Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра математики

РОЗПІЗНАВАННЯ ДОРІГ НА СУПУТНИКОВИХ ЗОБРАЖЕННЯХ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

Текстова частина до курсової роботи за спеціальністю “Інженерія програмного забезпечення” 121

Керівник курсової роботи кандидат фізико-математичних наук, доцент
Крюкова Галина Віталіївна

_____

(підпис)

“____”_____2021 р.

Виконав студент
Василенко Олександр Сергійович

“____”_____2021 р.

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра математики

ЗАТВЕРДЖУЮ
Керівник курсової роботи кандидат фізико-математичних наук, доцент
_____Крюкова Г.В. (підпис)
“____”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту 5-го курсу факультету інформатики
Василенко Олександру Сергійовичу
ТЕМА: Розпізнавання доріг на супутникових зображеннях з використанням нейронних мереж
Зміст ТЧ до курсової роботи:

Вступ
1. Сегментація зображень
2. Розробка програми для розпізнавання доріг

Висновки
Список літератури
Додатки

Дата видачі “___” _____ 2021 р. Керівник _____
(підпис)

Завдання отримав _____
(підпис)

Тема: Розпізнавання доріг на супутникових зображеннях з використанням нейронних мереж

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	листопад 2020 р.	
2.	Огляд літератури за темою роботи.	грудень 2020 р.	
3.	Розробка програми	лютий 2021 р.	
4.	Написання текстової частини курсової роботи	березень 2021 р.	
7.	Створення слайдів для доповіді та оформлення курсової роботи.	21.03.2021	
8.	Аналіз отриманих результатів керівником.	11.04.2021	
8.	Корегування роботи за результатами аналізу.	25.04.2021	
10.	Остаточне оформлення курсової роботи та слайдів.	11.05.2021	
11.	Захист курсової роботи.	17.05.2021	

Студент **Василенко Олександр Сергійович**

Керівник **Крюкова Галина Віталіївна**

“_____”_____ 2021 р.

Зміст

Анотація

Суть курсової роботи полягає у дослідженні результатів роботи програми для розпізнавання доріг на супутникових зображеннях з використанням методів сегментації зображень на основі нейронних мереж. Результатом роботи є три програми. Перша програма – утиліта, що здійснює первинну обробку вхідних супутникових фотографій, фрагментує, фільтрує їх та формує набори даних для подальшого використання. Друга програма здійснює тренування нейронної мережі типу U-Net, використовуючи фрагменти супутникових знімків та відповідні бінарні маски. Третя програма відповідає за тестування розробленої моделі та відтворення результату. Продуктом розпізнавання є чорно-білі зображення, на яких білі пікселі вказують на ділянки фотографій, де мають бути дороги. Тренування нейронної мережі, що є основним компонентом програми, та тестування ефективності її роботи виконується з використанням колекції супутникових зображень великого розміру. Всі частини набору утиліт розроблено мовою Python.

Вступ

Завдяки розвитку супутникової фотографії вдалось досягнути високої якості супутникових зображень, що дозволяє зробити зображення всієї Земної поверхні високої роздільної здатності. Однак постають проблеми аналізу великих масивів даних, отриманих з супутників та виявлення корисної інформації. Однією з таких проблем є задача виявлення доріг на супутникових зображеннях. Наприклад, за статистикою, що надає уряд Індії, у 2010 році в середньому дороги будувалися зі швидкістю 12 кілометрів на день, а у 2019 це число зросло до 30 кілометрів на день. Завдяки такому бурхливому розвитку будівництва за останні п’ять років в Індії було прокладено більше мільйона кілометрів доріг. [3] Створення карт вручну – це дуже

кропітка праця, а програми, завдяки яким можна згенерувати основу цих карт, значно полегшать цей процес. Саме тому розробка подібних систем є актуальною.

Метою розробки є створення програми для аналізу супутникових фотографій та їх сегментації для виявлення доріг, а також аналіз результатів роботи програми. Для цього необхідно знайти масив даних з супутниковими зображеннями міста і набір відповідних бінарних карт, де будуть позначені дороги. Ці дані мають бути оброблені відповідним чином для їх подальшої обробки та тренування нейронної мережі. Розроблену систему необхідно протестувати для оцінки якості вихідних даних.

Робота складається з двох розділів. Перший розділ присвячено теорії сегментації зображень, принципам згорткових нейронних мереж і, зокрема, U-Net. У другому розділі наведено деталі розробленої системи та дослідження результатів реалізованого рішення.

1. СЕГМЕНТАЦІЯ ЗОБРАЖЕНЬ

1.1. Визначення

У першу чергу необхідно надати визначення основним поняттям, що є ключовими для розуміння теми. Комп’ютерний зір – це галузь наукового дослідження, що дозволяє комп’ютерним системам отримувати інформацію з цифрових зображень, відео та інших засобів передачі візуальної інформації, після чого здійснювати певні дії або давати рекомендації, спираючись на аналіз отриманих даних. Комп’ютерний зір працює аналогічно людському, однак зір людини має перевагу у тому, що людина розуміє контекст, що дозволяє їй розділяти зображення на окремі об’єкти. [2] Саме тому основною метою цієї галузі є створення програмних засобів, завдяки яким обчислювальні машини зможуть розпізнавати візуальну інформації для виявлення контексту. Комп’ютерний зір широко використовується у різноманітних сферах, серед яких безпека, автоматизація, землеробство та ін. Однією з них є транспортна навігація, що пов’язана з рядом проблем, що вимагають детального вивчення. До таких проблем відноситься розпізнавання доріг на супутникових зображеннях.

Проблема розпізнавання доріг на супутникових зображеннях високої роздільної здатності відноситься до класу задач сегментації зображень. З точки зору обчислювальної машини, сегментація – це процес розділення зображення, записаного у цифровому вигляді, на певну множину сегментів, де сегмент (суперпіксель) – це деяка множина пікселів. Метою сегментації є певне спрощення або зміна представлення зображення для полегшення подальшого аналізу. Основною задачею цього методу є пошук груп пікселів, кожен з яких певним чином характеризує єдиний смисловий об’єкт. Результатом сегментації є множина суперпікселів, або множина контурів об’єктів. Інакше кажучи, кожному пікселю призначається мітка, що визначає, до якого класу належить сегмент зображення.

Сегментація використовується у різних сферах для виявлення об’єктів та їх меж на зображеннях. Взагалі кажучи, для задачі сегментації не існує єдиного рішення, але є декілька універсальних методів та алгоритмів. Їх поєднують та модифікують для конкретної ситуації, використовуючи знання про предметну область, що дозволяють ефективно знаходити необхідні суперпікселі. [1] Наведемо конкретні приклади основних методів для визначення, які алгоритми найбільш прийнятні для вирішення поставленої задачі.

Одним із базових алгоритмів є визначення порогів. Цей метод є одним із найпростіших у сфері сегментації. Основою цього методу є розбиття вхідного зображення на 2 або більше часток, спираючись на певні заздалегідь визначені порогові значення. Розглядають два базових підходи:

a) Метод з глобальним порогом;

b) Метод з адаптивним порогом.

Всі інші методи є похідними від методу з глобальним порогом або від методу з адаптивним порогом. [4]

Рисунок 1.1 - Зображення з монетами для сегментації

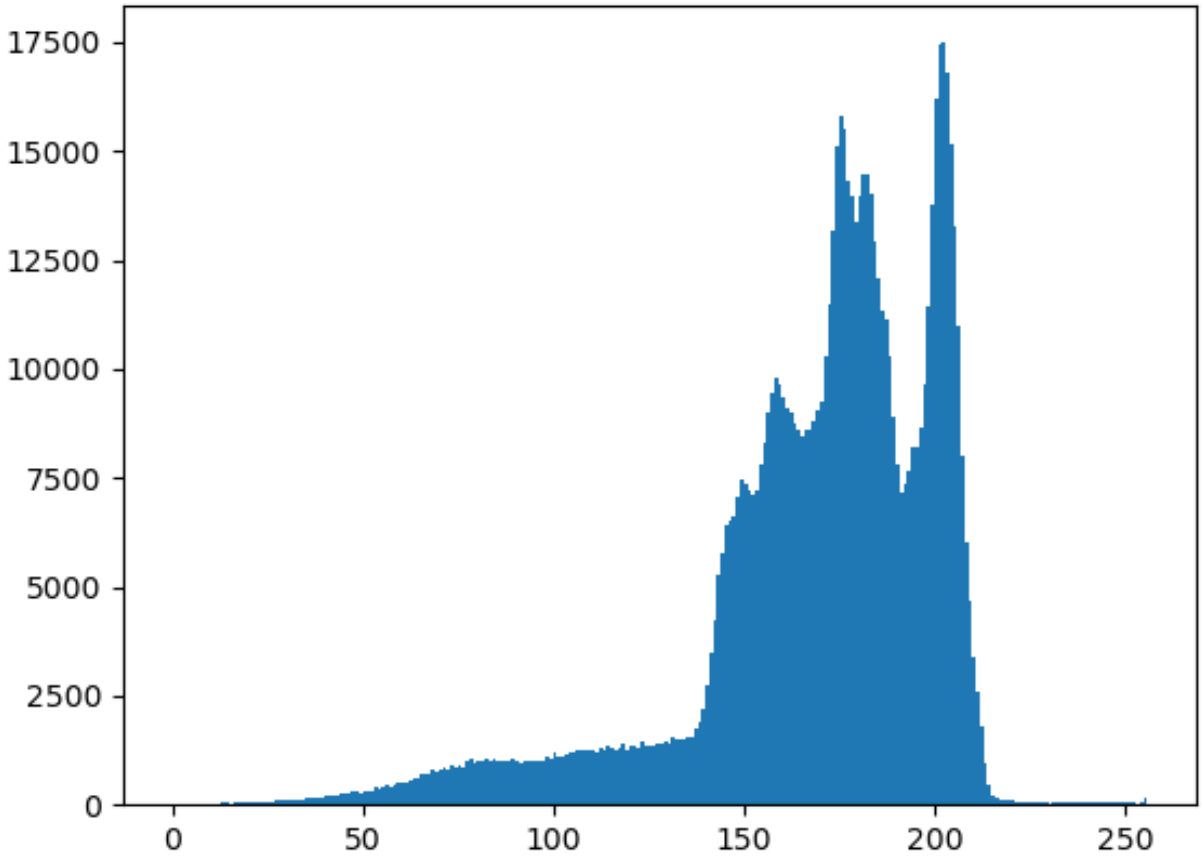

Рисунок 1.2 – Гістограма яскравості зображення

У більшості випадків варіації цього методу базуються на відсіканні певної кількості кольорів для отримання вихідного бінарного зображення з вхідного зображення у градаціях сірого. Тепер розглянемо цей метод більш детально. На графіку (рисунок 1.2) зображено гістограму, що показує залежність між кількістю пікселів та їх яскравістю (від 0 до 255) для зображення з кількома монетами на білому фоні (рисунок 1.1). Проаналізувавши цей графік, можна зробити висновок про те, що яскравість пікселів згрупована в околі кількох основних локальних максимумів. Для того, щоб виділити ці області, достатньо обрати деяке значення T і визначити всі пікселі, що задовільнять нерівності $f(x,y) \leq T$. У такому випадку вихідне бінарне зображення, отримане методом порогів, матиме вигляд функції $g(x,y) = 1, if\ f(x,y) > T; 0, if\ f(x,y) \leq T$. [4] Точки, що належать функції є частиною виділеного об’єкту, а всі інші – це фон зображення. Постає питання – як

саме визначити це порогове значення, від якого залежать межі виділеного об’єкту. Всього виділяють три типи порогів:

a) Глобальні;

b) Динамічні;

c) Адаптивні.

Якщо поріг T є глобальним, то це значення є однаковим для всіх точок вхідного зображення. Якщо значення T залежить від координат точок, то такий поріг називають динамічним. Якщо T залежить від значення функції $f(x, y)$, то такий поріг є адаптивним. [4]

У якості прикладу розглянемо метод з глобальним порогом. Це найпростіший метод, що передбачає вибір одного спільного порогу, після чого відбувається покрокова перевірка всіх пікселів вхідного зображення, і вони поділяються на дві групи: об’єкт та фон. Успішність цього методу та якість кінцевого результату залежить від того, чи можна поділити діаграму яскравості на окремі частини з локальними максимумами. Найкраще підходить для аналізу зображень в умовах з контрольованим освітленням. Для визначення порогу застосовують наступний алгоритм:

a) Обираємо певне початкове значення T;

b) Поділяємо зображення на два сегменти, користуючись попереднім значенням порогу, отримуємо сегменти G_1 та G_2;

c) Знаходимо середні значення яскравості для кожної області μ_1 та μ_2;

d) Рахуємо нове значення порогу $T = (\mu_1 + \mu_2)/2$;

e) Повертаємось на крок 2;

f) Зупиняємо алгоритм, коли різниця між поточною та попередньою та попередньою ітерацією менше деякого ΔT.

Однак проблема методів визначення порогів полягає в тому, що правила, за якими з вхідного зображення виділяється шуканий об’єкт, недостатньо гнучкі. По-

перше, повністю ігнорується інформація про колір, беручи до уваги лише яскравість. По-друге, деякі класи об’єктів дуже складно описати лише за допомогою градацій сірого, виділяючи певні діапазони. По-третє, якщо контрастність об’єкта по відношенню до фону є недостатньою, розпізнати його стає майже неможливо. Існує багато «традиційних» методів сегментації зображень, і кожен з цих методів орієнтований для певних умов, за яких вони виявляються найбільш ефективними. Однак всім цим методам притаманні схожі недоліки, пов’язані з спотвореннями вхідних даних. На результат роботи цих алгоритмів дуже сильно впливають просторові перетворення, складний фон з великою кількістю деталей, зміна масштабу або проекції, погодні умови та шуми.

1.2. Згорткові нейронні мережі

1.2.1. Загальна характеристика згорткової нейронної мережі

На відміну від «традиційних» методів сегментації, використання нейронних мереж для вирішення різноманітних задач демонструє меншу чутливість до спотворень вхідних даних, що значно спрощує роботу з супутниковими світлинами. Класичні нейронні мережі (багатошаровий перцептрон) дуже часто використовуються в задачах класифікації та сегментації, однак для даної проблематики такий спосіб має суттєві недоліки:

a) Супутникові фотографії мають велику роздільну здатність, що призводить до зростання кількості нейронів в мережі;

b) Чим більше нейронів та параметрів використовується для розв’язання задачі – тим більше потрібно тренувальних даних для налаштування нейронної мережі, що значно ускладнює процес навчання;

c) Цей метод є чутливим до зміни масштабу та ракурсу зйомки.

Тому для цієї задачі було обрано згорткові нейронні мережі, бо вони забезпечують стійкість по відношенню до зміщень, поворотів, масштабу та подібних спотворень. До того ж, у таких мережах використовується менша кількість нейронів та вхідних даних для тренування.

Згорткова нейронна мережа (Convolutional Neural Network) в машинному навчанні – це вид штучних нейронних мереж, що успішно застосовується для аналізу зображень. ЗНМ визначає ваги для різних ознак зображення, завдяки чому може відрізняти об’єкти. Кількість необхідної попередньої обробки є малим відносно інших алгоритмів класифікації зображень. Тоді як у більш примітивних методах фільтри створюються розробниками вручну, достатньо натреновані ЗНМ мають можливість генерувати необхідні фільтри. Вони широко використовуються для сегментації візуальної інформації, у рекомендаційних системах та програмах обробки природної мови. [6]

1.2.2. Структура згорткової нейронної мережі

Архітектурно ЗНМ складається з шарів входу, виходу та деякої кількості прихованих шарів: шарів згортки, шарів субдискритизації та повнозв’язних шарів.

Згортковий шар нейронної мережі виконує роль застосування операції згортки до виходу попереднього шару. Цей шар включає набір ядр навчання (фільтрів), які мають невелике рецептивне поле, але вони застосовують на всю глибину вхідних даних. Таким чином відбувається згортка за кожним фільтром у ширину та висоту вхідних даних. У результаті отримуємо двовимірні карти збудження кожного фільтру, кожен елемент результату – це скалярний добуток ядра та частини даних. Нейронна мережа навчається, визначаючи, які фільтри активуються, коли вона знаходить певний тип ознаки у певній просторовій частині вхідних даних. [6] Варто зазначити декілька особливостей:

a) Ядра згортки можуть бути тривимірними. Це може бути корисно для аналізу інформації про колір зображення, так як він зберігається у трьох шарах (червоний, зелений та синій). Наприклад, для усереднення інформації про кольори, можна застосувати згортку розміру $3 \times w \times h$, де w – висота, а h – ширина фільтру. В результаті на виході буде одне зображення, а не три (по зображенню для кожного кольорового каналу);

b) Операція згортки зменшує зображення, через що пікселі, що знаходяться на границі, використовуються у меншій кількості згорток, ніж центральні. Тому для збереження розміру результату, до виходу додаються пікселі по краям, що генеруються різними методами, залежно від ситуації. [7]

В основі згорткового шару лежить операція згортки. Згортка – це операція над парою матриць A (розміру $a_x \times a_y$) та B (розміру $b_x \times b_y$), в результаті операції на виході отримуємо матрицю C (розміру $(a_x - b_x + 1) \times (n_y - m_y + 1)$). [7] Кожен елемент рахується наступним чином

$$C_{i,j} = \sum_{u=0}^{b_x-1} \sum_{v=0}^{m_y-1} A_{i+u,j+v} * B_{u,v}\,. \qquad (1.1)$$

Основна ціль шару субдискритизації (агрегувальний шар) – зниження дискритизації, інакше кажучи, зменшення розмірності зображення. Вхід поділяють на декілька прямокутних секторів розміру $w \times h$, що не перетинаються, після чого до кожного блоку застосовують певну функцію. Результати застосування цієї функції до кожного з блоків і є виходом агрегувального шару. У більшості випадків на цьому етапі використовується функція максимізації, така субдискритизація називається максимізаційною. [6] Альтернативно замість функції максимізації іноді застосовують інші функції, наприклад, математичне середнє, але, за результатами останніх досліджень, функція максимізації частіше демонструє найкращі

результати. [8] Агрегувальні шари вставляють між шарами згортки. Як результат, субдискритизація зменшує кількість параметрів у мережі, пришвидшує обчислення та забезпечує інваріантність відносно незначних паралельних переносів.

Перед виходом нейронної мережі після певної кількості згорткових шарів та шарів субдискритизації, високорівневі міркування в нейронній мережі здійснюють повноз'єднані шари. У цих шарах кожен нейрон має зв'язок з усіма збудженими нейронами попереднього шару. [6]

1.2.3. U-Net

U-Net було створено в першу чергу для обробки біомедичних зображень та їх сегментації. В основі U-Net лежить згорткова мережа, структура якої змінена та розширена таким чином, аби досягти кращих результатів тренування за меншої кількості зображень. Таким чином сегментація зображення розміром 512 пікселів в ширину та 512 у висоту займає не більше секунди з новими моделями GPU. [9]

ЗНМ активно використовуються для вирішення задач, пов'язаних з класифікацією, тобто визначення міток для всього зображення. Однак, у таких сферах як біомедицина, частіше виникає проблема сегментації, коли необхідно визначити мітку для кожного пікселя. До того ж, при вирішенні класу подібних задач нерідко виникає проблема нестачі даних для тренування. [10] Саме для вирішення таких задач було розроблено U-Net.

Архітектура U-Net випливає з так званої «повністю згорткової мережі», що на відміну від звичайної ЗНМ, не містить повністю зв'язних шарів. Однією з особливостей є те, що до шарів субдискритизації додаються шари з операцією збільшення розмірності. [9] Таким чином U-Net складається з двох основних частин:

a) Шлях стискання (шлях субдискритизації). Ця частина мережі слідує архітектурі типової ЗНМ. Застосовується послідовність шарів, що повторюється певну кількість разів. Ця послідовність складається з двох шарів згортки, за кожним з яких слідує функція активації ReLU, після чого застосовується шар субдискритизації, що зменшує розмірність зображення в два рази. Після стискання кількість каналів ознак збільшується в два рази. [10] Шлях субдискритизації використовується для вивчення тимчасових ознак зображення та стискання цієї інформації;

b) Шлях розширення (шлях збільшення розмірності). Ця частина нейронної мережі виконує протилежну операцію відносно шляху субдискритиації. На кожному кроці застосовується операція збільшення розмірності в два рази, що зменшує кількість каналів ознак в два рази, після чого відбувається об’єднання з відповідною картою ознак з шляху стискання. Далі застосовуються два шари згортки, після кожного з яких застосовується функція активації ReLU. [11]

У результаті послідовного використання операції субдискритизації під час роботи шляху стискання, було втрачено частину просторової інформації про вхідне зображення. Канали ознак містять інформацію про зміст зображення, але через зменшення розмірності, недостатньо інформації про розташування сегментів на зображенні. Саме для вирішення цієї проблеми і застосовується шлях розширення, що здійснює відновлення просторової інформації зображення. Для цього використовуються так звані «мости», що слугують у якості зав’язків між шарами шляху стискання та шляху розширення. «Мости» пропускають проміжні шари та надають необхідну просторову інформацію для шарів розширення за допомогою конкатенації. Саме через свою структуру, що нагадує літеру U, мережа U-Net і отримала таку назву (рисунок 1.3).

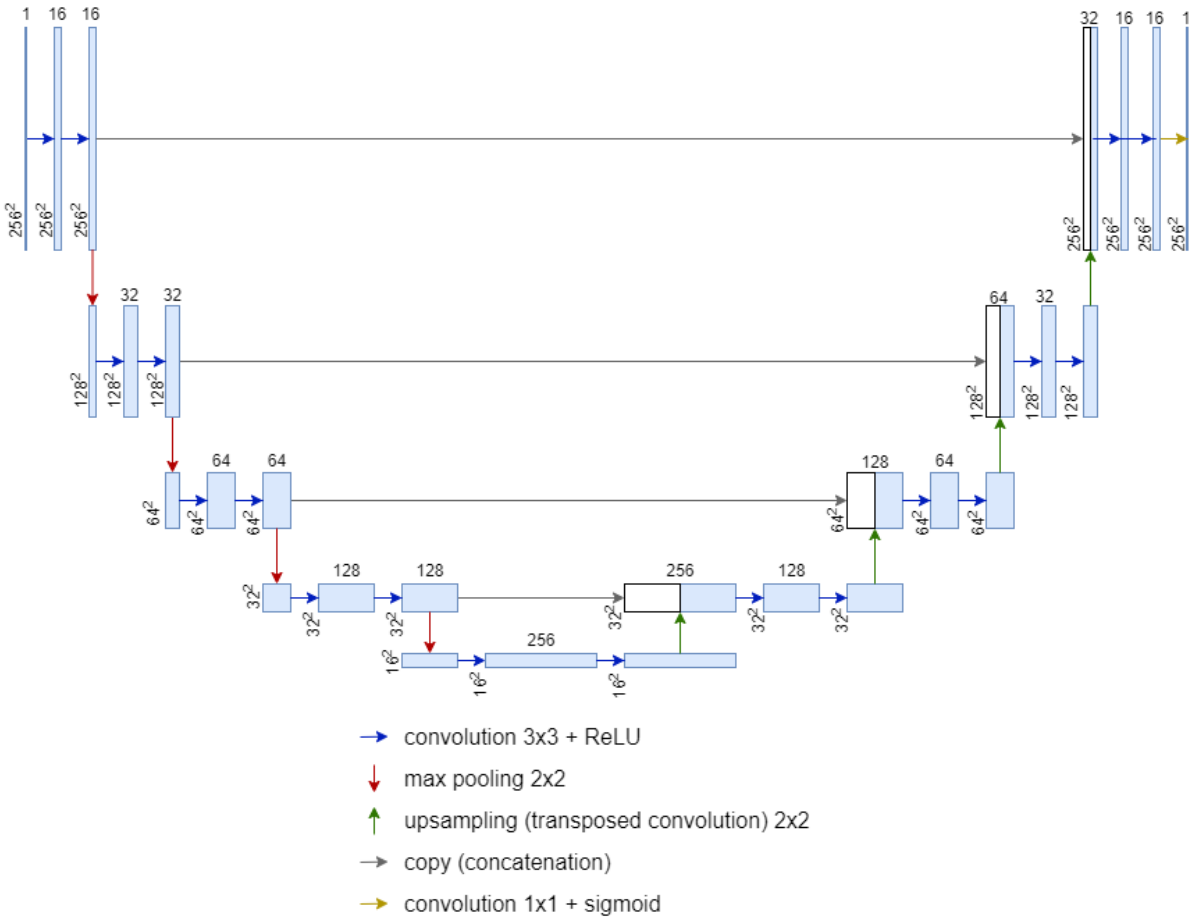

Рисунок 1.3 – Діаграма архітектури U-Net

2. РОЗРОБКА ПРОГРАМИ ДЛЯ РОЗПІЗНАВАННЯ ДОРІГ

2.1. Структура програмного рішення

Розробка системи розпізнавання доріг з супутникових зображень високої роздільної здатності пов’язана з рядом задач та проблем, які необхідно вирішувати покроково. Глобально цей процес можна поділити на три етапи: первинна підготовка вхідних даних, тренування нейронної мережі та безпосереднє використанні моделі для знаходження доріг на тестових зображеннях. Тому для кожної з цих задач розроблено окремий програмний модуль:

a) Програма обробки вхідних даних;

b) Програма тренування нейронної мережі;

c) Програма тестування розробленої моделі.

2.2. Характеристика вхідних даних

Для тренування та тестування нейронної мережі необхідно мати велику кількість вхідних даних. Від їх обсягу та якості залежить те, наскільки точно програма зможе розпізнати шукані сегменти зображень. У якості вхідного масиву використано набір даних з дорогами міста Массачусетс. Цей набір даних містить 1108 супутникових фотографій у форматі TIFF і таку ж кількість відповідних бінарних масок, на яких білим кольором позначені дороги, а чорним – все інше (рисунок 2.1). Загальна площа всіх фотографій та бінарних карт сягає 2600 квадратних кілометрів та покриває міську, приміську та сільську місцевісті з різними типами доріг (асфальтовані, ґрунтові тощо). Зображення зберігаються у форматі TIF (Tagged Image File Format), а розмір кожного екземпляру 1500 пікселів у ширину та висоту. Загальний обсяг пам’яті, що займає на диску вся колекція фотографій та масок, сягає 9,3 ГБ. [14]

а) б)

Рисунок 2.1 - Приклад фотографії (а) та відповідної бінарної маски (б)

Нажаль, обраний набір даних не є ідеальним для навчання і має деякі недоліки. Перш за все, розмір фотографій є завеликим, адже збільшення розмірності вхідних даних призводить до збільшення кількості необхідних для тренування нейронів, що сповільнить процес навчання. Цю проблему можна вирішити фрагментацією зображень та їх масок на частини однакового розміру. Наступним недоліком є некоректність деяких бінарних масок по відношенню до відповідних фотографій, яким вони належать. Це проявляється у тому, що не всі дороги правильно відображені на бінарних масках, а іноді – зовсім відсутні деякі ділянки доріг (рисунок 2.2). Дуже рідкісними є випадки, коли на певній частині фотографії немає дороги, а на відповідній масці вона є. Нажаль, для цієї проблеми неточності вхідних даних, немає певного рішення. І третя проблема – фізичні перешкоди, що приховують дороги та заважають їх розпізнаванню, такі як тіні від багатоповерхівок (рисунок 2.3), густа рослинність (зокрема, високі дерева) та відсутність чіткого розмежування доріг тротуаром або газоном. Нажаль, знайти ці проблеми можна тільки перевіряючи зображення вручну, але здебільшого набір даних є коректним та підходить для тренування.

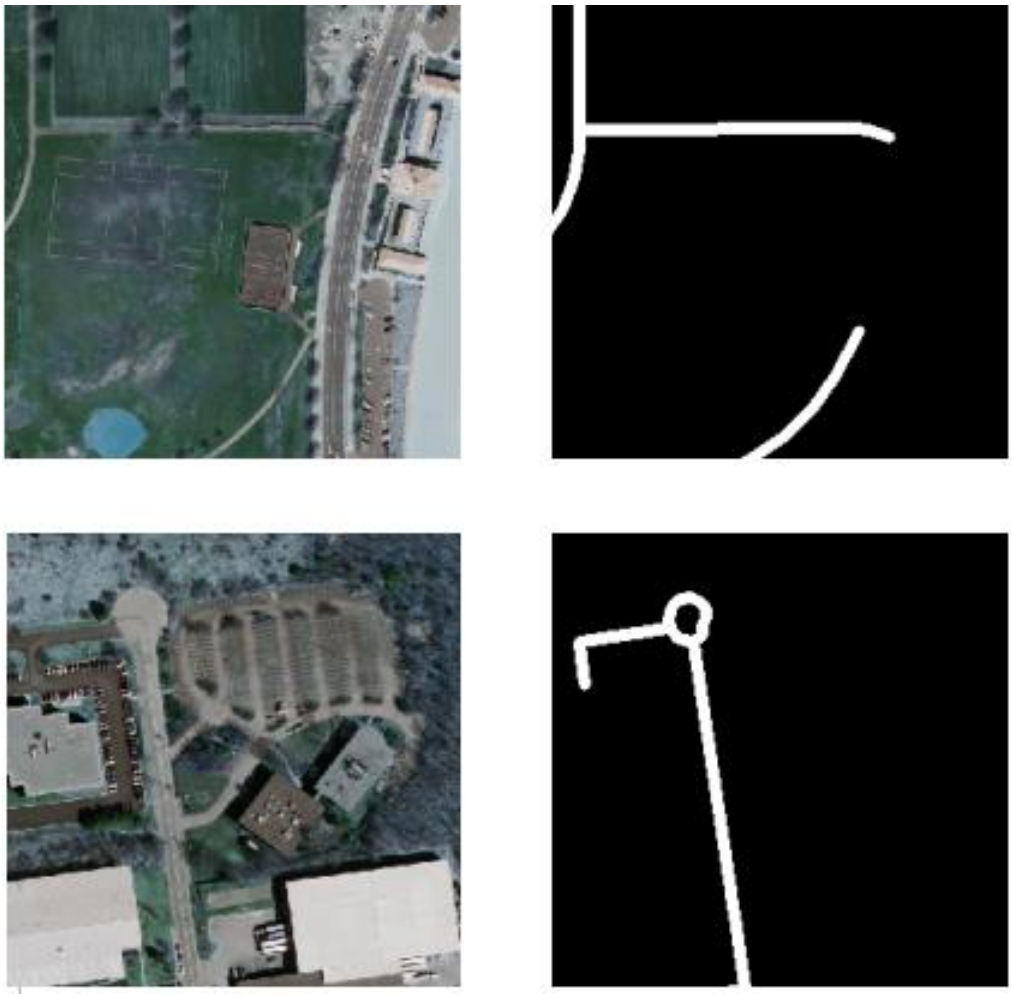

Рисунок 2.2 – Фрагменти зображень з дефектними бінарними масками

Рисунок 2.3 – Фрагмент зображення, на якому дорогу приховує тінь

2.3. Використані технології

Для розробки даної програмної системи було використано наступний набір технологій:

a) Усі модулі реалізовано мовою Python. Вибір саме цієї мови програмування мотивовано широким набором інструментів для роботи з великими масивами даних та для машинного навчання. Перевагами Python над іншими мовами, для яких є бібліотеки для роботи з нейронними мережами (наприклад, Java), є гнучкість та динамічна типізація, що спрощує роботу з різними типами даних. До того ж, популярність цією мови стрімко зростає, що спрощує пошук рішень технічних проблем під час розробки.

b) Для створення структури нейронної мережі та навчання використано бібліотеку Keras. Ця бібліотека надає доступ до широкого ряду будівельних блоків (шари, цільові функції і т.д.), що спрощує побудову глибоких нейронних мереж. Варто зазначити, що Keras надає доступ лише до високорівневого API, у той час, коли Tensorflow надає доступ як до високорівневих, так і до низькорівневих функції. Однак набір інструментів Keras більш простий для використання та розуміння, і їх достатньо для вирішення поставленої задачі. [12]

c) Для роботи з графічними зображеннями застосовано бібліотеку OpenCV. Вона надає достатньо інструментів для різноманітних маніпуляцій з зображеннями. Перевагою цієї бібліотеки над іншими подібними інструментами є робота з фотографіями як з масивами NumPy, що спрощує їх обробку.

d) Для зберігання даних використано формат HDF5. Це ієрархічний формат, що дозволяє зберігати в одному файлі певну структуру даних. При цьому при зчитуванні можна прочитати як весь файл, так і певну частину структури. Дані зберігаються в двійковому форматі і підтримується декілька фільтрів стискання, таких як GZIP, LZF та SZIP.

e) Для тренування та тестування нейронної мережі використано Google Colab. Це хмарний сервіс на основі Jupyter Notebook, що надає усі необхідні інструменти для розробки та доступ до потужних GPU та TPU. Безкоштовна версія цього ресурсу також надає 12,72 ГБ оперативної пам’яті та 68,4 ГБ дискового простору. Це набагато більше, ніж у аналогічного сервісу Azure Notebook, що надає 4 ГБ оперативної пам’яті.

2.4. Обробка вхідних даних

У першу чергу необхідно здійснити збір та обробку великого масиву вхідних даних для подальшої їх обробки. Для цього розроблено окремий модуль prepare_data.py, у якому здійснюються початкові маніпуляції з даними. Основними задачами цього модуля (create_dataset.py) є поділ зображень великої роздільної здатності на менші фрагменти однакового розміру, фільтрації фрагментів, та їх розподіл на масив для тренування і масив для тестування. Налаштування програми вказуються у відповідному конфігураційному файлі config.ini:

a) root – шлях до кореневого каталогу

b) img_size – розмір квадратних фрагментів зображень (за змовченням 256);

c) test_files_percent – відсоток зображень для тестування;

d) images_subfolder – підкаталог з супутниковими фотографіями;

e) masks_subfolder – підкаталог з бінарними масками, кожна з яких відповідає фотографії з images_subfolder;

f) result_images_subfolder – підкаталог, у який будуть записані отримані у результаті фрагменти фотографії;

g) result_masks_subfolder – підкаталог, у який будуть записані отримані у результаті фрагменти бінарних масок;

h) save_num – загальна кількість фрагментів.

Першим кроком цього модуля є поділ зображень та масок, розташованих у відповідних каталогах, вказаних у файлі налаштувань. Для цього збираються всі назви файлів, що знаходяться у каталозі з фотографіями, після чого відбувається їх послідовний поділ на фрагменти разом з відповідними бінарними масками. У випадку, якщо розмір зображення не ділиться націло на розмір фрагменту, додаються чорні пікселі, щоб всі фрагменти були однакового розміру. Стандартні ширина та висота фрагменту дорівнюють 256 пікселів.

Далі, перед тим як записати отриманий фрагмент та його маску на диск, перевіряється, чи достатньо у ньому корисної інформації. Якщо його бінарна маска майже повністю чорна – це означає, що на ньому немає доріг, отже він не несе користі для тренування, і його можна відкинути. Тому пари, у яких маски на 99% чорні, не зберігаються. При записі фрагменту на диск, його назва – це назва оригінального зображення, до якого додано номер фрагменту.

Наступним кроком роботи модуля є розподіл отриманих фрагментів для тестування та тренування нейронної мережі. Для цього фрагменти поділяються на дві частини таким чином, щоб розмір даних для тестування складав певний відсоток (test_files_percent) від загальної кількості (save_num), а для тренування – решта. Обидві структури складаються з двох масивів зображень: фрагменти фотографій та фрагменти бінарних масок. При цьому для кожної фотографії має бути відповідна маска з такою ж назвою. Таким чином зберігаються два файли формату HDF5: для тренування та тестування, кожен з яких містить два масиви: масив фотографій та масив відповідних бінарних масок.

2.5. Тренування нейронної мережі

Після обробки зображень та створення необхідних для роботи масивів даних, наступним кроком є безпосередня розробка моделі нейронної мережі та її

тренування. Модуль road_finder.ipynb реалізує модель U-Net та використовує її для тренування з великими масивами зображень. Для зручності розробки та виконання коду, програму написано у Google Colab, а отримані з попереднього кроку дані завантажено у Google Drive.

У цьому модулі використовується ряд параметрів для налаштування нейронної мережі та її тренування:

a) ACT_FUNCTION – назва функції активації для згорткових шарів;

b) KERNEL_INIT – тип ініціалізатора початкових ваг шару;

c) PADDING_TYPE – вказує, чи потрібно збільшити вихід, щоб його розмір відповідав розміру входу;

d) EPOCH_NUM – кількість епох тренування на всьому масиві даних;

e) LEARNING_RATE – коефіцієнт швидкості навчання, що дозволяє керувати величиною корекції ваг на кожній ітерації;

f) BATCHES_SIZE – кількість елементів з масиву даних в одному блоці для тренування;

g) CHECKPOINT_MODEL_PATH – шлях для збереження проміжних результатів тренування;

h) FINAL_MODEL_PATH – шлях для збереження результату тренування;

У першу чергу необхідно вирішити проблему, пов’язану з дисбалансом вхідних даних. Дисбаланс полягає в тому, що класи для сегментації розподілені нерівномірно, через що деякі з них мають недостатньо репрезентації. Це поширена проблема у задачах, пов’язаних з сегментацією, так як більшість алгоритмів сегментації припускають, що кількість зразків кожного класу однакова. Якщо не боротися з дисбалансом, якість передбачень для класів з недостатньою репрезентацією буде дуже низька. Тому для боротьби з дисбалансом необхідно налаштувати модель таким чином, щоб вона була більш чутлива до помилок, пов’язаних з цими класами.

Проблема дисбалансу притаманна задачі сегментації доріг на супутникових зображеннях, адже кількість білих пікселів (що позначають дороги) на бінарних масках значно менша за кількість чорних пікселів. Це пов’язано з тим, що дороги займають відносно малу площу зображення, однак саме їх сегментація має більший пріоритет. Тобто правильно передбачити білий піксель набагато важливіше, ніж чорний.

Для боротьби з дисбалансом у поставленій задачі перевизначено функцію втрат. Функція втрат відображає деяку подію у вигляді дійсного числа, що інтуїтивно представляє вартість цієї події. Для нейронних мереж це вартість помилки, тому метою навчання є мінімізація втрат. Для даної задачі у якості функції втрат використано Soft Dice Loss, заснований на Dice коефіцієнті (формула 2.1). Цей коефіцієнт визначає подібність двох масивів даних, а у випадку сегментації він обчислює міру відповідності між передбаченням та істиною. Значення коефіцієнта може бути від 0 до 1, де 0 означає, що між зразками немає подібності, а 1 – що зразки повністю співпадають. У свою чергу функція втрат Soft Dice Loss визначається як обернений Dice коефіцієнт (формула 2.2).

$$D = \frac{2 * |X \cap Y|}{|X| + |Y|}, \tag{2.1}$$

де X та Y – скінченні множини;

D – Dice коефіцієнт.

$$SDL = 1 - D, \tag{2.2}$$

де SDL – функція втрат Soft Dice Loss.

Використання Soft Dice Loss вирішує проблему дисбалансу, адже коефіцієнт Dice враховує лише клас сегментації (пікселі дороги). Так як під час підрахунку клас фону ігнорується, він не домінує над шуканим класом.

Окрім функції втрат, необхідно оцінити якість передбачень, що генерує модель. Для даної задачі використано метрику Intersection Over Union, також відому як коефіцієнт Жаккара. Це бінарна міра подібності, що використовується для оцінки схожості скінченних множин, і визначається як відношення кількості елементів перетину до кількості елементів об'єднання. [13]

$$IOU = \frac{|X \cap Y|}{|X \cup Y|}, \tag{2.3}$$

Де A і B – скінченні множини;

IOU – коефіцієнт Жаккара.

Таким чином, у чисельнику формули (2.3) знаходиться площа перетину передбачення та істини, а у знаменнику – їх об'єднана площа, отже знайдено «відсоток істинності» передбачення.

Далі необхідно визначити модель нейронної мережі для тренування. Структура розробленої нейронної мережі відповідає архітектурі U-Net та складається з наступної послідовності шарів:

a) Шар входу;

b) Чотири послідовності шарів для шляху стискання:

1) Два згорткових шари з активаційним шаром ReLU після кожного;

2) Шар субдискритизації;

c) Два згорткових шари з активаційним шаром ReLU після кожного;

d) Чотири послідовності шарів для шляху розширення:

1) Транспонований згортковий шар;

2) Конкатенація попереднього поточного виходу з відповідним шаром з шляху стискання;

e) Два згорткових шари з активаційним шаром ReLU після кожного;

f) Шар виходу.

Процес тренування моделі полягає в оптимізації параметрів ваг задля мінімізації функції втрат. Однак крім функції втрат у цьому процесі використовується функція-оптимізатор. Задача оптимізатора – корекція ваг моделі та швидкості навчанням таким чином, щоб пришвидшити навчання та досягнути мінімальних втрат. Для задачі сегментації доріг обрано оптимізатор Adam. Це один із найефективніших методів оптимізації для глибоких нейронних мереж, що поєднує в собі ідеї середньоквадратичного поширення (RMSProp) та адаптивного градієнтного алгоритму (AdaGrad). На відміну від стохастичного градієнтного спуску, що використовує стале значення швидкості навчання для всіх ваг, оптимізатор Adam корегує значення швидкості для кожного параметра окремо. Завдяки цьому цей метод швидко досягає бажаного результату під час тренування. Іншими перевагами цього методу є ефективність обчислень та ефективність використання оперативної пам’яті. [16]

Для покращення процесу тренування нейронної мережі використано ряд функцій зворотного виклику (callbacks). Ці функції застосовуються в певні моменти під час навчання для виводу інформації про поточний результат та для інших задач. Для тренування цієї моделі використано три функції зворотного виклику:

a) EarlyStopping – функція, що зупиняє процес навчання після того, як певний параметр не покращується. Для даної задачі перевіряється, чи покращилось (зменшилось) значення функції втрат порівняно з поточним знайденим мінімумом. Якщо після останньої епохи не відбулось покращення, то поточні значення ваг моделі відкидаються і встановлюються ваги епохи з мінімальним значенням

функції втрат. Якщо після 5-и епох підряд модель не покращила свої результати, то процес тренування зупиняється.

b) ReduceLROnPlateau – функція, що здійснює корекцію швидкості навчання. Якщо обрана метрика (функція втрат) не покращується протягом 4-ох епох підряд, параметр швидкості навчання зменшується на один порядок.

c) ModelCheckpoint – функція, що зберігає модель та поточні ваги під час навчання. Це необхідно для продовження тренування з останнього збереженого стану без необхідності будувати модель та тренувати її спочатку. Для даної задачі ця функція налаштована таким чином, щоб модель зберігалась лише після епохи, під час якої значення функції втрат покращилось. [15]

2.6. Результати роботи обробки вхідних даних

Всього на вхід є 1108 супутникових фотографій та стільки ж бінарних масок. Кожен знімок та маска поділяються на 36 частин розміру 256 пікселів у ширину та висоту. Згенеровано 39888 фрагментів, що підлягають фільтрації. Після відкидання зображень, що не містять корисної для тренування інформації (фотографії з майже повністю чорними бінарними масками), залишилось 19825 частин (рисунок 2.4). Загалом цей процес зайняв 362,25 секунди. Далі ці фрагменти використовуються для тренування моделі U-Net. Нажаль, ресурси Google Colab є обмеженими, тому необхідно обрати таку кількість зображень, з якою можна оперувати в оперативної пам’яті. Експериментальним шляхом було визначено, що при тренуванні нейронної мережі з 8000 зображень, максимальне використанням ресурсів пам’яті сягає 12,5 ГБ, що дуже близько до верхньої межі (12,72 ГБ). Саме тому з відфільтрованих фрагментів випадковим чином обрано 8000 (1,48 ГБ) для тренування та 2000 (0,35 ГБ) для тестування.

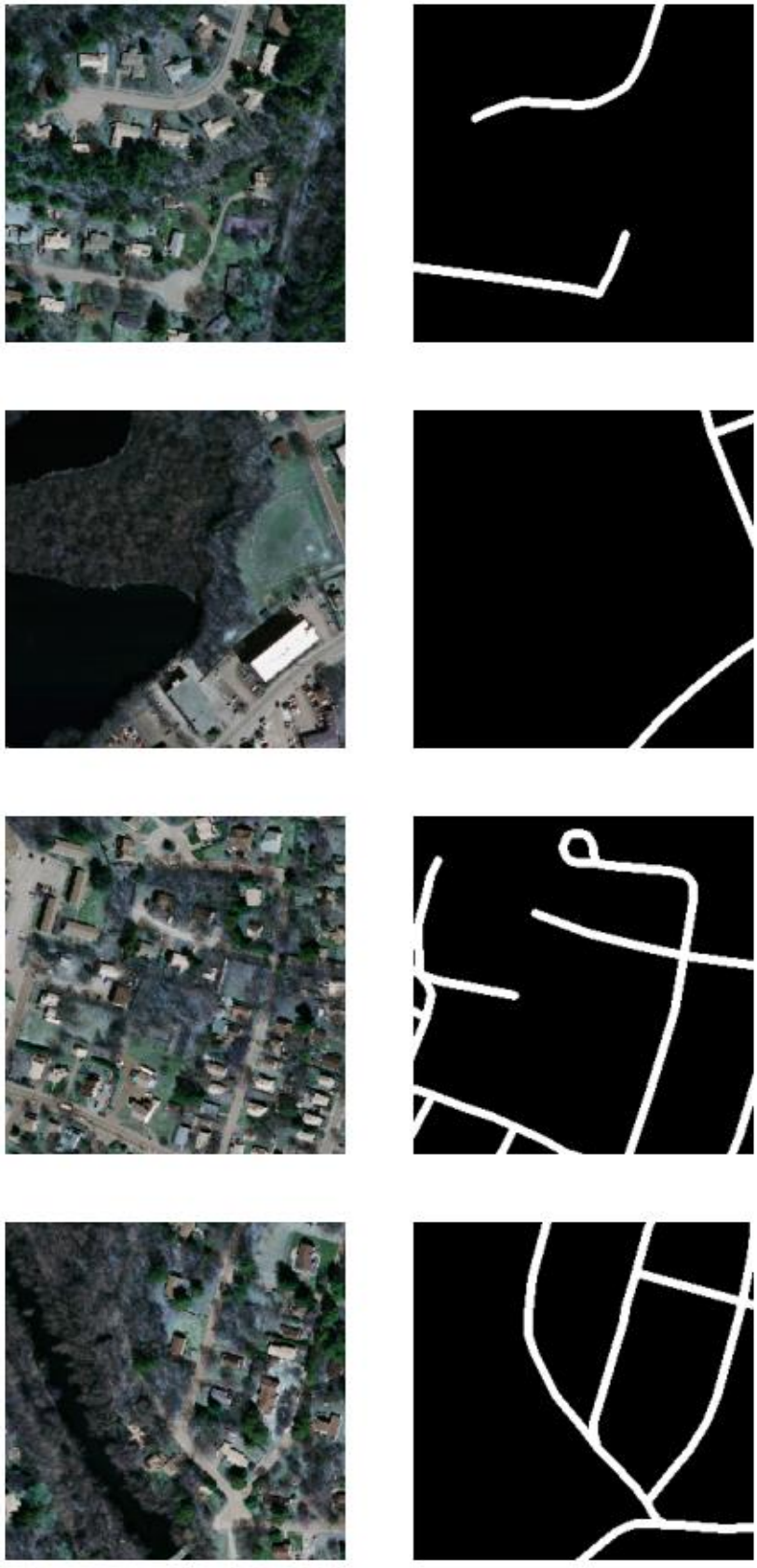

Рисунок 2.4 – Приклади фрагментів фотографій (зліва) та їх масок (справа)

2.7. Результати тренування

Далі розглянемо результати роботи модуля тренування нейронної мережі та проведемо відповідні тести розробленої моделі. У першу чергу важливо визначити, яким чином кількість вхідних даних впливає на процес тренування та якість передбачень, що надає система у результаті. Всього створено 7 окремих моделей, що тренувалися з різною кількістю фрагментів супутникових зображень: 100, 250, 500, 1000, 2000, 4000, 8000. Усі тести здійснено у модулі road_finder_test.ipynb Звичайно, кількість зображень сильно впливає на швидкість навчання (рисунок 2.5), адже зі зростанням об’єму вхідних даних, збільшується кількість блоків тренування (batches). Збільшення кількості зображень в два рази збільшує час виконання в також приблизно вдвічі.

Рисунок 2.5 – Графік залежності часу навчання від об’єму вхідних даних

Окрім часу витраченого на навчання, необхідно визначити ефективність моделі, оцінивши якість передбачень. Для цього фіксуємо середні значення функції втрат Soft Dice Loss та метрики Intersection Over Union. Так як ці дві функції дуже схожі за своєю будовою, їх значення корелюються: зростання IOU призводить до спадання SDL та навпаки. На графіку (рисунок 2.5) спостерігаємо покращення показника IOU при збільшенні об’єму вхідних даних, однак після 2000 зображень зростання поступово сповільнюється.

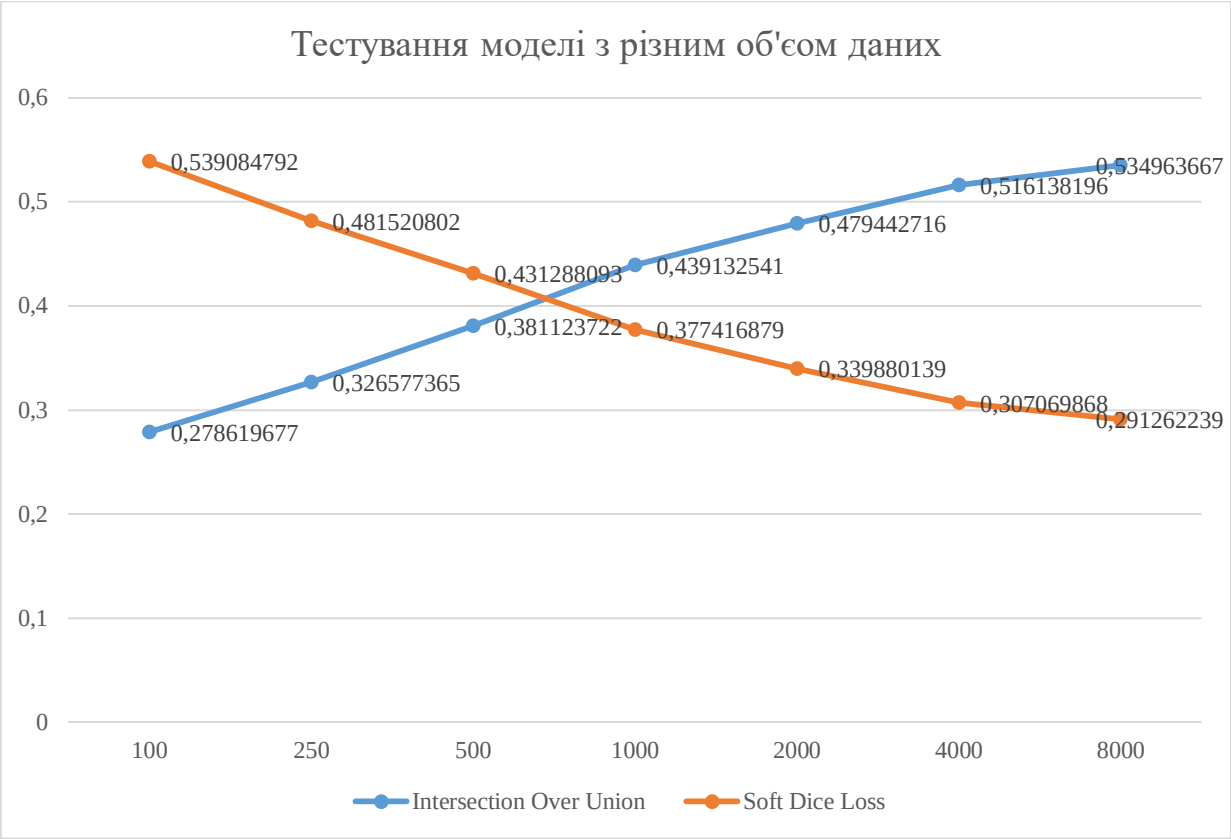

Рисунок 2.6 – Графік, що демонструє результати тренування

В результаті найкращі показники демонструє модель, для навчання якої використано 8000 зображень. IOU сягає приблизно 0,53, а значення функції SDL зменшилось до 0,29.

2.8. Передбачення моделі

Окрім статистичних показників та значень метрик, важливо порівняти результати передбачень для кожної моделі, що навчалися з різною кількістю вхідних даних. Нажаль, через особливості вхідних даних, показників IOU та SDL недостатньо, аби оцінити правильність передбачень. Необхідно перевіряти їх вручну, порівнюючи з фотографіями та відповідними бінарними масками.

У першу чергу розглянемо варіант з найменшою кількістю вхідних даних – 250 фотографій. Як бачимо, модель не може ефективно впоратися з розпізнаванням дороги на пересічній місцевості (рисунок 2.7), бо на знімку дорогу приховують дерева та тіні. Тому видно лише окремі білі плями, що відповідають освітленим ділянкам дороги. До того ж, модель розпізнала довгу будівлю як широкий шматок дороги. На наступному знімку (рисунок 2.8) ситуація набагато краща. Видно, що у випадках, коли дорога добре освітлена та має чіткі межі, модель може добре її розпізнавати, але через тіні та інші перешкоди, великі ділянки вулиць виявилися роз’єднаними. На знімках, де сонячні промені освітлюють територію майже перпендикулярно (рисунок 2.9), передбачення моделі досить близько наближається до еталонного. Основні вулиці добре видно, але ділянки доріг, приховані за деревами, система не змогла розпізнати.

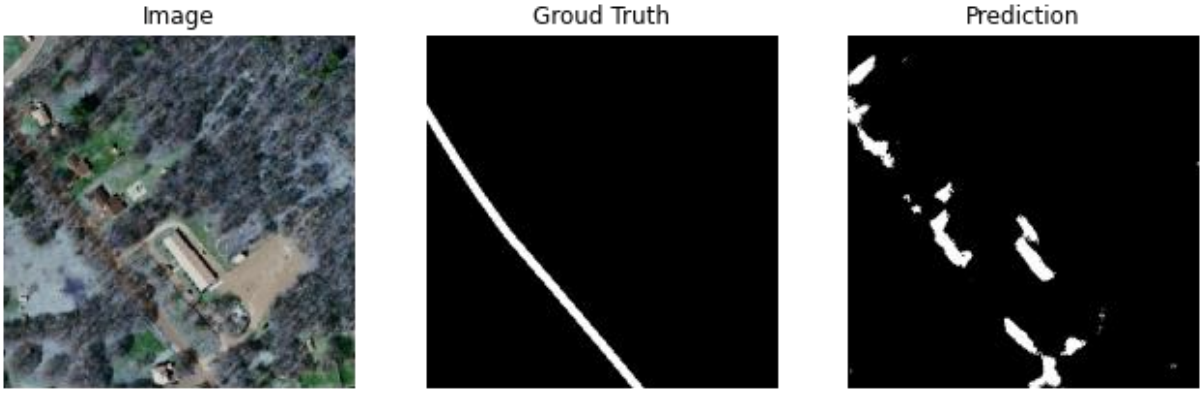

Рисунок 2.7 – Розпізнавання дороги на пересічній місцевості (250 фото)

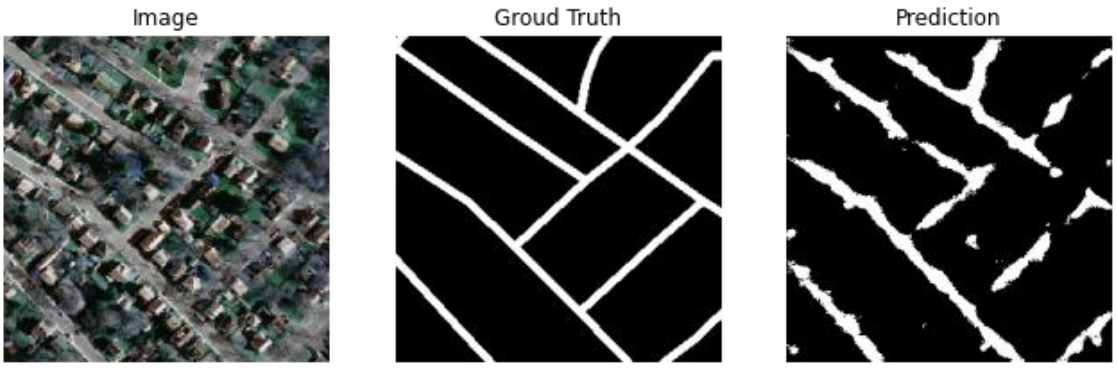

Рисунок 2.8 – Розпізнавання міських вулиць (250 фото)

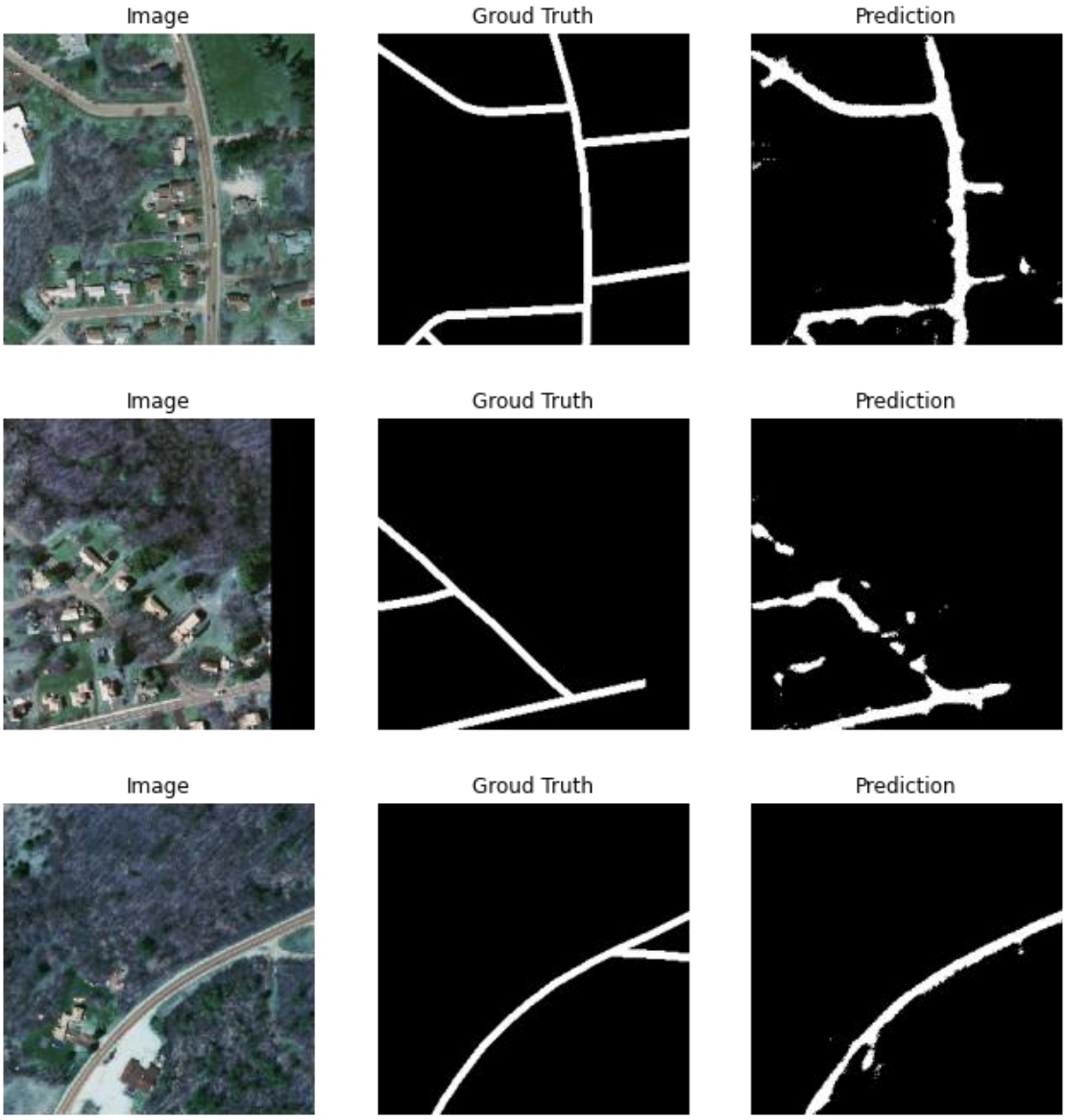

Рисунок 2.9 – Розпізнавання доріг під прямим освітленням (250 фото)

Переглянувши результати тренування з низькою кількістю вхідних даних, можна зробити висновок, що система непогано розпізнає дороги, що знаходяться під прямим освітленням. Однак перешкоди різного типу, такі як дерева, тіні та будинки, призводять до спотворень результатів у більшості випадків. До того ж, межі вулиць криві та нечіткі.

Далі розглянемо роботу системи після тренування з 500 знімками. Одразу видно покращення на першому знімку (рисунок 2.10). Лінії стали більш чіткими, зникли випадкові білі плями, а будинки з сірими дахами більше не розпізнаються як дороги. Однак ділянки доріг все ще «порізані» через дерева. Передбачення на міських вулицях працює краще. Ділянки доріг стали з’єднаними, але все ще є прогалини через тіні від будинків, що перекривають шляхи. На інших фотографіях також видно значні зміни: більш чіткі лінії та менше шуму. Очевидно, що система працює набагато краще, але модель вимагає додаткового тренування, особливо для розпізнавання доріг, прихованих різними перешкодами.

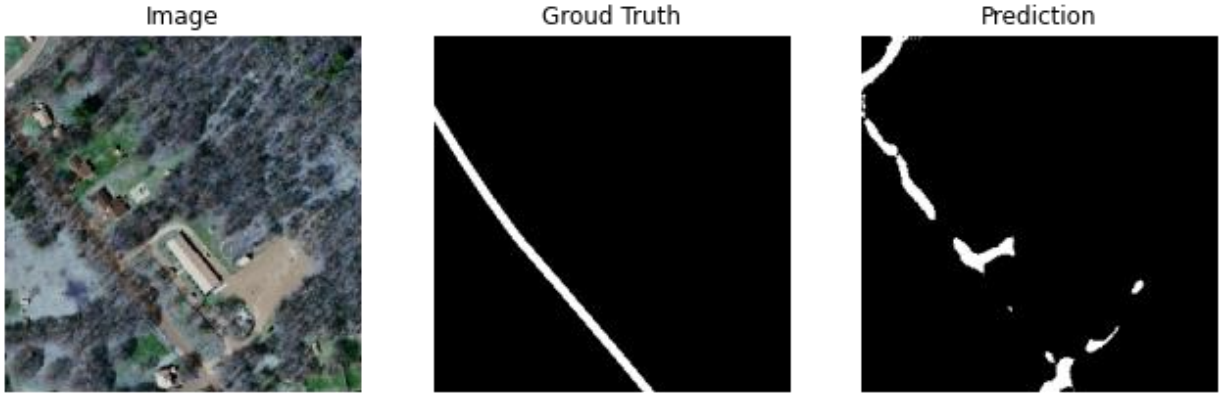

Рисунок 2.10 – Розпізнавання дороги на пересічній місцевості (500 фото)

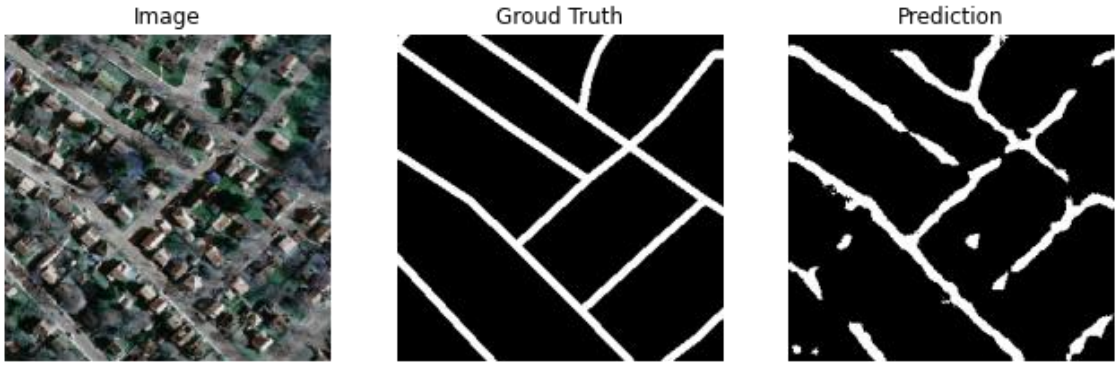

Рисунок 2.11 - Розпізнавання міських вулиць (500 фото)

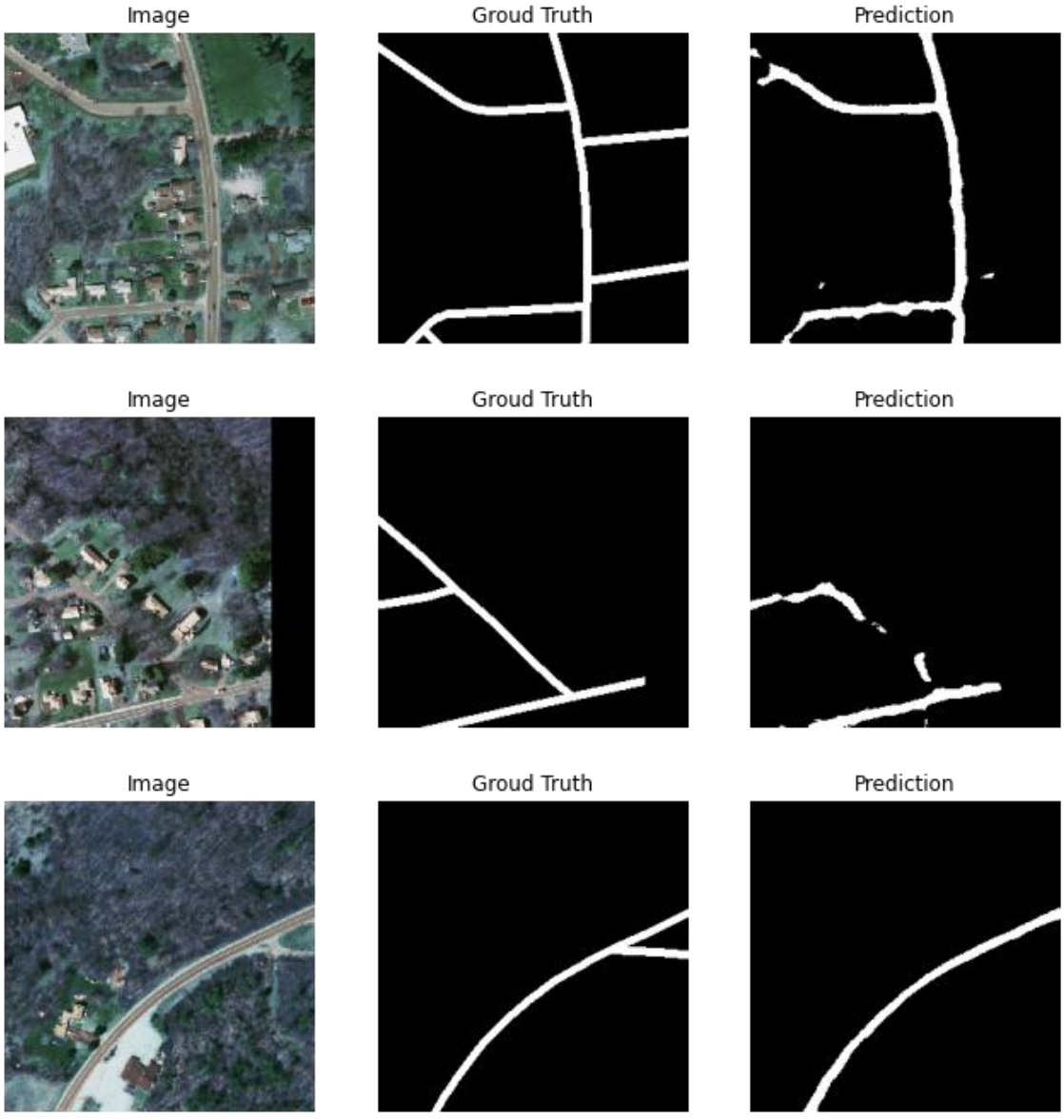

Рисунок 2.12 - Розпізнавання доріг під прямим освітленням (500 фото)

Наступний крок – 1000 фотографій. Після значного збільшення вхідних даних, якість передбачень помітно зросла. Тепер дорога на пересічній місцевості (рисунок 2.13) являє собою суцільну смугу. Але варто зазначити, що ця смуга має різну товщину, і поруч з нею розпізнано розірвані шматки доріг. Їх немає на еталонній бінарній масці, але на фото видно невеликі ділянки доріг поруч з довгою будівлею. Міські вулиці розпізнано набагато краще (рисунок 2.14). Тепер майже всі дороги з’єднані правильно, окрім однієї, прихованої за тінню. Щодо інших прикладів, розпізнані ділянки доріг стали більш рівними та чіткими, але не всі вони з’єднані правильно.

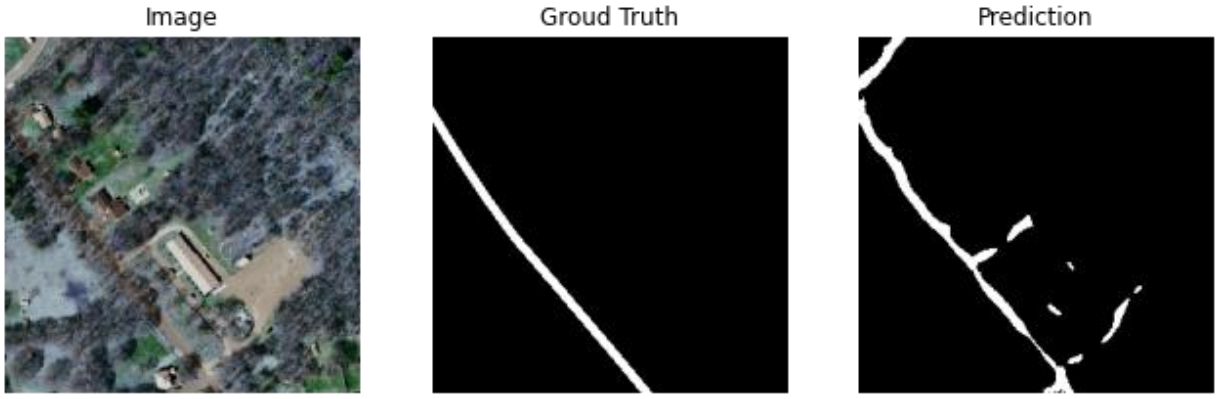

Рисунок 2.13 – Розпізнавання дороги на пересічній місцевості (1000 фото)

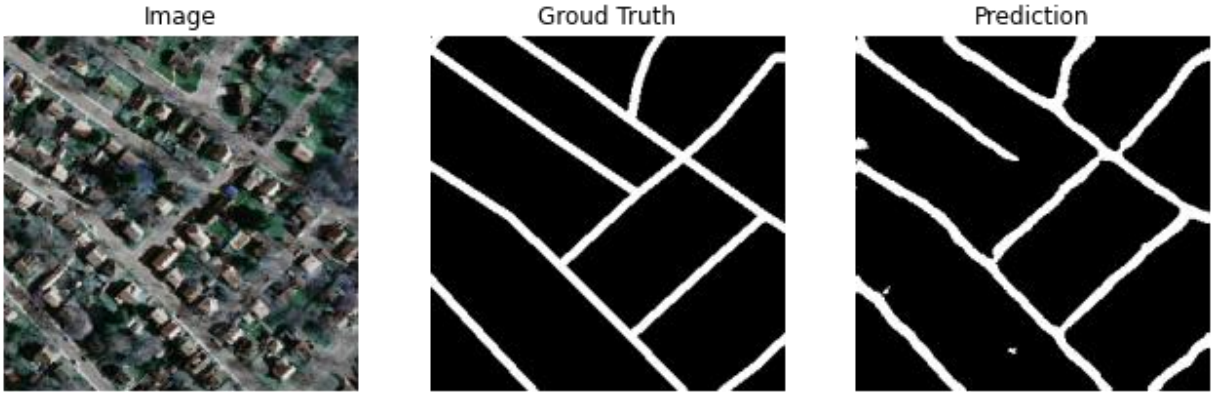

Рисунок 2.14 – Розпізнавання міських вулиць (1000 фото)

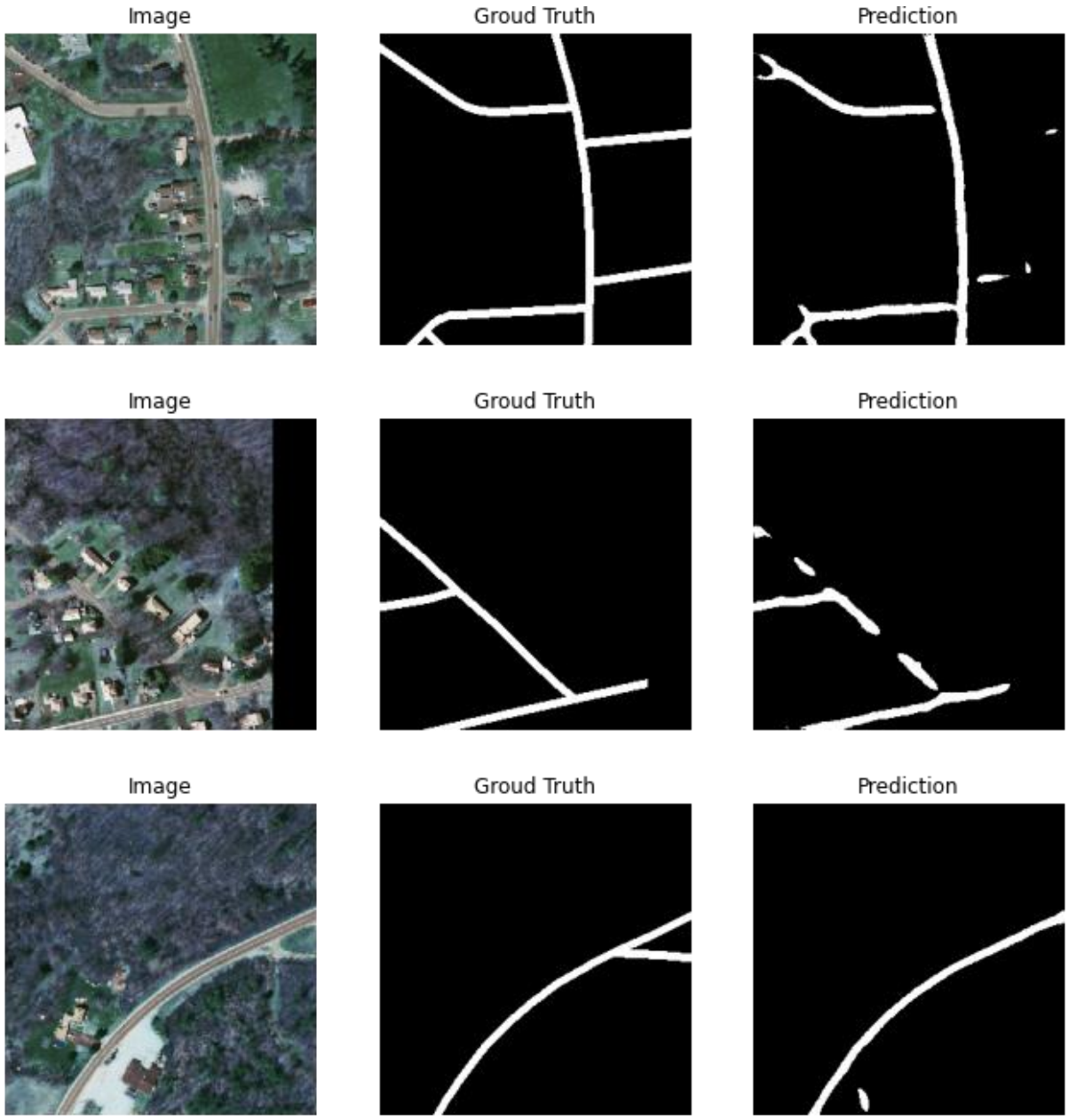

Рисунок 2.15 - Розпізнавання доріг під прямим освітленням (1000 фото)

Далі кількість фото для тренування збільшується до 2000. Переглянемо результати передбачень для зображень з попередніх прикладів (рисунок 2.16). Якість передбачень збільшилась, адже розірваних ділянок доріг стало менше, а товщина білих смуг тепер рівномірна та відповідає реальній ширині доріг. Хоч покращення і незначне, але помітне, особливо для фотографій у міській місцевості.

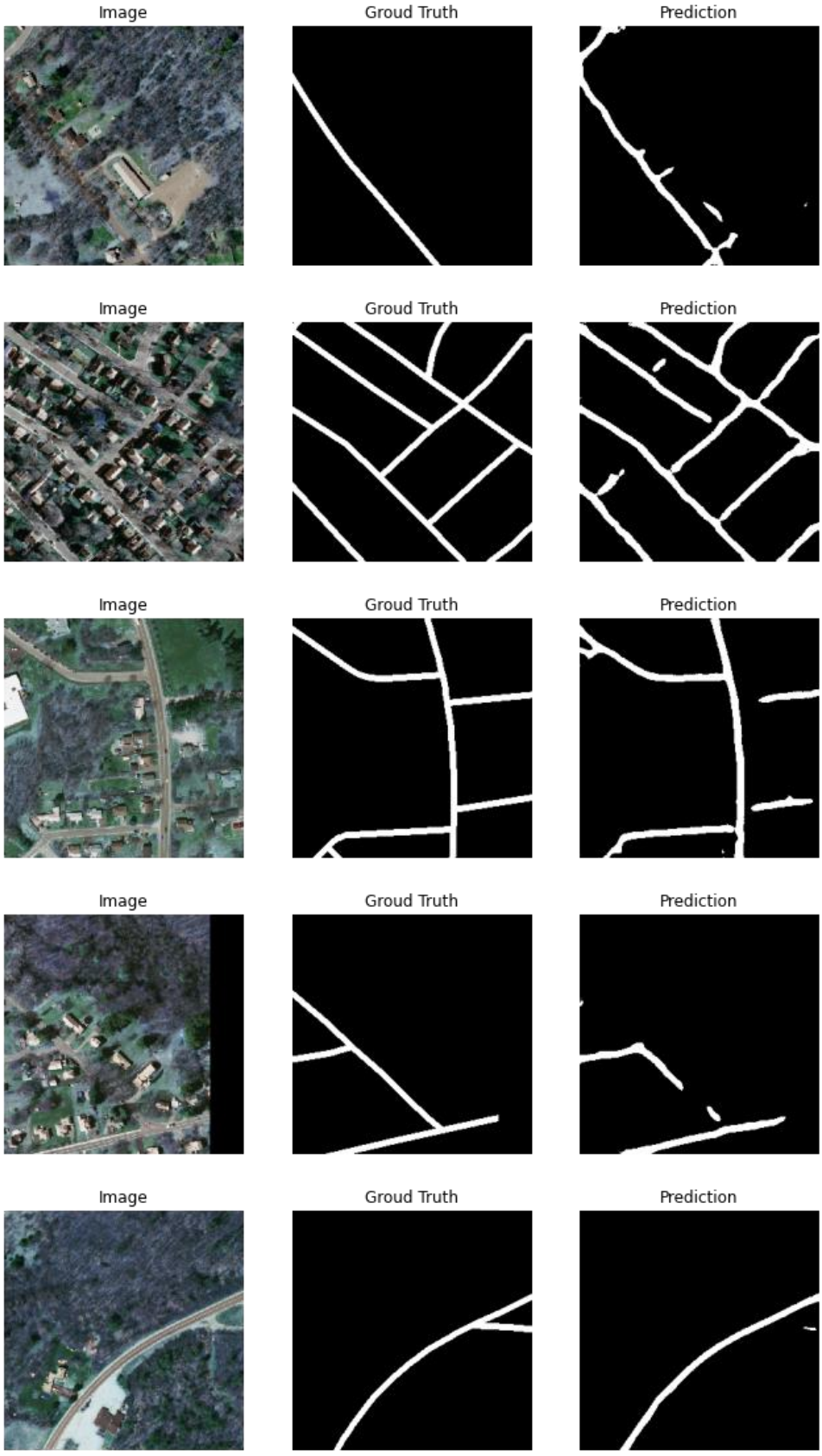

Рисунок 2.16 – Розпізнавання доріг (2000 фото)

Розглянемо різницю між тренуванням з 2000 зображень та 4000 зображень. Відмінностей у результатах досить мало, але в цілому передбачення покращились. Тепер передбачення у міській місцевості (рисунок 2.17) близькі до еталонних: майже всі ділянки доріг з’єднані правильно. Значно покращилось передбачення на приміській та пересічній місцевості (рисунок 2.18): зменшилась кількість шуму, а лінії доріг стали більш прямими та чіткими. Щодо інших прикладів, система краще впоралась з розпізнаванням доріг, закритих деревами, але різниця з попередніми прикладами незначна (рисунок 2.19).

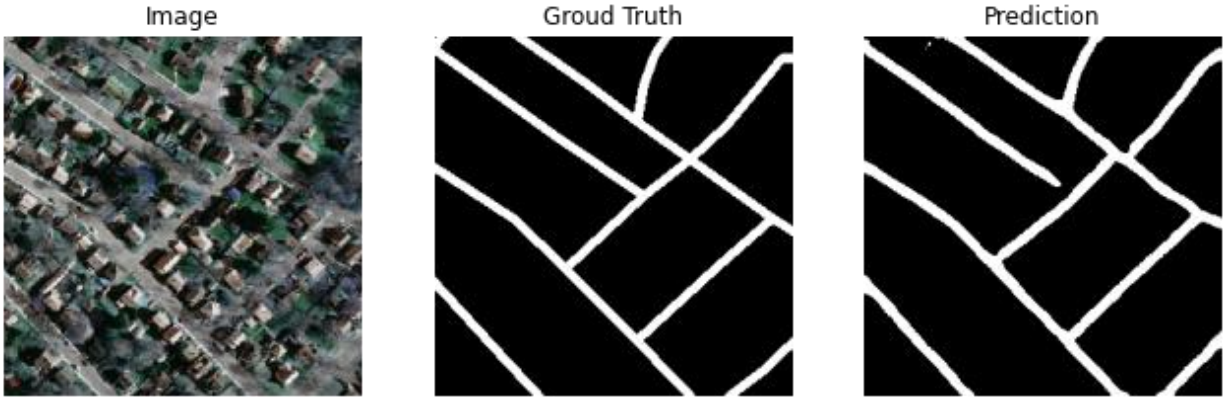

Рисунок 2.17 – Розпізнавання міських вулиць (4000 фото)

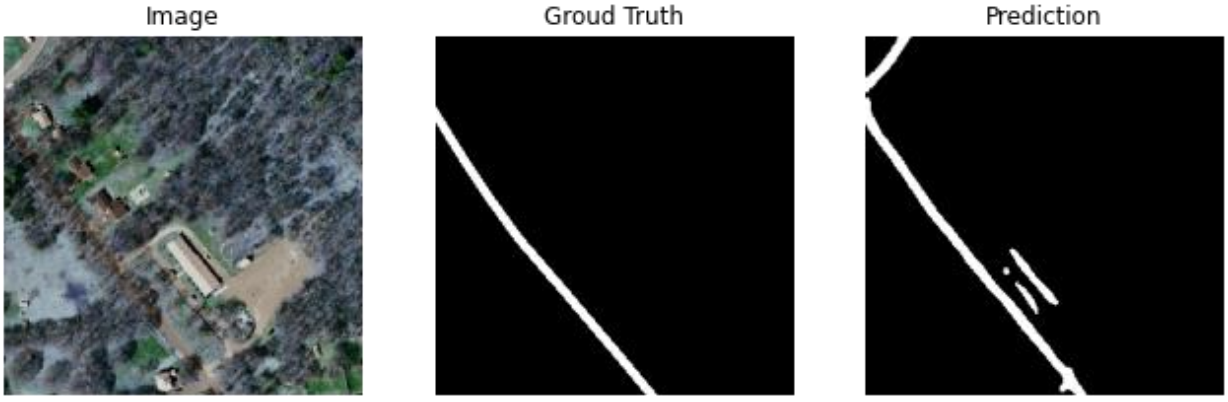

Рисунок 2.18 – Розпізнавання дороги на пересічній місцевості (4000 фото)

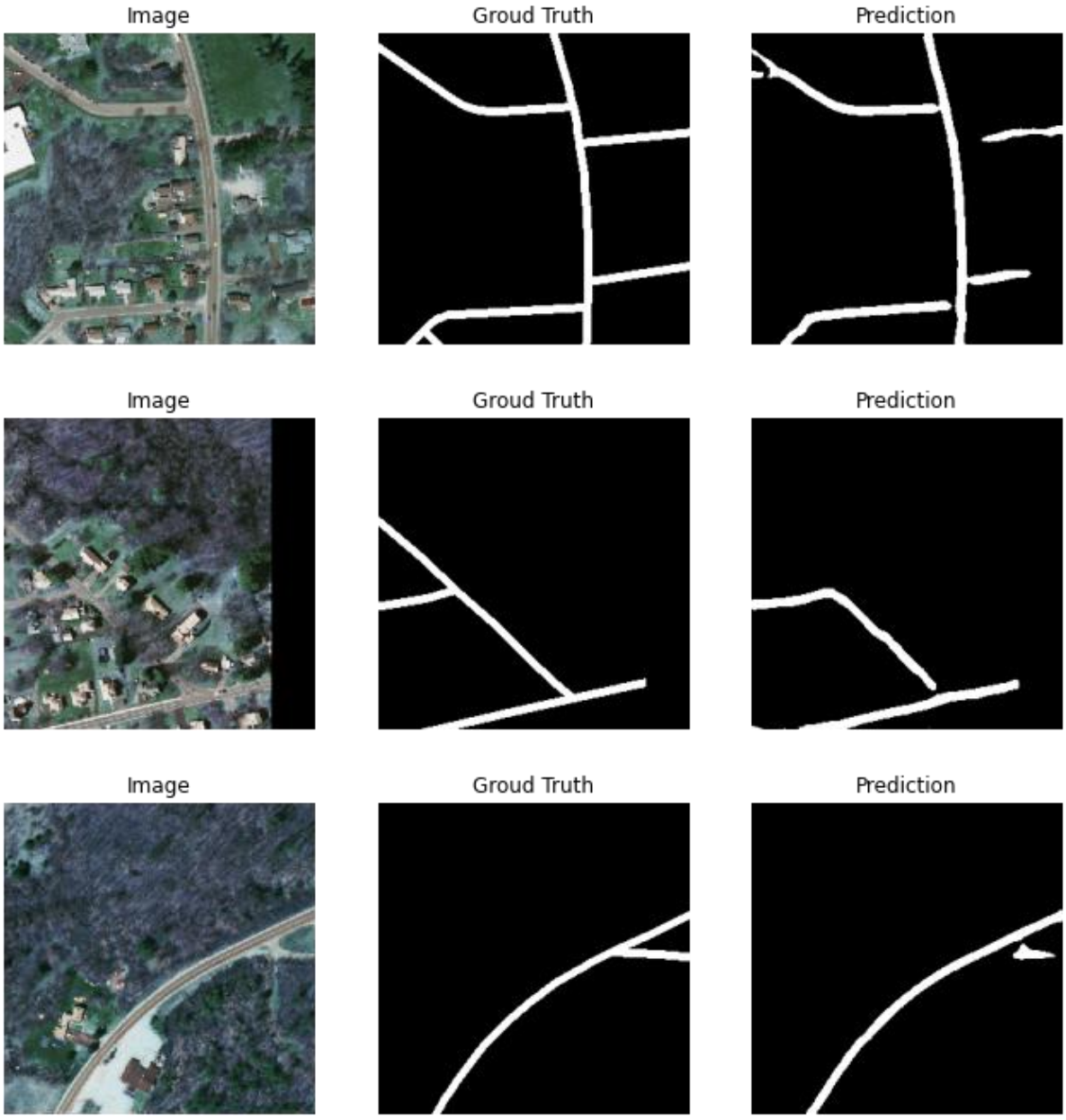

Рисунок 2.19 – Розпізнавання доріг (4000 фото)

Нарешті, для перевірки роботи системи кількість фотографій для тренування збільшено до 8000 супутникових знімків. Можна зазначити, що модель краще розпізнала дорогу на пересічній місцевості, виділивши ділянку дороги, якої немає на бінарній масці (рисунок 2.20). Однак інші передбачення залишилися майже без змін або навіть погіршилися. Наприклад, на деяких зображеннях зникли невеликі ділянки доріг, які розпізнала попередня модель (рисунок 2.21).

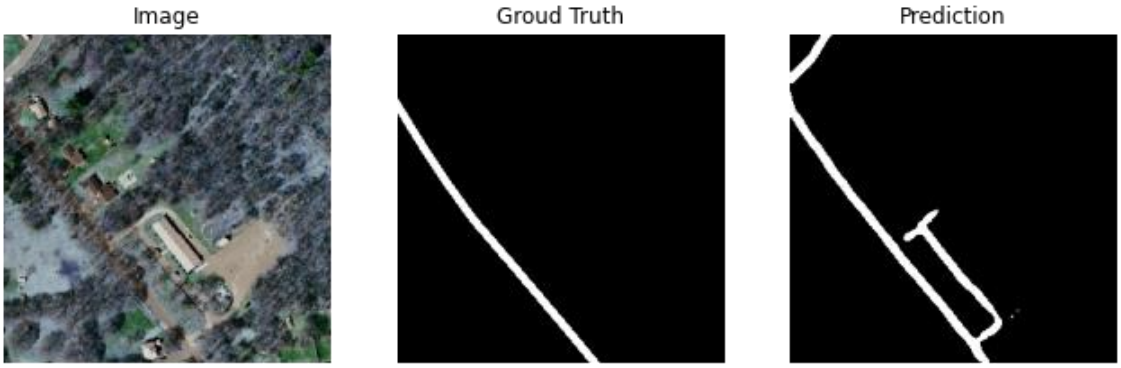

Рисунок 2.20 – Розпізнавання дороги на пересічній місцевості (8000 фото)

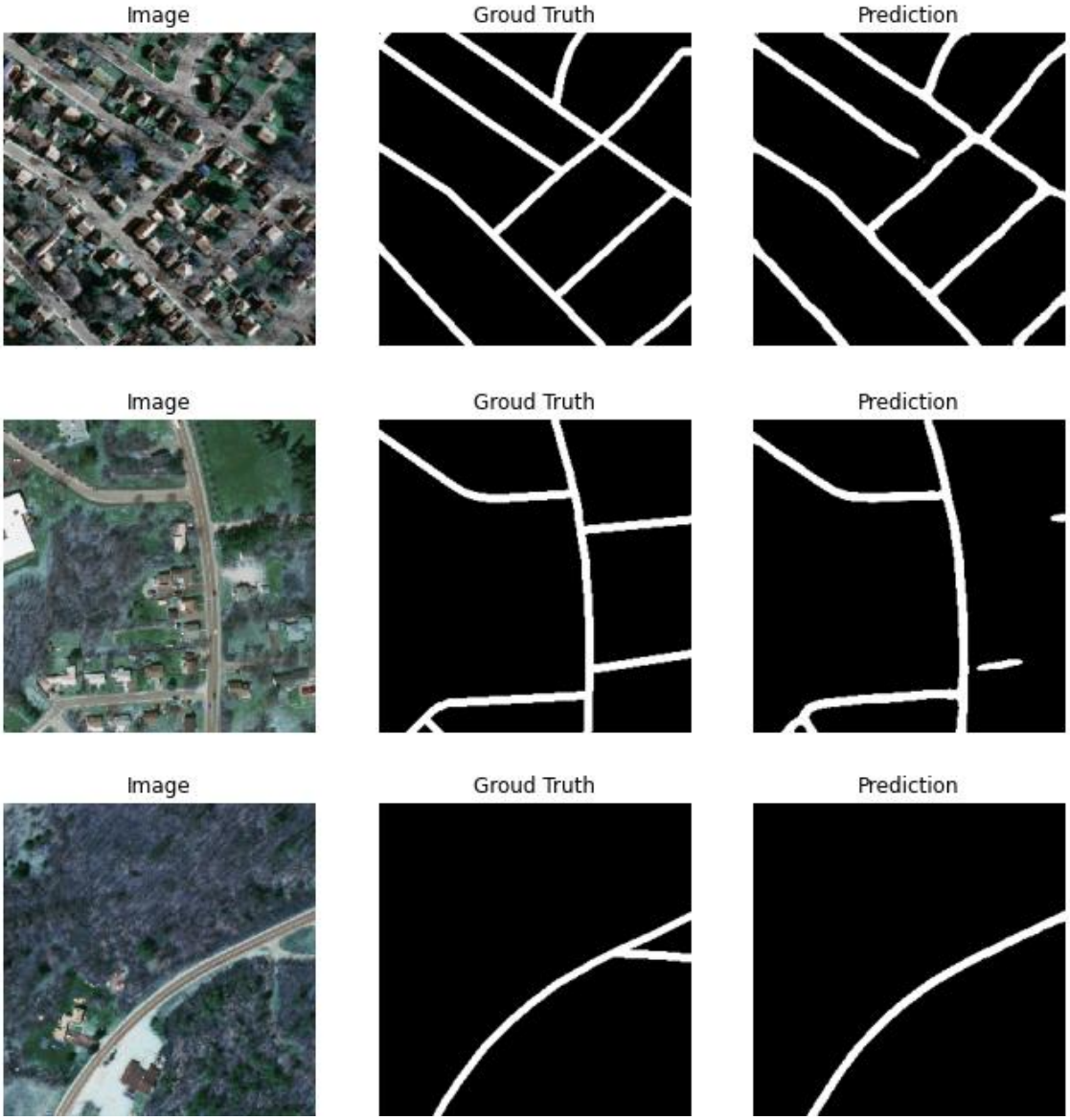

Рисунок 2.21 – Розпізнавання доріг (8000 фото)

2.9. Проблеми та перспективи для покращення

Розробка системи для розпізнавання доріг на супутникових зображеннях високої роздільної здатності пов’язана з великою кількістю труднощів. Однією з найбільших перешкод для вдалого тренування нейронної мережі та перевірки коректності її роботи є недосконаліть вхідних даних. Дуже серйозним недоліком є невідповідність між бінарними масками, що мають вказувати, на яких пікселях знаходяться дороги, та супутниковими знімками. Найпоширенішою проблемою є відсутність розмітки для деяких ділянок доріг, через що система розпізнає дороги там, де на еталонній масці їх немає, а на фото вони є. Однак метрики IOU та SDL розцінюють таку поведінку як помилкову, через що значення функції втрат зростає. Інша проблема пов’язана з вхідними даними – це різноманітні перешкоди, що тим чи іншим чином приховують ділянки доріг. Дерева та тіні від високих будинків перекривають частини вулиць, через що їх важко розпізнати. Через це утворюються «дірки» на бінарних картах у місцях, де не видно доріг.

Нажаль, проблему з некоректністю вхідних даних можна подолати лише вручну, перебравши всі фрагменти та вибравши лише ті, де розмітка доріг на бінарних масках відповідає дійсності. Впоратися з розпізнаванням вулиць, прихованих за механічними перешкодами теж не тривіальна задача. Можна збільшити кількість вхідних даних та надати більше прикладів доріг з різних міст, аби система з більшою вірогідністю розпізнавала дороги з різної місцевості. Інший варіант - розробити додатковий алгоритм, що буде з’єднувати розпізнані ділянки доріг, що знаходяться досить близько одна до одної. Але у такого підходу є ризик, що випадково з’єднаються частини, що мають стояти окремо. Отже, основними методами для покращення результатів роботи системи є збільшення об’єму вхідних даних, жорсткий відбір зображень та додаткова обробка передбачень.

Висновки

В результаті проведеної роботи було продемонстровано принцип роботи ЗНМ, зокрема U-Net, та приклад їх використання для розробки системи розпізнавання доріг на супутникових зображеннях високої роздільною здатності.

Розроблена нейронна модель демонструє позитивні результати після тривалого тренування. Програми для її запуску, тестування та підготовки вхідних даних мають гнучкі конфігурації для налаштування. Завдяки цьому вдалось провести ряд досліджень для виявлення залежності між кількістю зображень, використаних для тренування, показниками SDL, IOU та якістю передбачень системи.

У системи для розпізнавання супутникових зображень на основі нейронної мережі U-Net є деякі проблеми, пов’язані у першу чергу з необхідністю підбору великого масиву коректних вхідних даних для тренування моделі. Бінарні маски з неправильною розміткою доріг спричиняють неточності при підрахунку функції втрат: розпізнавання пікселів доріг, що не відмічені на масці, вважається хибним. Однак не зважаючи на це, є варіанти для покращення роботи системи, що передбачають більш жорстку фільтрацію вхідних даних, збільшення кількості фотографій та застосування алгоритмів покращення вихідних передбачень.

Система для розпізнавання доріг на супутникових зображеннях має деякі перспективи для подальшого розвитку. Можна змінити представлення результату, згенерувавши карту доріг у вигляді графу для кращого відображення результату. Також можна здійснювати пошук по такому графу для знаходження найкоротшого шляху між точками на карті. Це дозволить реалізувати основу для розробки системи навігації для автомобілів та інших видів наземного транспорту.

Список використаних джерел

1. Учасники проектів Вікімедіа. Сегментація зображення – Вікіпедія [Електронний ресурс] / Учасники проектів Вікімедіа // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Сегментація_зображення (дата звернення: 02.02.2021). – Назва з екрана.

2. What is computer vision? [Електронний ресурс] // IBM - Deutschland | IBM. – Режим доступу: https://www.ibm.com/topics/computer-vision (дата звернення: 02.02.2021). – Назва з екрана.

3. Basic Road Statistics of India [Електронний ресурс] – Режим доступу: https://morth.nic.in/sites/default/files/Basic%20_Road_Statics_of_India.pdf (дата звернення: 09.02.2021). – Назва з екрана.

4. Порогові методи (сегментація) – національна бібліотека ім. Н. Е. Баумана [Електронний ресурс] // Національна бібліотека ім. Н. Е. Баумана. – Режим доступу: https://ru.bmstu.wiki/Пороговые_методы_(Сегментация) (дата звернення: 09.02.2021). – Назва з екрана.

5. Чумаченко О. Глибокі нейронні мережі для вирішення завдань розпізнавання і класифікації зображення [Електронний ресурс] / Олена Чумаченко // ITCM-2021. – Режим доступу: http://itcm.comp-sc.if.ua/2017/Sineglazov.pdf (дата звернення: 09.02.2021). – Назва з екрана.

6. Учасники проектів Вікімедіа. Згорткова нейронна мережа – Вікіпедія [Електронний ресурс] / Учасники проектів Вікімедіа // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Згорткова_нейронна_мережа (дата звернення: 16.02.2021). – Назва з екрана.

7. Згорткові нейронні мережі – Вікіконспекти [Електронний ресурс] // Northern Eurasia Contests. – Режим доступу:

http://neerc.ifmo.ru/wiki/index.php?title=Сверточные_нейронные_сети (дата звернення: 16.02.2021). – Назва з екрана.

8. Behnke S. Evaluation of pooling operations in convolutional architectures for object recognition [Електронний ресурс] / Sven Behnke // University of Bonn, Computer Science VI, Autonomous Intelligent Systems. – Режим доступу: http://ais.uni-bonn.de/papers/icann2010_maxpool.pdf (дата звернення: 23.02.2021). – Назва з екрана.

9. Учасники проектів Вікімедіа. U-Net [Електронний ресурс] / Учасники проектів Вікімедіа // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/U-Net (дата звернення: 23.02.2021). – Назва з екрана.

10. Ronneberger O. U-Net: convolutional networks for biomedical image segmentation [Електронний ресурс] / Ronneberger Olaf // arXiv.org e-Print archive. – Режим доступу: https://arxiv.org/pdf/1505.04597.pdf (дата звернення: 02.03.2021). – Назва з екрана.

11. Muriuki G. Paper summary: U-Net: convolutional networks for biomedical image segmentation [Електронний ресурс] / Gerald Muriuki // Medium. – Режим доступу: https://towardsdatascience.com/paper-summary-u-net-convolutional-networks-for-biomedical-image-segmentation-13f4851ccc5e (дата звернення: 03.03.2021). – Назва з екрана.

12. Учасники проектів Вікімедіа. Keras [Електронний ресурс] / Учасники проектів Вікімедіа // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Keras (дата звернення: 19.03.2021). – Назва з екрана.

13. Учасники проектів Вікімедіа. Коефіцієнт Жаккара – вікіпедія [Електронний ресурс] / Учасники проектів Вікімедіа // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Коефіцієнт_Жаккара (дата звернення: 19.03.2021). – Назва з екрана.

14. Massachusetts roads dataset [Електронний ресурс] // Kaggle: Your Machine Learning and Data Science Community. – Режим доступу: https://www.kaggle.com/balraj98/massachusetts-roads-dataset (дата звернення: 25.03.2021). – Назва з екрана.

15. Chen B. A practical introduction to Keras callbacks in Tensorflow 2 [Електронний ресурс] / B. Chen // Medium. – Режим доступу: https://towardsdatascience.com/a-practical-introduction-to-keras-callbacks-in-tensorflow-2-705d0c584966 (дата звернення: 25.03.2021). – Назва з екрана.

16. Ba J. L. Adam: a method for stochastic optimization [Електронний ресурс] / Jimmy Lei Ba // arXiv.org e-Print archive. – Режим доступу: https://arxiv.org/pdf/1412.6980.pdf (дата звернення: 13.04.2021). – Назва з екрана.

Додаток А

Код для розбиття зображення на фрагменти

```
def crop_save(images_path, masks_path, result_images_path, result_masks_path, img_size):
    """
    Функція поділяє зображення на частини, фільтрує їх та зберігає фрагменти фото і масок на диску.

    Параметри:
    images_path -- шлях до підкаталогу зі знімками
    masks_path -- шлях до підкаталогу з масками
    result_images_path -- шлях для збереження знімків результату
    result_masks_path -- шлях для збереження масок результату
    img_size -- розмір фрагментів
    """
    dropped_num = 0
    saved_num = 0

    print(images_path)

    image_files = get_filenames(images_path +"*.tiff")
    print('Кількість файлів: {}'.format(len(image_files)))

    start_time = time.time()

    for image_filename in tqdm(image_files, total = len(image_files)):
```

```
    image = cv2.imread(images_path + image_filename)

    mask_path = masks_path + image_filename[:-1]
    mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)

    counter = 0

    for r in range(0, image.shape[0], img_size):
        for c in range(0, image.shape[1], img_size):
            counter += 1

            cropped_image, cropped_mask = crop(image, mask, r, r + img_size, c, c + img_size)
            cropped_mask[cropped_mask>1] = 255

            if np.any(cropped_mask):
                black_mask_num, white_mask_num = np.unique(cropped_mask, return_counts=True)[1]
                black_image_num = np.sum(cropped_image == 0)

                if white_mask_num/black_mask_num < 0.01 or black_image_num/cropped_image.size > 0.1:
                    dropped_num += 1
                else:
                    saved_num += 1
```

```
                cv2.imwrite(result_images_path + str(counter) + '_' + image_filename,
cropped_image)
                cv2.imwrite(result_masks_path + str(counter) + '_' + image_filename,
cropped_mask)
            else:
                dropped_num += 1


    print("Час: {}c.".format(round((time.time()-start_time), 2)))
    print("Пропущено зображень: ", dropped_num)
    print("Збережено зображень: ", saved_num)
    return saved_num
```

Додаток Б

Код для моделі нейронної мережі U-Net

```
inputs = Input((256, 256, 3))

conv1 = Conv2D(16, 3, activation=ACT_FUNCTION,
kernel_initializer=KERNEL_INIT, padding=PADDING_TYPE) (inputs)
conv1 = Conv2D(16, 3, activation=ACT_FUNCTION,
kernel_initializer=KERNEL_INIT, padding=PADDING_TYPE) (conv1)
pooling1 = MaxPooling2D() (conv1)

conv2 = Conv2D(32, 3, activation=ACT_FUNCTION,
kernel_initializer=KERNEL_INIT, padding=PADDING_TYPE) (pooling1)
conv2 = Conv2D(32, 3, activation=ACT_FUNCTION,
kernel_initializer=KERNEL_INIT, padding=PADDING_TYPE) (conv2)
pooling2 = MaxPooling2D() (conv2)

conv3 = Conv2D(64, 3, activation=ACT_FUNCTION,
kernel_initializer=KERNEL_INIT, padding=PADDING_TYPE) (pooling2)
conv3 = Conv2D(64, 3, activation=ACT_FUNCTION,
kernel_initializer=KERNEL_INIT, padding=PADDING_TYPE) (conv3)
pooling3 = MaxPooling2D() (conv3)

conv4 = Conv2D(128, 3, activation=ACT_FUNCTION,
kernel_initializer=KERNEL_INIT, padding=PADDING_TYPE) (pooling3)
conv4 = Conv2D(128, 3, activation=ACT_FUNCTION,
kernel_initializer=KERNEL_INIT, padding=PADDING_TYPE) (conv4)
```

```
        pooling4 = MaxPooling2D() (conv4)

        conv5 = Conv2D(256, 3, activation=ACT_FUNCTION,
kernel_initializer=KERNEL_INIT, padding=PADDING_TYPE) (pooling4)
        conv5 = Conv2D(256, 3, activation=ACT_FUNCTION,
kernel_initializer=KERNEL_INIT, padding=PADDING_TYPE) (conv5)

        upsample6 = Conv2DTranspose(128, 2, strides=(2,2),
padding=PADDING_TYPE) (conv5)
        upsample6 = concatenate([upsample6, conv4])
        conv6 = Conv2D(128, 3, activation=ACT_FUNCTION,
kernel_initializer=KERNEL_INIT, padding=PADDING_TYPE) (upsample6)
        conv6 = Conv2D(128, 3, activation=ACT_FUNCTION,
kernel_initializer=KERNEL_INIT, padding=PADDING_TYPE) (conv6)

        upsample7 = Conv2DTranspose(64, 2, strides=(2, 2),
padding=PADDING_TYPE) (conv6)
        upsample7 = concatenate([upsample7, conv3])
        conv7 = Conv2D(64, 3, activation=ACT_FUNCTION,
kernel_initializer=KERNEL_INIT, padding=PADDING_TYPE) (upsample7)
        conv7 = Conv2D(64, 3, activation=ACT_FUNCTION,
kernel_initializer=KERNEL_INIT, padding=PADDING_TYPE) (conv7)

        upsample8 = Conv2DTranspose(32, 2, strides=(2, 2),
padding=PADDING_TYPE) (conv7)
        upsample8 = concatenate([upsample8, conv2])
```

```
        conv8 = Conv2D(32, 3, activation=ACT_FUNCTION,
kernel_initializer=KERNEL_INIT, padding=PADDING_TYPE) (upsample8)
        conv8 = Conv2D(32, 3, activation=ACT_FUNCTION,
kernel_initializer=KERNEL_INIT, padding=PADDING_TYPE) (conv8)

        upsample9 = Conv2DTranspose(16, 2, strides=(2, 2),
padding=PADDING_TYPE) (conv8)
        upsample9 = concatenate([upsample9, conv1], axis=3)
        conv9 = Conv2D(16, 3, activation=ACT_FUNCTION,
kernel_initializer=KERNEL_INIT, padding=PADDING_TYPE) (upsample9)
        conv9 = Conv2D(16, 3, activation=ACT_FUNCTION,
kernel_initializer=KERNEL_INIT, padding=PADDING_TYPE) (conv9)

        outputs = Conv2D(1, 1, activation='sigmoid') (conv9)

        model = Model(inputs=[inputs], outputs=[outputs])
        model.summary()
```

Додаток В

Перелік прийнятих скорочень

SDL – Soft Dice Loss;

IOU – Intersection Over Union;

ЗНМ – згорткова нейронна мережа.