НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Олександр Коваль

«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення веб-технологій та мобільних пристроїв»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Аналіз супутникових знімків на основі семантичної сегментації»

Виконала:
студентка IV курсу, групи ТІ-62
Куцик Анастасія Ярославівна _____

Керівник:
асистент
Москаленко Юрій Володимирович _____

Рецензент:
к.т.н., старший викладач
Мажара Ольга Олександрівна _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення веб-технологій та мобільних пристроїв

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту
Куцик Анастасії Ярославівні
(прізвище, ім’я, по батькові)

1. Тема роботи «Аналіз супутникових знімків на основі семантичної сегментації»

керівник роботи асистент Москаленко Юрій Володимирович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020р. № **1168-с**

2. Строк подання студентом роботи ___ _____ 2020р.

3. Вихідні дані до роботи мови програмування: Python, Dart; бібліотеки: Keras, TensorFlow, NumPy, Matplotlib; фреймворки: Flask, Flutter.

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі алгоритми семантичної сегментації; побудувати алгоритм аналізу супутникових знімків на основі семантичної сегментації; практично реалізувати алгоритм аналізу супутникових знімків на основі семантичної сегментації; розробити та реалізувати додаток із зручним для користувача графічним інтерфейсом для демонстрації роботи системи; провести практичні дослідження роботи створеної системи.

5. Перелік ілюстративного матеріалу

Головна сторінка змагання; Приклади сегментації для одного зображення; Інформація про датасет; Оцінювання результатів; Аугментація; Відмінності між сегментацією та детекцією; Найпоширеніші моделі для сегментації; Покращена U-Net; Крос-валідація; TTA; Результати тестування різних моделей; Лідерборд змагання.

6. Дата видачі завдання ”11” жовтня 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	12.02.2020 – 18.03.2020	
2.	Вивчення та аналіз задачі	27.03.2020 – 03.04.2020	
3.	Розробка архітектури та загальної структури системи	09.04.2020 – 16.04.2020	
4.	Розробка структур окремих підсистем	17.04.2020 – 29.04.2020	
5.	Програмна реалізація системи	30.04.2020 – 12.05.2020	
6.	Оформлення пояснювальної записки	03.05.2020 – 05.06.2020	
7.	Захист програмного продукту	09.06.2020	
8.	Передзахист	09.06.2020	
9.	Захист	18.06.2020	

Студент _____ Куцик А. Я.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Москаленко Ю. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Мета дипломної роботи розробка системи для аналізу супутникових знімків на основі семантичної сегментації. Було розроблено та натреновано модель для аналізу знімків. Під час розробки було використано методи семантичної сегментації та згорткові нейронні мережі. Результати дослідження було представлено на вісімнадцятій міжнародній науково-практичній конференції.[1] Завдяки використанню зазначених методів було збільшено точність та швидкість аналізу. Також було розроблено користувацький інтерфейс, який дозволяє користувачеві провести аналіз знімку, та наочно представити працездатність системи. Записка містить 84 сторінки, 41 рисунок та 20 посилань.

ABSTRACT

The purpose of the thesis is to develop a system for the analysis of satellite images based on semantic segmentation. A model for image analysis was developed and trained. Semantic segmentation methods and convolutional neural networks were used during development. The results of the study were presented at the eighteenth international scientific-practical conference. [1] Due to the use of these methods, the accuracy and speed of analysis were increased. A user interface has also been developed that allows the user to analyze the image and visualize the system's performance. The note contains 84 pages, 41 pictures and 20 links.

ЗМІСТ

АНОТАЦІЯ 4

ABSTRACT 4

ЗМІСТ 5

ВСТУП 7

1. ЗАДАЧА АНАЛІЗУ ЗОБРАЖЕНЬ НА ОСНОВІ СЕМАНТИЧНОЇ СЕГМЕНТАЦІЇ 8

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ, ПОЗНАЧЕНЬ, ТЕРМІНІВ 9

2. СЕГМЕНТАЦІЯ. ОСНОВНІ МЕТОДИ СЕГМЕНТАЦІЇ 10

2.1 Основні методи сегментації 10

2.2 Нейромережеві методи сегментації 13

2.2.1 CNN для класифікації 13

2.2.2 CNN для сегментації 16

2.3 Висновки до розділу 22

3. СЕМАНТИЧНА СЕГМЕНТАЦІЯ. АЛГОРИТМИ ТА МЕТОДИ 23

3.1 U-Net 24

3.2 DeepLabv3 25

3.3 Об’єднання підходів для підвищення точності семантичної сегментації 26

3.4 PSPNet 27

3.5 Аналіз методів оцінки якості сегментації 27

3.6 Висновки до розділу 32

4. РЕАЛІЗАЦІЇ ТА АРХІТЕКТУРИ 33

4.1 Функції втрат 33

4.1.1 Dice Loss 33

4.1.2 Focal Loss 34
4.2.3 Tversky 36
4.2 Висновки до розділу 38
5. ОБЧИСЛЮВАЛЬНІ ЕКСПЕРИМЕНТИ 39
5.1 Набір даних, параметри навчання, крос-валідація 39
5.2 Аугментація 43
5.3 Аугментація під час тесту (TTA) 47
5.4 Використані моделі 47
5.5 Висновки до розділу 51
6. ПРОГРАМНА РЕАЛІЗАЦІЯ 53
6.1 Засоби розробки 53
6.2 Серверна частина 55
6.3 Користувацький інтерфейс 57
6.4 Висновки до розділу 59
ВИСНОВКИ 60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 61
ДОДАТОК А 63
ДОДАТОК Б 65
ДОДАТОК В 76

ВСТУП

Нейронні мережі набувають широкого поширення. Основними сферами застосування нейронних мереж є оптимізація, класифікація, розпізнавання, прогнозування та керування складними процесами.

Нейронні мережі набули широкого використання для аналізу зображень, адже в сучасному світі автоматичний аналіз зображень на базі алгоритмів машинного навчання поступово входить в усі сфери людської діяльності. Аналіз зображень на базі машинного навчання використовується для вирішення ряду досить складних задач: керування безпілотними автомобілями, розпізнавання об’єктів на камерах спостереження, для забезпечення безпеки, у військових системах та в медицині для виявлення аномалій.

Не менш важливим є завдання розпізнавання різних способів організації хмар. Зміна клімату вже багато років є однією з найбільш обговорюваних проблем. Дрібні хмари відіграють величезну роль у визначенні клімату Землі, але їх також досить важко зрозуміти та представити у кліматичних моделях. Існує багато видів організації хмар, але межі між різними формами організації є досить невиразними. Це ускладнює побудову традиційних алгоритмів, заснованих на правилах для розділення хмар. Однак, людське око справді добре виявляє особливості в організації, наприклад хмари, що схожі на квітку.

Класифікуючи різні типи організації хмар можна покращити наше фізичне розуміння цих хмар, що в свою чергу допоможе будувати кращі кліматичні моделі.

Метою даної роботи є описання та реалізація алгоритму для аналізу супутникових знімків методом семантичної сегментації зображень.

1. ЗАДАЧА АНАЛІЗУ ЗОБРАЖЕНЬ НА ОСНОВІ СЕМАНТИЧНОЇ СЕГМЕНТАЦІЇ

Метою даної роботи є розробка системи для аналізу супутникових знімків методом семантичної сегментації зображень.

Виділяємо такі підзадачі:

- аналіз існуючих алгоритмів семантичної сегментації;
- побудова алгоритму аналізу супутникових знімків на основі семантичної сегментації;
- практична реалізація алгоритму аналізу супутникових знімків на основі семантичної сегментації;
- розробка та реалізація додатку із зручним для користувача графічним інтерфейсом для демонстрації роботи системи;
- проведення практичних досліджень роботи створеної системи;

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ, ПОЗНАЧЕНЬ, ТЕРМІНІВ

CNN — згорткова нейронна мережа.

Кластер — об’єднання однорідних елементів, яке може розглядатися як самостійна одиниця.

RNN — рекурентна нейрона мережа.

gradient boosting — це алгоритм машинного навчання для проблем регресії та класифікації.

random forest — є алгоритмом класифікації, що складається з багатьох дерев рішень.

SVM (support vector machine) — є моделлю навчання з учителем, з пов'язаними алгоритмами навчання, які аналізують дані, що використовуються для класифікації та регресійного аналізу

LeNet, AlexNet, VGG, ResNet — моделі CNN для класифікації.

SegNet, U-Net, PSPNet, FPN, DeepLab, DeepLabv3, DeepLabv3+ — моделі CNN для сегментації.

IoU (Intersection over Union) — метрика для оцінки якості сегментації.

FP — False positive.

TP — True positive.

FN — False negative.

Dice — метрика для оцінки якості сегментації.

BCE (Binary Cross-Entropy) — метрика для оцінки якості сегментації.

2. СЕГМЕНТАЦІЯ. ОСНОВНІ МЕТОДИ СЕГМЕНТАЦІЇ

2.1 Основні методи сегментації

Є дві основні задачі аналізу зображень нейронними мережами: класифікація та сегментація.

Сегментація — задача розбиття зображення на менші частини (сегменти). Завдяки такому розбиттю вдається спростити представлення та вигляд зображення, в результаті чого проведення аналізу стає простішим.

На даний момент існують такі методи сегментації:

- Кластеризація;
- З використанням гістограм;
- На основі моделі;
- Нарощування областей;
- Виділення країв;
- Багатомасштабна;
- На основі графу;
- Водорозподілу;

Графічне представлення методів сегментації наведено на рисунку 2.1.

Кластеризацію найчастіше представлено методом k-середніх. Метод базується на розподіленні спостережень між кластерами з найближчим середнім значенням. Основна ідея методу — мінімізація суми квадратів відхилення між спостереженням та центром кластера до якого це спостереження відноситься.

Метод гістограм має свою перевагу в тому, що є можливість зменшити час виконання, адже для роботи цього методу потрібно один раз проходитися по пікселям. Гістограма обчислюється завдяки знаходженню максимуму і мінімуму кожного пікселя, які потім об’єднуються у кластери.

В наступному методі за основу береться різниця в яскравості на краях області, для виділення окремих зон. Саме завдяки такій особливості даний спосіб називається методом виділення країв. Але також цей метод має певні недоліки, серед яких те, що не завжди вдається точно виділити границі, в результаті чого немає можливості виокремити об’єкти. В статті [3] автор наводить приклад роботи алгоритмів Sobel, Prewitt та Roberts. В алгоритмі Prewitt [4] використовується різниця в інтенсивності різних частин зображення і цей алгоритм добре підходить для виділення вертикальних та горизонтальних границь.

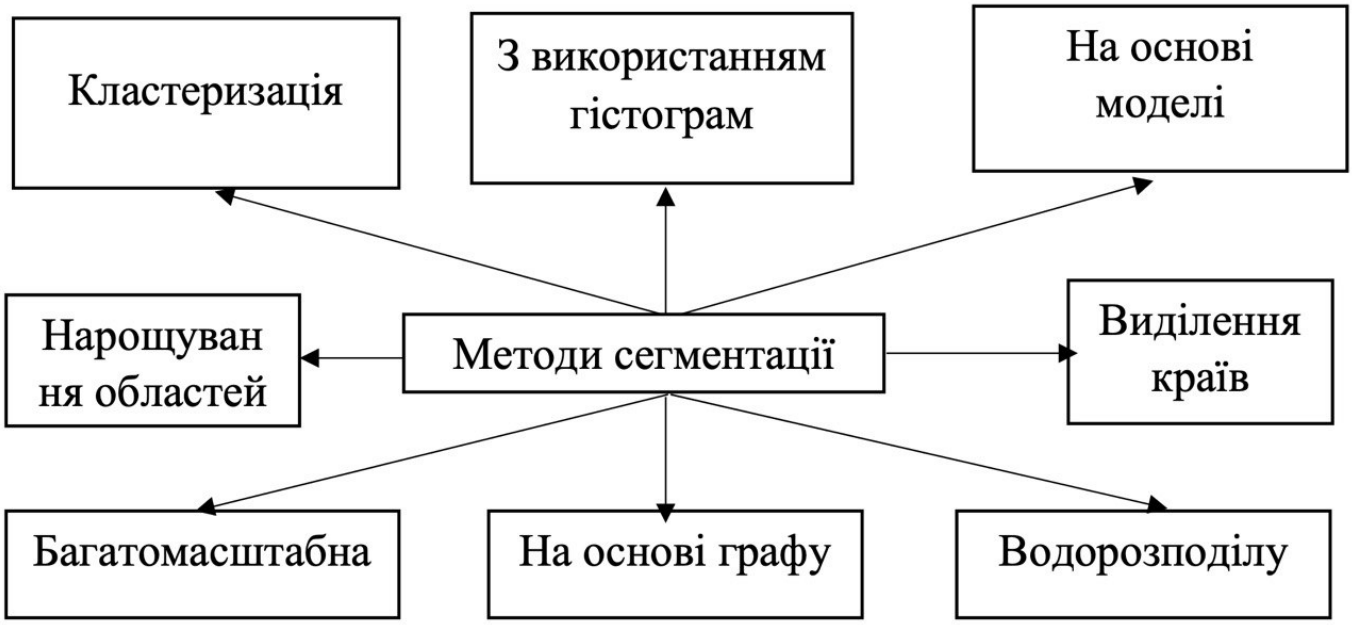

Рисунок 2.1 — Методи сегментації зображень

Метод розрізу графа є досить ефективним для задач сегментації зображень. Суть метода полягає в тому, що зображення представляється у вигляді зваженого, неорієнтовного графа. Пікселі, або об’єднання пікселів асоціюються вершиною, а ваги ребер визначаються (не) аналогічністю пікселів, які знаходяться поряд. Дуже часто під час дослідження алгоритмів на базі графів виділяють три основні типи: швидкісний граф, інтерактивний граф та сегментація за попередньо виділеними областями. Саме такі типи автор виділяє в дослідженні [5].

Метод водоподілу має в своїй основі виділення областей з використанням методів математичної морфології. Назва даного методу бере свої витоки з географічної термінології, де водорозділ — це умовна топографічна лінія на земній поверхні , яка розділяє басейни двох, або більше річок, озер, морів, або океанів, направляючи стік атмосферних опадів двома протилежними схилами. Якщо по аналогії розглядати зображення, як земну поверхню, то можна так само виділити лінії водоподілу, які будуть розмежовувати різні області зображення. Основним мінусом даного методу є те що відбувається надмірна сегментація, а оскільки усі шуми подаються через градієнт то відбувається значне спотворення результатів і зрештою виникає потреба проводити видалення шумів. Видалення шумів проводиться в декілька етапів, перш за все шуми видаляються в початковому зображенні за допомогою використання морфологічних операцій закриття або розкриття. На другому етапі обчислюється морфологічний градієнт зображення без шуму і виконується нелінійне перетворення для рівнів сірого на градієнті зображення на основі принципу Вебера, на кінцевому етапі етап — обчислення водоподілу по нелінійному, розбитому на області, градієнтному зображенню. [6]

Є досить велика кількість методів нарощування областей. Найпростішим є поточкове нарощування. Для реалізації методів поточкового нарощування використовуються функції Байеса, функції енергії, а також властивості фракталів та апарат нейронних мереж.

Метод на основі моделі базується на припущені, що об’єкти, які нас цікавлять мають повторювані геометричні форми. Виходячи з цього припущення можна знайти модель, яка буде вважатися найбільш ймовірною для обґрунтування змін форми, після чого під час подальшої сегментації будуть накладатися обмеження, вважаючи цю модель апріорною. Таке завдання включає в себе:

- приведення тренувальних прикладів до загального положення;
- ймовірнісне представлення змін наведених зразків;
- статистичний висновок для моделі і зображення.

Сегментація може використовуватися як в великих так і дрібних масштабах..

Критерій сегментації не завжди можна назвати об’єктивним, адже він може бути надмірно складним і до уваги можуть прийматися не лише локальні, а й глобальні критерії. Загальною вимогою є лише те, що області повинні бути пов’язані між собою.

Але всі ці методи не надають достатньої точності розпізнавання. Найбільш точними на даний момент є нейромережеві методи сегментації.

2.2 Нейромережеві методи сегментації

Згорткова нейронна мережа (CNN) — нейронна мережа прямого поширення, що містить згорткові шари [7]. В 1988 році Ян Лекун запропонував перший варіант згорткової нейронної мережі. Спершу було проведено дослідження на невеликих вибірках зображень під час розпізнавання рукописного вводу. Пізніше з’явилися більш потужні обчислювальні потужності, а також з’явилося більше вибірок з більшою кількість даних, що стало поштовхом для стрімкого розвитку згорткових нейронних мереж. Згорткові нейронні мережі почали показувати набагато кращі результати на задачах класифікації, випереджаючи усі попередні розробки. Також було встановлено, що згорткові нейронні мережі показують вражаючі результати при вирішенні задач комп’ютерного зору.

2.2.1 CNN для класифікації

LeNet-5

LeNet-5 була новаторською 7-ступінчастою мережею. Ян Лекун запропонував її в 1998 році для розпізнавання символів. Її було застосовано декількома банками для розпізнавання рукописних цифр на чеках, які було відцифровано у форматі 32x32 пікселі у відтінках сірого. Для можливості обробки зображень з більшою роздільною здатністю потрібні більші та більш згорнуті шари, тому ця методика обмежується наявністю обчислювальних ресурсів.

На рисунку 2.2 продемонстровано архітектуру даної мережі.

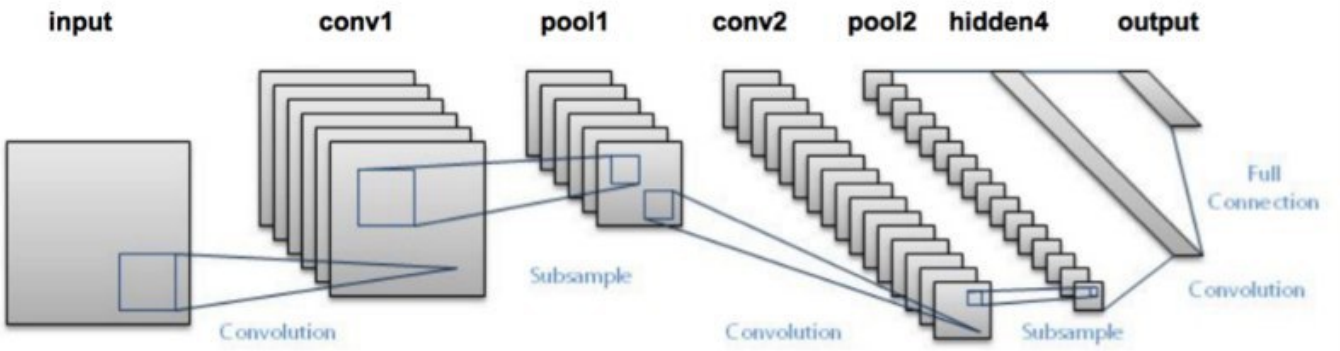

Рисунок 2.2 — Архітектура мережі LeNet

AlexNet

У 2012 році AlexNet значно перевершив усіх попередніх конкурентів і зменшив показник помилки з 26% до 15,3%. Мережа мала дуже схожу архітектуру з LeNet, але була глибшою, з більшою кількістю фільтрів на шарах та зі складеними згортковими шарами. AlexNet складається з 11x11, 5x5,3x3, згорткок, максимальне пулінгом, аугментацією, активацією ReLU, та SGD з імпульсом. Він додає активацію ReLU після кожного згорткового і повністю пов'язаного шару.(Рисунок 2.3) AlexNet навчався 6 днів одночасно на двох графічних процесорах Nvidia Geforce GTX 580, що є причиною того, що їх мережа розбита на два конвеєри. AlexNet був розроблений групою SuperVision, до складу якої входили Алекс Крижевський, Джеффрі Хінтон та Ілля Суцкевер.

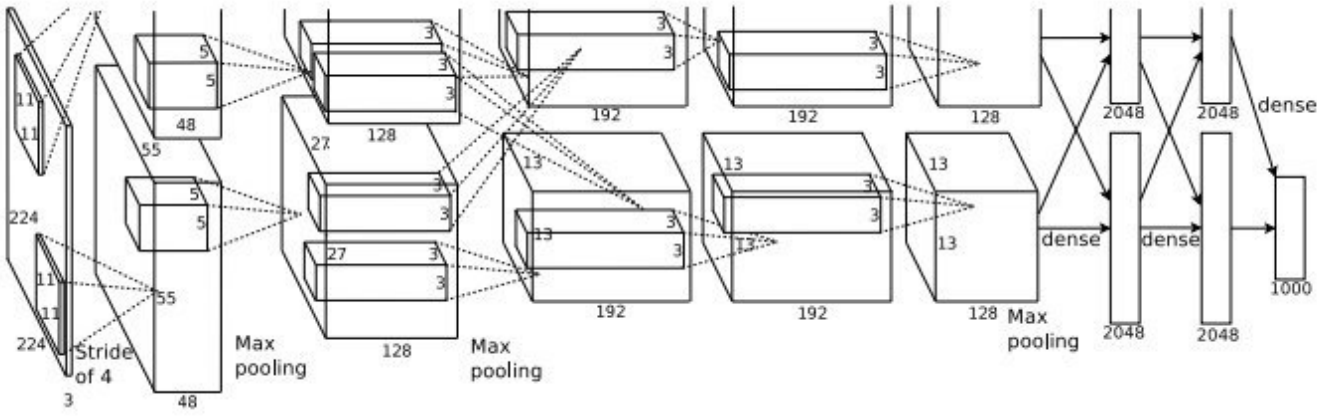

Рисунок 2.3 — Архітектура мережі AlexNet

VGGNet

VGGNet складається з 16 згорткових шарів і дуже привабливий через свою дуже єдину архітектуру. Подібно до AlexNet, лише 3x3 згортки, але багато фільтрів. Навчався на 4 GPU протягом 2-3 тижнів. Наразі це найкращий вибір для отримання функцій із зображень. Вагова конфігурація VGGNet є загальнодоступною і використовувалася в багатьох інших програмах і завданнях як базовий екстрактор функцій. Однак VGGNet складається з 138 мільйонів параметрів, з чим може бути досить складно впоратися.

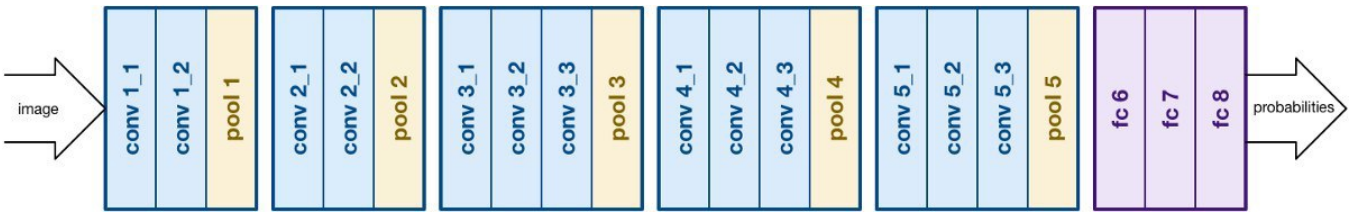

Рисунок 2.4 — Архітектура мережі VGGNet

ResNet

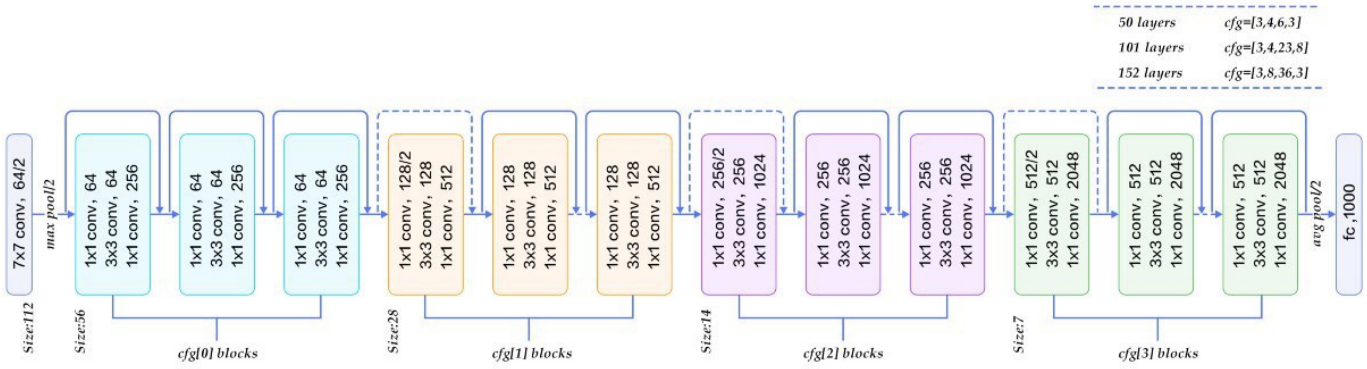

Рисунок 2.5 — Архітектура мрежі ResNet

Так звана Залишкова нейронна мережа (ResNet) Запровадила архітектуру anovel із "пропускними з'єднаннями" та має велику нормальну пакетну нормалізацію. Такі пропускні з'єднання також відомі як одиниці з закритим типом або періодично закриті одиниці і мають сильну схожість з останніми успішними елементами, застосованими в RNN. Завдяки цій техніці вони змогли тренувати мережу із 152 шарами, зберігаючи при цьому меншу складність, ніж VGGNet. ResNet вдалося покращити відсоток

помилки до 3,57%, що перевищує ефективність людського рівня на цьому наборі даних.

2.2.2 CNN для сегментації

SegNet

SegNet — нейронна мережа з повністю згортковою архітектурою для піксельної сегментації зображень. Цей основний механізм сегментування складається з мережі енкодера, відповідної мережі декодера з наступним піксельним рівнем класифікації. Архітектура мережі енкодерів топологічно ідентична 13 згортковим шарам у мережі VGG16. Роль мережі декодерів полягає у зіставлянні мап функцій енкодера низької роздільної здатності на картках з повною вхідною роздільною здатністю для піксельної класифікації. Новизна SegNet полягає в тому, як декодер покращує вибір карти вхідних зображень нижчої роздільної здатності. Зокрема, декодер використовує індекси об'єднання, обчислені на етапі максимального об'єднання відповідного енкодера, для здійснення нелінійного підсилення.

SegNet в першу чергу мотивувалас програмами для розуміння сцени. Отже, він розроблений таким чином, щоб бути ефективним як з точки зору пам'яті, так і з розрахункового час. Вона також має значно меншу кількість параметрів, ніж інші конкуруючі архітектури. [8]

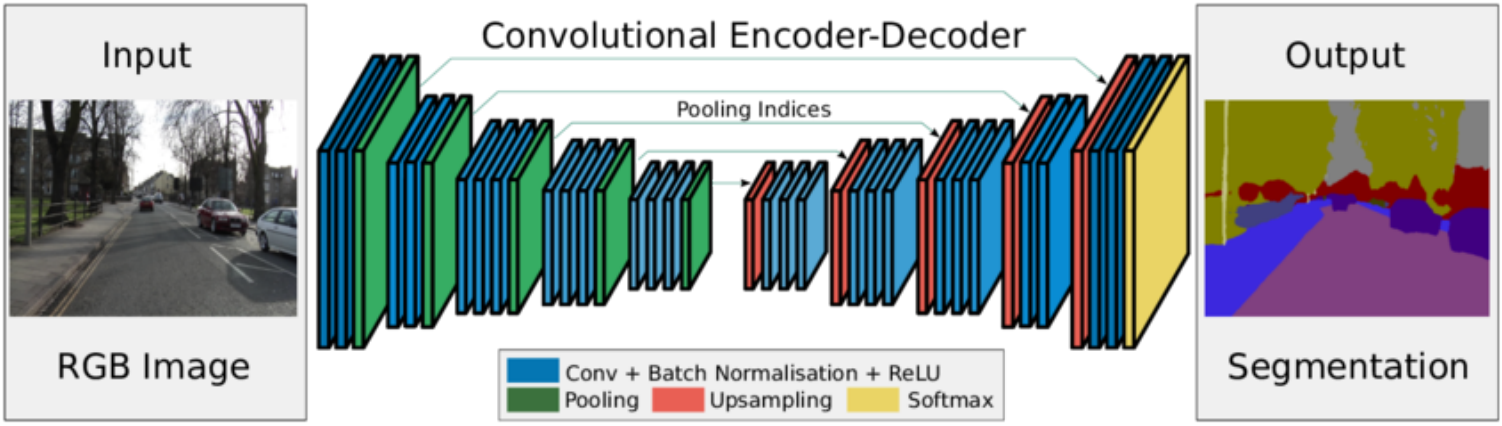

Рисунок 2.6 — Архітектура SegNet

U-Net

Автори створили мережу під назвою U-net, складену з двох частин: підрядна частина для обчислення функцій та розширювана частина для просторової локалізації патернів у зображенні. Демонстраційна частина має функції FCN-подібного вилучення архітектури зі згортками 3x3. Обрізані карти функцій із нижчої частини мережі, що копіюються, скопіюються всередину розгорнутої частини, щоб уникнути втрати інформації про патерн. Нарешті, згортка 1x1 обробляє карти функцій, щоб генерувати карту сегментації і таким чином класифікувати кожен піксель вхідного зображення. Архітектура U-net отримала широке поширення в останніх роботах (FPN, PSPNet, DeepLabv3 тощо).

Зауважте, що вона не використовує жодного повністю пов'язаного шару. Як наслідок, кількість параметрів моделі зменшується, і її можна навчати за допомогою невеликого набору розмічених даних (використовуючи відповідне збільшення даних). Наприклад, автори використовували відкритий набір даних із 30 зображеннями для тренувань під час своїх експериментів.

PSPNet

X. Чжао та ін. в 2016 розробили мережу Pyramid Scene Parsing (PSPNet), щоб краще вивчити глобальне контекстне представлення сцени. Шаблони витягуються із вхідного зображення за допомогою витяжки функцій з розширеною мережевою стратегією. Карти функцій подають модуль об'єднання пірамід для розрізнення шаблонів з різними масштабами. Вони об'єднані чотирма різними масштабами, кожна з яких відповідає рівню піраміди, і обробляються згортковим шаром 1x1 для зменшення їх розмірів. Таким чином кожен рівень піраміди аналізує області зображення з різним розташуванням. Виходи рівнів піраміди збільшені до вибірки та з'єднані з картами первинних ознак, щоб остаточно містити локальну та глобальну інформацію про контекст. Потім вони обробляються згортковим шаром для створення піксельних прогнозів. Найкращий PSPNet з попередньо перевіреним

ResNet (використовуючи набір даних COCO) досяг 85,4% mIoU за результатами сегментації PASCAL VOC 2012 року.

FPN

Feature Pyramid Network (FPN) використовується в системах виявлення об'єктів або сегментації зображень. Її архітектура складається із шляху знизу вгору, шляху зверху вниз та бічних з'єднань для того, щоб приєднати функції низької та високої роздільної здатності. Шлях знизу вгору приймає зображення з довільним розміром як вхід. Він обробляється згортковими шарами і складається з об'єднаних шарів. Зауважимо, що кожен пучок карт функцій з однаковим розміром називається етапом, виводи останнього шару кожного етапу — це функції, які використовуються для рівня піраміди. Шлях зверху вниз полягає в тому, щоб збільшити карти останніх функцій з відхиленням, одночасно покращуючи їх картами зображень з тієї ж стадії шляху "знизу вгору" за допомогою бічних з'єднань. Ці з'єднання полягають у об'єднанні карт функцій шляху "знизу вгору", обробленого згорткою 1x1 (для зменшення їх розмірів) з картами функцій шляху зверху вниз.

DeepLab

DeepLab було створено надихаючись моделлю FPN. На даний момент DeepLab вважається однією з найкращих моделей для задач сегментації. Ця модель поєднує жорстку згортку, просторове об'єднання пірамід та повністю пов'язані CRF. Модель, представлена в цьому документі, також називається DeepLabv2, оскільки вона є коригуванням початкової моделі DeepLab (деталі про первісну не будуть надані, щоб уникнути надмірності). На думку авторів, послідовне об'єднання максимумів та крокування знижує роздільну здатність карт функцій у глибоких нейронних мережах. Вони ввели жорстку згортку, яка в основному є розширеною згорткою. Модель складається з фільтрів, орієнтованих на розріджені пікселі з фіксованою швидкістю. Наприклад, якщо частота дорівнює 2, фільтр орієнтується на один піксель над двома

на вході; якщо коефіцієнт, рівний 1, жорстка згортка є базовою згорткою. Жорстка згортка дозволяє захоплювати кілька об'єктів масштабу.

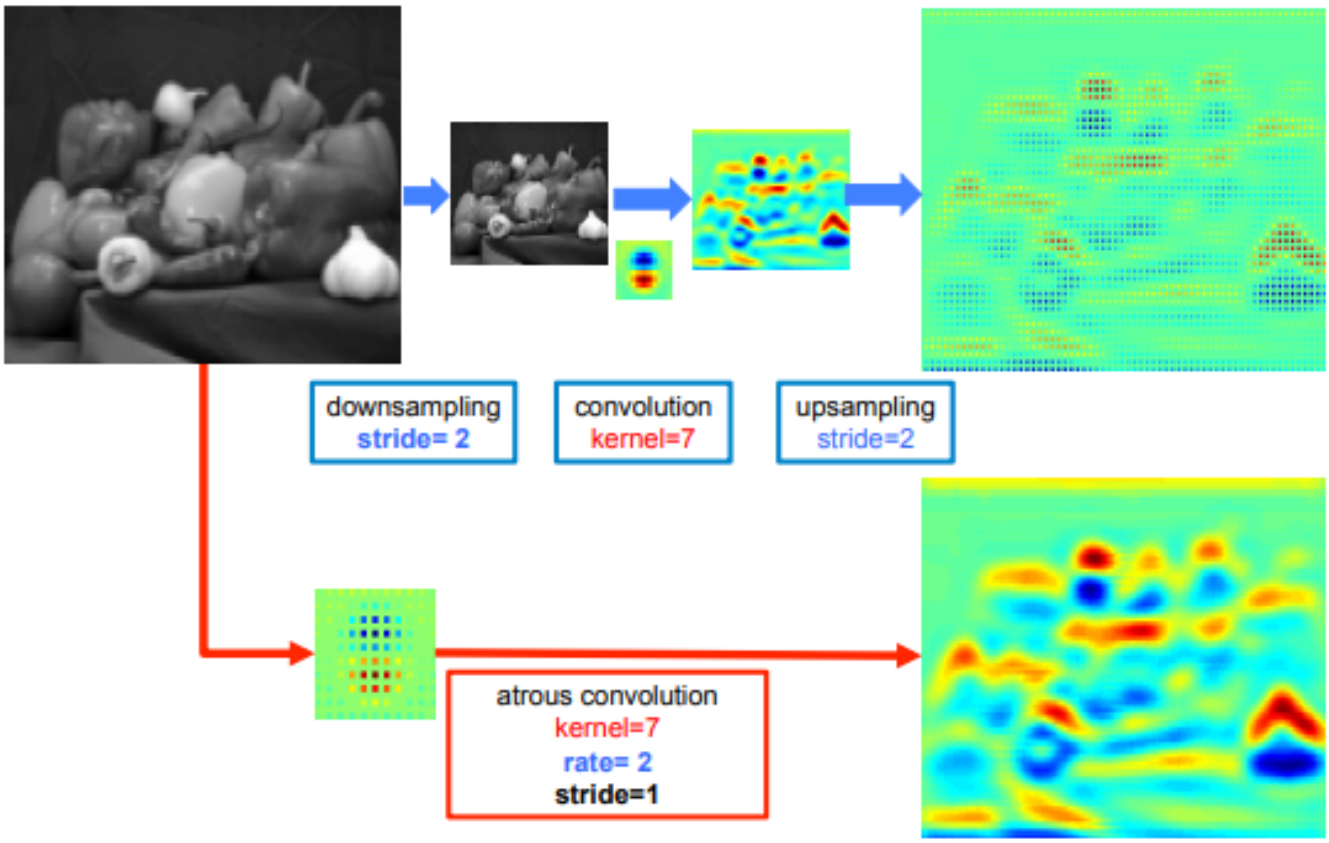

Рисунок 2.7 — Порівняння застосування звичайних згорток та жорстких згорток

The Atrous Spatial Pyramid Pooling (ASPP) полягає в застосуванні декількох жорстких згорток одного і того ж входу з різною швидкістю для виявлення

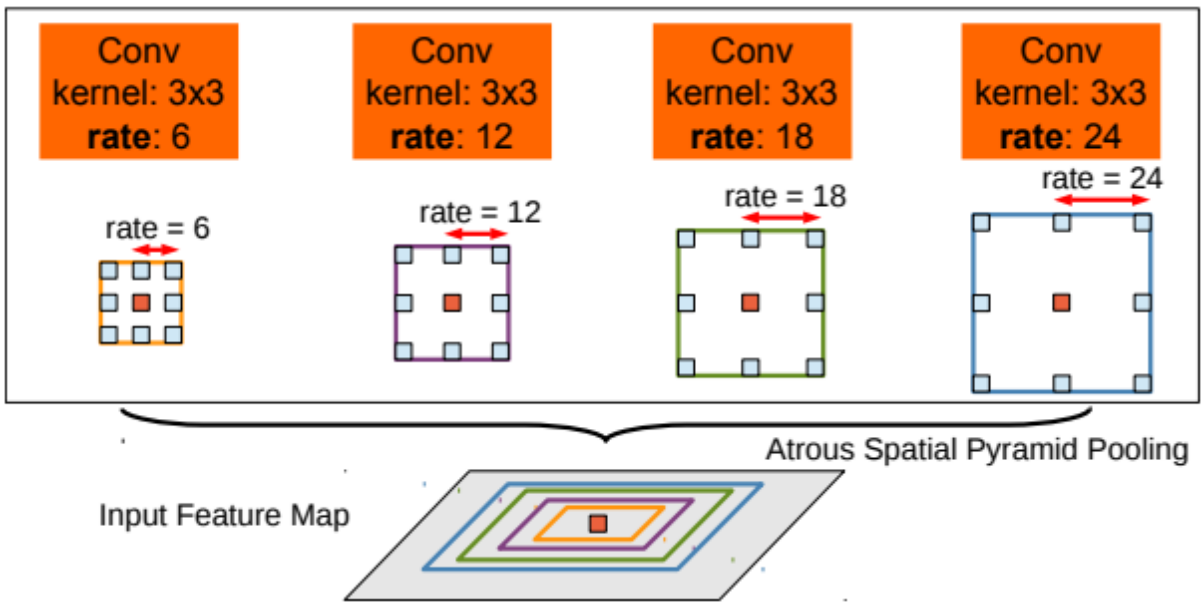

Рисунок 2.8 — The Atrous Spatial Pyramid Pooling

просторових закономірностей. Карти функцій обробляються в окремих гілках і об'єднуються за допомогою білінеарної інтерполяції для відновлення вихідного розміру вводу. Вихід подає повністю пов'язане умовне випадкове поле (CRF) обчислюючи межі між характеристиками та довгими термозалежностями для отримання семантичної сегментації.

Найкраща мережа DeepLab, що використовує ResNet-101, в якості основної системи, досягла 79,7% mIoU балів у виклику PASCAL VOC 2012.

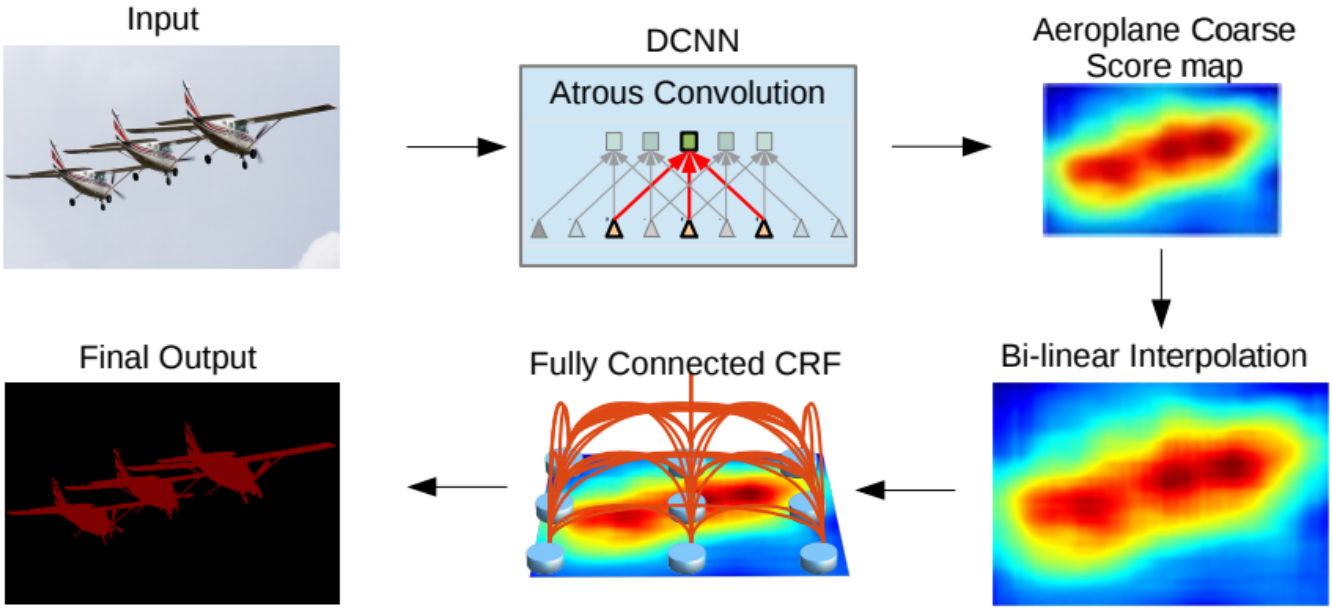

Рисунок 2.9 — Мережа DeepLab

DeepLabv3

В 2017 було переглянуто DeepLab, щоб створити DeepLabv3, що поєднує каскадні та паралельні модулі жорстких згорток. Автори модифікували архітектуру ResNet, щоб зберегти карти зображень високої роздільної здатності у глибоких блоках, використовуючи жорсткі згортки.

DeepLabv3+

В 2018 нарешті було випущено модель DeepLabv3+ з використанням структури енкодера-декодера. Автори впровадили жорстку відокремлювану згортку, що

складається з глибинної згортки (просторова згортка для кожного канала введення) та точкової згортки (1x1 згортка з глибинною згорткою як вхід).

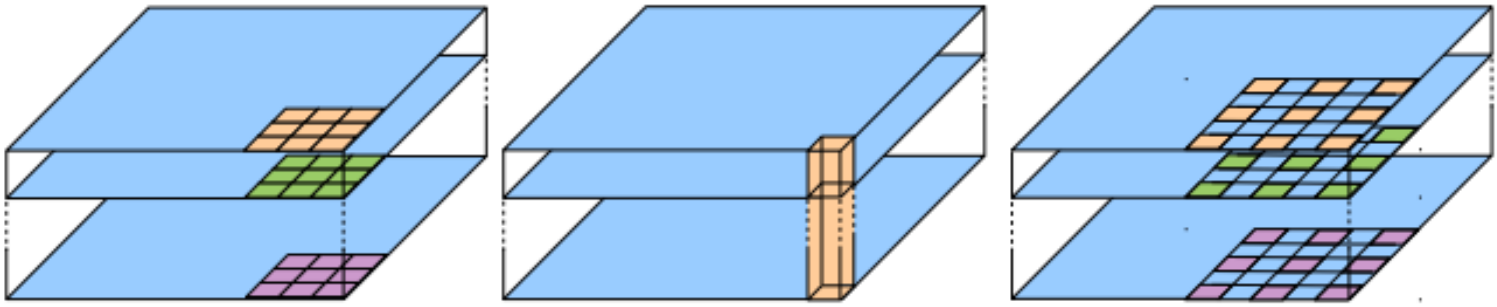

Рисунок 2.10 — Поєднання глибинної згортки та точкової згортки для генерації жорсткої відокремлюваної згортки

Вони використовували модель DeepLabv3 в якості енкодера. Найефективніша модель має модифіковану магістраль Xception з більшою кількістю шарів, жорсткими глибинними роздільними згортками, замість максимального пулінгу та нормалізації партії. Виходи ASPP обробляються згорткою 1x1 і збільшені з коефіцієнтом 4. Виходи ланцюга енкодера мають за основу CNN і також обробляються ще однією згорткою 1x1 і приєднуються до попередніх. Карти функцій подають два згорткових шари 3x3, а виходи збільшені на коефіцієнт 4, щоб створити остаточне сегментоване зображення.

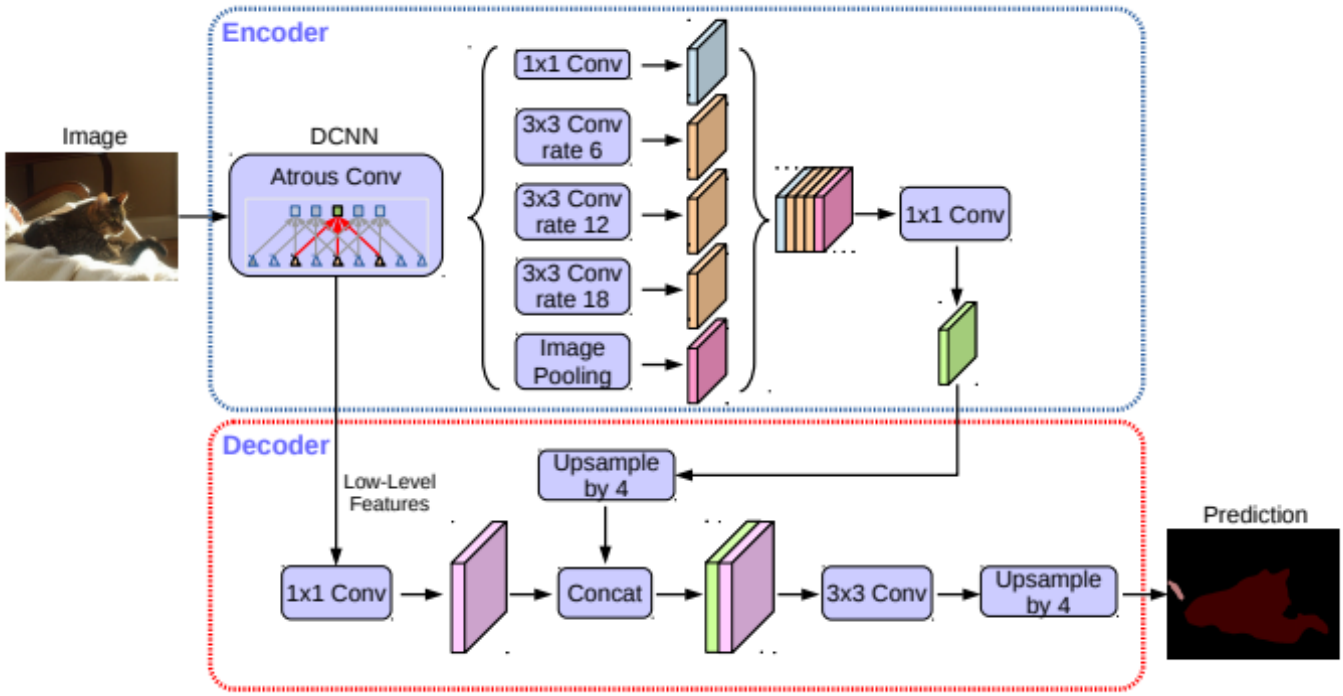

Рисунок 2.11 — Архітектура мережі DeepLabv3+

2.3 Висновки до розділу

В цьому розділі було дано визначення сегментації, а також було розглянуто основні методи сементації. Представлені методи мають як свої переваги так і недоліки.

Також було описано використання згорткових нейронних мереж для задач класифікації. Було розглянуто моделі: LeNet, AlexNet, VGG, ResNet. Представлено хронологічний розвиток моделей для класифікації, описано їх архутектурні особливості, а також наведено схематичні рисунки, для більш наочного представлення.

Моделі для сегментації також було представлено у цьому розділі. Було розглянуто моделі: SegNet, U-Net, PSPNet, FPN, DeepLab, DeepLabv3, DeepLabv3+. Дані моделі було також описано в хронологічному порядку, а також в порядку покращення їх показників точності. Наведено ілюстративні матеріали, а також перелік архутектурних особливостей.

Виходячи з представленої інформації вдалося відслідкувати розвиток згорткових нейронних мереж для задач класифікації та сегментації, а також зрозуміти, які моделі вважаються більш ефективними.

3. СЕМАНТИЧНА СЕГМЕНТАЦІЯ. АЛГОРИТМИ ТА МЕТОДИ

Семантична сегментація зображень набула щирокого застосування в задачах комп'ютерного зору, зокрема в розпізнаванні образів, розуміння контексту, аналіз вмісту зображення та багатьох інших.

Основна задача семантичної сегментації полягає в тому щоб класифікувати кожен піксель зображення, для розуміння контексту в цілому. Задача сегментації є більш простою, адже є необхідність лише визначити однорідні області зображення, а в задачах семантичної сегментації необхідно ще визначити який клас об'єктів, який ці області визначають. Досить часто задачу семаничної сегментації порівнюють з задачею класифікації, але класифікація теж є більш простою, оскільки в семантичній сегментації окрім класу необхідно ще встановити точне просторове представлення об'єкту, його структури та частин.

Є декілька випадків задач семантичної сегментації. В одному випадку задача полягає в тому, щоб класифікувати кожен піксель окремо. В другому випадку виникає потреба встановлювати класи областей виділених на зображенні.

Вперше вдалося досягти позитивних результатів при розробці алгоритмів семантичної сегментації, тренуючи на класифікатори на певних строго визначених характеристиках. Прикладами таких класифікаторів є gradient boosting, random forest, SVM, та інші. Не зважаючи на те, що для тренування наведених класифікаторів використовували досить специфічні техніки такі як структуроване передбачення та врахування контексту, все ж існувало обмеження у вигляді недостатньої виразності та інформативності виділених характеристик.

Після того як почали використовувати методи та алгоритми машинного навчання все ж таки вдалося досягти певних успіхів в збільшенні якості результатів виконная семантичної сегментації.

В даній роботі було використано методи, які базуються на технології згорткових нейронних мереж. Згорткові нейронні мережі зробили прорив в задачах семантичної сегментації і зрештою набули широкого використання в задачах комп’ютерного зору. В роботі застосовувалися методи: U-Net, DeepLabv3, PSPNet.

3.1 U-Net

U-Net — згорткова нейронна мережа, яку була розроблено для сегментації біомедичних зображень на факультеті комп'ютерних наук університету Фрайбурга, Німеччина [10]. U-Net було запропоновано в 2015 році [10]. Мережа має U-подібну архутектуру та складається з двох частин: звуження та розширення. Звуження представляється у вигляді згорткової нейронної мережі, до складу якої входять повторні згортки, після кожної згортки використовується активації ReLU і операція макспулінга. В процесі звуження інформація про ознаки збільшується, в той час, як просторова інформація зменшується. Коли відбувається розширення, завдяки згорткам, які збільшують розмірність, а також конкатенації з ознаками, які вирізняються високою розподільною здатністю на шлюху звуження, відбувається процес поєднання просторової інформації та ознак.[10]. Прямі-з'єднання (Skip-з'єднання) між блоками енкодера та декодера дозволяють збільшити точність мережі. На рисунку 3.1 представлено класичну архітектуру U-Net мережі.

U-Net широко використовується в біомедичній сегментації зображень, наприклад сегментація зображення мозку і сегментація зображення печінки та сегментації нервової системи. Нижче наведено кілька варіантів і застосунків U-Net:

- Піксельна регресія з використанням U-Net та її застосування для підвищення якості[11];
- 3D U-Net: навчання щільної об'ємної сегментації у випадку, коли розмічені дані рідко зустрічаються[12];
- TernausNet: U-Net з енкодером VGG11, попередньо тренованим на ImageNet для сегментації зображень[13].

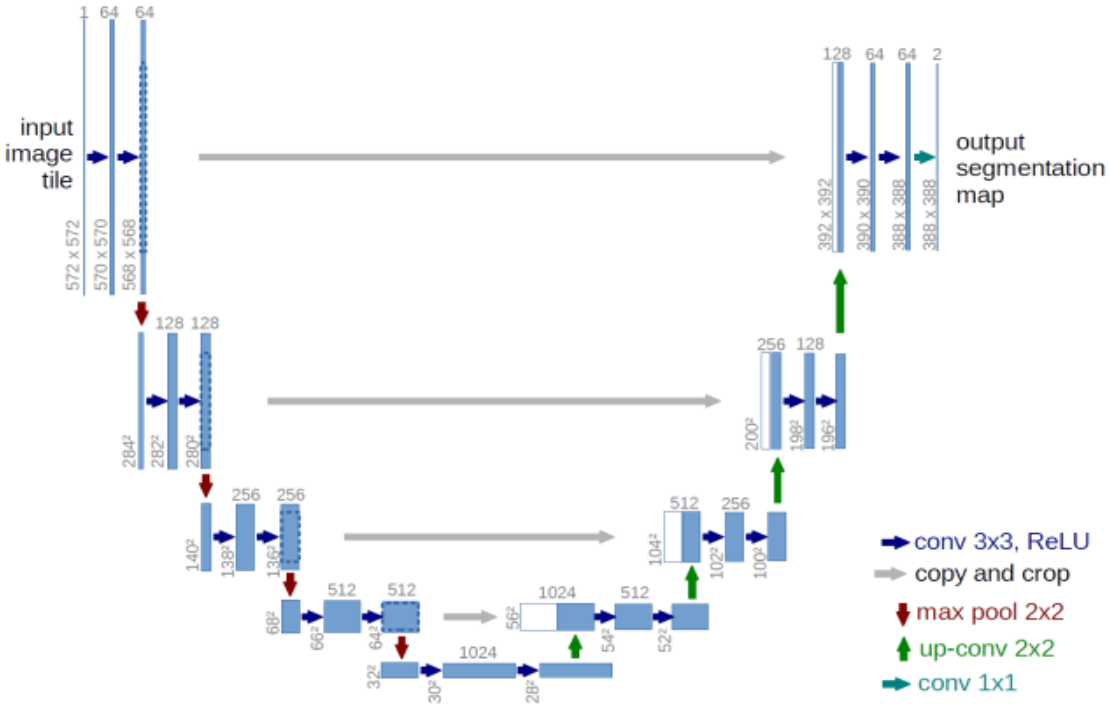

Рисунок 3.1 — Архітектура U-Net [10]

3.2 DeepLabv3

Мережа DeepLab v3 запропонована в 2017 році [14]. Її концепція полягає в тому, що робиться проріжена згортка (dilated convolution). В архітектурі Deeplabv3 використовуються новий енкодер-декодер зі складною віддільною згорткою. Модель енкодер-декодер здатна отримати чіткі межі об'єкта. Загальні мережі енкодер-декодер були успішно застосовані до багатьох завдань комп'ютерного зору, включаючи виявлення об'єктів, оцінку пози людини, а також семантичну сегментацію.

Як правило, мережі енкодер-декодер містять:

- Модуль енкодера, який поступово скорочує функціональні можливості і захоплює більш високу семантичну інформацію.
- Модуль декодера, який поступово відновлює просторову інформацію.

На додаток до вищезгаданої мережі енкодер-декодер, це також стосується віддільної глибинної згортки, яка збільшує обчислювальну ефективність. Це досягається шляхом розкладання стандартної згортки на глибинну згортку, за якою слідує точкова згортка (тобто згортка 1 × 1). Зокрема, глибинна згортка виконує просторову згортку незалежно для кожного вхідного каналу, в той час як точкова

згортка використовується для об'єднання вихідних даних від глибинної згортки.

На рисунку 3.2 представлено класичну архітектуру мережі DeepLabv3.

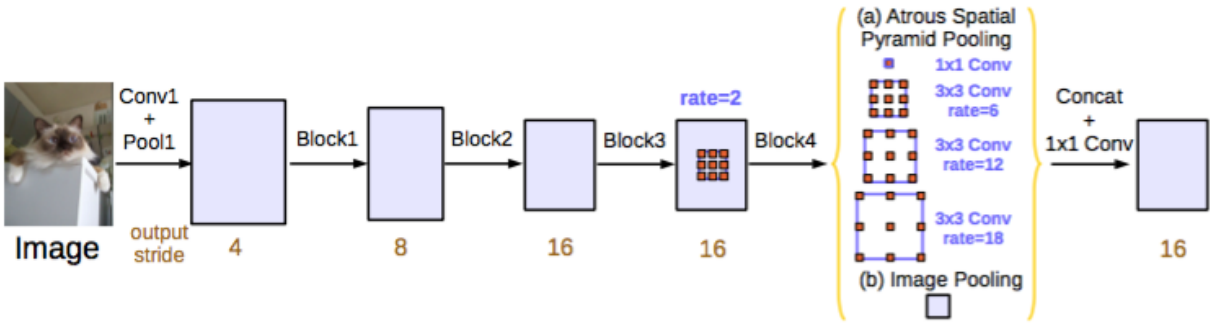

Рисунок 3.2 — Архітектура DeepLabv3 [14]

3.3 Об’єднання підходів для підвищення точності семантичної сегментації

На основі проведених досліджень в даній роботі запропоновано об'єднати ці підходи. В bottleneck мережі U-Net вбудовано проріджені згортки (Рис.3.3), які застосовуються послідовно. Кроками прорідження вибрано 1, 2, 4, 8. Окремо всі результати після блоків прорідженої згортки складаються (конкатенуються) і подаються на праву частину мережі U-Net (декодер).[1]

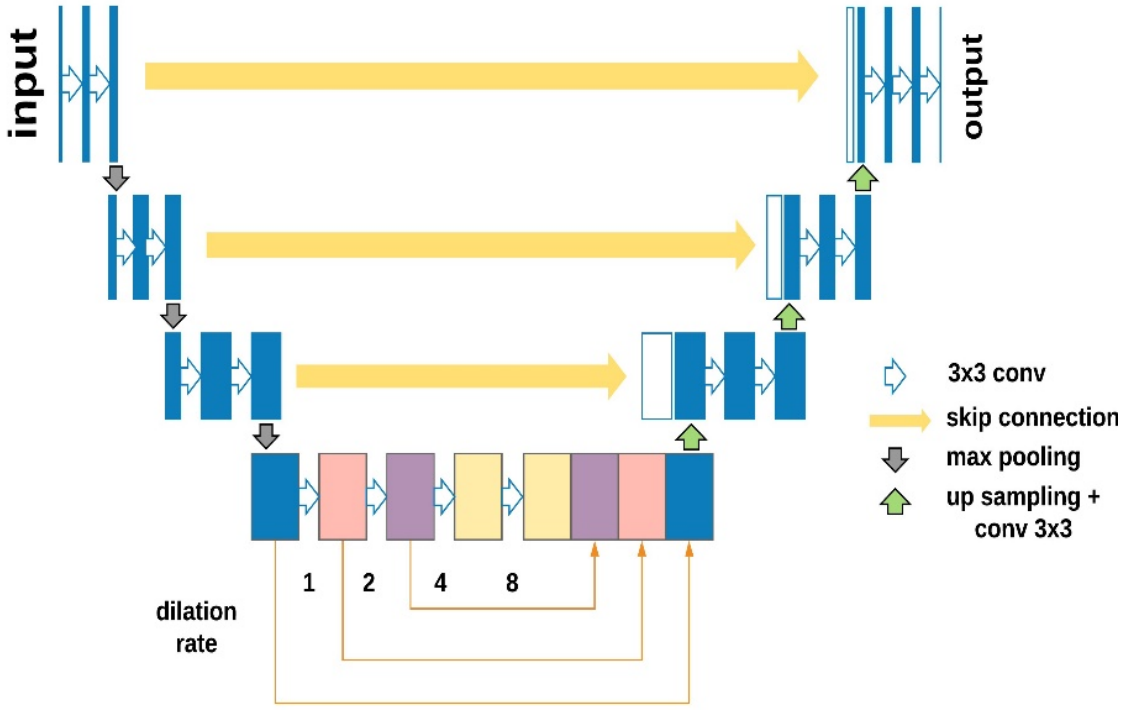

Рисунок 3.3 — Схема модифікації мережі U-Net

3.4 PSPNet

Деякі моделі глибокого навчання явно представляють методи роботи з інформацією різних масштабів. Наприклад, мережа Pyramid Scene Parsing (PSPNet) виконує операцію об'єднання (через функцію max або average) за допомогою згорток різного розміру і з різними кроками, застосованих до вихідних відображеннь ознак з зі згорковою нейронною мережею (наприклад, ResNet). Далі за допомогою білінійної інтерполяції відбувається збільшення розмірності виходу; потім модель об'єднує уздовж осі каналу все нові виходи. Для генерації передбачення фінальна згортка виконується вже на об'єднаному виході.

На рисунку 3.4 показано класичну архітектуру PSPNet.

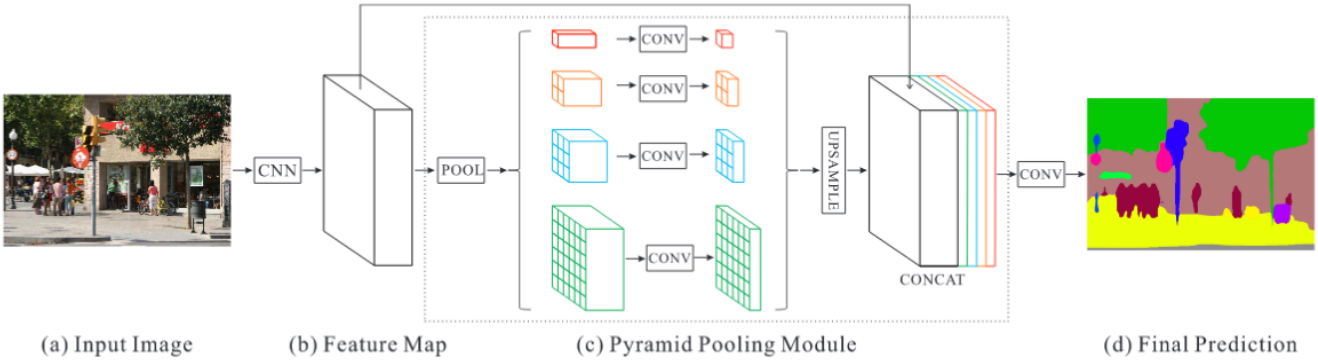

Рисунок 3.4 — Архітектура PSPNet

3.5 Аналіз методів оцінки якості сегментації

Процес підбору алгоритму сегментації для аналізу певного типу зображень є досить непростим та трудомістким. Зазвичай прийнято виділяти два основні типи оцінки якості сегментації: суб’єктивний та об’єктивний.

Під суб’єктивним методом мають на увазі візуальну оцінку, а це означає, що оцінка проводиться людиною, що є досить великим недоліком даного методу.

Об’єктивні методи, на відміну від суб’єктивних, базуються на кількісних

показниках, а не на візуальних. На рисунку 3.5 представлено класифікацію методів оцінки якості сегментації.

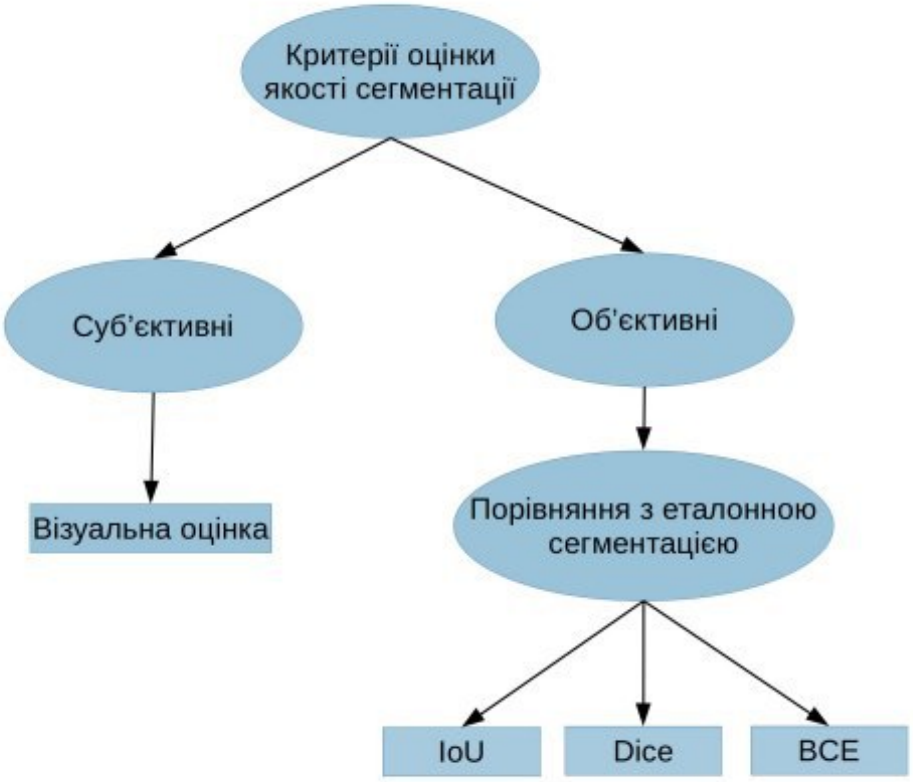

Рисунок 3.5 — Класифікацію критеріїв оцінки якості сегментації

IoU і Dice використовують дещо різні підходи для вимірювання того, наскільки схожі результати алгоритму сегментації зображення відповідним результатам сегментації, які було підтверджено спостереженнями.

Спершу розглянемо IoU, його можна представити у геометричному вигляді:

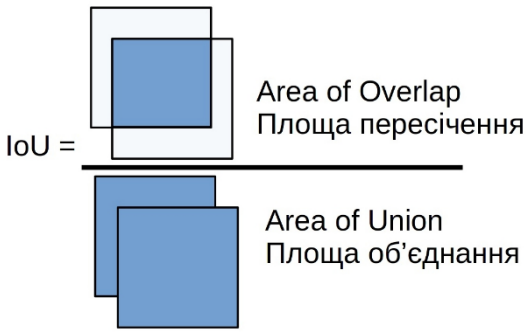

Рисунок 3.6 — Графічний зміст метрики IoU

Розглядаючи цей вираз, ми бачимо, що IoU — це просто співвідношення. У чисельнику ми обчислюємо область перекриття між передбачуваним обмежувальною областю і областю яку буд насправді обмежено. Знаменник — область об'єднання, або, простіше кажучи, область, яка охоплюється як передбачуваною обмежувальною областю так і дійсною обмежувальною областю.

Поділ площі перекриття на площу об’єднання дає остаточний показник — IoU.

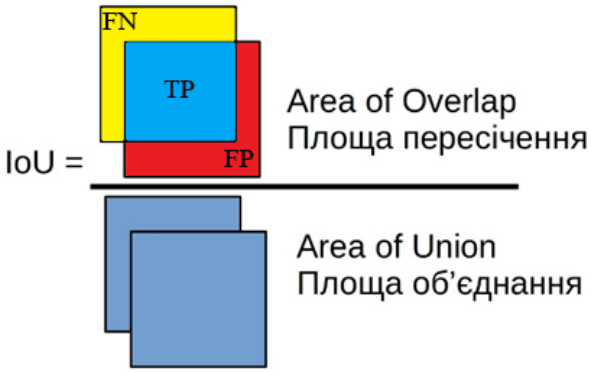

Рисунок 3.7 — Графічний зміст IoU з позначенням FN, TP, FP

На рисунку 3.7, припустимо, що поле у верхньому лівому куті, яке позначено жовтим кольором — це область сегментації, розмічена людиною. Поле яке позначено червоним є результатом сегментації, який створив алгоритм глибокого навчання на зображенні. Площа перекриття між результатами людини та штучного інтелекту — це синій квадрат у зображенні чисельника. Це область, де алгоритм сегментації зображення ідентифікує пікселі, які точно відповідають анотованій сегментації. Ці пікселі відомі як True positives (TP). Область об'єднання в знаменнику поєднує результати сегментації як людини, так і штучного інтелекту, а потім віднімає TP, щоб запобігти подвійному підрахунку цих пікселів.

На малюнку 3.7 пікселі в червоній області були помилково сегментовані алгоритмом і відомі як False positives (FP). Пікселі в жовтій області повинні бути сегментовані алгоритмом, але вони були пропущені. Ці пропущені пікселі відомі як False negatives (FN). Якщо площа перекриття дорівнює площі об'єднання, ми маємо ідеальну сегментацію і IoU дорівнює 1 (Рисунок 3.8). У цьому випадку FP, TP і FN всі рівні 0.

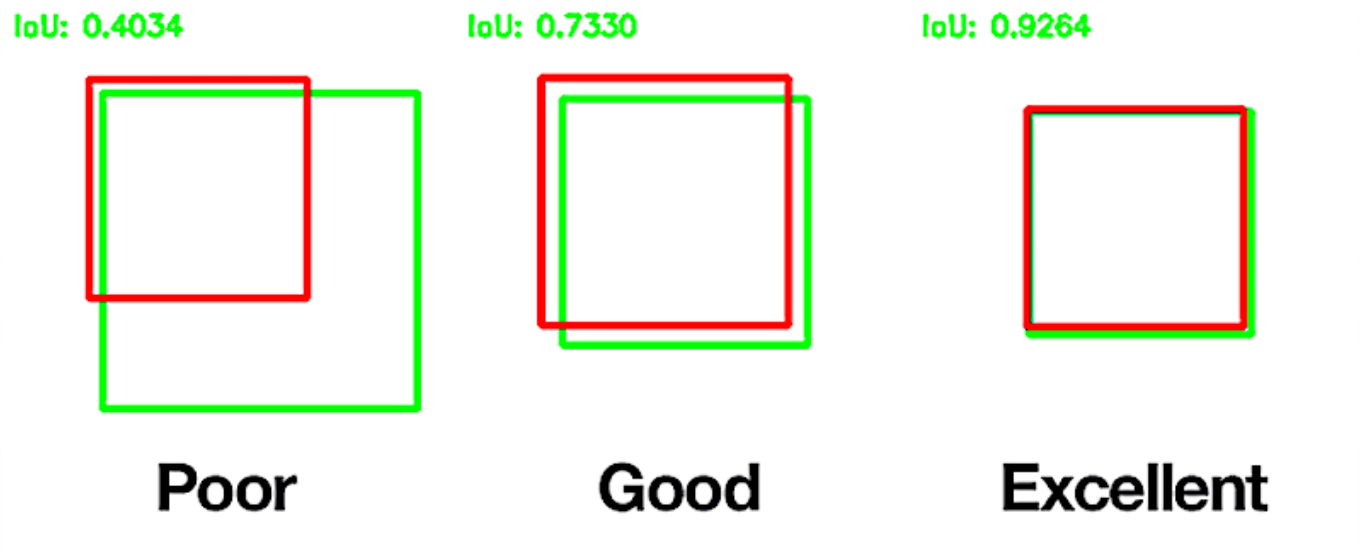

Рисунок 3.8 — Ілюстрація впливу співвідношень площ перекриття і об’єднання на значення IoU

Зараз розглянемо як IoU відноситься до коефіцієнта подібності Dice. Для початку спробуємо виразити IoU через області FP, TP і FN, які позначено на рисунку 3.7:

$$IoU = \frac{TP}{(TP + FP + FN)} \tag{3.1}$$

Визначення метрики Dice не так легко описано геометрично, як IoU. На рисунку 3.9 подається графічне представлення цієї мерики.

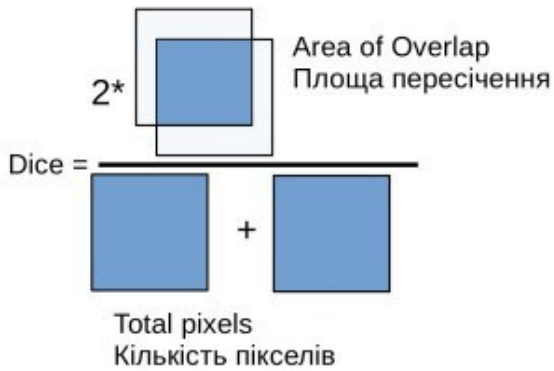

Рисунок 3.9 — Графічний сенс метрики Dice

Dice може бути виражена через TP, FP та FN наступним чином:

$$Dice = \frac{2 \times TP}{(TP + FP) + (TP + FN)} \tag{3.2}$$

Зробивши декілька перетворень отримаємо наступний вираз:

$$IoU = \frac{Dice}{2 - Dice} \tag{3.3}$$

Binary Cross-Entropy (BCE) також називають Sigmoid Cross-Entropy. Складається з функції сигмоїдної активації, а також з перехресної ентропії. На відміну від втрати Softmax, вона не залежить від кожного векторного компонента (класу), тобто функції втрат, обчислені для кожного компонента векторного виходу CNN, не впливають на інші значення компонентів.

Дуже часто BCE використовується для рішень типу "так / ні", наприклад, класифікації з багатьма значеннями. Функція втрат відображає те, наскільки помилкові прогнози вашої моделі. Наприклад, у проблемах з декількома мітками, де приклад може належати до кількох класів одночасно, модель намагається вирішити для кожного класу, чи належить приклад цьому класу чи ні.

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=0}^{N}(y * \log(\hat{y}_i) + (1 - y) * \log(1 - \hat{y}_i)) \tag{3.4}$$

де y — значення мітки (1 або 0), ŷ - передбачуване значення.

Binary Cross-Entropy вимірює, наскільки далеко від справжнього значення (що дорівнює 0 або 1), є прогноз для кожного з класів, а потім узагальнює ці класові помилки для отримання остаточної функції втрат.

3.6 Висновки до розділу

В цьому розділі було розглянуто алгоритми і методи, які було використано для розробки системи. Було представлено такі моделі: U-Net, DeepLabv3, PSPNet. Описано архітектури, а також зроблено огляд методу об'єднання двох моделей, який і було використано при розробці системи.

Також було проведено аналіз методів оцінки сегментації. Було виділено основні методи оцінки якості сегментації, а також було визначено які методи краще застосовувати. Також було описано та проілюстровано геометричний зміст методів IoU та Dice. Також було описано метод Binary Cross-Entropy (BCE) і наведено обчислення функції втрат.

4. РЕАЛІЗАЦІЇ ТА АРХІТЕКТУРИ

4.1 Функції втрат

В машинному навчанні, як і в математиці та статистиці функція втрат — це функція яка відображає вплив події або певної величини на дійсне число. Ще досить часто можна зустріти формулювання в якому функцією втрат називають похибку, яку необхідно приписувати під час вирішення певної задачі.

Функції втрат використовують в різних галузях. Наприклад, в статистиці функція втрат описує оцінку параметрів. Також функції втрат використовують в економіці і там вони зазвичай є економічною вартістю. В класифікації функцію втрат використовують як штраф за неправильну класифікацію прикладу.

4.1.1 Dice Loss

Dice Loss бере свої витоки з коефіцієнта Соренсена-Дайса, який було винайдено ще в 1940х. В задачах комп'ютерного зору дану функцію почали розглядати лише в 2016 році, після того як Міллетарі запропонував її для 3D сегментації медичних зображень[16].

В розділі 3.5 вже було описано коефіцієнт Дайса (Dice). Dice — це міра перекриття між двома множинами. Наприклад, якщо два набори A і B ідеально перекривають одна одну, тоді Dice отримує максимальне значення близьке до 1. В іншому випадку Dice починає знижуватися, отримуючи мінімальне значення близьке до 0, якщо ці множини взагалі не перетинаються. Тому діапазон Dice становить від 0 до 1, чим більший, тим краще. Таким чином, ми можемо використовувати Dice Loss для максимізації перекриття між двома наборами. Функцію Dice Loss можна описати наступним виразом:

$$Dice\ Loss = 1 - Dice \tag{4.1}$$

4.1.2 Focal Loss

Focal Loss призначена для вирішення одноетапного сценарію виявлення об'єктів, в якому спостерігається надзвичайний дисбаланс між класами переднього плану та фоновими класами під час навчання (наприклад, 1: 1000). [17]

Почнемо виведення Focal Loss з перехресної ентропії (Cross Entropy) для бінарної класифікації:

$$CE(p,y) = \begin{cases} -\log(p) & \text{якщо } y = 1 \\ -\log(1-p) & \text{інакше.} \end{cases} \tag{4.2}$$

де, $y \in \{\pm 1\}$ визначає клас ґрунтової істинності і $p \in [0,1]$ — розрахункова ймовірність моделі для класу з міткою $y = 1$. Для наочної зручності визначаємо pt:

$$pt = \begin{cases} p & \text{якщо } y = 1 \\ 1-p & \text{інакше.} \end{cases} \tag{4.3}$$

і перепишемо $CE\,(p,y) = CE\,(pt) = -log\,(pt)$. Функція втрат CE може бути розглянута як синя (верхня) крива на рисунку 4.1. Однією помітною властивістю цієї функції втрат, яку легко помітити на її графіці, є те, що навіть приклади, які легко класифікуються, несуть збитки з нетривіальною величиною. Якщо підсумовувати велику кількість простих прикладів, ці малі значення втрат можуть перекрити рідкісний клас.

Загальним методом усунення дисбалансу класів є введення коефіцієнта вагів $\alpha \in [0,1]$ для класу 1 і $1 - \alpha$ для класу −1. На практиці α може встановлюватися за частотою зворотного класу або розглядатись як гіперпараметр, встановлений шляхом перехресної перевірки. Для наочної зручності ми визначаємо αt аналогічно тому, як ми визначили pt. Ми записуємо α-збалансовані втрати CE як:

$$CE(pt) = -\alpha t\ log(pt) \tag{4.4}$$

Ця функція втрат — це просте розширення до CE, яке розглядається як експериментальна база для запропонованих Focal Loss.

Як показують експерименти, великий дисбаланс класу, що виникає під час тренування щільних детекторів, перекриває функцію втрат перехресної ентропії. Легко класифіковані негативні приклади складають більшість втрат і домінують у градієнті. Хоча α врівноважує важливість позитивних / негативних прикладів, він не відрізняє легких / важких прикладів. Натомість, запропоновано змінити функцію втрати на легкі приклади, щоб зменшити вагу, і, таким чином, зосередити тренування на важких негативних прикладах. Більш формально, пропонується додати модулюючий коефіцієнт $(1 - pt)^{\gamma}$ до поперечної функції втрат ентропії з регульованим параметром фокусування $\gamma \geq 0$. Визначаємо Focal Loss як:

$$FL(pt) = -(1 - pt)^{\gamma}\ log(pt) \tag{4.5}$$

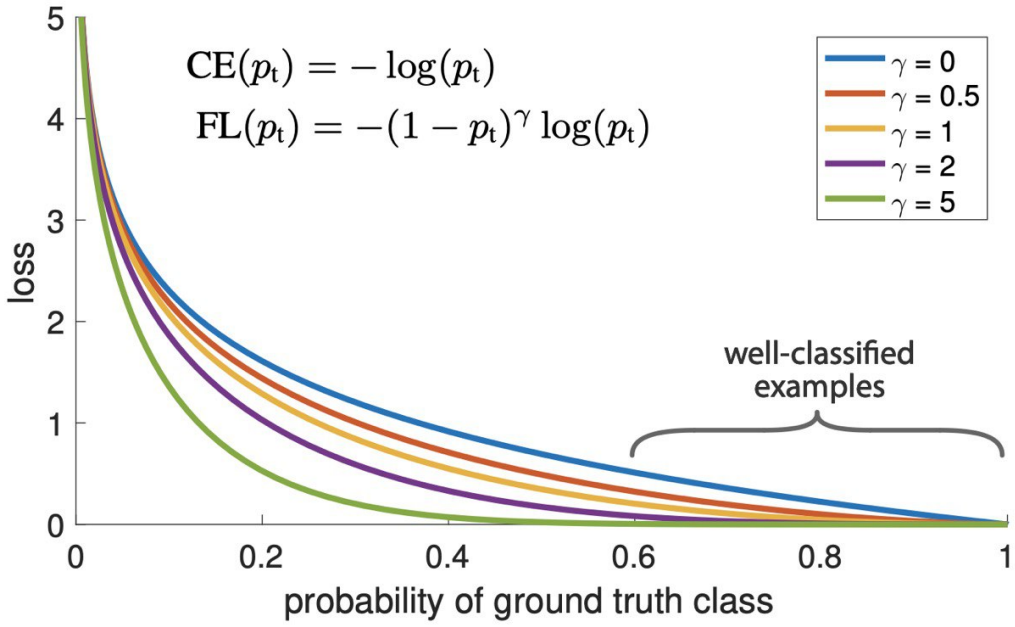

Рисунок 4.1 — Візуалізація Focal Loss на кількох значеннях $\gamma \in [0, 5]$

Відзначимо дві властивості Focal Loss:

- Якщо приклад неправильно класифікується, а pt є малим, модулюючий коефіцієнт становить близько 1, а функція втрат не впливає. Як $pt \to 1$, коефіцієнт переходить до 0, а величина втрати для добре класифікованих прикладів зменшується.
- Параметр фокусування γ плавно регулює швидкість, з якою легкі приклади зменшуються. Коли $\gamma = 0$, FL еквівалентний CE, а при збільшенні γ ефект модулюючого коефіцієнта також збільшується.

Інтуїтивно, модулюючий коефіцієнт зменшує внесок збитків з простих прикладів і розширює діапазон, в якому приклад отримує низькі втрати Це, у свою чергу, збільшує важливість виправлення неправильно класифікованих прикладів.

4.2.3 Tversky

Tversky Loss встановлює різні ваги до false negatives (FN) і false positives (FP), що відрізняється від Dice Loss, використовуючи рівні ваги для FN і FP.

Вихідний шар у мережі складається з площин c, по одному на клас (c = 2 при виявленні ураження)[18]. Нехай P і G — сукупність бінарних міток прогнозованої та основної істини відповідно. Коефіцієнт подібності Dice між двома двійковими томами визначається як:

$$D(P, G) = \frac{2|PG|}{|P|+|G|} \tag{4.6}$$

Якщо це використовується в шарі втрат у тренуванні, воно зважує FP та FN (точність та відкликання) однаково. Для того, щоб зважити FN більше, ніж FP, під час навчання нашої мережі для врівноважених даних, де виявлення невеликих пошкоджень має вирішальне значення, ми пропонуємо рівень втрат на основі індексу Тверського. Індекс Тіверського визначається як:

$$S(P, G; \alpha, \beta) = \frac{|PG|}{|PG| + \alpha|P \setminus G| + \beta|G \setminus P|} \quad (4.7)$$

де α і β контролюють величину штрафних санкцій для FPs та FNs відповідно.

Для визначення функції втрати Тверського ми використовуємо наступне формулювання:

$$T(\alpha, \beta) = \frac{\sum_{i=1}^{N} p_{0i} g_{0i}}{\sum_{i=1}^{N} p_{0i} g_{0i} + \alpha \sum_{i=1}^{N} p_{0i} g_{1i} + \beta \sum_{i=1}^{N} p_{1i} g_{0i}} \quad (4.8)$$

де на виході шару softmax p_{0i} - це ймовірність вокселя i бути ураженням, а p_{1i} — ймовірність вокселя i бути не ураженням. Також g_{0i} є 1 для вокселя з ураженням і 0 для вокселя без ураження та навпаки для g_{1i}. Градієнт втрати в рівнянні 4.8 щодо p_{0i} та p_{1i} можна обчислити так:

$$\frac{\partial T}{\partial p_{0i}} = 2 \frac{g_{0i}\left(\sum_{i=1}^{N} p_{0i} g_{0i} + \alpha \sum_{i=1}^{N} p_{0i} g_{1i} + \beta \sum_{i=1}^{N} p_{1i} g_{0i}\right) - (g_{0j} + \alpha g_{1j}) \sum_{i=1}^{N} p_{0i} g_{0i}}{\left(\sum_{i=1}^{N} p_{0i} g_{0i} + \alpha \sum_{i=1}^{N} p_{0i} g_{1i} + \beta \sum_{i=1}^{N} p_{1i} g_{0i}\right)^2} \quad (4.9)$$

$$\frac{\partial T}{\partial p_{1i}} = -\frac{\beta g_{1j} \sum_{i=1}^{N} p_{0i} g_{0i}}{\left(\sum_{i=1}^{N} p_{0i} g_{0i} + \left(\sum_{i=1}^{N} p_{0i} g_{0i} + \alpha \sum_{i=1}^{N} p_{0i} g_{1i} + \beta \sum_{i=1}^{N} p_{1i} g_{0i}\right)\right)^2} \quad (4.10)$$

Використовуючи це формулювання, нам не потрібно врівноважувати ваги для тренувань. Також, регулюючи гіперпараметри α і β, ми можемо контролювати компроміс між FP та FN. Примітно, що у випадку $\alpha = \beta = 0{,}5$ індекс Тверського спрощується таким самим чином, як коефіцієнт Dice, який також дорівнює балу F1. При $\alpha = \beta = 1$ формула 4.7 виробляє коефіцієнт Танімото, а встановлення $\alpha + \beta = 1$ виробляє набір балів F_β. Більші β мають вагу нагадування вище, ніж точність (роблячи більший акцент на FN). Припускається, що використання більш високих β в нашій узагальненій функції втрат у навчанні призведе до більш високого узагальнення та підвищення продуктивності для незбалансованих даних; і ефективно допомагає нам перенести акцент на нижчі FN та збільшити виклик.

4.2 Висновки до розділу

В даному розділі було розглянуто основні архітектури та функції втрат.

Під час розгляду функцій втрат було дано визначення функції втрат та розглянуто 3 функції: Dice Loss, Focal Loss, Tversky Loss.

Було наведено математичне та теоретичне обгрунтування функцій, також було описано покрокове виведення формул функцій. Описані в розділі функці є актуальними, дозволяють покращувати результати аналізу зображень та підходять для вирішення даної задачі.

5. ОБЧИСЛЮВАЛЬНІ ЕКСПЕРИМЕНТИ

5.1 Набір даних, параметри навчання, крос-валідація

Для тестування був обраний конкурс, який проходив на платформі Kaggle [19] Understanding Clouds from Satellite Images [20], який полягав у виділенні областей хмар на кожному знімку областей, які відповідають класам Fish (риба), Flower (квітка), Gravel (гравій), Sugar (цукор). Змагання проходило з 16 серпня 2019 року до 19 листопада 2019 року (3 місяці) і було організовано інститутом метеорології ім.Макса Планка (Max Planck Institute for Meteorology), який базується в Гамбурзі (Германія). Призовий фонд змагання $10.000, який ділиться між першими трьома переможцями.

Специфікою платформи Kaggle є те, що платформа надає два набори даних: тренувальний та тестовий, в тренувальному відомі вихідні результати, в тестовому — ні. Після опрацювання тестового набору даних певним алгоритмом потрібно відправити результати (сабміт) на платформу Kaggle, де буде оцінено результат. Вхідні маски та результат потрібно відправляти в форматі RLE (run-length encoding), що дозволяє не завантажувати цілі картинки на сервер. Код для кодування та декодування масок наданий організаторами.

Тестовий набір даних в свою чергу ділиться на 2 частити у співвідношенні 25%:75%. По 25% результат оголошується негайно. Проте фінальна оцінка дається по 75% даних, які відкриваються після закінчення змагання. Серед всіх сабмітів користувач повинен обрати найкращі два, які будуть рахуватися як фінальні.

Дані були не рівномірно розподілені: серед 5546 змімків було 2781 змінків з класом ‘fish’, 2365 — з класом ‘flower’, 2939 — з класом ‘gravel’, 3751 — з класом ‘sugar’ (рисунок 5.1)

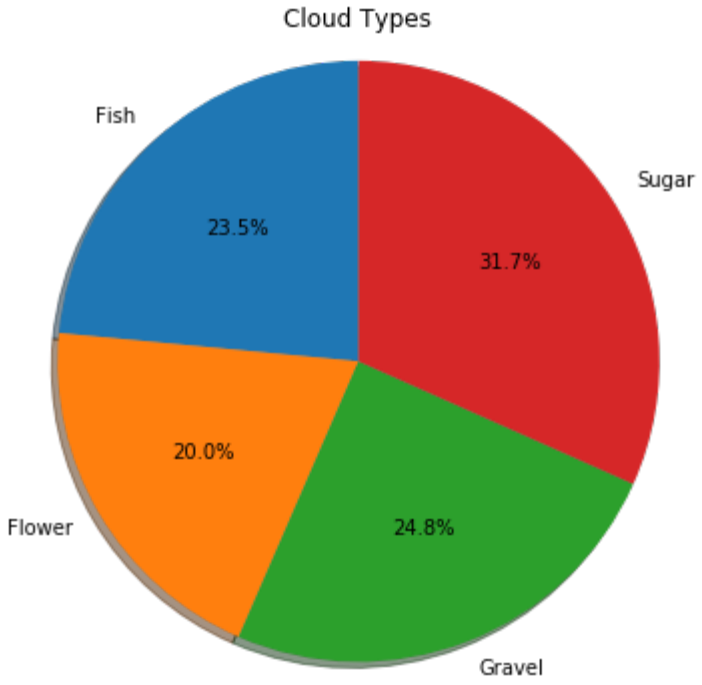

Рисунок 5.1 — Розподілення класів у вхідному наборі даних

На одному знімку може бути декілька класів. Найчастіше зустрічалися знімки, які мали по 2 класи (рисунок 5.2).

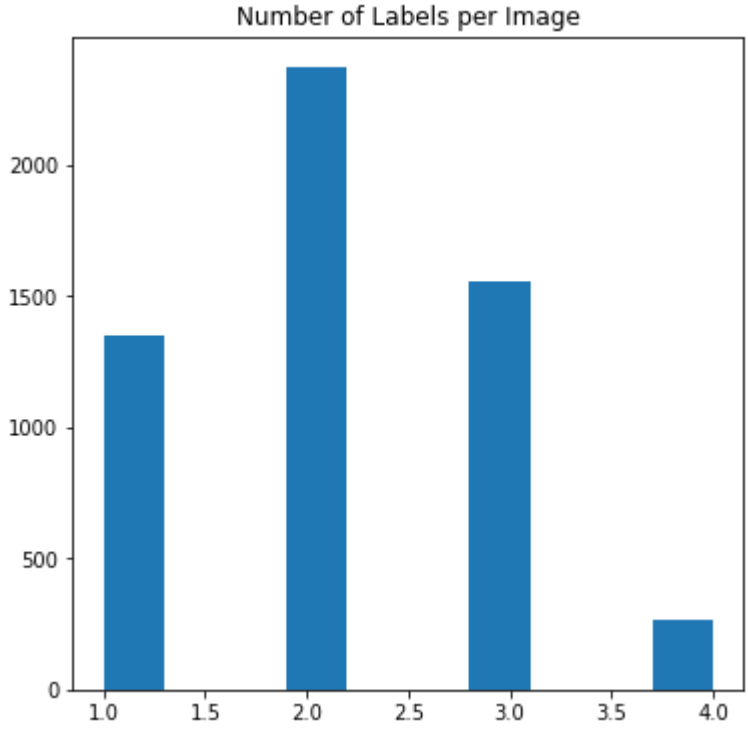

Рисунок 5.2 — Гістограма розподілення кількості класів на одне зображення.

Розглянемо приклади сегментованих зображень (рисунок 5.3). Як видно з

рисунків, вихідні маски сегментації наближені до квадратних розмірів. Існує 2 способи розв’язання задач такого типу: детекція та сегментація. Суть семантичної сегментації зводиться до класифікації кожного пікселя, а детекції — потрібно виділити область, де знаходиться потрібний об’єкт (рисунок 5.4). В даній роботі обрано метод семантичної сегментації, оскільки архітектури нейронних мереж, які призначені для детекції навчаються в рази довше, проте результат не завжди задовільний.

Також на знімках можна помітити чорні полоси, які пов’язані з особливістю захвату зображення супутником, причому на різних знімках ці полоси були в різних місцях або були відсутні.

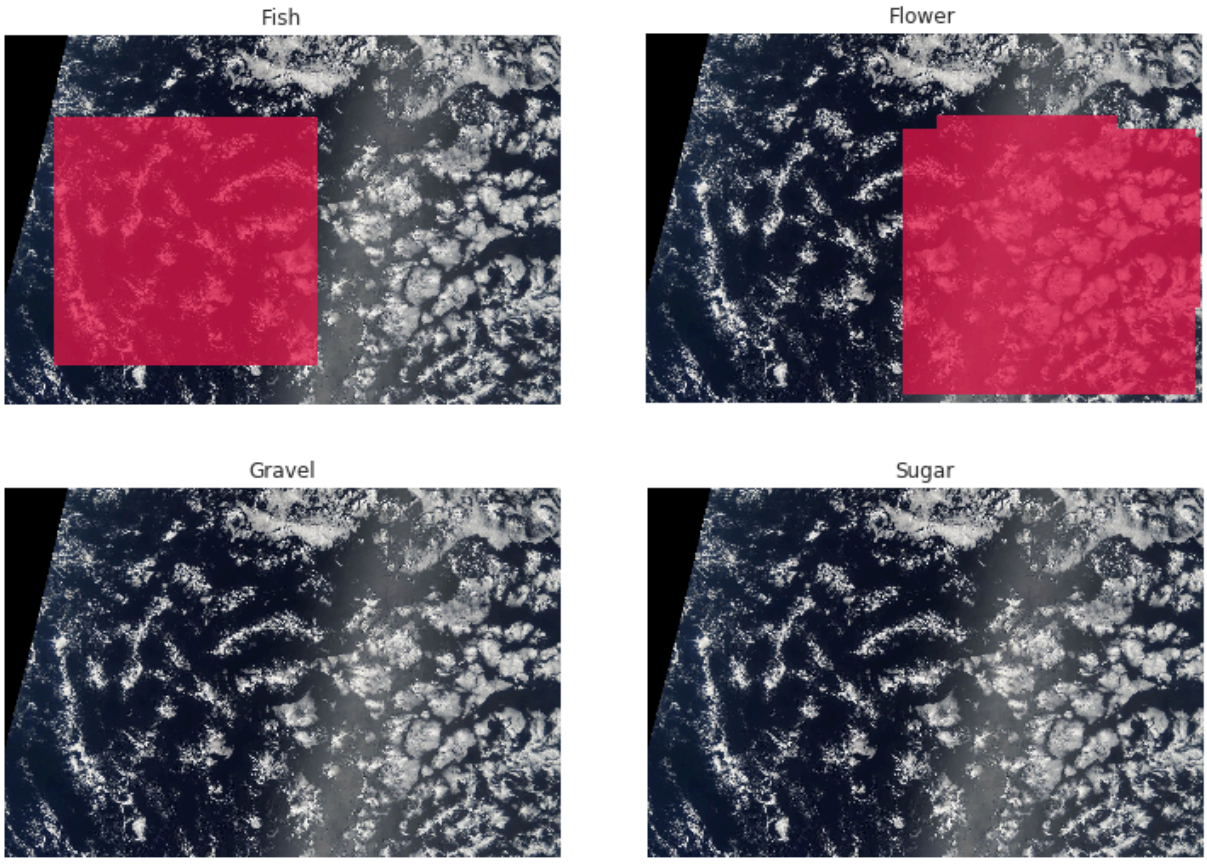

Рисунок 5.3 — Приклади сегментації для одного зображення.

Рисунок 5.4 — Відмінність між детекцією та семантичною сегментацією

Запропонована модифікація мережі U-Net (розділ 3.3) апробовано на задачі Understanding Clouds from Satellite Images [20]. Задача полягає в аналізі супутникових знімків на основі семантичної сегментації. Набір даних складається з 5546 зображень в навчальному наборі та 3698 — в тестовому, по якій і оцінювалася фінальна оцінка. Метрикою точності був коефіцієнт Дайса (dice coeficient), який узагальнювався по всій виборці даних. В якості тестової мережі була застосована U-Net, енкодером якої була мережа SE-ResNet50. Тренування проводилося протягом 50 епох, початковий крок навчання - 0.01. Якщо коефіцієнт Дайса не змінювався протягом 5 епох — крок навчання понижувався в 5 разів. Для боротьби з явищем перенавчання застосовувалася аугментація - спотворення вхідного і вихідного зображення. Запропоноване рішення модифікації мережі U-Net дозволило підвищити точність з 0,638 до 0,647.[1]

Після того, як ми закінчимо навчання нашої моделі, ми не можемо бути впевнені, що модель матиме бажану точність та дисперсію у робочому середовищі. Нам потрібна якась впевненість у точності прогнозів, які висуває наша модель. Для цього нам потрібно перевірити нашу модель. Цей процес вирішення питання про те, чи є чисельні результати кількісної оцінки гіпотезованих зв'язків між змінними, прийнятними як описи даних, відомий як валідація.

Для оцінки продуктивності будь-якої моделі машинного навчання нам потрібно перевірити її на деяких небачених даних. Виходячи з продуктивності моделей за небаченими даними, ми можемо сказати, що наша модель недооцінена / надмірна /

добре узагальнена. Крос-валідація (CV) - одна з методик, що використовуються для перевірки ефективності моделей машинного навчання, це також процедура повторного відбору проб, яка використовується для оцінки моделі, якщо у нас є обмежені дані.

.

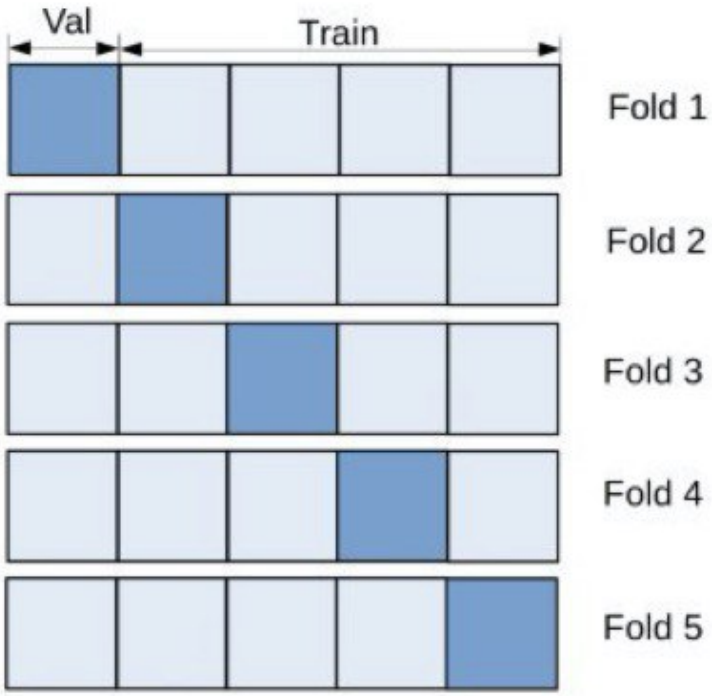

Рисунок 5.5 — Крос-валідація

5.2 Аугментація

Для боротьби з явищем перенавчання застосовувалася аугментація, яка полягала в тому, що зображення та маска спотворюються одинаково. Можна виділити 2 типи спотворень: після яких можна відновити початкове зображення та не можна. До першої можна віднести аугментації групи D4: повороти зображень та відображення відносно вертикальної осі та горизонтальної. Оскільки зображення після зміни розмірності не були квадратні, застосовувалися лише відображення відносно вертикальної та горизонтальної осі (рисунок 5.6).

Інший тип можна поділити на аугментації, які спотворюють вхідне зображення та маску та які спотворюють лише вхідне зображення.

До першого можна віднести повороти зображення на довільний кут у межах (-

90; +90) і вирізання частини зображення (RandomCrop) (рисунок 5.7).

Також до цього типу можна віднести еластичну трансформацію (ElasticTransform), оптичне спотворення (OpticalDistortion), спотворення осей (GridDistortion) (рисунок 5.8).

Серед аугментацій, які спотворюють лише вхідне зображення були: Jpeg-компресія (JpegCompression) різною яскравістю, зміна контрасту та гамми вхідного зображення (RandomContrast, RandomBrighness), випадкова перестановка кольорів, використання розмиття (GaussianBlur).

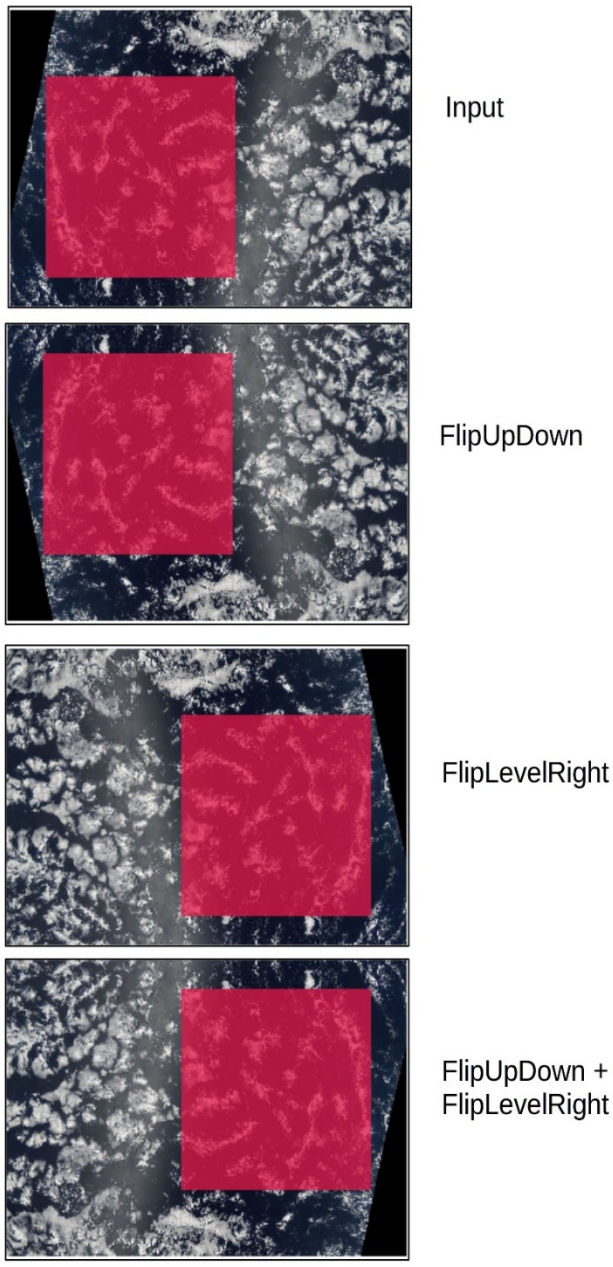

Рисунок 5.6 — Приклад аугментації відображення відносно вертикальних та горизонтальних осей

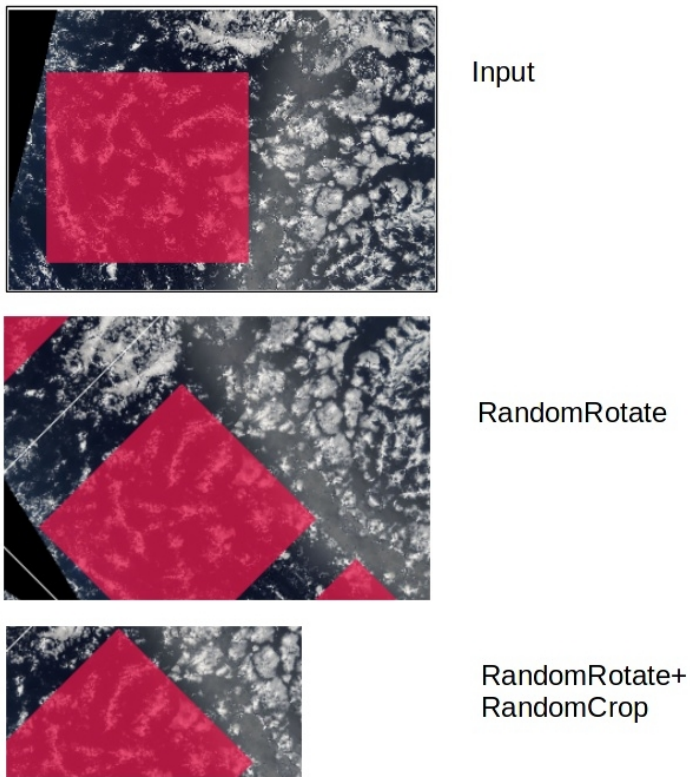

Рисунок 5.7 — Приклади аументації з поворотом та вирізанням

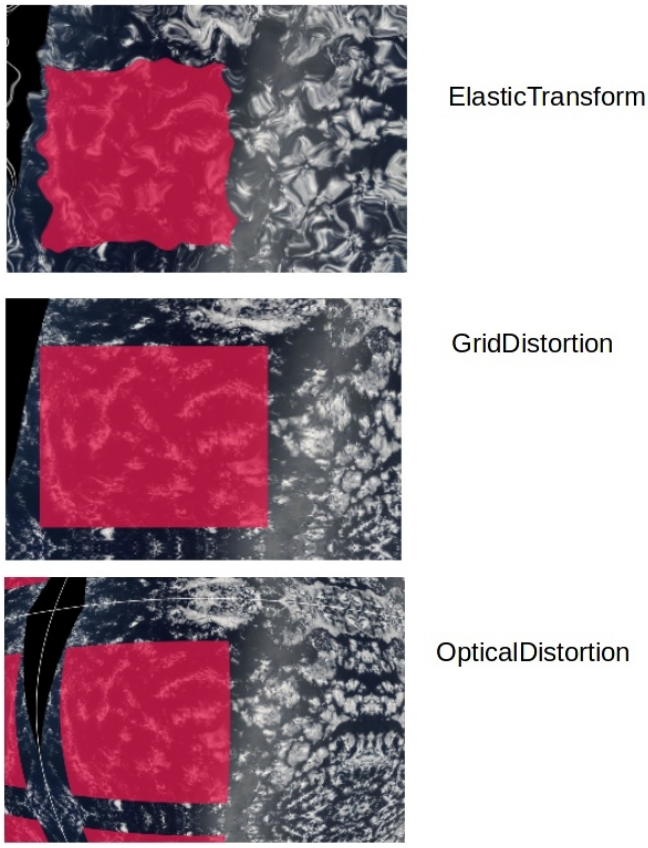

Рисунок 5.8 — Приклади аугментації зі зміною кольорів, якравості, конрасту

Кожна з аугментацій застосовувалася незалежно від інших з певною ймовірністю (рисунок 5.9). Ймовірність застосовування кожної з аугментацій наведено в таблиці 5.1.

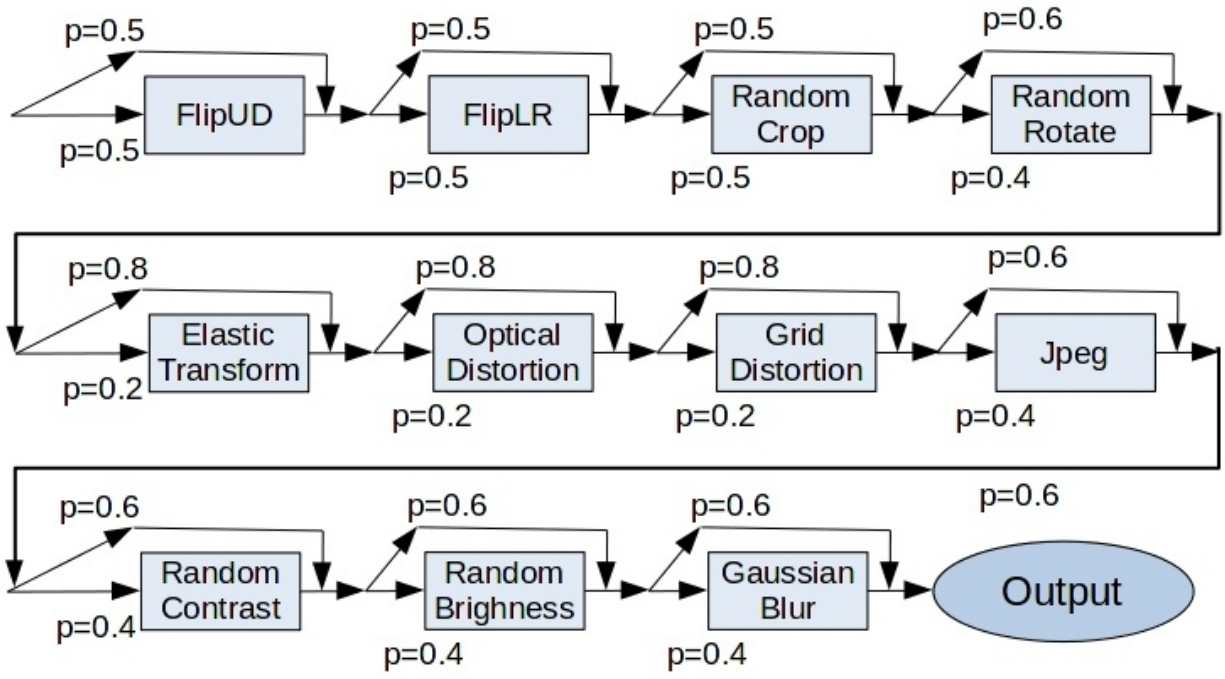

Рисунок 5.9 — Схема аугментації

Таблиця 5.1 — Ймовірність застосування аугментації

№	**Назва аугментації**	**Ймовірність застосування, %**
1.	FlipUD	50%
2.	FlipLR	50%
3.	RamdomCrop	50%
4.	RandomRotate	40%
5.	ElasticTransform	20%
6.	OpticalDistortion	20%
7.	GridDistortion	20%
8.	Jpeg	40%
9.	RandomContrast	40%
10.	RandomBrightness	40%
11.	GausianBlur	40%

5.3 Аугментація під час тесту (TTA)

Для збільшення точності фінального результату, було використано TTA (test time augmentation). На нейронну мережу подається не лише оригінальне зображення, а також зображення після поворотів відносно вертикальних і горизонтальних відображень. До масок, які отримується на виході нейронної мережі, повторно застосовуються ці самі ж аугментації. В результаті отримуються 4 зображення. Після сумування та знаходження середнього, залишається лише одне (рисунок 5.10).

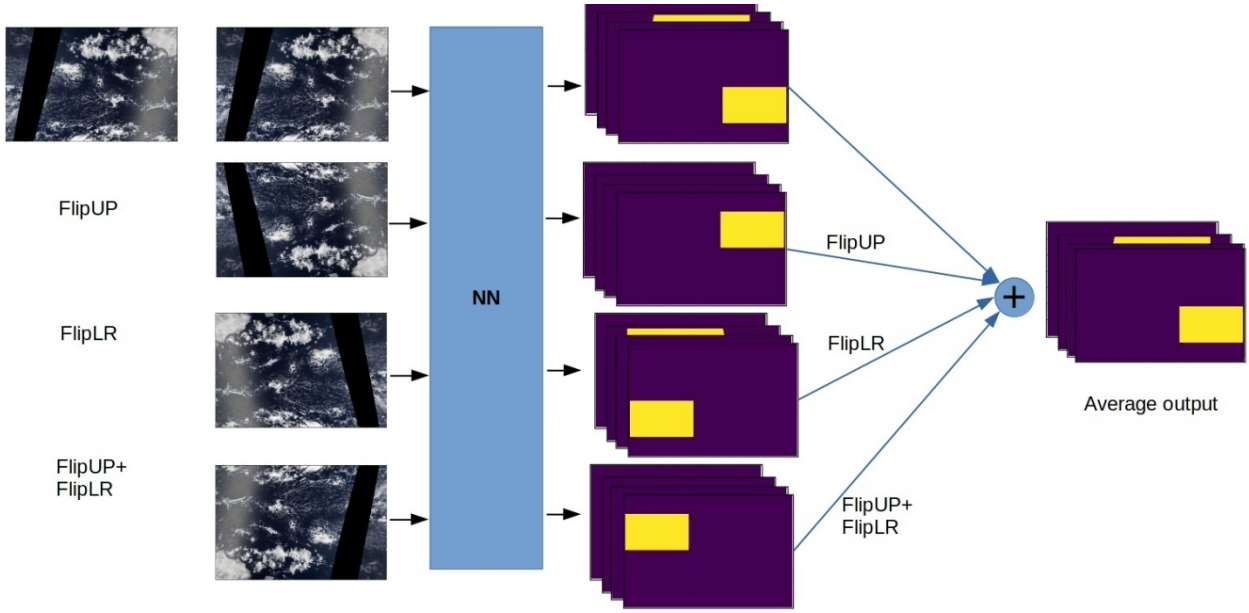

Рисунок 5.10 — Предікт з використанням TTA

Такий метод дозволяє підвищити точність роботи нейронної мережі на локальній крос-валідації з 0.5690 до 0.5710 (на тренувальній вибірці) та з 0.64928 до 0.6504 на тестовій вибірці (оцінка проводилася платформою kaggle).

5.4 Використані моделі

Фінальний результат базувався на мережах архітектури U-Net. Моделі було реалізовано з використанням бібліотеки Keras. Реалізації моделей використано з [15]. Моделі було модифіковано для конкретної задачі.Всі моделі навчалися з використанням крос-валідації (на 5 фолдах). В якості backbone-мережі вибиралися

мережі різних архітектур, таких як ResNet-34, ResNet-50, ResNet-101, ResNext-50, DenseNet-56, SeRersNet-34, SeResNet-50. Функція втрат вибиралася експериментально, та було визначено, що бінарна крос-ентропія разом з focal-loss [розділ 4] у пропорції та dice-loss у пропорціях 0.25, 0.25 та 0.5 показує найкращі результати. Тобто, фінальна функція втрат була:

$$loss = 0.25 * BCE(y_{true}, y_{pred}) + 0.25focal(y_{true}, y_{pred}) + 0.5Dice(y_{true}, y_{pred}) \quad (5.1)$$

де y_{true} — очікуваний результат,

y_{pred} — результат, передбачений нейронною мережею.

Функція втрат бінарної крос-ентропії BCE показує, наскільки точно нейронна мережа визначила клас конкретного пікселю (в нашому випадку або клас фон або один з потрібних класів), focalLoss дозволяє ‘звернути’ увагу нейронній мережі на важкі приклади, функція втрат Dice дозволяє оперувати не конкретними пікселями, а цілими областями.

Під час навчання,перевірялася точність на валідаційнійній частині даних. Якщо коефіцієнт Дайса не збільшувався на протязі 5 епох, коефіцієнт навчанння зменшувався в 2 рази. Якщо коефіцієнт Дайса не збільшувався на протязі 10 епох, навчання зупинялося. На рисунку 5.11 наведено графік лос-функції однієї з моделей, на рисунку 5.12 коефіцієнт Дайса та коефіцієнт навчання на рисунку 5.13.

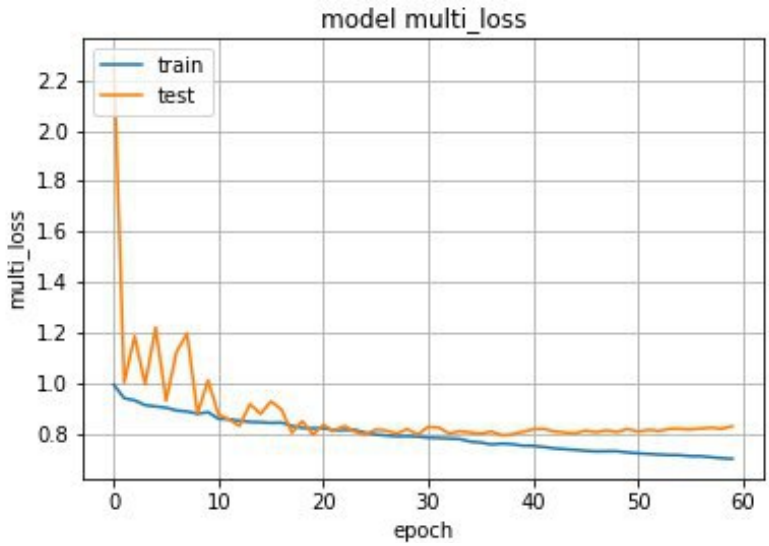

Рисунок 5.11 — Графік лос-функції однієї з моделей

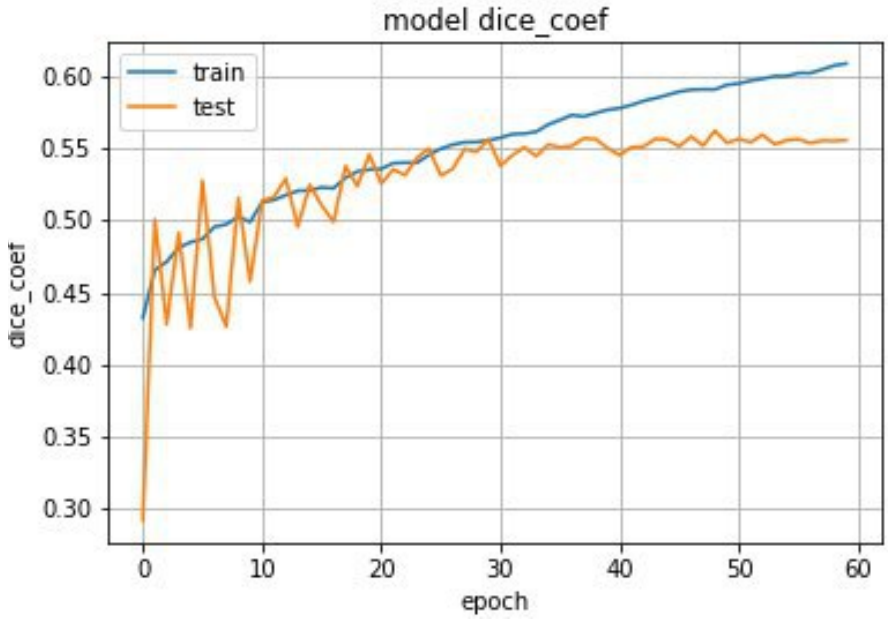

Рисунок 5.12 — Графік коефіцієнта Дайса

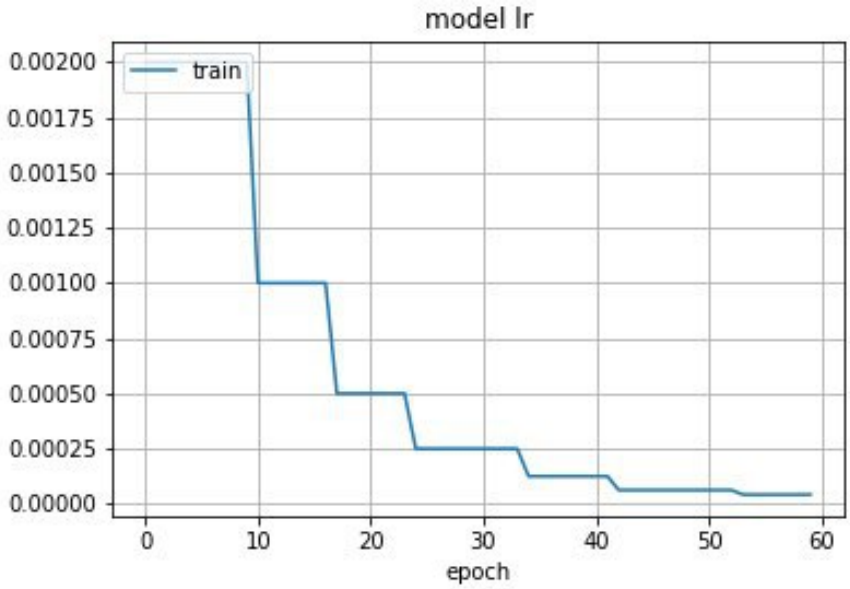

Рисунок 5.13 — Графік коефіцієнту навчання

Оскільки нейронна мережа видає ймовірність, для того, щоб привести цю ймовірність в 0 або 1 було використано пошук оптимального трешхолду. Визначено, що при ймовірності > 0.5 піксель буде рахуватися екземпляром конкретного класу, без менше — фоном:

$$pixel \geq 0.5 \rightarrow class$$
$$pixel < 0.5 \rightarrow background \quad (5.2)$$

Результат порівняння на локальній крос-валідації еталонної U-Net з

покращеною (розділ 3.3) наведено в таблиці 5.2. Як видно, покращення суттєво збільшує точність розпізнавання.

Таблиця 5.2 — Результат тестування моделей U-Net з та без покращення

№	Backbone-мережа	Локальний результат крос-валідації	
		Без покращення	З покращенням
1	ResNet-34	0.5594	0.5616
2	ResNet-50	0.5603	0.5628

Результати тестування різних моделей наведено в таблиці 5.3. Оскільки платформа kaggle дозволяє зробити лише обмежену кількість сабмітів, не всі результати були оцінені платформою kaggle. Результат на kaggle відсилався без використання TTA.

Таблиця 5.3 — Результати тестування різних моделей.

№	Тип мережі	Backbone-мережа	Локальний результат крос-валідації	Результат, оцінений платформою kaggle
1	U-Net	ResNet-34	0.5616	0.647
2	U-Net	ResNet-50	0.5628	0.648
3	U-Net	ResNet-101	0.5599	-
4	U-Net	ResNext-50	0.5619	-
5	U-Net	DenseNet-56	0.5616	0.648
6	U-Net	SeRersNet-34	0.5617	-
7	U-Net	SeResNet-50	0.5616	0.649
8	FPN	ResNet-34	0.5678	0.650
9	FPN	ResNet-50	0.5693	-
10	FPN	ResNet-101	0.5624	-
11	FPN	ResNext-50	0.5702	0.651
12	FPN	DenseNet-56	0.5675	-
13	FPN	SeRersNet-34	0.5694	0.649
14	FPN	SeResNet-50	0.5706	0.650

Як видно з таблиці, використання більш важких моделей, таких як ResNet-101 не покращує результат. В результаті у фінальні моделі були обрані всі вище наведені

моделі без ResNet-101. Фінальний результат обраховувався вже з використанням TTA та з узагальненням результату.

Результати сабміту з використанням TTA та без наведено в таблиці 5.4.

Таблиця 5.4 — Результати ансамблю моделей

Результат, оцінений платформою kaggle	
Без TTA	З TTA
0.65284	0.65371

Ансамбль моделей займав 264 місце на відкритій частині даних. Після закінчення змагання результат піднявся до 179 місця (рисунок 5.14). (до призового місця було 15 місць), що говорить по те, що результат був стабільний та не переобучився на тренувальному наборі даних.

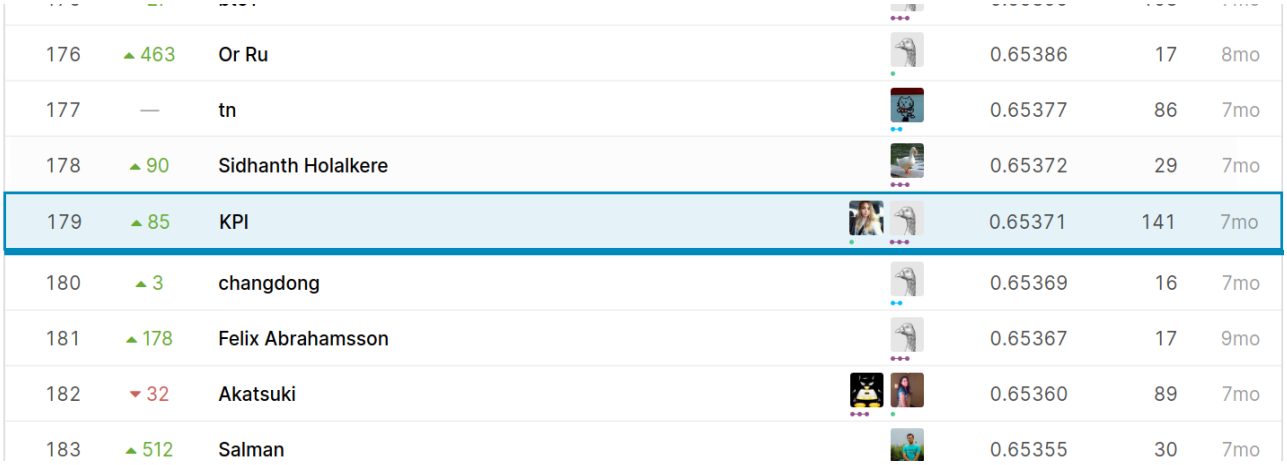

Рисунок 5.14 — Результат на приватній частині даних

Аналіз описів кращих результатів на форумі kaggle виявив, що всі дії були вірними. Кращі результати базувалися на більш високих розмірностях вхідних даних, що в рази збільшує час тренування та вимоги до обладнання.

5.5 Висновки до розділу

В цьому розділі було описано змагання на кому проводилося тестування, набір даних на якому модель навчалася і параметри навчання. Також було розглянуто основні методи аугментації, наведено ілюстративні матеріали до кожного з них та

таблицю з ймовірностями використання кожного з методів.

Також розглядається аугментація під час тесту та моделі які використовувалися. Надаються результати тестування моделей у вигляді таблиці.

6. ПРОГРАМНА РЕАЛІЗАЦІЯ

6.1 Засоби розробки

Під час розробки системи було використано такі мови програмування: Python та Dart; бібліотеки: Keras, TensorFlow, NumPy, Matplotlib; фреймворки: Flask, Flutter.

Python — це високорівнева мова програмування з динамічною типізацією. Основна перевага python в тому, що використання її дозволяє збільшувати продуктивність розробника, а також зробити код більш зрозумілим для читання. Синтаксис мови є досить мінімалістичним, але не зважаючи на це, стандартна бібіліоттека містить в собі велику кількість функцій.

Python підтримує низку парадигм програмування, таких як: об'єктно-орієнтоване, структурне, імперативне, функціональне і аспектно-орієнтоване програмування.

Основні архітектурні риси — динамічна типізація, автоматичне керування пам'яттю, повна інтроспекція, механізм обробки виключень, підтримка багатопоточних обчислень, високорівневі структури даних. Підтримується розбиття програм на модулі, які, в свою чергу, можуть об'єднуватися в пакети.

Python — стабільна то поширена мова програмування. Ця мова програмування використовується багатьма великими компаніями як основна мова розробки, а також як мова для створення прототипів для майбутніх проектів.

При розробці даної системи, використовувся з бібліотеками Keras, NumPy, TensorFlow, Matplotlib для розробки та тренування моделі для аналізу зображень.

Keras — відкрита нейромережева бібліотека, написана мовою Python. ЇЇ було розроблено, як частину дослідницького проекту і в подальшому вона використовується для уможливлення швидких експериментів з глибинними нейронними мережами. Основні її переваги це гнучкість, модульність та розширюваність. Ця бібліотека містить в собі ряд поширених нейромережевіх компонентів, особливо зручно працювати з задачами пов’язаними з аналізом

зображень та тексту. На додачу до стандартних глибинних нейронних мереж, Keras містить підтримку згорткових нейронних мереж.

TensorFlow — відкрита програмна бібліотека, була створена компанією Google, для виконня задач, під час вирішення яких необхідна розробка та тренування нейронних мереж.

TensorFlow реалізує через графи потоків даних (data flow graphs) алгоритми чисельних обчислень і надає нам доступ до них. В таких графах є ребра та вузли. Точки входу/виходу, або математичні операції реалізовуються вузлами. Між вузлами перетікають багатовимірні масиви даних, які представляють ребра. Є можливість закріпляти вузли за певними пристроями в результаті чого вони будуть виконуватися асинхронно. Вузли будуть паралельно обробляти тензори які до них весь час надходять, це дозволяє вузлам нейронної мережі працювати одночасно.

Найчастіше використовується для роботи з текстом, аналізом зображень, визначень спаму, та поєднання зображення з відео.

NumPy — бібліотека для Python, яка в собі містить доступ до великих багатовимірних масивів та матриці, а також до великої кількості високорівневих математичних функцій які можна застосовувати на цих масивах.

Це розширення з’явилося в зв’язку з тим, що Python як інтерпретована мова програмування опрацьовує математичні алгоритми повільніше ніж компільовані мови такі як C та Java. Як раз щоб спробувати вирішити цю проблему і було створено NumPy, який підтримує багатовимірні масиви та функції для роботи з ними тим самим вирішуючи проблему для великої кількості математичних алгоритмів. Виходячи з цього в основному будь-який алгоритм можна виразити у вигляді послідовності операцій над масивами та матрицями при цьому зберігаючи таку саму швидкість виконання як у аналогічного коду на C.

Matplotlib — розширення для Python, яке надаэ можливість візуалізовувати дані за допомогою 2D та 3D графіки. Отримані зображення часто використовують як ілюстрації до наукових публікацій.

Matplotlib є гнучким, легко конфігурованим пакетом, який разом з NumPy, SciPy і IPython надає можливості, подібні до MATLAB.

Flask — мікрофреймворк для веб-додатків, створений з використанням Python. Оскільки Flask не потребує спеціальних засобів чи бібліотек його прийнято називати мікрофрейморком. Він не підримує абстракції для роботи з базами данних та не не підтримує багато інших компонентів які надаються більш популярними бібілотеками. Проте, у Flask існують розширення які дозволяють встановлювати об'єктно-реляційні зв'язки, перевіряти форми, контролювати процес завантаження, підтримувати різноманітні відкриті технологій аутентифікації та декілька поширених засобів для фреймворку.

Dart — мова програмування для створення крос-платформних додатків, але є можливість програмування для Веб. Розробником Dart є компанія Google. Вважається, що в майбутньому Dart цілком можливо стане більш зручною заміною JavaScript, який має такі недоліки як відсутність розширюваності, недостатня продуктивність та відсутність підтримки розробки складних застосунків. Dart має Java-подібний синтаксис, може використовуватись як для створення клієнтських додатків, так і для серверних.

Flutter — це фреймворк з відкритим кодом для крос-платформної розробки. Як і Dart його розробила компанія Google. Весь графічний інтерфейс Google Fuchsia створено за допомогою Flutter, а також він є основним інструментом створення додатків для цієї ОС.

Архітектура Flutter відрізняється від інших програмних каркасів (React, Apache Cordova) тим, що він не використовує для побудови інтерфейсу мови HTML, CSS та Javascript, відповідно і вбудований рушій WebView.

Flutter підтримує тільки одну мову програмування Dart.

6.2 Серверна частина

Під час розробки застосунку було розгорнуто веб-сервер на базі Flask, на який було поміщено натреновану модель. Таким чином вдалося розвантажити пристрій, на якому застосунок виконується.

Взаємодія між клієнтською та серверною частиною відбувається наступним чином:

1. Зображення поміщається в POST запит, який відправляється на сервер для аналізу;
2. Проводиться аналіз моделлю, яка розміщена на сервері;
3. Сервер надсилає у відповідь архів з чотирма зображеннями, які є результатом аналізу.

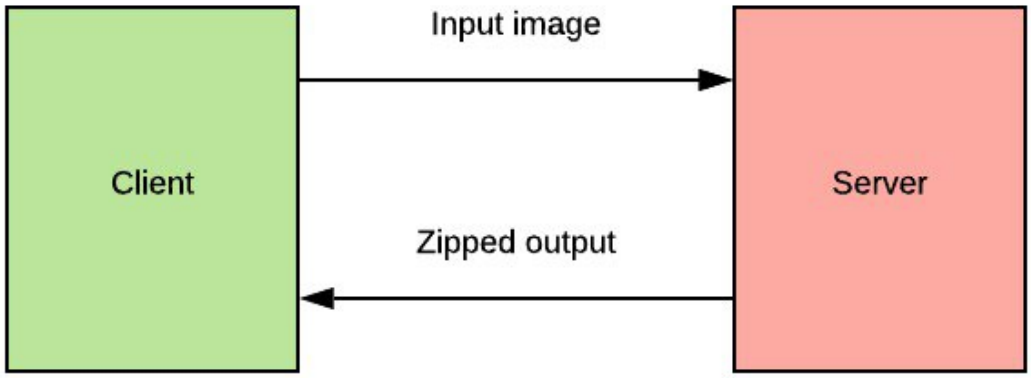

Рисунок 6.1 — Ілюстрація взаємодії клієнт-сервер

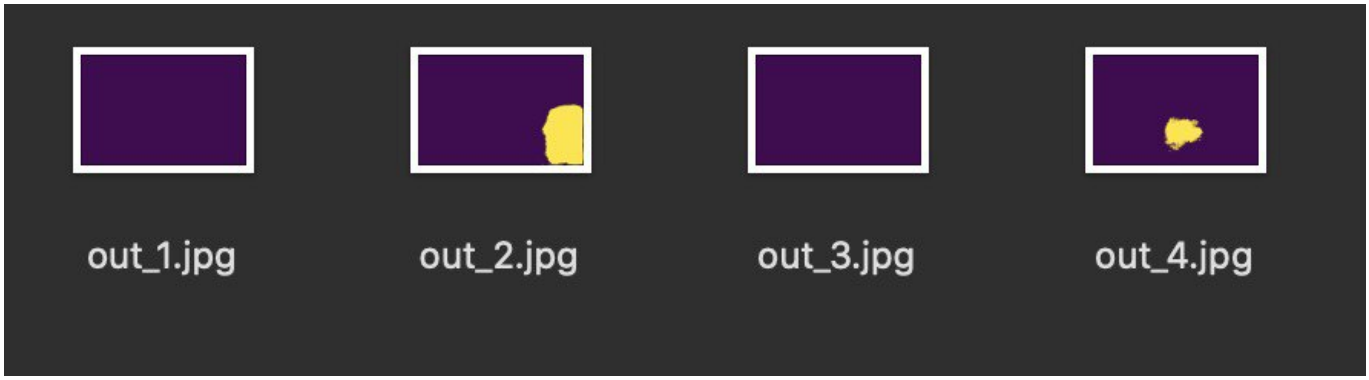

Рисунок 6.2 — Вміст архіву надісланного сервером у відповідь.

6.3 Користувацький інтерфейс

Для демонстрації працездатності системи було розроблено користувацький інтерфейс. Дизайн інтерфейсу є досить мінімалістичним, але повністю покриває всі функції системи.

Розроблений інтерфейс дозволяє користувачам обрати зображення для аналізу, завантажити його, та відобразити результат.

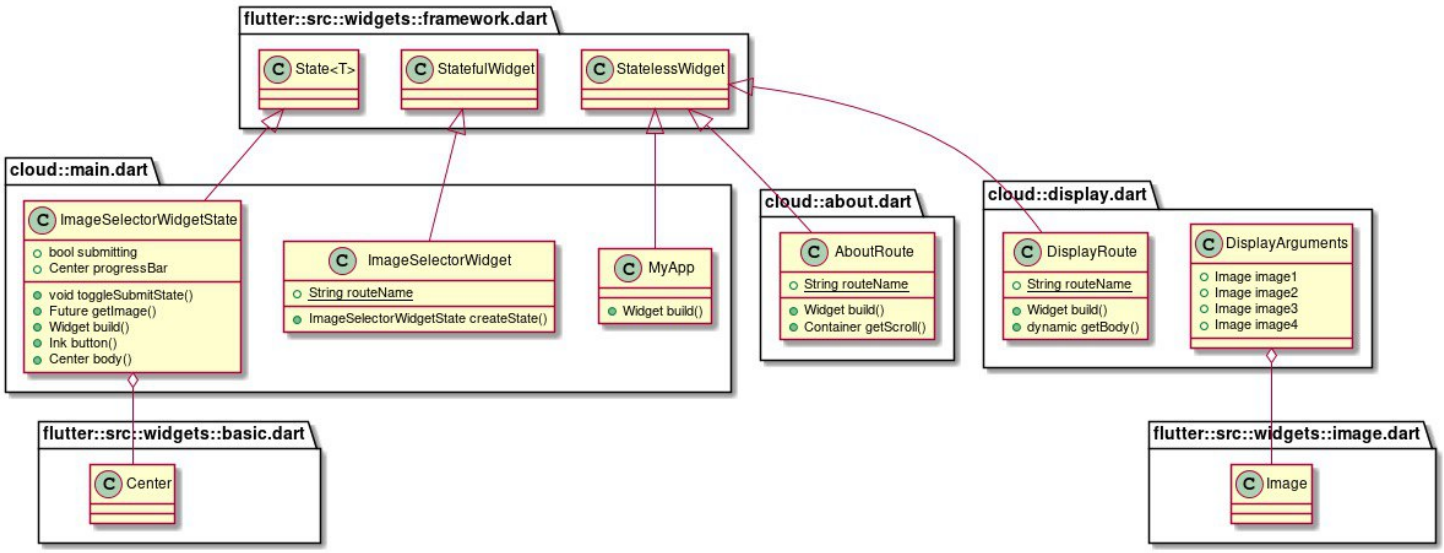

Рисунок 6.3 — Діаграма класів застосунку

Під час розробки використовувалися Flutter та Dart. Демонстрація відбувається на веб-версії застосунку, але також є варіант з мобільною версією застосунку.

Користувацький інтерфейс складається з трьох сторінок:

1. Початкова сторінка. Містить кнопку для вибору зображення для аналізу (Рисунок 6.4)

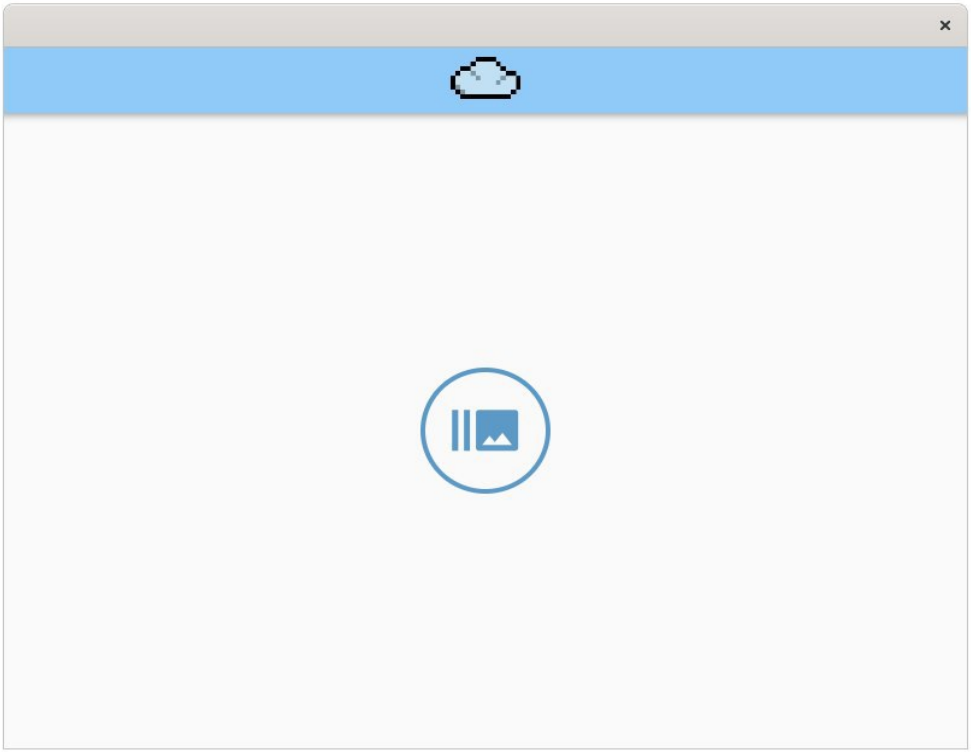

Рисунок 6.4 — Початкова сторінка

2. Сторінка очікування результату від сервера. Містить ProgressBar (Рисунок 6.5)

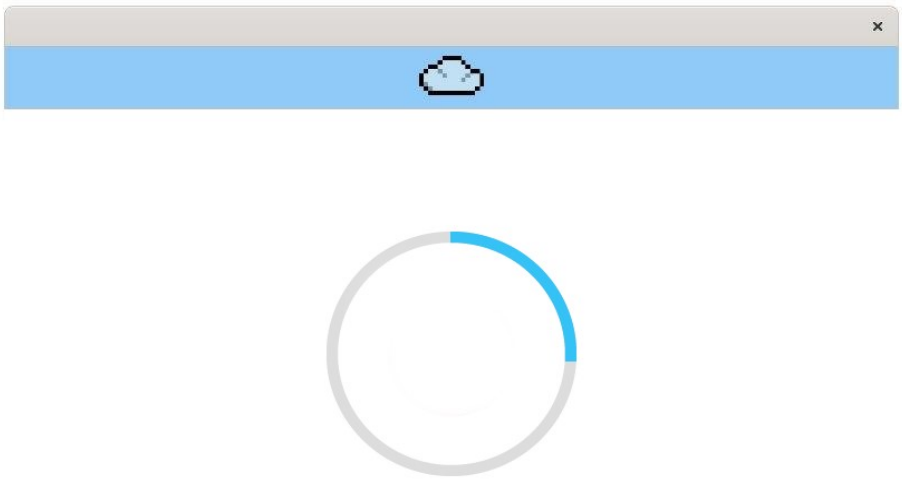

Рис. 6.5 — Сторінка очікування

3. Сторінка з відображенням результату. Містить результат — чотири зображення, та кнопку, при натисканні на яку користувач потрапляє на початкову сторінку і має змогу обрати нове зображення .(Рисунок 6.6)

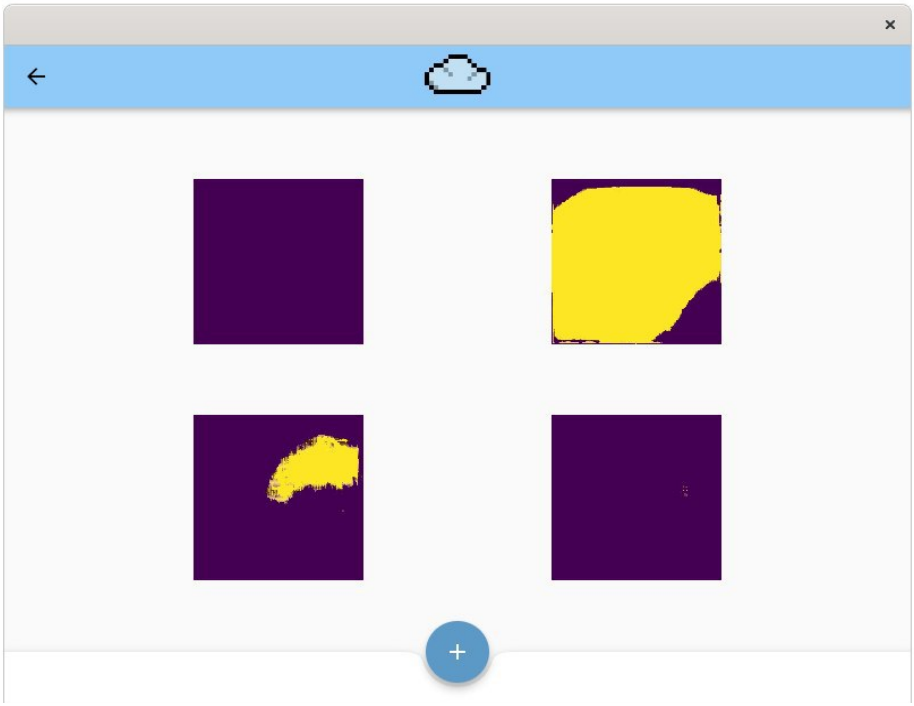

Рисунок 6.6 — Сторінка відображення результату

6.4 Висновки до розділу

В цьому розділі було описано засоби, які використовувалися для розробки системи. Було описано серверну та користувацьку частини. Описано як відбувається взаємодія з сервером, на який поміщено натреновану модель, та продемонстровано який вміст сервер надсилає у відповідь на запит.

Під час опису клієнтської частини було побудовано діаграму класів, а також уло наведено ілюстрації для основних екранів застосунку.

ВИСНОВКИ

Роботу присвячено актуальному напрямку розпізнавання зображень за допомогою штучного інтелекту. Задачі комп'ютерного зору є дуже популярним напрямком, який використовується в багатьох галузях.

При вирішенні поставлених задач було отримано наступні результати:

- було проведено аналіз існуючих алгоритмів семантичної сегментації;
- було побудовано алгоритм аналізу супутникових знімків на основі семантичної сегментації;
- зроблено практичну реалізацію алгоритму аналізу супутникових знімків на основі семантичної сегментації;
- розроблено та реалізовано додаток із зручним для користувача графічним інтерфейсом для демонстрації роботи системи;
- проведено практичні дослідження роботи створеної системи;

Для тестування був обраний конкурс, який проходив на платформі Kaggle [19] Understanding Clouds from Satellite Images [20].

Під час розробки проекту було використано такі мови програмування: Python та Dart; бібліотеки: Keras, TensorFlow, NumPy, Matplotlib; фреймворки: Flask, Flutter.

Створена система приймає на вхід зображення, виконує аналіз супутникових знімків на основі семантичної сегментації, і подає на вихід чотири зображення з результатом обробки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ПІДВИЩЕННЯ ТОЧНОСТІ СЕМАНТИЧНОЇ СЕГМЕНТАЦІЇ https://tef.kpi.ua/files/pdf/tom2-tezy-fin_2080.pdf

2. О. Й. Піцун, «Методи і засоби опрацювання біомедичних зображень в системах автоматизованої мікроскопії »: дис. … канд. техн. наук : 05.13.23 / Тернопіл. нац. екон. ун-т, нац. ун-т «Львів. політехніка». Львів, 2018, 166 с.

3. Sujatha P. Performance Analysis of Different Edge Detection Techniques for Image Segmentation / P. Sujatha, K. K. Sudha // Indian Journal of Science and Technology. – 2015. – Vol 8(14), DOI: 10.17485/ijst/2015/v8i14/72946

4. Prewitt J. M. S. Object enhancement and extraction / J. Prewitt, B. Lipkin, A. Rosenfeld // Picture Processing and Psychopictorics. - Eds. New York: Academic, 1970. – P. 75-149.

5. Faliu Yi Image segmentation: A survey of graph-cut methods / Faliu Yi, Inkyu Moon // International Conference on Systems and Informatics (ICSAI2012) Yantai, China. – 2012. – P. 1936-1941.

6. Manisha Bhagwat, R. K. Krishna & Vivek Pise: «GSimplified Watershed Transformation», *International Journal of Computer Science & Communication*, Vol. 1, No. 1, January-June 2010, pp. 175–177

7. Long, J.; Shelhamer, E.; Darrell, T. (2014). «Fully convolutional networks for semantic segmentation». arXiv:1411.4038 [cs.CV].

8. Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla (2015). «SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation» arXiv:1511.00561 [cs.CV]

9. Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, Serge Belongie (2016). Feature Pyramid Networks for Object Detection arXiv:1612.03144 [cs.CV]

10. O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *MIC-CAI*, pages 234–241. Springer, 2015.

11. Yao, Wei; Zeng, Zhigang; Lian, Cheng; Tang, Huiming (2018-10-27). Pixel-wise regression using U-Net and its application on pansharpening. Neurocomputing (en)

312: 364–371. ISSN 0925-2312. doi:10.1016/j.neucom.2018.05.103.

12. Çiçek, Özgün; Abdulkadir, Ahmed; Lienkamp, Soeren (2016). «3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation». arXiv:1606.06650 [cs.CV].

13. Iglovikov, Vladimir; Shvets, Alexey (2018). «TernausNet: U-Net with VGG11 Encoder Pre-Trained on ImageNet for Image Segmentation». arXiv:1801.05746 [cs.CV].

14. Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, Hartwig Adam (2018). «Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation». arXiv:1802.02611v3 [cs.CV].

15. Segmentation models with pretrained backbones. Keras and TensorFlow Keras. https://github.com/qubvel/segmentation_models

16. Fausto Milletari, Nassir Navab, Seyed-Ahmad Ahmadi (2016). V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation arXiv:1606.04797 [cs.CV]

17. Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár (2017). Focal Loss for Dense Object Detection arXiv:1708.02002 [cs.CV]

18. Seyed Sadegh Mohseni Salehi, Deniz Erdogmus, Ali Gholipour (2017). Tversky loss function for image segmentation using 3D fully convolutional deep networks arXiv:1706.05721 [cs.CV]

19. Kaggle https://www.kaggle.com/

20. Understanding Clouds from Satellite Images (Kaggle) https://www.kaggle.com/c/understanding_cloud_organization

ДОДАТОК А

Аналіз супутникових знімків на основі семантичної сегментації

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ6290_20Б

Аркушів 1

Київ 2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ6290_20Б	Записка.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ6290_20Б 12-1	CloudRecognition.exe	Викоувний застосунок

ДОДАТОК Б

Аналіз супутникових знімків на основі семантичної сегментації

Лістинг програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ6290_20Б

Аркушів 10

Київ 2020

Код мережі з файлу unet_with_dilation.py:

```
from keras_applications import get_submodules_from_kwargs

from ._common_blocks import Conv2dBn
from ._utils import freeze_model
from ..backbones.backbones_factory import Backbones

backend = None
layers = None
models = None
keras_utils = None

# ---------------------------------------------------------------------
#  Utility functions
# ---------------------------------------------------------------------

def get_submodules():
    return {
        'backend': backend,
        'models': models,
        'layers': layers,
        'utils': keras_utils,
    }

# ---------------------------------------------------------------------
#  Blocks
# ---------------------------------------------------------------------

def Conv3x3BnReLU(filters, use_batchnorm, name=None):
    kwargs = get_submodules()

    def wrapper(input_tensor):
        return Conv2dBn(
            filters,
            kernel_size=3,
            activation='relu',
            kernel_initializer='he_uniform',
            padding='same',
            use_batchnorm=use_batchnorm,
            name=name,
            **kwargs
        )(input_tensor)

    return wrapper

def DecoderUpsamplingX2Block(filters, stage, use_batchnorm=False):
    up_name = 'decoder_stage{}_upsampling'.format(stage)
    conv1_name = 'decoder_stage{}a'.format(stage)
    conv2_name = 'decoder_stage{}b'.format(stage)
    concat_name = 'decoder_stage{}_concat'.format(stage)

    concat_axis = 3 if backend.image_data_format() == 'channels_last' else 1

    def wrapper(input_tensor, skip=None):
```

```
        x = layers.UpSampling2D(size=2, name=up_name)(input_tensor)

        if skip is not None:
            x = layers.Concatenate(axis=concat_axis, name=concat_name)([x, skip])

        x = Conv3x3BnReLU(filters, use_batchnorm, name=conv1_name)(x)
        x = Conv3x3BnReLU(filters, use_batchnorm, name=conv2_name)(x)

        return x

    return wrapper

def DecoderTransposeX2Block(filters, stage, use_batchnorm=False):
    transp_name = 'decoder_stage{}a_transpose'.format(stage)
    bn_name = 'decoder_stage{}a_bn'.format(stage)
    relu_name = 'decoder_stage{}a_relu'.format(stage)
    conv_block_name = 'decoder_stage{}b'.format(stage)
    concat_name = 'decoder_stage{}_concat'.format(stage)

    concat_axis = bn_axis = 3 if backend.image_data_format() == 'channels_last' else 1

    def layer(input_tensor, skip=None):

        x = layers.Conv2DTranspose(
            filters,
            kernel_size=(4, 4),
            strides=(2, 2),
            padding='same',
            name=transp_name,
            use_bias=not use_batchnorm,
        )(input_tensor)

        if use_batchnorm:
            x = layers.BatchNormalization(axis=bn_axis, name=bn_name)(x)

        x = layers.Activation('relu', name=relu_name)(x)

        if skip is not None:
            x = layers.Concatenate(axis=concat_axis, name=concat_name)([x, skip])

        x = Conv3x3BnReLU(filters, use_batchnorm, name=conv_block_name)(x)

        return x

    return layer


# ---------------------------------------------------------------------
#  Unet Decoder
# ---------------------------------------------------------------------

def build_unet(
        backbone,
        decoder_block,
        skip_connection_layers,
```

```
        decoder_filters=(256, 128, 64, 32, 16),
        n_upsample_blocks=5,
        classes=1,
        activation='sigmoid',
        use_batchnorm=True,
        dilation_rates_list = [],
):
    input_ = backbone.input
    x = backbone.output

    #add dilation list
    if dilation_rates_list:
        dilated_layers_list = [x]
        for i in dilation_rates_list:
            x = layers.Conv2D(decoder_filters[0], 3,
                    activation='relu', padding='same', dilation_rate=i, name='dilation_' + str(i))(x)
            dilated_layers_list.append(x)

        x = layers.Concatenate()(dilated_layers_list)

    # extract skip connections
    skips = ([backbone.get_layer(name=i).output if isinstance(i, str)
              else backbone.get_layer(index=i).output for i in skip_connection_layers])

    # add center block if prev

ious operation was maxpooling (for vgg models)
    if isinstance(backbone.layers[-1], layers.MaxPooling2D):
        x = Conv3x3BnReLU(512, use_batchnorm, name='center_block1')(x)
        x = Conv3x3BnReLU(512, use_batchnorm, name='center_block2')(x)

    # building decoder blocks
    for i in range(n_upsample_blocks):

        if i < len(skips):
            skip = skips[i]
        else:
            skip = None

        x = decoder_block(decoder_filters[i], stage=i, use_batchnorm=use_batchnorm)(x, skip)

    # model head (define number of output classes)
    x = layers.Conv2D(
        filters=classes,
        kernel_size=(3, 3),
        padding='same',
        use_bias=True,
        kernel_initializer='glorot_uniform',
        name='final_conv',
    )(x)
    x = layers.Activation(activation, name=activation)(x)

    # create keras model instance
    model = models.Model(input_, x)
```

```
    return model

# ---------------------------------------------------------------------
#  Unet Model
# ---------------------------------------------------------------------

def Unet(
        backbone_name='vgg16',
        input_shape=(None, None, 3),
        classes=1,
        activation='sigmoid',
        weights=None,
        encoder_weights='imagenet',
        encoder_freeze=False,
        encoder_features='default',
        decoder_block_type='upsampling',
        decoder_filters=(256, 128, 64, 32, 16),
        decoder_use_batchnorm=True,
        dilation_rates_list = [],
        **kwargs
):
    """ Unet_ is a fully convolution neural network for image semantic segmentation

    Args:
        backbone_name: name of classification model (without last dense layers) used as feature
            extractor to build segmentation model.
        input_shape: shape of input data/image `(H, W, C)`, in general
            case you do not need to set `H` and `W` shapes, just pass `(None, None, C)` to make your model be
            able to process images af any size, but `H` and `W` of input images should be divisible by factor `32`.
        classes: a number of classes for output (output shape - ``(h, w, classes)``).
        activation: name of one of `keras.activations` for last model layer
            (e.g. `sigmoid`, `softmax`, ``linear``).
        weights: optional, path to model weights.
        encoder_weights: one of `None` (random initialization), `imagenet` (pre-training on ImageNet).
        encoder_freeze: if `True` set all layers of encoder (backbone model) as non-trainable.
        encoder_features: a list of layer numbers or names starting from top of the model.
            Each of these layers will be concatenated with corresponding decoder block. If `default` is used
            layer names are taken from `DEFAULT_SKIP_CONNECTIONS`.
        decoder_block_type: one of blocks with following layers structure:

            - upsampling:  `UpSampling2D` -> `Conv2D` -> `Conv2D`
            - transpose:   `Transpose2D` -> `Conv2D`

        decoder_filters: list of numbers of `Conv2D` layer filters in decoder blocks
        decoder_use_batchnorm: if `True`, `BatchNormalisation` layer between `Conv2D` and `Activation` layers
            is used.

    Returns:
        `keras.models.Model`: Unet

    .. _Unet:
        https://arxiv.org/pdf/1505.04597

    """
```

```
    global backend, layers, models, keras_utils
    backend, layers, models, keras_utils = get_submodules_from_kwargs(kwargs)

    if decoder_block_type == 'upsampling':
        decoder_block = DecoderUpsamplingX2Block
    elif decoder_block_type == 'transpose':
        decoder_block = DecoderTransposeX2Block
    else:
        raise ValueError('Decoder block type should be in ("upsampling", "transpose"). '
                         'Got: {}'.format(decoder_block_type))

    backbone = Backbones.get_backbone(
        backbone_name,
        input_shape=input_shape,
        weights=encoder_weights,
        include_top=False,
        **kwargs,
    )

    if encoder_features == 'default':
        encoder_features = Backbones.get_featu
re_layers(backbone_name, n=4)

    model = build_unet(
        backbone=backbone,
        decoder_block=decoder_block,
        skip_connection_layers=encoder_features,
        decoder_filters=decoder_filters,
        classes=classes,
        activation=activation,
        n_upsample_blocks=len(decoder_filters),
        use_batchnorm=decoder_use_batchnorm,
        dilation_rates_list = dilation_rates_list,
    )

    # lock encoder weights for fine-tuning
    if encoder_freeze:
        freeze_model(backbone, **kwargs)

    # loading model weights
    if weights is not None:
        model.load_weights(weights)

    return model
```

Код серверної частини з файлу server.py:

```
import io
import zipfile
import cv2
import imageio
import keras
import numpy as np
```

```
from flask import Flask, request, send_file
import matplotlib.image as mpimg
app = Flask(__name__)

app.secret_key = "super secret key"

class MetaSingleton(type):

    _instances = {}

    def __call__(cls, *args, **kwargs):

        if cls not in cls._instances:

            cls._instances[cls] = super(MetaSingleton, cls).__call__(*args, **kwargs)

        return cls._instances[cls]

class Model(metaclass=MetaSingleton):

    model = None

    def get(self):

        if self.model is None:

            self.model = keras.models.load_model('model')

        return self.model

def generate_zip(files):

    mem_zip = io.BytesIO()

    with zipfile.ZipFile(mem_zip, mode="w", compression=zipfile.ZIP_DEFLATED) as zf:

        for f in files:

            zf.writestr(f[0], f[1])

    return mem_zip.getvalue()

@app.route('/', methods=['POST'])

def upload_file():

    model_my = Model().get()

    img = request.get_data()

    img = imageio.imread(img)

    img = cv2.resize(img, dsize=(480, 320), interpolation=cv2.INTER_LINEAR)

    img = img / 255
```

```
    img = np.expand_dims(img, axis=0)
    predict = model_my.predict(img)
    predict[predict >= 0.5] = 1
    predict[predict < 0.5] = 0
    file_names = ["out_1.jpg", "out_2.jpg", "out_3.jpg", "out_4.jpg"]
    files = []
    i = 1
    for f in file_names:
        buffer = io.BytesIO()
        mpimg.imsave(buffer, predict[0, ..., i - 1], format="jpg")
        file = buffer.getvalue()
        buffer.close()
        files.append((f, file))
        i += 1
    return send_file(io.BytesIO(generate_zip(files)),
                     attachment_filename='out.zip',
                     mimetype='application/zip')
if name == '__main__':
    app.secret_key = 'super secret key'
    app.run()
```

Частина коду для інтерфейсу:

файл main.dart:

```
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
```

```
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}

class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);
  final String title;
  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),

      body: Center(
        // Center is a layout widget. It takes a single child and positions it in the middle of the parent.
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'You have pushed the button this many times:'
            )
              Text(
              '$_counter',
              style: Theme.of(context).textTheme.headline4,
            ),
          ],
        ),
      ),

      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: Icon(Icons.add),
      ), // This trailing comma makes auto-formatting nicer for build methods.
```

```
    );
  }
}
```

Файл main_desktop.dart:

```
import 'package:flutter/foundation.dart'
    show debugDefaultTargetPlatformOverride;
import 'package:flutter/material.dart';
import 'main.dart' as original_main;

void main() {
  debugDefaultTargetPlatformOverride = TargetPlatform.fuchsia;
  original_main.main();
}
```

Файл upload.dart:

```
import 'dart:io';
import 'package:archive/archive.dart';
import 'package:archive/archive_io.dart';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

Future uploadImage(final File imageFile) async {
  final http.Response response = await http.post("http://185.67.1.180:5000/",
      body: imageFile.readAsBytesSync());
  final Archive archive = ZipDecoder().decodeBytes(response.bodyBytes);
  final List<Image> list = new List();
  archive.forEach((file) => list.add(
      Image.memory(file.content, width: 100, height: 100, fit: BoxFit.cover)));
  return list;
}
```

Файл display.dart:

```
import 'package:cloud/main.dart';
import 'package:flutter/material.dart';
import 'drawer.dart';

@immutable

class DisplayArguments {
  final Image image1;
  final Image image2;
  final Image image3;
  final Image image4;

  DisplayArguments(this.image1, this.image2, this.image3, this.image4);
}

const buttonColor = const Color(0xFF5D99C6);

class DisplayRoute extends StatelessWidget {
  static const routeName = '/display';
```

```
 @override
 Widget build(BuildContext context) {
  final DisplayArguments displayArguments =
      ModalRoute.of(context).settings.arguments;

  return Scaffold(
   appBar: applicationBar(),
   body: getBody(context, displayArguments),
   drawer: getDrawer(context),
   bottomNavigationBar: BottomAppBar(
    shape: const CircularNotchedRectangle(),
    child: Container(
     height: 50.0,
    ),
   ),
   floatingActionButton: FloatingActionButton(
    onPressed: () =>
        Navigator.pushNamed(context, ImageSelectorWidget.routeName),
    tooltip: 'Add new',
    backgroundColor: buttonColor,
    child: Icon(
     Icons.add,
     color: Colors.white,
    ),
   ),
   floatingActionButtonLocation: FloatingActionButtonLocation.centerDocked,
  );
 }

 getBody(final BuildContext context, final DisplayArguments displayArguments) {
  return Column(
   mainAxisAlignment: MainAxisAlignment.spaceEvenly,
   children: [
    Container(
       child: new Row(
     mainAxisAlignment: MainAxisAlignment.spaceEvenly,
     children: <Widget>[
      displayArguments.image1,
      displayArguments.image2,
     ],
    )),
    Container(
       child: new Row(
     mainAxisAlignment: MainAxisAlignment.spaceEvenly,
     children: <Widget>[
      displayArguments.image3,
      displayArguments.image4,
     ],
    )),
   ],
  );
 }
}
```

ДОДАТОК В

Аналіз супутникових знімків на основі семантичної сегментації

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ6290_20Б

Аркушів 8

Київ 2020

АНОТАЦІЯ

Даний додаток містить опис програмної системи розробленої для підготовки набору даних, навчання мережі для сегментації та обробку результатів. Створена програмна система показує процес обробки вхідних даних та обробку результатів передбачення.

Для розробки було використано мови програмування: Python, Dart; бібліотеки: Keras, TensorFlow, NumPy, Matplotlib; фреймворки: Flask, Flutter.

ЗМІСТ

1. ЗАГАЛЬНІ ВІДОМОСТІ 79
2. ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ..... 80
3. ОПИС ЛОГІЧНОЇ СТРУКТУРИ 81
4. ТЕХНІЧНІ ЗАСОБИ, ЩО ВИКОРИСТОВУЮТЬСЯ 82
5. ВИКЛИК І ЗАВАНТАЖЕННЯ..... 83
6. ВХІДНІ ТА ВИХІДНІ ДАНІ 84

ЗАГАЛЬНІ ВІДОМОСТІ

У додатку розглядається застосунок та модуль для навчання нейронної мережі з кодом УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТІ6290_20Б, що міститься у файлі unet_with_dilation.py.

Модуль реалізовано за допомогою бібліотеки Keras та мови програмування Python.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Навчання нейронної мережі відбувається на наборі даних, який було вято з kaggle.

Після проведення передбачення мережею його потрібно перетворити до зображення для відображення користувачу. На виході мережі ми отримуємо маску передбачень належності пікселя зображення до того чи іншого пікселя. Модель виконує перетворення маски передбачення на зображення маски та її накладання на вхідне зображення для зручного аналізу.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Користувач через застосунок комунікує з сервером на який поміщено натреновану модель. За допомогою моделі, яку розміщнено на сервері відбувається аналіз зображення. Користувач обирає зобюраження з папки на пристрої після чого надсилається POST запит після чого отримується у відпоівдь архів з 4 зображеннями, який розпаковується та зображення відображаються у застосунку.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для повноцінного використання застосунку користувач повинен мати ПК із активним підключенням до мережі інтернет на базі будь якої оперативної системи. Робота застосунку в офлайн режимі не є можливою, оскільки необхідна комунікація з сервером, але є можливість запуску на різних видах пристроїв.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Для того аби встановити застосунок необхідно скопіювати EXE файл у будь-яке місце на диску, та запустити його кліком миші на іконку застосунку.

ВХІДНІ ТА ВИХІДНІ ДАНІ

Вхідними даними для застосунку є зображення (знімок з супутнику). Вихідними даними програми є 4 зображення після аналізу на яких видно області з об’єктами.