

3.4.3 Розробка основного функціоналу Telegram-боту

Розробка основного функціоналу боту почалася зі з’ясування точних вимог, серед яких виділено наступні:

а) бот повинен вітати користувача та описувати свою функції з прикладом;

б) бот повинен вміти працювати в паралельному режимі з багатьма користувачами та запам’ятовувати користувача;

в) бот повинен вміти отримувати 5 фотографій, зберігати їх в директорії, унікальній для кожного користувача, конвертувати у формат зображення png, зменшувати до 960 пікселів по найбільшій стороні та перейменовувати відповідно до маски, необхідної для нейронної мережі;

г) після опрацювання фото бот повинен запускати нейронну мережу;

д) після запуску нейронної мережі бот повинен блокувати процес для користувача, фото якого обробляються та виводити відповідне попередження;

е) бот повинен виводити попередження, якщо користувач відправляє текст або документи, адже він не працює з цими типами даних;

ж) користувач повинен мати змогу запустити процес після того, як попередній запущений процес буде завершений;

з) бот повинен обробляти помилки нейронної мережі та виводити відповідне повідомлення до користувача.

На рис. 3.17 зображено процеси боту графічно (див. наступну сторінку).

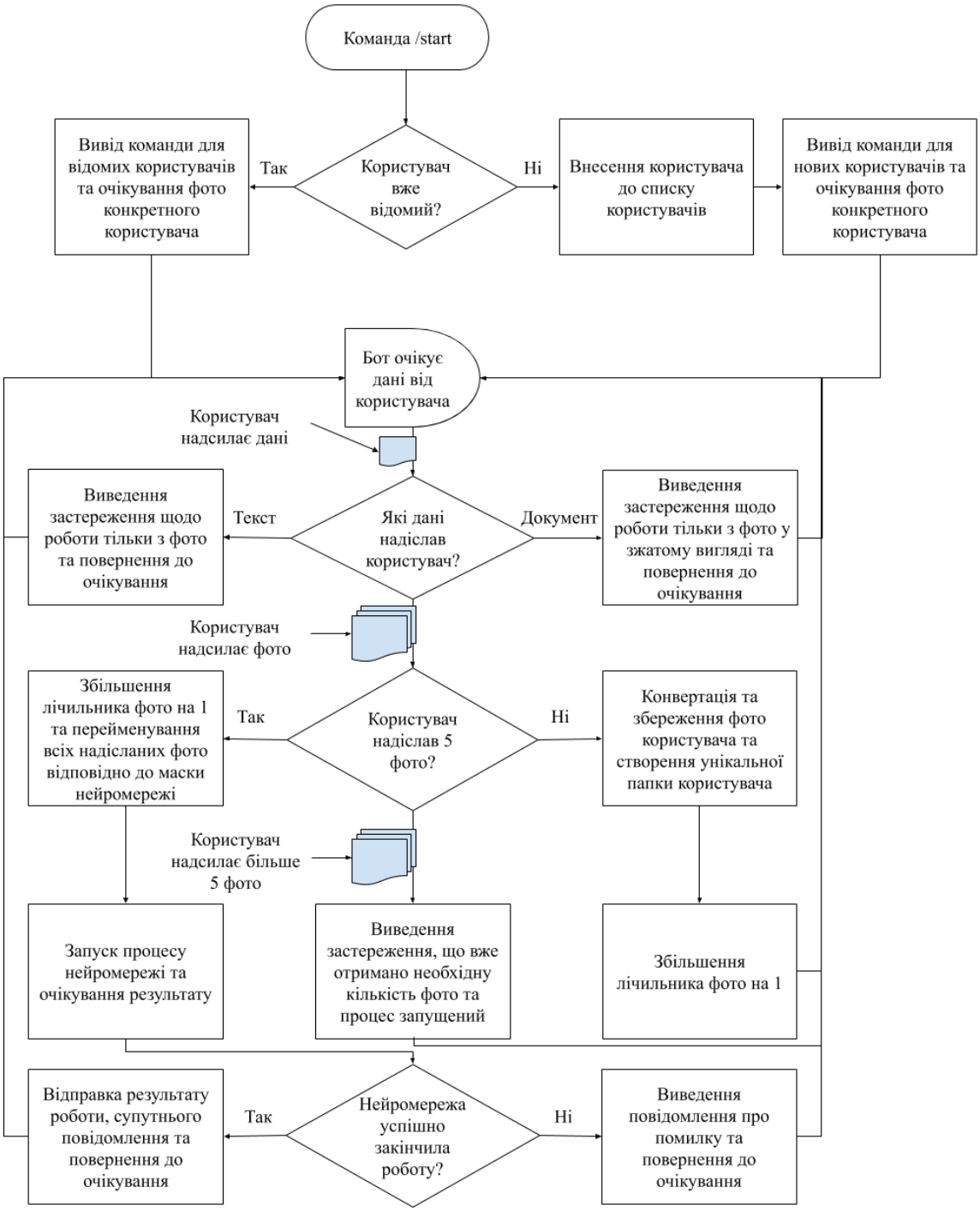

Рисунок 3.17 — Блок-схема роботи Telegram-боту з нейронною мережею

Після розробки блок-схеми була розпочата реалізація боту за допомогою мови Python та раніше згаданої бібліотеки pyTelegramBotAPI. Також був використаний попередньо підготовлений функціонуючий код боту, що відповідав «ОК» на будь-яке повідомлення.

Крім бібліотеки pyTelegramBotAPI були використані (імпортовані) наступні модулі зі стандартної бібліотеки Python:

а) модуль os – забезпечує портативний спосіб використання функціональних можливостей, що залежать від операційної системи [58];

б) модуль shutil – надає широкий функціонал по копіюванню та видаленню файлів [59];

в) модуль subprocess – дає змогу запускати нові підпроцеси з материнського процесу [60];

г) модуль PIL – бібліотека, яка додає можливості обробки зображень [44].

У Додатку А наведено код імпортування раніше згаданих бібліотек та файлів.

Враховуючи, що бот повинен запам’ятовувати користувачів та паралельно опрацьовувати їх запити, було оголошено список для користувачів, а також два словника: usersPhotos для лічильника фотографій кожного користувача та usersResults – для окремого запуску процесу нейронної мережі ботом для кожного користувача:

```
1. users = []
2. usersPhotos = {}
3. usersResults = {}
```

Надалі, за допомогою вже раніше згаданого обробника повідомлень (message_handler) бібліотеки pytTelegramBotAPI, було створено окремі функції для обробки повідомлень наступних типів:

а) команди start або help – коли користувач спілкується з ботом вперше, він завжди запускає його за допомогою команди /start. Отже, був прописаний обробник, що запускав функцію send_welcome щоразу, коли отримував команди start або help. У свою чергу функція send_welcome перевіряє наявність користувача (user_id =

message.chat.id) у списку користувачів (users = []), додає його до списку та виводить привітання. Також ця функція обробляє команди від вже відомого користувача. Нижче наведений код обробника та функції:

```
### Welcome Message
@bot.message_handler(commands=['start', 'help'])
def send_welcome(message):
    user_id = message.chat.id
    welcome_gif = open('example.gif', 'rb')
    if user_id not in users:
        users.append(user_id)
        bot.send_message(user_id, "Привіт! Я бот, що допоможе тобі прибрати зайве з твоїх фото.\
            Зараз я вмію прибирати лише сітку.\
  Надішли мені 5 фото зроблених як на гіфці нижче (у послідовному русі вправо або вліво), а решта за мною!")
        bot.send_document(user_id, welcome_gif)
    else:
        bot.send_message(user_id, "Хочешь ще допомоги з фото? Я готовий!\
  Надішли мені 5 фото зроблених як на гіфці нижче (у послідовному русі вправо або вліво), а решта за мною!")
        bot.send_document(user_id, welcome_gif)
```

б) тип повідомлення text – коли користувач надсилає будь-який текст, то бот відповідає, що не працює з текстом. Код обробника та функції нижче:

```
### Handling text messages
@bot.message_handler(content_types=["text"])
def text_warning(message):
    user_id = message.chat.id
    bot.send_message(user_id, "Чекаю твої фото \U0001F60A!\
 З текстом не працюю \U0001F609")
```

в) тип повідомлення document – коли користувач надсилає будь-який документ (і навіть фото, як документ), то бот відповідає, що не працює з таким типом даних. Код обробника та функції нижче:

```
### Handling documents
@bot.message_handler(content_types=["document"])
def document_warning(message):
    user_id = message.chat.id
    bot.send_message(user_id, "Я не працюю з іншими файлами, окрім зображень \U0001F61E.\
 Не треба присилати зображення як файл,\
 використовуй стандартний спосіб надсилання зображень у Telegram! \U0000261D")
```

г) тип повідомлення photo – зрештою, цей обробник фотографій запускає основну функцію боту user_sending_photo. Враховуючи, що в неї зашита основна логіка роботи, то на ній потрібно зупинитися більш детально.

Функція user_sending_photo складається з декількох блоків, які реалізують логіку, представлену на рис. 3.17. По-перше, функція створює директорію для кожного користувача та присвоює їй ім'я відповідно до user_id, який передає користувач з кожним своїм повідомленням. Для створення використовується модуль os:

```
1. user_id = message.chat.id
2. msg_id = message.message_id
3. prnt_dir = "source_images"
4. user_dir = str(user_id)
5. user_path = os.path.join(prnt_dir, user_dir)
6. try:
7.     os.makedirs(user_path)
8. ### Error handler if a directory already exists
9. except OSError as error:
10.     print (error)
```

Так як код виконується кожного разу, коли відбувається обробка фотографії, було додано обробник помилки (error handler), який дозволяв не переривавати процес виконання функції, якщо директорія вже існує.

Наступний блок відповідає за збереження та конвертування кожного отриманого фото. Збереження фото реалізується за допомогою методів get_file та download_file бібліотеки pyTelegramBotAPI.

```
1. photo_id = bot.get_file(message.photo[-1].file_id)
2. dwnl_photo = bot.download_file(photo_id.file_path)
3. im_src = user_path + message.photo[-1].file_id
4. with open(im_src, 'wb') as new_file:
5.     new_file.write(dwnl_photo)
```

Для підрахунку надісланих фотографій використовується лічильник, реалізований за допомогою бібліотеки usersPhotos.

```
1. if usersPhotos.get(user_id, 1) <= 5:
```

Як раніше згадувалося, таке рішення дає змогу реалізувати паралельну роботу боту для багатьох користувачів (кожен користувач отримує свою власну перемінну). Коли задається умова if, то, за допомогою метода get, викликається значення (value) для ключа в бібліотеці. Ключем завжди є унікальний user_id. Якщо ключ порожній та не включає в себе ніяких значень, то метод get повертає значення 1. Такий підхід потрібний для обробки першого фото. Надалі з наступними новими фотознімками значення для кожного користувача збільшується на 1, аж доки бот не отримає всі 5 фото:

```
1. if usersPhotos.get(user_id, 1) < 5:
2.             usersPhotos[user_id] = usersPhotos.get(user_id, 1) + 1
```

Третім блоком є процес збереження та конвертації фото за допомогою бібліотеки pillow. По-перше, було написано функцію image_converting та поміщено її до окремого файлу imgprocessing.py:

```
1. def image_converting(img):
2.     if img.size[0] > img.size[1]:
3.         basewidth = 960
4.         w_percent = (basewidth / float(img.size[0]))
5.         h_size = int((float(img.size[1]) * float(w_percent)))
6.         img_resized = img.resize((basewidth, h_size))
7.     elif img.size[0] < img.size[1]:
8.         baseheight = 960
9.         h_percent = (baseheight / float(img.size[1]))
10.          w_size = int((float(img.size[0]) * float(h_percent)))
11.          img_resized = img.resize((w_size, baseheight))
12.     else:
13.          img_resized = img.resize((basewidth, baseheight))
14.     return img_resized
```

Ця функція знаходить найбільшу сторону зображення (довжину чи ширину) та зменшує зображення відповідно до 960 пікселів по найбільшій стороні. Повний код наведено у додатку Б.

По-друге, зображення конвертується до формату png за допомогою методу save бібліотеки pillow, перейменовується, надаючи унікальне ім’я відповідно до номера повідомлення, в якому було переслано фото. Директорія очищується від проміжного фото за допомогою методу remove:

```
1. im_name = user_path + '/' + str(msg_id) + '.png'
2. image_converting(Image.open(im_src)).save(im_name)
3. os.remove(im_src)
```

Вказаний процес повторюється для кожного з 5 фото, які потрібні для роботи нейронної мережі. Коли всі 5 фото отримані та оброблені, вони перейменовуються відповідно до маски, яка потрібна для нейронної мережі 00001_Ix.png, де x – порядковий номер зображення від 0 до 4. Це можна легко зробити використавши цикл for та методи модулю os – listdir (повертає список файлів у директорі) та rename (перейменовую файли). Команда має такий вигляд:

```
1. for count, filename in enumerate(os.listdir(user_path)):
2.     os.rename(os.path.join(user_path, filename), user_path + "/00001_I" +
   str(count) + ".png")
```

Після цього відбувається запуск використаної нейронної мережі за допомогою методу run модуля subprocess:

```
1. usersResults[user_id] = subprocess.run("python3.7 test_fence.py --
   output_dir " + user_path + "/" + " --test_dataset_name " + user_path +
   "/00001", shell=True)
```

Для забезпечення паралельності виконання процесів та їх незалежності використано словник usersResults, де знову застосовано user_id як ключ (key), якому надано значення процесу нейронної мережі.

Останнім блоком функції є перевірка виконання процесу нейронної мережі. Для цього використано метод returncode – він завжди повертає 0 у разі успішного завершення процесу (без помилок), та негативне значення – у разі будь-яких помилок виконання. Завдяки унікальному ключу, збереженому у словнику usersResults, така перевірка відбувається окремо для кожного користувача. Якщо значення returncode дорівнює 0, то користувачу надсилається готове фото та декілька супровідних повідомлень. Також видаляється папка користувача разом зі всіма вхідними фото (метод rmtree модуля shutil) та видаляється ключ користувача зі словника usersPhotos разом зі всіма значеннями. Це потрібно для того, аби дати змогу користувачу знову

використати бот по завершенню процесу обробки першої партії фото. Далі наведено частину коду, що відповідає за реалізацію:

```
1. if usersResults[user_id].returncode == 0:
2.    bot.send_message(user_id, "Твоє фото готове!")
3.    bot.send_photo(user_id, open(user_path+"/00001_final.png", 'rb'))
4.    bot.send_message(user_id, "Якщо ти не задоволений результатом, то
   спробуй ще раз з новими фото. Радимо використовувати HDR
   режим.\U0001F64B")
5.     shutil.rmtree(user_path)
6.     del usersPhotos[user_id]
7. else:
8.     bot.send_message(user_id, "Щось пішло не так \U0001F635. Зроби нові
   фото та спробуй ще.")
9.     shutil.rmtree(user_path)
10.    del usersPhotos[user_id]
```

Також слід згадати, що якщо відбувається помилка виконання процесу нейронної мережі, то виконуються ті самі команди (shutil.rmtree та del usersPhotos[user_id]), за винятком того, що користувачу надсилається повідомлення про помилку замість готового фото.

Останнім елементом функції user_sending_photo є обробка помилки, якщо користувач продовжує надсилати фото, коли нейронна мережа вже оброблює попередньо надіслані фотографії. Якщо значення ключу користувача у словнику більше 5, то користувач отримує повідомлення:

```
1. bot.send_message(user_id, "Я вже маю 5  фото та працюю над ними. Більше
   не потрібно. Почекай \U0001F609")
```

Для спілкування з серверами Telegram було використано вже згаданий метод infinity_polling класу TeleBot. На цьому етапі розробка боту завершена, а повний його код можна переглянути у додатку А.

3.4.4 Розгортання Telegram-боту у Google Cloud Platform

Розгортання боту буде відбуватися по вказаним крокам у підрозділах 3.3 та 3.4.2. Отже, створено Docker-образ відповідно до наведеного нижче Dockerfile:

```
1. #Instruction to get Python 3.7.9 (Debian)
2. FROM python:3.7.9-slim
```

```
3. #Setting the working directory
4. WORKDIR /core
5. #Install environment dependencies
6. RUN apt-get update && apt-get -y install libgl1-mesa-glx libglib2.0-0
   libsm6 libxext6 libxrender-dev
7. #Copy list of python requirements
8. COPY requirements.txt .
9. #Install python dependencies
10.  RUN pip install -r requirements.txt
11.  #Copy main code
12.  COPY . .
13.  #Run bot
14.  ENTRYPOINT [ "python", "bot.py" ]
```

Команда ENTRYPOINT вказує, що в запущеному контейнеру має стартувати вищеописаний процес bot.py.

Зміст файлу requirements.txt:

```
1. # Dependencies without versions
2. pandas
3. tqdm
4. matplotlib
5. scikit-image
6. pillow
7. # Dependencies with versions
8. opencv-python==4.2.0.34
9. tensorflow==1.15
10.  pytelegrambotapi==3.7.2
```

Після виконання команди docker build було створено Docker-образ. Процес створення показаний на рис. 3.18.

```
1. docker build -t or_bot:final
```

```
yar4ick@yar4ick-e530:~/git/OR_TelegramBot$ docker build -t or_bot:final .
Sending build context to Docker daemon   1.31GB
Step 1/7 : FROM python:3.7.9-slim
 ---> 217e85391449
Step 2/7 : WORKDIR /core
 ---> Using cache
 ---> 1b2712345fcc
Step 3/7 : RUN apt-get update && apt-get -y install libgl1-mesa-glx libglib2.0-0 libsm6 libxext6 libxrender-dev
 ---> Using cache
 ---> 2acd8015c721
Step 4/7 : COPY requirements.txt .
 ---> Using cache
 ---> e2ad0ceb8bba
Step 5/7 : RUN pip install -r requirements.txt
 ---> Using cache
 ---> 1f0e9860a555
Step 6/7 : COPY . .
 ---> 15de903e74f6
Step 7/7 : ENTRYPOINT [ "python", "bot.py" ]
 ---> Running in 1374bd55c60c
Removing intermediate container 1374bd55c60c
 ---> b9884b3df56c
Successfully built b9884b3df56c
Successfully tagged or_bot:final
yar4ick@yar4ick-e530:~/git/OR_TelegramBot$
```

Рисунок 3.18 — Створення фінального Docker-образу

Після цього створений Docker-образ було завантажено до Google Container Registry (GCR), згаданий у підрозділі 3.4.2:

```
1. docker tag or_bot:final
   eu.gcr.io/obstructionremovalbot/or_bot_cloud:final
2. docker push eu.gcr.io/obstructionremovalbot/or_bot_cloud:final
```

Процес завантаження образу зображено на рис. 3.19.

```
yar4ick@yar4ick-e530:~/git/OR_TelegramBot$ docker tag or_bot:final  eu.gcr.io/obstructionremovalbot/or_bot_cloud:final
yar4ick@yar4ick-e530:~/git/OR_TelegramBot$ docker push eu.gcr.io/obstructionremovalbot/or_bot_cloud:final
The push refers to repository [eu.gcr.io/obstructionremovalbot/or_bot_cloud]
3b51c088269c: Pushed
07fb6dc6f4f9: Layer already exists
d75af5aaf7ed: Layer already exists
8f7776341ac7: Layer already exists
0fc93564e10e: Layer already exists
a125932d2011: Layer already exists
8baa92983f67: Layer already exists
37c6ab5b57f8: Layer already exists
225ef82ca30a: Layer already exists
d0fe97fa8b8c: Layer already exists
final: digest: sha256:c5a6f11824191d4a1b8be8f9a836dd9ac15b45d9a50931fd5c57e7d76fa2d1bf size: 2423
```

Рисунок 3.19 — Завантаження образу до GCR

Після цього було створено віртуальну машину з контейнером на основі даного образу. Для цього використано консольний інструмент gcloud, який вже був описаний у розділі 3.4.2:

```
1. gcloud compute instances create-with-container or_bot --machine-type=n1-
   standard-16 --boot-disk-size=15GB --boot-disk-type=pd-ssd --container-
   image=eu.gcr.io/obstructionremovalbot/or_bot_cloud:final
```

Як зображено на рис. 3.20, віртуальна машина була успішно створена та відбувся її запуск:

```
 [34] europe-west2-a
 [35] europe-west2-b
 [36] europe-west2-c
 [37] europe-west3-a
 [38] europe-west3-b
 [39] europe-west3-c
 [40] europe-west4-a
 [41] europe-west4-b
 [42] europe-west4-c
 [43] europe-west6-a
 [44] europe-west6-b
 [45] europe-west6-c
 [46] northamerica-northeast1-a
 [47] northamerica-northeast1-b
 [48] northamerica-northeast1-c
 [49] southamerica-east1-a
 [50] southamerica-east1-b
Did not print [23] options.
Too many options [73]. Enter "list" at prompt to print choices fully.
Please enter your numeric choice:  39

Created [https://www.googleapis.com/compute/v1/projects/obstructionremovalbot/zones/europe-west3-c/instances/orbot].
WARNING: Some requests generated warnings:
 - Disk size: '15 GB' is larger than image size: '10 GB'. You might need to resize the root repartition manually if the operating s
ystem does not support automatic resizing. See https://cloud.google.com/compute/docs/disks/add-persistent-disk#resize_pd for detail
s.

NAME   ZONE           MACHINE_TYPE    PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP   STATUS
orbot  europe-west3-c  n1-standard-16             10.156.0.3   35.234.82.66  RUNNING
yar4ick@yar4ick-e530:~/git/OR_TelegramBot$
```

Рисунок 3.20 — Створення віртуальної машини GCP

Віртуальна машина має наступні характеристики – 16 vCPU (віртуальних процесорів), 32 ГБ RAM та SSD накопичувач на 15 ГБ.

Після запуску віртуальної машини було проведено тестування боту, яке виявилося успішним та детально описане у підрозділі 3.5.

3.5 Тестування Telegram-боту

Процес тестування боту відбувався за допомогою застосунка Telegram версії 7.2.1 (2139) на мобільному телефоні на базі Android OS версії 6.0.1. Також додатково використовувалися відповідні застосунки Telegram версії 2.4.7 для Windows 10 Pro (версія збірки 18363.1198) та Ubuntu 20.04.1. Всі наведені нижче знімки екрану були зроблені на комп’ютері для зручності, проте всі тести відбувалися на телефоні, адже це основна цільова платформа.

Після запуску боту у хмарному сервісі Google Cloud Platform був запущений додаток Telegram та розпочато спілкування з ботом за допомогою команди start. Як вказано на рис. 3.21, бот надіслав привітальне повідомлення.

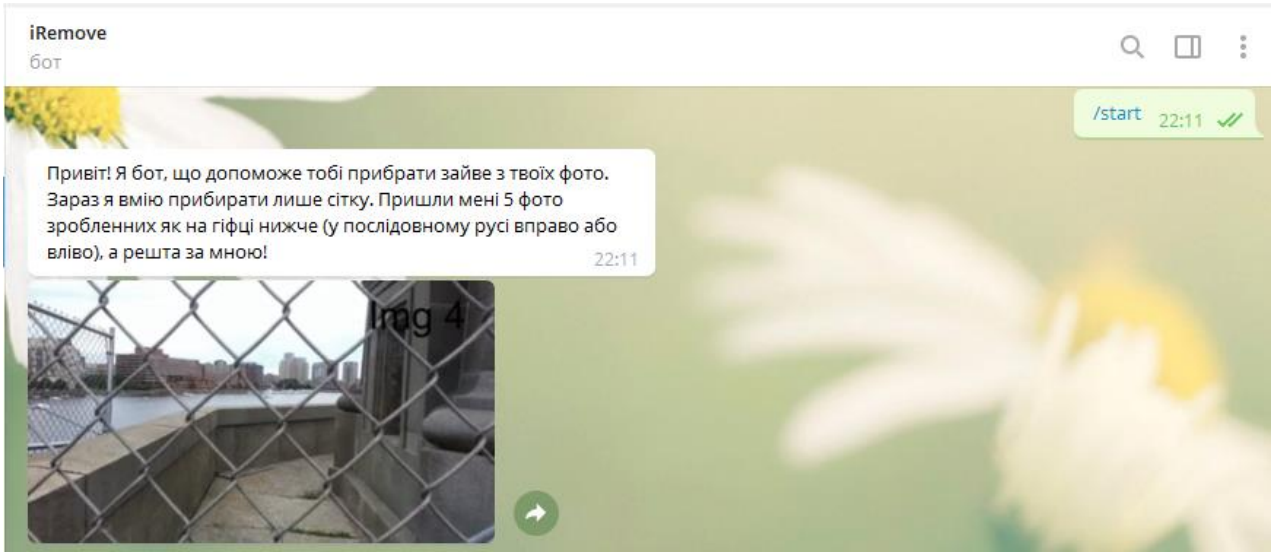

Рисунок 3.21 — Початок спілкування з ботом

Боту було надіслано 5 тестових фото, що знаходяться в репозиторії використаної нейронної мережі ObstructionRemoval. Як зображено на рис. 3.22, бот успішно отримав фото та повідомив про це користувача.

Рисунок 3.22 — Бот отримав фото та повідомив користувача

На рис. 3.23 зображено, що бот успішно виконав свою функцію та прибрав сітку з фото:

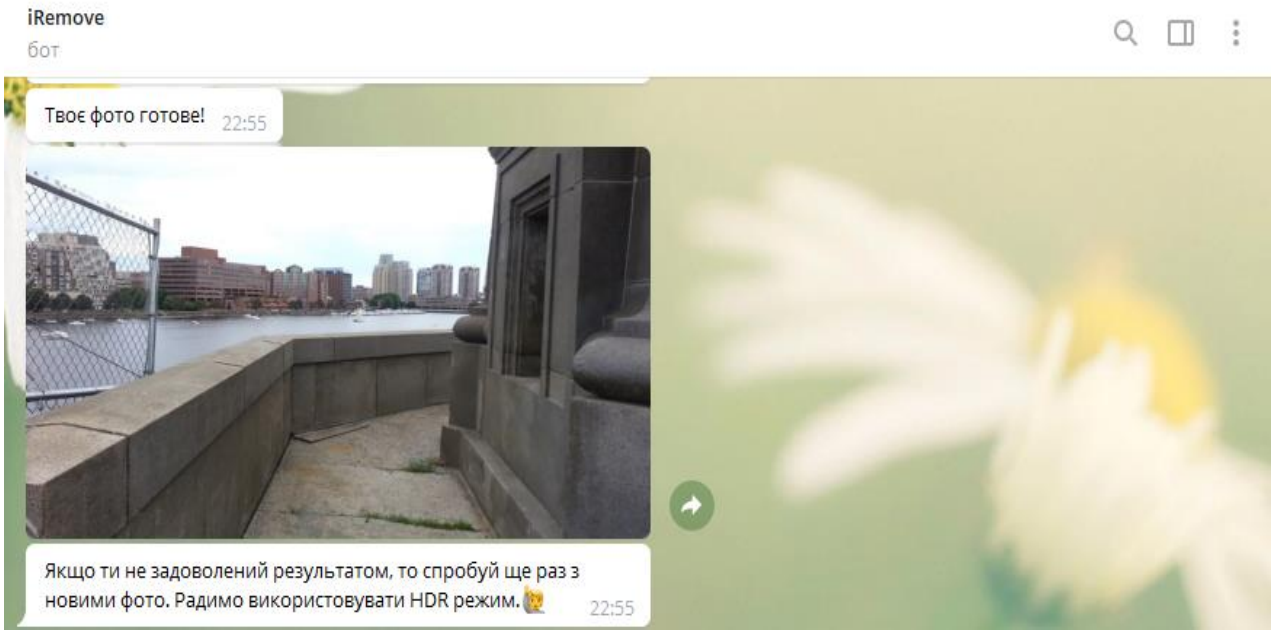

Рисунок 3.23 — Бот повертає фінальне зображення

Було перевірено, чи сприймає бот інші типи файлів, та відправлено йому текст і документ. Як і було заплановано, бот не обробив ці типи даних та надіслав користувачу відповіді щодо цього, як це зображено на рис. 3.24.

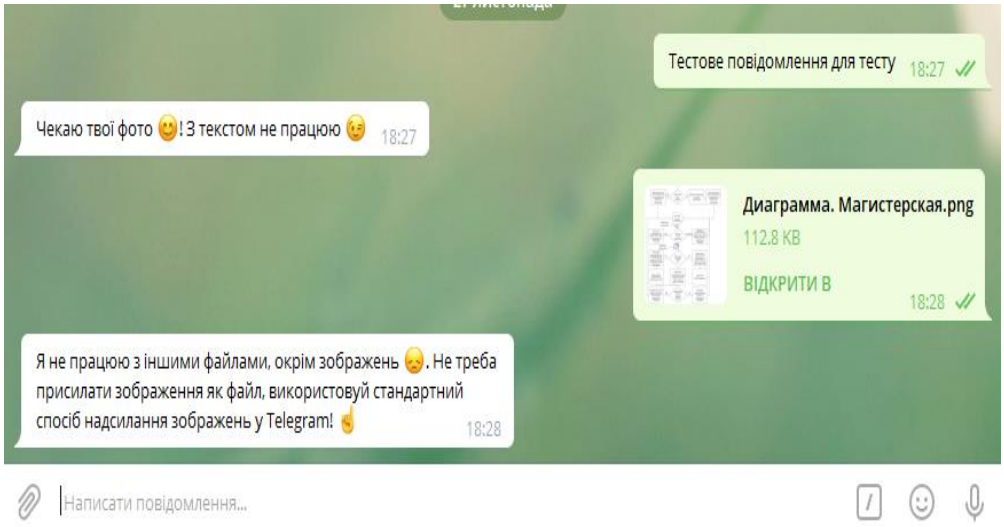

Рисунок 3.24 — Тест за допомогою тексту та документу

Було перевірено, як працює бот з реальними даними. Для цього були зняті 5 фото сітки у місті. Фотографії були відправлені до боту з мобільного пристрою. Тест був успішним, а його результат зображено на рис. 3.25.

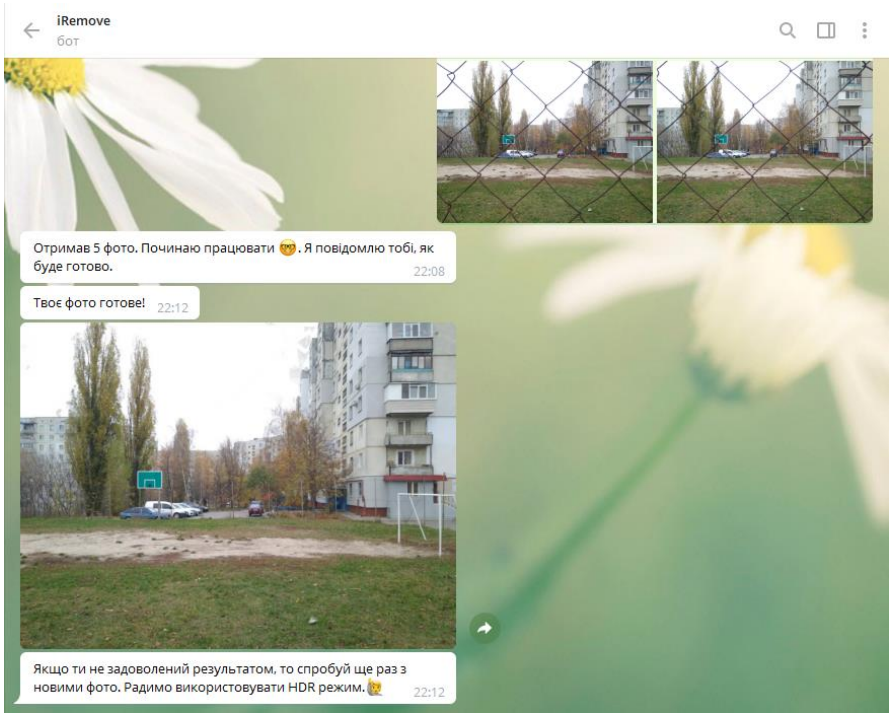

Рисунок 3.25 — Результат роботи з реальними фотографіями

Наступний тест – перевірка паралельної роботи з декількома користувачами. Для цього знадобився ще один аккаунт Telegram та ще один

набір фото. З інтервалом в 15 секунд було надіслано різні фото з 2 різних аккаунтів та отримано успішні результати. Слід зазначити, що час обробки збільшився із приблизно 190 секунд до 260 секунд для першого аккаунту, та 272 секунд для другого. Результати зображені на рис. 3.26 та 3.27.

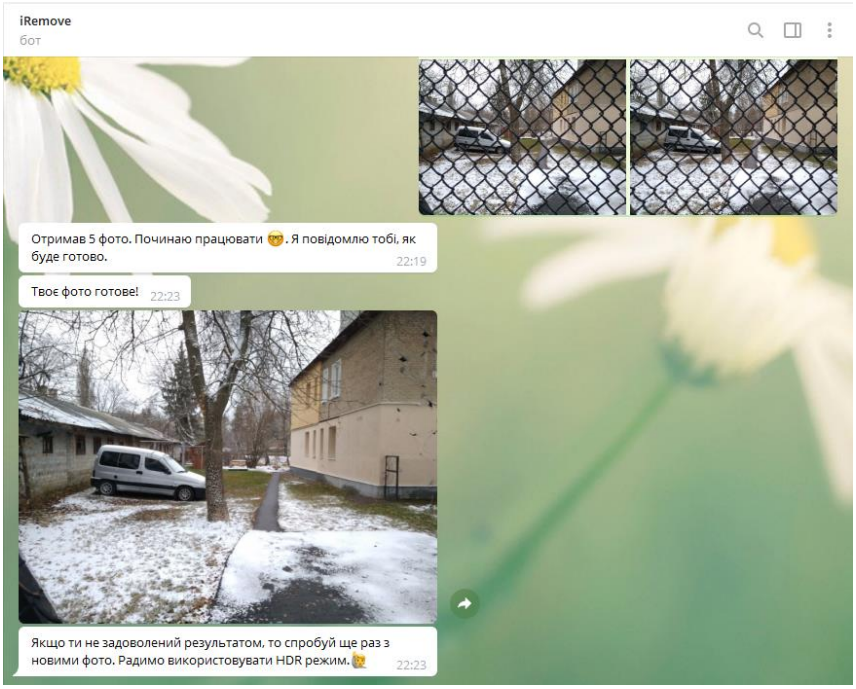

Рисунок 3.26 — Паралельна обробка, аккаунт 1

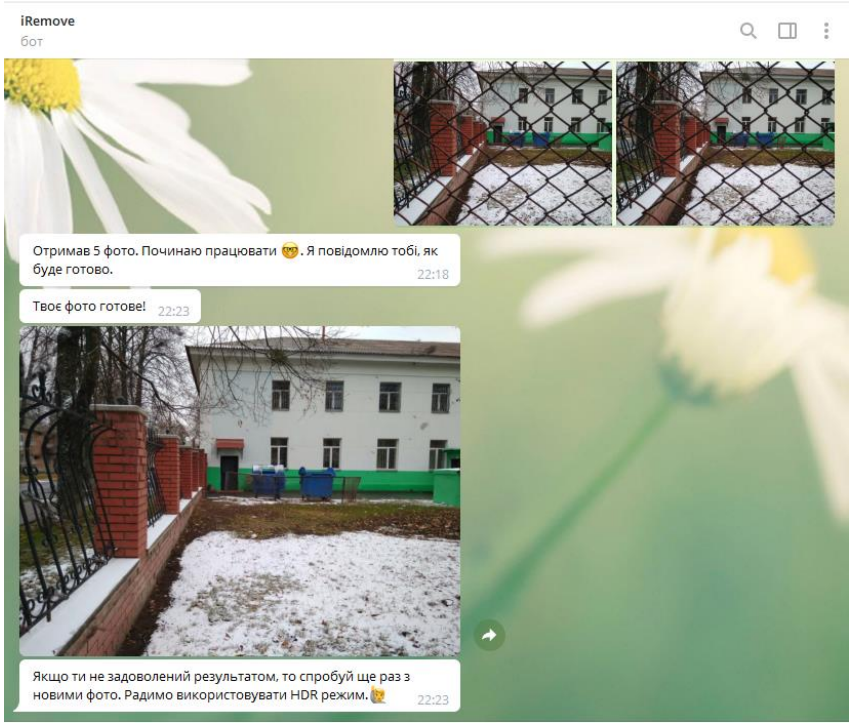

Рисунок 3.27 — Паралельна обробка, аккаунт 2

Протестувавши функціонал боту та зрозумівши, що все працює відповідно до заданої логіки, було вирішено випробувати бота на різних зразках сітки. Бот успішно виконав своє завдання у більшості випадків, проте деякі сітки прибрати йому не вдалося. Наприклад, на рис. 3.28 зображена одна з таких сіток. Було зроблено висновок, що недостатньо контрастні сітки створюють проблему для нейронної мережі, тому є доцільним робити фотографії у режимі HDR, що підвищує контрастність зображення.

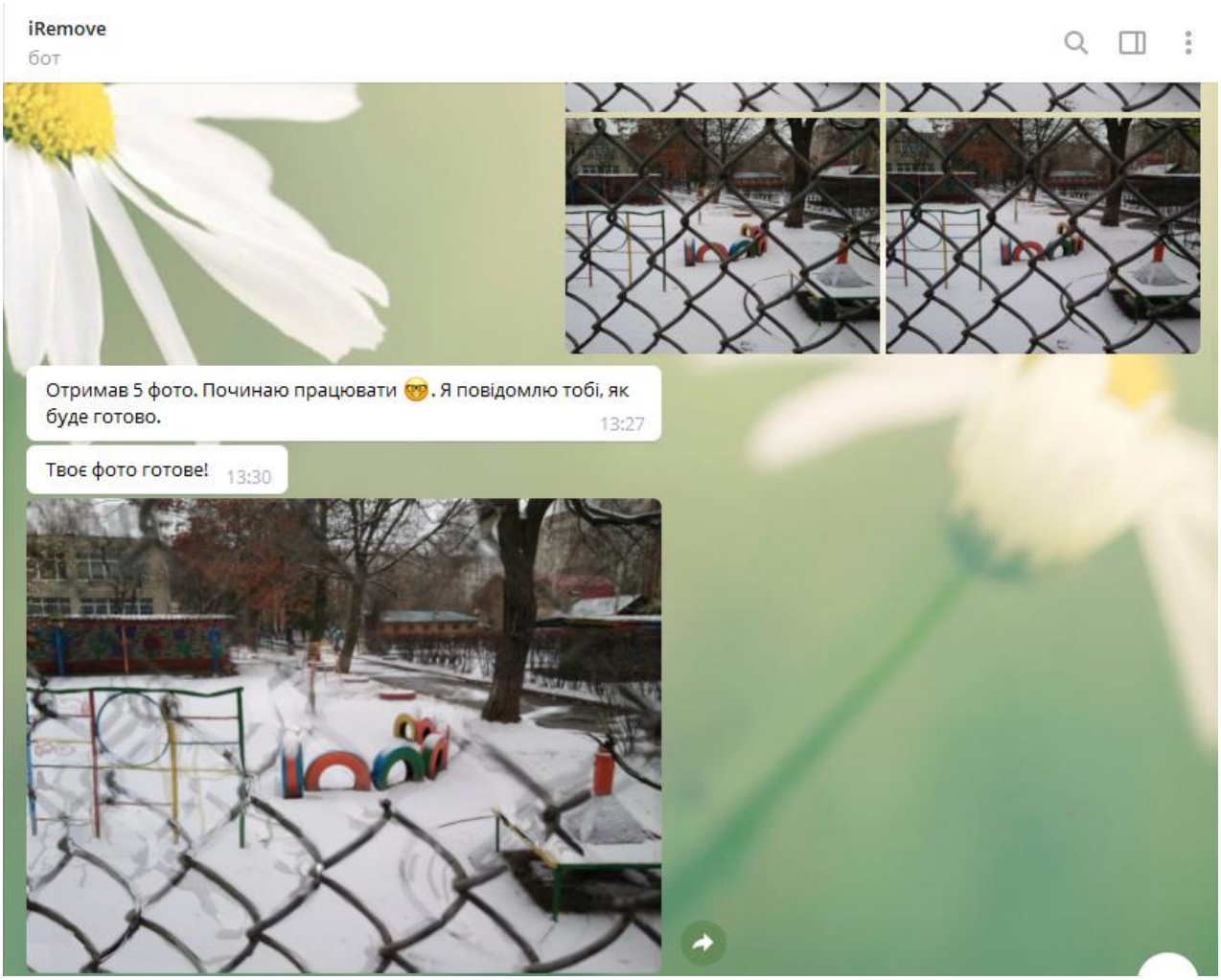

Рисунок 3.28 — Приклад невдалої обробки фотографій

Незважаючи на цей факт, можна підсумувати, що «польові випробування» бот пройшов успішно, адже зміг прибрати сітку у 75% випадків. Таким чином, подальше навчання нейронної мережі та її вдосконалення дасть змогу підвищити цей результат до 95-99%.

ВИСНОВКИ

Щороку проблема обробки цифрових фотознімків стає ще більш актуальною, адже кількість електронних пристроїв з функцією фотозйомки невпинно збільшується. У подоланні викликів цієї проблеми мають допомогти системи та алгоритми видалення шуму з фотографій.

У даній роботі були досліджені праці науковців, які вже працювали над рішенням цієї проблем та були наведені результати їх роботи. За результатом проведеного дослідження було обрано найсучасніший алгоритм по видаленню шуму на фотографіях. Його реалізація передбачає видалення оклюзій, тобто небажаних елементів на фотознімках, які перешкоджають спогляданню основної сцени зображення. Цей процес відбувається за допомогою багатокадрового (multi-frame) підходу, який об’єднує в собі переваги як оптимізаційних, так і навчальних методів. Запропонований алгоритм чергує кроки оцінки щільності руху (dense motion) з кроками реконструкції шару з оклюзією та цільовим зображенням за допомогою підходу «від не точного до точного». Реалізований за допомогою нейронних мереж алгоритм був використаний як модуль розробленої системи видалення шуму (артефактів) з фотознімків.

У даній роботі був розроблений та програмно реалізований інший модуль системи – модуль взаємодії з користувачем. Способом його реалізації було обрано Telegram-бота. Конкретним об’єктом роботи даної системи стало видалення решітчастих огорож на фотознімках.

Програмно представлена у даній роботі система була реалізована за допомогою таких інструментів, як-от мова програмування Python, додаток для обміну повідомленнями Telegram, система контролю версій git, сервіс спільної розробки програмного забезпечення GitHub, система контеризації програмного забезпечення Docker та хмарний сервіс Google Cloud Platform.

Проведені тести показали чудову роботу системи з точки зору не тільки результатів, але й зручності для кінцевого користувача. Без сумніву, у

подальшому така система може бути використана для видалення інших типів оклюзій на фотознімках. Враховуючи, що в цій системі були використані доробки, розміщені на GitHub для загального користування, розроблена система також відкрита для широкого загалу в аккаунті GitHub.

Отримані результати свідчать, що наявні алгоритми обробки зображень будуть удосконалені подальшими дослідниками, а розроблена система може стати зразком використання таких алгоритмів у практичних цілях для широкого кола користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. • Smartphone users 2020 | Statista [Electronic resource]. – Electronic data. – Access mode : https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. – 28.11.2020.
2. Mahajan M. M. Image inpainting a novel technique to remove object from an image / M. M. Mahajan // International Journal of Scientific Research in Science, Engineering and Technology. – 2016. – Vol. 2, № 2. – P. 17–20.
3. Kulkarni S. Removal of unwanted objects from images using statistics / S. Kulkarni // ICTACT Journal on Image and Video Processing. – 2018. – Vol. 9, № 2. – P. 1887–1893.
4. Liu Y.-L. Learning to see through obstructions / Y.-L. Liu, W.-S. Lai, M.-H. Yang[et al.] // 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). – IEEE, 2020. – P. 14203–14212.
5. Gai K. Blind separation of superimposed images with unknown motions / K. Gai, Z. Shi, C. Zhang // 2009 IEEE Conference on Computer Vision and Pattern Recognition. – IEEE, 2009. – P. 1881–1888.
6. Xue T. A computational approach for obstruction-free photography / T. Xue, M. Rubinstein, C. Liu, W. T. Freeman // ACM Transactions on Graphics. – 2015. – Vol. 34, № 4. – P. 1–11.
7. Li Y. Exploiting reflection change for automatic reflection removal / Y. Li, M. S. Brown // 2013 IEEE International Conference on Computer Vision. – IEEE, 2013. – P. 2432–2439.
8. Nandoriya A. Video reflection removal through spatio-temporal optimization / A. Nandoriya, M. Elgharib, C. Kim[et al.] // 2017 IEEE International Conference on Computer Vision (ICCV). – IEEE, 2017. – P. 2430–2438.
9. Guo X. Robust separation of reflection from multiple images / X. Guo, X. Cao, Y. Ma // 2014 IEEE Conference on Computer Vision and Pattern Recognition. – IEEE, 2014. – P. 2195–2202.
10. Fan Q. A generic deep architecture for single image reflection removal and image

smoothing / Q. Fan, J. Yang, G. Hua[et al.] // 2017 IEEE International Conference on Computer Vision (ICCV). – IEEE, 2017. – P. 3258–3267.

11. Wei K. Single image reflection removal exploiting misaligned training data and network enhancements / K. Wei, J. Yang, Y. Fu[et al.] // 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). – IEEE, 2019. – P. 8170–8179.
12. Jin M. Learning to see through reflections / M. Jin, S. Susstrunk, P. Favaro // 2018 IEEE International Conference on Computational Photography (ICCP). – IEEE, 2018. – P. 1–12.
13. Yang J. Seeing deeply and bidirectionally: a deep learning approach for single image reflection removal / J. Yang, D. Gong, L. Liu, Q. Shi // European Conference on Computer Vision. – Springer, 2018. – Vol. 11207 LNCS. – P. 675–691.
14. Alayrac J.-B. The visual centrifuge: model-free layered video representations / J.-B. Alayrac, J. Carreira, A. Zisserman // 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). – IEEE, 2019. – P. 2452–2461.
15. Du C. Accurate and efficient video de-fencing using convolutional neural networks and temporal information / C. Du, B. Kang, Z. Xu[et al.] // 2018 IEEE International Conference on Multimedia and Expo (ICME). – IEEE, 2018. – P. 1–6.
16. Jonna S. Deep learning based fence segmentation and removal from an image using a video sequence / S. Jonna, K. K. Nakka, R. R. Sahay // European Conference on Computer Vision. – Springer, 2016. – Vol. 9915. – P. 836–851.
17. Yi R. Automatic fence segmentation in videos of dynamic scenes / R. Yi, J. Wang, P. Tan // 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – IEEE, 2016. – P. 705–713.
18. Park M. Image de-fencing revisited / M. Park, K. Brocklehurst, R. T. Collins, Y. Liu // Computer Vision – ACCV 2010. – Springer, 2011. – Vol. LNCS 6495. –

P. 422–434.

19. YiChang Shih Reflection removal using ghosting cues / YiChang Shih, D. Krishnan, F. Durand, W. T. Freeman // 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – IEEE, 2015. – P. 3193–3201.
20. Scale-invariant feature transform - Wikipedia [Electronic resource]. – Electronic data. – Access mode : https://en.wikipedia.org/wiki/Scale-invariant_feature_transform. – 29.11.2020.
21. Liu C. SIFT flow: dense correspondence across different scenes / C. Liu, J. Yuen, A. Torralba[et al.] // European Conference on Computer Vision. – Springer, 2008. – Vol. 1. – P. 28–42.
22. Szeliski R. Layer extraction from multiple images containing reflections and transparency / R. Szeliski, S. Avidan, P. Anandan // 2000 IEEE Conference on Computer Vision and Pattern Recognition. – IEEE, 2000. – Vol. 1. – P. 246–253.
23. alex04072000/ObstructionRemoval: [CVPR 2020] Learning to See Through Obstructions [Electronic resource]. – Electronic data. – Access mode : https://github.com/alex04072000/ObstructionRemoval. – 21.11.2020.
24. Sun D. PWC-net: cnns for optical flow using pyramid, warping, and cost volume / D. Sun, X. Yang, M.-Y. Liu, J. Kautz // 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. – IEEE, 2018. – Vol. D. – P. 8934–8943.
25. Telegram Bot Platform [Electronic resource]. – Electronic data. – Access mode : https://telegram.org/blog/bot-revolution. – 02.12.2020.
26. Статистика Телеграм в Украине - BOT: Все о Телеграм-маркетинге. Украина [Электронный ресурс]. – Электронные данные. – Режим доступа : https://tlgrm.in.ua/vsjo-o-telegram/stati/statistika-telegram-v-ukraine/. – 27.11.2020.
27. Cockburn A. Using both incremental and iterative development / A. Cockburn // CrossTalk. – 2008. – Vol. 21, № 5. – P. 27–30.
28. Agile Development: Iterative and Incremental [Electronic resource]. – Electronic

data. – Access mode : https://www.visual-paradigm.com/scrum/agile-development-iterative-and-incremental/. – 19.11.2020.

29. Larman C. Iterative and incremental developments. a brief history / C. Larman, V. R. Basili // Computer. – 2003. – Vol. 36, № 6. – P. 47–56.
30. Добро пожаловать в Colaboratory! - Colaboratory [Электронный ресурс]. – Электронные данные. – Режим доступа : https://colab.research.google.com/notebooks/intro.ipynb. – 20.11.2020.
31. Best language for Machine Learning: Which Programming Language to Learn | Springboard Blog [Electronic resource]. – Electronic data. – Access mode : https://in.springboard.com/blog/best-language-for-machine-learning/. – 02.12.2020.
32. Matthes E. Python crash course; a hands-on, project-based introduction to programming / E. Matthes. – San Francisco : No Starch Press, 2019. – 506 p.
33. FocalFossa/ReleaseNotes - Ubuntu Wiki [Electronic resource]. – Electronic data. – Access mode : https://wiki.ubuntu.com/FocalFossa/ReleaseNotes#Python3_by_default. – 21.11.2020.
34. What Is Pip? A Guide for New Pythonistas – Real Python [Electronic resource]. – Electronic data. – Access mode : https://realpython.com/what-is-pip/. – 21.11.2020.
35. Home | The Mesa 3D Graphics Library [Electronic resource]. – Electronic data. – Access mode : https://mesa3d.org/. – 21.11.2020.
36. Git - Як зробити внесок до проекту [Електроний ресурс]. – Електроні дані. – Режим доступу : https://git-scm.com/book/uk/v2/GitHub-Як-зробити-внесок-до-проекту. – 21.11.2020.
37. Git on the commandline — Don’t be afraid to commit 0.3 documentation [Electronic resource]. – Electronic data. – Access mode : https://dont-be-afraid-to-commit.readthedocs.io/en/latest/git/commandlinegit.html. – 21.11.2020.

38. Generating a new SSH key and adding it to the ssh-agent - GitHub Docs [Electronic resource]. – Electronic data. – Access mode : https://docs.github.com/en/free-pro-team@latest/github/authenticating-to-github/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent. – 21.11.2020.
39. pandas documentation — pandas 1.1.4 documentation [Electronic resource]. – Electronic data. – Access mode : https://pandas.pydata.org/docs/index.html. – 21.11.2020.
40. tqdm documentation [Electronic resource]. – Electronic data. – Access mode : https://tqdm.github.io/. – 21.11.2020.
41. Matplotlib: Python plotting — Matplotlib 3.3.3 documentation [Electronic resource]. – Electronic data. – Access mode : https://matplotlib.org/index.html. – 21.11.2020.
42. scikit-image: Image processing in Python — scikit-image [Electronic resource]. – Electronic data. – Access mode : https://scikit-image.org/. – 21.11.2020.
43. Pylint User Manual — Pylint 2.6.1-dev1 documentation [Electronic resource]. – Electronic data. – Access mode : http://pylint.pycqa.org/en/latest/. – 21.11.2020.
44. Pillow · PyPI [Electronic resource]. – Electronic data. – Access mode : https://pypi.org/project/Pillow/. – 21.11.2020.
45. OpenCV: Introduction [Electronic resource]. – Electronic data. – Access mode : https://docs.opencv.org/4.5.0/d1/dfb/intro.html. – 21.11.2020.
46. TensorFlow Core | Machine Learning for Beginners and Experts [Electronic resource]. – Electronic data. – Access mode : https://www.tensorflow.org/overview. – 21.11.2020.
47. philferriere/tfoptflow: Optical Flow Prediction with TensorFlow. Implements “PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume,” by Deqing Sun et al. (CVPR 2018) [Electronic resource]. – Electronic data. – Access mode : https://github.com/philferriere/tfoptflow. – 21.11.2020.
48. wkentaro/gdown: Download a large file from Google Drive (curl/wget fails

because of the security notice). [Electronic resource]. – Electronic data. – Access mode : https://github.com/wkentaro/gdown. – 21.11.2020.

49. What is Docker? | Opensource.com [Electronic resource]. – Electronic data. – Access mode : https://opensource.com/resources/what-docker. – 22.11.2020.
50. What is a Container? | App Containerization | Docker [Electronic resource]. – Electronic data. – Access mode : https://www.docker.com/resources/what-container. – 22.11.2020.
51. Dockerfile reference | Docker Documentation [Electronic resource]. – Electronic data. – Access mode : https://docs.docker.com/engine/reference/builder/. – 22.11.2020.
52. python - Docker Hub [Electronic resource]. – Electronic data. – Access mode : https://hub.docker.com/_/python. – 22.11.2020.
53. Telegram Bot API [Electronic resource]. – Electronic data. – Access mode : https://core.telegram.org/bots/api. – 22.11.2020.
54. Урок 1: Введение, простой echo-бот | Пишем ботов для Telegram на языке Python [Электронный ресурс]. – Электронные данные. – Режим доступа : https://mastergroosha.github.io/telegram-tutorial/docs/lesson_01/. – 22.11.2020.
55. eternnoir/pyTelegramBotAPI: Python Telegram bot api. [Electronic resource]. – Electronic data. – Access mode : https://github.com/eternnoir/pyTelegramBotAPI. – 22.11.2020.
56. Home – Google Cloud Platform [Electronic resource]. – Electronic data. – Access mode : https://console.cloud.google.com/. – 25.11.2020.
57. Kubernetes [Electronic resource]. – Electronic data. – Access mode : https://kubernetes.io/uk/. – 25.11.2020.
58. os — Miscellaneous operating system interfaces — Python 3.7.9 documentation [Electronic resource]. – Electronic data. – Access mode : https://docs.python.org/3.7/library/os.html. – 27.11.2020.
59. shutil — High-level file operations — Python 3.7.9 documentation [Electronic

resource]. – Electronic data. – Access mode : https://docs.python.org/3.7/library/shutil.html. – 27.11.2020.

60. subprocess — Subprocess management — Python 3.9.1rc1 documentation [Electronic resource]. – Electronic data. – Access mode : https://docs.python.org/3/library/subprocess.html. – 27.11.2020.

ДОДАТОК А

```
1. import config
2. from imgprocessing import image_converting
3. import os
4. import subprocess
5. import shutil
6. import PIL
7. from PIL import Image
8. import telebot
9. from telebot import types
10.
11.  bot = telebot.TeleBot(config.token)
12.  users = []
13.  usersPhotos = {}
14.  usersResults = {}
15.
16.  ### Welcome Message
17.  @bot.message_handler(commands=['start', 'help'])
18.  def send_welcome(message):
19.      user_id = message.chat.id
20.      welcome_gif = open('example.gif', 'rb')
21.      if user_id not in users:
22.          users.append(user_id)
23.          bot.send_message(user_id, "Привіт! Я бот, що допоможе тобі
    прибрати зайве з твоїх фото.\
24.              Зараз я вмію прибирати лише сітку.\
25.    Надішли мені 5 фото зроблених як на гіфці нижче (у послідовному русі
    вправо або вліво), а решта за мною!")
26.          bot.send_document(user_id, welcome_gif)
27.      else:
28.          bot.send_message(user_id, "Хочешь ще допомоги з фото? Я
    готовий!\
29.    Надішли мені 5 фото зроблених як на гіфці нижче (у послідовному русі
    вправо або вліво), а решта за мною!")
30.          bot.send_document(user_id, welcome_gif)
31.
32.  ### Handling text messages
33.  @bot.message_handler(content_types=["text"])
34.  def text_warning(message):
35.      user_id = message.chat.id
36.      bot.send_message(user_id, "Чекаю твої фото \U0001F60A!\
37.    З текстом не працюю \U0001F609")
38.
39.  ### Handling documents
40.  @bot.message_handler(content_types=["document"])
41.  def document_warning(message):
42.      user_id = message.chat.id
43.      bot.send_message(user_id, "Я не працюю з іншими файлами, окрім
    зображень \U0001F61E.\
44.    Не треба присилати зображення як файл,\
45.    використовуй стандартний спосіб надсилання зображень у Telegram!
    \U0000261D")
46.
47.  ### Bot downloads 5 photos, run a subprocess and sends a final image
    back to the user
48.  @bot.message_handler(content_types=["photo"])
49.  def user_sending_photo(message):
50.      user_id = message.chat.id
51.      msg_id = message.message_id
52.      ### Creating a directory for each user
```

```
53.      prnt_dir = "source_images"
54.      user_dir = str(user_id)
55.      user_path = os.path.join(prnt_dir, user_dir)
56.      ### Making a directory for each user
57.      try:
58.          os.makedirs(user_path)
59.      ### Error handler if a directory already exists
60.      except OSError as error:
61.          print (error)
62.      ### Creating a variable for each user based on user ID using
   dictionary usersPhotos
63.      if usersPhotos.get(user_id, 1) <= 5:
64.          ### Saving photo as a source file to a user folder
65.          photo_id = bot.get_file(message.photo[-1].file_id)
66.          dwnl_photo = bot.download_file(photo_id.file_path)
67.          im_src = user_path + message.photo[-1].file_id
68.          with open(im_src, 'wb') as new_file:
69.              new_file.write(dwnl_photo)
70.          ### Creating a path to a photo
71.          im_name = user_path + '/' + str(msg_id) + '.png'
72.          ### Using image_converting function from imgprocessing.py to
   resize image
73.          image_converting(Image.open(im_src)).save(im_name)
74.          ### Removing initial source image file
75.          os.remove(im_src)
76.          ### Counter for 5 images received
77.          if usersPhotos.get(user_id, 1) < 5:
78.              usersPhotos[user_id] = usersPhotos.get(user_id, 1) + 1
79.          else:
80.              ### The main process starts when bot receives 5 photos
81.              usersPhotos[user_id] += 1
82.              bot.send_message(user_id, "Отримав 5 фото. Починаю
   працювати \U0001F913. Я повідомлю тобі, як буде готово.")
83.              ### Loop to rename photos to the format needed for the
   obstruction removal ANN
84.              for count, filename in enumerate(os.listdir(user_path)):
85.                  os.rename(os.path.join(user_path, filename), user_path
   + "/00001_I" + str(count) + ".png")
86.              ### Starting ANN as a subprocess
87.              usersResults[user_id] = subprocess.run("python3.7
   test_fence.py --output_dir " + user_path + "/" + " --test_dataset_name "
   + user_path + "/00001", shell=True)
88.              ### Check the subprocess result: if it is 0 than subprocess
   was executed succesfuly
89.              if usersResults[user_id].returncode == 0:
90.                  bot.send_message(user_id, "Твоє фото готове!")
91.                  bot.send_photo(user_id,
   open(user_path+"/00001_final.png", 'rb'))
92.                  bot.send_message(user_id, "Якщо ти не задоволений
   результатом, то спробуй ще раз з новими фото. Радимо використовувати HDR
   режим.\U0001F64B")
93.                  ### Deleting all items in the users folder since they
   are no more needed
94.                  shutil.rmtree(user_path)
95.                  ### Deleting user key from the dictionary in order to
   allow to run a subprocess again
96.                  del usersPhotos[user_id]
97.              else:
98.                  bot.send_message(user_id, "Щось пішло не так
   \U0001F635. Зроби нові фото та спробуй ще.")
99.                  ### Deleting all items in the users folder since they
   are no more needed
```

```
100.                 shutil.rmtree(user_path)
101.                 ### Deleting user key from the dictionary in order to
   allow to run a subprocess again
102.                 del usersPhotos[user_id]
103.     else:
104.         bot.send_message(user_id, "Я вже маю 5  фото та працюю над
   ними. Більше не потрібно. Почекай \U0001F609")
105.
106. if __name__ == '__main__':
107.     bot.infinity_polling()
```

ДОДАТОК Б

```
1. from PIL import Image
2.
3. def image_converting(img):
4.     if img.size[0] > img.size[1]:
5.         basewidth = 960
6.         w_percent = (basewidth / float(img.size[0]))
7.         h_size = int((float(img.size[1]) * float(w_percent)))
8.         img_resized = img.resize((basewidth, h_size))
9.     elif img.size[0] < img.size[1]:
10.          baseheight = 960
11.          h_percent = (baseheight / float(img.size[1]))
12.          w_size = int((float(img.size[0]) * float(h_percent)))
13.          img_resized = img.resize((w_size, baseheight))
14.      else:
15.          img_resized = img.resize((basewidth, baseheight))
16.      return img_resized
```