Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних систем управління

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інтелектуальна автоматизована система розпізнавання об'єктів на зображеннях земної поверхні»

Виконав: студент 2 курсу, групи ЗАКІТ-21 м
спеціальності 151 – Автоматизація та
комп'ютерно-інтегровані технології

Дмитрій Сідак

Керівник: к.т.н., доц. каф. АІІТ
Володимир Севастьянов

«12» грудня 2022 р.

Опонент: к.т.н., доц. каф. САІТ
Олексій Козачко

«18» грудня 2022 р.

Допущено до захисту
Зав. кафедри КСУ
В'ячеслав Ковтун
«14» грудня 2022

Вінниця ВНТУ – 2022 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних систем управління
Рівень вищої освіти другий (магістерський)
Галузь знань – 15 – Автоматизація та приладобудування
Спеціальність – 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітньо - професійна програма – Інформаційні системи і Інтернет речей

ЗАТВЕРДЖУЮ
Завідувач кафедри КСУ

В'ячеслав КОВТУН

"03" жовтня 2022 року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ
студенту Сідаку Дмитрію Анатолійовичу

1. Тема роботи: «Інтелектуальна автоматизована система розпізнавання об'єктів на зображеннях земної поверхні»
керівник роботи Севастьянов Володимир Миколайович
затверджені наказом ВНТУ від "14" вересня 2022 року №203
2. Термін подання студентом роботи "13" грудня 2022 року
3. Вихідні дані до роботи: Застосунок для розпізнавання об'єктів на зображеннях земної поверхні за допомогою нейронних мереж: виділений чотирма координатами участок земної поверхні,збережений оброблений файл
4. Зміст текстової частини: Аналіз проблем та постановка дослідницьких задач, вибір програмних засобів для розробки системи розпізнавання образів, розробка системи розпізнавання зображень аерофотозйомки, розгорнуте тестування системи розпізнавання зображень аерофотозйомки
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень)
Приклад використання Jupyter Notebook, Приклад використання моделі ResNet для задачі аналізу непрозорої облачності над Пекіном, Семантична сегментація пікселів зображень за допомогою згорткової нейромережі, Архітектура моделі U-Net, Головне вікно Earth Engine Code, Вибір точки у Earth Engine Code, Вікно Inspector, Прогнозування непрозорих хмарних областей

6.Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1	Ковтун В.В., к.т.н., доц.каф. КСУ		
2	Ковтун В.В., к.т.н., доц.каф. КСУ		
3	Ковтун В.В., к.т.н., доц.каф. КСУ		
4	Ковтун В.В., к.т.н., доц.каф. КСУ		

7.Дата видачі завданняя "3" жовтня 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Дослідження предметної області	6.10.2022	20.10.2022	
2	Дослідження існуючих архітектур	22.10.2022	14.11.2022	
3	Програмна реалізація додатку	15.11.2022	30.11.2022	
4	Попередній захист			
5	Остаточний захист			

Студент (підпис) **Сідак Д.А.** (*Ім'я ПРІЗВИЩЕ*)

Керівник роботи (підпис) **Севастьянов В.М.** (*Ім'я ПРІЗВИЩЕ*)

АНОТАЦІЯ

УДК 621.374.415

Сідак Д.А. Інтелектуальний експериментальний застосунок для розпізнавання об’єктів на зображеннях земної поверхні. Магістерська кваліфікаційна робота за спеціальністю 151 – Автоматизація та комп’ютерно-інтегровані технології, освітня програма – Інформаційні системи та Інтернет речей. Вінниця: ВНТУ, 2022.

У магістерській кваліфікаційній роботі розроблено застосунок для розпізнавання об’єктів земної поверхні на зображеннях за допомогою нейронних мереж. У оглядово-аналітичній частині роботи досліджено існуючі методи та підходи для розпізнавання об’єктів, проаналізовано існуючі архітектури нейронних мереж. У теоретично-методичній частині обрано архітектуру для розробки. У практичній частині розроблено застосунок для розпізнавання об’єктів земної поверхні на зображеннях за допомогою нейронних мереж.

Ключові слова: нейронна мережа, комп’ютерний зір, jupyter notebook, pyton.

ANNOTATION

UDC 621.374.415

Sidak D.A. An intelligent experimental approach for object recognition in images of the Earth's surface. Master's qualification work on specialty 151 - Automation and computer-integrated technologies, educational program - Information systems and the Internet of Things. Vinnytsia: VNTU, 2022. 1

In the master's thesis, an application was developed for the recognition of objects on the earth's surface in images using neural networks. In the review and analytical part of the work, the existing methods and approaches for object recognition were investigated, and the existing architectures of neural networks were analyzed. In the theoretical and methodological part, the architecture for development is chosen. In the practical part, an application was developed for the recognition of objects on the Earth's surface in images using neural networks.

Keywords: neural network, computer vision, jupyter notebook, python.

Відгук
керівника магістерської кваліфікаційної роботи

студента Сідака Дмитра Анатолійовича група ЗКІТ-21м
на тему: Інтелектуальна автоматизована система розпізнавання об'єктів на зображеннях земної поверхні.

Актуальність роботи в контексті спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» доведена результатами інформаційного пошуку та аналізу літературних джерел. Додатковим підтвердженням актуальності роботи слугують опубліковані тези на Науково-технічній конференції факультету інтелектуальних інформаційних технологій та автоматизації.

Представлені в магістерській кваліфікаційній роботі рішення обґрунтовані послідовним ланцюгом дій, які включають пошук інформації про проблему, її узагальнення, формулювання прикладних задач для її вирішення, проектування відповідного програмного засобу для вирішення поставлених задач, його тестування і формулювання висновків. Раціональність та ефективність прийнятих рішень доведені результатами тестування.

Дипломник показав хороший рівень спеціальних знань навичок. Дипломник продемонстрував вміння: - вирішувати поставлені керівником завдання самостійно, згідно власноруч розробленої схеми заходів; - здійснювати пошук і узагальнення інформації; - комунікативні навички.

Основні результати, представлені в роботі отримані дипломником самостійно. Матеріалу роботи властивий високий ступінь оригінальності, що доведено результатами перевірники на наявність запозичень.

Дипломник працював ритмічно, без суттєвих відхилень від затвердженого графіку. Втрати зв'язку з керівником не було.

Недоліки: Бажано було аналізувати свіжіші літературні джерела. Не всі рисунки достатньо якісні. Автор навів лише необхідні і достатні діаграми, що описують процес проектування створеної системи. Результати багатоваріантного аналізу бажано було звести в таблицю. Присутні поодинокі вади форматування.

Загалом магістерська кваліфікаційна робота ***відповідає*** спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології», заслуговує на оцінку B, а її автор ***заслуговує*** присудження кваліфікації: ступінь вищої освіти магістр, спеціальність «Автоматизація та комп'ютерно-інтегровані технології», освітня програма «Інтелектуальні системи і Інтернет речей».

Керівник магістерської кваліфікаційної роботи

к.т.н., доц. каф. АПТ 12. 12. 2021 Севастьянов В.М.

Відгук
опонента на магістерську кваліфікаційну роботу

студента Сідака Дмитра Анатолійовича група ЗКІТ-21м
на тему: Інтелектуальна автоматизована система розпізнавання об"єктів на зображеннях земної поверхні.

Актуальність роботи в контексті спеціальності 151 «Автоматизація та комп"ютерно-інтегровані технології» доведена результатами інформаційного пошуку та аналізу літературних джерел. Додатковим підтвердженням актуальності роботи слугують опубліковані тези на Науково-технічній конференції факультету інтелектуальних інформаційних технологій та автоматизації.

Перший розділ магістерської кваліфікаційної роботи повністю присвячений огляду літературних та інформаційних джерел за обраною темою. Проаналізовано не менш ніж три аналоги створюваної системи. Функціональність та форм-фактор створеної системи цілком визначені на основі результатів критичного огляду літератури.

Прийняті рішення обґрунтовані результатами огляду літератури, результатами проектування та втілені в функціонуючу програмну систему. Результати її тестування доводять правильність прийнятих рішень. Експериментальні дослідження продумані і повні. Тести охоплюють як функції інтерфейсу створеної системи, так і доводять якість та повноту виконання нею функціонального призначення, обґрунтованого на етапі проектування. Вміст графічної частина повною мірою репрезентує всі отримані в магістерській кваліфікаційній роботі результати. Якість рисунків в графічній частині прийнятна.

Недоліки: Варто було чіткіше визначити про розпізнавання яких об"єктів йдеться вже у вступі до магістерської дипломної роботи. Занадто багато уваги приділяється питанню підготовки навчальних та тестових даних. Загалом магістерська кваліфікаційна робота відповідає спеціальності 151 «Автоматизація та комп"ютерно-інтегровані технології», заслуговує на оцінку B, а її автор **заслуговує** присудження кваліфікації: ступінь вищої освіти магістр, спеціальність «Автоматизація та комп"ютерноінтегровані технології», освітня програма «Інтелектуальні системи і Інтернет речей».

Опонент на магістерську кваліфікаційну роботу

Козачко О.М.

к.т.н., доц. каф. САІТ

ЗМІСТ

ВСТУП

Розпізнавання шаблонів – це автоматична ідентифікація шаблонів у наборі даних. Це зріла, але захоплююча галузь із досить високими темпами розвитку, яка є основою для розвитку суспільних сфер, таких як комп'ютерне бачення, обробка зображень, аналіз тексту та документів, а також нейронні мережі[1]. Розпізнавання образів має застосування в статистичному аналізі даних, обробці сигналів, аналізі зображень, пошуку інформації, біоінформатиці, стисненні даних і комп'ютерній графіці. Він бере свій початок у статистиці та технологіях; завдяки доступності великих даних і збільшенню нових потужностей обробки деякі сучасні методи розпізнавання образів включають використання машинного навчання. Розпізнавання образів можна визначити як класифікацію даних на основі отриманих знань або статистичної інформації, отриманої із зображень або їх представлень.

У процесі біологічної еволюції багато тварин добре вирішували завдання розпізнавання образів за допомогою органів зору і слуху. Складною теоретико-технічною проблемою залишається створення систем штучного розпізнавання образів. Потреба в такій ідентифікації виникає в найрізноманітніших сферах - від військових і охоронних систем до оцифровки різноманітних аналогових сигналів.

Перспективним напрямком використання цієї методики є обробка результатів аерофотозйомки для визначення певних характеристик, наприклад непроникності, досліджуваного географічного положення. Існує кілька реалізацій розпізнавання образів за результатами обробки отриманих зображень, наприклад, за допомогою супутників, кожна з яких, на жаль, має певні недоліки, такі як низька швидкість навчання розроблених нейронних

мереж, низька якість результатів розпізнавання, Або багато даних, які необхідні для роботи програми.

Щоб розробити програму розпізнавання образів на основі результатів повітряних досліджень, вам слід дослідити існуючі рішення розпізнавання образів, розглянути та вибрати мову програмування та відповідні бібліотеки для реалізації програми. Також необхідно створити або реалізувати існуючу модель нейронної мережі, яка буде відповідати за розпізнавання за результатами навчання. Необхідно детально розкрити алгоритм створеної програми та її компоненти, а також процес, за допомогою якого програма виконується. Програми слід розробляти так, щоб мінімізувати прояв недоліків, виявлених в інших подібних програмах, або повністю їх уникати.

Метою даної роботи є розробка комп'ютерної програми розпізнавання образів, яка може бути використана в навчальній та науковій роботі.

Об'єктом дослідження є процес розпізнавання образів аерофотозйомки.

Предмет дослідження – алгоритми та процедури розпізнавання образів для аерофотозйомки.

Методи дослідження: методи побудови нейромережевих архітектур U-Net, VGGNet, ResNet[2].

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ДОСЛІДНИЦЬКИХ ЗАДАЧ

1.1 Аналіз актуальних напрямків застосування розпізнавання образів

В останні роки розпізнавання образів все частіше використовується в повсякденному житті. Розпізнавання мови та рукописного тексту значно спрощує взаємодію людини з комп'ютером, а розпізнавання друку використовується для перекладу документів в електронну форму.

Встановлено основні поняття, які використовуються в розпізнаванні образів. У комп'ютерах набір представлено набором унікальних елементів одного типу. Поняття «неповторюваність» означає наявність або відсутність певних елементів у наборі. Універсальна множина включає всі можливі елементи для вирішення задачі, порожня множина не містить жодного елемента.

У класичній постановці задачі розпізнавання універсальна множина розбивається на частини, які називаються зображеннями. Образ предмета задається сумою його конкретних проявів. У разі розпізнавання тексту універсальний набір включав би всі можливі символи, зображення «А» включало б усі можливі стилі літери, а програма розпізнавання визначала б, яка буква представлена певною буквою для обробки.

Спосіб призначення елементів будь-якому зображенню називається правилом рішення. Іншим важливим поняттям є метрика - спосіб визначення відстані між елементами корпусу. Чим менша ця відстань, тим схожіші символи чи звуки, які ми розпізнаємо. Як правило, елементи задаються як набір чисел, а міри – як функції. Ефективність цієї схеми залежить від вибору представлення зображення та реалізації метрик; один і той же алгоритм розпізнавання з різними метриками буде помилятися з різною частотою

(право на помилку так само характерно для програм розпізнавання, як і для людини).

Базовий алгоритм, заснований на багатокритеріальному підході, чудово демонструє принципи розпізнавання образів. Його входом є навчальна вибірка – вибірка Aij для кожного зображення Ai, метрика d і сам розпізнаний об’єкт x.

Використовуючи цю метрику, ми обчислюємо відстань d(x, aij) від x до кожного елемента навчальної вибірки та знаходимо умовну відстань d(x, Ai) як відстань від x до елемента, найближчого до Ai. Елемент x відноситься до останнього зображення.

На практиці вам потрібно знайти мінімальну відстань для кожного класу і взяти мінімальну від результату.

Іншим основним алгоритмом є метод k-найближчого сусіда. Як випливає з назви, він вводить додатковий вхідний параметр, ціле число k. Тут все відносно просто – ми беремо k елементів навчальної вибірки, найближчих до x, і підраховуємо, скільки з них належить якому зображенню. x буде належати зображенню з більшою кількістю елементів.

В обох алгоритмах може виникнути невизначеність, коли x знаходиться на однаковій відстані від кількох зображень. У цьому випадку програма повинна запитати користувача, якому зображенню призначити елемент, або вибрати випадковим чином. Це залежить від вимог до точності, з одного боку, і зручності використання, з іншого. Найкраще реалізувати обидва варіанти одночасно.Зображення або категорія – це категоріальне групування в системі класифікації, яке пов’язує (або виділяє) певну групу об’єктів за певними ознаками.

Репрезентативне сприйняття світу – одна із таємничих властивостей живого мозку, яка дозволяє розуміти нескінченний потік перцептивної інформації та зберігати орієнтацію в морі розрізнених даних про зовнішній

світ. Сприймаючи зовнішній світ, ми завжди класифікуємо відчуття, які сприймаємо, тобто поділяємо їх на групи подібних, але не ідентичних явищ. Наприклад, незважаючи на суттєві відмінності, в набір входять всі літери Y, написані різними шрифтами, або всі звуки, що відповідають одній ноті, зіграної на будь-якій октаві і на будь-якому інструменті, а також оператор, який керує технічним об'єктом у всій сукупності станів об'єкт обробляється таким же чином. Її особливість полягає в тому, що для формування поняття про певний тип групи сприйняття необхідно лише ознайомитися з нечисленними її представниками. Ця властивість мозку дозволяє виражати такі поняття як образи.

Імідж має особливість, яка проявляється в знайомстві з кінцевою кількістю явищ в одній і тій же сукупності, що дозволяє ідентифікувати як завгодно велику кількість представників. Прикладами зображень можуть бути: річки, океани, рідини тощо. Певний набір станів заводу також можна розглядати як образ, і цей набір станів характеризується необхідністю однаково впливати на об’єкт управління для досягнення заданої мети.

Зображення мають характерні об'єктивні властивості, і різні люди, які вивчають різні матеріали спостережень, здебільшого класифікують ті самі об'єкти однаково і незалежно один від одного. Саме об'єктивність цього образу дозволяє людям у всьому світі розуміти один одного.

Здатність сприймати зовнішній світ у вигляді образів дозволяє впевнено ідентифікувати нескінченну кількість об'єктів на основі знайомства з кінцевою кількістю, а об'єктивний характер основних властивостей образів дозволяє моделювати процес їх розпізнавання. У літературі, що спеціалізується на проблемах розпізнавання образів, часто вводиться поняття класів, а не зображень.

1.2 Актуальність задачі розпізнавання образів

Взагалі кажучи, проблема розпізнавання образів складається з двох частин: навчання та розпізнавання. Навчається показом окремих предметів і вказівкою на їх приналежність до того чи іншого зображення. В результаті навчання система розпізнавання повинна набути здатності однаково реагувати на всі об'єкти в одному зображенні та здатності реагувати на всі об'єкти в інших зображеннях. Дуже важливо, щоб процес навчання завершувався лише показом обмеженої кількості об’єктів без будь-яких інших підказок. Об'єктами навчання можуть бути малюнки чи інші візуальні зображення (наприклад, літери, цифри), різні зовнішні явища, наприклад звуки, фізичний стан медичної діагностики, стан технічних об'єктів медичної діагностики. Для систем керування тощо важливо лише вказувати на сам об’єкт і те, що належить зображенню під час процесу навчання. Навчання супроводжується процесом ідентифікації нових об'єктів, що характеризують дії вже навченої системи. Автоматизація цих процедур є проблемою в навчанні розпізнавання образів. Коли люди здогадуються або винаходять, а потім накладають правила класифікації на машини, проблема розпізнавання частково вирішується, оскільки люди беруть на себе більшість проблем навчання[3].

Спектр завдань, які можна вирішити за допомогою системи ідентифікації, дуже широкий. Це включає не тільки завдання розпізнавання зорових і слухових образів, а й завдання розпізнавання складних процесів і явищ, які виникають, наприклад, коли бізнес-лідери вибирають відповідні дії або обирають найкраще управління технологіями, економікою, рухом або військовими операціями. . У кожному питанні аналізуються деякі явища, процеси, стани зовнішнього світу, які називаються об’єктами спостереження. Перш ніж приступити до аналізу будь-якого об'єкта, необхідно отримати про

нього певним чином упорядковану інформацію. Ця інформація є характеристиками об'єкта, і вони відображаються на сукупності органів чуття системи розпізнавання.

Однак кожен спостережуваний об'єкт впливає на систему по-різному, в залежності від умов сприйняття. Наприклад, будь-яка буква, навіть якщо вона написана однаково, може рухатися за бажанням щодо органів чуття. Крім того, об’єкти одного зображення можуть сильно відрізнятися один від одного, природно, по-різному впливаючи на органи сприйняття.

Спектр завдань, які можна вирішити за допомогою системи ідентифікації, дуже широкий. Це включає не тільки завдання розпізнавання зорових і слухових образів, а й завдання розпізнавання складних процесів і явищ, які відбуваються, наприклад, коли керівники підприємств вибирають відповідні дії або вибирають найкраще управління технологіями, економікою, рухом або військовими операціями. У кожному питанні аналізуються, називаються якісь явища, процеси, стани зовнішнього світуоб'єкт спостереження. Перш ніж приступити до аналізу будь-якого об'єкта, необхідно отримати про нього певним чином упорядковану інформацію. Ця інформація є характеристиками об'єкта, і вони відображаються на сукупності органів чуття системи розпізнавання.

Однак кожен спостережуваний об'єкт впливає на систему по-різному, в залежності від умов сприйняття. Наприклад, будь-яка буква, навіть якщо вона написана однаково, може рухатися відносно органу чуття. Крім того, об’єкти одного зображення можуть сильно відрізнятися один від одного, природно, по-різному впливаючи на органи сприйняття.

Кожне представлення об’єкта органами сприйняття системи розпізнавання, незалежно від його положення щодо цих органів, часто називають зображенням об’єкта, а сукупність таких зображень, об’єднаних деякими загальними властивостями, називають зображенням.

При використанні методів розпізнавання зображень для вирішення адміністративних завдань замість терміна «зображення» використовується термін «стан». Стан – це певна форма відображення виміряних поточних (або миттєвих) характеристик об’єкта, що спостерігається. Ситуацію визначає вся країна. Поняття «ситуація» схоже з поняттям «імідж». Однак ця аналогія неповна, оскільки не кожен образ можна назвати контекстом, хоча будь-яку ситуацію можна назвати образом.

Контекст визначається як набір конкретних станів складного об'єкта, кожен з яких характеризується однаковими або подібними характеристиками об'єкта. Наприклад, якщо об'єкт управління розглядається як об'єкт спостереження, то ситуація об'єднує ці стани цього об'єкта, в яких має бути застосована одна і та ж керуюча дія.

Вибір початкового опису об'єкта є одним із основних завдань задач розпізнавання образів. Якщо початковий вибір опису вдалий, проблема ідентифікації може стати тривіальною, навпаки, невдалий вибір початкового опису може призвести до дуже складної подальшої обробки інформації або взагалі не мати рішення. Наприклад, якщо вирішується задача ідентифікації предметів різного кольору, а в якості вихідного опису вибирається сигнал, отриманий від датчика ваги, то задача ідентифікації не може бути вирішена взагалі. Кожного разу, коли ми стикаємося з незнайомими завданнями, природно бажати представити їх у формі якоїсь зрозумілої моделі, яка дозволить нам зрозуміти завдання так, як це легко уявити. А оскільки ми існуємо в просторі та часі, то нам найпростіше зрозуміти просторово-часову інтерпретацію завдання.

Будь-яке зображення, що виникає в результаті спостереження за об'єктом під час навчання або тестування, можна представити у вигляді вектора, а отже, як точки в деякому об'єктному просторі. Якщо стверджується, що коли зображення відображаються, вони можуть бути

однозначно віднесені до одного з двох (або більше) зображень, тоді це, таким чином, стверджується, що в певному просторі існують дві (або більше) області, які не мають спільної основи, і що зображення з тих регіонів. Кожній такій області можна присвоїти назву, яка відповідає назві зображення.

Тепер ми пояснимо процес розпізнавання візерунків з геометричної точки зору, обмежуючись випадком, коли розпізнаються лише два візерунки. Припустімо, що заздалегідь відомо, що дві області в деякому просторі потрібно розділити, і відображаються лише точки з цих областей. Самі регіони не визначені заздалегідь, тобто немає інформації про розташування їхніх кордонів і немає правил для визначення належності точки до регіону. Під час навчання випадково вибрані точки з цих регіонів представляють і передають інформацію про те, до якого регіону належать представлені точки. Іншої інформації про ці регіони, тобто про розташування їхніх кордонів, під час навчань не було надано. Метою навчання є або побудова поверхні, яка розділяє не лише точки, показані під час навчання, але й усі інші точки, які належать до цих областей, або побудова поверхні, яка пов’язує ці області так, щоб кожна область містила лише одне зображення точки. Іншими словами, мета навчання полягає в тому, щоб побудувати функцію з вектора зображення, який, наприклад, є позитивним у всіх точках одного зображення та негативним у всіх точках іншого зображення. Оскільки в регіонах немає точки дотику, завжди є цілий набір таких роздільних функцій, одна з яких повинна бути побудована в результаті навчання.

Якщо представлені зображення належать не двом, а більшій кількості зображень, завдання полягає в тому, щоб по точках, показаних під час навчання, побудувати поверхню, відокремлюючи один від одного всі відповідні цим зображенням області. Наприклад, цю проблему можна вирішити, задавши функцію, яка приймає однакове значення для точок у

кожному регіоні; значення цієї функції має бути різним для точок у різних регіонах.

На перший погляд здається, що знати лише певну кількість точок області недостатньо, щоб відокремити всю область. Насправді можна вказати нескінченну кількість різних областей, що містять ці точки, і незалежно від того, як з них будується поверхня, що визначає область, завжди можна вказати іншу область, яка перетинає поверхню і також містить вказані точки. Проте загальновідомо, що задача наближення функції за інформацією про неї в скінченній множині точок, яка значно вужча за всю множину заданих функцій, є загальною математичною задачею наближення функції. Звичайно, вирішення таких завдань вимагає введення певних обмежень на клас розглянутих функцій, і вибір цих обмежень залежить від характеру інформації, яку викладач може додати в процесі навчання. Однією з цих підказок є припущення про компактність зображення. Цілком інтуїтивно зрозуміло, що апроксимація функції розділення буде чим простіше завдання, чим більш компактною та розпорошеною буде розділена область. На додаток до геометричної інтерпретації задач розпізнавання образів існує інший підхід, який називається структурним або мовним. Пояснимо мовний метод на прикладі візуального розпізнавання образів. Спочатку визначте набір початкових понять. Це характеристики типових сегментів, які зустрічаються на зображенні, та їх взаємне розташування – «зліва», «знизу», «всередині» тощо. Ці початкові поняття утворюють словниковий запас, який дозволяє будувати різні логічні твердження, які іноді називають гіпотезами. Завдання полягає в тому, щоб із величезної кількості тверджень, які можна сконструювати за допомогою цих концепцій, вибрати твердження, яке найбільше відповідає даному випадку.

Далі, враховуючи обмежену і, можливо, невелику кількість об'єктів на кожному зображенні, необхідно побудувати описи для цих зображень. Опис

збірки має бути достатньо повним, щоб відповісти на питання, до якого зображення належить об’єкт. При реалізації лінгвістичного методу виникають дві задачі: задача побудови вихідного словника, набору канонічних фрагментів і задача побудови правил опису з елементів цього словника. У рамках інтерпретації мови була проведена аналогія між структурою зображення та синтаксисом мови. Прагнення до цієї аналогії виникає через уміння використовувати засоби математичної лінгвістики, тобто методи мають синтаксичний характер. Використання інструментів математичної лінгвістики для опису структури зображення може бути використано лише після того, як зображення буде сегментовано на складові частини, тобто розроблено слова для опису типових фрагментів і методів їх пошуку. Після початкової роботи із забезпечення відбору слів з’явилися практичні мовні завдання, зокрема завдання автоматичного граматичного синтаксичного аналізу описів для розпізнавання зображень. Водночас виникає самостійний напрямок досліджень, який потребує не лише базових знань математичної лінгвістики, а й володіння техніками, спеціально розробленими для обробки мовних образів.

Якщо припустити, що простір ознак формується в процесі навчання на основі очікуваної класифікації, то можна сподіватися, що сама специфікація простору ознак задає властивість, під впливом якої зображення в цьому просторі можна легко розрізнити. Саме ці надії спонукали до гіпотези компактності під час розробки розпізнавання образів. Припущення такі: компактний набір у просторі ознак відповідає зображенню. На даний момент компактний набір відноситься до певних «скупчень» точок у просторі зображення, які, як передбачається, розділені «розрідженістю» серед них.

Однак не завжди вдається підтвердити цю гіпотезу експериментально, але головне, що всі без винятку задачі, де спостерігалося підтвердження гіпотези компактності, знайшли просте рішення. І навпаки, завдання,

гіпотези яких не були обґрунтовані, або не вирішувалися взагалі, або дуже важко розв’язувалися із залученням додаткових «хитрощів». Цей факт ставить під сумнів справедливість гіпотези компактності, оскільки для спростування будь-якої гіпотези достатньо прикладу, який їй суперечить. У той же час підтвердження гіпотези стимулює інтерес до цієї гіпотези, якщо проблема навчання розпізнавання образів може бути вирішена. Саме припущення про компактність стає ознакою ймовірності виявлення задовільного рішення проблеми.

1.3 Класифікація методів розпізнавання образів

Розпізнавання образів відноситься до проблеми формалізації числових або символічних представлень об'єктів у реальному чи ідеальному світі, результати якої вирішують еквівалентність між цими об'єктами. Відношення еквівалентності вказує на те, що об'єкт оцінки належить до будь-якого класу, який розглядається як незалежна семантична одиниця.

При побудові алгоритму розпізнавання класи еквівалентності можуть бути задані дослідником, який може використовувати власні ідеї або зовнішню додаткову інформацію для розуміння подібності та відмінності об’єктів у контексті розв’язуваної задачі. Це призводить до процесу, який часто називають «навчання з учителем». В іншому випадку, коли автоматизована система вирішує проблему класифікації без залучення зовнішньої навчальної інформації, це називається автоматичною класифікацією або «навчанням без контролю». Більшість алгоритмів розпізнавання образів вимагають участі дуже потужних обчислювальних потужностей, які можуть бути забезпечені тільки за допомогою високопродуктивної комп'ютерної техніки.

Різні автори пропонували різні типи методів розпізнавання образів. Деякі автори розрізняють параметричні, непараметричні, евристичні методи та інші групи методів, що базуються на історичних школах розвитку та напрямках у цій галузі. Д. А. Поспелов (1990) визначає два основних способи вираження знань:

• Поглиблене відображення у вигляді діаграми зв'язку між атрибутами (ознаками);

• Розгорнуте уявлення на основі використання конкретних фактів (предметів, прикладів).

Щільне представлення фіксує закономірності та зв’язки, які пояснюють структуру даних. Для діагностичних завдань ця фіксація включає операції, що визначають властивості (характеристики) об'єктів, які призводять до необхідного результату діагностики. Щільні представлення реалізуються за допомогою операцій над значеннями атрибутів і не передбачають операцій над конкретними інформаційними фактами (об'єктами).

Широка репрезентація знань, у свою чергу, пов'язана з описом і фіксацією конкретних об'єктів предметної області і реалізується в операціях, елементами яких є об'єкти як цілісна система.

Можна провести аналогію між інтенціональним і екстенсіональним представленням знань і механізмами діяльності в лівій і правій півкулях людського мозку. Якщо права півкуля характеризується загальним архетиповим уявленням про навколишній світ, то ліва півкуля виступає як образ, що відображає зв'язок властивостей цього світу.

Два основних способи представлення знань забезпечують таксономію методів розпізнавання образів: інтенсивний, заснований на операціях із ознаками, і широкий, заснований на операціях з об’єктами.У наведеній нижче класифікації основна увага приділяється формальним методам розпізнавання образів, таким чином оминаючи розгляд евристичних методів

розпізнавання, розроблених в експертних системах. Ми обмежимося кількома коментарями щодо цього підходу.

Евристика базується на знаннях та інтуїції, які дослідникам важко формалізувати. Таким чином дослідники самі визначають, що таке інформація і як її використовувати для досягнення бажаного ефекту ідентифікації.

Відмінною рисою щільних методів є те, що вони використовують різні ознаки ознак та їхні зв’язки як операційні елементи при побудові та застосуванні алгоритмів розпізнавання образів. Такими елементами можуть бути окремі значення або інтервали значень атрибутів, середні та дисперсії, матриці агрегації атрибутів тощо, над якими виконуються дії, виражені в аналітичній або конструктивній формі. Водночас об’єкти в цих методах розглядаються не як цілісні одиниці інформації, а як індикатори взаємодії та поведінки для оцінки їхніх властивостей. Група щільних методів розпізнавання образів велика, і їх поділ на підкласи є дещо умовним.

Метод, заснований на оцінці щільності розподілу власних значень, спирається на класичну статистичну теорію рішень, в якій об’єкт дослідження розглядається як реалізація багатовимірних випадкових величин, розподілених у власному просторі за певним законом. Вони засновані на байєсівській схемі прийняття рішень, яка використовує попередню ймовірність належності об’єктів до деякого ідентифікованого класу та умовну щільність розподілу значень власних векторів. Ці методи спрощені для визначення коефіцієнтів ймовірності для різних областей багатовимірного простору ознак.

Ця група методів, заснованих на оцінці щільності розподілу власних значень, безпосередньо пов'язана з методами дискримінантного аналізу. Байєсівські методи прийняття рішень є одним із так званих параметричних методів, найбільш розроблених у сучасній статистиці, припускаючи, що

аналітичний вираз для закону розподілу (у цьому випадку нормального закону) відомий і що потрібно оцінити лише кілька параметрів (Вектор матриці середнього значення та коваріації).

Група також включає методи для обчислення відношення ймовірностей незалежних ознак. Окрім припущення незалежності ознак (яка насправді ніколи не досягається), цей підхід не вимагає знання функціональної форми законів розподілу. Тому його можна віднести до непараметричного.

Особливе місце займають інші непараметричні методи, які використовуються, коли форма кривої щільності розподілу невідома і не можна зробити припущень про її властивості. До них відносяться метод багатовимірної гістограми, метод k-найближчого сусіда, метод евклідової відстані, метод потенційної функції тощо. Ці методи формально мають справу з об’єктами як монолітними структурами, але вони можуть працювати інтенсивно та широко залежно від типу завдання розпізнавання.

Непараметричні методи аналізують відносну кількість об’єктів, що потрапляють у певний багатовимірний об’єм, і використовують різні функції відстані між об’єктами в навчальних вибірках і розпізнаними об’єктами. Для кількісних ознак операції з об'єктами відіграють проміжну роль в оцінці локальної щільності умовних розподілів ймовірностей, коли їх кількість значно менша за обсяг вибірки, а об'єкти не несуть семантичного навантаження самостійних одиниць інформації. У той же час, коли кількість ознак співмірна або перевищує кількість досліджуваних, а ознаки мають якісний або дихотомічний характер, немає проблеми будь-якої локальної оцінки щільності розподілу ймовірностей. У цьому випадку об'єкти в цих непараметричних методах розглядаються як самостійні одиниці інформації (синтезують емпіричні факти), і ці методи набувають значення оцінки подібності та відмінності об'єктів дослідження.

Тому, залежно від умов задачі, однакові технічні маніпуляції непараметричними методами мають сенс для локальних оцінок щільності ймовірності Власні значення, або використовуються для оцінки подібності та несхожості об'єктів.

Основна складність у застосуванні цих методів полягає в необхідності запам'ятовування всієї навчальної вибірки для обчислення локальних оцінок щільності розподілу ймовірностей і високої чутливості до непредставництва навчальної вибірки.

Методи, засновані на припущеннях про клас функцій рішення, є іншою групою методів підкріплення. У цій групі методів загальний вигляд вирішальної функції вважається відомим і встановлюється її масова функція. На основі цієї функції, використовуючи навчальну послідовність, знайти найкраще наближення вирішальної функції. Найбільш поширеними є представлення вирішальних функцій у вигляді лінійних і узагальнених нелінійних поліномів. Функція якості для рішень щодо правил зазвичай пов’язана з помилкою класифікації.

Основною перевагою підходу, заснованого на припущеннях про клас вирішальних функцій, є чіткість математичної постановки задачі ідентифікації як задачі пошуку екстремуму. Зазвичай для вирішення цієї проблеми використовується будь-який градієнтний алгоритм. Різноманітність цього набору методів пояснюється широким використанням вирішальних правил і масою функціоналів алгоритмів пошуку екстремуму. Метод стохастичної апроксимації є узагальненням розглянутого алгоритму, який включає, зокрема, алгоритм Ньютона, алгоритм персептронного типу та ін. На відміну від методів ідентифікації параметрів, успіх цієї групи методів не надто залежить від розбіжності між теоретичною концепцією закону розподілу об’єктів у просторі ознак та емпіричною реальністю. Всі операції підпорядковані одній головній меті - знайти екстремуми якісних

функціоналів для вирішальних правил. При цьому результати параметричного методу та розглянутого методу можуть бути схожими. Параметричний підхід для нормально розподілених випадків об’єктів у різних класах з рівними коваріаційними матрицями призводить до лінійної функції рішення.

Можливість градієнтних алгоритмів знаходити екстремальні значення, особливо в наборах лінійних вирішальних правил, добре вивчена. Збіжність цих алгоритмів демонструється, лише якщо ідентифіковані класи об’єктів представлені в просторі об’єктів компактними геометричними структурами. Проте бажання реалізувати правила прийняття рішень достатньої якості часто можна задовольнити алгоритмами, які демонструють збіжність розв’язків до глобальних екстремумів без строгих математичних доказів.

Ці алгоритми включають великий набір процедур евристичного програмування, які представляють напрямок еволюційного моделювання. Еволюційне моделювання є біонічним підходом, запозиченим у природи. Він заснований на використанні відомих еволюційних механізмів для заміни процесу змістовного моделювання складних об’єктів феноменологічним моделюванням їх еволюції.

Відомим представником еволюційного моделювання в розпізнаванні образів є метод групового розгляду аргументів (MGRA). MGRA базується на принципі самоорганізації, а алгоритм MGRA відтворює величезний вибір схем[4]. В алгоритмі MGRA коефіцієнти узагальнених поліномів (часто їх називають поліномами Колмогорова-Габора) синтезуються та підбираються особливим чином. Складність цього синтезу та відбору зростає, і неможливо заздалегідь передбачити, якою буде кінцева форма узагальненого полінома. Спочатку зазвичай розглядається проста попарна комбінація вихідних ознак, з якої складається рівняння вирішальної функції, зазвичай не вище другого порядку. Кожне рівняння аналізується як незалежна вирішальна функція, а

значення параметрів для рівняння якимось чином знаходяться з навчальних зразків. Потім з набору отриманих вирішальних функцій виберіть кілька найкращих. Контроль якості окремих функцій прийняття рішень здійснюється на контрольній вибірці, що іноді називають принципом накладення. Вибрані часткові функції прийняття рішень далі розглядаються як проміжні змінні, як синтетичні вихідні параметри для нових функцій рішення тощо. Цей процес ієрархічного синтезу триває до досягнення екстремального значення критерію якості функції розв'язку, що на практиці проявляється у погіршенні цієї якості при спробі подальшого збільшення порядку членів функції розв'язку. Поліном щодо вихідної функції.

Принцип самоорганізації MGRA називається евристичною самоорганізацією, оскільки весь процес базується на зовнішніх доповненнях, які вводять евристичний вибір. Результат рішення може значною мірою залежати від цих евристик. Результуюча діагностична модель залежить від того, як об'єкт розбивається на навчальну і тестову вибірки, як визначаються критерії якості розпізнавання, скільки змінних передається в наступну серію відбору і т.д.

Логічний підхід до розпізнавання образів базується на апараті логічної алгебри і дозволяє використовувати інформацію, що міститься не тільки в окремих символах, а й у комбінаціях значень символів. У цих методах значення будь-якої властивості розглядається як базова подія.

У найзагальнішому вигляді логічний метод можна описати як пошук навчального зразка логічних законів і формування певної системи логічних вирішальних правил (наприклад, у вигляді зв'язків елементарних подій), кожне з яких має свої вага. Група логічних методів різноманітна і включає методи різної складності та глибини аналізувати

Лінгвістичні підходи до розпізнавання образів засновані на використанні спеціальної граматики генеративних мов, які можуть бути використані для опису набору властивостей розпізнаваних об'єктів.

Для різних класів об'єктів виділяються непохідні елементи (зображення, символи) і можливі зв'язки між ними. Синтаксис стосується правил побудови об’єктів із цих непохідних елементів. Кожен об’єкт, таким чином, представлений набором непохідних елементів, які так чи інакше «пов’язані» один з одним, або, іншими словами, «реченнями» деякої «мови». Синтаксичний аналіз (розбір) «речення» визначає його синтаксичну «правильність» або, що еквівалентно, чи може якась фіксована граматика (опис класу) дати існуючий опис об’єкта. Розбір виконується так званим «парсером», який представляє повний синтаксичний опис об’єкта у формі дерева розбору, якщо об’єкт синтаксично правильний (тобто належить до класу, описаного цією граматикою). В іншому випадку об’єкт або відхиляється, або аналізується іншими граматиками, що описують об’єкти інших класів. Розрізняють контекстно-вільну, автоматичну та інші види граматик. Однак завдання відновлення (визначення) граматики з набору конкретних речень (речень, що описують об’єкти, що породжують дану мову) важко формалізувати. Евристичні правила для автоматичного відновлення граматики описані в літературі для побудови та застосування алгоритмів мови розпізнавання образів.

При широкому підході, на відміну від щільного напрямку, кожному досліджуваному об'єкту надається самостійне діагностичне значення. Фактично ці методи близькі до медичних методів, які розглядають людину не як ланцюжок класифікованих за тими чи іншими показниками об'єктів, а як цілісну систему, кожна з яких є самостійною і має особливу діагностичну цінність. Такий обережний підхід до досліджуваних суб’єктів не допускає виключення або втрати інформації про кожного окремого суб’єкта, що

відбувається при використанні щільно орієнтованих методів, використання лише суб’єкта для виявлення та відновлення поведінкових моделей його властивостей.

Основною операцією розпізнавання образів за допомогою розглянутого методу є операція визначення подібності та несхожості об'єктів. Об'єкти цього набору методів служать діагностичними прецедентами. При цьому, залежно від умов конкретного завдання, роль окремих прецедентів може варіюватися в найширшому діапазоні - від базової до дуже опосередкованої участі в процесі розпізнавання. У свою чергу, умови проблеми можуть вимагати участі різної кількості діагностичних прецедентів для успішного вирішення – від одного в кожному розпізнаному класі до загального обсягу вибірки, а також різних методів і розрізнення об’єктів для обчислення показників подібності.

Метод порівняння прототипів є найпростішим вдосконаленим методом ідентифікації. Він використовується, наприклад, коли визначені класи виглядають як компактні геометричні групи в просторі об’єктів. У цьому випадку за точку-прототип зазвичай вибирають центр геометричного угруповання класу (або найближчий до центру об’єкт). Щоб класифікувати невідомий об'єкт, знайдіть найближчий прототип, і об'єкт належить до того ж класу, що й цей прототип. Очевидно, що використання цього методу не генерує загальний образ класу.

Як міру близькості можна використовувати різні типи відстані. У цьому випадку відстань Хеммінга дорівнює квадрату евклідової відстані, яка зазвичай використовується для дводольних елементів. У цьому випадку вирішальне правило для класифікації об’єктів еквівалентне лінійній вирішальній функції. Цей факт заслуговує на особливу увагу. Це чітко демонструє взаємозв’язок між архетипами та представленнями функцій інформації про структуру даних. І навпаки, якщо аналіз просторової

структури ідентифікованих класів дозволяє зробити висновок, що вони є геометрично компактними, то кожен із цих класів може бути замінений прототипом, що фактично є лінійною діагностичною моделлю.

Звичайно, на практиці ситуація часто відрізняється від описаного ідеалізованого прикладу. Дослідники, які мають намір застосувати методи ідентифікації на основі порівняння з прототипами діагностичного класу, стикаються зі складною проблемою. Це спочатку вибір близькості (метрики), З нього може бути істотно змінена просторова конфігурація розміщення об'єкта. Окремим питанням є також аналіз багатовимірної структури експериментальних даних. Ці дві проблеми особливо гостро постають для дослідників у високій розмірності простору ознак, характерній для практичних задач.

Метод k-найближчого сусіда для вирішення проблем дискримінантного аналізу вперше був запропонований у 1952 році. Його формула така.

Під час класифікації невідомого об’єкта відображається задана кількість (k) інших об’єктів (найближчих сусідів), які геометрично найближчі до нього в просторі атрибутів і, як відомо, належать до ідентифікованого класу. Рішення про віднесення невідомого об'єкта до тієї чи іншої діагностичної категорії приймається шляхом аналізу інформації про цю відому приналежність найближчих сусідів, наприклад, за допомогою простого підрахунку голосів.

Спочатку метод k-найближчого сусіда вважався непараметричним методом для оцінки відношень правдоподібності. Для цього методу отримано теоретичну оцінку його ефективності порівняно з найкращим байєсівським класифікатором.

Доведено, що ймовірність асимптотичної помилки методу k-найближчого сусіда не більше ніж вдвічі перевищує ймовірність байєсівського методу.

Як зазначалося вище, в практичних задачах часто доводиться оперувати об'єктами, що описуються великою кількістю якісних (дихотомічних) ознак. У цьому випадку розмір ознакового простору співмірний або перевищує обсяг досліджуваної вибірки. У цьому випадку кожен об'єкт навчальної вибірки зручно інтерпретувати як окремий лінійний класифікатор. Тоді та чи інша діагностична категорія представляється не прототипом, а набором лінійних класифікаторів. Кумулятивні взаємодії лінійних класифікаторів призводять до кусково-лінійної поверхні, яка розділяє ідентифіковані класи в просторі ознак. Зовнішній вигляд поверхні сегментації, що складається з кількох гіперплощин, може змінюватись і залежить від відносного розташування агрегатів, які потрібно класифікувати.

Можна також використати іншу інтерпретацію механізму класифікації k-найближчих сусідів. Він заснований на ідеї існування деяких прихованих змінних, які є абстрактними або пов’язаними з певними перетвореннями в оригінальному просторі ознак.

Якщо в прихованому просторі змінних попарні відстані між об’єктами такі ж, як і в початковому просторі ознак, а кількість цих змінних набагато менша за кількість об’єктів, то можна вивести інтерпретацію методу k-найближчого сусіда з точки зору порівняння непараметричних щільностей ймовірності умовних розподілів

Представлена тут концепція прихованих змінних близька за своєю природою до концепції справжньої розмірності та інших уявлень, що використовуються в різних техніках зменшення розмірності.

Використовуючи метод k-найближчого сусіда для розпізнавання образів, дослідники повинні вирішити складну проблему вибору показників виявлення

Близькість об'єкта, що діагностується. Оскільки метод є досить трудомістким, проблема стає надзвичайно гострою в умовах великої

розмірності об’єктного простору, що важливо навіть для високопродуктивних комп’ютерів. Отже, тут, як і в методі порівняння прототипів, необхідно вирішувати творче завдання аналізу багатовимірної структури експериментальних даних для мінімізації кількості об’єктів, Представляє діагностичну категорію. Недоліком такого підходу, на думку авторів, є необхідність зменшення кількості об’єктів у навчальній множині, оскільки знижується репрезентативність навчальних вибірок.

Алгоритм обчислення оцінки (АОО) працює за принципом обчислення оцінки подібності, яка характеризує «близькість» ідентифікації та посилання на об’єкти через систему набору ознак, яка є системою підмножини для даного набору ознак.

На відміну від усіх методів, розглянутих раніше, алгоритм, який використовується для обчислення оцінки, описує об’єкт абсолютно по-новому. Для цих алгоритмів об’єкти існують одночасно в дуже різних підпросторах простору ознак. Клас АОО доводить ідею використання ознак до логічного завершення: оскільки не завжди відомо, які комбінації ознак є найбільш інформативними, в АОО подібність об'єктів обчислюється шляхом порівняння всіх можливих або певних комбінацій ознак, що містяться в описі об'єкта.

Комбінація використовуваних ознак (підпросторів) називається опорним набором або частковим набором опису об’єкта. Вводиться поняття узагальненої близькості між ідентифікованим об’єктом і об’єктом (званим еталонним об’єктом) навчальної вибірки (з відомою класифікацією). Ця близькість представлена комбінацією близькості об’єкта, що ідентифікується, до еталонного об’єкта, обчисленого на частковому наборі опису. Тому АОО є продовженням методу k-найближчого сусіда, де в даному просторі ознак розглядається лише близькість об’єктів.

Іншим розширенням АОО є те, що в цих алгоритмах задача визначення подібності та відмінності об’єктів формулюється як параметризована задача, а етап встановлення АОО виділяється відповідно до навчальних вибірок, а оптимальний параметр вхідного значення вибрано. Критерій якості полягає у виявленні помилок, насправді все параметризовано:

• Правила розрахунку близькості об'єктів за індивідуальними ознаками,

• правила обчислення близькості об'єктів у підпросторі об'єктів,

• ступінь важливості того чи іншого еталонного об'єкта як діагностичного прецеденту,

• Важливість кожного набору контрольних ознак для остаточної оцінки подібності ідентифікованого об'єкта до діагностичної категорії.

Параметр АОО задається у вигляді порогу та/або ваги зазначеного компонента. Теоретична потужність АОО перевищує або, принаймні, не поступається потужності будь-якого іншого алгоритму розпізнавання образів, оскільки всі можливі маніпуляції з об'єктами дослідження можуть бути реалізовані за допомогою АОО. Однак, як це часто буває, розширення потенційних можливостей стикається з величезними труднощами при своїй практичній реалізації, особливо на етапі побудови (налаштування) таких алгоритмів. Деякі труднощі були відмічені раніше при обговоренні методу k-найближчого сусіда, який можна інтерпретувати як усічену версію АОО. Її також можна розглядати в параметричній формі, а задачу можна звести до знаходження зваженої міри обраного типу. Водночас для проблеми розмірності тут виникли важкі теоретичні проблеми та проблеми, пов’язані з ефективною організацією. процес розрахунку. Для АОО ці труднощі збільшуються в рази, якщо ми намагаємося використати весь потенціал цих алгоритмів.

Вищезазначена проблема пояснює той факт, що розв'язання великомасштабних задач за допомогою АОО на практиці супроводжується

введенням будь-яких евристичних обмежень і припущень. Зокрема, добре відомий приклад використання АОО в психодіагностиці, де тестуються різні АОО, фактично еквівалентні методу k-найближчого сусіда.

1.4 Постановка задачі на дослідження

Існуючі методи розпізнавання образів для аерофотозйомки БПЛА мають недоліки, а саме:

• Визнати, що мережа повільно навчається;

• робота з дисками великої ємності;

• Результати низької якості.

У зв'язку з цим, щоб подолати ці недоліки, необхідно вибрати модель нейронної мережі для навчання, яка уникає (якщо можливо, взагалі не дозволяє) вищевказаних проблем, а також може бути реалізована за допомогою існуючих програмних засобів. Зберігання даних, а також виконання самої ідентифікації можна перенести в хмарне сховище та сервери, що значно зменшить навантаження на персональні комп’ютери дослідників. Тому програму можна запускати навіть на слабких машинах з доступом в інтернет.

Розпізнавання шаблонів – це процес ідентифікації шаблонів за допомогою алгоритмів машинного навчання. Розпізнавання зображень можна визначити як класифікацію даних на основі отриманих знань або статистичної інформації, отриманої з зображень та/або їх зображень. Одним із важливих аспектів розпізнавання образів є його потенціал застосування, наприклад, розпізнавання мови, розпізнавання мовця, розпізнавання мультимедійних документів, автоматична медична діагностика тощо.

Подано огляд методів розпізнавання образів та визначено їх основні недоліки. Виходячи з цього, було вирішено створити програму, яка б

допомогла ідентифікувати різні зображення на основі аерофотозйомки та уникнути вищевказаних недоліків. Якщо хтось вирішить покращити мій проект, то інші студенти також можуть ним скористатися. Створити таку комп’ютерну програму, використовуючи лише мову програмування високого рівня і жодних інших засобів, практично неможливо. Може бути платформа, яка полегшить розвиток такого проекту. Наступний розділ буде присвячений пошуку таких інструментів.

2 ВИБІР ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РОЗРОБКИ СИСТЕМИ РОЗПІЗНАВАННЯ ОБРАЗІВ

У попередньому розділі було представлено характеристики розпізнавання образів та його основні поняття. Тепер необхідно розглянути процес розробки справжньої програми розпізнавання образів та інструменти, які використовуються для її створення.

2.1 Аналіз та вибір мови програмування для реалізації системи розпізнавання образів

Вирішувати задачі розпізнавання образів можна на досить широкому діапазоні мов програмування. Серед них, наприклад, C#, C++, Java.

Простота вивчення та кодування, а також наявність великої кількості сторонніх бібліотек, призначених для завдань розпізнавання образів, призвели до вибору мови Python і використання інших бібліотек як платформи розробки.

Python був створений наприкінці 1980-х років і вперше випущений у 1991 році Гвідо ван Россумом як наступник мови програмування ABC. Випущений у 2000 році Python 2.0 представив нові функції, такі як розуміння списків і система збору сміття з підрахунком посилань, і був припинений у версії 2.7 у 2020 році. Випущений у 2008 році Python 3.0 був основною редакцією мови, але він не був повністю сумісним із попередніми версіями, і більшість коду Python 2 не змінилася, коли він працював на Python 3.

Python – це інтерпретована об’єктно-орієнтована мова програмування високого рівня з динамічною семантикою. Вбудовані високорівневі структури даних у поєднанні з динамічною типізацією та динамічним

зв’язуванням роблять його ідеальним для швидкої розробки додатків, а також як мову сценаріїв або з’єднувальної мови для з’єднання існуючих компонентів. Простий у вивченні синтаксис Python підкреслює читабельність, таким чином зменшуючи витрати на обслуговування програми. Python підтримує модулі та пакунки, які заохочують модульність програми та повторне використання коду. Інтерпретатор Python і обширна стандартна бібліотека вільно доступні у вихідній або двійковій формі на всіх основних платформах, включаючи Windows під час розробки основної програми, і вільно розповсюджуються. На даний момент вона зрівняна з Java як друга за популярністю мова програмування у світі.

Python є багатопарадигмальною мовою програмування. Об’єктно-орієнтоване програмування та структуроване програмування повністю підтримуються, і багато його функцій підтримують функціональне та аспектно-орієнтоване програмування (включаючи метапрограмування та метаоб’єкти (магічні методи)). Розширення підтримують багато інших парадигм, включаючи розробку контрактів і логічне програмування.

Python використовує динамічну типізацію та комбінацію підрахунку посилань і визначеного циклом збирача сміття для керування пам’яттю. Він також має динамічне визначення імен (пізнє зв’язування), яке зв’язує імена методів і змінних під час виконання.

Python був розроблений, щоб забезпечити певну підтримку функціонального програмування в традиції Lisp. Має можливості фільтрації, відображення та скорочення; розуміння списків, словники, набори та генератор виразів. Стандартна бібліотека має два модулі (itertools і functools), які реалізують функціональні інструменти, запозичені з Haskell і Standard ML.

Основна філософія мови коротко викладена в документації Python (PEP 20), яка містить такий девіз:

• Красиве краще, ніж потворне.

• Явне краще, ніж неявне.

• Просте краще складного.• Складність краще складності.

• Читабельність важлива.

Python не має всіх своїх функціональних можливостей, вбудованих у його ядро, але розроблений таким чином, щоб бути дуже розширюваним. Ця компактна модульність робить його особливо популярним як спосіб додавання програмованих інтерфейсів до існуючих програм. Бачення Ван Россума щодо невеликої базової мови з великою стандартною бібліотекою та перекладачем, що легко розширюється, виникло через його розчарування ABC, які віддавали перевагу протилежному підходу.

Python був розроблений, щоб бути легкою для читання мовою. Його форматування, як правило, абсолютно безладне, і він використовує англійські ключові слова, тоді як інші мови використовують знаки пунктуації. На відміну від багатьох інших мов, у ній не використовуються фігурні дужки для розділення блоків, а крапка з комою необов’язкова для операторів після. Замість цього, Python використовує пробільний відступ для розділення блоків. Відступ відбувається після певних операторів; зменшення відступу означає кінець поточного блоку. Тому візуальна структура програми точно відображає змістову структуру програми. Цю особливість іноді називають «зовнішніми» правилами, і в деяких інших мовах вона є, але в більшості мов відступ не має семантичного значення.

Я використовую різні аспекти парадигми функціонального програмування для досягнення ефективного процесу розробки програми розпізнавання образів.

Функціональне програмування – це парадигма програмування, у якій програми будуються за допомогою застосування та композиції функцій. Це декларативна парадигма програмування, у якій визначення функцій є

деревами виразів, кожне з яких повертає значення, а не послідовність імперативних операторів, які змінюють стан програми.

У функціональному програмуванні функції розглядаються як громадяни першого класу, що означає, що їх можна прив’язувати до імен (включаючи локальні ідентифікатори), передавати як аргументи та повертати з інших функцій, як і будь-який інший тип даних. Це дозволяє писати програми в декларативному та складному стилі, де невеликі функції поєднуються в модульний спосіб.

Функціональне програмування іноді трактується як синонім функціонального програмування, підмножини функціонального програмування, яке розглядає всі функції як детерміновані математичні функції або чисті функції. Коли чиста функція викликається з деякими заданими аргументами, вона завжди повертає той самий результат і на неї не впливає будь-який стан змінної чи інші побічні ефекти. Це відрізняється від нечистих підпрограм, поширених у імперативному програмуванні, які можуть мати побічні ефекти (такі як зміна стану програми або отримання введення користувача). Функціональне програмування виникло в академічних колах і розвинулося з лямбда-числення, яке є формальною обчислювальною системою, заснованою лише на функціях. Функціональне програмування історично було менш популярним, ніж імперативне програмування, але сьогодні багато функціональних мов використовуються в промисловості та освіті.

Найпомітніші особливості функціонального програмування такі:

• Мови функціонального програмування розроблені навколо концепції математичних функцій, які використовують умовні вирази та рекурсію для виконання обчислень.

• Функціональне програмування підтримує функції вищого порядку та функції відкладеного оцінювання.

• Мови функціонального програмування не підтримують керування потоком, наприклад оператори циклу та умовні оператори, такі як оператори If-Else та Switch. Вони безпосередньо використовують функції та виклики функцій.

• Як ООП, функціональні мови програмування підтримують такі популярні концепції, як абстракція, інкапсуляція, успадкування та поліморфізм. Функціональне програмування має такі переваги:

• Функціональне програмування не має стану, тому немає побічних ефектів і можна писати код без помилок.

• Функціональні мови програмування не мають змінних стану, тому змінити стан не проблема. «Функції» можуть бути запрограмовані паралельно як «інструкції». Такий код підтримує легке повторне використання та перевірку.• Функціональна програма складається з незалежних блоків, які можуть працювати одночасно. Тому такі програми більш ефективні.

• Функціональне програмування підтримує вкладені функції.

• Функціональне програмування підтримує ліниві функціональні конструкції, такі як ліниві списки, ліниві карти тощо.

Як недолік, функціональне програмування вимагає багато пам'яті. Оскільки немає поняття стану, нові об’єкти повинні створюватися кожного разу, коли виконується дія. Функціональне програмування використовується в ситуаціях, коли над одним набором даних необхідно виконати багато різних операцій.

2.2 Аналіз та вибір для використання датасетів для тестування системи розпізнавання образів

Велику стандартну бібліотеку Python часто вважають однією з найбільших переваг Python, оскільки вона надає інструменти, придатні для багатьох завдань. Однак для нашого конкретного завдання нам потрібно використовувати кілька зовнішніх бібліотек і компонентів.

TensorFlow є найбільш широко використовуваною бібліотекою під час розробки додатків[5]. Це відкрита бібліотека програмного забезпечення для машинного навчання, розроблена Google для вирішення проблеми побудови та навчання нейронних мереж для автоматичного пошуку та класифікації зображень для сприйняття, схожого на людину. Він використовується як для досліджень, так і для розробки власних продуктів Google. Основний API для використання бібліотеки реалізований для Python (використовується, як описано вище). Також є реалізації в R, C#, C++, Haskell, Java, Go та Swift.

Це продовження закритого проекту DistBelief. TensorFlow спочатку був розроблений командою Google Brain для внутрішнього використання в Google. У 2015 році система отримала відкритий доступ за ліцензією Apache 2.0 Open[6].

Важливою частиною цієї бібліотеки є Keras API. Він використовується для реалізації нашої основної моделі прогнозування. Keras – це API високого рівня TensorFlow 2.0: сумісний високопродуктивний інтерфейс для вирішення проблем машинного навчання з акцентом на сучасне глибинне навчання. Він надає фундаментальні абстракції та будівельні блоки для розробки та надання рішень машинного навчання з високою частотою ітерацій. Keras також є дуже гнучкою основою для сучасних дослідницьких ідей. Keras дотримується принципу інкрементальної складності: це полегшує початок роботи, але все ще дозволяє обробляти будь-які складні випадки використання, вимагаючи лише інкрементального навчання на кожному кроці.

Я використовую бібліотеку Folium Python для відображення карти з результатами виконання програми. Folium дозволяє легко візуалізувати дані, оброблені в Python, на інтерактивних листівках. Це дозволяє прив'язувати дані до карти, щоб відтворювати контури та передавати вдосконалені векторні, растрові або HTML-візуалізації як маркери на карті.

Бібліотека містить багато вбудованих наборів плиток із OpenStreetMap, Mapbox і Stamen, а також підтримує спеціальні набори плиток за допомогою API-ключів Mapbox або Cloudmade. folium підтримує накладання зображень, відео, GeoJSON і TopoJSON.

Я використовую Jupyter Notebook для розміщення та запуску програми. Jupyter Notebooks (раніше IPython Notebooks) – це веб-інтерактивне обчислювальне середовище для створення документації для Jupyter Notebooks. Термін «ноутбук» у розмовній мові може стосуватися багатьох різних об'єктів, головним чином веб-програм Jupyter, веб-серверів Jupyter Python або форматів документів Jupyter, залежно від контексту. Документ Jupyter Notebook – це документ JSON, що відповідає схемі керування версіями, що містить упорядкований список блоків введення/виведення, який може містити код, текст (з використанням Markdown), математичні, графічні та мультимедійні файли, зазвичай із розширенням ". ipynb". Jupyter Notebook став популярним інтерфейсом користувача хмарних обчислень, А основні постачальники хмарних послуг використовують ноутбуки Jupyter або похідні від них як інтерфейси для користувачів хмарних технологій. Приклади включають SageMaker Notebook від Amazon, Google Collaboratory і Azure Notebook від Microsoft.

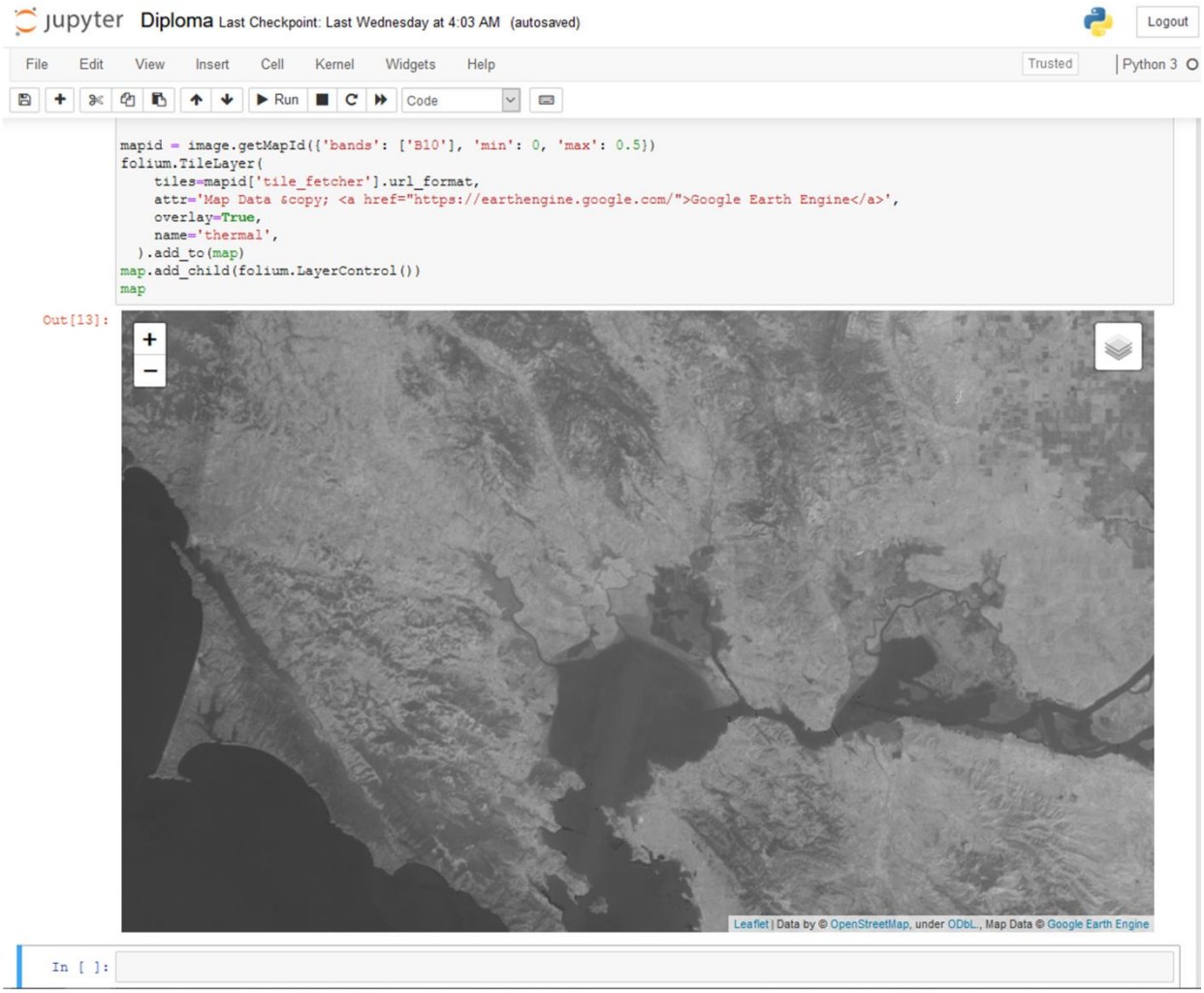


Рисунок 2.1 - Приклад використання Jupyter Notebook

Будь-який процес розпізнавання образів потрібно навчати (і оцінювати) на певному наборі даних. Для цього я використав API Google Earth Engine.

Google Earth Engine поєднує багатопетабайтні каталоги супутникових зображень і наборів геопросторових даних із можливостями планетного аналізу, що дозволяє вченим, дослідникам і розробникам використовувати його для виявлення змін на поверхні Землі, картографування тенденцій і кількісного визначення відмінностей. Загальнодоступний архів даних містить понад три десятиліття історичних зображень і наборів наукових даних, який щодня оновлюється та розширюється. Він містить понад 20 петабайт геопросторових даних, готових для аналізу. API Earth Engine доступний на

Python і JavaScript. Для ефективної роботи Earth Engine API потрібен уже створений обліковий запис Google Cloud Storage.

2.3 Аналіз методів побудови нейромережевих моделей для подальшого використання

При підготовці до розробки програми було розглянуто кілька моделей нейронних мереж, у тому числі VGGNet, ResNet та ін.

Архітектура мережі VGGNet була представлена Сімоніаном і Цисерманом у їхній статті 2014 року «Глибокі згорткові мережі для розпізнавання великомасштабних зображень».

Основними типами мереж, що використовуються, є VGG16 і VGG19. «16» і «19» представляють кількість вагових шарів у мережі.

Мережа характеризується своєю простотою, використовуючи лише 3×3 згорткові шари, складені разом на зростаючій глибині. Зменшення розміру тома відбувається за рахунок максимальних агрегатів. Два повністю пов’язаних шару, кожен з яких має 4096 вузлів, за якими йде класифікатор softmax.

Автори виявили, що VGGNet важко навчити (особливо для конвергенції в глибших мережах), тому для полегшення навчання вони спочатку навчили меншу версію VGG з меншою кількістю шарів ваги.

Менші мережі об’єднуються, а потім використовуються як ініціалізація для більших, глибших мереж – процес називається попереднім навчанням.

Логічно, що попереднє навчання є дуже трудомістким і виснажливим завданням, яке потребує навчання всієї мережі, перш ніж її можна буде використовувати як ініціалізацію для більш глибоких мереж. Це виявило перший істотний недолік моделі, а саме надзвичайно довгий час навчання

мережі. Другий недолік - висока вимогливість до місця на жорсткому диску і пропускної здатності інтернет-каналу.

ResNet – це штучна нейронна мережа, заснована на відомій структурі пірамідних клітин кори головного мозку. Типова модель ResNet реалізована з двома або трьома шарами каналів з нелінійностями (ReLU) і нормалізацією між ними. Однією з мотивів для пропуску шарів є уникнення проблеми зникнення градієнтів шляхом повторного використання активацій із попередніх шарів, доки наступний шар не дізнається свої ваги. Під час тренування вагові коефіцієнти регулюються, щоб придушити верхні шари (що потребують уточнення) і підсилити раніше пропущені шари. У найпростішому випадку регулюються лише ваги з'єднань суміжних шарів без явних вагових коефіцієнтів верхнього рівня. Цей метод найкраще працює, коли окремі нелінійні шари перетинаються або коли всі проміжні шари є лінійними. Якщо ні, слід дослідити явну матрицю ваг для втрачених з'єднань (слід використовувати HighwayNet).

Traversal ефективно спрощує мережу, використовуючи менше рівнів на початкових етапах навчання. Це прискорює навчання, зменшуючи ефект зникнення градієнтів, оскільки існує менше шарів для поширення. Потім мережа поступово відновлює втрачені шари, вивчаючи простір об'єктів. Наприкінці навчання, коли всі шари розгорнуті, він знаходиться ближче до колектора і, таким чином, навчається швидше. Нейронні мережі без залишків досліджують більшу частину простору функцій.

Якість розпізнавання погіршується через швидше навчання, як показано на малюнку 2.1

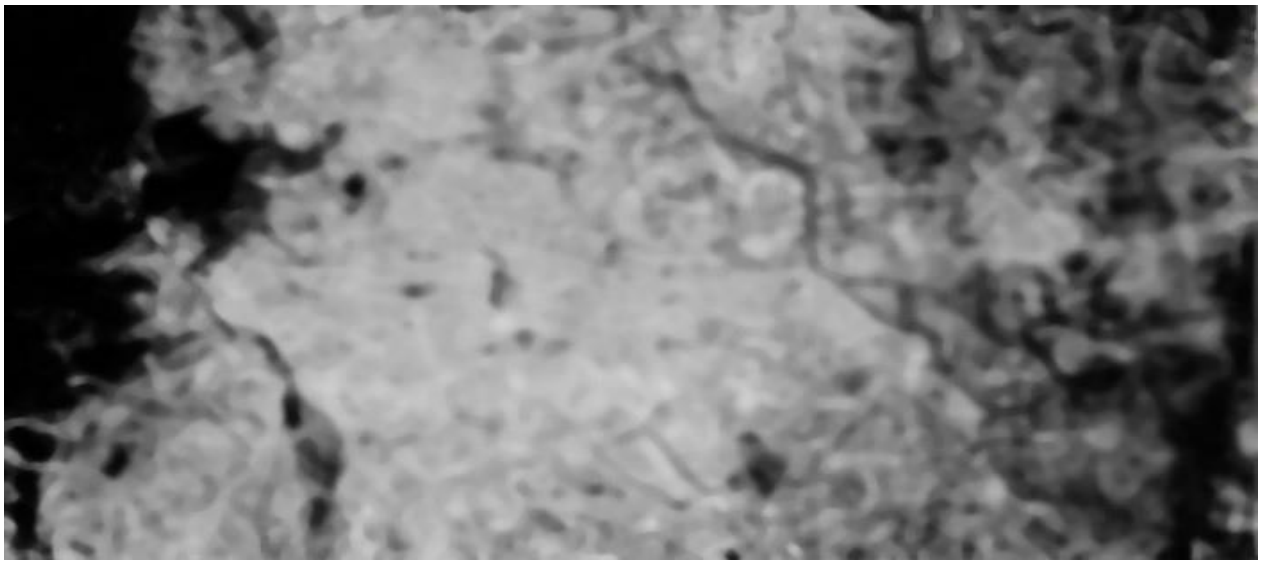


Рисунок 2.2 - Приклад використання моделі ResNet для задачі аналізу непроникної області над Пекіном

Тому в результаті дослідження для реалізації була обрана повністю згорточна нейронна мережа U-Net, яка дала найкращі результати серед розглянутих моделей нейронних мереж. Давайте пояснимо це, починаючи з поняття простої згорткової мережі.

Згорткові нейронні мережі (convnets) – це клас глибоких нейронних мереж, які найчастіше використовуються для аналізу візуальних зображень. На основі їх спільної вагової архітектури та властивостей, інваріантних до зміщення, вони також відомі як інваріантні до зсуву або простору штучні нейронні мережі. Вони мають застосування в розпізнаванні зображень і відео, системах рекомендацій, класифікації зображень, аналізі медичних зображень, обробці природної мови, інтерфейсах мозок-комп’ютер і фінансових часових рядах.

ZNM є регуляризованою версією багатошарового персептрона. Багатошаровий персептрон зазвичай означає повністю зв’язану мережу, що означає, що кожен нейрон одного шару з’єднаний з усіма нейронами наступного шару. «Повна зв’язність» цих мереж дозволяє легко їх перевчити.

Типова техніка регуляризації передбачає додавання певної форми вагової метрики до функції втрат. ШНМ використовують різні методи регуляризації: вони використовують ієрархічні представлення в даних і використовують менші та прості шаблони для складання більш складних шаблонів. Таким чином, SNM знаходиться на нижньому полюсі з точки зору масштабу актуальності та складності.

Згорткові мережі створені під впливом біологічних процесів, оскільки схеми зв'язку між нейронами подібні до організації зорової кори тварин. Окремі нейрони кори реагують на стимули лише в межах обмеженої ділянки поля зору, яка називається рецептивним полем. Рецептивні поля різних нейронів частково перекриваються, таким чином охоплюючи все поле зору.

Порівняно з іншими алгоритмами класифікації зображень, ШНМ використовують відносно невелику попередню обробку. Це означає, що мережа вивчає розроблені вручну фільтри за традиційними алгоритмами. Ця незалежність людини від попередніх знань і елементів дизайну є головною перевагою.

Назва згорткової нейронної мережі означає, що мережа використовує математичну операцію під назвою згортка. Згорточна мережа – це особливий тип нейронної мережі, яка використовує згортку принаймні в одному зі своїх шарів замість загального множення матриць.

Згорточна нейронна мережа складається з вхідного та вихідного шарів і багатьох прихованих шарів. Прихований шар HRM зазвичай складається з серії згорткових шарів, які згортаються за допомогою множення або інших скалярних добутків. Функція активації зазвичай є шаром ReLU, за яким слідують додаткові згортки, такі як агрегування, повністю зв'язаний і нормалізаційний шари. Ці шари називаються прихованими, оскільки їхні входи та виходи визначаються функцією активації та остаточною згорткою.

Давайте пояснимо концепцію рівня або випрямляча ReLU.У контексті штучних нейронних мереж випрямляч – це функція активації, яка визначається як позитивна частина його параметрів:

$$f(x) = x^x + \max(0, x)$$

де x – вхід до нейрона. Це є аналогом напівперіодичного випрямляча у схемотехніці.

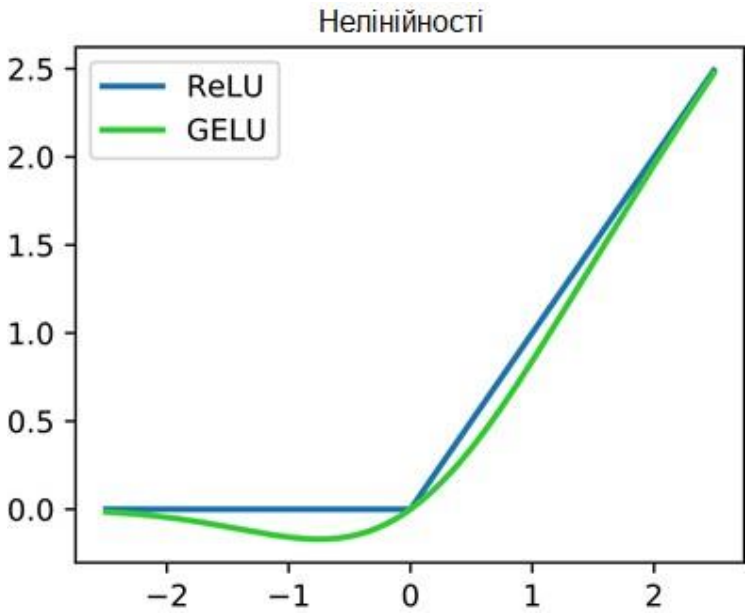


Рисунок 2.3 - Функція ReLU (синя) поблизу x = 0

Ця функція активації була вперше запропонована в динамічних мережах Hahnloser та ін. у 2000 році, і вона має сильну біологічну мотивацію та математичну основу. Порівняно з функціями активації, які зазвичай використовувалися до 2011 року, такими як логістична сигмоїда (натхненна теорією ймовірностей) і її більш практичний аналог гіперболічний тангенс, у 2011 році вперше було продемонстровано, що вона забезпечує кращу продуктивність для глибшого навчання мереж. Станом на 2017 рік

випрямляч є найпопулярнішою функцією активації в глибоких нейронних мережах.

Блок, який використовує випрямляч, також називається випрямленим лінійним блоком (ReLU).

Повернемося до концепції згорткових шарів у нейронних мережах. Вони повинні мати такі властивості:

• Ядро згортки, визначене шириною та висотою (гіперпараметри).

• Кількість вхідних і вихідних каналів (гіперпараметр).

• Глибина (вхідні канали) фільтра згортки має дорівнювати кількості каналів (глибині) вхідної карти ознак.

Згорткові шари перетворюють вхідні дані та передають результат до наступного шару. Це аналогічно тому, як нейрони зорової кори реагують на певний стимул. Кожен згортковий нейрон обробляє дані лише зі свого рецептивного поля. Хоча повністю зв’язані нейронні мережі прямого зв’язку можна використовувати для вивчення функцій і класифікації даних, застосовувати цю архітектуру до зображень непрактично. Навіть у неглибоких архітектурах потрібна дуже велика кількість нейронів, оскільки вхідні розміри, пов’язані з зображеннями, дуже великі, де кожен піксель є відповідною змінною. Наприклад, повністю зв’язаний шар із (маленьких) 100 × 100 зображень має 10 000 ваг для кожного нейрона другого шару. Операція згортки забезпечує вирішення цієї проблеми, оскільки вона зменшує кількість вільних параметрів, дозволяючи мережі стати глибшою з меншою кількістю параметрів. Наприклад, для графіків 5 × 5, кожен з однаковими загальними вагами, незалежно від розміру зображення, потрібно вивчити лише 25 параметрів. Використовуючи вагові коефіцієнти регуляризації для меншої кількості параметрів, можна уникнути проблем із зникаючим градієнтом і викидом, які виникають при зворотному поширенні в традиційних нейронних мережах.

Зворотне поширення, або скорочено «зворотне поширення помилок», – це контрольований алгоритм навчання з використанням штучних нейронних мереж із градієнтним спуском. Враховуючи штучну нейронну мережу та функцію помилок, цей метод обчислює градієнт функції помилок відносно ваг нейронної мережі. «Зворотна» частина назви походить від обчислення градієнта у зворотному напрямку через мережу, спочатку обчислення градієнта остаточного вагового шару і, нарешті, обчислення градієнта першого вагового шару. Під час обчислення градієнта попереднього шару частина обчислення градієнта одного шару використовується повторно. Цей зворотний потік дезінформації дозволяє ефективно обчислювати градієнти на кожному шарі порівняно з простим підходом обчислення градієнтів для кожного шару окремо.

Згорткові мережі знаходяться в авангарді досягнень у розпізнаванні. Штучні нейронні мережі не тільки покращили класифікацію цілих зображень, але й досягли прогресу в локальних завданнях зі структурованими результатами. До них належать часткові та ключові прогнози та локальне зіставлення. Наступним природним кроком від грубого висновку до точного є попіксельне передбачення.

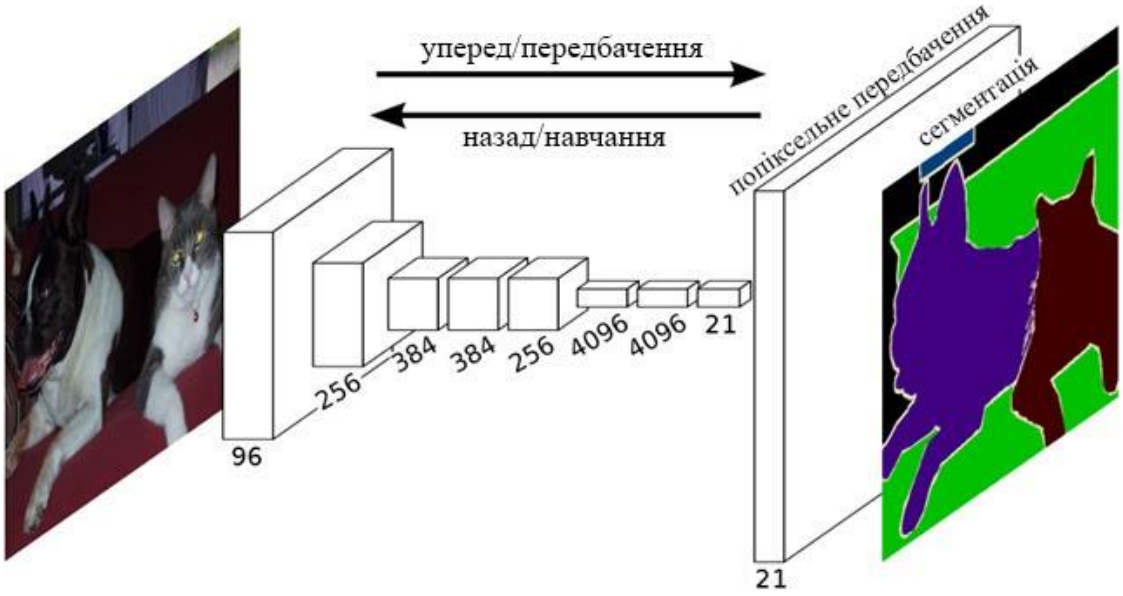


Рисунок 2.4 - Приклад семантичної сегментації пікселів зображень за допомогою згорткової нейромережі

Лонг та ін. показують, що повністю згорточні мережі, навчені семантичній сегментації, перевершують найсучасніші без додаткових механізмів. Повністю згорнуті версії існуючих мереж передбачають щільні виходи від вхідних даних довільного розміру. Виконуйте тренування та висновки для всього зображення одночасно, використовуючи інтенсивні обчислення прямого та зворотного поширення. Рівні збільшення частоти дискретизації в мережі дозволяють прогнозувати на рівні пікселів і навчатися в мережах із зменшенням дискретизації.

Повністю згорточні версії існуючих мереж припускають щільні виходи від входів довільного розміру. Виконуйте тренування та висновки для всього зображення одночасно, використовуючи інтенсивні обчислення прямого та зворотного поширення. Рівні підвищення дискретизації в мережі підтримують прогнозування на рівні пікселів і навчання в мережах із недостатньою дискретизацією. Цей підхід ефективний як асимптотично, так і абсолютно, і усуває потребу в ускладненнях в іншій роботі.

Кожен шар даних у SNM є тривимірним масивом розміром h × w × d, де h і w – просторові розміри, а d – функція або розмір каналу. Перший шар – це зображення з розміром пікселів h × w і d колірні канали. Розташування на вищих шарах відповідають розташуванням на зображенні, і вони з'єднані з цими місцями шляхами, які називаються інтегральними полями. ЗНМ побудована на незмінності під час руху. Їх основні компоненти (функції згортання, агрегації та активації) працюють на локальних вхідних областях і покладаються лише на відносні просторові координати. Записуючи xij для вектора даних у (i, j) на даному шарі та yij для наступного шару, ці функції використовують обчислений результат yij

$$y_{ij} = f_{ks}(\{x_{si+\delta i,sj+\delta j}\}_{0<\delta i,\delta j<k})$$

де k називається розміром ядра, s – коефіцієнт кроку або субдискретизації, а fks визначає тип шару: множення матриці для згортки або середнього агрегування, просторовий максимум для максимального агрегування або елементна нелінійність для функції активації тощо для інших типів шарів.

Ця функціональна форма підтримується за складом, причому розмір ядра та крок підкоряються правилу трансформації

$$g_{k's'} = (f - g)_{k'}$$

Хоча загальні глибокі мережі обчислюють загальні нелінійні функції, ця форма мережевих обчислень лише на рівні називається глибоким фільтром або нелінійним фільтром повністю згорткової мережі. Повністю згорткові мережі працюють із входами довільного розміру та створюють вихідні дані відповідних просторових розмірів. Справжня функція втрат, що

складається з повністю згорткової мережі, визначає завдання. Якщо функція втрат є сумою просторових розмірів кінцевого шару, $\ell(x; 8) = \sum ij\ \ell'(xij; 8)$, то її градієнт буде сумою градієнтів кожного з його просторів.

елемент. Таким чином, стохастичний градієнтний спад на ℓ, обчислений для всього зображення, буде таким самим, як і стохастичний градієнтний спад на ℓ', приймаючи

Усі рецептивні поля останнього шару як невеликий пакет. Коли ці сприйнятливі поля значно перекриваються, пряме та зворотне розповсюдження є обчислювально ефективнішим завдяки виконанню пошарових обчислень для всього зображення, а не з використанням окремих частин зображення.

Сама U-Net є згортковою нейронною мережею, розробленою для сегментації біомедичних зображень кафедрою комп'ютерних наук Фрайбурзького університету, Німеччина. Мережа базується на повністю згортковій мережі, архітектуру якої було змінено та розширено, щоб використовувати менше навчальних зображень і отримати точнішу сегментацію. Щоб сегментувати зображення 512 × 512 на сучасному GPU, потрібно менше секунди.

Архітектура U-Net походить від так званої «повністю згорткової мережі», вперше запропонованої Лонгом, Шеллхамером і Даррелом.

Основна ідея полягає в тому, щоб доповнити традиційні контрактні мережі послідовними шарами, де операції з'єднання замінюються операціями вибірки. Таким чином, ці шари збільшують роздільну здатність виходу. Крім того, послідовні згорточні шари можуть навчитися збирати точні результати на основі цієї інформації.

Важливою модифікацією U-Net є те, що передискретизація має велику кількість каналів функцій, що дозволяє мережі поширювати контекстну інформацію на вищому рівні роздільної здатності. В результаті шлях

розширення є більш-менш симетричним до частини, що скорочується, і дає U-подібну структуру. Мережа використовує лише ефективну частину кожної згортки, немає повністю пов’язаних шарів. Щоб передбачити пікселі в областях меж зображення, відсутній контекст екстраполюється за допомогою дзеркального відображення вхідного зображення. Ця стратегія мозаїки важлива для застосування мережі до великих зображень, інакше роздільна здатність буде обмежена пам’яттю GPU.

U-Net створили Олаф Роннебергер, Філіп Фішер, Томас Брокс у статті «U-Net: згорткові мережі для сегментації біомедичних зображень» у 2015 році. Це вдосконалення та розвиток роботи Евана Шелхамера, Джонатана Лонга та Тревора Даррелла «Повністю згорткові мережі для семантичної сегментації» (2014).

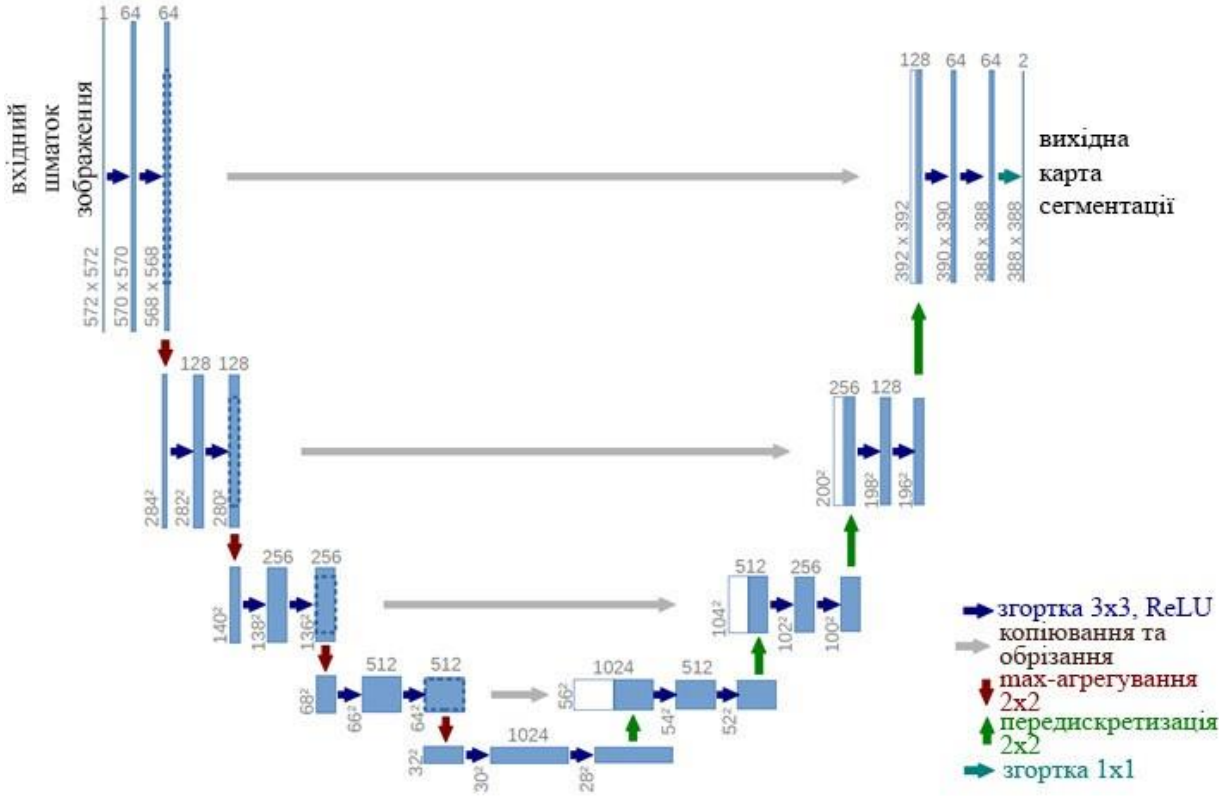


Рисунок 2.5 - Архітектура моделі U-Net

Архітектура U-Net виглядає як буква «U», що виправдовує її назву. Архітектура складається з трьох частин: стиснення, вузьке місце та розширення. Стиснута частина складається з багатьох термоусадочних блоків. Два шари згорток 3 × 3 застосовуються до кожного блоку, який приймає вхідні дані, з наступним агрегуванням 2 × 2 max. Кількість ядер або карт функцій після кожного блоку подвоюється, щоб архітектура могла ефективно вивчати складні структури. Найнижчий шар є проміжним шаром між шарами стиснення та розширення. Він використовує двошарову ШНМ 3 × 3, за якою слідує згортковий шар 2 × 2.

Але фундамент цієї архітектури лежить у прибудовній частині. Як і компресійний шар, він також складається з кількох блоків розширення. Кожен блок подає вхідні дані на два шари ШНМ 3 × 3, за якими слідує рівень масштабування 2 × 2. Крім того, після кожного блоку кількість карт функцій, які використовуються згортковими шарами, зменшується вдвічі для підтримки симетрії. Однак щоразу вхідні дані також додаються до карти функцій відповідного шару стиснення. Ця операція гарантує, що функції, отримані під час стиснення зображення, будуть використані для реконструкції зображення. Кількість розширювальних блоків така ж, як і кількість термоусадочних блоків. Після цього сформовані карти проходять ще один шар SNM 3×3 з кількістю карт об’єктів, що дорівнює кількості необхідних фрагментів.

U-Net використовує досить нову схему зважування втрат для кожного пікселя: більше ваг на межах сегментованих об’єктів. Ця схема зважування втрат допомагає моделі U-Net сегментувати клітини на біомедичних зображеннях безперервно, щоб окремі клітини можна було легко ідентифікувати на картах бінарної сегментації. Спочатку застосуйте піксельне softmax до згенерованого зображення,

Супроводжується крос-ентропійною функцією. Тоді кожен піксель належить до одного з класів. Ідея полягає в тому, що навіть із сегментацією кожен піксель має належати до певного класу, і вам потрібно переконатися, що він належить до певного класу. Таким чином, проблема сегментації перетворюється на задачу багатокласової класифікації та працює дуже добре порівняно з традиційними функціями втрат.

Розширення даних має вирішальне значення для навчальних мереж із бажаною інваріантністю та надійністю, коли доступно лише кілька навчальних зразків. Зокрема, випадкова пружна деформація навчальних зразків є ключовою концепцією для навчання мереж сегментації з дуже невеликою кількістю анотованих зображень. В U-Net плавна деформація генерується за допомогою випадкових векторів переміщення на грубій сітці 3 × 3. Зміщення вибираються з розподілу Гауса зі стандартним відхиленням 10 пікселів. Потім за допомогою бікубічної інтерполяції обчислюється зсув на піксель. Рівень вилучення в кінці шляху стиснення виконує подальше неявне збільшення даних.

Отже, в результаті методу дослідження згорточна нейронна мережа U-Мережа для цілей розпізнавання образів. Існуючі методи демонструють деякі незадовільні характеристики, а саме низьку точність результатів розпізнавання, низьку швидкість навчання та використання великого місця на жорсткому диску.

В результаті дослідження мови програмування було обрано Python, який забезпечує розгалужену структуру для розпізнавання образів за допомогою нейронних мереж, а також зручний для читання код.

У результаті бібліотечного дослідження були обрані розширення TensorFlow, Folium і Jupyter Notebook, які найбільше відповідають деталям програми, що розробляється.

Репозиторій Google Earth Engine, використаний для розробки програми, виявився найкращим публічним архівом даних.

Розділ 3 описує етап розробки програми розпізнавання образів та її алгоритмів.

3 РОЗРОБКА СИСТЕМИ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ АЕРОФОТОЗЙОМКИ

Як описано в розділі 2, створення справжньої програми розпізнавання образів є складним процесом, який вимагає широких математичних знань і навичок програмування. Це завдання можна розділити на 3 менші завдання: налаштування набору даних для навчання та оцінювання, впровадження та навчання моделі U-Net, прогнозування та виведення. Розглянемо загальний алгоритм програми, розробленої для розпізнавання образів за результатами аерофотозйому, і кожен етап розробки докладніше.

3.1 Алгоритм розпізнавання зображень аерофотозйомки

Мета програми розпізнавання образів, яку я розробляю, полягає в тому, щоб передбачити непроникність міста на основі повітряної (у моєму випадку супутникової) фотографії місцевості. Непроникні поверхні – це переважно штучні конструкції, такі як покриті водою тротуари (дороги, тротуари, під'їзні шляхи та стоянки, а також промислові зони, такі як аеропорти, порти та логістичні центри розподілу, де використовується багато бруківки) – корозія - стійкі матеріали, особливо асфальт, бетон, цегла, камінь і плитка. Грунт, ущільнений міською забудовою, також є дуже водонепроникним. У нашому випадку ми хочемо передбачити безперервний вихід із групи безперервних вхідних даних. Вхідним сигналом буде складене зображення із супутника Landsat 8, а виходом – непроникна область. Для розробки додатка я буду використовувати Національну базу даних земельного покриву США (NLCD) як найкращий доступний ресурс, який також є частиною загальнодоступного набору даних Earth Engine[7].

Загальний потік програми розпізнавання образів можна описати так:

• Імпортуйте зображення, які використовуватимуться для введення даних.

• Підготуйте відповідь (що потрібно передбачити; тут непроникна поверхня).

• Компіляція вхідних зображень і зображень відповідей.

• Відбір зразків у стек у заздалегідь визначених місцях.

• Впровадження прогнозної моделі U-Net.

• Модельне навчання.

• Виконайте прогнози на вказаному наборі даних.

• показати результат.

Далі частини цього алгоритму будуть розглянуті більш детально.

3.2 Вибір та підготовка датасету

Програма починає імпортувати вхідне зображення. Як згадувалося, ми використовували три роки безхмарних композитних зображень Landsat 8:

```
# Specify inputs (Landsat bands) to the model and the response variable. opticalBands = [‘B1’, ’B2’, ’B3’, ’B4’, ’B5’, ’B6’, ’B7’]
thermalBands = [‘B10’, ’B11’]
# Use Landsat 8 surface reflectance data.
l8sr = ee.ImageCollection(‘LANDSAT/LC08/C01/T1_SR’)
# Cloud masking function. def maskL8sr(image):
cloudShadowBitMask = ee.Number(2).pow(3).int() cloudsBitMask = ee.Number(2).pow(5).int()
qa = image.select(‘pixel_qa’)
```

```
        mask1 = qa.bitwiseAnd(cloudShadowBitMask).eq(0).And(
qa.bitwiseAnd(cloudsBitMask).eq(0))
        mask2 = image.mask().reduce('min')
        mask3 = image.select(opticalBands).gt(0).And(
image.select(opticalBands).lt(10000)).reduce('min')
        mask = mask1.And(mask2).And(mask3)
        return image.select(opticalBands).divide(10000).addBands(
image.select(thermalBands).divide(10).clamp(273.15, 373.15)
        .subtract(273.15).divide(100)).updateMask(mask)
        # The image input data is a cloud-masked median composite.
        image = l8sr.filterDate('2015-01-01', '2017-12-31').map(maskL8sr).median()
        # Use folium to visualize the imagery.
        mapid = image.getMapId({'bands': ['B4', 'B3', 'B2'], 'min': 0, 'max': 0.3})
map = folium.Map(location=[38., -122.5])
        folium.TileLayer( tiles=mapid['tile_fetcher'].url_format,
        attr='Map Data &copy; <a href="https://earthengine.google.com/">Google
Earth Engine</a>',
        overlay=True, name='median composite',).add_to(map)
        mapid = image.getMapId({'bands': ['B10'], 'min': 0, 'max': 0.5})
folium.TileLayer(
        tiles=mapid['tile_fetcher'].url_format,
        attr='Map Data &copy; <a href="https://earthengine.google.com/">Google
Earth Engine</a>',
        overlay=True, name='thermal',).add_to(map)
map.add_child(folium.LayerControl())
```

Діапазон частот, отриманий супутниками Landsat:

Діапазон	Довжина хвилі (мкм)	Роздільна здатність (м)
1 - Coastal aerosol	0.43-0.45	30
2 - Blue	0.45-0.51	30
3 - Green	0.53-0.59	30
4 - Red	0.64-0.67	30
5 - Near Infrared (NIR)	0.85-0.88	30
6 - SWIR 1	1.57-1.65	30
7 - SWIR 2	2.11-2.29	30
10 - Thermal Infrared (TIRS) 1	10.6-11.19	100
11 - Thermal Infrared (TIRS) 2	11.50-12.51	100

Складене вхідне зображення моделі рендериться за допомогою бібліотеки Folium Python. Ось інтерактивна карта, яка виглядає так:

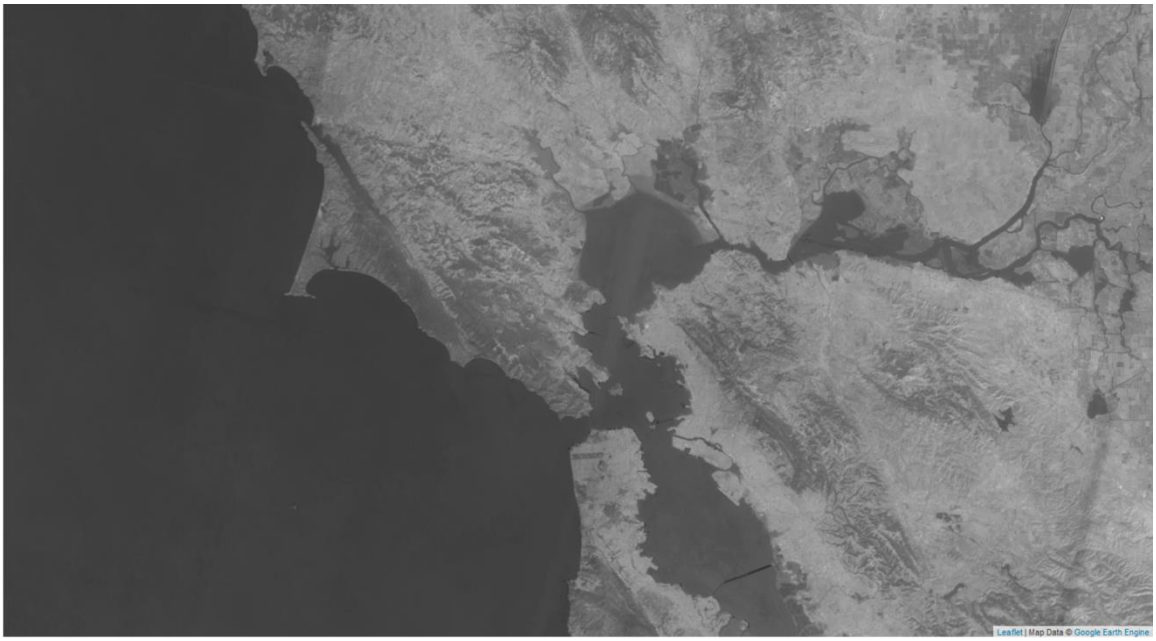


Рисунок 3.1 - Фрагмент візуалізації зображення від супутнику Landsat 8

Після цього ми також готуємо відповідь для навчання нашої моделі. Його візуалізацію можна побачити нижче:

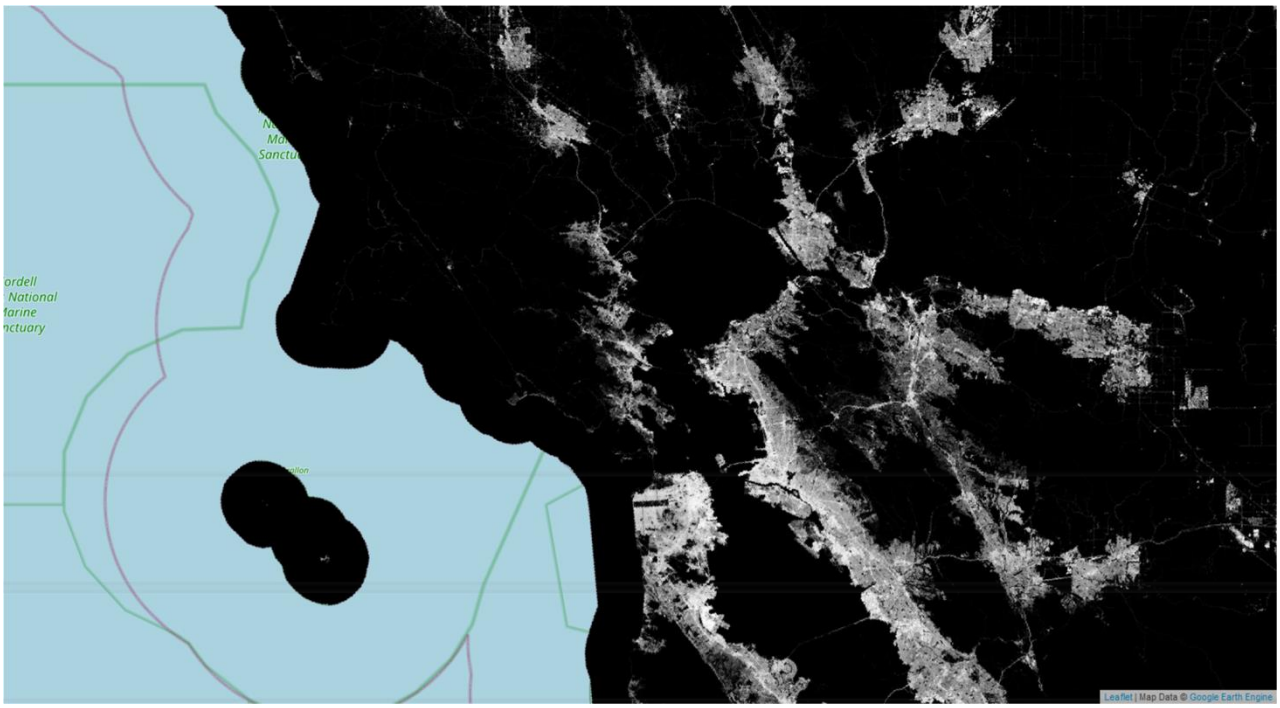


Рисунок 3.2 - Фрагмент візуалізації зображень з NLCD

На цьому зображенні повністю непроникні ділянки пофарбовані в білий колір.

Потім ці 2D-зображення (композитне зображення Landsat і непроникна поверхня NLCD) потрібно об’єднати, щоб створити одне зображення для вибірки. Для цього ми перетворюємо зображення на зображення масиву, де кожен піксель зберігає 256 x 256 піксельних блоків для кожної смуги. Пізніше це буде експортовано в Google Cloud Storage.

Синтетичні зображення слід вибирати в стратегічних місцях, які містять найрізноманітніші зразки непроникних міських територій, на яких може працювати наша модель. Зокрема, це вибрані вручну полігони поверхні з частотою дискретизації 256 x 256. Прямокутники навчання та оцінювання показані нижче. Освітні області позначені червоним, а оцінювальні – синім.

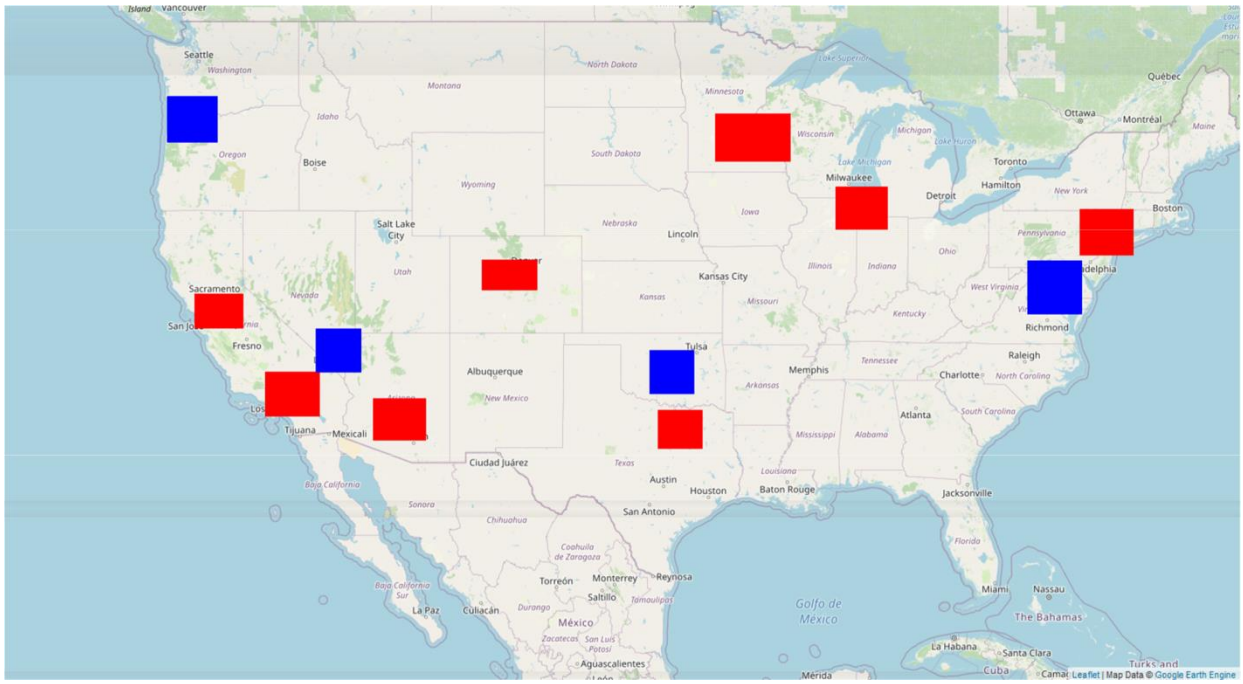


Рисунок 3.3 - Навчальні та оцінювальні мітки для моделі

Ключовим кроком є вибірка масиву зображень у точках, щоб отримати всі пікселі в області 256 x 256 у кожній точці. Зокрема, для створення навчальних і тестових даних ми експортуємо файл TFRecord, який містить блоки значень пікселів у кожному записі[8].

eval_patches_g0.tfrecord.gz	1.7 GB
eval_patches_g1.tfrecord.gz	2.1 GB
eval_patches_g2.tfrecord.gz	1.9 GB
eval_patches_g3.tfrecord.gz	1.9 GB
training_patches_g0.tfrecord.gz	1.7 GB
training_patches_g1.tfrecord.gz	2 GB
training_patches_g2.tfrecord.gz	2 GB
training_patches_g3.tfrecord.gz	2.1 GB
training_patches_g4.tfrecord.gz	2.1 GB
training_patches_g5.tfrecord.gz	2.1 GB
training_patches_g6.tfrecord.gz	1.5 GB
training_patches_g7.tfrecord.gz	1.6 GB

Рисунок 3.4 - Датасет TFrecord для моделі в Google Cloud Storage

Оскільки кожен запис може містити велику кількість даних (особливо великі блоки зображень або багато вхідних смуг), потрібна ручна обробка, щоб уникнути помилок «обчислене значення надто велике». Зокрема, у кожній геометрії було взято декілька менших зразків, а результати об'єднано для створення єдиного експорту.

Convert the feature collections to lists for iteration. trainingPolysList = trainingPolys.toList(trainingPolys.size()) evalPolysList = evalPolys.toList(evalPolys.size())

These numbers determined experimentally. n = 200 # Number of shards in each polygon.

N = 2000 # Total sample size in each polygon.

Export all the training data (in many pieces), with one task

per geometry.

for g in range(trainingPolys.size().getInfo()): geomSample = ee.FeatureCollection([])

for i in range(n):

sample = arrays.sample(

region = ee.Feature(trainingPolysList.get(g)).geometry(), scale = 30,

numPixels = N / n, # Size of the shard. seed = i,

tileScale = 8

)

geomSample = geomSample.merge(sample) desc = TRAINING_BASE + '_g' + str(g)

task = ee.batch.Export.table.toCloudStorage(

collection = geomSample, description = desc, bucket = BUCKET,

```
fileNamePrefix = FOLDER + '/' + desc, fileFormat = 'TFRecord',
selectors = BANDS + [RESPONSE]
)
task.start()
# Export all the evaluation data.
for g in range(evalPolys.size().getInfo()): geomSample = ee.FeatureCollection([])  for i in range(n):
sample = arrays.sample(
region = ee.Feature(evalPolysList.get(g)).geometry(), scale = 30,
numPixels = N / n, seed = i, tileScale = 8
)
geomSample = geomSample.merge(sample)
desc = EVAL_BASE + '_g' + str(g)
task = ee.batch.Export.table.toCloudStorage( collection = geomSample,
description = desc, bucket = BUCKET,
fileNamePrefix = FOLDER + '/' + desc, fileFormat = 'TFRecord',
selectors = BANDS + [RESPONSE]
)
task.start()
```

Після цього кроку дані, експортовані з Earth Engine, завантажуються в набір даних TensorFlow[9]. Повторіть цей процес для навчальних і оціночних зразків:

```
def get_training_dataset():
"""Get the preprocessed training dataset Returns:
```

```
A tf.data.Dataset of training data. """
glob = 'gs://' + BUCKET + '/' + FOLDER + '/' + TRAINING_BASE + '*'
dataset = get_dataset(glob)
dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE).repeat() return dataset

def get_eval_dataset():
"""Get the preprocessed evaluation dataset Returns:
A tf.data.Dataset of evaluation data. """
glob = 'gs://' + BUCKET + '/' + FOLDER + '/' + EVAL_BASE + '*'
dataset = get_dataset(glob)  dataset = dataset.batch(1).repeat() return dataset
```

Набір даних – це двигун TensorFlow, який представляє потенційно великий набір елементів. Ці набори даних будуть використані для навчання моделі пізніше.

У наведеному вище коді gs:// – це внутрішня файлова система, яку використовує Google Cloud Storage. Більшість ресурсів посилаються безпосередньо, додаючи префікс gs:// до їх відносного шляху. Після завантаження даних нам потрібно створити модель для обробки наборів даних навчання та оцінювання.

3.3 Реалізація і навчання моделі U-Net

Ця програма розпізнавання образів використовує реалізацію Keras моделі U-Net. Модель U-Net приймає фрагмент зображення розміром 256x256 пікселів як вхідні дані та виводить попіксельні ймовірності класу,

мітки або безперервний вихід. Модель може бути реалізована по суті без модифікації, але необхідно використовувати середньоквадратичну втрату помилки на сигмоїдальному виведенні, оскільки розглядається проблема регресії, а не проблема класифікації. Ці дві проблеми мають на меті побудувати стислу модель, яка передбачає значення властивостей залежностей від властивостей змінних. Різниця між цими двома завданнями полягає в тому, що властивості залежності є числовими для регресії та категоріальними для класифікації. Оскільки фракція непроникної поверхні обмежена [0,1], з багатьма значеннями, близькими до нуля або одиниці, доцільною є насичена функція активації.

Модель реалізована за допомогою чотирьох функцій: conv_block (реалізація блоку згортки), encoder_block (реалізація блоку кодера), decoder_block (реалізація блоку декодера) і get_model (загальне представлення моделі).

Кожен згортковий блок застосовує два згорткові шари 3 x 3 із функціями активації ReLU. Процес нормалізації пакетів відбувається між ними. Пакетна нормалізація (також відома як пакетна нормалізація) – це техніка, яка використовується, щоб зробити штучні нейронні мережі швидшими та стабільнішими шляхом нормалізації вхідного рівня шляхом повторного центрування та зміни масштабу. Він координує оновлення кількох рівнів у моделі шляхом масштабування виходу шарів, зокрема шляхом нормалізації активації кожної вхідної змінної в невеликий пакет, наприклад, активації вузла з попереднього рівня. Стандартизація тут стосується масштабування даних із середнім значенням 0 і стандартним відхиленням 1, наприклад, використання розподілу Гауса. Нормалізація активації попереднього рівня означає, що припущення наступного рівня щодо поширення та розподілу вхідних даних під час оновлення ваг не зміняться, принаймні не суттєво. Це має ефект стабілізації та прискорення

процесу навчання нейронної мережі. Нормалізація вхідних даних до шару впливає на навчання моделі, значно зменшуючи кількість необхідних епох. Він також може діяти як регуляризатор, зменшуючи помилку узагальнення, подібно до використання регуляризації активації. Хоча мотивацією для розробки цього методу було зменшення «внутрішнього коваріаційного зміщення», було припущено, що пакетна нормалізація є більш ефективною, оскільки вона згладжує та, у свою чергу, спрощує функцію оптимізації, яка вирішується під час навчання мережі.

```
def conv_block(input_tensor, num_filters):
encoder = layers.Conv2D(num_filters, (3, 3), padding='same')(input_tensor)
encoder = layers.BatchNormalization()(encoder)
encoder = layers.Activation('relu')(encoder)
encoder = layers.Conv2D(num_filters, (3, 3), padding='same')(encoder)
encoder = layers.BatchNormalization()(encoder)
encoder = layers.Activation('relu')(encoder) return encoder
```

Згорткові шари в згорткових нейронних мережах систематично застосовують навчені фільтри до вхідних зображень для створення карт функцій, які підсумовують присутність цих функцій у вхідних даних.

Згорткові шари виявилися дуже ефективними, а накопичення згорткових шарів у глибокій моделі дозволяє шарам, розташованим поблизу входу, вивчати функції низького рівня (наприклад, лінії), а шарам, розташованим глибше в моделі, вивчати функції вищого порядку або більш абстрактні функції, такі як фігури. або конкретні предмети.

Одним з обмежень вихідних даних карти об'єктів згорткових шарів є те, що вони записують точне розташування вхідних об'єктів. Це означає, що

невеликі переміщення позицій об’єктів у вхідному зображенні призведуть до різних відображень об’єктів. Це може статися, коли вхідне зображення повторно обрізається, повертається, зміщується та вноситься інші незначні зміни.

Загальноприйнятий метод вирішення цієї проблеми в обробці сигналів називається зниженням дискретизації. Це створює версію вхідного сигналу з низькою роздільною здатністю, яка все ще містить великі або важливі структурні елементи, без дрібних деталей, які можуть бути некорисними для завдання.

Згорткові шари можуть зменшити роздільну здатність шляхом додавання нового шару (шару агрегації), особливо після застосування нелінійностей (таких як ReLU у нашому випадку) до карт функцій, виведених згортковими шарами.

Додавання рівня агрегації після згорткового шару – це загальний шаблон, який використовується для впорядкування шарів у згортковій нейронній мережі, який може повторюватися один або кілька разів у даній моделі.

Рівень агрегації діє на кожну карту функцій окремо, щоб створити новий набір із такою ж кількістю об’єднаних карт функцій.

Агрегація передбачає вибір операції агрегації для застосування до цільової карти, подібно до фільтра. Операції агрегації або фільтрації менші за розміром, ніж карти об’єктів; зокрема, майже завжди використовується 2 × 2 із кроком у 2 пікселі.

Це означає, що рівень агрегації завжди зменшуватиме розмір кожної карти функцій у 2 рази, наприклад, зменшуючи кожен вимір наполовину, зменшуючи кількість пікселів або значень у кожній карті функцій до чверті розміру. Наприклад, шар агрегації, застосований до карти об’єктів розміром

6 × 6 (36 пікселів), створить об’єднану карту об’єктів розміром 3 × 3 (9 пікселів).

Операції агрегації визначаються дослідниками, а не вивчаються мережею. Дві поширені функції, які використовуються в операціях об’єднання:

• Об’єднання середніх значень: обчислює середнє значення кожного зрізу на карті об’єктів.

• Максимальне об’єднання: обчислює максимальне значення кожного блоку на карті об’єктів.

Результатом використання шару агрегації та створення карти об’єктів зі зниженою роздільною здатністю є загальна версія об’єктів, виявлених у вхідних даних. Вони корисні, оскільки невеликі зміни в розташуваннях вхідних об’єктів, виявлені згортковим шаром, призведуть до створення агрегованої карти об’єктів з однаковим розташуванням. Ця можливість, додана агрегацією, називається інваріантністю моделі до локального зміщення.

У нашому випадку блок кодера використовує максимальну агрегацію.

Метою блоку кодера є кодування вхідного зображення в представлення об’єкта на кількох різних рівнях.

```
def encoder_block(input_tensor, num_filters): encoder = conv_block(input_tensor, num_filters)
encoder_pool = layers.MaxPooling2D((2, 2), strides=(2, 2))(encoder) return encoder_pool, encoder
```

Блок декодера є другою половиною архітектури U-Net. Його мета полягає в семантичному проектуванні дискримінаційних ознак (нижча

роздільна здатність), отриманих кодером, на простір пікселів (вища роздільна здатність) для щільної класифікації. Декодер включає передискретизацію та стиснення з подальшою традиційною деконволюцією.

```
def decoder_block(input_tensor, concat_tensor, num_filters):
decoder = layers.Conv2DTranspose(num_filters, (2, 2), strides=(2, 2), padding= ‘same’)(input_tensor)
decoder = layers.concatenate([concat_tensor, decoder], axis=-1) decoder = layers.BatchNormalization()(decoder)
decoder = layers.Activation(‘relu’)(decoder)
decoder = layers.Conv2D(num_filters, (3, 3), padding=‘same’)(decoder) decoder = layers.BatchNormalization()(decoder)
decoder = layers.Activation(‘relu’)(decoder)
decoder = layers.Conv2D(num_filters, (3, 3), padding=‘same’)(decoder) decoder = layers.BatchNormalization()(decoder)
decoder = layers.Activation(‘relu’)(decoder) return decoder

decoder = layers.Activation(‘relu’)(decoder) return decoder
```

Функція get_model () збирає блоки відповідно до архітектури, описаної в розділі 2.3.

```
def get_model():
inputs = layers.Input(shape=[None, None, len(BANDS)]) # 256 encoder0_pool, encoder0 = encoder_block(inputs, 32) # 128 encoder1_pool, encoder1 = encoder_block(encoder0_pool, 64) # 64 encoder2_pool, encoder2 = encoder_block(encoder1_pool, 128) # 32 encoder3_pool, encoder3 = encoder_block(encoder2_pool, 256) # 16 encoder4_pool, encoder4 =
```

```
encoder_block(encoder3_pool, 512) # 8 center = conv_block(encoder4_pool, 1024) # center
    decoder4 = decoder_block(center, encoder4, 512) # 16 decoder3 = decoder_block(decoder4, encoder3, 256) # 32 decoder2 = decoder_block(decoder3, encoder2, 128) # 64 decoder1 = decoder_block(decoder2, encoder1, 64) # 128 decoder0 = decoder_block(decoder1, encoder0, 32) # 256
    outputs = layers.Conv2D(1, (1, 1), activation='sigmoid')(decoder0) model = models.Model(inputs=[inputs], outputs=[outputs]) model.compile(
    optimizer=optimizers.get(OPTIMIZER), loss=losses.get(LOSS),
    metrics=[metrics.get(metric) for metric in METRICS]) return model
```

Продовжуючи реалізацію, нам потрібно навчити реалізовану модель Keras, викликавши метод .fit(). Для цієї реалізації ми навчимо модель протягом 10 епох, що означає, що робочий алгоритм обійде весь навчальний набір даних 10 разів.

```
    m = get_model() m.fit(
    x=training, epochs=10,
    steps_per_epoch=int(TRAIN_SIZE / BATCH_SIZE), validation_data=evaluation, validation_steps=EVAL_SIZE)
```

Модель є реалізацією, побудованою на API Keras і працює лише з зображеннями в певному форматі TFRecord (запис TensorFlow). Формат запису TFR

це простий формат для зберігання послідовностей двійкових записів. Він оптимізований для TensorFlow кількома способами. По-перше, це

полегшує поєднання кількох наборів даних і бездоганно інтегрується з можливостями імпорту та попередньої обробки даних, які надає бібліотека. Це перевага, особливо для наборів даних, які занадто великі, щоб повністю поміститися в пам’ять, оскільки з диска завантажуються та обробляються лише дані (наприклад, пакети), необхідні на даний момент. Ще одна головна перевага TFRecords полягає в тому, що дані про послідовність, такі як часові ряди або кодування слів, можна зберігати таким чином, щоб забезпечити дуже ефективний і (з точки зору кодування) простий імпорт таких даних.

3.4 Тестування та оптимізація навченої моделі

Тепер, коли ми успішно визначили набір даних для навчання та оцінки моделі, розгорнули модель і навчили її, настав час самостійно запустити прогнози.

Процес прогнозування виглядає наступним чином:

• Зображення, передбачені Earth Engine, можна експортувати у хмарне сховище у форматі TFRecord.

• Навчена модель використовується для прогнозування.

• Прогнози записуються у файли TFRecord у хмарному сховищі.

• Файли прогнозів у форматі TFRecord завантажуються в Earth Engine.

Наступна функція обробляє цей процес:

```
def doExport(out_image_base, kernel_buffer, region): """Run the image export task. Block until complete. """
    task = ee.batch.Export.image.toCloudStorage( image = image.select(BANDS),
    description = out_image_base, bucket = BUCKET,
```

```
        fileNamePrefix = FOLDER + '/' + out_image_base, region = region.getInfo()['coordinates'],  scale = 30,
        fileFormat = 'TFRecord', maxPixels = 1e10, formatOptions = {
        'patchDimensions': KERNEL_SHAPE, 'kernelSize': kernel_buffer, 'compressed': True, 'maxFileSize': 104857600
        }
        )
        task.start()

        # Block until the task completes.
        print('Running image export to Cloud Storage...') import time
        while task.active(): time.sleep(30)
        # Error condition
        if task.status()['state'] != 'COMPLETED': print('Error with image export.')
        else:
        print('Image export completed.')

        def doPrediction(out_image_base, user_folder, kernel_buffer, region): """Perform inference on exported imagery, upload to Earth Engine. """
        print('Looking for TFRecord files...')
        # Get a list of all the files in the output bucket. filesList = !gsutil ls 'gs://'{BUCKET}'/'{FOLDER}
        # Get only the files generated by the image export. exportFilesList = [s for s in filesList if out_image_base in s]
        # Get the list of image files and the JSON mixer file. imageFilesList = []
        jsonFile = None
```

```
for f in exportFilesList:
if f.endswith('.tfrecord.gz'): imageFilesList.append(f)
elif f.endswith('.json'): jsonFile = f
# Make sure the files are in the right order. imageFilesList.sort()
from pprint import pprint pprint(imageFilesList) print(jsonFile)
import json
# Load the contents of the mixer file to a JSON object. jsonText = !gsutil cat {jsonFile}
# Get a single string w/ newlines from the IPython.utils.text.SList mixer = json.loads(jsonText.nlstr)
pprint(mixer)
patches = mixer['totalPatches']
# Get set up for prediction. x_buffer = int(kernel_buffer[0] / 2) y_buffer = int(kernel_buffer[1] / 2) buffered_shape = [
KERNEL_SHAPE[0] + kernel_buffer[0], KERNEL_SHAPE[1] + kernel_buffer[1]]
imageColumns = [
tf.io.FixedLenFeature(shape=buffered_shape, dtype=tf.float32) for k in BANDS
]

imageFeaturesDict = dict(zip(BANDS, imageColumns))

def parse_image(example_proto):
return tf.io.parse_single_example(example_proto, imageFeaturesDict)

def toTupleImage(inputs):
```

```
inputsList = [inputs.get(key) for key in BANDS] stacked = tf.stack(inputsList, axis=0)
stacked = tf.transpose(stacked, [1, 2, 0]) return stacked

# Create a dataset from the TFRecord file(s) in Cloud Storage.
imageDataset = tf.data.TFRecordDataset(imageFilesList, compression_type='GZIP'
)
imageDataset = imageDataset.map(parse_image, num_parallel_calls=5) imageDataset = imageDataset.map(toTupleImage).batch(1)
# Perform inference. print('Running predictions...')
predictions = m.predict(imageDataset, steps=patches, verbose=1)
# print(predictions[0])

print('Writing predictions...')
out_image_file = 'gs://' + BUCKET + '/' + FOLDER + '/' + out_image_base + '.TF Record'
writer = tf.io.TFRecordWriter(out_image_file) patches = 0
for predictionPatch in predictions: print('Writing patch ' + str(patches) + '...') predictionPatch = predictionPatch[
x_buffer:x_buffer+KERNEL_SIZE, y_buffer:y_buffer+KERNEL_SIZE]

# Create an example. example = tf.train.Example(
features=tf.train.Features( feature={
'impervious': tf.train.Feature( float_list=tf.train.FloatList(
value=predictionPatch.flatten()))
```

```
}
)
)
# Write the example. writer.write(example.SerializeToString()) patches += 1
writer.close()
# Start the upload.
out_image_asset = user_folder + '/' + out_image_base
!earthengine upload image -- asset_id={out_image_asset} {out_image_file} {jsonFile}
```

Під час експорту в TFRecord Earth Engine створює додатковий файл TFRecord під назвою «mixer». Це простий файл JSON, який визначає просторове розташування (тобто геоприв'язку) фрагмента зображення. Цей файл потрібен для завантаження прогнозів, зроблених на зображенні[9].

Все, що залишилося, це вказати області виводу, де мають бути зроблені передбачення, імена вихідних файлів, місце їх розміщення та форму виведення. З точки зору форми, модель навчена на блоках 256 x 256, але теоретично може працювати з будь-яким досить великим зображенням однакового розміру. Через артефакти на межах фрагментів зображення моделі слід надати трохи більший фрагмент для прогнозування, а потім обрізати середній (центральний) фрагмент 256 x 256. Це контролюється буферами ядра, половина з яких знаходиться за межами буферів ядра. Наприклад, якщо вказати ядро 128 x 128, до кожної сторони фрагмента зображення буде додано 64 пікселі, гарантуючи, що вихідні пікселі будуть взяті з вхідних даних, повністю покритих ядром.

Отже, дані експортуються, модель робить прогнози та записує їх у файл. Зображення, імпортовані в Earth Engine, дозволяють відображати

отримані ресурси Earth Engine. Наприклад, тут демонструються прогнози для непроникних областей над Пекіном, Китай.

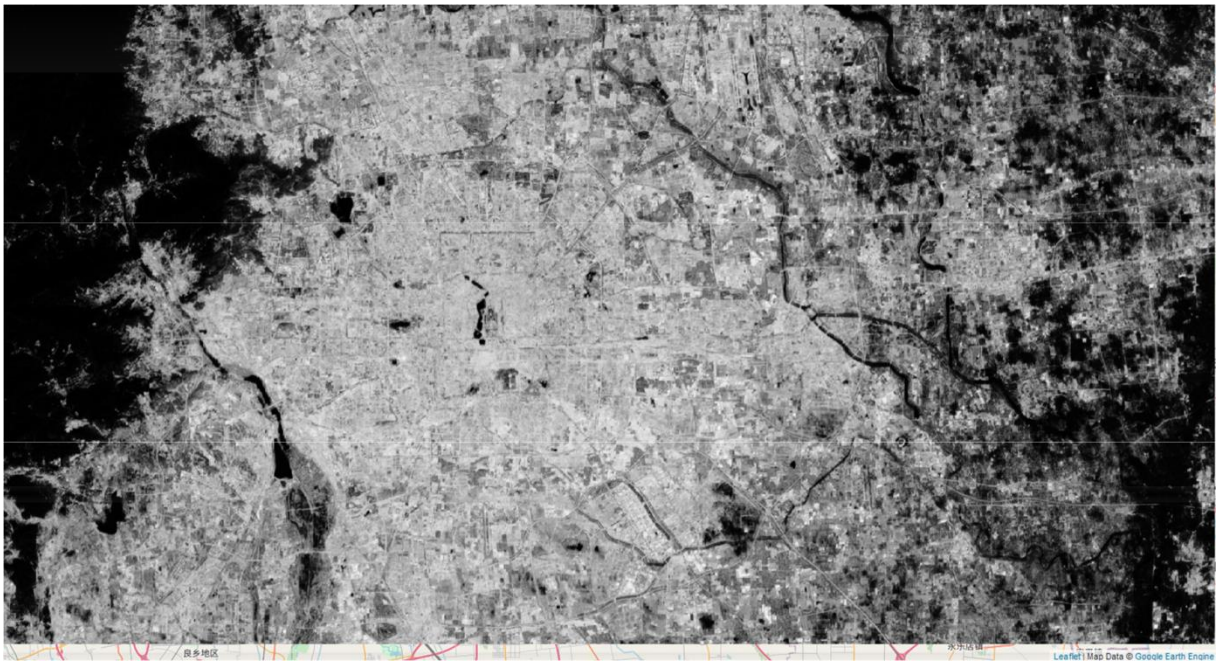


Рисунок 3.5 - Прогнозування непрозорих хмарних областей над Пекіном

Знімок Landsat 8, зроблений над Пекіном, майже повністю вважається абсолютно непроникною територією.

Розробка програми розпізнавання образів – завдання не з легких. Крім того, деталі створеної мною програми вимагали використання зовнішніх, важкодоступних даних, таких як набори даних Google Earth Engine, які вимагали реєстрації, перевірені вручну співробітниками Google перед наданням доступу до даних.

TensorFlow, особливо його Keras API, виявився чудовим інструментом для побудови та навчання нейронних мереж, які є основою програм розпізнавання образів. Наявність фреймворка Python полегшує для мене процес розробки програм, оскільки я маю певний досвід роботи з Python порівняно з іншими мовами програмування з доступним програмуванням

нейронних мереж, такими як Haskell, D або Lisp. Це разом із моїми зусиллями зробити код зрозумілим і гнучким дає змогу використовувати та вдосконалювати мою програму розпізнавання образів; однак, мають бути виконані деякі передумови, як-от доступ до Google Earth Engine і Google Cloud Storage, інакше програма буде не можна використовувати.

Створена програма розпізнавання образів справляється із завданням передбачення непроникних ділянок, про що детально йтиметься в наступному розділі.

4 РОЗГОРНУТЕ ТЕСТУВАННЯ СИСТЕМИ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ АЕРОФОТОЗЙОМКИ

4.1 Алгоритм фнкціонування програми

Сценарій виконання цієї програми відрізняється від будь-якої програми, яку ми розробляємо в нашому курсі програмування. Код не є виконуваним, але виконується поетапно (або блок за блоком) у Jupyter Notebook або будь-якій іншій похідній програмі, як-от Amazon SageMaker Notebooks, Google Collaboratory або Microsoft Azure Notebook. У моєму випадку я використовую Jupyter Notebook для локального виконання коду на своєму комп'ютері.

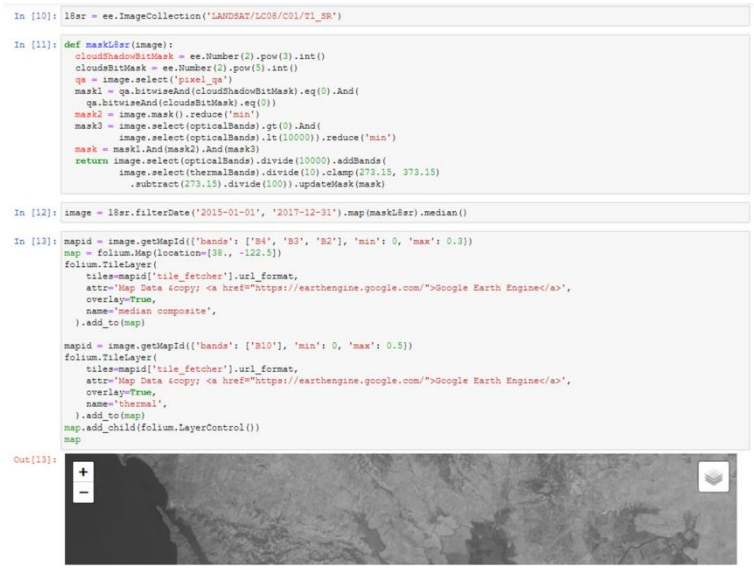


Рисунок 4.2 - Локальне виконання Jupyter Notebook

Початкові рядки коду присвячені аутентифікації в Google Cloud Storage та Earth Engine:

```
# Cloud authentication.
```

```
from google.colab import auth auth.authenticate_user()
# Import, authenticate and initialize the Earth Engine library. import ee
ee.Authenticate() ee.Initialize()
```

Для виконання цих кроків потрібно перейти за посиланням, щоб отримати код авторизації, як показано нижче. Після цих операцій Google зможе читати та записувати дані в сегменти хмарного сховища та сховище ресурсів Earth Engine.

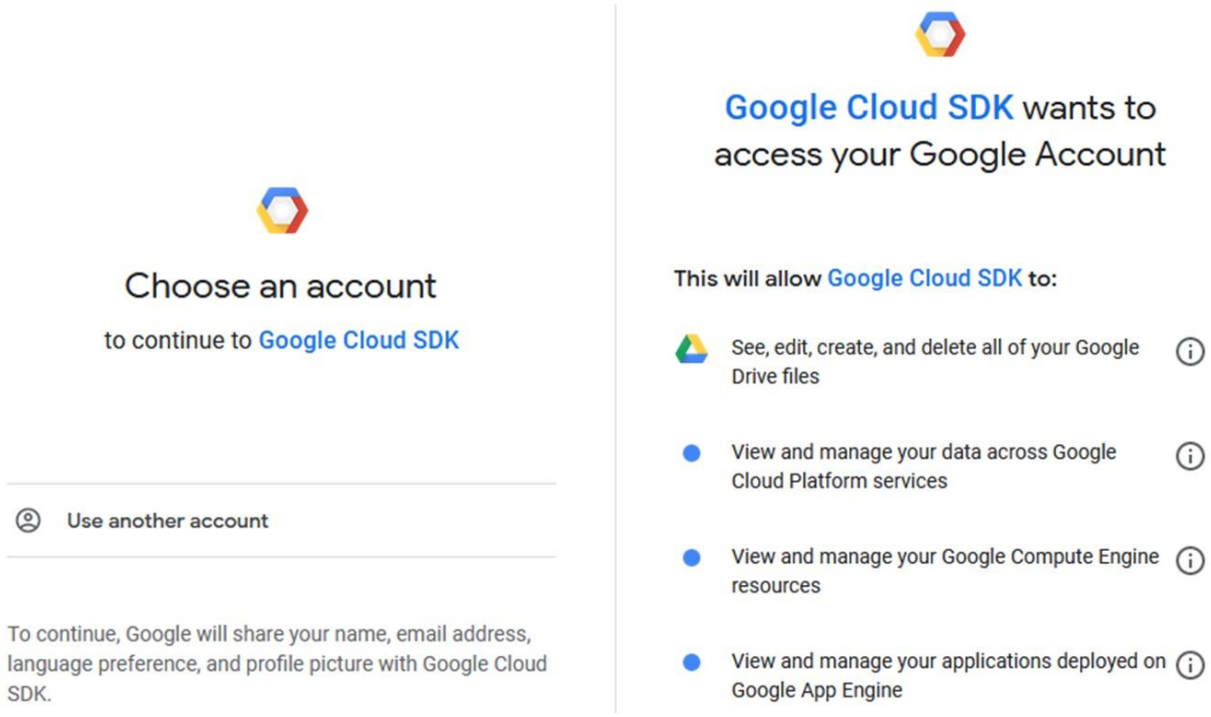


Рисунок 4.2 - Доступ до Google Cloud SDK

Після цього екрана Google поверне код підтвердження, який ви повинні вставити у відповідне поле у своєму блокноті Jupyter. Знову ж таки, потрібен доступ до Earth Engine:

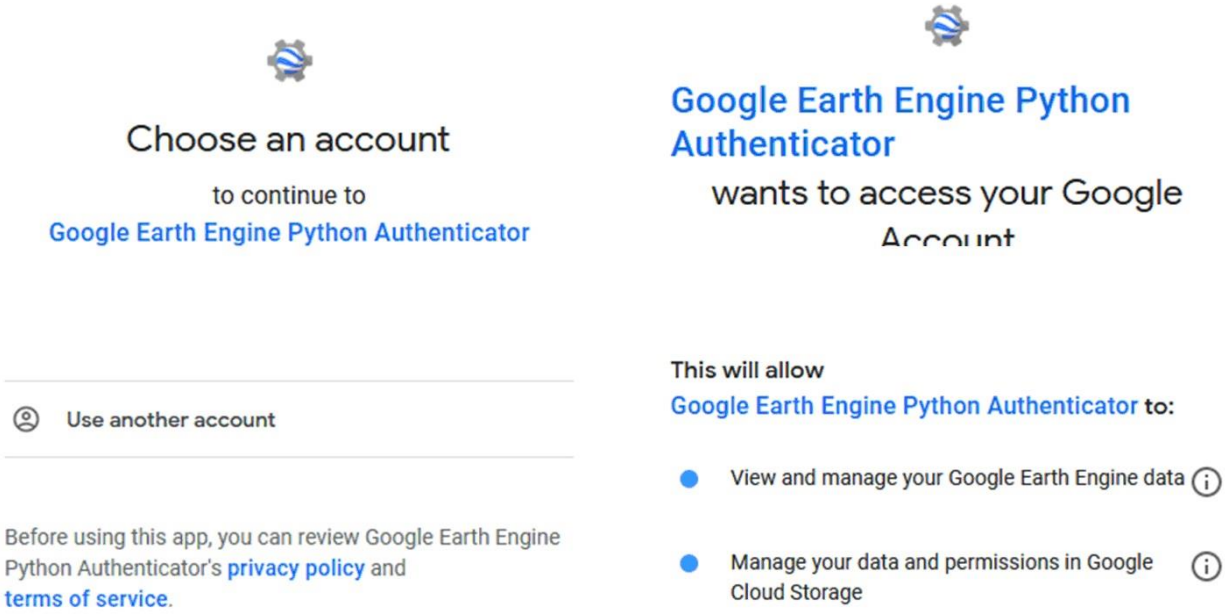


Рисунок 4.3 - Доступ до Google Earth Engine

Оскільки ці налаштування жорстко закодовані, програма розпізнавання образів наразі використовує моє власне відро Cloud Storage і сховище ресурсів Earth Engine.

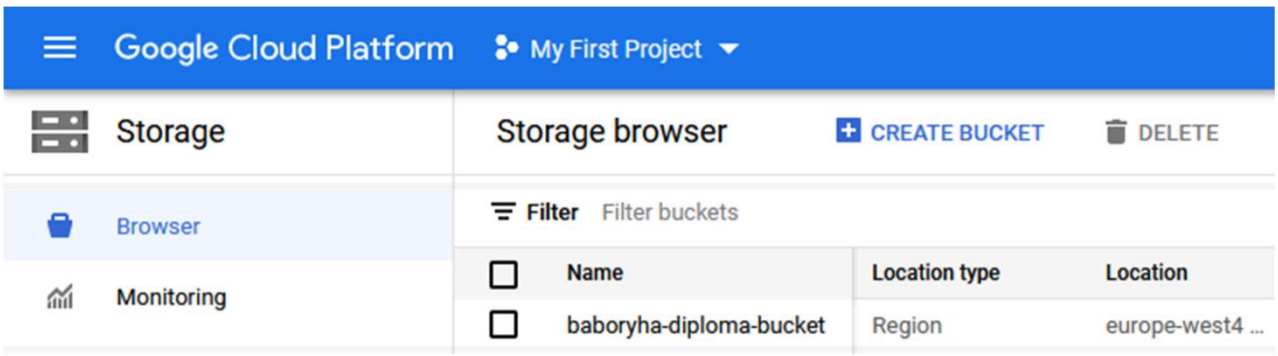


Рисунок 4.3 - Сховище Google Cloud Storage

Якщо ця програма працює з використанням можливостей зберігання сторонніх розробників, потрібно оновити посилання на сховища Cloud Storage і Earth Engine.

Очевидно, що будь-яка діяльність із зображеннями вимагає активного та стабільного підключення до Інтернету, тому цю програму неможливо запустити в автономному режимі.

Функції doExport і doPrediction, визначені в розділі 3.3, виконують завдання експорту до репозиторію Google. Ці завдання надсилаються до черги обробки на серверах Google. Код Google Earth Engine використовується для перевірки статусу завдання.

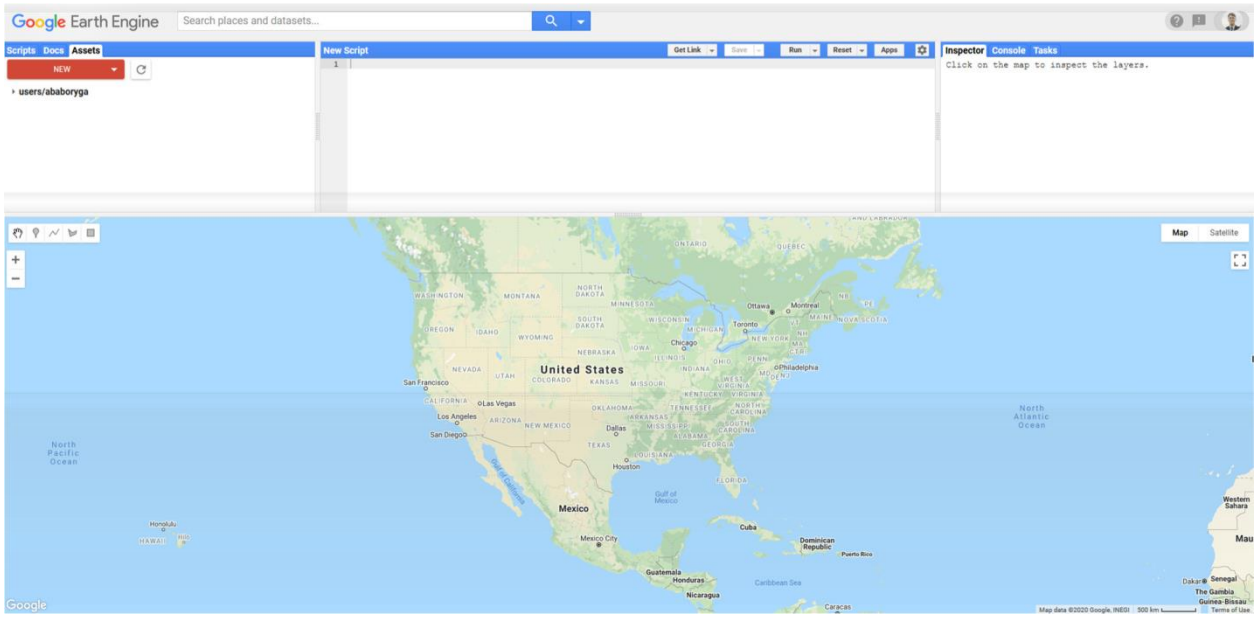


Рисунок 4.4 - Головне вікно Earth Engine Code

Хід і стан завдань, відправлених на обробку, можна будь-коли переглянути на вкладці «Tasks» у верхньому правому куті програми:

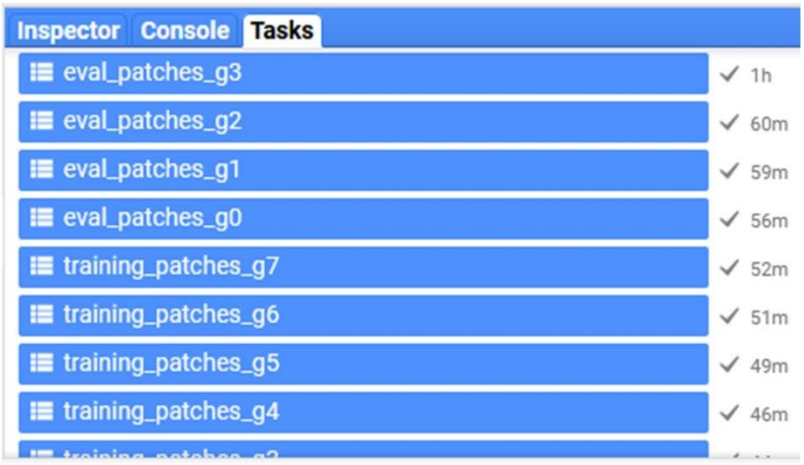


Рисунок 4.5 - Виконані завдання у Earth Engine Code

Завдання поділяються на дві групи: читання та письмо. Цю програму можна запустити з доступом лише для читання до деяких сегментів Cloud Storage, що містять дані, але в цьому випадку неможливо буде експортувати зображення нових регіонів для додаткового навчання. Програма відображатиме лише доступні результати.

Модель можна навчати при кожному запуску програми, що може зайняти значний час, тобто десятки годин у разі недостатньої апаратної потужності. Крім того, модель можна зберегти та завантажити під час виконання, як показано у фрагменті коду нижче:

```
# Load a trained model.
MODEL_DIR = 'insert-your-model-location' m = tf.keras.models.load_model(MODEL_DIR)
```

Короткий зміст нашої моделі, навченої за 10 епох, можна описати наступним

чином:

• Загальні параметри: 31 128 225

• Параметри, що піддаються навчанню: 31 112 225

• Параметри, що не піддаються навчанню: 16 000

Вона містить 26 шарів активації та нормалізації, 22 згорткових шари та 4 шари стиснення та збільшення роздільної здатності.

Повний код програми, яку потрібно виконати, наведено в Додатку Г.

Крім свого безпосереднього призначення, використання програми для передбачення непроникність міста на основі повітряної фотографії місцевості програма може бути використана вчителями та учнями для подальшого вдосконалення, як наочний матеріал для предметів, пов'язаних з нейронними мережами.

4.2 Приклади застосування програмного забезпечення

Пекін, місце, вибране для демонстраційної програми, представлено прямокутником або багатокутником, чотири вершини якого визначені географічними координатами, широтою та довготою.:

```
bj_region = ee.Geometry.Polygon( [[[115.9662455210937, 40.121362012835235],
[115.9662455210937, 39.64293313749715],
[117.01818643906245, 39.64293313749715],
[117.01818643906245, 40.121362012835235]]], None, False)
```

Цей багатокутник використовується у функціях doExport і doPrediction. Якщо користувач хоче запустити програму для іншого місця, вершини цього багатокутника потрібно замінити координатами, необхідними для іншого регіону, щоб ініціювати експорт даних і прогнози на навченій моделі.

Широту та довготу певної точки можна визначити у вікні механізму коду, клацнувши точку на інтерактивній карті. Для прикладу виберемо точку на Майдані Незалежності. Результат нижче, але сам момент ніяк не виділений.

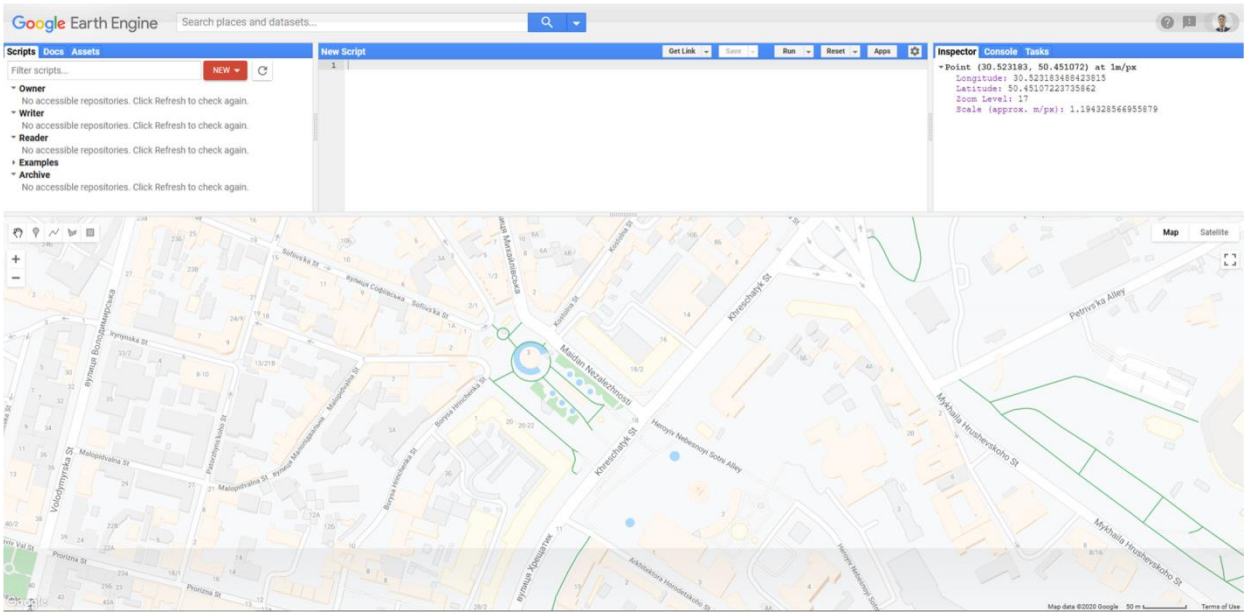


Рисунок 4.6 - Вибір точки у Earth Engine Code

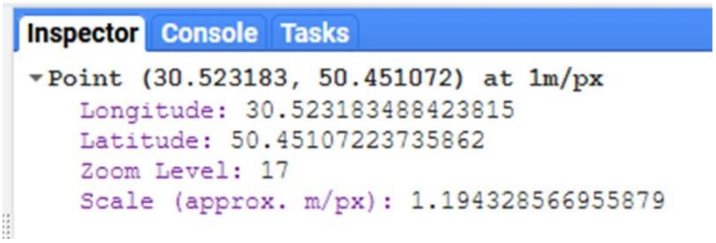


Рисунок 4.7 - Вікно Inspector

Знаючи це, запустимо завдання експорту та прогнозування на прямокутнику, який повністю охоплює місто Вінниця.

bj_region1 = ee.Geometry.Polygon([[[30.218764072867696, 50.595819362344685],

[30.218764072867696, 50.218595524781925],

[30.864210850211446, 50.218595524781925],

[30.864210850211446, 50.595819362344685]]], None, False)

Результати передбачення моделі можна побачити нижче.

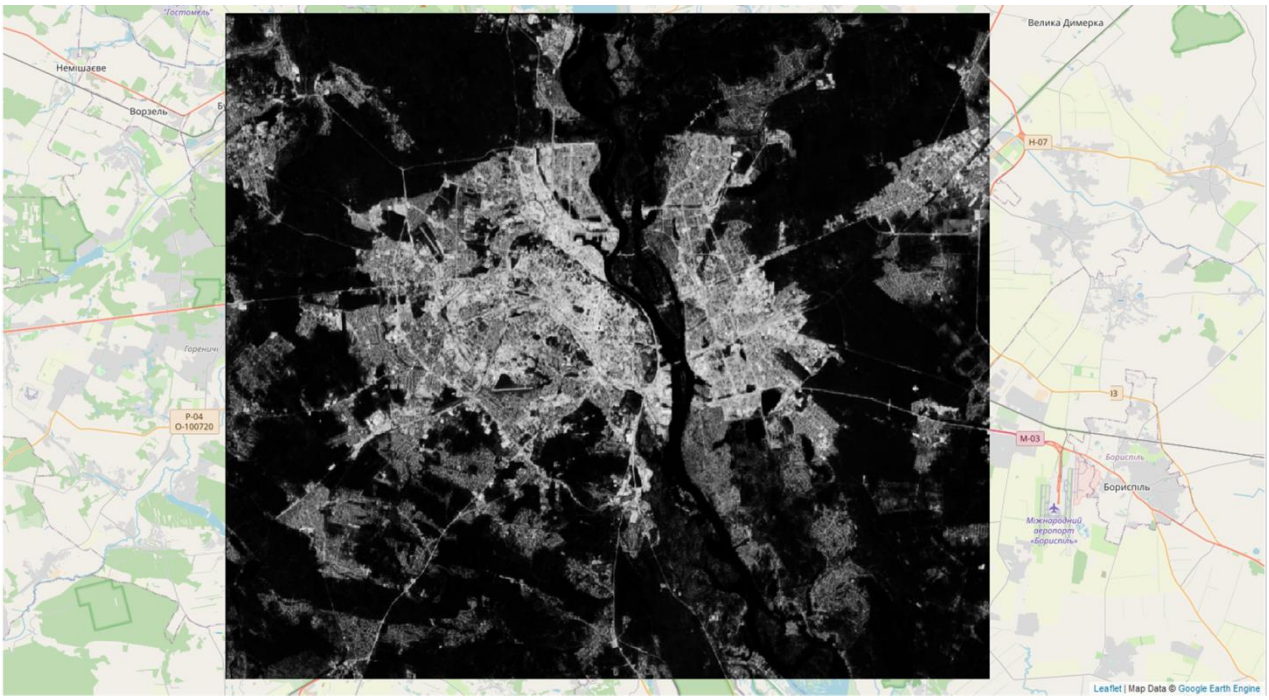


Рисунок 4.8 - Передбачення фотонепроникної території над Києвом

Як ми бачили, на неприступну територію Києва, порівняно з Пекіном, сильний вплив зелених насаджень. Однак результат програми не є досконалим, і модель може бути вдосконалена, в основному шляхом збільшення кількості навчених епох, скажімо, з 40 до 50. Однак це значно збільшить час, необхідний для навчання моделі. Тим не менш, це може стати суттєвим покращенням цієї програми розпізнавання образів.

У четвертому розділі розглядається алгоритм розробленої програми розпізнавання образів аерофотозйомки та приклади її застосування. Сценарій

виконання цієї програми не схожий на будь-яку програму, яку ми розробляємо в нашому курсі програмування - команди потрібно виконувати одну за одною у вікні Jupyter Notebook або іншому середовищі, вибраному для розміщення. Також показано приклад застосування процедури, зокрема прогноз непроникної території Києва. Для цього програмі потрібен прямокутник, вершини якого будуть географічними координатами краю області, що обробляється.

ВИСНОВОК

Створення програми розпізнавання образів на основі результатів аерофотозйомок є складним процесом, який часто залучає велику групу розробників експертів із різних галузей і з різним досвідом. Я проаналізував існуючі реалізації подібних програм і виявив у них деякі недоліки, зокрема низьку якість розпізнавання, великий використаний простір на диску та повільне навчання мережі для фактичного розпізнавання. Тож моє завдання полягало в тому, щоб розробити програму, яка б не мала цих недоліків або мінімізувала б їх продуктивність. При цьому використовував усі знання та навички, набуті під час навчання у ВНТУ..

Розробка програми розпізнавання образів складається з багатьох етапів. Почав з теоретичної підготовки. На цьому етапі я глибоко зрозумів концепцію розпізнавання образів і почав шукати моделі нейронних мереж для реалізації (які виявились архітектурами U-Net) та програмні засоби для їх реалізації. За результатами дослідження було обрано мову програмування Python з використанням додаткових бібліотек TensorFlow, Folium. Каталог Google Earth Engine був обраний як каталог даних для навчання мережі та подальшої ідентифікації.

Після цього мені потрібно створити специфікацію – вимоги до моєї програми (що вона має робити, як має бути реалізована тощо). Згодом мені довелося розробити структуру проекту, щоб усі специфікації могли бути успішно реалізовані. На цьому етапі найважливішими є структура програмного коду та специфікація формату даних. Тут я використовую принципи, отримані під час навчання з дисциплін «Комп’ютерні технології та програмування» та «Математичне моделювання».

Фаза прототипу була для мене досить важкою. На цьому етапі важливо не залишати помилок чи будь-яких інших проблем, перш ніж переходити до

наступного етапу, оскільки їх буде складніше виправити. Коли всі підсистеми та компоненти побудовані та протестовані, відбувається перехід до кінцевої фази розробки продукту.

В результаті я задоволений виконаною роботою. Мені вдалося розробити програму, яка відносно швидко навчає нейронну мережу розпізнаванню, не використовує великі обсяги даних у локальному сховищі та показує високу якість розпізнавання. Це досягається за рахунок використання хмарних сервісів, які повністю беруть на себе завдання зберігання та обробки інформації. Однак доступ до Інтернету є ключовою вимогою для запуску програм.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методи розпізнавання образів: Навч. посіб. для студ. / В. М. Заяць, Р. М. Камінський; Нац. ун-т "Львів. політехніка". – Л., 2004. – 173 с. – Бібліогр.: 21 назв.
2. Fukunaga, Keinosuke (2010). Introduction to Statistical Pattern Recognition (2nd ed.). Boston: Academic Press.
3. Дэвид А. Форсайт, Джин Понс. [Computer Vision: A Modern Approach Компьютерное зрение. Современный подход]. – М. : «Вильямс», 2012. – 928 с.
4. Джордж Стокман, Линда Шапиро. [Computer Vision Компьютерное зрение]. –

М. : Бином. Лаборатория знаний, 2016. – 752 с.
5. В. Н. Вапник, А. Я. Червоненкис. Теория распознавания образов М.: Наука, 2008. – 416 с.
6. Поспелов Д.А. Искусственный интеллект. Справочник. Книга 2. Модели и методы // М.: Радио и связь, 2007. – 304 с.
7. Файн В. С. Опознавание изображений, М. 2008.
8. Bishop, Christopher M. (2012). Pattern Recognition and Machine Learning. Springer.
9. Schuermann, Juergen (2016). Pattern Classification: A Unified View of Statistical and Neural Approaches. New York: Wiley.
10. Duda, Richard O.; Hart, Peter E.; Stork, David G. (2020). Pattern Classification (2nd ed.). Wiley-Interscience.
11. Kulikowski, Casimir A.; Weiss, Sholom M. (2011). Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems. Machine Learning. San Francisco: Morgan Kaufmann Publishers.

12. What is Python? Executive Summary [Online]. Available: https://www.python.org/doc/essays/blurb/

13. The Cain Gang Ltd. Python Metaclasses: Who? Why? When? [Online] Available: https://web.archive.org/web/20090530030205/http://www.python.org/community/ p ycon/dc2004/papers/24/metaclasses-pycon.pdf

14. "Extending and Embedding the Python Interpreter: Reference Counts" [Online] Available: https://docs.python.org/3/extending/extending.html#reference-counts

15. Peters, Tim. PEP 20 – The Zen of Python. [Online] Available: https://www.python.org/dev/peps/pep-0020/

16. Guttag, John V. Introduction to Computation and Programming Using Python: With Application to Understanding Data. MIT Press. ISBN 978-0-262-52962-4.

17. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, 2012.

18. K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014.

19. P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In ICLR, 2014.

20. R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In Computer Vision and Pattern Recognition, 2014.

21. N. Zhang, J. Donahue, R. Girshick, and T. Darrell. Part-based r-cnns for fine- grained category detection. In Computer Vision–ECCV 2014, pages 834–849. Springer, 2014.

22. J. Long, E. Ehelhamer, and T. Darrell. Fully Convolutional Networks for Semantic Segmentation. [Online] Available: https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Long_Fully_Convolutional_Networks_2015_CVPR_paper.pdf

23. C. M. Bishop. Pattern recognition and machine learning. Springer-Verlag New York, 2006

24. D. C. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmid-huber. Deep neural networks segment neuronal membranesin electron microscopy images. In NIPS, pages 2852–2860, 2012.

25. D. Eigen, C. Puhrsch, and R. Fergus. Depth map predictionfrom a single image using a multi-scale deep network. arXiv preprint arXiv:1406.2283, 2014.

26. C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. Pattern Analysis andMachine Intelligence, IEEE Transactions on, 2013.

27. J. Long, N. Zhang, and T. Darrell. Do convnets learn correspondence? In NIPS, 2014.

28. J. Tighe and S. Lazebnik. Finding things: Image parsing withregions and per- exemplar detectors. In CVPR, 2013.

29. He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian (2015-12-10). "Deep Residual Learning for Image Recognition". arXiv:1512.03385

30. Huang, Gao; Liu, Zhuang; Weinberger, Kilian Q.; van der Maaten, Laurens (2016- 08-24). "Densely Connected Convolutional Networks". arXiv:1608.06993

31. Iberdrola, S.A., WHAT IS TECHNOLOGICAL WASTE? [Online]. Available: https://www.iberdrola.com/environment/what-is-e-waste

ДОДАТКИ

Додаток А
(обов'язковий)
Протокол перевірки МКР на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Інтелектуальна автоматизована система розпізнавання об'єктів на зображеннях земної поверхні»

Тип роботи: Магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ КСУ, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 93,4% Схожість 6,6%

Аналіз звіту подібності (відмітити потрібне)

☐ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на розгляд експертної комісії кафедри.
☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Галущак А.В.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unichek щодо роботи.

Автор роботи _____ Сідак Д.А.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Севастьянов В.М.
(підпис) (прізвище, ініціали)

Додаток Б
(обов'язковий)
ВНТУ

ЗАТВЕРДЖЕНО
Зав. кафедри КСУ ВНТУ

_____ В'ячеслав КОВТУН

"30" жовтня 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

Інтелектуальна автоматизована система розпізнавання об'єктів на зображеннях земної поверхні

(Тема)

Студент групи ЗАКІТ-21м

Дмитрій Сідак

Керівник к.т.н., доц. каф. АІІТ

Володимир Севастьянов

Вінниця 2022

1. Назва та галузь застосування

1.1. Інтелектуальна автоматизована система розпізнавання об’єктів на зображеннях земної поверхні.

1.2. Галузь застосування – розпізнавання об’єктів земної поверхні на зображеннях.

2. Підстава для проведення розробки.

Тема магістерської кваліфікаційної роботи затверджена наказом по ВНТУ від “14” вересня 2022 року №203

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є розробка комп’ютерної програми розпізнавання образів, яка може бути використана для навчальної та дослідницької роботи.

4. Джерела розробки.

Магістерська кваліфікаційна робота виконується вперше. В ході проведення розробки повинні використовуватись такі документи:

1. Методи розпізнавання образів: Навч. посіб. для студ. / В. М. Заяць, Р. М. Камінський; Нац. ун-т "Львів. політехніка". – Л., 2014. – 173 с. – Бібліогр.: 21 назв.

2. Fukunaga, Keinosuke (2010). Introduction to Statistical Pattern Recognition (2nd ed.). Boston: Academic Press.

3. Распознавание образов [Електронний ресурс]: Хабрахабр / Режим доступу : https://habrahabr.ru/post/208090/ - Назва з екрану.

5. Вимоги до розробки.

5.1. Перелік головних функцій:

- розпізнавання образів на основі результатів аерофотозйомок

5.2. Основні технічні вимоги до розробки.

- WINDOWS 10;
- Jupyter Notebook;

- Python 3.8+.

6. Стадії та етапи розробки.

6.1 Пояснювальна записка:

1. Аналіз методів, принципів, та постановка задач дослідження «03»_09_ 2022 р.
2. Вибір програмних засобів для розробки системи розпізнавання образів «17»_10_ 2022 р.
3. Дослідження існуючих архітектур нейронних мереж «22»_10_ 2022 р.
4. Розробка системи розпізнавання зображень аерофотозйомки «02»_11_ 2022 р.

6.2 Графічні матеріали:

1. Тестування програмного забезпечення «28»_11_ 2022 р.

7. Порядок контролю і приймання.

7.1. Хід виконання роботи контролюється керівником роботи. Рубіжний контроль провести до «27»_11_ 2022 р.

7.2. Атестація МКР здійснюється на попередньому захисті. Попередній захист магістерської кваліфікаційної роботи провести до «16»_16_ 2022 р.

7.3. Підсумкове рішення щодо оцінки якості виконання роботи приймається на засіданні ЕК. Захист магістерської кваліфікаційної роботи провести до «22»_12_ 2022р.

Додаток В
(обов'язковий)
Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

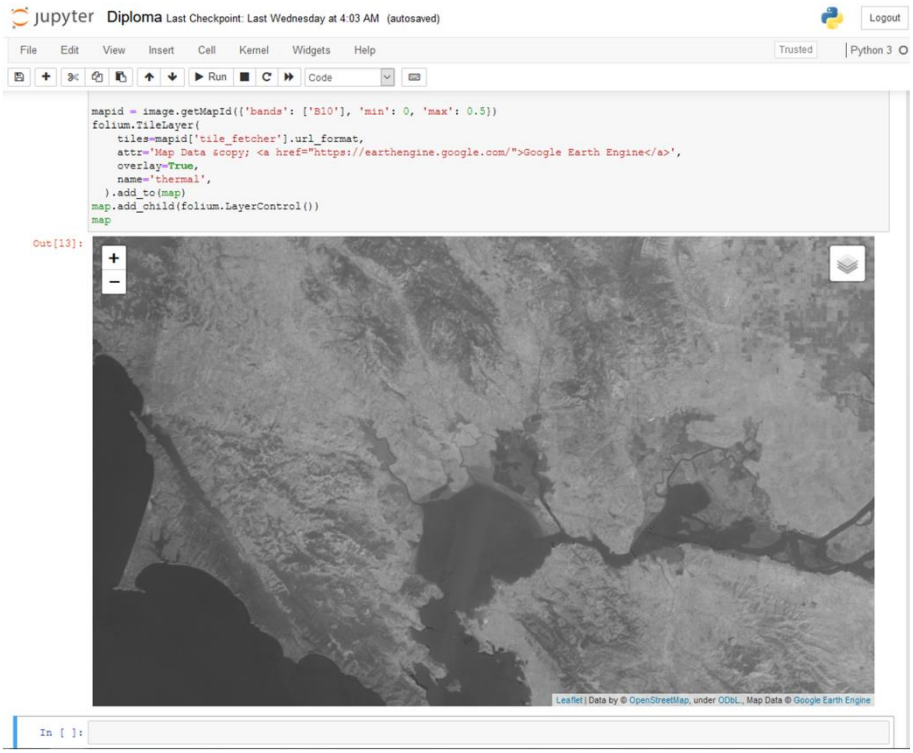
Інтелектуальна автоматизована система розпізнавання об'єктів на зображеннях земної поверхні

Студент групи ЗАКІТ-21м

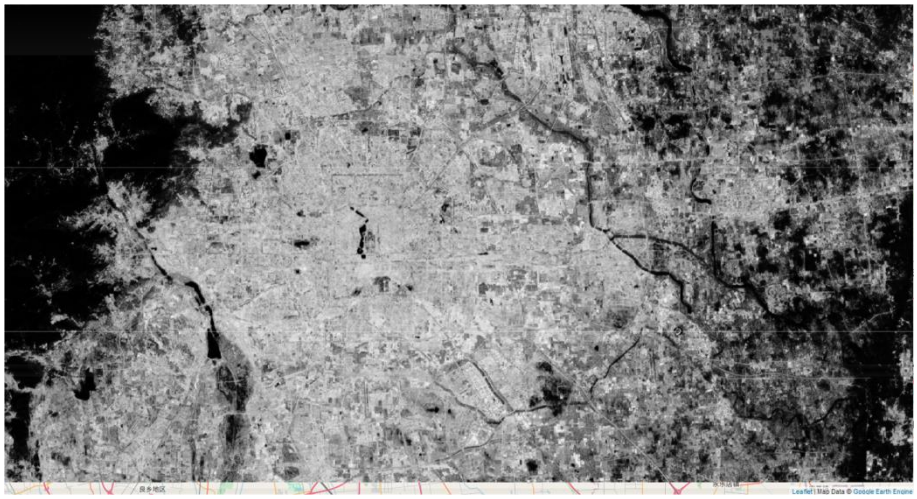
Сідак Дмитрій

Керівник к.т.н., доц. каф. АПТ

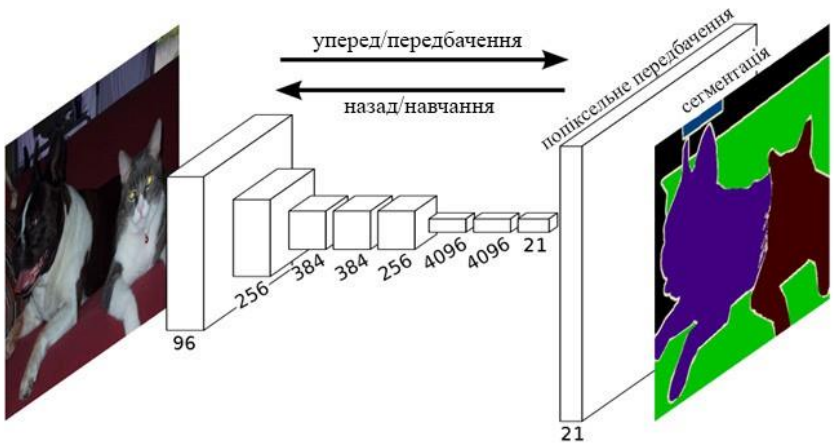
Володимир Севастьянов



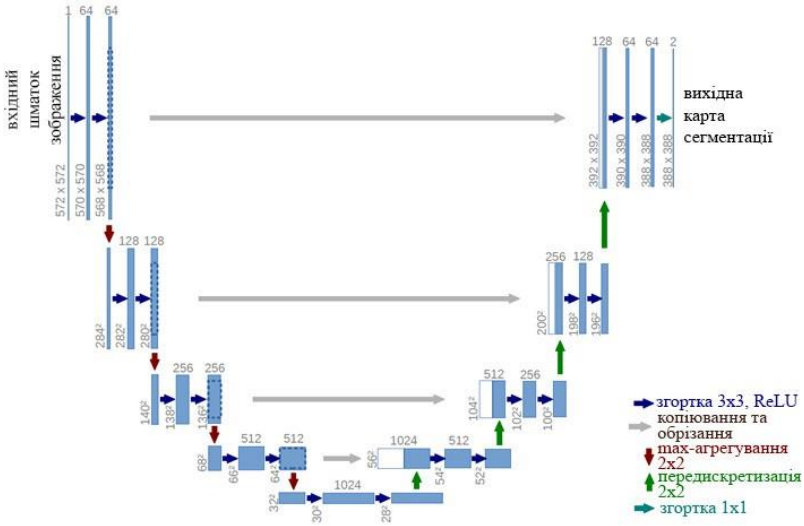
Приклад використання Jupyter Notebook



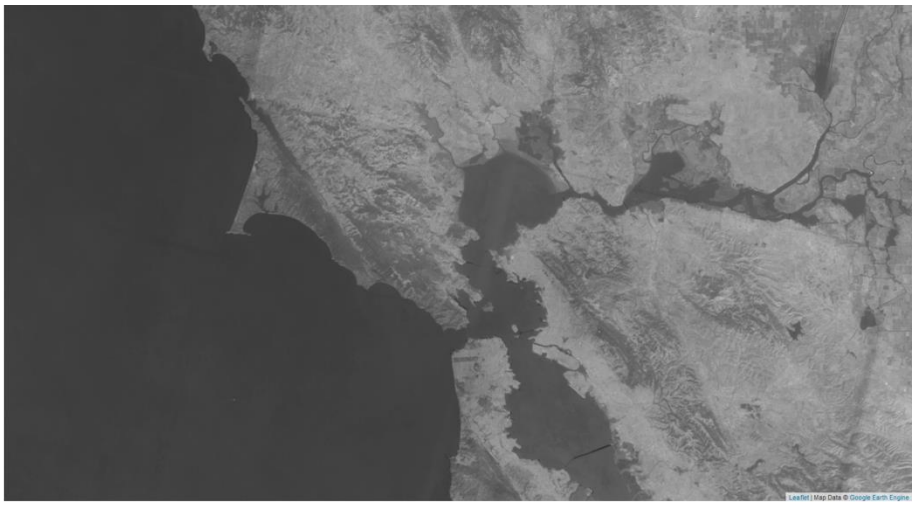
Прогнозування непрозорих хмарних областей над Пекіном



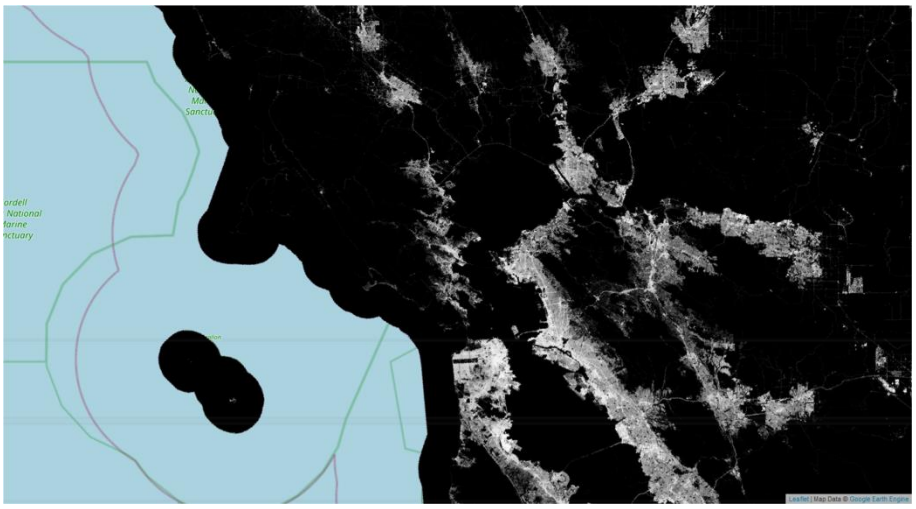
Семантична сегментація пікселів зображень за допомогою згорткової нейромережі



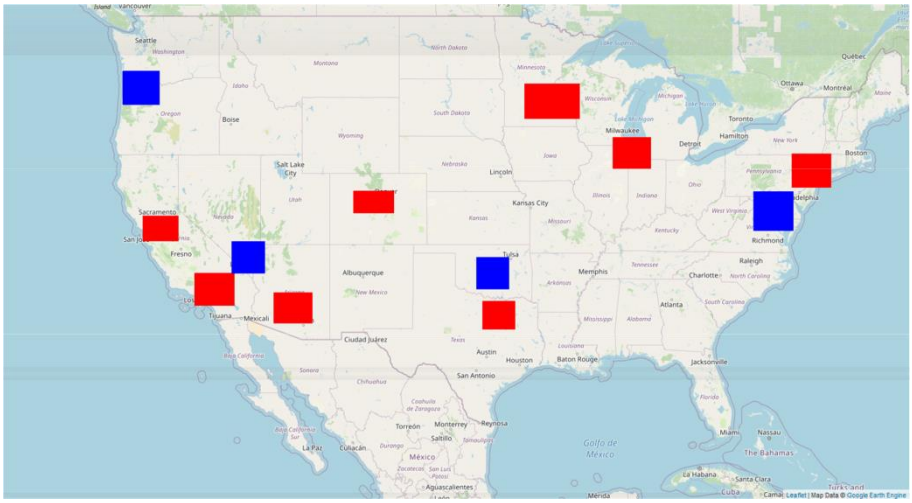
Архітектура моделі U-Net



Фрагмент візуалізації зображення від супутнику Landsat 8



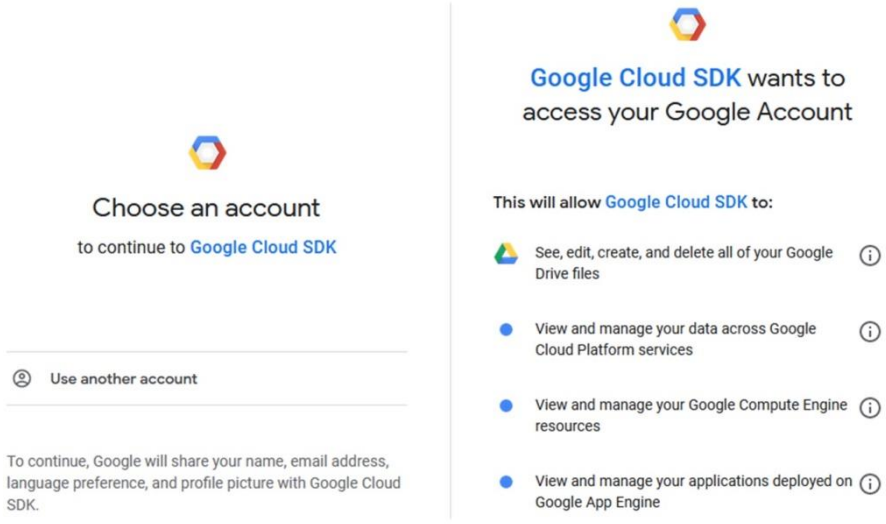
Фрагмент візуалізації зображень з NLCD



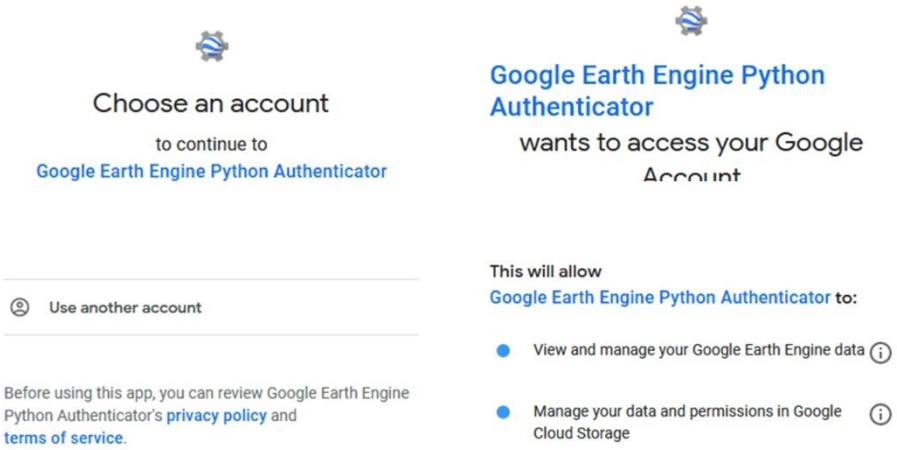
Навчальні та оцінювальні мітки для моделі

eval_patches_g0.tfrecord.gz	1.7 GB
eval_patches_g1.tfrecord.gz	2.1 GB
eval_patches_g2.tfrecord.gz	1.9 GB
eval_patches_g3.tfrecord.gz	1.9 GB
training_patches_g0.tfrecord.gz	1.7 GB
training_patches_g1.tfrecord.gz	2 GB
training_patches_g2.tfrecord.gz	2 GB
training_patches_g3.tfrecord.gz	2.1 GB
training_patches_g4.tfrecord.gz	2.1 GB
training_patches_g5.tfrecord.gz	2.1 GB
training_patches_g6.tfrecord.gz	1.5 GB
training_patches_g7.tfrecord.gz	1.6 GB

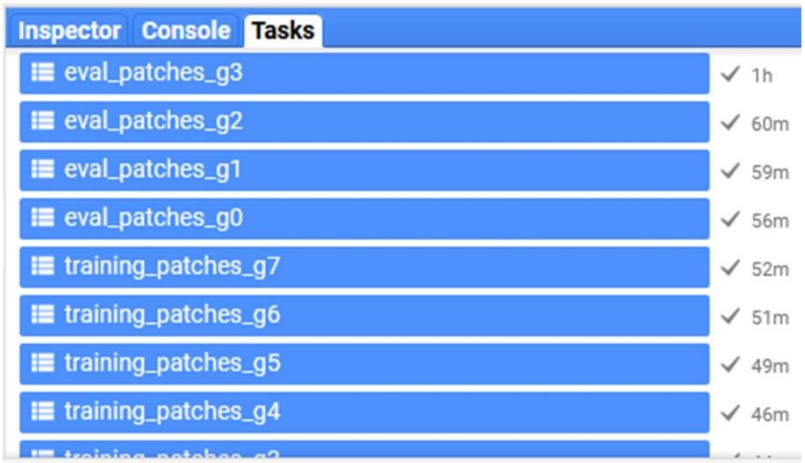
Датасет TFrecord для моделі в Google Cloud Storage



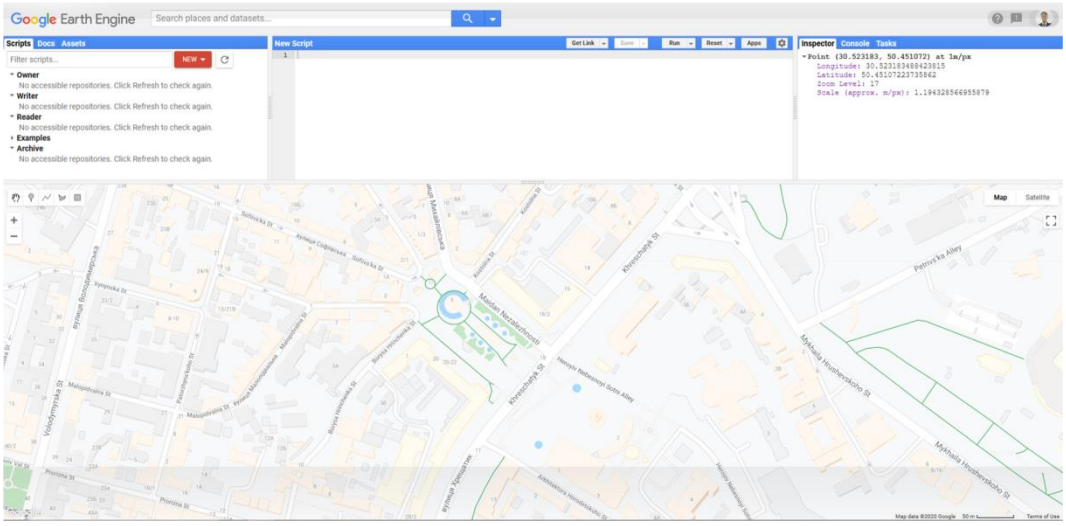
Доступ до Google Cloud SDK



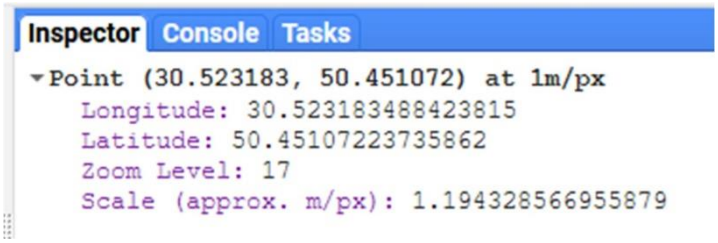
Доступ до Google Earth Engine



Виконані завдання у Earth Engine Code



Вибір точки у Earth Engine Code



Вікно Inspector

Додаток Г

Вибрані лістинги

Main.py

```
# Import, authenticate and initialize the Earth Engine library import ee
ee.Authenticate() ee.Initialize()

# Tensorflow setup import tensorflow as tf print(tf.  version  )

# Folium setup import folium
print(folium.  version  )

# CLOUD STORAGE BUCKET
BUCKET = 'baboryha-diploma-bucket'

# Specify names locations for Google Cloud Storage FOLDER = 'fcnn'
TRAINING_BASE = 'training_patches' EVAL_BASE = 'eval_patches'

# Specify inputs (Landsat bands) to the model and the response variable
opticalBands = ['B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7']
thermalBands = ['B10', 'B11']
BANDS = opticalBands + thermalBands RESPONSE = 'impervious'
FEATURES = BANDS + [RESPONSE]

# Specify the size and shape of patches expected by the model KERNEL_SIZE = 256
KERNEL_SHAPE = [KERNEL_SIZE, KERNEL_SIZE] COLUMNS = [
tf.io.FixedLenFeature(shape=KERNEL_SHAPE, dtype=tf.float32) for k in FEATURES
]
FEATURES_DICT = dict(zip(FEATURES, COLUMNS))

# Sizes of the training and evaluation datasets TRAIN_SIZE = 16000
EVAL_SIZE = 8000

# Specify model training parameters BATCH_SIZE = 16
```

```
EPOCHS = 10
BUFFER_SIZE = 2000 OPTIMIZER = 'SGD'
LOSS = 'MeanSquaredError'
METRICS = ['RootMeanSquaredError']
# Use Landsat 8 surface reflectance data
l8sr = ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')

# Cloud masking function def maskL8sr(image):
cloudShadowBitMask = ee.Number(2).pow(3).int() cloudsBitMask = ee.Number(2).pow(5).int()
qa = image.select('pixel_qa')
mask1 = qa.bitwiseAnd(cloudShadowBitMask).eq(0).And( qa.bitwiseAnd(cloudsBitMask).eq(0))
mask2 = image.mask().reduce('min')
mask3 = image.select(opticalBands).gt(0).And( image.select(opticalBands).lt(10000)).reduce('min') mask =
mask1.And(mask2).And(mask3)
return                                                    image.select(opticalBands).divide(10000).addBands(
image.select(thermalBands).divide(10).clamp(273.15,  373.15)
.subtract(273.15).divide(100)).updateMask(mask)

# The image input data is a cloud-masked median composite image = l8sr.filterDate('2015-01-01', '2017-
12- 31').map(maskL8sr).median()
# Use folium to visualize the imagery
mapid = image.getMapId({'bands': ['B4', 'B3', 'B2'], 'min': 0, 'max': 0.3})
map = folium.Map(location=[38., -122.5]) folium.TileLayer(
tiles=mapid['tile_fetcher'].url_format, attr='Map Data &copy; <a
href="https://earthengine.google.com/">Google Earth Engine</a>', overlay=True,
name='median composite',
).add_to(map)
mapid = image.getMapId({'bands': ['B10'], 'min': 0, 'max': 0.5}) folium.TileLayer(
tiles=mapid['tile_fetcher'].url_format, attr='Map Data &copy; <a
href="https://earthengine.google.com/">Google Earth Engine</a>', overlay=True,
name='thermal',
).add_to(map) map.add_child(folium.LayerControl()) map

# Prepare the response (what we want to predict)
nlcd = ee.Image('USGS/NLCD/NLCD2016').select('impervious') nlcd = nlcd.divide(100).float()
mapid = nlcd.getMapId({'min': 0, 'max': 1})
map = folium.Map(location=[38., -122.5]) folium.TileLayer(
```

```
tiles=mapid['tile_fetcher'].url_format, attr='Map Data &copy; <a
href="https://earthengine.google.com/">Google Earth Engine</a>', overlay=True,
name='nlcd impervious',
).add_to(map) map.add_child(folium.LayerControl()) map

# Stack the 2d images to create a single image from which samples can be taken
featureStack = ee.Image.cat([ image.select(BANDS), nlcd.select(RESPONSE)
]).float()
list = ee.List.repeat(1, KERNEL_SIZE) lists = ee.List.repeat(list, KERNEL_SIZE)
kernel = ee.Kernel.fixed(KERNEL_SIZE, KERNEL_SIZE, lists) arrays = featureStack.neighborhoodToArray(kernel)
# Use some pre-made geometries to sample the stack in strategic locations
trainingPolys = ee.FeatureCollection('projects/google/DemoTrainingGeometries') evalPolys = ee.FeatureCollection('projects/google/DemoEvalGeometries')
polyImage = ee.Image(0).byte().paint(trainingPolys, 1).paint(evalPolys, 2)
polyImage = polyImage.updateMask(polyImage)
mapid = polyImage.getMapId({'min': 1, 'max': 2, 'palette': ['red', 'blue']})
map = folium.Map(location=[38., -100.], zoom_start=5) folium.TileLayer(
tiles=mapid['tile_fetcher'].url_format, attr='Map Data &copy; <a
href="https://earthengine.google.com/">Google Earth Engine</a>', overlay=True,
name='training polygons',
).add_to(map) map.add_child(folium.LayerControl()) map

# Sampling
# Convert the feature collections to lists for iteration trainingPolysList = trainingPolys.toList(trainingPolys.size())

evalPolysList = evalPolys.toList(evalPolys.size())

# These numbers determined experimentally n = 200 # Number of shards in each polygon
N = 2000 # Total sample size in each polygon

# Export all the training data (in many pieces), with one task
# per geometry
for g in range(trainingPolys.size().getInfo()): geomSample = ee.FeatureCollection([])
for i in range(n): sample = arrays.sample(
region = ee.Feature(trainingPolysList.get(g)).geometry(), scale = 30,
```

```
numPixels = N / n, # Size of the shard seed = i,
tileScale = 8
)
geomSample = geomSample.merge(sample)
desc = TRAINING_BASE + '_g' + str(g)
task = ee.batch.Export.table.toCloudStorage( collection = geomSample,
description = desc, bucket = BUCKET,
fileNamePrefix = FOLDER + '/' + desc, fileFormat = 'TFRecord',
selectors = BANDS + [RESPONSE]
)
task.start()

# Export all the evaluation data
for g in range(evalPolys.size().getInfo()): geomSample = ee.FeatureCollection([])
for i in range(n): sample = arrays.sample(
region = ee.Feature(evalPolysList.get(g)).geometry(), scale = 30,
numPixels = N / n, seed = i, tileScale = 8
)
geomSample = geomSample.merge(sample)
desc = EVAL_BASE + '_g' + str(g)
task = ee.batch.Export.table.toCloudStorage( collection = geomSample,
description = desc, bucket = BUCKET,
fileNamePrefix = FOLDER + '/' + desc, fileFormat = 'TFRecord',
selectors = BANDS + [RESPONSE]
)
task.start()

# Training data
def parse_tfrecord(example_proto): """The parsing function.
Read a serialized example into the structure defined by FEATURES_DICT.
Args:
example_proto: a serialized Example. Returns:
A dictionary of tensors, keyed by feature name. """
return tf.io.parse_single_example(example_proto, FEATURES_DICT)
def to_tuple(inputs):
"""Function to convert a dictionary of tensors to a tuple of (inputs, outputs).
Turn the tensors returned by parse_tfrecord into a stack in HWC shape.
```

```
    Args:
    inputs: A dictionary of tensors, keyed by feature name. Returns:
    A tuple of (inputs, outputs). """
    inputsList = [inputs.get(key) for key in FEATURES] stacked = tf.stack(inputsList, axis=0)
    # Convert from CHW to HWC
    stacked = tf.transpose(stacked, [1, 2, 0])
    return stacked[:,:,:len(BANDS)], stacked[:,:,len(BANDS):]
    def get_dataset(pattern):
    """Function to read, parse and format to tuple a set of input tfrecord files.
    Get all the files matching the pattern, parse and convert to tuple. Args:
    pattern: A file pattern to match in a Cloud Storage bucket. Returns:
    A tf.data.Dataset """
    glob = tf.io.gfile.glob(pattern)
    dataset = tf.data.TFRecordDataset(glob, compression_type='GZIP') dataset = dataset.map(parse_tfrecord,
num_parallel_calls=5) dataset = dataset.map(to_tuple, num_parallel_calls=5)
    return dataset
    # Use helpers to read in training dataset def get_training_dataset():
    """Get the preprocessed training dataset Returns:

    A tf.data.Dataset of training data. """
    '*'
    glob = 'gs://' + BUCKET + '/' + FOLDER + '/' + TRAINING_BASE +
    dataset = get_dataset(glob)
    dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE).repeat() return dataset
    training = get_training_dataset()

    # Making sure we retrieved the dataset successfully print(iter(training.take(1)).next())
    # Repeat for evaluation dataset def get_eval_dataset():
    """Get the preprocessed evaluation dataset Returns:
    A tf.data.Dataset of evaluation data. """
    glob = 'gs://' + BUCKET + '/' + FOLDER + '/' + EVAL_BASE + '*'
    dataset = get_dataset(glob)
    dataset = dataset.batch(1).repeat() return dataset
    evaluation = get_eval_dataset()

    # Model: we use Keras implementation of the U-Net model from tensorflow.python.keras import layers
```

from tensorflow.python.keras import losses from tensorflow.python.keras import models from tensorflow.python.keras import metrics from tensorflow.python.keras import optimizers

def conv_block(input_tensor, num_filters): encoder = layers.Conv2D(num_filters, (3, 3),

padding='same')(input_tensor)

encoder = layers.BatchNormalization()(encoder) encoder = layers.Activation('relu')(encoder) encoder = layers.Conv2D(num_filters, (3, 3),

padding='same')(encoder)

encoder = layers.BatchNormalization()(encoder) encoder = layers.Activation('relu')(encoder) return encoder

def encoder_block(input_tensor, num_filters): encoder = conv_block(input_tensor, num_filters)

encoder_pool = layers.MaxPooling2D((2, 2), strides=(2, 2))(encoder)

return encoder_pool, encoder

def decoder_block(input_tensor, concat_tensor, num_filters):

decoder = layers.Conv2DTranspose(num_filters, (2, 2), strides=(2,

2), padding='same')(input_tensor)

decoder = layers.concatenate([concat_tensor, decoder], axis=-1) decoder = layers.BatchNormalization()(decoder)

decoder = layers.Activation('relu')(decoder) decoder = layers.Conv2D(num_filters, (3, 3),

padding='same')(decoder)

decoder = layers.BatchNormalization()(decoder) decoder = layers.Activation('relu')(decoder) decoder = layers.Conv2D(num_filters, (3, 3),

padding='same')(decoder)

decoder = layers.BatchNormalization()(decoder) decoder = layers.Activation('relu')(decoder) return decoder

def get_model():

inputs = layers.Input(shape=[None, None, len(BANDS)]) # 256 encoder0_pool, encoder0 = encoder_block(inputs, 32) # 128 encoder1_pool, encoder1 = encoder_block(encoder0_pool, 64) # 64 encoder2_pool, encoder2 = encoder_block(encoder1_pool, 128) # 32 encoder3_pool, encoder3 = encoder_block(encoder2_pool, 256) # 16 encoder4_pool, encoder4 = encoder_block(encoder3_pool, 512) # 8 center = conv_block(encoder4_pool, 1024) # center

decoder4 = decoder_block(center, encoder4, 512) # 16 decoder3 = decoder_block(decoder4, encoder3, 256) # 32 decoder2 = decoder_block(decoder3, encoder2, 128) # 64 decoder1 = decoder_block(decoder2, encoder1, 64) # 128 decoder0 = decoder_block(decoder1, encoder0, 32) # 256 outputs = layers.Conv2D(1, (1, 1),

activation='sigmoid')(decoder0)

model = models.Model(inputs=[inputs], outputs=[outputs]) model.compile(

optimizer=optimizers.get(OPTIMIZER), loss=losses.get(LOSS),

metrics=[metrics.get(metric) for metric in METRICS]) return model

```
# Train the model
m = get_model() m.fit(
x=training, epochs=EPOCHS,
steps_per_epoch=int(TRAIN_SIZE / BATCH_SIZE), validation_data=evaluation, validation_steps=EVAL_SIZE)


# Prediction pipeline
# 1. Export imagery on which to do predictions from Earth Engine in TFRecord format
def doExport(out_image_base, kernel_buffer, region): """Run the image export task. Block until complete. """
task = ee.batch.Export.image.toCloudStorage( image = image.select(BANDS),
description = out_image_base, bucket = BUCKET,
fileNamePrefix = FOLDER + '/' + out_image_base, region = region.getInfo()['coordinates'], scale = 30,
fileFormat = 'TFRecord', maxPixels = 1e10, formatOptions = {
'patchDimensions': KERNEL_SHAPE, 'kernelSize': kernel_buffer, 'compressed': True, 'maxFileSize': 104857600
}
)
task.start()

# Block until the task completes.
print('Running image export to Cloud Storage...') import time
while task.active(): time.sleep(30)

# Error condition
if task.status()['state'] != 'COMPLETED': print('Error with image export.')
else:
print('Image export completed.')

# 2. Use the trained model to make the predictions
def doPrediction(out_image_base, user_folder, kernel_buffer, region): """Perform inference on exported imagery, upload to Earth Engine. """
print('Looking for TFRecord files...')

# Get a list of all the files in the output bucket. filesList = !gsutil ls 'gs://'{BUCKET}'/'{FOLDER}
```

```
# Get only the files generated by the image export. exportFilesList = [s for s in filesList if out_image_base
in s]
# Get the list of image files and the JSON mixer file. imageFilesList = []
jsonFile = None
for f in exportFilesList:
if f.endswith('.tfrecord.gz'): imageFilesList.append(f)
elif f.endswith('.json'): jsonFile = f

# Make sure the files are in the right order. imageFilesList.sort()
from pprint import pprint pprint(imageFilesList) print(jsonFile)

import json
# Load the contents of the mixer file to a JSON object. jsonText = !gsutil cat {jsonFile}
# Get a single string w/ newlines from the IPython.utils.text.SList mixer = json.loads(jsonText.nlstr)
pprint(mixer)
patches = mixer['totalPatches']

# Get set up for prediction. x_buffer = int(kernel_buffer[0] / 2) y_buffer = int(kernel_buffer[1] / 2)
buffered_shape = [
KERNEL_SHAPE[0] + kernel_buffer[0], KERNEL_SHAPE[1] + kernel_buffer[1]]
imageColumns = [
tf.io.FixedLenFeature(shape=buffered_shape,  dtype=tf.float32) for k in BANDS
]
imageFeaturesDict = dict(zip(BANDS, imageColumns)) def parse_image(example_proto):
return tf.io.parse_single_example(example_proto, imageFeaturesDict)
def toTupleImage(inputs):
inputsList = [inputs.get(key) for key in BANDS] stacked = tf.stack(inputsList, axis=0)
stacked = tf.transpose(stacked, [1, 2, 0]) return stacked

# Create a dataset from the TFRecord file(s) in Cloud Storage. imageDataset =
tf.data.TFRecordDataset(imageFilesList,
compression_type='GZIP')
imageDataset = imageDataset.map(parse_image, num_parallel_calls=5) imageDataset =
imageDataset.map(toTupleImage).batch(1)

# Perform inference. print('Running predictions...')
predictions = m.predict(imageDataset, steps=patches, verbose=1)
```

```
# print(predictions[0])
print('Writing predictions...')
out_image_file = 'gs://' + BUCKET + '/' + FOLDER + '/' + out_image_base + '.TFRecord'
writer = tf.io.TFRecordWriter(out_image_file) patches = 0
for predictionPatch in predictions: print('Writing patch ' + str(patches) + '...') predictionPatch = predictionPatch[
x_buffer:x_buffer+KERNEL_SIZE,  y_buffer:y_buffer+KERNEL_SIZE]

# Create an example. example = tf.train.Example( features=tf.train.Features( feature={
'impervious': tf.train.Feature( float_list=tf.train.FloatList( value=predictionPatch.flatten()))
}
)
)

# Write the example. writer.write(example.SerializeToString()) patches += 1
writer.close()

# Start the upload.
out_image_asset = user_folder + '/' + out_image_base
!earthengine upload image --asset_id={out_image_asset}
{out_image_file} {jsonFile}

# Output assets folder: user_folder = 'users/ababoryga'
# Base file name to use for TFRecord files and assets. bj_image_base = 'FCNN_beijing_384_'
# Half this will extend on the sides of each patch. bj_kernel_buffer = [128, 128]
# Defining Beijing
bj_region = ee.Geometry.Polygon( [[[115.9662455210937, 40.121362012835235],
[115.9662455210937, 39.64293313749715],
[117.01818643906245, 39.64293313749715],
[117.01818643906245, 40.121362012835235]]], None, False)

# Run the export.
doExport(bj_image_base, bj_kernel_buffer, bj_region)

# Run the prediction.
doPrediction(bj_image_base, user_folder, bj_kernel_buffer, bj_region)
```

```
# Display the output
out_image = ee.Image(user_folder + '/' + bj_image_base) mapid = out_image.getMapId({'min': 0, 'max':
1})
map = folium.Map(location=[39.898, 116.5097]) folium.TileLayer(
tiles=mapid['tile_fetcher'].url_format, attr='Map Data &copy; <a
href="https://earthengine.google.com/">Google Earth Engine</a>', overlay=True,
name='predicted impervious',
).add_to(map) map.add_child(folium.LayerControl()) map
```