КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня «магістр»

НА ТЕМУ:

Інтелектуальний модуль розпізнавання інсульту

Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітньо-наукова програма «Технології штучного інтелекту»

Виконав:
студент(ка) 2 курсу магістратури, групи ТШІ-21

Москаленко Д.О
(ПІБ)

Науковий керівник: Іларіонов О.Є.
(ПІБ)
Ктн, доцент
(науковий ступінь, вчене звання)

Засвідчую, що в цій кваліфікаційній роботі немає запозичень з праць інших авторів без відповідних посилань

Студент(ка) _____
підпис

Кваліфікаційна робота допущена до захисту
рішенням кафедри *інтелектуальних технологій*

Протокол № _8_ від «_5_» _травня_ 2022 р.

Зав. кафедри _____ ПІБ
підпис

Київ 2022

РЕФЕРАТ

Робота містить 63 сторінок тексту, 47 рисунки, 5 таблиці, 17 посилань на літературні джерела та 1 додаток.

Актуальність. Кожен рік інсульт вражає близько 80,1 млн людей в усьому світі та пов’язаний з величезними соціальними витримками, тому залишається найважливішою медико-соціальною проблемою через його високу частку. У структурі захворюваності та смертності населення значні показники тимчасової втрати роботи та первинна інвалідність.

Розуміння зображення та видобування інформації із зображення є важливою областю застосування в технології графічних зображень. На практиці сегментація зображень не зосереджена на усіх частинах зображення, а лише на певних областях, які мають однакові характеристики. Сегментація зображення - це важлива область в обробці зображень та основа для розпізнавання зображень.

Тому актуальним є вдосконалення методів і пристроїв діагностики, які дозволять провести загальне, регулярне обстеження та забезпечити доступність і масовість при загальному регулярному діагностуванні широких верст населення. Оптичні прилади візуальної реєстрації досить поширені на індивідуальному рівні, так, наприклад, прилади для оптичної тканинної оксиметрії - аналізатори об’ємного капілярного кровонаповнення м’яких біологічних тканин.

Метою даної роботи є дослідження проблеми розробки системи розпізнавання ішемічного інсульту на основі КТ зображень з використанням методів машинного навчання та надати алгоритмічні частини у вигляді коду, розробка програмної системи та проведення експериментів.

Об’єктом дослідження є розпізнавання інсульту за допомогою методів машинного навчання.

.

Предметом дослідження є алгоритми апаратного та програмного розв’язання задачі розпізнавання ішемічного інсульту за допомогою КТ зображень.

Новизна отриманих результатів. Запропоновано та реалізовано користувацький алгоритм та модель розпізнавання ішемічного інсульту за допомогою КТ зображень з метою покращення точності передбачення уражень мозку.

Апробація результатів дослідження. Практично опрацьовано запропонований користувацький алгоритм розпізнавання ішемічного інсульту. 1 стаття була опублікована в журналі «Інноваційні наукові дослідження в умовах світових змін».

Структура та обсяг роботи. Кваліфікаційна робота на здобуття освітнього ступеня «магістр» складається з вступу, трьох розділів та висновків.

У вступі до магістерської роботи надано загальну характеристику роботи, визначено актуальність теми досліджень, сформульовано мету досліджень та наукову новизну отриманих результатів.

У першому розділі «Огляд літератури» проаналізовано сучасні відомості про стан питання, що досліджується, обґрунтовано актуальність досліджень, розглянуто ретроспективу рішень питання розпізнавання ішемічного інсульту людини за допомогою КТ зображень.

У другому розділі надано методичні аспекти досліджень, що проводились, сформульовано методичну основу питання розпізнавання ішемічного інсульту людини.

У третьому розділі описано алгоритм програмного застосунку та приведено його тестування з різними параметрами, логічно обґрунтовано вибір

архітектури та методів розробки системи. Проведено тестування програмного застосунку та зроблено аналіз отриманих результатів.

У висновках представлені підсумки проведених досліджень.

Ключові слова: розпізнавання інсульту, обробка зображень, методи машинного навчання, нейронні мережі, прогнозування.

ABSTRACT

Qualification work contains 63 pages of text, 47 figures, 5 tables, 17 references and 1 application.

Actuality of theme. Every year, stroke affects about 80.1 million people worldwide and is associated with enormous social exposure, so it remains a major medical and social problem due to its high proportion. In the structure of morbidity and mortality of the population there are significant indicators of temporary job loss and primary disability.

Understanding images and extracting information from images is an important area of application in graphic imaging technology. In practice, image segmentation does not focus on all parts of the image, but only on certain areas that have the same characteristics. Image segmentation is an important area in image processing and the basis for image recognition.

The purpose of the qualification work is to study the problem of developing a system for recognizing ischemic stroke based on CT images using machine learning methods and to provide algorithmic parts in the form of code, software development and experiments.

The object of research is the recognition of stroke using machine learning methods.

The subject of research is algorithms for hardware and software solution of the problem of recognizing ischemic stroke with the help of CT images.

Scientific novelty of the obtained results. For the first time, a custom algorithm and model for ischemic stroke recognition using CT images were proposed and implemented to improve the accuracy of brain lesion prediction.

Approbation of research results. The proposed user algorithm for ischemic stroke recognition is practically developed. 1 article was published in the journal "Innovative research in the context of world change."

The structure and scope of thesis. Qualifying work for the master's degree consists of an introduction, three sections and conclusions.

In the introduction to the master's thesis the general characteristic of the work is given, the urgency of the research topic is determined, the purpose of the research and the scientific novelty of the obtained results are formulated.

The first section "Literature Review" analyzes current information about the state of the issue under study, substantiates the relevance of research, considered a retrospective of solutions to the issue of recognizing ischemic stroke with the help of CT images.

The second section presents the methodological aspects of the research conducted, formulated the methodological basis for the recognition of ischemic stroke.

The third section describes the algorithm of the software application and presents its testing with various parameters, logically substantiates the choice of architecture and methods of system development. The software application was tested and the obtained results were analyzed.

The conclusions present the results of the research.

Keywords: stroke recognition, image processing, machine learning methods, neural networks, prediction.

Зміст

Вступ

Життєдіяльність людини та активне переміщення в просторі є найважливішою задачею тіла людини. Всі компоненти рухомої системи, включаючи реакцію на збудників, емоції, вибір направлення руху, подолання переваг, потребують чіткої комунікації між рухами кінцівок та тулуба, ефективного контролю м'язового тонусу та підтримки рівноваги. Порушення будь-якого компоненту цього контролю при захворюваннях та травмах нервової системи можуть призвести до рухових розладів, які сильно обмежують життєдіяльність пацієнтів, а часто і до смерті. За даними Всесвітньої організації охорони здоров'я (ВООЗ), злоякісні новоутворення (рак) і хвороби серця були першою і другою причинами смерті у 2016 році. Третьою провідною причиною смерті у 2016 році був інсульт, померли 5,7 мільйона. [1] У 2016 році інсульт був причиною 11% усіх смертей у світі, це виникає, коли приплив крові до мозку переривається, що призводить до некрозу клітин мозку. Взагалі ця хвороба частіше зустрічається у людей похилого віку, і може призвести до церебральної дисфункції, такої як геміплегія, неправильна вимова та втрата свідомості, також може спричинити тяжку інвалідність у дорослих і навіть смерть [2-3]. Інсульт є виліковним захворюванням, і при ранньому виявленні чи прогнозуванні його тяжкість може бути значно знижена.

Діагностика інсульту потребує високоякісної нейровізуалізації та подальшої сегментації поразки. Вимірювання розміру, розташування та збігу локації уражень інсульту з існуючими областями та структурами мозку потенційно можна використовувати для прогнозування ймовірності реабілітації та рекомендацій щодо лікування, яке буде максимально ефективним для конкретної людини. В якості предмету дослідження обрано зображення МРТ, так як вони являються золотим стандартом дослідження по даним ВОЗ.

На основі аналізу МРТ досліджень можна дослідити хворобу на ранніх етапах та запобігти летальним наслідкам. Завдяки досягненням в галузі штучного інтелекту, а саме у напрямку розпізнавання образів, детектування хвороби за допомогою штучних нейронних мереж стає все більш актуальним. Це відкриває нові можливості у сфері медичної діагностики.

Ця робота організована наступним чином:

- У розділі 1 представлена інформація про оригінальні методи дослідження інсульту, проводиться дослідження предметної області автоматичної сегментації уражених ділянок мозку на МРТ зображеннях, наводиться огляд існуючих медичних систем та методів сегментації. Формується постановка задачі та вимоги.
- У розділі 2 представлена модель прогнозування інсульту на основі штучного інтелекту, здійснюється проектування інформаційного та математичного програмного забезпечення.
- У розділі 3 проаналізовано експериментальні результати та характеристики моделей, описано програмну реалізацію процесу розпізнавання захворювання.

Об’єкт дослідження – засоби автоматичного розпізнавання ішемічного інсульту на основі Т1-зважених МРТ зображень.

Предмет дослідження – методи розпізнавання ішемічного інсульту на базі використання нейромережних технологій.

Мета - розробка алгоритму аналізу ішемічного інсульту на основі МРТ зображень мозку за допомогою нейромережевого модуля.

1 АНАЛІТИЧНИЙ ОГЛЯД ТА ПОСТАВНОКА ЗАВДАННЯ

1.5Актуалність дослідженя проблеми

За даними аналітичних агентств найпоширенішими захворюваннями головного мозку є: інсульт, хвороби Альцгеймера та Паркінсона, епілепсія, онкологія. Ці та багато інших захворювань можна досліджувати за допомогою МР - томографа, деякі з них мають характерні відмінні риси, що не важко поставити діагноз за знімком, але є хвороби, які складно лише за знімком відрізнити один від одного.

Наприклад, такі захворювання, як:

- Абсцес;
- ішемічний інсульт (інфаркт мозку);
- понтинний мієлініоліз розсіяний склероз;
- енцефаліт.

У цих захворювань різна причина виникнення, але на МРТ вони виглядають схоже, всі мають спільні характерні риси: високий сигнал на DWI зображенні і низький на ADC картах. Є ряд ознак, за якими лікар рентгенолог може відрізнити між собою захворювання, наприклад, за інтенсивністю сигналу, за контурами та формою ураження, за місцем знаходження ураження, із застосуванням додаткових МРТ програм дослідження. Однак не всі захворювання можна розпізнати за знімком. Таке захворювання, наприклад, як енцефаліт потребує додаткової діагностики для постановки діагнозу, так як МР-сигнал на DWI і сигнал на ADC картах схожий із сигналом ішемічного інсульту.

Кожен рік інсульт вражає близько 80,1 млн людей в усьому світі та пов’язаний з величезними соціальними витримками, тому залишається найважливішою медико-соціальною проблемою через його високу частку. У структурі захворюваності та смертності населення значні показники тимчасової втрати роботи та первинна інвалідність. [4]

Абсолютні показники захворюваності, смертності та поширеності інсульту дозволяють розраховувати соціально-економічні втрати. Опубліковані в 2015 році дані міжнародного проєкту з вивчення глобального тягаря хвороб (Global World Diseases - GBD) показали, що щорічно реєструється 10,3 млн випадків інсульту, у тому числі 6,5 мільйонів закінчуються смертю. Тому на сьогоднішній день проблема залишається актуальною.

1.2 Аналіз методів розпізнавання образів за допомогою графічних образів

Розуміння зображення та видобування інформації із зображення є важливою областю застосування в технології графічних зображень. На практиці сегментація зображень не зосереджена на усіх частинах зображення, а лише на певних областях, які мають однакові характеристики. Сегментація зображення - це важлива область в обробці зображень та основа для розпізнавання зображень.

Сегментація зображень заснована на певних критеріях для поділу вхідного зображення на декілька категорій для видобування потрібної інформації. [8]

1.2.1 Методи засновані на кластеризації

Є багато теорій для сегментації зображень, але з вивченням нових теорій і методів різних дисциплін багато методів сегментації зображень поєднано з деякими окремими теоріями та методами. Набору подібних елементів відповідає клас. Певним вимогам відповідає і закон класифікації об’єктів Для сегментації пікселів у просторі зображення з відповідним точок простору ознак використовується метод кластеризації простору ознак. Відповідно до їх агрегації в просторі ознак, простір ознак сегментується, а потім вони відображаються на початковому просторі зображення для отримання результату сегментації. Метод k-середніх є одним із найпоширеніших алгоритмів кластеризації. Основна методу k-середніх - зібрати зразки в різні кластери відповідно до відстані. Базовий алгоритм наведений нижче:

1. Вибрати K центрів кластерів, випадково або на основі деякої евристики.

2. Помістити кожен піксель зображення в кластер, центр якого найближче до цього пікселя.

3. Знову визначити центри кластерів, усереднюючи всі пікселі в кластері.

4. Повторювати кроки 2 і 3 до збіжності (наприклад, коли пікселі будуть залишатися в тому ж кластері).

Перевага алгоритму К-середніх полягає в тому, що алгоритм є швидким і простим, він є високо ефективним і масштабованим для великих наборів даних. І його часова складність близька до лінійної, і підходить для інтелектуального аналізу великих наборів даних. Недоліком алгоритму К-середніх є те, що його кількість кластерів K не має явного критерію вибору і її важко оцінити. Також на кожній ітерації алгоритму метод К-середніх проходить через всі зразки, тому час роботи алгоритму є затратним. Нарешті, алгоритм К-середніх є методом розділення на основі відстані. Він може застосовуватись лише до опуклих наборів даних і не підходить для кластеризації невипуклих кластерів.

1.2.2 Метод сегментації з виявленням границь

Сегментація зображень є важливим етапом аналізу зображень. Сегментація розділяє зображення на його складові частини або об'єкти. Рівень, за яким здійснюється поділ, залежить від проблеми, яка вирішується. Якщо об’єкти, що цікавить програму, будуть недоступні, то сегментація повинна припинитися. Алгоритми сегментації зображень, як правило, засновані на розривність і подібність значень інтенсивності зображення. Підхід до розриву полягає в розділенні зображення, засноване на різких змінах інтенсивності, і схожість заснована на розділенні зображення на регіони, подібні за набором попередньо визначених критеріїв.

Сегментація зображення — це процес поділу цифрового зображення на кілька областей або наборів пікселів. По суті, в розділах зображень є різні об’єкти, які мають однакову текстуру або колір. Результати сегментації зображення — це набір областей, які охоплюють все зображення разом і набір

контурів, витягнутих із зображення. Усі пікселі в регіоні мають однакові характеристики, такі як колір, інтенсивність або текстура.

Існують різні підходи: шляхом знаходження границь між регіонами на основі розривів у рівнях інтенсивності, порогові значення на основі розподілу властивостей пікселів, таких як значення інтенсивності, і ті, які основані на безпосередньому пошуку регіонів. Таким чином, вибір техніки сегментації зображення залежить від проблема, що розглядається.

Основні ознаки можна виділити з границь зображення. Виявлення границь – це основна функція для аналізу зображень. Ця функція використовується розширеним комп'ютерним баченням алгоритму. Виявлення границь використовується для виявлення об'єктів, які обслуговують різні програми, наприклад: обробка медичних зображень, біометричні дані тощо. Виявлення границь є активною сферою досліджень, оскільки це полегшує аналіз зображень вищого рівня. Існує три різні типи розривів рівню сірого : точка, лінія та краї. Просторові маски можна використовувати для виявлення всіх трьох типів розривів в зображенні.

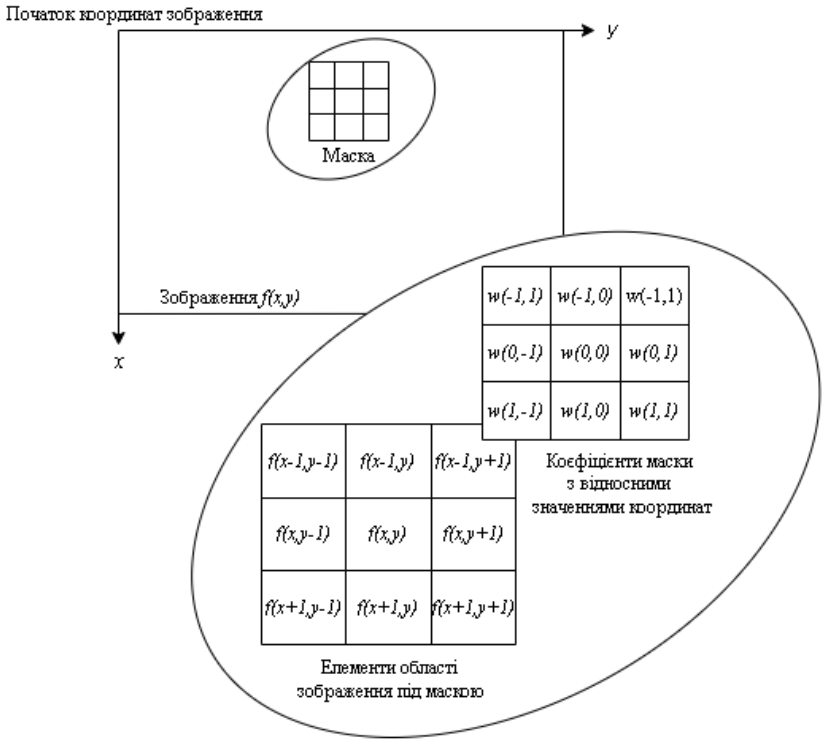


Рисунок 1.1. Схема просторової фільтрації

Процес просторової фільтрації відбувається шляхом переміщення маски від точки до точки на зображенні; У точках (x, y) фільтр відповіді обчислюється за допомогою фіксованих посилань на зображення. У лінійній фільтрації фільтр відповідає сумі коефіцієнтів добутку та відповідних значень пікселів у області фільтра для маскування фільтра. Для маски 3x3, відгуком в точці (x, y) лінійне рівняння :

$$R = w(-1,-1)f(x-1,y-1)+w(-1,0)f(x-1,y)+\ldots+w(0,0)f(x,y)+\ldots+w(1,0)f(x+1,y)+w(1,1)f(x+1,y+1)$$

(1.1)

Виходячи з формули, можна побачити, що це сума добутків коефіцієнтів маски та значення пікселя безпосередньо під маскою. Зокрема коефіцієнт w(0,0) розташований у точці f(x,y), що вказує на те, що маска зосереджена в точці (x,y).

Існує багато методів виявлення країв для сегментації зображення. Одним з найпоширеніших оператор першого порядку є оператор Прюітта, оператор Робертса та оператор Собеля .

Оператори які мають дефиренциальний оператор другого порядку є такі оператори як Лапласа , Кірша та Уолліса.

Оператор Робертса (Roberts)

Оператор Робертса виконує прості та швидкі обчислення, вимірюючи 2D просторові градієнти на зображеннях. Цей підхід підкреслює області високої просторової частоти, де часто зустрічаються кордони. Вхід оператора – це зображення сірих відтінків, таке ж, як і на виході, що головним в використанні цією технології. Значення пікселів кожної вихідної точки представляє оцінений загальний простір градієнта вхідного зображення в цій точці.

Нехай область 3x3 представляє значення яскравості біля певного елемента зображення.

Талиця 1.1 Окрестність 3*3 всередині зображення

z1	*z2*	*z3*
z4	z5	*z6*
z7	*z8*	*z9*

Оператор перехресного градієнта Робертса є одним с способів знайти часткові похідні першого порядку на точкових зображенням[13] :

$$Gx = (z9 - z5) \quad 1.2$$

$$Gy = (z8 - z6) \quad 1.3$$

Шляхом обробки зображення можуть бути знайдені похідні, за допомгою процесу фільтрації, який був описаній вище.

Таблиця 1.2. Маски оператора Робертса

-1	0
0	1
0	-1
1	0

Оператор Превитта

Оператор Превитта так, як и оператор Робертса використовує область зображення 3*3, але оперує іншими виразами.

$$Gx = (z7 + z8 + z9) - (z1+z2+z3) \quad (1.4)$$

$$Gy = (z3 + z6 + z9) - (z1 + z4 + z7) \quad (1.5)$$

В указаних вище формулах різниця між сумами вздовж верхнього та нижнього рядків околиці 3x3 є апроксимацією похідної вздовж осі x, тоді як різниця між сумами вздовж першого та останнього стовпців околиці — вздовж похідної осі

х та осі у. Для реалізації цих формулювань використовується оператор, описаний маскою на малюнку 4, який називається оператором Превітта.

Таблиця1.3 Маски оператора Превитта

-1	-1	-1
0	0	0
1	1	1
-1	0	1
-1	0	1
-1	0	1

Результати обробки зображень операторами Робертса та Превитта показано на малюнках нижче.

Рисунок 1.2 Початке зображення

Рисунок 1.3 Зображення після обробки методов Робертса

Рисунок 1.4 Зображення після обробки методом Превитта

1.3 Розпізнавання зображень в медицині

1.3.1 Магнітно-резонансна томографія як метод діагностичної візуалізації

Незалежно від тяжкості захворювання частота інсульту у широкому діапазоні серед наукових дисциплін, процес відновлення після інсульту, досі не вивчений до кінця. Вивчення головного мозку та поведінка людини після інсульту допоможе більш широко розібратися в цьому. Візуалізація мозку дає змогу отримати перспективні біомаркери (наприклад, показують структуру мозку або його функції), які потенційно можуть передбачити імовірність відновлення , допомогти обрати лікування, яке може бути максимально ефективне та індивідуальне . Це дозволить підвищити ефективність ресурсу у клінічній практиці та клінічних випробуваннях. Крупномасштабні мережі передачі даних нейрнні-зображення після інсульту є перспективним підходом для досягнення найкращого розуміння процесу поновлення.

Ішемічний інфаркт (інсульт) - органічна поразка ЦНС, спричинена гострим порушенням мозкового кровообігу з розвитком ішемії нервової тканини та появою інфаркту, що супроводжується характерними морфологічними проявами на візуалізації (МРТ та КТ). Одним з 5 Методами діагностики інсульту є МРТ головного мозку. Це безпечний метод обстеження,

в основі якого лежить ефект магнітного резонансу та електромагнітне випромінювання, що по-різному відбивається від більш менш щільних тканин. МРТ показує результати сканування пацієнта у трьох площинах, а також дозволяє чітко побачити проблеми із м'якими тканинами. Існує кілька режимів МРТ, наприклад. T1 BI, T2 BI, T2 FLAIR, DWI і т.д. Режими призначені для розгляду тканин, які мають різні фізичні властивості, щоб відрізнити нормальні тканини від патологічних. На рисунку 1.1 представлені види інсульту та інтенсивність сигналу залежно від періоду та режиму МРТ.

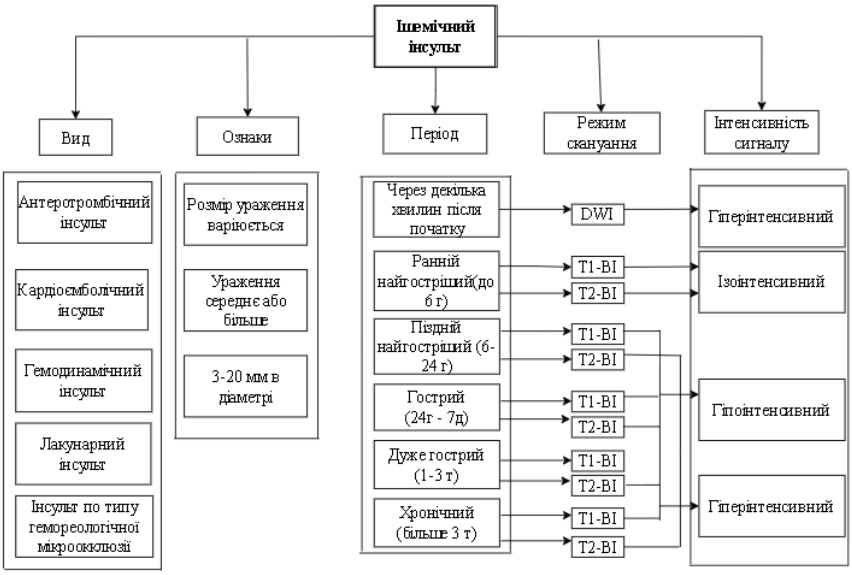


Рисунок 1.5 Види інсульту та режими сканування МРТ

На рисунку 1.6 можна побачити, що сканування у перші хвилини після інсульту лише режимі DWI простежується сигнал, який характеризується світлими ділянками на зображенні. У режимі T1 BI та T2 BI перші ознаки інсульту виявляються приблизно через 6 годин після його початку, на Tl BI сигнал стає гіпоінтенсивним, тобто ділянка поразки стає темною, на T2 BI сигнал стає гіперінтенсивним. На рисунку 1.2 наочно представлена порівняльна характеристика КТ та режимів МРТ у різні фази розвитку ішемічного інсульту

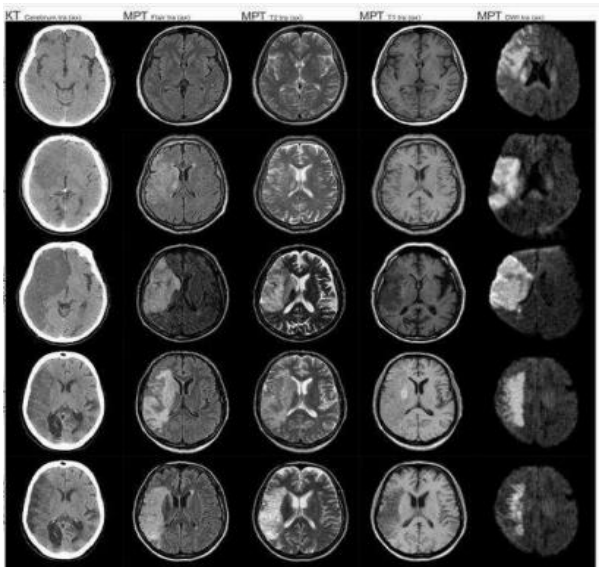


Рисунок 1.6 Порівняльна характеристика КТ та режимів МРТ

Як показано на рис. 1.6 найбільш інформативним методом діагностики ішемічного інсульту є МРТ в режимі DWI. На сьогоднішній день DWI є одним із найшвидших і специфічних методів діагностики інфаркту головного мозку на ранніх стадіях його розвитку (до 6 годин), коли є «терапевтичне вікно» для відновлення пошкодженої мозкової тканини, що є плюсом цього методу, на жаль, при цьому пацієнти звертаються за допомогою набагато пізніше. Але в будь-якому випадку на DWI-зображеннях в першу добу інсульт візуалізується вже як обмеження дифузії, що не видно в інших режимах МРТ. У гострій фазі інсульту для DWI область ураження головного мозку зазвичай має високий МР-сигнал тоді, як нормальні тканини мозку виглядають темними. На рисунку 1.7 представлені зображення МРТ в режимі DWI, зліва на МРТ порушення мозкового кровообігув речовині правого великого полушару, патологічне порушення має гіперінтенсивний синал, справа зображення здорового мозкуз ізоінтенсивним сигналом.

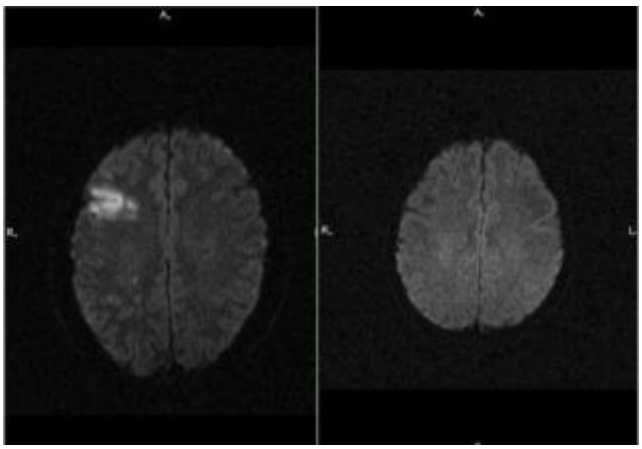


Рисунок 1.7 МРТ зображення ішемічного інсульту та здорової людини

На сьогоднішній день більшість алгоритмів сегментації ураження є для дослідження гострого інсульту. Нейровізуалізація на стадії загострення звичай підтримує мультимодальні послідовності МРТ (наприклад, дифузійно-зважене зображення, перфузійно-зважена візуалізація, зображення T2-FLAIR тощо) [5]. Увага дослідження спрямована на розробку оптимальних алгоритмів швидкої сегментації ураження та прогнозувати загальні клінічні результати, використовуючи їх мультимодальні послідовності. На рисунку 1.8 і 1.9 показано порівняння зображення МРТ з різними параметрами.

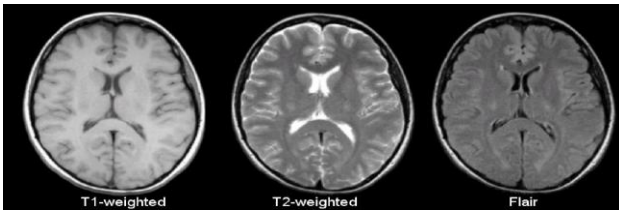


Рисунок 1.8 Порівняння T1, T2 та Flair-зважених МРТ зображень

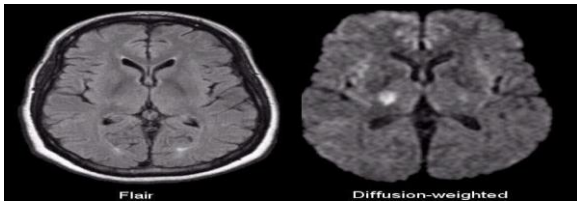


Рисунок 1.9 Порівняння Flair-зважених та дифузійно-зважених МРТ зображень

Дослідження в області постінсультної реабілітації зазвичай зосереджені на розуміння аналізу функціональних результатів після травми, наприклад, рух верхніх чи нижніх кінцівок, афазія та інше. Ці дослідження часто отримують дані від жертв інсульту на менш гострій стадії, від кількох тижнів до декількох років після інсульту. Як правило, для визначення ураження головного мозку отримують T1-зважені МРТ однакової

інтенсивності, а не множинні клінічні зображення, які зазвичай одержують у гострій фазі. Це пов'язано з тимчасовими та фінансовими обмеженнями, а також з тим, що у пізніших хронічних стадіях набряк та запалення не очікуються видимими. Натомість T1-зважені МРТ-зображення більш чутливі до прояву некрозу кори через два тижні після початку інсульту , отже, більше підходить для виявлення хронічних уражень інсульту. Нарешті, на реабілітацію часто ставиться завдання більш детального функціонального сканування мозку, наприклад у стані спокою або функціональна МРТ (фМРТ), або детальні анатомічні сканування, наприклад для дифузійної трактографії, розуміння конкретних функціональних і структурних закономірностей, пов'язаних з функціональним відновленням після інсульту.

Головна мета сегментації ішемічного інсульту (ISLES) – це щорічний конкурс із забезпечення стандартизованих мультимодальних клінічних МРТ наборів даних, що складається приблизно з 50-100 мізків із вручну сегментованими ураженнями [6]. Ця наукова ініціатива заохочує дослідницькі групи використовувати створені набори даних для оцінки алгоритмів сегментації ураження та дозволяє порівняти результати з іншими групами. За результатами представленими у вересні 2018, методи глибокого навчання зайняли переможні місця і показали кращу точність.

1.3.2 Виявлення діабетичної ретинопатії за допомогою цифрової фотографії дна сітківки ока.

Виявлення діабетичної ретинопатії за цифрової фотографії дна сітківки ока є одним із найуспішніших прикладів використання технології глибокого навчання в медицині [9]. Ранній скринінг на діабетичну ретинопатію дуже важливий, так як, раннє виявлення та лікування захворювання дозволяє запобігти погіршенню зору та сліпоту у швидко зростаючої кількості пацієнтів з діабетом. Такий скринінг також дозволяє діагностувати інші хвороби ока. Зростаюча потреба у проведенні такого скринінгу мотивувала на створення низької вартості технології його проведення Стандарти для скринінгу

діабетичної ретинопатії вимагають як мінімум 80%-ї чутливості тесту та 95%-ї специфічності. Аналіз цифрової фотографії медичним фахівцем забезпечує відповідно 78% та 96% при використанні фотографії під одним кутом зору, 96% та 89% рівні чутливості та специфічності при аналізі фотографій під двома кутами зору, 92% та 97% при використанні фотографій під трьома кутами зору.

Нещодавно були продемонстровані результати автоматичної діагностики з використанням алгоритму глибокого навчання. Навчання нейронної мережі здійснювалося на наборі із 100 000 фотографій під одним кутом зору. Кожне зображення попередньо оцінювалося групою з 3-7 офтальмологів для вироблення правильного укладання. Результати тестування алгоритму на двох наборах зображень при використанні однієї фотографії дуже вражають – якщо віддається пріоритет високої специфічності (тобто як можна меншій кількості хибнонегативних результатів), то досягається 90,3/98,1% та 87,0/98,5% чутливість/специфічність. Якщо віддається пріоритет високої чутливості, то досягаються 97,5/93,4% та 96,1/93,9% чутливість/специфічність. Таким чином, алгоритм, використовуючи тільки одну фотографію, справляється краще за людину, яка використовує кілька фотографій. Іншим прикладом успішного застосування технології нейронних мереж є дерматологічна діагностика новоутворень шкіри [10].

Рання діагностика раку та меланоми шкіри є непростою завданням, т.к. у структурі цих пухлин лише 3–5% становить меланома, яку припадає 75% смертей від новоутворень шкіри. Своєчасна діагностика меланоми дуже важлива, і у зв'язку з тим, що діагностика може бути виконана за фотографії, вже давно з'явилися послуги, які дозволяли людям надсилати фото ділянки шкіри, зробленого звичайним смартфоном, лікарю для аналізу. Точність такої діагностики не дуже велика: чутливість 49%, а специфічність 98%. Результати автоматичної діагностики з використанням алгоритму, що базується на технології згортаються нейронні мережі, дуже вражаючі.

Для навчання нейронної мережі використовувався набір з 125000 фотографій, сформований з 18 різних електронних сховищ. Потрібно було виконати трирівневу діагностику: перший рівень потрібно було визначити, чи містить зображення доброякісну або злоякісну пухлину або на ньому немає пухлинного ураження тканин. На цьому рівні точність склала 72,1%, що виявилося краще, ніж це вдалося зробити двом лікарям-дерматологам (у яких вийшло 66% і 65,56% відповідно). На другому рівні потрібно було віднести захворювання до одного з 9 класів, автоматична діагностика дала 55,4% точність, практично таку ж, як і в лікарів – 53,3% та 55% відповідно.

1.3.3 Розпізнавання та локалізації пухлини та її метастазів на цифровій мікрофотографії

Дуже вражають і результати застосування технології глибокого навчання до завдання автоматичного розпізнавання та локалізації пухлини та її метастазів на цифровій мікрофотографії [11]. Це завдання є дуже амбітним у зв'язку з тим, що навіть людський інтелект із цим класом задач справляється важко - відомо, наприклад, що потрібні роки та роки практики, щоб стати кваліфікованим патологоанатомом, здатним з високою точністю діагностувати пухлинні клітини та тканини. І навіть у цьому випадку рівень згоди у діагнозах патологоанатомів для деяких видів раку молочної залози становить лише 48%, схожий рівень – й у діагностиці раку простати.

Проект, спрямований на вирішення цього завдання, реалізовувався у компанії Google. Постановка завдання полягала в наступному: розробити програму, здатну по цифровій мікроскопічній фотографії розміру 1000000x1000000 пікселів визначити, чи міститься на зображенні пухлина, і, якщо міститься, вказати її місцезнаходження. Мінімальний розмір пухлини, що діагностується 100x100 пікселів. Для навчання та тестування алгоритму використовувався набір фотографій Camelion 16 dataset, містить 400 мікроскопічних фотографій тканин лімфовузлів: 270 слайдів з описом, що

використовувалися для навчання нейромережі, та 130 слайдів для її тестування На слайдах утримувалися як макропухлини (розміром понад 2000 μm), такта мікропухлини (розміром понад 200 μm і менше2000 μm). Додатково для оцінки точності діагностики дослідники оцифрували ще 110 фотографій тканин лімфатичних вузлів (57 з яких містили пухлини), отриманих від 20 пацієнтів. Для порівняння здібностей людського та штучного інтелекту тестовий набір аналізувався кваліфікованим патологоанатомом. За даними дослідження, найкращий варіант навченої нейронної мережі правильно ідентифікував 92,4% пухлин на тестовому наборі слайдів, причому патологоанатом-людина – лише 73,3%. Як видно з наведених даних, цей результат свідчить про те, що штучний інтелект у цій задачі впорався навіть краще, ніж людський.

1.3.4 Рентгенографія як метод діагностичної візуалізації

Рентгеноскопія застосовується для діагностики широкого спектра захворювань і пошкоджень. Таким чином, рентгенограми головного мозку є найпоширенішим медичним тестом візуалізації у світі та вирішальним для діагностики поширених захворювань головного мозку. Важливо, що в діагностиці частини цих захворювань рентгенівський знімок і його інтерпретація є превалюючим інструментом в постановці діагнозу.

Якщо говорити яка саме архітектура нейронних мереж є найбільш оптимальною для задач подібного типу. Для порівняння в дослідженні [7] використовувалися мережі трьох архітектур: нейронна мережа з навчанням методом зворотного поширення помилки (BPNN), нейронна мережа, яка працює по конкурентному принципу (CpNN) і згорткова нейронна мережа (CNN), розроблена фахівцями Care Mentor (Росія). Для навчання мереж по 12 рентгенологічним синдромам та оцінки ефективності роботи застосовували роздільні набори цифрових зображень рентгенограм органів грудної клітки у прямій проекції розміром 32 на 32 пікселі в форматах JPEG або PNG, отримані з відкритої бази Chestx-ray8.

Точність BPNN в розпізнаванні окремих рентгенологічних феноменів становить близько 81,03%, досягалася при невисоких витратах часу на навчання та помірному числі повторень. Значення середньоквадратичного відхилення не перевищувало 0,0026.

CpNN в силу особливостей архітектури та самонавчального алгоритму при мінімальних витратах часу на навчання дозволяла підвищувати точність визначення окремого рентгенологічного синдрому до 90,12%, однак величина похибки була відносно високою.

Випробування CNN на тестовій сукупності зображень показали найкращі результати по точності розпізнавання рентгенологічних змін і величини похибки, тоді як ресурсні витрати на навчання були найбільшими. Основними джерелами помилок є: помилки, обумовлені самою архітектурою нейронної мережі і алгоритмом її навчання; помилки, пов'язані з некоректною розміткою навчальних зображень.

Таблиця 1.4 Порівняння результатів продуктивності і точності синдромної класифікації типів нейронних мереж [7]

Архітектура нейронної мережі	Час навчання, сек	Точність розпізнавання, %	Максимальна величина середньоквадратичного відхилення	Число повторень
BPNN	640	81,03	0,0026	6000
CpNN	320	90,12	0,0034	1000
CNN	2700	93,61	0,0012	30000

Отже, найбільш ефективними і точними системами для інтерпретації мультикласифікаційних медичних зображень, зокрема рентгенограм ,є моделі, побудовані на основі архітектур багатошарових згорткових нейронних мереж.

1.4 Постановка задачі

Метою даної роботи є створення інтелектуальної системи детектування (виявлення) хвороби головного мозку, а саме інсульту, на основі рентгенівських, мрт знімків за допомогою штучних нейронних мереж.

Для досягнення даної мети були поставлені наступні завдання:

- розглянути існуючі нейромережеві алгоритми в контексті застосування до медичних завдань;
- знайти і підготувати набір даних, необхідний для навчання алгоритму;
- використовуючи комбінації та покращення існуючих алгоритмів реалізувати алгоритм детектування хвороби;
- провести тестування системи та оцінити точність її роботи;
- проаналізувати отримані результати.

Вхідна інформація:

- набір зображень, що входять до обраного датасету.

Вихідною інформацією є висновок про факт наявності хвороби головного мозку.

РОЗДІЛ 2 ПРОЕКТНІ РІШЕННЯ

2.1 Розробка архітектури інформаційної технології

2.1.1 Контекстна діаграма IDEF0

Перш, ніж почати розробляти програмне забезпечення, необхідно зрозуміти та описати бізнес-процеси, що дозволить в майбутньому не допустити примітивних помилок при розробці. Для опису процесів використовуємо нотацію IDEF0. Ідея цієї нотації лежить в відображені бізнес-процесу у вигляді прямокутника будут входити та віходити стрілки. Спершу проводимо опис системи та її взаємодії з навколишнім середовищем – контекста діаграма, а потім робимо функціональну декомпозицію, де вся система розбивається на окремі процеси та показується їх взаємодія – діаграма першого рівня.

Контекстна діаграма, виконана за допомогю онлайн інструменту FlowChart, зображена на рисунку 2.1.

Описуючи діаграму було визначено основні процеси, стрілки, сторона росположення яких має важливе значення :

- зліва входить стрілка «КТ знімок головного мозку», що і буде оброблятися в процесі, та дані пацієнта.

- зверху входить стрілка з регулюючим документом «Законодавство України про охорону здоров’я», що регулює процес лікування.

- справа виходить стрілка «результат класифікації», що показує що ми отривали на виході, обробляючи те, що отримали на початку, та карта с попередніми діагнозами, для відслідковування динаміки та порівняння з попередніми результатами.

- знизу входить стрілка «Лікар» та «Лаборант» - це регулюючі процеси, які перетворюють вхід на вихід.

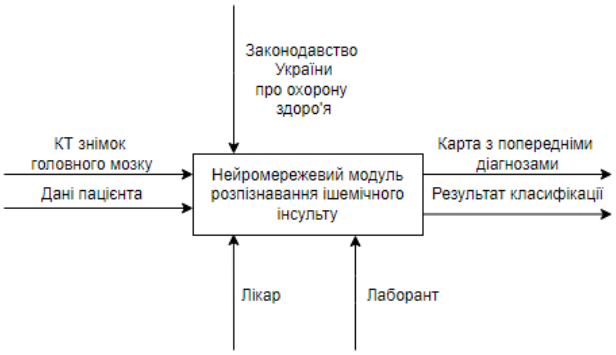


Рисунок 2.1 Контекстна діаграма IDEF0

2.1.2 Діаграма IDEF0 першого рівня

Далі, для точнішого аналізу, потрібно розбити наші процеси ще на процеси(декомпозиція). Декомпозуємо систему на пов'язані між собою процеси. Система поділяється на такі основні етапи:

1. Введення даних – результатом виконання даного процесу є вхідне КТ зображення, яке передається на наступний процес. КТ ;

2. Робота з зображенням – на виході маємо результат розпізнавання;

3. Формування результату класифікації.

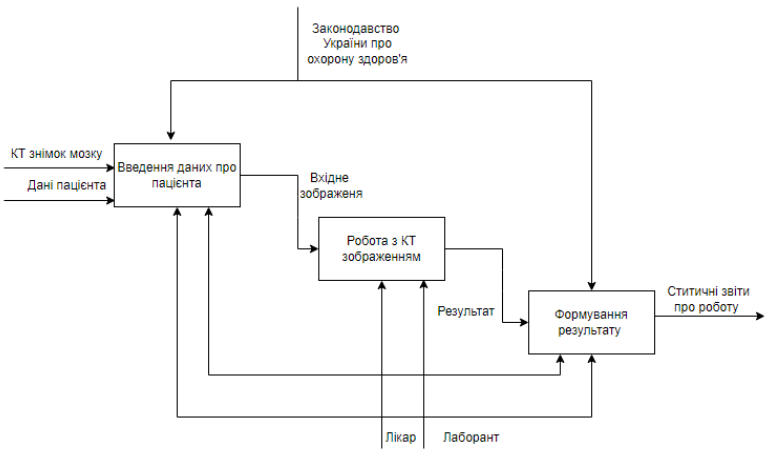


Рисунок 2.2 Діаграма IDEF0 першого рівня

2.1.3 Проектування архітектури за допомогою UML-діаграм

В першу чергу створюється діаграма, яка має показати загальну архітектуру. Такою UML-діаграмою є діаграма компонентів, що зображена на рисунку 2.4 Вона включає в себе два компоненти наборів даних (Brain Stroke CT Image Dataset), два компоненти сторонніх бібліотек (TensorFlow та Keras), основний класифікатор та інтерфейс вводу/виводу для роботи з класифікатором.

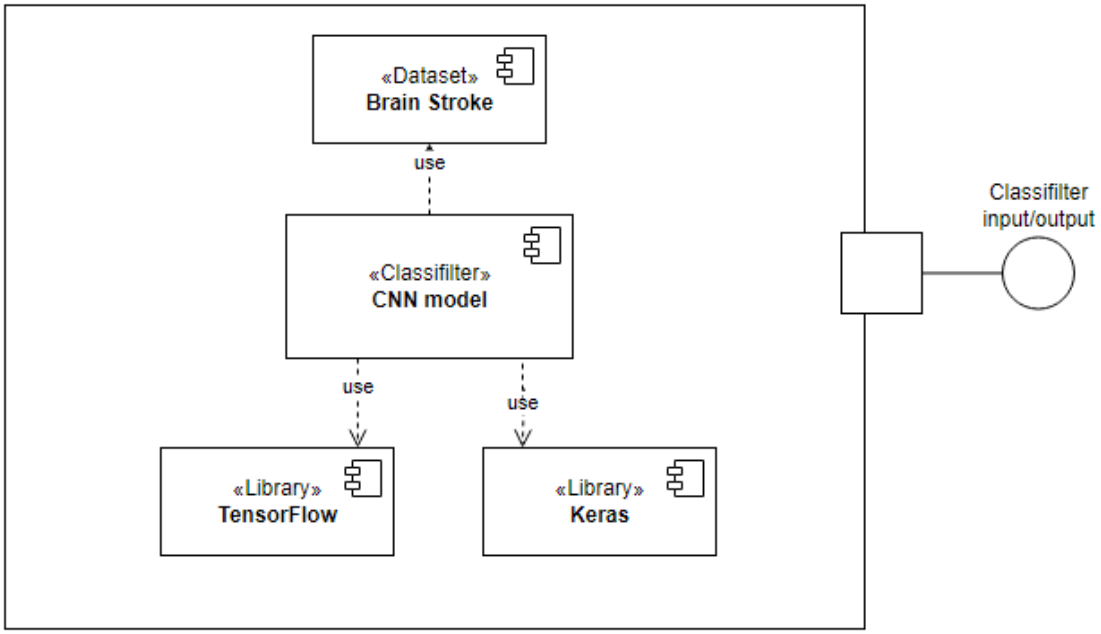


Рисунок 2.3 Діаграма компонентів

2.2 Математичне забезпечення системи

Набір даних повинен бути зібраний з реального виробничого процесу. Оскільки для розробки та перевірки розпізнавача образів потрібна велика кількість шаблонів, а оскільки вони недоступні з економічної точки зору, часто використовуються змодельовані дані.

Датасет «Brain Stroke Image Dataset», який було використано для навчання та аналізу було взято з сервісу kaggle.[14]

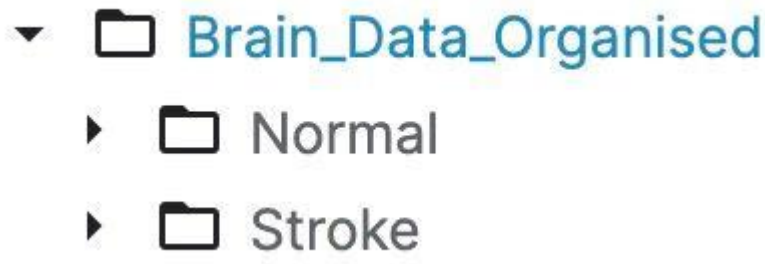


Рисунок 2.4 Структура датасету

В на набір даних входить:

- 1551 КТ зображень нормального стану головного мозку, без уражень та пухлин(Normal)
- 950 КТ зображень ураженого головного мозку інсультом(Stroke)

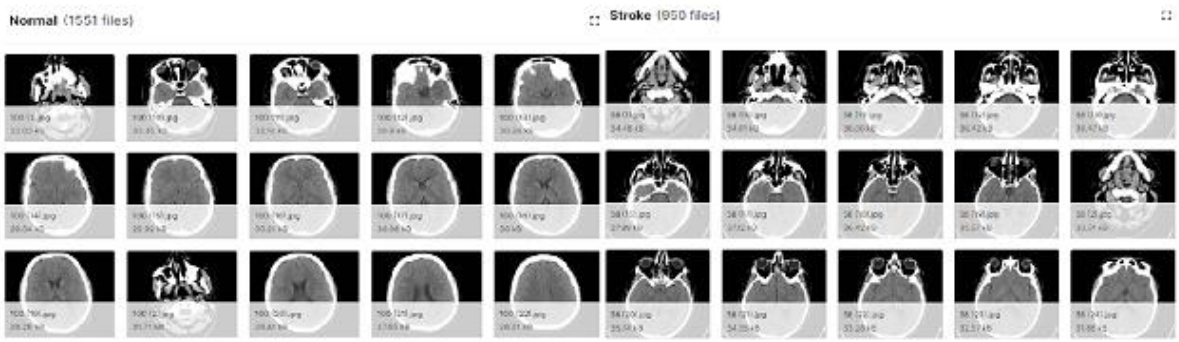


Рисунок 2.5 Вміст папок(stroke, normal)

Вибір статичних ознак, які будуть представлені як вхідний вектор є дуже важливим. Наявність занадто великої кількості вхідних функцій може як обтяжувати навчальний процес і призвести до неефективного розпізнавання, так і покращити результати та точність, використовуючи правильні параметри та метод.

2.2.1 Переробка знімків та пошук крайніх точок у контурах

КТ дозволяє побачити лікарю області органів, але випадковий шум або інші фактори можуть погіршити якість зображення. Для цього потрібно зробити попередню обробку зображень, для створення умов, які будуть покращувати

результати. Існує багато методів обробки, обрати метод дослідження можна за допомогою вивчення задачі, яку потрібно вирішити. Методи можуть включати:

- виділення кращих фрагментів;
- збільшення;
- отримання 3-мірних зображень;
- Кольорокорегуваня;
- реалізація високопрострового розширення;
- покращення якості зображень і т.д.

В даній роботі застосовувався метод пошуку крайніх точок у контурах, як етап попередньої обробки до більш просунутих програм комп'ютерного зору.

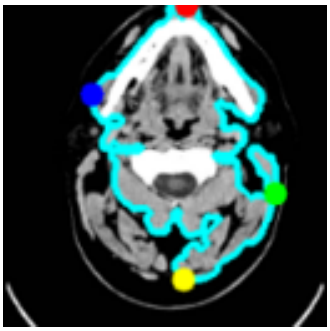


Рисунок 2.4 Знаходження крайніх точок на контурах

На малюнку вище ми обчислили опуклу оболонку контуру головного мозку, а потім знайшли крайні точки вздовж опуклої оболонки.

Знаходження крайніх точок або знаходження точок екстремуму, також називають знаходженням точок максимуму та мінімуму.

Значення функції у точці максимуму називається локальним максимумом, значення функції у точці мінімуму – локальним мінімумом в даній функції.

Точка x_0 називають точкою строгого локального максимуму функції

$y = f(x)$, якщо для всіх x з околиці цієї точки буде справедлива сувора нерівність $f(x) < f(x_0)$.

Точка x_0 називають точкою строгого локального мінімуму функції

$y = f(x)$, якщо для всіх x з околиці цієї точки буде справедлива сувора нерівність $f(x) > f(x_0)$.

Вимірювання розміру об’єктів на зображенні подібне до обчислення відстані від нашої камери до об’єкта — в обох випадках нам потрібно визначити співвідношення, яке вимірює кількість пікселів на дану метрику.

Щоб визначити розмір об’єкта на зображенні, нам спочатку потрібно виконати «calibration» за допомогою еталонного об’єкта. Наш довідковий об'єкт повинен мати дві важливі властивості:

- Потрібно знати розміри (з точки зору ширини чи висоти) у одиниці вимірювання (наприклад, міліметри, дюйми тощо).
- Потрібно знайти цей опорний об’єкт на зображенні або на основі розташування об’єкта .

2.2.2 Згорткові нейронні мережі(CNN)

Згорткові нейронні мережі (CNN), є особливим типом нейронних мереж з прямим зв’язком. Вони успішно використовуються в різних обчислювальних задачах, включаючи розпізнавання зображень, виявлення відео.

Вхідні дані CNN - це багатовимірна матриця (тензор), яка представляє різні типи даних у вигляді зображень, відео чи текстів. Основна математична операція за CNN є лінійною згортковою функцією. У більшості мережевих архітектур за згортковими рівнями слідують шари об’єднання. На рисунку 2.4 показано графічне зображення для повної архітектури мережі. [15-16]

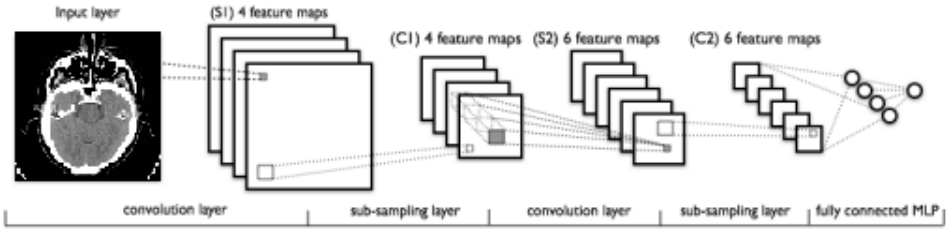


Рисунок 2.5 Архітектура згорткової нейронної мережі[17]

Згорткові шари виконують ключову операцію в CNN. Ця операція включає 3 істотних тензора: вхідний , $x \in R^{m \cdot n}$ фільтр (ядро) $w \in R^{h \cdot l}$, де $h \leq m$ і $l \leq ni$ вихід (карта ознак) $k \in R^{m-k+1 \cdot n-l+1}$. Це простий випадок 2D тензорів, наприклад, вхідним може бути двовимірне зображення де кожен піксель має одиничне значення. Тензори вищого порядку величини також можуть застосовуватися. Наприклад, тривимірні тензори можна використовувати з зображеннями для представлення піксельні значення RGB. У разі введення тексту додаткові розміри можуть бути вкажіть різні уявлення для слів. Фільтри також можуть мати кілька вимірів для створення численних карт об'єктів. Фільтр - це просто тензор ваги. Ідея застосування фільтрів проста (рисунок 2.4). Фільтр $w \in R^{h \cdot l}$, обертається навколо входу $x \in R^{m \cdot n}$ і в кожному місці поелементне множення між w і на вході застосовується вікно розміру $h \times l$. Елементи отриманої матриці підсумовується, щоб утворити нову характеристику ki,j .

Застосування фільтра до різних позицій у вхідних даних створює карту об'єктів k з різними виявленими особливостями. Фільтр зміщує по вертикалі та горизонталі вхідне зображення. Однак те, що він може рухатися лише в одному напрямку є поширеним у завданнях обробки природної мови (NLP). Крім того,це демонструє простий спосіб застосування 2D-фільтра.

Етапи роботи CNN:

1. На початку маємо вхідне зображення.
2. Застосування фільтрів для створення карти об'єктів.
3. Застосування функції ReLU для збільшення нелінійності.

4. Застосування шару об’єднання до кожної карти об’єктів.
5. Вирівнювання об’єднаних зображень в один довгий вектор.
6. Ввод вектору у повністю підключену штучну нейронну мережу.
7. Обрабка функцій через мережу.
8. Тренування шляхом прямого і зворотного поширення протягом багатьох епох. Це повторюється до тих пір, поки ми не отримаємо чітко визначену нейронну мережу з навченими ваговими показниками та детекторами функцій.

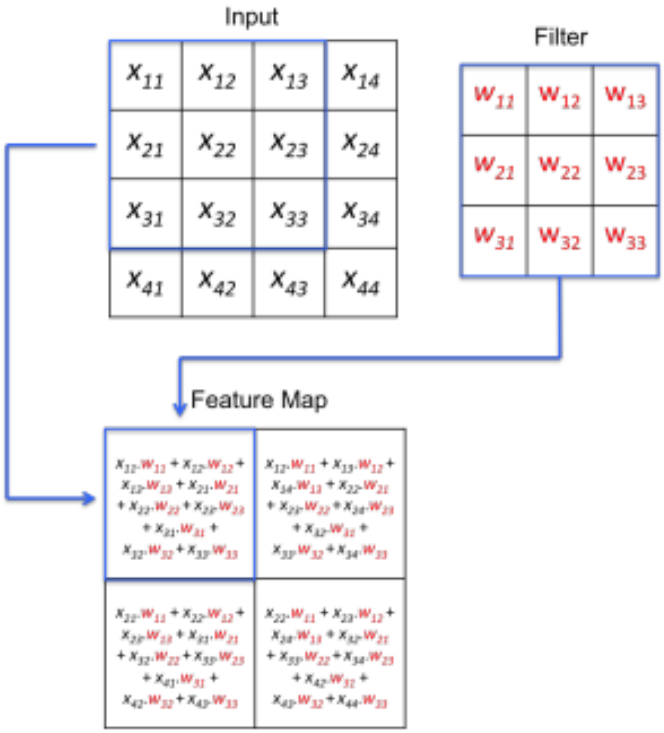


Рисунок 2.6 Згорткова операція[18]

Згортковий фільтр працює як детектор функцій, який витягує локальні ознаки, якы застосовуються до локальних вікон у вводі. Наприклад, при розпізнаванні зображень фільтри можуть виявляти форми або об’єкти в різних місцях зображення. Щоб витягти нелінійні ознаки вищого рівня, лінійний згортковий шар перемежовується нелінійною функцією активації (формула 2.1). Ця функція застосовується поелементно до карти об’єктів, видобуті раніше для

створення локальних нелінійних об’єктів. У цій роботі будемо використовувати функцію активації – ReLU, що визначена формулою:

$$f(s) = max(0, x), \quad 2.1$$

де x – вхідне значення нейрона.

У CNN операція об’єднання традиційно застосовується до карт об’єктів, які витягуються згортковим шаром. Об’єднання – це функції агрегації такі як: max, середнє, середньозважене та норма[18]; найпоширенішими є максимальне та середнє об’єднання [19]. Максимальне об’єднання спроб виділяє важливі функції, вибравши об’єкт(и) з найвищою цінністю, тоді як середнє об’єднання усереднює ці об’єкти. Загалом, об'єднання — це спосіб вилучення глобальних ознак із локальних, виявлених раніше за допомогою згорткової операції. Існують різні переваги об’єднання.

По-перше, виділення особливостей найвищих значень (максимальний пул) або їх усереднення допомагає зберегти важливу інформацію з вхідних даних і зменшити можливий шум. По-друге, об’єднання – це зменшення розмірності (підвибірки) механізу, який призводить до більш ефективних обчислень, особливо в глибоких мережах. Він також створює уніфіковане представлення для всіх екземплярів put зі змінним розміром; отже, рівень прогнозування мережі отримає вхідні дані однакової розмірності. Нарешті, об’єднання досягає перехідної інваріантності. Карта об’єктів вказує на наявність певного об’єкта у вхідних даних. Завдяки зведенню значень на цій карті ця функція виявляється незалежно від дото де він відображається у вводі. Це може бути корисним у деяких завданнях, де необхідна просторова інваріантність, тоді як вона менш ефективна в інших, які є більшими чутливий до порядку функцій.

Після об’єднання буде йти повне об’єднаня. Як обговорювалося раніше, згорткові нейрони локально з’єднані з певними ділянками на вході, що створює розріджену мережу. Це відрізняється від традиційних нейронних

архітектур з прямим зв'язком, де кожний вхідний нейрон підключений до кожного вихідного нейрона .Це повне з'єднання досягається на кінцевому рівні в CNN, де вихід останньої операції об'єднання повністю пов'язаний з рівнем прогнозування. Вихідний рівень може виконувати операції лінійної регресії або softmax залежно від завдання.

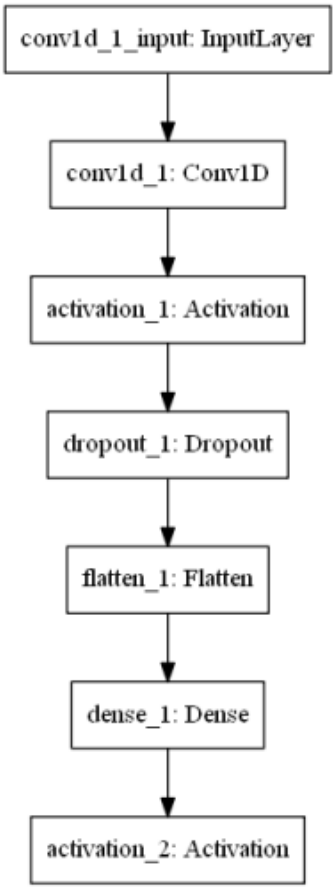


Рисунок 2.7 Блок-схема згорткової нейронної мережі[20]

2.2.3 Компоненти згорткової нейронної мережі

Нижче наведено типові компоненти згорткової нейронної мережі: інтенсивність пікселів зображення буде зберігатися у вхідному шарі. Для каналів червоного, зеленого та синього (RGB) кольорів вхідне зображення шириною 64, висотою 64 і глибиною 3 буде мати розмір входу 64x64x3. Згортковий шар бере зображення з попередніх шарів і згортає їх за допомогою заданої кількості фільтрів для створення зіставлення вихідних об'єктів. Надана кількість фільтрів дорівнює кількості зіставлень вихідних об’єктів. CNNshave

від TensorFlow або PyTorch до цього часу в основному використовували 2D-фільтри, однак нещодавно були додані 3D-фільтри згортки. Функціями активації для CNN зазвичай є ReLU. Після проходження через шари ReLUactivation вихідне вимірювання збігається з вхідним значенням. Рівень ReLU надає мережі нелінійність, а також забезпечує ненасичені градієнти для позитивних мережевих входів. З точки зору висоти та ширини, шар злиття зменшить розмірність карт 2Dactivation. Глибина або кількість карт активації не змінюється і залишається постійною. Традиційні нейрони в повністю пов'язаних шарах отримують різні набори ваг від попередніх шарів; на відміну від процесів згортки, між ними немає розподілу ваги. Через незалежні вагові коефіцієнти кожен нейрон у цьому шарі буде пов'язаний з усіма нейронами в попередньому шарі або з усіма виходами координат у вихідних картах. Вихідні нейрони класу отримують вхідні дані від кінцевих повністю пов'язаних шарів для класифікації.

2.2.4 VGG Net

Концепція моделі VGG16 (також VGGNet-16) така ж, як і VGG19, за винятком того, що вона підтримує 16 шарів. «16» і «19» означають кількість вагових шарів у моделі (згорткові шари). Це означає, що VGG16 має на три згорткові шари менше, ніж VGG19.

Інтуїтивно, чим більше шарів, тим краще. Однак було виявлено, що VGG-16 краще, ніж VGG-19. Якщо порівняти ці конфігурації, VGG-19 (Config E) отримала найнижчий рівень помилок, тоді як ряд параметрів збільшився лише на 3,6%, тому в даній роботі використовувалась модель VGG16.

Таблиця 2.1

Назва моделі	Кількість параметрів	Похибка (%)
VGG13	133	9.6
VGG16	138	7.5
VGG19	144	7.6

Число 16 у назві VGG вказує на те, що це 16 шарів глибокої нейронної мережі (VGGnet). Це означає, що VGG16 є досить розгалуженою мережею, має загалом близько 138 мільйонів параметрів. Навіть за сучасними стандартами це величезна мережа. Однак простота архітектури VGGNet16 робить мережу більш привабливою. Лише поглянувши на архітектуру, можна сказати, що мережа досить однорідна. Існує кілька шарів згортки, за якими слідує шар об'єднання, який зменшує висоту та ширину. Якщо ми подивимося на кількість фільтрів, які ми можемо використовувати, доступно близько 64 фільтрів, які ми можемо подвоїти приблизно до 128, а потім до 256 фільтрів. В останніх шарах ми можемо використовувати 512 фільтрів. Кількість фільтрів, які ми можемо використовувати, подвоюється на кожному кроці або через кожен стек шару згортки. Це основний принцип, який використовується для розробки архітектури мережі VGG16.

Одним із важливих недоліків мережі VGG16 є те, що це величезна мережа, а це означає, що для навчання її параметрів потрібно більше часу. Завдяки глибині та кількості повністю підключених шарів, модель VGG16 становить понад 533 МБ. Це робить впровадження мережі VGG трудомістким завданням.

Модель VGG16 використовується в кількох проблемах класифікації зображень глибокого навчання, але менші архітектури мережі, такі як GoogLeNet і SqueezeNet, часто є кращими. У будь-якому випадку, VGGNet є чудовим будівельним блоком для цілей навчання, оскільки його легко реалізувати. VGG16 значно перевершує попередні версії моделей на змаганнях ILSVRC-2012 і ILSVRC-2013. Більше того, результат VGG16 конкурує за переможця завдання на класифікацію (GoogLeNet з похибкою 6,7%) і значно перевершує подання-переможець ILSVRC-2013 Clarifai. Він отримав 11,2% із зовнішніми навчальними даними і близько 11,7% без них.

З точки зору продуктивності однієї мережі, модель VGGNet-16 досягає найкращого результату з похибкою тесту близько 7,0%, тим самим перевершуючи один GoogLeNet приблизно на 0,9%.

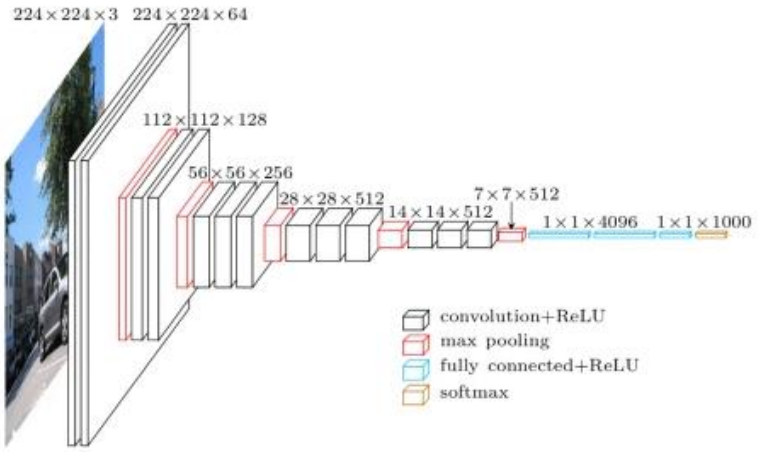


Рисунок 2.8 Архітектура моделі VGG[20]

2.2.5 Inception

Inception модулі використовуються в згорткових нейронних мережах, щоб забезпечити більш ефективні обчислення та глибші мережі за рахунок зменшення розмірності за допомогою згорток 1×1. Модулі були розроблені для вирішення проблеми обчислювальних витрат, а також переобладнання, серед інших питань. Рішення полягає в тому, щоб взяти кілька розмірів фільтрів ядра в CNN і замість того, щоб об’єднувати їх послідовно, наказавши їм працювати на одному рівні.

Inception модулі вбудовуються в згорткові нейронні мережі (CNN) як спосіб зменшити витрати на обчислення. Оскільки нейронна мережа має справу з величезним набором зображень із широкими варіаціями у пропонованих зображеннях, також відомих як помітні частини, їх потрібно спроектувати належним чином. Найбільш спрощена версія початкового модуля працює, виконуючи згортку на вході з не одним, а трьома різними розмірами фільтрів (1x1, 3x3, 5x5). Також виконується максимальний пул. Потім отримані вихідні дані об’єднуються і надсилаються на наступний шар,

структуруючи CNN, щоб виконувати свої згортки на тому самому рівні, мережа стає все ширшою, а не глибшою.

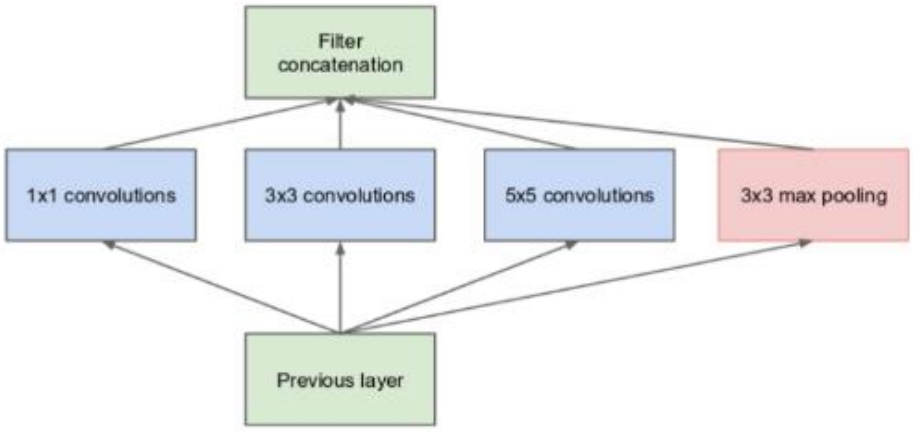


Рисунок 2.9 Inception модуль[20]

Щоб зробити процес кращим, нейронну мережу можна спроектувати так, щоб додати додаткову згортку 1x1 перед шарами 3x3 ad 5x5. Таким чином кількість вхідних каналів обмежена, а згортки 1x1 набагато краще, ніж згортки 5x5. Однак важливо зазначити, що згортка 1x1 додається не після шару максимального об’єднання, а не раніше[20].

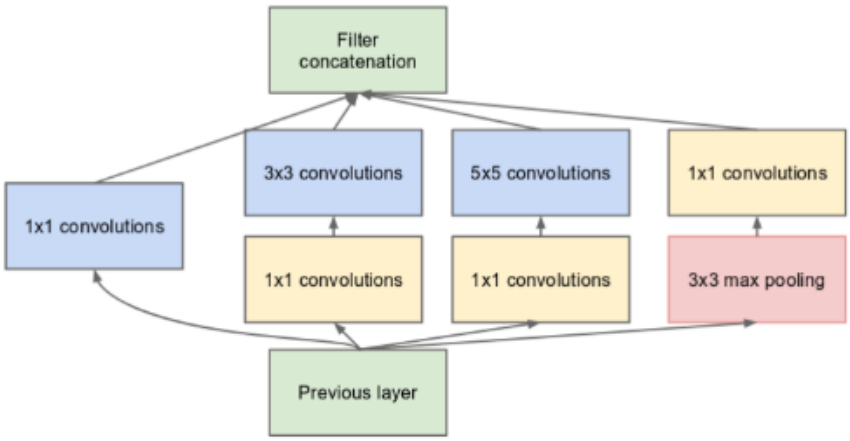


Рисунок 2.10 Inception модуль зі зменшенням розмірів

2.2.6 Xception

Xception — це архітектура глибокої згорткової нейронної мережі, яка включає згортки, які можна розділити по глибині. Його розробили дослідники Google.

Google представив інтерпретацію початкових модулів у згорткових нейронних мережах як проміжного кроку між звичайною згорткою та операцією згортки по глибині (згортка по глибині, за якою слідує точкова згортка). У цьому світлі згортка з роздільною глибиною можна розуміти як початковий модуль з максимально великою кількістю опор.

XCeption — це ефективна архітектура, яка спирається на два основні моменти:

- Глибинороздільна згортка
- Ярлики між блоками Convolution, як у ResNet

XCeption пропонує архітектуру, яка складається з блоків Depthwise Separable Convolution + Maxpooling, які пов'язані з ярликами, як у реалізаціях ResNet.

Специфіка XCeption полягає в тому, що за глибинною згорткою не слідує точкова згортка, але порядок змінюється.

2.3 Програмне забезпечення системи

Розробка системи автоматичної сегментації уражень мозку на КТ зображеннях вимагає наявності інструментів для попередньої обробки медичних файлів, реалізації алгоритму CNN, застосування шаблонів, та роботи з зображеннями і візуалізацією даних. Найголовнішими критеріями для вибору програмної платформи для розроблюваної системи можна виділити наступні:

- наявність бібліотек для роботи з форматами медичних файлів, перегляду,
- маніпуляції та перетворенням даних;
- наявність бібліотек для реалізації алгоритмів машинного навчання
- потужні бібліотеки для візуалізації даних та роботи з різними типами зображень;

Виходячи з вищеперерахованих критеріїв, найбільш зручною мовою програмування з великим набором потужних бібліотек та інструментів є мова програмування Python.

Розглянемо окремо бібліотеки та фреймворки програмної мови Python, які будуть використовуватись для реалізації системи сегментації.

Google Colab було обрано як середу розробки інтелектуальної системи

Плюси:

- Попередньо створений з великою кількістю бібліотеки Python;
- Швидкий початок навчання Python;
- Інфраструктура не потрібна;
- Без плати за використання графічного процесора;
- Може виконувати ваш код протягом 24 годин без перерв;
- Ваші блокноти зберігаються лише на диску Google.

Мінуси:

- Необхідно встановити всі конкретні бібліотеки, які не постачаються зі стандартним Python (це потрібно повторювати з кожним сеансом);
- Google Диск є вашим джерелом сховища, є й інші, наприклад локальні (які споживають вашу пропускну здатність, якщо набір даних великий);
- Google надав код для підключення та використання з Google Drive, але він не працюватиме з великою кількістю інших форматів даних;
- Google Storage використовується з поточним сеансом, тому, якщо ви завантажили файл і хочете використати його пізніше, краще збережіть його перед закриттям сеансу;
- Важко працювати з більшими наборами даних, оскільки їх потрібно завантажувати та зберігати на диску Google (15 ГБ вільного місця з ідентифікатором Gmail, додатково потрібна оплата на користь Google)

2.3.1 Аналіз можливостей бібліотек TensorFlow та Keras

Для розробки штучної нейронної мережі доцільно скористатися вже існуючим програмним пакетом, який дозволить спростити на пришвидшити розробку. Одним з найпопулярніших пакетів для машинного навчання є

TensorFlow. TensorFlow — це відкрита (open source) бібліотека для машинного навчання. Основне призначення — навчання глибинних нейронних мереж. Створена в компанії Google командою Google Brain та випущена для публічного доступу у 2015 році. Tensorflow доступна в чотирьох варіантах:

- розробка на мові Python;

- розробка на мові JavaScript;

- розробка для мобільних платформ (iOS та Android) та вбудованих пристроїв (для IoT — Інтернету речей);

- розширена версія TensorFlow для великих підприємств.

Крім того, TensorFlow можна використовувати за допомогою C API з багатьма мовами програмування: C++, Go, Java, Swift. Враховуючи простоту та велику кількість бібліотек, в якості мови програмування обрана мова Python та відповідна їй версія TensorFlow. Для початку роботи з TensorFlow потрібно встановити пакет за допомогою команди:

```
pip install tensorflow
```

Далі у проекті на мові Python достатньо здійснити імпорт бібліотеки TensorFlow:

```
import tensorflow as tf
```

Після цього бібліотека TensorFlow повністю встановлена та готова до роботи.

Для роботи зі штучними нейронними мережами на мові Python найкращим рішенням буде використання поверх TensorFlow спеціальної бібліотеки Keras. Keras — відкрита бібліотека, що надає інтерфейс мовою Python для роботи зі штучними нейронними мережами. Дана бібліотека служить інтерфейсом між TensorFlow та Python. Перевагами бібліотеки Keras є:

1. вузька спеціалізація саме на глибинних нейронних мережах;

2. тісна інтеграція з TensorFlow;
3. модульність;
4. швидкість налаштування та простота роботи;
5. можливість швидко перевіряти ідеї та використовувати короткі ітерації в процесі розробки;
6. підтримка згорткових та рекурентних нейронних мереж.

Для роботи з Keras достатньо в існуючому проекті на мові Python та при завантаженому TensorFlow додати наступний код:

```
from tensorflow import keras

from tensorflow.keras import layers
```

Для створення згорткової нейронної мережі за допомогою бібліотеки Keras найкраще підходить модель Sequential (послідовна). Як зрозуміло з назви, вона дозволяє створювати послідовний набір шарів. Схематично, модель має наступний вигляд:

```
model = keras.Sequential(

[

layers.Dense(2, activation="relu"),

layers.Dense(3, activation="relu"),

layers.Dense(4),

] )
```

В даному прикладі до моделі додано 3 повнозв’язні шари з двома, трьома та чотирма нейронами відповідно В якості функції активації виступає ReLU.

Також додавати шари до даної моделі можна інкрементально:

```
model = keras.Sequential()
```

```
model.add(layers.Dense(2, activation="relu"))
```

В цьому прикладі спочатку була створена порожня модель, а потім до неї був доданий повнозв’язний шар (в якості функції активації також використана ReLU).

2.3.2 Аналіз можливостей бібліотеки scikit-learn

Scikit-learn – це бібліотека на мові Python, яка надає багато алгоритмів навчання без нагляду та котролю. Вона побудована на основі таких технологій, як NamPY, Pandas і Matplotlib.

Функціональність, яку надає scikit-learn:

- Регресія, включаючи лінійну та логістичну регресію
- Класифікацію, включаючи k-найближчі сусіди
- Кластеризацію, включаючи k-means, k-means++

3 ПРОГРАМНА РОЗРОБКА ТА РЕЗУЛЬТАТИ

3.1 Обґрунтування вибору середовища програмування

Для реалізації програмного коду та нейронної мережі CNN було обрано мову програмування Python так, як ця мова має весь необхідний функціонал та бібліотеки для побудови модуля.

Python - це широко використовувана, інтерпретована, об’єктно-орієнтована й високорівнева мова програмування з динамічною семантикою, яка використовується для програмування загального призначення. Однією з дивовижних особливостей Python є той факт, що це фактично робота однієї людини. Зазвичай нові мови програмування розробляються і публікуються великими компаніями, в яких працює багато професіоналів, і через правила авторського права дуже важко назвати когось із людей, задіяних у проекті. Python є винятком.

Переваги:

- легка та інтуїтивно зрозуміла мова, настільки ж потужна, як і мова основних конкурентів;
- відкритий вихідний код, тож будь-хто може зробити свій внесок у його розвиток;
- код, зрозумілий так само, як і проста англійська;
- підходить для повсякденних завдань, що забезпечує короткий час розробки.

Python широко використовується для впровадження складних інтернет-сервісів, таких як пошукові системи, хмарне сховище та інструменти.

Однією з головних переваг Python - це Keras [21]. Keras - високорівневий API глибокого навчання, розроблений Google для впровадження нейронних мереж. Він написаний на Python і використовується для полегшення реалізації нейронних мереж. Він також підтримує обчислення кількох серверних нейронних мереж.

Keras відносно легко вивчати та працювати з ним, оскільки він забезпечує інтерфейс Python з високим рівнем абстракції, одночасно маючи можливість кількох серверних елементів для цілей обчислень. Це робить Keras повільнішим, ніж інші фреймворки глибокого навчання, але надзвичайно зручним для початківців. Keras дозволяє перемикатися між різними бекендами. Фреймворки, які підтримує Keras:

- Tensorflow

- Теано

- PlaidML

- MXNet

- CNTK (Microsoft Cognitive Toolkit)

З цих п’яти фреймворків TensorFlow прийняв Keras як офіційний високорівневий API. Keras вбудований в TensorFlow і може використовуватися для швидкого глибокого навчання, оскільки він надає вбудовані модулі для всіх обчислень нейронної мережі. У той же час обчислення, що включають тензори, графіки обчислень, сеанси тощо, можуть бути виконані на замовлення за допомогою Tensorflow Core API, що надає вам повну гнучкість і контроль над вашим додатком, а також дозволяє реалізувати свої ідеї за відносно короткий час. Для середи розробки було обрано Google Colab.

Colab - це безкоштовне середовище для ноутбуків Jupyter, що повністю працює в хмарі. Найголовніше, Colab не вимагає налаштування, плюс блокноти, які створюються, можуть одночасно редагуватися декількома членами команди - подібним чином редагуються документи в документах Google. Найбільшою перевагою є те, що Colab підтримує більшість популярних бібліотек машинного навчання, які можна легко завантажити у ваш ноутбук.

Переваги Google Colab:

- Створення/завантаження/надавання доступ до блокнотів;
- Імпорт/збереження блокнотів із/на Google Диск;
- Імпортуваня/публікація блокнотів з GitHub;
- Імпортування зовнішнішніх наборів даних;
- Інтеграція PyTorch, TensorFlow, Keras, OpenCV;
- Безкоштовний хмарний сервіс із безкоштовним графічним процесором

Рисунок 3.1 Емблема Google Colab

3.2 Реалізація Xception

Xception - це архітектура згорткової нейронної мережі повністю заснований на глибинних шарах згортки. Фактично, ми висуваємо гіпотезу: що відображення міжканальних кореляцій і просторових кореляцій в картах ознак згорткових нейронних мереж може бути повністю відокремлені. Оскільки ця гіпотеза є більш сильною версією гіпотези, що лежить в основі архітектури Inception, була винайдена архітектура Xception, яка інсує для «Екстремального початку».

3.2.1 Попередня обробка зображень

Для обробки та створення набору даних було використано ImageDataGenerator, який належить модулю попередньої обробки зображено Keras, а також інтегрував API Keras в Tensorflow 2.0.

ImageDataGenerator використовується для базового процеса машинного навчаня поетапно:

1. Розуміння та перевірка даних.
2. Створення вхідного конвоїру .
3. Моделювання.
4. Модель навчання.
5. Тестова модель.

Основні етапи попередньої обробки даних xception:

- Імпорт необхідних пакетів;
- Завантаження зображень;
- Розділення набору даних на навчальну та тестову;
- Визначення параметрів для полегшення начання.

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/x
83689472/83683744 [==============================] - 1s 0us/step
83697664/83683744 [==============================] - 1s 0us/step
Model: "sequential"
```

Рисунок 3.2 Завантаження даних

3.2.2 Проведення експериментів і аналіз отриманих результатів xception

Перед навчанням було зроблено налаштування параметрів навчання. Отже, навчання проводиться за наступних умов:

- кількість епох: 20;
- batch size (кількість навчальних прикладів за одну ітерацію) для

навчальної вибірки: 16;

- learning rate = 0.00001
- навчальна та тестова вибірка 90/10

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 xception (Functional)       (None, 10, 10, 2048)      20861480

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 100)               204900

 batch_normalization_4 (Batc  (None, 100)              400
 hNormalization)

 dropout (Dropout)           (None, 100)               0

 dense_1 (Dense)             (None, 50)                5050

 batch_normalization_5 (Batc  (None, 50)               200
 hNormalization)

 dense_2 (Dense)             (None, 2)                 102

=================================================================
Total params: 21,072,132
Trainable params: 21,017,304
Non-trainable params: 54,828
_________________________________________________________________
```

Рисунок 3.3 Побудова мережі xception

```
141/141 [==============================] - 938s 6s/step - loss: 0.9594 - accuracy: 0.5322 - val_loss: 0.6933 - val_accuracy: 0.5440
Epoch 2/20
141/141 [==============================] - 191s 1s/step - loss: 0.8380 - accuracy: 0.5837 - val_loss: 0.7280 - val_accuracy: 0.5280
Epoch 3/20
141/141 [==============================] - 191s 1s/step - loss: 0.7648 - accuracy: 0.6273 - val_loss: 0.7632 - val_accuracy: 0.5080
Epoch 4/20
141/141 [==============================] - 191s 1s/step - loss: 0.7318 - accuracy: 0.6362 - val_loss: 0.7612 - val_accuracy: 0.5880
Epoch 5/20
141/141 [==============================] - 191s 1s/step - loss: 0.6718 - accuracy: 0.6744 - val_loss: 0.7101 - val_accuracy: 0.6880
Epoch 6/20
141/141 [==============================] - 191s 1s/step - loss: 0.6326 - accuracy: 0.6908 - val_loss: 0.6339 - val_accuracy: 0.7040
Epoch 7/20
141/141 [==============================] - 191s 1s/step - loss: 0.5934 - accuracy: 0.7143 - val_loss: 0.6298 - val_accuracy: 0.7000
Epoch 8/20
Epoch 9/20
141/141 [==============================] - 191s 1s/step - loss: 0.5228 - accuracy: 0.7521 - val_loss: 0.6159 - val_accuracy: 0.7160
Epoch 10/20
141/141 [==============================] - 191s 1s/step - loss: 0.4965 - accuracy: 0.7681 - val_loss: 0.6620 - val_accuracy: 0.7120
Epoch 11/20
141/141 [==============================] - 192s 1s/step - loss: 0.4695 - accuracy: 0.7797 - val_loss: 0.6682 - val_accuracy: 0.7080
Epoch 12/20
141/141 [==============================] - 191s 1s/step - loss: 0.4095 - accuracy: 0.8108 - val_loss: 0.6629 - val_accuracy: 0.7480
Epoch 13/20
141/141 [==============================] - 192s 1s/step - loss: 0.3792 - accuracy: 0.8361 - val_loss: 0.6716 - val_accuracy: 0.7320
Epoch 14/20
141/141 [==============================] - 191s 1s/step - loss: 0.3522 - accuracy: 0.8427 - val_loss: 0.6466 - val_accuracy: 0.7480
Epoch 15/20
141/141 [==============================] - 191s 1s/step - loss: 0.3389 - accuracy: 0.8512 - val_loss: 0.6472 - val_accuracy: 0.7280
Epoch 16/20
141/141 [==============================] - 191s 1s/step - loss: 0.3193 - accuracy: 0.8601 - val_loss: 0.6161 - val_accuracy: 0.7400
Epoch 17/20
141/141 [==============================] - 191s 1s/step - loss: 0.3035 - accuracy: 0.8778 - val_loss: 0.6878 - val_accuracy: 0.7520
Epoch 18/20
141/141 [==============================] - 193s 1s/step - loss: 0.2836 - accuracy: 0.8761 - val_loss: 0.7396 - val_accuracy: 0.7520
Epoch 19/20
141/141 [==============================] - 191s 1s/step - loss: 0.2547 - accuracy: 0.8925 - val_loss: 0.7149 - val_accuracy: 0.7200
Epoch 20/20
141/141 [==============================] - 192s 1s/step - loss: 0.2401 - accuracy: 0.9018 - val_loss: 0.6962 - val_accuracy: 0.7240
```

Рисунок 3.4 Навчання мережі епохах

Виходячи з результатів можна побачити, що на 7-й та 8-й епохах втрати були найнижчі, а точність була порівняно однакова та найкраща, але не такая к подтрібно, тому параметри було змінено.

7 епоха:

- val_accuracy = 0,70;
- val_loss = 0,62.

8 епоха:

- val_accuracy = 0,72;
- val_loss = 0,61.

Далі параметри було змінено:

- Learning_rate = 0.0001
- 30 епох;

```
Epoch 3/40
141/141 [==============================] - 85s 603ms/step - loss: 0.3777 - accuracy: 0.8276 - val_loss: 0.6957 - val_accuracy: 0.6560
Epoch 4/40
141/141 [==============================] - 86s 609ms/step - loss: 0.2621 - accuracy: 0.8960 - val_loss: 0.7219 - val_accuracy: 0.7440
Epoch 5/40
141/141 [==============================] - 86s 609ms/step - loss: 0.2349 - accuracy: 0.9138 - val_loss: 0.6142 - val_accuracy: 0.8360
Epoch 6/40
141/141 [==============================] - 89s 626ms/step - loss: 0.1895 - accuracy: 0.9258 - val_loss: 0.8707 - val_accuracy: 0.7920
Epoch 7/40
141/141 [==============================] - 88s 619ms/step - loss: 0.1336 - accuracy: 0.9494 - val_loss: 0.7783 - val_accuracy: 0.7560
Epoch 8/40
141/141 [==============================] - 85s 603ms/step - loss: 0.1312 - accuracy: 0.9556 - val_loss: 0.9448 - val_accuracy: 0.7560
Epoch 9/40
141/141 [==============================] - 86s 608ms/step - loss: 0.1027 - accuracy: 0.9622 - val_loss: 0.9574 - val_accuracy: 0.7600
Epoch 10/40
141/141 [==============================] - 87s 615ms/step - loss: 0.0934 - accuracy: 0.9689 - val_loss: 0.9968 - val_accuracy: 0.7600
Epoch 11/40
141/141 [==============================] - 87s 617ms/step - loss: 0.0938 - accuracy: 0.9685 - val_loss: 0.7290 - val_accuracy: 0.7800
Epoch 12/40
141/141 [==============================] - 87s 613ms/step - loss: 0.1068 - accuracy: 0.9631 - val_loss: 1.0982 - val_accuracy: 0.7440
Epoch 13/40
141/141 [==============================] - 89s 626ms/step - loss: 0.0715 - accuracy: 0.9760 - val_loss: 0.8244 - val_accuracy: 0.8200
Epoch 14/40
141/141 [==============================] - 91s 642ms/step - loss: 0.0524 - accuracy: 0.9867 - val_loss: 1.0961 - val_accuracy: 0.7720
Epoch 15/40
141/141 [==============================] - 89s 632ms/step - loss: 0.0947 - accuracy: 0.9671 - val_loss: 1.0614 - val_accuracy: 0.7800
Epoch 16/40
141/141 [==============================] - 90s 633ms/step - loss: 0.0720 - accuracy: 0.9787 - val_loss: 0.9150 - val_accuracy: 0.8040
Epoch 17/40
141/141 [==============================] - 89s 631ms/step - loss: 0.0828 - accuracy: 0.9720 - val_loss: 0.8861 - val_accuracy: 0.7520
Epoch 18/40
141/141 [==============================] - 89s 631ms/step - loss: 0.0742 - accuracy: 0.9738 - val_loss: 0.8908 - val_accuracy: 0.8120
Epoch 19/40
141/141 [==============================] - 89s 629ms/step - loss: 0.0521 - accuracy: 0.9858 - val_loss: 1.0184 - val_accuracy: 0.7680
Epoch 20/40
```

Рисунок 3.5 Навчання мережі

Видно, як val_loss спочатку падає, а потім починає рости. Отже, модель починає перенавчатися. Тому, було взяту ту модель, у якої val_loss мінімальна. Найкраща виявилась модель на епосі 5:

- val_accuracy = 0,83.
- val_loss = 0,61

3.3 Реалізація VGG net

Модель VGG16 – це pre-trained (попередньо навчена) згорткова нейронна мережа. Для розпізнавання ішемічного інсульту було прийнято рішення перенавчати лише останні чотири шари, цього достатньо для досягнення високої точності розпізнавання. Також на рисунку 3.6 видно, що було додано два нових шари, останній з яких визначає кількість вихідних класів.

3.3.1 Визначення критеріїв порівняння моделей

Серед усієї множини можливих критеріїв, за якими можна порівняти моделі нейронних мереж, слід обрати такі, які є найбільш значущими з точки зору розробника та користувача.

Критерії:

- точність для навчальної вибірки, тобто відсоток правильних відповідей алгоритму (accuracy);
- точність для тестової вибірки (accuracy);
- функція втрат для навчальної вибірки - вимірює точність моделі під час навчання.

Мета: мінімізувати цю функцію щоб "направити" модель в правильному напрямку (loss function);

- функція втрат для тестової вибірки;
- час, витрачений на навчання (епоха/сек);
- розмір вхідного зображення;

3.3.2 Програмний модуль для попередньої обробки зображень

Зараз усі зображення знаходяться в одній папці з підпапками "так" і "ні".Для цього було розділено дані на папки train, val і test, що полегшить роботу.
Нова ієрархія папок матиме такий вигляд:

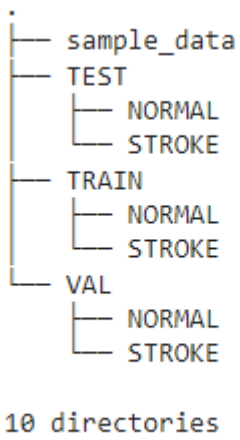


Рисунок 3.6 Ієрархія папок датасету

Датасет було розділено на дві папки з яких 950 зображень головного мозку з інсультом та 1551 здорового.

Одним з головних процесів є імпортування. Для точного аналізу знімків розмір їх було змінено при імпортуванні за допоиогою бібліотеки Nampy та команди np.arrays. Створюємо діаграму сітки для потрібної кількості зображень (n) із зазначеного набору.

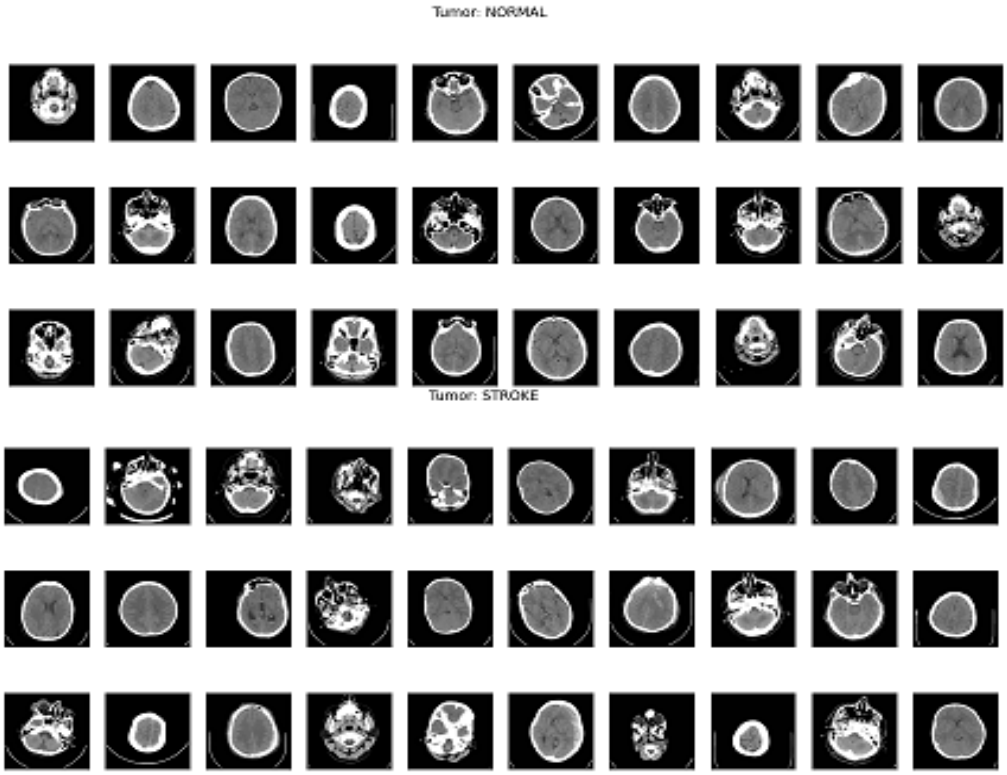


Рисунок 3.7 Сітка зображень

Як бачимо, зображення мають різну «ширину» і «висоту», а також різний розмір «чорних кутів». Оскільки розмір зображення для шару введення VGG-16 дорівнює "(224,224)", деякі широкі зображення можуть виглядати дивно після зміни розміру. Гістограма розподілу співвідношення ("співвідношення = ширина/висота").

Першим кроком «нормалізації» є вирізання мозку із зображень – знаходження точок екстремуму .

Знаходження точок екстремуму використовується як етап попередньої обробки до більш просунутих програм комп’ютерного зору.

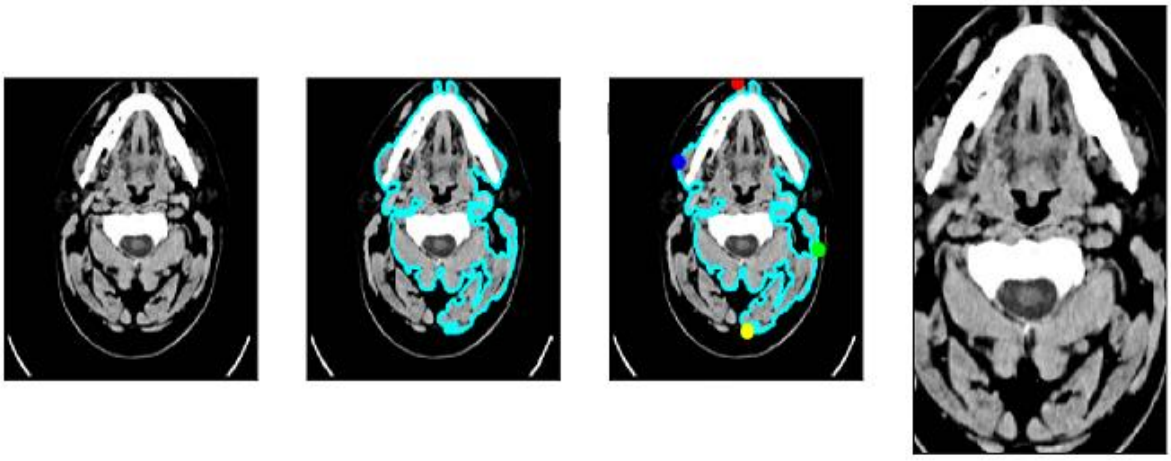


Рисунок 3.8 Знаходження точок екстремуму

Підготовка зображення, вирізання головного мозку відбулося в 4 етапи та було зроблено для кожного набору:

1. Отримання оригінального зображення.
2. Знаходження найбільшого контуру.
3. Знаходження точок екстремуму.
4. Обрізання зображення по точкам.

Экстремум - це будь-яка точка, в якій значення функції є найбільшим (максимум) або найменшим (мінімумом). Існують як абсолютні, так і відносні

(або локальні) максимуми і мінімуми. При відносному максимумі значення функції більше її значення в безпосередньо сусідніх точках, тоді як при абсолютному максимумі значення функції більше, ніж її значення в будь-якій іншій точці інтервалу, що цікавить. При відносних максимумах всередині інтервалу, якщо функція є гладкою, а не піковою, її швидкість зміни або похідна дорівнює нулю.

Теорія екстремумів застосовується до практичних задач оптимізації, таких як знаходження розмірів для контейнера, який уміщатиме максимальний об’єм для заданої кількості матеріалу, використаного при його конструкції. Розташування крайніх точок також допомагає в побудові графіків функцій.

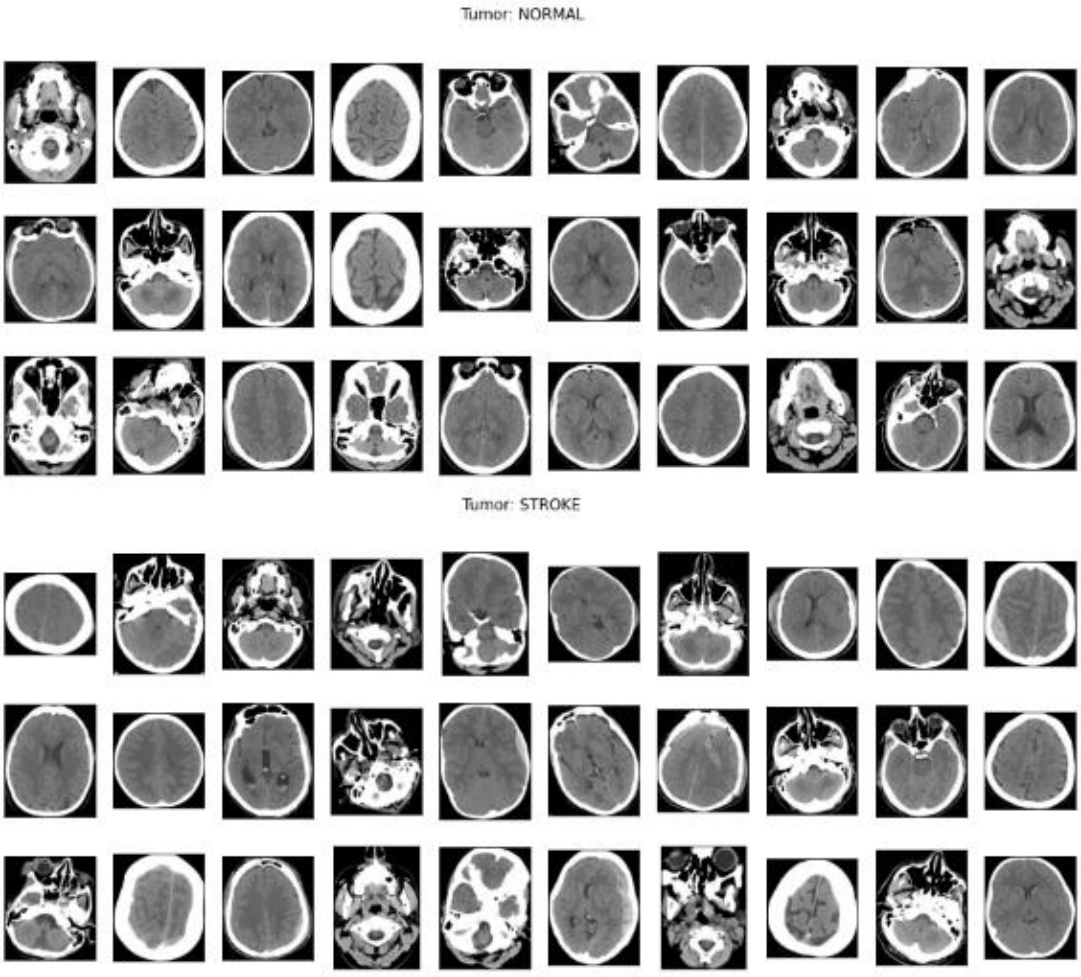


Рисунок 3.9 Результат знаходження точок екстремуму в наборы даних

Після того, як всі зображення оброблені, вони зберігаються для подальшого вікористання в модулі.

Наступним кроком буде зміна розміру зображень до (224x224) і застосування попередньої обробки, необхідної для введення моделі VGG-16.

3.3.3 Проведення експериментів і аналіз отриманих результатів VGG net

Для кожної моделі нейронної мережі зображення підлягають попередній обробці – нормалізації. Також була застосована методика аугментації даних. Під аугментаційними даними розуміється збільшення вибірки даних для навчання через модифікацію реальних даних.

Отже, кожна з наведених моделей має свої особливості, що, відповідно, стає причиною різних кінцевих результатів для даного навчального набору даних.

На першому етапі тестову та навчальну вибірку поділили на 80/20, з яких 80% навчальна та 20% тестова. Якщо навчальна вибірка включає всі об'єкти генеральної сукупності, тобто вони збігаються, то достовірність висновків буде найбільш високою (за всіх інших рівних умов). Якщо ж навчальна вибірка дуже мала, то навряд чи на її основі можуть бути зроблені достовірні висновки про генеральну сукупність, оскільки в цьому випадку в навчальну вибірку можуть навіть не входити приклади об'єктів всіх або переважної більшості класів.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 xception (Functional)       (None, 10, 10, 2048)      20861480

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 100)               204900

 batch_normalization_4 (Batc  (None, 100)              400
 hNormalization)

 dropout (Dropout)           (None, 100)               0

 dense_1 (Dense)             (None, 50)                5050

 batch_normalization_5 (Batc  (None, 50)               200
 hNormalization)

 dense_2 (Dense)             (None, 2)                 102

=================================================================
Total params: 21,072,132
Trainable params: 210,352
Non-trainable params: 20,861,780
_________________________________________________________________
```

Рисунок 3.10 Побудова мережі VGG net

Перед навчанням моделей слід звернути увагу на налаштування параметрів навчання. Отже, навчання проводиться за наступних умов:

- кількість епох: 20;

- batch size (кількість навчальних прикладів за одну ітерацію) для

навчальної вибірки: 128;

Збереження моделі відбувається після кожної епохи. Тому епоха, що має найбільшу точність, вважається найкращою. Кінцева модель зберігається у форматі .h5.

Всього епох 20, але з представленої моделі мережі видно, що навчання відбулось тільки для перших шарів, за допомогою використання умови ранньої зупинки. Так, як точність не покращувалась, робота зупинилась.

В першому тесті кількість картинок розділили на під частини, тому що аналізувати одразу 2000 зображень складно.

```
73s 5s/step - loss: 0.6549 - prc: 0.6391 - val_loss: 0.6422 - val_prc: 0.6645
74s 5s/step - loss: 0.6398 - prc: 0.6682 - val_loss: 0.6459 - val_prc: 0.6616
73s 5s/step - loss: 0.6555 - prc: 0.6429 - val_loss: 0.6471 - val_prc: 0.6517
73s 5s/step - loss: 0.6566 - prc: 0.6420 - val_loss: 0.6403 - val_prc: 0.6726
73s 5s/step - loss: 0.6611 - prc: 0.6306 - val_loss: 0.6303 - val_prc: 0.6877
73s 5s/step - loss: 0.6382 - prc: 0.6683 - val_loss: 0.6245 - val_prc: 0.7133
73s 5s/step - loss: 0.6399 - prc: 0.6706 - val_loss: 0.6243 - val_prc: 0.6995
76s 5s/step - loss: 0.6502 - prc: 0.6551 - val_loss: 0.6423 - val_prc: 0.6922
74s 5s/step - loss: 0.6422 - prc: 0.6660 - val_loss: 0.6285 - val_prc: 0.6824
74s 5s/step - loss: 0.6397 - prc: 0.6742 - val_loss: 0.6237 - val_prc: 0.6826
73s 5s/step - loss: 0.6296 - prc: 0.6839 - val_loss: 0.6297 - val_prc: 0.7040
73s 4s/step - loss: 0.6321 - prc: 0.6804 - val_loss: 0.6443 - val_prc: 0.6762
```

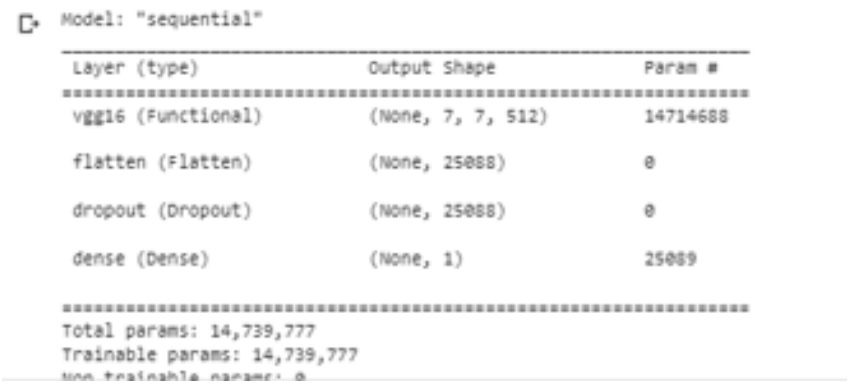
Рисунок 3.11 Процес навчання мережі по епохах

Виходячи з результатів можна побачити, що на 6-тій епосі, точність була порівняно найкраща.

6 епоха:

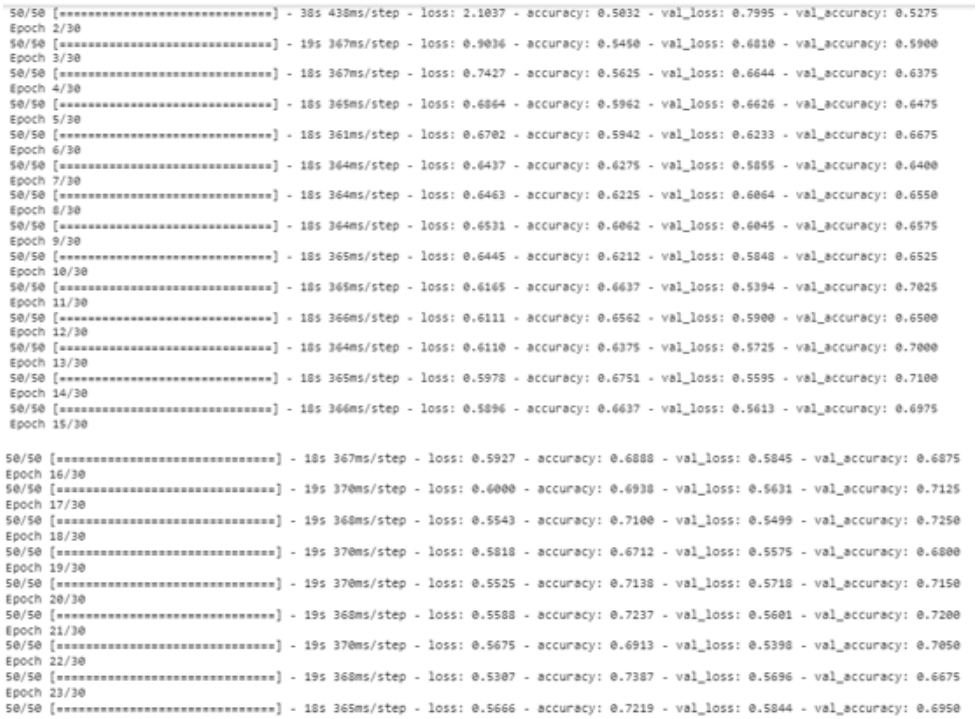
- val_accuracy = 0,71;
- val_loss = 0,62.

В першому експерименті рекурсивне налаштування атрибута trainable = False. Якщо встановити trainable = False на моделі або на будь-якому шарі, який має підшари, усі дочірні шари також не підлягають навчанню. Тому атрибут було змінено на trainable = True, для того активізувати всі шари і дозволити моделі навчатись.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dropout (Dropout)           (None, 25088)             0

 dense (Dense)               (None, 1)                 25089

=================================================================
Total params: 14,739,777
Trainable params: 14,739,777
Non-trainable params: 0
```

Рисунок 3.12 Побудова мережі VGG net



```
50/50 [==============================] - 38s 438ms/step - loss: 2.1037 - accuracy: 0.5032 - val_loss: 0.7995 - val_accuracy: 0.5275
Epoch 2/30
50/50 [==============================] - 19s 367ms/step - loss: 0.9036 - accuracy: 0.5450 - val_loss: 0.6810 - val_accuracy: 0.5900
Epoch 3/30
50/50 [==============================] - 18s 367ms/step - loss: 0.7427 - accuracy: 0.5625 - val_loss: 0.6644 - val_accuracy: 0.6375
Epoch 4/30
50/50 [==============================] - 18s 365ms/step - loss: 0.6864 - accuracy: 0.5962 - val_loss: 0.6626 - val_accuracy: 0.6475
Epoch 5/30
50/50 [==============================] - 18s 361ms/step - loss: 0.6702 - accuracy: 0.5942 - val_loss: 0.6233 - val_accuracy: 0.6675
Epoch 6/30
50/50 [==============================] - 18s 364ms/step - loss: 0.6437 - accuracy: 0.6275 - val_loss: 0.5855 - val_accuracy: 0.6400
Epoch 7/30
50/50 [==============================] - 18s 364ms/step - loss: 0.6463 - accuracy: 0.6225 - val_loss: 0.6064 - val_accuracy: 0.6550
Epoch 8/30
50/50 [==============================] - 18s 364ms/step - loss: 0.6531 - accuracy: 0.6062 - val_loss: 0.6045 - val_accuracy: 0.6575
Epoch 9/30
50/50 [==============================] - 18s 365ms/step - loss: 0.6445 - accuracy: 0.6212 - val_loss: 0.5848 - val_accuracy: 0.6525
Epoch 10/30
50/50 [==============================] - 18s 365ms/step - loss: 0.6165 - accuracy: 0.6637 - val_loss: 0.5394 - val_accuracy: 0.7025
Epoch 11/30
50/50 [==============================] - 18s 366ms/step - loss: 0.6111 - accuracy: 0.6562 - val_loss: 0.5900 - val_accuracy: 0.6500
Epoch 12/30
50/50 [==============================] - 18s 364ms/step - loss: 0.6110 - accuracy: 0.6375 - val_loss: 0.5725 - val_accuracy: 0.7000
Epoch 13/30
50/50 [==============================] - 18s 365ms/step - loss: 0.5978 - accuracy: 0.6751 - val_loss: 0.5595 - val_accuracy: 0.7100
Epoch 14/30
50/50 [==============================] - 18s 366ms/step - loss: 0.5896 - accuracy: 0.6637 - val_loss: 0.5613 - val_accuracy: 0.6975
Epoch 15/30

50/50 [==============================] - 18s 367ms/step - loss: 0.5927 - accuracy: 0.6888 - val_loss: 0.5845 - val_accuracy: 0.6875
Epoch 16/30
50/50 [==============================] - 19s 370ms/step - loss: 0.6000 - accuracy: 0.6938 - val_loss: 0.5631 - val_accuracy: 0.7125
Epoch 17/30
50/50 [==============================] - 19s 368ms/step - loss: 0.5543 - accuracy: 0.7100 - val_loss: 0.5499 - val_accuracy: 0.7250
Epoch 18/30
50/50 [==============================] - 19s 370ms/step - loss: 0.5818 - accuracy: 0.6712 - val_loss: 0.5575 - val_accuracy: 0.6800
Epoch 19/30
50/50 [==============================] - 19s 370ms/step - loss: 0.5525 - accuracy: 0.7138 - val_loss: 0.5718 - val_accuracy: 0.7150
Epoch 20/30
50/50 [==============================] - 19s 368ms/step - loss: 0.5588 - accuracy: 0.7237 - val_loss: 0.5601 - val_accuracy: 0.7200
Epoch 21/30
50/50 [==============================] - 19s 370ms/step - loss: 0.5675 - accuracy: 0.6913 - val_loss: 0.5398 - val_accuracy: 0.7050
Epoch 22/30
50/50 [==============================] - 19s 368ms/step - loss: 0.5307 - accuracy: 0.7387 - val_loss: 0.5696 - val_accuracy: 0.6675
Epoch 23/30
50/50 [==============================] - 18s 365ms/step - loss: 0.5666 - accuracy: 0.7219 - val_loss: 0.5844 - val_accuracy: 0.6950
```

Рисунок 3.13 Навчання мережі за епохами

Видно, що val_loss залишається однаковою, але точність трішки більше– модель навчається Було автоматично змережено модель з мінімальною val_loss та обрано найкращу, 20 епоху.

20 епоха:

- val_accuracy = 0,72;
- val_loss = 0,56.

3.4 Створеня кастомної моделі

У цій роботі розглянуто та розроблено згорткову нейронну мережу (CNN) для класифікації зображень, не покладаючись на попередньо навчені моделі. Існує ряд популярних попередньо навчених моделей (такі як розглянуто вище), які допомагають подолати недоліки вибірки; вони вже пройшли навчання щодо багатьох зображень і можуть розпізнавати різноманітні особливості.

Ці моделі зазвичай мають складну архітектуру, яка необхідна для розшифровки різниці між сотнями або тисячами класів. Складність, яка пропонує передбачувані можливості для різноманітних об’єктів, може стати перешкодою для більш спрощених завдань, оскільки попередньо навчена модель може переповнювати дані. Аналізуючи відкриті джерела, було вирішено спробувати створити свою власну модель.

3.4.1 Програмний модуль для попередньої обробки зображень кастомної моделі

Як попередню обробку даних було використано метод збільшення даних(Data augmentation). Збільшення даних в аналізі даних – це методи, які використовуються для збільшення обсягу даних шляхом додавання дещо змінених копій уже існуючих даних або знову створених синтетичних даних із наявних даних.Він діє як регулятор під час навчання моделі машинного навчання.

У папці "Stroke" було 950 зображень спочатку. Для того, щоб навчити модель правильно було вирішено поділити зображення рівномірно шляхом збільшення даних, для цього було створено цикл за наступними кроками:

- рандомно вибирано картинку із цієї папки;
- сгенеровано кут повороту (від 0 до 20 градусів);
- повернуто рандомно взяту картинку на рандомне число градусів;
- записано нову картинку в ту саму папку;
- поки кількість картинок не зрівнялася в обох папках, дії повторяються.

3.4.2 Проведення експериментів і аналіз отриманих результатів кастомної моделі

Виводимо, як приклад, на екран одне зображення із набору даних, щоб перевірити, чи коректно зчиталися зображення.

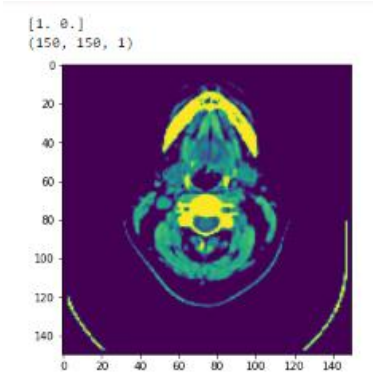


Рисунок 3.14 Зчитування зображення

Задаємо наступні параметри для першого навчання мережі:

- batch_size = 32 size of batch (2^n)
- epochs = 20

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param#
=================================================================
conv2d_5 (Conv2D)            (None, 150, 150, 32)      320

batch_normalization_5 (Batc  (None, 150, 150, 32)     128
hNormalization)

max_pooling2d_5 (MaxPooling  (None, 75, 75, 32)       0
2D)

conv2d_6 (Conv2D)            (None, 75, 75, 64)        18496

dropout_4 (Dropout)          (None, 75, 75, 64)        0

batch_normalization_6 (Batc  (None, 75, 75, 64)       256
hNormalization)

max_pooling2d_6 (MaxPooling  (None, 38, 38, 64)       0
2D)

conv2d_7 (Conv2D)            (None, 38, 38, 64)        36928

batch_normalization_7 (Batc  (None, 38, 38, 64)       256
hNormalization)

max_pooling2d_7 (MaxPooling  (None, 19, 19, 64)       0
2D)

conv2d_8 (Conv2D)            (None, 19, 19, 128)       73856

dropout_5 (Dropout)          (None, 19, 19, 128)       0

batch_normalization_8 (Batc  (None, 19, 19, 128)      512
hNormalization)
max_pooling2d_8 (MaxPooling  (None, 10, 10, 128)      0
2D)

conv2d_9 (Conv2D)            (None, 10, 10, 256)       295168

dropout_6 (Dropout)          (None, 10, 10, 256)       0

batch_normalization_9 (Batc  (None, 10, 10, 256)      1024
hNormalization)

max_pooling2d_9 (MaxPooling  (None, 5, 5, 256)        0
2D)

flatten_1 (Flatten)          (None, 6400)              0

dense_2 (Dense)              (None, 128)               819328

dropout_7 (Dropout)          (None, 128)               0

dense_3 (Dense)              (None, 2)                 258

=================================================================
Total params: 1,246,530
Trainable params: 1,245,442
Non-trainable params: 1,088
```

Рисунок 3.15 Побудова мережі

```
Epoch 1/20
78/78 [==============================] - 13s 146ms/step - loss: 1.5571 - accuracy: 0.5669 - val_loss: 3.2113 - val_accuracy: 0.5000
Epoch 2/20
78/78 [==============================] - 10s 133ms/step - loss: 0.6299 - accuracy: 0.6612 - val_loss: 0.7135 - val_accuracy: 0.5000
Epoch 3/20
78/78 [==============================] - 10s 133ms/step - loss: 0.4770 - accuracy: 0.7816 - val_loss: 3.2522 - val_accuracy: 0.5000
Epoch 4/20
78/78 [==============================] - 10s 133ms/step - loss: 0.3315 - accuracy: 0.8751 - val_loss: 4.9754 - val_accuracy: 0.5000
Epoch 5/20
78/78 [==============================] - 10s 133ms/step - loss: 0.1853 - accuracy: 0.9363 - val_loss: 2.7676 - val_accuracy: 0.5000
Epoch 6/20
78/78 [==============================] - 10s 134ms/step - loss: 0.1415 - accuracy: 0.9517 - val_loss: 3.6935 - val_accuracy: 0.5000
Epoch 7/20
78/78 [==============================] - 10s 134ms/step - loss: 0.1158 - accuracy: 0.9662 - val_loss: 3.4371 - val_accuracy: 0.6290
Epoch 8/20
78/78 [==============================] - 10s 134ms/step - loss: 0.0999 - accuracy: 0.9766 - val_loss: 1.4909 - val_accuracy: 0.7403
Epoch 9/20
78/78 [==============================] - 11s 135ms/step - loss: 0.0633 - accuracy: 0.9815 - val_loss: 1.6646 - val_accuracy: 0.6984
Epoch 10/20
78/78 [==============================] - 10s 134ms/step - loss: 0.0839 - accuracy: 0.9819 - val_loss: 2.2209 - val_accuracy: 0.7306
Epoch 11/20
78/78 [==============================] - 10s 134ms/step - loss: 0.0618 - accuracy: 0.9839 - val_loss: 3.4038 - val_accuracy: 0.7016
Epoch 12/20
78/78 [==============================] - 10s 134ms/step - loss: 0.0714 - accuracy: 0.9851 - val_loss: 1.9131 - val_accuracy: 0.7210
Epoch 13/20
78/78 [==============================] - 10s 133ms/step - loss: 0.0527 - accuracy: 0.9867 - val_loss: 3.2456 - val_accuracy: 0.7226
Epoch 14/20
78/78 [==============================] - 10s 134ms/step - loss: 0.0560 - accuracy: 0.9863 - val_loss: 2.2241 - val_accuracy: 0.8097
Epoch 15/20
78/78 [==============================] - 11s 137ms/step - loss: 0.0611 - accuracy: 0.9883 - val_loss: 3.7730 - val_accuracy: 0.7306
Epoch 16/20
78/78 [==============================] - 10s 134ms/step - loss: 0.0428 - accuracy: 0.9911 - val_loss: 5.0969 - val_accuracy: 0.6887
Epoch 17/20
78/78 [==============================] - 10s 132ms/step - loss: 0.0403 - accuracy: 0.9911 - val_loss: 6.9570 - val_accuracy: 0.7323
Epoch 18/20
78/78 [==============================] - 10s 133ms/step - loss: 0.0797 - accuracy: 0.9871 - val_loss: 4.3465 - val_accuracy: 0.7581
Epoch 19/20
78/78 [==============================] - 10s 134ms/step - loss: 0.0327 - accuracy: 0.9927 - val_loss: 2.1110 - val_accuracy: 0.8161
Epoch 20/20
78/78 [==============================] - 10s 134ms/step - loss: 0.0271 - accuracy: 0.9956 - val_loss: 3.5824 - val_accuracy: 0.7919
```

Рисунок 3.16 Навчання мережі за епохами

За результатами навчання видно, що val_loss залишається однаковою. Також можна сказати, що точність стала краще – модель навчається. Найкращий результат на 19 епосі val_accuracy = 0,8161, що вже краще ніж навчання за попередніми моделями:

- val_loss = 2.2;
- val_accuracy = 0.809.

Для наступного навчаня було змінено batch_size = 16.

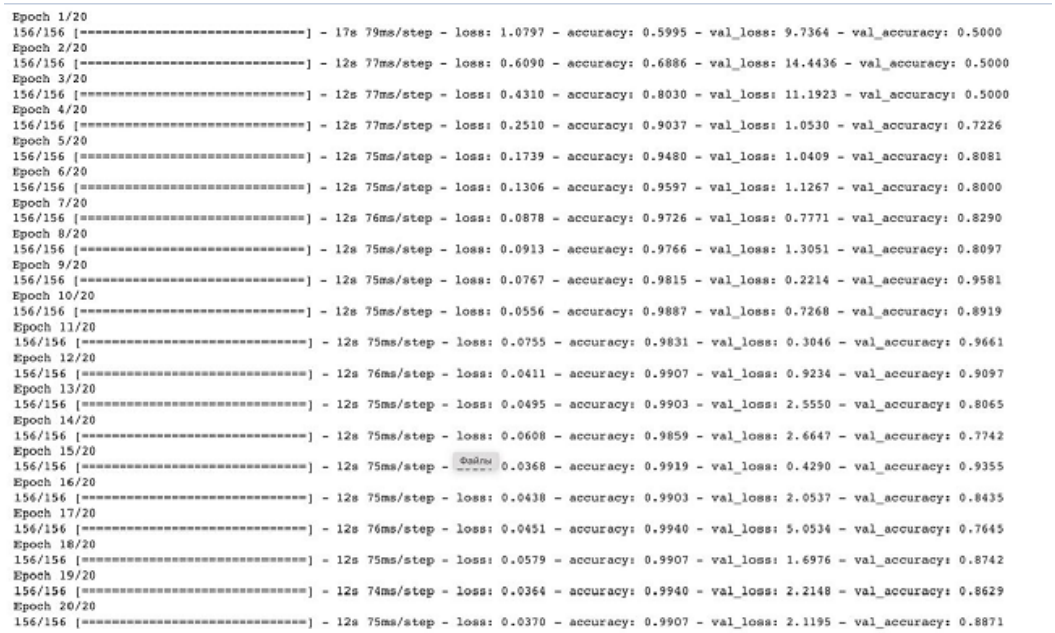


Рисунок 3.17 Навчання мережі за епохами

За результатами останнього навчання можна сказати, що val_accuracy збільшилась, а val_loss значно зменшилось. На епохах 9 та 11 точність найбільша.

9 епоха:

- val_loss = 0.22;
- val_accuracy = 0.958.

11 епоха:

- val_loss = 0.3;
- val_accuracy = 0.966.

3.4 Тестування продуктивності моделі

3.4.1 Результати навчання моделі Xception

Продуктивність навчання Xception на першому тесті 8 епоха:

- test_accuracy = 0,23;
- val_accuracy = 0,72.

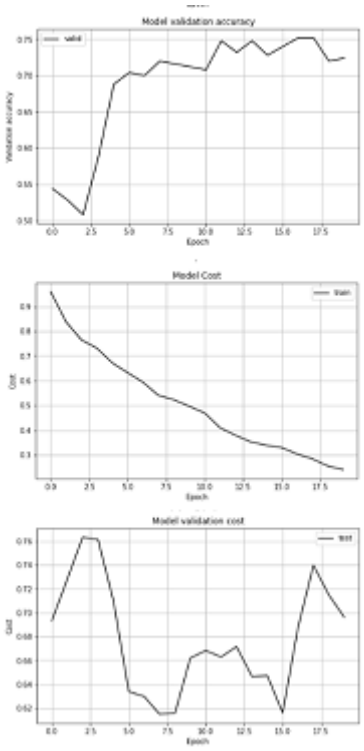


Рисунок 3.18 Графік точності та втрат Xception

Продуктивність навчання Xception на першому тесті 23 епоха:

- test_accuracy = 0,62;
- val_accuracy = 0,67

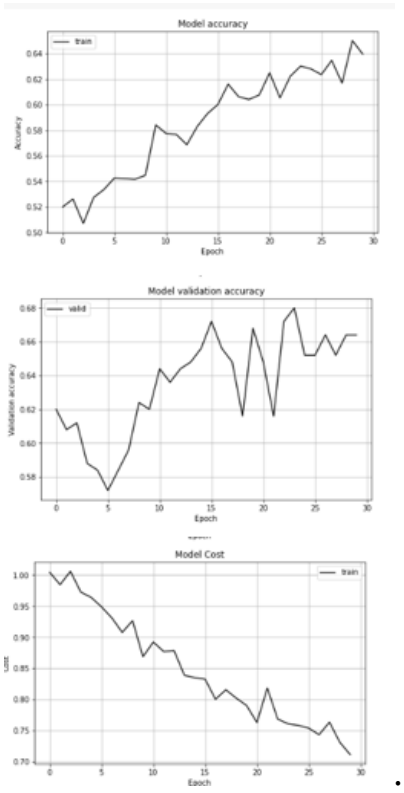


Рисунок 3.19 Графік точності та втрат Xception

3.4.2 Результати навчання моделі VGG net

Продуктивність навчання мережі VGG net на першому тесті:

- test_accuracy = 0.90
- val_accuracy = 0.65

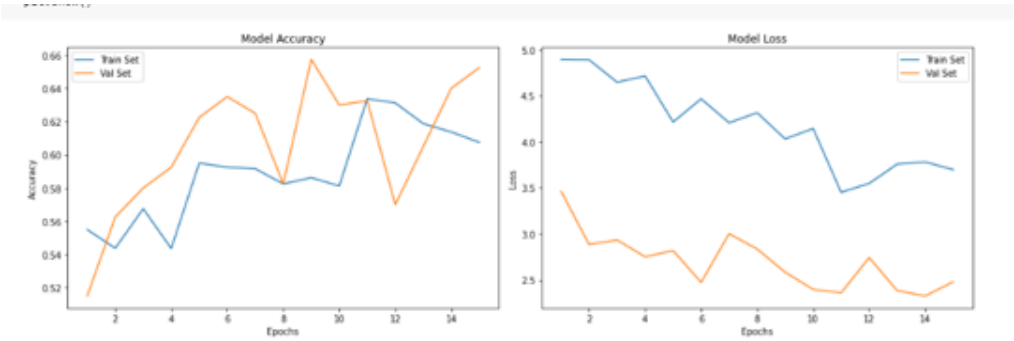


Рисунок 3.20 Графік точності та втрат VGG net

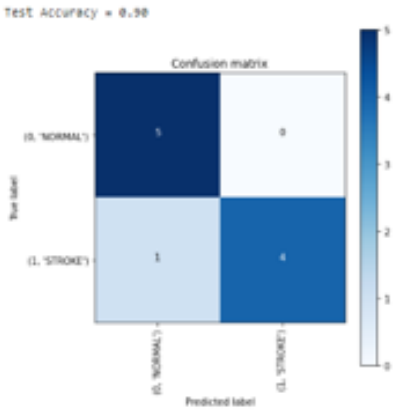


Рисунок 3.21 Точність на тестовій вибірці

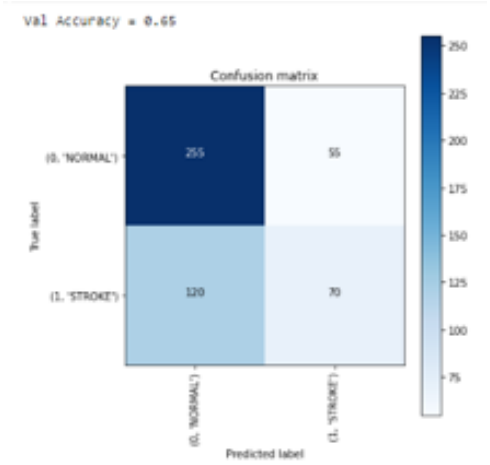


Рисунок 3.22 Точність валідаційна

Продуктивність навчання мережі VGG net на другому тесті:

- test_accuracy = 0.90
- val_accuracy = 0.65

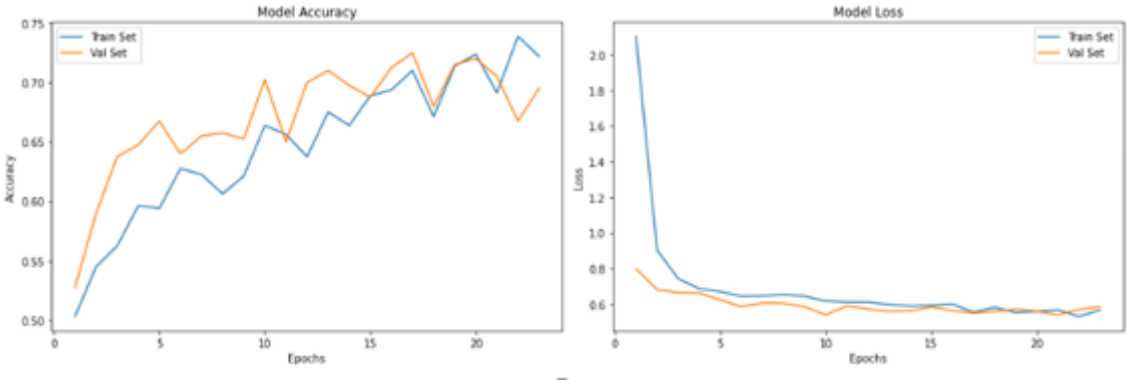


Рисунок 3.23 Графік точності та втрат VGG net

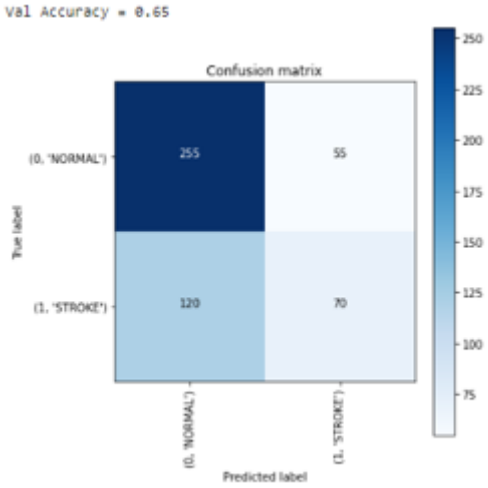


Рисунок 3.24 Точність на тестовій вибірці

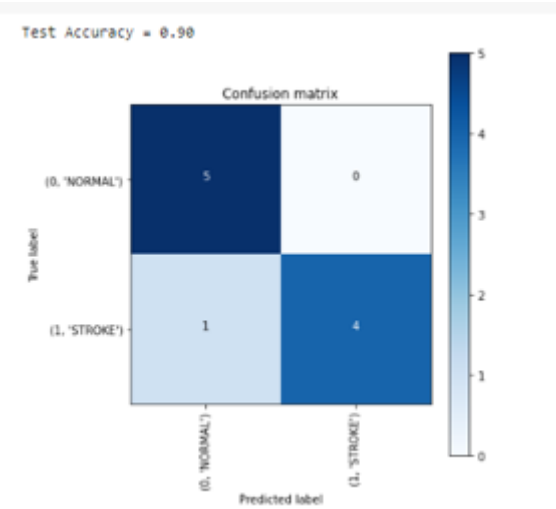


Рисунок 3.25 Точність валідаційна

3.4.3 Результати навчання кастомної моделі

Продуктивність кастомної мережі на першому тесті:

- val_loss = 2.2;
- val_accuracy = 0.809.

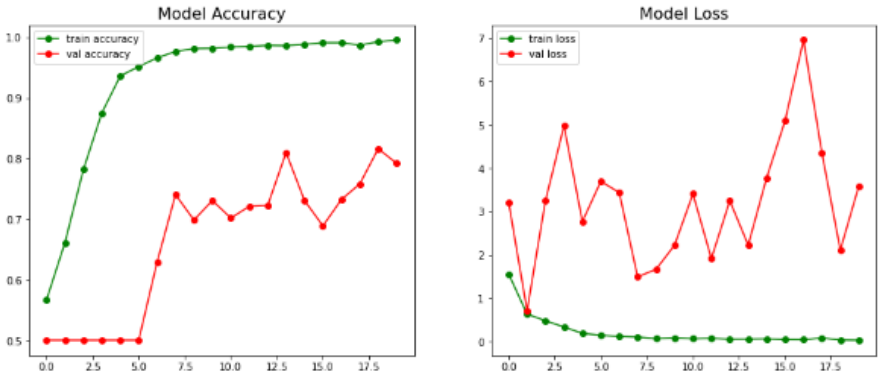


Рисунок 3.26 Графік точності та втрат кастомної моделі для першого навчання

Продуктивність кастомної мережі на першому тесті:

- val_loss = 0.3;
- val_accuracy = 0.966.

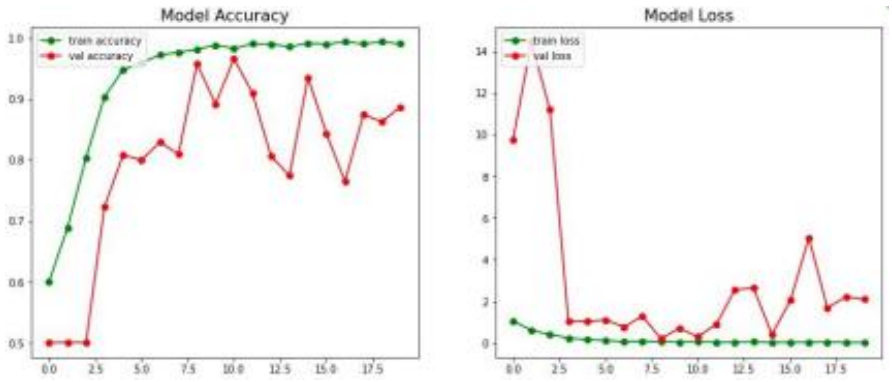


Рисунок 3.27 Графік точності та втрат кастомної моделі для другого навчання

Порівняння показників різних моделей за визначеними критеріями представлено у таблиці 3.1.

Таблиця 3.1 Порівняння моделей згорткових нейронних мереж

Модель	Batch_size	Learning_rate	Epoch	Accuracy	Loss	Test
Xception	16	0,001	30	0,83	0,61	0,25

VGG net	128	0,001	20	0,72	0,56	0,57
Custom	16	0,001	20	0.97	0,3	0,67

Отже, можна зробити висновок, що за більшістю критеріїв найкращою з даних моделей є користувацька. Не зважаючи на те, що середній час навчання набагато більший порівняно з іншими, для задачі розпізнавання захворювання точність є найважливішим критерієм оцінки роботи моделі.

3.5 Висновки до розділу

Таким чином, алгоритм нечіткої згорткових нейроних мереж дозволив покращити якість та точність розпізнавання порівняно зі стандартним алгоритмом та проведенимимпопередніми дослідженнями. Модель Xception та VGG net виявились неефективними у деяких випадках, а саме при наявності чорно-білих КТ знімків та нерівній кількості зображень на валідаційній та тестовій вибірках. У подальших дослідженнях можливе впровадження паралельних обчислень для пришвидшення роботи системи, оскільки у даній роботі алгоритм обробляє серединні розрізи кожного КТ файлу послідовно.

Висновок

Тема роботи є дуже актуальною, адже розпізнавання захворювань за допомогою нейронних мереж – це новий перспективний напрямок розвитку сфери медичної діагностики і штучного інтелекту.

У першому розділі було проаналізовано актуальність роботи, та статистику захворювань які часто зустрічаються в світі, та та в Україні. Аналіз доступних літературних джерел свідчить, що на сьогоднішній день більшість досліждень зосереджена на розробці та оптимізації алгоритмів для сегментації уражень на зображеннях для швидкої сегментації та діагностики на гострій стадії після перенесеного інсульту. Розглянуто основні методи вирішення задачі розпізнавання хвороби головного мозку на основі КТ знімків та проблеми, що виникають у разі впровадження подібних систем у реальному житті.

У другому розділі було розроблено архітектуру інтелектуальної системи, розглянуто існуючі архітектури згорткових нейронних мереж(VGG net, Xception) та обрано оптимальний варіант для реалізації поставленої задачі.

У третьому розділі було проведено аналіз процесу сегментації зображень за допомогою згорткових нейронних мереж та дослідження його ефективності на основі моделей із навчанням згорткових нейромереж з глибокими шарами. Отримані значення точності дозволяють стверджувати про правильність вибору архітектури мережі та підбору параметрів, що дає можливість використовувати її для практичних задач сегментації зображень і розпізнавання об’єктів, зокрема для пристроїв із обмеженими обчислювальними ресурсами.

Було використано мову програмування Python, відкриту програмну библиотеку для машинного навчання Tensorflow та вбудовану в неї нейроморежну бібліотеку Keras та інші програмні бібліотеки. Для розробки модуля було обрано блоктот Google Colab, який дозволяє будь-яким чином писати та виконувати відповідний код Python через браузер і особливо добре підходить для машинного навчання, аналізу даних.

Для подальших досліджень та оптимізації системи рекомендується розширити список захворювань головного мозку, які може розпізнавати система, а також вдосконалити систему забезпечення конфіденційності даних

Список використаної літератури

1. Johnson, C.O.; Nguyen, M.; Roth, G.A.; Nichols, E.; Alam, T. Global, regional, and national burden of stroke, 1990-2016: A systematic analysis for the Global Burden of Disease Study 2016. *Lancet Neurol.* **2019**, *18*, 439–458.
2. Subudhi, A.; Dash, M.; Sabut, S. Automated segmentation and classification of brain stroke using expectation-maximization and random forest classifier. *Biocybern. Biomed. Eng.* **2020**, *40*, 277–289.
3. Lee, H.J.; Lee, J.S.; Choi, J.C.; Cho, Y.J.; Kim, B.J.; Bae, H.J.; Kim, D.E.; Ryu, W.S.; Cha, J.K.; Kim, D.H.; et al. Simple estimates of symptomatic intracranial hemorrhage risk and outcome after intravenous thrombolysis using age and stroke severity. *J. Stroke* **2017**, *19*, 229–231.
4. Global, regional, and national burden of stroke, 1990–2016: a systematic analysis for the Global Burden of Disease Study 2016. GBD 2016 Stroke Collaborators // Lancet Neurol. 2019. Vol. 18. P. 439–458. Published Online March 11. 2019. doi: 10.1016/ S1474-4422(19)30034-1.
5. Marie-Héléne, M. & Cramer, S. C. Biomarkers of recovery after stroke. Curr.Opin. Neurol. 21, 654–659 (2008).
6. Ischemic Stroke Lesion Segmentation. (2018). http://www.isleschallenge.org/.
7. Blinov DS, Lobishcheva AE, Varfolomeeva AA, Kamishanskaya IG, Blinova EV. Chest x-rays analysis by neural network: contemporary achievements and causes of misinterpretation. Health Care Standardization Problems.2019; 9-10: 4-9. DOI: 10.26347/1607-2502201909-10004-009
8. Song Yuheng, Yan Hao, Image Segmentation Algorithms Overview,https://arxiv.org/ftp/arxiv/papers/1707/1707.02051.pdf
9. Abràmoff M.D., Garvin M.K., Sonka M. Retinal Imaging and Image Analysis, IEEE Trans Med Imaging. 2010. – Vol. 3. – P. 169–208.

10. Prokhorov D.A. A Convolutional Learning System for Object Classification in 3-D LIDAR Data. IEEE Transactions on Neural Networks. 2010; 21(5): 858-863.
11. LiuYu., Gadepalli Kr., Norouzi M., Dahl G.E., Kohlberger T., Boyko A., Venugopalan S., Timofeev A., Nelson P.Q., Corrado G.S., Hipp J.D., Peng L., Stumpe M.C. Yun Liu Detecting Cancer Metastases on Gigapixel Pathology Images. Eprintar Xiv: 1703.02442. – 03/2017.
12. International Journal of Computer Science & Information Technology (IJCSIT) Vol 3, No 6, Dec 2011
13. Р. Гонсалес, Р. Вудс Цифровая обработка изображений — М: Техносфера, 2005 – 1007с
14. https://www.kaggle.com/datasets/afridirahman/brain-stroke-ct-image-dataset?sort=votes
15. Yann LeCun et al. Generalization and network design strategies. Connections in Perspective. North-Holland, Amsterdam, pages 143–55, 1989.
16. B Boser Le Cun, John S Denker, D Henderson, Richard E Howard, W Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In Advances in neural information processing systems. Citeseer, 1990.
17. http:/deeplearning.net/tutorial/lenet.html

Додаток 1

Модель Xception

```
[ ] from IPython.display import clear_output
    !pip install imutils
    clear_output()
```

```
[ ] import numpy as np
    from tqdm import tqdm
    import cv2
    import os
    import shutil
    import itertools
    import imutils
    import matplotlib.pyplot as plt
    from sklearn.preprocessing import LabelBinarizer
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score, confusion_matrix

    import plotly.graph_objs as go
    from plotly.offline import init_notebook_mode, iplot
    from plotly import tools

    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
    from tensorflow.keras import layers
    from tensorflow.keras.models import Model, Sequential
    from tensorflow.keras.optimizers import RMSprop
    from tensorflow.keras.callbacks import EarlyStopping
    import tensorflow

    init_notebook_mode(connected=True)
    RANDOM_SEED = 123
```

```
[ ] !apt-get install tree
    clear_output()
    # create new folders
    !mkdir TRAIN TEST VAL TRAIN/STROKE TRAIN/NORMAL TEST/STROKE TEST/NORMAL VAL/STROKE VAL/NORMAL
    !tree -d
```

```
.
├── sample_data
├── TEST
│   ├── NORMAL
│   └── STROKE
├── TRAIN
│   ├── NORMAL
│   └── STROKE
└── VAL
    ├── NORMAL
    └── STROKE

10 directories
```

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
def load_data(dir_path, img_size=(100,100)):
    """
    Load resized images as np.arrays to workspace
    """
    X = []
    y = []
    i = 0
    labels = dict()
    for path in tqdm(sorted(os.listdir(dir_path))):
        if not path.startswith('.'):
            labels[i] = path
            for file in os.listdir(dir_path + path):
                if not file.startswith('.'):
                    img = cv2.imread(dir_path + path + '/' + file)
                    X.append(img)
                    y.append(i)
            i += 1
    X = np.array(X)
    y = np.array(y)
    print(f'{len(X)} images loaded from {dir_path} directory.')
    return X, y, labels



def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
```

```
[ ]  IMG_PATH = '/content/drive/MyDrive/brain/Brain_Data_Organised/'
     # split the data by train/val/test
     print(os.listdir(IMG_PATH))
     for CLASS in os.listdir(IMG_PATH):
         if not CLASS.startswith('.'):
             # print(os.listdir(IMG_PATH + CLASS))
             IMG_NUM = len(os.listdir(IMG_PATH + CLASS))
             print(IMG_NUM)
             for (n, FILE_NAME) in enumerate(os.listdir(IMG_PATH + CLASS)):
                 img = IMG_PATH + CLASS + '/' + FILE_NAME
                # print("n = ", n)
                 #print(img,"\n")
                 if n < 5:
                     shutil.copy(img, '/content/TEST/' + CLASS.upper() + '/' + FILE_NAME)
                 elif n < 0.8*IMG_NUM:
                     shutil.copy(img, '/content/TRAIN/'+ CLASS.upper() + '/' + FILE_NAME)
                 else:
                     shutil.copy(img, '/content/VAL/'+ CLASS.upper() + '/' + FILE_NAME)
```

```
['Stroke', 'Normal']
950
1551
```

```
def load_data(dir_path, img_size=(100,100)):
    """
    Load resized images as np.arrays to workspace
    """
    X = []
    y = []
    i = 0
    labels = dict()
    for path in tqdm(sorted(os.listdir(dir_path))):
        if not path.startswith('.'):
            labels[i] = path
            for file in os.listdir(dir_path + path):
                if not file.startswith('.'):
                    img = cv2.imread(dir_path + path + '/' + file)
                    X.append(img)
                    y.append(i)
            i += 1
    X = np.array(X)
    y = np.array(y)
    print(f'{len(X)} images loaded from {dir_path} directory.')
    return X, y, labels



def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.figure(figsize = (6,6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
```

```
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    cm = np.round(cm,2)
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1]
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

```
TRAIN_DIR = 'TRAIN/'
TEST_DIR = 'TEST/'
VAL_DIR = 'VAL/'
IMG_SIZE = (299,299)

# use predefined function to load the image data into workspace
X_train, y_train, labels = load_data(TRAIN_DIR, IMG_SIZE)
X_test, y_test, _ = load_data(TEST_DIR, IMG_SIZE)
X_val, y_val, _ = load_data(VAL_DIR, IMG_SIZE)
```

```
100%|██████████| 2/2 [00:06<00:00,  3.24s/it]
1991 images loaded from TRAIN/ directory.
100%|██████████| 2/2 [00:00<00:00, 35.07it/s]
10 images loaded from TEST/ directory.
100%|██████████| 2/2 [00:01<00:00,  1.36it/s]
500 images loaded from VAL/ directory.
```

```
[ ]  y = dict()
     y[0] = []
     y[1] = []
     for set_name in (y_train, y_val, y_test):
         y[0].append(np.sum(set_name == 0))
         y[1].append(np.sum(set_name == 1))

     trace0 = go.Bar(
         x=['Train Set', 'Validation Set', 'Test Set'],
         y=y[0],
         name='No',
         marker=dict(color='#33cc33'),
         opacity=0.7
     )
     trace1 = go.Bar(
         x=['Train Set', 'Validation Set', 'Test Set'],
         y=y[1],
         name='Yes',
         marker=dict(color='#ff3300'),
         opacity=0.7
     )
     data = [trace0, trace1]
     layout = go.Layout(
         title='Count of classes in each set',
         xaxis={'title': 'Set'},
         yaxis={'title': 'Count'}
     )
     fig = go.Figure(data, layout)
     iplot(fig)
```

```
[ ]  def plot_samples(X, y, labels_dict, n=50):
         """
         Creates a gridplot for desired number of images (n) from the specified set
         """
         for index in range(len(labels_dict)):
             imgs = X[np.argwhere(y == index)][:n]
             j = 10
             i = int(n/j)

             plt.figure(figsize=(15,6))
             c = 1
             for img in imgs:
                 plt.subplot(i,j,c)
                 plt.imshow(img[0])

                 plt.xticks([])
                 plt.yticks([])
                 c += 1
             plt.suptitle('Tumor: {}'.format(labels_dict[index]))
             plt.show()
```

```
[ ]  plot_samples(X_train, y_train, labels, 30)
```

```
[ ]  RATIO_LIST = []
     for set in (X_train, X_test, X_val):
         for img in set:
             RATIO_LIST.append(img.shape[1]/img.shape[0])

     plt.hist(RATIO_LIST)
     plt.title('Distribution of Image Ratios')
     plt.xlabel('Ratio Value')
     plt.ylabel('Count')
     plt.show()
```

```python
def crop_imgs(set_name, add_pixels_value=0):
    """
    Finds the extreme points on the image and crops the rectangular out of them
    """
    set_new = []
    for img in set_name:
        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        gray = cv2.GaussianBlur(gray, (5, 5), 0)

        # threshold the image, then perform a series of erosions +
        # dilations to remove any small regions of noise
        thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
        thresh = cv2.erode(thresh, None, iterations=2)
        thresh = cv2.dilate(thresh, None, iterations=2)

        # find contours in thresholded image, then grab the largest one
        cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)
        c = max(cnts, key=cv2.contourArea)

        # find the extreme points
        extLeft = tuple(c[c[:, :, 0].argmin()][0])
        extRight = tuple(c[c[:, :, 0].argmax()][0])
        extTop = tuple(c[c[:, :, 1].argmin()][0])
        extBot = tuple(c[c[:, :, 1].argmax()][0])

        ADD_PIXELS = add_pixels_value
        new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS, extLeft[0]-ADD_PIXELS:extRight[0]+ADD_PIXELS].copy()
        set_new.append(new_img)

    return np.array(set_new)
```

```
[ ]  img = cv2.imread('/content/drive/MyDrive/brain/Brain_Data_Organised/Stroke/58 (1).jpg')
     img = cv2.resize(
                 img,
                 dsize=IMG_SIZE,
                 interpolation=cv2.INTER_CUBIC
             )
     gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
     gray = cv2.GaussianBlur(gray, (5, 5), 0)

     # threshold the image, then perform a series of erosions +
     # dilations to remove any small regions of noise
     thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
     thresh = cv2.erode(thresh, None, iterations=2)
     thresh = cv2.dilate(thresh, None, iterations=2)

     # find contours in thresholded image, then grab the largest one
     cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
     cnts = imutils.grab_contours(cnts)
     c = max(cnts, key=cv2.contourArea)

     # find the extreme points
     extLeft = tuple(c[c[:, :, 0].argmin()][0])
     extRight = tuple(c[c[:, :, 0].argmax()][0])
     extTop = tuple(c[c[:, :, 1].argmin()][0])
     extBot = tuple(c[c[:, :, 1].argmax()][0])

     # add contour on the image
     img_cnt = cv2.drawContours(img.copy(), [c], -1, (0, 255, 255), 4)

     # add extreme points
     img_pnt = cv2.circle(img_cnt.copy(), extLeft, 8, (0, 0, 255), -1)
     img_pnt = cv2.circle(img_pnt, extRight, 8, (0, 255, 0), -1)
     img_pnt = cv2.circle(img_pnt, extTop, 8, (255, 0, 0), -1)
     img_pnt = cv2.circle(img_pnt, extBot, 8, (255, 255, 0), -1)
```

```
# crop
ADD_PIXELS = 0
new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS, extLeft[0]-ADD_PIXELS:extRight[0]+ADD_PIXELS].copy()
```

```
plt.figure(figsize=(15,6))
plt.subplot(141)
plt.imshow(img)
plt.xticks([])
plt.yticks([])
plt.title('Step 1. Get the original image')
plt.subplot(142)
plt.imshow(img_cnt)
plt.xticks([])
plt.yticks([])
plt.title('Step 2. Find the biggest contour')
plt.subplot(143)
plt.imshow(img_pnt)
plt.xticks([])
plt.yticks([])
plt.title('Step 3. Find the extreme points')
plt.subplot(144)
plt.imshow(new_img)
plt.xticks([])
plt.yticks([])
plt.title('Step 4. Crop the image')
plt.show()
```

```
# apply this for each set
X_train_crop = crop_imgs(set_name=X_train)
X_val_crop = crop_imgs(set_name=X_val)
X_test_crop = crop_imgs(set_name=X_test)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_

Creating an ndarray from ragged nested sequences
```

```
plot_samples(X_train_crop, y_train, labels, 30)
```

```
def save_new_images(x_set, y_set, folder_name):
    i = 0
    for (img, imclass) in zip(x_set, y_set):
        if imclass == 0:
            cv2.imwrite(folder_name+'NO/'+str(i)+'.jpg', img)
        else:
            cv2.imwrite(folder_name+'YES/'+str(i)+'.jpg', img)
        i += 1
```

```
# saving new images to the folder
!mkdir TRAIN_CROP TEST_CROP VAL_CROP TRAIN_CROP/YES TRAIN_CROP/NO TEST_CROP/YES TEST_CROP/NO VAL_CROP/YES VAL_CROP/NO

save_new_images(X_train_crop, y_train, folder_name='TRAIN_CROP/')
save_new_images(X_val_crop, y_val, folder_name='VAL_CROP/')
save_new_images(X_test_crop, y_test, folder_name='TEST_CROP/')
```

```
[ ]  def preprocess_imgs(set_name, img_size):

         set_new = []
         for img in set_name:
             img = cv2.resize(
                 img,
                 dsize=img_size,
                 interpolation=cv2.INTER_CUBIC
             )
             set_new.append(preprocess_input(img))
         return np.array(set_new)
```

```
[ ]  X_train_prep = preprocess_imgs(set_name=X_train_crop, img_size=IMG_SIZE)
     X_test_prep = preprocess_imgs(set_name=X_test_crop, img_size=IMG_SIZE)
     X_val_prep = preprocess_imgs(set_name=X_val_crop, img_size=IMG_SIZE)
```

```
[ ]  # set the paramters we want to change randomly
     demo_datagen = ImageDataGenerator(
         rotation_range=15,
         width_shift_range=0.05,
         height_shift_range=0.05,
         rescale=1./255,
         shear_range=0.05,
         brightness_range=[0.1, 1.5],
         horizontal_flip=True,
         vertical_flip=True
     )
```

```
[ ]  os.mkdir('preview')
     x = X_train_crop[0]
     x = x.reshape((1,) + x.shape)

     i = 0
     for batch in demo_datagen.flow(x, batch_size=1, save_to_dir='preview', save_prefix='aug_img', save_format='jpg'):
         i += 1
         if i > 20:
             break
```

```
     plt.imshow(X_train_crop[0])
     plt.xticks([])
     plt.yticks([])
     plt.title('Original Image')
     plt.show()

     plt.figure(figsize=(15,6))
     i = 1
     for img in os.listdir('preview/'):
         img = cv2.cv2.imread('preview/' + img)
         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
         plt.subplot(3,7,i)
         plt.imshow(img)
         plt.xticks([])
         plt.yticks([])
         i += 1
         if i > 3*7:
             break
     plt.suptitle('Augemented Images')
     plt.show()
```

```
[ ] TRAIN_DIR = 'TRAIN_CROP/'
    VAL_DIR = 'VAL_CROP/'

    train_datagen = ImageDataGenerator(
        rotation_range=15,
        width_shift_range=0.1,
        height_shift_range=0.1,
        shear_range=0.1,
        brightness_range=[0.5, 1.5],
        horizontal_flip=True,
        vertical_flip=True,
        preprocessing_function=preprocess_input
    )

    test_datagen = ImageDataGenerator(
        preprocessing_function=preprocess_input
    )


    train_generator = train_datagen.flow_from_directory(
        TRAIN_DIR,
        color_mode='rgb',
        target_size=IMG_SIZE,
        batch_size=128,
        class_mode='categorical',
        seed=RANDOM_SEED
    )
```

```
    validation_generator = test_datagen.flow_from_directory(
        VAL_DIR,
        color_mode='rgb',
        target_size=IMG_SIZE,
        batch_size=128,
        class_mode='categorical',
        seed=RANDOM_SEED
    )
```

```
    Found 1991 images belonging to 2 classes.
    Found 500 images belonging to 2 classes.
```

```
[ ] # load base model
    base_model = tensorflow.keras.applications.Xception(
        include_top=False,
        input_shape=IMG_SIZE + (3,)

    )
    base_model.trainable = False
```

```
[ ] NUM_CLASSES = 2

    model = tensorflow.keras.Sequential([
        base_model,
        tensorflow.keras.layers.GlobalAveragePooling2D(),
        tensorflow.keras.layers.Dense(100, activation="relu"),
       tensorflow.keras.layers.BatchNormalization(trainable = True,axis=1),

        tensorflow.keras.layers.Dropout(0.5),

        tensorflow.keras.layers.Dense(50, activation="relu"),
        tensorflow.keras.layers.BatchNormalization(trainable = True,axis=1),

        tensorflow.keras.layers.Dense(NUM_CLASSES,activation='softmax')
    ])

    model.summary()
```

```
[ ]  model.compile(optimizer=tensorflow.keras.optimizers.Nadam(learning_rate=0.001),
                   loss=tensorflow.keras.losses.CategoricalCrossentropy(),
                   metrics=[tensorflow.keras.metrics.AUC(name='prc', curve='PR')])
```

```
[ ]  EPOCHS = 30
     es = EarlyStopping(
         monitor='val_prc',
         mode='max',
         patience=6
     )

     history = model.fit_generator(
         train_generator,
         epochs=EPOCHS,
         validation_data=validation_generator

     )
```

```
[ ]  # plot model performance
     acc = history.history['prc']
     val_prc = history.history['prc']
     loss = history.history['loss']
     val_loss = history.history['val_loss']
     epochs_range = range(1, len(history.epoch) + 1)

     plt.figure(figsize=(15,5))

     plt.subplot(1, 2, 1)
     plt.plot(epochs_range, acc, label='Train Set')
     plt.plot(epochs_range, val_prc, label='Val Set')
     plt.legend(loc="best")
     plt.xlabel('Epochs')
     plt.ylabel('Accuracy')
     plt.title('Model Accuracy')

     plt.subplot(1, 2, 2)
     plt.plot(epochs_range, loss, label='Train Set')
     plt.plot(epochs_range, val_loss, label='Val Set')
     plt.legend(loc="best")
     plt.xlabel('Epochs')
     plt.ylabel('Loss')
     plt.title('Model Loss')

     plt.tight_layout()
     plt.show()
```

```
[ ]  # validate on val set
     predictions = model.predict(X_val_prep)
     predictions = [1 if x>0.5 else 0 for x in predictions]

     accuracy = accuracy_score(y_val, predictions)
     print('Val Accuracy = %.2f' % accuracy)

     confusion_mtx = confusion_matrix(y_val, predictions)
     cm = plot_confusion_matrix(confusion_mtx, classes = list(labels.items()), normalize=False)
```

```
[ ]  # validate on test set
     predictions = model.predict(X_test_prep)
     predictions = [1 if x>0.5 else 0 for x in predictions]

     accuracy = accuracy_score(y_test, predictions)
     print('Test Accuracy = %.2f' % accuracy)

     confusion_mtx = confusion_matrix(y_test, predictions)
     cm = plot_confusion_matrix(confusion_mtx, classes = list(labels.items()), normalize=False)
```

```
ind_list = np.argwhere((y_test == predictions) == False)[:, -1]
if ind_list.size == 0:
    print('There are no missclassified images.')
else:
    for i in ind_list:
        plt.figure()
        plt.imshow(X_test_crop[i])
        plt.xticks([])
        plt.yticks([])
        plt.title(f'Actual class: {y_val[i]}\nPredicted class: {predictions[i]}')
        plt.show()
```

```
[ ]  # clean up the space
     !rm -rf TRAIN TEST VAL TRAIN_CROP TEST_CROP VAL_CROP
     # save the model
     model.save('/content/drive/MyDrive/brain/2022-04-18_XCEPTION_model.h5')
```

Додаток 2

VGG net

```
[ ]  from IPython.display import clear_output
     !pip install imutils
     clear_output()
```

```
[ ]  import numpy as np
     from tqdm import tqdm
     import cv2
     import os
     import shutil
     import itertools
     import imutils
     import matplotlib.pyplot as plt
     from sklearn.preprocessing import LabelBinarizer
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import accuracy_score, confusion_matrix

     import plotly.graph_objs as go
     from plotly.offline import init_notebook_mode, iplot
     from plotly import tools

     from tensorflow.keras.preprocessing.image import ImageDataGenerator
     from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
     from tensorflow.keras import layers
     from tensorflow.keras.models import Model, Sequential
     from tensorflow.keras.optimizers import RMSprop
     from tensorflow.keras.callbacks import EarlyStopping

     init_notebook_mode(connected=True)
     RANDOM_SEED = 123
```

```
[ ]  !apt-get install tree
     clear_output()
     # create new folders
     !mkdir TRAIN TEST VAL TRAIN/STROKE TRAIN/NORMAL TEST/STROKE TEST/NORMAL VAL/STROKE VAL/NORMAL
     !tree -d
```

```
[ ]  IMG_PATH = '/content/drive/MyDrive/brain/Brain_Data_Organised/'
     # split the data by train/val/test
     print(os.listdir(IMG_PATH))
     for CLASS in os.listdir(IMG_PATH):
         if not CLASS.startswith('.'):
             # print(os.listdir(IMG_PATH + CLASS))
             IMG_NUM = len(os.listdir(IMG_PATH + CLASS))
             print(IMG_NUM)
             for (n, FILE_NAME) in enumerate(os.listdir(IMG_PATH + CLASS)):
                 img = IMG_PATH + CLASS + '/' + FILE_NAME
                 print("n = ", n)
                 print(img,"\n")
                 if n < 5:
                     shutil.copy(img, '/content/TEST/' + CLASS.upper() + '/' + FILE_NAME)
                 elif n < 0.8*IMG_NUM:
                     shutil.copy(img, '/content/TRAIN/'+ CLASS.upper() + '/' + FILE_NAME)
                 else:
                     shutil.copy(img, '/content/VAL/'+ CLASS.upper() + '/' + FILE_NAME)
```

```
def load_data(dir_path, img_size=(100,100)):
    """
    Load resized images as np.arrays to workspace
    """
    X = []
    y = []
    i = 0
    labels = dict()
    for path in tqdm(sorted(os.listdir(dir_path))):
        if not path.startswith('.'):
            labels[i] = path
            for file in os.listdir(dir_path + path):
                if not file.startswith('.'):
                    img = cv2.imread(dir_path + path + '/' + file)
                    X.append(img)
                    y.append(i)
            i += 1
    X = np.array(X)
    y = np.array(y)
    print(f'{len(X)} images loaded from {dir_path} directory.')
    return X, y, labels



def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.figure(figsize = (6,6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    cm = np.round(cm,2)
```

```
TRAIN_DIR = 'TRAIN/'
TEST_DIR = 'TEST/'
VAL_DIR = 'VAL/'
IMG_SIZE = (224,224)

# use predefined function to load the image data into workspace
X_train, y_train, labels = load_data(TRAIN_DIR, IMG_SIZE)
X_test, y_test, _ = load_data(TEST_DIR, IMG_SIZE)
X_val, y_val, _ = load_data(VAL_DIR, IMG_SIZE)
```

```
[ ]  y = dict()
     y[0] = []
     y[1] = []
     for set_name in (y_train, y_val, y_test):
         y[0].append(np.sum(set_name == 0))
         y[1].append(np.sum(set_name == 1))

     trace0 = go.Bar(
         x=['Train Set', 'Validation Set', 'Test Set'],
         y=y[0],
         name='No',
         marker=dict(color='#33cc33'),
         opacity=0.7
     )
     trace1 = go.Bar(
         x=['Train Set', 'Validation Set', 'Test Set'],
         y=y[1],
         name='Yes',
         marker=dict(color='#ff3300'),
         opacity=0.7
     )
     data = [trace0, trace1]
     layout = go.Layout(
         title='Count of classes in each set',
         xaxis={'title': 'Set'},
         yaxis={'title': 'Count'}
     )
     fig = go.Figure(data, layout)
     iplot(fig)
```

```
[ ]  def plot_samples(X, y, labels_dict, n=50):
         """
         Creates a gridplot for desired number of images (n) from the specified set
         """
         for index in range(len(labels_dict)):
             imgs = X[np.argwhere(y == index)][:n]
             j = 10
             i = int(n/j)

             plt.figure(figsize=(15,6))
             c = 1
             for img in imgs:
                 plt.subplot(i,j,c)
                 plt.imshow(img[0])

                 plt.xticks([])
                 plt.yticks([])
                 c += 1
             plt.suptitle('Tumor: {}'.format(labels_dict[index]))
             plt.show()
```

```
[ ]  plot_samples(X_train, y_train, labels, 30)
```

```
[ ]  RATIO_LIST = []
     for set in (X_train, X_test, X_val):
         for img in set:
             RATIO_LIST.append(img.shape[1]/img.shape[0])

     plt.hist(RATIO_LIST)
     plt.title('Distribution of Image Ratios')
     plt.xlabel('Ratio Value')
     plt.ylabel('Count')
     plt.show()
```

```
[ ]  def crop_imgs(set_name, add_pixels_value=0):
         """
         Finds the extreme points on the image and crops the rectangular out of them
         """
         set_new = []
         for img in set_name:
             gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
             gray = cv2.GaussianBlur(gray, (5, 5), 0)

             # threshold the image, then perform a series of erosions +
             # dilations to remove any small regions of noise
             thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
             thresh = cv2.erode(thresh, None, iterations=2)
             thresh = cv2.dilate(thresh, None, iterations=2)

             # find contours in thresholded image, then grab the largest one
             cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
             cnts = imutils.grab_contours(cnts)
             c = max(cnts, key=cv2.contourArea)

             # find the extreme points
             extLeft = tuple(c[c[:, :, 0].argmin()][0])
             extRight = tuple(c[c[:, :, 0].argmax()][0])
             extTop = tuple(c[c[:, :, 1].argmin()][0])
             extBot = tuple(c[c[:, :, 1].argmax()][0])

             ADD_PIXELS = add_pixels_value
             new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS, extLeft[0]-ADD_PIXELS:extRight[0]+ADD_PIXELS].copy()
             set_new.append(new_img)

         return np.array(set_new)
```

```
[ ]  img = cv2.imread('/content/drive/MyDrive/brain/Brain_Data_Organised/Stroke/58 (1).jpg')
     img = cv2.resize(
                img,
                dsize=IMG_SIZE,
                interpolation=cv2.INTER_CUBIC
            )
     gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
     gray = cv2.GaussianBlur(gray, (5, 5), 0)

     # threshold the image, then perform a series of erosions +
     # dilations to remove any small regions of noise
     thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
     thresh = cv2.erode(thresh, None, iterations=2)
     thresh = cv2.dilate(thresh, None, iterations=2)

     # find contours in thresholded image, then grab the largest one
     cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
     cnts = imutils.grab_contours(cnts)
     c = max(cnts, key=cv2.contourArea)

     # find the extreme points
     extLeft = tuple(c[c[:, :, 0].argmin()][0])
     extRight = tuple(c[c[:, :, 0].argmax()][0])
     extTop = tuple(c[c[:, :, 1].argmin()][0])
     extBot = tuple(c[c[:, :, 1].argmax()][0])

     # add contour on the image
     img_cnt = cv2.drawContours(img.copy(), [c], -1, (0, 255, 255), 4)

     # add extreme points
     img_pnt = cv2.circle(img_cnt.copy(), extLeft, 8, (0, 0, 255), -1)
     img_pnt = cv2.circle(img_pnt, extRight, 8, (0, 255, 0), -1)
     img_pnt = cv2.circle(img_pnt, extTop, 8, (255, 0, 0), -1)
     img_pnt = cv2.circle(img_pnt, extBot, 8, (255, 255, 0), -1)

     # crop
     ADD_PIXELS = 0
     new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS, extLeft[0]-ADD_PIXELS:extRight[0]+ADD_PIXELS].copy()
```

```
     plt.figure(figsize=(15,6))
     plt.subplot(141)
     plt.imshow(img)
     plt.xticks([])
     plt.yticks([])
     plt.title('Step 1. Get the original image')
     plt.subplot(142)
     plt.imshow(img_cnt)
     plt.xticks([])
     plt.yticks([])
     plt.title('Step 2. Find the biggest contour')
     plt.subplot(143)
     plt.imshow(img_pnt)
     plt.xticks([])
     plt.yticks([])
     plt.title('Step 3. Find the extreme points')
     plt.subplot(144)
     plt.imshow(new_img)
     plt.xticks([])
     plt.yticks([])
     plt.title('Step 4. Crop the image')
     plt.show()
```

```
[ ]  # apply this for each set
     X_train_crop = crop_imgs(set_name=X_train)
     X_val_crop = crop_imgs(set_name=X_val)
     X_test_crop = crop_imgs(set_name=X_test)
```

```
[ ]  def save_new_images(x_set, y_set, folder_name):
         i = 0
         for (img, imclass) in zip(x_set, y_set):
             if imclass == 0:
                 cv2.imwrite(folder_name+'NO/'+str(i)+'.jpg', img)
             else:
                 cv2.imwrite(folder_name+'YES/'+str(i)+'.jpg', img)
             i += 1
```

```
[ ]  # saving new images to the folder
     !mkdir TRAIN_CROP TEST_CROP VAL_CROP TRAIN_CROP/YES TRAIN_CROP/NO TEST_CROP/YES TEST_CROP/NO VAL_CROP/YES VAL_CROP/NO

     save_new_images(X_train_crop, y_train, folder_name='TRAIN_CROP/')
     save_new_images(X_val_crop, y_val, folder_name='VAL_CROP/')
     save_new_images(X_test_crop, y_test, folder_name='TEST_CROP/')
```

The next step would be resizing images to `(224,224)` and applying preprocessing needed for VGG-16 model input.

```
[ ]  def preprocess_imgs(set_name, img_size):
         """
         Resize and apply VGG-15 preprocessing
         """
         set_new = []
         for img in set_name:
             img = cv2.resize(
                 img,
                 dsize=img_size,
                 interpolation=cv2.INTER_CUBIC
             )
             set_new.append(preprocess_input(img))
         return np.array(set_new)
```

```
[ ]  X_train_prep = preprocess_imgs(set_name=X_train_crop, img_size=IMG_SIZE)
     X_test_prep = preprocess_imgs(set_name=X_test_crop, img_size=IMG_SIZE)
     X_val_prep = preprocess_imgs(set_name=X_val_crop, img_size=IMG_SIZE)
```

```
[ ] # set the paramters we want to change randomly
    demo_datagen = ImageDataGenerator(
        rotation_range=15,
        width_shift_range=0.05,
        height_shift_range=0.05,
        rescale=1./255,
        shear_range=0.05,
        brightness_range=[0.1, 1.5],
        horizontal_flip=True,
        vertical_flip=True
    )
```

```
[ ] os.mkdir('preview')
    x = X_train_crop[0]
    x = x.reshape((1,) + x.shape)

    i = 0
    for batch in demo_datagen.flow(x, batch_size=1, save_to_dir='preview', save_prefix='aug_img', save_format='jpg'):
        i += 1
        if i > 20:
            break
```

```
[ ] plt.imshow(X_train_crop[0])
    plt.xticks([])
    plt.yticks([])
    plt.title('Original Image')
    plt.show()

    plt.figure(figsize=(15,6))
    i = 1
    for img in os.listdir('preview/'):
        img = cv2.cv2.imread('preview/' + img)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        plt.subplot(3,7,i)
        plt.imshow(img)
        plt.xticks([])
        plt.yticks([])
        i += 1
        if i > 3*7:
            break
```

```
[ ] TRAIN_DIR = 'TRAIN_CROP/'
    VAL_DIR = 'VAL_CROP/'

    train_datagen = ImageDataGenerator(
        rotation_range=15,
        width_shift_range=0.1,
        height_shift_range=0.1,
        shear_range=0.1,
        brightness_range=[0.5, 1.5],
        horizontal_flip=True,
        vertical_flip=True,
        preprocessing_function=preprocess_input
    )

    test_datagen = ImageDataGenerator(
        preprocessing_function=preprocess_input
    )


    train_generator = train_datagen.flow_from_directory(
        TRAIN_DIR,
        color_mode='rgb',
        target_size=IMG_SIZE,
        batch_size=32,
        class_mode='binary',
        seed=RANDOM_SEED
    )


    validation_generator = test_datagen.flow_from_directory(
        VAL_DIR,
        color_mode='rgb',
        target_size=IMG_SIZE,
        batch_size=16,
        class_mode='binary',
        seed=RANDOM_SEED
    )
```

```
[ ] # load base model
    vgg16_weight_path = '/content/drive/MyDrive/brain/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5'
    base_model = VGG16(
        weights=vgg16_weight_path,
        include_top=False,
        input_shape=IMG_SIZE + (3,)
    )
```

```
[ ] NUM_CLASSES = 1

    model = Sequential()
    model.add(base_model)
    model.add(layers.Flatten())
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(NUM_CLASSES, activation='sigmoid'))

    model.layers[0].trainable = False

    model.compile(
        loss='binary_crossentropy',
        optimizer=RMSprop(lr=1e-4),
        metrics=['accuracy']
    )

    model.summary()
```

```
[ ]  EPOCHS = 30
     es = EarlyStopping(
         monitor='val_accuracy',
         mode='max',
         patience=6
     )

     history = model.fit_generator(
         train_generator,
         steps_per_epoch=50,
         epochs=EPOCHS,
         validation_data=validation_generator,
         validation_steps=25,
         callbacks=[es]
     )
```

```
[ ]  # clean up the space
     !rm -rf TRAIN TEST VAL TRAIN_CROP TEST_CROP VAL_CROP
     # save the model
     model.save('2022-04-03_VGG_model.h5')
```

Додаток 3

Користувацька модель

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import cv2
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import json
import os
import random
import datetime
from tqdm import tqdm, tqdm_notebook
import tensorflow
from PIL import Image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense
from tensorflow.keras.layers import Input, Dense, Activation, BatchNormalization, Flatten, Conv2D
from tensorflow.keras.layers import AveragePooling2D, MaxPooling2D, Dropout, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.applications import VGG16
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
[ ] n_classes = 2 # count of classes
    label_shape = (n_classes,) # input shape of label
    batch_size = 32 # size of batch (2^n)
    epochs = 20 # count of epochs
    validation_split = 0.2 # 20% is val data
    seed = 51 # random seed

    image_size = (150, 150) # size of image, read from directory
    input_shape = (image_size[0], image_size[1], 3) # input shape for NN
```

```
[ ]  path_to_normal = "/content/drive/MyDrive/brain/Brain_Data_Organised/Normal"
     path_to_stroke = "/content/drive/MyDrive/brain/Brain_Data_Organised/Stroke"

     count_of_files_normal = len(os.listdir(path_to_normal))
     count_of_files_stroke = len(os.listdir(path_to_stroke))

     print("Count of files (class 'Normal') = ", count_of_files_normal)
     print("Count of files (class 'Stroke') = ", count_of_files_stroke)


     while (count_of_files_stroke != count_of_files_normal):
         random_img_path = path_to_stroke + "/" + random.choice(os.listdir(path_to_stroke))
         random_img = Image.open(random_img_path)
         # display(random_img)
         random_angle = random.randint(0,20) # random angle in range (0,180)
         augm_img = random_img.rotate(random_angle, Image.NEAREST, expand = 1)
         # display(augm_img)

         my_date = str(datetime.datetime.now().date())
         my_time = str(datetime.datetime.now().time())

         my_time = my_time.replace(":", "-")  # замінюємо непотрібні символи на бажані
         my_time = my_time.replace(".", "-")  # замінюємо непотрібні символи на бажані
         new_name = my_date + "_" + my_time  # створюємо бажану назву зображення

         augm_img_path = path_to_stroke + "/" + new_name + ".jpg"
         augm_img = augm_img.save(augm_img_path)

         count_of_files_stroke = len(os.listdir(path_to_stroke))

     print()
     print("Count of files (class 'Stroke') after augmentation = ", count_of_files_stroke)
```

```
[ ]  path_to_data = "/content/drive/MyDrive/brain/Brain_Data_Organised"

     image_generator = ImageDataGenerator(validation_split=validation_split,
                                          rescale=1./255
                                         )

     train_dataset = image_generator.flow_from_directory(directory=path_to_data,
                                                         subset="training",
                                                         seed = seed,
                                                         target_size=image_size,
                                                         batch_size=batch_size,
                                                         shuffle=True,
                                                         color_mode = "grayscale",
                                                         class_mode="categorical"
                                                         )

     test_dataset = image_generator.flow_from_directory(directory=path_to_data,
                                                        subset="validation",
                                                        seed = seed,
                                                        target_size=image_size,
                                                        batch_size=batch_size,
                                                        shuffle=True,
                                                        color_mode = "grayscale",
                                                        class_mode="categorical"
                                                        )
```

```
[ ]  images, labels = next(test_dataset)

     plt.figure(figsize=(5, 5))
     plt.imshow(np.squeeze(images[0], axis=2))
     print(labels[0])
     print(images[0].shape)
```

```
[ ]  model = tensorflow.keras.Sequential()

     model.add(tensorflow.keras.layers.Conv2D(32 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu' , input_shape = (150,150,1)))
     model.add(tensorflow.keras.layers.BatchNormalization())
     model.add(tensorflow.keras.layers.MaxPool2D((2,2) , strides = 2 , padding = 'same'))
     model.add(tensorflow.keras.layers.Conv2D(64 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
     model.add(tensorflow.keras.layers.Dropout(0.1))
     model.add(tensorflow.keras.layers.BatchNormalization())
     model.add(tensorflow.keras.layers.MaxPool2D((2,2) , strides = 2 , padding = 'same'))
     model.add(tensorflow.keras.layers.Conv2D(64 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
     model.add(tensorflow.keras.layers.BatchNormalization())
     model.add(tensorflow.keras.layers.MaxPool2D((2,2) , strides = 2 , padding = 'same'))
     model.add(tensorflow.keras.layers.Conv2D(128 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
     model.add(tensorflow.keras.layers.Dropout(0.2))
     model.add(tensorflow.keras.layers.BatchNormalization())
     model.add(tensorflow.keras.layers.MaxPool2D((2,2) , strides = 2 , padding = 'same'))
     model.add(tensorflow.keras.layers.Conv2D(256 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
     model.add(tensorflow.keras.layers.Dropout(0.2))
     model.add(tensorflow.keras.layers.BatchNormalization())
     model.add(tensorflow.keras.layers.MaxPool2D((2,2) , strides = 2 , padding = 'same'))
     model.add(tensorflow.keras.layers.Flatten())
     model.add(tensorflow.keras.layers.Dense(units = 128 , activation = 'relu'))
     model.add(tensorflow.keras.layers.Dropout(0.2))
     model.add(tensorflow.keras.layers.Dense(units = n_classes , activation = 'softmax'))

     model.compile(optimizer = "rmsprop" , loss = 'categorical_crossentropy' , metrics = ['accuracy'])

     model.summary()
```

```
learning_rate_reduction = tensorflow.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy',
                                                                       patience=5,
                                                                       verbose=1,
                                                                       factor=0.1,
                                                                       min_lr=1e-7)
```

```
[ ]  now = datetime.datetime.now()
     name = "/content/drive/MyDrive/brain/StrokeCustom" + now.strftime("%Y-%m-%d--%H-%M-%S") + "_epoch_{epoch}_acc_{accuracy:.4f}_val_acc_{val_accuracy:.4f}.h5"
     callbacks = [
         tensorflow.keras.callbacks.ModelCheckpoint(filepath=name),
         tensorflow.keras.callbacks.TensorBoard(log_dir='./logs'),
     ]
```

```
[ ]  H = model.fit(train_dataset,
                   callbacks=callbacks,
                   validation_data=test_dataset,
                   epochs=epochs,
                   verbose=1)
```

```
[ ]  fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15,6))

     axes[0].plot(H.history['accuracy'], label='train accuracy', color='g', marker='o', axes=axes[0])
     axes[0].plot(H.history['val_accuracy'], label='val accuracy', color='r', marker='o',  axes=axes[0])
     axes[0].set_title("Model Accuracy", fontsize=16)
     axes[0].legend(loc='upper left')

     axes[1].plot(H.history['loss'], label='train loss', color='g', marker='o',  axes=axes[1])
     axes[1].plot(H.history['val_loss'], label='val loss', color='r', marker='o',  axes=axes[1])
     axes[1].set_title("Model Loss", fontsize=16)
     axes[1].legend(loc='upper left')

     plt.show()
```

```
[ ]  loaded_model = tensorflow.keras.models.load_model('/content/drive/MyDrive/brain/StrokeCustom)

     # Check its architecture
     loaded_model.summary()
```

```
  File "<ipython-input-23-bf56ac6a2df7>", line 1
    loaded_model = tensorflow.keras.models.load_model('/content/drive/MyDrive/brain/StrokeCustom)
                                                                                                 ^
SyntaxError: EOL while scanning string literal
```

SEARCH STACK OVERFLOW

```
[ ]  eval_ = loaded_model.evaluate(test_dataset)
     print("Model loss:", eval_[0])
     print("Model accuracy:", eval_[1])
```