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

Сегментація зображень з використанням нейронних мереж

Текстова частина до курсової роботи
за спеціальністю „Програмна інженерія” 6.050103

Керівник курсової роботи
с.в. _Бучко О.А. (прізвище та ініціали)
_____
(підпис)

“____” _____ 2022 р.
Виконав студент Савкін Г.І.
(прізвище та ініціали)
“____” _____ 2022 р.

м. Київ – 2021 рік

Календарний план виконання курсової роботи

Тема: Розробка нейронної мережі для розпізнавання графічних об’єктів

Календарний план виконання роботи:

№ п\п	Назва етапу дипломного проекту(роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	Жовтень - листопад 2021р.	
2.	Огляд літератури за темою роботи	Листопад-лютий 2021р.	
3.	Аналіз методів сегментації зображення	Лютий-березень 2022р.	
4.	Аналіз мережі U-Net	Березень 2022р.	
5.	Написання текстової частини	Квітень 2022р.	
6.	Перегляд змісту роботи керівником		
7.	Створення презентації		
8.	Захист курсової роботи		

Студент Савкін Г.І. _____

Керівник Бучко О.А. _____

“_____”_____ 2022 р.

Зміст

ВСТУП

Проблема сегментації зображень є однією з найрелевантніших на сьогоднішній день. Велика кількість індустрій, від автономних транспортних засобів до медичної індустрії потребують не лише правильної класифікації цільного зображення чи приблизного визначення положення певного об’єкту на ньому, а по-піксельно правильного визначення границь цих об’єктів.

Досягнути потрібних результатів для даної проблеми дозволили сучасні мережі машинного навчання, особливо глибокі згорткові мережі, що на початку 2010-х років повністю змінили підхід до проблеми як сегментації зображень, так і до усіх інших проблем комп’ютерного зору. Індустрія комп’ютерного зору та зокрема сегментації зображень все ще дуже активна і змінюється неймовірно швидко. Лише 16-го травня 2021 року була опублікована нова state-of-the-art мережа, яка вже встановила кілька нових рекордів.

Анотація

Роботу присвячено дослідженню проблеми сегментації зображень, зокрема у контексті машинного навчання та глибоких нейронних мереж, аналізу складових глибоких згорткових мереж, backbone мереж та імплементацій для сегментації зображень. На основі досліджень було розроблено дві згорткові мережі, з backbone мережею та без, порівняно їх точність та швидкість.

РОЗДІЛ 1: Проблема сегментації зображень

1.1 Аналіз проблеми сегментації зображень

Сегментація зображень – це задача зі сфери комп’ютерного зору, ціль якої перетворити вхідне зображення на певну кількість масок, що позначають границі об’єктів. Є два основних типи сегментації: семантична сегментація, об’єктна сегментація та паноптична сегментація (semantic segmentation, instance segmentation, panoptic segmentation).

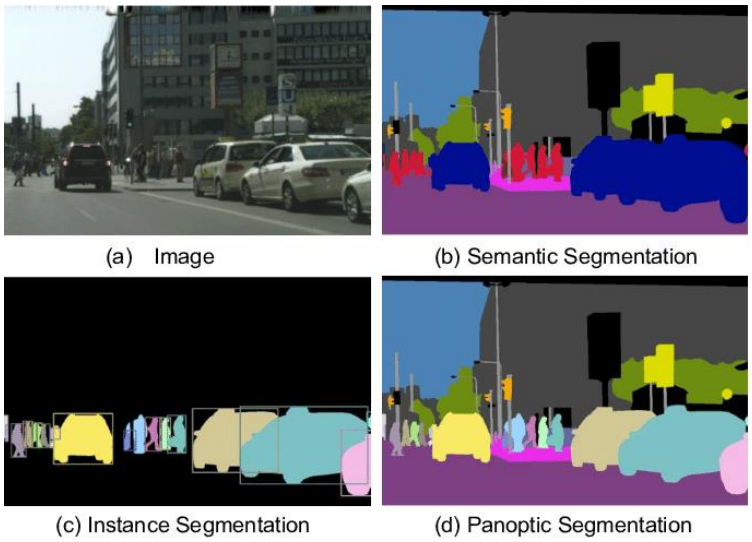

Рисунок 1, приклад семнантичної сегментації, сегментації об'єктів та паноптичної сегментації

Семантична сегментація розділяє зображення на класи, але не розрізняє між різними об’єктами що належать одному класу. Об’єктна сегментація виявляє об’єкти на зображенні та класифікує їх. Паноптична сегментація поєднує обидві попередні, створюючи повністю сегментоване зображення на якому окремо виділено кожний об’єкт.

Сегментація зображень використовується у багатьох сучасних індустріях, зокрема у:

- Автономному транспорті

- Медицині
- Аналізі супутникових даних
- Контроль якості на заводах

На відміну від багатьох інших проблем комп’ютерного зору як, наприклад, класифікація зображень чи виявлення об’єктів (object detection), сегментація зображень потребує класифікації кожного пікселя зображення. Через це й анотації на тренувальних даних це не просто один клас чи приблизні координати прямокутника навколо об’єкту, а чітко визначена група регіонів.

1.2 Рішення проблеми сегментації без глибокого машинного навчання

Машинне навчання це дуже потужна технологія, здатна вирішувати великий відсоток задач не тільки комп’ютерного зору, я і комп’ютерної науки загалом. Але не кожна задача варта створення нейронної мережі чи будь якого іншого рішення машинного навчання. Дуже часто задача може бути вирішена простим алгоритмом, який простіше написати, простіше підтримувати довгий час та який буде працювати на кілька порядків швидше працювати. Через це має сенс хоча б швидко розглянути варіанти сегментації зображень без інструментів машинного навчання.

1.2.1 Сегментація на основі регіонів

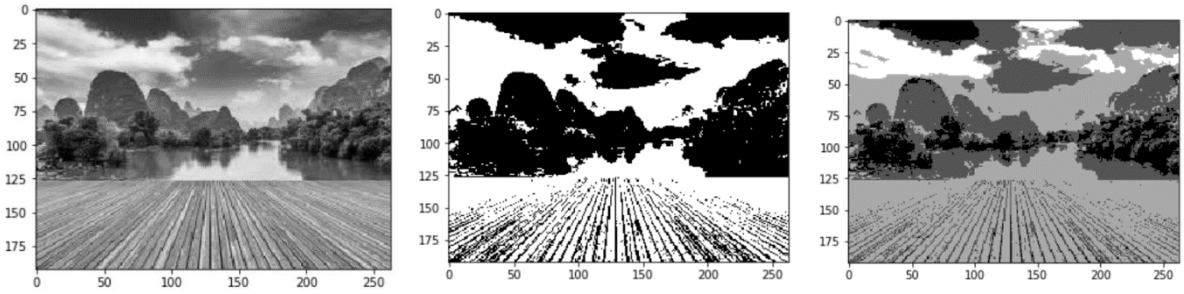

Рисунок 2, graysclale зображення, ділення одним порогм, ділення кількома порогами

Ідея цього алгоритму полягає в конвертації кольорового зображення на grayscale, після чого задається поріг, усі пікселі що мають значення менше за заданий поріг класифікуються як перший клас, усі що мають вище значення – як другий. Таким чином дуже просто, наприклад, поділити зображення на фоновий план та передній план, або якщо шукані об’єкти добре контрастують в порівнянні з іншим наповненням зображення, або навпаки. Поріг може бути встановлений вручну, або визначений по гістограмі зображення автоматично. Якщо середина гістограми має чітко виражений локальний мінімум, це майже гарантовано найкращий поріг для поділу.

Альтернативно можна використати більше одного порогу, якщо є необхідність поділити зображення на більше ніж два класи

1.2.2 Сегментація зображень за допомогою кластеризації k-means

Алгоритм кластеризації - це ще один метод швидкої сегментації. Він відноситься до машинного навчання, проте в багато разів простіший за глибоке навчання. До того ж цей алгоритм не потребує класифікації вхідних даних, що робить генерацію великого датасету набагато простіше.

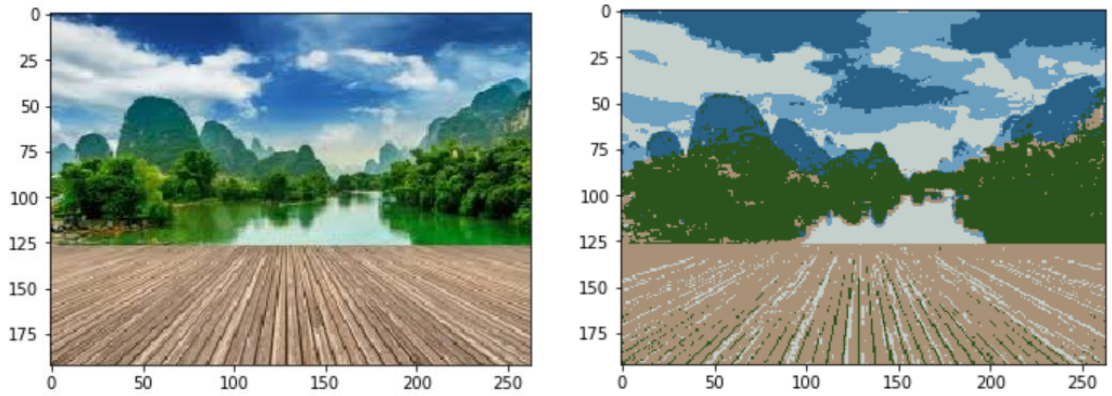

Рисунок 3, вхідне зображення (зліва) та результуючі кластери (справа)

Алгоритм полягає у випадковому обранні певного числа ядер, після чого проводится оцінка для кожного з пікселів, яке з ядер на нього схоже найбільше. Пікселі групуються відповідно до своїх найкращих кандидатів. Після цього обираються нові ядра, що лежать в центрі обраних на попередньому кроці ядер. Проце повторується доки не буде циклу, на якому жодний з пікселів не змінив класу.

Цей алгоритм трохи складніший за попередній як в реалізації так і в часі, який потрібен для його виконання. Проте великим плюсом порівняно з

РОЗДІЛ 2: Згорткові нейронні мережі (Convolutional neural networks), інструменти та методики їх розробки

2.1 Структура згорткових мереж

Традиційні нейронні мережі – так звані нейронні мережі прямого поширення (feed forward) – дуже неєффективно працюють з зображеннями. Шари такої мережі тісно пов’язані – тобто кожний нейрон слою k_i пов’язаний з кожним нейроном слою k_{i+1} – і оскільки зображення має велику кількість нейронів на вхідному шару, кількість тренованих параметрів дуже швидко зростає. Навіть для дуже малих за розміром зображень датасету Fashion MNIST[1], що мають розмір 28x28x1, feed forward мережа з усього лише двома прихованими шарами може мати близько півмільйона тренованих параметрів.

До того ж зазвичай задачі комп’ютерного зору вимагають від мережі розуміння просторового та часового контексту, накшталт того, як це робить людський мозок.

Для вирішення проблем цього типу зазвичай прийнято використовувати згорткові нейронні мережі (Convolutional neural networks або CNN). Від feed forward мереж вони відрізняються наявністю згорткових, або конволюційних шарів, інколи одного але зазвичай багатьох. У більшості архітектур конволюційні шари супроводжуються агрегувальним шаром. Кінцевим шаром мережі зазвичай є один повноз’єднаний шар. Крім того майже завжди CNN використовують виключення, нормалізацію та штучні дані.

Варто зазначити, що на початку 2020-х років з’явилась дуже перспективна альтернатива згортковим мережам у вигляді трансформерів, проте на даний момент CNN все ще мають кращі показники.

2.2 Конволюція

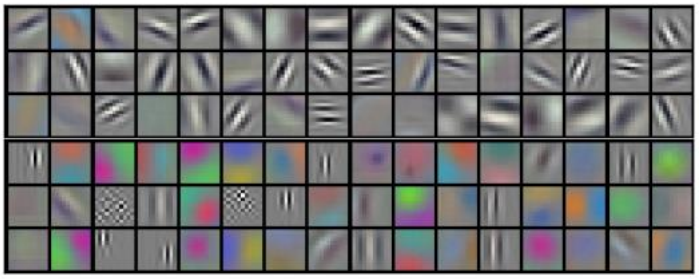

Рисунок 4, приклади фільтрів після першого шару Conv2d в AlexNet

Шар конволюції застосовує певного розміру фільтр до вхідного шару. На кожному кроці обирається наступне «вікно» зображення, виконується операція множення матриць між ним та елементами фільтру, в результаті повертається значення для одного пікселю вихідного шару. Ціль конволюції – перетворити вхідне зображення на масив мапи ознак (feature map), надати тренувабельні фільтри, які після тренування можуть витягати контекстуальні ознаки зображень. Якщо вхідна матриця це оригінальне зображення, то такими фільтрами можуть бути краї зображення, плями кольору, тощо. Якщо ж вхідна матриця – вихідний результат одної з попередніх конволюційних груп (шар конволюції разом з агрегувальним шаром та шаром-активатором), то фільтри будуть представляти більше складні елементи, наприклад певні геометричні фігури. Цей шар мало коли змінює перші два виміри вхідного шару на виході, проте часто збільшує розмір шару вглиб, створюючи багато нових фільтрів. Конволюція має різний ефект в залежності від кількох параметрів:

- Розмір ядра – впливає на кількість пікселів які задаватимуть вихідне значення. Чим цей розмір більший, тим більшими будуть проаналізовані елементи зображення. Стандартний розмір – 3x3
- Крок (stride) – кількість пікселів на яке зсувається вікно на кожному кроці. Зазвичай має значення «1» аби отримати якомога більше ознак

(features) з зображення, проте в деяких ситуаціях може бути підвищене аби пришвидшити виконання конволюції.

- Padding – зазвичай padding – це додаткові пікселі, які додає конвуляційний шар, щоб зберегти розміри вхідного зображення. Проте часто для конволюційного шару визначають цей параметр як “same” – додатковий шар нулів навколо вхідної матриці як padding, чи “valid” – padding відсутній.
- Кількість вихідних фільтрів

2.3 Агрегувальний шар

Задача цього шару – знизити розмір вихідних фільтрів з попереднього конволюційного шару. Стандартною імплементацією цього шару є операція пулінгу, конкретніше MaxPooling. Алгоритм розбиває вхідну матриці на сегменти 2x2, і вираховує максимальне значення в кожному з них. Ці значення і формують результуючу матрицю, що зберігає свій розмір в глибину, але має довжину та ширину вдвічі меншу за вхідні. Цей шар дозволяє сильно зменшити кількість тренованих параметрів, зберігаючи при цьому інформацію щодо наявності певних ознак у визначеній зоні.

2.4 Повнозв’язаний шар

Останні кілька шарів CNN нагадують приховані шари feed forward мережі. Кожний нейрон пов’язаний з кожним іншим нейроном наступного та попереднього шарів. На цьому етапі мережа тренується на видобутих фільтрах та певним чином класифікує зображення. Цей шар присутній не завжди, зокрема в мережах для сегментації зображень використовуються інші методи для отримання результату.

2.5 Виключення

Класичний інструмент машинного навчання – виключення, або dropout – з певною імовірністю ігнорує кілька нейронів в шарі для цієї епохи тренування. Використовується dropout для запобігання ефекту перенавчання (overfitting). Проте у сучасних CNN виключення використовуються нечасто[2], оскільки цей підхід має кілька значних недоліків. По-перше, виключення збільшують кількість тренувальних епох для досягнення того ж результату, хоча й час потрібний на тренування однієї епохи менший. По-друге, ефект регуляризації який надають виключення набагато ефективніше досягається шляхом нормалізації[3]

2.6 Нормалізація (Batch Normalization)

Нормалізація вхідних даних використовувалась ще під час найперших імплементацій нейронних мереж. Масштаб даних може сильно відрізнятись залежно від їх домену, під час нормалізації масштаб їх усіх регаляризується, аби бути однаковим в усіх випадках. Для batch нормалізації цей процес дуже схожий, але виконується над вихідними даними конволюційних шарів. Цей процес підвищує регуляризацію даних, зменшує ефект перенавчання.

На сьогоднішній день batch нормалізація майже повністю витиснула інші підходи до регуляризації, зокрема їй віддають перевагу в порівнянні з виключеннями.

2.7 Штучні дані

Будь-яка глибока мережа вимагає великої кількості тренувальних даних, зазвичай цей об’єм вимірюється в сотнях тисячах чи мільйонах. Чим більше вхідних даних отримує глибока мережа, тим точніше її прогнозування. Проте часто отримати таку кількість даних, до того ж правильно анотованих, дуже складно. Особливо сильно ця проблема проявляється у задачах сегментації зображень, адже

процес анотування потребує експертів, на відміну від, наприклад, датасетів класифікації зображень, маркування яких могла б робити й дитина.

Через це для підвищення кількості доступних даних часто використовують штучні дані (Data Augmentation). На кожній епосі над вхідними зображеннями проводиться череда змін: горизонтальне віддзеркалювання, оберти, zoom, cropping, фільтр Гаусса, тощо. В результаті CNN отримує сильно видозмінене зображення, яке для мережі є фактично абсолютно новим.

2.8 Tensorflow та keras

На даний момент розробка нейронних мереж сильно відрізняється від того, з чим доводилось працювати на етапах зародження індустрії. Існує кілька дуже потужних фреймворків для побудови та тренування моделей. Одним з таких є TensorFlow, розроблений Google в 2017 році. Ім’я бібліотеки походить від двох понять – тензор (Tensor) та графи потокових даних (stateful dataflow graphs). Тензори – це багатовимірні масиви даних, а через потокові графи бібліотека визначає операції на ними.

TensorFlow часто використовують як бібліотеку для реальних проектів, частково через TPU (Tensor Processing Unit) розроблений Google щоб зменшити вартість роботи вже натренованої моделі. Отримати доступ до них можна через хмарний сервіс Google.

Крім того TensorFlow має гарну підтримку персональних проектів у вигляді Google Colab – онлайн сервіс що дозволяє запускати Jupyter блокноти прямо з браузера, до того ж безкоштовно. Таким чином можна тренувати відносно великі мережі не маючи потужного графічного процесору.

Не зважаючи на усі плюси TensorFlow, писати код на ньому може бути складно та потребує значних затрат часу. Для виправлення цього використовується бібліотека Keras. Ціль її – спростити написання мереж, зменшити когнітивне

навантаження на розробника. Keras побудовано на Tensorflow 2.0 і код написаний за допомогою нього так само можна використовувати в Google Colab. На даний момент це абсолютно найпопулярніший спосіб написання мереж, топ 5 найкращих користувачів Kaggle, популярного сайту для змагань з машинного навчання, використовують саме Keras.

2.9 Датасети комп'ютерного зору

Історично для тренування мереж сфери комп'ютерного зору існує кілька важливих датасетів. Має сенс розглянути їх зміст та цілі, аби краще розуміти розвиток CNN мереж.

Для демонстрації потужності backbone-мереж часто використовується датасет ImageNet. ImageNet має в собі 14 мільйонів зображень, кожне з яких має один з більше ніж двадцяти тисяч визначених класів. Через надзвичайний розмір датасету зазвичай усі змагання тренуються на його підмножині ILSVRC. На ранніх етапах комп'ютерного зору за допомогою машинного навчання цей датасет був найпрестижнішою ціллю. І зараз більшість state-of-the-art мереж використовує ImageNet для тренування. На сьогоднішній день рекорд тримає мережа CoCa з 91% точності в класифікації.

Наступні два визначних датасети напряму використовуються для сегментації зображень. На відміну від датасетів для класифікації, ці датасети не використовують точність як основний показник якості мережі. Натомість використовується MIoU – Mean Intersection over Union.

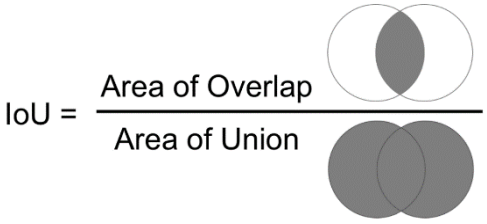

Рисунок 5, формула обчислення Inersection over Union з графічною репрезентацією

Intersection over Union вираховується окремо для кожного класу, шляхом ділення площі пересічення прогнозованого класу з реальним класом на анотації на загальну площу об’єднання прогнозованого та реального класу. Mean частина означає що вираховується середня IoU для усіх класів.

Cityscapes складається з 25000 тисяч зображень урбаністичних доріг в 50-ти різних містах під час різних пір року та годин дня. 5000 тисяч з цих зображень чітко анотовані, інші анотовані грубо. Датасет має 8 класів: плоскі поверхні, люди, транспортні засоби, будівлі, об’єкти, природа, небо та пустота. Одна з визначних характеристик датасету це великий розмір зображень в ньому (2048x1024) і близькі до реальних проблем умови, зокрема для автономних транспортних засобів. На даний момент цей датасет є найпопулярнішим у сфері сегментації зображень.

MS Coco (Microsoft Common Objects in Context) – це великий (328 тис., 164 тис. анотованих зображень) датасет для виявлення багатьох проблем комп’ютерного зору. В ньому присутні анотації для виявлення об’єктів, опису зображень природньою мовою, сегментації об’єктів (з 91 класом), паноптичної сегментації, тощо.

РОЗДІЛ 3: Архітектури згорткових мереж

3.1 Високорівнева структура CNN

В минулих розділах було розглянуто низькорівневу структуру типової конволюційної мережі. Проте в більшості випадків не прийнято будувати усю мережу повністю. Причиною для цього є те, що конволюційна частина усіх мереж в цілому дуже схожа, а її багатошарова архітектура потребує великої кількості тренувальних даних та багато часу для досягнення гарних показників при тренуванні. Через це зазвичай CNN розділені на 2 частини – основа (backbone) та імплементація, з яких основа це одна з state-of-the-art претренованих моделей без оригінальної імплементації (без останніх кількох повнозв’язаних шарів). Для повного аналізу доступних CNN рішень має сенс розглянути представників обох частин.

3.2 Основи (backbone) для сегментації зображень

Створення потужної backbone архітектури це зазвичай дуже складна задача. Над створенням нових state-of-the-art моделей в цій категорії працюють великі команди експертів за підтримки гігантів індустрії, таких як Google чи Microsoft. Найбільші прориви в сфері глибокого машинного навчання пов’язані якраз з новими розробками у цій сфері. В цьому пункті буде розглянуто історично важливі, часто вживані та новітні backbone мережі.

3.2.1 AlexNet

На початку 2010-х не було загально визнаного підходу до проблеми комп’ютерного зору в контексті нейронних мереж. Було багато підходів, проте жоден з них не давав значних покращень. В 2012 році група дослідників випустила у світ свою наукову роботу «ImageNet Classification with Deep Convolutional Neural

Networks», в якій було описано мережу майже ідентичну сучасним. Згодом описана архітектура отримала назву AlexNet на честь одного з дослідників. Конволюційні мережі

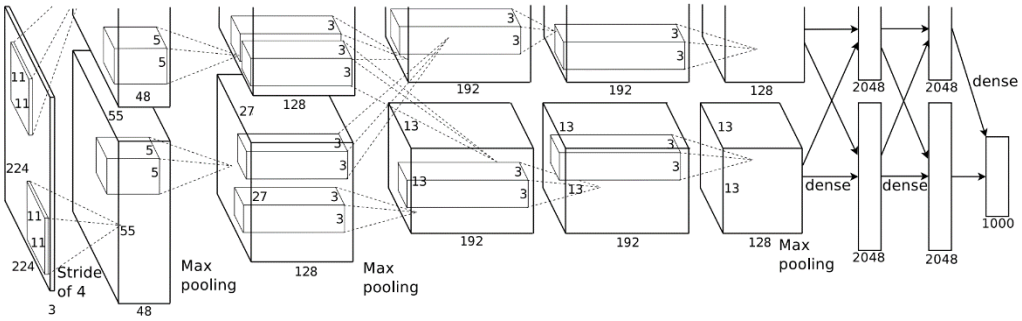

Рисунок 6, архітектура AlexNet

існували і до того, проте не в такому машстабі.

Архітектура зазначера в роботі має усі типічні сьогодні елементи – шари конволюцій з RelU активацією, слідом за ними Max pooling, в кінці мережі 3 dense шари, з фінальним вектором у 1000 елементів для класифікації датасету ILSVRC-2010. Важливо відмітити імплементацію у вигляді, по суті, двох паралельних мереж. Таке рішення виходило з технічних причин – тренування моделі довелося реалізувати на двох графічних процесорах GTX 580 з 3 GB пам’яті, оскільки пам’яті одного не вистачало щоб вмістити усю модель. Кожний GPU тренував свою модель, після певних етапів вони оновлювали параметри одне одного, записуючи їх напряму в пам’ять процесора.

Сама архітектура AlexNet була революційною сама по собі, але крім цього автори також дослідили альтернативи для функцій активації. На той час типовою активацією була сигмоїдна функція. Автори ж зазначили, що мережа з простою ReLU сходилася в 6 разів швидше.

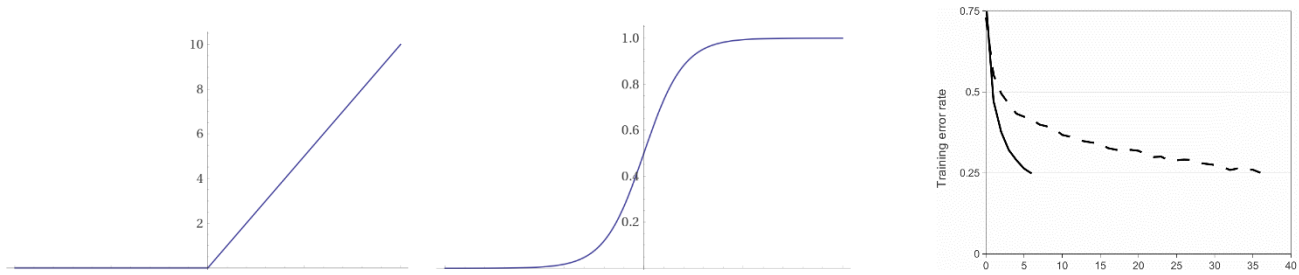

Рисунок 7, функція активації ReLU (зліва), сигмоїдна функція (по центру) та різниця в швидкості тренування з ними (справа, цільна лінія - ReLU, пунктирна - сигмоїдна)

Також це одна з перших мереж які використовували штучні дані, особливо дуже важливі зміни, такі як горизонтальне відзеркалювання та обрізання, які зараз є майже обов'язковими про аугментації даних.

Результати мережі на той час вражали, 37% помилок на тестовому датасеті ILSVRC, в порівнянні з ~45% в найближчих конкурентів.

3.2.2 ResNet

Після великого прориву яким був AlexNet довгий час глибокі конволюційні мережі не показували значного розвитку. Зі збільшенням пам'яті та обчислювальної потужності на графічних процесорах з'явилась можливість будувати ще глибші мережі, тим самим покращуючи їх результати. Найбільша мережа на той час була VGG19 з 16-ма конволюційними шарами та трьома dense шарами. Проте в певний момент подальше збільшення кількості шарів перестало давати кращі результати. Однією цілком вірогідною причиною могло бути перенавчання, проте група дослідників з Microsoft виявила невідповідність між цим припущенням та фактичними тестовими даним.

За їх дослідженнями зі збільшенням кількості шарів в конволюційній мережі, зростала не тільки кількість помилок на тестововму датасеті, а й на тренованому. Відповідно причиною цьому не могло бути перенавчання.

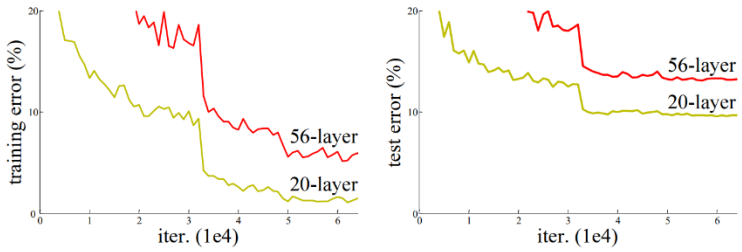

Рисунок 8, відсоток помилок на тренувальних та тестових датасетах двох різних мереж

Їх гіпотеза була наступна: кожна мережа з більшою кількість шарів може бути призведена до мережі з меншою кількістю шарів, якщо усі надлишкові шари першої мережі будуть виконувати identity операцію, тобто взагалі не змінювати вхідні дані.

Оскільки відсоток помилок росте з кількістю шарів, і після певного шару вхідні дані не мають зазнавати значних змін, то мережа не може навчитися проводити identity операцію і відповідно з часом погіршується.

Аби перевірити цю теорію вони створили так звані skip-connections, або shortcut connections – після обчислення блоку з двох конволюційних шарів F(x) перед другою функцією активації до F(x) додавалися вхідні дані x. Оскільки вивчити нульову функцію набагато простіше ніж identity, в теорії така операція дозволяла дуже просто отримати identity функцію як результат.

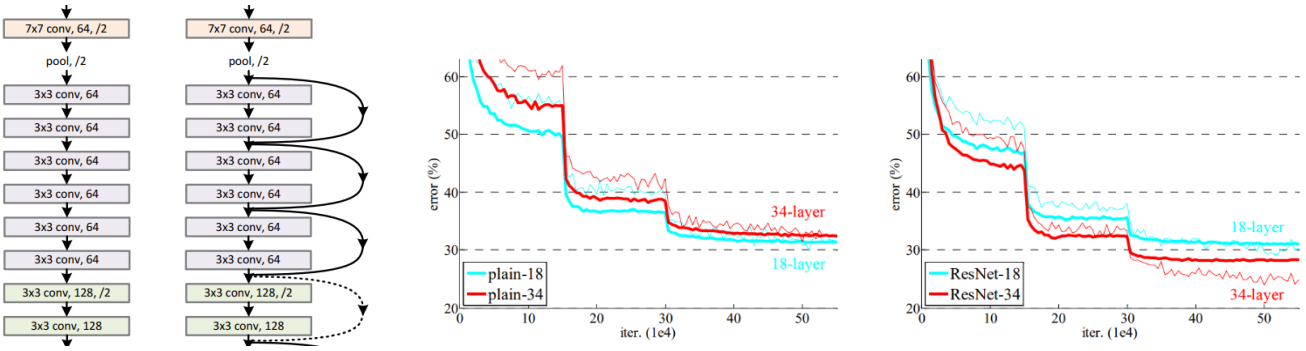

Рисунок 9, архітектура без skip-connections та з (зліва) та порівняння відсотку помилок в них (зліва)

Для перевірки було побудовано мережу глибиною в 34 шари з імплементованими в ній shortcut зв'язками. Результат довів правильність оригінальної теорії, відсоток помилок як в тренувальному так і в тестовому сеті даних сильно знизився. Для перевірки лімітів глибини з такими зв'язками було натреновано мережі з 50, 101 та 152-ма шарами, величезний розмір порівняно з конкурентами. Ці мережі автори називають ResNet-ами, оскільки вони використовують залишкові (residual) дані з попередніх шарів. В якості експерименту дослідники також розробили мережу з 1202-ома шарами, і хоча в ній відсоток помилок в тестовому датасеті підвищився, відсоток помилок в тренувальному сеті залишився низьким, відповідно було доведено що попередня проблема глибоких мереж була усунена.

Підвищення глибини значно покращило точність мережі. Тести на ILSVRC показали результат 21.43% помилок для ResNet-152, порівняно з 28.07% в VGG-16

та 24.27% в PReLU-net. Це значне покращення надало ResNet статус state-of-the-art мережі, і ResNet-152 та ResNet-101 все ще часто використовуються.

3.2.3 HRNet

HRNet – це одна з найновіших backbone архітектур. Вона була розроблена у 2019 році командою дослідників Microsoft. Типові backbone архітектури мають проблему зменшення якості оригінального зображення. Це випливає з самого підходу задачі, розміри зображення поступово зменшуються чим глибше в мережі воно оброблюється. Але наприклад для сегментації зображень чи аналізу поз людей позиція кожного пікселю часто має суттєве значення. Підхід HRNet, повна назва якої є High Resoltuion Net, обійти цю проблему оброблюючи одразу кілька різних розмірів зображення паралельно – оригінальне майже повно розмірне зображення (зменшене усього в 4 рази), та серія менших матриць. Як і у класичних CNN зі зниженням розміру росте кількість фільтрів.

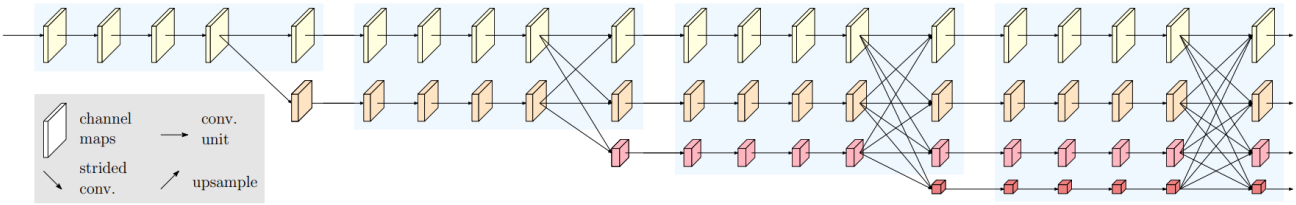

Рисунок 10, архітектура HRNet

В кінці кожного блоку кожний з розмірів обмінюється отриманими результатами з іншими. В якомусь сенсі цей підхід повторює ідеї, що були присутні

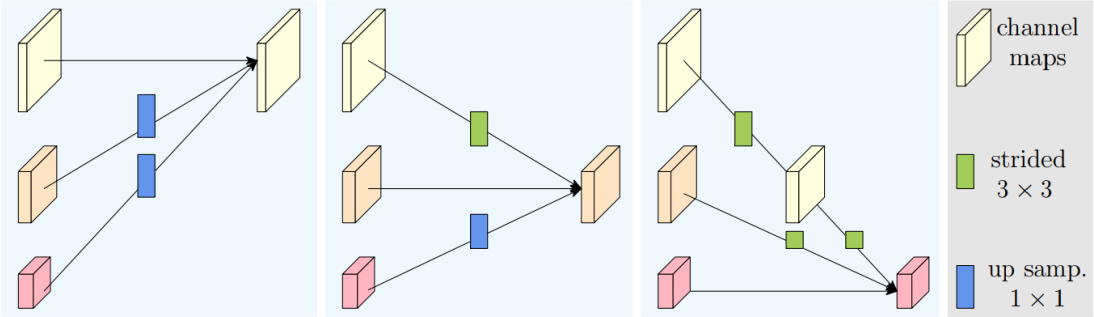

Рисунок 11, злиття у HRNet

ще в AlexNet, але тепер в іншому контексті. Обмін інформацією відбувається через злиття, під час яких для збільшення розмірів матриці застосовується одиничний UpSampling2D, для зменшення – конволюція.

На сьогоднішній день HRNet є найкращою по продуктивності архітектурою на ринку. Автори розробили мережі двох різних степенів глибини, w32 та w48. На тестуванні мережа показала усю свою потужність, досягнувши оцінки mIoU 81.6 на датасеті Cityscapes у 2019-ому році, з використанням імплементації сегментації написаної цією ж командою. В кінці того ж року рекорд знову було побито вже покращеною версією HRNet, HRNetV2+, тепер досягнувши 84.5%. Цей рекорд залишається неподоланим до сих пір.

3.3 Імплементації сегментації зображень

Незважаючи на важливість backbone для загального покращення точності конволюційних мереж, без потужної імплементації рішення задачі це лише пів процесу. В наступному сегменті буде розглянуто імплементації сегментації зображень.

3.3.1 U-Net

U-Net – це повністю ковнолюційна мережа (складається повністю з ковнолюційних шарів, без dense), створена за дизайном encoder-decoder. Суть дизайну в двох кроковій системі. Перша – кодування (encoding), видобування та тренування фільтрів, як і в будь-якій іншій CNN. Цей етап має сенс реалізовувати використовуючи одну з претренованих backbone мереж, наприклад VGG чи ResNet. Другий крок, декодування (decoding), проводить upsampling отриманої в попередньому кроці feature map. Останній етап – конволюція 1x1. В результаті отримується матриця з тими самим висотою та шириною що і вхідна, але з глибиною рівною кількості класів.

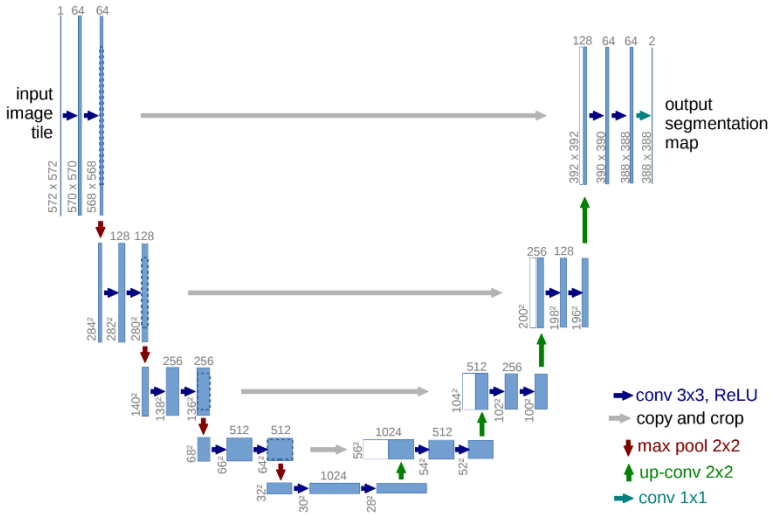

Рисунок 12, архітектура U-Net

Варто зазначити, що хоч конволюційні шари при декодінгу ідентичні конволюційним шарам, використаним на попередньому кроці, процес збільшення зображення на етапі upsample-інгу трохи відрізняється від max pooling-у. Є два еквівалентно дійсних підходи (як вони визначені у keras):

- UpSampling2D
- Conv2DTranspose

UpSampling2D з ядром 2x2 проходить по кожному пікселю вхідного зображення та інтерполює їх значення на вихідну матрицю квадратом 2x2. Плюс цього підходу в простоті. Інший, більш популярний підхід це Conv2DTranspose. Він більше схожий на конволюційний шар, але вхідні дані певним чином модифікуються шляхом додання різного padding-у. Значення пікселю рахується так само, через множення матриць. Цей варіанта додає ще один тренувабельний шар, дає можливість мережі визначити найважливіші характеристики.

Архітектура U-Net в цілому відповідає типовій encoder-decoder мережі. Але велика відмінність її в використанні певного роду skip з’єднань, які виконують

конкатенацію шару з енкодінгу та нового шару, отриманого з upsampleing-у. Через

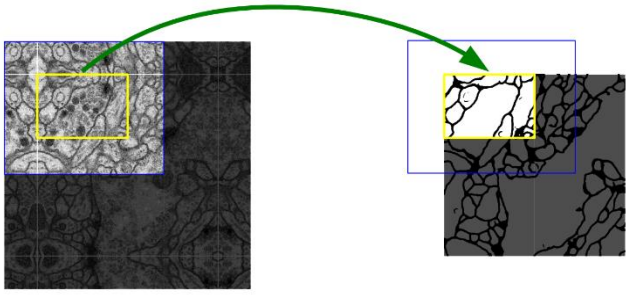

Рисунок 13, демонстрація padding-у в оригінальному U-Net

ці зв’язки мережу зображають у вигляді літери “U”, звідки й походить її назва.

Одне цікаве рішення оригінальної мережі у форматі padding-у, замість того щоб додавати певну кількість пікселів до матриці на етапі конволюції, було додано велику кількість віддзеркалених пікселів на краях зображення, які поступово, через використання “valid” padding-у, зникають при конволюції. Результуюче зображення менше ніж вхідне, проте втрачені пікселі усі були штучно додані. Через цей ефект доводиться обрізати матриці на skip з’єднаннях до того ж розміру що і матриця отримана при декодінгу.

Оригінальне призначення U-Net була сегментація зображень клітин на класи клітина та фон (через це вихідна матриця має глибину лише 2). На “ISBI cell tracking challenge” 2015 року мережа досягла показника IoU 92%, порівняно з 83% у другого кращого результату.

3.4 Імплементація U-Net з мережою backbone та без, демонстрація різниці у продуктивності

У наступному розділі буде розроблено та побудовано описану вище імплементацію для демонтрації процесу створення згорткової мережі та для безпосередньої демонстрації переваги у використанні backbone претренованих мереж.

3.4.1 Середа та датасети

Для демонстрації та порівняння процесу сегментації зображень було реалізовано серію мереж в Google Colab. Для задачі було використано Tensorflow та Keras. Датастетом на якому було б проведено тестування оригінально мав бути Cityscapes Image Pairs, датасет форматом дуже схожий на класичний Cityscapes, але менний і у відкритому доступі. Зображення в ньому подані у розмірі 512x256 та включають в себе як тренувальне зображення, так і його анотацію.

Рисунок 14, приклади елементів датасету Cityscapes Image Pairs

Проблема з якою довелося стикнутися в процесі розробки це дуже низька якість кольорів на масках. Через це об’єкти помічені, наприклад, червоним, не були однотонними, натомість мали велику кількість відтінків та артефактів. Для вирішення проблеми було випробувано використати певну обробку зображень, щоб отримати неушкоджені зображення. Проте ця проблема виявилась складнішою ніж очікувалось, границі між різними класами почали сильно зміщуватись, через великий відсоток артефактів біля них.

Через це було вирішено використати інший датасет, конкретніше ADE20K. Цей датасет є значно більшим, нараховує в собі 20 тисяч різноманітних зображень для семантичної сегментації. Усі зображення дуже чітко анотовані. Всього в

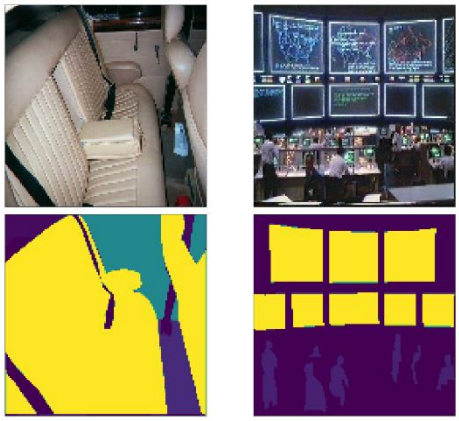

Рисунок 15, приклади зображень з ADE20K та їх масок

датасеті 150 семантичних категорій для об’єктів абсолютно різного типу: небо, людина, ліжко, тощо. Найбільшим недоліком датасету є його низька якість зображень (усього 128x128). До того ж велика кількість класифікованих об’єктів є складнішим завданням для конволюційної мережі, відповідно і кількість епох необхідна для гарної точності буде більша.

3.4.2 Перенесення датасету в середі Colab

Першим кроком було завантаження датасету в серед Colab. Аби не бути прив’язаним до системи на якій запускається датасет, його було завантажено на Google Drive, звідки система Colab може завантажити його напряму. Крім можливості вільно змінювати робочі машини за потребою, завантаження та збереження датасету на Google Drive також прибирає потребу завантажувати датасет при відключенні від хмарної машини, на якій працює середовище Colab.

Наступний крок після отримання датасету – завантажити його в середовище та опрацювати в потрібний спосіб. Для збереження датасету best practice підхід – це спеціалізований об’єкт tensorflow.Dataset. Для спрощення його створення було використано list_files, функція яка за наданим шляхом збирає усі підходящі файли. При генерації важливо вказати однаковий статичний seed для тренувального да валідаційного датасетів, який буде визначати порядок в якому зображення збираються в датасет. Усувати будь-які випадкові елементи вважається гарною

практикою в нейронних мережах, адже дозволяє в разі помилки або різкого погіршення продуктивності мережі дуже швидко і просто її відтворити та вирішити.

3.4.3 Опис архітектури моделей

Як основу CNN було взято натреновану на ImageNet ResNet50, що доступна напряму з бібліотеки Keras. В якості вхідних даних було надано тензор inputs розміром (256, 256, 3). Додатковим параметром `include_top=False` останні dense шари мережі було прибрано. Важливо також змінити параметр `trainable` на `False` аби при навчанні моделі ваги backbone не змінювались. Разом з цим також було створено варіант з власноруч написаною моделлю, що повністю повторює форму класичної U-Net моделі.

3.4.4 Попередня обробка вхідних даних

У варіанті з без backbone було застосовано типову нормалізацію даних – зведення значень пікселів у простір від 0 до 1. У backbone варіанті ж було використано спеціальний шар `tf.keras.applications.resnet.preprocess_input`, який виконує кілька перетворень, необхідних для ResNet, такі як стандартна нормалізація, перетворення зображення з формату RGB у формат BGR, тощо. Також до певного відсотка зображень було застосовано горизонтальне віддзеркалення. Нормалізація та функції завантаження тестового та валідаційного дадасетів були анотовані `@tf.function`. Ця анотація будує з функції граф, що пришвидшує її виконання. Оскільки нормалізація проводиться для кожного зображення в датасеті, навіть мізерна прибавка в швидкості виконання таких невеликих функцій дає суттєву різницю в часі виконання операцій. Для завантаження тренувальних зображень також використовується паралелізація, кількість потоків визначається параметром TensorFlow AUTOTUNE. Цей параметр перекладає відповідальність за обрання необхідної кількості потоків на TensorFlow,

залежно від ресурсів доступних системі. Особливо це корисно в середовищі Colab, в якому реальне залізо на якому виконуються обчислення може змінитися при наступному запуску.

3.4.5 Використані інструменти роботи з датасетами

Наступним кроком до датасету додається випадковий shuffle, який змінює порядок вхідних даних на кожній епосі. Для підтримання практики усування непередбачуваних випадковостей, описаної на етапі завантаження, тут також вказується seed, такий же як і в перший раз. Крім того shuffle потребує заданого розміру буферу. Через те що великі датасети не вміщуються повністю в оперативну пам’ять, перемішувати їх класичними алгоритмами не вийде. Через це TensorFlow записує дані у буфер вказаного розміру та покроково обирає випадковий елемент який буде заміщений. Розмір буфера сильно впливає на рівень змішаності даних. Наприклад якщо буфер має всього 1 елемент, послідовність завжди буде однаковою. Чим менше в ньому елементів, тим відповідно меншою буде змішаність. У цьому випадку датасет не надто великий і має сенс його добре змішувати на кожному кроці, тому розмір буферу було обрано у 1000 елементів.

У цій мережі також було використано інструмент repeat(). Завдяки ньому можна повторювати датасет кілька разів на кожній епосі. Проте в цей раз кількість повторі не була вказана, з ціллю зациклити датасет у нескінченний генератор. Такий підхід дозволяє не контролювати кількість оброблюваних елементів на етапі налаштування мережі, натомість регулювання відбувається прямо на етапі тренування. В цьому випадку це дозволяє повністю використовувати усі батчі, не залишаючи останній на половину порожнім. Щодо самих батчів даних, або за інакшою назвою партій даних, то це інструмент, який дозволяє оброблювати одразу кілька зображень одночасно. Ефективність досягається парною роботою CPU та GPU. Процесор підготовлює наступний батч, поки графічний процесор тренує

мережу. Таким чином GPU не простоює без роботи, поки збирається наступна партія. Розмір буферу суцільно залежить від потужності системи, але чим більші батчі за розміром, тим менше ітерацій в кожній епосі і тим швидше виконуються обчислення. Останній параметр визначений на датасеті – розмір prefetch-у. Prefetch ідеєю схожий на батч даних, його ціллю також є зменшити час простою між етапами обробки даних.

3.4.6 Опис побудови моделей

Класична імплементація U-Net має в собі 4 конволюційні групи в енкодері та 4 в декодері. ResNet же стискає зображення 5 разів, тому для класичної архітектури замість вихідних даних п’ятої групи буде використано вихідні дані четвертої, одразу після останньої активації. Крім того необхідно отримати матриці, що будуть конкатеновані на skip з’єднаннях. Отримати вихідне значення певного шару мережі можна за допомогою `get_layer`. На щастя в цьому випадку не треба реалізовувати обрізання матриць skip з’єднань, оскільки розміри вхідного зображення є ступенями двійки.

Після отримання усіх вихідних тензорів будується декодер частина імплементації. Як і в класичному U-Net кожна група складається з транспованої конволюції (`Conv2DTranspose(strides=2, padding="same")`), конкатинації з відповідним тензором skip з’єднань (`Concatenate()`), та двох конволюціних шарів (`Conv2D(num_filters, 3, padding="same", activation='relu')`), усього 4 групи. Останній крок, 1x1 конволюція з кількістю фільтрів рівний кількості класів датасету. На відміну від усіх інших шарів, в яких функція активації це ReLU, це шар має активацію softmax, яка розподіляє ймовірності для кожного класу.

3.4.7 Тренування моделей

Після зібрання моделей, їх було перевірено на правильність через їх summary, після чого вони були скомпільовані. Як параметри компіляції було використано оптимізатор Adam, найпопулярніший зараз оптимізатор, з низьким (0.0001) темпом навчання. Також було вказано функцію втрати, яка використовується при навчанні, було обрано `SparseCategoricalCrossentropy` як найпопулярніший та найстабільніший варіант для багато класових мереж. По завершенню чиста модель U-Net вийшла розміром у 31 мільйон тренованих параметрів, а мережа з backbone ResNet налічувала 20 мільйонів параметрів загалом, але тільки 8.5 мільйонів з них потребували тренування, решта відносилась до параметрів ResNet.

Перед початком тренування було проведено sanity check – прогноз моделі без будь якого тренування її параметрів та прогноз після однієї епохи. Цікаво зазначити ефект у ResNet на нетренованій системі. Порівняно з чистим U-Net на масці ResNet чітко видно грані комплексних об’єктів, в цілому зображення у багато способів нагадує вхідне фото. Це через претернований backbone, який вже зараз показує наявність великої кількості проаналізованих фільтрів.

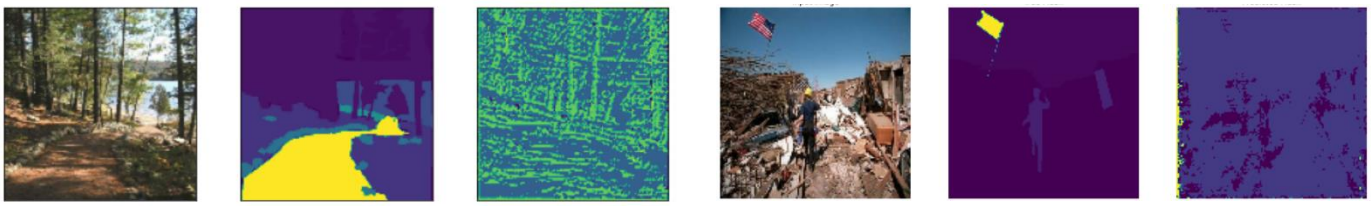

Рисунок 16, нетреновані передбачування ResNet (зліва) та чистого U-Net (справа)

Після успішного sanity check можна було приступати до тренування. На цьому етапі виник ряд проблем з небажанням Google Colab виділяти графічні процесори під тренування. Кілька спроб тренування без них показали що одна епоха займе близько 12-ти годин. Після переналаштування середовищ під використання GPU, обидві мережі закінчили тренування тривалістю у 20 епох. Такий об’єм

тренування не є достатнім для справді вражаючих результатів, проте досягти чогось більшого в доволі лімітованому середовищі Colab виявилось непросто.

3.4.8 Результати після тренування

Результати виконання є наступними:

```
Sample Prediction after epoch 20

Epoch 20: val_loss did not improve from 2.63558
4042/4042 [==============================] - 336s 83ms/step - loss: 2.7146 - accuracy: 0.3675 - val_loss: 2.7090 - val_accuracy: 0.3738
Epoch 20: early stopping
```

Рисунок 17, результати чистого U-Net

Оскільки чиста мережа мала тренуватись без попередньо вирахуваних значень для етапу енкодінгу, її точність залишає бажати кращого: 36.75% на тестовому датасеті та 37.38% на валідаційному. Зображення також демонструє неготовність мережі до повноцінної сегментації, хоча б не на цій епосі. Крім того варто відмітити дуже повільне зменшення loss параметру. Це свідчить що є ймовірність що модель навчається дуже повільно.

Результати U-Net з backbone демонструють значно кращі результати:

```
Epoch 18/20
202/202 [==============================] - 108s 533ms/step - loss: 1.2048 - accuracy: 0.6858 - val_loss: 1.7563 - val_accuracy: 0.5859
Epoch 19/20
202/202 [==============================] - 107s 532ms/step - loss: 1.1554 - accuracy: 0.6987 - val_loss: 1.7105 - val_accuracy: 0.5877
Epoch 20/20
202/202 [==============================] - 108s 533ms/step - loss: 1.1200 - accuracy: 0.7082 - val_loss: 1.7405 - val_accuracy: 0.5873
```

Рисунок 18, результати тренування U-Net з backbone після 20 епох

По перше варто відмітити набагато більшу швидкість тренування моделі, більше ніж у 3 рази. Така різниця у швидкостях є цікавою, адже кількість тренованих параметрів менша всього у приблизно 2 рази. Крім того точність за 20 епох дісталась 70% на тестовому датасеті і 58% на валідації. Варто зазначити що жодна з моделей не використовувала BatchNormalization щоб продемонструвати

Рисунок 19, передбачення U-Net з ResNet50 після першої епохи

ефект перенавчання на реальній ситуації, і в цих результатах це очевидно помітно.

Цікаве порівняння можна зробити, порівнявши результати мережі з backbone архітектурою після лише однієї епохи і фінальним результатом мережі без.


```
Sample Prediction after epoch 20

Epoch 20: val_loss did not improve from 1.78291
202/202 [==============================] - 123s 610ms/step - loss: 0.9316 - accuracy: 0.7643 - val_loss: 2.0289 - val_accuracy: 0.5607
Epoch 20: early stopping
```

Рисунок 20, результати покращеної backbone мережі з batch нормалізацією та спрогнозованою маскою на 20-ій епосі

Для експерименту кращу систему з backbone було запущено на 50 епох з batch нормалізацією. Проте вже після 20-ї епохи тренування зупинилось через відсутність росту точності на валідаційному датасеті. Це свідчить про присутню проблему

перетренування навіть з batch нормалізацією. Для теоретичного покращення роботи системи було б варто застосувати більш суттєві зміни на штучних даних та, можливо, додати до системи певний dropout.

Висновок

В ході роботи було розглянуто основні властивості згорткових нейронних мереж, принцип побудови, ефект кожного шару на процес навчання та основні засоби для покращення результатів навчання. Крім цього було розглянуто архітектури найпопулярніших та найвпливовіших backbone мереж, як саме вони змінили підхід до комп’ютерного зору у контексті машинного навчання. Також було розглянуто архітектуру імплементації сегментації зображень U-Net, інструменти які були використані в оригінальній реалізації моделі. В останньому розділі було розроблено та натреновано дві різні моделі для сегментації зображень U-Net в середовищі Google Colab на датасеті ADE20K, з backbone моделлю та без. Було явно продемонстровано переваги першого способу, як в швидкості тренування і необхідної кількості епох для задовільного результату, так і в загальній точності на тренувальному та на валідаційному .

Список джерел

1. “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms” [Електронний ресурс] / Xiao H., Rasul K., Vollgraf R. – 2017. – Режим доступу до ресурсу: https://arxiv.org/pdf/1708.07747.pdf.
2. Harrison J. “Don’t Use Dropout in Convolutional Networks” [Електронний ресурс] / Jansma Harrison. – 2018. – Режим доступу до ресурсу: https://kdnuggets.com/2018/09/dropout-convolutional-networks.html
3. “Batch normalization: Accelerating deep network training by reducing internal covariate shift” [Електронний ресурс] / Sergey Ioffe, Christian Szegedy – 2015. – Режим доступу до ресурсу: https://arxiv.org/pdf/1502.03167.pdf
4. “Semantic Segmentation with tf.data in TensorFlow 2 and ADE20K dataset” [Електронний ресурс] / Yann Le Guilly – 2020. – Режим доступу до ресурсу: https://yann-leguilly.gitlab.io/post/2019-12-14-tensorflow-tfdata-segmentation/
5. Papers With Code state-of-the-art networks leaderboard [Електронний ресурс] - Режим доступу до ресурсу: https://paperswithcode.com/task/semantic-segmentation
6. “Building a data pipeline” - [Електронний ресурс] / Andrew Ng, Kian Katanforoosh – 2021. – Режим доступу до ресурсу: https://cs230.stanford.edu/blog/datapipeline/
7. “ImageNet Classification with Deep Convolutional Neural Networks” - [Електронний ресурс] / Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton – 2012. – Режим доступу до ресурсу: http://www.cs.toronto.edu/~hinton/absps/imagenet.pdf
8. “Deep High-Resolution Representation Learning for Visual Recognition” - [Електронний ресурс] / Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, Wenyu Liu, Bin Xiao – 2020. – Режим доступу до ресурсу: https://arxiv.org/pdf/1908.07919.pdf
9. “U-Net: Convolutional Networks for Biomedical Image Segmentation” – [Електронний ресурс] / Olaf Ronneberger, Philipp Fischer, and Thomas Brox – 2015. – Режим доступу до ресурсу: https://arxiv.org/pdf/1505.04597.pdf
10. “Computer Vision Tutorial: A Step-by-Step Introduction to Image Segmentation Techniques (Part 1)” – [Електронний ресурс] / Pulkit Sharma – 2019. – Режим доступу до ресурсу: https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/

11. “Deep Residual Learning for Image Recognition” – [Електронний ресурс] / Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun – 2019. – Режим доступу до ресурсу: https://arxiv.org/pdf/1512.03385.pdf