МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ АЕРОНАВІГАЦІЇ, ЕЛЕКТРОНІКИ ТА ТЕЛЕКОМУНІКАЦІЙ
КАФЕДРА АЕРОКОСМІЧНИХ СИСТЕМ УПРАВЛІННЯ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускної кафедри
_____ О.М. Тачиніна
«___» _____ 2020 р.

ДИПЛОМНА РОБОТА

(Пояснювальна записка)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

ЗА СПЕЦІАЛЬНІСТЮ 151 «АВТОМАТИЗАЦІЯ ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ»

Тема: «Розпізнавання візуальних образів за результатами зйомок безпілотних апаратів»

Виконавець: студент групи СУ-201М Баборига Андрій Сергійович

Керівник: д.т.н., с.н.с. Тачиніна Олена Миколаївна

Консультант розділу «Охорона праці»: _____ (підпис) __Козлітін О.О.__ (П.І.Б)

Консультант розділу «Охорона навколишнього середовища»: _____ (підпис) __Фролов В.Ф.__ (П.І.Б)

Нормоконтролер: _____ (підпис) __Дивнич М.П.__ (П.І.Б)

КИЇВ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет аеронавігації, електроніки та телекомунікацій

Кафедра аерокосмічних систем управління

Напрям освіти: 15 «Автоматизація та приладобудування»

ЗАТВЕРДЖУЮ

Завідувач випускної кафедри

_____ О.М. Тачиніна

«___» _____ 2020 р.

ЗАВДАННЯ

на виконання дипломної роботи

Бабориги Андрія Сергійовича

1. Тема роботи «Розпізнавання візуальних образів за результатами зйомок безпілотних апаратів» затверджена наказом ректора від «27» листопада 2020 р. №2356/ст.
2. Термін виконання роботи: з 01 грудня 2020 року по 31 грудня 2020 року.
3. Вихідні дані роботи: моделі архітектури нейронних мереж: U-Net, VGGNet, ResNet, каталог зображень Google Earth Engine.
4. Зміст пояснювальної записки: аналітичний огляд літературних джерел та статей з тематики диплому. Розробка програми з розпізнавання непроникної поверхні з подальшими висновками щодо працездатності, ефективності та роботи отриманої системи.
5. Перелік обов’язкового ілюстративного матеріалу: рисунки, графіки.

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1	Розробити деталізований зміст розділів диплому	01.12.2020-02.12.2020	
2	Вступ	03.12.2020-04.12.2020	
3	Розділ 1. Аналіз існуючої проблеми	05.12.2020-06.12.2020	
4	Розділ 2. Дослідження та вибір програмних засобів	07.12.2020-08.12.2020	
5	Розділ 3. Розробка алгоритму та програми	09.12.2020-11.12.2020	
6	Розділ 4. Алгоритм виконання та приклади застосування	12.12.2020-13.12.2020	
7	Усунення недоліків дипломної роботи	14.12.2020-15.12.2020	
8	Оформлення дипломної роботи	16.12.2020-18.12.2020	

7. Консультація з окремих розділів

Назва розділу	Консультант (посада, П.І.Б.)	Підпис	
		Завдання видав	Завдання прийняв
Розділ 5. Охорона навколишнього середовища	Фролов В.Ф.		
Розділ 6. Охорона праці	Козлітін О.О.		

8. Дата видачі завдання: 5 вересня 2020 року

Керівник дипломної роботи: _____ Тачиніна О.М.
(підпис керівника)

Завдання прийняв до виконання: _____ Баборига А.С.
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту зі складної проблеми «Розпізнавання візуальних образів за результатами зйомок безпілотних апаратів»: 103 сторінки, 19 рисунків, 43 посилання.

Ключові слова: РОЗПІЗНАВАННЯ ОБРАЗІВ, НЕПРОНИКНА ПОВЕРХНІСТЬ, PYTHON, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, U-NET.

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

Об'єктом дослідження є процес розпізнавання образів для аерофотозйомки.

Предметом дослідження є алгоритм та програма розпізнавання образів для аерофотозйомки.

Методи дослідження: методи побудови архітектури нейронної мережі U-Net, VGGNet, ResNet.

Актуальність даної дипломної роботи. Програми розпізнавання образів є дуже перспективним напрямком програмного забезпечення для обробки та автоматизації. Програми розпізнавання образів конкретно з використанням аерофотозйомок можуть використовуватися для різних цілей, таких як надання додаткових джерел навігаційних даних, в надзвичайних ситуаціях (для виявлення потерпілих).

Результати, отримані в дипломній роботі, можуть бути використані при проведені наукових досліджень, у навчальному процесі та в практичній діяльності фахівців з автоматизації.

ВСТУП

Розпізнавання образів – це автоматичне розпізнавання закономірностей у наборі даних. Це – зріла, але захоплююча сфера з доволі високим темпом розвитку, яка лежить в основі розвитку спільних областей, таких як комп'ютерний зір, обробка зображень, аналіз тексту та документів та нейронні мережі. Розпізнавання образів має застосування в статистичному аналізі даних, обробці сигналів, аналізі зображень, пошуку інформації, біоінформатиці, стисненні даних та комп'ютерній графіці. Воно бере свій початок у статистиці та техніці; деякі сучасні підходи до розпізнавання образів включають використання машинного навчання завдяки збільшенню доступності великих даних та новому обсягу обробної потужності. Розпізнавання образів можна визначити як класифікацію даних на основі вже отриманих знань або статистичної інформації, вилученої із образів або їх подання.

У процесі біологічної еволюції багато тварин за допомогою зорового й слухового апарата вирішили задачу розпізнавання образів досить добре. Створення штучних систем розпізнавання образів залишається складною теоретичною й технічною проблемою. Необхідність у такому розпізнаванні виникає в найбільш різноманітних областях — від військової справи й систем безпеки до оцифровування різних аналогових сигналів.

Перспективним напрямком використання даної технології видається її використання для обробки результатів аерофотозйомки з метою визначення певних характеристик географічної локації, що досліджується, наприклад, непроникності. Існує кілька імплементацій розпізнавання образів за результатами обробки зображень, отриманих, наприклад, за допомогою супутників, кожна з яких, на жаль, має певні недоліки, як-от повільне навчання розроблених нейронних мереж, погана якість результатів розпізнавання, або ж великий об’єм даних, що вимагається для роботи програм.

Для розробки програми розпізнавання образів за результатами аерофотозйомок слід дослідити існуючі рішення розпізнавання образів, розглянути та вибрати мову програмування та супутні бібліотеки для реалізації застосунку. Необхідно також

створити або реалізувати існуючу модель нейронної мережі, що буде відповідати за розпізнавання за результатами навчання. Потрібно детально розкрити алгоритм створеної програми та його складові, а також процес виконання програми. Розроблений застосунок повинен мінімізувати проявлення недоліків, що знайдені у інших програмах даного типу, або ж і зовсім їх уникнути.

Програма, окрім свого прямого призначення, може бути використана викладачами та студентами для подальшого вдосконалення, а також у якості наочного матеріалу для дисциплін, пов’язаних з нейромережами.

У реалізації всіх вимог у розробленій програмі розпізнавання образів для аерофотозйомки значно допомогла вивчена на кафедрі дисципліна «Програмування».

1. АНАЛІЗ ІСНУЮЧОЇ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ НА ДОСЛІДЖЕННЯ

Кожен процес розробки починається з аналізу теоретичної бази проекту. Розробка програми розпізнавання образів починається з огляду концепції розпізнавання образів, її тонкощів та методів реалізації, а також аналізу існуючих рішень.

1.1. Аналіз існуючих рішень застосування розпізнавання образів

В останні роки розпізнавання образів все частіше застосовується у повсякденному житті. Розпізнавання мови та почерку значно спрощує взаємодію людина-комп'ютер, розпізнавання друкованого тексту використовується для перекладу документів в електронну форму.

Основною концепцією, яка використовується при розпізнаванні образів, є множина. У комп'ютері множина представлена набором неповторюваних елементів одного типу. Поняття "неповторюваний" означає, що якийсь елемент у множині або існує, або його немає. Універсальна множина включає всі можливі для вирішення проблеми елементи, порожня множина не містить жодної.

У класичній постановці проблеми розпізнавання універсальна множина ділиться на частини, які називаються образами. Образ предмета задається сукупністю його конкретних проявів. У випадку розпізнавання тексту, універсальна множина включатиме всі можливі символи, образ "А" включатиме всі можливі стилі цього листа, а програма розпізнавання визначатиме, яку букву символізує певна закарлючка, дана для обробки.

Метод присвоєння елементу будь-якому зображенню називається правилом прийняття рішення. Іншим важливим поняттям є метрика – спосіб визначення відстані між елементами універсальної множини. Чим менша ця відстань, тим більш

схожими є символи або, скажімо, звуки, які ми розпізнаємо. Зазвичай елементи вказуються як набір чисел, а метрика – як функція. Ефективність програми залежить від вибору подання образу та реалізації метрики; один алгоритм розпізнавання з різними показниками буде робити помилки з різною частотою (право на помилку є настільки ж типовим для програм розпізнавання, як і для людей).

Елементарний алгоритм, заснований на методі множинних стандартів, добре демонструє принцип розпізнавання образів. На його вході є навчальний зразок – вибірка A_{ij} для кожного зображення A_i, метрика d та сам розпізнаний об'єкт x. Використовуючи метрику, ми обчислюємо відстань від x до кожного елемента навчальної вибірки $d(x, a_{ij})$ і знаходимо умовну відстань $d(x, A_i)$ як відстань від x до найближчого елемента від A_i. Елемент x відноситься до найближчого образу.
На практиці тут потрібно знайти мінімальну дистанцію для кожного класу і узяти мінімум від результату.
Іншим елементарним алгоритмом є метод k-найближчих сусідів. Як випливає з назви, він вводить додатковий вхідний параметр, ціле число k. Тут все ще простіше – ми беремо k елементів навчальної вибірки, найближчих до x, і обчислюємо, скільки з них належить до якого образу. x буде належати до образу, що містить більше елементів.

В обох алгоритмах може виникнути невизначена ситуація, коли x буде знаходитися на однаковій відстані від декількох образів. У цьому випадку програма повинна або запитати у користувача, якому образу слід призначити елемент, або вибрати навмання. Це залежить від вимог до точності, з одного боку, та зручності використання, з іншого. Найкраще реалізовувати обидва варіанти одночасно.

Образ, або ж клас – це класифікаційне угрупування в системі класифікації, яке об'єднує (або виділяє) певну групу об'єктів за певною ознакою.

Образне сприйняття світу – одна з таємничих властивостей живого мозку, що дає змогу зрозуміти нескінченний потік сприйманої інформації та зберегти орієнтацію в океані розсіяних даних Про зовнішній світ. Сприймаючи зовнішній світ, ми завжди робимо класифікацію сприйманих відчуттів, тобто ділимо їх на групи подібних, але не однакових явищ. Наприклад, незважаючи на суттєву

різницю, одна група включає всі літери Ї, написані різними почерками, або всі звуки, що відповідають одній і тій же ноті, що відтворюється в будь-якій октаві та на будь-якому інструменті, а оператор, що керує технічним об’єктом, до реагує однаковим чином на цілу множину станів об’єкта. Характерно, що для того, щоб сформулювати поняття групи сприйняття певного класу, досить ознайомитися з незначною кількістю його представників. Ця властивість мозку дає можливість сформулювати таке поняття, як образ.

Образи мають характерну властивість, яка виявляється в тому, що ознайомлення з скінченною кількістю явищ з однієї і тієї ж множини дає можливість розпізнати довільно велику кількість його представників. Прикладами образів можуть бути: річка, море, рідина і т.д. Певна множина станів рослини також може розглядатися як образ, і вся ця множина станів характеризується тим, що для досягнення заданої мети вимагається однаковий вплив на об’єкт управління. Зображення мають характерні об’єктивні властивості в тому сенсі, що різні люди, які вивчають різні спостережні матеріали, здебільшого класифікують одні й ті самі об’єкти однаково і незалежно один від одного. Саме ця об’єктивність зображень дозволяє людям у всьому світі зрозуміти одне одного.

Здатність сприймати зовнішній світ у вигляді образів дозволяє з певною впевненістю розпізнати нескінченну кількість об’єктів на основі ознайомлення з їх скінченною кількістю, а об’єктивний характер основної властивості образів дозволяє нам моделювати процес їх визнання. У літературі, присвяченій проблемі розпізнавання образів, часто вводиться поняття класу замість поняття образу.

1.2. Актуальність застосування розпізнавання образів

Загалом, проблема розпізнавання образів складається з двох частин: навчання та розпізнавання. Навчання здійснюється показом окремих предметів із зазначенням їх приналежності до того чи іншого образу. В результаті навчання система розпізнавання повинна набути здатності реагувати однаковими реакціями на всі об’єкти одного зображення, а іншими – на всі об’єкти інших зображень. Дуже важливо, щоб процес навчання закінчувався лише показом скінченної кількості

об’єктів без будь-яких інших підказок. Об’єктами навчання можуть бути або картинки, або інші наочні зображення (наприклад, букви, цифри), або різні явища зовнішнього світу, скажімо, звуки, стан організму під час медичної діагностики, стан технічного об’єкта в системах управління, тощо. Важливо, щоб у процесі навчання вказувались лише самі предмети та їх приналежність до зображення. За навчанням йде процес впізнавання нових об’єктів, що характеризує дії вже навченої системи. Автоматизація цих процедур становить проблему навчання розпізнаванню образів. У тому випадку, коли людина здогадується або придумує, а потім накладає на машині правило класифікації, проблема розпізнавання частково вирішується, оскільки людина бере на себе основну частину проблеми, яка полягає у навчанні.

Коло завдань, які можна вирішити за допомогою систем розпізнавання, надзвичайно широке. Сюди входять не тільки завдання розпізнавання зорових та слухових зображень, але також завдання розпізнавання складних процесів і явищ, що виникають, наприклад, при виборі відповідних дій керівником підприємства або виборі оптимального управління технологічними, економічними, транспортними або військових операцій. У кожній із цих проблем аналізуються деякі явища, процеси, стани зовнішнього світу, які надалі називаються об’єктами спостереження. Перш ніж розпочати аналіз будь-якого об’єкта, потрібно отримати деяку інформацію про нього, упорядковану певним чином. Така інформація є характеристикою об’єктів, їх відображення на сукупності сприймаючих органів системи розпізнавання.

Але кожен об’єкт спостереження може впливати на систему по-різному, залежно від умов сприйняття. Наприклад, будь-яка буква, навіть якщо вона написана однаково, може бути зміщена щодо сприймаючих органів за бажанням. Крім того, предмети одного зображення можуть сильно відрізнятися один від одного і, природно, по-різному впливати на сприймаючі органи.

Коло завдань, які можна вирішити за допомогою систем розпізнавання, надзвичайно широке. Сюди входять не тільки завдання розпізнавання зорових та слухових образів, але і завдання розпізнавання складних процесів і явищ, що виникають, наприклад, при виборі відповідних дій керівником підприємства або виборі оптимального управління технологічними, економічними, транспортними або

військовими операціями. У кожній із цих проблем аналізуються деякі явища, процеси, стани зовнішнього світу, які надалі називаються об’єктами спостереження. Перш ніж розпочати аналіз будь-якого об’єкта, потрібно отримати деяку інформацію про нього, упорядковану певним чином. Така інформація є характеристикою об’єктів, їх відображення на сукупності сприймаючих органів системи розпізнавання.

Щоправда, кожен об’єкт спостереження може впливати на систему по-різному, залежно від умов сприйняття. Наприклад, будь-яка літера, навіть якщо вона написана однаково, може бути зміщена щодо сприймаючих органів. Крім того, предмети одного зображення можуть сильно відрізнятися один від одного і, природно, по-різному впливати на сприймаючі органи.

Кожне відображення об’єкта до сприймаючих органів системи розпізнавання, незалежно від його положення щодо цих органів, зазвичай називається зображенням об’єкта, і множини таких зображень, об’єднані якимись загальними властивостями, називаються образами.

При вирішенні завдань управління методами розпізнавання зображень замість поняття “образ” використовується термін “стан”. Стан – це певна форма відображення виміряних поточних (або миттєвих) характеристик спостережуваного об’єкта. Сукупність держав визначає ситуацію. Поняття "ситуація" є аналогічним поняттю "образ". Однак ця аналогія є неповною, оскільки не кожне зображення можна назвати ситуацією, хоча кожну ситуацію можна назвати образом.

Ситуація визначається як певна сукупність станів складного об’єкта, кожен з яких характеризується однаковими або подібними характеристиками об’єкта. Наприклад, якщо певний об’єкт управління розглядається як об’єкт спостереження, то ситуація об’єднує такі стани цього об’єкта, в яких слід застосовувати ті самі дії управління.

Вибір початкового опису об’єктів є одним із центральних завдань проблеми розпізнавання образів. При вдалому виборі початкового опису проблема розпізнавання може виявитися тривіальною і, навпаки, невдало обраний початковий опис може призвести або до дуже складної подальшої обробки інформації, або до відсутності рішення взагалі. Наприклад, якщо вирішується проблема розпізнавання

об’єктів, що відрізняються за кольором, а в якості початкового опису вибрані сигнали, отримані від датчиків ваги, тоді проблему розпізнавання взагалі не можна вирішити.

Кожного разу, коли ми стикаємось із незнайомими завданнями, виникає природне бажання представити їх у вигляді якоїсь легкозрозумілої моделі, яка дозволила б нам зрозуміти завдання в термінах, які легко відтворюються нашими уявленнями. І оскільки ми існуємо в просторі та в часі, найбільш зрозумілим для нас є просторово-часове тлумачення завдань.

Будь-який образ, яке з’являється в результаті спостереження за об’єктом під час тренувань або іспиту, може бути представлений як вектор, а, отже, як точка в якомусь просторі об’єктів. Якщо стверджується, що під час показу зображень можна однозначно віднести їх до одного з двох (або декількох) зображень, то тим самим стверджується, що в певному просторі є дві (або кілька) областей, які не мають спільних точок, і що образи – це точки з цих областей. Кожній такій області може бути присвоєно ім’я, тобто ім’я, яке відповідає образу.

Тепер інтерпретуємо процес розпізнавання образів з точки зору геометричної картини, обмежившись поки випадком розпізнавання лише двох образів. Заздалегідь вважається відомим лише те, що потрібно відокремити дві області в певному просторі і що відображаються лише точки з цих областей. Самі ці ділянки не визначені заздалегідь, тобто немає інформації про розташування їх меж або правила визначення належності точки до певної області. Під час навчання подаються випадково обрані з цих областей бали та передається інформація про те, до якої області належать представлені бали. Під час навчання жодної додаткової інформації про ці райони, тобто про розташування їх меж, не надається. Метою тренування є або побудова поверхні, яка розділяла б не тільки точки, показані в процесі тренування, але і всі інші точки, що належать до цих областей, або побудова поверхонь, які обмежували б ці області так, щоб кожна з них містила лише точки одного зображення. Іншими словами, метою навчання є побудова функцій із векторів зображень, які, наприклад, були б додатніми в усіх точках одного і від’ємними в усіх точках іншого зображення. Через те, що в областях немає спільних точок, завжди існує ціла множина таких

розділювальних функцій, і в результаті навчання одна з них повинна бути побудована.

Якщо представлені зображення належать не двом, а більшій кількості зображень, то завдання полягає в тому, щоб побудувати на основі точок, показаних під час тренування, поверхню, що відокремлює всі ділянки, що відповідають цим зображенням, одна від одної. Цю проблему можна вирішити, наприклад, задавши функцію, яка приймає однакове значення по точках кожного з регіонів; значення цієї функції повинно бути різним по точках з різних регіонів.

На перший погляд здається, що знання лише певної кількості точок з області недостатньо, щоб відокремити всю область. Дійсно, можна вказати нескінченну кількість різних областей, що містять ці точки, і незалежно від того, як з них будується поверхня, яка виділяє область, завжди можна вказати іншу область, яка перетинає поверхню і одночасно містить вказані точки. Однак відомо, що проблема апроксимації функції з інформації про неї в обмеженій множині точок, яка набагато вужча від усієї множини, на якій дана функція, є загальною математичною проблемою апроксимації функцій. Звичайно, вирішення таких проблем вимагає введення певних обмежень щодо класу функцій, що розглядається, і вибір цих обмежень залежить від характеру інформації, яку вчитель може додати в процесі навчання. Однією з цих підказок є гіпотеза компактності зображення. Інтуїтивно зрозуміло, що наближення розділювальної функції буде тим простішим завданням, чим компактнішими і більше рознесеними будуть області, які слід розділити. Поряд з геометричним тлумаченням проблеми розпізнавання образів існує ще один підхід, який називається структурним, або лінгвістичним. Пояснимо лінгвістичний підхід на прикладі розпізнавання зорових образів. Спочатку визначається сукупність початкових понять. Це типові фрагменти, знайдені на зображеннях, та характеристики їх взаємного розташування – «ліворуч», «знизу», «всередині» тощо. Ці початкові поняття утворюють словник, що дозволяє будувати різні логічні висловлювання, які іноді називають припущеннями. Завдання полягає у виборі тверджень, найбільш значущих для даного конкретного випадку, з великої кількості тверджень, які можна було б побудувати за допомогою цих понять.

Далі, розглядаючи скінченну і, якщо можливо, невелику кількість об’єктів з кожного образу, потрібно побудувати опис цих образів. Побудовані описи повинні бути достатньо повними, щоб вирішити питання, до якого образу належить цей об’єкт. При реалізації лінгвістичного підходу виникають два завдання: завдання побудови початкового словника, тобто набору типових фрагментів, та завдання побудови правил опису з елементів даного словника. У рамках лінгвістичної інтерпретації проводиться аналогія між структурою образів та синтаксисом мови. Прагнення до цієї аналогії було викликане вмінням користуватися апаратом математичної лінгвістики, тобто методи мають синтаксичний характер. Використання апарату математичної лінгвістики для опису структури зображень можна використовувати лише після того, як було здійснено сегментацію зображень на їх складові частини, тобто були розроблені слова для опису типових фрагментів та методів їх пошуку. Після попередньої роботи, яка забезпечує підбір слів, виникають власне мовні завдання, що складаються із завдань автоматичного граматичного синтаксичного аналізу описів для розпізнавання зображень. Водночас з’являється самостійний напрямок досліджень, який вимагає не лише знання основ математичної лінгвістики, а й оволодіння техніками, розробленими спеціально для лінгвістичної обробки зображень.

Якщо припустити, що в процесі навчання простір ознак формується на основі задуманої класифікації, то можна сподіватися, що специфікація простору ознак сама по собі задає властивість, під впливом якої образи в цьому просторі можна легко розрізнити. Саме ці надії стимулювали появу гіпотези про компактність під час розвитку розпізнавання образів. Гіпотеза формулюється наступним чином: компактні множини в просторі ознак відповідають зображенням. На даний момент під компактною множиною маються на увазі певні «скупчення» точок у просторі зображень, припускаючи, що між цими скупченнями існують «розрідження», що розділяють їх.

Однак не завжди вдалося експериментально підтвердити цю гіпотезу, але, головне, усі без винятку завдання, в рамках яких спостерігалося підтвердження гіпотези компактності, знайшли просте рішення. І навпаки, ті завдання, для яких

гіпотеза не була підтверджена, або взагалі не вирішувались, або вирішувались із великими труднощами із залученням додаткових «фокусів». Цей факт сіяв сумніви у справедливості гіпотези про компактність, оскільки для спростування будь-якої гіпотези достатньо одного прикладу, що заперечує її. У той же час, підтвердження гіпотези скрізь, де можна було вирішити проблему навчання розпізнавання образів, підігрівало інтерес до цієї гіпотези. Сама гіпотеза компактності стала ознакою можливості задовільного вирішення проблем розпізнавання.

1.3. Класифікація методів розпізнавання образів

Розпізнавання образів відноситься до проблеми побудови та застосування формальних операцій над числовими або символічними поданнями об’єктів у реальному чи ідеальному світі, результати розв’язання яких відображають відношення еквівалентності між цими об’єктами. Співвідношення еквівалентності виражають належність оцінюваних об’єктів до будь-яких класів, що розглядаються як незалежні семантичні одиниці.

При побудові алгоритмів розпізнавання класи еквівалентності можуть бути вказані дослідником, який використовує власні ідеї або зовнішню додаткову інформацію про подібність та відмінність об’єктів у контексті вирішуваної проблеми. Це призводить до процесу, який зазвичай називають «навчанням з учителем». В іншому випадку, тобто коли автоматизована система вирішує проблему класифікації без залучення зовнішньої навчальної інформації, це називається автоматичною класифікацією, або ж «навчанням без нагляду». Більшість алгоритмів розпізнавання образів вимагають залучення дуже значних обчислювальних потужностей, які можуть бути забезпечені лише за допомогою високопродуктивних комп’ютерних технологій.

Різні автори подають різні типології методів розпізнавання образів. Деякі автори розрізняють параметричні, непараметричні та евристичні методи, а інші - групи методів, заснованих на історично складених школах та тенденціях у цій галузі. Д. А. Поспєлов (1990) визначає два основні способи представлення знань:

- *Інтенсивне представлення*, у формі діаграми взаємозв'язків між атрибутами (ознаками);
- *Екстенсійне представлення*, засноване на використанні конкретних фактів (об'єктів, прикладів).

Інтенсивне представлення фіксує закономірності та взаємозв'язки, що пояснюють структуру даних. Що стосується діагностичних завдань, то така фіксація полягає у визначенні операцій над атрибутами (ознаками) об'єктів, які призводять до необхідного діагностичного результату. Інтенсивні представлення реалізуються за допомогою операцій над значеннями атрибутів і не передбачають операцій над конкретними інформаційними фактами (об'єктами).

У свою чергу, екстенсивні представлення знань пов'язані з описом та фіксацією конкретних об'єктів із предметної області і реалізуються в операціях, елементами яких є об'єкти як цілісні системи.

Можна провести аналогію між інтенсіональним та екстенсіональним уявленнями про знання та механізмами, що лежать в основі діяльності лівої та правої півкуль мозку людини. Якщо для правої півкулі характерне цілісне прототипове представлення навколишнього світу, то ліва півкуля оперує образами, що відображають зв'язки атрибутів цього світу.

Два основні способи представлення знань пропонують таку класифікацію методів розпізнавання образів: інтенсивні, які базуються на операціях з ознаками, та екстенсивні, які базуються на операціях з об'єктами.

У наведеній нижче класифікації основна увага приділяється формальним методам розпізнавання образів, а отже, розгляд евристичного підходу до розпізнавання, який розвивається в експертних системах, опущений. Обмежимося лише кількома зауваженнями щодо цього підходу.

Евристичний підхід базується на важко формалізованих знаннях та інтуїції дослідника. При такому підході дослідник сам визначає, яку інформацію та як використовувати для досягнення необхідного ефекту розпізнавання.

Відмінною особливістю інтенсивних методів є те, що вони використовують різні характеристики ознак та їх взаємозв'язки як елементи операцій при побудові та

застосуванні алгоритмів розпізнавання образів. Такими елементами можуть бути окремі значення або інтервали значень атрибутів, середніх значень і дисперсій, матриці агрегування атрибутів тощо, над якими виконуються дії, виражені в аналітичній або конструктивній формі. У той же час об'єкти в цих методах не розглядаються як цілісні інформаційні одиниці, а виступають як показники для оцінки взаємодії та поведінки їх атрибутів. Група методів інтенсивного розпізнавання образів велика, і їх поділ на підкласи певною мірою є умовним.

Методи, засновані на оцінках щільності розподілу значень ознак, запозичені з класичної теорії статистичних рішень, в якій об'єкти дослідження розглядаються як реалізації багатовимірної випадкової величини, розподіленої в просторі ознак за деяким законом. Вони базуються на байєсівській схемі прийняття рішень, яка апелює до апріорних ймовірностей належності об'єктів до певного впізнаваного класу та умовних щільностей розподілу значень вектора ознак. Ці методи зводяться до визначення коефіцієнта ймовірності в різних областях багатовимірного простору ознак.

Група методів, заснованих на оцінці щільності розподілу значень ознак, безпосередньо пов'язана з методами дискримінантного аналізу. Байєсівський підхід до прийняття рішень є одним із так званих параметричних методів, найбільш розроблених у сучасній статистиці, для яких аналітичне вираження закону розподілу (в даному випадку нормального закону) вважається відомим і лише невелика кількість параметрів (вектори середніх значень та матриці коваріації) повинна бути оціненою.

Ця група також включає метод розрахунку коефіцієнта ймовірності для незалежних ознак. Цей метод, за винятком припущення про незалежність ознак (що насправді практично ніколи не виконується), не передбачає знання функціональної форми закону розподілу. Отож, його можна класифікувати як непараметричний.

Інші непараметричні методи, що застосовуються, коли форма кривої щільності розподілу невідома і жодних припущень про її природу взагалі не можна робити, займають особливе положення. До них належать метод багатовимірної гістограми, метод *k*-найближчих сусідів, метод евклідової відстані, метод потенційних функцій

тощо. Ці методи формально діють з об’єктами як цілісні структури, але залежно від типу завдання розпізнавання вони можуть діяти як інтенсивно, так і екстенсивно.

Непараметричні методи аналізують відносну кількість об’єктів, що потрапляють у задані багатовимірні об’єми, і використовують різні функції відстані між об’єктами навчальної вибірки та розпізнаними об’єктами. Для кількісних ознак, коли їх кількість набагато менша за обсяг вибірки, операції з об’єктами відіграють проміжну роль в оцінці локальних щільностей умовного розподілу ймовірності, а об’єкти не несуть семантичного навантаження незалежних інформаційних одиниць. У той же час, коли кількість ознак співмірна з чи перевищує кількість досліджуваних об’єктів, і ознаки мають якісний або дихотомічний характер, тоді не може бути й мови про якісь локальні оцінки щільності розподілу ймовірностей. У цьому випадку об’єкти в цих непараметричних методах розглядаються як самостійні інформаційні одиниці (цілісні емпіричні факти), і ці методи набувають значення оцінки подібності та різниці досліджуваних об’єктів.

Таким чином, однакові технологічні операції непараметричних методів, залежно від умов задачі, мають сенс або для локальних оцінок щільності ймовірності значень ознак, або для оцінок подібності та різниці об’єктів.

Основними труднощами при застосуванні цих методів вважаються необхідність запам’ятовувати всю навчальну вибірку для обчислення оцінок місцевих щільностей розподілу ймовірностей та висока чутливість до нерепрезентативності навчальної вибірки.

Методи, засновані на припущеннях про клас функцій прийняття рішення є іншою групою інтенсивних методів. У цій групі методів загальний вигляд функції прийняття рішення вважається відомим і встановлюється функціонал її якості. На основі цього функціоналу, використовуючи послідовність тренувань, шукається найкраще наближення функції прийняття рішення. Найбільш поширеними є подання функцій рішення у вигляді лінійних та узагальнених нелінійних многочленів. Функціонал якості рішення правила зазвичай пов’язаний з класифікаційною помилкою.

Основною перевагою методів, заснованих на припущеннях про клас функцій прийняття рішень, є чіткість математичного формулювання задачі розпізнавання як проблеми пошуку екстремуму. Рішення цієї проблеми часто досягається за допомогою будь-яких градієнтних алгоритмів. Різноманітність методів у цій групі пояснюється широким спектром використовуваних функціоналів якості правил прийняття рішень та алгоритмами пошуку екстремуму. Методом стохастичного наближення є узагальнення розглянутих алгоритмів, які включають, зокрема, алгоритм Ньютона, алгоритми типу персептрон тощо. На відміну від методів параметричного розпізнавання, успіх цієї групи методів не стільки залежить від розбіжності теоретичних концепцій законів розподілу об'єктів у просторі ознак з емпіричною реальністю. Всі операції підпорядковані одній головній меті – знайти екстремум якісного функціоналу правила прийняття рішення. У той же час результати параметричного та розглянутих методів можуть бути подібними. Параметричні методи для випадку нормальних розподілів об'єктів у різних класах з рівними матрицями коваріації призводять до лінійних функцій рішення.

Можливості градієнтних алгоритмів для пошуку екстремуму, особливо в групі лінійних правил прийняття рішень, добре вивчені. Збіжність цих алгоритмів доведена лише для випадку, коли впізнавані класи об'єктів відображаються у просторі об'єктів компактними геометричними структурами. Однак бажання досягти достатньої якості правила прийняття рішення часто можна задовольнити за допомогою алгоритмів, які не мають суворого математичного доказу збіжності рішення до глобального екстремуму.

Ці алгоритми включають велику групу евристичних процедур програмування, що представляють напрямок еволюційного моделювання. Еволюційне моделювання – це біонічний метод, запозичений у природи. Він заснований на використанні добре відомих механізмів еволюції, щоб замінити процес змістовного моделювання складного об'єкта феноменологічним моделюванням його еволюції.

Відомим представником еволюційного моделювання в розпізнаванні образів є метод групового розгляду аргументів (МГРА). МГРА базується на принципі самоорганізації, а алгоритми МГРА відтворюють схему масового відбору. В

алгоритмах МГРА коефіцієнти узагальненого многочлена, який часто називають многочленом Колмогорова-Габора, синтезуються та відбираються особливим чином. Цей синтез та відбір здійснюється із зростаючою складністю, і неможливо заздалегідь передбачити, якою буде остаточна форма узагальненого многочлена. Спочатку зазвичай розглядаються прості попарні комбінації вихідних ознак, з яких формуються рівняння функцій прийняття рішень, як правило, не вище другого порядку. Кожне рівняння аналізується як незалежна функція прийняття рішень, і значення параметрів рівнянь так чи інакше знаходять із навчальної вибірки. Потім із отриманого набору функцій прийняття рішення вибираються одні з найкращих. Перевірка якості окремих функцій прийняття рішень проводиться на контрольній вибірці, яку іноді називають принципом зовнішнього додавання. Вибрані часткові функції рішення розглядаються далі як проміжні змінні, які служать початковими аргументами для подібного синтезу нових функцій рішення тощо. Процес такого ієрархічного синтезу триває до досягнення екстремуму критерію якості функції рішення, який на практиці виявляється в погіршенні цієї якості при спробі подальшого збільшення порядку доданків полінома відносно вихідних ознак.

Принцип самоорганізації, що лежить в основі МГРА, називається евристичною самоорганізацією, оскільки весь процес заснований на введенні зовнішніх доповнень, вибраних евристично. Результат рішення може суттєво залежати від цих евристик. Отримана діагностична модель залежить від того, як об’єкти поділяються на навчальні та тестові зразки, як визначається критерій якості розпізнавання, скільки змінних передається в наступну серію відбору тощо.

Логічні методи розпізнавання образів базуються на апараті алгебри логіки і дозволяють оперувати інформацією, що міститься не тільки в окремих ознаках, але і в комбінаціях значень ознак. У цих методах значення будь-якого атрибута розглядаються як елементарні події.

У найзагальнішому вигляді логічні методи можна охарактеризувати як різновид пошуку навчальної вибірки логічних закономірностей та формування певної системи правил логічного рішення (наприклад, у формі сполучників елементарних подій),

кожне з яких має власну вагу. Група логічних методів різноманітна і включає методи різної складності та глибини аналізу.

Лінгвістичні методи розпізнавання образів засновані на використанні спеціальних граматик генеративних мов, за допомогою яких можна описати набір властивостей розпізнаних об'єктів.

Для різних класів об'єктів виділяють непохідні елементи (образи, ознаки) та можливі зв'язки між ними. Граматика посилається на правила побудови об'єктів з цих непохідних елементів. Таким чином, кожен об'єкт представлений набором непохідних елементів, «пов'язаних» між собою тим чи іншим чином, або, іншими словами, «реченням» якоїсь «мови». За допомогою синтаксичного аналізу (граматичного аналізу) «речення» встановлюється його синтаксична «правильність» або, що еквівалентно, чи може якась фіксована граматика (що описує клас) створити існуючий опис об'єкта. Синтаксичний аналіз виконується так званим «синтаксичним аналізатором», який представляє повний синтаксичний опис об'єкта у вигляді дерева синтаксичного аналізу, якщо об'єкт синтаксично правильний (тобто належить до класу, описаного цією граматикою). В іншому випадку об'єкт або відхиляється, або аналізується з іншими граматиками, що описують інші класи об'єктів. Існують безконтекстні, автоматичні та інші типи граматики. Однак завдання відновлення (визначення) граматики з певного набору висловлювань (речень, що є описом об'єктів), що породжують дану мову, важко формалізувати. Література містить опис евристичних правил для автоматичного відновлення граматики для побудови та застосування лінгвістичних алгоритмів для розпізнавання шаблонів.

У екстенсивних методах, на відміну від інтенсивного напрямку, кожному досліджуваному об'єкту надається незалежне діагностичне значення. По суті, ці методи близькі, скажімо, до медичного підходу, який розглядає людей не як ланцюжок об'єктів, класифікованих за тим чи іншим показником, а як цілісні системи, кожна з яких індивідуальна і має особливе діагностичне значення. Таке дбайливе ставлення до об'єктів дослідження не дозволяє виключити або втратити інформацію про кожен окремий об'єкт, що трапляється при використанні методів

інтенсивного напрямку, використовуючи об’єкти лише для виявлення та фіксації закономірностей поведінки їх атрибутів.

Основними операціями розпізнавання образів за допомогою обговорюваних методів є операції визначення подібності та різниці об’єктів. Об’єкти цієї групи методів відіграють роль діагностичних прецедентів. Водночас, залежно від умов конкретного завдання, роль окремого прецеденту може коливатися в найширшому діапазоні – від основної до дуже опосередкованої участі у процесі розпізнавання. У свою чергу, умови задачі можуть вимагати для успішного рішення участі різної кількості діагностичних прецедентів – від одного у кожному розпізнаному класі до загального обсягу вибірки, а також різних методів обчислення показників подібності та різниці об’єктів.

Метод порівняння прототипів – це найпростіший метод розширеного розпізнавання. Він використовується, наприклад, коли розпізнані класи відображаються у просторі об’єктів як компактні геометричні групування. У цьому випадку зазвичай точкою прототипу вибирається центр геометричної групування класу (або найближчий до центру об’єкт). Для класифікації невідомого об’єкта знаходиться найближчий прототип, і об’єкт належить до того ж класу, що і цей прототип. Очевидно, що за допомогою цього методу не генеруються зображення загального класу.

Різні типи відстаней можуть бути використані як міра близькості. Відстань Хеммінга, яка в цьому випадку дорівнює квадрату евклідової відстані, часто використовують для дихотомічних ознак. У цьому випадку правило прийняття рішення для класифікації об’єктів еквівалентно лінійній функції прийняття рішення.

Цей факт слід особливо відзначити. Це наочно демонструє зв’язок між прототипом та характерним поданням інформації про структуру даних. У свою чергу, якщо аналіз просторової структури розпізнаних класів дозволяє зробити висновок, що вони геометрично компактні, то кожен з цих класів може бути замінений одним прототипом, що насправді еквівалентно лінійній діагностичній моделі.

На практиці, звичайно, ситуація часто відрізняється від описаного ідеалізованого прикладу. Дослідник, який має намір застосувати метод

розпізнавання, заснований на порівнянні з прототипами діагностичних класів, стикається зі складними проблемами. Це, насамперед, вибір міри близькості (метрики), від якої просторова конфігурація розподілу об'єктів може суттєво змінитися. Також самостійною проблемою є аналіз багатовимірних структур експериментальних даних. Обидві ці проблеми особливо гострі для дослідника в умовах високої розмірності простору ознак, що характерно для реальних проблем.

Метод k-найближчих сусідів для вирішення проблем дискримінантного аналізу був вперше запропонований ще в 1952 році. Він сформульований наступним чином. При класифікації невідомого об'єкта виявляється задана кількість (k) інших об'єктів (найближчих сусідів), геометрично найближчих до нього в просторі атрибутів, з уже відомою належністю до розпізнаних класів. Рішення про присвоєння невідомого об'єкта тому чи іншому діагностичному класу приймається шляхом аналізу інформації про цю відому приналежність найближчих сусідів, наприклад, за допомогою простого підрахунку голосів.

Спочатку метод k-найближчих сусідів розглядався як непараметричний метод оцінки коефіцієнта правдоподібності. Для цього методу отримано теоретичні оцінки його ефективності у порівнянні з оптимальним байєсівським класифікатором. Доведено, що ймовірності асимптотичних помилок для методу k-найближчих сусідів перевищують помилки правила Байєса не більше ніж у два рази.

Як зазначалося вище, у реальних проблемах часто доводиться оперувати об'єктами, які описуються великою кількістю якісних (дихотомічних) ознак. У цьому випадку розмір простору ознак співмірний або перевищує обсяг досліджуваної вибірки. В таких умовах зручно інтерпретувати кожен об'єкт навчальної вибірки як окремий лінійний класифікатор. Тоді той чи інший діагностичний клас представлений не одним прототипом, а набором лінійних класифікаторів. Кумулятивна взаємодія лінійних класифікаторів призводить до кусково-лінійної поверхні, яка відокремлює впізнавані класи в просторі ознак. Вигляд розділової поверхні, що складається з шматків гіперплощин, може бути різним і залежить від взаємного розташування заповнювачів, що підлягають класифікації.

Також можливо використовувати іншу інтерпретацію механізмів класифікації k-найближчого сусіда. Вона базується на ідеї існування деяких прихованих змінних, абстрактних або пов'язаних з певним перетворенням з вихідним простором ознак. Якщо в просторі прихованих змінних попарні відстані між об'єктами такі ж, як у просторі початкових ознак, і кількість цих змінних набагато менша за кількість об'єктів, то інтерпретацію методу k-найближчих сусідів можливо розглядати з точки зору порівняння непараметричних оцінок щільності умовного розподілу ймовірності. Представлена тут концепція прихованих змінних близька за своїм характером до концепції справжньої розмірності та інших подань, що використовуються в різних методах зменшення розмірності.

Використовуючи метод k-найближчих сусідів для розпізнавання образів, дослідник повинен вирішити складну проблему вибору метрики для визначення близькості об'єктів, що діагностуються. Ця проблема в умовах високої розмірності простору об'єктів надзвичайно загострюється через достатню трудомісткість цього методу, що стає значною навіть для високопродуктивних комп'ютерів. Тому тут, як і в методі порівняння з прототипом, необхідно вирішити творчу задачу аналізу багатовимірної структури експериментальних даних, щоб мінімізувати кількість об'єктів, що представляють діагностичні класи. На думку авторів, необхідність зменшення кількості об'єктів у навчальній множині є недоліком цього методу, оскільки зменшує репрезентативність навчальної вибірки.

Принцип дії алгоритмів обчислення оцінок (АОО) полягає в обчисленні балів подібності, що характеризує «близькість» розпізнаних та опорних об'єктів системою ансамблів ознак, що є системою підмножин заданого набору ознак .

На відміну від усіх розглянутих раніше методів, алгоритми обчислення оцінок функціонують принципово по-новому з описом об'єктів. Для цих алгоритмів об'єкти існують одночасно у дуже різних підпросторах простору ознак. Клас АОО підводить ідею використання ознак до їх логічного завершення: оскільки не завжди відомо, які комбінації ознак є найбільш інформативними, в АОО ступінь подібності об'єктів обчислюється шляхом порівняння всіх можливих або певних комбінацій ознак, включених до описи об'єктів.

Використовувані комбінації ознак (підпростори) називаються наборами підтримки або наборами часткових описів об’єктів. Вводиться поняття узагальненої близькості між розпізнаним об’єктом та об’єктами навчальної вибірки (з відомою класифікацією), які називаються еталонними об’єктами. Ця близькість представлена комбінацією близькості об’єкта, що розпізнається, з опорними об’єктами, розрахованими на наборах часткових описів. Таким чином, АОО є продовженням методу k-найближчих сусідів, при якому близькість об’єктів розглядається лише в одному заданому просторі ознак.

Ще одним розширенням АОО є те, що в цих алгоритмах проблема визначення подібності та різниці об’єктів формулюється як параметрична і виділяється етап налаштування АОО на основі навчальної вибірки, на якій оптимальні значення введених параметрів вибраний. Критерієм якості є помилка розпізнавання, і буквально все параметризується:

- правила розрахунку близькості об’єктів за індивідуальними ознаками,
- правила розрахунку близькості об’єктів у підпросторах об’єктів,
- ступінь важливості того чи іншого еталонного об’єкта як діагностичного прецеденту,
- значимість внеску кожної еталонної множини ознак до остаточної оцінки схожості об’єкта, що розпізнається, з деяким діагностичним класом.

Параметри АОО встановлюються у вигляді порогових значень та (або) як вага зазначених компонентів. Теоретичні можливості АОО перевищують або принаймні не нижче можливостей будь-якого іншого алгоритму розпізнавання образів, оскільки за допомогою АОО можуть бути реалізовані всі мислимі операції з досліджуваними об'єктами. Але, як це зазвичай буває, розширення потенційних можливостей стикається з великими труднощами при їх практичній реалізації, особливо на етапі побудови (настройки) алгоритмів цього типу. Деякі труднощі були відзначені раніше при обговоренні методу k-найближчих сусідів, який можна трактувати як усічену версію АОО. Його також можна розглянути у параметричній формі, а задачу можна звести до пошуку зваженої метрики обраного типу. У той же час вже тут для задач розмірності виникають складні теоретичні питання та

проблеми, пов'язані з організацією ефективного обчислювального процесу. Для АОО, якщо ми спробуємо повністю використати потенційні можливості цих алгоритмів, ці труднощі зростають у рази.

Зазначені проблеми пояснюють той факт, що на практиці використання АОО для вирішення великих розмірних задач супроводжується введенням будь-яких евристичних обмежень та припущень. Зокрема, відомий приклад використання АОО в психодіагностиці, в якому випробовуються різноманітні АОО, що насправді еквівалентно методу k-найближчих сусідів.

1.4. Постановка задачі на дослідження

Існуючі методи розпізнавання образів, що використовуються для аерозйомок з БПЛА, мають недоліки, а саме:

- Повільне навчання мереж, що займаються розпізнаванням;
- Використання великого об’єму диску для роботи;
- Низька якість результатів.

У зв’язку з цим для подолання цих недоліків необхідно обрати модель нейронної мережі для навчання, що буде уникати (а по можливості і зовсім не мати) описаних вище проблем, а також буде доступною для реалізації наявними програмними засобами. Збереження даних, а також виконання самого розпізнавання можуть бути перенесені на хмарні сховища та сервери, що надзвичайно сильно розвантажить персональний комп’ютер дослідника. Таким чином, програма зможе бути запущена і на слабких машинах, що мають доступ до мережі Інтернет.

Висновки до розділу 1

Розпізнавання образів – це процес розпізнавання образів за допомогою алгоритму машинного навчання. Розпізнавання образів можна визначити як класифікацію даних на основі вже отриманих знань або статистичної інформації, вилученої з образів та/або їх подання. Одним з важливих аспектів розпізнавання образів є його прикладний потенціал, наприклад, розпізнавання мови, ідентифікація

мовця, розпізнавання мультимедійних документів, автоматична медична діагностика.

Проведено огляд методів розпізнавання образів, а також визначено їхні ключові недоліки. На основі цього вирішено створити програму, яка допоможе розпізнавати різні образи на основі аерофотозйомки і уникне вищеназваних недоліків. Вона може бути використана іншими студентами університету, якщо хтось вирішить вдосконалити мій проект. Навряд чи можливо створити таку комп'ютерну програму просто за допомогою мови програмування високого рівня без використання будь-яких додаткових інструментів. Може існувати платформа, яка може сприяти розробці такого проекту. Наступний розділ буде присвячений пошуку такого інструменту.

2. ДОСЛІДЖЕННЯ ТА ВИБІР ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РОЗРОБКИ МОДЕЛІ ТА ПРОГРАМИ РОЗПІЗНАВАННЯ ОБРАЗІВ

У попередньому розділі було розглянуто особливості розпізнавання зразків та які його основні поняття. Тепер необхідно розглянути процес розробки реальної програми розпізнавання образів та інструменти, використані для її створення.

2.1. Дослідження використання та вибір мови програмування для розпізнавання образів

Вирішення задач розпізнавання образів можливе на достатньо широкому діапазоні мов програмування. Серед них, наприклад, C#, C++, Java.

Легкість освоєння і написання коду, а також доступність великої кількості сторонніх бібліотек, призначених саме для задач розпізнавання образів, призвели до вибору мови Python із використанням додаткових бібліотек як платформи для розробки.

Python був створений наприкінці 1980-х і вперше випущений в 1991 році Гвідо ван Россумом як наступник мови програмування ABC. Python 2.0, випущений в 2000 році, представив нові функції, такі як розуміння списків, та систему збору сміття з підрахунком посилань, і був припинений з версією 2.7 у 2020 році. Python 3.0, випущений у 2008 році, був серйозним переглядом мови, при тому не повністю зворотньо сумісним, і більша частина коду Python 2 не працює незміненим на Python 3.

Python – інтерпретована, об’єктно-орієнтована мова програмування високого рівня з динамічною семантикою. Вбудовані структури даних високого рівня в поєднанні з динамічним набором тексту та динамічним прив’язуванням роблять його дуже привабливим для швидкої розробки додатків, а також для використання в якості мови сценаріїв або склеювання для з’єднання існуючих компонентів разом. Простий,

легкий у засвоєнні синтаксис Python підкреслює читабельність і, отже, зменшує витрати на обслуговування програми. Python підтримує модулі та пакети, що заохочує модульність програми та повторне використання коду. Інтерпретатор Python та велика стандартна бібліотека доступні у вихідній або двійковій формі безкоштовно для всіх основних платформ, включаючи Windows, яка використовувалася під час розробки основної програми, і можуть вільно розповсюджуватися. В даний час вона пов’язана з Java як другою за популярністю мовою програмування у світі.

Python – мова програмування з багатьма парадигмами. Об'єктно-орієнтоване програмування та структуроване програмування повністю підтримуються, і багато його функцій підтримують функціональне програмування та аспектно-орієнтоване програмування (в тому числі метапрограмуванням та метаоб'єктами (магічні методи)). Багато інших парадигм підтримуються за допомогою розширень, включаючи проектування за контрактом та логічне програмування.

Python використовує динамічне введення тексту та комбінацію підрахунку посилань та збирач сміття, що визначає цикл, для управління пам’яттю. Він також має динамічну роздільну здатність імен (пізнє прив'язування), яка зв'язує імена методів та змінних під час виконання програми.

Дизайн Python пропонує певну підтримку функціонального програмування за традицією Lisp. Він має функції фільтрування, картографування та зменшення; перелічити розуміння, словники, набори та вирази генератора. Стандартна бібліотека має два модулі (itertools та functools), що реалізують функціональні інструменти, запозичені у Haskell та Standard ML.

Основна філософія мови узагальнена в документі Zen of Python (PEP 20), який включає наступні афоризми:

- Красиве – краще, ніж потворне.
- Явне – краще, ніж неявне.
- Просте – краще, ніж комплексне.
- Комплексне – краще, ніж складне.
- Читабельність має значення.

Замість того, щоб усі його функціональні можливості були вбудовані в його ядро, Python був розроблений таким чином, щоб бути дуже розширюваним. Ця компактна модульність зробила його особливо популярним як засіб додавання програмованих інтерфейсів до існуючих додатків. Бачення Ван Россумом малої основної мови з великою стандартною бібліотекою та легко розширюваним перекладачем випливало з його розчарувань у роботі з ABC, яка підтримувала протилежний підхід.

Python був створений як мова, яку легко читати. Його форматування, як правило, повністю безладне, і в ньому використовуються англійські ключові слова там, де інші мови використовують розділові знаки. На відміну від багатьох інших мов, він не використовує фігурні дужки для розмежування блоків, а крапки з комою після операторів необов’язкові. Натомість Python використовує відступи з пробілів для розмежування блоків. Збільшення відступу відбувається після певних тверджень; зменшення відступу означає кінець поточного блоку. Отже, візуальна структура програми точно відображає семантичну структуру програми. Цю функцію іноді називають правилом "поза стороною", яке мають деякі інші мови, але у більшості мов відступ не має семантичного значення.

Для ефективного процесу розробки програми розпізнавання образів я використав аспекти парадигми програмування функціонального програмування.

Функціональне програмування – це парадигма програмування, де програми будуються за допомогою застосування та складання функцій. Це декларативна парадигма програмування, в якій визначеннями функцій є дерева виразів, кожна з яких повертає значення, а не послідовність імперативних висловлювань, що змінюють стан програми.

У функціональному програмуванні функції розглядаються як громадяни першого класу, що означає, що вони можуть бути прив'язані до імен (включаючи локальні ідентифікатори), передані як аргументи та повернуті з інших функцій, як і будь-який інший тип даних. Це дозволяє писати програми в декларативному та складному стилі, де невеликі функції поєднуються модульним способом.

Функціональне програмування іноді трактується як синонім власне функціонального програмування, підмножини функціонального програмування, яка 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 може впливати будь-який змінний стан чи інші побічні ефекти. Це відрізняється від нечистих процедур, поширених у імперативному програмуванні, які можуть мати побічні ефекти (наприклад, зміна стану програми або отримання вхідних даних від користувача). Функціональне програмування сягає корінням в академічні кола, розвиваючись від лямбда-числення, формальної системи обчислень, заснованої лише на функціях. Функціональне програмування історично було менш популярним, ніж імперативне програмування, але сьогодні багато функціональних мов спостерігають використання у промисловості та освіті.

Найвизначніші характеристики функціонального програмування такі:

- Мови функціонального програмування розроблені на основі концепції математичних функцій, які використовують умовні вирази та рекурсію для виконання обчислень.
- Функціональне програмування підтримує функції вищого порядку та функції лінивого оцінювання.
- Функціональні мови програмування не підтримують елементи керування потоками, такі як оператори циклу та умовні оператори, такі як оператори If-Else та Switch. Вони безпосередньо використовують функції та функціональні виклики.
- Як і ООП, функціональні мови програмування підтримують такі популярні концепції, як абстракція, інкапсуляція, успадкування та поліморфізм.

Функціональне програмування пропонує наступні переваги:

- Функціональне програмування не підтримує стан, тому результатів побічних ефектів немає, і є можливість писати коди без помилок.
- Функціональні мови програмування не мають змінних станів, тому проблем зі зміною стану немає. Можна запрограмувати «функції» паралельно як «інструкції». Такі коди підтримують легке повторне використання та перевірку.

- Функціональні програми складаються з незалежних блоків, які можуть працювати одночасно. Як результат, такі програми є більш ефективними.
- Функціональне програмування підтримує вкладені функції.
- Функціональне програмування підтримує ледачі функціональні конструкції на кшталт ледачих списків, ледачих мапінгів тощо.

Як мінус, функціональне програмування вимагає великого простору пам'яті. Оскільки відсутнє поняття стану, потрібно створювати нові об’єкти щоразу, щоб виконувати дії. Функціональне програмування використовується в ситуаціях, коли доводиться виконувати безліч різних операцій над одним і тим же набором даних.

2.2. Дослідження та вибір для використання існуючих бібліотек та каталогів даних для розпізнавання образів

Велика стандартна бібліотека Python, яку зазвичай називають однією з найбільших її сильних сторін, пропонує інструменти, придатні для багатьох завдань. Однак у випадку нашого конкретного завдання потрібно використовувати кілька зовнішніх бібліотек і компонентів.

TensorFlow була найширшою бібліотекою, яка використовувалася під час розробки програми. Це відкрита бібліотека програмного забезпечення для машинного навчання, розроблена Google для вирішення проблем побудови та навчання нейронної мережі з метою автоматичного пошуку та класифікації зображень, з метою досягнення якості сприйняття, подібною до людської. Вона використовується як для досліджень, так і для розробки власних продуктів Google. Основний API для роботи з бібліотекою реалізований для Python (який використовувався, як описано вище). Існують також реалізації для R, C#, C ++, Haskell, Java, Go і Swift.

Це продовження закритого проекту DistBelief. Спочатку TensorFlow був розроблений командою Google Brain для внутрішнього використання в Google. У 2015 році система була переведена у вільний доступ з відкритою ліцензією Apache 2.0.

Важливою частиною цієї бібліотеки, яка була використана, є Keras API. Він був використаний для реалізації нашої основної моделі прогнозування. Keras – це

високорівневий API TensorFlow 2.0: відповідний, високопродуктивний інтерфейс для вирішення проблем машинного навчання, зосереджений на сучасному глибокому навчанні. Він надає основні абстракції та будівельні блоки для розробки та транспортування рішень для машинного навчання з високою швидкістю ітерацій. Keras – це також дуже гнучка структура, яка підходить для сучасних дослідницьких ідей. Keras дотримується принципу поступового розкриття складності: це полегшує початок роботи, але при цьому дає можливість обробляти довільно просунуті випадки використання, вимагаючи лише поступового навчання на кожному кроці.

Для відображення карт, що демонструють результати виконання програми, я використав бібліотеку Folium Python. Folium дозволяє легко візуалізувати дані, якими маніпулювали в Python, на інтерактивній картці листівки. Це дозволяє як прив'язувати дані до карти для візуалізації хороплетів, так і передавати розширені візуалізації вектора, растру чи HTML як маркери на карті.

Бібліотека має ряд вбудованих наборів плиток з OpenStreetMap, Mapbox та Stamen, а також підтримує власні набори плиток з ключами Mapbox або Cloudmade API. folium підтримує накладання Image, Video, GeoJSON і TopoJSON.

Для хостингу та виконання програми я використав Jupyter Notebook. Jupyter Notebook (раніше IPython Notebooks) - це веб-інтерактивне обчислювальне середовище для створення документів для ноутбуків Jupyter. Термін «блокнот» може в розмовній формі посилатися на багато різних сутностей, головним чином веб-додаток Jupyter, веб-сервер Jupyter Python або формат документа Jupyter залежно від контексту. Документ Jupyter Notebook - це документ JSON, який слідує за версійною схемою, що містить упорядкований список комірок вводу / виводу, який може містити код, текст (із використанням Markdown), математику, графіки та мультимедійні файли, як правило, закінчуючись розширенням ".ipynb" . Jupyter Notebook став популярним користувальницьким інтерфейсом для хмарних обчислень, і основні хмарні провайдери прийняли Jupyter Notebook або похідні інструменти як інтерфейсний інтерфейс для хмарних користувачів. Прикладом можуть бути ноутбуки SageMaker від Amazon, Google Colaboratory та ноутбук Azure від Microsoft.

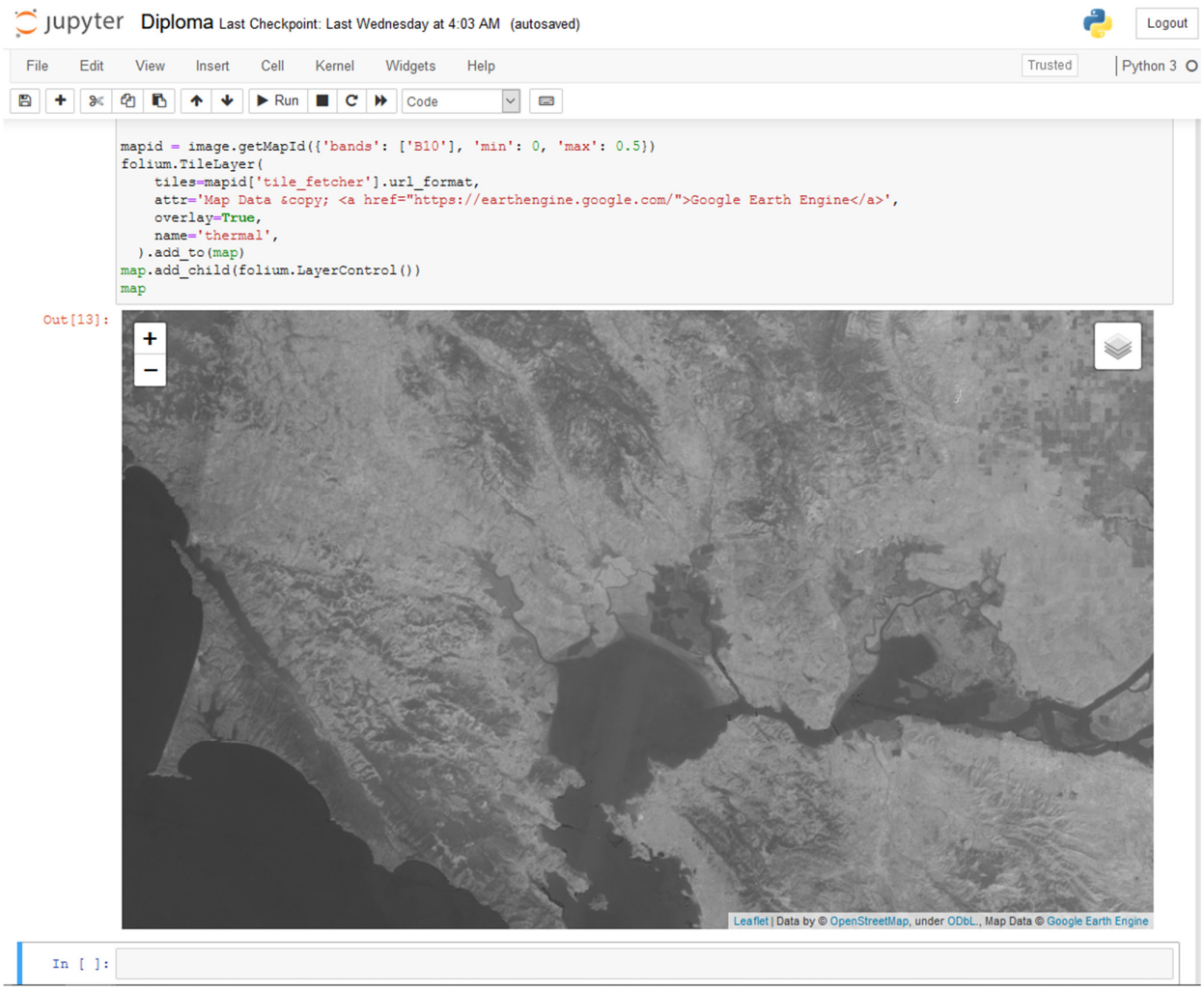

Рис. 2.1. Запущений екземпляр Jupyter Notebook

Будь-який процес розпізнавання образів потрібно навчити (і оцінити його навчання) на певному наборі даних. Для цього я використав API Google Earth Engine.

Google Earth Engine поєднує багатопетабайтний каталог супутникових зображень та геопросторових наборів даних із можливостями планетарного аналізу та робить його доступним для вчених, дослідників та розробників для виявлення змін, відображення тенденцій та кількісної оцінки відмінностей на поверхні Землі. Загальнодоступний архів даних включає понад тридцять років історичних зображень та наборів наукових даних, щоденно оновлюється та розширюється. Він містить понад двадцять петабайтів геопросторових даних, які миттєво доступні для аналізу.

API Earth Engine доступний на Python та JavaScript. Для ефективної роботи API Earth Engine потрібен обліковий запис Google Cloud Storage, який також було створено.

2.3. Дослідження існуючих методів побудови моделей нейронних мереж та вибір для використання

Під час підготовки до розробки програми було розглянуто кілька моделей нейронних мереж, зокрема VGGNet, ResNet та інші.

Архітектура мережі VGGNet була представлена Сімоняном та Ціссерманом у їх роботі 2014 року «Very Deep Convolutional Networks for Large Scale Image Recognition».

Основні використовувані різновиди мережі – VGG16 та VGG19. «16» та «19» означають кількість вагових шарів у мережі.

Ця мережа характеризується своєю простотою, використовуючи лише 3 × 3 згорткові шари, накладені один на одного на зростаючу глибину. Зменшення розміру обсягу здійснюється за рахунок максимального агрегування. Потім два повністю зв'язані шари, кожен з 4096 вузлами, супроводжуються класифікатором softmax.

Автори вважали навчання VGGNet складним (зокрема, щодо зближення на більш глибоких мережах), тому, щоб полегшити навчання, спочатку вони тренували менші версії VGG з меншими вагомими шарами.

Менші мережі сходились і потім використовувались як ініціалізації для більших, глибших мереж – цей процес називається попередньою підготовкою.

Маючи логічний сенс, попередня підготовка є дуже трудомістким, нудним завданням, що вимагає підготовки цілої мережі, перш ніж вона може послужити ініціалізацією для більш глибокої мережі. Це розкриває перший суттєвий недолік даної моделі, а саме надзвичайно довге навчання мережі. Другий недолік – великі вимоги до місця на жорсткому диску, а також до пропускної здатності каналу Інтернет.

ResNet – це штучна нейронна мережа, яка базується на конструкціях, відомих з пірамідальних клітин кори головного мозку. Типові моделі ResNet реалізовані з дво- або тришаровими пропусками, які містять нелінійності (ReLU) та нормалізацію між

ними. Однією з мотивацій пропускання шарів є уникнення проблеми зникнення градієнтів шляхом повторного використання активацій з попереднього шару, поки сусідній шар не вивчить свої ваги. Під час тренування ваги пристосовуються, щоб приглушити верхній шар (необхідне уточнення), і посилити попередньо пропущений шар. У найпростішому випадку адаптуються лише ваги для з'єднання сусіднього шару, без явних ваг для шару вище за течією. Це найкраще працює, коли перетинається один нелінійний шар, або коли всі проміжні шари є лінійними. Якщо ні, тоді для пропущеного з'єднання слід вивчити явну матрицю ваги (слід використовувати HighwayNet).

Пропуск ефективно спрощує мережу, використовуючи менше шарів на початкових етапах навчання. Це пришвидшує навчання за рахунок зменшення впливу зникаючих градієнтів, оскільки стає менше шарів для поширення. Потім мережа поступово відновлює пропущені шари, вивчаючи простір об’єктів. Ближче до кінця навчання, коли всі шари розширені, він залишається ближчим до колектора і, отже, швидше навчається. Нейронна мережа без залишкових частин досліджує більшу частину простору характеристик.

За рахунок швидшого навчання якість розпізнавання погіршується, що видно на Рис. 2.2.

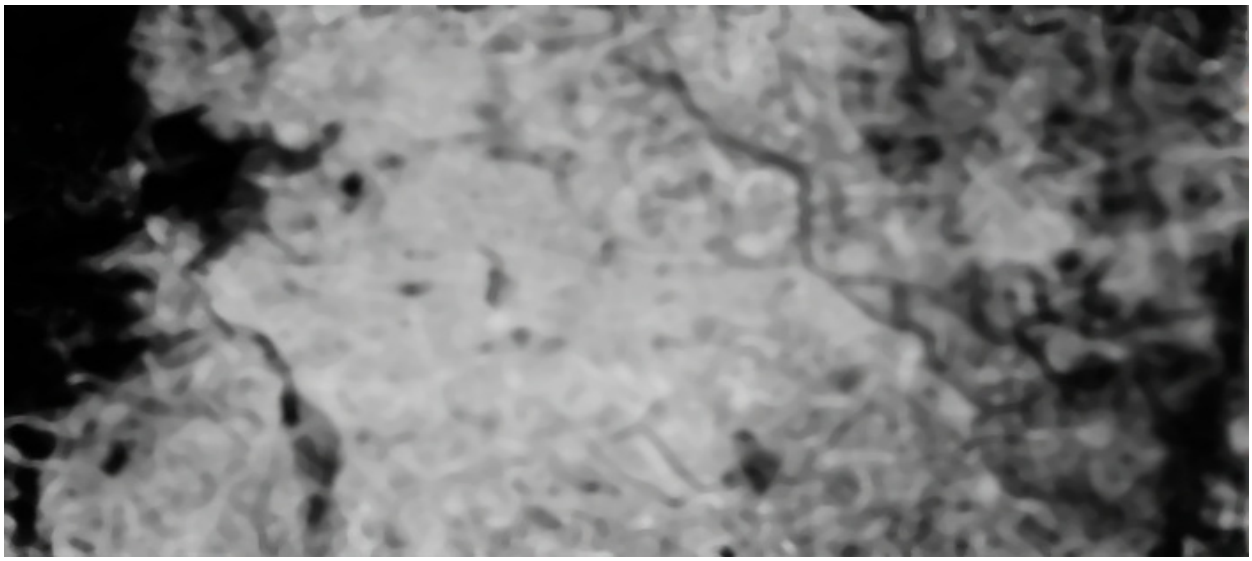

Рис. 2.2. Результат використання моделі ResNet для задачі розпізнавання непроникної області над Пекіном

Таким чином, у результаті досліджень для імплементації було вибрано U-Net, повнозгорткову нейронну мережу, що давала найкращі результати з-поміж розглянутих моделей нейронних мереж. Почнімо її пояснення з концепції простої згорткової мережі.

Згорткова нейронна мережа (ЗНМ, або convnet) – це клас глибоких нейронних мереж, який найчастіше застосовується для аналізу візуальних зображень. Вони також відомі як інваріантні до зсувів або інваріантні до простору штучні нейронні мережі, засновані на їхній архітектурі спільних ваг та характеристиках інваріантності переміщення. Вони мають застосування в розпізнаванні зображень та відео, системах рекомендацій, класифікації зображень, аналізі медичних зображень, обробці природної мови, інтерфейсах мозок-комп'ютер та фінансових часових рядах.

ЗНМ – це регуляризовані версії багатошарових перцептронів. Багатошарові перцептрони зазвичай означають повністю зв’язані мережі, тобто кожен нейрон в одному шарі пов’язаний з усіма нейронами наступного шару. «Повна зв’язаність» цих мереж робить їх схильними до перенавчання. Типові способи регуляризації включають додавання певної форми вимірювання ваги до функції втрат. ЗНМ застосовують інший підхід до регуляризації: вони використовують переваги ієрархічного образу в даних і збирають більш складні образи, використовуючи менші та простіші. Отже, за масштабами пов’язаності та складності ЗНМ знаходяться в нижній крайності.

Згорткові мережі були натхнені біологічними процесами, оскільки модель зв’язку між нейронами нагадує організацію зорової кори тварини. Окремі коркові нейрони реагують на подразники лише в обмеженій області поля зору, відомій як рецептивне поле. Сприйнятливі поля різних нейронів частково перекриваються так, що покривають все поле зору.

ЗНМ використовують порівняно мало попередньої обробки порівняно з іншими алгоритмами класифікації зображень. Це означає, що мережа засвоює фільтри, які в традиційних алгоритмах були розроблені вручну. Ця незалежність від попередніх знань та зусиль людини у розробці елементів є основною перевагою.

Назва «згорткова нейронна мережа» означає, що в мережі використовується математична операція, яка називається згорткою. Згорткові мережі – це спеціалізований тип нейронних мереж, які використовують згортку замість загального множення матриць принаймні в одному з їх шарів.

Згорнутий нейронна мережа складається з вхідного та вихідного шарів, а також безлічі прихованих шарів. Приховані шари ЗНМ, як правило, складаються з ряду згорткових шарів, які згортаються із множенням чи іншим крапковим продуктом. Функція активації, як правило, є шаром ReLU, за якою слідують додаткові звивки, такі як шари агрегування, повністю з'єднані шари та шари нормалізації, які називаються прихованими шарами, оскільки їх входи та виходи маскуються функцією активації та кінцевою згорткою.

Пояснімо концепцію шару ReLU, або ж випрямляча.

У контексті штучних нейронних мереж випрямляч є функцією активації, яка визначається як додатня частина його аргументу:

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

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

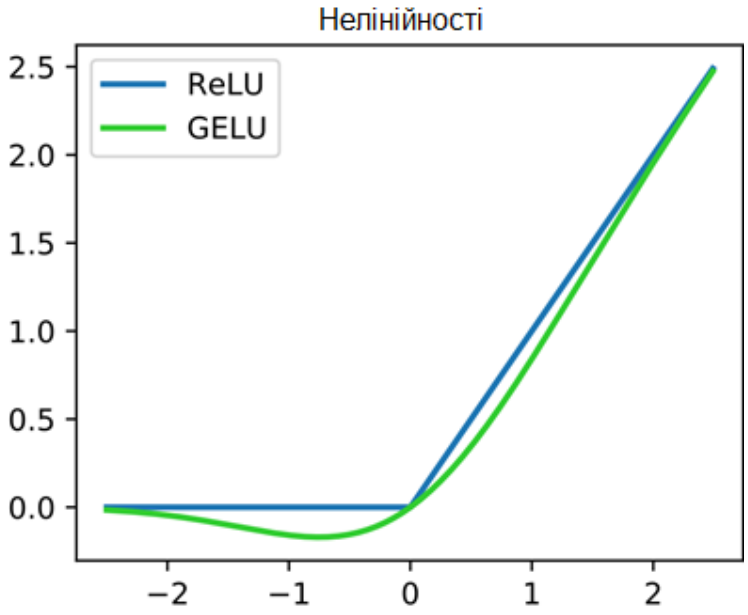

Рис. 2.3. Ділянка функції випрямляча ReLU (синя) поблизу x = 0

Ця функція активації була вперше представлена в динамічній мережі Hahnloser та співавторами у 2000 р. із сильними біологічними мотиваціями та математичними обґрунтуваннями. Це було вперше продемонстровано в 2011 році, щоб забезпечити кращу підготовку більш глибоких мереж порівняно з широко використовуваними функціями активації до 2011 року, наприклад, логістичною сигмоїдою (яка натхнена теорією ймовірностей) та її більш практичним аналогом, гіперболічним тангенсом. Станом на 2017 рік випрямляч є найпопулярнішою функцією активації глибоких нейронних мереж.

Блок, в якому використовується випрямляч, також називається випрямленою лінійною одиницею (ReLU).

Повернемося до концепції згорткових шарів у нейронній мережі. Вони повинні мати такі атрибути:

- Згорткові ядра, що визначаються шириною та висотою (гіперпараметри).
- Кількість вхідних та вихідних каналів (гіперпараметр).
- Глибина фільтра згортки (вхідні канали) повинна дорівнювати кількості каналів (глибині) карти вхідних характеристик.

Згорткові шари конвертують вхідні дані та передають результат наступному шару. Це схоже на реакцію нейрона в зоровій корі на певний подразник. Кожен згортковий нейрон обробляє дані лише для свого рецептивного поля. Незважаючи на те, що повністю підключені нейромережі прямого зв’язку можна використовувати для вивчення функцій, а також для класифікації даних, застосовувати цю архітектуру до зображень непрактично. Дуже велика кількість нейронів була б необхідна, навіть у неглибокій архітектурі, через дуже великі вхідні розміри, пов'язані з зображеннями, де кожен піксель є відповідною змінною. Наприклад, повністю зв’язаний шар для (маленького) зображення розміром 100×100 має 10000 ваг для кожного нейрона у другому шарі. Операція згортки пропонує вирішення цієї проблеми, оскільки зменшує кількість вільних параметрів, дозволяючи мережі бути глибшою з меншою кількістю параметрів. Наприклад, незалежно від розміру зображення, ділянки розміром 5×5, кожна з однаковими спільними вагами, вимагає лише 25 параметрів, що підлягають вивченню. Використовуючи регуляризовані ваги над меншою

кількістю параметрів, можна уникнути проблем з градієнтом зникнення та вибуху, що спостерігаються під час зворотного розповсюдження в традиційних нейронних мережах.

Зворотне розповсюдження, скорочення від «зворотне розповсюдження помилок», – це алгоритм контрольованого навчання штучним нейронним мережам з використанням градієнтного спуску. Враховуючи штучну нейронну мережу та функцію помилок, метод обчислює градієнт функції помилки щодо ваг нейронної мережі. Частина назви «зворотне» походить від того, що обчислення градієнта відбувається назад через мережу, при цьому градієнт кінцевого шару ваг обчислюється першим, а градієнт першого шару ваг розраховується останнім. Часткові обчислення градієнта з одного шару повторно використовуються при обчисленні градієнта для попереднього шару. Цей зворотний потік інформації про помилку дозволяє ефективно обчислювати градієнт на кожному шарі порівняно з наївним підходом обчислення градієнта кожного шару окремо.

Згорткові мережі стоять на чолі прогресу в розпізнаванні. ЗНМ не тільки вдосконалюються для класифікації цілих зображень, але й досягають прогресу у виконанні місцевих завдань зі структурованим результатом. Сюди входять передбачення частини та ключових точок та локальна відповідність. Наступним природним кроком на шляху від грубого до тонкого висновку є прогнозування на кожен піксель.

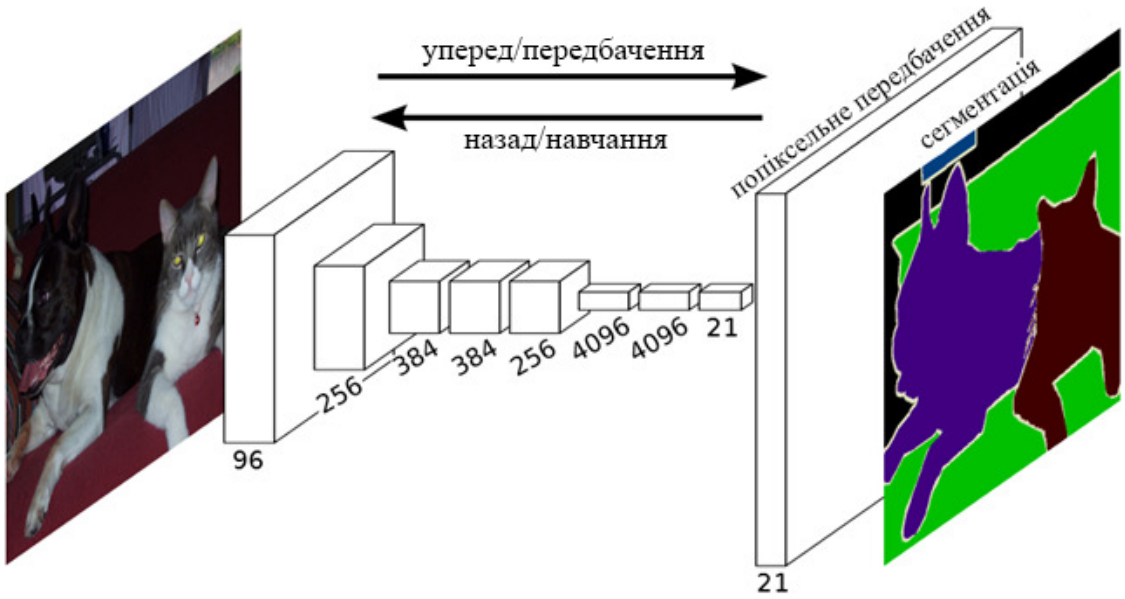

Рис. 2.4. Повністю згорткові мережі можуть ефективно навчитися робити щільні прогнози для завдань на піксель, таких як семантична сегментація

Лонг та ін. показують, що повністю згорткова мережа, навчена на семантичній сегментації, перевершує сучасний рівень без подальших механізмів. Повністю згорнуті версії існуючих мереж передбачають щільний вихід з входів довільного розміру. І навчання, і висновки виконуються на цілісних зображеннях за раз за допомогою щільних обчислень прямої передачі та зворотного розповсюдження. Шари підвищення вибірки в мережі дозволяють передбачати піксельне прогнозування та навчання в мережах з підвибіркою.

Повністю згорткові версії існуючих мереж передбачають щільний вихід з входів довільного розміру. І навчання, і висновки виконуються по цілісних зображеннях за раз за допомогою щільних обчислень прямої передачі та зворотного розповсюдження. Шари підвищення вибірки в мережі дозволяють передбачати піксельне прогнозування та навчання в мережах з під вибіркою. Цей метод ефективний, як асимптотично, так і абсолютно, і виключає необхідність ускладнень в інших роботах.

Кожен рівень даних у ЗНМ – це тривимірний масив розміром $h \times w \times d$, де h та w – просторові виміри, а d – характеристика, або розмір, каналу. Перший шар – це зображення з розміром пікселів $h \times w$ та d каналами кольорів. Розташування у вищих

шарах відповідають місцям на зображенні, до яких вони підключені шляхом, які називаються невід'ємними полями. ЗНМ побудовані на незмінності при переміщенні. Їх основні компоненти (функції згортки, агрегування та активації) діють на локальних областях введення і залежать лише від відносних просторових координат. Записуючи x_{ij} для вектора даних у точці (i, j) у певному шарі та y_{ij} для наступного шару, ці функції обчислюють виходи y_{ij} за допомогою

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

де k називається розміром ядра, s – коефіцієнт кроку або субдискретизації, а f_{ks} 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

$$f_{ks} \quad g_{k's'} = (f \quad g)_{k'+(k-1)s', ss'}.$$

У той час як загальна глибока мережа обчислює загальну нелінійну функцію, мережа лише з шарами цієї форми обчислює нелінійний фільтр, який називається глибоким фільтром або повністю згортковою мережею. Повністю згорткова мережа працює на вході будь-якого розміру і видає вихід відповідних просторових розмірів.

Функція реальних втрат, складена з повністю згорткової мережі, визначає завдання. Якщо функція втрат – це сума над просторовими розмірами кінцевого шару, $\ell(x; \theta) = \sum_{ij} \ell'(x_{ij}; \theta)$, її градієнт буде сумою градієнтів кожного з її просторових компонентів. Таким чином, стохастичний градієнтний спуск на ℓ, обчислений на цілих зображеннях, буде таким же, як і стохастичний градієнтний спуск на ℓ', беручи всі сприйнятливі поля остаточного шару як міні-пакет. Коли ці сприймаючі поля суттєво перекриваються, обчислення прямої передачі та зворотне розповсюдження набагато ефективніші за умов обчислення пошарово по всьому зображенню, замість використання окремих його частин.

Сама U-Net є згортковою нейронною мережею, розробленою для біомедичної сегментації зображень на Департаменті комп'ютерних наук Університету Фрайбурга, Німеччина. Мережа базується на повністю згортковій мережі, її архітектура була модифікована та розширена для роботи з меншою кількістю навчальних зображень та отримання більш точних сегментацій. Сегментація зображення розміром 512 × 512 займає менше секунди на сучасному графічному процесорі.

Архітектура U-Net випливає з так званої «повністю згорткової мережі», вперше запропонованої Лонгом, Шелхамером та Дарреллом.

Основна ідея полягає в тому, щоб доповнити звичайну контрактну мережу послідовними шарами, де операції об'єднання замінюються операторами вибору вибірки. Отже, ці шари збільшують роздільну здатність виходу. Більше того, послідовний згортковий рівень може навчитися збирати точний результат на основі цієї інформації.

Однією з важливих модифікацій U-Net є те, що в частині передискретизації є велика кількість функціональних каналів, які дозволяють мережі поширювати контекстну інформацію на рівні вищої роздільної здатності. Як наслідок, експансивний шлях більш-менш симетричний контрактуючій частині і дає u-подібну архітектуру. Мережа використовує лише дійсну частину кожної згортки без повністю зв’язаних шарів. Для прогнозування пікселів у прикордонній області зображення відсутній контекст екстраполюється за допомогою дзеркального відображення вхідного зображення. Ця стратегія викладання плитки є важливою для застосування мережі до великих зображень, оскільки в іншому випадку роздільна здатність буде обмежена пам'яттю GPU.

U-Net був створений Олафом Роннебергером, Філіпом Фішером, Томасом Броксом у 2015 році в роботі «U-Net: Convolutional Networks for Biomedical Image Segmentation». Це вдосконалення та розвиток роботи Евана Шелхамера, Джонатана Лонга та Тревора Даррелла «Fully convolutional networks for semantic segmentation» (2014).

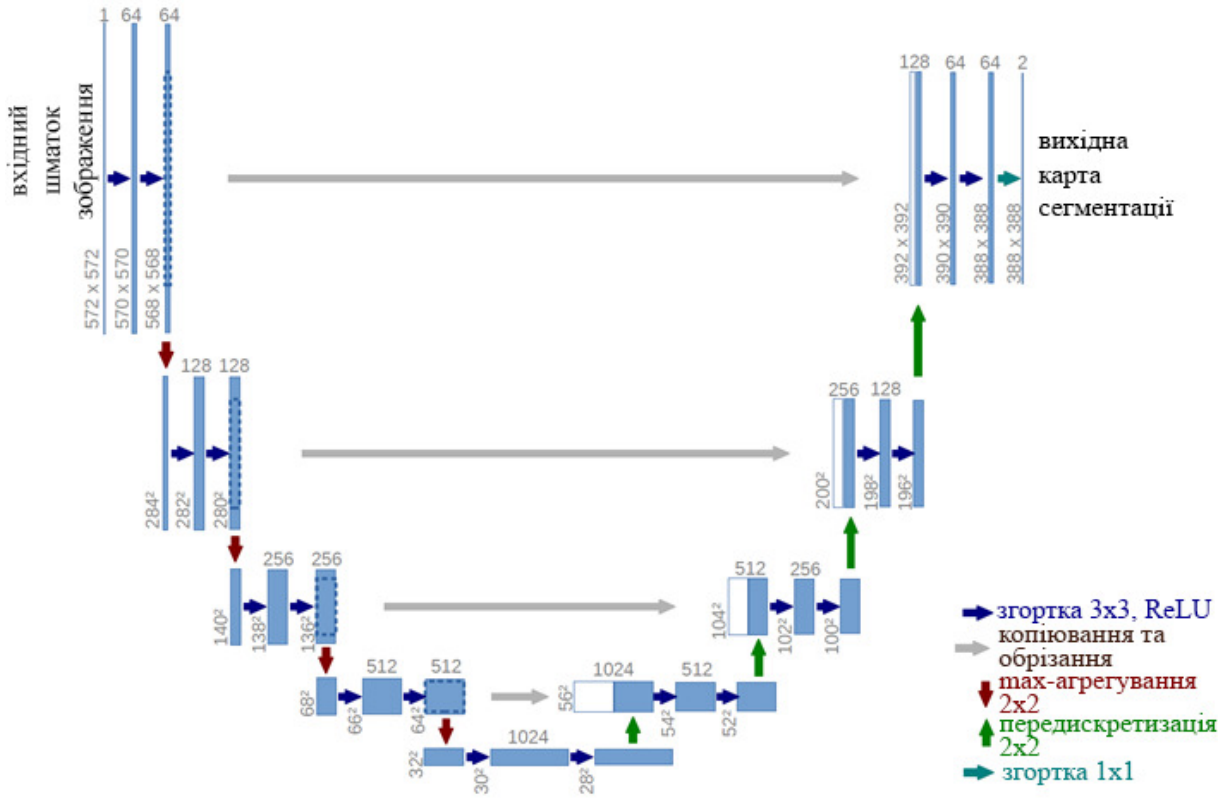

Рис. 2.5. Архітектура U-Net

Архітектура U-Net виглядає як буква «U», що виправдовує її назву. Ця архітектура складається з трьох розділів: стиснення, вузького місця та розширення. Секція стиснення виконана з безлічі контракційних блоків. Кожен блок, який приймає вхідні дані, застосовує два шари згортки 3 × 3 з подальшим max-агрегуванням 2 × 2. Кількість ядер або карт функцій після кожного блоку подвоюється, щоб архітектура могла ефективно вивчати складні структури. Найнижчий шар є посередником між стискаючим шаром і шаром розширення. Він використовує два шари ЗНМ 3 × 3, за якими слідує шар згортки 2 × 2.

Але основа цієї архітектури полягає в розділі розширення. Подібно до стискаючого шару, він також складається з декількох блоків розширення. Кожен блок передає вхідні дані на два шари ЗНМ 3 × 3, за якими слідує шар збільшення роздільної здатності 2 × 2. Також після кожного блоку кількість карт об'єктів, що використовуються згортковим шаром, отримує половину для збереження симетрії. Однак кожного разу вхідні дані також додаються до карт функцій відповідного шару

стиснення. Ця дія гарантує, що функції, які вивчаються під час стискання зображення, будуть використані для його реконструкції. Кількість блоків розширення така ж, як і кількість блоків скорочення. Після цього отримане відображення проходить через ще один шар ЗНМ 3 × 3 із числом карт об'єктів, рівним кількості бажаних сегментів.

U-Net використовує досить нову схему зважування втрат для кожного пікселя: на межі сегментованих об'єктів вага більша. Ця схема зважування втрат допомогла моделі U-Net сегментувати клітини на біомедичних зображеннях розривно, таким чином, що окремі клітини можна легко ідентифікувати на бінарній карті сегментації.

Перш за все, піксельний softmax застосовується до результуючого зображення, що супроводжується функцією перехресної ентропії. Кожен піксель тоді відноситься до одного з класів. Ідея полягає в тому, що навіть при сегментації кожен піксель повинен належати до певної категорії, і потрібно переконатися, що вони це роблять. Отже, проблема сегментації була перетворена на багатокласову класифікаційну та виконувалась дуже добре порівняно з традиційними функціями втрат.

Збільшення даних має важливе значення для навчання мережі бажаним властивостям незмінності та надійності, коли доступно лише кілька навчальних зразків. Особливо випадкові еластичні деформації навчальних зразків, є ключовою концепцією для підготовки сегментаційної мережі з дуже малою кількістю анотованих зображень. У U-Net плавні деформації генеруються за допомогою векторів випадкових переміщень на грубій сітці 3 × 3. Зміщення відбираються з гауссового розподілу зі стандартним відхиленням 10 пікселів. Потім зміщення на піксель обчислюється за допомогою бікубічної інтерполяції. Випадаючі шари в кінці шляху стиснення виконують подальше неявне збільшення даних.

Висновки до розділу 2

Таким чином, в результаті дослідження методів було обрано впровадження згорткової нейронної мережі U-Net для цілей розпізнавання образів. Існуючі методи показують певні незадовільні характеристики, а саме низьку точність результатів розпізнавання, повільне навчання, а також використання великого об’єму на жорсткому диску.

В результаті дослідження мов програмування було вибрано Python, що забезпечує розгалужену структуру для роботи з нейронними мережами для розпізнавання образів, а також код, що легко читається.

В результаті дослідження бібліотек було вибрано розширення TensorFlow, Folium та Jupyter Notebook, що найбільш відповідають специфіці програми, що розробляється.

Найбільш підходящим публічним архівом даних виявилося сховище Google Earth Engine, яке було використпно для розробки даної програми.

Етапи розробки програми розпізнавання образів та її алгоритм описані в розділі 3.

3. РОЗРОБКА АЛГОРИТМУ ТА ПРОГРАМИ РОЗПІЗНАВАННЯ ОБРАЗІВ ДЛЯ АЕРОФОТОЗЙОМКИ

Як було згадано в розділі 2, створення реальної програми розпізнавання шаблонів є складним процесом, що вимагає великих математичних знань та навичок програмування. Це завдання можна розділити на 3 менші: налаштування наборів даних про навчання та оцінку, впровадження та навчання моделі U-Net, прогнозування та вихід. Розглянемо загальний алгоритм програми, розробленої для розпізнавання образів за результатами аерофотозйомки, а також кожен із етапів її відпрацювання докладніше.

3.1. Алгоритм програми розпізнавання образів для аерофотозйомки

Метою програми розпізнавання образів, яку я розробляю, є прогнозування міської непроникності на основі аерофотозйомки (у моєму випадку супутникової) фотографії району. Непроникними поверхнями є в основному штучні споруди, такі як тротуари (дороги, тротуари, проїзди та паркінги, а також промислові зони, наприклад, аеропорти, порти та логістичні та розподільчі центри, всі з яких використовують значні мощені ділянки), які покриті водостійкими матеріали, зокрема асфальтом, бетоном, цеглою, каменем та черепицею. Грунти, ущільнені міською забудовою, також дуже непроникні. У нашому випадку ми хочемо передбачити безперервний вихід із стека безперервних входів. Вхідним сигналом буде складене зображення з супутника Landsat 8, а виходом буде непроникна область. Для цілей розробки програми я буду використовувати Національну базу даних земельного покриву США (National Land Cover Database, NLCD) як найкращий доступний ресурс, який також є частиною загальнодоступного набору даних Earth Engine.

Загальний процес перебігу програми розпізнавання образів можна описати

наступним чином:

- Імпорт зображень, які будуть використовуватися для введення даних.
- Підготовка відповіді (те, що необхідно передбачити; тут – непроникна поверхня).
- Складання вхідних зображень та зображень відповіді.
- Відбір проб стека в заздалегідь визначених місцях.
- Впровадження моделі прогнозування U-Net.
- Навчання моделі.
- Запуск передбачення на вказаному наборі даних.
- Відображення результатів.

Далі частини цього алгоритму будуть розглянуті більш детально.

3.2. Налаштування наборів даних навчання та оцінки

Програма починає роботу з імпорту вхідних зображень. Як було описано, ми використовуємо тирічні безхмарні композитні зображення 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())
```

Використовуються такі діапазони частот, що вловлюються супутником 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

Складені вхідні зображення для моделі візуалізуються за допомогою бібліотеки Folium Python. Це інтерактивна карта, яка виглядає наступним чином:

Рис. 3.1. Фрагмент візуалізації складеного зображення Landsat 8

Після цього ми аналогічно готуємо відповідь для тренування нашої моделі. Її візуалізацію можна побачити нижче:

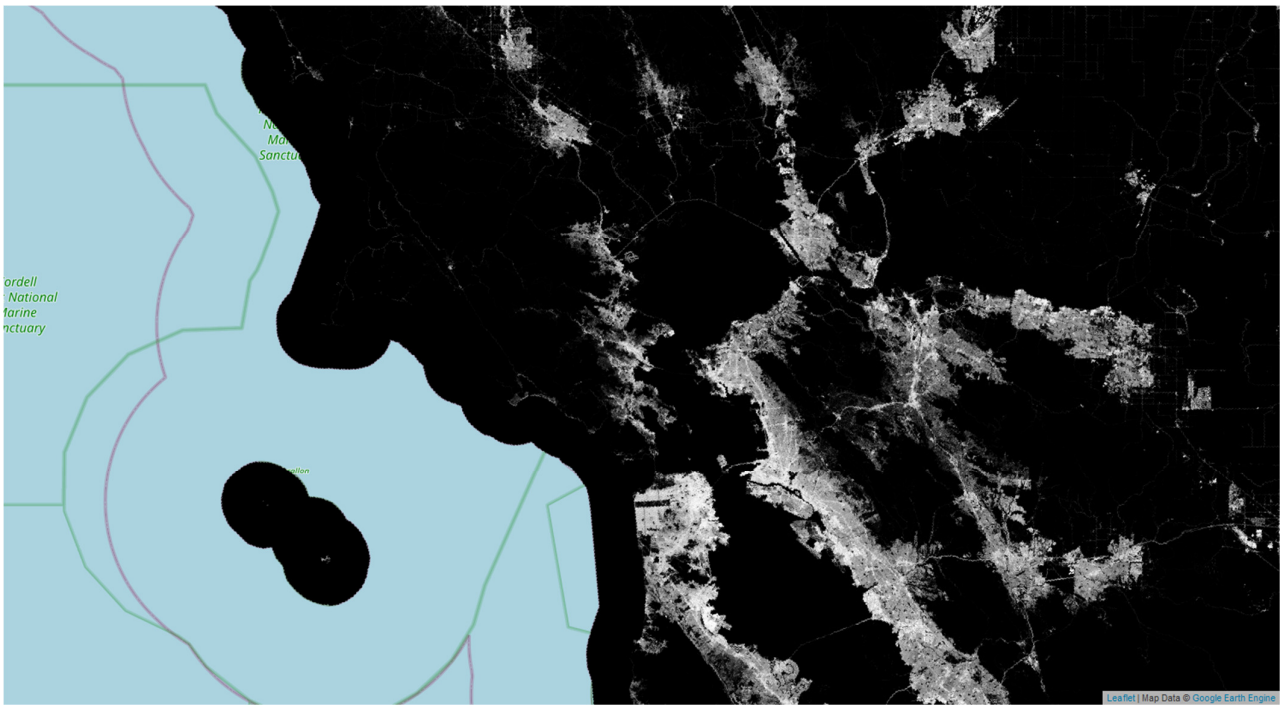

Рис. 3.2. Фрагмент візуалізації записів з NLCD

На цьому рисунку абсолютно непроникні ділянки забарвлені білим.

Потім ці 2D зображення (складене зображення з Landsat та непроникна поверхня NLCD) повинні бути складені, щоб створити єдине зображення, з якого можна взяти зразки. Для цього ми перетворюємо зображення в зображення масиву, в якому кожен піксель зберігає шматки розміром 256 x 256 пікселів для кожної смуги. Пізніше це буде експортовано до Google Cloud Storage.

Складене зображення має бути відібрано в деяких стратегічних місцях, які містять найбільш диверсифіковані зразки непроникних міських районів для нашої моделі, на яких можна навчитись. Зокрема, це вручну відібрані багатокутники поверхні, у яких відібрано зразки розміром 256 x 256. Прямокутники навчання та оцінки показані нижче. Червоним відмічено навчальні зони, синім – зони оцінювання.

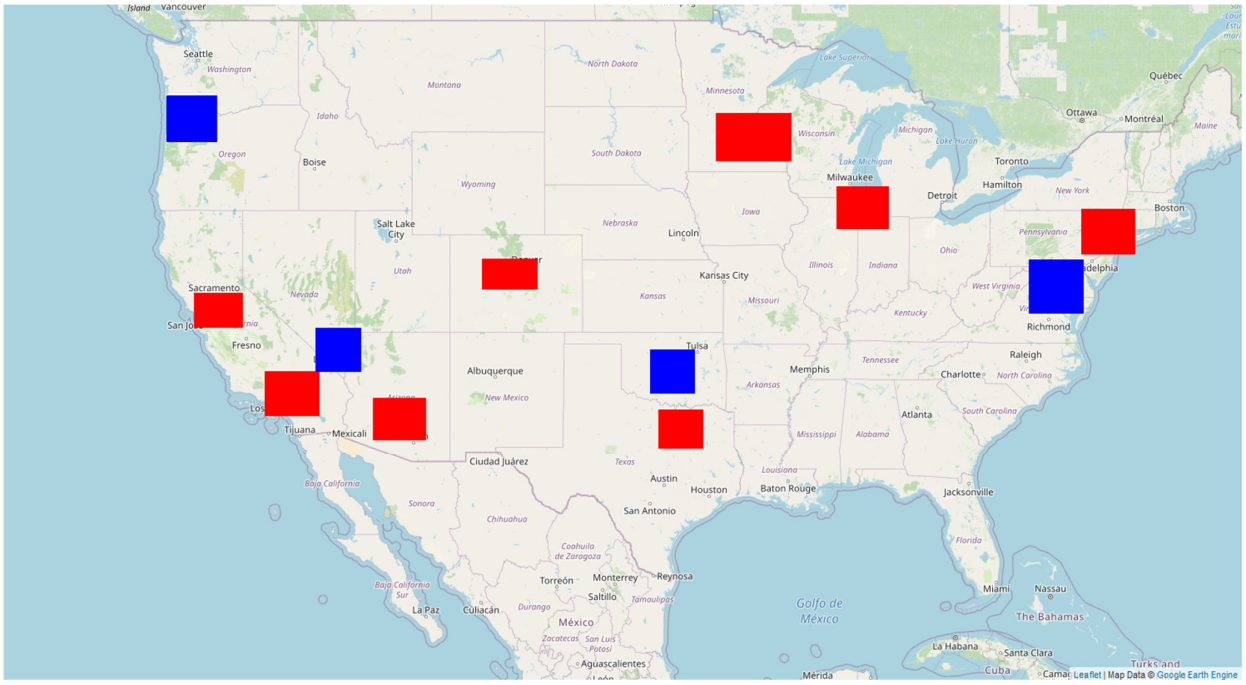

Рис. 3.3. Навчальні та оцінювальні прямокутники для моделі

Ключовим кроком є вибірка масиву зображень в точках, для отримання всіх пікселі в районі 256 x 256 у кожній точці. Варто зазначити, що для створення навчальних та тестових даних ми експортуємо один файл TFRecord, який містить шматки значень пікселів у кожному записі.

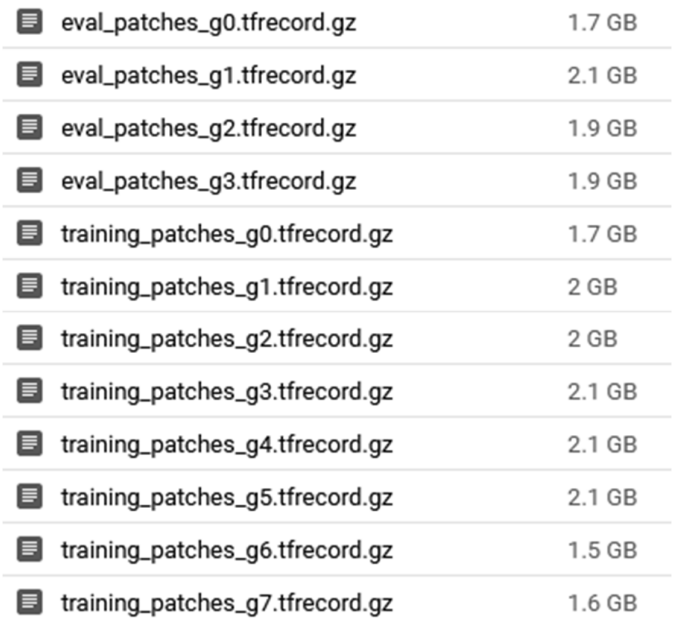

Рис. 3.4. Навчання та оцінка файлів TFrecord для моделі в Google Cloud Storage

Оскільки кожен запис потенційно містить багато даних (особливо з великими шматками зображень або безліччю вхідних смуг), для уникнення помилки «занадто велике значення обчислюваного значення» необхідно виконати певне дроблення вручну. Зокрема, було взято кілька менших зразків у межах кожної геометрії, та об'єднано результати, щоб отримати єдиний експорт.

```
# 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 із Earth Engine, завантажуються в TensorFlow *Dataset*. Ця процедура повторюється як для навчальних, так і для 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
```

Dataset – це механізм TensorFlow, який представляє потенційно великий набір елементів. Ці набори даних згодом будуть використані для навчання моделі.

У наведеному вище коді *gs://* – це внутрішня файлова система, яку використовує Google Cloud Storage. На більшість ресурсів посилаються безпосередньо, ставлячи *gs://* перед їх відносними шляхами. Завантаживши дані, ми повинні створити модель для роботи як з навчальними, так і з оціночними наборами даних.

3.3. Впровадження та навчання моделі U-Net

У даній програмі розпізнавання образів використовується Keras-реалізація моделі U-Net. Модель U-Net приймає шматки зображень розміром 256x256 пікселів як вхідні дані та виводить попіксельно імовірність класу, мітку або безперервний вихід. Модель можна реалізувати по суті немодифікованою, але необхідно використовувати середньоквадратичну втрату помилок на сигмоїдальному виведенні, оскільки розглядається проблема регресії, а не проблема класифікації. Обидві проблеми ставлять за мету побудову стислої моделі, яка може передбачити значення залежного атрибута на основі змінних атрибутів. Різниця між двома завданнями полягає в тому, що залежний атрибут є числовим для регресії та категоричним для класифікації. Оскільки фракція непроникної поверхні обмежена [0,1], з великою кількістю значень, близьких до нуля або одиниці, тут підходить насичувальна функція активації.

Модель реалізована з використанням чотирьох функцій: *conv_block* (реалізація блоку згортки), *encoder_block* (реалізація блоку кодування), *decoder_block* (реалізація блоку декодера) та *get_model* (загальне представлення моделі).

Кожен блок згортки застосовує два шари згортки 3 x 3 з функцією активації ReLU. Поміж ними відбувається процедура нормалізації пакета. Нормалізація пакетів

(також відома як пакетна норма) – це метод, який використовується для того, щоб зробити штучні нейронні мережі швидшими та стабільнішими завдяки нормалізації вхідного рівня шляхом повторного центрування та повторного масштабування. Він координує оновлення декількох шарів у моделі, масштабуючи вихідні дані шару, зокрема шляхом стандартизації активацій кожної вхідної змінної на міні-пакет, таких як активація вузла з попереднього шару. Під стандартизацією тут мається на увазі масштабування даних із середнім значенням рівним нулю та стандартним відхиленням рівним одиниці, наприклад, з використанням розподілення Гаусса. Стандартизація активацій попереднього рівня означає, що припущення, що наступний рівень робить щодо розповсюдження та розподілу входів під час оновлення ваги, не зміниться, принаймні не суттєво. Це має наслідком стабілізацію та пришвидшення тренувального процесу нейронних мереж. Нормалізація входів у шар впливає на підготовку моделі, різко зменшуючи кількість необхідних епох. Це також може мати регуляризуючий ефект, зменшуючи помилку узагальнення, подібно до використання регуляризації активації. Незважаючи на те, що зменшення «внутрішнього коваріантного зсуву» було мотивацією при розробці методу, існує певне припущення, що використання замість цього пакетної нормалізації є більш ефективноим, оскільки вона згладжує і, в свою чергу, спрощує функцію оптимізації, яка вирішується при навчанні 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 у згортковій нейронній мережі систематично застосовують вивчені фільтри до вхідних зображень, щоб створити карти функцій, які узагальнюють наявність цих ознак у вході.

Згорткові шари виявляються дуже ефективними, а укладання згорткових шарів у глибоких моделях дозволяє шарам, розташованим поруч із вхідними даними, вивчати особливості низького рівня (наприклад, лінії), а шарам глибше в моделі –

вивчати високі порядки або більш абстрактні особливості, такі як фігури або конкретні предмети.

Обмеженням виведення карти об’єктів згорткових шарів є те, що вони записують точне положення об’єктів на вході. Це означає, що невеликі рухи в положенні об’єкта на вхідному зображенні призведуть до іншої карти об’єктів. Це може статися при повторному обрізанні, обертанні, зсуві та інших незначних змінах вхідного зображення.

Загальноприйнятий підхід до вирішення цієї проблеми в процесі обробки сигналів називається зменшенням роздільної здатності (downsampling). Тут створюється версія вхідного сигналу з нижчою роздільною здатністю, яка все ще містить великі або важливі структурні елементи, без дрібних деталей, які можуть бути не такими корисними для завдання.

Зменшення роздільної здатності можна досягти за допомогою згорткових шарів, додавши новий шар, шар агрегування, зокрема, після того, як нелінійність (наприклад, ReLU, як у нашому випадку) застосовується до карт об'єктів, що виводяться згортковим шаром.

Додавання шару агрегування після згорткового шару є загальним шаблоном, який використовується для упорядкування шарів у згортковій нейронній мережі, який може повторюватися один або кілька разів у даній моделі.

Рівень агрегування діє на кожній карті об’єктів окремо, щоб створити новий набір з однаковою кількістю об’єднаних карт об’єктів.

Агрегування передбачає вибір операції агрегування, подібно до фільтру, який застосовуватиметься до об’єктивних карт. Розмір операції агрегування або фільтра менше, ніж розмір карти об'єктів; зокрема, майже завжди застосовується 2×2 з кроком у 2 пікселі.

Це означає, що шар агрегування завжди зменшить розмір кожної карти об’єктів у 2 рази, наприклад, кожен вимір зменшується вдвічі, зменшуючи кількість пікселів або значень на кожній карті об’єктів до однієї чверті розміру. Наприклад, шар агрегування, застосований до карти об’єктів 6×6 (36 пікселів), призведе до вихідної об’єднаної карти об’єктів 3×3 (9 пікселів).

Операція агрегування вказується дослідником, а не вивчається мережею. Двома загальними функціями, що використовуються в операції об'єднання, є:

- Середнє агрегування (Average pooling): обчислюється середнє значення для кожного шматка на карті об’єктів.
- Максимальне агрегування (Max pooling): обчислюється максимальне значення для кожного шматка на карті об’єктів.

Результатом використання шару агрегування та створення агрегованих карт об’єктів або таких зі зменшеною роздільною здатністю є узагальнена версія функцій, виявлених у вхідних даних. Вони корисні, оскільки невеликі зміни в розташуванні об’єкта на вході, виявлені згортковим шаром, призведуть до агрегованої карти об’єктів із об’єктом у тому самому місці. Ця можливість, додана агрегуванням, називається незмінністю моделі до локального переміщення.

У нашому випадку блок кодера використовує максимальне агрегування.

Призначення блоку кодера полягає в кодуванні вхідного зображення у представлення об’єктів на декількох 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 – це друга половина архітектури U-Net. Його мета полягає в семантичному проектуванні дискримінаційних ознак (менша роздільна здатність), засвоєних кодером, на простір пікселів (вища роздільна здатність) для отримання щільної класифікації. Декодер складається з передискретизації та стиснення з подальшим звичайним згортанням.

```
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
```

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 Keras-модель, викликавши на неї метод *.fit()*. Для цієї реалізації ми збираємося тренувати модель протягом 10 епох, тобто робочий алгоритм пройде 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, будучи реалізацією, побудованою на API Keras, працює лише із зображеннями у певному форматі TFRecord (TensorFlow Record). Формат TFRecord – це простий формат для зберігання послідовності двійкових записів. Він оптимізований для використання з TensorFlow різними способами. Для початку, це полегшує комбінування декількох наборів даних та безперешкодну інтеграцію з

функціями імпорту та попередньої обробки даних, що надаються бібліотекою. Особливо для наборів даних, які занадто великі для повного збереження в пам'яті, це є перевагою, оскільки лише ті дані, які потрібні на той момент (наприклад, пакет), завантажуються з диска і потім обробляються. Ще однією головною перевагою TFRecords є те, що можна зберігати дані послідовності – наприклад, часовий ряд або кодування слів – таким чином, що дозволяє дуже ефективно і (з точки зору кодування) зручно імпортувати дані цього типу.

3.4. Відпрацювання програми та виведення результатів

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

Процес передбачення перебігає наступним чином:

- Зображення, на яких можна робити прогнози із Earth Engine у форматі TFRecord, експортуються до відра (bucket) хмарного сховища.
- Навчена модель використовується для передбачення.
- Передбачення записуються у файл TFRecord у хмарному сховищі.
- Файл прогнозування у форматі TFRecord завантажується до Earth Engine.

Наступні функції обробляють 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 до TFRecord Earth Engine створив додатковий файл TFRecord під назвою «mixer». Це – простий файл JSON, який використовується для визначення просторового розташування шматків зображення (тобто георефернцій). Цей файл необхідний для завантаження передбачень, зроблених на зображеннях.

Залишається лише вказати область виводу, в якій слід робити прогноз, імена вихідних файлів, куди їх розміщувати, та форму виходів. Що стосується форми, модель навчається на шматках 256 x 256, але теоретично може працювати на будь-якому досить великому зображенні з рівними розмірами. Через артефакти на межах шматків зображень для моделі слід надавати дещо більші шматки для прогнозування, а тоді середній (центральний) шматок 256 x 256 слід висікати. Це контролюється за допомогою буфера ядра, половина розміру якого виходить за межі буфера ядра. Наприклад, зазначення ядра 128 x 128 додасть 64 пікселі на кожну сторону шматка зображення, щоб гарантувати, що пікселі на виході беруться з входів, повністю покритих ядром.

Отож, дані було експортовано, модель зробила передбачення і записала їх до файлу. Зображення, імпортоване в Earth Engine, дає змогу відобразити отриманий актив (asset) Earth Engine. Тут, наприклад, демонструється передбачення щодо непроникної області над Пекіном, Китай.

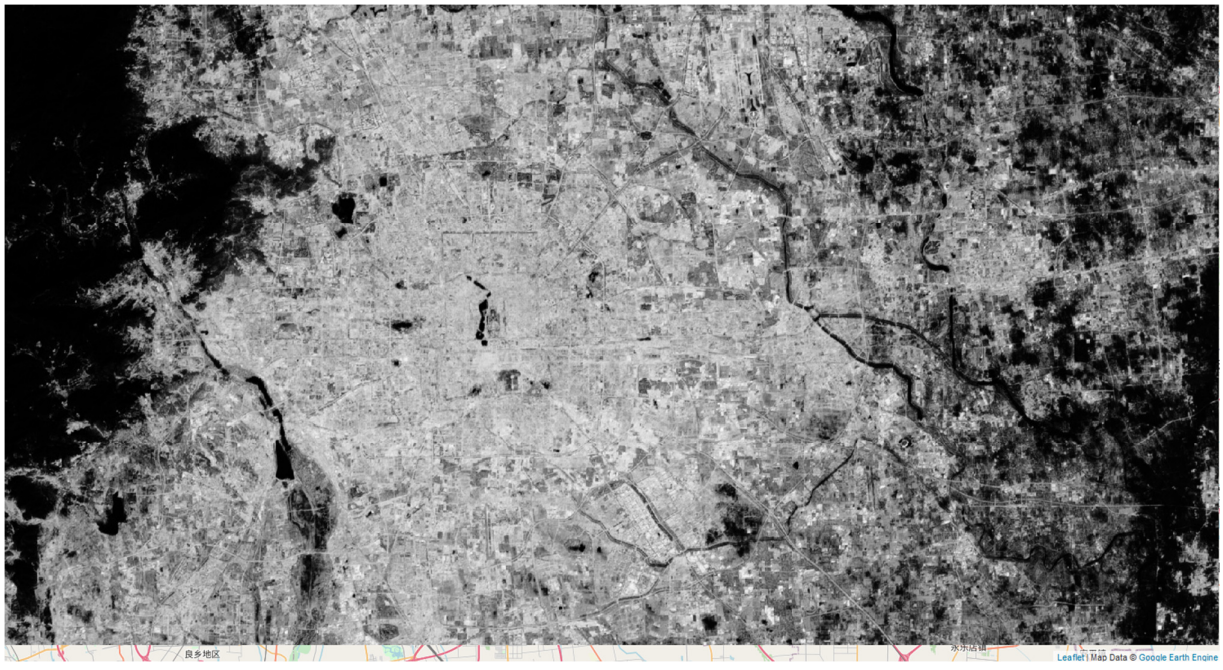

Рис. 3.5. Прогнозування непроникних районів над Пекіном, Китай

Колекція фотографій Landsat 8 над Пекіном була майже повністю визнана абсолютно непроникною областю.

Висновки до розділу 3

Розробка програми розпізнавання образів є непростим завданням. Більше того, особливості побудованої мною програми вимагають використання зовнішніх важкодоступних даних, таких як набір даних Google Earth Engine, що вимагало реєстрації, яку персонал Google перевіряв вручну перед наданням доступу до даних.

TensorFlow, і особливо його Keras API, виявився чудовим інструментом для створення та навчання нейронних мереж, які є основою програм розпізнавання образів. Наявність фреймворку на мові Python значно полегшило мені процес розробки програми, оскільки я маю певний досвід роботи з Python, порівняно з іншими мовами програмування з доступним програмуванням нейромереж, такими як Haskell, D або Lisp. Це та мої зусилля зробити код зрозумілим та гнучким зробили мою програму розпізнавання образів доступною для подальшого використання та вдосконалення; однак, деякі передумови, такі як доступ до Google Earth Engine і Google Cloud Storage, повинні бути виконані – інакше програма виявляється недоступною.

Створена програма розпізнавання образів має справу із завданням прогнозування непроникної області, яка буде детально розглянута в наступному розділі.

4. АЛГОРИТМ ВИКОНАННЯ ТА ПРИКЛАДИ ЗАСТОСУВАННЯ РОЗРОБЛЕНОЇ ПРОГРАМИ РОЗПІЗНАВАННЯ ОБРАЗІВ ДЛЯ АЕРОФОТОЗЙОМКИ

У розділі 3 було розглянуто процес розробки програми розпізнавання образів. Основною метою створеної програми є прогнозування даних про непроникність районів, сфотографованих з повітря або космосу, на основі навчальних зразків з Національних баз даних земельного покриву США.

4.1. Алгоритм виконання програми

Сценарій виконання програми не схожий на жодну з програм, які ми розробляли під час, скажімо, навчального курсу програмування. Замість виконуваних файлів код виконується оператором за оператором (або блоком за блоком) у Jupyter Notebook, або, як варіант, у будь-якому іншому похідному інструменті, скажімо, Amazon SageMaker Notebooks, Google Colaboratory або Microsoft Azure Notebook. У моєму випадку я виконую код локально на своєму комп'ютері за допомогою Jupyter Notebook.

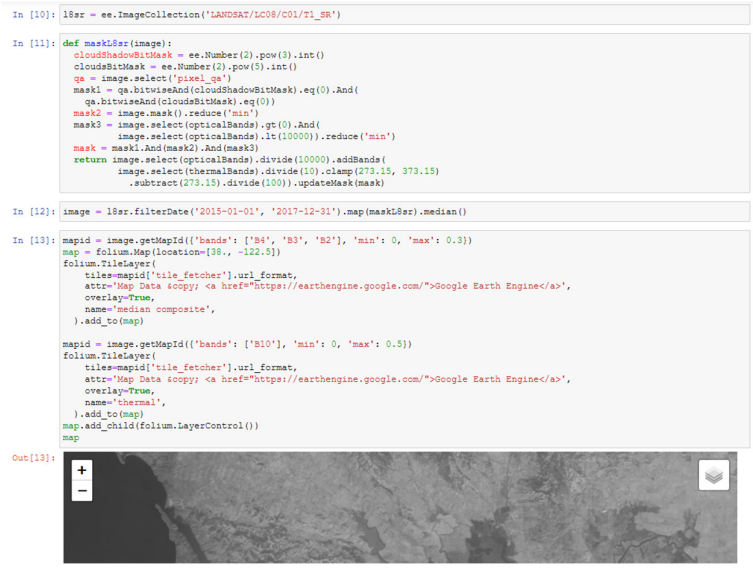

Рис. 4.2. Локальне виконання Jupyter Notebook

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()
```

Ці дії вимагають відвідування посилання для отримання коду авторизації, як показано нижче. Після цих дій Google зможе читати та записувати дані у відра хмарного сховища та сховища ресурсів Earth Engine.

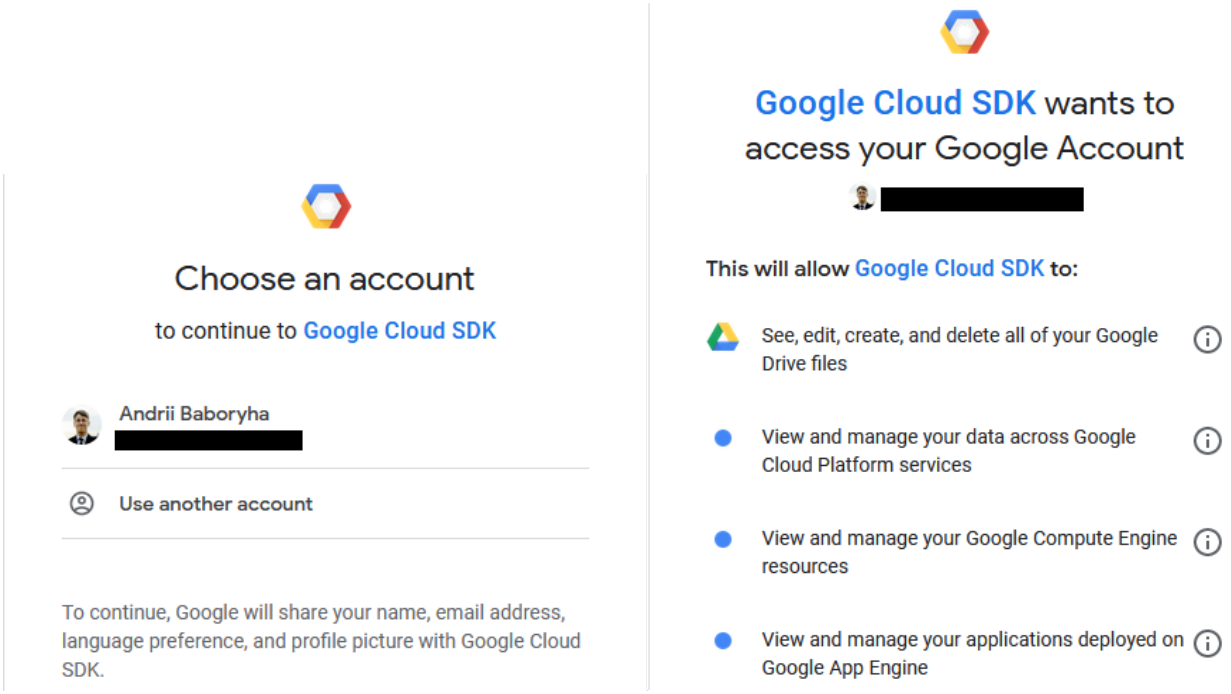

Рис. 4.2. Доступ до Google Cloud SDK

Після цього екрана Google повертає код підтвердження, який слід вставити до відповідного поля Jupyter Notebook. Подібним чином вимагається доступ до Earth Engine:

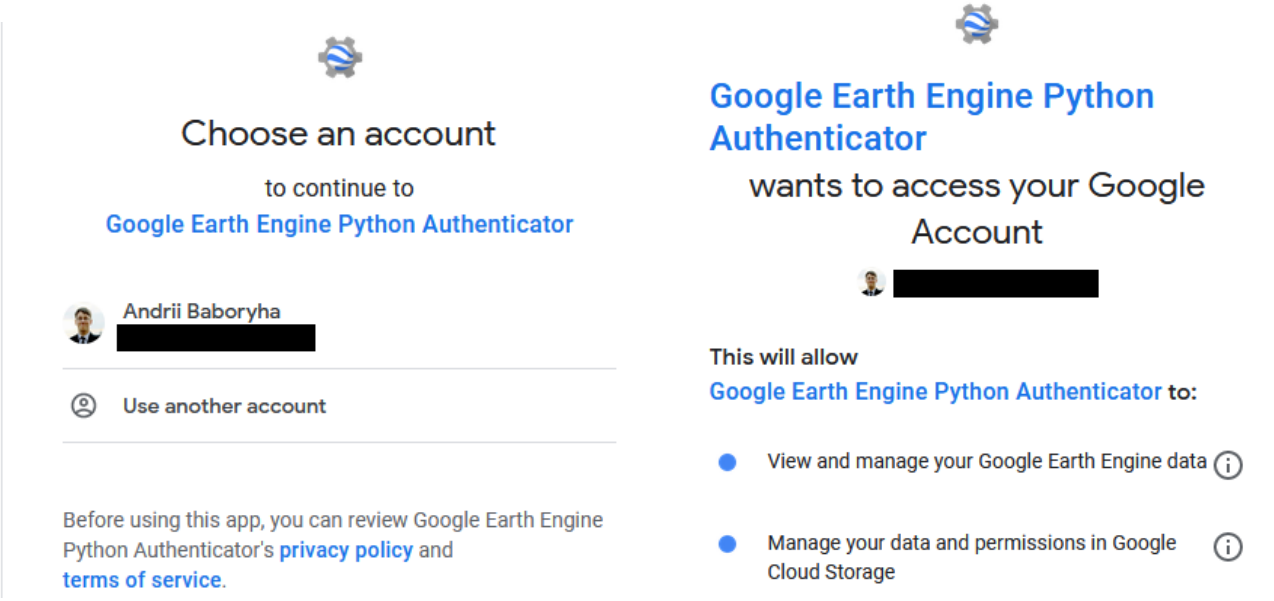

Рис. 4.3. Доступ до Google Earth Engine

Оскільки ці параметри жорстко встановлені в програмному коді, програма розпізнавання образів наразі використовує моє власне відро Cloud Storage та сховище ресурсів Earth Engine.

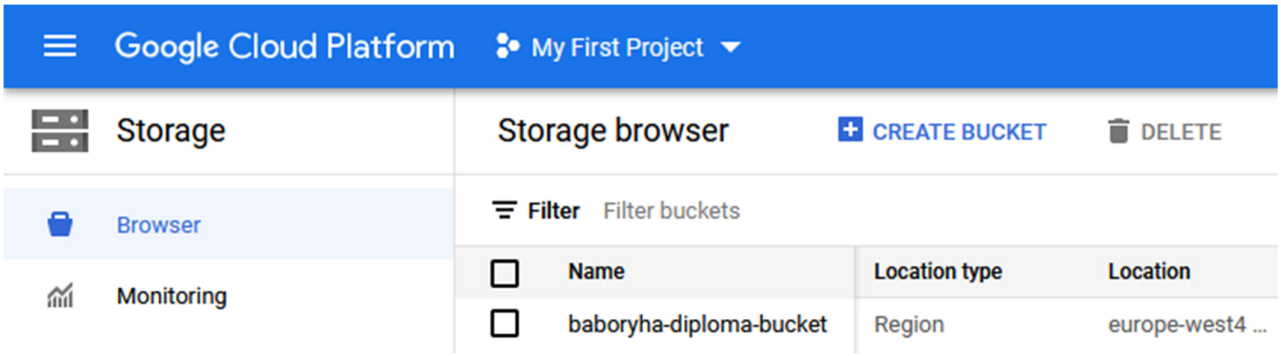

Рис. 4.3. Сховище Google Cloud Storage

Якщо ця програма виконується з використанням чужих можливостей зберігання даних, посилання на Cloud Storage та сховище ресурсів Earth Engine мають бути оновлені.

Будь-яка діяльність із зображеннями, очевидно, вимагає активного та стабільного з'єднання з Інтернетом, тому запуск цієї програми в автономному режимі неможливий.

Функції doExport та doPrediction, визначені в розділі 3.3, виконують завдання експорту до сховища Google. Ці завдання надсилаються до черги обробки серверів

Google. Для того, щоб перевірити стан завдань, використовується Google Earth Engine Code.

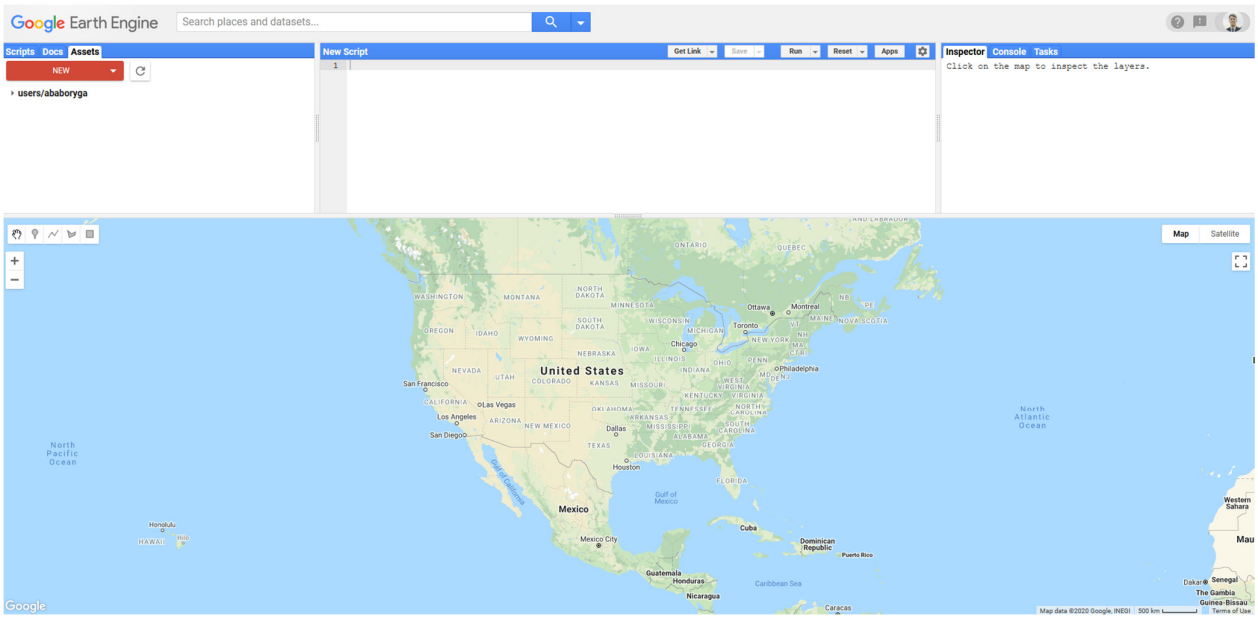

Рис. 4.4. Головне вікно Earth Engine Code

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

Рис. 4.5. Виконані завдання у Earth Engine Code

Завдання поділяються на дві групи: читання і запис. Цю програму можна запустити, використовуючи доступ лише для читання до певних сегментів Cloud Storage з даними, але в цьому випадку експорт зображень нових областей для додаткового навчання буде неможливим. Програма зможе відображати лише наявні результати.

Модель можна навчити кожного разу під час запуску програми, що може зайняти гігантську кількість часу, а саме десятки годин за умов недостатньої потужності обладнання. 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 А.

4.2. Приклади застосування розробленої програми

Місце, вибране для демонстрації програми, Пекін, описується прямокутником, тобто багатокутником, чотири вершини якого задані 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 та doPrediction. Якщо користувач хоче запустити програму для альтернативних місцеположень, потрібно замінити вершини цього багатокутника бажаними координатами для інших областей для запуску експорту даних та прогнозів на навченій моделі.

Широту та довготу даної точки можна визначити з вікна Code Engine, натиснувши на точку на інтерактивній карті. Як приклад, давайте виділимо точку на Майдані Незалежності. Результати наведені нижче, однак сам момент жодним чином не виділений.

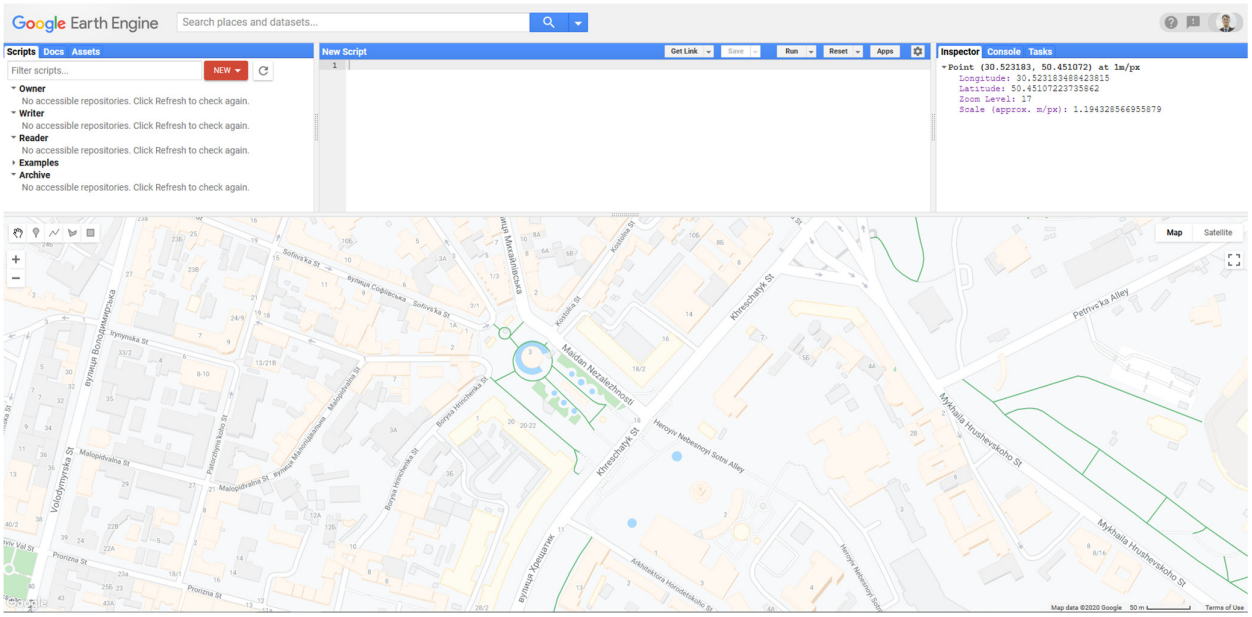

Рис. 4.6. Вибір точки у Earth Engine Code

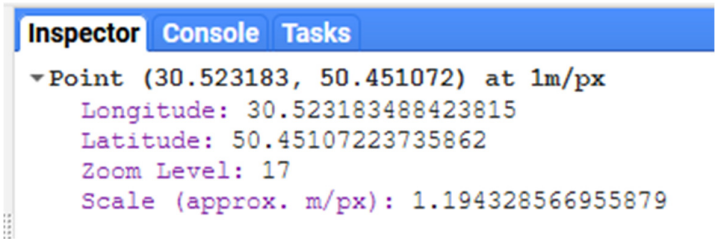

Рис. 4.7. Вікно Inspector

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

```
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.

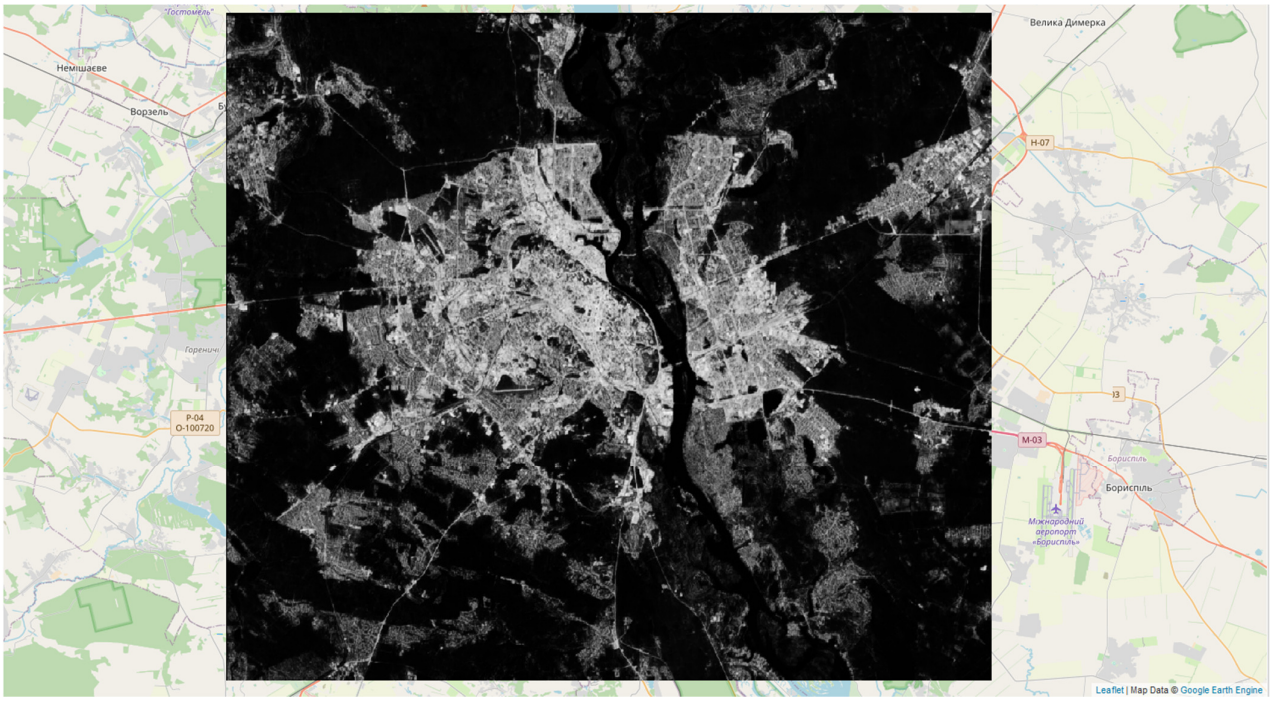

Рис. 4.8. Передбачення непроникної території над Києвом

Як ми бачимо, порівняно з Пекіном, непроникна територія Києва сильно постраждала від наявності зелених насаджень. Однак результат виконання програми не є ідеальним, і модель може бути доопрацьована далі, насамперед збільшенням кількості епох для навчання, скажімо, від 40 до 50. Однак, це значно збільшить час, необхідний для навчання моделі. Тим не менше, це може бути використано як суттєве вдосконалення даної програми розпізнавання образів.

Висновки до розділу 4

У розділі 4 було розглянуто алгоритм виконання розробленої програми розпізнавання образів для аерофотозйомок, а також приклади її застосування. Сценарій виконання даної програми не схожий на жодну з програм, які ми розробляли під час, скажімо, навчального курсу програмування – команди потрібно виконувати одна за одною у вікні Jupyter Notebook, або ж іншого середовища, обраного для хостингу. Також були показані приклади застосування програми, зокрема, передбачення непроникної території Києва. Для цього програмі необхідний

прямокутник, вершини якого будуть географічними координатами країв області, що повинна бути обробленою.

5. ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

5.1. Негативний вплив на людське здоров'я

Під час роботи з комп'ютером на людину впливають різні фактори: електромагнітні поля (діапазон радіочастот: ВЧ, УВЧ і НВЧ), інфрачервоне та іонізуюче випромінювання, шум і вібрація, статична електрика. Монітор комп'ютера видає кілька видів випромінювання: рентгенівське, ультрафіолетове, інфрачервоне, електромагнітне. Дослідження показують, що це не небезпечно для користувача ПК, оскільки інтенсивність такого випромінювання нижча за гранично допустимі межі.

Норми передбачають, що опромінюється все тіло, тоді як насправді піддається лише верхня частина тіла. Згадані норми встановлюються для кожного виду опромінення окремо, хоча всі поля діють одночасно і їх складний вплив ще не досліджено. Крім того, термінал відеодисплея порушує баланс між позитивно і негативно зарядженими іонами в повітрі. Електростатичне поле дисплея притягує негативні іони, порушуючи таким чином загальний баланс атмосфери. Це також шкодить здоров'ю. Протягом години роботи біля монітора відбувається майже повне зникнення негативних іонів. Ось чому необхідно, щоб свіже повітря надходило на робоче місце.

Оптичне випромінювання включає: ультрафіолетове (УФ), світлове та інфрачервоне (ІЧ). Як правило, УФ-випромінювання впливає на шкіру та очі людини. Аналіз досліджень робочих місць користувачів ПК показує, що в 86% вимірювань, тобто в більшості випадків, УФ-випромінювання не було виявлено. У тих же випадках, коли таке випромінювання було виявлено, його середній рівень був рівний 0.001 Вт/м^2.

Світлове випромінювання в основному впливає на очі і провокує їх втому та запалення райдужної оболонки. Однак ці симптоми швидко проходять і не викликають патологічних змін. ІЧ-випромінювання – довжина хвилі обмежена від

0,76 мм до 1 мм.

Для більшості біологічних матеріалів випромінювання в цьому діапазоні вважається непрозорим. Дослідження показали, що інтенсивність інфрачервоного випромінювання відеотерміналів нижча за показник, передбачений санітарними нормами.

Джерелом ЕРС є монітор. Тому, вибираючи робоче місце для комп'ютера, слід пам'ятати, що його задня та бічна стінки можуть бути джерелом значно більшої ЕРС, ніж екран.

5.2. Електронні відходи

Поняття «електронні відходи» описує викинуті електричні або електронні пристрої. Вживані електронні пристрої, призначені для реконструкції, повторного використання, перепродажу, утилізації через переробку матеріалів або власне утилізацію, також вважаються електронними відходами. Нерегламентована переробка електронних відходів у країнах, що розвиваються, може призвести до несприятливих наслідків для здоров'я людей та забруднення навколишнього середовища.

Електронні металобрухтні компоненти, наприклад, процесори, містять потенційно шкідливі матеріали, такі як свинець, кадмій, берилій або бромовані антипірени. Переробка та утилізація електронних відходів може становити значний ризик для здоров'я працівників та громад у розвинених країнах.

За даними MarketWatch, у 2018 році, для прикладу, споживачі замінювали свої мобільні телефони кожні 15 місяців. Організація економічного співробітництва та розвитку (ОЕСР) визначає електронні відходи як будь-які пристрої, що живляться від електричної енергії, що досягли кінця свого терміну експлуатації. Отже, мова йде не лише про мобільні телефони.

Електронні відходи можна класифікувати на основі їх складу та компонентів. Чорні та кольорові метали, скло, пластмаси, забруднюючі речовини та інші – це шість категорій матеріалів, що входять до складу електронних відходів. Залізо та сталь становлять основну частку у відходах матеріалів електричного та електронного

обладнання (ВЕЕО), а пластмаси є другими за величиною. Кольорові матеріали, включаючи метали, такі як мідь та алюміній, та дорогоцінні метали, такі як срібло, золото та платина, посідають третє місце за кількістю, а окрім цього, мають ще й значну комерційну цінність. До токсичних матеріалів відносяться свинець і кадмій в друкованих платах, оксид свинцю і кадмій в електронно-променевих трубках, ртуть у вимикачах і моніторах з плоским екраном, бромовані антипірени на друкованих платах, а також пластикові та ізольовані кабелі; коли вони перевищують порогові кількості, вони розглядаються як забруднюючі речовини і можуть ушкодити навколишнє середовище за умови неправильної утилізації.

Розглянемо деякі типи ВЕЕО, які існують відповідно до директиви Європейського Союзу (ЄС):

- Велика побутова техніка: холодильники, морозильні камери, пральні машини, сушарки для білизни, посудомийні машини, електричні кухонні плити та конфорки, мікрохвильові печі, електричні вентилятори та кондиціонери.
- Мала побутова техніка: пилососи, тостери, подрібнювачі, кавові машини, пристосування для стрижки та сушіння, чищення зубів та гоління.
- Інформаційні технології (ІТ) та телекомунікаційне обладнання: мейнфрейми, мінікомп’ютери, персональні комп’ютери, ноутбуки, ноутбуки, принтери, телефони та стільникові телефони.
- Споживче обладнання: радіо, телевізори, відеокамери, відеомагнітофони, стереомагнітофони, підсилювачі звуку та музичні інструменти.
- Освітлювальне обладнання: прямі та компактні люмінесцентні лампи та високоінтенсивні розрядні лампи.
- Електричні та електронні інструменти: свердла, пилки, швейні машини, паяльники, обладнання для токарної обробки, фрезерування, шліфування, свердління, виготовлення отворів, складання, згинання або подібної обробки деревини та металу.

- Іграшки, обладнання для відпочинку та спортивні товари: електрички або набори гоночних автомобілів, відеоігри та спортивне спорядження з електричними елементами.
- Медичні вироби: обладнання для радіотерапії, кардіологія, діаліз, легеневі вентилятори, ядерні ліки та аналізатори.
- Прилади моніторингу та управління: детектори диму, регулятори нагріву та термостати.
- Автоматичні дозатори: для гарячих напоїв, гарячих або холодних пляшок, твердих продуктів, грошей та всіх приладів, які автоматично доставляють різні продукти.

Згідно з доповіддю ООН, у 2018 році у світі було утворено 48,5 мільйонів тонн електронних відходів. Ця цифра підкреслює зростаючу важливість переробки, що також викликає деякі тривожні статистичні дані: лише 20% цих відходів переробляються. Якщо ми будемо продовжувати так, за оцінками ООН, людство може досягти 120 мільйонів тонн електронного брухту до 2050 року.

Обсяг вироблених електронних відходів у всьому світі та погане управління переробкою представляють небезпеку для навколишнього середовища. Серед найпоширеніших речовин, що містяться у цих викинутих предметах, є кадмій, свинець, оксид свинцю, сурма, нікель та ртуть. Ці токсичні елементи забруднюють річки, озера та моря та викидають в атмосферу гази, що порушує екосистеми. Тож повернення до моделі виробництва та споживання, яка зменшує кількість електронних відходів, більше не можна відкладати.

Складний склад та неправильне поводження з електронними відходами негативно впливають на здоров’я людей. Зростаючий обсяг епідеміологічних та клінічних доказів призвів до посилення занепокоєння щодо потенційної загрози електронних відходів для здоров'я людей, особливо в таких країнах, що розвиваються, таких як Індія та Китай. Примітивні методи, які використовуються нерегульованими операторами на задвірках (наприклад, неформальний сектор) для рекультивації, переробки та переробки електронних відходів, піддають працівників ряду токсичних речовин. Використовуються такі процеси, як демонтаж компонентів, мокра хімічна

обробка та спалювання, що призводить до безпосереднього впливу та вдихання шкідливих хімічних речовин. Засоби безпеки, такі як рукавички, маски для обличчя та вентиляційні вентилятори, практично невідомі, і працівники часто мало уявляють, з чим вони працюють.

Наприклад, що стосується небезпеки для здоров’я, відкрите спалення друкованих монтажних плат збільшує концентрацію діоксинів у прилеглих районах. Ці токсини викликають підвищений ризик розвитку раку, якщо їх вдихають працівники та місцеві жителі. Отруйні метали та отрута також можуть потрапляти в кров під час ручного вилучення та збору незначної кількості дорогоцінних металів, а працівники постійно зазнають дії отруйних хімічних речовин та випарів висококонцентрованих кислот. Відновлення перепродажу міді при спалюванні ізольованих проводів викликає неврологічні розлади, а гострий вплив кадмію, який міститься в напівпровідниках та мікросхемах резисторах, може пошкодити нирки та печінку та призвести до втрати кісткової маси. Тривалий вплив свинцю на друкованих платах та екранах комп’ютерів та телевізорів може пошкодити центральну та периферичну нервову систему та нирки, а діти більш сприйнятливі до цих шкідливих впливів.

Діти особливо вразливі до ризиків для здоров'я, які можуть виникнути внаслідок впливу електронних відходів, і тому потребують більш конкретного захисту. Оскільки вони все ще зростають, споживання дітьми повітря, води та їжі пропорційно їх вазі значно збільшується порівняно з дорослими, - а разом із цим і ризик небезпечного поглинання хімічних речовин. Крім того, функціональні системи їхніх органів, такі як центральна нервова, імунна, репродуктивна та травна системи, все ще розвиваються, і вплив токсичних речовин, перешкоджаючи подальшому розвитку, може завдати безповоротної шкоди. Багато дітей стикаються з хімічними речовинами, що походять від електронних відходів, у своєму повсякденному житті через небезпечні дії з переробки, які часто проводяться вдома - або членами сім'ї, або самими дітьми. Крім того, діти можуть потрапляти на сміттєзвалища, розташовані поблизу їхніх будинків, шкіл та ігрових майданчиків.

Хоча електроніка є незамінною частиною повсякденного життя, її шкідливий вплив на навколишнє середовище не можна ігнорувати або недооцінювати. Інтерфейс між електричним та електронним обладнанням та навколишнім середовищем відбувається під час виробництва, переробки та утилізації цих продуктів. Викиди диму, газів та твердих частинок у повітря, скидання рідких відходів у воду та дренажні системи та утилізація небезпечних відходів сприяють погіршенню стану навколишнього середовища. На додаток до більш жорсткого регулювання переробки та утилізації електронних відходів, існує потреба у політиці, яка розширює відповідальність усіх зацікавлених сторін, особливо виробників, за межі місця продажу та до кінця терміну служби продукції.

Висновки до розділу 5

За останні роки проблеми охорони природи стали більш глобальними. Найбільша увага приділяється збереженню екологічної рівноваги планети, збереженню природної спадщини. Навіть такі на перший погляд нешкідливі операції з комп'ютером мають багато негативного впливу на наше довкілля.

Загальний принцип захисту навколишнього середовища від забруднення складається з розробки комплексу заходів, що обмежує або виключає надходження шкідливих речовин у біосферу. Оскільки неможливо зупинити всі процеси, пов'язані з господарською діяльністю людини на Землі та в атмосфері, неможливо повністю перейти на безвідходну технологію виробництва. Сьогодні захист навколишнього середовища відбувається шляхом обмеження викидів, маса яких не перевищуватиме гранично допустимих концентрацій і не спричинятиме змін у складі та структурі ґрунту, тропосфери та верхніх шарів атмосфери.

6. ОХОРОНА ПРАЦІ

6.1. Вступ

У даній дипломній роботі було розроблено програму з розпізнавання образів, яка потребуватиме принаймні одного працівника для її використання. Дана програма – це відносно складне програмне забезпечення, яке потребуватиме значних апаратних ресурсів (потужні комп'ютери, які споживають багато електроенергії, часу, тощо) для її виконання.

Встановлення необхідного обладнання вимагає високої точності та обережності, оскільки роботи проводяться з крихким електрообладнанням. Крім того, сама робота льотного тренажера є надзвичайно інтелектуальною та ресурсомісткою; це вимагає відповідних заходів запобігання ризику.

В якості робочого місця для інженера з обслуговування програми було взято звичайний офісний кабінет.

6.2. Аналіз робочих умов

Роботодавець, який наймає працівників, повинен забезпечити комфорт та безпеку їх робочих місць. Розмір одного робочого місця повинен бути не менше 6 квадратних метрів. За необхідності сусідні робочі місця для працівників комп'ютерів слід розділити перегородками висотою до 2 метрів. Визначаючи розмір кімнати та робочого простору для однієї людини, додатково слід враховувати шафи, сейфи, шафи чи інші меблі чи обладнання в кімнаті. Можна розмістити на столі працівника допоміжні пристрої (принтери, динаміки, сканери) та місця зберігання, якщо це не обмежує видимість екрану і не заважатиме працівникові. У разі надмірного шуму або вібрації технічного обладнання роботодавець повинен надати працівникам антивібраційні килимки. Стілець працівника повинен бути поворотним, легко регулюватися по висоті та забезпечувати належну підтримку та зручне

положення спини та хребта людини. Слід прибирати кімнату щодня, а робоче місце та монітор комп’ютера очищати від пилу.

В офісі забороняється: проводити ремонт і обслуговування комп'ютера на робочому місці працівника; самостійно відремонтувати або спробувати відремонтувати комп’ютер без залучення компетентних фахівців; зберігати непотрібні документи, деталі та предмети, які не потрібні для робочого місця; використовувати нечіткі монітори та монітори, які мають поломку екрану; для роботи з матричним принтером без антивібраційного покриття та зі знятим покриттям. Особи, які не пройшли автоматизований курс охорони праці, не мають права працювати.

Під час розробки дипломної роботи всі розрахунки проводились за допомогою персонального комп’ютера. При цьому існували небезпечні та шкідливі фактори, що належать до групи фізичних факторів.

При використанні комп’ютера слід враховувати такі фактори:

1) недостатнє штучне освітлення робочої зони;
2) електробезпека;
3) підвищений рівень шуму.

Раціональне освітлення повинно відповідати ряду вимог та умов. Воно повинно бути:

- достатнім, щоб дозволити очам без напруги розрізняти деталі, що розглядаються;
- стабільним, для цього напруга в електричній мережі не повинна коливатися більше 4% від норми;
- рівномірно розподіленим по робочих поверхнях, щоб очам не довелося переходити з дуже темного місця на світлі і навпаки;
- таким, що не спричиняє сліпучих ефектів на людське око, як від самого джерела світла, так і від відображаючих поверхонь у полі зору працівника.

Зменшення відбиття джерел світла досягається за рахунок використання ламп таким чином, щоб на робочих місцях, у проходах, проходах не було різких тіней.

Також лампи повинні бути безпечними, тобто не мати можливості спричинити вибух або пожежу у виробничих приміщеннях.

Згідно з ДБН В.2.5-28:2018 «Природне та штучне освітлення», освітленість на робочому місці, що вимагає середньої зорової роботи, становить 200 люкс. Відповідно до розрахунків у розділі 6.5, освітленість у зазначеному робочому просторі в 1,5 рази перевищує необхідну.

Комп'ютер є однофазним споживачем змінного струму напруги 220 В від заземленої нейтральної мережі. ПК відноситься до електроустановок до 1000 В закритої конструкції, всі його струмопровідні частини ізольовано. Відповідно до способу захисту людини від ураження електричним струмом, комп'ютер та периферійне обладнання повинні відповідати 1 класу захисту.

Джерелами шуму на робочому місці інженера-програміста є як сам персональний комп'ютер (вентилятори систем охолодження системного блоку та блоку живлення), так і периферійне обладнання (ударні принтери, кондиціонер). Джерелами високочастотного шуму є електронна частина персонального комп'ютера та периферійного обладнання.

6.3. Заходи з охорони праці

Існує три системи засобів та заходів для забезпечення електробезпеки:

- система технічних засобів і заходів;
- система електрозахисних засобів;
- система організаційно-технічних заходів та засобів.

Технічні засоби та заходи з електробезпеки реалізуються при проектуванні електроустановок, їх виготовленні та монтажі відповідно до діючих стандартів. За своїми функціями технічні засоби та заходи щодо забезпечення електробезпеки поділяються на дві групи:

- технічні заходи та засоби забезпечення електробезпеки в нормальному режимі роботи електроустановок;
- технічні заходи та засоби забезпечення електробезпеки в аварійних режимах роботи електроустановок.

Основні технічні засоби та заходи для забезпечення електробезпеки в нормальному режимі роботи електроустановок включають:

- ізоляція провідних частин;
- недоступність провідних частин;
- охоронні замки;
- інструменти орієнтації в електроустановках;
- виконання електроустановок, ізольованих від землі;
- захисне розділення електричних мереж;
- компенсація ємнісних струмів замикання на землю;
- вирівнювання потенціалів.

З метою підвищення рівня безпеки, залежно від мети, умов експлуатації та конструкції, більшість перерахованих технічних засобів та заходів застосовуються в електроустановках одночасно.

Компанія Silent Systems виробляє шість комплектів Hushkits, які складаються з трьох компонентів: джерела живлення з низьким рівнем шуму; процесорний вентилятор, який є абсолютно тихим, і звуконепроникний корпус для жорстких дисків. Конструкція джерел живлення забезпечує менший опір потоку повітря, що зменшує ефективність та зменшує шум вентилятора. Для зменшення тепла від кожуха на ньому встановлені нагрівальні пластини. Деякі компанії, як Acer, LG, Samsung та Electronics, використовують набори Hushkits у своїх комп'ютерах. Використовуючи набір Hushkits, можна добитися зниження рівня шуму, що створюється ПК, на 90%.

6.4. Протипожежні заходи

За вибухонебезпекою та пожежею приміщення належить до категорії D.

Щоправда, пожежі в оргтехніці також дуже небезпечні, оскільки вони пов'язані з великим матеріальним збитком. Пожежа може виникнути при взаємодії горючих речовин та джерел займання. Горючими речовинами є будівельні та оздоблювальні матеріали, пластикові корпуси, шнури тощо. Джерелами займання є електронні схеми комп'ютерів, принтерів, джерел живлення, де внаслідок різних порушень

відбувається перегрів елементів, утворюються електричні іскри та дуги, які можуть спричинити горіння легкозаймистих матеріалів.

Під час обслуговування, ремонтних та профілактичних робіт використовуються різні легкозаймисті рідини, прокладаються тимчасові електричні провідники та здійснюється пайка. Існує додаткова небезпека пожежі, яка вимагає відповідного протипожежного захисту. Вогнегасники, призначені для локалізації невеликих пожеж, включають вогнегасники, сухий пісок, азбестові ковдри. Приміщення, де встановлені комп'ютери і де немає необхідності влаштовувати автоматичні системи пожежогасіння, повинні бути обладнані переносними вогнегасниками з вуглекислим газом із розрахунку 2 штуки на кожні 20 $м^2$ у приміщенні. Акустична обшивка стін, стель приміщень повинна бути виконана з негорючих та важко горючих матеріалів.

Підлоги в приміщеннях для ПК повинні бути виготовлені з негорючих матеріалів (або важкозаймистих матеріалів із вогнестійкістю не менше 0,5 години). Для миття деталей слід використовувати негорючі миючі засоби. Миття знімних пристроїв легкозаймистими рідинами дозволяється лише в спеціальних приміщеннях, обладнаних припливно-витяжною вентиляцією. У разі необхідності дрібного ремонту або технічного обслуговування ПК, якщо неможливо використовувати негорючі миючі засоби, дозволяється мати в ємності не більше 0,5 л легкозаймистих рідин, які не розбиваються і щільно закриваються.

Для виявлення початкової стадії займання необхідно використовувати пристрої автоматичних систем пожежогасіння, де це вимагається Правилами пожежної безпеки.

6.5. Розрахунок рівня освітленості робочого місця

Для запобігання впливу слабкого або надмірного освітлення необхідно: збільшити або зменшити кількість ламп; збільшити або зменшити яскравість ламп. Розрахунок освітленості з використанням світлового потоку можна провести за формулою:

$$E_l = \frac{nF_l\eta}{SK}, \tag{6.1}$$

де E_l – середня освітленість, люкс;

n – кількість ламп у кімнаті;

F_l – світловий потік однієї лампи, люмен;

η – коефіцієнт використання світлового потоку;

S – площа робочого місця, м^2;

K – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації ($K = 1{,}5$).

Якщо відома норма освітленості, світловий потік однієї лампи обчислюється за формулою:

$$F_l = \frac{E_n SKz}{n\eta}, \tag{6.2}$$

де z – відношення середньої освітленості до мінімальної ($z = 1{,}1$). Коефіцієнт світлового потоку η залежить від розмірів кімнати та коефіцієнтів відбиття її стелі та стін. Цей коефіцієнт може бути обчислений за формулою:

$$\eta = \frac{ab}{h_p(a+b)}, \tag{6.3}$$

де a та b – ширина та довжина кімнати відповідно, м;

h_p – висота підвісу лампи, м.

Початкові дані:

Коефіцієнт відбиття стіни – $p_w = 50\%$ та стелі – $p_c = 70\%$.

У нашому випадку, довжина кімнати a дорівнює 5 метрам, а ширина b – 3 метрам. Висота підвісу лампи h_p – 3 метри. Використовуючи формулу (6.3), ми отримуємо η:

$$\eta = \frac{5 \cdot 3}{3 \cdot (5+3)} = 0{,}625.$$

Наступний крок – обчислення світлового потоку. $E_n = 300$ люкс, $S = 10\text{м}^2, K = 1{,}5, z = 1{,}1, n = 4$. Можна обчислити світловий потік з використанням (6.2):

$$F_l = \frac{300 \cdot 10 \cdot 1{,}5 \cdot 1{,}1}{4 \cdot 0{,}625} = 1980 \text{ лм.}$$

Тепер обчислюємо освітленість за допомогою (6.1):

$$E_l = \frac{4 \cdot 1980 \cdot 0{,}625}{10 \cdot 1{,}5} = 330 \text{ люкс.}$$

Згідно з отриманими результатами, 4 лампи, які даватимуть освітленість 330 люкс, повинні мати принаймні 100 Вт потужності.

Висновки до розділу 6

Введення нової професії пов'язане з багатьма аспектами. Охорона праці – одна з основних. Зрештою, організація робочого процесу повинна повністю відповідати вимогам законодавства про екологію, пожежну безпеку, пристрій робочого місця тощо. В Україні багато галузей промисловості страждають від порушення цих норм роботодавцями.

Заходи безпеки та ремонту регулюються національними та галузевими стандартами, нормами охорони праці, правилами технічного обслуговування, технологіями ремонту, інструкціями та інструкціями з охорони праці тощо.

ВИСНОВОК

Створення програми розпізнавання образів за результатами аерофотозйомок – це складний процес, в якому зазвичай бере участь велика команда розробників, що складається з фахівців з різних галузей та має різні спеціальні навички. Я проаналізував існуючі імплементації схожих програм та знайшов у них певні недоліки, зокрема, низьку якість розпізнавання, великий об'єм використаного місця на накопичувачах та повільне навчання мережі, яка займається власне розпізнаванням. Тому моїм завданням було розробити програму, яка не буде мати таких вад, або ж буде мінімізувати їх прояви. Водночас я використав усі знання та навички, набуті за роки навчання в НАУ. Особливо знання про програмування були для мене найбільш корисними.

Розробка програми розпізнавання образів складається з багатьох етапів. Я почав з теоретичної підготовки. На цьому етапі я глибоко познайомився з поняттям розпізнавання образів та почав шукати як моделі нейромереж для імплементації (якою виявилась архітектура U-Net), так і програмні засоби для їх реалізації. За результатами дослідження була обрана мова програмування Python з використанням додаткових бібліотек TensorFlow, Folium. У якості каталогу даних для навчання мережі та подальшого розпізнавання було обрано каталог Google Earth Engine.

Після цього мені потрібно було створити специфікацію – вимоги до моєї програми (які функції вона повинна мати, як слід її виконувати тощо). Згодом мені довелося розробити таку структуру проекту, яка дозволила б успішно реалізувати всі специфікації. На цьому етапі найважливішими були структура програмного коду та специфікація форматів даних. Тут я використав принципи, засвоєні під час вивчення дисциплін «Програмування» та «Моделювання», та ті, що вивчив на курсах програмування поза університетом.

Стадія прототипу для мене була досить важкою. На цьому етапі важливо не залишати помилок чи будь-яких інших проблем перед переходом до наступного етапу, оскільки їх виправлення буде набагато складнішим. Коли всі підсистеми та

компоненти були створені та протестовані, було виконано перехід до остаточного етапу розробки продукту.

У результаті я задоволений виконаною роботою. Мені вдалося розробити програму, яка навчає нейромережу для розпізнавання відносно швидко, не використовує величезні об’єми даних у локальному сховищі, а також показує високу якість розпізнавання. Цього вдалось досягти за допомогою використання хмарних сервісів, які повністю беруть на себе задачі зі зберігання та обробки інформації. Щоправда, доступ до мережі Інтернет є критичною необхідністю для функціонування розробленого застосунку.

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 (1990). 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», 2004. — 928 с.
4. Джордж Стокман, Линда Шапиро. [Computer Vision Компьютерное зрение]. — М. : Бином. Лаборатория знаний, 2006. — 752 с.
5. В. Н. Вапник, А. Я. Червоненкис. Теория распознавания образов М.: Наука, 1974. — 416 с.
6. Поспелов Д.А. Искусственный интеллект. Справочник. Книга 2. Модели и методы // М.: Радио и связь, 1990. — 304 с.
7. Файн В. С. Опознавание изображений, М. 1970.
8. Bishop, Christopher M. (2006). Pattern Recognition and Machine Learning. Springer.
9. Schuermann, Juergen (1996). Pattern Classification: A Unified View of Statistical and Neural Approaches. New York: Wiley.
10. Duda, Richard O.; Hart, Peter E.; Stork, David G. (2000). Pattern Classification (2nd ed.). Wiley-Interscience.
11. Kulikowski, Casimir A.; Weiss, Sholom M. (1991). 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/pycon/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
32. Encyclopædia Britannica, Inc., Electronic waste. [Online]. Available: https://www.britannica.com/technology/electronic-waste
33. World Health Organization, Electronic waste. [Online]. Available: https://www.who.int/ceh/risks/ewaste/en/
34. Конституція України. Закон України «Про охорону праці» № 2694-XII від 14.10.1992 р.
35. Державні санітарні норми і правила «Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу». Наказ МОЗ України №248 від 08.04.2014 р.
36. ДСН 3.3.6.037-99 «Санітарні норми виробничого шуму, ультразвуку та інфразвуку».

37. Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями від 14.02.2018 №207
38. ДСН 3.3.6.039-99 «Державні гігієнічні посилання на промислові загальні та місцеві вібрації».
39. Правила пожежної безпеки в Україні. Наказ МВС України №1417 від 30.12.2014 р.
40. ДСТУ Б В.1.1-36:2016 Визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою від 15.06.2016 № 158
41. Правила експлуатації та типові норми належності вогнегасників. Наказ МВС України №25 від 15.01.2018 р.
42. ДБН В.2.5-28:2018 Природне та штучне освітлення від 03.10.2018р. № 264
43. ДБН В.2.5-27-2006 «Захисні заходи електробезпеки в електроустановках будинків і споруд»

ДОДАТОК А

Код **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
```