КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки на тему:

ОГЛЯД ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ ЕФЕКТИВНОСТІ МЕТОДІВ СЕГМЕНТАЦІЇ ЗОБРАЖЕНЬ

Виконав студент 4-го курсу
Ахадов Ісмаіл Гамілович _____ (підпис)

Науковий керівник:
_____ Терещенко Ярослав Васильович
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теоретичної кібернетики
«____» _____
2022 р., протокол № ___
Завідувач кафедри
проф. Терещенко В. М. _____
(підпис)

Київ – 2022

ЗМІСТ

СКОРОЧЕННЯ

ЗНМ - згорткова нейронна мережа
RGB - red, green, blue модель
FC - fully-connected
PSPNet - Pyramid Scene Parsing Network

ВСТУП

Сегментація зображень - це процес розподілу, або ж сегментування, за певним принципом об’єктів, що присутні на зображенні. Ціллю даного завдання є полегшення загального сприйняття людиною чи нейромережею за подальшої обробки зображення, оскільки на ньому будуть не різнорідні об’єкти, а чітко класифіковані та розподілені набори пікселів.

Сегментація буває семантичною, паноптичною та сегментацією екземплярів. В практичній частині буде наведено приклад створення застосунку з семантичною сегментацією.

Актуальність роботи полягає в зростаючій увазі світової ІТ-спільноти та спільноти науковців світу до сфери машинного навчання та комп’ютерного зору, в тому числі до питань розпізнавання образів та сегментації об’єктів на зображеннях. Ці завдання відіграють важливу роль не тільки в сфері відкриттів технічних, а й в нашому повсякденні, наприклад вже зараз багато медичних засобів вдало користуються технологією сегментації знімків для виявлення пухлин у пацієнтів, також дана технологія набула широкого розповсюдження в питанні освоєння космічного простору, адже допомагає сегментувати зображення поверхонь планет чи інших космічних об’єктів. Тому питання сегментації зображень наразі є важливим, тим паче питання доступу до технологій сегментації за допомогою зручних користувацьких інтерфейсів через веб-додатки, прототип чого було реалізовано в практичній частині завдання.

Метою роботи є вивчення підходів та архітектур для сегментації зображень, а також порівняння ефективності різних архітектур сегментації. В якості прикладу роботи моделі для сегментації розробити модель на одній з розглянутих архітектур (UNet).

Завдання: дослідити наявні архітектури мереж для побудови моделі для сегментації зображень, побудувати власну модель на основі однієї з архітектур, забезпечити зручний інтерфейс для взаємодії з моделлю.

Об’єкт дослідження: модель для сегментації зображень(семантичної).

Методи розроблення: технічні засоби високорівневої мови програмування Python, сторонні бібліотеки для мови програмування Python, мова розмітки тексту HTML, середовище розробки (IDE) PyCharm

Можливі сфери застосування: академічна сфера, засоби для навчання моделі, що використовується, наприклад, в автомобілях з автопілотом.

Обсяг і структура роботи: робота складається зі вступу, трьох розділів та висновку. Повний обсяг роботи складає 38 сторінок. Список використаних джерел містить 11 найменувань.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ВІДОМОСТІ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Комп’ютерний зір.

Комп'ютерний зір або комп'ютерне бачення — теорія та технологія створення машин, які можуть проводити виявлення, стеження та визначення об'єктів.

Завдяки досягненням у галузі штучного інтелекту та інноваціям у глибокому навчанні та нейронних мережах, ця галузь змогла зробити великі стрибки за останні роки та змогла перевершити людей у деяких завданнях, пов’язаних із виявленням та маркуванням об’єктів.

Одним із рушійних факторів розвитку комп’ютерного зору є кількість даних, які ми генеруємо сьогодні, які потім використовуються для навчання та покращення комп’ютерного зору.

Поряд із величезною кількістю візуальних даних (понад 3 мільярди зображень публікуються в Інтернеті щодня) тепер доступні обчислювальні потужності, необхідні для аналізу даних[1]. Оскільки область комп’ютерного зору розширюється з новим апаратним забезпеченням та алгоритмами, зростає точність ідентифікації об’єктів. Менш ніж за десятиліття сучасні системи досягли 99-відсоткової точності з 50%, що робить їх точнішими, ніж люди, у швидкому реагуванні на візуальні дані.

Історія комп’ютерного зору

Вчені та інженери вже близько 60 років намагаються розробити способи, щоб машини могли бачити й розуміти візуальні дані. Експерименти почалися в 1959 році, коли нейрофізіологи показали кішці низку зображень, намагаючись співвіднести реакцію в її мозку. Вони виявили, що вона спочатку реагує на жорсткі краї або лінії, і з наукової точки зору це означало, що обробка зображень починається з простих форм, таких як прямі краї.[2]

Приблизно в той же час була розроблена перша технологія комп’ютерного сканування зображень, яка дозволила комп’ютерам оцифровувати та

отримувати зображення. Ще одна віха була досягнута в 1963 році, коли комп’ютери змогли перетворити двовимірні зображення в тривимірні форми. У 1960-х роках штучний інтелект (ШІ) з'явився як академічна область дослідження, а також поклав початок пошукам ШІ для вирішення проблеми людського зору.

У 1974 році була запроваджена технологія оптичного розпізнавання символів (OCR), яка могла розпізнавати текст, надрукований будь-яким шрифтом або гарнітурою. [3] Аналогічно, інтелектуальне розпізнавання символів (ICR) могло розшифрувати рукописний текст за допомогою нейронних мереж. З тих пір, OCR і ICR знайшли свій шлях в обробці документів і рахунків-фактур, розпізнаванні номерних знаків транспортних засобів, мобільних платежах, машинному перекладі та інших поширених програмах.

У 1982 році нейробіолог Девід Марр встановив, що зір працює ієрархічно, і представив алгоритми для машин для виявлення країв, кутів, кривих і подібних основних форм. Одночасно вчений-комп’ютерник Куніхіко Фукусіма розробив мережу клітин, які могли розпізнавати закономірності. Мережа, яка називається Neocognitron, включала згорткові шари в нейронну мережу.

До 2000 року в центрі уваги дослідження було розпізнавання об'єктів, а до 2001 року з'явилися перші програми для розпізнавання облич у реальному часі. У 2000-х роках з’явилася стандартизація того, як набори візуальних даних позначаються та анотуються. У 2010 році став доступний набір даних ImageNet. Він містив мільйони позначених зображень у тисячі класів об’єктів і є основою для CNN і моделей глибокого навчання, які використовуються сьогодні. У 2012 році команда з Університету Торонто взяла участь CNN в конкурсі на розпізнавання зображень. Модель під назвою AlexNet значно знизила частоту помилок при розпізнаванні зображень. Після цього прориву рівень помилок впав лише до кількох відсотків.[2]

1.2. Сегментація зображень

У цифровій обробці зображень і комп’ютерному баченні сегментація зображення — це процес поділу цифрового зображення на кілька сегментів зображення, також відомих як області зображення або об’єкти зображення (набори пікселів). Мета сегментації полягає в тому, щоб спростити та/або змінити представлення зображення на щось, що є більш значущим і легшим для аналізу.[5] Сегментація зображень зазвичай використовується для визначення місця розташування об’єктів і меж (лінії, криві тощо) на зображеннях. Точніше, сегментація зображення — це процес присвоєння мітки кожному пікселю зображення таким чином, що пікселі з однаковою міткою мають певні характеристики.

Результатом сегментації зображення є набір сегментів, які разом охоплюють усе зображення, або набір контурів, витягнутих із зображення. Кожен з пікселів у регіоні подібний за певною характеристикою або обчисленою властивістю, як-от колір, інтенсивність або текстура. Суміжні області значно відрізняються за кольором щодо одних і тих же характеристик. При застосуванні до набору зображень, типового для медичної візуалізації, отримані контури після сегментації зображення можна використовувати для створення тривимірних реконструкцій за допомогою алгоритмів інтерполяції, таких як маршируючі куби.[6]

Деякі з практичних застосувань сегментації зображень:

- Пошук зображень на основі вмісту;
- Медичні зображення, включаючи об’ємні зображення з комп’ютерної та магнітно-резонансної томографії[7];
- Виявлення об'єктів[8];
    - Виявлення пішоходів;
    - Розпізнавання обличчя;
    - Виявлення стоп-сигналу;

    - Знаходження об’єктів на супутникових знімках (дороги, ліси, посіви тощо);
- Завдання на розпізнавання;
    - Розпізнавання обличчя;
    - Розпізнавання відбитків пальців;
    - Розпізнавання райдужної оболонки;
- Системи контролю дорожнього руху;
- Відеоспостереження.

Методи сегментації зображень

Методи сегментації зображень можна класифікувати на три категорії, а саме: семантична сегментація, сегментація екземплярів і паноптична сегментація.

Семантична сегментація пов’язує кожен піксель зображення з міткою класу, як-от людина, квітка, автомобіль тощо. Він розглядає кілька об’єктів одного класу як одну сутність. На відміну від цього, сегментація екземплярів розглядає кілька об’єктів одного класу як окремі окремі екземпляри.

Щоб поєднати поняття як семантичної, так і екземплярної сегментації, паноптична сегментація призначає дві мітки кожному з пікселів зображення – (i) семантична мітка (ii) ідентифікатор екземпляра. Ідентично позначені пікселі вважаються належними до одного семантичного класу, і їх ідентифікатори розрізняють його екземпляри.

Семантична сегментація та паноптична сегментація

Як семантична, так і паноптична сегментація вимагають, щоб кожному пікселю зображення було присвоєно семантичну мітку. Таким чином обидва методи подібні, якщо основна істина не визначає екземпляри або якщо всі класи є матеріалом. Однак включення класів речей (кожен з яких може мати кілька екземплярів на одне зображення) відрізняє ці завдання.

Сегментація екземплярів і паноптична сегментація

Сегментація екземплярів і паноптична сегментація сегментують кожен екземпляр об’єкта на зображенні. Однак різниця полягає в обробці сегментів, що перекриваються. Сегментація екземплярів дозволяє перекривати сегменти, тоді як завдання паноптичної сегментації дозволяє призначити унікальну семантичну мітку та унікальний ідентифікатор екземпляра кожному пікселю зображення. Отже, для паноптичної сегментації перекриття сегментів неможливе.

1.3. Нейронні мережі

1.3.1. Нейронні мережі

Нейронні мережі, також відомі як штучні нейронні мережі або змодельовані нейронні мережі , є підмножиною машинного навчання і лежать в основі алгоритмів глибокого навчання. Їх назва та структура натхненні людським мозком, імітуючи спосіб, яким біологічні нейрони сигналізують один одному.

Штучні нейронні мережі (ШНМ) складаються з шарів вузлів, які містять вхідний шар, один або кілька прихованих шарів і вихідний шар. Кожен вузол або штучний нейрон з’єднується з іншим і має відповідну вагу та поріг. Якщо вихід будь-якого окремого вузла перевищує вказане порогове значення, цей вузол активується, надсилаючи дані на наступний рівень мережі. В іншому випадку дані не передаються на наступний рівень мережі.[4]

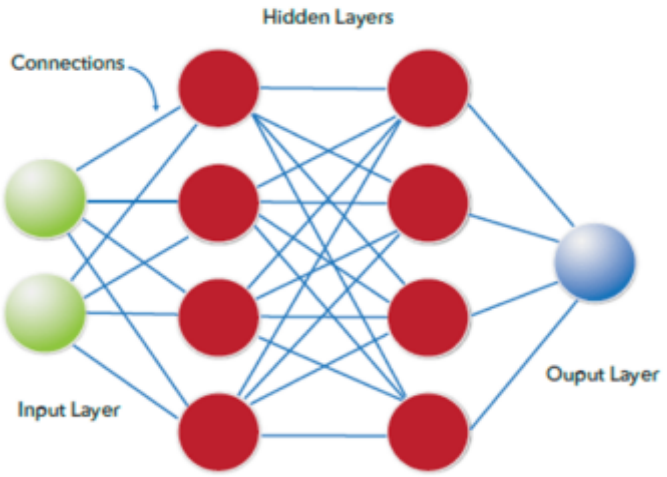

Рисунок 1.1. Архітектура нейронної мережі

Нейронні мережі покладаються на навчальні дані, щоб з часом вивчати та покращувати свою точність. Однак, як тільки ці алгоритми навчання будуть точно налаштовані на точність, вони стануть потужними інструментами в інформатиці та штучному інтелекті, що дозволить класифікувати та кластеризувати дані з високою швидкістю. Завдання розпізнавання мовлення або зображення можуть займати хвилини та години, якщо порівнювати з ручною ідентифікацією експертів. Однією з найвідоміших нейронних мереж є пошуковий алгоритм Google.

Думайте про кожен окремий вузол як про власну модель лінійної регресії, що складається з вхідних даних, вагових коефіцієнтів, зміщення (або порогу) та вихідних даних. Формула буде виглядати приблизно так:

$$\sum_{i=1}^{m} w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + \ldots + bias$$

Математична формула, яка використовується для визначення виходу

$$\text{output} = f(x) = \begin{cases} 1 \text{ if } \sum w_1 x_1 + b \geq 0 \\ 0 \text{ if } \sum w_1 x_1 + b < 0 \end{cases}$$

Рисунок 1.2. Формула для визначення виходу

Після визначення вхідного шару призначаються ваги. Ці коефіцієнти допомагають визначити важливість будь-якої даної змінної, причому більші з них роблять більш значний внесок у результат порівняно з іншими вхідними параметрами. Потім усі вхідні дані помножуються на відповідні ваги, а потім підсумовуються. Після цього вихідні дані передаються через функцію активації, яка визначає вихід. Якщо цей вихід перевищує заданий поріг, він «запускає» (або активує) вузол, передаючи дані наступному шару в мережі. Це призводить до того, що вихід одного вузла стає на вхід наступного вузла. Цей процес передачі даних від одного шару до наступного визначає цю нейронну мережу як мережу прямого зв’язку.

На практиці використання нейронних мереж, як-от розпізнавання або класифікація зображень, використовується контрольоване навчання або визначені набори даних для навчання алгоритму. Під час навчання моделі необхідно оцінити її точність за допомогою функції витрат (або втрат). Це також зазвичай називають середньоквадратичною помилкою (MSE).

Формула *MSE:*

$Cost(loss)Function = MSE = \frac{1}{2m}\sum_{i=1}^{m}(y\hat{} - y)^2$, де

i - індекс вибірки,

y-hat - це прогнозований результат,

y – фактичне значення,

m - кількість зразків.

Зрештою, мета полягає в тому, щоб мінімізувати функцію витрат, щоб забезпечити правильність відповідності для будь-якого даного спостереження. Коли модель коригує свої ваги та зміщення, вона використовує функцію вартості та навчання з підкріпленням, щоб досягти точки зближення або локального мінімуму. Процес, у якому алгоритм коригує свої ваги, здійснюється шляхом градієнтного спуску, що дозволяє моделі визначати напрямок для зменшення помилок (або мінімізації функції вартості). З кожним навчальним

прикладом параметри моделі коригуються, щоб поступово зближатися до мінімуму.

Нейронні мережі, зокрема згорткові, є важливою темою для розуміння в питання сегментації та класифікації предметів на зображеннях.

1.3.2 Згорткові нейронні мережі

Згорткові нейр.онна мережа - ЗНМ, CNN - основний інструмент для класифікації та розпізнавання певних об’єктів та зображеннях чи відеоматеріалах. Однією з помітних відзнак ЗНМ в порівнянні з іншими нейронними мережами є те, що ЗНМ демонструють значно кращу продуктивність та результати для вводу у вигляді зображень та звуку. Тобто ЗНМ є високоефективним засобом побудови додатків для розпізнавання об’єктів та їх класифікації на вхідних зображеннях. Саме тому цей підхід набув широкого поширення в задачах сегментації зображень та був використаний мною для імплементації моделі, що сегментуватиме вхідні зображення далі в практичній частині роботи.

ЗНМ складаються з трьох основних шарів:

- згортковий шар (Convolutional layer)
- агрегувальний шар (pooling layer)
- повноз'єднані шари(Fully-connected (FC) layer)

Згортковий шар являє собою перший шар ЗНМ, причому їх може бути кілька поспіль або за згортковим шаром може слідкувати агрегувальний шар. В основі згорткового шару, як і в основі всієї ЗНМ є операція згортки, в ході якої вхід з попереднього шару n * n * n_c проходить через набір фільтрів f * f * n_c, де n_c - це кількість каналів (для RGB - кількість каналів буде три, відповідно) та отримується “карта ознак” (feature map), яка переходить в наступний шар мережі. З математичної точки зору в ході згортки кожен фрагмент зображення буде перемножено на матрицю фільтра (ядра) поелементно, а результат цих

множенння просумовано і записано у відповідну позицію в результуючій матриці. Після чого аналогічні дії будуть проведено для всіх елементів вхідного зображення з певним зміщенням (stride s) від лівого верхнього кута, за зміщення s=1 фільтр буде перемножено на елементи, що знаходяться за один рядок чи одну колонку. Також, за необхідності можна задати відступи (padding p) для вхідної матриці зображення, щоб результат на виході не був замалої розмірності. Отже, можна вивести формулу, за якою в ході згортки можна визначити розмірність вихідного результату n_{out}:

$$n_{out} = \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor,$$

де n - розмірність вхідної матриці, p - додаткові відступи (padding), s - зсув (stride).

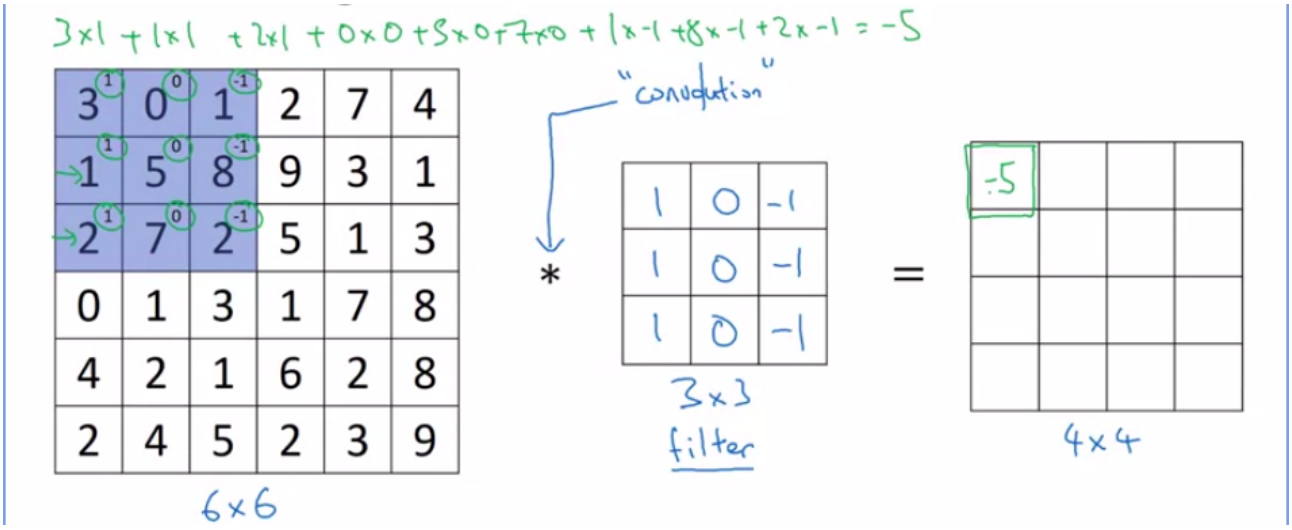

Рисунок 1.3. Приклад здійснення однієї ітерації операції згортки

Агрегувальний шар, або ж шар пулінга, слідує за шаром згортки, він допомагає зменшити загальну кількість параметрів в мережі, тим самим зменшує загальний об'єм зображення, який буде далі використовуватись та оброблятись в мережі, тобто зменшуються витрати на обробку зображення, зменшується ризик перетренування мережі та збільшується в цілому ефективність, хоча і втрачається певна кількість інформації з попереднього

шару. Операція агрегування семантично нагадує операцію згортки, бо також потребує певний набір фільтрів, але значення (ваги) в фільтрі не грають ролі в операції агрегації, тільки його розмірність. З видів агрегування (pooling) вирізняють максимізаційне (max pooling) або усереднювальне (average pooling). За максимізаційного агрегування з набору значень, що відповідають по розмірності фільтрові, обирається найбільше (максимальне), а за усереднювального використовує усереднене значення. На практиці максимізаційне агрегування показує себе краще за аналоги.

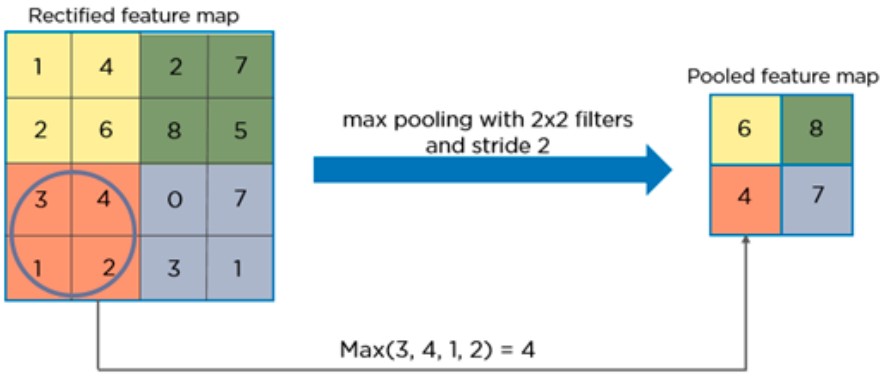

*Рисунок 1.4. Приклад максимізаційного агрегування з фільтром 2 * 2 та зсувом s = 2*

Повноз'єднані шари зазвичай слідують "наприкінці" нейронної мережі. Вони з'єднують кожен нейрон одного шару з відповідним кожним нейроном наступного шару, виконують задачу класифікації на основі вилучених ознак (features) з попередніх шарів мережі. Проте, якщо згорткові та агрегаційні шари використовують функцію активації ReLu, то повноз'єднаний шар використовує функцію softmax, за допомогою якої визначає ймовірність в рамках від 0 до 1.

РОЗДІЛ 2. ОГЛЯД АРХІТЕКТУР СЕГМЕНТАЦІЇ ЗОБРАЖЕНЬ

2.1 FCN

FCN є по суті однією з перших запропонованих моделей для наскрізної семантичної сегментації. В цій архітектурі такі стандартні моделі класифікації зображень, як VGG і AlexNet, перетворюються на повністю згорткові, замінюючи повно з'єднані шари (FC layers) на згортки розмірністю 1x1. У FCN транспоновані згортки (згортки, що збільшують розмірність на виході, а не зменшують) використовуються для upsampling (збільшення розмірності зображення до початкової, оригінальної), на відміну від інших підходів, де використовуються математичні інтерполяції. Існують три варіації цієї архітектури: FCN8, FCN16 і FCN32. У FCN8 і FCN16 використовуються з'єднання пропуску (skip connections).

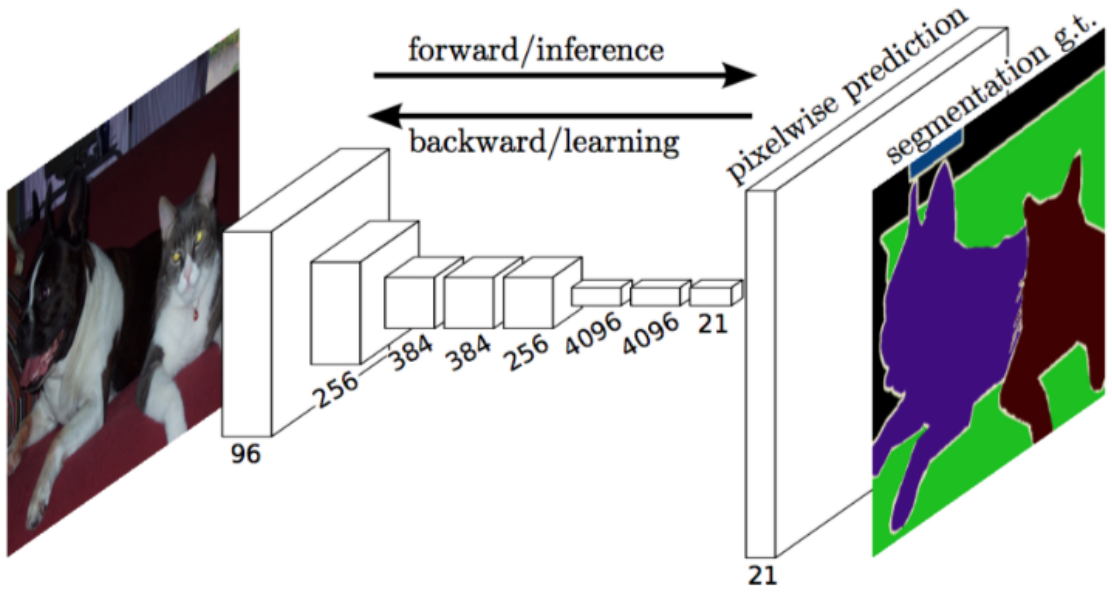

Рисунок 2.1. Приклад архітектури FCN32 з розписаними значеннями кількості елементів на кроках мережі

2.2 PSPNet

Pyramid Scene Parsing Network оптимізована для вивчення кращого загального представлення зображення. Спочатку зображення передається в базову мережу, щоб отримати карти ознак (feature map), які зменшуються (downsampling) до певних масштабів, після чого застосовується операція згортки. Після цього всі карти ознак проектуються операцією upsampling до звичного масштабу та об’єднуються разом. Врешті-решт, ще один шар згортки використовується для отримання остаточних результатів сегментації. Тут менші об’єкти добре фіксуються ознаками, змасштабованими до високої роздільної здатності, тоді як великі об’єкти захоплюються об’єктами, об’єднаними до меншого розміру.

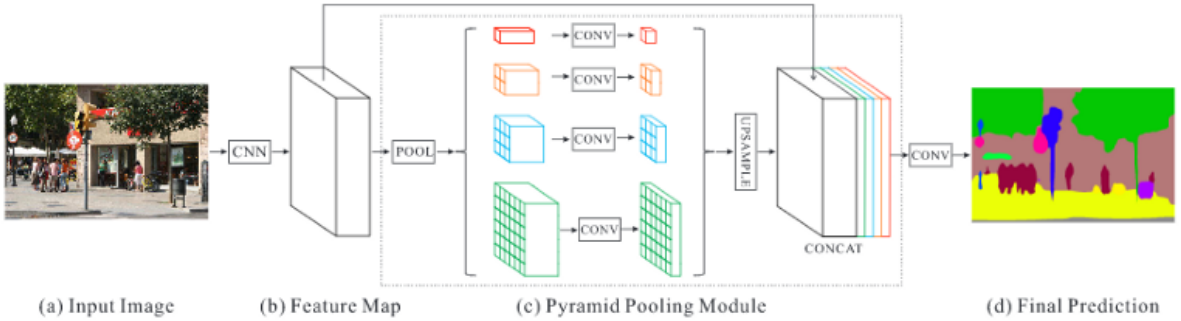

Рисунок 2.2. Діаграма архітектури PSPNet

2.3 UNet

Архітектура UNet була початково створена та розроблена для сегментації біомедицинських зображень. Вона отримала в назві літеру “U” завдяки саме свої діаграмі архітектури, що нагадує літеру “U”. Вона складається з двох основних частин: згорткової та розгорткової (або звуження та розширення). Перша являє собою серію операцій згортки та максимізаційної агрегації (макспулінгу), під час чого загальна просторова інформація зображення зменшується, а інформація про самі ознаки навпаки збільшується. Під час

розширення здійснюються операції транспонованих згорток, що збільшують розмірність результатів, та конкатенація з ознаками, які відповідають роздільній здатності зі шляху згортування. На останніх етапах мережі використовуються ще згортки розмірністю 1 на 1. Дана архітектура вміє працювати з малою кількістю зображень для навчання, але давати доволі точні результати самої сегментації.

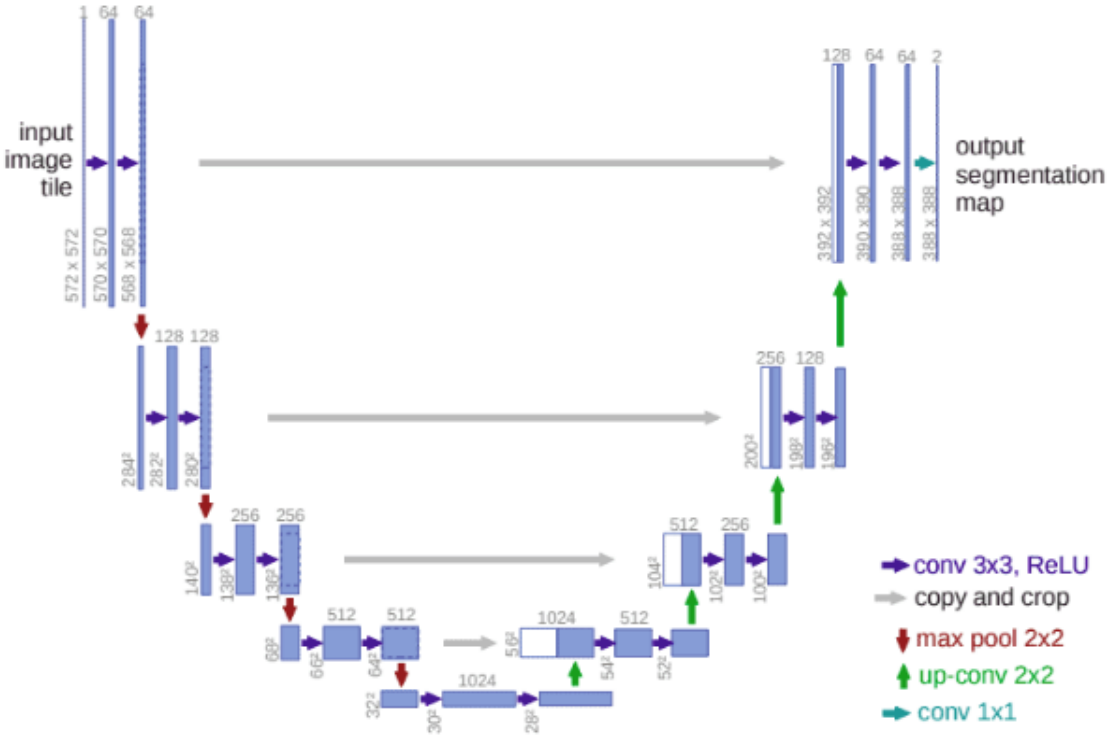

Рисунок 2.3. Діаграма архітектури UNet

2.4 SegNet

Архітектура, що має схожу будов до UNet - також складається з шляху звуження та розширення, але для розширення використовуються не карти ознак з попередніх кроків, а лише індекси максимізаційної агрегації (макспулінгу).

2.5 Висновки

Для обробки зображень невизначено різних розмірів, наприклад зображень вулиць, пейзажів тощо, краще за все використовувати архітектуру PSPNet, з доволі великим розміром вхідного зображення для моделі.

Для роботи, де важливі дрібні деталі, краще за все використовувати архітектуру UNet, оскільки вона завдяки конкатенації на шляху розширення зберігає велику кількість деталей високої роздільної здатності. Саме тому початково ця архітектура використовувалась в медичній сфері, де важлива точність. Проте, в ході практичної частини роботи буде визначено, що і для нестандартної для цієї архітектури сфери - зображень з невеликою деталізацією та невеликою кількістю дрібних об’єктів - можна досягти гарних результатів.

Для доволі простих вхідних даних з невеликою кількістю об’єктів краще за все використовувати моделі зразку FCN та SegNet, оскільки використання UNet та PSPNet буде надмірним по ресурсам. Але варто враховувати, що SegNet є повільнішим за FCN через те, що в SegNet є частина розширення, якої немає в FCN, але натомість SegNet потребує менше пам’яті ніж FCN.

РОЗДІЛ 3. ПОСТАНОВКА ПРАКТИЧНОЇ ЗАДАЧІ ТА ОПИС ВИКОРИСТАНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ

3.1 Опис задачі

В якості прикладу додатку для сегментації зображень натренуємо модель на основі вище розглянутої архітектури UNet. Для тренування скористаємось відносно нетиповим датасетом для даної архітектури - датасетом зображень з відеорегістраторів автомобілів майже без дрібних елементів, на яких так добре вміє працювати дана архітектура. Також надамо зручний спосіб взаємодії з моделлю - через веб-інтерфейс, внаслідок чого можна буде спостерігати відсегментовані зображення прямо в веб-браузері користувача додатку.

3.2 Опис використаних засобів для виконання завдання

Для виконання задачі було обрано мову програмування Python, яка є скриптовою мовою програмування високого рівня з динамічною типізацією та великою кількістю бібліотек для виконання завдань з машинного навчання та комп’ютерного зору зокрема. Крім цього можна зазначити ще декілька суттєвих переваги даної мови програмування:

- простий, чіткий, мінімалістичний синтаксис мови, який водночас дає можливість вирішувати дуже нетривіальні задачі,
- мультипарадигмальність (код можна писати і в функціональному стилі, і процедурному, і використовувати об’єктно-орієнтоване програмування)
- величезна стандартна бібліотека, що має купу бібліотек для найрізноманітніших завдань (робота з JSON, модулі для генерації рандомізованих даних, модулі для роботи з часом тощо), які навіть не треба завантажувати,
- динамічна типізація та фактори наведені вище разом дають ще одну перевагу - мова ідеально підходить для швидкого прототипування, тобто для створення основи будь-якого шаблонного додатку,

- можливість працювати з математичними завданнями (наприклад, в мові навіть є можливість працювати з комплексними числами).

Проте, очевидно, що у мови є й недоліки - один з основних це її швидкість, проте цю проблему також можна вирішити, наприклад, переписавши частини програми, що мають бути виконані доволі швидко, на компільовану мову програмування, наприклад C++, а потім просто викликати цей код напряму з коду написанного на Python.

Наш додаток складатиметься з двох основних частин:

1. Застосунок для імплементації моделі на архітектурі UNet
2. Застосунок для взаємодії (вводу-виводу) даних по протоколу HTTP, тобто через веб-браузер (або інший клієнт)

Для вирішення першого завдання мова програмування Python має кілька необхідних сторонніх (не вбудованих у саму стандартну бібліотеку мови) бібліотек, такі як TensorFlow та Keras. Перша була розроблена в компанії Google для загальних потреб в сфері машинного навчання, а друга являє собою надбудову над нею для роботи саме з нейронними мережами, причому має зручний інтерфейс, який надає можливість користуватись певними готовими шарами для конструювання власної мережі.

Для вирішення другого завдання в екосистемі мови програмування Python існує веб-фреймворк Flask. За його допомогою можна з залученням мінімальних зусиль можна створювати невеликі web-API(Application Programming Interface), тобто інтерфейси для комунікації з застосунком через певний клієнт, наприклад через звичайний браузер. Хоч Flask прийнято називати мінімалістичним фреймворком, проте за допомогою сторонніх бібліотек його функціональність можна розширювати (наприклад, додавати модулі для роботи з базами даних(flask-sqlalchemy), модулі з адміністративною панеллю(flask-admin) тощо - майже завжди вони ще й починаються з приставки

“flask”), тому він активно використовується навіть в великих проектах, зокрема він набув популярності в мікросервісній архітектурі веб-додатків.

Оскільки Flask в нашому додатку взаємодіятиме з користувачем через браузер, то він мусить повертати HTML-розмітку, щоб формувались веб-сторінки. HTML(HyperText Markup Language) являє собою мовою для розмітки веб-сторінок, тобто певний набір тегів, який веб-браузер може інтерпретувати як певний користувацький інтерфейс.

Як вище було зазначено, бібліотеки Tensorflow, Keras та фреймворк Flask є сторонніми для мови програмування Python, тобто потребують завантаження користувачем для початку роботи з ними. В екосистемі мови Python це виконується буквально в одну консольну команду за допомогою пакетного менеджеру pip: pip install library_name, де library_name це умовне позначення назви сторонньої бібліотеки.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ДОДАТКУ ТА ОТРИМАННЯ РЕЗУЛЬТАТІВ

Розглянемо загальну структуру проекту наведену на зображенні нижче:

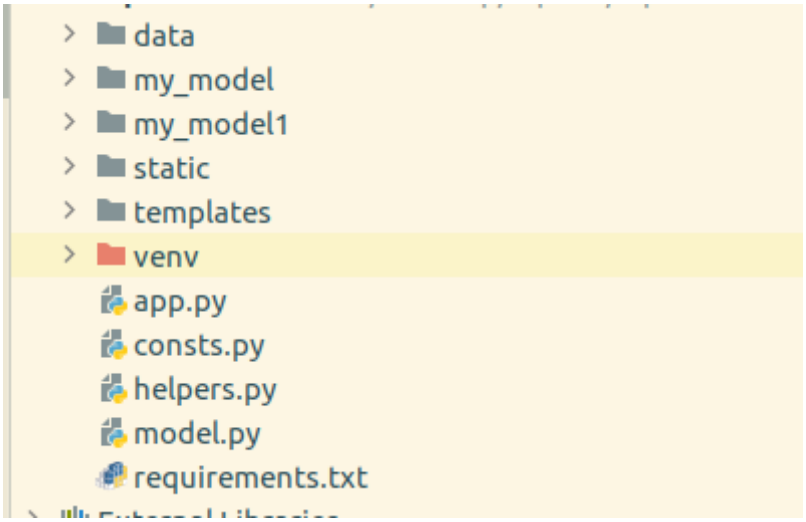

Рисунок 4.1. Структура проекту

Пройдемося по компонентам системи, серед яких є наступні:

- папка data, де власне і знаходяться усі зображення, що необхідні для тренування моделі
- папки під назвою my_model*, де * - певне число, (опціональна, з'являється лише після першого запуску проекту), де знаходиться модель (після першого запуску модель не перетреновується, щоб уникнути затрат на це, а береться вже створена, якщо тільки не було змінено якісь параметри моделі), для різних версій моделі створюються різні папки
- папка static, куди додаток зберігає всі файли, що будуть отримані від користувача
- папка templates, де знаходяться всі необхідні HTML-шаблони додатку
- папка venv, автогенерована, де зберігаються всі завантажені бібліотеки в даному віртуальному оточенні
- файл app.py, де знаходяться функції-обробники Flask, які забезпечують веб-інтерфейс додатку
- файл consts.py, де знаходяться всі константи додатку

- файл model.py, де знаходиться вся логіка зі створення мережі архітектури UNet
- файл helpers.py, де знаходяться допоміжні функції, наприклад функції для обробки датасету тощо
- файл requirements.txt, де прописані усі необхідні версії сторонніх бібліотек, які потрібні для локального запуску проекту (всі бібліотеки можна завантажити однією консольною командою pip install -r requirements.txt).

Файл requirements.txt зі списком сторонніх бібліотек:

```
Keras-Preprocessing==1.1.2
kiwisolver==1.4.2
libclang==14.0.1
Markdown==3.3.7
MarkupSafe==2.1.1
matplotlib==3.5.2
numpy==1.22.4
oauthlib==3.2.0
opt-einsum==3.3.0
packaging==21.3
pandas==1.4.2
Pillow==9.1.1
pkg_resources==0.0.0
protobuf==3.19.4
pyasn1==0.4.8
pyasn1-modules==0.2.8
pyparsing==3.0.9
python-dateutil==2.8.2
pytz==2022.1
requests==2.27.1
requests-oauthlib==1.3.1
rsa==4.8
six==1.16.0
tensorboard==2.9.0
tensorboard-data-server==0.6.1
tensorboard-plugin-wit==1.8.1
tensorflow==2.9.1
tensorflow-estimator==2.9.0
tensorflow-io-gcs-filesystem==0.26.0
termcolor==1.1.0
typing_extensions==4.2.0
urllib3==1.26.9
Werkzeug==2.1.2
wrapt==1.14.1
zipp==3.8.0
```

Рисунок 4.2. Файл requirements.txt.

Як видно з зображення вище, в списку в файлі зазначені не тільки бібліотеки, які було описано в попередньому розділі, а й допоміжні бібліотеки, що завантажуються додатково при завантаженні основних.

Шаблони з розширенням .html містять звичайну розмітку, яка потім трансформується в веб-сторінки у користувача. Приклад такого файлу з розміткою для завантаження файлів можна побачити нижче.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Завантаження зображення</title>
</head>
<body>
    <h1>Завантажити новий файл</h1>
    <form method=post enctype=multipart/form-data>
      <input type=file name=file>
      <input type=submit value=Завантажити>
    </form>
</body>
</html>
```

Рисунок 4.3. HTML-шаблон

Далі маємо файл consts.py, в якому просто зібрані всі константи проекту, щоб тримати їх в одному місці і проект було легше конфігурувати.

```
DEFAULT_IDX_FOR_ROOT_PAGE = 3
IMAGE_PATH = './data/RGBSet/'
MASK_PATH = './data/MaskSet/'
KERNEL_SIZE = 3
IMG_HEIGHT = 96
IMG_WITH = 128
NUM_CHANNELS = 3
DEFAULT_INPUT_SIZE = (IMG_HEIGHT, IMG_WITH, NUM_CHANNELS)
EPOCHS = 40
BUFFER_SIZE = 500
BATCH_SIZE = 32

```

Рисунок 4.4. Код файлу consts.py

Файл app.py містить декілька важливих складових додатку. Спочатку йде ініціалізація веб-серверу:

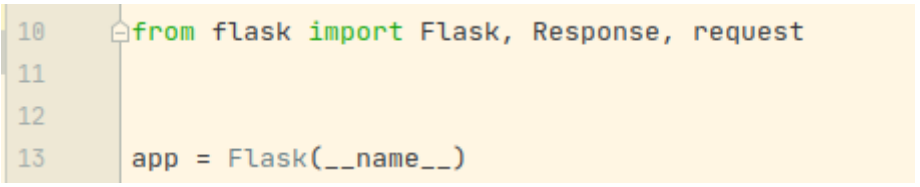

Рисунок 4.5. Створення Flask-додатку

Далі оголосимо першу функцію-обробник. В термінології веб-додатків це така функція, яка приймає запит від користувача, обробляє його певним чином та повертає клієнтові (в нашому випадку в веб-браузер) інформацію, тобто відповідь (web-response). Приклад, такої функції наведено на зображенні нижче, вона приймає параметр номеру картинки з датасету (або бере дефолтне константне значення) та повертає користувачеві зображення та сегментоване зображення з тренувального датасету, щоб користувач мав уявлення, як працює система. Щоб звернутись до цієї функції варто всього на всього перейти за адресою localhost:500/index в своєму браузері, коли запущено додаток локально (localhost - доменна адреса для всіх локально запущених веб-проектів, 5000 - стандартний порт для застосунків на фреймворці Flask, index - назва маршруту, яка присвоєна нашій функції). Також можна передати так званий query parameter, параметр запиту, в адресній стрічці, щоб вточнити, яке саме за номером зображння з датасету має бути відображено (приклад такої адреси - localhost:5000/index?n=12).

```python
@app.route('/index')
def index():
    idx = request.args.get('n') or DEFAULT_IDX_FOR_ROOT_PAGE
    img = imageio.imread(image_list[idx])
    mask = imageio.imread(mask_list[idx])

    fig, arr = plt.subplots(1, 2, figsize=(14, 10))
    arr[0].imshow(img)
    arr[0].set_title('Image')
    arr[1].imshow(mask[:, :, 0])
    arr[1].set_title('Segmentation')
    output = io.BytesIO()
    FigureCanvas(fig).print_png(output)
    return Response(output.getvalue(), mimetype='image/png')
```

Рисунок 4.6. Перша функція-обробник з файл app.py

Приклад роботи даної функції:

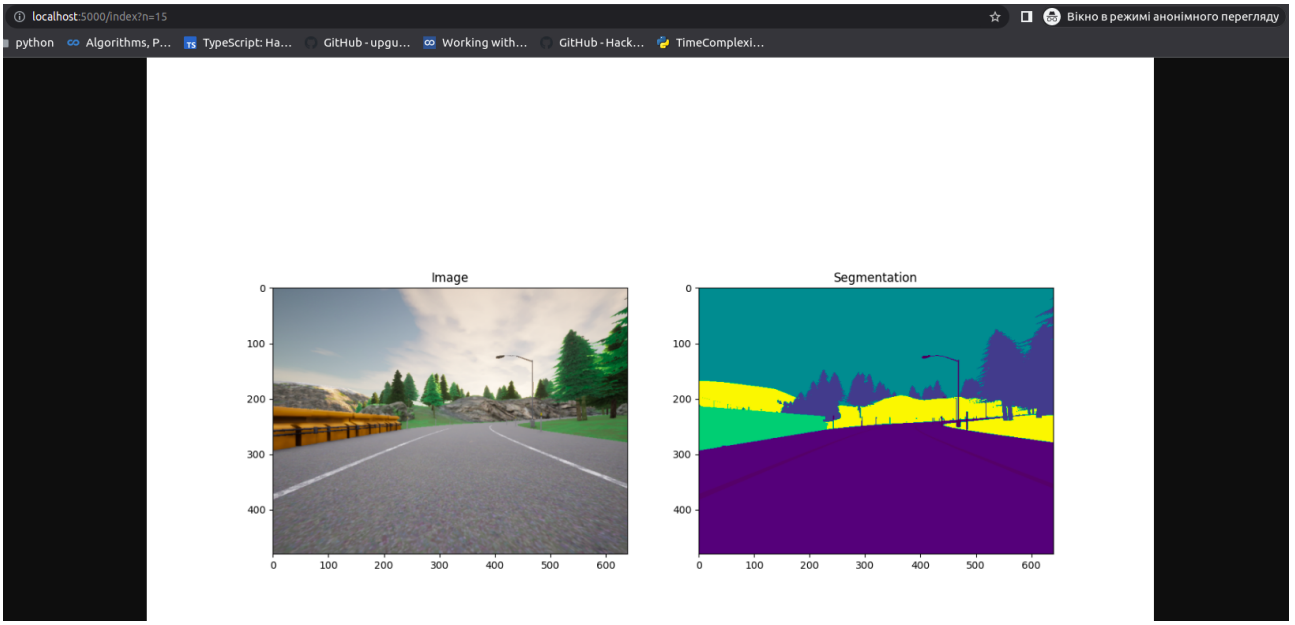

Рисунок 4.7. Веб-сторінка для перегляду зображень з датасету за їх номером

Наступна функція-обробник являє собою основну вхідну точку в наш додаток. Тобто вона приймає на вхід від користувача зображення, яке далі поступає в нашу модель і видає пару зображень: початкове зображення та сегментований результат. Для звернення необхідно перейти за адресою

localhost:5000/perform_segmentation, після чого завантажити необхідне зображення за допомогою форми, що буде на сторінці.

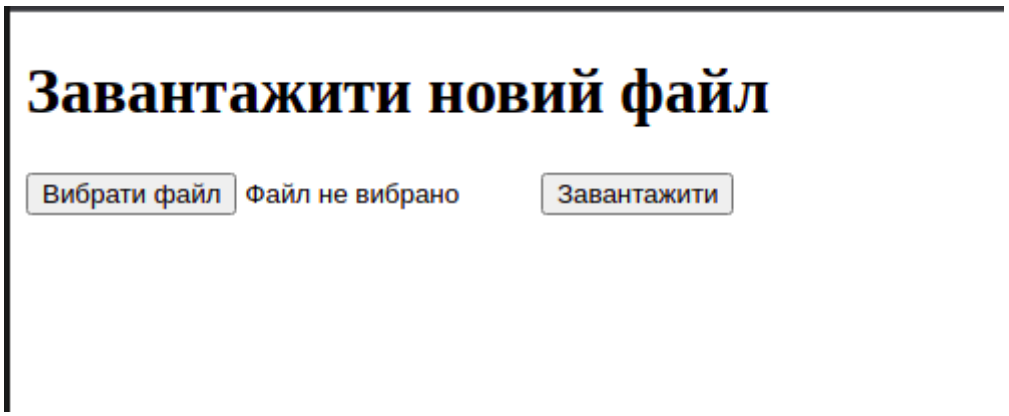

Рисунок 4.8. Форма для завантаження зображення.

Після натискання кнопки “завантажити” користувач отримує результат на сторінці у вигляді пари з початкового та вихідного зображень.

Рисунок 4.9. Результат роботи функції-обробнику.

```python
@app.route('/perform_segmentation', methods=["GET", "POST"])
def perform_segmentation():
    if request.method == 'POST':
        file = request.files['file']
        file_path = f'static/images/{file.filename}'
        file.save(os.path.join(app.root_path, file_path))

        img = tf.io.read_file(os.path.join('', file_path))
        img = tf.image.decode_png(img, channels=NUM_CHANNELS)
        img = tf.image.convert_image_dtype(img, tf.float32)
        img = tf.image.resize(img, (IMG_HEIGHT, IMG_WIDTH), method='nearest')
        unet = tf.keras.models.load_model("my_model40")
        pred_mask1 = unet.predict(img[tf.newaxis, ...])
        display_list = [img, create_mask(pred_mask1)]

        figure = plt.figure(figsize=(15, 15))
        title = ['Input Image', 'Predicted Mask']

        for idx, elem in enumerate(display_list):
            plt.subplot(1, len(display_list), idx+1)
            plt.title(title[idx])
            plt.imshow(tf.keras.preprocessing.image.array_to_img(elem))
            plt.axis('off')
        output = io.BytesIO()
        FigureCanvas(figure).print_png(output)
        return Response(output.getvalue(), mimetype='image/png')
    return render_template('image.html')

```

Рисунок 4.10. Друга функція-обробник з файлу app.py

Нарешті, маємо файл model.py, де описана вся модель, що використовується для сегментації поступаючих через веб-апі зображень. Оскільки, мережа була побудована за допомогою архітектури UNet, то чітко можна побачити наступні два блоки, що реалізовані як дві функції: downsampling та upsampling, тобто шлях звуження та шлях розширення. Реалізація цих етапів наведена на зображеннях нижче.

```python
def downsampling_block(
    inputs: Input = None,
    n_filters: int = 32,
    dropout_prob: t.Union[float, int] = 0,
    max_pooling=True
) -> tuple:
    conv = Conv2D(
        n_filters,
        KERNEL_SIZE,
        activation='relu',
        padding='same',
        kernel_initializer='he_normal'
    )(inputs)
    conv = Conv2D(
        n_filters,
        KERNEL_SIZE,
        activation='relu',
        padding='same',
        kernel_initializer='he_normal'
    )(conv)

    if dropout_prob > 0:
        conv = Dropout(dropout_prob)(conv)
    next_layer = MaxPooling2D(2, strides=2)(conv) if max_pooling else conv

    return next_layer, conv
```

Рисунок 4.11. Функція для реалізація шляху звуження мережі

```
def upsampling_block(expansive_input, contractive_input, n_filters=32):
    up = Conv2DTranspose(
        n_filters,
        KERNEL_SIZE,
        strides=2,
        padding='same'
    )(expansive_input)
    merge = concatenate([up, contractive_input], axis=3)
    conv = Conv2D(
        n_filters,
        KERNEL_SIZE,
        activation='relu',
        padding='same',
        kernel_initializer='he_normal'
    )(merge)
    conv = Conv2D(
        n_filters,
        KERNEL_SIZE,
        activation='relu',
        padding='same',
        kernel_initializer='he_normal'
    )(conv)
    return conv
```

Рисунок 4.12. Функція для реалізація шляху розширення мережі

Якщо поєднати всі етапи побудови моделі включно з імплементацією реалізації мережі, обробкою датасету тощо та тренуванням моделі, то отримаємо результат у вигляді коду, що зображено нижче.

```
106
107
108 def get_unet_model(
109     input_size: tuple = (96, 128, 3),
110     num_filters: int = 32,
111     n_classes: int = 23
112 ) -> tf.keras.Model:
113     inputs = Input(input_size)
114
115     dblock1 = downsampling_block(inputs=inputs, n_filters=num_filters * 1)
116     dblock2 = downsampling_block(inputs=dblock1[0], n_filters=num_filters * 2)
117     dblock3 = downsampling_block(inputs=dblock2[0], n_filters=num_filters * 4)
118     dblock4 = downsampling_block(inputs=dblock3[0], n_filters=num_filters * 8, dropout_prob=0.3)
119     dblock5 = downsampling_block(inputs=dblock4[0], n_filters=num_filters * 16, dropout_prob=0.3, max_pooling=False)
120
121     ublock6 = upsampling_block(dblock5[0], dblock4[1], num_filters * 8)
122     ublock7 = upsampling_block(ublock6, dblock3[1], num_filters * 4)
123     ublock8 = upsampling_block(ublock7, dblock2[1], num_filters * 2)
124     ublock9 = upsampling_block(ublock8, dblock1[1], num_filters * 1)
125
126     conv9 = Conv2D(
127         num_filters,
128         KERNEL_SIZE,
129         activation='relu',
130         padding='same',
131         kernel_initializer='he_normal'
132     )(ublock9)
133
134     conv10 = Conv2D(n_classes, 1, padding='same')(conv9)
135     model = tf.keras.Model(inputs=inputs, outputs=conv10)
136     return model
137
```

Рисунок 4.13. Побудова моделі в model.py

```
164
165 unet_model = get_unet_model(DEFAULT_INPUT_SIZE)
166
167 unet_model.compile(
168     optimizer='adam',
169     loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
170     metrics=['accuracy']
171 )
172
173 dataset.batch(BATCH_SIZE)
174 train_dataset = dataset.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
175 model_history = unet_model.fit(train_dataset, epochs=EPOCHS)
176 unet_model.save('my_model40')
177
```

Рисунок 4.13. Тренування моделі в model.py

Хід тренування протягом заданої кількості епох (якщо не задано, то дефолтне значення 40) друкується в стандартний потік виводу, тобто консоль.

```
Epoch 31/40
34/34 [==============================] - 204s 6s/step - loss: 0.1561 - accuracy: 0.9501
Epoch 32/40
34/34 [==============================] - 261s 8s/step - loss: 0.1457 - accuracy: 0.9535
Epoch 33/40
34/34 [==============================] - 267s 8s/step - loss: 0.1385 - accuracy: 0.9558
Epoch 34/40
34/34 [==============================] - 219s 6s/step - loss: 0.1340 - accuracy: 0.9568
Epoch 35/40
34/34 [==============================] - 252s 7s/step - loss: 0.1357 - accuracy: 0.9562
Epoch 36/40
34/34 [==============================] - 281s 8s/step - loss: 0.1290 - accuracy: 0.9582
Epoch 37/40
34/34 [==============================] - 243s 7s/step - loss: 0.1285 - accuracy: 0.9581
Epoch 38/40
34/34 [==============================] - 263s 8s/step - loss: 0.1201 - accuracy: 0.9607
Epoch 39/40
34/34 [==============================] - 632s 18s/step - loss: 0.1154 - accuracy: 0.9621
Epoch 40/40
34/34 [==============================] - 177s 5s/step - loss: 0.1145 - accuracy: 0.9625
```

Рисунок 4.14. Вивід в консоль під час тренування моделі.

Також інтерфейс бібліотеки keras дозволяє переглянути за допомогою методу model.summary() структуру моделі з уісма шарами мережі, які ми будували в коді вище.

```
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 96, 128, 3)  0           []
                                ]

 conv2d (Conv2D)                (None, 96, 128, 32)  896         ['input_1[0][0]']

 conv2d_1 (Conv2D)              (None, 96, 128, 32)  9248        ['conv2d[0][0]']

 max_pooling2d (MaxPooling2D)   (None, 48, 64, 32)   0           ['conv2d_1[0][0]']

 conv2d_2 (Conv2D)              (None, 48, 64, 64)   18496       ['max_pooling2d[0][0]']

 conv2d_3 (Conv2D)              (None, 48, 64, 64)   36928       ['conv2d_2[0][0]']

 max_pooling2d_1 (MaxPooling2D)  (None, 24, 32, 64)  0           ['conv2d_3[0][0]']

 conv2d_4 (Conv2D)              (None, 24, 32, 128)  73856       ['max_pooling2d_1[0][0]']

 conv2d_5 (Conv2D)              (None, 24, 32, 128)  147584      ['conv2d_4[0][0]']

 max_pooling2d_2 (MaxPooling2D)  (None, 12, 16, 128)  0          ['conv2d_5[0][0]']

 conv2d_6 (Conv2D)              (None, 12, 16, 256)  295168      ['max_pooling2d_2[0][0]']

 conv2d_7 (Conv2D)              (None, 12, 16, 256)  590080      ['conv2d_6[0][0]']

 dropout (Dropout)              (None, 12, 16, 256)  0           ['conv2d_7[0][0]']

 max_pooling2d_3 (MaxPooling2D)  (None, 6, 8, 256)   0           ['dropout[0][0]']

 conv2d_8 (Conv2D)              (None, 6, 8, 512)    1180160     ['max_pooling2d_3[0][0]']

 conv2d_9 (Conv2D)              (None, 6, 8, 512)    2359808     ['conv2d_8[0][0]']

 dropout_1 (Dropout)            (None, 6, 8, 512)    0           ['conv2d_9[0][0]']
```

Рисунок 4.15. Вивід методу summary(), шлях звуження мережі

```
conv2d_11 (Conv2D)              (None, 12, 16, 256)  590080      ['conv2d_10[0][0]']

conv2d_transpose_1 (Conv2DTran  (None, 24, 32, 128)  295040     ['conv2d_11[0][0]']
spose)

concatenate_1 (Concatenate)     (None, 24, 32, 256)  0           ['conv2d_transpose_1[0][0]',
                                                                 'conv2d_5[0][0]']

conv2d_12 (Conv2D)              (None, 24, 32, 128)  295040      ['concatenate_1[0][0]']

conv2d_13 (Conv2D)              (None, 24, 32, 128)  147584      ['conv2d_12[0][0]']

conv2d_transpose_2 (Conv2DTran  (None, 48, 64, 64)   73792       ['conv2d_13[0][0]']
spose)

concatenate_2 (Concatenate)     (None, 48, 64, 128)  0           ['conv2d_transpose_2[0][0]',
                                                                 'conv2d_3[0][0]']

conv2d_14 (Conv2D)              (None, 48, 64, 64)   73792       ['concatenate_2[0][0]']

conv2d_15 (Conv2D)              (None, 48, 64, 64)   36928       ['conv2d_14[0][0]']

conv2d_transpose_3 (Conv2DTran  (None, 96, 128, 32)  18464       ['conv2d_15[0][0]']
spose)

concatenate_3 (Concatenate)     (None, 96, 128, 64)  0           ['conv2d_transpose_3[0][0]',
                                                                 'conv2d_1[0][0]']

conv2d_16 (Conv2D)              (None, 96, 128, 32)  18464       ['concatenate_3[0][0]']

conv2d_17 (Conv2D)              (None, 96, 128, 32)  9248        ['conv2d_16[0][0]']

conv2d_18 (Conv2D)              (None, 96, 128, 32)  9248        ['conv2d_17[0][0]']

conv2d_19 (Conv2D)              (None, 96, 128, 23)  759         ['conv2d_18[0][0]']

==================================================================================================
Total params: 8,640,471
Trainable params: 8,640,471
Non-trainable params: 0
```

Рисунок 4.16. Вивід методу summary(), шлях розширення мережі

Також маємо файл helpers.py - в ньому знаходяться різні допоміжні функції, які виконують здебільшого операції над вхідним датасетом та обробляють його.

```python

def create_list(path: str) -> list:
    return [path + i for i in os.listdir(path)]


def create_dataset(first_list: list, second_list: list) -> tf.data.Dataset:
    return tf.data.Dataset.from_tensor_slices(
        (tf.constant(first_list), tf.constant(second_list))
    )


def _extract_and_decode_image(path: str) -> Tensor:
    img = tf.io.read_file(path)
    return tf.image.decode_png(img, channels=3)


def process_path(image_path: str, mask_path: str) -> tuple:
    img, mask = _extract_and_decode_image(image_path), _extract_and_decode_image(mask_path)
    return tf.image.convert_image_dtype(img, tf.float32), tf.math.reduce_max(mask, axis=-1, keepdims=True)


def _resize_img(img):
    return tf.image.resize(img, (96, 128), method='nearest')


def preprocess(image: Tensor, mask: Tensor) -> tuple:
    return _resize_img(image), _resize_img(mask)

```

Рисунок 4.17. Файл helpers.py

ВИСНОВКИ

Підсумовуючи всю проведену вище роботу, можна зазначити, що на даний момент сфера машинного навчання загалом та комп’ютерного зору зокрема є надзвичайно популярною та широко вживаною. Завдання, що залучають використання знань з комп'ютерного зору, включають в себе розпізнавання, класифікація, трекінг та сегментація об’єктів на зображеннях та відеоматеріалах. Ці всі завдання допомагають людству в багатьох сферах, як наприклад, при виявленні пухлин на зображеннях за допомогою сегментації зображень нутрощів людини.

В ході роботи було проведено дослідження з огляду різних архітектур побудови нейронних мереж для створення моделі для сегментації зображень, серед них було оглянуто: FCN, PSPNet, UNet та SegNet. В практичній частині було проведено експеримент з побудови власноруч мережі на основі архітектури UNet, модель тренувалась та працювала на нетиповому для себе типі вхідних зображень - outdoor-зображень без великою кількості дрібних деталей, але показала себе доволі непогано після тренування на 40 епох маючи точність порядку 0.95. Також було створено веб-інтерфейс для взаємодії з даною моделлю, що додає гнучкості системі, тобто її можна буде масштабувати в майбутньому додаючи нові ендпоінти з новими можливостями (завантажувати багато зображень для сегментації за раз, особистий кабінет користувача тощо).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 3.2 billion images and 720,000 hours of video are shared online daily. Can you sort real from fake? [Електронне джерело] - Режим доступу: https://theconversation.com/3-2-billion-images-and-720-000-hours-of-video-are-shared-online-daily-can-you-sort-real-from-fake-148630
2. A Brief History of Computer Vision (and Convolutional Neural Networks). [Електронне джерело] - Режим доступу: https://hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3
3. Optical character recognition [Електронне джерело] - Режим доступу: https://en.wikipedia.org/wiki/Optical_character_recognition
4. Neural Networks [Електронне джерело] - Режим доступу: https://www.ibm.com/cloud/learn/neural-networks
5. Barghout, Lauren, and Lawrence W. Lee. "Perceptual information processing system." Paravue Inc. U.S. Patent Application 10/618,543, filed July 11, 2003.
6. Zachow, Stefan, Michael Zilske, and Hans-Christian Hege. "3D reconstruction of individual anatomy from medical image data: Segmentation and geometry processing." (2007).
7. Forghani, M.; Forouzanfar, M.; Teshnehlab, M. (2010). "Parameter optimization of improved fuzzy c-means clustering algorithm for brain MR image segmentation". Engineering Applications of Artificial Intelligence
8. J. A. Delmerico, P. David and J. J. Corso (2011): "Building façade detection, segmentation and parameter estimation for mobile robot localization and guidance", International Conference on Intelligent Robots and Systems, pp. 1632–1639.
9. Olaf Ronneberger, Philipp Fischer, and Thomas Brox (2015): “U-Net: Convolutional Networks for Biomedical Image Segmentation”

10. Long, J.; Shelhamer, E. & Darrell, T. (2014), Fully convolutional networks for semantic segmentation
11. Hertz, J.; Palmer, Richard G.; Krogh, Anders S. (1991). Introduction to the theory of neural computation